Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили

Г. В. Горбань,
І. О. Кандиба

ОПЕРАЦІЙНА СИСТЕМА LINUX

Навчальний посібник

ЧНУ
ВИДАВНИЦТВО
Миколаїв – 2019

УДК 004.451.9(075.8)
Г 67

Рекомендовано до друку вченою радою Чорноморського національного університету імені Петра Могили (протокол № 7 від 28 березня 2019 р.).

Рецензенти:

Філатов В. О., доктор технічних наук, професор, завідувач кафедри штучного інтелекту Харківського національного університету радіоелектроніки.

Яковина В. С., доктор технічних наук, професор, завідувач кафедри Програмного забезпечення Інституту комп'ютерних наук та інформаційних технологій Національного університету «Львівська політехніка».

Устенко С. А., доктор технічних наук, доцент, завідувач кафедри комп'ютерної інженерії Миколаївського національного університету ім. В. О. Сухомлинського.

Г 67 **Горбань Г. В.** Операційна система Linux : навчальний посібник / Г. В. Горбань, І. О. Кандиба. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2019. – 276 с.

ISBN 978-966-336-412-4

У посібнику розглянуто основи роботи у командному рядку операційної системи Linux. Особлива увага приділяється командам для роботи з файловою системою, обробки текстових даних, встановлення прав доступу на файли та каталоги. Окремо розглядаються засоби програмування в ОС Linux: мова Shell для написання сценаріїв командного інтерпретатора bash та компілятори gcc та g++ для написання програми на мовах програмування C та C++ відповідно. Посібник призначено для студентів спеціальностей 121 – «Інженерія програмного забезпечення» та 122 – «Комп'ютерні науки», а також може бути корисним для студентів інших спеціальностей галузі знань 12 – «Інформаційні технології».

УДК 004.451.9(075.8)

ISBN 978-966-336-412-4

ЗМІСТ

Розділ 1.
ВВЕДЕННЯ ДО ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX. ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ РІЗНИХ ДИСТРИБУТИВІВ

Операційна система є комплексом програм, що забезпечують управління апаратними засобами комп'ютера, та організовують роботу з файлами (у тому числі запуск і управління виконанням програм), а також реалізують взаємодію з користувачем, тобто інтерпретацію команд, що ним вводяться, і виведенням результатів обробки цих команд.

У сучасному світі існує потреба використовувати операційні системи не тільки для персональних комп'ютерів та серверів, а й для більш специфічної техніки: мобільні пристрої, побутова техніка та навіть безпілотні автомобілі.

На сьогодні найбільш відомими операційними системами для комп'ютерів є сімейства операційних систем Microsoft Windows, UNIX-подібні системи і безпосередньо Unix. Перші ведуть свій родовід від операційної системи MS-DOS, якої оснащувалися перші персональні комп'ютери фірми IBM. Операційна система UNIX була розроблена групою співробітників Bell Labs під керівництвом Денніса Річі, Кена Томпсона і Брайана Керніган (Dennis Ritchie, Ken Thompson, Brian Kernighan) у 1969 р. Але нині, коли говорять про операційну систему UNIX, найчастіше мають на увазі не конкретну ОС, а сімейство UNIX-подібних операційних систем. UNIX-подібні ОС базуються на тих самих ідеях, що й UNIX, мають схожий функціонал, але використовують інше ядро. Саме ж слово UNIX (великими літерами) стало зареєстрованим товарним знаком AT & T.

1.1. Ядро Linux та поняття дистрибутиву

1.1.1. Історія ОС UNIX

Операційні системи, що сьогодні є популярними так чи інакше пов'язані з UNIX. Роботу над UNIX розпочав у 1969 р. працівник компанії AT & T Bell Labs Кен Томпсон, що мав намір написати усічений варіант операційної системи MULTICS, робота над якою деякий час тому велася у цій компанії. У результаті була створена операційна система, написана повністю на асемблері, яка згодом отримала жартівливу назву UNICS (UNiplexed Information and Computing Service – примітивна інформаційна та обчислювальна служба). Потім для зручності система була перейменована

у UNIX, тому, що кінець слова читається однаково в обох випадках. Пізніше до проекту приєднався Денніс Рітчі, а потім і весь його відділ. У процесі свого розвитку UNIX був переписаний для новішого типу комп'ютерів і перенесений з мови асемблер на мову високого рівня С, що була створена Деннісом Рітчі для спрощення перенесення UNIX на інший тип комп'ютера.

UNIX швидко прийняли всюди – від університетів до великих компаній. Разом з операційною системою поширювався вихідний код, і користувачі могли змінювати або додавати нові можливості, тобто зробити UNIX максимально придатною для себе. На підставі такого буму створювалося безліч підверсій UNIX, які відрізнялися від оригіналу. Однією з найбільш вдалих стала розробка університету Берклі – Berkeley UNIX, перша версія якої називалася 1BSD (First Berkeley Software Distribution – перше розповсюджене програмне забезпечення Берклі). Програмісти істотно доповнили систему новими можливостями і програмами, зробивши великий внесок в розвиток цієї операційної системи. Після цього багато розробників UNIX стали засновувати свої версії не на продукті компанії AT & T, а на версії Berkeley UNIX, і Berkeley UNIX став конкурентом оригіналу.

Протягом тривалого часу ОС UNIX розроблялася кожним програмістом для своїх потреб, тому програми, написані для однієї версії UNIX, могли не працювати на інших, і про комерційний успіх цієї системи говорити не доводилося. З цієї причини було задекларовано документ – System V Interface Definition (SVID), який містив основні стандарти, яких мали дотримуватися розробники. Це підвищило сумісність версій UNIX, але не вплинуло на версії BSD. Найбільш відомими версіями UNIX стали BSD, MINIX (які розробляються відомим голландським професором Ендрю Таненбаумом), SCO UNIX, System V (оригінальна версія компанії AT & T), Solaris (розробка корпорації Sun), XENIX (версія UNIX корпорації Microsoft).

Варто також зазначити, що вплив на світ IT операційної системи UNIX не обмежився створенням кількох версій цієї ОС. Під впливом UNIX було створено операційну систему GNU (GNU's Not UNIX) та ядро Linux, що реалізовували принципи UNIX, але не використовували його вихідний код.

1.1.2. Ядро Linux та поняття дистрибутиву

Перше, про що треба пам'ятати почавши роботу з Linux: Linux – це не операційна система, а ядро операційної системи. Встановити Linux категорично неможливо, але можна встановити дистрибутив, що базується на ядрі Linux.

Ядро операційної системи – це основна частина ОС, яка забезпечує доступ додаткам до ресурсів комп'ютера: ОЗУ, процесора, пристроїв вводу-виводу, периферії. Ця частина ОС завантажується в оперативну

пам’ять, що дозволяє їй реагувати на переривання та швидко координувати і розподіляти ресурси між додатками.

У 1991 р. Лінус Торвальдс опублікував ядро Linux, що базувалось на стандартах та принципах закладених в ОС Unix. Головною причиною розробки Linux послугувала обмеженість ліцензії UNIX та Minix. Ліцензія GNU GPL та відкритість коду дала можливість усім охочим брати участь у покращенні роботи ядра, що у свою чергу призвело до створення великої кількості UNIX-подібних операційних систем, які базуються на ядрі Linux.

Відкритість ядра Linux дозволила розробляти операційних систем для вирішення різного типу задач: серверні ОС, ОС для мікрокомп’ютерів (Raspberri Pi), ОС для персонального комп’ютера і т. д. Усі подібні рішення прийнято називати – дистрибутивами Linux.

Отже, встановлюючи операційну систему на комп’ютер ви встановлюєте не Linux, а дистрибутив що базується на ядрі Linux. До складу більшості дистрибутивів входить не лише ядро ОС, а й набір інструментів: GNU Tools, net-tools, x server і т. д. До кожного дистрибутиву входить різний набір додаткового програмного забезпечення.

На сьогодні існує чотири основні дистрибутиви Debian, Red Hat, Slackware, Gentoo (Source Based distributive) та безліч похідних від них.

Debian був анонсований у серпні 1993 р. Засновником проекту був Ian Ешлі Мердок (назва дистрибутиву походить від перших літер імені засновника та його дружини **Deb**ra + **Ian**). Серед переваг варто виділити найбільшу кількість доступних для встановлення пакетів, підтримку більшості платформ (у тому числі ARM). На Debian засновується велика кількість похідних дистрибутивів: Knoppix, Linux Mint, Maemo, MEPIS, SteamOS, TAILS, Ubuntu. Менеджер пакетів DPKG, advanced packaging tool.

Дистрибутиви компанії Red Hat. Більшість дистрибутивів цієї компанії позиціонуються, як ОС для корпоративного використання. Існує підтримка продукту на комерційній основі (10–13 років). Одним з напрямів діяльності є підтримка серверної архітектури, хмарних технологій (CentOS). Менеджер пакетів RPM.

Slackware – дистрибутив заснований Патриком Фолькердингом у 1993 р. Розповсюджений серед досвідчених користувачів, знайомих з операційними системами на базі ядра Linux. Slackware дуже гнучкий у налаштуванні, що робить його складною ОС для недосвідчених користувачів. Дистрибутив має невеликий обсяг вбудованих інструментів, що водночас є перевагою та недоліком системи. Менеджер пакетів також відрізняється, відсутністю перевірки залежностей пакетів, а файли пакетів представлені архівами **tgz** або для нових версій **txz**.

Gentoo є дистрибутивом, що підтримує філософію компіляції з додатків коду. Розробником вважається Деніел Роббінсон, який почав розробляти Enoch Linux у 1999 р., на якому базується Gentoo. Для керування пакетами використовується система Portage–аналогічна

технології портів у FreeBSD. Основна ідея дистрибутиву – компіляція додатків з урахуванням архітектури обладнання для збільшення швидкодії. Перевага Gentoo у добре оптимізованому програмному забезпеченні, а основний недолік у складному процесі встановлення та налаштування ОС.

1.2. Встановлення дистрибутиву на ядрі Linux

1.2.1. Встановлення дистрибутиву Debian

Знайомство з роботою ОС на основі Linux, як правило складний для користувача процес. У спільноті Linux, для першого знайомства рекомендують користуватися віртуальними машинами (наприклад VirtualBox).

Першим кроком для встановлення будь-якого дистрибутиву є завантаження інсталятора. На офіційному сайті Debian пропонує одразу декілька варіантів інсталятора: USB – носій, архів для запису на CD або DVD диск, а також тип інсталятора: повний набір інструментів (у тому числі кілька графічних оболонок), урізаний з обмеженою кількістю інструментів та мінімальний за завантаженням під час встановлення з мережі Інтернет.

Після завантаження, встановлення та запису його на відповідний носій необхідно в BIOS обрати завантаження з нього. У разі правильного налаштування користувач бачить меню інсталятора після ініціалізації обладнання (рис. 1.1).

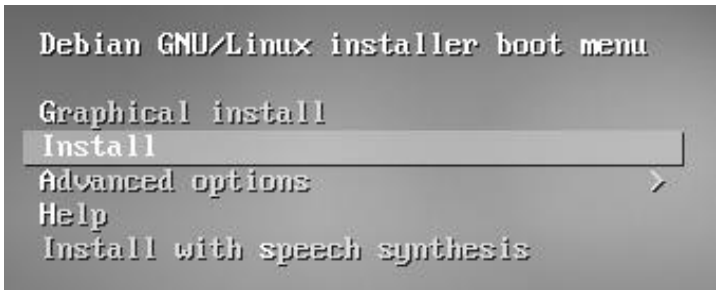

Рис. 1.1. Меню інсталятора Debian

Розробники надають можливість встановлювати ОС у графічному режимі, що дозволить користувачу використати більш звичний інтерфейс, але на функціональному рівні такий спосіб не має переваг над встановленням у консольному режимі.

Не всі пристрої підтримують можливість встановлення у графічному режимі, тому розглянемо консольний режим, у меню обираємо пункт «Install». Після чого бачимо запрошення обрати мову, яка буде використовуватися в ОС, далі необхідно обрати як регіон так і тип кодування.

Під час процесу встановлення користувачу буде запропоновано ввести дані для роботи комп'ютера: його назву, пароль root (користувача з розширеними можливостями) та створити простого користувача (рис. 1.2).

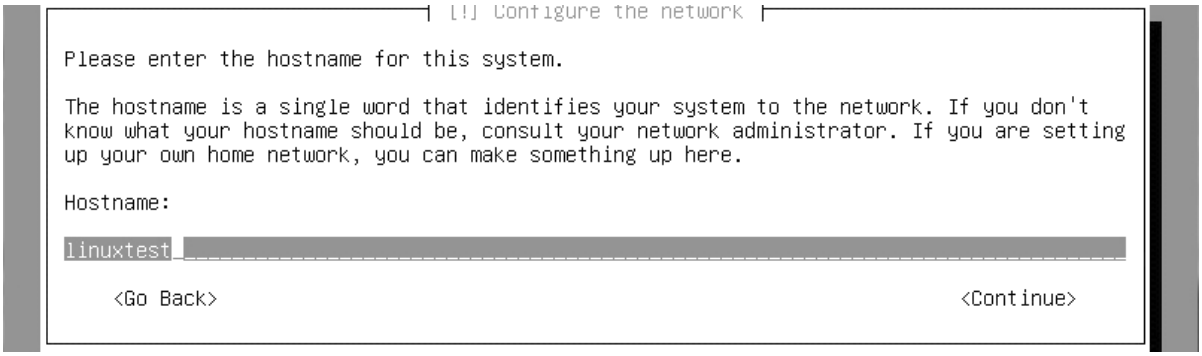

Рис. 1.2. Запрошення ввести назву комп'ютера

Після введення необхідних даних процес встановлення продовжиться, наступним кроком для користувача буде розділення простору на жорсткому диску.

Варто пам'ятати, що спосіб організації даних в ОС на базі Linux відрізняється від ОС Windows. Особливу увагу необхідно звернути на відсутність локальних дисків. Хоча інсталятор пропонує доступний дисковий простір у автоматичному режимі та це не завжди коректно, через наявність кількох фізичних носіїв.

Отже, для розділення дискового простору в ручному режимі, необхідно натиснути «Manual» (рис. 1.3).

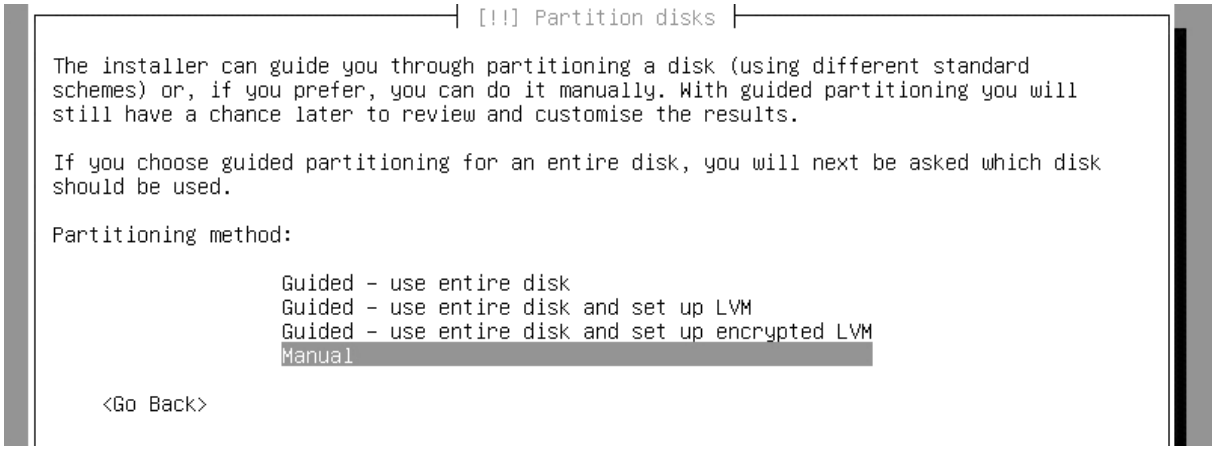

Рис. 1.3. Діалог вибору способу розділення дискового простору

Інсталятор, як правило, виявляє доступні накопичувачі даних та надає користувачу вибір (рис. 1.4).

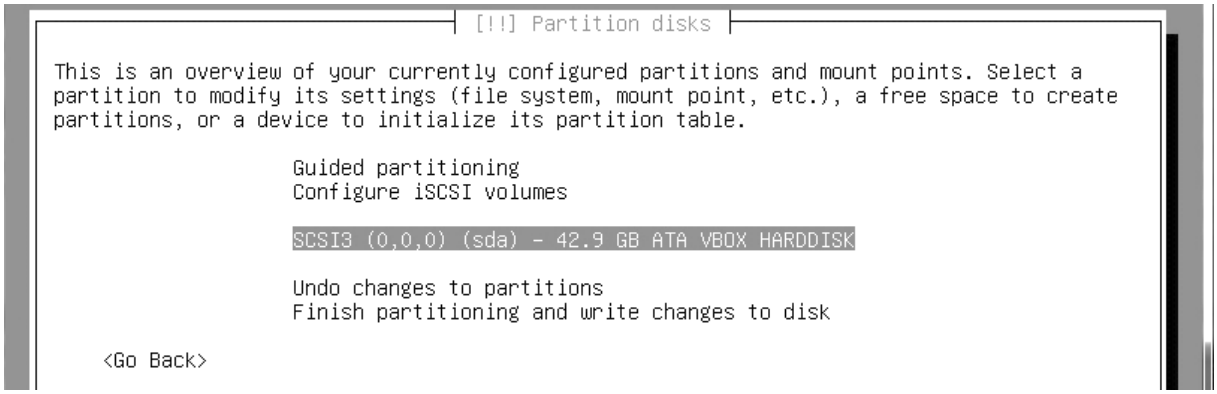

Рис. 1.4. Меню вибору накопичувача

Розподілення диску доволі простий процес. Варто пам’ятати про три основні розділи: SWAP, Home та кореневий каталог.

SWAP – це розділ з особливою розміткою, який слугує для переміщення даних з ОЗУ на жорсткий диск, як правило даних, що не використовуються. Рекомендовано вказувати розмір, що відповідає ОЗУ комп’ютера на який встановлюється ОС.

Home – це каталог, який містить особисті файли користувачів (окрім root), задається розмір відповідно до потреб.

Кореневий каталог містить усі дані системи, та файли необхідні для її роботи, за необхідності користувач з розширеними правами (root) може створювати каталоги та надавати права для роботи з ними. Як правило, займає на диску більшу частину простору (рис. 1.5).

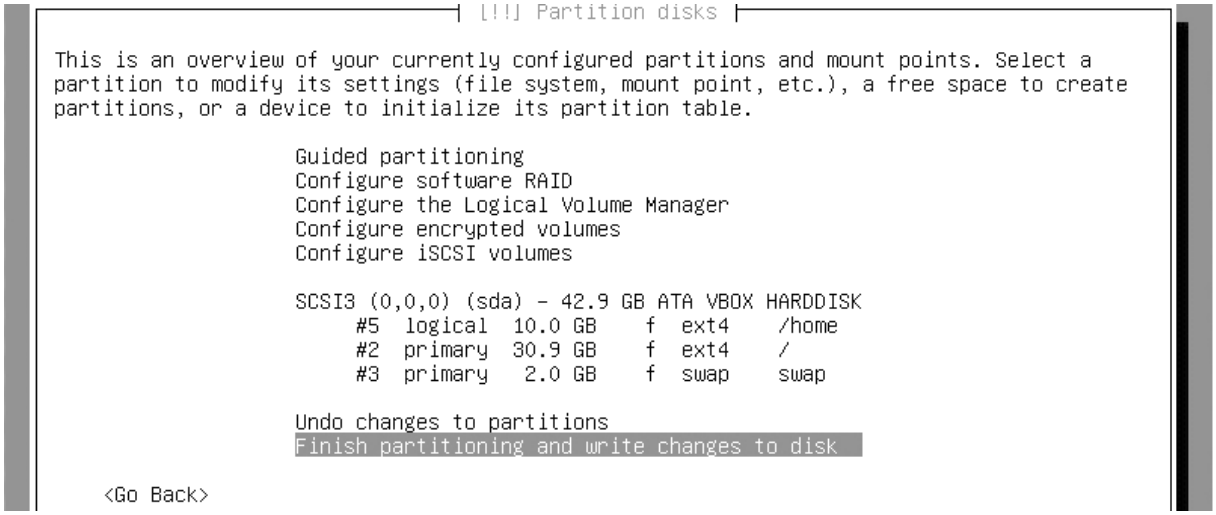

Рис. 1.5. Розподілений дисковий простір

Після розподілення жорсткого диска інсталятор запропонує налаштувати мережу для коректної роботи менеджера пакетів. Спочатку необхідно вибрати регіон та встановити «Ukraine», після чого буде запропоновано список джерел для завантаження (рис. 1.6).

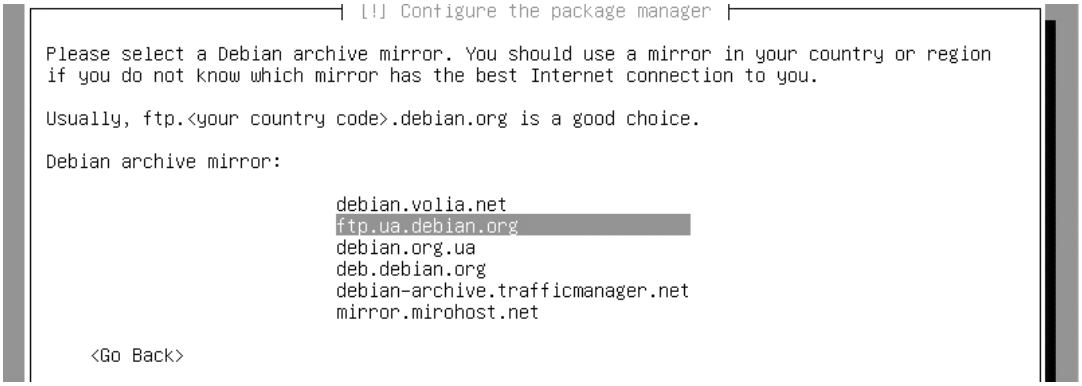

Рис. 1.6. Діалог вибору джерела завантаження пакетів

Останнім етапом встановлення є вибір графічної оболонки та додаткових інструментів (ssh-server, wev-server та графічної оболонки рис. 1.7). Необхідне зі списку можна виділити кнопкою пробілу. За замовчуванням у якості графічної оболонки встановлюється Gnome, але він потребує потужного відеообладнання та може некоректно працювати на старих комп'ютерах.

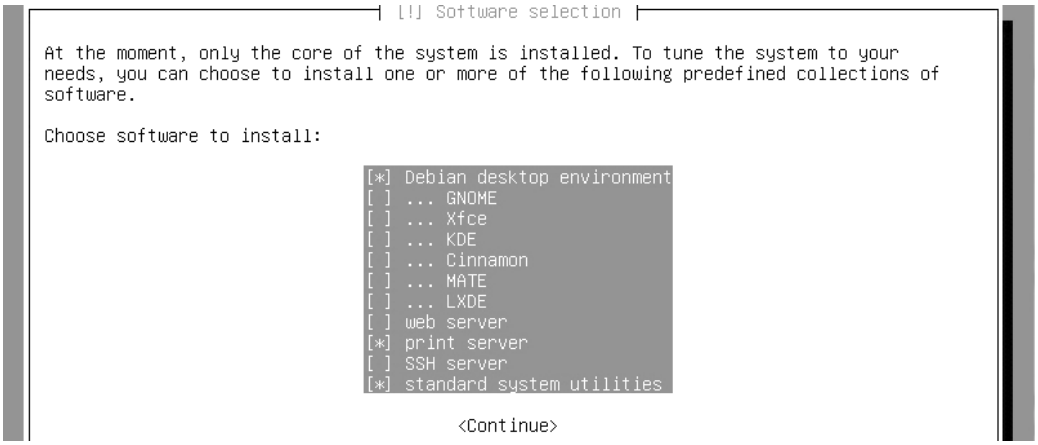

Рис. 1.7. Меню встановлення додаткового програмного забезпечення

Після встановлення додаткового програмного забезпечення система запропонує використовувати GRUB у якості системного завантажувача. Він має бути встановлений на пристрій з кореневим каталогом (рис. 1.8).

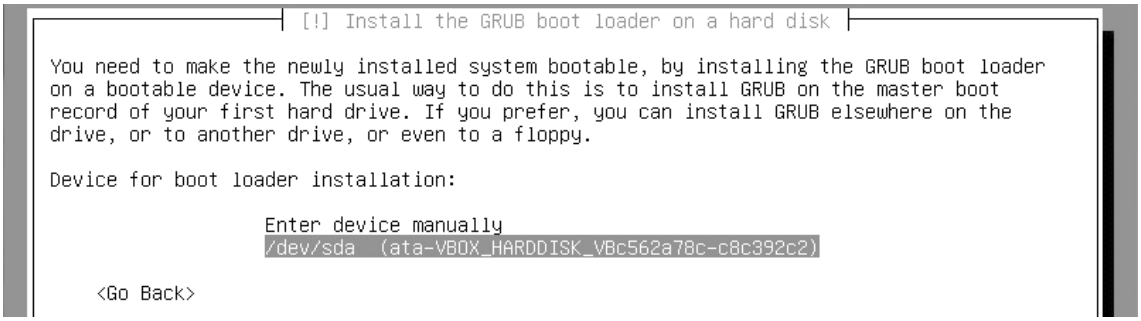

Рис. 1.8. Встановлення GRUB

GRUB – являє собою уніфікований завантажувач операційної системи. Основною функцією якого є надати користувачу можливість запустити встановлену операційну систему, а у випадку коли їх декілька можливість вибрати необхідну. Якщо користувач встановлює ОС на базі Linux «поруч» з Windows то GRUB створює меню для запуску обох ОС.

Якщо встановлення пройшло вдало, користувач після запуску комп'ютера, бачить на екрані запрошення GRUB обрати ОС (рис. 1.9).

Рис. 1.9. Запрошення GRUB обрати ОС для запуску

1.2.2. Встановлення дистрибутиву Fedora

Fedora є проектом компанії Red Hat, призначеним для тестування нових технологій. Призначення цієї ОС робить її найбільш сучасною, але дуже нестабільною.

Дистрибутив використовують користувачі, які вже мають досвід роботи з операційними системами на ядрі Linux але хочуть спробувати нові технології. Технології, які тестуються у зазначеній ОС згодом використовуються в інших продуктах Red Hat.

Отже, першим кроком є створення носія для завантаження (USB, CD/DVD). Завантажити образ диска, як і в минулому випадку можна з офіційного сайту спільноти. Після створення носія для завантаження та налаштування BIOS відкриється відповідне меню (рис. 1.10).

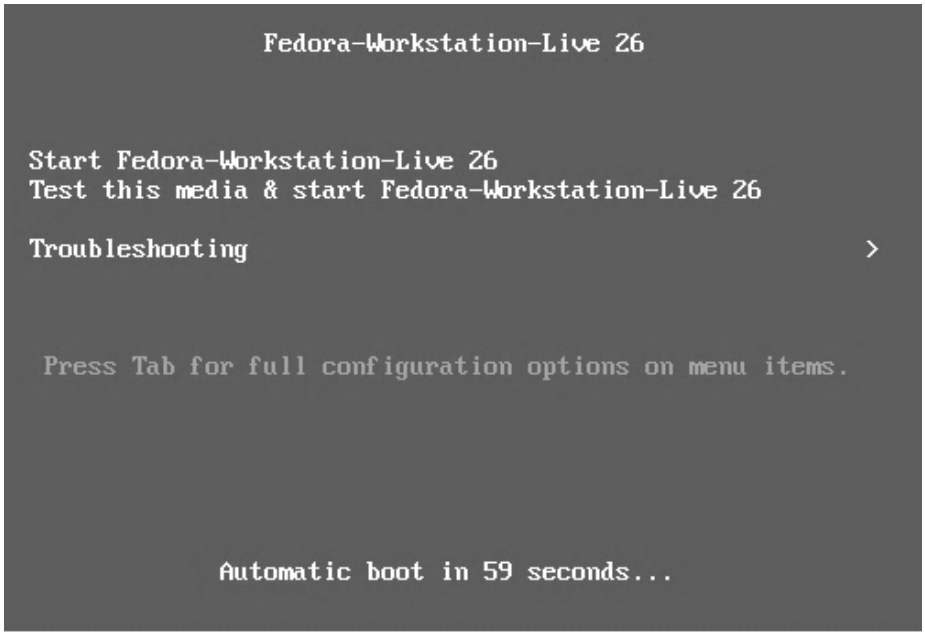

Рис. 1.10. Меню завантажувача Fedora

На відміну від Debian у Fedora відсутня можливість встановлення у консольному режимі, що водночас є недоліком для застарілих комп'ютерів.

Більшість сучасних дистрибутивів на базі ядра Linux дозволяють працювати з ОС без встановлення на жорсткий диск. Отже, для встановлення цього дистрибутиву необхідно запустити Live версію Fedora («Start Fedora-WorkStation-Live»).

Після завантаження користувач має увійти до операційної системи. За замовчуванням запускається графічний режим (графічна оболонка Gnome3), використовуючи комбінацію клавіш «ctrl»+F4 (або іншу клавішу з верхнього ряду) можна переключити систему в консольний режим.

За замовчуванням, графічний режим пропонує завантажитись, як «Live System User», для входу непотрібен пароль. Після авторизації користувачу запропоновано вибір: продовжити роботу в системі без встановлення або встановити ОС на жорсткий диск (рис. 1.11).

Рис. 1.11. Live Fedora

Обравши «Try Fedora», користувач отримає можливість працювати в ОС без її встановлення, але з обмеженими можливостями. Подібний принцип іноді використовується для роботи з ПК без жорсткого диска (наприклад термінал для підключення віддаленого робочого столу: RDP, VLC та ін.).

Fedora має дуже широкий спектр можливостей, що виражено кількістю доступного програмного забезпечення, тому використання її у якості операційної системи без встановлення вважається недоцільним.

Починаючи встановлення, користувач бачить запрошення обрати мову. Бажано обрати англійську (us), в іншому випадку користувач не зможе вказати розділи, при розподіленні дискового простору.

Обравши мову, користувач переходить до основного меню інсталятора (рис. 1.9). Це меню містить всього 4 пункти: клавіатура – налаштування розкладки відбувається під час вибору мови, дата і час – необов'язковий пункт налаштувань, але бажано зазначити часовий пояс, пункт меню який відповідає за розбиття простору на жорсткому диску називається «призначення системи», і останній пункт це налаштування мережі (рис. 1.12).

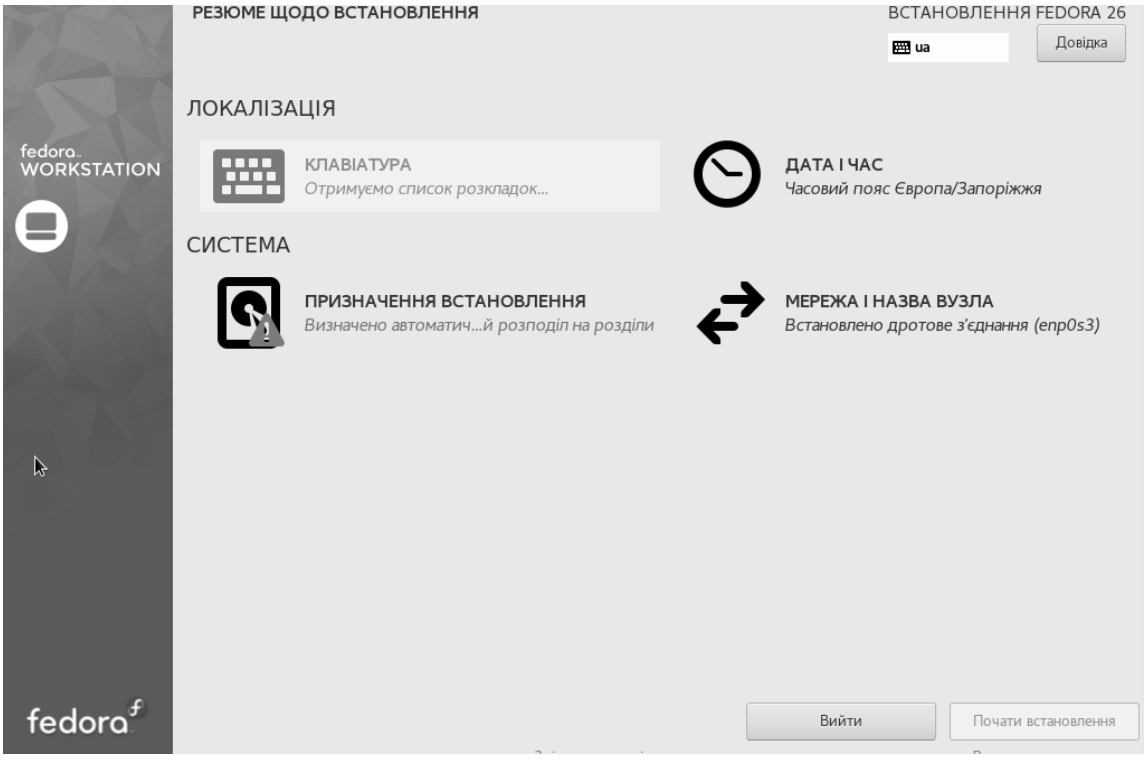

Рис. 1.12. Головне меню інсталятора Fedora

Для встановлення необхідно розподілити дисковий простір, як і минулого разу це можна зробити автоматично або в ручному режимі. Користувачу пропонують обрати з наявних жорстких дисків, той на який саме буде встановлено операційну систему (рис. 1.13).

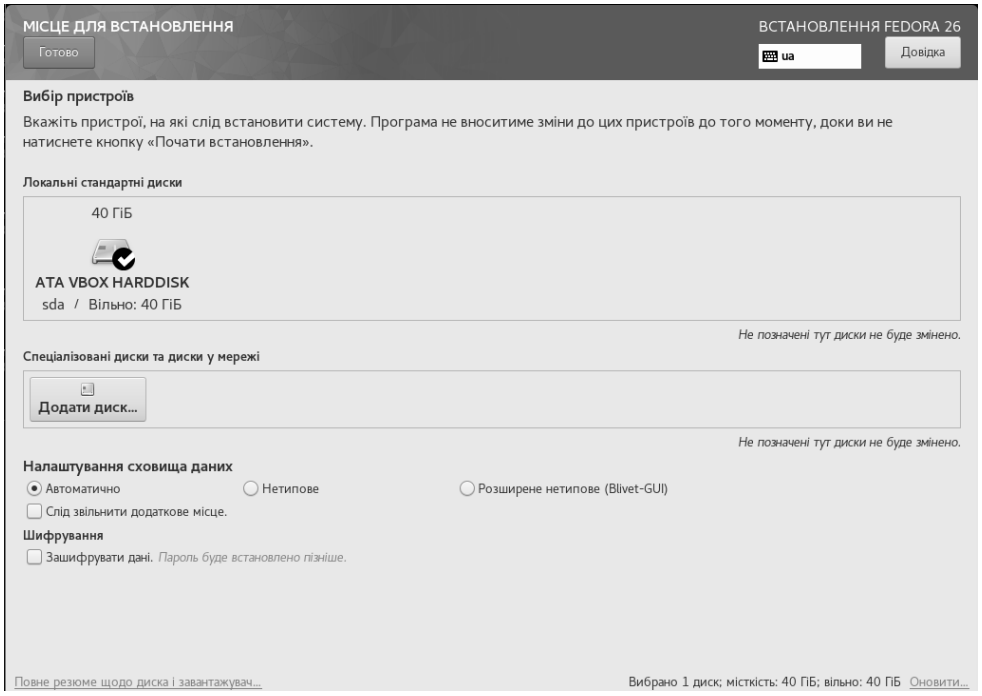

Рис. 1.13. Меню вибору жорсткого диска для встановлення ОС

Коли носій буде обрано необхідно розподілити вільний дисковий простір. Вибір точки монтування та розмір розділу представлений спеціальним діалоговим вікном (рис. 1.14).

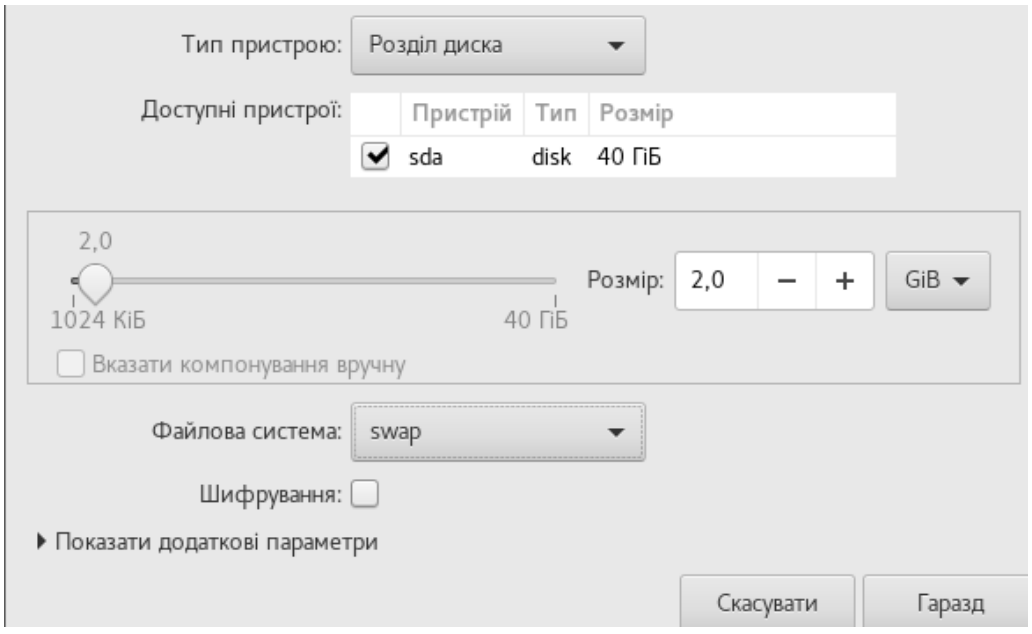

Рис. 1.14. Діалог розподілення дискового простору

Основними розподілами, як і в інших дистрибутивах є файл підкачки та кореневий каталог, окрім того, спільнота рекомендує виділити простір під домашні каталоги. Після розподілу простору користувач має побачити зміни (рис. 1.15).

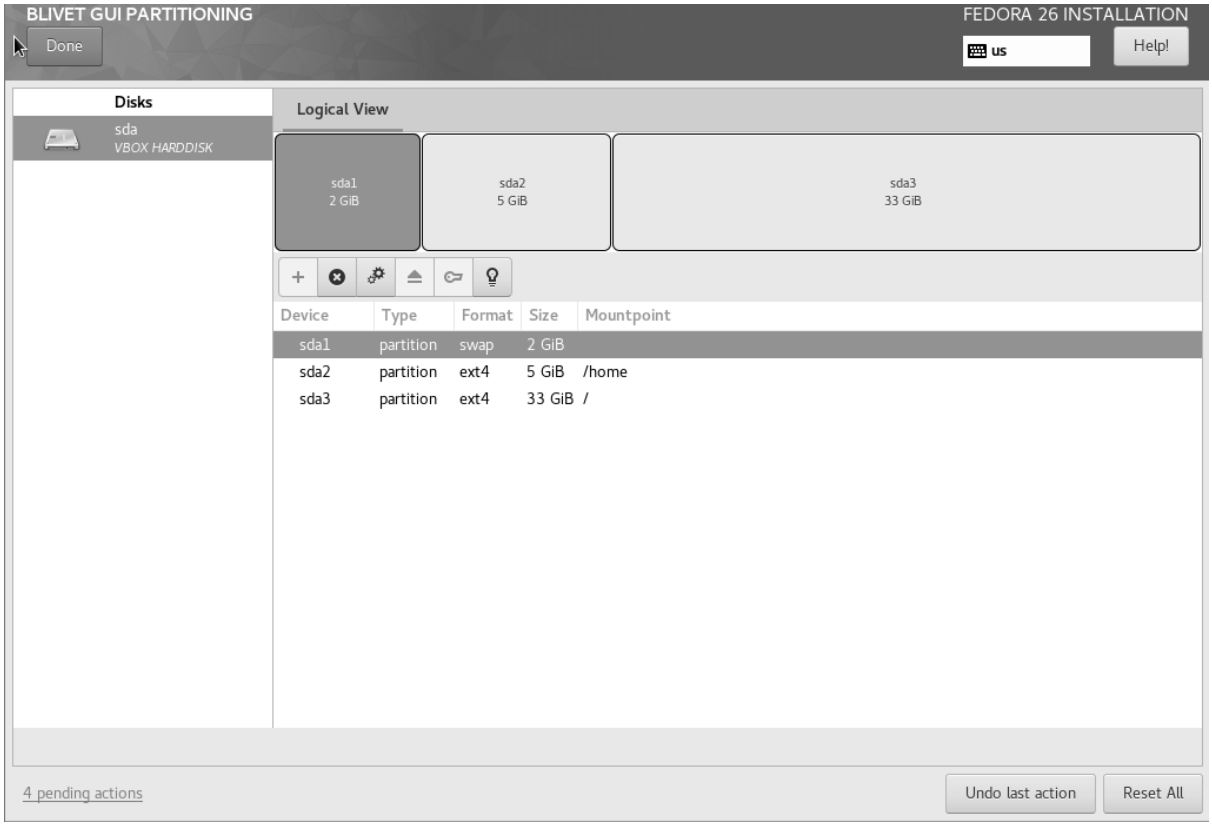

Рис. 1.15. Розподілений дисковий простір інсталятора Fedora

Після внесення всіх змін необхідно натиснути кнопку «Done», та зберегти зміни на жорсткому диску (рис. 1.16).

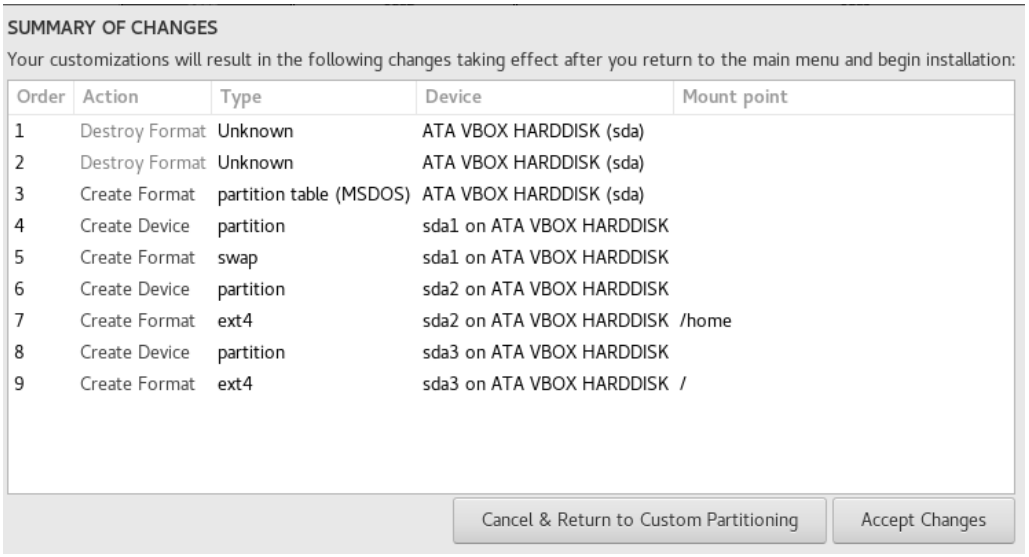

Рис. 1.16. Збереження змін дискового простору Fedora

Тепер у головному меню користувачу доступна кнопка «Почати встановлення». Після його завершення користувач має перезавантажити комп'ютер та вийняти диск з інсталятором. Якщо встановлення відбулося коректно користувач побачить меню GRUB (рис. 1.17).

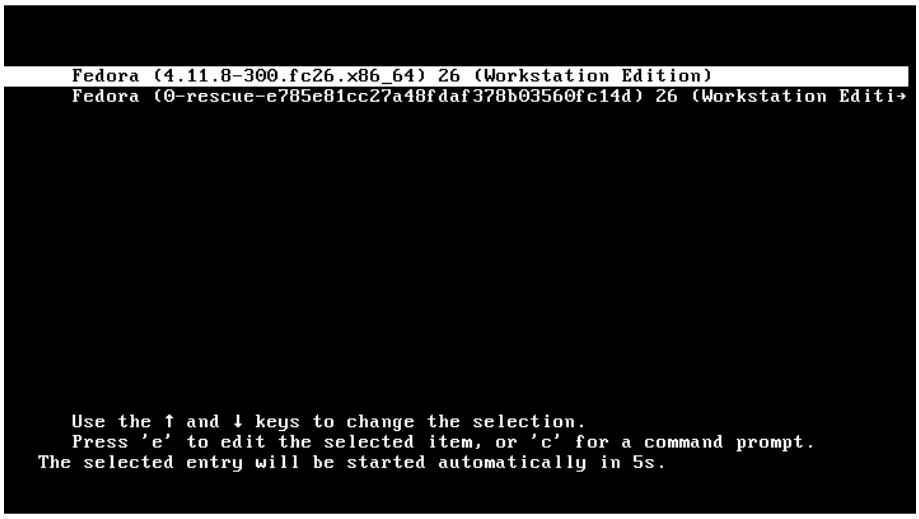

Рис. 1.17. Меню GRUB для запуску ОС Fedora

1.2.3. Встановлення дистрибутиву Gentoo

Дистрибутив Gentoo бере свій початок з дистрибутиву Enoch Linux, який був розроблений Деніелсом Робінсоном у 1999 р. Перший реліз відбувся 2002 р., основна перевага системи полягає у компіляції програмного забезпечення.

На відміну від інших дистрибутивів що базуються на ядрі Linux, Gentoo має спеціалізований пакетний менеджер Portage, що на відміну від інших працює за принципом портів системи FreeBSD. Головним засобом, для встановлення, є код, а небінарні файли (часто зазначений принцип називають «from source»). Отже, встановлення будь-якої утиліти або програмного засобу на цьому дистрибутиві є компіляція.

Перевагою операційної системи є швидкодія, адже кожен програмний засіб компілюється під конкретну архітектуру, але набір інструкцій I386 або AMD64 буде однаковим для бінарних файлів, тому на сучасному обладнанні користувач не отримує значних переваг у швидкодії, але витрачає час на компіляцію програмних засобів.

Початковий набір утиліт та програмних засобів є недоліком та перевагою одночасно. Під час встановлення відсутні інструменти для роботи з мережею (наприклад DNS client), але водночас відсутність непотрібних користувачу інструментів зменшує кількість обчислювальних ресурсів, що необхідні системі.

Мінімальні системні вимоги дистрибутиву Gentoo: процесор архітектури AMD64 або EM64T, ОЗУ: 256 МБ, вільний простір на жорсткому диску: 2.5 ГБ.

Розпочати встановлення необхідно з вибору носія даних для встановлення. Існує два варіанти: minimal release – для подальшого встановлення через мережу Інтернет та LiveDVD – містить усі необхідні пакети для встановлення.

Встановлення цього дистрибутиву доволі складний процес, спочатку достатньо обрати minimal release. Встановлення Gentoo своїм принципом схоже на встановлення дистрибутиву Fedora.

Після запису на носій та налаштування BIOS ми бачимо меню запрошення користувача обрати варіант запуску: натиснувши F1 – користувач може запустити встановлену операційну систему, натиснувши F2 – обрати запуск Gentoo носія для інсталяції (рис. 1.18).

Рис.1.18. Меню початку запуску встановлення Gentoo

Особливістю цього меню є те, що користувачу необхідно вводити назву операційної системи повністю, тобто, рядок запрошення для початку встановлення Gentoo матиме наступний вигляд: boot : gentoo. У більшості операційних систем та у завантажувачі Grub надається можливість обирати ОС натисканням клавіш стрілочок. Після вводу користувачем назви операційної системи для завантаження розпочинається процес запуску Live версії Gentoo (рис. 1.19).

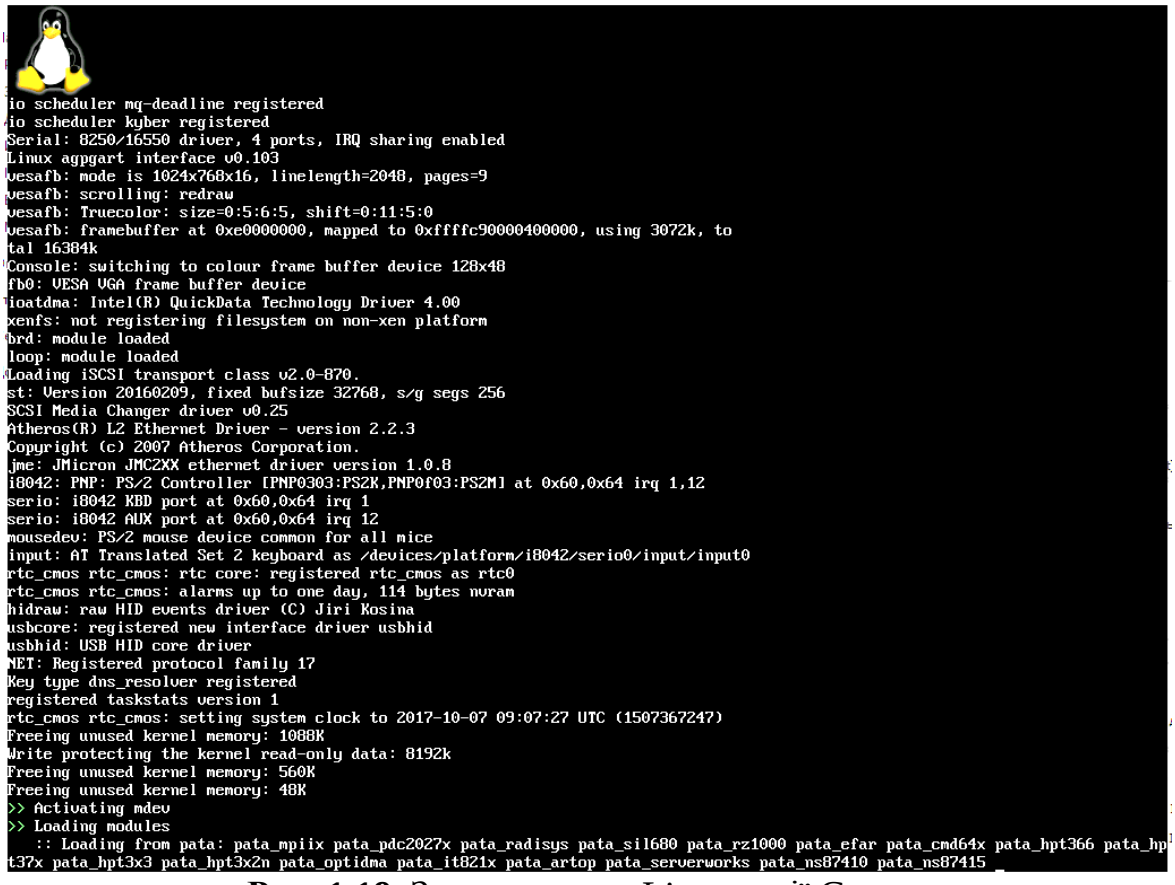

Рис. 1.19. Завантаження Live версії Gentoo

Завантаживши систему, користувач бачить рядок запрошення, наступним його кроком є розподіл жорсткого диска. Перш ніж розпочати розподіл простору необхідно перевірити наявність жорстких дисків, для цього можна використати команду: ***fdisk -l***, яка показує усі наявні для запису носії (рис. 1.20).

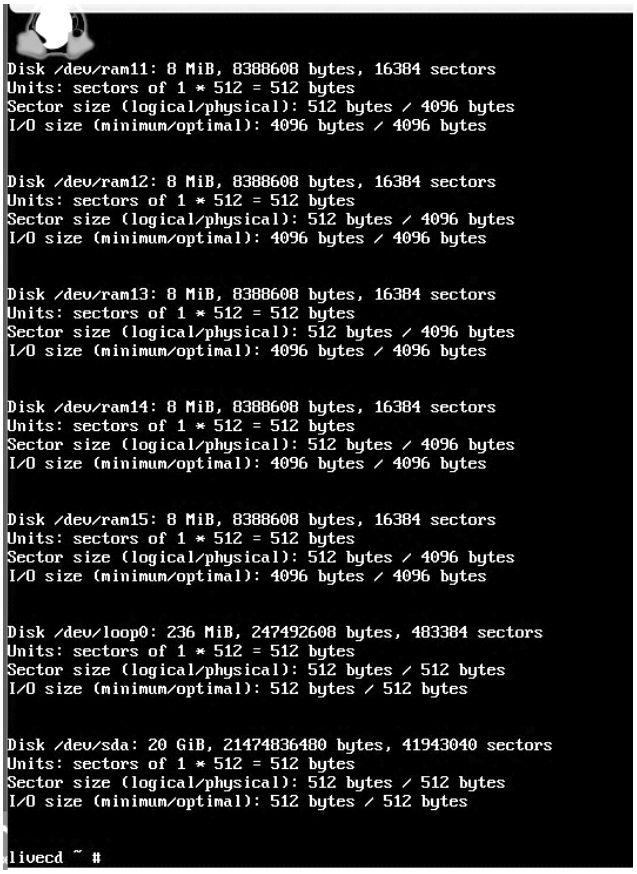

Рис. 1.20. Перегляд носіїв доступних у системі

Як правило, жорсткий диск позначається ***/dev/sda***. Для розмітки диска можна використовувати утиліту **parted (parted /dev/sda**. Вибрана утиліта має певний набір команд, наприклад для перегляду стану диска необхідно ввести print (рис. 1.21).

Рис. 1.21. Відображення інформації утилітою parted

Утиліта parted підтримує декілька варіантів створення розділів: *mkpart, mkpartfs*. Ми будемо використовувати варіант mkpart, який має наступну структуру: (parted) mkpart тип розділу [тип_FS] START END, де тип розділу може бути: primary, logocal та extended, наступний параметр [тип_FS] не обов'язковий, розділ можна форматувати після створення.

Розмітка диска має розпочинатися зі створення нової, порожньої таблиці розділів диска: mklabel msdos, зробимо розділ завантажувача системи set 1 boot on, розмітимо для цього частину дискового простору: mkpart primary ext4 2M 200. Після чого, необхідно виділити частину простору для створення файлу підкачки (SWAP): ***mkpart primary linux-swap 200M 2200M***. Дистрибутив Gentoo вимагає мінімальний розмір файлу підкачки 256 МБ, але як і в інших дистрибутивах бажано вказувати розмір файлу підкачки відповідно до розміру ОЗУ. Решту простору виділяємо під кореневий каталог: mkpart primary ext4 2.2G 100 %. Результатом розподілу дисків має бути таблиця розділів (рис. 1.22).

```
livecd ~ # parted /dev/sda
GNU Parted 3.2
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sda: 21.5GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      2097kB  200MB   198MB   primary               boot
 2      200MB   2200MB  2000MB  primary
 3      2200MB  21.5GB  19.3GB  primary

(parted)
```

Рис. 1.22. Таблиця розділів жорсткого диска для встановлення Gentoo

Parted повідомляє про відсутність файлової системи, отже створені розділи необхідно відформатувати. Форматування в ОС на ядрі Linux здійснюється за допомогою утиліти mkfs. Почнемо розмічати диски з підключення розділу підкачки: ***mkswap /dev/sda2 -L swap*** та безпосередньо його монтування: swapon /dev/sda2. Після цього форматуємо розділ завантажувача системи: ***mkfs.ext4 /dev/sda1*** та кореневого каталогу: ***mkfs.ext4 /dev/sda3***.

Монтуємо створені розділи (без файлу підкачки): mount ***/dev/sda3 /mnt/gentoo/*** – кореневий каталог, далі створюємо каталог для завантажувача у який буде монтуватися відповідний розділ: ***mkdir/mnt/gentoo/boot*** і монтування: ***mount /dev/sda1/mnt/gentoo/boot***.

Далі для встановлення ОС правильно вказуємо джерело, дистрибутив Gentoo у якості джерел використовує систему stage. Існує три рівні stage – архіву:

1) stage1 – використання цього джерела є найбільш оптимальним для швидкодії системи. Цей архів містить систему, але користувачу необхідно провести ряд налаштувань та скомпілювати базову систему, що призводить до оптимальної компіляції під відповідне апаратне забезпечення;

2) stage2 – мінімальна система (base layout) частково скомпільована, але додаткові компоненти користувачу доведеться компілювати самостійно та завантажувати з мережі Інтернет під час компіляції ядра;

3) stage3 – мінімальна система (base layout) скомпільована, додаткові елементи також завантажені та частково скомпільовані. Встановлення з цього джерела швидше, але система менш оптимізована під апаратне забезпечення.

Розпочати завантаження доволі просто, у більшості дистрибутивів на базі ядра Linux встановлена утиліта wget. Встановлення системи з stage3 рекомендовано для користувачів, які мають незначний досвід роботи з ОС на базі ядра Linux. Отже, для продовження встановлення завантаження вводимо: ***«cd/mnt/gentoo/ && wget http://distfiles.gentoo.org/releases/amd64/autobuilds/current-stage3-amd64/stage3-amd64-20171109.tar.bz2».***

Завантажується архів, але для подальшої роботи його треба розархівувати, командою: ***cd /mnt/gentoo/ && tar xvjpf stage3-*.tar.bz2 && rm -f stage3-*.tar.bz2***. Далі необхідно вибрати джерело (дзеркало) для встановлення додаткових проектів: **mirrorselect -i -o >>/mnt/gentoo/etc/portage/make.conf.**

Налаштування DNS можна скопіювати з завантаженої системи: ***cp -L /etc/resolv.conf /mnt/gentoo/etc/***. Подальше встановлення вимагає монтування наступних директорій: **proc, dev, sys.**

```
livecd ~ # mount -t proc /proc /mnt/gentoo/proc
livecd ~ # mount -o rbind /dev/ /mnt/gentoo/dev/
livecd ~ # mount --make-rslave /mnt/gentoo/dev
livecd ~ # mount --rbind /sys /mnt/gentoo/sys
livecd ~ # mount --make-rslave /mnt/gentoo/sys
```

Необхідно встановити додаткове програмне забезпечення, цей крок виконується шляхом переходу в створене середовище та виклику менеджера керування пакетами:

```
# chroot /mnt/gentoo/ /bin/bash
# env-update && source /etc/profile
# export PS1="(chroot) $PS1
```

У якості менеджера пакетів використовується, portage, який для початку роботи вимагає створення директорії: mkdir /usr/portage. Наступним кроком роботи з менеджером є створення «знімка» пакетів: emerge-webrsync та оновлення ***emerge –sync***.

Коректне встановлення пакетів вимагає налаштування часового поясу: ***ln -sf /usr/share/zoneinfo/Europe/Kiev /etc/localtime*** та локалізації яка відбувається у два етапи:

1) перегляд доступних локалізацій ***eselect locale list***, команда має наступний вивід (зірочкою відмічено вибране):

```
eselect locale list
  [1]   C
  [2]   POSIX *
  [3]   en_US.utf8
  [4]   ru_RU.utf8
  [ ]   (free form)
```

2) для запобігання конфлікту локалізації бажано вибрати US: ***eselect locale set 3***.

Одним з компонентів цього дистрибутиву є графічна оболонка, налаштування якої можливе командою ***eselect profile list***, яка видає список доступних середовищ та ***eselect profile set 7*** де 7 – номер середовища, що відповідає **plasma/systemd.**

Усі попередні кроки є налаштуванням перед початком збірки ОС, якщо всі описані кроки виконані, можна розпочати встановлення. Недосвідченим користувачам спільнота Linux рекомендує використовувати інструмент genkernel:

```
emerge gentoo-sources -av
emerge sys-kernel/genkernel -av
```

Genkernel – утиліта, була розроблена спільнотою Gentoo, для попереднього налаштування, компілювання ядра та завантажувача системи. За необхідності можна змінити безпосередньо налаштування Genkerne: nano /etc/genkernel.conf, але у більшості випадків додаткові налаштування не потрібні. Перед запуском компіляції ядра бажано провести попередні налаштування: ***genkernel --menuconfig all*** (рис. 1.23).

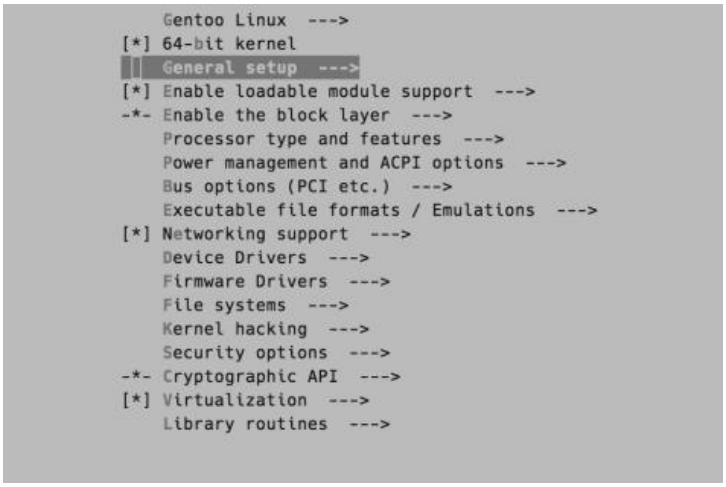

Рис. 1.23. Меню налаштуваня ядра

Виконуючи налаштування необхідно вказати підтримку файлової системи ext2-4. Після чого можна почати компілювати дистрибутив: ***genkernel all***. Цей процес може займати дуже багато часу (можливо понад добу).

На відміну від попередніх дистрибутивів Gentoo вимагає налаштування в режимі Live, а саме: налаштування мережі, монтування розділів, встановлення завантажувача та ін.

Розпочнемо встановлення додаткових утиліт із встановлення текстового редактора Vi та інструментів роботи з мережею: ***emerge vim net-tools net-dns/bind-tools –av.***

У цьому дистрибутиві не встановленні системи логування за замовчуванням, отже їх встановлення можливе командою: ***emerge logrotate syslog-ng vixie-cron udev sys-fs/e2fsprogs –av***, а додавання до автозаваження командою: ***rc-update add syslog-ng default, rc-update add vixie-cron default, rc-update add sshd default, rc-update add udev boot.***

Налаштування мережі в Live та встановленої системи відрізняються, тому бажано налаштувати мережу до запуску встановленої системи, отже почнемо з редагування файлу */etc/conf.d/hostname*, де вказуємо мережеву назву: *HOSTNAME="gentoo-linux"*. Робота в мережі з DHCP сервером, потребує встановлення ***dhcpcd: emerge --ask net-misc/dhcpcd***, після чого проводимо конфігурацію **/etc/conf.d/net:**

```
config_eno33554952="dhcp"
routes_eno33554952="default via 192.168.0.1"
```

Команда *ifconfig* виводить інформацію про всі доступні інтерфейси, разом з детальною інформацією:

```
(chroot) livecd / # ifconfig
eno33554952:
flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.1.126 netmask 255.255.255.0 broadcast
192.168.1.255
        inet6  fe80::20c:29ff:fef4:f752  prefixlen  64
scopeid 0x20<link>
        inet6       fd86:94c4:705c:0:20c:29ff:fef4:f752
prefixlen 64 scopeid 0x0<global>
        ether 00:0c:29:f4:f7:52 txqueuelen 1000 (Ethernet)
        RX packets 367 bytes 43349 (42.3 KiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 86 bytes 7883 (7.6 KiB)
        TX  errors  0  dropped  0  overruns  0  carrier  0
collisions 0
```

Налаштувавши */etc/conf.d/net*: необхідно створити посилання та додати його до автозавантаження:

(chroot) livecd / # ln -s /etc/init.d/net.lo /etc/init.d/net.eno33554952

(chroot) livecd / # rc-update add net.eno33554952 default

Після чого встановимо набір для адміністрування дистрибутиву Gentoo: ***emerge gentoolkit –av.***

Останнім кроком встановлення є налаштування встановлювача, але на відміну від попередніх дистрибутивів за замовчуванням grub не встановлений. Розпочнемо встановлення з імпорту UUID розділів: ***blkid | grep sda >> /etc/fstab***. Бажано перевірити імпорт: ***cat /etc/fstab | grep -Ev "#|^$",*** має вивести приблизно наступний текст:

/dev/sda1: UUID="2dee3fa9-1c6c-4b7d-97d8-3412e2a57eec" TYPE="ext4" PARTUUID="7d59c901-01"

/dev/sda2: LABEL="swap" UUID="2e04f1a7-604b-4ec4-8e70-399913b3b3ae" TYPE="swap" PARTUUID="7d59c901-02"

/dev/sda3: UUID="a2f91a0c-f9d9-4f4b-bc44-f0673f0647dc" TYPE="ext4" PARTUUID="7d59c901-03"

Існує декілька типів BIOS, перед встановленням бажано налаштувати систему portage для вашого типу *vim /etc/portage/make.conf :*

1). Стандартний ***PC (BIOS):GRUB_PLATFORMS="pc";***

2). ***UEFI*** на ***amd64:GRUB_PLATFORMS="efi-64";***

3). Обидва ***UEFI*** и **PC** ***:GRUB_PLATFORMS="efi-64 pc"***.

Завершивши налаштування, розпочнемо встановлення GRUB: *emerge --ask --newuse --deep sys-boot/grub:2*. Завантаження та компіляція пакетів може займати декілька годин, після встановлення необхідно налаштувати MBR диска. Головний завантажувальний запис (*master boot record, MBR*) – налаштовується утилітою *grub-install: grub-install /dev/sda* (у деяких випадках необхідно перед початком створити директорію «mkdir /boot/grub»).

На цьому етапі встановлення завершено, для подальшої коректної роботи необхідно задати пароль root: *passwd* та створити нового користувача: ***useradd -m -G users,wheel,audio -s /bin/bash TestUser*** і задаємо новий пароль: ***passwd TestUser***.

Отже, ОС Gentoo встановлено та налаштовано, тепер перезавантажимо операційну систему для виходу з режиму Live (перезавантаження командою reboot).

Після коректного встановлення запускається консоль з рядком запрошення (рис. 1.24).

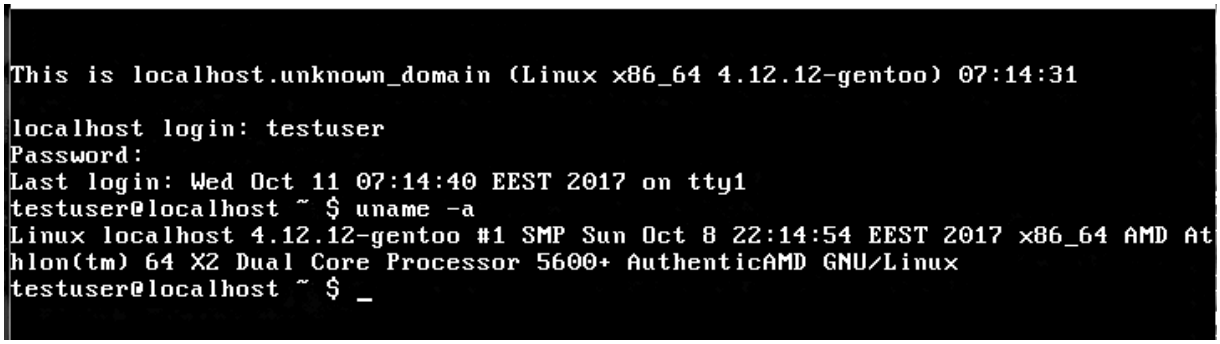

Рис. 1.24. Консоль з рядком запрошення

1.2.4. Встановлення дистрибутиву Slackware

Slackware заведено називати «найбільш UNIX'овим», є одним з найстаріших дистрибутивів, створений Патріком Волкердінґом у 1993 р. Дистрибутив підтримує лише стабільні пакунки. Спільнота Slackware використовує вислів «Якщо ви знаєте RedHat, то все, що ви знаєте – це RedHat. Якщо ви знаєте Slackware, ви знаєте Linux» у якості девізу.

Як і у двох минулих дистрибутивах, встановлення розпочинається з запуску Live системи (рис. 1.25). Після появи екрану запрошення для продовження встановлення необхідно натиснути Enter.

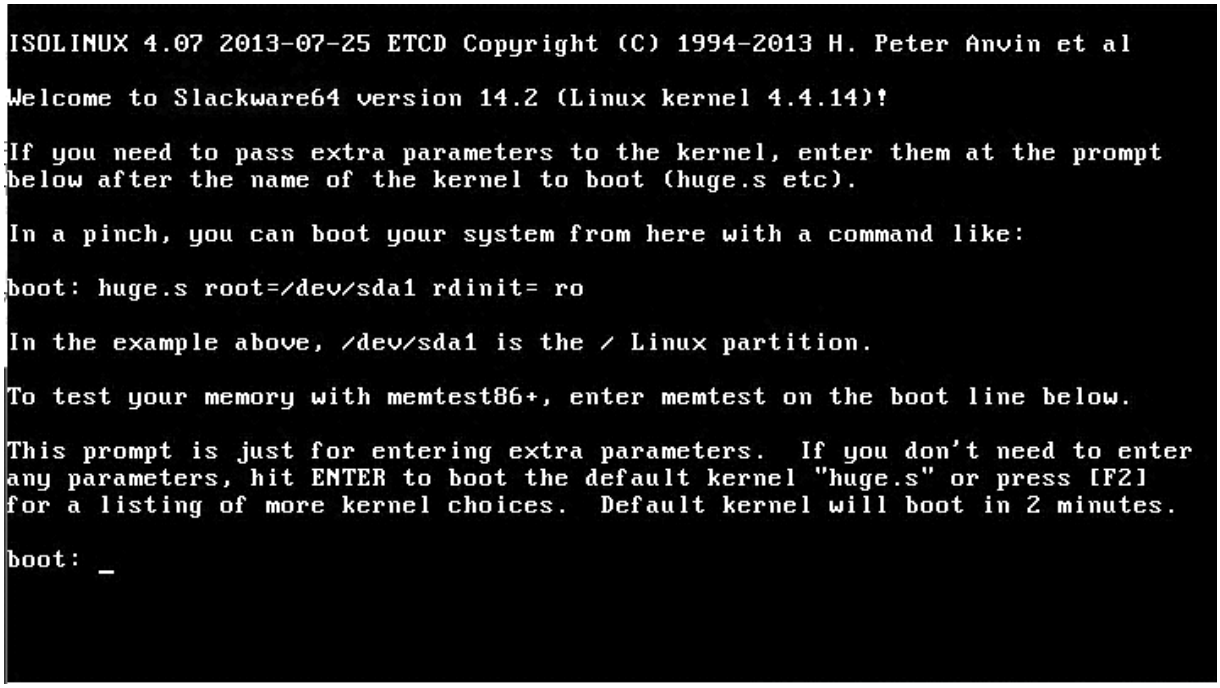

Рис. 1.25. Екран запрошення для запуску Slackware

На початку встановлення ОС запитує розкладку клавіатури (рис. 1.26), меню вибору представлено консольно (рис. 1.27).

Рис. 1.26. Запрошення вибрати розкладку

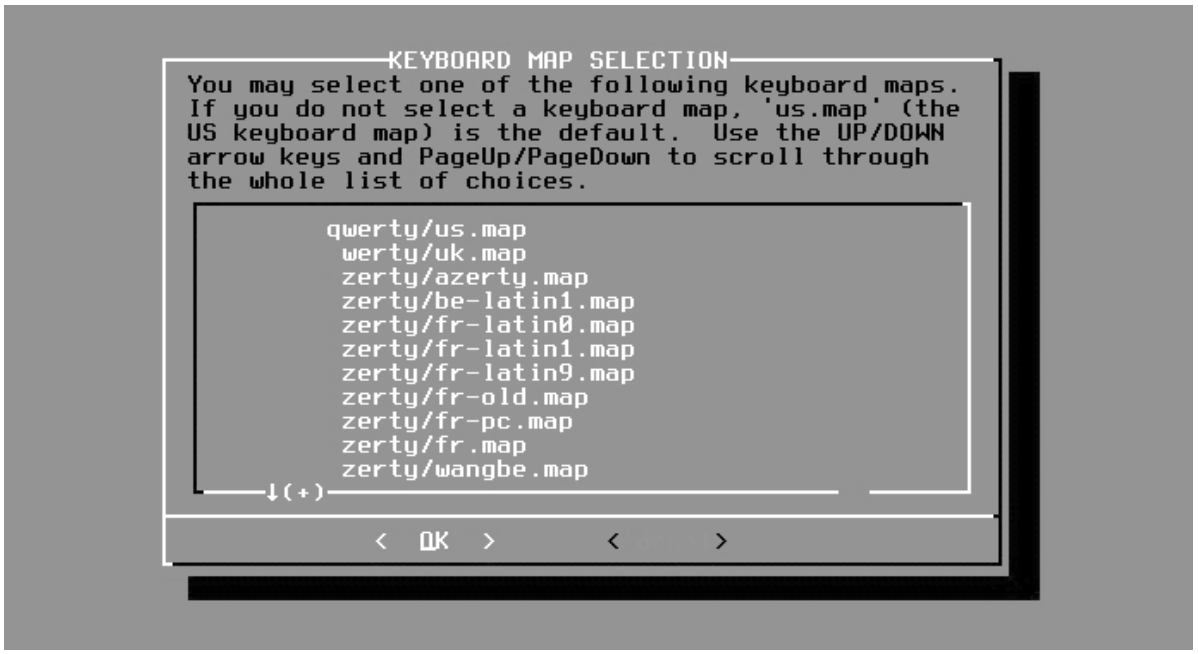

Рис. 1.27. Меню вибору розкладки клавіатури

Обравши розкладку, необхідно авторизуватися в системі (рис. 1.28), у Live версії доступний лише користувач root, що за замовчуванням не має пароля.

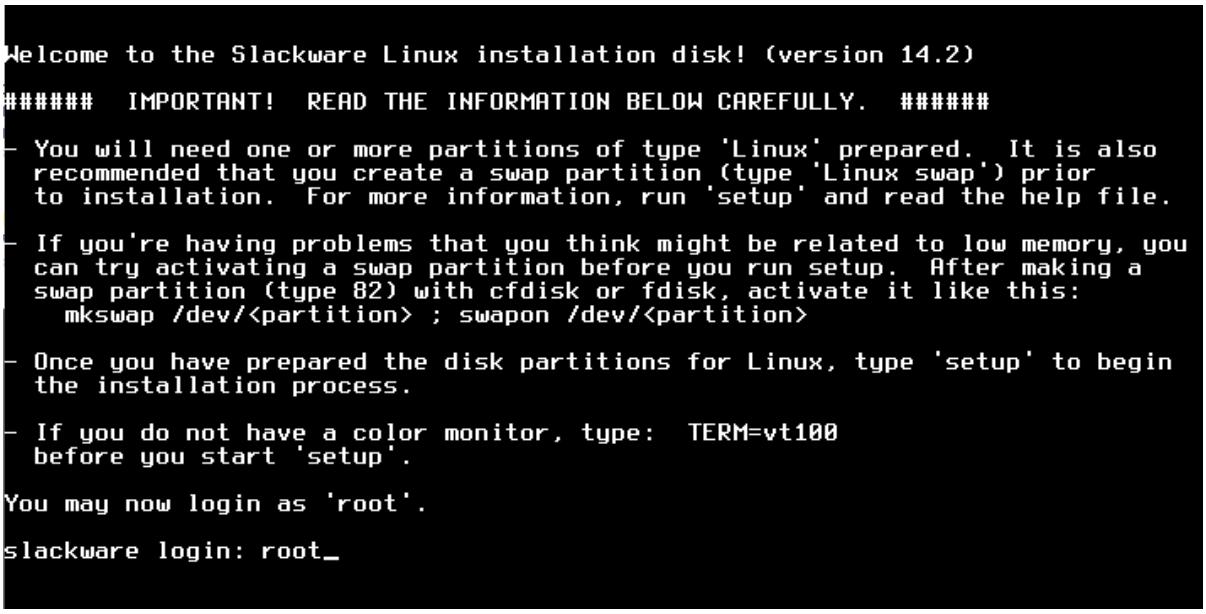

Рис. 1.28. Авторизація у Live версії

Наступний крок встановлення – розподіл жорсткого диска: перегляд наявних інтерфейсів необхідно здійснити утилітою ***fdisk -l***. Визначивши необхідний носій, розподіл простору здійснюється командою ***cfdisk /dev/sda1*** (де sda1 – ваш диск).

Cfdisk починає роботу з запиту мітки, як і у попередніх дистрибутивах виставляємо (dos). Найпростіше встановлення вимагає створення всього двох розділів: *SWAP* та кореневого. У результаті коректного розподілу утиліта виведе розділи (рис. 1.29).

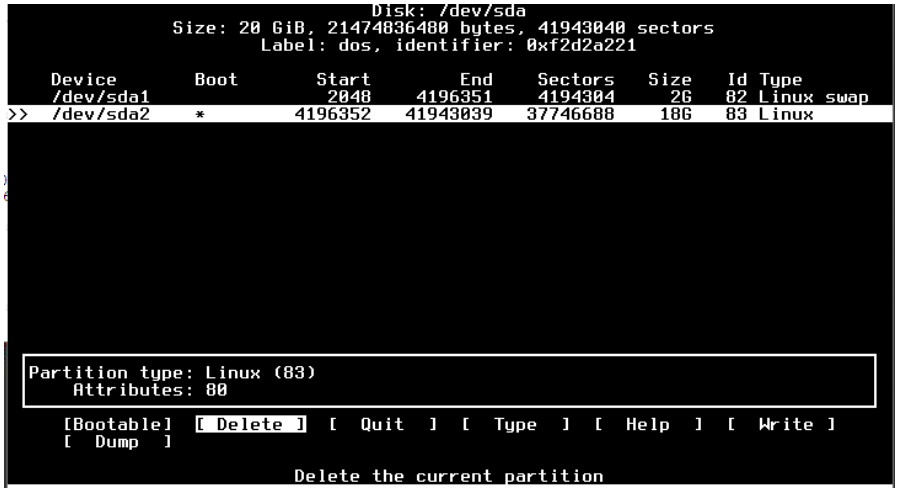

Рис. 1.29. Розподіл дискового простору утилітою cfdisk

Розподіливши простір, необхідно буде відформатувати розділи. Форматування можна виконати утилітою ***mkfs.ext4***.

Встановлення дистрибутиву починається запуском утиліти ***Setup*** (рис. 1.30). Після цього необхідно під'єднати SWAP розділ.

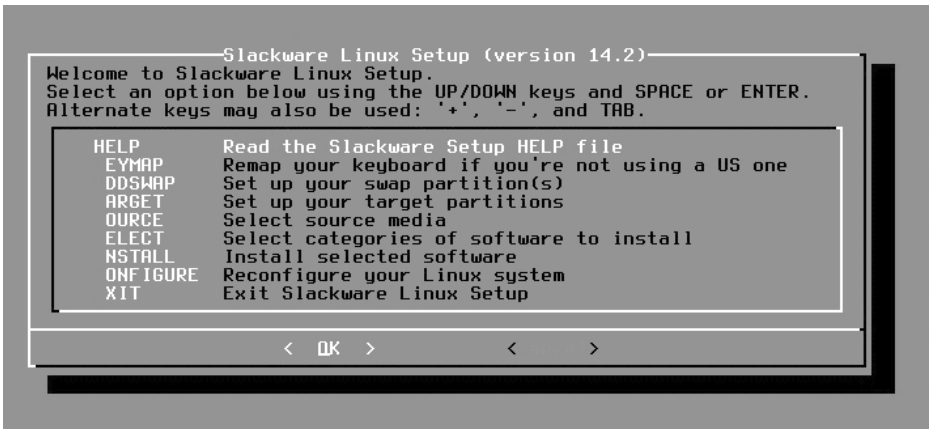

Рис. 1.30. Меню утиліти ***Setup***

Під'єднавши область підкачки, необхідно розпочати запис системних файлів на жорсткий диск (пункт меню «Install»), але спочатку потрібно вибрати джерело встановлення (рис. 1.31).

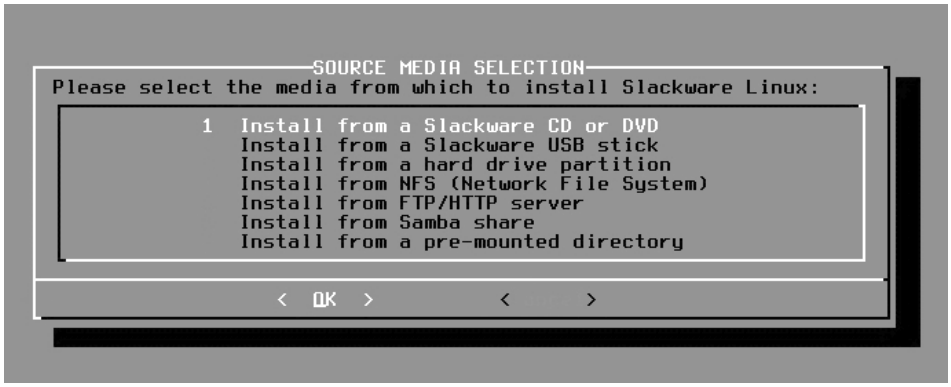

Рис. 1.31. Меню вибору джерела встановлення Slackware

На наступному кроці інсталятор пропонує вибрати комплектацію ОС (за кількістю базових пакетів), що буде встановлено (рис. 1.32).

Рис. 1.32. Меню вибору комплектації Slackware

Однією з особливостей цього дистрибутиву є завантажувач, на відміну від попередніх дистрибутивів використовується не Grab, а LILO, що встановлюється разом з системою (рис. 1.33).

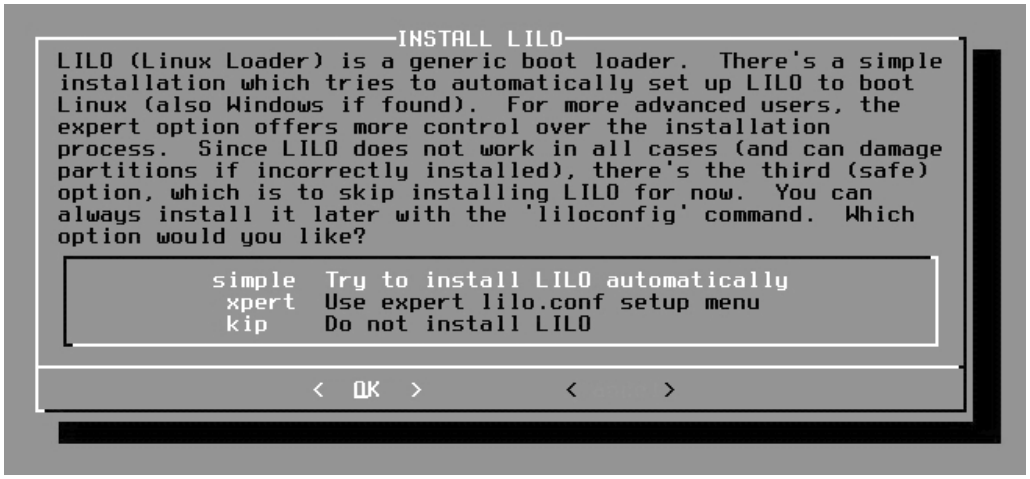

Рис. 1.33. Встановлення завантажувача LILO

На останньому етапі інсталяції необхідно сконфігурувати відео, мишу та мережу. Slackware також дозволяє користувачу обрати застосунки, що будуть автозавантажуватись (рис. 1.34).

Рис. 1.34. Меню вибору застосунків для автозапуску

Встановлення графічної оболонки доступне після налаштування локалізації в інсталяторі (рис. 1.35). Як і в дистрибутивах, що були розглянуті вище, доступно декілька різних графічних оболонок: xfce, Gnome, Mate, KDE.

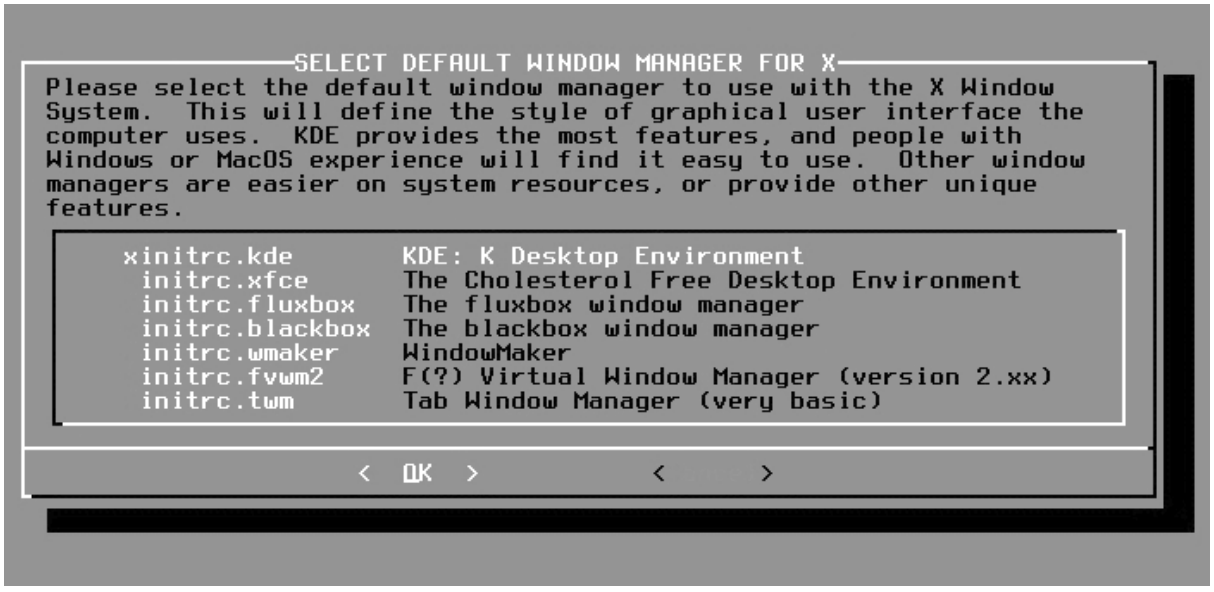

Рис. 1.35. Запрошення обрати графічну оболонку

Після коректного встановлення та налаштування необхідно перезавантажити комп’ютер, якщо воно було успішним користувач побачить завантаження Slackware (рис. 1.36).

Рис. 1.36. Запрошення LILO завантажити Slackware

1.3. Командна оболонка та інтерпретація командного рядка

Операційні системи, що побудовані на основі ядра Linux, використовують для побудови діалогу з користувачем «термінал». Історично термінал є пристроєм ведення даних, або засобом зв'язку із зовнішнім середовищем, на перших етапах розвитку комп'ютерної техніки терміналом вважався будь-який пристрій що міг надсилати байт інформації.

У сучасному світі використовують так звані «емулятори терміналу», що складаються з двох частин діалогового вікна (графічної оболонки дистрибутиву) та командної оболонки.

Командна оболонка (shell) – програма, що взаємодіє з користувачем за допомогою текстового інтерфейсу. Він називається інтерфейсом командного рядка (CLI), що дозволяє запускати програмне забезпечення та користуватися інструментами, які вбудовані в дистрибутив (пакетними менеджерами, тестовими редакторами, засобами обробки текстової інформації та ін.).

Командна оболонка інтерпретує введені користувачем команди, перетворюючи їх в інструкції операційної системи.

Усі оболонки мають схожі функції та властивості, згідно з їх основним призначенням – виконувати команди користувача і показувати результати їх виконання.

Прикладом оболонки може бути, інтерпретатор команд command операційної системи MS DOS, або оболонка bash, що входить до складу дистрибутивів ядра Linux.

Для операційних систем сімейства Linux можливе використання декількох різних оболонок, що відрізняються за властивостями і методами взаємодії з системою.

Найбільш поширеними є:

– **sh** – оболонка Bourne, класична оболонка для ОС Unix, підтримується більшістю дистрибутивів Linux;

– **bash** – оболонка Bourne Again (GNU Bourne-Again SHell), найбільш поширена на сьогодні, оболонка в середовищі ОС сімейства Linux;

– **ksh** – оболонка Korn, розроблена на основі оболонки Bourne з історією командного рядка та можливістю редагування команд;

– **csh** – оболонка C, яка використовує синтаксис популярної мови програмування C;

– **tcsh** – версія оболонки C з інтерактивним редагуванням командного рядка.

У системі може бути встановлено кілька різних оболонок, і для кожного користувача можливе використання своєї оболонки, яка запускається за замовчуванням. Налаштування оболонок зберігаються у файлі «***/etc/passwd***».

Після старту оболонки, на екран виводиться запрошення до введення команд (зазвичай у вигляді знака долара **$**, якщо робота виконується у контексті обліковим записом користувача, або фунта **#**, якщо оболонка використовується під обліковим записом привілейованого користувача **root**).

Команда – користувацьке введення у відповідь на запрошення оболонки. Команда Linux – рядок символів з назви команди і аргументів, розділених пробілами.

Команди часто супроводжуються одним або декількома параметрами, що змінюють їх поведінку, і додатковими, одним або декількома, аргументами, на які впливає команда. Тому більшість команд виглядають приблизно так:

команда – параметри аргумент

Аргументи надають команді додаткові параметри, що визначають її поведінку. У якості аргументів використовуються опції, назви файлів та каталогів.

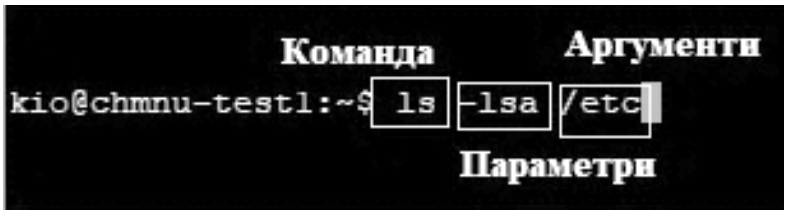

Рис. 1.37. Приклад команди з декількома параметрами

Наприклад, варіанти наступних команд ідентичні: **ls -l -d** і **ls -ld**. Команди, які є частиною оболонки, називаються вбудованими. Наприклад,

cd, **if**, **case** і т. д., вбудовані команди можуть відрізнятися для різних варіантів оболонок.

Більшість команд використовують параметри, що складаються з одного символу, якому передує дефіс, наприклад: **-l**. Але багато команд, у тому числі команди з проекту GNU, підтримують параметри з довгими назвами, що складаються з слова, якому передують два дефіса. Крім того, багато команд дозволяють обєднувати разом параметри з короткими назвами. У наступному прикладі команді **ls** передаються два параметри: параметр **l**, що вимагає використовувати довгий (long) формат виведення, і параметр **t**, що вимагає сортувати результати за часом (time) зміни:

```
$ ls -lt
```

Додамо параметр з довгою назвою --reverse, щоб змінити порядок сортування на зворотний:

```
$ ls -lt --reverse
```

Крім вбудованих команд, можливе використання програмних модулів, що представляють окремі виконувані файли, або файли скриптів, які виконуються послідовно.

Розділ 2.
ОСНОВИ РОБОТИ В ОПЕРАЦІЙНІЙ СИСТЕМІ LINUX

2.1. Файлова система Linux

На самому початку освоєння операційної системи Linux користувачам які мають досвід роботи з операційною системою Microsoft Windows може показатися досить дивним позначення файлів та каталогів у ній. Тому, перед вивченням системи Linux, спочатку необхідно розглянути як вона влаштована.

Першою відмінністю є те, що в назвах шляхів файлів і каталогів Linux не використовуються назви дисків на відміну від Windows, де вони обов'язково присутні. Наприклад, шлях **`C:\Users\Gleb\Documents\test.doc`** в ОС Windows вказує, що файл **`test.doc`** знаходиться в каталозі **`Documents`**, який у свою чергу є безпосередньо у каталозі **`Gleb`**. Каталог **`Gleb`** – у каталозі **`Users`**, який знаходиться у розділі жорсткого диска, якому присвоєна літера C (зазвичай таким є перший розділ жорсткого диска на персональному комп'ютері).

За шляхом до файлу в Windows можна завжди дізнатися, в якому фізичному розділі жорсткого диска знаходиться відповідний файл. Наприклад, якщо файл потрібно зберегти на USB-накопичувачі, то цей пристрій у системі може бути позначений буквами диска **`F`**, **`G`**, **`H`** і т. п. Після вибору відповідного накопичувача у вкладці «Мій комп'ютер» буде автоматично відкритий доступ до файлів та каталогів, що містяться в ньому. Шляхи до них завжди починатимуться літерою відповідного накопичувача. Наприклад, **`F:\test.doc`**, означає, що файл **`test.doc`** знаходить у кореневому каталозі диска, який позначається літерою **`J`**.

Фундаментальна відмінність Linux від Windows у позначенні шляхів до файлів та каталогів полягає у тому, що Linux все зберігає у єдиній структурі каталогів та називається віртуальним каталогом. Віртуальний каталог визначає шляхи до файлів, які містяться на всіх пристроях, що встановлені на персональному комп'ютері, об'єднуючи їх **у єдину структуру каталогів.**

Структура віртуального каталогу Linux складається з єдиного основного каталогу, який називається кореневим і позначається єдиним символом **`/`**. Каталоги та файли, що знаходяться у ньому, перераховуються

з урахуванням шляху до каталогу, що використовується для їх досягнення, аналогічно тому, як це відбувається в системі Windows, тільки на відміну від Windows у якості символу розділювача назв каталогів у шляху замість **** використовується **/**.

Як приклад, розглянемо шлях до файлу в системі Linux: **/home/gleb/Documents/test.doc**. Такий шлях вказує тільки на те, що файл **test.doc** знаходиться у каталозі **Documents**, який розташований у каталозі **gleb**, що міститься в каталозі **home**, який у свою чергу знаходиться у кореневому каталозі (на це вказує перший символ **/** у цьому шляху). Легко помітити, що розглянутий вище шлях не вказує, на якому фізичному диску персонального комп'ютера зберігається цей файл.

Складність, пов'язана з використанням віртуального каталогу Linux, полягає саме в тому, що в нього входять структури каталогів усіх запам'ятовуючих пристроїв на комп'ютері. Перший жорсткий диск, встановлений у персональному комп'ютері з ОС Linux, називається **кореневим диском**, який містить ядро віртуального каталогу і від якого виходять усі інші частини віртуального каталогу. На кореневому диску система Linux створює спеціальні каталоги, що називаються точками монтування.

Точками монтування є каталоги у віртуальному каталозі, за якими закріплюються додаткові запам'ятовуючі пристрої, монтування яких у віртуальному каталозі призводить до появи файлів і каталогів, що зберігаються у них, під точками монтування у каталогах монтування без урахування того, що вони фактично можуть перебувати на інших дисках.

На кореневому диску часто фізично розташовуються системні файли, тоді як файли користувача можуть зберігатись на іншому диску (рис. 2.1). Усе залежить від того, скільки розділів жорсткого диска було створено під час інсталяції ОС Linux.

У прикладі, який показаний на рис. 2.1, на персональному комп'ютері є два жорстких диски. Один пов'язаний з коренем віртуального каталогу, що позначено окремим символом **/**. Інший можуть бути змонтований в певних місцях віртуальної структури каталогів. У нашому прикладі другий жорсткий диск змонтований в місці розташування **/home**, яке є місцем, знаходження каталогів, призначених для користувачів системи.

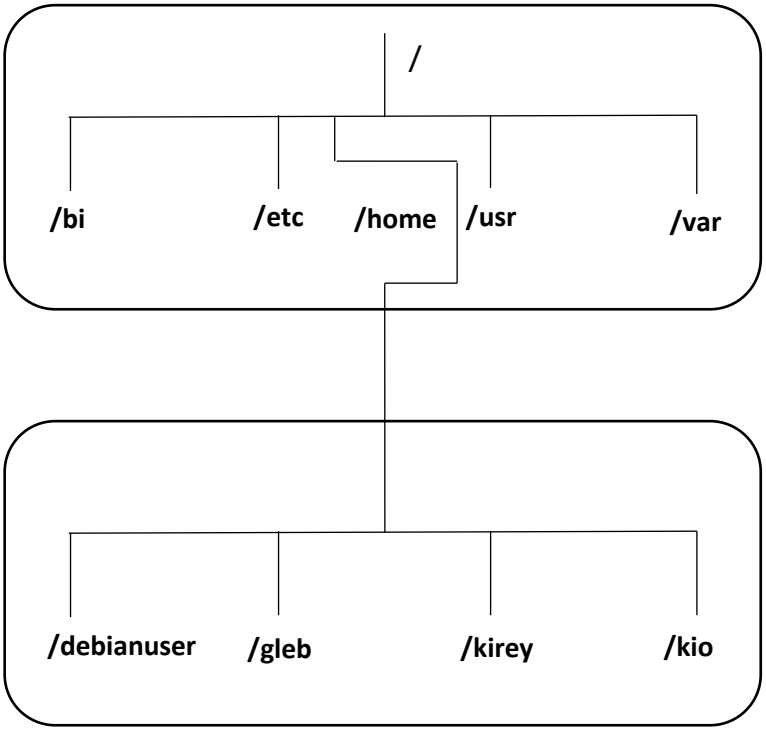

Рис. 2.1. Файлова архітектура системи Linux

Структуру файлової системи Linux було розроблено на основі файлової архітектури UNIX. На жаль, файлова архітектура UNIX за минулі роки у зв'язку з появою різновидів цієї операційної системи втратила колишню однаковість. Іноді може показатися, що не можна знайти такі дві системи UNIX або Linux, у яких застосовувалася б повністю однакова структура файлової системи. Однак передбачено кілька загальноприйнятих назв каталогів, які призначені для виконання приблизно одних і тих же функцій. У таблиці 2.1 перераховані деякі з найбільш широко застосовуваними назвами підкаталогів віртуального каталогу Linux.

Табл. 2.1. Загальноприйняті назви каталогів Linux

Каталог	Призначення
/	Кореневий каталог, у якому зазвичай не прийнято розміщувати будь-які файли
/bin	Каталог двійкових файлів, у якому зберігається велика кількість програм для користувача рівня, що відносяться до категорії утиліт GNU
/boot	Каталог, у якому зберігаються файли, необхідні для завантаження системи
/dev	Каталог, у якому Linux створює спеціальні файли пристроїв
/etc	Каталог файлів конфігурації системи
/home	Вихідний каталог, у якому Linux створює домашні каталоги користувачів

Закінчення табл. 2.1.

/lib	Каталог, у якому зберігаються файли бібліотек системи і програмних застосунків
/media	Каталог, який є загальним місцем для точок монтування, що використовуються для змінних носіїв
/mnt	У старих версіях відігравав роль каталогу **/media**. Зараз практично не використовується
/opt	Додатковий каталог, який часто служить для зберігання додаткових пакетів програм
/root	Домашній каталог користувача **root** (варто звернути увагу на те, що він розташовується окремо від домашніх каталогів інших користувачів)
/sbin	Системний каталог двійкових файлів, у якому зберігається багато адміністративних програм на рівні GNU
/tmp	Тимчасовий каталог, у якому можуть створюватися і знищуватися тимчасові робочі файли
/usr	Каталог програм, встановлених користувачем
/var	Каталог зі змінним вмістом, призначений для файлів, які часто змінюються, таких як файли журналів.

Після запуску нового запрошення командного інтерпретатора відкривається сеанс з переходом у вихідний каталог користувача, який є унікальним, та призначений для роботи тільки з певним обліковим записом користувача. Під час створення облікового запису користувача система зазвичай призначає унікальний каталог виключно для цього облікового запису.

Так у рядку-запрошенні до введення команд між символами **:** та **$** після входу певного користувача у систему буде виведено символ **~**, що є псевдонімом до повного шляху до домашнього каталогу користувача відносно його самого. При вході користувача до системи, його поточним робочим каталогом і буде домашній каталог, у цьому можна переконатися, виконавши досить просту команду **pwd** (**Print Working Directory**):

```
debianuser@debianServ2017:~$ pwd
/home/debianuser
debianuser@debianServ2017:~$
```

2.1.1. Перехід каталогами

У файловій системі Linux для переходу в інший каталог у сеансі командного інтерпретатора використовується команда зміни каталогу **cd** (**Change Directory**). Синтаксис команди **cd** є відносно нескладним:

cd <destination>

Команда **cd** може приймати єдиний аргумент – призначення, **destination**, який вказує на назву каталогу, до якого необхідно перейти. Якщо в команді **cd** не вказано призначення, здійснюється перехід

у домашній каталог користувача. Таку саму дію можна виконати, використавши у якості аргументу символ **~**:

```
cd ~
```

У цьому випадку назва каталогу, яка представлена знаком ~, що означає псевдонім домашнього каталогу користувача, але у загальному випадку може бути визначена з використанням двох різних способів:

- як абсолютний шлях до каталогу;
- як відносний шлях до каталогу.

Абсолютний шлях до файлу точно визначає, де знаходиться каталог у структурі віртуального каталогу, починаючи з кореневого каталогу і закінчуючи назвою самого каталогу, що призводить до отримання повної його назви. Для переходу в конкретне місце розташування у файловій системі з використанням абсолютного шляху до файлу, досить вказати повну назву шляху в команді **cd**:

```
debianuser@debianServ2017:~$ cd /usr
debianuser@debianServ2017:/usr$
```

Відносні шляхи до файлу дозволяють вказувати шлях до необхідного файлу відносно поточного робочого каталогу, не змушуючи починати з кореня. Позначення відносного шляху до файлу починається не з символу **/**, що вказує на кореневий каталог.

Замість цього позначення відносного шляху до файлу починається або з назви каталогу (якщо відбувається перехід до каталогу, який розташований у поточному каталозі), або зі спеціального символу, який зазначає місце розташування щодо поточного каталогу. Для цього використовуються два спеціальні символи:

- точка (**.**), яка позначає поточний каталог;
- подвійна точка (**..**), яка представляє батьківський каталог.

Наприклад, для переходу в каталог bin, який знаходиться у каталозі **/usr**, при цьому каталог **/usr** є поточний робочий каталог, достатньо виконати наступну команду:

```
debianuser@debianServ2017:/usr$ cd bin
debianuser@debianServ2017:/usr/bin$
```

Таку ж саму дію можна виконати, позначивши на початку шляху поточний робочий каталог (символ «**.**»):

```
debianuser@debianServ2017:/usr$ cd ./bin
debianuser@debianServ2017:/usr/bin$
```

Символ подвійної крапки стає надзвичайно зручним, якщо потрібно виконати перехід за ієрархією каталогів. Наприклад, якщо поточним є каталог **/usr/bin** а потрібно перейти у каталог **/usr/etc**, можна виконати наступну команду:

```
debianuser@debianServ2017:/usr/bin$ cd ../src
debianuser@debianServ2017:/usr/src$
```

Символ подвійної крапки забезпечує перехід вгору на один рівень, а потім частина відносного шляху **/src** дозволяє знову перейти на більш низький рівень, але вже в каталог **src**. Для переміщення структурою каталогів можна не один раз використовувати символ подвійної точки, а скільки буде потрібно. Наприклад, для переходу в каталог /etc з каталогу **/usr/src**, можна ввести наступне:

```
debianuser@debianServ2017:/usr/src$ cd ../../etc
debianuser@debianServ2017:/etc$
```

Одна з найбільш важливих функцій командного інтерпретатора полягає у можливості перегляду вмісту каталогів. Інструментом, який виконує цю дію, є команда **ls**, назва якої є скороченням від слова list (список). Найбільш простим застосуванням команди **ls** є її виконання без жодного ключа та аргументу. У цьому випадку команда виведе вміст поточного каталогу в алфавітному порядку.

```
debianuser@debianServ2017:/$ ls
bin   etc         initrd.img.old  lost+found  opt   run   sys  var
boot  home        lib             media       proc  sbin  tmp  vmlinuz
dev   initrd.img  lib64           mnt         root  srv   usr  vmlinuz.old
debianuser@debianServ2017:/$
```

У зазначеному прикладі показане виведення вмісту кореневого каталогу, оскільки на момент виконання команди **ls** саме кореневий каталог був поточним робочим. Якщо ж виконати команду **ls** з аргументом, то буде виведений вміст каталогу, шлях до якого (абсолютний або відносний) був зазначений у якості цього аргументу, навіть якщо користувач у поточний момент часу не знаходиться у відповідному каталозі. У наступному прикладі виводиться вміст каталогу **/var**, який наразі не є поточним робочим каталогом:

```
debianuser@debianServ2017:/$ ls /var
backups  cache  lib  local  lock  log  mail  opt  run  spool  tmp
debianuser@debianServ2017:/$
```

Крім того, у команді **ls** можна вказувати більш ніж один аргумент. У цьому випадку буде почергово виведений вміст каталогів, які вказані у якості аргументів. Наступний приклад демонструє виведення вмісту каталогів **/var** та **/sys**:

```
debianuser@debianServ2017:/$ ls /var /sys
/sys:
block  class  devices   fs          kernel  power
bus    dev    firmware  hypervisor  module

/var:
backups  cache  lib  local  lock  log  mail  opt  run  spool  tmp
debianuser@debianServ2017:/$
```

У результаті виконання можна помітити цікаву особливість, що при застосуванні більше ніж одного аргументу в команді **ls**, вона виводить вміст усіх вказаних у якості аргументів каталогів, але у зворотному порядку.

У свою чергу, якщо команда **ls** виконується без жодного ключа, вона виводить вміст каталогів у найбільш простому вигляді: в алфавітному порядку в декілька стовпчиків. Зазвичай це може бути зовсім не те, що потрібно користувачеві.

У таблиці 2.2 перерахована частина найбільш широко застосовуваних ключів, команди **ls**, за допомогою яких можна отримати необхідні дані. Досить багато однобуквених ключів мають свої аналоги серед повнослівних ключів.

Табл. 2.2. Часто застосовувані ключі команди `ls`

Однобуквений ключ	Повнослівний ключ	Опис
-a	***--all***	Виводить усі файли та каталоги, включаючи приховані, назви яких починаються з точки
-F	***--classify***	Виводить вміст каталогів, приєднуючи до записів спеціальні символи, що позначають певні типи файлів
-s	***--size***	Окрім назв файлів та каталогів, також виводить перед ними розмір відповідного файлу, або каталогу в блоках
-r	***--reverse***	Виводить вміст каталогів у зворотному порядку сортування, режим якого вказаний за допомогою відповідного ключа (за замовчуванням – за алфавітом)
-S	***--sort=size***	Виводить вміст каталогу відсортованого за розмірами файлів
-t	***--sort=time***	Виводить вміст каталогу відсортованого за часом модифікації файлів
-l		Виводить вміст каталогу з найбільш детальною інформацією (довгий формат)
-h	***--human-readable***	У сукупності з ключем **-l** виводить розмір у більш придатному вигляді, використовуючи позначення K – для кілобайтів, M – для мегабайтів і G – для гігабайтів
-i	***--inode***	У сукупності з ключем **-l** у якості першого стовпчика виводить індексний номер (так званий індексний вузол) для кожного файлу.

Тепер розглянемо найбільш часто застосовувані ключі детальніше. Ключ **-F** є так званим класифікатором, що дозволяє сформувати виведення, в якому простіше відрізнити звичайні файли від каталогів, символічних посилань та виконуваних файлів. Застосування цього ключа під час виведення вмісту каталогу додає до назви кожного елемента, якщо він не представляє звичайний файл, спеціальний символ, у якості якого може бути:

/ – для каталогу;

@ – для символічного посилання;

***** – для виконуваного файлу, який представляє собою.

У наступному прикладі представлено виведення вмісту кореневого каталогу з використанням ключа-класифікатора

```
debianuser@debianServ2017:/$ ls -F
bin/   home/            lib64/       opt/   sbin/  usr/
boot/  initrd.img@      lost+found/  proc/  srv/   var/
dev/   initrd.img.old@  media/       root/  sys/   vmlinuz@
etc/   lib/             mnt/         run/   tmp/   vmlinuz.old@
debianuser@debianServ2017:/$
```

З іншого боку, застосування команди **ls** в основній формі виводить вміст каталогу, показуючи не всі файли. У системі Linux інформація щодо конфігурації достатньо часто зберігається у прихованих файлах і каталогах, назви яких починаються з точки. Такі файли не з'являються в створюваному за замовчуванням лістингу **ls**. Для відображення прихованих файлів поряд зі звичайними файлами і каталогами потрібно використати ключ **-a**. Наступний приклад виводить вміст кореневого каталогу, включаючи приховані файли:

```
debianuser@debianServ2017:/$ ls -a
.    boot    etc         initrd.img.old  lost+found  opt   run   sys  var
..   .cache  home        lib             media       proc  sbin  tmp  vmlinuz
bin  dev     initrd.img  lib64           mnt         root  srv   usr  vmlinuz.old
debianuser@debianServ2017:/$
```

Як можна побачити, прихованим каталогом у кореневому є каталог ***.cache***. Крім того у якості прихованих каталогів у будь-якому каталозі зберігаються дескриптори, що вказують на поточний (**.**) та батьківський (**..**) каталоги. Хоча для кореневого каталогу дескриптор «**..**» практично ніякої ролі не відіграє, оскільки він не має батьківського каталогу (точніше він сам для себе і є батьківським каталогом), для всіх інших каталогів цей дескриптор має істотне значення.

Використання ключа ***-l*** з командою ***ls*** кардинально відрізняє виведення вмісту каталогів, від тих виведень, що вже були розглянуті раніше. Цей ключ дозволяє отримати так званий довгий формат лістингу, в якому наведена докладна інформація щодо кожного елементу в каталозі.

У кожному рядку лістингу в довгому форматі містяться відомості про різні елементи певного каталогу. Такий лістинг, окрім назви елемента, показує іншу корисну інформацію і являє собою інформацію у структурованому вигляді, де:

– **1-й стовпчик** – права доступу до файлів/каталогів (буде розглянуто у наступних розділах);

– **2-й стовпчик** – кількість жорстких посилань на файл;

– **3-й стовпчик** – ім'я користувача власника файлу;

– **4-й стовпчик** – назва основної групи користувачів, до якої відноситься власник файлу;

– **5-й стовпчик** – розмір файлу в байтах (при використанні ключа ***-h*** виводяться додаткові позначення);

– **6–8 стовпчики** – дата та час останньої модифікації файлу;

– **9-й стовпчик** – назва файлу.

У наступному прикладі представлено виведення вмісту кореневого каталогу в довгому форматі.

```
debianuser@debianServ2017:/$ ls -l
total 92
drwxr-xr-x   2 root root  4096 Aug 30 11:53 bin
drwxr-xr-x   3 root root  4096 Aug 30 11:54 boot
drwxr-xr-x  17 root root  3060 Sep 27 22:29 dev
drwxr-xr-x 133 root root 12288 Oct  1 11:16 etc
drwxr-xr-x 216 root root  4096 Sep  7 10:09 home
lrwxrwxrwx   1 root root    29 Aug 31  2017 initrd.img -> boot/initrd.img-4.9.0-
3-amd64
lrwxrwxrwx   1 root root    29 Aug 31  2017 initrd.img.old -> boot/initrd.img-4.
9.0-3-amd64
drwxr-xr-x  16 root root  4096 Aug 31  2017 lib
drwxr-xr-x   2 root root  4096 Aug 31  2017 lib64
drwx------   2 root root 16384 Aug 31  2017 lost+found
drwxr-xr-x   4 root root  4096 Aug 31  2017 media
drwxr-xr-x   2 root root  4096 Aug 31  2017 mnt
drwxr-xr-x   2 root root  4096 Aug 31  2017 opt
dr-xr-xr-x 167 root root     0 Oct 25  2018 proc
drwx------  18 root root  4096 Apr  3  2018 root
drwxr-xr-x  24 root root   740 Oct  3 10:34 run
drwxr-xr-x   2 root root 12288 Aug 30 11:53 sbin
drwxr-xr-x   2 root root  4096 Aug 31  2017 srv
dr-xr-xr-x  13 root root     0 Sep 28 09:16 sys
drwxrwxrwt  12 root root  4096 Oct  3 10:17 tmp
drwxr-xr-x  10 root root  4096 Aug 31  2017 usr
drwxr-xr-x  11 root root  4096 Aug 31  2017 var
lrwxrwxrwx   1 root root    26 Aug 31  2017 vmlinuz -> boot/vmlinuz-4.9.0-3-amd6
4
lrwxrwxrwx   1 root root    26 Aug 31  2017 vmlinuz.old -> boot/vmlinuz-4.9.0-3-
amd64
debianuser@debianServ2017:/$
```

Також можна звернути увагу на те, що виведення символічних посилань у довгому форматі достатньо відрізняється від виведення всіх інших елементів. Окрім назви файлу, що є символічним посиланням, також представлена його назва, на який вказує це символічне посилання.

2.1.2. Команди обробки файлів та каталогів

У командного інтерпретатора **bash** є велика кількість команд для маніпулювання файлами у файловій системі Linux. Далі будуть розглянуті команди, які застосовуються для роботи з файлами і каталогами з інтерфейсу командного рядка в усіх ситуаціях, коли потрібно здійснювати їх обробку.

Створення каталогів

Для створення каталогів призначена команда **mkdir**, яка має наступний синтаксис:

mkdir <каталог ...>

де ***каталог ...*** – одна або декілька назв каталогів, вказаних через пробіл. Усі каталоги створюються у поточному робочому каталозі.

Приклади

mkdir dir1 – створює один каталог під назвою dir1;

```
debianuser@debianServ2017:~$ ls
debianuser@debianServ2017:~$ mkdir dir1
debianuser@debianServ2017:~$ ls -il
total 4
132386 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:04 dir1
debianuser@debianServ2017:~$
```

mkdir dir2 dir3 – створює два каталоги з назвами dir2 та dir3.

```
debianuser@debianServ2017:~$ ls -il
total 12
132386 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:04 dir1
132393 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir2
132394 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir3
debianuser@debianServ2017:~$
```

Використання ключа -p призведе до створення ланцюжка каталогів, якщо вказаний відповідний шлях. Наприклад:

mkdir -p dir1/dir2 – створить каталог ***dir2***, створивши до цього і його батьківський каталог ***dir1***, який буде розміщений у поточному робочому каталозі.

Створення файлів

Час від часу виникають ситуації, у яких потрібно створити порожній файл. У подібних випадках за відсутності необхідного файлу можна використовувати команду ***touch***, що дозволить легко його створити.

```
debianuser@debianServ2017:~$ touch file1
debianuser@debianServ2017:~$ ls -il
total 12
132386 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:04 dir1
132393 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir2
132394 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir3
132397 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:11 file1
debianuser@debianServ2017:~$
```

Команда ***touch*** створює новий файл з вказаною назвою, а власником файлу призначає ім’я поточного користувача. У розглянутій команді ***ls*** використовувалися ключі ***-il***, тому перший запис в лістингу показує номер індексного вузла, присвоєний файлу. Кожен файл у файловій системі Linux має унікальний номер індексного вузла.

Насправді створення порожнього файлу командою ***touch*** не є основним призначенням цієї команди, а тільки її побічним ефектом, який

має місце, якщо у якості аргументу команди була вказана назва неіснуючого файлу. Основним призначенням команди ***touch*** є зміна часу останньої модифікації файлу. Наприклад, застосуємо цю команду ще раз, але до файлу, який до цього вже був створений:

```
debianuser@debianServ2017:~$ touch file1
debianuser@debianServ2017:~$ ls -il
total 12
132386 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:04 dir1
132393 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir2
132394 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir3
132397 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:13 file1
debianuser@debianServ2017:~$
```

Як можна помітити, під час повторного застосування команди touch для файлу file1 час його останньої зміни був оновлений, а саме збільшений на дві хвилини. Таким чином ми ніби «доторкнулися» до файлу.

Копіювання файлів

Для копіювання файлів і каталогів у Linux призначена команда ***cp***, яка означає скорочення від слова **CoPy**. Така команда повинна мати як мінімум 2 аргументи. У цьому випадку перший аргумент є джерелом копіювання, а другий – призначенням.

Розглянемо цю форму команди більш детально. Команда ***cp*** у цьому випадку буде мати наступний синтаксис:

cp <елемент1> <елемент2>,

де ***елемент1*** є джерелом копіювання і може бути файлом або каталогом, який повинен обов'язково існувати. Для копіювання каталогу необхідно застосувати ключ -r, який означає рекурсивне копіювання каталогу разом з його підкаталогами.

Натомість ***елемент2*** може бути файлом або каталогом. Система визначає, чим саме є цей аргумент та чи існує він взагалі.

Якщо елементу із зазначеною назвою не існує, то у випадку копіювання файлу система створить у поточному робочому файлі новий з відповідною до другого аргументу назвою.

Але якщо елемент з цією назвою існує і він є файлом, система перезапише його, тобто вся інформація, що зберігалася у ньому до цього, буде втрачена. У цьому випадку застосовувати команду копіювання варто досить обережно.

Якщо елемент із заданою назвою існує і він є каталогом, система скопіює елемент, вказаний у якості першого аргументу (файл або каталог) у нього.

Команда ***cp*** буде діяти зовсім іншим чином, якщо матиме три або більше аргументи. У цьому випадку команда буде мати наступний синтаксис:

cp <елемент...> <каталог>,

де ***елемент...*** – список елементів-джерел копіювання, якими можуть бути як файли, так і каталоги, але якщо серед джерел є хоча б один каталог, потрібно застосувати ключ ***-r***; ***каталог*** – каталог-призначення, який повинен обов'язково існувати, інакше команда буде виконана з помилкою.

Як і у більшості команд оболонки ***bash*** у команди ***cp*** є різноманітні ключі для вирішення багатьох різних задач, основні з яких представлені у таблиці 2.3.

Табл. 2.3. Основні ключі команди `cp`

Ключ	Опис
-i	Інтерактивний режим команди ***cp***: якщо у результаті виконання команди файл може бути перезаписаний, перед її виконанням виводиться додаткове питання для користувача перезаписати файл чи ні
-r	Копіювання каталогів разом з їх вмістом (рекурсивне копіювання)
-p	Копіювання зі збереженням атрибутів файлів (права доступу, власник, група власників і т. д.), якщо це можливо
-a	Запис файлів у архів зі збереженням їх атрибутів
-u	Режим оновлення: копіювання вихідного файлу, тільки якщо він має більш нові значення дати і часу, порівняно з цільовим файлом
-v	Застосування режиму докладного виведення з поясненням того, що відбувається.

Приклади

cp file1 file2 – копіювання файлу ***file1*** у файл ***file2***. Якщо ***file2*** існує, у ньому буде міститися перезаписана інформація, що збережена у файлі ***file1***. Якщо файл ***file2*** відсутній, то він буде створений.

```
debianuser@debianServ2017:~$ cp file1 file2
debianuser@debianServ2017:~$ ls -il
total 12
132386 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:04 dir1
132393 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir2
132394 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir3
132397 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:13 file1
177597 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:31 file2
debianuser@debianServ2017:~$
```

cp -i file1 file2 – те саме, що і вище, але якщо файл ***file2*** існує, у користувача буде запитано підтвердження перед перезаписом файлу.

```
debianuser@debianServ2017:~$ cp -i file1 file2
cp: overwrite 'file2'? y
debianuser@debianServ2017:~$ ls -il
total 12
132386 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:04 dir1
132393 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir2
132394 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir3
132397 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:13 file1
177597 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:33 file2
debianuser@debianServ2017:~$
```

cp file1 file2 dir1 – копіювання файлів ***file1*** і ***file2*** у каталог ***dir1***, який обов'язково повинен існувати.

```
debianuser@debianServ2017:~$ cp file1 file2 dir1
debianuser@debianServ2017:~$ ls -il dir1
total 0
177598 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:41 file1
177600 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:41 file2
debianuser@debianServ2017:~$
```

cp -r dir1 dir2 – копіювання каталогу ***dir1*** зі всім його вмістом у каталог ***dir2***. Якщо каталог ***dir2*** не існує, він буде створений і заповнений вмістом каталогу ***dir1***.

```
debianuser@debianServ2017:~$ cp -r dir1 dir2
debianuser@debianServ2017:~$ ls -il dir2
total 4
177601 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 12:44 dir1
debianuser@debianServ2017:~$ ls -il dir2/dir1
total 0
177602 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:44 file1
177603 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:44 file2
debianuser@debianServ2017:~$
```

Переміщення та перейменування файлів

Команда **mv** (скорочення від **MoVe**) виконує операції переміщення та перейменування файлів залежно від особливостей використання. З точки зору Linux, переміщення та перейменування є тією ж самою операцією, перейменування являє собою особливий випадок переміщення, коли один файл переміщується в інший. Команда ***mv*** використовується майже так само, як команда ***cp*** та може використовувати практично ті ж самі ключі, однак існують певні нюанси:

mv <файл1> <файл2> – перейменовує файл ***файл1*** у ***файл2***, не змінюючи індексного дескриптора файлу;

mv <каталог1> <каталог2> – якщо ***каталог2*** існує то ***каталог1*** переміщується разом зі всім його вмістом у ***каталог2***, інакше ***каталог1*** перейменовується у ***каталог2***;

mv <елемент...> <каталог> – переміщує один або більше елементів, якими можуть бути файли і каталоги, з поточного робочого каталогу в ***каталог***, який обов'язково має існувати.

Варто звернути увагу на те, що команда ***mv*** на відміну від команди ***cp*** не підтримує ключ ***-r***.

Приклади

mv file1 file3 – перейменування файлу ***file1*** у ***file3***. Якщо ***file3*** існує, він буде замінений на новий файл ***file1***, якщо ні, буде створений. У будь-якому випадку з'явиться новий файл ***file3***;

```
debianuser@debianServ2017:~$ mv file1 file3
debianuser@debianServ2017:~$ ls -il
total 12
132386 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 12:41 dir1
132393 drwxr-xr-x 3 debianuser debianuser 4096 Oct  3 12:44 dir2
132394 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir3
177597 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:33 file2
132397 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:13 file3
debianuser@debianServ2017:~$
```

mv -i file1 file3 – те саме, що і вище, але якщо файл ***file3*** існує, у користувача буде запитано підтвердження перед перезаписом файлу;

mv file2 file3 dir2 – переміщення файлів ***file2*** і ***file3*** у каталог ***dir2***, який повинен існувати;

```
debianuser@debianServ2017:~$ mv file2 file3 dir2
debianuser@debianServ2017:~$ ls -il
total 12
132386 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 12:41 dir1
132393 drwxr-xr-x 3 debianuser debianuser 4096 Oct  3 12:51 dir2
132394 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 11:08 dir3
debianuser@debianServ2017:~$ ls -il dir2
total 4
177601 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 12:44 dir1
177597 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:33 file2
132397 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:13 file3
debianuser@debianServ2017:~$
```

mv dir1 dir2 – переміщення каталогу ***dir1*** зі всім його вмістом у каталог ***dir2***. Якщо каталог ***dir2*** не існує, відбудеться перейменування каталогу ***dir1*** у ***dir2***.

```
debianuser@debianServ2017:~$ ls -il dir1
total 0
177598 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:41 file1
177600 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:41 file2
debianuser@debianServ2017:~$ mv dir1 dir3
debianuser@debianServ2017:~$ ls -il dir3
total 4
132386 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 12:41 dir1
debianuser@debianServ2017:~$ ls -il dir3/dir1
total 0
177598 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:41 file1
177600 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:41 file2
debianuser@debianServ2017:~$
```

Видалення файлів та каталогів

Для видалення файлів та каталогів використовується команда ***rm*** (скорочення від **ReMove**), яка має наступний синтаксис:

rm <елемент...>

де ***елемент*** – це один або декілька файлів чи каталогів.

Основні ключі, що підтримує команда ***rm***, наведені у таблиці 2.4.

Табл. 2.4. Параметри команди `rm`

Ключ	Опис
-r	Видалення каталогів разом з їх вмістом. Цей ключ є обов'язковим для видалення каталогів
-i	Інтерактивний режим: запитує у користувача підтвердження перед видаленням існуючого файлу, а інакше просто видаляє вказані файли та каталоги
-f	Виконання видалення у будь-якому разі. Зазначений ключ ігнорує відсутні файли і ключ ***-i***, якщо той вказаний разом з ним. Виконання команди з цим ключем у деяких випадках може бути достатньо небезпечним
-v	Виведення в процесі видалення інформаційних повідомлень.

Приклади

rm file1 – видалення файлу ***file1***;

```
debianuser@debianServ2017:~$ cd dir3/dir1
debianuser@debianServ2017:~/dir3/dir1$ ls -il
total 0
177598 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:41 file1
177600 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:41 file2
debianuser@debianServ2017:~/dir3/dir1$ rm file1
debianuser@debianServ2017:~/dir3/dir1$ ls -il
total 0
177600 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 12:41 file2
debianuser@debianServ2017:~/dir3/dir1$
```

rm -i file1 – те саме, але перед видаленням ***file1*** буде виведено запрошення підтвердження у користувача;

rm -r file2 dir1 – видалення файлу ***file2*** і каталогу ***dir1*** разом з його вмістом;

```
debianuser@debianServ2017:~/dir3$ cd ../dir2
debianuser@debianServ2017:~/dir2$ ls -il
total 4
177601 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 12:44 dir1
177597 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:33 file2
132397 -rw-r--r-- 1 debianuser debianuser    0 Oct  3 11:13 file3
debianuser@debianServ2017:~/dir2$ rm -r file2 dir1
debianuser@debianServ2017:~/dir2$ ls -il
total 0
132397 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 11:13 file3
debianuser@debianServ2017:~/dir2$
```

rm -rf file2 dir1 – те саме, але якщо файл ***file2*** або каталог ***dir1*** не існують, команда просто продовжує роботу, не виводячи ніяких повідомлень.

Також існує команда ***rmdir***, що призначена для видалення каталогів, але при цьому вони мають бути порожніми.

Групові символи

Оскільки назви файлів та каталогів всюди використовуються у командній оболонці, існують спеціальні символи, за допомогою яких можна швидко визначити цілі групи назв файлів. Такі символи називаються **груповими символами** або **символами підстановки** та дозволяють вибирати назви файлів за шаблоном.

Більшість групових символів буде розглянуто у розділі посібника щодо регулярних виразів, наразі розглянемо два основні символи, які використовуються у командах, яким у якості аргументів передаються назви файлів або каталогів. Такими символами є:

******* – символ, який заміщує від нуля до довільної їх кількості;

? – символ, який заміщує від нуля до одного будь-якого символу.

Наприклад, у домашньому каталозі користувача ***debianuser*** знаходяться каталоги, що мають назви ***dir1*** та ***dir2*** відповідно. Для того, щоб подивитися почергово вміст цих каталогів необов'язково вказувати їх у якості аргументів команди ***ls***, можна задати тільки один аргумент з груповим символом.

ls dir?

```
debianuser@debianServ2017:~$ ls dir?
dir1:
file1  file2  file3

dir2:
file4  file5  file6
debianuser@debianServ2017:~$
```

Як бачимо з наведеного прикладу, команда ***ls*** дійсно вивела вміст каталогів ***dir1*** та ***dir2***. Можна також зазначити, що у цьому випадку виконання команди ***ls dir**** призведе до такого самого результату.

Розглянемо інший приклад. Створимо у каталозі ***dir2*** файли з назвами ***file40*** та ***file400***, а потім виконаємо команду ***mv file4* ../dir1***.

```
debianuser@debianServ2017:~/dir2$ touch file40 file400
debianuser@debianServ2017:~/dir2$ ls
file4  file40  file400  file5  file6
debianuser@debianServ2017:~/dir2$ mv file4* ../dir1
debianuser@debianServ2017:~/dir2$ ls
file5  file6
debianuser@debianServ2017:~/dir2$ ls ../dir1
file1  file2  file3  file4  file40  file400
debianuser@debianServ2017:~/dir2$
```

Таким чином, з вищенаведеного прикладу можна побачити, що у каталог ***dir1*** були переміщені файли ***file4***, ***file40*** та ***file400***, які задовольняють вказаному шаблону ***file4****.

Використовуючи групові символи, можна видалити весь вміст поточного каталогу. Однак:

rm * – видалить весь вміст поточного каталогу, якщо у ньому знаходяться тільки файли і немає жодного каталогу;

rm -r * – видалить весь вміст поточного каталогу, включаючи файли і каталоги, на що вказує ключ ***-r***.

Наступний приклад видаляє весь вміст домашнього каталогу користувача ***debianuser***.

```
debianuser@debianServ2017:~$ ls -il
total 8
132393 drwxr-xr-x 2 debianuser debianuser 4096 Oct  3 13:04 dir2
132394 drwxr-xr-x 3 debianuser debianuser 4096 Oct  3 12:53 dir3
debianuser@debianServ2017:~$ rm -r *
debianuser@debianServ2017:~$ ls -il
total 0
debianuser@debianServ2017:~$
```

Однак не можна застосовувати групові символи у наступних випадках:

– у команді ***cd***, оскільки неможливо перейти одночасно у декілька каталогів одразу;

– у командах ***mkdir*** та ***touch***;

– у командах ***cd*** та ***mv*** у якості призначення (але можна у якості джерела).

Створення посилань

У файловій системі можна створити посилання на файли, які бувають двох типів: жорсткі та символічні. Обидва типи посилань можна створити командою ***ln*** (скорочення від **LiNk**). Виконання команди без ключа дозволяє створити жорстке посилання, а з ключем ***-s*** – символічне. Синтаксис команди наступний:

ln [-s] <елемент> <посилання>

Початковим способом створення посилань в операційній системі UNIX та похідних від неї операційних систем є жорсткі посилання. З точки зору файлової системи, жорстке посилання є нічим іншим як другою альтернативною назвою файлу. За замовчуванням кожен файл має одне жорстке посилання, яке і є його назвою. Під час створення жорсткого посилання у каталозі створюється додатковий запис, однак він має одне й те саме значення індексного дескриптора. Це можна чітко побачити у наступному прикладі.

```
debianuser@debianServ2017:~$ ls -il
total 0
132386 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 13:10 file1
debianuser@debianServ2017:~$ ln file1 hardlink1
debianuser@debianServ2017:~$ ls -il
total 0
132386 -rw-r--r-- 2 debianuser debianuser 0 Oct  3 13:10 file1
132386 -rw-r--r-- 2 debianuser debianuser 0 Oct  3 13:10 hardlink1
debianuser@debianServ2017:~$
```

У представленому прикладі було створено жорстке посилання ***hardlink1*** на файл ***file1***. З прикладу видно, що обидві назви вказують на одне й те саме місце у пам'яті, оскільки мають один і той самий індексний дескриптор.

Жорсткі посилання мають два важливих обмеження:

- жорстке посилання не може вказувати на файл за межами власної файлової системи, це означає, що посилання не може вказувати на файл, який знаходиться в іншому розділі диска;
- жорстке посилання не може вказувати на каталог.

Саме з метою подолання обмеження жорстких посилань був створений інший тип посилань – символічні посилання. Символічне посилання представляє собою спеціальний файл особливого типу, в якому записаний текстовий вказівник на файл або каталог. Тим самим, можна провести аналогію з ярликами в операційній системі Windows.

Наступний приклад демонструє створення символічного посилання ***simlink1*** на файл ***file1***:

```
debianuser@debianServ2017:~$ ln -s file1 simlink1
debianuser@debianServ2017:~$ ls -il
total 0
132386 -rw-r--r-- 2 debianuser debianuser 0 Oct  3 13:10 file1
132386 -rw-r--r-- 2 debianuser debianuser 0 Oct  3 13:10 hardlink1
132394 lrwxrwxrwx 1 debianuser debianuser 5 Oct  3 13:14 simlink1 -> file1
debianuser@debianServ2017:~$
```

У прикладі видно, по-перше, що індексний дескриптор символічного посилання відрізняється від індексного дескриптора файлу, що вказує на те, що це є абсолютно іншим файлом. По-друге, символічне посилання відрізняється від оригінального файлу своїм розміром. У цьому випадку воно має розмір 5 байт, хоча оригінальний файл має розмір 0 байт, оскільки є порожнім. Це пояснюється тим, що у посиланні зберігається назва файлу-оригіналу, яка і визначає розмір посилання. Так, дійсно назва ***file1*** складається з 5 символів, кожен з яких займає 1 байт. По-третє, у довгому форматі виведення змісту каталогу виведення назви символічного посилання істотно відрізняється від інших елементів файлової системи, яке має формат ***<ім'я_ посилання> -> <ім'я_файлу-оригіналу>***. До того ж символічні посилання виводяться іншим кольором (колір морської хвилі).

Відмінності між жорсткими та символічними посиланнями можна також побачити під час видалення файлів, на які вони вказують. При видаленні файлу, який має жорсткі посилання, насправді він буде мати на одну назву менше, а сам залишиться збереженим у пам'яті допоки не будуть видалені всі жорсткі посилання на нього.

Щодо видалення файлу, який має символічне посилання, то він дійсно буде видалений, але посилання залишається, проте не буде вказувати нінащо. Про такі посилання кажуть, що вони «биті». У багатьох реалізаціях команда ***ls*** виділяє биті посилання кольором, наприклад, червоним, щоб звернути на них увагу (Увага! Усі приклади представлені інверсними кольорами).

Наступний приклад демонструє все сказане вище:

```
debianuser@debianServ2017:~$ rm file1
debianuser@debianServ2017:~$ ls -il
total 0
132386 -rw-r--r-- 1 debianuser debianuser 0 Oct  3 13:10 hardlink1
132394 lrwxrwxrwx 1 debianuser debianuser 5 Oct  3 13:14 simlink1 -> file1
debianuser@debianServ2017:~$
```

Стосовно видалення файлу ***file1***, який мав як жорстке, так і символічне посилання, насправді він видалений не буде, оскільки залишатиметься жорстке посилання ***hardlink1***. Однак символічне посилання ***simlink1*** перестане працювати, оскільки воно зберігатиме назву ***file1***, а не ***hardlink1***.

2.2. Стандартні потоки даних у Linux

Існує досить багато команд, які виводять на консоль певний результат. Типовим прикладом подібних команд є команда ***ls***. Виведення на консоль можна поділити на два типи: перший – результати роботи програми, другий – повідомлення про помилки, якщо команду виконати правильно не вдалось. Наприклад, подивившись на виведення команди ***ls***, можна побачити, що іноді вона може вивести і повідомлення про помилки.

Згідно з головною ідеєю операційної системи UNIX, яка проголошує, що все являє собою файл, команди насправді виводять свої результати в спеціальний файл, який називається **стандартним виведенням** (*standard output, stdout*), а повідомлення про помилки – в спеціальний файл **стандартне виведення помилок** (*standard error*, *stderr*). За замовчуванням обидва файли пов'язані з екраном і не зберігаються на диску. Крім того, існує багато команд, які потребують вхідних даних, які вони зазвичай оброблюють. Такі команди приймають введення зі спеціального файлу, який називається **стандартним введенням** (*standard input, stdin*), та за замовчуванням прив'язаний до клавіатури.

Зазвичай виведення здійснюється на екран, а введення – з клавіатури, але механізм перенаправлення введення/виведення операційної системи UNIX дозволяє змінити цей порядок.

2.2.1. Перенаправлення стандартного виведення

Механізм перенаправлення виведення дозволяє позначити, куди здійснюватиметься стандартне виведення. Для перенаправлення стандартного виведення не на екран а в інший файл використовується оператор **>**, після чого зазначається його назва. Як приклад повідомимо командній оболонці, що вона має перенаправити результат команди ***ls*** у файл ***out.txt*** замість виведення його на екран. У прикладі використовується виведення вмісту каталогу ***/usr/bin***.

```
debianuser@debianServ2017:~$ ls /usr/bin > out.txt
debianuser@debianServ2017:~$ ls -l
total 20
-rw-r--r-- 1 debianuser debianuser 19281 Oct  3 13:29 out.txt
debianuser@debianServ2017:~$
```

Давайте дослідимо перенаправлене виведення команди. Якщо вивести вміст файлу ***out.txt*** за допомогою команди ***less***, можна побачити, що у файлі дійсно збереглися результати роботи команди ***ls***:

```
$ less out.txt
[
2to3
2to3-2.7
2to3-3.5
411toppm
aconnect
acyclic
add-apt-repository
addpart
addr2line
afm2pl
afm2tfm
aleph
allcm
allec
allneeded
alsabat
alsaloop
alsamixer
alsatplg
alsaucm
amidi
amixer
animate
animate-im6
animate-im6.q16
anytopnm
aplay
aplaymidi
appres
appstreamcli
out.txt
```

Тепер повторимо експеримент з перенаправленням, але з невеликим ускладненням: вкажемо назву неіснуючого каталогу (навмисне допустимо

помилку в шляху до каталогу – в каталозі ***/usr*** знаходиться каталог ***bin***, а не навпаки):

```
debianuser@debianServ2017:~$ ls /bin/usr > out.txt
ls: cannot access '/bin/usr': No such file or directory
debianuser@debianServ2017:~$
```

Оскільки команда з таким аргументом не може бути виконана нормальним чином, було отримано повідомлення про помилку. Як і багато типових команд UNIX, команда ***ls*** виводить повідомлення про помилки у стандартний потік виведення помилок. Оскільки ми вказали перенаправлення тільки стандартного потоку виведення, а не стандартного потоку виведення помилок, повідомлення про помилку було виведено на екран. Як перенаправити стандартний потік виведення помилок, буде показано трохи нижче, але перед цим подивимось, які зміни стались з файлом out.txt:

```
debianuser@debianServ2017:~$ ls -l
total 0
-rw-r--r-- 1 debianuser debianuser 0 Oct  3 13:36 out.txt
debianuser@debianServ2017:~$
```

Як можна побачити, файл був повністю очищений. Причиною такого неочікуваного результату є те, що простий оператор перенаправлення без попередньої йому команди (у нашому випадку в стандартний потік виведення ніякої інформації не було виведено) створить новий порожній, якщо його до цього не існувало, або очистить існуючий. Для додавання виведення в кінець існуючого файлу без його перезапису використовується інший оператор перенаправлення – ***>>***:

```
$ ls -l /usr/bin >> out.txt
```

Оператор ***>>*** додає результати виконання команди в кінець файлу. Якщо файл не існує, він буде створений, як при використанні оператора ***>***.

2.2.2. Перенаправлення стандартного виведення помилок

Перенаправлення стандартного потоку виведення помилок виконується складніше, ніж стандартне виведення. Команда може виробляти виведення у будь-який з кількох нумерованих файлових потоків, однак перші з них представляють стандартні потоки введення, виведення і виведення помилок. Командна оболонка посилається на них у якості файлових дескрипторів **0**, **1** і **2** відповідно.

Таким чином можна перенаправляти будь-який потік у файл. Так перенаправити стандартний потік виведення у файл можна було б при використанні оператора ***1>***, однак самостійно оператор ***>*** (а разом з ним і ***>>***) за замовчуванням вказують про перенаправлення саме стандартного потоку виведення. Для перенаправлення стандартного потоку помилок в один файл потрібно використати оператор ***2>***. Наприклад:

```
debianuser@debianServ2017:~$ ls -l /bin/usr 2> error.txt
debianuser@debianServ2017:~$ ls -l
total 4
-rw-r--r-- 1 debianuser debianuser 56 Oct  3 13:42 error.txt
-rw-r--r-- 1 debianuser debianuser  0 Oct  3 13:36 out.txt
debianuser@debianServ2017:~$
```

Таким чином, був створений файл error.txt, у який була записана помилка виконання команди ls /bin/usr, яка полягає у тому, що каталогу, який вказаний у якості її аргументу, не існує. Переконаємось у цьому:

$ less error.txt

```
ls: cannot access '/bin/usr': No such file or directory
error.txt (END)
```

Дійсно, у файлі ***error.txt*** міститься повідомлення про помилку.

2.2.3. Перенаправлення стандартного виведення і стандартного виведення помилок в один файл

Іноді необхідно зберегти все виведення команди в одному файлі незалежно від того, чи вивела команда нормальний результат або завершилася помилково. Для досягнення цієї мети потрібно перенаправити одразу два потоки: стандартного виведення і стандартного виведення помилок. У якості прикладу виконаємо почергово вміст трьох каталогів: ***/sys***, ***/usr*** та ***/root***. Виведення вмісту перших двох каталогів буде виконано нормально, а третього завершиться помилкою, оскільки у звичайного користувача немає прав на читання змісту домашнього каталогу користувача **root**.

```
debianuser@debianServ2017:/$ ls /sys /usr /root
ls: cannot open directory '/root': Permission denied
/sys:
block  bus  class  dev  devices  firmware  fs  hypervisor  kernel  module  power

/usr:
bin  games  include  lib  local  sbin  share  src
debianuser@debianServ2017:/$
```

Таким чином, у представленому прикладі інформація була виведена як у стандартний потік виведення, так і у стандартний потік помилок. Існує два способи перенаправлення обох потоків в один файл. Перший з них є традиційним і використовує два перенаправлення: спочатку перенаправлення стандартного потоку виведення у файл звичайним способом, а потім перенаправлення стандартного потоку виведення помилок у стандартний потік виведення. Це виглядає наступним чином:

$ ls /sys /usr /root > out.txt 2>&1

Варто зауважити, що для перенаправлення одного стандартного потоку в інший використовується знак ***&***. Тепер переглянемо зміст отриманого файлу out.txt та переконаємося, що у ньому збереглася інформація з обох потоків:

```
$ less out.txt
ls: cannot open directory '/root': Permission denied
/sys:
block
bus
class
dev
devices
firmware
fs
hypervisor
kernel
module
power

/usr:
bin
games
include
lib
local
sbin
share
src
out.txt (END)
```

Другий спосіб є більш сучасним та з'явився у більш нових версіях командного інтерпретатора **bash**. Замість двох перенаправлень у ньому використовується одне ***&>***, яке означає перенаправлення і стандартного потоку виведення і стандартного потоку виведення помилок в один і той самий файл:

```
$ ls /sys /usr /root &> out.txt
```

У Linux також існує можливість не виводити небажану інформацію, використовуючи спеціальний файл ***/dev/null***, куди можна перенаправити виведення. Файл представляє системний пристрій, що має назву **бітоприймач**, приймає будь-які вхідні дані та нічого з ними не робить. Наприклад для того, щоб приховати небажане виведення про помилки, достатньо здійснити наступне перенаправлення:

```
$ ls -l /bin/usr 2> /dev/null
```

2.2.4. Команда cat

Команда ***cat*** представляє собою типовий приклад команд, для роботи яких мають бути певні вхідні дані. За своїм визначенням ***cat*** читає вміст одного або декількох файлів, назви яких почергово передані їй у якості аргументів, та виводить їх вміст на екран, не виходячи при цьому з режиму командного рядка, на відміну від команди ***less***, що представляє програму посторінкового перегляду файлу, вихід з якої здійснюється

натисканням клавіші ***q***. Іншими словами можна сказати, що команда ***cat*** копіює вміст файлу в сандартний потік виведення.

Як приклад, виведемо вміст файлу ***out.txt***, який уже був створений та розглядався у прикладах вище.

```
debianuser@debianServ2017:~$ cat out.txt
ls: cannot open directory '/root': Permission denied
/sys:
block
bus
class
dev
devices
firmware
fs
hypervisor
kernel
module
power

/usr:
bin
games
include
lib
local
sbin
share
src
debianuser@debianServ2017:~$
```

Оскільки команда ***cat*** не виходить з режиму командного рядка, то достатньо часто використовується для виведення вмісту коротких текстових файлів. Однак якщо разом з командою ***cat*** також використовуються оператори перенаправлення, то за допомогою цієї команди можна додатково зробити деякі корисні дії.

Як було зазначено вище, команді ***cat*** для роботи необіохідні вхідні дані. Зазвичай ми забезпечуємо її ними, вказуючи назви файлів, вміст яких потрібно вивести, у якості аргументів. Однак ця команда може бути викликана і без аргументів. У цьому випадку вона починає читати дані зі стандартного потоку введення, що буде означати очікування введення користувача певних даних з клавіатури.

```
debianuser@debianServ2017:~$ cat
Hello, world!
```

Введення даних з клавіатури відбуватиметься допоки користувач не натисне комбінацію клавіш ***<CTRL>-<D>***, що означає досягнення кінця файлу на стандартному потоці введення.

```
debianuser@debianServ2017:~$ cat
Hello, world!
Hello, world!
debianuser@debianServ2017:~$
```

У результаті команда ***cat*** продублює виведення рядків, які до того ввів користувач. Такий результат легко пояснити, оскільки команда спрацювала так, як було потрібно: вивела вміст файлу на стандартний потік виведення, тільки у якості файлу у цьому випадку виступав стандартний потік введення. Утім, такий спосіб виконання команди ***cat*** є досить безглуздим, оскільки ніякої практичної цінності її результат не представляє. Однак все змінюється, якщо використовувати її разом з операторами перенаправлення виведення.

Розглянемо наступний приклад. Виконаємо команду ***cat*** без аргументів, але з оператором перенаправлення стандартного потоку виведення у файл.

```
debianuser@debianServ2017:~$ cat > file1
Hello, world!
debianuser@debianServ2017:~$
```

У цьому випадку команда не має аргументів, тому за вхідними даними знову звертається до стандартного потоку введення, тобто до клавіатури. Однак введені користувачем рядки не виводить, а записує їх у файл під назвою ***file1*** за допомогою оператора ***>***. Так само можна було б використати оператор перенаправлення ***>>***, якщо потрібно було б записати введені рядки в кінець файлу без втрати інформації, що можливо збереглася у відповідному файлі до цього. Щоб побачити результат, виконаємо команду ***cat*** звичайним способом, передавши їй у якості аргументу ***file1***:

```
debianuser@debianServ2017:~$ cat file1
Hello, world!
debianuser@debianServ2017:~$
```

Виконання команди ***cat*** у розглянутий вище спосіб представляє заповнення файлів даними «на льоту», не використовуючи жодного текстового редактора.

Розглянемо наступний приклад виконання команди ***cat***:

```
cat file1 > file2
```

У цьому випадку вхідні дані поступають до команди «нормальним» чином, оскільки у якості аргументу передається назва файлу. Якщо б не відбулося перенаправлення стандартного потоку виведення, вміст файлу був би виведений на екран, однак перенаправлення записує його вміст в інший файл, створюючи його копію. Таким чином, виконання команди cat описаним вище способом є абсолютною аналогією виконання команди:

```
cp file1 file2
```

У тому, що створена копія початкового файлу, можна переконатися, почергово розглянувши вміст першого та другого файлів:

```
debianuser@debianServ2017:~$ ls
file1  file2
debianuser@debianServ2017:~$ cat file1
Hello, world!
debianuser@debianServ2017:~$ cat file2
Hello, world!
debianuser@debianServ2017:~$
```

Також команда ***cat*** може мати більше ніж один аргумент. У цьому випадку команда почергово виведе вміст кожного з файлів, які вказані у якості аргументів. Однак при використанні операторів перенаправлення вміст усіх файлів буде записаний в інший файл, який представлятиме результат їх об'єднання:

```
cat file1 file2 > file3
```

У наступному прикладі спочатку змінимо зміст ***file2***, щоб він не був повною копією ***file1***, а потім об'єднаємо вміст даних файлів у ***file3*** та переглянемо його:

```
debianuser@debianServ2017:~$ cat > file2
Second line
debianuser@debianServ2017:~$ cat file1 file2 > file3
debianuser@debianServ2017:~$ ls
file1  file2  file3
debianuser@debianServ2017:~$ cat file3
Hello, world!
Second line
debianuser@debianServ2017:~$
```

Тепер, коли ми знаємо, що команда **cat** може приймати вхідні дані не тільки з файлів, назви яких зазначені у якості аргументів, але й зі стандартного потоку введення, виконаємо перенаправлення стандартного потоку введення шляхом використання оператора **<**:

```
debianuser@debianServ2017:~$ cat < file1
Hello, world!
debianuser@debianServ2017:~$
```

Використовуючи оператор перенаправлення **<**, ми змінили джерело даних для стандартного потоку введення на файл ***file1***. Звернемо увагу, що був отриманий такий самий результат як і під час виклику команди ***cat*** з аргументом. Такий описаний спосіб немає переваг порівняно з передачею простого аргументу, але він демонструє, що файли можна використовувати у якості джерела даних для стандартного потоку введення. Існують інші команди, які у перенапраленні стандартного потоку введення знаходять краще застосування.

2.2.5. Конвеєри та фільтри

Можливості команди зчитувати дані зі стандартного потоку введення і виводити результати у стандартний потік виведення у UNIX також використовуються в одному з механізмів об'єднання команд, який

називається **конвеєром**. Цей механізм можна впізнати за наявністю оператора **|** між двома командами:

<команда1> | <команда2>

Механізм роботи конвеєра полягає у перенаправлені результату виведення попередньої команди (***команда1***) у якості вхідних даних наступної (***команда2***). Таким способом можна сформувати цілий ланцюг команд, які передають результати своєї дії одна одній.

Зазвичай у конвеєрах використовують команди, які приймають вхідні дані, певним чином їх обробляють та виводять результат. Досить часто подібні команди називають **фільтрами**. Розглянемо деякі найбільш часто застосовувані фільтри. Однак детально ці команди будуть розглянуті у наступних розділах.

Команда sort

Як можна зрозуміти з назви, команда ***sort*** виконує сортування вхідних даних (за замовчуванням в алфавітному порядку). Подібно до команди ***cat***, у звичайному своєму застосуванні вона має аргумент або декілька, що представляють назви файлів, дані яких потрібно відсортувати, та виводить результат у стандартний потік виведення. Ті, хто тільки починає вивчення Linux зазначають, що при її використанні результат сортування зберігається, але це не так: команда ***sort*** виводить результат тільки у стандартний потік виведення, а початковий файл при цьому залишається таким, яким був. Власне ніщо не заважає перенаправити виведення команди ***sort*** у файл, однак і тут є нюанс: не можна перенаправляти вхідний файл у самого себе.

sort file1 > file1 **Неправильно!!!**

У цьому випадку вміст файлу буде очищений. Для коректного результату варто виконати перенаправлення результату в інший файл:

sort file1 > file2 **Правильно!**

Також це стосується і всіх інших команд-фільтрів.

Необов’язково вказувати команді вхідні дані у якості аргументів. За їх відсутності подібно команді ***cat***, команда ***sort*** почне приймати їх зі стандартного потоку виведення. Також цій команді можна передати вхідні дані конвеєром.

У якості прикладу виведемо назви всіх виконуваних файлів. Такі файли зберігаються у каталогах ***/bin*** та ***/usr/bin***.

$ ls /bin /usr/bin | sort | less

```
[
2to3
2to3-2.7
2to3-3.5
411toppm
aconnect
acyclic
add-apt-repository
addpart
addr2line
afm2pl
afm2tfm
aleph
allcm
allec
allneeded
alsabat
alsaloop
alsamixer
alsatplg
alsaucm
amidi
amixer
animate
animate-im6
animate-im6.q16
anytopnm
aplay
aplaymidi
appres
:
```

Припустимо, що ми хочемо вивести їх в алфавітному порядку, однак результат команди ***ls*** у цьому випадку складатиметься з двох відсортованих списків, по одному для кожного каталогу. Додавши в конвеєр команду ***sort*** можна отримати єдиний відсортований список. Наприкінці для подальшого перегляду в посторінковому режимі (список може виявитися доволі великим, щоб розміститися на екрані) виконаємо команду ***less***, передавши їй конвеєром результат команди ***sort***.

Команда uniq

Команда ***uniq*** призначена для пошуку та видалення повторюваних рядків. Зазвичай вона використовується разом з командою ***sort***, оскільки у якості вхідних даних приймає відсортований список даних з файлу, чи зі стандартного введення, або переданий конвеєром.

Команда ***uniq*** за замовчуванням виводить тільки унікальні рядки, видаляючи дублікати. Тому для гарантії відсутності повторень у списку виконуваних файлів з минулого прикладу, додамо ***uniq*** у конвеєр:

```
$ ls /bin /usr/bin | sort | uniq | less
```

Для того, щоб навпаки дізнатися, які назви виконуваних файлів є у каталогах /bin, та /usr/bin, виконаємо команду ***uniq*** з ключем ***-d***, який виводить список продубльованих рядків, але кожен рядок виводимо один раз:

```
$ ls /bin/usr/bin | sort | uniq -d | less
chacl
dumpkeys
getfacl
less
lessecho
lessfile
lesskey
lesspipe
loadkeys
setfacl
touch
which
(END)
```

Команда wc

Команда **wc** (**Word Count** – лічильник слів) є спеціальним фільтром, який приймає вхідні дані та підраховує кількість рядків, слів і символів у файлах. Наприклад, виведемо відповідну статистичну інформацію щодо даних у файлі ***out.txt***:

```
debianuser@debianServ2017:~$ wc out.txt
 23  28 180 out.txt
debianuser@debianServ2017:~$
```

У нашому прикладі було виведено три числа: кількість рядків, слів і символів (що також відповідають кількості байтів) у файлі ***out.txt***.

Відповідно до інших фільтрів, команда ***wc*** може викликатися без аргументів, приймаючи одночасно дані з клавіатури, або вхідні, які можуть бути передані їй конвеєром. Можна обмежити виведення команди за допомогою наступних ключів:

- -l – вивести тільки кількість рядків;
- -w – вивести тільки кількість слів;
- -c – вивести тільки кількість символів.

Наприклад, отримаємо кількість назв виконуваних файлів з відсортованого списку, що був отриманий у попередніх прикладах:

```
$ ls /bin/usr /bin | sort | uniq | wc -l
2728
```

Команда grep

Команда **grep** напевно є найбільш потужним фільтром, що використовується для пошуку рядків за шаблоном. Вона має наступний синтаксис:

```
grep шаблон [файл ...]
```

Обов'язковим аргументом для команди **grep** є тільки шаблон, усі інші аргументи представляють назви файлів, у вмісті яких потрібно шукати відповідні рядки і є необов'язковими (можна передати вхідні дані іншим чином). Коли команда **grep** знаходить у файлі збіг з шаблоном, вона виводить рядок зі знайденим збігом.

Для прикладу, серед назв виконуваних файлів знайдемо такі, які тим чи іншим чином пов'язані з архівацією файлів. Для простоти прикладу припустимо, що у назвах таких команд існує підрядок ***zip*** (насправді це не зовсім правильно). Зробити це можна наступним чином:

```
debianuser@debianServ2017:~$ ls /usr/bin /bin | sort | uniq | grep zip
bunzip2
bzip2
bzip2recover
funzip
gpg-zip
gunzip
gzip
preunzip
prezip
prezip-bin
unzip
unzipsfx
zip
zipcloak
zipdetails
zipgrep
zipinfo
zipnote
zipsplit
debianuser@debianServ2017:~$
```

Команда **grep** має декілька достатньо зручних ключів:

-i – ігнорувати регістр символів у процесі пошуку (за замовчуванням пошук виконується з урахуванням регістра символів);

-v – навпаки виводить лише рядки, де збіг з шаблоном не знайдений.

Команди head та tail

Наприкінці розглянемо дві схожі команди, які виводять перші та останні рядки тобто – ***head*** та ***tail***. За замовчуванням обидві команди виводять до 10 рядків, однак їх кількість можна змінити, використавши разом з відповідними командами ключ ***-n***.

Для прикладу виведемо перші та останні 5 рядків файлу ***out.txt***:

```
debianuser@debianServ2017:~$ head -n 5 out.txt
ls: cannot open directory '/root': Permission denied
/sys:
block
bus
class
debianuser@debianServ2017:~$ tail -n 5 out.txt
lib
local
sbin
share
src
debianuser@debianServ2017:~$
```

Також команди ***head*** та ***tail*** можна використовувати в конвеєрах, що й демонструє наступний приклад:

```
debianuser@debianServ2017:~$ last | head -n 5
debianus pts/0        192.168.96.112   Fri Oct  5 20:50   still logged in
kirey    pts/1        192.168.101.34   Fri Oct  5 12:43 - 13:33  (00:49)
debianus pts/0        192.168.96.216   Fri Oct  5 12:38 - 13:54  (01:16)
debianus pts/2        192.168.100.219  Fri Oct  5 10:45 - 11:51  (01:05)
1610413  pts/4        192.168.99.111   Fri Oct  5 10:42 - 11:48  (01:06)
debianuser@debianServ2017:~$
```

У прикладі була виконана команда **last**, що відображає історію роботи у системі, однак за замовчуванням її результат представив би досить великий список, тому ми вивели тільки 5 найбільш свіжих записів, які у списку знаходяться на самому початку.

2.3. Система керування пакетами

Процес встановлення програмного забезпечення в операційній системі на ядрі Linux дуже відрізняється від процесу встановлення в операційній системі Windows. Перш ніж розпочати встановлення будь-якого програмного забезпечення необхідно пам’ятати, що програмне забезпечення міститься в пакетах.

Пакет – це тип файлів, що використовується системами контролю версій, він містить код програмного продукту або є бінарним (виконуючим) файлом.

Більшість пакетів мають залежності – додаткові пакети, що необхідні для коректної роботи програмного забезпечення, яке міститься в пакеті, що встановлюється. Більшість дистрибутивів мають ***Систему керування пакетами***.

Система керування пакетами – набір програмного забезпечення, що дозволяє керувати встановленням програмного забезпечення: завантаженням необхідних пакетів, компілюванням коду, видаленням пакетів, завантаженням залежностей та їх контроль (у деяких випадках можливий конфлікт залежностей: наприклад у випадку наявності встановленої новішої версії пакету).

Джерело пакетів називають репозиторієм. Для підвищення швидкості завантаження часто репозиторії розміщують на різних регіональних серверах. У більшості випадків існує також розподіл репозиторіїв за типом ліцензії пакетів, наприклад існує ***non-free***, що містить пакети, які не відповідають умовам договору ***DFSG*** (***Debian Free Software Guidelines***). Кожен дистрибутив має власну систему керування ними.

2.3.1 Advanced Packaging Tool

Debian – цей дистрибутив у якості системи керування пакетами використовує ***Advanced Packaging Tool***(***apt***). Кожна система керування пакетами використовує власний тип, у випадку apt – це ****.deb***.

Конфігураційні файли цієї системи зберігаються в директорії ***/etc/apt***. Робота ***apt*** починається з редагування списку репозиторіїв, що міститься у файлі ***/etc/apt/sources.list***.

```
# deb cdrom:[Debian GNU/Linux 9.1.0 _Stretch_ - Official i386 NETINST 20170722-$

#deb cdrom:[Debian GNU/Linux 9.1.0 _Stretch_ - Official i386 NETINST 20170722-1$

deb http://ftp.ua.debian.org/debian/ stretch main
deb-src http://ftp.ua.debian.org/debian/ stretch main

deb http://security.debian.org/debian-security stretch/updates main
deb-src http://security.debian.org/debian-security stretch/updates main

# stretch-updates, previously known as 'volatile'
deb http://ftp.ua.debian.org/debian/ stretch-updates main
deb-src http://ftp.ua.debian.org/debian/ stretch-updates main
```

Варто пам’ятати, що ***Advanced Packaging Tool*** призначений для роботи з репозиторіями, а встановлення відбувається утилітою **dpkg**. Але встановлення пакетів без підключення до репозиторію складніше, користувачеві необхідно слідкувати за залежностями, сумісністю пакету з його дистрибутивом.

Використовуючи ***Advanced Packaging Tool*** користувачу потрібно ввести назву програмного продукту, який він хоче встановити і система сама обере необхідну версію та завантажить залежності. Наприклад, ПЗ для отримання показань з датчиків температури – ***lm-sensors*** встановлюється введенням у консоль команди ***apt-get install lm-sensors***.

```
root@linuxtest:~# apt-get install lm-sensors
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  fancontrol read-edid i2c-tools
The following NEW packages will be installed:
  lm-sensors
0 upgraded, 1 newly installed, 0 to remove and 102 not upgraded.
Need to get 112 kB of archives.
After this operation, 383 kB of additional disk space will be used.
Get:1 http://ftp.ua.debian.org/debian stretch/main i386 lm-sensors i386 1:3.4.0-
4 [112 kB]
Fetched 112 kB in 0s (1,756 kB/s)
Selecting previously unselected package lm-sensors.
(Reading database ... 141682 files and directories currently installed.)
Preparing to unpack .../lm-sensors_1%3a3.4.0-4_i386.deb ...
Unpacking lm-sensors (1:3.4.0-4) ...
Setting up lm-sensors (1:3.4.0-4) ...
Created symlink /etc/systemd/system/multi-user.target.wants/lm-sensors.service →
 /lib/systemd/system/lm-sensors.service.
Processing triggers for systemd (232-25+deb9u1) ...
Processing triggers for man-db (2.7.6.1-2) ...
```

Система також дозволяє видаляти пакети, існує декілька варіантів їх видалення: залишивши налаштування ***apt-get remove lm-sensors*** та ***apt-get purge lm-sensors***.

```
root@linuxtest:~# apt-get purge lm-sensors
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  lm-sensors*
0 upgraded, 0 newly installed, 1 to remove and 102 not upgraded.
After this operation, 383 kB disk space will be freed.
Do you want to continue? [Y/n] y
(Reading database ... 141717 files and directories currently installed.)
Removing lm-sensors (1:3.4.0-4) ...
Processing triggers for man-db (2.7.6.1-2) ...
(Reading database ... 141682 files and directories currently installed.)
Purging configuration files for lm-sensors (1:3.4.0-4) ...
Processing triggers for systemd (232-25+deb9u1) ...
root@linuxtest:~# _
```

Поновлення інформації стосовно наявних пакетів їх версій та залежностей можливе командою ***apt-get update***. Оновлення всіх пакетів репозиторію відбувається за допомогою команди ***apt-get upgrade***.

```
root@linuxtest:~# apt-get update
Ign:1 http://ftp.ua.debian.org/debian stretch InRelease
Hit:2 http://ftp.ua.debian.org/debian stretch-updates InRelease
Hit:3 http://ftp.ua.debian.org/debian stretch Release
Hit:4 http://security.debian.org/debian-security stretch/updates InRelease
Reading package lists... Done
root@linuxtest:~# _
```

Майже усі дистрибутиви, що побудовані на ядрі Linux, вимагають **root** – прав для роботи з системою керування ними.

2.3.2. Yellow dog Updater, Modified

Red Hat має іншу систему керування пакетами: ***Yellow dog Updater, Modified*** (***YUM***). Як і попередня система контролю версій

YUM використовує інтерфейс командного рядка для роботи. Тип пакетів що використовуються ****.rmp***.

Джерела репозиторіїв зберігаються в директорії ***/etc/yum.repos.d***, опис кожного репозиторію представлений файлом ****.repo***. Оновлення інформації щодо пакетів відбувається за допомогою команди **yum *check-update***, яка перевіряє наявність оновлень усіх пакетів та повертає користувачу список доступних оновлень.

```
usbutils.x86_64                                 009-1.fc26                  updates
util-linux.x86_64                               2.30.2-1.fc26               updates
vim-minimal.x86_64                              2:8.0.1428-3.fc26           updates
vte-profile.x86_64                              0.48.4-1.fc26               updates
vte291.x86_64                                   0.48.4-1.fc26               updates
webkitgtk4.x86_64                               2.18.5-1.fc26               updates
webkitgtk4-jsc.x86_64                           2.18.5-1.fc26               updates
webkitgtk4-plugin-process-gtk2.x86_64           2.18.5-1.fc26               updates
wget.x86_64                                     1.19.2-2.fc26               updates
wpa_supplicant.x86_64                           1:2.6-12.fc26               updates
xdg-desktop-portal.x86_64                       0.9-1.fc26                  updates
xdg-desktop-portal-gtk.x86_64                   0.9-1.fc26                  updates
xen-libs.x86_64                                 4.8.2-9.fc26                updates
xen-licenses.x86_64                             4.8.2-9.fc26                updates
xkeyboard-config.noarch                         2.21-3.fc26                 updates
xmlrpc-c.x86_64                                 1.49.02-2.fc26              updates
xmlrpc-c-client.x86_64                          1.49.02-2.fc26              updates
xorg-x11-drv-libinput.x86_64                    0.26.0-1.fc26               updates
xorg-x11-utils.x86_64                           7.5-23.fc26                 updates
Obsoleting Packages
julietaula-montserrat-fonts.noarch              1:7.200-1.fc26              updates
    julietaula-montserrat-fonts.noarch          1:6.002-2.fc26              @System
kernel-headers.x86_64                           4.14.14-200.fc26            updates
    kernel-headers.x86_64                       4.11.8-300.fc26             @System
[root@localhost ~]#
```

Система керування пакетами **yum** встановлює їх командою ***yum install [ім'я пакету]***: наприклад продукт lm-sensors можна встановити yum install lm_sensors.

```
[root@localhost ~]# yum install lm_sensors
Last metadata expiration check: 0:11:01 ago on Fri 02 Feb 2018 02:23:14 PM EET.
Dependencies resolved.
================================================================================
 Package                Arch          Version                Repository     Size
================================================================================
Installing:
 lm_sensors             x86_64        3.4.0-6.fc26           fedora        144 k
Installing dependencies:
 lm_sensors-libs        x86_64        3.4.0-6.fc26           fedora         45 k

Transaction Summary
================================================================================
Install  2 Packages

Total download size: 188 k
Installed size: 478 k
Is this ok [y/N]: y
Downloading Packages:
(1/2): lm_sensors-libs-3.4.0-6.fc26.x86_64.rpm   89 kB/s |  45 kB     00:00
(2/2): lm_sensors-3.4.0-6.fc26.x86_64.rpm       280 kB/s | 144 kB     00:00
--------------------------------------------------------------------------------
Total                                           123 kB/s | 188 kB     00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : lm_sensors-libs-3.4.0-6.fc26.x86_64                    1/2
  Running scriptlet: lm_sensors-libs-3.4.0-6.fc26.x86_64                    1/2
  Installing       : lm_sensors-3.4.0-6.fc26.x86_64                         2/2
  Running scriptlet: lm_sensors-3.4.0-6.fc26.x86_64                         2/2
  Verifying        : lm_sensors-3.4.0-6.fc26.x86_64                         1/2
  Verifying        : lm_sensors-libs-3.4.0-6.fc26.x86_64                    2/2

Installed:
  lm_sensors.x86_64 3.4.0-6.fc26       lm_sensors-libs.x86_64 3.4.0-6.fc26

```

Видалити пакет можливо командою ***yum remove [ім'я пакету]:*** наприклад продукт lm-sensors видаляється ***yum remove lm_sensors***. На відміну від ***Advanced Packaging Tool*** видалити конфігурацію автоматично неможливо.

```
[root@localhost ~]# yum remove lm_sensors
Dependencies resolved.
================================================================================
 Package                Arch          Version                Repository      Size
================================================================================
Removing:
 lm_sensors             x86_64        3.4.0-6.fc26           @fedora        418 k
Removing unused dependencies:
 lm_sensors-libs        x86_64        3.4.0-6.fc26           @fedora         60 k

Transaction Summary
================================================================================
Remove  2 Packages

Freed space: 478 k
Is this ok [y/N]: Y_
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Running scriptlet: None                                                  1/1
  Preparing        :                                                       1/1
  Running scriptlet: lm_sensors-3.4.0-6.fc26.x86_64                        1/2
  Erasing          : lm_sensors-3.4.0-6.fc26.x86_64                        1/2
  Running scriptlet: lm_sensors-3.4.0-6.fc26.x86_64                        1/2
  Erasing          : lm_sensors-libs-3.4.0-6.fc26.x86_64                   2/2
  Running scriptlet: lm_sensors-libs-3.4.0-6.fc26.x86_64                   2/2
  Verifying        : lm_sensors-3.4.0-6.fc26.x86_64                        1/2
  Verifying        : lm_sensors-libs-3.4.0-6.fc26.x86_64                   2/2

Removed:
  lm_sensors.x86_64 3.4.0-6.fc26       lm_sensors-libs.x86_64 3.4.0-6.fc26

Complete!
```

2.3.3. Portage

Gentoo у якості системи керування пакетами використовує **Portage**, що за принципом роботи схожий на систему ***Port,*** ОС ***FreeBSD***: пакет у такій системі представляє набір файлів для автоматизації процесу компілювання бінарних файлів з завантаженого коду (термін «port» походить від portability – портативність).

Використання системи ***Portage*** можливе через утиліту ***EMERGE***, що дозволяє встановлювати, видаляти та оновлювати пакетами шляхом завантаження та компіляції коду програми. Перед початком встановлення користувач має оновити дані **Portage,** для цього потрібно ввести команду ***emerge --sync***, на консолі з'явиться перелік пакетів даних якими вони оновлюються.

```
metadata/md5-cache/dev-texlive/texlive-mathextra-2013
metadata/md5-cache/dev-texlive/texlive-mathextra-2014
metadata/md5-cache/dev-texlive/texlive-mathextra-2015
metadata/md5-cache/dev-texlive/texlive-mathscience-2016
metadata/md5-cache/dev-texlive/texlive-mathscience-2017
metadata/md5-cache/dev-texlive/texlive-metapost-2015
metadata/md5-cache/dev-texlive/texlive-metapost-2016
metadata/md5-cache/dev-texlive/texlive-metapost-2017
metadata/md5-cache/dev-texlive/texlive-music-2015
metadata/md5-cache/dev-texlive/texlive-music-2016
metadata/md5-cache/dev-texlive/texlive-music-2017
metadata/md5-cache/dev-texlive/texlive-omega-2015
metadata/md5-cache/dev-texlive/texlive-omega-2016
metadata/md5-cache/dev-texlive/texlive-pictures-2015-r2
metadata/md5-cache/dev-texlive/texlive-pictures-2016
metadata/md5-cache/dev-texlive/texlive-pictures-2017
metadata/md5-cache/dev-texlive/texlive-plainextra-2015
metadata/md5-cache/dev-texlive/texlive-plainextra-2016
metadata/md5-cache/dev-texlive/texlive-plaingeneric-2017
metadata/md5-cache/dev-texlive/texlive-pstricks-2015
metadata/md5-cache/dev-texlive/texlive-pstricks-2016
metadata/md5-cache/dev-texlive/texlive-pstricks-2017
metadata/md5-cache/dev-texlive/texlive-publishers-2015
metadata/md5-cache/dev-texlive/texlive-publishers-2016
```

Пакети, що завантажені ***Portage*** являють собою програмний код та дані для автоматичного компілювання, а сам процес встановлення – компіляція. Отже, для його встановлення необхідно ввести команду ***emerge --ask sys-apps/lm_sensors*** (ключ ***-ask*** дозволяє бачити результат встановлення до його початку: список пакетів що будуть встановлені та їх сумарний розмір).

```
localhost ~ # emerge --ask sys-apps/lm_sensors

!!! Your current profile is deprecated and not supported anymore.
!!! Use eselect profile to update your profile.
!!! Please upgrade to the following profile if possible:

        default/linux/amd64/17.0/desktop/plasma/systemd

You may use the following command to upgrade:

        eselect profile set default/linux/amd64/17.0/desktop/plasma/systemd


 * IMPORTANT: 16 news items need reading for repository 'gentoo'.
 * Use eselect news read to view new items.


These are the packages that would be merged, in order:

Calculating dependencies... done!
[ebuild  N     ] sys-apps/lm_sensors-3.4.0_p20170901  USE="-contrib -sensord -st
atic-libs" ABI_X86="(64) -32 (-x32)"

Would you like to merge these packages? [Yes/No] _
```

Видалення пакетів у дистрибутиві ***Gentoo*** відбувається тим самим інструментом, що й встановлення ***emerge --unmerge sys-apps/lm_sensors***. Під час видалення виводиться детальна інформація про пакет: версія, розмір, та надається 5 секунд для відміни видалення.

```
 * This action can remove important packages! In order to be safer, use
 * `emerge -pv --depclean <atom>` to check for reverse dependencies before
 * removing packages.

 sys-apps/lm_sensors
    selected: 3.4.0_p20170901
   protected: none
     omitted: none

All selected packages: =sys-apps/lm_sensors-3.4.0_p20170901

>>> 'Selected' packages are slated for removal.
>>> 'Protected' and 'omitted' packages will not be removed.

>>> Waiting 5 seconds before starting...
>>> (Control-C to abort)...
>>> Unmerging in: 5 4 3 2 1
>>> Unmerging (1 of 1) sys-apps/lm_sensors-3.4.0_p20170901...

 * GNU info directory index is up-to-date.

 * IMPORTANT: 16 news items need reading for repository 'gentoo'.
 * Use eselect news read to view new items.
```

Slackware на відміну від попередніх дистрибутивів не підтримує роботу з віддаленим репозиторієм. За встановлення та видалення пакетів відповідають утиліти: ***installpkg, removepkg*** та ***upgradepkg*** або ***pkgtool***. Отже, встановлення пакету починається з його завантаження за допомогою утиліти ***wget***: «***wget http://slackware.cs.utah.edu/pub/slackware/slackware64-14.2 /slackware64/ap/lm_sensors-3.4.0-x86_64-1.txz***», ця утиліта завантажує вказаний файл (у цьому випадку пакет ***lm-sensors***) до поточної директорії.

```
root@slak:~/lm# wget http://slackware.cs.utah.edu/pub/slackware/slackware64-14.2

        /slackware64/ap/lm_sensors-3.4.0-x86_64-1.txz
--2018-02-03 13:35:08--  http://slackware.cs.utah.edu/pub/slackware/slackware64-
14.2
Resolving slackware.cs.utah.edu (slackware.cs.utah.edu)... 155.98.64.87
Connecting to slackware.cs.utah.edu (slackware.cs.utah.edu)|155.98.64.87|:80...
connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://slackware.cs.utah.edu/pub/slackware/slackware64-14.2/ [followin
g]
--2018-02-03 13:35:08--  http://slackware.cs.utah.edu/pub/slackware/slackware64-
14.2/
Reusing existing connection to slackware.cs.utah.edu:80.
HTTP request sent, awaiting response... 200 OK
Length: 6980 (6.8K) [text/html]
Saving to: 'slackware64-14.2.2'

slackware64-14.2.2  100%[==================>]   6.82K  36.6KB/s    in 0.2s

2018-02-03 13:35:09 (36.6 KB/s) - 'slackware64-14.2.2' saved [6980/6980]

/slackware64/ap/lm_sensors-3.4.0-x86_64-1.txz: Scheme missing.
FINISHED --2018-02-03 13:35:09--
Total wall clock time: 0.8s
Downloaded: 1 files, 6.8K in 0.2s (36.6 KB/s)
```

Після завантаження утилітою ***installpkg*** необхідно розпочати встановлення пакету до системи: ***installpkg lm_sensors-3.4.0-x86_64-1.txz***.

```
root@slak:~/lm# installpkg lm_sensors-3.4.0-x86_64-1.txz
Verifying package lm_sensors-3.4.0-x86_64-1.txz.
Installing package lm_sensors-3.4.0-x86_64-1.txz:
PACKAGE DESCRIPTION:
# lm_sensors (hardware monitoring package)
#
# lm_sensors provides tools for monitoring the temperatures, voltages,
# and fans of Linux systems with hardware monitoring devices.  Included
# are text-based tools for sensor reporting, and a library for sensors
# access called libsensors.  It also contains tools for sensor hardware
# identification and I2C bus probing.
#
Executing install script for lm_sensors-3.4.0-x86_64-1.txz.
Package lm_sensors-3.4.0-x86_64-1.txz installed.
```

Видалення відбувається утилітою ***removepkg:*** ***removepkg lm_sensors***. Під час видалення на консоль виводяться всі файли, що будуть видалені.

```
root@slak:~/lm# removepkg lm_sensors

Removing package /var/log/packages/lm_sensors-3.4.0-x86_64-1...
Removing files:

  --> Deleting symlink /usr/lib64/libsensors.so
  --> Deleting symlink /usr/lib64/libsensors.so.4
  --> Deleting symlink /usr/man/man5/sensors3.conf.5.gz
  --> /etc/sensors3.conf.new no longer exists. Skipping.
  --> Deleting /usr/bin/sensors
  --> Deleting /usr/bin/sensors-conf-convert
  --> Deleting /usr/doc/lm_sensors-3.4.0/CHANGES
  --> Deleting /usr/doc/lm_sensors-3.4.0/CONTRIBUTORS
  --> Deleting /usr/doc/lm_sensors-3.4.0/COPYING
  --> Deleting /usr/doc/lm_sensors-3.4.0/COPYING.LGPL
  --> Deleting /usr/doc/lm_sensors-3.4.0/INSTALL
  --> Deleting /usr/doc/lm_sensors-3.4.0/README
  --> Deleting /usr/doc/lm_sensors-3.4.0/doc/developers/applications
  --> Deleting /usr/doc/lm_sensors-3.4.0/doc/developers/lm_sensors.lsm
```

Розділ 3.
КОНСОЛЬНІ ТА ГРАФІЧНІ ТЕКСТОВІ РЕДАКТОРИ

3.1. Редактор vim

Програма, яка призначена для редагування файлів, що містять тільки текст, називається **текстовим редактором**. У дистрибутивах системи Linux встановлено достатньо багато різних текстових редакторів, єдиним редактором, який гарантовано встановлений у будь-якій системі UNIX або Linux є редактор **vi** *(visual editor)*.

Перша версія цього редактора була написана Біллі Джоем, студентом Каліфорнійського університету в місті Берклі, який пізніше став співзасновником Sun Microsystems у 1976 р. Він призначався для редагування на екрані відеотермінала з можливістю переміщення курсору всією його поверхнею, саме від цього і походить його назва. До екранних редакторів існували малі редактори, які дозволяли редагування тексту тільки по одному рядку. З появою відеотерміналів стало можливим візуальне редагування на екрані.

Варто також зазначити, що на сьогодні більшість дистрибутивів Linux мають несправжній редактор **vi**, а його поліпшену заміну, що має назву **vim** (скорочено від *Vi IMproved* – Vi покращений), яка була створена Брамом Моленаром (Bram Moolenaar). **Vim** досконаліше традиційного редактора **vi** і в системах Linux зазвичай використовується під символічним посиланням (або псевдонімом) **vi**.

Для початку роботи з редактором **vi** у командній оболонці використовується однойменна команда: ***vi***.

При виконанні команди без аргументу на початку редагування відкриється редактор з порожнім файлом, а також виведена довідкова інформація щодо нього (рис. 3.1).

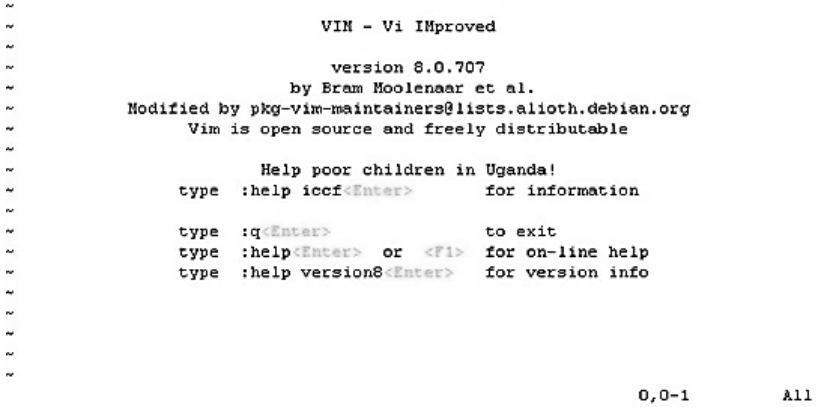

Рис. 3.1. Вікно редактора **Vim** з довідковою інформацією при створенні порожнього файлу для редагування

Таким чином, ми дізналися що у дистрибутиві Debian встановлено покращену версію редактора **vi** – **vim**, відкрити яку можна за допомогою команди ***vim***, а також псевдоніму ***vi***.

У якості аргументу до команди ***vi*** або ***vim*** можна задати назву файлу, який потрібно відкрити для редагування. Якщо такого файлу до цього не існувало, то він буде створений. У цьому випадку вікно редактора матиме вигляд, зображений на рис. 3.2:

```
vi textfile1
~
~
~
~
~
"textfile1" [New File]                                    0,0-1         All
```

Рис. 3.2. Вікно редактора **vim** при відкритті порожнього файлу, назву якого задано в аргументі команди

Якщо задати у якості аргументу назву існуючого файлу, в якому до того ж збережена інформація, то він відкриється для редагування (рис. 3.3).

```
Welcome to Ukraine You will never regret, if you visit one of the biggest Europe
an countries, Ukraine. Thousands of tourists from all over the world visit Ukrai
ne each year. They come to see the beauty of Ukrainian mountains, the Crimean Mo
untains in the south and the Carpathians in the west, with their highest point H
overla (2061 m). The climate of Ukraine is mild. It can be relatively warm winte
r, especially in the south. The South is popular in summer. The Black Sea and th
e Sea of Azov are popular places for summer rest not only among Ukrainians, but
also among citizens of neighboring countries, on which Ukraine borders: Russia.
Belarus, Poland, Slovakia, Hungary, Romania and Moldova.
The main river in Ukraine is the Dnieper, the third longest river in Europe whic
h divides Ukraine into the Right-Bank and the Left-Bank territories. The capital
 of Ukraine, Kiev is also divided by the Dnieper into two parts. Several other l
arge rivers flow through Ukraine, such as the Southern Buh, the Dniester, the De
sna, the Siversky Donets, and others. They make villages and towns of Ukraine mo
re picturesque.
~
~
~
~
~
~
~
~
"textfile1" 2L, 1113C                                     1,1           All
```

Рис. 3.3. Вікно редактора **vim** на початку редагування існуючого файлу

Однією з основних особливостей редактора **vim** (**vi**) є те, що він може працювати двома режимами: **командному режимі** та **режимі вставки**. Під час запуску він починає працювати у **командному режимі**, в якому можна задавати різні команди для редагування файлу, однак не вводити новий текст. Як правило, команди, які використовуються в командному режимі, складаються з одного або двох символів. Більшість команд будуть розглянуті нижче. Зокрема в **режимі вставки** відбувається основне редагування тексту.

Деякі досвідчені користувачі Linux також виділяють і третій режим редактора **vim (vi)** – **режим останнього рядка**. Фактично він є підрежимом режиму командного рядка, оскільки дозволяє ввести спеціальні команди, які починаються з символу ‘**:**’ та зазвичай складаються з декількох символів, або навіть слів. Подібні команди відображаються на екрані в останньому рядку тоді як більшість команд, які не починаються з символу ‘**:**’, не відображаються. Типовими їх прикладами є команди виходу з редактора зі збереженням (**:wq**) та без збереження змін (**:q!**). Приклад режиму останнього рядка представлений на рис. 3.4.

```
Welcome to Ukraine You will never regret, if you visit one of the biggest Europe
an countries, Ukraine. Thousands of tourists from all over the world visit Ukrai
ne each year. They come to see the beauty of Ukrainian mountains, the Crimean Mo
untains in the south and the Carpathians in the west, with their highest point H
overla (2061 m). The climate of Ukraine is mild. It can be relatively warm winte
r, especially in the south. The South is popular in summer. The Black Sea and th
e Sea of Azov are popular places for summer rest not only among Ukrainians, but
also among citizens of neighboring countries, on which Ukraine borders: Russia.
Belarus, Poland, Slovakia, Hungary, Romania and Moldova.
The main river in Ukraine is the Dnieper, the third longest river in Europe whic
h divides Ukraine into the Right-Bank and the Left-Bank territories. The capital
 of Ukraine, Kiev is also divided by the Dnieper into two parts. Several other l
arge rivers flow through Ukraine, such as the Southern Buh, the Dniester, the De
sna, the Siversky Donets, and others. They make villages and towns of Ukraine mo
re picturesque.
~
~
~
~
~
~
~
~
:wq
```

Рис. 3.4. Режим останнього рядка в редакторі **vim (vi)**

3.1.1. Робота редактора в режимі вставки

Для початку редагування тексту необхідно перейти у режим вставки. Такий перехід дозволяють здійснити три основні команди:

– **`i`** – розпочати вставку тексту на поточній позиції курсору;

– **`a`** – розпочати вставку після поточної позиції курсору;

– **`o`** – розпочати вставку на новому рядку нижче поточого абзацу (можна вводити команду, знаходячись у будь-якому місці поточного абзацу).

Кожна із зазначених команд також має модифікацію у вигляді команди з тією самою літерою, але у верхньому регістрі:

– **`I`** – розпочати вставку тексту на початку поточного абзацу перед першим ненульовим символом;

– **`A`** – розпочати вставку в кінці поточного абзацу;

– **O** – розпочати вставку на новому рядку нижче поточного абзацу (можна вводити команду, знаходячись у будь-якому місці поточного абзацу).

Будь-яка з описаних вище команд переведе редактор у режим вставки. Існують і інші команди, які переводять редактор у режим вставки (наприклад, під час заміни символу або слова), які будуть розглянуті далі. Для того, щоб повернутись у командний режим, потрібно натиснути клавішу **<ESC>**. Натиснення цієї клавіші у командному режимі, ніяк не впливає на текст у файлі, навіть якщо натиснути її декілька разів.

3.1.2. Робота редактора у командному режимі

У командному режимі редактора **vi** можна виконувати наступні дії:

- редагування та заміна існуючого тексту;
- видалення, переміщення та копіювання тексту;
- пересування всім файлом;
- інші завдання.

Команди переміщення курсору

Існує багато команд, що дозволяють зробити переміщення поточної позиції курсору від простих до більш складних.

Найбільш простим є переміщення курсору на сусідні позиції з 4 сторін. Для цього призначені наступні команди, які виконуються за допомогою клавіш, що розташовані поруч:

- **h** – перемістити курсор на один символ ліворуч;
- **j** – перемістити курсор вниз на один абзац;
- **k** – перемістити курсор вгору на один абзац;
- **l** – перемістити курсор на один символ праворуч.

Додамо, що команди переміщення курсору вниз і вгору (**j** та **k** відповідно) зберігають його позицію на абзаці. Наприклад, якщо курсор знаходився на 10-му символі абзацу, при здійсненні команди **j** або **k**, він переміститься на 10-й символ абзацу, що розташований нижче або вище відповідно. У випадку, якщо в абзаці, на який переходить курсор, кількість символів є меншою, то він переміщатиметься на останній символ абзацу.

Деякі термінали мають спеціальні символи управління курсором, помічені стрілками. Їх можна використовувати замість описаних вище команд. Однак більш універсальними все ж є команди, що представляють літеру. Зокрема, можна перемістити курсор більш ніж на один символ в одну зі сторін. Для цього перед відповідною клавішею, що означає команду переміщення курсору, необхідно вказати число, яке означає, на яку кількість символів потрібно перемістити курсор.

Наприклад:

- **5l** – перемістити курсор на 5 символів праворуч;
- **4k** – перемістити курсор на 4 абзаци вгору.

І так далі.

Якщо певна команда з числом не може перемістити курсор на зазначену кількість символів абзаців у відповідному напрямі, редактор подасть звуковий сигнал і курсор залишиться на колишньому місці.

Додатково для переміщення курсору ліворуч та праворуч використовують клавіші **<Spacebar>** та **<Backpace>**, причому разом з ними також можуть використовуватися числа, які вказують на скільки символів потрібно переміститися. Використання клавіш без чисел здійснить переміщення курсору на один символ у відповідному напрямі.

Більш складні команди переміщення курсору

Окрім переміщення курсору на сусідні символи також існують більш складні варіанти переміщення, яким відповідають спеціальні команди. Їх можна класифікувати таким чином:

- команди позиціонування за символами;
- команди позиціонування за абзацами (рядками);
- команди позиціонування за словами;
- команди позиціонування за реченнями;
- команди позиціонування за розділами (параграфами).

Позиціонування за символами

Крім розглянутих команд переміщення сусідніми символами, у **vi** існує багато додаткових команд переміщення на конкретний символ, якщо він не є сусіднім з поточною позицією курсору. Подібні команди дозволяють швидше переміщуватися текстом у межах файлу.

Розглянемо спочатку команди переміщення на певний символ в межах поточного рядка. Для цього призначені наступні команди:

- ***0*** (нуль) – перемістити курсор на перший символ в абзаці;
- **^** – перемістити курсор на перший ненульовий символ в абзаці;
- ***$*** – перемістити курсор на останній символ в абзаці.

Крім розглянутих вище команд, що є так званими командами безумовного переміщення символу, існують також спеціальні команди переміщення курсору на відповідний символ, який потрібно спочатку знайти. Якщо на поточному рядку шуканий символ не знайдено, редактор видасть звуковий сигнал попередження і курсор залишиться на місці. Для пошуку символу всередині рядка з подальшим переміщенням курсору редактор **vi** має 4 команди:

– ***f<x>*** – шукати зазначений символ ***x*** праворуч від поточної позиції курсору та перемістити курсор на нього;

– ***F<x>*** – шукати зазначений символ ***x*** ліворуч від поточної позиції курсору та перемістити курсор на нього;

– ***;*** – повторити попередній пошук символу: запам'ятати попередній шуканий символ і шукати наступну появу символу в поточному абзаці праворуч з подальшим переміщенням курсору на нього;

– ***,*** – повторити попередній пошук символу, але у зворотному напрямі: запам'ятати попередній шуканий символ і шукати наступну появу символу в поточному абзаці ліворуч з подальшим переміщенням курсору на нього.

До того ж існують додаткові команди переміщення за символами вниз та вгору, які доповнюють команди **j** та **k.** Різниця тільки у тому, що ці команди позиціонують курсор на першому ненульовому символі. Такими командами є ***+*** та ***–,*** які також можна використовувати з числами перед ними, що вказують кількість символів, на які потрібно зробити переміщення.

Наприклад:

– ***5-*** – перемістити курсор на 5 символів угору на ненульовий символ;

– ***7+*** – перемістити курсор на 7 символів вниз на ненульовий символ.

Позиціонування за абзацами

У редакторі **vi** існують також спеціальні команди, що дозволять перейти на потрібний абзац. Це наступні команди:

– ***gg*** – перемістити курсор на початок документу (перший символ першого абзацу);

– ***G*** – перемістити курсор в кінець документу (перший символ останнього абзацу);

– ***<n>G*** – перемістити курсор на перший символ абзацу, порядковим номером якого в документі є ***n.***

Позиціонування за словами

З точки зору редактора **vi** словом є рядок символів, у який можуть входити букви, цифри та символи підкреслення. У редакторі є 6 команд позиціонування за словами, які у якості роздільника розглядають будь-який символ, який відрізняється від букви, цифри або символу підкресленення. Роздільниками слів є символи початку та кінця рядка, а також знаки пунктуації до, та після порожнього простору.

Відповідні команди позиціонування за словами:

– ***b*** – перемістити курсор до першого символу попереднього слова;

– ***w*** – перемістити курсор на початок наступного слова;

– ***e*** – перемістити курсор до останнього символу наступного слова.

Перераховані вище команди можуть використовуватися з цілими числами попереду них. У цьому випадку буде здійснено переміщення курсору на відповідну кількість слів від його поточної позиції.

Для зазначених команд також існують модифікації, які у якості роздільників слів приймають тільки символ пробілу та переходу на новий рядок, ігноруючи при цьому всі знаки пунктуації. Такі команди представляють ті ж самі букви, але у верхньому регістрі: ***B***, ***W*** та ***E*** відповідно.

Позиціонування за реченнями

Редактор **vi** також розпізнає речення. У якості розділювачів речень можуть виступати символи '***.***', '***!***' або '***?***'. Однак, якщо ці символи з'являються всередині рядка, то за ними мають стояти два пробіли, щоб редактор **vi** міг їх розпізнати.

Командами позиціонування за реченнями є команди ***(*** та ***)***, які можна застосовувати з цілими числами перед ними, що буде означати переміщення на вказану кількість речень:

– ***(*** – перемістити курсор на початок поточного (попереднього) речення;

– ***)*** – перемістити курсор на початок наступного речення.

Позиціонування за параграфами

Редактор **vi** також вміє розпізнавати параграфи. Для цього вони мають бути розділені порожнім рядком. Для позиціонування курсору за параграфами використовуються команди ***{*** та ***}***, перед якими як і у випадках інших видів позиціонування можна вказати ціле число для переміщення на певну кількість параграфів:

– ***{*** – перемістити курсор на початок поточного параграфа;

– ***}*** – перемістити курсор на початок наступного параграфа.

Позиціонування у вікні

У редакторі **vi** є три досить цікаві команди, які дозволяють позиціонувати курсор у вікні. Ці команди представляють букви верхнього регістру:

– ***H*** – перемістити курсор до першого абзацу на екрані;

– ***M*** – перемістити курсор до середнього абзацу на екрані;

– ***L*** – перемістити курсор до останнього абзацу на екрані.

Усі команди переміщують курсор до першого символу відповідного абзацу.

Для пошуку символів за шаблоном ви можете використовувати команди ***/***, ***?***, ***n*** і ***N***, які дозволяють здійснювати пошук вперед і назад у

буфері наступної появи зазначеного шаблону символів. Команди ***n*** і ***N*** дозволяють вам повторити останній пошук.

Пошук символів та слів за шаблоном

Окремим способом позиціонування курсору є пошук символів та слів за шаблоном. Для такого пошуку в редакторі **vi** використовуються команди ***/, ?, n*** та ***N.*** Перші дві команди дозволяють здійснювати пошук наступної появи зазначеного шаблону символів у буфері вперед і назад, а останні дві повторити попередній пошук.

Команда ***/*** і наступний шаблон (***/pattern***) здійснює пошук появи символів, що зазначені у шаблоні та переміщує курсор на перший символ з відповідного шаблону. Наприклад, команда ***/hello world*** знаходить першу появу слів **'hello world'** у буфері і переміщує курсор на букву **h.**

Відповідно команда ***?*** здійснює пошук шаблону в зворотному порядку. Наприклад, команда ***?data set design*** знаходить попередню появу слів **'data set design'** і переміщує курсор під буквою **d** у слові **data.**

Важливим є те, що зазначені вище команди не виконують циклічного повернення у разі закінчення абзацу під час пошуку двох або більше слів. Наприклад, якщо потрібно знайти появу рядка ***'Hello world'*** і виявиться, що слово ***'Hello'*** знаходиться в кінці попереднього рядка, а слово ***'world'*** є першим словом наступного, то ці команди відповідний рядок не знайдуть.

Водночас, зазначені команди пошуку виконують циклічне повернення в кінці або на початку буфера для продовження пошуку. Наприклад, якщо ми знаходимося майже в кінці буфера, а символи, які відповідають шаблону, на початку, то команда пошуку ***/*** знайде шаблон.

Натомість команди ***n*** та ***N*** дозволяють повторити попередній запит пошуку ***/pattern*** або ***?pattern.***

Варто звернути увагу на те, що з командами ***/*** та ***?*** не можна використовувати цілі числа на відміну від багатьох інших розглянутих команд. Тобто ми не можемо, наприклад, знайти третю порядкову появу шаблону, починаючи з поточної позиції.

Команди модифікації тексту

Одним зі способів модифікації тексту є команди видалення і команди введення тексту. Іншим способом модифікації є використання команди, яка дозволяє видалення і створення тексту одночасно. Це команди заміни, підстановки та зміни тексту.

Заміна тексту

Для заміни тексту в редакторі **vi** призначені наступні команди:

– ***r<x>*** – замінити символ, на який показує курсор, на символ ***x***. Команда не переводить редактор у режим вставки, тому немає необхідності натиснення після неї клавіші ***<ESCAPE>***;

– ***<n>r<x>*** – замінити ***n*** символів на символ ***x***. Команда не переводить редактор у режим вставки, а отже не вимагає після її використання повернення у командний режим;

– ***R*** – перевести редактор у режим вставки, однак при введенні нового тексту символи, що знаходяться після курсору, не зміщуються, а замінюються новим текстом. Після завершення заміни цією командою потрібно повернутися у командний режим за допомогою клавіші ***<ESCAPE>***. Якщо під час редагування було досягнуто кінця рядка, то ця команда додасть введення як нового тексту.

Приклади

1. Замінити у реченні **‘The circus has many acts.’** слово **‘acts’** на **‘ants’**.

Для цього потрібно перемістити курсор на букву ‘c’ у слові ‘acts’, а потім ввести команду ***rn***. Речення набуде наступного вигляду:

The circus has many ants.

2. У тому самому реченні замінити слово **‘many’** на **‘7777’**.

Для цього переміщуємо курсор під буквою ‘m’ у слові ‘many’ і вводимо команду ***4r7***, яка виконає наступну заміну:

The circus has 7777 ants.

Підстановка тексту

Команди підстановки тексту замінюють символи, але потім дозволяють продовження введення тексту з того місця, на якому було натиснуто ***<ESCAPE>***. Є наступні команди підстановки тексту:

– ***s*** – видалити символ, на який вказує курсор і додати текст, тим самим перевівши редактор у режим вставки;

– ***<n>s*** – видалити n символів і додати текст, так само перевівши редактор у режим вставки;

– ***S*** – замінити всі символи в рядку (абзаці). При цьому весь абзац буде видалений.

Зміна тексту

Команди зміни замінюють об’єкт тексту, а потім продовжують додавати його допоки не буде натиснута клавіша ***<ESCAPE>***, тобто всі

відповідні команди як і команди підстановки також переводять редактор у режим вставки. Для зміни тексту призначені наступні команди:

– ***cw*** – замінити слово або символи, які залишились у слові, на новий текст;

– ***<n>cw*** – замінити ***n*** слів;

– ***cl*** – замінити символ на новий текст;

– ***<n>cl*** – замінити ***n*** символів;

– ***cc*** – замінити всі символи в рядку;

– ***C*** – замінити решту символів у рядку від курсору до кінця рядка.

Команди видалення тексту

Редактор **vi** дозволяє видалити текст як у режимі вставки, так і у командному режимі.

Якщо редактор перебуває в режимі вставки, то для видалення символу потрібно скористатися клавішею ***<BACKSPACE>***, яка видалить символ ліворуч від курсору. Ця клавіша переміщує курсор назад під час введення літер і видаляє кожен символ, на який він переходить. У режимі вставки також можна застосувати комбінацію клавіш ***<CTRL>+<w>***, яка скасовує введення поточного слова.

Для скасування останньої введеної команди потрібно у командному режимі застосувати одну з наступних команд:

– ***u*** – скасувати останню команду;

– ***U*** – відновити поточний рядок у стан до проведених змін.

Для скасування випадково видалених рядків доцільно виконати команду ***u***, у результаті чого видалені рядки знову з'являться на екрані. На відміну від команди ***u***, команда ***U*** анулює всі зміни в поточному рядку, поки курсор на ньому буде перебувати.

Однак, потрібно зауважити, що при виконанні команди ***u*** двічі, друга команда скасує дію першої. Тобто при помилковому видаленні рядка з подальшим його відновленням командою ***u***, при виконанні другої команди ***u*** рядок знову буде видалений.

Для видалення певного тексту також призначено багато корисних команд, які потрібно виконувати у командному режимі. Вони одразу ж видаляють вказаний текст з екрана, а решту відображують в оновленому вигляді.

Для видалення тексту в командному режимі редактора **vi** існують наступні команди:

– ***x*** – видалити поточний символ;

– ***dw*** – видалити слово над курсором і наступний за ним пробіл, або знак пунктуації;

– ***dW*** – видалити слово і знак пунктуації над курсором з наступним за ним пробілом;

– ***dd*** – видалити поточний рядок;

– ***d)*** – видалити поточне речення від поточної позиції курсору до кінця;

– ***d}*** – видалити поточний параграф від поточної позиції до кінця;

– ***<n>d<x>*** – видалити ***n***-ий об'єкт тексту ***<x>***, у якості якого може бути: ***d*** – рядок; ***w*** – слово; ***)*** – речення; ***}*** – параграф;

– ***D*** – видалити частину рядка праворуч від курсору.

Команди витягу і приєднання тексту

Редактор **vi** також надає набір команд, які розбивають текст у файлі, копіюють частину тексту та поміщають його в інше місце у файлі.

Переміщення тексту

Існує можливість переміщення тексту з одного місця в буфері редактора **vi** в інше, спочатку видаливши рядки, а потім помістивши їх у потрібне місце. Видалений текст буде зберігатися у тимчасовому буфері. Перемістивши курсор на те місце, куди потрібно вставити текст, і виконавши команду ***p***, видалені рядки будуть додані нижче поточного рядка.

Дистанційний рядок може бути розміщений також усередині іншого рядка. При позиціонуванні курсору між двома словами та виконанні команди ***p*** дистанційний рядок буде поміщений після курсору.

Оскільки у тимчасовому буфері зберігається результат тільки останньої команди, команда ***p*** повинна використовуватися відразу ж після команди видалення. Вона використовується також для копіювання тексту, який поміщений у тимчасовий буфер командою ***y***, яка буде розглянута нижче.

Переміщення букв

Швидко перемістити букви можна використавши команду ***xp***, яка насправді представляє комбінацію команд ***x*** та ***p***: ***x*** видаляє букву, а ***p*** поміщує її після наступного символу.

Наприклад, у наступному реченні є помилка: **tetx.** Помістивши курсор під другою літерою **"t"**, та потім натиснувши почергово клавіші ***x*** і ***p***, у результаті отримаємо: **text.**

Копіювання тексту

Редактор **vi** також надає набір команд для копіювання тексту в тимчасовий буфер з подальшою його вставкою в інше певне місце у файлі. Для того, щоб помістити текст на нове місце, використовується команда ***y*** в своїх різних модифікаціях. Загальний синтаксис команди ***y*** є наступним:

[<Число>]y[<Об'єкт_тексту>]

Потрібно звернути увагу, що при копіюванні тексту за допомогою команди ***y***, він не видаляється з колишнього місця в файлі, тобто можна помістити один і той же фрагмент тексту одразу в декількох місцях.

Команда ***y*** в редакторі **vi** має наступні різновиди:

– ***<n>y<x>*** – копіювання ***n***-го об'єкта тексту типу ***<x>***: ***)*** – речення, ***}*** – параграф;

– ***yw*** – копіювання слова;

– ***yy*** – копіювання поточного рядка;

– ***<n>yy*** – копіювання ***n*** рядків;

– ***y)*** – копіювання всього тексту до кінця речення;

– ***y}*** – копіювання всього тексту до кінця параграфа.

Утім переміщення і копіювання кількох фрагментів тексту в різні частини файлу іноді є досить кропіткою роботою. Для спрощення таких дій редактор **vi** пропонує використання регістрів, на яких можна зберігати текст, допоки він не знадобиться. Зберегти текст можна за допомогою команди ***y***, або команд видалення. Використання регістрів є особливо корисним, якщо частину тексту потрібно вставити в багатьох місцях у файлі.

Загальний формат команди:

[<Число>]["<X>]<команда>[<Об'єкт_тексту>]

де ***<X>*** – назва регістра, що складається з однієї літери верхнього регістру. Його назві при цьому мають передувати подвійні лапки.

Завершення роботи з редактором vim

Для завершення роботи з редактором **vi** можна використати команди, які починаються зі знаку '***:***', тобто відображуються на екрані у найнижчому рядку редактора у режимі останнього рядка. До того ж, після введення таких команд завжди потрібно натискати ***<ENTER>***.

Перерахуємо ці команди:

– ***ZZ*** – записати буфер в файл, якщо після останнього запису в буфері були зроблені зміни, і завершити роботу з **vi**;

– ***:wq*** – записати буфер у поточний файл і завершити роботу з **vi**;

– ***:w <filename>*** – записати вміст буфера в новий файл ***filename***;

– ***:w! <filename>*** – замінити вміст буфера в уже існуючий файл ***filename***;

– ***:q!*** – завершити роботу з **vi** без запису у файл, навіть якщо вміст буфера змінювався;

– ***:q*** – завершити роботу з **vi** без запису у файл, якщо вміст буфера не змінювався.

Остання з команд буде працювати у тому випадку, якщо у буфері не було ніяких змін. В іншому випадку редактор **vi** виведе попередження про те, що потрібно або зберегти буфер або для завершення роботи виконати команду: ***q!***.

3.2. Редактор nano

Для користувачів, які тільки починають вивчення системи Linux, редактор **vi** може здатися досить складним. Більш простим та зручним редактором у консольному режимі в актуальних версіях дистрибутивів Linux наразі є редактор, що має назву **Nano**. У багатьох дистрибутивах (включаючи також Debian та Ubuntu) його непотрібно встановлювати додатково, оскільки він вже є встановленим за замовчуванням.

Nano (а точніше **GNU nano**) – є клоном текстового редактора **Pico**, який поставляється разом поштовим клієнтом Alpine. Він з'явився у 1999 р., коли Кріс Аллегретта почав створювати клон редактора **Pico**. Спочатку новий редактор отримав назву **TIP**, яка представляла собою акронім: TIP is not Pico («TIP – це не Pico»). Однак, потім достатньо швидко назву програми довелося змінити на **Nano**. Причиною став конфлікт щодо її назви, оскільки вже була програма, яка також називалася **tip**, та відповідала за встановлення зв'язку терміналу з віддаленим хостом і входила до складу багатьох дистрибутивів UNIX.

Розробка програми Nano отримала подальший розвиток у 2001 р., коли Річард Столлман оголосив її офіційною програмою проекту GNU. Перший стабільний реліз редактора Nano побачив світ 22 березня 2001 р.

Початок роботи з редактором **nano** нагадує початок роботи з редактором **vi**. Для запуску редактора використовується однойменна команда, виконання якої без аргументу відкриє порожній файл для редагування, а з аргументом – існуючий файл на:

```
nano textfile1
```

На рис. 3.5 показано вікно текстового редактора **nano** при відкритті для редагування файлу **textfile1**.

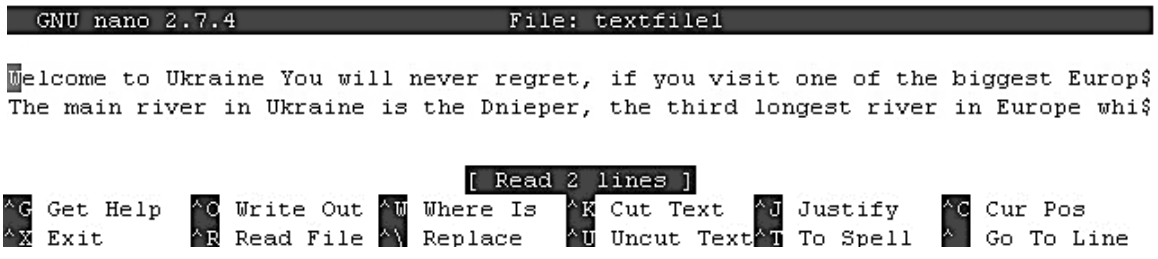

Рис. 3.5. Зовнішній вигляд редактора **nano** при відкритті існуючого файлу для редагування

Як можна побачити із зображення, за замовчуванням редактор **nano** не показує весь текст на рядку, він як і редактор **vi** приймає абзац. Якщо рядок при цьому незавершений, тобто у ньому ще залишилися невидимі символи.

Головною відмінністю редактора **nano** від **vi** є те, що у **nano** є тільки один режим, в якому можна як редагувати текст, так і виконувати певні додаткові команди. Характерною особливістю редактора **nano** є те, що він має область, яка відіграє роль підказки для введення відповідних команд. Вона розташована у самому низу екрана редактора. З рис. 3.5 можна побачити наступні інформуючі надписи: ***^G*** – **Get Help**, ***^X*** – **Exit** і т. д., які є підказками для виконання відповідних команд.

Натомість іншою справою є те, що символ '**^**' у цьому випадку не означає, що для виконання певної команди його потрібно ввести (тобто ***<Shift>+<6>***) разом з відповідною літерою, що і означає введення команди. Область підказки має наступну систему позначень:

– символом '**^**' позначені команди, які вводять за допомогою натискання клавіші ***<Ctrl>*** або подвійного натискання клавіші ***<Esc>***;

– символом '***M***' позначені команди, які вводять за допомогою клавіш ***<Esc>***, ***<Alt>*** або ***<Meta>*** залежно від використовуваної клавіатури (далі для спрощення).

Також при подвійному натисканні клавіші ***<Esc>*** і подальшого введення тризначного числа від 000 до 255 буде введений певний символ, який відповідатиме ASCII-коду введеного тризначного числа.

Варто зазначити, що з рис. 3.5 в області підказок не видно відповідних надписів із символом «***M***», оскільки вони не помістилися тому, що вікно редактора **nano** не було відкрите на весь екран. Якщо вікно відкрите на весь екран кількість підказок буде збільшена, і серед підказок можна буде побачити і комбінації, що починаються з символу «***M***».

У табл. 3.1 перераховані деякі важливі комбінації клавіш, що використовуються в редакторі **nano.** Зверніть увагу, що деякі команди можуть бути виконані натисканням функціональної клавіші, а також мати різні варіанти комбінації як з використанням клавіші ***<Ctrl>***, так і з ***<Esc>***.

Табл. 3.1. Важливі команди редактора nano

Комбінація клавіш	**Додаткова комбінація клавіш або функціональна клавіша**	**Дія, яку виконує команда**
<Ctrl>+<G>	***<F1>***	показати довідку
<Ctrl>+<X>	***<F2>***	закрити поточний буфер / вийти з **nano**
<Ctrl>+<O>	***<F3>***	записати поточний файл на диск
<Ctrl>+<J>	***<F4>***	вирівняти поточний абзац
<Ctrl>+<R>	***<F5>***	вставити вміст іншого файлу у поточний
<Ctrl>+<W>	***<F6>***	шукати текст або регулярний вираз

Закінчення табл. 3.1

<Ctrl>+<Y>	**<F7>**	перейти на попередній екран
<Ctrl>+<V>	**<F8>**	перейти на наступний екран
<Ctrl>+<K>	**<F9>**	вирізати поточний рядок і зберегти її у буфері обміну
<Ctrl>+<U>	**<F10>**	вставити вміст буфера обміну в поточний рядок
<Ctrl>+<C>	**<F11>**	показати положення курсора
<Ctrl>+<T>	**<F12>**	перевірити орфографію
<ESC>+<\>	**<ESC>+<|>**	перейти на перший рядок файлу
<ESC>+</>	**<ESC>+<?>**	перейти на останній рядок файлу
<Ctrl>+<\>	**<Esc>+<R>**	замінити текст або регулярний вираз
<Ctrl>+<^>	**<Esc>+<Alt>**	відзначити текст в поточній позиції курсору
<Esc>+<W>		повторити останній пошук
<Esc>+<^>	**<Esc>+<6>**	копіювати поточний рядок і зберегти її у буфері обміну
<Esc>+<}>		збільшити відступ рядка
<Esc>+<{>		зменшити відступ рядка
<Ctrl>+<F>		перейти вперед на один символ
<Ctrl>+<B>		перейти назад на один символ
<Ctrl>+<Space>		перейти вперед на одне слово
<Esc>+<Space>		перейти назад на одне слово
<Ctrl>+<P>		перейти на попередній рядок
<Ctrl>+<N>		перейти на наступний рядок
<Ctrl>+<Alt>		перейти на початок поточного рядка
<Ctrl>+<E>		перейти в кінець поточного рядка
<Esc>+<(>	**<Esc>+<9>**	перейти на початок поточного абзацу; при другому виклику перейти на початок наступного абзацу
<Esc>+<)>	**<Esc>+<0>**	перейти на поточний абзац; при другому виклику перейти на кінець наступного абзацу
<Esc>+<->	**<Esc>+<_>**	прокрутити один рядок вгору, не переміщаючи курсор
<Esc>+<+>	**<Esc>+<=>**	прокрутити один рядок вниз, не переміщаючи курсор
<Esc>+<<>	**<Esc>+<,>**	переключити редактор на попередній буфер
<Esc>+<>>	**<Esc>+<.>**	переключити редактор на наступний буфер.

Більш детально окремо розглянемо команду виходу з файлу (**<CTRL>+<X>** або **<F2>**). Залежно від того, чи змінювався вміст файлу, команда буде діяти по-різному. Якщо змін не було, команда закриє редактор, повернувшись у командну оболонку. Якщо зміни у файлі все ж відбулися, перед виходом з редактора **nano** буде виведено додаткове запитання, чи бажає користувач зберегти зміни у файлі з варіантами відповіді **<Y>** – так (Yes), **<N>** – ні (No) та **<CTRL>+<C>** – відмінити (Cancel). Зовнішній вигляд редактора у цьому випадку показаний на рис. 3.6.

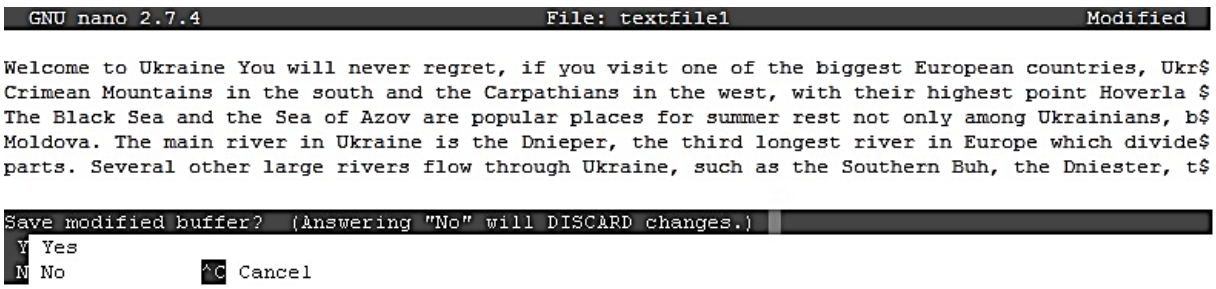

Рис. 3.6. Питання про збереження змін у файлі, у редакторі **nano**

При позитивній відповіді редактор **nano** також поставить додаткове питання щодо збереження змін у поточний файл або інший файл та запропонує ввести альтернативну назву файлу (рис. 3.7).

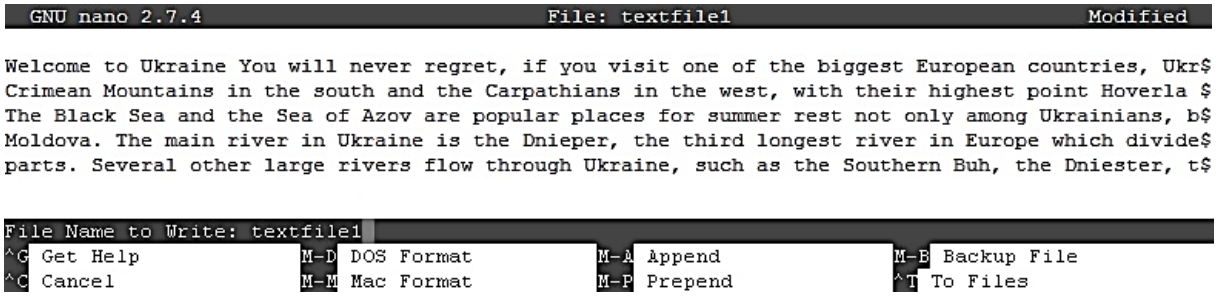

Рис. 3.7. Питання про файл, у якому потрібно зберегти інформацію, у редакторі **nano**

3.3. Інші консольні текстові редактори

У вищезазначеному розділі було розглянуто редактори **vi** та **nano**, які працюють у консольному режимі. Хоча вони є напевно найбільш популярними та застосовуваними у консольному режимі Linux, утім це не єдині консольні редактори. Існують також і інші для консольного режиму, які відомі трохи менше.

Наведемо деякі з них найбільш відомі.

3.3.1. Редактор Emacs

Emacs – найстаріший текстовий редактор, з якого розпочалася історія вільного програмного забезпечення. Проте редактором його можна назвати тільки умовно. Серед досвідчених користувачів цього редактора ходить жарт, що він уміє все, за винятком приготування кави, однак потім користувачі уточнюють, що до кінця ще не розібралися з усіма можливостями **Emacs**.

Вікно редактора **Emacs** представлене на рис. 3.8.

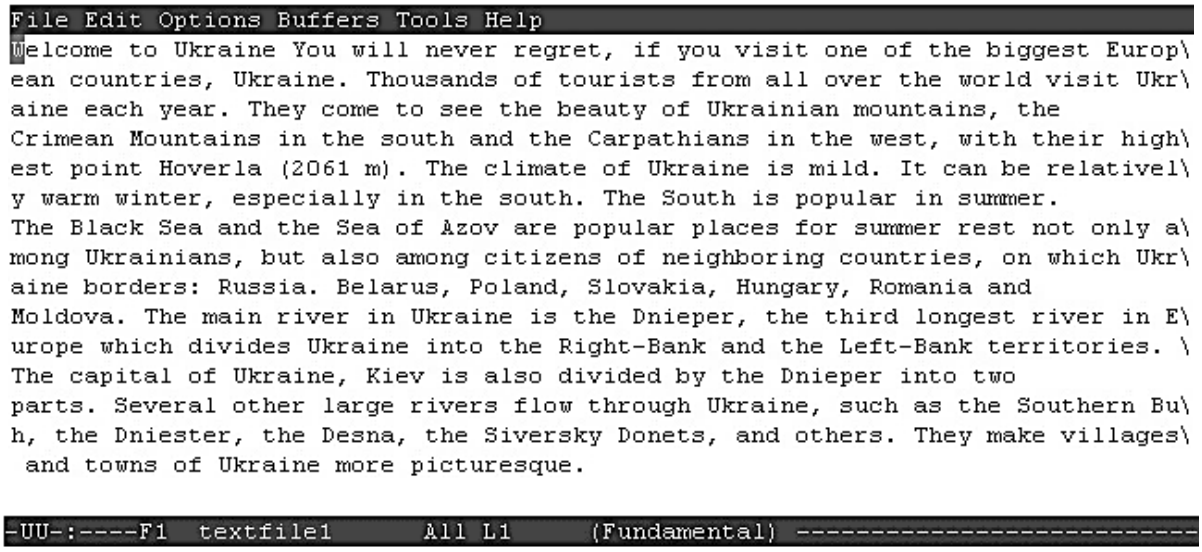

Рис. 3.8. Вікно редактора **Emasc**
з відкритим для редагування файлом

Головною перевагою редактора **Emacs** є практично необмежена можливість його налаштування. Однак у тому ж полягає і його основний недолік. Навіть грамотному і терплячому користувачеві, якому подобається читання інструкцій, необхідно витратити чимало часу для налаштування редактора згідно зі своїм власним уявленням про зручність. До того ж розвиток **Emacs** в сторону його багатогранності став причиною того, що інтуїтивно незрозуміла програма є надзвичайно складною у застосуванні. Зокрема, управління здійснюється за допомогою різних клавіатурних комбінацій, які запам’ятати доволі непросто. Частково цей недолік редактора компенсується чудовою довідковою системою, тому на практиці буває достатньо запам’ятати клавіатурну комбінацію для її виклику.

Зрозуміло, що застосування **Emacs** для первинного разового редагування конфігураційних файлів є недоцільним, оскільки витрачення часу на його вивчення обґрунтовано, якщо у користувача є намір активно його експлуатувати для вирішення своїх поточних завдань – написання коду або звичайного тексту. До речі, це зовсім не є прагненням до оригінальності, оскільки багато програмістів і нині користуються **Emacs** і вважають його одним з кращих інструментів для роботи з кодом.

3.3.2. Редактор Jed

Розглянемо ще один консольний редактор, застосовуваний у системі Linux, **Jed**. Він дозволяє використовувати меню, яке спрощує роботу з ним та робить більш наочною та простою. Однак простота насправді є тільки видимістю, оскільки деякі особливості редактора **Jed**, спрощуючи роботу для недосвідченого користувача, є досить серйозним випробу-ванням на витримку.

На рис. 3.9 представлене вікно консольного редактора **Jed**.

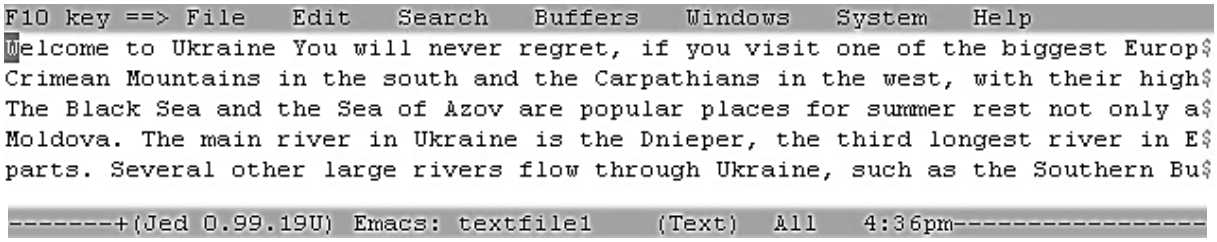

Рис. 3.9. Вікно редактора **Jed** з відкритим для редагування файлом

Зокрема, для видалення символу в **Jed** використовується тільки клавіша **Backspace** – при натисканні кнопки **Delete** редактор не реагує. Якщо ж запустити програму без назви файлу, який потрібно відредагувати, то завантажити його через меню не вийде – потрібно тільки вводити його в ручному режимі.

Однак у цього редактора є й позитивні риси: він дозволяє одночасно працювати з декількома файлами, підтримує режими підсвічування програмного коду і т. д. Деякі досвідчені користувачі називають **Jed** полегшеним редактором **Emacs**, і у цьому є певна доля правди. Принаймні у них схожі клавіатурні комбінації.

3.3.3. Редактор Joe

Joe – це консольний текстовий редактор, який керується за допомогою клавіатурних комбінацій. Його основна особливість полягає у багатогранності, оскільки він вміє маскуватися під інші аналогічні програми. Наприклад, для запуску редактора в emacs-режимі варто скористатися командою ***jmacs***

Вікно редактора Joe зображено на рис. 3.10.

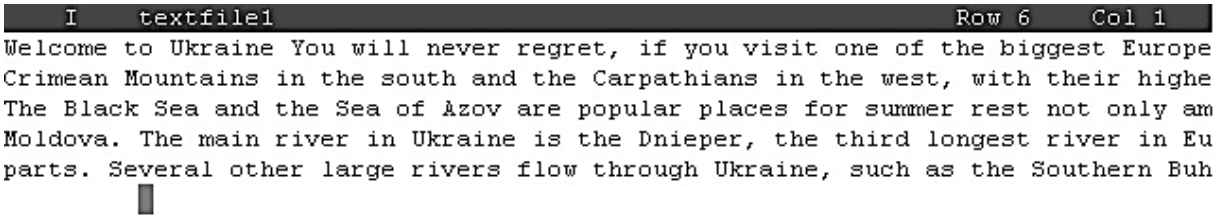

Рис. 3.10. Вікно редактора **Joe** з відкритим для редагування файлом

Незважаючи на зовнішню простоту, редактор **Joe** має багато можливостей, зокрема одночасна робота з декількома файлами, автодоповнення, підсвічування синтаксису програмного коду, він надзвичайно багатий різними опціями і навіть має власну макромову. Недолік цього редактора є традиційний – витрачання часу на його налаштування для одноразового редагування декількох файлів є недоцільним.

3.3.4. Редактор Mcedit

Mcedit є простий текстовий редактор, який вбудований у файловий менеджер **Midnight Commander**. Проте його можна використовувати для самостійного застосування.

На рис. 3.11 представлене вікно редактора **Mcedit**.

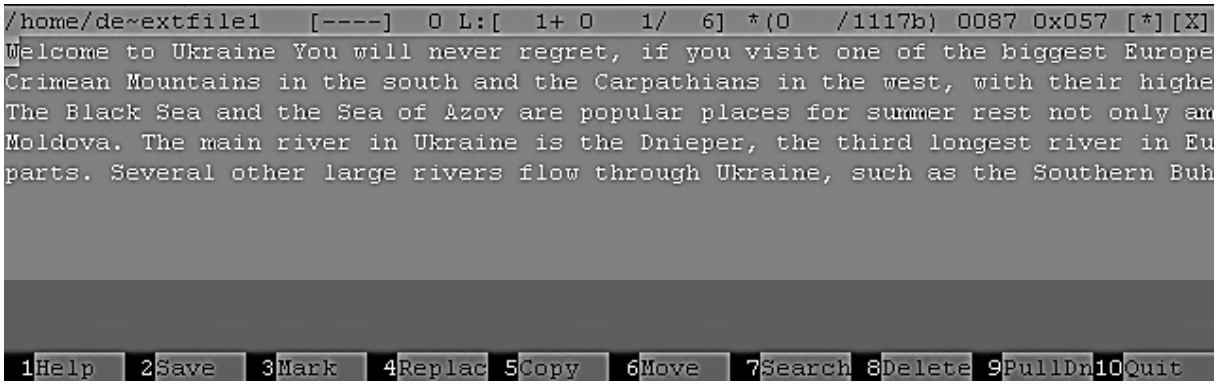

Рис. 3.11. Вікно редактора **Mcedit** з відкритим для редагування файлом

Найголовнішою перевагою цього редактора є те, що для виконання найпростіших операцій користувачеві непотрібно читати документацію. Усе інтуїтивно зрозуміло, а коротку довідку щодо необхідних функціональних клавіш можна побачити у нижній частині екрана. Також редактор **Mcedit** має налаштування, за якими можна задати деякі важливі параметри.

Mcedit – найкращий редактор для нечастого застосування, коли треба виправити буквально декілька рядків у будь-якому файлі конфігурації, однак використовувати його для чогось більш складного недоцільно.

Окрім консольних текстових редакторів існують також багато текстових редакторів для різних графічних оболонок Linux, найбільш застосовувані з яких будуть розглянуті в наступних розділах.

3.4. Графічні текстові редактори

Операційні системи на базі ядра Linux можуть використовувати не лише консольний режим роботи. Більшість дистрибутивів використовують Desktop environment (DE) – програмне забезпечення для графічного робочого столу.

3.4.1. Редактор GVIM – версія редактора vim для графічного режиму

DE викликало необхідність створення графічних інтерфейсів для багатьох консольних додатків, наприклад текстовий редактор **vi** має графічну версію **GVIM** (рис. 3.12).

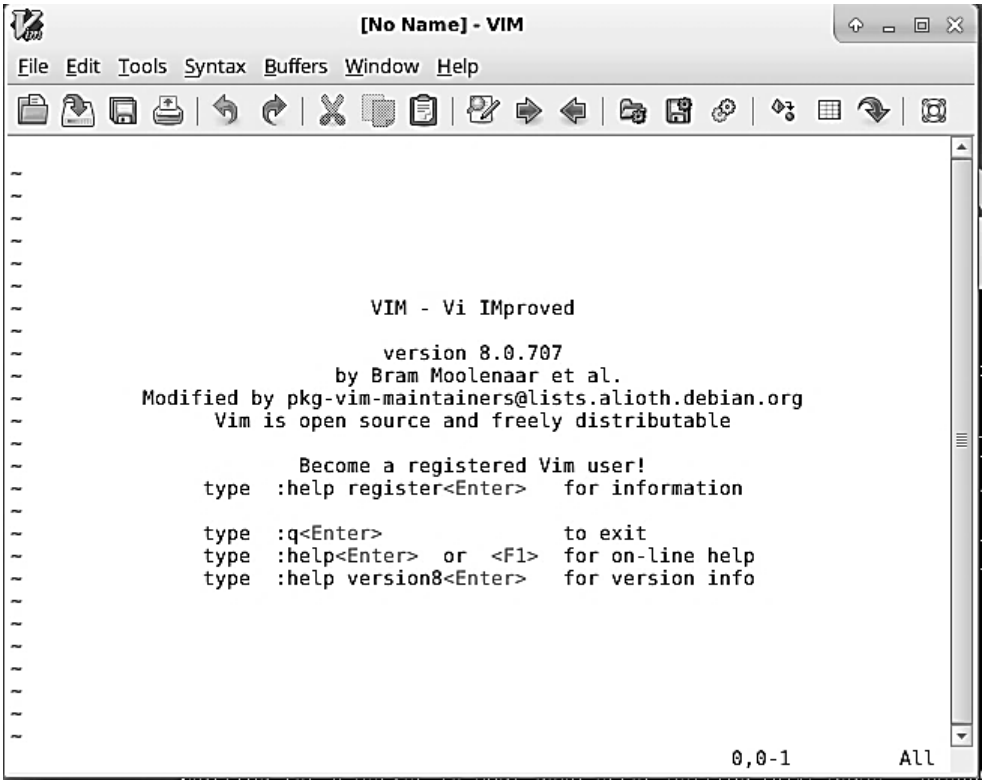

Рис. 3.12. Вікно графічного редактора **GVIM**

GVIM міститься в пакеті **vim-gtk**. На дистрибутиви, що базуються на Debian, встановлюється командою ***apt-get install vim-gtk*** (більш сучасна версія **vim-gtk3**). Запуск здійснюється командою ***vi -g***. Функції цього редактора схожі з його консольною версією, але є додаткове меню, що дозволяє користувачу працювати у текстовому режимі без виклику командного.

Робота редактора **GVIM** вимагає наявності **GTK**. **GTK** (The GIMP ToolKit) – набір інструментів для роботи графічного інтерфейсу користувача.

3.4.2. Редактор Gedit

Gedit – текстовий редактор **DE Gnome**, що працює з різними типами кодувань. Він дуже гнучкий у налаштуванні, як і попередній текстовий редактор підтримує підсвітку синтаксису.

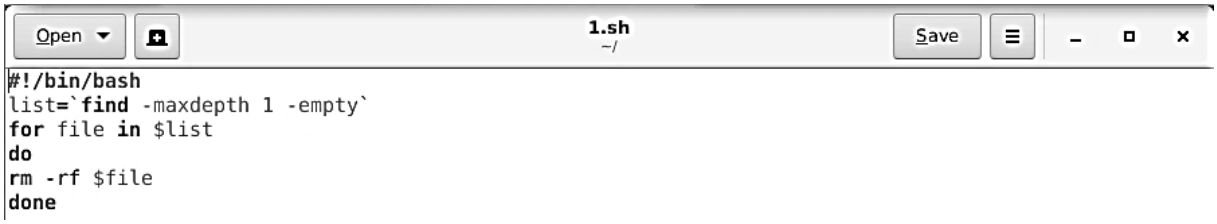

Рис. 3.13. Інтерфейс користувача **Gedit**

Серед переваг Gedit виділяють простоту інтерфейсу, можливість друку з попереднім переглядом, підтримку різних кодувань та автоматичне збереження.

Підтримується синтаксис наступних мов:

– HTML;
– C++;
– Java;
– Python.

Розповсюджується **Gedit** за ліцензією GNU General Public License, розроблений на мовах C та Python, що дає можливість використовувати цей продукт на різних ОС, зокрема на ОС Windows.

3.4.3. Версія редактора Emacs для графічного режиму

Emacs – текстовий редактор представлений в різних дистрибутивах на ядрі Linux. Був створений у 1976 р. Річардом Столлманом и Гаєм Стілом, як набір макросів для редагування тексту. Ними підтримується розмітка HTML та TeX, а також синтаксис мов програмування:

– C;
– C++;
– Java;
– Emacs Lisp.

Доступні два режими роботи консольний та графічний, для запуску графічного необхідно запустити «EmcasUI» (рис. 3.3). Встановлення у дистрибутив **Debian** відбувається за допомогою команди «***apt-get install emacs***». Для роботи потребує наявності **gtk**.

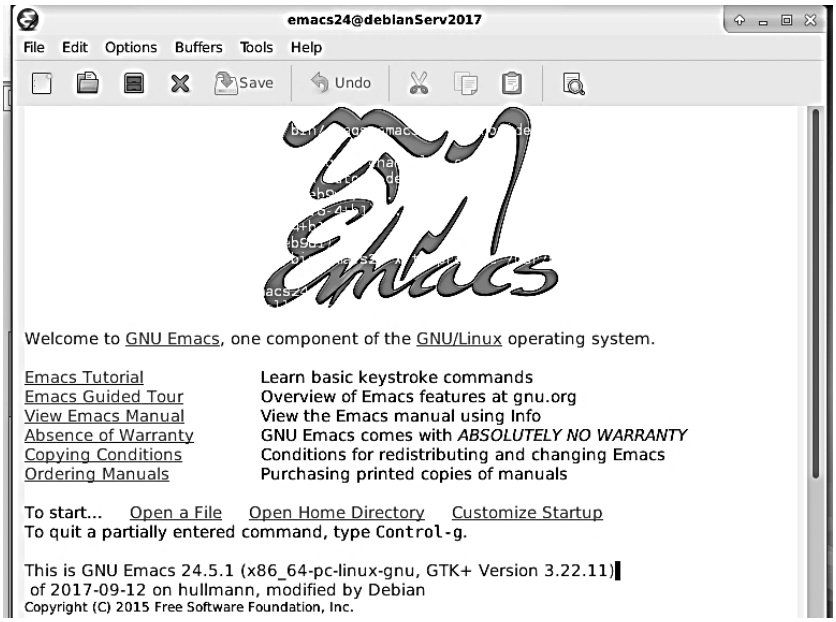

Рис. 3.14. Інтерфейс **EmacsUI**

У деяких випадках необхідне налаштування кодування, наприклад для підключення кирилиці необхідно використовувати koi8-r. Підключення окремого кодування відбувається шляхом редагування файлу конфігурацій ~/.emacs, а саме додаванням наступних рядків:

```
(set-language-environment ‘Cyrillic-KOI8)
(set-terminal-coding-system ‘koi8-r)
(set-keyboard-coding-system ‘koi8-r)
(setq default-buffer-file-coding-system ‘koi8-r)
(prefer-coding-system ‘koi8-r)
(define-coding-system-alias ‘koi8-u ‘koi8-r)
(put-charset-property ‘cyrillic-iso8859-5 ‘preferred-coding-system ‘koi8-r)
(codepage-setup 1251)
(define-coding-system-alias ‘windows-1251 ‘cp1251)
(set-input-mode nil nil ‘We-will-use-eighth-bit-of-input-byte)
(setq-default coding-system-for-read ‘koi8-r)
(set-selection-coding-system ‘koi8-r)
```

3.4.4. Редактор Pluma

Pluma – текстовий редактор, форк від Gedit, що має схожий функціонал, але й має додаткові плагіни та реалізацію функцій drag and drop у файловому менеджері Caja (графічної оболонки Mate). Використовувався у якості базового для дистрибутиву одноплатного комп’ютера Raspberry Pi. За замовчуванням використовує кодування UTF-8.

Як і **Gedit** має доволі простий інтерфейс та вимагає наявності **gtk**. У нього більш зручний інтерфейс, присутня верхня панель з кнопками збереження відкриття файлу, збільшення та зменшення масштабу (рис. 3.15).

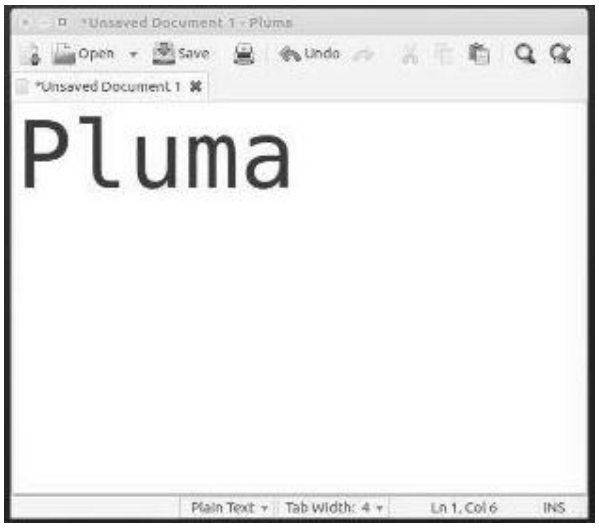

Рис. 3.15. Текстовий редактор **Pluma**

Розділ 4.
ЗАБЕЗПЕЧЕННЯ ПРАВ ДОСТУПУ ДО ФАЙЛІВ ТА КАТАЛОГІВ

4.1. Ідентифікація користувачів у системах UNIX

Операційні системи, які дотримуються традицій UNIX, істотно відрізняються від систем, які наслідують традиції MS-DOS. Різниця між ними полягає у тому, що крім їх багатозадачності, вони ще й багатокористувацькі. Це означає, що комп'ютером можуть одночасно користуватись декілька користувачів. Попри те, що зазвичай комп'ютер має всього одну клавіатуру і монітор, це не заважає спільному користуванню. Наприклад, якщо комп'ютер підключений до локальної мережі або до Інтернету, віддалені користувачі можуть зайти на нього через протокол SSH і виконувати операції. Також вони мають можливість запуску застосунків з графічним інтерфейсом і отримувати зображення на віддаленому дисплеї, оскільки таку можливість підтримує X Window System.

У моделі безпеки UNIX користувач може бути власником файлів або каталогів. У цьому випадку можна керувати доступом до них. За замовчуванням будь-який користувач є власником свого домашнього каталогу. Користувач стає власником файлів та каталогів, які він створив, і може бути примусово призначений адміністратором системи у якості власника інших файлів та каталогів.

Користувачі можуть також належати до певної групи, до якої входять декілька користувачів, і отримати права доступу до файлів і каталогів для членів групи, які визначаються власниками. Крім прав доступу для групи, власник може визначити деякі права доступу для всіх інших користувачів. Для отримання інформації щодо облікового запису певного користувача використовується команда ***id***. Застосування команди без аргументу в результаті виведе інформацію про обліковий запис поточного користувача.

```
debianuser@debianServ2017:~$ id
uid=1000(debianuser) gid=1000(debianuser) groups=1000(debianuser),24(cdrom),25(flo
ppy),29(audio),30(dip),44(video),46(plugdev),108(netdev),113(bluetooth),114(lpadmi
n),118(scanner)
debianuser@debianServ2017:~$
```

Розглянемо результат команди ***id*** детально. Під час створення облікового запису користувача, йому присвоюється число, яке називається **ідентифікатором користувача (User ID)**, або **UID**. Це число, яке відображується як ім'я користувача (логін) виключно заради зручності самого користувача. У наведеному прикладі його іменем є **debianuser**, а UID – **1000**.

Користувачеві призначається **ідентифікатор основної групи (primary Group ID)**, або **GID**, і він додатково може включатися до складу інших груп. У наведеному прикладі основна група називається **debianuser**. Це поведінка системи за замовчуванням, якщо при створенні користувачем у якості основної групи не було вказано жодної іншої групи. GID має значення **1000**. Значення UID та GID для певного користувача можуть співпадати, однак вони ніяк не пов'язані один з одним. В інших дистрибутивах виведення команди ***id*** може трохи відрізнятися.

У якості аргументу команді ***id*** можна задати ім'я користувача, відмінного від поточного. У такому випадку буде виведена інформація щодо облікового запису користувача. Це демонструє наступний приклад.

```
debianuser@debianServ2017:~$ id 1710101
uid=1001(1710101) gid=1010(gr201) groups=1010(gr201),1008(students)
debianuser@debianServ2017:~$
```

У наведеному прикладі виводиться інформація про користувача **1710101** (подібні назви мають облікові записи користувачів студентів ЧНУ ім. Петра Могили де перші 2 цифри означають рік вступу, наступні 3 – номер групи на першому році навчання, останні 2 – порядковий номер студента у списку групи). Таким чином, ми дізналися, що користувач має UID зі значенням **1001** та входить до 2 груп (**gr201** та **students**), основною з яких є група **gr201**, оскільки її ідентифікатор (**1010**) вказаний у якості GID для облікового запису користувача.

Додатково можна скористатися командою ***groups***, яка виводить інформацію, до яких груп входить певний користувач. Застосуємо таку команду для поточного користувача (***debianuser***).

```
debianuser@debianServ2017:~$ groups
debianuser cdrom floppy audio dip video plugdev netdev bluetooth lpadmin scanner
```

А тепер застосуємо команду ***groups*** і для користувача **1710101**:

```
debianuser@debianServ2017:~$ groups 1710101
1710101 : gr201 students
```

Може виникнути слушне питання, де у системи зберігається інформація щодо облікових записів користувачів. Як і багато іншої системної інформації у Linux, вона зберігається у конфігураційних файлах: інформація щодо облікових записів користувачів у файлі **/etc/passwd**, а про групи – у файлі **/etc/group**. Обидва файли мають жорстку структуру, отже розглянемо їх.

Для того, щоб переглянути файл з інформацією щодо облікових записів користувачів, необхідно виконати команду ***less /etc/passwd***:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/fa
lse
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
_apt:x:104:65534::/nonexistent:/bin/false
rtkit:x:105:110:RealtimeKit,,,:/proc:/bin/false
/etc/passwd
```

Це тільки початок файлу **`/etc/passwd`**. Тут ми можемо бачити спеціальні облікові записи, які насправді не належать кінцевим користувачам. Облікові записи можуть належати спеціальним програмам, що працюють у системі. Це, наприклад, записи **daemon**, **bin**, **sys**, **sync** та інші. Перейшовши файлом далі, ми побачимо облікові записи звичайних користувачів, більшістю з яких є студенти ЧНУ ім. Петра Могили.

```
1710109:x:1008:1010::/home/1710109:/bin/bash
1710110:x:1009:1010::/home/1710110:/bin/bash
1710115:x:1010:1010::/home/1710115:/bin/bash
1710116:x:1011:1010::/home/1710116:/bin/bash
1710117:x:1012:1010::/home/1710117:/bin/bash
1718321:x:1013:1010::/home/1718321:/bin/bash
1710118:x:1014:1010::/home/1710118:/bin/bash
1710119:x:1015:1010::/home/1710119:/bin/bash
1710121:x:1016:1010::/home/1710121:/bin/bash
1710123:x:1017:1010::/home/1710123:/bin/bash
1710124:x:1018:1010::/home/1710124:/bin/bash
1710125:x:1019:1010::/home/1710125:/bin/bash
1710126:x:1020:1010::/home/1710126:/bin/bash
1710127:x:1021:1010::/home/1710127:/bin/bash
1710128:x:1022:1010::/home/1710128:/bin/bash
1710129:x:1023:1010::/home/1710129:/bin/bash
1710130:x:1024:1010::/home/1710130:/bin/bash
1710131:x:1025:1010::/home/1710131:/bin/bash
1710202:x:1026:1011::/home/1710202:/bin/bash
1710203:x:1027:1011::/home/1710203:/bin/bash
1710204:x:1028:1011::/home/1710204:/bin/bash
1820201:x:1029:1011::/home/1820201:/bin/bash
1710206:x:1030:1011::/home/1710206:/bin/bash
1710207:x:1031:1011::/home/1710207:/bin/bash
:
```

Першим у файлі ***`/etc/passwd`*** знаходиться рядок, що відповідає обліковому запису користувача **root**, який є адміністратором системи. Цей обліковий запис завжди має значення ідентифікатора користувача, яке

дорівнює **0**. Наступними у файлі будуть системні облікові записи, які є спеціальними записами, що використовують певні існуючі служби для того, щоб отримати доступ до ресурсів системи.

Усі служби, які працюють у фоновому режимі, мають бути зареєстровані в системі Linux, під певним обліковим записом користувача системи. На початкових етапах розвитку Linux, коли ще не приділялося значної уваги безпеці даних, ці служби часто реєструвалися завдяки застосуванню облікового запису користувача **root**. Однак у випадку зламу зловмисником захисту однієї зі служб він отримав би доступ до всієї системи. Тому в сучасних версіях Linux практично всі служби, що працюють у фоновому режимі, мають свій власний обліковий запис користувача і реєструються за його допомогою.

У системі Linux ідентифікатори користувачів з числовими значеннями менше 1000 зарезервовані для системних облікових записів (принаймні у системі Debian, яка розглядається у посібнику, в інших дистрибутивах можливі інші принципи).

Під час детального розгляду файлу ***/etc/passwd***, можна помітити, що в ньому наведено не тільки реєстраційне ім'я та ідентифікатор користувача. Кожен рядок файлу має жорстку структуру, представлену у вигляді полів, що розділяються символом '***:***'. Інформація про обліковий запис певного користувача має 7 полів, що містять наступну інформацію:

1) реєстраційне ім'я користувача;
2) пароль користувача;
3) унікальний ідентифікатор користувача (UID);
4) унікальний ідентифікатор основної групи користувача (GID);
5) необов'язковий коментар, що може бути опущений (зазвичай у ньому міститься певна довідкова інформація про користувача: повне ім'я, номер телефону);
6) абсолютний шлях до домашнього каталогу користувача;
7) командна оболонка, яка застосовується для користувача за замовчуванням при вході у систему.

Звернемо увагу на друге поле, де знаходиться пароль користувача. У вищенаведеному прикладі в кожному рядку задане значення '**x**'. Однак це зовсім не означає, що всі облікові записи користувачів мають однаковий пароль. У більш ранніх версіях у файлі ***/etc/passwd*** дійсно містилися зашифровані версії паролів користувачів. Однак з часом це стало небезпечно, оскільки почало з'являтися програмне забезпечення, яке дозволяло відносно легко розшифровувати засекречені паролі. Тому сьогодні у більшості дистрибутивів Linux, паролі користувачів зберігаються в окремому файлі, що має назву ***/etc/shadow*** та доступ до якого дозволений тільки спеціальним програмам. Однак порожнім залишати поле, яке призначалося для пароля, у файлі ***/etc/passwd*** також не

можна, оскільки у цьому випадку система дійде висновку, що користувачу взагалі дозволений вхід без пароля. У другому полі файлу ***/etc/passwd*** можна побачити символ '***x***' або '*******', хоча може знаходитися практично будь-який символ.

Є очевидним і те, що файл ***/etc/passwd*** є звичайним текстовим файлом, який можна відредагувати будь-яким текстовим редактором. Редагуючи цей файл можна управляти користувачами (додавати, змінювати та видаляти), однак це є доволі небезпечним, оскільки при порушенні структури файлу система не зможе його прочитати, і у цьому випадку всі користувачі включаючи і користувача **root** не зможуть увійти в систему. Тому існує більш безпечний шлях, що полягає у використанні спеціальних команд роботи з користувачами, які доступні до виконання тільки адміністратору. Ці команди розглянемо далі у розділі.

Розглянемо файл ***/etc/shadow***, про який вже було зазначено вище та який має набагато більший контроль над управлінням паролів у системі Linux. Порівняно з файлом ***/etc/passwd*** файл ***/etc/shadow*** захищений набагато краще, оскільки до нього має доступ тільки користувач **root**. Введення в дію системи тіньових паролів дозволило забезпечити більш деталізований контроль над паролями користувачів. Зокрема можливе управління того, як часто користувач має змінювати свій пароль і коли повинен відключитися його обліковий запис, якщо зміна пароля не відбулася.

Зареєструвавшись у системі як адміністратор, переглянемо вміст файлу ***/etc/shadow***.

```
1710131:$6$hjF4cYQ/$tD9wmd/iaQM1VBDlYhJByfdqXxwnbKg4pLDKNfYsHp3s1HOWCMALU1/rh03c
Xs./OGQb9.60vWDBgNRPq.19G.:17778:0:99999:7:::
1710202:$6$mEz6WR8o$Y1WlOdJvXfiEkFuw/JgYPCxeTTEjV7Mf/hG32IaDZlvti89w6CozCfRdQSXg
cZVFMrc1P1Y7tnQDuGKR6wv7T1:17781:0:99999:7:::
1710203:$6$fXKYbmGQ$m881yojW4z3rdM7wzpcbcxcGvs3wHRs8tvfeHxywBK3NqzjHLKk1SCqPc37o
ZdesI1Dy8tYO6EEbrIUwWC0jL.:17781:0:99999:7:::
1710204:$6$nRGuaBpw$76HFUkH/et4CWGr0kgQzKlcQ4GeNZhdMda0nz2Gbu7o99lJT5igrWrQOpkKI
cVf.0s6ZKF7VhOqitUyZ2bCyM0:17788:0:99999:7:::
1820201:$6$nQKRORXz$dGipGz2Bs.lO0YKgRL4/aM/F4GTQwFulyr2QDrnFG5cOskWZWqHQY.sR5TsL
eG1I1DNVQ1OUQrRgnKX8zYF.e.:17781:0:99999:7:::
1710206:$6$dzn9BMk/$Nj3DU7whqgBLPSh///6/uj8KcH3KaOpmBUuPFchOHKi7NyH/TdXroButIxkx
Z9IVv.Bd/xviRVjozqhnDgjRd0:17781:0:99999:7:::
1710207:$6$G/XkV/H7$kSqeeHAl46LdvQTLV3hCYXquG9Hrvs6vaUBMn3Zri5lTuEDFEAU57kFX5RId
79a5BrY3eHe1pTddOzmqQE1dO1:17784:0:99999:7:::
1710208:$6$cyGluyUW$N0yKt.mOoBMb1lBmWo2IOWDk0Yi0vBXANsTYylvQdqY1cYlmfr9FszA5yOzQ
J1acCuUOHkN2zGdSG83rjWgnA/:17781:0:99999:7:::
1710209:$6$vWZcaia5$HTx22QFYrasUaIZKIMKKLeHr7UwBgupzG0745GIeXS7VPoT48c5VWA4wjVfL
BbXltCZSXhkS2kKLXKmqKVae91:17781:0:99999:7:::
1710210:$6$FK7uH2tl$VVy5wRYyJbGC5DILdHaC/NH1HCVqvn5NFt9bhZGCYXXZNvzUmNsLThDJkMb0
CCEwI8OAtDQA92Y6UkJUrry/4/:17781:0:99999:7:::
1710211:$6$qtZYQBo5$t2C0lrTc8qtnXUNTpUHJQbX3p2Fz0GnKZkHmBoOZQt2aa9MybQw6zN30KlXD
xyCmDnfa3NX28fh1qqk6u.dpn.:17781:0:99999:7:::
1710212:$6$NoLpfuCj$rJLivCtgcnDb2.31NDwTcX7Mx88wc6P6jqAp1bu4t5Pc1GlurtUGrOzDnDPu
:
```

У прикладі була представлена тільки певна частина файлу ***/etc/shadow***, однак уже можна зрозуміти його структуру. Подібно до файлу ***/etc/passwd*** кожен рядок файлу ***/etc/shadow*** складається з полів, що розділяються символом ‘***:***’. Таких полів дев’ять:

1) реєстраційна назва, відповідна реєстраційній назві у файлі ***/etc/passwd***;
2) зашифрований пароль;
3) кількість днів, що відраховується з 1 січня 1970 р., що дозволяє визначити дату останньої зміни пароля;
4) мінімальна кількість днів, після закінчення якої може бути змінений пароль;
5) кількість днів до того, як може змінитися пароль;
6) кількість днів до закінчення терміну дії пароля, після чого користувач отримає попередження в зв’язку з необхідністю його змінити;
7) кількість днів після закінчення терміну дії пароля, після чого його обліковий запис буде відключено;
8) дата (збережена як кількість днів з 1 січня 1970 р.), після якої обліковий запис користувача було відключено;
9) поле, зарезервоване для подальшого використання.

Відомості про групи зберігаються у файлі ***/etc/group***, кожен рядок якого також жорстко структурований. Аналогічно до двох інших файлів, розглянутих вище, рядок розділяється на поля за допомогою символу ‘***:***’. Переглянемо зазначений файл:

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
uucp:x:10:
man:x:12:
proxy:x:13:
kmem:x:15:
dialout:x:20:
fax:x:21:
voice:x:22:
cdrom:x:24:debianuser
floppy:x:25:debianuser
tape:x:26:
sudo:x:27:
audio:x:29:pulse,debianuser
dip:x:30:debianuser
:
```

Пояснимо структуру файлу ***/etc/group*** більш детально. Подібно до ідентифікаторів користувачів, ідентифікатори груп присвоюються за допомогою спеціального формату. У цьому файлі використовуються наступні чотири поля:

1) назва групи;
2) пароль групи;
3) унікальний ідентифікатор групи (GID);
4) список облікових записів користувача, які належать до групи, якщо вона для них не є основною.

Групам, які призначені для системних облікових записів, присвоюються ідентифікатори груп з числовим значенням менше **1000** (принаймні у дистрибутиві Debian), а групам для користувачів – ідентифікатори груп, що починаються з **1000**.

Під час створення певної групи їй можна задати пароль, який дозволить користувачу, що не є членом групи стати ним на певний час. Хоча схожа можливість застосовується не дуже часто, але все ж вона існує.

Аналогічно, як і у випадку з файлом ***/etc/passwd***, не рекомендується створювати, змінювати та видаляти групи шляхом редагування файлу ***/etc/group***. Краще для цього застосувати спеціальні команди, що будуть розглянуті далі.

4.2. Команди модифікації користувачів та груп

4.2.1. Додавання нового користувача

Основним інструментом для додавання нового користувача у системі Linux є команда ***useradd***, яка представляє простий спосіб створення нового облікового запису користувача і його домашнього робочого каталогу. У команді поєднуються системні значення за замовчуванням і ключі команди для визначення нового облікового запису належним чином. Для перегляду системних значень за замовчуванням потрібно застосувати команду ***useradd*** з ключем ***-D***, як показано в наступному прикладі. Зауважимо, що ця команда, як і інші, не може бути виконана звичайним користувачем, для її виконання потрібні права адміністратора.

```
root@debianServ2017:~# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
root@debianServ2017:~#
```

За необхідності використати інші значення або операції при створенні нового користувача потрібно застосувати відповідні ключі команди ***useradd***, які описані у табл. 4.1.

Табл. 4.1. Ключі команди `useradd`

Ключ	Опис
-c <коментар>	Додати текст до поля коментаря нового користувача
-d <директорія>	Задати для домашнього каталогу користувача інше ім'я, що відрізняється від реєстраційного імені
-e <дата>	Задати дату в форматі YYYY-MM-DD, коли закінчується термін дії облікового запису
-f <кількість_днів>	Задати кількість днів після закінчення терміну дії пароля, коли обліковий запис буде відключений Спеціальні значення: – 0 – обліковий запис має бути відключений одразу після закінчення терміну дії пароля; – -1 – скасування відключення
-g <група>	Задати основну групу користувача (її GID або ім'я)
-G <групи…>	Задати одну або декілька додаткових груп, до яких належатиме користувач
-m	Створити домашній каталог користувача
-M	Не створювати домашній каталог користувача, якщо налаштування за замовчуванням передбачає його створення
-n	Створити нову групу з використанням імені, що збігається з реєстраційним іменем
-r	Створити системний обліковий запис
-p <пароль>	Задати застосовуваний за замовчуванням пароль для облікового запису
-s <оболонка>	Задати застосовуваний за замовчуванням командний інтерпретатор входу в систему
-u <ідентифікатор_користувача>	Задати унікальний ідентифікатор користувача для облікового запису.

Таким чином, завдання зміни значень за замовчуванням є зовсім не складним.

Наприклад, створимо нового користувача з іменем ***user1*** та додамо його у групу ***students*** (але вкажемо її у якості основної), створимо його домашній каталог та вкажемо у якості оболонки за замовчуванням ***bash***. Після цього переглянемо інформацію щодо створеного облікового запису в файлі ***/etc/passwd*** та перевіримо, чи був створений його домашній каталог.

```
root@debianServ2017:~# useradd -g students -m -s /bin/bash user1
root@debianServ2017:~# grep user1 /etc/passwd
user1:x:1219:1008::/home/user1:/bin/bash
root@debianServ2017:~# ls -l /home | grep user1
drwxr-xr-x  2 user1      students      4096 Oct 29 16:34 user1
root@debianServ2017:~#
```

Отже, ми переконалися, що новий обліковий запис користувача ***user*** дійсно був створений.

4.2.2. Зміна поточного користувача

У системі Linux передбачені різні команди, які змінюють облікові записи користувачів тим чи іншим чином. Вони наведені у табл. 4.2.

Табл. 4.2. Команди зміни облікових записів користувачів

Команда	Опис
usermod	Команда редагування полів облікового запису користувача, яка дозволяє визначати приналежність до первинної та вторинної групи
passwd	Команда зміни пароля для існуючого користувача
chpasswd	Команда читання файлу, в якому у вигляді окремих пар задані реєстраційні імена і паролі та оновлення паролів
chage	Команда зміни дати закінчення терміну дії пароля
chfn	Команда зміни вмісту коментаря до облікового запису користувача
chsh	Команда зміни заданої за замовчуванням командної оболонки для облікового запису користувача.

Команда ***usermod*** має найбільш широку область дії серед усіх команд зміни облікових записів користувачів, оскільки надає можливість вносити зміни в більшість полів файлу **/etc/passwd**. Для цього достатньо скористатися відповідними ключами цієї команди, які в основному збігаються з параметрами команди ***useradd***, однак існують і додаткові параметри, які можуть виявитися необхідними, наприклад:

– ***-p*** – змінити пароль облікового запису;

– ***-l*** – змінити реєстраційне ім’я облікового запису користувача;

– ***-L*** – заблокувати обліковий запис для того, щоб користувач не зміг увійти в систему;

– ***-U*** – розблокувати обліковий запис для того, щоб користувач зміг увійти в систему.

Як приклад змінимо основну групу користувача ***user1*** зі ***students*** на ***gr201***, а також додамо його до груп ***gr202***, ***gr208*** та ***gr209***:

```
root@debianServ2017:~# usermod -g gr201 -G gr202,gr208,gr209 user1
```

А тепер переглянемо інформацію щодо облікового запису, спочатку виконавши команду ***id***, а потім знайшовши відповідний запис у файлі ***/etc/passwd***:

```
root@debianServ2017:~# id user1
uid=1219(user1) gid=1010(gr201) groups=1010(gr201),1011(gr202),1012(gr208),1013(
gr209)
root@debianServ2017:~# cat /etc/passwd | grep user1
user1:x:1219:1010::/home/user1:/bin/bash
```

Як бачимо, основна група дійсно була змінена, а користувач був доданий до іншої групи, при цьому група ***students***, до якої користувач входив раніше, була видалена з його облікового запису.

4.2.3. Видалення користувача

Якщо коли-небудь буде потрібно видалити користувача з системи, то достатньо застосувати команду ***userdel***, яка за замовчуванням видаляє тільки відомості про користувача з файлу ***/etc/passwd***. Якщо використовувати ключ ***-r***, то команда видалить домашній каталог користувача разом з каталогом ***mail***, якщо він існував. Однак в системі можуть ще залишатися інші файли, які належать видаленому обліковому запису користувача, що у певних випадках призведе до виникнення проблем.

Тепер видалимо обліковий запис користувача ***user1***, ніби його ніколи не існувало, видаливши також і домашній каталог:

```
root@debianServ2017:~# userdel -r user1
userdel: user1 mail spool (/var/mail/user1) not found
root@debianServ2017:~# id user1
id: 'user1': no such user
root@debianServ2017:~# cat /etc/passwd | grep user1
root@debianServ2017:~# ls /home | grep user1
```

Застосувавши у наведеному прикладі спочатку команду ***userdel*** з ключем ***-r***, ми видалили користувача ***user1*** разом з домашнім каталогом. У результаті команда вивела попередження, що у системі немає каталогу, який асоціюється з електронною поштою користувача.

Подальші команди у прикладі демонструють, що після видалення користувача ні його облікового запису, ні домашнього каталогу не залишилося.

4.2.4. Команди маніпулювання групами

Для створення нових груп у системі застосовується команда ***groupadd***. Наголосимо, що ця команда не надає можливості додавати облікові записи користувачів до груп, а тільки створює нову, яка після створення порожня. Для додавання нових користувачів до групи призначена команда ***usermod*** з відповідними ключами ***-g*** при зазначенні основної групи, та ***-G*** для інших груп.

Змінити інформацію про групу дозволяє команда ***groupmod***. Як показує файл ***/etc/group***, обсяг інформації, що підлягає зміні, невеликий. Команду можна використати такими ключами як:

1) ***-g*** – змінити ідентифікатор групи;

2) ***-n*** – змінити назву групи.

При зміні назви групи, залишаються незмінними її ідентифікатор та список членів. Оскільки всі визначення прав доступу засновані на ідентифікаторі групи, допускається змінювати її назву настільки часто, наскільки це потрібно, не здійснюючи негативного впливу на безпеку файлу.

Для видалення відповідної групи потрібно використати команду ***groupdel***.

4.2.5. Команди зміни ідентичності користувача

Іноді може виникнути необхідність придбати ідентичність іншого користувача. Це частіше всього відбувається для виконання певних адміністративних завдань, що потребують повноваження суперкористувача.

У такому разі може виникнути необхідність зареєструватися під іменем іншого звичайного користувача, наприклад, для того, щоб перевірити налаштування облікового запису. Існує три способи для придбання ідентичності іншого користувача:

- вийти з системи і ввійти знову з обліковими даними іншого користувача;
- виконати команду ***su***;
- виконати команду ***sudo***.

Команда ***su*** використовується для запуску нового сеансу роботи з командною оболонкою від імені іншого користувача поверх поточного сеансу. Команда має наступний синтаксис:

su [-l] <користувач>

Під час застосування ключа ***-l*** запущена командна оболонка стане оболонкою входу для вказаного користувача, тобто поточним робочим каталогом стане його домашній каталог.

Після введення команди буде запропоновано ввести відповідний пароль вказаного користувача. Після введення правильного пароля з’явиться нове запрошення до вводу, яке буде показувати, що робота у системі вже ведеться від імені іншого користувача.

Для наступного прикладу запустимо сеанс системи від імені користувача ***root***, поверх поточного сеансу користувача ***debianuser***.

```
debianuser@debianServ2017:~$ su root
Password:
root@debianServ2017:/home/debianuser# head /etc/shadow
root:$6$Le68KZ0g$MtlBg2wOKcDUq/LPY6idq3OMDUMI4jNGdA21fi9CZO/W4Mvvvgz1mF7dzHz138c
tIhho1E1ac4htFlaHPlzMh/:17833:0:99999:7:::
daemon:*:17409:0:99999:7:::
bin:*:17409:0:99999:7:::
sys:*:17409:0:99999:7:::
sync:*:17409:0:99999:7:::
games:*:17409:0:99999:7:::
man:*:17409:0:99999:7:::
lp:*:17409:0:99999:7:::
mail:*:17409:0:99999:7:::
news:*:17409:0:99999:7:::
root@debianServ2017:/home/debianuser# exit
exit
debianuser@debianServ2017:~$
```

У наведеному вище прикладі спочатку з сеансу звичайного користувача ***debianuser*** здійснюється вхід до системи від імені суперкористувача ***root***. Система вимагатиме введення пароля, після цього відбудеться вхід у командну оболонку користувача ***root***, однак поточний робочий каталог не зміниться (ним залишиться домашній каталог користувача ***debianuser***).

Під іменем суперкористувача виконується команда, що виводить перші 10 рядків файлу ***/etc/shadow***, доступ до якого, як ми знаємо, має тільки суперкористувач.

Потім ми виконуємо команду ***exit***, яка у цьому випадку не здійснює вихід із системи, а тільки повертає у зовнішню командну оболонку користувача ***debianuser***.

У наступному прикладі також застосуємо команду ***su***, але з ключем ***-l***:

```
debianuser@debianServ2017:~$ su -l root
Password:
root@debianServ2017:~# pwd
/root
root@debianServ2017:~# exit
logout
debianuser@debianServ2017:~$
```

Як можна побачити, що під час запуску сеансу від імені суперкористувача змінився і поточний робочий каталог, ***/root***, тобто домашній каталог користувача ***root***.

Також з командою ***su*** можна застосувати ключ ***-c***, що передбачає виконання єдиної команди під іменем іншого користувача. Необхідну команду потрібно зазначити у якості значення цього ключа, взявши її в одинарні лапки для запобігання додаткової інтерпретації. Наприклад, виведемо перші 10 рядків користувача ***/etc/shadow***, звісно виконавши це під іменем користувача ***root***:

```
debianuser@debianServ2017:~$ su -c 'head /etc/shadow' root
Password:
root:$6$Le68KZOg$MtlBg2wOKcDUq/LPY6idq3OMDUMI4jNGdA21fi9CZO/W4Mvvvgz1mF7dzHz138c
tIhho1E1ac4htFlaHPlzMh/:17833:0:99999:7:::
daemon:*:17409:0:99999:7:::
bin:*:17409:0:99999:7:::
sys:*:17409:0:99999:7:::
sync:*:17409:0:99999:7:::
games:*:17409:0:99999:7:::
man:*:17409:0:99999:7:::
lp:*:17409:0:99999:7:::
mail:*:17409:0:99999:7:::
news:*:17409:0:99999:7:::
debianuser@debianServ2017:~$
```

Таким чином, під іменем суперкористувача було виконано тільки одну команду.

Багато в чому до команди ***su*** з ключем ***-s*** схожа команда ***sudo***, однак вона має певні важливі особливості. По-перше, порядок використання ***sudo*** звичайними користувачами завжди визначає адміністратор. Тим самим, він може дозволити або заборонити виконання команд. По-друге, команда ***sudo*** не вимагає введення пароля суперкористувача, а для аутентифікації користувач має ввести свій власний пароль, тобто підтвердити свою ідентичність.

Наведемо приклад, у якому спробуємо вивести 10 перших рядків файлу ***/etc/shadow***.

```
debianuser@debianServ2017:~$ sudo 'head /etc/shadow'
[sudo] password for debianuser:
debianuser is not in the sudoers file.  This incident will be reported.
```

Отже можна зробити висновок, що у користувача ***debianuser*** немає ніяких привілеїв для здійснення вищевказаної команди від імені суперкористувача за допомогою команди ***sudo***.

Для того, щоб побачити, що можна виконати за допомогою команди ***sudo*** поточному користувачу, необхідно скористатися командою з ключем ***-l***, як показано у наступному прикладі:

```
debianuser@debianServ2017:~$ sudo -l
[sudo] password for debianuser:
Sorry, user debianuser may not run sudo on debianServ2017.
debianuser@debianServ2017:~$
```

Таким чином, ми дізналися, що користувач ***debianuser*** не може виконувати команду ***sudo*** без відповідного дозволу, який надається адміністратором системи.

4.3. Права доступу до файлів і каталогів

У цьому розділі розглянемо, які права доступу до файлів і каталогів визначаються у системах, що є нащадками UNIX, як їх можна переглянути та змінити. Існують 3 різні права доступу, які можуть бути надані певній категорії користувачів системи (нагадаємо, що такими категоріями є

власник файлу або каталогу; група, до якої належить власник; усі інші користувачі). Це такі права як:

1) на читання;
2) на запис;
3) на виконання.

Для того, щоб переглянути поточні права на певний елемент файлової системи, необхідно скористатися вже знайомою нам командою ***ls*** з ключем ***-l***, яка виводить найбільш детальну інформацію щодо вмісту певного каталогу. До неї входять повідомлення про права доступу, що представляється у першому стовпчику виведення команди. Наприклад:

```
debianuser@debianServ2017:~$ ls -l /var
total 48
drwxr-xr-x  2 root root   4096 Oct 30 00:09 backups
drwxr-xr-x 16 root root   4096 Jan  5  2018 cache
drwxr-xr-x 48 root root   4096 Oct 12 11:14 lib
drwxrwsr-x  2 root staff  4096 Jul 13  2017 local
lrwxrwxrwx  1 root root      9 Aug 31  2017 lock -> /run/lock
drwxr-xr-x  9 root root   4096 Oct 30 00:12 log
drwxrwsr-x  2 root mail   4096 Aug 31  2017 mail
drwxr-xr-x  2 root root   4096 Aug 31  2017 opt
lrwxrwxrwx  1 root root      4 Aug 31  2017 run -> /run
drwxr-xr-x  7 root root   4096 Jan  5  2018 spool
drwxrwxrwt 100 root root  16384 Oct 30 10:55 tmp
debianuser@debianServ2017:~$
```

Інформація щодо прав доступу представляє 10 символів. Схематично вона показана на рис. 4.1.

Рис. 4.1. Інформація про права доступу файлу

Перший символ не відноситься до прав доступу, він показує тип елемента. Значення першого символу та їх опис представлені у табл. 4.3.

Табл. 4.3. Значення першого символу та їх опис

Значення	Опис
-	звичайний файл
d	каталог
l	символічне посилання
c	файл символьного пристрою (термінал, модем)
b	файл блочного пристрою (жорсткий диск, cd-rom і т. п.).

Усі інші 9 символів представляють інформацію про права доступу. Їх можна розподілити на 3 групи по 3 символи, кожна з яких представляє інформацію щодо права доступу для різних категорій:

– символи **2–4** – права для власника;

– символи **5–7** – права для користувачів, що входять до основної групи власника;

– символи **8–10** – права для всіх інших користувачів.

Права доступу представляються відповідними символами та завжди йдуть в одному й тому самому порядку:

– ***r*** – право на читання;

– ***w*** – право на запис;

– ***x*** – право на виконання.

За наявності відповідного права зображена відповідна літера, якщо ні – то символ ‘**-**’.

Але одне й те саме право у контексті файлів та каталогів може означати абсолютно різні привілеї, які детально описані у табл. 4.4.

Табл. 4.4. Привілеї, які дають права доступу відносно файлів та каталогів

Право	Привілеї для файлів	Привілеї для каталогів
r	Відкриття та читання вмісту файлу	Читання вмісту каталогу
w	Запис даних у файл або видалення інформації з нього (право не дає привілеїв перейменування та видалення файлів, потрібне відповідне право для каталогу, в якому файл розташовується)	Створення, видалення й перейменовування файлів, що розташовуються у каталозі
x	Інтерпретація файлу як програми і її виконання (файли, які є програмами на мовах сценаріїв, мають бути доступними для читання, інакше вони не будуть виконуватися)	Дозвіл переходу в каталог (з використанням команди ***cd***).

Зазначимо, що для символічних посилань завжди будуть відображуватись максимальні права (***rwxrwxrwx***), однак це не відображує дійсні права доступу. У випадку символічних посилань фактичні права доступу до файлу визначаються правами доступу самого файлу, на який вказує це символічне посилання.

Наведемо декілька прикладів встановлених прав доступу для різних елементів файлової системи у табл. 4.5.

Табл. 4.5. Приклади встановлених прав доступу до файлів

Права доступу	Опис
-rwx------	Звичайний файл, на який повні права має власник. Інший користувач при цьому не має ніяких прав доступу до нього
-rw-------	Звичайний файл, який доступний власнику для читання і запису. Ніякий інший користувач при цьому не має прав доступу до нього
-rw-r--r--	Звичайний файл, який доступний власнику для читання і запису. Право на читання файлу мають члени основної групи власника та всі інші користувачі, але інших прав доступу на цей файл у них немає
-rwxr-xr-x	Звичайний файл, право на читання і виконання якого має власник, члени основної групи власника та всі інші користувачі. Власнику файлу він доступний також для запису
-rw-rw-----	Звичайний файл, на який мають право читання та запису власник файлу та члени його основної групи. Усі інші не мають ніяких прав доступу до нього
lrwxrwxrwx	Символічне посилання. Це означає, що всі права доступу є недійсними, а фактичні права на нього визначаються правами доступу файлу, на який вказує символічне посилання
drwxrwx---	Каталог, у який можуть входити, а також створювати, перейменовувати і видаляти файли власник та члени його основної групи. Усі інші не мають ніяких прав доступу до нього
drwxr-x---	Каталог, у який можуть входити власник та члени його основної групи. Власник може створювати, перейменовувати і видаляти файли всередині каталогу. Всі інші не мають ніяких прав доступу до нього.

4.3.1. Зміна режиму доступу до файлу

Для зміни прав доступу до файлу або каталогу призначена команда ***chmod***. Запам'ятайте, що права доступу до файлу або каталогу може змінити тільки його власник. Ця команда має наступний синтаксис:

chmod <права_доступу> <елементи…>

Під час виконання команди потрібно обов'язково задати хоча б один елемент, ними може бути файл або каталог, однак можна змінити права доступу і більш ніж одному елементу одразу. Права доступу можна задати 2 способами: за допомогою вісімкового та символічного представлення.

Вісімкове представлення

Використання вісімкової форми запису права доступу визначаються вісімковими числами. Оскільки кожна цифра у вісімковому числі

визначається трьома двійковими розрядами, вона відображається в схемі зберігання режиму доступу до файлу (табл. 4.6).

Табл. 4.6. Режими доступу до файлу в двійковому та вісімковому представленні

Вісімкове	Двійкове	Режим доступу	Опис
0	***000***	***---***	відсутність будь-яких прав
1	***001***	***--x***	тільки виконання
2	***010***	***-w-***	тільки запис
3	***011***	***-wx***	запис і виконання
4	***100***	***r--***	тільки читання
5	***101***	***r-x***	читання і виконання
6	***110***	***rw-***	читання і запис
7	***111***	***rwx***	читання, запис і виконання.

Права доступу за допомогою вісімкового представлення визначаються трьома цифрами від ***0*** до ***7***, перша з яких представляє права для власника, друга – для членів його основної групи, а третя – для всіх інших користувачів.

У наступному прикладі створимо новий файл з назвою ***file1.txt*** та встановимо на нього наступні права доступу: власник може читати та писати, члени основної групи – тільки читати, а всі інші не мають на нього ніяких прав. Ці права у вісімковому представленні будуть закодовані числом ***640***.

```
debianuser@debianServ2017:~$ touch file1.txt
debianuser@debianServ2017:~$ chmod 640 file1.txt
debianuser@debianServ2017:~$ ls -l file1.txt
-rw-r----- 1 debianuser debianuser 0 Oct 31 09:06 file1.txt
```

Наведемо інший приклад. Створимо новий каталог ***dir1*** та встановимо для власника повні права доступу, членам основної групи власника права, що дозволяють входити до каталогу та розглядати його вміст, усім іншим тільки розглядати вміст каталогу без можливості входу до нього (як це не дивно, але можна продивитися вміст каталогу за допомогою команди ***ls***, вказавши його назву в якості аргументу, знаходячись при цьому абсолютно в іншому каталозі). Описані права доступу можна представити за допомогою числа ***754***.

```
debianuser@debianServ2017:~$ chmod 754 dir1
debianuser@debianServ2017:~$ ls -l dir1
-rwxr-xr-- 1 debianuser debianuser 0 Oct 31 09:16 dir1
```

Таким чином, з двох наведених вище прикладів ми побачили, що права доступу після їх встановлення командою ***chmod*** дійсно відповідають їх вісімковому кодуванню.

Символічне представлення

Іншим способом задати права командою ***chmod*** є символічне представлення, яке передбачає символічну форму запису, яка має наступний вигляд:

<категорія_користувачів><операція><права>,

де: ***<категорія_користувачів>=<u, g, o, a>***, ***<операція>=<+, -, =>***, ***<права>=<r, w, x>***.

Стосовно категорії користувачів, для яких встановлюються права доступу, використовуються наступні символи:

– ***u*** – власник файлу або каталогу (**user**);

– ***g*** – члени основної групи власника файлу або каталогу (**group**);

– ***o*** – усі інші користувачі (**other**);

– ***a*** – усі користувачі системи: комбінація всіх трьох попередніх символів (**all**).

Для зазначення операцій використовуються наступні символи:

- ***+*** – додати задані права;
- ***–*** – відняти задані права;
- ***=*** – встановити задані права (різниця між знаком ***+*** полягає у тому, що задані права встановлюються, а всі інші віднімаються).

На останок вказуються права доступу звичними літерами: ***r***, ***w*** та ***x***. Наведемо деякі приклади:

– ***u+x*** – додати власнику право на виконання;

– ***g-w*** – відібрати у групи право на читання.

При зазначенні всіх користувачів, літеру ***a*** також можна опустити:

+x – додати право на виконання всім користувачам у системі (власнику, членам основної групи та іншим користувачам).

Різні категорії та права доступу можна об’єднувати, вказавши їх разом. Наприклад:

– ***go=rw*** – встановити для основної групи та інших користувачів право на читання і запис. Але якщо вони мали право на виконання, то воно відбирається;

– ***o-rw*** – відібрати право на читання і запис в усіх інших користувачів, крім власника і групи.

Нарешті, можна виконати одразу декілька операцій, перерахувавши їх через кому. Наприклад:

– ***u+x,go=rx*** – додати право на виконання для власника і водночас встановити права на читання і виконання для членів його основної групи та інших користувачів.

Порівняно з вісімковим представленням перевага символічного представлення прав доступу полягає у дозволі встановлення певного права, ніяк не впливаючи на інші.

4.3.2. Права доступу до файлів і каталогів за замовчуванням

Під час створення нових файлів та каталогів у своєму домашньому каталозі їм надаються певні права доступу. Наприклад, створимо новий файл та новий каталог у домашньому каталозі користувача **debianuser** та подивимось які права доступу були надані для них.

```
debianuser@debianServ2017:~$ touch file1
debianuser@debianServ2017:~$ mkdir dir1
debianuser@debianServ2017:~$ ls -l
total 4
drwxr-xr-x 2 debianuser debianuser 4096 Nov  2 20:43 dir1
-rw-r--r-- 1 debianuser debianuser    0 Nov  2 20:43 file1
```

Таким чином, права доступу на новий файл та каталог є наступними:

– для файлу: власник може читати та писати, члени основної групи власника – тільки читати, інші користувачі – так само тільки читати;

– для каталогу: власник може входити, читати його вміст, а також створювати, видаляти та перейменовувати файли; члени основної його групи – входити та читати вміст каталогу, однак не можуть створювати, видаляти та перейменовувати файли; всі інші користувачі – так само тільки входити та читати його вміст.

Може виникнути слушне запитання, чому для новоствореного файлу та каталогу були надані саме такі права доступу? Відповідь полягає у тому, що кожному новоствореному файлу та каталогу надаються певні права за замовчуванням. А визначити та змінити їх дає команда ***umask***.

Спочатку визначимо поточні права, що надаються за замовчуванням. Це можна здійснити за допомогою виконання команди ***umask*** без аргументу.

```
debianuser@debianServ2017:~$ umask
0022
```

З огляду на це команда ***umask*** видала значення ***0022***. Що ж означають ці цифри? Для того, щоб було зрозуміло, проводимо експеримент. Виконаємо команду ***umask***, задавши їй як аргумент значення ***0000***, а потім створимо ще один новий файл та каталог, після чого переглянемо права доступу, які були йому надані.

```
debianuser@debianServ2017:~$ umask 0000
debianuser@debianServ2017:~$ touch file2
debianuser@debianServ2017:~$ mkdir dir2
debianuser@debianServ2017:~$ ls -l
total 8
drwxr-xr-x 2 debianuser debianuser 4096 Nov  2 20:43 dir1
drwxrwxrwx 2 debianuser debianuser 4096 Nov  2 21:19 dir2
-rw-r--r-- 1 debianuser debianuser    0 Nov  2 20:43 file1
-rw-rw-rw- 1 debianuser debianuser    0 Nov  2 21:19 file2
```

Отже, нові файл та каталог після встановлення нової маски мають наступні права доступу: для каталогу всі користувачі мають повні права доступу, а щодо файлу, то користувачі можуть читати та писати, ніхто не має можливості його виконувати.

Встановивши маску ***0000***, ми тим самим фактично вимкнули її. Але для створених після її вимкнення файлу та каталогу, були надані максимальні права доступу, які можна надати за замовчуванням для файлу та каталогу відповідно. Таким чином, для каталогу максимальними правами доступу за замовчуванням є сукупність прав ***rwx*** для всіх користувачів, що відповідає вісімковому числу ***777***, а для файлу відповідна сукупність прав ***rw-***, що відповідає вісімковому числу ***666***. Таким чином, для файлу за замовчуванням не буде надано право на виконання. Зроблено це з точки зору безпеки, оскільки не кожен файл є виконуваною програмою. Написавши файл, що представляє програму, користувач завжди має власноруч надати право на його виконання.

Тепер нарешті розберемося, які права за замовчуванням надає маска. Для цього ми знову маємо повернутися до вісімкового представлення прав доступу. Нагадаємо, що початкова маска мала значення ***0022***. За цією маскою були надані права доступу за замовчуванням для файлу ***file1*** та каталогу ***dir1***. Представимо таблицю 4.7, у якій зазначимо максимальні права за замовчуванням, маску та результат (поточні права доступу). Зокрема поки не будемо розглядати першу цифру (***0***), далі буде описано, що вона означає.

Табл. 4.7. Вплив маски на призначення прав за замовчуванням для файлу та каталогу

Для каталогу	
максимальні права за замовчуванням	rwx rwx rwx
маска	000 010 010
поточні права	rwx r-x r-x
Для файлу	
максимальні права за замовчуванням	rw- rw- rw-
маска	000 010 010
поточні права.	rw- r-- r--.

З таблиці досить легко побачити, що біт, який відповідає тій позиції, де в масці стоїть 1, був скинутий, – в цьому випадку право на запис членів основної групи власника та всіх інших користувачів. Тепер зрозуміло, що робить маска: за її допомогою ми можемо задати права, які будуть заборонені за замовчуванням для новостворених файлів та каталогів.

Задаючи нову маску, ми вказуємо інверсне значення, тобто ті права, які хочемо заборонити певній категорії користувачів. Тут навпаки цифра ***0***,

буде означати повний доступ, а цифра **7** – відсутність усіх прав за замовчуванням.

4.3.3. Розширені права доступу

Зазвичай права доступу у вісімковому представленні позначаються трьома цифрами, кожна з яких означає права для певної категорії користувачів, однак правильно їх представляти чотирма цифрами. Четверта вісімкова цифра найстаршого розряду позначає спеціальні права доступу.

Існує 3 спеціальних права доступу, кожне з яких представляється відповідним бітом. На рис. 4.2. представлені біти четвертого вісімкового розряду.

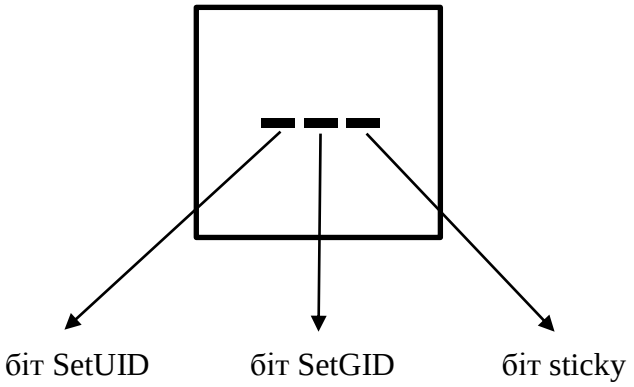

Рис. 4.2. Біти, що визначають спеціальні права доступу

Сукупність цих трьох бітів, як і у випадку прав доступу на читання, запис та виконання, можна представити однією вісімковою цифрою від ***0*** до ***7***. Однак у деяких випадках встановлений тільки один з трьох бітів, оскільки їх сукупність практично не дає ніяких ефектів. Так, встановлений біт **SetUID** дає цифру ***4*** у четвертому вісімковому розряді при кодуванні прав доступу, **SetGID** – цифру ***2***, а **sticky** – цифру ***1***.

Якщо встановити найстарший біт **SetUID** для виконуваного файлу (програми), то у якості ефективного ідентифікатора користувача для процесу встановлюється не його ідентифікатор, який фактично запустив програму, а ідентифікатор власника цієї програми. Найчастіше він встановлюється для програм, власником яких є привілейований користувач. Тобто коли звичайний користувач запускає виконуваний файл, для якого встановлений біт **SetUID**, ця програма виконується з привілеями суперкористувача. Однак через проблеми безпеки кількість таких програм у системі має бути зведена до мінімуму.

Для встановлення біту **SetUID** можна скористатися вісімковим або символічним представленням прав доступу в команді ***chmod***: у вісімковому представленні спеціальне право встановлюється за допомогою

значення 4 у четвертому розряді, а у символічному за допомогою спеціального запису ***u+s***.

У наступному прикладі створимо два файли ***prog1*** та ***prog2*** та встановимо для них біт **SetUID** за допомогою вісімкового представлення у першому випадку і символічного представлення у другому. Потім проаналізуємо як представляються права доступу з встановленим бітом.

```
debianuser@debianServ2017:~$ touch prog1 prog2
debianuser@debianServ2017:~$ ls -l
total 0
-rw-r--r-- 1 debianuser debianuser 0 Nov  3 10:59 prog1
-rw-r--r-- 1 debianuser debianuser 0 Nov  3 10:59 prog2
debianuser@debianServ2017:~$ chmod 4644 prog1
debianuser@debianServ2017:~$ chmod u+s prog2
debianuser@debianServ2017:~$ ls -l
total 0
-rwSr--r-- 1 debianuser debianuser 0 Nov  3 10:59 prog1
-rwSr--r-- 1 debianuser debianuser 0 Nov  3 10:59 prog2
```

При виведені інформації про файли, на які встановлений біт **SetUID**, у правах доступу як символ, призначеного для позначення дозволу виконання власником файлу, відображається літера ***S***.

Але другий, середній біт, що має назву **SetGID**, дозволяє підмінити основну групу власника. Його можна встановити як для файлу так і для каталогу. У випадку встановлення для файлу цей біт аналогічно до біту **SetUID** для власника файлу встановлює ефективний ідентифікатор основної групи для процесу, вибираючи замість ідентифікатора основної групи реального користувача ідентифікатор основної групи власника файлу. Під час встановлення цього біту для каталогу, новостворювані файли у ньому будуть належати основній групі власника каталогу, а не групі власника файлу, який його створив.

Для встановлення цього біту за допомогою вісімкового представлення у найстаршому розряді встановлюється значення 2, тоді як у символичному представленні необхідно скористатися спеціальним записом ***g+s***.

У наступному прикладі створимо новий файл та новий каталог, і встановимо для них біт **SetGID** двома різними способами, потім подивимося на права доступу до них.

```
debianuser@debianServ2017:~$ touch prog3
debianuser@debianServ2017:~$ mkdir dir1
debianuser@debianServ2017:~$ chmod g+s prog3
debianuser@debianServ2017:~$ chmod 2755 dir1
debianuser@debianServ2017:~$ ls -l
total 4
drwxr-sr-x 2 debianuser debianuser 4096 Nov  3 12:00 dir1
-rwSr--r-- 1 debianuser debianuser    0 Nov  3 10:59 prog1
-rwSr--r-- 1 debianuser debianuser    0 Nov  3 10:59 prog2
-rw-r-Sr-- 1 debianuser debianuser    0 Nov  3 12:00 prog3
```

У прикладі можна побачити, встановлення біту в інформацію щодо прав доступу на місці символу, який відповідає за право виконання для

основної групи, встановлена літера **s**, але для каталогу вона маленька, а для файлу – велика.

Наостанок розглянемо молодший біт, що має назву **sticky**. У перших версіях UNIX він дозволяв позначити виконуваний файл таким, який неможливо видалити. Наразі для файлів система Linux ігнорує цей біт, однак встановлення його для каталогу має певний сенс. Якщо встановити біт **sticky** для каталогу, то він не дозволить видалення та перейменовування файлів та каталогів у представленому каталозі користувачу, який не є їх власником або суперкористувачем. Часто біт **sticky** встановлюється для управління доступом до загальних каталогів, наприклад, таких як ***/tmp***. У вісімковому представленні він встановлюється присвоюванням значення найстаршого розряду цифри 1, а за допомогою символічного представлення присвоюванням спеціального запису ***+t***.

У наступному прикладі створимо 2 каталоги з назвами ***temp1*** та ***temp2***, а потім встановимо для них біт **sticky** двома різними способами.

```
debianuser@debianServ2017:~$ mkdir temp1
debianuser@debianServ2017:~$ mkdir temp2
debianuser@debianServ2017:~$ chmod 1755 temp1
debianuser@debianServ2017:~$ chmod +t temp2
debianuser@debianServ2017:~$ ls -l
total 12
drwxr-sr-x 2 debianuser debianuser 4096 Nov  3 12:00 dir1
-rwSr--r-- 1 debianuser debianuser    0 Nov  3 10:59 prog1
-rwSr--r-- 1 debianuser debianuser    0 Nov  3 10:59 prog2
-rw-r-Sr-- 1 debianuser debianuser    0 Nov  3 12:00 prog3
drwxr-xr-t 2 debianuser debianuser 4096 Nov  3 12:32 temp1
drwxr-xr-t 2 debianuser debianuser 4096 Nov  3 12:32 temp2
```

З прикладу видно, що встановлений біт **sticky** у правах доступу позначається літерою ***t*** на місці символу, який відповідає за право на виконання для всіх інших користувачів.

4.4. Зміна власника та основної групи власника файлів та каталогів

Система Linux надає можливість змінити для файлів та каталогів їх власника, а також основну групу власника файлу, але у нову групу, яка вказується як основна, власник не обов’язково має входити.

Універсальною командою для зміни власника та групи є команда ***chown***, однак вона може бути запущена тільки від імені суперкористувача. Ця команда має наступний синтаксис:

chown [користувач][:[група]] <файл...>

Команда ***chown*** залежно від першого аргументу змінює власника, групу, або те й інше одразу. У табл. 4.8 представлені приклади перших аргументів команди ***chown*** та опис що виконує команда за цим аргументом.

Табл. 4.8. Приклади перших аргументів команди `chown`

Аргумент	Опис
user1	призначити власником користувача ***user1***
user1:group1	призначити власником користувача ***user1*** та групу ***group1*** у якості групи власників
:group1	призначити групу ***group1*** у якості групи власників
user1:	призначити власником користувача ***user1***, а його основну групу – групою власників.

У наступному прикладі під користувачем ***debianuser*** створимо новий каталог ***dir1***.

```
debianuser@debianServ2017:~$ mkdir dir1
debianuser@debianServ2017:~$ ls -l
total 4
drwxr-xr-x 2 debianuser debianuser 4096 Nov  3 13:25 dir1
```

Тепер під суперкористувачем змінимо власника файлу на користувача **test301**, а групу власників **gr302**.

```
root@debianServ2017:~# cd /home/debianuser
root@debianServ2017:/home/debianuser# ls -l
total 4
drwxr-xr-x 2 debianuser debianuser 4096 Nov  3 13:25 dir1
root@debianServ2017:/home/debianuser# chown test301:gr302 dir1
root@debianServ2017:/home/debianuser# ls -l
total 4
drwxr-xr-x 2 test301 gr302 4096 Nov  3 13:25 dir1
```

У старих версіях системи UNIX команда ***chown*** змінювала тільки власника файлу, але не групу. Для зміни групи використовувалася інша команда, під назвою ***chgrp***. Вона також є у системах Linux і доступна для використання як суперкористувачем, так і звичайним користувачем. Єдине обмеження для звичайного користувача полягає у тому, що він може змінити групу своїх файлів та каталогів, обравши її серед груп, членом яких він є. Але суперкористувач може змінити групу, вказавши будь-яку існуючу.

Розділ 5.
РОБОТА З ТЕКСТОВИМИ ДАНИМИ

5.1. Відображення текстових даних

Більшість налаштувань ОС на базі ядра Linux зберігається у вигляді текстових файлів, тому існує декілька інструментів для обробки текстових даних. Відображення текстових даних файлу перше питання, з яким стикається кожен користувач операційної системи, що базуються на Linux. Існує декілька інструментів для виконання цієї дії: ***cat, less*** та ***MORE***.

5.1.1. Інструмент cat

Інструмент ***cat*** (від catenate) доволі простий та функціональний засіб відображення текстових даних. Синтаксис виклику: «***cat foo.txt***» – де foo.txt текстовий файл. Варто пам'ятати, що в операційних системах на базі Linux розширення файлів необов'язкове та не завжди використовується, а отже, можливо, текстова інформація зберігається у файлі без розширення.

Команда ***cat*** має параметри для більш детального відображення даних, наприклад ключ ***-a*** дозволяє виводити керуючі символи у тексті, тобто знак табуляції (***^I***), переходу на новий рядок (***$***).

Ключ ***-n*** відображає номери рядків виводу ***cat***, а ключ ***-s*** при виводі ігнорує повторення пустих рядків. Використання цих двох параметрів дозволяє полегшити читання файлів налаштувань великого розміру.

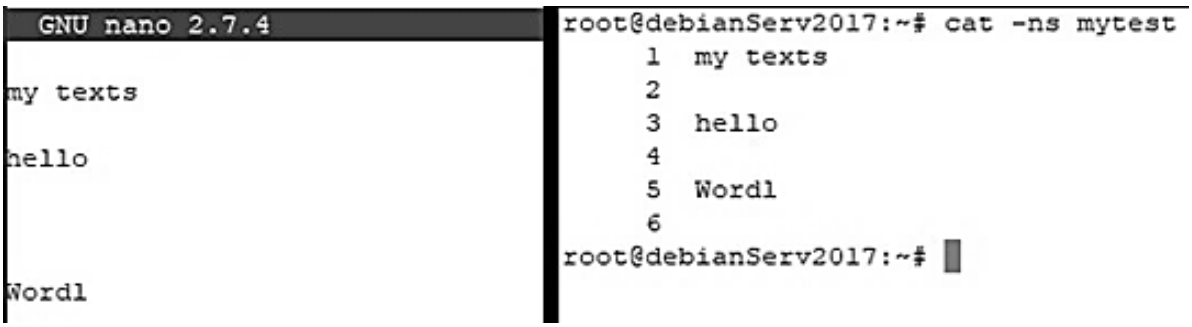

Рис. 5.1. Приклад виведення ***cat*** з ключами ***-s*** та ***-n***

Cat можна використовувати для запису у файл текстової інформації, використавши перенаправлення системного потоку виводу: ***cat >foo.txt*** – дані введені з клавіатури будуть записані у файл foo.txt, закінчити запис можна введенням керуючого символу, що означає кінець рядка: ***^D(Ctrl+D).***

Незважаючи на свою простоту інструмент ***cat*** дуже функціональний:

– використовуючи перенаправлення стандартних потоків виводу, інструментом ***cat*** можна об'єднати декілька файлів: ***$ cat file1 file2 ... file > new-file***;

– створення нового файлу також можливе шляхом перенаправлення потоку: ***$ cat > new-file;***

– за необхідності команда ***cat*** дозволяє копіювати зміст файлу в інший файл: ***$ cat file > new-file;***

Ураховуючи, що більшість консольних команд працюють через командну оболонку, підтримуються спеціалізовані символи: груповими символами (wildcards, або у деяких джерелах символи підстановки (globbing)). Вони дозволяють створювати шаблон (маску) назви файлу. В табл. 5.1. перераховані групові символи і їх відповідності.

Табл. 5.1. Групові символи

Груповий символ	Відповідає
*****	будь-яка послідовність будь-яких символів
?	будь-який символ
[Символи]	будь-який символ з вказаної множини символів
[!Символи]	будь-який символ, що не належить зазначеному
[[:Клас:]]	будь-який символ, що належить зазначеному класу.

Як і більшість інструментів операційної системи на базі ядра Linux, ***cat*** підтримує можливість модифікації виводу ключами. У таблиці 5.2. наведено ключі для модифікації виводу ***cat***.

Табл. 5.2. Ключі команди `cat`

Ключ	Значення
-b	нумерація непорожніх рядків
-E	у кінці кожного рядка буде виведено символ $
-a	відображення керуючих символів
-n	нумерація рядків включно з порожніми
-s	видалення порожніх рядків, що повторюються
-T	заміна символу табуляції на «^I» при виведенні тексту
-h	викликає довідку
-v	виводить поточну версію ***cat***.

5.1.2. Інструмент less

Іншим інструментом для перегляду тексту є ***less***. Під час виведення тексту він створює тимчасовий буфер для можливості «гортати сторінки» тексту. Керується введенням окремих символів, що наведені в таблиці 5.3.

Табл. 5.3. Символи керування `less`

Символ	Дія
G	остання сторінка файлу
g	перша сторінка файлу
k та j	переміщення курсору рядками
space	перехід на наступну «сторінку»
b	перехід на попередню «сторінку»
/pattern	пошук шаблону «pattern»
n	наступний збіг з шаблоном
N	попередній збіг з шаблоном
s file	збереження у файлі відображеної частини тексту
q	вихід
h	довідка.

5.1.3. Інструмент MORE

MORE – текстовий фільтр, що дозволяє гортати сторінки тексту на екрані. Підтримує можливість пошуку за шаблонами за допомогою керуючого символу «/» та можливість виводу тексту починаючи від знайденого шаблону: cat /etc/passwd | more +/kio – де «kio» – виступає шаблоном, а дані надходять із застосунку.

```
root@debianServ2017:~# cat /etc/passwd | more +/kio

...skipping
kio:x:0:0:ihor,2-504,,:/home/kio:/bin/bash
gleb:x:0:0:,,,:/home/gleb:/bin/bash
1610801:x:1001:1003::/home/1610801:/bin/bash
1610802:x:1002:1003::/home/1610802:/bin/bash
1610803:x:1003:1003::/home/1610803:/bin/bash
1610804:x:1004:1003::/home/1610804:/bin/bash
1610805:x:1005:1003::/home/1610805:/bin/bash
1610806:x:1006:1003::/home/1610806:/bin/bash
1610807:x:1007:1003::/home/1610807:/bin/bash
1610808:x:1008:1003::/home/1610808:/bin/bash
1610809:x:1009:1003::/home/1610809:/bin/bash
1610810:x:1010:1003::/home/1610810:/bin/bash
1610811:x:1011:1003::/home/1610811:/bin/bash
1610812:x:1012:1003::/home/1610812:/bin/bash
1610813:x:1013:1003::/home/1610813:/bin/bash
1610814:x:1014:1003::/home/1610814:/bin/bash
1610815:x:1015:1003::/home/1610815:/bin/bash
1610816:x:1016:1003::/home/1610816:/bin/bash
1610817:x:1017:1003::/home/1610817:/bin/bash
1610818:x:1018:1003::/home/1610818:/bin/bash
1610819:x:1019:1003::/home/1610819:/bin/bash
```

Як і інструмент ***less, MORE*** керується за допомогою спеціалізованих символів, що наведені у таблиці 5.4.

Табл. 5.4. Символи керування `more`

Символ	Дія
g	перехід до останньої сторінки файлу
b	повернення на половину «сторінки» назад
space	перехід на наступну сторінку
d	перехід на пів «сторінки» вперед
/pattern	пошук шаблону «pattern»
=	відображення номера рядка, в якому знаходиться курсор
:n	перехід до наступного файлу
:p	повернення до попереднього файлу
:f	виведення номера рядка, в якому курсор, та назва поточного файлу
!	виконання Shell команди
v	виклик текстового редактора, та встановлює курсор на поточну позицію
q	вихід
h	виклик довідки.

5.2. Інструменти роботи з текстовою інформацією

5.2.1. Інструмент для сортування: sort

Сортування текстових даних в операційній системі на базі ядра Linux можливе за допомогою утиліти ***sort***, що сортує дані з файлу в порядку зростання та виводить результат у вихідний потік. Використовуючи команду ***sort*** для кількох файлів одночасно їх вміст буде об'єднано та відсортовано. Задати певні параметри команди ***sort*** можливо ключам, що наведені в таблиці 5.5.

Табл. 5.5. Ключі команди `sort`

Скорочений запис	Повний запис	Значення
-b	***--ignore-leading-blanks***	Ігнорування початкових порожніх рядків
-f	***--ignore-case***	Сортування не враховуючи регістр літер
-h	***--human-numeric-sort***	Порівняння числових значень, враховуючи скорочення степенів
-r	***--reverse***	Зворотне сортування
-u	***--unique***	Працює у двох режимах: з додатковим ключем ***-c*** перевіряє увесь файл та без цього ключа виводить лише перший рядок серед кількох однакових
-o	***--output***	Виводить відсортовані дані не у системний потік, а у файл обраний користувачем
-k	***--key***	Сортування заданим користувачем полю, має наступний синтаксис: ***--key = поле1 [, поле2]***
-t	***--field-separator***	Ключ визначає знак для розмежування полів.

31.03.1955
30.10.1946
11.12.1956
06.02.1962

Користувач може задати команду «***sort -t . -k 3 test***», де ***test*** – назва файлу з даними, знаком розмежування використаємо крапку, а полем для сортування обрано рік народження тобто 3.

```
kio@chmnu-testl:~$ sort -t . -k 3 test
30.10.1946
31.03.1955
11.12.1956
06.02.1962
kio@chmnu-testl:~$
```

5.2.2. Інструмент вибірки даних: cut

Дистрибутиви, що базуються на ядрі ***Linux***, також підтримують можливість вибірки текстових даних, що зберігаються у вигляді простих електронних таблиць, наприклад CSV.

Вибірку даних, що зберігається в подібному вигляді можливо здійснити утилітою ***cut***. Ця утиліта працює аналогічно до команди ***sort*** з ключами ***-t*** та ***-k***, але не сортує дані.

Ключі команди ***cut*** наведені у таблиці 5.6.

Табл. 5.6. Ключі команди `cut`

Скорочений запис	Повний запис	Значення
-c	***--characters***	Вибірка лише заданих користувачем символів, вибірка здійснюється за індексом у кожному рядку
-d	***--delimiter***	Задає знак розмежування полів, за замовчуванням знаком розмежування є табуляція
-f	***--fields***	Задає номер поля, що необхідно вивести на екран
-s	***--only-delimited***	Виводить лише ті рядки у яких є символ розмежування;
	--output-delimiter	Встановлює рядок розмежування полів при виводі даних, за замовчуванням
	--help	Довідка
	--version	Поточна версія ***cut***.

Синтаксис виклику ***cut*** виглядає наступним чином: ***cut -d "; " -f 2-3 --output-delimiter="----" -s data.csv***. Вхідним файлом є data.csv, що містить наступні рядки:

```
Time;Temp *C;Humibiliti %
00:02;19;83
00:06;19;82
213812939123891
00:08;19;81
00:10;19;82
832788212.
00:12;19;82
00:15;19;83
00:17;19;82
```

Виконавши задану команду, користувач отримає другий та третій стовпчик, що розділені вказаним рядком та без рядків, що не містять розмежовуючого символу.

```
kio@chmnu-test1:~$ cut -d ";" -f 2-3 --output-delimiter="----" -s data.csv
Temp *C----Humibiliti %
19----83
19----82
19----81
19----82
19----82
19----83
19----82
kio@chmnu-test1:~$
```

5.2.3. Інструмент додавання даних у текстові файли: paste

Окрім використання можливості перенаправлення даних системного потоку в текстовий файл, підтримується можливість додавання окремих стовпців за допомогою команди ***paste***.

Ключі команди ***paste*:**

– ***«-s»*** – змінює положення рядків та стовпців;

– ***«-d»*** – змінює розмежовуючий символ, за замовчуванням знаком розмежування є табуляція.

Синтаксис виклику команди ***paste: paste -d "-" test test2***, де файл ***test*** – містить дні народження (див. розділ 5.2.1) та файл ***test2*** містить імена:

```
Angus McKinnon Young
Chris Slade
Stephen Crawford Young
Axl Rose
```

```
kio@chmnu-test1:~$ paste -d "-" test test2
31.03.1955-Angus McKinnon Young
30.10.1946-Chris Slade
11.12.1956-Stephen Crawford Young
06.02.1962-Axl Rose
kio@chmnu-test1:~$
```

5.2.4. Інструмент фільтрування повторень: uniq

Uniq – є командою, що дозволяє виведення повторюваних значень, або їх видалення з відсортованого файлу. Команда також підтримує роботу з потоком введення даних. За замовчуванням виведення даних відбувається в стандартний потік виведення. Список ключів команди ***Uniq*** наведено у таблиці 5.7.

Табл. 5.7. Ключі команди ***uniq***

Ключ	Значення
-c	Друкує перед кожним рядком кількість його повторів (без відображення повторів)
-d	Відображає лише рядки, що повторюються
-i	Порівняння рядків не враховуючи регістр
-s	Встановлює кількість символів, що будуть проігноровані при порівнянні
-u	Виводить лише унікальні рядки, параметр встановлено за замовчуванням
-f	Встановлює кількість стовпців (від початку), що будуть проігноровані при перевірці, символ розмежування – проміжок (відсутня можливість зміни цього символу)
-w	Вказує кількість символів, від початку рядка, що порівнюються, усі інші символи ігноруються.

Приклад синтаксису: ***uniq -c data.csv***, де data.csv – файл, що містить наступні дані:

```
Time;Temp *C;Humibiliti %
00:02;19;83
00:02;19;83
213812939123891
kio@chmnu-test1:~$ uniq -c data.csv
      1 Time;Temp *C;Humibiliti %
      2 00:02;19;83
      1 213812939123891
```

5.2.5. Інструменти head та tail

Робота з великими об’ємами даних часто потребує використання обмеження. Виконати обмеження виводу даних можливо за допомогою спеціалізованих команд ***head*** та ***tail***.

head задає обмеження на вивід строк з початку файлу, тобто виводить N строк. Синтаксис команди: ***head -n 20 data.csv***. За відсутності ключа ***-n*** за замовчуванням виводить 10 строк з початку файлу.

tail схожий за принципом дії на команду ***head***, але показує певну кількість рядків з кінця файлу. Синтаксис команди: ***tail -n 20 data.csv***.

```
kio@chmnu-test1:~$ tail -20 data.csv
08:58;18;82
09:00;18;82
09:02;18;82
09:04;18;84
09:06;18;83
09:08;18;84
09:10;18;85
09:14;18;86
09:16;18;84
09:18;18;84
09:20;18;82
09:22;18;84
09:25;18;84
09:27;18;84
09:30;18;83
09:32;18;84
09:34;18;83
09:36;18;84
09:39;18;84
09:41;18;83
```

head та ***tail*** підтримують можливість модифікації виводу за допомогою ключів наведених в таблиці 5.8.

Табл. 5.8. Ключі команди `head` та `tail`

Скорочений запис	Повний запис	Значення
-c	***--bytes***	Друкує задану кількість байт (спочатку або кінця залежно від команди)
-f	***--follow***	Динамічний друк змін у файлі
-n	***--lines***	Друк вказаної кількості рядків
	--pid	Використовується разом з *-f*, завершує процес виведення інформації після завершення роботи процессу з заданим **PID**
-q	***--quiet, --silent***	Не друкує назви файлів
	--retry	Ключ, що вказує на необхідність повторних спроб відкриття файлу, в разі проблем з доступом.

5.2.6. Інструмент підрахунку кількості рядків, символів, байт: wc

Обробка даних деяких файлів журналів вимагає не лише параметричного виводу інформації, але й подальшої її обробки. Одним з варіантів обробки є підрахунок кількості записів, наприклад для визначення навантаження на сервер необхідно підрахувати кількість запитів за певний період. Для виконання цієї дії в Linux використовується інструмент ***wc***.

Команда ***wc*** (word count – лічильник слів) використовується для підрахунку числа рядків, слів і байтів у файлах. Наприклад $ ***wc***

/etc/passwd – надрукує у стандартний потік виведення кількість рядків, символів та байт у файлі passwd:

```
kio@chmnu-testl:~$ wc /etc/passwd
  32   45 1710 /etc/passwd
```

Використання цього інструмента для роботи зі стандартним потоком виведення також можливе шляхом використання конвеєра, та не встановлюючи файл призначення:

```
kio@chmnu-testl:~$ ls /etc | wc
    163     163    1507
```

Модифікація виводу команди ***wc*** здійснюється ключами наведеними у таблиці 5.9.

Табл. 5.9. Ключі команди `wc`

Скорочений запис	Повний запис	Значення
-c	***--bytes***	друкує лише кількість байт
-m	***--chars***	друкує лише кількість символів
-l	***--lines***	друкує лише кількість рядків
-L	***--max-line-length***	друкує розмір найдовшого рядка
-w	***--words***	друкує кількість слів
	--help	довідка
	--version	поточна версія wc.

Незважаючи на простоту цей інструмент є надзвичайно корисним, наприклад використовуючи ключ ***-l*** можна визначити кількість користувачів у ОС:

```
kio@chmnu-testl:~$ wc -l /etc/passwd
32 /etc/passwd
```

5.2.7. Інструмент пошуку шаблону в тексті: grep

Доволі потужним інструментом обробки текстових даних в операційних системах на базі ядра Linux є ***grep***. Його задачею є пошук шаблону всередині файлів. Коли ***grep*** знаходить в файлі схожість з «шаблоном», він виводить рядок зі знайденим збігом. Шаблони, що використовуються командою ***grep*** для пошуку, можуть бути дуже складними, наприклад для друку інформації щодо певного користувача з файлу /etc/passw можливо ввести наступну команду:

```
kio@chmnu-testl:~$ grep kio /etc/passwd
kio:x:1001:1002:22121,121211,12121,121,1212:/home/kio:/bin/bash
```

Але можливість використання складних шаблонів дозволяє вирішувати складніші задачі, наприклад визначити усіх користувачів, командною оболонкою для яких встановлено bash:

```
kio@chmnu-testl:~$ grep "/bin/bash" /etc/passwd
root:x:0:0:root:/root:/bin/bash
ihor_kandyba:x:1000:1001::/home/ihor_kandyba:/bin/bash
kio:x:1001:1002:22121,121211,12121,121,1212:/home/kio:/bin/bash
k3y_vl:x:1002:1003:User2,00,00,00:/home/k3y_vl:/bin/bash
```

Модифікувати команду ***grep*** можливо кількома ключами, що наведені в таблиці 5.10.

Табл. 5.10. Ключі команди `grep`

Ключ	Значення
-i	ігнорує регістр літер
-n	друкує номер рядка в якому знайдено шаблон разом з самим рядком
-c	друкує лише номер рядка з шаблоном
-v	друкує рядки без шаблону
-w	шукає шаблон, що відокремлений пробілами (шаблон як слово)
-r	рекурсивний пошук файлів за директорією
-l	виведення назв файлів у яких знайдено шаблон (назва файлу виводиться лише один раз)
-e	пошук за списком шаблонів
-E	у якості шаблону використовується регулярний вираз
-x	збігом вважається рядок із повним збігом регулярного виразу.

Можливість використання регулярних виразів робить інструмент дуже функціональним: наприклад для пошуку користувачів імена яких починаються на літеру *k* достатньо використати команду:

```
kio@chmnu-testl:~$ cut -f 1 -d ":" /etc/passwd | grep ^k
kio
k3y_vl
```

Розділ 6.
ПОШУК ТА АРХІВАЦІЯ ФАЙЛІВ

6.1 Пошук файлів

Однією із задач, що часто виникає у системі Linux, є пошук певних файлів та каталогів. Для цього призначені 2 команди: ***locate***, яка шукає файли або каталоги за назвами у системній базі даних назв файлів, та ***find***, яка шукає файли або каталоги рекурсивно за ієрархією каталогів. Далі вищезазначені команди будуть розглянуті більш детально.

6.1.1. Простий пошук файлів

Найбільш простий пошук файлів у файловій системі здійснює команда ***locate***, яка, як було наголошено, шукає файли у системній базі даних назв файлів. Наприклад, здійснимо пошук файлів, які у своїй назві мають підрядок «***bin/zip***», тобто програми, назва яких починається на ***zip*** та які знаходяться у каталозі ***bin***.

```
debianuser@debianServ2017:~$ locate bin/zip
/usr/bin/zip
/usr/bin/zipcloak
/usr/bin/zipdetails
/usr/bin/zipgrep
/usr/bin/zipinfo
/usr/bin/zipnote
/usr/bin/zipsplit
```

Команда ***locate*** не є надійним способом пошуку файлів. Річ у тому, що база даних назв файлів оновлюється періодично через регулярні інтервали часу, в більшості дистрибутивів системи Linux один раз на добу. Тому найновіші файли команда може не знайти одразу.

Однак базу даних назв файлів можна примусово оновити шляхом виконання команди ***updatedb***, але вона може бути виконана тільки від імені суперкористувача.

6.1.2. Складний спосіб пошуку файлів

Іншим способом пошуку файлів є використання команди ***find***. Він є більш надійним, оскільки команда виконує його згідно з заданими атрибутами у зазначеному каталозі, причому робить це рекурсивно, тобто виконує пошук і у підкаталогах. Однак такий спосіб є складнішим, тому що команда ***find*** має досить велику кількість спеціальних аргументів, які можуть бути використані при пошуку.

Найпростіше застосування цієї команди як першого аргументу полягає у зазначенні відповідного каталогу, однак при цьому будуть

виведені всі файли та каталоги, що знаходяться не тільки у каталозі, але й у підкаталогах, враховуючи і приховані файли. Наприклад, задамо команді find у якості аргументу домашній каталог користувача ***debianuser***:

```
debianuser@debianServ2017:~$ find ~
/home/debianuser
/home/debianuser/.swm
/home/debianuser/statistics
/home/debianuser/.swo
/home/debianuser/.xsession-errors
/home/debianuser/fileC
/home/debianuser/.wrt.swp
/home/debianuser/.cache
/home/debianuser/.cache/gnome-software
/home/debianuser/.cache/gnome-software/3.22
/home/debianuser/.cache/gnome-software/3.22/ratings
/home/debianuser/.cache/gnome-software/3.22/ratings/odrs.json
/home/debianuser/.cache/evolution
/home/debianuser/.cache/evolution/memos
/home/debianuser/.cache/evolution/memos/trash
/home/debianuser/.cache/evolution/mail
/home/debianuser/.cache/evolution/mail/trash
/home/debianuser/.cache/evolution/sources
/home/debianuser/.cache/evolution/sources/trash
/home/debianuser/.cache/evolution/addressbook
/home/debianuser/.cache/evolution/addressbook/trash
/home/debianuser/.cache/evolution/calendar
/home/debianuser/.cache/evolution/calendar/trash
```

Таким чином, ми бачимо, що команда ***find*** у подібному застосуванні виводить достатньо довгий список. Насправді це є цілком природним, оскільки ми не задали жодного критерію пошуку. Виконання команди ***find*** без застосування критеріїв пошуку, скоріш схоже на виконання команди ***ls -Ra***, де ключ ***-a*** примушує виводити приховані файли та каталоги, а ***-R*** – виводить вміст каталогу рекурсивно. Однак зазначимо, оскільки результат команди ***find*** як і результати майже всіх команд виводиться у стандартний потік виведення, він може бути переданий конвеєром іншим командам. Наприклад, підрахуємо загальне число файлів та каталогів, що знаходяться у домашньому каталозі користувача ***debianuser*** та його підкаталогах. Для цього скористаємося командою ***wc -l***.

```
debianuser@debianServ2017:~$ find ~ | wc -l
160
```

Однак найбільший ефект від використання команди ***find*** буде отриманий при застосуванні у пошуку файлів певних критеріїв. Ці критерії вказують за допомогою спеціальних аргументів, які можна поділити на 4 типи: **перевірки**, **оператори**, **операції** та **параметри**.

Перевірки

Першим типом спеціальних аргументів для команди ***find*** є перевірки. Їх призначення полягає у завданні певного критерію пошуку. Існує доволі багато перевірок, найбільш застосовувані з яких представлені у табл. 6.1.

Табл. 6.1. Перевірки, які підтримуються командою `find`

Перевірка	Опис
-type <type>	Відповідає елементам, що мають тип ***type***
-name <template>	Відповідає файлам або каталогам, назви яких збігаються з зазначеним шаблоном ***template***
-iname <template>	Діє так само, як перевірка ***-name***, але розрізняє регістр символів
-inum <n>	Відповідає файлам з номером індексного вузла (inode) ***n***. Перевірку зручно використовувати для пошуку всіх жорстких посилань на певний індексний вузол
-samefile <name>	Діє так само, як перевірка ***-inum***. Відповідає файлам з тим же номером індексного вузла (inode), що і файл з вказаною назвою
-cmin <n>	Відповідає файлам або каталогам, вміст або атрибути яких останній раз змінювалися ***n*** хвилин тому. У перевірці можна використовувати знаки ***–*** та ***+*** перед відповідним числом для того, щоб задати умови, що файл/каталог змінювався менш або більш ніж ***n*** хвилин тому відповідно
-ctime <n>	Те саме, що і ***-cmin***, тільки у якості числа вказуємо кількість днів, а не хвилин (***0*** – сьогодні, ***1*** – вчора і т. д.)
-cnewer <name>	Відповідає файлам або каталогам, вміст або атрибути яких останній раз змінювалися пізніше, ніж у файлі або каталозі з вказаною назвою ***<name>***
-mmin <n>	Відповідає файлам або каталогам, вміст яких останній раз змінювався ***<n>*** хвилин тому
-mtime <n>	Те саме, що і ***-mmin***, тільки у якості числа вказуємо кількість днів, а не хвилин
-newer <name>	Відповідає файлам або каталогам, вміст яких останній раз змінювався пізніше, ніж у файлі або каталозі з вказаною назвою ***<name>***
-size <n>	Відповідає файлам з розміром ***n***. У перевірці можна використовувати знаки ***+*** та ***–*** перед відповідним числом для пошуку файлів або каталогів, що є більшими або меншими відповідно до заданого розміру
-empty	Відповідає порожнім файлам і каталогам
-user <user>	Відповідає файлам або каталогам, що належать зазначеному користувачеві ***user***. Аргумент ***user*** може бути назвою або унікальним ідентифікатором користувача
-nouser	Відповідає файлам або каталогам, які не належать якомусь допустимому користувачеві. Таку перевірку доцільно використовувати для пошуку файлів, що належать видаленим обліковим записам, або для виявлення слідів зловмисників
-group <group>	Відповідає файлам або каталогам, що належать зазначеній групі. Аргумент ***group*** може здаватися назвою або числовим ідентифікатором групи
-nogroup	Відповідає файлам або каталогам, що не належать будь-якій допустимій групі
-perm <permission>	Відповідає файлам або каталогам з зазначеним режимом доступу ***permission***. Режим може виражатися як у вісімковій так і символічній формі.

У свою чергу в перевірці за типом (***-type***) у якості значення потрібно використовувати наступні символи, що перераховані у табл. 6.2.

Табл. 6.2. Символи, що використовуються при перевірці за типом у команді *find*

Тип файлів	Опис
b	Спеціальний файл блочного пристрою
c	Спеціальний файл символьного пристрою
d	Каталог
f	звичайний файл
l	символічне посилання.

Як приклад отримаємо кількість каталогів включаючи і приховані у домашньому каталозі користувача ***debianuser***,. Для цього виконаємо наступний пошук:

```
debianuser@debianServ2017:~$ find ~ -type d | wc -l
90
```

Таким чином, ми обмежилися пошуком тільки каталогів. Так само можна знайти у домашньому каталозі користувача ***debianuser*** тільки звичайні файли:

```
debianuser@debianServ2017:~$ find ~ -type f | wc -l
123
```

У якості наступного прикладу знайдемо всі файли, які є сценаріями командного рядка. Зазвичай такі файли мають розширення ***.sh***, що є скороченням від слова **shell**. Програмування сценаріїв командного рядка буде детально розглянуто у наступних розділах посібника.

Для віднаходження таких файлів командою ***find*** необхідно використати перевірку ***-name***:

```
debianuser@debianServ2017:~$ find ~ -name "*.sh"
/home/debianuser/first.sh
/home/debianuser/scripts/until.sh
/home/debianuser/scripts/array.sh
/home/debianuser/scripts/for2.sh
/home/debianuser/scripts/test_string.sh
/home/debianuser/scripts/condition.sh
/home/debianuser/scripts/for3.sh
/home/debianuser/scripts/graphic/inputPassword.sh
/home/debianuser/scripts/graphic/menu.sh
/home/debianuser/scripts/graphic/yesno.sh
/home/debianuser/scripts/graphic/deleteFile.sh
```

Тепер розглянемо детальніше пошук елементів за розміром. Як було зазначено у табл. 6.1 при завданні значення розміру можна використовувати символи ***+*** та ***–*** для пошуку файлів, які є більшими або меншими заданого розміру. Число без знаку буде означати: «в точності відповідає значенню». Крім того, можна вказати певну одиницю виміру

розміру за допомогою відповідних спеціальних символів, які перераховані у табл. 6.3.

Табл. 6.3. Одиниці виміру розміру файлу, що підтримуються командою *find*

Символ	Одиниця виміру
b	блоки розміром по 512 байт (використовується за замовчуванням, якщо інше не зазначено)
c	байти
w	2-байтні слова
k	кілобайти (Kilobytes, блоки по 1024 байт)
M	мегабайти (Megabytes, блоки по 1 048 576 байт)
G	гігабайти (Gigabytes, блоки по 1 073 741 824 байт).

У наступному прикладі спробуємо знайти у каталозі ***/home***, тобто батьківському каталозі для домашніх каталогів усіх користувачів, прості файли, що мають розмір більше ніж 1 Мегабайт:

```
debianuser@debianServ2017:~$ find /home -type f -size +1M
find: '/home/1610304/TextFiles/New': Permission denied
find: '/home/1610304/my1': Permission denied
find: '/home/1610304/rmp': Permission denied
/home/1610301/home.txt
find: '/home/1610301/dir1': Permission denied
/home/1610301/big.txt
find: '/home/1610301/tmp': Permission denied
/home/1710817/memory/stat
find: '/home/1710125/exmp': Permission denied
/home/1710102/.vim/spell/ru.koi8-r.sug
find: '/home/1610325': Permission denied
find: '/home/1710906': Permission denied
```

Таким чином, у представленому прикладі ми знайшли у каталозі ***/home*** та його підкаталогах всі файли, розмір яких є більшим за 1 Мегабайт. У результаті виведення можна помітити, що для деяких файлів було виведено повідомлення, що у поточного користувача, яким є ***debianuser*** немає прав доступу до нього.

Оператори

У останньому прикладі під час пошуку було вказано та об’єднано одразу дві перевірки: по-перше, за типом, по-друге за розміром,. У команді ***find*** можна задати більше двох перевірок та комбінувати їх за допомогою логічних операторів, що дозволяє визначити більш складні критерії для пошуку елементів у файловій системі.

Логічні оператори є другим типом спеціальних аргументів команди ***find***. Вони представлені у табл. 6.4.

Табл. 6.4. Логічні оператори, які підтримуються командою *find*

Оператор	Опис
-and	**Кон'юнкція.** Є істинною під час виконання умов у перевірках з обох сторін від оператора. Можна скоротити до **-a**. Саме цей оператор використовується за замовчуванням за відсутності операторів (саме оператор **-and** був використаний в останньому прикладі)
-or	**Диз'юнкція.** Є істинною під час виконання однієї з умов у перевірках в будь-якій зі сторін від оператора. Можна скоротити до **-o**
-not	**Заперечення.** Є істинним, якщо умова в перевірці, що слідує за оператором, не виконується. Можна скоротити до **-!**.

Застосування декількох перевірок та операторів може сформувати достатньо великі вирази. Якщо при цьому використовується більш ніж один оператор для визначення порядку виконання, то найбільш пріоритетним є оператор **-not**, а найменш пріоритетним – **-or**. Для зміни порядку виконання операторів у великих виразах можна використати круглі дужки, навіть якщо вони не потрібні, іноді корисно застосовувати їх, щоб зробити команди більш наочними. Однак пам'ятайте, що круглі дужки мають спеціальне значення для командної оболонки, тому їх потрібно екранувати та передавати команді **find** як аргументи. Зазвичай екранування виконують за допомогою символу зворотного слешу, який екранує символ, що слідує за ним.

Представимо приклад використання команди **find** зі складною умовою: знайдемо у домашньому каталозі користувача **debianuser** всі елементи, які не є каталогами та за розміром більше ніж 100 Кілобайт.

```
debianuser@debianServ2017:~$ find ~ \( -size +100k \) -and \( -not -type d \)
/home/debianuser/.cache/tracker/ontologies.gvdb
/home/debianuser/.cache/tracker/meta.db
/home/debianuser/.cache/tracker/meta.db-wal
/home/debianuser/.cache/gstreamer-1.0/registry.x86_64.bin
/home/debianuser/.local/share/tracker/data/tracker-store.journal
```

Операції

У команді **find** крім завдання умов пошуку за допомогою перевірок та операторів, що їх поєднують, можна вказувати певну дію, яку потрібно зробити за результатом пошуку. Відповідні дії вказують за допомогою операцій, які є третім типом спеціальних аргументів команди **find**. Найбільш часто застосовувані операції представлені у табл. 6.5.

Табл. 6.5. Зумовлені операції, підтримувані командою *find*

Операція	Опис
-delete	Видаляє поточний знайдений файл
-ls	Діє еквівалентно команді **ls -dils** щодо знайденого файлу. Результат виводиться у стандартний потік виведення
-print	Виводить повний шлях до знайденого файлу в стандартне виведення. Операція виконується за замовчуванням, якщо не вказана ніяка інша
-quit	Завершує виконання команди після виявлення першого збігу.

Підтримуваних операцій набагато більше, ніж представлено у таблиці. Повний список можна знайти на сторінці довідкового керівництва (***man***) для команди ***find***.

Якщо у команді ***find*** не вказано жодної операції, передбачається операція ***-print***. Так, при виконанні команди ***find ~*** буде виведено список усіх файлів і підкаталогів, що зберігаються в домашньому каталозі. Таку саму команду можна було б представити і наступним чином:

```
find ~ -print
```

Досить зручно використовувати команду ***find*** для видалення файлів, які відповідають певним критеріям. Наприклад, наступна команда видалить усі порожні файли:

```
find ~ -type f -empty -delete
```

Параметри

Четвертим типом аргументів спеціального призначення у команді ***find*** є параметри. Вони використовуються нечасто, однак допомагають керувати областю пошуку. Найбільш часто використовувані параметри представлені у табл. 6.6.

Табл. 6.6. Параметри команди *find*

Параметр	Опис
-depth	Вимагає від ***find*** обробити спочатку файли в каталогах і тільки потім каталоги. Цей параметр автоматично застосовується з операцією ***-delete***
-maxdepth <число_рівнів>	Встановлює максимальне число рівнів, на яке команда ***find*** може опускатися в дереві каталогів, виконуючи перевірки та операції
-mindepth <число_рівнів>	Встановлює мінімальне число рівнів, на яке команда ***find*** має опуститися в дереві каталогів перед виконанням перевірок і операцій
-mount	Вимагає від ***find*** не виконувати обхід каталогів, у які вмонтовані інші файлові системи
-noleaf	Вимагає від ***find*** неоптимізованого пошуку, спираючись на припущення, що він здійснюється в UNIX-подібній файловій системі. Цей параметр необхідно використовувати при обході файлових систем DOS / Windows CD-ROM.

Для прикладу виконаємо пошук простих файлів у домашньому каталозі користувача ***debianuser***, опустившись максимально тільки на один рівень у дереві каталогів.

```
debianuser@debianServ2017:~$ find ~ -maxdepth 1 -type f
/home/debianuser/.swm
/home/debianuser/statistics
/home/debianuser/.swo
/home/debianuser/.xsession-errors
/home/debianuser/.wrt.swp
/home/debianuser/.file2.swp
/home/debianuser/testing
/home/debianuser/.Xauthority
/home/debianuser/.selected_editor
/home/debianuser/.bash_history
```

6.2. Стиснення файлів

Задача стиснення даних протягом усієї історії розвитку обчислювальної техніки завжди мала надважливе значення, оскільки представляє способи розміщення великої кількості даних у меншому обсязі, незалежно від того чи зберігаються дані у пам'яті, певному пристрої або смузі пропускання мережі.

Для стиснення даних можна використати алгоритми, які тим чи іншим чином відносяться до двох основних типів: стиснення без втрат (lossless) та з втратами (lossy). Після здійснення алгоритму першого типу під час відновлення початкового файлу з його стислої версії він буде мати той самий вміст, що і нестислий оригінал. З іншого боку, при здійсненні алгоритмів стиснення другого типу відновлений зі стислої версії початковий файл не буде збігатися з файлом-оригіналом, хоча буде досить близьким за вмістом до нього. Типовими прикладами стиснення з втратами є файли, що мають формат JPEG та MP3. Однак у більшості випадків втрати даних у комп'ютерах не допускаються.

В ОС Linux за задачу стиснення файлів відповідають команди ***gzip*** та ***bzip2***. Зокрема обидві розглянуті вище команди здійснюють стиснення без втрат.

Щоб зрозуміти, як працюють зазначені команди розглянемо наступний приклад. Для цього скопіюємо файл ***/etc/passwd*** під назвою ***passwd_copy*** у той каталог, у якому користувач ***debianuser*** має привілеї для запису.

```
debianuser@debianServ2017:~/dir1$ cp /etc/passwd passwd_copy
debianuser@debianServ2017:~/dir1$ ls -l
total 12
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd copy
```

Після цього виконаємо команду ***gzip***, вказавши їй у якості аргументу назву цього файлу.

```
debianuser@debianServ2017:~/dir1$ gzip passwd_copy
debianuser@debianServ2017:~/dir1$ ls -l
total 4
-rw-r--r-- 1 debianuser debianuser 2745 Dec 30 13:30 passwd_copy.gz
```

З результату можна помітити, що команда ***gzip*** замінила оригінальний файл, його стислою версією. Новий стислий файл при цьому матиме назву ***passwd_copy.gz***, тобто за замовчуванням назва стисненого файлу формується простим додаванням розширення ***.gz*** до назви основного файлу. Також можна побачити, що стиснений файл зменшився за розміром, який є практично в 5 разів меншим, ніж розмір оригінального файлу. Також помітним є те, що дозволи та час створення стисненого файлу мають таке ж значення, що і файл-оригінал.

Тепер спробуємо повернути файл оригінального розміру. Для цього використаємо команду ***gunzip***, що повертає оригінальний розмір файлу, який був стиснений командою ***gzip***.

```
debianuser@debianServ2017:~/dir1$ gunzip passwd_copy.gz
debianuser@debianServ2017:~/dir1$ ls -l
total 12
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
```

Таким чином, стиснена версія була замінена початковим файлом-оригіналом, дозволи та час його створення знову ніяк не змінилися.

Нагадаємо, використання команди ***gzip*** за замовчуванням призводить до заміни оригінального файлу його стисненою версією, хоча іноді вважатиметься за потрібне мати обидві версії файлу: і файл-оригінал, і стиснений аналог. Для цього варто використати ключі ***-c*** або ***-k***. Розглянемо, чим же відрізняються виконання команди ***gzip*** з використанням заданих ключів.

Ключ ***-c*** з командою ***gzip*** не створює стиснену версію файлу, натомість виводить її нестандартний потік виведення. Утім таке застосування цієї команди навряд чи буде корисним, однак вказавши після аргумента команди оператор перенаправлення стандартного потоку виведення, ми можемо перенаправити стиснену версію у файл, який можна назвати так, як ми захочемо. Розгянуте вище можна побачити у наступному прикладі.

```
debianuser@debianServ2017:~/dir1$ gzip -c passwd_copy > passwd.gz
debianuser@debianServ2017:~/dir1$ ls -l
total 16
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
-rw-r--r-- 1 debianuser debianuser  2745 Dec 30 13:51 passwd.gz
```

Отже, ми створили стиснену версію, але основна її назва буде відрізнятися від основної назви файлу-оригіналу.

Іншою опцією, що дозволяє збереження як файлу-оригіналу, так і його стисненої версії, є ключ ***-k***. Однак назва стисненого файлу буде складатися з назви оригінального файлу та доданого розширення ***.gz***, тобто так само як і при здійсненні команди ***gzip*** за замовчуванням. Усе вищезазначене демонструє наступний приклад.

```
debianuser@debianServ2017:~/dir1$ gzip -k passwd_copy
debianuser@debianServ2017:~/dir1$ ls -l
total 20
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
-rw-r--r-- 1 debianuser debianuser  2745 Dec 30 13:30 passwd_copy.gz
-rw-r--r-- 1 debianuser debianuser  2745 Dec 30 13:51 passwd.gz
```

Окрім зазначених вище ключів команда ***gzip*** має також й інші досить важливі ключі, основні з яких наведені у табл. 6.7. Деякі з них також мають аналогії, що складаються з повних слів.

Табл. 6.7. Основні ключі команди `gzip`

Короткий ключ	Довгий ключ	Опис
-c	***--stdout*** ***--to-stdout***	Збереження оригінальних файлів та виведення результату стиснення на стандартний потік виведення
-d	***--decompress*** ***--uncompress***	Повернення файлу-оригіналу з його стисненої версії (аналогічно команді ***gunzip***)
-f	***--force***	Примусове стиснення файлу, навіть якщо його стиснена версія вже існує
-h	***--help***	Виведення довідкової інформації щодо використання команди
-l	***--list***	Виведення списку статистик для кожного стисненого файлу
-t	***--test***	Перевірка цілісності стисненого файлу
-v	***--verbose***	Виведення у процесі роботи повідомлення з інформацією про хід стиснення
-<число>		Встановлення ступеня стиснення (можливі значення від 1 до 9). За замовчуванням використовується значення 6
-1	***--fast***	Найнижчий ступінь стиснення файлу, що характеризується високою швидкістю роботи
-9	***--best***	Найвищий ступінь стиснення, що характеризується низькою швидкістю роботи.

Наведемо приклади застосування деяких описаних вище ключів. Спочатку видалимо всі раніше стиснені файли, потім зробимо 3 різні стиснені версії файлу ***passwd_copy*** зі ступенем стиснення 1, 6 та 9.

```
debianuser@debianServ2017:~/dir1$ rm *.gz
debianuser@debianServ2017:~/dir1$ ls -l
total 12
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
debianuser@debianServ2017:~/dir1$ gzip -c passwd_copy > passwd_copy6.gz
debianuser@debianServ2017:~/dir1$ gzip -c -1 passwd_copy > passwd_copy1.gz
debianuser@debianServ2017:~/dir1$ gzip -c -9 passwd_copy > passwd_copy9.gz
debianuser@debianServ2017:~/dir1$ ls -l
total 24
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
-rw-r--r-- 1 debianuser debianuser  3141 Dec 30 14:02 passwd_copy1.gz
-rw-r--r-- 1 debianuser debianuser  2745 Dec 30 14:01 passwd_copy6.gz
-rw-r--r-- 1 debianuser debianuser  2697 Dec 30 14:02 passwd_copy9.gz
```

Далі переглянемо статистику для кожного зі стиснених файлів, виконавши команду ***gzip*** з ключем ***-l***. Найбільшої уваги надаємо відсотку стиснення кожного з файлів.

```
debianuser@debianServ2017:~/dir1$ 'gzip -l passwd_copy1.gz
         compressed        uncompressed  ratio uncompressed_name
               3141               11980  74.0% passwd_copy1
debianuser@debianServ2017:~/dir1$ gzip -l passwd_copy6.gz
         compressed        uncompressed  ratio uncompressed_name
               2745               11980  77.3% passwd_copy6
debianuser@debianServ2017:~/dir1$ gzip -l passwd_copy9.gz
         compressed        uncompressed  ratio uncompressed_name
               2697               11980  77.7% passwd_copy9
```

Тепер розглянемо питання перегляду стиснених файлів. Разом з командою ***gzip*** існують команди ***zcat*** та ***zless***, що є аналогами команд ***cat*** та ***less*** для стиснених файлів.

Іншою командою, що призначена для стиснення файлів, є ***bzip2***, автором якої є Джуліан Стюард. Вона застосовує алгоритм, який використовує перетворення Барроуза-Уілера. Цей алгоритм забезпечує більш високий ступінь стиснення за рахунок зниження швидкості роботи.

Команда ***bzip2*** діє практично так само як і команда ***gzip***, однак стиснена версія файлу отримує розширення ***.bz2***.

```
debianuser@debianServ2017:~/dir1$ ls -l
total 20
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
-rw-r--r-- 1 debianuser debianuser  2376 Dec 30 13:30 passwd_copy.bz2
-rw-r--r-- 1 debianuser debianuser  2745 Dec 30 13:30 passwd_copy.gz
```

Наразі у нас є дві стиснені версії файлу ***passwd_copy***, одна утворилася в результаті виконання командою ***gzip***, а інша – командою ***bzip2***. Зі скріншоту, розташованого вище, можна помітити, що стиснена командою ***bzip2*** версія файлу за розміром є меншою, ніж версія, утворена командою ***gzip***.

Додамо, що команда ***bzip2*** підтримує всі ключі, які має команда ***gzip***, окрім ключа ***-l***. Виконана команда ***bzip2*** без жодного ключа стисне файл оригінального розміру зі ступенем стиснення ***6***, однак число, що вказує ступінь стиснення, для цієї команди має трохи інший сенс.

Для відновлення файлу-оригіналу з його стисненої версії командою ***bzip2*** призначена команда ***bunzip2***. Для перегляду відповідних стиснених файлів призначені команди ***bzcat*** та ***bzless***. Також існує команда для відновлення пошкоджених файлів формату ***.bz2***, яка має назву ***bzip2recover***.

6.3. Архівація файлів

У багатьох випадках разом з операцією стиснення використовується і операція архівації, є процесом збору множини файлів і пакування їх в один великий файл. Доволі часто архівація є одним з етапів створення

резервних копій системи. Зазвичай архівація використовується при переміщенні старих даних з системи в певне довгострокове сховище.

У системах Linux, а також багатьох інших UNIX-подібних, у якості класичного інструменту для архівації файлів виступає команда ***tar***. Її назва це інформація про те, що спочатку призначенням цієї команди було створення архівів на магнітних стрічках (***tar*** розшифровується як **tape archive**). Утім команда ***tar*** призначена тільки для архівації, але ніяк не для стиснення. Зазвичай у Linux (саме так і розповсюджуються вихідні коди програм) спочатку створюється архівний файл, який має розширення ***.tar***, а потім до нього застосовується команда ***gzip*** або ***bzip2***, після чого утворюється так званий стиснений архівний файл, який матиме розширення ***.tar.gz*** або ***.tar.bz2*** відповідно. Архівний файл може містити окремі файли, ієрархії каталогів або і те й інше.

Команда ***tar*** має наступний синтаксис:

tar -<режим> [-<параметр>] <елементи...>

Ключі команди ***tar*** поділяються на режими і параметри. Режими представляють ті ключі, які вказують на певні режими роботи. У табл. 6.8 представлені основні режими.

Табл. 6.8. Основні режими команди *tar*

Ключ	Призначення
-c	створити архів зі списку файлів і/або каталогів
-x	витягти файли з архіву до поточного каталогу
-r	додати вказаний файл і/або каталог у кінець архіву
-t	вивести список вмісту архіву.

Окрім ключів, що вказують на режими, є ключі, що представляють параметри. Одним з найбільш часто застосовуваних параметрів є ***-f***, який визначає назву архіву. Потрібно завжди пам'ятати, що при зазначенні ключів, першими мають вказуватися режими, а вже потім параметри. Інформацію щодо інших ключів та параметрів можна переглянути у довідковому керівництві ***man***.

Створення архіву

Для створення нового архіву в команді tar використовується ключ ***-c***. Звичайно команда створення архіву має наступний синтаксис:

tar -cf <ім'я_архіву> <список_елементів...>

Як приклад створимо архів, який міститиме файли, що є сценаріями командної оболонки (вони будуть розглянуті у наступних розділах). Їх

знаходять у каталозі ***scripts***, який зберігається у домашньому каталозі користувача ***debianuser***.

```
debianuser@debianServ2017:~/dir1$ tar -cf scripts.tar ../scripts
tar: Removing leading `../' from member names
debianuser@debianServ2017:~/dir1$ ls -l
total 60
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
-rw-r--r-- 1 debianuser debianuser  2376 Dec 30 13:30 passwd_copy.bz2
-rw-r--r-- 1 debianuser debianuser  2745 Dec 30 13:30 passwd_copy.gz
-rw-r--r-- 1 debianuser debianuser 40960 Dec 30 15:28 scripts.tar
```

Таким чином, був створений архівний файл під назвою ***scripts.tar***. Його розмір є сумарним розміром усіх файлів, що містяться у ньому.

За допомогою команди ***tar*** з ключем ***-t*** розглянемо, з яких файлів та каталогів складається архів:

```
debianuser@debianServ2017:~/dir1$ tar -tf scripts.tar
scripts/
scripts/until.sh
scripts/array.sh
scripts/for2.sh
scripts/easy
scripts/test_string.sh
scripts/condition.sh
scripts/for3.sh
scripts/graphic/
scripts/graphic/inputPassword.sh
scripts/graphic/menu.sh
scripts/graphic/yesno.sh
scripts/graphic/deleteFile.sh
scripts/first.sh
scripts/while.sh
scripts/arguments.sh
scripts/forC.sh
scripts/select.sh
scripts/value_exit.sh
scripts/test_number.sh
scripts/case.sh
scripts/country.txt
scripts/for1.sh
```

Для того, щоб витягти весь вміст архіву до поточного робочого каталогу, скористаємося командою ***tar*** з ключем ***-x***:

```
debianuser@debianServ2017:~/dir1$ tar -xf scripts.tar
debianuser@debianServ2017:~/dir1$ ls -l
total 64
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
-rw-r--r-- 1 debianuser debianuser  2376 Dec 30 13:30 passwd_copy.bz2
-rw-r--r-- 1 debianuser debianuser  2745 Dec 30 13:30 passwd_copy.gz
drwxr-xr-x 4 debianuser debianuser  4096 Nov 30 13:40 scripts
-rw-r--r-- 1 debianuser debianuser 40960 Dec 30 15:28 scripts.tar
```

Таким чином каталог ***scripts*** зі всіма файлами, що знаходилися у ньому, був витягнутий з архіву ***scripts.tar*** до поточного робочого каталогу.

Альтернативною програмою, яка здійснює архівацію файлів у ОС Linux, є ***zip***. До того ж вона стискає архівний файл, який створюється в результаті її виконання. Цей файл буде мати розширення ***.zip***.

Однак ***zip*** використовується у Linux не дуже часто, зокрема для обміну файлами з системами Windows, а не у якості основного інструменту стиснення та архівації файлів.

Синтаксис команди ***zip*** наступний:

zip -<ключі> <архівний_файл> <елементи...>

Для прикладу створимо командою ***zip*** архівний файл з уже наявних файлів ***passwd_copy*** та його двох стиснених версій за допомогою різних команд стиснення.

```
debianuser@debianServ2017:~/dir1$ ls -l
total 20
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
-rw-r--r-- 1 debianuser debianuser  2376 Dec 30 13:30 passwd_copy.bz2
-rw-r--r-- 1 debianuser debianuser  2745 Dec 30 13:30 passwd_copy.gz
debianuser@debianServ2017:~/dir1$ zip passwd.zip passwd_copy passwd_copy.bz2 pas
swd_copy.gz
  adding: passwd_copy (deflated 77%)
  adding: passwd_copy.bz2 (deflated 0%)
  adding: passwd_copy.gz (deflated 3%)
debianuser@debianServ2017:~/dir1$ ls -l
total 32
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
-rw-r--r-- 1 debianuser debianuser  2376 Dec 30 13:30 passwd_copy.bz2
-rw-r--r-- 1 debianuser debianuser  2745 Dec 30 13:30 passwd_copy.gz
-rw-r--r-- 1 debianuser debianuser  8248 Dec 30 16:02 passwd.zip
```

З наведеного прикладу видно, що під час виконання команди ***zip*** були виведені деякі повідомлення. Вони показують стан кожного файлу, який додається до архіву. Програма ***zip*** додає файли в архів, використовуючи один з двох методів: ***«store»*** – просте збереження без стиснення, та ***«deflate»*** – збереження зі стисненням.

Також можна створити zip-файл, вказавши у якості джерела даних не список файлів, а каталог. Утім при виконанні команди без жодного ключа до архіву буде включений тільки каталог без свого вмісту. Для того, щоб до zip-файлу був включений каталог разом зі своїм вмістом, варто скористатися ключем ***-r***, який і відповідає за рекурсивний обхід каталогів.

У якості прикладу створимо командою ***zip*** архівний файл, до якого включимо каталог ***scripts*** разом зі своїм вмістом.

```
debianuser@debianServ2017:~/dir1$ zip -r scripts.zip ../scripts
  adding: ../scripts/ (stored 0%)
  adding: ../scripts/until.sh (deflated 13%)
  adding: ../scripts/array.sh (deflated 37%)
  adding: ../scripts/for2.sh (deflated 16%)
  adding: ../scripts/easy (stored 0%)
  adding: ../scripts/test_string.sh (deflated 26%)
  adding: ../scripts/condition.sh (deflated 51%)
  adding: ../scripts/for3.sh (deflated 29%)
  adding: ../scripts/graphic/ (stored 0%)
  adding: ../scripts/graphic/inputPassword.sh (deflated 39%)
  adding: ../scripts/graphic/menu.sh (deflated 45%)
  adding: ../scripts/graphic/yesno.sh (deflated 30%)
  adding: ../scripts/graphic/deleteFile.sh (deflated 39%)
  adding: ../scripts/first.sh (deflated 6%)
  adding: ../scripts/while.sh (deflated 17%)
  adding: ../scripts/arguments.sh (deflated 36%)
  adding: ../scripts/forC.sh (deflated 1%)
  adding: ../scripts/select.sh (deflated 38%)
  adding: ../scripts/value_exit.sh (deflated 16%)
  adding: ../scripts/test_number.sh (deflated 47%)
```

Для того, щоб витягти файли з архівного файлу з розширенням ***.zip***, необхідно скористатися командою ***unzip***, результат виконання якої представлений у наступному прикладі.

```
debianuser@debianServ2017:~/dir1$ rm passwd_*
debianuser@debianServ2017:~/dir1$ ls -l
total 12
-rw-r--r-- 1 debianuser debianuser 8248 Dec 30 16:02 passwd.zip
debianuser@debianServ2017:~/dir1$ unzip passwd.zip
Archive:  passwd.zip
  inflating: passwd_copy
  inflating: passwd_copy.bz2
  inflating: passwd_copy.gz
debianuser@debianServ2017:~/dir1$ ls -l
total 32
-rw-r--r-- 1 debianuser debianuser 11980 Dec 30 13:30 passwd_copy
-rw-r--r-- 1 debianuser debianuser  2376 Dec 30 13:30 passwd_copy.bz2
-rw-r--r-- 1 debianuser debianuser  2745 Dec 30 13:30 passwd_copy.gz
-rw-r--r-- 1 debianuser debianuser  8248 Dec 30 16:02 passwd.zip
```

Таким чином, для перевірки правильності роботи команди ***unzip***, спочатку були видалені всі файли, що містять у своїй назві слово ***passwd***, а потім ці самі файли були витягнуті з zip-архіву.

Однак для того, щоб файли з архіву не витягувати, а тільки подивитися вміст архіву, потрібно виконати команду ***unzip*** з ключем ***-l***.

```
debianuser@debianServ2017:~/dir1$ unzip -l passwd.zip
Archive:  passwd.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
    11980  2018-12-30 13:30   passwd_copy
     2376  2018-12-30 13:30   passwd_copy.bz2
     2745  2018-12-30 13:30   passwd_copy.gz
---------                     -------
    17101                     3 files
```

Наостанок виділимо ще одну характерну особливість команди ***zip***. Якщо ця команда була виконана випадково, однак при цьому за якоїсь помилки було вказано назву архівного файлу, який вже збережений у системі, то відповідні файли, назви яких вказані наступними параметрами, будуть додані до існуючого архіву. На відміну команда ***tar*** при помилковому зазначенні архівного файлу, який вже існує у системі, просто перезапише файл заново.

Розділ 7.
БАГАТОЗАДАЧНІСТЬ У LINUX. ПРОЦЕСИ ТА ЇХ СТРУКТУРА

7.1. Поняття процесу

Станом на сьогодні багато сучасних операційних систем є багатозадачними. Вони характеризуються своєю спроможністю створення ілюзії одночасного вирішення декількох завдань, швидко переключаючись з виконання одного на інше. У цьому сенсі всі дистрибутиви ОС Linux не є виключенням та характеризуються своєю багатозадачністю, яка досягається шляхом керування виконанням завдань ядром Linux. За допомогою процесів воно організовує призупинення виконання програм для очікування черги центрального процесора.

Під час завантаження системи ядро Linux ініціює виконання завдань у вигляді процесів і запускає програму, що має назву **init.** Програма запускає на виконання спеціальні сценарії командної оболонки, що містяться у каталозі ***/etc***, які називаються **сценаріями початкового завантаження (init scripts)** та запускають усі системні служби. Багато цих служб представляють програми, які працюють без участі користувача, тобто у фоновому режимі. Подібні програми мають назви **програм-демонів (daemon programs)**. Тому, навіть якщо у певний момент часу не працює жодний кінцевий користувач, система виконує службові процедури.

Для впорядкування роботи всіх процесів ядро Linux зберігає інформацію про них. Кожен процес при запуску ідентифікується своїм унікальним числом, яке називається **ідентифікатором процесу (Process ID, PID)**. Такі числа присвоюються у порядку зростання, при цьому значення ідентифікатора процесу, що дорівнює 1, має процес init, оскільки завжди запускається першим.

Будь-який процес у системі може запускати інші процеси. Такий принцип виражається правилом: батьківський процес запускає дочірній процес. Натомість ядро Linux стежить за пам’яттю, яка виділяється кожному процесу, а також за їх готовністю процесів відновити виконання. Подібно до файлів та каталогів процеси мають ідентифікатори власника і користувача, ефективний (або діючий) ідентифікатор користувача і т. д.

7.1.1. Перегляд списку процесів

Для перегляду списку процесів частіше всього використовується команда ***ps***. Вона має досить багато опцій, однак спочатку розглянемо як

працює ця команда у найпростішому вигляді, тобто без жодного ключа та аргументу. Результат роботи команди ***ps*** представлений у наступному прикладі:

```
debianuser@debianServ2017:~$ ps
  PID TTY          TIME CMD
26405 pts/0    00:00:00 bash
26455 pts/0    00:00:00 ps
```

У вказаному вище прикладі було виведено список з двома процесами: PID=26405 та з PID=26455, що представляють програми ***bash*** і ***ps*** відповідно. Можна помітити, що за замовчуванням команда ***ps*** виводить зовсім небагато інформації, а саме інформацію щодо процесів, які пов'язані з поточним сеансом роботи. Інформація у свою чергу представляється полями ***PID***, ***TTY***, ***TIME*** та ***CMD***. Зрозуміло, що поле ***PID*** містить унікальний ідентифікатор процесу, а поле ***CMD***, що скороченням від **CoMmanD**, – назву програми. Поле ***TTY*** є скороченням від слова **TeleTYpe (телетайп)** та має інформацію про керуючий термінал процесу. Поле ***TIME*** містить обсяг процесорного часу, який був спожитий процесом. З наведеного прикладу чітко видно, що ані програма ***bash***, ані ***ps***, не є занадто обтяжливою для комп'ютера.

Щодо виконання команди ***ps*** з параметром ***x*** (звернемо особливу увагу, що перед ***x*** відсутній дефіс), можна отримати трохи більше інформації, розглянувши наступний приклад:

```
debianuser@debianServ2017:~$ ps x
  PID TTY      STAT   TIME COMMAND
26395 ?        Ss     0:00 /lib/systemd/systemd --user
26396 ?        S      0:00 (sd-pam)
26404 ?        R      0:00 sshd: debianuser@pts/0
26405 pts/0    Ss     0:00 -bash
26557 pts/0    R+     0:00 ps x
```

Аргумент **x** повідомляє команді **ps**, що вона має вивести усі процеси, якими володіє поточний користувач, незалежно від того, які термінали (якщо такі є) керують ними. На відсутність керуючого терміналу вказує символ ***?*** у полі ***TTY***. Однак це не єдине у виведенні, що додається аргументом ***x***. Також з'являється новий стовпець ***STAT***, що виводить інформацію щодо поточного стану процесу (***STAT*** є скороченням від **state**). Значенням стовпця ***STAT*** є один певний символ, який характеризує відповідний поточний стан процесу. У таблиці 7.1 представлені можливі символи, що можуть бути значеннями поля ***STAT***.

Табл. 7.1. Можливі значення поля `STAT`

Символ	Опис
R	**Діючий процес.** Процес виконується або готовий до виконання
S	**Призупинений процес.** Процес тимчасово не виконується; швидше за все, знаходиться в очікуванні певної події, такої як натискання клавіші або прибуття мережевого пакету
D	**Призупинений процес без можливості переривання**. Процес очікує завершення операції введення/виведення, наприклад, дисковим пристроєм
T	**Зупинений процес.** Процес примусово зупинено
Z	**Недіючий процес-«зомбі».** Це дочірній процес, який завершився, але не був видалений батьківським процесом
<	**Високопріоритетний процес.** Існує можливість найбільш важливим процесам виділити більше процесорного часу. Ця властивість процесу називається **niceness (поступливість)**. Щодо процесів з більш високими пріоритетами, то вони менш поступливі, тому що споживають більше процесорного часу, залишаючи менше іншим процесам
N	**Фоновий процес.** Процес з низьким пріоритетом (або поступливий процес) отримує процесорний час тільки після того, як будуть обслуговані процеси з більш високим пріоритетом.

Символ, що описує стан процесу, може супроводжуватися іншими символами, які зазвичай відображають певні екзотичні характеристики процесів.

Більше інформації можна отримати під час виконання команди ***ps*** з різними параметрами. Команда ***ps*** підтримує три різних типи параметрів командного рядка:

1. Параметри в стилі UNIX, перед якими ставиться дефіс;
2. Параметри в стилі BSD, перед якими не ставиться дефіс;
3. Довгі параметри GNU, перед якими ставляться два дефіси.

Параметри в стилі UNIX вперше були розроблені для початкової версії команди ***ps***, призначеної для систем AT&T UNIX, які було створено в корпорації Bell Labs. Основні параметри представлені у табл. 7.2.

Табл. 7.2. Основні параметри команди `ps` у стилі UNIX

Параметр	Опис
-a	показувати всі процеси, крім заголовків сеансів і процесів, що виконуються поза терміналами
-d	показувати всі процеси, крім заголовків сеансів
-e	показувати всі процеси
-g <grplist>	показувати процеси за сеансами або за ідентифікаторами груп, перерахованих у списку ***grplist***
-p <pidlist>	показувати процеси з ідентифікаторами процесів, перерахованими в списку ***pidlist***
-u <userlist>	показувати процеси за дійсним кодом користувача, перерахованим в списку ***userlist***

Закінчення табл 7.2

`-f`	відображати лістинг у повному форматі
`-j`	показувати інформацію про завдання
`-l`	відображати довгий лістинг
`-o <format>`	відображати тільки конкретні стовпці, перераховані в списку ***format***
`-w`	використовувати широкий вихідний формат, призначений для дисплеїв з необмеженою шириною виведення.

Системні адміністратори Linux зазвичай вибирають власні набори параметрів, які вони застосовують та завжди можуть пригадати, отримавши необхідну інформацію. Наприклад, якщо виникає необхідність переглядати всі процеси, що працюють у системі, то можна скористатися комбінацією параметрів **`-ef`**, як у наступному прикладі:

```
debianuser@debianServ2017:~$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Jan24 ?        00:00:04 /sbin/init
root         2     0  0 Jan24 ?        00:00:00 [kthreadd]
root         3     2  0 Jan24 ?        00:00:54 [ksoftirqd/0]
root         5     2  0 Jan24 ?        00:00:00 [kworker/0:0H]
root         7     2  0 Jan24 ?        00:03:24 [rcu_sched]
root         8     2  0 Jan24 ?        00:00:00 [rcu_bh]
root         9     2  0 Jan24 ?        00:00:00 [migration/0]
root        10     2  0 Jan24 ?        00:00:00 [lru-add-drain]
root        11     2  0 Jan24 ?        00:00:02 [watchdog/0]
root        12     2  0 Jan24 ?        00:00:00 [cpuhp/0]
root        13     2  0 Jan24 ?        00:00:00 [kdevtmpfs]
root        14     2  0 Jan24 ?        00:00:00 [netns]
root        15     2  0 Jan24 ?        00:00:00 [khungtaskd]
root        16     2  0 Jan24 ?        00:00:00 [oom_reaper]
root        17     2  0 Jan24 ?        00:00:00 [writeback]
root        18     2  0 Jan24 ?        00:00:00 [kcompactd0]
root        19     2  0 Jan24 ?        00:00:00 [ksmd]
root        21     2  0 Jan24 ?        00:00:00 [khugepaged]
root        22     2  0 Jan24 ?        00:00:00 [crypto]
root        23     2  0 Jan24 ?        00:00:00 [kintegrityd]
root        24     2  0 Jan24 ?        00:00:00 [bioset]
root        25     2  0 Jan24 ?        00:00:00 [kblockd]
root        26     2  0 Jan24 ?        00:00:00 [devfreq_wq]
```

У зазначеному вище прикладі використовуються два параметри: **`-e`**, який показує всі процеси, що працюють у системі, і **`-f`**, що забезпечує виведення в широкому форматі, з включенням декількох корисних стовпців інформації.

Далі розглянемо параметри команди ***ps*** у стилі BSD. Berkeley Software Distribution (BSD) – це версія UNIX, розроблена Каліфорнійським університетом м. Берклі. Вона має багато незначних відмінностей від системи AT&T Unix, і саме їх існування було причиною істотних розбіжностей у співтоваристві користувачів UNIX, які спостерігалися минулими роками. Основні параметри команди ***ps*** в стилі BSD наведені в табл. 7.3.

Табл. 7.3. Основні параметри команди `ps` у стилі BSD

Параметр	Опис
T	Показувати всі процеси, асоційовані з поточним терміналом
a	Показувати всі процеси, асоційовані з будь-яким терміналом
r	Показувати тільки діючі процеси
x	Показувати всі процеси, навіть ті, для яких не призначено термінальний пристрій
U <userlist>	Показувати процеси, що належать користувачам з ідентифікаторами, перерахованими в списку ***userlist***
p <pidlist>	Показувати процеси зі значеннями PID, перерахованими в списку ***pidlist***
j	Показувати інформацію про завдання
o <format>	Відображати тільки стовпці, зазначені в списку ***format***
u	Застосовувати формат, орієнтований на користувача
S	Підсумувати чи відноситься до дочірніх процесів числова інформація, така як використання процесора і пам’яті, з прив’язкою до батьківського процесу
c	Відображати справжню назву команди (програми, яка використовувалася для запуску процесу)
f	Відображати процеси в ієрархічному форматі, показавши, які процеси були запущені тими чи іншими процесами
h	Не додавати інформацію заголовка
k <sort>	Визначити один або декілька стовпців для використання під час сортування вихідних даних
w	Формувати широкий висновок для більш широких терміналів.

Таким чином, розглянутий приклад з виконанням команди ***ps*** з параметром ***x*** без дефіса є одним з варіантів використання цієї команди з параметрами в стилі BSD. Як і у випадках використання параметрів у стилі UNIX, та BSD їх також можна об’єднувати. Досить популярною комбінацією є ***aux***, яка складається з трьох параметрів, що описані у табл. 7.3. Результат виконання команди з цією сукупністю параметрів представлений у наступному прикладі:

```
debianuser@debianServ2017:~$ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.3 139096  6876 ?        Ss    2018   0:06 /sbin/init
root          2  0.0  0.0      0     0 ?        S     2018   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S     2018   1:16 [ksoftirqd/0]
root          5  0.0  0.0      0     0 ?        S<    2018   0:00 [kworker/0:0H]
root          7  0.0  0.0      0     0 ?        S     2018   5:28 [rcu_sched]
root          8  0.0  0.0      0     0 ?        S     2018   0:00 [rcu_bh]
root          9  0.0  0.0      0     0 ?        S     2018   0:00 [migration/0]
root         10  0.0  0.0      0     0 ?        S<    2018   0:00 [lru-add-drain]
root         11  0.0  0.0      0     0 ?        S     2018   0:05 [watchdog/0]
root         12  0.0  0.0      0     0 ?        S     2018   0:00 [cpuhp/0]
root         13  0.0  0.0      0     0 ?        S     2018   0:00 [kdevtmpfs]
root         14  0.0  0.0      0     0 ?        S<    2018   0:00 [netns]
root         15  0.0  0.0      0     0 ?        S     2018   0:00 [khungtaskd]
root         16  0.0  0.0      0     0 ?        S     2018   0:00 [oom_reaper]
root         17  0.0  0.0      0     0 ?        S<    2018   0:00 [writeback]
root         18  0.0  0.0      0     0 ?        S     2018   0:00 [kcompactd0]
root         19  0.0  0.0      0     0 ?        SN    2018   0:00 [ksmd]
root         21  0.0  0.0      0     0 ?        SN    2018   0:00 [khugepaged]
root         22  0.0  0.0      0     0 ?        S<    2018   0:00 [crypto]
root         23  0.0  0.0      0     0 ?        S<    2018   0:00 [kintegrityd]
root         24  0.0  0.0      0     0 ?        S<    2018   0:00 [bioset]
root         25  0.0  0.0      0     0 ?        S<    2018   0:00 [kblockd]
```

Оскільки в системі одночасно виконується достатньо велика кількість процесів, команда ***ps*** може вивести досить довгі списки. Іноді буде корисно передати результат виконання команди ***ps*** команді ***less*** конвеєром для більш зручного перегляду або перенаправити результат виконання команди ***ps*** у файл.

Наостанок розглянемо останній тип параметрів, які можуть бути використані командою ***ps***. У кінцевому підсумку розробники програм GNU доклали власні зусилля до створення нової, поліпшеної команди ***ps***, а також додали ще кілька параметрів до загальної сукупності параметрів. Частина довгих параметрів GNU практично повторює існуючі параметри в стилі BSD або UNIX, тоді як інші надають нові можливості. У табл. 7.4 перераховані основні довгі параметри GNU, які є в розпорядженні користувача.

Табл. 7.4. Основні параметри команди *ps* у стилі GNU

Параметр	Опис
--deselect	Показати всі процеси, крім перерахованих у командному рядку
--group <grplist>	Показати процеси, дійсні ідентифікатори групи яких перераховані в списку ***grplist***
--pid <pidlist>	Показати процеси, ідентифікатори процесу яких перераховані в списку ***pidlist***
--ppid <pidlist>	Показати процеси, ідентифікатори батьківського процесу яких перераховані в списку ***pidlist***
--user <userlist>	Показати процеси, ефективний ідентифікатор користувача яких перераховано в списку ***userlist***
--format <format>	Відобразити тільки стовпці, зазначені в списку ***format***
--cumulative	Включити інформацію щодо зупинення дочірніх процесів
--forest	Відобразити процеси у вигляді ієрархічного лістингу, що показує батьківські процеси
--headers	Повторити заголовки стовпців на кожній сторінці виведення
--no-headers	Не додавати заголовки стовпців
--sort <order>	Визначити один або декілька стовпців для використання при сортуванні вихідних даних.

У складі довгих параметрів GNU передбачено один дійсно потужний параметр ***–forest***, який забезпечує відображення інформації щодо процесів у вигляді ієрархічної структури, але з використанням символів ASCII, в результаті чого формуються візуально привабливі графіки. Покажемо це на прикладі:

```
debianuser@debianServ2017:~$ ps aux --forest
message+   343  0.0  0.2  45496  4172 ?        Ss   Jan24   0:00 /usr/bin/dbus-daemon -
root       386  0.0  0.3 286136  6368 ?        Ssl  Jan24   0:00 /usr/lib/policykit-1/p
systemd+   419  0.0  0.2 123072  4068 ?        Ssl  Jan24   0:00 /lib/systemd/systemd-t
root       464  0.0  0.2  40868  4436 ?        Ss   Jan24   0:46 /usr/sbin/lircd --noda
root       465  0.0  0.0  19516  1832 ?        Ss   Jan24   0:00 /usr/sbin/lircd-uinput
root       468  0.0  0.3  69952  6172 ?        Ss   Jan24   0:00 /usr/sbin/sshd -D
root     22879  0.0  0.3  99348  6816 ?        Ss   12:57   0:00  \_ sshd: debianuser [
debianu+ 22932  0.0  0.2  99348  4140 ?        S    12:58   0:00      \_ sshd: debianus
debianu+ 22933  0.0  0.2  20940  4928 pts/0    Ss   12:58   0:00          \_ -bash
debianu+ 23562  0.0  0.1  38456  3384 pts/0    R+   13:11   0:00              \_ ps aux
root       470  0.0  0.0  36572   484 ?        S    Jan24   0:00 /usr/sbin/xrdp-sesman
root       478  0.0  0.0  14536  1728 tty1     Ss+  Jan24   0:00 /sbin/agetty --noclear
root       482  0.0  0.3 363692  6408 ?        Ssl  Jan24   0:00 /usr/sbin/lightdm
root       490  0.0  2.2 369232 45172 tty7     Ssl+ Jan24   0:11  \_ /usr/lib/xorg/Xorg
root       707  0.0  0.3 233260  6300 ?        Sl   Jan24   0:00  \_ lightdm --session-
lightdm    819  0.0  3.2 674124 65156 ?        Ssl  Jan24   1:43  |   \_ /usr/sbin/ligh
root       934  0.0  0.2  87496  4576 ?        S    Jan24   0:00  \_ lightdm --session-
xrdp       505  0.0  0.0  24560   540 ?        S    Jan24   0:00 /usr/sbin/xrdp
root       513  0.0  0.4 230352  9200 ?        Ss   Jan24   1:25 /usr/sbin/nmbd
root       520  0.0  0.7 318144 15260 ?        Ss   Jan24   0:00 /usr/sbin/smbd
root       521  0.0  0.2 310020  5628 ?        S    Jan24   0:00  \_ /usr/sbin/smbd
root       522  0.0  0.2 310036  5528 ?        S    Jan24   0:00  \_ /usr/sbin/smbd
root       525  0.0  0.3 318128  7204 ?        S    Jan24   0:00  \_ /usr/sbin/smbd
lightdm    762  0.0  0.3  64956  6708 ?        Ss   Jan24   0:00 /lib/systemd/systemd -
```

Таким чином, за допомогою команди ***ps*** можна отримати повну інформацію про процеси, що виконуються у системі. Однак для цього існує більш зручний засіб, який буде розглянуто далі.

7.1.2. Перегляд стану процесів у динаміці

Незважаючи на те, що команда **ps** надає достатньо багато інформації щодо роботи з комп'ютером, вона все ж має суттєвий недолік. Він полягає у тому, що результат команди ***ps*** фіксує лише миттєвий знімок, тобто вихідна інформація є дійсною тільки на момент виклику команди. Іноді досить часто доводиться відслідковувати інформацію про процеси у системі в реальному часі. Для цього в ОС Linux призначена програма ***top***.

Команда постійно оновлює інформацію про процеси (за замовчуванням з періодичністю в 3 секунди), щоб показати їх активність з плином часу. Виведення команди **top** поділяється на дві частини:

1. Зведена інформація про систему;
2. Таблиця процесів, відсортованих за споживанням ними процесорного часу.

Розглянемо результат виконання команди ***top*** у наступному прикладі:

```
debianuser@debianServ2017:~$ top
```

```
top - 16:14:58 up 6 days, 55 min,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  97 total,   1 running,  96 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.7 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem :  2028064 total,   761996 free,   109048 used,  1157020 buff/cache
KiB Swap:  2050044 total,  2050044 free,        0 used.  1706776 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  189 root      20   0   59472   7904   7352 S   0.3  0.4  16:10.94 systemd-jou+
  335 root      20   0  284092   6960   5564 S   0.3  0.3   2:37.13 accounts-da+
27075 debianu+  20   0   44788   3568   3000 R   0.3  0.2   0:00.04 top
    1 root      20   0  139096   6876   5220 S   0.0  0.3   0:06.26 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.06 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   1:16.27 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0.0  0.0   5:28.59 rcu_sched
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 migration/0
   10 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 lru-add-dra+
   11 root      rt   0       0      0      0 S   0.0  0.0   0:05.41 watchdog/0
   12 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kdevtmpfs
   14 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 netns
   15 root      20   0       0      0      0 S   0.0  0.0   0:00.30 khungtaskd
   16 root      20   0       0      0      0 S   0.0  0.0   0:00.00 oom reaper
```

Угорі представляється зведена інформація про систему, яка складається з 5 рядків. Розглянемо детально, з чого вона складається.

1-й рядок: назва програми (***top***); поточний час (***16:14:58***); тривалість роботи системи (***up 6 days, 55 min***); кількість користувачів, працюючих у системі в цей момент (***1 user***); середня навантаженість – число процесів, що очікують відновлення роботи за останню хвилину, 5 хвилин та 15 хвилин (значення, що є нижчими за 1, означають, що система не навантажена);

2-й рядок: загальна кількість процесів у різних станах (всього процесів, діючих, сплячих, примусово зупинених та «зомбі»);

3-й рядок: процесорний час (у відсотках), що був витрачений на виконання різних типів процесів;

4-й рядок: об’єм використовуваної оперативної пам’яті (ОЗП);

5-й рядок: об’єм використовуваного простору у файлі підкачки (віртуальної пам’яті).

Наступний розділ являє собою таблицю процесів, що виконуються у поточний час. Таблиця включає певні стовпці, які багато у чому аналогічні стовпцям при виведенні програмою ***ps***. Опис відповідних полів наведений у табл. 7.5.

Табл. 7.5. Стовпці таблиці програми *top*

Стовпець	Опис
PID	Ідентифікатор процесу, що розглядається
USER	Ім'я користувача, відповідне власнику процесу
PR	Пріоритет процесу
NI	Величина зміни пріоритету процесу
VIRT	Загальний обсяг віртуальної пам'яті, яку використовує процес
RES	Обсяг фізичної пам'яті, яку використовує процес
SHR	Обсяг пам'яті, яку процес використовує спільно з іншими процесами
S	Поточний стан процесу (див. табл. 7.1)
%CPU	Частка процесорного часу, яку використовує процес
%MEM	Частка фізичної пам'яті, яку використовує процес
TIME+	Загальна кількість процесорного часу, витраченого процесом з моменту запуску
COMMAND	Назва процесу, задана в командному рядку (запущена програма).

За замовчуванням після запуску команди ***top*** сортування процесів здійснюється з урахуванням значення ***%CPU***. Однак можна перейти до іншого порядку сортування з використанням однієї з декількох інтерактивних підкоманд команди ***top***.

Кожна інтерактивна підкоманда має окремий символ, який можна ввести в ході виконання програми. Виконання інтерактивної підкоманди тим чи іншим чином призведе до зміни поведінки програми ***top***. Основні інтерактивні підкоманди наведені в табл. 7.6.

Табл. 7.6. Інтерактивні підкоманди команди *top*

Підкоманда	Опис
l	Переключитися на відображення рядка інформації про середнє завантаження
t	Переключитися на відображення рядка інформації про процесор
m	Переключитися на відображення рядків інформації **MEM** і **SWAP**
f	Додати або видалити ті чи інші інформаційні стовпці
o	Змінити порядок відображення інформаційних стовпців
F або ***O***	Вибрати поле, у якому будуть відсортовані процеси (за замовчуванням застосовується ***%CPU***)
< або **>**	Перемістити поле сортування на один стовпець ліворуч (**<**) або праворуч (**>**)
r	Переключитися на звичайний або зворотний порядок сортування
c	Переключитися на відображення для процесів назви команди або повного командного рядка (включаючи параметри)
i	Переключитися на відображення процесів, що простоюють
S	Переключитися на відображення сукупного процесорного часу або відносного процесорного часу
x	Переключитися на виділення підсвічуванням поля сортування
y	Переключитися на виділення підсвічуванням працюючих завдань

Закінчення табл. 7.6

u	Показати процеси для конкретного користувача
n або ***#***	Задати кількість відображуваних процесів
k	Знищити конкретний процес (підкоманда доступна тільки власнику процесу або користувачеві **root**)
r	Змінити (**renice**) пріоритет конкретного процесу (підкоманда доступна тільки власнику процесу або користувачеві **root**)
d або ***s***	Змінити інтервал оновлення (значення за замовчуванням – 3 секунди)
W	Записати поточні значення параметрів у файл конфігурації
q	Вийти з команди ***top***.

Використання команди ***top*** часто дозволяє знаходити процеси, які діють у системі та порушують нормальну роботу. Після виявлення такого процесу наступним завданням стає припинення роботи, що і буде розглянуто далі.

7.2. Управління процесами

Однак спочатку розглянемо різні засоби управління процесами, що включає у себе призупинення процесу, переведення його у фоновий режим, повернення на передній план, а також остаточна його зупинка.

Виконання процесів у певний момент часу можна перервати. Для цього призначена комбінація клавіш ***<CTRL>+<C>*** у терміналі. Саме таким способом можна перервати виконання багатьох програм командного рядка, однак не всіх.

Іноді виникає потреба перевести певний процес у фоновий режим. Звичайно така дія може бути потрібна для програм, що не повертають автоматично управління командній оболонці (***less***, ***vi***, ***nano***, ***top*** і т. п.). Можна запустити процес відразу в фоновому режимі. Це виконується простим додаванням у кінець команди символу амперсанда (***&***) після пробілу. Наприклад, запустимо редактор ***vi*** у фоновому режимі.

```
debianuser@debianServ2017:~$ vi &
[1] 27464
```

Після введення вищезазначеної команди вікно редактора ***vi*** не відкриється, замість цього командна оболонка повернеться у запрошення до введення інших команд, однак перед цим буде виведена певна інформація, яка представляє собою два числа. Саме таким способом командна оболонка повідомила, що ми запустили завдання з номером 1 (перше число в квадратних дужках), яке отримало ідентифікатор процесу 27464. Повідомлення є **частиною механізму управління завданнями (job control)**. При виконанні команди ***ps*** інформація про процес, що представляє собою редактор ***vi***, вже буде виведена.

```
debianuser@debianServ2017:~$ ps
  PID TTY          TIME CMD
26405 pts/0    00:00:00 bash
27464 pts/0    00:00:00 vi
27564 pts/0    00:00:00 ps

[1]+  Stopped                 vi
```

Механізм управління завданнями також дає можливість вивести список завдань, запущених у терміналі. Для цього призначена команда ***jobs***.

```
debianuser@debianServ2017:~$ jobs
[1]+  Stopped                 vi
```

У результаті виконання команди ***jobs*** буде виведена інформація щодо завдання, яке виконується. Воно було запущено командою ***vi &*** та має номер 1.

Однак процес у фоновому режимі не може отримувати введення з клавіатури, а також його не можна перервати за допомогою комбінації ***<CTRL>+<C>***. Для того, щоб процес міг реагувати на введення з клавіатури, його потрібно повернути на передній план. Для цього використовується команда ***fg***. У наступному прикладі повернемо на передній план запущений до цього у фоновому режимі редактор ***vi***.

```
debianuser@debianServ2017:~$ fg %1
~                           VIM - Vi IMproved
~
~                            version 8.0.707
~                       by Bram Moolenaar et al.
~           Modified by pkg-vim-maintainers@lists.alioth.debian.org
~                 Vim is open source and freely distributable
~
~                        Become a registered Vim user!
~                 type  :help register<Enter>   for information
~
~                 type  :q<Enter>               to exit
~                 type  :help<Enter>  or  <F1>  for on-line help
~                 type  :help version8<Enter>   for version info
~
~
~
~
~                                                          0,0-1         All
```

Таким чином, редактор ***vi*** був переведений на передній план. Як видно з прикладу, була виконана команда ***fg*** з певним аргументом, який складається зі знаку відсотка (***%***) та номера завдання. Така комбінація має назву **специфікатора завдання** (**jobspec**). Його можна опустити, якщо існує тільки одне фонове завдання.

Іноді виникає потреба не завершувати процес, а тільки призупинити його на деякий час. Це потрібно виконувати для переведення процесу з переднього у фоновий режим. Для його призупинення призначена

комбінація клавіш ***<CTRL>+<Z>***. Далі можна або повернути програму на передній план за допомогою команди ***fg***, або перевести її у фоновий режим за допомогою команди ***bg***.

У наступному прикладі переведемо редактор ***vi*** у фоновий режим:

```
[1]+  Stopped                 vi
debianuser@debianServ2017:~$ bg %1
[1]+ vi &
```

Команда ***bg*** має такий самий аргумент, як і команда ***fg***. Специфікатор завдання можна опустити, якщо на поточний момент воно тільки одне.

Можливість переведення у фоновий режим корисна і в тому випадку, якщо при запуску програми з графічним інтерфейсом з командного рядка користувач випадково не додав у кінець команди символ ***&***.

7.2.1. Зупинка процесів

Одним з найважливіших завдань системного адміністратора є виконання своєчасної та правильної зупинки процесів. Іноді процес призупиняється і вимагає лише невеликого втручання для продовження своєї роботи або повного зупинення. Однак можуть трапитися і ситуації, коли процес стає некерованим, і не може бути припинений звичайним способом і до того ж безмірно поглинає процесорний час. У такому випадку для управління процесом необхідно скористатися певною командою.

У системі Linux використовується такий же спосіб організації взаємодії між процесами, як і в системі UNIX: процеси обмінюються даними один з одним за допомогою сигналів. Сигналом процесу є стандартне повідомлення, яке розпізнається процесом, після отримання якого він може вибрати, чи виконати відповідну до сигналу дію або взагалі ігнорувати його. Застосовувані в процесі способи обробки сигналів закладаються розробниками при написанні програми. У більшості якісно написаних програмних застосунків передбачена здатність отримувати стандартні сигнали процесу UNIX і діяти відповідно до цих сигналів.

Система Linux передбачає дві описані нижче команди, що дозволяють відправляти сигнали працюючим процесам: ***kill*** та ***killall***.

Команда ***kill*** відправляє сигнали процесам з урахуванням їх ідентифікатора (PID). Повний список сигналів, які відправлятимуться цією командою можна отримати за допомогою її виконання разом з ключем ***-l***. Саме це показано у наступному прикладі:

```
debianuser@debianServ2017:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Таким чином у системі Linux доступні 64 сигнали. Їх прийнято позначати номерами або символічними назвами, що починаються на **SIG**, але іноді приставку пропускають: наприклад, сигнал за номером 1 позначають як **SIGHUP**, або просто **HUP**.

У табл.7.7 наведені найбільш використовувані в ОС Linux сигнали разом з їх описом.

Табл. 7.7. Найбільш використовувані сигнали процесів Linux

№	Назва	Опис	Можливість перехоплення	Можливість блокування	Комбінація клавіш
1	***HUP***	**Hangup.** Відбій	Так	Так	
2	***INT***	**Interrupt.** У разі виконання простих команд викликає припинення виконання, в інтерактивних програмах – припинення активного процесу	Так	Так	***<Ctrl>+ <C> або <Del>***
3	***QUIT***	Припинити виконання. Як правило, сильніше сигналу Interrupt	Так	Так	***<Ctrl>+ <\>***
4	***ILL***	**Illegal Instruction.** Центральний процесор отримав незнайому команду (у більшості випадків це означає, що допущена програмна помилка). Сигнал відправляється програмі, в якій виникла проблема	Так	Так	
8	***FPE***	**Floating Point Exception.** Обчислювальна помилка, наприклад, ділення на нуль	Так	Так	
9	***KILL***	Завжди припиняє виконання процесу	Ні	Ні	
11	***SEGV***	**Segmentation Violation.** Доступ до недозволеної області пам’яті	Так	Так	
13	***PIPE***	Було зроблено спробу передачі даних за допомогою конвеєра або черги FIFO, проте не існує процесу, здатного прийняти ці дані	Так	Так	
15	***TERM***	**Software Termination.** Вимога закінчити процес (програмне завершення)	Так	Так	

Закінчення табл. 7.7

17	***CHLD***	Зміна статусу навяного процесу	Так	Так	
18	***CONT***	Продовження виконання припиненого процесу	Так	Так	
19	***STOP***	Призупинення виконання процесу	Ні	Ні	
20	***TSTP***	Сигнал зупинки, що генерується клавіатурою. Переводить процес у фоновий режим.	Так	Так	***<Ctrl>+ <Z>***

За замовчуванням команда ***kill*** відправляє сигнал ***TERM*** усім процесам з ідентифікаторами процесів, перерахованими в командному рядку. На жаль, передбачена можливість використовувати лише **PID** процесу замість назви команди, за допомогою якої він був запущений, тому іноді завдання правильного виконання команди ***kill*** стає досить незручним.

Щоб мати право передати сигнал процесу, необхідно бути або його власником або користувачем **root**. У наступному прикладі показано невдалу спробу завершити процес, власником якого є **root**, звичайним користувачем, яким є **debianuser**.

```
debianuser@debianServ2017:~$ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.3 139096  6876 ?        Ss    2018   0:06 /sbin/init
root          2  0.0  0.0      0     0 ?        S     2018   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S     2018   1:16 [ksoftirqd/0]
root          5  0.0  0.0      0     0 ?        S<    2018   0:00 [kworker/0:0H]
root          7  0.0  0.0      0     0 ?        S     2018   5:30 [rcu_sched]
root          8  0.0  0.0      0     0 ?        S     2018   0:00 [rcu_bh]
root          9  0.0  0.0      0     0 ?        S     2018   0:00 [migration/0]
root         10  0.0  0.0      0     0 ?        S<    2018   0:00 [lru-add-drain]
root         11  0.0  0.0      0     0 ?        S     2018   0:05 [watchdog/0]
root         12  0.0  0.0      0     0 ?        S     2018   0:00 [cpuhp/0]
root         13  0.0  0.0      0     0 ?        S     2018   0:00 [kdevtmpfs]
root         14  0.0  0.0      0     0 ?        S<    2018   0:00 [netns]
root         15  0.0  0.0      0     0 ?        S     2018   0:00 [khungtaskd]
root         16  0.0  0.0      0     0 ?        S     2018   0:00 [oom_reaper]
root         17  0.0  0.0      0     0 ?        S<    2018   0:00 [writeback]
root         18  0.0  0.0      0     0 ?        S     2018   0:00 [kcompactd0]
root         19  0.0  0.0      0     0 ?        SN    2018   0:00 [ksmd]
root         21  0.0  0.0      0     0 ?        SN    2018   0:00 [khugepaged]
root         22  0.0  0.0      0     0 ?        S<    2018   0:00 [crypto]
root         23  0.0  0.0      0     0 ?        S<    2018   0:00 [kintegrityd]
root         24  0.0  0.0      0     0 ?        S<    2018   0:00 [bioset]
root         25  0.0  0.0      0     0 ?        S<    2018   0:00 [kblockd]
    debianuser@debianServ2017:~$ kill 24
    -bash: kill: (24) - Operation not permitted
```

Сигнал ***TERM*** вимагає коректного завершення процесу, якщо це можливо, в іншому випадку він може бути проігнорованим. Зазвичай такий випадок відбувається з тими процесами, що вийшли з-під контролю. Для примусового завершення процесу потрібно командою ***kill*** застосувати сигнал ***–s***, який дозволяє застосування й інших сигналів.

У загальному випадку з командою ***kill*** можна вказати інший сигнал за допомогою наступного синтаксису:

kill -<назва_сигналу> <PID_процесу> [<PID_процесу…>], або

kill -<назва_сигналу> <PID_процесу> [<PID_процесу…>]

Для завершення процесу прийнято спочатку застосовувати сигнал ***TERM*** (номер ***15***). При ігноруванні сигналу процесом, виконується спроба відправлення сигналу ***INT*** (номер ***2***) або ***HUP*** (номер ***1***). Якщо програма-процес розпізнає сигнали, то вона робить спробу коректного завершення всіх операцій, що виконуються. Сигнал ***KILL*** (номер ***9***) має найвищий ступінь примусового завершення, після отримання якого процес негайно припиняє виконання. Жодний процес ніколи не може його проігнорувати. Однак застосовувати сигнал ***KILL*** потрібно досить обережно, оскільки це може призвести до спотворення інформації у файлах.

Іншою командою, що призначена для припинення виконання процесів, є команда ***killall***. Вона також діє, як і команда ***kill***, та має практично такий самий синтаксис. Відмінність полягає лише в тому, що в команді ***killall*** замість **PID** процесів задаються назви програм. Наприклад, зупинити всі процеси, що пов’язані з текстовим редактором ***vi***, можна у такий спосіб:

killall vi

Таким чином, будуть закриті всі файлі, що були відкритими для редагування у програмі ***vi***.

7.3. Планування регулярного виконання завдань

У системі Linux є механізм планування завдань, які кінцеві користувачі можуть призначити для регулярного виконання через певний період часу. Для цього призначена спеціальна програма **cron**, яка виконується у фоновому режимі. Вона перевіряє спеціальні таблиці на наявність призначених завдань для регулярного виконання. Ці таблиці використовують спеціальний формат, за допомогою якого вказується, у який момент часу завдання має бути виконано.

Формат *cron*-таблиць має 6 колонок, у кожної з яких своє окреме призначення, а саме:

<хвилина> <година> <день_місяця> <місяць> <день_тижня> <команда>

Cron-таблиця дозволяє задавати записи в якості конкретних значень або діапазонів значень (наприклад, таких як 1–5), а також символів-шаблонів (зірочок). Наприклад, якщо певна команда має виконуватися щодня о 12:30, то можна задати наступний запис у *cron*-таблиці:

30 12 * * * <команда>

Символи-шаблони, які використовуються в полях ***<день_місяця>***, ***<місяць>*** і ***<день_тижня>***, вказують на те, що програма **cron** має виконувати команду кожен день кожного місяця о 12:30.

Поле ***<день_тижня>*** може бути задано у вигляді трьохсимвольного текстового значення (***mon***, ***tue***, ***wed***, ***thu***, ***fri***, ***sat***, ***sun***) або числового від 0, що відповідає неділі, до 6, що відповідає суботі. Наприклад, для того, щоб вказати, що команда має виконуватися о 17:40 кожної середи, можна зробити такий запис:

```
40 17 * * 3 <команда>
```

Натомість поле ***<день_місяця>*** вказує значення числа дня місяця, що може приймати значення від 1 до 31, а поле ***<місяць>*** може мати значення від 0 до 11, де 0 відповідає січню, а 11 – грудню. У наступному прикладі показане завдання команди на виконання о 14:30 першого числа кожного місяця:

```
30 14 1 * * <команда>
```

У полі, що відповідає назві команди, також передбачена можливість задавати будь-які ключі та аргументи, а також оператори перенаправлення, як зазвичай у командному рядку. У перших п’яти полях можна зазначати символ «**/**», який вказує на додаткову періодичність завдання. Наприклад, «****/3***» в першому полі означає «кожні 3 хвилини».

У наступному прикладі задамо виконання команди ***ls –l*** з перенаправленням її результату у файл ***file1.txt*** кожні 3 хвилини.

```
*/3 * * * * ls –l > file1.txt
```

Програма **cron** виконує сценарій із застосуванням облікового запису користувача, який передав завдання на виконання. Таким чином, користувач матиме належні дозволи на доступ до команди і до вихідних файлів, що зазначені у виклику команди.

Кожен користувач системи (включаючи користувача **root**) може мати власну *cron*-таблицю, призначену для виконання запланованих завдань. У Linux для обробки *cron*-таблиці передбачена команда ***crontab***. Для виведення існуючої таблиці **cron** на зовнішній пристрій використовується параметр ***-l***:

```
debianuser@debianServ2017:~$ crontab -l
no crontab for debianuser
debianuser@debianServ2017:~$ crontab forCron
debianuser@debianServ2017:~$ crontab -l
* * * * * vmstat >> stat.txt
*/3 * * * * ls -l > file1.txt
```

На рис. 7.1 представлений файл ***forCron***, відкритий у текстовому редакторі **nano**.

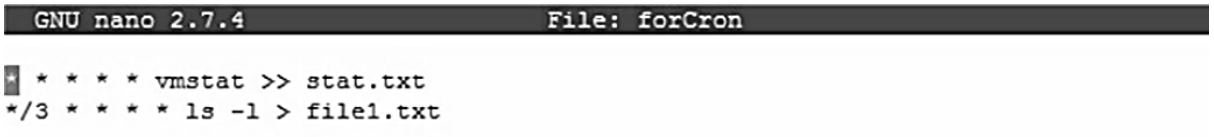

Рис. 7.1. Файл *cron*-таблиці, відкритий у редакторі **nano**

За замовчуванням файл *cron*-таблиці не створюється автоматично для кожного користувача. Для додавання записів у *cron*-таблицю застосовується ключ **`-e`**. Команда **`crontab`** після виклику з цим параметром запускає текстовий редактор стосовно існуючої *cron*-таблиці (або до порожнього файлу, якщо вона ще не була створена).

```
debianuser@debianServ2017:~$ crontab -e
```

На рис. 7.2 представлений результат наведеної вище команди.

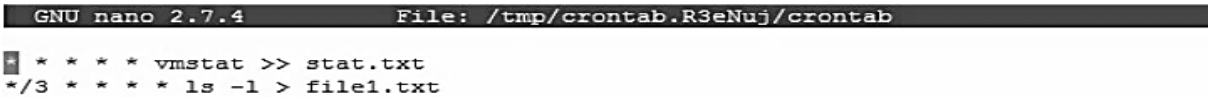

Рис. 7.2. Редагування поточної *cron*-таблиці у редакторі nano

Для видалення всіх регулярних завдань поточного користувача потрібно виконати команду **`crontab`** з ключем **`-r`**.

```
debianuser@debianServ2017:~$ crontab -r
debianuser@debianServ2017:~$ crontab -l
no crontab for debianuser
```

Таким чином, були видалені всі завдання користувача **debianuser** для регулярного виконання.

Розділ 8.
МОНІТОРИНГ СТАНУ ОПЕРАЦІЙНОЇ СИСТЕМИ ТА РОБОТА У МЕРЕЖІ

8.1. Контроль продуктивності

8.1.1 Відстеження продуктивності процесора і пам'яті

Серед множини інструментів, що призначені для відстеження продуктивності системи, однією з найстаріших команд, які містять мінімум необхідної інформації, є команда ***vmstat***. Команда дозволяє отримати інформацію про те, наскільки завантажений процесор, як часто ядро виконує підкачування сторінок і як використовуються ресурси введення та виведення.

Особливість команди ***vmstat*** полягає у розумінні її звіту. У якості прикладу виконаємо команду, яка буде виводити статистичну інформацію кожні 3 секунди.

```
debianuser@debianServ2017:~$ vmstat 3
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 750028 435684 731776    0    0     1    11   61   14  0  0 99  0  0
 0  0      0 749872 435684 731768    0    0     0    12   55   83  0  0 99  0  0
 0  0      0 749912 435684 731748    0    0     0    16   68   99  0  1 99  0  0
 0  0      0 751068 435684 731748    0    0     0    35   77  115  0  0 99  0  0
 0  0      0 751076 435684 731740    0    0     0    19   65  100  0  1 99  0  0
^C
debianuser@debianServ2017:~$
```

Таким чином, при виконанні команди ***vmstat*** з одним аргументом визначається інтервал виведення статистичної інформації щодо стану системи. Виведення може продовжуватись до нескінченності, допоки ми не здійснимо переривання цього процесу за допомогою комбінації ***<Ctrl>+<C>***.

Так само під час виконання команди ***vmstat*** з двома аргументами вказується скільки разів має бути отримана інформація. Наступний приклад демонструє виведення статистичної інформації протягом 30 секунд, а саме 10 разів з інтервалом 3 секунди.

```
debianuser@debianServ2017:~$ vmstat 3 10
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 750160 435692 731820    0    0     1    11   61   14  0  0 99  0  0
 0  0      0 749536 435692 731852    0    0     0     8   71  201  1  4 95  0  0
 0  0      0 749576 435692 731848    0    0     0     7   67   95  0  0 100  0
0
 0  0      0 749672 435692 731844    0    0     0    35   56   86  0  0 99  0  0
 0  0      0 750816 435692 731836    0    0     0    12   49   80  0  0 100  0
0
 1  0      0 750948 435692 731840    0    0     0    27   66   94  0  1 99  0  0
 0  0      0 750980 435692 731820    0    0     0     9   54   85  0  0 99  0  0
 0  0      0 750988 435692 731820    0    0     0     0   65   92  0  0 100  0
0
 0  0      0 750956 435692 731820    0    0     0     5   65   99  0  1 99  0  0
 0  0      0 750988 435692 731828    0    0     0    17   56   79  0  0 100  0
0
```

Отже, результат виконання команди ***vmstat*** представляє 10 рядків, у яких міститься інформація за кожні останні 3 секунди. Неважко зрозуміти, що виконання команди ***vmstat*** без жодного аргументу виведе статистичну інформацію на поточний момент часу, а результат буде складатися тільки з одного рядка.

Тепер розглянемо, з чого складається відповідна інформація. Вона розподіляється за такими категоріями: **procs** – процеси; **memory –** використання пам’яті; **swap** – сторінки, які переміщуються в область підкачування та з неї; **io –** для використання диска; **system** – кількість перемикань ядра на його код; **cpu –** кількість часу, витраченого різними частинами системи.

У правій колонці під заголовком **cpu** можна побачити розподіл процесорного часу (стовпці **us**, **sy**, **id** і **wa**). Вони повідомляють відповідне співвідношення часу у відсотках, який процесор витрачає на завдання користувача, системні завдання (завдання ядра), бездіяльність і очікування введення/виведення.

Інформацію стосовно середньої завантаженості системи за період 5с, 10с і 15с, а також час безперервної роботи системи, можна отримати за допомогою утиліти ***uptime***.

```
debianuser@debianServ2017:~$ uptime
 17:17:51 up 6 days,  1:58,  1 user,  load average: 0.54, 0.27, 0.09
```

Отримати загальну інформацію про наявну фізичну і віртуальну пам’ять у системі можна за допомогою команди ***free***. Крім цього, виведення містить інформацію щодо розміру буферів системи.

```
debianuser@debianServ2017:~$ free
              total        used        free      shared  buff/cache   available
Mem:        2028064      109004      751524       24276     1167536     1706824
Swap:       2050044           0     2050044
```

8.1.2 Визначення версії дистрибутиву

Дистрибутиви на базі ядра Linux підтримують можливість виведення інформації про поточну версію, за допомогою інструмента ***uname***. Ця

утиліта підтримує можливість виведення поточної інформації про версію дистрибутиву та ядра.

```
kio@chmnu-testl:~$ uname
Linux
```

Детальнішу інформацію про версію ОС можна отримати за допомогою ключів, які описані у табл. 8.1.

Табл. 8.1. Основні ключі команди *uname*

Ключ	Опис
-a	Надає користувачу повну інформацію про систему
-s	Виводить назву ядра ОС
-n	Призначений для виведення мережевої назви комп'ютера
-r	Виводить у стандартний потік версію ядра ОС
-v	Надає користувачу більш детальну інформацію
-m	Виводить на екран архітектуру ОС («x86 або 64»)
-o	Виводить назву ОС.

Утиліта ***uname*** працює з ОС на базі ядра UNIX, що так само було основою для ядра Linux. При використанні цієї утиліти найбільш детальний вивід можливий за допомогою додавання ключа ***-a***.

```
Linux chmnu-testl 4.9.0-6-amd64 #1 SMP Debian 4.9.88-1+deb9ul (2018-
05-07) x86 64 GNU/Linux
```

8.1.3. Визначення обсягу вільного дискового простору

Наступним важливим фактором стану системи є моніторинг розподілення інформації на жорсткому диску. Його здійснення можливе за допомогою інструмента ***df*** (**disk free**).

df є інструментом, що працює з операційними системами на базі UNIX, його основна задача полягає у відображенні файлових систем та їх розміру на всіх пристроях, що підключені до комп'ютера, зайнятого та вільного дискового простору.

```
kio@chmnu-testl:~$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
udev             1885332       0   1885332   0% /dev
tmpfs             379296    7632    371664   3% /run
/dev/sda1       10253588 5328220   4384800  55% /
tmpfs            1896472       0   1896472   0% /dev/shm
tmpfs               5120       0      5120   0% /run/lock
tmpfs            1896472       0   1896472   0% /sys/fs/cgroup
```

Зазначений інструмент також має набір ключів, що представлені у табл 8.2 дозволяють налаштувати виведення інформації.

Табл. 8.2. Основні ключі команди *df*

Ключ	Опис
-h	Перетворює розмір у зрозумілий людині вигляд, додає назву одиниць (Кілобайт, Мегабайт і т. д.)
-a	Відображає усі файли, включаючи блоки, що мають розмір 0 байт
-k	Дозволяє змінити розмір блоків, та відображає їх розміром по 1 кілобайту на відміну від встановленого за замовчуванням розміру 512 байт
-i	Відображає файловий дискриптор (inodes, інформацію про файл стандарту)
-T	Відображає лише тип файлових систем, що використовуються.

Для коректного відображення прийнято комбінувати ключі, наприклад відображення типу файлової системи та розміру з відображенням одиниці вимірювання:

```
kio@chmnu-test1:~$ df -Th
Filesystem     Type      Size  Used Avail Use% Mounted on
udev           devtmpfs  1.8G     0  1.8G   0% /dev
tmpfs          tmpfs     371M  7.5M  363M   3% /run
/dev/sda1      ext4      9.8G  5.1G  4.2G  55% /
tmpfs          tmpfs     1.9G     0  1.9G   0% /dev/shm
tmpfs          tmpfs     5.0M     0  5.0M   0% /run/lock
tmpfs          tmpfs     1.9G     0  1.9G   0% /sys/fs/cgroup
```

Команда ***df*** дозволяє легко визначити, чи існує на пристрої стан браку вільного простору. Системний адміністратор має визначити, що робити у зв'язку з виникненням цієї проблеми.

Для визначення браку вільного простору призначена ще одна команда, що має назву ***du***. Вона показує використання дискового простору стосовно конкретного каталогу (за замовчуванням поточного каталогу). За допомогою команди ***du*** можна швидко визначити, чи є такі файли, які марно витрачають дисковий простір у системі.

За замовчуванням команда ***du*** відображує усі файли, каталоги і підкаталоги, розташовані під поточним каталогом, а також показує, скільки дискових блоків займає кожен файл або каталог.

```
debianuser@debianServ2017:~$ cd scripts
debianuser@debianServ2017:~/scripts$ du
20      ./graphic
100     ./simple
48      ./functions
172     .
```

8.2. Моніторинг апаратного забезпечення

Операційні системи, що базуються на ядрі Linux дозволяють контролювати та моніторити апаратну складову комп'ютера. Наприклад, в операційних системах Windows відображення підключених пристроїв відбувається за допомогою інструмента «***devmgmt.msc***», що викликається з консолі, або графічно у панелі керування.

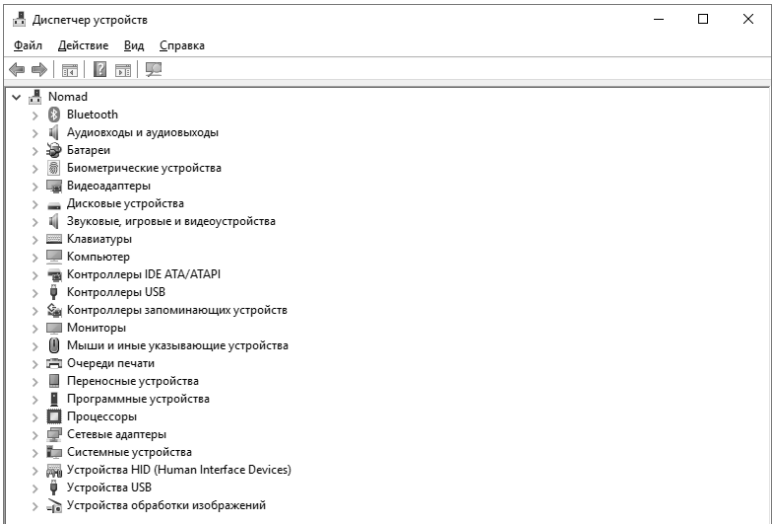

Рис. 8.1. Інтерфейс інструмента **devmgmt**

8.2.1. Інструмент lshw

Найближчим аналогом у дистрибутивах linux є ***lshw,lspci*** та ***lsusb***. Повний перегляд підключеного обладнання здійснюється за допомогою Linux Hardware Lister, що потребує встановлення, але відображає найбільш повну інформацію про підключені пристрої на комп'ютері.

Серед можливостей програми: виведення назви комп'ютера, виробника системної плати, версії та виробника BIOS, архітектури та тактової частоти процесора, виробника, об'єму та частоти пристроїв ОЗУ.

```
root@debianServ2017:~# lshw
debianserv2017
    description: Computer
    product: VirtualBox
    vendor: innotek GmbH
    version: 1.2
    serial: 0
    width: 64 bits
    capabilities: smbios-2.5 dmi-2.5 vsyscall32
    configuration: family=Virtual Machine uuid=4AE6BB2C-C64B-404E-8043-D64D204A2
824
  *-core
       description: Motherboard
       product: VirtualBox
       vendor: Oracle Corporation
       physical id: 0
       version: 1.2
       serial: 0
     *-firmware
          description: BIOS
          vendor: innotek GmbH
          physical id: 0
          version: VirtualBox
          date: 12/01/2006
          size: 128KiB
          capabilities: isa pci cdboot bootselect int9keyboard int10video acpi
     *-memory
          description: System memory
          physical id: 1
          size: 1980MiB
     *-cpu
          product: Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
          vendor: Intel Corp.
          physical id: 2
          bus info: cpu@0
          width: 64 bits
```

Інструмент ***lshw*** відображає загальну інформацію щодо підключення мережевих адаптерів та пристроїв зберігання інформації (виробника та

модель), але за наявності необхідного програмного забезпечення – драйвера. Детальнішу інформацію про спеціалізовані роз'єми та пристрої надають утиліти ***lspci*** та ***lsusb.***

8.2.2. Інструмент lspci

lshw – відображає лише загальні відомості про шину PCI, що являється базовою для підключення периферійних пристроїв до системної плати. Більш поглиблену інформацію про пристрої, що підключені цією шиною надає інструмент ***lspci***.

Інформація виводиться у стандартний потік виводу:

```
debianuser@debianServ2017:~$ lsusb
Bus 001 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

Для коректної роботи цього інструмента необхідна додаткова бібліотека ***libpci***, що забезпечує доступ до адресного простору шини ***PCI***. За замовчуванням вона входить до набору більшості дистрибутивів Linux.

Інформація виводу цієї утиліти надається системою та зберігається у файлі: ***/usr/share/pci.ids*** або у більш сучасних версіях каталогу «/proc/bus/pci», що містить підкаталоги для кожної шини.

8.2.3. Інструмент lsusb

У сучасному світі також активно використовується шина ***usb***, що дозволяє підключення різноманітного обладнання: флеш-накопичувачів, зовнішніх жорстких дисків, мережевих адаптерів, різноманітних інтерфейсів і т. д. Моніторинг цього інтерфейсу в ОС, що базується на ядрі Linux, здійснюються за допомогою інструмента lsusb:

```
debianuser@debianServ2017:~$ lsusb
Bus 001 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

Інформація для його виводу зберігається у системному файлі ***/dev/bus/usb***. Усі сучасні версії ядра підтримують зазначений інтерфейс.

8.3. Відображення показань датчиків

Під час роботи комп'ютера важливим фактором є показники температури. Перевищення максимуму температури ЦП, або іншої частини комп'ютера призводить до поломки. Ядро Linux підтримує можливість отримання даних від більшості сучасних датчиків. Інструмент,

що дозволяє моніторити температуру центрального процесора та системної плати є пакет ***lm-sensors***.

```
$ sensors
coretemp-isa-0000
Adapter: ISA adapter
Core 0:       +46.0°C  (high = +74.0°C, crit = +100.0°C)
Core 1:       +39.0°C  (high = +74.0°C, crit = +100.0°C)
Core 2:       +41.0°C  (high = +74.0°C, crit = +100.0°C)
Core 3:       +39.0°C  (high = +74.0°C, crit = +100.0°C)

atk0110-acpi-0
Adapter: ACPI interface
Vcore Voltage:       +1.26 V  (min =  +0.85 V, max =  +1.60 V)
 +3.3 Voltage:       +3.23 V  (min =  +2.97 V, max =  +3.63 V)
 +5 Voltage:         +4.92 V  (min =  +4.50 V, max =  +5.50 V)
 +12 Voltage:       +12.25 V  (min = +10.20 V, max = +13.80 V)
CPU FAN Speed:      1147 RPM  (min =  600 RPM, max = 7200 RPM)
CHASSIS1 FAN Speed:    0 RPM  (min =  800 RPM, max = 7200 RPM)
CHASSIS2 FAN Speed:    0 RPM  (min =  800 RPM, max = 7200 RPM)
POWER FAN Speed:       0 RPM  (min =  800 RPM, max = 7200 RPM)
CPU Temperature:     +49.0°C  (high = +60.0°C, crit = +95.0°C)
MB Temperature:      +36.0°C  (high = +45.0°C, crit = +95.0°C)
```

lm-sensors не входить до складу дистрибутивів за замовчуванням, але встановлюється за допомогою пакетного менеджера. Перед початком роботи необхідно провести операцію пошуку датчиків: ***sensors-detect***.

8.4. Робота у мережі

8.4.1. Мережеві інтерфейси

Визначити наявний мережевий інтерфейс можна за допомогою команди *ifconfig*, у деяких випадках вона відсутня у змінній ***$PATH***, тому потребує виклику з вказанням абсолютного шляху: ***/sbin/ifconfig***.

```
debianuser@debianServ2017:~$ /sbin/ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.96.164  netmask 255.255.240.0  broadcast 192.168.111.255
        inet6 fd0a:e490:57f9:0:a00:27ff:fe88:3b00  prefixlen 64  scopeid 0x0<glc
        inet6 fe80::a00:27ff:fe88:3b00  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:88:3b:00  txqueuelen 1000  (Ethernet)
        RX packets 41467665  bytes 5372101079 (5.0 GiB)
        RX errors 4618  dropped 9896602  overruns 0  frame 0
        TX packets 1812334  bytes 791203184 (754.5 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 19  base 0xd020

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 234  bytes 30068 (29.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 234  bytes 30068 (29.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Ми бачимо два наявні інтерфейси, а саме: lo (Loopback) та ***enp0s3***. lo (Loopback) – мережевий інтерфейс, що за замовчуванням доступний у більшості ОС і має адресу 127.0.0.1 та призначений для надсилання даних

назад на комп'ютер без додаткового обладнання («внутрішня петля»). Інтерфейс enp0s3 є віртуалізацією дротового підключення, що зазвичай підписаний eth0.

Ifconfig має декілька корисних параметрів для більш розширеного відображення інформації про інтерфейси:

– ***a*** – відображення усіх наявних інтерфейсів, у тому числі відключених;

– ***s*** – скорочений вигляд інформації, у вигляді таблиці;

– ***v*** – відображення детальної інформації про помилки.

8.4.2. Налаштування інтерфейсу Ethernet

Операційні системи що базуються на ядрі Linux дозволяють просте та швидке налаштування мережі. Усі налаштування зберігаються у файлі ***/etc/apt/network/interfaces***, далі для встановлення статичної адреси необхідно додати наступний запис:

```
auto eth1
iface eth1 inet static
 address 192.168.1.2
 netmask 255.255.248.0
 gateway 192.168.1.1
```

За такого запису інтерфейс ***eth1*** отримує адресу: ***192.168.1.2*** та використовуватиме у якості сервера ***192.168.1.1***. У більшості сучасних мереж використовується автоматична видача адреси за допомогою ***DNC*** сервера. Автоматичне налаштування адреси виглядає наступним чином:

```
iface eth0 inet dhcp
auto eth0
```

8.4.3. Налаштування бездротового інтерфейсу Wi-Fi

На відміну від інтерфейсу Ethernet бездротовий інтерфейс Wi-Fi потребує дещо складніших налаштувань. Першим кроком буде налаштування файлу: ***/etc/apt/network/interfaces***:

```
auto wlan0
iface wlan0 inet dhcp
  wpa-conf
/etc/wpa_supplicant/wpa_supplicant.conf
```

За такого синтаксису додаткові налаштування авторизації WPA-EAP (Extensible Authentication Protocol) зберігаються у файлі «/etc/wpa_supplicant/ wpa_supplicant.conf». Файл з даними авторизації має наступний вигляд:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=wheel
```

```
network={
    ssid="Chmnu_test"
    scan_ssid=1
    key_mgmt=WPA-PSK
    psk="passsword"
}
```

У конфігурації назва мережі – «Chmnu_test», протокол безпеки який використовує мережа WPA, а параметр ***psk*** вказує її пароль.

8.5. Діагностичні інструменти мережі

8.5.1. Використання протоколу ARP

ARP (Address Resolution Protocol) – протокол, що використовується у комп'ютерних мережах для визначення ***MAC***-адреси, та назву комп'ютера за допомогою IP. Робота застосунку вимагає спеціальних прав ***root***. Для загального сканування мережі з визначенням усіх підключених пристроїв використовується параметр ***-l***:

```
root@debianServ2017:~# arp-scan -l
Interface: enp0s3, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.9 with 4096 hosts (http://www.nta-monitor.com/tools/arp-scan
/)
192.168.96.0    ff:ff:ff:ff:ff:ff (00:15:e9:ab:c3:b4)   Broadcast
192.168.96.6    40:16:7e:65:40:ef       (Unknown)
192.168.96.11   00:e0:7d:cf:89:4c       NETRONIX, INC.
192.168.96.13   00:1d:92:d7:fd:c1       MICRO-STAR INT'L CO.,LTD.
192.168.96.14   00:1d:92:d7:fd:a6       MICRO-STAR INT'L CO.,LTD.
192.168.96.15   00:13:8f:52:d6:79       Asiarock Incorporation
192.168.96.16   00:1d:92:d7:fd:d8       MICRO-STAR INT'L CO.,LTD.
```

8.5.2. Використання інструмента ping

Найвідомішим діагностичним інструментом є саме ping, що дозволяє визначати час затримки відповіді цільового хосту. Спектр його використання дуже широкий: налаштування швидкодії застосунків клієнт-серверної архітектури, діагностика апаратного забезпечення, діагностика налаштування програмного забезпечення, діагностика проблем підключення.

Синтаксис команди виклику дуже простий «ping [ключі] адреса_ цільового_хосту»:

```
root@chmnu-testl:/home/kio# ping chmnu.edu.ua
PING chmnu.edu.ua (217.77.213.44) 56(84) bytes of data.
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=1 ttl=54 time=41.9 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=2 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=3 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=4 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=5 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=6 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=7 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=8 ttl=54 time=41.6 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=9 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=10 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=11 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=12 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=13 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=14 ttl=54 time=41.5 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=15 ttl=54 time=41.6 ms
64 bytes from 217.77.213.44 (217.77.213.44): icmp_seq=16 ttl=54 time=41.6 ms
^C
--- chmnu.edu.ua ping statistics ---
16 packets transmitted, 16 received, 0% packet loss, time 15024ms
rtt min/avg/max/mdev = 41.500/41.585/41.959/0.162 ms
```

У операційних системах на базі Linux інструмент ***ping*** працюватиме до надходження команди зупинки: комбінація клавіш **ctrl+c** або до закінчення часу заданого відповідним ключем.

Після закінчення роботи цей інструмент надає користувачу детальну інформацію про кількість надісланих та отриманих пакетів, відсоток втрат та час найдовшої, найшвидшої, середньої відповіді.

Окрім режиму за замовчуванням **ping,** має широкі можливості налаштування:

– ***-4*** та ***-6*** зміна версії протоколу ***IP***, на ***ipv4*** або ***ipv6***;

– ***-b*** використати широкомовну адресу для команди ***ping***;

– ***-c*** встановлення обмеження кількості пакетів;

– ***-i*** час затримки між відправкою пакетів;

– ***-I*** задає інтерфейс через який надсилатимуться пакети.

8.5.3. Визначення маршруту передачі даних

У процесі налаштування іноді не достатньо знати адресу комп’ютера, але для діагностування та вирішення проблеми необхідно знати усі вузли через які відбувається передача даних на цільовий хост. У цьому випадку використовується інструмент «TRACEROUTE».

«TRACEROUTE» у операційних системах на базі ядра Linux входить до набору інструментів за замовчуванням. Робота цього застосунку також потребує наявності спеціальних прав ***root***, а синтаксис виклику наступний: «traceroute [ключі] адреса_цільового_хосту»:

```
root@chmnu-testl:/home/kio# traceroute chmnu.edu.ua
traceroute to chmnu.edu.ua (217.77.213.44), 30 hops max, 60 byte packets
 1  216.239.46.120 (216.239.46.120)  30.928 ms  30.937 ms  31.466 ms
 2  108.170.248.133 (108.170.248.133)  32.071 ms  31.671 ms  31.539 ms
 3  217.77.209.158 (217.77.209.158)  41.401 ms  42.074 ms  41.432 ms
 4  217.77.208.250 (217.77.208.250)  42.530 ms  41.801 ms  42.882 ms
 5  217.77.213.44 (217.77.213.44)  42.787 ms  42.194 ms  41.703 ms
root@chmnu-testl:/home/kio#
```

Для визначення проміжних вузлів ***traceroute*** надсилає цільовому хосту декілька ***ICMP***-пакетів, на кожному кроці збільшуючи значення поля пакету ***TTL*** («час життя») на 1. Це поле зазвичай вказує максимальну кількість маршрутизаторів, що може бути пройдена пакетом. Перша серія пакетів надсилається з TTL=1, тому перший маршрутизатор повертає назад ICMP-повідомлення «time exceeded in transit», яке показує неможливість доставки даних. ***Traceroute*** фіксує адресу маршрутизатора, а також час між відправленням пакета й отриманням відповіді. Потім ***traceroute*** повторює відправлення серії пакетів, але вже з TTL=2, що змушує перший маршрутизатор зменшити TTL пакетів на одиницю та скерувати їх до другого маршрутизатора, який отримавши пакети з TTL=1, так само повертає «time exceeded in transit».

Серед ключів TRACEROUTE варто виділити наступні:

– ***-4*** та ***-6*** зміна версії протоколу ***IP***, на ***ipv4*** або ***ipv6***;

– ***-f*** вказати фазу життя ***TTL*** з якої почнеться відстеження;

– ***-m*** вказує обмеження кількості вузлів через які може пройти пакет;

– ***-p*** замінює порт за замовчуванням;

– ***--mtu*** задає розмір пакету;

– ***-w*** задає максимальний час очікування відповіді від вузла.

8.6. Мережевий доступ до файлів

Більшість дистрибутивів побудованих на ядрі Linux дозволяють мережевий доступ до файлів через декілька різних мережевих протоколів прикладного рівня (моделі ***OSI***): ***FTP, SMB/CIFS, SSH, HTTP***.

8.6.1. Протокол SMB

Найбільш поширений серед усіх є ***SMB*** – що, як правило використовується у локальній мережі для передачі файлів між комп'ютерами та ***FTP*** – що розповсюджений, найчастіше використовується під час роботи web-сервісів для віддаленого завантаження файлів. Розглянемо можливість встановлення та налаштування деяких пакетів, що дозволяють мережевий доступ, на прикладі дистрибутиву ***Debian***.

Пакет що дозволяє працювати з протоколом ***SMB*** називається ***samba***, він поділяється на дві частини: серверну – ту, що дозволяє доступ до файлів на комп'ютері користувача, та клієнтську – що дозволяє користувачу підключатися до віддаленого сервера та працювати з фалами.

Встановлення та налаштування серверної частини ***samba*** починається з команди***: apt-get install samba*** (перед початком встановлення

варто оновити інформацію про наявні пакети). Цей пакет містить відразу обидві частини (клієнт та сервер) необхідні для роботи з протоколом ***SMB***.

```
root@linuxtest:~# apt-get install samba
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  attr libaio1 python-crypto python-dnspython python-ldb python-samba
  python-tdb samba-common samba-common-bin samba-dsdb-modules
  samba-vfs-modules tdb-tools
Suggested packages:
  python-crypto-dbg python-crypto-doc python-gpgme bind9 bind9utils ctdb
  ldb-tools ntp | chrony smbldap-tools winbind ufw heimdal-clients
The following NEW packages will be installed:
  attr libaio1 python-crypto python-dnspython python-ldb python-samba
  python-tdb samba samba-common samba-common-bin samba-dsdb-modules
  samba-vfs-modules tdb-tools
0 upgraded, 13 newly installed, 0 to remove and 102 not upgraded.
Need to get 4,291 kB of archives.
After this operation, 27.8 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Завершивши встановлення, користувач має налаштувати пакет для роботи у локальній мережі. Налаштування пакету ***samba*** зберігаються у файлі ***/etc/samba/smb.conf***. Першим кроком потрібно відредагувати область ***[global]*** – загальні налаштування сервера: назва комп'ютера в локальній мережі (***netbios name***), робочу групу (***workgroup***), та файл з іменами користувачів що мають право підключатися (***username map***) та бажано вказати шлях до зберігання файлу журналів (***log file***). Отже, у налаштованому вигляді файл конфігурацій матиме такий вигляд:

```
[global]
netbios name = debianSamba
workgroup = WORKGROUP
server string = Samba Server
log file = /var/log/samba/%m.log
max log size = 50
username map= /etc/samba/smbusers
```

Налаштування директорії для мережевого доступу виглядає аналогічно: спочатку потрібно у квадратних дужках вказати назву, що буде відображатися при мережевому підключенні – [SMB_TEST], після чого обов'язковим параметром є шлях для локальної директорії (***path***) та для найпростішого налаштування два прапорці: відображення цієї директорії під час підключення (***browseable***) і режим запису (***read only***, для запису необхідно вказати ***no***). Варто також зважати на права що має сам каталог. Загальний вигляд конфігурацій каталогу наступний:

```
[SMB_TEST]
path = /sambaTest
browseable = yes
read only = no
```

Останнім кроком налаштувань є керування користувачами, а саме додавання прав віддаленого доступу користувачам ОС. Надання прав доступу до сервера ***samba*** відбувається за допомогою команди ***smbpasswd:*** для існуючого користувача в системі ***smbpasswd -a user***, де ***user*** – ім'я користувача, після чого необхідно ввести пароль для віддаленого підключення. Насамкінець активація користувача ***smbpasswd -e user***.

```
root@linuxtest:/# smbpasswd -a user
New SMB password:
Retype new SMB password:
Added user user.
root@linuxtest:/# smbpasswd -e user
Enabled user user.
```

Завершивши налаштування користувач має перезавантажити пакет *samba* (***/etc/init.d/samba restart***) та перевірити коректність підключившись до сервера, для цього в командному рядку необхідно ввести IP адресу наприклад «\\192.168.43.217». У разі коректного налаштування користувач побачить запрошення ввести ім'я та пароль для підключення (рис. 8.2).

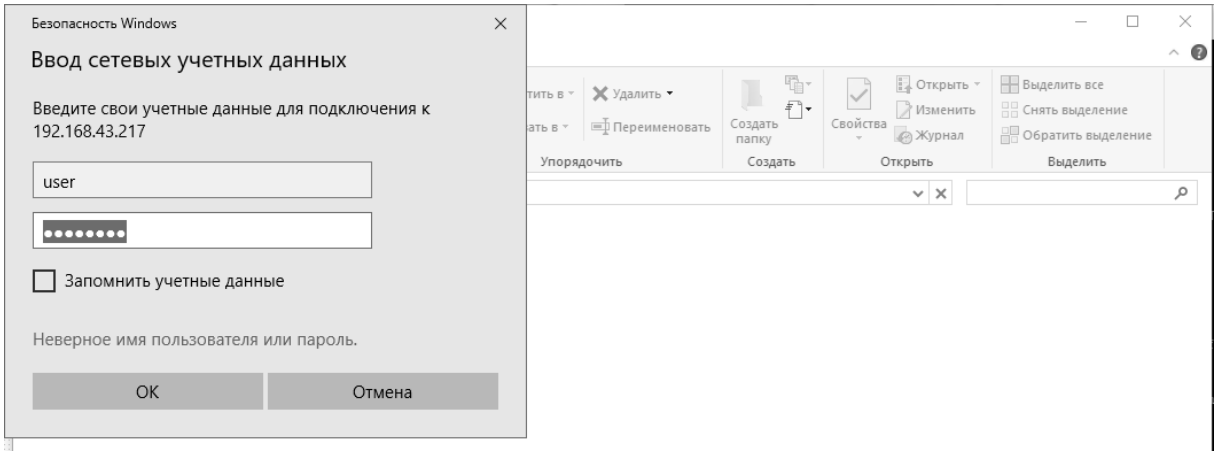

Рис. 8.2. Підключення до налаштованого віддаленого сервера

Клієнт для підключення до віддаленого сервера працює без додаткових налаштувань. Підключитися можливо шляхом введення команд у рядку адреси файлового менеджера (рис. 8.3).

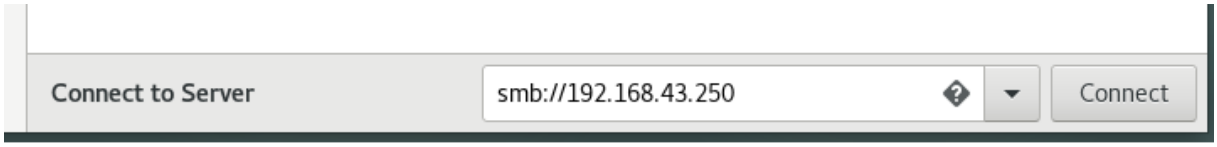

Рис. 8.3. Підключення до віддаленого комп'ютера **smb**

8.6.2. Протокол FTP

Операційні системи на базі ядра ***Linux*** часто застосовують у якості серверів, а під час роботи з віддаленим сервером ***SMB*** прийнято використовувати протокол ***FTP***.

FTP(File Transfer Protocol) – протокол призначений для передачі файлів мережею через протокол прикладного рівня ***TCP***. Він працює на архітектурі клієнт-сервер, для автентифікації використовується логін та пароль користувача, як і у протоколі SMB, але за необхідності можливе налаштування сервера на анонімне використання. Передача даних може бути зашифрована іншим протоколом – ***SSH***.

Пакет, що забезпечує роботу ***FTP*** називається VSFTP та встановлюється через менеджер керування пакетами ***apt-get install vsftpd***. Файл налаштувань цього пакету: ***/etc/vsftpd.conf***, для налаштування анонімного доступу необхідно додати рядок ***anon_upload_ enable=YES***, але задля безпеки це не рекомендується робити. Авториза-ція локальних користувачів системи можлива після додавання рядка: ***local_enable=YES*** та дозвіл на додавання файлів ***write_enable= YES***. Обмежити використання сервера від перевантаження великою кількістю людей доцільно звуженням області їхньої дії домашнім каталогом: ***chroot_local_user=YES***. Підключення та перевірка налаштування сервера найшвидше здійснюється додатковими утилітами (наприклад FileZilla).

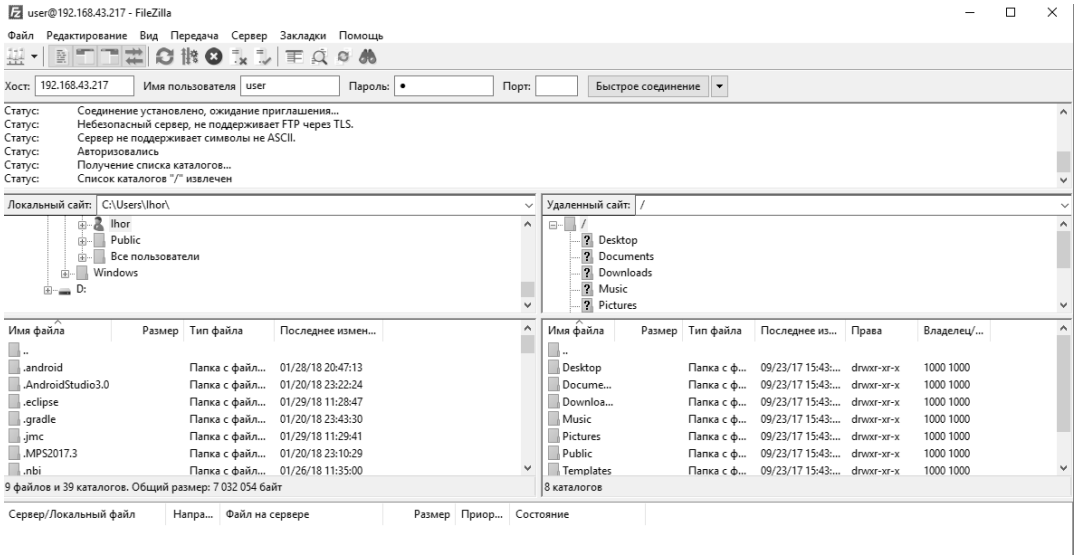

Рис. 8.4. Підключення до налаштованого FTP сервера

Дистрибутиви засновані на ядрі Linux мають можливість підключення до ***FTP*** сервера через файловий менеджер. Аналогічно з протоколом ***smb*** необхідно вказати у рядку адреси протокол та адресу (рис. 8.5).

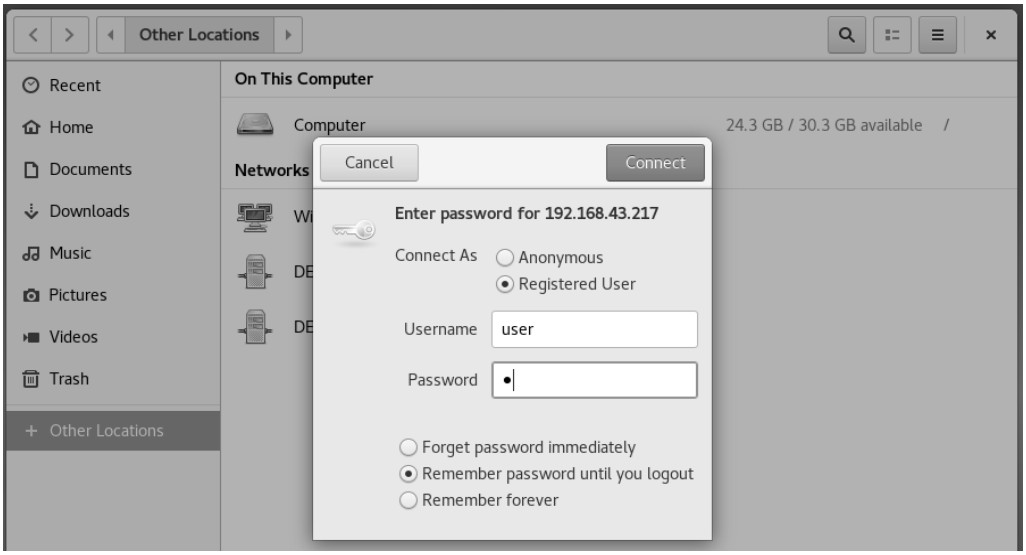

Рис. 8.5. Підключення до сервера FTP через файловий менеджер

Протокол ***SSH*** нечасто використовується самостійно, зазвичай його застосовують для передачі файлів іншим протоколом у цілях безпеки. Одним з прикладів роботи SSH з файлами є ***SCP*** (***Secure CoPy***) або ***FTP***. Передача файлів протоколом ***HTTP*** – це дуже специфічний спосіб передачі даних що використовується лише у ***web*** технологіях. У більшості випадків ***web*** встановлюється пакетам ***Apache***.

Розділ 9.
ПРОГРАМУВАННЯ СЦЕНАРІЇВ КОМАНДНОЇ ОБОЛОНКИ BASH У ОС LINUX

У попередніх розділах навчального посібника були розглянуті принципи роботи з інтерфейсом командного рядка, а саме виконання окремих команд та перегляд їх результатів. Іншими словами можна сказати, що розглядався інтерактивний режим командного рядка. Також можна додати, що інтерпретатор командного рядка дозволяє складати ланцюжки команд так, що їх виконання відбувається за один крок. Наприклад, для виконання двох команд одначасно можна ввести їх в одному й тому самому рядку з розділенням їх назв крапкою з комою.

```
debianuser@debianServ2017:~$ date ; who
Thu Jan  3 14:15:02 EET 2019
debianuser pts/0        2019-01-03 14:14 (192.168.96.112)
```

Таким чином, у представленому вище прикладі був фактично написаний найпростіший сценарій командного інтерпретатора **bash**, який використовує тільки дві команди. Спочатку буде виконана команда ***date***, яка показує поточні значення дати і часу, а потім команда ***who***, яка виводить список користувачів, що на цей час працюють із системою. З використанням цього способу можна об'єднати в один ланцюжок скільки завгодно команд, однак існує обмеження, що полягає у граничній кількості знаків у командному рядку, що досягає 255 символів. Безумовно, застосування цього способу є зручним для створення невеликих сценаріїв, але він має вагомий недолік у тому, що доводиться знову і знову набирати весь текст ланцюжка команд у командному рядку, коли виникає необхідність викликати його на виконання. А тепер представимо, що системному адміністратору доводиться кожен раз виконувати один і той самий порядок, наприклад, для налаштування сервера або мережі. Якщо порядок команд є досить великим, то вводити кожну з них ручним режимом незручно.

Саме тому, крім інтерактивного режиму командного рядка є ще й неінтерактивний, принцип роботи якого полягає у тому, що замість введення кожної команди вручну їх визначений порядок можна записати у файл і потім запустити його на виконання. Такі файли називаються **сценаріями командного інтерпретатора**.

9.1. Створення файлу сценарію командного інтерпретатора

Щоб помістити команди інтерпретатора **bash** у текстовий файл, скористаємося текстовим редактором для створення файлу, а потім

введемо в нього команди. У першому рядку файлу необхідно вказати командний інтерпретатор, який буде використовуватися при виконанні сценарію. Наприклад, якщо потрібно вказати командний інтерпретатор **bash**, у першому рядку потрібно написати наступне:

```
#!/bin/bash
```

Отже, був вказаний повний шлях до виконуваного файлу командного інтерпретатора **bash**, а перед ним зазначено порядок з двох символів ***#*** та ***!***, який називається **ше-бангом.** У всіх інших рядках сценарію командного інтерпретатора знак ***#*** розглядається як ознака початку рядка коментаря, який не обробляється командним інтерпретатором. Далі після рядка з вказанним командним інтерпретатором потрібно ввести необхідні команди у файл, вставляючи після кожного рядка символ повернення каретки. Розглянемо наступний приклад найпростішого сценарію командного інтерпретатора **bash**:

Файл first

```
#!/bin/bash
# Цей сценарій відображає дату і показує, хто зареєстрований в системі
date
who
```

У представленому файлі знаходяться всі елементи сценарію, які зазначалися вище. За бажанням у сценаріях можна використовувати крапку з комою для того, щоб помістити одразу декілька команд в один рядок. Командний інтерпретатор обробляє команди у тому порядку, в якому вони вказані у файлі.

Після збереження файлу ми будемо майже готові до його запуску на виконання. Однак перед цим потрібно виконати ще деякі дії, які будуть розглянуті далі. Під час спроби виконати файл сценарію одразу запустити його не вдасться, що й показано у наступному прикладі:

```
debianuser@debianServ2017:~/dir1$ first
-bash: first: command not found
```

Таким чином, при спробі запуску файлу було виведено помилку, що команда не знайдена. Вона стане цілком зрозумілою, якщо згадати, що у командному інтерпретаторі для пошуку виконуваних файлів команд використовується змінна середовища оточення ***PATH***. Виведемо значення цієї змінної на екран:

```
debianuser@debianServ2017:~/dir1$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

Змінна середовища оточення ***PATH*** задана так, що пошук команд здійснюється тільки в декількох заданих каталогах. Для забезпечення можливості знайти сценарій ***first*** командним інтерпретатором ***bash*** існує два різні способи:

– задати абсолютний або відносний шлях до файлу як посилання на створення нами файлу сценарію командного інтерпретатора в запрошенні до введення інформації;

– додати позначення каталогу, в якому знаходиться наш файл сценарію командного інтерпретатора, у змінну середовища ***PATH***.

Спочатку скористаємося першим способом і вкажемо командному інтерпретатору точний шлях до файлу сценарію. При завданні відносного шляху до файлу, що знаходиться у поточному каталозі, використаємо знак точки на початку:

```
debianuser@debianServ2017:~/dir1$ ./first
-bash: ./first: Permission denied
```

Після цього командний інтерпретатор успішно знайде наш файл сценарію, однак виникне інша помилка, яка вказуватиме на те, що у нас немає необхідного дозволу на виконання файлу. Подивимося на права доступу до цього файлу за допомогою команди ***ls*** з використанням ключа ***-l***:

```
debianuser@debianServ2017:~/dir1$ ls -l first
-rw-r--r-- 1 debianuser debianuser 150 Jan  3 14:20 first
```

Наступним кроком до виконання нашого файлу сценарію командного інтерпретатора є надання власникові файлу дозволу на його виконання. Як ми вже знаємо, це можна зробити за допомогою використання команди ***chmod***:

```
debianuser@debianServ2017:~/dir1$ chmod u+x first
debianuser@debianServ2017:~/dir1$ ls -l first
-rwxr--r-- 1 debianuser debianuser 150 Jan  3 14:20 first
```

Дозвіл на виконання сценарію для власника, яким є користувач **debianuser**, є, тепер спробуємо запустити файл:

```
debianuser@debianServ2017:~/dir1$ ./first
Thu Jan  3 14:30:33 EET 2019
debianuser pts/0        2019-01-03 14:14 (192.168.96.112)
```

Спроба запустити сценарій командного інтерпретатора нарешті завершилася вдало. Як видно з представленого вище прикладу, файл сценарію спрацював абсолютно правильно: спочатку була виведена поточна дата, а потім список користувачів, які працювали у системі під час запуску файлу.

Тепер розглянемо другий спосіб для запуску файлу сценарію командного інтерпретатора. Додамо до змінної середовища оточення ***PATH*** шлях до каталогу, в якому розташований наш файл сценарію, а потім спробуємо запустити, вказавши його назву як звичайну команду:

```
debianuser@debianServ2017:~/dir1$ PATH=$PATH:/home/debianuser/dir1
debianuser@debianServ2017:~/dir1$ first
Thu Jan  3 14:32:24 EET 2019
debianuser pts/0        2019-01-03 14:14 (192.168.96.112)
```

Як бачимо, можна задавати файл сценарію на виконання, просто зазначивши його назву.

9.2. Відображення повідомлень

Команди командного інтерпретатора найчастіше створюють власне виведення, яке відображується на консолі при виконанні сценарію. Проте іноді виникає необхідність у додаванні своїх власних текстових повідомлень, які могли б допомогти користувачеві слідкувати, які дії здійснюються в ході виконання сценарію. Для цього передбачена команда ***echo***, що може відобразити простий текстовий рядок, який буде введений наступним:

```
debianuser@debianServ2017:~/dir1$ echo Hello world
Hello world
```

Як видно з останнього прикладу, виведення рядка за замовчуванням зовсім не потребує використання лапок для позначення початку і кінця рядка, що відображується. Однак якщо в рядку використовуються лапки, то іноді вихідні дані можуть бути спотворені. Це можна побачити у наступному прикладі:

```
debianuser@debianServ2017:~/dir1$ echo Let's go
> -bash: unexpected EOF while looking for matching `''
-bash: syntax error: unexpected end of file
```

Тут була спроба виведення рядка, який містив одні одинарні лапки, але таке виведення завершилось помилкою. Справа у тому, що в інтерпретаторі **bash** одинарні та подвійні лапки відіграють роль символів екранування. Їх зазначення на початку та у кінці рядка дозволить трактувати спеціальні символи звичайним чином. Так само спеціальними символами є: ***$***, *******, ***?***, ***(***, ***)***, ***[***, ***]***, ***^*** та деякі інші. Одинарні та подвійні лапки є спеціальними символами, і тому їх також потрібно екранувати.

Існує ще один символ екранування, який при цьому діє тільки на один символ, що йде у рядку наступним, ним є символ ****. Отже, правильне виведення рядка з попереднього прикладу має бути зазначене наступним чином:

```
debianuser@debianServ2017:~/dir1$ echo Let\'s go
Let's go
```

З цього випливає, що для виведення символу \ звичайним чином, потрібно перед ним поставити ще один такий символ.

Крім цього можна вивести рядки зі спеціальними символами з використанням лапок. Якщо ж вони будуть задані у тексті рядка, то лапки одного типу повинні використовуватися в тексті, а іншого– служити для позначення початку і кінця рядка. Подивимося на наступний приклад:

```
debianuser@debianServ2017:~/dir1$ echo "Let's go"
Let's go
debianuser@debianServ2017:~/dir1$ echo 'He said: "Hello"'
He said: "Hello"
```

Після цього всі лапки відображаються у вихідних даних належним чином.

Хоча одинарні та подвійні лапки використовуються до однієї й тієї самої дії, між ними все ж існує невелика різниця, яка буде описана при розгляді використання змінних у сценаріях командного інтерпретатора.

9.3. Використання змінних у сценаріях

Сценарії командного інтерпретатора, які містять окремі команди, без сумніву є важливими, але іноді цього може бути замало. Наприклад, досить часто виникає необхідність включення певних даних у команди командного інтерпретатора з метою обробки інформації. Подібні завдання можна легко вирішити, використавши у сценарії змінні, які дозволяють на певний час зберегти інформацію для її подальшого застосування в інших командах.

9.3.1. Змінні середовища оточення

Інтерпретатор командного рядка підтримує змінні середовища оточення, які відстежують конкретну системну інформацію, таку як назва системи, ім'я та ідентифікатор користувача, зареєстрованого в системі, і т. д. Для виведення з метою ознайомлення всіх змінних середовища оточення призначена команда ***set***. Значення будь-якої змінної можна вставляти в сценарії, але необхідно вказати її назву та поставити перед нею символ ***$***. Як приклад наведемо наступний сценарій:

Файл user_info.sh

```
#!/bin/bash
# Відображення інформації про користувача
echo "User info for userid: $USER"
echo UID: $UID
echo HOME: $HOME
```

```
debianuser@debianServ2017:~/scripts/simple$ ./user_info.sh
User info for userid: debianuser
UID: 1000
HOME: /home/debianuser
```

Зазначимо, що файл сценарію командного інтерпретатора зазвичай має розширення ***.sh***, однак це не є обов'язковим. Це створено більше для зручності редагування таких файлів, оскільки використання консольних текстових редакторів **vi** та **nano** при редагуванні файлів з розширенням ***.sh*** будуть підсвічувати їхні коди.

9.3.2. Змінні, визначені користувачем

У сценаріях командного інтерпретатора можна не тільки використовувати змінні середовища, але й задавати і включати свої власні змінні.

Назви користувацьких змінних можуть представляти будь-який текстовий рядок довжиною до 20 символів, що може складатись з літер, цифр та символів підкреслення. У назвах користувацьких змінних символи чутливі до регістра, тому змінні **Var1** та **var1** не є однією й тією самою змінною.

Користувацьким змінним за допомогою знака рівності (**=**) присвоюються значення. Але між назвою змінної, знаком рівності і значенням не має бути жодного пробілу, оскільки для командного інтерпретатора він є розділювачем між різними аргументами командного рядка. З використанням пробілу в присвоєнні змінній певного значення виникне помилка.

Далі наведено приклад правильного присвоювання значень змінним:

```
var1=10
var2=-57
var3=testing
var4="still more testing"
```

Під час обробки сценаріїв тип даних, що використовується для представлення значення змінної, визначається автоматично. Змінні, що визначені у сценарії, зберігають свої значення протягом усього часу і знищуються після завершення. Для отримання значення користувацької змінної, так само як і у випадку змінної середовища оточення, використовується знак долара (**$**). Розглянемо у якості прикладу наступний сценарій, що демонструє роботу зі змінними.

Файл variables.sh

```
#!/bin/bash
# Перевірка змінних
days=15
guest="Іван"
echo "$guest зареєструвався $days днів тому"
days=10
guest="Олексій"
echo "$guest зареєструвався $days днів тому"
```

Виконання цього сценарію призводить до отримання наступного виведення:

```
debianuser@debianServ2017:~/scripts/simple$ ./variables.sh
Іван зареєструвався 15 днів тому
Олексій зареєструвався 10 днів тому
```

У наведеному прикладі при виведенні рядків командою **echo** ми використали подвійні лапки на початку та в кінці рядка. Однак під час виведення замість «**$guest**» та «**$days**» були виведені значення відповідних змінних. Це означає, що подвійні лапки у якості символу екранування не діють на знак долара, що вказаний перед назвою змінної, на відміну від одинарних лапок, які екранують його, вимагаючи всі

символи виводити так, як вони зазначені у рядку. Саме у цьому і полягає відмінність одинарних від подвійних лапок як символів екранування рядка.

При кожному звертанні до змінної повертається значення, що присвоєне змінній на цей час. Варто пам'ятати, що при посиланні на значення змінної перед її назвою має бути вказаний знак долара, а якщо змінна вказана з метою привласнення їй значення, то знак долара не використовується. Вищезазначене демонструється у наступному прикладі.

Файл assignment.sh

```
#!/bin/bash
# Присвоєння значення однієї змінної іншій
var1=10
var2=$var1
echo Результуючим значенням є $var2
```

Тепер запустимо даний сценарій:

```
debianuser@debianServ2017:~/scripts/simple$ ./assignment.sh
Результуючим значенням є 10
```

У більшості випадків змінні у сценаріях командного інтерпретатора **bash** інтерпретуються як рядки. Однак при створенні змінних можна також примусово задати їх тип. Для цього у командному інтерпретаторі призначена команда ***declare*** або ***typeset***, що є її повним аналогом.

За тип змінної відповідають певні ключі відповідної команди, які представлені у табл. 9.1.

Табл. 9.1. Ключі команди *declare* (*typeset*)

Ключ	Опис
-a	масив
-i	ціле число
-f	функція
-r	константа (тільки для читання)
-x	змінна середовища оточення (змінна на експорт).

Для остаточного видалення змінної з пам'яті призначена команда ***unset***.

Далі розглянемо наступний приклад:

```
debianuser@debianServ2017:~/scripts/simple$ declare -i val
debianuser@debianServ2017:~/scripts/simple$ val=3
debianuser@debianServ2017:~/scripts/simple$ echo $val
3
debianuser@debianServ2017:~/scripts/simple$ val=abc
debianuser@debianServ2017:~/scripts/simple$ echo $val
0
debianuser@debianServ2017:~/scripts/simple$ unset val
```

У цьому прикладі була оголошена змінна ***val***, та був вказаний її тип, який є цілим числом. Спочатку їй було присвоєне цілочисельне значення

3, а потім ми спробували присвоїти їй значення, яке є рядком «***abc***». При виведенні її значення у цьому випадку отримане значення ***0***. Це можна пояснити тим, що командний інтерпретатор намагався трактувати значення змінної ***val***, як ціле число. І оскільки це значення не містило жодної цифри, воно було визначене як ***0***. У самому кінці прикладу змінна ***val*** була видалена з пам'яті.

Для будь-якої змінної, крім виведення її значення, існує можливість отримати кількість символів, яку має її значення. Для цього використовується конструкція ***${#<ім'я_змінної>}***. Покажемо її застосування на наступному прикладі:

```
debianuser@debianServ2017:~/scripts/simple$ str="Hello world"
debianuser@debianServ2017:~/scripts/simple$ echo $str
Hello world
debianuser@debianServ2017:~/scripts/simple$ echo ${#str}
11
debianuser@debianServ2017:~/scripts/simple$ a=387
debianuser@debianServ2017:~/scripts/simple$ echo $a
387
debianuser@debianServ2017:~/scripts/simple$ echo ${#a}
3
```

Таким чином, ми отримали значення змінної ***str***, яким є рядок ***"Hello world"***, що містить разом з пробілом 11 символів. А змінна ***a***, яка має значення, що є числом ***387***, містить 3 символи.

Значеннями змінних можуть бути і масиви. Найбільшою особливістю створення масивів у сценаріях командного інтерпретатора Bash є те, що вони не потребують, щоб послідовність елементів була неперервною. Деякі елементи масиву можуть залишатися неініціалізованими.

Існує 2 різні способи ініціалізації масивів. Перший полягає у переліку всіх його елементів одразу. Це можна зробити наступним чином:

<змінна>=(<список_елементів_через_проміжок>)

Другим способом ініціалізації масиву є завдання значення кожного елемента окремо:

<змінна>[<n>]=<значення>,

де ***<n>*** – індекс елемента масиву, який починається з ***0***.

Для отримання значення певного елемента масиву використовується синтаксис ***${<змінна>[<n>]***, а для виведення значень всіх його елементів одразу – ***${<змінна>[*]***.

Приклад роботи з масивом демонструє наступний сценарій.

Файл array.sh

```
#/bin/bash
# Робота з масивом

countries=(Україна Польща Угорщина Італія Іспанія
Німеччина)
```

```
echo ${countries[1]}
echo ${countries[*]}
countries[2]=Великобританія
echo ${countries[*]}
```

Далі покажемо результат виконання цього сценарію:

```
debianuser@debianServ2017:~/scripts/simple$ ./array.sh
Польща
Україна Польща Угорщина Італія Іспанія Німеччина
Україна Польща Великобританія Італія Іспанія Німеччина
```

9.3.3. Виконання математичних обчислень

Ще одним достатньо важливим засобом будь-якої мови програмування є здатність маніпулювати числами. На жаль, в сценаріях командного інтерпретатора це завдання вирішується доволі складно. Нижче розглянуто як у командному інтерпретаторі *bash* реалізовані операції з числами.

Команда expr

Першим засобом для виконання операцій над числами у командному інтерпретаторі **bash** була спеціальна команда, що має назву ***expr***, яка підтримується командною оболонкою і донині. Команда ***expr*** дозволяє обробляти вирази, що задані у командному рядку, однак застосовує досить складні конструкції.

Команда **expr** розпізнає певні математичні оператори, основні з яких представлені у табл. 9.2.

Табл. 9.2. Основні математичні оператори, що підтримуються командою *expr*

Оператор	Опис
a < b	повертає 1, якщо ***a*** менше ***b***; в іншому випадку повертає 0
a <= b	повертає 1, якщо ***a*** менше або дорівнює ***b***; в іншому випадку повертає 0
a = b	повертає 1, якщо ***a*** дорівнює ***b***; в іншому випадку повертає 0
a != b	повертає 1, якщо ***a*** не дорівнює ***b***; в іншому випадку повертає 0
a >= b	повертає 1, якщо ***a*** більше або дорівнює ***b***; в іншому випадку повертає 0
a > b	повертає 1, якщо ***a*** більше ***b***; в іншому випадку повертає 0
a + b	повертає арифметичну суму ***a*** і ***b***
a – b	повертає арифметичну різницю ***a*** і ***b***
a * b	повертає арифметичний добуток ***a*** і ***b***
a / b	повертає арифметичну частку від ділення ***a*** на ***b***
a % b	повертає арифметичний залишок від ділення ***a*** на ***b***.

У якості прикладу знайдемо за допомогою команди ***expr*** арифметичну суму чисел 1 та 5.

```
debianuser@debianServ2017:~/scripts/simple$ expr 1 + 5
6
```

У представленому прикладі була виконана команда ***expr*** з трьома аргументами, розділеними пробілом: перший операнд, операція та другий операнд. Про цю особливість команди завжди варто пам'ятати.

Подивимося, що буде виведене командою ***expr***, якщо операнди та операцію не розділювати пробілами.

```
debianuser@debianServ2017:~/scripts/simple$ expr 1+5
1+5
```

У цьому випадку ми передали все команді ***expr*** одним аргументом. Однак у такому разі команда просто розпізнає вказаний у якості аргументу вираз як рядок. До речі, команда ***expr*** також підтримує і деякі рядкові оператори, інформацію щодо них можна знайти у довідковому керівництві ***man***.

Без сумніву, використання стандартних операторів у команді ***expr*** є зручним, однак іноді можуть виникнути проблеми. Справа у тому, що деякі символи, які представляють арифметичні оператори у команді ***expr*** у командному інтерпретаторі **bash**, можуть мати спеціальне значення. Типовим прикладом цього є символ *******, який з математичної точки зору представляє операцію множення, однак з точки зору системи Linux одним з символів групування назв файлів та каталогів. Застосування подібних символів у команді ***expr*** може призвести до того, що будуть отримані зовсім неочікувані результати. Для прикладу застосуємо оператор множення у команді ***expr*** безпосередньо:

```
debianuser@debianServ2017:~/scripts/simple$ expr 5 * 7
expr: syntax error
```

Для вирішення цієї проблеми доводиться використовувати символ екранування ****, який екранує символ, що слідує одразу за ним. Наступний приклад демонструє правильну операцію множення двох чисел у команді ***expr***:

```
debianuser@debianServ2017:~/scripts/simple$ expr 5 \* 7
35
```

Спрощений вигляд команди expr з використанням квадратних дужок

Хоча командна оболонка **bash** продовжує підтримку команди ***expr***, у ній передбачений ще один спосіб виконання арифметичних операцій, який є більш простим. Він полягає у включенні математичного виразу в конструкцію, яка складається зі знаку долара і квадратних дужок:

$[<математичний_вираз>]

У наступному прикладі застосуємо розглянуту вище конструкцію у командному рядку для обчислення суми та добутку двох чисел.

```
debianuser@debianServ2017:~/scripts/simple$ var1=$[5+7]
debianuser@debianServ2017:~/scripts/simple$ echo $var1
12
debianuser@debianServ2017:~/scripts/simple$ var2=$[var1*3]
debianuser@debianServ2017:~/scripts/simple$ echo $var2
36
```

Завдяки використанню квадратних дужок при виконанні операції множення символ «***» не був розпізнаний як один з символів групування назв файлів. Справа у тому, що командний інтерпретатор визначає, що він не є спеціальним, оскільки взятий у квадратні дужки.

Здійснення арифметичних дій представленим вище способом взагалі є набагато простішим, ніж застосування команди ***expr***. Його так само можна використати і у сценаріях командної оболонки. Приклад наступного сценарію це демонструє:

Файл expr.sh

```
#!/bin/bash
a=5
b=7
c=$[$a+$b]
echo "a=$a"
echo "b=$b"
echo "c=$c"
```

Виконання представленого вище сценарію призводить до отримання наступного виведення:

```
debianuser@debianServ2017:~/scripts/simple$ ./expr.sh
a=5
b=7
c=12
```

9.3.4. Підстановка результатів виконання команди

Одним з найкорисніших засобів у сценаріях командного інтерпретатора є символ зворотних лапок (`), який у мовах програмування використовується нечасто. Потрібно запам’ятати, що це не той звичайний символ одинарних лапок, який використовується в складі рядків. Він дозволяє присвоїти певній змінній результат виведення команди, яка вказана між двома символами, наприклад:

```
list=`ls -a`
```

Тут бачимо деяку аналогію з перенаправленням потоку виведення у файл за допомогою операторів «**>**» або «**>>**». Замість виведення на консоль результат виводився у певний файл, а тут – у змінну. На перший погляд може здатися, що описана конструкція не має майже нічого

особливого, однак фактично вона є однією з найжливіших будівельних блоків у програмуванні сценаріїв командного інтерпретатора **bash**.

Наступний приклад показує сценарій командного інтерпретатора, у якому оголошена змінна, якій присвоюється результат виконання команди виведення вмісту каталогу, включаючи також і приховані файли та підкаталоги.

Файл command.sh

```
#!/bin/bash
# Виведення вмісту поточного каталогу, включаючи приховані файли, у змінну
list=`ls -a`
echo "Вміст каталогу $PWD"
echo $list
```

Виконання цього сценарію призводить до виведення наступних результатів:

```
debianuser@debianServ2017:~/scripts/simple$ ./command.sh
Вміст каталогу /home/debianuser/scripts/simple
. .. arguments.sh array.sh assignment.sh case.sh command.sh condition.sh country
.txt expr.sh for1.sh for2.sh for3.sh forC.sh select.sh test_file.sh test_number.
sh test_string.sh until.sh user_info.sh value_exit.sh variables.sh while.sh
```

9.4. Вихід зі сценарію

Кожна команда, яка може бути виконана у командній оболонці **bash**, має свій статус виконання для передачі командному інтерпретатору інформації про те, як саме закінчилось її виконання. Статус виконання команди представляється цілим числом, що може приймати значення від 0 до 255, яке можна перехоплювати і використовувати у сценаріях командного інтерпретатора.

9.4.1. Перевірка статусу виконання

У командній оболонці існує спеціальна змінна ***$?***, у якій зберігається статус виконання останньої команди. Її значення потрібно обробити одразу після виконання команди, статус виконання якої варто буде перевірити. Наступний приклад показує виведення статусу виконання команди ***date***, що виводить поточну дату та час:

```
debianuser@debianServ2017:~/scripts/simple$ date
Thu Jan  3 21:09:09 EET 2019
debianuser@debianServ2017:~/scripts/simple$ echo $?
0
```

Відповідно до прийнятої угоди статус виконання успішно завершеної команди завжди дорівнює нулю. У випадку завершення виконання команди з помилкою статус її виконання буде дорівнювати певному позитивному

числу залежно від характеру помилки. У табл. 9.3 представлені основні значення статусу виконання команди.

Табл. 9.3. Значення статусу виконання команди у ОС Linux

Значення	Опис
0	успішне завершення команди
1	загальна невідома помилка
2	неправильне вживання команди командного інтерпретатора
126	команда не може виконуватися
127	невідома команда
128	неприпустимий аргумент виходу
128+x	фатальна помилка з сигналом ***x*** системи Linux
130	виконання команди завершено натисканням клавіш **<Ctrl + C>**
255	значення статусу виходу не відповідає допустимому діапазону.

Наступний приклад показує виведення статусу виконання команди, що завершилася посиланням сигналу переривання.

```
debianuser@debianServ2017:~/scripts/simple$ cat > file1
Hello world
^C
debianuser@debianServ2017:~/scripts/simple$ echo $?
130
```

9.4.2. Використання команди exit у сценаріях командного інтерпретатора

Оскільки сценарій командного інтерпретатора є командою, то він теж має статус виконання. За замовчуванням вихід з нього відбувається зі встановлення статусу виконання останньої команди у ньому. Для прикладу виконаємо вже розглянутий сценарій ***expr.sh***, у якому демонструється присвоєння змінних значень та виконання арифметичних операцій, а після цього виведемо статус його виконання:

```
debianuser@debianServ2017:~/scripts/simple$ ./expr.sh
a=5
b=7
c=12
debianuser@debianServ2017:~/scripts/simple$ echo $?
0
```

Оскільки останньою командою цього сценарію є команда ***echo "c=$c"***, цілком очікувано, що статусом її виконання буде ***0***. Цим і пояснюється результат виведення змінної ***$?***. Однак існує можливість формування і повернення статусу виконання сценарію, який може обрати сам програміст. Для цього у якості останньої команди у сценарії має бути вказана команда ***exit*** та у якості аргумента вказане значення статусу його виконання. Для прикладу розглянемо наступний сценарій:

Файл value_exit.sh

```
#!/bin/bash
#перевірка коду виходу
var1=10
var2=30
var3=$[$var1+$var2]
echo "Відповідь: $var3"
exit $var3
```

Під час перевірки статусу виконання сценарію буде отримано значення, що задане в якості параметра команди ***exit***, яким у цьому випадку є значення змінної ***var3***:

```
debianuser@debianServ2017:~/scripts/simple$ ./value_exit.sh
Відповідь: 40
debianuser@debianServ2017:~/scripts/simple$ echo $?
40
```

Також з останнього прикладу стає зрозуміло, що як аргумент команди ***exit*** можна використовувати змінні. Зокрема, необхідно дотримуватися обережності та пам'ятати, що значення статусу виконання сценарію не може перевищувати число 255. Змінимо трохи останній розглянутий приклад, замінивши операцію знаходження суми на знаходження добутку, і уважно подивимося, що буде отримано в результаті.

Файл value_exit.sh

```
#!/bin/bash
#перевірка коду виходу
var1=10
var2=30
var3=$[$var1*$var2]
echo "Відповідь: $var3"
exit $var3
```

Виклик зміненого сценарію призведе до виведення наступних результатів:

```
debianuser@debianServ2017:~/scripts/simple$ ./value_exit.sh
Відповідь: 300
debianuser@debianServ2017:~/scripts/simple$ echo $?
44
```

Отже, спочатку результат добутку чисел 10 та 30 був виведений абсолютно правильно, однак у якості статусу виконання сценарію було отримано абсолютно інше значення, незважаючи навіть на те, що у виведенні і завданні статусу виходу використовувалось значення однієї й тієї самої змінної. Це можна пояснити тим, що значення статусу виконання команди може бути відсічено для того, щоб воно не перевищувало число 255. Для цього у командному інтерпретаторі застосовується арифметика обчислення залишку від ділення. Так, отримане число 44 є залишком ділення числа 300 на 256.

9.5. Організація взаємодії сценарію командного інтерпретатора з користувачем

Іноді в сценарії потрібно організувати більш тісну взаємодію з кінцевим користувачем. Наприклад, може виникнути необхідність задати питання і чекати отримання відповіді від особи, яка викликала сценарій на виконання. Для цього у командному інтерпретаторі **bash** призначена команда ***read***.

Команда приймає вхідні дані зі стандартного потоку введення або з файлу з іншим дескриптором, після отримання яких вона поміщає їх у стандартну змінну. Наступним прикладом є сценарій, у якому застосовується найбільш простий варіант команди ***read***:

Файл input.sh

```
#!/bin/bash
# Перевірка команди read
echo -n "Введіть ваше ім'я:"
read name
echo "$name, ласкаво просимо до системи"
```

Запустимо представлений вище сценарій на виконання:

```
debianuser@debianServ2017:~/scripts/simple$ ./input.sh
Введіть ваше ім'я: Гліб
Гліб, ласкаво просимо до системи
```

Цілком очевидно, що у застосуванні команди ***read*** немає абсолютно ніякої складності. Звернемо увагу на те, що у розглянутому сценарії з командою ***echo*** застосовується ключ ***-n***, який не переводить курсор на наступний рядок. Це дозволяє кінцевому користувачеві ввести дані одразу після рядка запрошення, а не в наступному рядку. У результаті, робота зі сценарієм стає більшою мірою схожою на роботу з формами. Подібним чином працює і команда ***read*** з ключем ***-p***, що дозволяє перед введенням користувацьких даних вивести запрошення. Переробимо останній приклад, вказавши замість команди ***echo*** з ключем ***-n*** команду ***read*** з ключем ***-p***.

Файл input.sh

```
#!/bin/bash
# Перевірка команди read
read -p "Введіть ваше ім'я:" name
echo "$name, ласкаво просимо до системи"
```

Команда ***read*** може привласнювати всі введені користувачем дані в одній змінній або відразу в декількох. В останньому випадку кожне введене значення даних присвоюється черговій змінній у списку. Якщо список змінних містить менше змінних порівняно з кількістю елементів даних, то решта введених даних буде присвоєна останній змінній.

Наступний сценарій у якості прикладу демонструє введенні імена та прізвища та їх присвоєння двом різним відповідним змінним.

Файл input_more.sh

```
#!/bin/bash
# Введення декількох змінних
read -p "Введіть ваше ім'я та прізвище:" name surname
echo "Перевірка даних для $surname $name ..."
```

Результат виконання цього сценарію буде наступним:

```
debianuser@debianServ2017:~/scripts/simple$ ./input_more.sh
Введіть ваше ім'я та прізвище: Гліб Горбань
Перевірка даних для Горбань Гліб ...
```

При виклику команди ***read*** можна взагалі не ставити змінні. У такому випадку вона помістить всі введені користувацькі дані у спеціальну змінну середовища ***REPLY***, яка в такому випадку включає всі дані, задані як вхідні, та може використовуватися в сценарії командного інтерпретатора поряд з будь-якою іншою змінною.

9.6. Передача та обробка аргументів командного рядка у сценарії

Альтернативним способом організації взаємодії сценарію командної оболонки з кінцевим користувачем є передача аргументів командного рядка, використання яких дозволяє додатково задавати значення даних при виконанні сценарію. Наприклад:

```
./first.sh 6 8
```

У представленому прикладі у сценарій ***first.sh*** передаються два аргументи командного рядка ***6*** та ***8*** відповідно. Обробка таких аргументів у коді сценаріїв здійснюється за допомогою використання спеціальних змінних, які мають назву **позиційних параметрів** та у число яких входить назва самого файлу сценарію, що у поточний момент часу виконується командним інтерпретатором. Назви даних змінних представляють стандартні числа, тим самим змінна ***$0*** позначає назву виконуваного сценарію, ***$1*** – значення першого аргументу командного рядка, ***$2*** – значення другого аргументу і т. д. до змінної ***$9***, яка позначає дев'ятий аргумент командного рядка. Усі змінні позиційних параметрів можна використовувати так само як і будь-яку іншу змінну. Сценарій автоматично привласнить відповідній змінній значення відповідного аргументу командного рядка, які розділятимуться пробілами. У наступному прикладі буде розглянуто сценарій, що приймає два аргументи командного рядка, а потім обчислює їх суму та виводить її:

Файл args.sh

```
#!/bin/bash
# Перевірка з двома параметрами командного рядка
sum=$[$1+$2]
echo "Перший аргумент: $1"
```

```
echo "Другий аргумент: $2"
echo "Сума двох аргументів: $sum"
```

Наведемо результат виконання наведеного вище сценарію для аргументів командного рядка, що дорівнюють ***13*** та ***17*** відповідно:

```
debianuser@debianServ2017:~/scripts/simple$ ./args.sh 13 17
Перший аргумент: 13
Другий аргумент: 17
Сума двох аргументів: 30
```

Якщо у сценарії потрібно обробити більше дев'яти аргументів командного рядка, то назви змінних позиційних параметрів дещо змінюватимуться. Після дев'ятої змінної номери позиційних параметрів необхідно звяти у фігурні дужки, наприклад ***${10}***. Таким чином номер аргументу екранується, в іншому випадку було б виведено значення першого аргументу командного рядка, до якого був би дописаний нуль.

Наступний приклад демонструє обробку аргументів командного рядка з двознаковим порядковим номером:

Файл args2.sh

```
#!/bin/bash
# Обробка великої кількості параметрів
echo "Десятий аргумент: ${10}"
echo "Одинадцятий аргумент: ${11}"
```

Результат виконання цього простого сценарію представлений далі.

```
debianuser@debianServ2017:~/scripts/simple$ ./agrs2.sh 1 2 3 4 5 6 7 8 9 10 11 1
2
Десятий аргумент: 10
Одинадцятий аргумент: 11
```

Використаний метод дозволяє вводити таку кількість параметрів командного рядка для сценаріїв, яка буде потрібна.

9.6.1. Спеціальні змінні, що можуть бути використані під час обробки аргументів

У командному інтерпретаторі **bash** існує декілька спеціальних змінних, які дозволяють відслідковувати аргументи командного рядка додатково. Наприклад, іноді замість перевірки кожного аргумента потрібно тільки визначити, скільки їх було передано у командному рядку. Для цієї мети у командному інтерпретаторі **bash** існує спеціальна змінна **$#**, що зберігає значення кількості аргументів командного рядка, переданих при виклику сценарію на виконання.

В інших випадках буває потрібне просте захоплення всіх аргументів командного рядка для подальшої ітерації. У подібних випадках замість визначення кількості переданих аргументів у командному рядку можна застосувати інші спеціальні змінні, а саме ***$**** та ***$@***. Вони дозволяють обробити всі аргументи командного рядка відразу, оскільки включають їх у єдину змінну. Тільки між ними існує деяка незначна різниця.

У змінну ***$**** присвоюються всі аргументи командного рядка, що представляються у вигляді одного слова, в якому міститься кожен з них у тій послідовності, у якій вони наведені у командному рядку. Іншими словами, змінна ***$**** розглядає всі аргументи командного рядка не кожного окремо, а як одиного великого аргументу.

На відміну від змінної ***$****, змінна ***$@*** розглядає всі аргументи командного рядка як окремі слова у одному й тому самому рядку. Це дозволяє проводити ітерації, виділяючи кожен із заданих аргументів, для цього зазвичай використовується цикл ***for***, який детально буде розглянутий далі у наступному розділі.

Наведемо приклад простого сценарію командного інтерпретатора, що використовує описані вище змінні ***$#*** та ***$****.

Файл args3.sh

```
#!/bin/bash
# Використання спеціальних змінних
echo "Було передано $# аргуметів"
echo "Список аргуметів: $*"
```

Після виконання цього сценарію буде отримано наступний результат:

```
debianuser@debianServ2017:~/scripts/simple$ ./args3.sh
Було передано 0 аргуметів
Список аргуметів:
debianuser@debianServ2017:~/scripts/simple$ ./args3.sh 4 7 3 9 2
Було передано 5 аргуметів
Список аргуметів: 4 7 3 9 2
debianuser@debianServ2017:~/scripts/simple$ ./args3.sh 6 8 14 11 3 9 12 10 2 15
Було передано 10 аргуметів
Список аргуметів: 6 8 14 11 3 9 12 10 2 15
debianuser@debianServ2017:~/scripts/simple$ ./args3.sh Горбань Гліб Валентинович
Було передано 3 аргуметів
Список аргуметів: Горбань Гліб Валентинович
```

9.6.2. Застосування зсуву під час обробки аргументів командного рядка

Ще одним засобом, який може бути використаний у сценаріях командного інтерпретатора **bash** для обробки аргументів командного рядка, є можливість їх зсуву. Це забезпечується використанням команди ***shift***, яка відповідно до своєї назви зміщує аргументи командного рядка, змінюючи їх відносні положення.

Після кожного виконання команди за замовчуванням відбувається зміщення кожної змінної позиційного параметра з правого на лівий бік на одну позицію. Тим самим значення змінної ***$2*** буде переміщене у змінну ***$1***, значення змінної ***$3*** – у ***$2***, а значення змінної ***$1*** взагалі буде втрачено. Однак зрушення параметрів не діє на змінну ***$0***, яка зберігає назву виконуваного сценарію та завжди залишається незмінною.

Застосування команди ***shift*** представляє ще один зручний спосіб ітерації, особливо якщо неможливо дізнатися зазделегідь кількість

переданих аргументів у командному рядку. Достатньо провести спочатку обробку першого аргументу, потім застосувати команду зсуву, після чого ще раз обробити перший.

Команда ***shift*** має також можливість проведення зсуву не на одну позицію, як це відбувається при її використанні за замовчуванням, а відразу на декілька. Це забезпечується використанням команди з аргументом, який є числом, що вказує кількість елементів, на яку потрібно зробити зрушення. Так наприклад, виконання у сценаріях команди ***shift 2*** призведе до того, що початкові значення змінних ***$1*** та ***$2*** будуть втрачені, у змінну ***$1*** перейде значення змінної ***$3***, у ***$2*** – значення змінної ***$4*** і т. д.

Наведемо приклад застосування команди ***shift*** у наступному сценарії:

Файл shifting.sh

```
#!/bin/bash
# Аргументи командного рядка до зрушення
echo "Нульовий аргумент: $0"
echo "Перший аргумент: $1"
echo "Другий аргумент: $2"
echo "Зараз аргументи командного рядка будуть зрушені"
# Зрушення
shift
# Аргументи командного рядка після зрушення
echo "Нульовий аргумент: $0"
echo "Перший аргумент: $1"
echo "Другий аргумент: $2"
```

Далі представимо результат виконання представленого вище сценарію з переданими у нього трьома аргументами командного рядка.

```
debianuser@debianServ2017:~/scripts/simple$ ./shifting.sh 1 2 3
Нульовий аргумент: ./shifting.sh
Перший аргумент: 1
Другий аргумент: 2
Зараз аргументи командного рядка будуть зрушені
Нульовий аргумент: ./shifting.sh
Перший аргумент: 2
Другий аргумент: 3
```

9.7. Використання структурованих команд у сценаріях командного інтерпретатора

При написанні багатьох сценаріїв зазвичай потрібно організувати логічне управління потоком даних, що передаються між командами. Це означає, що командний інтерпретатор має вміти обробляти умови, за якими будуть виконані відповідні команди. Командна оболонка **bash** має

цілий набір команд, що дозволяють пропускати у сценарії певну послідовність команд або обробляти їх у циклі після перевірки умов, що визначаються значеннями змінних або результатами інших команд. Подібні команди мають назву **структурованих команд**.

Кількість таких команд у **bash** є дуже великою, тому далі розглянемо кожну з них окремо.

9.7.1. Інструкція if-then

Однією з найбільш важливих структурованих команд є інструкція ***if-then***, яка у **bash** має наступний синтаксис:

```
if <команда>
then
     <блок_команд>
fi
```

Тим, хто має досвід програмування на інших мовах, синтаксис цієї команди спочатку може показатися досить дивним, оскільки у багатьох мовах програмування за інструкцією ***if*** слідує умова, значення якої перевіряється на істинність або хибність. Однак інструкція ***if*** у командному інтерпретаторі **bash** працює дещо інакше.

В інструкції спочатку виконується ***<команда>***, що вказана після слова ***if***, потім перевіряється статус її виконання, який зберігається у змінній ***$?***. Якщо команда була виконана без помилок, тобто значення статусу її виконання дорівнює 0, то будуть виконанні команди ***<блок_команд>***, що перераховані після слова ***then***. Якщо статус виконання команди має ненульове значення, тобто команда була виконана з певною помилкою, то у розділі ***then*** вони виконані не будуть а буде виконана наступна за інструкцією ***if*** команда у сценарії.

9.7.2. Інструкція if-then-else

У командному інтерпретаторі **bash** є розширена версія інструкції ***if-then***, що дозволяє виконати альтернативний набір команд, якщо команда, що зазначена у розділі ***if*** була виконана з помилкою. Такою інструкцією є ***if-then-else***, синтаксис якої є наступним:

```
if <команда>
then
     <блок_команд1>
else
     <блок_команд2>
fi
```

Інструкція працює наступним чином: якщо команда в рядку інструкції ***if*** повертає нульове значення статусу виконання, то виконуються команди ***<блок_команд1>***, що перераховуються у розділі ***then***, як і в звичайній інструкції ***if-then***. Якщо ж команда в рядку інструкції ***if*** повертає значення статусу виконання, що є відмінним від нуля, будуть виконані команди ***<блок_команд2>***, що зазначені у розділі ***else***.

У наступному прикладі розглянемо сценарій, у якому використовуються три інструкції ***if-then-else***. Перша з них перевіряє виконання цілком правильної команди ***date***, друга перевіряє виконання неіснуючої у системі команди, а третя перевіряє існування облікового запису користувача **debianuser** у файлі ***/etc/passwd***, що здійснюється командою ***grep***.

Файл condition.sh

```
#!/bin/bash
# Перевірка роботи існуючої команди
if date
then
 echo "Команда ‘date’ була виконана"
else
 echo "Команда ‘date’ не була виконана"
fi

# Перевірка роботи неіснуючої команди
if sdfdfg
then
 echo "Команда ‘sdfdfg’ була виконана"
else
 echo "Команда ‘sdfdfg’ не була виконана "
fi

# Перевірка існування користувача debianuser
testuser="debianuser"
if grep $testuser /etc/passwd
then
 echo "Файли ініціалізації користувача $testuser: "
 ls -a /home/$testuser/.bash*
fi
```

Далі представимо результат виконання сценарію, код якого наведено вище:

```
debianuser@debianServ2017:~/scripts/simple$ ./condition.sh
Sat Jan  5 13:07:20 EET 2019
Команда 'date' була виконана
./condition.sh: line 11: sdfdfg: command not found
Команда 'sdfdfg' не була виконана
debianuser:x:1000:1000:debianuser,,,:/home/debianuser:/bin/bash
Файли ініціалізації користувача debianuser:
/home/debianuser/.bash_history  /home/debianuser/.bashrc
/home/debianuser/.bash_logout
```

Тепер прокоментуємо результат виконання наведеного вище сценарію. Спочатку здійснюється перевірка виконання команди ***date***. Цілком очікувано, що вона буде виконана правильно, саме тому була виконана команда, що зазначена у розділі ***then***. Наступним кроком є перевірка виконання команди ***sdfdfg***, якої у командному інтерпретаторі **bash** не існує. Саме тому спроба її виконання завершиться помилкою, і після цього буде виконана команда, що вказана у розділі ***else***. Третя перевірка завершиться правильно, оскільки команда ***grep*** знайде обліковий запис користувача ***debianuser*** у файлі ***/etc/passwd***. У цьому випадку знову будуть виконані команди, зазначені у розділі ***if***, тобто виведені всі файли ініціалізації (сценарії командного інтерпретатора **bash**, що виконуються автоматично), власником яких є відповідний користувач.

Іноді виникає необхідність перевірки більш ніж однієї умови. Замість використання декількох окремих інструкцій ***if-then*** існує можливість використання альтернативним варіантом розділу ***else***, який позначається як ***elif*** та продовжує розділ ***else***, задаючи у ньому ще одну інструкцію ***if-then***. Варіант структурованої команди ***if*** представляється наступним синтаксисом:

```
if <команда1>
then
      <блок_команд1>
elif <команда2>
then
      <блок_команд2>
fi
```

Таким чином, у рядку інструкції ***elif*** вказується інша команда, яку потрібно виконати, після чого буде перевірено значення статусу її виконання так само як і в інструкції ***if***. Інструкції ***elif*** можна пов’язувати у достатньо великі ланцюжки, що іноді призводить до створення величезної їх кількості.

9.7.3. Команда test

До цього моменту було розглянуто застосування тільки звичайних команд для перевірки статусу їх виконання у інструкціях ***if***. Однак у тих,

хто мав досвід програмування на інших мовах, може виникнути слушне питання, чи має здатність команда ***if*** обробляти якісь інші умови, а не тільки статус виконання команд. На жаль відповідь не це питання є негативною. Однак у командному інтерпретаторі **bash** є доволі зручна команда, яка призначена саме для перевірки цих умов, що застосовується разом з командою ***if*** та має назву ***test***.

Команда працює наступним чином: якщо перевірка умови, що задана у ній, призводить до отримання істинного значення, вона повертає статус свого виконання, що дорівнює нулю, в іншому випадку повертає статус виконання, що дорівнює одиниці. Таким чином, виконання інструкції ***if-then*** у сукупності з командою ***test*** дуже нагадує інструкції ***if-then*** в інших мовах програмування.

При цьому синтаксис команди ***test*** є доволі простим:

test <умова>,

де ***<умова>*** – це ряд параметрів і значень, що обробляються командою ***test***.

При використанні команди ***test*** у команді ***if-then*** остання набуває наступного вигляду:

```
if test <умова>
then
      <блок_команд>
fi
```

Командний інтерпретатор **bash** має також спрощений синтаксис використання команди ***test***, що представляється наступним синтаксисом:

```
if [ <умова> ]
then
      <блок_команд>
fi
```

Зауважимо, що квадратні дужки визначають умову, яка використовується у команді ***test***. Варто зазначити, що при використанні такого варіанту перевірки умови обов'язково має бути введений пробіл, після першої і перед останньою квадратною дужкою, оскільки в іншому випадку буде отримано повідомлення про помилку. Нагадаємо, що квадратні дужки використовуються і в завданні арифметичних виразів, однак у такому разі додаткові пробіли зазначати не потрібно.

Командою ***test*** можна обробити умови, що належать до трьох типів:

1. Порівняння чисел;
2. Порівняння рядків;
3. Порівняння файлів.

Порівняння чисел

Методом роботи з командою ***test***, який застосовується напевно найчастіше, є порівняння числових значень. Для цього можуть бути використані певні оператори, перелік яких наведений у табл. 9.4.

Табл. 9.4. Оператори порівняння чисел у команді *test*

Оператор	Опис
a -eq b	***a*** дорівнює ***b***
a -ge b	***a*** більше або дорівнює ***b***
a -gt b	***a*** більше ***b***
a -le b	***a*** менше або дорівнює ***b***
a -lt b	***a*** менше ***b***
a -ne b	***a*** не дорівнює ***b***.

Умови перевірки числових значень можуть бути використані, як для роботи з числами, так і роботи зі змінними, значення яких є числом. У якості прикладу наведений сценарій командного інтерпретатора, у якому використовуються деякі з вищезазначених операцій порівняння чисел.

Файл test_number.sh

```
#!/bin/bash
# Використання числових перевірок у сценарії

if [ $1 -gt $2 ]
then
 echo "$1 більше ніж $2"
else
 echo "$1 не більше ніж $2"
fi

if [ $1 -eq $2 ]
then
 echo "Аргументи дорівнюють один одному"
else
 echo "Аргументи різні"
fi
```

Нижче представлені приклади виконання наведеного вище сценарію для різних вхідних даних.

```
debianuser@debianServ2017:~/scripts/simple$ ./test_number.sh 4 6
4 не більше ніж 6
Аргументи різні
debianuser@debianServ2017:~/scripts/simple$ ./test_number.sh 6 4
6 більше ніж 4
Аргументи різні
debianuser@debianServ2017:~/scripts/simple$ ./test_number.sh 4 4
4 не більше ніж 4
Аргументи дорівнюють один одному
```

Порівняння рядків

Команда ***test*** надає можливість порівняння рядків. У табл. 9.5 представлені оператори, що призначені для порівняння рядків.

Табл. 9.5. Оператори порівняння рядків у команді *test*

Оператор	Опис
str1 = str2	***str1*** дорівнює ***str2***
str1 != str2	***str1*** не дорівнює ***str2***
str1 < str2	***str1*** менше ніж ***str2***
str1 > str2	***str1*** більше ніж ***str2***
-n str1	***str1*** є непорожнім
-z str1	***str1*** є порожнім.

Таким чином, оператори порівняння двох рядків за допомогою команди ***test*** набувають більш звичного вигляду для операторів порівняння, застосовуваних у багатьох мовах програмування. Звернемо увагу на те, що між першим операндом та оператором, а також між оператором та другим операндом обов'язково має бути пробіл. Також до операторів порівняння рядків відносять і два унарні оператори, що представляють ключі ***–n*** та ***–z***, які визначають є рядок непорожнім та порожнім відповідно.

Нижче наведений приклад сценарію, у якому застосовуються певні оператори порівняння рядків у команді ***test***.

Файл test_string.sh

```
#!/bin/bash
# Використання перевірок рядків у сценарії

echo -n "Введіть ім'я користувача: "
read testuser
if [ $USER = $testuser ]
then
 echo "Ласкаво просимо, $testuser!"
else
 echo "Вибачте, але ви не $testuser!"
fi
```

Результати виконання представленого вище сценарію для різних вхідних наведено нижче:

```
debianuser@debianServ2017:~/scripts/simple$ ./test_string.sh
Введіть ім'я користувача: debianuser
Ласкаво просимо, debianuser!
debianuser@debianServ2017:~/scripts/simple$ ./test_string.sh
Введіть ім'я користувача: gleb
Вибачте, але ви не gleb!
```

Наостанок зазначимо особливості застосування операторів порівняння рядків у команді ***test***. Оператори порівняння рядків на рівність та нерівність (**=** та ***!=*** відповідно) визначають, чи мають два рядки однакове значення чи ні. Однак враховуються всі знаки пунктуації, а також відмінності у застосуванні великих та малих літер.

Тим самим оператори **<** та **>** під час порівняння рядків дозволяють визначити, чи передує перший рядок другому або слідує за ним у послідовності лексичного сортування за зростанням. Використання цих операторів за наявного виклику команди ***test*** може спричинити певні труднощі. Справа у тому, що з точки зору операційної системи Linux основним призначенням символів **<** та **>** є перенаправлення стандартних потоків введення та виведення відповідно. Саме тому їх необхідно екранувати:

test str1 \< str2 та ***test str1 \> str2***

При застосуванні цих операторів у команді ***test*** у спрощеному вигляді за допомогою квадратних дужок таких проблем не виникає, отже немає потреби і в екрануванні відповідних символів.

Порівняння файлів

За допомогою команди ***test*** також можна перевірити статус файлів та каталогів у файловій системі. Для цього призначено дуже багато операторів, основні з яких наведені у табл. 9.6.

Табл. 9.6. Оператори порівняння файлів у команді *test*

Оператор	Опис
-d file	***file*** існує і є каталогом
-e file	***file*** існує
-f file	***file*** існує і є файлом
-r file	***file*** існує і призначений для читання
-s file	***file*** існує і не порожній
-w file	***file*** існує і призначений для запису
-x file	***file*** існує і призначений для виконання
-O file	***file*** існує і належить поточному користувачеві
-G file	***file*** існує і його задана за замовчуванням група є такою ж, як і у поточного користувача

Закінчення табл. 9.6

file1 -nt file2	***file1*** є новішим ніж ***file2***
file1 -ot file2	***file1*** є старішим ніж ***file2***.

Представлені інші умови надають можливості перевірки у коді сценаріїв командного інтерпретатора файлів у файловій системі та часто використовуються у сценаріях, що засновані на доступі до файлів.

Прикладом використання операторів порівняння файлів у команді ***test*** є наступний сценарій командного інтерпретатора, програмний код якого загалом зрозумілий для уважних читачів:

Файл test_file.sh

```
#!/bin/bash
# перевірка існування директорії

dir=$HOME/scripts
if [ -d $dir ]
then
 echo "Ваша директорія зі сценаріями командного рядка існує"
 ls -l $dir
else
 echo "Ваша директорія зі сценаріями командного рядка не існує"
fi
echo

# перевірка існування елемента
elem=$HOME/testing
if [ -e $elem ]
then
 echo "Запис у кінець існуючого файлу"
 date >> $elem
else
 echo "Створення нового файлу $elem"
 date > $elem
fi
echo

# перевірка власності
echo -n "Введіть назву файлу або каталогу для перевірки вашої власності на нього: "
```

```
read obj
if [ -O $obj ]
then
 echo "Ви є власником $obj"
else
 echo "Ви не є власником $obj"
fi
```

Далі наведені результати виконання зазначеного сценарію для різних вхідних даних.

```
debianuser@debianServ2017:~/scripts/simple$ ./test_file.sh
Ваша директорія зі сценаріями командного рядка існує
total 16
drwxr-xr-x 2 debianuser debianuser 4096 Feb  6 13:02 functions
drwxr-xr-x 2 debianuser debianuser 4096 Feb 18 14:25 graphic
drwxr-xr-x 2 debianuser debianuser 4096 Feb  6 13:40 labs
drwxr-xr-x 2 debianuser debianuser 4096 Jan 23 13:51 simple

Запис у кінець існуючого файлу

Введіть назву файлу або каталогу для перевірки вашої власності на нього: /etc/pa
sswd
Ви не є власником /etc/passwd
debianuser@debianServ2017:~/scripts/simple$
debianuser@debianServ2017:~/scripts/simple$ ./test_file.sh
Ваша директорія зі сценаріями командного рядка існує
total 16
drwxr-xr-x 2 debianuser debianuser 4096 Feb  6 13:02 functions
drwxr-xr-x 2 debianuser debianuser 4096 Feb 18 14:25 graphic
drwxr-xr-x 2 debianuser debianuser 4096 Feb  6 13:40 labs
drwxr-xr-x 2 debianuser debianuser 4096 Jan 23 13:51 simple

Створення нового файлу /home/debianuser/testing

Введіть назву файлу або каталогу для перевірки вашої власності на нього: /home/de
bianuser/scripts
Ви є власником /home/debianuser/scripts
debianuser@debianServ2017:~/scripts/simple$
```

9.7.4. Команда case

Під час написання сценаріїв командного інтерпретатора також виникає необхідність обчислення значення деякої змінної, а потім треба провести пошук цього значення серед цілого ряду можливих. Для виконання цього завдання варто використати команду ***case***, робота якої полягає у перевірці значення деякої змінної на відповідність декільком різним значенням.

Команда ***case*** має наступний синтаксис:

```
case <змінна> in
      <шаблон1> | <шаблон2> ) <блок_команд1> ;;
      <шаблон3> ) <блок_команд2> ;;
      * ) <блок_команд_за_замовчуванням> ;;
esac
```

Команда **case** виконує порівняння значення змінної ***<змінна>*** з декількома різними шаблонами. При знаходженні співпадіння значення змінної з будь-яким із шаблонів, вказаних у ***<шаблон>***, командний інтерпретатор переходить до виконання команд (***<блок_команд>***), які зазначені для них. В одному рядку можна задавати відразу декілька шаблонів, розділюючи їх між собою за допомогою символу «**|**». Натомість символ «*****» відіграє роль так званої універсальної пастки для тих значень, які не співпали з жодним із перерахованих вище шаблонів. Приклад, що наведений нижче, являє собою сценарій, який за допомогою команди ***case*** перевіряє, до якого типу користувачів системи відноситься певний користувач: адміністратор системи, викладач чи студент.

Файл case.sh

```
#!/bin/bash

echo -n "Введіть ім'я користувача: "
read testuser
case $testuser in
 root )
  echo "Цей користувач є адміністратором";;
 gleb | kio | kirey )
  echo "Цей користувач є викладачем, який має привілеї адміністратора";;
 debianuser )
  echo "Цей користувач є тестовим користувачем";;
 * )
  echo "Цей користувач є студентом";;
esac
```

Далі наведені результати виконання вищепредставленого сценарію для різних даних, що були введені користувачем:

```
debianuser@debianServ2017:~/scripts/simple$ ./case.sh
Введіть ім'я користувача: root
Цей користувач є адміністратором
debianuser@debianServ2017:~/scripts/simple$ ./case.sh
Введіть ім'я користувача: gleb
Цей користувач є викладачем, який має привілеї адміністратора
debianuser@debianServ2017:~/scripts/simple$ ./case.sh
Введіть ім'я користувача: debianuser
Цей користувач є тестовим користувачем
debianuser@debianServ2017:~/scripts/simple$ ./case.sh
Введіть ім'я користувача: 2710101
Цей користувач є студентом
```

9.8. Застосування циклів у програмуванні сценаріїв командного інтерпретатора bash

На практиці програмування широко застосовуваним є спосіб організації роботи шляхом виконання ітерацій, оскільки необхідність повторення певних операцій, допоки не буде виконана якась конкретна умова, виникає досить часто. Для цього у мовах програмування високого рівня призначені цикли. Під час програмування сценаріїв командного інтерпретатора **bash** використання циклів також не є виключенням. **Bash** підтримує декілька циклів, які детально будуть розглянуті далі.

9.8.1. Цикл for

У сценаріях командного інтерпретатора є можливість використання команди ***for*** для створення циклу, в якому проводиться ітерація за рядом певних значень. Команда ***for*** у командному інтерпретаторі **bash** представляється наступним синтаксисом:

```
for <змінна> in <список_значень>
do
    <блок_команд>
done
```

Ряд значень, що використовується в ітераціях, задається за допомогою параметра ***<список_значень>***. Існує декілька різних способів завдання значень у цьому списку.

У будь-якому випадку на кожній ітерації ***<змінна>*** містить поточне значення у списку ***<список_значень>***. При першій ітерації використовується перший елемент у списку, при другій ітерації – другий і так далі до закінчення всіх елементів у ньому.

Насамперед у ***<блок_команд>***, що вводиться між інструкціями ***do*** і ***done***, можна задавати одну або декілька команд інтерпретатора **bash**. Також можна використовувати поточне значення змінної ***$<змінна>***.

Завдання значень виконуватимуться декількома способами, кожен з них буде розглянуто далі.

Читання значень у списку

Найпростішим способом використання команди ***for*** є здійснення з її допомогою ітерацій за списком значень, що визначений у самій команді. У такому випадку значення списку задаються відразу після ключового слова ***in***, розділяючи кожне значення пробілом. Приклад наступного сценарію демонструє використання команди ***for*** з завданням кожного елемента списку.

Файл for1.sh

```
#!/bin/bash
# Команда for в основній формі
for city in Миколаїв Одеса Херсон Київ Запоріжжя Вінниця Львів
do
     echo "Наступне місто: $city"
done
```

Далі наведено результат виконання сценарію, код якого представлено вище:

```
debianuser@debianServ2017:~/scripts/simple$ ./for1.sh
Наступне місто: Миколаїв
Наступне місто: Одеса
Наступне місто: Херсон
Наступне місто: Київ
Наступне місто: Запоріжжя
Наступне місто: Вінниця
Наступне місто: Львів
```

Читання списку зі змінної

Під час програмування сценаріїв командного інтерпретатора часто бувають випадки, коли список значень для обробки у циклі заздалегідь не відомий, а накопичується програмним шляхом, після чого зберігається у змінній. Обробку можна зробити за допомогою команди ***for***. Наступним прикладом буде трохи змінений сценарій, у якому список значень для обробки у циклі був записаний у змінну, після чого був доданий ще один елемент.

Файл for2.sh

```
#!/bin/bash
# Використання змінної для зберігання списку
cities="Миколаїв Одеса Херсон Київ Запоріжжя Вінниця Львів"
cities=$cities" Харків"
for city in $cities
do
     echo "Наступне місто: $city"
done
```

Результат виконання цього сценарію порівняно з результатом виконання попереднього сценарію практично не змінився:

```
debianuser@debianServ2017:~/scripts/simple$ ./for2.sh
Наступне місто: Миколаїв
Наступне місто: Одеса
Наступне місто: Херсон
Наступне місто: Київ
Наступне місто: Запоріжжя
Наступне місто: Вінниця
Наступне місто: Львів
Наступне місто: Харків
```

У розглянутому вище сценарії у змінній ***cities*** зберігається стандартний список текстових значень, що їх використовують для здійснення ітерацій у циклі ***for***. Також у цьому сценарії демонструється операція додавання (конкатенації) нового елемента до списку, що є загальноприйнятим методом додавання тексту до кінця існуючого текстового рядка, збереженого у змінній.

Читання значень з команди

Ще одним способом формування значень списку для його обробки циклом ***for*** є застосування результатів підстановки виведення команди. У такому випадку використовуються символи зворотних одинарних лапок для підстановки результатів будь-якої команди, що виводить певні дані у структурованому вигляді, а потім застосовуються у циклі ***for***. Наступний приклад представляє трохи перероблений сценарій з обробкою назв українських міст, тільки у цьому разі вони зберігаються у певному файлі.

Файл for3.sh

```
#!/bin/bash
# Читання значень з файлу
file="city.txt"
for city in `cat $file`
do
     echo "Наступне місто $city"
done
```

Для перегляду результату виконання представленого вище сценарію спочатку розглянемо вміст файлу ***city.txt***, у якому містяться назви міст України, кожна з яких потім оброблюється у циклі ***for***.

```
debianuser@debianServ2017:~/scripts/simple$ cat city.txt
Миколаїв
Одеса
Херсон
Київ
Запоріжжя
Вінниця
Львів
debianuser@debianServ2017:~/scripts/simple$ ./for3.sh
Наступне місто: Миколаїв
Наступне місто: Одеса
Наступне місто: Херсон
Наступне місто: Київ
Наступне місто: Запоріжжя
Наступне місто: Вінниця
Наступне місто: Львів
```

З останнього прикладу бачимо, що кожне значення для обробки у циклі ***for*** у файлі має зберігатися як окремий рядок, а не бути заданим в одному й тому самому рядку через пробіл як розділювач. Однак, якщо у

деяких рядках файлу все ж зустрічаються пробіли, то команда ***for*** розглядає кожен елемент даних, що розділений пробілом, як окремий.

Команда for в стилі мови C

Якщо згадати цикл ***for*** у мові **C**, можна зазначити, що там він діє зовсім інакше. У ньому зазначаються:

1. Початкове значення змінної циклу перед першою ітерацією;
2. Умова, що має бути істинною для здійснення ітерацій у циклі;
3. Спосіб зміни значення змінної при кожній ітерації.

Цикл ***for***, який заданий подібним чином, завершує роботу, якщо зазначена умова приймає істинне значення.

Як приклад розглянемо наступний код на мові C:

```
for (i = 0; i < 10; i++)
{
      printf("Наступне число: %d\n", i);
}
```

Під час виконання описаного коду виконується простий ітеративний цикл, у якому змінна ***i*** використовується як лічильник. Перший розділ інструкції ***for*** присвоює початкове значення змінній ***i***. Другий визначає умову, при виконанні якого тривають ітерації у циклі. Останній, третій визначає ітеративний процес, а саме вказує вираз, який виконується після кожної ітерації. У цьому прикладі змінна ***i*** після кожної ітерації збільшується на одиницю.

Така версія циклу ***for*** підтримується і у командному інтерпретаторі **bash**. Вона виглядає аналогічно циклу ***for*** у мові **C**, однак має деякі відмінності. Розглянемо синтаксис такої версії циклу ***for*** у **bash**.

```
for ((<початкове_значення>; <умова_тривалості>; <процес_ітерації>))
```

Синтаксис циклу ***for*** у так званому стилі мови **C** досить часто стає причиною плутанини для програмістів сценаріїв командного інтерпретатора **bash** через використання у ньому посилань на змінні у стилі **C** замість посилань на змінні у **bash**. Далі показано приклад циклу ***for*** у стилі **C** у сценарії командного інтерпретатора **bash**:

```
for ((i=1; i<10; i++))
```

Від стандартного циклу ***for*** у командному інтерпретаторі **bash** цикл ***for*** у стилі мови **C** відрізняється за наступними особливостями:

1. Інструкція присвоєння початкового значення змінної може містити пробіли;
2. У зазначенні умови перед назвою змінної не потрібно ставити знак долара;
3. Вираз, за допомогою якого організовується процес ітерації, не відповідає формату команди ***expr***.

Така версія циклу ***for*** була створена розробниками командного інтерпретатора **bash** таким чином, щоб вона найбільшою мірою відповідала циклу ***for*** у стилі мови **C**. З одного боку, це дозволяє спростити роботу програмістів, які володіють мовою C, але з іншого викликає певні незручності для програмістів сценаріїв командного інтерпретатора **bash**.

Далі наводиться приклад сценарію із застосуванням команди ***for*** у стилі мови **C** у ньому.

Файл forC.sh

```
#!/bin/bash
# Перевірка в циклі for в стилі C
for ((i=1; i<=10; i++))
do
     echo "Наступне число: $i"
done
```

Результат виконання описаного сценарію наведено нижче:

```
debianuser@debianServ2017:~/scripts/simple$ ./forC.sh
Наступне число: 1
Наступне число: 2
Наступне число: 3
Наступне число: 4
Наступне число: 5
Наступне число: 6
Наступне число: 7
Наступне число: 8
Наступне число: 9
Наступне число: 10
```

9.8.2. Цикл while

Командний інтерпретатор **bash** підтримує й інший цикл, що є стандартним у багатьох мовах програмування високого рівня, який має назву ***while***. У **bash** цей цикл теж представляється у вигляді команди, передусім певною мірою є поєднанням інструкції ***if-then*** і циклу ***for***. Саме ця команда дозволяє визначити значення статусу виконання команди, яка у ній перевіряється, з подальшою обробкою у циклі певний перелік команд, допоки задана для перевірки команда повертає статус виконання, що дорівнює нулю. Зазвичай аналогічно до інструкції ***if-then*** у команді ***while*** для перевірки використовується команда ***test***, однак можуть бути використані й інші команди.

Перевірка статусу виконання команди здійснюється на початку кожної ітерації циклу. Як тільки команда, що перевіряється, поверне ненульовий статус виконання, дія циклу ***while*** припиняється.

Синтаксис команди ***while*** є наступним:

```
while <команда_для_перевірки>
do
     <блок_команд>
done
```

Ключовою особливістю команди ***while*** є те, що в ній має змінитися статус виконання команди перевірки в результаті виконання команд, що задані на виконання у тілі циклу. Якщо статус виконання не буде змінено на певній ітерації циклу ***while***, то він може стати нескінченним.

Найчастіше команда перевірки застосовується у формі, що передбачає використання квадратних дужок для перевірки значення однієї зі змінних командного інтерпретатора, яка застосовується в командах циклу. Саме таке використання циклу ***while*** представлено у наступному сценарії у якості прикладу.

Файл while.sh

```
#!/bin/bash
# Використання циклу while
var1=10
while [ $var1 -gt 0 ]
do
     echo $var1
   var1=$[$var1-1]
done
```

Результат виконання цього сценарію наведений нижче, та майже не представляє нічого особливого.

```
debianuser@debianServ2017:~/scripts/simple$ ./while.sh
10
9
8
7
6
5
4
3
2
1
```

9.8.3. Цикл until

Третім циклом, що підтримується у командному інтерпретаторі **bash**, є цикл ***until***. Він діє прямо протилежним способом порівняно з циклом ***while*** та вимагає від команди, яка буде перевірятися, ненульовий статус її виконання. Поки статус виконання команди перевірки є ненульовим, командним інтерпретатором **bash** виконуються команди, що входять у тіло

циклу ***until***. Однак коли команда перевірки поверне нульовий статус свого виконання, то цикл одразу завершить свою роботу.

Синтаксис команди ***until*** схожий на синтаксис команди ***while*** та має наступний вигляд:

```
until <команда_для_перевірки>
do
    <блок_команд>
done
```

Приклад сценарію з використанням команди ***until*** наведено далі.

Файл until.sh

```
#!/bin/bash
# Використання циклу until
var1=100
until [ $var1 -eq 0 ]
do
    echo $var1
    var1=$[$var1-25]
done
```

Представлено результат виконання розглянутого вище сценарію.

```
debianuser@debianServ2017:~/scripts/simple$ ./until.sh
100
75
50
25
```

9.8.4. Команди управління циклом

У командному інтерпретаторі **bash** передбачено кілька команд, які допомагають керувати тим, що відбувається у циклі. Зокрема такими командами є ***break*** та ***continue***, кожна з яких має різне призначення.

Команда ***break*** сприяє простому способу виходу з циклу в ході його роботи. Її можна використовувати для виходу з циклу будь-якого типу. Під час виконання команди командний інтерпретатор робить спробу вийти з циклу, який оброблюється на цей момент. Якщо робота програми організована за принципом застосування декількох вкладених циклів, то команда ***break*** автоматично завершує самий внутрішній цикл, що виконується. Однак іноді під час виконання внутрішнього циклу виникає необхідність зупинити зовнішній і цикл. Для цього у команді ***break*** передбачено додатковий аргумент, який вказує рівень вкладеності циклу, з якого має бути виконаний вихід. За замовчуванням аргумент дорівнює 1, що завжди означає, що вихід повинен бути виконаний з поточного циклу. Якщо, наприклад, буде встановлено значення аргумента, яке дорівнює 2, то команда ***break*** зупинить виконання зовнішнього циклу наступного рівня.

Таким чином, команда ***continue*** надає спосіб передчасного припинення обробки поточної ітерації циклу без його повного завершення. Як і команда ***break***, ця команда дозволяє також вказати, на якому рівні циклу має відбуватися пропуск команд, які залишилися, за допомогою свого додаткового аргументу, що визначає рівень циклу, на якому має відбуватися продовження роботи циклу без виконання решти операторів.

9.9. Програмування функцій у сценаріях bash

9.9.1. Оголошення функцій

Оскільки сценарії командного інтерпретатора **bash** стають складнішими, то певні частини коду, що виконують деякі завдання, можуть повторюватися в одному й тому самому сценарії не один раз. Іноді вони є досить простими фрагментами коду, однак у певних випадках може виконуватися доволі складне обчислення, що проводитиметься у сценарії декілька разів у складі більш загального процесу.

У подібних випадках доводиться займатися достатньо трудомісткою роботою з написання одних й тих самих блоків коду в тексті сценарію. Саме тому розробники з часом дійшли висновку, що є набагато зручнішим написання певного блоку програмного коду тільки один раз, а потім звернення до нього у будь-якому місці сценарію.

Для цього у командному інтерпретаторі **bash** існує засіб, що полягає у написанні функцій. Функціями є блоки коду сценарію, яким привласнюється певна назва, після чого з'являється можливість звернутися до них за цією назвою у тому місці сценарію, де це є необхідно. Для створення функцій у сценаріях передбачено цілих два формати. Перший використовує ключове слово ***function*** поряд з назвою функції, якою позначається блок коду:

```
function <назва_функції>
{
    <блок_команд>
}
```

Атрибут ***<назва_функції>*** визначає унікальну назву, що присвоюється функції. Так само ***<блок_команд>*** є однією або кількома командами командного інтерпретатора **bash**, з яких складається зазначена функція. Після виклику функції зазначені у ній команди виконуються командним інтерпретатором у тому порядку, в якому вони йдуть у тілі функції, так само, як під час виконання звичайного коду сценарію.

Другий формат визначення функції в сценарії командного інтерпретатора **bash** більшою мірою нагадує формат, застосований для визначення функцій в інших мовах програмування:

```
name()
{
     <блок_команд>
}
```

Порожні круглі дужки після назви функції вказують на те, що далі має бути їх визначення. На цей формат поширюються такі самі правила іменування функцій як і на формат з ключовим словом ***function*** попереду.

Для виклику визначені функції у сценарії варто вказати у певному рядку назву функції за тим самим принципом як і для виклику будь-якої іншої команди командного інтерпретатора. У наступному прикладі демонструється сценарій, у якому оголошується та використовується проста функція:

Файл easy_func.sh

```
#!/bin/bash
# Оголошення функції
function func
{
 echo "Це приклад функції"
}

count=1
while [ $count -le 5 ]
do
 echo -n "$count. "
 func
 count=$[$count+1]
done
echo "Кінець циклу"
func
echo "Кінець сценарію"
```

Далі наведено результат виконання представленого вище сценарію:

```
debianuser@debianServ2017:~/scripts/functions$ ./easy_func.sh
1. Це приклад функції
2. Це приклад функції
3. Це приклад функції
4. Це приклад функції
5. Це приклад функції
Кінець циклу
Це приклад функції
Кінець сценарію
```

У цьому випадку при кожному посиланні на назву функції ***func*** командний інтерпретатор звертається до визначення ***function func*** і виконує всі команди, що зазначені у ньому. Цікавим є факт, що визначення

функції у сценарії командного інтерпретатора є зовсім не обов'язковим. Однак при визначенні все ж необхідно бути обережним, оскільки під час спроби використати функцію до її оголошення буде виведено повідомлення про помилку.

Також варто зазначити, якщо у сценарії зустрічається ще одне визначення функції з тією самою назвою, то відбувається так зване перевизначення, і нова версія функції перекриває її попередню, але при цьому не буде виведено жодного повідомлення про помилку. Наступний приклад сценарію демонструє перевизначення функції:

Файл redefining.sh

```
#!/bin/bash
# Перевизначення функції

function func
{
 echo "Це перше визначення назви функції"
}
func

function func
{
 echo "Це повторне визначення однієї й тієї самої назви функції"
}
func
echo "Кінець сценарію"
```

Результат виконання представленого вище сценарію демонструє, що за другого виклику функції ***func*** дійсно відбулося перевизначення.

```
debianuser@debianServ2017:~/scripts/functions$ ./redefining.sh
Це перше визначення назви функції
Це повторне визначення однієї й тієї самої назви функції
Кінець сценарію
debianuser@debianServ2017:~/scripts/functions$
```

Однак набагато краще використовувати функції, задаючи кожній з них унікальну назву. Це дозволить запобігти певній плутанині у роботі сценарію командного інтерпретатора загалом.

9.9.2. Повернення значення з функції

З точки зору командного інтерпретатора **bash**, усі функції розглядаються як свого роду міні-сценарії, кожен з яких має свій статус виконання. Передбачається три способи формування значення статусу виконання функцій, що оголошені користувачем. Усі ці способи будуть розглянуті далі у поточному розділі.

Статус виконання, що задається за замовчуванням

За замовчуванням статус виконання функції визначається як статус виконання останньої команди, що була зазначена у її тілі. Цей статус як і для будь-якої стандартної команди командного інтерпретатора **bash**, присвоюється в уже відому нам змінну ***$?***.

У якості прикладу розглянемо наступний сценарій, а потім і результат його виконання:

Файл return_value1.sh

```
#!/bin/bash

func()
{
 echo "Спроба виведення інформації про неіснуючий файл"
 ls -l noExistFile.txt
}

echo "Тестування функції..."
func
echo "Статус виходу функції: $?"
```

```
debianuser@debianServ2017:~/scripts/functions$ ./return_value1.sh
Тестування функції...
Спроба виведення інформації про неіснуючий файл
ls: cannot access 'noExistFile.txt': No such file or directory
Статус виходу функції: 2
```

У цьому випадку статус виконання функції дорівнює ***1***, оскільки виконання останньої команди у функції завершилося помилкою, яка полягає у тому, що не був знайдений файл, назву якого вказана у якості аргументу команди ***ls***. Однак це не дозволяє нам дізнатися, чи закінчилося виконання всіх інших команд у функції успішно.

У якості прикладу розглянемо наступний сценарій та результат його виконання:

Файл return_value1.sh

```
#!/bin/bash

func()
{
 ls -l noExistFile.txt
 echo "Файлу noExistFile.txt не існує"
}

echo "Тестування функції..."
func
```

```
echo "Статус виходу функції: $?"
```

```
debianuser@debianServ2017:~/scripts/functions$ ./return_value1.sh
Тестування функції...
ls: cannot access 'noExistFile.txt': No such file or directory
Спроба виведення інформації про неіснуючий файл
Статус виходу функції: 0
```

У наведеному вище сценарії останньою командою у тілі функції була команда ***echo***, виконання якої було завершено цілком успішно. Саме тому функція має статус виконання, який дорівнює нулю, незважаючи на те, що виклик однієї з команд у ній завершився з помилкою. Таким чином, розглянутий підхід, який передбачає використання заданого за замовчуванням статусу виходу функції, не завжди є виправданим. Існують і більш зручні способи формування значення статусу виконання функції, які будуть розглянуті далі.

Використання команди return

Другим способом повернення статусу виконання функції у командному інтерпретаторі **bash** є використання команди ***return***, яка дозволяє задати одне цілочисельне значення для визначення статусу виконання функції, що є простим способом його завдання програмним шляхом. Розглянемо наступний приклад сценарію, у якому використовується команда ***return***.

Файл return_value2.sh

```
#!/bin/bash
# Використання команди return в функції

function func
{
    read -p "Введіть число: " value
    echo "Подвоєння числа $value"
    return $[$value*2]
}

func
echo "Нове число дорівнює $?"
```

Далі наведений результат виконання сценарію, код якого представлено вище:

```
debianuser@debianServ2017:~/scripts/functions$ ./return_value2.sh
Введіть число: 52
Подвоєння числа 52
Нове число дорівнює 104
```

У сценарії оголошена функція ***func***, яка подвоює значення, що вводиться кінцевим користувачем, а потім записується у змінну ***value***. Потім у функції здійснюється повернення сформованого результату за

допомогою команди ***return***, а повернене значення присвоюється у змінну ***$?***.

Такий спосіб повернення значення є набагато кращим ніж спосіб повернення за замовчуванням, однак використовуючи його потрібно бути обережним. Зокрема, завжди потрібно пам'ятати наступне:

– після виконання функції варто відразу обробити значення змінної ***$?***, оскільки після виконання іншої команди воно буде втрачено;

– статус виконання функції має знаходитися у діапазоні значень від ***0*** до ***255***.

Спроба повернути будь-яке значення, що є більшим за ***255***, призведе до неочікуваних результатів. Виконаємо ще раз сценарій з останнього прикладу, увівши у якості вхідних даних число ***130***.

```
debianuser@debianServ2017:~/scripts/functions$ ./return_value2.sh
Введіть число: 130
Подвоєння числа 130
Нове число дорівнює 4
```

Очікувано, що результатом подвоєння числа ***130*** є число ***260***, однак функція повернула статус виконання, що дорівнює ***4***. Це досить легко пояснити, згадавши, що при спробі повернення у якості статусу виконання команди число, що є більшим за ***255***, командний інтерпретатор **bash** застосує до нього операцію залишку від ділення на число ***256***.

Таким чином, цей спосіб повернення значення не може застосовуватися, якщо необхідно виконати повернення цілочисельного значення, яке перевищує ***255***, а також рядкового значення.

Перехоплення результатів виведення функції

Аналогічно до того, що можна присвоїти у певну змінну результати виконання деякої команди, подібну дію можна виконати, якщо замість стандартних команд **bash** використовувати фуннції. Такий спосіб може бути використаний для отримання виведення будь-якого значення з функції для присвоювання його у змінну. Він має назву **перехоплення** та виглядає наступним чином:

<змінна>=`<функція>`

Така команда присвоює результат виведення функції ***`<функція>`*** у змінну командного інтерпретатора ***<змінна>***. Однак для цього потрібно, щоб функція обов'язково щось виводила на стандартний потік виведення. Тобто для того, щоб функція виводила якісь дані, які потім можна було б присвоїти у змінну, останньою командною у її тілі має бути команда ***echo***. Саме у такому випадку і можна здійснити перехоплення.

Наступний приклад демонструє сценарій командного інтерпретатора **bash**, у якому використовується перехоплення результатів виведення функції у змінну.

Файл return_value3.sh

```
#!/bin/bash
# Використання команди echo для повернення значення

function func
{
      read -p "Введіть число: " value
      echo $[$value*2]
}

result=`func`
echo "Нове число дорівнює $result"
```

Наведений приклад фактично показує змінений попередній приклад, який представляв сценарій з реалізованою функцією, що подвоює введене користувачем значення. Однак у такому випадку значення, яке буде повернене з функції, завжди буде правильним. У цьому можна переконатися, викликавши функцію на виконання.

```
debianuser@debianServ2017:~/scripts/functions$ ./return_value3.sh
Введіть число: 130
Нове число дорівнює 260
debianuser@debianServ2017:~/scripts/functions$ ./return_value3.sh
Введіть число: 2000
Нове число дорівнює 4000
```

Тепер у новій версії функції подвоєння числа замість команди ***return*** використовується команда ***echo*** для відображення результату обчислення. Таким чином, у сценарії відбувається перехоплення результатів виведення функції ***func***, а не перегляд статусу її виконання для отримання відповіді.

Однак під час використання прийому перехоплення результату виведення функцій також необхідно бути обережним. Справа у тому, що коли у функції використовується більш ніж одна команда ***echo***, у змінну буде присвоєне все, що було виведено почергово всіма такими командами.

Якщо подивитися уважно на код представленого вище сценарію, то можна зауважити, що у функції ***func*** фактично виводяться два повідомлення. Тільки різниця у тому, що перше виводиться за допомогою команди ***read*** з ключем ***-p*** і відображає запрошення кінцевому користувачу ввести число, яке необхідно буде подвоїти. Повідомлення, що виводиться командою ***read*** фактично не розглядається як частина виведення

стандартного потоку **STDOUT**, саме тому воно не було присвоєне у змінну ***result*** разом з подвоєним значенням. У випадку, якщо б для повідомлення запрошення ввести вхідне число використовувалася команда ***echo***, воно також було б перехоплено та збережено у змінну ***result*** разом з подвоєним числовим значенням.

9.9.3. Передача аргументів у функцію

Як зазначалося вище, командний інтерпретатор **bash** завжди розглядає функцію як міні-сценарій. Це означає, що у функцію, як і у звичайний сценарій можна передавати аргументи. Обробляються передані аргументи так само як і у звичайному сценарії, а саме за допомогою змінних позиційних параметрів. За аналогією до сценаріїв зміна ***$0*** зберігає назву функції, а змінні ***$1***, ***$2*** і так далі – аргументи командного рядка, передані у функцію. Усі інші розглянуті раніше спеціальні змінні також можна використовувати у функціях.

При виклику функції у сценарії їй необхідно передавати аргументи у тому ж рядку після її назви, наприклад:

```
func1 $value1 10
```

У наступному прикладі демонструється сценарій, у якому визначена функція, що виконує певні дії залежно від кількості переданих їй аргументів.

Файл params.sh

```
#!/bin/bash

function add
{
 if [ $# -eq 0 ] || [ $# -gt 2 ]
 then
  echo "Помилка!"
 elif [ $# -eq 1 ]
 then
  echo -n "Подвоєння числа $1: "
  echo $[$1*2]
 else
  echo -n "Додавання чисел $1 та $2: "
  echo $[$1+$2]
 fi
}

echo -n "Введіть одне або два числа: "
read a b
add $a $b
```

Тепер дамо пояснення. У сценарії, код якого представлено вище, оголошена функція ***add***, яка спочатку перевіряє кількість переданих до неї аргументів з головного сценарію. Якщо не було передано жодного аргументу або більше двох, то ця функція виводить повідомлення про помилку. Так само якщо заданий один аргумент, функція подвоює його, а якщо два, то складає їх для отримання суми.

Наведемо результати виконання сценарію для всіх розглянутих вище випадків:

```
debianuser@debianServ2017:~/scripts/functions$ ./params.sh
Введіть одне або два числа:
Помилка!
debianuser@debianServ2017:~/scripts/functions$ ./params.sh
Введіть одне або два числа: 6
Подвоєння числа 6: 12
debianuser@debianServ2017:~/scripts/functions$ ./params.sh
Введіть одне або два числа: 6 9
Додавання чисел 6 та 9: 15
debianuser@debianServ2017:~/scripts/functions$ ./params.sh
Введіть одне або два числа: 6 9 12
Помилка!
```

9.9.4. Обробка змінних у функції

Під час програмування сценаріїв командної оболонки **bash** можна зіштовхнутися ще з одним поняттям, неправильне розуміння якого може викликати проблему стосовно області визначення змінної. Мова йде про ділянку програмного коду, в якому змінна є видимою. Порівняно зі звичайними змінними, визначені у тілі функцій змінні можуть мати іншу область свого визначення, тобто приховані від іншої частини сценарію. У функціях можуть використовуватися змінні двох типів: **локальні** та **глобальні.**

Глобальні змінні

Глобальними змінними є змінні, що видимі у будь-якому місці сценарію. Якщо глобальна змінна була визначена в основній частині сценарію, то її значення можна отримати також і у функції. І навпаки, якщо вона визначена в тілі функції, до неї можна звернутися і в основній частині сценарію. Усі змінні, що визначені у сценарії, за замовчуванням розглядаються як глобальні.

Наступний сценарій у якості прикладу представляє роботу з глобальною змінною у функції.

Файл global.sh

```
#!/bin/bash

function double
{
```

```
  #value – глобальна змінна
  value=$[$value*2]
}

read -p "Введіть число: " value
double
echo "Нове число: $value"
```

Далі представлений результат виконання зазначеного вище сценарію:

```
debianuser@debianServ2017:~/scripts/functions$ ./global.sh
Введіть число: 45
Нове число: 90
```

У розглянутому сценарії змінна ***value*** була оголошена поза функцією. Також за межами функції їй було присвоєно значення. Однак під час виконання функції ***double*** і ця змінна, і її значення залишаються дійсними. Присвоєнне нове значення змінній ***value*** буде залишатися дійсним під час обробки змінної в основній частині сценарію.

Однак організація роботи зі змінними за таким принципом може призвести до небажаних помилок, особливо якщо одна й та сама функція має застосовуватися в різних сценаріях командного інтерпретатора. Для цього потрібно точно знати, які змінні використовуються у функції, включаючи будь-які змінні, застосовані для обчислення значень, які не призначені для повернення у сценарій.

У якості наступного прикладу представлено сценарій, виконання якого призводить до отримання зовсім неочікуваних результатів:

Файл global1.sh

```
#!/bin/bash

function func
{
 temp=$[$value+5]
 result=$[$temp*2]
}

temp=4
value=6
func
echo "Результат: $result"
if [ $temp -gt $value ]
then
 echo "temp більше ніж value"
else
 echo "temp менше ніж value"
```

```
fi
```

Результат виконання такого сценарію є наступним:

```
debianuser@debianServ2017:~/scripts/functions$ ./global1.sh
Результат: 22
temp більше ніж value
```

У цьому випадку змінна ***temp*** застосовується як у функції, так і у самому сценарії. Тому зміна її значення в основній частині сценарію, яка не була передбачена у функції, і призводить до отримання неочікуваного результату. Для вирішення подібних проблем у функціях можна оголосити змінні у якості локальних, що буде розглянуто нижче.

Локальні змінні

Замість того, щоб використовувати у функціях глобальні змінні, можна оголосити ті, які використовуються тільки у самій функції виключно у якості локальних. Для цього перед оголошенням змінної достатньо задати ключове слово ***local***, наприклад:

```
local temp
```

Також оголошення змінної у якості локальної може бути задано і під час присвоєння їй певного значення, наприклад:

```
local temp=$[$value+5]
```

Ключове слово ***local*** є гарантією того, що область дії змінної буде обмежуватися тільки межами функції. Якщо й будуть існувати змінні з такими самими назвами поза функцією, то це вже будуть дві різні змінні, що матимуть абсолютно різні значення. Приклад наступного сценарію демонструє оголошення та використання локальної змінної у функції.

Файл local.sh

```
#!/bin/bash

function func
{
 local temp=$[$value+5]
 result=$[$temp*2]
}

temp=4
value=6
func
echo "Результат: $result"
if [ $temp -gt $value ]
then
 echo "temp більше ніж value"
else
```

```
 echo "temp менше ніж value"
fi
```

У цьому випадку зміна значення змінної ***temp*** в межах функції ***func*** не призводить до зміни значення, що було присвоєне змінній ***temp*** в основній частині сценарію. Тепер результат виконання такого сценарію є цілком очікуваним:

```
debianuser@debianServ2017:~/scripts/functions$ ./local.sh
Результат: 22
temp менше ніж value
```

9.9.5. Змінні типу масиву і функції

Окремого розгляду потребує робота з масивами у функціях, що й буде описуватися у наступному розділі.

Передача масивів у функції

Особливої уваги заслуговує спосіб передачі змінної, що представляє масив, у функцію як аргумент командного рядка. Для початку розглянемо наступний приклад сценарію командного інтерпретатора, у якому здійснюється спроба передачі масиву у функцію, але завершується невдало.

Файл func_array1.sh

```
#!/bin/bash
# Спроба передати змінну з типом масиву

function testit
{
      echo "Аргументи командного рядка: $@"
      thisarray=$1
      echo "Переданий масив: ${thisarray[*]}"
}

myarray=(1 2 3 4 5)
echo "Оригінальний масив: ${myarray[*]}"
testit $myarray
```

При спробі використання змінної-масиву в якості аргументу функції, у неї буде переданий тільки перший елемент масиву, що видно з результату виконнаня зазначеного вище сценарію:

```
debianuser@debianServ2017:~/scripts/functions$ ./func_array1.sh
Оригінальний масив: 1 2 3 4 5
Аргументи командного рядка: 1
Переданий масив: 1
```

Для вирішення такої проблеми потрібно розібрати змінну, що представляє масив, на окремі значення, після чого задати їх у якості аргументів функції окремо. А вже у тілі самої функції можна повторно

зібрати всі передані аргументи у абсолютно нову змінну, яка при цьому теж буде представляти масив. Як приклад розглянемо наступний сценарій.

Файл func_array2.sh

```
#!/bin/bash
# Перевірка передачі змінної з типом масиву в функцію

function testit
{
      local newarray
      newarray=(`echo "$@"`)
      echo "Новий масив: ${newarray[*]}"
}

myarray=(1 2 3 4 5)
echo "Оригінальний масив: ${myarray[*]}"
testit ${myarray[*]}
```

У сценарії використовується змінна ***myarray*** для збереження всіх окремих значень елементів масиву, що передаватимуться у командному рядку при виклику функції. У функції оголошується локальна змінна ***myarray***, яка міститиме масив усіх переданих у функцію аргументів командного рядка. У функції отриманий масив може бути використаний так само, як і будь-який інший.

За результатом виконання наведеного вище сценарію, переконаємося, що відображений прийом передачі масиву в якості аргументу у функцію дійсно правильно спрацював.

```
debianuser@debianServ2017:~/scripts/functions$ ./func_array2.sh
Оригінальний масив: 1 2 3 4 5
Новий масив: 1 2 3 4 5
```

Також розглянемо приклад сценарію, у якому оголошується функція, що обчислює суму елементів масиву, що переданий як параметр.

Файл func_array3.sh

```
#!/bin/bash
# Розрахунок суми елементів масиву

function addarray
{
      local sum=0
      local newarray
      newarray=(`echo "$@"`)
      for value in ${newarray[*]}
      do
          sum=$[$sum+$value]
      done
```

```
        echo $sum
}
myarray=(1 2 3 4 5)
echo "Оригінальний масив: ${myarray[*]}"
arg1=`echo ${myarray[*]}`
result=`addarray $arg1`
echo "Результат: $result"
```

Тут, у функції ***addarray***, здійснюється ітерація за значеннями елементів масиву для отримання їх загальної суми. У змінну з типом масиву ***myarray*** можна помістити будь-яку кількість значень, і функція ***addarray*** підсумує їх. Результат виконання зазначеного сценарію буде наступним:

```
debianuser@debianServ2017:~/scripts/functions$ ./func_array3.sh
Оригінальний масив: 1 2 3 4 5
Результат: 15
```

Повернення масивів з функцій

Аналогічний спосіб може бути використаний і для повернення масиву з функції до сценарію командного інтерпретатора. У такому разі функцією має бути використана команда ***echo*** з метою виведення окремих значень елементів масиву у відповідному порядку. В основній частині сценарію командного інтерпретатора повинна бути проведена повторна збірка цих елементів як нової змінної, що є масивом.

Наступний приклад відображає сценарій, що демонструє прийом повернення масиву з функції.

Файл return_array.sh

```
#!/bin/bash
# Повернення значення масиву

function double_arrValues
{
      local origarray
      local newarray
      local elements
      local i
      origarray=(`echo "$@"`)
      newarray=(`echo "$@"`)
      elements=$[$#-1]
      for ((i=0; i<=$elements; i++))
      {
          newarray[$i]=$[${origarray[$i]}*2]
      }
      echo ${newarray[*]}
```

```
}

myarray=(1 2 3 4 5)
echo "Оригінальний масив: ${myarray[*]}"
arg=`echo ${myarray[*]}`
result=(`double_arrValues $arg`)
echo "Новий масив: ${result[*]}"
```

Прокоментуємо програмний код представленого вище сценарію. В основній його частині оголошується змінна ***myarray***, якій присвоюється масив з п'яти елементів. У функцію, призначення якої полягає у подвоєнні кожного елемента масиву, він передається через допоміжну змінну ***arg***. Тим самим у функції здійснюється повторна збірка масиву та отримується нова змінна, що представляє собою масив та має назву ***newarray***. Саме ця змінна і є вихідною змінною, яка потім перехоплюється в основній частині сценарію. Завершується ж функція операціями подвоєння кожного елемента масиву, після цього його кожне нове значення поміщується у копію змінної-масиву. Таким чином, в основній частині сценарію результат виведення функції застосовується для повторного складання нової змінної з типом масиву на основі отриманих значень.

Результат сценарію командного інтерпретатора, представленого в останньому прикладі, наведено нижче.

```
debianuser@debianServ2017:~/scripts/functions$ ./return_array.sh
Оригінальний масив: 1 2 3 4 5
Новий масив: 2 4 6 8 10
```

9.9.6. Рекурсивний виклик функцій

Однією з особливостей, якій локальні змінні наділяють функцію, є автономність. Якщо функція є автономною, то їй не потрібно використовувати ніяких ресурсів за її межами, окрім тих змінних, що були передані їй у командному рядку зі сценарію.

Зокрема ця особливість дозволяє здійснити рекурсивний виклик функції, яка полягає у виклику функції у самій собі. Зазвичай, у рекурсивній функції передбачено певне базове значення, до якого сходитиметься рекурсія у кінцевому підсумку. Одним з класичних прикладів рекурсивного алгоритму є обчислення факторіала числа. Нагадаємо, що факторіалом певного числа є добуток членів натурального ряду чисел, що закінчується відповідним числом. Наприклад, для знаходження факторіала числа 4, потрібно виконати таке обчислення:

4! = 1 * 2 * 3 * 4 = 120

Якщо використовувати рекурсію, представлений вираз можна скорочено уявити у наступній формі:

x! = x * (x-1)!,

що означатиме «факторіал числа **x** дорівнює добутку числа **x** на факторіал числа **x-1**». Таке обчислення може бути реалізоване за допомогою наступного простого рекурсивного сценарію:

Файл recursion.sh

```
#!/bin/bash
# Рекурсивний виклик функції на прикладі обчислення факторіала

function factorial
{
 if [ $1 -eq 1 ]
 then
  echo 1
 else
  local temp=$[$1-1]
  local result=`factorial $temp`
  echo $[$result*$1]
 fi
}

read -p "Введіть число: " num
res=`factorial $num`
echo "Факторіал числа $num дорівнює $res"
```

У розглянутому прикладі сценарію функція ***factorial*** викликає саму себе для обчислення значення факторіала. Далі наведений результат його виконання.

```
debianuser@debianServ2017:~/scripts/functions$ ./recursion.sh
Введіть число: 5
Факторіал числа 5 дорівнює 120
```

9.9.7. Створення бібліотеки

Як вже було зазначено вище, функції дозволяють позбутися від необхідності багаторазового написання одного й того самого коду в одному сценарії, однак іноді бувають випадки, що однаковий код може зустрітися в абсолютно різних сценаріях командного інтерпретатора. У таких випадках немає ніякого сенсу реалізовувати у кожному сценарії окрему функцію, що включатиме код, який повторюється багато разів, тільки для того, щоб використовувати її у кожному сценарії тільки один раз.

Для цієї проблеми передбачено інше рішення, що полягає у створенні файлу бібліотеки, у якому можна оголосити власні функції і після цього

замість повторного визначення однакових функцій у кожному сценарії окремо посилатися на файл бібліотеки у всіх сценаріях, де будуть потрібні оголошені у ньому функції.

Першим кроком цього процесу є створення загального файлу бібліотеки, у якому будуть міститися функції, що можуть бути необхідними у сценаріях. Далі наводиться приклад простого файлу бібліотеки з назвою ***lib***, у якому оголошені чотири нескладні функції, що виконують відомі всім арифметичні операції над двома числами.

Файл lib

```
# Бібліотека функцій для застосування в сценаріях

# Додавання
function add
{
 echo $[$1+$2]
}

# Віднімання
function sub
{
 echo $[$1-$2]
}

# Множення
function mult
{
 echo $[$1*$2]
}

# Ділення
function div
{
 if [ $2 -ne 0 ]
 then
  echo $[$1/$2]
 else
  echo -1
 fi
}
```

Для використання функцій, що містяться у файлі бібліотеки, спочатку потрібно його включити у якості джерела. Для цього призначена команда

source, яка має скорочений псевдонім – крапку (**.**). Після зазначення цієї команди необхідно як її аргумент вказати асболютний або відносний шлях до файлу бібліотеки.

Наступним прикладом є сценарій, який використовує описаний вище файл бібліотеки у якості джерела, завдяки чому можна скористатися відповідними функціями.

Файл use_lib.sh

```
#!/bin/bash
# Використання функцій з бібліотеки lib

. ~/scripts/functions/lib
read -p "Введіть 2 числа: " val1 val2
res1=`add $val1 $val2`
res2=`sub $val1 $val2`
res3=`mult $val1 $val2`
res4=`div $val1 $val2`

echo "Результат додавання чисел: $res1"
echo "Результат віднімання чисел: $res2"
echo "Результат множення чисел: $res3"
echo "Результат ділення чисел: $res4"
```

У представленому сценарії виклик функцій, що визначені у файлі бібліотеки ***lib***, відбувається цілком успішно, що й продемонстровано у результаті його виконання.

```
debianuser@debianServ2017:~/scripts/functions$ ./use_lib.sh
Введіть 2 числа: 75 15
Результат додавання чисел: 90
Результат віднімання чисел: 60
Результат множення чисел: 1125
Результат ділення чисел: 5
```

РОЗДІЛ 10.
ГРАФІЧНІ РОБОЧІ СТОЛИ, ПРОГРАМУВАННЯ СЦЕНАРІЇВ ДЛЯ ГРС

10.1. Графічні оболонки Linux

Графічна оболонка середовища являє собою оболонку операційної системи, яка забезпечує зв'язок користувача з різними функціями системи, що, дозволяє управляти нею.

У **Windows** фактично всього одна графічна оболонка, тобто користувач може змінювати графічну тему, змінювати налаштування деяких графічних елементів (наприклад змінити іконку в папці), але сам графічний інтерфейс буде залишатися тим же.

У **Linux** таких графічних оболонок декілька і користувач може встановити їх усі, а при вході в систему обрати ту, яка йому більше до вподоби. Найбільш поширені середовища в Linux – це **KDE** та **GNOME**. Вони представляють два конкуруючих середовища, якщо у користувача більш-менш потужний комп'ютер, то вибір графічної оболонки зупиниться на цих двох середовищах, а вже яку встановити буде залежати тільки від нього.

Обираючи графічне середовище, користувач обирає набір програм, з якими буде працювати. Середовище **KDE** використовує для роботи бібліотеку **Qt**, а **GNOME** – **GTK**. Якщо був обраний KDE, то будуть встановлені програми, що працюють завдяки бібліотеці Qt, якщо в GNOME, то відповідно встановлюються програми, засновані на елементах бібліотеки GTK. Як приклад, у якості файлового менеджера під час вибору KDE буде встановлено Dolphin, а при виборі GNOME – Nautilus. Але якщо дисковий простір дозволяє, то звичайно ж, можна встановити їх разом і при завантаженні операційної системи можна вибирати в якій графічній оболонці ми сьогодні захочемо працювати, але запустити програму під невластивим їй середовищем не вийде.

10.1.1. Графічна оболонка GNOME

GNOME (GNU Network Object Model Environment) – графічний робочий стіл, який побудований повністю з вільних і відкритих бібліотек. Він був створений двома мексиканськими програмістами – Мігелем де Ікаса і Федеріко Мена. GNOME є частиною проекту GNU і використовується у якості графічної оболонки для дистрибутивів побудованих на ядрі Linux. Як графічні і мультимедіа-інструменти в GNOME використовуються деякі проекти freedesktop.org. Бібліотека **Cairo** забезпечує виведення векторної

графіки. Вона використовується в GTK+ для відображення елементів інтерфейсу.

GNOME побудований на кількох відкритих проектах, наприклад GTK+, GConf, LibXML, GNOME VFS та інших. Застосунки до GNOME пишуться на різних мовах програмування C++ (gtkmm), C# (Gtk#), Java (Java-GNOME), Python (PyGTK), Perl (gtk2-perl) тощо, хоча сам GNOME написаний на мові C.

Скріншот робочого столу GNOME для дистрибутиву Debian представлений на рис. 10.1.

Рис. 10.1. Графічний робочий стіл GNOME

Для зберігання налаштувань графічного робочого столу і застосунків та відстеження їхніх змін використовується – GConf, які зберігаються в XML-файлах. Для користувача і застосунків вони представляються у вигляді опцій представлених графічно у вигляді дерева, подібно до реєстру Windows. Застосунки, що використовують GConf, можуть надавати описи для окремих опцій, які потім будуть перекладені іншими мовами в рамках загального процесу локалізації середовища.

При зміні якогось значення налаштування система GConf автоматично сповіщає про це всі «зацікавлені» застосунки, чим забезпечується миттєве застосування нових налаштувань. Крім того, GConf дозволяє системним адміністраторам обмежити зміну тих або інших налаштувань, зробивши їх обов’язковими для користувачів.

10.1.2. Графічна оболонка XFCE

XFce є графічною оболонкою, що побудована на основі інструментального пакету GTK+, який використовується в Gnome, але набагато

легше і призначена для тих, кому потрібен простий, ефективно працюючий стіл, який зручно використовувати і налаштовувати. Переваги XFce:

– простий, зручний у використанні робочий стіл, який повністю налаштовується за допомогою миші, з інтерфейсом «drag and drop»;

– інтегрований віконний менеджер, менеджер файлів, управління звуком, модуль сумісності з Gnome та інше;

– стандартизовані меню і панелі інструментів, комбінації клавіш, колірні схеми і т. д.;

– можливість використання тем, так як використовує **GTK+;**

– XFce швидка, легка і ефективна оболонка: ідеальна для застарілих/ слабких машин або машин з обмеженою пам'яттю.

XFce підтримується більшою кількістю операційних систем побудованих на ядрі Linux. Для коректної роботи потребує наявності додаткових бібліотек:

- **libxfce4util** – бібліотека для не графічних допоміжних функцій;
- **libxfcegui4** – допоміжні функції для створення графічних інтерфейсів;
- **libxfce4mcs** – бібліотека керування налаштуваннями, яка використовується практично всіма компонентами Xfce 4.

На рис. 10.2 представлений скріншот робочого столу XFCE для дистрибутиву Xubuntu.

Рис. 10.2. Графічний робочий стіл XFCE

10.1.3. Графічна оболонка KDE

Зазначимо, що третім найбільш відомим графічним робочим столом є KDE PLASMA. Його графічне середовище розроблене на основі бібліотек QT та підтримується командою KDE.

Розробка KDE розпочалася в 1996 р. Маттіасом Еттріхом, студентом Тюбінгенського університету. Його турбували проблеми робочого столу в

UNIX, однією з яких була відсутність програм, інтерфейс яких виглядав би і поводив себе однаково.

Маттіас Еттріх запропонував не просто створення набору програм, а середовище для персональних комп'ютерів, у якому користувач міг очікувати однакової поведінки програм. Крім того, він хотів зробити це середовище простим і зрозумілим в експлуатації.

Інструментарієм розробки інтерфейсу користувача було вибрано Qt. Ініціатива отримала розповсюдження, а також стараннями розробників, до початку 1997 р. середовище складалося з достатнього набору програм. Водночас бібліотека Qt розповсюджувалася відповідно до вільних умов ліцензування, учасники проекту GNU були стурбовані тим фактом, що вільне середовище і програми, що входять до його складу, створюються з використанням невільних інструментів. Це стало причиною створення двох проектів: «Harmony» і GNOME. Маючи однакові цілі (створення вільного середовища вільними коштами), два проекти обрали зовсім різні шляхи реалізації задуманого. Проект Harmony ставив своїм завданням переписування бібліотек Qt, з наступним випуском результату відповідно до вільних умов ліцензування. Проект GNOME – повну відмову від використання Qt.

Часто графічний робочий стіл встановлюється з набором KDE Software Compilation (KDE SC). KDE SC складається з трьох складових частин: базової платформи, набору додаткових програм і графічного робочого столу Plasma. KDE містить велику кількість програм та документації, які змінили відношення до *nix систем (Unix, Linux, BSD тощо), як до серверних систем. Маючи стабільний фундамент, що базується на надійності ядра Linux, цей проект забезпечує простоту користування та однаковий зовнішній вигляд вікон програм, що робить зручним та простим використання ОС навіть для недосвідченого користувача.

На рис. 10.3 представлений скріншот робочого столу KDE PLASMA.

Рис. 10.3. Графічний робочий стіл KDE PLASMA

10.2. Написання сценаріїв з використанням візуальних елементів графічних середовищ

Колись сценарії командного інтерпретатора представляли один з найпоширеніших способів організації роботи в системі, та з часом вони стали сприйматися як похмурі та нудні. Але так не має бути, якщо сценарій призначений для виконання в графічному середовищі. Передбачено широкий спектр можливостей забезпечення взаємодії з користувачем сценарію, не базованих виключно на інструкціях ***read*** і ***echo***.

Утиліта ***dialog*** – одна з небагатьох, яка допомагає створювати доброзичливі до користувача (в плані призначеного для користувача інтерфейсу) сценарії і програми. Вона може конструювати різні діалогові вікна, не потребуючи при цьому запущений графічний інтерфейс – достатньо лише консолі, адже для малювання використовується бібліотека **ncurses**. Як наслідок, ***dialog*** добре пристосований для використання у сценаріях **bash**, в яких він застосовується у більшості випадків.

Варто відзначити, що існують аналоги ***dialog'a***, спеціально розраховані на використання в графічному оточенні – універсальний ***Xdialog*** (повністю сумісний з dialog), ***gtkdialog*** (вікна малюються за допомогою графічної бібліотеки GTK), ***kdialog*** (для роботи у графічному середовищі **KDE**) і ***zenity*** (призначений для використання разом з графічним середовищем **Gnome**). Зазначимо, що Xdialog, крім стандартних діалогових вікон з ***dialog'a***, має ще й свої: деревовидні списки, поле для редагування багаторядкових текстових даних і т. п.

У команді ***dialog*** використовують параметри командного рядка для визначення того, який графічний елемент Windows має бути сформований. Термін графічний елемент (widget) застосовують у пакеті ***dialog*** для позначення графічного об'єкта, відповідного одному з об'єктів Windows.

Короткий список діалогових вікон, які ***dialog*** здатний малювати:

- вікна з кнопками Yes/No;
- меню;
- вікна з полями введення;
- інформуючі вікна і вікна з текстом;
- поля з радіокнопками;
- вікна для вибору файлів;
- вікна для вибору необхідного часу, дати;
- спеціальні вікна для запиту пароля;

і багато іншого.

10.2.1. Програма zenity для створення діалогових вікон для робочих столів GNOME та XFce

Zenity – це засіб створення діалогових вікон в режимі командного рядка. Варто зазначити, що насправді діалогові вікна створюються засобами **Gtk+**, тому в системі мають бути встановлені відповідні бібліотеки. Від аналогічних програм ***zenity*** відрізняється більш витонченими засобами реалізації GUI-елементів.

Завантажити актуальну версію цієї утиліти можна на Web-сайті розробників [**http://library.gnome.org/users/zenity/**], та в репозиторіях деяких дистрибутивів Linux.

Застосування ***zenity*** для окремих команд в інтерактивному режимі не настільки ефективне, як при написанні сценаріїв. У сценаріях командної оболонки часто потрібна взаємодія з користувачем, щоб повідомити якусь інформацію, наприклад, про виникнення «нестандартної» ситуації. Також варто відображати інформацію щодо стану виконання операції, що продовжується тривалий час. Крім цього, іноді цим сценарієм можна отримати деяку інформацію від користувача: вибір варіанта відповіді на поставлене запитання, вибір файлу із запропонованого списку і т. д. Усе це можна організувати за допомогою ***zenity***.

Необхідно уточнити, що після закриття діалогового вікна, ***zenity*** повертає числовий код завершення операції:

– 0 – означає, що користувач натиснув у діалоговому вікні кнопку **«OK»** або **«Закрити» (Close)**;

– 1 – означає, що користувач натиснув кнопку **«Скасувати» (Cancel)** або скористався функціями (кнопками) вікна, щоб закрити його;

– -1 – повідомляє про те, що операція завершилася з помилкою;

– 5 – діалогове вікно було закрите після закінчення інтервалу часу очікування.

Далі розглянемо, які типи діалогових вікон можна створити з використанням програми ***zenity***.

Діалогове вікно для виведення повідомлень

Щоб розпочати використовувати ***zenity*** на практиці, не потрібно володіти особливими знаннями або вміннями, досить познайомитися з різними ключами, що дозволяють повною мірою використовувати можливості цієї програми.

У ***zenity*** визначено чотири типи діалогових вікон для виводу повідомлень:

– помилка (ключ ***--error***);

– інформація (ключ ***--info***);

– питання (ключ ***--question***);

– попередження (ключ ***--warning***).

Просте повідомлення визначається наступною командою:

zenity --info --title="Testing" --text="This is a test"

Результат виконання цієї команди представлений на рис. 10.4.

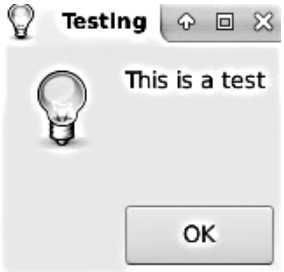

Рис. 10.4. Діалогове вікно виведення повідомлень, створене командою ***zenity***

Якщо застосовуваний емулятор термінала підтримує мишу, то з'являється додаткова можливість натиснути на кнопку ОК для закриття діалогового вікна. Для моделювання натиснення можна використовувати команди клавіатури; у цьому випадку достатньо натиснути клавішу **<Enter>**.

Діалогове вікно для вибору варіанта відповіді

Діалогове вікно з питанням є більш складним варіантом графічного елемента повідомлення у ***zenity***, який дозволяє користувачеві відповісти на питання, що відображене у вікні, позитивно або негативно. Цей елемент створює в нижній частині вікна дві кнопки: **«Yes»** та **«No»**. Користувач може переходити від однієї до іншої за допомогою миші, клавіші табуляції або клавіш зі стрілками. Щоб обрати кнопку, користувач може натиснути клавішу пробілу або **<Enter>**. Для створення діалогового вікна командою ***zenity*** потрібно вказати ключ ***--question***.

Нижче наведено приклад використання графічного елемента ***yesno***, а результат представлений на рис. 10.5.

zenity --question --title="Please answer" --text="Is this thing on?"

echo $? (була натиснута кнопка «No»)

1

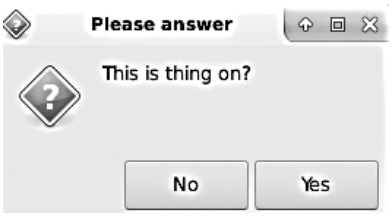

Рис. 10.5. Діалогове вікно для вибору варіанта відповіді, створене командою ***zenity***

Статус виходу команди ***zenity*** встановлюється залежно від того, яку кнопку обрав користувач. Якщо **«No»**, статусом виходу є 1, а якщо обрано кнопку **«Yes»**, статус приймає значення 0.

Також наведемо приклад сценарію **bash** для графічних робочих столів зі створенням діалогового вікна вибору варіанта відповіді.

Файл yesno.sh

```
#!/bin/bash

zenity --question \
   --title="Виведення вмісту поточної директорії" \
   --text="Чи хочете ви застосувати довгий формат"

case $? in
     '0 ')
      ls -l;;
     '1')
      ls;;
     '-1')
      echo "There was an error!"
      exit 1;;
esac
```

Діалогове вікно для введення та редагування тексту

При взаємодії з користувачем неможливо обійтися без засобів введення текстової інформації, однак ***zenity*** надає і ці засоби. Якщо разом з командою використати ключ ***–entry***, то графічний елемент створить дві кнопки – **«ОК»** і **«Cancel»**. Якщо вибрати кнопку **«Cancel»**, то статусом виходу команди буде 1; в іншому випадку статус виходу дорівнюватиме 0, а введені користувачем дані виводяться у стандартний потік **STDOUT**.

У наступному прикладі представлене виведення діалогового вікна введення та редагування тексту. Його результат зображений на рис. 10.6.

$ zenity --entry --text="Enter your age"

Рис. 10.6. Діалогове вікно для введення тексту, створене командою ***zenity***

Також можна результат виконання команди ***zenity*** присвоїти певній змінній.

```
age=`zenity --entry --text="Enter your age"`
echo $age
30
```

У полі введення може міститися слово, визначене за замовчуванням (ключ ***--entry-text***).

```
age=`zenity --entry --text="Enter your age" --entry-text="Enter your age"`
```

Введені дані можна приховати, застосувавши ключ ***--hide-text***, який використовується у наступному прикладі та представляє сценарій **bash**, що виводить діалогове вікно для введення пароля кінцевим користувачем.

Файл inputPassword.sh

```
#!/bin/bash

if zenity --entry --title="Verify access rights" \
     --text=" Input your password:" \
     --entry-text="password" \
     --hide-text
then
  echo $?
else
  zenity --error --text="Password is not entered"
fi
```

Вигляд діалогового вікна введення тексту з прихованим значенням зображений на рис. 10.7.

Рис. 10.7. Діалогове вікно для введення тексту з прихованим значенням, створене командою ***zenity***

Діалогове вікно зі списком

У ***zenity*** не залишені без уваги і такі важливі елементи інтерфейсу, як списки (ключ ***--list***). Поряд зі звичайним текстовим списком є можливість створення списків з перемикачами і радіо-кнопками. Елементи для діалогового вікна, що містить список, мають вказуватися в наступному

порядку: стовпці формують рядок, далі символ нового рядка, після чого вводяться стовпці, що становлять наступний рядок і т. д. Рядки елементів списку можуть бути передані у діалогове вікно через потік стандартного введення. Новий стовпець вказують за допомогою ключа ***--column***.

Перемикачі (ключ ***--checklist***) і радіо-кнопки (***--radiolist***) можуть розташовуватися тільки в першому стовпці, тому в рядках для цих варіантів списку першим елементом обов'язково має бути рядок «TRUE» або «FALSE».

За допомогою списків з радіо-кнопками у першому стовпці можна реалізувати інтерактивне меню. Наступний приклад демонструє створення діалогового вікна зі списком, зовнішній вигляд якого представлений на рис. 10.8.

zenity --list --radiolist --title "Sys Admin Menu" --column "Select" --column "Menu Item" FALSE "Display disk space" FALSE "Display users" FALSE "Display memory usage" FALSE "Exit"

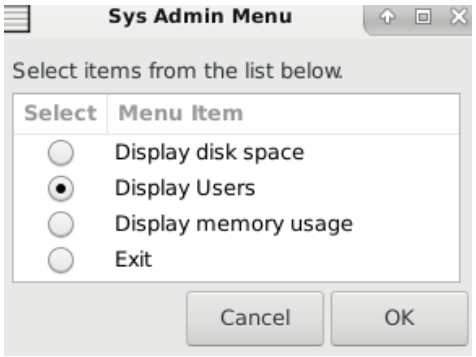

Рис. 10.8. Діалогове вікно для введення певного значення зі списку, створене командою ***zenity***

Далі для прикладу візьмемо сценарій **bash**, який виводить інтерактивне меню за допомогою діалогового вікна зі списком.

Файл menu.sh

```
#!/bin/bash

command=` zenity --list --radiolist --title "Sys Admin Menu" \
          --column "Select" --column "Menu Item" \
          FALSE "Display disk space" \
          FALSE "Display users" \
          FALSE "Display memory usage" \
          FALSE "Exit"`

if [ $? -eq 0 ]
```

```
then
     case $command in
      1)
       df ;;
      2)
       who ;;
      3)
       vmstat ;;
      4)
       exit ;;
     esac
fi
```

Діалогове вікно вибору файлу

Команда ***zenity*** дозволяє скористатися ще кількома привабливими вбудованими графічними елементами. Під час роботи з назвами файлів надзвичайно зручним є графічний елемент ***--file-selection***, який дозволяє позбутися від необхідності введення назв використовуваних файлів, оскільки з його допомогою можна перейти безпосередньо до того каталогу, де знаходиться файл, і зробити його вибір.

У наступному прикладі застосуємо команду ***zenity*** для виведення діалогового вікна вибору файлу, зовнішній вигляд якого зображений на рис. 10.9.

$ zenity --file-selection --title="Select a file" --filename="$HOME/"

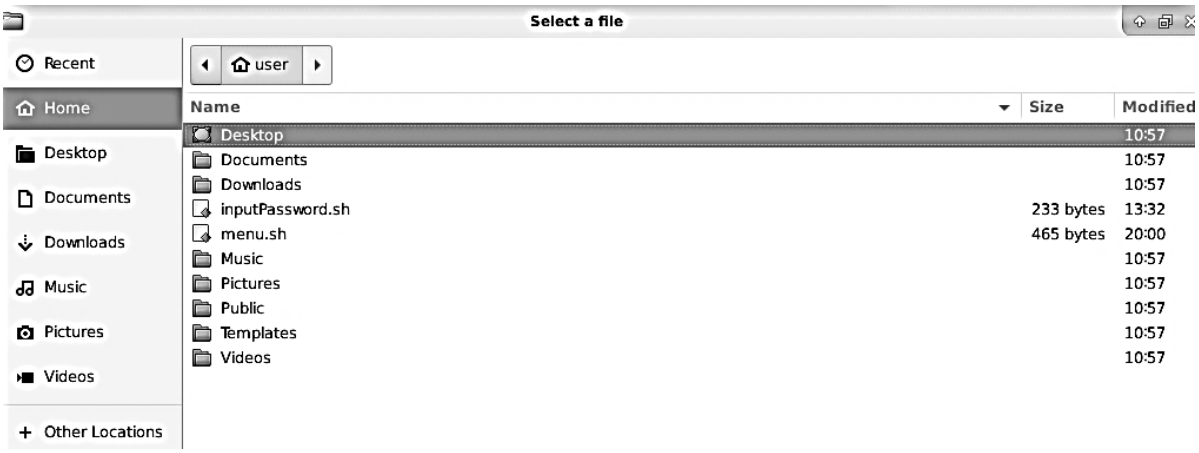

Рис. 10.9. Діалогове вікно вибору файлу, створене командою ***zenity***

Додатковий ключ ***--filename*** задає місце розташування каталогу, з якого починається перехід каталогами у вікні, або файл, який позначається вибраним за замовчуванням. Вікно графічного елемента ***--file-***

selection містить лістинг каталогів, що знаходиться ліворуч, та лістинг файлів, який представлений праворуч. Повна назва вибраного файлу в результаті виводиться до стандартного потоку виведення.

Нижче наведений приклад сценарію, який містить вікно вибору файлу, який потрібно видалити.

Файл Deletefile.sh

```
#!/bin/bash

delete_file()
{
     local f=$1
     local m="$0: file $f failed to delete"
     if [ -f $f ]
     then
          rm $FILE && m="$0: $f file deleted"
     else
          m="$0: $f is not a file"
     fi

     zenity --info --title="Remove file" --
text="$m"
}

file=`zenity --file-selection --title="Select a
file" --filename=" $HOME/"`

if [ ! -z $file ]
then
     delete_file $file
fi
```

Діалогове вікно вибору каталогу

Подібно до діалогового вікна вибору існуючого файлу у ***zenity*** передбачена опція вибору існуючого каталогу. Для створення діалогового вікна вибору файлу необхідно вказати додатковий ключ ***--directory***. Оброблюється воно так само, як і ділогове вікно вибору файлу.

Наступним прикладом буде виведення діалогового вікна вибору каталогу, зовнішній вигляд у свою чергу представлений на рис. 10.10.

$ zenity --file-selection --directory --title "Select a directory" ~

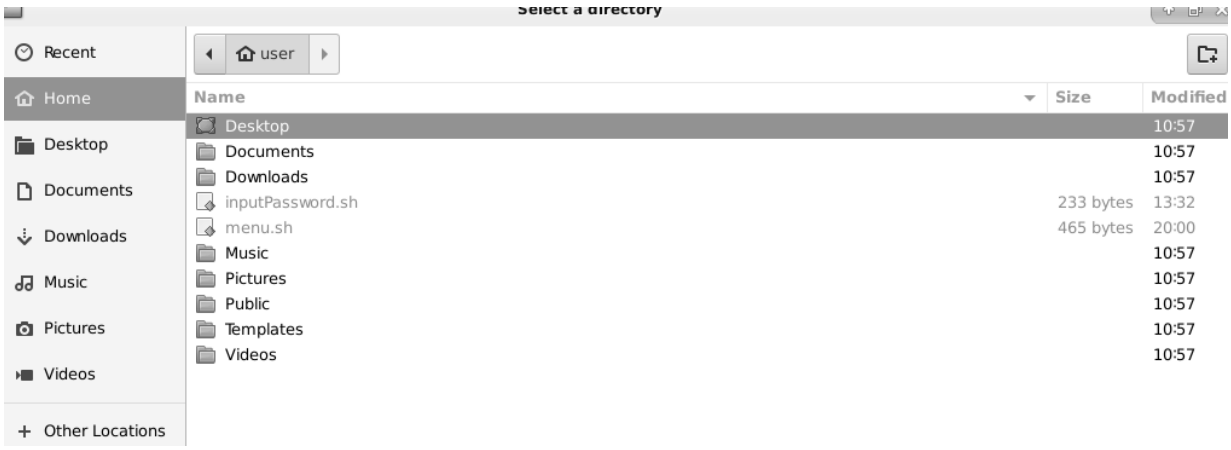

Рис. 10.10. Діалогове вікно вибору каталогу, створене командою ***zenity***

10.2.2. Програма kdialog для створення діалогових вікон для робочого столу KDE

Команда ***kdialog*** є аналогом команди ***dialog*** для KDE і встановлена у ній за замовчуванням. Передбачена велика кількість графічних елементів, які можуть бути використані у ***kdialog***. У табл. 10.1 перераховані основні опції команди ***kdialog***.

Табл. 10.1. Ключі команди *kdialog*

Параметр	Опис
--yesno <текст>	питальний діалог з кнопками так / ні
--yesnocancel <текст>	питальний діалог з кнопками так / ні / скасування
--sorry <текст>	діалог «Невдача»
--error <текст>	діалог «Помилка»
--msgbox <текст>	інформаційний діалог
--inputbox <текст> <Init>	діалог введення
--password <текст>	діалог введення пароля
--textbox <файл> [ширина] [висота]	діалог з текстом
--textinputbox <текст> <Init> [ширина] [висота]	діалог введення тексту
--combobox <текст> пункт [пункт] [пункт] ...	діалог з випадаючим списком
--menu <текст> [бірка пункт] [пункт бірка] ...	діалог з меню
--checklist <текст> [статусний_тег пункт] ...	діалог з опціями
--radiolist <текст> [статусний_тег пункт] ...	діалог з перемикачем
--getopenfilename [startDir] [фільтр]	діалог відкриття існуючого файлу

Закінчення табл. 10.1

--getsavefilename [startDir] [фільтр]	діалог збереження файлу
--getexistingdirectory [startDir]	діалог відкриття існуючого каталогу
--progressbar <текст> [totalsteps]	діалог ходу процесу
--getcolor	діалог вибору кольору
--title <текст>	тема діалогу
--default <текст>	пункт за замовчуванням у випадних списках, меню і списках кольорів
--slider <текст> [MinValue] [MaxValue] [крок]	діалог з повзунком, повертає вибране значення
--calendar <текст>	діалог з календарем, повертає обрану дату.

Діалогове вікно для виведення повідомлень

Графічний елемент ***-msgbox*** є найбільш широко застосовуваний тип діалогового вікна. Цей елемент відображає у вікні просте текстове повідомлення і чекає натиснення користувачем кнопки ОК, після чого зникає. Для використання графічного елемента ***msgbox*** має бути заданий такий формат:

kdialog –msgbox <текст> <висота> <ширина>

Як параметр, ***<текст>*** може бути заданий будь-який текстовий рядок, що буде виведений у вікні. Команда ***dialog*** забезпечує автоматичне перенесення тексту на наступний рядок, щоб не відбувався вихід за межі розмірів створеного вікна, для позначення яких служать параметри ***height*** і ***width***. Якщо з'явиться необхідність помістити назву у верхній частині вікна, то можна також скористатися параметром ***–title***, за допомогою якого задається текст назви. Нижче наведено приклад використання графічного елемента ***msgbox***.

Наступний приклад демонструє створення діалогового вікна командою ***kdialog***. Результат представлений на рис. 10.11.

```
kdialog –title "Testing" –msgbox "This is a test"
```

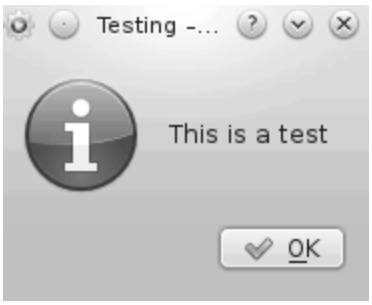

Рис. 10.11. Діалогове вікно виведення повідомлень, створене командою ***kdialog***

Діалогове вікно для вибору варіанта відповіді

Графічний елемент ***-yesno*** являє собою більш складний варіант графічного елемента ***msgbox***, який дозволяє користувачеві відповісти на питання, відображене у вікні, позитивно або негативно. Цей елемент створює в нижній частині вікна дві кнопки: **«Yes»** та **«No»**. Користувач може переходити від однієї до іншої за допомогою миші, клавіші табуляції або клавіш зі стрілками. Щоб обрати кнопку, користувач може натиснути клавішу пробілу або **<Enter>**.

Нижче наведено приклад використання графічного елемента ***–yesno***, а його зовнішній вигляд представлено на рис. 10.12.

kdialog –title "Please answer" –yesno "Is this thing on?"

Рис. 10.12. Діалогове вікно для вибору варіанта відповіді, створене командою ***kdialog***

Статус виходу команди ***kdialog*** встановлюється залежно від того, яку кнопку обрав користувач. Якщо вибрано кнопку **«No»**, статусом виходу є 1, а якщо кнопку **«Yes»**, статус виходу приймає значення 0. Наступний приклад представляє простий сценарії **bash**, що використовує команду ***kdialog*** для відображення діалогового вікна для обрання варіанта відповіді.

Файл yesno.sh

```
#!/bin/bash

kdialog –title "Вихід з поточного сеансу" \
    --yesno "Ви дійсно хочете вийти з системи?"

case "$?" in
     '0')
      logout;;
     '1')
      exit;;
     '-1')
      echo "Ви вийшли, натиснувши ESC, або ж всередині діалогу виникла помилка"
      exit 1;;
esac
```

Діалогове вікно для введення та редагування тексту

Графічний елемент ***inputbox*** створює просту область з текстовим полем, у якому користувач може ввести текстовий рядок. Команда ***kdialog*** відправляє значення введеного текстового рядка в потік **STDOUT** (на відміну від ***dialog***, що відправляє значення в потік **STDERR**). Щоб отримати відповідь, це значення необхідно перенаправити.

Елемент ***inputbox*** створює дві кнопки – **«ОК»** і «**Cancel»**. Якщо вибрано **«Cancel»**, то статусом виходу команди є 1, в іншому випадку статус виходу дорівнюватиме 0.

Покажемо на прикладі, як застосовувати команду ***kdialog*** для відображення діалогового вікна цього типу.

```
kdialog –inputbox "Enter your age:"
```

Результат наведеної вище команди представлений на рис. 10.13.

Рис. 10.13. Діалогове вікно для введення тексту, створене командою ***kdialog***

Також можна результат виконання команди ***kdialog*** присвоїти певній змінній.

```
$ age=`kdialog –inputbox "Enter your age:"`
$ echo $age
30
```

Після заголовка, що є першим параметром, можна задати другий параметр, який представляє значення за замовчуванням:

```
$ age=`kdialog –inputbox "Enter your age:" 30`
```

Зазначимо, якщо для відображення вмісту текстового файлу використовується команда ***cat***, то після виведення на екран відповідного текстового значення не формується символ позначення кінця рядка. Це дозволяє легко перенаправляти вміст файлу в змінну в сценарії командного інтерпретатора, щоб витягти рядок, введений користувачем.

Файл inputbox.sh

```
#!/bin/bash

cmdrun=`kdialog –title "Виконати команду" \
    --inputbox "Введіть команду на виконання:" 
"ls –la"`
```

```
if [ $? -eq 1 ]
 then
      $cmdrun
 else
      echo "Діалогове вікно повернуло код помилки"
fi
```

Діалогове вікно з інтерактивним меню

Створити варіант вікна з текстовим меню командою ***kdialog*** дозволяє графічний елемент ***-menu***. Для його використання достатньо лише вказати дескриптор вибору і надати текст для кожного елемента, що показано у наступному прикладі.

kdialog –menu "Sys Admin Menu" 1 "Display disk space"
2 "Display users" 3 "Display memory usage" 4 "Exit"

У результаті буде виведене діалогове вікно, що зображене на рис. 10.14.

Рис. 10.14. Діалогове вікно з меню, створене командою ***kdialog***

Перший параметр визначає назву меню, далі необхідно додати пари, що визначають елементи меню. Першим елементом пари є дескриптор, призначений для вибору елемента меню. Кожен дескриптор, що визначає елемент меню, повинен бути унікальним, його треба зробити шляхом натискання відповідної клавіші.

Як і у випадку використання команди ***zenity*** покажемо, як можна застосовувати графічне діалогове вікно ***–menu*** команди ***kdialog*** у сценарії **bash**.

Файл menu.sh

```
#!/bin/bash

command=`kdialog –menu "Sys Admin Menu" 1 "Display disk space" 2 "Display users" 3 "Display memory usage" 4 "Exit"`
```

```
if [ $? -eq 0 ]
then
     case $command in
      1)
       df ;;
      2)
       who ;;
      3)
       vmstat ;;
      4)
       exit ;;
     esac
fi
```

Діалогове вікно вибору файлу

Команда ***kdialog*** дозволяє скористатися ще декількома привабливими вбудованими графічними елементами. Під час роботи з назвами файлів надзвичайно зручним є графічний елемент ***-getopenfilename***, який дозволяє позбутися необхідності введення назв використовуваних файлів, оскільки з його допомогою можна перейти безпосередньо до того каталогу, де знаходиться файл, і зробити вибір.

Графічний елемент ***fselect*** має приблизно такий формат виклику:

kdialog -title "Select a file" -getopenfilename $HOME/

Результат виконання наведеної вище команди представлений на рис. 10.15.

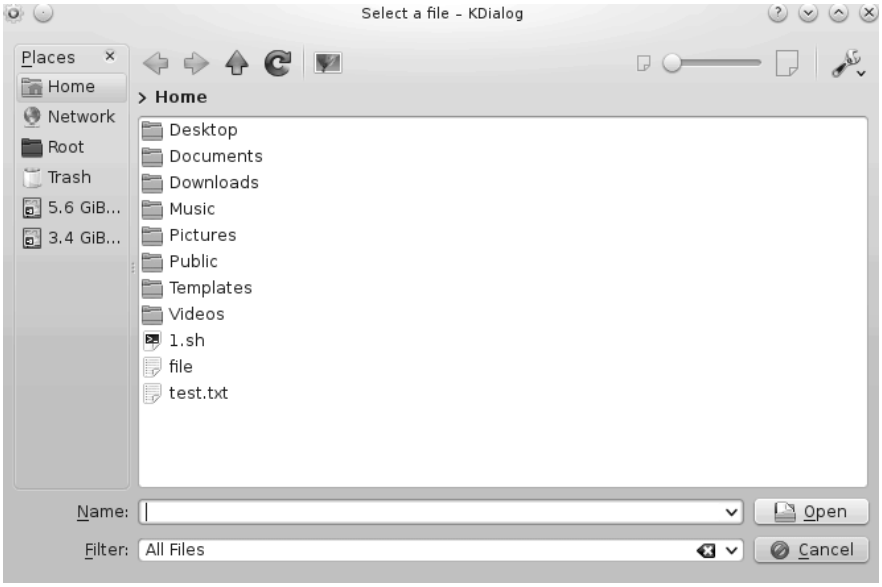

Рис. 10.15. Діалогове вікно вибору файлу, створене командою ***kdialog***

Перший параметр після опції ***-getopenfilename*** задає місце розташування каталогу, з якого починається перехід каталогами у вікні. Вікно графічного елемента ***-getopenfilename*** містить лістинг каталогів, що знаходиться ліворуч; лістинг файлів, який представлений праворуч; і просте текстове поле з назвою обраного на цей час файлу або каталогу. Передбачена можливість вручну ввести назву файлу в цьому текстовому полі або використовувати для вибору файлу лістинги каталогів і файлів.

Нижче наведений приклад сценарію, що містить вікно вибору файлу, який потрібно видалити.

Файл Deletefile.sh

```
#!/bin/bash

delete_file()
{
     local f=$1
     local m="$0: file $f failed to delete"
     if [ -f $f ]
     then
          rm $FILE && m="$0: $f file deleted"
     else
          m="$0: $f is not a file"
     fi
     kdialog –title "Remove file" –msgbox "$m"
}

   file=`kdialog –title "Select a file" –
getopenfilename $HOME/`

    if [ ! –z $file ] then
       delete_file $file
```

Діалогове вікно вибору каталогу

Подібно до діалогового вікна вибору існуючого файлу в ***kdialog*** передбачена опція вибору існуючого каталогу (***--getexistingdirectory***). Воно оброблюється так само, як і ділогове вікно вибору файлу.

$ kdialog –title «Select a file» -- getexistingdirectory ~

На рис. 10.16 відображений зовнішній вигляд діалогового вікна, що було виведене у результаті виконання представленої вище команди.

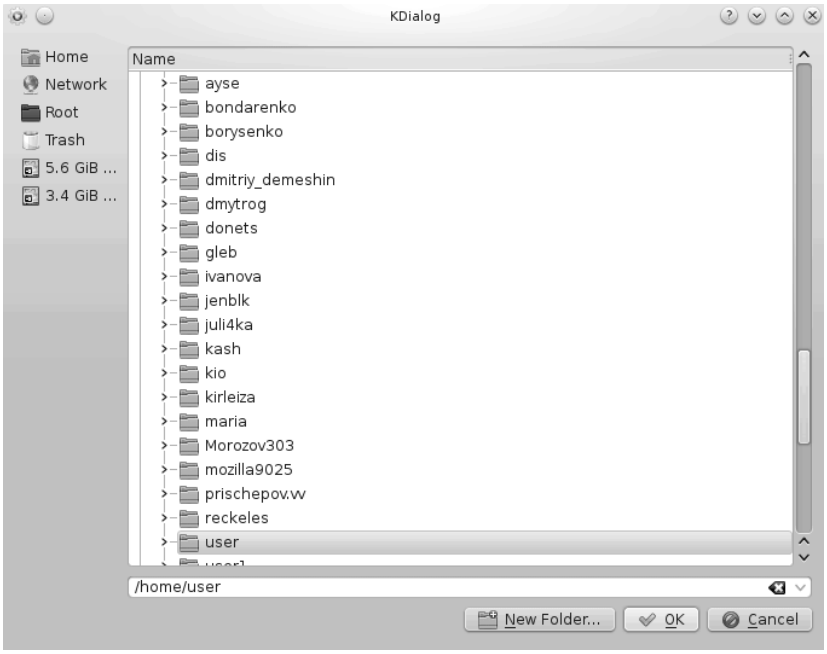

Рис. 10.16. Діалогове вікно вибору каталогу, створене командою ***kdialog***

РОЗДІЛ 11.
КОМПІЛЯТОРИ ТА ІНТЕРПРЕТАТОРИ В ОС НА БАЗІ LINUX

11.1 Компілятори

11.1.1 Введення до компілятора GCC

Серед засобів, які традиційно використовуються для написання програм для відкритих операційних систем, досить популярними є інструменти розробника **GNU**. Власне проект **GNU** був заснований Річардом Столлманом у 1984 р. Його задумом було створення комплекту програмного забезпечення під єдиною ліцензією, яка не допускала б можливості привласнення будь-ким ексклюзивних прав на нього. Однією з частин комплекту і є набір інструментів для розробника, у який зокрема входить компілятор GCC, який розглядатиметься далі.

Спочатку абревіатура **GCC** розшифровувалась як **GNU C Compiler**, оскільки спочатку компілятор підтримував тільки мову **C**, однак сьогодні вона означає – **GNU Compiler Collection**, тобто колекцію компіляторів.

Створимо першу програму за допомогою компілятора **GCC**, яка виводитиме на консоль текст ***«Hello world!»***. При цьому файли з вихідними кодами програм будуть звичайними текстовими файлами, які можна створювати та редагувати за допомогою будь-якого текстового редактора, який встановлений у системі (наприклад, ***vi***, ***nano*** або інші для консольного режиму Linux або ***GEdit***, ***KWrite*** та інші для графічних оболонок.

Для написання програм на мові **C** сформуємо окремий каталог під назвою ***programs***, у якому створимо файл з простим програмним кодом та назвемо його ***hello.c***. Його вміст наведено далі.

Файл hello.c

```
#include <stdio.h>

int main(void)
{
      printf("Hello world!\n");
      return 0;
}
```

Тепер подивимося чи був він створений.

```
debianuser@debianServ2017:~/programs$ ls -l
total 4
-rw-r--r-- 1 debianuser debianuser 75 Jan  8 15:38 hello.c
```

Як бачимо, відповідний файл був створений. Тепер спробуємо скомпілювати його за допомогою команди ***gcc*** та його назву у якості аргумента.

```
debianuser@debianServ2017:~/programs$ gcc hello.c
debianuser@debianServ2017:~/programs$ ls -l
total 16
-rwxr-xr-x 1 debianuser debianuser 8640 Jan  8 15:40 a.out
-rw-r--r-- 1 debianuser debianuser   75 Jan  8 15:38 hello.c
```

У результаті виконання команди ***gcc*** можна помітити, що у каталозі з'явився новий файл, що має назву ***a.out***. Він і є виконуваною програмою, що була отримана в результаті компіляції, тепер запустимо його.

```
debianuser@debianServ2017:~/programs$ ./a.out
Hello world!
```

Виконуваний файл дійсно запустився, і на екран був виведений відповідний текст ***«Hello world!»***.

За замовчуванням компілятор ***gcc*** присвоює назву ***a.out*** усім створеним виконуваним файлам, однак це не завжди зручно. Для того, щоб назвати виконуваний файл інакше, потрібно до команди ***gcc*** додати ключ ***-o*** та назву, яку ми хочемо дати майбутньому виконуваному файлу. Продемонструємо це у наступному прикладі, отримавши у результаті компіляції виконуваний файл ***hello***.

```
debianuser@debianServ2017:~/programs$ gcc hello.c -o hello
debianuser@debianServ2017:~/programs$ ls -l
total 28
-rwxr-xr-x 1 debianuser debianuser 8640 Jan  8 15:40 a.out
-rwxr-xr-x 1 debianuser debianuser 8640 Jan  8 15:46 hello
-rw-r--r-- 1 debianuser debianuser   75 Jan  8 15:38 hello.c
debianuser@debianServ2017:~/programs$ ./hello
Hello world!
```

Таким чином, був отриманий такий самий виконуваний файл, який має більш зручну назву для кінцевого користувача. Зазначимо, що ключ ***-o*** є тільки одним з багатьох ключів програми ***gcc***, про які можливо дізнатися за допомогою довідкової системи ***man***.

Далі детальніше розглянемо дії, які виконує програма ***gcc***. Робота цього компілятора складається з трьох різних етапів: препроцесорна обробка, компіляція та компоновка (лінковка).

Препроцесор включає в основний файл вміст усіх заголовних файлів, які зазначені в директивах ***#include*** та мають розширення ***.h***. У подібних файлах зазвичай містяться оголошення функцій, що використовуються у програмі, однак не визначені у її тексті. Визначення цих функцій знаходяться десь в іншому місці: або в інших файлах з вихідним кодом, або у бінарних бібліотеках.

Другою стадією є власне компіляція, що полягає в перетворенні тексту програми на мові **C** або **C++** у набір машинних команд. При цьому

результат буде зберігатися в об’єктному файлі. На комп’ютерах, що мають різну архітектуру процесора, виконувані файли представлені у зовсім різних форматах. Зокрема, на одному комп’ютері неможливо запустити бінарний файл, який був зібраний на іншому, тільки якщо вони мають неоднакові архітектури процесора та операційні системи. Саме тому програми для UNIX-подібних систем поширюються у вигляді вихідних кодів: вони мають бути доступні всім користувачам, незалежно від типу процесора та операційної системи.

Останньою стадією є компоновка, що полягає у зв’язуванні всіх об’єктних файлів проекту в один, викликів функцій з їх визначеннями, а також приєднанням файлів бібліотек, що містять функції, які викликаються, однак не визначені в проекті. У якості результату буде сформований виконуваний файл. Якщо певна функція використовується у програмі, але компонувальник не може знайти місце її визначення, буде виведено повідомлення про помилку, а виконуваний файл не буде створений.

Тепер розглянемо вищезазначене на прикладі. Для цього створимо програму, що буде являти собою простий калькулятор, який може виконувати арифметичні дії: додавання, віднімання, множення та ділення. При цьому програма-калькулятор буде запитувати спочатку 2 числа, а потім дію, яку потрібно виконати над ними. Після цього програма виведе кінцевий результат, а потім зупиниться, повернувши кінцевого користувача до командного рядка.

Створимо у каталозі ***programs*** новий дочірній каталог ***calc***, у ньому створимо файл ***calc.c***.

Файл calc.c

```
#include <stdio.h>

int main(void)
{
 float a;
 float b;
 char oper;
 printf("Введіть перше число: ");
 scanf("%f",&a);
 printf("Введіть друге число: ");
 scanf("%f",&b);
 printf("Зазначте арифметичну дію (+,-,* або /): ");

 while ((oper = getchar()) != EOF)
 {
     if (oper == '+')
     {
```

```
            printf("%6.2f\n",a + b);
            break;
        }
        else if (oper == '-')
        {
            printf("%6.2f\n",a - b);
            break;
        }
        else if (oper == '*')
        {
            printf("%6.2f\n",a * b);
            break;
        }
        else if (oper == '/')
        {
            if (b == 0)
            {
                printf("Помилка: ділення на нуль!\n");
                break;
            }
            else
            {
                printf("%6.2f\n",a / b);
                break;
            }
        }
    }

    return 0;
}
```

Найпершим, як зазначено вище, виконується препроцессінг. Для цього потрібно запустити компілятор ***gcc*** з ключем ***-E***, який зупиняє виконання програми на етапі обробки препроцесором. Результатом такої дії стає отримання вихідного файлу, в який включений вміст заголовків файлів. У цьому випадку був включений тільки один заголовок – файл ***stdio.h***, що представляє колекцію стандартних функцій введення-виведення.

Розглянемо на прикладі виконання команди ***gcc*** з ключем ***-E***:

```
debianuser@debianServ2017:~/programs/calc$ gcc -E calc.c -o calc.cpp
debianuser@debianServ2017:~/programs/calc$ ls -l
total 24
-rw-r--r-- 1 debianuser debianuser   841 Jan  8 16:06 calc.c
-rw-r--r-- 1 debianuser debianuser 17988 Jan  8 16:07 calc.cpp
```

У результаті обробки препроцесором був отриманий вихідний файл, який ми назвали ***calc.cpp***, зазначивши перед ним ключ ***-o***. Якщо його

відкрити, можна побачити, що він є досить довгим. Це пояснюється тим, що в нього увійшов увесь код файлу заголовків ***stdio.h***. До того ж, препроцесор додав у файл деякі теги, які вказують компілятору спосіб зв'язку з оголошеними функціями. Скріншот початку файлу ***calc.cpp*** у відкритому текстовому редакторі **nano** представлений на рис. 11.1.

Рис. 11.1. Заголовки та теги на початку файлу ***calc.cpp***

Основний текст програми знаходиться тільки в самому низу файлу, в цьому можна переконатися з рис. 11.2.

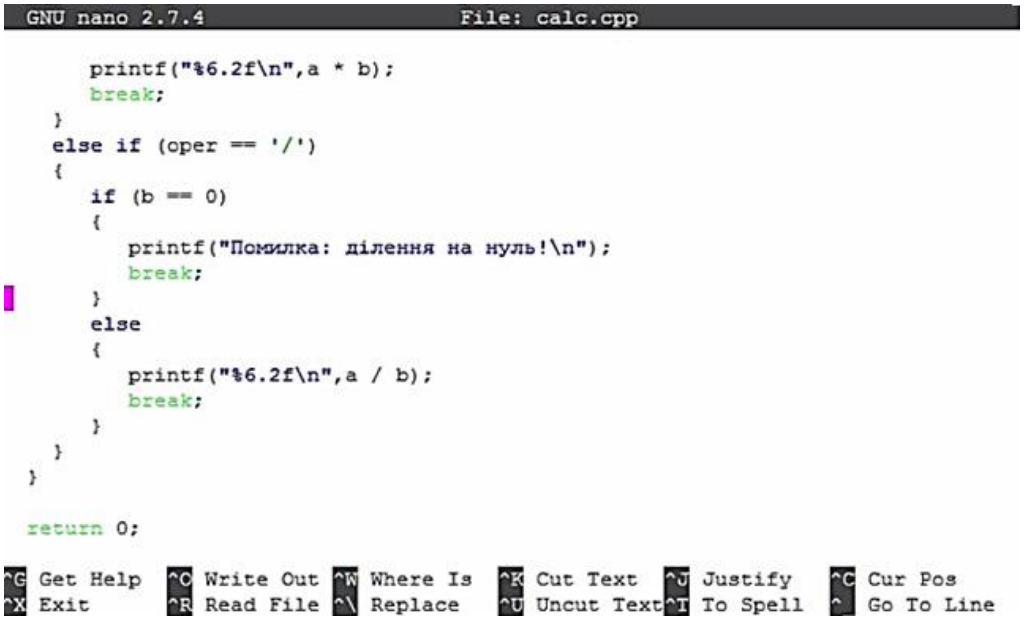

Рис. 11.2. Код основної програми у кінці файлу ***calc.cpp***

Тепер перейдемо до наступного етапу, основою якого є створення об'єктного файлу. Для його формування призначений ключ ***-c***. Покажемо створення об'єктного файлу на прикладі.

```
debianuser@debianServ2017:~/programs/calc$ gcc -c calc.c
debianuser@debianServ2017:~/programs/calc$ ls -l
total 28
-rw-r--r-- 1 debianuser debianuser   841 Jan  8 16:06 calc.c
-rw-r--r-- 1 debianuser debianuser 17988 Jan  8 16:07 calc.cpp
-rw-r--r-- 1 debianuser debianuser  2632 Jan  8 16:25 calc.o
```

Розглянувши вміст каталогу програми ***calc***, можна помітити, що у ньому з'явився новий файл ***calc.o***, який і є об'єктним та був отриманий після виконання команди ***gcc*** з ключем ***-c***. Об'єктний файл має таку саму назву, що й файл з програмним кодом, тільки з іншим розширенням. Таким чином, об'єктний файл за замовчуванням не змінює назву, хоча опціонально можна задати й іншу, відповідно вказавши її після ключа ***-o***.

Разом з тим під час створення об'єктного файлу з файлу, який вже був оброблений препроцесором (у нашому випадку це був файл ***calc.cpp***), потрібно вказати те, що компільований файл є файлом вихідного коду, оброблений препроцесором, і має теги препроцесора. В іншому випадку він буде оброблюватися як звичайний файл **C++**, без урахування тегів препроцесора, це буде означати, що зв'язок з оголошеними функціями не буде встановлено. Для явного вказування на мову та формат оброблюваного файлу в команді ***gcc*** призначений ключ ***-x***, а файл **C++**, що був оброблений препроцесором, треба позначити як ***cpp-output***.

У наступному прикладі ми отримуємо альтернативий об'єктний файл з файлу, що був попередньо оброблений препроцесором.

```
debianuser@debianServ2017:~/programs/calc$ gcc -x cpp-output -c calc.cpp -o calc
1.o
debianuser@debianServ2017:~/programs/calc$ ls -l
total 32
-rw-r--r-- 1 debianuser debianuser  2632 Jan  8 16:26 calc1.o
-rw-r--r-- 1 debianuser debianuser   841 Jan  8 16:06 calc.c
-rw-r--r-- 1 debianuser debianuser 17988 Jan  8 16:07 calc.cpp
-rw-r--r-- 1 debianuser debianuser  2632 Jan  8 16:25 calc.o
```

Тепер перейдемо до останнього етапу, що представляє компоновку, та отримаємо з об'єктного файлу кінцевий виконуваний файл, що й представлено у наступному прикладі.

```
debianuser@debianServ2017:~/programs/calc$ gcc calc.o -o calc
debianuser@debianServ2017:~/programs/calc$ ./calc
Введіть перше число: 6
Введіть друге число: 9
Зазначте арифметичну дію (+,-,* або /): +
 15.00
debianuser@debianServ2017:~/programs/calc$ ./calc
Введіть перше число: 7
Введіть друге число: 0
Зазначте арифметичну дію (+,-,* або /): /
Помилка: ділення на нуль!
```

Таким чином, був створений та запущений виконуваний файл. Усі етапи його створення представляють досить нескладні дії, тільки якщо програма складається з одного файлу. Насправді у реальності програми, що складаються тільки з одного файлу, бувають доволі рідко. Зазвичай вони складаються більш ніж з одного файлу та об'єднані в проект.

Але можуть бути випадки, коли різні частини однієї програми написані різними мовами, тоді доводиться запускати компілятори різних мов, щоб кожен отримав об'єктний файл зі свого вихідного файлу, а потім компонувати їх у виконувану програму.

11.1.2. Компіляція програми, що складається з декількох файлів

Тепер спробуємо створити проект, що складається більше ніж з одного файлу. Удосконалимо програму-калькулятор, додавши до неї додаткові операції, такі як підведення до ступеня, обчислення квадратного кореня та обчислення синуса, косинуса, тангенса. Останні операції є унарними, тобто виконують дію тільки над одним числом. Тому в новій версії калькулятора після введення першого числа відразу буде запит дії. Якщо зазначена користувачем дія оперує з двома числами, то додатково буде запит введення другого числа.

Для проекту створимо новий каталог та назвемо його ***calc2***. Програма буде складатися з трьох файлів: файлу заголовків ***calc.h***, файлу з основним кодом ***calc.c*** та головного файлу, що містить функцію ***Main***, яка є точкою входження у програму, ***main.c***. Наведемо вміст кожного з них.

Файл calculate.h

```
#ifndef CALC_H_
#define CALC_H_

float Calc(float num, char oper[4]);

#endif /*CALC_H_*/
```

Файл calculate.c

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calc.h"

float Calc(float num, char oper[4])
{
 float num2;
```

```
if(strncmp(oper, "+", 1) == 0)
{
    printf("Другий доданок: ");
    scanf("%f",&num2);
    return(num + num2);
}
else if(strncmp(oper, "-", 1) == 0)
{
    printf("Від'ємник: ");
    scanf("%f",&num2);
    return(num - num2);
}
else if(strncmp(oper, "*", 1) == 0)
{
    printf("Добуток: ");
    scanf("%f",&num2);
    return(num * num2);
}
else if(strncmp(oper, "/", 1) == 0)
{
    printf("Дільник: ");
    scanf("%f",&num2);
    if(num2 == 0)
    {
        printf("Помилка: Ділення на нуль! ");
        return(HUGE_VAL);

    }
    else
        return(num / num2);
}

else if(strncmp(oper, "pow", 3) == 0)
{
    printf("Ступінь: ");
    scanf("%f",&num2);
    return(pow(num, num2));
}
else if(strncmp(oper, "sqrt", 4) == 0)
    return(sqrt(num));
   else if(strncmp(oper, "sin", 3) == 0)
    return(sin(num));
else if(strncmp(oper, "cos", 3) == 0)
```

```
        return(cos(num));
    else if(strncmp(oper, "tan", 3) == 0)
        return(tan(num));
    else
    {
        printf("Неправильно введена дія ");
        return(HUGE_VAL);
    }
}
```

Файл main.c:

```
#include <stdio.h>
#include "calc.h"

int main(void)
{
    float num;
    char oper[4];
    float res;
    printf("Введіть число: ");
    scanf("%f",&num);
    printf("Зазначте арифметичну дію (+,-
,*,/,pow,sqrt,sin,cos або tan): ");
    scanf("%s",&oper);
    res = Calc(num, oper);
    printf("%6.2f\n",res);
    return 0;
}
```

Тепер скомпілюємо кожен з файлів окремо, отримавши відповідні об'єктні файли для них, але для спрощення пропускаємо етап обробки препроцесором.

```
debianuser@debianServ2017:~/programs/calc2$ gcc -c calc.c
debianuser@debianServ2017:~/programs/calc2$ gcc -c main.c
debianuser@debianServ2017:~/programs/calc2$ ls -l
total 20
-rw-r--r-- 1 debianuser debianuser 1099 Jan  8 17:03 calc.c
-rw-r--r-- 1 debianuser debianuser   90 Jan  8 16:38 calc.h
-rw-r--r-- 1 debianuser debianuser 3696 Jan  8 17:10 calc.o
-rw-r--r-- 1 debianuser debianuser  323 Jan  8 17:09 main.c
-rw-r--r-- 1 debianuser debianuser 2072 Jan  8 17:11 main.o
```

Таким чином, були отримані 2 об'єктні файли: ***calc.o*** та ***main.o*** відповідно. Тепер з них зробимо один кінцевий виконуваний файл.

```
debianuser@debianServ2017:~/programs/calc2$ gcc calc.o main.o -o calc
calc.o: In function `Calc':
calc.c:(.text+0x1d6): undefined reference to `pow'
calc.c:(.text+0x205): undefined reference to `sqrt'
calc.c:(.text+0x234): undefined reference to `sin'
calc.c:(.text+0x260): undefined reference to `cos'
calc.c:(.text+0x28c): undefined reference to `tan'
collect2: error: ld returned 1 exit status
```

Однак як бачимо з прикладу, під час спроби створення виконуваного файлу було виведено повідомлення про помилку. Повідомлення «**Undefined reference**» у цьому разі означає посилання на невизначену функцію, тобто компілятор ***gcc*** не зміг знайти визначення функцій, що виконують математичні операції, які були додані нами у новій версії калькулятора.

Як відомо, визначення різних функцій можуть знаходитись у бібліотеках, що відображають скомпільовані виконувані файли, у яких містяться колекції однотипних операцій, які часто викликаються у багатьох програмах. Саме тому немає жодного сенсу багаторазово писати їх код у різних програмах. У системах Linux стандартними розташуваннями файлів бібліотек є ***/usr/lib*** та ***/usr/local/lib***. Такі файли можуть мати розширення ***.a*** у випадку статичних бібліотек або ***.so*** у випадку динамічних бібліотек.

Разом з тим у виконуваний файл поміщується тільки посилання на файл бібліотеки, а вже з нього запускається відповідна функція. Під час компоновки будь-якої програми компілятор ***gcc*** за замовчуванням включає у виконуваний файл бібліотеку **libc**, що є стандартною бібліотекою мови С. Усі інші потрібно включати явно, оскільки не можна в програмі розмістити абсолютно всі бібліотеки, тому що виконуваний файл матиме досить великий розмір. Отже відповідні бібліотеки потрібно вказувати в разі потреби.

У цьому випадку нам потрібна бібліотека, яка містила б всі основні математичні функції. Такою бібліотекою є **libm**, вона вимагає включити у текст програми файл заголовків ***math.h***.

Дистрибутиви Linux мають багато інших бібліотек, основні з яких представлені у табл. 11.1.

Табл. 11.1. Основні бібліотеки для мови С у дистрибутивах Linux

Бібліотека	Опис	Заголовочний файл
libGL	виведення тривимірної графіки в стандарті OpenGL	***<GL/gl.h>***
libcrypt	криптографічні функції	***<crypt.h>***
libcurses	псевдографіка в символьному режимі	***<curses.h>***
libform	створення екранних форм в текстовому режимі	***<form.h>***
libgthread	підтримка багатопотокового режиму	***<glib.h>***
libgtk	графічна бібліотека в режимі X Window	***<gtk/gtk.h>***

Закінчення табл. 11.1

libjpeg	робота з зображенням у форматі JPEG	***<jpeglib.h>***
libpng	робота з графікою у форматі PNG	***<png.h>***
libpthread	багатопотокова бібліотека POSIX	***<pthread.h>***
libreadline	робота з командним рядком	***<readline/ readline.h>***
libtiff	робота з графікою у форматі TIFF	***<tiffio.h>***.

Назви всіх цих бібліотек починаються з буквосполучення **lib-**. Для включення їх у виконуваний файл потрібно додати до команди ***gcc*** ключ ***-l***, та назву бібліотеки без префіксу **lib**.

Отже, для того, щоб включити потрібну нам бібліотеку **libm**, необхідно запустити команду ***gcc*** разом з ключем ***-lm***. Продемонструємо це у наступному прикладі.

```
debianuser@debianServ2017:~/programs/calc2$ gcc calc.o main.o -o calc -lm
debianuser@debianServ2017:~/programs/calc2$ ls -l
total 36
-rwxr-xr-x 1 debianuser debianuser 13152 Jan  8 17:21 calc
-rw-r--r-- 1 debianuser debianuser  1094 Jan  8 17:18 calc.c
-rw-r--r-- 1 debianuser debianuser    90 Jan  8 16:38 calc.h
-rw-r--r-- 1 debianuser debianuser  3688 Jan  8 17:19 calc.o
-rw-r--r-- 1 debianuser debianuser   323 Jan  8 17:09 main.c
-rw-r--r-- 1 debianuser debianuser  2072 Jan  8 17:20 main.o
```

Тепер під час створення виконуваного файлу помилки не виявлено, отже ми можемо його запустити.

```
debianuser@debianServ2017:~/programs/calc2$ ./calc
Введіть число: 8
Зазначте арифметичну дію (+,-,*,/,pow,sqrt,sin,cos або tan): +
Другий доданок: 3
 11.00
```

Компілятор ***gcc*** є досить потужним, однак має недолік, який полягає у його неспроможності компіляції програм, написаних на мові C++. Для цього існує інший компілятор, який буде розглянуто далі.

11.1.3. Компілятор G++

Давайте знову змінимо нашу програму, написавши її на цей раз уже на мові C++. А саме використаємо парадигму об’єктно-орієнтованого програмування, оформивши наш калькулятор у вигляді класу. Для проекту створимо новий каталог з назвою ***calc_cpp***, а програма аналогічно до минулого прикладу матиме три файли з назвами ***calculator.h***, ***calculator.cpp***, ***main.cpp***. Наведемо вміст кожного з них.

Файл calculator.h

```
#ifndef CALC_H_
#define CALC_H_

#include <string>
```

```
using namespace std;

class Calculator
{
 private:
      float num;
      float num2;
      string oper;
      float res;
      string err;
      bool Calculate();

 public:
      void SetValues();
      void Solve();
};

#endif /*CALC_H_*/
```

Файл calculator.cpp

```
#include <iostream>
#include <cmath>
#include "calculator.h"

using namespace std;

void Calculator::SetValues()
{
 cout << "Введіть число: ";
 cin >> num;
 cout << "Зазначте арифметичну дію (+,-
,*,/,pow,sqrt,sin,cos або tan): ";
 cin >> oper;
}

bool Calculator::Calculate()
{
 if(oper == "+")
 {
      cout << "Другий доданок: ";
      cin >> num2;
      res = num + num2;
      return true;
```

```
}
else if(oper == "-")
{
    cout << "Від'ємник: ";
    cin >> num2;
    res = num - num2;
    return true;
}
else if(oper == "*")
{
    cout << "Добуток: ";
  cin >> num2;
    res = num * num2;
    return true;
}
else if(oper == "/")
{
    cout << "Дільник: ";
    cin >> num2;
    if(num2 == 0)
    {
        err = "Помилка: ділення на нуль";
        return false;
    }
    else
    {
        res = num / num2;
        return true;
    }
}
else if(oper == "pow")
{
    cout << "Ступінь: ";
    cin >> num2;
    res = pow(num,num2);
    return true;
}
else if(oper == "sqrt")
{
    res = sqrt(num);
    return true;
}
else if(oper == "sin")
```

```
 {
      res = sin(num);
      return true;
 }
 else if(oper == "cos")
 {
      res = cos(num);
      return true;
 }
 else if(oper == "tan")
 {
      res = tan(num);
      return true;
 }
 else
 {
      err = "Помилка введення дії";
      return false;
 }
}

void Calculator::Solve()
{
 if(Calculate() == true)
      cout << res << "\n";
 else
      cout << err << "\n";
}
```

Файл main.cpp

```
#include <iostream>
#include "calculator.h"

using namespace std;

int main(void)
{
  Calculator *calc;
  calc = new Calculator;
  calc->SetValues();
  calc->Solve();
  delete calc;
  return(0);
}
```

Для компіляції програм на мові C++ у наборі GNU існує інший компілятор, що має назву ***g++***. Його відмінність від ***gcc*** полягає у тому, що у ньому за замовчуванням підключається не стандартна бібліотека мови C, а бібліотека мови C++. Однак у цьому разі всі його ключі повністю співпадатимуть з ключами компілятора ***gcc***. У наступному прикладі створимо виконуваний файл самої нової версії калькулятора, застосувавши компілятор ***g++***.

```
debianuser@debianServ2017:~/programs/calc_cpp$ g++ calculator.cpp main.cpp -o ca
lculator
debianuser@debianServ2017:~/programs/calc_cpp$ ls -l
total 28
-rwxr-xr-x 1 debianuser debianuser 15216 Jan  9 13:16 calculator
-rw-r--r-- 1 debianuser debianuser  1667 Jan  9 13:15 calculator.cpp
-rw-r--r-- 1 debianuser debianuser   243 Jan  9 12:48 calculator.h
-rw-r--r-- 1 debianuser debianuser   197 Jan  9 13:08 main.cpp
```

Тепер запустимо отриманий виконуваний файл.

```
debianuser@debianServ2017:~/programs/calc_cpp$ ./calculator
Введіть число: 5
Зазначте арифметичну дію (+,-,*,/,pow,sqrt,sin,cos або tan): *
Добуток: 3
15
```

З наведеного вище можна зауважити, що ми не підключали ніяких додаткових бібліотек, і це означає, що математичні функції входять у стандартну бібліотеку C++. Це без сумніву є великою перевагою компілятора ***g++*** порівняно з ***gcc***.

11.1.4. Компіляція програм за допомогою утиліти make

У системах Linux зазвичай присутня утиліта ***make***, що призначена для автоматизації перетворення файлів з однієї форми в іншу. Правила перетворення задаються у спеціальному файлі, який обов'язково повинен мати назву ***Makefile*** та знаходитися у корені робочого каталогу проекту. Сам файл має містити набір правил, які описуються:

1) цілями, які означають, що виконує це правило;

2) реквізитами, які вказують на те, що необхідно для виконання правила і отримання цілей;

3) командами, які виконують дані перетворення.

У загальному вигляді синтаксис файлу Makefile є наступним:

<Цілі>: <реквізити>
<Команда # 1>
...
<Команда #n>

У якості найпростішого прикладу створимо ***Makefile*** для самої першої написаної нами програми, що має назву ***Hello***. Його вміст є наступним.

Файл Makefile

```
hello: hello.c
	gcc hello.c -o hello
```

Файл складається тільки з одного правила, яке натомість складається з мети – «***hello***», реквізиту – «***hello.c***», і команди – «***gcc -o hello main.c***». Тепер для компіляції програми цілком достатньо виконати команду ***make***, знаходячись у каталозі проекту. За замовчуванням команда намагатиметься виконати найперше правило, якщо ціль виконання була чітко вказана при виклику.

```
debianuser@debianServ2017:~/programs$ make hello
gcc hello.c -o hello
debianuser@debianServ2017:~/programs$ ls -l
total 32
drwxr-xr-x 2 debianuser debianuser 4096 Jan  8 16:32 calc
drwxr-xr-x 2 debianuser debianuser 4096 Jan  8 17:21 calc2
drwxr-xr-x 2 debianuser debianuser 4096 Jan  9 13:16 calc_cpp
-rwxr-xr-x 1 debianuser debianuser 8640 Jan  9 13:52 hello
-rw-r--r-- 1 debianuser debianuser   75 Jan  8 15:38 hello.c
-rw-r--r-- 1 debianuser debianuser   37 Jan  9 13:52 Makefile
debianuser@debianServ2017:~/programs$ ./hello
Hello world!
```

Таким чином, був отриманий виконуваний файл ***hello*** за допомогою утиліти ***make***. Так само можна скомпілювати проект, що містить більше одного файлу. Для прикладу скомпілюємо першу версію нашої програми-компілятора, ***Makefile*** якого буде мати наступний вигляд.

Файл Makefile

```
calc: main.c calc.c
	gcc -o calc main.c calc.c -lm
```

Тепер застосуємо утиліту ***make***.

```
debianuser@debianServ2017:~/programs/calc2$ make calc
gcc -o calc main.c calc.c -lm
debianuser@debianServ2017:~/programs/calc2$ ls -l
total 32
-rwxr-xr-x 1 debianuser debianuser 13152 Jan  9 14:05 calc
-rw-r--r-- 1 debianuser debianuser  1094 Jan  8 17:18 calc.c
-rw-r--r-- 1 debianuser debianuser    90 Jan  8 16:38 calc.h
-rw-r--r-- 1 debianuser debianuser   323 Jan  8 17:09 main.c
-rw-r--r-- 1 debianuser debianuser    51 Jan  9 14:04 Makefile
```

Представлений ***Makefile*** загалом є працездатним, однак дійсно має один істотний недолік. У чому він полягає, детально буде розглянуто далі.

Спочатку представимо, що наша програма складається з досить великої кількості вихідних файлів. Наприклад, у певний момент часу ми внесли зміни в один з них, і тепер хочемо перекомпілювати програму. Використання підходу, описаного у останньому прикладі, призведе до того, що всі вихідні файли будуть знову скомпільовані, навіть ті у які ми не вносили змін, а це без сумніву негативно позначиться на часі

перекомпіляції. Рішення, яке може реально допомогти, є розділення компіляції на два етапи: трансляцію та компоновку.

Під час використання описаного вище підходу, змінивши один певний вихідний файл, достатньо зробити його трансляцію і компоновку всіх об'єктних файлів. При цьому буде пропущений етап трансляції реквізитів, у яких зміни не відбулися, що загалом істотно скорочує час компіляції. Такий підхід називається **інкрементною компіляцією**. Для її підтримки програма ***make*** зіставляє час зміни цілей і їх реквізитів, використовуючи дані файлової системи, завдяки чому самостійно вирішує, які правила варто виконати, а які проігнорувати.

Використаємо інкрементну компіляцію для прикладу отримання виконуваного файлу останньої версії калькулятора, написаної на мові C++. **Makefile** для неї матиме наступний вміст.

Файл Makefile

```
calculator: calculator.o main.o
	g++ calculator.o main.o -o calculator
calculator.o: calculator.cpp calculator.h
	g++ -c calculator.cpp
main.o: main.cpp calculator.h
	g++ -c main.cpp
```

Після запуску програма ***make*** спробує відразу отримати ціль ***calculator***, однак для її досягнення необхідні файли ***main.o*** і ***calculator.o***, яких поки ще не існує. Тому виконання правила буде відкладено і ***make*** почне шукати правила, що описують отримання відсутніх реквізитів. Як тільки всі реквізити будуть отримані, ***make*** повернеться до виконання відкладеної цілі. Звідси випливає, що ***make*** виконує правила рекурсивно.

Для отримання вихідного файлу виконаємо команду ***make***.

```
debianuser@debianServ2017:~/programs/calc_cpp$ make calculator
g++ -c calculator.cpp
g++ -c main.cpp
g++ calculator.o main.o -o calculator
debianuser@debianServ2017:~/programs/calc_cpp$ ls -l
total 48
-rwxr-xr-x 1 debianuser debianuser 15216 Jan  9 14:19 calculator
-rw-r--r-- 1 debianuser debianuser  1667 Jan  9 13:15 calculator.cpp
-rw-r--r-- 1 debianuser debianuser   243 Jan  9 12:48 calculator.h
-rw-r--r-- 1 debianuser debianuser  9200 Jan  9 14:19 calculator.o
-rw-r--r-- 1 debianuser debianuser   197 Jan  9 13:08 main.cpp
-rw-r--r-- 1 debianuser debianuser  4064 Jan  9 14:19 main.o
-rw-r--r-- 1 debianuser debianuser   183 Jan  9 14:19 Makefile
```

Насправді у якості цілей програми ***make*** можуть виступати не тільки реальні файли. Усі, кому доводилося збирати програми з вихідних кодів, напевно знайомі з двома стандартними командами: ***make*** та ***make install***.

Командою ***make*** без аргументу здійснюють компіляцію програми, командою ***make install*** її встановлення. Такий підхід є досить зручним, оскільки все необхідне для збірки і розгортання програми у системі вже є включеним в один файл. Звернемо увагу на те, що в першому випадку ми не вказуємо ціль взагалі, а в другому ціллю є зовсім не створення файлу ***install***, а процес встановлення програми в систему. Виконувати такі дії дозволяють так звані фіктивні (phony) цілі. Існують наступні стандартні цілі:

– ***all*** – стандартна ціль за замовчуванням, при виклику її можна явно не вказувати;

– ***clean*** – очищення каталогів від всіх файлів отриманих в результаті компіляції;

– ***install*** – проведення інсталяції програми;

– ***uninstall*** – проведення деінсталяції програми.

Для того щоб програма ***make*** не шукала файли з такими назвами, їх слід визначити в ***Makefile*** за допомогою директиви ***.PHONY***.

Переробимо ***Makefile*** для останнього прикладу, додавши до нього описані вище фіктивні цілі.

Файл Makefile

```
.PHONY: all clean install uninstall
all: calculator
calculator: calculator.o main.o
	g++ calculator.o main.o -o calculator
calculator.o: calculator.cpp calculator.h
	g++ -c calculator.cpp
main.o: main.cpp calculator.h
	g++ -c main.cpp
clean:
	rm -f calculator calculator.o main.o
install:
	cp calculator /usr/local/bin/calculator
uninstall:
	rm -f /usr/local/bin/calculator
```

Отже, спробуємо застосувати фіктивні цілі. Спочатку очистимо робочий каталог проекту від файлів з попередньої компіляції.

```
debianuser@debianServ2017:~/programs/calc_cpp$ make clean
rm -f calculator calculator.o main.o
debianuser@debianServ2017:~/programs/calc_cpp$ ls -l
total 16
-rw-r--r-- 1 debianuser debianuser 1667 Jan  9 13:15 calculator.cpp
-rw-r--r-- 1 debianuser debianuser  243 Jan  9 12:48 calculator.h
-rw-r--r-- 1 debianuser debianuser  197 Jan  9 13:08 main.cpp
-rw-r--r-- 1 debianuser debianuser  374 Jan  9 14:32 Makefile
```

Тепер знову скомпілюємо програму-компілятор, але вже правильним способом, який полягає у виконанні програми ***make*** за замовчуванням.

```
debianuser@debianServ2017:~/programs/calc_cpp$ make
g++ -c calculator.cpp
g++ -c main.cpp
g++ calculator.o main.o -o calculator
debianuser@debianServ2017:~/programs/calc_cpp$ ls -l
total 48
-rwxr-xr-x 1 debianuser debianuser 15216 Jan  9 14:35 calculator
-rw-r--r-- 1 debianuser debianuser  1667 Jan  9 13:15 calculator.cpp
-rw-r--r-- 1 debianuser debianuser   243 Jan  9 12:48 calculator.h
-rw-r--r-- 1 debianuser debianuser  9200 Jan  9 14:35 calculator.o
-rw-r--r-- 1 debianuser debianuser   197 Jan  9 13:08 main.cpp
-rw-r--r-- 1 debianuser debianuser  4064 Jan  9 14:35 main.o
-rw-r--r-- 1 debianuser debianuser   374 Jan  9 14:32 Makefile
```

Наостанок покажемо як можна встановлювати та видаляти програму. Як видно з представленого вище файлу ***Makefile***, що при інсталяції він буде скопійований в один зі стандартних каталогів виконуваних файлів у системах Linux ***/usr/local/bin***, що дасть змогу запускати його у якості звичайної команди. При деінсталяції відповідний виконуваний файл буде видалений зі стандартного каталогу. Оскільки копіювати та видаляти файли з каталогу може тільки привілейований користувач, то виконувати команди ***make install*** та ***make uninstall*** ми можемо тільки від його імені. Це можна зробити, виконавши спочатку команди ***su***, зайшовши у систему як привілейований користувач, або команду ***sudo***, якщо нам дозволяється ця дія.

Розглянемо це на прикладі, спочатку інсталювавши, а потім деінсталювавши програму ***calculator***.

```
debianuser@debianServ2017:~/programs/calc_cpp$ su
Password:
root@debianServ2017:/home/debianuser/programs/calc_cpp# make install
cp calculator /usr/local/bin/calculator
root@debianServ2017:/home/debianuser/programs/calc_cpp# exit
exit
debianuser@debianServ2017:~/programs/calc_cpp$ calculator
Введіть число: 4
Зазначте арифметичну дію (+,-,*,/,pow,sqrt,sin,cos або tan): +
Другий доданок: 7
11
debianuser@debianServ2017:~/programs/calc_cpp$ su
Password:
root@debianServ2017:/home/debianuser/programs/calc_cpp# make uninstall
rm -f /usr/local/bin/calculator
root@debianServ2017:/home/debianuser/programs/calc_cpp# exit
exit
debianuser@debianServ2017:~/programs/calc_cpp$ calculator
-bash: /usr/local/bin/calculator: No such file or directory
```

Таким чином, після інсталяції програми її можна виконати як звичайну команду, а після її деінсталяції власне слово ***calculator*** у якості назви команди стане недоступним, що загалом є природно.

11.2. Інтерпретатори

У попередньому підрозділі детально було розглянуто компілятори ***gcc*** та ***g++***. Далі проаналізуємо основну інформацію про найбільш застосовувані у системах **Linux**, якими без сумніву є **Python** та **Java**.

11.2.1. Python

Python – є високорівневою, об'єктно-орієнтовною мовою програмування загального призначення. Наразі **Python** завдяки своїй універсальності та переносимості є дуже популярною мовою програмування. Саме тому, більшість дистрибутивів на ядрі Linux мають вбудований інтерпретатор цієї мови.

Станом на 2018 р. є актуальними дві версії Python: Python 3.x і Python 2.x. Кожна з них вимагає наявності власного інтерпретатора. У разі його відсутності можливе встановлення шляхом використання стандартного менеджера пакетів.

Запуск застосунків розроблених цією мовою програмування можливо запустити застосовуючи командний рядок: «python test.py».

```
root@chmnu-testl:/home/kio# python test.py
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
root@chmnu-testl:/home/kio#
```

Як і bash-сценарії застосунки **python** можливо запустити без явного виклику інтерпретатора шляхом введення адреси відповідного інтерпретатора у перший рядок, що називається «shebang» (#!).

```
#!/usr/bin/python
def casel():
    s = ""
    for i in range(10):
        print("Hello");
casel()
```

Але за такого варіанта запуску застосунку, необхідно задати відповідні права доступу:

```
root@chmnu-testl:/home/kio# chmod u+x test.py
```

У випадку відсутності необхідних прав доступу на етапі запуску, застосунок виведе повідомлення про помилку.

```
root@chmnu-testl:/home/kio# ./test.py
bash: ./test.py: Permission denied
root@chmnu-testl:/home/kio#
```

11.2.2. Java

Високорівневою об'єктно-орієнтованою мовою програмування також є Java. Обробка коду здійснюється в декілька етапів: спочатку код написаний на мові Java перетворюється в байт-код Java, а байт-код Java натомість передається віртуальній машині Java (JVM), а потім уже передає інструкції обладнанню.

Усі дистрибутиви, що базуються на ядрі Linux підтримують можливість роботи з Java без встановлення додаткових пакетів. Робота з Java можлива шляхом встановлення кількох пакетів:

– **default-jre** (Java Runtime Environment, мінімальна реалізація віртуальної машини Java для запуску застосунків) та **default-jdk** (Java Development Kit, комплект розробника включаючи javac) – пакети, що входять до репозиторіїв Debian;

– **OpenJDK –** проект (та однойменний пакет), що містить лише відкритий програмний код віртуальна машина ньому має назву HotSpot. На цей пакет за замовчуванням встановлено певні сучасні дистрибутиви (наприклад Debian 9);

– **Oracle JDK –** пакет компанії Oracle, який для встановлення потребує підключення додаткового репозиторію: «**add-apt-repository ppa:webupd8team/java**» (адреса репозиторію може відрізнятися для кожного дистрибутиву). Після підключення репозиторію і оновлення списку пакетів стандартний пакетний менеджер може завантажити інсталятор **java** та передати керування процесом встановлення «***apt-get install oracle-java8-installer***».

Отже для перевірки наявності встановленого пакету для роботи з Java, користувач може ввести у консолі «***java –version***», на що у випадку їх наявності отримає відповідь про версію встановленого Java Development Kit:

```
kio@chmnu-testl:~$ java  -version
openjdk version "1.8.0_181"
OpenJDK Runtime Environment (build 1.8.0_181-8u181-b13-2~deb9u1-b13)
OpenJDK 64-Bit Server VM (build 25.181-b13, mixed mode)
kio@chmnu-testl:~$
```

На відміну від **Python** мова **Java** потребує наявності додаткового файлу з байт-кодом для JVM. Сгенерувати його можливо з консолі, на основі файлу з кодом Java. Першим кроком буде створення файлу Java:

```
public class HelloWorld
{
    public static void main(String [] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Для генерації байт-коду Java необхідно виконати команду javac, що генерує відповідний код для JVM. У випадку відсутності помилок можна відразу запустити застосунок.

```
kio@chmnu-testl:~/java$ javac HelloWorld.java
kio@chmnu-testl:~/java$ java -cp . HelloWorld
Hello, World!
kio@chmnu-testl:~/java$
```

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бондаренко М. Ф. Операційні системи : навч. посібник / М. Ф. Бондаренко, О. Г. Качко. – Х. : Компанія СМІТ, 2008. – 432 с.
2. Шеховцев В. А. Операційні системи. К. : Видавнича група BHV, 2005. – 576 с.
3. Шоттс У. Командная строка Linux. Полное руководство / У. Шоттс– СПб. : Питер, 2017. – 480 с. : ил.
4. Блум Р., Бреснахэн К. Командная строка Linux и сценарии оболочки. Библия пользователя : 2-е изд. / Р. Блум, К. Бреснахэн ; пер. с англ. – М. : ООО «И. Д. Вильямс», 2012. – 784 с. : ил.
5. Операционная система Linux : курс лекций : учебное пособие / Г. В. Курячий, К. А. Маслинский – М. : ALT Linux ; Издательство ДМК Пресс, 2010. – 348 с. : ил. : 2-е изд., исправленное.
6. Немет Э. Руководство администратора Linux : 2-е издание / Э. Немет, Г. Снайдер, Т. Хейн ; пер. с англ. – М. : ООО «И. Д. Вильямс», 2011. – 1072 с. : ил.
7. Уорд Б. Внутреннее устройство Linux / Б. Уорд. – СПб. : Питер, 2016. – 384 с. : ил.
8. Кофлер М. Linux. Полное руководство / М. Кофлер. – СПб. : Питер, 2011. – 800 с. : ил.
9. Столлингс В. Операционные системы. Внутреннее устройство и принципы проектирования / В. Столлингс ; пер. с англ. Д. Я. Иваненко и др. – 4-е изд. – М. ; СПб. ; К. : Издательский дом «Вильямс», 2004. – 843 с.
10. Таненбаум Э. Современные операционные системы / Э. Таненбаум. – 4. изд. – СПб. : Питер, 2015. – 1120 с.
11. Олифер В. Г. Сетевые операционные системы : учеб. пособие для студ. вузов, обуч. по направлению подгот. дипломированных спец. «Информатика и вычислительная техника» / В. Г. Олифер, Н. А. Олифер. – СПб. : ПИТЕР ; М. ; СПб. ; Нижний Новгород : [б. и.], 2003. – 538 с.
12. Робачевский А. М. Операционная система UNIX / А. М. Робачевский. – СПб. : BHV – Санкт-Петербург, 1997. – 528 с.
13. Стахнов А. А. Linux : наиболее полное руководство / А. А. Стахнов. – СПб. : БХВ – Петербург, 2003. – 881 с.

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

Навчальне видання

Гліб Валентинович
ГОРБАНЬ,
Ігор Олександрович
КАНДИБА

ОПЕРАЦІЙНА СИСТЕМА LINUX

Навчальний посібник

Редактор *А. Грубкіна.*
Технічний редактор, комп'ютерна верстка *Н. Хасянова.*
Друк *С. Волинець*. Фальцювально-палітурні роботи *О. Мішалкіна.*

Підп. до друку 03.10.2019.
Формат 60 × 84$^{1}/_{16}$. Папір офсет.
Гарнітура «Times New Roman». Друк ризограф.
Ум. друк. арк.16,04. Обл.-вид. арк. 9,80.
Тираж 30 пр. Зам. № 5716.

54003, м. Миколаїв, вул. 68 Десантників, 10.
Тел.: 8 (0512) 50–03–32, 8 (0512) 76–55–81, e-mail: rector@chmnu.edu.ua.
Свідоцтво суб'єкта видавничої справи ДК № 6124 від 05.04.2018.