МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені І. І. МЕЧНИКОВА
ФАКУЛЬТЕТ МАТЕМАТИКИ, ФІЗИКИ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

ОПЕРАЦІЙНІ СИСТЕМИ І СЕРЕДОВИЩА

Методичні вказівки
до виконання лабораторних робіт
для студентів факультету математики, фізики та
інформаційних технологій
першого (бакалаврського) рівня освіти,
спеціальності 126 «Інформаційні системи та технології»

Одеса
Олді+
2023

УДК 004.451:004.633(072)
О-609

Укладачі:
Розновець О. І., старший викладач кафедри математичного забезпечення комп'ютерних систем;
Трубіна Н. Ф., старший викладач кафедри математичного забезпечення комп'ютерних систем.

Рецензенти:
Вербицький В. В., к.ф.-м.н, доцент кафедри оптимального управління та економічної кібернетики Одеського національного університету імені І. І. Мечникова;
Антоненко О. С., к.ф.-м.н., доцент кафедри математичного забезпечення комп'ютерних систем Одеського національного університету імені І. І. Мечникова.

Рекомендовано Вченою радою
факультету Математики, фізики та інформаційних технологій
ОНУ імені І. І. Мечникова.
Протокол № 2 від 30 жовтня 2023 р.

О-609 **Операційні** системи і середовища : метод. вказівки до виконання лаб. робіт студентів факультету математики, фізики та інформаційних технологій першого (бакалаврського) рівня освіти, спец. 126 «Інформаційні системи та технології» / уклад.: О. І. Розновець, Н. Ф. Трубіна, – Одеса : Олді+, 2023. – 84 с.

Пропоновані методичні вказівки стануть у нагоді при виконанні лабораторних робіт з обов'язкової дисципліни «Операційні системи і середовища», яка викладається студентам першого (бакалаврського) рівня вищої освіти спеціальності 126 «Інформаційні системи та технології» факультету математики, фізики та інформаційних технологій Одеського національного університету імені І. І. Мечникова. Методичні вказівки можуть бути корисні для студентів ІТ-спеціальностей закладів вищої освіти, які вивчають операційні системи.

УДК 004.451:004.633(072)

ЗМІСТ

ВСТУП

Дисципліна «Операційні системи і середовища» є обов'язковим компонентом освітньо-професійної програми підготовки здобувачів вищої освіти першого (бакалаврського) рівня вищої освіти за спеціальністю 126 – «Інформаційні системи та технології».

Дисципліна викладається студентам першого курсу. Загальна кількість кредитів, відведених на вивчення дисципліни – 4,5. Загальна кількість годин – 135, з них лекцій – 34 години, лабораторних робіт – 34 години, самостійна робота – 67 годин.

Мета викладання дисципліни: формування у студентів знань про архітектуру, компоненти та функції сучасних ОС, а також про окремі аспекти функціонування та особливості різних ОС.

Завдання дисципліни: надання уявлення про ОС, ознайомлення з історією розвитку ОС, ознайомлення з архітектурними принципами побудови ОС, ознайомлення зі структурою ОС та її функціональними компонентами, надання уявлення про особливості різних ОС, формування вмінь і навичок роботи у ОС UNIX.

Отримані при вивченні дисципліни знання та практичні навички є безпосередньою базою при опануванні студентами дисциплін «Комп'ютерна схемотехніка та архітектура комп'ютерів», «Комп'ютерні мережі», «Технології захисту інформації», «Проектування інформаційних систем», «Інженерія програмного забезпечення», «Технології розподілених систем та паралельних обчислень», а також для дипломного проектування.

У результаті вивчення навчальної дисципліни студент повинен

знати*:*

– архітектурні принципи організації сучасних операційних систем (ОС);

– структуру ОС та її функціональні компоненти;

– особливості окремих аспектів функціонування ОС на прикладі ОС сімейств Windows та UNIX.

вміти*:*

– працювати в різних операційних системах;

– застосовувати команди ОС UNIX та мову програмування командного інтерпретатора bash з метою створення елементів системного програмного забезпечення для виконання різноманітних задач.

Цикл лабораторних робіт курсу «Операційні системи і середовища» направлений на вивчення принципів роботи в UNIX-подібних ОС, зокрема, ОС Linux, та на здобуття навичок програмування мовою командного інтерпретатора.

UNIX – одна з найпопулярніших в світі ОС, і це стало можливим через наступні причини:

– код ОС UNIX написаний мовою високого рівня С, що зробило її простою для розуміння, змін та перенесення на інші платформи;

– UNIX – відмовостійка, багатозадачна, розрахована на багатьох користувачів ОС, здатна виконувати велику кількість різноманітних функцій, зокрема, працювати, як обчислювальний сервер, сервер бази даних, веб-сервер, поштовий сервер і т.д.;

– основою всього сімейства операційних систем UNIX є принципово однакова архітектура і ряд стандартних інтерфейсів;

– наявність простого, але потужного модульного користувацького інтерфейсу;

– використання єдиної ієрархічної файлової системи, що легко обслуговується, та застосування уніфікованого інтерфейсу файлової системи для здійснення доступу до різноманітних зовнішніх пристроїв;

– існування великої кількості дистрибутивів від різних постачальників ОС, що розповсюджуються за умовами вільних ліцензій;

– велика кількість застосунків, призначених для роботи в ОС UNIX, в тому числі вільно розповсюджуваних.

Однією з найпоширеніших некомерційних UNIX-подібних ОС є Linux. Будучи системою з відкритим кодом, Linux успішно застосовується в учбовому процесі в освітніх установах всього світу. На сьогоднішній момент Linux – найсучасніша ОС, що швидко розвивається, майже миттєво вбираючи в себе найостанніші технологічні новинки. Системи Linux наразі є основними для суперкомп'ютерів і серверів, для вбудованих систем і мобільних пристроїв.

Мета даних методичних вказівок – навчити студента працювати в середовищі Linux, прищепити навички використання різних команд цієї ОС та створення сценаріїв командного інтерпретатора. Методичні вказівки містять теоретичні відомості з кожної теми, що розглядається, контрольні питання та завдання до виконання.

Порядок виконання лабораторних робіт. Студент повинен ознайомитися з теоретичним матеріалом до кожної лабораторної роботи, відповісти на контрольні питання, виконати завдання, оформити і захистити звіт. У звіті мають міститися номери і формулювання завдань та команди, що необхідні для їх виконання.

Під час захисту лабораторних робіт студент повинен:

– пояснити вибір обраних алгоритмів та засобів виконання завдань;

– роз’яснити структуру і принцип роботи створених ним команд (сценаріїв) та продемонструвати їх роботу.

Кількість балів, що отримує студент за виконання лабораторних робіт, визначається робочою програмою дисципліни.

ЛАБОРАТОРНА РОБОТА №1.
РОБОТА З ФАЙЛАМИ І КАТАЛОГАМИ

Мета роботи: отримати уявлення про структуру файлової системи ОС UNIX (Linux), ознайомитися з основами функціонування командного інтерпретатора, освоїти основні принципи роботи в ОС UNIX (Linux), вивчити основні команди і отримати практичні навички роботи з файлами та каталогами.

Постановка задачі: виконати завдання, використовуючи відповідні команди ОС UNIX (Linux).

Теоретичні відомості

Поняття файлу та файлової системи

Файлова система – це частина ОС, призначення якої полягає в тому, щоб організувати ефективну роботу з даними, що зберігаються в зовнішній пам'яті, і забезпечити користувачеві зручний інтерфейс при роботі з такими даними.

Файл – це іменована логічно пов'язана сукупність даних, асоційована з носієм інформації і використовувана як базовий об'єкт взаємодії з даними в операційних системах. Інформація на носіях інформації може зберігатися лише у вигляді файлів. Кожен файл характеризується деякою кількістю атрибутів.

Файлова система UNIX зберігає інформацію про кожен файл в структурі, званій **індексним дескриптором** (inode). Кожен inode містить дані про тип файлу, його розмір, кількість жорстких посилань на файл, власника файлу, права доступу до файлу, дату/час останньої модифікації і дату/час останнього доступу до файлу. Саме з індексним дескриптором працює ОС при зверненні до файлу.

В ОС UNIX підтримуються наступні типи файлів.

– **Звичайні файли**, що являють собою просто послідовність байтів; на структуру таких файлів не накладається ніяких обмежень – це можуть бути текстові документи, виконувані програми, мультимедійні дані.

– **Каталоги**. **Каталог** – це файл, який містить імена файлів, що в ньому знаходяться, а також покажчики на їхні індексні дескриптори. Каталоги визначають положення файлу в дереві файлової системи, оскільки сам файл не містить інформації про своє місцезнаходження. Каталог, на який є посилання в даному каталозі, називається **підкаталогом** або **вкладеним каталогом**. **Батьківським** називається каталог, в якому міститься даний каталог. Для кореневого каталогу (див. нижче) батьківським є він сам.

– **Посилання**, які діляться на 2 типи:

1) **Жорстке посилання** (hard links). На файл можна посилатися з декількох каталогів одночасно і навіть з кількох елементів одного і того ж каталогу, причому у всіх посилань можуть бути різні імена. Це створює

ілюзію того, що файл в один і той же час знаходиться в різних каталогах. Атрибути файлу при цьому є загальними для всіх посилань. UNIX підраховує кількість посилань, що вказують на кожен файл, і при видаленні файлу не звільняє блоки даних до тих пір, поки не буде видалене останнє посилання на нього. Посилання такого роду називаються жорсткими. Вони *не є окремим типом файлів*. Жорстке посилання в UNIX неможливо відрізнити від імені файлу – вони ідентичні. Жорстке посилання вказує безпосередньо на індексний дескриптор файлу.

2) **Символьне (м'яке) посилання** (symbolic links) вказує на файл по його імені. Цей вид посилань *є окремим типом файлів* і забезпечує можливість замість шляхового імені файлу вказувати псевдонім. Коли ОС стикається з символьним посиланням при пошуку файлу, вона витягує з нього шляхове ім'я. Файл, що адресується символьним посиланням, і саме посилання мають різні типи об'єктів файлової системи. Файл, на який створюється символьне посилання, не обов'язково повинен існувати. При видаленні файлу автоматично знищуються всі пов'язані з ним символьні посилання.

– **Файли пристроїв**, що дозволяють програмам взаємодіяти з зовнішніми пристроями обчислювальної системи. За всю роботу по управлінню конкретним пристроєм відповідає спеціальна програма – драйвер пристрою. Файли пристроїв можна уявити як шлюзи, через які драйверам пристроїв передаються запити. В UNIX розрізняють символьні та блочні файли пристроїв. **Файли символьних пристроїв** дозволяють проводити з пристроєм послідовний обмін даними по одному байту та не використовують буферизацію в процесі виконання операцій введення-виведення; на противагу цьому **файли блокових пристроїв** дозволяють проводити обмін даними у вигляді пакетів фіксованої довжини – блоків.

– **Сокети** – інкапсулюють з'єднання між процесами, які виконуються на одному або на різних комп'ютерах мережі. Звернення до них здійснюється через відповідні об'єкти файлової системи. Незважаючи на те, що ці об'єкти розпізнаються як файли, процеси, які беруть у з'єднанні, не можуть здійснювати над файлами сокетів операції читання і запису. Однак, на відміну від звичайних файлів, сокет являє собою віртуальний об'єкт, який існує, поки на нього посилається хоча б один з процесів.

– **Іменовані канали**. Подібно до сокетів, іменовані канали забезпечують взаємодію двох процесів, які виконуються на одному комп'ютері.

Структура файлової системи UNIX

Файлова система ОС UNIX має ієрархічну (деревоподібну) структуру (рис. 1.1), основою якої є кореневий каталог, що має ім'я /. Розташування файлів у

файловому дереві не визначається їх розташуванням на тому чи іншому фізичному або логічному диску. Файлові структури, що знаходяться на різних дисках, в тому числі на дисках інших комп'ютерів, за допомогою спеціальної команди (`mount`) монтуються (підключаються) на файлове дерево UNIX, стаючи частиною єдиного дерева файлів та каталогів. Всі операції над файлами з точки зору користувача виконуються однаковим чином, незалежно від їх фізичного розташування.

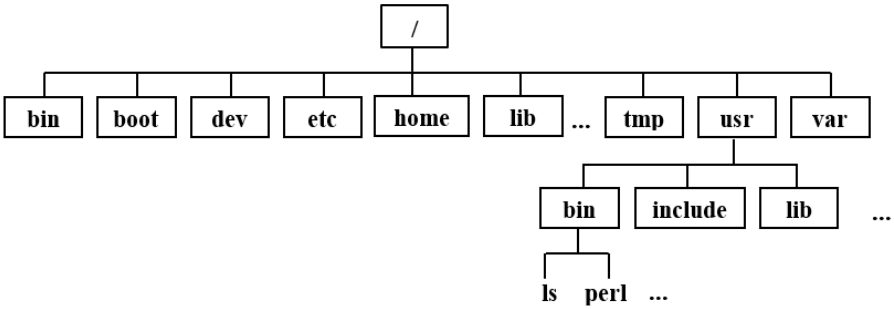

Рисунок 1.1 – Деревоподібна структура файлової системи ОС UNIX

Кореневий каталог UNIX зазвичай містить наступні каталоги:

bin містить найбільш використовувані команди

boot містить завантажувач операційної системи

dev зберігає файли пристроїв

etc зберігає більшість конфігураційних файлів системи

home зберігає домашні каталоги користувачів

lib зберігає найважливіші бібліотеки, необхідні для роботи програм з каталогів **/bin** і **/sbin**

mnt використовується для монтування нових файлових систем

opt використовується для установки додаткового програмного забезпечення

proc віртуальна файлова система, що представляє стан ядра ОС і запущених процесів у вигляді файлів

run в Linux інформація про систему з моменту початку її завантаження, в тому числі файли ідентифікаторів запущених процесів, сокети і т.п.

root домашній каталог суперкористувача

sbin містить програми для адміністрування і налаштування системи

sys містить інформацію про пристрої, драйвери і деякі властивості ядра ОС

tmp для зберігання тимчасових файлів

usr містить дані, які не змінюються програмами (цей каталог може бути підключений в режимі «лише для читання» без шкоди для функціональності)

var містить файли, які піддаються найбільш частій зміні, наприклад, кеші різних програм, файли системних журналів, поштові скриньки користувачів і інші дані, які мають тенденцію часто і швидко змінювати свій розмір

Послідовність імен каталогів, розділених символом «/», що веде від деякого каталогу до каталогу, в якому розташовується даний файл, називається **маршрутом до** цього **файлу**. Послідовність *ім'я_маршрута*/*ім'я_файла* називається **шляховим ім'ям файлу**. Якщо шляхове ім'я відраховується від кореневого каталогу, воно називається **абсолютним (повним)**, в іншому випадку – **відносним**.

Наприклад, каталог **/usr**, крім інших, містить каталоги **bin** і **include**. Повні імена цих каталогів будуть такими: **/usr/bin** і /**usr/include**. Якщо в каталозі **/usr/bin** міститься файл **perl**, то повним ім'ям файлу **perl** буде **/usr/bin/perl**. Якщо ж користувач в даний момент знаходиться в каталозі **/usr** і йому необхідно відкрити файл **/usr/include/sys/conf**, то він може звернутися до цього ж файлу за відносним іменем **include/sys/conf**.

Існує два спеціальних імені каталогів: **.** – ім'я поточного каталогу, **..** – ім'я батьківського каталогу.

У іменах файлів може використовуватися будь-яка послідовність з букв, цифр і спецсимволів, окрім символу «/», тому що даний символ використовується як роздільник імен в складі шляху. Довжина імені обмежується 255 символами (при використанні кирилиці – 127 символів). Великі і малі літери в іменах файлів розрізняються: файл **myfile** і файл **MyFile** – це різні файли. Файл може включати в себе розширення, яке визначається як частина імені файлу, що розташовується після останньої точки. Файли, імена яких починаються з точки, є прихованими.

Існують символи, які в іменах файлів і каталогів потрібно використовувати з обережністю, оскільки вони мають спеціальне призначення: «*», «\», «&», «<», «>», «;», «(», «)», «|», а також символи пробілу і табуляції. Для того, щоб ці символи сприймалися як частина імені файлу або каталогу, кожен спецсимвол необхідно попередити символом «\» (ця операція називається екрануванням спецсимволу).

Командний інтерпретатор, командний рядок і команди

Для організації інтерфейсу користувача до складу UNIX входить **командна оболонка або командний інтерпретатор** (shell), що виконує наступні функції:

– виконує команди, що задані в командному рядку або надходять зі стандартного введення або вказаного файлу;

– здійснює обробку шаблонів імен файлів;

– здійснює перенаправлення введення і виведення;

– організовує конвеєризацію команд;

– є інтерпретатором розвиненої командної мови, яка дозволяє користувачеві створювати складні сценарії виконання програм.

Існують різні версії командних інтерпретаторів, найбільш популярними з них є sh (Bourne Shell), bash (Bourne Again Shell), csh (C Shell), ksh (Korn Shell). Далі (якщо не зазначено інше) розглядається інтерпретатор bash, який часто використовується в Linux за замовчуванням.

Командний рядок має наступний вигляд:

```
$ команда
```

Символ «$» є запрошенням командного інтерпретатора. Йому можуть передувати інші символи, що позначають, наприклад, ім'я поточного каталогу, ім'я комп'ютера та ім'я користувача.

Командний інтерпретатор обробляє команди трьох типів. По-перше, в якості імені команди може бути вказано ім'я виконуваного файлу в об'єктному коді, отриманого в результаті компіляції вихідного тексту програми. По-друге, ім'ям команди може бути ім'я командного файлу, що містить набір інструкцій, оброблюваних командним інтерпретатором (такі файли називаються сценаріями або скриптами). По-третє, команда може бути внутрішньою командою мови командного інтерпретатора (такими є програми, що знаходяться в каталогах **/bin** та **/usr/bin**).

Внутрішні команди мови командного інтерпретатора зазвичай мають формат:

```
ім'я_команди прапори аргументи
```

Прапори (або ключі) служать для модифікації режимів роботи команди і є наборами символів, яким передує символ «-» (короткі імена ключів, наприклад, `-d`) або комбінація «--» (довгі імена ключів, наприклад, `--wildcards`). Деякі ключі можуть позначатися як коротким, так і повним ім'ям, наприклад ключ з коротким ім'ям `-u` має аналог з довгим ім'ям `--user`.

Аргументи (або параметри) команди вказують на об'єкти, над якими виконуються операції. У більшості випадків аргументами команд є імена файлів. Наприклад, наведена нижче команда означає: «Виконати команду `ls` (відобразити інформацію про файл) з ключем `-l` (детальна інформація) для файлу `a.out`»:

```
ls -l a.out
```

Якщо необхідно використовувати два і більше однобуквених ключа, більшість команд дозволяють об'єднувати їх. Наприклад, дві наведені нижче команди ідентичні:

```
ls -laR a.out
ls -l -a -R a.out
```

Деякі ключі вимагають наявності параметра, який вказується після ключа, в цьому випадку останній не можна об'єднувати з іншим ключем. Наприклад,

наведена нижче команда означає: «Виконати команду `gzip` (архівування файлів) над усіма файлами каталогу **work** з ключами `-v` (відображати в процесі обробки імена файлів і відсоток стиснення), `-r` (обробити весь вміст каталогу) і `-S` (замість стандартного розширення .gz використовувати розширення .arch)»:

```
gzip -vr -S arch work
```

В одному рядку може міститися декілька команд; в цьому випадку їх необхідно відокремлювати одну від одної символом «`;`».

Одну команду можна розмістити на декількох екранних рядках. Для цього перед натисканням клавіші [Enter] необхідно поставити символ «`\`».

Командний інтерпретатор шукає імена команд в зазначеному наборі каталогів, який можна змінити за бажанням користувача. Список цих каталогів є значенням змінної середовища `PATH`.

Шаблони підстановки

При роботі часто потрібно виконувати однотипні дії над групою файлів. Описувати ці дії по відношенню до кожного окремого файлу є трудомісткою задачею, а шаблони імен файлів дозволяють швидко виділяти групи файлів, чиї імена задовольняють певним умовам.

Шаблони імен файлів можна формувати з використанням таких спецсимволів:

`*` відповідає будь-якій (можливо, порожній) послідовності символів;

`?` відповідає точно одному будь-якому символу;

`[]` використовуються для групування символів в набори, що можуть задаватися:

– явним перерахуванням символів, без позначення роздільників між ними, наприклад `[atz56,=]` – символ відповідає одній із трьох букв a, t і z, або одній з цифр 5 і 6, або одному з спецсимволів кома і знак рівності;

– шляхом визначення діапазону, наприклад `[a-n]` – символ відповідає маленькій букві від a до n;

– комбінацією цих способів, наприклад `[0-9ij]` – символ відповідає або цифрі, або одній з букв i або j;

`!` використовується всередині квадратних дужок перед початком визначення набору символів для заперечення цього набору, тобто ім'я файлу *не повинно* містити символів, зазначених в дужках.

Приклади:

`f*`	відповідає файлам з іменами, що починаються з літери f
`*F*`	відповідає файлам з іменами, що містять літеру f
`??[a-d]*`	відповідає файлам з іменами, в яких третьою літерою є a, b, c чи d
`program.?`	відповідає файлам з іменами **program**, що мають однобуквені розширення

`[!A-Z]*[2468]` відповідає файлам з іменами, які не починаються з великої літери і закінчуються цифрою 2, 4, 6 або 8

Якщо ім'я файлу повинно містити будь-який спецсимвол, то при визначенні шаблону цей спецсимвол необхідно екранувати, наприклад:

`[AEIOUYaeiouy]\???` відповідає файлам з іменами, що починаються з голосної літери, за якою слідує символ знака питання, а за ним – ще два довільних символи

При використанні діапазонів символів слід враховувати, що вони можуть залежати від обраних налаштувань локалізації. Наприклад, діапазон `[b-e]` означає символи від b до e включно. В англійській мові, де сортування букв йде по порядку ABC...XYZabc…xyz, вказаному набору відповідає набір символів b, c, d, e. Згідно з правилами, наприклад, російської мови, сортування тих же символів виконується в іншому порядку (аАбБвВ ... юЮяЯаAbBcC ... xXyYzZ), і тому ж діапазону відповідають символи b, B, c, C, d, D, E. Для вирішення таких проблем використовуються класи символів:

`[:upper:]`	латинські літери верхнього регістру
`[:lower:]`	латинські літери нижнього регістру
`[:alpha:]`	латинські літери верхнього і нижнього регістрів
`[:alnum:]`	латинські літери верхнього і нижнього регістрів, цифри
`[:digit:]`	цифри
`[:xdigit:]`	шістнадцяткові цифри
`[:punct:]`	знаки пунктуації
`[:blank:]`	пробіл і табуляція
`[:space:]`	символи пропуску (пробіл, табуляція, перевід рядка і т.ін.)
`[:cntrl:]`	управляючі символи (клавіша [Ctrl] + символ)
`[:graph:]`	друковані символи (без пробілу і табуляції)
`[:print:]`	друковані символи (з пробілом і табуляцією)

Отримання довідки про команди UNIX

Отримати довідку про роботу команд можна декількома способами.

1) **Man-сторінки** (від англ. manual pages – сторінки посібника) містять найбільше корисної інформації. Кожна сторінка присвячена якомусь одному об'єкту системи, наприклад, команді, системному виклику, файлу пристрою, бібліотечній функції тощо. Для того щоб продивитися сторінку з інформацією про роботу якої-небудь команди, потрібно виконати команду

```
man ім'я_команди
```

Наприклад, для переглядання довідки про роботу команди `find` необхідно виконати команду

```
man find
```

Man-сторінка займає, як правило, більше однієї сторінки екрану. Сторінки перегортаються вниз за допомогою клавіші пробілу або клавіші [Page Down], вгору – за допомогою клавіші [Page Up], для зсуву на один рядок вперед можна застосовувати [Enter] або клавішу [↓], а на один рядок назад – клавішу [↑]. Перехід на початок і кінець тексту виконується після натискання клавіш [g] і [G] відповідно (Go). Вихід здійснюється після натискання клавіші [q] (Quit).

2) **Використання ключа `--help`**

У кожної UNIX-команди є ключ `--help`, і запустивши команду з цим ключем, можна переглянути коротку довідку про використання даної команди.

Домашні каталоги користувачів

В UNIX у кожного користувача обов'язково є домашній каталог, який зберігається в каталозі **/home**. Наприклад, для користувача anna домашнім каталогом є **/home/anna**.

Домашній каталог (home directory) – це каталог, призначений для зберігання власних даних користувача. Як правило, домашній каталог є поточним безпосередньо після реєстрації користувача в системі. Зазвичай доступ інших користувачів до чужого домашнього каталогу обмежений: найчастіше користувачі можуть читати вміст файлів один одного, але не можуть їх змінювати або видаляти.

Операції над файлами і каталогами

Визначення імені поточного каталогу

Команда `pwd` (від англ. print working directory – надрукувати ім'я робочого каталогу) повертає повне ім'я поточного каталогу, в якому знаходиться користувач.

Отримання інформації про каталог або файл

Вміст будь-якого каталогу можна переглянути за допомогою команди **`ls`** (від англ. list – список), що приймає як параметр ім'я каталогу. Ім'я може бути задано у вигляді повного або відносного шляху. Подана без параметрів, команда `ls` виводить список файлів і каталогів, що містяться в поточному каталозі.

Наприклад, для того щоб отримати список файлів в каталозі **/usr/sbin**, необхідно ввести команду

```
ls /usr/sbin
```

У команди `ls` є велика кількість прапорів, які потрібні головним чином для того, щоб виводити додаткову інформацію про файли в каталозі або виводити

зазначений список файлів (замість повних імен файлів можна використовувати шаблони). Нижче наведені найбільш уживані прапори команди `ls`.

`-a, --all`	виводить список всіх імен файлів каталогу, включаючи приховані
`-A, --almost-all`	виводить список всіх імен файлів каталогу, крім поточного (.) і батьківського (..) каталогу
`-d, --directory`	виводить імена вкладених каталогів без їх вмісту
`-I` ***`pattern`***`,` **`--ignore=`**`pattern`	не включає в виведений список файли, імена яких збігаються з шаблоном *`pattern`*
`-R, --recursive`	відображає рекурсивно вміст всіх каталогів, що входять в зазначений
`-l, --format=long`	виводить детальну інформацію про файли, включаючи тип файлу, права доступу, кількість жорстких посилань, ім'я власника і групи-власника, розмір в байтах, дату/час останньої модифікації
`-r, --reverse`	при використанні впорядкування порядок змінюється на зворотний
`-S, --sort=size`	впорядковує файли за розміром: найбільші файли йдуть першими
`-t, --sort=time`	впорядковує файли за часом модифікації: найновіші файли йдуть першими
`-X, --sort=extension`	впорядковує файли по розширенням; файли без розширень розташовуються раніше
`-U, --sort=none`	не виконує впорядкування і включає в список файли в тому порядку, в якому вони йдуть в каталозі
`-u, --time=atime`	при використанні ключа `-l` замість дати/часу останньої модифікації виводить дату/час останнього доступу до файлу
`--color`	для розпізнавання типів файлів використовує різні кольори
`-F, --classify`	додає до кожного імені файлу символ, який вказує тип: * – звичайні виконувані файли, / – каталоги, @ – символьні посилання, = – сокети, – іменовані канали.
`-T cols, --tabsize=`*`cols`*	табулює відповідно до ширини стовпця, що дорівнює *`cols`* (за замовчуванням 8)

Приклади:

`ls -l ./[0-9]*[0-9] -r`	виведення докладної інформації про файли поточного каталогу, імена яких починаються і закінчуються цифрами, із застосуванням зворотного впорядкування

`ls -lda /etc`	виведення докладної інформації про вміст каталогу **/etc**, включаючи приховані файли, без відображення вмісту вкладених каталогів

Переміщення по дереву каталогів

Для зміни поточного каталогу використовується команда **`cd`** (від англ. change directory – змінити каталог), що приймає один параметр: повний або відносний шлях до каталогу, в який потрібно переміститися, тобто зробити поточним. Наприклад:

```
cd /usr/lib
```

Для переміщення в батьківський каталог зручно скористатися командою `cd ..`

Необхідність повернутися в домашній каталог з довільної точки файлової системи виникає досить часто, тому командний інтерпретатор підтримує позначення домашнього каталогу за допомогою символу «~». Для того щоб перейти в домашній каталог з будь-якого іншого каталогу, досить виконати команду `cd ~`

За допомогою символу «~» можна посилатися і на домашні каталоги інших користувачів, використовуючи конструкцію *~ім'я_користувача*. Наприклад, користувач anna при виконанні команди `cd ~victor` може перейти в домашній каталог користувача victor.

Команда `cd`, подана без параметрів, еквівалентна команді `cd ~` і робить поточним каталогом домашній каталог користувача.

Створення каталогів

Для створення каталогу використовується команда **`mkdir`** (від англ. make directory – створити каталог). Вона застосовується з обов'язковим параметром, що вказує ім'я створюваного каталогу (або список імен каталогів, вказаних через пробіл).

Приклади:

`mkdir example test`	створення каталогів з іменами **example** і **test** в поточному каталозі
`mkdir /home/anna/example`	при зазначенні повного або відносного шляху каталог буде створений відповідно до вказаної ієрархії каталогів

Створення файлів

Створення файлу здійснюється за допомогою команди `touch`. Взагалі, ця команда використовується для зміни тимчасових міток доступу (визивається з прапором `-a, --time=`*`access_time`*) і модифікації (визивається з прапором `-m`,

`--time=`*`modification_time`*) зазначених файлів. Однак, спроба виконання даної команди над неіснуючим файлом призведе до його створення.

Параметром команди `touch` є ім'я файлу (або список імен файлів через пробіл). Наприклад, наступна команда призведе до створення у поточному каталозі трьох файлів з іменами **f1.txt**, **f2.lst** і **f3**:

```
touch f1.txt f2.lst f3
```

Прапор `-c` (або `--no-create`) запобігає створенню файлу, якщо він не існує.

Копіювання файлів і каталогів

Для копіювання файлів і каталогів застосовується команда **`cp`** (від англ. copy – копіювати). Команда `cp` має два обов'язкових параметри: перший – файл або каталог, що копіюється (або їх список), другий – файл або каталог призначення. В іменах файлів і каталогів можна використовувати повні та відносні шляхи і шаблони. Якщо останній аргумент є ім'ям існуючого каталогу, то команда `cp` копіює кожен вказаний вихідний файл в файл з тим же ім'ям у каталог призначення. Якщо аргументами є імена двох файлів (каталогів), то команда `cp` копіює вихідний файл (каталог) в файл (каталог), ім'я якого задано файлом (каталогом) призначення. Якщо останній аргумент не є ім'ям каталогу і при цьому задано декілька імен файлів, то команда `cp` завершується з помилкою.

Команда `cp` нерідко налаштована таким чином, що при спробі скопіювати файл поверх вже існуючого файлу ніякого попередження не виводиться. У цьому випадку файл буде просто перезаписаний, а дані, які містилися в старій версії файлу, безповоротно втрачені. Тому при використанні команди `cp` слід бути уважним.

Деякі прапори команди `cp`:

`-i, --interactive`	вимагає підтвердження при перезаписі існуючих файлів
`-R, --recursive`	копіює каталоги рекурсивно (разом з вмістом каталогу)
`-v, --verbose`	друкує ім'я кожного файлу перед його копіюванням
`-u, --update`	копіює лише за умови, що вихідний файл новіший, ніж файл призначення, або файл призначення взагалі відсутній

Приклади:

`cp -R prog dir`	рекурсивне копіювання каталогу **prog** в каталог **dir**
`cp ~/labs/lab1 /copy`	копіювання файлу **~/labs/lab1** в каталог **/copy**
`cp ~/labs/lab1 /copy/Lab_1`	копіювання файлу **~/labs/lab1** в каталог **/copy** з новим ім'ям **Lab_1**

Переміщення/перейменування файлів і каталогів

Для переміщення файлів і каталогів застосовується команда **`mv`** (від англ. move – рухати, пересувати). Переміщення файлу всередині однієї файлової

системи рівнозначно його перейменуванню: дані самого файлу при цьому залишаються на тих же секторах диска, а змінюються каталоги, в яких відбулося переміщення. Переміщення передбачає видалення посилання на файл з того каталогу, звідки він переміщений, і додавання посилання на цей же самий файл в той каталог, куди він переміщений. В результаті змінюється повне ім'я файлу, тобто розташування файлу в файловій системі.

Команда `mv` має два обов'язкових аргументи: перший – переміщуваний файл або каталог (або їх список), другий – файл або каталог призначення. Імена файлів і каталогів можуть бути задані за допомогою повного чи відносного шляху або шаблонів. Якщо останній аргумент є ім'ям існуючого каталогу, то команда `mv` переносить кожен вказаний вихідний файл в файл з тим же ім'ям у каталог призначення. Якщо аргументами є два файли, то команда `mv` перейменовує вихідний файл в файл, ім'я якого задано файлом призначення. Якщо останній аргумент не є ім'ям каталогу і при цьому задано кілька імен файлів, то команда `mv` завершується з помилкою.

Деякі прапори команди `mv`:

`-i, --interactive`	вимагає підтвердження при перезаписі існуючих файлів
`-v, --verbose`	друкує ім'я кожного файлу перед його переміщенням
`-u, --update`	не переміщує вихідний файл, якщо файл призначення має такий самий або пізніший час модифікації

Приклади:

`mv /project/arch/1.ar archives`	переміщення файлу **/project/arch/1.ar** в каталог **archives**
`mv lab[1-3] ~/old_files`	переміщення файлів **lab1**, **lab2** і **lab3** з поточного каталогу в каталог **~/old_files**
`mv file FILE`	перейменування файлу **file** в **FILE**
`mv ~/labs/lab1 /copy/work1`	переміщення файлу **~/labs/lab1** в каталог **/copy** з новим ім'ям **work1**

Створення посилань

1) **Жорсткі посилання**

Кожен файл являє собою область даних на жорсткому диску (або на іншому носії інформації), яку можна знайти по імені. У файловій системі UNIX вміст файлу зв'язується з його ім'ям за допомогою жорстких посилань. Створення файлу за допомогою будь-якої програми означає, що буде створене жорстке посилання – ім'я файлу, і відкрита нова область даних на диску.

Причому кількість посилань на одну і ту ж область даних (файл) не обмежена, тобто файл може мати декілька імен.

Користувач може створити ще одне ім'я для файлу (тобто ще одне жорстке посилання на файл) за допомогою команди **`ln`** (від англ. link – з'єднувати, зв'язувати). Перший параметр цієї команди – це ім'я файлу, на який потрібно створити посилання, другий – ім'я нового посилання.

Приклад створення в поточному каталозі жорсткого посилання з ім'ям **new_name** на файл **/home/peter/docs/1.doc**:

```
ln /home/peter/docs/1.doc new_name
```

При створенні жорстких посилань у файлів та посилань на них збігаються розмір і час створення. Також, і файлу, і посиланню на нього відповідає один індексний дескриптор. Всі операції з файлової системою – створення, видалення, копіювання і переміщення файлів – виконуються насправді над індексними дескрипторами, а імена потрібні лише для того, щоб користувач міг легко орієнтуватися в файловій системі.

Доступ до одного і того ж файлу за допомогою кількох імен може знадобитися в наступних випадках:

– одна і та ж програма відома під кількома іменами;

– доступ користувачів до деяких каталогів в системі може бути обмежений з міркувань безпеки. Однак якщо все ж потрібно організувати доступ користувачів до файлу, який знаходиться в такому каталозі, можна створити жорстке посилання на цей файл в іншому каталозі;

– сучасні файлові системи можуть нараховувати до сотень тисяч файлів і каталогів. Зазвичай у таких файлових систем складна багаторівнева ієрархічна організація – в результаті шляхи до багатьох файлів стають дуже довгими. Щоб організувати більш зручний доступ до файлу, який знаходиться дуже «глибоко» в ієрархії каталогів, можна використовувати жорстке посилання в більш доступному каталозі;

– повне ім'я деяких програм може бути вельми довгим (наприклад, i586-alt-linux-gcc-3.3), і до таких програм зручніше звертатися за допомогою скороченого імені (жорсткого посилання) – gcc-3.3.

2) **Символьні (м'які) посилання**

Жорсткі посилання мають два суттєвих обмеження:

– жорстке посилання може вказувати лише на файл, але не на каталог, тому що в іншому випадку в файловій системі можуть виникнути цикли – нескінченні шляхи;

– неможливо створити на жорсткому диску жорстке посилання на файл, розташований на знімному носії.

Символьні посилання допомагають уникнути цих обмежень. Символьне посилання – це файл особливого типу, в якому міститься ім'я іншого файлу (каталогу). Символьні посилання, як і жорсткі, надають можливість звертатися до одного й того ж файлу (каталогу) за різними іменами. Символьні посилання називаються так тому, що містять символи – шлях до файлу або каталогу. Якщо на шляху до файлу зустрічається символьне посилання, система виконує підстановку: вихідний шлях замінюється тим, що міститься у посиланні. Якщо символьне посилання містить ім'я неіснуючого файлу, то воно не буде працювати.

Символьні посилання створюються за допомогою команди **`ln`** з прапором **`-s`**.

Приклад створення символьного посилання з ім'ям **mary_file** в домашньому каталозі користувача andrew на файл **mylist.lst**, який знаходиться в каталозі **lists** в домашньому каталозі користувача mary:

```
ln -s /home/mary/lists/mylist.lst /home/andrew/mary_file
```

Номер індексного дескриптора, розмір і час створення файлу і символьного посилання на нього різняться. Це свідчить про те, що сам файл і символьне посилання є різними файлами.

Видалення файлів і каталогів

Для видалення файлів призначена команда **`rm`** (від англ. remove – видаляти), аргументом якої є ім'я файлу (каталогу) або список імен файлів (каталогів) через пробіл. В іменах файлів і каталогів можна використовувати повні та відносні шляхи і шаблони. Команда `rm` призначена для видалення жорстких посилань, а не самих файлів. В UNIX, для того щоб повністю видалити файл, потрібно послідовно видалити всі жорсткі посилання на нього. При цьому всі жорсткі посилання на файл рівноправні – серед них немає «головного», зі зникненням якого зникне файл. Поки є хоч одне жорстке посилання, файл продовжує існувати. Втім, у більшості файлів в UNIX є лише одне ім'я (одне жорстке посилання на файл), тому команда `rm` *`ім'я_файлу/список_імен_файлів`* в більшості випадків успішно видаляє файл. Наприклад, команда

```
rm test
```

призведе до видалення файлу **test**. А команда

```
rm *.txt
```

призведе до видалення всіх файлів з розширеннями txt.

Деякі прапори команди `rm`:

`-i, --interactive` вимагає підтвердження при видаленні файлів

`-v, --verbose` друкує ім'я кожного файлу перед його видаленням

Для видалення каталогів призначена інша команда – **rmdir** (від англ. remove directory – видалити каталог), яка погодиться видалити каталог лише в тому випадку, якщо він порожній. Видалити каталог разом з усім його вмістом можна командою `rm` з ключем `-r` (`--recursive`).

Увага! Команда `rm -r` *`ім'я_каталога`* є дуже зручним способом втратити файли. Вона рекурсивно обходить весь каталог, видаляючи все, що в ньому міститься: файли, підкаталоги, символьні посилання. А ключ `-f` (`--force`) робить її роботу ще більш невідворотною, оскільки пригнічує запити виду «Видалити захищений від запису файл?», так що `rm` працює безмовно і безупинно.

Деякі прапори команди `rmdir`:

`-p, --parents`	після видалення зазначеного каталогу намагається видаляти кожен каталоговий компонент з повного маршруту
`-v, --verbose`	друкує ім'я кожного файлу перед його видаленням

Приклади:

`rmdir dir32`	видалення порожнього каталогу з ім'ям **dir32**
`rmdir tests temp`	видалення порожніх каталогів з іменами **tests** і **temp**
`rm -r /home/anna/tmp`	видалення каталогу **/home/anna/tmp** з його вмістом

Перегляд вмісту файлів

Виведення на екран вмісту файлу здійснюється декількома способами.

1) Команда **more** виводить на екран вміст одного або декількох зазначених файлів. Наприклад, для виведення на екран вмісту файлу **/etc/passwd** використовується команда

```
more /etc/passwd
```

Недолік цієї команди полягає в тому, що неможливо перегорнути інформацію в зворотному напрямку, а лише вперед (за допомогою клавіш [Пробіл] або [Enter]).

2) Команда **less** виводить на екран вміст одного або декількох зазначених файлів і дозволяє переглядати його в обох напрямках. Повернення до попередньої сторінки виконується після натискання клавіші [b] (Back).

3) Команда **cat** здійснює конкатенацію (об'єднання) вмісту зазначених файлів і виведення його на екран. Цю команду можна застосовувати для перегляду вмісту невеликих файлів. При спробі перегляду великих файлів, а тим більше об'єднання їх вмісту текст швидко промайне на екрані, тому для перегляду великих файлів краще користуватися командами `less` і `more`.

Деякі прапори команди `cat`:

-b, --number-nonblank	при виведенні вмісту файлів нумерує всі непорожні рядки, починаючи з 1
-n, --number	при виведенні вмісту файлів нумерує всі рядки, починаючи з 1
-s, --squeeze-blank	замінює послідовність порожніх рядків на один порожній рядок
-v, --show-nonprinting	показує недруковані символи (крім табуляцій, переводів рядків і переходів до нової сторінки). Управляючі символи зображуються у вигляді `^X` (CTRL+X); символ DEL – у вигляді `^?`. Символи, що не входять в набір ASCII, видаються у вигляді `M-`*x*, де *x* – символ, код якого визначається молодшими сімома бітами

З прапором `-v` можна використовувати наступні прапори:

-t	візуалізація символів табуляції у вигляді `^I`
-e	візуалізація символів переведення рядка у вигляді `$`

Якщо прапор `-v` не вказаний, то прапори `-t` і `-e` ігноруються.

Приклади використання команди `cat`:

`cat -vt f4 f5 f7`	виводить на екран вміст файлів **f4**, **f5** і **f7** з візуалізацією недрукованих символів
`cat -n f[12] > f3`	поміщає поєднаний вміст файлів **f1** і **f2** в файл **f3**, нумеруючипри цьому рядки
`cat f8 >> f6`	додає вміст файлу **f8** до вмісту файлу **f6**

Введення і виведення

Будь-яка програма – це автомат, призначений для обробки даних: отримуючи на вході одну інформацію, він в результаті роботи видає іншу (хоча вхідна та/або вихідна інформація може взагалі бути відсутня). Ті дані, які передаються програмі для обробки – це її введення, те, що вона видає в результаті роботи – виведення. Організація введення і виведення для кожної програми є завданням ОС.

Для того щоб записати дані в файл або прочитати їх звідти, процесу необхідно спочатку відкрити цей файл (при відкритті на запис, можливо, доведеться попередньо створити його). При цьому процес отримує дескриптор (описувач) відкритого файлу – унікальне для цього процесу число, яке він і буде використовувати у всіх операціях запису. Перший відкритий файл отримає дескриптор 0, другий – 1 і т.д. Закінчивши роботу з файлом, процес закриває його, при цьому дескриптор звільняється і може бути використаний повторно. Якщо

процес завершується, не закривши файли, за нього це робить система. Як «файли» використовуються звичайні файли, файли пристроїв (найчастіше – термінали), канали і сокети. Подальші операції – читання, запис і закриття – працюють з дескриптором, як з потоком даних, а куди саме веде цей потік, неважливо.

Кожен процес UNIX отримує при запуску три «файли», відкритих для нього системою. Перший з них (з дескриптором 0) відкритий на читання і називається **стандартним введенням** процесу (standard input, STDIN) – це потік даних, призначений для введення даних. Саме зі стандартним введенням працюють всі операції читання, якщо в них не вказаний дескриптор файлу. Другий «файл» (з дескриптором 1) відкритий на запис і називається **стандартним виведенням** процесу (standard output, STDOUT) – це потік даних, призначений для даних, що виводяться процесом. З ним працюють всі операції запису, якщо дескриптор файлу не вказаний в них явно. Нарешті, третій потік даних (з дескриптором 2) призначається для виведення діагностичних повідомлень і називається **стандартним виведенням помилок** (standard error, STDERR).

Оскільки ці три дескриптори вже відкриті до моменту запуску процесу, перший файл, відкритий самим процесом, буде, швидше за все, мати дескриптор 3.

Перенаправлення введення та виведення

В UNIX стандартне введення здійснюється з клавіатури, стандартне виведення направлене на термінал, на нього ж спрямоване і стандартне виведення помилок. Замість цього можна шляхом перепризначення записати вихідну інформацію будь-якої команди в файл, введення команді передати також з файлу, і помилки теж можна записати в файл. Для цього існують спеціальні оператори перенаправлення введення та виведення:

`> файл` перенаправлення виведення: помістити вихідну інформацію в файл, а не надсилати її на термінал; те, що знаходилося в файлі, буде знищено;

`>> файл` перенаправлення виведення: дописати вихідну інформацію в файл слідом за його поточним вмістом;

`< файл` перенаправлення введення: взяти вхідну інформацію з файлу, а не з клавіатури.

Приклади:

`ls -l >> filelist` список файлів поточного каталогу дописати в файл **filelist**

`cat f[1-3] > f4` вміст файлів **f1**, **f2** і **f3** помістити в файл **f4**

`> textfile` створення файлу з ім'ям **textfile**

`cat < mylist.txt` виведення на екран вмісту файлу **mylist.txt**. Це найпростіший приклад і не найкорисніший: більшість команд в UNIX як аргумент і так приймають ім'я файлу, без

позначення перенаправлення введення, так що можна було просто виконати команду `cat mylist.txt`. Але дуже важливо розуміти різницю: в наведеному прикладі команда `cat` *не отримала* і *не знає* ніяких вхідних файлів. Отже, вона очікує введення з клавіатури або вхідного потоку даних, який і надано їй за допомогою перенаправлення введення з файлу **mylist.txt**

Використання стандартного виведення помилок поряд зі стандартним виведенням дозволяє відокремити власне результат роботи програми від різноманітної супровідної інформації, наприклад, направивши їх в різні файли. Стандартне виведення помилок може бути перенаправлене так само, як і стандартне введення/виведення, для цього використовується комбінація символів `2 >` або `2 >>`. Наприклад, при виконанні команди

```
cat info 2 >> cat.err
```

в разі, якщо файл **info** не існує, помилка не буде видана на екран, а допишеться в файл **cat.err**.

Іноді, однак, потрібно об'єднати стандартне виведення і стандартне виведення помилок в одному файлі, а не розділяти їх. У командному інтерпретаторі для цього є спеціальна послідовність `2 > &1`, яка означає «направити стандартне виведення помилок туди ж, куди і стандартне виведення»:

```
cat info > info_out 2 >> &1
```

В даному прикладі спочатку вказано, куди перенаправити стандартне виведення (`>info_out`) і лише потім зазначено направити туди ж стандартне виведення помилок (`2 >> &1`). Якби було зазначено перенаправлення потоку помилок перед перенаправленням виведення (`2 > &1 > info_file`), в файл потрапило б лише стандартне виведення, а діагностичні повідомлення з'явилися б на терміналі. На момент здійснення операції `2 > &1` стандартне виведення було пов'язане з терміналом, значить, після її виконання стандартне виведення помилок теж буде пов'язане з терміналом. А подальше перенаправлення стандартного виведення в файл ніяк не відіб'ється на стандартному виведенні помилок.

Іноді наперед відомо, що якісь дані, виведені програмою, не знадобляться. Наприклад, попередження з стандартного виведення помилок. В цьому випадку можна перенаправити стандартне виведення помилок в файл порожнього пристрою – **/dev/null**. Все, що записується в цей файл, буде просто проігноровано:

```
cat info > info_out 2 > /dev/null
```

Аналогічно можна позбутися і стандартного виведення, відправивши його в **/dev/null**.

Конвеєр

Нерідко виникають ситуації, коли потрібно обробити виведення однієї програми якоюсь іншою програмою. Користуючись перенаправленням введення-виведення, можна зберегти виведення однієї програми в файлі, а потім направити цей файл на введення іншій програмі. Однак те ж саме можна зробити і більш ефективно: перенаправляти виведення не в файл, а безпосередньо на стандартне введення іншій програмі. У цьому випадку замість двох команд буде потрібна лише одна – програми будуть передавати одна одній дані «з рук в руки». В UNIX такий спосіб передачі даних називається **конвеєром**.

Для перенаправлення стандартного виведення однієї програми на стандартне введення іншої служить символ «|». Найпростіший і найпоширеніший випадок, коли потрібно використовувати конвеєр, виникає, якщо виведення програми не вміщується на екрані монітора і дуже швидко «пролітає» перед очима, так що користувач не встигає його прочитати. В цьому випадку можна направити виведення в програму перегляду `less`, яка дозволить перегорнути весь текст і повернутися до його початку:

```
cat employers.txt | less
```

Можна послідовно обробити дані кількома різними програмами, перенаправляючи виведення на введення наступній програмі і організовуючи який завгодно довгий конвеєр для обробки даних. В результаті можуть бути отримані командні рядки виду *`команда1 | команда2 | ... | командаN`*.

Організація конвеєра влаштована за тією ж схемою, що і перенаправлення в файл, але з використанням особливого об'єкта системи – каналу (від англ. pipe – труба, що негайно доставляє дані від входу до виходу). Каналом користуються відразу два процеси: один пише туди, інший читає. Пов'язуючи дві команди конвеєром, командний інтерпретатор відкриває канал (заводиться два дескриптори – вхідний і вихідний), підміняє стандартне виведення першого процесу на вхідний дескриптор каналу, а стандартне введення другого процесу – на вихідний дескриптор каналу, потім запускає по команді в цих процесах, і стандартне виведення першої команди потрапить на стандартне введення другої. Таким чином, канал – це неподільна пара дескрипторів (вхідний і вихідний), пов'язаних один з одним таким чином, що дані, записані у вхідний дескриптор, будуть негайно доступні на читання з вихідного дескриптора.

Деякі корисні команди

Команда **`echo`** – виводить на стандартне виведення зазначений рядок символів і здійснює перехід на новий рядок. Команда підтримує прапор `-n`, що дозволяє не виконувати перехід на новий рядок після виведення, і прапор `-e`, що дозволяє обробляти спеціальні управляючі послідовності:

`\`*`символ`*	вставка спецсимволу *символ* (наприклад, \\ – вставка зворотного слеша)
`\b`	видалення попереднього символу
`\c`	придушення подальшого виведення
`\e`	відображення символу Escape
`\f`	відображення символу подачі форми (переходу до наступної сторінки)
`\n`	вставка символу переходу на новий рядок
`\r`	вставка символу повернення каретки до початку рядка
`\t`	вставка символу горизонтальної табуляції
`\0`*`nnn`*	вставка ASCII-символу з вісімковим кодом *nnn*

Приклади:

`echo My name is John`	виводить на екран текст *My name is John*
`echo My name is John > f`	перенаправляє текст *My name is John* в файл **f**
`echo -e My name \n is John`	виводить текст на екран так, що *My name* розміщується на одному рядку, а *is John* – на наступному

Команда **`date`**, викликана без параметрів, виводить поточну дату/час.

Команда **`who`**, викликана без параметрів, виводить інформацію про користувачів, які в даний момент зареєструвалися в системі (по стовпцях: ім'я користувача, термінал, час реєстрації, ім'я комп'ютера).

Команда **`users`**, викликана без параметрів, виводить інформацію про користувачів, які в даний момент зареєструвалися в системі (лише їх імена в рядок).

Команда **`hostname`**, викликана без параметрів, виводить ім'я комп'ютера.

Команда **`uname`** виводить інформацію про комп'ютер і запущену на ньому ОС. Прапори:

`-a, --all`	виводить детальну інформацію у вигляді наступних стовпців: SYSNAME – ім'я ОС, NODENAME – ім'я комп'ютера, RELEASE – реліз ОС, OSVERSION – версія ОС, включаючи дату випуску, MACHINE – відомості про архітектуру комп'ютера.
`-m, --machine`	виводить тип архітектури комп'ютера
`-n, --nodename`	виводить ім'я комп'ютера
`-p, --processor`	виводить тип процесора даного комп'ютера
`-r, --release`	виводить реліз ОС
`-s, --sysname`	виводить найменування ОС
`-v`	виводить версію ОС

Команда **wc** (від англ. word count – підрахунок слів) використовується для підрахунку числа рядків, символів і слів в зазначених файлах або стандартному введенні, якщо ім'я файлу не зазначене або замість нього стоїть символ «-». Якщо зазначено більше одного файлу, виводяться імена цих файлів і значення лічильників, а в кінці виведення виводиться підсумкова сума накопичених лічильників.

Прапори:

-c, --chars	виводить кількість символів
-l, --lines	виводить кількість рядків
-w, --words	виводить кількість слів
-L, --max-line-length	виводить кількість символів в найдовшому рядку

За замовчуванням команда викликається з прапорами `-clw`.

Приклади:

`wc mytext > result`	виводить кількість символів, рядків і слів у файлі **mytext** і записує результат в файл **result**
`wc -cL ~/texts/mytext`	виводить кількість символів в файлі **~/texts/mytext** і кількість символів в найдовшому рядку цього файлу
`ls /dev | wc -l`	підраховує кількість файлів каталогу **/dev**

Команда **tee** одночасно копіює стандартне введення на стандартне виведення і в зазначений файл (файли). Якщо файл не існує, він буде створений. При існуванні файлу він буде перезаписаний, якщо тільки не вказаний прапор `-a` (`--append`) – додати стандартне виведення до вмісту зазначених файлів. Команда `tee` найчастіше використовується в конвеєрі. Вона корисна тоді, коли необхідно не лише відправити дані далі по конвеєру, а й зберегти їх копію.

Приклади:

`hostname | tee a b c d`	поміщає ім'я комп'ютера в файли **a**, **b**, **c** та **d** і виводить на екран
`cat text[135] | tee copy`	копіює об'єднаний вміст файлів **text1**, **text3** і **text5** в файл **copy** і виводить на екран

Зауваження про спеціальні символи

Деякі символи в командному інтерпретаторі мають спеціальне значення. Наприклад, шаблонні символи «`[]`», «`*`», «`?`», символ доступу до значення змінної «`$`», символи перенаправлення виведення і введення «`>`» і «`<`», символ конвеєра «`|`», пробіл, який використовується як роздільник. Іноді виникає необхідність скасувати їх спеціальне значення, наприклад, якщо необхідно використовувати один з цих символів в імені файлу. У цьому випадку таке ім'я слід укласти в

подвійні лапки, які скасовують дію всіх спецсимволів, крім «$» і «!». Наприклад, в результаті виконання команди

```
cp f* /tmp
```

в каталог **/tmp** будуть скопійовані всі файли з поточного каталогу, імена яких починаються з літери f. А при виконанні команди

```
cp "f*" /tmp
```

в каталог **/tmp** з поточного каталогу буде скопійований файл з ім'ям **f***.

Для скасування дії спеціального символу «$» рядок символів необхідно укласти в одинарні лапки. Наприклад, нехай введена команда

```
var='Hello world!'
```

Результатом роботи команди `echo $var` буде рядок *Hello world!*. Результатом роботи команди `echo "$var"` також буде рядок *Hello world!*. Результатом роботи команди `echo '$var'` буде рядок *$var*. А в результаті виконання команди `echo \$var` на екран буде виведений рядок *$var*.

Контрольні питання

1) Які типи файлів підтримуються в ОС UNIX?
2) У чому відмінності між «жорсткими» і «м'якими» посиланнями?
3) Яку структуру має файлова система ОС UNIX?
4) Для чого потрібні прапори (ключі) і аргументи командного рядка UNIX?
5) Як здійснюється перенаправлення введення/виведення в UNIX?
6) Які операції над файлами і каталогами в UNIX Вам відомі?

Завдання до виконання

1) Створити в домашньому каталозі каталоги з іменами **dir1**, **dir2**, **dir3** (рис. 1.2).

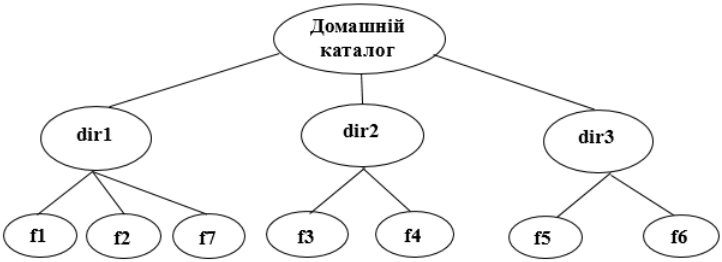

Рисунок 1.2 – Ієрархія файлів, що створюються при виконанні лабораторної роботи №1

2) В каталозі **dir1** створити файли з іменами **f1** і **f2**.

3) У файл **f1** помістити інформацію про ім'я комп'ютера, в файл **f2** – вміст каталогу **/dev** і всіх його підкаталогів. Переглянути вміст файлів **f1** і **f2**.

4) Скопіювати файл **f1** в каталог **dir2** з ім'ям **f3**.

5) В каталозі **dir2** створити жорстке посилання з ім'ям **hardlink** на файл **f2**, а також м'яке посилання з ім'ям **softlink** на файл **f3**. М'яке посилання видалити, а жорстке перейменувати в **f4**.

6) В каталозі **dir3** створити файл **f5**, в якому об'єднати інформацію, що зберігається в файлах **f1** і **f2**.

7) Оновити вміст файлу **f2**, помістивши туди замість раніше доданої інформації розширену інформацію про вміст каталогу **/var** і всіх його підкаталогів. Переглянути вміст файлу **f2** посторінково.

8) В каталозі **dir3** створити файл **f6**, що містить наступний текст: *File list*

9) Додати в файл **f6** поточну дату і вміст файлу **f2**. Переглянути його вміст.

10) В каталозі **dir1** створити файл **f7** для зберігання списку деяких файлів каталогу **/etc** в наступному вигляді:

Файли з іменами на букву р

[*список*]

Для формування списку файлів необхідно використати відповідну команду.

11) Файл **f7** скопіювати в неіснуючий каталог **~/dir4** з ім'ям **f8**, при цьому повідомити про помилки, які записати в файл **myerr**, що повинен знаходитися в каталозі **dir3**. Переглянути вміст файлу **myerr**. Повторити копіювання кілька разів, при цьому в файлі **myerr** число записів має дорівнювати числу спроб копіювання.

12) Переглянути вміст неіснуючого каталогу **~/dir4**, не виводячи при цьому повідомлення про помилку.

13) Вміст файлу **f1** скопіювати в файли **f6** і **f7** таким чином, щоб при виконанні команди вміст файлу **f1** був відображений на екрані.

14) Видалити файли **f3** і **f4**. Видалити каталог **dir2**. Видалити каталог **dir1** рекурсивно.

15) Вивести на екран список файлів каталогу **/bin**, імена яких містять цифри.

16) Підрахувати кількість файлів каталогу **/bin**, імена яких не починаються з літер f і p.

17) Записати в файл **spisok**, що знаходиться у вашому домашньому каталозі, імена файлів каталогу **/bin**, що не починаються з голосної букви і закінчуються шістнадцятковою цифрою.

18) Скопіювати в каталог **dir3** файли каталогу **/bin**, імена яких складаються

з 3 або 4 символів.

19) Підрахувати кількість слів та символів у вмісті файлів каталогу **/etc**, імена яких складаються мінімум з 3 символів, другим символом є літера від k до s або a.

ЛАБОРАТОРНА РОБОТА №2.
УПРАВЛІННЯ ПРАВАМИ ДОСТУПУ КОРИСТУВАЧІВ ДО ФАЙЛІВ І КАТАЛОГІВ

Мета роботи: вивчити основні і спеціальні права доступу у файловій системі UNIX (Linux), навчитися змінювати права доступу користувачів ОС до файлів і каталогів.

Постановка задачі: виконати завдання, використовуючи відповідні команди ОС UNIX (Linux).

Теоретичні відомості

Користувачі системи

Перш ніж ОС буде готова до роботи з користувачем, відбувається процедура завантаження системи, що завершується виведенням на екран запрошення до реєстрації в системі «login:». Процедура реєстрації в ОС для UNIX є обов'язковою. Для кожного користувача визначена сфера його повноважень в системі: програми, які він може запускати, файли, які він має право переглядати, змінювати або видаляти. При спробі виконання операції, що виходить за рамки повноважень, користувач отримає повідомлення про помилку.

Багатокористувацька модель розмежування доступу, що використовується в UNIX, дозволяє вирішити ряд задач, вельми актуальних і для сучасних персональних комп'ютерів, і для серверів, що працюють в локальних і глобальних мережах, і взагалі в будь-яких системах, що одночасно виконують різні задачі, за які відповідають різні люди. У багатокористувацькій моделі виділяються звичайні користувачі та адміністратори. У повноваження звичайного користувача входить все необхідне для виконання прикладних задач, проте йому заборонено виконувати дії, що змінюють саму систему. Таким чином можна уникнути пошкодження системи в результаті помилки користувача (натиснув не ту кнопку), помилки в програмі або в результаті роботи програми-вірусу. Повноваження адміністратора зазвичай не обмежені.

Для персонального комп'ютера, з яким працюють кілька людей, важливо забезпечити кожному користувачеві незалежне робоче середовище. Це знижує ймовірність випадкового пошкодження чужих даних, а також дозволяє кожному

користувачеві налаштувати зовнішній вигляд робочого середовища на власний розсуд. В багатокористувацькій моделі ця задача вирішується наступним чином: організовується домашній каталог, де зберігаються дані користувача, налаштування зовнішнього вигляду і поведінки його системи і т. п., а доступ інших користувачів до цього каталогу обмежується. Якщо комп'ютер підключений до глобальної або локальної мережі, то цілком ймовірно, що якусь частину ресурсів, що на ньому зберігаються, має сенс зробити публічною і доступною по мережі. І навпаки, частину даних, наприклад, особисте листування, робити публічними не слід. За допомогою обмеження доступу користувачів до персональних даних один одного, можна вирішити і цю задачу.

Облікові записи

ОС може бути «знайома» з людиною лише в переносному сенсі: в ній повинен зберігатися запис про користувача з даною назвою і про пов'язану з ним системну інформацію – **обліковий запис** (від англ. account – облік). Саме з обліковими записами, а не з самими користувачами, працює ОС. Насправді, співвідношення облікових записів і користувачів в UNIX зазвичай не є однозначним: кілька людей можуть використовувати один обліковий запис. І в той же час в UNIX є облікові записи для псевдокористувачів, від імені яких працюють деякі програми, але не люди.

Облікові записи можуть бути створені під час установки системи або після неї.

У обліковому записі містяться наступні відомості про користувача.

– Реєстраційне (або вхідне) ім'я – назва облікового запису користувача, яку потрібно вводити при реєстрації в системі.

– Ідентифікатор користувача (UID – User IDentifier) – унікальне невід'ємне ціле число, що однозначно ідентифікує обліковий запис користувача в UNIX. Таким числом забезпечені всі процеси UNIX і всі об'єкти файлової системи. Використовується для персонального обліку дій користувача і визначення прав доступу до інших об'єктів системи. UID не може бути довільним. Наприклад, в ОС Linux є деякі угоди щодо того, якому типу користувачів можуть бути видані ідентифікатори з того чи іншого діапазону. Зокрема, UID від 0 до 100 використовуються для зарезервованих і системних користувачів, а також для псевдокористувачів.

– Ідентифікатор групи (GID – Group IDentifier) – унікальне позитивне ціле число, яке ідентифікує приналежність користувача до групи. Групи користувачів застосовуються для організації доступу декількох користувачів до деяких ресурсів. В UNIX користувач повинен належати як мінімум до однієї групи – групи за замовчуванням (або основної групи). У Linux, наприклад, під час створення облікового запису користувача зазвичай створюється і група, ім'я якої збігається з

вхідним ім'ям, саме ця група буде використовуватися як група за замовчуванням для даного користувача. Користувач може входити більше ніж в одну групу, але в обліковому записі вказується лише номер групи за замовчуванням.

– Повне ім'я (ім'я та прізвище) людини, яка використовує даний обліковий запис. Ця інформація необхідна не стільки системі, скільки людям, щоб мати можливість визначити, кому належить обліковий запис.

– Домашній каталог користувача, в якому він може зберігати свої дані. Зазвичай він має шляхове ім'я **/home/*ім'я_користувача***. Доступ інших користувачів до домашнього каталогу даного користувача обмежується. Користувач, що реєструється в системі, починає роботу з домашнього каталогу.

– Командний інтерпретатор, що повинен бути запущений для кожного зареєстрованого в системі користувача. Оскільки в UNIX є декілька різних командних інтерпретаторів, в обліковому записі вказується, який із них необхідно запустити для даного користувача. Якщо не вказувати командний інтерпретатор при створенні облікового запису, він буде призначений за замовчуванням, найімовірніше це буде bash.

За допомогою команди `id` *`ім'я_користувача`* можна дізнатися реєстраційне ім'я користувача і відповідний йому UID, а також назву групи за замовчуванням, її GID і повний список груп, членом яких є вказаний користувач.

В UNIX адміністратором системи є користувач з ідентифікатором 0. Зазвичай обліковий запис користувача з UID, рівним 0, називається root (від англ. «корінь»). Обліковий запис для root обов'язково присутній в будь-якій системі UNIX, навіть якщо в ній немає ніяких інших облікових записів. Користувачеві з UID, рівним 0, дозволено виконувати будь-які дії в системі, а значить, будь-яка помилка або неправильна дія може пошкодити систему або знищити дані. Тому не рекомендується реєструватися в системі під ім'ям root для повсякденної роботи. Працювати в root слід лише тоді, коли це дійсно необхідно: при налаштуванні та оновленні системи або при відновленні після збоїв.

Саме привілейований користувач root володіє достатніми повноваженнями для створення нових облікових записів.

Файли, що містять відомості про користувачів і групи

В UNIX облікові записи зберігаються в файлі **/etc/passwd**, що доступний для читання всім користувачам. Паролі зберігаються в зашифрованому вигляді і можуть бути «заховані», тобто можуть зберігатися окремо в файлах, недоступних для загального читання – це називається механізмом тіньових або прихованих паролів. Файл, в якому зберігаються паролі, залежить від версії UNIX. У Linux паролі зазвичай зберігаються в файлі **/etc/shadow**. Інформація про приналежність користувачів до груп зберігається в файлі **/etc/group**.

Кожен рядок файлу **/etc/passwd** описує одного користувача і містить 7 полів, розділених двокрапкою.

– Реєстраційне ім'я (логін).

– Зашифрований пароль, якщо не використовується механізм тіньових паролів. При використанні цього механізму в полі пароля знаходиться символ `x`. Дане поле може бути порожнім, і це означає, що користувач може входити в систему без пароля. Символ `*` показує, що вхід для користувача заблокований. Також в даному полі може стояти посилання на логін іншого користувача у вигляді *##логін_іншого_користувача*, і це означає, що для входу в систему даний користувач повинен використовувати пароль користувача, зазначеного у посиланні.

– Ідентифікатор користувача.

– Ідентифікатор групи за замовчуванням (основної групи).

– Повне ім'я користувача (насправді тут можна зберігати не лише його повне ім'я, а й довільну інформацію про нього, наприклад дані про посаду та номер телефону).

– Домашній каталог.

– Командний інтерпретатор за замовчуванням. Якщо тут знаходиться значення **/bin/false** або **/sbin/nologin**, це означає, що користувачеві заборонено входити в систему.

Приклади записів в файлі **/etc/passwd**:

```
root:x:0:0:root:/root:/bin/bash
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/false
john:x:500:500::/home/john:/bin/bash
bill:##john:501:501::/home/bill:/bin/sh
```

Файл **/etc/group** описує групи користувачів. Кожен рядок в цьому файлі представляє одну групу і містить 4 поля, розділених двокрапкою:

– ім'я групи;

– зашифрований пароль (застаріле, рідко використовуване поле);

– ідентифікатор групи;

– список членів групи, перерахованих через кому (у списку членів групи не вказуються ті користувачі, для яких дана група є основною).

Приклади записів в файлі **/etc/group**:

```
root:x:0:root,admin
sys:x:3:root,bin,adm
john:x:500:
```

Права доступу в файловій системі UNIX

Варто зауважити, що в дійсності маніпулює файлами не сам користувач, а запущений ним процес (наприклад, команда `rm` або `cat`). Кожен процес системи

обов'язково належить якомусь користувачу, і ідентифікатор користувача (UID) – обов'язкова властивість будь-якого процесу UNIX. Коли при реєстрації в системі запускається командний інтерпретатор, ОС приписує йому UID, отриманий в результаті ідентифікації користувача. Всі процеси, запущені користувачем під час роботи, матимуть його UID.

Ключ `-l` команди `ls` визначає формат видачі інформації про файли (справа наліво): ім'я файлу, час останньої зміни файлу, розмір в байтах, група-власник, користувач-власник, кількість жорстких посилань, права доступу і тип файлу (перший символ першого стовпчика). Наприклад:

```
drwxr-xr-x  2  work  work  4096   Окт 31 15:21 examples
-rw-r--r--  1  work  work   26    Сен 22 15:21 loop
-r--------  1  root  root    0    Сен 10 02:08 /etc/shadow
```

Тип «-» відповідає звичайному файлу, тип `d` – каталогу, тип `l` – символьному посиланню, тип `s` – сокету, тип `p` – іменованому каналу, тип `c` – файлу символьного пристрою, тип `b` – файлу блокового пристрою.

Наступні дев'ять символів мають вигляд `rwxrwxrwx`, де деякі `r`, `w` та `x` можуть замінюватися на «-». Очевидно, букви відображають прийняті в UNIX три види доступу – читання (read), запис (write) і виконання (execute). У рядку атрибутів вони присутні в трьох примірниках, тому що будь-який користувач (процес) UNIX по відношенню до будь-якого файлу може виступати в трьох ролях: як **власник**, як **член групи**, якій належить файл (при цьому власник-користувач може не бути членом групи-власника), і як **сторонній користувач**, який ніяких відносин власності на цей файл не має. Права доступу – це три трійки `rwx`, що описують права доступу до файлу власника цього файлу (перша трійка), групи-власника (друга трійка) і сторонніх користувачів (третя трійка). Якщо в якій-небудь трійці не вистачає букви, а замість неї стоїть «-», значить, користувачу у відповідній ролі у відповідному виді доступу буде відмовлено.

При з'ясуванні відносин між файлом і користувачем, який запустив процес для доступу до файлу (наприклад, команду `rm`), роль користувача визначається так:

1) якщо UID файлу збігається з UID процесу, користувач є власником файлу, і на процес поширюються права доступу власника файлу, а всі інші права ігноруються;

2) в разі розбіжності UID файлу з UID процесу, якщо GID файлу збігається з GID будь-якої групи, в яку входить користувач, він є членом групи, якій належить файл, і на процес поширюються права доступу для групи-власника файлу, а всі інші права ігноруються;

3) якщо і UID файлу не збігається з UID процесу, і GID файлу не входить до списку груп, членом яких є користувач, то користувач є стороннім, і на процес поширюються права доступу для сторонніх користувачів.

Права власника і групи не підсумовуються. Якщо UID файлу збігається з UID процесу і GID файлу збігається з GID будь-якої групи, в яку входить користувач, працюють лише права власника файлу.

Права доступу зберігаються в 16-бітовому полі індексного дескриптора файлу. 4 старші біти визначають тип файлу, який встановлюється при створенні файлу і не підлягає зміні, інші біти відведені під зберігання режиму доступу (рис. 2.1).

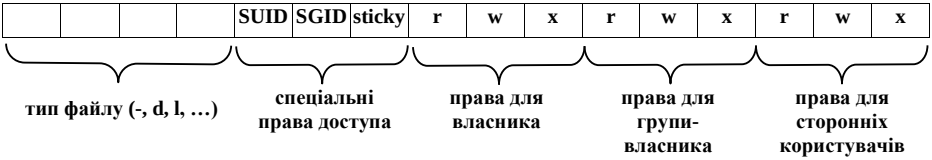

Рисунок 2.1 – Поле індексного дескриптора файлу, де зберігаються права доступу

Основні права доступу до файлів і каталогів розрізняються (див. табл. 2.1).

Таблиця 2.1 – Основні права доступу до файлів і каталогів

Права доступу	Файл	Каталог
`r` (читання)	читання вмісту файлу	отримання списку імен файлів каталогу
`w` (запис)	запис у файл	зміна вмісту каталогу (створення, перейменування і видалення файлів в ньому)
`x` (виконання)	виконання файлу, якщо він є сценарієм або програмою	вхід до каталогу та отримання властивостей файлів, що містяться в ньому

Існує два типи виконуваних файлів: бінарні файли, які виконуються безпосередньо процесором, і сценарії, оброблювані командним інтерпретатором або якою-небудь іншою аналогічною програмою. Сценарії зазвичай починаються з рядка, в якому задається відповідний інтерпретатор, наприклад, `#!/bin/bash`, і містять команди, які послідовно виконуються даним командним інтерпретатором.

Дозволи, встановлені для каталогу, мають більш високий пріоритет, ніж дозволи, встановлені для файлів цього каталогу.

Далі розглянемо спеціальні права доступу.

Ідея, що лежить в основі застосування біта **SUID** (Set User ID – встановити ідентифікатор користувача), полягає в тому, що користувач, який запустив виконуваний файл, для якого власник встановив біт SUID, на час виконання цього файлу отримує всі права його власника. Наприклад, якщо привілейований користувач створив сценарій і встановив для нього біт SUID, а звичайний користувач запускає цей сценарій, то повноваження суперкористувача на час

виконання сценарію переходять до звичайного користувача. Той же принцип застосовується і до біту **SGID** (Set Group ID – встановити ідентифікатор групи), тільки в даному випадку змінюються привілеї групи, що володіє сценарієм. Біти підміни ідентифікаторів користувача і групи несуть потенційну загрозу безпеці системи і повинні використовуватися з обережністю.

Біт збереження завдання **sticky** (від англ. «липучка») при установці для файлу вказує системі на необхідність зберегти копію програми в пам'яті після її завершення для прискорення наступних запусків цієї програми. sticky був необхідний на старих моделях комп'ютерів. На сучасних швидкодіючих системах він практично не використовується. Якщо ж sticky встановлений для каталогу, то в цьому каталозі файли може видаляти або перейменовувати лише власник каталогу, власник файлів або привілейований користувач, навіть якщо член групи має ті ж права, що і власник файлів.

Зміна прав доступу

Змінювати права доступу до файлів і каталогів може їх власник або привілейований користувач. Для зміни прав доступу до файлів і каталогів застосовується команда **`chmod`** (від англ. change mode – змінити режим). Аргумент цієї команди, що позначає встановлювані права доступу, може бути заданий або в символьному, або в числовому (абсолютному) форматі.

Синтаксис команди `chmod` при використанні символьного формату:

```
chmod [хто] оператор права ім'я_файлу
```

Параметр *`хто`* може приймати наступні значення:

`u` – власник;

`g` – група;

`o` – сторонні користувачі;

`a` – всі користувачі.

Параметр *`оператор`* може приймати наступні значення:

`+` – додавання прав до наявних

`-` – видалення прав з наявних

`=` – установка прав замість наявних

Параметр *`права`* може приймати наступні значення:

`r` – читання;

`w` – запис;

`x` – виконання;

`s` – SUID або SGID (при використанні разом з `u` мається на увазі SUID, при використанні з `g` – SGID);

`t` – sticky;

`u` – установка тих же прав, що і у власника;

`g` – установка тих же прав, що і у групи;

`o` – установка тих же прав, що і у сторонніх користувачів.

У наступних прикладах перша колонка демонструє команду зміни прав доступу, а в другій колонці наведений символьний запис прав доступу після виконання команди. Також передбачається, що для файлу **myfile** встановлені права на читання, запис і виконання для всіх категорій користувачів, і їх символьний запис має вигляд `rwxrwxrwx`.

`chmod a-x myfile`	`rw-rw-rw-`	відібрати у всіх категорій користувачів право на виконання
`chmod og-w myfile`	`rw-r--r--`	відібрати у групи і сторонніх користувачів право на запис
`chmod g+w myfile`	`rw-rw-r--`	додати право на запис для групи
`chmod u+x myfile`	`rwxr--r--`	додати право на виконання для власника
`chmod g+wx myfile`	`rwxrwxr--`	додати права на запис і виконання для групи
`chmod g=o myfile`	`rwxr--r--`	встановити для групи такі ж права доступу, як у сторонніх користувачів
`chmod g+x,o-r myfile`	`rwxr-x---`	додати право на виконання для групи і відібрати право на читання для сторонніх користувачів
`chmod g+s myfile`	`rwxr-s---`	додати SGID
`chmod u-x myfile`	`rw-r-s---`	відібрати у власника право на виконання
`chmod u+s myfile`	`rwSr-s---`	додати SUID
`chmod g-s,o+r myfile`	`rwSr-xr--`	зняти SGID і додати стороннім користувачам право на читання
`chmod –r myfile`	`-wS--x---`	відібрати у всіх категорій користувачів право на читання
`chmod +t myfile`	`-wS--x--T`	встановити sticky
`chmod uo+x myfile`	`-ws--x--t`	додати для власника і сторонніх користувачів право на виконання

Синтаксис команди `chmod` при використанні числового формату:

`chmod` *`права ім'я_файлу`*

Тут параметр *`права`* є вісімковим числом. У найпростішому випадку воно складається з трьох трьохбітових наборів, кожен з яких відноситься до однієї з

категорій користувачів. Старший біт відповідає дозволу на читання, середній – дозволу на запис, а молодший – дозволу на виконання. Значення 1 означає, що право встановлено, 0 – знято (див. табл. 2.2).

Таблиця 2.2 – Основні права доступу у вісімковому форматі

Вісімкова цифра	r w x	Символьний запис прав доступу
0	0 0 0	`---`
1	0 0 1	`--x`
2	0 1 0	`-w-`
3	0 1 1	`-wx`
4	1 0 0	`r--`
5	1 0 1	`r-x`
6	1 1 0	`rw-`
7	1 1 1	`rwx`

Наприклад, записані в символьній формі права доступу `rwxrwxrwx` мають числовий еквівалент 777, а права `rw-r-x---` мають числовий еквівалент 650.

При установці/знятті спеціальних прав доступу зліва додається ще один трьохбітовий набір (див. табл. 2.3).

Наприклад, записані в символьній формі права доступу `rw-r-s--x` мають числовий еквівалент 2651, права `rwSr-s-w-` мають числовий еквівалент 6652, права `r-xr-s-wT` мають числовий еквівалент 3552, а права `rw-r-S-wt` мають числовий еквівалент 3643.

Таблиця 2.3 – Спеціальні права доступу у вісімковому форматі

Вісімкова цифра	SUID	SGID	sticky	Символьний запис прав доступу
0	0	0	0	`---------`
1	0	0	1	`--------t` (при встановленому x для сторонніх користувачів) `--------T` (`x` для сторонніх користувачів відсутній)
2	0	1	0	`-----s---` (при встановленому `x` для групи) `-----S---` (`x` для групи відсутній)
3	0	1	1	Встановлено SGID і sticky
4	1	0	0	`--s------` (при встановленому `x` для власника) `--S------` (`x` для власника відсутній)
5	1	0	1	Встановлено SUID і sticky
6	1	1	0	Встановлено SGID і SGID
7	1	1	1	Встановлено SGID, SGID і sticky

Приклади (перший стовпець демонструє команду зміни прав доступу, а в другому наведений символьний запис прав доступу після виконання команди):

`chmod 504 myfile`	`r-x---r--`	власник має права читання і виконання, група не має ніяких прав, сторонні користувачі мають право читання
`chmod 4755 myfile`	`rwsr-xr-x`	для файлу встановлений SUID, власник має права читання, запису та виконання, група і сторонні користувачі мають права читання і виконання
`chmod 6701 myfile`	`rws--S--x`	для файлу встановлено SUID і SGID, власник має права читання, запису та виконання, група не має ніяких прав, сторонні користувачі мають право виконання
`chmod 5765 myfile`	`rwsrw-r-t`	для файлу встановлено SUID і sticky, власник має права читання, запису та виконання, група має права читання і запису, сторонні користувачі мають права читання і виконання

Права доступу за замовчуванням

При створенні нових файлів і каталогів встановлюються наступні права доступу за замовчуванням: для каталогів – 777 (`rwxrwxrwx`), для файлів – 666 (`rw-rw-rw-`).

Права доступу за замовчуванням можна регулювати за допомогою команди **`umask`** (від англ. user file creation mode mask – користувацька маска створення файлів). **Користувацька маска** – це число, яке віднімається від прав доступу за замовчуванням, а отримані в результаті віднімання права застосовуються до створюваного файлу (каталогу). Єдиний параметр команди `umask` – вісімкове число, що задає права доступу, які *не потрібно* встановлювати для нового файлу або каталогу. Подана без параметрів, ця команда виводить значення поточної встановленої маски.

Права доступу файлів, створених при конкретному значенні `umask`, обчислюються за допомогою побітового І між правами доступу та унарним доповненням маски (тобто аргументу команди `umask`).

Наприклад:

```
umask 027
```

Для каталогів обчислити отримані права просто: права доступу за замовчуванням (777_8) мінус значення маски (027_8) дорівнює 750_8.

Для файлів ця процедура трохи складніша. Спочатку напишемо двійкове подання вісімкового значення маски: $027_8 = 000010111_2$ та прав доступу за замовчуванням: $666_8 = 110110110_2$. Потім напишемо унарне доповнення значення маски: 111101000_2 (для його отримання необхідно інвертувати біти двійкового подання значення маски). А тепер виконуємо побітове множення прав доступу та унарного доповнення значення маски:

```
1 1 0 1 1 0 1 1 0
1 1 1 1 0 1 0 0 0
-----------------
1 1 0 1 0 0 0 0 0
```

Переводимо результат в вісімкову систему і отримуємо права доступу 640 або, в символьному представленні, `rw-r-----`.

Приклади:

`umask 0` файли будуть створюватися з правами доступу `rw-rw-rw-` (666), каталоги – з правами доступу `rwxrwxrwx` (777)

`umask 022` файли будуть створюватися з правами доступу `rw-r--r--` (644), каталоги – з правами доступу `rwxr-xr-x` (755)

`umask 077` файли будуть створюватися з правами доступу `rw-------` (600), каталоги – з правами доступу `rwx------` (700)

Контрольні питання

1) У чому полягає багатокористувацька модель розмежування доступу?
2) Які основні права доступу до файлів і каталогів Вам відомі?
3) До чого призводить встановлення біта SUID (SGID)?
4) Навіщо потрібна команда `umask`?
5) Як змінити права доступу на файли і каталоги?
6) Чому основні права доступу записуються у трьох екземплярах?

Завдання до виконання

1) У домашньому каталозі створити каталоги **dir1** і **dir2**. Зробити **dir1** загальнодоступним (всім користувачам можна виконувати в цьому каталозі будь-які операції).

2) У каталозі **dir1** створити підкаталог **dir11** і вкладений в нього каталог **dir111**. Зробити **dir111** темним (користувачі всіх категорій повинні мати можливість створювати файли, перейменовувати та видаляти *лише свої* файли і не бачити, що є в цьому каталозі).

3) Створити в каталог **dir11** файли з іменами **f1** і **f2**. За допомогою числового аргументу команди `chmod` призначити їм наступні права доступу:

– **f1**: групі – читання, виконання, стороннім користувачам – нічого, власнику – всі права;

– **f2**: групі та стороннім користувачам – виконання, власнику – читання.

4) У каталогах **dir1** і **dir2** створити файли з однаковими іменами: **a454**, **a\a**, **\4a**, **44\?**, **F4a41**, **f442**, ??**12**, **abcdf**, **1**, **4**, **a**, **?**, **\b**, **44**.

5) Прибрати право запису для всіх категорій користувачів на файли каталогу **dir1**, імена яких містять не менше трьох символів.

6) Прибрати всі права доступу для сторонніх користувачів на файли каталогу **dir1**, імена яких містять цифри.

7) Додати права запису і виконання для власника на всі файли каталогів **dir1** і **dir2**, імена яких містять не менше двох цифр і не закінчуються буквою.

8) Додати право виконання для групи і сторонніх користувачів на всі файли каталогів **dir1** і **dir2**, імена яких містять символ \.

9) Зробити закритим ваш домашній каталог (власнику можна виконувати всі дії, іншим категоріям користувачів ніяких дій виконувати не можна).

10) За встановленою маскою і значенням прав, отриманих при створенні файлів і каталогів, визначити, які права встановлені в системі за замовчуванням.

11) Задати маску, що дає можливість створювати файли і каталоги, доступні для всіх операцій лише власнику і стороннім користувачам, при цьому члени групи-власника не повинні мати ніяких прав. Перевірити дію маски.

12) Встановити маску, значення якої було отримано при виконанні завдання 10.

ЛАБОРАТОРНА РОБОТА №3.
СТВОРЕННЯ РЕЗЕРВНИХ КОПІЙ ТА АРХІВІВ

Мета роботи: вивчити основні програми архівації, стиснення і розпаковування файлів та отримати практичні навички створення архівних файлів.

Постановка задачі: виконати завдання, використовуючи відповідні команди ОС UNIX (Linux).

Теоретичні відомості

У наші дні ми зустрічаємося з файлами архівів дуже часто: у вигляді архівних файлів зберігаються резервні копії розділів дисків або баз даних, часто створюються архіви для передачі кількох файлів по електронній пошті або для їх завантаження на файлообмінник, у вигляді архівів поширюються пакети програм і вихідні коди.

Важливо відзначити, що архівація в UNIX – це не одне і те ж, що стиснення файлів. Архівація – це об'єднання декількох файлів в один з метою більш зручної подальшої їх передачі, зберігання, шифрування або стиснення.

Основним засобом архівації в UNIX (а, отже, і в Linux) є комплекс з двох програм – `tar` і `gzip`.

Програма архівації файлів tar

Програма **`tar`** (від англ. tape archiver – архіватор на стрічці, оскільки спочатку ця програма призначалася для створення архівів саме на магнітній стрічці) використовується для архівації файлів і витягання їх з архіву. При архівації декілька файлів об'єднуються в один файл-архів. Програма `tar` дозволяє створювати архіви на жорсткому диску і на знімних носіях.

Команда для виконання програми `tar` має наступний синтаксис:

`tar` *`прапори ім'я_архіва`* [*`додаткові_прапори`*] *`ім'я_файлу/каталогу`*

Першим в командному рядку повинен йти один з прапорів, що позначають дію, яка виконується командою: `-A`, `-c`, `-d`, `-r`, `-t`, `-u`, `-x`. Далі йдуть необов'язкові прапори, що позначають спосіб виконання дії: `-w`, `-z`, `-v`. За ними слідує прапор `-f`, який вказує, що далі йде ім'я архівного файлу, над яким виконується дія. При виконанні архівації файлів на диску прапор `-f` необхідно зазначати обов'язково, тому що, в силу історичних причин, за замовчуванням програма виконує архівацію файлів на магнітну стрічку, з якою пов'язаний файл пристрою **/dev/rmt0**. Після імені архівного файлу можуть йти додаткові прапори.

При зазначенні в якості імені архіву символу «-» запис даних здійснюється на стандартне виведення, що дозволяє використовувати команду `tar` в конвеєрах.

Останніми в командному рядку вказуються одне або більше імен файлів або каталогів, які необхідно помістити в архів. При зазначенні імені каталогу передбачається, що всі його підкаталоги будуть включені в архів.

Прапори команди `tar`, що позначають дію:

`-A, --concatenate`	додає файли в кінець архіву
`-c, --create`	створює новий архів
`-d, --compare`	виявляє відмінність між членами архіву і їх вихідними файлами на диску
`-r, --append`	додає файли в архів, при цьому не має значення, чи існують вже зазначені файли серед файлів архіву. Якщо в архів додається файл з ім'ям, яке вже має один з файлів архіву, старий файл архіву не знищується. Щоб записати нові версії членів архіву в архів, необхідно використовувати операцію `-u` (`--update`)
`-t, --list`	виводить вміст архіву
`-u, --update`	додає до архіву лише ті файли, яких немає в архіві, або змінені файли

-x, --extract витягає файли з архіву

Прапори команди `tar`, що позначають спосіб виконання дії:

-v, --verbose	виводить список оброблюваних файлів
-w, --interactive	запитує підтвердження для кожної дії
-z, --gzip	стискає/розпаковує файли за допомогою програми `gzip` (див. нижче)
-T, --files-from *file*	вказує взяти імена оброблюваних файлів з файлу *file*
-W, --verify	перевіряє цілісність архіву після його створення

Додаткові прапори команди `tar`:

--delete	видаляє файли з архіву
--remove-files	видаляє оригінальні файли після включення в архів
--exclude file	виключає з обробки файл *file*
-X *file*, --exclude-from *file*	виключає з обробки файли, перераховані у файлі *file*
-C *dir*, --directory *dir*	розпаковує файли в зазначений каталог *dir*
--wildcards	дозволяє використовувати в іменах файлів шаблони підстановки

Приклади:

`tar -cvWf arc/bin.tar /bin`	архівує каталог **/bin** в файл **bin.tar**, який знаходиться в підкаталозі **arc** поточного каталогу, при цьому виводить імена оброблюваних файлів і перевіряє цілісність архіву після його створення
`tar -czwf docs.tar a1 b2 c3`	поміщає в архів **docs.tar** файли поточного каталогу **a1**, **b2** і **c3** зі стисненням за допомогою `gzip` і підтвердженням кожної виконуваної дії
`tar -tf sample.tar`	перегляд вмісту архівного файлу **sample.tar**
`tar -cf myarc.tar ~/documents --exclude ~/documents/file.txt --exclude /documents/doc.txt`	архівує каталог **~/documents** в файл **myarc.tar**, виключаючи з обробки файли **~/documents/file.txt** і **~/documents/doc.txt**
`tar -xf example.tar -C /tmp`	розпаковує архів example.tar в каталог **/tmp**
`tar -cf file.tar --wildcards a*m`	створює архів **file.tar** з файлів поточного каталогу, імена яких починаються з літери a і закінчуються літерою m

Програма стиснення (упаковки) файлів gzip

Програми стиснення, використовуючи спеціальні алгоритми кодування даних, дозволяють записувати вміст файлів в більш компактному вигляді. Однією з найпопулярніших програм в UNIX для виконання стиснення є програма **gzip**. Формат команди для виконання програми `gzip`:

`gzip [`*`прапори`*`] `*`ім'я_файлу/каталогу`*

При виконанні стиснення кожен файл заміщається стислою версією з таким же ім'ям, як у оригінального файлу, до якого додається розширення .gz. При цьому зберігаються дані про власника файлу, групу-власника, права доступу, а також часові мітки оригінального файлу. Якщо імена файлів не вказані або замість імені файлу стоїть символ «-», вміст стандартного введення стискається і пересилається на стандартне виведення. Програма `gzip` стискає лише звичайні файли, ігноруючи символьні посилання.

Прапори команди `gzip`:

-c, --stdout	спрямовує результат обробки на стандартне виведення, зберігаючи оригінальні файли незміненими
-d, --decompress	виконує розпакування файлів
-f, --force	перезаписує вихідні файли і стискає посилання
-l, - list	виводить інформацію про розмір стисненого і оригінального файлу, відсоток стиснення і т. ін.
-r, --recursive	рекурсивно виконує обробку файлів у всіх каталогах
-S, --suffix *suffix*	вказує замість .gz використовувати розширення *suffix*
-t, --test	перевіряє цілісність стиснених файлів
-v, --verbose	відображає в процесі обробки імена файлів і відсоток стиснення
-1 ...- 9	вказує ступінь стиснення (-1 - мінімальне, найбільш швидке, -9 – максимальне, найбільш повільне); за замовчуванням прийнята ступінь стиснення -6

Приклади:

`gzip –rv /home`	стиснути рекурсивно всі файли каталогу **/home**, при цьому вивести імена оброблюваних файлів
`gzip –d arch/a*.gz`	розпакувати файли каталогу **arch**, імена яких починаються з літери a
`gzip –c7 myfile > archive.gz`	стиснути файл **myfile** в файл **archive.gz** зі ступенем стиснення 7 і збереженням оригінальних файлів

Програма розпакування архівних файлів gunzip

Програма **gunzip** розпаковує файли, раніше створені за допомогою програми gzip. При цьому вона заміщає стиснені файли їх оригінальними версіями, видаляючи розширення .gz з імен файлів.

Формат команди gunzip:

```
gunzip [прапори] ім'я_файлу/каталогу
```

Прапори команди gunzip:

-c, --stdout	спрямовує результат обробки на стандартне виведення, зберігаючи вихідні файли незміненими
-l, - list	виводить для кожного файлу інформацію про розмір стисненого і оригінального файлу, відсоток стиснення тощо
-r, --recursive	рекурсивно виконує обробку файлів у всіх каталогах
-S, --suffix *suffix*	вказує замість .gz використовувати розширення *suffix*
-v, --verbose	відображає в процесі обробки імена файлів і відсоток стиснення

Контрольні питання

1) Для чого призначена програма tar?

2) Для чого призначена програма gzip?

3) Чим відрізняються програми tar і gzip?

4) За допомогою яких команд можна розпакувати файл, стиснений за допомогою програми gzip?

Завдання до виконання

1) У домашньому каталозі створити каталог **dir1**. У ньому створити два підкаталоги – **dir11** та **dir12**. Створити в цих каталогах файли (імена файлів вказані на рис. 3.1). В файли, які позначені знаком «>», помістити довільну інформацію.

2) Створити архів **arc1.tar** для каталогу **dir11** з видаленням оригінальних файлів, при цьому файли **dat** і **eee** не повинні потрапити в архів. Переглянути вміст архіву.

3) Створити в каталозі **dir1** каталог **dir13**. Розпакувати в каталог **dir13** архів **arc1.tar** в інтерактивному режимі.

4) Витягти з архіву **arc1.tar** файли з іменами, що починаються з літери g, при цьому вивести список оброблюваних файлів.

5) Видалити з архіву **arc1.tar** файли, імена яких закінчуються цифрою.

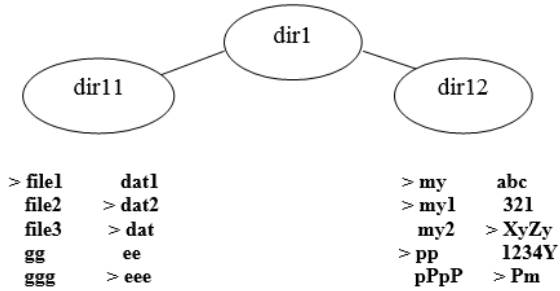

Рисунок 3.1 – Ієрархія файлів, що створюються при виконанні лабораторної роботи №3

6) Створити архів **arc2.tar** і помістити в нього файли з каталогу **dir11**, імена яких складаються з трьох символів. Переглянути вміст архіву.

7) Створити в каталозі **dir1** архів **arc3.tar** зі стисненням за допомогою **gzip**, включивши в нього файли каталогу **dir13**, в іменах яких міститься буква е.

8) Упакувати всі файли каталогу **dir12** за допомогою команди `gzip`, при цьому відобразити імена файлів та відсоток стиснення. Переглянути інформацію про упаковані файли.

9) Розпакувати файли каталогу **dir12**, імена яких складаються з трьох символів, за допомогою команди `gzip`.

10) Розпакувати файли каталогу **dir12**, в іменах яких присутні великі літери, за допомогою команди `gzip`, при цьому вивести список оброблюваних файлів.

11) Розпакувати файли каталогу **dir12**, імена яких складаються з двох символів, за допомогою команди `gunzip`.

ЛАБОРАТОРНА РОБОТА №4.
ПОШУК ФАЙЛІВ

Мета роботи: отримати практичні навички пошуку файлів (каталогів тощо) з певними характеристиками.

Постановка задачі: виконати завдання, використовуючи відповідні команди ОС UNIX (Linux).

Теоретичні відомості

Часто в процесі роботи виникає необхідність пошуку файлів з певними характеристиками, такими як права доступу, ім'я, розмір, тип, власник і т. ін. Команда **`find`** (від англ. «шукати») являє собою універсальний інструмент

пошуку файлів по багатьох критеріях, і дозволяє виконувати зі знайденими файлами необхідні дії. Загальний формат команди:

```
find каталог критерії_пошуку [дії]
```

Як перший аргумент вказується каталог, з якого необхідно почати пошук. При пошуку рекурсивно проглядаються всі підкаталоги в зазначеному каталозі.

Спочатку розглянемо критерії пошуку команди `find`.

-name *ім'я/шаблон*	пошук файлів з вказаним ім'ям або іменами, що задовольняють шаблону
-iname *ім'я/шаблон*	пошук файлів з вказаним ім'ям або іменами, що задовольняють шаблону, регістр букв ігнорується
-user *користувач*	пошук файлів, що належать зазначеному користувачеві
-group *група*	пошук файлів, що належать зазначеній групі користувачів
-nouser	пошук файлів, що належать неіснуючому користувачеві (запис для якого відсутній в файлі **/etc/passwd**)
-nogroup	пошук файлів, що належать неіснуючій групі користувачів (запис для якої відсутній в файлі **/etc/group**)
-type *тип*	пошук файлів, зазначеного типу: f – звичайний файл, d – каталог, l – символьне посилання, p – іменований канал, s – сокет, c – файл символьного пристрою, b – файл блочного пристрою
-empty	пошук порожніх файлів
-perm *права*	пошук файлів, що мають зазначені права доступу
-size *N*	пошук файлів, що мають розмір *N* одиниць: c – байтів, k – кілобайтів, b – блоків.
-depth	при пошуку спочатку проглядається вміст поточного каталогу і лише потім перевіряється запис, який відповідає самому каталогу (сенс цієї дії – отримати доступ до файлів в каталозі, для якого користувач не має прав читання або виконання)
-mindepth *N*	спускатися при пошуку файлів як мінімум на *N* рівнів нижче зазначеного каталогу
-maxdepth *N*	спускатися при пошуку файлів не нижче *N* рівнів щодо зазначеного каталогу
-atime *N*	пошук файлів, звернення до яких виконано *N* днів тому
-amin *N*	пошук файлів, звернення до яких виконано *N* хвилин тому
-mtime *N*	пошук файлів, модифікованих або створених *N* днів тому
-mmin *N*	пошук файлів, модифікованих або створених *N* хвилин тому

-ctime *N* — пошук файлів, характеристики яких (розмір, права доступу тощо) змінювалися *N* днів тому

-cmin *N* — пошук файлів, характеристики яких змінені *N* хвилин тому

-newer *файл* — пошук файлів, створених пізніше, ніж зазначений файл

-prune *ім'я_каталога* — вказує підкаталоги всередині шляху пошуку, в яких не потрібно здійснювати пошук

-L — при пошуку слідувати символьним посиланням

У значення *N* можуть бути присутні модифікатори + або -, які означають відповідно «більше, ніж *N*» і « менше, ніж *N*».

Далі наведені дії команди `find`.

-print — вивести на стандартне виведення імена знайдених файлів (ця дія використовується за замовчуванням)

-exec команда {} \; — виконати команду для всіх знайдених файлів. У даній конструкції відсутній пробіл між символами «{}», які означають «підставити ім'я кожного знайденого файлу як аргумент команди». «\;» означає «кінець команди»

-ok команда {} \; — те ж саме, що і `-exec`, але перед обробкою чергового файлу буде виводитися запит на підтвердження виконання команди

Критерії пошуку можна об'єднувати, використовуючи логічні оператори, які наведені в таблиці 4.1 у порядку зниження їхнього пріоритету.

Таблиця 4.1 – Логічні оператори, використовувані в команді `find`

Коротка форма запису оператора	Довга форма запису оператора	Опис оператора
`!` *критерій*	`-not` *критерій*	Оператор НЕ – критерій не повинен виконуватися
критерій1 `-a` *критерій2*	*критерій1* `-and` *критерій2*	Оператор І – повинні виконуватися обидва критерії одночасно (цей оператор використовується в команді `find` при об'єднанні критеріїв пошуку за замовчуванням)
критерій1 `-o` *критерій2*	*критерій1* `-or` *критерій2*	Оператор АБО – повинен виконуватися один з двох критеріїв: або *критерій1*, або *критерій2*

У загальному випадку при використанні логічних операторів критерії перевіряються в порядку їх пріоритету. Змінити порядок можна за допомогою дужок, які необхідно екранувати за допомогою символа «\».

Приклади:

`find programs -type f -name '*.c' -perm 777 -print`	знайти в каталозі **programs** загальнодоступні файли з розширенням .c.
`find /home/chris -name 'music' -type d`	знайти в каталозі **/home/chris** каталог з ім'ям **music**
`find /var/www -mtime +30 -name '*.php' -type f`	знайти в каталозі **/var/www** файли з розширенням .php, модифіковані більш ніж 30 днів тому
`find . \( -name '~*' -or -name 'temp*' \) -type f > list.lst`	знайти в поточному каталозі файли з іменами, які починаються з символу «~» або з буквосполучення temp і помістити список знайдених файлів в файл **list.lst**
`find . -size -500k -type f -exec ls -l {} \;`	знайти в поточному каталозі файли з розміром менше 500 кілобайт і вивести про них детальну інформацію
`find . -name '~*' -type f -ok rm {} \;`	знайти в поточному каталозі файли з іменами, які починаються з символу «~», і видалити їх із запитом на підтвердження виконання дії
`find . -user lion -type f -exec cp {} /home/ray/lion_files \;`	знайти в поточному каталозі файли, що належать користувачеві lion, і скопіювати їх в каталог **/home/ray/lion_files**
`find ./temp -type f ! -empty -exec gzip -8 {} \;`	знайти в підкаталозі **temp** поточного каталогу непорожні файли і заархівувати їх за допомогою програми `gzip`
`find /var -name '*[A-Z]*' | wc -l`	знайти в каталозі **/var** файли, каталоги і т. ін., імена яких містять великі літери, і підрахувати кількість знайдених файлів

Підстановка команд

Командний інтерпретатор дозволяє результат виконання однієї команди підставити як аргумент в іншу команду. Для цього команду, результат виконання якої повинен бути підставлений, необхідно взяти в зворотні лапки (символи «`»).

Наприклад, нехай задана команда пошуку в поточному каталозі і розпаковки файлів з розширеннями .gz, доступ до яких проводився більше двох місяців тому, з іменами, які починаються літерами від a до f:

```
find -name '[a-fA-F]*.gz' -type f -atime +60 -exec gunzip {} \;
```

За допомогою підстановки дану команду можна записати в такий спосіб:

```
gunzip `find. -name '[a-fA-F]*.gz' -type f -atime + 60`
```

Контрольні питання

1) Як в UNIX здійснюється пошук файлів за допомогою команди `find`?

2) Як можна об'єднувати критерії пошуку команди `find`?

3) Як здійснюється підстановка команд в UNIX?

Завдання до виконання

1) У домашньому каталозі створити каталоги **test1**, **test2** і **test3**. У них створити файли, імена яких вказані на рис. 4.1. В файли, які позначені знаком «>», помістити довільну інформацію (наприклад, текст, довідку з якої-небудь команді або результат виконання будь-якої команди).

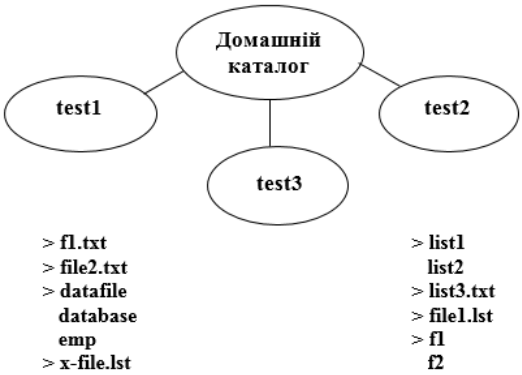

Рисунок 4.1 – Ієрархія файлів, що створюються при виконанні лабораторної роботи №4

2) Видати на екран назви файлів каталогу **/tmp**, які не належать користувачеві root.

3) Знайти в каталозі **/etc** і видати на екран назви всіх файлів розміром більше 1 байта, в іменах яких є цифри та імена яких не закінчуються літерою.

4) Знайти в домашньому каталозі і видати на екран назви всіх каталогів, створених за останній тиждень.

5) Переглянути детальну інформацію про файли і каталоги домашнього каталогу, створені за поточну пару.

6) Підрахувати кількість слів у файлах, створених в домашньому каталозі за останній день.

7) Стиснути за допомогою команди `gzip` файли каталогу test1, що містять в імені літеру f. Пошук файлів, що відповідають умові, виконати за допомогою команди `find`.

8) Розпакувати за допомогою команди `gunzip` всі файли каталогу **test1**, що мають розширення .gz. Пошук файлів, що відповідають умові, виконати за допомогою команди `find`.

9) Видалити всі порожні файли з домашнього каталогу.

10) Підрахувати у домашньому каталозі кількість файлів з іменами, що починаються з великої літери.

11) Помістити список каталогів, що знаходяться в каталозі **/etc**, в файл **test2/catalogues.lst**.

12) Видати на екран імена всіх непорожніх файлів домашнього каталогу, що мають розширення .txt і .lst.

13) Знайти в каталозі **/usr** і видати на екран назви всіх файлів з розширеннями .gif і .png розміром більше 2 Кбайт.

14) Знайти в каталозі **/usr/bin** і скопіювати в каталог **test3** всі файли з правами доступу `rwxr-xr-x`, імена яких починаються з літери b і мають в назві букву o.

ЛАБОРАТОРНА РОБОТА №5.
ВИКОРИСТАННЯ РЕГУЛЯРНИХ ВИРАЗІВ

Мета роботи: навчитися складати регулярні вирази, отримати практичні навички використання регулярних виразів при пошуку тексту у вмісті файлів.

Постановка задачі: виконати завдання, використовуючи відповідні команди ОС UNIX (Linux).

Теоретичні відомості

Регулярні вирази (англ. regular expressions, скорочено regex) – це система пошуку фрагментів в тексті, заснована на спеціальній системі запису зразків для пошуку. Зразок (англ. pattern), що задає правило пошуку, також називають

шаблоном або маскою. Регулярні вирази використовуються багатьма текстовими редакторами і утилітами для пошуку і зміни тексту на основі вибраних правил. Багато мов програмування мають підтримку роботи з регулярними виразами.

Регулярні вирази використовуються для стислого опису деякої множини рядків за допомогою шаблонів, без необхідності перерахування всіх елементів цієї множини. При складанні шаблонів використовується спеціальний синтаксис. Конкретний синтаксис регулярних виразів залежить від реалізації. Ми будемо розглядати синтаксис базових регулярних виразів UNIX, у якому більшість символів відповідають самі собі. Метасимволи базових регулярних виразів UNIX перераховані в таблиці 5.1.

Таблиця 5.1 – Метасимволи, застосовувані в регулярних виразах

Метасимвол	Значення в регулярних виразах
`.`	Відповідає будь-якому одиничному символу.
`[]`	Відповідає будь-якому одиничному символу з числа символів, що містяться в квадратних дужках. Символ «-» інтерпретується буквально лише в тому випадку, якщо він розташований безпосередньо після відкриваючої або перед закриваючою дужкою: `[abc-]` або `[-abc]`. В іншому випадку, він позначає інтервал символів. Наприклад, `[abc]` відповідає буквам *a*, *b* або *c*; `[0-9]` відповідає цифрам; `[][ab]` відповідає «*]*», «*[*», *a* або *b*.
`[^]`	Відповідає одиничному символу з числа тих, які не містяться в дужках. Наприклад, `[^abc]` відповідає будь-якому символу, крім *a*, *b* або *c*. `[^0-9]` відповідає будь-якому символу, крім цифр.
`|`	Логічне АБО: означає вибір альтернативи з заданих варіантів. Наприклад, `one|two` відповідає *one* або *two*, а `m(e|y)` відповідає *me* або *my*.
`^`	Використаний на початку регулярного виразу, відповідає початку рядка тексту.
`$`	Використаний в кінці регулярного виразу, відповідає кінцю рядка тексту.
`()`	Позначає групу символів. Наприклад, шаблони `abd|acd` і `a(b|c)d` описують одне й те саме поєднання символів: *abd* і *acd*.
`*`	Відповідає нулю або більше копій попереднього символа або групи символів. Наприклад, `go*gle` відповідає *ggle*, *gogle*, *google*, *goooogle* і. т.д.

Продовження таблиці 5.1

Метасимвол	Значення в регулярних виразах
`?`	Відповідає нулю або однієї копії попереднього символа або групи символів. Наприклад, `colou?r` відповідає і *color*, і *colour*.
`+`	Відповідає одній або більше копій попереднього символа або групи символів. Наприклад, go+gle відповідає gogle, google, gooogle і т. д. (але не ggle).
`\`	Скасовує спеціальне значення наступного за ним метасимвола (екранування). Наприклад, щоб представити безпосередньо символ «крапка», треба написати «`\.`». Сам метасимвол «`\`» теж може бути екранований, тобто представлений як «`\\`», і тоді інтерпретатор регулярних виразів буде сприймати його як простий символ зворотного слешу.
`{x,y}`	Відповідає останньому символу або групі символів, що зустрічається не менше *x* і не більше *y* раз. Наприклад, `go{1,3}gle` відповідає *gogle*, *google* або *gooogle*
`{x,}`	Відповідає останньому символу або групі символів, що зустрічається не менше *x* раз. Наприклад, `a{3,}` відповідає *ааа*, *аааа*, *ааааа* і т.д.
`{,y}`	Відповідає останньому символу або групі символів, що зустрічається не більше *y* раз. Наприклад, `a{,4}` відповідає порожньому рядку, *а*, *аа*, *ааа* і *аааа*.
`{x}`	Відповідає останньому символу або групі символів, що зустрічається точно *x* разів. Наприклад, `a{6}` відповідає *аааааа*.

При використанні діапазонів символів слід враховувати, що вони можуть залежати від обраних налаштувань локалізації. Для подолання такої проблеми використовуються класи метасимволів `[:upper:]`, `[:lower:]`, `[:alpha:]`, `[:digit:]` і т.д. (див. лабораторну роботу №1).

При складанні регулярних виразів слід також враховувати їх дві основні риси – вони є «ледачими» і «жадібними». Перше означає, що в рядку, де є кілька збігів з шаблоном, шаблон співпаде з першим з них. Наприклад, регулярний вираз `'шаблон(..)'` для рядка «*в рядку, де є кілька збігів з шаблоном, шаблон співпаде з першим з них*», поверне символи «*ом*», що відповідають першому збігу («*шаблоном*»). Друге можливе місце збігу («*шаблон с*») розглянуто не буде.

«Жадібність» регулярних виразів полягає в тому, що при використанні квантифікаторів `*` і `+`, шаблон буде збігатися з максимально довгим з можливих

варіантів. Для того ж рядка, регулярний вираз `'шаблон.*н'`, який означає підрядок, що починається з «*шаблон*», закінчується на «*н*» і з будь-якою кількістю (`*`) будь-яких (`.`) символів між «*шаблон*» і «*н*», співпаде з підрядком «*шаблоном, шаблон співпаде з першим з н*», а не з коротшим «*шаблоном, шаблон*».

Команда grep

Команда **`grep`** (від англ. global regular expression print – друк глобальних регулярних виразів) виконує пошук та виведення на екран виразів, що відповідають шаблону, в текстових файлах або в стандартному вхідному потоці (якщо імена файлів не задані).

Формат команди `grep`:

`grep [`*`прапори`*`] `*`регулярний_вираз`*` [`*`ім'я_файлу(файлів)`*`]`

Рядок, що заданий як регулярний вираз і складається з декількох слів, необхідно укладати в лапки. В іншому випадку перше слово рядка буде сприйняте як зразок пошуку, а решта слів будуть вважатися іменами файлів. Якщо зразок пошуку містить значення якої-небудь системної змінної (наприклад, `$PATH`), то рекомендується укласти його в подвійні лапки. Це необхідно тому, що, перш ніж передати параметри команді `grep`, командний інтерпретатор виконує підстановку значення змінної, яке може містити пробіли. Якщо до складу регулярного виразу входять метасимволи, зразок пошуку необхідно укладати в одинарні лапки.

Основні прапори команди `grep`:

`-c, --count`	для кожного файлу друкує кількість рядків, що збіглися з шаблоном (якщо також вказано прапор `-v`, друкує кількість рядків, що не збіглися)
`-e`	використовується, якщо необхідно задати більше одного шаблону для порівняння
`-h, --no-filename`	не виводить ім'я файлу перед кожним знайденим рядком
`-f` ***`file`*****`, --file=`*****`file`**	отримує з зазначеного файлу шаблони для пошуку (по одному в рядку)
`-i, - ignore-case`	ігнорує регістр символів в шаблоні і у вхідних файлах
`-l, --files-with-matches`	виводить імена файлів, рядки яких містять зразок пошуку
`-L, --files-without-matches`	виводить імена файлів, рядки яких не містять зразок пошуку

-n, --line-number	перед кожним знайденим рядком виводить його номер в межах файлу
-s, --no-messages	забороняє видачу повідомлень про помилки з приводу неіснуючих файлів або файлів, що не читаються
-v, --invert-match	вибирає рядки, що не збігаються з шаблоном
-w, --word-regexp	повна відповідність регулярного виразу слову
-x, --line-regexp	повна відповідність регулярного виразу рядкові

Приклади:

`grep –v '^Hello.*' file.txt`	вивести на екран рядки файлу **file.txt**, які не містять рядків, що збігаються з шаблоном '^Hello.*'
`grep 'help me' ./* | wc –l`	знайти в файлах поточного каталогу рядки, в яких присутній текст *help me*, і вивести на екран кількість цих рядків
`ls /bin | grep '2$'`	вивести на екран список файлів каталогу **/bin**, імена яких закінчуються цифрою 2
`ls /bin | grep '^b.*a'`	вивести на екран список файлів каталогу **/bin**, імена яких починаються літерою b і містять літеру a
`cat ./* | grep 's\{2\}'`	вивести на екран рядки файлів поточного каталогу, які містять подвоєну літеру s (аналогічно команді `grep 's\{2\}' *`)
`grep '[0-9]\{3\}-[0-9]\{2\}-[0-9]\{2\}' tel_numbers`	вивести на екран рядки файлу **tel_numbers**, які містять номери телефонів в форматі XXX-XX-XX
`mv `grep –L –e 'Hello world' -e 'one of us' dir1/*` dir2`	перемістити в каталог **dir2** файли каталогу **dir1**, які не містять тексту *Hello world* і *one of us*
`tar –czf arch.tar `find . –type f –atime +30 –exec grep –l 'go+gle' {} \;``	заархівувати і стиснути за допомогою команди `tar` файли поточного каталогу, доступ до яких проводився більше 30 днів тому і які містять текст *gogle*, де буква о повторюється хоча б один раз
`grep -n -w 'is' file.txt`	вивести на екран пронумеровані рядки файлу file.txt, які містять слово is

Контрольні питання

1) Що таке регулярні вирази і для чого вони використовуються?

2) Які особливості використання команди `grep`?

3) Які метасимволи застосовуються при складанні базових регулярних виразів UNIX та що вони означають?

4) Що означає, що регулярні вирази є жадібними та ледачими?

Завдання до виконання

1) У домашньому каталозі створити каталоги **cat1** і **cat2**. В каталозі **cat1** створити файли з іменами **file1**, **file2**, **file3**, **relatives**, **friends**, **others**, **keywords**, **birthdays**, **list**. За допомогою команди `man` в file1 помістити довідку по команді `grep`, в **file2** – довідку по команді `find`, а в **file3** – довідку по команді `gzip`.

В файли **relatives**, **friends** і **others** помістити імена та номери телефонів відповідно Ваших родичів, друзів і знайомих (кожен запис з нового рядка). Номери телефонів повинні бути записані в форматі XXX-XXX-XX-XX (як номери мобільного або міжміського зв'язку), так і в форматі XXX-XX-XX (як міські номери) і перераховані через кому. Наприклад: *Ivanov Ivan: 097-234-56-78, 726-15-89*

Якщо для когось Ви вказуєте лише один номер телефону, кому після нього ставити не потрібно.

У файл **birthdays** помістити імена і дати народження родичів, друзів і знайомих (кожен запис з нового рядка). У файл **keywords** помістити наступні слова і словосполучення: *Unix, argument, symbolic link, command, question, option, one more* (кожне слово або словосполучення з нового рядка, без розділових знаків).

2) Отримати список файлів каталогу **cat1**, у вмісті яких присутнє слово *name*.

3) Підрахувати кількість рядків у файлах каталогу **cat1**, у вмісті яких присутнє словосполучення *to be*.

4) Видати на екран всі рядки файлів каталогу **/etc**, що містять входження слів *terminal* або *keyboard*.

5) За допомогою команди `grep` отримати список файлів каталогу **/bin**, імена яких починаються з буквосполучення ch.

6) За допомогою команди `grep` отримати список файлів каталогу **/etc**, імена яких починаються літерами від a до m і які мають в назві буквосполучення at. Помістити цей список у файл **list**.

7) Видати на екран всі імена файлів каталогу **cat1**, що містять в тексті чотири цифри поспіль.

8) Підрахувати, скільки людей, перерахованих в файлах **relatives**, **friends** і **others**, мають більше одного номера телефону.

9) Заархівувати за допомогою команди `gzip` файли каталогу **cat1**, у вмісті яких присутні номери телефонів в форматі 09X-XXX-XX-XX та 7XX-XX-XX.

10) Скопіювати в каталог **cat2** файли каталогу **cat1**, у вмісті яких присутні слова і словосполучення, зазначені в файлі **keywords**.

11) Видати на екран (з їх номерами в файлі) всі рядки файлів каталогу **cat1**, в яких є слова, що складаються лише з великих літер (двох і більше, без цифр).

12) Видати на екран імена людей, перерахованих у файлі **birthdays**, які народилися під знаком Овна (21 березня – 20 квітня). Завдання виконати за допомогою одного регулярного виразу.

ЛАБОРАТОРНА РОБОТА №6.
ВИКОРИСТАННЯ ФІЛЬТРІВ

Мета роботи: ознайомитись з основними командами фільтрації тексту і отримати практичні навички створення конвеєрів з використанням фільтрів.

Постановка задачі: виконати завдання, використовуючи відповідні команди ОС UNIX (Linux).

Теоретичні відомості

Фільтрація тексту – це процес перетворень над вхідним потоком тексту до того як він буде виданий в вихідний потік. Хоча як вхідний, так і вихідний потік можуть надходити з файлу, в UNIX фільтрація переважно здійснюється через конвеєр команд, коли стандартне виведення однієї команди перенаправляється на стандартне введення наступної команди (ці дві команди поєднуються оператором конвеєризації введення/виведення «`|`»). Отже, **фільтр** – це команда, яка бере введення з аргументів командного рядка, якщо вони присутні, інакше введення читається зі стандартного вхідного потоку. Використання фільтрів в конвеєрах дозволяє здійснювати декілька послідовних операцій над даними в одній команді.

Фільтрами є команди `cat` і `wc`, розглянуті в лабораторній роботі №1.

Наприклад, для підрахунку кількості слів в файлах з іменами **file1**, **file2** і **file3** команда `wc` викликається наступним чином:

```
wc -w file[1-3]
```

або

```
cat file[1-3] | wc -w
```

Шляхом попередньої конкатенації файлів дані передаються на вхід команді `wc`, яка підсумовує підраховані кількості символів у вмісті файлів.

Для підрахунку кількості символів у виведенні команди `ls` необхідно написати

```
ls file[1-3] | wc -c
```

Команда **head [*прапори*] [*файл1*] [*файл2*] ...** виводить на стандартне виведення перші рядки зазначених файлів. За замовчуванням кількість рядків дорівнює 10. Якщо не вказано жодного імені файлу або серед аргументів зустрівся символ «-», команда `head` читає дані зі стандартного вхідного потоку. Якщо в командному рядку вказано більше одного імені файлу, то перед виведенням рядків кожного з них буде виводитися заголовок, що містить ім'я файлу у вигляді `==> ім'я_файлу <==`

Прапори команди `head`:

-*number*	цей прапор розпізнається лише якщо він вказаний першим, визначає кількість рядків для виведення і є десятковим числом, за яким може слідувати буква (одиниця виміру)
-c *n*, --bytes=*n*	замість початкових рядків друкувати вміст зазначеної кількості байтів. Додавши до вказаної кількості букву, можна виконувати роздруківку в блоках (`b`), кіло- (`k`) і мегабайтах (`m`)
-n *n*, --lines=*n*	вивести перші *n* рядків
-q, --quiet	не друкувати заголовки, що містять імена файлів
-v, --verbose	друкувати заголовки, що містять імена файлів

Команда **tail [*прапори*] [*файл1*] [*файл2*] ...** виводить на стандартне виведення останні рядки зазначених файлів. За замовчуванням кількість рядків дорівнює 10. Якщо не вказано жодного імені файлу або серед аргументів зустрівся символ «-», команда `tail` читає дані зі стандартного вхідного потоку.

Прапори команди `tail`:

-n *n*, --lines=*n*	виводити останні *n* рядків. Якщо перший символ параметра *n* є знаком +, то виведення здійснюється з *n*-го байта або рядка від початку файлу
-c *n*, --bytes=*n*	замість останніх рядків вивести вміст зазначеної кількості останніх байтів. Додавши до вказаної кількості букву, можна робити роздруківку в блоках (`b`), кіло- (`k`) і мегабайтах (`m`)
-q, --quiet	не друкувати заголовки, що містять імена файлів
-v, --verbose	друкувати заголовки, що містять імена файлів

Команда **tr [*прапори*] [*набір1* [*набір2*]]** копіює стандартне введення на стандартне виведення із заміною або видаленням обраних символів. Введені символи, знайдені в *наборі1*, замінюються на відповідні символи з *набора2*. Якщо який-небудь символ з'являється в *наборі1* кілька разів, а відповідні символи з *набора2* різні, то береться останній варіант.

Допускаються будь-які комбінації прапорів `-cds`:

-c, --complement *набір1* замінюється його доповненням до множини символів з вісімковими кодами від 001 до 377

-d, --delete видаляє всі символи, що належать *набору1*

-s, --squeeze-repeats замінює послідовність однакових символів в *наборі1* на один такий символ

Щоб поміщати в ланцюжки відрізки алфавіту і повторювані символи, можна використовувати наступні скорочення:

`[c1-c2]` позначає множину символів, ASCII коди яких належать відрізку від коду символу *c1* до коду символу *c2*

`[c*n]` позначає символ `c`, повторений `n` разів. Якщо перша цифра в *n* є 0, *n* розглядається як вісімкове число, інакше – як десяткове. Нульове або відсутнє *n* сприймається як «дуже багато»; ця можливість корисна при доповненні *набору1* до довжини *набору2*

Для скасування трактування символу як спеціального застосовується екранування символу за допомогою зворотного слеша (\).

В команді `tr` можна використовувати найменування класів символів, які розглянуті в лабораторній роботі №1, наприклад, `[:alpha:]` або `[:punct:]`.

Команда **sort [*прапори*] [*файл1*] [*файл2*] ...** упорядковує рядки, що входять в зазначені файли, і видає результат на стандартне виведення. Якщо не вказано жодного імені файлу або серед аргументів зустрівся символ «-», вихідна інформація надходить зі стандартного вхідного потоку.

При впорядкуванні використовується один або кілька ключів, що виділяються з кожного введеного рядка. За замовчуванням ключ упорядкування – весь рядок, а порядок є лексикографічним, відповідним прийнятому кодуванні символів.

При впорядкуванні команда `sort` враховує структуру файлу. Якщо при її виклику не вказати жодного параметра, рядки впорядковуються спочатку по першому полю (за замовчуванням роздільником полів є пробіл або табуляція; поля нумеруються починаючи з одиниці). Якщо в кількох рядках вміст першого поля однаковий, здійснюється впорядкування цих рядків по другому полю. Якщо друге поле однакове – по третьому полю і т.д.

Прапори команди `sort`:

-c, --check перевірити, чи є вихідний файл впорядкованим. На стандартне виведення нічого не виводиться. У разі порушення впорядкованості рядків виводиться відповідне повідомлення

-o *file*, --output=*file* результат спрямувати не на стандартне виведення, а в зазначений файл

-u, --unique	з усіх співпадаючих рядків виводити лише один
-d, --dictionary-order	«словниковий» порядок упорядкування: при порівнянні є значущими лише букви, цифри, пробіли і знаки табуляції
-f, --ignore-case	перетворювати малі літери в великі
-i, --ignore-nonprinting	при нечислових порівняннях ігнорувати не-ASCII символи
-M	порівнювати як місяці. Перші три символи, відмінні від пробілу, порівнюються таким чином, що «JAN» < «FEB» <... < «DEC» (малі літери перетворюються в великі). Решта трохсимвольних поєднань вважаються меншими, ніж «JAN». Цей прапор включає прапор `-b` (див. нижче)
-n, --numeric-sort	порівнювати як числа. Початкові пробіли відкидаються, потім цифрові ланцюжки символів, які, можливо, містять знак мінус і десяткову точку, порівнюються як числа. Цей прапор включає прапор `-b` (див. нижче)
-r, --reverse	виконати впорядкування в зворотному порядку

Якщо прапори, що задають спосіб порівняння, вказані до обмежень на ключі впорядкування, то вони застосовуються глобально до всіх ключів. Якщо ж відповідні прапори асоційовані з певними ключами впорядкування (див. нижче), вони впливають лише на «свої» ключі.

Полем вважається мінімальна послідовність символів, за якою слідує роздільник полів або новий рядок. За замовчуванням символом-роздільником вважається пробіл чи табуляція. Пробіли і табуляції, що відразу слідують за роздільником (якщо вони є), належать наступного поля. Всі пробіли на початку рядка входять в перше поле. На трактування роздільників впливають такі прапори:

-b, --ignore-leading-blanks	ігнорувати початкові пробіли
-t *sep*, --field-separator=*sep*	використовувати символ *sep* як роздільник полів (при застосуванні прапора `-k`). Два роздільники, що стоять поруч, обмежують порожнє поле. За замовчуванням роздільником є послідовність пробілів (два пробіли, що стоять поруч, не обмежують порожнє поле). Пробіл та інші спеціальні символи необхідно укладати в лапки

-k *field1* [, *field2*] встановити поле (або ключ) впорядкування, починаючи з *field1* і закінчуючи *field2* (включно). Номери полів вказуються, починаючи з одиниці. Якщо поля не задані, впорядкування виконується по всьому рядку

Якщо вказано кілька полів упорядкування, то більш пізні використовуються лише в разі рівності більш ранніх. Якщо значення полів упорядкування двох рядків збігаються, рядки впорядковуються з урахуванням всіх символів.

Команда **uniq [*прапори*] [*вихідний_файл* [*результуючий_файл*]]** видаляє дубльовані рядки з раніше впорядкованого файлу і поміщає їх на стандартне виведення або в зазначений результуючий файл.

Для того щоб повторювані рядки були виявлені, вони повинні бути сусідніми. Тому команду `uniq` рекомендовано застосовувати до файлу після застосування команди `sort`.

Прапори команди `uniq`:

-u, --unique видати лише рядки вихідного файлу, що не повторюються

-d, --repeated видати одну копію рядків, які мають дублікати

-c, --count перед кожним рядком друкувати кількість його повторень

-f *n*, --skip-fields=*n*, -*n* при порівнянні ігнорувати перші ***n*** полів, розташованих на початку кожного рядка, разом з пробілами, які їм передують

-s *n*, --skip-chars=*n*, +*n* при порівнянні ігнорувати перші ***n*** символів на початку кожного рядка. Спочатку відкидаються поля, потім символи

-i, --ignore-case ігнорувати регістр букв при порівнянні

-w, --check-chars=*N* визначити кількість символів, починаючи з початку рядка, що беруть участь в порівнянні; всі інші символи ігноруються

Команда **cut [*прапори*] [*файл1*] [*файл2*] ...** використовується для вибірки полів з кожного рядка файлу і відправки їх на стандартне виведення. Поля можуть мати фіксовану або змінну довжину (в цьому випадку межею поля є символ-роздільник, наприклад, пробіл). Якщо не вказано жодного імені файлу або серед аргументів зустрівся символ «-», команда `cut` читає дані зі стандартного вхідного потоку.

Прапори команди `cut`:

-c *список*	*список* специфікує позиції символів (наприклад, `-c 1-72` задає перші 72 символи кожного рядка)
-b *список*, **--bytes *список***	показати лише байти з позицій, вказаних в *списку*. Символи табуляції і повернення на один символ (Backspace) трактуються подібно іншим символам і займають 1 байт
-f *список*	*список* є списком номерів полів; передбачається, що в файлі поля розділені символом-роздільником (див. прапор `-d`); наприклад, `-f1,7` визначає лише перше і сьоме поля. Якщо не заданий прапор `-s`, то рядки, що не містять роздільників, поміщаються в результат без будь-якої обробки
-d *символ*, **--delimiter *символ***	*символ* є роздільником полів (у разі застосування прапора `-f`). За замовчуванням роздільником є символ табуляції. Пробіл або інші символи, що мають спеціальне значення, повинні бути укладені в лапки
-s, **--only-delimited**	ігнорувати рядки без символів-роздільників в разі застосування прапора `-f`. Якщо прапор не заданий, то рядки без роздільників поміщаються в результат без обробки
--output-delimiter *символ*	розділяє зазначеним роздільником поля результуючого файлу. Застосовується спільно з прапором `-f`. За замовчуванням використовується роздільник вихідного файлу

Один з прапорів `-b`, `-c` або `-f` повинен бути заданий обов'язково.

Параметр *список* може складатися з одного або декількох діапазонів, розділених комами. Діапазон задається наступним чином:

- *n* – *n*-й байт, символ або поле, відраховується від 1;
- `n-` – від *n*-го байта, символу або поля до кінця рядка;
- *n*-*m* – від *n*-го до *m*-го байта, символу або поля (включно);
- -*m* – від 1-го до *m*-го байта, символу або поля (включно).

Список цілих номерів полів вказується в порядку зростання, номери перераховуються через кому; також використовується символ «-» для позначення інтервалів, наприклад: 1,4,7; 1-3,8; -5,10 (позначення для інтервалу 1-5,10); 3- (позначення для інтервалу від третього до останнього поля).

Далі наведена структура конфігураційного файлу **/etc/mtab**, використаного у прикладах. Даний файл містить інформацію про змонтовані файлові системи. Кожний його рядок складається з 6 полів, розділених пробілами:

– змонтований пристрій;

– точка монтування (ім'я каталогу, в який змонтована файлова система);

– тип файлової системи;

– параметри монтування;

– параметр, який вказує, чи потрібно включати розділ з даною файловою системою в архів при створенні резервних копій;

– параметр, який вказує порядок перевірки файлових систем під час початкового завантаження.

Приклади записів в файлі **/etc/mtab**:

```
# Розділ диска /dev/sda1 змонтований в кореневий каталог
/dev/sda1      /              ext3      defaults       0          1

# CD-привід /dev/cdrom змонтований в каталог /mnt/cdrom
/dev/cdrom     /mnt/cdrom     iso9660   noauto,ro      0          0
```

Приклади:

`head –n 12 file1`	виводить перші 12 рядків файлу **file1**
`ls | tr '\n'`	виводить список імен файлів поточного каталогу в стовпчик по одному
`cat 1.text | tr ' ' '\t' > 2.text`	перетворює в файлі **1.text** пробіли в символи табуляції і результат поміщає в файл **2.text**
`sort –k2 f1`	впорядковує файл **f1**, використовуючи в якості ключа друге поле
`sort –r –k3 f[23] –o f4`	впорядковує по спадаючій вміст файлів **f2** і **f3**, результат поміщає в файл **f4**. Ключем упорядкування є третє поле
`cat /etc/mtab | cut –d ' ' –f1,2`	виводить список змонтованих файлових систем та точок монтування
`sort –t ':' –n –k3 /etc/passwd`	виводить вміст файлу **/etc/passwd**, упорядкувавши облікові записи по числовим значенням ідентифікаторів користувачів (UID)
`cat /etc/group | grep 'root' | cut –d ':' –f1`	виводить список груп, членом яких є користувач root

`ls -S `find –maxdepth 1 –type f` | tr ' ' '\n' | head -n 1`	виводить ім'я найбільшого за розміром файлу поточного каталогу (`ls -S` – впорядкування за зменшенням розміру файлу)
`cat /etc/passwd | cut -d ':' –f1 | wc -L`	підраховує кількість букв в найдовшому реєстраційному імені користувача
`who | cut -d ' ' –f1 | sort | uniq | wc –l`	підраховує кількість користувачів, що працюють в даний момент в системі
`ls -l | tr –s ' ' | cut -d ' ' –f 6-9`	виводить на екран час модифікації файлів поточного каталогу і імена файлів

Контрольні питання

1) Що таке фільтри і для чого вони необхідні?

2) Які команди фільтрації тексту Вам відомі?

3) Що являє собою конвеєр?

4) Як спільно використовуються команди `uniq` та `sort`?

5) Як можна отримати рядки файлу, починаючи з десятого і закінчуючи шістнадцятим?

Завдання до виконання

1) У домашньому каталозі створити каталог з ім'ям **mygroup**. У ньому створити файл з ім'ям **list**, що містить список студентів Вашої групи у форматі:

Ivanenko Ivan Ivanovich
Petrenko Petro Petrovich
Sidorenko Olena Andriivna
...

2) Записати в файл **spisok,** (останнім символом імені файлу є кома) перші 5 рядків файлу **list**.

3) Видати на екран в алфавітному порядку лише імена студентів групи, перерахованих у файлі **spisok,** (останнім символом імені файлу є кома). Якщо однакових імен декілька, вивести необхідно лише одне з них.

4) Створити файл з ім'ям **spisok1**, де буде зберігатися список студентів групи, відсортований в зворотному алфавітному порядку.

5) Створити файл з ім'ям **spisok2**, де всі великі літери файлу **list** будуть замінені малими, пробіли – плюсами, літери а – літерами о, а літери о – літерами а.

6) Вивести на екран останні 2 рядки кожного з файлів **spisok1**, **spisok2** і **spisok,** (останнім символом імені файлу є кома).

7) Підрахувати кількість виконуваних файлів каталогу **/bin**.

8) Вивести на екран інформацію про змонтовані розділи дисків і про типи їх файлових систем (з файлу **/etc/mtab**).

9) Вивести на екран наступну інформацію про каталоги, які містяться в каталозі **/etc**: права доступу, імена власників, імена каталогів.

10) Вивести на екран детальну інформацію про файли каталогу **/etc**, розташувавши файли в порядку зростання їх розміру.

11) Визначити, скільки в системі зареєстровано користувачів, реєстраційні імена яких починаються з голосної букви.

12) Вивести на екран реєстраційні імена користувачів системи, їх UID і додаткову інформацію про користувачів (з файлу **/etc/passwd**), відсортувавши дані в порядку убування значення UID.

13) Вивести на екран реєстраційні імена користувачів, у яких інтерпретатором команд за замовчуванням є /bin/sh.

14) Записати в файл з ім'ям info.txt ім'я першого файлу каталогу **/usr/bin**, в якому міститься найбільша кількість рядків.

15) Визначити, скільки букв у найдовшому імені файлу каталогу **/bin**.

ЛАБОРАТОРНА РОБОТА №7.
СЦЕНАРІЇ КОМАНДНОГО ІНТЕРПРЕТАТОРА

Мета роботи: здобути навички створення сценаріїв на мові командного інтерпретатора.

Постановка задачі: створити сценарії, використовуючи відповідні команди та управляючі конструкції мови командного інтерпретатора ОС UNIX (Linux).

Теоретичні відомості

Командний інтерпретатор обробляє команди, які вводяться користувачем з клавіатури, або містяться в файлі, який називається сценарієм і є програмою, що інтерпретується.

Командний інтерпретатор може бути запущений на виконання в двох режимах – інтерактивному і неінтерактивному. Якщо інтерпретатор видає користувачеві запрошення для введення команд, а потім їх виконує, то він працює в інтерактивному режимі. Завершення роботи з командним інтерпретатором в цьому випадку відбувається по команді користувача.

У неінтерактивному режимі командний інтерпретатор не взаємодіє з користувачем. Замість цього він читає команди з деякого файлу (сценарію) і виконує їх. Коли буде досягнутий кінець файлу, командний інтерпретатор

завершить своє виконання. Запуск командного інтерпретатора в неінтерактивному режимі можна здійснити в наступний спосіб:

ім'я_командного_інтерпретатора ім'я_сценарію

Тут *ім'я_сценарію* – це ім'я файлу, що містить команди для виконання. Він може бути створений за допомогою будь-якого текстового редактора. Як *ім'я_командного_інтерпретатора* може бути зазначений, наприклад, sh або bash.

Для того щоб мати можливість виконувати сценарій, набираючи лише його ім'я, необхідно зробити сценарій виконуваним. Для цього необхідно встановити відповідні права доступу до файлу за допомогою команди

chmod +x *ім'я_сценарію*

Також необхідно явно вказати, який командний інтерпретатор повинен застосовуватися для виконання даного сценарію. Для цього в першому рядку сценарію розміщується заголовок `#!`*шлях_до_командного_інтерпретатора*. Наприклад, якщо необхідно вказати, що для виконання сценарію слід використовувати інтерпретатор bash, першим рядком сценарію повинен бути рядок `#!/bin/bash`

Сценарій може містити коментарі, які починаються з символу `#`.

Приклад короткого сценарію:

```
#!/bin/sh
date
who
```

Декілька команд в сценарії можуть записуватися в один рядок. При цьому як роздільник між ними необхідно вказувати крапку з комою. Наприклад:

```
#!/bin/sh
date; who
```

Змінні

Змінна – це програмний об'єкт, здатний приймати значення. Командний інтерпретатор дозволяє створювати і видаляти змінні, а також присвоювати їм значення. Імена змінних визначаються за тими ж правилами, що і в мові програмування С.

Визначаються змінні наступним чином (відсутність пробілів до і після знаку рівняння є істотною):

ім'я_змінної=значення

Всі змінні за замовчуванням мають рядковий тип. Наприклад, `MY_NAME=Sergiy` визначає змінну з ім'ям `MY_NAME` і привласнює їй значення

`Sergiy`. В змінній можна зберігати будь-яке потрібне значення. Якщо значення змінної містить пробіли, то його необхідно укласти в лапки.

Якщо змінній присвоєна послідовність чисел, то значення цієї змінної буде трактуватися як число. Наприклад, `a=12345`.

Для того щоб отримати значення змінної, перед її ім'ям необхідно поставити знак `$`, а ім'я змінної укласти в фігурні дужки.

Наприклад, в результаті виконання команди `echo a` на термінал виведеться буква a, а в результаті виконання команди `echo ${a}` буде надруковане число 12345.

При підстановці значень змінних фігурні дужки в більшості випадків можна опускати. Але зверніть увагу, що в результаті виконання команди `echo ${a}1` до значення змінної a буде дописана одиниця, тобто отримаємо значення 123451, а в результаті виконання команди `echo $a1` на термінал буде виведене значення змінної `a1`, якщо вона визначена в програмі, і нічого не буде виведено, якщо змінна `a1` не визначена.

За допомогою конструкції `${#`*`ім'я_змінної`*`}` можна отримати кількість символів у змінній (довжину змінної).

За допомогою конструкції `${`*`ім'я_змінної`*`:`*`m`*`:n}` можна виділити підрядок зі змінної. Тут *m* – номер позиції, починаючи з якої буде проводитися виділення підрядка (відлік ведеться з нуля), а *n* – кількість виділених символів. Значення *n* можна опускати. Тоді буде проводитися виділення символів до кінця рядка.

У тому випадку, коли деяка змінна стає непотрібною, її можна видалити за допомогою команди `unset` *`ім'я_змінної`*.

Нижче наведений приклад, який ілюструє роботу зі змінними в сценаріях.

```
#!/bin/bash
MY_NAME=Ivan
MY_FULL_NAME="Ivan A. Petrov"
echo name = $MY_NAME and full name = $MY_FULL_NAME
echo Length of my full name is ${#MY_FULL_NAME} symbols
echo My surname is ${MY_FULL_NAME:8}
m=0
n=4
echo My name is ${MY_FULL_NAME:m:n} # підстановка
          # розповсюджується на всі змінні в фігурних дужках
echo ${MY_FULL_NAME:8:4}
unset MY_NAME
```

В результаті виконання сценарію на термінал буде видане таке повідомлення:

```
name = Ivan and full name = Ivan A. Petrov
Length of my full name is 14 symbols
My surname is Petrov
```

```
My name is Ivan
Petr
```

Всі розглянуті вище приклади змінних – це приклади **скалярних змінних**, тобто таких, які можуть зберігати лише одне значення. Поряд з ними можна використовувати **змінні-масиви**. Доступ до елементів масиву здійснюється операцією індексування `[]`. Мова програмування командного інтерпретатора не вимагає попереднього оголошення змінної масивом із зазначенням його розмірності. При цьому окремі елементи масиву створюються по мірі доступу до них. Нижче наведений приклад роботи з масивом `NAME`.

```
#!/bin/sh
NAME[0]=Sergiy
NAME[10]=Katerina
NAME[2]=Anna
echo All names: ${NAME[*]}
echo ${NAME[10]} and ${NAME[2]} are sisters
```

Якщо замість індексу елемента використовувати символ «*», результатом виразу буде список значень всіх елементів масиву (в даному випадку таких три).

Область видимості змінних

У мові командного інтерпретатора всі змінні діляться на три категорії і кожна змінна має свою зону видимості.

1) **Локальна змінна** існує лише всередині конкретного екземпляра командного інтерпретатора. Вона не доступна іншим програмам, що запускаються на виконання з цього інтерпретатора. Всі розглянуті вище змінні були локальними.

2) **Змінна оточення** доступна будь-якій програмі, запущеної з даного командного інтерпретатора.

3) **Змінна командного інтерпретатора** – це спеціальна змінна, яка встановлюється командним інтерпретатором і необхідна йому для коректної роботи. Деякі з них є змінними оточення, а деякі – локальними. Змінну можна розмістити в оточенні, виконавши команду експорту:

```
export ім'я_змінної
```

В результаті виконання наступного сценарію на термінал будуть видані значення всіх змінних оточення командного інтерпретатора і, в тому числі, змінної `MY_NAME`.

```
#!/bin/sh
MY_NAME=Ivan ; export MY_NAME;
set  # команда для виведення всіх змінних оточення і їх значень
```

Нижче наведений список деяких стандартних змінних оточення і змінних командного інтерпретатора із зазначенням їх призначення.

`HOME` містить шлях до домашнього каталогу користувача

`HOSTNAME` містить ім'я комп'ютера, на якому виконується командний інтерпретатор

`PATH` визначає список каталогів (розділених двокрапкою), в яких командний інтерпретатор шукає програми, що запускаються на виконання

`MANPATH` визначає список каталогів, в яких програма `man` шукає довідкову інформацію по запитаній команді

`USER` містить ім'я користувача, що працює з командним інтерпретатором в даний момент

`PS1` основний рядок запрошення (за замовчуванням `$`)

`PS2` додатковий рядок запрошення (за замовчуванням `>`). В інтерактивному режимі перед введенням команди інтерпретатор виводить основний рядок запрошення. Якщо натиснута клавіша [Enter], але для завершення команди потрібно подальше введення, то виводиться додатковий рядок запрошення

Команда `echo ${PATH}` виведе на екран значення змінної оточення `PATH`, наприклад `/bin:/usr/bin`.

Для додавання ще одного каталогу (наприклад, **/home/user**) для пошуку програм, що запускаються на виконання, необхідно записати команду

```
PATH=${PATH}:/home/user
```

Позиційні змінні (або параметри) – це аргументи сценаріїв. Їхніми іменами служать цифри: `0` – ім'я команди, `$1` – перший аргумент, `$2` – другий аргумент і т.д. до `$9`. Значення позиційним змінним можуть бути присвоєні і командою `set`.

Наприклад, програма з іменем `prog` служить для виведення максимального серед трьох введених чисел і викликається наступним чином:

```
prog -d 23 45 38
```

В цьому випадку позиційна змінна `$0` містить значення `prog`, змінна `$1` – значення `-d`, змінна `$2` – значення `23`, змінна `$3` – значення `45`, а змінна `$4` – значення `38`. Значення інших позиційних змінних будуть порожніми рядками.

Кількість переданих програмі позиційних параметрів визначається за допомогою конструкції `$#`.

Програмі може бути передано більше 9 параметрів. За допомогою команди `shift` *n* виконується зсув позиційних параметрів. Значення *n* визначає, на скільки позицій вліво необхідно зсунути параметри. Наприклад, якщо вказано `shift 3`, то `$4` стає `$1`. Значення *n* може бути відсутнім, і тоді передбачається, що *n* дорівнює 1. В цьому випадку, наприклад, `$2` стає `$1`.

Наступні змінні автоматично встановлюються командним інтерпретатором:

`$#` кількість позиційних параметрів

`$-` прапори, вказані при запуску командного інтерпретатора або командою `set`

`$?` десяткове значення, повернене попередньою виконаною командою

`$$` номер поточного процесу

`$!` номер процесу останньої фонової команди

`$@` всі аргументи командного рядка (еквівалентно `$1 $2 $3 ...`)

`$*` всі аргументи командного рядка (еквівалентно "`$1 $2 $3 ...`")

Засоби введення/виведення

Задачі введення/виведення в сценаріях вирішуються за допомогою використання команд `echo` і `read`.

Команда **`echo`** видає на стандартний висновок значення всіх своїх параметрів.

Команда **`read`** ***`ім'я1 ім'я2... ім'яN`*** зчитує зі стандартного введення рядок, виділяє з нього окремі слова (групи символів, що відокремлюються пробілами) і кожне слово заносить у відповідні змінні, зазначені як параметри. При цьому якщо змінних менше, ніж виділених слів, то в останню з них буде занесена решта рядка.

Як приклад наведений сценарій, в якому у користувача запитується рядок, після чого він виводиться на термінал. Параметр `-n` забороняє перехід на наступний рядок після закінчення виведення командою `echo`, що дозволено за замовчуванням.

```
#!/bin/sh
echo -n Input your name:
read INPUT
echo Hello, $INPUT !
```

Обчислення арифметичних виразів

Оператори для обчислення арифметичних виразів представлені в таблиці 7.1 в порядку зниження їхнього пріоритету. Для зміни порядку обчислення використовуються дужки.

Таблиця 7.1 – Оператори для обчислення арифметичних виразів

Оператор	Значення
`-`	унарний мінус
`!-`	логічне заперечення, двійкова інверсія (додаток до одиниці)
`*, /, %`	множення, ділення, взяття залишку від ділення
`+, -`	додавання, віднімання
`<<, >>`	порозрядний зсув вліво, порозрядний зсув вправо

Продовження таблиці 7.1

Оператор	Значення
==, !=	перевірка на рівність, перевірка на нерівність
&	порозрядне І
^	порозрядна виключна диз'юнкція
|	порозрядне АБО
&&	логічне І
||	логічне АБО
=	присвоєння значення
+=, -=	присвоєння після складання, присвоєння після віднімання
*=, /=, %=	присвоєння після множення, присвоєння після ділення, присвоєння після взяття залишку від ділення
<<=, >>=	присвоєння після порозрядного зсуву вліво, присвоєння після порозрядного зсуву вправо

Команда `let` *вираз* дозволяє обчислювати цілочисельні арифметичні вирази. Вирази складаються з чисел, операторів і змінних командного інтерпретатора (без попереднього символу `$`). Вирази повинні укладатися в лапки, якщо вони містять пробіли або інші спеціальні символи. Наприклад:

```
let 2*7                 # видає значение 14
let a=11                # те ж, що й a=11
let a=a+5               # еквівалентно "a = a + 5"
let "a /= 4"            # еквівалентно let "a = a / 4"
let "a -= 5"            # еквівалентно let "a = a - 5"
let "a %= 8"            # еквівалентно let "a = a % 8"
let "count=0" "i=i+1"   # присвоїти значення змінним і та count
let "num % 2"           # перевірка числа на парність
```

Команда `expr` *вираз* є універсальним обробником виразів і застосовується для обчислення заданого виразу. Операнди виразу обов'язково повинні відділятися один від одного пробілами, а спецсимволи (наприклад, оператор множення) повинні бути екрановані. Вирази можуть бути арифметичними, логічними або рядковими. Наприклад:

```
expr 3 + 5          # поверне 8
expr 5 % 3          # поверне 2
expr 5 \* 3         # поверне 15
```

Приклади підстановки арифметичних виразів:

```
A=`expr \(5 + 3 \)  / 3` # змінній A присвоїти результат
(5+3)/3
y=`expr $y + 1`     # операція інкремента змінної y,
                    # те ж саме, що let y=y+1 або y=$(($y+1))
```

В арифметичних підстановках зворотні одинарні лапки можуть бути замінені на подвійні круглі дужки `$((...))` або на конструкції із застосуванням команди `let`:

```
z=$(($z+3))
z=$((z+3))                        # аналогічно попередній команді
foo=$(( ((5 + 3*2) - 4) / 2 ))# в результаті виконання
     # цієї команди змінна foo прийме значення 3
```

Всередині подвійних круглих дужок підстановка значень змінних виконується автоматично.

Управляючі конструкції

Проста команда – це послідовність слів, поділена пробілами. Перше слово є ім'ям команди, а решта слів будуть передані їй як аргументи. Ім'я команди передається їй як аргумент номер 0 (тобто ім'я команди є значенням змінної `$0`). Значення, що повертається простою командою, – це статус її завершення, якщо вона завершилася нормально, або вісімкове 200 + статус, якщо команда завершилася аварійно.

Список команд являє собою послідовність з однієї або декількох команд, розділених символами «`;`», «`&`», «`&&`» або «`||`» і, можливо, закінчується символом «`;`» або «`&`». З чотирьох зазначених операцій «`;`» і «`&`» мають рівні пріоритети, менші, ніж у «`&&`» і «`||`». Пріоритети останніх є рівними між собою. Символ «`;`» означає, що команди будуть виконуватися послідовно, а «`&`» – паралельно. Операція «`&&`» («`||`») означає, що список, наступний за нею, буде виконуватися лише в тому випадку, якщо код завершення попереднього конвеєра нульовий (ненульовий).

Команда – це або проста команда, або одна з управляючих конструкцій. Кодом завершення команди є код завершення її останньої простої команди.

Умовний оператор

```
if умова1
   then
        список1
 [ elif умова2
   then
        список2 ]
   ...
 [ else
        список3 ]
fi
```

Виконується *умова1* і, якщо код її завершення 0, то виконується *список1*, інакше – *умова2* і, якщо і її код завершення 0, то виконується *список2*. Якщо ж це не так, то виконується *список3*. Частини `elif` і `else` можуть бути відсутні.

У ролі *умов* можуть використовуватися звичайні команди, однак найбільш часто – це виклик однієї або декількох команд `test` у вигляді `test` *вираз* або просто `[` *вираз* `]`. Пробіли, що відокремлюють вираз від квадратних дужок, обов'язкові. Детально команда `test` розглянута далі.

Приклад. Програма, що викликається з двома параметрами і визначає їх суму:

```
#!/bin/bash
n=$#                # визначаємо кількість переданих параметрів
if [ $n -eq 0 ]     # якщо не передано жодного
then
     echo Input a b # виводимо запрошення до вводу параметрів
     read a b       # читаємо параметри в змінні a і b
elif [ $n -eq 1 ]   # інакше, якщо переданий один параметр
then
     a=$1           # змінній a надаємо його значення
     echo Input b   # виводимо запрошення до введення другого
                    # параметра
     read b         # читаємо параметр в змінну b
elif [ $n -eq 2 ]   # інакше, якщо передані обидва параметри
then
     a=$1      # змінній a надаємо значення першого параметра
     b=$2      # змінній b надаємо значення другого параметра
else           # інакше (якщо не передано жодного параметра)
     echo Usage: $0 [a [b]]   # виводимо довідку по роботі
                              # програми
     exit           # і виходимо з програми
fi                  # кінець умовного оператора
s=$((a+b))          # або s=`expr $a + $b`   або let s=a+b –
                 # обчислюємо суму чисел, помістивши її в змінну s
echo sum=$s         # виводимо результат
```

Команда test

Команда `test` *вираз* або просто `[` *вираз* `]` застосовується для перевірки умови. Вона обчислює *вираз* і, якщо його значення істина, повертає код завершення 0; інакше – ненульове значення. Ненульовий код завершення повертається і в тому випадку, якщо відсутні аргументи.

Командний інтерпретатор розпізнаватиме цю команду по відкриваючій дужці «`[`», як по слову, що відповідає команді `test`. Між дужками і умовою, яка в них міститься, обов'язково повинні бути пробіли. Пробіли повинні бути і між

значеннями і символом порівняння або операції, як і в команді `expr` (не плутати з протилежною вимогою для присвоєння значень змінним!).

У сценаріях командного інтерпретатору використовуються умови різних типів.

1) Умови перевірки файлів:

`-f file` - файл *file* є звичайним файлом;

`-d file` - файл *file* є каталогом;

`-s file` - файл *file* є спеціальним файлом;

`-r file` - є дозвіл на читання файлу *file*;

`-w file` - є дозвіл на запис в файл *file*;

`-s file` - файл *file* не порожній.

Приклад: вводячи з клавіатури командні рядки, в першому випадку отримаємо підтвердження (код завершення 0), а в другому – спростування (код завершення 1); **myText** – це ім'я існуючого файлу.

```
[ -f myText ] ; echo $?
0
[ -d myText] ; echo $?
1
```

2) Умови перевірки рядків:

str1 = str2 - рядки *str1* і *str2* збігаються;

str1 ! = str2 - рядки *str1* і *str2* не збігаються;

`-n str1` - рядок *str1* існує (непорожній);

`-z str1` - рядок *str1* не існує (порожній).

Приклади:

```
x="who is who"; export x; [ "who is who" = "$x" ]; echo $?
0
x=abc ; export x ; [ abc = "$x" ] ; echo $?
0
x=abc ; export x ; [ -n "$x" ] ; echo $?
0
x="" ; export x ; [ -n "$x" ] ; echo $?
1
```

Команда `test` повертає значення «істина» (код завершення 0), якщо в дужках стоїть просто непорожнє слово:

```
[ hello ] ; echo $?
0
[ ] ; echo $?
1
```

Крім того, існують два стандартних значення умови, які можуть використовуватися замість умови (для цього не потрібні дужки):

```
true ; echo $?
0
false ; echo $?
1
```

3) Умови порівняння цілих чисел:

x -eq *y* – *x* дорівнює *y*;

x -ne *y* – *x* не дорівнює *y*;

x -gt *y* – *x* більше *y*;

x -ge *y* – *x* більше або дорівнює *y*;

x -lt *y* – *x* менше *y*;

x -le *y* – *x* менше або дорівнює *y*.

Команда `test` сприймає рядки символів як цілі числа. Тому у всіх інших випадках нульовому значенню відповідає порожній рядок. В даному ж випадку, якщо треба обнулити змінну `x`, це досягається присвоєнням `x=0`.

Приклади:

```
x=abc ; export x ; [ abc -eq "$x" ] ; echo $?
"[": integer expression expected before -eq
x=321 ; export x ; [ 321 -eq "$x" ] ; echo $?
0
x=3.21 ; export x ; [ 3.21 -eq "$x" ] ; echo $?
"[": integer expression expected before -eq
x=321 ; export x ; [ 123 -lt "$x" ] ; echo $?
0
```

Складні умови реалізуються за допомогою типових логічних операцій:

`!` інвертує значення коду завершення;

`-o` відповідає логічному АБО;

`-a` відповідає логічному І.

Приклади:

```
[ ! privet ] ; echo $?
1
x=privet; export x; [ "$x" -a -f specific ] ; echo $?
0
x="";export x; [ "$x" -a -f specific ] ; echo $?
1
x="";export x; [ "$x" -a -f specific -o privet ] ; echo $?
0
```

```
x="";export x; [ "$x" -a -f specific -o ! privet ] ; echo $?
1
```

Оператор вибору

```
case змінна in
шаблон11 | шаблон12 ...) список1 ;;
шаблон21 | шаблон22 ...) список2 ;;
...
esac
```

Оператор вибору виконує *список*, що відповідає першому шаблону, якому задовольняє *змінна*. Форма шаблону та ж, що і використовувана в іменах файлів. Частина `| шаблон ...` може бути відсутньою.

Приклад. Ввести назву фрукта і в залежності від введеного значення вивести його опис, а якщо не введено нічого, то вивести відповідне повідомлення:

```
#!/bin/bash
echo Input FRUIT # виводимо запрошення до введення назви фрукта
read FRUIT       # читаем параметр в змінну FRUIT
case $FRUIT in   # умова
     apple) echo Apple is red ;;        # альтернатива 1
     banana) echo Banana is yellow ;;   # альтернатива 2
     plum) echo Plum is blue ;;         # альтернатива 3
     *) echo Nothing to say about $1 ;; # інакше
esac
```

Оператори циклу

Мова командного інтерпретатора дозволяє організовувати циклічне виконання команд за допомогою операторів `while`, `until`, `for` і `select`.

Оператор циклу **`while`** має наступний синтаксис:

```
while умова
do
     список
done
```

При виконанні циклу спочатку виконується список команд, що представляють *умову*. Якщо результат ненульовий, то відбувається вихід з циклу, в іншому випадку виконується тіло циклу (*список*) і відбувається перехід на наступну ітерацію. Хоча як *умова* може використовуватися будь-яка UNIX-команда, найчастіше це команда `test`.

Приклад сценарію, в якому здійснюється виведення на термінал десяти послідовних чисел від 0 до 9:

```
#!/bin/sh
x=0
while [ $x –lt 10 ]      # значення змінної x менше 10?
do
     echo $x             # виводимо x
     x=`expr $x + 1`     # збільшуємо x на 1
done
```

У наступному прикладі в файл **log** кожну хвилину записуються поточні дата і час:

```
#!/bin/bash
while (true)
do
     date >> log
     sleep 60       # почекати хвилину
done
```

Оператор циклу **until** аналогічний оператору while і має наступний синтаксис:

```
until умова
do
     список
done
```

Відмінність між циклами while і until полягає в тому, що результат, який повертається при виконанні списку команд *умови*, береться з запереченням: *список* виконується в тому випадку, якщо остання команда *умови* повертає ненульовий статус виходу.

Нижче наведений приклад сценарію, в якому здійснюється видача на термінал десяти послідовних чисел від 0 до 9, але вже з використанням циклу until.

```
#!/bin/sh
x=0
until [ $x -ge 10 ]      # значення x більше або дорівнює 10?
do
     echo $x             # виводимо x
     x=`expr $x + 1`     # збільшуємо x на 1
done
```

Оператор циклу **for** виконує команди зі *списку* для кожного елемента і має наступний синтаксис:

```
for змінна [in елемент1 елемент2 ... елементN]
do
     список
done
```

Оператор `for` працює не так, як в звичайних мовах програмування. Замість того щоб при кожному проході циклу організовувати збільшення або зменшення на одиницю значення деякої змінної, він при кожному проході циклу присвоює змінній чергове значення із заданого списку елементів. Як *елементи* можна використовувати шаблони.

Якщо конструкція `[in` *елемент1* `елемент2 ... елементN]` відсутня, то *список* команд виконується один раз для кожного заданого позиційного параметра.

Приклад використання циклу `for`:

```
#!/bin/sh
for FILE in $HOME/*.txt
do
      cp $FILE ${HOME}/texts
      chmod a+r ${HOME}/texts/${FILE}
done
```

У наведеному прикладі всі файли з домашнього каталогу користувача, імена яких закінчуються на .txt, копіюються в каталог **texts** і робляться доступними для читання всім користувачам.

Мова програмування командного інтерпретатора містить оператори, які порушують нормальне виконання циклу. Це оператори `break` (дозволяє виконати безумовний вихід із циклу) та `continue` (призводить до того, що програма негайно переходить до наступної ітерації циклу без виконання інших команд в циклі).

Оператор циклу **`select`** дозволяє створювати зручні меню. Він корисний, коли необхідно, щоб користувач вибрав один елемент із запропонованого списку. Оператор `select` має такий же синтаксис, як і оператор `for`:

```
select змінна [in елемент1 елемент2 ... елементN]
do
      список
done
```

При виконанні даного оператора циклу всі елементи зі списку видаються на екран разом з їх порядковими номерами, після чого з'являється спеціальне запрошення для введення. Зазвичай воно має вигляд «`#?`». Введений користувачем номер пункту меню записується в змінну `REPLY`, а в *змінну* заноситься значення відповідного елемента зі списку елементів. Якщо введений порожній рядок, то список для вибору буде виданий заново. Після того як користувач зробить вибір, виконаються команди зі *списку*, після чого виконання циклу повторюється з самого початку. Вихід з циклу здійснюється тими ж засобами, що і для `while` та `for`.

Приклад:

```
#!/bin/bash
echo What Linux do you prefer?
select name in "Fedora" "Ubuntu" "Slackware" "Other"; do
break
done
if [ $REPLY == 1 ]
     then
          echo You chose $name. Good choice!
     elif [ $REPLY == 2 -o $REPLY == 3 -o $REPLY == 4 ]
          then
               echo You chose $name.
     else
          echo You must choose the right number!
fi
```

Після запуску цього сценарію на екран буде виведений наступний запит:

```
What operating system do you prefer?
1) Fedora
2) Ubuntu
3) Slackware
4) Other
#?
```

Інтерпретатор очікує введення номеру пункту меню на стандартному введенні. При натисканні будь-якої з чотирьох запропонованих цифр (1, 2, 3 або 4) можна побачити відповідне повідомлення. Якщо ввести 1, то буде виведене повідомлення

```
You chose Fedora. Good choice!
```

При введенні цифри, відмінної від 1, 2, 3 або 4, буде виведене повідомлення

```
You must choose the right number!
```

Контрольні питання

1) Що являє собою сценарій?

2) Якими способами можна виконати запуск сценарію?

3) Що таке змінна? Які типи змінних мови програмування командного інтерпретатора Вам відомі?

4) Що визначає область видимості змінної? Наведіть приклади.

5) Які управляючі конструкції мови програмування командного інтерпретатора Ви знаєте?

6) Чим відрізняються один від одного оператори циклу мови програмування командного інтерпретатора?

7) Які засоби введення/виведення використовуються в сценаріях?

8) Як обчислюються арифметичні вирази за допомогою мови програмування командного інтерпретатора?

Завдання до виконання

1) Скласти сценарій, який з командного рядка приймає три цілочисельних аргументи (a, b і c) і виводить значення (a+b)/c. Якщо сценарій запускається з одним, двома параметрами або взагалі без них, необхідно передбачити можливість введення відсутніх параметрів.

2) Скласти сценарій, який приймає два цілочисельних аргументи і виводить номер більшого з них.

3) Скласти сценарій, який чекає введення будь-якого рядка і після цього замінює в введеному рядку всі літери b на літеру p, літери p на літеру b, літери c на літеру k, а літери k – на літеру c.

4) Написати сценарій, який по введеному реєстраційному імені користувача визначає, чи зареєстрований даний користувач в системі, і видає відповідне повідомлення.

5) Написати сценарій, який в заданому каталозі (шляхове ім'я каталогу передається в якості аргументу сценарію) визначає імена файлів, що містять найбільшу кількість рядків, і виводить їх на екран.

6) Скласти сценарій, який серед переданих йому цілочисельних аргументів визначає кількість позитивних і негативних чисел та нулів.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна література

1. Christopher Negus. Linux Bible, 10th Edition. – Wiley, 2020. – 928 p.
2. Evi Nemeth, Garth Snyder, Trent R. Hein, Ben Whaley, Dan Mackin. UNIX and Linux System Administration Handbook, 4th edition. – Addison-Wesley, 2017. – 1885 p.
3. Горбань Г.В., Кандиба І. О. Операційна система Linux: навчальний посібник. – Миколаїв: Вид-во ЧНУ ім. Петра Могили, 2019. – 276 с.

Допоміжна література

1. Ellen Siever, Stephen Figgins, Robert Love, Arnold Robbins. Linux in a Nutshell, 6th Edition. – O'Reilly, 2009. – 944 p.
2. Arnold Robbins. Linux Programming by Example: The Fundamentals. – Prentice Hall, 2004. – 720 p.
3. David Tansley. Linux and Unix Shell Programming. – Addison-Wesley Professional, 1999. – 528 p.
4. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2010. – 656 с.
5. Семеренко В. П., Крилик Л. В. Операційна система Linux. – Навчальний посібник. – Вінниця: ВНТУ, 2006. – 88 с.

ЕЛЕКТРОННІ ІНФОРМАЦІЙНІ РЕСУРСИ

1. The UNIX® Standard | The Open Group Website – Режим доступу: http://www.opengroup.org/membership/forums/platform/unix
2. Linux.com – Режим доступу: https://www.linux.com/
3. Linux.org – Режим доступу: https://www.linux.org/
4. Enterprise Open Source And Linux | Ubuntu – Режим доступу: http://ubuntu.com/
5. Путівник по Linux – Режим доступу: http://linuxguide.rozh2sch.org.ua/

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

Навчальне видання

ОПЕРАЦІЙНІ СИСТЕМИ І СЕРЕДОВИЩА

Методичні вказівки
до виконання лабораторних робіт
для студентів факультету математики, фізики та
інформаційних технологій
першого (бакалаврського) рівня освіти,
спеціальності 126 «Інформаційні системи та технології»

Укладачі
Розновець Ольга Ігорівна
Трубіна Наталія Федорівна

В авторській редакції

Підписано до друку 30.10.2023 р. Формат 60x84/16.
Папір офсетний. Гарнітура Times. Цифровий друк.
Ум. друк. арк. 4,88. Наклад 30. Зам. № 1123-0905.
Віддруковано з готового оригінал-макета.

Видавництво та друк: ОЛДІ+
65101, Україна, м. Одеса, вул. Інглезі, 6/1
Свідоцтво ДК № 7642 від 29.07.2022 р.

Тел.: +38 (098) 559-45-45,
+38 (095) 559-45-45, +38 (093) 559-45-45
Для листування: 65101, Україна, м. Одеса, вул. Інглезі, 6/1
E-mail: office@oldiplus.ua