ЗМІСТ

ВВЕДЕННЯ

UNIX - одна з найпопулярніших в світі операційних систем завдяки тому, що її супроводжує і розповсюджує велике число компаній. Спочатку вона була створена як багатозадачна система для мінікомп'ютерів і мейнфреймів в середині 70-их років, але з тих пір вона виросла в одну з найпоширеніших операційних систем.

Існує багато різних версій Unix. Умовно ці версії можна розділити на два напрями - комерційні системи і некомерційні. Однією з некомерційних гілок Unix є Linux. Ця система спочатку була розроблена Лінусом Торвальдсом в Університеті Хельсінкі (Фінляндія). ОС Linux не пов'язана з конкретною моделлю комп'ютерів. Її ядро реалізовано на мові високого рівня (мові СІ), що дозволяє досить легко переносити цю систему з однієї платформи на іншу. Будучи системою з відкритим кодом, Linux успішно застосовується в учбовому процесі в освітніх установах всього світу. На сьогоднішній момент Linux - найсучасніша, стійка система, що швидко розвивається - майже миттєво вбирає в себе самі останні технологічні новини.

Цикл лабораторних робіт курсу «програмування в UNIX» направлений на вивчення і роботу в системі Linux, здобуття основ програмування в командному інтерпретаторі Bash.

Мета методичних вказівок - навчити студента працювати в середовищі Linux, прищепити навики роботи з різними командами цієї операційної системи. Методичні вказівки містять теоретичні відомості по вказаних темах і практичні роботи, в якості закріплення набутого теоретичного і практичного матеріалу по кожній темі приведені контрольні питання.

Правила техніки безпеки і охорони праці

Згідно з «Правилами техніки безпеки в лабораторіях інформатики» студентам забороняється:

- з`являтися та знаходитись приміщенні в нетверезому стані;
- ставити поруч з клавіатурою ємності з рідиною;
- перебувати в приміщенні в верхній одежі та завалювати нею робочі столи та стільці;
- працювати в лабораторії більше 6-ти годин на день (для вагітних жінок – більше 4-х годин);
- за власною ініціативою змінювати закріплені за ними робочі місця та знаходитись в приміщенні під час роботи іншої учбової групи;
- самостійно виконувати вмикання електроживлення лабораторії та заміну складових частин ПК, що вийшли із ладу.

У випадку виявлення несправностей обчислювальної техніки студент повинен сповістити про це викладача чи будь-кого з навчально-допоміжного персоналу лабораторії.

Лабораторна робота №1

Робота з файлами і каталогами

Мета роботи: освоїти основні принципи роботи в операційній системі Linux - вивчити основні команди і одержати практичні навики роботи в операційній системі Linux.

Постановка задачі: виконати необхідні дії, використовую відповідні команди операційної системи Linux.

Теоретичні відомості

Поняття файлу і файлової системи

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

Під файлами розуміється логічно зв'язана сукупність даних, що асоційована з носієм інформації і зовнішнім пристроєм.

Кожний файл характеризується деякою кількістю атрибутів. Типовий набір атрибутів включає наступні:

- Ім'я. Символьне ім'я файлу є єдиним атрибутом, що зберігся у формі придатної для читання людиною.
- Тип. Інформація, яка необхідна для управління файлами.
- Розташування. Покажчик на зовнішній пристрій і на місцезнаходження файлу на цьому зовнішньому пристрої.
- Розмір. Поточний розмір (в байтах, словах або блоках) і, можливо, максимально допустимий розмір.
- Захист. Інформація управління доступом, що управляє рівнем доступу (читання, запис, виконання і т.д.)
- Час, дата і ідентифікатор користувача. Ця інформація може характеризувати: створення, останню модифікацію, останнє звертання.

Файлова система зберігає інформацію про кожний файл в структурі, що зветься індексним дескриптором. Кожний індексний дескриптор містить близько 40 полів, у тому числі ім'я файлу, тип, розмір, кількість жорстких посилань, інформацію про власника файлу, про права доступу до нього, дату/час останньої модифікації файлу і доступу до нього. Саме з індексним дескриптором працює ОС при зверненні до файлу.

Типи файлів

У операційній системі Unix підтримуються декілька типів файлів:

– Звичайні файли. Це просто послідовність байтів; на структуру таких файлів не накладається жодних обмежень – це можуть бути текстові документи, виконувані програми, мультимедійні дані.

– Каталоги. Каталог - це файл, що містить імена файлів, які знаходяться в ньому, а також покажчики на додаткову інформацію - метадані,

що дозволяють операційній системі проводити операції над цими файлами. Каталоги визначають положення файлу в дереві файлової системи, оскільки сам файл не містить інформації про своє місцезнаходження.

– Посилання. Діляться на 2 типи:

a) «жорсткі» посилання;
b) символічні посилання («м'які»).

– Спеціальні файли пристроїв. Файли пристроїв дозволяють Unix-програмам взаємодіяти з апаратними засобами і периферійними пристроями системи. В UNIX розрізняють символьні і блочні файли пристроїв. Символьні файли пристроїв використовуються для небуферизованного обміну даними з пристроєм, в протилежність цьому блочні файли дозволяють проводити обмін даними у вигляді пакетів фіксованої довжини - блоків.

– Сокети. Сокети інкапсулюють з'єднання між процесами, дозволяючи їм взаємодіяти, не підпадаючи під вплив інших процесів. Звернення до них здійснюється через відповідні об'єкти файлової системи. Сокети є віртуальним об'єктом, що існує, поки на нього посилається хоча б один з процесів.

– Іменовані канали. Подібно сокетам, іменовані канали забезпечують взаємодію двох процесів, що виконуються на одному комп'ютері.

Структура файлової системи Unix

Файлова система ОС Unix має ієрархічну структуру (рис.1.1), основою якої є кореневий каталог, який має ім'я /. Кореневий каталог файлової системи Unix завжди один. Розташування файлів у файловому дереві не визначається їх розташуванням на тому або іншому фізичному або логічному диску. Файлові структури, що знаходяться на різних дисках, у тому числі на дисках інших комп'ютерів, за допомогою спеціальної команди (`mount`) вмонтовуються на файлове дерево Unix, стаючи частиною єдиного файлового дерева.

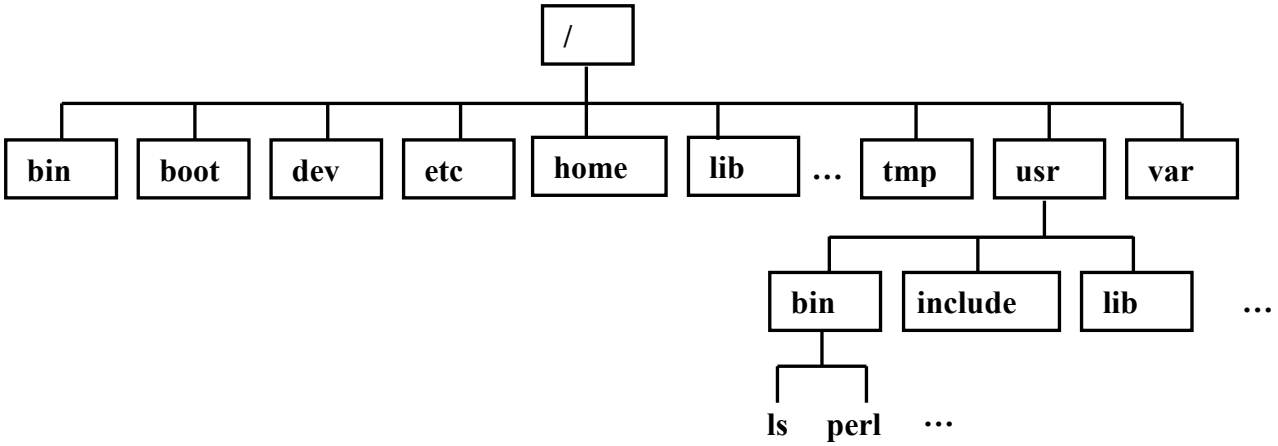


Рисунок 1.1 – Ієрархічна структура файлової системи ОС Unix

Кореневий каталог Unix звичайно містить такі каталоги, як:

bin для команд, що найбільш використовуються;

boot завантажувач операційної системи;

dev для спеціальних файлів, що представляють пристрої (дисплеї, диски, CD-ROM, принтери і т.п.);

etc для зберігання більшості конфігураційних файлів системи;

home для зберігання домашніх каталогів користувачів;

lib найважливіші бібліотеки;

mnt для підключення нових файлових систем;

tmp для зберігання тимчасових файлів;

usr містить каталоги і звичайні файли, що містять інформацію, що залучаються при рішенні задач користувача, а також більшість стандартних програм;

var містить буферні каталоги, файли реєстрації, облікову інформацію, бази даних і інші дані, які швидко розростаються і змінюються.

Послідовність імен каталогів, які розділені символом «/», що веде від деякого каталогу до каталогу, в якому розташовується даний файл, називається маршрутом до цього файлу. Послідовність ім'я_маршруту/ім'я_файлу називається *путнім ім'ям файлу*. Якщо путнє ім'я відлічується від кореневого каталогу, воно називається абсолютним (повним), інакше – відносним.

Існує два спеціальні імені:

. - це ім'я поточного каталогу;

.. - це ім'я батьківського каталогу (тобто каталогу, що знаходиться на рівень вище даного на шляху до кореневого каталогу).

В якості імен файлів, як правило, може використовуватися будь-яка послідовність з букв, цифр і спецсимволів. Довжина імені обмежується 256 символами. Прописні і рядкові букви в іменах файлів розрізняються. Ім'я файлу може включати суфікс, що звичайно використовується для вказівки на тип файлу. Файли, імена яких починаються з крапки, є прихованими.

Є декілька символів, допустимих в іменах файлів і каталогів, які потрібно використовувати з обережністю. Це так звані спецсимволи: «*», «\», «&», «<«, «>« «;» «(», «)», «|», а також символи пропуску і табуляції. Річ у тому, що ці символи мають спеціальне значення, щоб вони сприймалися як частина імені файлу або каталогу, необхідно передувати символом «\» (зворотний слеш).

Шаблони підстановки

Використовуючи спеціальні символи, можна формувати шаблони імен файлів:

** відповідає будь-якій (можливо, пустій) послідовності символів

? відповідає точно одному будь-якому символу

[] використовуються для групування символів в набори. Набори можуть задаватися:

– явним переліком символів, без вказівки роздільників між

ними, наприклад [at56,=] – символ відповідає або однієї з букв a і t, або однієї з цифр 5 і 6, або одному із спецсимволів кома і знак рівності;

– шляхом вказівки діапазону, наприклад [a-z] – символ відповідає рядковій букві від a до z;

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

! використовується для заперечення набору символів, тобто ім'я файлу не повинне містити вказаних символів.

Наприклад,

```
*f*          відповідає файлам з іменами, що містять букву f ;
program.?    відповідає файлам з іменами program, які мають
             однобуквений суфікс
??[a-d]*     відповідає файлам з іменами, в яких третьою бук-
             вою є a, b, c або d
```

Якщо в ім'я файлу повинен входити який-небудь спецсимвол, то при вказівці шаблона цьому спецсимволу необхідно передувати за допомогою зворотного слеша («\»), або екранувати, наприклад:

```
                  відповідає файлам з іменами, що починаються
[AEIOUYaeiouy]\*??? з голосної букви, за якою слідує символ *,
                  а за нею – три довільні символи
```

Командний рядок має наступний вигляд

```
команда ключі аргументи
```

Командна оболонка обробляє команди трьох типів. По-перше, в якості імені команди може бути вказано ім'я виконуваного файлу в об'єктному коді. По-друге, ім'ям команди може бути ім'я командного файлу, що містить набір інструкцій, які оброблюються командною оболонкою (такі файли називаються сценаріями або скриптами). По-третє, команда може бути внутрішньою командою мови shell.

Внутрішні команди мови shell є програмами, що знаходяться в каталозі /usr/bin. Більшість з них має наступний формат:

```
ім'я_команди ключі аргументи
```

Ключі служать для модифікації режимів роботи команди і є наборами певних символів, яким передує символ «-» (короткі імена ключів) або комбінація «--» (довгі імена ключів). Частіше за все в якості короткого ім'я ключа використовується початкова буква дії, яка визначається даним ключем, наприклад ключ -u – це скорочення від user.

Властивість ключа бути, з одного боку, гранично коротким, а з другого боку – інформативним, називається аббревіативністю. Не лише ключі, але і імена найпоширеніших команд Unix володіють цією властивістю.

Аргументи команди (або параметри команди) вказують на об'єкти, над якими виконуються операції. Аргументами команд в більшості випадків є імена файлів. Наприклад, приведена нижче команда означає: «Вико-

нати команду `ls` (відображення інформації про файл) з ключем `-l` (докладна інформація) для файлу a.out»:

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

Якщо необхідно використовувати два і більш однобуквених ключа, більшість команд дозволяє об'єднувати їх. Наприклад, дві приведені нижче команди ідентичні:

```
ls -lg a.out
1s -l -g a.out
```

Деякі ключі вимагають наявності параметра. В цьому випадку параметр дається після ключа, в цьому випадку останній не можна об'єднувати з іншим ключем.

У одному рядку може міститися декілька команд. Вони відокремлюються один від одного символом «;».

Командний процесор shell шукає імена команд у вказаному наборі каталогів, який можна змінити за бажанням користувача. Список цих каталогів є значенням змінної середи PATH.

Здобуття довідки про команди Unix

Одержати довідку про роботу команд Unix можна кількома способами.

Сторінки керівництва (manpages) містять більш всього корисної інформації. Кожна сторінка керівництва присвячена якому-небудь одному об'єкту системи. Щоб подивитися сторінку керівництва по якій-небудь команді, потрібно дати команду `man [номер_розділу] [ім'я_об'єкту]`.

Команда man сама знаходить розділ з необхідною довідкою, проглядаючи всі розділи по черзі, тому для здобуття довідки по команді find досить ввести `man find`.

Розділ керівництва займає, як правило, більше однієї сторінки екрану. Сторінки перегортаються вниз за допомогою клавіші пропуску або клавіші [Page Down], вгору – за допомогою клавіші [Page]. Вихід здійснюється по натисненню клавіші [q] (Quit).

Використовування ключів -h і --help

У кожної команди Unix є ключ -h і еквівалентний йому ключ --help, і користувач, який запустить команду з одним з цих ключів, може прогляну-ти коротку довідку про використовування даної команди.

Домашні каталоги користувачів

У Unix у кожного користувача обов'язково є власний каталог, який і стає поточним відразу після реєстрації в системі – домашній каталог. Домашні каталоги користувачів зберігаються в каталозі /home. Наприклад, для користувача anna домашнім каталогом є `/home/anna`.

Домашній каталог – це каталог, який призначений для зберігання власних даних користувача Unix. Як правило, домашній каталог є поточним безпосередньо після реєстрації користувача в системі. Повний шлях до домашнього каталогу зберігається в змінній середи HOME.

Операції над файлами і каталогами

Для визначення імені поточного каталогу використовують команду `pwd`, що викликається без параметрів і повертає повний шлях поточного каталогу, в якому знаходиться користувач.

Щоб проглянути вміст будь-якого каталогу використовують команду `ls`. Подана без параметрів, команда `ls` виводить список файлів і каталогів, що містяться в поточному каталозі. Команда `ls` приймає в якості параметру ім'я каталогу, вміст якого потрібно вивести. Ім'я може бути задано будь-яким доступним способом: у вигляді повного або відносного шляху.

Нижче приведені самі споживані 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;
-R --recursive	рекурсивно відображує вміст всіх каталогів;
-l, --format=long	виводить докладну інформацію про файли, включаючи тип файлу, права доступу, кількість жорстких посилань на нього, імена користувача і групи, розмір в байтах, дату/час останньої модифікації;
-S, --sort=size	сортує файли за розміром: найбільші файли йдуть першими;
-t, --sort=time	сортує файли за часом модифікації: найновіші файли йдуть першими;
--color	для розпізнавання типів файлів використовувати різні кольори;

Приклад, виведення докладної інформації про файли поточного каталогу, імена яких починаються і закінчуються на цифру із застосуванням різних кольорів:

```
ls -l ./[0-9]*[0-9] --color
```

Для зміни поточного каталогу командної оболонки використовується команда `cd`. Команда `cd` приймає один параметр: ім'я каталогу, в який потрібно переміститися, - зробити поточним. Щоб перейти в домашній каталог з довільної точки файлової системи досить виконати команду `cd ~` . Команда `cd` без параметрів, еквівалентна команді `cd ~`. За допомогою символу «~» можна посилатися і на домашні каталоги інших користувачів за допомогою конструкції `~ім'я_користувача`.

Для створення каталогу використовується команда `mkdir`. Вона застосовується з обов'язковим параметром: ім'ям створюваного каталогу (або списком імен каталогів, вказаних через пропуск). Якщо вказати перед ім'ям створюваного каталогу повний або відносний шлях до нього, то каталог буде створений відповідно до вказаної ієрархії каталогів:

```
mkdir /home/anna/examples
```

Створення файлу здійснюється за допомогою команди `touch`. Взагалі кажучи, дана команда використовується для зміни тимчасових міток доступу. Проте, спроба виконання даної команди над неіснуючим файлом приведе до його створення. Параметром команди `touch` є ім'я файлу (або список імен файлів через пропуск).

Для копіювання файлів і каталогів застосовується команда cp. Синтаксис команди `cp`:

```
cp[ключі]ім'я_існуючого_файлу/каталогу ім'я_цільового_файлу/каталогу
```

Якщо останній аргумент є ім'ям існуючого каталогу, то команда `cp` копіює кожний вказаний файл у файл з тим же ім'ям в цільовий каталог.

Якщо аргументами команди `cp` є імена двох файлів (каталогів), то команда `cp` копіює початковий файл (каталог) у файл (каталог), ім'я якого задано цільовим файлом (каталогом).

Деякі прапори команди `cp`:

-i --interactive вимагає підтвердження при перезаписі існуючих цільових файлів

-R --recursive копіює каталоги рекурсивно

Приклади:

```
cp -R prog dir            рекурсивне копіювання каталогу prog
                          в каталог dir
cp ~/labs/lab1 ~/copy     копіювання файлу ~/labs/lab1 в ката-
                          лог ~/copy
cp ~/labs/lab1 ~/copy/l1  копіювання файлу ~/labs/lab1 в ката-
                          лог ~/copy з новим ім'ям l1
```

Для переміщення файлів і каталогів застосовується команда `mv`. Переміщення файлу всередині однієї файлової системи рівнозначне його перейменуванню. Переміщення передбачає видалення посилання на файл з того каталогу, звідки він переміщений, і додавання посилання на цей самий файл в той каталог, куди він переміщений.

Синтаксис команди `mv`:

```
mv[ключі]ім'я_початкового_файлу/каталогу ім'я_цільового_файлу/каталог
```

Якщо останній аргумент є ім'ям існуючого каталогу, то команда `mv` переносить кожний вказаний початковий файл у файл з тим же ім'ям в цільовий каталог.

Якщо аргументами команди `mv` є два файли, то команда перейменовує початковий файл у файл, ім'я якого задано цільовим файлом.

Приклади:

```
mv /home/pet/1.ar arch  переміщення файлу /home/pet/1.ar в ката-
                        лог arch;
mv lab1 lab2 ~/old_files переміщення файлів  lab1, lab2 з поточ-
                        ного каталога в каталог ~/old_files;
mv file FILE            перейменування файлу file в FILE;
mv ~/labs/lab1 ~/copy/l1 переміщення файлу ~/labs/lab1 в каталог
                        ~/copy з новим ім'ям l1.
```

Створення посилань

Жорсткі посилання (hard links)

Користувач Unix може додати файлу ще одне ім'я (створити ще одне жорстке посилання на файл) за допомогою команди `ln`. Перший параметр цієї команди – це ім'я файлу, на який потрібно створити посилання, другий – ім'я нового посилання.

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

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

І файлу, і посиланню на нього відповідає один індексний дескриптор. Всі операції з файловою системою – створення, видалення і переміщення файлів – проводяться над індексними дескрипторами, а імена потрібні лише для того, щоб користувач міг орієнтуватися у файловій системі. Жорстке посилання є записом вигляду ім'я файлу + номер індексного дескриптора в каталозі. По суті, жорстке посилання задає ще одне ім'я для файлу.

Символічні (м'які) посилання

У жорстких посилань є два істотні обмеження:

1) Жорстке посилання може вказувати лише на файл, але не на каталог.

2) Неможливо створити на жорсткому диску жорстке посилання на файл, що розташований на знімному носії.

Щоб уникнути цих обмежень, були розроблені символічні (їх ще називають символьними) посилання. Символьне посилання (symbolic link) – це просто файл особливого типа, в якому міститься ім'я іншого файлу. Символьні посилання можуть вказувати і на каталог, чого не дозволяють жорсткі посилання. Символьні посилання називаються так тому, що містять символи – шлях до файлу або каталогу.

Символьне посилання можна створити за допомогою команди `ln` з ключем `-s`.

Приклад створення символьного посилання з ім'ям olga_file в домашньому каталозі користувача andrew на файл mylist.lst, який знаходиться в каталозі lists у в домашньому каталозі користувача olga:

```
ln -s /home/olga/lists/mylist.lst /home/andrew/olga_file
```

Символьне посилання цілком може містити ім'я неіснуючого файлу. В цьому випадку посилання існуватиме, але не працюватиме.

Видалення файлів і каталогів

Для видалення файлів призначена команда `rm ім'я_файлу/список_імен_файлів`. Вона призначена для видалення жорстких посилань, а не самих файлів. В Unix, щоб повністю видалити файл, вимагається послідовно видалити всі жорсткі посилання на нього. Поки є хоч одне посилання, файл продовжує існувати.

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`. Втім, `rmdir` видалить каталог лише в тому випадку, якщо він пустий: в ньому немає жодного файлу і підкаталогу. Видалити каталог разом зі всім його вмістом можна командою `rm` з ключем `-r` (--recursive).

Приклади:

`rmdir catalog tests`	видалення порожніх каталогів з іменами catalog, tests;
`rm -r /home/anna/tmp`	видалення каталога /home/anna/tmp зі всім його вмістом.

Проглядання вмісту файлів

Виведення вмісту файлу здійснюється декількома способами.

1. Команда `more ім'я _файлу/список_імен_файлів` виводить на екран вміст вказаних файлів, при цьому немає необхідності запускати текстовий редактор. Недолік цієї команди в тому, що неможливо перегорнути інформацію у зворотному напрямі.

2. Команда `less ім'я_файлу/список_імен_файлів` здійснює виведення вмісту одного або декількох вказаних файлів на екран і дозволяє проглядати його в обох напрямах. Повернення на попередню сторінку виконується після натиснення клавіші [b].

3. Команда `cat ім'я_файлу/список_імен_файлів` здійснює конкатенацію вмісту вказаних файлів і виведення його на екран. Її можна застосовувати для проглядання вмісту невеликих файлів.

Введення і виведення

Для того, щоб записати дані у файл або прочитати їх звідти, процесу необхідно спочатку відкрити цей файл. При цьому процес одержує дескриптор відкритого файлу – унікальне для цього процесу число, яке він і використовуватиме у всіх операціях доступу. Перший відкритий файл одержить дескриптор 0, другий – 1 і так далі. Закінчивши роботу з файлом, процес закриває його, при цьому дескриптор звільняється і може бути використаний повторно.

Кожний процес Unix одержує при старті три «файли», відкриті для нього системою. Перший з них (з дескриптором 0) відкритий на читання, це стандартне введення процесу. Саме із стандартним введенням працюють всі операції читання, якщо в них не вказаний дескриптор файлу. Другий «файл» (з дескриптором 1) відкритий на запис, це стандартне виведення процесу. Третій потік (з дескриптором 2) призначається для виведення діагностичних повідомлень, він називається стандартне виведення помилок.

Перенаправлення введення і виведення

У Unix стандартне введення здійснюється з клавіатури, стандартне виведення і стандартне виведення помилок направлені на термінал. Щоб було можливе записати вихідну інформацію будь-якої команди у файл, введення команді передати також з файлу, і помилки також можна записати у файл. Існують спеціальні оператори перенаправлення введення і виведення:

> файл Перенаправлення виведення. Помістити вихідну інформацію у

файл, а не посилати її на екран. Те, що знаходилося у файлі раніше, буде знищене.

>> файл Перенаправлення виведення. Дописати вихідну інформацію у файл слідом за його вмістом.

< файл Перенаправлення введення. Взяти вхідну інформацію з файлу, а не з клавіатури.

Приклади:

```
ls -l >> filelist  список файлів поточного каталогу дописати у файл filelist
cat f1 f2 f3 > f4  вміст файлів f1, f2 і f3 помістити у файл f4
sort < mylist.txt  сортування файлу mylist.txt (якщо викликати команду sort без параметра, вона читатиме рядки із стандартного введення)
```

Перенаправлення виведення помилок

Стандартне виведення помилок може бути перенаправлено так само, як і стандартне введення-виведення, для цього використовується комбінація символів «2>» або «2>>» . Наприклад, при виконанні команди

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

У випадку, якщо файл info не існує, помилка не буде видана на екран, а допишеться у файл cat.err.

Іноді, проте, вимагається об'єднати стандартне виведення і стандартне виведення помилок в одному файлі, а не розділяти їх. В командній оболонці bash для цього є спеціальна послідовність «2>&1». Це означає «спрямувати стандартне виведення помилок туди ж, куди і стандартне виведення»: `cat info >info_file 2>>&1`

У даному прикладі спочатку вказано, куди перенаправити стандартне виведення (>info_file) і тільки потім вказано спрямувати туди ж стандартне виведення помилок (2>>&1).

Номер в конструкції &номер – це номер відкритого дескриптора. Щоб не набирати громіздку конструкцію > файл 2>&1 в bash використовуються скорочення: &> файл або, що те ж саме >& файл.

Перенаправлення в нікуди

Іноді явно відомо, що якісь дані, виведені програмою, не знадобляться. Наприклад, попередження із стандартного виведення помилок. В цьому випадку можна перенаправити стандартне виведення помилок у файл пристрою, спеціально призначений для знищення даних – /dev/null. Цей пристрій називається порожнім або «сміттєвою корзиною». Все, що записується в цей файл, буде просто проігноровано:

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

Таким же чином можна позбавитися і стандартного виведення, відправивши його в /dev/null.

Конвеєр

Нерідко виникають ситуації, коли потрібно обробити виведення однієї програми іншою програмою. Для цього можна зберегти виведення од-

нієї програми у файлі, а потім спрямувати цей файл на введення іншій програмі. Те ж саме можна зробити більш ефективно: перенаправляти виведення можна безпосередньо на стандартне введення іншій програмі. В Unix такий спосіб передачі даних називається конвеєром.

Для перенаправлення стандартного виведення на стандартне введення іншій програмі служить символ «|». Наприклад, якщо виведення файлу дуже велике і «пролітає» на екрані, в цьому випадку можна спрямувати виведення в програму проглядання `less`, яка дозволить не поспішаючи перегорнути весь текст, повернутися до початку і т. п.:

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

Можна послідовно обробити дані декількома різними програмами, перенаправляючи виведення на введення наступній програмі і організовуючи скільки завгодно довгий конвеєр для обробки даних. В результаті виходять дуже довгі командні рядки вигляду cmd1 | cmd2 | ... | cmdN, які можуть показатися громіздкими і незручними, але виявляються дуже корисними і ефективними при обробці великої кількості інформації.

Деякі корисні команди

`echo` – виводить на стандартне виведення вказаний рядок символів;

`date` – викликана без параметрів, виводить поточну дату/час;

`who` – викликана без параметрів, виводить інформацію про користувачів, що в даний момент реєструвалися в системі (по стовпцях: ім'я користувача, термінал, час реєстрації, ім'я видаленого комп'ютера);

`users` – викликана без параметрів, виводить інформацію про користувачів, що в даний момент реєструвалися в системі;

`hostname` – викликана без параметрів, виводить інформацію про ім'я поточного мережного вузла;

`uname` - виводить інформацію про комп'ютер і операційну системі;

`wc` – використовується для підрахунку числа рядків, символів і слів у вказаних файлах або стандартному введенні, якщо ім'я файлу не задано. Якщо вказано більше одного файлу, виводяться їх імена і значення лічильників, а в кінці виведення виводиться підсумкова сума накопичених лічильників.

Прапори:

-c --chars	виводить лише кількість символів
-l --lines	виводить лише кількість рядків
-w --words	виводить лише кількість слів
-L --max-line-length	виводить лише довжину щонайдовшого рядка

За умовчанням команда викликається з прапорами -clw.

Приклади:

```
wc mytext  виводить кількість символів, рядків і слів у файлі mytext
wc –lL ~/texts/mytext  виводить кількість рядків і довжину щонайдовшого рядка файлу ~/texts/mytext
```

`tee` – одночасно копіює стандартне введення на стандартне виведення і у вказані файли. Якщо файли не існують, вони створюються. Якщо ж файл є, то він буде перезаписаний, якщо лише не вказаний прапор -a.

Приклади:

```
cat file | tee a b c              копіює вміст файлу file у файли a, b, c
                                  і виводить на екран
cat txt1 txt2 txt3 | tee copy     копіює з'єднаний вміст файлів  txt1, txt2
                                  і txt3 у  файл copy виводить на екран
```

Спеціальні символи

Деякі символи в shell мають спеціальне значення. Наприклад, шаблонні символи «[]», «*» «?», символ доступу до значення змінної «$», символи перенаправлення введення і виведення «>« і «<«, символ конвеєра «|», пропуск, що використовується як роздільник. Іноді виникає необхідність відмінити їх спеціальне значення, наприклад якщо ми хочемо використовувати один з цих символів в імені файлу. В цьому випадку таке ім'я слід укласти в подвійні лапки. Подвійні лапки відміняють дію всіх спецсимволів, крім «$» і «!». Наприклад, внаслідок виконання команди

```
cp ./f* /tmp
```

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

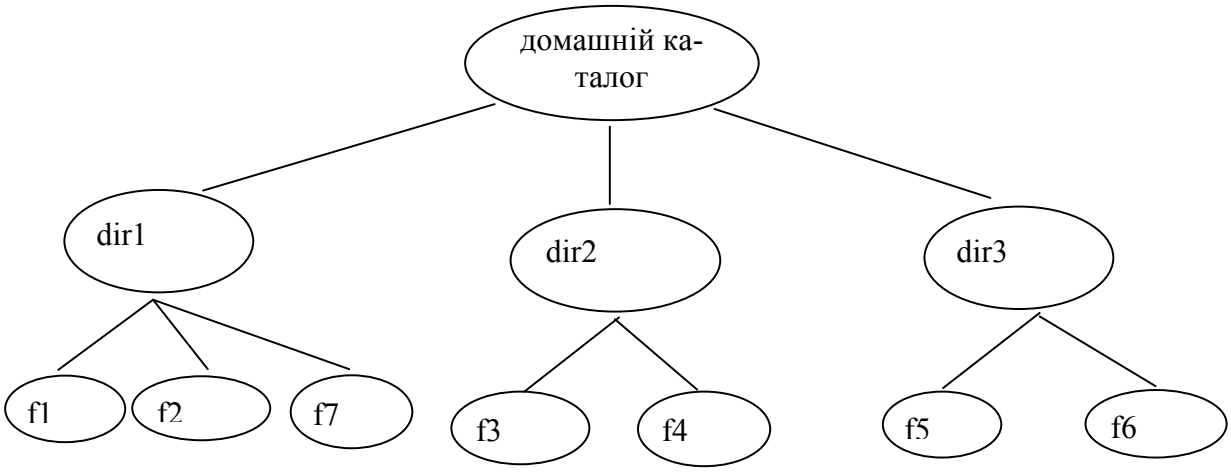
```
cp ./«f*» /tmp
```

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

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 ключі і аргументи командного рядка Unix?
5. Як здійснюється перенаправлення введення-виведення в Unix?
6. Які операції над файлами і каталогами в Unix Ви знаєте?

Завдання до виконання



1. Створити в домашньому каталозі три каталоги з іменами dir1, dir2, dir3.

2. У каталозі dir1 створити два файли з іменами f1 і f2.

3. У файл f1 помістити інформацію про користувачів, зареєстрованих в системі. У файл f2 – вміст домашнього каталогу (і всіх його підкаталогів), проглянути вміст f 1 і f2.

4. Скопіювати файл f1 в каталог dir2 з ім'ям f3. Для файлу f2 в каталозі dir2 створити жорстке посилання з ім'ям hardlink, а для f3 – м'яке з ім'ям softlink. М'яке посилання видалити, а жорстке перейменувати в f4.

5. У каталозі dir3 створити файл f5, в якому буде з'єднана інформація, що зберігається у файлах f1 і f2.

6. Відновити вміст файлу f2, помістивши туди замість раніше доданої інформації розширену інформацію про вміст домашнього каталогу (і всіх його підкаталогів). Проглянути вміст посторінково.

7. У каталозі dir3 створити файл f6, що містить наступне повідомлення:

Список вмісту домашнього каталогу

8. Додати в f6 поточну дату і вміст файлу f2. Проглянути його вміст. В каталозі dir1 створити файл f7, в якому зберігатиметься список файлів домашнього каталогу в наступному вигляді:

Файли, які починаються на букву f

[список]

Файли, які починаються на букву h

[список]

9. Файл f7 скопіювати в каталог dir4, при цьому повідомити про помилки, які записати у файл myerr. Переглянути. Повторити копіювання кілька разів, і при цьому у файлі myerr число записів повинне бути рівне числу спроб копіювання.

10. Проглянути вміст каталогу dir4, подавивши повідомлення про помилки.

11. Вміст файлу f1 скопіювати в f7 так, щоб вміст f1 був відображений на екрані.

12. Видалити файли f3 і f4. Видалити каталог dir2. Видалити вміст каталогу dir1 рекурсивно.

Лабораторна робота №2

Управління повноваженнями на файли і каталоги

Мета роботи: вивчити основні і спеціальні права доступу у файловій системі Unix, навчитися змінювати права доступу на файли і каталоги.

Постановка задачі: виконати завдання, використовуючи відповідні команди.

Теоретичні відомості

Права доступу

Процедура реєстрації в системі для Unix обов'язкова: працювати в системі, не реєструючись під тим або іншим ім'ям користувача, просто неможливо. Для кожного користувача визначена сфера його повноважень в системі: програми, які він може запускати, файли, які він має право переглядати, змінювати, видаляти. При спробі зробити щось, що виходить за рамки повноважень, користувач одержить повідомлення про помилку.

У розрахованій на багато користувачів моделі розділяються звичайні користувачі і адміністратори. В повноваження звичайного користувача входить все необхідне для виконання прикладних задач, проте йому заборонено виконувати дії, що змінюють саму систему. Повноваження адміністратора зазвичай не обмежені.

Облікові записи

Звичайно, система може бути «знайома» з людиною лише у переносному розумінні: в ній повинні зберігатися запис про користувача і системна інформація, яка з ним пов'язана – обліковий запис.

У обліковому записі містяться наступні відомості про користувача:

– Вхідне ім'я (login name) – назва облікового запису користувача, яка потрібна вводити при реєстрації в системі.

– Ідентифікатор користувача (UID – User IDentifier) –унікальне позитивне ціле число, що однозначно ідентифікує користувача в Unix.

– Ідентифікатор групи (GID – Group IDentifier). – Групи користувачів застосовуються для організації доступу декількох користувачів до деяких ресурсів. У групи, є ім'я і ідентифікаційний номер – GID. В Unix користувач повинен належати як мінімум до однієї групи – групі за умовчанням.

– Коментарій. – Крім вхідного імені в обліковому записі містяться відомості про користувача, що використовує даний обліковий запис. Зміст відомостей визначає адміністратор системи.

– Домашній каталог. – Файли всіх користувачів в Unix зберігаються роздільно – у власному домашньому каталозі. Доступ інших користувачів до домашнього каталогу користувача може бути обмежений. Інформація про домашній каталог обов'язково повинна бути присутня в обліковому записі.

– Командна оболонка. – Кожному користувачу потрібно надати спосіб взаємодії з системою: передача їй команд і здобуття від неї відповідей. Для цієї мети служить спеціальна програма – командна оболонка (або інтерпретатор командного рядка). Вона повинна бути запущена для кожного користувача, який реєструвався в системі і вказана в обліковому записі.

За допомогою команди `id ім'я_користувача` можна узнати реєстраційне ім'я користувача і відповідний йому UID, а також групу за умовчанням, її GID і повний список груп, членом яких він є.

Поняття «адміністратор»

У Unix є лише один користувач, повноваження якого в системі принципово відрізняються від повноважень решти користувачів – це користувач з ідентифікатором 0. Зазвичай обліковий запис користувача з UID=0 називається root. Користувач root – це «адміністратор» системи (суперкористувач), обліковий запис для root обов'язково присутній в будь-якій системі Unix, навіть якщо в ній немає жодного іншого облікового запису. Користувачу з таким UID дозволено виконувати будь-які дії в системі.

Права доступу у файловій системі Unix

Кажучи про права доступу користувача до файлів, варто помітити, що насправді маніпулює файлами не сам користувач, а запущений ним процес. Кожний процес системи обов'язково належить якому-небудь користувачу, і ідентифікатор користувача – обов'язкова властивість будь-якого процесу Unix. Коли програма login запускає стартовий командний інтерпретатор, вона приписує йому UID, одержаний внаслідок діалогу. Всі процеси, запущені користувачем під час термінальної сесії, матимуть його ідентифікатор.

Ключ `-l` програми `ls` визначає формат видачі (справа наліво): ім'я файлу, час останньої зміни файлу, розмір в байтах, група, власник, кількість жорстких посилань і строчка атрибутів. Наприклад:

```
drwxr-xr-x 2 methody methody 4096 Окт 31 15:21 examples
-rw-r--r-- 1 methody methody 26   Січн 22 15:21 loop
```

Перший символ в строчці атрибутів визначає тип файлу. Тип «-» відповідає звичному файлу, а тип «d» – каталогу, тип «l» - символічному посиланню, тип «s» - сокету, тип «p» - іменованому каналу.

Тепер більш докладно про те, чому відповідають дев'ять символів в рядку атрибутів, видаваною програмою `ls`. Ці дев'ять символів мають вид rwxrwxrwx, де деякі «r», «w» і «x» можуть замінюватися на «-». Очевидно, букви відображають прийняті в Unix три види доступу – читання (read), запис (write) і виконання (execute). В рядку атрибутів вони присутні у трьох екземплярах. Це тому, що будь-який користувач (процес) Unix по відношенню до будь-якого файлу може виступати в трьох ролях: як власник (user), як член групи, якій належить файл (group), і як сторонній користувач (other), жодних стосунків власності на цей файл не має. Рядок атрибутів – це три трійки rwx, що описують права доступу до файлу власника цього файлу (перша трійка), групи, якій належить файл (друга трійка) і

сторонніх користувачів (третя трійка). Якщо в якій-небудь трійці не вистачає букви, а замість неї стоїть «-», значить, користувачу у відповідній ролі буде у відповідному виді доступу відмовлено.

Права доступу зберігаються в 16-бітовому полі індексного дескриптора файлу. 12 бітів відведено під зберігання режиму доступу, а 4 біти, що залишилися, визначають тип файлу, який встановлюється при створенні файлу і не підлягає зміні (рис. 2.1).

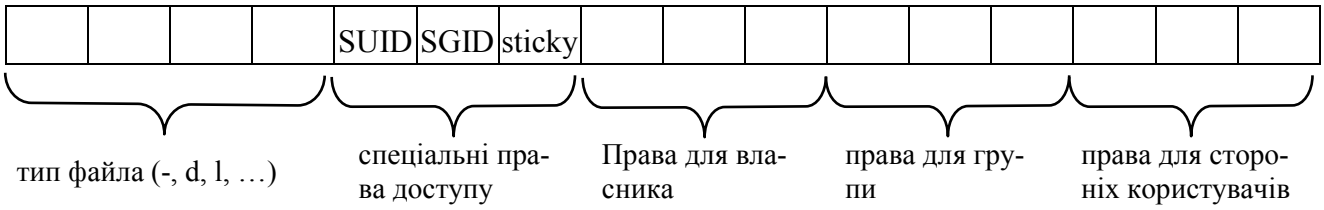


Рисунок 2.1 – 16-бітове поле індексного дескриптора файлу

Права доступу до файлів і каталогів дещо розрізняються:

Права доступу	Файл	Каталог
r (читання)	читання з файлу	здобуття списку файлів каталогу
w (запис)	запис у файл	зміна вмісту каталогу (створення і видалення файлів в ньому)
x (виконання)	виконання файлу, якщо він є сценарієм або програмою	вхід в каталог і здійснення в ньому пошуку

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

Спеціальні права доступу

Ідея, що лежить в основі вживання біта SUID (Set User ID – встановити ідентифікатор користувача), полягає в тому, що користувач, що запустив програму, для якої власник встановив біт SUID, на час виконання програми одержує всі права її власника. Той же принцип застосовний і до біта SGID (Set Group ID – встановити ідентифікатор групи), лише в даному випадку міняються привілеї групи, що володіє сценарієм.

Біт збереження задачі (sticky, від англ. «липучка») при установці для файлу вказує системі на необхідність зберегти копію програми в пам'яті після її завершення для прискорення подальших запусків цієї програми. Якщо ж sticky-біт встановлений для каталогу, то в цьому каталозі файли може видаляти або перейменовувати лише їх власник або суперкористувач, 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 суперкористувач. Для зміни прав доступу до файлів і каталогів застосовується команда `chmod`. Аргументи цієї команди можуть бути задані або в символьному, або в числовому (абсолютному) форматі.

Символьний формат

Синтаксис команди `chmod` при використовуванні символьного формату такий:

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

Параметр `хто` може приймати такі значення:

u – власник o – сторонні користувачі

g – група a – всі користувачі

Параметр `оператор` може приймати такі значення:

+ – додавання прав до тих, що є

- – видалення прав з тих, що є

= – установка прав замість тих, що є

Параметр `права` може приймати такі значення:

r – читання w – запис x – виконання

s – SUID або 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

Приклади (передбачається, що для файлу myfile встановлені права доступу читання, записи і виконання для всіх категорій користувачів і їх символьний запис має вид rwxrwxrwx):

Команда	Символьний запис прав доступу	Результат
chmod a-x myfile	rw-rw-rw-	Відібрати у всіх категорій користувачів право на виконання
chmod g+wx myfile	rwxrwxr--	Додати права на запис і виконання
chmod g=o myfile	rwxr--r--	Встановити для групи такі ж права доступу, як у сторонніх користувачів
chmod g+s myfile	rwxr-s---	Додати SGID
chmod uo+x myfile	-ws--x--t	Додати для власника і сторонніх користувачів право на виконання

Числовий (абсолютний) формат

Синтаксис команди `chmod` при використовуванні числового формату:

```
chmod права ім'я_файлу
```

Тут параметр `права` є восьмеричне число. В найпростішому випадку воно складається з трьох трьохбітових наборів, кожний з яких відноситься до однієї з категорій користувачів. Старший біт відповідає дозволу на читання (1 – встановлено, 0 – знято), середній – дозволу на запис, а молодший – дозволу на виконання.

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

Права доступу за умовчанням

При створенні нових файлів і каталогів встановлюються права доступу за умовчанням:

– для каталогів – 777 (rwxrwxrwx);
– для файлів – 666 (rw-rw-rw-).

Права доступу за умовчанням можна регулювати за допомогою команди `umask`. Призначена для користувача маска – це число, яке віднімається від прав доступу за умовчанням, а одержані в результаті віднімання права застосовуються до створюваного файлу (каталогу). Параметр команди `umask` – восьмеричне число, що задає атрибути, які не треба встановлювати новому файлу або каталогу. Задана без параметрів, ця команда виводить значення поточної встановленої маски.

Права доступу файлів, створених при конкретному значенні umask, обчислюються за допомогою наступних побітових операцій: побітове І між унарним доповненням аргументу (використовуючи побітове НЕ) і режимом повного доступу.

Наприклад:

```
umask 027
```

Для каталогів обчислити права, що вийшли, просто: права доступу за умовчанням (777) мінус значення маски (027) рівно 750.

Для файлів ця процедура трохи складніше. Спочатку запишемо двійкове представлення восьмеричного значення маски: `0278=0000101112` і прав доступу за умовчанням: `6668=1101101102`. Потім запишемо унарне доповнення значення маски: `111101000` (виходить інвертуванням бітів двійкового представлення значення маски). А зараз виконуємо побітове множення прав доступу за умовчанням і 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 у восьмеричну систему і маємо права доступу 640 або, в символьному вигляді, rw-r-----.

Приклади:

```
umask 0     файли створюватимуться з правами доступу rw-rw-rw-
            (666), каталоги – з правами доступу rwxrwxrwx (777)
umask 022   файли створюватимуться з правами доступу rw-r--r--
            (644), каталоги – з правами доступу rwxr-xr-x (755)
umask 077   файли створюватимуться з правами доступу rw-------
            (600), каталоги – з правами доступу rwx------ (700)
```

Контрольні питання

1. У чому полягає розрахована на багато користувачів модель розмежування доступу?
2. Які права доступу до файлів і каталогів Ви знаєте?
3. Що дає встановлення біта SUID?
4. Яке призначення команди umask?
5. Як змінити права доступу на файли і каталоги?

Завдання до виконання

1. По встановленій масці і значенню прав, одержаних при створенні файлів і каталогів, визначити спочатку встановлені в системі права. Описати їх в протоколі.
2. Задати маску, що дає можливість створювати файли і каталоги, доступні для всіх операцій лише власнику і стороннім користувачам. Перевірити дію маски.
3. У домашньому каталозі створити каталоги dir1 і dir2. Зробити dir1 загальнодоступним.
4. У dir1 створити підкаталог dir11 і вкладений в нього каталог dir111. Зробити dir111 темним (можливість лише створювати і видаляти свої файли і не бачити, що в ньому є ще).
5. Створити в dir11 два файли з іменами f1 і f2. За допомогою числового аргументу команди chmod призначити ним наступні права:

f1: g – читання, виконання o – нічого u – всі права

f2: g і o – виконання u – читання

6. У каталогах dir1 і dir2 створити файли з однаковими іменами: a454, a\a \4a, 44\?, F4a41, f442 ??12, abcdf, 1, 4, a ? \b, 44.
7. Прибрати право запису для всіх категорій користувачів на всі ті файли каталога dir1, імена яких містять не менше три символів.
8. Прибрати всі права доступу для сторонніх користувачів на всі ті файли каталога dir1, імена яких містять цифри.
9. Додати право запису для власника на всі ті файли каталогів dir1 і dir2, імена яких містять не менше дві цифр і не закінчуються на букву.
10. Додати право читання для групи і сторонніх користувачів на всі ті файли каталогів dir1 і dir2, імена яких містять символ \.
11. Створити зведену таблицю властивостей файлів вашого домашнього каталогу, відобразити його структуру у вигляді дерева.
12. Зробити закритим ваш домашній каталог.

Лабораторна робота № 3

Створення резервних копій і архівів

Мета роботи: вивчити основні програми архівації, стиснення і розпаковування файлів.

Постановка задачі: виконати завдання, використовуючи відповідні команди.

Теоретичні відомості

Програма архівації файлів tar

Програма `tar` призначена для архівації файлів і витягання їх з архіву. Під архівацією розуміється приміщення вмісту декількох файлів в один файл – так званий файл-архів, використовування якого спрощує зберігання і передачу файлів.

Команда для виконання програми `tar` має наступний синтаксис:

```
tar прапори ім'я_архіву ім'я_файлу/каталогу
```

Першим в командному рядку повинен йти один з прапорів, що позначають дію, виконувану командою: `-A, -c, -d, -r, -t, -u, -x`. Далі слідують необов'язкові прапори, що позначають спосіб виконання дії: `-w, -z, -v`. За ними слідує прапор -f, який указує, що далі слідує `ім'я архівного файлу`, над яким виконується дія. Після імені архівного файлу можуть йти додаткові прапори. При вказівці в якості ім'я архіву символу **«-»** запис даних здійснюється на стандартне виведення, що дозволяє використовувати команду `tar` в конвеєрах.

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

Прапори команди `tar`, що позначають дію:

-A --concatenate	додає файли в архів;
-c --create	створює новий архів;
-d --compare	знаходить відмінність між членами архіву і їх «ісходниками» на диску;
-r --append	додати файли в кінець архіву;
-t --list	виводить вміст архіву;
-u --update	додає в архів лише ті файли, які раніше не були включені в архів;
-x --extract	витягує файли з архіву.

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

-v --verbose	виводить список оброблюваних файлів;
-w --interactive	запрошує підтвердження для кожної дії;
-z --gzip	стискає/розпаковує файли за допомогою програми gzip;

-M --multi-volume працює з багатотомним архівом;
-T --files-from file указує узяти 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 їх в архів;
--exclude file виключає з обробки файл file;
-X -exclude-from file виключає з обробки файли, перераховані у файлі file.
--wildcards дозволяє використовувати в іменах файлів шаблони підстановки.

Приклади:

`tar –cfvW arc/bin.tar /bin` архівує каталог /bin у файл bin.tar, який знаходиться в підкаталозі arc поточного каталогу, при цьому виводить імена оброблюваних файлів з перевіркою цілісності

`tar –cfzw docs.tar a1 b2` поміщає в архів docs.tar файли поточного каталогу a1, b2 із стисненням за допомогою gzip і підтвердженням кожної виконуваної дії

Програма стиснення і 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 упаковки кожний файл заміщається стислою версією з таким же ім'ям, як у оригінального файлу, до якого додається розширення `.gz`. При цьому зберігаються власник файлу, група, права доступу, а також тимчасові мітки оригіналу. Програма `gzip` стискає лише звичайні файли, ігноруючи символічні посилання.

Прапори команди `gzip`:

-c --stdout спрямовує результат обробки на стандартне виведення, зберігаючи оригінальні файли незміненими;
-d --decompress виконує розпаковування файлів ;
-f --force перезаписує вихідні файли і стискає посилання;
-l--list виводить для кожного упакованого файлу інформацію про розмір упакованого і оригінального файлу, відсоток стиснення і т.д.;
-r --recursive рекурсивно виконує обробку файлів у всіх каталогах;
-S --suffix suffix вказує замість .gz використовувати вказаний суфікс suffix;
-t --teat перевіряє цілісність стислих файлів;
-v --verbose відображує в процесі обробки імена файлів і відсоток стиснення;

-1...-9 указує ступінь стиснення (-1 – мінімальне найшвидше, -9 – максимальне, найповільніше); за умовчанням прийнятий ступінь стиснення -6

Приклади:

`gzip –rv /home`	упакувати рекурсивно всі файли каталогу /home, при цьому вивести імена оброблюваних файлів
`gzip –d ar/a*.gz`	розпакувати файли каталогу ar, імена яких починаються на букву a
`gzip –c7 myfile > archive.gz`	упакувати файл myfile у файл archive.gz із ступенем стиснення 7 і збереженням оригінальних файлів

Програма розпаковування архівних файлів gunzip

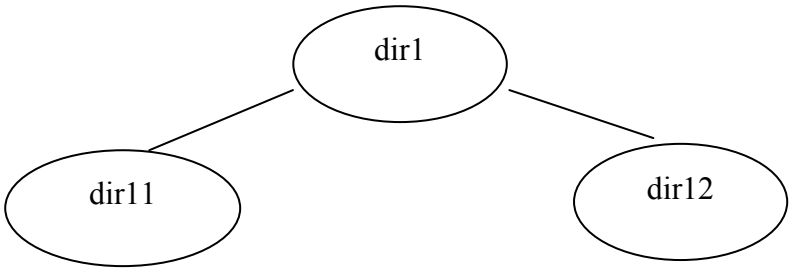
Програма `gunzip` розпаковує файли, раніше створені за допомогою програми `gzip`. При цьому вона заміщає стислі файли їх оригінальними версіями, видаляючи розширення з імен файлів.

Формат команди виклику програми `gunzip` і її прапори збігається з форматом і прапорами команди `gzip`.

Контрольні питання

1. Для чого призначена програма tar?
2. Для чого служить програма gzip?
3. Для чого призначена програма gunzip з ключем –l?

Завдання до виконання



> file1	dat1	> my	abc
file2	> dat2	> my1	321
file3	> dat	my2	> XyZy
gg	ii	> pp	1234Y
ggg	> eee	pPpP	> Pm

1. У домашньому каталозі створити каталог dir1. В ньому створити два підкаталоги – dir11 і dir12. Створити в цих каталогах файли (імена вказані на малюнку вище). В ті, які помічені знаком >, помістити довільну інформацію.

2. Створити архів arc1.tar для каталогу dir11 з видаленням оригінальних файлів, при цьому файли file1 і dat не повинні потрапити в архів. Проглянути вміст архіву.

3. Витягнути з архіву arc1.tar файли з іменами, що починаються на «d», в інтерактивному режимі. Витягнути з архіву файли з іменами, що починаються на «g», при цьому вивести список оброблюваних файлів.

4. Видалити з архіву arc1.tar файли, імена яких закінчуються на цифру.

5. Додати в архів arc2.tar файли з каталогу dir11, імена яких складаються з 5 символів. Проглянути вміст архіву.

6. Упакувати всі файли каталогу dir12 за допомогою команди gzip, при цьому відобразити імена і відсоток стиснення. Проглянути вміст архіву.

7. Розпакувати файли, імена яких складаються з трьох символів, за допомогою команди gzip.

8. Розпакувати файли, в іменах яких присутні прописні букви, за допомогою команди gzip, при цьому вивести список оброблюваних файлів.

9. Розпакувати файли, імена яких складаються з двох символів, за допомогою команди gunzip.

Лабораторна робота № 4

Пошук файлів

Мета роботи: набути практичних навичок в пошуку інформації з певними характеристиками.

Постановка задачі: виконати завдання, використовуючи відповідні команди.

Теоретичні відомості

Часто в процесі роботи виникає необхідність пошуку файлів з певними характеристиками, такими як, наприклад, права доступу, ім'я, розмір, тип і т.д. Команда `find` є універсальним інструментом пошуку: вона дозволяє шукати файли і каталоги, проглядаючи лише поточний каталог або всі каталоги в системі. Загальний формат команди такий:

```
find путнє_ім'я прапори [дії]
```

Тут `путнє_ім'я` – це каталог, з якого необхідно почати пошук.

Спочатку розглянемо основні прапори команди.

-name ім'я/шаблон	пошук файлів з вказаним ім'ям або іменами, що задовольняють шаблону;
-user користувач	пошук файлів, що належать вказаному користувачу;
-group група	пошук файлів, що належать вказаній групі користувачів;
-nouser	пошук файлів, що належать неіснуючому користувачу (запис для якого відсутній у файлі /etc/passwd);
-nogroup	пошук файлів, що належать неіснуючій групі користувачів (запис для якої відсутній у файлі /etc/groups);
-type тип	пошук файлів, що мають вказаний тип: f – звичайний файл; d – каталог; l – посилання; p – іменований канал; s – сокет; c – файл символьного пристрою; b – файл блокового пристрою;
-empty	пошук порожніх файлів;
-perm права	пошук файлів, що мають вказані права доступу;
-size N	пошук файлів, що мають розмір N одиниць: c – байт; k – кілобайт; b – блоків (ділянки по 512 байт, але їх величина може змінюватися залежно від ОС);
-depth	при пошуку файлів спочатку є видимим вміст поточного каталогу і лише потім перевіряється запис, відповідний самому каталогу;
-mindepth N	указує спускатися при пошуку файлів як мінімум на N рівнів нижче за вказаний каталог;
-maxdepth N	указує спускатися при пошуку файлів не нижче на N рівнів щодо вказаного каталогу;
-atime N	пошук файлів, звернення до яких було виконано N днів тому;

-amin N — пошук файлів, звернення до яких було виконано N хвилин тому;

-mtime N — пошук файлів, модифікованих або створених N днів тому;

-mmin N — пошук файлів, модифікованих або створених N хвилин тому;

-newer файл — пошук файлів, створених пізніше, ніж вказаний файл.

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

Нижче приведені дії команди find.

-print — виведення на стандартне виведення імен знайдених файлів (ця дія використовується за умовчанням);

-exec команда {} \; — виконання вказаної команди для всіх знайдених файлів. Зверніть увагу на наявність пропуску між символами «{}», що означає «підставити ім'я кожного знайденого файлу в якості аргумент команди» і «\;», що означає «кінець команди»;

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

Критерії пошуку можна об'єднувати, використовуючи логічні оператори. Нижче приведені оператори в порядку убування їх пріоритету.

Коротка форма запису оператора	Довга форма запису оператора	Опис
! критерій	-not	Критерій не повинен виконуватися
критерій1 -a критерій2	-and	Оператор І – повинні виконуватися обидва критерії одночасно (цей оператор використовується в команді find при об'єднанні критеріїв пошуку за умовчанням)
критерій1 -o критерій2	-or	Оператор АБО – повинен виконуватися або критерий1 або критерий2

В загальному випадку при використовуванні операторів критерії перевіряються в порядку їх пріоритету. Змінити порядок можна за допомогою дужок. Дужки потрібно виділяти за допомогою зворотного слеша.

Приклади:

```
find programs -type f -name '*.c'  знайти в каталозі programs загально-
-perm 777 -print                   доступні файли з суфіксом .c
 find /var/www/ -mtime +30 -name знайти в каталозі /var/www/ файли з
'*.php' -type f                    суфіксом .php, модифіковані більше 30
                                   днів тому
find . \( -name '~*' -or -name знайти в поточному каталозі файли з
'temp*' \) -type f > list.lst      іменами, що починаються з символу «~»
```

	або з «temp» і помістити список знайдених файлів у файл list.lst
`find . -user lion -type f –exec cp {} /home/ray/lion_files \;`	знайти в поточному каталозі файли, що належать користувачу lion, і скопіювати їх в каталог /home/ray/lion_files

Підстановка команд

Командна оболонка дозволяє результат виконання однієї команди підставити в якості аргумент в іншу команду. Для цього використовуються символи «`» (зворотні лапки), в які полягає команда, результат виконання якої повинен бути підставлений. Наприклад, нехай задана команда пошуку в поточному каталозі і розархівування файлів з розширеннями .gz, доступ до яких проводився більше двох місяців тому:

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

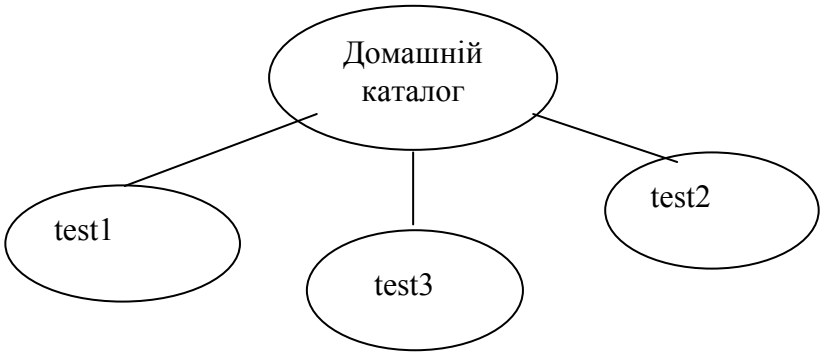
Її можна б було записати і так:

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

Контрольні питання:

1. Як в UNIX здійснюється пошук файлів, каталогів?
2. Чи можна об'єднувати критерії пошуку?
3. Як здійснюється підстановка команд в UNIX ?

Завдання до виконання




```
> f1.txt                    > list1
> file2.txt                   list2
> datafile                  > list3.txt
  database                  > file1.lst
  emp                       > f1
> x-file.lst                  f2
```

1. У домашньому каталозі створити каталоги test1, test2 і test3. В них створити файли (імена вказані на малюнку вище). В ті, які помічені знаком >, помістити довільну інформацію.
2. Знайти в каталозі /tmp і видати на екран назви всіх файлів, що не належать користувачу root.
3. Знайти в каталозі /etc і видати на екран назви всіх файлів, в іменах яких є цифри і які не закінчуються на букву, розміром більше 1 байта.

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

5. Проглянути докладну 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. Заархівувати за допомогою команди gzip всі файли каталогу test1, що містять в імені букву f.

8. Розархівувати за допомогою команди gunzip всі файли каталогу test1, що мають суфікс .gz.

9. Видалити всі файли нульового розміру з домашнього каталога.

10. Підрахувати, скільки файлів домашнього каталога мають ім'я, що починається з великої букви.

11. Помістити список каталогів з каталога /var/www у файл test2/catalogues.lst.

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

13. Знайти в каталозі /var/www/icons і видати на екран назви всіх файлів з розширеннями .gif і .png розміром більше 2 кбайт.

14. Скопіювати в каталог test3 всі файли каталогу/usr/bin з правами доступу rwxr-x-r-x, імена яких починаються з букви b і мають в назві букву o.

Лабораторна робота № 5

Використовування регулярних виразів

Мета роботи: вивчення синтаксису шаблонів, що використовуються в регулярних виразах.

Постановка задачі: виконати завдання, використовуючи відповідні команди.

Теоретичні відомості

Регулярні вирази — це система пошуку фрагментів в тексті, заснована на спеціальній системі запису зразків для пошуку. Зразок, що задає правило пошуку, також називають шаблоном або маскою.

Регулярні вирази використовуються для стислого опису деякої безлічі рядків за допомогою шаблонів, без необхідності переліку всіх елементів цієї множини. При складанні шаблонів використовується спеціальний синтаксис, що підтримує, звичайно, наступні операції:

Перелік: вертикальна межа розділяє допустимі варіанти. Наприклад, `one|two` відповідає `one` або `two`.

– Угрупування: круглі дужки використовуються для визначення області дії і пріоритету операторів. Наприклад, шаблони `abd|acd` і `a(b|c)d` описують одне і те ж поєднання символів: `abd` і `acd`.

– Квантифікація: квантифікатор після символу або групи визначає, скільки разів може зустрічатися попередній вираз:

- {m,n} – повторень може бути від m до n включно. Наприклад, `go\{1,3\}gle` відповідає `gogle`, `google` або `goooogle`.
- ? – означає повторення 0 або 1 разів, те ж саме, що і {0,1}. Наприклад, `colou?r` відповідає і `color`, і `colour`.
- * – означає повторення 0, 1 або будь-яке число раз (або {0}). Наприклад, `go*gle` відповідає `ggle`, `gogle`, `google` і ін.
- + – означає повторення хоча б 1 разів (або {1}). Наприклад, `go+gle` відповідає `gogle`, `google` і т.д. (але не `ggle`).

Конкретний синтаксис регулярних виразів залежить від реалізації. Ми розглядатиме синтаксис базових регулярних виразів Unix. В цьому синтаксисі більшість символів відповідає самі собі. Виключення з цього правила називаються метасимволами:

Метасимвол	Значення в регулярних виразах
.	Відповідає будь-якому одиничному символу.
[ ]	Відповідає будь-якому одиничному символу з числа символів, укладених в квадратні дужки. Символ «-» інтерпретується буквально лише в тому випадку, якщо він розташований безпосе-

	редньо після відкриваючої або перед закриваючою дужкою: [abc-] або [-abc]. Інакше, він позначає інтервал символів.
[^]	Відповідає одиничному символу з числа тих, яких немає в дужках. Наприклад, [^abc] відповідає будь-якому символу, крім «a», «b» або «c». [^0-9] відповідає будь-якому символу, крім цифр.
^	На початку регулярного виразу, відповідає початку рядка тексту.
$	В кінці регулярного виразу, відповідає кінцю рядка тексту.
*	Зірочка після виразу, відповідного одиничному символу, відповідає нулю або більш копій цього виразу. Наприклад «[xyz]*» відповідає порожньому рядку, «x», «y», «zx», «zyx», і т.д.
?	Знак питання після виразу, відповідного одиничному символу, відповідає нулю або одній копії цього виразу. Наприклад, «19?0» відповідає «10» або «190».
\	Відміняє спеціальне значення наступного за ним метасимвола. Наприклад, щоб представити символ «крапка», треба написати \.
\ {x,y\}	Відповідає останньому блоку, що зустрічається не менше x і не більш y разів. Наприклад, «a\{1,3\}» відповідає «a», «aa» або «aaa».
\ {x,\}	Відповідає останньому блоку, що зустрічається не менше x раз. Наприклад, «a\{3,\}» відповідає «aaa», «aaaa», «aaaaa» і т.д.
\ {,y\}	Відповідає останньому блоку, що зустрічається не більш y разів. Наприклад, «a\{,4\}» відповідає порожньому рядку, «a», «aa»,»aaa» і «aaaa».
\ {x\}	Відповідає останньому блоку, що зустрічається точно x раз. Наприклад, «a\{6\}» відповідає «aaaaaa».

При складанні регулярних виразів слід також враховувати їх дві основні риси — вони є т.з. «ледачими» і «жадібними». Перше означає, що в рядку, де є декілька збігів з шаблоном, шаблон співпаде з першим з них. «Жадібність» регулярних виразів полягає в тому, що, при використовуванні квантифікаторів «*» і «+», шаблон співпадатиме з максимально довгим з можливих варіантів.

Команда grep

Команда `grep` є інструментальним засобом для виконання пошуку з використанням регулярних виразів. Вона виконує пошук в текстових файлах або в стандартному вхідному потоці виразів, відповідних шаблону, з подальшим відображенням результату на екрані.

```
grep [прапори] регулярний вираз [файл/список файлів]
```

Якщо імена файлів не задані, то текст береться із стандартного вхідного потоку.

Рядок, який заданий в якості регулярного виразу і складається з декількох слів, необхідно брати в подвійні лапки. Інакше перше слово рядка буде сприйнято як зразок пошуку, а решта слів вважатиметься іменами файлів. Якщо зразок пошуку складається із значення змінної (наприклад $PATH), то також рекомендується узяти її в подвійні лапки. Це необхідне тому, що, перш ніж передати параметри команді `grep`, shell виконує підстановку значень змінної, яке може містити пропуски.

Якщо до складу регулярного виразу входять метасимволи, зразок пошуку необхідно брати в одинарні лапки.

Основні прапори команди grep:

-c --count	замість звичного виведення для кожного вхідного файлу друкує кількість рядків, що співпали (якщо також вказаний прапор -v, друкує кількість рядків, що не співпали);
-e	використовується, якщо необхідно задати більше одного шаблона для порівняння;
-h --no-filename	не виводить ім'я файлу перед кожним знайденим рядком;
-f, file --file=file	одержує з файлу шаблони для пошуку поодинці в рядку;
-i--ignore-case	ігнорує чутливість до регістра в шаблоні і у вхідних файлах;
-l --files-with-matches	виводить лише імена файлів, рядки яких містять зразок пошуку;
-L --files-without-matches	виводить лише імена файлів, рядки яких не містять зразок пошуку (перегляд завершується на першому знайденому збігу);
-n --line-number	перед кожним знайденим рядком виводить її номер в переділах її вхідного файлу;
-s --no-messages	забороняє видачу повідомлень про помилки з приводу неіснуючих або нечитаних файлів;
-v --invert-match	вибирає рядки, які не співпадають з шаблоном.

Приклади:

`grep –v 'Hello.*' file.txt`	вивести на екран рядка файлу file.txt, що не містять рядків, співпадаючих з шаблоном 'Hello.*';
`grep 'help me' ./* | wc –w`	знайти у файлах поточного каталогу рядки, в яких присутнє словосполучення «help me» і вивести на екран кількість слів в цих рядках;
`ls /bin | grep '2$'`	вивести на екран файли з каталогу /bin, імена яких починаються на «b»;
`cat ./*|grep 's\{2\}'`	вивести на екран рядки файлів поточного каталога, які містять подвоєння букви «s» (аналогічно grep ?s\ \}? *);
`mv `grep –L –e 'Hello world' -e 'one of us' dir1/*` dir2`	перемістити в каталог dir2 файли каталогу dir1, які не містять словосполучення «Hello world» і «one of us».

Контрольні питання:

1. Що таке регулярні вирази і для чого вони використовуються?
2. Який формат команди grep?
3. Які операції підтримуються при складанні шаблонів?

Завдання до виконання

1. У домашньому каталозі створити каталоги cat1и cat2. В каталозі cat1 створити файли з іменами file1, file2, file3, relatives, friends, others, keywords, birthdays, list. В file1 помістити довідку по команді grep. В file2 помістити довідку по команді find. В file3 помістити довідку по команді gzip. У файли relatives, friends і others помістити імена і номери телефонів відповідно ваших родичів, друзів і знайомих. Номери телефонів повинні бути записані у форматі `xxx-xxx-xx-xx` (як номери мобільного або міжміського зв'язку), так і у форматі `xxx-xx-xx` (як міські номери) і перераховані через кому. Наприклад:

Ivanov Ivan: 097-234-56-78, 726-15-89.

У файл birthdays помістити імена і дати народження родичів, друзів і знайомих (кожний 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. Одержати список файлів каталогу cat1, в яких присутнє слово «name».
3. Підрахувати кількість рядків у файлах каталогу cat1, в яких присутнє словосполучення «to be».
4. Видати на екран всі рядки файлів каталога /etc, що містять входження слів «terminal» і «keyboard».
5. За допомогою команди grep одержати список файлів каталогу /bin, імена яких починаються з «ch».
6. За допомогою команди grep одержати список файлів каталогу /etc, імена яких починаються з букв від «a» до «m» і які мають в назві букводрукувальне «at». Помістити цей список у файл list.
7. Видати на екран (з їх номерами у файлі) всі рядки файлів каталогу cat1, в яких є слова, що складаються лише з великих букв (двох і більш).
8. Видати на екран всі рядки файлів каталогу cat1, що містять, чотири цифри підряд.
9. Видати на екран імена людей, перерахованих у файлі birthdays, які народилися під знаком Овна (21 березня – 20 квітня).
10. Підрахувати, скільки людей, перерахованих у файлах relatives, friends і others, мають більше одного номера телефону.
11. Заархівувати за допомогою команди gzip файли каталогу cat1, що містять номери телефонів у форматі 09X-XXX-XX-XX і 7XX-XX-XX.
12. Скопіювати в каталог cat2 файли каталогу cat1, в яких присутні слова і словосполучення, визначені у файлі keywords.

Лабораторна робота № 6

Управління завданнями

Мета роботи: вивчити поняття процесу і основні команди для управління процесами.

Постановка задачі: виконати завдання, використовуючи відповідні команди.

Теоретичні відомості

UNIX відстежує роботу процесів, призначаючи кожному з них свій ідентифікатор процесу (Process ID – PID). Ідентифікаційні номери привласнюються процесам по порядку у міру їх створення. Коли номери закінчуються, ядро ОС скидає лічильник в одиницю і знову починає привласнювати їх по порядку, пропускаючи ті ідентифікатори, які ще зайняті.

У ОС UNIX є окремі системні виклики для створення (породження) процесу і для запуску нової програми. Початковий процес називається батьківським, а породжений – дочірнім. Крім власного PID кожний процес має ідентифікатор батьківського процесу (Parent Process ID – PPID).

У ОС UNIX користувач може управляти ходом виконання команди (процесу) і має нагоду:

- запускати команди (процеси) у фоновому режимі, одночасно виконуючи інші команди;
- переривати команди (процеси);
- відновлювати роботу команди (процесу) з тієї крапки, де вони були перервані.

Примітка. Команди, що поступають від користувачів, називають завданнями, щоб відрізнити їх від системних процесів.

Команда jobs

Вона завжди викликається без аргументів і показує завдання, запущені з поточного екземпляра shell. На початку кожного рядка виведення цієї команди указується порядковий номер завдання у вигляді числа в квадратних дужках. Після номера указується стан процесу: stopped (зупинений), running (виконується) або suspended (припинений). В кінці рядка указується команда, яка виконується даним процесом.

Переведення процесу у фоновий режим

Якщо процес створюється шляхом запуску програми з командного рядка, то зазвичай він запускається «на передньому плані». Це значить, що процес «прив'язується» до терміналу, з якого він запущений, сприймаючи введення з цього терміналу і здійснюючи на нього виведення. Можна запустити процес у фоновому режимі, коли він не пов'язаний з терміналом. Для запуску процесу у фоновому режимі в кінці командного рядка додають символ &. В цьому випадку управління без очікування завершення повертається батьківському процесу.

У оболонці bash є дві вбудовані команди, які служать для переведення процесів на передній план або повернення їх у фоновий режим. В якості аргументу обом цим командам передаються номери тих завдань, які присутні у виведенні команди jobs. Якщо аргументи відсутні, то мається на увазі завдання, помічене +. Команда fg перекладає вказаний в аргументі процес на передній план, а команда bg — переводить процес у фоновий режим. Однією командою bg можна перевести у фоновий режим відразу декілька процесів, а ось повертати їх на передній план необхідно поодинці.

`fg %n, де n – порядковий номер завдання (який виводиться командою jobs), що виконується, – переводить завдання з фонового в пріоритетний режим.`

`bg %n, де n – порядковий номер завдання, що виконується, – переводить задачу у фоновому режимі зі стану Stopped в стан Running.`

Натиснення клавіш <Ctrl> +<z> – перериває задачу і переводить її у фоновий режим в стан Stopped.

А зараз навчимося визначати, які процеси в системі запущені. Для цього служить наступна команда.

Команда ps

`ps [опції]`

Команда `ps` видає для кожного процесу окремий рядок, але вміст цього рядка може бути різним. Залежно від заданих опцій можуть бути присутні наступні поля:

- USER — ім'я власника процесу;
- PID — ідентифікатор процесу в системі;
- PPID — ідентифікатор батьківського процесу;
- %CPU — частка часу центрального процесора (у відсотках), виділеного даному процесу;
- %MEM — частка реальної пам'яті (у відсотках), що використовується даним процесом;
- VSZ — віртуальний розмір процесу (в кілобайтах);
- RSS — розмір резидентного набору (кількість 1K-страниц в пам'яті);
- STIME — час старту процесу;
- TTY — вказівка на термінал, з якого запущений процес;
- S або STAT — статус процесу;
- PRI — пріоритет планування;
- NI — відносний пріоритет – значення nice (враховується при створенні черги на виконання планувальником);
- TIME — скільки часу центрального процесора зайняв даний процес;
- CMD або COMMAND — командний рядок запуску програми, виконуваної даним процесом;

а також і інші поля.

У полі «Статус процесу», як вже мовилося вище, можуть стояти наступні значення:

- R — здійснимий процес, що чекає лише моменту, коли планувальник задач виділить йому черговий квант часу;
- S — процес «спить»;
- D — процес знаходиться в стані підкачки на диску;
- T — зупинений процес;
- Z — процес-зомбі.

Поряд з покажчиком статусу можуть стояти додаткові символи з наступного набору:

- W — процес не має резидентних сторінок;
- < — високо-пріоритетний процес;
- N — низько-пріоритетний процес;
- L — процес має сторінки, заблоковані в пам'яті.

Щоб одержати список всіх процесів, треба використовувати команду `ps` з опціями `ax` або `-A`. Вивід в цих двох випадках відрізняється лише в полі CMD: в першому випадку видається повний командний рядок запуску програми, а в другому — лише ім'я запущеної програми.

Деякі опції команди ps:

- A Вибирає всі процеси
- a Вибирає всі процеси, пов'язані з конкретним терміналом, крім головних системних процесів сеансу
- d Вибирає всі процеси, крім головних системних процесів сеансу
- e Вибирає всі процеси

a Вибирає всі процеси, які пов'язані з даним терміналом, у тому числі і інших користувачів

r Виводить інформацію лише про працюючі процеси

x Вибирає всі процеси, які від’єднанні від терміналу

-o Вивід у форматі, заданому користувачем

-f Для виведення використовує графічне представлення дерева процесів

-u Вибирає процеси по діючому ідентифікатору користувача

-C Вибирає процеси за ім'ям команди

-G Вибирає процеси по GID

-U Вибирає процеси по UID

-p Вибирає процеси по ідентифікаторах процесів

Приведемо декілька прикладів вживання команди ps, які покажуть, як користуватися цією командою в типових ситуаціях.

Для того, щоб побачити всі процеси в системі, використовуючи стандартну форму виведення (PID, TTY, TIME, CMD):

```
ps -e
```

Можна до тієї ж команди додати опцію -o (вибір полів для виведення), після якої вказати через кому, які саме поля ви хочете бачити у виведенні команди (поля указуються рядковими буквами!): `ps -eo pid,user,cmd`

Команда top

Команда `ps` дозволяє зробити як би «моментальний знімок» процесів, запущених в системі. На відміну від `ps` команда `top` відображує полягання процесів і їх активність «в реальному режимі часу».

Вміст вікна оновлюється кожні 5 секунд. Список процесів може бути відсортований по часу ЦПУ (за умовчанням), що використовується, по використовуванню пам'яті, по PID, за часом виконання. Перемикати режими відображення можна за допомогою команд, які сприймає програма `top`. Це наступні команди (просто натискуйте відповідні клавіші з урахуванням регістру):

- <Shift>+<N> — сортування по PID;
- <Shift>+<A> — сортувати процеси по віку;
- <Shift>+<P> — сортувати процеси по використовуванню ЦПУ;
- <Shift>+<M> — сортувати процеси по використовуванню пам'яті;
- <Shift>+<T> — сортування за часом виконання.

Крім команд, що визначають режим сортування, команда `top` сприймає ще ряд команд, які дозволяють управляти процесами в інтерактивному режимі. За допомогою команди <K> можна завершити деякий процес (його PID буде запитаний), а за допомогою команди <R> можна перевизначити значення nice для деякого процесу.

Сигнали і команда kill

Сигнали — це засіб, за допомогою якого процесам можна передати повідомлення про деякі події в системі. Самі процеси теж можуть генерувати сигнали, за допомогою яких вони передають повідомлення ядру і іншим процесам. За допомогою сигналів можна здійснювати управління процесами: припинення процесу, запуск припиненого процесу, завершення роботи процесу.

Сигнали прийнято позначати номерами або символічними іменами. Всі імена починаються на SIG, але цю приставку іноді опускають: наприклад, сигнал з номером 1 позначають або як SIGHUP, або просто як HUP.

Коли процес одержує сигнал, то можливий один з двох варіантів розвитку подій. Якщо для даного сигналу визначена підпрограма обробки, то викликається ця підпрограма. Інакше ядро виконує від імені процесу дію, визначену за умовчанням для даного сигналу. Виклик підпрограми обробки називається перехопленням сигналу. Коли завершується виконання підпрограми обробки, процес поновлюється з тієї крапки, де був одержаний сигнал.

Для посилки сигналу процесу (або групі процесів) можна скористатися командою `kill` в наступному форматі:

```
kill [-сигн] PID [PID..]
```

де `сигн` - це номер сигналу, причому якщо вказівка сигналу опущена, то посилається сигнал 15 (TERM - програмне завершення процесу). Частіше за все використовується сигнал 9 (KILL), за допомогою якого суперкористувач може завершити будь-який процес. Але сигнал цей дуже «грубий», якщо можна так виразитися, тому його використовування може привести до порушення

порядку в системі. Тому в більшості випадків рекомендується використовувати сигнали TERM або QUIT, які завершують процес більш «м'яко».

Замість `PID` процесу може бути використаний порядковий номер завдання, що виконується, який виводиться командою `jobs`, тобто команда `kill %n`, де `n` – порядковий номер завдання, що виконується, – відміняє завдання, виконуване у фоновому режимі.

Віртуальна файлова система /proc

Файлова система `/proc` є механізмом для ядра ОС і його модулів, що дозволяє посилати інформацію процесам. За допомогою цієї віртуальної файлової системи Ви можете працювати з внутрішніми структурами ядра, одержувати корисну інформацію про процеси і змінювати установки на льоту. Файлова система `/proc` є віртуальною – ядро створює її в пам'яті комп'ютера, на відміну від інших файлових систем, які розташовуються на диску.

Деякі файли і каталоги файлової системи `/proc` розглянуті нижче.

`/proc/cpuinfo` – Інформація про процесор, така як тип процесора, його модель, продуктивність, розмір кеша і др.

`/proc/meminfo` – Інформація про використовування пам'яті, як фізичної так і swap-області (області підкачки).

`/proc/n` – Каталог, що містить інформацію про процес з номером n. Для кожного процесу існує окремий каталог в /proc, ім'ям якого є його числовий ідентифікатор. Всередині цих каталогів знаходяться файли, що містять важливу інформацію про відповідні процеси. Наприклад, у файлі `cmdline` міститься інформація про те, як був запущений процес, а у файлі `status` знаходяться змінні оточення цього процесу, ідентифікатори групи і користувача, що запустив процес, ідентифікатор батьківського процесу і поточний стан процесу. У файлі `/proc/kcore` відображається фізична пам'ять системи в даний момент. Розмір цього файлу точно такий же, як і у пам'яті комп'ютера, лише він не займає місця в самій пам'яті, а генерується на льоту при доступі до нього програм.

Запуск команд в певний час

Замість того щоб переводити команду у фоновий режим, можна вказати час, коли її потрібно виконати. Для цього служить команда `at`.

```
at  час [дата] [+затримка]
at  -r ідентифікатор_завдання ...
at  -l [ідентифікатор_завдання ...]
```

Команда `at` в першому з приведених варіантів читає із стандартного введення завдання, виконання якого планується на вказаний час. Значення опцій двох інших варіантів команди `at` таке:

r Видалити завдання, заплановані раніше за допомогою at або batch, по ідентифікаторах_завдань. Ідентифікатори повідомляються командами at і batch. Їх можна взнати також по команді at -l. Лише суперкористувач може видаляти чужі завдання.

Вивести інформацію про заплановані завдання по ідентифікаторах_завдань. Якщо ідентифікатори не вказані, видається список всіх завдань, запланованих користувачем і ще не виконаних.

Користувачу дозволяється виконувати команду at лише за умови, що його ім'я зустрічається у файлі /usr/lib/ cron/at.allow. Якщо цього файлу не існує, то перевіряється файл /usr/lib/cron/at.deny, для того, щоб взнати, чи не заборонений користувачу доступ до at. Якщо обидва файли відсутні, то лише суперкористувачу дозволено планувати виконання завдання.

Час може бути вказаний 1, 2 або 4 цифрами. Якщо час складається з однієї або двох цифр, то він позначає годинну; чотиризначне число позначає годинну і хвилини. Час також може бути заданий як два числа, які розділені двокрапкою. Можуть бути додані суфікси am або pm, інакше години указуються від 0 до 23.

Дата може бути вказана двома способами: по-перше, у вигляді назви місяця, за яким слідує число [і, можливо, рік (через кому)], а по-друге, як день тижня (повністю або скорочений до 3 букв).

Додаткова затримка є просто числом, за яким слідує одне з наступних слів: minutes (хвилини), hours (годинник), days (дні), weeks (тижні), months (місяці), або years (роки). Можна вказувати одиницю вимірювання і без числа, наприклад at now +minutes.

Далі приведені приклади коректних завдань дати/часу в команді at:

```
0815am Jan 16        8:15am Jan 16      now +1 day
5 pm Friday          next Saturday
```

Приклад планування завдання:

```
at  8:15am Jan 16
```

Далі натискуємо <Enter> і пишемо в наступному рядку

```
echo Hello world
```

А зараз натискуємо <Ctrl>+<D> для завершення введення команди.

Інший варіант – коли список команд, які необхідно запланувати для виконання, знаходиться у файлі (наприклад, з ім'ям commands). Тоді в команді at потрібно використовувати опцію -f для вказівки імені файлу:

```
at 15:15 Friday –f commands
```

або так:

```
at 15:15 Monday < commands
```

Контрольні питання:

1. Що таке процес?
2. Як запустити процес у фоновому режимі?
3. Як перевести процес з фонового режиму в пріоритетний?
4. Чим відрізняються фоновий і пріоритетний режими?
5. Для чого використовуються команди ps і top?
6. Що відбувається при виконанні команди kill?
7. Як запустити команду в певний час?

Завдання до виконання

1. За допомогою команд fg, bg, kill провести наступні дії:

- запустити команду mc у фоновому режимі;
- перевести команду з фонового режиму в пріоритетний;
- перевести команду з пріоритетного режиму у фоновий з припиненням;
- відновити роботу команди у фоновому режимі;
- завершити роботу команди.

Після кожної виконаної дії перевірити стан команди, занести результати в протокол лабораторної роботи.

2. Запустити команду editor у фоновому режимі. Одержати наступні відомості про неї:

- ідентифікатор процесу;
- ідентифікатор батьківського процесу;
- пріоритет планування;
- ім'я власника процесу;
- частку часу центрального процесора;
- частку реальної пам'яті;
- час запуску процесу;
- статус процесу;
- термінал, з якого був запущений процес.

3. Запустити команду top. По її виведенню знайти по три процеси, які більше всіх використовують процесор і займають пам'ять. За допомогою команди kill припинити, а потім завершити виконання команди editor, що виконується у фоновому режимі. Після посилки кожного відповідного сигналу за допомогою команд jobs і ps проконтролювати стан процесу і занести результати в протокол лабораторної роботи.

4. За допомогою файлової системи /proc з'ясувати розмір оперативної пам'яті та марку і частоту процесора. Запустити програму mc, визначити її PID і за допомогою файлової системи /proc узнати, як був запущений процес, його PPID, UID і GID, поточний статус процесу. Занести результати в протокол лабораторної роботи.

5. За допомогою команди at запустити команду mc в 18:00 поточного дня.

- Створити файл з ім'ям cmd_file, в якому задати список команд для запуску в 8:20 ранку 15 грудня:
- створення архівного файлу, що складається зі всіх непорожніх файлів домашнього каталогу;
- запис у файл processes.txt лише імен команд запущених процесів і імен власників процесів;
- запис у файл processes.txt кількості запущених процесів.

Лабораторна робота №7

Сценарії оболонки

Мета роботи: вивчити поняття сценарію оболонки, область видимості змінних, основних керівників конструкції.

Постановка задачі: виконати завдання, використовуючи відповідні команди.

Теоретичні відомості

Взаємодія користувача з операційною системою здійснюється через оболонку. Оболонка (shell) – це програма, що оброблює команди, що вводяться користувачем або що містяться у файлі (при цьому файл називається сценарієм і є програмою, що інтерпретується). Аналогами оболонки в середовищі Windows є командний процесор command.com або cmd.exe, а аналогами сценаріїв – командні файли (bat-файли). В Linux існує ряд різних оболонок, найпопулярнішої з яких є bash.

Відразу після запуску оболонки проводиться її ініціалізація для установки ряду параметрів. При цьому оболонкою проводиться читання двох файлів: /etc/profile і ~/.profile. В першому з них містяться настройки параметрів, загальні для всіх користувачів. В другому файлі кожний користувач може розмістити свої власні настройки для роботи з оболонкою.

Призначена для користувача оболонка може бути запущена на виконання в двох режимах – інтерактивному і не інтерактивному. Коли оболонка видає користувачу запрошення, вона працює в інтерактивному режимі. У неінтерактивному режимі оболонка не взаємодіє з користувачем. Натомість вона читає команди з деякого файлу і виконує їх. Коли буде досягнутий кінець файлу, оболонка завершить своє виконання. Запуск оболонки в неінтерактивному режимі можна здійснити наступним способом:

```
ім'я_оболонки ім'я_файлу
```

Тут `ім'я_файлу` – ім'я файлу, що містить команди для виконання. Такий файл називається сценарієм оболонки. Він є текстовим файлом і може бути створений будь-яким доступним текстовим редактором.

В якості `імені_оболонки` може виступати, наприклад, sh, bash або ім'я будь-якої іншої оболонки.

Запуск сценарію

Користувачу може представляти незручність кожного разу вказувати ім'я програми-оболонки для виконання сценарію. Для того, щоб мати нагоду виконувати сценарій, набираючи лише його ім'я, перш за все необхідно зробити його виконуваним. Для цього необхідно встановити відповідні права доступу до файлу за допомогою команди:

```
chmod +x ім'я_файлу
```

Крім цього, необхідно явно вказати, яка оболонка повинна використовуватися для виконання даного сценарію. Це роблять, розмістивши в

його першому рядку послідовність символів `#!шлях_до_програми`. Наприклад, якщо необхідно вказати, що для виконання сценарію слід використовувати оболонку bash, першим рядком сценарію повинен бути рядок

```
#!/bin/bash
```

Сценарій може містити коментарі. Коментар – це оператор, який може розміщуватися в сценарії оболонки, але оболонкою не виконується. Коментар починається з символу # і продовжується до кінця рядка.

Нижче приведений приклад короткого сценарію:

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

Примітка: декілька команд в сценарії можуть записуватися в один рядок. В якості роздільника між ними необхідно вказувати крапку з комою (;). Наприклад:

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

Змінні

Змінна – це програмний об'єкт, здатний приймати значення. Оболонка дозволяє створювати і видаляти змінні та привласнювати їм значення. Імена змінних визначаються за тими ж правилами, що і в мові програмування С.

Визначаються змінні таким чином (відсутність пропусків до і після символу = істотно):

```
ім'я_змінної=значення
```

Всі змінні за умовчанням мають рядковий тип. В змінній можна зберігати будь-яке потрібне значення. Особливий випадок – коли це значення містить пропуски. Для правильного виконання такої дії вказане значення досить укласти в лапки.

Якщо змінній привласнена послідовність чисел, то значення цієї змінної трактуватиметься як число. Наприклад, `a=12345`. Тепер змінну а можна використовувати в арифметичних виразах.

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

Наприклад, внаслідок виконання команди

```
echo a
```

на термінал виведеться буква а, а внаслідок виконання команди

```
echo ${a}
```

буде надруковано число `12345`.

Примітка: при підстановці значень змінних фігурні дужки в більшості випадків можна опускати. Але зверніть увагу, внаслідок виконання команди

```
echo ${a}1
```

до значення змінної а буде дописана 1, тобто набудемо значення 123451, а внаслідок виконання команди

```
echo $a1
```

на термінал буде виведено 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 одержати кількість символів в певній змінній (довжину змінної).

За допомогою конструкції `${ім'я_змінної:m:n}` можна виділити підрядок з певної змінної (тут де m – номер позиції, починаючи з якої проводитиметься виділення підрядка, а n – кількість символів, що виділяються).

Примітка: значення n можна опускати. Тоді проводитиметься виділення символів до кінця рядка.

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

Нижче приведений приклад, що ілюструє роботу зі змінними в сценаріях.

```
#!/bin/bash
MY_NAME=Sergey
MY_FULL_NAME=«Sergey B. Sidorov»
echo name = $MY_NAME and full name = $MY_FULL_NAME
echo Length my full name is ${#MY_FULL_NAME} symbols
echo My surname is ${MY_FULL_NAME:10}
m=0
n=6
echo My name is ${MY_FULL_NAME:m:n}
echo ${MY_FULL_NAME:10:5}
unset MY_NAME
```

У результаті виконання сценарію на термінал буде видано повідомлення:

```
name = Sergey and full name = Sergey B. Sidorov
Length my full name is 17 symbols
My surname is Sidorov
My surname is Sergey
Sidor
```

Всі розглянуті вище приклади змінних – це приклади скалярних змінних. Разом з ними можна використовувати змінні-масиви. Доступ до елементів масиву здійснюється операцією індексації []. Мова програмування оболонки не вимагає попереднього оголошення змінної масивом з вказівкою його розмірності. При цьому окремі елементи масиву створюються у міру доступу до них. Нижче приведений приклад роботи з масивом NAME.

```
#!/bin/sh
NAME[0]=Сергій
NAME[10]=Катя
NAME[2]=Ліза
echo всі імена: ${NAME[*]}
echo ${NAME[10]} і ${NAME[2]} сестри
```

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

Область видимості змінних

Кожна змінна має свою область видимості. У мові оболонки всі змінні діляться на три категорії: локальні змінні, змінні оточення і змінні оболонки.

Локальна змінна – це така змінна, яка існує лише усередині конкретного екземпляра оболонки. Вона не доступна програмам, що запускаються на виконання з цієї оболонки. Всі змінні, що розглядалися раніше, були локальними.

Змінна оточення – це змінна, яка доступна будь-якій програмі, запущеній з даної оболонки. Деякі програми потребують певних змінних оточення. У такому разі в сценарії їх можна визначити.

Змінна оболонки – це спеціальна змінна, яка встановлюється оболонкою і необхідна їй для коректної роботи. Деякі з них є змінними оточення, а деякі – локальними. В змінних з іменами 1, 2, 3. зберігаються параметри командного рядка. Тобто якщо якийсь сценарій `script` запустити у вигляді `script something`, то в його оболонці `$1=something`.

Змінну можна розмістити в оточенні, виконавши команду експорту:

```
export ім'я_змінної
```

У результаті виконання наступного сценарію на термінал будуть видані значення всіх змінних оточення оболонки і, у тому числі, змінною MY_NAME.

```
MY_NAME=Sergey ; export MY_NAME;
set  #це команда для виведення всіх змінних оточення і їх значень
```

Нижче приводиться список деяких стандартних змінних оточення і змінних оболонки з вказівкою їх призначення і і прикладу їх ініціалізації.

HOME містить шлях до домашнього каталога користувача

PATH визначає список каталогів (розділених двокрапкою), в яких оболонка шукає ті, програми, що запускаються на виконання

HOSTNAME ім'я комп'ютера, на якому виконується оболонка

USER містить ім'я користувача, що працює з оболонкою в даний момент

PS1 основний рядок запрошення (за умовчанням $)

PS2 додатковий рядок запрошення (за умовчанням >).

Команда echo ${PATH} виведе на екран значення змінної оточення PATH, наприклад `/bin:/usr/bin`. А щоб додати ще один каталог для пошуку програм, що запускаються на виконання необхідно записати команду `PATH=${PATH}:/home/user`.

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

Наприклад, програма prog служить для виведення максимального серед трьох введених чисел і викликається так:

```
prog –d 23 45 38
```

Тоді позиційна змінна $0 містить значення prog, змінна $1 – значення -d, змінна $2 – значення 23, змінна $3 – значення 45, а змінна $4 – значення 38. Значення решти позиційних змінних будуть порожніми рядками.

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 вказане shift 3, то $4 стає $1.

Наступні змінні автоматично встановлюються оболонкою:

$# - кількість позиційних параметрів
$- - прапори, вказані при запуску оболонки або командою set
$? - десяткове значення, повернене попередньою 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 можуть бути вирішені за допомогою використовування спеціальних команд echo і read.

Команда echo видає на стандартний пристрій виведення значення всіх своїх параметрів.

Команда read ім'я1 ім'я2 ... ім'яN прочитує із стандартного пристрою введення рядок, виділяє з неї окремі слова (групи символів, відокремлювані пропусками) і кожне слово заносить у вказані в якості параметри відповідні змінні. При цьому якщо змінних менше ніж виділених слів, то в останню з них буде занесена частина рядка, що залишилася.

Обчислення арифметичних виразів

Оператори для обчислення арифметичних виразів представлені в таблиці в порядку пониження пріоритету. Для зміни порядку обчислення використовуються дужки.

Оператор	Значення
-	унарний мінус
!-	логічне заперечення, двійкова інверсія
*, /, %	множення, розподіл, взяття залишку від розподілу
+, -	складання, віднімання
<< >>	порозрядний зсув вліво, порозрядний зсув вправо
==, !=	перевірка на рівність, перевірка на нерівність
&	порозрядне І
^	що порозрядне ВИКЛЮЧАЄ АБО
|	порозрядне АБО
&&	логічне І
||	логічне АБО
=	привласнення значення
+=, -+	привласнення після складання, привласнення після віднімання
*=, %=	привласнення після множення, привласнення після взяття залишку від ділення

<<= >>=	привласнення після порозрядного зсуву вліво, привласнення після порозрядного зсуву вправо

Обчислення беруться у подвійні круглі дужки. Всередині подвійних круглих дужок підстановка змінних виконується автоматично.

Управляючі конструкції

Проста команда – це послідовність слів, розділена пропусками. Перше слово є ім'ям команди, яка виконуватиметься, а інші будуть передані їй як аргументи. Ім'я команди передається їй як аргумент номер 0 (тобто ім'я команди є значенням $0).

Список – це послідовність однієї або декількох команд, розділених символами ; & && або || і що мабуть закінчується символом ; або &. З чотирьох вказаних операцій ; і & мають рівні пріоритети, менші, ніж у && і ||. Пріоритети останніх також рівні між собою. Символ ; означає, що команди виконуватимуться послідовно, а & - паралельно.

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 його код завершення 0, то виконується <список2>. Якщо ж це не так, то виконується <список3>. Частини elif і else можуть бути відсутні.

В якості умови можуть використовуватися звичайний команди, проте найбільш часто – це виклик однієї або декількох команд test у вигляді

```
test вираз
```

або просто [вираз]. Зверніть увагу: пропуски, що відділяють вираз від квадратних дужок, обов'язкові.

Приклад. Програма, яка 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 введення параметрів
     read a b   #читаємо параметри в змінні a і b
elif [ n -eq 1 ]     #інакше, якщо переданий один параметр
then
     a=$1       #змінною a надаємо його значення
```

```
     echo Input b #запрошення до введення другого параметра
     read b       #читаємо параметр в змінну b
elif [ n –eq 2 ]  #інакше, якщо передано обидва параметри
then
     a=$1 #змінній a надаємо значення першого параметра
     b=$2 #змінній b надаємо значення першого параметра
else      #інакше (якщо не передано жодного параметра)
echo Usage: $0 [a [b]]#виводимо довідку по роботі програми
fi                    # кінець умовного оператора
((s=a+b))             # помістивши її в змінну s
echo $s               # виводимо результат
```

Команда test

Команда `test` застосовується для перевірки умови. Формат виклику:

```
test <вираз>
```

або просто

[<вираз>]

Команда test обчислює <вираз> і, якщо його значення – істина, повертає код завершення 0 (true); інакше – ненульове значення (false).

У сценаріях оболонки використовуються умови різних типів.

1. Умови перевірки файлів:
   - -f file - файл file є звичним файлом;
   - -d file - файл file - каталог;
   - -c file - файл file - спеціальний файл;
   - -r file - є дозвіл на читання файлу file;
   - -w file - є дозвіл на запис у файл file;
   - -s file - файл file не пустий.

2. Умови перевірки рядків:
   - str1 = str2 - рядки str1 і str2 збігається;
   - str1 != str2 - рядки str1 і str2 не збігається;
   - -n str1 - рядок str1 існує (непорожній);
   - -z str1 – рядок str1 не існує (порожній).

3. Умови порівняння цілих чисел:
   - x -eq y - x рівно y;
   - x -ne y - x не рівно y;
   - x -gt y - x більше y;
   - x -ge y - x більше або рівно y;
   - x -lt y - x менше y;
   - x -le y - x менше або рівно y.

Тобто в даному випадку команда `test` сприймає рядки символів як цілі (!) числа. Тому у всій решті випадків нульовому значенню відповідає пустий рядок. В даному ж випадку, якщо треба обнулити змінну, скажімо, x, то це досягається привласненням 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:

! - (not) інвертує значення коду завершення;

-o - (or) логічне АБО;

-a - (and) логічне І.

Приклади:

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

Оператор вибору

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

Оператор вибору виконує <список>, відповідний першому <шаблону>, якому задовольняє <змінна>. Форма шаблона та ж, що і що використовується в іменах файлів. Частина | <шаблон>... може бути відсутня.

Приклад. Вводимо назву фрукта і залежно від введеного значення виводимо його опис, а якщо не введено нічого, то виводимо відповідне повідомлення.

```
#!/bin/bash
echo Input FRUIT    # виводимо запрошення до введення
read FRUIT          # читаємо 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 оболонки дозволяє організовувати циклічне виконання команд. Пропонується 3 варіанти циклів: `while, until, for`.

Цикл while

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, інакше виконується тіло циклу і відбувається перехід на наступну ітерацію.

Нижче приведений приклад сценарію, в якому проводиться видача на термінал десяти послідовних чисел від 0 до 9.

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

Цикл until

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, що повертається при виконанні списку команд <умова>, береться із запереченням: <список> виконується в тому випадку, якщо остання команда в списку <умова> повертає ненульовий статус виходу.

Цикл for

Цикл for виконує команди із <списку> для кожного <елементу> і має наступний синтаксис:

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

Оператор for працює трохи не так, як в звичайних мовах програмування. Він при кожному проході циклу привласнює змінній чергове значення із заданого списку елементів. В якості елементів можна використовувати різні шаблони.

Якщо конструкція `[in <елемент1> <елемент2> . <елементN>]` опущена, то список команд <список> виконується один раз для кожного позиційного параметра, який заданий.

Розглянемо приклад використовування циклу for:

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

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

Мова програмування оболонки містить оператори, що порушують нормальне виконання циклу. Ці оператори мають назви – `break` (виконує безумовний вихід з циклу) і `continue` (програма негайно переходить до наступної ітерації циклу без виконання решти команд в циклі).

Контрольні питання

1. Що таке сценарій? Як відбувається запуск сценарію?
2. Що таке змінна? Яка область видимості змінної?
3. Які управляючі конструкції Ви знаєте?
4. Які засоби уведення-виведення використовуються в сценаріях?

Завдання до виконання

1. Скласти сценарій, який приймає 3 цілочисельні аргументи a, b і c (з командного рядка) і виводить значення (a+b)/c. При цьому дати можливість ввести параметри,яких бракує (якщо сценарій запускається з одним, двома параметрами або взагалі без них).
2. Скласти сценарій, який приймає 2 цілочисельні аргументи a і b і виводить номер більшого з них.
3. Скласти сценарій, який чекає введення будь-якого рядка і після цього замінює у введеному рядку всі символи «a» на символ «b», а символи «b» - на символ «a».
4. Написати сценарій, який по введеному імені користувача (логіну) визначає, чи існує користувач в системі і видає відповідне повідомлення.
5. Написати сценарій, який по визначає файл в каталозі (ім'я каталогу передається в якості аргумент сценарію), що містить максимальну кількість рядків.
6. Скласти сценарій, який серед переданих йому цілочисельних аргументів визначає кількість позитивних, негативних і нульових елементів.
7. Скласти сценарій, який кожну хвилину записує у файл час і поточне число процесів в системі. При запуску сценарій повинен створювати файл в директорії /tmp і записувати в нього свій PID.

СПИСОК ЛІТЕРАТУРИ

1. Семеренко В. П., Крилик Л. В. Операційна система Linux.- Навчальний посібник.-Вінниця: ВНТУ, 2006. - 88 с.
2. Робачевский А.М. Операционная система UNIX. – СПб.: БХВ - Петербург, 2002. – 528 с.
3. Глас Г., Эйблс К. Unix для программистов и пользователей. \Пер. с англ.– СПб.: БХВ–Петербург, 2004. – 848 с.
4. Белломо М. Unix: наглядный курс освоения операционной системы. – Киев: Диалектика, 2001. – 336 с.: ил.
5. Скловская С.Л. Команды Linux. Справочник, 3-е изд., перераб. и доп. – СПб.: ООО «ДиаСофтЮП», 2004. – 848 с.
6. Сивер Э., Спейнауэр С., Фиггинс С., Хекман Д. Linux. Справочник: Пер. с англ. – СПб.: Символ-Плюс, 2004. – 256 с.
7. Костромин В. А. Linux для пользователя. – СПб.: БХВ - Петербург, 2002. – 672 с.