Київський національний університет імені Тараса Шевченка

Факультет радіофізики, електроніки та комп'ютерних систем

Кафедра комп'ютерної інженерії

Програмування в середовищі UNIX-систем

Методичні вказівки до виконання лабораторних робіт

з освітньої компоненти "Програмування для UNIX-систем"

для студентів спеціальності

F7 "Комп’ютерна інженерія"

Київ 2026

Програмування в середовищі UNIX-систем. Методичні вказівки до виконання лабораторних робіт з освітньої компоненти «Програмування для UNIX-систем» для студентів спеціальності F7 «Комп'ютерна інженерія» [Електронний ресурс] /

Укладачі С.Д.Погорілий, П.В.Білецький.

– К.: Київський національний університет імені Тараса Шевченка, 2026. – 80 с.

Рецензенти: В.А.Заславський, д.т.н., проф.
Ю.А.Коба, к.т.н., доц.

Методичні вказівки до виконання лабораторних робіт включають рекомендації для підготовки, проведення, оформлення та здачі викладачеві звітів з семи лабораторних робіт з метою опанування методами роботи в операційній системі UNIX FreeBSD, створення файлів і сценаріїв різними засобами і мовами. Всі лабораторні роботи містять розділ «Контрольні запитання», матеріали якого дозволяють викладачеві визначити ступінь готовності студента до виконання поточної роботи.

Перша лабораторна робота призначена для засвоєння студентами методів і навичок інсталяції операційної системи UNIX FreeBSD на персональних комп'ютерах. У другій роботі студенти повинні засвоїти архітектуру операційної системи, її файлову систему, основні каталоги, їх призначення, і первісний набір команд. Третя робота присвячена опануванню основних текстових редакторів. У наступній лабораторній роботі вивчається потоковий редактор SED, надається поняття сценарію і вивчається процесор мови обробки шаблонів AWK.

Подальші лабораторні роботи присвячені різним методам створення сценаріїв. Задачею п'ятої лабораторної роботи є одержання навичок під час роботи із командним процесором shell, створення, налагоджування та виконання shell-сценаріїв. Шоста робота присвячена вивченню основних рекурсивних методів при програмуванні в shell і одержанню навичок при створенні, налагоджуванні та виконанні рекурсивних shell-сценаріїв і функцій. Сьома лабораторна робота передбачає одержання навичок при створенні, налагоджуванні та виконання сценаріїв мовами PYTHON та Java, а також системного програмування мовами C/C++.

Рекомендовано вченою радою факультету радіофізики, електроніки та комп'ютерних систем.

Протокол № 9 від 24.04.2026 р.

ЗМІСТ

Лабораторна робота № 1. Інсталяція операційної системи UNIX FreeBSD 4

Лабораторна робота № 2. Файлова система та основні команди FreeBSD..... 12

Лабораторна робота № 3. Текстові редактори в UNIX 26

Лабораторна робота № 4. Потоковий редактор SED. Процесор мови обробки шаблонів AWK 35

Лабораторна робота № 5. Командний процесор Shell 48

Лабораторна робота № 6. Рекурсія в сценаріях Shell..... 63

Лабораторна робота № 7. Програмування сценаріїв мовою PYTHON. Системне програмування..... 72

Лабораторна робота № 1

Інсталяція операційної системи UNIX FreeBSD

Метою роботи є опанування методики інсталяції ОС UNIX FreeBSD та одержання первісних навичок її конфігурації.

Інсталяція ОС UNIX FreeBSD із погляду подальшого функціонального застосування можлива як сервера, робочої станції, шлюзу тощо. В лабораторній роботі розглянуто загальні питання інсталяції ОС FreeBSD версії 14.1 для робочої станції. Основні принципи інсталяції операційної системи залишаються незмінними значний час, що дає можливість застосування наведеної послідовності і для більш ранніх версій ОС.

Інсталяція ОС UNIX FreeBSD можлива із різних носіїв: CD-диска, DVD-диска, FTP-сервера, дискети, FAT та FreeBSD-розділів, NFS (для інсталяції за NFS та з FreeBSD-розділу необхідна встановлена система UNIX).

Конфігурація технічних засобів, що рекомендується для інсталяції та подальшого ефективного функціонування ОС FreeBSD залежить від налаштувань операційної системи (наприклад, чи буде використано графічний інтерфейс для роботи) та необхідного програмного забезпечення (наприклад: HTTP-, FTP-, DNS-сервери та інші).

У лабораторній роботі буде розглянуто інсталяцію на віртуальну машину.

Перед початком інсталяції потрібно з'ясувати тип відеокарти для подальшого встановлення X-сервера.

Інсталяція ОС FreeBSD на віртуальну машину

Нижче наведено типову послідовність дій для інсталяції ОС FreeBSD на віртуальну машину:

1. Виберіть програмне забезпечення для віртуалізації (наприклад VMware, VirtualBox)
2. Завантажте образ системи FreeBSD 14.1 з офіційного сайту проекту [1].
3. Налаштуйте віртуальну машину (виберіть кількість процесорів, обсяг пам'яті, розмір віртуального диска тощо).
4. Запустіть віртуальну машину і підключіть образ диску ОС UNIX FreeBSD.
5. Для продовження процесу інсталяції заповнити низку меню:
   5.1. у меню Welcome to FreeBSD виберіть пункт меню за замовчуванням, або дочекатися його автоматичного спрацювання;
   5.2. у меню Keymap Selection виберіть Ukraine, це впливає на розкладку клавіатури;
   5.3. у меню Set Hostname введіть hostname для цієї машини;
   5.4. у пункті меню Distribution Select вибрати kernel-dbg, lib32, src (здійснюється натисканням клавіші Пробіл (Space), після чого з'являється позначка біля відповідного пункту);

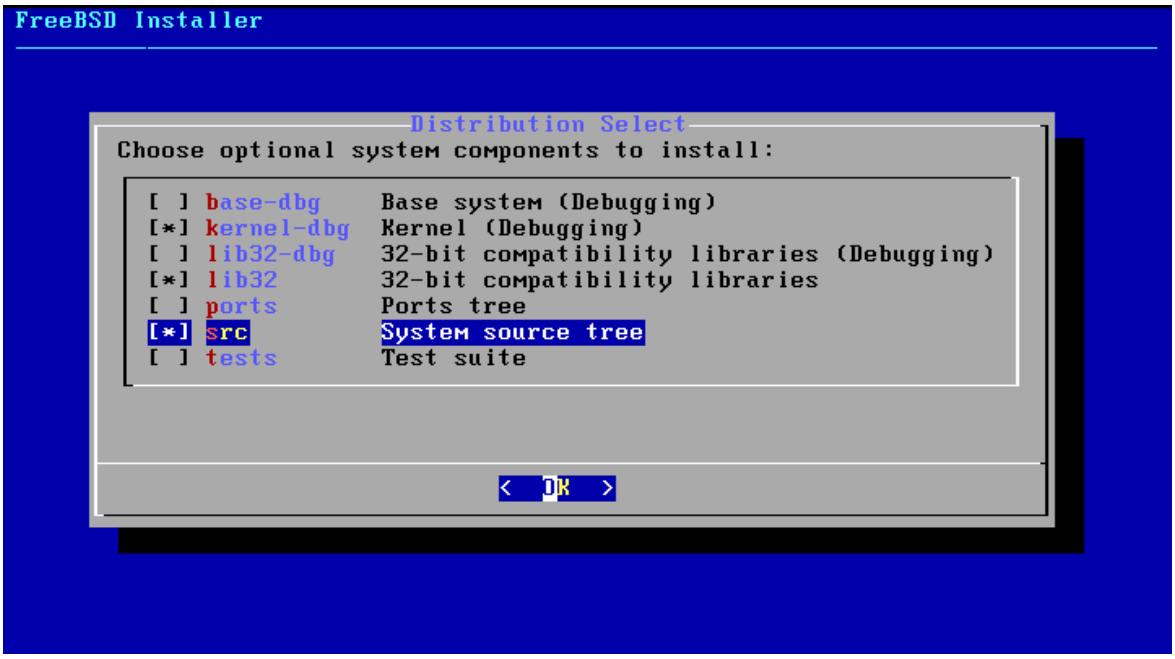

Рис. 1.1

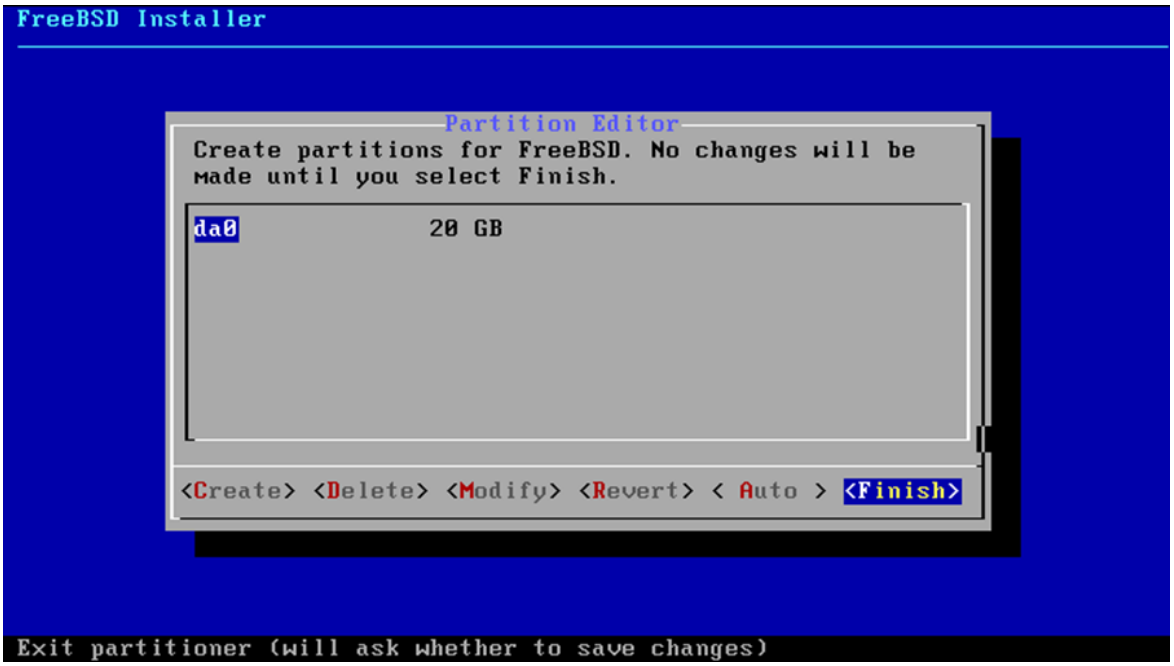

Рис. 1.2

5.5. у меню Partitioning виберіть пункт Manual. Далі, в Partition Editor можна створити freebsd-розділ (натиснути клавішу C (Create)). У наступних двох вікнах Value Required буде запропоновано задати розмір розділу та його тип.

5.6. для автоматичного створення підрозділів у меню Partition Editor натисніть клавішу A (виберіть пункт Auto). Далі оберіть Entire Disk та MBR (також, можна використовувати GPT). У цьому пункті меню можна модифікувати обсяги пам'яті для каталогів системи (видаляючи розділи та створюючи нові з іншим розміром): / (кореневий каталог), var, usr. Необхідно пам'ятати, що SWAP-розділ (область підкачки) має бути від 4 Гб, а для систем з малим об’ємом пам’яті ще більшим;

5.7. вибрати пункт меню Commit для початку інсталяції. Інсталяція триває, як правило, кілька хвилин;
5.8. після закінчення процесу інсталяції необхідно задати додаткові опції:
  5.8.1. встановити пароль адміністратора системи root (пункт меню RootPassword);
  5.8.2. налаштувати мережу;
  5.8.3. встановити дату та час;
  5.8.4. додати користувачів (для виконання лабораторної потрібно створити двох користувачів з налаштуваннями за замовчуванням);
  5.8.5. налаштувати мишу (у пункті меню System Configuration вибрати пункт moused);
5.9. завершіть налаштування (рис. 1.3);

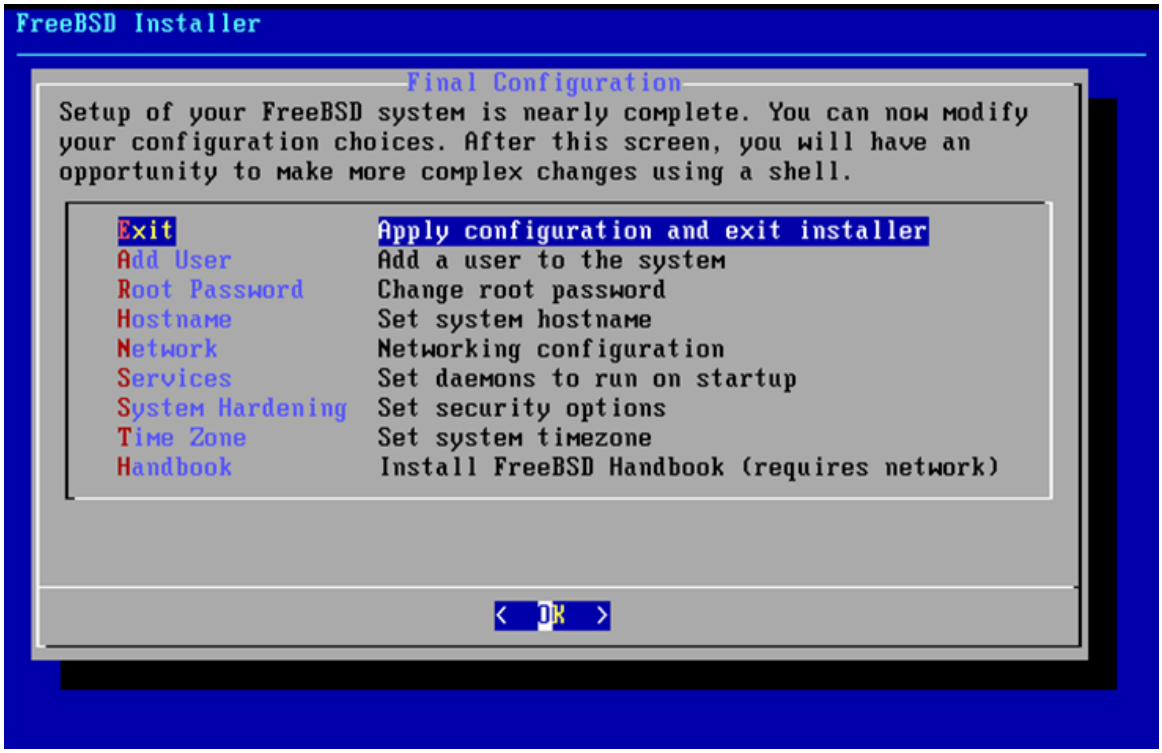

Рис. 1.3

Система X-Window

X-Window являє собою бібліотеку графічних програм, що використовуються для створення графічного користувальницького інтерфейсу. X-Window базується на моделі інформаційних систем клієнт–сервер. X-сервер – це програма, яка забезпечує взаємодію між інформацією, що вводиться через монітор (з використанням клавіатури або миші) і прикладними задачами. Функцією X-клієнта є обробка повідомлень, що отримані від X-сервера, робота із різними ресурсами системи (файловою системою, запуск і припинення процесів тощо). До X-Window входить більше п'ятдесяти X-клієнтів, які мають свої вікна та можуть працювати одночасно. X-клієнти не є стандартним інтерфейсом і можуть бути

налагоджені на будь-якому сервері. X11 – це одна з версій бібліотеки X-Windows, яка використовується в ОС FreeBSD, ця версія бібліотеки реалізована в пакеті Xorg.

У табл. 1.1 наведено деякі з клієнтів.

Таблиця 1.1

Тип	X-клієнт	Використання
Об'єкти робочого столу	xbiff	Поштова скринька
	Xclock oclock	Годинник
	xcalc	Калькулятор
	xman	X-версія команди man
Вхідні характеристики монітора	xmodmap	Визначає функції клавіш і кнопок
	xset	Визначає ресурси за замовчанням (опції зберігання екрана, швидкість руху курсору тощо)
	xsetroot	Встановлює характеристики кореневого вікна
Регулятори шрифту	xfd	Виводить символ у визначеному користувачем шрифті
	xfontsel	Перелічує дозволені шрифти
Регулятори графіки	bitmap	Растровий редактор
	xmag	Збільшувач
Керування вікном	xdpyinfo	Перелічує характеристики монітора
	xlsclients	Виводить імена клієнтів, які активізовано на поточному дисплеї
	xprop	Показує реквізити поточного вікна
	xwininfo	Показує характеристики вікна
Керування ресурсами	appres	Показує можливі ресурси для визначеного клієнта
	editres	Дозволяє тестувати та редагувати змінні ресурсів

Інсталяція X-Window

Так само, як й інсталяцію системи, інсталяцію X-Window можна проводити з різних носіїв. Розглянемо інсталяцію X-Window з мережі.

Якщо встановлювати X-Window після інсталяції, то потрібно додатково інсталювати програму Xorg. Для цього у командному рядку потрібно набрати команду pkg install xorg (попередньо Ви мали налаштувати мережу). По завершенні інсталяції файлу потрібно здійснити конфігурацію X-сервера, яка починається з введення команди Xorg -configure. Під час конфігурування виконується налаштування таких пристроїв як: миші, клавіатури, відеокарти та монітора. Опис встановлення та налаштування наведено на офіційному сайті FreeBSD [1]. Детальний опис встановлення KDE з особливостями конфігурації для VirtualBox можна знайти за посиланням [3].

Після встановлення X-Window потрібно вибрати та встановити один із віконних менеджерів, наприклад KDE, GNOME, VUE, CDE тощо.

KDE (K Desktop Environment) – вільно розповсюджувана віконна система, призначена для всіх основних платформ UNIX, надає користувачам єдине середовище та набір сервісних засобів. Зовнішній вигляд KDE нагадує робочій стіл Windows. KDE підтримує багато застосунків та інтегровану офісну систему Koffice.

Важливо відзначити, що під час роботи із графічним інтерфейсом у KDE існує можливість практично миттєво звернутися до інтерфейсу командного рядка. Для цього в KDE є спеціальне вікно Console.

Етапи початкового завантаження ОС

Зазвичай початкове завантаження системи складається з таких етапів:

- завантаження та ініціалізація ядра;
- розпізнавання та конфігурація пристроїв;
- створення системних процесів;
- виконання командних файлів завантаження системи;
- перехід до багатокористувацького режиму.

Ядро ОС FreeBSD – це програма, яка на етапі початкового завантаження зчитується у пам'ять для наступного виконання та відповідає за реалізацію таких елементів структури:

- процесів (процес – це послідовність операцій при виконанні програми, що являють собою набори байтів, інтерпретовані центральним процесором як машинні інструкції, дані та стекові структури);
- сигналів і семафорів;
- віртуальної пам'яті (свопінг, підкачування, керування пам'яттю);
- файлової системи (файли, каталоги);
- каналів і мережних з'єднань (взаємодія між процесами).

Після завантаження ядро намагається знайти та ініціалізувати всі указані в ньому пристрої. Якщо вказано драйвери відсутніх пристроїв, то їх буде вимкнено (навіть якщо пристрій підключений пізніше, він не буде доступний до перезавантаження системи).

Після закінчення базової ініціалізації ядро створює у пам'яті кілька процесів:

swapper – процес 0 (процес системного рівня, що завантажує процеси у пам'ять і, якщо йому не вистачає місця у пам'яті, вивантажує звідти деякі із процесів);

init – процес 1 (процес користувальницького рівня, що породжує інші процеси та ініціалізує нові);

pagedaemon – процес 2.

На етапі виконання командних файлів завантаження системи виконуються певні файли. Вони зберігаються у каталозі /etc, починаються літерами rc і виконують функції:

- установлення імені комп'ютера;
- установлення часового пояса;

- перевірка дисків командою **fsck** (в автоматичному режимі);
- монтування системних дисків;
- вилучення файлів із каталогу **/tmp**;
- конфігурація мережевих інтерфейсів;
- запуск процесів-демонів і мережевих служб;
- включення контролю квот (дозволяють обмежити кількість індексних дескрипторів і дискових блоків, що може бути надано кожному користувачу).

Нижче перелічено кілька конфігураційних файлів, що виконуються при завантаженні ОС: **rc, rc.bsdextended, rc.conf, rc.d, rc.firewall, rc.initdiskless, rc.resume, rc.sendmail, rc.shutdown, rc.subr, rc.suspend**. Більшість з них виводять на консоль детальну інформацію про виконувані задачі. Це надає суттєву допомогу при пошуку причин зависання системи у процесі початкового завантаження.

Крім того, існують конфігураційні файли, індивідуальні для кожного користувача. Наприклад, деякі із файлів запуску: .login (встановлює тип термінала та змінні середовища), .cshrc (встановлює псевдоніми команд, шлях їх пошуку, значення umask для контролю повноважень і змінні cdpath, prompt, history, savehist), .profile (аналог .login і .cshrc для команди sh), .mailrc (визначає персональні почтові псевдоніми та встановлює програми читання пошти), .xinitrc (встановлює початкове середовище X11: шрифти, кольори тощо).

Приклад файлу запуску .profile.

```
# $ FREEBSD : src/share/skel/dot.profile v1.212010/12/21
  23:24:03 kensmithExp$
#
PATH=/sbin:/bin:/usr/sbin:/bin:/usr/bin:/usr/local/bin: ~/bin; export
PATH
HOME=/home/user1 export HOME
TERM=${TERM:-cons25}
export TERM
BLOCKSIZE=K;  export BLOCKSIZE
EDITOR=vi;    export EDITOR
PAGE=more;    export PAGE
ENV=$HOME/.shrc;  export
```

Після виконання командних файлів запуску процес init породжує процеси getty, що належать кожному порту термінала. Процес getty встановлює початкові характеристики порту (швидкість передачі даних, контроль парності тощо) і виводить на екран реєстраційне запрошення login:. Процес завантаження ОС закінчено.

Створення груп і додавання користувачів

Кожний користувач в ОС FreeBSD є членом групи. Різним групам можна призначити різні можливості та повноваження. Файл /etc/group містить імена UNIX-груп і списки членів кожної групи. Кожний рядок файлу містить

інформацію про окрему групу та складається із чотирьох полів, відокремлених двокрапками:

- ім'я групи;
- пароль (це поле не використовується);
- номер GID (ідентифікаційний номер групи, ціле число у діапазоні від 0 до 32767);
- список членів групи, через коми (користувач може бути членом кількох груп).

Для створення нової групи потрібно лише відредагувати файл /etc/group за допомогою текстового редактора.

Процес створення користувача включає етапи:

- редагування файлу /etc/passwd з метою визначення бюджету користувача;
- встановлення паролю;
- створення домашньої директорії для нового користувача;
- копіювання у домашню директорію нового користувача файлів запуску, що використовуються за замовчанням;
- встановлення адреси електронної пошти та створення поштових псевдонімів;
- встановлення дискових квот;
- додавання користувача у групу.

Для додавання користувача існує кілька способів. У лабораторній роботі буде розглянуто спосіб додавання користувача за допомогою команди adduser. Спочатку потрібно відповісти на кілька запитань: ім'я домашньої директорії і користувача; ім'я командного процесора, який обере користувач; пароль тощо. Команда adduser також копіює шаблони конфігураційних файлів у каталозі /etc/skel у робочу директорію користувача. Вона являє собою файл сценарію командного процесора bash і знаходиться у каталозі /usr/sbin, тобто його можна змінити для розв'язання специфічних задач при додаванні користувача, наприклад, додати запитання додаткової інформації про користувача.

Контрольні запитання

1. Відомі Вам способи, окрім розглянутого у лабораторній роботі, додавання користувачів у системі.
2. Яку інформацію містить файл /etc/master.passwd, etc/passwd?
3. Перелічіть основні поля файлу etc/passwd.
4. Призначення конфігураційних файлів, що виконуються при завантаженні ОС.
5. Назвіть кілька змінних, що містять файли конфігурації, їх призначення.
6. Яку дію виконує команда vipw?
7. Яку дію виконує команда edquota?
8. Які дії треба виконати для вилучення користувача?
9. Назвіть команди, що використовуються для перезавантаження ОС FreeBSD.

10. Призначення та правила формування файлу .profile.
11. Які процеси створюються при завантаженні ОС UNIX фірми AT&T, які дії вони виконують?
12. У чому полягає концепція використання X-термінала?
13. У чому полягає концепція використання X-сервера?
14. Відомі Вам графічні оболонки. Зробіть їх порівняльний аналіз.
15. Основоположні концепції архітектури інформаційних систем клієнт– сервер.
16. Які ще архітектури інформаційних систем Вам відомі?

Завдання

1. Виконати інсталяцію ОС FreeBSD на віртуальну машину.
2. Виконати інсталяцію в ОС FreeBSD програми X-Window.
3. Створити два профіля користувачів в системі.
4. Виконати інсталяцію графічної оболонки KDE (GNOME).
5. Виконати конфігурацію файлу .profile. Переглянути вміст конфігураційних файлів, які виконуються при завантажені системи, надати коментар.

Список літератури

1. http://www.freebsd.org – Офіційний сайт проекту FreeBSD.
2. http://wiki.freebsd.org/ – Офіційний FreeBSD Wiki.
3. https://community.kde.org/FreeBSD/Setup – Посібник з встановлення KDE на FreeBSD.
4. https://docs.freebsd.org/en/books/handbook/ – Офіційна документація FreeBSD Handbook.
5. https://www.freebsd.org/releases/14.1R/ – FreeBSD 14.1 документація.
6. https://docs.freebsd.org/en/books/handbook/bsdinstall/ – Посібник з інсталяції FreeBSD.

Лабораторна робота № 2

Файлова система та основні команди FreeBSD

Метою роботи є ознайомлення із файловою системою та вивчення основних команд ОС FreeBSD.

Файлова система – це структура, за допомогою якої ядро ОС упорядковує та відображує для користувача ресурси пам'яті, розподілені на різного роду носіях інформації: жорстких, гнучких дисках і CD-ROM.

Кожний комп'ютер, на якому встановлено ОС, має хоча б одну файлову систему на жорсткому диску, що є кореневою та завантажується при вмиканні живлення комп'ютера. Вона включає ядро FreeBSD і залишається активною впродовж сеансу роботи. Якщо на комп'ютері з ОС FreeBSD існують інші файлові системи, то вони можуть бути змонтовані або розмонтовані. Файлова система ОС FreeBSD є ієрархічною структурою та може бути представлена графом у вигляді дерева. Коренем файлової системи є каталог /(root directory). Файлове дерево формується з окремих частин, файлових систем, які можуть приєднуватися (монтуватися) до файлового дерева за допомогою команди mount. Монтування здійснюється в один із каталогів кореневої файлової системи, ім'я каталогу стає коренем змонтованої файлової системи. При виході із системи потрібно розмонтувати всі додатково змонтовані файлові системи за допомогою команди umount.

Запам'ятайте: **коренева файлова система ніколи не може бути розмонтована; файлова система не може бути розмонтована, доки її використовують; пошкоджена система не може монтуватись.**

У теперішній час у системі ОС FreeBSD використовується файлова система UFS (Unix File System), створена для сімейства ОС BSD, яке складається з наступних ОС: FreeBSD, OpenBSD, NetBSD. Ця файлова система підтримується також ОС Linux та Solaris.

UFS має один завантажувальний блок, суперблок і список індексних дескрипторів. Суперблок і список індексних дескрипторів поділяються на менші компоненти, що містяться поміж блоків даних. Файлова система UFS дозволяє створювати блоки розміром 8192 байт та підтримує імена розміром 255 байтів.

Перелік стандартних каталогів

Каталоги являють собою логічно впорядковані сховища для файлів системи. Користувачі мають змогу створити власні каталоги, але у системі існує й низка стандартних. Деякі з них наведено у табл. 2.1.

Таблиця 2.1

Шлях до каталогу	**Вміст каталогу**
/	Кореневий каталог файлової системи
/bin/	Команди забезпечення мінімального рівня працездатності системи: cat, chmod, cp, date, file, find, grep, mkdir, rmdir, rm, pwd, who

Шлях до каталогу	Вміст каталогу
/boot/	Програми та конфігураційні файли, необхідні для завантаження ОС
/dev/	Файли пристроїв
/etc/	Важливі файли конфігурації і завантаження системи (crontab, fstab, group, hosts, make.conf, passwd, rc.conf тощо)
/lib/	Бібліотеки С-компілятора
/tmp/	Тимчасові файли (знищуються при перезавантаженні системи)
/proc/	Образи всіх працюючих процесів у системі
/usr/bin/	Застосунки користувача та загальні застосунки. Включає команди, такі як awk, cut, man, more, news, vi тощо
/usr/include/	Зберігає різні некомпільовані процедури, які використовуються при компіляції ядра операційною системою
/usr/lib/	Файли стандартних бібліотек
/usr/local/man/	Сторінки електронних файлів допомоги, які містять інформацію про роботу кожної команди UNIX
/var/spool/	Буферні каталоги для принтерів, електронної пошти, UUCP тощо
/var/log/	Різноманітні файли системних журналів
/var/tmp/	Каталог для тимчасового зберігання файлів (після перевантаження файли не зникають)
/usr/local/bin/	Локальні виконувані файли
/usr/local/etc/	Локальні системні файли та команди конфігурації

Файли в UNIX

В ОС UNIX існує вісім типів файлів (табл. 2.2):

Таблиця 2.2

Тип файлу	Символ, яким тип файлу позначається в UNIX
Звичайні файли	–
Каталоги	d
Байт-орієнтовані файли пристроїв	c
Блок-орієнтовані файли пристроїв	b
Іменовані канали (FIFO)	p
Доменні гнізда UNIX	s
Жорсткі посилання	
Символічні посилання	l

Звичайний файл – це логічно впорядкована послідовність байтів. Формат файлу не контролюється ОС UNIX, уміст файлу залежить тільки від програми, яка його обробляє.

Каталог – це теж файл, тобто послідовність байтів. Але формат каталогу контролюється ядром ОС. Інформація, що зберігається у каталозі, являє собою ім'я файлу та номер індексного дескриптора; створюється командою mkdir, а знищується – rmdir.

Особливістю ОС UNIX є те, що її ядро розглядає все як файли (послідовність байтів), навіть зовнішні пристрої: дисководи, термінали, модеми, мережеві карти, жорсткі диски тощо.

Байт-орієнтовані файли пристроїв зчитують і записують інформацію символ за символом (байтом). Вони дозволяють зв'язатись з ними драйверам цих пристроїв при виконанні власної буферизації введення/виведення; створюються командою mknod, а знищуються – rm.

Блок-орієнтовані файли пристроїв обробляються драйверами, які виконують введення–виведення великими порціями (блоками) і передають виконання задач буферизації ядру ОС. Як правило, розмір блока кратний 512.

Іменований канал використовується для можливості кільком програмам, які працюють у системі, надсилати інформацію одна одній або керуючому процесу. Він забезпечує взаємодію двох незалежних процесів, що виконуються на одному комп'ютері. Іменований канал (буфер FIFO) поводиться як звичайний файл, тобто при запису збільшується, але на відміну від звичайного файлу при зчитуванні його розмір зменшується. Іменований канал створюється та знищується як у попередньому випадку.

На відміну від каналів доменні гнізда дозволяють організовувати взаємодію між процесами, які виконуються на різних підключених до мережі комп'ютерах. Файл-гніздо створюється, якщо виникає необхідність зв'язатися з іншим процесом у мережі. Інструментальні засоби Internet (а саме Web-броузери) використовують гнізда для зв'язку із серверами; створюються командою socket(2), знищуються командою rm або системним викликом unlink.

Жорстке посилання – це спеціальний тип файлу, який дозволяє одному файлу мати кілька імен (що дозволяє ввести для імені існуючого файлу синоніми). Жорсткі посилання мають два обмеження: файл і друге ім'я мають бути частиною однієї файлової системи, жорсткі посилання не можуть забезпечувати друге ім'я каталогу, вони працюють тільки із файлами; створюється командою ln, знищується – rm.

Символічне посилання, на відміну від жорсткого, може використовуватись як для файлів, так і для каталогів, а також з'єднувати файлові системи; створюється командою ln -s, знищується – rm.

Права доступу (повноваження)

У кожного файлу існують права доступу (повноваження), що визначають, хто та які операції може виконувати із файлом. В ОС UNIX існують три категорії користувачів файлів: власник, група власників та інші користувачі. Власник файлу має можливість встановлювати, а за потреби – й змінювати (команда chmod) права доступу до свого файлу чи каталогу. Для кожної категорії користувачів існують окремі набори бітів. Кожний користувач

належить тільки до одної із категорій, що відповідає одному із трьох наборів бітів прав доступу. У свою чергу кожний набір складається із трьох бітів: старшого – біта дозволу читання, середнього – біта дозволу запису та молодшого – біта дозволу виконання (для директорій останній називається бітом пошуку). Комбінація бітів для файлу може дозволити чи заборонити читання, модифікацію та виконання файлу. Дія каталогу встановлення біта виконання дозволяє вхід до нього, але при цьому не має можливості отримання списку його вмісту. Установка комбінації бітів читання та виконання дозволяє отримати список умісту каталогу. Комбінація бітів запису і виконання дозволяє створювати, вилучати та перейменовувати файли у даному каталозі. В ОС UNIX існує особливий користувач – суперкористувач або адміністратор, який може читати та змінювати будь-який файл у системі. Ці повноваження надає суперкористувачу його ім'я root, яке використовується адміністраторами системи для виконання робіт з її підтримки. Існує команда su, яка дозволяє користувачу тимчасово отримати статус суперкористувача за умови, що йому відомий пароль при вході під ім'ям root.

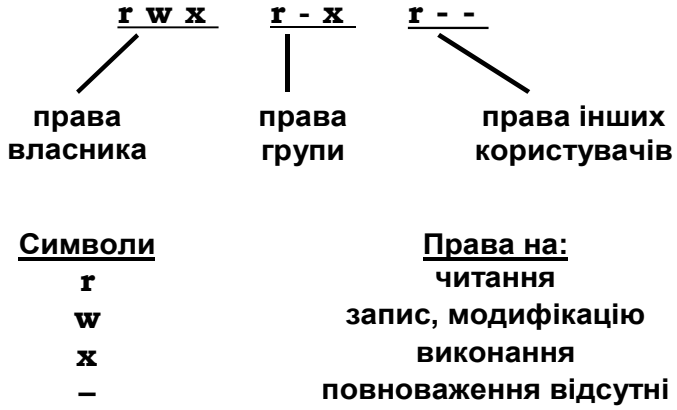

Рис. 2.1

Для зміни права доступу до файлу чи каталогу існує команда chmod із таким синтаксисом:

chmod режим файл(и)

Режим – це повноваження, які необхідно модифікувати. Існує два способи використання цієї команди: символьний та абсолютний. У символьному форматі для зміни повноважень використовуються літери, в абсолютному – вісімкові цифри, що представляють різні режими повноважень.

Коди прав доступу в абсолютному режимі подано у табл. 2.3.

Таблиця 2.3

Вісімкове число	Коди прав доступу	Вісімкове число	Коди прав доступу
0	---	4	r--
1	--x	5	r-x
2	-w-	6	rw-
3	-wx	7	rwx

Наприклад, абсолютний спосіб зміни повноважень chmod 711 myfile, символьний – chmod u=rwx, g=x, o=x myfile

Введення–виведення інформації в ОС UNIX

В ОС UNIX більшість команд за замовчанням передбачає виведення інформації на термінал (екран) і введення з термінала (клавіатури). Додатково дозволяється перепризначення введення/виведення, тобто зміни місця, до якого направлятиметься виведення команди чи будуть взяті вхідні дані для команди. Для перепризначення введення/виведення використовуються такі символи:

> – перепризначення виведення; створює файл, якщо він не існує, чи перезаписує файл за його існування;

>> – перепризначення виведення; дописує вхідний потік у кінець файлу чи створює файл, якщо він не існує;

< – перепризначення введення, указує, що вихідний потік запозичується із файлу;

<< – указує на конструкцію тут документ.

Синтаксис:

```
команда символ_перепризначення файл
```

Наприклад,

```
who > /tmp/myfile
```

Зазвичай на початку виконання кожної програми в ОС UNIX визначено три файли зі стандартними дескрипторами: стандартні вхідний потік stdin (0), вихідний потік stdout (1) і файл помилок stderr (2). Тому можна використовувати такі конструкції:

2> – перепризначення стандартного файлу помилок;

2>&1 – перепризначення стандартного файлу помилок у стандартний вихідний потік.

Ще одним перепризначенням введення/виведення в ОС UNIX є конвеєри. Конвеєр – це спосіб з'єднання результату виконання однієї команди зі вхідною інформацією другої. Конвеєри дозволяють зменшити потребу у тимчасових файлах, які використовуються другою командою, а потім знищуються. Символ | указує на конвеєр, з'єднує виведення команди, що знаходиться ліворуч, із введенням команди, що розташована праворуч.

Синтаксис:

```
команда1|команда2
```

Наприклад,

```
who | sort
```

Ще одним поняттям ОС UNIX є фільтр – команда, яка читає вхідну інформацію, певним чином її переробляє та виводить результат у вихідний потік. Прикладами фільтрів можуть бути команди: grep і tail (трансформують частину вхідного потоку), sort (сортує вхідний потік), wc (проводить підрахунок елементів у вхідному потоку).

Часто фільтр має вилучати інформацію у вхідному потоку за шаблоном (регулярним виразом). Регулярний вираз – це спосіб описання рядка з

використанням метасимволів (табл. 2.4). Регулярні вирази широко використовуються в сучасних мовах програмування для опису текстових рядків в операціях пошуку, заміни, виділення інформації та ін.

Таблиця 2.4

c	Символ, що не має спеціального значення
.	Задає будь-який символ
*	Задає рядок, що складається з нуля або більше символів, у назві файлу
?	Задає будь-який символ у назві файлу
[ccc]	Задає будь-який символ із [ccc] у назві файлу (можливі діапазони, напр. 0-9 або a-z)
[^ccc]	Задає будь-який символ не із [ccc], можливе задання діапазону
\c	Скасовує спеціальний зміст наступного символу
'текст'	Захищає текст, тобто процесор його не інтерпретує
"текст"	Безпосереднє використання
`текст`	Виконує текст як команду (початковий та останній символи не апострофи, а символи слабкого наголосу)
^	Початок рядка
$	Кінець рядка
r*	Нуль або більша кількість входжень регулярного виразу r
rx	Конкатенація регулярних виразів r та x
r\{m, n}\	Кількість входжень регулярних виразів r у межах від m до n

Групи основних команд

У табл. 2.5 наведено деякі групи команд ОС UNIX (у дужках вказано найпоширеніші опції).

Таблиця 2.5

Тип	Команда	Опис
Команди навігації	cd	Команда зміни каталогу, якщо записана без параметрів, то виконується повернення до домашнього каталогу
	pwd	Виведення шляху до поточного каталогу
Команди перегляду каталогів і файлів	ls(a,C,F,l, d,R)	Виводить уміст каталогу: a – виводить усі файли, включаючи приховані; l – виводить у довгому форматі; d – показує тільки каталоги; R – рекурсивний перегляд
Команди перегляду вмісту файлу	cat	Перегляд невеликих файлів і надсилання текстового файлу іншій програмі за конвеєром
	more	Показує вміст файлу порціями розміром в один екран
Команди створення та вилучення	touch	Створює порожній файл у поточному каталозі
	mkdir	Створює порожній каталог
	rm	Вилучає файл або групу файлів Опція I переводить команду в інтерактивний

Тип	Команда	Опис
файлів і каталогів		режим виконання (використовується при вилученні файлів згідно із шаблоном)
	rmdir	Вилучає порожню директорію
Команди надсилання повідомлень	mail	Надсилання користувачу пошти або її перегляд
	mailx	Надає зручний і гнучкий інтерфейс для надсилання та отримання електронних повідомлень
	write	Надсилання повідомлення іншому користувачу
	mesg	Керує отриманням повідомлень від інших користувачів
	echo	Записує у стандартний вихідний потік свій аргумент або рядок
	news	Читає отримані поштою новини з каталогу /usr/news
Команди роботи із архівами	compress	Стискує дані для зберігання
	uncompress	Відновлює дані з архіву
	tar	Зберігає і відновлює файли в архів c – створює новий архів; r – дописує файли у кінець архіву; t – перегляд умісту архіву; x – розпаковує файли з архіву
	cpio	Створює архів або вилучає файли з існуючого архіву: -o – створює архів; -i – вилучає перелічені файли з архіву; -p – копіює список файлів у вказаний каталог; -t – друкує список файлів архіву (використовується разом з опцією -i)
Команди для створення текстових процесорів	lex	Виконує фазу лексичного аналізу компілятора
	yacc	Виконує фазу синтаксичного аналізу компілятора
Фільтри	grep, fgrep, egrep	Проводить пошук тексту за шаблоном у вказаних файлах або стандартному вхідному потоці (коли не вказано файли) і повідомляє, коли шаблон знайдено. Команда egrep може шукати комбінації регулярних виразів, для цього використовуються наступні регулярні вирази: + – "+" після регулярного виразу означає один або більше збігів; ? – нуль або один збіг; | – визначає множинні регулярні вирази; ( ) – групує регулярні вирази
	find	Проводить пошук файлів за вказаними критеріями (ім'я, розмір, час створення, час модифікації тощо)

Тип	Команда	Опис
		у вказаних каталогах. Опції: name – пошук за ім'ям файлу; type – пошук за типом файлу; size – пошук за розміром файлу
	head	Виводить указане число рядків з початку файлу
	tail	Виводить указане число останніх рядків файлу
	sort	Дозволяє сортувати за критерієм та об'єднувати текстові файли
	comm	Виконує порівнювання двох відсортованих файлів, тобто знаходить однакові для файлів рядки
	wc	Проводить підрахунок елементів у вхідному потоці
	tee	Поділяє вивід конвеєра в один або кілька файлів. Дозволяє записувати стандартний вивід у файл, одночасно проводити вивід через стандартний потік

Регулярні вирази у shell

Регулярні вирази – це один із способів пошуку підрядків (відповідностей) в рядках. Здійснюється це за допомогою перегляду рядка у пошуках деякого шаблону. Загальновідомим прикладом можуть бути символи * і ?, використовувані в командному рядку DOS. Перший з них замінює нуль або більше довільних символів, другий же – один довільний символ. Так, використання шаблону пошуку типа text?.* знайде файли textf.txt, text1.asp і інші аналогічні, але не знайде text.txt або text.htm.

Особливо корисні регулярні вирази в програмах, написаних на інтерпретованих мовах, наприклад, VBScript, JScript і Perl. Через те, що весь їх код інтерпретується, розбір текстових рядків і виразів виконується повільно. Застосування регулярних виразів підвищує продуктивність.

Зазвичай за допомогою регулярних виразів виконуються три дії:

- перевірка наявності відповідного шаблону підрядка;
- пошук і видача користувачеві відповідних шаблону підрядків;
- заміна відповідних шаблону підрядків.

В цей час існує ціла низка програм і мов, що використовують регулярні вирази (такі як awk, sed і lex). Розробники, яким подобалася певна можливість однієї програми часто намагалися реалізувати її в своїй програмі.

Для стандартизації регулярних виразів було створено стандарти POSIX.

Стандарт POSIX (скорочено від слів Portable Operating System Interface, тобто «переносимий інтерфейс операційної системи») був запропонований в 1986 році для забезпечення переносимості програм між операційними системами. Деякі частини цього стандарту пов'язані з регулярними виразами і традиційними засобами, в яких вони використовуються.

З урахуванням вже створених до цього часу програм, для роботи з регулярними виразами, POSIX робить спробу упорядкувати цей хаос і ділить поширені діалекти на дві категорії:

- BRE (basic regular expressions, тобто «базові регулярні вирази»).
- ERE (extended regular expressions, тобто «розширені регулярні вирази»).

Однією з важливих особливостей стандарту POSIX є поняття локального контексту (locate) – сукупність параметрів, що описують мовні і національні правила: формат дати, часу і грошової величини, інтерпретація символів активного кодування і так далі. Локальні контексти спрощують адаптацію програм для інших мов. Вони не відносяться до специфіки регулярних виразів, проте можуть впливати на їх застосування. Наприклад, метасимвол \w, що зазвичай позначає «символ слова» (у багатьох діалектах це поняття еквівалентне [a-zA-Z0-9]. POSIX не вимагає, але допускає підтримку цього метасимвола. За підтримки \w у пошук включаються всі букви і цифри, визначені в локальному контексті, а не тільки ті, які визначені в кодуванні ASCII.

Регулярні вирази у команді grep

Команда grep шукає у файлах рядки, що містять співпадання з наданим набором шаблонів. Ці рядки виводяться на стандартний вивід, формат виводу можна задати за допомогою параметрів.

Незважаючи на те, що grep призначений для пошуку текстової інформації, рядки у файлах можуть містити будь-які символи, та можуть бути будь-якої довжини. Якщо останній символ файлу не є символом нового рядка, grep все одно буде вважати такий останній рядок звичайним рядком з символом нового рядка на кінці. Оскільки символ нового рядка є також розділовим символом у списку шаблонів, немає ніякої можливості вказати цей символ безпосередньо у шаблоні. Наприклад:

```
> grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
> grep -i ps ~/.bash* | grep -v history
/home/cathy/.bashrc:PS1="\[\033[1;44m\]$USER is in \w\[\033[0m\]
"
```

Остання команда виводить рядки з усіх файлів у домашньому каталозі, що починаються з символів ~/.bash, крім тих рядків, що містять слово history (наприклад, щоб пропустити рядки з історії команд).

Границі рядків та слів

Щоб відобразити лише ті рядки, що починаються з root:

```
> grep ^root /etc/passwd
```

Рядки, що закінчуються на двокрапку:

```
> grep :$ /etc/passwd
```

Вираз у кутових дужках — це список символів, взятий у кутові дужки [та]. Він співпадає з будь-яким символом зі списку; якщо перший символ у

списку є символом ^, то вираз співпадає з будь-яким символом, якого немає у списку. Наприклад, регулярний вираз [0123456789] співпадає з будь-якою цифрою.

У списку можна вказати інтервальний вираз — два символи, розділені одним знаком мінуса. Цей вираз співпадає з будь-яким символом, що за алфавітом знаходиться між вказаних двох символів включно, згідно з системною локалізацією та символьним набором. Наприклад вираз [a-d] відповідає виразу [abcd]. Деякі локалі розміщують символи у словниковий лад, то ж в них [a-d] зазвичай не є еквівалентом [abcd]; наприклад, цей вираз може відповідати [aAbBcCdD]. Щоб повернути звичайну інтерпретацію інтервалів, можна встановити для процесу grep локаль C; для цього потрібно призначити змінної LC_ALL значення C.

Символ . у регулярному виразі співпадає з одним будь-яким символом, крім символу нового рядка. Декілька символів можна знайти за допомогою метасимвола *.

Усі вирази складаються з певних значущих символів та сутностей притаманних регулярним виразів. Приклади задання значущих символів, або слів (примітка: регістр має значення) наведені в табл. 2.6.

Таблиця 2.6

Регулярний вираз	**Пояснення**	**Співпадіння**	**Нема співпадіння**
a	Символ a	cat	Dog
a-f	Діапазон від a до f	cat, red	Dog
1	Цифра 1	1998	2000
0-9	Діапазон 0-9	1998	Dog
[at]	Група символів: a або t	Cat	Dog
[a-ct]	Діапазон a-c та символ t	Cat	Dog
^a	Інверсія. Не символ a	Cat	Aaa
cat	Слово cat	cat, catched	Dog
cat | dog	Одне зі слів: cat, dog	cat, dog	Fox
^	Початок рядку		
^a		ater	Cat
$	Кінець рядку		

Для найбільш вживаних комбінацій передбачені скорочення, табл. 2.7.

Таблиця 2.7

Скорочення	**Еквівалент**
\d	0-9
\D	^0-9
\w	0-9a-zA-Z_
\W	^0-9a-zA-Z_
.	Будь-який друкований символ

Також є можливість впливати на кількість повторень певного символу або групи символів у слові що перевіряється, табл. 2.8.

Таблиця 2.8

Квантифікатор	Пояснення
a{n, m}	Від n до m символів a, що йдуть підряд
a{n,}	Не менше n символів a, що йдуть підряд
a{,n}	Не більше n символів a, що йдуть підряд
a{n}	Точно n символів a, що йдуть підряд
a?	Символ a або зустрічається, або ні
a+	Символ a або зустрічається не менше одного разу
a*	Символ a або зустрічається скільки завгодно разів

Деякі групи символів можна логічно відокремлювати. Для цього використовують оператор круглі дужки (). До групи у круглих дужках можна застосовувати квантифікатори. Вираз для знаходження повного імені:

[А-ЯЄІЇ][а-яіїє’]*(\s[А-ЯЄІЇ][а-яіїє’]*)+

Видно що кожна складова імені (слово) починається з великої букви, після якої йдуть будь-які літери алфавіту. Якщо після нього йде наступна частина імені, то вони відокремлені пробілом, тому доцільно першу літеру, наступні літери та пробіл рахувати як одну групу [А-ЯІЇ][а-яії’]*\s, яка може бути повторена не менше одного разу.

Контрольні запитання

1. Назвіть відомі вам засоби створення файлів в UNIX.
2. Відомі вам способи перегляду вмісту файлу.
3. Що таке символічні зв'язки для файлів, каталогів?
4. Що таке повноваження для файлів? Як вони утворюються?
5. Як утворюються повноваження при створенні файлу за допомогою команди touch? Чи можна їх модифікувати?
6. Яку інформацію містить перший символ, що формується у першому стовпчику командою ls –l?
7. Що таке група користувачів, як вона утворюється?
8. Що таке дескриптор файлу?
9. Як здійснюється пошук файлу за його ім'ям, датою, розміром?
10. Що таке відносний та абсолютний шляхи? Як вони використовують?
11. Які файли містять порожній каталог? Для чого вони призначені?
12. Яку дію виконує команда umask?
13. Чи може користувач за допомогою команди chown утратити власні повноваження на файл?
14. Чи можливо примусово демонтувати файлову систему під час її використання? Як це зробити?
15. Яку інформацію утримують файли inittab, magic, profile?
16. Як виконуються команди cut і paste?
17. Як переглянути вміст файлу у шістнадцятковому та вісімковому вигляді?
18. Що означають повноваження suid і sgid?
19. Запишіть регулярні вирази, які:
    - позначають слово The, що знаходиться на початку рядка;
    - позначають слово help;

- позначають літеру w, за якою слідують три будь-які символи, а потім літера d (це може бути слово world);
- позначає порожній рядок;
- позначає рядок, що починається з цифри;
- позначає рядок, що закінчується крапкою.

Завдання

1. Увійдіть до системи, використайте ім'я та пароль, отримані вами у попередній лабораторній роботі.
2. Комунікація користувачів:
   - подивіться, хто ще з користувачів працює у системі;
   - надішліть повідомлення Hello all! користувачам, що знаходяться у системі;
   - надішліть повідомлення одному із користувачів (якщо ваш login studi, то
   - надішліть повідомлення користувачу з ім'ям stud(i-1)), використайте команди mail, mailx і write. Зверніть увагу на різницю між цими командами;
   - перевірте, чи отримали ви пошту. Якщо так, то перегляньте її.
   - Встановіть режим ігнорування повідомлень.
3. Навігація у файловій системі:
   - Перед виконанням наступних завдання виконайте команди (від імені root):

```
mkdir -p /student/lab1/prohibit
chmod 000 /student/lab1/prohibit
echo "Test content" > /student/lab1/file1
```

   - подивіться назву поточного каталогу;
   - перейдіть до каталогу /student/lab1;
   - переконайтеся, що потрапили туди, куди треба;
   - перегляньте вміст каталогу, використайте короткий та довгий формати перегляду. Поясніть зміст кожного стовпчика у довгому форматі;
   - зайдіть у каталог prohibit. Поясніть, що сталося;
   - підніміться на два рівня вгору, використайте відносний шлях;
   - ознайомтеся із вмістом каталогів dev, etc, bin, usr. У каталозі dev покажіть файли для байт- та блок-орієнтованих пристроїв. Поясніть уміст перелічених каталогів;
   - ознайомтеся із вмістом файлів inittab і passwd;
   - поверніться до свого домашнього каталогу, використовуючи відносний шлях.
4. Операції над файлами та директоріями:
   - створіть у домашньому каталозі каталог book1;
   - скопіюйте із каталогу /student/lab1 усі файли, крім каталогу prohibit;
   - створіть у каталозі book1 власний файл;

- змініть повноваження власного файлу так, щоб власник міг читати, модифікувати та виконувати, група – читати та виконувати, всі інші – тільки виконувати;
- змініть повноваження на каталог book1 так, щоб власник міг переглядати вміст каталогу, додавати чи вилучати файли, для всіх інших – вхід в каталог заборонений.

5. Монтування та демонтування файлових систем:
   - змонтуйте USB-флешку та рекурсивно перегляньте її вміст;
   - демонтуйте USB-флешку;
   - змонтуйте ISO-образ, перегляньте його вміст (виведіть каталоги у "довгому" форматі);
   - демонтуйте ISO-образ.
6. Робота з архівами:
   - створіть архів arc і запишіть до нього кілька файлів із каталогу book1, використайте команду tar;
   - перегляньте вміст архіву;
   - скопіюйте архів у свою домашню директорію;
   - вилучіть файли з архіву.
7. Виконайте п. 6, використайте команду cpio.
8. Жорсткі та символічні посилання:
   - створіть жорстке посилання для файлу file1 на newfile;
   - створіть символічне посилання каталогу book1 на newdir;
   - перегляньте свій домашній каталог рекурсивно у довгому форматі. Поясніть останній стовпчик.
9. Фільтри та конвеєри:
   - здійсніть одночасний пошук слова hello та Hello у файлі letter;
   - здійсніть одночасний пошук слів something і somebody у файлі letter;
   - здійсніть пошук файлу file1 у поточному каталозі;
   - з'ясуйте, чи зареєстровано у системі користувач James_Bond;
   - виведіть п'ять перших рядків файлу letter;
   - виведіть десять останніх рядків файлу letter;
   - відсортуйте імена файлів у каталозі dev за алфавітом (розміром);
   - підрахуйте кількість файлів у каталозі dev;
   - перепризначте виведення команди ls –l у файл list;
   - перепризначте виведення команди ls –l у файл list1 та одночасно перегляньте цей список на екрані.
10. Модифікація рядка запрошення:
    - модифікуйте рядок запрошення так, щоб він містив поточний каталог (ім'я користувача);
    - продемонструйте роботу кількох команд;
    - відновіть початкове значення рядка запрошення.
11. Створіть псевдонім для команди cd, який після зміни каталогу відображатиме ім'я нового каталогу.

12. Вилучення файлів і каталогів:
    - Вилучіть файли з домашнього каталогу, створені під час лабораторної роботи.

Список літератури

1. https://docs.freebsd.org/en/books/handbook/ – керівництво FreeBSD.
2. https://docs.freebsd.org/en/books/handbook/basics/ – Основи FreeBSD.
3. https://docs.freebsd.org/en/books/handbook/basics/#dirstructure – структура директорій FreeBSD.
4. https://docs.freebsd.org/en/books/handbook/basics/#users-synopsis – управління користувачами та групами.
5. https://man.freebsd.org/cgi/man.cgi?query=grep&apropos=0&sektion=0&manpath=FreeBSD+14.3-RELEASE+and+Ports&arch=default&format=html – інструкція grep FreeBSD.
6. https://docs.freebsd.org/en/books/handbook/filesystems/ – файлові системи FreeBSD.

Лабораторна робота № 3

Текстові редактори в UNIX

Метою роботи є вивчення основних команд текстових редакторів vi та ed ОС UNIX та одержання навичок при роботі з ними.

У процесі роботи з ОС UNIX користувачу часто потрібно створювати та зберігати текстову інформацію. Для таких завдань в ОС UNIX постачаються текстові редактори, які дозволяють створювати або редагувати файли. Найпопулярнішими серед них є екранно-орієнтований редактор vi і рядково-орієнтований редактор, який працює тільки з одним рядком, ed (editor). Для ОС FreeBSD також часто використовують текстові редактори emacs та vim, вони відсутні в системі за замовчуванням, але їх можна знайти в колекції портів. Ці редактори вимагають значного часу на їх вивчення, але в подальшому дозволяють значно прискорити редагування файлів.

Текстовий редактор vi є одним із загальноприйнятих текстових редакторів на платформі UNIX. Назва редактора vi – це початкові букви словосполучення visual interpreter – візуальний інтерпретатор. Режим роботи редактора vi встановлюється системою автоматично при виконанні деяких команд (напр., для редагування імені файлу при використанні оболонки Korn Shell). Тому знання vi є обов'язковим для професійної роботи в UNIX.

Для запуску редактора vi необхідно набрати у командному рядку vi разом з ім'ям файлу, що редагуватиметься (або просто vi). Якщо файл для редагування існує, то на екран буде виведено його вміст, якщо ж це новий файл, то на екрані ви побачите стовпчик символів **~**, які означають порожній файл (рис. 3.1).

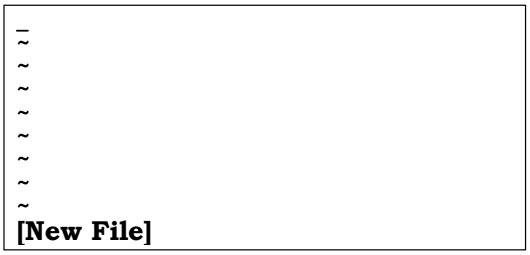

Рис. 3.1

Редактор vi має три робочі режими: командний (command), введення тексту (insert) та останнього рядка (last line mode). Режим останнього рядка характеризується тим, що всі команди починаються із символу ':', введення символу змушує курсор переміщуватися у нижній рядок екрана, де необхідно ввести завершальну частину команди. У цьому режимі кожна команда має закінчуватися натисканням клавіші Enter. При запуску vi автоматично встановлюється командний режим, який можна використовувати для пересування документом і внесення змін. У командному режимі всі клавіші та їх комбінації, включаючи індивідуальні клавіші, комбінації клавіш із клавішами Shift і Ctrl, функціонують як команди.

Режим введення тексту використовується для набору документу або внесення змін.

Щоб перейти з командного режиму до режиму введення тексту, потрібно натиснути клавішу a, i, o або r, а для повернення у командний режим – натиснути Esc (рис. 3.2).

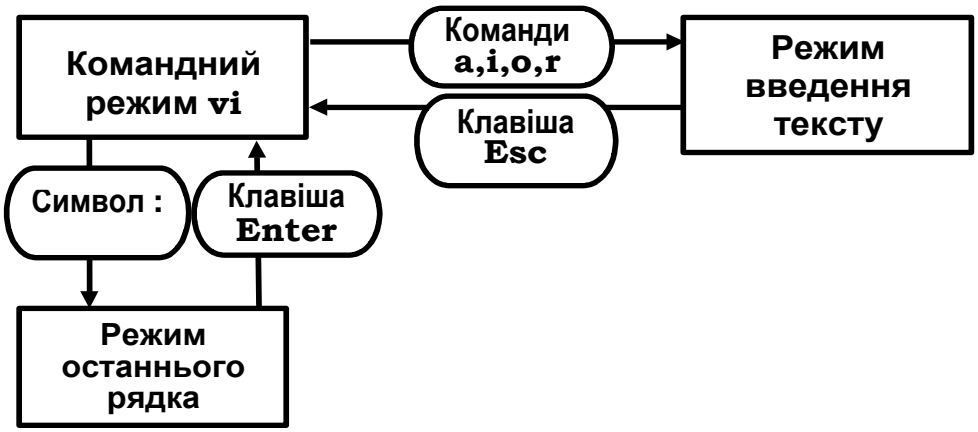

Рис. 3.2

Узагальнений синтаксис команд редактора vi:

```
:[префікс] команда [опції]
```

Префікс вказує на номер рядка або діапазон рядків до яких виконується команда.

Основні команди текстового редактора vi зведено у табл. 3.1.

Використання змінних і псевдонімів

При використанні команди abbr можна присвоювати значення змінним для автоматичного використання впродовж сеансу роботи у редакторі. Коли присвоєна така змінна, друге значення (як правило, довгіше) замінює перше.

Наприклад,

```
:abbr univ National Taras Shevchenko University
```

Для скасування скорочення без виходу із редактора використовується команда una.

Для створення псевдонімів команд використовується команда map із таким синтаксисом:

```
:map натискання_клавіші послідовність_команд
```

Наприклад,

```
:map ^U iNational Taras Shevchenko University
```

Таким чином, назва National Taras Shevchenko University of Kiev вставлятиметься (після переходу в режим введення тексту за рахунок команди ’i’) кожного разу при натисканні комбінації клавіш Ctrl–U в командному режимі. Це дає змогу, за бажанням користувача, призначити псевдоніми командам.

Перетворення команди map виконуються тільки у командному режимі. Аналогом команди map є команда map!, яка визначає заміни у режимі введення.

Налагодження робочого середовища vi

Редактор vi має кілька опцій роботи з монітором, які можна встановити на термін сеансу редагування. Вони викликаються у командному режимі за

допомогою команди :set ім'я_змінної. Для того, щоб побачити всі можливі змінні, треба набрати :set all; багато зі змінних встановлено за замовчанням, а щоб побачити встановлені для системи параметри – ввести команду :set.

Визначення деяких змінних:

- **number** – нумерує кожен рядок на дисплеї;
- **nonumber** – скасовує раніше встановлену нумерацію;
- **showmode** – вказує режим, в якому працює користувач (I – режим вставки, A – режим додавання);
- **noshowmode** – скасовує попередню змінну;
- **ignorecase** – ігнорує чутливість до регістру у всіх операціях пошуку;
- **noignorecase** – відновлює чутливість до регістру;
- **terse** – видає скорочені повідомлення про помилки;
- **noterse** – скасовує попередню зміну;
- **remap** – указує, чи активізована команда map;
- **window** – указує кількість рядків у текстовому вікні.

Таблиця 3.1

Тип	Команда	Опис
Команди редагування	a	Додавання тексту після позиції курсору
	Shift–a	Перехід у режим вводу та додавання тексту у кінець поточного рядка
	i	Додавання тексту перед позицією курсору
	Shift–i	Перехід у режим введення, додавання тексту із початку поточного рядка
	o	Вставка порожнього рядка після поточним для додавання тексту
	Shift–o	Вставка порожнього рядка перед поточного для додавання тексту
Команди читання – запису файлу	:r	Прочитати файл
	:w	Запис буфера у файл, що редагується vi
	:w ім'я_файлу	Запис буфера у вказаний файл
	:w! ім'я_файлу	Перезапис буфера у вказаний файл
Команди виходу із редактора	:q	Вихід, якщо дані у буфері не змінено, або вихід після модифікації буфера та зберігання змін у файлі
	:q!	Вихід з відміною всіх змін у буфері з часу останнього зберігання змін у файлі
	:wq, :x або ZZ	Запис буфера у робочій файл із наступним виходом із редактора
Команди пересування екраном	h	Зсув ліворуч на один символ
	j	Зсув вниз на один рядок
	k	Зсув угору на один рядок

Тип	Команда	Опис
	l	Зсув праворуч на один символ
	w	Перехід уперед на одне слово
	b	Перехід на початок поточного слова
	e	Перехід на кінець поточного слова
	+	Перехід до наступного рядка
	-	Перехід до попереднього рядка
	^	Перехід на початок поточного рядка
	$	Перехід на кінець поточного рядка
	(	Перехід на початок речення
	)	Перехід на кінець речення
	{	Перехід на початок абзацу
	}	Перехід на кінець абзацу
	Ctrl–f	Прокрутка вперед на екран
	Ctrl–b	Прокрутка назад на екран
	Ctrl–d	Прокрутка вперед на півекрана
	Ctrl–u	Прокрутка назад на півекрана
	:#	Перейти до рядка номер #
	#|	Перейти до стовпчика номер #
Команди вилучення тексту	x	Вилучення символу у позиції курсору (можлива комбінація із числами (префіксами), напр., 8x)
	dd	Вилучення поточного рядка (можлива комбінація із числами)
	dw	Вилучення поточного слова (можлива комбінація із числами)
	d0	Вилучення тексту, починаючи із позиції курсору до початку рядка
	d$	Вилучення тексту, починаючи із позиції курсору до завершення рядка
	dl	Вилучення наступної літери (можлива комбінація із числами)
	d)	Вилучення тексту, починаючи із позиції курсору до завершення речення
	d}	Вилучення тексту, починаючи із позиції курсору до завершення абзацу
Команди копіювання та вставки тексту	yy	Копіювання поточного рядка (можлива комбінація із числами)
	yw	Копіювання поточного слова (можлива комбінація із числами)
	y0	Копіювання тексту, починаючи із позиції курсору до початку рядка
	y$	Копіювання тексту, починаючи із позиції курсору до завершення рядка

Тип	Команда	Опис
	yl	Копіювання наступної літери (можлива комбінація із числами)
	y)	Копіювання тексту, починаючи із позиції курсору до завершення речення
	y}	Копіювання тексту, починаючи із позиції курсору до завершення абзацу
	p	Введення вмісту буфера після курсору
	P	Введення вмісту буфера перед курсором
	vi також має можливість зберігати більше одного буфера інформації. Для цього існують такі команди: "cY – копіює поточний рядок у буфер із позначкою c; "cP – вставляє вміст буфера c, де c набуває будь-якого значення від a до z	
Додаткові команди	J	З'єднання двох послідовних рядків
	u	Скасування результату виконання останньої команди
	U	Скасування всіх змін в останньому рядку
	.	Повторення попередньої команди
	;	Повторення попередньої команди пошуку
Команди пошуку тексту	/шаблон	Пошук текстового шаблону після курсору
	?шаблон	Пошук текстового шаблону перед курсором
	n	Продовження пошуку у поточному напрямку
	Shift–n	Продовження пошуку у протилежному напрямку
	При пошуку у шаблонах можуть використовуватися метасимволи: . – означає будь-який одинарний символ; * – означає нуль або більше повторень попереднього символу; […] – означає будь-який символ із перелічених у дужках; [..–..] – означає будь-який символ у визначеному діапазоні	
Команди заміни тексту	cw	Заміна слова після курсору
	cb	Заміна слова перед курсором
	cc	Заміна цілого рядка
	c$, C	Заміна рядка, починаючи із позиції курсору до кінця рядка
	rx	Заміна символу на x
	s	Пошук і заміна тексту Синтаксис: : перший_рядок, останній_рядок s/старий_текст/новий_текст У визначенні діапазонів можна використовувати шаблони та знаки підстановки: . – поточний рядок; $ – останній рядок; % – весь файл

Слід зазначити, що настанови, змінені командою set, перетворення, створені командою map, і зміни, виконані командою abbr, активні тільки впродовж сеансу. Для активізації їх при кожному сеансі роботи необхідно включати зміни у файл .exrc (має бути розташований в домашньому каталозі

користувача), який містить команди, що виконуються при кожному запуску сеансу vi.

Редактор vi дозволяє одночасно редагувати кілька файлів. Для цього слід визначити всі файли, що редагуватимуться у командному рядку: vi файл_1 файл_2 файл_3 і т.д. Перший указаний файл з'явиться на екрані. Для переходу між файлами слід ввести у командному режимі :n (перехід до наступного файлу) та :rew (повернутися до першого файлу).

Текстовий редактор ed

До появи екранно-орієнтованих редакторів існували рядково-орієнтовані редактори й серед них такий, як ed. Він вимагає набагато менше пам'яті та не використовує додаткові файли й каталоги. Ed менш зручний, ніж vi, але він може бути корисним у скриптах або на специфічних вбудованих системах.

Для завантаження редактора ed необхідно набрати у командному рядку ed разом з ім'ям файлу, який редагуватиметься (або просто ed).

Основні команди ed подано у табл. 3.2.

Таблиця 3.2

Команда	**Опис**
.	Вихід із режиму введення
a	Додати новий текст після поточного рядка: 0a – додати у початок файлу; $a – додати у кінець файлу; -a –додати перед поточним рядком; #a – додати за рядком номер #
c	Заміна цілого рядка тексту
d	Вилучення поточного рядка (можлива комбінація із числами, напр. 8d (вилучити рядок 8), або з діапазоном, напр. 2,6d (вилучити рядки 2–6))
H	Дозволяє читати повідомлення про помилки
i	Введення тексту перед поточним рядком
m	Пересування тексту. Синтаксис: початкова_адреса, кінцева_адреса m вказаний_рядок Наприклад, 5,7m10
n	Нумерація рядків
p	Друк поточного рядка на екрані (можлива комбінація з числами, напр. 5p (показати рядок 5) або з діапазоном, напр. 3,9p (показати рядки 3–9))
q	Вихід із редактора
r	Вставляє вміст іншого файлу або вивід команди у поточний файл
s/старе/нове/	Заміна старого виразу на новий
t	Копіювання тексту. Синтаксис: Початкова_адреса, кінцева_адреса t вказаний_рядок Наприклад, 5,7t10
u	Скасування останньої команди
w	Запис файлу
wq	Записує файл і виходить з редактора

Приклад створення файлу за допомогою редактора ed.

```
$ ed
a
text for lesson
.
w temp
16
q
```

Контрольні запитання

1. Як відобразити поточний режим роботи редактора vi?
2. Що треба зробити, коли невідомо, в якому режимі працює vi?
3. Як редактор vi здійснює режим редактора ex?
4. Чи чутливі команди vi до регістру?
5. Відомі вам режими роботи vi.
6. Якщо при реєстрації користувача не встановлено значення змінної TERM, як її можна встановити у редакторі vi?
7. Як з'ясувати точне ім'я та тип термінала?
8. Які команди викликають примусове оновлення екрана?
9. Як відображаються команди командного режиму на екрані?
10. В якому режимі зберігаються зміни у файлі?
11. Що означає префікс, який можна вводити для більшості команд vi?
12. Які групи буферів існують у редакторі vi?
13. Як задати діапазон пошуку у вигляді всіх рядків текстового файлу?
14. Як у редакторі vi викликати тимчасово поточну оболонку shell?
15. Як встановити опції редактора vi? Які опції Вам відомі?
16. Як встановити опції за межами редактора vi?
17. Що таке редактор joe?

Завдання

1. Робота із текстовим редактором ed:
   - завантажте редактор ed;
   - наберіть кілька рядків тексту (не менше п'яти);
   - завершіть набір тексту та збережіть файл під назвою f1_ed;
   - завершіть режим роботи із редактором;
   - завантажте файл f2_ed за допомогою текстового редактора ed;
   - виведіть на екран другий рядок файлу;
   - виведіть на екран третій і четвертий рядки файлу;
   - виведіть на екран весь файл;
   - вставте вміст файлу f1_ed у файл f2_ed, перегляньте файл і збережіть його;
   - вставте новий рядок тексту між рядками 4 та 5, перегляньте весь файл;
   - вилучіть рядок 5;
   - вилучіть рядки 2–6;

- замініть рядок 5 на рядок * 12345 *, перегляньте рядок 5;
- перемістіть рядки 1 та 2 у кінець файлу та збережіть його;
- скопіюйте рядки 2 та 3 на місце рядків 5 і 6;
- збережіть файл і вийдіть із редактора.
- підготуйте детальну електронну версію звіту з роботи в редакторі ed.

2. Робота із текстовим редактором vi:
   - Завантажте редактор vi.
   - В режимі введення наберіть текст файлу Imagine (пісня "Imagine" Джона Леннона).

     Imagine there's no heaven
     It's easy if you try
     No hell below us
     Above us only sky
     Imagine all the people living for today
     -
     Imagine there's no countries
     It isn't hard to do
     Nothing to kill or die for
     And no religion too
     Imagine all the people living life in peace, you

     You may say I'm a dreamer
     But I'm not the only one
     I hope some day you'll join us
     And the world will be as one

     Imagine no possessions
     I wonder if you can
     No need for greed or hunger
     Our brotherhood of man
     Imagine all the people sharing all the world, you

     You may say I'm a dreamer
     But I'm not the only one
     I hope some day you'll join us
     And the world will live as one

   - За допомогою vi виконати такі дії:
   - продемонструйте викладачеві знання команд введення, редагування, переміщення тексту в інший файл, копіювання тексту в інший файл тощо;
   - запишіть текст файлу в заданий викладачем буфер;
   - запишіть файл і вийдіть з vi;

- викличте vi із завантаженням файлу Imagine;
- замініть 5 порожніх рядків після куплетів на рядки *******…******;
- перепишіть текст модифікованого файлу в інший заданий викладачем буфер;
- в новому буфері замініть рядки *******…****** на рядки пробілів;
- запишіть відновлений файл Imagine під іменем Imagine2;
- замінити друге входження рядка 'people' на 'PEOPLE';
- замінити друге входження з початку файлу рядка 'all' на 'ALL';
- замінити друге входження з кінця файлу рядка 'all' на 'ALL';
- підготуйте детальну електронну версію звіту з роботи в редакторі vi.

Список літератури

1. http://docs.freebsd.org/44doc/usd/12.vi/paper.html - текстовий редактор VI.
2. https://www.freecodecamp.org/news/how-to-use-the-vim-text-editor-intro-for-devs/ - як використовувати текстовий редактор Vim.
3. https://www.vim.org/ - офіційний сайт Vim website with documentation and downloads.
4. https://vim.rtorr.com/lang/uk - управління в Vim українською мовою.
5. https://github.com/iggredible/Learn-Vim - підручник для вивчення Vim.

Лабораторна робота № 4

Потоковий редактор SED. Процесор мови обробки шаблонів AWK

Мета роботи – одержання навичок під час роботи із потоковим редактором sed; вивчення awk; створення, налагоджування та виконання сценаріїв мовою awk.

Як зазначалося у лабораторній роботі 2 "Файлова система та основні команди UNIX" в ОС UNIX існує велика кількість програм, які читають вхідний потік, виконують прості операції над ним і записують дані у вихідний потік, тобто фільтри. До фільтрів належать відомі програми grep, tail, head, sort, wc, cut тощо. Але в ОС UNIX користувач має можливість не тільки застосовувати стандартні фільтри, а й створювати нові за допомогою перетворювачів даних загального призначення (їх ще називають фільтрами, що програмуються). До цих програм належать потоковий редактор sed (stream editor) і процесор мови пошуку та обробки шаблонів awk (ім'я складається із перших літер імен авторів).

Команда wc підраховує кількість рядків, символів, чи байт у вказаному файлі, а також їх суму, якщо вказано більше одного файлу.

Опції команди wc:

l – виводить кількість рядків;

c – виводить кількість байт;

m – виводить кількість символів;

w – виводить кількість слів.

Наприклад,

```
wc file1 file2
```

Команда cut – це команда, яка дозволяє друкувати лише певну частину рядків з файлу. Деякі опції команди cut:

b – вибрати лише задані байти;

c – вибрати лише задані символи;

f – друкує лише поля задані в списку;

s – не друкує рядки, які не містять роздільників.

Наприклад, для виведення з другого по восьмий символ кожного рядка файлу file1 при цьому вивести всі рядки починаючи з шостого, використовується команда:

```
cut -c 2-8
cut -f 6- file1
```

SED – це потоковий редактор, отже команди, що маніпулюють даними, необхідно вводити із файлу або у командному рядку. Спочатку був розроблений для ОС UNIX в 1973 році, зараз sed доступний фактично для всіх операційних систем, які підтримують роботу з командним рядком. Sed отримує вхідний потік (найчастіше файл) порядково, редагує кожний рядок окремо відповідно до заданого правила, який визначений в sed-скрипті та виводить результат на екран.

Використання sed доцільно для:

- редагування файлів великих розмірів, коли незручно редагувати в інтерактивному режимі;
- редагування файлу будь-якого розміру, коли відредагований текст громіздкий для виведення в інтерактивному режимі;
- ефективного виконання багатьох глобальних функцій редагування за один прохід через введення.

Синтаксис для команд редактора sed:

```
sed [опції] 'команди' ім'я_файлу(ів)
```

Основні опції командного рядка потокового редактора sed:

f – файл-команди редактора sed читаються зі вказаного файлу;

n – подавляється передбачений за замовчанням режим виведення даних (за замовчанням редактор sed друкує всі рядки, як модифіковані, так і не модифіковані. Тому деякі рядки друкуються двічі. Якщо ж використовується опція –n, то виводяться тільки модифіковані рядки).

Поле команди використовує наступний формат:

```
[адреса [, адреса ]] функція [аргументи]
```

Адресні компоненти визначають номер рядка, діапазон рядків або регулярний вираз (див. лабораторну роботу 2, табл. 2.4, крім символу \, що використовується у багаторядкових командах для екранування символу нового рядка), до яких буде застосовано функцію (команду), а аргумент – це текст, який потрібно знайти. Основні команди sed зведено у табл. 4.1.

Як вже зазначалося, команди редактора sed можуть бути введені як у командному рядку, так і з файлу. У командному рядку частіше розміщуються малі сценарії, при цьому команди вміщуються в апострофи.

Наприклад,

```
sed '$d' file1
```

команда здійснює вилучення останнього рядка із файлу file1.

Команди, що надходять або подаються із командного рядка, можуть бути складними. Коли сценарії складні, доцільно використовувати файл, до якого буде внесено список команд редактора sed.

Наприклад, розглянемо роботу команди sed використовуючи команд з файлу

```
cat file1
s/Hi/Hello/g
s/text/TEXT/g
5a\
Welcome to UNIX\
7r /tmp/file2

sed –f file1 sample
```

Сценарії редактора sed можуть використовуватися у конвеєрі з іншими командами ОС UNIX. Наприклад,

```
who | sed '2a\
!новий рядок!
```

```
b 1b
2d
:1b 3d'
```

Таблиця 4.1

Команда	Аргумент	Опис
a	текст	Додає вказаний текст після
B	мітка	Переходить на команду, позначену міткою
C	текст	Замінює текст, вилучає шаблон і вставляє текст
d		Вилучає область шаблонів
D		Вилучає область шаблонів до першого нового рядка
g		Заміщає область шаблонів на вміст області зберігання
G		До області шаблонів додає вміст області зберігання
h		Заміщує область зберігання на вміст області шаблонів
H		До області зберігання додає вміст області шаблонів
i	текст	Вставляє текст у стандартний вивід
I		Роздруковує вміст області шаблонів у стандартному виводі та незображені символи у системі кодів ASCII
n		Копіює область шаблонів у стандартний вивід і заміщує її наступним рядком вводу
N		Додає наступний рядок вводу до області шаблонів із вбудованим новим рядком
p		Направляє область шаблонів на стандартний вивід
P		Друкує початковий сегмент області шаблонів у першому новому рядку стандартного виводу
q		Здійснює вихід із редактора (перехід у кінець сценарію)
r	файл	Читає вміст файлу та вміщує його у вивід перед наступним рядком вводу
s		Аргумент у вигляді: /пошуковий_шаблон/заміна/ознака замінює рядки шаблона пошуку, який може бути регулярним виразом. Ознаки можуть складатися з елементів: n = 1 – 400 – заміщує тільки n-е входження шаблона; g – глобально заміщує всі примірники входжень регулярного виразу, якщо вони не перекриваються; p – друкує область шаблона, якщо заміщення виконано; wfile – записує область шаблона у файл, якщо заміщення виконано
t	мітка	Здійснює перехід на вказану мітку, якщо виконано будь-яку підстановку
w	файл	Записує область шаблону у вказаний файл
X		Проводить обмін вмісту області шаблону та області зберігання
Y	/рядок1/	Заміщує символ у рядку 1 на щодо символу у рядку 2.

Команда	Аргумент	Опис
	рядок2	Довжини рядків мають бути однаковими
!команда		Виконує команду для рядка(ів), що не вибрано, за адресами
:	мітка	Призначає мітку для команд b, t
=		Друкує номер рядка у стандартному виводі у вигляді окремого рядка
{...}		Виконує команду як групу команд тільки тоді, коли область шаблона збіглася

Процесор мови обробки шаблонів awk за принципом роботи певною мірою нагадує редактор sed. За допомогою програмування на awk можна створювати невеликі програми (сценарії, скрипти), які читають вхідні дані, обробляють їх, виконують потрібні дії (у т. ч. математичні), генерують звіти. Багато з можливостей awk взято із мови програмування С:

- дії при виконанні певних умов;
- організація циклів;
- числові змінні;
- рядкові змінні;
- регулярні вирази;
- команда printf.

Синтаксис для awk:

```
awk [опції] 'команди' ім'я_файлу(ів)
```

Існує три опції командного рядка процесора мови обробки шаблонів awk:

Fc – визначає альтернативний роздільник поля з використанням символу с;

f-файл – читає файл з метою отримання програмних інструкцій для виконання;

- – читає стандартний ввід для обробки файлів.

Структура програми awk має такий вигляд:

```
шаблон {дія}
```

У часткових випадках одна із конструкцій може бути відсутня: {дія}, коли дії виконуються для усіх рядків; шаблон, коли виводяться рядки згідно із даним шаблоном.

Дії можуть складатися із послідовності операторів, відокремлених один від одного ; – символом нового рядка, або дужкою, що закривається.

Команди процесора мови обробки шаблонів awk, як і редактора sed, можуть бути введені як у командному рядку, так і у файлі.

Оскільки мова awk обробляє кожний вхідний рядок, то він має справу із записами та полями. Запис – це індивідуальний рядок вводу. Поля – це запис, поділений роздільниками. Роздільниками полів за замовчанням є пропуск і символ табуляції, а записів – символ нового рядка.

Змінні awk

У мові програмування awk виділяють дві групи змінних: декларовані та перевизначені. Декларованим користувач сам дає початкове значення та

може вставити їх у формулу, там де вони потрібні. Значення ж перевизначених змінних задає інтерпретатор awk. У мові програмування awk також є можливість використання одномірних масивів. Масив не оголошується, а починає існувати у момент першого використання. Індексом масиву може бути як число (крім 0), так і рядок.

У табл. 4.2 наведено деякі з перевизначених змінних, які підтримує awk.

Операції та функції мови програмування awk

У програмах awk можна виконувати математичні дії. У табл. 4.3–4.5 наведено основні змінні, операції та арифметичні функції, які можна використовувати у програмах awk.

Таблиця 4.2

Змінна	Значення	За замовчанням
ARGC	Кількість аргументів командного рядка	
ARGV	Масив аргументів командного рядка	
FILENAME	Ім'я поточного введеного файлу	
FNR	Номер запису у поточному файлі	
FS	Роздільник у введеному файлі	Пропуск і/чи табуляція
NF	Число полів у поточному запису	
NR	Число прочитаних на даний момент записів	
OFMT	Вивідний формат для цифр	%6g
OFS	Роздільник поля вивідного файлу	Пропуск
ORS	Роздільник запису вивідного файлу	Символ нового рядка
RS	Роздільник запису введеного файлу	Символ нового рядка
RSTART	Індекс першого вибраного символу за допомогою match( )	
RLENGTH	Довжина рядка, вибраного за допомогою match( )	
SUBSTEP	Нижній роздільник	"\034"

Таблиця 4.3

Операція	Виконувана дія
= += -= *= /= %= ^=	Присвоєння
?<дія1>:<дія2>	Умовний вираз
||	Логічне OR
&&	Логічне AND
~	Пошук регулярного виразу
!~	Виключення регулярного виразу
< > = != ==	Відношення
+ -	Додавання, вирахування
* /	Множення, ділення
++ --	Інкремент, декремент
$	Поле

Таблиця 4.4

Функція	Значення, що повертається
atan2(y,x)	Арктангенс y/x у межах від $-\pi$ до π
cos(x)	Косинус x
exp(x)	Експоненціальна функція x
int(x)	Ціла частина x
log(x)	Натуральний логарифм x
rand(x)	Випадкове число між 0 та 1
sin(x)	Синус x
sqrt(x)	Квадратний корінь з x
srand(x)	x – нове початкове значення для rand()

Таблиця 4.5

Функція	Значення, що повертається
gsub(r, s)	Глобальна заміна s на r у поточному запису; повертає кількість замінених символів
gsub(r, s, t)	Глобальна заміна s на r у рядку t; повертає кількість замінених символів
index(s, t)	Повертає позицію t в s: 0 – якщо t немає в s
length(s)	Повертає довжину s
match(s, r)	Повертає позицію s, в якій зустрічається r: 0 – якщо r немає
split(s, a)	Відокремлює s на масив a за FS; повертає кількість полів
split(s, a, r)	Відокремлює s на масив a за r; повертає кількість полів
sub(r, s)	Заміщує s на перше r у поточному запису, повертає кількість замін
sub(r, s, t)	Заміщує s на перше r у рядку t, повертає кількість замін
substr(s, p)	Повертає індекс s, починаючи із позиції p
substr(s, p, n)	Повертає підрядок s довжиною n, починаючи із позиції p

Перевизначені шаблони та оператори контролю даних

BEGIN – це перевизначений шаблон, дії якого виконуються, коли програма awk починає працювати.

END – це перевизначений шаблон, дії якого виконуються перед завершенням програми awk.

Використовуючи ці конструкції, користувач може встановити спеціальні змінні чи заголовки, які друкуватимуться перед початком обробки та по її завершенні.

У процесорі мови обробки шаблонів awk існує можливість контролю даних, для чого введено такі оператори, як умовний оператор if – else, цикли while та for.

Синтаксис цих конструкцій збіжний з аналогічними конструкціями у мові програмування С.

Вихід із циклу можна здійснити за допомогою таких операторів як:

break – змушує процесор awk вийти за межі циклу та почати обробку з першої команди за межами циклу;

continue – припиняє роботу ітерації циклу та повертається на його початок;

exit – змушує програму awk здійснити перехід на оператор END і продовжувати обробку. Якщо оператор END відсутній чи у ньому знаходиться команда exit, то сценарій миттєво припиняє свою роботу.

Виведення у програмах awk

Найпростішою формою для програм awk є виведення на екран кожного вхідного рядка. Це здійснюється за допомогою команди print.

Наприклад, awk '{print}' users виводить на екран користувачів, що є на цей момент у системі, awk '{print""}' file1 – порожній рядок.

Іноді трапляються випадки, коли простого виведення недостатньо. Тоді використовується команда print. Синтаксис цієї команди подібний до аналогічної команди у мові програмування С, аналогічні й особливості формату.

При виведенні у програмах awk є можливість використання конвеєра та перенаправлення виведення.

Наприклад,

```
awk '{printf ("%d\t%s\n", $1)>"/tmp/file2"}'street
```

перенаправляє стандартний вивід у файл file2;

```
awk'{printf $1 | "sort"}'street.
```

Приклад демонструє використання конвеєра.

Обмеження awk

При роботі з awk необхідно дотримуватися таких обмежень (зокрема, для сумісності з попередніми версіями) (табл. 4.6):

Таблиця 4.6

Максимальне значення	Тип обмеження	Максимальне значення	Тип обмеження
100	Полів	1024	Символів у printf
2500	Символів у ввідному запису	400	Символів у рядку, взятому у лапки
2500	Символів у вивідному запису	15	Відкритих файлів
1024	Символів у індивідуальному полі	1	Конвеєр

Оскільки кількість відкритих файлів і конвеєрів обмежена, то в awk існує механізм закриття файлів і конвеєрів за допомогою команди close.

Наприклад,

```
awk '{printf $1 > "/tmp/file"}' file1
…
close ("tmp/file")
#закриття файлу ;
awk '{printf $1 | "sort"}' file1
…
close ("sort")
#закриття конвеєра.
```

Приклад сценарію мовою awk

```
# Print frequency histogram of column of numbers
BEGIN {na=0; nb=0; nc=0; nd=0; ne=0; nf=0; ng=0; nh=0; ni=0; nj=0}
$2 <= 0.1 {n=n+1}
($2 > 0.1) && ($2 <= 0.2) {nb = nb+1}
($2 > 0.2) && ($2 <= 0.3) {nc = nc+1}
($2 > 0.3) && ($2 <= 0.4) {nd = nd+1}
($2 > 0.4) && ($2 <= 0.5) {ne = ne+1}
($2 > 0.5) && ($2 <= 0.6) {nf = nf+1}
($2 > 0.6) && ($2 <= 0.7) {ng = ng+1}
($2 > 0.7) && ($2 <= 0.8) {nh = nh+1}
($2 > 0.8) && ($2 <= 0.9) {ni = ni+1}
($2 > 0.9) {nj = nj+1}
END {printf na, nb, nc, nd, ne, nf, ng, nh, ni, nj, NR}
```

Регулярні вирази у мові AWK

Для здійснення пошуку в мові AWK допускається використання регулярних виразів, вкладених у / /. В мові AWK дозволяється використання у регулярних виразах таких конструкцій:

- **()**– дужки допускаються для групування;
- | – вказівка альтернативи "або";
- + – плюс, що стоїть за регулярним виразом означає будь-яку послідовність входжень цього виразу, починаючи з 1;
- **?** – Знак питання за регулярним виразом означає 0 або 1 входжень цього виразу;
- **[AZ]** – допускається скорочена форма запису для діапазонів ASCII символів;
- Встановлений порядок виконання операторів на одному дужковому рівні: "[]*+? конкатенація | ".

Наприклад:

/Olga/ – Вказує на рядки, що містять Olga.

/[Oo]lga|[Mm]ike|[Mm]al/– вказує на рядки, що містять Olga або olga або Mike або mike або Mal або mal.

/number[0-9]/ – Вказує на рядки, що містять number0 або number1 або ... number9.

/\.+\/ – Вказує на рядки, що містять будь-яку кількість символів, більшу або рівну 1, вкладених в / /.

Контрольні запитання

1. Як працює потоковий редактор sed? Синтаксис його виклику.
2. Назвіть основні символи створення шаблонів у sed, поясніть їх призначення.
3. Як здійснюється при роботі із sed перехід на мітку? Що відбувається за відсутності мітки?
4. Як у sed здійснюється читання файлу?
5. Які ознаки існують у команді sed s, що вони визначають?
6. Як відбувається у sed групування команд?
7. Як сформувати у командному рядку сукупність команд редактора sed?
8. Для чого в awk використовуються шаблони BEGIN та END?
9. Яку структуру може мати контекст команди у виклику awk?
10. Відомі вам напередвизначені змінні в awk, їх призначення.
11. Як модифікувати значення змінної awk?
12. Що таке програмний файл і сценарій awk? У чому їх різниця?
13. Як в awk визначаються змінні користувача?
14. Відомі вам оператори мови awk?
15. Як відбувається форматування виведення інформації в awk?
16. Як формуються масиви в мові awk? Вимоги, що висуваються до них.
17. Як в awk визначити функцію користувача?
18. Поясніть різницю між опцією c та m для команди wc? Коли ці опції дадуть різний результат?
19. Покажіть один з прикладів в якому команди sed та cut дадуть однаковий результат?

Завдання

Розділ 1. Обов'язкові завдання з потокового редактора SED (для всіх студентів)

1. Створіть текстовий файл з 12 рядками різного змісту, який буде використовуватися для виконання всіх операцій SED. Файл повинен містити:
   - Рядки з повторюваними словами та фразами.
   - Числові дані.
   - Спеціальні символи.
   - Рядки різної довжини.
2. За допомогою sed виконати такі дії (кожна дія реалізується окремим викликом sed):
   - Після перших 6 рядків додати порожній рядок після кожного.
   - Після останніх 6 рядків додати два порожніх рядки після кожного.

- Вилучити всі порожні рядки (повернутися до первісного файлу).
- Замінити 1-е і 3-є входження певного шаблона в кожному рядку.
- Замінити певний шаблон у всьому файлі.
- Замінити всі входження шаблона в рядку, починаючи з k-го.

3. За допомогою sed (для одержаного після завершеного п. 2. файлу) виконати такі дії:
   - Замінити всі входження шаблона в рядку, починаючи з k-го до n-го.
   - Вилучити рядки 3 і 5 у файлі.
   - Виконати нумерацію рядків.
   - Переглянути файл з рядка m до k.
   - Переглянути файл з k рядка до рядка, що містить заданий шаблон.
   - Вивести всі рядки, що містять шаблон, в тому числі наступні за кожним із них k рядків.
4. Виконати послідовно (за 1 крок) п. 2 та п. 3
   - Створіть два командних пакетних файли SED:
     - Файл script1.sed - містить команди з пункту 2.
     - Файл script2.sed - містить команди з пункту 3.
   - Виконайте послідовно: `sed -f script1.sed filename | sed -f script2.sed`.
5. Порівняти результати виконання
   - Проаналізуйте відмінності між покроковим виконанням (п. 2-3) та пакетним виконанням (п. 4).

Розділ 2. Індивідуальні завдання з процесора AWK (варіанти 1-45) Кожен студент виконує завдання згідно зі своїм варіантом, отриманим у викладача. Всі завдання передбачають створення файлів даних та AWK сценаріїв.

1. За допомогою vi створіть файл наступної структури (не менше 15 рядків)

Прізвище	Рік народження	Посада	Домашній телефон	Заробітна плата

   та виконайте:
   - Виведіть прізвище, посаду й дані про заробітну плату.
   - Виведіть прізвище та посаду робітників, яким більше 40 років.
   - Виведіть прізвище та домашні телефони робітників, чиє прізвище починається на літеру А.
   - Додайте у кінець таблиці кілька робітників.
   - Вилучіть четвертий рядок.

   При виведенні таблиці мають мати відповідні заголовки.
2. Напишіть сценарій, що запитує введення курсу долара, карбованця та євро щодо гривні, а потім друкує у вигляді таблиці достоїнство кожної купюри у гривнях (1, 2, 5, 10, 20, 50, 100, 200 грн) для відповідної валюти.

3. Напишіть сценарій awk, що читає із файлу числа (градуси), розраховує і виводить на екран таблицю з колонками: Градуси, Радіани, Sin, Cos, Tg. У першому рядку файлу має бути відповідний заголовок.

Градуси	Радіани	Sin	Cos	Tg

4. Напишіть сценарій, що вилучає дублююче слово із файлу.
5. Напишіть сценарій, що заносить кожний вхідний рядок до елемента масиву, що індексується номером рядка, а потім друкує їх з останнього рядка по перший.
6. Напишіть сценарій, що підраховує частоту появи слова у вхідному потоці.
7. Напишіть сценарій, який формує новий файл, де всі стовпчики сформовано в інверсному порядку (перший стає останнім, другий передостаннім і т. д.).
8. Напишіть сценарій, що із файлу (кожен із трьох стовпчиків якого містить числа) друкує число зі стовпчика 3, якщо число у першому стовпчику більше, ніж у другому і 0 – у протилежному випадку.
9. Є вхідний файл, що містить два стовпчики із числами. Напишіть сценарій awk, який формує новий файл із трьох стовпчиків: якщо число у першому стовпчику більше, ніж у другому, то у третьому формується їх сума, інакше – різниця.
10. Виведіть вміст текстового файлу у зворотному порядку.
11. Знайдіть рядок з максимальною кількістю символів в текстовому файлі.
12. Знайдіть у текстовому файлі рядки, які містять певне слово або фразу.
13. Обчисліть кількість рядків у текстовому файлі та виведіть результат на екран.
14. Є файл, що містить два стовпчики із числами. Напишіть сценарій awk, який знаходить максимальне та мінімальне значення у кожному стовпчику.
15. Є файл, що містить стовпчик чисел. Напишіть сценарій awk, який підраховує кількість чисел за діапазонами: менші 0; 0-10; 10-20; ...; 90-100; більші 100.
16. Напишіть сценарій awk, що створює файл з трьома колонками: номер (1-100), exp(номер), ln(номер). Додайте відповідний заголовок.
17. Обчисліть суму всіх чисел у файлі та їх середнє значення.
18. Обчисліть кількість входжень кожного слова в текстовому файлі та виведіть результат.
19. Витягніть перші літери кожного слова в рядку та надрукуйте їх у вигляді абревіатури (великими літерами).
20. Напишіть сценарій awk, що створює файл з випадковими числами (50 записів) та обчислює квадрат і куб кожного числа. Додайте заголовки колонок.
21. Напишіть сценарій awk, що для введеної дати визначає, якому дню тижня вона відповідає.
22. Створіть словник (40+ слів) формату <англійське_слово> – <переклад слова українською мовою (німецькою, французькою, польською тощо>. Напишіть сценарій для перекладу речень з файлу з підрахунком статистики перекладу.
23. Відсортуйте файл в алфавітному порядку по першій літері в рядку.

24. Відсортуйте файл на основі певного стовпця із числами в порядку спадання.
25. Виявлення та видалення рядків, що містять не-ASCII символи в файлі.
26. Перетворення файлу, що містить римські числа в десяткові.
27. Виявлення та видалення рядків у файлі, що містять конфіденційну інформацію (наприклад, номери кредитних карток).
28. Напишіть сценарій, що виконує пошук ключових слів в тексті та підраховує частоту їхнього входження.
29. Реалізуйте сценарій, що підраховує загальну кількість слів у файлі та виводить найчастіше та найрідше вживані слова.
30. Напишіть сценарій awk, що формує таблицю сигналів вашої версії UNIX (Сигнал, Назва, Функція).
31. Напишіть сценарій awk, що друкує список груп користувачів вашої версії UNIX і підраховує їх кількість.
32. Напишіть сценарій awk, що аналізує лог-файл веб-сервера та підраховує кількість запитів від кожної IP-адреси, виводячи топ-10 найактивніших адрес.
33. Створіть сценарій awk, що читає файл з координатами точок (x, y) та обчислює відстань між послідовними точками, виводячи загальну довжину шляху.
34. Напишіть сценарій awk, що обробляє файл продажів (дата, товар, кількість, ціна) та генерує щомісячний звіт з прибутками по кожному товару.
35. Створіть сценарій awk, що симулює роботу банкомату: читає операції (депозит/зняття) з файлу та відстежує баланс, сигналізуючи про недостатність коштів.
36. Напишіть сценарій awk, що аналізує файл з температурними даними та визначає найтепліший/найхолодніший день, середню температуру по місяцях.
37. Створіть сценарій awk, що реалізує простий калькулятор матриць: читає дві матриці з файлів та виконує їх додавання або множення.
38. Напишіть сценарій awk, що обробляє файл студентських оцінок та генерує академічну статистику: середній бал, відсоток відмінників, список боржників.
39. Створіть сценарій awk, що аналізує файл мережевого трафіку (час, розмір пакета, протокол) та будує графік завантаження мережі по годинах.
40. Напишіть сценарій awk, що реалізує простий генератор паролів з заданою довжиною та складністю (цифри, літери, спецсимволи).
41. Створіть сценарій awk, що обробляє файл бібліотечного каталогу (автор, назва, рік, жанр) та генерує звіти по авторах, жанрах, роках видання.
42. Напишіть сценарій awk, що симулює роботу простої бази даних: виконує операції SELECT, INSERT, DELETE над табличними даними у файлі.
43. Створіть сценарій awk, що аналізує файл з результатами спортивних змагань та обчислює рейтинг команд за очковою системою.

44. Напишіть сценарій awk, що обробляє файл з інформацією про співробітників (ім'я, відділ, зарплата, стаж) та генерує HR-звітність: середня зарплата по відділах, стаж роботи.
45. Створіть сценарій awk, що реалізує простий аналізатор коду: підраховує рядки коду, коментарі, порожні рядки в програмних файлах та обчислює метрики складності.

Список літератури

1. Robbins A., Beebe N.H.F. Classic Shell Scripting: Hidden Commands that Unlock the Power of Unix. O'Reilly Media, 2005. – написання сценаріїв для Unix.
2. https://www.gnu.org/software/sed/manual/ - офіційна документація SED.
3. https://www.gnu.org/software/gawk/manual/ - офіційна документація AWK.

Лабораторна робота № 5

Командний процесор Shell

Метою лабораторної роботи є одержання навичок під час роботи із командним процесором bash, створення, налагоджування та виконання shell-сценаріїв.

Оболонка shell – це інтерпретатор команд, який забезпечує інтерфейс у вигляді командного рядка між користувачем і ядром ОС UNIX за допомогою виконання команд, введених із термінала. Користувач вводить у командному рядку команди, shell інтерпретує їх і надсилає у вигляді інструкцій ОС для виконання. Shell не тільки інтерпретує команди, але й створює середовище для програмування. Фактично командний процесор shell є також мовою програмування.

У реалізаціях ОС UNIX є кілька версій shell: Bourne Shell, C Shell, Korn Shell, Bourne Again Shell, TC Shell, POSIX Shell тощо. Основні особливості цих оболонок:

Bourne Shell (sh) – стандартний Shell ОС UNIX, є у всіх реалізаціях ОС UNIX. Найкращий із погляду сумісності сценаріїв, розроблених для різних версій UNIX;

C Shell (csh) – розроблений у Каліфорнійському університеті у Берклі. Має С-подібний синтаксис та інтерактивну обробку команд;

Korn Shell (ksh) – розроблений співробітником AT&T Bell Labs Девідом Корном. Створено додаткові засоби програмування;

Bourne Again Shell (bash) – включає кращі можливості C Shell і Korn Shell. Має зручний інтерфейс введення команд;

TC Shell (tcsh) – поліпшений варіант C Shell. Є можливість редагування командного рядка та доповнення імен файлів;

POSIX Shell (sh) – поліпшений варіант Korn Shell. Із 1992 р. постачається разом з ОС HP-UX. Відповідає стандарту POSIX.

Далі буде розглянуто Bourne Again Shell (bash).

Структура команд shell

Команди shell зазвичай мають такий формат:

```
<команда> <опції> <аргументи>
```

<опції> – це літерний код, що модифікує тип дії, яку виконує команда. Ознакою опції є символ –.

<аргументи> – визначають об'єкт, над яким виконуватиметься дія. Наприклад,

```
ls -l /usr/bin
```

Це не обов'язкова структура, іноді опції та/або аргументи можуть бути відсутні.

Наприклад, якщо користувач знаходиться у каталозі, уміст якого потрібно переглянути, він вводить команду ls.

Як зазначалося у попередніх лабораторних роботах, команди можуть використовуватися за конвеєрної обробки

```
команда1|команда2|...
```

Наприклад,

```
ls|sort
```

Також команди можуть включатися до списку. Список – це послідовність з кількох команд або конвеєрних ланцюжків.

```
команда1; команда2; команда 3 ...
команда1|команда2; команда3;| команда4 ...
```

Наприклад,

```
ls –l; wc
```

Списки також можуть використовуватися, коли виконання команди має залежати від статусу завершення попередньої команди. Для цього використовуються логічні оператори:

команда1 && команда2 – команда2 виконується у тому випадку, якщо статус завершення команди1 дорівнює 0.

команда1 || команда2 – команда2 виконується у тому випадку, якщо статус завершення команди1 не дорівнює 0. Наприклад,

```
mkdir dir1 && cd dir1
grep usr1 /etc/passwd || echo "no one"
```

Псевдоніми (аліаси)

Іноді користувачу потрібно застосовувати команди, які мають довгий і незручний вигляд. Для цього випадку в ОС UNIX існує механізм псевдонімів. Для створення псевдонімів (додаткових імен команд) використовується команда alias, яка працює як макрос, що перетворює псевдонім на команду, ім'я якої йому присвоєно. Псевдонім не змінює ім'я команди, а дає команді ще одне. Синтаксис команди:

```
alias ім'я=команда
```

Команду можна використовувати разом з опціями. У цьому випадку опції беруться в апострофи. Наприклад,

```
alias go='cd /usr/ports'
```

Команда alias без аргументів видає список усіх псевдонімів, що існують на цей момент у системі. Знищити псевдоніми можна за допомогою команди unalias. Наприклад,

```
unalias go
```

Функції

Функція сценарію – це іменована послідовність команд із синтаксисом:

```
ім'я( ) {список;}
```

Функції являють другий механізм створення псевдонімів. Проте на відміну від alias функціям можна передавати аргументи, і вони виконують дії. Функції також можуть взаємодіяти між собою або використовувати спільні дані. У тілі функції доступ до аргументів здійснюється за допомогою конструкції $змінна.

Змінні оточення

У shell користувач має можливість створювати змінні та присвоювати їм значення. Ім'ям змінної може бути послідовність літер, цифр і символів

підкреслення, що починаються з літери або символу підкреслення. Ім'я змінної не може збігатися з назвами команд shell і зарезервованими змінними. У табл. 5.1 наведено деякі з цих змінних.

Таблиця 5.1

Змінна	Визначення
PWD	Визначає поточний каталог
HOME	Визначає вхідний каталог користувача. Встановлюється відповідно до інформації із файлу etc/passwd. Команда cd використовує її значення для повернення у початковий каталог, коли не вказано аргумент
PATH	Визначає шлях, що використовує shell для знаходження команд
LOGNAME	Визначає вхідне ім'я користувача. Встановлюється при вході користувача у систему
MAIL	Визначає місце розташування поштової скриньки користувача
TERM	Містить назву типу використовуваного термінала. Встановлюється при вході користувача у систему
SHLVL	Збільшує значення на одиницю за кожного запуску примірника bash
EDITOR	Визначає редактор, що використовуватиметься для корегування списку протоколу команд

За допомогою команди env можна отримати весь список зарезервованих змінних.

Присвоєння значення змінним має такий вигляд:

```
<змінна>=<значення>
```

У shell існує два типи даних: рядок символів і текстовий файл. Тому, якщо значення має пропуски, у команді його потрібно взяти у подвійні лапки.

Для отримання значення змінної потрібно безпосередньо перед її ім'ям поставити знак $. Значенням ще невизначеної змінної є порожній рядок.

У bash, на відміну від Bourne Shell, передбачено можливість роботи із масивами. Значення масивів можуть задаватися у два способи. Перший спосіб – це присвоєння значення одному елементу:

```
ім'я[індекс]=значення
```

Другий спосіб – це присвоєння значення одразу кільком елементам масиву:

```
ім'я=(значення_1,..., значення_N) або
ім'я=([індекс]=значення_1,..., [індекс]=значення_N)
```

Індексом масиву не обов'язково має бути число, ним може бути й рядкове ім'я.

Для звернення до елемента масиву потрібно використати таку конструкцію:

```
${ім'я[індекс]}
```

Арифметичні операції

bash дозволяє виконувати прості арифметичні обчислення, для чого використовується команда let. Як аргумент вона сприймає арифметичний

вираз (якщо аргументи розділені пропусками, то вираз потрібно брати в лапки).

Наприклад, shell надає можливість включати у вираз присвоєння.

```
let "res=3*5+7"
echo $res
22
```

Керуючі структури

У bash існує два типи керуючих структур: умовні оператори та оператори циклів.

Виконання перевірок умов виконуються за допомогою команди

```
test умова або конструкції
[умова]
```

Якщо умова виконується, то команда перевірки повертає 0, у протилежному випадку – 1. У табл. 5.2 наведено значення опцій для перевірки об'єктів.

Умовні оператори

Для керування потоком команд у shell існують такі умовні оператори:

Оператор if – fi

Синтаксис:

```
if умова1 then
команда(и)
elif умова2 then
команда(и)
else
команда(и)
fi
```

Тут оператор elif є скороченим варіантом операторів else та if і може використовуватися наряду з if, тобто допускаються вкладені структури. Наявність операторів elif та else є необов'язковою.

Оператор case – esac

Синтаксис:

```
case зразок in
шаблон1)
        команда(и) ;;
шаблон2)
        команда(и) ;;
...
шаблонN)
        команда(и) ;;
esac
```

Таблиця 5.2

Опція	Об'єкт	Визначення
d	файл	Повертає значення true, якщо вказаний файл існує та є каталогом
e	файл	Повертає значення true, якщо вказаний файл існує

Опція	Об'єкт	Визначення
f	файл	Повертає значення true, якщо вказаний файл існує та є звичайним файлом
L	файл	Повертає значення true, якщо вказаний файл існує та є символічним посиланням
r	файл	Повертає значення true, якщо вказаний файл існує та дозволений для читання
s	файл	Повертає значення true, якщо вказаний файл існує та має ненульовий розмір
w	файл	Повертає значення true, якщо вказаний файл існує та дозволений для запису
x	файл	Повертає значення true, якщо вказаний файл існує та є виконуваним
o	файл	Повертає значення true, якщо вказаний файл існує та належить даному користувачу
z	рядок	Повертає значення true, якщо вказаний рядок має нульову довжину
n	рядок	Повертає значення true, якщо вказаний рядок має ненульову довжину
=	рядок1 = рядок2	Повертає значення true, якщо рядок1 і рядок2 збігаються
!=	рядок1 != рядок2	Повертає значення true, якщо рядок1 і рядок2 не збігаються
a	вираз1 –a вираз2	Повертає значення true, якщо результатом логічної операції AND над указаними виразами є true
o	вираз1 –o вираз2	Повертає значення true, якщо результатом логічної операції OR над указаними виразами є true
!	вираз	Повертає значення true, якщо вказаний вираз має значення false
eq	вираз1 -eq вираз2	Вираз (число, значення змінної) Повертає значення true, якщо вираз1 дорівнює виразу2
ne	вираз1 -ne вираз2	Повертає значення true, якщо вираз1 не дорівнює виразу2
lt	вираз1 –lt вираз2	Повертає значення true, якщо вираз1 < вираз2
le	вираз1 –le	Повертає значення true, якщо вираз1 менший або дорівнює виразу2

Опція	Об'єкт	Визначення
	вираз2	
gt	вираз1 –gt вираз2	Повертає значення true, якщо вираз1 > вираз2
ge	вираз1 –ge вираз2	Повертає значення true, якщо вираз1 більший або дорівнює виразу2

Зразок – це рядок символів або змінна, що порівнюється із шаблонами, доки не буде виконано критерій порівняння. Якщо такий шаблон є, то команди, належні до нього, запускаються на виконання. Команда ;; передає керування за межі оператора case – esac. Якщо ж зразок не збігається із жодним із шаблонів, то оператор case припиняє роботу. Проте, якщо потрібно виконати якусь дію за замовчанням, то використовується шаблон *, якому відповідає будь-яке значення. В операторі case має бути присутній хоча б один шаблон. Максимальну кількість шаблонів не обмежено. У шаблонах дозволено застосування символів, що використовуються у регулярних виразах.

Оператори циклу

У bash передбачено чотири види циклів:

Оператор for – do – done

Синтаксис:

```
for змінна in список do
команда(и)
done
```

Змінній послідовно присвоюються всі значення зі списку, й для кожного із цих значень виконуються команди. Наприклад,

```
for i in 1 2 3 6 7 10;
do
echo $i;
done
```

Зазвичай цикл for використовується у випадках необхідності багаторазового виконання визначеного набору команд.

Оператор while – do – done

Синтаксис:

```
while список1
do
список2
done
```

На кожній ітерації цього циклу обчислюється список1, і доки повертається значення true, виконується список2. Цикл while використовується за необхідності неодноразового виконання певних дій, доки не виконається умова.

Наприклад,

```
x=1
while [ $x –lt 10 ]
do
echo hello
x=`expr $x + 1`
done
```

Оператор until – do – done

Синтаксис:

```
until список1
do
список2
done
```

Цей цикл є різновидом циклу while. У циклі until список2 виконується, доки список1 повертає значення false. Наприклад,

```
x=1
until [$x –ge 10]
do
echo hello
x=`expr $x +1`
done
```

Оператор select – do – done

Синтаксис:

```
select змінна in список1
do
список2
done
```

Цикл select дозволяє створювати нумероване меню. Пункти списку1 виводяться у нумерованому списку, потім робиться запит, команда приймає дані від користувача (тобто номер пункту) та виконує певні дії.

При роботі із циклами іноді виникає потреба негайно його припинення. Для цього у shell використовується команда break. Параметром цієї команди може бути задане ціле число, що більше або дорівнює одиниці та вказує на число рівнів дії команди. Цей механізм використовується при обробці вкладених циклів.

Введення/виведення інформації

Для виведення у bash використовується команда echo, яка виводить передані їй дані на термінал. Для введення використовується команда read, яка читає рядок зі вхідного потоку та присвоює його змінним, імена яких передано їй як аргументи. На вхід команди можна також передати рядки, які йдуть за символом <<, що розшифровується як тут документ:

```
команда<<рядки
```

У цьому випадку рядки передаються на стандартний вхід команди. Обмеженням рядків, що передаються за замовчанням, є комбінація Ctrl – D.

Виконання shell-сценаріїв

Із командами командного процесора shell можна працювати як в інтерактивному режимі, так і створювати програми зі збереженням їх у файлах.

Для створення файлів із shell-програмою (для виконання програми певною оболонкою) обов'язково потрібно перед написанням програми вставити рядок:

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

За відсутності такого рядка програма виконуватиметься поточною оболонкою.

Для запуску файлу зі сценарієм потрібно визначити цей файл як такий, що виконується (за допомогою команди chmod). Якщо цього не зробити, то виконання shell-програми відбуватиметься за допомогою команди

```
bash ім'я_сценарію
```

Нижче наведено приклад сценарію shell, який роздруковує всі підкаталоги вказаного каталогу.

```
#!/bin/bash
if [ ! -e $1 ]; then
 echo "$1 doesn't exist"
 exit
fi

if [ ! -d $1 ]; then
 echo "$1 is not a directory"
 exit
fi

spwd=`pwd`
cd $1
set `ls –a`
echo "subdirs of `pwd`"
for i
do
 if [ "$i" != "." -a "$i" != ".." -a -d "$i" ]; then
echo $i
fi
done
echo "end of subdirs"
```

Лапки у shell-сценаріях

Різноманітні види лапок використовуються по-різному:

" – подвійні лапки, при використанні таких лапок виконуються всі текстові підстановки. Наприклад для виведення змінної PATH може бути використана команда

```
echo "The value of variable PATH is: $PATH"
```

′ – одинарні лапки, в середині таких лапок текст друкується без підстановок. Наприклад,

```
echo ′$PATH – this is only text′
```

` – зворотні лапки, вони використовуються для підстановки результату вказаної в середині команди. Наприклад,

```
echo "Current time: `date "+%H:%M:%S"`"
```

Контрольні запитання

1. Як формуються та де записуються функції у сценаріях shell?
2. Що означає аргумент у команді break?
3. Як у сценаріях формуються коментарі?
4. Що таке оточення?
5. Як змінні перемістити в оточення? Для чого це потрібно?
6. Екранування символів. Як воно здійснюється? Наведіть приклади.
7. Відомі вам варіанти активізації сценаріїв. Порівняйте їх.
8. Відомі вам спеціальні змінні.
9. Як записуються аргументи у сценаріях?
10. Як виконується команда shift, її призначення?
11. Як здійснюється підстановка змінних у сценаріях?
12. Як здійснюється аналіз опцій, що передаються сценарію? Які для цього існують команди? Наведіть приклади.
13. У чому полягає концепція сигналу, для чого він потрібний?
14. Як відбувається перехоплення сигналів та їх ігнорування?
15. Як відбувається налагодження сценаріїв? Які для цього існують засоби?
16. Як протоколюється виконання сценаріїв?
17. Що таке код завершення сценарію, як він формується?
18. Які підстановки здійснюються shell перед початком інтерпретації?
19. Здійсніть порівняльний аналіз різноманітних оболонок shell.
20. Що таке файл ініціалізації shell, його призначення? Яке він має ім'я? Які файли виконуються при завершенні роботи користувача із системою?
21. Які режими роботи із shell вам відомі, що вони означають?

Завдання

Кожен студент виконує завдання згідно зі своїм варіантом, отриманим у викладача.

1. Написати сценарій переведення 10-ого числа у 16-ву систему числення.
2. Написати сценарій переведення 16-ого числа у 10-ву систему числення.
3. Написати shell-сценарій, який визначає тип указаного файлу та виконує наступні дії: якщо це каталог, то виводить його вміст; якщо це звичайний файл, то за допомогою команди more виводить його вміст на екран; в інших випадках виводить тип файлу.
4. Написати сценарій переведення 16-ого числа у 10-у систему числення і навпаки. Число задається аргументом, напрям переведення задається опцією (d – переведення в 10-ву систему числення; h – переведення в 16-у систему

числення). Примітка. Якщо вводиться число 72, то невідомо, в якій воно системі числення.

5. Написати shell-сценарій, що порівнює файли у двох заданих аргументами каталогах і друкує файли, що відрізняються. Результати друкуються трьома списками: файли, відсутні у першому каталозі, але наявні у другому; файли, відсутні у другому каталозі, але наявні у першому; файли, що існують в обох каталогах, але не збігаються. Передбачити обробку опції –v, яка змушує спочатку друкувати файли, що існують в обох каталогах, але не збігаються.
6. Написати сценарій сортування масиву з 20 чисел за зростанням (метод бульбашки). Початкові значення задаються генератором випадкових чисел. Результат надрукувати.
7. Написати сценарій сортування масиву з 20 чисел за зростанням та спаданням (метод вставок). Початкові значення задаються генератором випадкових чисел. Метод сортування задається опцією (-u – за зростанням; -d – за спаданням).
8. Написати сценарій сортування масиву (метод вставок) з 20 цілих чисел, початкові значення задаються генератором випадкових чисел. Виклик сценарію має містити 3 опції: перша задає ім'я вихідного масиву, друга – відсортованого, а третя – вид сортування (u – за зростанням, d - за спаданням). Вихідний і відсортований масиви надрукувати.
9. Написати сценарій сортування масиву (швидке сортування Хоара) з 20 цілих чисел, початкові значення задаються генератором випадкових чисел. Виклик сценарію має містити 3 опції: перша задає ім'я вихідного масиву, друга – відсортованого, а третя – вид сортування (u – за зростанням, d - за спаданням). Вихідний і відсортований масиви надрукувати.
10. Написати сценарій сортування масиву (швидке сортування Хоара) з 20 цілих чисел, початкові значення задаються генератором випадкових чисел. Виклик сценарію має містити опцію: яка задає вид сортування (u – за зростанням, d - за спаданням). Вихідний і відсортований масиви надрукувати. Сценарій має містити функцію для сортування.
11. Написати сценарій сортування масиву за зростанням трьома методами (метод бульбашки, метод вставок і швидке сортування Хоара) з 20 цілих чисел, початкові значення задаються генератором випадкових чисел. Виклик сценарію має містити 2 опції: перша d задає тип сортування: зростання, спадання; друга – метод сортування. Сценарій має містити три функції (для кожного методу сортування). Створити бібліотеку із цих функцій і викликати їх в сценарії з бібліотеки. Вихідний і відсортований масиви надрукувати.
12. Мовою оболонки bash написати сценарій обчислення ряду чисел Фібоначчі до заданого числа N. Завдання реалізувати у двох варіантах: - рекурсивний сценарій; - звичайний сценарій, в якому викликається рекурсивна функція обчислення ряду чисел Фібоначчі. Сценарій прокоментувати. Навести приклади роботи сценарію для обох варіантів.

13. Мовою оболонки bash написати сценарій, який буде друкувати таблицю всіх програмних сигналів для вашої системи UNIX (LINUX). В таблиці має бути 3 стовпчики: в першому – номер сигналу, у другому – назва, у третьому – призначення (англійською мовою). Сценарій прокоментувати, навести приклад роботи сценарію для вашої платформи.
14. Мовою оболонки bash написати сценарій сортування за зростанням цілочислового масиву довжиною 16 елементів за методом злиття (Вікіпедія, сортування злиттям). Початкові значення задаються генератором випадкових чисел. Для реалізації алгоритму створити в сценарії рекурсивну функцію сортування. Після впорядкування результат роботи надрукувати. Сценарій прокоментувати, навести приклади його роботи.
15. Написати shell-сценарій з ім'ям clock, що розділяє години, хвилини та секунди із команди date, присвоює ці значення змінним і виводить на екран інформацію у вигляді: The time is now hr o'clock min minutes and sec seconds. Модифікувати попередній сценарій таким чином, щоб години виводились до 12, а потім друкувалося значення AM (після 12 години) або PM (до 12 години).
16. Написати shell-сценарій, який підраховує та друкує на екрані кількість опцій (аргументи, яким передує символ –) та аргументів (аргументи, яким не передує символ –) у командному рядку. Передбачити сортування та друк усіх опцій в одному рядку та аргументів – у наступному.
17. Модифікувати попередній сценарій таким чином, щоб години виводились до 12, а потім друкувалося значення AM (після 12 години) або PM (до 12 години).
18. Написати shell-сценарій з ім'ям ison, що виконується у фоновому режимі й кожні 60 сек. перевіряє, чи зареєструвався спеціальний користувач у системі (його login_id передається сценарію як аргумент). Якщо користувач зареєструвався, на терміналі має формуватися відповідне повідомлення.
19. Написати shell-сценарій, що виконується у фоновому режимі й кожні 60 сек. перевіряє, чи зареєструвався новий користувач у системі. Якщо новий користувач зареєструвався, на терміналі має формуватися відповідне повідомлення, яке місить login_id тільки-но зареєстрованого користувача.
20. Створити каталог з ім'ям $HOME/.waste. Написати shell-сценарій з ім'ям myrm, що вилучає всі задані як аргументи файли до каталогу .waste. Це корисний інструмент для відновлення файлів після їх вилучення командою myrm. Включити до сценарію обробку двох опцій: -l – роздрук умісту каталогу .waste, -r – знищення вмісту каталогу .waste.
21. Мовою оболонки bash написати сценарій (скористатися командою getopts) сортування за зростанням (опція u) або спадінням (опція d) цілочислового масиву довжиною 16 елементів за методом злиття (Вікіпедія, сортування злиттям). Початкові значення задаються генератором випадкових чисел. Для реалізації алгоритму створити в сценарії рекурсивну функцію сортування. Після впорядкування результат роботи надрукувати. Сценарій прокоментувати, навести приклади його роботи.

22. Мовою оболонки bash написати сценарій, який має 2 опції: l – роздруківка змінних локальної області; e - роздруківка змінних оточення (скористатися командою getopts). Навести приклади всіх варіантів виклику сценарію при використанні різних комбінацій опцій (наприклад, –lx – усього 4 комбінації).
23. Написати сценарій, який підраховує в заданому аргументом каталозі кількість файлів і каталогів. Опції: - f – підраховує кількість файлів; d – підраховує кількість каталогів. При заданні обох опцій підраховуються файли і каталоги (скористатися командою getopts). Навести 4 приклади роботи сценарію.
24. Мовою оболонки bash написати сценарій сортування за методом вставки цілочислового масиву довжиною 20 елементів: за зростанням, за спаданням. Дані вводяться з консолі. Після впорядкування результат роботи надрукувати. Сценарій прокоментувати. Навести приклади роботи сценарію для обох варіантів.
25. Написати shell-сценарій, який друкує у вигляді таблиці перелік груп у системі з номером gid і всіх користувачів у кожній групі з номером uid.
26. Мовою оболонки bash написати сценарій, який має 3 опції: l – роздруківка змінних локальної області; e - роздруківка змінних оточення; s – роздруківка експортованих змінних (скористатися командою getopts). Навести приклади всіх варіантів виклику сценарію при використанні різних комбінацій опцій (наприклад, –les – усього 9 комбінацій).
27. Написати сценарій обходу всіх файлів та директорій в поточному каталозі та вивід їх типу (файл або каталог). Сценарій має обробляти опцію -k – вивід не на пристрій STDOUT, а у файл myfile.txt.
28. Написати сценарій виведення всіх виконуваних файлів системи, що містяться у змінній середовища PATH. Сценарій має обробляти опцію -p – вивід не на STDOUT, а у файл myfile.txt.
29. Написати сценарій, в якому замість стандартного потоку STDIN використовується файл testfile.
30. Написати сценарій, який ілюструє перехоплення сигналу SIGINT ([ctrl_C]). Наприклад, виведення на сигнал повідомлення «^C Trapped Ctrl-C». Для спрощення сприйняття події доцільно скористатися командою sleep 1.
31. Написати shell-сценарій, який виводить на екран усі групи користувачів, що існують у системі, і підраховує їх кількість.
32. Написати shell-сценарій, який виконує друк імен усіх файлів, присутніх у змінній PATH-файлу .profile вашого домашнього каталогу.
33. Написати сценарій, який імітує функції «кошика» у Windows. Передбачити ситуацію наявності кількох файлів із однаковими іменами. Функції кошика – вилучення файлів та відновлення файлів.
34. Написати сценарій, який містить змінну кількість аргументів (n<10) і друкує їх у зворотному порядку.
35. Написати сценарій, який містить змінну кількість аргументів (n>10) і друкує їх у зворотному порядку.

36. Написати сценарій з ім'ям my_menu, що виводить на екран меню з трьох елементів: - надрукувати перші n чисел ряду Фібоначчі; - надрукувати перші n чисел ступенів 2; - вихід. Меню має оновлюватись після здійснення кожного вибору, крім п. 3.
37. Написати shell-сценарій, який містить два аргументи – назву місяця (англійською мовою) і номер року, а далі виводить інформацію аналогічно тому, як це виконує команда cal.
38. Написати shell-сценарій, що формує зведення змін файлу, дату змін і login_id останнього користувача, який ці зміни виконав.
39. Написати shell-сценарій, що створює оболонку, аналогічну за функціями netcat.
40. Написати shell-сценарій моніторингу системних ресурсів, який:
    - Перевіряє використання CPU, RAM, диску.
    - Виводить попередження при перевищенні порогу (опція -t threshold).
    - Логує результати у файл (опція -l logfile).
    - Надсилає повідомлення користувачу при критичних значеннях.

    Приклад виклику:
    ./monitor.sh -t 80 -l /var/log/monitor.log
    Вивід:
    [WARNING] CPU usage: 85% (threshold: 80%)
    [OK] RAM usage: 45%
    [CRITICAL] Disk /home: 95% (threshold: 80%)
41. Написати shell-сценарій генерації випадкових паролів з опціями:
    -l <length> - довжина пароля (за замовчанням 12).
    -n <number> - кількість паролів (за замовчанням 1).
    -u - використовувати великі літери.
    -d - використовувати цифри.
    -s - використовувати спецсимволи.
    -o <file> - зберегти в файл.
42. Написати shell-сценарій перевірки доступності сервісів:
    - Перевірка доступності хостів (ping).
    - Перевірка відкритих портів (telnet/nc).
    - Перевірка HTTP/HTTPS сервісів (curl).
    - Вимірювання часу відповіді.
    - Логування результатів.
    - Консольне повідомлення при недоступності

    Конфігураційний файл (services.conf):
    host1:80:http
    host2:443:https
    host3:22:ssh
    host4::ping
    Опції:
    -c <config> - конфігураційний файл.
    -i <interval> - інтервал перевірки (секунди).

-l <logfile> - лог файл.
-m <email> - email для alerts.
…

43. Написати shell-сценарій аналізу історії команд користувача:
- Топ-N найбільш використовуваних команд.
- Статистика за часом використання (година дня, день тижня).
- Аналіз помилок (команди з ненульовим exit code).
- Виявлення потенційно небезпечних команд (rm -rf, chmod 777 тощо).
- Генерація звіту з візуалізацією.

Опції:
-f <file> - файл історії (за замовчанням ~/.bash_history).
-n <number> - топ-N команд (за замовчанням 10).
-t <type> - тип звіту (commands, time, errors, security).
-o <output> - файл виводу (текст або HTML).

44. Написати shell-сценарій синхронізації двох директорій:
- Одностороння та двостороння синхронізація.
- Виявлення конфліктів (файл змінений в обох місцях).
- Збереження атрибутів файлів (права, час).
- Видалення файлів в папці призначення, при відсутності в початковій папці.
- Логування всіх операцій.

Опції:
-s <source> - джерело.
-d <destination> - призначення.
-m <mode> - режим: one-way або two-way.
-D - видаляти файли в destination.
-n - не виконувати, тільки показати можливі зміни.
-l <logfile> - лог файл.

45. Написати shell-сценарій інтерактивного калькулятора:
- Базові операції: +, -, *, /, %, ^.
- Математичні функції: sqrt, sin, cos, log, exp.
- Змінні: можна зберігати результати.
- Історія обчислень з можливістю повторення.
- Збереження/завантаження сесії.

Команди:
calc <expression> - обчислити.
let <var>=<expr> - зберегти в змінну.
history - показати історію.
replay <id> - повторити обчислення.
save <file> - зберегти сесію.
load <file> - завантажити сесію.
help – довідка.
quit – вихід.

Приклад роботи:
> calc 2 + 3 * 4
Result: 14
> let a = 5
> calc a^2 + sqrt(16)
Result: 29
> history
1: 2 + 3 * 4 = 14
2: let a = 5
3: a^2 + sqrt(16) = 29

Список літератури

1. Robbins A. Bash Pocket Reference. 2nd Edition. – O'Reilly Media, 2016.
2. https://tldp.org/LDP/abs/html/ - інструкція та приклади написання сценаріїв Bash.
3. https://pubs.opengroup.org/onlinepubs/9699919799/ - стандарт Shell POSIX.1-2017 (IEEE 1003.1).
4. https://www.gnu.org/software/bash/manual/ - інструкція GNU Bash.
5. http://www.kornshell.com – офіційний сайт KornShell.

Лабораторна робота № 6

Рекурсія в сценаріях Shell

Метою лабораторної роботи є вивчення основних принципів рекурсії в мові програмування bash; одержання навичок при створенні, налагоджуванні та виконанні рекурсивних shell-сценаріїв та функцій.

Поняття рекурсії

Рекурсія - це програмна техніка, при якій функція викликає саму себе для розв'язання підзадач більшої проблеми. У контексті системного програмування UNIX/FreeBSD рекурсія є потужним інструментом для роботи з ієрархічними структурами даних та файловими системами.

Рекурсивна функція складається з двох основних компонентів:

- Базовий випадок (base case) - умова, при якій рекурсія припиняється
- Рекурсивний випадок (recursive case) - виклик функції самою собою з модифікованими параметрами.

Рекурсія особливо ефективна для розв'язання задач, які мають рекурсивну природу: обхід дерев каталогів, обчислення факторіалів, числа Фібоначчі, сортування (швидке сортування, сортування злиттям) та інші.

Основи shell-функцій в Bash

Перед вивченням рекурсії необхідно розуміти, як працюють функції в bash. Функція в bash - це іменований блок коду, який можна викликати багаторазово.

Синтаксис оголошення функції:

```
function_name() {
    # тіло функції
    local variable_name
    # команди
    return value
}
```

або

```
function function_name {
    # тіло функції
}
```

Принципи побудови рекурсивних функцій

При створенні рекурсивної функції в bash необхідно дотримуватися наступних принципів:

- Визначити базовий випадок - умову, при якій рекурсія припиняється.
- Забезпечити, що кожний рекурсивний виклик наближає до базового випадку.
- Використовувати локальні змінні для уникнення конфліктів імен.
- Передавати параметри явно через аргументи функції.

- Обробляти результати рекурсивних викликів.
- Контролювати глибину рекурсії для уникнення переповнення стеку.

Важливо розуміти, що кожний виклик рекурсивної функції створює новий контекст виконання з власним набором локальних змінних та параметрів.

Далі наведено приклад простої рекурсивної функції:

```
#!/bin/bash

recursive_function() {
    if [ $# -lt 1 ]; then
        echo "Помилка: недостатня кількість аргументів" >&2
        return 1
    fi

    local param=$1

    if [ $param -eq 0 ]; then
        echo 1
        return 0
    fi

    local prev_result=$(recursive_function $((param - 1)))

    echo $((param * prev_result))
}
```

Класичні приклади рекурсії

Розглянемо кілька класичних задач, які ілюструють використання рекурсії в bash.

Обчислення факторіалу

Факторіал числа n (позначається n!) - це добуток всіх натуральних чисел від 1 до n. Формально: n! = n × (n-1) × (n-2) × ... × 2 × 1, при цьому 0! = 1.

Рекурсивна реалізація базується на визначенні: n! = n × (n-1)!

Далі наведений код ітеративної та рекурсивної реалізації алгоритму обчислення факторіалу та порівняння їх швидкодії

```
# Рекурсивна версія факторіалу
factorial_recursive() {
    local n=$1
    if [ $n -le 1 ]; then
        echo 1
    else
        local prev=$(factorial_recursive $((n-1)))
        echo $((n * prev))
    fi
}

# Ітеративна версія факторіалу
```

```
factorial_iterative() {
    local n=$1
    local result=1
    local i

    for ((i=2; i<=n; i++)); do
        result=$((result * i))
    done

    echo $result
}

# Порівняння швидкості роботи
time factorial_recursive 20
time factorial_iterative 20
```

Рекурсивний обхід каталогів

Файлова система UNIX має деревоподібну структуру, що робить рекурсію природним способом її обходу. Рекурсивна функція може відвідати всі файли та підкаталоги, починаючи з заданого каталогу.

```
#!/bin/bash

# Рекурсивний обхід каталогів
traverse_directory() {
    local dir=$1
    local depth=${2:-0}  # Глибина каталогу, необхідна для відображення відступів
    # Перевірка існування каталогу
    if [ ! -d "$dir" ]; then
        echo "Error: $dir is not a directory" >&2
        return 1
    fi

    # Відступ для візуалізації глибини, зручно для директорій з багатьма файлами
    local indent=""
    for ((i=0; i<depth; i++)); do
        indent+="  "
    done

    echo "${indent}[DIR] $dir"

    # Обхід вмісту каталогу
    local item
    for item in "$dir"/*; do
        # Перевірка існування (для порожніх каталогів)
        [ -e "$item" ] || continue
```

```
        if [ -d "$item" ]; then
            # Рекурсивний виклик для підкаталогу
            traverse_directory "$item" $((depth + 1))
        else
            # Файл
            echo "${indent}  - $(basename "$item")"
        fi
    done
}

# Приклад використання
# traverse_directory "/home/user/test"
```

Числа Фібоначчі

Послідовність Фібоначчі визначається як: F(0)=0, F(1)=1, F(n)=F(n-1)+F(n-2) для n≥2. Кожне число в послідовності є сумою двох попередніх.

Рекурсивна реалізація безпосередньо відображає математичне визначення, але має експоненційну складність через повторні обчислення.

В наступній програмі реалізовано **рекурсивний сценарій**, а **не рекурсивну функцію**, як у попередніх прикладах. Ці два варіанти реалізації рекурсії відрізняються тим, що в рекурсивній функції, сама функція викликає саму ж себе, а в рекурсивному сценарії, при новому виклику рекурсії сценарій запускає нову копію себе, але з іншими параметрами.

```
#!/bin/bash
# fibonacci.sh - Рекурсивний сценарій

if [ $# -ne 1 ]; then
    echo "Usage: $0 <n>" >&2
    exit 1
fi

n=$1

if [ $n -eq 0 ]; then
    echo 0
    exit 0
fi

if [ $n -eq 1 ]; then
    echo 1
    exit 0
fi

fib1=$($0 $((n - 1)))
fib2=$($0 $((n - 2)))

echo $((fib1 + fib2))
```

Обмеження рекурсії в bash

При роботі з рекурсивними алгоритмами в інтерпретаторі bash необхідно усвідомлювати ряд суттєвих обмежень, які випливають з архітектури shell-інтерпретатора та специфіки його виконання. На відміну від компільованих мов програмування, де рекурсія може бути оптимізована на етапі компіляції, bash виконує кожний виклик функції як окремий процес інтерпретації, що накладає додаткові обмеження на використання цього підходу.

Найбільш критичним обмеженням є розмір стеку викликів функцій, який в операційних системах сімейства UNIX/FreeBSD зазвичай обмежений значенням ulimit. Кожний рекурсивний виклик функції створює новий фрейм у стеку, який містить локальні змінні, параметри функції та адресу повернення. При надто глибокій рекурсії стек може переповнитися, що призведе до аварійного завершення сценарію з повідомленням про помилку сегментації або переповнення стеку. Для практичних задач в bash рекомендується обмежувати глибину рекурсії до 100-500 рівнів залежно від конфігурації системи та складності рекурсивної функції. Перевірити поточне обмеження стеку можна командою ulimit -s, яка виведе розмір стеку в кілобайтах.

Продуктивність є ще одним суттєвим обмеженням рекурсивних алгоритмів у bash. Оскільки кожний виклик функції вимагає інтерпретації коду, створення нового контексту виконання та управління стеком, рекурсивні реалізації працюють значно повільніше за свої ітеративні аналоги. Для ілюстрації розглянемо обчислення 20-го числа Фібоначчі: наївна рекурсивна реалізація може виконуватись десятки секунд, тоді як ітеративна версія завершується практично миттєво. Ця різниця стає ще більш вираженою для складніших алгоритмів і більших вхідних даних.

З огляду на ці обмеження, при розробці рекурсивних сценаріїв необхідно впроваджувати явний контроль глибини рекурсії. Рекомендованою практикою є передача додаткового параметра, що відслідковує поточну глибину, та перевірка його значення на початку кожної рекурсивної функції. При досягненні максимально дозволеної глибини функція повинна коректно завершитися з відповідним повідомленням про помилку, замість того щоб продовжувати виконання до переповнення стеку. Також корисно логувати глибину рекурсії під час виконання для моніторингу поведінки програми та раннього виявлення потенційних проблем.

Порівняння рекурсії та ітерації

Багато задач можна розв'язати як рекурсивно, так і ітеративно. Вибір підходу залежить від природи задачі та вимог до продуктивності:

Переваги рекурсії:

- Код часто більш читабельний та відповідає математичному визначенню
- Природний спосіб роботи з деревоподібними структурами
- Менше коду для складних алгоритмів

Недоліки рекурсії:

- Більше витрат пам'яті на стек викликів
- Повільніше виконання через накладні витрати на виклики функцій
- Ризик переповнення стеку при великій глибині

Ітеративні алгоритми зазвичай більш ефективні за пам'яттю та швидкістю, але можуть бути складнішими для розуміння.

Контрольні запитання

1. Що таке рекурсія і коли доцільно її використовувати?
2. Які обов'язкові компоненти має містити рекурсивна функція?
3. Що таке базовий випадок і чому він важливий?
4. Як уникнути переповнення стеку при рекурсії?
5. Які переваги та недоліки рекурсивних алгоритмів?
6. Що таке хвостова рекурсія і як вона оптимізується?
7. Чим відрізняється пряма рекурсія від непрямої?
8. Як визначити оптимальну глибину рекурсії?
9. Коли краще використовувати ітерацію замість рекурсії?

Завдання

Кожен студент виконує завдання згідно зі своїм варіантом, отриманим у викладача. Кожне завдання необхідно виконати у двох варіантах:

- ***у вигляді рекурсивної функції, яку треба включити в сценарій;***
- ***у вигляді рекурсивного сценарію.***

Функція повертає числове значення.

1. Обчислення функції Аккермана, яка визначається за наступною формулою:

$$A(m,\ n) = \begin{cases} n+1, & m = 0; \\ A(m-1,\ 1), & m > 0,\ n = 0; \\ A(m-1,\ A(m,\ n-1)), & m > 0,\ n > 0. \end{cases}.$$

   Вхідні дані: невід’ємні цілі числа m і n.
2. Обчислити функцію, яка реалізує операцію множення лише через операції додавання та декременту. Вхідні дані: два множники a і b.
3. Обчислити функцію, яка обраховує суму перших n членів арифметичної прогресії. Вхідні дані: перший член прогресії a, різниця між сусідніми членами d та кількість членів n.
4. Обчислити функцію, яка перетворює число з двійкової системи числення в десяткову. Вхідні дані: двійкове число bin.
5. Обчислити функцію, яка виконує бінарний пошук елемента у відсортованому масиві. Вхідні дані: відсортований масив array, елементи якого передаються через кому: “a0,a1,a2,…,an”; шуканий елемент item. Повертає: індекс шуканого елемента; якщо відсутній, то -1.

6. Обчислити функцію, яка обраховує біноміальний коефіцієнт C(n,k). Вхідні дані: невід’ємні цілі числа n і k.
7. Функція перевіряє парність послідовності круглих дужок. Вхідні дані: послідовність круглих дужок brackets. Повертає: значення True, якщо послідовність парна, інакше False.
8. Обчислити функцію, яка виконує сортування масиву методом «бульбашки». Вхідні дані: масив array, елементи якого передаються через кому: “a0,a1,a2,…,an”. Повертає: відсортований масив.
9. Обчислити функцію, яка перевіряє рядок, чи є він паліндромом (якщо зробити реверс рядка, то він буде ідентичним). Вхідні дані: рядок word. Повертає: значення True, якщо рядок - паліндром, інакше False.
10. Обчислити функцію, яка перетворює число з десяткової системи числення у двійкову. Вхідні дані: невід’ємне ціле десяткове число num.
11. Обчислити функцію, яка підраховує суму цифр числа. Вхідні дані: невід’ємне ціле число num.
12. Обчислити функцію переведення десяткового числа у шістнадцяткову систему.
13. Написати функцію, яка виводить всі файли у заданому каталозі та його підкаталогах. Вхідні дані: шлях до каталогу dir. Повертає: список файлів.
14. Обчислити функцію, яка шукає файл за іменем у вказаному каталозі та його підкаталогах. Вхідні дані: шлях каталогу для пошуку dir, ім’я шуканого файлу filename. Повертає: шлях до знайденого файлу.
15. Обчислити функцію, яка обраховує найбільший спільний дільник двох чисел. Вхідні дані: цілі числа a і b. Повертає: числове значення функції.
16. Обчислити функцію, яка виконує операцію додавання двох чисел лише за допомогою операцій інкременту та декременту. Вхідні дані: невід’ємні цілі числа a і b.
17. Обчислити функцію, яка знаходить максимальний елемент масиву. Вхідні дані: масив array, елементи якого передаються через кому: “a0,a1,a2,…,an”.
18. Обчислення функції, яка реалізує алгоритм сортування методом злиття. Вхідні дані: масив array, елементи якого передаються через кому: “a0,a1,a2,…,an”. Повертає: відсортований масив.
19. Обчислити функцію, яка знаходить мінімальний елемент масиву. Вхідні дані: масив array, елементи якого передаються через кому: “a0,a1,a2,…,an”.
20. Обчислити функцію, яка обраховує суму натуральних чисел. Вхідні дані: кількість чисел n.
21. Обчислення функції, яка знаходить всі перестановки елементів масиву. Вхідні дані: масив arr, елементи якого передаються через кому: “a0,a1,a2,…,an”. Повертає: список усіх можливих унікальних перестановок.
22. Обчислити функцію, яка реалізовує математичну операцію піднесення числа до степеню. Вхідні дані: число x, невід’ємний цілий степінь y.
23. Обчислити функцію, яка реалізує алгоритм швидкого сортування масиву (Quick sort). Вхідні дані: масив array, елементи якого передаються через кому: “a0,a1,a2,…,an”. Повертає: відсортований масив.

24. Обчислити функцію, яка реалізує послідовний пошук елемента в відсортованому масиві. Вхідні дані: відсортований масив array, елементи якого передаються через кому: “a0,a1,a2,…,an”; шуканий елемент item. Повертає: індекс шуканого елемента; якщо відсутній, то -1.
25. Обчислити функцію, яка реалізує операцію ділення лише через операції віднімання та інкременту. Вхідні дані: невід’ємні цілі числа a і b.
26. Обчислити функцію, яка обраховує суму елементів масиву. Вхідні дані: масив array, елементи якого передаються через кому: “a0,a1,a2,…,an”. Повертає: числове значення функції.
27. Обчислення функції, яка виконує обхід бінарного дерева в префіксному порядку. Вхідні дані: бінарне дерево tree, вказане наступний чином:

“parent1->child1,child2;parent2->child3”,

де: parenti – назва батьківського вузла дерева, -> - позначення зв’язку вузлів, childi – назва дочірнього вузла. Через кому вказуються дочірні вузли одного батьківського вузла, а через крапку з комою – окремі гілки дерева. Повертає: список вузлів після обходу дерева в префіксному порядку.
28. Обчислення функції, яка виконує обхід бінарного дерева в інфіксному порядку. Вхідні дані: бінарне дерево tree, вказане наступний чином:

“parent1->child1,child2;parent2->child3”,

де: parenti – назва батьківського вузла дерева, -> - позначення зв’язку вузлів, childi – назва дочірнього вузла. Через кому вказуються дочірні вузли одного батьківського вузла, а через крапку з комою – окремі гілки дерева. Повертає: список вузлів після обходу дерева в інфіксному порядку.
29. Обчислення функції, яка виконує обхід бінарного дерева в суфіксному порядку. Вхідні дані: бінарне дерево tree, вказане наступний чином:

“parent1->child1,child2;parent2->child3”,

де: parenti – назва батьківського вузла дерева, -> - позначення зв’язку вузлів, childi – назва дочірнього вузла. Через кому вказуються дочірні вузли одного батьківського вузла, а через крапку з комою – окремі гілки дерева. Повертає: список вузлів після обходу дерева в суфіксному порядку.
30. Написати рекурсивну функцію підрахунку кількості файлів у каталозі та всіх підкаталогах.
31. Написати рекурсивну функцію знаходження всіх файлів із заданим розширенням у каталозі та підкаталогах.
32. Написати рекурсивну функцію обчислення розміру каталогу (суми розмірів всіх файлів).
33. Написати рекурсивну функцію копіювання каталогу з усім вмістом.
34. Написати рекурсивну функцію видалення порожніх підкаталогів.
35. Написати рекурсивну функцію генерації всіх комбінацій n-елементів із заданої множини.
36. Написати рекурсивну функцію розв'язання задачі про Ханойську вежу.
37. Написати рекурсивну функцію обходу дерева каталогів з фільтрацією за датою модифікації.

38. Написати рекурсивну функцію пошуку рядка в файлах каталогу та підкаталогів (аналог grep -r).
39. Написати рекурсивну функцію обчислення числа шляхів у прямокутній сітці від (0,0) до (n,m).
40. Написати рекурсивну функцію генерації коду Грея для n біт.
41. Написати рекурсивну функцію обчислення детермінанта матриці.
42. Написати рекурсивну функцію знаходження k-го найменшого елемента в масиві.
43. Написати рекурсивну функцію розв'язання задачі про пакування рюкзака.
44. Написати рекурсивну функцію генерації всіх коректних комбінацій дужок для n пар.
45. Написати рекурсивну функцію розбиття числа на суму натуральних чисел.
46. Написати рекурсивну функцію знаходження довжини найдовшої спільної підпослідовності (LCS).
47. Написати рекурсивну функцію розв'язання задачі N ферзів на шахівниці.
48. Написати рекурсивну функцію генерації фрактала (наприклад, трикутник Серпінського).
49. Написати рекурсивну функцію пошуку всіх підмножин множини, сума елементів яких дорівнює заданому числу.
50. Написати рекурсивну функцію обчислення редакційної відстані між двома рядками (Левенштейна).

Список літератури

1. https://www.oreilly.com/library/view/linux-shell-scripting/9781785881985/ - книга з окремим розділом про рекурсії в Bash та прикладами.
2. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms. 3rd Edition. MIT Press, 2009.
3. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. Addison-Wesley, 1983.

Лабораторна робота № 7

Програмування сценаріїв мовою PYTHON. Системне програмування

Метою роботи є вивчення основних особливостей і механізмів функціонування мови PYTHON при створенні сценаріїв для ОС UNIX; одержання навичок для створення, налагоджування та виконання сценаріїв мовами PYTHON, Java, а також системного програмування мовою C++.

Мова оболонки bash (або ksh) не є єдиним засобом створення сценаріїв для ОС UNIX. Це можна робити мовами PYTHON, Java, Perl, Tcl та іншими. Основною мовою системного програмування є C++ (для зв'язку з ядром UNIX використовуються функції API). Для створення сценаріїв необхідно мати інтерфейс між ОС UNIX і мовою PYTHON (це здійснюється за допомогою модулів та пакетів, які необхідно підключити).

Модулі, наведені нижче, забезпечують інтерфейси для функцій, які є унікальними для ОС Unix, або в деяких випадках для кількох її варіантів. Ось основні з них: os (альтернативна реалізація environ), posix, pwd, grp, termios, tty, pty, fcntl, resource, syslog, stat, sys, string, subprocess тощо.

Приклад 1. Сценарій мовою PYTHON для моніторингу активних процесів.

Потрібно перевірити, чи працює певний процес в ОС UNIX.

```
# Сценарій перевіряє активність заданого процесу,
# виводячи повідомлення про його наявність
# або відсутність. Це забезпечує зручний спосіб
# моніторингу важливих процесів у системі. Сценарій сприяє швидкому
# реагуванню на  можливі збої або зупинки служб.

#!/usr/bin/python
import subprocess
def check_process(process_name):
# Виконуємо команду для отримання списку активних процесів
result = subprocess.run(["ps", "aux"], capture_output=True, text=True)
# Перевіряємо, чи є процес у виводі команди
if process_name in result.stdout:
print(f"Process '{process_name}' is running.")
else:
print(f"Process '{process_name}' is not running.")
process_name = "nginx"
check_process(process_name)
```

Приклад 2. Розглянемо приклад простого CGI-сценарія. Класичний шлях створення програм для Інтернет – написання CGI-сценаріїв. CGI (Common Gateway Interface, загальний шлюзовий інтерфейс) – це стандарт, що

регламентує взаємодію сервера із зовнішніми програмами. У випадку з Інтернет веб-сервер може направити запит на генерацію сторінки визначеному сценарію. Цей сценарій, отримавши на вхід дані від веб-сервера (той, у свою чергу, міг отримати їх від користувача), генерує готовий об'єкт (зображення, аудіодані, таблицю стилів тощо).

При виклику сценарію веб-сервер передає йому інформацію через стандартний потік уведення, змінні оточення і, для ISINDEX, через аргументи командного рядка (вони доступні через sys.argv).

Два основні методи передавання даних із заповненої у браузері форми веб-сервера (і CGI-сценарію) – GET та POST. Залежно від методу дані передаються по-різному. У першому випадку вони кодуються та містяться прямо в URL, наприклад: http://host/cgi-bin/a.cgi?a=1&b=3. Сценарій отримує їх у змінній оточення з іменем QUERY_STRING. У випадку методу POST вони передаються на стандартний потік уведення.

Для коректної роботи сценарії розміщують у призначеному для цього каталозі на веб-сервері (звичайно він називається cgi-bin) або, якщо це дозволено конфігурацією сервера, у будь-якому місці серед документів html. Сценарій повинен мати атрибут виконуваності (у файловій системі). У системі Unix його можна встановити за допомогою команди chmod.

Наступний найпростіший сценарій виводить значення зі словника os.environ та дозволяє побачити, що ж було йому передано:

```
#!/usr/bin/python
import os
print """Content-Type: text/plain
%s""" % os.environ
```

За його допомогою можна побачити встановлені веб-сервером змінні оточення. CGI-сценарій, що видає веб-серверу файл, має заголовок, у якому містяться поля з метаінформацією (тип вмісту, час останнього відновлення документа, кодування тощо).

Основні змінні оточення:

- **QUERY_STRING** – рядок запиту.
- **REMOTE_ADDR** – IP-адреса клієнта.
- **REMOTE_USER** – ім'я клієнта (якщо він був ідентифікований).
- **SCRIPT_NAME** – ім'я сценарію.
- **SCRIPT_FILENAME** – ім'я файлу зі сценарієм.
- **SERVER_NAME** – ім'я сервера.
- **HTTP_USER_AGENT** – назва браузера клієнта.
- **REQUEST_URI** – рядок запиту (URI).
- **HTTP_ACCEPT_LANGUAGE** – бажана мова документа.

Ось що може містити словник os.environ у CGI-сценарії:

```
{
'DOCUMENT_ROOT': '/var/www/html',
'SERVER_ADDR': '127.0.0.1',
'SERVER_PORT': '80',
```

```
'GATEWAY_INTERFACE': 'CGI/1.1',
'HTTP_ACCEPT_LANGUAGE': 'en-us, en;q=0.50',
'REMOTE_ADDR': '127.0.0.1',
'SERVER_NAME': 'rnd.onego.ru',
'HTTP_CONNECTION': 'close',
'HTTP_USER_AGENT': 'Mozilla/5.0 (X11; U; Linux i586; en-US;
. . . . .
```

Приклад 3. Наступний CGI-сценарій виводить чорний квадрат (у ньому використовується модуль Image для обробки зображень):

```
#!/usr/bin/python
import sys
print """Content-Type: image/jpeg"""
import Image
i = Image.new("RGB", (10,10))
i.im.draw_rectangle((0,0,10,10), 1)
i.save(sys.stdout, "jpeg")
```

Приклад 4. Написати сценарій мовою Java, який відображає список всіх запущених процесів в ОС UNIX.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class MonitorProcesses {
    public static void main(String[] args) {
        try {
            Process process = Runtime.getRuntime().exec("ps");
            BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            process.waitFor();
        } catch (Exception e) {
            System.err.println("Помилка моніторингу процесів: " +
e.getMessage());
        }
    }
}
```

Контрольні запитання

1. Яким чином CGI-сценарій і сервери обмінюються даними?
2. Які існують засоби підвищення продуктивності CGI-сценаріїв, написаних мовою Python?
3. Яким чином можна використовувати сокети у мові Python?
4. Опишіть призначення модуля **smtplib**.
5. Опишіть призначення модуля **poplib**.
6. Опишіть призначення модулів **urllib, urlparse** та **urllib2**.

7. Опишіть призначення й використання методів модулів urllib, **urlparse** та **urllib2**.
8. Опишіть призначення модуля **environ**.

Завдання

Написати сценарій (або програму C/C++), за завданням та мовою, яка задається викладачем (оболонки bash, C/C++, Python, Java) що виконує таку дію.

1. Хронометрування виконання іншої програми (time). Ім'я програми, її опції та аргументи задаються як аргумент сценарію (наприклад, time ls -R).
2. Реалізує сервіс ping Internet (приєднання до порту 25). Адреса віддаленого мережного комп'ютера задається у вигляді аргументу.
3. Роздруковує час, який минув з моменту останнього звернення певного користувача до mail-файлу. Ім'я користувача задається у вигляді аргументу сценарію.
4. Роздруковує всі пари ключ–значення у певному хеші (реалізація мовою оболонки bash).
5. Роздруковує всі пари ключ–значення у певному хеші (реалізація мовою Python у «зручному форматі»).
6. Ілюструє сортування елементів визначеного хеша у певному порядку (кілька варіантів (зростання, спадання, для заданих значень хешу)). Реалізація мовою оболонки bash.
7. Ілюструє сортування елементів визначеного хеша у певному порядку (кілька варіантів (зростання, спадання, для заданих значень хешу)). Реалізація мовою Python у «зручному форматі».
8. Для заданого аргументом файлу роздруковує час його останньої модифікації (запис або зміни) і звернення (читання).
9. Для двох заданих аргументами імен файлів з'ясовує, чи є вони іменами одного і того самого файлу на диску (завдяки жорстким або символічним посиланням).
10. Для заданого аргументом каталогу рекурсивно роздруковує список усіх його регулярних файлів та підкаталогів.
11. Імітує сеанс telnet-зв’язку із сервером, мережна адреса якого задається аргументом сценарію.
12. За IP-адресою мережного хост-комп'ютера, заданою аргументом, визначає його ім'я.
13. Приєднує комп'ютер до заданого аргументом ftp-сервера та роздруковує вміст його директорії.
14. Виклик функції, при кожному надходженні певного сигналу. Написати сценарій, що ілюструє роботу цього механізму.
15. Видає повідомлення, якщо вказаний аргументом файл було кимось прочитано, записано або вилучено. Якщо файл вилучено, програма має завершуватися. Відстежити модифікацію полів st_atime, st_mtime та значення stat<0, відповідно.

16. Очікує введення інформації із клавіатури протягом 10 сек. Якщо не введено нічого, сценарій друкує «Не введено нічого», інакше – друкує «Дякую». Для введення інформації скористатися викликом read або функціями gets чи getchar. Дослідіть, яке значення повертає fgets (gets) у випадку переривання її системним викликом.
17. Видає поточний час кожні три секунди.
18. Роздруковує імена файлів каталогу та їх повноваження у форматі rwxrwxrwx (аналог команди ls –l). Для одержання повноважень використати системний виклик stat( ).
19. Реєструється за допомогою сервісу telnet на віддаленому TCP/IP-вузлі. Для встановлення з'єднання потрібно у командному рядку набрати команду telnet [ім'я_вузла], де ім'я_вузла – це символьне ім'я або IP-адреса віддаленого комп'ютера. Якщо ім'я_вузла не задано, то команда запускається в інтерактивному режимі. На екран виводиться запрошення telnet> та очікується введення команди. Провести сеанс зв’язку із заданим (викладачем) TCP/IP-вузлом і оформити електронну версію протоколу взаємодії.
20. Копіює файли з/на віддалений комп'ютер за допомогою сервісу ftp. Необхідно встановити ftp-з'єднання (задати у командному рядку команду ftp [ім'я_вузла], після чого зареєструватися: ввести ім'я користувача та пароль. (На ftp-вузлі можна отримати певну інформацію, якщо звернутися на анонімний ftp-сервер: як ім'я користувача вказується слово anonymous, а паролем є слово guest або адреса електронної пошти (за анонімного доступу копіювання файлів на віддалений вузол заборонено)).

    Провести сеанс зв’язку із заданим (викладачем) ftp-вузлом. Прочитати всі команди програми ftp, оформити їх у вигляді таблиці і оформити електронну версію протоколу взаємодії.
21. Встановлює ftp-з'єднання із зовнішньою UNIX-системою та читає всі команди програми ftp. Оформити ці результати у вигляді пронумерованої таблиці у два стовпчики, подати електронну версію протоколу взаємодії.
22. Встановлює telnet-з'єднання із зовнішньою UNIX-системою та підраховує кількість груп. Оформити ці результати у вигляді пронумерованої таблиці у два стовпчики, подати електронну версію протоколу взаємодії.
23. Здійснює обхід (стартовий каталог задається аргументом) всіх розташованих нижче файлів та підкаталогів ієрархії (для C/C++ скористатися файлом включення <ftw.h>).
24. Для програми мовою C/C++ зі змінною кількістю аргументів виводить на екран ім'я програми та перевіряє, чи є аргументи і якщо так, то скільки і їх значення (див. лекцію 9).
25. Для програми мовою C/C++ перевіряє можливість відкриття її файлу на читання. (Перед роботою з файлом його потрібно відкрити. Функція fopen() відкриває існуючий файл або створює новий). У разі успіху повертається вказівник на потік, інакше повертається NULL.

26. Для програми мовою C/C++ виводить на екран монітора значення всіх змінних середовища оточення.
27. Забезпечує роботу сценарію вигляду:
    ім’я_сценарію [-a|-b] [-o вихідний файл],
    де -a і -b несумісні опції:
    - якщо задано опцію -a команда відображає час в секундах, що пройшов з із моменту завантаження ОС;
    - якщо задано опцію -b команда відображає число процесів, що працюють в системі.

    Одержана інформація записується у файл, ім'я якого задається обов’язковим аргументом опції -o.
28. Забезпечує роботу сценарію вигляду:
    ім’я_сценарію [-a|-b] [-o вихідний файл],
    де -a і -b несумісні опції:
    - якщо задано опцію –a, команда відображає ім'я домашнього каталогу, в якому користувач опиняється після входу в систему;
    - якщо задано опцію –b, команда відображає ім'я терміналу.

    Одержана інформація записується у файл, ім'я якого задається обов’язковим аргументом опції -o.
29. Забезпечує роботу сценарію вигляду:
    ім’я_сценарію [-a|-b] [-o вихідний файл],
    де -a і -b несумісні опції:
    - якщо задано опцію –a, команда відображає ім'я поштової скриньки;
    - якщо задано опцію –b, команда відображає ім'я поточного каталогу.

    Одержана інформація записується у файл, ім'я якого задається обов’язковим аргументом опції -o.
30. Забезпечує роботу сценарію вигляду:
    ім’я_сценарію [-a|-b] [-o вихідний файл],
    де -a і -b несумісні опції:
    - якщо задано опцію –a, команда відображає використовувану SHELL-оболонку;
    - якщо задано опцію –b, команда відображає загальний об'єм оперативної пам'яті.

    Одержана інформація записується у файл, ім'я якого задається обов’язковим аргументом опції -o.
31. Підраховує кількість записів в каталозі /dev для символьно-орієнтованих зовнішніх пристроїв і друкує їх загальну кількість, перший та останній записи.
32. Підраховує кількість записів в каталозі /dev для блок-орієнтованих зовнішніх пристроїв і друкує їх загальну кількість, другий та передостанній записи.
33. Створює архів (zip-формат) певного каталогу та зберігає його в іншому каталозі.
34. Видаляє старі файли (часовий поріг задається) в заданому каталозі.

35. Перевіряє доступність певного веб-сайту.
36. Відстежує зміни у певному log-файлі.
37. Видаляє старі резервні копії файлів (видаляються найстаріші файли, якщо їх кількість перевищує заданий ліміт).
38. Вимірює швидкість інтернет-з'єднання. Необхідно виміряти швидкість завантаження та відправлення даних у мережі, для чого встановити бібліотеку speedtest-cli: pip install speedtest-cli. Результати виводяться в мегабітах на секунду (Mbps) для обох параметрів.
39. Перевіряє доступність порту (чи відкритий заданий порт на віддаленому сервері).
40. Архівує файли, старші за певну дату в заданому каталозі.
41. Перевіряє використання дискового простору (аналогічно команді UNIX df –h).
42. Змінює (повноваження) права доступу до файлів (аналогічно команді UNIX chmod).
43. Переглядає системні журнали (аналогічно команді UNIX tail -f /var/log/syslog).
44. Відображає активні мережеві підключення та порти (аналогічно команді UNIX netstat –tuln).
45. Завантажує файл за вказаним URL (аналогічно команді UNIX wget http://example.com/file.zip).
46. Перевіряє доступність сервера (аналогічно команді UNIX ping google.com).
47. Створює жорстке та символічне посилання на вказаний файл, перевіряє їх властивості та демонструє різницю в поведінці при видаленні оригінального файлу. Ім'я файлу задається аргументом.
48. Демонструє міжпроцесну комунікацію через іменований канал (FIFO): сервер створює канал і читає повідомлення, клієнт записує повідомлення до каналу. Ім'я каналу задається аргументом.
49. Демонструє безпечну роботу з тимчасовими файлами та каталогами: створення, запис даних, обробку та гарантоване видалення ресурсів навіть при виникненні помилок.
50. Демонструє низькорівневу роботу з файловими дескрипторами: відкриття файлу системним викликом, позиціонування, читання фрагменту та запис. Ім'я файлу задається аргументом.
51. Знаходить файли за заданим шаблоном у вказаному каталозі, включаючи вкладені підкаталоги, та виводить статистику: кількість знайдених файлів, загальний розмір, найбільший та найменший файл. Каталог та шаблон задаються аргументами.
52. Відображає та модифікує часові позначення файлу (час доступу та модифікації). Дозволяє переглядати поточні значення, встановлювати нові та копіювати позначення з іншого файлу. Імена файлів задаються аргументами.
53. Відстежує зміни у вказаному каталозі в реальному часі: створення, модифікацію, видалення та перейменування файлів. Результати записуються до лог-файлу з часовими мітками. Каталог задається аргументом.

54. Виводить розширену інформацію про поточний процес або про довільний процес за його ідентифікатором: ідентифікатори користувача та групи, статус, використання пам'яті, відкриті файли. Ідентифікатор процесу задається аргументом.
55. Зчитує список чисел зі стандартного введення, сортує їх, фільтрує за заданим критерієм (парні/непарні, більші за поріг) та виводить результат. Критерій фільтрації задається аргументом.
56. Виводить у вигляді таблиці перелік груп системи з їх ідентифікаторами та всіх користувачів з їх ідентифікаторами у кожній групі (на основі файлів /etc/group та /etc/passwd).
57. Переглядає вказаний каталог, для кожного знайденого файлу запитує користувача, чи бажає він переглянути його вміст. Для каталогів пропонує рекурсивний спуск. Каталог задається аргументом.
58. Переміщує вказані файли до спеціального каталогу ($HOME/.waste) замість остаточного видалення. Реалізує опції: -l — перегляд вмісту кошика, -r — відновлення файлу, -e — очищення кошика. Імена файлів задаються аргументами.
59. Порівнює файли у двох каталогах і виводить три списки: файли, присутні лише в першому каталозі; файли, присутні лише в другому каталозі; файли з однаковими іменами, але різним вмістом. Каталоги задаються аргументами.
60. Виконується у фоновому режимі й періодично (кожні 60 сек.) перевіряє, чи зареєструвався вказаний користувач у системі. При виявленні виводить сповіщення та записує до лог-файлу. Ім'я користувача задається аргументом.
61. Реалізує базову функціональність утиліти nc (netcat): режим сервера (прослуховування порту) та режим клієнта (підключення до сервера). Передача даних у текстовому режимі. Адреса та порт задаються аргументами.
62. Створює нового користувача в системі. Приймає ім'я користувача, домашній каталог та командну оболонку як аргументи, створює відповідні записи в системних файлах та домашній каталог з відповідними правами доступу.

Список літератури

1. Special Edition Using Unix (4th Edition). by Peter Kuo, Peter Galvin. Publisher. Que Pub; 4th edition (January 1, 1999).
2. Andrew Tanenbaum. Modern Operating Systems. Prentice Hall PTR, Upper Saddle River, New Jersey 07458, 5th Edition, 2022, 1010 pp.
3. А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий. Програмування числових методів мовою PYTHON. За редакцією чл.-кор. НАН України А. В. Анісімова. Підручник із грифом МОН України. ВПЦ «Київський університет», 2016 р. 640 с.
4. М.Ф. Копитко, К.С. Іванків. Основи програмування мовою Java. Тексти лекцій. Львів, Видавничий центр ЛНУ імені Івана Франка, 2002, 83 с.

Навчальне видання

ПРОГРАМУВАННЯ В СЕРЕДОВИЩІ UNIX-СИСТЕМ
МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ