

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет "Запорізька політехніка"

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни
"Системне програмне забезпечення"

для студентів спеціальності 123 "Комп'ютерна інженерія"
всіх форм навчання

Частина 4

2024

Методичні вказівки до виконання лабораторних робіт з дисципліни "Системне програмне забезпечення" для студентів спеціальності 123 "Комп'ютерна інженерія" всіх форм навчання. Частина 4. / Укл. М.М. Хохлов, О.В. Зелік – Запоріжжя: НУ "Запорізька політехніка", 2024. – 38 с.

Укладачі: М.М. Хохлов, ст..викл.
О.В. Зелік, зав. навч. лаб.

Рецензент: С.Ю. Скрупський, к.т.н., доцент

Відповідальний за випуск: М.М. Хохлов, ст. викладач

Затверджено:
на засіданні кафедри
"Комп'ютерні системи та мережі"
Протокол № 7 від 22 березня 2024

Рекомендовано до видання НМК ФКНТ
Протокол № 8 від 27 березня 2024

ЗМІСТ

Вступ	4
1 Лабораторна робота № 4 - Командний інтерпретатор shell.	
Скрипти.....	5
1.1 Теоретичні відомості	5
1.1.1 Команди, функції та програми.....	7
1.1.2 Змінні shell. Внутрішні зміни	8
1.1.3 Перенаправлення вводу-виводу.....	11
1.1.4 Загальний синтаксис скрипту. Основні конструкції мови програмування shell	13
1.2 Завдання.....	15
1.3 Контрольні питання.....	29
Рекомендована література.....	31
Додаток А. Внутрішні функції shell	32
Додаток Б. Часто використовувані утиліти	33
Додаток В. Вирази, що вживаються в команді test	37
Додаток Г. Спеціальні символи в рядку запрошення	38

ВСТУП

Лабораторні роботи спрямовані на засвоєння базових знань щодо складу системного програмного забезпечення в ОС Linux.

У лабораторних роботах розглядається застосування системного програмного забезпечення що входить до складу сучасних операційних систем, створення програм для реалізації задач взаємодії з операційною системою на системному рівні, доступу до файлової системи.

Відповідно до графіку, студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою.

Для одержання заліку з кожної роботи студент здає викладачу оформлений звіт, а також демонструє на екрані комп'ютера результати виконання лабораторної роботи.

Звіт має містити:

- титульний аркуш;
- тему та мету роботи; завдання до роботи;
- лаконічний опис теоретичних відомостей;
- результати виконання лабораторної роботи;
- змістовний аналіз отриманих результатів, висновки та відповіді до контрольних запитань.

Звіт виконують на білому папері формату А4 (210x297 мм) або подають в електронному вигляді. Під час співбесіди при захисті лабораторної роботи студент повинний виявити знання про зміст роботи та методи виконання кожного етапу роботи, а також вміти продемонструвати результати роботи на конкретних прикладах та правильно аналізувати отримані результати.

1 ЛАБОРАТОРНА РОБОТА № 4 - КОМАНДНИЙ ІНТЕРПРЕТАТОР SHELL. СКРИПТИ

Мета роботи: ознайомитися з основними можливостями стандартних командних інтерпретаторів на прикладі Bourne-Again shell (bash), навчитися створювати власні скрипти, розбиратися в скриптах.

1.1 Теоретичні відомості

Командний інтерпретатор shell - програма базового середовища користувача, яка забезпечує виконання всіх додатків: знаходження програм, що викликаються, їх запуск та організацію вводу-виводу. Окрім цього, shell відповідає за роботу зі змінними середовища та виконує деякі перетворення (підстановки) аргументів. Shell містить в собі просту мову програмування (включаючи умовні оператори та оператори циклу). Shell використовує всі утиліти та програми (як у складі операційної системи, так і ті, що інсталиуються окремо) як базові операції мови програмування, яка підтримується нею, забезпечує передавання їм аргументів, а також передавання результатів їх роботи іншим програмам або користувачу. Таким чином, shell є потужним інструментом, який використовується при адмініструванні системи та конфігурації процесу ініціалізації UNIX.

Всі сучасні системи UNIX поставляються з командними інтерпретаторами:

- Bourne shell (/bin/sh);
- C shell (/bin/csh);
- Korn shell (/bin/ksh);
- Bourne-Again shell (/bin/bash).

Перераховані інтерпретатори мають подібні функції. Командний інтерпретатор займає важливе місце в ОС UNIX внаслідок таких обставин:

а) shell є першою програмою, з якої починається робота користувача.

В UNIX реалізується такий сценарій роботи в системі:

- при завантаженні терміналу активізується процес `getty(1M)`, який є сервером термінального доступу та завантажує програму `login(1)`, яка, в свою чергу, запитує пароль у користувача;
- якщо користувач зареєстрований в системі і увів правильний пароль, `login(1)` запускає програму, яка вказана в останньому рядку запису користувача в файлі `/etc/passwd` (командний інтерпретатор shell);
- shell виконує відповідний командний файл ініціалізації та виводить на термінал користувача запрошення:

\$

З цього моменту користувач може вводити команди;

- shell читає ввід користувача, здійснює синтаксичний аналіз введеного рядка, підстановку шаблонів, роботу зі змінними середовища та виконує дію, задану користувачем (це може бути запуск програми, виконання внутрішньої функції інтерпретатора), або повідомляє про помилку, якщо програма або функція не знайдені;
- при завершенні роботи користувач завершує роботу з інтерпретатором за допомогою команди `exit(1)`, та виходить з системи;

б) командний інтерпретатор є зручним засобом програмування. За допомогою shell можна створювати складні програми – командні файли, будуючи їх з існуючих утиліт UNIX. Програми на мові shell називають **скриптами** (script) або **сценаріями**. Інтерпретатор зчитує рядки з файлу скрипту та виконує їх послідовно, як би вони були уведені користувачем в командному рядку;

в) при вході користувача в систему запускається його скрипт ініціалізації, який виконує декілька функцій: установлення шляху пошуку програм, ініціалізацію терміналу. Окрім цього може бути виконано ряд корисних дій, наприклад, встановлення запрошення. Скрипт ініціалізації знаходиться в домашньому каталозі користувача. Для різних командних інтерпретаторів використовуються різні скрипти ініціалізації. Для `bash` це файли `.bash_profile` та `.bashrc`. Скрипт `.bash_profile` виконується при першому вході в систему. Скрипт `.bashrc` виконується при кожному запуску інтерпретатора;

г) основна ініціалізація ОС здійснюється в результаті виконання скриптів `shell`. Якщо необхідно модифікувати процес ініціалізації (наприклад, додати новий системний сервіс), то необхідно внести зміни до скриптів ініціалізації. Детальну інформацію про командний інтерпретатор `shell` можна отримати користуючись сторінкою електронного довідника `bash(1)`.

1.1.1 Команди, функції та програми

У попередній лабораторній роботі Ви вже ознайомились з утилітами: `man(1)`, `kill(1)` та `ps(1)`.

Всі команди, які вводяться в рядку запрошення `shell`, належать до однієї з категорій:

- внутрішні функції;
- функції `shell`, що визначаються користувачем;
- зовнішні програми та утиліти (пошук програм здійснюється в каталогах, визначених в змінній `PATH`, якщо ім'я команди задано без шляху).

Безпосереднє відношення до `shell` мають тільки перші дві категорії, а програми та утиліти є звичайними файлами виконання. Запуск внутрішньої функції не вимагає породження нового процесу, тому що ця функція реалізована в самій

програмі shell (/bin/bash). Тому внутрішні функції shell виконуються швидше.

Внутрішні функції shell, які найбільш часто використовуються, подані в додатку А.

Користувач може визначити функцію командного інтерпретатора і використовувати її як внутрішню функцію shell. Функції схожі на скрипти, включаючи синтаксис та передавання аргументів. Проте, функції працюють швидше, оскільки є частиною shell. Синтаксис функції має вигляд:

```
function()
{
    command1
    command2
    ...
}
```

Утиліти UNIX, що вживаються найбільш часто, подані в додатку Б.

1.1.2 Змінні shell. Внутрішні змінні

В командному рядку або скрипті командного інтерпретатора можна визначати та використовувати змінні. Значенням змінної є рядок, який передається привласненням:

```
var=value
```

де var – назва змінної, а value – її значення. Значення змінної можна отримати, використовуючи знак \$:

```
$var
```

Окрім того, shell забезпечує зручний спосіб генерації рядків з потоку виводу команди. Синтаксис привласнення при цьому такий:

```
var=`command`
```

де `var` – назва змінної, `command` – назва команди, ``` – символ слабкого наголосу.

Для нормальної роботи в UNIX ряд змінних повинен бути визначений. Найбільш часто вживані змінні середовища такі:

Назва	Опис	Можливе значення
HOME	Каталог користувача верхнього рівня	/home/logname
PATH	Шлях пошуку	/bin:/etc:
TERM	Ім'я терміналу	Xterm
PS1	Первинне запрошення shell	\$
PS2	Вторинне запрошення shell	>
SHELL	Інтерпретатор сеансу	bin/bash
RANDOM	Генератор випадкових чисел	0..32767
PWD	Поточний робочий каталог	/home/logname

Змінна `PATH` призначена для пошуку командним інтерпретатором програм, що запускаються на виконання, якщо їх ім'я не містить шляху. Наприклад, при запуску програми:

```
$ run
```

інтерпретатор буде шукати файл з назвою `run` в каталогах шляху пошуку. Каталоги пошуку в змінній `PATH` розділені символом `'.'`. В той же час при запуску програми `run` із зазначенням шляху, змінна `PATH` використовуватись не буде:

```
$ ./run
```

Тут задане відносне ім'я програми (відносно поточного каталогу, який позначається крапкою). Передбачається, що файл програми існує в поточному каталозі, інакше shell виведе повідомлення про помилку. Зверніть увагу, що поточний каталог пошуку має бути заданий явно ('.'), shell не виконує пошук в поточному каталозі за замовчанням.

Змінна TERM містить ім'я терміналу та використовується програмами для доступу до бази даних терміналів.

Змінні PS1 та PS2 встановлюють первинне та вторинне запрошення командного інтерпретатору. Первинне запрошення вказує на готовність інтерпретатора до вводу команд. Значення змінної PS1 встановлюється при виконанні скрипту ініціалізації (.bash_profile) при вході користувача до системи, та має вигляд "\$" для звичайних користувачів та "#" для суперкористувача. Вторинне запрошення з'являється, якщо натискається клавіша <ENTER>, але при цьому синтаксично не завершено ввід команди.

Крім змінних, що визначаються явно, shell має ряд **внутрішніх змінних**, значення яких встановлюються самим інтерпретатором:

\$1,	– позиційні параметри скрипту – аргументи скрипту
\$2, ...	– який виконується, \$0 – назва власне скрипту;
\$#	– число позиційних параметрів скрипту;
\$?	– код повернення останнього виконаного процесу;
\$\$	– PID поточного shell
\$!	– PID останнього процесу, який виконується в фоновому режимі;
\$*	– всі параметри, які передані скрипту. Зберігаються як єдине слово, і містяться в лапках: "\$1 \$2 \$3 ...";
\$@	- всі параметри, які передані скрипту. Зберігаються як окремі слова, і містяться в лапках: "\$1" "\$2" "\$3" "...".

Назва внутрішньої змінної поза контекстом отримання її значення не має сенсу (тобто не існує змінної #, має сенс лише значення \$#). Ці змінні рідко використовуються при роботі в командному рядку, основне місце їх застосування – скрипти. Слід звернути увагу на виконання команди `set` з параметром–командою ``command``:

```
$ set `command`
```

Вона привласнює позиційним параметрам `$1`, `$2`, ... значення, які є результатом виводу команди ``command``.

1.1.3 Перенаправлення вводу-виводу

Кожна запущена з командного інтерпретатора програма отримує три відкритих потоки вводу-виводу:

- стандартний ввід;
- стандартний вивід;
- стандартний вивід помилок.

За замовчанням всі ці потоки асоційовані з терміналом. Тобто якщо будь-яка програма, не використовує потоки, крім стандартних, то вона буде чекати на введення з клавіатури терміналу, а вивід цієї програми, включно з повідомленнями про помилки, буде здійснюватись на екран терміналу. В основному, стандартні утиліти використовують тільки стандартні потоки. Для таких програм `shell` дозволяє незалежно перенаправляти потоки вводу-виводу. Наприклад, можна подавити ввід повідомлень про помилки, встановити ввід або вивід з файлу, передати вивід однієї програми на ввід іншої.

Стандартним потокам вводу, виводу та виводу помилок привласнюються дескриптори – числові значення, які є покажчиками на відповідний потік. Вони, відповідно, дорівнюють 0, 1 та 2.

Синтаксис перенаправлення вводу-виводу такий:

>file	– перенаправлення стандартного потоку виводу до файлу file;
>>file	– додавання до файлу file даних зі стандартного потоку виводу;
<file	– отримання стандартного потоку вводу з файлу file;
p1 p2	– передавання стандартного потоку виводу програми p1 до потоку вводу p2;
n>file	– переключення потоку виводу з файлу з дескриптором n до файлу file;
n>>file	– те ж, але записи додаються до файлу file;
n>&m	– злиття потоків з дескрипторами n та m;
<<str	– "ввід тут": використовується стандартний потік вводу до підрядка str. При цьому виконуються підстановки метасимволів командного інтерпретатора;
<<\str	– те ж, але підстановки не виконуються.

Підстановки, які виконує shell

Правила зіставлення шаблонів в інтерпретаторі такі:

*	– задає будь-який рядок, включно пустий (будь-яка назва файлу або його частина, крім тих, що починаються з символу '.');
?	– задає будь-який одиночний символ;
[ccc]	– задає будь-який з символів в ccc. [a-d0-3] еквівалентно [abcd0123];
"..."	– задає в точності ...; лапки захищають від спеціальних символів. Аналогічно діє '...';
\c	– задає c буквально;
a b	– тільки для виразів вибору; задає a або b;
/	– для назв файлів відповідає тільки символу / у виразі; для виразів вибору зіставляється як будь-який інший;
.	– якщо це перший символ в назві файлу, то зіставляється тільки з явно заданою крапкою у виразі.

1.1.4 Загальний синтаксис скрипту. Основні конструкції мови програмування shell

Синтаксис мови різних командних інтерпретаторів трохи відрізняється. Як базовий в лабораторній роботі вивчається командний інтерпретатор `bash`.

Скрипт являє собою звичайний текстовий файл, в якому записано команди, вирази `shell` або функції. Командний інтерпретатор зчитує ці інструкції з файлу та послідовно виконує їх. Коментарі в **bash** починаються з символу `#` і можуть займати не весь рядок. Оскільки в системі можуть існувати скрипти для різних інтерпретаторів, ім'я команди, що інтерпретує, звичайно поміщають в перший рядок таким чином:

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

Наступний текст скрипту буде інтерпретувати Bourne - Again shell. При запуску скрипта з командного рядка буде запущено новий командний інтерпретатор, ввід команд до якого буде виконуватись з файлу скрипта. Для виконання скрипт повинен мати відповідний атрибут виконання (атрибут `x`).

Мова `bash` дозволяє здійснювати розгалуження програми за допомогою оператора `if`, його синтаксис має вигляд `if`:

```
if умова
then
    command1
    command2
    ...
else
    command11
    command22
    ...
fi
```

Команди `command1`, `command2` і т.д. будуть виконані, якщо умова є істиною, в протилежному випадку будуть виконані команди `command11`, `command22` і т.д. Умова може генеруватися однією або декількома командами. Істинність умови визначається кодом повернення останньої виконаної команди: істина відповідає коду повернення 0 (на відміну від мови C). Часто умова для оператора **if** генерується командою **test**.

Команда **test** має такий синтаксис:

test вираз
або
[вираз]

Команда обчислює логічний вираз (див. додаток Г) та повертає 0, якщо вираз є істинним, та 1 в протилежному випадку.

Оператор **case** є зручною формою селектора:

```
case слово in
  шаблон1)
    command
    ...
  ;;
  шаблон2)
    command
    ...
  ;;
...
*)
  command
  ...
;;
esac
```

Значення слово порівнюється з шаблонами, починаючи з першого. Якщо відповідність знайдено, то виконуються команди

відповідного розділу, який завершується двома символами ';;'. Шаблони допускають наявність маски. Розділ з шаблоном '*' аналогічний розділу **default** у синтаксисі оператора switch мови C, якщо зіставлення з іншими шаблонами не відбулося, то буде виконано команди розділу '*').

Мова програмування Bourne-Again shell має декілька операторів циклу. Оператори **while** та **for** мають такий синтаксис:

while умова do command1 command2 ... done	for var [in список] do command1 command2 ... done
--	--

За допомогою оператора while команди command1, command2 і т.д. будуть виконуватись, доки умова не буде помилковою. Як і у випадку з оператором if, умова генерується кодом повернення команди, наприклад, test.

Оператор for забезпечує виконання циклу стільки разів, скільки слів у списку. При цьому змінна var послідовно приймає значення, рівні словам з списку. Список може формуватися різними способами, наприклад з використанням внутрішніх змінних, або як вивід деякої команди, ('command') або за допомогою шаблонів shell.

В іншій формі for, коли список відсутній, змінна var приймає значення позиційних параметрів, які передані скрипту.

1.2 Завдання

1. Робота в командному рядку shell. Використання внутрішніх команд bash. Використання змінних.

Перенаправлення вводу-виводу.

1.1 Визначте змінну var1:

```
$ var1=5
```

Виведіть на екран значення даної змінної з деяким поясненням за допомогою команди echo:

```
$ echo -n 'var1=' $var1
$ echo 'var1 = ' $var1
```

В чому різниця виводу даних команд?

1.2 Виконайте команди: \$ cd
/usr/bin \$ pwd

Визначте змінну var2: (зверніть увагу на символ ` – це символ слабкого наголосу, на клавіатурі він знаходиться під клавішею <Esc>):

```
$ var2=`pwd`
```

Виконайте такі команди:

```
$ cd /home/logname
```

де logname – назва Вашого домашнього каталогу \$

```
pwd
$ cd $var2
$ pwd
```

Поясніть виконання кожної команди.

1.3 Виконайте команду set:

```
$ set
$ set | more
$ set | less
$ cd
```

```
$ set > set.txt
```

Поясніть роботу команди set. В чому різниця виводу даних команд? Де знаходиться файл set.txt? Що значать операції '>' та '|'?

Запишіть до звіту значення змінних: BASH, HISTFILE, HOME, PATH, PS1, PS2, PPID, PWD, RANDOM, TERM, USER та прокоментуйте їх.

1.4 Продивіться значення змінної PATH:

```
$ echo $PATH
```

Поясніть вивід даної команди.

Змініть значення змінної PATH – включіть в шлях пошуку файлів виконання свій каталог /home/logname/os/lab1:

```
$ PATH=$PATH:/home/logname/os/lab1
```

1.5 Перегляньте значення змінної PWD:

```
$ cd
```

```
$ echo $PWD
```

```
$ cd /usr/bin
```

```
$ echo $PWD
```

Поясніть виконання кожної команди.

1.6 Перегляньте значення змінної RANDOM:

```
$ echo $RANDOM
```

```
$ echo $RANDOM
```

1.7 Перегляньте значення змінної PS1 (запрошення командного рядка shell):

```
$ echo $PS1
```

Поміняйте змінну PS1. Спеціальні символи для формування запрошення див. в додатку B:

```
$ PS1="[\@ \u \#]$ "
```

Поясніть, що виводиться в даному запрошенні. Далі самостійно змініть запрошення командного рядка так , щоб в ньому були присутні поточна дата, назва оболонки, назва хосту, повна назва поточного робочого каталогу та символ '\$'. Запишіть формат даного запрошення до звіту.

2. Робота в командному рядку shell. Виклик найбільш часто вживаних команд та утиліт UNIX:

2.1 В своєму домашньому каталозі /home/logname/os створіть каталог lab_x, в якому будуть зберігатись Ваші файли з

цієї лабораторної роботи. Виконайте це за допомогою команди `mkdir`:

```
$ mkdir /home/logname/os/lab2
```

2.2 Створіть в каталозі `lab_x` два каталоги: `test1` та `test2`:

```
$ mkdir /home/logname/os/lab_x/test1
```

```
$ cd /home/logname/os/lab_x
```

```
$ mkdir test2
```

2.3 Створіть 5 файлів в каталозі `test1` за допомогою перенаправлення потоку виводу команд до файлів:

```
$ cd ./test1
```

```
$ date > date1.txt
```

```
$ date > date2.txt
```

```
$ cal > month.txt
```

```
$ cal -my > year.txt
```

```
$ cal -mj > julian.txt
```

2.4 Перегляньте зміст створених в п.2.3 файлів за допомогою команди `cat`:

```
$ cat date*
```

```
$ cat month.txt
```

```
$ cat year.txt
```

```
$ cat julian.txt
```

Проаналізуйте вивід команди **cal** в залежності від опцій (див. додаток Б).

2.5 Порівняйте файли `date1.txt` та `date2.txt` за допомогою команди `cmp`:

```
$ cmp date1.txt date2.txt
```

Поясніть результат виводу команди.

2.6 Перейменуйте файл `year.txt` в `calendar.txt` за допомогою команди `mv`:

```
$ mv year.txt calendar.txt
```

2.7 Скопіюйте файл calendar.txt до файлу new.txt за допомогою команди cp. Файл new.txt має бути в каталозі lab_x:

```
$ cp calendar.txt /home/logname/os_lab_x
```

2.8 Скопіюйте файли date1.txt та date2.txt в каталог test2:

```
$ cp date* /home/logname/os/lab_x/test2
```

2.9 Знайдіть файл julian.txt за допомогою команди find:

```
$ find / -name julian.txt -print
```

```
$ find / -name julian.txt -print 2> /dev/null
```

```
$ find /home -name julian.txt -print
```

```
$ find /home -name julian.txt -print 2> /dev/null
```

Зверніть увагу: для подавлення виводу помилок використовується перенаправлення 2> /dev/null, де /dev/null є псевдо-пристроєм, який вилучає всі введені до нього символи.

2.10 Отримайте статистику за файлом month.txt за допомогою команди wc:

```
$ wc month.txt
```

2.11 Вилучіть файл julian.txt з каталогу test1 за допомогою команди rm:

```
$ rm julian.txt
```

2.12 Вилучіть каталог test2. Зверніть увагу на опцію -r, яка використовується для вилучення непорожніх каталогів разом з усім змістом:

```
$ rm -r /home/logname/os/lab2/test2
```

3. Робота в командному рядку shell. Визначення функції користувача.

3.1 Визначте свою власну функцію mcd, що дозволяє виводити в запрошенні shell ім'я поточного каталогу:

```
$ mcd()
> {
> PS1=`pwd`
> PS1="$PS1 $"
> }
```

Перевірте роботу визначеної щойно функції:

```
$ mcd
$ cd /bin
$ mcd
$ cd /home
$ mcd
```

4. Робота зі скриптами.

4.1 Перехоплення сигналів

4.1.1 Вивчіть скрипт myscript_sigint.

Для того, щоб включити в скрипті відстеження сигналів Linux, використовується команда `trap`. Якщо скрипт отримує сигнал, зазначений при виклику цієї команди, він обробляє його самостійно, при цьому оболонка такий сигнал обробляти не буде.

Команда `trap` дозволяє скрипту реагувати на сигнали, в іншому випадку їх обробка виконується оболонкою без його участі.

Розглянемо приклад, в якому показано, як при виклику команди `trap` задається код, який треба виконати, і список сигналів, розділених пробілами, які ми хочемо перехопити. В даному випадку це всього один сигнал:

```
#!/bin/bash
trap "echo ' Trapped Ctrl-C'" SIGINT
echo This is a test script
count=1
while [ $count -le 10 ]
do
    echo "Loop #$count"
    sleep 1
```

```
count=$(( $count + 1 ))
done
```

Команда `trap`, використана в цьому прикладі, яка виводить текстові повідомлення щоразу, коли вона виявляє сигнал `SIGINT`, який можна згенерувати, натиснувши `Ctrl + C` на клавіатурі.

Кожен раз, коли ви натискаєте клавіші `CTRL + C`, скрипт виконує команду `echo`, зазначену при виклику `trap` замість того, щоб дозволити оболонці завершити його роботу.

4.1.2 Щоб скрипт можна було виконати, необхідно встановити атрибут виконання, для чого скористайтесь командою:

```
$ chmod +x myscript_sigint
```

4.1.3 Виконайте скрипт `$ ./myscript_sigint`

Проаналізуйте як працює скрипт. Поясніть результати виводу скрипту.

4.2 Перехоплення сигналу виходу з скрипта. Перехопити сигнал виходу зі скрипта можна, використавши при виклику команди `trap` назва сигналу `EXIT`

4.2.1 Ознайомтесь зі скриптом `myscript_exit`:

```
#!/bin/bash
trap "echo Goodbye..." EXIT
count=1
while [ $count -le 5 ]
do
    echo "Loop #$count"
    sleep 1
    count=$(( $count + 1 ))
done
```

При виході зі скрипта, чи то нормальне завершення його роботи чи завершення, викликане сигналом SIGINT, спрацює перехоплення і оболонка виконає команду echo.

4.2.2 Виконайте скрипт `$ ./myscript_exit`

Проаналізуйте як працює скрипт. Поясніть результати виводу скрипту.

4.3 Модифікація перехоплених сигналів і скасування перехоплення.

Для модифікації перехоплених скриптом сигналів можна виконати команду **trap** з новими параметрами.

4.3.1 Ознайомтесь зі скриптами `myscript_mod` та `myscript_mod1`:

```
#!/bin/bash
trap "echo 'Ctrl-C is trapped.'" SIGINT
count=1
while [ $count -le 5 ]
do
    echo "Loop #$count"
    sleep 1
    count=$(( $count + 1 ))
done
trap "echo ' I modified the trap!'" SIGINT
count=1
while [ $count -le 5 ]
do
    echo "Second Loop #$count"
    sleep 1
    count=$(( $count + 1 ))
done
```

Після модифікації сигнали будуть оброблятися по-новому.

Перехоплення сигналів можна і скасувати, для цього досить виконати команду `trap`, передавши їй подвійне тире і ім'я сигналу:

```
#!/bin/bash
trap "echo 'Ctrl-C is trapped.'" SIGINT
count=1
while [ $count -le 5 ]
do
    echo "Loop #$count"
    sleep 1
    count=$(( $count + 1 ))
done
trap -- SIGINT
echo "I just removed the trap"
count=1 while [ $count -le 5 ]
do
    echo "Second Loop #$count"
    sleep 1
    count=$(( $count + 1 ))
done
```

Якщо скрипт отримає сигнал до скасування перехоплення, він обробить його так, як задано в діючій команді trap.

4.3.2 Запустіть скрипт і натисніть CTRL + C

Перше натискання CTRL + C довелося на момент виконання скрипта, коли перехоплення сигналу був в силі, тому скрипт виконав призначену сигналу команду echo. Після того, як виконання дійшло до команди скасування перехоплення, команда CTRL + C спрацювала звичайним чином, завершивши роботу скрипта.

4.3.3 Проаналізуйте роботу скриптів. Поясніть результати виводу скриптів.

4.4 Робота зі скриптами. Використання внутрішніх змінних. Застосування оператора циклу **for**.

4.4.1 Вивчіть скрипт L2_param_for:

```
# Скрипт L2_param_for: внутрішні змінні та
цикл for
#!/bin/bash
echo ''
echo Name of script: $0
echo -n Number of script parameters:
echo $#
echo All parameters of script: $*
echo Each parameters of script:
for i in $*
do
    echo $i
done
echo The second parameter of script: $2
echo ''
echo -n Output of 'date' command:
date
set `date`
echo Setting of parameters:
echo Parameter 1 (week day) = $1
echo Parameter 2 (month) = $2
echo Parameter 3 (day) = $3
echo Parameter 4 (time) = $4
echo Parameter 5 = $5
echo Parameter 6 (year) = $6
echo ''
```

Примітка: зверніть увагу на формат відображення поточної дати командою `date` на Вашій робочій станції (зверніться до п.2.3-2.4) якщо значення не відповідають номерам \$1...6, які передбачено в скрипті, – виконайте відповідні зміни номерів (або змініть формат виводу команди `date`)

4.4.2 Вивчіть використання внутрішніх змінних shell, роботу оператора for. Проаналізуйте як працює скрипт.

4.4.3 Виконайте скрипт L2_param_for з аргументами:

```
$ ./L2_param_for aaa bbb ccc
```

```
$ ./L2_param_for aaa
```

```
$ ./L2_param_for
```

Поясніть результати виводу скрипту.

4.5 Робота зі скриптами. Використання оператора умови if та команди test ([]).

4.5.1 Ознайомтесь зі скриптом L2_if:

```
# Скрипт L2_if: оператор умови if
```

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

```
if [ $# -ne 1 ]
```

```
then
```

```
    echo usage: $0 arg1
```

```
    exit 1
```

```
else
```

```
    grep $1 /etc/passwd > /dev/null
```

```
    if [ $? -eq 0 ]
```

```
    then
```

```
        echo User $1 is registered in system
```

```
    else
```

```
        echo User $1 is not registered in system
```

```
    fi
```

```
fi
```

4.5.2 Вивчіть роботу оператора if, команди test ([]) (див. додаток В) та команди grep. Проаналізуйте як працює скрипт. Скільки аргументів запрограмовано для використання даного скрипту?

4.5.3 Виконайте скрипт L2_if з такими аргументами:

```
$ ./L2_if
$ ./L2_if root
$ ./L2_if abcdefg
```

Поясніть результати виводу скрипту.

4.6 Робота зі скриптами. Застосування оператора циклу **while**. Використання команди read.

4.6.1 Ознайомтесь зі скриптом L2_while:

```
# Скрипт L2_while: оператор циклу while
#!/bin/bash
echo "input:"
while read var1 var2 var3
do
    echo var1=$var1
    echo var2=$var2
    echo var3=$var3
    echo "input:"
done
```

4.6.2 Вивчіть роботу оператора while, команди read.

4.6.3 Виконайте скрипт L2_while таким чином:

```
$ ./L2_while
```

Після запрошення:

```
input:
```

введіть:

```
example of read command work
```

Зверніть увагу на значення змінних: var1, var2, var3. Після наступного звернення:

```
input:
```

введіть:

another example

Зверніть увагу на значення змінних: var1, var2, var3.

В черговий раз після запрошення:

input:

введіть:

<Ctrl>+<D>.

4.7 Робота зі скриптами. Використання оператора вибору **case**.

4.7.1 Ознайомтесь зі скриптом L2_case:

```
# Скрипт L2_case: використання команди cal-
календар
#!/bin/bash
case $# in
    0) set `date`; m=$2; y=$6;; # нема аргументів:
                                # поточні місяць, рік
    1) m=$1; set `date`; y=$6;; # 1 аргумент:
місяць
                                # рік - поточний
    *) m=$1; y=$2;; # 2 аргументи: місяць, рік
esac
case $m in # чи коректно заданий місяць?
    jan*|Jan*) m=1;; # може бути заданий
    feb*|Feb*) m=2;; # назвою або номером
    mar*|Mar*) m=3;;
    apr*|Apr*) m=4;;
    may*|May*) m=5;;
    jun*|Jun*) m=6;;
    jul*|Jul*) m=7;;
    aug*|Aug*) m=8;;
    sep*|Sep*) m=9;;
    oct*|Oct*) m=10;;
```

```

nov*|Nov*) m=11;;
dec*|Dec*) m=12;;
[1-9]|10|11|12) ;; # передбачається, що
заданий рік
esac
/usr/bin/cal $m $y # запуск команди cal

```

Примітка: зверніть увагу на формат відображення поточної дати командою `date` на Вашій робочій станції (аналогічно як у п.4.4.1)

4.7.2 Вивчіть роботу оператора `case`, використання символу `|` в шаблонах оператора `case`.

4.7.3 Проаналізуйте роботу скрипту. Виконайте скрипт `L2_case` з аргументами:

```

$ ./L2_case
$ ./L2_case 12
$ ./L2_case aug
$ ./L2_case 1995 22
$ ./L2_case 7 2004
$ ./L2_case March 2005

```

4.7.4* Модифікуйте скрипт `L2_case` так, щоб можна було задавати більше одного місяця, наприклад:

```

$ ./L2_case_n oct nov
і навіть діапазон місяців:
$ ./L2_case_n oct-dec

```

Текст модифікованого скрипту `L2_case_n` подайте у звіті.

4.8 Самостійно напишіть скрипт `L2_info`, аргументами якого є три назви файлів. Для цих файлів необхідно вивести на екран час їх останньої модифікації та час останнього доступу до

файлів (використовуйте команду `ls`). У випадку, якщо число аргументів не дорівнює трьом, необхідно вивести повідомлення про синтаксис запуску скрипту та завершити його виконання.

Продемонструйте роботу скрипту. Текст скрипту `L2_info` подайте у звіті.

4.9 * Робота з скриптами ініціалізації.

4.9.1 Проаналізуйте зміст скриптів ініціалізації: **.bash_profile** та **.bashrc**. Стисло опишіть дії, які виконує інтерпретатор при обробці даних файлів.

4.9.2 Розгляньте файл `/etc/bashrc`, який включений до `.bashrc` файла. Стисло опишіть дії, які виконує інтерпретатор при обробці даного файлу.

4.9.3 Поясніть зміст файлу **.bash_history**

1.3 Контрольні питання

1. Які функції виконує командний інтерпретатор shell? Назвіть відомі стандартні командні інтерпретатори?
2. Що таке скрипт? Як здійснюється виконання скрипту?
3. Для чого використовуються скрипти ініціалізації?
4. Опишіть загальний синтаксис скрипту.
5. Як використовувати змінні в скрипті? Що таке внутрішні змінні?
6. Наведіть синтаксис перенаправлення вводу-виводу.
7. Поясніть призначення основних операторів мови програмування **bash: if, case, for, while**.
8. Опишіть використання команди **test**.
9. Як виконується відстеження сигналів у Linux?

10. Як виконується перехоплення сигналів у скрипті, чи можна зупинити перехоплення, яким чином?
11. Поясніть, що таке модифікація перехоплених сигналів.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Геннадій Галісеєв. Системне програмування. – Університет "Україна". 2019 – 113 с.
2. Robert Love. Linux System Programming, Second Edition. – O'Reilly Media, Inc. 2013 – 456 с.
3. Carla Schroder. Linux Cookbook. Essential Skills for Linux Users and System & Network Administrators. 2nd Ed. - O'Reilly, 2021 - 600с.
4. Carl Albing. bash Cookbook: Solutions and Examples for bash Users 2nd Edition - O'Reilly, 2017 - 726с.
5. Шевцов В.А. Операційні системи. - К.: Видавнича група BHV, 2005 – 576 с.
6. Daniel J. Barrett. Linux Pocket Guide - O'Reilly Media; 3rd edition. 2016 - 267с.
7. Daniel J. Barrett. Linux Guide: Essential Commands - Oreilly & Associates Inc; Anniversary edition, 2024 - 347с.
8. Andrew S. Tanenbaum, Herbert Bos. Modern Operating Systems. Fifth Edition – Hoboken, New Jersey 07458 – 2024. – 1185 с. ISBN: 978-1-292-45966-0

Додаток А

Внутрішні функції shell

:	пуста команда. Код повернення завжди 0 (успіх). Зручна для створення нескінченних циклів;
break [n]	здійснює вихід з циклу for або while. Якщо параметр n вказаний, відбувається вихід з n вкладених циклів;
cd [dir]	здійснює перехід до каталогу dir. Якщо параметр не заданий, відбувається перехід до домашнього каталогу (\$HOME);
echo [st]	рядок st виводиться до стандартного пристрою виводу (термінал);
exit [n]	завершує роботу поточного інтерпретатора (або скрипту) з кодом повернення n. За замовчанням код повернення дорівнює 0;
export	розміщує змінні, що вказані як аргументи, в [name 1, середовищі поточного shell, після цього вони є name 2] доступними для програм, що запускаються з інтерпретатору;
kill [-sig] pid1 pid2 ...	відправити сигнал, що визначається параметром sig, процесам, які задаються параметрами pid. За замовчанням команда kill відправляє сигнал з номером 15 – SIGTERM, для якого дія за замовчанням – завершення виконання процесу, який отримує сигнал;
pwd	виводить назву поточного каталогу;
read v1 v2 ...	послідовно зчитує слова (групи символів, розділені пробілами) зі стандартного потоку вводу, послідовно привласнюючи змінним v, що задані як параметри, значення, які дорівнюють словам. Якщо число слів в рядку перевищує число змінних, тоді останній змінній привласнюється значення, що дорівнює залишку рядка;

Додаток Б

Часто використовувані утиліти

Поле [opt] містить конкретні опції кожної утиліти. Більш детально синтаксис команд див. в електронному довіднику man в розділі 1.

Таблиця Б - Часто використовувані утиліти

cmp [opt] file1 file2	порівнює два файли, вказаних як аргументи. Якщо файли однакові, ніяке повідомлення не виводиться. В протилежному випадку виводяться дані про першу невідповідність між цими файлами;
cp [opt] file1 file2	використовується для копіювання файлів;
cp [opt] file1 ... dir	використовується для копіювання файлів до каталогу;
mv [opt] file1 file2	змінює ім'я файлу;
mv [opt] file1 ... dir	переміщення вказаних файлів до каталогу;
rm [opt] file1...	вилучення файлів та каталогів;
rmdir dir1...	видалення пустих каталогів;
ls [opt] [file1 file2 ...]	без параметрів утиліта ls виводить список файлів та каталогів поточного каталогу в алфавітному порядку. Як параметри можна задати назви каталогів, зміст яких необхідно вивести, або назви файлів, інформацію про які треба отримати. Опції утиліти дозволяють отримати список різної інформативності та формату. Найбільш поширені з них: -a - вивід як і без опцій, але виводить й файли, що починаються з '.';

Продовження таблиці Б

	-l - виводить тип файлу, права доступу до файлу, кількість посилань на файл, ім'я власника, ім'я групи, розмір файлу в байтах, часовий штамп та ім'я файлу. За замовчанням часовий штамп є часом останньої модифікації; -u - використовувати при сортуванні час останнього доступу до файлу замість часу останньої модифікації; -F - для кожної назви каталогу додавати суфікс '/', для кожної назви виконавчого файлу '*'; -R - включити рекурсивну видачу списку каталогів;
mkdir [-m mode] [-p] dir1...	створити каталог;
fgrep [opt] <підрядок> file1...	утиліта пошуку фрагментів тексту в файлах. Може використовуватись як фільтри в програмних каналах. Застосовується для пошуку підрядка в файлах. Якщо підрядок пошуку містить пробіли або знаки табуляції, її потрібно вмістити в лапки. Якщо підрядок вже містить лапки, перед ними слід помістити символ '\' безпосередньо перед лапками. Якщо необхідно зробити пошук чутливим до великих/рядкових символів, використовуйте ключ -u. Для пошуку рядків, які не містять вказаний підрядок, використовується ключ -v;
grep[opt] <рег.вираз> file1...	утиліта пошуку фрагментів тексту у файлах. Може використовуватись як фільтри в програмних каналах. Дозволяє здійснювати більш складний пошук, наприклад, коли Ви не впевнені в написанні слова пошуку, або бажаєте знайти слова, розташовані у визначених місцях файлу.

Продовження таблиці Б

	В цьому випадку як підрядок вказується регулярний вираз <рег.вираз>;
cat [opt] file	cat file виводить зміст файлу file на екран. Якщо файл не текстовий, тобто містить "недруковані" символи, краще задати команду з ключом -v для подавлення виводу таких символів;
more [opt] file	дозволяє виводити файл порціями на екран (зручно використовувати у випадку великих файлів);
head [-n] file	дозволяє продивитись тільки початок (перші n рядків) файлу;
tail [opt] file	дозволяє продивитись тільки кінець (останні n рядків) файлу;
less	дозволяє виводити файл порціями на екран (інтерфейс як в команді man);
sort	використовується для сортування рядків файлу. Наприклад, для сортування тексту в алфавітному порядку необхідно ввести команду: \$ sort -d file >sorted_file Можна вказати номер слова рядка, за яким необхідно здійснити сортування (точніше, номер поля запису, за замовчанням записом є рядок, а поля розділені пробілами);
cut	дозволяє відфільтрувати вказані поля рядків файлу. Роздільник полів вказується опцією -d<sep>
wc file	дозволяє вивести число рядків, слів та символів тексту файлу;

Продовження таблиці Б

find dir [opt]	виконує пошук файлу в файловій системі UNIX, починаючи з каталогу dir. За допомогою опції -name вказується назва файлу пошуку, а за допомогою -print – дія (вивести повну назву). Можна здійснювати пошук файлів за другими критеріями, наприклад, розміру, останньому часу модифікації і т.д.;
chown user file...	змінює власника-користувача вказаних файлів;
chgrp group file...	змінює власника-групу вказаних файлів;
chmod mode file...	змінює права доступу та додаткові атрибути файлів;
date	виводить поточну дату;
cal [- m]y13] [[month] year]	виводить календар. Якщо немає аргументів, то виводиться тільки поточний місяць. Використовуються такі опції: -m – виводить понеділок як перший день тижня; -y – виводить календар поточного року; -j – виводить номери днів, відраховуючи їх з 1-го січня; -3 – виводить 3 місяця: попередній / поточний / наступний.

Додаток В

Вирази, що вживаються в команді test

Вирази з файлами:	
-s file	розмір файла file більше 0;
-r file	для файлу file дозволений доступ на читання;
-w file	для файлу file дозволений доступ на запис;
-x file	для файлу file дозволене виконання;
-f file	файл file існує і є звичайним файлом;
-d file	файл file є каталогом;
-c file	файл file є файлом символьного пристрою;
-b file	файл file є файлом блочного пристрою;
-p file	файл file є поймаючим каналом;
-u file	файл file має встановлений флаг SUID;
Вирази з рядками:	
-z str	рядок str має нульову довжину;
-n str	довжина рядка str більш 0;
str1=str2	два рядки ідентичні;
str1!=str2	два рядки різні.
Порівняння цілих чисел:	
i1 -eq i2	i1 дорівнює i2;
i1 -ne i2	i1 не дорівнює i2;
i1 -lt i2	i1 строго менше i2;
i1 -le i2	i1 менше або дорівнює i2;
i1 -gt i2	i1 строго більше i2;
i1 -ge i2	i1 більше або дорівнює i2.
Складні вирази можуть бути створені з логічними операторами:	
!вираз	істина, якщо вираз невірний (оператор NOT);
вираз1 -a вираз2	істина, якщо обидва вирази вірні (оператор AND);
вираз1 -o вираз2	істина, якщо хоч один з виразів вірний (оператор OR).

Додаток Г
Спеціальні символи в рядку запрошення

\a	звуковий сигнал (ASCII код 07);
\d	дата в форматі "день місяць число";
\h	ім'я хосту (до першої крапки);
\H	повна назва хосту;
\t	поточний час у 24-годинному форматі HH:MM:SS;
\T	поточний час в 12- годинному форматі HH:MM:SS;
\@	поточний час в 12- годинному форматі am/pm;
\u	ім'я користувача, який запустив оболонку;
\w	повне назва робочого каталогу (починаючи з кореня);
\W	поточний робочий каталог (без вказання шляху);
\\$	символ \$, якщо оболонку запустив звичайний користувач, та символ #, якщо оболонку запустив суперкористувач;
\s	назва оболонки;
\#	поточний номер команди.