МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни

“Вбудовані операційні системи”

для студентів напряму 6.050101
“Комп’ютерні науки”
спеціальності "Інформаційні технології проектування"

денної форми навчання

2015

Методичні вказівки до виконання лабораторних робіт з дисципліни “Вбудовані операційні системи” для студентів напряму 6.050101 “Комп’ютерні науки“ спеціальності "Інформаційні технології проектування" денної форми навчання /Уклад.: Сердюк С.М., Качан О.І. – Запоріжжя: ЗНТУ, 2015. – 47 с.

Укладачі: С. М. Сердюк, к.т.н., доцент кафедри ПЗ.
О.І. Качан, асистент кафедри ПЗ.

Рецензенти: С.К. Корнієнко, к.т.н., доцент кафедри ПЗ
О.О. Степаненко, к.т.н., доцент кафедри ПЗ

Відповідальний
за випуск: В.І. Дубровін, зав. каф. ПЗ, к.т.н., професор

Затверджено
на засіданні кафедри
"Програмні засоби"

Протокол № 9 від 15.05.2015 р.

ЗМІСТ

1 ЛАБОРАТОРНА РОБОТА №1

Командний інтерпретатор bash. Основи написання сценаріїв (скриптів)

Мета роботи: Освоїти командний інтерпретатор bash. Набути початкових навичок написання командних файлів (скриптів).

1.1 Стислі теоретичні відомості

Командний інтерпретатор (оболонка, shell) є інтерфейсом користувача в UNIX-системі. Командний інтерпретатор - це просто програма, яка дозволяє системі розуміти команди користувача (звідси назва), і дає йому можливість створювати зручне для себе середовище роботи в UNIX. Як правило, дії інтерпретатора не помітні користувачеві, вони відбуваються як наче за кулісами.

Командний інтерпретатор можна розглядати як захисну оболонку ядра системи. Як відомо, при запуску системи ядро завантажується в пам'ять і виконує багато низькорівневих системних функцій. Ядро регулює роботу процесора, здійснює і регулює протікання процесів, відповідає за ввід/вивід даних. Можливе існування тільки одного ядра. Інструкції ядра складні, громіздкі, і сильно прив'язані до апаратних засобів. Працювати на мові такого рівня дуже важко, тому і виникло багато командних інтерпретаторів (оболонок). Вони захищають користувача від складності ядра, а ядро - від некомпетентності користувача. Користувач дає команди інтерпретатору, той в свою чергу перекладає їх на системну мову і передає ядру.

Функціями якого завгодно інтерпретатора є:

- інтерпретація командного рядка;
- ініціалізація програм;
- перенаправлення потоків вводу/виводу;
- організація конвеєрного виконання програм (каналів);
- підстановка імен файлів;
- робота зі змінними;
- контроль за середовищем;
- надання засобів керування задачами;
- програмування.

В оболонці визначаються змінні, які керують поведінкою Вашої сесії роботи з UNIX. Вони повідомляють системі, який каталог вважати Вашим робочим каталогом, в якому файлі зберігати вхідну електронну пошту та ін. Деякі змінні попередньо установлюються операційною системою, інші можна визначити самому в файлах початкового завантаження.

В командних інтерпретаторах передбачені спеціальні вбудовані команди, які можуть використовуватися для побудови командних файлів. Командний файл (скрипт, сценарій) - це текстовий файл, що містить UNIX-команди (аналог bat-файлів в DOS).

Звичайно з тою, або іншою ОС поставляються кілька оболонок. Як правило, програмісти працюють в одній оболонці, більш гнучкою і зручною для користування (наприклад csh або bash), а командні файли пишуть для іншої, більш простої (такої як Bourne). Оболонка, яка буде використовуватися по замовченню при реєстрації, та інша особиста інформація визначається в файлі /etc/passwd для кожного користувача окремо. Користувач може в який завгодно момент змінити вибір оболонки, що використовується при реєстрації (команда chsh). Основними оболонками різних версій UNIX є:

Bourne Shell	sh
Korn Shell	ksh
C Shell	csh
Bourne Again Shell	bash
Public Domain Korn Shell	pdksh
A Shell	ash
Tcl Shell	tclsh
X-Windows Shell	wish
Remote Shell	rsh

Деякі оболонки можуть бути присутніми в одній версії ОС і не бути представлені в іншій. Іноді оболонки сполучаються. Так, в деяких версіях Linux sh і bash - це одна і та ж програма (sh і bash є посиланнями на один і той ж файл). Незалежно від того, який командний інтерпретатор використовується, його основною задачею є надання інтерфейсу для користувача.

1.1.1 Командний інтерпретатор bash (bash)

Bash shell - один з найбільш популярних командних інтерпретаторів в системі Linux. Ця оболонка була розроблена на основі оболонок Bourne Shell і C Shell з додаванням функцій, що помітно полегшують роботу з Linux. Виклик оболонки відбувається по команді bash (файл /bin/bash або /usr/bin/bash). Показником того, що ви знаходитесь в bash, є системний запит "$".

1.1.2 Стандартні командні файли

Кожен раз при реєстрації користувача в системі виконується командний файл .bash_profile (аналог autoexec.bat і config.sys в DOS), який знаходиться в його домашньому каталозі. Файл .bash_profile являє собою файл ініціалізації командного інтерпретатора bash. Він виконується автоматично при кожному завантаженні оболонки. Даний файл містить команди, які визначають спеціальні змінні середовища, як системні, так і користувачеві. Розглянемо приклад стандартного файлу .bash_profile:

```
# .bash_profile                                    (1)
# Get the aliases and functions                    (2)
if [ -f ~/.bashrc ]; then                          (3)
        . ~/.bashrc                                (4)
fi                                                 (5)
# User specific environment and startup programs   (6)
PATH=$PATH:$HOME/bin                               (7)
ENV=$HOME/.bashrc                                  (8)
USERNAME=""                                        (9)
export USERNAME ENV PATH                           (10)
```

Рядки, що починаються зі знака # (1, 2, 6), трактуються як коментарі. Їх вміст, як правило, ігнорується.

В рядку 3 перевіряється, чи існує в домашньому каталозі користувача файл .bashrc. Якщо такий присутній, то виконується гілка умови then (рядок 4) - файл .bashrc запускається на виконання. Рядок 5 - кінець конструкції if-then.

Рядки 7-9 визначають змінні середовища. Зверніть увагу на те, що змінні PATH і HOME - це системні змінні, які система попередньо визначила сама. В даному файлі значення змінної PATH модифікується (розширюється). Спеціальні змінні та режими оболонки bash наведені в Додатку А.

Спеціальні змінні, крім усього іншого, потрібно експортувати з допомогою команди export. Це робиться для того, щоб вони стали доступними для всіх можливих вторинних оболонок. Однією командою export можна експортувати кілька змінних, перерахувавши їх в командному рядку як аргументи (рядок 10).

Ще одним файлом ініціалізації оболонки bash є .bashrc. Він виконується кожен раз, коли користувач входить в оболонку. Цей файл також запускається кожен раз при запуску якого завгодно командного файлу (скрипта). В .bashrc звичайно містяться визначення псевдонімів і змінних, які служать для включення тих чи інших функцій командного інтерпретатора. Приклад файлу .bashrc наведено нижче:

```
# .bashrc                                          (1)
# User specific aliases and functions              (2)
# Source global definitions                        (3)
if [ -f /etc/bashrc ]; then                        (4)
     . /etc/bashrc                                 (5)
fi                                                 (6)
set -o noclobber                                   (7)
alias l='/bin/ls -al'                              (8)
```

Як і в попередньому прикладі, в файлі використовується конструкція if-then (рядка 4-5). В ній наведене посилання на стандартний файл завантаження, єдиний для всіх користувачів системи - /etc/bashrc.

В рядку 7 установлюється режим noclobber. Він охороняє існуючі файли від запису поверх них переадресованої вхідної інформації. Можливі ситуації, коли в якості імені файлу, в який переадресується вивід, користувач може випадково вказати ім'я існуючого файлу. Якщо режим noclobber установлено, то при переадресації вхідної інформації в уже існуючий файл цей файл не буде замінено стандартнім вихідним потоком. Файл-оригінал збережеться.

Іноді при роботі доводиться часто використовувати одну і ту ж команду або послідовність команд. Цю проблему можна вирішити, створивши командний файл і вказавши каталог, в якому він знаходиться, в змінній оточення PATH. Однак не завжди раціонально зберігати окремий скрипт для кожної команди, що часто застосовується (надто багато файлів малої довжини). Для цього існують псевдоніми (aliases). Наприклад, однією з найчастіше

використовуваних команд є ls, але формат вихідних даних незручний (якщо команда запущена без опцій). Набагато зручніше і наочніше виглядає результат роботи команди /bin/ls -al. Але це неможлива розкіш - вводити такий довгий рядок кожен раз при необхідності довгого лістингу каталогу. В рядку 8 ми створюємо псевдонім. Псевдонім працює як макрос, який перетворює його в команду.

Крім файлів .bash_profile і .bashrc в домашньому каталозі користувача, як правило, ведеться протокол команд, введених користувачем раніше (.bash_history), і скрипт виходу з системи (.bash_logout).

1.1.3 Робота командного інтерпретатора в інтерактивному режимі

Коли користувач вводить команду на місці запиту ($), він передає її на обробку командному інтерпретатору. Інтерпретатор сприймає рядок команди як послідовність символів, в кінці якої знаходиться "повернення каретки" (Enter). Оболонка сприймає кілька типів команд: команди Linux системи, вбудовані команди інтерпретатора, команди, визначені користувачем, і команди-псевдоніми.

На свій розсуд, користувач може вводити команди по черзі, за принципом "один рядок - одна команда". Однак оболонка не накладає в цьому плані жодних обмежень. Дозволяється вводити по кілька команд в одному рядку, розділяючи їх крапкою з комою. Можливий випадок, коли команда не вміщується на один рядок - тоді можна сховати "повернення каретки" від оболонки, поставивши перед ним зворотну риску "\", і продовжувати ввід команди на наступному рядку. Таким чином, всі нижче приведені команди приведуть до однакових результатів:

```
1)
$ who; ps; echo JUNK MESSAGE
 ...                    (результат роботи who)
 ...                    (результат роботи ps)
JUNK MESSAGE            (результат роботи echo)
2)
$ who
...                     (результат роботи who)
$ ps
...                     (результат роботи ps)
$ echo JUNK MESSAGE
JUNK MESSAGE            (результат роботи echo)
```

```
3)
$ who; ps; echo JUNK \
>MESSAGE
...                         (результат роботи who)
...                         (результат роботи ps)
JUNK MESSAGE                (результат роботи echo)
```

Команда (або група команд), поміщена в дужки, виконується у вторинній оболонці (підоболонці). Підоболонка - це нова оболонка, що викликається поверх старої (первинної) оболонки. При цьому виконавчі команди вносять зміни тільки в цю підоболонку, не змінюючи параметрів первинного командного інтерпретатора (змінних, поточного каталогу і та ін.). Порівняйте:

```
[stud@localhost stud]$ cd /home
[stud@localhost home]$
                  ^^^^
i
[stud@localhost stud]$ (cd /home)
[stud@localhost stud]$
                  ^^^^
```

В другому випадку команда зміни поточного каталогу (cd /home) виконувалась в підоболонці, тобто поточний каталог змінився тільки для цієї підоболонки. Після виконання команди існування підоболонки закінчилось, і керування перейшло назад в первинну оболонку. Поточний каталог первинної оболонки залишився старим.

Іноді необхідно, щоб вихідні дані однієї команди слугували параметром (але не вхідним потоком!) для іншої. Для цього команду поміщають в зворотні лапки і ставлять на місці параметрів для зовнішньої команди. Наприклад:

```
$ elm `whoami`
```

Команда whoami повертає ім'я, під яким користувач зареєструвався в системі. Це ім'я підставляється в командний рядок в якості параметра для команди elm (посилка поштового повідомлення). Таким чином користувач посилає самому собі e-mail.

Оболонка bash веде історію введених з консолі команд. Проглянути її можна по команді history. Крім того, введені команди можна використовувати повторно. Найпростішими прикладами використання введених раніше команд є !! та !n.

!! остання введена з консолі команда (рядок)

!n n-а команда історії

!-n n-а команда історії, взятої в зворотному порядку (!-1 еквівалентно !!)

!str найостанніша команда з історії, що починається рядком "str"

Вихід з оболонки здійснюється по команді exit [expr]. Ця команда забезпечує вихід з поточної оболонки (командного інтерпретатора) з кодом expr. Вихід з оболонки також здійснюється при досягненні символу "кінець файлу" (Ctrl-D).

1.1.4 Командний інтерпретатор як процес

Спробуємо розглянути командний інтерпретатор більш формально. Будучи звичайною виконуваною програмою, оболонка є процесом. Як і кожен процес в системі UNIX, командний інтерпретатор має унікальний номер процесу, свої вхідні і вихідні потоки даних і всі інші атрибути процесу. Коли інтерпретатор виконується в інтерактивному режимі, вхідний і вихідний потоки асоційовані з терміналом - користувач вводить команди з клавіатури, результат виводиться на екран. Наприклад, в деякому каталозі знаходиться файл script з таким вмістом.

```
ps
echo end of script
```

Нагадаємо, що команда echo виводить повідомлення в стандартний вихідний файл (потік). Команда ps друкує в вихідний файл інформацію про процеси, які запустив користувач. Запустимо скрипт на виконання: по команді . (крапка) виконується командний файл, що передається як параметр.

```
[stud@localhost stud]$ . script
  PID TTY STAT TIME COMMAND
  269   1 S    0:00 /bin/login -- stud
  270   1 S    0:00 -bash
  360   1 R    0:00 ps
end of script
[stud@localhost stud]$
```

Як бачимо з результатів, на момент виконання команди ps з командного файлу script, в системі виконувалось одночасно три процеси, якими володіє користувач stud. Перший (під номером 269) - це процес підключення до системи (реєстрація). Другий (270) - це командний інтерпретатор. Третій (360) - це безпосередньо команда ps.

Скористуємося механізмом перенаправлення потоків:

```
[stud@localhost stud]$ bash < script > outfile
[stud@localhost stud]$ cat outfile
  PID TTY STAT TIME COMMAND
  269   1 S    0:00 /bin/login -- stud
  270   1 S    0:00 -bash
  361   1 S    0:00 bash
  362   1 R    0:00 ps
end of script
[stud@localhost stud]$
```

Опишемо ситуацію, що відбулася. Користувач stud, знаходячись в оболонці bash (процес 270), запускає нову оболонку bash (процес 361), вторинну по відношенню до першої (первинна помічена дефісом). Вхідним потоком нової оболонки є не термінал, а файл. Результат роботи перенаправлюється в інший файл. Команди файлу script, зокрема, команда ps (процес 362), виконуються в новій оболонці. Результат був би аналогічний, якщо б користувач просто викликав підоболонку, а потім послідовно ввів з клавіатури команди ps, echo і символ кінця файлу (Ctrl-D).

Вхідний потік для командного інтерпретатора являє собою послідовність лексем. Основними синтаксичними елементами (лексемами) вважаються.

1. Коментарі. Коментар починається з символу # і продовжується до кінця рядка. Для того, щоб запобігти інтерпретації знака # як початку коментарю, необхідно помістити його в лапки, або поставити перед ним зворотну похилу риску "\".

2. Пропускові символи. Під пропусками розуміють символи "пропуск" (#20h), "tab", "повернення каретки". Пропуски використовуються для відокремлення окремих слів в рядку.

3. Відокремлювачі між висловлюваннями. До таких відокремлювачів відносяться крапка з комою (;) і повернення каретки. Кілька команд можуть бути введені з одного рядка, відокремлені крапками с комою - це еквівалентно вводу кожної команди з окремого рядка. Деякі команди вимагають кілька рядків вводу (if або while).

4. Оператори. Оператор - це спеціальний символ або послідовність символів, за якою оболонка закріплює окремий синтаксичний зміст. Знаки пунктуації, що мають значення для оболонки, повинні бути сховані від неї в лапках, щоб не привести до їх невірного тлумачення.

5. Слова. Словом будемо називати яку завгодно послідовність символів, заключених між пропусковими символами, відокремлювачами і операторами. Словом може бути група послідовних символів, рядок в лапках, посилання на змінну, маска файлу, заміщена команда та ін. Словом може бути комбінація всього вищезазначеного. Кінцеве значення слова - це результат виконання всіх підстановок і замін, який разом зі звичайними символами формує рядок. Цей рядок інтерпретується оболонкою як команда і список параметрів, що передаються.

1.1.5 Шаблони і підстановки

В кожному командному інтерпретаторі реалізовано механізм підстановки імен файлів. При цьому використовуються наступні конструкції:

*	яка завгодно послідовність символів, включаючи порожню;
?	який завгодно символ. Кілька знаків питання означають яку завгодно послідовність символів заданої довжини;
[]	який завгодно символ з списку;
[^]	все що завгодно, крім символів списку;
{}	кожен елемент списку. При цьому не відбувається перевірка існування файлу (каталогу), а виконується безумовна підстановка, причому стільки разів, скільки елементів в списку;
~	домашній каталог.

Наприклад, нехай деякий каталог містить такі файли:
aaa, bbb, abc, cba, cccc

Наведемо приклади підстановки імен файлів в командний рядок:

Мета-послідовність	Результат підстановки
*	aaa abc bbb cba cccc
??a	aaa cba
*[b,c]	abc bbb cccc
[a-z]?a	aaa cba
*[^b,c]	aaa cba
{a,b,c}bb	abb bbb cbb

(при цьому не перевіряється, чи існують ці файли в дійсності)

1.1.6 Спеціальні символи (метасимволи)

Багато знаків пунктуації інтерпретуються оболонкою як службові. До них відносяться:

~ ` ! @ # $ % ^ & * ( ) \ | { } [ ] ; ' " < > ?

Для того, щоб не дати інтерпретатору обробляти метасимволи по-своєму, необхідно перед ними ставити зворотну косу риску "\", або поміщати необхідну лексему в прямі одинарні або подвійні лапки. Зворотна коса риска "ховає" від оболонки значущу характеристику наступного символу і заставляє обробляти його як простий символ ASCII. Дія подвійних і одинарних лапок практично однакова, але подвійні лапки допускають дію деяких спеціальних символів. Наприклад:

```
$ touch a\ strange\ file
```

В результаті цієї команді в поточному каталозі буде створено файл, в імені якого будуть присутніми два пропуски - "a strange file".

Необхідно розрізняти метасимволи в шаблонах команд (що передаються як аргументи) і метасимволи підстановки імен файлів. Вводячи команду з термінала, не забувайте, що спочатку в неї "загляне" командний інтерпретатор, а лиш потім - програма. Тому потрібно стежити, щоб оболонка не перехопила спеціальні символи, їй не назначені, і не проводила підстановку імен файлів. Яскравим прикладом може служити програма grep - пошук в файлах по зразку.

Якщо з термінала ввести

```
$ grep [A-Z]* chap[12]
```

то інтерпретатор підставить в командний рядок всі імена файлів, що відповідають шаблону. В результаті підстановки може статися так:

```
$ grep Array.c Bug.c Comp.c README chap1 chap2
```

Таким чином, утиліта grep буде виконувати пошук рядка "Array.c" в файлах Bug.c, Comp.c, README, chap1, chap2. Для того щоб передати команді grep метасимволи, застосовуються лапки:

```
$ grep "[A-Z]*" chap[12]
```

При цьому буде виконуватися пошук послідовності з нуля і більш великих латинських букв в файлах chap1 chap2.

1.1.7 Програмування в bash

Як уже зазначалось, командні інтерпретатори володіють деякими властивостями мов програмування, які дозволяють створювати досить складні програми. Така програма об'єднує

звичайно ряд команд UNIX, направлених на виконання конкретної задачі. Інструкції, з яких складається програма, вводяться в командний файл (скрипт, сценарій), який підлягає виконанню. Створити такий файл можна з допомогою звичайного текстового редактора.

1.1.8 Оператор "документ тут"

Оператор "документ тут" (<<) дозволяє використовувати рядки командного файлу в якості вхідних даних (вхідного потоку) для якої-небудь команди. Цим усувається необхідність читати вхідні дані з зовнішнього джерела, наприклад з файлу або каналу. Після оператора << на тому ж рядку слід помістити обмежувач потоку, а з наступного рядка - самі дані для вхідного потоку. Наприклад:

```
$ cat << zzz
>   This line will be printed
>zzz
   This line will be printed
$
```

1.1.9 Виконання наступної команди за умовою

Іноді в процесі роботи необхідно виконувати умовне розгалуження програми, або послідовності дій. В оболонці Сі подвійний амперсанд "&&" еквівалентний логічному "І". Команда, що стоїть після "&&", буде виконуватися тільки в тому випадку, якщо попередня команда завершилась успішно (код виходу 0). Наприклад:

```
% cp hello.c hello.bak && rm hello.c
```

Файл hello.c не буде видалено, якщо сталася помилка при копіюванні, тобто якщо команда cp завершилась невдало.

"||" - еквівалент логічного "АБО". Команда, що стоїть після "||", буде виконуватися тільки в тому випадку, якщо попередня команда завершилась невдало (код виходу відрізняється від 0). Наприклад:

```
% cp hello.c hello.bak || echo Copy file error
```

Якщо пройшла помилка при копіюванні, то буде надруковано повідомлення Copy file error.

1.1.10 Заміна оболонки новою програмою - команда exec

Команда exec замінює поточний процес (командний файл, що виконується, або оболонку) новою задачею, ім'я якої передається в

якості параметра. Команда часто використовується при написанні скриптів для організації процесів, що повторюються.

1.1.11 Визначення і розрахунок змінних

Командний інтерпретатор bash володіє можливостями роботи зі змінними. Імена змінних не обмежені по довжині і можуть містити великі і малі букви латинського алфавіту, цифри і знак підкреслювання. Ім'я змінної не може починатися с цифри.

Значення змінної присвоюється за допомогою оператора присвоювання (=). Оператор присвоювання пропусками не відділяється. Змінній може бути присвоєна яка завгодно сукупність символів. Наприклад:

```
$ greeting="How do you do?"
```

Значення змінних часто використовуються як аргументи команд. Розрахунок (підстановка) значення змінної відбувається за допомогою оператора $. Результатом розрахунку є набір символів. Цей набір замінює ім'я змінної в командному рядку. Наприклад:

```
$ echo $greeting
How do you do?
```

Список всіх визначених змінних можна отримати по команді set. Якщо яка-небудь змінна більше не потрібна, її можна видалити командою unset.

1.1.12 Ввід і вивід даних в сценаріях

Для виводу даних в сценарії можна використовувати команду echo, а для зчитування вхідної інформації в змінні - команду read. Фактично команда read читає рядок зі стандартного вводу. Все, що посилається на стандартний ввід з клавіатури (файлу або каналу при переадресації) – аж до символу нового рядка – зчитується і присвоюється в якості значення змінної. Наприклад:

```
$ read greeting
How do you do
$ echo $greeting
How do you do
```

Крім того, за допомогою конструкції "документ тут" в сценарій можна вводити дані і переадресовувати їх в команду.

1.1.13 Аргументи командного рядка

Аналогічно тому, як це робиться в командах UNIX, в скриптах можна використовувати аргументи. Аргументи вводяться при виклику командного файла після його імені і нумеруються, починаючи з 1. Перший аргумент програми, що оброблюється, позначається $1, другий - $2, і т.ін. Аргумент $0 - це ім'я програми, що виконується на даний момент (назва оболонки, або командного файлу) - фактично перше слово в командному рядку. Аргументи можуть розглядатися як локальні змінні процесу. Існують і інші спеціальні змінні, пов'язані з характеристиками командного рядка і процесу (див. Додаток Б).

1.1.14 Арифметичні операції

В оболонці bash присутня команда let, яка дозволяє виконувати операції з арифметичними величинами. За допомогою цієї команди можна порівнювати числа і виконувати з ними такі операції, як додавання або множення. Команда let може замінюватися подвійними круглими дужками. Оператори let, що співпадають зі спеціальними символами оболонки, брати в лапки не потрібно. Якщо операнди арифметичного виразу розділені пропусками, цей вираз слід взяти в лапки. Наприклад:

```
$ let "res = 2 * 7"
$ echo $res
14
$
```

1.1.15 Команда порівняння test

Часто буває необхідно виконати перевірку, в ході якої порівнюються дві величини або перевіряється наявність того або іншого файлу. За допомогою команди test можна порівнювати цілі числа, рядки, і виконувати логічні операції. Результат перевірки - це код завершення операції test, який, як було сказано раніше, зберігається в спеціальній змінній $?. Замість ключового слова test можна використовувати квадратні дужки. Наприклад:

```
$ greeting="hallo"
$ num=5
$ test $num -eq 5 ; echo $?
0
$ [ $greeting = "hallo" ] ; echo $?
0
```

```
$ test -f main.c ; echo $?
1
```

В даному прикладі ми ініціалізуємо дві змінні. Потім виконуємо перевірки на рівність їх тому або іншому значенню (зверніть увагу, що рядки порівнюються за допомогою оператора =, а численні значення - за допомогою опції -eq). Остання перевірка - це перевірка наявності файлу main.c в поточному каталозі. Результат – неправда (файл не знайдено). Операції команди test наведені у Додатку В.

1.1.16 Умови

В оболонці bash є набір умовних керуючих структур, які забезпечують умовне розгалуження програм. Багато з цих структур аналогічні умовним керуючим структурам мов програмування, але є деяка різниця.

Конструкція if ставить умову для виконання команди. Цією умовою є код завершення якоїсь конкретної команди. Якщо команда виконана успішно (код завершення дорівнює нулю), то команди всередині структури if виконуються. В іншому випадку виконується гілка else (якщо вона присутня) або керування передається наступному за if-конструкцією оператору. Керуюча структура if повинна закриватися ключовим словом fi (if навпаки). Синтаксис if-конструкції такий:

```
if <команда-умова>
     then
          <команди>
     else
          <команди>
fi
```

В якості команди-умови як правило використовується команда test або її альтернативна форма - квадратні дужки [].

Вкладення умов if здійснюється за допомогою структури elif. Вкладеність elif не обмежується. Остання гілка каскаду elif повинна починатися зі слова else:

```
if <команда-умова>
     then
          <команди>
     elif
          <команди>
     else
          <команди>
fi
```

Керуюча структура case забезпечує вибір одного з кількох можливих варіантів. Вибір здійснюється шляхом порівняння заданого в структурі значення з кількома можливими зразками. Кожне можливе значення змінної, що перевіряється (зразок) пов'язується з сукупністю операцій. Кожен зразок являє собою регулярний вираз, що завершується круглою дужкою. Список команд, що виконуються завершується двома крапками з комами, що стоять на окремому рядку. Вся конструкція завершується ключовим словом esac (case навпаки). Синтаксис структури:

```
case <рядок> in
     зразок)
          команди
          ;;
     зразок)
          команди
          ;;
     *)
          команди по замовчуванню
          ;;
esac
```

Зразок може містити спеціальні символи: *, [], ?, |. Варіант по замовчуванню включати до структури не обов'язково.

1.1.17 Цикли

Командний інтерпретатор дозволяє створювати гнучкі циклічні структури. Конструкція while забезпечує виконання окремої команди, доки виконується умова (код виконання команди-умови дорівнює нулю). Замикає конструкцію ключове слово done:

```
while <команда-умова>
     do
          команди
     done
```

Як і в if-структурі, команда-умова частіше всього представляється перевіркою test (або квадратними дужками).

Цикл until аналогічний циклу while. Він відрізняється тим, що команди тіла циклу виконуються до того часу, доки умова залишається НЕ виконаною.

```
until <команда-умова>
     do
          команди
     done
```

Структура for-in призначена для послідовного звернення до значень, перерахованих у списку. В ній два операнди - змінна і список значень. Кожне зі значень по черзі присвоюється змінній структури. Цикл завершується, коли всі значення зі списку будуть вичерпані. Тіло циклу поміщується в операторні дужки do-done, синтаксис конструкції for-in:

```
for <змінна> in <список_значень>
     do
          команди
     done
```

Структура for без явно заданого списку значень використовує в якості такого аргументи командного рядка. При першому проході змінній присвоюється значення першого аргументу командного рядка, при другому - значення другого, та ін.

1.1.18 Приклади скриптів

Розглянемо простий інтерактивний скрипт whoareyou. Нижче наводиться його текст:

```
echo What is your name?                                  (1)
read reply                                               (2)
case $reply in                                           (3)
  root)                                                  (4)
    echo You are system administrator of $HOSTNAME       (5)
    ;;                                                   (6)
  stud)                                                  (7)
    echo You are a student                               (8)
    ;;                                                   (9)
  *)                                                     (10)
    echo You are a mortal user on $HOSTNAME              (11)
    ;;                                                   (12)
esac                                                     (13)
```

Скрипт пропонує користувачеві відрекомендуватися (рядок 1). Користувач вводить своє ім'я, яке заноситься в змінну reply. Далі відбувається розгалуження в залежності від значення reply (іншими словами, в залежності від відповіді користувача). Скрипт друкує одне з трьох повідомлень. При друку використовується значення іншої змінної - HOSTNAME. Вона визначається безпосередньо системою і містить мережеве ім'я UNIX-машини, на котрій виконується скрипт.

Розглянемо інший приклад. Іноді, при великій кількості файлів, буває важко за ними услідити. Наведена нижче програма chkown дозволяє знаходити файли, які не належать користувачеві.

```
for filename in `ls`                                    (1)
do                                                      (2)
   if [ ! -O $filename ]; then                          (3)
      echo $filename does not belong to `whoami`        (4)
   fi                                                   (5)

   if [ -d $filename -a -x $filename ]; then            (6)
      echo Entering $filename ...                       (7)
      cd $filename                                      (8)
      $0                                                (9)
      Echo Leaving $filename ...                        (10)
      cd ..                                             (11)
   fi                                                   (12)
done                                                    (13)
```

Розглянемо дію скрипта порядково.

В першому рядку ініціалізується for-цикл. Змінною циклу є filename. Список значень визначається результатом роботи команди ls (вона поміщена в зворотні лапки). Іншими словами, змінна filename по черзі приймає значення імен файлів і каталогів поточної директорії.

В рядку 3 відбувається перевірка, чи належить файл або каталог користувачеві, який запустив скрипт (операція -O команди test). Зауважимо, що якщо ключове слово then стоїть на тому ж рядку, що і умова, воно повинно відокремлюватися крапкою з комою (;). Якщо користувач не є хазяїном файлу (знак оклику - логічне заперечення), то друкується повідомлення (рядок 4). При виконанні рядка 4 замість `whoami` в команду echo підставляється ім'я користувача (зворотні лапки).

В рядку 6 відбувається подвійна перевірка: чи є значення змінної filename каталогом (операція -d), і чи є значення змінної filename файлом, виконавчим для користувача (-x). Обидва логічних результата перемножуються (операція -a відповідає логічному "І") і, згідно кінцевому результату, гілка then виконується або не виконується. Як відомо, права на виконання каталогу визначають можливість користувача увійти в нього (cd).

Якщо змінна filename дійсно містить ім'я доступного каталогу, скрипт входить в нього (рядок 8) і рекурсивно запускає самого себе уже в цьому каталозі (в рядку 9 змінна $0 відповідає імені виконавчої програми, тобто імені скрипта). Після завершення роботи цього другого скрипта, керування повертається в первинний процес,

відбувається перехід в початковий каталог (рядок 11), і на цьому гілка then завершується.

Необхідно відзначити, що якщо шлях до даного скрипта не буде визначено в змінній PATH, то його виконання буде перерване вже на другому рівні, оскільки програма вже покине початковий каталог (каталог першого рівня), в якому знаходиться файл chkown. Команда в рядку 9 не містить абсолютного імені файлу, тому скрипт звернеться до змінної PATH і почне його пошук у всіх визначених там каталогах.

1.2 Завдання до роботи

1.2.1 Ознайомитися с можливостями і принципами роботи командного інтерпретатора bash shell.

1.2.2 Оволодіти початковими навичками написання командних файлів (скриптів).

1.2.3 Отримавши номер варіанту у викладача, написати і відлагодити командний файл згідно завдання:

Варіант 1

Написати скрипт, що посилає всім користувачам, що знаходяться в даний момент в системі, яке-небудь повідомлення (електронною поштою або безпосередньо на екран). Прикладом повідомлення може бути поточна дата і час.

Варіант 2

Написати і відлагодити скрипт, який в домашньому каталозі користувача і в нижчеслідуючих підкаталогах знаходить найдовший файл, а потім визначає його тип.

Варіант 3

Написати скрипт, який в домашньому каталозі і підкаталогах користувача підраховує кількість файлів, що містять тексти вихідних програм на Сі.

Варіант 4

Написати скрипт, який в домашньому каталозі і підкаталогах знаходить вихідні тексти програм на Сі і виводить на екран імена всіх файлів-заголовків (stdio.h, stdlib.h, iostream.h, і т.ін.) що згадуються в них.

Варіант 5

Написати скрипт, який розраховує максимальну глибину дерева каталогів файлової системі.

1.2.4 Скласти звіт про пророблену роботу. Звіт повинен містити тему і мету роботи, тексти вихідних командних файлів, роздруківку повідомлень програми, висновки.

1.3 Домашнє завдання

1.3.1 Використовуючи методичні вказівки і конспект лекцій, ознайомитися з теоретичними відомостями про командні інтерпретатори.

1.3.2 Вивчити призначення, формат і дію команд, аргументів, операторів, спеціальних символів, змінних і конструкцій командного інтерпретатора bash shell.

1.4 Контрольні питання

1.4.1 Командний інтерпретатор, його основні функції.

1.4.2 Стандартні командні файли, їх вміст і послідовність виконання.

1.4.3 Локальні і глобальні змінні. Схожість і різниці змінних і псевдонімів.

1.4.4 Робота інтерпретатора в інтерактивному режимі. Історія (протокол) введених з консолі команд.

1.4.5 Лексеми оболонки.

1.4.6 Шаблони і підстановки. Пріоритети виконання (підстановки) спеціальних символів.

1.4.7 Умовні конструкції bash.

1.4.8 Команда exec. Приклад використання.

1.4.9 Робота з аргументами командного рядка.

1.4.10 Команда test. Її оператори і операнди.

1.4.11 Циклічні конструкції в командних файлах.

2 ЛАБОРАТОРНА РОБОТА №2

Системне адміністрування Linux

Мета роботи: освоєння програмного забезпечення, призначеного для заведення і видалення користувача і групи користувачів, зміни пароля користувача, зміни облікових записів про користувача і групу. Пакет sudo. Монтування файлових систем.

2.1 Стислі теоретичні відомості

Основними задачами системного адміністрування є:
- підключення і видалення користувачів;
- підключення і видалення апаратних засобів;
- резервне копіювання;
- установка нового програмного забезпечення;
- моніторинг системи;
- пошук несправностей;
- ведення локальної документації;
- контроль захисту;
- надання допомоги користувачам.

В даній лабораторній роботі будуть частково розглянуті пункти 1, 2 і 8 даного списку.

2.1.1 Заведення і видалення користувачів

Інформація про всіх користувачів системи Unix зберігається в файлі /etc/passwd. Детально структура цього файлу описана в секції 5 розділу passwd "оперативної інструкції користувача". Заведення нового користувача зводиться до внесення нового запису в цей файл. Однак, ідея самостійного внесення реєстраційного запису в цей файл за допомогою якого-небудь текстового редактора, не дивлячись на досить прозору структуру цього файлу, не є плідною. Не будемо зупинятися на можливості внесення простих синтаксичних помилок (людині властиво помилятися), через які даний обліковий запис буде просто ігноруватися. Також можливо вас не зупинить і те, що ви не бажаючи того порушите логічну цілісність даного файлу, що призведе до дірок в захисті вашої системи. Просто подумайте над тим, що станеться, якщо два адміністратори одночасно почнуть редагувати цей

файл, внесуть зміни і доповнення, але ваш колега збережеться на пару секунд пізніше.

Для заведення нового користувача в Linux призначені наступні утиліти:

- useradd (пакетна утиліта);

- adduser (інтерактивна утиліта).Програма призначена для роботи на алфавітно-цифрових терміналах. В режимі діалогу запитується вся необхідна інформація, після чого викликається утиліта useradd;

- glint (графічна утиліта), аналог User Manager Windows NT. В кінцевому підсумку також звертається до useradd.

Для видалення користувача призначена утиліта userdel. Для заведення і видалення груп користувачів призначені утиліти groupadd і groupdel. Змінити обліковий запис користувача можна утилітою usermod, для групи користувача існує утиліта groupmod. Всі ці утиліти, а також деякі інші, входять в пакет shadow.

2.1.2 Утіліта useradd

Розглянемо детальніше утіліту useradd:

USERADD

Section: Maintenance Commands (8)

NAME

useradd - створення нового користувача або зміна інформації для заведення нового користувача

useradd

[**-c** comment] [**-d** home_dir]
[**-e** expire_date] [**-f** inactive_time]
[**-g** initial_group] [**-G** group[,...]]
[**-m** [**-k** skeleton_dir]] [**-s** shell]
[**-u** uid [**-o**]] login

useradd

-D [**-g** default_group] [**-b** default_home]
[**-f** default_inactive] [**-e** default_expire_date]
[**-s** default_shell]

2.1.3 Заведення нових користувачів

При виклику без ключа -D команда useradd створює обліковий запис нового користувача, використовуючи значення, що визначені в командному рядку і значення по замовчуванню з системи. В залежності від ключів командного рядка при необхідності буде внесено обліковий запис в облікові файли, створено домашній каталог, а також скопійовані ініціалізаційні файли. Ключі, які можуть бути передані команді useradd:

-c comment

Вміст поля коментарю файлу паролів для користувача, що створюється.

-d home_dir

Новий користувач буде створений з використанням home_dir в якості значення домашнього каталогу. По замовчуванню реєстраційне ім'я login додається до default_home і отримане значення використовується як ім'я домашнього каталога.

-e expire_date

Дата блокування користувача. Дата задається в форматі MM/DD/YY.

-f inactive_days

Число днів після спливання строку дії пароля до блокування користувача. 0 блокує користувача зразу ж після спливання строку дії пароля, -1 відключає дану можливість. По замовчуванню використовується значення -1.

-g initial_group

Ім'я або номер початкової групи користувача. Група повинна існувати. Номер групи повинен посилатися на вже існуючу групу. Номер групи по замовчуванню 1.

-G group,[...]

Список додаткових груп, членом яких, також, є користувач. Групи відокремлюються комами, без пропускових символів. На групи накладаються ті ж обмеження, що і на групу, задану ключем -g. По замовчуванню користувач належить тільки до початкової групи.

-m

Створити домашній каталог користувача, якщо він не існує. При заданні ключа -k файли, що знаходяться в каталозі

skeleton_dir, будуть скопійовані в домашній каталог, інакше будуть використані файли з каталогу /etc/skel. Також всі каталоги, що містяться в skeleton_dir або /etc/skel, будуть створені в домашньому каталозі користувача. Ключ -k припустимий лише сумісно з ключем -m. По замовчуванню домашній каталог не створюється і ніякі файли не копіюються.

-s shell

Найменування реєстраційного командного інтерпретатора користувача. По замовчуванню це поле залишається порожнім, що примушує систему вибрати реєстраційний командний інтерпретатор по замовчуванню.

-u uid

Числове значення ідентифікатора користувача. Значення повинно бути унікальним, в випадку якщо не задано ключ -o. Значення повинно бути невід'ємним. По замовчуванню використовується найменший ідентифікатор, більший 99 і більший ніж ідентифікатор якого завгодно іншого користувача. Величини між 0 і 99 звичайно зарезервовані для системних облікових записів.

2.1.4 Зміна значень по замовчуванню

При виклику з ключем -D useradd покаже поточні значення по замовчуванню, або замінить значення по замовчуванню відповідними значеннями з командному рядка. Дозволеними ключами є:

-b default_home

Початкова частина для домашнього каталогу користувача. При створенні нового облікового запису для отримання імені домашнього каталогу користувача ім'я користувача додається в кінець default_home, за виключенням випадку, коли каталог користувача задано ключем -d.

-e default_expire_date

Дата блокування користувача.

-f default_inactive

Число днів після спливання строку дії пароля до блокування користувача.

-g default_group

Ім'я або номер початкової групи користувача. Група повинна існувати. Номер групи повинен посилатися на вже існуючу групу.

-s default_shell

Найменування реєстраційного комп'ютера і адміністратор відповідальний за розміщення користувальницьких файлів по замовчуванню в каталозі /etc/skel. Першими кандидатами для розміщення там є файл .inputrc і каталог .mc з настройками для роботи з кирилицею.

27.1.5 Неприємності

Ви не можете додати користувача в групу NIS. Ця операція повинна проводитись на сервері NIS.

2.1.6 Файли

/etc/passwd - файл облікових записів.
/etc/shadow - файл тіньових паролів.
/etc/group - інформація о групах користувачів.
/etc/default/useradd - інформація по замовчуванню.
/etc/skel - каталог, що містить файли по замовчуванню.

2.1.7 Монтування файлових систем

Файлове дерево формується з окремих частин, званих файловими системами. Для того, щоб зробити файлову систему доступну для процесів Unix, її треба змонтувати. Точкою монтування файлової системи може служити який завгодно каталог. Його файли і каталоги будуть недосяжні, доки файлова система змонтована поверх них. Файлові системи прикріплюються до файлового дерева з допомогою команди mount. Ця команда бере з існуючого файлового дерева каталог, званий точкою монтування, і робить його кореневим каталогом що приєднується до файлової системи. Наприклад команда

$ mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom

монтує пристрій CD-ROM в каталог /mnt/cdrom в режимі "тільки читання".

Демонтуються файлові системи з допомогою команди umount. Демонтувати файлову систему можна лише тоді, коли в ній нема

відкритих файлів і процесів, що використовують цю файлову систему (тобто файлова система не повинна бути зайнятою). Якщо файлова система, що демонтується, містить виконавчі програми, то вони не повинні бути запущені. Для визначення процесів, які використовують файлову систему, під Linux існує програма fuser.

2.1.8 Пакет sudo

На практиці часто виникає ситуація, коли для виконання своїх обов'язків деяким користувачам необхідно надати привілеї, які доступні звичайно тільки root'у. Існує кілька способів доступу до бюджету привілейованого користувача. Найпростіший з них - зареєструватися під іменем root. Але на жаль, вихід з власного бюджету і реєстрація в якості привілейованого користувача часто дуже незручні. Краще використовувати команду su. Будучи викликаною без аргументів, ця команда запросить вас ввести пароль привілейованого користувача, а потім запустить shell з відповідними правами. Привілеї цього інтерпретатора команд залишаються в силі до завершення його роботи.

З привілейованим користувальницьким доступом сполучені три проблеми: безмежні повноваження, відсутність обліку операцій, що виконуються, імовірність того, що під іменем root може працювати група користувачів. Оскільки повноваження привілейованого користувача розподілити не можна, то важко надати комусь можливість зняття резервних копій (що повинно робитися під ім’ям root), не даючи можливості вільної роботи в системі. Якщо ж бюджет root доступний групі користувачів, то ви і поняття не будете мати про того, хто їми користується і що робить.

Щоб вирішити ці проблеми, використовується програма sudo. Ця програма в якості аргументу приймає командний рядок, який підлягає виконанню з правами root. Команда sudo звертається до файлу /etc/sudoers, що містить список користувачів, що мають повноваження на її виконання, і перелік команд, які вони мають право виконувати на конкретній машині. Якщо команда, що пропонується, дозволена, sudo пропонує користувачеві ввести його власний пароль і виконує команду як root.

До спливання п'ятихвилинного періоду бездіяльності sudo можна виконувати інші sudo-команди, не вводячи пароля. Така міра -

захист від тих користувачів з sudo-привілеями, які кидають свої термінали без нагляду.

Файл /etc/sudoers виглядає приблизно так:

```
# Host alias specification Host_Alias
HUB=houdini.rootgroup.com:\
REMOTE=merlin,kodiakthorn,spirit
Host_Alias MACHINES=kalkan,alpo,milkbones
Host_Alias SERVERS=houdini,merlin,kodiakthorn,spirit
# Command alias specification
Cmnd_Alias  LPCS=/usr/etc/lpc,/usr/ucb/lprm
Cmnd_Alias  SHELLS=/bin/sh,/bin/csh,/bin/tcsh
Cmnd_Alias  SHUTDOWN=/etc/halt,/etc/shutdown
# User specification
Britt       REMOTE=SHUTDOWN:ALL=LPCS
Robh        ALL=ALL,!SHELLS
nieusma     SERVERS=SHUTDOWN,/etc/reboot:\,HUB=ALL,!SHELLS
jill        houdini.rootgroup.com=/etc/shutdown,MISC
markm       HUB=ALL,!MISC,!/etc/shutdown,!/etc/halt
billp       ALL=/usr/local/bin/top:MACHINES=SHELLS
davehieb    merlin=ALL:SERVERS=/etc/halt:\
kodiakthorn=ALL
```

В цьому прикладі користувачеві britt дозволено виконувати програми /etc/halt, /etc/shutdown на машинах merlin, kodiakthorn і spirit. robh може виконувати які завгодно програми, крім перерахованих в макрозмінній SHELLS, на всіх машинах.

Що можуть робити користувачі jill, markm, billp і davehieb і на яких машинах, ви розкажете самі, вивчивши документацію по пакету sudo. Також в документації перераховані деякі додаткові можливості, які з'явились в цій програмі, наприклад, можливість виконання програми, що вказана, не від root'а, а від іншого користувача.

Для модифікації файлу /etc/sudoers використовується програма visudo, яка дозволяє редагувати цей файл, перевіряючи синтаксис заданих змін. Зверніть увагу, що всі команди в цьому файлі задаються з абсолютними шляхами, щоб попередити можливість виконання користувальницьких програм з тими ж іменами з правами root.

Крім виконання вказаних команд, sudo веде файл реєстрації виконаних команд, осіб, що їх викликали, каталогів, з яких викликались програми і час їх виклику.

2.2 Завдання до роботи

2.2.1. Вивчити документацію по пакетам shadow, sudo.

2.2.2. Завести групу користувачів stud.

2.2.3. Завести користувача stud, первинною групою якого є stud.

2.2.4. Встановити пароль користувачеві stud.

2.2.5. Додати користувача stud в групу floppy.

2.2.6 Реалізувати на shell сценарій монтування ГМД з файловою системою vfat. За допомогою програми visudo дозволити користувачеві stud виконувати цей сценарій з привілеями користувача root.

2.2.7 Реалізувати на shell сценарій видалення із каталогу /usr/tmp усіх файлів до яких не звертались більше 10 діб. За допомогою програми visudo дозволити користувачеві stud виконувати цей сценарій з привілеями користувача root. Рекурсивно видалити користувача stud (разом з його домашнім каталогом).

2.2.8. Видалити групу stud.

2.3 Контрольні питання

2.3.1. Для чого необхідно включати користувачів в групу floppy?

2.3.2. Кому будуть належати файли користувача stud після видалення відповідного облікового запису?

2.3.3. Яким чином можна дозволити користувачам монтувати деякі файлові системи без використання пакета sudo?

2.3.4. Які типи файлових систем підтримує система Linux?

2.3.5. Створіть файл a, після чого файли b і z, що є жорсткими посиланнями на файл a. Чому при зміні режиму доступу файлу a змінились режими доступу файлів b і z?

2.3.6 Монтування різних файлових систем та пристроїв.

2.3.7. Яким чином користувач може змінити початковий командний інтерпретатор і поле коментарю свого облікового запису з допомогою системного адміністратора?

3 ЛАБОРАТОРНА РОБОТА №3

Прийом і передача даних з використанням інтерфейсів I2C та SPI

Мета роботи: Освоїти методи передачи та прийому даних по послідовному інтерфейсу I2C та SPI з використанням апаратної обчислювальної платформи Arduino.

3.1 Стислі теоретичні відомості

3.1.1 Апаратна обчислювальна платформа Arduino

Основними компонентами Arduino [1] є плата вводу/виводу та середовище розробки на мові Processing/Wiring. Arduino може використовуватися як для створення автономних інтерактивних об'єктів, так і підключатися до програмного забезпечення, яке виконується на комп'ютері (наприклад: Adobe Flash, Processing, Max/MSP, Pure Data, SuperCollider).

Апаратна частина. Плата Arduino складається з мікроконтролера Atmel AVR, а також елементів обв'язки для програмування та інтеграції з іншими пристроями. На багатьох платах наявний лінійний стабілізатор напруги +5В або +3,3В. Тактування здійснюється на частоті 16 або 8 МГц кварцовим резонатором. У мікроконтролер записаний завантажувач (bootloader), тому нема потреби у зовнішньому програматорі.

На концептуальному рівні усі плати програмуються через RS-232 (послідовне з'єднання), але реалізація даного способу різниться від версії до версії. Новіші плати програмуються через USB, що можливо завдяки мікросхемі конвертера USB-to-Serial FTDI FT232R. У версії платформи Arduino Uno в якості конвертера використовується контролер Atmega8 у SMD-корпусі. Дане рішення дозволяє програмувати конвертер таким чином, щоб платформа відразу розпізнавалася як миша, джойстик чи інший пристрій за вибором розробника зі всіма необхідними додатковими сигналами керування. У деяких варіантах, таких як Arduino Mini або неофіційній Boarduino, для програмування потрібно підключити до контролера окрему плату USB-to-Serial або кабель.

Плати Arduino дозволяють використовувати значну кількість I/O виводів мікроконтролера у зовнішніх схемах. Наприклад, у платі

Decimila доступно 14 цифрових входів/виходів, 6 із яких можуть видавати ШІМ сигнал, і 6 аналогових входів. Ці сигнали доступні на платі через контактні площадки або штирьові розніми. Також існує декілька видів зовнішніх плат розширення, які називаються "shields" ("щити"), які приєднуються до плати Arduino через штирьові розніми.

Програмне забезпечення. Інтегроване середовище розробки Arduino це багатоплатформовий додаток на Java, що включає в себе редактор коду, компілятор і модуль передачі прошивки в плату. Середовище розробки базується на мові програмування Processing та спроектоване для програмування новачками. Мова програмування аналогічна мові Wiring. Строго кажучи, це C++, доповнений деякими бібліотеками. Програми обробляються за допомогою препроцесора, а потім компілюється за допомогою AVR-GCC.

Програми Arduino пишуться на мові програмування C або C++. Середовище розробки Arduino поставляється разом із бібліотекою програм, яка називається "*Wiring*", яка бере початок від проекту Wiring, який дозволяє робити багато стандартних операцій вводу/виводу набагато простіше. Користувачам необхідно визначити лише дві функції, для того щоб створити програму, яка буде працювати за принципом циклічного виконання:

- setup(): функція виконується лише раз при старті програми і дозволяє задати початкові параметри;
- loop(): функція виконується періодично доки плата не буде вимкнена.

Приклад 1. Розглянемо типову програму для мікроконтролеру, яка посилає команду блимати світловому діоду в середовищі Arduino [2]. Для цього потрібні наступні апаратні засоби:

- плата Arduino або Genuino
- світловий діод
- резистор 220 Ом.

Для побудови схеми, один кінець резистора підключають до 13-го контакта Arduino,а довгу ніжку світлодіода (анод) на інший кінець резистора. Коротку ніжку світлодіода (катод) підключають до Arduino GND (земля), як показано на на рис. 3.1.

Програма.

```
#define LED_PIN 13
void setup () // обов'язкова процедура, що запускається на початку програми
```

```
{
  pinMode (LED_PIN, OUTPUT); // оголошення використовуваного порту, led - номер порту, другий аргумент - тип використання порту - на вхід (INPUT) або на вихід (OUTPUT)
}
void loop () //обов'язкова процедура loop, що запускається циклічно після процедури setup
 {
  digitalWrite (LED_PIN, HIGH); // Включити світлодіод - ця команда використовується для включення або виключення напруги на цифровому порту; led - номер порту, другий аргумент - включення (HIGH) або вимикання (LOW)
 delay (1000); // Зачекати одну секунду (1000 мілісекунд)
 digitalWrite (LED_PIN, LOW); // Вимкнути світлодіод
 delay (1000); // Зачекати одну секунду
}
```

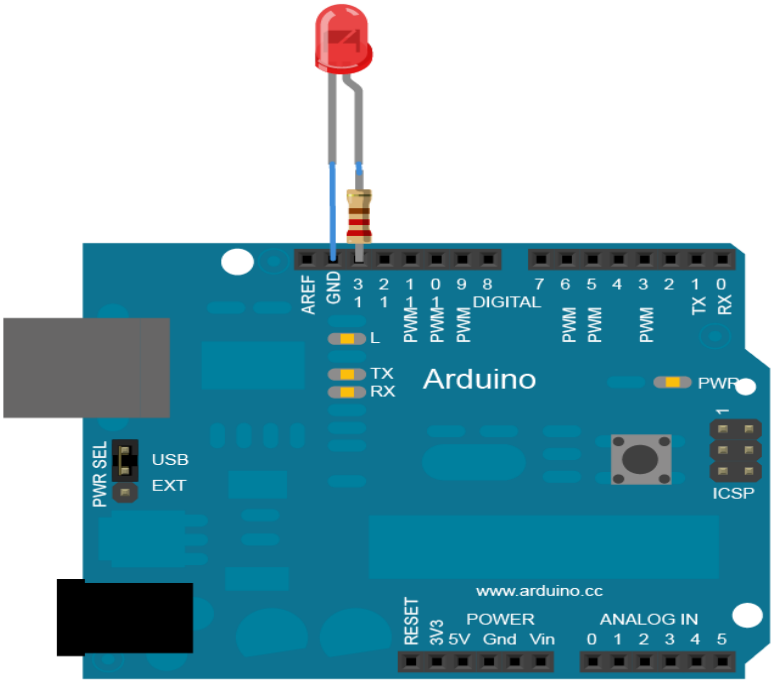

Рисунок 3.1 – Схема підключення світлодіоду на Arduino

3.1.2 Ультразвуковий далекомір PING)))

PING))) - це ультразвуковий далекомір від компанії Parallax. Він визначає дистанцію до найближчого об'єкта, який знаходиться перед ним (пошуковий діапазон від 2 см до 3 м). Принцип його роботи - він відсилає пучок ультразвукових хвиль, а потім чекає відлуння, що створюваного «відпружинюванням» цього сигналу від об'єкту, що знаходиться попереду. Arduino відсилає віддаленіру короткий імпульс, щоб запустити процес виявлення, а потім через той же контакт чекає імпульсу за допомогою функції *pulseIn*(). Тривалість відповідного імпульсу еквівалентна часу, який йде на те, щоб ультразвук дійшов до об'єкта і повернувся до датчика. За допомогою швидкості звуку цей час можна конвертувати в дистанцію.

Приклад 2. Розглянемо програму для мікроконтролеру, яка зчитує дані далекоміра PING))), а потім повертає дані про дистанцію до найближчого об'єкта [3]. Апаратні засоби:

- плата Arduino;
- ультразвуковий далекомір PING)));
- провода-перемички.

Електрична схема підключення PING))) до Arduino наведена на рис. 3.2. 5-вольтний контакт PING))) підключений до 5-вольтового контакту на Arduino, а контакт з «землею» на PING))) з'єднаний з «землею» на Arduino. Контакт SIG (тобто сигнал) на PING))) приєднаний до сьомого цифровому контакту на Arduino. Результатом є ланцюг наведений на рис.3.3.

Програма.

```
const int pingPin = 7; // Задали контакт для датчика
void setup() {
// Ініціалізуємо послідовну передачу даних:
Serial.begin(9600);
      }
void loop(){
```

// Задаємо змінні для тривалості імпульс,а також підсумкову дистанцію - в дюймах і сантиметрах:

```
long duration, inches, cm;
```

// PING))) активується імпульсом HIGH тривалістю 2 або більше μs

// Перед цим дамо короткий імпульс LOW, щоб «змити» імпульс HIGH (якщо той залишився з минулого разу):

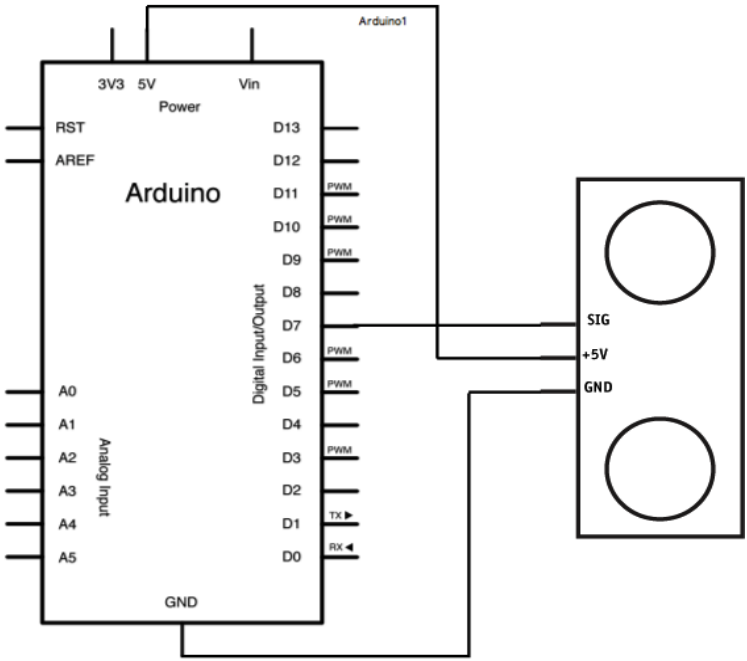

Рисунок 3.2 – Електрична схема підключення PING))) до Arduino

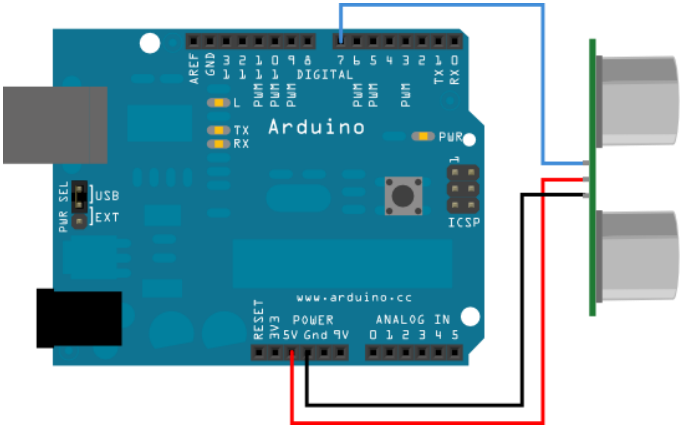

Рисунок 3.3 – Ланцюг підключення PING)))до Arduino

```
    pinMode(pingPin, OUTPUT);
    digitalWrite(pingPin, LOW);
    delayMicroseconds(2);
    digitalWrite(pingPin, HIGH);
    delayMicroseconds(5);
    digitalWrite(pingPin, LOW);
// Скористаємося тим же контактом для зчитування
сигналу від PING)))
// Тривалість сигналу HIGH - це і є час (в µs)від
відправки ультразвукових хвиль до отримання відлуння
//від об'єкту
    pinMode(pingPin, INPUT);
    duration = pulseIn(pingPin, HIGH);
// Перетворимо час в дистанцію:
    inches = microsecondsToInches(duration);
    cm = microsecondsToCentimeters(duration);
    Serial.print(inches);
    Serial.print("in, ");
    Serial.print(cm);
    Serial.print("cm");
    Serial.println();
    delay(100);
    }
    long microsecondsToInches(long microseconds)
    {
    // Згідно з паспортними даними до PING))),
швидкість ультразвуку 73,746 µs/дюйм це дозволяє
    // нам визначити дистанцію,пройдену сигналом (туди і
назад)
    // Тому ми ділимо її на 2, щоб отримати реальну
дистанцію до об'єкта
    return microseconds / 74 / 2;
    }
    long microsecondsToCentimeters(long microseconds)
    {
      // Швидкість звуку - 331 м/с або 30 µs/см
      // Імпульс рухається туди і назад, тому для
визначення реальної дистанції потрібно поділити
      // отримане значення на 2
    return microseconds / 30 / 2;
    }
```

3.1.3 Використання інтерфейсу SPI. Датчик температури та атмосферного тиску SCP1000

Датчик SCP1000 може зчитувати як атмосферний тиск, так і температуру, а потім передавати ці дані через SPI. Більш докладно про регістри управління дивіться в datasheet SCP1000 [4].

Для підключення SCP1000 потрібні наступні апаратні засоби:

- плата Arduino;
- плата з датчиком SCP1000;
- макетна плата Breadboard або плата для прототипування.

Електрична схема підключення SCP1000 до Arduino наведена на рис. 3.4.

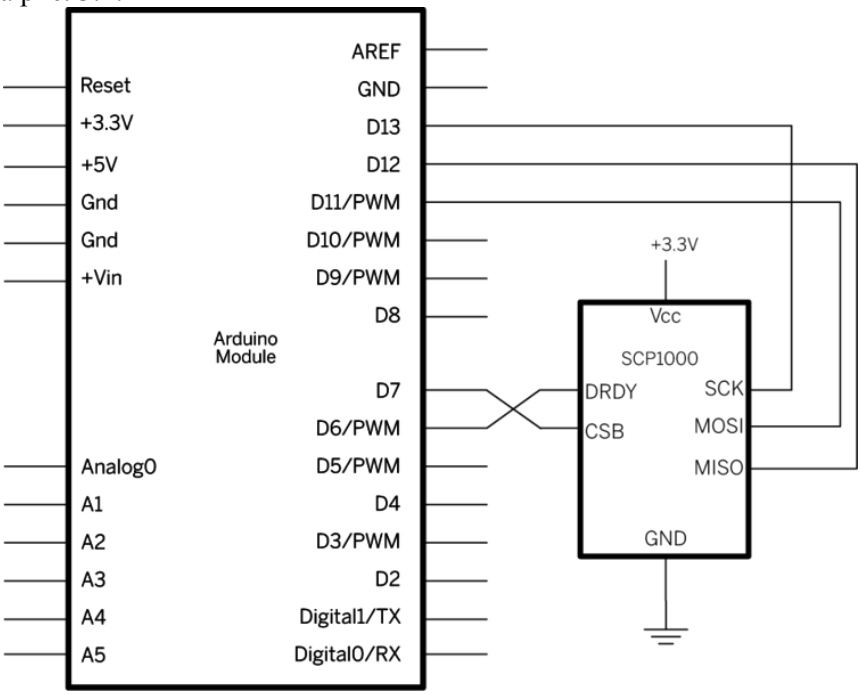

Рисунок 3.4 – Електрична схема підключення SCP1000 до Arduino

Датчик потрібно під'єднати до другого, сьомого, одинадцятого, дванадцятого і тринадцятого контактів тандему «Arduino і Shield», а також підключити живлення через вихідний контакт 3,3-вольта. Контакт DRDY (Data Ready - дані готові для зчитування) підключіть до цифровому контакту6, а контакт CSB (Chip Select - вибір веденого

пристрою) - до 7-го. Контакт MOSI (Master Out, Slave In; для передачі даних від ведучого пристрою до веденого) повинен бути підключений до цифровому контакту 11, а MISO (Master In, Slave Out - для передачі даних від веденого пристрою до ведучого) - до 12. Контакт SCK (Serial Clock - для передачі послідовного тактового сигналу) підключіть до цифрового контакту 13. Переконайтеся, що обидва пристрої підключені до однієї і тієї ж «землі». Результатом є ланцюг наведений на рис.3.5.

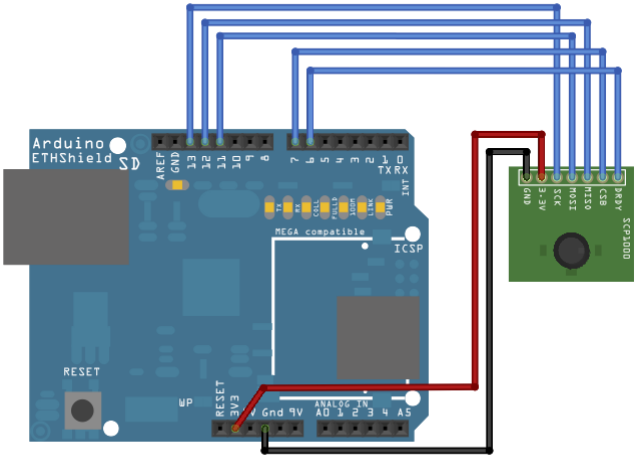

Рисунок 3.5 – Ланцюг підключення SCP1000 до Arduino

3.1.4 Використання бібліотеки Wire для роботи з I2C

Мікроконтролери (МК) Atmega (Ардуино) мають апаратну підтримку інтерфейсу I2C (TWI). Лінії інтерфейсу SDA і SCL у МК Atmega8 / 168/328, розташовані на пінах с номерами 27 (PC4) і 28 (PC5), відповідно.

На платах Arduino, лінія даних - SDA (data line) виведена на аналоговий пін 4, а лінія тактірованія - SCL (clock line) виведена на аналоговий пін 5. На Arduino Mega, SDA - це цифровий пін 20, а SCL - цифровий пін 21.

Для роботи з протоколом I2C, у Arduino є штатна бібліотека Wire [5], яка дозволяє взаємодіяти з I2C / TWI-пристроями, як в режимі Master, так і в режимі Slave.

У Arduino 1.0, бібліотека успадковується від Stream,

```
class TwoWire : public Stream
{
```

що робить її використання схожим з іншими бібліотеками читання/запису (read/ write libraries). У зв'язку з цим, методи *send()* і *receive()* були замінені на *read()* і *write()*.

Розглянемо методи бібліотеки.

Методи:

```
void begin ();
void begin (uint8_t address);
void begin (int address)
```

здійснюють ініціалізацію бібліотеки Wire і підключення до шині I2C в якості Master або Slave. Як правило, викликаються тільки один раз.

Параметри:

`Address` - 7-бітна адреса пристрою (в режимі Slave). Якщо не вказано, то контролер підключається до шині в ролі Master.

Значень, що повертаються немає.

Метод

```
uint8_t requestFrom (uint8_t address, uint8_t quantity;
```

Використовується майстром для запиту байта від веденого пристрою. Байти можуть бути отримані за допомогою методів *available()* і *read()*.

Параметри:

`Address` - 7-бітний адресу пристрою для запиту байтів даних

`Quantity` - кількість запитаних байт

Повертає число лічених байт.

Метод

```
void beginTransmission (uint8_t address);
```

Починає передачу I2C для веденого пристрою (Slave) із заданою адресою. Потім, потрібно викликати метод *write()* для додавання послідовності байт в чергу призначених для передачі, і виконати саму передачу даних методом *endTransmission()*.

Параметри:

`Address` - 7-бітна адреса пристрою для передачі

Значень, що повертаються немає.

Метод

```
uint8_t endTransmission (void);
```

завершує передачу даних для веденого пристрою, яке було розпочато

beginTransmission() і, фактично, здійснює перечу байт, які були поставлені в чергу методом *write()*.

Повертається байт, який вказує статус передачі:

0: успіх

1: даних занадто багато і вони не поміщаються в буфер передачі. Розмір буфера задається визначенням

```
#define BUFFER_LENGTH 32
```

2 отримали NACK на передачі адреси

3 отримали NACK на передачі даних

4: інша помилка

Методи

```
size_t write(uint8_t data);
size_t write(const uint8_t *data, size_t quantity);
```

Виклик:

```
Wire.write(value);
Wire.write(string);
Wire.write(data, length);
```

Ці методи записують дані від веденого пристрою у відповідь на запит майстра, або записують чергу байт для передачі від майстра до веденого пристрою (в проміжках між викликами *beginTransmission()* і *endTransmission()*).

Параметри:

`value` - значення для відправлення як одиничний байт;

`string` - рядок для відправлення як послідовність байт;

`data` -масив байт для відправлення;

`length` -число байт для передачі.

Повертається число записаних байт.

Приклад.

```
#include <Wire.h>
byte val = 0;
void setup()
{
   Wire.begin(); // підключення до шини I2C
}
void loop()
{
      Wire.beginTransmission(44); // передача для пристрою #44 (0x2c)
      // адреса пристрою вказується в документації (datasheet)
      Wire.write(val);          // відправка байта val
```

```
        Wire.endTransmission();    // передача даних
        val++;      // інкремент значення
        if(val == 64) // якщо досягли 64 (max)
        {
            val = 0;   // починаємо з мінімального значення
        }
        delay(500);
}
```

Метод

```
void onReceive (void (* function) (int));
```

Реєструє функцію, яка викликається, коли ведений пристрій отримує дані від майстра.

Параметри:

`function` - функція, яка викликається, коли ведений отримує дані; обробник повинен приймати один параметр - *int* (число байт, зчитаних від майстра) і нічого не повертати.

Наприклад:

```
void MyHandler (int numBytes);
```

Значень, що повертаються немає.

Метод

```
void onRequest (void (* function) (void));
```

Реєструє функцію, яка викликається, коли майстер запрошує дані з цього веденого пристрою.

Параметри:

`function` - функція, яка буде викликатися; не має параметрів і нічого не повертає.

Наприклад:

```
void MyHandler();
```

3.2 Завдання до роботи

3.2.1 Вивчити бібліотеку SPI [6], необхідну для обміну даних між SCP1000 та Arduino за допомогою шини SPI.

3.2.2 Підключити датчик SCP1000 згідно рис. 3.5.

3.2.3 Написати на мові С++ програму для відображення даних атмосферного тиску та температури від датчика на Serial Monitor з використанням бібліотеки SPI**.** Одиниці вимірювання атмосферного тиску – Паскалі, а температури – градуси Цельсія.

3.2.4 Вивчити бібліотеку Wire (п.3.1.4), необхідну для роботи з шиною I2C.

3.2.5 Написати програму роботи з годинником DS3231 по шині I2C з використанням бібліотеки Wire. Програма повинна забезпечувати установку часу (секунди, хвилини, години) і дати (день тижня, місяць, рік), а також зчитування даних про поточний час і дату.

Для визначення адреси мікросхеми DS3231SN на шині I2C необхідно запустити сканер пристроїв [7]. Більш докладну інформацію щодо годинника дивіться в datasheet DS3231 [8].

3.3 Контрольні питання

3.3.1. Які основні компоненти платформи Arduino?

3.3.2. На чому базується інтегроване середовище розробки Arduino?

3.3.3. Принцип роботи ультразвукового далекоміру PING))).

3.3.4. Що таке інтерфейс SPI? Особливості його використання.

3.3.5. Що таке інтерфейс I2C? Особливості його використання.

3.3.6 Яке призначення бібліотеки Wire?

3.3.7. Назвіть основні методи бібліотеки Wire.

СПИСОК ЛІТЕРАТУРИ

1. Соммер У. Программирование микроконтроллерных плат Arduino/Freeduino / У. Соммер. – СПб: БХВ, 2012. – 238 с.

2. Blink. [Електрон. ресурс]. – Режим доступу: https://www.arduino.cc/en/Tutorial/Blink.

3. Ping. [Електрон. ресурс]. – Режим доступу: http://www.arduino.cc/en/Tutorial/Ping

4. SCP1000 Datasheet. [Електрон. ресурс]. – Режим доступу: http://www.alldatasheet.com/view.jsp?Searchword=Scp1000

5. Wire Library. [Електрон. ресурс]. – Режим доступу: http://arduino.net.ua/file_archive/Arduino%20Library/Arduino%20Wire%20Library/

6. SPI library. [Електрон. ресурс]. – Режим доступу: https://www.arduino.cc/en/Reference/SPI

7. i2c_scanner. [Електрон. ресурс]. – Режим доступу: http://playground.arduino.cc/Main/I2cScanner

8. DS3231SN (Dallas). [Електрон. ресурс]. – Режим доступу: http://www.allcomponents.ru/dallas/ds3231sn.htm

Додаток А

Спеціальні змінні і режими *bash shell*

А.1 Системні змінні

HOME	Шляхове ім'я початкового каталога користувача
LOGNAME	Реєстраційне ім'я
USER	Реєстраційне ім'я
TZ	Годинниковий пояс, що використовується системою

А.2 Перевизначені змінні

SHELL	Шляхове ім'я програми командного інтерпретатора
PATH	Список шляхових імен каталогів, в яких слід шукати виконавчі каталоги
PS1	Основне запрошення (запит) оболонки
PS2	Додаткове запрошення оболонки
IFS	Символ-розділювач полів
MAIL	Ім'я файла поштової скриньки, в якому утиліта електронної пошти шукає вхідні повідомлення
MAILCHECK	Період між перевірками поштової скриньки

А.3 Змінні користувачів

MAILPATH	Список файлів поштових скриньок, в яких утиліта електронної пошти шукає вхідні повідомлення
TERM	Тип термінала
CDPATH	Шляхові імена каталогів, в яких інтерпретатор шукає виконавчі файли
EXINIT	Команди установки режимів для текстових редакторів ex і vi

А.4 Спеціальні режими

ignoreeof	Блокування можливості виходу з оболонки за допомогою символа кінця файла (Ctrl-D)
noclobber	Запобігання запису файлів поверх існуючих при переадресації
noglob	Блокування спеціальних символів, що використовуються для формування списку імен файлів: *, ?, ~ і []

Додаток Б

Аргументи bash shell

$0	Ім'я UNIX-команди, що виконується
$n	n-й аргумент командного рядка, починаючи з першого. Для зміни значень цих аргументів можна користуватися командою set
$*	Всі аргументи командного рядка, починаючи з першого
$@	Всі аргументи командного рядка, взяті окремо в лапки
$#	Кількість аргументів командного рядка
$$	Ідентифікаційний номер поточного процесу
$!	Ідентифікаційний номер останнього фонового завдання
$?	Код завершення останньої з виконаних команд

Додаток В

Операції команди порівняння test

В.1 Порівняння цілих

-gt	Більше ніж
-lt	Менше ніж
-ge	Більше або дорівнює
-le	Менше або дорівнює
-eq	Дорівнює
-ne	Не дорівнює

В.2 Порівняння рядків

-z	Перевірка на рядок нульової довжини
-n	Перевірка на рядкове значення
=	Перевірка на рівність рядків
!=	Перевірка на нерівність рядків
str	Перевірка на рядок ненульової довжини

В.3 Логічні операції

-a	Логічне І
-o	Логічне АБО
!	Логічне НЕ

В.4 Перевірка файлів

-f	Файл існує і є звичайним
-s	Файл не порожній
-r	Файл читаємий
-w	В файл можливий запис
-x	Файл виконавчий
-d	Ім'я файла - це ім'я каталога
-h	Ім'я файла - це символічне посилання
-O	Файл належить користувачеві