ТЕМА № 3

Складання сценаріїв

Зміст теми: Знайомство із текстовою операційною оболонкою ***bash***, вивчення мови програмування оболонки ***bash*** та практичне складання найпростіших сценаріїв за допомогою редактора ***vi*** (***vim***).

Теоретичні відомості

3.1 Загальні відомості про текстові оболонки в *Linux*

Досить часто в процесі роботи з комп'ютером потрібно часто повторювати одні і ті ж команди *Linux*. Операційна система дозволяє записати необхідну послідовність команд в спеціальний файл, який називається сценарієм оболонки. Далі цей сценарій можна виконувати подібно виконанню звичайної команди Linux, набравши ім'я файла. Такий принцип організації файлів існує в MS DOS – це так звані командні файли.

Створювати сценарії надають спеціальні програми – оболонки, які виступають посередником між користувачем та операційною системою. Існують як текстові, так і графічні оболонки. Найбільш відомі в *Linux* текстові оболонки – ***bash, csh, tcsh, ksh, pdksh***.

Наприклад, необхідно виконати таку послідовність команд:

```
mkdir  dir1
cp  file1.txt  /home/user/dir1
cd  dir1
```

Для того, щоб оформити вказані команди у вигляді сценарію, необхідно спочатку вказати назву оболонки, в рамках якої вони будуть виконані. Якщо у сценарії використовуються лише команди операційної системи, як у нашому випадку, вибір оболонки не є суттєвим. Тому виберемо найбільш поширену оболонку, яка практично завжди є в більшості дистрибутивів *Linux* – оболонку ***bash***. Таким чином, першим рядком сценарію має бути такий запис:

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

Цей рядок є по суті ознакою того, що даний файл відноситься до сценаріїв. А далі вже записуються команди, які мають бути виконані.

Після створення файла сценарію (наприклад, під іменем *run*) необхідно перевірити, чи надано право виконати цей сценарій даному користувачеві. Як правило, для того, щоб перетворити будь-який власний файл у виконуваний файл, рядовий користувач повинен надати собі таке право за допомогою команди *chmod:*

```
chmod  u+x run
```

Далі сценарій можна запустити на виконання із командного рядка:

```
./run
```

або

bash run

Для того, щоб зробити сценарій більш універсальним (в нашому випадку – щоб сценарій був придатний для різних імен каталогів та файлів), використовуються позиційні параметри. За допомогою позиційних параметрів операційна система може передавати оболонці конкретні параметри (імена файлів, каталогів, змінних і т.д.) під час виклику сценарію у командному рядку або з іншого сценарію. Такі позиційні параметри мають спеціальні імена. Перший параметр зберігається у змінній з іменем *1*(один) і отримати його значення в сценарії можна за допомогою виразу *$1*. Другий параметр зберігається у змінній з іменем *2*(два) і т. д. Кількість одночасно використовуваних позиційних параметрів обмежена – не більше 9. Однак дозволяється за допомогою оператора

shift n

пересувати кожний позиційний параметр на *n* позицій вліво і відкидати *n* попередніх значень справа.

В нашому прикладі сценарій з використанням двох позиційних параметрів матиме вигляд:

```
#!/bin/bash
mkdir %1
cp %2 /home/user/%1
cd %1
```

Запуск на виконання цього сценарію матиме вигляд:

./run dir1 file1.txt

В оболонці можна також використовувати і власні команди оболонки. Суттєво збільшуються можливості сценаріїв при використанні спеціальних операторів, які утворюють мову програмування оболонки.

Розглянемо основні змінні та оператори мови програмування оболонки **bash.**

3.2 Змінні в сценаріях для *bash*

Мова програмування оболонки підтримує три основних типи змінних:

- змінні середовища;
- вбудовані змінні;
- змінні користувача.

Змінні середовища надаються системою, їх не потрібно визначати, але до них можна звертатись. Значення деяких із них, наприклад *PATH*, можна змінювати в програмі оболонки. Основні змінні середовища наведені в таблиці 3.1.

Вбудовані змінні також надаються системою, але їх не можна змінювати. До цих змінних відносяться, зокрема, такі:

$# - ряд позиційних параметрів, які передаються сценарію оболонки;

$0 - ім'я сценарію;

*$** - рядок зі всіма параметрами, які передаються сценарію оболонки під час його виклику.

Таблиця 3.1 – Змінні середовища

Змінна середовища	Значення
EDITOR	Редактор, який викликається за замовчуванням
HOME	Домашній каталог користувача
LOGNAME	Імя, під яким користувач зареєструвався в системі
MAIL	Шлях до поштової скриньки (файла e_mail) користувача
PATH	Список каталогів, що переглядаються системою при пошуку команди
PS1	Початкове запрошення оболонки (*$* - для рядового користувача, # - для суперкористувача)
SHELL	Місцезнаходження оболонки
TERM	Тип термінала користувача

Змінні користувача визначаються самим користувачем під час написання сценарію оболонки. В своєму власному сценарії користувач може довільно використовувати та змінювати їх. На відміну від звичайних універсальних мов програмування, в командних інтерпретаторах не визначається тип змінної (як, наприклад, в мові Сі – int). Система самостійно "здогадується" про тип змінної за тим значенням, яке їй присвоюється. Це легко зробити, оскільки в сценарії можна оперувати тільки цілими числами або текстовими рядками. В такому випадку одна і та ж змінна в один момент часу може слугувати для збереження цілого числа, а через деякий час – для збереження рядка. Однак це не рекомендується робити.

Для присвоєння змінній користувача цілого значення або одного текстового слова досить записати оператор присвоєння в загальноприйнятій формі, наприклад

```
n=5
mas=0
str1=process
```

Характерною особливістю сценаріїв є те, що для доступу до значення змінної, перед її іменем ставиться знак долара (*$*). Наприклад, для того, щоб змінній *m* присвоїти значення змінної *n* необхідно записати

```
m=$n
```

Якщо текстовий рядок складається з кількох слів, тобто містить пропуски, тоді використовуються лапки, наприклад

str2='long text string'

Для присвоєння текстових рядків використовуються також і подвійні лапки. В цьому випадку забезпечується підстановка значень змінних всередині рядків. Наприклад, якщо записати

str2="long text string"

str3="Value of str2 is $str2"

тоді значенням змінної *str3* буде

Value of str2 is long text string

3.3 Програмування арифметичних виразів

Для програмування арифметичних виразів можна застосувати такі знаки операцій:

+ сума,
- різниця,
***** множення,
/ ділення,
% ділення з остачею.

Арифметичні вирази можна записати двома способами:

а) з використанням оператора ***let***;

б) з використанням оператора ***expr***.

Перший спосіб найбільш простий і зрозумілий, наприклад:

let *y=$a*$b-$c*

Оператор ***expr*** розглядає свої аргументи як арифметичний або логічний вираз. В такому випадку потрібно враховувати деякі додаткові особливості такого запису, наприклад, символи арифметичних операцій відділяти від операндів пропуском, символ операції множення а також і весь вираз брати в лапки:

y='expr $a '' $b '+' $c'*

3.4 Оператори введення і виведення

Для введення змінних з клавіатури використовується оператор ***read***. Наприклад, для введення змінних *var, var2, var3* в сценарії необхідно записати:

read *var1 var2 var3*

Для виведення повідомлень на екран дисплея використовується оператор ***echo***. Наприклад,

echo *This is message*

Якщо в сценарії необхідно вивести значення змінної, тоді використовуються подвійні лапки, наприклад

echo *"result is $y"*

Користуючись операторами введення та виведення можна написати найпростіший сценарій для обчислення арифметичних виразів:

```
#!/bin/bash
a=3
b=5
echo "Введіть значення змінної x"
read x
let y=($a+$b)*$x
echo "result is $y"
```

3.5 Порівняння виразів

Розглянемо порівняння чисел, рядків а також логічні і файлові операції порівняння.

3.5.1 Порівняння чисел

Для порівняння двох чисел можуть використовуватись такі операції:

a –eq b визначення рівності чисел ***a*** і ***b***;
a –ne b визначення нерівності чисел ***a*** і ***b***;
a –gt b визначення того, чи число ***a*** більше числа ***b*** ;
a –ge b визначення того, чи число ***a*** більше або дорівнює числу ***b***;
a –lt b визначення того, чи число ***a*** менше числа ***b***;
a –le b визначення того, чи число ***a*** менше або дорівнює числу ***b***.

3.5.2 Порівняння рядків

Для порівняння двох рядків можуть використовуватись такі операції:

str1 = str2 визначення рівності рядків *str1* і *str2*;
str1 != str2 визначення нерівності рядків *str1* і *str2*;
-n перевірка ненульової довжини рядка;
-z перевірка нульової довжини рядка.

3.5.3 Логічне порівняння

Логічні операції використовуються для порівняння виразів логічних операцій *NOT, AND* і *OR*:

! логічна операція *NOT* над логічним виразом;
-a логічна операція *AND* над двома логічними виразами;
-o логічна операція *OR* над двома логічними виразами.

3.5.4 Файлові операції порівняння

Такі операції можуть використовуватись для перевірки файлів:

-d перевірка того, чи є файл каталогом;
-f перевірка того, чи є файл звичайним файлом;
-r перевірка того, чи є право доступу для читання файла;
-w перевірка того, чи є право доступу для запису у файл;

-x перевірка того, чи є право доступу для виконання файла;
-s перевірка того, чи є файл з ненульовою довжиною.

3.6 Умовні оператори

Умовний оператор ***if*** дозволяє в залежності від виконання заданої умови <*виразу*> виконувати <*оператори 1*> або <*оператори 2*>.Формат цього оператора такий:

```
if <вираз>
  then <оператори 1>
else <оператори 2>
 fi
```

Умовні оператори можуть бути вкладеними, наприклад:

```
if <вираз 1>
   then <оператори 1>
else if <вираз 2>
   then <оператори 2>
else <оператори 3 >
 fi
 fi
```

Ключове слово ***fi*** означає закінчення одного умовного оператора, тому їх кількість у вкладеному умовному операторі повинна дорівнювати кількості ключових слів ***if***. Існує також форма скороченого запису вкладеного умовного оператора, коли достатньо лише одного ключового слова ***fi***:

```
if <вираз 1>
  then <оператори 1>
elif <вираз 2>
  then <оператори 2>
else <оператори 3>
 fi
```

3.7 Оператор-перемикач

Існує спеціальний оператор, який зручно використовувати при великій кількості розгалужень. Оформити такий запис дозволяє оператор ***case***, формат якого такий:

```
case var in
     S1) <оператори 1>;;
     S2) <оператори 2>;;
     S3) < оператори 3>;;
     *)  <оператори 4>;;
esac
```

В залежності від того, чи збігається значення змінної *var* із значенням ***S1, S2*** або ***S3***, виконуються відповідно <*оператори 1*>, <*оператори 2*> або <*оператори 3*>. Якщо вказаного збігу немає, тоді виконуються <*оператори 4*>.

3.8 Оператор циклу *for*

Оператор ***for*** має декілька форматів. Найпростіший формат цього оператор циклу, який використовує одновимірний список, має такий вигляд:

```
for  var in list
 do
      <оператори>
 done
```

В даному випадку <*оператори*> виконуються по одному разу для кожного значення змінної *var* із списку *list*. Приклад сценарію для знаходження суми елементів одновимірного масиву:

```
#!/bin/bash
mas=’3 7 12 5 8’
sum=0
for var in $mas
do
    let sum=$sum + $var
done
echo “result  is $sum”
```

Формат циклу ***for*** з використанням масивів дуже схожий на відповідний формат циклу в мові Сі.

Приклад сценарію з використанням циклу ***for*** для знаходження максимального значення серед елементів одновимірного масиву:

```
#!/bin/bash
mas[0]=3
mas[1]=7
mas[2]=12
mas[3]=5
mas[4]=8
max=mas[0]
for((i=0; i<5; i++))
do
   if  [ $m ax  –lt  ${mas[i]} ]
   then let max=${mas[i]}
fi
done
echo “result is $max”
```

3.9 Оператори циклу *while* та *until*

Оператор циклу ***while*** можна використовувати для повторного виконання <*операторів*> до тих пір, поки заданий <*вираз*> буде залишатись істинним:

```
while <вираз>
do
    <оператори>
done
```

Можливо, що цикл не буде виконано жодного разу, якщо заданий <*вираз*> виявиться хибним з самого початку.

Приклад сценарію з використанням циклу ***for*** для знаходження максимального значення серед елементів двовимірного масиву, який вводиться із клавіатури:

```
#!/bin/bash
for((i=0; i<5; i++))
do
for((j=0; j<5; j++))
do
    read mas[i][j]
done
done
    max=mas[0]
for((i=0; i<5; i++))
do
for((j=0; j<5; j++))
do
if [ $max –lt ${mas[i][j]} ]
   then let max=${mas[i][j]}
fi
done
done
echo “result is $max”
```

Оператор циклу ***until*** можна використовувати для повторного виконання <*операторів*> до тих пір, поки заданий <*вираз*> буде залишатись хибним:

```
until <вираз>
do
    <оператори>
done
```

3.10 Функції

Як і в мовах високого рівня, окремі частини сценаріїв можна записувати у вигляді функцій. Формат визначення функції такий:

```
func() {
  <оператори>
 }
```

Виклик функції, якій передаються параметри *param1, param2, param3* :

```
func param1 param2 param3
```

Можна також передати параметри у вигляді одного рядка, наприклад, $@. Функція може інтерпретувати параметри за тими же принципами, за якими виконується інтерпретація позиційних параметрів, що передаються сценарію оболонки. Наприклад, для обчислення виразу

$$y = \begin{cases} \dfrac{a+b}{c} & \textit{якщо} \quad x = 5 \\ (a+b)*c & \textit{якщо} \quad x \neq 5 \end{cases}$$

можна використати дві функції:

```
#! /bin/ bash
 a = 9
 b = 5
 c = 7
 d = 2
calc1() {
      let y= ($a+$b) /$1
     echo “Result is $y”
      }
 calc2() {
      let y = ($a+$b)*$1
     echo “Result is $y”
          }
echo “input x”
read x
if  [ $x  - eq 5 ]
      then calc1  c
      else calc2  d
 fi
```

3.11 Робота з файлами

Використовуючи файлові операції порівняння, можна із заданого списка імен знаходити файли або каталоги, а також визначати їх права доступу. Наприклад:

```
#! /bin/ bash
if  [ -d  name1 ]
        then  echo " name1 is directory"
el  if  [ -f  name2 ]
        then   echo " name2  is file"
else  echo " name1 and name2  is not directory or file"
fi
if  [ -w  name2 ]
        then  echo " file has write permission"
else  echo " file has not write permission"
fi
```

В системних сценаріях *Linux* часто зустрічаються випадки, коли потрібно виконати задану послідовність операцій в залежності від інформації, яка записана у відповідних файлах.Складемо сценарій, в результаті виконання якого на екрані з'являється вікно системної програми годинника або калькулятора, якщо у файлі */home/user/Select.txt* змінній *Program* присвоєно значення відповідно "*XCLOCK*" або "*XCALC*". Звертаємо увагу, що цей сценарій може бути виконано лише в графічній оболонці ***X*** (детальніше графічний режим *Linux* розглядається в наступній лабораторній роботі).

```
#! /bin/ bash
. /home/user/Select.txt
if [ "$Program" = "XCLOCK" ]
        then  exec  xclock &
elif [ "$Program" ="XCALC" ]
        then  exec  xcalc &
fi
```

3.12 Рекомендована література з теми 3

[1, с.266-331], [5, с.451-464], [6, с.324-327].

Повний список літератури знаходиться на стор. 87.

Порядок виконання роботи

1. Скласти сценарій за отриманим завданням.
2. Викликати редактор ***vi*** (***vim***) та набрати текст сценарію.
3. Надати сценарію право на використання (права доступу задаються командою ***chmod***).
4. Виконати сценарій. В разі отримання повідомлення про помилки виправити їх.