ЛЕКЦІЯ 21. ОСНОВИ СТВОРЕННЯ І ВИКОНАННЯ СЦЕНАРІЇВ SHELL. ОРГАНІЗАЦІЯ СКЛАДНИХ СТРУКТУР У SHELL – СЦЕНАРІЯХ.

У випадку, якщо періодично виконують ті самі команди, то прибігають до написання сценаріїв. Сценарії не будуть виконуватися також швидко як програма, написана на мові, що компілюється, але він займає значно менше місця й не вимагає для своєї роботи ніяких додаткових бібліотек, крім самої оболонки й використовуваних у системі команд. Процес створення й тестування сценаріїв значно простіше, ніж розробка програм. Послідовність створення й застосування сценарію:

1 Створити текстовий файл:

cat > my1

2 Внести в нього сценарій:

#!/bin/sh # у сценаріях використовують послідовність #(#!) для вказівки типу оболонки

alias ll='ls -l' # використовується для виводу #розгорнутих списків каталогів.

alias ldir='ls -a' # вивід додаткових індикаторів для #файлів, що виконуються, і каталогів.

3 Зробити файл, що виконується: chmod +x my1

4 Запустити цей файл на виконання: sh my1

Змінні

Мова програмування оболонки цілком розвинений і підтримує різні типи змінних. Змінні підрозділяються на три типи:

- *Змінні середовища* являють собою частину системного середовища. Їх не потрібно визначати, але до них можна звертатися.
- *Убудовані змінні* надаються системою. На відміну від змінні середовища їх не можна змінити.
- *Користувальницькі змінні* визначаються користувачем при написанні сценарію оболонки. У цьому сценарії користувач може довільно використовувати й модифікувати їх. Основна відмінність мов оболонки від інших мов програмування полягає в тому, що командних інтерпретаторах не визначається тип змінної. Присвоєння значення змінної виробляється через знак рівності, причому перед знаком рівності й після нього не повинне бути пробілів. p=0.

У мовах оболонки не використовуються типізовані змінні, тому та сама змінна в якийсь момент може служити для зберігання цілого числа, а через якийсь час для зберігання рядка, але це не бажано.

Якщо рядок не містить пробілів, то відбувається присвоювання значення через знак дорівнює, якщо містить пробіли, то значення змінної полягає в подвійні лапки: m1=Vasya, m2=”Vasya Pupkin”

Щоб одержати доступ до значення змінної, потрібно перед її ім'ям поставити знак долара: echo $m1 echo $m2

Вбудовані змінні

Вбудовані змінні - це спеціальні змінні, що надаються системою Linux, які можуть використовуватися для прийняття рішень про хід виконання програми. У сценарії оболонки не можна змінювати значення цих змінних.

До цих змінних відносяться:

- $# - ряд позиційних параметрів, переданих сценарію оболонки.
- $? - код завершення останньої команди, виконаного усередині даного сценарію оболонки.
- $0 - ім'я сценарію.
- $* - єдиний рядок з усіма параметрами, переданими сценарію оболонки під час
  його виклику.

Наприклад, тестова програма pr1

```
#!/bin/sh
# Тестова програма
echo “кількість параметрів”$# echo
“назва програми”$0
echo “передані наступні параметри”$*
```

при виконанні програми з командного рядка буде отриманий

Кількість параметрів 2 Ім'я програми pr1
Передані параметри Vasya Pupkin

Спеціальні символи

До таких символів ставляться подвійні лапки (“), одинарні лапки ('), зворотний слеш (\), і зворотні одинарні лапки (`). У сценаріях можна також використовувати оператори перенапрямку уведення й виводу, однак у цих випадках потрібно дотримувати обережності, тому що можна затерти існуючі файли.

Подвійні лапки використовуються для рядків з пробіли, для того що б оболонка

розглядала рядок як єдине ціле:

```
X=abc def # Команда видає помилку
X=”abc def” # Команда сприйме як весь рядок
```

Рядок беруть в одинарні лапки, щоб заборонити командному інтерпретаторові

виконувати підстановку значень змінних:

```
X=123
B='$X'
echo
B=$B
```

Виведе на екран:

```
B=$X
```

Зворотній слеш використовується перед символом, щоб заборонити командному інтерпретаторові розглядати даний символ у якості спеціального. Припустимо, що потрібно привласнити змінної var строкове значення $test,

якщо написати цю команду через оператор присвоювання, то в змінну буде занесене значення null.

Var=$test # занесеться значення null

Це відбувається тому, що оболонка розглядає $test як значення змінної test. Для
усунення помилки необхідно записати.

Var=\$test

Зворотний слэш перед знаком долара вказує оболонці, щоб вона розглядала $ як будь-який інший символ і не віддавала йому спеціального значення. Крім знака $ використовуються наступні спеціальні символи.

Спеціальні символи оболонки:

- | - направляє стандартний вивід наступній команді;
- # - позначає початок коментарю;
- & - забезпечує виконання процесу у фоновому режимі;
- ? - відповідає одному символу;
- * - відповідає будь-якій кількості символів;
- > - оператор перенапрямку виводу;
- < - оператор перенапрямку уведення;
- [] - визначає діапазон символів;
- [a -z] - позначає всі символи від a до;
- [a ,z] - позначає символи a або z;
- .ім'я_файлу - виконує файл (точка)
- Пробіл - роздільник між двома словами.

Зворотні одинарні лапки можуть використовуватися як вказівка оболонці виконати рядок, укладену в такі лапки. Це можна застосувати в сценаріях, коли потрібно зберегти в змінної результат виконання команди. Наприклад, якщо потрібно підрахувати число рядків у файлі test.txt у поточному каталозі, і зберегти результат у змінної var можна скористатися наступною командою: var ='wc -l test.txt'

Порівняння виразів

В оболонці bash підтримуються наступні типи порівнянь:

- порівняння рядків;
- порівняння чисел;
- файлові операції порівняння;
- логічне порівняння.

Порівняння рядків

Для порівняння двох строкових виражень можна використовувати наступні операції:

- =Визначення рівності двох рядків.
- != Визначення нерівності двох рядків.
- -nПеревірка ненульової довжини рядка.

- -z Перевірка нульової довжини рядка.

Приведемо приклади порівняння двох рядків:

```
#!/bin/sh
string1=”abc”
string2=”abd”
if [$string1=$string2]; then echo “Рядок1
      дорівнює рядку2”
else
      echo “Рядок1 не дорівнює рядку2”
fi
if [$string1!=$string2]; then
      echo “Рядок2 не дорівнює рядку1” else
      echo “Рядок2 дорівнює рядку1”
fi
if [$string1]; then
      echo “Рядок1 не порожня”
else
      echo “Рядок1 порожня”
fi
if [-n $string2]; then
      echo “Рядок2 має довжину більше чим нуль” else
      echo “Рядок2 має довжину, рівну нулю”
fi
if [-z $string1]; then
      echo “Рядок1 має довжину, що рівну нулю” else
      echo “Рядок1 має довжину, що більше чим нуль”
fi
```

Якщо два рядки не рівні по розмірі, система для порівняння доповнює більше короткий рядок кінцевими пробілами.

Порівняння чисел

Для порівняння двох чисел можуть бути використані наступні операції:

- -eq - Визначення рівності двох чисел.
- -ge – Визначення того, що одне число більше або дорівнює іншому.
- -le – Визначення того, що одне число менше або дорівнює іншому.
- -ne – Визначення нерівності двох чисел.
- -gt – Визначення того, що одне число більше іншого.
- -lt – Визначення того, що одне число менше іншого.

Розглянемо приклад, у якому виконується порівняння трьох чисел.

```
#!/bin/sh
numb1=5
```

```
numb2=10
numb3=5
if [$numb1 -eq $numb3]; then echo
      “Число1 дорівнює числу3”
else
      echo “Число1        не дорівнює числу3”
fi
if [$numb1 -ne $numb2]; then
      echo “Число1 не дорівнює числу2” else
      echo “Число1        дорівнює числу2”
fi
if [$numb1 -gt $numb2]; then echo
      “Число1 більше числа2”
else
      echo “Число1        не більше числа2”
fi
if [$numb1 -ge $numb3]; then
      echo “Число1 більше або дорівнює числу3” else
      echo “ Число1 не більше або дорівнює                числу3”
fi
if [$numb1 -lt $numb2]; then echo
      “Число1 менше числа2”
else
      echo “Число1        не менше числа2”
fi
if [$numb1 -le $numb3]; then
      echo “Число1 менше або дорівнює числу3” else
      echo “ Число1 не менше або дорівнює                числу3”
fi
```

Файлові операції порівняння

Наступні операції можуть використовуватися для перевірки файлів:

- -d – Перевірка того, чи є файл каталогом.
- -f - Перевірка того, чи є файл звичайним файлом.
- -r - Перевірка того, чи встановлений біт, що дозволяє читання файлу.
- -s - Перевірка того, чи має ім'я файлу ненульову довжину.
- -w - Перевірка того, чи встановлений біт, що дозволяє запис у файл.
- -x - Перевірка того, чи встановлений біт, що дозволяє виконання файлу.

Припустимо, що програма оболонки звертається до файлу file1 підкаталогу dir1 поточного каталогу. Допустимо також, що file1 має права доступу на читання й виконання й підкаталог dir1 має права доступу rwx. Тоді сценарій порівняння буде виглядати приблизно так:

```
#!/bin/sh
if [-d $dir1]; then
```

```
        echo “dir1 - це каталог” else
        echo “ dir1            - це не каталог”
 fi
   if [-f $dir1]; then
        echo “dir1 - це звичайний файл” else
        echo “ dir1            - це не звичайний файл”
 fi
 if [-r $file1]; then
        echo “file1 є можливість прочитати” else
        echo “ file1 закритий для читання ”
 fi
 if [-w $file1]; then
        echo “file1 є можливість записати” else
        echo “ file1 закритий для запису ”
 fi
 if [-x $dir1]; then
        echo “dir1 є права на запуск” else
        echo “ dir1 - немає прав на запуск”
 fi
```

Логічне порівняння

Логічні операції використовуються для порівняння виражень із використанням правил логіки. Наступні символи відповідають операціям NOT, AND, OR.

- ! - заперечення логічного вираження
- -a - логічна операція AND над двома логічними вираженнями
- -o - логічна операція OR над двома логічними вираженнями

```
#!/bin/sh
if [-x file1 -a -x dir1]; then
      echo “ file1 і dir1 є виконуваним” else
echo “ file1 або dir1 не є виконувим”
fi
if [-w file1 –o -w dir1]; then
        echo “file1 або dir1 є записуваними”
else
        echo “file1 або dir1 є закритими від запису”
fi
if [! -w file1]; then
        echo “file1 є закритим від запису ”
else
        echo “file1 відкритий для запису”
fi
```

Оператори циклу.

Оператор циклу використовується для повторення ряду команд, що втримуються всередині нього. Для організації циклів в Linux використовуються різні оператори.

Оператор for

Оператор має кілька форматів. Перша форма оператора виглядає в такий спосіб: for змінна in список

do

операторі

done

Оператор for у такій формі призначений для виконання операторів по одному разі для кожного значення в списку. При кожній ітерації змінної привласнюється поточне значення списку. Як список може застосовуватися змінна, утримуючих кілька елементів, або список значень, розділених пробілами.

Друга форма оператора виглядає в такий спосіб: for змінна

do

операторі

done

У цій формі оператори виконуються по одному разі для кожного з позиційних параметрів, переданих сценарію оболонки. При кожній ітерації змінної привласнюється поточне значення позиційного параметра. Ця форма може бути записана також у такий спосіб: for змінна in “$@”

do

операторі

done

Пам'ятайте, що $@ надає список позиційних параметрів, переданих сценарію оболонки у вигляді одного рядка.

Припустимо, що для кожного файлу в поточному каталозі потрібно створити резервну копію в підкаталозі backup. Це можна зробити в такий спосіб:

```
#!/bin/sh
for filename in “ls” do
        cp $filename backup/$filename if [$?
        -ne 0]; then
                echo “copy for $filename failed”
fi done
```

У прикладі створюється резервна копія кожного файлу. Якщо операція копіювання закінчується невдало, то видається повідомлення про помилку.

Оператор while

Оператор може використовуватися для циклічного виконання ряду команд доти, поки зазначена умова залишається щирим. Цикл завершується, як тільки зазначена умова стане помилковим. Якщо зазначена умова ложно споконвічно, то цикл не виконається жодного разу. При використанні команди while потрібно дотримувати обережності, оскільки цикл ніколи не закінчиться, якщо зазначена умова завжди буде залишатися щирим. Для оператора while застосовується наступний формат:

```
while вираз

do
        оператори
done
```

Для додавання перших п'яти парних чисел можна використовувати наступний сценарій:

```
#! /bin/bash
loopcount=0
result=0
while [$loopcount -lt 5]
do loopcount='expr $loopcount + 1' increment='expr
$loopcount\*2' result='$result + $increment'
          done
          echo “result is $result”
```

Оператор until

Може використовуватися для циклічного виконання ряду команд доти, поки зазначена умова не є щирим. Цикл завершується, як тільки зазначена умова стає щирим.

Для цього оператора застосовується наступний формат:

```
until вираз
do
        оператори
done
```

Умовні оператори

Використовуються в сценаріях оболонки для визначення того, яку частину програми потрібно виконати залежно від зазначених умов.

Оператор if

Оператор дозволяє виконати команди залежно від результатів перевірки логічних виражень. Формат умовного оператора наступний: if [вираження1]; then

```
оператор1
```

```
elif [вираження2]; then
      оператор2
else
      операт
      ор3
fi
```

Умовні оператори if можуть бути вкладеними, тобто містити усередині себе ще один умовний оператор if. Оператор if не обов'язково повинен мати у своєму составі оператори elif або else. Оператор else виконується, якщо ні умова в операторі if, ні умова в операторі elif не є щирими. Ключове слово fi служить для позначення кінця умовного оператора, що дуже зручно при використанні вкладених операторів if. У складних конструкціях необхідно ретельно перевіряти відповідність один одному парних ключових слів. У наступному прикладі змінна var може мати одне із двох значень: Yes або No. Всі інші значення є
неприпустимими. Для цього можна використовувати наступну

```
конструкцію: if [$var=”Yes”]; then
      echo “Value is Yes”
elif [$var=”No”]; then
      echo “Value is No”
  else
      echo “invalid value”
  fi
```

Оператор case

Оператор використовується для виконання операторів залежно від того, чи відповідає зазначена змінна одному конкретному значенню або одному з ряду значень. При перевірці великої кількості умов краще застосовувати оператор case замість
оператора if.

```
case рядок in шаблон1)
      список команд;;
шаблон2)
      список команд;;
   ...
*)
      список команд;;
esac
```

Приклад.

```
echo -n " А яку оцінку одержав на іспиті?: "
  read z
case $z in
      5)   echo "Молодець !                              ";;
```

```
        4)   echo "Однаково молодець !                    ";;
        3)   echo "Однаково !                           ";;
        2)   echo "Всі !                                ";;
        *) echo  "!                                     ";;
    esac
```

Оператори break і exit

Оператор break може використовуватися для завершення циклу, виконуваного за допомогою операторів for, until, while. Оператор exit може використовуватися для виходу зі сценарію оболонки. Після цього оператора можна вказати число - код завершення. Якщо один сценарій оболонки буде викликаний іншим сценарієм, то зухвала програма може перевірити код завершення й прийняти відповідне рішення.

Функції

Як і в інших мовах прогаммирования, у сценаріях оболонки можуть також застосовуватися функції. Функції - це частина сценарію, що призначена для виконання певного завдання й може бути викликана в цій програмі кілька разів. Застосування функцій дозволяє писати сценарії оболонки без дублювання коду. Формат визначення

функції

```
    насту
    пний:
    func()
    {
        оператори
    }
```

Фукцию можна викликати в такий спосіб:

```
    func параметр1 параметр2 параметр3
```

Параметри є не обов'язковими. Можна також передати параметри в у вигляді одного рядка, наприклад, $@. Функція може інтерпретувати параметри по тимі ж принципам, як виконується інтерпретація позиційних параметрів, переданих сценарію оболонки. Розглянемо приклад функції, що виводить назву місяця або повідомлення про
помилку у відповідь на уведений номер місяця.

```
    Func1(){
    case $1
    in

        01|1) echo “Місяць січень”;; 02|2)
        echo “Місяць лютий”;; 03|3) echo
        “Місяць березень”;; 04|4) echo
        “Місяць квітень”;; 05|5) echo
```

```
        “Місяць травень”;; 06|6) echo
        “Місяць червень”;; 07|7) echo
        “Місяць липень”;; 08|8) echo “Місяць
        серпень”;; 09|9) echo “Місяць
        вересень”;;
        10)   echo “Місяць жовтень”;;
        11)   echo “Місяць листопад”;;
        12)   echo “ Місяць січень”;;
        *) echo “Неправильний параметр” ;; esac
  }
Запуск функції для серпня місяця
  Func1 8
```