*ПЗ-10b:* ***Управляющие конструкции языка shell***

**1. Создание сценария оболочки** (см. ПЗ-10а).

На ПЗ*-*10b продолжим рассмотрение **bash-скриптов**. Это — сценарии командной строки, написанные для оболочки **bash**. Существуют и другие оболочки, например — **zsh, tcsh, ksh**, но мы сосредоточимся на **bash**.

**BASH** — **Bourne**-**Again** **SHell** (что может переводится как «перерожденный шел», или «*Снова шел Борн* (создатель **sh**)»), самый популярный командный интерпретатор в юниксоподобных системах, в особенности в **GNU**/**Linux**.

Итак,

1) Вход в систему под выданной пользовательской учетной записью (например):

login: user

psw: 12345678

2) Создание в домашнем каталоге или tmp (/tmp/g3ХXX, где ХХХ – последние 3 цифры – год набора + № группы) каталога группы и ПЗ-2/11. Например: **/tmp/g3771/pz12**

~$ cd tmp

~$ ls

~$ mkdir gNN (где NN – последние 2 цифры группы, например g21)

~$ ls

***Сценарии командной строки*** — это наборы тех же самых команд, которые можно вводить с клавиатуры, собранные в файлы и объединённые некоей общей целью. При этом результаты работы команд могут представлять либо самостоятельную ценность, либо служить входными данными для других команд. ***Сценарии*** — это мощный способ автоматизации часто выполняемых действий.

Итак, если говорить о командной строке, она позволяет выполнить несколько команд за один раз, введя их через точку с запятой: **pwd; whoami**

Если вы опробовали это в своём терминале, ваш первый **bash-скрипт**, в котором задействованы две команды, уже написан. Работает он так. Сначала команда **pwd** выводит на экран сведения о текущей рабочей директории, потом команда **whoami** показывает данные о пользователе, под которым вы вошли в систему

***Командная строка*** — отличный инструмент, но команды в неё приходится вводить каждый раз, когда в них возникает необходимость. Что если записать набор команд в файл и просто вызывать этот файл для их выполнения? Собственно говоря, тот файл, о котором мы говорим, и называется *сценарием командной строки*.

Создайте пустой файл с использованием команды ***touch***: **touch myscript.sh**

В его первой строке нужно указать, какую именно оболочку мы собираемся использовать. Нас интересует **bash**, поэтому первая строка файла будет такой:

**#!/bin/bash** # This is a comment

Команды оболочки отделяются знаком перевода строки, комментарии выделяют знаком решётки. Вот как это выглядит:

|  |  |
| --- | --- |
| **#!/bin/bash**  **# This is a comment**  **pwd**  **whoami** | Сохраните файл, дав ему имя **myscript.sh**, и работа по созданию bash-скрипта почти закончена. Сейчас осталось лишь сделать этот файл исполняемым, иначе, попытавшись его запустить, вы столкнётесь с ошибкой ***Permission denied:*** |
| ***Permission denied:***  *Попытка запуска файла сценария с неправильно настроенными разрешениями* | https://habrastorage.org/getpro/habr/post_images/29f/743/3ec/29f7433ec002900e7518396a814fe351.png |
| **chmod +x ./myscript** | Сделаем файл исполняемым |
| **./myscript** | Теперь попытаемся его выполнить |

После настройки разрешений всё работает как надо.

**Вывод сообщений**

|  |  |  |
| --- | --- | --- |
| **echo** | Для вывода текста в консоль Linux применяется команда **echo**. | |
| **#!/bin/bash**  **#** *our comment is here*  **echo "The current directory is:"**  **pwd**  **echo "The user logged in is:"**  **whoami** | | https://habrastorage.org/getpro/habr/post_images/4d0/173/e3d/4d0173e3ddcad01d0790b6b98dd91315.png |
| **#!/bin/bash**  **echo "Hello World!"**  **printf "%s\n" "Hello World"** | | **Hello World!**  **Hello World!** |

А что если надо вывести на экран значок доллара? Попробуем так:

**echo "I have $1 in my pocket"**

Система обнаружит знак доллара в строке, ограниченной кавычками, и решит, что мы сослались на переменную. Скрипт попытается вывести на экран значение неопределённой переменной **$1**. Это не то, что нам нужно. Что делать?

В подобной ситуации поможет использование управляющего символа, обратной косой черты, перед знаком доллара (теперь сценарий выведет именно то, что ожидается):

|  |  |
| --- | --- |
| Использование управляющей последовательности для вывода знака доллара:  **echo "I have \$1 in my pocket"** | https://habrastorage.org/getpro/habr/post_images/771/a8a/710/771a8a7102c7a25f1ecd1e739a00de6e.png |

**2. Математические операции.**

|  |  |
| --- | --- |
| Для выполнения математических операций в файле скрипта можно использовать конструкцию вида $((a+b)): | #!/bin/bash  var1=$(( 5 + 5 ))  echo $var1  var2=$(( $var1 \* 2 ))  echo $var2 |

**3. Конструкция if**

Как и во многих языках программирования, в **bash** есть условные конструкции. Они имеют формат, описанный ниже. Будьте внимательны: слова «**if**» и «**then**» должны находится на разных строках. Старайтесь выравнивать горизонтально всю конструкцию, включая заключительный «**fi**» и все «**else**». Это делает код намного удобнее для чтения и отладки. В дополнении к простой форме «**if**, **else**» есть еще несколько других форм условных конструкций:

|  |  |  |
| --- | --- | --- |
| **if [ условие ]**  **then**  **действие**  **fi** | В приведенном выше примере ‘действие’ выполняется только если ‘условие’ верно, в противном случае скрипт продолжает выполнение инструкций со строки идущей за «fi». | |
| if [ условие ]  then  действие  elif [ условие\_2 ]  then  действие\_2  elif [ условие\_3 ]  then  # *Это комментарий: …****команды Bash****…*  else  действие\_x  fi | А эта конструкция последовательно проверяет условия и если они верны, то исполняет соответствующее действие. Если ни одно из условий не верно, то выполняется ‘действие\_x’ стоящее после ‘else’ (если оно есть). Потом продолжается исполнение команд идущих за этой конструкцией «if,then,else», если таковые есть. | |
| Пример 1:  #!/bin/bash  if pwd  then  echo "It works"  fi | В данном случае, если выполнение команды pwdзавершится успешно, в консоль будет выведен текст «it works». | |
| Пример 2:  #!/bin/bash  user=likegeeks  if grep $user /etc/passwd  then  echo "The user $user Exists"  fi | Здесь мы воспользовались командой **grep** для поиска пользователя в файле **/etc/passwd**. В этом примере, если пользователь найден, скрипт выведет соответствующее сообщение. А если найти пользователя не удалось? В данном случае скрипт просто завершит выполнение, ничего нам не сообщив.  *Хотелось бы, чтобы он сказал нам и об этом, поэтому усовершенствуем код*. | |
| **if команда**  **then**  **команды**  **else**  **команды**  **fi** | Для того, чтобы программа смогла сообщить и о результатах успешного поиска, и о неудаче, воспользуемся конструкцией if-then-else. | |
| Пример 3:  #!/bin/bash  user=anotherUser  if grep $user /etc/passwd  then  echo "The user $user Exists"  else  echo "The user $user doesn’t exist"  fi | **if команда1**  **then**  **команды**  **elif команда2**  **then**  **команды**  **fi** | Зададимся вопросом о более сложных условиях. Что если надо проверить не одно условие, а несколько? Например, если нужный пользователь найден, надо вывести одно сообщение, если выполняется ещё какое-то условие — ещё одно сообщение, и так далее. В подобной ситуации нам помогут вложенные условия. |

**Сравнение чисел**

В скриптах можно сравнивать числовые значения. Ниже приведён список соответствующих команд.

**n1 -eq n2** Возвращает истинное значение, если n1 равно n2.

**n1 -ge n2**  Возвращает истинное значение, если n1больше или равно n2.

**n1 -gt n2** Возвращает истинное значение, если n1 больше n2.

**n1 -le n2** Возвращает истинное значение, если n1меньше или равно n2.

**n1 -lt n2** Возвращает истинное значение, если n1 меньше n2.

**n1 -ne n2** Возвращает истинное значение, если n1не равно n2.

|  |  |
| --- | --- |
| #!/bin/bash  val1=6  if [ $val1 -gt 5 ]  then  echo "The test value $val1 is greater than 5"  else  echo "The test value $val1 is not greater than 5"  fi | Значение переменной val1 больше чем 5, в итоге выполняется ветвь then-оператора сравнения и в консоль выводится соответствующее сообщение. |

**Сравнение строк**

В сценариях можно сравнивать и строковые значения. Операторы сравнения выглядят довольно просто, однако у операций сравнения строк есть определённые особенности, которых мы коснёмся ниже. Вот список операторов.

**str1 = str2**  Проверяет строки на равенство, возвращает истину, если строки идентичны.

**str1 != str2** Возвращает истину, если строки не идентичны.

**str1 < str2** Возвращает истину, если str1меньше, чем str2.

**str1 > str2**  Возвращает истину, если str1больше, чем str2.

**-n str1** Возвращает истину, если длина str1больше нуля.

**-z str1** Возвращает истину, если длина str1равна нулю.

|  |  |
| --- | --- |
| #!/bin/bash  user ="likegeeks"  if [$user = $USER]  then  echo "The user $user is the current logged in user"  fi  ПРАВИЛЬНО: | Вот одна особенность сравнения строк, о которой стоит упомянуть. А именно, операторы «>» и «<» необходимо экранировать с помощью обратной косой черты, иначе скрипт будет работать неправильно, хотя сообщений об ошибках и не появится.  Скрипт интерпретирует знак «>» как команду перенаправления вывода. |

#!/bin/bash

val1=text

val2="another text"

if [ $val1 \> $val2 ]

then

echo "$val1 is greater than $val2"

else

echo "$val1 is less than $val2"

fi

Обратите внимание на то, что скрипт, хотя и выполняется, выдаёт предупреждение:

|  |  |
| --- | --- |
| ./myscript: line 5: [: too many arguments  Для того, чтобы избавиться от этого предупреждения, заключим $val2 в двойные кавычки: | #!/bin/bash  val1=text  val2="another text"  if [ $val1 \> "$val2" ]  then  echo "$val1 is greater than $val2"  else  echo "$val1 is less than $val2"  fi |

**Проверки файлов**

Пожалуй, нижеприведённые команды используются в bash-скриптах чаще всего. Они позволяют проверять различные условия, касающиеся файлов. Вот список этих команд.

**-d file**  Проверяет, существует ли файл, и является ли он директорией.

**-e file**  Проверяет, существует ли файл.

**-f file**  Проверяет, существует ли файл, и является ли он файлом.

**-r file**  Проверяет, существует ли файл, и доступен ли он для чтения.

**-s file**  Проверяет, существует ли файл, и не является ли он пустым.

**-w file**  Проверяет, существует ли файл, и доступен ли он для записи.

**-x file**  Проверяет, существует ли файл, и является ли он исполняемым.

**file1 -nt file2**  Проверяет, новее ли file1, чем file2.

**file1 -ot file2** Проверяет, старше ли file1, чем file2.

**-O file**  Проверяет, существует ли файл, и является ли его владельцем текущий пользователь.

**-G file**  Проверяет, существует ли файл, и соответствует ли его идентификатор группы идентификатору группы текущего пользователя.

Этот скрипт, для существующей директории (например **/tmp/g3771/pz12)** выведет её содержимое:

#!/bin/bash

mydir=/home/likegeeks

if [ -d $mydir ]

then

echo "The $mydir directory exists"

cd $ mydir

ls

else

echo "The $mydir directory does not exist"

fi

**4. Команда test и ее аналоги. Проверка файлов и сравнение строк**

|  |  |
| --- | --- |
| #!/bin/bash  file=/etc/fstab # Объявляем переменную  if **test** -e $file # Используем команду test  then  echo "Файл fstab найден."  else  echo "Что-то нету такого файла"  fi | Как мы видим, мы использовали знакомые нам операторы if / then, которые я уже описывал ранее.  После оператора if стоит команда **test** которая и производит проверку нашего файла, который мы объявили в переменной. Также, команду test мы использовали с опцией "-e", данная опция возвращает значение "true", если файл найден.  В случае "true" запускается оператор then и выводит строку echo, если же файл не найдет, что означает "false", то запускается оператор else и выводит сообщение "Что-то нету такого файла". |

Теперь давайте рассмотрим аналоги команды test.

|  |  |
| --- | --- |
| #!/bin/bash  file=/etc/fstab # Объявляем переменную  if /usr/bin/[ -e $file ] # Используем команду "[", также  # мы закрыли ее правой ], потому bash требует  # ее наличие  then  echo "Файл fstab найден."  else  echo "Что-то нету такого файла"  fi | Использовали вместо команды test, команду "[" с закрытой правой "]". Вообще, старые версии bash не требовали наличие правой закрывающейся ]. Теперь это необходимо, чтобы избежать ошибок.  Рекомендуем вместо **[ ]** использовать конструкцию  **[[ ]]** |

Тот же скрипт, только вместо test используем другие команды:

|  |  |
| --- | --- |
| #!/bin/bash  file=/etc/fstab # Объявляем перменную  if [[ -e $file ]] # Используем конструкцию [[ ]]  then  echo "Файл fstab найден."  else  echo "Что-то нету такого файла"  fi | Почему [[ ]] лучше, чем [ ]? Потому что внутри [[ ]] вы можете использовать например такие операторы, как: &&, ||, < и >, в то время как внутри [ ] будут сообщения об ошибках.  Кстати, вы можете использовать [ ] и [[ ]] без if / then / else. Те команды как бы заменяют данные операторы:  var1=5  var2=10  [[ "$var1" -ne "$var2" ]] && echo "$var1 не равно $var2" |

**5. Условия. case. Множественный выбор**

|  |  |
| --- | --- |
| Если необходимо сравнивать, какую-то одну переменную с большим количеством параметров, то целесообразней использовать оператор **case:** (создайте файл **menu2.sh**) | |
| #!/bin/bash  echo "Выберите редатор для запуска:"  echo "1 Запуск программы nano"  echo "2 Запуск программы vi"  echo "3 Запуск программы emacs"  echo "4 Выход"  read doing #здесь мы читаем в переменную $doing со стандартного ввода  case $doing in  1)  /usr/bin/nano # если $doing содержит 1, то запустить nano  ;;  2)  /usr/bin/vi # если $doing содержит 2, то запустить vi  ;;  3)  /usr/bin/emacs # если $doing содержит 3, то запустить emacs  ;;  4)  exit 0  ;;  \*) #если введено с клавиатуры то, что в case не описывается, выполнять следующее:  echo "Введено неправильное действие"  esac #окончание оператора case. | Результат работы:  user:~$ ./menu2.sh  Выберите редактор для запуска:  1 Запуск программы nano  2 Запуск программы vi  3 Запуск программы emacs  4 Выход  После выбор цифры и нажатия Enter запуститься тот редактор, который вы выбрали (если конечно все пути указаны правильно, и у вас установлены эти редакторы. |

Законспектируйте в тетрадь список логических операторов, которые используются для конструкции

**if-then-else-fi** и ей подобных:

**-z** # строка пуста

**-n** # строка не пуста

**=, (==)**  # строки равны

**!=** # строки неравны

**-eq** # равно

**-ne** # неравно

**-lt,(< )**  # меньше

**-le,(<=)**  # меньше или равно

**-gt,(>)**  #больше

**-ge,(>=)**  #больше или равно

**!** #отрицание логического выражения

**-a,(&&)**  #логическое «И»

**-o,(||)** # логическое «ИЛИ»