Занятие 11. Сценарии командной оболочки, переменные

Переменные

Часто в сценарии командной оболочки результаты работы одной команды нужно передать другой команде. Эту задачу можно решить с использованием переменных. Переменные позволяют сохранять информацию во время выполнения сценария для использования в других командах сценария.

Переменные среды

Речь идет о переменных среды, которые рассматривались в занятии № 10. Значения этих переменных можно вставлять в сценарии.

```
#!/bin/bash
# Отображение информации о пользователе
echo "User info for $USER"
echo UID : $UID
echo HOME: $HOME
```

Вывод этого сценария

```
user@linux-pc:~/bin$ user_info
User info for user
UID : 1000
HOME: /home/user
user@linux-pc:~$
```

Пользовательские переменные

В командных сценариях можно задавать собственные переменные.

Имена пользовательских переменных могут включать буквы и цифры и символ подчеркивания. Первый символ в имени может быть только буквой или символом подчеркивания. Имена переменных чувствительны к регистру символов.

Командная оболочка не различает константы и переменные. Типичное соглашение – использовать буквы верхнего регистра для обозначения констант и нижнего регистра для переменных.

```
#!/bin/bash
# Простейшие действия с переменными
msg="Hello!"
favorite_number=321
echo $msg
echo "My favorite number is $favorite_number"
```

Присваивание значений переменных выполняется так

```
переменная=значение
```

Значения переменным присваиваются с помощью знака равенства (=). Обратите внимание на отсутствие пробелов в операции присваивания между именем переменной, знаком = и значением.

При ссылке на значение переменной знак доллара должен быть приведен, а если переменная указана в целях присваивания ей значения, то знак доллара не используется.

Командная оболочка не заботится о типе значения переменной, все значения она интерпретирует как строки.

Переменные, которые определены в сценарии, сохраняют свои значения на протяжении всего времени выполнения сценария и разрушаются при завершении сценария.

Возможные значения переменной

a=z	**Строка “z”.**
b=“a string”	Строка “a string”. Пробелы должны находиться в кавычках.
c=“a string and $b”	При присваивании допускается выполнять подстановку.
d=$(ls -l foo.txt)	Результат выполнения команды.
e=$((5 * 7))	Результат вычисления арифметического выражения.
f=“\t\ta string\n”	Строка с экранированными управляющими символами (табуляции и перехода на новую строку).

Функции

Функции — это «мини-сценарии», находящиеся внутри другого сценария, которые работают как автономные программы. Функции имеют две синтаксические формы.

```
function имя {
  команды
  return
}
```

```
имя() {
  команды
  return
}
```

```
#!/bin/bash
# Простейшее использование функции
function func {
  echo "Step 2"
  return
}
# Основная программа
echo "Step 1"
func
echo "Step 3"
```

Командная оболочка читает сценарий и пропускает строки с 1 по 7. Выполнение сценария начинается со строки 8, на которой записана первая команда echo. Затем выполняется строка 9, в которой вызывается функция func и управление передается на строку 4. Выполняется вторая команда echo. Следующей выполняется строка 5. Команда return в этой строке возвращает управление на строку, следующую за вызовом функции func (строка 10). После этого выполняется последняя команда echo.

Все переменные, которые использовались в сценариях до сих пор, были глобальными. Глобальные переменные существуют и доступны в любой точке программы. Внутри функций можно (и нужно) определять локальные переменные. Локальные переменные доступны только внутри функции, в которой они определены, и прекращают свое существование после завершения выполнения функции.

```
#!/bin/bash
# Локальные переменные


# Глобальная переменная
foo=0
func_1 () {
  local foo
  foo=1
  echo "func_1: foo = $foo"
}
func_2 () {
  local foo
  foo=2
  echo "func_2: foo = $foo"
```

```
}
echo "global foo: foo = $foo"
func_1
echo "global foo: foo = $foo"
func_2
echo "global foo: foo = $foo"
```

При определении локальных переменных используется слово local. Когда управление выйдет за пределы функции, в которой определена локальная переменная, переменная перестанет существовать. Вывод сценария иллюстрирует это:

```
user@linux-pc:~/bin$ func_local_vars
global foo: foo = 0
func_1: foo = 1
global foo: foo = 0
func_2: foo = 2
global foo: foo = 0
user@linux-pc:~/bin
```

Условный оператор

Условный оператор имеет следующий синтаксис

```
if команды; then
  команды
[elif команды; then
  команды]
[else
  команды]
fi
```

Как командная оболочка определяет успешно или неуспешно выполнилась команда? Любая команда при завершении возвращает операционной системе значение, которое называется кодом завершения. Это значение — целое число. По принятому соглашению значение 0 служит признаком успешного завершения команды.

```
if cmp "$file_1" "$file_2" > /dev/null; then
  echo "Файлы $a и $b идентичны."
else
  echo "Файлы $a и $b имеют различия."
fi
```

Командная оболочка поддерживает специальную переменную, которая содержит код завершения команды:

```
user@linux-pc:~/bin$ ls -d /bin
/bin
user@linux-pc:~/bin$ echo $?
0
user@linux-pc:~/bin$ ls -d /bon
ls: cannot access '/bon': No such file or directory
user@linux-pc:~/bin$ echo $?
2
user@linux-pc:~/bin$
```

Чаще всего с оператором if используется команда test, которая выполняет различные проверки и сравнения. Команда имеет две эквивалентные формы:

```
test выражение
```

и (более популярную)

```
[ выражение ]
```

Команда test возвращает значение 0, если выражение истинно, и 1 в противном случае. Ниже приведены несколько примеров выражений для проверки истинности различных условий. С полным набором таких выражений можно познакомиться в документации.

Выражения для проверки файлов

Выражение	**Истинно, если…**
файл*1* *-nt* *файл*2	файл*1 новее файл*2
-d файл	файл существует и является каталогом
-e файл	файл существует
-f файл	файл существует и является обычным файлом

```
#!/bin/bash
# Информация о файле

# Имя файла
FILE=~/.bashrc

if [ -e "$FILE" ]; then
  if [ -f $FILE ]; then
    echo "$FILE: обычный файл"
  fi
  if [ -d "$FILE" ]; then
    echo "$FILE: каталог"
  fi
  if [ -r "$FILE" ]; then
    echo "$FILE: чтение разрешено"
  fi
  if [ -w "$FILE" ]; then
```

```
    echo "$FILE: запись разрешена"
  fi
  if [ -x "$FILE" ]; then
    echo "$FILE: выполнение разрешено"
  fi
else
  echo "$FILE не существует"
  exit 1
fi
exit
```

Обратите внимание на несколько моментов. Во-первых, имя файла внутри команды test заключено в кавычки. В случае если переменная FILE окажется не определена, вместо ее значения будет подставлена пустая строка и выполнение скрипта не завершится аварийно. Во-вторых, в конце сценария используется команда exit. У команды exit есть единственный необязательный аргумент — код возврата. Если аргумент не указан, команда exit вернет значение по умолчанию — 0. В примере выше использование exit позволяет сценарию сообщить об ошибке. Команда exit в самом конце сценария добавлена формально. Когда командная оболочка достигает конца сценария, она завершает его с кодом возврата 0.

Функции могут возвращать свой код завершение с помощью команды return.

Выражение	**Истинно, если…**
строка	строка не пустая
-n строка	длина строки больше нуля
-z строка	длина строки равна 0
строка*1* == *строка*2	строка*1 и строка*2 равны
строка*1* > *строка*2	строка*1 больше, чем строка*2
строка*1* < *строка*2	строка*1 меньше, чем строка*2

```
#!/bin/bash
# Работа со строками

# Анализируемый ответ
ANSWER=yes
if [ -z $ANSWER ]; then
  echo "Ответа нет (пустая строка)"  >&2
  exit 1
fi

if [ "$ANSWER" == "yes" ]; then
  echo "Ответ ДА"
elif [ "$ANSWER" == "no" ]; then
  echo "Ответ НЕТ"
else
  echo "Ответ непонятен"
fi
```

Обратите внимание на перенаправление вывода в первой команде echo: сообщение будет выведено в стандартный вывод ошибок.

Выражение	**Истинно, если…**
число*1* *-eq* *число*2	число*1* *и число*2 равны
число*1* *-ne* *число*2	число*1* *и число*2 не равны
число*1* *-lt* *число*2	число*1* *меньше, чем число*2
число*1* *-gt* *число*2	число*1* *больше, чем число*2

```
#!/bin/bash
# Работа с целыми числами

# Анализируемое число
INT=-5

if [ -z "$INT" ]; then
  echo "Число пустое" >&2
  exit 1
fi

if [ $INT -eq 0 ]; then
  echo "Число равно нулю"
else
  if [ $INT -lt 0 ]; then
    echo "Число отрицательное"
  else
    echo "Число положительное"
  fi

  if [ $((INT % 2)) -eq 0 ]; then
    echo "Число четное"
  else
    echo "Число нечетное"
  fi
fi
```

Выражения для проверки могут объединяться с помощью логических операций:

- И (-a);
- ИЛИ (-o);
- НЕ (!).

```
MIN_VAL=1
MAX_VAL=100

INT=-1

if [ $INT -ge $MIN_VAL -a $INT -le $MAX_VAL ]; then
  echo "$INT принадлежит отрезку от $MIN_VAL до $MAX_VAL."
else
  echo "$INT вне отрезка"
fi
```

Та же самая проверка, но записанная по другому

```
if [ ! \( $INT -ge $MIN_VAL -a $INT -le $MAX_VAL \) ]; then
  echo "$INT вне отрезка"
else
  echo "$INT принадлежит отрезку от $MIN_VAL до $MAX_VAL."
fi
```

Выражение в команде test заключено в круглые скобки, потому что операция отрицания имеет самый высокий приоритет и, в случае отсутствия скобок, будет применяться к результату первого выражения, а не к объединению двух выражений.

Круглые скобки в выражении команды test нужно экранировать.

Литература

1. Шотсс У. “Командная строка Linux.”, главы 25, 26, 27
2. Блум Р., Бреснахэн К. “Командная строка Linux и сценарии оболочки.”, главы 10, 11