Конвейеры

Конвейеры — это команды, которые соединены операторами `;`, `&&`, `||`для выполнения в определенной последовательности. Операторы организации конвейеров работают следующим образом:

- `команда1` ; `команда2`— `команда2` выполняется после `команды1` независимо от результата её работы ;
- `команда1 && команда2`— `команда2` выполняется только после успешного выполнения команды1(то есть с кодом завершения 0);
- `команда1 || команда2`— `команда2` выполняется только после неудачного выполнения команды1 (то есть код завершения `команды1` будет отличным от 0)

Условные операторы

В скриптовом языке bash поддерживаются два оператора ветвления: `if` и `case`. Оператор `if`, как и в других языках, выполняет определенный блок указаний, в зависимости от условия. Условие помещают в двойные квадратные скобки `[[ ... ]]`, которые bash рассматривает как один элемент с кодом выхода. Внутри блока операторов помещенных в `[[ ]]` разрешается использовать операторы `&&` и `||`. Например:

```
# Однострочная запись
if [ ... ]; then echo "true"; else echo "false"; fi;

## Вложенные условия
if [ ... ] &amp;amp;&amp;amp; [ ... ]; then
        ...
elif [[ ... &amp;amp;&amp;amp; ... ]]; then
        ...
else
        ...
fi;

```

Обратите внимание, что `[`, условие и `]` обязательно должны быть разделены пробелами, иначе оболочка воспримет в качестве команды `[условие`.

Ниже приведена таблица с возможными условиями сравнения:

```
# Работа с файлами
-e    Проверить существует ли файл или директория (-f, -d)
-f    Файл существует (!-f - не существует)
-d    Каталог существует (!-f - не существует)
-s    Файл существует и он не пустой
-r    Файл существует и доступен для чтения
-w    ... для записи
-x    ... для выполнения
-h    символьная ссылка

# Работа со строками
-z    Пустая строка
-n    Не Пустая строка
==    Равно
!=    Не равно

# Операции с числами
-eq   Равно
-ne   Не равно
-lt   Меньше
-le   Меньше или равно
-gt   Больше
-ge   Больше или равно

```

Пример:

```
if [ `uname` == "Adam"]; then
    echo "Не ешь яблоко!"
elif [ `uname` == "Eva"] then
    echo "Не бери яблоко!"
else
    echo "Яблоки сейчас очень дорогие!"
fi;

```

Если необходимо сделать выбор из нескольких альтернатив, пригодится оператор case. Принцип его работы легче понять на примере:

```
case "$extension" in
    (jpg|jpeg)
        echo "Это изображение в формате jpeg."
    ;;
    png)
        echo "Это изображение в формате png"
    ;;
    gif)
       echo "А это ))"
    *)
        echo "Оу!Это вообще не изображение!"
    ;;
esac

```

В примере оператор проверяет значение переменной `$extension` на совпадение с одним из шаблонов и в случае совпадения выполнит соответствующий блок

кода. Если же совпадений не будет, выполнятся указания, соответствующие шаблону `*`.

Циклы

Язык оболочки дает пользователю возможность организовывать циклическое выполнение инструкций при помощи циклов: `*`

```
1
2  while
3  for
4  *select
5
```

while

Оператор `while` описывается следующим образом:

```
1  while условие do
2         тело
3  done
4
```

Интерпретатор в первую очередь выполняет команды, описанные в `условии`. Если результат выполнения нулевой, то выполняется тело, а после ее выполнения, переход к следующей итерации, в противном случае происходит выход из цикла. В условии может быть любая допустимая команда. Например:

```
#!/bin/sh
# Квадраты чисел от 1 до 10
x=0
while [ $x -lt 10 ] do #значение переменной x меньше 10?
    echo $(($x*$x))
    x=`expr $x + 1` # увеличиваем x на 1
done

```

for

Цикл for выполняет тело для каждого элемента из списка. Синтаксис цикла for таков:

```
for имя in элемент1 элемент2 ... элементN do
       тело
done

```

В качестве элементов обычно используют различные шаблоны (wildcards). Очень удобно применять for для прохождения по каталогам и выполнения операций над группой файлов. В примере ниже, цикл проходит по всем файлам с расширением `*.bash`, перемещает их в директорию `~/scripts` и добавляет их права на исполнение.

```
#!/bin/sh
# Перемещение всех скриптов из ~ в директорию ~/scripts
for FILE in $HOME/*.bash do
```

```
4         mv $FILE ${HOME}/scripts
5         chmod +x ${HOME}/scripts/${FILE}
6   done
7
```

select

Цикл `select` помогает организовать удобное меню выбора и применяется тогда, когда пользователь должен выбрать один элемент из предложенного списка. В общем цикл `select`имеет такой же синтаксис, как и цикл `for`:

```
1   select ответ in элемент1 элемент2 ... элементN do
2         тело
3   done
4
```

При выполнении этого оператора, все элементы из списка высвечиваются на экране со своими порядковыми номерами в виде списка вариантов ответа, после списка выводится специальное приглашение для ввода. Обычно оно имеет вид `#?`. Введенный пользователем номер списка записывается в переменную ответ. Если ответ содержит номер пункта меню, то в переменную заносится значение соответствующего элемента из списка. Если в списке нет введенного пункта, список будет показан снова. После того, как пользователь сделает правильный выбор, выполнятся указания в теле, а цикл перейдет к следующей итерации и все действия повторятся с самого начала — именно поэтому работу цикла `select` желательно прерывать.

```
#!/bin/sh
echo -n "Введите название пакета: " &amp;amp;&amp;amp; read PACKAGE
PS3=" Выберите пакетный менеджер: "
select ITEM in bower, npm, pip do
        case $ITEM in
        bower) bower install $PACKAGE ;;
        npm) npm install $PACKAGE ;;
        pip) pip install $PACKAGE ;;
        esac
        break
done

```

Пример выше запрашивает у пользователя название пакета, который он желает установить, далее интересуется тем какой пакетный менеджер использовать и в зависимости от выбора устанавливает нужный пакет и останавливает выполнение.

Оболочка также имеет команды, которые изменяют нормальное выполнение цикла. Оператор `break` полностью останавливает выполнение цикла, оператор `continue` — переходит к следующей итерации.

Функции

В сценариях оболочки возможны объявление и

вызов функций. Стоит отметить, что само понятие функций в `bash` несколько урезано. На самом деле, функции в `bash` — это именуемая группа команд, которые выполнятся при обращении к функции. В любом случае функциями

следует пользоваться везде, где есть код, повторяющиеся с небольшими вариациями.

Объявление функции имеет следующий вид:

```
Имя функции () {
        команды
}
Имя функции     # обращение к функции

```

Объявление функции обязательно должно предшествовать ее первый вызов. Обращение к функции происходит путем указания ее имени в качестве команды.

Функция может принимать аргументы и возвращать после своего выполнения результат — код выхода. Функция обращается к своим аргументам точно так же, как и к локальным переменным, с помощью позиционных переменных — `$1`, `$2` и тд. Результат работы можно возвращать с помощью команды `return`. Например, функция, которая принимает параметр (имя) и завершает свою работу с кодом 0:

```

#!/bin/sh
#функция с параметром
greeting() {
        if [ -n "$1" ]; then
        echo "Привет, $1!"
        else
            echo "Привет, незнакомец!"
        fi

```

```
10          return 0
11  }
12
13  greeting пользователь    #&gt; Привет, пользователь!
14  greeting             #&gt; Привет, незнакомец!
15
```

Команда `return` возвращает код завершения 0 — это код успешного завершения сценария. Каждая программа по завершению работы записывает в переменную окружения `#?` код завершения — число от 0 до 255. С помощью этой переменной можно определять статус выполнения каждой отдельной команды или скрипта. Если программа завершилась ошибкой, кодом завершения будет целое число отличное от нуля. Обратите внимание, на то что, если сценарий завершается командой `exit` без параметров, кодом завершения сценария будет код завершения последней выполненной команды.

Отладка сценариев

Оболочка дает несколько средств для отладки сценариев. Для активации режима отладки, он должен быть запущен с помощью специальных опций.

Первая строка сценария должна иметь вид:

```
#!/bin/sh опция
```

Можно выбирать среди следующих функций:

`-n` — читать все команды, но не выполнять их;

`-v` — выводить все строки по мере их обработки интерпретатором;

`*-x` — выводить все команды и их аргументы по мере их выполнения.

Для отладки сценария частями, нужный фрагмент отмечают вызовом команды `set` с помощью соответствующей опции из таблицы. Причем, для включения режима отладки, перед опцией указывают символ `-`, для отключения режима отладки используют `+`:

```
1  set -x # включаем  режим отладки
2  ...
3  set +x # выключаем режим отладки
4
```

Общая практика отладки заключается в том, чтобы прежде чем запустить ее, необходимо проверить его синтаксис с помощью опции `-n`. Для большей детальности можно комбинировать ключи `-nv`. После исправления синтаксических ошибок проводится отладка с помощью опции `-x`.