

Пишем скрипты в Linux (обучение на примерах)



forkostya написал 6 лет назад#1Автор

Автор: Кузнецов Константин

[size=7]Пишем скрипты в Linux (обучение на примерах)[/size]

Содержание:

1. Введение
2. Обучение написанию сценариев на внутреннем языке BASH (Перевод с англ.)
3. Используемая и рекомендуемая литература

[size=6]1. Введение[/size]

Главная задача системного администратора - как сделать так, чтобы ничего не делать. Для достижения данной шуточной цели администраторы часто прибегают к написанию сценариев (скриптов). В этом случае всю рутинную, периодическую работу выполняет их верный помощник - компьютер.> Материал из Википедии — свободной энциклопедии

Скриптовый язык (англ. scripting language, в русскоязычной литературе принято название язык сценариев) — язык программирования, разработанный для записи «сценариев», последовательностей операций, которые пользователь может выполнять на компьютере. Простые скриптовые языки раньше часто называли языками пакетной обработки (batch languages или job control languages). Сценарии обычно интерпретируются, а не компилируются (хотя часто сценарии компилируются каждый раз перед запуском). В прикладной программе, сценарий (скрипт) — это программа, которая автоматизирует некоторую задачу, которую без сценария пользователь делал бы вручную, используя интерфейс программы.

[size=6]2. Обучение написанию сценариев на внутреннем языке BASH[/size]

оригинал: http://www.linuxconfig.org/Bash_scripting_Tutorial

Это руководство предполагает отсутствие предварительных знаний о методике написания сценариев (далее скриптов) с помощью внутреннего языка Bash. С помощью данного руководства вы обнаружите в скором времени, что написание скриптов очень простая задача. Давайте начнем наше обучение с простого сценария, выполняющего вывод строки "Hello World!" (в перев. с англ. - Всем привет!)

1) Сценарий "Всем привет"

Вот ваш первый пример bash-скрипта:> #!/bin/bash
echo "Hello World"

Переходим в директорию, содержащую наш файл hello_world.sh и делаем его исполняемым:

```
$ chmod +x hello_world.sh
```

Запускаем скрипт на выполнение

```
$ ./hello_world.sh
```

2) Простой архивирующий bash-скрипт> #!/bin/bash
tar -czf myhome_directory.tar.gz /home/user

```
$ ./backup.sh
tar: Removing leading '\ ' from member names
$ du -sh myhome_directory.tar.gz
41M myhome_directory.tar.gz
```

3) Работа с переменными

В данном примере мы объявляем простую переменную и выводим её на экран с помощью команды `echo`

```
#!/bin/bash
STRING="HELLO WORLD!!!"
echo $STRING
```

```
$ ./hello_world.sh
HELLO WORLD!!!
Наш архивирующий скрипт с переменными:> #!/bin/bash
OF=myhomedirectory$(date +%Y%m%d).tar.gz
IF=/home/user
tar -czf $OF $IF
```

```
$ ./backup.sh
tar: Removing leading '\ ' from member names
$ du -sh *tar.gz
41M myhome_directory_20100123.tar.gz
3.1) Глобальные и локальные переменные> #!/bin/bash
```

Объявляем глобальную переменную

Такая переменная может использоваться в любом месте этого скрипта

```
VAR="global variable"
function bash {
```

Объявляем локальную переменную

Такая переменная действительна только для функции, в которой её объявили

```
local VAR="local variable"
echo $VAR
}
echo $VAR
bash
```

Обратите внимание, что глобальная переменная не изменилась

```
echo $VAR
```

```
$ ./variables.sh
global variable
local variable
global variable
4) Передаем аргументы в скрипт> #!/bin/bash
```

Используйте предопределенные переменные для доступа к аргументам

Выводим аргументы на экран

```
echo $1 $2 $3 ' -> echo $1 $2 $3'
```

Мы так же можем получить доступ к аргументам через специальный массив `args=("$@")`

Выводим аргументы на экран

```
echo ${args[0]} ${args[1]} ${args[2]} ' -> args=("$@"); echo ${args[0]} ${args[1]} ${args[2]}
```

Используйте переменную `$@` для вывода всех аргументов сразу

```
echo $@ ' -> echo $@'
```

Используйте переменную `$#` для вывода количества переданных в скрипт аргументов

```
echo Number of arguments passed: $# ' -> echo Number of arguments passed: $#'
```

```
$ ./arguments.sh Bash Scripting Tutorial
Bash Scripting Tutorial -> echo $1 $2 $3
Bash Scripting Tutorial -> args=("$@"); echo ${args[0]} ${args[1]} ${args[2]}
Bash Scripting Tutorial -> echo $@
Number of arguments passed: 3 -> echo Number of arguments passed: $#
5) Выполнение в скрипте команд оболочки> #!/bin/bash
```

используйте обратные кавычки `" "` для выполнения команды оболочки

```
echo `uname -o`
```

теперь попробуем без кавычек

```
echo `uname -o`
```

```
$ `uname -o`
```

```
GNU/Linux
$ ./bash_backtricks.sh
GNU/Linux
uname -o
```

Как видим, во втором случае вывелась сама команда, а не результат её выполнения

6) Читаем пользовательский ввод (интерактивность)> #!/bin/bash

```
echo -e "Hi, please type the word: \c "
read word
echo "The word you entered is: $word"
echo -e "Can you please enter two words? "
read word1 word2
echo "Here is your input: \"$word1\" \"$word2\""
echo -e "How do you feel about bash scripting? "
```

read command now stores a reply into the default build-in variable \$REPLY

```
read
echo "You said $REPLY, I'm glad to hear that! "
echo -e "What are your favorite colours ? "
```

-a makes read command to read into an array

```
read -a colours
echo "My favorite colours are also ${colours[0]}, ${colours[1]} and ${colours[2]}:-)"
```

```
$ ./read.sh
Hi, please type the word: something
The word you entered is: something
Can you please enter two words?
Debian Linux
Here is your input: "Debian" "Linux"
How do you feel about bash scripting?
good
You said good, I'm glad to hear that!
What are your favorite colours ?
blue green black
My favorite colours are also blue, green and black :-)
```

7) Использование ловушки> #!/bin/bash

объявляем ловушку

```
trap bashtrap INT
```

очищаем экран

```
clear;
```

функция ловушки выполняется, когда пользователь нажимает CTRL-C:

**На экран будет выводиться =>
Executing bash trap subroutine !**

**но скрипт будет продолжать
выполняться**

```
bashtrap()
{
echo "CTRL+C Detected !...executing bash trap !"
}
```

скрипт будет считать до 10

```
for a in seq 1 10; do
echo "$a/10 to Exit."
sleep 1;
done
echo "Exit Bash Trap Example!!!"
$ ./trap.sh
1/10
2/10
3/10
4/10
5/10
6/10
CTRL+C Detected !...executing bash trap !
7/10
8/10
9/10
CTRL+C Detected !...executing bash trap !
10/10
Exit Bash Trap Example!!!
```

Как видим, сочетание клавиш Ctrl-C не остановило выполнение скрипта.

8) Массивы

8.1) Объявляем простой массив> `#!/bin/bash`

**Объявляем простой массив с 4
элементами**

```
ARRAY=( 'Debian Linux' 'Redhat Linux' 'Ubuntu Linux' )
```

Получаем количество элементов в массиве

```
ELEMENTS=${#ARRAY[@]}
```

Выводим в цикле каждый элемент массива

```
for (( i=0;i<$ELEMENTS;i++)); do  
echo ${ARRAY[$i]}  
done
```

```
./arrays.sh  
Debian Linux  
Redhat Linux  
Ubuntu  
Linux
```

8.2) Заполняем массив значениями из файла> #!/bin/bash

Объявляем массив

```
declare -a ARRAY
```

Команда `exec <filename` перенаправляет ввод со `stdin` на файл. С этого момента весь ввод, вместо `stdin` (обычно это клавиатура), будет производиться из этого файла. Это дает возможность читать содержимое файла, строку за строкой, и анализировать каждую введенную строку с помощью `sed` и/или `awk`.

```
exec 10<bash.txt  
let count=0  
while read LINE <&10; do  
ARRAY[$count]=$LINE  
((count++))  
done  
echo Number of elements: ${#ARRAY[@]}
```

Вывод значений массива

```
echo ${ARRAY[@]}
```

закрываем файл

```
exec 10>&-
```

```
$ cat bash.txt
Debian Linux
Redhat Linux
Ubuntu
Linux
$ ./arrays.sh
Number of elements: 4
Debian Linux Redhat Linux Ubuntu Linux
```

9) Условия "если-то-иначе"

9.1) Простое использование "если-иначе" условий

Обратите внимание на пробелы в квадратных скобках, без которых условие работать не будет.> #!/bin/bash

```
directory="./BashScripting"
```

проверяем наличие директории

```
if [ -d $directory ]; then
echo "Directory exists"
else
echo "Directory does not exists"
fi
```

```
$ ./if_else.sh
Directory does not exists
$ mkdir BashScripting
$ ./if_else.sh
Directory exists
```

9.2) Вложенные "если-иначе" условия> #!/bin/bash

Объявляем переменную со значением 4

```
choice=4
```

Выводим на экран

```
echo "1. Bash"
echo "2. Scripting"
echo "3. Tutorial"
echo -n "Please choose a word [1,2 or 3]? "
```

Выполняем, пока переменная равна четырем

Заикливание

```
while [ $choice -eq 4 ]; do
```

читаем пользовательский ввод

```
read choice
```

вложенное "если-иначе" условие

```
if [ $choice -eq 1 ] ; then
echo "You have chosen word: Bash"
else
if [ $choice -eq 2 ] ; then
echo "You have chosen word: Scripting"
else
if [ $choice -eq 3 ] ; then
echo "You have chosen word: Tutorial"
else
echo "Please make a choice between 1-3 !"
echo "1. Bash"
echo "2. Scripting"
echo "3. Tutorial"
echo -n "Please choose a word [1,2 or 3]? "
choice=4
fi
fi
fi
done
```

```
$ ./nested.sh
1. Bash
2. Scripting
3. Tutorial
Please choose a word [1,2 or 3]?
5
Please make a choice between 1-3 !
1. Bash
2. Scripting
3. Tutorial
Please choose a word [1,2 or 3]?
4
Please make a choice between 1-3 !
1. Bash
2. Scripting
3. Tutorial
Please choose a word [1,2 or 3]?
3
You have chosen word: Tutorial
```

Таким образом сначала тело цикла "while" выполняется, т.к. переменная choice изначально равна четырём. Потом читаем в неё пользовательский ввод, и если ввод не равен 1,2 или 3 то

делаем нашу переменную снова равную 4, в связи с чем тело цикла повторяется (снова необходимо вводить 1,2 или 3).

10) Сравнения

10.1) Арифметические сравнения

-lt <
-gt >
-le <=
-ge >=
-eq ==
-ne !=

#!/bin/bash

Объявляем переменные с целочисленными значениями

```
NUM1=2
NUM2=2
if [ $NUM1 -eq $NUM2 ]; then
echo "Both Values are equal"
else
echo "Values are NOT equal"
fi
```

```
$ ./equals.sh
Both Values are equal
```

#!/bin/bash

Объявляем переменные с целочисленными значениями

```
NUM1=2
NUM2=3
if [ $NUM1 -eq $NUM2 ]; then
echo "Both Values are equal"
else
echo "Values are NOT equal"
fi
```

```
$ ./equals.sh
Values are NOT equal
```

#!/bin/bash

Объявляем переменные с целочисленными значениями

```
NUM1=2
NUM2=1
```

```
if [ $NUM1 -eq $NUM2 ]; then
echo "Both Values are equal"
elif [ $NUM1 -gt $NUM2 ]; then
echo "$NUM1 is greater then $NUM2"
else
echo "$NUM2 is greater then $NUM1"
fi
```

```
$ ./equals.sh
2 is greater then 1
```

10.2) Символьно-текстовые сравнения > = одинаковые
!= не одинаковые
< меньше чем

больше чем
-n s1 переменная s1 не пустая
-z s1 переменная s1 пустая

#!/bin/bash

Объявляем символьную переменную S1

S1="Bash"

Объявляем символьную переменную S2

```
S2="Scripting"
if [ $S1 = $S2 ]; then
echo "Both Strings are equal"
else
echo "Strings are NOT equal"
fi
```

```
$ ./statement.sh
Strings are NOT equal
```

#!/bin/bash

Объявляем символьную переменную S1

S1="Bash"

Объявляем символьную переменную S2

```
S2="Bash"
if [ $S1 = $S2 ]; then
echo "Both Strings are equal"
else
echo "Strings are NOT equal"
fi
```

```
$ ./statement.sh
Both Strings are equal
```

11) Проверка файлов> -b filename Block special file
-c filename Special character file
-d directoryname Check for directory existence
-e filename Check for file existence
-f filename Check for regular file existence not a directory
-G filename Check if file exists and is owned by effective group ID.
-g filename true if file exists and is set-group-id.
-k filename Sticky bit
-L filename Symbolic link
-O filename True if file exists and is owned by the effective user id.
-r filename Check if file is a readable
-S filename Check if file is socket
-s filename Check if file is nonzero size
-u filename Check if file set-ser-id bit is set
-w filename Check if file is writable
-x filename Check if file is executable

#!/bin/bash

```
file="./file"
if [ -e $file ]; then
echo "File exists"
else
echo "File does not exists"
fi
```

```
$ ls
file.sh
$ ./file.sh
File does not exists
$ touch file
$ ls
file file.sh
$ ./file.sh
File exists
```

Аналогично для примера мы можем использовать "в то время как" петли, чтобы проверить, если файл не существует. Этот сценарий будет спать, пока файл не существует. Обратите внимание на Bash отрицатель "!" что сводит на нет (инвертирует) -е опцию.

12) Циклы

12.1) Цикл For> #!/bin/bash

for цикл

```
for f in $( ls /var/ ); do
echo $f
done
```

Запуск for-цикла из командной строки bash:

```
$ for f in $( ls /var/ ); do echo $f; done
$ for f in $( ls /var/ ); do echo $f; done
backups
cache
crash
games
lib
local
lock
log
mail
opt
run
spool
tmp
www
```

12.2) While цикл> #!/bin/bash
COUNT=6

while цикл

```
while [ $COUNT -gt 0 ]; do
echo Value of count is: $COUNT
let COUNT=COUNT-1
done
```

```
$ ./while_loop.sh
Value of count is: 6
Value of count is: 5
Value of count is: 4
Value of count is: 3
Value of count is: 2
Value of count is: 1
```

12.3) Until цикл> #!/bin/bash
COUNT=0

until цикл

```
until [ $COUNT -gt 5 ]; do
echo Value of count is: $COUNT
let COUNT=COUNT+1
done
```

```
$ ./until_loop.sh
Value of count is: 0
Value of count is: 1
Value of count is: 2
Value of count is: 3
Value of count is: 4
Value of count is: 5
```

12.4) Циклы с неявными условиями

В следующем примере условием while-цикла является наличие стандартного ввода.

Тело цикла будет выполняться пока есть чему перенаправляться из стандартного вывода в команду read.> #!/bin/bash

Данный скрипт будет искать и удалять пробелы

в файлах, заменяя их на подчеркивания

DIR=". "

Управление циклом с командой read путем перенаправления вывода в цикл.
find \$DIR -type f | while read file; do

используем POSIX-класс [:space:] чтобы найти пробелы в именах файлов

if [["\$file" = [[:space:]]]]; then

замена пробелов подчеркиваниями

```
mv "$file" echo $file | tr ' ' '_'
fi;
done
$ ls -l
script.sh
$ touch "file with spaces"
$ ls -l
file with spaces
script.sh
$ ./script.sh
$ ls -l
file_with_spaces
script.sh
13) Функции> #!/bin/bash
```

Функции могут быть объявлены в любом порядке

```
function function_B {
echo Function B.
}
function function_A {
echo $1
}
function function_D {
echo Function D.
}
function function_C {
echo $1
}
```

Вызываем функции

передаем параметр в функцию function A

```
function_A "Function A."  
function_B
```

передаем параметр в функцию function C

```
function_C "Function C."  
function_D
```

```
$ ./functions.sh  
Function A.  
Function B.  
Function C.  
Function D.
```

14) Оператор выбора - Select> #!/bin/bash
PS3='Choose one word: '

select

```
select word in "linux" "bash" "scripting" "tutorial"  
do  
echo "The word you have selected is: $word"
```

Прерываем, в противном случае цикл будет бесконечный.

```
break  
done  
exit 0
```

```
$ ./select.sh  
1) linux  
2) bash  
3) scripting  
4) tutorial  
Choose one word: 4  
The word you have selected is: tutorial  
15) Оператор выбора - Case> #!/bin/bash  
echo "What is your preferred programming / scripting language"  
echo "1) bash"  
echo "2) perl"  
echo "3) python"  
echo "4) c++"  
echo "5) I do not know !"  
read case;
```

простая структура case-выбора

обратите внимание, что в данном примере \$case - всего лишь переменная

и не обязана так называться. Это лишь пример

```
case $case in
1) echo "You selected bash";;
2) echo "You selected perl";;
3) echo "You selected phyton";;
4) echo "You selected c++";;
5) exit
esac
```

```
$ ./case.sh
What is your preferred programming / scripting language
1) bash
2) perl
3) phyton
4) c++
5) I do not know !
4
You selected c++
```

[size=6]3. Используемая и рекомендуемая литература[/size]

Более подробную информацию можно получить из различных источников, например отсюда
оригинал: http://www.linuxconfig.org/Bash_scripting_Tutorial
http://ruslandh.narod.ru/howto_ru/Bash-Prog-Intro/
<http://bug.cf1.ru/2005-03-17/programming/bash/abs-book.html>