Session 10:
Bash Shell Scripting

Contents

- Command Execution: sequenced, parallel, grouped
- Arithmetic Expressions
- Conditional Statements: test statement, if-then-else, ...
- Loops: for, while
- Script Arguments
- Functions

Command Execution

- Sometimes we need to combine several commands.
- There are several formats for combining commands into one line: *sequenced, grouped, and conditional*.
- A sequence of commands can be entered on one line. Each command must be separated from its predecessor by semicolon.
- There is no direct relationship between the commands.

```
command1; command2; command3
```

- Parallel execution? Using &.

Grouped Commands

- Commands are grouped by placing them into parentheses
- Commands are run in a subshell
  - Subshell: a clone/child of current shell

Example:

```
echo "Month" > file; cal 10 2000 >> file
(echo "Month" ; cal 10 2000 ) > file
```

- What about { ... } ?

Conditional Expressions

- To perform *ifs* and *whiles* we need to be able to construct conditional expressions.
- A conditional expression is one that evaluates to true or false depending on its operands
- A process' return value of 0 is taken to be *true* ; any nonzero value is *false*

`test` – Conditional Expressions

- [is just shorthand
- Provides for a great many tests
- Is available to all shells

test *expression*

Or

`[ expression ]`

Spaces between expression and brackets are mandatory.

Conditional Expressions (cont)

- *test* returns an exit status of zero (success) if the expression evaluates to true.
- *test* uses a variety of operators
  - Unary file operators can test various file properties. Here are just a few:
    - -e True if file exists
    - -f True if file is a regular file
    - -d True if file is a directory
    - -w True if file exists and is writable
    - -O True if I own the file
  - E.g.

```
if [ -e ~kschmidt/public_html ] ; then
  echo "Kurt has a public web directory"
fi
```

[] – file and string operators

- Binary file operators "*file1 op file2*"
  - -nt True is *file1* is newer than *file2*
  - -ot True is *file1* is older than *file2*
  - -ef True if *f1* and *f2* refer to the same inode
- Unary string operators "*op string*"
  - -z True if string is of zero length
  - -n True if string is not of zero length
- E.g.

```
if [ -z "$myVar" ] ; then
  echo "\$myVar has null length"
fi
```

[] – string operators

- These compare lexical order
  - == != < > <= >=
  - Note, < > are file redirection. Escape them
- E.g.

```
if [ "abc" != "ABC" ] ; then
  echo 'See.  Case matters.' ; fi
if [ 12 \< 2 ] ; then
  echo "12 is less than 2?" ; fi
```

[] – arithmetic operators

- Only for integers
- Binary operators:

```
-lt  -gt  -le  -ge  -eq  -ne
```

- E.g.

```
if [ 2 -le 3 ] ; then ;echo "cool!" ; fi
x=5
if [ "$x" -ne 12 ] ; then
  echo "Still cool" ; fi
```

[] – Logical Operators

- Logical expression tools
  - ! *expression* Logical not (I.e., changes sense of expression)
  - *e1* -a *e2* True if both expressions are true.
  - *e1* -o *e2* True if *e1* or *e2* is true.
  - \(*expression* \) Works like normal parentheses for expressions; use spaces around the expression.

Examples:

`test -e bin -a -d /bin` is true

```
[ -e ~/.bashrc -a ! -d ~/.bashrc ] && echo
  true
```

[[*test*]]

- Bash added [[]] for more C-like usage.
  - More operators, but less portable.

```
if [[ -e ~/.bashrc && ! -d ~/.bashrc ]]
then
   echo "Let's parse that puppy"
fi

if [[ -z "$myFile" || ! -r $myFile ]]
…
```

- It's a built-in
- Why sometimes quote $myFile, sometimes not (it's usually a good idea to do so)?

Arithmetic Expressions

- Bash usually treats variables as strings.
- You can change that by using the arithmetic expansion syntax: ((*arithmetic expression*))
- (()) shorthand for the **let** built-in statement
  - `let x=x+1 OR ((x=x+1))`
- Arithmetic Expansion/Substitution:
  - `var1=$((var2+var3-var4))`
- Caution -> just integer calculations
- Use `bc' for floating point math:
  - `echo "scale=3; $x/($y*$z)" | bc`
  - `bc <<< "scale=3; $x/($y*$z)"`

if-then-else

```
if [ "$var" == "str" ]
then
  echo OK
fi

if [ "$var" == "str" ]
then
  echo "Value of var is str"
else
  echo "Value of var is not str"
fi
```

`else if` -> **`elif`**

Always quote variables in scripts!

How to rewrite in a single line?

Checking return value of a command

```
if diff "$fileA" "$fileB" > /dev/null
then
  echo "Files are identical"
else
  echo "Files are different"
fi
```

Exit status, `$?`

`/dev/null`, `-q` or `-s`

Conditional Commands: using && and ||

- We can combine two or more commands using conditional relationships AND (&&) and OR (||).
- If we AND two commands, the second is executed only if the first is successful.
- If we OR two commands, the second is executed only if the first fails.
- **`cp file1 file2 && echo "Copy successful"`**
- **`cp file1 file2 || echo "Copy failed"`**
- `cp file1 file2 || echo "Copy failed" && exit 1`

Case Statement

```
case $opt in
    a  ) echo "option a";;
    b  ) echo "option b";;
    c  ) echo "option c";;
    \? ) echo \
     'usage: alice [-a] [-b] [-c] args...'
          exit 1;;
esac
```

دستور کار: 3 - shell scripting

اسکریپتی بنویسید که:

1. تعداد فایل های واقع در دایرکتوری جاری را بدست آورده و در صورتی که این عدد کمتر از 5 باشد خروجی Less than 5، بشتر از 5 خروجی More than 5 و برابر 5 خروجی Equal to 5 تولید کند.

1. در صورت وجود فایل /var/log/syslog پیغام Log file exists را نمایش دهد. با استفاده از هر دوی if-then و && یا || این کار را انجام دهید.

2. در صورت وجود رشته 127.0.0.1 در فایل /etc/hosts پیغام Loopback is OK و در غیر این صورت Loopback not configured را نمایش دهد.

3. در صورتی که تعداد رخدادهای رشته ی authentication failure در فایل /var/log/auth.log بیشتر از 5 باشد، این فایل را به دایرکتوری /tmp/ کپی کرده و پیغام زیر را به انتهای فایل failed_logins.log اضافه کند:

<DATE>: Number of failed logins: <N>

با استفاده از هر دوی if-then و && یا || این کار را انجام دهید.

Loops: for and while

For samples

```
for x in 1 2 a
do
   echo $x
done
```

```
for x in *
do
   echo $x
done
```

```
for x in {1..4} {5..20..3}
do
   echo -n "$x "
done
1 2 3 4 5 8 11 14 17 20
```

For samples

```
for i in $(cat list.txt) ; do
  echo item: $i
done
```

```
for (( i=0; i<10; ++i )) ; do
  echo item: $i
done
```

While sample

```
COUNTER=0
while [  $COUNTER -lt 10 ] ; do
    echo The counter is $COUNTER
    let COUNTER=COUNTER+1
done
```

Iterate over Lines of a File

```
while read $Line
do
    # do sth with line
done <filename
```

Loop Control

- `break` terminates current loop
- `continue` causes a jump to the next iteration of the current loop

- `break n`: break n levels
- `continue n`: resume at the nth enclosing loop

دستور کار: 4 - shell scripting

- برای نوشتن اسکریپت‌های زیر از حلقه استفاده کنید.

اسکریپتی بنویسید که:

1. تعداد فایل‌های واقع در مسیر /usr/bin/ را که با هر یک از حروف شروع می‌شود در فایل /tmp/letter-no قرار دهد؛ یعنی به صورت زیر:

a: 50
b: 38
c: 32
...

2. تعداد فایل‌های واقع در مسیر /usr/bin/ که از اندازه‌ی آنها از 1MB بیشتر است را در خروجی چاپ کند.

- راهنمایی: دستور du با آپشن s– اندازه‌ی فایل داده شده برحسب کیلوبایت را به همراه نام فایل نمایش می‌دهد. برای استخراج اندازه از خروجی du از دستور awk '{print $1}' استفاده کنید.

Functions

- As in almost any programming language, you can use functions to group pieces of code in a more logical way or practice the divine art of recursion.
- Declaring a function is just a matter of writing function my_func { my_code }.
- Calling a function is just like calling another program, you just write its name.

Local variables

```
#!/bin/bash
HELLO=Hello
function hello {
    local HELLO=World
    echo $HELLO
}

$ echo $HELLO
$ hello
$ echo $HELLO
```

Functions with parameters sample

```
#!/bin/bash

function quit {
    echo 'Goodbye!'
    exit
}
function hello {
  echo "Hello $1"
}


for name in Vera Kurt ;
do
  hello $name
done
quit
```

Output:

```
Hello Vera
Hello Kurt
Goodbye!
```

Scripts and Arguments

- Scripts can be started with parameters, just like commands

  aScript arg1 arg2 …

- The scripts can access these arguments through shell variables:
  - "$*n*" Is the value of the n^{th} paramter.
    - The command is parameter zero
  - "$#" Is the number of parameters entered.
  - "$*" Expands as a list of all the parameters

Some Special Variables

- `$#` the number of arguments
- `$*` all arguments
- `$@` all arguments (quoted individually)
- `$?` return value of last command executed
- `$$` process id of shell

Handle User Input

- Use `read` command:
  - `read VAR1`
  - `read VAR1 VAR2`

Debugging Tip

- If you want to watch the commands actually being executed in a script file, insert the line "`set -x`" in the script.

```
set -x
for n in *; do
  echo $n
done
```

- Will display the expanded command before executing it:

```
+ echo bin
bin
+ echo mail
mail
```

Parameter Expansion

- `${parameter:-word}`
  - Use Default Values.
- `${parameter:=word}`
  - Assign Default Values.
- `${parameter:?word}`
  - Display Error if Null or Unset.

More Parameter Expansion

- We can remove parts of a value:
  - `${param#pattern}`
  - `${param##pattern}`
    - removes shortest (#) or longest (##) leading pattern, if there's a match
  - `${param%pattern}`
  - `${param%%pattern}`
    - removes shortest(%) or longest (%) trailing pattern, if match
- *pattern* is expanded just as wildcards - *, ?, [] – (not regexes).

More Parameter Expansion

- Find the length of a string:
  - `echo ${#foo}`
- Extract substrings
  - `echo ${foo:2:3}`
- Regex search and replace
- There are more. See the Bash manpage

Example – Parameter Expansion

```
$ foo=j.i.c
$ echo ${foo#*.}
i.c
$ echo ${foo##*.}
c
$ echo ${foo%.*}
j.i
$ echo ${foo%%.*}
j
```

دستور کار: 4 - shell scripting

1. اسکریپتی بنویسید که دو عدد را به عنوان پارامتر گرفته، که اولی باید کوچکتر یا مساوی دومی باشد، و در خروجی فهرست اعداد بین این دو را (شامل دو عدد ورودی) چاپ کند.
2. یک دایرکتوری ایجاد کرده و به آن وارد شوید. چند فایل در آن ایجاد کنید. اسکریپتی بنویسید که دو رشته را از ورودی گرفته و همه ی فایل های واقع در دایرکتوری جاری را با استفاده از دو رشته ی ورودی تغییر نام دهد. رشته ای اول را پیشوند و رشته ی دوم را پسوند در نظر بگیرید؛ مثلاً:

```
./myrename.sh trip jpg
file1,file2, … becomes trip1.jpg, trip2.jpg,…
```

اسکریپت قبل از انجام کار باید به کاربر هشدار داده و از وی تایید بگیرد.

Last Note: Text Processing Tools

- awk, cut, tr, sed, grep, sort, wc, ...