Mahasarakham University

มหาวิทยาลัยมหาสารคาม

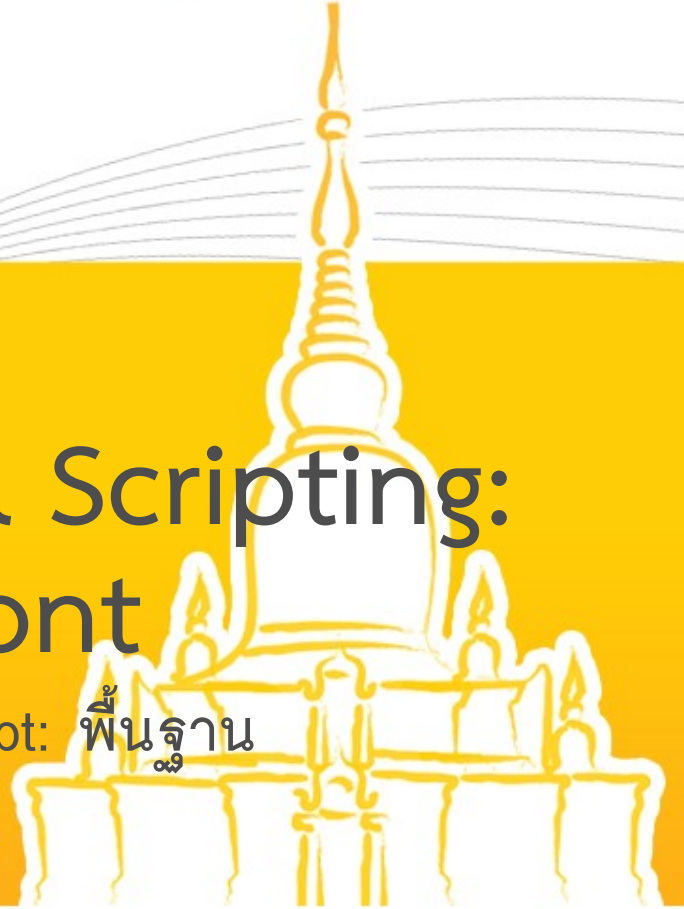
Heart of
the Northeast

www.msu.ac.th

Mahasarakham University
มหาวิทยาลัยมหาสารคาม

Introduction to Shell Scripting: The Basics: cont

แนะนำการทำงานของ Shell Script: พื้นฐาน



Heart of the Northeast

www.msu.ac.th

โดย อาจารย์โอฬาริก สุรินต๊ะ

Jan 16, 2017

Agenda

- Basic Unix commands
- 10 essential Unix commands
- Unix command line & trick
- Flow control: if
- Case statement

Basic Unix Commands

- The **UNIX** operating system has for many years formed the backbone of the Internet, especially for large servers and most major university campuses.
- A free version of UNIX called **Linux** has been making significant gains against Macintosh and the Microsoft Windows environments, so often associated with personal computers.

Basic Unix Commands

- Developed by a number of volunteers on the Internet such as the Linux group and the GNU project, much of the open-source software is copyrighted, but available for free.
- This is especially valuable for those in educational environments where budgets are often limited.

10 essential UNIX commands

Command	Example	Description
1) ls	$ ls	Lists files in current directory
	$ ls -alF	List in long format
2) cd	$ cd tmpDir	Change directory to **tmpDir**
	$ cd ..	Move back one directory
	$ cd ~olarik/web-docs	Move into olarik's web-docs directory ***olarik = user***

Unix Command Line: --help

- **--help Online manual (help) about command**

$ ls --help

Usage: ls [OPTION]... [FILE]...

List information about the FILEs (the current directory by default).

Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.

-a, --all	do not ignore entries starting with .
-A, --almost-all	do not list implied . and ..
--author	with -l, print the author of each file

10 essential UNIX commands

Command	Example	Description
3) mkdir	$ mkdir tempDir	Make a directory called **"tempDir"**
	$ mkdir tmpDir1 tmpDir2 tmpDir3 tmpDir4	Make multiple directories; tmpDir1, tmpDir2, tmpDir3 and tmpDir4
4) rmdir	$ rmdir tmpDir1	Remove directory (must be empty)
	$ rmdir tmpDir2 tmpDir3	Remove multiple directories; tmp2 and tmp3
5) rm	$ rm file.txt	Remove or delete file (file.txt)
	$ rm file1.txt file2.txt	Remove multiple files; file1.txt and file2.txt
	$ rm file*.txt $ rm *.txt	Remove multiple files
	$ rm -r tmpDir4	Remove non empty directory

10 essential UNIX commands

Command	Example	Description
6) cp	$ cp file1.txt tempDir/	Copy file into directory
	$ cp file1.txt file2.txt tempDir/	Copy multiple files into directory
	$ cp file1.txt file2.txt	Make copy/backup of file1.txt
7) mv	$ mv file1.txt file2.txt	Rename files
	$ mv file2.txt tempDir/	Move source(s) to directory
8) more	$ more file1.txt	Look at file, one page at a time
9) man	$ man ls	Online manual (help) about command
10) lpr	$ lpr file1.txt	Send file t o printer

Unix Command Line: history

- **History history – GNU History Library**

```
$ history
  2006   cal 9 2000
  2007   cal 10 1978
  2008   history
  2009   cal 10 1978
  2010   history --help
  2011   man history
```

Unix Command Line: grep

- **grep - Search for PATTERN in each FILE of standard input.**

```
$ history | grep "echo"
1946   echo "$((expr 3 '*' '(' 2 '+' 1 ')'))"
1984   echo ${BASH_VERSION}
1991   echo $name
1992   echo ${name[*]}
1993   echo ${name[@]}
```

Unix Command Line: grep

- **grep**, which stands for "global regular expression print," processes text line by line and prints any lines which match a specified pattern.
- Syntax:
  - grep [OPTIONS] PATTERN [FILE...]

Unix Command Line: grep

$ grep --help > manual.txt

- Sending output of "grep --help" to manual.txt

$ cat -n manual.txt

- Concatenate FILE(s) to standard output, show detail of selected file

$ tail manual.txt

– Print the last 10 lines of each FILE to standard output

$ grep "print" manual.txt

-l, --files-with-matches print only names of FILEs containing matches
-c, --count print only a count of matching lines per FILE
-Z, --null print 0 byte after FILE name
-B, --before-context=NUM print NUM lines of leading context
-A, --after-context=NUM print NUM lines of trailing context
-C, --context=NUM print NUM lines of output context

Unix Command Line: grep

$ grep --color -n "print" manual.txt

```
49:  -l, --files-with-matches   print only names of FILEs containing matches
50:  -c, --count                print only a count of matching lines per FILE
52:  -Z, --null                 print 0 byte after FILE name
55:  -B, --before-context=NUM   print NUM lines of leading context
56:  -A, --after-context=NUM    print NUM lines of trailing context
57:  -C, --context=NUM          print NUM lines of output context
```

Unix Command Line: grep

$ grep -H "print" manual.txt

```
manual.txt:   -b, --byte-offset             print the byte offset with output lines
manual.txt:   -n, --line-number             print line number with output lines
manual.txt:   -H, --with-filename           print the file name for each match
manual.txt:   -L, --files-without-match     print only names of FILEs containing no match
manual.txt:   -l, --files-with-matches      print only names of FILEs containing matches
manual.txt:   -c, --count                   print only a count of matching lines per FILE
manual.txt:   -Z, --null                    print 0 byte after FILE name
manual.txt:   -B, --before-context=NUM      print NUM lines of leading context
manual.txt:   -A, --after-context=NUM       print NUM lines of trailing context
manual.txt:   -C, --context=NUM             print NUM lines of output context
```

Unix Command Line: grep

$ls

tmp1.txt tmp2.txt tmp3.txt tmp4.txt

$ grep --color -i -n "Our*" tmp*.txt

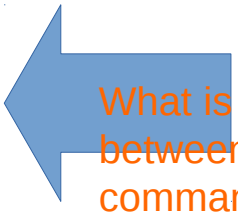
tmp2.txt:3: We can extend **our** search to subdirectories and any files they

tmp2.txt:5:search recursively. Let's change **our** FILE name to just an

tmp4.txt:4:a wildcard in **our** FILE name. Instead of specifying product-listing.html,

$ grep --color -n "our.*" tmp*.txt

$ grep --color -i -n "our.*" tmp*.txt



What is the difference between these 2 commands?

-i is case-insensitive

Unix Command Line: grep

- In regular expressions, the period (".") is interpreted as a single-character wildcard. It means any character that appears in this place will match.
- The asterisk ("*") means the preceding character, appearing zero or more times, will match.
- The combination ".*" will match any number of any character.
  - For instance, **"our amazing products"**, **"ours, the best-ever products"**, and even **"ourproducts"** will match.

Unix Command Line: grep

$ grep --color -rn "in.*is" *

tmp1.txt:3:appears **in this place will match." The asteris**k ("*") means

tmp4.txt:4:a wildcard **in our FILE name. Instead of specifying product-lis**ting.html,

tmpDir2/tmp5.txt:4:a wildcard **in our FILE name. Instead of specifying product-lis**ting.html,

Unix Command Line: trick

- **cal** displays a calendar of a specific month and year

```
$ cal 10 1978
      October 1978
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

Unix Command Line: trick

- **Ncal** displays calendar of a current month and year

```
      January 2017
Mo        2  9 16 23 30
Tu        3 10 17 24 31
We        4 11 18 25
Th        5 12 19 26
Fr        6 13 20 27
Sa        7 14 21 28
Su  1     8 15 22 29
```

Unix Command Line: trick

$ ispell

The program 'ispell' can be found in the following packages:

* tmispell-voikko
* ispell

Try: sudo apt install <selected package>

$ sudo apt install ispell

Unix Command Line: trick

$ ispell tmp1.txt

instanceds File: tmp1.txt

will match." So the combination ".*" will match any number of any character. For instanceds,

0: instance's
1: instanced
2: instanced s
3: instanced-s
4: instances

[SP] <number> R)epl A)ccept I)nsert L)ookup U)ncap Q)uit e(X)it or ? for help

Unix Command Line: trick

$ tar –help

Usage: tar [OPTION...] [FILE]...

GNU 'tar' saves many files together into a single tape or disk archive, and can

restore individual files from the archive.

Examples:

tar -cf archive.tar foo bar # Create archive.tar from files foo and bar.

tar -tvf archive.tar # List all files in archive.tar verbosely.

tar -xf archive.tar # Extract all files from archive.tar.

Unix Command Line: trick

- **$ tar -cf subdir.tat subdir**
  - Create an archive called subdir.tar of a directory
- **$ tar -xvf subdir.tar**
  - Extract files from an archive file

Unix Command Line: trick

- **df** - See how much free disk space
- **du** – Estimate disk usage of directory in Bytes
  - **du -b subdir**
  - **du -b**

Unix Command Line: trick

- nautilus
- gimp
- top
- htop

Flow Control: If

- The if command is fairly simple on the surface; it makes a decision based on the exit status of a command.
- The if statement has the following syntax:

Flow Control: If

if commands; **then**

commands

*[**elif** commands; **then***

commands...]

*[**else***

commands]

fi

Flow Control: If

- **Exit Status**
  - Commands issue a value to the system when they terminate, called an exit status.
  - This value, which is an integer in the range of **0** to **255**, indicates the success or failure of the command's execution.
    - A value of **zero** indicates **success** and
    - **Any** other value indicates **failure**

Flow Control: If

$ ls -d /usr/bin/

/usr/bin/

$ echo $?

0

$ ls -d /bin/usr

ls: cannot access '/bin/usr': No such file or directory

$ echo $?

2

Flow Control: If

```
$ true
$ echo $?
0
$ false
$ echo $?
1
$ if true; then echo "It's true."; fi
It's true.
```

Flow Control: If

- **test**
  - The **test** command is used most often with the **if** command to perform true/false decision.

```
# First form
test expression
# Second form
[ expression ]
```

Flow Control: If

```
if [ -f .bash_profile ]; then
    echo "You have a .bash_profile. Things are fine."
else
    echo "Yikes! You have no .bash_profile!"
fi
```

Flow Control: If

$ help test

File operators:

-a FILE	True if file exists.
-b FILE	True if file is block special.
-c FILE	True if file is character special.
-d FILE	True if file is a directory.
-e FILE	True if file exists.
-f FILE	True if file exists and is a regular file.
-g FILE	True if file is set-group-id.
-h FILE	True if file is a symbolic link.
-L FILE	True if file is a symbolic link.
-k FILE	True if file has its `sticky' bit set.
-p FILE	True if file is a named pipe.
-r FILE	True if file is readable by you.
-s FILE	True if file exists and is not empty.
-S FILE	True if file is a socket.
-t FD	True if FD is opened on a terminal.
-u FILE	True if the file is set-user-id.
-w FILE	True if the file is writable by you.
-x FILE	True if the file is executable by you.
-O FILE	True if the file is effectively owned by you.
-G FILE	True if the file is effectively owned by your group.
-N FILE	True if the file has been modified since it was last read.

exit

- In order to be good script writers, we must set the exit status when our scripts finish.
- To do this, use the **exit** command.

exit: example

```
#!/bin/bash

echo hello
echo $?     # Exit status 0 returned because command executed successfully.

lskdf       # Unrecognized command.
echo $?     # Non-zero exit status returned -- command failed to execute.

echo

exit 113    # Will return 113 to shell.
            # To verify this, type "echo $?" after script terminates.

#  By convention, an 'exit 0' indicates success,
#+ while a non-zero exit value means an error or anomalous condition.
#  See the "Exit Codes With Special Meanings" appendix.
```

exit: example

```
#!/bin/bash

echo  hello
echo  $?
exit  $?          #or  exit
```

if & exit: example

```
#!/bin/bash

echo hello
ex="$?"

if [ $ex == 0 ]
then
        echo "Exit status = 0"
fi

lskdf
ex="$?"

if [ $ex != 0 ]
then
        echo "Exit status = $ex"
        exit $?
fi

echo "list directories and files"
ls
```

if: example.sh

```
#!/bin/bash

# Basic if statement
if [ $1 -gt 100 ]
then
        echo Hey that's a large number.
        pwd
fi
        date
```

$ bash example.sh
ex.sh: line 3: [: -gt: unary operator expected
Tue 17 Jan 23:39:44 ICT 2017

$ bash example.sh 15
Tue 17 Jan 23:40:27 ICT 2017

$ bash example.sh 150
Hey that's a large number
/home/mrolarik/Desktop/tmpDir
Tue 17 Jan 23:41:59 ICT 2017

nested-if: example

```
#!/bin/bash

if [ $1 -gt 100 ]
then
        echo "Hey that's a large number"

        if (( $1 % 2 == 0 ))
        then
                echo "And is also an even number"
        fi
fi
```

if-else: example

```
#!/bin/bash

Count=99

if [ $count -eq 100 ]
then
   echo "Count is 100"
else
   echo "Count is not 100"
fi
```

if-elif: example.sh

```
#!/bin/bash

if [ $1 -ge 18 ]
then
        echo "You may go to the party."
elif [ $2 == 'yes' ]
then
        echo "You may go to the party but be back before midnight."
else
        echo "You may not go to the party."
fi
```

Boolean operation

- and - &&
- or - ||

-r FILE True if file is readable by you.
-s FILE True if file exists and is not empty.

```
#!/bin/bash

if [ -r $1 ] && [ -s $1 ]
then
        echo This file is useful.
fi
```

Case Statement

```
case <variable> in
<pattern 1>)
    <commands>
    ;;
<pattern 2>)
    <other commands>
    ;;
esac
```

case: example.sh

```
#!/bin/bash
# case example

case $1 in
      start)
         echo starting
         ;;
      stop)
         echo stoping
         ;;
      restart)
         echo restarting
         ;;
      *)
         echo don\'t know
         ;;
esac
```

$ bash example.sh start
starting

How can we input number 1 and return word “starting”?
$ bash example.sh 1
starting

case: condition of patterns

?() - zero or one occurrences of pattern

*() - zero or more occurrences of pattern

+() - one or more occurrences of pattern

@() - one occurrence of pattern

!() - anything except the pattern

case: example

```
case $1 in
    a*              ) foo;;     # matches anything starting with "a"
    b?              ) bar;;     # matches any two-character string starting with "b"
    c[de]           ) baz;;     # matches "cd" or "ce"
    me?(e)t         ) qux;;     # matches "met" or "meet"
    @(a|e|i|o|u)    ) fuzz;;    # matches one vowel
esac
```

References:

- https://www.tjhsst.edu/~dhyatt/superap/unixcmd.html
- http://www.computerhope.com/unix/ugrep.htm
- http://linuxcommand.org/lc3_wss0080.php
- http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html

References:

- http://tldp.org/LDP/abs/html/exit-status.html
- http://ryanstutorials.net/bash-scripting-tutorial/bash-if-statements.php
- http://www.thegeekstuff.com/2010/07/bash-case-statement

References:

- http://wiki.bash-hackers.org/syntax/ccmd/case
- http://stackoverflow.com/questions/455471 8/patterns-in-case-statement-in-bash-scripting