Linux Shell

Jan Pribošek

7.4.2017

I. Basic commands

MAN

Manual pages - built in documentation

```
# usage
man [command]

man man
man ls
```

LS

List directory contents

```
# usage
ls -[options]

ls
ls -a      # display hidden files
ls -la     # display hidden files in long format
ls -la -r # display all in reverse order
ls -la -h # display file sizes in human readable format
```

- hidden files: dotfiles [.filename]
- file extensions: no file extensions, everything in Linux is file
- naming files: use [file_name], [fileName], [file-name] instead of spaces

PWD

Print working directory

```
pwd
```

SHEBANG LINE

Shebang line is used to tell the system the name of the interpreter that should be used to execute the script

```
#!/bin/bash
#!/usr/bin/python3
```

CD

Change directory

```
# usage
cd [path]

cd        # change to home folder
cd -      # change to previous working directory
cd ..     # change to parent folder
```

```
# Tips

/var/log  # absolute path - begins with root directory
var/log   # relative path - begins with current working directory

.         # working directory
..        # working directory parent directory

(TAB)     # autocomplete
(TAB,TAB) # show all possibilities
(Select,middle mouse button)  # copy & paste
```

CAT

Concatenate files and print on standard input

```
# usage
cat [file1] [file2] [file3] ...

# print from bottom to top
tac [file1] [file2] [file3] ...
```

TAIL

Display last n lines

```
# usage
tail -n 5 [file]    # display last 5 lines of file
```

HEAD

Display first n lines

```
# usage
head -n 3 [file]   # display first 3 lines of file
```

II. Manipulating files and directories

TOUCH

Create new empty file

```
touch [filename]
```

MKDIR

Create new empty folder

```
mkdir [directory]
mkdir [dir2] [dir2] [dir3]
```

CP

Copy files

```
# usage
cp [/path/to/input] [/path/to/output]

cp [file1] [file2] [dir]
cp -r [dir1] [dir2]     # copy recursively (when copying directories)
cp -u [input] [output]  # copy only files that do not exist or are newer
```

MV

Move or rename files

```
# usage
mv [path/to/input] [path/to/output_folder]

# rename file
mv ~/Documents/file123 ~/Documents/file
```

RM

Remove files or folders

THIS IS DANGEROUS COMMAND, BE CAREFUL!

```
# Be careful when using these
rm [file]          # remove file
rm -r [folder]     # remove folder recursively
rm -rf [folder]    # remove folder recursively, and do not prompt

rm -r -i [folder] # prompt before every removal
rm -r -I [folder] # prompt once
```

III. Intermediate commands

WILDCARDS

Select filenames based on patterns

```
*          # Any characters
?          # Any single character
[:alnum:] # Any alphanumeric character
[:alpha:] # Any alphabetic character
[:digit:] # Any digit
[:lower:] # Any lowercase letter

           # There are more but we are ending here
```

```
# Examples:
a*             # show everything that starts with a
*.txt          # show everything that ends with .txt
a*.txt         # Any file beginning with 'a' and ending with .txt
ab??def        # File: ab + "any 2 characters" + def
[[:digit:]]*   # Any file starting with digit
[![:lower:]]* # Any file not starting with lowercase
```

GREP

Print lines matching a pattern (filter)

```
grep [text]     # display lines containing [text]
grep -v [text] # display lines not containing [text]
grep -E '[exp1]|[exp2]|[exp3]' # display lines containing exp1 or exp2 or exp3
```

PIPE

Piping standard output of left command to standard input of the right command

```
# pipe from left to right command
cmd1 | cmd2 | cmd3 | cmd 4

cat [file] | grep [text]  # read file and print lines matching a pattern

# helpers
uniq    # removes any duplicates from the list
uniq -d # show only duplicates
sort    # sorting output of commands

[cmd] | sort | uniq   # show sorted unique lines of cmd output
```

REDIRECT (>, >>)

Used to redirect standard output to files. All errors are sent to standard error.

```
[cmd] [operator] [filename]

ls -l /usr/bin > output.txt   # text file is always overwritten
ls -l /usr/bin >> output.txt  # output is appended to text file
ls -l /bin/usr 2> error.txt   # 2> is used to redirect error stream
ls -l /usr/bin &> output.txt  # redirect both outputs to text file
[cmd] 2> /dev/null            # throw away unwanted output
```

ARCHIVING

Archiving and compressing

```
# c - create, x - extract, v - verbose, f - file, - standard IO
tar cvf archive.tar file1 file2 # create archive
tar xvf archive.tar             # unpack tar
gzip [file]                     # compress
gunzip [file]                   # uncompress file.gz
# one liner
tar z[xvf/cvf] file.tar.gz      # zxvf -> extract; zcvf -> create tar.gz

zip [name.zip] file1 file2      # zip file1 file2 into name.zip
unzip [name.zip]                # unzip name.zip
zip -r [name.zip] dir1 dir2     # zip directories
```

CHMOD

Change permissions of file or folder

A) Octal representation (base 8)

Number	Meaning
0	- - -
1	- - x
2	- w -
3	- w x
4	r - -
5	r - x
6	r w -
7	r w x

```
chmod [owner group other] [file]

# Examples
chmod 777 build_script.sh
chmod 755 build_script.sh

chmod -R 000 /some/directory   # change permissions recursively
```

B) Symbolic representation

User	Meaning	Operators	
u	owner	+	Add
g	group	-	Remove
o	other	=	Only specified permission should be applied
a	all		

```
chmod [user][operator] [file]

# Examples:
chmod a+x build_script.sh  # Added execution rights to all
chmod g-w build_script.sh  # Remove write permissions from group
chmod u-x build_script.sh  # Remove execute permissions from owner
```

Use what works for you

SCP

Moving files over network

```
# local files to remote
scp [file1] [file2]  [username]@[server_ip]:[/path/to/remote/folder]
# remote files to local
scp [username]@[server_ip]:[filename] [/local/folder]
# copying local directory to remote server
scp -r [directory] [username]@[server_ip]:[/path/to/remote/folder]

# if you are transferring large numbers of files often, take a look at rsync
```

IV. Advanced commands

AWK

Awk is a programming language. Nowadays it's mostly used for picking a single field out of an input stream. For complicated string manipulations use Python or Perl.

```
# usage:
[cmd] | awk [awk_cmd]  # or
awk [awk_cmd] [file]

awk '{print $1}'               # print only the first field of each line from file
awk '/[expr]/ {print $1}'      # print first field of lines containing [expr]
awk '/^[expr]/ {print $1}'     # match beggining of the field with expr
ps aux | awk 'length($0) > 40' # displays lines from ps aux longer than 40 chars
awk -F: '{print $1}' /etc/passwd | sort # display sorted list of login user names

-F: # change field separator to : (default is space)
```

SED

Takes an input stream, change it according to the expression and prints results to standard input

```
sed 's/[exp]/[text]'   # substitute expression with text
sed '/[exp]/d'           # deletes line that matches regular expression exp

# Examples
sed 's/:/%/g'          # substitute [:] with [%] globally
sed 2,4d               # delete 2nd to 4th line
```

V. Shell scripts

COMMAND SUBSITUTION

Use bash commands in bash scripts

```
$(command)

# Example:
echo $(uname -a) # prints system information
echo $((1+5))    # note the double braces around arithmetic expressions

echo `uname -a`  # alternative notation
```

BRACE EXPANSIONS

Execute the same command

```
echo folder_{1, 2, 3}
echo folder_{1..5}
echo folder_{A..Z}
```

QUOTES

A) Double quotes (")

All special characters lose their meaning, exceptions: `$, \, '`

```
echo "$(hostname) computer uptime: $(uptime)"
```

B) Single quotes (')

Supress all expansions

```
echo '$(hostname) computer uptime: $(uptime)'
```

SHELL SCRIPT

Shell scripts are meant to be small, aimed at manipulating files and executing simple commands. When complicated string manipulation is needed use high level scripting language such as Python or Perl.

VI. Tips & Tricks

1. Subshell

```
(cd [path] && [cmd]) # execute this and jump to current working directory
```

2. Bang Bang

```
!!      # execute previous command
sudo !! # execute previous command with sudo priviliges
```

3. Quick copy

Share files to another computer on localhost

```
# start basic web server on port 8000 from current working directory
python -m SimpleHTTPServer  # python2
python3 -m http.server      # python3
```