

LECTURE NOTES IN CIS300
YUZHE (RICHARD) TANG
SPRING, 2018

SECTION 1: BASH

REFERENCES

- "Basic UNIX commands" [[link](#)]
- "Bash Guide for Beginners" [[link](#)]
- "Advanced Bash-Scripting Guide" [[link](#)]

GETTING STARTED

Access Shell terminal in your computer

- Option 1: Web terminal
 - [<http://www.webminal.org/terminal/>]
- Option 2: Setting up Ubuntu through VirtualBox
 - TA will talk about this.

LECTURE 2: FILES & DIRECTORIES

DIRECTORIES

- List files and directories: `ls`
 - `ls ~`: list files in home directory
 - `ls .`, `ls`: current directory
 - `ls /`: root directory
 - `ls -al`: list all files in "long" format
- Enter a directory: `cd`
 - `cd ..`: go to parent directory
 - `cd`, `cd ~`: home directory
 - `cd /`: root directory

DIRECTORIES (2)

- Print the current pathname: `pwd`
- Create a directory: `mkdir`
 - `mkdir dir_a`

BASIC FILE MANAGEMENT (2)

- Show the content of a file: `cat`, `more`
 - `cat file_a`
 - `more file_a`: use `q` to quit, `/` to search
 - Write text to a file: `echo >>`
 - `echo "Alice Bob" >> file_a`
 - `echo "Alice" >> file_b,`
`echo "Alice" >> file_c`
- Show the count of lines/words/chars a file: `wc`
 - `wc file_a`
- Show difference between files: `diff`
 - `diff file_a file_b`

BASIC FILE MANAGEMENT

- Create a file: `touch`
 - `touch file_a`
- Move a file (change file name): `mv`
 - `mv file_a file_b`
- Copy a file: `cp`
 - `cp file_a file_b`
- Remove a file: `rm`
 - `rm file_a`

EXERCISE 2.1

1. Run command `ls -a /`. Copy and paste (C&P) the printout on BB.
2. Run command `cat file_b`. C&P printout on BB.
3. Create a directory `dir_b` under `dir_a` and enter it. C&P the commands on BB.
4. Create a text file named `file_d.txt` and put there the following string: `Charlie is a student`. Run `cat file_d.txt`.
 - C&P the list of commands and their printout on BB

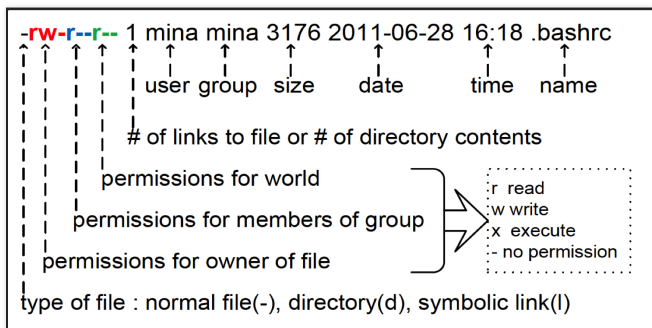
LECTURE 3: FILE PERMISSION

REFERENCES

- Understanding Linux file permissions [[link](#)]

BASIC CONCEPT

- file permission: access right, or file mode
 - permission controls the ability of a *user* to take *actions* on a *file*
 - user: owner, group, all users
 - group: group of users and files.
 - type: read, write, execute



`ls -al`

VIEWING PERMISSION

`ls -l`

- owner and group
- permissions
 - users: owner (u), group (g), others (o), all users (a)
 - type: read (r), write (w), execute (x)

CHANGING PERMISSION

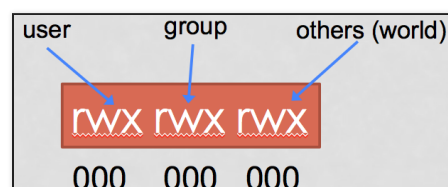
- `chmod`: change mode
 - add +:
 - `chmod a+wx file_a`: add write/execute permission to all users
 - `chmod g+r file_a`: add read permission to group users
 - assign/copy =:
 - `chmod g=rw file_a`: assign read/write permission to group
 - `chmod g=u file_a`: copy owner permission to group permission

CHANGING PERMISSION (2)

Options	Definitions
u	Owner
g	Group
o	Other
a	All (same as ugo)
x	Execute
w	Write
r	Read
+	Add permission
-	Remove permission
=	Set permission

CHANGING PERMISSION: NUMERIC MODE (3)

- `chmod 777 file_a; chmod a+rxw file_a`
 - `chmod 666 file_a; chmod a=rw file_a`
 - `chmod 000 file_a; chmod a-rwx file_a`



CHANGE OWNERSHIP

- `chown owner:group filename`
 - `chown user1:staff file_a`

EXERCISE 3.1

1. Run command `chmod o-r file_a; cat file_a`. C&P the printout on BB.
2. Design the command to make a file read-only to group. C&P your command on BB.
3. Design the command to make a file read-only to all users. C&P your command on BB.
4. Convert the following two commands to numeric mode:
`chmod a-rwx file_a; chmod o+x file_a`. C&P your command on BB.

LECTURE 4: TEXT EDITING

- `gedit`: text editor with GUI
 - `gedit filename &`
- `vim`: text editor in terminal
- other editors: `emacs`, etc.

VIM

- basic movement (character-wise): `h,j,k,l`
 - word movement: `w,e,b`
 - word, end, begin
 - number powered movement: `5w`
- find character in current line: `f`
 - `fq`: find char `q` in current line
 - `2fn`: find the second char `n` in current line
 - find word under cursor: `*` (next) and `#` (previous)
 - go to matching parentheses: `%`

VIM (2)

- begin and end of line: `0` and `$`
- go to line: `g`
 - first line: `gg`
 - last line: `G`
 - 10th line: `10gg`
- search: `/keyword` with `n` and `N`

VIM (3)

- modes: normal and insert
- from normal to insert: i, o, R
 - backward: esc
- editing in normal mode
 - copy/yank : v+y
 - and paste: p
 - cut: v+x
 - delete: v+d
 - undo/redo: u,R
- save/exit a file
 - write: :w
 - quit: :q

EXERCISE

1. Write down the action sequence that searches String Alice in a file opened in vim
2. Open file_a.txt using vim. Insert your name in the file, save it, and close the file. C&P the actions you used.
3. What do the following action sequence do? (You can test it in vim)
 - v+p
 - 6e
 - j

REFERENCE

- "Bash Guide for Beginners" [[link](#)]

DEMO/PRACTICE

- Install vim on your VM: `sudo apt-get install vim`
- Or use online Vim: <http://www.openvim.com/>

LECTURE 5: SHELL SCRIPTING (1)

GETTING STARTED

Access Shell terminal in your computer

- Option 1: Web terminal
 - [<http://www.webminal.org/terminal/>]
- Option 2: Setting up Ubuntu through VirtualBox
 - TA will talk about this.

Lecture 2: Files & directories Introduction: Shell ---

- Linux Shell
 - Linux shell is a *program* that interprets user *commands* from users and execute them by interacting with Linux OS kernel.
 - Different distributions/versions:
 - `sh`: Bourne shell
 - `bash`: Bourne Again shell, superset of `sh`
 - others: `ksh`, `csk`

INTRODUCTION: SHELL SCRIPTING

- Two ways to put shell commands
 1. terminal: run commands one by one
 2. scripting: put shell commands in a file
- Why learn shell scripting/programming?
 - automate administrative tasks, save your efforts!
 - e.g. automatic software update, file backup, resource monitoring

SHELL SCRIPT BASICS: SHA-BANG

- Your first script
 - `#!` sha-bang is a two-byte magic number
 - basically says it's an executable shell script
- To execute a script `script.sh`:
 - `./script.sh`
 - `source script.sh`

SHELL SCRIPT BASICS: LANGUAGE

- Script is a group of commands:
 1. `#!/bin/bash echo 'hello world';`
- Variable:
 - Reference a variable: `$a`
 - untyped: integer, char, etc.
- 2. `#!/bin/bash a=1;b=2;a=$a+$b;echo $a;`
- 3. `#!/bin/bash a=1;b=2; a=$((a+b));echo $a;`
- If/else
 3. `if [$a -gt $b];then echo 'a larger than`
 - `fi` declares the end of if/else clause
 - there is mandatory whitespace in `[$a (and $b)]`

EXERCISE

1. Put commands `a=1;b=2;c=$a;a=$b;b=$c;echo $a,$b;` in a script, execute it, and put the printout to BB.
2. Write a script to initialize variables `a`, `b`, `c` with 1,2,3, and print their sum.
3. Write a script to swap the names of two files, `file1` and `file2`. For example if input `file1` contains Alice and `file2` contains Bob at the beginning, after the execution, `file1` should contain Bob and `file2` should contain Alice.

COMMENTING

- `#` is used to comment in bash

LECTURE 6: SHELL SCRIPTING (2)

SHELL INITIALIZATION

- `~/.bash_profile`
 - The script runs when you open a terminal (CTRL+T) or so-called login
 - sample: `export PS1='\W> '`
- `~/.bashrc` (bash run commands)
 - The script runs when you run a new bash program (bash)
- These are user configuration files (not system-wide)

SHELL VARIABLES

- Global versus local
 - Global: environment variables
 - `env` to list all environment variables system-wide.
 - Global variable is propagated through all children bash, local var isn't
 - `export gvar=1; bash; echo $gvar; exit`
 - `lvar=1; bash; echo $lvar; exit`
- naming convention:
 - shell var name include char and digit
 - name does not start with digit: `1x=5` is invalid

SHELL VARIABLES: LIFE CYCLE

life cycle	local var	global var
define & init	<code>lvar=6</code>	<code>export gvar=7</code>
reference	<code>echo \$lvar</code>	<code>echo \$gvar</code>
destroy	<code>unset lvar</code>	

RESERVED AND SPECIAL VARIABLES

- Bash reserved variables:
 - `echo $HOME`
 - `echo $PATH`
 - `echo $PS1`: Prompt String
 - `echo $BASH; $BASH_VERSION`
- Bash special parameters:
 - `echo $?`: exit status of the last command executed
 - `echo alice; echo $?;`
 - `rm nonexistentfile; echo $?`

PASSING PARAMETERS

- Bash special parameters:
 - `echo $0`: name of shell
 - `echo $1, $2`: 1th, 2nd shell parameter
 - `echo $*, $#`: all positional parameters and the number of these parameters
- Demo:
 - `#!/bin/bash echo $1; echo $2; echo $#;`

EXERCISE

1. Run script

```
#!/bin/bash a=$1; b=$2; echo $((a*b));
```

 - Put your command (to the script) and the result in BB.
Explain briefly what it does.
2. Write a script to get 3 integers from the command-line and prints their product.
 - What happens if you do not pass the 3 required integers when running the script?
3. Write the command to add your name to the prompt string (PS1)
 - Test your command, and put the command to the BB.
 - Hint: to prepend x to variable v, use v=x\$V

LECTURE 7: SHELL SCRIPTING (3)

QUOTING CHARACTERS

- Escape characters: \
 - `echo $date; echo \$date`
- Single quotes: '
 - Strongly preserve literal value
 - `echo '$date'`
- Double quotes: "
 - Weakly preserve literal value, except \$ and \
 - `echo "$date"`
 - `echo "today is $date"`
 - `echo "I'd say \"Go for it\""`

SHELL EXPANSION

- Expand input string to output string
1. arithmetic expansion
 - `echo $((1+2)) #print the sum 3`
 - `echo ${1+2} #print 3`
 2. tilde expansion
 - `ls ~:` list files in home directory
 3. variable expansion
 - `echo $BASH; echo ${BASH}apple`
 - print the value of \$BASH and literal string apple
 - `echo ${FRANKY:=franky}`

4. brace expansion
 - `echo sp{el,il,al}l`
 - `ls Lecture1.{pdf,pptx}`
5. file name expansion
 - `ls ./*.pdf`
 - `ls ./Lecture-T?.pdf`
 - `ls ./Lecture-T[12].pdf`
 - `ls ./Lecture-T[34].pdf`
6. command substitution:
 - replace command substitution with execution result
 - `echo $(date)`

ALIASES

- An alias allows a string to be replaced for another string when used as the first word of a command
- `alias cd='cd /Users/me/space/teaching/cis342'`
- `alias ls='ls -la'`
- `unalias ls`

EXERCISE

1. Write a command to printout the following text:
You said 'Today is Monday.'
 - Upload the command to BB.
2. Edit your `~/.bash_profile` file so that you will be greeted upon login with words
welcome to the terminal.
 - Upload the content of file `.bash_profile` to BB
3. Write *one* command to print the content of all files whose names are `file_a` and whose extensions are `txt` or `csv`.
 - Upload your command to BB.
4. Say your current directory has 9 files
`1.txt, 2.txt, ..., 9.txt`. Write *one* command to print the contents of files `4.txt` and `9.txt`.
 - Upload your command to BB.

FILENAME EXPANSION AND FIND

Find command: find file by filename

1. `find . -name "*.c"`
2. `find / -maxdepth 1 -type d`
 - `-maxdepth`: depth of directories to search
 - `-type d`: find directory, not files

LECTURE 8: GREP & FIND

GREP COMMAND

- Grep command: search file content
1. `grep hello hello.c`
 2. `grep -r hello .`
 - `-r`: search files recursively in all descendant directories
 3. `grep -i HELLO hello.c`
 - `-i`: the pattern is case-insensitive
- Notes
 - `find` searches file name, while `grep` searches file content
 - filename matching is *parameter expansion*, file content matching is *regular expression*.

REGULAR EXPRESSION

- A classic matching problem:
 - takes as input a string and "pattern", outputs a binary decision.
 - `match(al.*ce,alice)=1`
- Format of the pattern: **regular expression** (regex).
- Relevance to Linux shell: `grep`, search in `vim`

REGULAR EXPRESSION (2)

1. asterisk *: matching previous character repeating arbitrary times (including zero time).
 - `match(1133*,113)=1`
 - `demo: echo 113 | grep 1133*`
 - `grep(p,s)` finds *all* substrings in *s* that match pattern *p*
2. dot .: matching arbitrary single character
 - `match(13.,13)=0`
 - `match(13.,134)=1`
3. Brackets [...]: enclose a set of characters
 - `match(1[345],13)=1, match(1[345],15)=1, match(1[345],18)=0`
 - `match(1[3-5],14)=1`
 - `match(1[^3-5],14)=0, match(1[^3-5],18)=1`
3. caret ^: beginning of a line
4. dollar sign \$: end of a line
 - `^$` matches blank lines.

EXERCISE

1. Write a command to find all the files with name starting with `fil` under the current directory.
2. Given a file `file1` with `hello too`, try following commands, and report the result
 - `grep ^hello file1`
 - `grep hello$ file1`
 - `grep t[wo]o file1`
 - `grep ^[A-Z] file1`
3. Write a command to find all lines of all files under current directory recursively that contain a single word "hello".

LECTURE 9: COMMAND EXECUTION & PROCESSES

COMMAND EXECUTION

- Format: Command flag/options parameters
 - Execution result: `echo $?`
 - Repeat the same command: `history, !ls`
- Executing a command usually starts a new **process**
 - Run script in new process: `./script script.sh`
 - Run script in current process: `source script.sh, . script.sh`

OS PROCESS (1)

- Commands to list processes
 - `top`: Show all active OS processes on the machine (global)
 - For performance monitoring
 - `jobs`: Show all *children* processes created by the current process
 - `ps` (default): Show the processes attached (foreground) to a terminal
 - `ps aux`: Show all processes (similar to `top`)

REDIRECTION

OS PROCESS (2)

- File descriptor: binding (channel) between a file and process
 - standard output: `stdout`
 - standard err output: `stderr`
 - standard input: `stdin`
- Files: devices (keyboard, display), disk data files
 - A process can run in background/foreground of a display
- IPC: Inter-process communication
 1. Pipe
 2. Signals

- Redirect descriptor from one file to another
 - Descriptor `stdout` is attached to display (as a file).
 - It can be redirected to a disk file by append `>file_a` to a command
 - Overwrite: `echo 'hello Alice' >file_a`
 - Append: `echo 'hello Alice' >> file_a`
- Explicitly mentioning descriptor numbers
 - 1 is `stdout`
 - 2 is `stderr`
 - Ampersand `&` is both `stderr` and `stdout`
 - `rm nonexistentfile > file_a`
 - `rm nonexistentfile 1>file_a,`
`rm nonexistentfile 2>file_a,`
`rm nonexistentfile &>file_a`

EXERCISE 9.1

1. Try `pwd > zzz`; explain what this command does?
2. Write a command to store the list of files in current directory to a file named by 'f'

IPC: PIPE

- Chaining multiple commands
 - Connect the `stdout` descriptor of a previous command to the `stdin` of the current command. (like a pipe)
 - A pipe is a method of interprocess communication (IPC)
- Demo:
 1. `ls /etc | more`
 2. `ls /etc | vim -`
 3. `pwd | ls`

EXERCISE 9.2

1. Run command `ls /etc | grep conf$ > output`; explain what it does.
2. Design a command to output all files with `cis` in their name, using pipe. Don't use `find`.
 - hint: You can use `ls` and `grep`

LECTURE 10: COMMAND EXECUTION & PROCESSES (2)

KILL A PROCESS

```
ps aux
kill -15 1234 #kill
kill -1 #list exit signal name
```

IPC: SIGNALS & BACKGROUND

- Foreground/background:
 - multiple processes contend for a file
 - a process runs in background/foreground
- 1. demo: run in foreground: `gedit, vim`
- 2. demo: run in background: `gedit &, vim &`
- signals:
 - `<CTRL+Z>`: (keyboard) sends pause signal to the process in foreground
 - `<CTRL+C>`: (keyboard) sends quit signal to the process in foreground
- 3. `<CTRL+C>`, `<CTRL+Z>`, `fg`
- 4. Demo: `jobs, fg 1`: switch between multiple background processes

EXERCISE 10

1. Run `top`. now use `<ctrl+c>` to terminate it. run in another time and this time use `<ctrl+z>`. what is the difference?
2. Run `vim f1` in the background. also run `vim f2` in the background. try switching between them in one terminal.
3. Open firefox and use related commands (e.g., `top`, `jobs`, `ps`) to close it
4. Open firefox again. open some websites and tabs and see how they affect the values in command `top`.
5. use `grep` to find all the processes running as root
6. open firefox web browser and find its pid
7. terminate firefox using the `kill` command. Suppose firefox is crashed and you can't close it using graphical interface. What you need to do to close it?