LECTURE NOTES IN CIS342

YUZHE TANG, AMIN FALLAHI

SPRING, 2017

SECTION 1: BASH

REFERENCES

- "Bash Guide for Beginners" [link]
- "Advanced Bash-Scripting Guide" [link]

TEXT EDITOR

- For Unbuntu/Linux novice, recommend `gedit`. To launch the editor:
  - `gedit filename &`
- For advanced user, recommend `vim`

BASIC COMMANDS

- `man commandname`
- `echo,cat,more`
- environment variables
  - `echo $PATH,echo $PWD`
  - `printenv`
  - `export`

FILE MANAGEMENT

- basic commands
  - `touch`, `mv`, `rm`
  - `mkdir`,ls`
  - `pwd`
- `chmod`: permission bits
  - `u`,`g`,`o`,`a`,`x`,`w`,`r`,`+`,`-`,`=`
  - umask
- `ln`: symbolic/hard link

SHELL PROGRAMMING (FEB. W3)

INTRODUCTION

- Why learn shell programming?
  - automate administrative tasks, save your efforts!
  - e.g. automatic software update, file backup, resource monitoring
- Script: tie shell commands in a file
- Execute script `script.sh`:
  - `./script.sh`
  - `source script.sh`

BASICS: SHELL LANGUAGE

- `#!` sha-bang is a two-byte magic number that designates a file t
  - basically says it's an executable shell script
- Language syntax: if/else, variable
- demo:

1. `#!/bin/bash echo 'hello world';`
2. `#!/bin/bash a=1;b=2;a=$a+$b;echo $a;`
3. `#!/bin/bash a=1;b=2; a=$((a+b));echo $a;`
4. `if [ $a -gt $b ];then echo 'a larger than b`

- exercise:
  1. * try `a=1;b=2;c=$a;a=$b;b=$c;echo $a,$b;`, and put the output to the **blackboard**.
  2. write a script to initialize variables `a`, `b`, `c` 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.

PASSING ARGUMENTS

- demo:
  - `#!/bin/bash echo $1; echo $2; echo $#;`
- exercise:
  1. `#!/bin/bash a=$1; b=$2; echo $((a*b));;` try this script and tell 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 executing the bash script?

COMMENTING

- # is used to comment in bash

GREP & FIND (FEB. W4)

INTRODUCTION

- 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 & GREP

1. asterisk `*`
   - `match(1133*,113)=1`
   - demo: `echo 113 | grep 1133*`
   - `grep(p,s)` finds *all* substrings in `s` that match pattern `p`
2. dot `.`
   - `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.

GREP

- Demos
  1. `grep hello hello.c`
  2. `grep -r hello .`
  3. `grep -i HELLO hello.c`

- Exercise:
  1. 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`
  2. Write a command to find all lines of all files under current directory recursively that contain a single word "hello".

FIND

- Intro
  - `find` searches files, while `grep` searches text
  - note: `find` name matching is *parameter expansion*, not regex.
- demo:
  1. `find . -name "*.c"#`
  2. `find / -maxdepth 1 -type d`
- exercise:
  4. write a command to find all the files with name starting with `fil` under the current directory.

COMMAND EXECUTION AND PROCESSES (MAR. W1)

REDIRECTION

- Intro: redirect from one file to another
  - standard printout `stdout` is a file,
  - `stderr` is another file
  - ampersand `&`: both `stderr` and `stdout`
- demo:
  1. `echo 'hello Alice' > somefile`
  2. `echo 'hello Alice' >> somefile`
  3. `rm nonexistfile > somefile`
  4. `rm nonexistfile 1>somefile`,
     `rm nonexistfile 2>somefile`,
     `rm nonexistfile &>somefile`

- exercise:
  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'

PROCESS MANAGEMENT [OPT]

- `top` (global), `jobs` (children processes)
- `ps`: (default) all processes that have controlling terminals
  - `ps aux`: global

PIPE (IPC1)

- Chaining multiple commands
  - commands run in processes; a process has descriptors or channels to files
  - 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:
  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. Note you can't use `find`.
     - hint: use `ls` and `grep`

SIGNALS & BACKGROUND PROCESSES (IPC2)

- 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:
  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.

EXECUTION MODE [OPT]

- launch a new process: `./script script.sh`, `command`
- run in current process: `source script.sh`,
  `. script.sh`