UNIX: Simple Commands

ITER, Bhubanewar

Text Books

- **Brain W. Kernighan, & Rob Pike**

  The Unix Programming Environment

  PHI

- **Kay A. Robbins, & Steve Robbins**

  Unix^{TM} Systems Programming

  Communications, concurrency, and Treads

  Pearson Education

What is **UNIX**?

[In the narrowest sense] It is a time-sharing operating system **`kernel:`** A program that controls the resources of a computer and allocates them among its users.

- ☞ It lets users run their programs
- ☞ It controls the peripheral devices (discs, terminals, printers, and etc.) connected to the machine.
- ☞ It provides a file system that manages the long-term storage of information such as programs, data, and documents

[In a broader sense] Unix is often taken to include not only the kernel, but also essential programs like compilers, editors, command languages, programs for copying and printing files, and so on.

[Still more broadly] Unix may even include programs developed by you or other users to be run on your system.

Getting started: Terminals and Typing

- **The UNIX system is full duplex:** the characters we typed on the keyboard are sent to the system, which sends them back to the terminal to be printed on the screen.
- **echo:** It is the process, who copies the characters directly on the screen. The echo is turned off, when we are typing a secret password.
- Most of the keyboard characters are ordinary printing characters with no special significance. Few keys are different.
- The RETURN key (i.e. enter key) significes the end of a line of input; system echoes it by moving the terminal's cursor to the beginning of the next line on the screen.
- **Control character**
  - RETURN KEY or `ctrl + m` [holding **control key** and typing **m**]
  - No more input exit → `ctrl + d`
  - backspace → `ctrl + h`
  - tab → `ctrl + i`
  - rings the bell on the terminal → `ctrl + g`
  - break → `ctrl + c`

Unix File Colors

Color	File Type
Blue or dark blue	Directory
Green	Executable or recognized data file
Sky blue	Linked file
Yellow with black background	Device file
Pink	Graphics or image file
Red	Archive file or Zip file

Basic Unix Commands

1. Unix is collection of commands
2. **Command:** A command is a specific instruction given to computer application to perform a partcular task or function
3. A command;
   (a) case-sensitive
   (b) alphabets lower case only
   (c) no unnecessary charactes should not be given
   (d) keywords or reserved words
4. **Few Examples:**
   (a) who
   (b) whoami
   (c) who am i
   (d) date, cal
   (e) ls, ls-l
   (f) ed, cat
   (g) ⋮

File processing commands:

Commands:: **`wc`**, **`grep`**, **`sort`**, **`cmp`**, and **`diff`**. Refer manual for more

- **`$ wc`** Counts the lines, words and characters in one or more files.

  Syntax: **`$ wc filename`**

  Output: **8** **46** **263** **`filename`**

  Description: The file has **8** lines, **46** words, and **263** characters

  word: any string of characters that doesn't contain a blank, tab or newline

- **`wc`** will count more than one file and print the totals

  Syntax: **`$ wc file1 file2 file3`** $\cdots$ **`filen`**

  Output: Run and check the output

grep

- **grep** seaches files for lines that match a pattern.

  Syntax: **$ grep <pattern_name> filename**
  Output: Lines that will match to **<pattern_name>**

- **grep** will also look for lines that donot match the pattern, when **-v** option is used.

  Syntax: **$ grep -v <pattern_name> filename**
  Output: Lines that **will not** match to **<pattern_name>**

- **grep** can be used to search several files.

  Syntax: **$ grep -v <pa_name> fil1 file2** $\cdots$
  Output: It will prefix the filename to each line that matches.

- Look into manual for **grep** options.

sort

- **sort** Sorts its input into alphabetical order line by line.

  Syntax: **$ sort filename**

  Output: sort line by line, but the default sorting order puts blanks first, then uppercase letters, then lower case

- **sort** options;

```
sort -r    Reverse normal order
sort -n    Sort in numeric order
sort -nr   Sort in reverse numeric order
sort -f    Fold upper and lower case together
sort +n    Sort starting at n +1 -st field
```

`tail`, and `head`

- **`tail`** It prints the last 10 lines of a file

  Syntax: **`$ tail filename`**
  Output: print last 10 lines
  Options: **`$ tail -n filename`** `print last n lines`

- **`head`** It prints the first 10 lines of a file

  Syntax: **`$ head filename`**
  Output: print first 10 lines
  Options: **`$ head -n filename`** `print first n lines`

`cmp`, and `diff`

- **`cmp`** finds the first place where two files differ

  Syntax: **`$ cmp file1 file2`**
  Output: file1 file2 differ: char #, line #

- **`diff`** reports on all lines that are changed, added or deleted

  Syntax: **`$ diff file1 file2`**
  Output: exactly which line differs

The **Shell**

- the command interpreter;
- it sits between the user and the kernel;
- it is just a program like **`date, cal`** etc.
- the prompt **`$`** is printed by the shell, it is the main interface to the system
- **some facilities**
  - Filename shorthands
  - Input-Output redirection
  - personalizing the environment

Filename shorthand

Consider a book of several chapters **ch1**, **ch2**, **ch3**, $\cdots$. Each chapter is broken into section like

ch1.1	**ch1.2**	**ch1.3**,	**ch1.4**,	**ch1.5** ...
ch2.1	**ch2.2**	**ch2.3**,	**ch2.4**,	**ch2.5** ...
$\vdots$				
ch10.1	**ch10.2**	**ch10.3**,	**ch10.4**,	**ch10.5** ...

Printing:: `$ pr <filename(s)>` or `$ cat <filename(s)>`

Difficulty:: get bored in typeing file names, and may lead to mistakes

Better:: To use filename shorthand as `$ pr ch*`

Description:: the shell takes the `*` to mean `any string of characters`, so `ch*` is a pattern that matches all filenames in the current directory that begin with `ch`.

Note:: filename shorthand is the service of the shell, not a property of the command like `pr`

Filename shorthand examples try yourself

Try the commands and write the output;

- `$ wc ch1.*`
- `$ wc *.*`
- `$ ls *.*`
- `$ ls *`
- `$ pr *`
- `$ rm *` removes all file(s) in your current directory
- `$ ls *.txt`

Other pattern-matching features provided by the shell

The pattern [$\cdots$] matches any of the character inside the bracket

```
$ pr ch[12346789]*    print files 1-9 except ch5
$ pr ch[1-46-9]*      same as above
wc d[1-5]
$ pr temp[a-z]
```

The pattern ? matches any single character

```
$ ls ?          list files with single character name
$ ls -l ch?.1   list chapters 1-9 not ch10.1 etc
$ wc ?
$ rm temp?
```

Note::

- `$ mv ch.* chapter*` does not work
- Pattern characters like * can be used in pathname as well as in filenames

echo program

It echo its arguments

- `$ echo Hello world`
- `$ echo *`
- `$ echo ch1.*`

Write the difference

- `$ ls Junk`
- `$ ls /`
- `$ ls *`
- `$ ls`
- `$ ls '*'`

`$ echo Junk`

`$ echo /`

`$ echo *`

`$ echo`

`$ ech '*'`

Input-Output redirection

Most of the commands produce output on the terminal. The terminal can be replaced by a file for either or both of input and output.

Example:: **`$ ls`** makes a list of filenames or directory on the terminal

Let write:: **`$ ls > fname`** same list of filenames or directory placed in the file named **`fname`**

Output redirection:: The symbol > and >>

- ☞ **>** put the output in the following file rather on the terminal. The file will be created if doesnot already exist, or the previous contents overwritten, if exist. Nothing is produced on the terminal
- ☞ **>>** same as >, but this will add the contents to the end of the file

Example:

- ✍ **`$ cat f1 >temp`**
- ✍ **`$ cat f1 f2 f3 >temp`**
- ✍ **`$ cat f1 f2 f3 >>temp`** copies the contents of **`f1`**, **`f2`**, **`f3`** onto the end of what already exist in **`temp`**

Input-Output redirection contd...

Input redirection:: The symbol <

- ☞ **<** take the input for a program from the following file, instead of from the terminal

Example:

- ✍ `$ sort <temp`
- ✍ `$ ./a.out <in.txt`

Combining both:: > and <

- ✍ `$ who >temp`
- ✍ `$ sort <temp`

- ✍ `$ ls >temp`
- ✍ `$ wc -l <temp`

`$ sort temp` *VS* `$ sort <temp`

Differentiate:: `$ sort temp` *VS* `$ sort <temp`

Pipes

A pipe is way to connect the output of one program to the input of another program without any temporary file.
A `pipeline` is a connection of two or more programs through pipe

pipe:: The symbol |

☞ | The vertical bar character | tells the shell to set up a pipeline

Example:

✍ `$ who | sort`
✍ `$ who | wc -l`
✍ `$ ls | wc -l`
✍ `$ who | grep mary`

many programs in a pipeline:: | pr | pr |

✍ `$ who | grep mary | wc -l`

Differentiate:: `$ who | sort` *VS* `$ who >sort`

Process

- Running two or more programs with one command line:
  - `$ who ; sort`
  - `$ date ; who`
- The shell recognizes the `;` and breaks the line into two commands
- Both the commands get executed in sequence before the shell return to the prompt character`$`.

How? **More than one program will be running simultaneously:**

The & at the end of a command line says to the shell *start this command running, then take further commands from the terminal immediately*, donot wait for it to complete.

Example:: `$ gedit&` ↵

Output:: [1] 4516

`$`

So, an instance of a running program is called a **process**.

Program vs Process in short

- **wc** is a program; each time we run the program **wc**, that creates a new process.

- If several instances of the same program are running at the same time, each is a separate process with different process with a different process-id.

- If a pipeline is initiated with **$**, as in

```
$ pr ch* | wc &

  5516          process id of $ wc
$
```

  - The processes in it are all started at once. The **&** applies to the whole pipeline.
  - Only one process-id is printed, however, for the last process in the sequence

Command :: `ps`

Report a snapshot of the current processes.

- ✍ `$ ps`
- ✍ `$ ps -ag`
- ✍ `$ ps -p process-id`

Command :: `kill`

Used to stop a process initiatd with **$**

Syntax:: **`$ kill process-id`**

Example:: **`$ kill 4578`**

More Command :: `nohup`, `nice`, and `at`

Command:: `nohup`

Syntax:: `$ nohup command &`

Description:: Run COMMAND, ignoring hangup signals (i.e. turn off the terminal or break connection). The command will continue to run if you log out. Any out put from the command is saved in a file called **nohup.out**.

Command:: `nice`

Syntax:: `$ nice command &`

Description:: Run a program with modified scheduling priority. `nuhup` automatically calls `nice`

Command:: `at`

Syntax:: `$ at <time>`

Description:: Tell the system to start the processes at some time. Time can be written in 24-hour style like 2130 or 12-hour style like 930pm.

Example:: `$ at 3am <file`

Tailoring the environment

✍ **$ tty** ↵

Description:: print the file name of the terminal connected to standard input

Output:: **/dev/pts/10**

✍ **$ stty**

Description:: set terminal options. i.e. Print or change terminal characteristics.

Example:: Setting **erase** character to # instead of *backspace*

✍ **$ stty erase '#'** ↵

Example:: Setting **line kill** character to @

✍ **$ stty kill '@'** ↵

☞ **stty** to be typed every time we log in. Other options are there to set permanently.

Practice:: Unix Commands

0. Use **`pwd`** command to know where you are.
1. Create a directory named as **`Registration number`**
2. Change to the directory, you have created.
3. Over the directory **`Registration number`**, create one more directory **`Branch`**
4. Store few files such as graphics, ZIP, C-program and a.out, audio, onto the directory named **`Registration number`** through browsing from the window to the directory.
5. Visit to the directory **`Registration number`** from the command prompt.
6. Use **`ls`** command to view the related information stored on the directory **`Registration number`**. carefully watch the **colors** related to the files, directory is used under UNIX.

Assignment-1

1. Create a directory named **`04012018`**[command: mkdir], change the directory to **`04012018`**[cd]
2. Create two files named **Junk** and **Temp** using the line editor **`ed`**.
   - The content of Junk as

     `The Unix system if full duplex: The character you type on the keyboard are sent to the system, which sends them back to the terminal to be printed on the screen.`
   - The content of temp as

     `Normally, this` *`echo`*
     `process copies the characters directly to the screen, so you can see what you are typing, but some times, such as when you are typing a secret password, the echo is turned of so the characters do not appear on the screen`
3. Create two different file of your own that will contain at least 5 line each.

Assignment-1 contd...

4. use **`ls`** command to lists the names of files.
5. **`ls`** has options; try **`ls -t`**, **`ls -l`**

   `ls -t`: causes the files to be listed in time order; the order in which they are last changed, most recent first

   The **`ls -l`**: gives long listing that provides more information about each file.
6. use the command **`ls -l <anyfilename>`**, write the description of the output.
7. **options can be grouped:** Try the command: **`ls -lt`**, Note the output
8. To print index number of each file **`ls -i`**

   For more options visit man page
9. Display the content of the file Junk and temp using **`cat`** command
10. Use **`cat <file-1 file-2 file-3 ...>`**. Observer the output
11. use **`cat -n filename`**. Output?

More on ed

1. Invoke ed editor: $ **`ed`**
2. Dispaly line 1, 2 of Junk file
3. Append two new line after line no 4 of Jubk [n a]
4. Insert a text ITER CSE Sec-b before line 5 [n i]
5. change lines 2 throgh 5 in Junk [m,n c]