Lecture 6

Using the bash Shell and Strings

Objectives

Upon completion of this unit, you should be able to:

- Use command-line shortcuts
- Use command-line expansion
- Use History and editing tricks
- Use the **gnome-terminal**

Command Line Shortcuts File Globing:

- **Globing is wildcard expansion:**
  - *** - matches zero or more character**
  - **? – matches any single character**
  - **[0-9] –matches a range of numbers**
  - **[abc] – matches any of the character in the list**
  - **[^abc]- matched all except the character in the list**
  - **Predefined character classes can be used**

Typing the following command:

[student@stationx ~] $ rm *mp3

Is the same as typing:

[student@stationx ~] $ **rm gonk.mp3 zonk.mp3**

The result is that all files in the directory that have names ending in mp3 (in the case, just the two listed) will be removed.

Echo can be used to test the expansion of metacharacters before using them in a destructive command like **rm:**

[student@stationx ~] $ echo ?o*
Joshua.txt gonk.mp3 zonkmp3

Uses of character ranges, as those used for numbers are considered harmful. They are not portable and may result in unexpected list after expansion.

You would rather use character classes. Which are also used in other applications, and only in bash.

The sysntex for a character class is. [: keyword:], where keyword can be: alpha, upper, lower, digit.alnum, punct, space

If you want to match only numbers, you might use:

[[:digit:]]

If you want to match anything but alphanumeric characters:

[A[:alnum:]]

Command Line Shortcuts the *Tab* key:

- Type *Tab* to complete command lines:
  - For the command name, it will complete a command name
  - For an argument, it will complete a file name
- Examples:

$ xte<tab>
$ xterm
$ ls myf<tab>
$ ls myfile.txt

Command Line Shortcuts History:

- **bash** stores a history of commands you've entered, which can be used to repeat commands
- Use **history** command to see list of "remembered" commands

$ history
14 cd /tmp
15 ls –l
16 cd
17 cp /etc/passwd
18 vi passwd
…output truncated

More History Tricks:

- Use the up and down keys to scroll through previous commands
- Type ctrl-r to search for a command in command history
  - (reverse-i-search)
- To recall last argument from previous command:
  - Esc:- (the escape key followed by a period
  - Alt:- (hold down the alt key while pressing the period)

History:

Using your history is great productivity-enhancing tool. Linux users who develop a habit of using their history can streamline and speed their use of the shell. Try playing with keystrokes listed above.

You can ignore repeated duplicate commands and speed their use of the shell. Try playing with the keystrokes listed above.

You can ignore repeated duplicate commands and repeated lines that only differ in pepended spaces by adding the following to your .bashc.

[stuent@stationX ~] **export histcontrol=ignoreboth**

Command Line Expansion the Tilde:

- Tilde (~)
- May refer to your home directory

$ cat ~/.bash_profile

- May refer to another user's home directory

$ ls ~julie/public_html

Command Line Expansion Commands and braced sets:

- Command Expansion:$ ( ) or ``
    - Prints Output of one command as an argument to another

$ echo "this system's name is $ (hostname)"
This system's name is server1.example.com

- Brace Expansion: { }

$ echo file {1,3,5}
File1 file3 file5
$ rm –f file {1,3,5}

The bash shell has some special features relating to expansion which significantly improve the power of working at the command line.

Curly braces are useful for generating patterned strings. For Example without the curly braces, the **mkdir** command below would take almost two hundred keystrokes to execute.
[student@stationX ~] $ **mkdir –p work/ {inbox, outbox, pending} / {normal,urgent.important}**

Use of the backquotas is called command substitution. In command substitution. The command in backquotas is executed and the output of the command is placed on the command line as if the user had typed it in. an alternative sysntex for the backquotas is to place the command in parentheses preceded by a dollar sign $ ( ).

Command Editing Tricks:

- **Ctrl-s moves to beginning of liine**
- **Ctrl-e moves to end of the line**
- **Ctrl –u deletes to beginning of line**
- **Ctrl-k delete to end of line**
- **Ctrl-*arrow* moves left or right by word**

Gnome-Terminal:

- Applications◊ accessories◊ terminal
- Graphical terminal emulator that support multiple “tabbed” shells
    - Ctrl-sift-t creates a new tab
    - Ctrl-pgUp/pgDn switches to next/prev tab
    - Ctrl-sift-c copies selected text
    - Ctrl-sift-v pastes text to the prompt

Scripting Basics:

- Shell script are text files that contain a series of commands or statements to be executed
- Shell scripts are useful for:
    - Automating commonly used commands
    - Performing system administration and troubleshooting
    - Creating simple applications
    - Manipulation of text or files

Creating Shell scripts:

- Step 1: use such as vi to create a text file containing commands
    - First line contain the magic shebang sequence:# !
    - #!/bin/bash
- Comments your scripts !
    - Comments start with a #

Shell script are text file that generally contain one command per line, but you can have multiple commands on a line if you separate them with semicolons (;) in order to continue command on to the next line you use the line continuation character. For the Bourne shell (/bin/sh) and its derived shells such as bash, this is a backslash followed by a new line. You can enter this by pressing the \ key followed by the enter key on most keyboards. This will enable you to enter one command that spans multiple lines.

The first line a shell scripts should contain ‘magic’ which is commonly referred to as the shebang. This tells the operating system which interpreter to use in order to execute the script. Some examples are.

Shebang use

- **#!/bin/bash**-used for bash scripts (most common on Linux)
- **#!/bin/sh**-used for Bourne shell scripts (common on all UNIX-like systems)
- **#!/bin/csh**-used for C shell scripts (common on BSD derived systems)
- **#!/user/bin/per1**-used for per1 (an advanced scripting and programming language)
- **#!/user/bin/python**-used for python scripts (an object oriented programming language)

Commenting shell scripts

It is extremely important to put comments in your shell scripts. Anything following the # symbol is comment and therefore ignored by the interpreter. It is good practice to make a shell script self documenting. Self documenting. Means that someone with almost no scripting knowledge can read your comments in the script and reasonably understand what the script does. The easiest way to do this is to write the comments. Before you actually write the code. This has an additional benefit of clarifying to yourself, what your objective is. In addition, this practice makes the script easier to mainten for both yourself and others. As time progresses, you may not recall what you were trying to do at the time and comments may help clarify the original objective.

Creating Shell Scripts:

- Step 2: Make the scripts Executable

$ chmode u+x myscript. Sh

- To execute the new script:
  - Place the script file in a directory in the executable path –OR-
  - Specify the absolute or relative path to the script on the command line

An example of making a script you own executable:

[student@stationX ~] $ Chmod u+x scriptfile

Or
Chmod 750 scriptfile

Ensure that the script is located in a directory listed by the PATH environment variable. To do this. Enter the following command:

[student@stationX ~] **$ echo #PATH**

If the script is not in a directory listed in the PATH variable. Either move the script to a directory that is (such as SHOME/bin) or specify the absolute or relative path on the command line when executing the script:
[student@stationX ~] $ /home/user/mytestscript

Or
[student@stationX ~] **$. /mytestscript**

You can create a bin directory in your directory and store your own script here. Since this directory is normally added in your PATH environment variable. You can run any of your scripts without having to type in the full path.

Sample Shell Script:

#! /bin/bash
This script display some information about your environment
Echo “Greetings. The sate and time are $(date)”.
Echo “your working directory is: $ (pwd)”