CS242: Systems Software Lab

Tutorial 3

TA: Abhishek

Course Instructor: Dr. Ashish Anand

Indian Institute of Technology Guwahati

August 2019

Shell

The shell is simply a program that allows the system to understand your commands. (That's why the shell is often called a command interpreter.)

There are three uses for the shell:

- Interactive use
- Customization of your Unix session
- Programming

Different shells: **bash**, zsh, sh, fish, tcsh, csh

The ~/.bashrc file

Reference: Chapter 3 and Chapter 4 (only bash shell) of Unix in a nutshell book.

Environment Variables

An **environment variable** is a dynamic-named value that can affect the way running processes will behave on a computer.

They are part of the environment in which a process runs.

For example, a running process can query the value of the TEMP environment variable to discover a suitable location to store temporary files, or the HOME or USERPROFILE variable to find the directory structure owned by the user running the process.

Popular variables: $HOME, $PATH, $DISPLAY, $LANG, $TERM, $SHELL, $EDITOR

How to set variables: Eg. set the home variable

Bash: Shell Expansions

- **Brace expansion**
- **Tilde expansion**
- Parameter and variable expansion
- Command substitution
- Arithmetic expansion
- Word splitting
- **Filename expansion**

Reference: https://www.gnu.org/software/bash/manual/html_node/Shell-Expansions.html#Shell-Expansions

Brace Expansion

- Brace expansion is a mechanism by which arbitrary strings may be generated.
- Brace expansion is performed before any other expansions, and any characters special to other expansions are preserved in the result.
- It is strictly textual. Bash does not apply any syntactic interpretation to the context of the expansion or the text between the braces.
- Example:
    - echo a{b,d,e}c
    - mkdir -p /tmp/a{a,b,c}/a{a,b,c}
    - echo {a..z}

Tilde Expansion

- Tilde: ~
- Examples:
    - ~
    - ~/foo
    - ~user/foo
    - ~+/foo

Filename Expansion

- Bash scans each word for the characters ‘*’, ‘?’, and ‘[’. If one of these characters appears, then the word is regarded as a pattern, and replaced with an alphabetically sorted list of filenames matching the pattern.
- Special pattern characters:
  - *
  - ?
  - [

- Examples: echo *, echo a?, echo [a-c]c, ls *.txt

Grep

- grep (***g**lobally search a **r**egular **e**xpression and **p**rint*) is a command-line utility for searching plain-text data sets for lines that match a regular expression.
- grep [OPTION...] PATTERNS [FILE...]
- Examples:
  - grep root /etc/passwd
  - grep -n root /etc/passwd
  - grep -nv root /etc/passwd
  - grep -i root /etc/passwd
  - ps -aux | grep firefox
  - cat file_path | grep pattern

Grep regular expressions

- ^ : grep "^GNU" GPL
- $: grep "and$" GPL
- . : grep "..cept" GPL
- [ ] Bracket Expressions: grep "[^c]ode" GPL-3, grep "^[A-Z]" GPL-3
- Character classes: grep "^[[:upper:]]" GPL-3, :digit:
- * Repeat Pattern Zero or More Times: grep "([A-Za-z]*)" GPL-3
- Escaping Meta-Characters: grep "^[A-Z].*\.$" GPL-3
- Alternation: grep -E "(GPL|General Public License)" GPL-3

Reference: https://www.digitalocean.com/community/tutorials/using-grep-regular-expressions-to-search-for-text-patterns-in-linux

Find - search for files in a directory hierarchy

Some of the key Examples:

- Find file by names: find . -name “*.png”
- Find all directories: find . -type d
- Find files by size: find ~/Downloads/ -size +4500M
- Find files by file permission: find . -perm 777
- Find files that did not match a pattern: find . -not -iname "*.jpg"

Reference: https://danielmiessler.com/study/find/

Sed: Stream Editor

- **It parses and transforms text**
- Some important features
  - Substitution:
    - echo day | sed s/day/night/
    - echo Sunday | sed 's/day/night/'
    - Using & as matched string: sed "s/[a-z1-9]*/(&)/" test_file.txt

Reference: https://www.grymoire.com/Unix/Sed.html

Awk

- **AWK** is a programming language designed for text processing and typically used as a data extraction and reporting tool.