Unix CLI

San Skulrattanakulchai

2/8/2016

File System Structure

- Virtually all modern operating systems have a **hierarchical file structure**.
- I am assuming you are using some kind of UNIX system, e.g., linux, OS X.
- A **directory** (or **folder**) contains other files and/or directories.
- The **root directory** is the starting point of the access path of all files and directories in the system. It is denoted by the forward slash character /
- When interacting with the user, the OS assumes you are currently at some specific point in the file system. This point is called your **current directory**, denoted by the . character.

File System Structure

- Double dots .. denotes your parent directory. It is the directory containing your current directory, unless you are at the root of the file system, in which case the parent directory is the root itself.
- You can refer to a file or directory by writing its access path either starting from root or from your currect directory. The forward slash character / separates the directories on the path to your target file or directory.
- Every user has a home directory. The ~ character denotes your home directory. It is where you are put when you start a terminal session. In general ~`user` denotes the home directory of user `user`.

Command Line Interface (CLI)

- In a CLI, the user interacts with the computer system by typing in **commands** on the keyboard, and the sytem responds by printing out text messages on the user's monitor (terminal).
- Compared to the Graphical User Interface (GUI), a CLI is
  - harder to learn & master
  - not as pretty
  - more powerful, in fact, some functionality may not have a GUI equivalent

Command Interpreter

- A **shell** is a command interpreter. It interprets the commands you type in from the keyboard, executes the commands if possible, prints out the results of execution, or prints out error messages if it can't execute your commands or if it encounters any problem.
- Examples of shells are
  - **sh**
  - **csh**
  - **bash**
  - **tcsh**
  - **zsh**
- I will assume you'll be using the Bourne-Again Shell (bash) since it is the default shell on the OS X machines in our lab.

bash

- bash is a programming language. It has conditionals, loops, and functions.
- A command is either **internal** or **external**.
- All command entered interactively through the keyboard must end in a newline character by the user's pressing the `Enter` key.
- The shell understands an internal command and is able to execute it immediately.
- If the shell does not understand a command, it assumes that it is an external command, and will search all directories as specified in the `PATH` variable for a file having that command as name. If the file is found and is executable, the shell loads it into memory and executes it.
- The format of a simple command is
  command [flags] [arguments]
  where white spaces (blanks or tabs) separate the components.

Useful Commands

- some common commands:
  - ls
  - cd
  - mkdir
  - rmdir
  - rm
  - mv
  - cp
  - cmp
  - diff
  - pwd
  - echo
  - cat
  - less
  - man
  - info
  - help

Text Files & Text Editors

- A *text file* is meant to be read by humans. It contains characters that can be safely printed to the screen and printers. It doesn't contain formatting characters or control characters or other binary data.
- A *text editor* is an app that allows the user to conveniently read, write, modify, and save text files. It actually works on **buffers** in RAM until such time when the user tells it to write those buffers onto secondary storage.
- Examples of simple text editors are **nano** and **TextEdit** (in plain text mode).
- Some editors are meant to be used for writing program source code. They have features useful to programmers. Examples are **TextWrangler** and **Sublime Text**.
- Some advanced general-purpose text editors have features that are useful for editing all kinds of text files. Examples are **emacs** and **vim**.

I/O Redirection

- A **process** is a program in execution.
- Every process are given 3 standard *text streams* when it starts:
  - *standard input*
  - *standard output*
  - *standard error output*
- A *text file* is a kind of text stream so for interactive programs, it is reasonable to redirect any of its standard I/O stream. E.g.

I/O Redirection

- The command

  prog $<$ *infile*

  runs `prog` with its stardard input coming from `infile`,
- The command

  prog $>$ *outfile*

  runs `prog` with its stardard output redirected to `outfile`,
- The command

  prog1 | *prog2*

  runs `prog1` with its stardard output being the stardard input of `prog2`.