Programmer's Survival Guide for Mac & Ubuntu

Terminal, File System, Users and Editors

TABLE OF CONTENTS (HIDE)

1. Command-line "Terminal"

Programmers uses command-line to issue commands to the operating system, instead of graphical user interface (GUI). This is because command-line is much more flexible than graphical interface - you can provide additional options to a command; you can pipe the output of one command into another command; you can put a set of commands in a script to automate a task.

A *shell* (or *command interpreter*) is a program that lets you issue commands to interact with the operating system, via a *text-baed* interface . It provides a set of commands and utilities. It also has its own programming language for writing shell scripts. There are many shell programs available, from the legacy and obsoleted `sh` (Bourne Shell), `csh` (C Shell), `ksh` (Korn Shell), to newer `bash` (Bourne Again Shell), `tcsh` (Tenex C Shell).

Terminal is an application that runs a shell program. By default, the Terminal in Ubuntu and Mac OS X runs the `bash` shell, although you are free to switch into another shell program, such as `tcsh`.

To launch a Terminal:

- In Mac OS X: Open "Finder" ⇒ Go ⇒ Utilities ⇒ Select "Terminal". Drag the "Terminal" to your dock since you need to use it frequently.
- In Ubuntu: Open "Dash" ⇒ type "Terminal"; or choose "Applications" lens ⇒ Installed ⇒ Select "Terminal". Drag the "Terminal" to your Launcher since you need to use it frequently.

A Terminal displays a *command prompt* ending with "`$`" sign, in the form of:

- In Mac OS X: "*`ComputerName:CurrentDirectory Username`*`$`"
- In Linux: "*`Username@ComputerName:CurrentDirectory`*`$`"

You can enter commands after the command prompt. For example, enter "`pwd`" to print the current working directory:

```
$ pwd
.......
```

In this article, I shall denote the command prompt simply as "`$`".

2. Unix File System

Root Directory

Everything in Unix is a *file* - data files, program files, even input and output device files. Files are organized in *directories* (aka *folders*). The directories are organized in a hierarchical *tree* structure, starting from the *root* directory. A directory may contain sub-directories and files. A sub-directory may contain sub-sub-directories and files. A file is identified via the directories and filename, e.g., "`/usr/lib/jvm/jdk1.7.0_07/bin/javac`". The *leading* "`/`" (forward slash) denotes the *root* directory. The sub-directories are also separated by a "`/`".

There is only one root directory for the entire Unix's file system.

Notes: Windows use "`\`" (back slash) as the directory separator, and may contain multiple root directories - one for each drive (e.g., `c:\`, `d:\`).

Home Directory

Unix is a *multi-user* operating system (although most of you, in particular the Mac users, use it as a single-user *personal* computer). Each user on the system is allocated a directory for storing his files, known as *home* directory. The users' home directories are allocated under `/home` for Ubuntu, or `/Users` for Mac OS X, with a sub-directory name the same as the username (or login ID). The home directory of the current login user is denoted as "`~`". It contains sub-directories such as `~/Desktop`, `~/Downloads`, `~/Documents`, `~/Movies`, `~/Music`, and etc.

Pathname and Filename

To reference a file, you need to provide the *pathname* (directory and sub-directories names) and the *filename*. For example, in "`/usr/lib/jvm/jdk1.7.0_07/bin/javac`", the pathname is "`/usr/lib/jvm/jdk1.7.0_07/bin/`" and the filename is "`javac`".

The pathname can be specified in two ways:

1. Absolute Pathname: An absolute path begins from the root directory (starts with "`/`"), and contains all the sub-directories (separated with "`/`") leading to the file, e.g., "`/usr/lib/jvm/jdk1.7.0_07/bin/`". An absolute path can also begin with the current user's home directory (starts with "`~`"), e.g., "`~/Downloads/jdk`". Suppose the current user is "peter", it maps to "`/home/peter/Downloads/jdk`" (in Linux) or "`/Users/peter/Downloads/jdk`" (in Mac).
2. Relative Pathname: A relative path is relative to the so-called *current working directory*. A relative path does not begin with "`/`" or "`~`". For example, if the current working directory is "`/usr/lib/jvm`", then the relative pathname "`jdk1.7.0_07/bin`" refer to "`/usr/lib/jvm/jdk1.7.0_07/bin`".

Unix system is case sensitive, a *rose* is NOT a *Rose*, and is NOT a *ROSE*.

3. Basic Commands

3.1 `pwd` (Print Current Working Directory)

The Terminal session maintains a so-called *current working directory*. All relative pathnames/filenames are relative to the current working directory. To display the current directory, issue command "`pwd`" (print working directory):

```
// Print Current Working Directory
$ pwd
......
```

When a Terminal is launched, it sets the initial working directory to the *home* directory of the current login user (denoted as "`~`").

The current working directory is often included as part of the command prompt.

3.2 `cd` (Change Working Directory)

To change current working directory, issue command "`cd` *new-pathname*" (change directory). You can specify new pathname in two ways: *absolute* or *relative*. An absolute path begins with a "`/`" (root directory) or "`~`" (home directory). A relative path is relative to the current working directory and does NOT begin with "`/`" or "`~`". For example,

```
$ cd /                // Change directory (absolute) to the root
$ cd /usr/local       // Change directory (absolute) to "/usr/local"
$ cd mysql            // Change directory (relative) to mysql of the current directory
$ cd myproject/bin    // Change directory (relative) to myproject/bin of the current directory
```

You can `cd` in multiple stages (e.g., one `cd` for each sub-directory - recommended), or `cd` in a single stage with the full pathname.

```
$ cd /          // "/"
$ cd usr        // "/usr"
$ cd local      // "/usr/local"
$ cd mysql      // "/usr/local/mysql"
$ cd bin        // "/usr/local/mysql/bin"

// Same As
$ cd /usr/local/mysql/bin
```

You can use "`/`" to denote the root; "`~`" to refer to the home directory of the current login user; "`..`" (double-dot) to refer to the parent directory; "`.`" (single-dot) to refer to the current directory; and "`-`" (dash) to refer to the previous directory. For example,

```
$ cd ~            // Change directory to the home directory of the current user
$ cd              // same as above, default for "cd" is home directory
$ cd ~/Documents  // Change directory to the sub-directory "Documents" of the home directory of the current user
$ cd ..           // Change directory to the parent directory of the current working directory
$ cd -            // Change directory to the previous working directory (OLDPWD)
```

Setting proper working directory can greatly simplify your work. For example, to compile a Java program called "`Hello.java`" in "`~/myproject/java`":

1. Set the working directory to "`~/myproject/java`", and reference the file with filename only (without the path):

```
$ cd ~/myproject/java  // Set the working directory
$ javac Hello.java     // Filename only, in current directory
```

2. You can also refer to a file with its full pathname in any working directory:

```
// Any working directory
$ javac ~/myproject/java/Hello.java   // Using fully-qualified filename
```

Although setting working directory is NOT really a necessity, it can be very messy at times without the correct working directory (e.g., the output file "`Hello.class`" is created in the current directory). You should always set a proper working directory before issuing any command with file references.

3.3 `ls` (List Directory's Contents)

You can use command `ls` to list the contents of the current working directory, e.g.,

```
// List contents of current working directory in short format
$ ls
Desktop    Downloads         Music     Public     Videos
Documents  examples.desktop  Pictures  Templates

// List in long format
$ ls -l
total xx
drwxr-xr-x 2 myuser myuser 1024 Mar 22 21:32 Desktop
drwxr-xr-x 2 myuser myuser 1024 Mar 22 21:32 Documents
drwxr-xr-x 2 myuser myuser 1024 Mar 22 21:32 Downloads
-rw-r--r-- 1 myuser myuser 8445 Mar 22 17:30 examples.desktop
......

// Append file indicator (e.g., trailing / indicates a directory)
$ ls -F
Desktop/    Downloads/        Music/     Public/     Videos/
Documents/  examples.desktop  Pictures/  Templates/
```

Wildcard `*`

You can list selected files using wildcard `*`, which matches 0 or more (any) characters. For example,

```
$ ls *.java     // List files ending with ".java" in short format (default)
$ ls -l *.java  // List files ending with ".java" in long format
$ ls -ld my*     // List files and directories beginning with "my" in long format
```

You could, of course, view the contents of a directory using a File Manager (such as "Finder" in Mac, or "Home Folder" in Ubuntu) more conveniently.

3.4 `cat`, `less` (Viewing File Contents)

You can use commands `cat`, `less` or `more` to display contents of a text file on console. `cat` shows the entire file, which can be inconvenient for long file. `less` shows one page of the file. For example,

```
$ cat /proc/cpuinfo
  // Display the entire file
  // Use window's scroll-bar to scroll

$ less /proc/cpuinfo
  // Display one page of the file
  // Use Up|Down|PgUp|PgDown key to scroll, and press "q" to quit
```

3.5 Shortcut Keys - IMPORTANT

Previous Commands: You can use the up/down arrow keys to retrieve the previous/next command in the command history.

Copy/Paste:

- In Mac OS X: use `cmd+c` and `cmd+v`.
- In Ubuntu: use `shift+ctrl+c` and `shift+ctrl+v`. You can also use middle mouse-button to copy/paste the highlighted text on the command-line. (The `ctrl+c` is used as interrupt signal; ctrl+d for end of text input; ctrl-z for suspending current process.)

3.6 Other Frequently-used Commands

```
$ clear                 // Clear Screen
$ who                   // List all login users
$ whoami                // Show the current login user
$ which program-name    // Show the location of the program, e.g., "which javac"
```

3.7 Getting Helps

```
$ help                 // Show all the available commands
$ help command-name    // Show help for a specific command, e.g., "help cd"
$ man command-name     // Show the manual page for a specific command, e.g., "man cd"
```

4. If you need to Install Software or Perform System tasks...

4.1 Regular Users vs. Superuser

Unix is a multi-user operating system. Each user is identified via an username (or login ID). It is allocated a home directory, typically called `/home/username` (for Ubuntu) or `/Users/username` (for Mac OS X). Each user is also assigned to a group.

User and group are used for access control. A "regular" user can only modify resources specific to his account, and cannot modify other user's resources or perform system-wide changes.

On the other hand, superuser (or privileged user) has unrestricted access to the entire system. Superuser can carry out any task, even destructive ones. Using superuser to perform daily operations can be dangerous. A special superuser, called `root`, is created during the installation. However, in many system (such as Ubuntu and Mac OS X), the `root` account is disabled by default for safety and security.

You should perform daily works with a regular user. However, at times, you need the superuser privilege to perform a task, such as install a new software or launch a system program.

An *authorized* user can run a command with superuser privilege by prefixing it with `sudo` (superuser do). The system will prompt you for password. Reply with the login (authorized) user's password. The entered password will not be shown (not even showing asterisks) for maximum security. The password entered is stored for 15 minutes (by default) for all subsequent `sudo` commands. After that, you will need to enter the password again. For example,

```
$ cat /etc/shadow
cat: /etc/shadow: Permission denied  // regular user has no permission to view this file
$ sudo cat /etc/shadow
[sudo] password for myuser:          // entered password will not be shown
......
```

To enable a user to `sudo`:

- In Ubuntu: System Settings ⇒ User Accounts ⇒ Select the user ⇒ Unlock ⇒ Set the account type to "Administrator" (instead of "Standard User"), which adds the user to "`sudo`" group.
- In Mac OS X: System Preferences ⇒ Users and Groups ⇒ Select the user ⇒ Unlock ⇒ Check "Allow users to administrate this computer".

4.2 Processes

Unix is a multi-process, multi-user operating system. It can run many processes concurrently.

You can use GUI applications to view all running processes and terminate a particular process (similar to "Task Manager" in Windows).

- In Mac OS X: launch "**Activity Monitor**" (Under /Applications/Utilities) and select "All Processes".
- In Ubuntu: launch "**System Monitor**" (search from Dash) and select "Processes".

On command-line, you can use command `ps` to list all the processes and `kill` to terminate a particular process:

```
$ ps            // Print processes of current user
$ ps -e         // Print all processes
$ ps -ef        // Print all processes in full-listing
$ ps -ef | grep "mysql"  // Print processes containing mysql
$ ps aux | grep "mysql"  // same as above
$ top           // Display resource usage and top processes

$ kill pid      // Kill a particular possess with the given processID
$ kill -9 pid   // Force kill
```

5. Programming Text Editors

A *program editor* (or *source code editor*) is *programming language sensitive* and *context-aware*. It highlights the syntax elements of your programs; and provides many features that aid in your program development (such as auto-complete, compile/build/run, help menu, etc.). On the other hand, a *plain text editor* is not language-sensitive and, therefore, is NOT suitable for writing programs. For full-scale software development, you should use an appropriate IDE (Integrated Developement Environment).

It is important to use a *mono-space font* (such as "Courier", "Consola") for programming, so that the columns are properly aligned.

5.1 Graphical Programming Text Editors - `gedit` and others

A good graphical programming text editor, which provides syntax highlighting, is crucial in programming and development.

- In Ubuntu, the default "gedit" is an excellent programming text editor.
- In Mac OS X, you can use the default "TextEdit" to create plain text file by choosing the option "Make Plain Text" (under "Format"). However, TextEdit is NOT a programming text editor and does not provide syntax highlighting. I suggest that you install gedit, jedit, or other programming text editors.

Other popular graphical editors include kate (KDE text editor) and kedit (another KDE editor).

5.2 Display Text files in Command-line - `cat`, `less`, `head` and `tail`

If possible, use a graphical text editor to display a text file. Nonetheless, you can use the following commands to display a text file in command-line:

- **`cat` *`filename`***: Concantenate file or print the file to console.
- **`less` *`filename`***: Display the file in pages. You can scroll the display with up, down, pageup, pagedown keys. `less` replaces the legacy `more`.
- **`head|tail` *`filename`***: Display the first|last part of the file.

For example, let's display the /proc/cpuinfo file with these commands:

```
$ cat /proc/cpuinfo
   // Show entire file
   // Use the window's scroll-bar to scroll

$ less /proc/cpuinfo
   // Show one page
   // Use the Up|Down|PgUp|PgDown key to scroll

$ head /proc/cpuinfo
   // Show first page of the file

$ tail /proc/cpuinfo
   // Show last page of the file
```

More on `cat`

The command `cat` has many usages. You can use `cat` to concatenate (join) a few files. For example,

```
$ cat file1 file2 file3
   // display file1, file2 and file3 on the console

$ cat file1 file2 file3 > file4
   // Concatenate file1, file2 and file3 as file4
```

You can also use `cat` command to create a small text file. `cat` without argument uses standard input as its input. You could use ctrl+D to signal EOF. For example:

```
$ cat > out.txt
The quick brown fox jumps over the lazy dog[ctrl+D]
$ cat out.txt
The quick brown fox jumps over the lazy dog
```

5.3 Console-based Text editors - `nano`, `vim`, `emacs`

You should use graphical text editor if possible. Use console-based text editor as the last resort.

Many console-based text editors are available, such as: `ed`, `jed`, `joe`, `mcedit`, `nano`, `red`, `sed`, `vi`, `vim`, `emacs`, and etc.

nano

`nano` is small, simple and easy-to-use text editor, that is suitable for creating/editing small files. `nano` is a GNU text editor available for Mac and Unix Systems. `nano` replaces its predecessor called `pico`.

To start `nano`, open a Terminal and issue:

```
$ nano                // Create a new file
$ nano filename       // Edit an existing file
```

To exist `nano`, press ctrl+X, followd by "y" (yes) to save the file.

You can run nano with `sudo` for editing restricted system files, as follows:

```
$ sudo nano filename  // Run nano with superuser to edit an existing system file
```

vim

An improved version of the classical vi (visual) text editor. Launch via "`vim filename`".

`vim` operates in two mode: insert mode (for text editing) and command mode (for issuing commands such as quit and save). Press "`i`" to enter insert mode and start editing. Press ESC to switch to command mode.

The frequently-used commands are:

- "`:q`" to quit; "`:q!`" to quit without saving.
- "`:w`" to write (save); "`:w filename`" to save to the filename.

You could run `vimtutor` to learn about `vim`.

emacs

The comprehensive GNU emacs editing environment.

REFERENCES & RESOURCES

[TODO]

Latest version tested: Mac OS X 10.7.5 (Lion), Ubuntu 12.10
Last modified: April, 2013