Laboratory Handout # 1: Review of Linux Basic Commands

Prepared by Marie Yvette de Robles and Joseph Anthony Hermocilla

Objectives

At the end of this meeting, students should be able to:

- understand the general concept of an operating system;
- understand the Linux filesystem and directory structure;
- learn to use relevant Linux commands.

What is an Operating System?

An *operating system (OS)* is a resource manager. It takes the form of a set of software routines that allow users and application programs to access system resources in a **safe, efficient,** and **abstract** way.

Linux is a free **open source** UNIX OS for PCs that was originally developed in 1991 by Linus Torvalds, a Finnish undergraduate student.

As Linux has become more popular, several different development streams or distributions have emerged, e.g. Redhat, Slackware, Fedora, Mandrake, Debian, Ubuntu, and Caldera.

Architecture of the Linux Operating System

Linux has all of the components of a typical OS:

- **Kernel**
- **Shells and GUIs**
- **System Utilities**
- **Application programs**

General format of Linux commands

A Linux command line consists of the name of a Linux command (actually the "command" is the name of a *built-in shell command*, *a system utility* or *an application program*) followed by its "arguments" (options and the target filenames and/or expressions). The general syntax for a Linux command is

```
$ command -options targets
```

The dollar sign is called the command prompt. Here command can be thought of as a verb, options as an adverb and targets as the direct objects of the verb. In the case that the user wishes to specify several options, these need not always be listed separately (the options can sometimes be listed altogether after a single dash).

The Linux Filesystem

Every item stored in a Linux filesystem belongs to one of four types:

1. Ordinary files
2. Directories
3. Devices
4. Links

Typical Linux Directory Structure

The path to a location can be defined by an **absolute path** from the root /, or as a **relative path** from the current working directory. To specify a path, each directory along the route from the source to the destination must be included in the path, with each directory in the sequence being separated by a slash. To help with the specification of relative paths, Linux provides the shorthand `"."` for the current directory and `".."` for the parent directory.

Directory	Typical Contents
/	The "root" directory
/bin	Essential low-level system utilities
/usr/bin	Higher-level system utilities and application programs
/sbin	Superuser system utilities (for performing system administration tasks)
/lib	Program libraries (collections of system calls that can be included in programs by a compiler) for low-level system utilities
/usr/lib	Program libraries for higher-level user programs
/tmp	Temporary file storage space (can be used by any user)
/home or /homes	User home directories containing personal file space for each user. Each directory is named after the login of the user.
/etc	Linux system configuration and information files
/dev	Hardware devices
/proc	A pseudo-filesystem which is used as an interface to the kernel. Includes a sub-directory for each active program (or process).

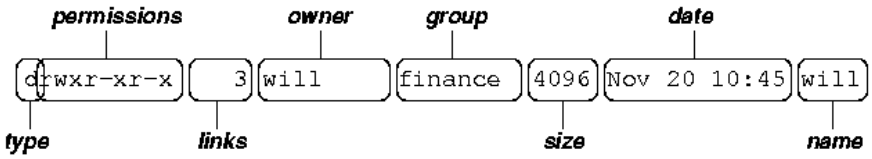
When you log into Linux, your current working directory is your user home directory. You can refer to your home directory at any time as "~" and the home directory of other users as "~<login>".

TRY THIS!

```
pwd
ls
```

- ls –a
- ls –a –l (or equivalently ls –al)

Each line of the output looks like this:



TRY THIS!

```
man ls
info ls
cd path
cd
cd -
mkdir
rmdir
cp source-file(s) destination
```

- cp -rd *source-directories destination-directory*

```
rm
```

- rm -i myfile

```
cat
   $cat > hello.txt
   hello world!
   [ctrl-d]
   $ ls hello.txt
   hello.txt
   $ cat hello.txt
   hello world!
   $
more and less
ls -l | more
```

Making Hard and Soft (Symbolic) Links

Direct **(hard)** and indirect **(soft or symbolic)** links from one file or directory to another can be created using the `ln` command.

```
$ ln filename linkname
```

creates another directory entry for *filename* called *linkname* (i.e. *linkname* is a hard link). Both directory entries appear identical (and both now have a link count of 2). If either *filename* or *linkname* is modified, the change will be reflected in the other file (since they are in fact just two different directory entries pointing to the same file).

```
$ ln -s filename linkname
```

creates a shortcut called *linkname* (i.e. *linkname* is a soft link). The shortcut appears as an entry with a special type ('l'):

```
$ ln -s hello.txt bye.txt
$ ls -l bye.txt
lrwxrwxrwx   1 will finance 13 bye.txt -> hello.txt
```

Specifying multiple filenames

Multiple filenames can be specified using special pattern-matching characters. The rules are:

- '?' matches any single character in that position in the filename.
- '*' matches zero or more characters in the filename. A '*' on its own will match all files. '*.*' matches all files with containing a '.'.
- Characters enclosed in square brackets ('[' and ']') will match any filename that has one of those characters in that position.
- A list of comma separated strings enclosed in curly braces ("{" and "}") will be expanded as a Cartesian product with the surrounding characters.

Quotes

As we have seen certain special characters (e.g. '*', '-','{' etc.) are interpreted in a special way by the shell. In order to pass arguments that use these characters to commands directly (i.e. without filename expansion etc.), we need to use special quoting characters. There are three levels of quoting that you can try:

1. Try insert a '\' in front of the special character.
2. Use double quotes (") around arguments to prevent most expansions.
3. Use single forward quotes (') around arguments to prevent all expansions.

There is a fourth type of quoting in Linux. Single backward quotes or backticks (`) are used to pass the output of some command as an input argument to another. For example:

```
$ hostname
  batsheba
$ echo this machine is called `hostname`
  this machine is called batsheba
```

File and Directory Permissions

Permission	File	Directory
read	User can look at the contents of the file	User can list the files in the directory
write	User can modify the contents of the file	User can create new files and remove existing files in the directory
execute	User can use the filename as a Linux command	User can change into the directory, but cannot list the files unless (s)he has read permission. User

		can read files if (s)he has read permission on them.

- `chmod <options> <files>`
- `chgrp`
- `chown`

Inspecting File Content

- `cat`
- `file`
- `head, tail`
- `objdump`
- `od`

Finding Files

- `find`
- `which`
- `locate`

Finding Text in Files

- `grep <options> <pattern> <files>`

Sorting Files

- `sort <filenames>`
- `uniq <filename>`

File Compression and Backup

- `tar`
- `cpio`
- `compress`
- `gzip`

Handling Removable Media

- `mount`
- `unmount`

Reference:

http://www.doc.ic.ac.uk/~wjk/UnixIntro/