Introduction to Using Linux

Author: H.L. John (School of Information Technology, Murdoch University)
Date Created 23rd July 2000
Last Modified: 23rd July 2000

Note: This is modified from a tutorial sheet originally used in my B227 Data Communications unit in Semester 1 2000.

1. MOTIVATION

To get you familiar enough with the linux command line to
1. Execute commands
2. Navigate around the linux file system
3. Create, delete and modify files

Most linux distributions today also come with graphical user interfaces (GUI). You may not need to learn all of the below if you are NOT using the machine remotely and your linux user account is set-up to start-up the GUI on log in.

In the instructions below, the words in Courier font, `like this`, are examples of what you may see on your screen. The words in Courier bold font, **`like this`**, you can type at the command line and execute.

These instructions are modified from http://floppix.ccai.com/labs.html.

2. LOGGING ONTO A LINUX SYSTEM

Users on a linux system need to log in before they are able to do anything. To be able to do so,
1. your system administrator must create an account for you on the linux machine, with a username and password;
2. if you are not physically at the machine, know the machine's network address, either as an IP number like `134.115.157.138`, or domain name like `red.it.murdoch.edu.au (`your system administrator will tell you this); and
3. make a connection to the machine which allows you to send executable commands. The most common way of doing this is using a TELNET program from another machine that has Internet connection. Using your TELNET program, connect to the address in (2).

Once you make a connection to the linux machine, you will see a login prompt like

```
Red Hat Linux release 6.1 (Cartman)
Kernel 2.2.12-20 on an i686
login:
```

Type in the username and password given to you by your system administrator. If your login is successful, you will get your user prompt like

```
[john@red john]$
```

The first part of the prompt indicates your username (`john`) and the name of the machine you are on (`red`), and the second part shows the directory you are currently in (`john`).

3. EXECUTING COMMANDS

Entering commands:

You control the operations of linux by typing in commands at your user prompt, or at the *command line*. Eg. if you type **`cal`** and press [Enter], linux will display a calendar for the current month:

```
[john@red john]$ cal
      July 2000
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
[john@red john]$
```

Command arguments:

Commands can have one or more arguments. For example:

1. **`cal`** (no arguments) prints a calendar for the current month.
2. **`cal 2020`** (one argument) prints a calendar for the year 2020.
3. **`cal 10 2000`** (10 and 2000 are both arguments) prints a calendar for October, 2000.

Command options:

You can also specify options which modify the behaviour of a command. Options appear immediately after the command, they are usually entered as a hyphen (-) followed by a single letter. For example:

1. **`cal -j 2000`** (j is an option) produces a julian calendar for the year 2000.
2. **`cal -y`** (y is an option) prints a calendar for the current year.
3. **`cal -yj`** (2 options) prints a Julian calendar for the current year.

Getting help:

The help system in linux is the **`man`** (short for manual) command. To get help about the cal command, type: **`man cal`**.

```
[john@red john]$ man cal
Formatting page, please wait...

CAL(1)              UNIX Reference Manual               CAL(1)

NAME
     cal - displays a calendar

SYNOPSIS
     cal [-mjy] [month [year]]

...
...
```

When viewing a man page, you can use the following keys:

[SpaceBar] - display the next screen
[Enter] - advance 1 line
b - go backwards
/xyz - search for the string xyz
q - quit

To get a listing of available commands on a certain subject, use **`man -k.`** Eg.
`man –k calendar`.

Command history:

The default environment you work in is a shell called bash. The bash shells saves a list of previous commands you have typed in. You can use the [up-arrow] key to recall previous commands, so you won't have to type them again. Once the command is displayed, you can execute it immediately by pressing [enter] or modify the command using the arrow and delete keys, and then press [enter].

Command completion:

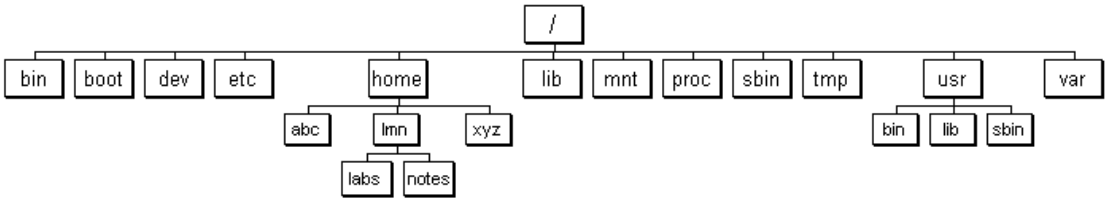
If you are entering long commands or filenames, you seldom have to type the entire string; type enough letters for bash to recognize the text and then press the [tab] key. Bash will fill in the remaining letters for you. If there is more than one option, the shell will beep. Pressing the [tab] key a second time will produce a list of possible matches for the text you have entered.

Ending commands:

If for whatever reason you make a mistake and are caught in the middle of a command execution, you can kill it by typing <Ctrl-C> (hold down the Ctrl key and press C). This usually gives a signal to the command process to terminate itself.

4. THE LINUX FILE SYSTEM

Here is an example diagram of what the structure of the linux file system looks like. Note that this is just an example, and your linux system may not have exactly the same structure.



Path:

1. An *absolute* path is the exact location of a file or directory **starting from the root directory**. For example: the absolute path to the abc directory is /home/abc .
   An absolute path always starts with /

2. A *relative* path gives the location of a file or directory **relative to the current directory**. If the working directory is /home, then:
   /home/abc is the absolute path to the abc directory from /home, the relative path to the abc directory is just abc

3. Some special symbols may be used in relative paths.
   . is the current directory
   .. is the parent directory
   ~ is your own home directory.
   If the working directory is /home/lmn/labs, then
   .. refers to /home/lmn, and
   ../notes refers to /home/lmn/notes

Commands for navigating through the file system:

1. **`pwd`** - This command prints the working directory. Use it to find out where you are in the directory tree.

2. **`cd`** - This command changes to a specified directory (access to some directories may be denied).
   eg: **`cd /`**
   eg: **`cd /home/xyz`**
   eg: **`cd ..`**

5. DIRECTORY LISTINGS

The **`ls`** command - examples of common options:
The **`ls`** command is used to show the contents of a directory. Some of the options of the ls command are:

1. **`ls`**
   List the contents of the current directory

2. **`ls /etc`**
   List the contents of the /etc directory

3. **`ls -a`**
   Display all entries in the current directory including those beginning with a period. File and directory names that start with a period are "hidden" files in unix; they do not appear in ordinary directory listings.

4. **`ls -l`**
   Display a detailed listing of the current directory; one line of data for each entry

5. **`ls --help`**
   The -h option is the "help" option, this gives a list of all of the options for the `ls` command and brief explanations.

6. To get a list of all of the other options of ls, use the man command. **`man ls`**.

Fields in a directory listing entry:

There are a number of things that appear in a unix directory listing:

1. Permissions:
   Users and the system administrator need a way to control the way other users can access files and directories. The standard unix filesystem does this with permissions. Each file has read, write and execute permission set for the owner, the group and then all other users.

2. Links:
   Users can create links to existing files. The filesystem keeps track of the number of links.

3. Owner and group:
   Unix is a multi-user operating system. For each file and directory, the filesystem keeps track of the file owner and group.

4. Size in bytes:

5. Date:
   The unix filesystem actually maintains 3 dates for each file: the last access date, the last modification date (the last date that the file contents were modified) and the last change date (the last date that the file "inode" or directory entry was modified). The date that appears in a standard directory entry in the last modification date.

6. Filename:

A sample directory entry:

```
[john@red notes]$ ls -l
total 4
-rwxrw-r--   1 john   staff     7 Jul 23 15:41 lectures
[john@red john]$
```

Description of the line:

```
-rwxrw-r--   1 john   staff     7 Jul 23 15:41 lectures
```

`-`	This is an ordinary file
`rwx`	The permissions for the owner (john): read, write, execute
`rw-`	The permissions for the group (staff): read, write, not execute
`r--`	The permissions for others: read, not write, not execute
`1`	The number of links to the file
`john`	The owner
`staff`	The group
`12`	The size (in bytes)
`Mar 7 12:35`	The last time that the file was modified
`lectures`	The name of the file

6. SOME DIRECTORY COMMANDS:

Some directory commands:

1. **`pwd`** - print working directory (from the lab on navigating the filesystem)

2. **`cd`** - change directory (from the lab on navigating the filesystem)

3. **`ls`** - display a directory listing (from the lab on directory listings)

4. **`mkdir`** - This command creates a new directory. Examples:
   `mkdir bin` – creates a directory called bin in the current directory
   `mkdir ~/tmp` – creates a directory called tmp in your home directory

5. **`rmdir`** - This command removes an existing directory. The directory must be empty before it can be removed. Example:
   `rmdir ~/tmp`

Note: You can only use `mkdir` and `rmdir` in the directories which you have write permissions to.

7. WILDCARD CHARACTERS

The bash shell allows you to use wildcard characters for paths:

* * Matches any string, including the null string.
* ? Matches any single character.

Eg.

`ls lecture*`
list all files in the current directory which has a name starting with lecture.

`ls lecture?.zip`
list all files in the current directory with the name lecture1.zip, lecture2.zip, lectureA.zip, etc.
But not lecture11.zip

8. FILE COMMANDS:

Some file commands:

`cat` - display file contents. Eg.

```
cat  /etc/passwd
```

`more` - display file contents one screen at a time. Eg.

```
more  /etc/passwd
```

Keystroke commands for `more`:

[SpaceBar] - display the next screen
[Enter] - advance 1 line
b - go backwards
/xyz - search for the string xyz
q – quit

`less` - a more powerful version of `more`. The keystrokes for `more` can also be used in `less`.

```
less  /etc/passwd
```

`head` - display lines from the beginning of a file The default is 10 lines.

```
head  -5  /etc/passwd      - displays the first 15 lines of the file /etc/passwd
```

`tail` - display lines from the end of a file

```
tail  -5  /etc/passwd      - displays the last 5 lines of the file /etc/passwd
```

9. FINDING FILES USING FILENAMES

The find command searches for files in the filesystem by name, owner, size, date, ...

Examples:

`find /home -name readme`
This starts the search in the directory /home and looks for a file named readme.

`find / -user john`
This starts the search at the root directory and looks for all files which belong to john.

As find scans through the directory tree, there will be directories which you do not have permissions to search. The find command will not scan these directories but will display permission denied messages on standard error. To suppress the permission denied messages, redirect standard error to /dev/null.

`find / -mtime -1 2> /dev/null`
This searches for all files that were modified in the last day (24 hours) starting the search in the root directory and sending all 'permission denied' messages to the bit bucket

10. FINDING FILES USING FILE CONTENTS

The `grep` command searches the contents of a file for lines that contain a certain pattern.

Examples:

`grep john /etc/passwd`
Display the lines from the file /etc/passwd that contain the string john.

`grep john *`
Display the lines from any file the in the current directory that contain the string john.

`grep -v root /etc/passwd`
Display the lines from the file /etc/passwd that do not contain the string root.

`grep -n root /etc/passwd`
Display line numbers and the lines from the file /etc/passwd that contain the string root.

`grep ^f /etc/passwd`
Display the lines from the file /etc/passwd that start with f.

`grep h$ /etc/passwd`
Display the lines from the file /etc/passwd that end with h.

11. STANDARD I/O AND REDIRECTION:

1. Standard I/O: most unix commands are setup to:
   get input from standard input (generally the keyboard)
   send output to standard output (the display screen)
   send error messages to standard error (also the display screen).

2. Redirecting output. When you redirect standard output, the data that would normally appear on the screen is stored in a file instead. Redirect output to a file using > or >>

   `echo "first line" > flist.txt`
   puts the word "first line" in the file flist.txt ; if flist.txt already exists, it is overwritten ; `echo` is a command which prints out whatever arguments you give it. Since flist.txt is in the current directory, you must have write permission in the current directory to do this. You can view the file using the `cat` command

   `echo "another line" >> flist.txt`
   appends the another line to the file flist.txt; if flist.txt does not exist, it is created

3. Redirecting input. When you redirect standard input, any data that would normally be entered from the keyboard is obtained from a file instead. Redirect input from a file using <

   `cat < /etc/passwd`
   sends the contents of /etc/passwd to the `cat` command, which normally reads from the keyboard.

4. Redirecting standard error. Redirect standard error using 2>

   `ls –l /nowhere 2> /dev/null`
   tries to do a listing of the file /nowhere, and sends any error messages to /dev/null; /dev/null is a rubbish tip for output which is no longer needed. Compare with:
   `ls –l /nowhere`

12. PIPES AND FILTERS:

1. A pipe takes the output from one command and uses it as input to the next command. Example:

   `ls -l /etc | more`
   display a long listing of the /etc directory and pipe the output through the more command so that it is displayed on the screen one screen at a time.

2. To send the output of a command to a file, use redirection
   To send the output of a command to another command, use piping

3. Commands that are used in pipelines to modify the output from other commands are being used as filters. All of these commands can be used on their own as well. Commands that are frequently used in pipelines include: `grep, sort, wc`

4. `wc`: Count bytes, words and lines

   `wc /etc/passwd`
   Count the number of bytes, words and lines in /etc/passwd

   `wc -l /etc/passwd`
   Count the number of lines in /etc/passwd

   `wc -w /etc/login.defs`
   Count the number of words in /etc/passwd

5. `sort`: sort lines in a file

   `sort /etc/group`
   Sort the contents of /etc/group

`sort -r /etc/group`
Sort the contents of /etc/group in reverse order

`ls -l | sort -k5n`
Sort the output of ls -l on the size of the files. k5 specifies sort starting on field 5 and n selects a sort in numerical order

6. More examples:

`ls /var/spool/mail | wc –w`
Count the number of files in the directory var/spool/mail

`grep /bin/bash$ /etc/passwd | wc -l`
Count the number of users whose login shell is /bin/bash? (Recall that the login shell is the last entry in the line for each user in the file /etc/passwd

`who | sort`
Display a list of everyone who is logged on sorted by their usernames

13. TEXT EDITING:

1. To edit the contents of a file (rather then just view it), you need a text editor. A simple editor available in most linux systems is pico. The editor by default contains on-screen help. Try creating a new file temp.txt in the current directory by:

```
pico temp.txt
```

2. Another commonly text editor for power users is `vi`, for visual (a complete misnomer), or `vim`, for Vi IMproved. This editor requires significant learning curve to use efficiently. For systems with vim, type the command **`vimtutor`** for a tutorial.