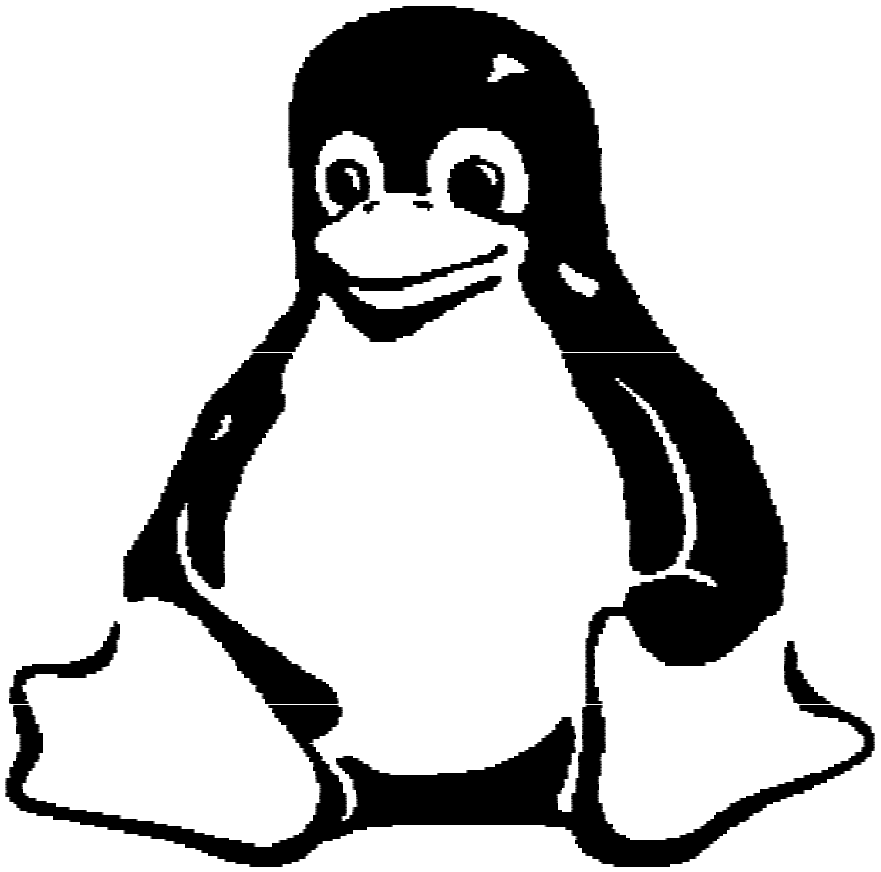
An Introduction to the Linux Command Shell For Beginners



Presented by:

Victor Gedris

In Co-Operation With:

The Ottawa Canada Linux Users Group

and

ExitCertified

Copyright and Redistribution

This manual was written with the intention of being a helpful guide to Linux users who are trying to become familiar with the Bash shell and basic Linux commands. To make this manual useful to the widest range of people, I decided to release it under a free documentation license, with the hopes that people benefit from it by updating it and re-distributing modified copies. You have permission to modify and distribute this document, as specified under the terms of the GNU Free Documentation License. Comments and suggestions for improvement may be directed to: vic@gedris.org.

This document was created using an Open Source office application called *Open Office*. The file format is non-proprietary, and the document is also published in various other formats online. Updated copies will be available on Vic Gedris' web site [http://vic.dyndns.org/]. For more information on Open Office, please visit http://www.openoffice.org/.

Document Version: 1.2, 2003-06-25

1.0 Introduction

The purpose of this document is to provide the reader with a fast and simple introduction to using the Linux command shell and some of its basic utilities. It is assumed that the reader has zero or very limited exposure to the Linux command prompt. This document is designed to accompany an instructor-led tutorial on this subject, and therefore some details have been left out. Explanations, practical examples, and references to DOS commands are made, where appropriate.

1.1 What is a command shell?

- A program that interprets commands
- Allows a user to execute commands by typing them manually at a terminal, or automatically in programs called *shell scripts*.
- A shell is *not* an operating system. It is a way to interface with the operating system and run commands.

1.2 What is BASH?

- BASH = **B**ourne **A**gain **SH**ell
- Bash is a shell written as a free replacement to the standard Bourne Shell (`/bin/sh`) originally written by Steve Bourne for UNIX systems.
- It has all of the features of the original Bourne Shell, plus additions that make it easier to program with and use from the command line.
- Since it is Free Software, it has been adopted as the default shell on most Linux systems.

1.3 How is BASH different from the DOS command prompt?

- **Case Sensitivity:** In Linux/UNIX, commands and filenames are case sensitive, meaning that typing "`EXIT`" instead of the proper "`exit`" is a mistake.
- **"\" vs. "/":** In DOS, the forward-slash "`/`" is the command argument delimiter, while the backslash "`\`" is a directory separator. In Linux/UNIX, the "`/`" is the directory separator, and the "`\`" is an escape character. More about these special characters in a minute!
- **Filenames:** The DOS world uses the "eight dot three" filename convention, meaning that all files followed a format that allowed up to 8 characters in the filename, followed by a period ("dot"), followed by an option extension, up to 3 characters long (e.g. `FILENAME.TXT`). In UNIX/Linux, there is no such thing as a file extension. Periods can be placed at any part of the filename, and "extensions" may be interpreted differently by all programs, or not at all.

1.4 Special Characters

Before we continue to learn about Linux shell commands, it is important to know that there are many symbols and characters that the shell interprets in special ways. This means that certain typed characters: a) cannot be used in certain situations, b) may be used to perform special operations, or, c) must be "escaped" if you want to use them in a normal way.

Character	*Description*
`\`	Escape character. If you want to reference a special character, you must "escape" it with a backslash first. Example: `touch /tmp/filename\*`
`/`	Directory separator, used to separate a string of directory names. Example: `/usr/src/linux`
`.`	Current directory. Can also "hide" files when it is the first character in a filename.
`..`	Parent directory
`~`	User's home directory
`*`	Represents 0 or more characters in a filename, or by itself, all files in a directory. Example: `pic*2002` can represent the files `pic2002`, `picJanuary2002`, `picFeb292002`, etc.
`?`	Represents a single character in a filename. Example: `hello?.txt` can represent `hello1.txt`, `helloz.txt`, but not `hello22.txt`
`[ ]`	Can be used to represent a range of values, e.g. `[0-9]`, `[A-Z]`, etc. Example: `hello[0-2].txt` represents the names `hello0.txt`, `hello1.txt`, and `hello2.txt`
`|`	"Pipe". Redirect the output of one command into another command. Example: `ls | more`
`>`	Redirect output of a command into a new file. If the file already exists, over-write it. Example: `ls > myfiles.txt`
`>>`	Redirect the output of a command onto the end of an existing file. Example: `echo "Mary 555-1234" >> phonenumbers.txt`
`<`	Redirect a file as input to a program. Example: `more < phonenumbers.txt`
`;`	Command separator. Allows you to execute multiple commands on a single line. Example: `cd /var/log ; less messages`
`&&`	Command separator as above, but only runs the second command if the first one finished without errors. Example: `cd /var/logs && less messages`
`&`	Execute a command in the background, and immediately get your shell back. Example: `find / -name core > /tmp/corefiles.txt &`

1.5 Executing Commands

The Command `PATH`:

- Most common commands are located in your shell's "`PATH`", meaning that you can just type the name of the program to execute it.

  Example: Typing "`ls`" will execute the "`ls`" command.

- Your shell's "`PATH`" variable includes the most common program locations, such as `/bin`, `/usr/bin`, `/usr/X11R6/bin`, and others.

- To execute commands that are not in your current PATH, you have to give the complete location of the command.

  Examples: `/home/bob/myprogram`

  `./program` (Execute a program in the current directory)

  `~/bin/program` (Execute program from a personal `bin` directory)

Command Syntax

- Commands can be run by themselves, or you can pass in additional arguments to make them do different things. Typical command syntax can look something like this:

  ```
  command [-argument] [-argument] [--argument] [file]
  ```

- Examples:

`ls`	List files in current directory
`ls -l`	Lists files in "long" format
`ls -l --color`	As above, with colourized output
`cat filename`	Show contents of a file
`cat -n filename`	Show contents of a file, with line numbers

2.0 Getting Help

When you're stuck and need help with a Linux command, help is usually only a few keystrokes away! Help on most Linux commands is typically built right into the commands themselves, available through online help programs ("man pages" and "info pages"), and of course online.

2.1 Using a Command's Built-In Help

Many commands have simple "help" screens that can be invoked with special command flags. These flags usually look like "`-h`" or "`--help`".

Example: `grep --help`

2.2 Online Manuals: "Man Pages"

The best source of information for most commands can be found in the online manual pages, known as "man pages" for short. To read a command's man page, type "`man command`".

Examples: `man ls` Get help on the "ls" command.

`man man` A manual about how to use the manual!

To search for a particular word within a man page, type "`/word`". To quit from a man page, just type the "Q" key.

Sometimes, you might not remember the name of Linux command and you need to search for it. For example, if you want to know how to change a file's permissions, you can search the man page descriptions for the word "permission" like this:

```
man -k permission
```

If you look at the output of this command, you will find a line that looks something like:

```
chmod                (1)  - change file access permissions
```

Now you know that "`chmod`" is the command you were looking for. Typing "`man chmod`" will show you the `chmod` command's manual page!

2.3 Info Pages

Some programs, particularly those released by the Free Software Foundation, use info pages as their main source of online documentation. Info pages are similar to man page, but instead of being displayed on one long scrolling screen, they are presented in shorter segments with links to other pieces of information. Info pages are accessed with the "`info`" command, or on some Linux distributions, "`pinfo`" (a nicer info browser).

For example: `info df` Loads the "`df`" info page.

3.0 Navigating the Linux Filesystem

The Linux filesystem is a tree-like hierarchy hierarchy of directories and files. At the base of the filesystem is the "/" directory, otherwise known as the "root" (not to be confused with the root user). Unlike DOS or Windows filesystems that have multiple "roots", one for each disk drive, the Linux filesystem mounts all disks somewhere underneath the / filesystem. The following table describes many of the most common Linux directories.

3.1 The Linux Directory Layout

Directory	*Description*
	The nameless base of the filesystem. All other directories, files, drives, and devices are attached to this root. Commonly (but incorrectly) referred to as the "slash" or "/" directory. The "/" is just a directory separator, not a directory itself.
/bin	Essential command binaries (programs) are stored here (`bash`, `ls`, `mount`, `tar`, etc.)
/boot	Static files of the boot loader.
/dev	Device files. In Linux, hardware devices are acceessd just like other files, and they are kept under this directory.
/etc	Host-specific system configuration files.
/home	Location of users' personal home directories (e.g. `/home/susan`).
/lib	Essential shared libraries and kernel modules.
/proc	Process information pseudo-filesystem. An interface to kernel data structures.
/root	The root (superuser) home directory.
/sbin	Essential system binaries (`fdisk`, `fsck`, `init`, etc).
/tmp	Temporary files. All users have permission to place temporary files here.
/usr	The base directory for most shareable, read-only data (programs, libraries, documentation, and much more).
/usr/bin	Most user programs are kept here (`cc`, `find`, `du`, etc.).
/usr/include	Header files for compiling C programs.
/usr/lib	Libraries for most binary programs.
/usr/local	"Locally" installed files. This directory only really matters in environments where files are stored on the network. Locally-installed files go in `/usr/local/bin`, `/usr/local/lib`, etc.). Also often used for software packages installed from source, or software not officially shipped with the distribution.
/usr/sbin	Non-vital system binaries (`lpd`, `useradd`, etc.)
/usr/share	Architecture-independent data (icons, backgrounds, documentation, terminfo, man pages, etc.).
/usr/src	Program source code. E.g. The Linux Kernel, source RPMs, etc.
/usr/X11R6	The X Window System.
/var	Variable data: mail and printer spools, log files, lock files, etc.

3.2 Commands for Navigating the Linux Filesystems

The first thing you usually want to do when learning about the Linux filesystem is take some time to look around and see what's there! These next few commands will: a) Tell you where you are, b) take you somewhere else, and c) show you what's there. The following table describes the basic operation of the `pwd`, `cd`, and `ls` commands, and compares them to certain DOS commands that you might already be familiar with.

Linux Command	*DOS Command*	*Description*
`pwd`	`cd`	"Print Working Directory". Shows the current location in the directory tree.
`cd`	`cd, chdir`	"Change Directory". When typed all by itself, it returns you to your home directory.
`cd directory`	`cd directory`	Change into the specified directory name. Example: `cd /usr/src/linux`
`cd ~`		"~" is an alias for your home directory. It can be used as a shortcut to your "home", or other directories relative to your home.
`cd ..`	`cd..`	Move up one directory. For example, if you are in `/home/vic` and you type "`cd ..`", you will end up in `/home`.
`cd -`		Return to previous directory. An easy way to get back to your previous location!
`ls`	`dir /w`	List all files in the current directory, in column format.
`ls directory`	`dir directory`	List the files in the specified directory. Example: `ls /var/log`
`ls -l`	`dir`	List files in "long" format, one file per line. This also shows you additional info about the file, such as ownership, permissions, date, and size.
`ls -a`	`dir /a`	List all files, including "hidden" files. Hidden files are those files that begin with a ".", e.g. The `.bash_history` file in your home directory.

`ls -ld directory`		A "long" list of "directory", but instead of showing the directory contents, show the directory's detailed information. For example, compare the output of the following two commands:
		`ls -l /usr/bin`
		`ls -ld /usr/bin`

`ls /usr/bin/d*`	`dir d*.*`	List all files whose names begin with the letter "d" in the `/usr/bin` directory.

4.0 Piping and Re-Direction

Before we move on to learning even more commands, let's side-track to the topics of piping and re-direction. The basic UNIX philosophy, therefore by extension the Linux philosophy, is to have many small programs and utilities that do a particular job very well. It is the responsibility of the programmer or user to combine these utilities to make more useful command sequences.

4.1 Piping Commands Together

The pipe character, "|", is used to chain two or more commands together. The output of the first command is "piped" into the next program, and if there is a second pipe, the output is sent to the third program, etc. For example:

```
ls -la /usr/bin | less
```

In this example, we run the command "`ls -la /usr/bin`", which gives us a long listing of all of the files in `/usr/bin`. Because the output of this command is typically very long, we pipe the output to a program called "`less`", which displays the output for us one screen at a time.

4.2 Redirecting Program Output to Files

There are times when it is useful to save the output of a command to a file, instead of displaying it to the screen. For example, if we want to create a file that lists all of the MP3 files in a directory, we can do something like this, using the "**>**" redirection character:

```
ls -l /home/vic/MP3/*.mp3 > mp3files.txt
```

A similar command can be written so that instead of creating a new file called `mp3files.txt`, we can append to the end of the original file:

```
ls -l /home/vic/extraMP3s/*.mp3 >> mp3files.txt
```

5.0 Other Linux Commands

The following sections describe many other commands that you will find on most Linux systems. I can't possibly cover the details of all of these commands in this document, so don't forget that you can check the "man pages" for additional information. Not all of the listed commands will be available on all Linux or UNIX distributions.

5.1 Working With Files and Directories

These commands can be used to: find out information about files, display files, and manipulate them in other ways (copy, move, delete).

Linux Command	*DOS Command*	*Description*
file		Find out what kind of file it is. For example, "`file /bin/ls`" tells us that it is a Linux executable file.
cat	`type`	Display the contents of a text file on the screen. For example: `cat mp3files.txt` would display the file we created in the previous section.
head		Display the first few lines of a text file. Example: `head /etc/services`
tail		Display the last few lines of a text file. Example: `tail /etc/services`
tail -f		Display the last few lines of a text file, and then output appended data as the file grows (very useful for following log files!). Example: `tail -f /var/log/messages`
cp	`copy`	Copies a file from one location to another. Example: `cp mp3files.txt /tmp` (copies the mp3files.txt file to the /tmp directory)
mv	`rename, ren, move`	Moves a file to a new location, or renames it. For example: `mv mp3files.txt /tmp` (copy the file to /tmp, and delete it from the original location)
rm	`del`	Delete a file. Example: `rm /tmp/mp3files.txt`
mkdir	`md`	Make Directory. Example: `mkdir /tmp/myfiles/`
rmdir	`rd, rmdir`	Remove Directory. Example: `rmdir /tmp/myfiles/`

5.2 Finding Things

The following commands are used to find files. "`ls`" is good for finding files if you already know approximately where they are, but sometimes you need more powerful tools such as these:

Linux Command	*Description*
`which`	Shows the full path of shell commands found in your path. For example, if you want to know exactly where the "grep" command is located on the filesystem, you can type "`which grep`". The output should be something like: `/bin/grep`
`whereis`	Locates the program, source code, and manual page for a command (if all information is available). For example, to find out where "`ls`" and its man page are, type: "`whereis ls`" The output will look something like: `ls: /bin/ls /usr/share/man/man1/ls.1.gz`
`locate`	A quick way to search for files anywhere on the filesystem. For example, you can find all files and directories that contain the name "mozilla" by typing: `locate mozilla`
`find`	A *very* powerful command, but sometimes tricky to use. It can be used to search for files matching certain patterns, as well as many other types of searches. A simple example is: `find . -name \*mp3` This example starts searching in the current directory "." and all sub-directories, looking for files with "mp3" at the end of their names.

5.3 Informational Commands

The following commands are used to find out some information about the user or the system.

Linux Command	*Explanation*
`ps`	Lists currently running process (programs).
`w`	Show who is logged on and what they are doing.
`id`	Print your user-id and group id's
`df`	Report filesystem disk space usage ("Disk Free" is how I remember it)
`du`	Disk Usage in a particular directory. "`du -s`" provides a summary for the current directory.
`top`	Displays CPU processes in a full-screen GUI. A great way to see the activity on your computer in real-time. Type "Q" to quit.
`free`	Displays amount of free and used memory in the system.
`cat /proc/cpuinfo`	Displays information about your CPU.
`cat /proc/meminfo`	Display lots of information about current memory usage.
`uname -a`	Prints system information to the screen (kernel version, machine type, etc.)

5.4 Other Utilities

Here are some other commands that are useful to know.

Linux Command	*Description*
clear	Clear the screen
echo	Display text on the screen. Mostly useful when writing shell scripts. For example: echo "Hello World"

more	Display a file, or program output one page at a time. Examples:
	`more mp3files.txt`
	`ls -la | more`

less	An improved replacement for the "`more`" command. Allows you to scroll backwards as well as forwards.
grep	Search for a pattern in a file or program output. For example, to find out which TCP network port is used by the "nfs" service, you can do this: `grep "nfs" /etc/services` This looks for any line that contains the string "nfs" in the file "/etc/services" and displays only those lines.
lpr	Print a file or program output. Examples: `lpr mp3files.txt` - Print the mp3files.txt file `ls -la | lpr` - Print the output of the "`ls -la`" command.
sort	Sort a file or program output. Example: `sort mp3files.txt`
su	"Switch User". Allows you to switch to another user's account temporarily. The default account to switch to is the root/superuser account. Examples: `su` - Switch the root account `su -` - Switch to root, and log in with root's environment `su larry` - Switch to Larry's account

5.5 Shortcuts to Make it all Easier!

When you start using the Bash shell more often, you will appreciate these shortcuts that can save you very much typing time.

Shortcut	*Description*
Up/Down Arrow Keys	Scroll through your most recent commands. You can scroll back to an old command, hit ENTER, and execute the command without having to re-type it.
"`history`" command	Show your complete command history.
TAB Completion	If you type a partial command or filename that the shell recognizes, you can have it automatically completed for you if you press the TAB key. Try typing the first few characters of your favourite Linux command, then hit TAB a couple of times to see what happens.

Complete recent commands with "!"	Try this: Type "`!`" followed by the first couple of letters of a recent command and press ENTER! For example, type:
	`find /usr/bin -type f -name m\*`
	...and now type:
	`!fi`

Search your command history with `CTRL-R`	Press `CTRL-R` and then type any portion of a recent command. It will search the commands for you, and once you find the command you want, just press ENTER.
Scrolling the screen with `Shift-PageUp` and `Page Down`	Scroll back and forward through your terminal.

6.0 Further Reading

Link Address	*Description*
http://www.oclug.on.ca	Ottawa Canada Linux Users Group. A group with an active mailing list, monthly meetings, and much more.
http://www.exitcertified.com	Ottawa's source for Sun training, and the host of OCLUG's technology seminars.
http://www.fsf.org	The Free Software Foundation. Documentation, source code, and much more for many programs commonly found on Linux systems.
http://linux.org.mt/article/terminal	"A Beginner's Bash". Another very good introduction to Bash.
http://www.oreilly.com/catalog/bash2	An excellent book if you want to learn how to customize Bash and use it for shell script programming.