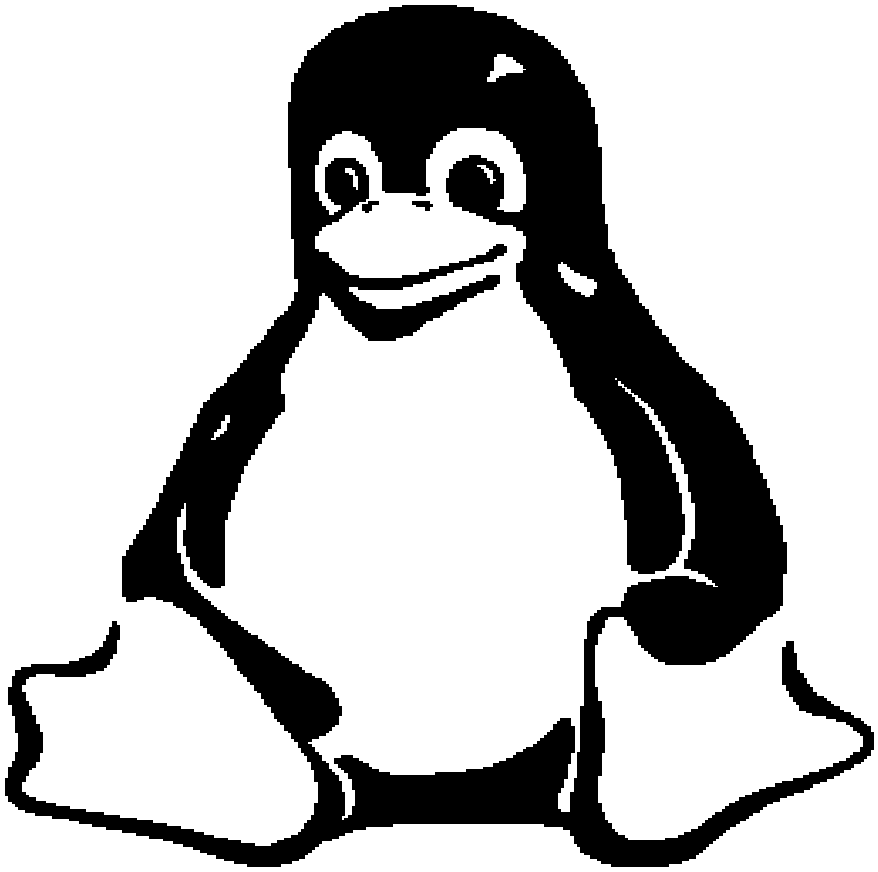
An Introduction to the Linux Command Shell For Beginners



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