

## Preface

While not an expert on the subject, I hope to achieve an appropriate blend of simplicity and technicality to suit this audience. Most of this material is valid for any Linux distribution and many shells.

However, my focus is Ubuntu 6.06 using a `bash` (Bourne Again SHell) shell in terminal windows of the Gnome Desktop Environment.

I will cover some simple customization, shell features and some common Linux commands. Along the way I may diverge into background and/or historical aspects which I hope will be of interest. Once again, this talk is not meant to be thorough coverage of any one subject, but I hope will prove helpful or enlightening to someone.

## Background

Unix and DOS both originated with textual command line interfaces only. Windows evolved as a commercial entity from DOS by aiming at ease of use for general consumers. Microsoft did not invent the GUI (Graphics User Interface) but did make it a major focus for Windows as one means of making computing usable and attractive to the general public as well as to corporate users outside of IT (Information Technology) departments.

Unix evolved primarily in IT shops and universities where the focus was more on raw power and usability to technical people. In a time when computing resources were limited and expensive, GUIs were considered wasteful of system resources. GUIs and GUI tools evolved in a few research labs, universities, and companies with a focus on the future and non-technical users, but command line remained king in the unix world for getting jobs done.

That remains true for Linux techies despite increasingly easy to use Linux distros and applications. GUI based applications and system tools are available now for virtually every aspect of using Linux systems. Nevertheless, the command line interface remains an extremely powerful tool for those who understand its use. It will long remain a basic in Linux due to the ability to quickly develop powerful and low overhead tools and versatility in investigating and resolving issues.

Delving deeply into command line uses and tool development is not for the faint of heart and is not the intent here. This is merely an attempt to show the command line is also useful to new users.

## Shells

Shells (`bash`, `csh`, `ksh`, `sh`, etc.) in Linux are command line interpreters as are `command.com` and `cmd.exe` in the DOS and Windows. Scripts in Linux are collections of commands to be processed by the command line interpreter as are batch files in DOS. Anyone who has used command lines and batch files in DOS and/or a Windows command prompt has no reason to feel intimidated. Using a shell (or command prompt) in Linux should already be a somewhat familiar environment.

The most basic difference from DOS is that you have the choice which shell to use. The most basic differences between shells are what commands they implement internally. You can use any or all of them as you choose. We will focus on doing things in `bash` since it is provided with virtually every

Linux distribution and is commonly the default shell. But most of this material is not specific to bash.

Comparing command line capabilities in Linux to those of DOS is rather like comparing a Hummer to a Model T. Anyone can drive either one but you can do a lot more with the Hummer. Basically that is an accident of history as explained earlier. In fact, Microsoft is bringing back the command line in a much more powerful way akin to Linux shells.

## Helpful Shell Features

These are mostly not unique to bash and may operate somewhat differently in other shells. They are also only a minuscule portion of the features available and partial descriptions of the ones listed.

**Path & File Completion** is a feature that lets the user hit the TAB key to tell bash to try to complete the command instead of the user [mis]typing it. Bash will fill in as much as it is able to do unambiguously and then stop to allow the user to accept the result or add more characters to finish or disambiguate<sup>1</sup> it (and then press TAB again if desired). Should the user just press TAB a second time, the shell will display the available choices and redisplay the command line for the user to continue.

**Copy & paste (3 button mouse alternative)** in a terminal program is very handy. There was no such thing in the days of the simple terminal which had no mouse. Today programs which emulate terminals have an alternative to the typical GUI based copy and paste. Double clicking selects an entire “word” and triple clicking selects an entire line. The user may also click and drag to highlight desired text in the terminal window. Once highlighted, the user can click the middle button on a three button mouse to paste the highlighted at the current text cursor<sup>2</sup> location.

**History** is available in various shells and while quite similar they do not work identically. Basically the shell stores commands you execute in a history file. The user can display that history but can also reuse commands and sequences of commands without re-typing them. He/she can also select and edit a previously run command before executing the modified command. Bash users can use:

- up and down arrows to move through the history list one line at a time [optionally editing the line as mentioned below] followed by ENTER to execute it or ctrl-C to abort.
- history to see the numbered list of previous commands.
- ! followed by a number and ENTER to execute the so numbered command.
- ! followed by the initial characters of a previous command and ENTER to execute the last command which started with those characters
- Ctrl-R to initiate a reverse search, type initial characters of a previous command followed by another ctrl-R to search back further, ENTER to execute the command found, ctrl-C to abort, or ctrl-O to execute it and on completion of that command, place the next command from the history list on the command line ready for execution or editing.

---

1 Given two files, bobs.book and my\_bobsphone.nbr in the current directory, typing `ls b<TAB>` would fill in as `ls bobs` unambiguously. Then typing either a '!' or a 'p' would disambiguate so that another `<TAB>` would fill in the rest.

2 There was no mouse with a terminal historically. The ONLY cursor was a text cursor and by definition, the point at which the next character would be typed. Mouse pointers are not truly cursors although we have come to call them so. A mouse pointer in a terminal window is independent from the text cursor.

**Command Editing** using the left and right arrows, HOME and END keys in conjunction with backspace, delete, and/or character keys to edit a command typed or recalled from history. The INSERT key toggles insert/overwrite mode. This works very much like DOSKEY in Windows.

## Customization

Just about everything in Linux can be customized one way or another. Most customization is done by changing configuration files whether by editing the files directly or using a GUI based configuration program. While the GUI configuration programs frequently easier to use and sometimes easier to understand, they are generally not as flexible as directly editing the configuration files.

Each bash user has `.bashrc` and `.bash_profile` files in their home directory. These began as copies of system wide defaults and were copied to the user's home directory when the user was created. If changes are made to the system default files, the changes become the default for all new users. After a user is created, changes can be made to the local files without affecting other users. This allows each user to customize as they please but avoids every user having to setup all of the most common things.

**Aliases** are frequently defined in a shell configuration file like `.bashrc` even though they can be done other ways as well. They are used to modify existing commands and create new ones. For example:

```
alias ls='ls --color=auto --classify'
```

causes the existing ls command to display results color coded and with special symbols that help the user understand what kind of files are being listed.

```
alias ll='ls -l'
```

creates a command which makes the colored, annotated listing display in long form with much more information about each file.

```
alias dir='ls --color=auto --format=vertical'
```

creates a new command which works much like the DOS dir command.

```
alias la='ls -a'
```

creates a new command that works like ls but also shows hidden files.

These aliases are so common that most default resource files include them (although some may be commented out). Looking at your `.bashrc` file is a good place to see some of the possibilities and raise a few questions/suggestions. The manual page on bash runs over 5000 lines of terse language and a Google search on “customize bash” yields 2.2 million hits. You can spend years finding all the customizations you would like.

The Command Line **Prompt** is also very customizable. The syntax for doing so can be very obtuse but there are many examples available on the web in addition to some handy ones right in your `.bashrc` file. I like to use a highly visible (colored) prompt that gives useful information such as the current directory. Some folks like to see the time of day or even the date. Networked users may wish the prompt to contain the user name by which they are logged on as along with the name of the machine they are logged on to.

Add a **Terminal Icon** to the panel if you use the command line a lot. You can do that by going to the Applications/Accessories menu, right clicking on the Terminal selection and then click the Add to

Panel selection from the context menu. Having done so, you may wish to open a terminal and customize the Current Profile from the Edit menu.

**Terminal appearance** seems mostly unimportant but there are several issues of varying import you may want to be aware of. People commonly prefer black characters on a white background because they are used to paper. Some experts contend that white on black is easier on the eyes for computer users. Look for a menu entry for Profiles or Preferences where you can modify a lot of things for the terminal.

Linux does not allow you to execute a command in your current directory just by naming it for security reasons. It also prevent the common user from placing executables in the directories used system wide for executables. Yet users, and especially command line users frequently create executables they wish to run with frequency. One way to do this is to create a **~/bin directory** for your private executables.

```
mkdir ~/bin          # remember ~ is short for /home/user
chmod 700 ~/bin      # read/write/execute by owner only
PATH=~:/bin:$PATH   # you could do this in a startup script which needs
                    # different syntax (see ~/.bash_profile for an example)
```

One of the very strengths of Linux is the security that **no normal user has Administrative or Super User privileges**. Many modern Linux distributions [including Ubuntu] make the super user, or root, account with no password so it cannot be logged into. The idea is to prevent people using the big gun account which is far more capable of shooting off the systems feet. Forcing the system administrator/user to use a special syntax when they do dangerous things is supposed to make them more alert and less likely to do harm even accidentally.

If you need to **become the root user** but still want to use some GUI programs, Issue **sudo su** (and your password when requested) to become the root user but retain your normal user environment which owns the display. You will need to start any gui programs from the command line since your normal user still owns the desktop. For example, typing **firefox** will open a browser owned by root.

Nevertheless, I find it convenient to **enable logging in as root** even if I rarely use it. You can do this with the command **sudo passwd root**. You will need to enter your password (assuming you have sudo privileges) followed by entering a password twice for root. Make it a strong password and not one you use elsewhere.

## Redirection

Linux systems (and all it's parents and siblings) have an extremely strong history of having programs designed to take input from one file (stdin), provide output to another (stdout) and send error messages to a third (stderr). Stdin is normally the keyboard, stdout is the display, and stderr is also the display. That sounds very simplistic, but it is actually quite elegant due to redirection. Redirection allows them to be nearly anything besides a keyboard or display. As a result, simplistic programs designed to use stdin, stdout, and stderr can be connected together like parts of a plumbing system to do bigger jobs.

This is a great source of flexibility and power for the command line user. The `|` symbol can be used to redirect stdout from one program to stdin of another. One can use “`< fileX`” to tell a program to get its input from fileX instead of stdin and “`> fileY`” to redirect its output from stdout to fileY. The program known as tee takes input from stdin and sends it both to stdout and to a file named in the command line. For example, `sort < friends | uniq | tee > friend.list` says for the `sort` program to read the file `friends` in place of `stdin`, pipe its `stdout` to the program `uniq` and `uniq`'s `stdout` to the program `tee` which will display the result AND store it in the file `friends.list`.

Confusing, but imagine there were a program which knew how to shuffle items randomly given a description of them and another which knew how to distribute a specified number of those items to each of a number of named places. Using redirection would allow you to use those two programs to shuffle cards from arbitrary decks and deal hands to an arbitrary number of players for an unspecified card game. You could also use the same programs to shuffle lists of questions and make randomized tests for students.

## Command Arguments & Options

Commands have arguments the user can or must provide. Arguments which are not required are called options. Options to a command may also take arguments. In the early days options were generally terse and inscrutable. Very often they were single letters and the same letter was used for different things by different commands. It can be confusing, but there is help readily available (see below).

While the old syntax is still in use, it is gradually being enhanced by more readable option tags. Old style options are generally preceded by a `'-'` and new style options by `'--'`. Frequently old style options can be combined after a single `'-'` while new style ones cannot. For example;

`ls -lahF` means the same thing as `ls -l --all --human-readable --classify`

One of the earliest options you should learn is the option for getting help about a command. Many commands that require arguments will provide some help if the user simply uses the command without arguments. But that can be dangerous for commands you do not know since some commands can be dangerous with no arguments. Far better is to get help explicitly as shown in the next section.

## Finding Information on Commands

Most commands implement `-help` as an optional argument. Most that do not implement it will realize they do not understand `-help` and display the help anyway. The help you get this way is terse but can be handy if you know the command and just need a reminder of the syntax. Sometimes the help is long and you may want to pipe the output through a pager<sup>3</sup> to make it easier to read. For example:

`ls --help | less`

---

<sup>3</sup> A pager is a program that display text but does so a page at a time rather than scrolling text out of view before you can read it. Many pagers exist, but “less” is my favorite. It is an improved version of “more”.

Virtually all commands in Linux have a description on the system called a man[ual] page. These are accessible in Ubuntu/Kubuntu via Help on the System Menu (of the Gnome desktop) and on the K Menu (of KDE). The organization requires you to find the section desired so it is frequently easier to open a terminal and get the information via the command line.

From the command line you can request the specific man page for a command you need information about or a list of man pages relating to a key word relevant to your need.

```
man <commandname>    # e.g., man iwconfig yields the man page for iwconfig
man -k <keyword>      # e.g., man -k wireless would yield a list of all man
                      pages relating to wireless
apropos <keyword>    # e.g., apropos wireless is the same
```

Probably the first thing to do is look at the man page for man (command is `man man`) to learn about man pages and how to read them.

A few years back some people pushed for a more usable replacement for man pages called info pages but it never really came to fruition. Most man pages end with an advisory to look at the info page but the info page is most often simply a reproduction of the man page. The web-like pages with indexes and crosslinks under the Help menu is much better than the info project and probably an end result of the info project.

## Some Common Commands

This presentation was originally intended to include usage for many commands. It became clear as the paper developed that would be less important than some of the concepts I have explained and less than some I have not. However, I have presented the means to learn about a command so here is a list of commands to look up and try out:

clear, less, more, pager, grep, sort, tee, uniq, whereis, whence, which, locate, updatedb, lspci, lsmod, lshw, lsusb, lsmod, lsof, fdisk, hdparm, ifconfig, iwconfig, route, sudo, gksudo, sudo -i, su, cron, at, anacron, dmesg, logger, tail, head.

Don't worry if you do not understand some of them. That comes with time and experience. They are also just the tip of the iceberg and you will learn what you need as time goes on. You have the tools and knowledge to explore what else is available so go and play.

## Other Subjects for Learning

Environment variables, scripts, paths, file system directories, escaping characters, wild carding, regular expressions, consoles, logs, /proc directory.

## **Finding Information More Generally**

General information about Linux and how to do things in Linux is probably best found using the GUI based help files provided with Ubuntu/Kubuntu or on the web. HOW-TO articles are common on the web and can be extremely useful. Information about your particular hardware and software is harder to come by. Modern GUI tools for this purpose are still evolving and generally less flexible than using the command line. A lot of system information is not yet available in the GUI tools.