

Command Line Interface the basics.

Nigel Gunton 05/06/07

Grouping	Individual
Prerequisites	A *nix account
Courses	Computer Networks, Sys. Admin, any, none
Requirements	Dogged persistence and rugged determination
Summary	An introduction to the key features of the *nix command line or shell
Objectives	To enable a student to tackle the basic use of the command line interface and to have an understanding of more complex use, leading to shell scripting.

1. Background

It is useful, perhaps even essential, in these days of mouse waving, WIMPs and GUIs to have an understanding of the command line interface in unix-like systems. In fact many windows systems administrators install toolkits that give them a similar degree of power and flexibility.

Microsoft Windows PowerShell command line shell and scripting language helps IT professionals achieve greater control and productivity. Using a new admin-focused scripting language, more than 130 standard command line tools, and consistent syntax and utilities, Windows PowerShell allows IT professionals to more easily control system administration and accelerate automation.

<http://www.microsoft.com/>

If Bill says it's a good thing then who are we to argue?. Seriously though, it's a steep (unending) learning curve that will save blood, sweat, tears, time, money and much more in the long run. It'll also have you cursing, tearing your hair out, ranting etc. etc. Eventually you might achieve mastery of the command line. Equally it is actually quite good fun in an innocent kind of way.

You will need

- a basic grasp of the file system standard;
- knowledge of some key command line tools and utilities;
- a good text editor that doesn't rely on a GUI and the ability to use it;
- the ability to read man pages, or otherwise find things out;
- a basic understanding of programming concepts:
 - selection eg `if condition ;`
 - iteration eg `for, while`
 - subroutines or functions
- and probably some other stuff as well.

1.1. The File System Standard

All *nix systems have their directories (folders) and files laid out and named in a similar fashion, typically across multiple hard drives or partitions and often across several machines. Where the files and directories are physically located is invisible to the ordinary user. The users view of the file system is of a single directory tree with the top of the tree, the root, known as / (pronounced slash) and everything dangling from this point.

Some definitions:

path The location of all files and directories is described by means of a 'path'. The path can be 'absolute' or 'relative'

absolute describing the location of a file or directory relative to / eg /usr/X11R6/lib/X11/app-defaults/Xterm identifiable by ALWAYS having a leading /

relative describing the location of a file or directory relative to the current directory. eg Usb_stick/Vhdl/Work/Worksheets/lattice_setup.ms identifiable by NOT having a leading /

./ Shorthand for the current directory so ./Usb_stick/Vhdl/Work/Worksheets/lattice_setup.ms is a relative path.

../ Shorthand for the parent of the current directory, so if our current directory is /home/ngunton/Usb_stick/Vhdl/Work then ../ is a relative path and equivalent to the absolute path/home/ngunton/Usb_stick/Vhdl;

Current Directory The *nix command line interface, CLI, has the concept of the current, or working, directory. The location within the file system from which you are currently working.

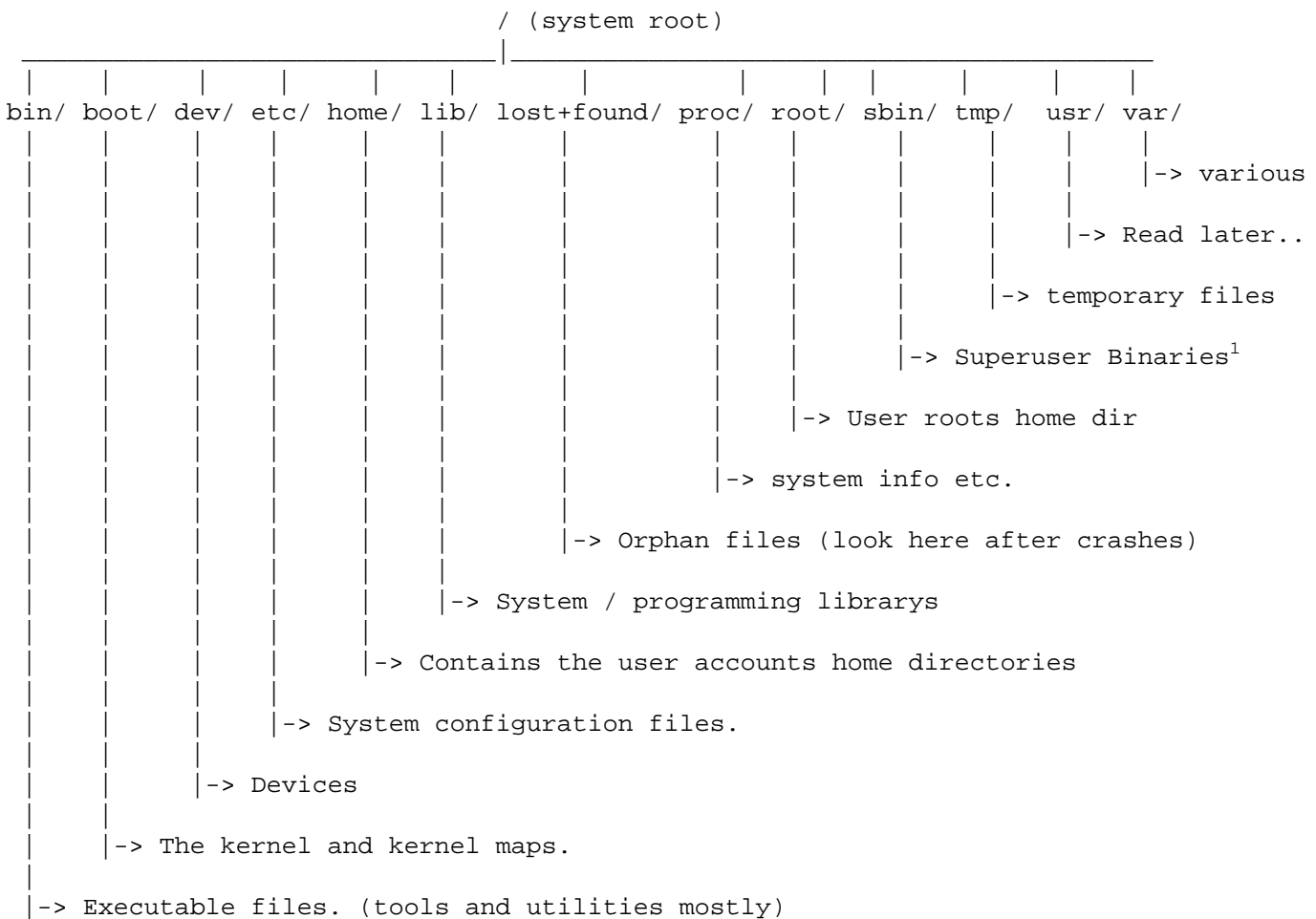


diagram modified from 'Aliens Bash Tutorial'

figure 1: Typical file system layout, first level only

¹ Used to stand for static binaries, ie compiled with their libraries rather than needing dynamic linking

Exercise 1: The filesystem

Key commands for moving around, listing and manipulating files and directories:
Most (all?) have some (many?) options; RTFM².

<code>pwd</code>	print working directory, display your current location.
<code>cd [dir]</code>	change directory to <i>dir</i> , if none given, default is your home directory.
<code>cp foo bar</code>	copy file <i>foo</i> to file <i>bar</i>
<code>mv foo bar</code>	move (rename) file <i>foo</i> to file <i>bar</i>
<code>rm foobar</code>	remove (delete) file <i>foobar</i> (there is NO undelete)
<code>ls [options][dir]</code>	list directory contents, default is current directory, short listing.
<code>ls -l</code>	long listing, gives size, owner, date stamp etc.
<code>ls -a</code>	all files including 'hidden' files.
<code>rmdir foobar</code>	remove (delete) directory <i>foobar</i> (only if empty)
<code>mkdir foobar</code>	create directory (folder) <i>foobar</i>
<code>touch foobar</code>	create empty file <i>foobar</i> (suprisingly useful)
<code>chmod [options]dir_or_file</code>	change mode of directory or file,ie change permissions (access rights).
<code>chmod a=r foo</code>	Make file or directory <i>foo</i> read only for everybody.
<code>chmod 444 foo</code>	Cryptic way of doing the same thing.

- i) What directory are you currently in?
- ii) Create a directory based on your name or other unique identifier³.
- iii) Change directory to the new one that you have created.
- iv) What happens if you use spaces in the directory name? Are spaces a good thing?
- v) Make another directory in your current one, change into it and then create some files using `touch`.
- vi) What restrictions are there on the characters that can be used for file and directory names
- vii) How can you tell a file from a directory on the command line?
- viii) Experiment with `chmod` and change the permissions on some files and directories.
- ix) Move around the directory tree using `cd`, use `pwd` to confirm your location. How deep can you go? Is anywhere forbidden? If so, why? (hint `ls -l`)

Exercise 2: File manipulation

<code>cat foo</code>	dump file <i>foo</i> to the screen
<code>more bar</code>	scroll through file <i>bar</i>
	<code>q</code> = quit, spacebar = page down, <code>b</code> = page back
<code>less bar</code>	less is more, only better
<code>file foobar</code>	reads the <i>magic number</i> (identifies the file type)
<code>head [-In]snafu</code>	show the first 10 [-n] lines in a file
<code>tail [-In]snafu</code>	show the last 10 [-n] lines in a file
<code>grep pattern file</code>	look for <i>pattern</i> in <i>file</i>

- i) Play with these commands, try using `file` on various files in `/bin` /etc and so on.

² **Note:** Some of the commands do more than is stated above, if in doubt the **Read The Fine Manual** pages, eg `man touch`

³ Execute the commands `ls`, `ls -l` and `ls -a` after creating/deleting files and directories so that you become familiar with the results

Other important stuff

<code>ls * >foo.txt</code>	redirect output of <code>ls</code> into file <code>foo.txt</code> Create <code>foo.txt</code> if it doesn't exist, OVERWRITE if it does.
<code>ls * >>foo.txt</code>	append output of <code>ls</code> to file <code>foo.txt</code> . ie add output to the end of existing file, otherwise create file.
<code>emacs bar.txt &</code>	Start application in background, keeps the prompt active.
<code>./myscript.sh</code>	Execute the script <code>myscript.sh</code> which is in the current directory. You are giving a relative path to the executable as it is not found in <code>\$PATH</code>
<code>ls * more</code>	using the 'pipe' symbol. Pipes output of <code>ls</code> into <code>more</code>

Stuff that is very useful

RTFM for these

<code>cut -c 2-5bar</code>	cut character positions 2 through 5 from file, send to standard out
<code>sortfoo.txt</code>	sorts file contents, default is ASCII order
<code>splitbar</code>	split input into fixed size lumps
<code>sed</code>	Stream editor, needs a worksheet on its own.
<code>find</code>	does what it says, see seperate worksheet
<code>/dev/null</code>	bottomless pit or black hole used as destination for unwanted output.
<code>printenv</code>	dump all the environment variables to screen