We all love bash

Dulcie Vousden
Stephan Sanders

August 21 2017

Contents

1 What is UNIX?

UNIX is an operating system, i.e. a framework of tools that makes computers usable. You probably use UNIX already. It forms the foundation of both Android (phones and tablets) and the Mac OS (Apple computers) and, to some extent, iOS (iPhones and iPads). These are all examples of graphical operating systems with UNIX underlying them. For example you press a button to open a file which sends an Open command to UNIX, rather than you writing the Open command directly. For 'serious' computing it remains more efficient to write these codes directly rather than using the graphical interface since it is easier to replicate text than mouse clicks. What we are really teaching is how to interact with a UNIX-based computer more directly using a command line interface (i.e. text), rather than the graphical interface you are probably used to.

2 What is a terminal?

This section is from http://ryanstutorials.net/linuxtutorial.

UNIX (Linux) has a graphical user interface and it works pretty much like the GUI's on other systems that you are familiar with such as Windows and OSX. This tutorial won't focus on these as I reckon you can probably figure

that part out by yourself. This tutorial will focus instead on the command line (also known as a terminal) running Bash.

A command line, or terminal, is a text based interface to the system. You are able to enter commands by typing them on the keyboard and feedback will be given to you similarly as text.

The command line is an interesting beast, and if you've not used one before, can be a bit daunting. Don't worry, with a bit of practice you'll soon come to see it as your friend. Don't think of it as leaving the GUI behind so much as adding to it. While you can leave the GUI all together, most people open up a command line interface just as another window on their desktop (in fact you can have as many open as you like). This is also to our advantage as we can have several command lines open and doing different tasks in each at the same time. We can also easily jump back to the GUI when it suits us. Experiment until you find the setup that suits you best. As an example I will typically have 3 terminals open: 1 in which I do my working, another to bring up ancilliary data and a final one for viewing Manual pages (more on these later).

A question that may have crossed your mind is "Why should I bother learning the command line? The Graphical User Interface is much easier and I can already do most of what I need there." To a certain extent you would be right, and by no means am I suggesting you should ditch the GUI. Some tasks are best suited to a GUI, word processing and video editing are great examples. At the same time, some tasks are more suited to the command line, data manipulation (reporting) and file management are some good examples. Some tasks will be just as easy in either environment. Think of the command line as another tool you can add to your belt. As always, pick the best tool for the job.

3 I wish that someone had told me...

Learning to use the command line interface can be both inspiring and frustrating. The following hints, tips, and concepts will help you to learn these skills quicker:

1. The computer does EXACTLY what you tell it. This is its greatest strength, as it will faithfully reproduce a result or apply a process equally across all data. It is also the computer's greatest weakness - if you tell it to delete all your work it will do so without a second's hesitation

2. The computer is not intelligent. If you ask it to go to a directory (aka folder) named myHomeDirectory, but misspell it as myHomeDirectory, it will simply tell you that it can't find the directory without volunteering that a similarly named directory is right there.

3. Location matters, both of yourself and the files. Along with what files, directory, or programs you want to run, you need to tell the computer where they are relative to yourself. You can find where you are using the command 'pwd'.

4. The files and directories are arranged like a tree so it is easier to move up and down the branches rather than sideways between the branches.

5. Keep a copy of successful commands (and sometimes unsuccessful ones). The next time you try to do the same task you can just copy and paste.

6. Almost all UNIX commands follow the same pattern: command/program options files

4 UNIX pointers

1. If a command fails assume there is a typo before trying anything else.

2. If that does not work, ask yourself, 'am I in the right place (pwd), are the files in the right place, are the programs in the right place?'.

3. If that fails, ask yourself, have I told the computer everything it needs to know? For example does it know the program to use, the input files, and the output files?

4. Spaces and punctuation (except for period, underscore, dash) in file or directory names only lead to misery and hurt. Avoid them.

5. The most useful keys are tab, up, and down.

6. Anything is possible, but some things are easier than others.

7. After you run a command try to confirm that it has worked. For example, is there an output file? Is there anything inthe output file? Do the contents of the output file look as you would expect?

8. It pays to be systematic. Think about ways to name and organize files that will make sense in two years time.

5 Navigating and moving files in UNIX

Telling the computer where programs and files are located relative to where you are (pwd) is a fundamental skill in UNIX. Editing and moving files is also critical. Table 1 shows a list of basic commands for this navigation and the treasure hunt activity will give you practical experience of using these.

Table 1: Bash command examples

Command	Example	What it does it tell us
Tab	Tab	Autocomplete the name of the program/file/directory

Tab Tab	Tab Tab	Show a list of potential autocomplete options for the program/file/directory
UpPress	the up arrow	Show the last command that you wrote; press again to see the one before it and down arrow to go in reverse
Ctrl-cHold	Ctrl + press c	STOP! Kills the currently running command.
ls	ls	Lists contents of your current working directory
	ls path/to/dir/	Lists contents of path
	ls file*txt	Lists all files that start with 'file' and end with 'txt'. An * is what's called a 'wildcard' character.
	ls -l	Lists contents (long listing format – more information)
	ls -lh	Long listing format, human readable file size
	ls -R	Lists contents recursively (including subdirectories)
	ls -a	List all files, including hidden files that do not show by default. The names of hidden files will start with a .
	ls -t	List files by time/date
	ls -r	Reverse order while sorting (i.e. ls –t will display newest to oldestwhereas ls –tr will display in oldest to newest)
mkdir	mkdir foo	Makes a directory called foo within the current directory
rmdir	rmdir foo	Permanently removes the directory foo; only works if foo is empty
cd	cd path/to/dir/	Change directory; takes you to the directory path/to/dir/
	cd ../	Go back one directory
	cd ../../	Go back two directories
	cd ../../foo	Go back two directories and forward into the directory foo
	cd	Go to home directory
	cd /mnt	Go to mnt directory
mv	mv file1.txt file2.txt	Rename file1.txt to file2.txt
	mv file.txt /path/to/dir/	Move file.txt into path/to/dir/
	mv -i file.txt /path/to/dir/	Move file.txt into path/to/dir/ and prompt before overwriting

	mv file1.txt /path/to/dir/file2.txt	Rename file1.txt to file2.txt in directory path/to/dir
cp	cp file1.txt file2.txt	Copy file1.txt to a second file named file2.txt
cp	file.txt /path/to/dir/	Copy file.txt into path/to/dir/
	cp -i file.txt /path/to/dir/	Copy file.txt into path/to/dir/ and prompt before overwriting
	cp -R dir1 /path/to/dir/	Copy dir1 into /path/to/dir/
rm	rm file.txt	Permanently remove/delete file.txt
	rm -R foo	Permanently remove/delete directory foo and all its contents
	rm -i file.txt	Permanently remove file.txt, with prompting
find	find /mnt -name file.txt	Find all instances of a file called file.txt in /mnt or any directories within /mnt.
.	cp path/to/dir/file.txt .	The period means 'here'. In the example the file is copied to the current directory
pwd	pwd	Shows where you are: present working directory
man	man ls	Displays unix manual on ls (all commands have a man page)
gzip	gzip file.txt	Compresses file.txt to make file.txt.gz
gunzip	gunzip file.txt.gz	Uncompresses file.txt.gz to make file.txt
gunzip -c	gunzip -c file.txt.gz	Uncompresses file.txt.gz to make file.txt but keeps a copy of file.txt.gz
tar	tar xfvz file.tar.gz	'gunzip's and expands the tarball (a collection of files and directories)file.tar.gz
touch	touch file.txt	Creates empty file file.txt if it does not exist
nano	nano file.txt	Opens file.txt in a text editor
vim	vim file.txt	Opens file.txt in another text editor
head	head file.txt	Displays first 10 lines (by default) of file.txt
tail	tail file.txt	Displays last 10 lines (by default) of file.txt
less	less file.txt	View (but not change) file.txt
grep	grep banana file.txt	Searches file.txt for lines containing banana
>	ls> foo.txt	STDOUT: take output and create/overwrite to foo.txt

>>	ls >> foo.txt	STDOUT: take output and append to foo.txt
2 >	ls > foo.txt 2 > foo.err	As above, but also STDERR error messages are written to foo.err
<	perl test.pl < file.txt	STDIN: use file.txt as the input
|	ls | grep .txt > textFiles.txt	Pipes STDOUT from one command (ls) to be the STDIN for thenext (grep)