Working on the command line

Graphical interfaces might look pretty, but the command line gives you the power to get exactly what you want – if you know how!

MOST PEOPLE haven't seen a text computer prompt since typing **win** to launch Windows 3.1 about 15 years ago. In Windows 95, Microsoft made a concerted effort to hide DOS behind splash screens and animations, and the Windows NT line of operating systems (including XP and Vista) relegate DOS to a mere terminal window that only a handful dare touch.

Not so in Linux. In fact, a good way of thinking about Linux is that it's what Windows NT would have been if Microsoft had continued basing it on DOS. In Linux, the command line is king, with only the thin veneer of a graphical user interface stretched over it to make newbies feel welcome. But you're not a newbie, are you? You turned to these pages because you have a thirst for unlimited power – it's time to turn to the dark side of computing!

GET HELP!

If you want to know more about any command, read its manual page using the *man* command. For example, to see all the options for *ls*, run man ls. Manual pages are often called man pages because that's the name of the program that reads them.

```
[paul@localhost ~]$ ls /bin
arch*           domainname@        hostname*  nisdomainname@  stty*
awk@            echo*              id*        open*           su*
basename*       egrep*             igawk*     ping*           sync*
bash*           env*               ipcalc*    procps3-kill*   tar*
bash3@          ex@                ipv6calc*  progress*       touch*
bluepin@        expr*              kill*      ps*             true*
cat*            false*             link*      pwd*            umount*
chgrp*          fbmngplay.static*  ln*        rbash@          uname*
chmod*          fbresolution*      login*     rm*             unicode_star
chown*          fbtruetype.static* ls*        rmdir*          unlink*
cp*             fgrep*             mail*      rpm*            usleep*
cpio*           find*              mkdir*     rvi@            vi@
cut*            gawk*              mknod*     rview@          view@
date*           gawk-3.1.4*        mktemp*    rvim@           vim@
dd*             gettext*           more*      sed*            vim-minimal*
df*             grep*              mount*     sh@             ypdomainname
dmesg*          gtar@              mv*        sleep*          zcat*
dnsdomainname@  gunzip*            netstat*   sort*
doexec*         gzip*              nice*      stat*
[paul@localhost ~]$ uptime
```

Using a graphical console gives you more flexibility than traditional command-line interpreters with screen elements such as font sizes and background pictures.

Pressing Ctrl+Alt+F1 will switch you to terminal 1, which is all raw text. Hit Ctrl+F7 to get back to the safety of the GUI!

"There are a number of reasons to use the command line: to get complete control of your system, fix your graphical interface... or impress your friends."

If you've managed to find your feet in the world of Linux graphical user interfaces (GUIs), you might wonder why you should bother changing to boring old text input. There are a number of reasons:

- You want to have complete control of your system.
- You want to combine commands to get specific output.
- You have broken your graphical interface and need to fix it.
- You want to impress your friends.

The command line is good for all of those and more. However, before you begin on the path to enlightenment, it is important that you understand that the terms console, command line, command prompt, shell and terminal are all virtually interchangeable. Sometimes people use 'CLI', standing for command-line interpreter, and this can also be lumped in with the group of synonyms.

FIRST STEPS

There are two ways to get to the command line in Linux. As we said earlier, the Linux graphical user interface is only a thin layer on top of the command line, and you can switch out of the GUI into the command line with Ctrl+Alt+F1. This switches to terminal 1, but Linux has five other text-based terminals available on F2 to F6. The graphical 'terminal' that is your GUI is on F7, and you can press Ctrl+F7 to return to it from terminal 1.

Note that to switch terminals from the GUI you need to use Ctrl+Alt+F key, but to switch terminals from another terminal you need only use Ctrl+F key. This is because the Control key by itself is mapped to other shortcuts inside *X* (the graphic display manager), so Alt is added to clear things up. When you switch away from the GUI, everything is left running, untouched and as you left it.

The other way to get to the command line is to open a terminal emulator window inside your GUI. This looks and works just like a normal terminal, but is usually more flexible because it's embedded inside a GUI display – you can hide it, move it around and open multiple terminals at a time.

The standard terminal program in Gnome is called *Terminal*, and the KDE equivalent is *Konsole*. Most distros install one of them by default, but failing that you will almost certainly have the ancient *X* terminal, known as *xterm*.

If you open a terminal window inside *X*, it will use your current user account login and probably start you in your home directory. If you use Ctrl+Alt+F key to switch to a full terminal, you will be asked to log in. You can log in as the same account many times – once in *X*, and once each in every terminal from F1 to F6 if you wanted.

In your terminal window you will see a short line of text and symbols known as the prompt. It should look something like: **[paul@linux ~]$**. This tells you several things:

- you are logged in as the user 'paul.'
- you are accessing a machine called 'linux.'
- ~ means you are in your home directory.
- **$** means you do not have root privileges.

Next to that will be a cursor where you can type your commands, then hit Enter to send them to Linux. For example, type **whoami** and press Enter to have Linux print out your username, then try **date** to have it print out the current time and date. You can use the Up and Down cursor keys to select previous commands.

From here, many distros let you switch to the root user by running the **su** command, but Ubuntu doesn't have any concept of a root user. Instead, you should run administration commands by using **sudo your_command**. This prompts you for the root password, and if you enter it correctly your command will be executed with root privileges. *Sudo* can be dangerous, so be careful what you do.

BROWSE YOUR FILESYSTEM

Most terminals place you in your home directory to start with, usually referred to as ~ (a tilde, usually typed by pressing Shift+#). The *cd* (change directory) command will navigate you to other

ADVANCED COMMAND LINE TIPS

The real power of the command line comes when you start to combine commands together to create smarter results. For example, the *ps aux* command lists all the processes that are running on your machine, but what if you only want to see *Vim* processes? The solution is to send the output of ps aux through another command, called *grep*, which filters out text and only prints lines that match your criteria. Sending data from program to program is known as piping, and you need to use the pipe symbol, |, to do it. This is usually on the keyboard as Shift+\, but it may be different for you.

So, to list all processes on the machine that contain the word 'vim', you would use the command ps aux | grep vim. That runs ps aux, which returns all programs being run, one per line. Those lines are then sent to *grep* along with the parameter vim, which tells *grep* to only print lines that match the word 'vim'. You can see how this command acts like a filter on the output of *ps*.

We can take this a step further by piping the output of grep. For example, what if you only want to know the number of *Vim* processes being run? To solve this problem, we need to pipe the output of *ps aux* to *grep vim*, then pipe the output of that to the *wc* command. This new command stands for 'word count', and reports the number of lines, words and letters in its input. We know that *ps* outputs processes one per line, so to find out how many *Vim* processes are running we need to count the lines that match our *grep* filter.

To get that output, we need to pass the -l (lowercase L) parameter to *wc*, giving this command: ps aux | grep vim | wc -l. That accomplishes our goal – or does it?

Try running two instances of *Vim*, and running our command. What output do you get? Is it three instances, or two? The result is likely to vary between the two possibilities, because grep vim is a command in its own right, and as it matches the string 'vim' it will be returned. The reason the result varies is down to whether *grep* is quick enough to catch itself!

We can solve this problem by simply running the result from grep through another *grep* filter, this time only allowing lines to pass if they don't have the word 'grep' in.

This is done by specifying 'grep' as the search parameter, but also using -v to tell *grep* to invert the search. Inverting the search means it will allow things through only if they *don't* match the criteria. This expands the final command to: ps aux | grep "vim" | grep -v "grep" | wc -l. Long, but exactly what we were after.

directories. For example, to switch to the root directory, type **cd /**, to switch to the home directory type **cd /home** and to switch to the *X.org* configuration directory use **cd /etc/X11**. You can change back to your home directory at any time by typing **cd** by itself.

The **/**, **/home**, and **/etc/X11** parts of the command are called parameters – **cd** is the command, and you pass to it as a parameter the name of the directory you want to change to.

Listing the contents of a directory is done with the *ls* (that's a lowercase L) command. By default this lists all the files and folders in the directory, but you can also use **ls -l** to list more information, and **ls -a** to show hidden files. Hidden files are called dotfiles in Linux, as they begin with a full stop, and there are two special directories known as '**.**' and '**..**' in every other directory except the root directory.

The '**.**' directory is equivalent to the current directory. So when you run the command **./configure**, for instance, it means, 'Run the configure program situated in the current directory, as opposed to any other configure command'. The '**..**' directory means 'parent directory', so you can type **cd ..** to change from **/home** to **/**.

The *mkdir* command creates new directories. For example, **mkdir foo** in your home directory would create a directory **/home/paul/foo** (or whatever your username is). To delete a directory, use the **rm -rf** command, which stands for 'remove file/directory, recursively, and force'. This command will delete a directory, as well as all the files and directories it contains (recursively) without asking any further questions. This is why you should not run as root: a command like **sudo rm -rf /** will wipe your entire hard drive.

To read the contents of a file, employ the *less* command. This opens a file and allows you to scroll around inside it using your cursor keys. Type **less /etc/passwd**, and you'll open your system password file for viewing. Press **q** to quit *less* and go back to your terminal, or press **v** to open *Vim* to edit files, which leads us nicely on to the next section!

"The rm –rf command will delete a directory and all the files within it without asking any further questions. This is why you should not run as root!"

EDITING FILES

Being able to open and edit files from the command line is a core skill that you should try to master, simply because sooner or later you'll need it anyway so you might as well learn when you're relaxed.

To launch the *Vim* text editor, just type **vi** at the command line. You can specify the file you want to open by using **vi /path/to/your/file**. When *Vim* loads, use the cursor keys to move the cursor around as you want. The Delete key deletes one character from under the cursor, but you can delete whole words by pressing **dw** (an abbreviation of 'delete word'), or delete whole lines by pressing **dd**. You can add numbers before these abbreviations to have them done several times, so **5dw** deletes five words from under the cursor.

When you're ready to add text, press **i** to enter Insert mode. Here you can type as you need to, but you'll find that the abbreviations no longer work. To exit Insert mode, hit the Escape button and you'll be back viewing as before. You can send commands to *Vim* by typing a colon then your command, so to save your changes to a file, type **:w** (short for 'write') and hit Enter. To quit *Vim*, type **:q**, or if you want to save then quit, use **:wq**.

Although the commands for writing and quitting have been given easy to remember names, not all commands are so obvious. Searching a file for text, for instance, uses the command **/**. So, to search for the line 'paul' in your current file you would type **/paul**. As you type, *Vim* will search for the first instance of the text and show it on the screen. When you hit Enter, *Vim* will highlight all matches in orange, and you can cycle through them by typing **/** and pressing Enter again. Or, to go backwards, use **?** followed by your search term, such as **?paul**.

Copying text to the clipboard is called yanking, and activated by pressing **yy**. You can yank more than one line of text by using the normal number prefix, such as **5yy** to yank five lines of text or **100yy** to yank 100 lines – the current line is included in that count. To paste the text into your document, hit **p**, or again use a number prefix to paste a number of times, such as **8p** to paste the clipboard eight times.

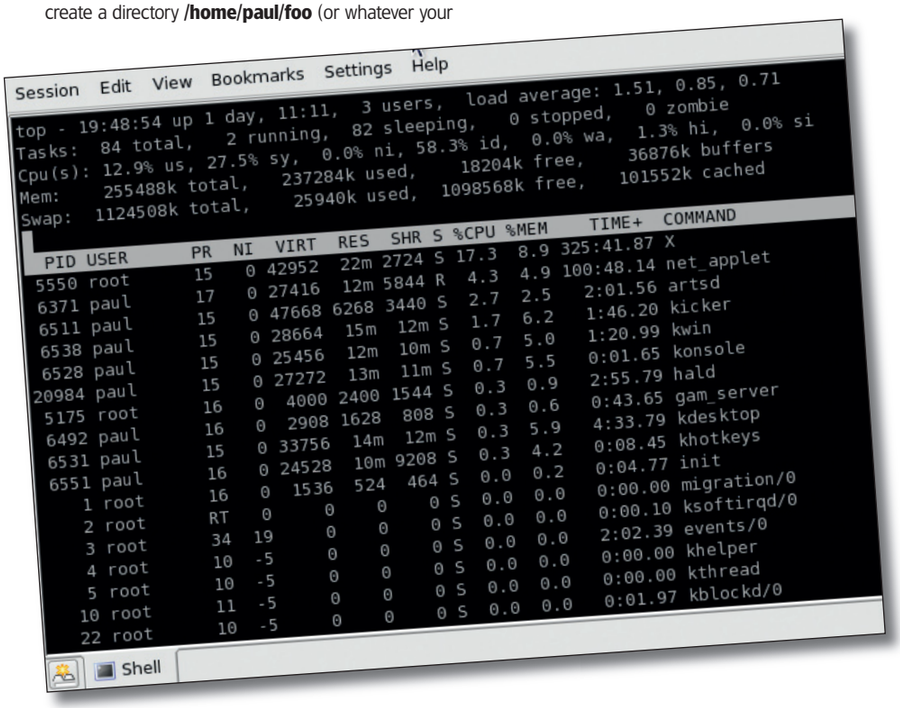
SYSTEM MAINTENANCE

The command line can help you administer a system, which is great for taking remote control of a server. For example, you can view all the programs being run, see how much CPU time they are using and kill off any that are rogue.

The basic command here is *ps*, which prints a list of all the processes being run on your machine. A process is the technical term for a program that is running – if you run *Vim* twice, it will appear as two separate processes. If you type **ps** with no parameters, you don't get very useful information, but if you run it with **ps aux**, you get a full list of all programs being run on the machine. It's pretty rare that you'll want all that information, though; a much better solution is the *top* command, which lists the top 20 resource-hungry programs on your machine, along with how much CPU time they are using.

The *top* command runs interactively, which means that it has a little interface of its own and accepts keypresses to modify the display. To get started, just type **top** and hit Enter. You'll see something similar to the screen at left, which shows *top* in action.

The processes are sorted by CPU usage, ordered most intensive first. The PID column shows the process ID for each process, which is a unique identifier you can use to reference a process, and the %CPU column shows current CPU usage for each process.



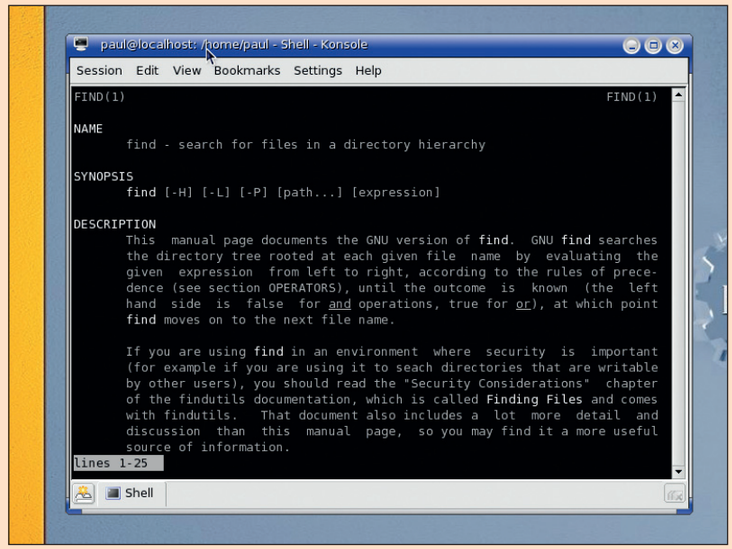
Find out which applications are the CPU hogs at a glance through *top*, then hit k to kill them.

HOWTO... FIND FILES

There are two ways to find files on the command line, and both are aptly named: the *find* command, and the *locate* command. The first starts wherever you tell it, and searches through each directory until it finds the file you're after. The second looks inside a filesystem index for matching names. Using *locate* is perfect if your cache is up to date. But if you have recently downloaded a new file and try to find it, it won't be in the cache yet, because Linux only updates the *locate* cache once a day. On the flipside, if the file *is* in *locate*'s cache, it will be found much faster.

As it's easier than *find*, we'll try *locate* first. Switch to root using su, then run the command updatedb. This could take a few minutes to run, but it essentially forces Linux to recreate the *locate* file index for quick searching, rather than wait for the nightly rebuild. When it has finished, type exit to become a normal user again, and run the command locate lib to find all files that have the word 'lib' somewhere in their name or path.

Using *find* is much harder, as you need to be very precise with its parameters. However, the advantage is that you can search for much more than just the name of a file – look for modification times, sizes, permissions and more. The basic usage of *find* is this: find / -name myfile.txt. This tells *find* to start in the root directory, and look for all files that match the name 'myfile.txt', searching recursively through all subdirectories. If you want to find files that are of a specific size, you can add the -size parameter along with the size you're interested in. So, find / -name myfile.txt -size -5k will find all files called myfile.txt that are less than 5 kilobytes in size. If you specify -size 5k then *find* will return files only if they are exactly 5kb, but you can also use +5k to find files larger than 5kb.



The *find* command is very powerful: it's worth taking the time to learn it. See the manual page for all the features that it supports.

With this information, we're going to create an out-of-control process, watch it appear in *top*, then kill it. The power!

To get started, bring up a fresh terminal window. If you used Ctrl+Alt+F1 to get to your current terminal, hit Ctrl+Alt+F2. If you used *Gnome Terminal* or *Konsole*, just open another one. Now, in this terminal you should run the *yes* command, which does nothing other than output line after line of letter Ys. This command seems useless at first, but it was designed to send input to programs such as the ones that ask you, 'Are you sure you want to delete XYZ?'.

The *yes* command will keep on running until it's interrupted, so if you switch back to your other terminal you'll see *yes* high up your priority list in *top*, using up a great deal of CPU time. You'll also see *Gnome Terminal* (or *Konsole*) and *X* using up a lot of CPU time, because they need to do a lot of work to keep displaying all of *yes*'s output.

Note the PID next to the *yes* process: we need to try to shut that one down. To stop a process – that's what we mean by killing it – you need to press **k** in your *top* window. This asks you for the PID you want to kill, and you should enter the PID of the out-of-control *yes* process here, then hit Enter. It will then ask you which signal you want to use, which enables you to choose how you want to kill *yes*. It's all rather polite. The default is 15, which is a kind way to kill a process: Linux tells the process it is going to be stopped, and gives it time to finish what it is doing and clean up after itself.

However, if that fails you can enter the kill number 9 – equivalent to Force Quit – where the process is halted immediately with no further questions or waiting.

So far we've only been looking at simple applications, but Linux is capable of communicating with the outside world entirely through the command line. You can upload and download files through FTP, connect to and control other Linux machines using SSH (the secure shell), and even browse the web using *Lynx*. That last one is the easiest, so we'll start there!

THE OUTSIDE WORLD

To browse the web through the command line, type **lynx www.google.com**. If your distro doesn't have the *Lynx* browser installed, you may have *w3m* (an equivalent). If not, check your package manager to try to install them.

With *Lynx*, you can use the cursor keys to move around the screen and type into text boxes. For example, on the Google site use the Down cursor key to scroll through to the search box, then type in something to search for and hit Enter twice. To go to other websites, either follow links (select the one you want, then press Enter) or press G and type in a new URL. To quit, press Q then Y to confirm. You can use *Lynx* (or whichever browser you choose in the end) to download files to your computer, simply by following the link in the browser.

To connect to FTP servers, use the *ftp* command and pass it the name of the server to connect to. For example, **ftp ftp.gnu.org** will connect to the GNU FTP server. You will be prompted for a username (enter 'anonymous', without the quotes), then your email address for the password. This will log you in as an anonymous user, able to download files but not upload them. FTP servers use standard Linux commands, so you can use *ls* to list the contents of directories and *cd* to change directories. If you see a file you want, use **get <your_filename>** to download it. Note that if you are trying to download a file that does not contain text (such as a zip file or an executable) you should enable binary data transfers by typing **binary** and hitting Enter. You can also enable a progress bar for long file downloads by typing **hash**.

Finally, to connect to other Linux and Unix servers, you can use SSH. This enables you to log in to a remote machine as if it were local, transmitting all the data over an encrypted connection so that no one can monitor what you are doing. To use the SSH client, you need to tell it the name of the server and the username it should log in as. The command **ssh myserver.com -l paul** (that's a lowercase L) would log in to that server as Paul.

Type **exit** to close your SSH connection when you're done.