

# JEDI MASTER THE TERMINAL



Journey, you must, into the often feared land of the shell with **Mayank Sharma** and discover the firepower of a fully armed and operational command line!

**D**espite the leaps and bounds that Linux desktops environments have made over the years, there are still moments when accomplishing a task requires you to fire up a terminal. Since none of the other mainstream operating systems rely on the Command Line Interface (CLI) in the same way as Linux, it's often quite intimidating for new users. Fear of the CLI is simply not acceptable to anyone at LXF towers.

The command line is a powerful portal to some amazing tools that makes it possible to accomplish tasks much more quickly than you could with graphical tools. Whether you want to install software,

configure your network or tweak other system settings, you can do it from the terminal. In fact, the graphical front-end applications use these command-line tools to get things done. Still, as you might have noticed, we often ask you to switch

**“while the graphical tools have made the CLI optional, most only expose a small subset.”**

to the command line to complete a task. That's because that while the graphical tools have made the CLI optional, most only expose a small subset of the features offered by their command-line equivalents. Since there's no escaping the Linux CLI,

it's a good idea to embrace this mature and powerful interface and get familiar with its ways.

We've compiled a set of commands that you should be familiar with to work more efficiently with your Linux install.

There's obviously a learning curve to using the terminal to compose your own commands and over the next few pages we'll handhold you through the CLI and help you realise the true potential of

this interface. Once you get comfortable, you'll notice its speed and efficiency for common admin tasks. The skills you learn over the next few pages will also elevate your Linux competence and help you join the ranks of experienced campaigners.

# Easing into a CLI

How to break terminal velocity and not your leg.

A terminal emulator is one of the core applications that's bundled with every Linux desktop environment. Gnome has the *gnome-terminal* while KDE uses *konsole* though they are usually simply labelled as 'Terminal' in the application menu. When you fire up a terminal you get a prompt like `bodhi@epoch: ~ $`. While it may vary in appearance somewhat depending on the distribution (distro), it's usually made up of your username@machinename, followed by the current working directory and a dollar sign. Note that the last character changes from `$` to `#` when the terminal session has superuser privileges.

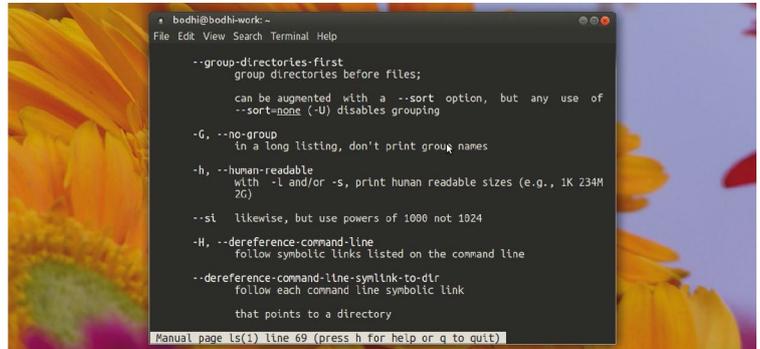
Let's begin our terminal travels by learning to navigate the Linux file system. Files under Linux are organised in a hierarchical directory structure that you can think of as an inverted tree-like pattern. The top most directory is called the **root** directory which contains files and subdirectories that further contain more files and subdirectories and so on. In the shell you are always inside a directory. To display the current working directory, we use the `pwd` (print working directory) command, such as:

```
bodhi@epoch: ~ $ pwd
/home/bodhi
```

By default the terminal session sets the current working directory to your **home** directory. As you are probably aware, every user who has an account on your distribution gets their own **home** directory. You can list the contents of a directory with the `ls` command which can also be used to display various attributes about the files and folders, such as:

```
bodhi@epoch: ~ $ ls -l
total 196
-rw-rw-r-- 1 bodhi bodhi 5701 Feb  4 18:20 example-text-file.txt
-rw-rw-r-- 1 bodhi bodhi 16405 Dec 13 08:45 chart.ods
-rw-rw-r-- 1 bodhi bodhi 23339 Dec 13 08:47 an-ebook.pdf
drwxr-xr-x 3 bodhi bodhi 4096 Feb  4 12:32 Desktop
drwxr-xr-x 10 bodhi bodhi 20480 Feb  6 15:17 Documents
....
```

We'll equip you with the skills to make sense of this output over the next few pages. The Linux shell also keeps a log of all the commands you've executed on the shell. You can press the Up arrow key to bring up the previous command. This list of commands is kept in a hidden file named `.bash_history` under your **home** directory. To navigate the filesystem you



› You can add options to most commands. Some consist of single characters preceded by a dash, (e.g. `-l`), but many are longer (e.g. `--human-readable`).

can use the `cd` command to change the current working directory. By entering `cd` without any attributes, this will return you to your **home** directory. To change to another directory you'll have to append its pathname as an attribute. A pathname can be understood as the route taken along the filesystem tree to get to a desired directory.

## Move around

There are two distinct mechanisms for specifying pathnames: absolute pathnames and relative pathnames. An absolute pathname begins with the **root** directory and follows the tree one from branch to branch until it reaches the desired location. For example, the pathname to get to the directory that apps and various utilities use to house their logs is `/var/log`. The leading `/` represents the **root** directory under which there is a directory named **var** that further contains the **log** directory. Type `cd /var/log` to move into that directory. Notice the change in the shell prompt that now displays the pathname of the current working directory.

In contrast to absolute pathnames, a relative pathname traces its steps to the destination directory with respect to the current working directory. To this end it uses a couple of special notations, a single dot (`.`) and two dots (`..`). The single dot notation represents the current directory while the double dot represents its parent, e.g. if you wish to move to the `/var` directory from the `/var/log` directory, you can type `cd ..` instead of the absolute pathname which is `cd /var`.

**Top Tip:** Shift+PgUp/PgDown: Use these key combinations to scroll the command-line interface.

**Top Tip:** Ctrl+R: Use this key combo and type any keyword to search for matching commands in the history.

## The CLI flows through Linux

The Linux command line offers various options to help you be more productive and find your way around the unfamiliar environment.

For instance, many commands and utilities support a `--help` option that displays information about its supported options. For example, the `rm --help` command will print the various options supported by the `rm` command along with their description and brief usage instructions.

Similarly, most command-line utilities also bundle a formal piece of documentation known as the manual page and usually referred to as its man page.

The command `man mkdir` will bring up the man page for the `mkdir` utility. There are also a couple of helpful utilities that can extract brief information from a man page. For example, the `whatis` command displays a very brief description of a command from its manual page

that's usually enough to gauge the purpose of an unfamiliar utility. Then there's `apropos` that looks for a string inside a man page's name and description sections. It comes in handy when you can't remember the right command for a particular task. For example, `apropos "split a zip file"` will point you to the `zipsplit` utility that helps you split a ZIP file into smaller ZIP files. You can then read through its man page to get familiar with its syntax and usage.

# Users and permissions

Crowd control from the blank void of the dark terminal.

Typically most desktop Linux distros only create a single user during installation. But Linux is a multiuser system that can be used by multiple users at the same time (as we'll learn later in the feature). There are a number of command-line tools that help you create and maintain users and groups, such as:

```
$ useradd mayank  
$ passwd mayank
```

The first command creates a new user account called mayank that's locked until you assign a password to it with the second command. When a new user account is created, Linux will also create a **home** directory for the account with the username, such as **/home/mayank**. The user account will have a bunch of hidden files that'll provide the environment variables for the user's session. The command also creates a group with the same name as the new user.

Once you've added an account, you can edit it with the **usermod** command. So for example, the command **usermod --append --groups sudo,vboxusers mayank** adds the user mayank to the comma separated list of groups. If you issue this command without the **--append** option, the user will be added to the specified groups but removed from any other groups it already belongs to. A useful security option is to set an expiry date for an account.

**Top Tip:**  
split: chop a file into pieces, e.g. **split <FILE> --bytes=1000** splits <FILE> into 1,000 byte files.

The command **usermod --expiredate 2017-05-21 mayank** will disable logins into the account post 21 May 2017. In the same vein, you can also delete a user account with the **userdel** command. For example, **userdel --remove mayank** will delete the mayank user along with its **home** directory and all the files residing within.

## Change identities

As you get familiar with the command line utilities your distro has to offer you'll often run into some that require superuser privileges. This is especially true of most commands that help carry out some administrative task. The two main commands that help you take on escalated identities are **su** and **sudo**. Some distros include one of the two commands while others include both.

The **su** command enables you to assume the identity of another user and gives you the option to either initiate a new shell session or only issue a single command as that user. The command **su mayank** will prompt you for the password for the mayank user and after successful authentication change the working environment from the existing user to that of the mayank user. If you don't specify a username with the **su** command, it'll assume you wish to log in as the superuser and prompt you for the password of the root user. Any command you enter now will run with the privileges of the root user. Type **exit** when you wish to terminate the session and return to your own shell.

It is also possible to execute a single command rather than starting a new interactive shell with the **-c** switch. For example, **su -c "ls -l /root/"** will first prompt you for the password of the root user and then run the command between the quotes with superuser privileges.

The **sudo** command, on the other hand, allows an administrator to set up a configuration file called **/etc/sudoers**, and define specific commands that particular users are permitted to execute with superuser privileges. Another important difference is that the use of **sudo** doesn't require access to the password of the root user and you instead authenticate using your own password. These differences make **sudo** the preferred mechanism of the two. So you can list the contents of the root directory with **sudo ls -l /root/** after authenticating with the password for your account.

```
bodhi@bodhi-work:~  
File Edit View Search Terminal Help  
bodhi@bodhi-work:~$ whoami  
bodhi  
bodhi@bodhi-work:~$ who  
bodhi tty7 2017-02-11 19:09 (:0)  
bodhi pts/1 2017-02-11 19:11 (:0)  
linuxala tty3 2017-02-11 19:18  
bodhi tty2 2017-02-11 19:19  
bodhi@bodhi-work:~$ w  
19:19:29 up 12 min, 4 users, load average: 0.13, 0.55, 0.58  
USER TTY FROM LOGIN# IDLE JCPU PCPU WHAT  
bodhi tty7 :0 19:09 11:50 33.00s 0.14s mate-session  
bodhi pts/1 :0 19:11 1.00s 0.07s 0.00s w  
linuxala tty3 19:18 31.00s 0.09s 0.06s -bash  
bodhi tty2 19:19 17.00s 0.12s 0.09s -bash  
bodhi@bodhi-work:~$ last  
bodhi tty2 Sat Feb 11 19:19 still logged in  
linuxala tty3 Sat Feb 11 19:18 still logged in  
bodhi pts/1 :0 Sat Feb 11 19:11 gone - no logout  
bodhi tty7 :0 Sat Feb 11 19:09 gone - no logout  
bodhi tty7 :0 Sat Feb 11 19:08 19:09 (00:01)  
reboot system boot 4.4.0-59-generic Sat Feb 11 19:07 still running  
bodhi tty7 :0 Fri Feb 10 12:23 crash (1+06:43)  
reboot system boot 4.4.0-59-generic Fri Feb 10 12:23 still running  
bodhi tty7 :0 Thu Feb 9 13:43 crash (22:39)  
reboot system boot 4.4.0-59-generic Thu Feb 9 13:42 still running
```

► The commands **whoami**, **who**, **w** and **last** help you track user activity.

## The power of permissions

Each file in Linux has a set of user and group permissions, and you can use **ls -l** to see a file's full set of permissions and attributes. The output begins with the permission flags like this **drwxr-xr-x** that has four components:

```
$ ls -l  
-rwxr-xr-x 1 bodhi bodhi 23339 Dec 13 08:47  
some-document.pdf  
drwxr-xr-x 3 bodhi bodhi 4096 Feb 4 12:32  
Desktop
```

The first character indicates whether the corresponding item is a directory or a file with a directory marked with a **d** character and a regular file marked with a hyphen (**-**). The next nine characters are broken up into sets of three characters each, which indicate permissions for the user, group and everyone else respectively. The **r** denotes read permission, **w** is for write permissions and **x** points to execute permissions. The first entry in the example

reads **rwxr-xr--**, which means the user has read, write and execute permissions, while other members of the group have read and write permissions and everyone else can only read the file.

Sometimes the permissions are represented with a numerical notation. Read permissions weigh in at 4, write is 2 and execute is 1. So the permission **rwxr-xr-x** can be presented as **(4+2+1)(4+0+1)(0+0+1)** or **751**.

# Files and directories

Get a choking force grip on file management from the text-only CLI.

**A**nother CLI skill that will help you manage your distro more effectively is the ability to manipulate the filesystem from the terminal. Before we get to the commands, first know that under Linux, everything (including programs, directories and devices) is a file and every file must be owned by a user. Users can only access what they own or have been given permission to run. Permission is granted because the user either belongs to the file's group or because the file is accessible to all users. The one exception to this rule is the root user who is allowed to access all files and programs in the system.

Many times you'll need to change the permissions of a file to make it accessible or inaccessible. The `chmod` command helps you change the permissions or mode of a file. Be aware that only the file's owner, or the superuser, can change the mode of a file. The command `chmod` supports two distinct ways of specifying mode changes: numerical number representation or symbolic representation. Symbolic notation offers the advantage of allowing you to set a single attribute without disturbing any of the others and is divided into three parts. The first part denotes whom the change will affect and is marked with a combination of the characters u, g, o, and a:

- » **u** is for user and refers to the owner of the file or directory,
- » **g** is for group owner.
- » **o** is short for others.
- » **a** is short for all or a combination of all the characters and is the default value if no character is specified:

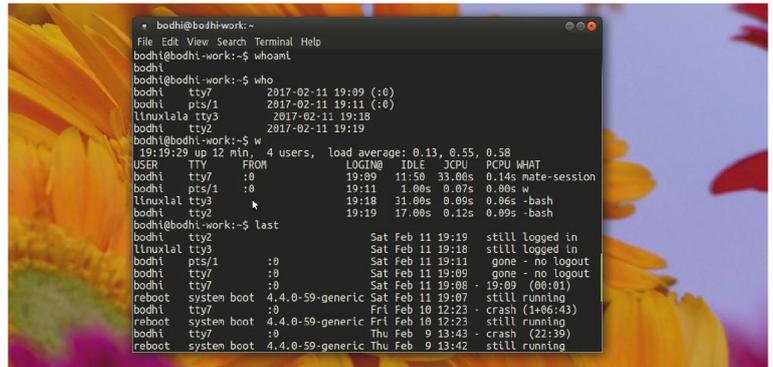
The second part of a symbolic notation marks which operation will be performed:

- » **Plus (+)** Indicates that a permission is to be added.
- » **Minus (-)** Indicates that a permission is to be taken away.
- » **Equals (=)** Indicates that only the specified permissions are to be applied and that all others are to be removed.

The third part denotes what permission will be set and is specified with the earlier mentioned r, w, and x characters.

Putting this into practice a `u+x` adds execute permission for the owner, while `u-x` removes execute permission from the owner. Similarly, `go=rw` sets the group owner and anyone besides the owner to have read and write permission. Any previous execute permissions given to users besides the owner are removed.

While it's easy to perform simple file manipulations with a graphical file manager, complicated tasks can be easier with the command line programs. For example, it'll be a time-



```

bodhi@bodhi-work:~$ whoami
bodhi@bodhi-work:~$ who
bodhi  tty7      2017-02-11 19:09 (:0)
bodhi  pts/1      2017-02-11 19:11 (:0)
linuxlala  tty3      2017-02-11 19:18
bodhi  tty2      2017-02-11 19:19
bodhi@bodhi-work:~$ w
 19:19:29 up 12 min,  4 users,  load average: 0.13, 0.55, 0.58
USER      TTY      FROM          LOGIN@   IDLE   JCPU   PCPU   WHAT
bodhi     tty7     :0            19:09    11:50  33.06s  0.14s  mate-session
bodhi     pts/1    :0            19:11    1.08s  0.07s  0.00s  w
linuxlala tty3     :0            19:18   31.08s  0.09s  0.06s  -bash
bodhi     tty2     :0            19:19   17.09s  0.12s  0.09s  -bash
bodhi@bodhi-work:~$ last
bodhi     tty2     Sat Feb 11 19:19   still logged in
linuxlala tty3     Sat Feb 11 19:18   still logged in
bodhi     pts/1    :0              Sat Feb 11 19:11   gone - no logout
bodhi     tty7     :0              Sat Feb 11 19:09   gone - no logout
bodhi     tty7     :0              Sat Feb 11 19:08   19:09 (06:01)
reboot   system boot  4.4.0-59-generic Sat Feb 11 19:07   still running
bodhi     tty7     :0              Fri Feb 10 12:23   - crash (1+66:43)
reboot   system boot  4.4.0-59-generic Fri Feb 10 12:23   still running
bodhi     tty7     :0              Thu Feb  9 13:43   - crash (22:39)
reboot   system boot  4.4.0-59-generic Thu Feb  9 13:42   still running

```

» The `less` command is a program to view text files, where you use the arrow keys to scroll through a text file.

consuming process to copy all PDFs from one folder to another except for the ones that already exist in the destination folder. In the terminal however `cp -u *.pdf destination-folder` will do this without any fuss.

## Manipulate files

The Linux shell provides special characters called wildcards that help specify a bunch of filenames. Two of the most commonly used wildcards are `*` that matches any characters and `?` for matching a single character, e.g. `d*.pdf` matches all files that begins with the character d and ends with .pdf while `d????.pdf` limits itself to PDF files that begin with the character d but are followed by exactly three more characters, such as `d001.pdf` and `d002.pdf`.

Equipped with this knowledge we can now comprehend the `cp` command. The `-u` option only copies files that either don't exist, or are newer than the existing corresponding files, in the destination directory. You can also use the `-r` option to recursively copy directories and their contents, such as `cp -r Documents/ /shared/All_files/`.

Similar to `cp` is the `mv` command that's used to move files from the source to the destination. If the destination file exists, it's overwritten by the source file. Then there is the `rm` command that's used to delete files and directories following the command, such as `rm file1.txt` and `rm -r documents`. However, be careful when using `rm` since it zaps the files straightaway instead of moving them to a recycle bin.

**Top Tip:**  
Ctrl+Shift+C/V:  
Use these key combinations to copy and paste text from inside the terminal.



**Top Tip:**  
Append `&` to a command to run it in the background, such as `[command] &`. Use the job ID with the `fg` command.

## Hard and symbolic links

Links enable you to create an association between two files or directories. This is useful for maintaining multiple versions of a file or directory without the overhead of additional disk space for storing multiple copies. Links can either be hard or symbolic and we can use `ln` to create a link between two files.

By default, the `ln` command will create a hard link between these files. Hard links create an identical copy of the linked file on disk that gets updated automatically as the source file gets updated. While the content of the two files are linked, if the source file gets deleted, the target file will continue to exist as an

independent file. An important thing to note about hard links is that they only work on the current file system. You can't create a hard link to a file on a different file system. Also, hard links don't work for directories. You can however overcome these issues by using the `-s` switch to create a symbolic link.

# Handling filesystems

This may seem like plug and pray but it's not too complicated.

**T**he filesystem layout of a Linux system is as per the Filesystem Hierarchy Standard. The table (see *bottom right*) lists some of the important directories though there may be slight differences depending on your distro. Unlike Windows, Linux maintains a single filesystem tree and even external devices attach to it at various points.

The first step in managing a storage device is attaching it to the filesystem tree. This process is called mounting. A file named `/etc/fstab` lists the devices that will be mounted at boot. Read through `fstab`'s man page (`man fstab`) for an explanation of each of the fields in the file.

The `mount` command is used to mount file systems. Entering the command without arguments will display a list of the filesystem's currently mounted. There are two main components to mounting a device. The first is the actual name of the device file associated with the physical device. The Linux kernel treats all disk-like devices, such as SATA hard disks and USB drives as SCSI disks with names such as `/dev/sdaX`. The `X` is replaced with a number that represents the partition number. However, many distros associate a device with a text label that can be either a simple text or a randomly generated UUID (Universally Unique Identifier). The second component is the mount point, which is the directory where the device is attached to the filesystem tree.

Virtually all distros will automatically mount a removable disk and you can find its mount point with the `mount` command. However, when you're done writing to a USB disk, make sure you first unmount it before yanking it from the USB port, e.g. `umount /dev/sda1` unmounts the device associated with `/dev/sda1`. The unmounting process makes sure all data destined for the device has been transferred.

Data as we encounter it every day is in the form of files organised into structures that we can comprehend. However, in reality data exists in the form of blocks and there are times when you'll need to move it around in its raw form. For this purpose the Linux shell includes the `dd` utility (short for data

duplicator) that helps copy blocks of data from one place to another. While `dd` can be used for a variety of purposes, you'll mostly encounter this command when dealing with ISO images. Virtually all distros put out new releases in the form of ISO images that you can either burn onto an optical media or transfer to USB disk with the use of `dd`, such as:

```
$ sudo umount /dev/sdd
$ sudo dd if=/path/to/fedora.iso of=/dev/sdd bs=4M
$ sync
```

In these set of commands, we've first unmounted the USB drive connected to `/dev/sdd` before asking `dd` to copy the ISO file to the USB disk in chunks of four megabytes. The operation will take some time to complete but before yanking the disk issue the `sync` command to make sure all data has been transferred to the disk.

You can also use the `mount` command to mount an ISO image while it's still on the hard disk:

```
$ sudo mkdir /mnt/iso_image
$ sudo mount -t iso9660 -o loop image.iso /mnt/iso_image
```

Here we've first created a mount point named `/mnt/iso_image` and mounted the image file at that mount point which is now a loop device. A loop device is a pseudo-device that makes a file accessible as a block device. After the image is mounted, it can be treated just as though it were a real CD-ROM or DVD. Remember to unmount the image when it's no longer needed.

## Remote operations

Linux is well-known for its networking dexterity. Its popular graphical tools and applications derive their power from feature-rich command line utilities that you can use directly for more control over network operations. Two of the most popular commands used for debugging network-related issues are `ping` and `traceroute`. For instance, the command `ping linuxformat.com` will send a packet called an ICMP ECHO\_REQUEST to the specified host. Most web servers are configured to reply to this packet, which enables you to verify

**Top Tip:** `mkdir` creates directories, `mkdir all_documents/all_images/` creates two directories.

**Top Tip:** `netstat` displays loads of network stats: `netstat -ap | grep [name]` to see an app's ports.

## Keep files in order

The shell has various mechanisms to effectively create compressed archives and backup files. Archiving is the process of gathering up many files and bundling them together into a single large file. The `tar` program is the classic tool for archiving files, e.g. the command `tar cvf /documents.tar Documents/` would archive the whole `Documents` directory inside a single file called `documents.tar`. Conversely, `tar xvf /path/to/documents.tar` will extract the contents of the archive under the current directory.

Once you've created an archive, you can also compress it before backing it up. The `gzip` program is used to compress one or more files. When executed, it replaces the original file with

a compressed version of the original. The corresponding `gunzip` program is used to restore compressed files to their original, uncompressed form. The `bzip2` program is similar to `gzip`, but uses a different compression algorithm that achieves higher levels of compression at the cost of compression speed. A file compressed with `bzip2` has the extension `.bz2`, e.g. the command `bzip2 documents.tar` would replace the `documents.tar` file with the `documents.tar.bz2` file.

Once you've rolled your files into an archive, you can use `rsync` to back them up to a different location. `Rsync` does more than make copies of your files. You can use it to sync files on two

directories on the same PC, on two different computers on the same network or on machines thousands of miles apart over the internet, e.g. This command will sync a single file (`backup.tar`) on a local machine from one location to another location (`/tmp/backups/`):

```
$ rsync -vh backup.tar /tmp/backups/
```

The `v` option increases verbosity and `h` prints numbers in a human-readable format. You can as easily sync the file to a remote PC:

```
$ rsync -zvh --progress backup.tar
pi@192.168.3.101:/home/pi/backups/
```

The `z` option compresses the files before ferrying them and saves you bandwidth and time and `--progress` helps you track the transfer.

the connection. The command prints performance statistics after it is interrupted by pressing Ctrl+C. A properly functioning network will have no packet loss. You can also use the `traceroute` program which displays a listing of all the hops the network traffic takes to get from your local system to a specified host, e.g. type `traceroute techradar.com` to see the route taken by packets flowing from your computer to techradar.com's web server.

The Linux CLI offers several options for sending files over the network. A popular command-line program for file downloading is `wget`. It is useful for downloading content from both web and FTP sites. The utility can download everything from single files and multiple files to even entire sites. You'll most often use it to fetch ISO images:

```
$ wget -c http://releases.ubuntu.com/16.04.1/ubuntu-16.04.1-desktop-amd64.iso
```

The `-c` option asks `wget` to grab a partially-downloaded file and is useful for resuming interrupted downloads.

It's often useful to verify the integrity of an ISO image that you have downloaded. In most cases, a distributor of an image will also supply a checksum file which contains a string of alphanumeric characters that have been calculated from image. If the contents of the image file changes by even one bit, the resulting checksum will be very different. Checksums are most commonly generated by the `md5sum` program. After downloading an image, you should run `md5sum` against the image and compare the results with the `md5sum` value supplied by the publisher:

```
$ md5sum downloaded-image.iso
85f43dc4c4ceb007661a3044845g243c downloaded-image.iso
```

## Secure transfers

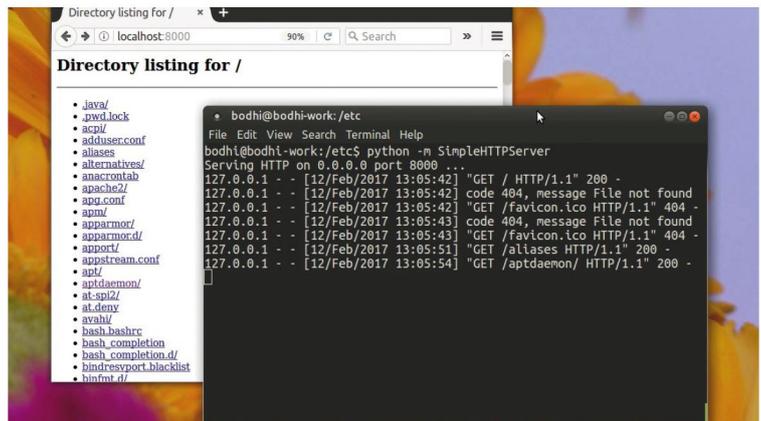
The first generation of tools that were designed to access remote computers conducted their business in cleartext. Later, a protocol called SSH was written to securely communicate with the remote machine. Virtually all Linux distros ship with an implementation of SSH which is called OpenSSH. You can use this to run a secure terminal session over the network. All traffic passing over the session is encrypted including passwords. A SSH setup includes an SSH server that runs on the remote host and listens for incoming connections on port 22 and an SSH client that's used on the local system to communicate with the remote server. You'll find various tutorials about SSH in previous issues of *Linux Format* issues.

The most common use for SSH these days is to remotely access a Raspberry Pi. Assuming the IP address of the Pi is **192.168.3.101**, you can access it via SSH with:

```
$ sudo ssh pi@192.168.3.101
```

This SSH client will connect to the SSH server running on the Raspberry Pi and prompt you for the password of the pi user. The first time a connection is attempted, you'll have to accept the credentials of the remote host. Once authenticated, you'll get the shell prompt of the remote Raspberry Pi host. Any commands you enter now will be executed on the Raspberry Pi. When you're done, type `exit` to terminate the remote session and return to your local shell.

The OpenSSH package also includes `scp` (secure copy) that makes use of an SSH encrypted tunnel to copy files across the network. Unlike the tradition `cp` command, when using the `scp` command you'll have to ensure that the



▶ The command `python -m SimpleHTTPServer` uses Python's built-in HTTP server to show the current directory in a web browser on localhost:8000.

source or destination pathnames also include the name of a remote host. For example, to copy a file named **document.txt** from our home directory to the remote Pi, we could do this:

```
$ scp ~/document.txt pi@192.168.3.101:/home/pi/
Documents/
```

To copy files from a remote system down to a local machine, the order of the `scp` command just needs to be reversed so that the first, or source, argument is the remote system and the second, or destination, argument is a directory on the local system. To illustrate, the following we'll copy an ISO image in the Downloads directory on the remote Pi into the current directory on the local machine:

```
$ scp pi@192.168.3.101:/home/pi/Downloads/distro.iso .
```

**Top Tip:**  
file: use the file command to determine a file's type. It prints details of the file's contents.

## Important directories inside your distro

Directory	Purpose
/	The root directory of everything.
/boot	Contains the Linux kernel and the boot loader.
/bin	Contains essential programs that must be present for the system to boot and run.
/dev	This is where the kernel maintains a list of all the devices that it understands.
/etc	Houses all of the system-wide configuration files.
/home	Each user is given a directory under <b>/home</b> which is the only place they can write files without escalated permissions.
/lib	Contains shared library files used by the core system programs. Your installation will also have <b>/lib32</b> and <b>/lib64</b> that contain architecture-specific shared libraries.
/media	Contains the mount points for removable media, such as USB drives and DVDs, and other partitions that are mounted automatically at insertion.
/opt	This is used to install some software, such as <i>VirtualBox</i> .
/proc	This is a virtual file system maintained by the Linux kernel that doesn't contain real files but rather system information.
/tmp	This directory is intended for storage of temporary files created by various programs, clear every restart.
/usr	The largest directory that contains all the programs and support files. Also houses other important directories such as <b>/usr/bin</b> that holds executable programs and commands installed by Linux. There's also <b>/usr/share</b> that contains all the shared data used by programs in <b>/usr/bin</b> .

# Secure your system

Don't trust your feelings, how do you know that you are yourself?

**Top Tip:**  
**fsck:** Checks the integrity of a file system before it is mounted, such as `sudo fsck /dev/sdb1`.

Linux is known for its security prowess, although we admit it hasn't been that good in the last few years with the numerous exploits and breaches. Earlier in the feature we mentioned the `/etc/passwd` file that records details about the users in the installation. The file uses the following format:

```
[username]:[x]:[UID]:[GID]:[Comment]:[Home directory]:[Default shell]
```

Some fields such as `[username]` and `[Home directory]` are self explanatory. The `x` in the second field points to the fact that the account is protected by a shadow password. The `[UID]` and `[GID]` are the numerical representation of the user and primary group that a user belongs to. The `[Default shell]` is the shell that will be made available to this user when they login into the system. Most Linux distros will default to the `Bash` shell but there are several others on offer. Then there's the `/etc/shadow` file which contains the encrypted password as well as other information, such as account or password expiration values.

The permissions on the `/etc/shadow` are set to prevent any user from even reading the file. But if no one can access the file how can users change their passwords which are stored in this file? This is because the `passwd` utility uses a special permission known as SUID (Set User ID).

Thanks to this special provision, the user running the `passwd` command temporarily becomes root while the command is running and can proceed to write to the `/etc/`

`shadow` file. To append the setuid bit to a file, add `+s` for the user, for example `chmod u+s /path/to/file`. Now anyone who attempts to access this file will do so as if they are the owner of the file.

## Sudo primer

To grant access to `sudo`, a system administrator must edit the `/etc/sudoers` file. It's recommended that this file is edited using the `visudo` command instead of opening it directly with a text editor. Entering `sudo visudo` opens the `/etc/sudoers` file using `vi` text editor. The Defaults `secure_path=` line specifies the path use for every command run from `sudo`. It's followed by lines that specify permissions for root, any other user and certain groups (marked by the `%` sign) on the installation. The rule for the root user reads something like this:

```
root ALL=(ALL) ALL
```

This means that the root user can run any command on any host as any user. To allow user mayank to add and remove users on all machines, we can add this line:

```
mayank ALL=/usr/bin/useradd, /usr/bin/userdel
```

The file has an extensive man page (`man sudoers`) that has more examples and features.

## Superintend processes

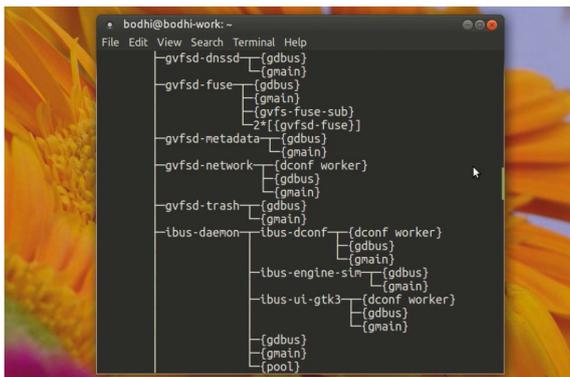
The average home user doesn't need to have the proficiency and the dexterity of a professionally qualified system administrator. However, there are a few skills that should be in your repertoire to manage your computer efficiently.

One important aspect of a system administrator's job is to make sure that the computer you are looking after runs smoothly and you can do so by monitoring it. The `ps -e` command will print a long list of all the processes on the system. You can truncate the list to display only processes that have the same UID as the current user by invoking the `ps` command without any switches. The output of the `ps` command includes the unique process ID (PID) along with the name of the running program.

It's more common to pipe (see *Pied Piper box, bottom right*) the list to display information about a specific program. The command `ps aux | grep libreoffice` will display detailed

**Top Tip:**  
**find:** Helps search for files, for example, `find . -iname 'doc'` looks for files whose name begins with 'doc'.

» The `pstree` command shows all the processes that are currently running along with their associated child processes in a tree-like format.



## Control your services

Many popular distros have traditionally used `SysVinit` for starting and controlling services. But this has now been replaced, in a somewhat controversial fashion for some users, by a new system and services manager called `Systemd`. The `Systemd` service manager uses the `systemctl` command to control the services. If you enter the `systemctl` command into a terminal window it will list the status of everything that is controlled by `Systemd`.

If you follow this up with `systemctl list-units -t service` command, which will list the active services. You can check the status of any individual service with the `systemctl status` command, such as `systemctl status sshd.service`. Similarly, you can change the state of the service by replacing `status` with `start`, `restart` and `stop`. For example, `sudo systemctl start sshd.service` will start the service, if it isn't already running.

If you want a service to start at boot, you will need to use the enable option, such as `sudo systemctl enable sshd.service`. Similarly, the command `sudo systemctl disable sshd.service` disables it from starting at boot. You can also use `systemctl` to power cycle the computer. The command for this is `systemctl poweroff` which will power down the system and `systemctl reboot` will restart it.



➤ Since crontabs use an odd format, you can use the online utility at [www.corntab.com](http://www.corntab.com) to easily setup crontabs.

information only about the processes associated with *LibreOffice*, for instance. Once you know the PID of a process you can terminate that process with the `kill` command. Assuming that *LibreOffice* has a pid of 8899, you can terminate an unresponsive session with the `sudo kill 8899` command. It's also possible to send signals to multiple processes matching a specified program or username by using the `killall` command, such as `killall vlc`.

You can also use the `top` command to view a list of running tasks with the most CPU-intensive ones listed first. The `top` command also displays various bits of useful information about the processes including their PID. It also has a few interactive commands. For example, you can kill a process from within `top` by pressing the K key which prompts you to enter the PID of the process you want to terminate.

## Schedule tasks

There are tasks that are more effective when you run them on a schedule, for example regular backups and downloads.

The `at` command helps you schedule tasks that you need to run at a specific time and date. For example, if you need to download a large file, it's best to schedule it late in the night, for example 1am in the morning. The command `at 1am tomorrow` will change the prompt to `at>` and everything that you type at this prompt will be run at the time that you specify. To download a file, point to its location with the `wget` command. Press the Enter key to specify another command, for example, if you want to move the file to a specific folder. When you are done, press Ctrl+D to save the job. At the time, the `at` command will perform the actions you've told it to and the `at` command allows for fairly complex time specifications. In addition to AM and PM, the command also accepts times in the HH:MM format and particular dates.

When you press Ctrl+D to submit a job, the command prints a job ID. Use the `atq` command to list all submitted

## Greater force powers

Sage advice for your journey to master-level command-line Jedi:

**1 screen** If you SSH into remote machine regularly, you'll find `screen` to be a wonderful companion. Using the utility you can run a process without maintaining an active shell session. To run a time consuming process inside a remote machine, first connect to it with SSH. Then type `screen` to start the screen session. You can now initiate the time-consuming process like an update or a download. When it's underway, press Ctrl+A and D to detach from the `screen` session but leave the process running. You can then terminate the SSH process. Later on when you reconnect to the SSH session, you can check on the status of the update or download by restoring the screen session with `screen -r`. Use `screen -ls` if you have multiple screen sessions and then reconnect to the one you want by specifying its session ID, such as `screen -r 6754`.

**2 renice** If you have a process that's hogging the resources on your computer, you can use the `renice` command to assign it a lower priority. Linux assigns a priority to each process and the ones with higher priority are first given access to the system's resources while others with lower priority have to wait for their turn. The `renice` command can assign a priority value between -20 and 19. The lower the number the higher the priority. If you want to take away resources from LibreOffice (PID: 8899) for example, you can assign it a lower priority number with `renice 15 8899`.

**3 xkill** Helps you terminate graphical process. You can either type the command in the terminal or the run dialog box which changes the pointer into a cross-hair cursor. Now click on any non-responsive window to kill it. Right-click to dismiss `xkill` without killing a process.

**4 dmidecode** You can use the `dmidecode` command to know more details about your computer's hardware. For example, use `dmidecode -t 16` to list details about the physical memory. Try browsing the `dmidecode` man page for a comprehensive list of supported options. Once you get comfortable working with SSH, use `sshfs` to mount remote partitions. For example, `sshfs user@remotehost:/home/bodhi/media/remotefs` mounts the remote home directory under the local filesystem.

jobs which you can then remove with the `atrm` command by suffixing the ID of the job you wish to delete.

If you wish to run a task repeatedly, then you're better off using the *Cron* daemon. Use the `/ crontab -e` command to edit your crontab file. The first time you run the command you'll be asked to select one of the available command-line text editors. Each job is specified in seven fields that define the time to run, owner, and command. The first five commands specify the execution time, the sixth defines the day of the week, and the last field lists the command to be executed. You can use the `/ crontab -l` command to list your scheduled jobs. System-wide crontabs are stored in `/etc/crontab` and user-specific crontabs are under the `/var/spool/cron` directory. Taken your first steps have you, but mastery of the terminal is a long journey, so continue to buy *Linux Format* you must! **LXF**

**Top Tip:** `tmux`: Can create many sessions, run tasks in each and switch between them without stopping the others.



## Combine the force with pipes

One of the best things about the CLI is that it lets you combine small individual commands into a complex and useful task without much effort. This is accomplished with the pipe (`|`) operator. It is used to sending the output of one command as the input for another. For example, `ls -a | grep _` uses the `ls` and `grep` commands to list all files that contain the underscore

character. In the example, we've used the `|` operator to sort through the verbose output of the `ps` command to find the processes that are associated with a particular application, such as *LibreOffice*.

You can do some amazing things with the `|` operator. We mentioned earlier that *Bash* maintains a history of commands. You can print

the entire log with the `history` command or scroll through the list by piping the output to the `less` command such as `history | less`. You can also use `|` to chain together a number of commands. For example, use `sed` with the underscore (`_`) to change it to a hyphen (`-`) with `ls -a | grep _ | sed "s/_/-g"`.