I'VE GOT THE POWER

UNLEASH THE HIDDEN STRENGTH OF THE LINUX COMMAND LINE

If you haven't mastered the command line, you're missing out on the most powerful features of Linux. **Mike Saunders** has tips galore for both newbies and old-timers...

"The command line can do certain jobs much more efficiently than GUI apps."

You know how in films, when they want to portray a computer genius/nerd/hacker at work, they always show someone tapping incomprehensible gobbledygook into a command line? Sometimes it'll be a green-on-black text terminal from the 1980s, accompanied by various beeping noises, just to add to the mystique. And thanks to stereotypes like these, many non-technical people assume that the command line is a weird and arcane tool, only to be used if there's no pointy-clicky GUI goodness at hand.

Now, as a Linux user, you already know that that's nonsense, and the command line interface has its benefits. But have you really delved deeply into it? Have you discovered all its hidden tricks? And have you been able to ditch the mouse and start working more quickly? Over the next seven pages we'll show you how the command line interface (CLI) can do certain jobs much more efficiently than GUI applications, making your day-to-day Linux life smarter, easier and faster. Even if you've been using the CLI for a while, you'll find plenty of new gems in here, so let's get started.

Better file management

GUI file managers are clunky and slow. Here's how to work at light speed.

Dolphin, Nautilus, Thunar and co. are OK for simple drag-and-drop jobs, but in all honesty they don't compare to the CLI. As soon as you need to do something complicated, you end up with a horribly long workflow involving countless mouse clicks until your wrists get overloaded with RSI.

Let's take a complex job and see how it can be made simpler with some CLI magic. Even though you might not need this specific command on a day-to-day basis, you can break out the component parts and use them on your Linux travels in future, saving you heaps of time. So: imagine that you have a bunch of files without any extensions, and you don't know what's inside them. (You can see hundreds of these in Firefox's cache, for example.) They look like this:

```
3F7DFd01
E64C7d01
C42F9d01
F0887d01
...
```

Let's say you want to open the 10 biggest JPEG files in Gimp to have a look at them. Think about how you'd do that in your graphical file manager – if it's even possible. Providing that your file manager can peek inside the contents of files to determine their format, you might be able to click around and somehow sort the list by file format and size simultaneously (very few file managers can do that), and then click and drag to select the top 10, and right-click on the selection to open them in a program, then click down the list to find Gimp... Ugh.

Now check this command out:

```
file * | grep JPEG | sed s/:.*// | xargs ls -S | head -n10
| xargs gimp
```

It's a beast, isn't it? But actually, it's a bunch of smaller commands linked together, done in such as way that you can understand what each part does.

First, the **file *** part looks at every file in the current directory, and works out the filetype from the bytes contained inside. So you get lines like this:

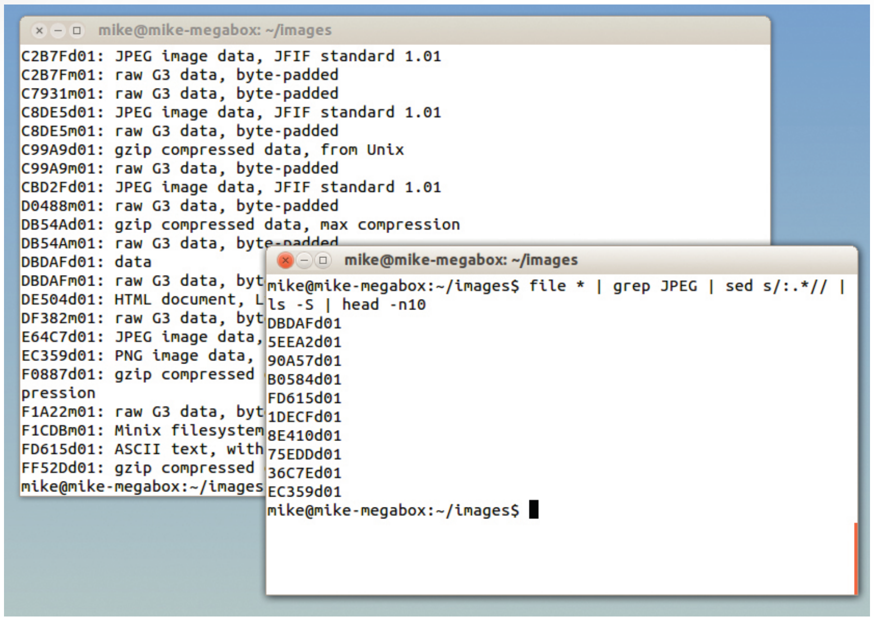
```
CBD2Fd01: JPEG image data, JFIF standard 1.01
D0488m01: raw G3 data, byte-padded
DB54Ad01: gzip compressed data, max compression
```

We only want JPEG files, so we take the output from the **file *** command via a pipe (**|**), then **grep** to just retrieve lines containing **JPEG**. After this point we don't need any information other than the filename, so we pipe the text to **sed**, the stream editor, which does a replacement. It takes a colon **:** followed by any sequence of characters (**.***) and replaces it with nothing (**//** – ie nothing between the slashes). So it gets rid of everything but the filenames.

Then, using **xargs ls**, we bundle together all the filenames we've got so far and list them, sorting by size (**-S**). The **head** part retrieves the top 10 items of the list, and then using **xargs** again, we bundle up all the filenames into a single string and tell Gimp to open them.

It might take a few re-reads to really grok all this, but once you have your head around it, you can see how powerful the CLI is for working with files. (For instance, you could replace **xargs gimp** with **xargs rm** to delete those 10 biggest JPEG files.) Try adding your own components to the command, and making new ones using parts of it.

"Once you have your head around it, you can see how powerful the CLI is for working with files."



```
C2B7Fd01: JPEG image data, JFIF standard 1.01
C2B7Fm01: raw G3 data, byte-padded
C7931m01: raw G3 data, byte-padded
C8DE5d01: JPEG image data, JFIF standard 1.01
C8DE5m01: raw G3 data, byte-padded
C99A9d01: gzip compressed data, from Unix
C99A9m01: raw G3 data, byte-padded
CBD2Fd01: JPEG image data, JFIF standard 1.01
D0488m01: raw G3 data, byte-padded
DB54Ad01: gzip compressed data, max compression
DB54Am01: raw G3 data, byte-padded
DBDAFd01: data
DBDAFm01: raw G3 data, byt
DE504d01: HTML document, L
DF382m01: raw G3 data, byt
E64C7d01: JPEG image data,
EC359d01: PNG image data,
F0887d01: gzip compressed
pression
F1A22m01: raw G3 data, byt
F1CDBm01: Minix filesystem
FD615d01: ASCII text, with
FF52Dd01: gzip compressed
mike@mike-megabox:~/images
```

```
mike@mike-megabox:~/images$ file * | grep JPEG | sed s/:.*// |
ls -S | head -n10
DBDAFd01
5EEA2d01
90A57d01
B0584d01
FD615d01
1DECFd01
8E410d01
75EDDd01
36C7Ed01
EC359d01
mike@mike-megabox:~/images$
```

Using pipes and multiple commands, you can narrow down to just the filenames you need, all without hundreds of tedious mouse clicks.

Essential tips

If you're new to the command line, here are some things you absolutely need to know. In most Linux distros, the CLI is accessible in your desktop's program menu as Terminal, XTerm or Konsole.

- **ls/cd/rm/mv** The most common commands (list files, change directory, remove file and move/rename file). Each command has a manual page (eg **man ls** – hit Q to quit the viewer). Many commands have extra options; for example, **ls -la** lists all files, including hidden ones, with details.
- **Tab** Hit the Tab key to automatically complete a filename or directory. If you want to delete **foobarlongfile.txt**, for instance, enter **rm foo** and hit Tab, and it should be completed.
- **History** Use the up and down cursor keys to navigate through previous commands. You can edit them as well.
- **~** Your home directory (eg **/home/bob/**)
- **>** and **>>** sends output of a command to a file, overwriting (**>**) or appending (**>>**). Eg **ls -l > list.txt**.

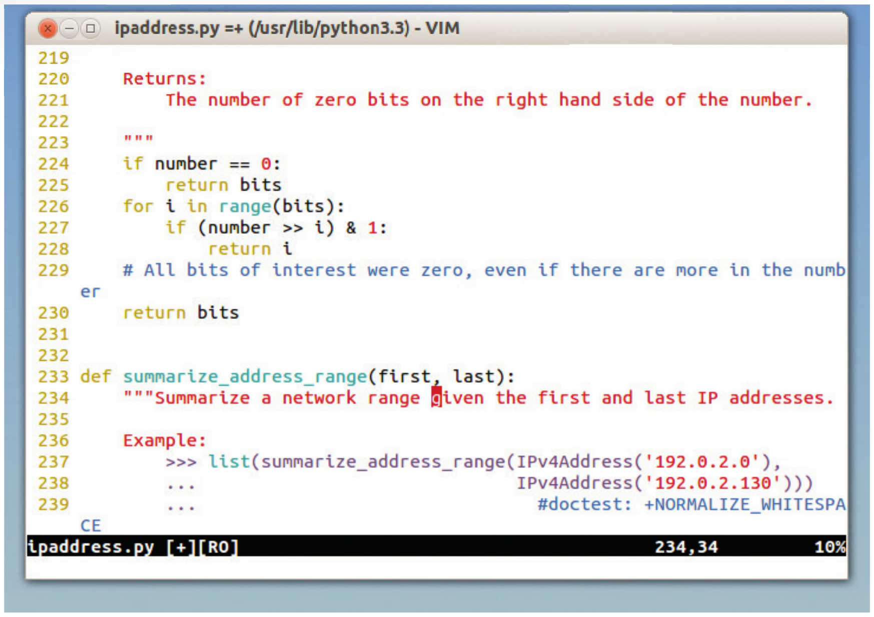
Better editing

Learning a good text editor on the command line is essential.

We can't stress enough how important it is to learn a good text editor. It really makes a vast difference to how you work – even if you're not a programmer. Some GUI text editors are well-specced with plenty of features, but when you're working with plain text, why should you keep moving your hands away from the keyboard to grab the mouse?

The two most notable text editors are Emacs and Vim. They both have their strengths and weaknesses, but we'll focus on the latter here because it's installed in nigh-on every Linux distro by default.

This isn't a guide to the basics – we did that in issue 1. If you don't have that issue and you've never used Vim before, see the "Micro guide" box and then do the **vimtutor**. Here we'll explain why it's well worth learning, and if you're a regular Vim user, we'll show you some tricks that you might not have come across.



```
ipaddress.py =+ (/usr/lib/python3.3) - VIM

        Returns:
            The number of zero bits on the right hand side of the number.

        """
        if number == 0:
            return bits
        for i in range(bits):
            if (number >> i) & 1:
                return i
        # All bits of interest were zero, even if there are more in the numb
    er
        return bits


    def summarize_address_range(first, last):
        """Summarize a network range given the first and last IP addresses.

        Example:
            >>> list(summarize_address_range(IPv4Address('192.0.2.0'),
            ...                              IPv4Address('192.0.2.130')))
            ...                                #doctest: +NORMALIZE_WHITESPA
    CE
ipaddress.py [+][RO]                                   234,34          10%
```

A well-tuned **~/.vimrc** file makes Vim more attractive, informative and welcoming.

How to love Vim

Many people try Vim and come away frustrated, because they don't spend time getting into the right mindset. Some people use it often but never learn to enjoy it. That's fair enough – it's not a very welcoming program. But bear these in mind and you'll learn to love it:

- Always switch back command mode (with Esc) straight after editing. Make command mode your default mode. Vim is a modal editor, which means sometimes you're editing text, and sometimes you're giving commands. You should only be in insert mode when you're editing text, so always hit Esc as soon as you're done. You'll learn commands better this way, instead of accidentally adding them to your text.
- Use the H, J, K and L keys to navigate. These are on the home row, ie under your fingers, so you don't have to move your hands down to the cursor keys. They really help you work faster – although it might take a few days to get used to them.
- Treat commands as a language. At first, Vim's commands look weird and cryptic, but when you piece them together they make more sense. For instance, in Vim-speak **d** is delete, **a** means around an object, and **P** refers to a paragraph. Hit **dap** and voila: delete text around a paragraph (that is, text inside the paragraph and trailing spaces).

It's also worth customising the **.vimrc** file in your home directory to make the editor a bit friendlier. Here's what we have:

```
set number ruler laststatus=2 hlsearch ignorecase title
syntax on
```

This adds: line numbering; a ruler showing the current line number; a status line with the filename; highlighted searches; case-insensitive searches; more information in the terminal window title; and syntax highlighting for various programming languages.

Advanced Vim tricks

Vim is chock-full of keyboard shortcuts and commands that make life easier. Want to search for the next instance of the word under the cursor? Hit * (asterisk). Doing some programming, and want to find a matching bracket or brace? Move the cursor over the bracket and hit **%**. Want to quickly go back to the last place you entered text? Use **gi**.

Earlier we mentioned using **dap** to delete a whole paragraph. This is an example of using a text object, and this is where Vim is ridiculously powerful. For instance, **das** deletes a sentence, while **ci"** (C-I-double quotes) changes text inside a pair of double quotes. Say you have this text:

```
<img src="foo/bar/baz.jpg" />
```

To change the filename: move the cursor anywhere inside the quotes, and hit **ci"**. Vim will remove all text inside the quotes and place you in insert mode, so you can type in some new text and hit Esc when you're done. And this opens up possibilities for the equally awesome **.** (dot) command. Basically, **.** repeats the last text editing action – both the command(s) you used in Vim and the text you typed. So if you move the cursor into another **<img src...** line, between the quotes, and hit **.** then the text will be replaced again, exactly like the first time.

This is tremendously useful if you need to do a lot of quick replacements: do the command once, then jump around to other places and tap **.** where necessary. Because **.**

"Vim is chock-full of keyboard shortcuts and commands that make life easier."

includes a whole text action, it can even be used to repeat editing operations with backspaces inside.

Imagine you have some function prototypes copied from a header file:

```
int foo(int a, int b);
void bar(char *d);
void baz(int a, bool d);
```

Now you want to implement the functions themselves. Go to the first line (**int foo**) and tap A (capital) to append text onto the end of the line. Hit Backspace to remove the semi-colon, Enter (for a newline), **{**, Enter again, **}**, and Enter once more. Then hit Esc to get back to command mode. Now the first line has changed into this:

```
int foo(int a, int b)
{
}
```

So, we've converted a prototype into a proper function. Now move the cursor to the second prototype line (void bar), hit **.** and *voilà*, it is also converted, using the exact same editing action as before. You can then hit **.** again to convert the third line. It's a massive time saver.

Globalisation

Another hugely powerful (and not well known) command is **:g** – the global command. Take this for instance:

```
:g/someword/m0
```

This takes all lines containing the word **someword** and moves them to line zero, ie the top of the file. Or you could use **:g/someword/d** to delete all lines containing **someword**.

An especially useful add-on option for **:g** is **norm**, which puts Vim into normal (command) mode, and then executes the commands as written. For instance, say you have some Python code and you want to comment out all lines containing **DEBUG** by putting hash marks at the start:

```
:g/DEBUG/norm 0i#
```

Here, for each line containing **DEBUG**, Vim executes **0i#** – that is, go to the start of the line, switch to insert mode, and add a hash. How cool is that? And then you can even add Esc keystrokes when entering a **:g** command by tapping Ctrl+V and then Esc.

Save time and energy

Imagine you want to add C-like comments to DEBUG lines, so that:

```
printf("DEBUG: Blah blah");
```

becomes:

```
/* printf("DEBUG: Blah blah"); */
```

Use this:

```
:g/DEBUG/norm 0i/* ^[A */
```

(Again, use Ctrl+V followed by Esc to input the **^[** escape character here.) This goes to the start of the line, inserts **/*** followed by a space, then Escapes back out of insert mode, goes to the end, and adds ***/**.

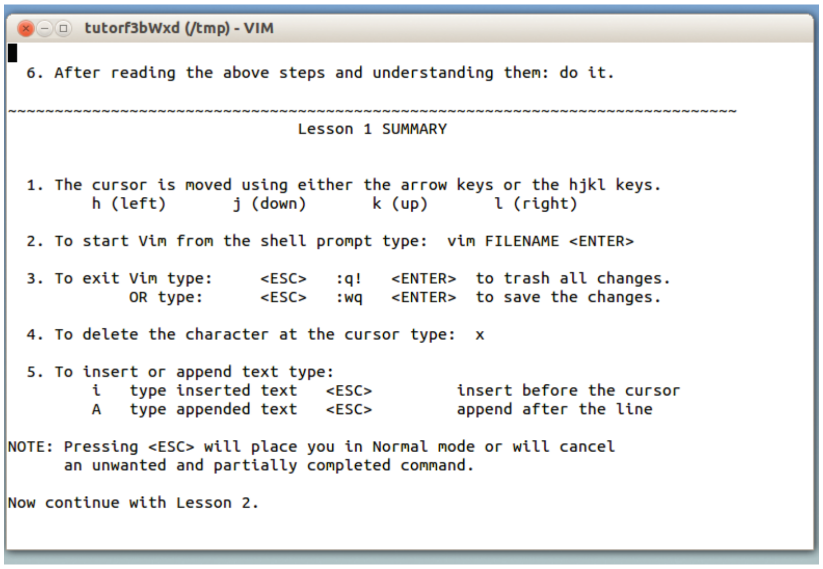
Just like we said – ridiculously powerful. And more fun than using dreaded regular expressions. (PS: Turn to the inside back cover of this very magazine for a great cheat-sheet for Vim commands!)

New to Vim? Here's a micro guide

Enter **vim newfile.txt** to edit a new file. Hit **I** and you'll see **-- INSERT --** at the bottom, which means you're in the Vim mode for adding text. Type in a few lines. When you're done, hit Esc to return to command mode.

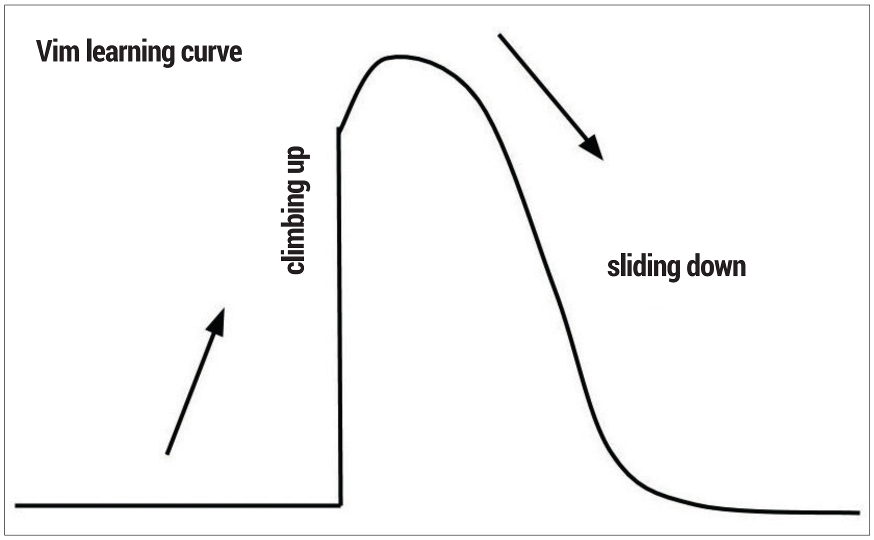
Use the **H/J/K/L** keys to move around. Hit **X** to delete a character under the cursor and **DD** to delete a line. Use **0** (zero) to go to the start of a line, and **$** to jump to the end. Type a number and press **Shift+G** to go to that line. Hit **Ctrl+G** to view the current line number and U to undo an operation.

To save, make sure you're in command mode (hit Esc to be sure) and type **:W**. To quit, use **:Q**. To quit without saving, **:Q!**. Those are the basics – now enter **vimtutor** and follow the more detailed guide, which will take about 20 minutes. Then you'll be ready to use the tips here.



```
  6. After reading the above steps and understanding them: do it.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                               Lesson 1 SUMMARY


  1. The cursor is moved using either the arrow keys or the hjkl keys.
         h (left)       j (down)       k (up)       l (right)

  2. To start Vim from the shell prompt type:  vim FILENAME <ENTER>

  3. To exit Vim type:     <ESC>   :q!   <ENTER>  to trash all changes.
             OR type:      <ESC>   :wq   <ENTER>  to save the changes.

  4. To delete the character at the cursor type:  x

  5. To insert or append text type:
         i   type inserted text   <ESC>         insert before the cursor
         A   type appended text   <ESC>         append after the line

NOTE: Pressing <ESC> will place you in Normal mode or will cancel
      an unwanted and partially completed command.

Now continue with Lesson 2.
```

Although the Vimtutor doesn't make you an expert in Vim, it gets you well-versed with the basics of this powerful, flexible text editor (and its many offshoots).



Vim's learning curve, jocularly depicted at **http://tinyurl.com/nfbj3mv** ("Why I use Vim" by Pascal Precht). There's a huge amount to learn at the start, but it all makes sense with time.

Better image editing

Huh? The CLI is good at editing pictures? Surely you can't be serious…

I am serious, and don't call me Shirley. Yes, there are many cases where it's easier and faster to edit images at the command line, rather than doing them in pointy-click fashion via a graphical tool like Gimp. This is especially true if you want to perform editing or processing operations on multiple files at the same time – in other words, batch processing.

The suite of programs we're using here is ImageMagick, which you've probably heard of if you've been on the Linux scene for a while, because it has been in development since the early 90s. ImageMagick is monumentally versatile and supports over 100 different file formats – so it will handle nigh-on anything you throw at it.

The most commonly used tool in ImageMagick is **convert**, which works like this example command:

```
convert image.png image.jpg
```

Pretty simple, right? This just makes a JPEG version of the PNG file. Of course, when you're generating JPEGs you'll often want to alter the quality:

```
convert -quality 90 image.png image.jpg
```

Resizing is possible as well. The first command here specifies a percentage of the original size, while the second uses exact dimensions:

```
convert -resize 75% image.png image.jpg
convert -resize 300x300 image.png image.jpg
```

Don't be so square

Something interesting happens with the second command, and it's to do with aspect ratios. If the source image isn't a square, the resulting image will be 300 pixels wide and however many pixels tall to match the original aspect ratio. If you want to force the image to be 300 x 300 pixels and ignore the aspect ratio, add blackslash-exclamation to the dimensions, like so:

```
convert -resize 300x300\! image.png image.jpg
```

Along with file format conversions and resizing, another common job is to crop an image – that is, only save a portion of the original. This is fairly straightforward too:

```
convert -crop 250x100+20+40 image.png image.jpg
```

This takes a 250-pixel-wide and 100-pixel-high chunk of the original picture, from 20 pixels across and 40 pixels down, and saves it into **image.jpg**. For crop operations you might not want to convert the file – instead, you just want to overwrite the original. You can do this by changing the **convert** command to **mogrify** and omitting the destination file:

```
mogrify -crop 250x100+20+40 image.png
```

Another useful option is **rotate**, which takes the amount of degrees (clockwise):

```
mogrify -rotate 90 image.png
```

Lightning fast batch jobs

So far so good, but these commands aren't much quicker than doing the same jobs in a graphical editor. But! When we add some command line scripting into the mix, it all becomes a lot more efficient. Say you have 200 **.png** files in the current directory, and you want to shrink them all to 50% of their original sizes:

```
for f in *.png; do mogrify -resize 50% $f; done
```

Here we create a loop, saying that for every file in the current directory that has a **.png** extension, we perform a **mogrify** operation on that file (the filename is contained within the **$f** variable).

What about if you want to convert all the PNGs to a different format? You could do this:

```
for f in *.png; do convert $f $f.jpg; done
```

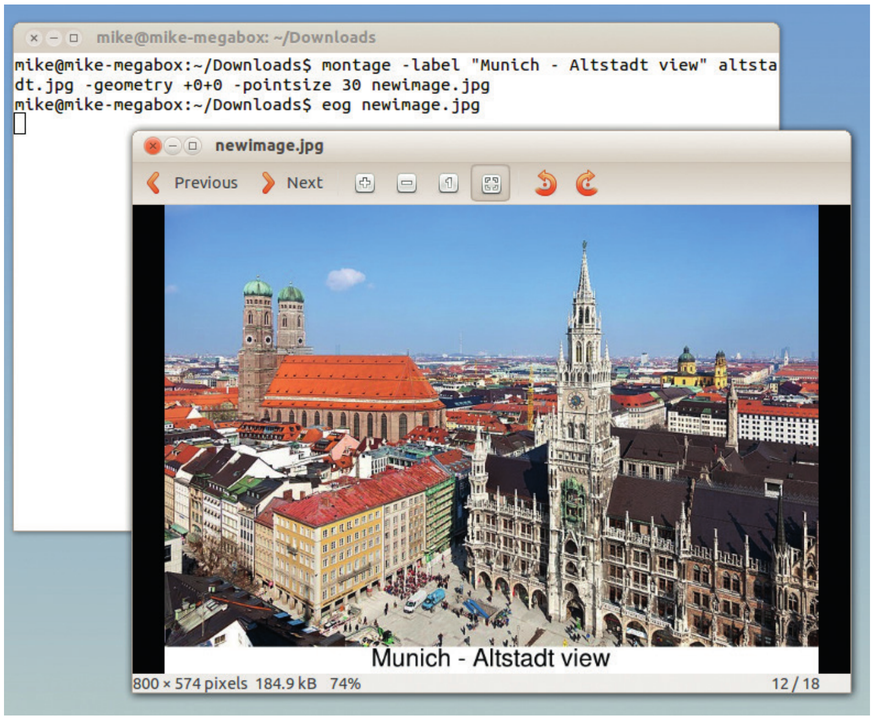
But the resulting filenames are a bit ugly here – **foo.png** becomes **foo.png.jpg**, **blah.png** becomes **blah.png.jpg**, and so forth. However, using a command line trick called parameter substitution, we can remove the **.png** from the destination filenames:

```
for f in *.png; do convert $f ${f%.png}.jpg; done
```

Here, the **${f%.png}.jpg** bit does the clever work, removing **.png** and then adding **.jpg** on to the filename stem. (You can also use **mogrify** to convert images and replace their extensions, but it's worth knowing these tricks for the future.)

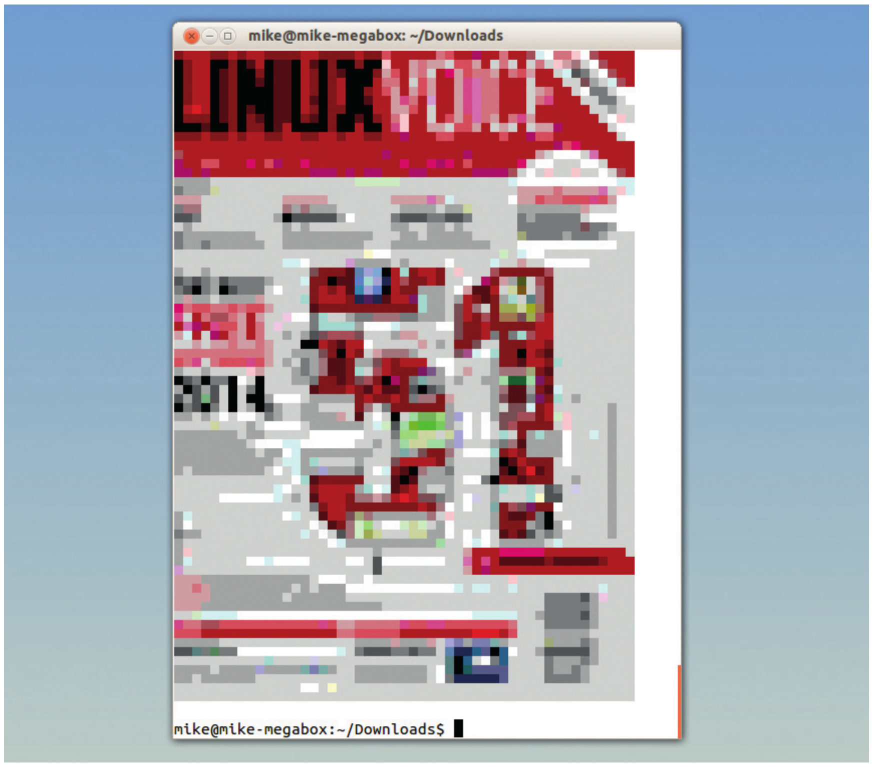
So, with the **convert** tool and some command line scripting, you can do

"Image Magick is monumentally versatile and supports over 100 different file formats."



```
mike@mike-megabox:~/Downloads$ montage -label "Munich - Altstadt view" altsta
dt.jpg -geometry +0+0 -pointsize 30 newimage.jpg
mike@mike-megabox:~/Downloads$ eog newimage.jpg
```

Convert, resize, flip, crop and add captions to hundreds of images in seconds, thanks to ImageMagick.



```
mike@mike-megabox:~/Downloads$
```

Use shellpic (**https://github.com/larsjsol/shellpic**) to view images in the terminal – handy if you're SSHed into a remote server and want a quick preview of an image file.

conversion, resizing and cropping jobs on hundreds of images in a matter of seconds. If you had to do all the alterations by hand, it'd take hours or even days. ImageMagick has more cunning features though, so let's take a closer look.

If you're working with batches of photos, you'll often need to correct their brightness and contrast settings. The **convert** and **mogrify** tools have an option for this:

```
mogrify -brightness-contrast 20x-30 image.jpg
```

This improves the brightness of the image by 20%, and reduces the contrast by 30%. Again, you could include this **mogrify** command in a 'for' loop as discussed earlier, to fix hundreds of images at once.

ImageMagick is packed full of filters, such as blurring:

```
mogrify -blur 5x2 image.jpg
```

The first number here is the radius, while the second is the sigma (the actual amount of blurring). Try playing around with different values. You may not think it, but you can even turn pictures into charcoal drawings with a single command:

```
mogrify -charcoal 5 image.jpg
```

Another tool included in ImageMagick is **montage**, which creates a single image from a bunch of images. It's also useful for adding captions onto images, like so:

```
montage -label "My caption" image.jpg -geometry
+0+0 -pointsize 30 newimage.jpg
```

This adds the words "My caption" in 30 point font to the bottom of the picture, without resizing the picture (hence the +0+0), and writes out the result to **newimage.jpg**.

Command-line line drawing

One of ImageMagick's most powerful features is its set of drawing commands. You can add all kinds of shapes to images via the command line, which is also useful when you're doing batch processing jobs and want to add labels or diagrams to individual images. Take a look at this simple example:

```
mogrify -fill white -stroke black -draw "rectangle
30,10 200,100" file.png
```

This creates a white rectangle with a 1-pixel black border, 200 pixels wide and 100 pixels tall, and places it at 30 pixels across and 10 pixels down on **file.png**. Many other options are available for drawing circles, polygons and Bézier curves – see the full list at **www.imagemagick.org/script/command-line-options.php**.

Better calculating

Doing calculations at the command line makes much more sense than clicking loads of little buttons, over and over and over. With Qalc, part of the Qulculator suite (which also includes GUI tools) you can do some very funky stuff. On Debian/Ubuntu/Mint-based systems, grab the command line tool with **sudo apt-get install qalc**. The program's manual page is disappointingly small, and there's little else in the way of documentation, so the best way to learn it is via examples. Like so:

```
qalc "((78*30)+(13*19))/2"
```

Fair enough, that's a normal calculation. But Qalc is capable of a lot more:

```
qalc "addDays(2014-06-18, 50)"
```

This asks Qalc to perform its internal **addDays** routine - you can guess what that does. In our case, we tell it to add 50 days onto the 18th of June, and it spits out the result:

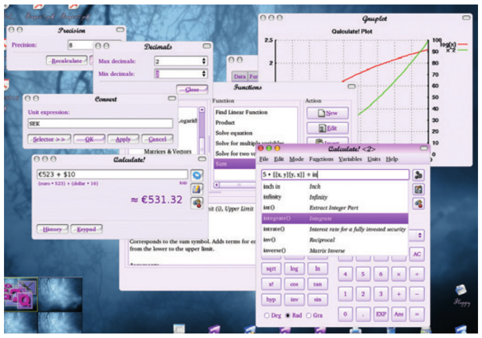
```
addDays("2014-06-18", 50) = "2014-08-07"
```

When you first run Qalc it downloads exchange rate information from the internet, so you can do:

```
qalc "500 EUR to GBP"
```

The program also understands lots of other units and conversions:

```
qalc "1300 feet to metres"
qalc "70 mph to kmh"
```



Qalc is part of Qalculate, a bigger suite of tools that include fancy GUI front-ends.

It's especially useful for doing bandwidth calculations:

```
qalc "10Gibyte / 300(Kibyte/second) to hours"
```

This tells us how many hours it will take to download 10GB at 300k/sec. Qalc's data files are stored in the **/usr/share/qalculate/** directory, so it's well worth having a nosey in there to see what other units are supported. You can even do calculations with planets and atomic elements…

Better email

"All mail clients suck. This one just sucks less." This is the motto for Mutt…

Given that most emails are plain text, you don't lose much by switching from a GUI to a CLI mail client. And indeed you gain a lot more, especially if you choose a client like Mutt. Like many of the programs we've covered in this feature, Mutt has been around for most of Linux's history – its first release was in 1995. And although it might look old-fashioned and complicated, in the right hands it's a superb program, and it's available in almost every distribution's package repositories.

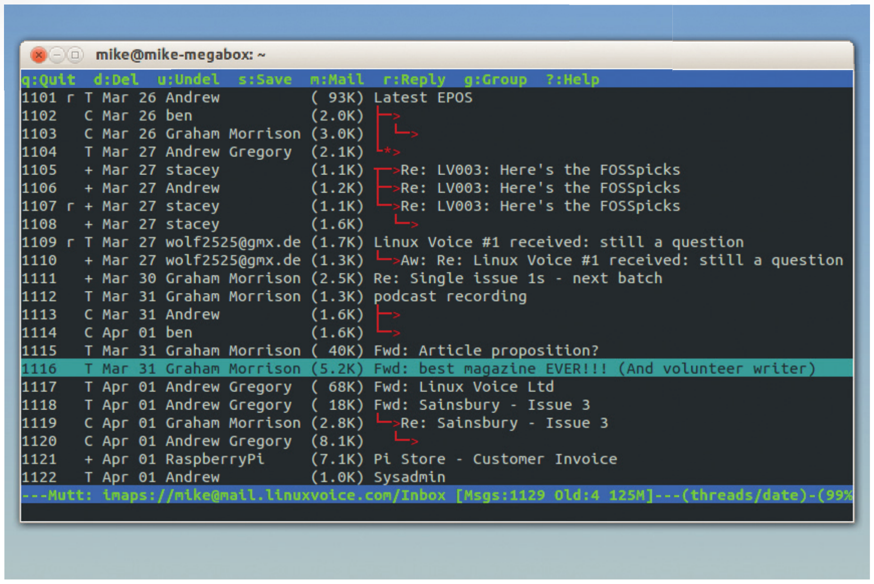
Before starting Mutt for the first time, you'll need to create a **.muttrc** file in your home directory. This contains the program's settings, and an example for connecting to an IMAP server (with SMTP for sending) is:

```
set spoolfile="imaps://user:password@server.com/Inbox"
set folder="imaps://server.com/Inbox"
set smtp_url="smtp://user:password@server.com:25"
set ssl_starttls=yes
set from="name@domain.com"
set use_from=yes
set record="=Sent"
set postponed="=Drafts"
```

Change **user**, **server.com**, **name** and **domain.com** here to match your mail server settings. If you access your mail via POP3, see the relevant section of the Mutt documentation at **http://tinyurl.com/64j7tzp**.

Now enter **mutt** to start the program, and it'll retrieve the headers for your emails. Right away you can see that Mutt does a decent job given the limits of text mode: it uses colours and highlighting effectively, and even displays threaded conversations via red arrow symbols.

To select a message, use the up and down cursor keys (or J and K in proper Vi fashion) and then hit Enter. The mail contents will be displayed – hit D to delete the mail, R to reply, and I to go back to the message list. Use **/** (forward slash) and enter a word to search for a mail, and N to repeat the search. In the main list view, tapping Q quits the program and returns you to the command line. And in most views, you'll see keyboard shortcuts displayed at the top of the screen.



```
mike@mike-megabox: ~
q:Quit  d:Del  u:Undel  s:Save  m:Mail  r:Reply  g:Group  ?:Help
1101 r T Mar 26 Andrew          ( 93K) Latest EPOS
1102   C Mar 26 ben             (2.0K) ├─>
1103   C Mar 26 Graham Morrison (3.0K) │ └─>
1104   T Mar 27 Andrew Gregory  (2.1K) └*>
1105   + Mar 27 stacey          (1.1K) ┬─>Re: LV003: Here's the FOSSpicks
1106   + Mar 27 Andrew          (1.2K) ├─>Re: LV003: Here's the FOSSpicks
1107 r + Mar 27 stacey          (1.1K) └─>Re: LV003: Here's the FOSSpicks
1108   + Mar 27 stacey          (1.6K)   └─>
1109 r T Mar 27 wolf2525@gmx.de (1.7K) Linux Voice #1 received: still a question
1110   + Mar 27 wolf2525@gmx.de (1.3K) └─>Aw: Re: Linux Voice #1 received: still a question
1111   + Mar 30 Graham Morrison (2.5K) Re: Single issue 1s - next batch
1112   T Mar 31 Graham Morrison (1.3K) podcast recording
1113   C Mar 31 Andrew          (1.6K) ├─>
1114   C Apr 01 ben             (1.6K) └─>
1115   T Mar 31 Graham Morrison ( 40K) Fwd: Article proposition?
1116   T Mar 31 Graham Morrison (5.2K) Fwd: best magazine EVER!!! (And volunteer writer)
1117   T Apr 01 Andrew Gregory  ( 68K) Fwd: Linux Voice Ltd
1118   T Apr 01 Andrew Gregory  ( 18K) Fwd: Sainsbury - Issue 3
1119   C Apr 01 Graham Morrison (2.8K) └─>Re: Sainsbury - Issue 3
1120   C Apr 01 Andrew Gregory  (8.1K)   └─>
1121   + Apr 01 RaspberryPi     (7.1K) Pi Store - Customer Invoice
1122   T Apr 01 Andrew          (1.0K) Sysadmin
---Mutt: imaps://mike@mail.linuxvoice.com/Inbox [Msgs:1129 Old:4 125M]---(threads/date)-(99%
```

The Mutt email client makes decent use of colour in the terminal, and like everything in this venerable application, these colours are highly configurable.

Interestingly, Mutt doesn't include its own editor; instead, it uses one already installed on your system. So if you reply to a mail (or hit m in the message list to create a new mail), you'll be thrown into Vim by default. But you can change the editor in your **~/.muttrc** like so:

```
set editor="nano"
```

(Or you could change that to **"emacs"**, or even **"gedit"** if you need some GUI love.)

Macho macros

So Mutt is great: it's lightning fast, looks good, and has loads of keyboard shortcuts so you don't have to mess around with the mouse. But it has some brilliant advanced features too. Instead of using **/** to search, hit L and then type a word. This is the "limit" command, and it narrows down the displayed messages to match your specifications. You can set some very specific limits:

```
~N|~d<7d
```

This tells Mutt to display only new messages (**~N**) or messages less than 7 days old (**~d<7d**). The pipe (**|**) character is used in the middle to create the "or" part. To switch back to the full message list, hit L and then type all. (Mutt's documentation has a detailed list of all the options – see **http://tinyurl.com/yzwbrur**.)

Additionally, Mutt has excellent support for macros – that is, pre-determined sequences of actions. For instance, in the message composition view, after you've entered the text in your editor and Mutt is asking if you're ready to send, you can hit the A key to attach a file. Enter a filename, hit Enter, and the file will be attached. But you could create a macro for this in your **.muttrc**:

```
macro compose \cb '<attach-file>file.txt<enter>'
```

This means: in the compose view, if the user hits Ctrl+B, the **attach-file** command will be executed. The word **file.txt** is inserted automatically, and a virtual Enter key is pressed. So Ctrl+B now does the whole action at once – useful if you frequently attach the same file to a message.

This is just one example; Mutt supports hundreds of functions that you can use in your macros, and really speed up your day-to-day work. See **http://tinyurl.com/677feer** for the full list.

"Mutt has loads of keyboard shortcuts, so you don't have to mess around with the mouse."

Better administration

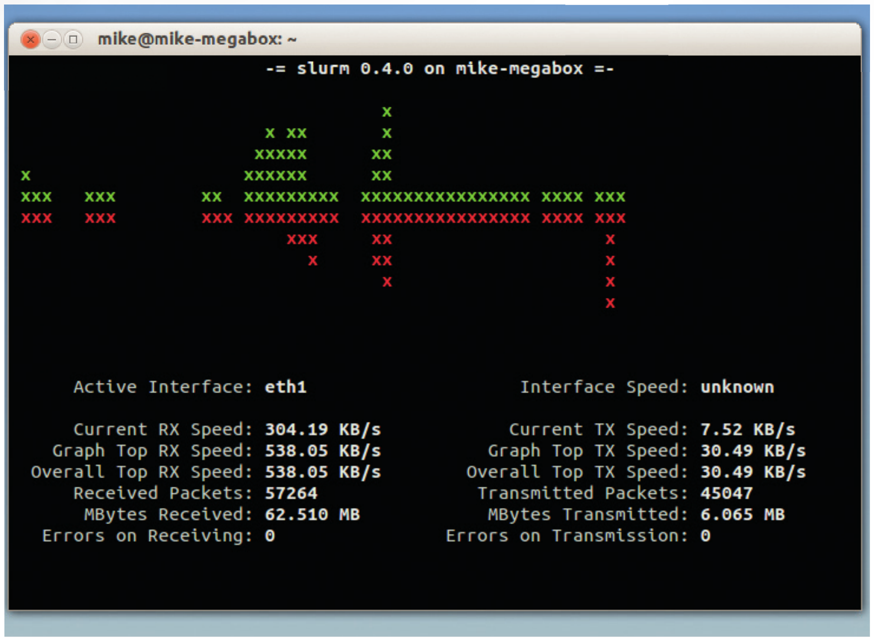
Keep tabs on your Linux boxes, wherever in the world they are.

It goes without saying that the command line is the best way to administer a Linux box. Sure, there are some decent GUI tools, but if you're working with mail, web or database servers, chances are they don't have anything graphical installed and you're logged in via SSH. Or even on your desktop Linux box, if X goes down you'll need some way to fix and monitor things.

Slurm (**https://github.com/mattthias/slurm**) is a great little network bandwidth monitor. Start it by providing the name of a network interface, eg:

```
slurm -i eth1
```

If you don't know the name of the network interface(s) on your Linux box, enter **ifconfig** for a list. Slurm displays textual information about the current data send and receive rates, along with the total number of transmitted packets and megabytes. It also shows a colourful graph of bandwidth using ASCII characters – so if you're administering multiple machines, you can leave it running in an SSH session on one, and quickly check it to see if it's being maxed out.


mike@mike-megabox: ~

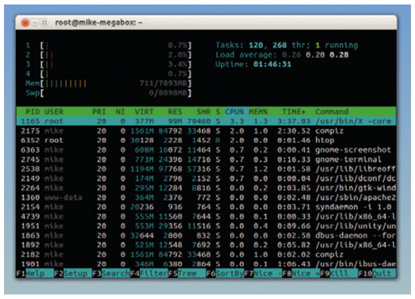
```
                       -= slurm 0.4.0 on mike-megabox =-

                                  x
                       x xx       x
                      xxxxx      xx
x                    xxxxxx      xx
xxx   xxx        xx  xxxxxxxxx  xxxxxxxxxxxxxxxx xxxx xxx
xxx   xxx        xxx xxxxxxxxx  xxxxxxxxxxxxxxxx xxxx xxx
                         xxx     xx                    x
                           x     xx                    x
                                  x                    x
                                                       x



     Active Interface: eth1                    Interface Speed: unknown

     Current RX Speed: 304.19 KB/s            Current TX Speed: 7.52 KB/s
   Graph Top RX Speed: 538.05 KB/s          Graph Top TX Speed: 30.49 KB/s
 Overall Top RX Speed: 538.05 KB/s        Overall Top TX Speed: 30.49 KB/s
     Received Packets: 57264               Transmitted Packets: 45047
      MBytes Received: 62.510 MB            MBytes Transmitted: 6.065 MB
  Errors on Receiving: 0                Errors on Transmission: 0
```

A sprinkling of ASCII art provides an at-a-glance overview of network activity in Slurm.

Monitor machine activity

Htop (**http://htop.sf.net**), meanwhile, is a souped-up version of the **top** utility. Like **top**, it displays information about currently running processes, but with much more flair and interactivity. As the program is running, hit F4 to filter processes based on name – or hit F5 to switch to a tree view, so you can see which processes were launched by other ones.

A series of bar charts at the top shows the current usage of your RAM banks and CPU cores, and you can hit F2 to configure various settings in the program. Once you've tried it, you'll never go back to using plain old **top** again.


root@mike-megabox: ~

```
  1  [|                      0.7%]     Tasks: 120, 268 thr; 1 running
  2  [||                     2.0%]     Load average: 0.26 0.20 0.28
  3  [||                     3.4%]     Uptime: 01:46:31
  4  [|                      0.7%]
  Mem[|||||||||        711/7893MB]
  Swp[                   0/8098MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1165 root       20   0  377M   99M 79460 S  3.3  1.3  3:37.03 /usr/bin/X -core
 2175 mike       20   0 1561M 84792 33468 S  2.0  1.0  2:30.52 compiz
 6352 root       20   0 30128  2228  1452 R  2.0  0.0  0:01.46 htop
 6363 mike       20   0  608M 16072 11464 S  0.7  0.2  0:00.41 gnome-screenshot
 2745 mike       20   0  773M 24396 14716 S  0.7  0.3  0:16.33 gnome-terminal
 2538 mike       20   0 1194M 97768 57316 S  0.7  1.2  0:01.58 /usr/lib/libreoff
 2149 mike       20   0  174M  2796  2152 S  0.7  0.0  0:00.04 /usr/lib/dconf/dc
 2264 mike       20   0  295M 12284  8816 S  0.0  0.2  0:03.85 /usr/bin/gtk-wind
 1360 www-data   20   0  364M  2376   772 S  0.0  0.0  0:02.48 /usr/sbin/apache2
 2154 mike       20   0 20236   936   764 S  0.0  0.0  0:03.71 syndaemon -i 1.0
 4739 mike       20   0  555M 11560  7644 S  0.0  0.1  0:00.33 /usr/lib/x86_64-l
 1951 mike       20   0  553M 29356 11516 S  0.0  0.4  0:09.66 /usr/lib/unity/un
 1863 mike       20   0 32644  2800   832 S  0.0  0.0  0:02.58 dbus-daemon --for
 1892 mike       20   0  521M 12548  7692 S  0.0  0.2  0:05.82 /usr/lib/x86_64-l
 2182 mike       20   0 1561M 84792 33468 S  0.0  1.0  0:02.02 compiz
 1901 mike       20   0  346M  6380  2864 S  0.0  0.1  1:06.43 /usr/bin/ibus-dae
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Htop is a process monitor like the standard 'top' command, but literally a jillion times better.

After all of the command line goodness of the last seven pages, wouldn't it be great if you could record your favourite tricks and share them with others? You could use some screen recording software and upload the results to YouTube, but a more elegant solution is Asciinema (**www.asciinema.org**). You can get it on Debian/Ubuntu like so:

```
sudo apt-get install python-pip
sudo pip install --upgrade asciinema
```

(The package might have another name than **python-pip** in other distros.) Now enter **asciinema rec**, do some work, and type **exit** when you're done. Asciinema will offer to automatically upload the recording of your session to its website, and provide you with an URL you can then share with others. For instance, here's a recording of us demonstrating the mighty power of Figlet: **http://asciinema.org/a/8746**

Better downloads

The download dialogs included in web browsers are very limited, and although more featureful standalone GUI alternatives exist, sometimes it's best to go straight to the CLI. Aria2 is arguably the best command line download manager in existence, supporting a gigantic range of features and options. For instance, say you want to grab an ISO image that's hosted on two servers, and they're both rather slow:

```
aria2c -s2 http://foo.com/blah.iso http://another.com/blah.iso
```

Here Aria2 downloads one half of the file from **foo.com**, and the other half from **another.com**, simultaneously, so you get the file much more quickly than you would using a single connection to one server.

It's possible to limit download speeds, so adding **--max-download-limit=100K** to the command line will restrict Aria2 to using 100KB/second of your bandwidth. And you can even tell it to give up if a connection becomes too slow:

```
--lowest-speed-limit=10K
```

(So if the bandwidth drops to less than 10KB/sec, Aria2 quits.) Other useful options include **--on-download-complete=command**, which automatically performs a command after a file has been downloaded. There's also the **--on-download-error** argument, which is handy for dealing with connection failures.

See Aria2's website at **http://aria2.sf.net** for the full documentation – it's immensely powerful when you include it in Bash scripts.