

Mastering Linux, part 5

Having demystified the X Windows desktop, *Jarrod Spiga* returns to the command line console to explain some of its more advanced functions.

Part 3 of this series (*APC* February, page 98) provided an introduction to the Linux command line console, demonstrating how to work with the file system. But the command line is useful for so much more than managing data.

In the days before graphical user interfaces (GUIs), everything had to be done via the command line. Even applications such as word processors, where WYSIWYG interfaces are now taken for granted, once ran in a command line environment. While using the GUI is an aesthetically pleasing way of doing things, there are many occasions where it may not be available. Knowing how to perform tasks via the command line is extremely handy and more efficient.

TEXT EDITING

Editing text files is one of the most fundamental tasks you can perform under Linux for a multitude of reasons. For instance, almost all configuration and script files under Linux are text files of some description.

The lightweight vi application is one of the more ubiquitous text editors under Linux and Unix and is ideal for creating and editing text files quickly. It's very rare to come across a Linux installation that doesn't include vi, even if a minimal OS installation has been performed.

Start the vi editor with the `vi` command. Launching vi in this manner will display an almost

blank screen, allowing the creation of a new file. However, most users find it easier to launch vi by passing the name of the file they wish to create or edit as an argument to the command. For example, to create or edit a file named "readme.txt" in your home directory, you could use:

```
vi ~/readme.txt
```

If you're editing an existing file, vi will display the contents of the beginning of that file on all lines of the screen except the last. This line is reserved for information about what's happening within the application. Upon loading a file, the bottom line will display the name of the file, how many lines and characters it contains, the line and column position of the cursor and any other detail about the file that it can provide (for example, if you're at the top of the file).

If you're creating a new file, vi won't display much. A tilde (~) character will appear at the beginning of each line of the screen indicating that all lines are blank. Much like when editing a file, the bottom line will also display the name of the file that you are editing (and the fact that it's a new file), the line and column position of the cursor, and the status (in this case, that all of the file is being displayed).

MODES OF OPERATION

When vi is loaded, it runs in command mode — instead of being included in the document, any text that you enter will appear at the bottom of the screen and will be interpreted by the application as a command. Command mode allows you to save the changes you've made to the document, perform searches within the document, browse through the document (using the Arrow, Page Up and Page Down keys) and other functions.

This is useful for displaying the text inside a file (even if that file is empty), but what if you want to perform some editing tasks? In order to make changes, you'll need to put vi into insert or replace mode.

As you'd expect, insert mode allows you to insert text into the document (an insert cursor), while replace mode allows you change the text in the document (an overwrite cursor). Switch to insert mode by pressing the Insert key once while in command mode. Hitting Insert again will switch you to replace mode. Alternatively, hitting I in command mode will switch you to insert mode, while entering R in command mode will take you straight to replace mode.

Skill level

Beginner

Requirements

An installation of Linux (Fedora Core 3 was used for this article).

Time to complete

3 hours (approx)

```

jspsiga@fedora:~
File Edit Options Buffers Tools Minibuf Help
Again, I'm just entering random text for the fun of it.

That looks like a menu above, doesn't it. D'oh! I can't click on the menus to s\
elect any options. Damm. Well, looks like you'll just have to remember to uyse \
the F10 key.

Having to use the F10 key can be problematic if you are on a remote system that\
doesn't let you put through F10, hence why I prefer to use vi for my editing, \
but this Linux article is supposed to make things easy. So even though I'm a pu\
rist, I'll explain how to use emacs.

--uuu:**-F1  readme.txt      (Text)--L1--Top-----
Press PageUp Key to reach this buffer from the minibuffer.
Alternatively, you can use Up/Down keys (or your History keys) to change
the item in the minibuffer, and press RET when you are done, or press the
marked letters to pick up your choice.  Type C-g or ESC ESC ESC to cancel.
In this buffer, type RET to select the completion near point.

Possible completions are:
f=>File          e=>Edit
o=>Options      b=>Buffers
t=>Tools        h=>Help
--uuu:**-F1  *Completions*  (Completion List)--L1--All-----
Menu bar (up/down to change, PgUp to menu): f=>File
    
```

1 The F10 key brings up the menu in emacs. From there, select the letter corresponding to the menu option you want.

► In either mode you can enter text and it will appear inside the document where the cursor is located. When you reach the end of a line on the screen, your text will start to appear on the next line.

But be careful — a carriage return hasn't actually been entered here, so you're still typing on the first line of your document. In other words, vi doesn't have a word wrapping function, and you'll need to press Enter every time you intentionally want to start a new line. If you fill the entire screen without pressing Enter, your document will be saved as a file with one very long string of text.

COMMAND MODE

When you've finished changing your document, press the Escape key to return to command mode. You can perform some editing functions in command mode, you just can't enter text into the document. Here's a list of frequently used commands:

Command	Action
Left arrow+h	Moves the cursor one character to the left
Right arrow+l	Moves the cursor one character to the right
Up arrow+k	Moves the cursor up one line of text (not one line on the display)
Down arrow+j	Moves the cursor down one line of text
dd	Deletes the line that the cursor is on
d<number>	Deletes the number of characters under and to the right of the cursor
Insert, i	Insert mode
r	Replace mode
/<string>	Find the next occurrence of the search string in the file
:<number>	Go to line number
:w	Write file
:q / :q!	Quit vi / Quit vi with override
:help	Displays help screens

SAVING AND EXITING

Saving the changes you've made is fairly straightforward by entering `:w` from the command mode. If you'd like to save your changes under a different filename, specify that filename after the write command (for instance, `:w saveas.txt`). However, exiting from vi isn't always so simple.

In order to prevent accidental data loss, vi won't exit unless you've saved any changes that you've made to your file. If you

save your file first, vi will allow you to exit when the `:q` command is issued. If you haven't saved your work, it will tell you.

But what happens if you've made some changes to a file, but you want to get out of vi without committing to them? In this case, you'll need to override vi's checks by adding an exclamation mark to the end of the command (`:q!`). This will force vi to exit, discarding any changes that you've made since your last save operation.

THE EMACS TEXT EDITOR

The advantage to using vi is that it adopts the KISS approach (Keep It Simple, Stupid!). This is great for making quick changes to text files, but its minimalist feel can be frustrating.

Another frequently used text editor is emacs — an extensible and programmable editor that seems to have limitless functionality. This extensibility makes it much easier to use, especially for people who are new to Linux and Unix. However, there's a lot more to know about emacs, from how to use it to edit files, to programming using the emacs LISP language.

Launch the emacs editor in much the same way as you launched vi — execute the `emacs` command with the name of the file you'd like to edit as an argument:

```
emacs readme.txt
```

Depending on the configuration of your Linux PC, emacs may take a few seconds to start up. Remember, it's much larger and more complex than vi, so it takes longer to load into memory.

Once emacs loads up, you'll see a screen similar to that of vi, with two differences. The first is what appears to be a menu across the top of the screen. And secondly, the status bar at the bottom of the screen appears over two lines and contains much more detail.

Emacs is easier to use than vi, you don't have to be concerned about which mode you're in. If you start typing, your text will go straight into the document. Also, the Arrow, Page, Home and End keys all operate as you'd expect them to. But there are also some similarities. Emacs doesn't perform word wrapping either, so if you don't enforce your carriage returns you'll be left with a big, single-lined file.

DECEPTIVE APPEARANCES

While the line at the top of the emacs editor

looks like a menu, it isn't. However, if you hit the F10 key, the emacs screen will split in two, revealing the menu options you can select with a few key presses in the lower half. For instance, hitting the F key after bringing up the menu in this fashion will display the options you'd normally expect to see under a file menu. If you then hit the S key, all changes you've made to the file will be committed to disk.

1 This isn't the only way to access commands from within emacs. In a similar way to Windows applications, pressing keys in conjunction with the Control key can perform a variety of tasks. For instance, Ctrl+X followed by Ctrl+S will save the changes to your file, or Ctrl+X followed by Ctrl+C will exit out of emacs.

Armed with this information, you should now be able to perform most editing functions under both vi and emacs. There are literally thousands of other uses for emacs in particular, so if you're the adventurous type, feel free to explore the emacs manual online at www.delorie.com/gnu/docs/emacs/emacs_toc.html.

GROUPING FILES ON THE COMMAND LINE

Part 3 of this series demonstrated a number of commands that could be performed to help you manage the data on the Linux file system. When these commands were entered, you would have only used one file with each command. However, it's possible to use a number of wild-card characters to group many filenames for processing by a single command at any given time.

For instance, let's assume you've used vi and/or emacs to create a small library of text files in your home directory. And further that you want to move these files to a more manageable location — a docs directory under your home directory. Using the knowledge gained a couple of months ago, you could enter in the following commands to achieve the task:

```
mkdir docs
mv document1.txt docs
mv document2.txt docs
mv document3.txt docs
```

The same result can be achieved using:

```
mkdir docs
mv document1.txt document2.txt
document3.txt docs
```

While this method is a little quicker, you still have to type out the complete name ►

► of each document — frustrating if you have to move a lot of files.

The solution is to use file-grouping techniques. The most common way is by substituting an asterisk (*) for a section of the filename, leaving the components of the filename that are common to all of the files you'd like to perform the command on. Enter:

```
mkdir docs
mv *.txt docs
```

When the command shell processes the second line, it substitutes *.txt with all of the files that end with .txt. Going back to the original scenario, the same result could be achieved with:

```
mv docu* docs
```

This command makes the shell replace the docu* file grouping with the file names of all files that begin with docu, moving all of those documents to the docs directory.

DIRECTORIES INCLUDED

Did you notice that doc* wasn't used for the grouping? If it was, the mv command would have returned a warning, telling you that you won't be able to move the docs folder inside itself because the docs directory also fits this group mask. Files and directories can be grouped using the asterisk (along with any other grouping symbol), which is why the "u" was added.

As hinted at earlier, the asterisk character isn't the only one that can be used to group files. Below is a table containing the most frequently substituted characters, followed by another table containing sample commands and their effects:

Pattern	Effect
*	Matches any amount of any characters
?	Matches one of any character
[a-d]	Matches a single character in the range of a through to d. i.e. a, b, c or d
[abc]	Matches a single character from the list provided

Command	Effect
rm *.txt	Deletes all files ending with ".txt"
Rm d*.txt	Deletes all files beginning with "d" and ending with ".txt"

Rm document?.txt	Deletes all files starting with document, with a single character following, and ending with .txt
Rm [b-f]rat.txt	Deletes brat.txt, crat.txt, erat.txt and frat.txt
Rm [jkl]*	Deletes all files beginning with j, k or l
Rm ?	Deletes all files with one character in their file name

EXPANSION CONTROL

So, what happens if you want to create a file or directory that contains an asterisk, question mark or bracket? Surprisingly, these characters aren't reserved and can be used in file and directory names, although it's a little tricky.

In order to use these characters, you need to quote your proposed file name inside either single or double quotation marks. For example:

```
Touch 'why?'
```

Be wary of using these characters in filenames because you'll have to remember to encapsulate the filename in quotes every time you refer to it.

LOOKING FOR STUFF

In the past, you've used the ls command to generate listings of all files in a directory. If you needed to search the file system for a particular file, you could generate a directory listing for all of the locations where you suspect the file may be, but this would be tedious and you still may not find it. What's more, as your file system grows you'll have to manually look through thousands of files in hundreds of different directories. It becomes like looking for a needle in a haystack.

Because Linux has many small tools that perform small functions, it comes as no surprise that there are a couple of search tools available. The find and locate tools allow you to track down files, but work in very different ways.

LOCATION, LOCATION

Your Linux system is scheduled to perform maintenance on the file system and its applications on a regular basis. One task it performs during this maintenance is to update a database of all of the files on your system and where they are located.

The locate command will search this database for the filename or filenames (perhaps using wild-cards) you nominate,

and return the full path to all files matching the supplied criteria. For example, to search for all files ending with .txt on your system, use the following command:

```
locate *.txt
```

FINDERS KEEPERS

The find command effectively performs the same function but uses a completely different methodology. When you run this command, it manually searches the file system for results that match your criteria and then displays them.

The syntax for the find command is somewhat more complicated — you need to supply a search location, the search criteria, and an argument stating what you want to do with the output of the command. In order to get the results printed to your screen in the same way as the locate command, use the following syntax:

```
find location1 [location2 ...] -name criteria
-print
```

The find command will then search the location1 directory and all subdirectories (as well as location2, if specified) for files that match the criteria. Finally, the result is printed to the screen. To search for all .txt files in your home directory using the find command, enter:

```
find ~ -name *.txt -print
```

WHICH ONE TO USE?

Because the find command manually searches the file system, it takes significantly longer to run. However, because the locate command refers to a database that's only updated daily (although this can easily be changed), the find command is much more accurate.

It's always best to use the right tool for the right task — even though it's possible to use a pair of pliers to tighten a nut and bolt, you wouldn't do so because using a set of spanners is so much easier. For the same reasons, locate should be used when you need to quickly search large sections of the file system, while find should be used when you require accurate results on a small section of the file system.

SMOKIN' PIPES

If you entered the locate command on your system, a long list of text files would have quickly scrolled past your view. At the

► console, there's no quick or easy way to scroll up and see what's scrolled off the screen, so how do you get the chance to read what the locate command puts out before it disappears?

Another tool found on most Linux systems is `more` (on other systems, a similar tool called `less` also exists though both effectively perform the same task). When used by itself, the `more` tool does nothing because it takes the input supplied and displays one screen at a time. While `locate` can display a lot of text on the screen, `more` can take that text and display it one screen at a time. To get `locate`'s output and use it as the input to the `more` tool, you need to use a pipe:

```
locate *.txt | more
```

2 When run, the first screen of text generated by the `locate` command is displayed, along with a prompt at the bottom of the screen. Pressing the space bar will display the next page of text.

Another tool to pipe data into is the `grep` command. `Grep` is a filter tool that reduces the amount of data that is output from a command. To demonstrate, look at the following syntax:

```
locate * | grep .txt
```

In this example, the `locate` command returns a list of the locations of all files on your system. As you can imagine, this list is extremely long. However, the output is being piped into the `grep` tool, which applies a filter on all of the lines output by `locate`, and only displays the lines containing the `.txt` string. In essence, it's generating a list of `.txt` files on the file system in a similar manner to the `locate *.txt` command.

REDIRECTING THE OUTPUT

Piping the output to the `more` command is just one way you can manage the flow of information across your screen. Another is to redirect the output of the command so that it's placed inside a text file instead of being displayed on the screen.

To do this, simply use a greater-than sign (`>`) after the command, followed by the name of the file you want to put the output of the command in. For instance:

```
locate *.txt > textfiles.txt
```

The `locate` command takes a while to run,

```
admin@berlin:/home/admin
[root@berlin admin]# locate *.txt | more
/var/www/html/phpMyAdmin-2.5.6/Documentation.txt
/var/www/icons/small/README.txt
/var/analog-5.32/lang/README.txt
/var/analog-5.32/lang/bgdesc.txt
/var/analog-5.32/lang/bgmdesc.txt
/var/analog-5.32/lang/bradesc.txt
/var/analog-5.32/lang/brdesc.txt
/var/analog-5.32/lang/esadesc.txt
/var/analog-5.32/lang/esdesc.txt
/var/analog-5.32/lang/fradesc.txt
/var/analog-5.32/lang/frdesc.txt
/var/analog-5.32/lang/itadesc.txt
/var/analog-5.32/lang/itdesc.txt
/var/analog-5.32/lang/jpedesc.txt
/var/analog-5.32/lang/jpjdesc.txt
/var/analog-5.32/lang/jpsdesc.txt
/var/analog-5.32/lang/jpudesc.txt
/var/analog-5.32/lang/nladesc.txt
/var/analog-5.32/lang/se2adesc.txt
/var/analog-5.32/lang/nldesc.txt
/var/analog-5.32/lang/ptadesc.txt
/var/analog-5.32/lang/ptdesc.txt
--More--
```

2 Piping output to the `more` command allows you to go through the output one screen at a time.

but when it completes, you'll be given another prompt.

No output is displayed on the screen because it has all been redirected to the file that you specified. Once the command has completed running, load up the `textfiles.txt` file in your editor (`vi` or `emacs`) and you should have a listing of all text files on your system.

ANSWER OR ARGUMENT?

Another way to utilize `more` than one command at once is command substitution, where the output from one command is used as part of the input to another.

Imagine you have an `images` directory located under the home directory. Not only are there a few hundred JPEG images strewn across a couple of dozen subdirectories, there are also text files describing each image. To quickly make an image gallery containing every JPEG file in every subdirectory, use the following commands:

```
mkdir gallery
ln -s $(locate '*.jpg' |grep /home/jspiga/
images/) gallery
```

The first line creates a directory named `gallery`. When the second line is executed, the first command that is run is the `locate` command, which generates a listing of all files on the system matching the argument — in this case, all `jpg` images on your file system.

The output from this command is then piped into the `grep` command, which filters

the output from `locate` to leave you with a list of JPEGs under the `images` directory in the home directory. Lastly, the output is then substituted as an argument to the `ln` command, which creates symbolic links when used in conjunction with the `-s` switch.

When run, this command creates a shortcut to all JPEG images under the `images` directory. These shortcuts are all stored under the `gallery` directory. But why shortcuts? Why have two copies of a file on your hard disk when you can just link to the first and save space?

In order to use a command substitution, prepend your substituted command with `$(` (and append it with `)`.

PAUSING PROCESSES

One of the advantages to using a GUI is that you can switch between running tasks by simply changing which window is in focus. You can also switch tasks under the command line console.

To pause a process from the command line, the `Ctrl+Z` key combination can be used. For example, load up a file in the `vi` editor, make a few changes and hit `Ctrl+Z`. You'll be taken back to the command prompt and a line similar to the following will appear above the cursor:

```
[1]+ Stopped vim sample.txt
```

This tells you that the `vi` process has been stopped for the time being (more technically, it's sleeping — still active in ►

▶ memory, but awaiting instructions). The number between the brackets tells you the job number that's been assigned to the task — in this case the vi task is job number 1. This job number is critical to switching between applications.

Load up another vi editor, make some edits and again, hit Ctrl+Z. This time, the following appears:

[2]+ Stopped vim sample2.txt

As you can see, this vi task has been given a job number of 2. In order to get a list of all jobs currently running or suspended, use the `jobs` command. This will show similar detail to the two lines above.

To resume an application, simply execute the `fg` (foreground) command and pass a percentage sign, followed by the job number to indicate which task you want to resume:

`fg %1`

Your initial vi session should pop up again and you can resume where you left off.

THE RUNNER IN THE BACKGROUND

A short time ago, you would've run the `locate` command while redirecting the output to a text file. Depending on the speed of your Linux PC, this process could have taken a couple of minutes to run. Or you could hit Ctrl+Z while the command is executing and resume it at a more convenient time. That way, you won't have to sit around waiting for the task to finish before running your next command.

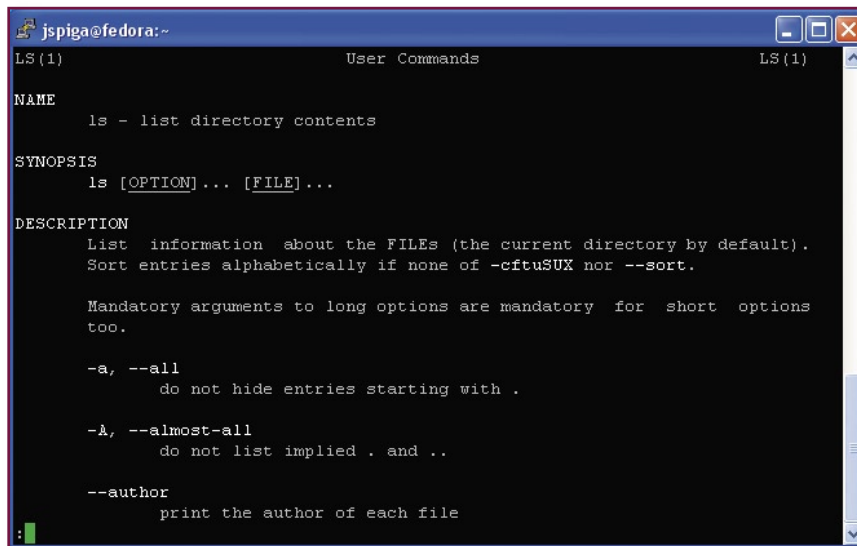
Lastly, you could run the command in the background by placing an ampersand at the end of your command:

```
locate *.txt > textfiles.txt &
```

The ampersand tells the shell to run the command in the background. Executing the command would have returned something like this:

[3] 3807

The number in brackets relates to the job number given to the task. Indeed, if you run the `jobs` command, you should see the task running in the background. The latter number refers to the process



```
jspiga@fedora:~
LS(1)                                User Commands                                LS(1)
NAME
ls - list directory contents

SYNOPSIS
ls [OPTION]... [FILE]...

DESCRIPTION
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of -cftuSUX nor --sort.

Mandatory arguments to long options are mandatory for short options
too.

-a, --all
do not hide entries starting with .

-l, --almost-all
do not list implied . and ..

--author
print the author of each file
```

3 The man pages look similar to the vi editor, but they can't be edited.

identification number (pid) of the task. We'll cover pids in greater detail in a future instalment of this series.

PUSHING IT BACK

Assume for a minute that you did run the `locate` command in the foreground and then hit Ctrl+Z to suspend it. You don't actually have to wait until you go away from the system to let the command run without interfering with your command line usage — simply make the process continue to run in the background using the `bg` (background) command:

`bg %3`

When any background task is completed, a message similar to the following will appear on your screen after you've entered the next command:

[3] Exit 1 locate *.txt > textfiles.txt &

Note that the ampersand appears at the end of the command, even if you didn't launch the process in the background. This just tells you that the task ran in the background.

KILLING YOUR JOB PROSPECTS

If you suspend a job and then decide you don't need to continue working on it (or one of your jobs crashes or stops responding), you can always cancel the job. Not surprisingly, the `kill` command allows you to do this. So, in order to terminate the other vi task that was running, enter:

`kill %2`

Note that the application will be removed from memory in the state it's in — you won't be prompted to save your data before killing a process.

RTFM: READ THE FREAKIN' MANUAL

If you haven't encountered the RTFM abbreviation before, you will see it whenever you ask for help in relation to Linux. While Fedora Core 3 doesn't come with printed manuals, there's a wealth of knowledge in the electronic `man` (manual) pages.

For instance, to find out more information about the `ls` command, run:

```
man ls
```

E A screen similar to a vi screen will appear. The man page for each command explains in detail exactly what it does, its syntax and what arguments can be used with it. To scroll to the next page in the document, hit the space bar. The B key will take you up one page. Entering a forward slash (/) followed by a search string will search the document for that string, while the Q key will exit the man page viewer. **BTB**

Next month . . .

In next month's *Mastering Linux* guide, we'll head back into X Windows and take you through a number of the applications in day-to-day Linux use.