Searching Text Files with grep

Digital Fishing for Text

Scholars in the Middle Ages would have sold their souls for the literary treasures that abound on the Internet today. Now they simply clutter up our hard disks a result of indiscriminate downloading. Just how do you go about finding a passage of text in your digital datastore? The Shell command, grep, can help you hunt down that elusive quote.

BY ELISABETH BAUER

If you can't remember things, you should at least put them on your computer." That is not such a bad idea, but in contrast to human memory, which will normally retrieve information stored in it, quite reliably – with the exception of final exams of course – it is not always that easy to find information on a hard disk. If you forget the filename and where you stored the file, you might find yourself wasting a lot of time searching through directories. Even if you know exactly which file contains information you are looking for, this may be of little use in the case of longer text files.

The shell command, *grep*, which locates text patterns within texts, is useful in both cases. In the most simple case, you call *grep* with the search key and the file to be searched. Grep will respond by outputting all the lines in the specified file that contain the search key.

Imagine you wanted to search the King James Bible for instances of "Garden of Eden". To do so you would type

```
grep Eden bible.txt
```

in the shell, and *grep* would output the appropriate passages from the Bible. If the search key contains spaces, you need to enclose it within quotes. You need to take some care and watch out for pitfalls with special characters: *, ?, and ! have a special meaning for the shell. Another group of characters (., *, ^, $, and \) is not interpreted at face value by grep. Instead, the tool will assume a regular expression. The upside is that you can construct powerful and complex queries, although you might want to avoid using these characters with grep until you become more accustomed to the tool.

If you are not sure which file contains a text passage you are looking for, you can call *grep* with a wildcard (*). The Gutenberg Project's version of Herman Melville's "Moby Dick" comprises of a collection of text files.

```
grep white moby.*
```

will show all the occurrences of the word white in Melville's whaling masterpiece (see Figure 1). The asterisk means any letters. The shell will replace this expression with the names of any files in the current directory with the *moby.* prefix.

If the files you want to search for is distributed across multiple directories, you can use the *-r* option to tell *grep* to search a folder recursively:

```
grep -r white Melville/works/
```

will search the *works* folder and any subdirectories below it for *white*.

The Trio – ps, grep, and kill

grep is not only useful for philosophical and theological text searching, but can also be used in combination with other shell commands. If a command produces a lot of text output, you can use the command followed by a pipe character (|) and *grep searchkey* to filter the output, retaining only the parts you are interested in. A typical case where you would

Grep Overview

Command	Action
grep pattern file	searches *file* for *pattern*
*grep pattern *.htm*	searches all files in the current directory that end with the *.htm* suffix
grep -r pattern folder	performs a recursive search in *folder* and its subdirectories
grep -i pattern file	ignores case
grep -A n	outputs *n* lines after the line containing the searchkey

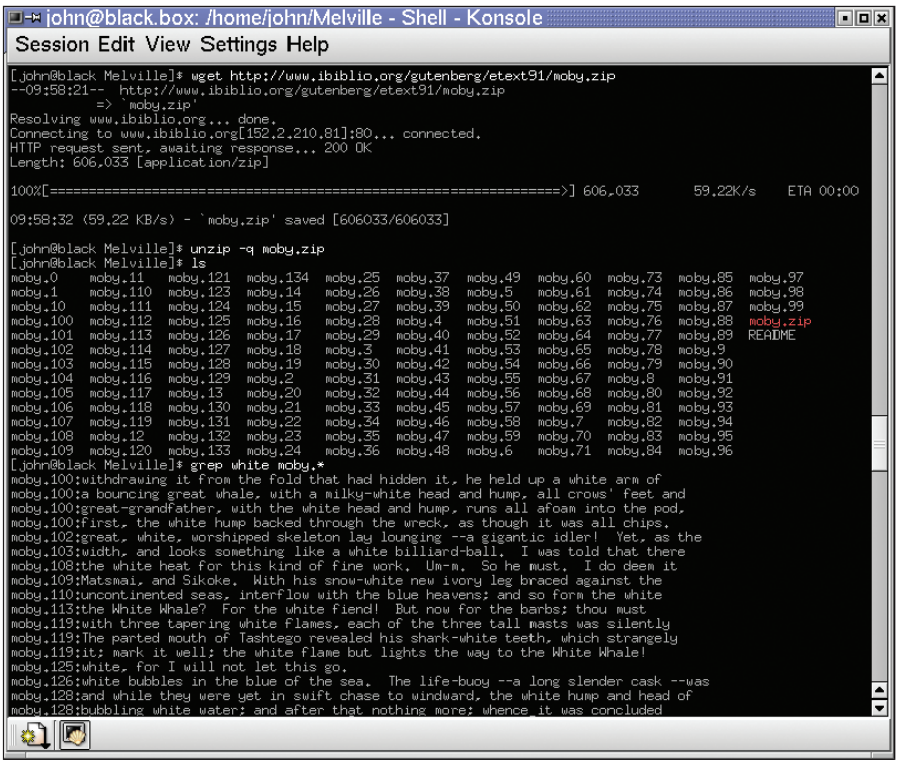

Figure 1: Melville's "Moby Dick" is subdivided into a number of a text files. The "moby.*" wildcard tells grep to search through all the files in the directory

use grep in this way is killing a program that has crashed in the shell.

The *ps ax* command displays the processes. You can use a pipe to pass this output to *grep*, to apply a filter for the program you are looking for. Searching for Mozilla will return the following results, for example:

```
> ps ax | grep mozilla
2500 ?      S  1:40 /usr/lib/➥
mozilla-1.3/mozilla-bin
5645 pts/4  S  0:00 grep mozilla
```

Grep returns two matches containing Mozilla – not only the active browser, but also itself. The part we are interested is output at the start of each line: the process ID, which we need to *kill* the program in the shell. We can now type *kill 2500* to kill the hanging Mozilla browser.

As shell gurus are notoriously lazy, we will want to find a way to avoid typing this command each time we need it: in other words, we need an *alias*. Alias definitions should be saved in *.bashrc* in your home directory. This file is run each time you open an interactive shell. Use your favourite editor to open the file, this could be *kwrite ~/.bashrc &* or *vi ~/.bashrc* in the shell, depending on your preferences. As *vi* is non-trivial, let's have looked at some simple commands. Typing *G* tells *vi* to go to the end of the file. You can then type *o* to toggle the editor to input mode – in contrast to the *a* and *i* commands, this tells *vi* to start and place the cursor in that line.

You can now enter your alias in the last line of *.bashrc*. Instead of *pss* you could use any name you find easy to remember – but avoid overwriting the name of an existing command:

```
alias pss="ps ax |
grep"
```

Simple as that! You can then press *[Esc] ZZ* or *[Escape] :wq* to store the file and quit *vi*. If you want to use your new alias in the current shell, you need to parse the configuration file. To do so, type:

```
. ~/.bashrc
```

You can now use the *pss program* command to search for an active program.

An Address Book for Purists

Grep is extremely flexible. One of my own favorite applications for grep is a simple address book.

All you need for this, are the *grep*, *alias*, and *cat* commands, plus a text file where you store the names, phone numbers, email, and snailmail addresses of your friends, acquaintances, and relatives. An address entry might be as follows:

```
Charly Penguin
+12345 678
tux@linux.org
1 South Pole Road
Tux Village, Antarctica
```

Now save the file as *addressbook* in your home directory, and add the following alias to *.bashrc*:

```
alias tel="cat ~/addressbook | ➥
grep -i -A 4"
```

The *cat* command outputs the *addressbook* file. The pipe (|) sends this output to *grep*. The *-i* ("case-insensitive") option ensures that the search will not be case-sensitive. And finally, *-A 4* tells *grep* to return the next four lines after the line with the match.

Again

```
. ~/.bashrc
```

will re-parse your configuration file. In future, you can type *tel name* in the shell to retrieve the address of the person you are looking for – now that's what I call quick!

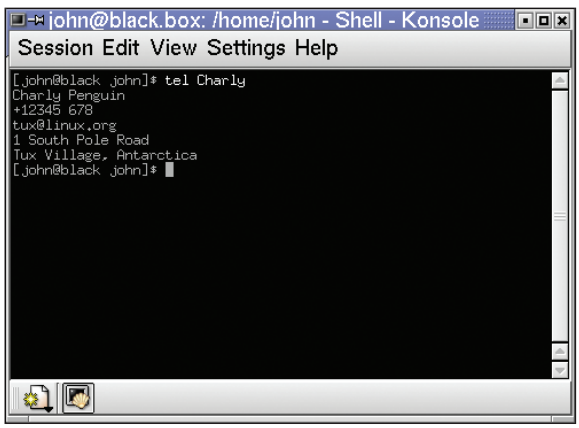

Figure 2: You can use grep to quickly construct a shell-based address book. The "tel" alias outputs the address entry that matches the search key