Searching in Files

# Searching for Text in ASCII Files

If you are looking for text within a file, use the grep command.

grep pattern file - Search for pattern in file.

grep -v pattern file - Invert match. Return lines from file that do not match pattern.

$ cat secret

site: facebook.com user: bob

pass: Abee!

$ grep user secret user: bob

$ grep o secret site: facebook.com user: bob

$ grep -v o secret pass: Abee!

Here are some more common options to use with grep.

grep -i - Perform a search, ignoring case.

grep -c - Count the number of occurrences in a file.

grep -n - Precede output with line numbers from the file.

$ grep User secret

$ grep -i User secret user: bob

$ grep -ci User secret 1

$ grep -ni User secret 2:user: bob

# Searching For Text in Binary Files

If you run grep against a binary file, it will simply display whether or not that information was found in the file, but it will not display the surrounding text. To look at textual data within a binary file use the strings command.

strings file - Display printable strings in binary files.

$ grep -i john BlueTrain.mp3 Binary file BlueTrain.mp3 matches

$ strings BlueTrain.mp3 | grep -i john John Coltrane

John Coltrane

$

# Pipes

You will notice that two commands have been chained together with a vertical bar, also known as the pipe symbol. The pipe ( | ) means take the standard output from the preceding command

and pass it as the standard input to the following command. If the first command displays error messages those will not be passed to the second command. Those error messages are called "standard error" output. You will learn how to manipulate standard error output in the "Redirection" chapter.

Also notice that in the first occurrence of the grep command the format of grep -i pattern file was used. In the second, the format of grep -i pattern was used. In the first format the input for grep came from file . In the second format the input for grep came from the preceding command via the pipe.

If you run strings BlueTrain.mp3 a lot of text will be displayed on the screen. Instead of letting that text pass you by, you can feed it to grep -i john using a pipe. The result, as you can see, is that 'John Coltrane' was found twice in the strings BlueTrain.mp3 output.

Pipes aren't limited to just two commands. You can keep chaining commands together until you get the desired result you are looking for. Let's feed the output from grep to head -1 to limit the output to just one line.

$ strings BlueTrain.mp3 | grep -i john | head -1 John Coltrane

$

Let's say you only want to display the second word of the above output. You can use the cut

command to accomplish that goal.

cut [file] - Cut out selected portions of file. If file is omitted, use standard input.

cut -d delimiter - Use delimiter as the field separator.

cut -f N - Display the Nth field.

To extract 'Coltrane' from 'John Coltrane', use a space as the delimiter (-d ' ' ) and print the second field (-f 2 ). The space was quoted since spaces are typically ignored by the shell. Single quotes or double quotes work the same in this situation.

$ strings BlueTrain.mp3 | grep -i john | head -1 | cut -d ' ' -f 2 Coltrane

$

You will find that there are many small commands that do just one thing well. Some examples are awk , cat , cut , fmt , join , less , more , nl , pr , sed , seq , sort , tr , and uniq . Let's take an example using some of those commands and chain them together with pipes.

The /etc/passwd file contains a list of accounts on the system and information about those accounts. In this example, the goal is to find all of the users named "bob" listed in the

/etc/passwd file and print them in alphabetical order by username in a tabular format. Here is one way you could do that.

$ grep bob /etc/passwd bob:x:1000:1000:Bob:/home/bob:/bin/bash bobdjr:x:1001:1000:Robert Downey:/home/bobdjr:/bin/bash bobh:x:1002:1000:Bob Hope:/home/bobh:/bin/bash bobs:x:1003:1000:Bob Saget:/home/bobs:/bin/bash bobd:x:1004:1000:Bob Dylan:/home/bobd:/bin/bash bobb:x:1005:1000:Bob Barker:/home/bobb:/bin/bash

$ grep bob /etc/passwd | cut -f1,5 -d: bob:Bob

bobdjr:Robert Downey

bobh:Bob Hope bobs:Bob Saget bobd:Bob Dylan bobb:Bob Barker

$ grep bob /etc/passwd | cut -f1,5 -d: | sort bob:Bob

bobb:Bob Barker bobd:Bob Dylan bobdjr:Robert Downey bobh:Bob Hope bobs:Bob Saget

$ grep bob /etc/passwd | cut -f1,5 -d: | sort | sed 's/:/ /' bob Bob

bobb Bob Barker bobd Bob Dylan

bobdjr Robert Downey bobh Bob Hope

bobs Bob Saget

$ grep bob /etc/passwd | cut -f1,5 -d: | sort | sed 's/:/ /' | column -t bob Bob

bobb Bob Barker

bobd Bob Dylan bobdjr Robert Downey bobh Bob Hope

bobs Bob Saget

The above example shows the step-by-step thought process of how to go from one set of output and pipe it as the input to the next command. If you need to perform this action often you could save the final command for later use. As you can see, this simple concept of piping makes Linux extremely powerful.

**Pipe Output to a Pager**

Another common use of pipes is to control how output is displayed to your screen. If a command produces a significant amount of output it can scroll off your screen before you have the chance to examine it. To control the output use a pager utility such as more or less . You've already used those commands directly on files, but keep in mind they can take redirected input too.

$ grep bob /etc/passwd | less bob:x:1000:1000:Bob:/home/bob:/bin/bash bobdjr:x:1001:1000:Robert Downey:/home/bobdjr:/bin/bash bobh:x:1002:1000:Bob Hope:/home/bobh:/bin/bash bobb:x:1005:1000:Bob Barker:/home/bobb:/bin/bash

...

$ ls -l /usr/bin | less total 62896

-rwxr-xr-x 1 root root

-rwxr-xr-x 1 root root

-rwxr-xr-x 1 root root

-rwxr-xr-x 1 root root

-rwxr-xr-x 1 root root

....

$ ps -ef | more

35264 Nov 19

2012 [

96 Sep 26 20:28 2to3-2.7

96 Sep 25 18:23 2to3-3.2

16224 Mar 18

55336 Jul 12

2013 a2p

2013 ab

UID

root root root root root

...

$

PID

1

2

3

6

7

PPID C STIME TTY TIME CMD

0 0 Jan08 ? 00:00:00 /sbin/init

0 0 Jan08 ? 00:00:00 [kthreadd]

2 0 Jan08 ? 00:00:01 [ksoftirqd/0]

2 0 Jan08 ? 00:00:00 [migration/0]

2 0 Jan08 ? 00:00:04 [watchdog/0]