Grymoi
Navigat

Unix/Linux
Quotes
Bourne Shell
C Shell
File Permissions
Regular Expressions
grep
awk
UPDATED
sed
UPDATED
find
tar
inodes
Security
IPv6
Wireless
Hardware
spam
Deception
PostScript

Grep - An introduction to grep and egrep. How to search for strings inside of files.

Table of Contents

grep - the basics

I assume you have (or will soon) read the section on regular expressions. *Grep* uses regular expressions, and most of the power comes from their flexibility. I will only use simple examples in this section, so you understand the essentials of *grep*. Real mastery comes after mastering regular expressions.

How did grep get it's name?

The name *grep* comes from a command used in one of the early Unix editors. The command searched for a regular expression, and printed it out. As an example, if you wanted to search for the string "junk," the command to print the first line containing the word was "/junk/p" and the command to print all lines that contains the word was "g/junk/p." The "g" was an abbreviation for "global search."

This feature was used so much, that somebody decided to make it easier to use, smaller, and faster by creating a smaller program that only did this global search for regular expressions, and print. The called it *grep*, which was short for "g/*regular expression*/p," or "g/re/p." You may see similar commands today in *vi* and *sed*.

The Simple Example

Most people first use *grep* as a way to search the contents of their files. If you wanted to find the file that contained the password to another computer, you could execute

```
grep password *
```

All lines in all files that contain this word are printed out. The output might be this:

```
notes: password for the system "bigvax" is "guest", remember
to
notes: delete this message, as it is a bad idea to keep
passwords
message: Do you know the password for bigvax? I forgot what
```

The example found two files that contained the word, and one file contained it twice. That was easy, wasn't it? Yes. It is also easy for someone who has access to your files to find this information out. **Never** store your passwords in a file on the computer. That is almost as bad as writing them down on a piece of paper and taping it to your display.

Search for uppercase and lower case words

If any of your files included the words "PASSWORD," or "Password," the above example would not print them out. When you tell *grep* to search for an exact string, it does what it is told. While you could write an regular expression that includes upper and lower case patterns, *grep* has a feature for this exact purpose: "-i," or ignore case. That is,

```
grep -i password *
```

would find all variations, including a mixture of upper and lower case letters.

Using grep as a filter

You can have *Grep* operate on standard input, as well as files specified on the command line. Of course, if used as a filter, *grep* does not list the filename, as it doesn't know the name.

Forcing grep to print a filename

You may have noticed that when you give *grep* one filename as an argument, it does not list the filename. For instance, if you typed

```
grep password message
```

the output would be

```
Do you know the password for bigvax? I forgot what
```

As you can see, the "message: " was omitted. If you want to force *grep* to print a filename, always make sure that it is given more than one file as an argument. This might seem difficult, because if you wanted two files, you would have specified two files. *U has a simple solution: use a file that is always there, and always empty. This file is called */dev/null*. Example:

```
grep password message /dev/null
```

This is very convenient when you are writing shell scripts, and don't know how many files you will be told to search. Here is a simple script called *igrep* that does almost the same as "grep -i," with the exception taht it always prints the filename:

```
#!/bin/sh
grep -i $* /dev/null
```

Click here to get file: **igrep.sh**

Showing lines that don't contain a pattern

A very simple use of *grep* is to remove lines that contain a pattern. To remove all lines that contain the work "junk," use the "-v" option:

```
grep -v junk
```

This is typically used as a filter:

```
grep -i password * | grep -v junk
```

I often use it to eliminate excess lines. Suppose I wanted to search for the word "every," but I don't want "everyone," "everybody," or "everywhere." I could use the command

```
grep every * | grep -v one | grep -v body | grep -v where
```

This is handy with the C shell command repeat feature. I can execute the last command, and remove lines that contain certain words:

```
!! | grep -v ignorethisword
```

As an example, when I use *find* to look for a file, but don't want to look for backups of the file, I keep adding additional strings to ignore, especially at the end of the filename:

```
find . -print | grep -v '.old$' | grep -v '[%~]$'
```

Searching for a hyphen

Looking for certain strings can be difficult. Suppose you wanted to search for the combination "-i?" As you can image, typing

```
grep -i file
```

does not work. In fact, *grep* considers "file" to be the pattern, and then searches standard input for the word "file." If you type the above command, it will wait for you to type control-D before it ends, as it is reading standard input. You see, *grep* considers the hyphen as an indication of an option. In a case like this, you must deal with two problems:

Getting the argument past the shell

Getting the argument into the right form for the command.

This requires an understanding of the shell quoting functions, as well as regular expressions. In this case, the hyphen is not s special shell character, unless it is in square brackets. So the characters "-i" are ignored by the shell. To put it another way, all four commands below operate the same way, treating the "-i" as a command line option:

```
grep -i *
grep '-i' *
grep "-i" *
grep -i *
```

However, *grep* does thing the hyphen is special, so we must get *grep* to treat it differently. The best way is to create a regular expression that does not start with a hyphen. As you recall, the "." character, in regular expressions, matches any character. Therefore the command

```
grep .-i *
```

will match every line containing "-i" except when it is the first character on a line. Another way to create a regular expression is to use the square brackets to list the options: "[-]i." Remember that the hyphen is special in square brackets, except when the first of last character. However, the shell command

```
grep [-]i *
```

does not work. The square brackets are special to the shell, which will search for files in the current directory. In this case, the shell will search for files that match "-i." Either it finds this file, and expands it to

```
grep -i <all the other files>
```

or it doesn't find this file, and expands it to

```
grep <all the other files>
```

In either case, the action is not what you wanted. Therefore the proper way to do this is to enclose the regular expression with single quotation marks, so the pattern is passed to grep unchanged. That is, the proper command is

```
grep '[-]i' *
```

In general, it is best to quote the regular expression in single quotes, and then modify the regular expression so the proper pattern is passed to *grep*.

There is another solution. The special option "-e" means the next argument is a pattern, and not an option. Therefore

```
grep -e -i *
```

would also work. However, the other solutions work for any command, which "-e" is a feature of *grep*, and may not be available in other utilities.

grep -s
grep -l
grep -w

fgrep egrep

egrep - show lines containing one of several

patterns

TBD *This document was translated by troff2html v0.21 on September 22, 2001.*