Lecture 9A: Grep

Grep

As mentioned before, `grep` is a great tool for searching. This lecture is going to cover more use-cases for grep, and introduce the idea of regular expressions!

The Origin of Grep Basically, *Global/(Regular Expression)/Print*

Grep returns zero if a match is found. We will talk more about exit codes when we discuss Bash scripting. Grep also has lots of options. Some common ones:

- `-c` - counts matched lines instead of printing them
- `-i` - ignores case *This one is very important!*
- `-n` - precedes each line with a line number
- `-v` - reverses sense of test, eg. finds lines not matching pattern
- `-r` - performs a recursive search
- `-w` - ignores results where pattern is contained inside a larger word

We are going to work with a file called `cars`. You might want to copy this file into your home folder:

```
cp ~eric.brauer/uli101/cars ~
```

Introduction

Now, let's try our command. We're going to use the 'cars' file to test grep. For reference, this is what's in the 'cars' file:

```
cat cars
```

```
 chevy   nova    79      60      3000
ford    mustang 65      45      17000
volvo   gl      78      102     9850
honda   civic   98      112     3200
ford    ltd     83      15      10500
Chevy   nova    80      50      3500
honda   accord  81      30      6000
toyota  tercel  82      180     750
toyota  rav4    08      65      12000
chevy   impala  65      85      1550
ford    thndbd  70     110      8005
```

First, we'll search for 'chevy.'

```
grep 'chevy' cars
```

```
 chevy   nova    79      60      3000
chevy   impala  65      85      1550
```

Turn on line numbers:

```
grep -n 'chevy' cars
```

```
 2:chevy   nova    79      60      3000
11:chevy   impala  65      85      1550
```

Now let's try something else.

```
grep -n 'Chevy' cars
```

```
6:Chevy   nova    80      50      3500
```

Notice that this returned a completely different line, since grep is by default case specific. This can be tricky, since often I've found with grep that you want to search broadly for an expression, and then get more specific. If you want grep to ignore case by default, you could use an alias. But let's leave it for now.

So let's try this:

```
grep -i 'chevy' cars
```

```
 2:chevy   nova    79      60      3000
6:Chevy   nova    80      50      3500
11:chevy   impala  65      85      1550
```

Now since we used `-i` to ask grep to `--ignore-case`, we are getting both `Chevy` and `chevy`.

Count and Invert

Here's the count option.

```
grep -ic 'chevy' cars
```

```
3
```

That makes sense, right? Instead of printing the whole line, just count the number of results you find. Finally if you want to in**V**ert your search:

```
grep -v 'chevy' cars
```

```
 plym    fury    77      73      2500
ford    mustang 65      45      17000
volvo   gl      78      102     9850
ford    ltd     83      15      10500
fiat    600     65      115     450
honda   accord  81      30      6000
ford    thundbd 84      10      17000
toyota  tercel  82      180     750
ford    bronco  83      25      9525
```

This returns all the lines that *don't* match the pattern.

Word

For the next example, let's use our `frankenstein.txt` example.

```
grep -i 'eat' frankenstein.txt
```

```
 heart.  Unable to endure the aspect of the being I had created, I
the hue of death; her features appeared to change, and I thought that I
created.  He held up the curtain of the bed; and his eyes, if eyes they
during the rest of the night, walking up and down in the greatest
I passed the night wretchedly.  Sometimes my pulse beat so quickly and
he, “how great was the difficulty to persuade my father that all
answer to my unwearied entreaties was the same as that of the Dutch
```

Notice that these lines ds *not* contain 'eat', but words like 'created', 'death', and 'greatest'. **One important thing to note is that `grep` will return the pattern if it is contained in another word.**

Let's limit our results so that we only find the word 'eat', and not any word that contain the letters *eat*. We will use the `-w` option to limit our results to the *word* eat. A better way to say this, is that we will match 'eat' *as long as it is surrounded by space characters*. This includes spaces but also tabs, newlines, and so on.

```
grep -iw 'eat' frankenstein.txt
```

```
 a year without Greek, I eat heartily without Greek.’ But his
eat.  The meal was quickly dispatched.  The young woman was again
‘monster! Ugly wretch! You wish to eat me and tear me to pieces. You
you follow not too tardily, a dead hare; eat and be refreshed. Come on, my
```

Recursive

Finally, let's look at recursive searches. Like we saw last week, a recursive grep command can take a very long time to complete. But it lets you search in many files at a time. Let's use `cd ..` to return to the top of our sample_dir1 directory. Since we're following a symbolic link, we should see this:

```
.
|-- gen_ed -> sample_dir/stenton/gen_ed/
`-- sample_dir
    |-- admin
    |-- cambridge
    |   |-- cafeteria
    |   `-- library
    |       `-- dir_practice
    |-- faculty
    |-- history.exe
    |-- markham
    |   |-- annex
    |   |-- annex2
    |   |-- building1
    |   `-- parking
    |-- outline.doc
    |-- oxford
    |   |-- outline.doc
    |   |-- programming
    |   |   `-- report.pdf
    |   `-- security
    `-- stenton
```

```
|-- gen_ed
|   `-- cars
|-- lib_arts
|   |-- english.txt
|   `-- match.doc
`-- parking2
```

So we are at `.` looking down, and we see many searchable files. Try this command: `grep -ir 'chevy' .` Remember, `-i` to ignore case. `-r` to perform a recursive search. `.` indicates that we want to start in our current directory (sample_dir1) and repeat the search in every subdirectory.

```
./sample_dir/stenton/gen_ed/cars:chevy   nova    79      60      3000
./sample_dir/stenton/gen_ed/cars:Chevy   nova    80      50      3500
./sample_dir/stenton/gen_ed/cars:chevy   impala  65      85      1550
```

We are finding those three results that we saw before in 'cars', however none of those other files contain either 'Chevy' or 'chevy.' Here's another view, this time using `-c` :

```
grep -irc 'chevy' .
```

```
./sample_dir/cambridge/library/dir_practice:0
./sample_dir/history.exe:0
./sample_dir/markham/annex:1
./sample_dir/markham/building1:0
./sample_dir/markham/parking:0
./sample_dir/markham/annex2:0
./sample_dir/oxford/outline.doc:0
./sample_dir/oxford/programming/report.pdf:0
./sample_dir/stenton/gen_ed/cars:3
./sample_dir/stenton/lib_arts/english.txt:0
./sample_dir/stenton/lib_arts/match.doc:0
./sample_dir/stenton/parking2:0
./sample_dir/outline.doc:0
```

This gives us a long list of results: we see *every file* in the location along with the number of 'chevies' that the file contains. If we want to *exclude* the files with zero results, we can use pipes to do this:

```
grep -irc 'chevy' . | grep -v ':0'
```

The *first* grep command is matching with the word 'chevy' and creating that long output that we saw before. We are then piping that output into a *second* grep, and *excluding* the results that contain ':0'.

```
./sample_dir/markham/annex:1
./sample_dir/stenton/gen_ed/cars:3
```

This gives us a nice summary of results. What about a different word? Let's use the word 'help', let's ignore case and search in our home directory.

```
ps -ef | grep firefox
```

...will return the process ID for Firefox, if you have it running.

Summary

- `grep`
  - `-c` : counts matched lines instead of printing them
  - `-i` : ignores case *This one is very important!*
  - `-n` : precedes each line with a line number
  - `-v` : reverses sense of test, eg. finds lines not matching pattern
  - `-r` : performs a recursive search