Lecture 5B: Filtering II

Programming is often a process of trying to maximize efficiency. As you proceed in your career, you will often get into discussions about performance– which languages are 'faster' in terms of CPU cycles, for example? How can we successfully scale a solution? Sometimes this comes down to a discussion of 'how can we solve this problem in as few lines of code as possible?'

There is another way to look at efficiency. We've often remarked in this class that as programmers, your time is expensive. No matter where you end up, chances are that project managers will care less about lines of code and more about how many hours it will take to create a solution to a problem. Some projects are an exception, of course. Sometimes you will have time and resources thrown at you, and your project manager will be interested in an ideal, perfect (fast!) program. Most likely there will be tight deadlines, and the expectation that CPU cycles are cheap.

This isn't to say that you shouldn't search for efficient solutions to problems, just that it is a mistake to *let perfect become the enemy of the good*.

All this is a way to re-introduce the filtering commands we've already discussed. As we progress with this example, I encourage you to think about how you might attempt to solve this problem using other tools. How would you go about solving this problem in C? How about Java, or Python? What are the tradeoffs?

The Example

"*How would you find out the ten largest files in /usr/bin? Print only the size and program name, separated with tabs.*"

There are a lot of ways to solve this problem (if you read the man pages for `ls`, you'll find a very fast solution!) but let's use the tools we've been given.

Let's break the problem into steps. First step: list the files in /usr/bin.

```
 ls -lh /usr/bin | head
```

```
 total 13M
-rwxr-xr-x 1 root root 1.1M Apr  4 14:30 bash
-rwxr-xr-x 1 root root 732K Aug 29 03:57 brltty
-rwxr-xr-x 1 root root  35K Jan 29  2017 bunzip2
-rwxr-xr-x 1 root root 2.0M Dec 12  2017 busybox
-rwxr-xr-x 1 root root  35K Jan 29  2017 bzcat
lrwxrwxrwx 1 root root    6 Apr  2  2018 bzcmp -> bzdiff
-rwxr-xr-x 1 root root 2.1K Jan 29  2017 bzdiff
lrwxrwxrwx 1 root root    6 Apr  2  2018 bzegrep -> bzgrep
-rwxr-xr-x 1 root root 4.8K Jan 29  2017 bzexe
```

Note I'm using `head` on each step just to demo the steps.

`ls` separates everything with variable numbers of spaces. Let's replace one or more spaces with commas.

```
ls -lh /usr/bin | tr -s ' ' ',' | head
```

```
 total,13M
-rwxr-xr-x,1,root,root,1.1M,Apr,4,14:30,bash
-rwxr-xr-x,1,root,root,732K,Aug,29,03:57,brltty
-rwxr-xr-x,1,root,root,35K,Jan,29,2017,bunzip2
-rwxr-xr-x,1,root,root,2.0M,Dec,12,2017,busybox
-rwxr-xr-x,1,root,root,35K,Jan,29,2017,bzcat
lrwxrwxrwx,1,root,root,6,Apr,2,2018,bzcmp,->,bzdiff
-rwxr-xr-x,1,root,root,2.1K,Jan,29,2017,bzdiff
lrwxrwxrwx,1,root,root,6,Apr,2,2018,bzegrep,->,bzgrep
-rwxr-xr-x,1,root,root,4.8K,Jan,29,2017,bzexe
```

Next step: let's cut out all the extra information.

```
ls -lh /usr/bin | tr -s ' ' ',' | cut -d, -f 5,9 | head
```

```
 1.1M,bash
732K,brltty
35K,bunzip2
2.0M,busybox
35K,bzcat
6,bzcmp
2.1K,bzdiff
6,bzegrep
4.8K,bzexe
```

With `cut`, we need to specify a delimiter (,). To option to do this is `-d,` .Next, we want two fields: filename and size. (I tried to start with name but it didn't work and I forgot to try and fix it...)

Next step: Sorting. Notice our file sizes are in *human readable* format. Notice also that we will need to specify our delimiter again.

```
ls -lh /usr/bin | tr -s ' ' ',' | cut -d, -f 5,9 | sort -h -t, -k1 | head
```

```
4,lsmod
4,ping4
4,ping6
4,rbash
4,rnano
4,sh
4,sh.distrib
6,bzcmp
6,bzegrep
```

What happened? We are getting very *small* results. We will talk soon about links, because there's no way that nano or ping are actually 4 bytes in size.

So we need to *reverse* our sort, or use `tail` . Let's reverse the sort:

```
ls -lh /usr/bin | tr -s ' ' ',' | cut -d, -f 5,9 | sort -h -t, -k1 -r | head
```

```
 2.0M,busybox
1.1M,bash
732K,brltty
571K,udevadm
542K,ip
414K,tar
241K,nano
215K,grep
207K,loadkeys
193K,hciconfig
```

Much better. The last thing I want to do is to replace commas with tabs. We could have started with tabs, right? Yeah, unfortunately not. I don't want to overwhelm you, so look here if you *really* want to know more, otherwise, let's just do it my way. :)

```
ls -lh /usr/bin | tr -s ' ' ',' | cut -d, -f 5,9 | sort -h -t, -k1 -r | tr ',' '\t' | head
```

```
 2.0M    busybox
1.1M    bash
732K    brltty
571K    udevadm
542K    ip
414K    tar
241K    nano
215K    grep
207K    loadkeys
193K    hciconfig
```

...And we're done! When I did this the first time, I was able to get this done in about two minutes, and in "one" line of my own code. Hopefully my project manager was pleased... Now of course, my "two minute" job doesn't include the literal months of trial and error I had to go through on the beginning of my Linux journey. I reached for these tools because these are tools I'm used to working with. Your mileage may vary, and that's okay! Your solution might be better than mine (in fact there's a very good chance of that :)) and that's okay too. All we can do is show you some of the tools that others have found useful, and the skills that employers find useful.

---

tee

To demonstrate `tee`, let's take our previous long one-line command, and redirect it into a file for safekeeping.

```
ls -lh /usr/bin | tr -s ' ' ',' | cut -d, -f 5,9 | sort -h -t, -k1 -r | tr ',' '\t' | head > top-bins
```

Nothing is printed. As you know by now, we are *redirecting* our output into that file. We are picking up the garden hose that is STDOUT, and pointing it into a bucket. The grass is dry.

What if we connect up a T-shaped attachment to that hose so that we can fill our bucket and water our grass at the same time?

```
rm top-bins  ls -lh /usr/bin | tr -s ' ' ',' | cut -d, -f 5,9 | sort -h -t, -k1 -r
| tr ',' '\t' | tee top-bins | head
```

```
 2.0M    busybox
1.1M    bash
732K    brltty
571K    udevadm
542K    ip
414K    tar
241K    nano
215K    grep
207K    loadkeys
193K    hciconfig
```

```
cat top-bins | head
```

```
 2.0M    busybox
1.1M    bash
732K    brltty
571K    udevadm
542K    ip
414K    tar
241K    nano
215K    grep
207K    loadkeys
193K    hciconfig
```

We have STDOUT going to our display, as normal. But now we also have our **complet** output copied into a file. This is useful for logging.

Grep

Grep is a fantastically useful tool. We are diving deeper into it after the break. But here is the easiest way to use grep:

```
grep ls top-bins
```

> 131K **ls**
>
> 83K **ls**blk
>
> 32K ntfs**ls**
>
> 31K fa**ls**e
>
> 4 **ls**mod

What does this tell us about grep? We are searching for a *pattern* (in this case, 'ls') inside a *file* (top-bins). This is essentially your Find tool from 1983.

Try this:

```
ls -lh /home | grep <your name>
```

We will be using grep quite a bit as we go, this is just the tip of the iceberg. Another thing you can try is this:

```
cp /home/eric.brauer/dune.txt .

grep worm dune.txt
```

Your results are going to be very hard to read. Why? Grep is returning every line where your pattern matches.

```
grep -n worm dune.txt
```

The `-n` option also includes a line number.

Summary

The following are important to remember for quizzes and tests:

- `tr 'SET1' 'SET2'` : replace SET1 with SET2.
- `cut -d#` : specify delimiter #.
- `cut -f#` : specify field (column).
- `sort -k#` : specify key (column).
- `sort -t#` : specify delimiter #.
- `sort -n` : sort numerically.
- `sort -r` : reverse the sort.
- `grep PATTERN FILE` : find a pattern in a file.
- `tee FILE` : make a copy of STDIN to FILE, and *also* pass it to STDOUT.
- ...as well as `head` , `tail` , `wc` , and so on.

Review for Quiz #2

0. What is passed between `command1 | command2` ?

   The *output* of `command1` is passed to the *input* of `command2` . *Data* is being passed.</passed.

1. Convert the decimal number **103** to hexadecimal.

   0x67

2. What does **pass-through** refer to?

   pass-through is a type of permission for a directory. Without pass-through, no user can access any of the files in a directory, regardless of any other permission. ~uli101 has pass-through so that you can execute the assignments.

3. Set the permissions of `file1` so that the user can read and write, and all other users (group members and others) have the minimum required permissions to execute the file.

chmod 655 file1

4. Add write permissions for the group members and others to the file `public-sketchbook` . Use a *symbolic* command.

   chmod go+w public-sketchbook

5. How can you set the default permissions so that all files will only be readable and writable by users?

   The permission level we want to set is 600. So our command will be the inverse of that. `umask 177`

6. Log into Matrix, if you aren't already. Type in `who am i` (with spaces). Use a pipe and `cut` to print only your user ID and pts value.

   who am i | cut -d" " -f1,2

7. Copy `/home/eric.brauer/uli101/SSDexample` to your home folder. How can you find all SDDs made by Kingston? Sort these alphabetically, and return only the first result.

   grep Kingston SSDexample | sort | head -1

Challenge

Work with another student. Use `vi` to open a text file and `watch date` to keep track of time. In one minute, answer the question: *Name a bad job for someone who's accident prone*. Enter your answers, one on each line. (Or search for some other Family Feud type question online).

Check your file permissions so that you can share your list with the other student. Use `sort` to sort your answers, then use either `diff` or another tool to compare your answers.

Research ways online that you could compare the two outputs, find only the unique lines, and then count up your score. This is an advanced exercise, but if you can manage it then you will be in great shape for tests..!

Even more to read, if you have the time...

STDIN redirection with tr

You will notice that `tr` does *not* accept filenames as arguments. In order to use `tr` in this way, we need to use the rarest of birds, STDIN redirection!

command < filename

`tr 'a-z' 'A-Z' < SDDexample

More redirection (2>&1)

In some cases you will want to send STDOUT and STDERR to the same place, possibly a logfile. You can do this with the following:

```
find ~ -iname report.pdf > logfile 2>&1
```

In this case, `find` is told to send all results to logfile, and then STDERR is redirected into STDOUT.