`sed` and Pipelines

4.6 `sed`: Stream Editing

`grep` is an incredibly powerful filtering tool, however, it's not always the case that you wish to only search for something that matches a pattern. Often enough, you'll want to edit that which you've already searched. `sed` works very similarly to grep, however, instead of just filtering the input based on the given regex pattern like `grep` does, it also transforms the data before it sends the newly transformed data to standard out. You could also think of `sed` as a significantly more powerful and complex version of `tr`.

`sed` has a processing cycle:

1. If no [more] input lines, terminate, otherwise, read line into buffer and increment line count.
2. Apply provided editing schema to buffered line.
3. Write buffered line to stdout.
4. Go to step 1.

Ultimately, if a file's lines don't match the criteria set by the expression in the `sed` command, nothing will be edited in the stdout.

`sed` has a general construction of:

```
sed script fileName
```

The script that gets passed to `sed` can do a number of things:

Substitution

A good first example of sed is to substitute a specific character (or string of characters). The parts of a substitution command are:

- `s`: substitute command
- `../.../.../..`: The delimiter for the script
- `stringToFind`: The fixed string or regular expression pattern to search
- `stringToReplace`: The string to replace the values of those found by the fixed string / regex pattern

```
sed 's/stringToFind/stringToReplace/' fileName
```

Remember back to our `.csv` file. Often times, many of the data that we wish to store will have commas in them (think of a field titled "full_name" as opposed to "first_name" and a second field "last_name"). Another example could be that a user may have a suffix, such as `.jr` or `the 3rd`. Often times these suffixes may also have an additional comma. How could we get around that?

Even though `.csv` files have *comma* in the name, they don't necessarily need to have a comma as a delimiter. In fact, many `.csv` files use `|` as a delimiter (note: literally anything can be used as a delimiter, but it should arguably be a relatively uncommon character).

To prepare our `.csv` file for a potential of obtaining data that could have a comma in it (so that adding this new comma filled data won't destroy our `.csv`), let's change the current delimiter from a `,` to a `|` with a basic substitution:

```
sed 's/,/|/' employeeData.csv
```

```
1|Barr,Mateev,bmateev0@sphinn.com,Occupational Therapist,$111420.66
2|Winifred,Shreenan,wshreenan1@miibeian.gov.cn,Nurse,$308885.35
3|Alfy,McGlade,amcglade2@patch.com,Human Resources Assistant IV,$678731.10
4|Batsheva,Gask,bgask3@slate.com,Paralegal,$831875.51
5|Cecil,Skones,cskones4@blogger.com,Chemical Engineer,$593684.83
6|Clarine,Coskerry,ccoskerry5@people.com.cn,Paralegal,$502580.02
7|Pierette,Allerton,pallerton6@aol.com,Payment Adjustment Coordinator,$562292.95
8|Gwendolen,McIlwrick,gmcilwrick7@sciencedaily.com,Design Engineer,$127821.14
9|Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83
10|Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

Curiously, we only seem to have replaced the first instance of our data. By default, sed will only find and replace the first pattern that the substitution string matches. In order to replace all, we'll have to add the **global** pattern flag as a suffix to our command. Let's add the global suffix on our substitution script:

```
sed 's/,/|/g' employeeData.csv
```

```
id|first_name|last_name|email|job_title|salary
1|Barr|Mateev|bmateev0@sphinn.com|Occupational Therapist|$111420.66
2|Winifred|Shreenan|wshreenan1@miibeian.gov.cn|Nurse|$308885.35
3|Alfy|McGlade|amcglade2@patch.com|Human Resources Assistant IV|$678731.10
4|Batsheva|Gask|bgask3@slate.com|Paralegal|$831875.51
5|Cecil|Skones|cskones4@blogger.com|Chemical Engineer|$593684.83
6|Clarine|Coskerry|ccoskerry5@people.com.cn|Paralegal|$502580.02
7|Pierette|Allerton|pallerton6@aol.com|Payment Adjustment Coordinator|$562292.95
8|Gwendolen|McIlwrick|gmcilwrick7@sciencedaily.com|Design Engineer|$127821.14
9|Delmor|Reynish|dreynish8@friendfeed.com|Analyst Programmer|$441017.83
```

```
10|Mel|Marvell|mmarvell9@booking.com|Web Developer II|$85394.75
```

It's not always the case, though, that you'll find yourself wishing to replace ever single occurrence of a value. What if, however, you wished to replace a specific occurrence? You can do that by specifying the occurrence number. By using the `i` as a command as well, we are using **case insensitive** matching. The **2** is denoting which specific instance:

```
sed 's/Mateev/matiev/2i' employeeData.csv
```

```
id,first_name,last_name,email,job_title,salary
1,Barr,Mateev,bmatiev0@sphinn.com,Occupational Therapist,$111420.66
2,Winifred,Shreenan,wshreenan1@miibeian.gov.cn,Nurse,$308885.35
3,Alfy,McGlade,amcglade2@patch.com,Human Resources Assistant IV,$678731.10
4,Batsheva,Gask,bgask3@slate.com,Paralegal,$831875.51
5,Cecil,Skones,cskones4@blogger.com,Chemical Engineer,$593684.83
6,Clarine,Coskerry,ccoskerry5@people.com.cn,Paralegal,$502580.02
7,Pierette,Allerton,pallerton6@aol.com,Payment Adjustment Coordinator,$562292.95
8,Gwendolen,McIlwrick,gmcilwrick7@sciencedaily.com,Design Engineer,$127821.14
9,Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83
10,Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

In the above, we replaced Barr Mateev's email without overwriting the last name field by specifying that we only update the second occurrence of the name. This is a lot of data to look at for only one change. It would be nice if we could see only the lines that were updated.

Printing Edited Lines Only

By default, sed prints the entire file to standard out. You can turn this off with the `-n` flag. By doing this, however, nothing prints to standard out, so we're in the dark as to whether or not we've actually changed anything. In order see only what was edited in a sed command, we can use the `/p` pattern flag:

```
sed 's/Mateev/matiev/2ip' employeeData.csv
```

```
1,Barr,Mateev,bmatiev0@sphinn.com,Occupational Therapist,$111420.66
```

By doing this, we can clean up our output a bit so we're not crowding the screen with walls of data!

Writing to a File

While you can write to Writing to a file with sed looks a bit curious. Like printing, or using case insensitivity, we need to add a `w` to the pattern flag, however, we need to specify an output file name from within the script:

```
sed -n 's/Mateev/matiev/2ipw newDataFile.csv' employeeData.csv
cat newDataFile.csv
```

```
1,Barr,Mateev,bmatiev0@sphinn.com,Occupational Therapist,$111420.66
```

Multiple Scripts

It's not always the case that you'll want to make only one change. Taking the knowledge that we already have, we could pipe together two separate `sed` commands (**note** in the second command, we're no longer reading from the file employeeData.csv):

```
sed 's/,/|/g' employeeData.csv | sed 's/Mateev/matiev/2iw newDataFile.csv'
```

```
id|first_name|last_name|email|job_title|salary
1|Barr|Mateev|bmatiev0@sphinn.com|Occupational Therapist|$111420.66
2|Winifred|Shreenan|wshreenan1@miibeian.gov.cn|Nurse|$308885.35
3|Alfy|McGlade|amcglade2@patch.com|Human Resources Assistant IV|$678731.10
4|Batsheva|Gask|bgask3@slate.com|Paralegal|$831875.51
5|Cecil|Skones|cskones4@blogger.com|Chemical Engineer|$593684.83
6|Clarine|Coskerry|ccoskerry5@people.com.cn|Paralegal|$502580.02
7|Pierette|Allerton|pallerton6@aol.com|Payment Adjustment Coordinator|$562292.95
8|Gwendolen|McIlwrick|gmcilwrick7@sciencedaily.com|Design Engineer|$127821.14
9|Delmor|Reynish|dreynish8@friendfeed.com|Analyst Programmer|$441017.83
10|Mel|Marvell|mmarvell9@booking.com|Web Developer II|$85394.75
```

By piping the data together, we can do multiple commands and start to make our `sed` commands really powerful! BUT! Piping together multiple sed commands can get hairy. That's a lot to write. Instead, we can make use of the `-e` or `--expression` flag:

```
sed -e 's/,/|/g' -e 's/Mateev/matiev/2iw newDataFile.csv' employeeData.csv
```

```
id|first_name|last_name|email|job_title|salary
1|Barr|Mateev|bmatiev0@sphinn.com|Occupational Therapist|$111420.66
2|Winifred|Shreenan|wshreenan1@miibeian.gov.cn|Nurse|$308885.35
3|Alfy|McGlade|amcglade2@patch.com|Human Resources Assistant IV|$678731.10
4|Batsheva|Gask|bgask3@slate.com|Paralegal|$831875.51
5|Cecil|Skones|cskones4@blogger.com|Chemical Engineer|$593684.83
6|Clarine|Coskerry|ccoskerry5@people.com.cn|Paralegal|$502580.02
```

```
7|Pierette|Allerton|pallerton6@aol.com|Payment Adjustment Coordinator|$562292.95
8|Gwendolen|McIlwrick|gmcilwrick7@sciencedaily.com|Design Engineer|$127821.14
9|Delmor|Reynish|dreynish8@friendfeed.com|Analyst Programmer|$441017.83
10|Mel|Marvell|mmarvell9@booking.com|Web Developer II|$85394.75
```

`sed` Script Files

It is also possible that you could have a file filled with `sed` scripts so you don't have to constantly write the scripts over and over. Let's create a file with two scripts:

`sedScriptFile.txt`

```
# sed comment - These scripts change the delimiters and all data to lower case
s/,/|/g
s/[A-Z]/\L&/g
```

To run the script file, we need to use sed with a `-f` flag:

```
sed -f sedScriptFile.txt employeeData.csv
```

```
id|first_name|last_name|email|job_title|salary
1|barr|mateev|bmateev0@sphinn.com|occupational therapist|$111420.66
2|winifred|shreenan|wshreenan1@miibeian.gov.cn|nurse|$308885.35
3|alfy|mcglade|amcglade2@patch.com|human resources assistant iv|$678731.10
4|batsheva|gask|bgask3@slate.com|paralegal|$831875.51
5|cecil|skones|cskones4@blogger.com|chemical engineer|$593684.83
6|clarine|coskerry|ccoskerry5@people.com.cn|paralegal|$502580.02
7|pierette|allerton|pallerton6@aol.com|payment adjustment coordinator|$562292.95
8|gwendolen|mcilwrick|gmcilwrick7@sciencedaily.com|design engineer|$127821.14
9|delmor|reynish|dreynish8@friendfeed.com|analyst programmer|$441017.83
10|mel|marvell|mmarvell9@booking.com|web developer ii|$85394.75
```

In our script file, we changed the delimiter of our file, however, we also decided to turn all of our uppercase characters to lowercase. We did that by in the second portion of our script, using `\L&`. The preceding `\L` is a lowercase command. Using the `&`, we can pull in that which is captured by the provided pattern (in this case, `[A-Z]`), and use it. We could also add an underscore after every capitalized letter:

```
# sed comment - These scripts change the delimiters and all data to lower case
s/,/|/g
s/[A-Z]/&_/g
```

Then run the command:

```
sed -f sedScriptFile.txt employeeData.csv
```

```
id|first_name|last_name|email|job_title|salary
1|B_arr|M_ateev|bmateev0@sphinn.com|O_ccupational T_herapist|$111420.66
2|W_inifred|S_hreenan|wshreenan1@miibeian.gov.cn|N_urse|$308885.35
3|A_lfy|M_cG_lade|amcglade2@patch.com|H_uman R_esources A_ssistant
I_V_|$678731.10 4|B_atsheva|G_ask|bgask3@slate.com|P_aralegal|$831875.51
5|C_ecil|S_kones|cskones4@blogger.com|C_hemical E_ngineer|$593684.83
6|C_larine|C_oskerry|ccoskerry5@people.com.cn|P_aralegal|$502580.02
7|P_ierette|A_llerton|pallerton6@aol.com|P_ayment A_djustment
C_oordinator|$562292.95
8|G_wendolen|M_cI_lwrick|gmcilwrick7@sciencedaily.com|D_esign
E_ngineer|$127821.14 9|D_elmor|R_eynish|dreynish8@friendfeed.com|A_nalyst
P_rogrammer|$441017.83 10|M_el|M_arvell|mmarvell9@booking.com|W_eb D_eveloper
I_I_|$85394.75
```

By using the `&` we can do some more complex forms of formatting!

Pattern Matching

`sed` can also be used entirely for pattern matching and finding data, just like `grep`. While `grep` takes in the regular expression's pattern to match on, `sed` still requires that you use the script syntax:

```
sed -n '/[md]el/Ip' employeeData.csv
```

```
9,Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83
10,Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

In the above, we write our script to search for either `mel` or `del` and then, to ensure it was case insensitive we added the `I` pattern to make it case insensitive. Notice, though, that this `I` is capital? Try the command with a lower case `i`, and see what happens. Additionally, try `sed 's/Mateev/matiev/2iw newDataFile.csv'` that we used earlier with an uppercase `I` and see what happens.

Deleting Lines

Because `sed` is a file editor, it is able to just substitute or append, but it can also delete! In order to delete a specific line, you need only the `d` command in the script after a pattern to match:

```
sed '/[md]el/Id' employeeData.csv
```

```
id,first_name,last_name,email,job_title,salary
1,Barr,Mateev,bmateev0@sphinn.com,Occupational Therapist,$111420.66
2,Winifred,Shreenan,wshreenan1@miibeian.gov.cn,Nurse,$308885.35
3,Alfy,McGlade,amcglade2@patch.com,Human Resources Assistant IV,$678731.10
4,Batsheva,Gask,bgask3@slate.com,Paralegal,$831875.51
5,Cecil,Skones,cskones4@blogger.com,Chemical Engineer,$593684.83
6,Clarine,Coskerry,ccoskerry5@people.com.cn,Paralegal,$502580.02
7,Pierette,Allerton,pallerton6@aol.com,Payment Adjustment Coordinator,$562292.95
8,Gwendolen,McIlwrick,gmcilwrick7@sciencedaily.com,Design Engineer,$127821.14
```

Now, instead of selecting the lines to print like above, the `d` command lets us delete the lines entirely!

Adding Text:

Append

With the `a` command, it is possible to append text after the line:

```
sed '3a HELLO! I AM AFTER THE LINE' employeeData.csv
```

```
id,first_name,last_name,email,job_title,salary
1,Barr,Mateev,bmateev0@sphinn.com,Occupational Therapist,$111420.66
2,Winifred,Shreenan,wshreenan1@miibeian.gov.cn,Nurse,$308885.35 HELLO! I AM
AFTER THE LINE 3,Alfy,McGlade,amcglade2@patch.com,Human Resources Assistant
IV,$678731.10 4,Batsheva,Gask,bgask3@slate.com,Paralegal,$831875.51
5,Cecil,Skones,cskones4@blogger.com,Chemical Engineer,$593684.83
6,Clarine,Coskerry,ccoskerry5@people.com.cn,Paralegal,$502580.02
7,Pierette,Allerton,pallerton6@aol.com,Payment Adjustment Coordinator,$562292.95
8,Gwendolen,McIlwrick,gmcilwrick7@sciencedaily.com,Design Engineer,$127821.14
9,Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83
10,Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

With this script's syntax, you specify the line after which to append data!

Insert

Sometimes you'll want to add data before a line with `i`, such as adding a title:

```
sed '1i EMPLOYEE DATA' employeeData.csv
```

```
EMPLOYEE DATA id,first_name,last_name,email,job_title,salary
1,Barr,Mateev,bmateev0@sphinn.com,Occupational Therapist,$111420.66
2,Winifred,Shreenan,wshreenan1@miibeian.gov.cn,Nurse,$308885.35
3,Alfy,McGlade,amcglade2@patch.com,Human Resources Assistant IV,$678731.10
4,Batsheva,Gask,bgask3@slate.com,Paralegal,$831875.51
5,Cecil,Skones,cskones4@blogger.com,Chemical Engineer,$593684.83
6,Clarine,Coskerry,ccoskerry5@people.com.cn,Paralegal,$502580.02
7,Pierette,Allerton,pallerton6@aol.com,Payment Adjustment Coordinator,$562292.95
8,Gwendolen,McIlwrick,gmcilwrick7@sciencedaily.com,Design Engineer,$127821.14
9,Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83
10,Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

Replacing a Line:

And, it's possible to entirely replace a line with `c`:

```
sed '7c 6,Indiana,Jones,doctorjones@marshall.edu,Archaeologist,$40000.00'
employeeData.csv
```

```
id,first_name,last_name,email,job_title,salary
1,Barr,Mateev,bmateev0@sphinn.com,Occupational Therapist,$111420.66
2,Winifred,Shreenan,wshreenan1@miibeian.gov.cn,Nurse,$308885.35
3,Alfy,McGlade,amcglade2@patch.com,Human Resources Assistant IV,$678731.10
4,Batsheva,Gask,bgask3@slate.com,Paralegal,$831875.51
5,Cecil,Skones,cskones4@blogger.com,Chemical Engineer,$593684.83
6,Indiana,Jones,doctorjones@marshall.edu,Archaeologist,$40000.00
7,Pierette,Allerton,pallerton6@aol.com,Payment Adjustment Coordinator,$562292.95
8,Gwendolen,McIlwrick,gmcilwrick7@sciencedaily.com,Design Engineer,$127821.14
9,Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83
10,Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

Adding New Lines

Often times, you'll wish to add new lines in a file. You can do that with the `G` command:

```
sed '1G' employeeData.csv
```

```
id,first_name,last_name,email,job_title,salary
```

```
1,Barr,Mateev,bmateev0@sphinn.com,Occupational Therapist,$111420.66
2,Winifred,Shreenan,wshreenan1@miibeian.gov.cn,Nurse,$308885.35
3,Alfy,McGlade,amcglade2@patch.com,Human Resources Assistant IV,$678731.10
4,Batsheva,Gask,bgask3@slate.com,Paralegal,$831875.51
5,Cecil,Skones,cskones4@blogger.com,Chemical Engineer,$593684.83
6,Clarine,Coskerry,ccoskerry5@people.com.cn,Paralegal,$502580.02
7,Pierette,Allerton,pallerton6@aol.com,Payment Adjustment Coordinator,$562292.95
8,Gwendolen,McIlwrick,gmcilwrick7@sciencedaily.com,Design Engineer,$127821.14
9,Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83
10,Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

Or, suppose you wished to double space an entire file:

```
sed G employeeData.csv
```

```
id,first_name,last_name,email,job_title,salary

1,Barr,Mateev,bmateev0@sphinn.com,Occupational Therapist,$111420.66

2,Winifred,Shreenan,wshreenan1@miibeian.gov.cn,Nurse,$308885.35

3,Alfy,McGlade,amcglade2@patch.com,Human Resources Assistant IV,$678731.10

4,Batsheva,Gask,bgask3@slate.com,Paralegal,$831875.51

5,Cecil,Skones,cskones4@blogger.com,Chemical Engineer,$593684.83

6,Clarine,Coskerry,ccoskerry5@people.com.cn,Paralegal,$502580.02

7,Pierette,Allerton,pallerton6@aol.com,Payment Adjustment
Coordinator,$562292.95

8,Gwendolen,McIlwrick,gmcilwrick7@sciencedaily.com,Design Engineer,$127821.14

9,Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83

10,Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

Or, in a much more complex method, adding new lines around a matched pattern:

```
sed '/@aol\.com/{x;p;x;G}' employeeData.csv
```

```
id,first_name,last_name,email,job_title,salary
1,Barr,Mateev,bmateev0@sphinn.com,Occupational Therapist,$111420.66
2,Winifred,Shreenan,wshreenan1@miibeian.gov.cn,Nurse,$308885.35
3,Alfy,McGlade,amcglade2@patch.com,Human Resources Assistant IV,$678731.10
```

```
4,Batsheva,Gask,bgask3@slate.com,Paralegal,$831875.51
5,Cecil,Skones,cskones4@blogger.com,Chemical Engineer,$593684.83
6,Clarine,Coskerry,ccoskerry5@people.com.cn,Paralegal,$502580.02

7,Pierette,Allerton,pallerton6@aol.com,Payment Adjustment
Coordinator,$562292.95

8,Gwendolen,McIlwrick,gmcilwrick7@sciencedaily.com,Design Engineer,$127821.14
9,Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83
10,Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

What is happening in the above script is relatively advanced. We won't be going too much into this, but it's good to know it exists. What's occurring is that we're using the first portion of our script to match on the string `@aol.com` with the `.` escaped because we want the character and not the wild card.

We then use the *command grouping* (i.e. the commands inside of the `{...}` where we'll only execute those commands on the lines that match the pattern) to

- `x`: exchange data from the read in line from the pattern space with the hold buffer (since nothing was placed in hold earlier, it is empty). That is to say, the emptiness in the hold buffer is now in the pattern space, and Pierette Allerton's line is now in the hold buffer.
- `p`: prints out new pattern space, which just so happens to be empty. (First new line)
- `x`: exchanges data between the hold buffer and the pattern space. (Pierette Allerton's data is now back in the pattern space).
- `G`: a new line is appended!

Printing Out Line Numbers

You may find that you wish to know a specific line number. You can do that with the `=` command. This command prints the line number, then prints the data on a new line:

```
sed = employeeData.csv
```

```
1 id,first_name,last_name,email,job_title,salary 2
1,Barr,Mateev,bmateev0@sphinn.com,Occupational Therapist,$111420.66 3
2,Winifred,Shreenan,wshreenan1@miibeian.gov.cn,Nurse,$308885.35 4
3,Alfy,McGlade,amcglade2@patch.com,Human Resources Assistant IV,$678731.10 5
4,Batsheva,Gask,bgask3@slate.com,Paralegal,$831875.51 6
5,Cecil,Skones,cskones4@blogger.com,Chemical Engineer,$593684.83 7
6,Clarine,Coskerry,ccoskerry5@people.com.cn,Paralegal,$502580.02 8
7,Pierette,Allerton,pallerton6@aol.com,Payment Adjustment Coordinator,$562292.95
9 8,Gwendolen,McIlwrick,gmcilwrick7@sciencedaily.com,Design Engineer,$127821.14
10 9,Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83 11
```

```
10,Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

Granted, the above doesn't necessarily look that great, especially given that we have numerical data to begin with. Suppose we just wanted to know a specific line of only the matching data. Let's revisit our attempt to find Mel and Delmor:

```
sed -n '/[md]el/I{=;p;}' employeeData.csv
```

```
10 9,Delmor,Reynish,dreynish8@friendfeed.com,Analyst Programmer,$441017.83 11
10,Mel,Marvell,mmarvell9@booking.com,Web Developer II,$85394.75
```

We first set the command to make sure that the search is case insensitive, and we then move into the command grouping to be executed with the lines that match our search pattern.

Quitting A Script

It's also entirely possible that you only wish to see up to a certain point in a file. You can quit a script by specifying the line number at which to stop:

```
sed 2q employeeData.csv
```

```
id,first_name,last_name,email,job_title,salary
1,Barr,Mateev,bmateev0@sphinn.com,Occupational Therapist,$111420.66
```

Here we got to the second line, and then immediately quit the command!

4.7 Pipelines

We already know it's possible to create a pipeline of data from chapter 3, however, we have not remotely begun to fully explore the possibilities. Given that almost all bash commands accept data from the standard input and return data via the standard output, it is entirely possible for any number of commands to be piped to and from each other, like:

```
cmd1 | cmd2 -flag | cmd3 | cmd4 >> printDataToSomeFile
```

We've already seen the example:

```
history | grep echo
```

Where we search our history for a specific command (or anything) that we've made in the past.

You can pipe any number of commands together, such as:

```
ls -l | grep ^[^d]rwx | sed 's/^.*\s// '
```

> `basicScript.sh failforever.sh simpleWriteToFile.sh takeforever.sh takesArguments.sh testFile.sh`

This command will list all files that have read, write, and execute permissions for the user. The pipe's process is:

- List all of the files and subdirectories within a directory
- Search with `grep` the beginning of each line (`^`),
- Make sure the first character is not d (`[^d]`)
- Ensure the next 3 characters are `r` `w` and `x`, which would have the output:

  > `-rwx------ 1 mjlny2 401451 59 Jun 16 23:50 basicScript.sh -rwx------ 1 mjlny2 401451 88 Jun 15 02:53 failforever.sh -rwx------ 1 mjlny2 401451 637 Jun 19 12:35 simpleWriteToFile.sh -rwx------ 1 mjlny2 401451 122 Jun 15 02:47 takeforever.sh -rwx------ 1 mjlny2 401451 78 Jun 17 01:37 takesArguments.sh -rwx------ 1 mjlny2 401451 24 Jun 17 19:04 testFile.sh`

- Then, use `sed` to match the pattern of everything from the beginning of the line to the final space `^.*\s`
- Replace it with nothing (notice nothing between the two `//`s).
- By default, `sed` will print the output to the standard out.

  > basicScript.sh failforever.sh simpleWriteToFile.sh takeforever.sh takesArguments.sh testFile.sh