Learning Objectives

Learners will be able to edit documents using the following editors:
- sed
- awk
- printf
- nano
- vi(m)

Stream Editor (sed)

The name `sed` is short for stream editor. The `sed` command is most often used for finding and replacing, or searching and deleting. The `sed` utility in its simplest form works like `grep`, you can use regular expressions to find a string in a file. It's much more powerful though as you can provide a substitute string for the string you found, or delete every line that matches a string.

First take a look at the `man` pages for the `sed` command.

```
man sed
```

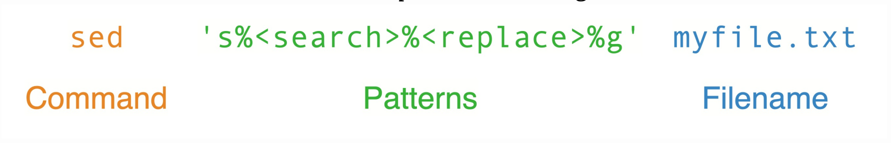
Using `sed` to find a string in a file

Try the following command:

```
sed -n '/Pooh/p' myfile.txt
```

1. The `-n` means it will only produce output when explicitly told to via the `p` command (the default is to print each line)
2. The `'/Pooh/p'` tells it to print out lines that have the word "Pooh" in them.

You can also use `sed` to find *and replace* something in a file:



```
sed   's%<search>%<replace>%g'  myfile.txt
```

1. The `s` means substitute.
2. The `%` is the delimiter (we can use any characters here).
3. The `search pattern` follows the first `%`.
4. The `replace pattern` follows the second `%`.
5. The `g` stands for global replace - replace all occurrences in the file.

For example, you can use `sed` to delete underscores in a file. Take a look at myfile.txt There are underscores around some words that probably have to do with the formatted version of this material. To get rid of them we create a pattern that looks for an underscore and replaces it with nothing.

Try the following command:

```
sed 's%_%%g' myfile.txt
```

Try these variations:

- Change the name of one of the characters in the text to something else.

  ▼

  Solution

  The solution is similar to the previous one, you just need something in the “replace” pattern slot

  ```
     sed 's%Christopher%Chris%g' myfile.txt
  ```

- Make a change and send the output to a file instead of the console.

  ▼

  Solution

  Use the redirection operator `&gt` to accomplish this.

  ```
     sed 's%_%%g' myfile.txt > newfile.txt
  ```

  You can then view the contents of `newfile.txt`:
  `markdown   cat newfile.txt`

- Make a change and save it in the original file. Hint: there is a command line option for this.

  ▼

  Solution

  The `-i` command line option means edit “in place”.

  ```
     sed -i 's%_%%g' myfile.txt
  ```

Using `sed` to delete lines

There are numerous blank lines at the end of `myfile.txt`. Use the following command to delete them:

```
sed '45,54d' myfile.txt
```

1. In `'45,54'` the comma indicates a range.
2. The `d` lets `sed` know it will be a deletion

Try these variations:

- Delete all blank lines. Hint: you need a pattern that will match a blank line.

  ▼

  Solution

  The `^` indicates the beginning of the line and the `$` indicates the end of the line and since there is nothing between the two, it means a blank line. This is deleting all the blank lines, not just the ones at the end so not the perfect solution for this case.

  ```
    sed '/^$/d' myfile.txt
  ```

- From a specific line to the end of the file.

  ▼

  Solution

  The last line is the string “So I tried”, starting with this search pattern it will delete everything until the end of the file, unfortunately including this line. In this case, it isn’t exactly what we want.

  ```
    sed '/So I tried/,$d' myfile.txt
  ```

- Delete everything **after** a specific line and save it “in place”. Hint: you want to **not** match the specific line.

  ▼

  Solution

  The `-i` command line option means edit “in place”. Finally, the right way to do this! The `’1,/So I tried/` specifies a range from the first line up to and including “So I tried”. The `!d` means things is this range will **not** be deleted but everything else will.

```
sed -i '1,/So I tried/!d' myfile.txt
```

For more information, reference the documentation for the sed command.

awk

The name `awk` is a combination of the first letters of the last names of the people who created the language - Alfred Aho, Peter J. Weinberger, and Brian Kernighan.

The `awk` command provides a way to search for a pattern, and perform actions on the found text. `awk` reads lines from the input file one at a time. The line is scanned for each pattern in the program and if there is a match the associated action is executed.

Either the pattern or the action can be omitted, but not both.
* If the pattern is omitted, then the action is performed for every line.
* If the action is omitted, then all lines that match the pattern will be printed out.

Take a look at the `man` pages for the awk command:

```
man awk
```

As you will likely notice, `awk` is very powerful and you can write complete programs to perform complex tasks using it.

A few built-in AWK variables (not a complete list)

Variable	Description
`$0`	The entire line - not including the newline at the end
`$1..$n`	The fields in a line (delimited by the field separator)
`FNR`	Current line number - just spans the current file
`FS`	Field separator - default is space
`NF`	Number of fields
`NR`	Current line number - spans multiple files
`RS`	Record separator - the default is newline

AWK special patterns

Rules	Description
`BEGIN`	Startup actions
`END`	Cleanup actions

Using `awk` to count the number of occurrences of the word “Pooh” in the text

`awk`	`'/Pooh/`	`{x++}END{print x}'`	`myfile.txt`
Command	Pattern	Action	Filename

The awk command is followed by the pattern (such as /Pooh/) and action (such as {x++}END{print x}) followed by the file name (such as myfile.txt)

```
awk '/Pooh/{x++}END{print x}' myfile.txt
```

Try these variations:

- Print the number of words in a file.

  ▼

  Solution

  Add up the number of “fields” on each line. The default field separator is space so this counts words and prints the total when it gets to the end of the line.

  ```
  awk '{ total = total + NF }; END {print total}' myfile.txt
  ```

- Print the number of lines in a file.

  ▼

  Solution

  The AWK variable `NR` contains the number of lines in a file so you just have to print it at the end.

  ```
  awk 'END{print NR}' myfile.txt
  ```

  - What happens if you don’t include the `END`?

Using `awk` to pull out items in field separated data

The `awk` command line option `-F` sets the field separator, the default is space.

Parse the comma-separated text and output two fields from each line. Don’t print the header line.

```
awk -F"," '{if (NR!=1){ print $1 " wrong answers " $5 " out of
       total " $8 }}' data.csv
```

Parsing comma separated text using the -F option and specifying which column using the $ (such as $5)

Try these variations:

- Print out just the second column of the file.

  ▼

  Solution

  You don’t need the if statement because you’re printing the entire column including the title line.

  ```
  awk  -F"," '{print $2}' data.csv
  ```

- Print out the number of answered assessments for “Jane Smith”.

  ▼

  Solution

  You use the same syntax you would use for any regular expression.

  ```
  awk  -F"," '/Jane Smith/{print $6}' data.csv
  ```

For more information, reference the documentation for the awk command

printf

The `printf` command formats and prints data. It works similarly to printf in C, C++ and Java. In case you are unfamiliar or need a reminder, here is a summary of the common parameters used with `printf`:

Parameter	Description
%d	integer number printed in decimal
%f	floating point number
%c	character
%s	string

Unlike `echo`, `printf` does not send an automatic newline at the end of the output. The `printf` command allows you to optionally assign the result to a variable rather than outputting to the screen (which is useful in Bash scripts).

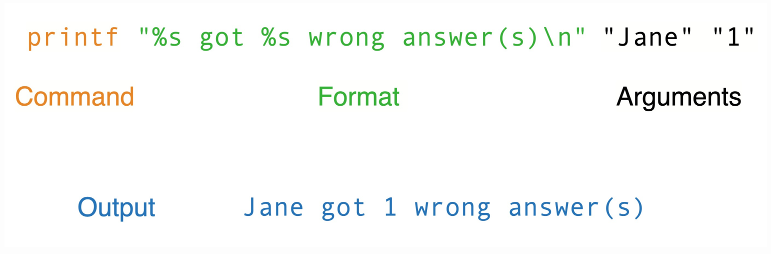
Take a look at the `man` pages for the `printf` command:

```
man printf
```

Let's take a look at the first syntax pattern (`printf format [arguments]`) which prints ARGUMENT(s) according to FORMAT.

Try the following example to print a formatted string:

```
printf "%s got %s wrong answer(s)\n" "Jane" "1"
```



```
printf "%s got %s wrong answer(s)\n" "Jane" "1"
```

```
Jane got 1 wrong answer(s)
```

The printf command can be followed by a format (such as "%s got %s wrong answer(s)") followed by arguments (such as "Jane" "1"). This example would output Jane got 1 wrong answer(s)

Try these variations:

- Print the result of a math equation.

  ▼

  Solution

  To execute a math expression in Bash you use a `$` followed by the expression enclosed in double parentheses. Use the `%d` to format a decimal number.

  ```
  printf "%d\n" $((8+4))
  ```

- Print only the first 4 digits beyond the decimal point of a floating point number.

  ▼

  Solution

  You can use the precision modifier `.` followed by a number to set the number of digits beyond the decimal point to display. Use the `%f` to format a floating point number.

  ```
  printf "%.4f\n" 3.1415926535
  ```

nano

Typing `nano` at the command line will bring up a no-frills text editor. You can specify a file name on the command line and that file will open.

`Ctrl-x` exits from the nano editor. Nano does not autosave, you must type `Ctrl-o`. The other commands that can be used with `nano` are listed at the bottom of the edit screen.

Take a look at the `man` pages for the `nano` command

```
man nano
```

You can specify `-B` or `--backup` on the command line if you want to create a backup version of the current file before changes are saved.

Using `nano` to create and edit files

Create a new file called `hello.txt` and enter the text `Hello World!` by following these steps:

1. `nano hello.txt`
2. Type `Hello World!`
3. Type `Ctrl-o` and then the Return/Enter key to confirm the file name.
4. Type `Ctrl-x` to exit.
5. Type `cat hello.txt` to print a listing of the file to the console.

vim

Vim is a free and open-source text editor controlled from the keyboard. Some versions of Vim do not support navigation with a mouse and it does not provide menus.

Take a look at the `man` pages for the `vim` command:

```
man vim
```

Typing `vim` at the command line will bring up the `vim` text editor. You can specify a file name on the command line and that file will open.

Overview of Vim navigation

Vim has different modes you can switch between. To add text to a file, you will need to be in **insert** mode. To end a mode, you press `esc` in the top-left of your keyboard.

important

The `:` in Vim

The `:` is used in Vim to enter command mode - always remember to hold down the shift key to type it - otherwise you will type a `;`

Here is a summary of mode switching

Key	Description
`i`	Enter insert mode
`:`	Enter command mode
`R`	Enter replace mode
`v`	Enter visual mode (highlighting)
`V`	Enter visual line mode (highlighting lines)
`esc`	Return to normal mode from insert or replace mode
`esc+esc`	Return to normal mode from command or visual mode

Aside from insert mode, Vim users also regularly use **command mode**. For example, Vim does not auto-save, you must type `:w` and click `Return` or `Enter`. The `:` enters command mode and the `w` specifies save. `Return` or `Enter` tell vim you have completed the command.

Here is a summary of commands - remember to press `:` to enter command mode and `Enter` or `Return` to end the command:

Key	Description
`w`	Save the current file
`wq`	Save the current file and close it, exits vim if no other open files
`w newname`	Save a copy of the current file as `newname` but continue editing the current file
`sav newname`	Save a copy of the current file as `newname` and edit `newname`
`q!`	Close a file without saving
`e filename`	Open another file
`x`	Write changes made to the current file and close it

Create and edit a file using vim

Create a new file called `testvim.txt` and enter the text `I'm learning about the vim editor` by following these steps:
1. Type `vim testvim.txt`
1. Type `i` to enter “insert” mode.
1. Enter the following text `I'm learning about the vim editor`
1. Press `esc` followed by `:wq` followed by the **Return** or **Enter** key to save and quit
1. Type `cat testvim.txt` to list out the file you created

For more information, reference the documentation for vim. There is also a quick reference.