SUMMER SCHOOL 2021

Variables and Loops

Dr Krishna K. Govender

Summary of commands used in this lecture

Command	Meaning
echo	Print text to screen
var=XX	Create a variable called var that stores XX
$var	Print the contents of the variable var
for var in XX; do …; done	Perform an iterative task
cat	Concatenate a file/s
history	Provides the terminal command history
|	Takes the output of one command and passes it as an input to another
grep	Used to find text in a file

- As in the previous lectures all commands provided in **BOLD** and center aligned from here onwards are commands that can be typed/executed directly in a terminal.

- Variables are used as place holders which can store information that you can make use of later.

  This is considerably important when making use of **bash scripting**, something that we will look further into later in this course.

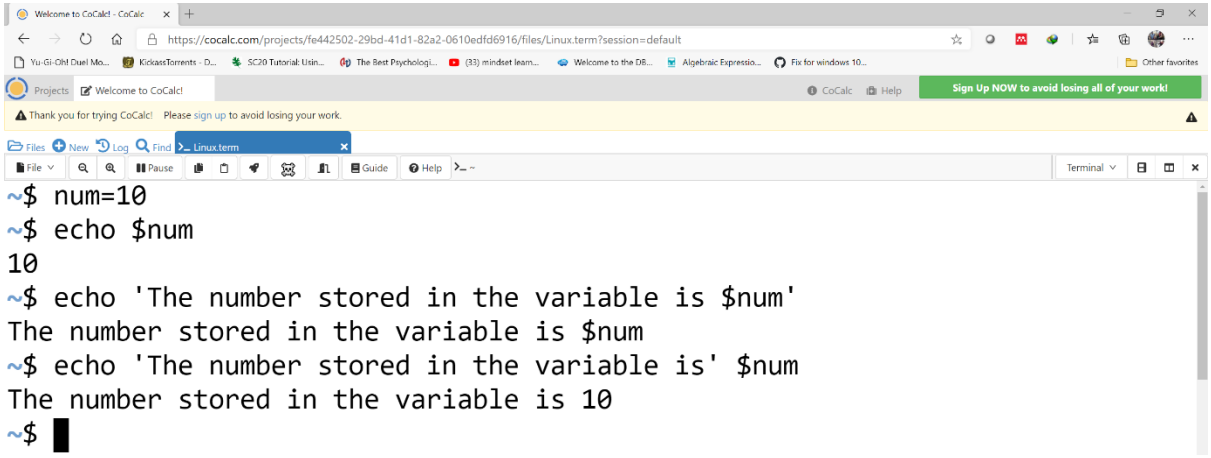
- We will create a variable called **num** that stores the number 10 and then use **echo** to print the variable to the screen:

num=10

echo $num

echo 'The number stored in the variable is $num'

echo 'The number stored in the variable is' $num



- Once a variable is declared it can be called up by making use of **$** followed by the variable name.

Note with Linux commands there should never be any spaces between the variable name, the equals sign and the value being assigned to the variable. If you include a space the variable will remain empty.
In addition, when you close a terminal and open a new one the variable no longer exists, so you will need to redeclare the variable.

- A variable can also be called together with text.

  The primary difference when calling up a variable with something like an **echo** statement is that it should be outside of the quotes.

  In the case above we see that keeping the variable in quotes produces **$num** instead of the actual number (10) that is stored inside of the variable.

- There can be multiple variables declared and combined:

num1=10

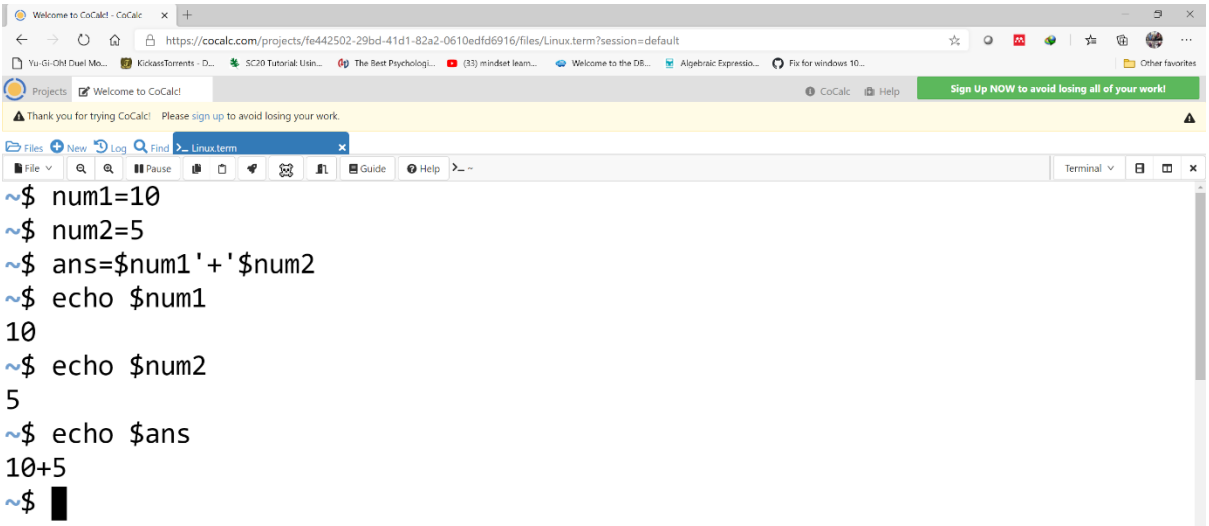
num2=5

ans=$num1'+'$num2

echo $num1

echo $num2

echo $ans



- We have two numbers (10 and 5) stored in variables **num1** and **num2**, while the variable **ans** shows **num1** and **num2** separated by an addition sign.

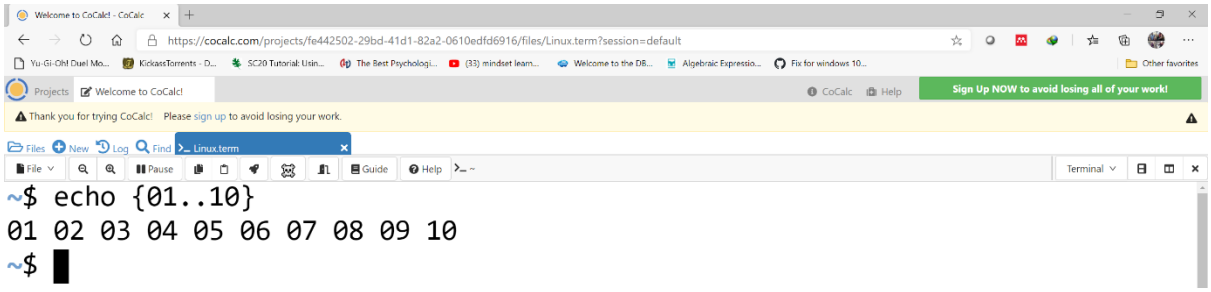
  It maybe more beneficial to have the variable **ans** store the mathematical solution to the addition of the two numbers (producing 15 when doing **echo $ans**), but we will look at how this can be done later in the course.

- There are often cases where you would like to repeat a task multiple times, but you don't wish to do the repetition manually.
  This is where making use of loops can be very beneficial.

- The construct of a bash for loop looks as follows:
  for variable **in** something; **do** task/command; **done**
  You need to have a variable (which, as mentioned above, is a place holder). Thereafter you **do** some task on the variable and when that task is complete the loop is **done**.

- First, we will try to print out the numbers 01 to 10 on the screen:

echo {01..10}



- **{01..10}** is a counter tool in Linux that will provide the numbers 01 to 10 in step sizes of 1.
  If this is printed to screen, we get all the numbers being displayed in a single row.

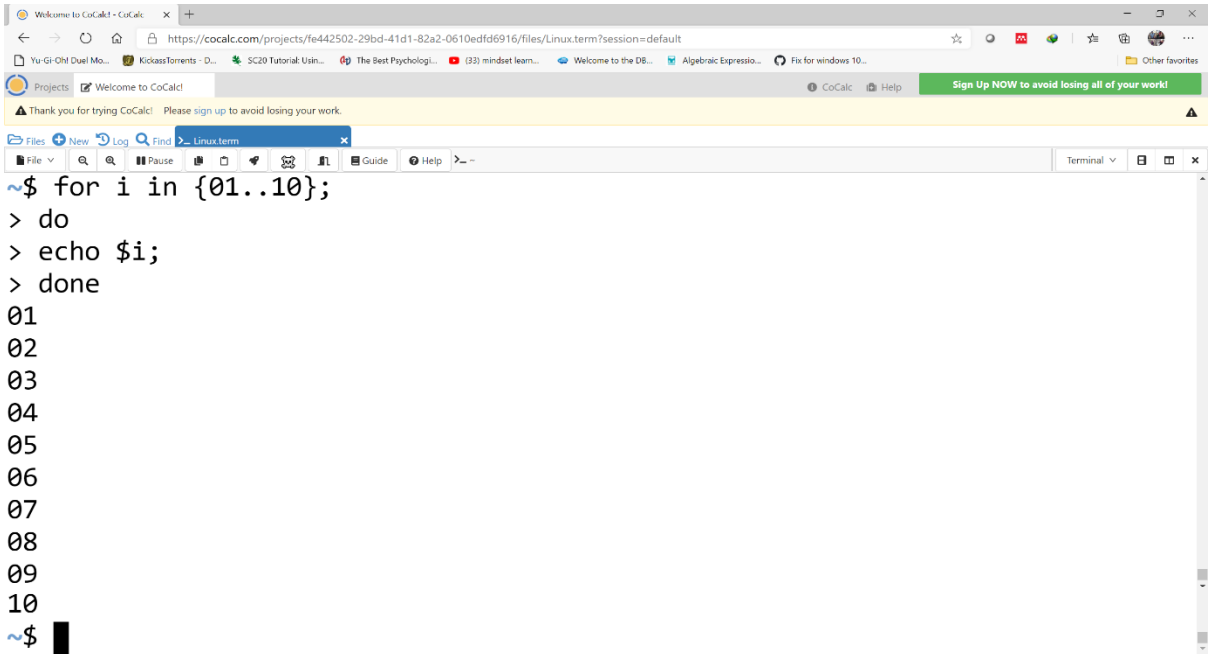
- Now if we wish to have the same thing printed as separate rows:

for i in {01..10};

do

echo $i;

done



- In the case above you will notice that there are lines with a **>** character. These **DO NOT** mean redirect, but instead they are showing you that you are inside a loop and when the loop is complete you are directed back to the terminal **~$**.

- The variable used above is called **i**. In the first iteration **i** takes on the value **01** and when we move into the loop all we **do** is print this value to the screen (**echo $i**).
  We then loop around (or go back) for a second iteration where **i** now gets the value **02** and this gets printed to the screen.
  The process continues until **i** gets the value of **10** and this gets printed to the screen.
  Thereafter the loop stops as we only requested that **i** go from 01 to 10.

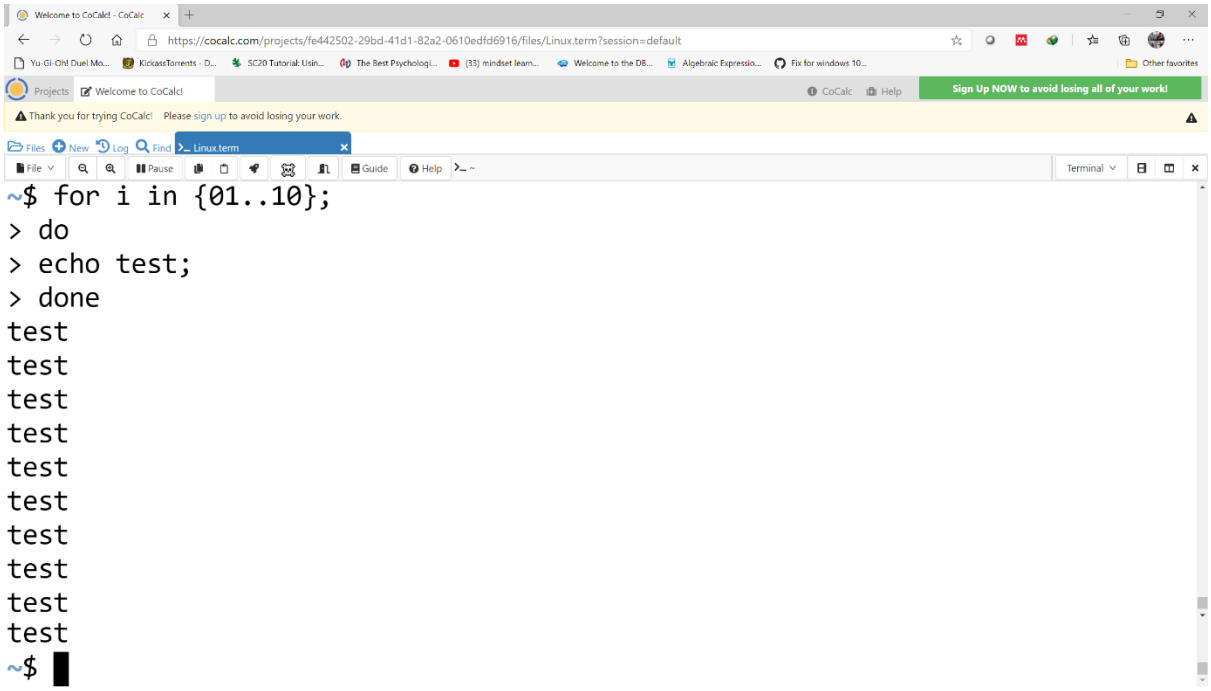
- Let us have a look at what happens if we choose not to print out the variable inside the loop:

for i in {01..10};

do

echo test;

done



- The loop does still work, but instead of printing out the numbers 01 to 10 on the screen all we see is the word test printed 10 times.

- Loops can also be used in conjunction with text instead of numbers:

for names in 'michael' 'christopher' 'jessica' 'matthew' 'ashley' 'jennifer' 'amanda' 'james';

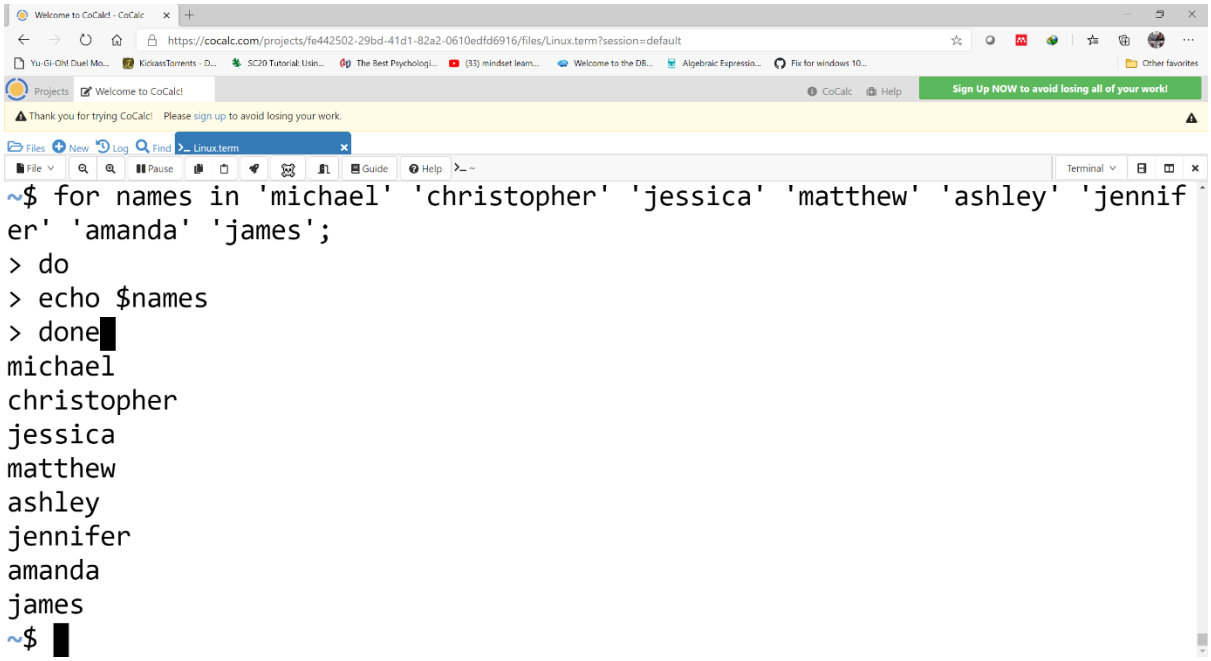
do

echo $names;

done

Note you can provide your variables with whatever name you like. They do not need to be the same as the names that I made use of.

Just remember that when you print this to screen, the name you provide after **for** must correspond to the name used after the **echo**.



- We now loop through individual names of people and print each name to the screen (**echo $names**).

- We can also iterate through information contained in a file:

echo 'line1' > somefile.txt

echo 'line2' >> somefile.txt

echo 'line3' >> somefile.txt

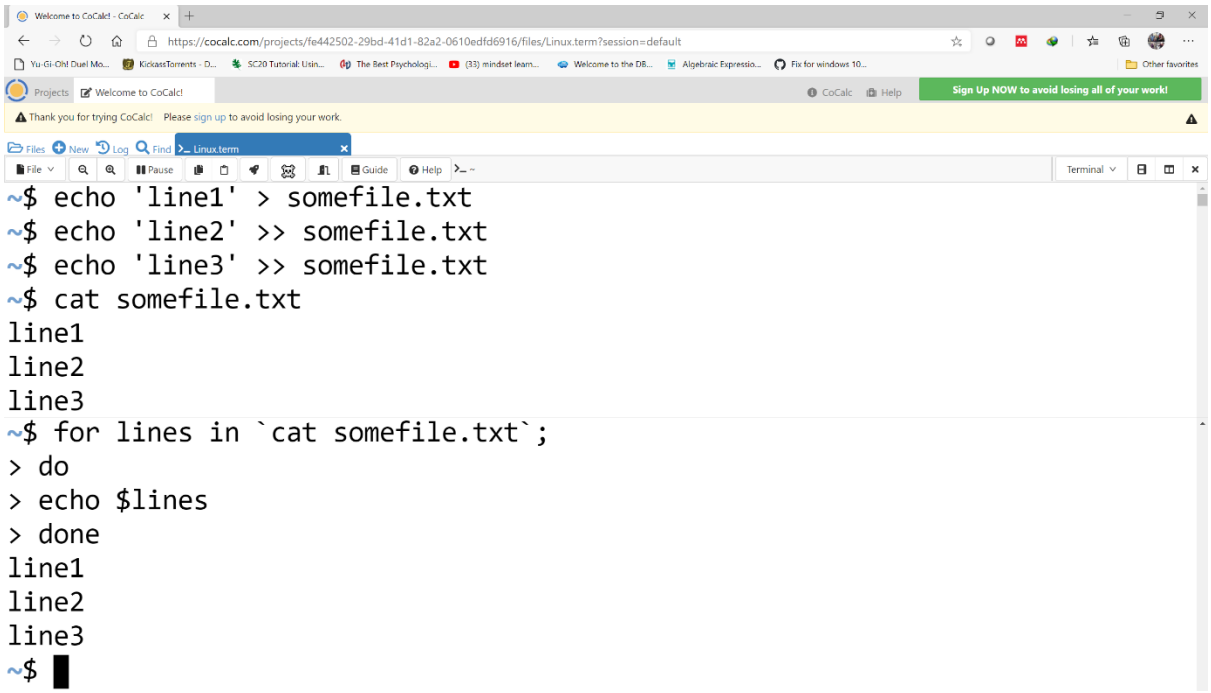
cat somefile.txt

for lines in `cat somefile.txt`;

do

echo $lines;

done



- After creating a file called **somefile.txt** with a few lines of text we can display the contents (**cat somefile.txt)**.

  Alternatively, we can also iterate through each row in **somefile.txt** using a **for loop**.

  This **for loop** is a little different to those that we have seen so far in that we need to make use of special ticks (``), located to the left of the number 1 on a conventional keyboard. These ticks tell Linux to execute the **cat** command on the terminal and place the result into the variable **lines**.

  Unfortunately, the output looks the same weather we use **cat** or a **for loop**.

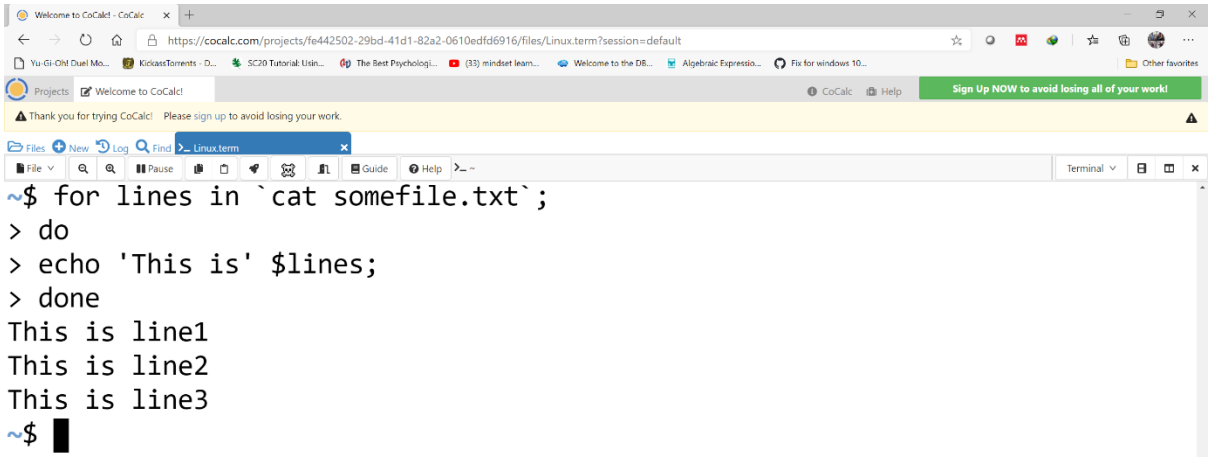
- What if we wished to have some text before each line:

for lines in `cat somefile.txt`;

do

echo 'This is' $lines;

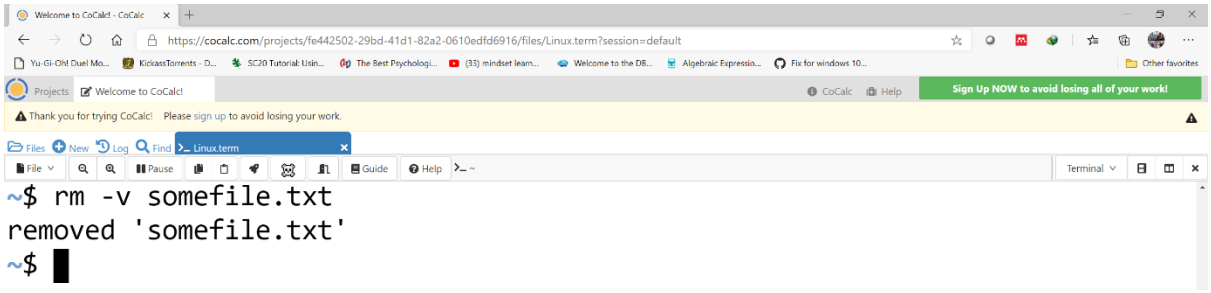
done



- Now the text “This is” appears on every line, which is not as easily achievable when not using a loop (such as with a combination of **echo** and **cat** for example).

- Remove the file somefile.txt:

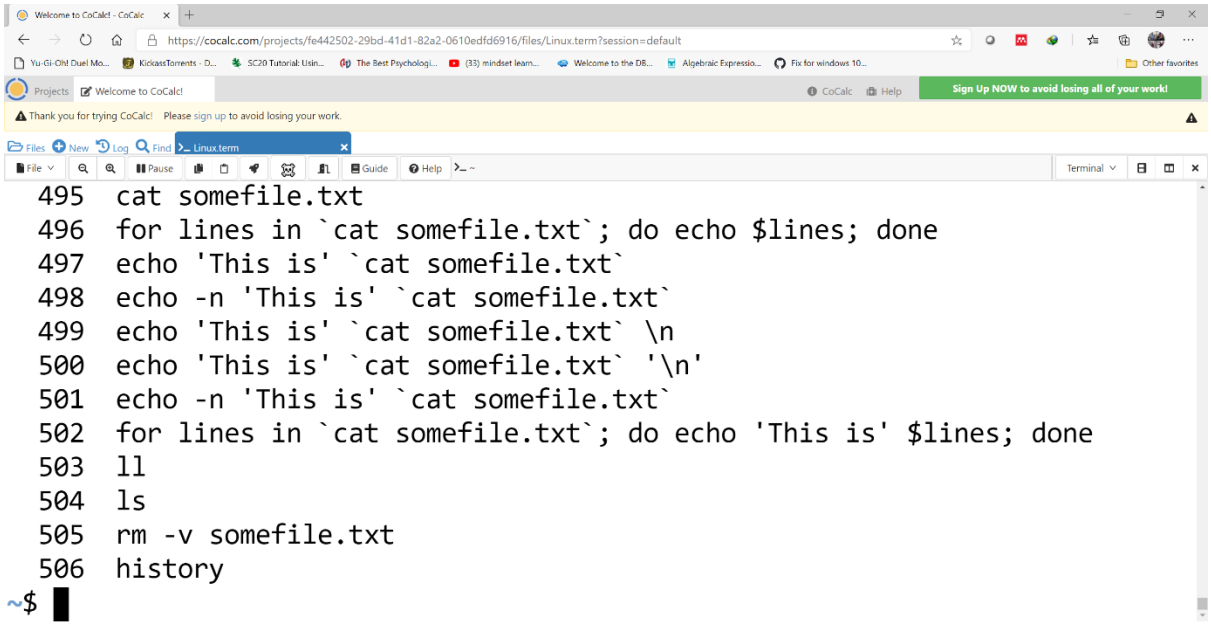
rm -v somefile.txt



- Since this is the last section before we begin **bash scripting** it is important that you take note of an important Linux command:

history

Note the number of lines that appear in your history will probably be different to mine.



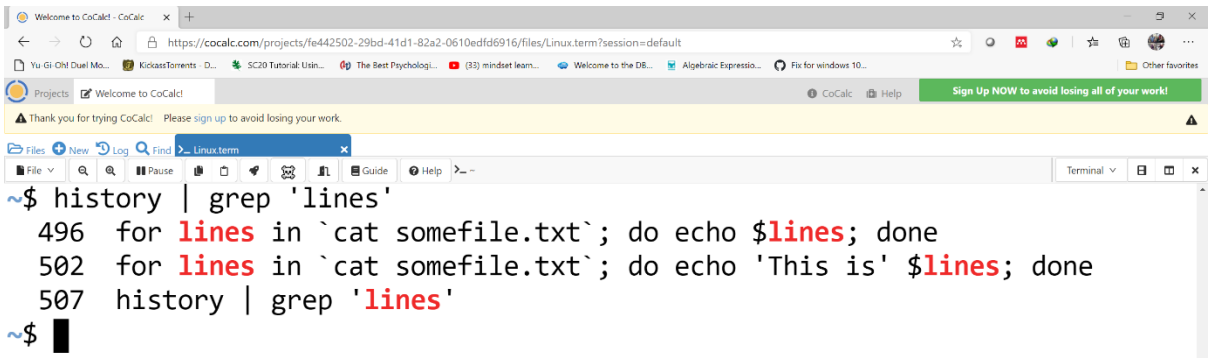
- This provides a list of all the commands that we have made use of up to this point.

  The numbers to the left correspond to the line number in the history file where a command was executed.

  In the case above at line 505 in my **history** I executed the command **rm -v somefile.txt**

- If there is a specific command you need to search for you can do so with the aid of **grep**:

  history | grep 'lines'



- Now only the commands that contain the word “lines” in our **history** are displayed.

- This brings us to the end of **Variables and Loops**.

- At this point you should give **Exercise 6** a try.