<https://www.tutorialspoint.com/unix_commands/expr.htm>

Before we can execute a script it must have the execute permission set (for safety reasons this permission is generally not set by default).

The shorthand 755 is often used for scripts as it allows you the owner to write or modify the script and for everyone to execute the script.

Example:

1. #!/bin/bash
2. # A sample Bash script, by Ryan
3. echo Hello World!

* **Line 1** - Is what's referred to as the **shebang**. See below for what this is.
* **Line 2** - This is a comment. Anything after **#** is not executed. It is for our reference only.
* **Line 4** - Is the command echo which will print a message to the screen. You can type this command yourself on the command line and it will behave exactly the same.

**The Shebang (#!)**

**#!/bin/bash**

This is the first line of the script above. The hash exclamation mark ( #! ) character sequence is referred to as the Shebang. Following it is the path to the interpreter (or program) that should be used to run (or interpret) the rest of the lines in the text file. (For Bash scripts it will be the path to Bash, but there are many other types of scripts and they each have their own interpreter.)

Formatting is important here. The shebang must be on the very first line of the file (line 2 won't do, even if the first line is blank). There must also be no spaces before the **#** or between the **!** and the path to the interpreter.

Whilst you could use a relative path for the interpreter, most of the time you are going to want to use an absolute path. You will probably be running the script from a variety of locations so absolute is the safest (and often shorter than a relative path too in this particular case).

It is possible to leave out the line with the shebang and still run the script but it is unwise. If you are at a terminal and running the Bash shell and you execute a script without a shebang then Bash will assume it is a Bash script. So this will only work assuming the user running the script is running it in a Bash shell and there are a variety of reasons why this may not be the case, which is dangerous.

You can also run Bash, passing the script as an argument.

1. bash myscript.sh

<https://ryanstutorials.net/bash-scripting-tutorial/bash-variables.php>

**Variables:**

**$1, $2, …**The first, second, etc command line arguments to the script.

**variable=value** To set a value for a variable. Remember, no spaces on either side of =

**Quotes " '** Double will do variable substitution, single will not.

**variable=$( command )** Save the output of a command into a variable

**export var1** Make the variable var1 available to child processes.

**Formatting** The presence or absence of spaces is important.

**Manageability** If a particular value is used several times within a script (eg a file or directory name) then using a variable can make it easier to manage.

Sed Command:-

SED command in UNIX is stands for stream editor and it can perform lot’s of function on

file like, searching, find and replace, insertion or deletion. Though most common use of

SED command in UNIX is for substitution or for find and replace. By using SED you can

edit files even without opening it, which is much quicker way to find and replace something

in file, than first opening that file in VI Editor and then changing it.

How Can We Perform Opreations :-

print file

$ sed -r ‘’ file.txt

For delete line : -

$ sed -r ‘2,4d’ # it will delete line beteween 2 to 4

For print lines:-

$ sed -n -r ‘2,4p’ file.txt # print line beteween 2,4

$ sed -r ‘/name/d’ file.txt # Delete a line that contain name

$ sed -n -r ‘/name/p’ file.txt # print a line that contain name

$ sed -r ‘s/name/sname/’ file.txt# that command replace name with the sname

$ sed -r ‘s/name/sname/g’ file.txt # that command replace globaly with name with sname

$ sed -r ‘s/name/sname/gi’ file.txt # that will be replace without case sensitive

$ sed -r ‘3,5 s/name/sname/gi’ file.txt # perform opration beteween line 3 to 5

$ sed -r ‘3, +3 s/name/sname/gi’ file.txt # perform opration start line number 3

+3

AWK command:-

Awk is a scripting language which is used for processing or analyzing text files. Or we

can say that awk is mainly used for grouping of data based on either a column or field ,

or on a set of columns. Mainly it’s used for reporting data in a usefull manner. It also

employs Begin and End Blocks to process the data.

some basic command to use awk :-

awk ‘{print $0}’ file.txt # to print all row and column

awk’{print $4}’ file.txt # to print the row number 4

awk ‘/deva/{print $0}’ file.txt # to print line that contain ‘deva’

awk ‘{print NR, $0 }’ file.txt # print line number

awk ‘{print NF, $0 }’ file.txt # print total line number in all rows

awk ‘{if (length($0) &gt; max==lentgth($0)} END {print $0} file.txt # to print maximum nuber

Exit Status:-

Every Linux or Unix command executed by the shell script or user has an exit status. Exit status is an

integer number. 0 exit status means the command was successful without any errors. A non-zero (1-

255 values) exit status means command was a failure.

0 --&gt; Sucessful

1 to 255 --&gt; error or Unsucessful

126 ---&gt; Permision Denied

127 ---&gt; Command not found

128 +n ---&gt; Kill Singnal Status