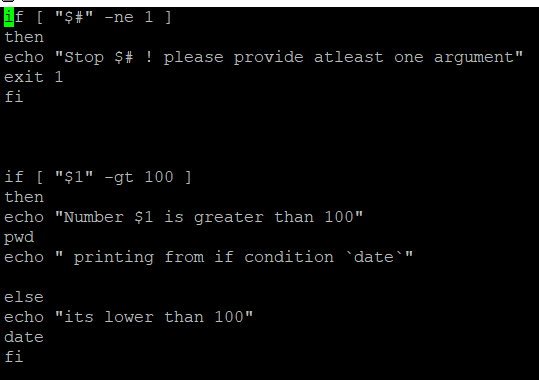
# IF else block usage

Simple IF example.

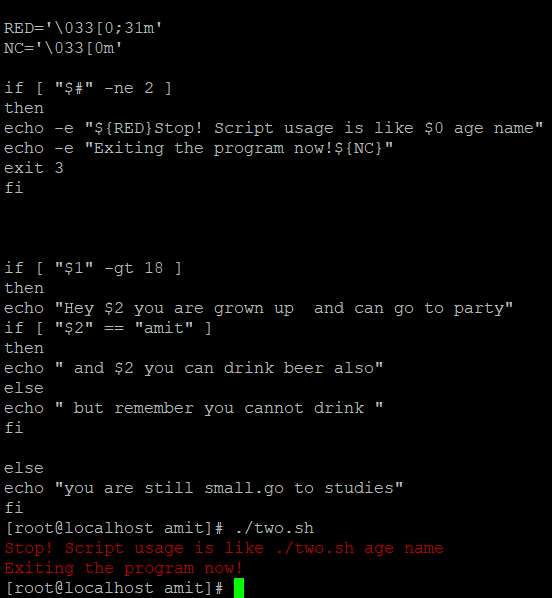


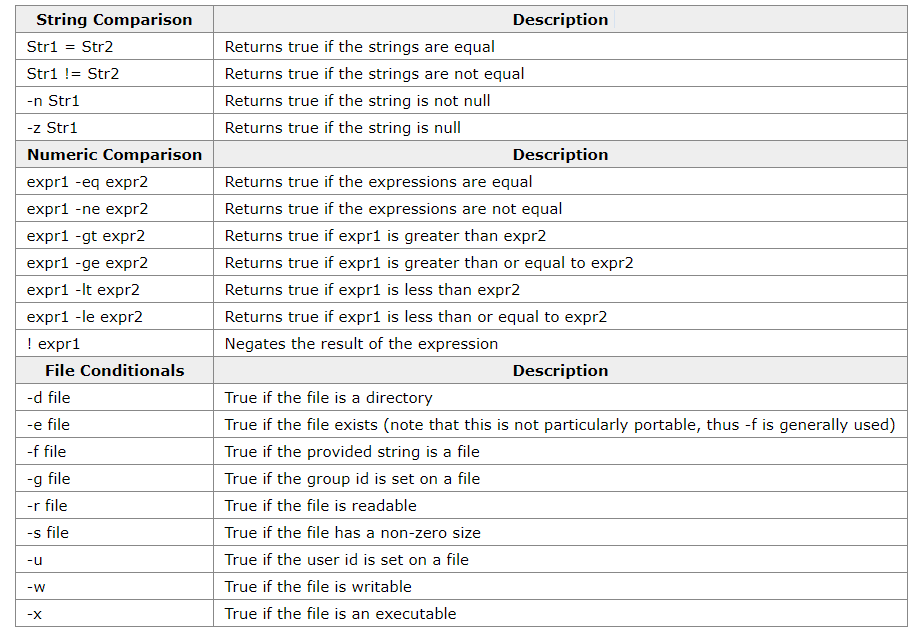
**$?** – returns 0 if the last command run was successful else non-zero value. It can be checked with echo $? Command.

**$#** - Here **$#** represent the total number of arguments.

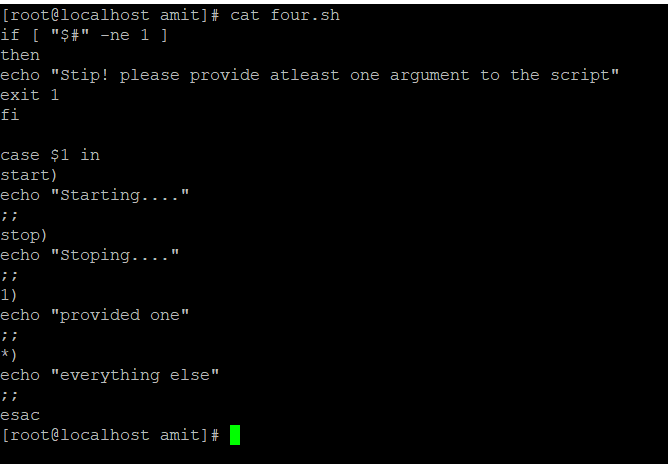
# Nested IF else

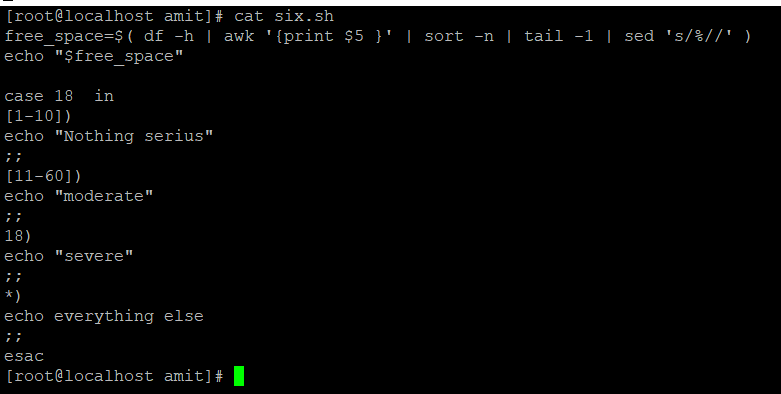
Here RED and NC are color constants which can be used in echo to print the error message in red





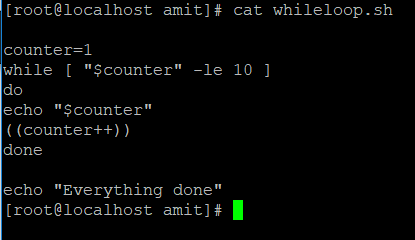
# Using Case





# Loop

## While Loop



## Until Loop

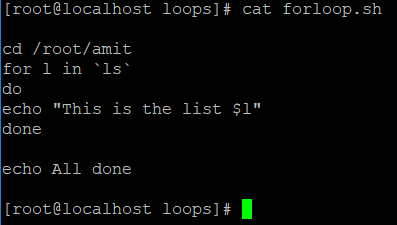
While and until loops are exactly same in syntax (replace while with until)

In while, loop iterate till condition become false

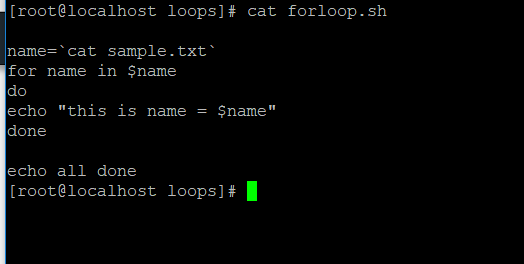
In until, Loop iterate until the condition become true.

# For Loop

Example 1 – Iterating in command output



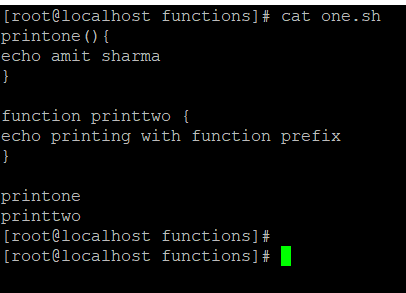
Example 2 – Iterating in file contents



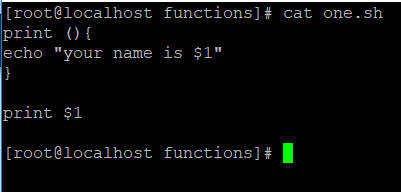
Break and continue works in the same was as java

# Function ()

Normal function declaration of two types

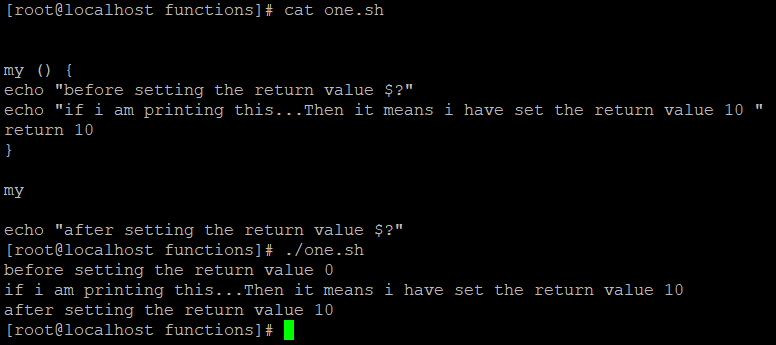


## Passing arguments



## Returning Value

In Unix, function don’t return any value like other programing languages

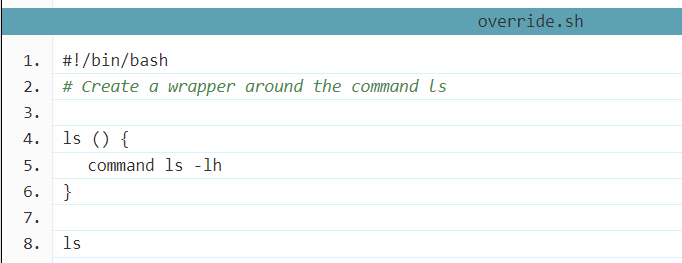
But we can set the return code (as in default case, for successful execution it is 0 and for failure its non-zero . It is represented by $?)

## Variable in Script

By default, all variables in script are global variables and can be accessed anywhere. If we want to create local variable inside the function the put **local** var\_name=”value”

## Overriding Commands

Put command keyword if you want to override some existing command of unix.



# Variables

## Unix Variables

* **$0** - The name of the Bash script.
* **$1 - $9** - The first 9 arguments to the Bash script. (As mentioned above.)
* **$#** - How many arguments were passed to the Bash script.
* **$@** - All the arguments supplied to the Bash script.
* **$?** - The exit status of the most recently run process.
* **$$** - The process ID of the current script.
* **$USER** - The username of the user running the script.
* **$HOSTNAME** - The hostname of the machine the script is running on.
* **$SECONDS** - The number of seconds since the script was started.
* **$RANDOM** - Returns a different random number each time is it referred to.
* **$LINENO** - Returns the current line number in the Bash script.

**Export variable** - To export a variable **export** var. This will make the variable available across shell.

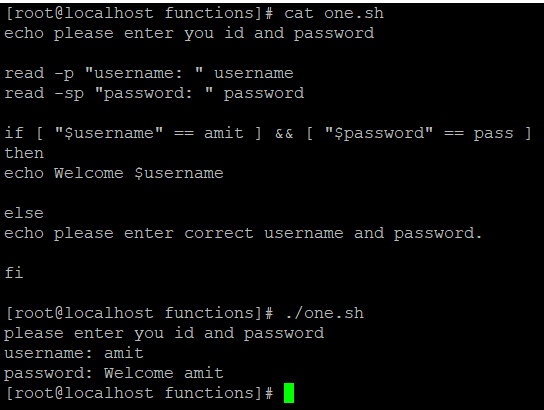
## Command Substitution Variables

Command substitution allows us to take the output of a command or program (what would normally be printed to the screen) and save it as the value of a variable. To do this we place it within brackets, preceded by a $ sign.

1. myvar=$(ls /etc | wc -l)

# Input command **read**

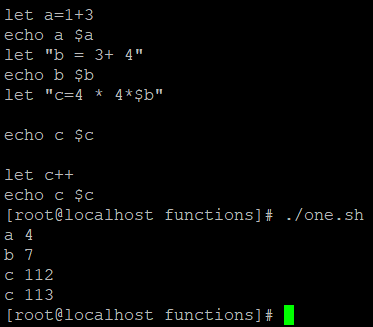
**Read** command is used to take the input from the user on runtime. This usually comes handy if we silently (-s) use it to take users password



# Arithmetic expressions

## Let

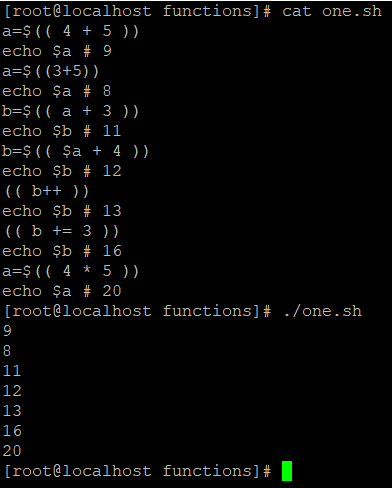
KeyWord **let** give you flexibility to perform basic mathematical operations Just say **let** “ put any mathematic operation here regardless of space”



# Double paranthesis expression

We can calculate any expression using double brackets like below.

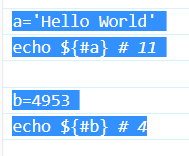
$(( expression ))



## Find length of a variable

We can find the length of variable as well.

${#variable}



# How to compress and decompress a file

To tar a file or folder

*tar -cvf file\_name (this c means create, v means verbose, and f means file)*

**untar**

*tar -xvf file\_name (here x means extract)*

**to gzip a tar file (compress)**

*gzip file\_name.tar*

**to decompress**

*gunzip file\_name.tar.gz*

**Important** - **here tar only group the files into a single file and does not actually decrease the size(keeps sames size as the actual size)**

**when we do the gzip of the tar file, this compress the file)**

# How to do tar and gz in a single command

**Compress**

tar -zcf file\_name ( this converst the file into file\_name.tgz (which means file\_name.tar.gz)

**To decompress**

tar -zxvf file\_name

# Vi Editor Command

dd => delete the current line. If number precedes dd, then those many lines will be deleted. e.g. 10dd will delete 10 lines from current position.  
cc => change entire line  
yy => yank/copy the line in buffer. It can be placed in the file using p option. If number precedes yy, then those many lines will be copied in buffer. e.g. 10yy will copy 10 lines in buffer, which can be put into file anywhere using p.  
cw => change word.  
ce => change word.

:g/^$/d => delete all the blank lines

0 or ^ => to reach the beginning of line

$ => takes you the end of the line

gg => to reach the beginning of the file

G=> to reach end of file  
ZZ => save changes and exit the file.

# PS command

Ps command and top command are same.ps is just the snapshot at one time while top always gets update with refreshed.

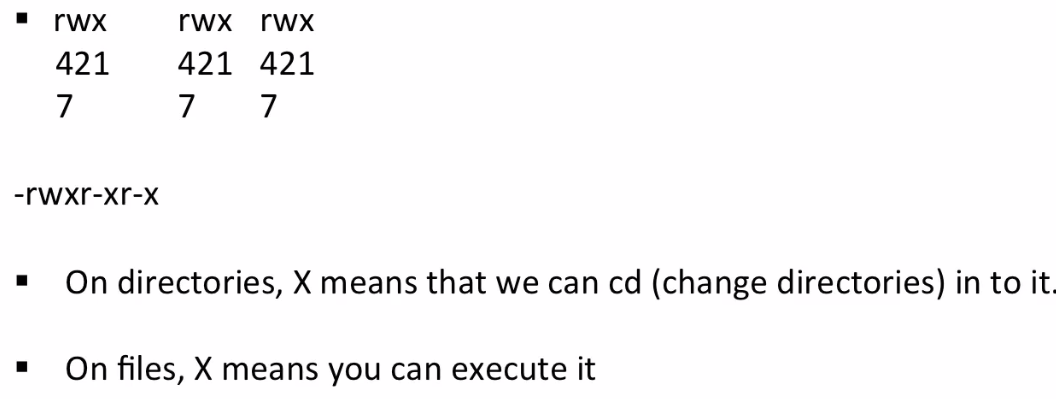
ps aux – this gives all the processes with users

top (this command can also be used to kill some process by hitting k and process id)

# File Permissions

**drwxrwxrwx** 4 root root 96 Sep 16 16:19 file\_comp

First character represents the directory or file



# Links

### Hard Link

### Soft Link (Symbolic links or sim links)

## Hard Link

Hard link has got the same inode value as file. Hence even if the file is removed, hardlink can access the file because it check the reference address.

Hard link cannot be created for directory. This is to avoid recursive loops.

Hard Links don’t work across file systems.

If original file is removed then the link will still show the content of the file.

**Command to create a hard link is:**

**$ ln [original filename] [link name]**

## SoftLinks

A soft link like window shortcut. If you remove the main file, soft link will stop working.

Soft Link contains the path for original file and not the contents. A soft link can link to a directory

Softlink can work across file systems.

**Command to create a Soft link is:**

**$ ln -s [original filename] [link name]**

# SED Command

SED (Stream Editor) command in UNIX is used 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.

**For Replacing and substituting String**

**$sed 's/unix/linux/' geekfile.txt** – this will by default replace only first occurrence of unix with linux.

**$sed 's/unix/linux/2' geekfile.txt** – This will replace the first 2 occurrence of unix

**$sed 's/unix/linux/g' geekfile.txt** – This will replace all occurrence.

**$sed '3 s/unix/linux/' geekfile.txt** - Replacing string on a specific line number

**$sed -n 's/unix/linux/p' geekfile.txt -** Printing only the replaced lines

**$sed '1,3 s/unix/linux/' geekfile.txt** - Replacing string on a range of lines

## Deleting lines from a particular file

SED command can also be used for deleting lines from a particular file. SED command is used for performing deletion operation without even opening the file

**Examples**:

**1. To Delete a particular line say n in this example**

Syntax:

$ sed 'nd' filename.txt

Example:

$ sed '5d' filename.txt

**2. To Delete a last line**

Syntax:

$ sed '$d' filename.txt

**3. To Delete line from range x to y**

Syntax:

$ sed 'x,yd' filename.txt

Example:

$ sed '3,6d' filename.txt

**5. To Delete from nth to last line**

Syntax:

$ sed 'nth,$d' filename.txt

Example:

$ sed '12,$d' filename.txt

**6. To Delete pattern matching line**

Syntax:

$ sed '/pattern/d' filename.txt

Example:

$ sed '/abc/d' filename.txt