Devops notes

1. Linux commands
2. Shell script
3. Git(SCM)
4. Jenkins(CI/CD)
5. AWS(Cloud)
6. Ansible(CD)
7. Docker
8. Portainer
9. K8(Kubernetes)
10. Terraform(IAC)
11. JIRA
12. Prometheus
13. Grafana(MT)
14. Python(Scripting)
15. Maven(Build tool)
16. NPM, nodeJS

Contents

[Introduction: 5](#_Toc198920775)

[Linux commands: 5](#_Toc198920776)

[1. pwd 5](#_Toc198920777)

[2. ls 5](#_Toc198920778)

[3. mkdir <directory name> 5](#_Toc198920779)

[4. touch <file\_name> 5](#_Toc198920780)

[5. vi <filename> vim <filename> 5](#_Toc198920781)

[6. cat <filename> 5](#_Toc198920782)

[7. cd <directory name> or cd <path> 6](#_Toc198920783)

[8. commands inside VI editor 6](#_Toc198920784)

[9. rm <filename> 6](#_Toc198920785)

[10. wc <filename> 6](#_Toc198920786)

[11. echo 6](#_Toc198920787)

[12. > [Redirect] 6](#_Toc198920788)

[13. >> [append] 6](#_Toc198920789)

[14. du -sh <filename> , du -sh <directory> 7](#_Toc198920790)

[15. df -h 7](#_Toc198920791)

[16. head <filename> 7](#_Toc198920792)

[17. tail <filename> 7](#_Toc198920793)

[18. | [pipe] 7](#_Toc198920794)

[19. free -m 7](#_Toc198920795)

[20. clear or ctrl + l 7](#_Toc198920796)

[21. uniq & sort 7](#_Toc198920797)

[22. chmod [change mode] 7](#_Toc198920798)

[23. umask (don’t practice) 8](#_Toc198920799)

[24. cp [copy] 8](#_Toc198920800)

[25. mv <file> <f1> 8](#_Toc198920801)

[26. chown <newname> <file> 8](#_Toc198920802)

[27. sed [stream editor] 8](#_Toc198920803)

[28. cut 9](#_Toc198920804)

[29. awk 9](#_Toc198920805)

[30. grep 9](#_Toc198920806)

[31. find 9](#_Toc198920807)

[32. xargs 10](#_Toc198920808)

[33. ps [process] 10](#_Toc198920809)

[34. service service\_name start/stop/restart 10](#_Toc198920810)

[35. kill -9 <PID> 10](#_Toc198920811)

[36. links: 10](#_Toc198920812)

[37. uname -a 10](#_Toc198920813)

[38. sleep NUMBER[SUFFIX] 11](#_Toc198920814)

[39. Export 11](#_Toc198920815)

[40. Hostname 11](#_Toc198920816)

[41. ctrl + c , ctrl +z , bg ↵ 11](#_Toc198920817)

[Server commands 11](#_Toc198920818)

[42. SSH[secure shell] 11](#_Toc198920819)

[43. scp: >server copy 11](#_Toc198920820)

[44. ping IP\_address/hostname 11](#_Toc198920821)

[45. netstat -na 12](#_Toc198920822)

[46. top 12](#_Toc198920823)

[47. uptime 12](#_Toc198920824)

[48. rsync 12](#_Toc198920825)

[Shell Script 13](#_Toc198920826)

[basic script to display some meaning full message 13](#_Toc198920827)

[Input arguments 13](#_Toc198920828)

[script to pass input arguments in command line and display message 14](#_Toc198920829)

[pass input arguments in command line, store in local variable and display message 14](#_Toc198920830)

[Conditional statements (if, if-else, if-elif-else) 14](#_Toc198920831)

[Equality operators in shell script (=, ==, !=, -eq , -ne, -lt, -le, -gt, -ge) 15](#_Toc198920832)

[script to read a number, check if it is equal to 5 and display 15](#_Toc198920833)

[script to find biggest of two numbers: 15](#_Toc198920834)

[special characters of global variables: 16](#_Toc198920835)

[script to restrict two i/p arguments in finding biggest of two 16](#_Toc198920836)

[Looping statements (for loop, while loop) 16](#_Toc198920837)

[nohup[no hang up] 16](#_Toc198920838)

[Script to find biggest of three numbers 17](#_Toc198920839)

[Script to add, sub, multiplication and divide two numbers 17](#_Toc198920840)

[Script to find factorial of a number 17](#_Toc198920841)

[Script to add given set of numbers 18](#_Toc198920842)

[Script to check given name is a file, directory or not exist 18](#_Toc198920843)

[script to list files having a <pattern>, display message if pattern not found in any file 18](#_Toc198920844)

[script to count no of words, characters in each line 19](#_Toc198920845)

[Script to display name of employees whose age is greater than selection 19](#_Toc198920846)

[Script to check disk m/m, if crosses threshold, send the mail notification! 20](#_Toc198920847)

[cron jobs: crontab 20](#_Toc198920848)

[Script to check services, trigger notification if any service is stopped state 20](#_Toc198920849)

[cleanup script: script to retain recent 20 builds 21](#_Toc198920850)

[Git: 22](#_Toc198920851)

[1. git init 22](#_Toc198920852)

[2. git config 22](#_Toc198920853)

[3. git clone <url> 22](#_Toc198920854)

[4. git status 22](#_Toc198920855)

[5. git add <filename> 22](#_Toc198920856)

[6. git commit -m "message" 23](#_Toc198920857)

[7. git push 23](#_Toc198920858)

[8. git revert 23](#_Toc198920859)

[9. git reset 23](#_Toc198920860)

[10. git log 23](#_Toc198920861)

[11. git pull 23](#_Toc198920862)

[12. git tag <tag\_name> 23](#_Toc198920863)

[13. git branch 23](#_Toc198920864)

[14. git merge <branch> 24](#_Toc198920865)

[15. git checkout 24](#_Toc198920866)

[16. git switch 24](#_Toc198920867)

[17. git stash 25](#_Toc198920868)

# Introduction:

Windows uses GUI(graphical User Interfaces) whereas LINUX uses CLI(Command Line Interface). To begin with, search “git for windows”, open the website <https://git-scm.com/downloads/win> download and install git for windows, create and open work space folder on desktop, right click and select open Git Bash here. Then start typing following commands.

# Linux commands:

## pwd

present working directory: terminal displays path of present working directory

ex. pwd ↵

output: /c/Users/asus/Desktop/Devops/shell

## ls

list of files and directories in pwd (similar to files and folders structure in windows)

* 1. ls : list of files and directories in pwd
  2. ls -l : list of files and directories with access details, date and time created in alphabetical order
  3. ls -lt : with latest to oldest file created
  4. ls -ltr : in reverse order
  5. ls \*/\*/….. to list complete list of parent directory
  6. ls -a >display hidden files
  7. ls -I >to check inode of a file

## mkdir <directory name>

to create directory/s in pwd

* 1. mkdir dir1
  2. mkdir d1 d2 d3 d4 ....

## touch <file\_name>

to create file/s in pwd

* 1. touch file1
  2. touch f1 f2 f3 f4 ....

## vi <filename> vim <filename>

vi editor is an editor on linux(just like ms-word, txt, notepad, vs in windows), vim is improved version, following commands to be used inside vi editor, to copy content from bowser(shift + insert/right click + paste)

* 1. press 'i' for insert mode
  2. press 'Esc' + :wq! --> save file
  3. press 'Esc' + :q! --> without save

## cat <filename>

to display content of a file in terminal

## cd <directory name> or cd <path>

change directory used to change to pwd to required directory

* 1. cd <dir1>, will change to the directory within pwd
  2. cd /c/Users/asus/Desktop/Devops/shell, by using path
  3. cd .. back from pwd
  4. cd ../../.. back from pwd three step

## commands inside VI editor

* 1. :/<pattern> search for a pattern(word) ex :/devops, :/development
  2. :%s/<current\_pattern>/<change\_pattern>/gi : search and replace a pattern
  3. :%s/DevOps/testing/gi , in whole file
  4. :1s/DevOps/testing/gi , first line
  5. :5,$s/testing/development/gi , fifth word till end
  6. :4,10s/dev/test/gi , 4th to 10th line
  7. :dd , delete whole line
  8. :set nu , set numbers inside file

## rm <filename>

command to remove a file or directory

* 1. rm <filename>
  2. rm f1 f2 f3 f4 .....
  3. rm -r <directory> (-r used as recursive to remove files inside directory one by one)
  4. rm -rf <directory> (-rf recursive forcefully, to remove files opened as well inside dir)
  5. rm -rf d1 d2 d3 ....

## wc <filename>

* 1. wc , word count command
  2. wc -l <file> , count lines
  3. wc -w f1 , count words
  4. wc -m f1 , count characters
  5. wc -c f1

## echo

to write <something> on terminal

* 1. echo "This is devops class"
  2. echo -e "This is devops class \nStarted on April” , \n used to next line

## > [Redirect]

Redirect is used to overwrite existing file, if file not there, it’ll create new file

* 1. echo "Testing" > f1 , wil write testing in file “f1”
  2. ls -l > f2

## >> [append]

append is used to add content to existing file, if file not there, it’ll create new file

* 1. echo "Testing" >> f1
  2. ls -ltr >> log

## du -sh <filename> , du -sh <directory>

displays memory of files or directory

* 1. du -sh \* , displays memory of all files and directories in pwd

## df -h

displays memory status of all drives mounted in the server/system

* 1. df -h . , displays current drive

## head <filename>

used to display content of a file from top (default ten lines)

* 1. head -3 <filename> , displays top three lines of a file

## tail <filename>

used to display content of a file from boom (default ten lines)

* 1. tail -1 <filename> , displays bottom first line of a file

## | [pipe]

Pipe is used to join/bridge content from previous command

* 1. ls | wc -l , list of files + number of lines, displays number of files in pwd
  2. head -20 f1 | tail -1 , top 20 lines in f2 + bottom 1st of top 20, display’s 20th line in f2

## free -m

displays ram memory in MB

* 1. free -g, displays ram memory in GB
  2. free -b , displays ram memory in Bytes

## clear or ctrl + l

clears the terminal

## uniq & sort

* 1. sort <filename> , arrange content of a file in ascending order line by line
  2. sort -r <filename>, arrange content in ascending order
  3. uniq <filename> , will remove consecutive duplicates
  4. sort <f1> | uniq , arrange content + remove duplicates (in a file)

## chmod [change mode]

this command is used to manage permissions of a file/directory, to check permissions, use ls command (ex. -rw-r--r-- 1 asus 197121 43 May 7 20:33 f1)

|  |  |
| --- | --- |
| 2^0--> 1, 2^1--> 2, 2^2 --> 4  Owner group others  rwx rwx rwx  r --> read  w --> write  x --> execute  22 21 20  4 2 1  r w x  owner group other  rwx r-x r--  7 5 4 | 1 only execute (001)(--x)  2 only write (010) (-r-)  3 write and execute (011) (w-x)  4 only read (100) (r--)  5 read & execute (101) (r-x)  6 read and write (110) (rw-)  7 read, write & execute(111) (rwx)  666  rw-rw-rw-  rwxrw-r-x  765 |

* 1. chmod 777 <f1> (rwxrwxrwx) read, write & execute access for user, group and others
  2. chmod 666 <directory> (rw-rw-rw-) read & write access for user, group and others
  3. chmod 765 f1 f2 f3 f4 .. (rwxrw-r-w)
  4. chmod -R 644 <d1> (recursive) permission applicable for files & directories inside d1

## umask (don’t practice)

umask is used to set default directory permissions during initial server creation, its gives opposite permissions as compared to of chmod(i.e. 000 gives full permission instead of 777)

in chmod rw- r-- r-- chmod 644

in umask --x -wx -wx umask 133

* 1. umask 000 (rwxrwxrwx) read, write & execute access for user, group and others
  2. umask 777 (---------) no access for user, group and ohers

## cp [copy]

used to copy and paste content of one file/directory to another, destination content is overwritten by source, if destination file doesn’t exist, new file is created

* 1. cp <file1> <file2>
  2. cp -r <dir1> <dir2>
  3. cp -p f1 f4 , destination file copies content and permissions from source
  4. cp file /home/dir1/dir2/f1
  5. cp file /home/dir1/dir2/
  6. cp -i f1 f5 , interactive mode, will ask to overwrite content of destination from source

## mv <file> <f1>

used to move file/directory to another directory if destination directory is same, this command acts as rename.

* 1. mv <file> /home/test/
  2. mv <d1> /home/test/d1

## chown <newname> <file>

this command is used to change ownership of a file/directory

* 1. chown developer file , transfer file’s ownership to developer
  2. chown test directory , transfer direcotory’s ownership to test
  3. chown dev:group1 dir1, transfer dir1’s ownership to dev of group1
  4. chown test:test\_group file

## sed [stream editor]

sed command will take content from file and changes made in terminal directly(not using vi) without affecting parent file

* 1. sed 's/pattern/new\_pattern/gi' f1 ,pattern to new pattern in f1(i->case insensitive)
  2. sed -i 's/testing/dev/gi' f3 , -i interactive mode
  3. sed '2s/pattern/new\_pattern/gi' filename , pattern change in 2nd line only
  4. sed '5,10s/pattern/new\_pattern/gi' filename , change in 5th and 10th line only
  5. sed '20,$s/pattern/new\_pattern/gi' filename , change from 20th till end of file
  6. sed -n '5p' filename , to display only 5th line of file
  7. sed -n '4,10p' f2 , to display 4th to 10th line
  8. sed -n '6,$p' f2 , to display 6th till end of file
  9. sed '3d' f1 , delete 3rd line of file
  10. sed '2,6d' f3 , delete 2nd and 6th line of file
  11. sed '10,$d' f3 , delete 10th to last line

## cut

cut is used to display column wise segregated content(separated by a delimiter), cut have limitation if there are more than specified delimiter(ex. extra space)

* 1. cut -d " " -f1 file , displays 1st column separated by <space>(delimiter)
  2. cut -d "\*" -f2 file , displays 2nd column separated by <\*>(delimiter)
  3. cut -d " " -f2,5,7 file , displays 2nd 5th and 7th column separated by <space>(delimiter)

## awk

awk is used to display column wise segregated content(separated by a delimiter), awk will ignore if there are extra spaces in segregating columns

* 1. awk -F " " '{print $1}' filename , displays 1st column separated by <space>
  2. awk -F " " '{print $3}' filename , displays 3rd column
  3. awk -F " " '{print $NF}' filename , displays last column
  4. awk -F " " '{print $(NF-1)}' filename , displays last but one column
  5. awk -F " " '{print $2,$4}' filename , displays 2nd and 4th column

## grep

grep is used to search a pattern

* 1. grep -i "linux" <filename> , displays line containing linux in file, -i -> case insensitive
  2. grep -in "Windows" f2 , displays line number containing Windows in f2 (-n)
  3. grep -ic "Windows" f2 , counts number of “windows” in f2 (-c)
  4. grep -iw "win" f2 , display only “Win” not windows (-w)
  5. grep -l "linux" \* , display files containing linux in all files and directories
  6. grep -lR "linux" \* , for recursive search
  7. grep -ie "pattern1" -ie "pattern2" filename , multiple pattern search

## find

is used to find location of file or directory

* 1. find . -iname "f1" used to find file location in PWD
  2. find /home/test -iname "f1" used to find in destination path
  3. find . -type f -mtime +10 >file before 10 days
  4. find . -type d -mtime +10 > directories before 10 days
  5. find . -type f -mtime -10 > file within 10 days
  6. find . -type d -mmin +5 > directories before 10 mins
  7. find . -type f -perm 0777 > find file and directories with particular permissions
  8. find . -type d -perm 0755
  9. find . -perm 0766
  10. find . -type f -empty >find empty files
  11. find . -type f -not -empty >find non empty files
  12. find . -maxdepth 2 -iname "file1" find file for till 2nd level
  13. find . -maxdepth 5 -iname "file1"

## xargs

argument is used to pass output of a particular command as an argument to next command

* 1. find . -type -f -empty|xargs rm -f >remove empty files
  2. find . -type d -mtime +5|xargs rm -rf >remove directories created before 5 days

## ps [process]

this is like task manager in windows, it lists out all processes running in PC

* 1. ps -ef >lists all processes
  2. ps -ef|grep -i "chrome" >lists all processes running under chrome
  3. ps -ef|grep -ie "chrome" -ie "edge" >lists all processes running under chrome & edge
  4. ps -u "username" >processes under user

## service service\_name start/stop/restart

this command is used to gracefully start/stop/restart an application

* 1. sudo service apache2 start
  2. sudo service apache2 stop
  3. sudo service apache2 restart
  4. sudo service nginx stop

## kill -9 <PID>

kill command will forcefully stops a process(like terminating task in task manager of windows)

* 1. kill -9 7684 >here 7684 is PID number of a task obtained using Ps -ef command

## links:

links are shortcuts to a file/directory, if you make any changes in original file, it’ll get reflected in links

* 1. ln -s f1 <softlink> > creates softlink of file1
     1. ln -s /home/test/f1 softlink1 > creates softlink of f1
     2. softlink / symbolic link / sym link:
     3. delete the orginal file symlink won't work
  2. ln f1 <hardlink> >creates hardlink of f1
     1. if we delete original file hardlink still work
     2. its points to 'i' node of a file
     3. ln /home/test/f1 hardlink1 >creates hardlink of f1

## uname -a

* 1. -a, --all > print all information
  2. -s, --kernel-name
  3. -n, --nodename > print the network node hostname
  4. -r, --kernel-release
  5. -v, --kernel-version
  6. -m, --machine
  7. -p, --processor
  8. -i, --hardware-platform
  9. -o, --operating-system

## sleep NUMBER[SUFFIX]

The sleep command in a shell script pauses execution for a specified duration. By default, the sleep command interprets numerical arguments as seconds. It also accepts suffixes to specify other time units.

* 1. sleep 5 🡪sleep for 5 seconds
  2. sleep 2.5 🡪sleep for 2.5 seconds
  3. sleep 30m 🡪sleep for 30 minutes
  4. sleep 2h 🡪sleep for 2 hours
  5. sleep 5d 🡪sleep for 5 days

## Export

To create global variable

Ex. export var1=”devops” , to display echo $var1

## Hostname

Displays host name

## ctrl + c , ctrl +z , bg ↵

ctrl + c 🡪 cancel the terminal, ctrl +z 🡪pause the terminal, bg ↵ 🡪 to resume

## Server commands

## SSH[secure shell]

* 1. port 22
  2. ssh username@ip\_address/servername/DNS
  3. ssh ubuntu@192.13.45.68
  4. password: \*\*\*\*
  5. ssh ubuntu@shstechacademy.com
  6. pass: \*\*\*\*\*
  7. ssh -i aws.pem [ubuntu@192.13.58.3](mailto:ubuntu@192.13.58.3)

## scp: >server copy

* 1. scp file user@server:/home/test
  2. scp -i aws.pem file ubuntu@192.13.24.5:/home/dir1
  3. scp -i aws.pem -r directory [ubuntu@178.42.45.6:/home/d1](mailto:ubuntu@178.42.45.6:/home/d1)

## ping IP\_address/hostname

used to check status of a server

* 1. ping googl.com
  2. ping 192.13.58.32

## netstat -na

to list/check the port outputs of a server

* 1. netstat -na|grep "8080" > check 8080 port of a server
  2. netstat -na|grep "90"

## top

list of all tasks running, similar to task manager

## uptime

displays since when my server is up and running

## rsync

used to transfer files in stages(if internet lost during transfer, sync resumes when internet is back)

* 1. rsync -avl --progress -e "ssh -i test.pem" file1 ubuntu@server

# Shell Script

A shell script is a sequence of commands written in a text file, designed to be executed by a Unix-based shell, acting as a command-line interpreter.

**initialize:**

* echo $SHELL 🡪check the default shell
* chsh bash 🡪 to change default shell
* chsh cshell 🡪 to change default shell

**Create**: Write the script in a text editor and save it with a .sh extension.

* vim script1.sh
* touch script1.sh

**Permissions**: Make the script executable using chmod 777 script1.sh or chmod +x script1.sh.

**Execute**: Run the script with

* ./script1.sh 🡪 to execute shell script
* bash script1.sh 🡪 to execute shell script
* sh script1.sh 🡪 to execute shell script

**Basic Structure:**

A shell script typically begins with a shebang (#!) followed by the path to the interpreter, such as #!/bin/bash. This line specifies which shell should execute the script. Subsequent lines contain commands, comments (prefixed with #), and control structures.

#!/bin/bash --> shebang [ invoke the bash shell]

**Debugging:**

Selective Debugging with set

The set command can enable or disable debugging options within specific parts of the script. set -x turns on verbose mode, while set +x turns it off.

set -x # Enable debugging  
# Code to debug  
set +x # Disable debugging

## basic script to display some meaning full message

|  |  |
| --- | --- |
| Script: script1.sh | Output: |
| #!/bin/bash  echo "This is to test"  echo "the shell script" | this is to test  the shell script |

## Input arguments

$0 -->

$1 --> first i/p argument

$2 --> second i/p argument

$3 --> third i/p argument

;;;

;;;

;;;

$9 --> 9th i/p argument

${10} --> 10th i/p arg

${11}

;;

;;

${100}

## script to pass input arguments in command line and display message

|  |  |
| --- | --- |
| Script: script2.sh | Input & output: |
| #!/bin/bash  echo "this is $1"  echo "started on $2"  echo "there are $3 students" | ./script2.sh devops apr 80  this is devops  started on apr  there are 80 students |

## pass input arguments in command line, store in local variable and display message

|  |  |
| --- | --- |
| Script: script3.sh | Input & output: |
| #!/bin/bash  var1="$1"  var2="$2"  var3="$3"  echo "this is $var1"  echo "started on $var2"  echo "there are $var3 students" | ./script3.sh devops apr 80  this is devops  started on apr  there are 80 students |
| Script: script3a.sh |  |
| #!/bin/bash  var1="devops training"  var2="16th apr"  var3="80"  echo "this is $var1"  echo "started on $var2"  echo "there are $var3 students" | ./script3a.sh  this is devops training  started on 16th apr  there are 80 students |

## Conditional statements (if, if-else, if-elif-else)

Basic syntax as follows

* if statement:

if [ expression/condition]

then

statements/commands to execute if the expression is true

fi

* if – else statement:

if [ expression/condition]

then

statements/commands to execute if the expression is true

else

statements/commands to execute if the expression is false

fi

* if-elif-else statement:

if [ expression/condition]

then

statements/commands to execute if the expression is true

elif [ expression/condition]

then

statements/commands to execute when elif expression is true

else

statements/commands to execute if the expression is false

fi

## Equality operators in shell script (=, ==, !=, -eq , -ne, -lt, -le, -gt, -ge)

String Equality Operators

* = 🡪 Equal to
* == 🡪 Equal to (same as =)
* != 🡪 Not equal to

Integer Equality operators

* -eq 🡪 equals
* -lt 🡪 less than
* -le 🡪 less than or equal
* -gt 🡪 greater than
* -ge 🡪 greater than or equal
* -ne 🡪 not equals

## script to read a number, check if it is equal to 5 and display

|  |  |
| --- | --- |
| Script: Script4.sh | Input & output: |
| #!/bin/bash  #set -x  #sleep 30  read -p "enter a number: " num  if [ $num -eq 5 ]  then  echo "$num is five"  else  echo "$num is not a five"  fi | ./script4.sh  enter a number: 10  10 is not a five  ./script4.sh  enter a number: 5  5 is five |

## script to find biggest of two numbers:

|  |  |
| --- | --- |
| Script: Script5.sh | Input & output: |
| #!/bin/bash  if [ $1 -gt $2 ]  then  echo "$1 is BIG"  else  echo "$2 is BIG"  fi | ./script5.sh 65 600  600 is BIG  ./script5.sh 100 6  100 is BIG |

## special characters of global variables:

* $# 🡪 Total no of arguments pass to a shell script
* $? 🡪 use to check the status of last executed command
  + '0' --> success
  + 'non zero' --> failure
* $$ 🡪 PID of current running process
* $! 🡪 PID of last command went into background
* $@ 🡪 All the arguments pass to a shell script stored in an array format
* $\* 🡪 All the arguments pass to a shell script

## script to restrict two i/p arguments in finding biggest of two

|  |  |
| --- | --- |
| Script :Script5a.sh | Input & output: |
| #!/bin/bash  if [ $# -ne 2 ]  then  echo "we accept only two input arguments"  exit  fi  if [ $1 -gt $2 ]  then  echo "$1 is greater than $2"  else  echo "$2 is greater than $1"  fi | ./script5a.sh  we accept only two input arguments  ./script5a.sh 67 56  67 is greater than 56  ./script5a.sh 67 56 89  we accept only two input arguments  ./script5a.sh 56 89  89 is greater than 56 |

## Looping statements (for loop, while loop)

* For loop basic syntax as follows

for <var> in <value1 value2 ... valuen>

do

commands/statements

done

* while loop Basic syntax as follows

while [ condition ]

do

commands/statements

done

## nohup[no hang up]

The nohup command allows a shell script to continue running even after the user who started the script logs out or the terminal is closed.

Ex. nohup ./script5.sh 60 70 &

Output: [1] 2315

nohup: ignoring input and appending output to 'nohup.out'

[1]+ Done nohup ./script5.sh 60 70

Here, 2315 is PID number and output is stored at nohup.out

Hence, cat nohup.out

70 is BIG

## Script to find biggest of three numbers

|  |  |
| --- | --- |
| Script : bigof3.sh | Input & output: |
| #!/bin/bash  #sleep 30  if [ $# -ne 3 ]  then  echo "we accept only three input arguments"  exit  fi  if [ $1 -gt $2 ] && [ $1 -gt $3 ]  then  echo "$1 is greater than $2 and $3"  elif [ $2 -gt $1 ] && [ $2 -gt $3 ]  then  echo "$2 is greater than $1 and $3"  else  echo "$3 is greater than $1 and $2"  fi | ./bigof3.sh 235 45 876 2  we accept only three input arguments  ./bigof3.sh 45 876 2  876 is greater than 45 and 2  ./bigof3.sh 235 45 8  235 is greater than 45 and 8  ./script5a.sh 56 89  89 is greater than 56  ./bigof3.sh 2 4 8  8 is greater than 2 and 4 |

## Script to add, sub, multiplication and divide two numbers

|  |  |
| --- | --- |
| Script: arithmetic.sh | Input & output: |
| #!/bin/bash  read -p "Enter first number: " num1  read -p "Enter second number: " num2  echo "Addition: $num1 + $num2 = $((num1 + num2))"  echo "Multiplication: $num1 \* $num2 = $((num1 \* num2))"  if [ $num1 -lt $num2 ]  then  echo "Subtraction: $num2 - $num1 = $((num2 - num1))"  echo "Division (integer): $num2 / $num1 = $((num2 / num1))" # Truncates decimals  else  echo "Subtraction: $num1 - $num2 = $((num1 - num2))"  echo "Division (integer): $num1 / $num2 = $((num1 / num2))" # Truncates decimals  fi | ./arithmetic.sh  Enter first number: 5  Enter second number: 10  Addition: 5 + 10 = 15  Multiplication: 5 \* 10 = 50  Subtraction: 10 - 5 = 5  Division (integer): 10 / 5 = 2  ./arithmetic.sh  Enter first number: 10  Enter second number: 5  Addition: 10 + 5 = 15  Multiplication: 10 \* 5 = 50  Subtraction: 10 - 5 = 5  Division (integer): 10 / 5 = 2 |

## Script to find factorial of a number

|  |  |
| --- | --- |
| Script: factorial.sh | Input & output: |
| #!/bin/bash  echo "Enter a number"  read num  temp="$num"  fact=1  while [ $num -gt 1 ]  do  fact=$((fact \* num)) #fact = fact \* num  num=$((num - 1)) #num = num - 1  done  echo "factorial of $temp is $fact" | ./factorial.sh  Enter a number  6  factorial of 6 is 720  ./factorial.sh  Enter a number  9  factorial of 9 is 362880 |

## Script to add given set of numbers

|  |  |
| --- | --- |
| Script: addnumbers.sh | Input & output: |
| #!/bin/bash  read -p "Enter the no's to add: " num  add=0  for i in $num  do  add=$(($i + $add))  done  echo "sum of given num:$num is $add" | ./addnumbers.sh  Enter the no's to add: 4 56 7 890  sum of given num:4 56 7 890 is 957 |

## Script to check given name is a file, directory or not exist

|  |  |
| --- | --- |
| Script: script6.sh | Input & output: |
| #!/bin/bash  read -p "Enter the name to check:" name  if [ -f "$name" ]; then  echo "The given name is a file"  elif [ -d "$name" ]; then  echo "The given name is directory"  else  echo "The given name does not exist"  exit  fi | ./script6.sh  Enter the name to check:f1  The given name is a file  ./script6.sh  Enter the name to check:d1  The given name is directory  ./script6.sh  Enter the name to check:meow  The given name does not exist |

## script to list files having a <pattern>, display message if pattern not found in any file

|  |  |
| --- | --- |
| Script: script7.sh | Input & output: |
| #!/bin/bash  read -p "Enter the pattern to check: " name  list\_of\_files=`grep -ilR "$name" \*`  if [ $? -ne 0 ]  then  echo "The pattern not found in any of the files"  else  echo "The below list of files having the $name pattern:"  echo "$list\_of\_files"  fi | ./script7.sh  Enter the pattern to check: read  The below list of files having the read pattern:  addnumbers.sh  arithmetic.sh  columnprint.sh  factorial.sh  script4.sh  script6.sh  script7.sh  script8.sh  script9.sh  ./script7.sh  Enter the pattern to check: king  The king pattern not found in any of the files |

## script to count no of words, characters in each line

|  |  |
| --- | --- |
| Script: script8.sh | Input & output: |
| #!/bin/bash  num=0  while read line  do  words=`echo $line | wc -w`  num=`expr $num + 1`  char=`echo $line | wc -m`  echo "line number: $num words: $words characters: $char"  done < $1 | ./script8.sh script8.sh  line number: 1 words: 1 characters: 12  line number: 2 words: 1 characters: 6  line number: 3 words: 3 characters: 16  line number: 4 words: 1 characters: 3  line number: 5 words: 5 characters: 27  line number: 6 words: 4 characters: 20  line number: 7 words: 5 characters: 26  line number: 8 words: 8 characters: 57  line number: 9 words: 3 characters: 10 |

## Script to display name of employees whose age is greater than selection

Employee details Input file(f1) has below content

|  |  |
| --- | --- |
| cat f1 | Logic |
| slno name ID mail age  1 tomato 121 tomato.com 21  2 potato 232 potato.com 32  3 apple 343 apple.com 43  4 orange 454 orange.com 54  5 grape 565 grape.com 65 | 1🡪remove top row and save file to temp  2🡪 select first line from temp and increment counter(j)  3🡪compare column 5 with age (ex. 33) and display name if yes, else interment counter(k)  4🡪 compare counters(j and k) and display suitable message |
| Script: script8.sh | Input & output: |
| $ cat columnprint.sh  #!/bin/bash  #set -x  read -p "Enter the file name to pass emp details: " file  read -p "Enter the age to check: " check  sed '1d' $file > temp  k=0  j=0  while read line  do  age=`echo "$line"|awk -F " " '{ print $5}'`  j=$(($j + 1))  if [ $age -gt $check ]  then  echo "$line"|awk -F " " '{ print $2}'  else  k=$(($k + 1))  fi  done < temp  if [ $k -eq $j ]  then echo " none of the employees present in org is greater than age $check"  else  echo "are the employees above age $check"  fi | ./columnprint.sh  Enter the file name to pass emp details: f1  Enter the age to check: 33  apple  orange  grape  are the employees above age 33  ./columnprint.sh  Enter the file name to pass emp details: f1  Enter the age to check: 66  none of the employees present in org is greater than age 66  ./columnprint.sh  Enter the file name to pass emp details: f1  Enter the age to check: 11  tomato  potato  apple  orange  grape  are the employees above age 11 |

## Script to check disk m/m, if crosses threshold, send the mail notification!

|  |  |
| --- | --- |
| Script: script9.sh | Input & output: |
| #!/bin/bash  read -p "enter the threshold limit of disk space:" disk  space=`df -h .|tail -1|awk -F " " '{print $5}'|sed 's/%//g'`  if [ $space -ge $disk ]  then  echo "The disk storage exceeding threshold, Please take the action|mail -s "Disk m/m full" -c "devops.com" team\_devops.com"  else  remaining=$((100 - $space))  echo "need not o worry, remaining disk space is $remaining%"  fi | ./script9.sh  enter the threshold limit of disk space:50  need not o worry, remaining disk space is 82%  ./script9.sh  enter the threshold limit of disk space:15  The disk storage exceeding threshold, Please take the action|mail -s Disk m/m full -c devops.com team\_devops.com |

## cron jobs: crontab

Cron jobs are used to enable automation of tasks at scheduled times. They are used to automate repetitive tasks, such as backups, system maintenance, and report generation, without manual intervention.

|  |  |
| --- | --- |
| \* 🡪 min (00-59)  \* 🡪 hour (00-23)  \* 🡪 Date (1-31/30/28/29)  \* 🡪 Month (1-12)  \* 🡪 Day of the week (0-6)  \* \* \* \* \* ./shell.sh  crontab -l 🡪 list the cron jobs  crontab -e 🡪 edit the cronjobs | Run crontab -e to open crontab file  set the cronjobs and close  Examples:  everyday1pm script should trigger  00 13 \* \* \* script.sh  5th sep 9AM tuesday  00 09 05 09 02 script.sh |

## Script to check services, trigger notification if any service is stopped state

|  |  |
| --- | --- |
| Script: script10.sh | Input & output: |
| #!/bin/bash  services="apache2 Nginx Mysql"  for i in $services  do  sudo service $i status  if [ $? -ne 0 ]  then  echo "service $i is not running, Please take the action"|mail -s "Service stopped!!" -c "test.com" group.com  fi  done | Sudo service not work in git bash |

## cleanup script: script to retain recent 20 builds

|  |  |
| --- | --- |
| Script: script11.sh | Input & output: |
| #!/bin/bash  total=`ls|wc -l`  delete\_num=`expr $total - 20`  if [ $delete\_num -gt 0 ]  then  ls -rt|head -$delete\_num|xargs rm -rf  fi | Logic:  Count no of files (file list + line count)  Files to be deleted = totalfilecount-20 recent  If files to be deleted not equal to zero  File list in reverse (latest first), exclude top 20, remove remaining files be cautious: this will remove your older files (before 20 build) |

# Git:

Git is a distributed version tool, used to manage and track source code, file and directories, to begin with, Repository(repo) is called as a directory in project level, which contains source code related content stored in many folders.

There are two ways to start with git account, initially you can create a directory in local and push to github, otherwise clone existing project directory from remote to local.

## git init

* 1. Transform current directory into a Git repository(ex. git init inside devops directory)

ex. git init

Initialized empty Git repository in C:/Users/asus/Desktop/Devops/.git/

* 1. git init <directory>: Transform a directory in the current path into a Git repository

ex. git init testing (initialize testing repository inside devops)

Initialized empty Git repository in C:/Users/asus/Desktop/Devops/testing/.git/

* 1. git init --bare: Create a new bare repository (used as a remote repository to store content only, that we can’t do active development, most of the commands not work)
  2. To revert git repo into normal directory

run ls -al and look for hidden .git directory, remove using rm -rf .git, repository will be un-initialized.

## git config

once the repository is created in local and Git account created at <https://github.com> use your credential to configure/sync github with local using following command

git config –-global user.name “raghavendra1611” 🡪 only for the first time using github

git config –-global user.email “raag444@gmail.com” 🡪 only for the first time using github

git remote add origin https://github.com/Raghavendra1611/Devops.git 🡪to sync “devops” repository created in github with local workspace

git push --set-upstream origin <branch\_name> 🡪 to sync branch with remote git repository(this should be done on every new branch for first push)

## git clone <url>

used to bring remote repo (copy repo url from github.com) for the first time to local workspace. Here, we download code/files in existing repo of github.com to local workspace.

ex. git clone https://github.com/Raghavendra1611/testing

## git status

can be used anytime to check whether files are in workspace or staging area or in github remote.

## git add <filename>

used to add the files to staging area, As you're working, you change and save a file, or multiple files. Then, before you commit, you must git add. This step allows to choose what you are going to commit.

* git add <filename> 🡪 will add specific file to staging
* git add <path> 🡪 Stage a specific directory or file
* git add -A 🡪Stage all files in the entire repository
* git add . 🡪adds the entire directory recursively, including files whose names begin with a dot
* git add -u 🡪stages modified and deleted files only, NOT new files
* git add -p🡪 to walk through the changes
* git restore --staged <file> 🡪 to unstage added file
* git reset HEAD 🡪to undo git add

## git commit -m "message"

* git commit -m "message" 🡪 this command moves the files from workspace to staging.
* git commit -am "commit message": this option allows you to skip the staging phase. Addition of -a will automatically stage any files that are already being tracked by Git (changes to files that you've committed before).
* git commit --amend 🡪 allows to edit changes commit on your current branch which haven't been pushed to the remote yet

## git push

used to push the committed changes from local workspace to remote repo.

git push --set-upstream origin <branch\_name> 🡪 to push for the first time from a branch

## git revert

acts as "undo commit" but keeps the commit history, to perform this, do git log first, pick commit ID.

Ex. git revert <commit\_ID>

## git reset

will move the HEAD pointer and the branch pointer to another point in time – maybe making it seem like the commits in between never happened! (--soft, --mixed, --hard)

## git log

use to check the history of repo.

## git pull

used to bring the changes from remote repo and merges to local workspace automatically (update the latest code to local from remote)

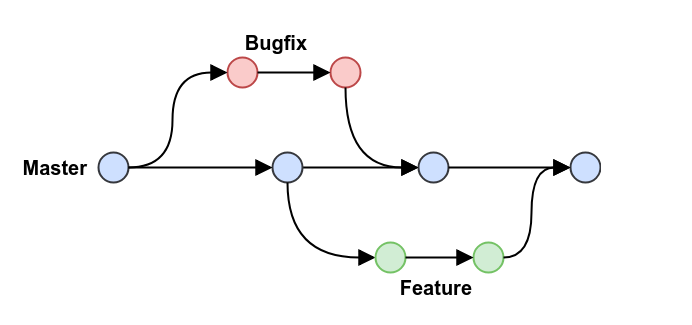
## git tag <tag\_name>

tag is a name given to a set of version of files and directories, indicates milestone of a project, to remember set of versions of code we use tags

* git tag <milestone-1> 🡪 creates tag with milestone-1 name
* git push origin –tags 🡪 push tags to remote
* git tag 🡪 lists all the tags
* git tag -d <tagname> 🡪 to delete a tag
* git push --delete origin <tagname> 🡪 delete a tag in remote

## git branch

GIT branch is copy of the main codebase which will be used for new feature development or bug fix, which will later be merged with the main codebase once after verified.



* git branch -M <new branch> 🡪 create new branch
* Git checkout -b <new branch> 🡪 create new branch
* Git push origin <new branch> 🡪to push new branch to remote
* Git checkout <branch to be switched> 🡪 switch branches
* git branch -d <branch> 🡪 delete a branch in local
* git push origin -d <branch> 🡪 delete a branch in remote
* git branch –-all 🡪 to list all branches

## git merge <branch>

git merge will merge content from one branch to another branch,

ex. merge code of a developed feature from xyz branch to master branch:

* merge in local: commit in xyz branch🡪 checkout to master branch🡪 git merge <xyz>
* merge in remote: commit in xyz branch🡪 push to remote🡪 merge in github.com

## git checkout

used to switch the main version of files, or switch to branches or switch to tags.

Git checkout -b <new branch> 🡪 create new branch

Git push origin <new branch> 🡪to push new branch to remote

git remote add origin https://github.com/hareeshab/repo\_name

git push -u origin main

## git switch

used to switch the main version of files, or switch to branches or switch to tags.

Git checkout -b <new branch> to create new branch

git branch -M main

git push -u origin main

git rebase

git cherry-pick

git fetch

forking: to copy and own a repository of other in git hub

## git stash

git stash list🡪 Show the *files* in the most recent stash:

git stash show 🡪 Show the *changes* of the most recent stash:

git stash show -p 🡪 Show the *changes* of the named stash:

git stash show -p 1 🡪 If you want to view changes of only the last stash: