**SHELL SCRIPTING**

SHELL SCRIPTING :

BASH

#!/bin/bash – shebang

TASKS :

1. Install and configure httpd on ubuntu or linux
2. Install jenkins on tomcat server on RHEL / ubuntu
3. Find all servers FQDN and uptime and export it to a csv file
4. Install and configure ansible setup on AWS cloud
5. User defined shell scripting as system commands

\*\*User defined shell scripting of system commands\*\*

|  |
| --- |
| $ free -m | awk 'NR==2{print $4,"MB(MegaBytes)"}' ==> To get free RAM size in MB  $ export PATH=${PATH}:usr/ec2-user/folder/bin 🡺 To add a new path to the PATH directory  This will work only at the present terminal and will not work when we reconnect to the server.  In order to add it permanently,  $ vi .bash\_profile (In this file add the path)  $ source .bash\_profile (to run it and take effect) |

**TASKS :**

**Find version, status and port used by httpd**

|  |
| --- |
|  |

**SHELL COMMANDS**

|  |
| --- |
| pwd, whoami, id, uname, ls, ls -lrt, touch, clear, cd, date, shell, hostname, ifconfig, sort, seq, wc, bc, head, tails, cat, less more, history, cp, mv, mkdir, rmdir, rm -rf, tar, tar -xvf, zip, unzip, wget, curl, file, stat, ln, grep, awk, sed, cut, last paste, du, df, df -h, free -m or free -mt, netstat, find, whatis (description of a command) , man (to see usage of a command) echo "$(date)", echo "$USER", "$(pwd)", "$hostname" |

$ which $SHELL - to find which shell is assigned to the server  
$ cat /etc/shells - to see all the shells in the system  
BASH - is the advanced shell - extra features

**TASKS**

1. Install and configure httpd on ubuntu or linux  
   Write manual commands, add logic to the commands and run \
2. Install jenkins on tomcat server on RHEL / ubuntu \
3. Find all servers FQDN and uptime and export it to a csv file \
4. Install and configure ansible setup on AWS cloud

**User defined shell scripting of system commands**  
Write shell script, add the path of the script to the systems pre-defined path  
$ free -m | awk 'NR==2{print $4,"MB(MegaBytes)"}' ==> To get free RAM size in MB  
$ export PATH=${PATH}:usr/ec2-user/folder/bin ⎝ To add a new path to the PATH directory

This will work only at the present terminal and will not work when we reconnect to the server

In order to add it permanently,  
**vi .bash\_profile**(In this file add the path)  
**source .bash\_profile**(to run it and take effect)  
echo "$(ls)" ==> will list the files or dir, echo '$(ls)' - will display $(ls), echo $(ls) - will display files or dir in a singleline

echo -e "apple \nbanana \n home" - To display in different lines

echo -n "This is my base this is what i want" - the next line will come immediately  
$ env - to get the system defined variables like $USER, $PATH, $SHELL etc

**FOR LOOPS**  
for a in $(seq 1 10)  
do  
echo "Print my message 10 times"  
done

**Loops - FOR, WHILE, UNTIL, SELECT**  
Sequence of numbers - SEQ, BRACE - eg: seq startnum step endnumber - seq 1 2 10 - Ans - 1,3,5,7,9  
BRACE expansion - echo {start..end}, echo {start..end..step}  
SELECT loop - For select loop you need to enter the number in the list  
use select .. case.. esac.. always use *) invalid option at last*

*Loop control commands - break or Continue  
break - to stop one paticular loop based on a condition, exit - to stop the entire shell script  
continue - is used to skip current iteration of the loop and continue to next iteration*

***FOR LOOP - INFINITY*** *for (( ; ; )) - infinite number of times*

***WHILE LOOP - INFINITY*** *while :  
do... done*

***free\_ram=$(free -mt | grep Total: | awk '{print $4}')******Automate Server Inventory***

*Need to find servername, IP address, OS type, uptime, JAVA\_version, web\_server, web\_server\_version, tomcat, weblogic - Inventories  
Find the above inventory and generate it into a CSV file - S\_NO,server\_name,IP\_address,UPtime,*

*PS1 - Primary prompt screen -****[ec2-user@ip\_address ~]*** *PS1 - many formats are there like \t, \T, @ etct*

*To change PS1 permanently export PS1="\u\t.." on .bash\_profile or .bashrc file  
printf - same as echo with advanced features like can be on the center with colors and have any designs \ syntax - printf - eg: printf*

*PATH\_httpd="/var/www/conf/httpd.conf" - this is saved  
In order to get only the directory****dirname $PATH\_httpd****, to get only the filename****basename $PATH\_httpd***

***Eg: $path\_info="/tmp/user/apache.xy.tar.tz"*** *echo "{$path\_info%tar.tz}" - output: /tmp/user/apache.xy, echo "{$path\_info#/tmp/}" - output: user/apache.xy.tar.tz  
echo "{$path\_info%%.*}" - output: /tmp/user/apache.xy (it will remove all the dots)  
echo "${path\_tmp#*.}" - xy.tar.tz, echo "${path\_tmp##*.}" - tz (will remove everything before the last "."

**AWK command**  
syntax : awk options 'pattern/condition {action}' filename command | awk options 'pattern/condition {action}' \ awk 'BEGIN {start\_action} pattern/condition {action} END {stop\_action}' filename  
eg: awk -f script.awk filename (logic can be written in .awk file)  
BEGIN - it is performed before reading the file, END - It is performed after processing the file

**EG:** **awk 'BEGIN { print "=========Starting AWK action========" } /root/ { print NR, $2} END { print "=================END of AWK \ Action==========" }' /etc/passwd** ==> the same script from BEGIN till end can be created as script.awk file and passed as  
awk -f script.awk /etc/passwd  
BEGIN - can be used to initialize a variable, eg: awk ' BEGIN { a=5 ; print a }' - multiple statements separated by ";"  
#!/bin/awk -f and the remaining command from BEGIN to END in .awk file and run \

$NAME - To refer the variable  
**READ, READONLY, UNSET  
READ** - To read a variable input entered by the user  
echo $$, $0, $n, $# - special character arguments  
**#!/bin/sh  
for TOKEN in $@ \ do  
echo $TOKEN  
done\*\*  
./test.sh Tom 10 years old**

array\_name[index]=value, NAME[0]="Tom"  
TEST=expr 19 + 3 - expr (+,-,\*,/)  
**A=10  
B=40  
if [ $A == $B ]  
then  
echo "They are equal"  
elif [ $A != $B ]  
then  
echo "They are not equal"  
fi**

**REGULAR EXPRESSIONS**

\*\* SED(Stream Editor), ED, AWK, GREP \*\*  
sed - used for insert, delete or replace any string in a file  
Eg: $ sed 's/hello/good/g' file1.txt - This will replace all the hello with good. 'g' is global.  
If g not specified it will replace only the first occurrence of a sentence \

**LINUX COMMANDS** cat, mv, cp, mkdir, rm, rm -rf, more (to go through the file), tail (to go to the end of file), grep (to search)  
'|' to use the output of one file as input of another, find, head, whereis, which, df -k (disk space)

**MOUNTING THE FILE SYSTEM :  
mount -t file\_system\_type device\_to\_mount directory\_to\_mount\_to**  
User administrative files :  
/etc/passwd, /etc/shadow, /etc/group, /etc/gshadow  
groupadd, groupmod, groupdel (to create, modify and delete groups in /etc/groups)  
useradd, usermod, userdel (to create, modify and delete user)  
/etc/syslog, /etc/syslogd.conf (all the system logging information can be found here)  
Vi editor - to come out of the screen SHIFT+ZZ, :wq! - To save and quit, !q (to exit without saving) \

**STRINGS OPERATIONS :**name="shell", echo "${#name}" - output 5- length of the string

Concatenate 2 strings - eg: path1="/etc/xyz" path2="file1.txt" con\_path=$path1/$path2

(${string1}${string2} will concatenate 2 strings)

Convert strings to lower or upper case - name="shell" echo "${ name^^ }" - to upper, echo "${ name,,}" - to lower

**dirname, basename** on path/strings - eg: dirname /etc/conf/httpd.conf - output : /etc/conf, basename path - output: httpd.conf  
basename /etc/conf/httpd.conf .conf --> in the result .conf will be removed

String slicing or extracting a portion of a string - Eg: myname="shell" echo "${myname:1}" ==> hell (will start from 1st position)  
syntax : ${variable:(starting\_position):(how many characters we need)} ==> "${myname:2:2}" ==> el (string counts like array)

**INPUT AND OUTPUT COMMANDS**  
Input - read - $ read -p "Enter your name : " my\_name - (-p - prompt a message)  
If you dont specify a variable for read - REPLY is the default variable for READ command

Output command is ECHO

Input through CLI - $1, $2, $3 - ./file1.txt tom jane apple - $1-tom, $2-jane, $3-apple - more than 10 ${10},${21}  
"$#" - number of command line arguments given  
"$@" or "$\*" - All the command line arguments passed

**ARITHMETIC OPERATIONS** - let, declare, expr - old ones  
NEW ONES - Paranthesis , Bash calculator (BC) - Eg: ((sum=3+5)), ((div=50/10)) - but float result not possible with (())  
((x++)), ((y--)), ((x=y+10)), NOTE: inside (()) no need to add $ symbol for variable

for float ((sum=4.5+5.6)) - will not work, Bash calculator must be installed "sudo yum install bc"  
bc<<<"4.5+5.6", bc<<<"scale=2; div=$y/$x" - scale is 2 means 2 digits after the decimal

**NOTE: need to use $ for bc**case... esac [0-9]) .... ;;, [a-z].. ;;, [A-Z]).. ;;, \*) exit ;;  
comparison operators : integers, letters - -eq, -lt, -gt, -le, -ge ==> will return 0 or 1  
for string comparison : my\_name=" ", [-z my\_name](https://github.com/gayathriprashant/shellscripts/wiki/-z-my_name) - will test whether the variable is empty  
[string == string](https://github.com/gayathriprashant/shellscripts/wiki/string-%3D%3D-string), [string != string](https://github.com/gayathriprashant/shellscripts/wiki/string-%21%3D-string) - check the result as echo $? - 0 for true, 1 for false  
File test operator : [-d directory name](https://github.com/gayathriprashant/shellscripts/wiki/-d-directory-name), [-f filename](https://github.com/gayathriprashant/shellscripts/wiki/-f-filename)  
command chaining operators : semicolon (;) operator, logical AND (&&), logical OR (||) operator

**eg:** ls && pwd && date, first command should be successful to execute the next command  
**eg:** ls || hsuhgius || date - command 2 will execute only if cmd 1 fails \ systemctl status httpd || sudo systemctl start httpd (if it is not running status 1 then it will start )  
cmd 1 && cmd2 || cmd3 ==> [-e "jkghsjk"](https://github.com/gayathriprashant/shellscripts/wiki/-e-%22jkghsjk%22) && echo "yes file" || echo " no file "  
cmd1 || cmd2 && cmd3 ==> [-e "file1.txt"](https://github.com/gayathriprashant/shellscripts/wiki/-e-%22file1.txt%22) || echo "There is not file" && touch file1.txt  
if.. then.. else.. fi, if.. then.. elif.. then.. else..fi

**Creating backup logs**   
tar to zip or unzip a file, options - "-c" - to create, \ "-v" - verbose mode to print all the file that are archived,  
"-p" - to preserve file and dir  
"-z" - to compress the files  
"-f" - to get the file name  
timestamp=date '+%b-%d-%Y-%H-%M-%S' - To add timestamp to the backup files created  
tar -cvpzf /etc/conf/backup/logs\_${timestamp}.tar.gz Source - create cron job to run automatically \

**scheduling with "at" or "crontab"** : at - to schedule for only one day, crontab- for everyday  
have a shell script ready to run the backup  
$ at eg: at 11:50, >at bash scriptname.sh >at press ctrl+d (it will save the job and run thefile)  
or $ echo "bash scriptname.sh" | at 3:00 AM - check the status using the cmd "atq", atrm - to remove a job

**CRONTAB - Syntax** - minute hour day month weekday command/script  
crontab ==> -e (to create a job), -l (to list the job), -r (to remove the job)  
**eg:** **crontab -e (itwill open a window** ) ==> 58 2 10 2 5 /root/filename.sh (give the location of the file)  
if you need every month (**58 2 10 \* 5 ...)** month fields "*",* ***\* \* \* \* \**** *- every min hour day etc  
9AM and 9PM - 0 9 \* \* \* scriptname , 0 21 \* \* \* scriptname, 0 9,21 \* \* \* - both 9am and 9pm*

*Every 2 hours -* ***0 /2 \* \* \**** *- every 2 hours, 0 0 1 1 \* - yearly once or @yearly,@monthly, @daily, @reboot, @hourly  
0 2 10 1 0,6 - for weekends sat or sun  
free -mt (memory size with total)  
mail -s "SUBJECT $(date)" - To send an email alert with subject, /bin/mail - to work mail command, 0/30 \* \* \* \**[*-x httpd.conf*](https://github.com/gayathriprashant/shellscripts/wiki/-x-httpd.conf)*==> To check if the file has execution permission  
for each in $(ls).. do.. if*[*-x $each*](https://github.com/gayathriprashant/shellscripts/wiki/-x-%24each)*.. done (this will check the execution permission of all the files in the ls list  
c lang type for loop : eg: for((a=1;a<=10;a++)), for(( ; ; )) ==> Infinity for loop*

***Install packages using loops :****if*[*$(id -u) -ne 0*](https://github.com/gayathriprashant/shellscripts/wiki/%24%28id--u%29--ne-0)*==> Means it is not a root user  
if which vim &> /dev/null ==> if vim is installed it will be true, if not install vim  
for each\_pkg in vim httpd nginx .. do .. if conditions .. done, for each\_pkg in $@ ==> can pass multiple as command line arguments  
$# --> To check the number of arguments passed through CLI eg: ./package.sh vim jenkins https  
Difference between $@ and $ --> for each in "$@" , "$*" ==> ./scripts.sh 1 2 "3 4" 5  
"$\*" ==> 1 2 3 4 5 (this will take the cli arguments as a single value), "$@" - 1,2, 3 4, 5 (this will take each value separately)  
break - It terminates the only the loop, exit - to come out of the script completely, continue- is to not execute one iteration and skip to the next one \

**Connecting to remote servers :**  
**using password ==> /etc/ssh/sshd\_conf ==> password authentication yes** must be there in order to connect to the remote server  
systemctl restart sshd or service sshd restart ==> This will restart ssh in order to implement them  
ssh username@ipaddress ==> password : ? ==> Then it will connect local server to remote server \ **PASSWORD LESS AUTHENTICATION**  
$ ssh key-gen ==> This will create a .ssh folder with id.rsa and id\_rsa.pub (private and public keys), this must be copied to the remote server  
$ ssh-copy-id username@ipaddress ==> first time will ask for password ==> .ssh folder will be copied ==> ls -a  
.ssh/authorized\_keys (it will be stored here ) \ same id can be used for another user, eg: $ ssh-copy-id diffusername@ipaddress

**EXECUTING COMMANDS ON REMOTE SERVER WITHOUT LOGGING REMOTE SERVER**  
$ ssh -o StrictHostKeyChecking=No username@ipaddress ==> Will not ask yes/no to logging ==> not for automation

**AUTOMATION TO CONNECT TO REMOTE SERVER** $ ssh -o **StrictHostKeyChecking=No** username@ipaddress "command(date)" ==> will ask for password and then displays output  
$ server\_1=$(ssh -o StrictHostKeyChecking=No username@ipaddress "command(date)") ==> This will give the output of server 1 without logging  
$ ssh -t -o ... (-t is for command which need ctrl+c to stop the command)  
$ ssh -o StrictHostKeyChecking=No username@ipaddress "date;uptime;pwd;free -m" >> file1.txt ==> run all commands together and store in a file  
\*\* providing password for ssh using SSHPASS \*\*  
If you need to use 10 remote server commands each time need to give a password. To automate using shell script SSHPASS can be used

$ **sshpass -p** "password" ssh -o StrictHostKeyChecking=No username@ipaddress "date;uptime;pwd"  
store password in a file eg: pass, $ sshpass -f pass ssh -t -o ...  
export SSHPASS="password" save it, sshpass -e ssh -t -o ..., yum install sshpass - to install  
If sshpass could not install, first install epel according to the OS and then install sshpass

**EXECUTE MULTIPLE COMMANDS ON MULTIPLE SERVERS**  
for loop in another for loop, one for for ip address, another for loop for logic for commands  
can be done without sshpass by using ssh-copy-id and copy the authorized\_keys first  
store all the ip\_address in a file like 10 ip address, for each\_ip in $(cat filename) do.. done

**EXECUTE DIFFERENT COMMANDS ON DIFFERENT SERVERS WITH DIFF USERS AND DIFF PASSWORDS**  
eg: 10.9.8.0 username password command ==> save n number of this in a file  
while read server uname pass cmd .. do.. sshpass -p $pass ssh -n -o StrictHostKeyChecking=No $uname@$server "$cmd"... done < filename.txt(where the details are stored) "-n" --> to execute all the records in the file, do not expect STDIN

**WHILE LOOP**  
while true ... do .. done ==> will be done infinite times, while :.. do.. done, while cmd.. do.. done  
eg: while date &> /dev/null.. do.. done or while test eg: while [4 -gt 1](https://github.com/gayathriprashant/shellscripts/wiki/4--gt-1).. do.. done , while read variable.. do.. done  
Read command output : command | while read .. do. .done, Eg: cat filename.txt | while read line.. do.. echo "$line"... done  
FOR will take as each field WHILE LOOP will take as a single line, eg: ls -lrt to print each line while loop can be used

**IFS Internal Field Separator :**By default WHILE LOOP will take space, if you are providing CSV, field separator must be specified  
while "IFS"="," read line.. do.. echo .. done < filename.txt \ If you dont need header of csv file filter using the command ==> cat file1.txt | awk ' NR!=1 {print}' ==> this will print everything except first line **FUNCTIONS - block of code, it can be reused**  
function\_name(){...}, eg: code () { read -p " Enter num1 : " num1 ; read -p " Enter num2 : " num2 }, if then.. code (calling the function) .. fi  
2 ways to define : function func\_name { commands/ statements }, function\_name() { cmds/ stmts }

scope of global variable and local variables ==> x=5 func\_name() { local x=10 } echo "$x"(it will print 5) ==> local can be used only inside the function  
global x=10 ==> can be used outside of a function, x=10 ==> can be used outside of a function

Return a variable value ==> func() {local x=10; echo "$x" } func ; y=$(func) ; echo "$y"  
return will give exit status as output of the function ; return must have numeric value, echo is better way to use \ \*\*passing parameters to a function : \*\*  
addition() { m=$1, n=$2 c=$((m+n)) echo "$c" } x=10, y=5 addition $x $y \ **PRINTF COMMAND**

Same as echo with some advantages - printf does not give new line at the end, eg: printf "Enter a number 1:" num1  
To decorate the output -------test---- ==> To get on a full screen find the number of columns and rows with \

**$ tput cols ; tput lines eg: 80, 25 \**

printf "%113s" " " | tr " " "-" ----------- for the entire screen  
useful for AWK commands ==> syntax : printf "format w/ modifiers \n" "arguments "  
%s - string, %d- integer, %f- float, eg: printf "%s %d %f\n" "$name" "$x" "$y" ==> printf with formats  
formats with modifiers ==> "%09d" ==> will leave 9 spaces and replace with 0s ==> 000000008  
"%0.3f\n" ==> 9.876,

**ARRAYS**  
x=(1 2 3 "shell" "scripts") echo "${x[@]}" or ${x[\*]} ==> to get the entire values in the array, "${x[0]}", "$x[3,4]".. so on  
${x[@]:2} ==> from index 2, ${x[@]:2:4} ==> range of values, ${#x[@]} ==> # is used to find the length  
declare -a x=(1 2 3 4 5) ==> another way of declaring an array; z[0]=20;z[1]=30; echo "$z[@]" => declare based on index position  
read -a array1 ==> then give array inputs in the command line

Add elements or value to an existing array or remove from array ==> eg: list=("apple" "banana" "xyz")  
list=("${list[@]}" "berries") ==> to add arrays

TO REMOVE ==> unset list[2] (will remove that particular index value)  
my\_arr=(1 2 3 4) for each in ${my\_arr[@]} do. done ==> this will read each value from the declared array  
array=($(date)) ==> date is declared as array

**AWK COMMANDS**  
syntax : awk 'pattern/commands' filename Eg: awk -F ',' 'NR==1 { print $1 }' filename.txt  
options : -F (field separator), -v var=value (variable value), -f (to specify the file that contains awk script) \ pattern/commands : '/root/' or '/version/' (can be filtered),NR==1 (num of records), NF=Number of Fields, 'NR==1 {print $1}'  
filename.awk (save the commands that needs to be executed) ==> awk -f filename.awk 'print $1'  
"|" one awk command can be passed as input to another awk command  
AWK 'BEGIN {start\_action} pattern/condition {action} END {End\_action}' ./filename.awk ==> #!/bin/awk -f, if running as $ awk -f filename.awk then no need to specify the shebang line  
ab.txt ==> 2 6, cat ab.txt | awk ' {print "a="$1, "b="$2 }' \

**ARRAYS**  
How to declare an array in shell scripting ?  
eg: date\_array=($(date)) , echo "${date\_array[@]}" - This will take the date command in an array  
if [ $(whoami) != "root" ] - to see if it is in root with sudo privileges or not  
**sudo yum remove git wget vim** - This will remove installed software

**OUTPUT REDIRECTION :** \ '>' - Will help to create a file and move the output to a new file. IF used again, will replace the previous content and override  
'>>' - This will append the new content will not override

**INPUT REDIRECTION:**  
'<' - To provide the input  
\*\*PIPELINE '|' : \*\* - Input of one command is output of next command using '|' - eg: ls | grep filename

**How to separate STDOUT and STDERR ?**STDIN - 0, STDOUT - 1, STDERR - 2 \ Eg: ls 1>success.txt 2>error.txt (It will store the status of the command in appropriate file), jghskjgh 1>succ.txt 2>err.txt

**Eg:** java -version 1>java\_ver.txt 2>java\_ver.txt (to store either success or error in the same file)  
or java -version 1>java\_ver.txt 2>&1 (Store the error in the same file as success "2>&1")  
or java -version &> java\_ver.txt  
'1>' or '2>' or '2>&1' or '&>'  
echo [-e, -n, -E] --> \n, \t, \T, \v, \b, \r, \c,\a, \(eliminates the special purpose

**Access one shell script in another shell script**  
eg: #!/bin/bash ./start\_httpd.sh ./stop\_httpd.sh - This can be executed this way

**To use the variable from other scripts or create a separate variables file and call the variables**  
Eg: file1.sh (#!/bin/bash test="Testing value"), file2.sh (#!/bin/bash source ./file1.sh echo "$test")

**Eg:** variables(filename) (x=10 y=20), test.sh (#!/bin/bash source ./variables echo "$x, $y") \ **EXIT STATUS**  
$ echo $? (will give the status of previously executed command - 0 -for success, 1 - cmd failed during execution but cmd valid)  
2 - Incorrect cmd usage, 127 - cmd is wrong )

**GREP USAGE \ grep** - It is used to search a text in a file. It needs input to search for a text "|"  
syntax - grep [options] "string/pattern" Eg: grep "line" file1.txt  
options ==> -i, -w, -v, -o, -n, -c, -A, -B, -C, -r, -l, -h

|  |
| --- |
| grep -i "Above" file1.txt (To ignore case sensitivity) grep -w "above" file1.txt (exact matching word) grep -v "above" file1.txt (print everything except "above") grep -o "above" file1.txt (to display just the searching word) grep -n "above" file1.txt ( to display the line number ) -no (to display line number with exact word) grep -c "above" file1.txt (count of number of lines matching) or -cw (count and word match) grep -A 1 "above" file1.txt (print one line after "above" match) grep -B 2 "above" file1.txt (print 2 lines before above match) grep -C 1 "above" file1.txt (print one line before and after above match) grep -r "above" file1.txt (to search under current dir or sub directory recursively) Eg: grep -r "bash" \* grep -l "above" \* (print only filenames) grep -h "above" \* (will hide the file names and will display only the line matching the text) grep -f my\_search file1.txt (my\_search is a file with words like line, book, sample etc with words we need to search. This will show all the words to be searched through a file grep -e "line" -e "book" -e "sample" file1.txt (execute multiple search strings through "-e") grep -E "line|book|sample|string" file1.txt (with pipeline multiple word match with "-E") |

**RULES TO CREATE PATTERNS :**

**"|" - pipeline "^" -** To search for starting of the word. Eg: grep -E "^line" file1.txt  
eg: grep -E "^Listen" /etc/httpd/conf/httpd.conf (to search only Listen 80 port number)  
"$" - To search the word at the end of the line. Eg: grep -E "line$" file1.txt  
"^$" - Matches the empty lines  
"" - To remove the purpose of any symbol. Eg: grep -E "^" file1.txt (It will search for "^" symbol in the text if "" used before the special symbol. "." - Eg: grep -E "t..s" file1.txt, grep -E "." file1.txt

"\b" - needs space before and after the string - Eg : grep -E "\bline" file1.txt  
"?" - that character can appear 0 or more times - Eg: grep -E "thi?" file1.txt - This will show either that or this  
"*" - Preceding character will search o or more times. Eg: grep -E "thi*" file1.txt - i can be many times

"+" - Atleast one time that character must be there. Eg: grep -E "xy+" file1.txt  
"[]" - will search [TtfFgG] - will search each character individually  
"[a-d]" - will search characters from a to d  
"[a-dp-r]" - to search two set of sequences  
"^[ab]" - will search only for lines starting with a or b, "[^ab]" - will search lines other than ab

"{N}" - number of times - eg: grep -E "xffff" file1.txt or grep -E "xf{3}" file1.txt or grep -E "xf{3,4}" file1.txt  
**Eg: grep** -E "xf{2,}" file1.txt - f must be min 2 times max any time  
[[[:digit:](https://github.com/gayathriprashant/shellscripts/wiki/%3Adigit%3A)]] - grep -E "[[[:digit:](https://github.com/gayathriprashant/shellscripts/wiki/%3Adigit%3A)]]" file1.txt - will display the lines that has numbers  
[[[:upper:](https://github.com/gayathriprashant/shellscripts/wiki/%3Aupper%3A)]], [[[:lower:](https://github.com/gayathriprashant/shellscripts/wiki/%3Alower%3A)]], [[[:blank:](https://github.com/gayathriprashant/shellscripts/wiki/%3Ablank%3A)]], [[[:alpha:](https://github.com/gayathriprashant/shellscripts/wiki/%3Aalpha%3A)]], [[[:space:](https://github.com/gayathriprashant/shellscripts/wiki/%3Aspace%3A)]], [[[:alnum:](https://github.com/gayathriprashant/shellscripts/wiki/%3Aalnum%3A)]]

**IMPORTANT COMMAND :**To find the ipv4 address from a file - 10.0.9.7, 192.3.45.6, 4567.89.9.9 \ Eg: cat file1.txt | grep "\b[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}\b" - it will eliminate 4567 number  
CUT COMMAND - It is used to get the character from a set of lines in a file

**Eg:** cut -c 1 file1.txt - will get only the first character, cut -c 4,9 file1.txt, cut -c 4-9 file1.txt  
cut -c 4- file1.txt, cut -c -10 file1.txt  
FIELDS - must be separated by tab space, or if it is separated by anything else need to mention that

**EG :** cut -f 1 file1.txt, cut -f 1-3 file1.txt - This will work only if fields are separated by tab  
root:etc:/root:file:xyz ==> cut -d ':' -f 2 file1.txt ==> output - etc - this will consider : as a delimiter  
cut -sf 1 file1.txt (This will display only the fields that has tab space and will eliminate lines)

**Eg :** $ httpd -v ==> server version : Apache/2.4.2 () server built: oct 29 - I want only the server version value

$ httpd -v | grep -E "version" | cut -d '/' -f 2 | cut -d ' ' -f 1 ==> output is 2.4.2  
The same above requirement can be done using awk command  
server version : Apache/2.4.2 () ==> $ httpd -v | awk -F '[ /]' ' /version/ {print $4} ' - -**F is field separator [ /] means take space and also / as field separators**

**SYNTAX OF AWK  
awk[options] '[selection criteria] {action}' input file** eg: awk -F '[/]' '[NR==1] { print $3 }'  
awk options : -F, -f, -v, NR - number of Records (Condition), NF - number of Fields  
Default delimiter for awk command is " " space  
awk default commands : $0 - entire file, { print } - prints entire file \ To reverse the fields in the output - awk '{ print $3,$1 }' file1.txt\

**Sed command** :

**Sed is Stream Editor 🡺 to search or replace a letter or word**

Edit file without opening the file with vi/vim editors

**Sed [options] ‘commands’ [filename]**

**Viewing file content with sed command :**

**Only for print use “-n”**

$ sed ‘ ‘ filename.txt 🡺 can be used as cat command

$ sed -n ‘p’ filename.txt 🡺 to print the output one time need to put -n

$ sed -n ‘4p’ filename.txt 🡺 4th line printed

$ sed -n ‘$p’ filename.txt 🡺 last line printed

$ sed -n ‘3,10p’ filename.txt 🡺 range of lines

$ sed -n ’12,+8p’ filename.txt 🡺 from 12 line to 8more lines to be printed

$ sed -n ‘1~3p’ filename.txt 🡺 1,4,7,10.. with 3 difference

**Deleting lines : It will not delete permanently, it will remove and display**

$ sed ’14,$d’ filename.txt 🡺 will display only from 1 to 13 and remove the remaining

$ sed ’10,20d’ filename.txt 🡺 will remove lines from 10 to 20 and will display the remaining lines

**Changing original file and delete in the file itself** : “**-i**” – interactive mode

$ sed **-i** ‘10,20d’ filename.txt 🡺 This will remove the lines from the original file

$ sed -i.back ’10,20d’ filename.txt 🡺 will take backup of the data before deleting it

**Searching for file content** :

$ sed -n ‘/echo/p’ filename.txt 🡺 will search for echo word and print the lines

$ sed -n **-e** “/bin/p” -**e** “/echo/p” filename.txt 🡺 to search multiple words

**Find and replace with sed command** :

$ sed **‘s/root/route’** filename.txt 🡺 This will replace root with route. But it will replace only first occurrence of each line

$ sed **‘s/root/route/g’** filename.txt 🡺 to replace everything with g globally

$ sed **‘s/root/route/2’** filename.txt 🡺 replace 2 times on a line

The above commands will not replace the original file, include ‘-i’ to replace permanently

$ sed ‘/dev/s/bin/root/’ filename.txt 🡺 this will search for lines with dev and in that line it will replace bin with root 🡺 eg of find and replace

**Insertion and deletion with SED command** : “1i” (before 1st), “1a” (after 1st line)

$ sed ‘1i SNO NAME’ filename.txt 🡺 will insert the given information before the 1st line

$ sed -i ‘1i SNO NAME’ filename.txt 🡺 This will insert permanently

$ sed ‘$a ----------’ filename.txt 🡺 after last line insert

$ sed -i ‘/dev/i boot camp’ filename.txt 🡺 will insert before dev line

$ sed -i ‘/dev/a boot camp’ filename.txt 🡺 after dev line insert the lines

**REGEX – REGULAR EXPRESSION** :

Any expression that uses a pattern is called Regex

**Different ways to write a pattern** :

$ sed -n ‘/p[uo]t/p’ filename.txt 🡺 it will search for “put” and “pot” and prints

**Special characters : \s, \t, . , \*, \+, \? And \**

**“\s” – Matches for space** , **\ -- escape character**

Eg: $ sed -n ‘/\s/p’ file.txt 🡺 prints lines that has spaces

Eg: $ sed -n ‘/\\s/p’ file.txt 🡺 will search for “\s” in the line

“**\t**” – **prints tab space**

“.” – matches any character eg: $ sed -n ‘/p.t/p’ file.txt 🡺 p\_t will be printed

“\*” 🡪 can be repeated or not

Eg: $ sed -n ‘/ches\*/p’ file.txt 🡺 will print ches and also chess or chessss

**“\+” 🡺 ‘/this\+/p’ 🡺** thi**s** must be there atleast once or more than once

**“\?”** 🡺 ‘/this\?/p’ 🡺 thi.. this.. thissss.. will be printed

**“^” – change to upper case,**

**Eg:** $ sed -n ‘/^this/p’ file.txt 🡺 prints lines with this only at the starting of the line

**“$” --** $ sed -n ‘/cat$/p’ file.txt 🡺 Will search for word cat only at the end of the file

**“^$” 🡺 find empty lines 🡺 eg:** $ sed -n ‘^$/p’ file.txt 🡺 will search for empty lines

$ sed -i ‘^S/d’ file.txt 🡺 permanently deleting empty files

**[], {}, () – special characters**

**“[]”** 🡺 matches any single characters in array. eg : ‘/p[auo]t/p’ 🡺 put, pat, pot, ‘p[a-e]t’

**“{}”** 🡺 matches for required number of repetitions, eg: sed -n ‘/this\{3\}/p’ – thisss,thissss

eg: sed -n ‘/this\{3,4\}/p’, eg: sed -n ‘/this\{3,\}\b/p’ 🡺 after s need a space

**“()” 🡺 zero or more whole sequence,**

Eg: $ sed -n ‘/\(cat\)\{2\}/p’ file.txt 🡺 catcatcat

$ sed -n ‘/\b\(cat\)\{2\}/p’ file.txt 🡺 before cat there should be space

**\{\} , \(\)**

**USAGE OF SED, CUT, AWK and ARRAYS :**

Get git versions from git-scm :

**AUTOMATE MULTIPLE SERVERS** :

OS TYPE

OS VERSION

ARC\_TYPE

CPU\_TYPE

User=devops, password = automation

Sshpass -f remotepass ssh -n -o StrictHostKeyChecking=No -o PubkeyAuthentication=No devops@$server “command”

Remotepass 🡺 file to save the password

Remoteuser 🡺 file to save the username

List\_of\_servers 🡺 list of ip address to connect

**FILESYSTEM USAGE and SEND MAIL ALERTS** :

$ df -pH | grep -vE “Filesystem|none|tmpfs” 🡺 To eliminate the rows to not monitor