UNIX

➢ Chapter 1 : Introduction to Unix

• **SH** > is light weight language which process a sequence of unix commands.

• **awk** > is used to manipulating data and generating reprots. Do not require compiling and allows user to use variables, numeric func,string fun & logical operator.

• **Sed** > receives text input as a “stream” and edits the stream according to your instructions.

• **grep** > is used to search text or searches the given file for lines containing a match to the given strings or words.

1. File Spacing

2. Numbering lines

3. Deleting lines

4. View/Print the files

5. Pattern Matching

6. Replacement/substitution

vim > is text editor like notepad in windows.

➢ Chapter 2 : Basic Commands

Commands	Descriptions
echo	> print the messages
read	> accept input
ls	> list the files & Directories
ls -ltr	
ls -tr	
ls -F	
pwd	> present working directory
cd ..	> Change Directory
tar cvf	> to archive
tar xvf	> extracting from archive
tar tvf	> view from existing tar
gunzip	> compress the file & Directories
gzip	
unzip	
gunzip -c	> view contents in the file without unzip
find . -name 'file'	> find or search file
diff	> difference
cmp	> compare
sort	> asc order
sort -r	> Desc order
cat	> create file or display content
less	> display
more	> display
tail -N	> display bottom n records
head -N	> display top n records
ps -ef | more	> process ID
ps -efH | more	
top	> displays top process in the system
df	> Displays the file system disk space.
df -h	> displays output in human readable form
kill -9 PID	
rm	> delete file or directory
cp	> copy file or directory
touch	> create file
mkdir	> create file or folder
mv	> move file or directory

chmod	> provide read,write permission
ifconfig	> to check IP address
uname	> dispalys imp info abt system such as kernel name, host name, realases
lsb_release -a	> displays version
echo $?	> to check last processed command 0 is success != 0 means failure
echo $!	> last process job id
echo $$	> processed number of current shell
cal	> to check calender
date +%m	> date formatting
mail	> working with emails
uuencode	> working with emails
wc -l	> display the count of rows
head -1 df.csv | sed 's/[^,]//g' | wc -c	> display the count of columns
cut -d ',' 1 df.csv	> display the columns
bc	> perform arithmatic operations
expr 10 + 10	> to mathematical calculation
split -n df.csv	> split large file into multiple files
scp	> copy or transfer file or directory from one server to another

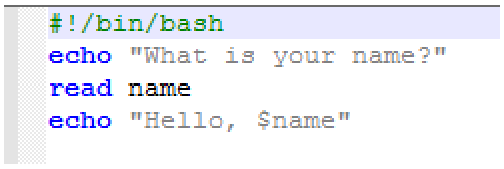
Subject wise commands			
communications	**Miscellaneous**	**Shells**	**System Status**
ftp	banner	**Bourne family Shells:**	at
login	bc	bash	crontab
mailx	cal	ksh	date
slogin	calendar	pdksh	df
ssh	clear	sh	du
Comparisons	info	zsh	env
cmp	man	**C shell family shell:**	finger
comm	nice	csh	kill
diff	nohup	tcsh	ps
diff3	passwd	**Shell programming**	stty
dircmp	script	basename	who
sdiff	spell	dirname	**Text Processing**
File Management	su	echo	awk
cd	**Printing**	expr	cat
cdgrp	lpr	id	cut

chmod	lpq	line	ex
chown	lprm	printf	fmt
cksum	pr	sleep	iconv
cp	**Printing**	test	join
csplit	cancel	**storage**	paste
file	lpqlpstat	bunzip	sed
head	pr	bzip2	sort
less	**programming**	cpio	tr
ln	cc	gunzip	uniq
locate	ctags	gzcat	vi
ls	ld	gzip	xargs
md5sum	lex	tar	
mkdir	make	zcat	
more	od		
mv	splint		
pwd	strace		
rm	strip		
rmdir	truss		
scp	yacc		
split	**Searching**		
tail	egrep		
wc	fgrep		
	find		
	grep		
	strings		

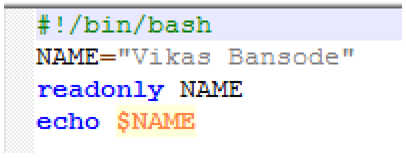
```
find . -name "*string*" -o -name "*string"
```

```
ssh username@servername
```

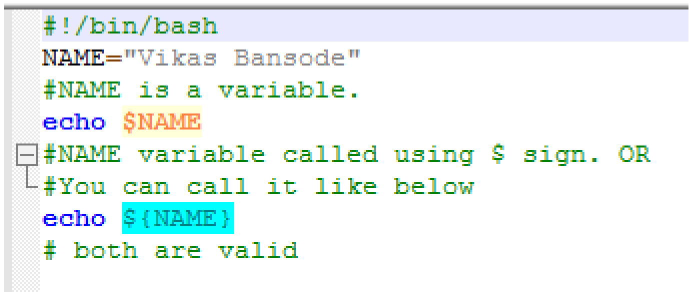
➢ Chapter 3 : Shell Scripting



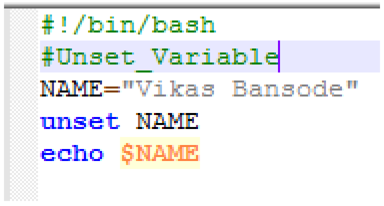
```
#!/bin/bash
echo "What is your name?"
read name
echo "Hello, $name"
```



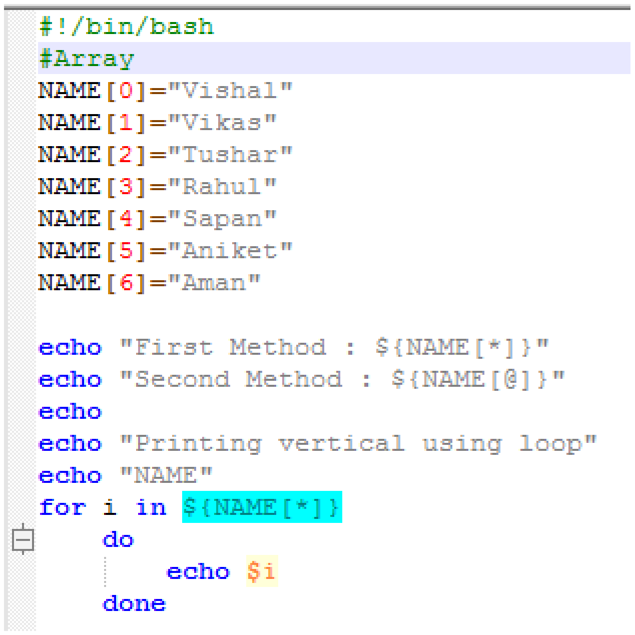
```
#!/bin/bash
NAME="Vikas Bansode"
readonly NAME
echo $NAME
```



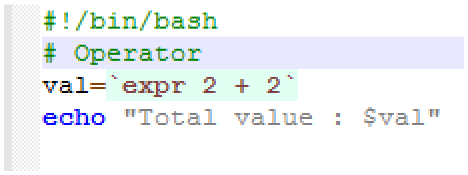
```
#!/bin/bash
NAME="Vikas Bansode"
#NAME is a variable.
echo $NAME
#NAME variable called using $ sign. OR
#You can call it like below
echo ${NAME}
# both are valid
```



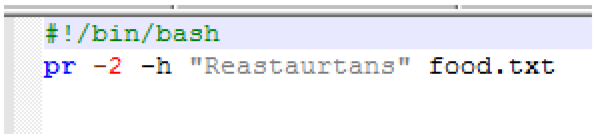
```
#!/bin/bash
#Unset_Variable
NAME="Vikas Bansode"
unset NAME
echo $NAME
```



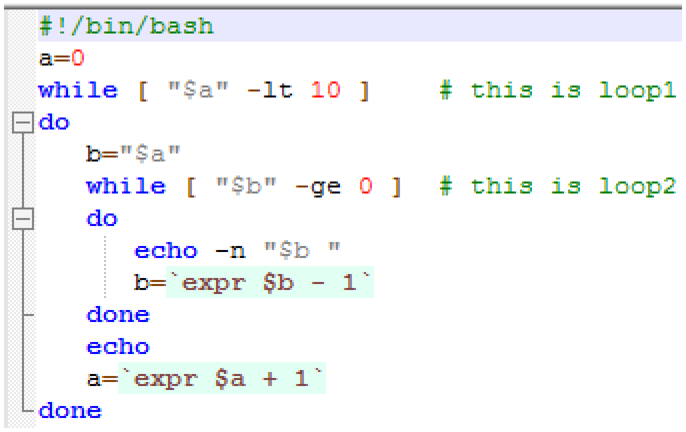
```
#!/bin/bash
#Array
NAME[0]="Vishal"
NAME[1]="Vikas"
NAME[2]="Tushar"
NAME[3]="Rahul"
NAME[4]="Sapan"
NAME[5]="Aniket"
NAME[6]="Aman"

echo "First Method : ${NAME[*]}"
echo "Second Method : ${NAME[@]}"
echo
echo "Printing vertical using loop"
echo "NAME"
for i in ${NAME[*]}
    do
        echo $i
    done
```



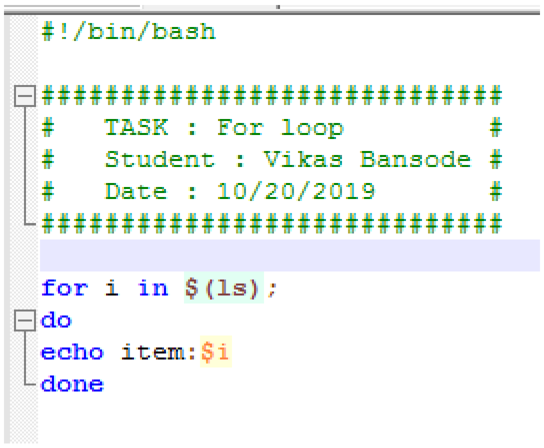
```bash
#!/bin/bash
# Operator
val=`expr 2 + 2`
echo "Total value : $val"
```



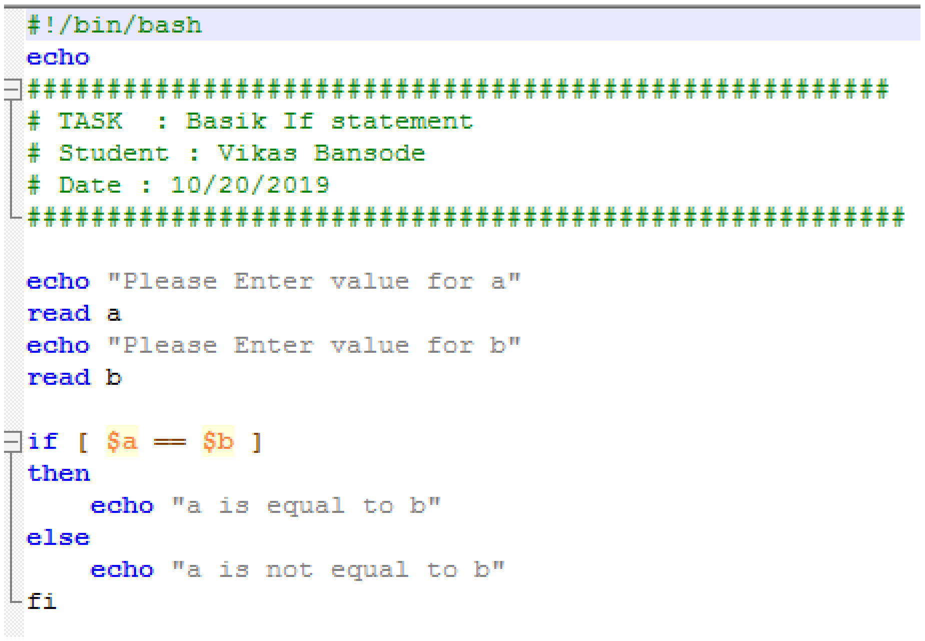
```bash
#!/bin/bash
pr -2 -h "Reastaurtans" food.txt
```



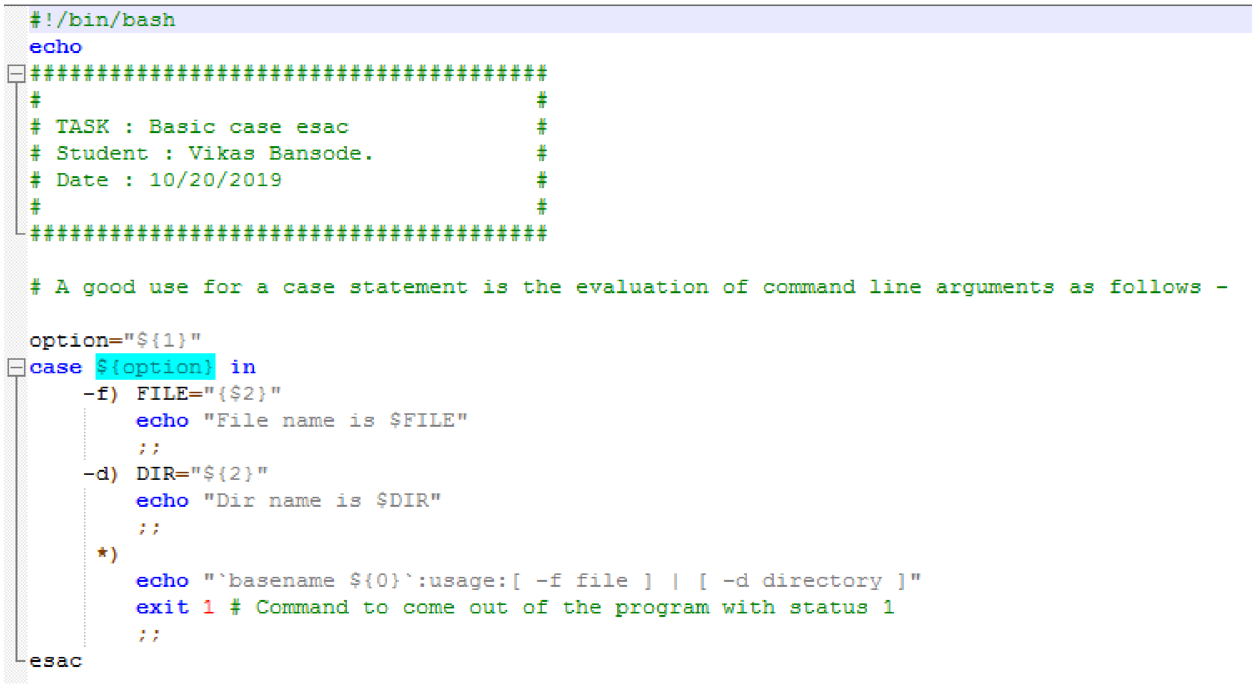
```bash
#!/bin/bash
a=0
while [ "$a" -lt 10 ]     # this is loop1
do
   b="$a"
   while [ "$b" -ge 0 ]  # this is loop2
   do
      echo -n "$b "
      b=`expr $b - 1`
   done
   echo
   a=`expr $a + 1`
done
```



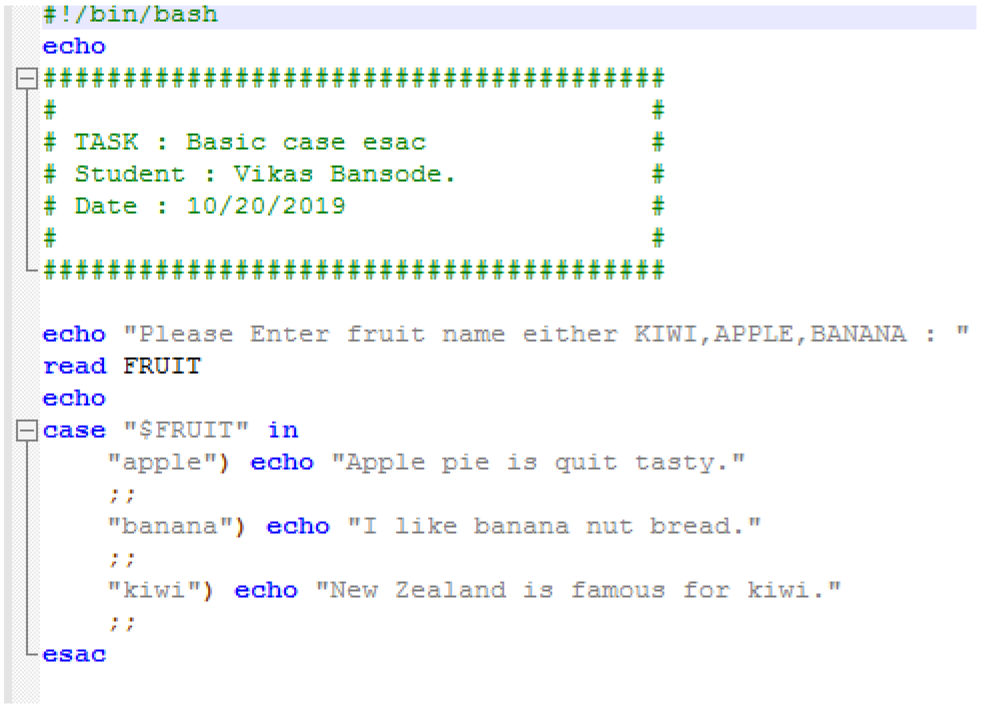
```bash
#!/bin/bash

#############################
#    TASK : For loop        #
#    Student : Vikas Bansode #
#    Date : 10/20/2019       #
#############################

for i in $(ls);
do
echo item:$i
done
```



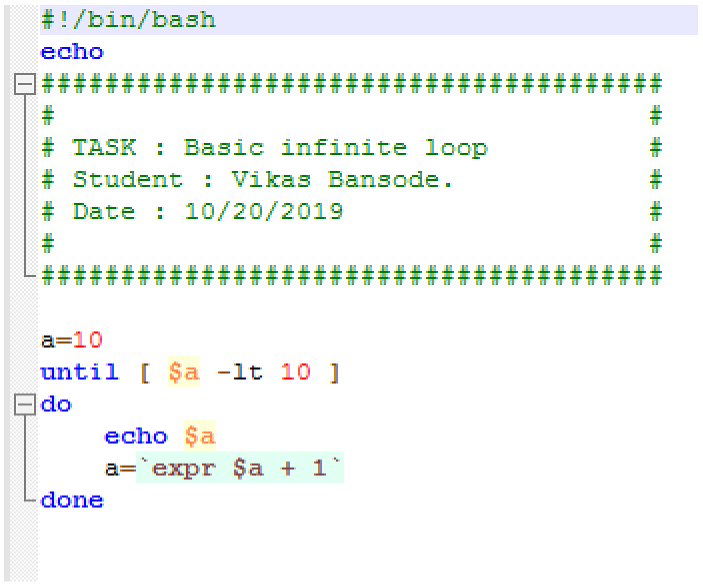
```bash
#!/bin/bash
echo
#######################################################
# TASK  : Basik If statement
# Student : Vikas Bansode
# Date : 10/20/2019
########################################################

echo "Please Enter value for a"
read a
echo "Please Enter value for b"
read b

if [ $a == $b ]
then
    echo "a is equal to b"
else
    echo "a is not equal to b"
fi
```



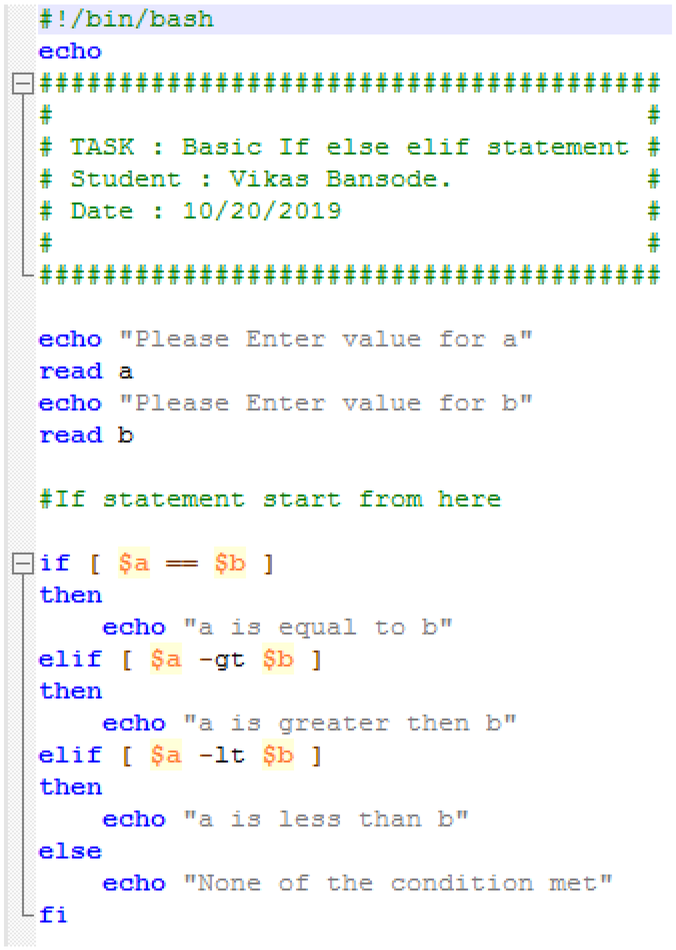
```bash
#!/bin/bash
echo
#######################################
#                                     #
# TASK : Basic case esac              #
# Student : Vikas Bansode.            #
# Date : 10/20/2019                   #
#                                     #
#######################################

# A good use for a case statement is the evaluation of command line arguments as follows -

option="${1}"
case ${option} in
    -f) FILE="{$2}"
        echo "File name is $FILE"
        ;;
    -d) DIR="${2}"
        echo "Dir name is $DIR"
        ;;
     *)
        echo "`basename ${0}`:usage:[ -f file ] | [ -d directory ]"
        exit 1 # Command to come out of the program with status 1
        ;;
esac
```



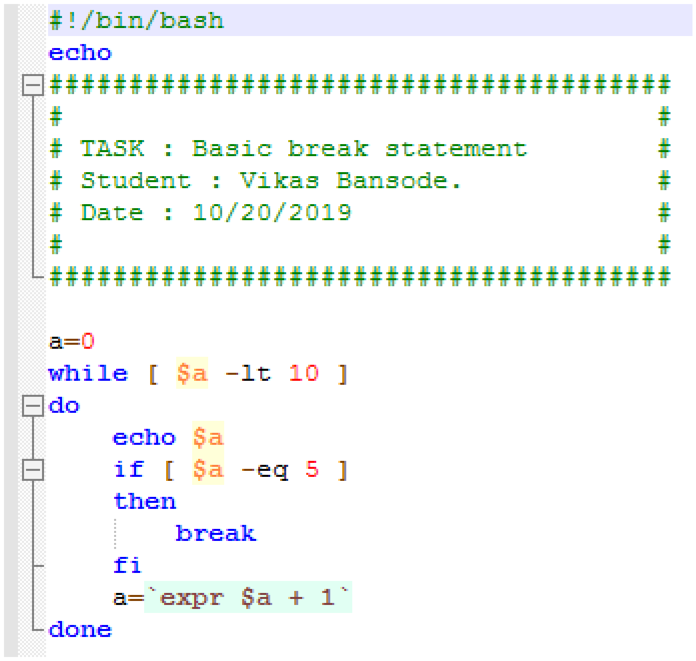
```bash
#!/bin/bash
echo
########################################
#                                      #
# TASK : Basic case esac               #
# Student : Vikas Bansode.             #
# Date : 10/20/2019                    #
#                                      #
########################################

echo "Please Enter fruit name either KIWI,APPLE,BANANA : "
read FRUIT
echo
case "$FRUIT" in
    "apple") echo "Apple pie is quit tasty."
    ;;
    "banana") echo "I like banana nut bread."
    ;;
    "kiwi") echo "New Zealand is famous for kiwi."
    ;;
esac
```



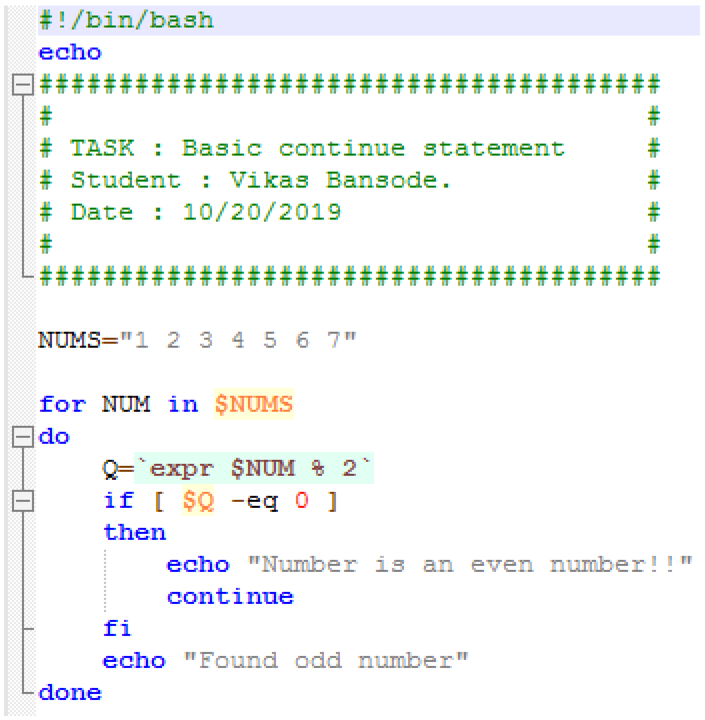
```bash
#!/bin/bash
echo
########################################
#                                      #
# TASK : Basic infinite loop           #
# Student : Vikas Bansode.             #
# Date : 10/20/2019                    #
#                                      #
########################################

a=10
until [ $a -lt 10 ]
do
    echo $a
    a=`expr $a + 1`
done
```



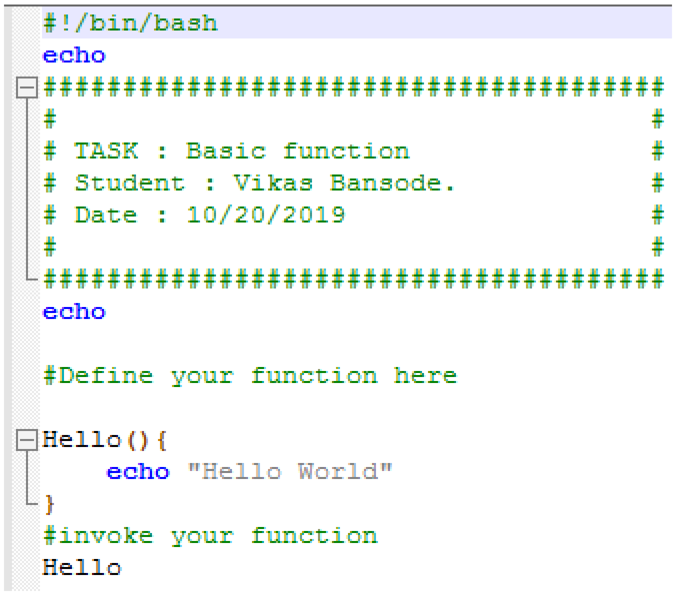
```
#!/bin/bash
echo
#######################################
#                                     #
# TASK : Basic If else elif statement #
# Student : Vikas Bansode.            #
# Date : 10/20/2019                   #
#                                     #
#######################################

echo "Please Enter value for a"
read a
echo "Please Enter value for b"
read b

#If statement start from here

if [ $a == $b ]
then
    echo "a is equal to b"
elif [ $a -gt $b ]
then
    echo "a is greater then b"
elif [ $a -lt $b ]
then
    echo "a is less than b"
else
    echo "None of the condition met"
fi
```



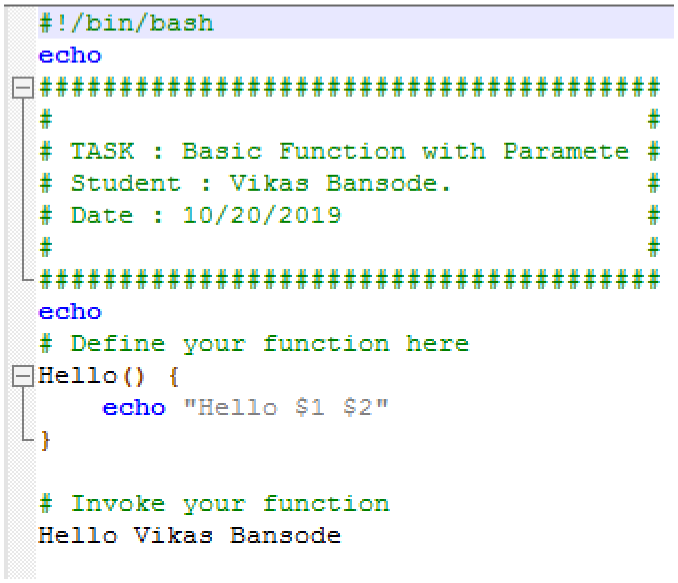
```
#!/bin/bash
echo
#######################################
#                                     #
# TASK : Basic break statement        #
# Student : Vikas Bansode.            #
# Date : 10/20/2019                   #
#                                     #
#######################################

a=0
while [ $a -lt 10 ]
do
    echo $a
    if [ $a -eq 5 ]
    then
        break
    fi
    a=`expr $a + 1`
done
```



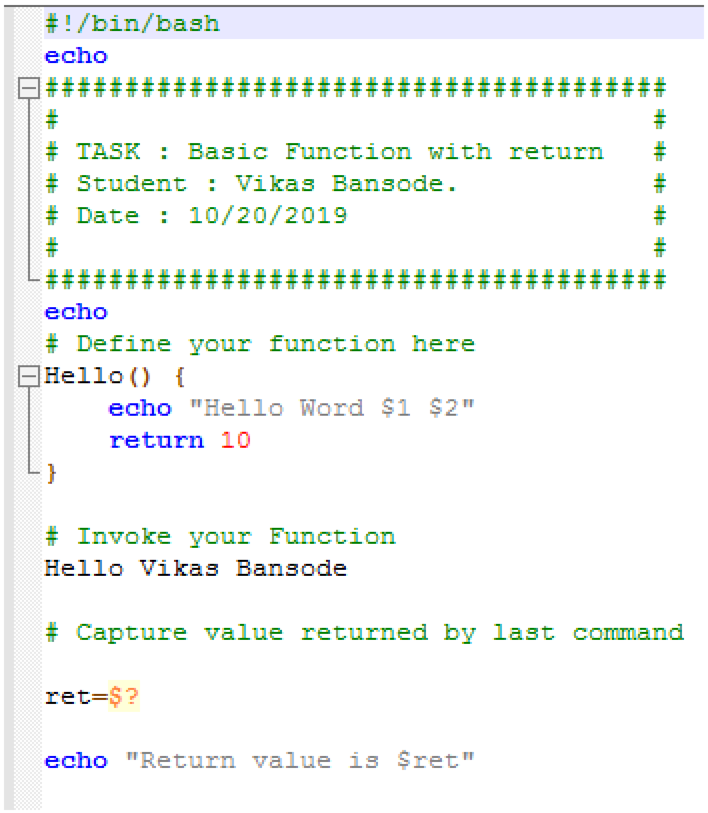
```bash
#!/bin/bash
echo
#######################################
#                                     #
# TASK : Basic continue statement     #
# Student : Vikas Bansode.            #
# Date : 10/20/2019                   #
#                                     #
#######################################

NUMS="1 2 3 4 5 6 7"

for NUM in $NUMS
do
    Q=`expr $NUM % 2`
    if [ $Q -eq 0 ]
    then
        echo "Number is an even number!!"
        continue
    fi
    echo "Found odd number"
done
```



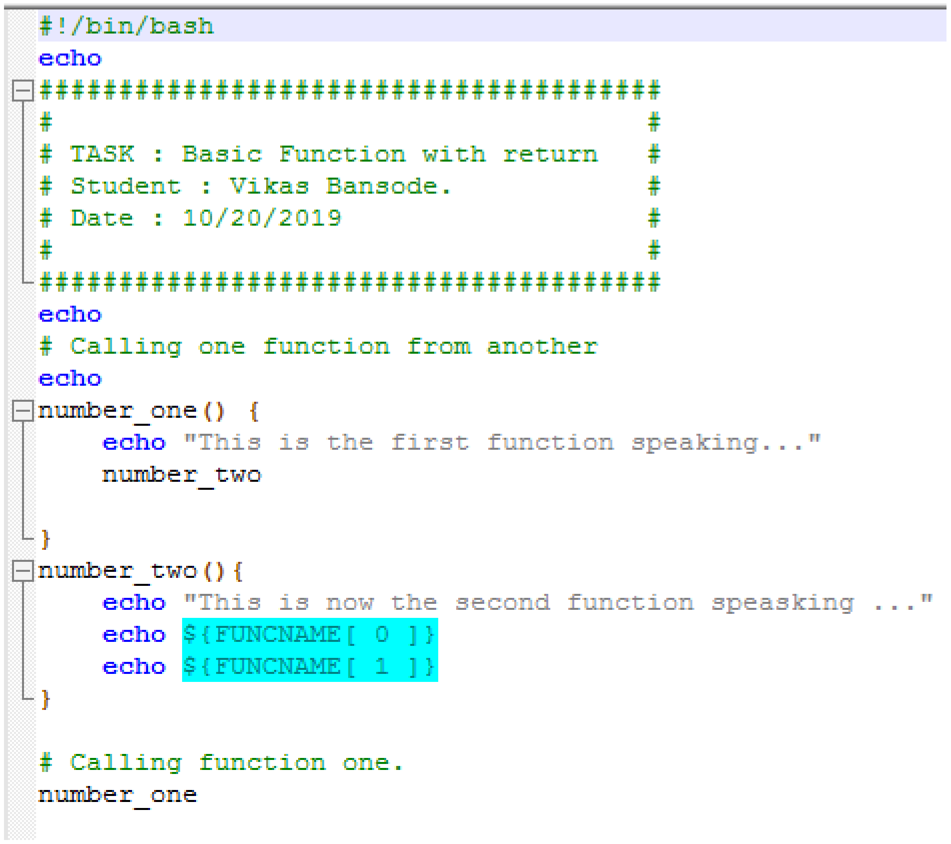
```bash
#!/bin/bash
echo
#######################################
#                                     #
# TASK : Basic function               #
# Student : Vikas Bansode.            #
# Date : 10/20/2019                   #
#                                     #
#######################################
echo

#Define your function here

Hello(){
    echo "Hello World"
}
#invoke your function
Hello
```



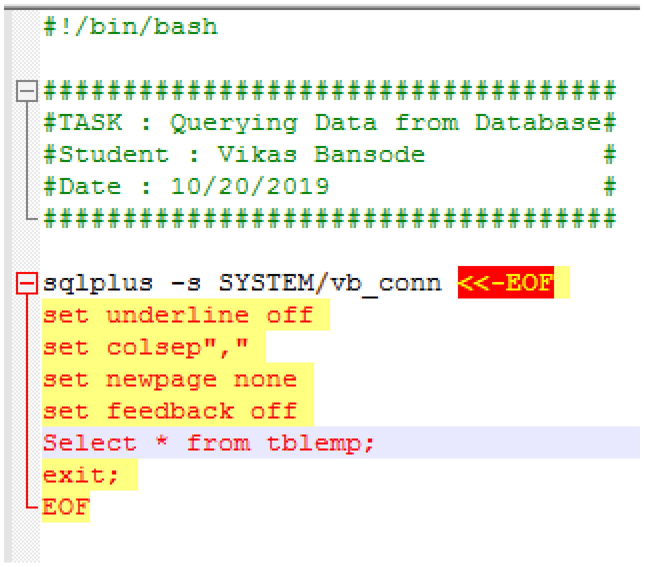
```
#!/bin/bash
echo
#######################################
#                                     #
# TASK : Basic Function with Paramete #
# Student : Vikas Bansode.            #
# Date : 10/20/2019                   #
#                                     #
#######################################
echo
# Define your function here
Hello() {
    echo "Hello $1 $2"
}

# Invoke your function
Hello Vikas Bansode
```



```
#!/bin/bash
echo
#######################################
#                                     #
# TASK : Basic Function with return   #
# Student : Vikas Bansode.            #
# Date : 10/20/2019                   #
#                                     #
#######################################
echo
# Define your function here
Hello() {
    echo "Hello Word $1 $2"
    return 10
}

# Invoke your Function
Hello Vikas Bansode

# Capture value returned by last command

ret=$?

echo "Return value is $ret"
```



```
#!/bin/bash
echo
#######################################
#                                     #
# TASK : Basic Function with return   #
# Student : Vikas Bansode.            #
# Date : 10/20/2019                   #
#                                     #
#######################################
echo
# Calling one function from another
echo
number_one() {
    echo "This is the first function speaking..."
    number_two

}
number_two(){
    echo "This is now the second function speasking ..."
    echo ${FUNCNAME[ 0 ]}
    echo ${FUNCNAME[ 1 ]}
}

# Calling function one.
number_one
```



```
#!/bin/bash

####################################
#TASK : Querying Data from Database#
#Student : Vikas Bansode           #
#Date : 10/20/2019                 #
####################################

sqlplus -s SYSTEM/vb_conn <<-EOF
set underline off
set colsep","
set newpage none
set feedback off
Select * from tblemp;
exit;
EOF
```

➢ Chapter 4 : AWK Programming

• AWK > used for data manipulation & report generation.

Create below csv file using cat > df.csv and type below data in blank space to play with commands

Item_Name	Amount
Item1	200
Item2	500
Item3	900
Item2	800
Item1	600

1. Data Manipulation & Report generation

Solved Example with underneath Description
awk -F"," '{x+=$2}END{print x}' df.csv
> to find the sum of all numbers
awk -F"," 'END{print NR}' df.csv
> to count the number of records in file
awk -F"," '$1=="Item1" {x+=$2;}END{print x}' df.csv
> to find the sum of particular group entry alone
awk -F, '{a[$1];}END{for (i in a)print i;}' df.csv
> to find unique values of first columns
awk -F, '{a[$1]+=$2;}END{for(i in a)print i", "a[i];}' df.csv
> To find the sum of individual group records.
awk -F"," '{x+=$2;print}END{print "Total,"x}' df.csv
> To find the sum of all entries in second column and add it as the last record.
awk -F, '{if (a[$1] < $2)a[$1]=$2;}END{for(i in a){print i,a[i];}}' OFS=, df.csv
> To print the maximum or the biggest record of every group
awk -F, '{a[$1]++;}END{for (i in a)print i, a[i];}' df.csv
> To find the count of entries against every group
awk -F, '!a[$1]++' df.csv
> To print only the first record of every group
awk -F"," 'FNR==NR{a[$1];next}!($1 in a)'
> To print non matching records not in right file & vice versa
awk 'FNR==1 && NR!=1{next;}{print}' *.csv > ARDevice.csv
> To Merging files

2. Pattern matching with awk

Solved Example with underneath Description
awk '$0 ~ /North/{print}' df.csv
> To print only the records containing North
awk '/Rent/{print}' df.csv
> while doing pattern matching, by default does on the entire line, and hence $0 can be left off
awk '/Rent/' df.csv
> Since awk prints the line by default on a true condition, print statement can also be left off.
awk -F, '$1 ~ /Rent/' df.csv
> however, the pattern we are looking for is only on the first column.
awk -F, '$1=="North"' df.csv
> To match exactly for the word "Rent" in the first column:
awk -F, '$1 == "Medicine"{print $2}' df.csv
> To print only the 2nd column for all "Medicine" records
awk '/Rent|Medicine/' df.csv
> To match for patterns "Rent" or "Medicine" in the file:
awk -F, '$1 ~ /Rent|Medicine/' df.csv
> Similarly, to match for this above pattern only in the first column:
awk '!/Medicine/' df.csv
> To print the lines which does not contain the pattern Medicine
awk -F, '$1 !~ /Medicine/' df.csv
> To negate the pattern only on the first column alone.
awk -F, '{gsub ("-"," ",$5);print $5}' df.csv
> to print the abs
awk -F, '$2>500' df.csv
> To print all records whose amount is greater than 500
awk 'NR==1 && /Medicine/' df.csv
> To print the Medicine record only if it is the 1st record:
awk -F, '/Medicine/ && $2>500' df.csv
> To print all those Medicine records whose amount is greater than 500
awk -F, '/Medicine/ || $2>600' df.csv
> To print all the Medicine records and also those records whose amount is greater than 600

➢ Chapter 5 : GREP Programming

•**grep** > is used to search text or searches the given file for lines containing a match to the given strings or words.

Solved Examples	Description
grep "literal string" df.csv	> Search for the given string in a single file
grep "string" df.csv	> Checking for the given string in multiple files
grep -i "string" df.csv	> Case insensitive search using grep -i
grep -iw "is" df.csv	> Checking for full words, not for sub-strings using grep -w
grep -A 3 -i "string" df.csv	> Display N lines after match
grep -B 3 "string" df.csv	> Display N lines before match
grep -C 2 "string" df.csv	> Display N lines around match
grep -r "string" *	> Searching in all files recursively using grep -r
grep -v "go" df.csv	> Invert match using grep -v
grep -v -e "pattern" -e "pattern" df.csv	> display the lines which does not matches all the given pattern
grep -c "go" df.csv	> Counting the number of matches using grep -c
grep -o "is.*line" df.csv	> Show only the matched string
grep -o -b "3" df.csv	> Show the position of match in the line
grep -n "go" df.csv	> Show line number while displaying the output using grep -n

➢ Chapter 6 : SED Programming

•Sed > receives text input as a “stream” and edits the stream according to your instructions.

1. File Spacing

Solved Example	Description
sed G df.csv	> Insert one blank line after each line
sed 'G;G' df.csv	> To insert two blank lines
sed '/^$/d;G' df.csv	> Delete blank lines and insert one blank line after each line
sed '/love/{x;p;x;}' df.csv	> Insert a black line above every line which matches “love”
sed 's/^/ /' a.txt	> Insert 5 spaces to the left of every lines

3. Numbering Lines

Solved Examples with Description below
sed = a.txt | sed 'N;s/\n/\t/' df.csv
> Number each line of a file (left alignment). **=** is used to number the line. \t is used for tab between number and sentence –
sed = a.txt | sed 'N; s/^/   /; s/ *\(.\{4,\}\)\n/\1 /'
> Number each line of a file (number on left, right-aligned). This command is similar to \`cat -n filename\`.
sed '/./=' a.txt | sed '/./N; s/\n/ /'
> Number each line of file, only if line is not blank

4. Deleting Lines

Solved Examples	Description
sed '5d' a.txt	> Delete a particular line
sed ‘$d’ filename	> Delete the last line
sed '3,5d' a.txt filename	> Delete line from range x to y
sed '2,$d' a.txt	> Delete from nth to last line
sed '/life/d' a.txt	> Delete the pattern matching line
sed '3~2d' a.txt	> Delete lines starting from nth line and every 2nd line from there
sed '/easy/,+2d' a.txt	> Delete the lines which matches the pattern and 2 lines after to that
sed '/^$/d' a.txt	> Delete blank Lines
sed -i '/^#/d;/^$/d' a.txt	> Delete empty lines or those begins with “#”

5. View/Print the files

Solved Examples	Description
sed -n '2,5p' a.txt	> Viewing a file from x to y range
sed '2,4d' a.txt	> View the entire file except the given range
sed -n '4'p a.txt	> Print nth line of the file
sed -n '4,6'p a.txt	> Print lines from x^{th} line to y^{th} line
sed -n '3,$'p a.txt	> Print from nth line to end of file

6. Pattern matching

Solved Examples	Description
sed -n /every/p a.txt	> Print the line only which matches the pattern
sed -n '/everyone/,5p' a.txt	> Print lines which matches the pattern i.e. from input to xth line
sed -n '1,/everyone/p' a.txt	> Prints lines from the xth line of the input, up-to the line which matches the pattern. If the pattern doesn't found then it prints up-to end of the file.
sed -n '/learn/,+2p' a.txt	> Print the lines which matches the pattern up-to the next xth lines

7. Substitution

Solved Examples	Description
sed 's/life/leaves/' a.txt	> Change the first occurrence of the pattern
sed 's/to/two/2' a.txt	> Replacing the nth occurrence of a pattern in a line
sed 's/life/learn/g' a.txt	> Replacing all the occurrence of the pattern in a line.
sed 's/to/TWO/2g' a.txt	> Replace pattern from nth occurrence to all occurrences in a line
sed -n 's/to/TWO/p' a.txt	> If you wish to print only the replaced lines, then use "-n" option along with "/p" print flag to display only the replaced lines –
sed 's/to/TWO/p' a.txt	> And if you wish to print the replaced lines twice, then only use "/p" print flag without "-n" option
sed '3 s/every/each/' a.txt	> Replacing pattern on a specific line number. Here, "m" is the line number
sed -n '3 s/every/each/p' a.txt	> If you wish to print only the replaced lines –
sed '2,5 s/to/TWO/' a.txt	> Replace string on a defined range of lines
sed 's/life/Love/i' a.txt	> If you wish to replace pattern in order to ignore character case (beginning with uppercase or lowercase), then there are two ways to replace such patterns –

sed 's/ */ /g' a.txt	> To replace multiple spaces with a single space
sed '/is/ s/live/love/' a.txt	> Replace one pattern followed by the another pattern –
sed -i '5!s/life/love/' a.txt	> Replace a pattern with other except in the nth line

➢ Chapter 7 : Pattern Matching Descriptions

Pattern	Description
^	> Matches the beginning of lines
$	> Matches the end of lines
.	> Matches any single character
/a.c/	> Matches lines that contain strings such as a+c, a-c, abc, match, and a3c
/a*c/	> Matches the same strings along with strings such as ace, yacc, and arctic
/[tT]he/	> Matches the string The and the
/^$/	> Matches blank lines
/^.*$/	> Matches an entire line whatever it is
/ */	> Matches one or more spaces
/^$/	> Matches blank lines
[a-z]	> Matches a single lowercase letter
[A-Z]	> Matches a single uppercase letter
[a-zA-Z]	> Matches a single letter
[0-9]	> Matches a single number
[a-zA-Z0-9]	> Matches a single letter or number