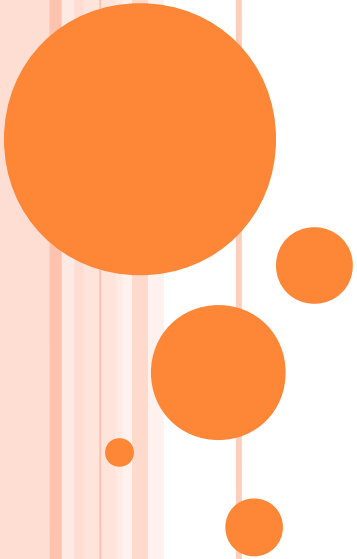


CSCI 330
THE UNIX SYSTEM

Shell Programming

A. Sawarkar

BASH CONTROL STRUCTURES

- if-then-else
- case
- loops
  - for
  - while
  - until
  - select

IF STATEMENT

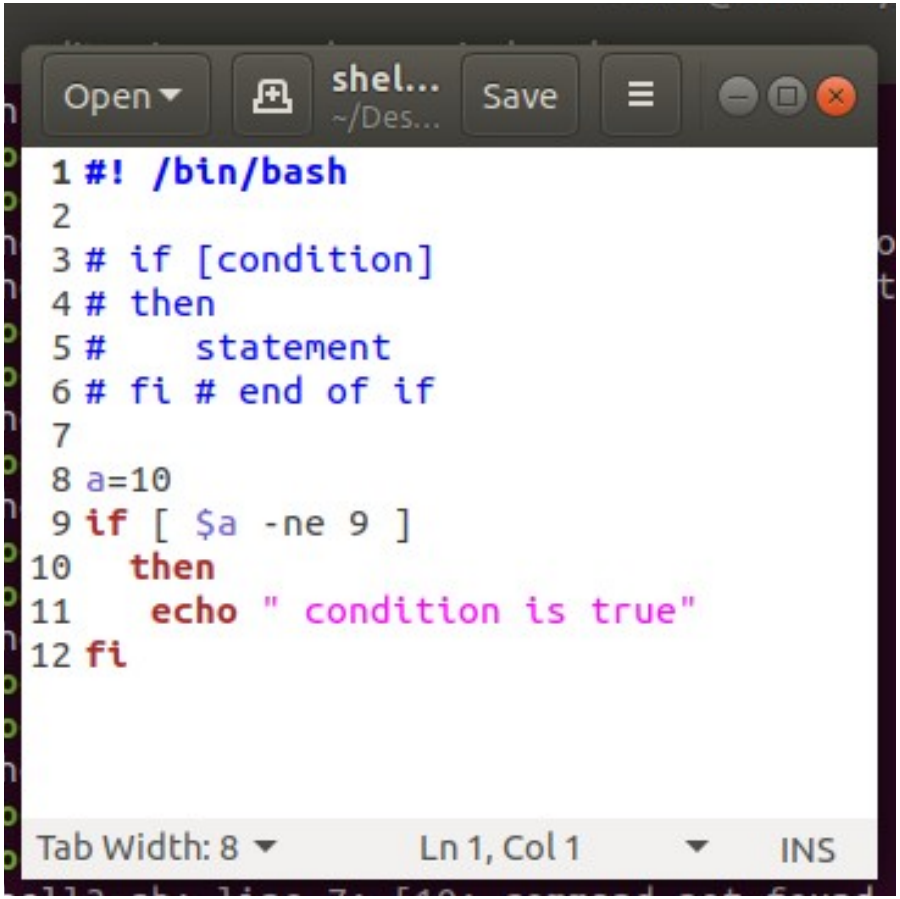
```
if command
then
  statements
fi
```

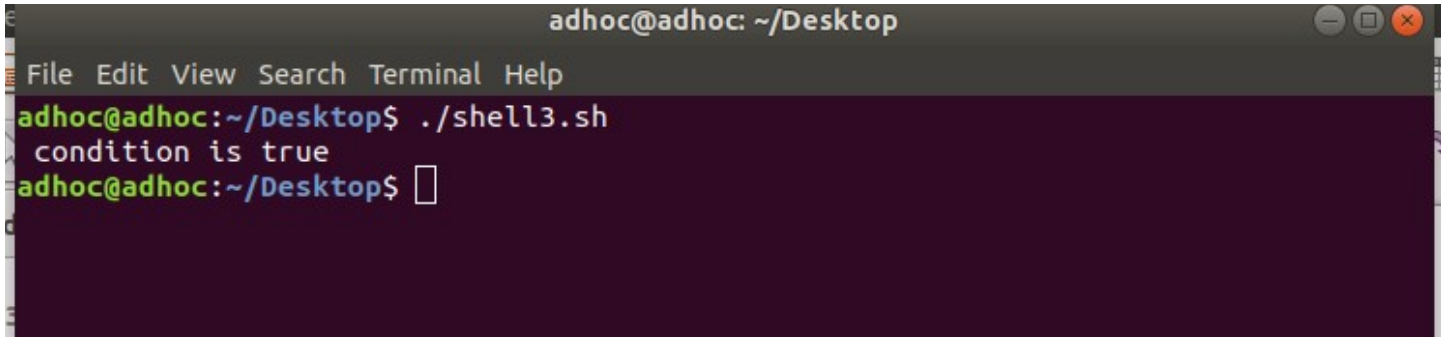
- statements are executed only if **command** succeeds, i.e. has return status “0”

THE SIMPLE IF STATEMENT

```
if [ condition ]
 then
  statements
fi
```

- executes the statements only if **`condition`** is true


Open
shel...
~/Des...
Save
1 #! /bin/bash
2
3 # if [condition]
4 # then
5 # statement
6 # fi # end of if
7
8 a=10
9 if [$a -ne 9]
10 then
11 echo " condition is true"
12 fi
Tab Width: 8
Ln 1, Col 1
INS


adhoc@adhoc: ~/Desktop
File Edit View Search Terminal Help
adhoc@adhoc:~/Desktop$ ./shell3.sh
condition is true
adhoc@adhoc:~/Desktop$

RELATIONAL OPERATORS

Meaning	Numeric	String
Greater than	-gt	
Greater than or equal	-ge	
Less than	-lt	
Less than or equal	-le	
Equal	-eg	= or ==
Not equal	-ne	!=

RELATIONAL OPERATORS

```
integer comparison

-eq - is equal to - if [ "$a" -eq "$b" ]
-ne - is not equal to - if [ "$a" -ne "$b" ]
-gt - is greater than - if [ "$a" -gt "$b" ]
-ge - is greater than or equal to - if [ "$a" -ge "$b" ]
-lt - is less than - if [ "$a" -lt "$b" ]
-le - is less than or equal to - if [ "$a" -le "$b" ]
< - is less than  - (("$a" < "$b"))
<= - is less than or equal to - (("$a" <= "$b"))
> - is greater than - (("$a" > "$b"))
>= - is greater than or equal to  - (("$a" >= "$b"))

string comparison
= - is equal to - if [ "$a" = "$b" ]
== - is equal to - if [ "$a" == "$b" ]
!= - is not equal to - if [ "$a" != "$b" ]
< - is less than, in ASCII alphabetical order - if [[ "$a" < "$b" ]
> - is greater than, in ASCII alphabetical order - if [[ "$a" > "$b
-z - string is null, that is, has zero length
```

THE IF-THEN with Relationa operator If ((cond))...

Open ▾ shell3.sh ~/Desktop Save

```
1 #! /bin/bash
2
3 # if ((<,<=,>,>=))
4 # then
5 #    statement
6 # fi # end of if
7
8 a=10
9 if (($a >= 10))
10   then
11    echo " condition is true"
12 fi
```

sh ▾ Tab Width: 8 ▾ Ln 3, Col 19 ▾ INS

adhoc@adhoc: ~/Desktop

File Edit View Search Terminal Help

```
adhoc@adhoc:~/Desktop$ ./shell3.sh
 condition is true
adhoc@adhoc:~/Desktop$
```

TEST COMMAND

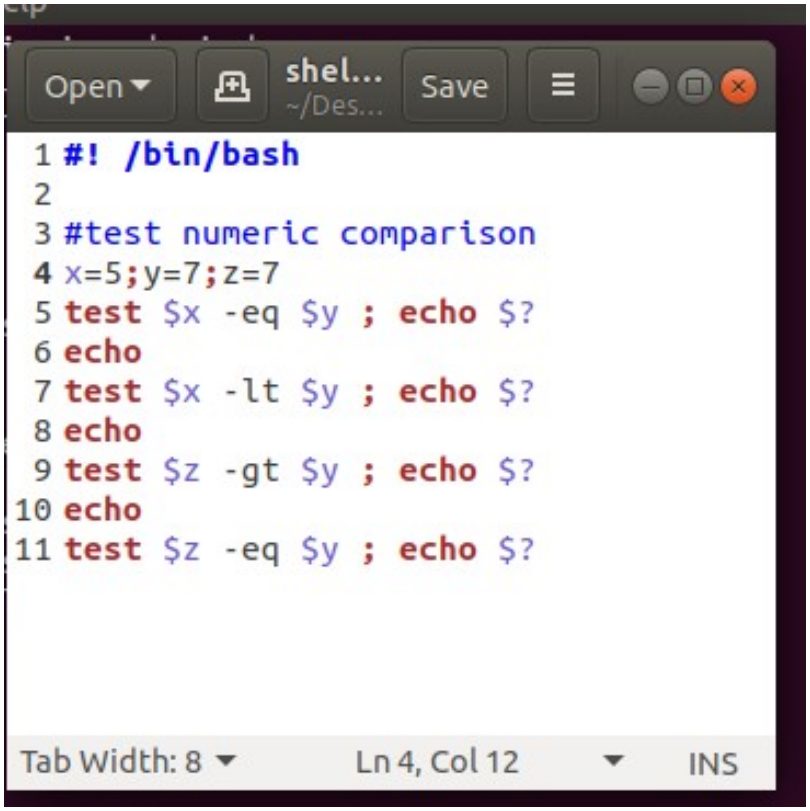
Test work in 3 ways:

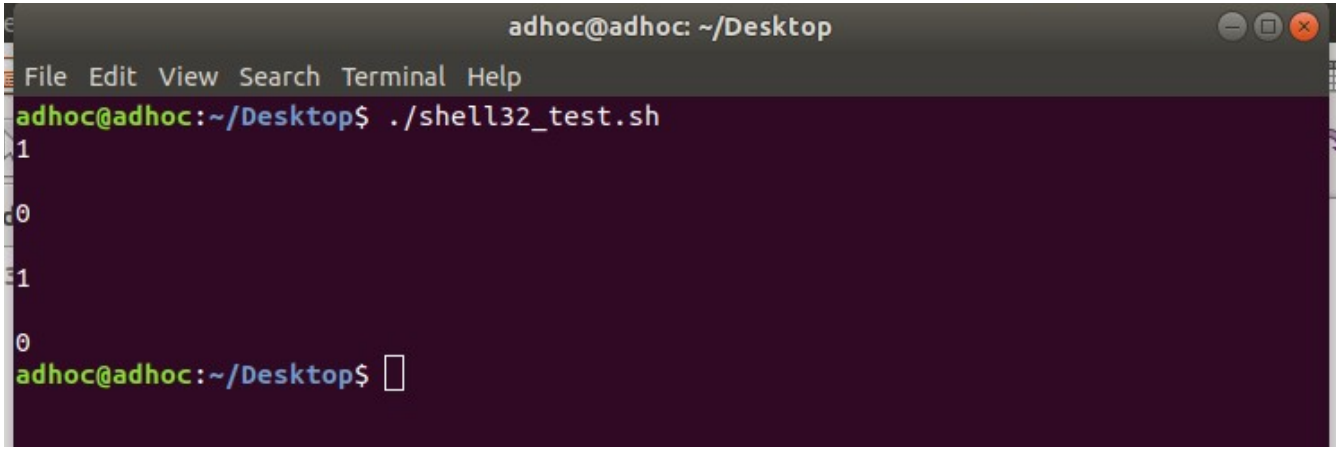
1. Compares two number
2. Compare two string or single one for null
3. Checks a file attributes

Syntax:

```
test expression
[ expression ]
```

- evaluates ‘expression’ and returns true (0) or false(1)


Open
shel...
~/Des...
Save
1 #! /bin/bash
2
3 #test numeric comparison
4 x=5;y=7;z=7
5 test $x -eq $y ; echo $?
6 echo
7 test $x -lt $y ; echo $?
8 echo
9 test $z -gt $y ; echo $?
10 echo
11 test $z -eq $y ; echo $?
Tab Width: 8
Ln 4, Col 12
INS


adhoc@adhoc: ~/Desktop
File Edit View Search Terminal Help
adhoc@adhoc:~/Desktop$ ./shell32_test.sh
1
0
1
0
adhoc@adhoc:~/Desktop$

THE IF-THEN " String" STATEMENT

Open ▾ | shell3.sh ~/Desktop | Save

```
1 #! /bin/bash
2
3 # if with string
4 echo "enter name "
5 read str
6 if [ $str = "unix" ]
7   then
8    echo " condition is true"
9 fi
```

sh ▾ Tab Width: 8 ▾ Ln 6, Col 20 ▾ INS

adhoc@adhoc: ~/Desktop

File Edit View Search Terminal Help

```
adhoc@adhoc:~/Desktop$ ./shell3.sh
enter name
unix
 condition is true
adhoc@adhoc:~/Desktop$
```

THE IF-THEN-ELSE STATEMENT

```
if [ condition ]; then
    statements-1
else
  statements-2
fi
```

- executes statements-1 if condition is true
- executes statements-2 if condition is false

Open ▾ shell3.sh ~/Desktop Save

```
1 #! /bin/bash
2
3 # Even or odd
4 echo "enter number to check even odd "
5 read num
6 rem=$(( $num%2 ))
7 if [ $rem -eq 0 ]
8   then
9    echo " Even"
10 else
11   echo " odd"
12 fi
```

sh ▾ Tab Width: 8 ▾ Ln 6, Col 16 ▾ INS

adhoc@adhoc: ~/Desktop

File Edit View Search Terminal Help

```
adhoc@adhoc:~/Desktop$ ./shell3.sh
enter number to check even odd
24
 Even
adhoc@adhoc:~/Desktop$ ./shell3.sh
enter number to check even odd
25
 odd
adhoc@adhoc:~/Desktop$
```

THE IF…STATEMENT

```
if [ condition ]; then
    statements
elif [ condition ]; then
    statement
else
    statements
fi
```

- The word **elif** stands for “else if”
- It is part of the if statement and cannot be used by itself

Open ▾ shell3.sh ~/Desktop Save

```
1 #! /bin/bash
2 echo "Enter the year (YYYY)"
3 read year
4
5 if [ $((year % 4)) -eq 0 ]
6 then
7      if [ $((year % 100)) -eq 0 ]
8          then
9               if [ $((year % 400)) -eq 0 ]
10              then
11               echo "its a leap year"
12              else
13               echo "its not a leap year"
14               fi
15     else
16         echo "Its a leap year"
17      fi
18 else
19     echo "its not a leap year"
20 fi
```

sh ▾ Tab Width: 8 ▾ Ln 20, Col 1 ▾ INS

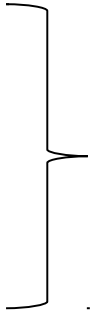
adhoc@a

File Edit View Search Terminal Help

```
adhoc@adhoc:~/Desktop$ ./shell3.sh
Enter the year (YYYY)
4000
its a leap year
adhoc@adhoc:~/Desktop$ ./shell3.sh
Enter the year (YYYY)
1900
its not a leap year
adhoc@adhoc:~/Desktop$ ./shell3.sh
Enter the year (YYYY)
1982
its not a leap year
adhoc@adhoc:~/Desktop$
```

COMPOUND LOGICAL EXPRESSIONS

!	not
&&	and
||	or

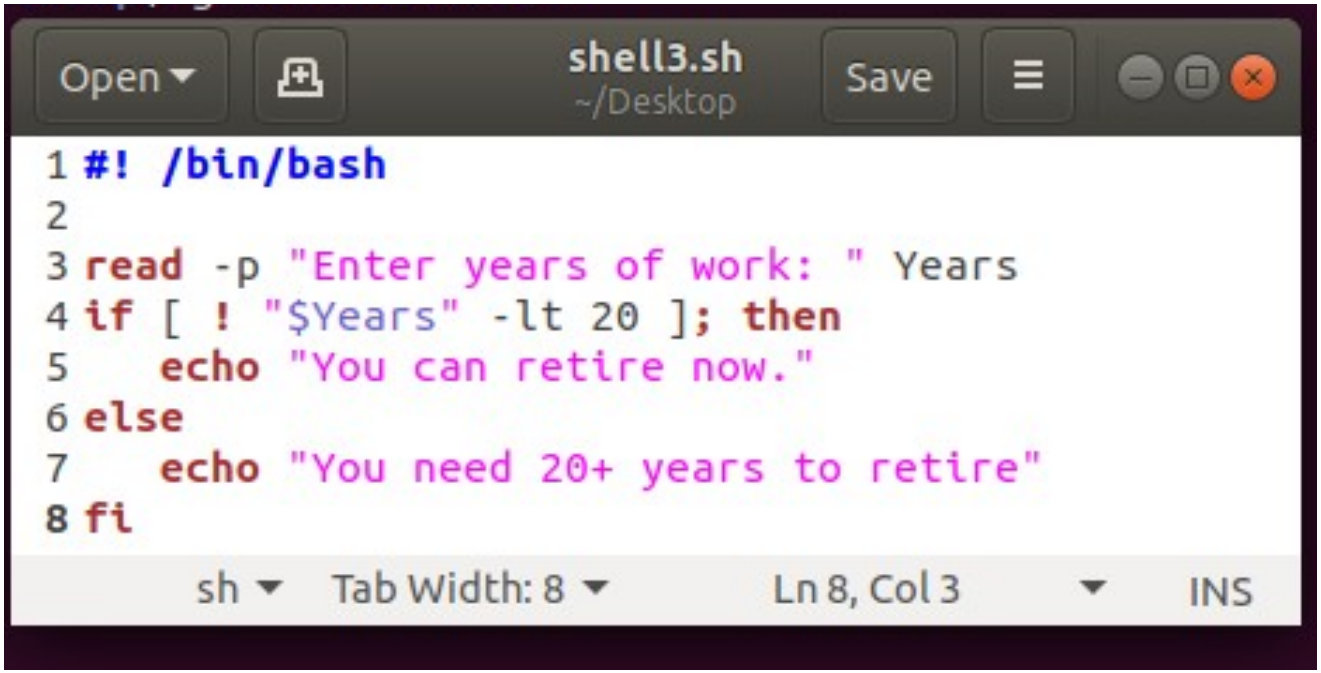


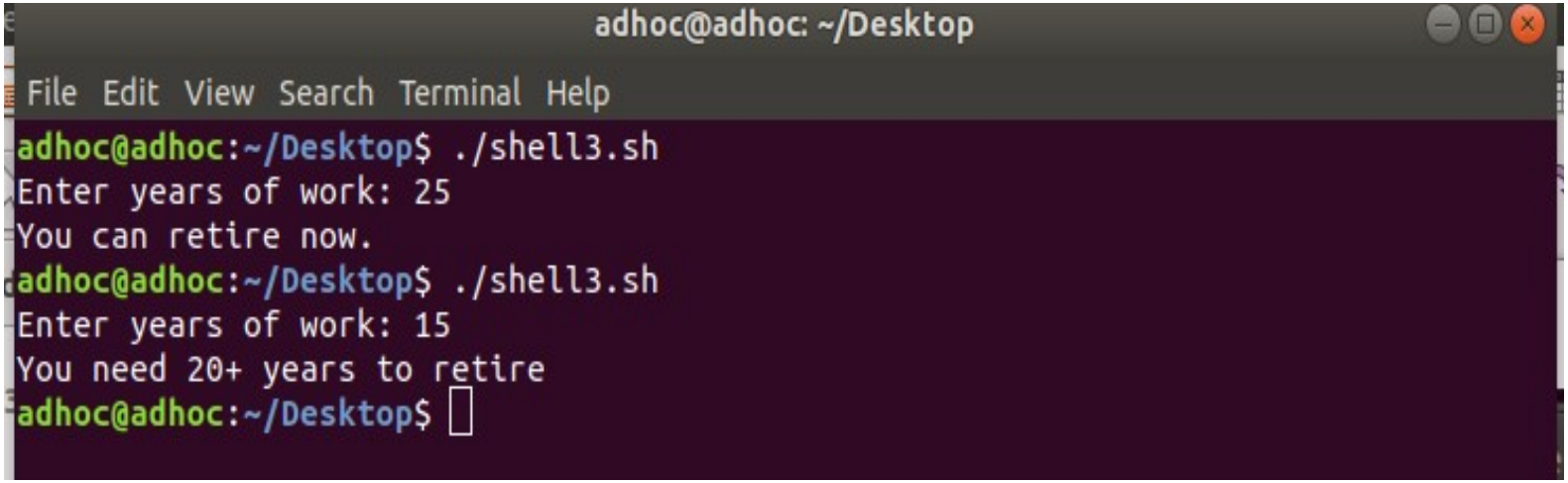
and, or
must be enclosed within

[[ ]]

EXAMPLE: USING THE ! OPERATOR

```
#!/bin/bash

read -p "Enter years of work: " Years
if [ ! "$Years" -lt 20 ]; then
   echo "You can retire now."
else
   echo "You need 20+ years to retire"
fi
```


Open
shell3.sh
~/Desktop
Save
1 #! /bin/bash
2
3 read -p "Enter years of work: " Years
4 if [! "$Years" -lt 20]; then
5 echo "You can retire now."
6 else
7 echo "You need 20+ years to retire"
8 fi
sh
Tab Width: 8
Ln 8, Col 3
INS


adhoc@adhoc: ~/Desktop
File Edit View Search Terminal Help
adhoc@adhoc:~/Desktop$ ./shell3.sh
Enter years of work: 25
You can retire now.
adhoc@adhoc:~/Desktop$ ./shell3.sh
Enter years of work: 15
You need 20+ years to retire
adhoc@adhoc:~/Desktop$

EXAMPLE: USING THE && OPERATOR

```
#!/bin/bash

num=150
if [ $num -gt 100 ] && [ $num -lt 200 ]
then
  echo "The number lies between 100 and
  200"
fi
```

EXAMPLE: USING THE || OPERATOR

```
#!/bin/bash

read -p "Enter calls handled:" CHandle
read -p "Enter calls closed: " CClose
if [[ "$CHandle" -gt 150 || "$CClose" -gt 50 ]]
   then
   echo "You are entitled to a bonus"
else
   echo "You get a bonus if the calls"
   echo "handled exceeds 150 or"
   echo "calls closed exceeds 50"
fi
```

Shell Script (And,Or oprtaor)

day3_shell7.sh (~/day3) - gedit

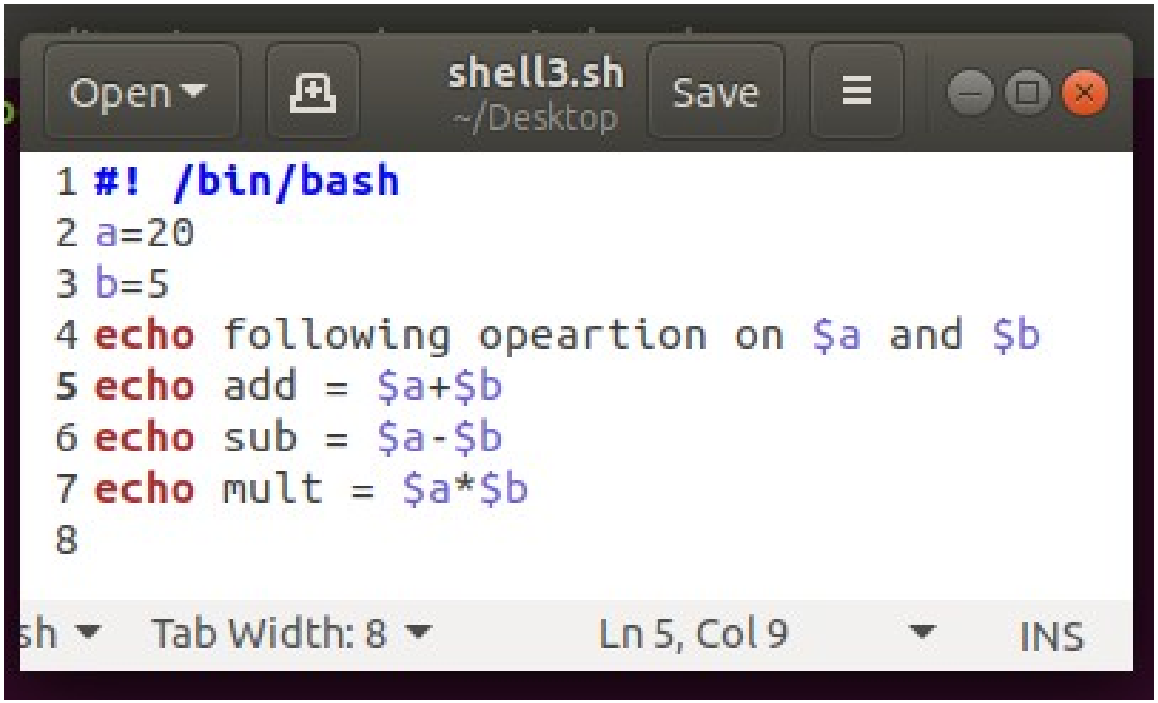
Open ▾ Save

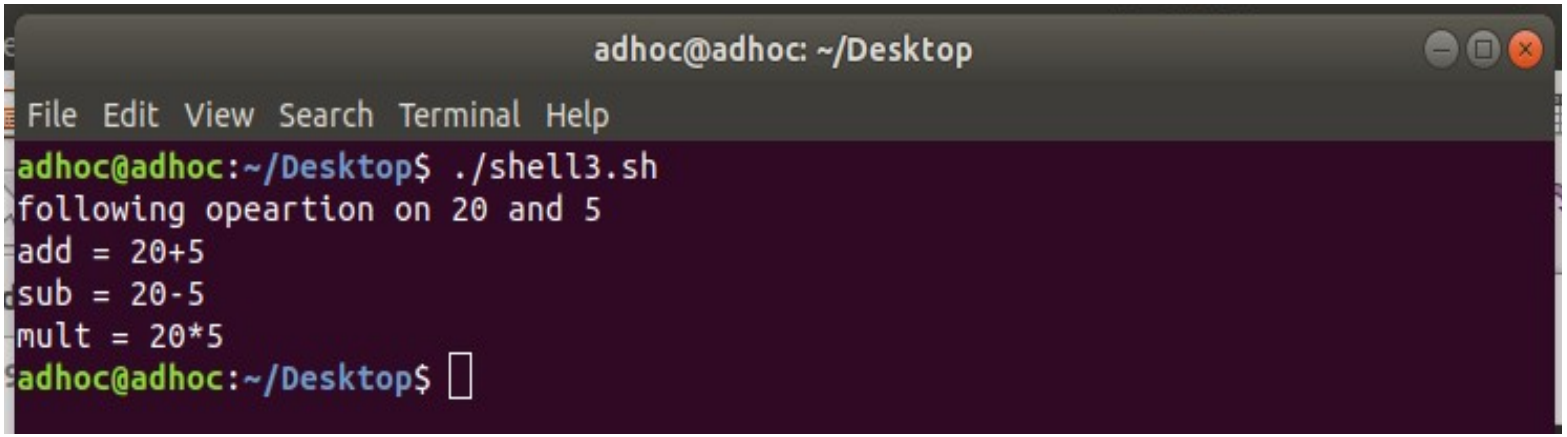
```
#! /bin/bash

income=200000
income=300000

# &&  ===== -a

if [ "$income" -gt 250000 ] && [ "$income" -lt 1000000 ]
if [ "$income" -gt 250000 -a "$income" -lt 100000 ]
if [[ "$income" -gt 250000  &&  "$income" -lt 100000 ]]
 then
 echo " now income is 3L for tax it may > 2.5L < 10L  "
   echo "you have to pay tax"
else
  echo " now income is 2L for tax it may > 2.5L < 10L  "
  echo "you Don't have to pay tax"

fi
#*********************************************

#  || ====  -o
if [ "$income" -gt 250000 ] || [ "$income" -lt 100000 ]
if [ "$income" -gt 250000 -o "$income" -lt 100000 ]
 if [[ "$income" -gt 250000  ||  "$income" -lt 100000 ]]
then
  echo " now income is either > 2.5L or < 10L  "
   echo "you have to pay tax"
else
   echo " now income is nither > 2.5L nor < 10L  "
   echo "you Don't have to pay tax"
fi
```

adhoc@adhoc: ~/day3

```
adhoc@adhoc:~/day3$ ./day3_shell7.sh
 now income is 2L for tax it may > 2.5L < 10L
you Don't have to pay tax
adhoc@adhoc:~/day3$ gedit day3_shell7.sh
adhoc@adhoc:~/day3$ ./day3_shell7.sh
 now income is 3L for tax it may > 2.5L < 10L
you have to pay tax
adhoc@adhoc:~/day3$ gedit day3_shell7.sh
adhoc@adhoc:~/day3$ ./day3_shell7.sh
 now income is nither > 2.5L nor < 10L
you Don't have to pay tax
adhoc@adhoc:~/day3$ gedit day3_shell7.sh
adhoc@adhoc:~/day3$ ./day3_shell7.sh
 now income is either > 2.5L or < 10L
you have to pay tax
adhoc@adhoc:~/day3$ gedit day3_shell7.sh
```

Calculator


Open
shell3.sh
~/Desktop
Save
1 #! /bin/bash
2 a=20
3 b=5
4 echo following opeartion on $a and $b
5 echo add = $a+$b
6 echo sub = $a-$b
7 echo mult = $a*$b
8
sh
Tab Width: 8
Ln 5, Col 9
INS


adhoc@adhoc: ~/Desktop
File Edit View Search Terminal Help
adhoc@adhoc:~/Desktop$ ./shell3.sh
following opeartion on 20 and 5
add = 20+5
sub = 20-5
mult = 20*5
adhoc@adhoc:~/Desktop$

Calculator

Open ▾ | shell3.sh ~/Desktop | Save

```
1 #! /bin/bash
2 a=20
3 b=5
4 #using double bracket(())
5 echo following opeartion on $a and $b
6 echo add = $(( a+b ))
7 echo sub = $(( a-b ))
8 echo mult = $(( a*b ))
9 echo div= $(( a/b ))
10 echo rem=$(( a%b ))
```

Tab Width: 8 ▾ Ln 9, Col 15 ▾ INS

adhoc@adhoc: ~/Desktop

File Edit View Search Terminal Help

```
adhoc@adhoc:~/Desktop$ ./shell3.sh
following opeartion on 20 and 5
add = 25
sub = 15
mult = 100
div= 4
rem=0
adhoc@adhoc:~/Desktop$
```

Calculator
Using expr and single bracket

Open ▾ shell3.sh ~/Desktop Save

```
1 #! /bin/bash
2 a=20
3 b=5
4 #using double bracket(())
5 echo following opeartion on $a and $b
6 echo add = $(expr $a + $b )
7 echo sub = $(expr $a - $b )
8 echo mul = $(expr $a * $b )
9 echo "for expr * not work use \*"
10 echo mul= $(expr $a \* $b )
11 echo div=$(expr $a / $b )
12 echo rem=$(expr $a % $b )
13
```

Tab Width: 8 ▾ Ln 12, Col 22 ▾ INS

adhoc@adhoc: ~/Desktop

File Edit View Search Terminal Help

```
adhoc@adhoc:~/Desktop$ ./shell3.sh
following opeartion on 20 and 5
add = 25
sub = 15
expr: syntax error
mul =
for expr * not work use \*
mul= 100
div=4
rem=0
adhoc@adhoc:~/Desktop$
```

Flaoting calculator

Open ▾ | *shell3.sh ~/Desktop | Save

```
1 #! /bin/bash
2 a=10.5
3 b=5
4 echo " for fraction / decimal use
5 special tool bc- basic caculator"
6 echo
7 echo "add sub mul div as for $a and $b "
8 echo "$a+$b" | bc
9 echo "$a-$b" | bc
10 echo "$a*$b" | bc
11 echo "$a/$b" | bc
12 echo " division not correct"
```

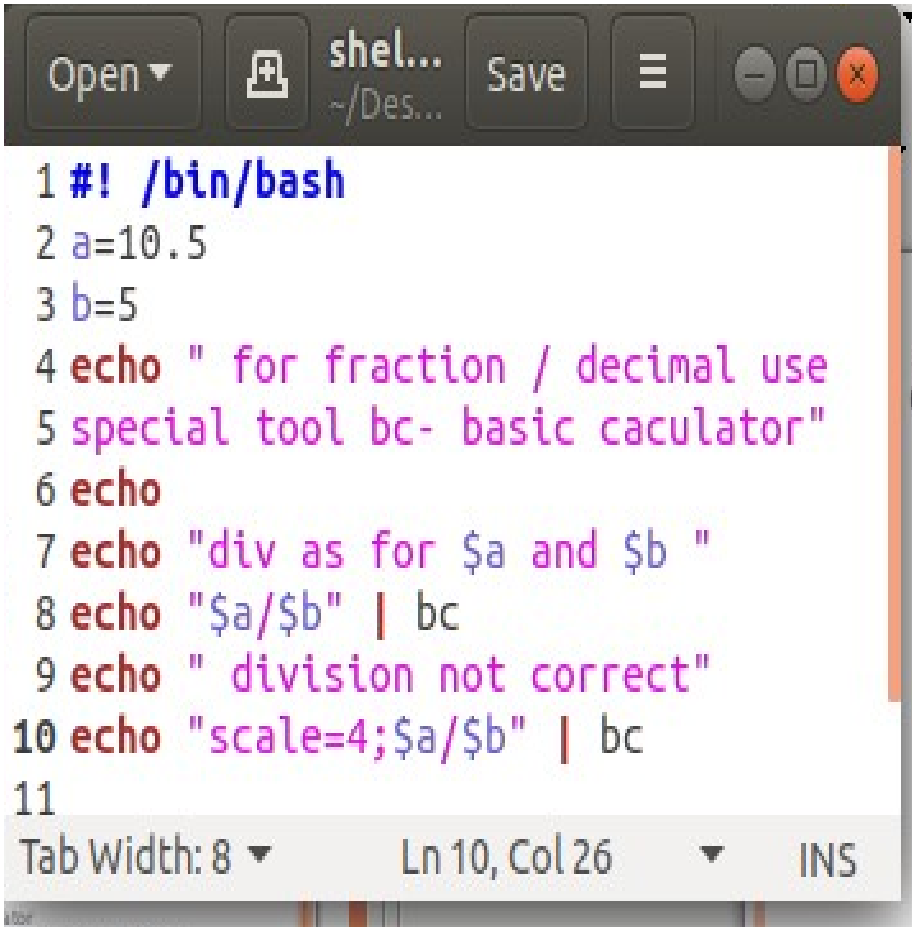
sh ▾ Tab Width: 8 ▾ Ln 12, Col 29 ▾ INS

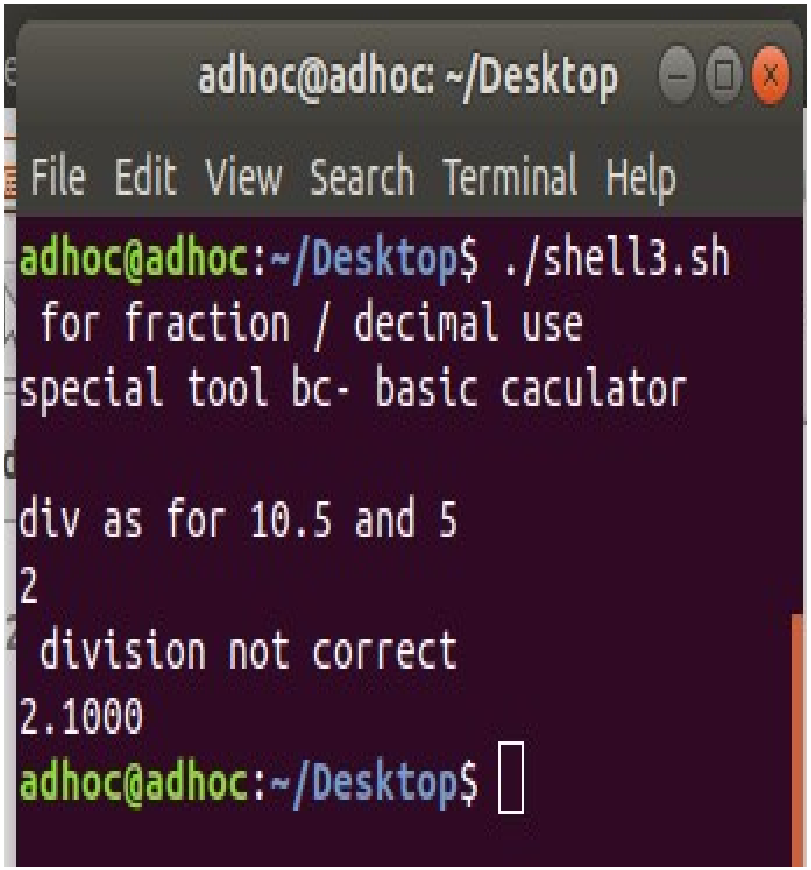
adhoc@adhoc: ~/Desktop

File Edit View Search Terminal Help

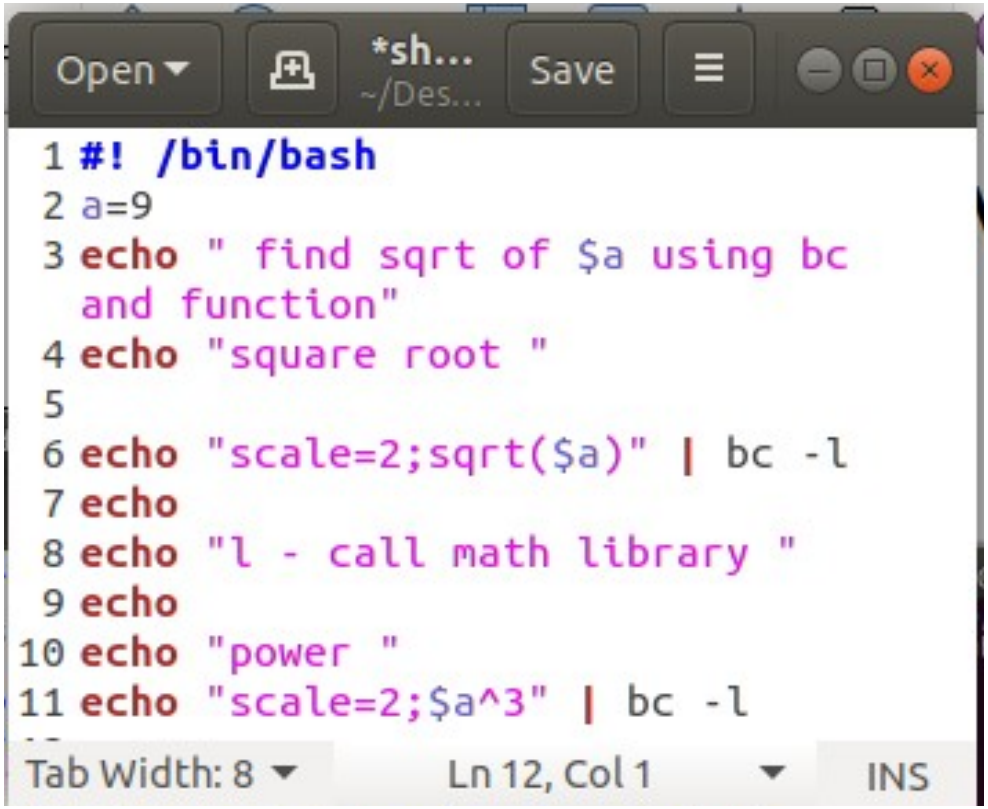
```
adhoc@adhoc:~/Desktop$ ./shell3.sh
 for fraction / decimal use
special tool bc- basic caculator

add sub mul div as for 10.5 and 5
15.5
5.5
52.5
2
 division not correct
adhoc@adhoc:~/Desktop$
```

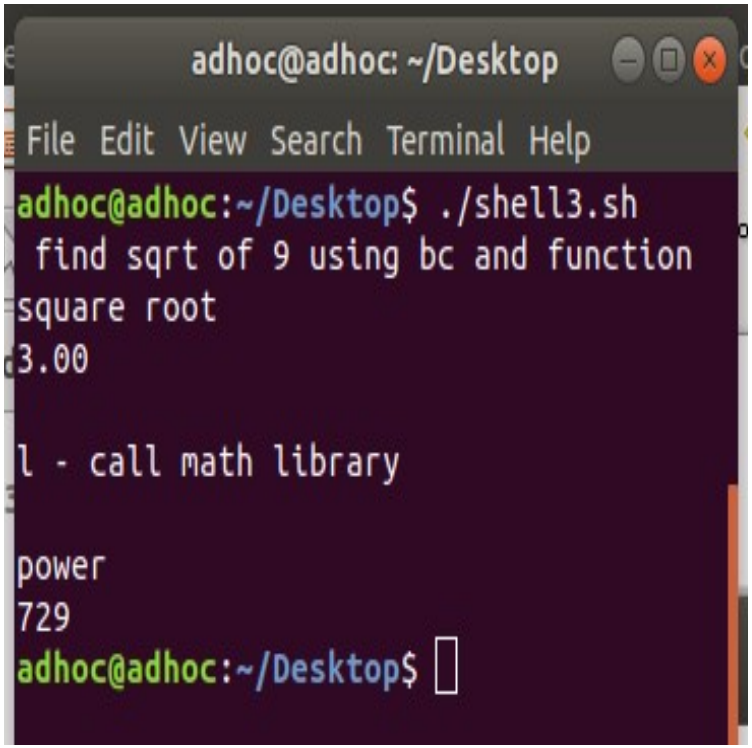
Flaoting calculator (Division - scale)





Flaoting calculator (math function)





Questions..

1. How to assigen floationg result of expression to any variable.
2. check enterd number is even or odd
3. Greatest of 3 number
4. Palindron for 4 digit number

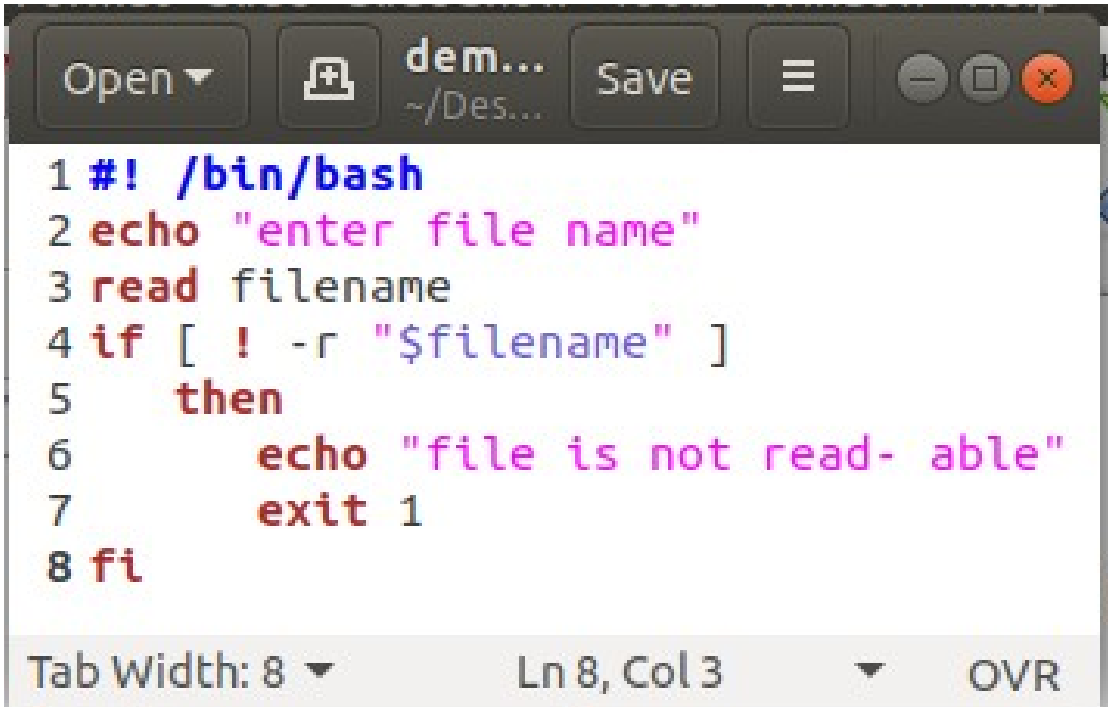
Day4- File Handling and Case

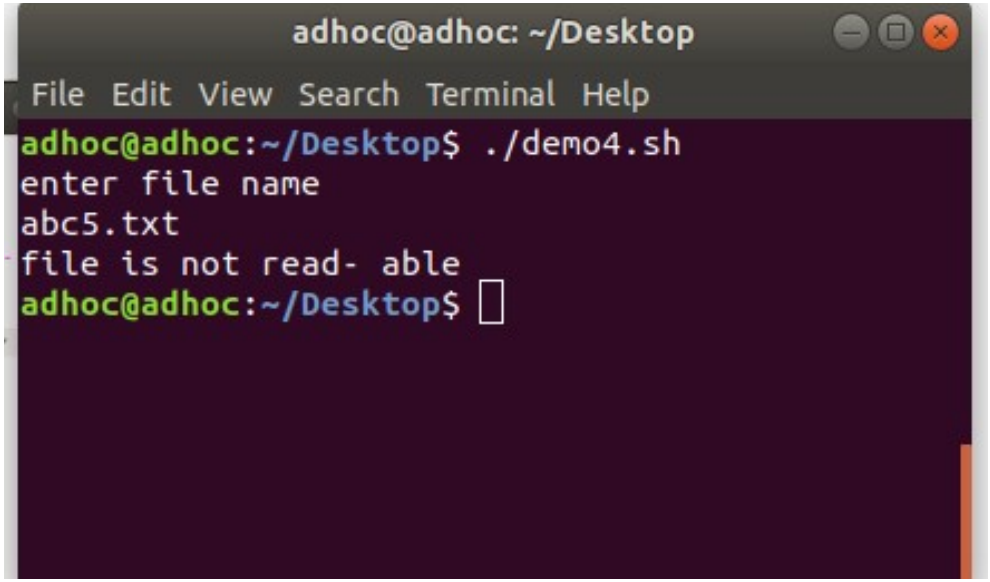
FILE TESTING

	Meaning
-d file	True if ‘file’ is a directory
-f file	True if ‘file’ is an ord. file
-r file	True if ‘file’ is readable
-w file	True if ‘file’ is writable
-x file	True if ‘file’ is executable
-e file	True if ‘file’ is exist
-s file	True if length of ‘file’ is nonzero

EXAMPLE: FILE TESTING

```
#!/bin/bash
echo "Enter a filename: "
read filename
if [ ! –r "$filename" ]
 then
   echo "File is not read-able"
 exit 1
fi
```


Open
dem...
~/Des...
Save
1 #! /bin/bash
2 echo "enter file name"
3 read filename
4 if [! -r "$filename"]
5 then
6 echo "file is not read- able"
7 exit 1
8 fi
Tab Width: 8
Ln 8, Col 3
OVR


adhoc@adhoc: ~/Desktop
File Edit View Search Terminal Help
adhoc@adhoc:~/Desktop$ ./demo4.sh
enter file name
abc5.txt
file is not read- able
adhoc@adhoc:~/Desktop$

File Handling (check exsit /not)

```
1 #! /bin/bash
2
3 echo -e "enter the name of file : \c"
4 # use \c for maintining cursor on cureent line
5 #use -e for not print \c on screen
6
7 read file_name
8 #use -e for ching file present or not
9 if [ -e $file_name ]
10 then
11   echo "$file_name found"
12 else
13   echo "$file_name not found"
14 fi
15
16 #-b block special file (audio, vedio etc)
17 #-c character sepcial file
```

```
adhoc@adhoc: ~/Desktop/Unix_shell/day3_shell
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell5.sh
enter the name of file : xyz
xyz not found
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ touch xyz
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell5.sh
enter the name of file : xyz
xyz found
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ gedit day3_shell5.sh
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$
```

EXAMPLE: FILE TESTING

```
#! /bin/bash

if [ $# -lt 1 ]; then
        echo "Usage: filetest filename"
        exit 1
fi
if [[ ! -f "$1" || ! -r "$1" || ! -w "$1" ]]
then
  echo "File $1 is not accessible"
  exit 1
fi
```

File Handling (Enter data in file)

day3_shell6.sh (~/Desktop/Unix_shell/day3_shell) - gedit

Open Save

```
#! /bin/bash

echo -e "enter the name of file : \c"
# use \c for maintining cursor on cureent line
#use -e for not print \c on screen

read file_name
#***********append file********
if [ -f $file_name ]
then
     if [ -w $file_name ] #check file have write condition
        then
        echo "type your text or for stop press ctr+d"
#ctr+d: come out from cat command
           cat >> $file_name
#cat > : overwritten file
#cat >> : append the file
      else
      echo "file does not have permission"
      fi
else
  echo "$file_name not fexist"
fi
```

abc (~/Desktop/Unix_shell/day3_shell) - gedit

Open Save

The file "/home/adhoc/Desktop/Unix_shell/day3_shell/abc" changed on disk. Reload Ignore

```
1 hi students
```

abc (~/Desktop/Unix_shell/day3_shell) - gedit

Open Save

```
1 hi students
2 this is shell script
```

En 4:10

adhoc@adhoc: ~/Desktop/Unix_shell/day3_shell

```
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell6.sh
bash: ./day3_shell6.sh: Permission denied
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ chmod +w day3_shell6.sh
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell6.sh
bash: ./day3_shell6.sh: Permission denied
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ chmod +x day3_shell6.sh
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell6.sh
enter the name of file : abc
abc not fexist
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ touch abd
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell6.sh
enter the name of file : abc
abc not fexist
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ touch abc
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell6.sh
enter the name of file : abc
type your text or for stop press ctr+d
hi students
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell6.sh
enter the name of file : abc
type your text or for stop press ctr+d
this is shell scriptadhoc@adhoc:~/Desktop/Unix_shell/day3_shell$
```

EXAMPLE: IF… STATEMENT

```
# The following THREE if-conditions produce the same result

* DOUBLE SQUARE BRACKETS
read -p "Do you want to continue?" reply
if [[ $reply = "y" ]]; then
   echo "You entered " $reply
fi

* SINGLE SQUARE BRACKETS
read -p "Do you want to continue?" reply
if [ $reply = "y" ]; then
   echo "You entered " $reply
fi
```

Let command

- let is a builtin command of the Bash shell which evaluates arithmetic expressions.
- let "varone = 1" "vartwo = varone++"; echo $varone, $vartwo

  Set varone to 1, set vartwo to the value of varone, then increment the value of varone by one.
  Output:

  2, 1

EXAMPLE: IF..ELIF... STATEMENT

```
#!/bin/bash
read -p "Enter Income Amount: " Income
read -p "Enter Expenses Amount: " Expense

let Net=$Income-$Expense

if [ "$Net" -eq "0" ]; then
   echo "Income and Expenses are equal -
  breakeven."
elif [ "$Net" -gt "0" ]; then
   echo "Profit of: " $Net
else
   echo "Loss of: " $Net
fi
```

THE CASE STATEMENT

- use the case statement for a decision that is based on multiple choices

Syntax:

```
case word in
   pattern1) command-list1
   ;;
   pattern2) command-list2
   ;;
   patternN) command-listN
   ;;
esac
```

Case

Case syntax :

```
Case expression in
        pattern1 )
                Statements ;;
        pattern2 )
                Statements ;;
esac
```

```
#! /bin/bash

#*** Even or odd ****
echo -e "enetr num and check even or odd :\c"
read num
r=$(( $num % 2 ))
case $r in
   0 )
     echo enter num $num is even ;;
   1 )
     echo enter num $num is odd ;;
    *)
      echo invalid ;;
esac

#**** Match string *****

echo -e "enetr your class  :\c"
read cl
case $cl in
     "Btech1st" )
           echo  B.tech 1st year ;;
     "Btech2nd" )
           echo B.tech 2nd year ;;
      *)
           echo invalid class input ;;
esac
```

```
adhoc@adhoc: ~/Desktop/Unix_shell/day3_shell
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell8.sh
enetr num and check even or odd :8
enter num 8 is even
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell8.sh
enetr num and check even or odd :7
enter num 7 is odd
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell8.sh
enetr num and check even or odd :10.5
./day3_shell8.sh: line 6: 10.5 % 2 : syntax error: invalid arithmetic
error token is ".5 % 2 ")
invalid
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ gedit day3_shell8.sh
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell8.sh
enetr your class  :Btech1st
B.tech 1st year
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell8.sh
enetr your class  :Btech2nd
B.tech 2nd year
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell8.sh
enetr your class  :Btech5th
invalid class input
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$
```

CASE PATTERN

- checked against word for match
- may also contain:

  *****

  ?

  […]

  [:class:]
- multiple patterns can be listed via:

  |

EXAMPLE 1: THE CASE STATEMENT

```
#!/bin/bash
echo "Enter Y to see all files including hidden files"
echo "Enter N to see all non-hidden files"
echo "Enter q to quit"

read -p "Enter your choice: " reply

case $reply in
  Y|YES) echo "Displaying all (really…) files"
         ls -a ;;
  N|NO)  echo "Display all non-hidden files..."
         ls ;;
  Q)     exit 0 ;;

  *) echo "Invalid choice!"; exit 1 ;;
esac
```

EXAMPLE 2: THE CASE STATEMENT

```
#!/bin/bash
ChildRate=3
AdultRate=10
SeniorRate=7
read -p "Enter your age: " age
case $age in
  [1-9]|[1][0-2])   # child, if age 12 and younger
     echo "your rate is" '$'"$ChildRate.00" ;;
  # adult, if age is between 13 and 59 inclusive
  [1][3-9]|[2-5][0-9])
     echo "your rate is" '$'"$AdultRate.00" ;;
  [6-9][0-9])       # senior, if age is 60+
     echo "your rate is" '$'"$SeniorRate.00" ;;
esac
```

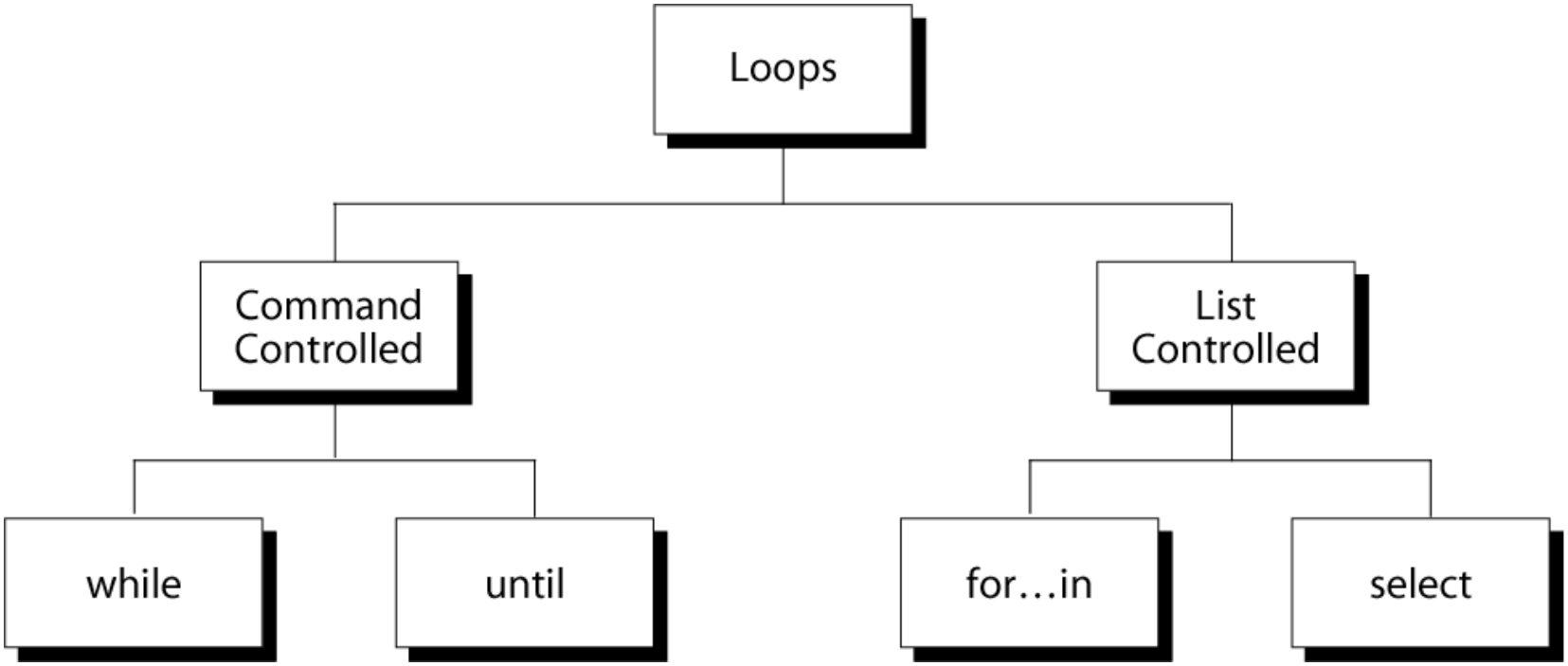
BASH PROGRAMMING: SO FAR

- Data structure
  - Variables
  - Numeric variables
  - Arrays
- User input
- Control structures
  - if-then-else
  - case

BASH PROGRAMMING: STILL TO COME

- Control structures
  - Repetition
    - do-while, repeat-until
    - for
    - select
- Functions
- Trapping signals

REPETITION CONSTRUCTS



THE WHILE LOOP

- Purpose:

  To execute commands in “command-list” as long as “expression” evaluates to true

Syntax:

```
while [ expression ]
do
   command-list
done
```

EXAMPLE: USING THE WHILE LOOP

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]
do
    echo The counter is $COUNTER
    let COUNTER=$COUNTER+1
done
```

Loops - While

Syntax:	

while [$a -le 10]
do
  Stat1
  Stat2..
done
	while (($a <=10))
	do
	  state1
	  state2
	done

*day3_shell9.sh (~/Desktop/Unix_sh

Open Sa

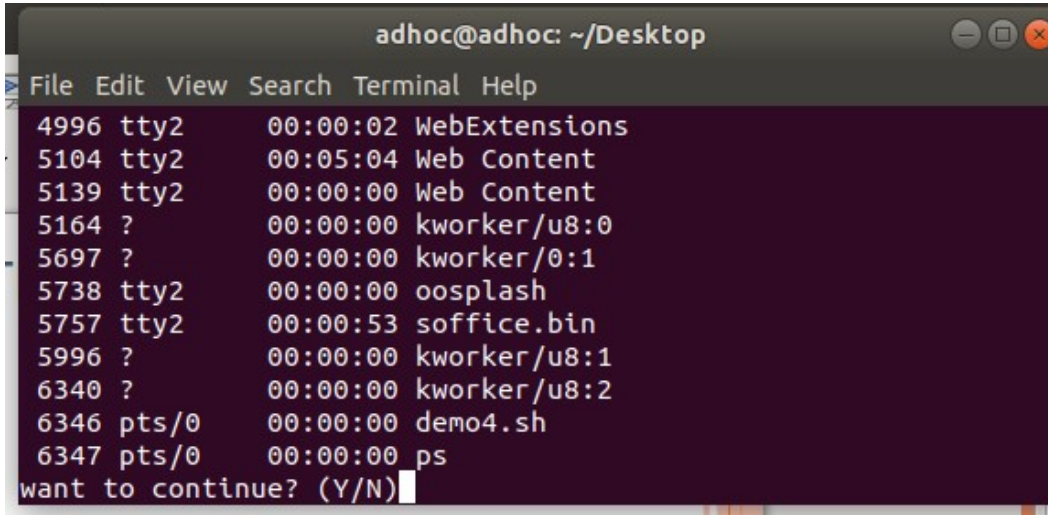
```
#! /bin/bash
n=2
#while [ $n -le 20 ]
while (( $n <= 3 ))
do
    echo $n
    n=$((n+1)) #(( n++ ))
     # for sleep
    sleep 1 #sleep for 1 sec
    #gnome command it open termnals
    gnome-terminal &
done
#***************
```

adhoc@adhoc: ~/Desktop/Unix_shell/day3_shell

```
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell9.sh
2
4
6
8
10
12
14
16
18
20
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$
```

EXAMPLE: USING THE WHILE LOOP

```
#!/bin/bash
#ps -A Select all processes.
Cont="Y"
while [ $Cont = "Y" ]; do
  ps -A
  read -p "want to continue? (Y/N)" reply
  Cont=`echo $reply | tr [:lower:] [:upper:]`
done
echo "done"
```



Read file using while loop

*day3_shell10.sh (~/Desktop/Unix_shel

Open Save

```
#! /bin/bash
#********* 1st *********
while read p
do
   echo $p
# < means redirecting the input to
this loop till file not end
done < day3_shell10.sh

#*******  2nd  ********
#using cat,day3 file work as input
to p
cat day3_shell10.sh | while read p
do
   echo $p
done
#*********** 3rd *******
# IFS internal filed separator
while IFS= read -r p
do
   echo $p
done < day3_shell10.sh
```

adhoc@adhoc: ~/Desktop/Unix_shell/day3_shell

```
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$ ./day3_shell10.sh
#! /bin/bash
#********* 1st day3_shell10.sh day3_shell.odp de
while read p
do
echo $p
# < means redirecting the input to this loop till file not end
done < day3_shell10.sh

#******* 2nd day3_shell10.sh day3_shell.odp de
#using cat,day3 file work as input to p
#cat day3_shell10.sh | while read p
#do
# echo $p
#done
#*********** 3rd day3_shell10.sh day3_shell.odp de
# IFS internal filed separator
#while IFS= read -r p
#do
# echo $p
#done < day3_shell10.sh
adhoc@adhoc:~/Desktop/Unix_shell/day3_shell$
```

EXAMPLE: USING THE WHILE LOOP

```
#!/bin/bash
# copies files from home- into the webserver- directory
# A new directory is created every hour

PICSDIR=/home/carol/pics
WEBDIR=/var/www/carol/webcam
while true; do
   DATE=`date +%Y%m%d`
   HOUR=`date +%H`
   mkdir $WEBDIR/"$DATE"
   while [ $HOUR -ne "00" ]; do
      DESTDIR=$WEBDIR/"$DATE"/"$HOUR"
      mkdir "$DESTDIR"
      mv $PICSDIR/*.jpg "$DESTDIR"/
      sleep 3600
      HOUR=`date +%H`
   done
done
```

THE UNTIL LOOP

- Purpose:

  To execute commands in “command-list” as long as “expression” evaluates to false

Syntax:

```
until [ expression ]
do
   command-list
done
```

EXAMPLE: USING THE UNTIL LOOP

```
#!/bin/bash

COUNTER=20
until [ $COUNTER -lt 10 ]
do
    echo $COUNTER
    let COUNTER-=1
done
```

EXAMPLE: USING THE UNTIL LOOP

```
#!/bin/bash

Stop="N"
until [ $Stop = "Y" ]; do
  ps -A
  read -p "want to stop? (Y/N)" reply
  Stop=`echo $reply | tr [:lower:] [:upper:]`
done
echo "done"
```

THE FOR LOOP

- Purpose:

  To execute commands as many times as the number of words in the “argument-list”

Syntax:

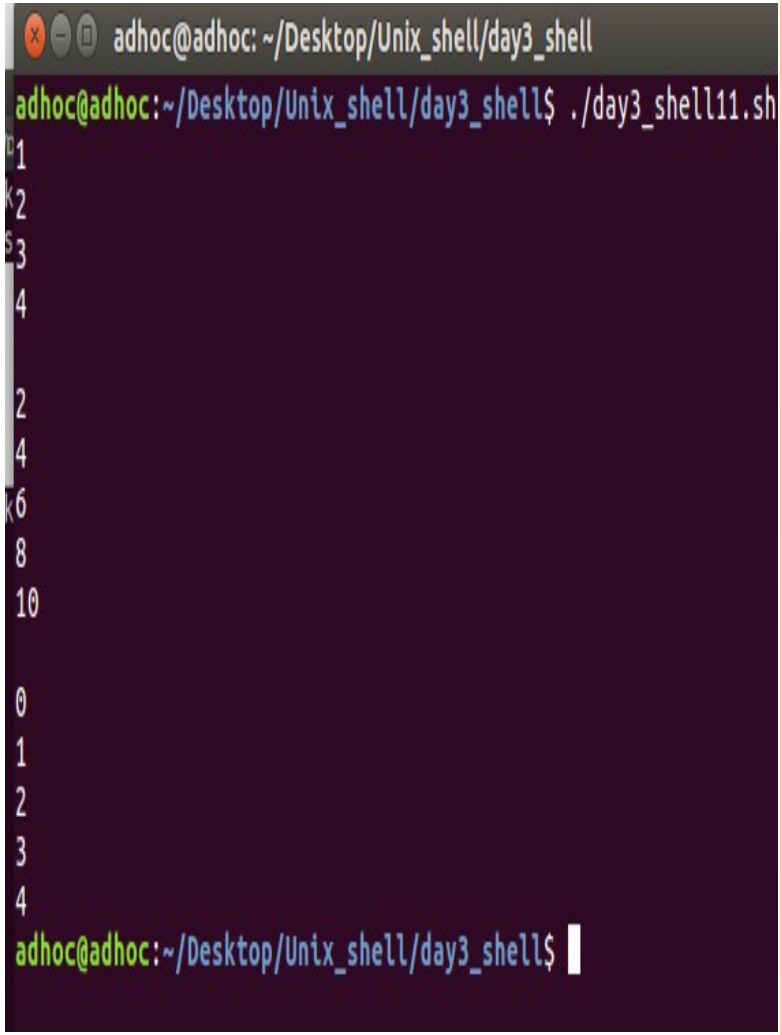
```
for variable in argument-list
do
   commands
done
```

EXAMPLE 1: THE FOR LOOP

```
#!/bin/bash

for i in 7 9 2 3 4 5
do
    echo $i
done
```

For Loop

```
#! /bin/bash

#****  1st simple til n   ********

for var in 1 2 3 4
do
  echo $var
done

#******* 2nd -- Range  ************
for var in {1..10}
for var in {2..10..2} # {start end incr}
do
  echo $var
done
#******* 3rd  -- like C  ************
echo
for (( i=0; i<5; i++ ))
do
  echo $i
done
```



EXAMPLE 2: USING THE FOR LOOP

```
#!/bin/bash
# compute the average weekly temperature

for num in 1 2 3 4 5 6 7
do
   read -p "Enter temp for day $num: " Temp
   let TempTotal=$TempTotal+$Temp
done

let AvgTemp=$TempTotal/7
echo "Average temperature: " $AvgTemp
```

SELECT COMMAND

- Constructs simple menu from word list
- Allows user to enter a number instead of a word
- User enters sequence number corresponding to the word

Syntax:

```
select WORD in LIST
do
   RESPECTIVE-COMMANDS
done
```


- Loops until end of input, i.e. ^d (or ^c)

SELECT EXAMPLE

```
#! /bin/bash
select var in alpha beta gamma
do
      echo $var
done
```

- Prints:

```
1) alpha
2) beta
3) gamma
#? 2
beta
#? 4
#? 1
alpha
```

SELECT DETAIL

- PS3 is select sub-prompt
- $REPLY is user input (the number)

```
#! /bin/bash
PS3="select entry or ^D: "
select var in alpha beta
do
      echo "$REPLY = $var"
done
```

Output:

```
select ...
1) alpha
2) beta
? 2
2 = beta
? 1
1 = alpha
```

SELECT EXAMPLE

Open ▾ demo4.sh ~/Desktop Save

```
1 #!/bin/bash
2 echo "script to make files private"
3 echo "Select file to protect:"
4
5 select FILENAME in *
6 do
7   echo "You picked $FILENAME ($REPLY)"
8   chmod go-rwx "$FILENAME"
9   echo "it is now private"
10 done
```

Tab Width: 8 ▾ Ln 10, Col 5 ▾ INS

adhoc@adhoc: ~/Desktop

File Edit View Search Terminal Help

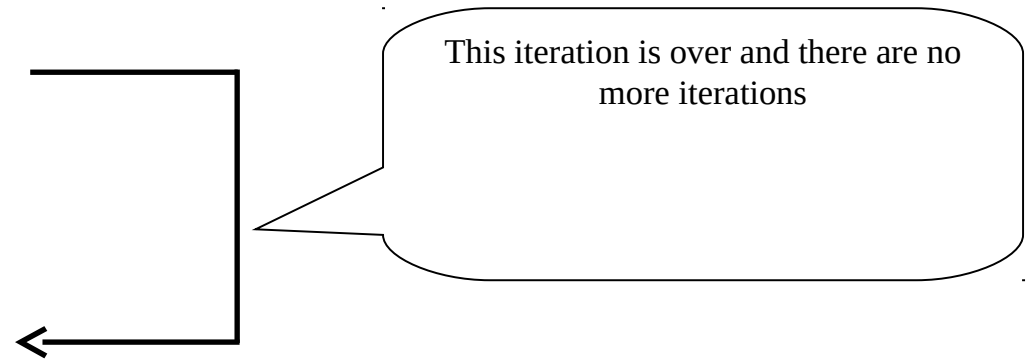
```
adhoc@adhoc:~/Desktop$ clear

adhoc@adhoc:~/Desktop$ ./demo4.sh
script to make files private
Select file to protect:
1) Babu                         5) saravan
2) demo4.sh                     6) t2~
3) FWP                          7) Ubantu_desktop
4) Room_electricity_bill.pdf
#? 4
You picked Room_electricity_bill.pdf (4)
it is now private
#? 2
You picked demo4.sh (2)
it is now private
#?
adhoc@adhoc:~/Desktop$
```

BREAK AND CONTINUE

- Interrupt for, while or until loop
- The break statement
  - transfer control to the statement AFTER the done statement
  - terminate execution of the loop
- The continue statement
  - transfer control to the statement TO the done statement
  - skip the test statements for the current iteration
  - continues execution of the loop

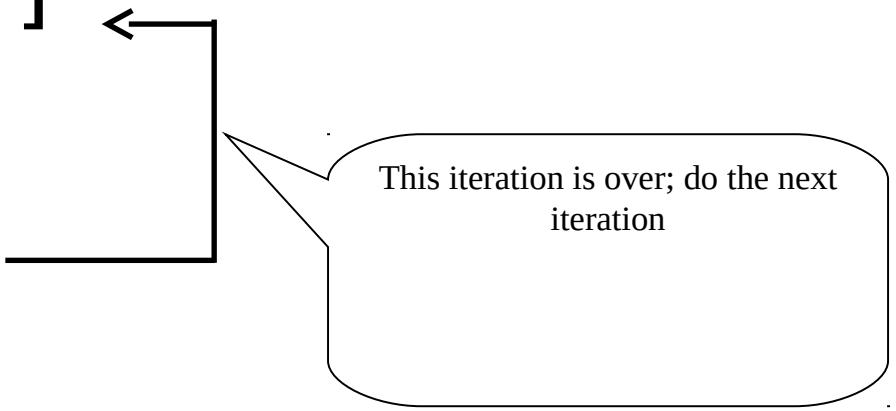
THE BREAK COMMAND

```
while [ condition ]
do
    cmd-1
    break
    cmd-n
done
echo "done"
```



THE CONTINUE COMMAND

```
while [ condition ]
do
     cmd-1
     continue
     cmd-n
done
echo "done"
```



EXAMPLE:

```
for index in 1 2 3 4 5 6 7 8 9 10
do
        if [ $index –le 3 ]; then
             echo "continue"
             continue
        fi
        echo $index
        if  [ $index –ge 8 ]; then
             echo "break"
             break
        fi
done
```

BASH SHELL PROGRAMMING

- Sequence
- Decision:
  - if-then-else
  - case
- Repetition
  - do-while, repeat-until
  - for
  - select

DONE !

- Functions
- Traps

still to come

SHELL FUNCTIONS

- A shell function is similar to a shell script
  - stores a series of commands for execution later
  - shell stores functions in memory
  - shell executes a shell function in the same shell that called it
- Where to define
  - In .profile
  - In your script
  - Or on the command line
- Remove a function
  - Use unset built-in

SHELL FUNCTIONS

- must be defined before they can be referenced
- usually placed at the beginning of the script

Syntax:

```
function-name () {
    statements
}
```

EXAMPLE: FUNCTION

```
#!/bin/bash

fun_demo () {
  # This is a simple function
  echo "This is a funky function."
  echo "Now exiting funky function."
}

# declaration must precede call:

fun_demo
```

EXAMPLE: FUNCTION

```
#!/bin/bash
fun_demo () { # A somewhat more complex
  function.
  JUST_A_SECOND=1
  let i=0
  REPEATS=30
  echo "And now the fun really begins."
  while [ $i -lt $REPEATS ]
  do
     echo "-------FUNCTIONS are fun_demo--------
  >"
     sleep $JUST_A_SECOND
     let i+=1
  done
}
fun_demo
```

FUNCTION PARAMETERS

- Need not be declared
- Arguments provided via function call are accessible inside function as $1, $2, $3, ...

$# reflects number of parameters

$0 still contains name of script
(not name of function)

EXAMPLE: FUNCTION WITH PARAMETER

```
#! /bin/sh
testfile() {
  if [ $# -gt 0 ]; then
     if [[ -f $1 && -r $1 ]]; then
        echo $1 is a readable file
     else
        echo $1 is not a readable file
     fi
  fi
}

testfile .
testfile funtest
```

EXAMPLE: FUNCTION WITH PARAMETERS

```
#! /bin/bash
checkfile() {
   for file
   do
      if [ -f "$file" ]; then
         echo "$file is a file"
      else
         if [ -d "$file" ]; then
            echo "$file is a directory"
         fi
      fi
   done
}
checkfile . funtest
```

LOCAL VARIABLES IN FUNCTIONS

- Variables defined within functions are global, i.e. their values are known throughout the entire shell program

- keyword “local” inside a function definition makes referenced variables “local” to that function

EXAMPLE: FUNCTION

```
#! /bin/bash

global="pretty good variable"

foo () {
        local inside="not so good variable"
        echo $global
        echo $inside
        global="better variable"
}

echo $global
foo
echo $global
echo $inside
```

HANDLING SIGNALS

- Unix allows you to send a signal to any process

- -1 = hangup **`kill -HUP 1234`**
- -2 = interrupt with ^C **`kill -2 1235`**
- no argument = terminate **`kill 1235`**
- -9 = kill **`kill -9 1236`**
  - -9 cannot be blocked

- list your processes with

  `ps -u userid`

SIGNALS ON LINUX

```
% kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

- ^C is 2 - SIGINT

HANDLING SIGNALS

- Default action for most signals is to end process
  - term: signal handler

- Bash allows to install custom signal handler

Syntax:

```
trap 'handler commands' signals
```

Example:

```
trap 'echo do not hangup'  1 2
```

EXAMPLE: TRAP HANGUP

```
#! /bin/bash
# kill -1 won’t kill this process
# kill -2 will

trap 'echo dont hang up' 1

while true
do
        echo "try to hang up"
        sleep 1
done
```

EXAMPLE: TRAP MULTIPLE SIGNALS

```
#! /bin/sh
# plain kill or kill -9 will kill this
trap 'echo 1' 1
trap 'echo 2' 2

while true; do
    echo -n .
    sleep 1
done
```

EXAMPLE: REMOVING TEMP FILES

```
#! /bin/bash
trap 'cleanup; exit' 2

cleanup () {
        /bin/rm -f /tmp/tempfile.$$.?
}

for i in 1 2 3 4 5 6 7 8
do
        echo "$i.iteration"
        touch /tmp/tempfile.$$.$i
        sleep 1
done
cleanup
```

RESTORING DEFAULT HANDLERS

- **`trap`** without a command list will remove a signal handler
- Use this to run a signal handler once only

```
 #! /bin/sh
trap 'justonce' 2
justonce() {
  echo "not yet"
  trap 2              # now reset it
  }

while true; do
   echo -n "."
   sleep 1
done
```

DEBUG SHELL PROGRAMS

- Debugging is troubleshooting errors that may occur during the execution of a program/script
- The following two commands can help you debug a bash shell script:
  - echo

    use explicit output statements to trace execution
  - set

DEBUGGING USING “SET”

- The “set” command is a shell built-in command
- has options to allow flow of execution
  - –v option prints each line as it is read
  - –x option displays the command and its arguments
  - –n checks for syntax errors
- options can turned on or off
  - To turn on the option: set -xv
  - To turn off the options: set +xv
- Options can also be set via she-bang line

```
#! /bin/bash -xv
```

SUMMARY: BASH SHELL PROGRAMMING

- Sequence
- Decision:
  - if-then-else
  - case
- Repetition
  - do-while, repeat-until
  - for
  - select
- Functions
- Traps

DONE !