

17/11/2021  
What is Scope of a variable? Difference between them with example  
Local variable → Those variables whose scope is within the function where it is declared.

Global variable → These variables can be accessed globally in the entire program.

### Local variable

- The local keyword is used to declare the local variable
- It is declared inside the function
- It doesn't provide data sharing
- These are stored on the stack
- Passing parameters is necessary
- Garbage value is stored if not initialized
- If modification or modified in one function that particular modification doesn't reflect in other function

### Global variables

- No keyword is used
- It can be declared anywhere in the program
- It provides data sharing.
- These variables are stored in the fixed memory location by the compiler.
- Parameters passing is not necessary
- Zero is stored by default if not initialized
- If modified in one function that modification will be visible in whole program.



Example for creating local variable.

```
function hello() {
```

```
    local x = "RVCE"
```

```
    echo "I am studying in x=$x"
```

```
}
```

```
hello
```

```
echo "I am studying in x=$x"
```

Output:

I am studying in RVCE

I am studying in x =

Example for creating a Global variable.

```
function LSS() {
```

```
    x = "Linux shell scripting"
```

```
    echo "full form of LSS is x=$x"
```

```
}
```

```
LSS
```

```
echo "full form of LSS is x=$x"
```

full form of LSS is x=Linux shell scripting  
full form of LSS is x=Linux shell scripting.

Q1) Explain all the programming constructs in shell programming with examples.

### Loop constructs

The loop constructs are for and while looping constructs allow a program to execute a command or sequence of commands several times.

### For loop

The for loop executes a sequence of commands once for each member of a list.

Syntax:-

```
for variable
  in a-list-of-values
do
  command-1
  command-2
  :
  :
  last-command
done
```

### Format of the for loop construct

For each iteration of the loop, the next member of the list is assigned to the variable given in the for clause. References to that variable may be made anywhere in the commands within the do clause.

It is easier to read a shell programming if the looping constructs are visually clear. Because the shell ignores spaces at the beginning of the lines, each section of a command can be indented as it was in the above format. Each do has to have a corresponding done



classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

at the end of the loop  
when the command have been executed for the last  
value in the list, the program will execute the next  
line below done. If there is no line the program  
will end

Create a program that will move files to another  
directory.

```
$ cat mv.file  
echo Please type in the directory path  
read path  
for file  
  in memo1 memo2 memo3  
do  
  mv $file $path/$file  
done  
$
```

### While loop

The other loop construct, the while loop, uses two  
groups of commands. It will continue executing the  
sequence of commands in the second group, the do...done  
as long as the final command in the first group, the  
while list, returns a status of meaning the  
statements after the do can be executed.

Syntax:-

while

Command - 1

:

last - command

do

Command - 1

:

last - command

done

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

```
$ cat file enter.name
while
    read x
do
    echo $x >> file
done
$
```

→ \$ cat enter.name  
 echo "please type in each person's name and then a  
 echo "please end the list of names with a CTRL-d" <return>  
 while read x  
 do  
 echo \$x >> xfile  
 done  
 echo xfile contains the following names:  
 cat xfile  
 \$

(After the `do` loop is completed, the program executes the command below the `done`)

Conditional Constructs : if and case

The `if` command tells the shell program to execute the then sequence of commands only if the final command in the `if` command list is successful. The `if` construct ends with the keyword `fi`.

Syntax:-

```
if
    command-1
    :
    last-command
then
```



Command-1

:

last-command

fi

\$ cat search

echo Type in the word and the filename  
read word file

if grep \$word \$file

then echo \$word is in \$file

fi

\$

→ if ... then ... else

The if ... then construction can also issue an alternate set of commands with else, when the if command sequence is false when the if command sequence is false. It has the general format:

Syntax:

if

Command-1

:

last-command

then

Command-1

:

last-command

else

Command-1

:

last-command

fi

\$ cat search

echo Type in the word and the filename.

read word file

if

grep \$word \$file >/dev/null

then

echo \$word is in \$file

else

echo \$word is NOT in \$file

fi

\$

→ Case --- esac

Syntax:-

case word

in

pattern - 1)

command - line - 1

-----

last - command - line

;;

pattern - 2)

command - line - 1

-----

last - command - line

;;

\*)

command 1

-----

last - command

};

esac



Example:-

```
$ cat set-term
echo if you have a TTY HH20 type in HH20
echo if you have a TTY SH10 type in SH10
echo if you have a TTY SH20 type in SH20
read term
```

```
case $term
```

```
in
```

```
    HH20)
```

```
        TERM=TY
```

```
;;
```

```
    SH10)
```

```
        TERM=TS
```

```
;;
```

```
    SH20)
```

```
        TERM=TT
```

```
;;
```

```
*)
```

```
echo not a correct terminal type
```

```
;;
```

```
esac
```

```
export TERM
```

```
echo end of program
```

```
$
```

Break → The break command unconditionally stops the execution of any loop in which it is encountered and goes to the next command after the done, fi or esac statement.

Continue → The command causes the program to go immediately to the next iteration of a while or for loop without executing the remaining command in loop.