GETTING STARTED

WITH

COMMAND LINE ARGUMENTS

➢ COMMAND LINE ARGUMENTS (OR) POSITIONAL PARAMETERS:

- The shell reserves some variable names for its use. $1 to $9 are nine shell variables, called positional parameters or command line arguments, which automatically collect the arguments known as command line.
- At the time of execution of shell script, if user passes any arguments known as command line arguments or positional parameters.

The positional parameter values are from $1 to $9:

$1 : first parameter value
$2 :Second parameter value
$3 :Third parameter value
-
-
-
$9 :9th parameter value

➢ THE SPECIAL PARAMETERS ARE:

$0	**:** Name of the program (command being executed)
$$	**:** PID of current shell
$?	**:** Exit status of the last executed command.
$!	**:** PID of last background process.
$_	**:** Current shell settings.
$#	**:** Total number of positional parameters
$*	**:** List of all shell arguments. Can't yield each argument separately.
$@	**:** Similar to $*, but yields each argument separately when enclosed with double quotes.

Consider the following statement, where pname is any executable shell script file and the remaining are the arguments.

$pname pro is to can as progress is to congress

Where: $0 would be assigned pname
$1 would be assigned 'pro'
$2 would be assigned 'is' and so on, till 'congress' which is assigned to $9.

1. **Write a program copying one file to another**

```
$program.sh <sourcefilename> <targetfilename>
cp $1 $2
cat $2
```

The statement cp $1 $2 is translated by the shell as cp file1 file2 , as $1 called the first argument and $2, the second. Hence file1 is copied to file2, and then cat $2 displays its contents.

2. **Why we reminded you every time to change the mode of a shell script before executing it:**

```
$program1.sh <filename>
chmod 744 $1
$1
```

3. **Example of special Positional Parameters:**

```
#!/bin/sh
#Example of positional parameters
IFS=","
echo "Displaying all animal names using \$@"
echo "$@"
echo
echo "Displaying all animal names using \$*"
echo "$*"

$./script.sh  Cat Dog Fox Monkey
```

4. **special Positional Parameters:**

```
#!/bin/bash
if [ $# -gt 0 ]
then
echo "Your command line contains $# arguments"
else
echo "Your command line contains no arguments"
fi
```

5. **Another Example of special Positional parameters:**

```
#!/bin/bash
if [ $# -lt 3 ]
then
echo "ERROR: minimum 3 paramerts required"
echo "Example: myprog.sh fname lname city"
else
  echo "Program Name is $0"
  echo "First Name is $1"
  echo "Last Name is $2"
  echo "City is $3"
fi
```

➢ SETTING VALUES OF POSITIONAL PARAMETERS:

We have compared the positional parameters with variables they are in essence quite different.

For example you can't assign values to $1, $2....etc. As we do to any other user defined variables or system variables

```
a=10; but $1=10
b=alpha; $2=alpha Simply not done.
```

How positional parameters are set up by the command line arguments. There is one more way to assign values to the positional parameters the **set** command.

Examples : $set friends come and go, but enemies accumulate

```
$1 :Friends
$2 :come
S3 :and ....so on

$set $1 $2 $3 $4 $5 $6 $7
$set Do you want credit or results
$set A smiling face is always beautiful
$echo $1 $2 $3 $4 $5 $6 $7
o/p : A smiling face is always beautiful
```

NOTE: On giving another set command, the old values of $1 $2...etc values are discarded and the new values get collected.

Let us now see another way of setting values in positional parameters:

```
$cat > lucky
Give luck a little time and
it will surely change
Ctrl+D
$set `cat lucky`
$echo $1 $2 $3 $4 $5
  Give luck a little time
```

Renames any file aaa to aaa.aa1, where aa1 is the user login name.

```
name=$1
set `who am i`
mv $name $name.$1
```

Displaying date in desired format:

```
$date
Fri Apr 19 11:30:45 IST 2016
To display the information in any order
Fri Apr 19 11:30:45 IST
$set `date`
$echo $1 $3 $2 $6
```

- **IFS:**

- The IFS is a special shell variable.
- You can change the value of IFS as per your requirments.
- The **Internal Field Separator (IFS)** that is used for word splitting after expansion and to split lines into words with the read builtin command.
- The default value is <space><tab><newline>.

Example with Internal Field saparator

```
#!/bin/bash
line="shell:scripting:is:fun."
IFS=:
set $line
echo $1 $2 $3 $4
```

Write a program user password file Revisited

```
#!/bin/sh
#user password file Revisited
echo "Enter a Username:\c"
```

```
read logname
line=`grep $logname /etc/passwd`
IFS=:
set $line
echo "Username:$1"
echo "User ID:$3"
echo "Group ID:$4"
echo "Comment Field:$5"
echo "Home Directory:$6"
echo "Login Shell:$7"
```

To find how many positional parameters were set either by set command or by command line arguments.

```
$vim myscript.sh
echo "Total number of files = $#"

$ myscript.sh file1 file2 file3
Total number of files=3
$ myscript.sh *
Total number of files=18
```

How come 18 positional parameters were reported to be set when there exist only 9-$1,$2,$3...$9 ? fact is, we can supply any number of arguments, but can access only nine of them at a time.

➢ USING SHIFT ON POSITIONAL PARAMETERS:

We have used the set command to set up 9 words. But we can use it for more.

$set you have the capacity to learn from mistakes. You will learn a lot in your life

```
$echo  $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11
You have the capacity to learn from mistakes. You You0 You1
```

Observe the last two words in the output. These occurred in the output because at a time we can access only 9 positional parameters. When we tried to refer to $10 it was interpreted by the shell as if you wanted to output the value of $1 and a 0. Hence, we got **You0** in the output. same as the story with $11. Does that mean the words following the ninth word have been lost?

To avoid this problem using shift
$shift 7
$echo $1 $2....$9
mistakes. You will learn a lot in your life.

Now where first 7 words are gone? They have been shifted out. Each word vacated a position for the one on its right with the first word getting lost in the bargain. This occurred 7 times, hence we find the last 9 words in $1 through $9. The first seven are lost forever.

```
$a=$1 $c=$3 $e=$5 $g=$7
$b=$2 $d=$4 $f=$6 $shift 4
$echo $a $b $c...$g $1 $2 $3...$9
(or)
$echo $*
```