Command-Line Arguments

Palani Karthikeyan

abpalanikarthik@gmail.com

Command-Line Arguments

- Like UNIX/Linux commands, shell scripts also accept arguments from the command line.
- In Shell script Command line arguments are useful to pass the inputs at run time.
- Mainly used in Function call and Array.
- The command-line arguments $1, $2, $3,...$9

Positional Parameters

- Arguments are passed from the command line into a shell program .
- In shell script all the command line arguments are called **positional parameters**($1 through to $9.)
- Each parameter corresponds to the position of the argument on the command line.

Cont…...

- The first argument is read by the shell into the parameter $1.
- The second argument into $2, and so on.
- After $9, the arguments must be enclosed in brackets.
- For example, ${10}, ${11}, ${12}.Some shells doesn't support this method.

Using Shift on Positional Parameters

- To access 10th and the above arguments i.e (from ${10} and above) **shift** command can be used.
- shift [n]
- Shift the positional parameters to the left by n.
- The positional parameters from n+1 ... $# are renamed to $1 ... $#-n.
- n must be a non-negative number less than or equal to $#.

Cont..

- If n is zero or greater than $#, the positional parameters are not changed.
- If n is not supplied, it is assumed to be 1.
- The return status is zero unless n is greater than $# or less than zero, non-zero otherwise.

Example using shift command

```
# shift n ( n -- is a number )

echo "Total no.of args:$#"

echo $1 $2 $3

shift 9

echo $1 $2 $3

Run: ./p1.sh  10 20 30 40 50 60 70 80 90 100 110 120 130 140 150

Total no.of args: 15

10 20 30

100 110 120 # after shifting
```

With out using shift command

```
echo ${10} ${11} # print 10th and 11th arguments

echo $3          # print 3rd argument

echo ${3}         # print 3rd argument

echo ${9} ${15}   # print 9th and 15th arguments
```

Example 1

```
echo $1 $2 $3 $4 $5
echo $4 $7 $1 $6  $2
```

```
 Run: ./P1.sh 10 20 T  “Hello Unix” /etc/passwd 10.45   78
              $1  $2  $3    $4           $5          $6    $7
________  |<--------  Command line Inputs/Arguments->|
```

- Result:-
- 10 20 T Hello Unix /etc/passwd
- Hello Unix 78 10 10.45 20

Example 2

- Using command Line arguments Calculate Sum of two numbers:

```
echo `expr $1 + $2`
(or)
 v1=$1 # initialize from command line to UDV
 v2=$2 # initialize from command line to UDV
echo `expr $v1 + $v2`
```

- Run as: `./p1.sh  10   20`

The following table shows a number of special variables that you can use in your shell scripts

Variable	**Description**
$0	The file name of the current script.
$n	These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).
$#	The number of arguments supplied to a script.
$*	All the arguments.
$@	All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
$?	The exit status of the last command executed.
$$	The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
$!	The process number of the last background command.

Example 3

```
echo $1 $2 $3 $5 $7

echo

echo $4 $1 $5 $3

echo

echo "TotalNo.of args:$#"

echo

echo "Exit from $0 file"
```

```
Run : ./p1.sh  10 20 30 40 50 60 70 80 90 100 110 /etc/passwd /var/log/auth.log “Hello Linux scripts”
```

Example 4

- # File Test using command line arguments

```
if [ -e $1 ];then
echo "File:$1 is available"
else
echo "File:$1 is not available"
fi
Run: ./p1.sh  /etc/passwd
```

- # File Test using command line arguments

```
Fname=$1
if [ -e $Fname ];then
echo "File:$Fname is available"
else
echo "File:$Fname is not available"
fi
Run: ./p1.sh  /etc/passwd
```

Example 5

```
echo $1 $2 $3
echo
echo $@
echo
echo $*
echo "Total No.of args:$#"
```

Example 6

```
# Compare two input files using command line arguments

if ! [ $# -eq 2 ];then

echo "cmp need two files.."

exit

fi
```

```
# Compare two input files using command line arguments

if cmp $1 $2 2>Result.log

then

echo "Both Contents are same.."

else

echo "Failed.."

fi
```

Example 7

```
#!/bin/bash
echo "Input: $@“
shift 3
echo "After shift: $@“
```

<u>Output of the above script will be:</u>

```
./myscript.sh one two three four five six
Input: one two three four five six
After shift: four five six
```

Special Parameters $* and $@

- "$*" special parameter takes the entire list as one argument
- "$@" special parameter takes the entire list and separates it into separate arguments.

Example 8
Loops using command-line arguments

for loop using command-line argument example:

```
for i in $@
do
    echo $i
done
```

<u>Output will be</u>:

```
./for1.sh a b c f
a
b
c
f
```

Example 9

While loop example:

```
while $1
do
    if [ "$1" == "Ram" ]
    then
        echo “Matched”
        break
    else
        echo “Not-Matched”
    fi
done
```

<u>Output will be displayed</u>

```
./whilex5.sh  ram
Not-Matched
./whilex5.sh  Ram
Matched
```

Example 10

Set command initialized all the arguments to command line arguments

```
set "100 200 300 400 500"

echo $1 $2 $3 $4 $5

echo "Total:$#"

ABC


set `date`

echo $1 $4

echo $#
```

Example- 11
File Test

```
if [ "$0" = "./$1" -o "$0" = "$1" ];then
echo "Usage: Input file and script file both are same"
exit 10
fi
if [ -z $1 ];then
#if [ $# -eq 0 ];then
echo "Usage:Enter some file name:"
exit 15
fi
if [ $# -ge 2 ];then
echo "Usage:Enter only single input file"
exit 20
fi
if [ -e $1 ];then
echo "File $1 is available"
exit 50
else
echo "File $1 is not available"
exit 1
fi
```

Note: use **basename** command

```
script_name=`basename $0`
echo "The name of this script is:$script_name"
```

Example 12

```
echo " enter the filename : $1“                  #run time input
fname=$1
if [ -e "$fname" ]                               #check for the availability of the file
then
echo " file is available"
fi
if [ -f "$fname" ]                               #starts checking the type.
then
echo " regular type file"
elif [ -d "$fname" ]
then
echo " directory file"
elif [ -l "$fname" ]
then
echo " link type file"
elif [ -c "$fname" ]
then
echo " character type file"
elif [ -b "$fname" ]
then
echo " block type file"
elif [ -p "$fname" ]
then
echo " pipe file"
else
echo " file cannot be determined"
fi
```

Example 13

```
echo "the ip address :`ping -c $1 $2`"
```

This is to display the ip ping status using command-line arguments.

<u>Output will be:</u>

```
ping -c 3 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.173 ms
64 bytes from 127.0.0.1: icmp_req=2 ttl=64 time=0.107 ms
64 bytes from 127.0.0.1: icmp_req=3 ttl=64 time=0.044 ms
--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.044/0.108/0.173/0.052 ms
```

Thank you