Bash Special Parameters Explained with 4 Example Shell Scripts

thegeekstuff.com/2010/05/bash-shell-special-parameters/

5/12/2010

by Sasikala on May 12, 2010

As part of our on-going bash tutorial series, we discussed about bash positional parameters in our previous article. In this article let us discuss about the bash special parameters with few practical shell script examples.

Some of the bash special parameters that we will discuss in this article are: $*, $@, $#, $$, $!, $?, $-, $_

To access the whole list of positional parameters, the two special parameters $* and $@ are available. Outside of double quotes, these two are equivalent: Both expand to the list of positional parameters starting with $1 (separated by spaces).

Within double quotes, however, they differ: $* within a pair of double quotes is equivalent to the list of positional parameters, separated by the first character of IFS "$1c$2c$3…".

$@ within a pair of double quotes is equivalent to the list of positional parameters, separated by unquoted spaces, i.e., "$1" "$2″.."$N".

Example 1: Use Bash $* and $@ to Expand Positional Parameters

This example shows the value available in $* and $@.

First, create the expan.sh as shown below.

```
$ cat expan.sh
#!/bin/bash

export IFS='-'

cnt=1

# Printing the data available in $*
echo "Values of \"\$*\":"
for arg in "$*"
do
  echo "Arg #$cnt= $arg"
  let "cnt+=1"
done

cnt=1

# Printing the data available in $@
echo "Values of \"\$@\":"
for arg in "$@"
do
  echo "Arg #$cnt= $arg"
  let "cnt+=1"
```

```
done
```

Next, execute the expan.sh as shown below to see how $* and $@ works.

```
$ ./expan.sh "This is" 2 3
Values of "$*":
Arg #1= This is-2-3
Values of "$@":
Arg #1= This is
Arg #2= 2
Arg #3= 3
```

- The above script exported the value of IFS (Internal Field Separator) with the ‘-‘.
- There are three parameter passed to the script expan.sh $1=”This is”,$2=”2″ and $3=”3″.
- When printing the each value of special parameter “$*”, it gives only one value which is the whole positional parameter delimited by IFS.
- Whereas “$@” gives you each parameter as a separate word.

Example 2: Use $# to Count Positional Parameters

$# is the special parameter in bash which gives you the number of positional parameter in decimal.

First, create the arithmetic.sh as shown below.

```
$ cat arithmetic.sh
#!/bin/bash

if [ $# -lt 2 ]
then
  echo "Usage: $0 arg1 arg2"
  exit
fi

echo -e  "\$1=$1"
echo -e "\$2=$2"

let add=$1+$2
let sub=$1-$2
let mul=$1*$2
let div=$1/$2

echo -e "Addition=$add\nSubtraction=$sub\nMultiplication=$mul\nDivision=$div\n"
```

If the number of positional parameters is less than 2, it will throw the usage information as shown below,

```
$ ./arithemetic.sh 10
Usage: ./arithemetic.sh arg1 arg2
```

Example 3: Process related Parameters – $$ and $!

The special parameter $$ will give the process ID of the shell. $! gives you the process id of the most recently executed background process.

The following script prints the process id of the shell and last execute background process ID.

```
$ cat proc.sh
#!/bin/bash

echo -e "Process ID=$$"

sleep 1000 &

echo -e "Background Process ID=$!"
```

Now, execute the above script, and check the process id which its printing.

```
$ ./proc.sh
Process ID=9502
Background Process ID=9503
$ ps
  PID TTY          TIME CMD
 5970 pts/1    00:00:00 bash
 9503 pts/1    00:00:00 sleep
 9504 pts/1    00:00:00 ps
$
```

Example 4: Other Bash Special Parameters – $?, $-, $_

- **$?** Gives the exit status of the most recently executed command.
- **$-** Options set using set builtin command
- **$_** Gives the last argument to the previous command. At the shell startup, it gives the absolute filename of the shell script being executed.

```
$ cat others.sh
#!/bin/bash

echo -e "$_"; ## Absolute name of the file which is being executed

/usr/local/bin/dbhome  # execute the command.
#check the exit status of dbhome
if [ "$?" -ne "0" ]; then
  echo "Sorry, Command execution failed !"
fi

echo -e "$-"; #Set options - hB

echo -e $_  # Last argument of the previous command.
```

In the above script, the last echo statement "echo -e $_" ($ underscore) also prints hB which is the value of last argument of the previous command. So $_ will give the value after expansion

```
$ ./others.sh
```

```
./others.sh
/home/oracle
Sorry, Command execution failed !
hB
hB
```

> Add your comment

If you enjoyed this article, you might also like..