

Chapter 8

Special Variables

Overview

- Special parameters
- Command-line arguments
- Getting options

Lesson: Special parameters

Variable	Purpose
\$	Process ID (PID) of currently running shell
!	Process ID (PID) of the most recent background command executed from the current shell
?	Exit status of previous command
-	Currently enabled shell options

```
grep -l $1 * > /tmp/cleaner$$
num_found=`wc -l < /tmp/cleaner$$`
echo Found $num_found files with pattern $1
for file in `cat /tmp/cleaner$$`
do
  echo $file
done
rm /tmp/cleaner$$
```

Lesson: Command-line arguments

- ▣ Command line input
- ▣ Special parameters for positional parameters
- ▣ The shift command
- ▣ The set command

Command line input

▣ Command line

```
$ sh_program arg1 arg2 ... argX
$0    $1    $2    ...  ${X}
```

▣ Example

```
$ cat color3
echo You are now running program: $0
echo The value of command line argument \#1 is: $1
echo The value of command line argument \#2 is: $2

$ ./color3 red green
You are now running program: ./color3
The value of command line argument #1 is: red
The value of command line argument #2 is: green
```

Special parameters for command-line arguments

Variable	Purpose
#	Number of command-line arguments
*	All command-line arguments starting from 1, all arguments as one (quoted) field.
@	All command-line arguments starting from 1, every argument as separate (quoted) field. Preserves multi-word arguments
0	Name of script

```
echo "The script name: $0"
echo "The total number of parameters: $#"
```

```
echo "All of the parameters as individual words: $*"
echo "All of the parameters as originally quoted: @$"
```

Shifting Parameters

Syntax:

```
shift [n]
```

- shift positional parameters by n places

Variable	Value String
user_input	String from stdin
2	argc
1	argb
0	prog.sh

Creating Positional Parameters

Syntax:

```
set word ...
```

```
$ set arga argb argc
```

Variable	Value String
user_input	String from stdin
3	argc
2	argb
1	arga
0	

Lesson: Getting options

- Processing Options and Arguments
- The getopt Command
- The OPTARG Variable
- The OPTIND Variable

Processing Options and Arguments

Variable	Value	String
#	5	
*	-a arga -b argb argc	
@	"-a" "arga" "-b"	"argb" "argc"
5	argc	
4	argb	
3	-b	
2	arga	
1	-a	
0	prog.sh	

The getopt Command

Syntax:

```
getopts optstring name [arg ...]
```

```
while getopts ab VAR
do
  case $VAR in
    a) command_a ;;
    b) command_b ;;
    esac
done
```

The getopt Command (Continued)

```
while getopts +ab VAR
do
  case $VAR in
    a) command_a ;;
    b) command_b ;;
    +b) command_plus_b ;;
    esac
done
```

The OPTARG Variable

The diagram illustrates how the `OPTARG` variable is populated. At the top, a terminal window shows the command `$ prog.sh -a -b -c`. Red arrows point from the options `-a`, `-b`, and `-c` to a code block below. The code block shows a `while` loop using `getopts` to parse these options. The `case` statement handles each option, and the `esac` block ends the loop. The `OPTARG` variable is implicitly set to the value of the option being processed.

```
while getopts :ab VAR
do
case $VAR in
a) command_a ;;
b) command_b ;;
\?) echo "Invalid option $OPTARG";;
esac
done
```

The OPTARG Variable (Continued)

This diagram continues the previous one, showing a more complex example. The terminal window shows `$ prog.sh -a argA -b -c`. Red arrows point from `-a`, `argA`, `-b`, and `-c` to the code block. The code block shows a `while` loop using `getopts` to parse these options. The `case` statement handles each option, and the `esac` block ends the loop. The `OPTARG` variable is implicitly set to the value of the option being processed.


```
while getopts :a:b VAR
do
case $VAR in
a) echo "option a, arg: $OPTARG" ;;
b) echo "option b" ;;
*) echo "Invalid option: $OPTARG";;
esac
done
```

The OPTIND Variable

The diagram illustrates how the `OPTIND` variable is used. At the top, a terminal window shows the command `$ prog.sh -ab -c argA argB argC`. A grey arrow points from the `-c` option to a code block below. The code block shows a `while` loop using `getopts` to parse these options. The `case` statement handles each option, and the `esac` block ends the loop. The `OPTIND` variable is used to track the current position in the argument list. The `shift` command is used to shift the remaining arguments to the left.


```
while getopts :abc VAR
do
case $VAR in
a) echo "option a" ;;
b) echo "option b" ;;
c) echo "option c" ;;
*) printf "Usage: %s: [-a] [-b] [-c] args\n" $0
exit 2 ;;
esac
done
shift $((OPTIND-1))
echo "Remaining arguments: $@"
```

Review Exercises



■ Complete the exercises from the Learning Guide

Topics for Review



- 1 Read the review topics
- 2 Think about what you learned in this Session in the context of your own work environment
- 3 Discuss your answers as a class
