Presenting data And Creating Functions

CHAPTER 5

Standard file descriptors

- **Linux Standard File Descriptors:-**

File Descri	**Abbreviation**	**Description**
0	STDIN	Standard input
1	STDOUT	Standard output
2	STDERR	Standard error

STDIN

```
$ cat < testfile
This is the first line.
This is the second line.
This is the third line.
$
```

STDOUT

```
$ ls -l test > test2

$ cat test2
-rw-rw-r-- 1 rich rich 53 2007-10-26 11:30 test

$
```

append data to a file

```
$ who >> test2


$ cat test2
-rw-rw-r-- 1 rich rich 53 2007-10-26 11:30 test
rich pts/0 2007-10-27 15:34 (192.168.1.2)


$
```

STDERR

- **Redirecting just errors**

```
$ ls -al badfile 2> test4
$ cat test4
ls: cannot access badfile: No such file or directory
$
```

- **Redirecting errors and data**

```
$ ls -al test test2 test3 badtest 2> test6 1> test7
$ cat test6
ls: cannot access test: No such file or directory
ls: cannot access badtest: No such file or directory
$ cat test7
-rw-rw-r-- 1 rich rich 158 2007-10-26 11:32 test2
-rw-rw-r-- 1 rich rich 0 2007-10-26 11:33 test3
$
```

Temporary redirections

```
# testing STDERR messages
echo "This is an error" >&2
echo "This is normal output"
$
$ ./test8 2> test9
This is normal output
$ cat test9
This is an error
$
```

Permanent redirection

- # redirecting output to different locations

```
exec 2>testerror
echo "This is the start of the script"
echo "now reidirecting all output to another location"
exec 1>testout
echo "This output should go to the testout file"
echo "but this should go to the testerror file" >&2
$ ./test11
This is the start of the script
now reidirecting all output to another location
$ cat testout
This output should go to the testout file
$ cat testerror
but this should go to the testerror file
```

1. Redirecting Input in Scripts

- # redirecting file input

```
exec 0< testfile
count=1
while read line
do
echo "Line #$count: $line"
cocount=$[ $count + 1 ]
done
$ ./test12
Line #1: This is the first line.
Line #2: This is the second line.
Line #3: This is the third line.
$
```

2. Redirecting file descriptors

- # storing STDOUT, then coming back to it

```
exec 3>&1
exec 1>test14out
echo "This should store in the output file"
echo "along with this line."
exec 1>&3
echo "Now things should be back to normal"
$ ./test14
Now things should be back to normal
$ cat test14out
This should store in the output file
along with this line.
$
```

3. Creating input file descriptors

- # redirecting input file descriptors

```
exec 6<&0
exec 0< testfile
count=1
while read line
do
echo "Line #$count: $line"
count=$[ $count + 1 ]
done
exec 0<&6
read -p "Are you done now? " answer
case $answer in
Y|y) echo "Goodbye";;
N|n) echo "Sorry, this is the end.";;
esac
```

3. Creating input file descriptors

```
$ ./test15
Line #1: This is the first line.
Line #2: This is the second line.
Line #3: This is the third line.
Are you done now? y
Goodbye
$
```

Creating Functions

Basic Script Functions

Reusing parts of code that perform specific tasks.

Creating a function

There are two formats you can use to create functions in bash shell scripts.

1. function name {

commands

}

2.

name() {

commands

}

Using functions

```
function func1 {
echo "This is an example of a function"
}
count=1
while [ $count -le 5 ]
do
func1
count=$[ $count + 1 ]
done
echo "This is the end of the loop"
```

Output

- $ ./test1
- This is an example of a function
- This is an example of a function
- This is an example of a function
- This is an example of a function
- This is an example of a function
- This is the end of the loop

Testing using duplicate function name

```
function func1 {
echo "This is the first definition of the function name"
}
func1
function func1 {
echo "This is a repeat of the same function name"
}
func1
echo "This is the end of the script"
```

OUTPUT

```
$ ./test3
This is the first definition of the function name
This is a repeat of the same function name
This is the end of the script
$
```

Returning a Value

1. The default exit status

```
func1() {
echo "trying to display a non-existent file"
ls -l badfile
}
echo "testing the function:"
func1
echo "The exit status is: $?"
$ ./test4
testing the function:
trying to display a non-existent file
ls: badfile: No such file or directory
The exit status is: 1
$
```

Returning a Value

2. Using the return command

```
function dbl {
read -p "Enter a value: " value
echo "doubling the value"
return $[ $value * 2 ]
}
dbl
echo "The new value is $?"
$ ./test5
Enter a value: 200
doubling the value
The new value is 1
```

Returning a Value

3. Using function output

```
function dbl {
read -p "Enter a value: " value
echo $[ $value * 2 ]
}
result=`dbl`
echo "The new value is $result"
$ ./test5b
```

Passing parameters to a function

- Use the special variable $#

```
function addem {
if [ $# -eq 1 ]
then
echo $1
Fi }
echo -n “Passing one numbers: "
value=`addem 10`
echo $value
$./test
Passing one numbers: -10
```

Array Variables and Functions

Passing arrays to functions

```
# array variable to function testit
function testit {
local newarray
newarray=(`echo "$@"`)
echo "The new array value is: ${newarray[*]}"
}
myarray=(1 2 3 4 5)
echo "The original array is ${myarray[*]}"
testit ${myarray[*]}
$ ./test10
The original array is 1 2 3 4 5
The new array value is: 1 2 3 4 5
$
```

Array Variables and Functions

```
# adding values in an array
function addarray {
local sum=0
local newarray
newarray=(`echo "$@"`)
for value in ${newarray[*]}
do
sum=$[ $sum + $value ]
done
echo $sum
}
myarray=(1 2 3 4 5)
echo "The original array is: ${myarray[*]}"
arg1=`echo ${myarray[*]}`
result=`addarray $arg1`
echo "The result is $result"
$ ./test11
The original array is: 1 2 3 4 5
The result is 15
```