

Operating System

**LAB-06**

**Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Roll No \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­\_**

**Marks Obtained \_\_\_\_\_\_\_\_\_\_\_\_**

**Signature\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

UNIX/LINUX Shell programming

Shell Substitution, Quoting Mechanisms, I/O Redirection, Man\_page Help.

**Objective: Understand and implement shell substitution, Quoting mechanisms, I/O redirection and Man\_page Help in Linux.**

## **Shell Substitution:**

When performing command substitution make sure that you are using the backquote, not the single quote character.

## **Example**

Command substitution is generally used to assign the output of a command to a variable. Each of the following examples demonstrate command substitution −

#!/bin/sh

DATE=`date`

echo "Date is $DATE"

USERS=`who | wc -l`

echo "Logged in user are $USERS"

UP=`date ; uptime`

echo "Uptime is $UP"

This will produce following result −

DateisThuJul203:59:57 MST 2009

Loggedin user are 1

UptimeisThuJul203:59:57 MST 2009

03:59:57 up 20 days,14:03,1 user, load avg:0.13,0.07,0.15

## **Quoting mechanisms:**

## **The Metacharacters**

Unix Shell provides various metacharacters which have special meaning while using them in any Shell Script and causes termination of a word unless quoted.

For example **?** matches with a single charater while listing files in a directory and an **\*** would match more than one characters. Here is a list of most of the shell special characters (also called metacharacters) −

\*?[]' " \ $ ; & ( ) | ^ <> new-line space tab

A character may be quoted (i.e., made to stand for itself) by preceding it with a \.

## **Example**

Following is the example which show how to print a **\*** or a **?** −

#!/bin/sh

echo Hello;Word

This would produce following result −

Hello

./test.sh: line 2:Word: command not found

shell returned 127

Now let us try using a quoted character −

#!/bin/sh

echo Hello\; Word

This would produce following result −

Hello;Word

The $ sign is one of the metacharacters, so it must be quoted to avoid special handling by the shell −

#!/bin/sh

echo "I have \$1200"

This would produce following result −

I have $1200

There are following four forms of quotings −

|  |  |
| --- | --- |
| **Quoting** | **Description** |
| **Single quote** | All special characters between these quotes lose their special meaning. |
| **Double quote** | Most special characters between these quotes lose their special meaning with these exceptions:   * $ * ` * \$ * \' * \" * \\ |
| **Backslash** | Any character immediately following the backslash loses its special meaning. |
| **Back Quote** | Anything in between back quotes would be treated as a command and would be executed. |

## **The Single Quotes**

Consider an echo command that contains many special shell characters −

echo <-$1500.\*\*>;(update?)[y|n]

Putting a backslash in front of each special character is tedious and makes the line difficult to read −

echo \<-\$1500.\\*\\*\>\; \(update\?\) \[y\|n\]

There is an easy way to quote a large group of characters. Put a single quote ( ') at the beginning and at the end of the string −

echo '<-$1500.\*\*>; (update?) [y|n]'

Any characters within single quotes are quoted just as if a backslash is in front of each character. So now this echo command displays properly.

If a single quote appears within a string to be output, you should not put the whole string within single quotes instead you whouldpreceed that using a backslash (\) as follows −

echo 'It\'s Shell Programming'

## **The Double Quotes**

Try to execute the following shell script. This shell script makes use of single quote −

VAR=IQRA

echo '$VAR owes <-$1500.\*\*>; [ as of (`date +%m/%d`) ]'

This would produce following result −

$VAR owes <-$1500.\*\*>;[as of (`date +%m/%d`)]

So this is not what you wanted to display. It is obvious that single quotes prevent variable substitution. If you want to substitute variable values and to make invert commas work as expected then you would need to put your commands in double quotes as follows −

VAR=IQRA

echo "$VAR owes <-\$1500.\*\*>; [ as of (`date +%m/%d`) ]"

Now this would produce following result −

IQRA owes <-$1500.\*\*>;[as of (07/02)]

Double quotes take away the special meaning of all characters except the following −

* $ for parameter substitution.
* Backquotes for command substitution.
* \$ to enable literal dollar signs.
* \` to enable literal backquotes.
* \" to enable embedded double quotes.
* \\ to enable embedded backslashes.
* All other \ characters are literal (not special).

Any characters within single quotes are quoted just as if a backslash is in front of each character. So now this echo command displays properly.

If a single quote appears within a string to be output, you should not put the whole string within single quotes instead you whouldpreceed that using a backslash (\) as follows −

echo 'It\'s Shell Programming'

## **The Back Quotes**

Putting any Shell command in between back quotes would execute the command

## **Syntax:**

Here is the simple syntax to put any Shell **command** in between back quotes −

## **Example**

var=`command`

Following would execute **date** command and produced result would be stored in DATA variable.

DATE=`date`

echo "Current Date: $DATE"

This would produce following result −

CurrentDate:ThuJul205:28:45 MST 2009

## **I/O Redirection**

Most Unix system commands take input from your terminal and send the resulting output back to your terminal. A command normally reads its input from a place called standard input, which happens to be your terminal by default. Similarly, a command normally writes its output to standard output, which is also your terminal by default.

## **Output Redirection**

The output from a command normally intended for standard output can be easily diverted to a file instead. This capability is known as output redirection:

If the notation > file is appended to any command that normally writes its output to standard output, the output of that command will be written to file instead of your terminal −

Check following **who** command which would redirect complete output of the command in users file.

$ who > users

Notice that no output appears at the terminal. This is because the output has been redirected from the default standard output device (the terminal) into the specified file. If you would check *users* file then it would have complete content −

$ cat users

oko tty01 Sep1207:30

ai tty15 Sep1213:32

ruth tty21 Sep1210:10

pat tty24 Sep1213:07

steve tty25 Sep1213:03

$

If a command has its output redirected to a file and the file already contains some data, that data will be lost. Consider this example −

$ echo line 1> users

$ cat users

line 1

$

You can use >> operator to append the output in an existing file as follows −

$ echo line 2>> users

$ cat users

line 1

line 2

$

## **Input Redirection**

Just as the output of a command can be redirected to a file, so can the input of a command be redirected from a file. As the greater-than character > is used for output redirection, the less-than character < is used to redirect the input of a command.

The commands that normally take their input from standard input can have their input redirected from a file in this manner. For example, to count the number of lines in the file *users* generated above, you can execute the command as follows –

$ wc-l users

2 users

$

Here it produces output 2 lines. You can count the number of lines in the file by redirecting the standard input of the wc command from the file *users* −

$ wc-l < users

2

$

Note that there is a difference in the output produced by the two forms of the wc command. In the first case, the name of the file users is listed with the line count; in the second case, it is not.

In the first case, wc knows that it is reading its input from the file users. In the second case, it only knows that it is reading its input from standard input so it does not display file name.

## **Redirection Commands**

Following is the complete list of commands which you can use for redirection −

|  |  |
| --- | --- |
| **Command** | **Description** |
| pgm> file | Output of pgm is redirected to file |
| pgm< file | Program pgm reads its input from file. |
| pgm>> file | Output of pgm is appended to file. |
| n > file | Output from stream with descriptor n redirected to file. |
| n >> file | Output from stream with descriptor n appended to file. |
| n >& m | Merge output from stream n with stream m. |
| n <& m | Merge input from stream n with stream m. |
| << tag | Standard input comes from here through next tag at start of line. |
| | | Takes output from one program, or process, and sends it to another. |

## **Man\_page Help**

All the Unix commands come with a number of optional and mandatory options. It is very common to forget complete syntax of these commands.

Because no one can possibly remember every Unix command and all its options, there has been online help available since Unix's earliest days.

Unix's version of help files are called **man pages**. If you know any command name but you do not know how to use it, then Man Pages are here to help you at every step.

## **Syntax**

Here is the simple command to get the detail of any Unix command while working with the system −

$man command

## **Example**

Now you imagine any command for which you want to get help. Assuming you want to know about **pwd** then you simply need to use the following command −

$man pwd

The above command would open a help for you which would give you complete information about **pwd** command. Try it yourself at your command prompt to get more detail on

You can get complete detail on **man** command itself using the following command −

$man man

## **Man Page Sections**

Man pages are generally divided into sections, which generally vary by the man page author's preference. Here are some of the more common sections −

|  |  |
| --- | --- |
| **Section** | **Description** |
| NAME | Name of the command |
| SYNOPSIS | General usage parameters of the command. |
| DESCRIPTION | Generally describes of the command and what it does |
| OPTIONS | Describes all the arguments or options to the command |
| SEE ALSO | Lists other commands that are directly related to the command in the man page or closely resembling its functionUniversityty. |
| BUGS | Explains any known issues or bugs that exist with the command or its output |
| EXAMPLES | Common usage examples that give the reader an idea of how the command can be used. |
| AUTHORS | The author of the man page/command. |