TATA CONSULTANCY SERVICES

Experience certainty.

Confidentiality Statement

Tata Code of Conduct

We, in our dealings, are self-regulated by a Code of Conduct as enshrined in the Tata Code of Conduct. We request your support in helping us adhere to the Code in letter and spirit. We request that any violation or potential violation of the Code by any person be promptly brought to the notice of the Local Ethics Counselor or the Principal Ethics Counselor or the CEO of TCS. All communication received in this regard will be treated and kept as confidential.

TATA CONSULTANCY SERVICES

Virtual ILP – Introduction To Shell

Content Manual Version 1.1

December 2014

(ILP Guwahati)

Table of Content

1. Wild Card Characters

Symbol used to replace or represent one or more characters. Wildcards or wild characters are either asterisk (*), which represent one or more characters or question mark (?), which represent a single character.

Wild card /Shorthand	Meaning	Examples	
*	Matches any string or group of characters.	**$ ls ***	will show all files
		$ ls a*	will show all files whose first name is starting with letter 'a'
		$ ls *.c	will show all files having extension .c
		$ ls ut*.c	Will show all files having extension .c but file name must begin with 'ut'.
?	Matches any single character.	**$ ls ?**	will show all files whose names are 1 character long
		$ ls fo?	will show all files whose names are 3 character long and file name begin with fo
[...]	Matches any one of the enclosed characters	**$ ls [abc]***	Will show all files beginning with letters a,b,c

Note: [..-..] A pair of characters separated by a minus sign denotes a range.

***Example*:**

```
$ ls /bin/[a-c]*
```

Will show all File name beginning with letter a,b or c.

Output:

```
 /bin/arch  /bin/awk  /bin/bsh  /bin/chmod  /bin/cp  /bin/as  /bin/basename  /bin/cat
/bin/chown  /bin/ash.static  /bin/bash  /bin/chgrp  /bin/consolechars  /bin/csh
```

2. Shell Quoting Mechanism

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.

Example:

? Matches with a single character 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:

```
#!/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
```

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.

2.1. 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 would precede that using a backslash (\) as follows:

```
echo 'It\'s Shell Programming'
```

2.2. The Double Quotes

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

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

Now this would produce following result:

```
ZARA 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.

2.3. Back Quotes: Command Substitution

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

Syntax:

var=`command`

Example:

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:

```
Current Date: Thu Jul  2 05:28:45 MST 2009
```