TATA CONSULTANCY SERVICES

Experience certainty.

TATA CONSULTANCY SERVICES
Experience certainty.

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

TATA CONSULTANCY SERVICES
Experience certainty.

Virtual ILP – Shell Programming

Content Manual Version 1.1

December 2014

(ILP Guwahati)

Table of Content

1. *String Handling*

- String handling with test command:

test str	Returns true	if str is not null
test –n str	Returns true	if length of str is greater than zero
test –z str	Returns true	if length of str is equal to zero

- String handling with expr command

The expr is quite handy for finding the length of a string and extracting a sub-string:

Length of the string:

```
$ str=”abcdefghijk” ;
$ n=`expr "$str" : ‘.*’` ;
$ echo $n
11
```

expr gave how many times any character (.*) occurs. This feature is very useful in validating data entry.

Extracting a sub-string:

```
$ str=”abcdefghijk” ;
$ expr “$str” : ‘……\(..\)’
gh
```

Note that there are 6 dots preceding the sequence \(..\). This advanced regular expression signifies that the first six characters of the string are to be ignored and extraction should start from the 7th character. Two dots inside \(..\) suggests that this extraction is limited to two characters only (backslashes override the usual interpretation of ‘()’).

Extracting string from 3rd character to end of the string:

```
$ str="abcdefghijk"
$ expr "$str" : '..\(.*\)'
cdefghijk
```

Location of first occurrence of a character “d” inside string:

```
$ str=”abcdefghijk” ;
$ expr "$str" : '[^d]*d‘
4
```

Location of last occurrence if a character inside string:

Below will give the last occurrence of character 'a' from string str.

```
$str=”abc def abc”
$expr "$str" : '[^u]*a'
9
```

2. *Command Line Arguments*

While running the shell script, we need to give some arguments to the shell script which can be used in shell script program.

Parameters are essentially used to create generalized shell scripts. The command name (first word on command line) is put into a variable called $0, the first argument of command (second word) is put into $1, etc.

The command-line arguments $1, $2, $3...$9 are also called positional parameters, with $0 pointing to the actual command/program/shell script and $1, $2, $3, ...$9 as the arguments to the command.

Consider below sample shell script “test.sh”

```
$cat test.sh

#!/bin/sh
```

```
echo “Program name is $0”
echo “First argument is $1”
echo “Second argument is $2”
echo “Number of arguments passed = $#”
echo “The arguments are $*”
```

Let’s execute the above shell script “test.sh”. Remember to change File access permission of script to execute

```
$ sh test.sh arg1 arg2 arg3 <Enter>

Program name is test
First argument is arg1
Second argument is arg2
Number of arguments passed = 3
The arguments are arg1 arg2 arg3
```

There’s another way to execute the script using **./**

```
$ ./test.sh arg1 arg2 arg3 <Enter>

Program name is test
First argument is arg1
Second argument is arg2
Number of arguments passed = 3
The arguments are arg1 arg2 arg3
```

User Input: read

To get input from the keyboard, you use the read command. The read command takes input from the keyboard and assigns it to a variable.

Example: read.sh

```
#!/bin/sh

echo –n “Enter some text  > “
Read text
echo “You entered: $text”
```

Note that “-n” given to the echo command causes it to keep the cursor on the same line; i.e., it does not output a carriage return at the end of the prompt.

Next, we invoke the read command with “text” as its argument. What this does is wait for the user ro type something followed by a carriage return (the Enter key) and then assign whatever was typed to the variable text.

Execution:

```
$sh read.sh
Enter some text > This is some text
You entered: This is some text
```

2.1. *Validation of Command Line Arguments*

Let's assume, we have one shell script which requires exact 2 arguments to execute. Shell script should throw proper error message in case user has not give exact 2 arguments.

Let's have a small shell script "test.sh" for the same.

```
#!/bin/sh
if [ $# -ne 2 ]
then
    echo " Not sufficient arguments, please give exact 2 arguments"
else
    echo "Continue with our task"
fi
```

```
$ ./test.sh
Not sufficient arguments, please give exact 2 arguments

$ sh test.sh abc
Not sufficient arguments, please give exact 2 arguments

$ ./test.sh abc def ghi
Not sufficient arguments, please give exact 2 arguments

$ ./test.sh abc def 2
Not sufficient arguments, please give exact 2 arguments

$ ./test.sh abc def
Continue with our task
```

TATA CONSULTANCY SERVICES
Experience certainty.