BASH SCRIPTING

Substitution

CONTENTS

Shell Expansions

Expansion is performed on the command line after it has been split into tokens. There are seven kinds of expansion performed:

- brace expansion
- tilde expansion
- parameter and variable expansion
- command substitution
- arithmetic expansion
- word splitting
- filename expansion

The order of expansions is: brace expansion; tilde expansion, parameter and variable expansion, arithmetic expansion, and command substitution (done in a left-to-right fashion); word splitting; and filename expansion.

Brace Expansion

Let's run a couple of examples to understand how parameter expansion works:

```
$ price=5
$ echo "$priceUSD"
$ echo "${price}USD"
```

As we can observe from the output, **we use *{..}* here as a disambiguation mechanism for delimiting tokens.**Adding double quotes around a variable tells Bash to treat it as a single word. Hence, the first *echo* statement returns an empty output since the variable *priceUSD* is not defined.However, **using *${price}* we were able to resolve the ambiguity**. Clearly, in the second *echo* statement, we could concatenate the string *USD* with the value of the *price* variable that was what we wanted to do.

Brace Expansion

Brace expansion is a mechanism by which arbitrary strings may be generated. This mechanism is similar to *filename expansion* , but the filenames generated need not exist. Patterns to be brace expanded take the form of an optional preamble, followed by either a series of comma-separated strings or a sequence expression between a pair of braces, followed by an optional postscript. The preamble is prefixed to each string contained within the braces, and the postscript is then appended to each resulting string, expanding left to right.

Brace expansions may be nested. The results of each expanded string are not sorted; left to right order is preserved. For example,

$ echo a{d,c,b}e

Output: ade ace abe

Tilde Expansion

If a word begins with an unquoted tilde character (‘~’), all of the characters up to the first unquoted slash (or all characters, if there is no unquoted slash) are considered a *tilde-prefix*. If none of the characters in the tilde-prefix are quoted, the characters in the tilde-prefix following the tilde are treated as a possible *login name*. If this login name is the null string, the tilde is replaced with the value of the HOME shell variable. If HOME is unset, the home directory of the user executing the shell is substituted instead. Otherwise, the tilde-prefix is replaced with the home directory associated with the specified login name.

Example 1. Current Users Home

Tilde(~) as a separate word, expands to $HOME if it is defined, if $HOME is not defined, then it expands with the home directory of the current user.

Now the value of the HOME variable is /home/oracle so cd ~ changed the current directory to the value of $HOME.

Example 1. Home directory of the given user

~username

Working Directories

Tilde with + and – are used for representing the working directories.

- ~+ expands to the value of the PWD variable which holds the current working directory.
- ~- expands to the value of OLDPWD variable, which holds the previous working directory. If OLDPWD is unset, ~- is not expanded.

Escape Substitution

Sr.No.	Escape & Description
1	**** backslash
2	**\a** alert (BEL)
3	**\b** backspace
4	**\c** suppress trailing newline
5	**\f** form feed
6	**\n** new line
6	**\n** new line
7	**\r** carriage return
8	**\t** horizontal tab
9	**\v** vertical tab

Command Substitution

Command substitution is the mechanism by which the shell performs a given set of commands and then substitutes their output in the place of the commands.

Syntax

The command substitution is performed when a command is given as –

`command` or $(command)

When performing the command substitution make sure that you use the backquote (`), not the single quote character.

Command Substitution

Command substitution is generally used to assign the output of a command to a variable. Each of the following examples demonstrates the 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"
```

Upon execution, you will receive the following result –

```
Date is Thu Jul  2 03:59:57 MST 2009
Logged in user are 1
Uptime is Thu Jul  2 03:59:57 MST 2009
03:59:57 up 20 days, 14:03,  1 user,  load avg: 0.13, 0.07,
0.15
```

Command Substitution

```
#!/bin/bash

# your code goes here

seq 2 2 20
```

```
#!/bin/bash

# your code goes here

echo $(seq 2 2 20)
```

In the above example the seq command can be replaced using command substitution to make it a new command. The () redirect the commands within to be executed in a new shell and the output is substituted.

Variable Expansion

Variables and command expansion

During the process of command substitution, the output of the command can be assigned to a variable, just like any other value.

Example:

In the below script we have assigned the result of the echo command to both the strings in variables, “varaiable1” and “variable2” respectively. Then we have used these variables in the echo command.

```
#!/bin/bash

variable1=$(echo 'Hyderabad' )

variable2=$(echo 'EPAM')

echo "$variable1 : $variable2"
```

Word Splitting

Split Strings

Use the `set` command to split strings based on spaces into separate variables. For example, split the strings in a variable called `myvar`, which says *"This is a test"*.

```
myvar="This is a test"

set -- $myvar

echo $1

echo $2

echo $3

echo $4
```

Length of a Word

```
#!/bin/bash
E_NO_ARGS=65
if [ $# -eq 0 ]  # Must have command-line args to demo script.
then
  echo "Please invoke this script with one or more command-line arguments."
  exit $E_NO_ARGS
fi
var01=${@}
echo "var01 = ${var01}"
echo "Length of var01 = ${#var01}"
# Now, let's try embedding a space.
var02="${var01} EFGH28ij"
echo "var02 = ${var02}"
echo "Length of var02 = ${#var02}"
echo "Number of command-line arguments passed to script = ${#@}"
echo "Number of command-line arguments passed to script = ${#*}"
exit 0
```