Shell
Functions and Substitution

Shell Functions

- Writing huge scripts can be unmanageable.
- To keep the things simple, we can write functions in scripts.

Shell Functions (cntd…)

```
#function hello world defined
fnHelloWorld()
{
        #print hello world
        echo "Hello World";
}

#calling hello world function
fnHelloWorld
```

Shell Functions (cntd…)

- Passing arguments to functions

```
fnArgumentFunction()
{
        echo "Argument 1: $1"
        echo "Argument 2: $2"
       #we can also use $#
}

fnArgumentFunction Hello World
```

Shell Functions (cntd…)

- Returning values from functions

```
#!/bin/sh
rv () {
  echo "Hi $1 $2 $3"
  return 100
}

rv abc xyz 123
ret=$?
echo "Return value is $ret"
```

Shell Functions (cntd…)

- Returning values from functions

```
# can you try??
fnReturnFunction()
{
        echo $1$2
}

value=$(fnReturnFunction "holly" "wood")
echo $value
```

Shell Functions (cntd…)

- Sample program to check if the passed argument is directory or not.

Shell Functions (cntd…)

```
#!/bin/bash
usage(){
   echo "Usage: $0 some_name"
   echo "$0 finds if some_name is a directory"
   exit 1
}

is_dir_exists(){
   local f="$1"
   [[ -d "$f" ]] && return 1 || return 0
}
```

Shell Functions (cntd...)

```
# call usage() function if some_name !given
[[ $# -eq 0 ]] && usage

#call is_dir_exists
if ( is_dir_exists "$1" )
then
 echo "Its a directory"
else
 echo "Its not a directory"
fi
```

Shell Functions (cntd…)

- Local variables in functions

>> use keyword local

Shell Functions (cntd…)

```
#!/bin/bash
# global x and y
x=200
y=100

math(){
  local x=$1
  local y=$2
  echo $(( $x + $y ))
}
```

Shell Functions (cntd...)

```
echo "x: $x and y: $y"

echo "Calling math() with x: $x and y: $y"
math 5 10

# x and y are not modified by math()
echo "x: $x and y: $y after calling math()"
```

Shell Functions (cntd…)

- Recursive function to find the factorial of a number

Shell Functions (cntd…)

- **Function In Background (…….)**

```
name(){
  echo "Do something"
  sleep 1
}

# put a function in the background
name &

# do something
```

Escape Sequences

- Used with echo “string”;
- echo **-e** “string” to enable escape sequences.
- **-E** option to disable interpretation of backslash escapes (default).
- **-n** option to disable insertion of new line.

Escape Sequences (cntd…)

Escape	Description
****	backslash
\a	alert (BEL)
\b	backspace
\c	suppress trailing newline
\f	form feed
\n	new line
\r	carriage return
\t	horizontal tab
\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.

Command Substitution (cntd…)

- The command substitution is performed when a command is given as:

  `command`

Command Substitution (cntd…)

Example:

```
#!/bin/bash

DATE=`date`

echo "Date is $DATE"

USERS=`who | wc -l`

echo "Logged in user are $USERS "
```

This will produce following result:

```
Date is Thu Jul  2 03:59:57 MST 2009
Logged in user are 1
```

Parameter Substitution

- Getting Up Default Shell Variables Value

If parameter not set, use defaultValue.

${parameter:-defaultValue}

var=${parameter:-defaultValue}

Parameter Substitution (cntd…)

- Getting Up Default Shell Variables Value

```
#!/bin/bash
dir_main="${1:-/home/phpcgi}"
echo "Setting main directory at ${dir_main}…"
# rest of the script …
```

Parameter Substitution (cntd…)

- Getting Up Default Shell Variables Value

./script.sh /divyak # <--- set dir_main at /divyak
./script.sh /divyak2 # <--- set dir_main at /divyak2
./script.sh # <--- set dir_main at /home/phpcgi (default)

Parameter Substitution (cntd…)

- **Setting Default Values**

The assignment (:=) operator is used to assign a value to the variable if it doesn’t already have one.

${var:=value}

var=${USER:=value}

Parameter Substitution (cntd…)

- **Setting Default Values**

echo $USER
divyak

echo ${USER:=foo}
divyak

Unset value for $USER:
unset USER

echo ${USER:=foo}
foo

Parameter Substitution (cntd…)

- **Display an Error Message If $VAR Not Passed**

If the variable is not defined or not passed, you can stop executing the Bash script with the following syntax:

${varName?Error varName is not defined}

${varName:?Error varName is not defined or is empty}

Parameter Substitution (cntd…)

- **Display an Error Message If $VAR Not Passed**

```
${1:?"mkjail: Missing operand"}
MESSAGE="Usage: mkjail.sh domainname IPv4"
### define error message

_domain=${2?"Error: ${MESSAGE}"}
### you can use $MESSAGE too
```

Parameter Substitution (cntd…)

- **Display an Error Message and Run Command**

```
#!/bin/bash
_file="$HOME/.input"
_message="Usage: first use the options correctly"

# Run another command (compact format)
_cmd="${2:? $_message $(cp $_file $HOME/.output)}"
```

Parameter Substitution (cntd…)

- **Find Variable Length**

${#variableName}

echo ${#variableName}

len=${#var}

Parameter Substitution (cntd…)

- **Remove Pattern (Front of $VAR)**

${var#Pattern}

${var##Pattern}

Parameter Substitution (cntd…)

- **Remove Pattern (Front of $VAR)**

```
f="/etc/resolv.conf"
echo ${f#/etc/}
```

Parameter Substitution (cntd…)

- **Remove Pattern (Front of $VAR)**

```
_version="420"
_url=http://a/b/c/dnstop-${_version}.tar.gz
echo "${_url#*/}"
##  /a/b/c/dnstop-420.tar.gz
echo "${_url##*/}"
dnstop-420.tar.gz
```

Parameter Substitution (cntd…)

- **Remove Pattern (Front of $VAR)**

Print the name of this source file??

Parameter Substitution (cntd…)

- **Remove Pattern (Front of $VAR)**

```
#!/bin/bash
_self="${0##*/}"
echo "$_self is called"
```