Advanced Unix Programming

Module 03

Raju Alluri
askraju @ spurthi.com

Advanced Unix Programming: Module 3

- Shells & Shell Programming
  - Environment Variables
  - Writing Simple Shell Programs (shell scripts)
  - Condition checks
  - Iterative commands
  - Writing Advanced shell scripts

Environment Variables

- Environment Variables are the variables used by shell to determine certain values
  - How to see them
    - `env | more`
    - `echo $<variable>` as in `echo $DISPLAY`
    - `env | grep <variable>`
  - Commonly used environment variables
    - DISPLAY, USER, PATH etc.
  - It is a convention to use capitalized names

Setting Environment Variables

- A Variable can be set by a very simple syntax "VAR=value"
- Once you set, you can "export" the variable using the syntax "export VAR"
  - Then only it becomes an "Environment Variable"
- Environment variables are inherited by all the processes spawned by the current process

The PATH Environment Variable

- The PATH variable indicates the list of directories from which executable files are to be fetched.
  - e.g. Most of the executables are taken for /usr/bin and /usr/sbin
- The sequence of directories in PATH controls the sequence of search

Writing Simple Shell Scripts

- Create a file with .sh extension

```
$ touch myscript.sh
```

- Give it execute permissions

```
$ chmod +x myscript.sh
```

- Edit the file to write the code (see next slide for example code)

```
$ vi myscript.sh
```

- After saving the file, execute it

```
$ ./myscript.sh
```

Syntax of a shell script

- Always starts with the directive #!/bin/sh to indicate that this is a shell script

```
#!/bin/sh
```

- Comment lines start with # character

```
# This is a comment line.
```

- Any command will look like as if you executed it in the command shell

```
echo "Hello World!"
```

- It is a good idea to end the program with an exit value (0 for success, non zero for error value)

```
exit 0
```

Example Shell Script

- Example Shell Script

```
#!/bin/sh
# This is a comment line.
echo “Hello World!”
exit 0
```

Exit Values

- Significance of exit values
  - Traditionally, exit value 0 indicates successful execution. Non zero values indicates some error condition
- How to see the exit value of a script

```
$ ./myscript.sh
$ echo $?
0
$
```

Using Input Arguments

- The $ character is used to access input arguments to the script
  - $0 denotes name of the executing script
  - $n denotes nth argument
  - $* denotes all input arguments
  - $n denotes number of input arguments
  - $$ denotes PID of the executing script

```
#!/bin/sh
echo "Hello World from $0"
exit 0
```

Traversing Input Arguments

- $1 points to first argument and $# gives the count of arguments
- After you use shift, $1 points to the next argument and $# will be decremented by 1, until it reaches 0

Condition Checks: if

- The if construct checks if a particular check is successful or not
  - Example: Check if the input arguments are given or not

```
if [ $# -gt 0 ]; then
    echo "Input arguments are given"
  else
     echo "No input arguments"
fi
```

- The elif keyword is used for a sequence of condition checks.

Iteration: while

- The while construct checks if a particular condition is valid or not, and executes the commands repeatedly as long as the condition is valid
  - Example: Print the date 5 times

```
MYVAR=0;
while [ $MYVAR -lt 5 ]; do
    /usr/bin/date
    MYVAR=`expr $MYVAR + 1`
done
```

Iteration: for

- The for loop executes for each value of the string that exists in a variable.
- Example:

```
for val in $LIST
do
   echo "$val is the value"
done
```

Multiple Conditions: case

- The case statement begins with `case` and ends with `esac`. Each condition match is evaluated one after another. * matches with any value.

```
case "$1" in
   "-h") echo "printing help"
      ...
      break;;
   ...
   *) echo "default"
esac
```

Executing other programs

- You can execute other programs/scripts by
  - Using Absolute path
  - Using relative path from current directory
  - Depending on PATH environment variable
    - Prefered in some special cases, but not recommended all the time
- Taking the output from a command
  - Keep it in an environment variable

```
DATE=`/usr/bin/date`
HOWMANY=`/usr/bin/who | /usr/bin/wc -l `
```

  - Redirect the output to a file and process the file later

Using /dev/null

- Sometimes, you need to execute a program, and need only the exit value, but not the output
  - In such case, redirect the output to a special file /dev/null
- For example, all the errors (standard error file descriptor) can be redirected to /dev/null, if the errors are not important

Modularity: Subroutines

- At the beginning of the script, write all the subroutines

```
mySubroutine() {
}
```

- At any point later in the script, call the subroutine, with arguments

```
mySubroutine hello world
```

- Subroutines extract the arguments just like the regular scripts would do ($1 etc.)
- Sobroutines can call other subroutines, provided they are already defined earlier in the script

Including code from other scripts

- If you need some code to be used by several scripts, you can place it in a common file and include it in all the scripts

```
. mycommoncode.sh
```

-

Checking multiple conditions

- Logical operations can be done on multiple conditions to derive complex condition checks.
  - !, &&, ||
    - Note: && and || evaluate the second argument only when needed.

```
if [ $HOLIDAY && $GOOD_MOVIE ]; then
```

-

File checks

- Several checks can be done on files/directories to test the existence, permissions etc.
  - -f checks if the file exists or not and if it is a regular file
  - -d checks if the file exists and is a directory
  - -r checks if the file exists and is readable
  - -w checks if the file exists and is writable
  - -x checks if the file exists and is executable

```
if [ $DATA && -f $OUTFILE ]; then
```

  - For a complete list of possible checks, see “man sh”

Shell Programming Summary

- You can use “sh -vx <script_name>” to run shell scripts in verbose mode
- Shell Programming is very simple and powerful feature
- Shell scripts can be run on any Unix system
- Shell scripts are interpreted by the shell