Lanka Education and Research Network

LEARN

29th Nov 2022 : Session 02

Deepthi Gunasekara / LEARN

What are shells in Linux?

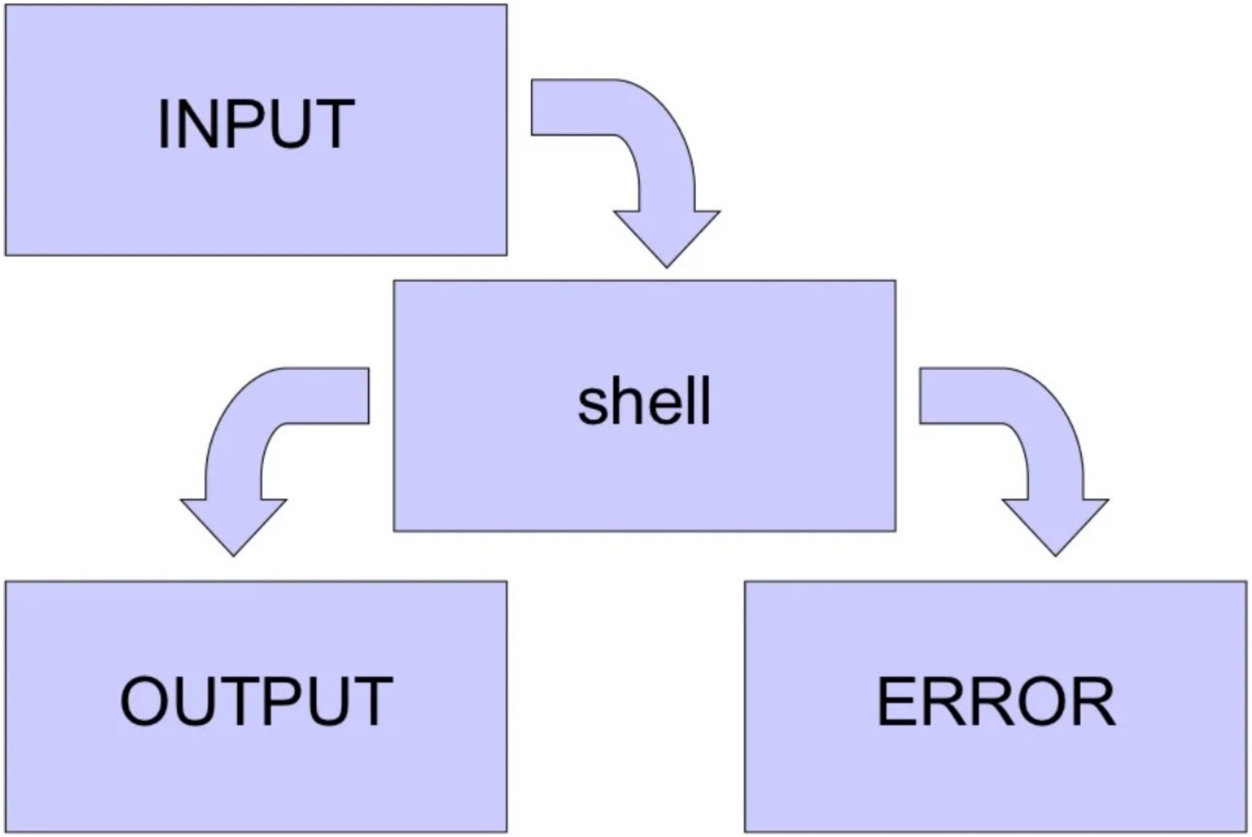
- Linux command line interpreter.
  - It provides an interface between the user and the kernel and executes programs called commands.

There are a few popular shells….

- Bourne shells
  - /bin/sh
  - /bin/bash – “Bourne-Again Shell”
- C Shell - /bin/csh
- Turbo C Shell - /bin/tcsh
- Korn Shell - /bin/ksh

Cont…

It provides an interface between the user and the kernel and executes programs called commands.



What is shell scripting?

❖ A text file + with instructions + Executable

❖ Typical operations performed by shell scripts include:
  - File manipulation
  - Program execution
  - Printing text

Using a shell script is most useful for

- Repetitive tasks that may be time consuming to execute by typing one line at a time.

The most used Linux commands

- ls - List
- alias - Replacement of a word by another string
- unalias - Remove entries from the current user's list of aliases
- pwd - Present working directory
- cd - Change directory
- cp - Copy
- rm - Remove (rmdir – remove directory)
- mv - Move/ rename

How do I create a shell scripting

Steps in creating a Shell Script:

1.Create a file using a text editor(any other editor).
2.Name script file with extension .sh.
3.Start the script with #! /bin/sh.
4.Write some code.
5.Save the script file as filename.sh.
6.For executing the script type bash filename.sh.

Things to remember:
- Always make sure it has executable status

```
chmod a+x filename
```

Let’s go through hands-on exercises to get familiar

Scripting…cont…

Here are some basic, but useful, tips for writing scripts

- Put in comments (to jog your memory when you write your paper months/years later)
- Put in some echo output commands so that you get some feedback on what your script is doing as it runs
- If your script starts doing something bad (or nothing at all) then use control-C to stop it
- It doesn't hurt to make a backup of key files before running a script, just in case

Scripting…cont…

We will now look systematically at the following shell and scripting concepts:

- Wildmasks
- Echo (printing to the screen/file)
- Variables
- Braces
- Command Line Arguments
- Single Quotes and Backslash
- Double Quotes
- Backquotes
- Pipes
- File Redirection

Scripting…cont…

Wildmask

- Can use wildmasks for matching patterns in *filenames*; expand into a list of *all* filename matches.

E.g.:

* matches any string
? matches any one character
[abgj] matches any one character in this range/list

```
$ ls
  sub1_t1.nii.gz sub1_t2.nii.gz sub2_t1.nii.gz sub2_t2.nii.gz
sub3_pd.nii.gz
$ ls sub*
  sub1_t1.nii.gz sub1_t2.nii.gz sub2_t1.nii.gz sub2_t2.nii.gz
sub3_pd.nii.gz
```

Scripting…cont…

```
$ ls sub1*
  sub1_t1.nii.gz sub1_t2.nii.gz

$ ls sub*t1*
  sub1_t1.nii.gz sub2_t1.nii.gz

$ ls sub[13]*
  sub1_t1.nii.gz sub1_t2.nii.gz sub3_pd.nii.gz

$ ls sub?_t2.nii.gz
  sub1_t2.nii.gz sub2_t2.nii.gz
```

Scripting…cont…

Echo

- echo prints the rest of the line to the screen (standard output).
- This is useful for providing output or updates in a script.
- Wildmasks (for filenames) and variables (values) are substituted in the argument *before* echo prints them.

•Examples:

```
$ echo Hello All!
  Hello All!
$ echo sub*t1*
  sub1_t1.nii.gz sub2_t1.nii.gz
$ echo j*k
  j*k
```

Scripting…cont…

Variables

- All shell variables store *strings*.
- A variable is set using > NAME=VALUE
- The variable name should start with a letter but can contain numbers and underscores

The value of a variable can be returned/used by adding a prefix $

Examples:

```
$ var1=im1.nii.gz
$ echo $var1
 im1.nii.gz
$ echo var1
 var1
$ ls $var1
 im1.nii.gz
```

Scripting…cont…

Braces

- Any name that starts with a letter can be used as a variable name.
  For instance: v, v1, v1_1, v_filename_4
- To add a string immediately after a variable name can be confusing.
  - The situation is solved by putting the variable name inside braces.

Examples:

```
$ v=im1
$ echo $v_new

$ echo ${v}_new
  im1_new
```

NB: all unused variables are blank by default (generate no error)

Scripting…cont…

Command Line arguments

- Inside a script the variables $1 $2 $3 *etc.* store the value of the command line arguments.

  e.g. if a script called reg_vol is executed as:
  $ reg_vol im1 3 abc
  then $1 = im1, $2 = 3, $3 = abc

- Other special variables are:
  - $0 = name of the script (often including the path)
  - $# = number of command line arguments given
  - $@ = all the command line arguments
    (i.e. $1 $2 $3 ...)
  - $$ = process ID number (unique to this process)

Scripting…cont…

Single quotes and backslash

- The shell substitutes variable names and wildmasks *before* executing the command - sometimes this is undesirable.
- To avoid substitutions either
  - prefix the special character (wildmask or $ sign) with a backslash: \
  - put the desired string in single quotes: '

- Examples:

```
$ var1=im1.nii.gz
$ echo $var1
 im1.nii.gz
$ echo \$var1
$var1
$ echo '$var1’
$var1
```

Scripting…cont…

Double quotes

- To group several strings together as one argument it is necessary to use double quotes: "

For example:

```
$ v=Hello World
$ echo $v
  Hello
$ v="Hello World"
$ echo $v
 Hello World
```

- ✓ NB: Variable substitutions are done inside double quotes but wildmasks are *not* expanded:

  e.g.   echo "*"   just prints a *

  but   echo "$v"   is the same as   echo $v

Scripting…cont…

Backquotes

- The (text) result of any command can be captured using backquotes: `
- This is very useful for setting variables.
  - Examples:

```
$ v=`ls sub[13]*`
$ echo $v
  sub1_t1.nii.gz sub1_t2.nii.gz sub3_pd.nii.gz
$ echo `fslval sub1_t1 pixdim2`
  4.0
```

✓ NB: the result is always treated as a single string, even if it contains spaces

Scripting…cont…

Pipe

- One of the most powerful features of the shell is the ability to chain commands together, each taking its input from the previous command's output.
- This is done using the pipe symbol: |
  - Examples (using the wordcount utility):

```
$ cat .bashrc | wc
  7   83   534
$ echo "Hello World" | wc
  1   2   12
```

- Technically this redirects standard output of one command to be the standard input of another.
- Error messages that are printed to standard error are *not* redirected with the pipe.

Scripting…cont…

File Redirection

- Command input can be taken from a file with: <
- Command output can be redirected to a file with: >
- Command output can be *appended* to a file with: >>
  - Examples:

```
$ echo "smoothing=10mm" > settings.txt
$ echo "No lowpass" >> settings.txt
$ cat settings.txt
smoothing=10mm
No lowpass
```

Scripting…cont…

For

- The for command executes a set of commands for every word in a list of words.
  - Syntax:

```
for VARIABLE in LIST OF VALUES ; do
    COMMANDS ;
done
```

- The commands are executed once for each entry in the words list.
- Each time the variable specified is equal to the current word.
  - Example:

```
for filename in im1 im2 im3 ; do
    echo $filename ${filename}_brain ;
done
```

Scripting…cont…

While

- The while command executes a set of commands as long as the condition is true.
  - Syntax:

```
while CONDITION ; do
    COMMANDS ;
done
```

- The condition is usually a test statement.
  - Example:

```
a=1
while [ $a -lt 4 ] ; do
    a=`echo $a + 1 | bc` ;
    echo $a;
done
```

T H A N K
Y O U