The Shell Interpretive Cycle

When you log on to a UNIX machine, you first see a prompt. This prompt remains there until you enter something. Even though the system looking idle, One UNIX command is in fact running at the terminal, it is the command which all time running, unless you log out. This command is Shell.

The following are the activities typically performed by the shell in its interpretive cycle.

- The shell issues the prompt and waits for you to enter a command.
- After a command is entered, the shell scans the command line for meta characters and expands abbreviations (like the * in **rm***) to recreate a simplified command.
- It then passes on the command line to the kernel for execution.
- The shell waits for the command to complete and normally can't do any work while the command is running.
- After command execution is complete, the prompt reappears and the shell returns to its waiting role to start the next cycle. You are now free to enter another command.

You can change this behavior and instruct the shell not to wait so you can fire one job after another without waiting for the previous one to complete.

Pattern Matching – The Wild Cards

Wild-Card	Matches
*	Any number of characters including none
?	A single character
[abc]	A single character either a,b or c
[!abc]	A single character that is not a,b or c
[a-d]	A single character that is within the ASCII range of characters a and d
[!a-d]	A single character that is not within the ASCII range of the characters a and d.

The * and ?

Examples

```
$ ls chap*

chap1       chap2       chap3       chap11      chap12      chap3
```

It will display the files start with chap and zero or more characters.

Vivekanand College of BCA

```
$ ls chap?

Chap1 Chap2 Chap3
```

It will display the files start with chap with single character.

```
$ ls .??*

.abc.txt       .t1.list
```

Display all hidden files with atleast two characters.

```
$ ls student*lst

Student.lst          student1.lst      student12.lst    student3lst
```

Above display the files which have zero or more character between student and lst.

Character Class

The character class comprises a set of characters enclosed by the rectangular brackets, [and]. It matches a single character an a,b,c or d.

Examples

```
$ ls chap[123]

chap1       chap2       chap3
```

The above example matches single character 1, 2 or 3.

```
$ ls chap[1-5]

chap1       chap2       chap3       chap4       chap5
```

Negation the Character Class

Vivekanand College of BCA

You can use the ! as the first character in the class to negate the class.

Examples

```
$ ls chap[!a-c]
```

Display the file which are not chapa, chapb or chapc.

Rounding up

Some of the wild-card characters have different meaning depending on where they are placed in the pattern.

$ ls *.c	List all files with extension c
$ cp f1 f1*	copy content of f1 to f1* file (* loses meaning here)
$ mv * ../tmp	move all files to tmp subdirectory of parent directory
$ lp file[0-1] [0-9]	print file file1 to file 19
$ cp ??? progs	copy all the three character files to progs directory

Escaping and Quoting

Escaping

Providing a \ (backslash) before the wild-card to remove (escape) its special meaning. This feature is known as escaping.

Examples

```
$ rm chap\*
```

It will remove file chap*. It will not remove file begin with chap, like chap1,chap2, chapa etc…

```
$ cat  > chap0[1-3]
Hi
[Ctrl –d]
```

Vivekanand College of BCA

The above example create file named chap0[1-3].

```
$ rm chap0[1-3]
```

It will remove file chap01, chap02 chap03.

If you want remove the file name chap0[1-3] then you use \ to remove the special meaning of [].

```
$ rm chap0[1-3\]

$
```

Escaping the Space

To remove the special meaning of space you escape it with \. For example to remove My Document.doc file, which has a space embedded

```
$ rm My\ Document.doc
```

Escaping the \ Itself

Sometime it is need to escape the \ itself, to remove its special meaning.

Examples

```
$ echo \\

\
```

Example 2:

```
$ echo -e "The newline character is \\n"

The newline character is \n
```

Quoting

The another way of remove meaning of special character (wild card character) is to enclosed it in single quotes. If we enclosed in single quotes, the meaning of special character are turned off.

Vivekanand College of BCA

Examples:

```
$ rm 'chap*'
```

It will remove the file named chap*

```
$ rm 'My Document.doc'
```

It will remove file name My Document.doc

Redirection : The three standard files

When a user logs in, the shell associate three files (streams) with the terminal- two for the display and one for the keyboard.

- Standard Input : The file (or stream) representing input, which is connected to the keyboard (default)
- Standard Output: The file (or stream) representing output, which is connected to the display
- Standard Error : The file (or stream) representing error messages that emanate from the command or shell. That is also connected to display.

Every command that uses streams will always find these files open and available. The files closed when the command completes execution.

Standard Input

There are three input sources

- The keyboard, the default source
- A file using redirection with the < symbol (< filename). It means take input from file.
- Another program using pipeline. It means output of first command will become input for second command using pipe.

Input from keyboard

Vivekanand College of BCA

The file representing input by default connected to the keyboard. When we use wc command without filename and we do not use < or | in the command line, wc get its input from the default source(keyboard).

```
$ wc
Hi
Good Bye
[Ctrl-d]
2       3       11
```

Input using another file

```
wc < sample.txt
3       14      100
```

In above command standard input source is not keyboard, but the < symbol, it makes the standard input source to file on disk.

Standard input using piping mechanism

```
$ who | wc
```

In above command output of who command become the standard input for wc command as a temporary file.

Standard Output

All commands displaying output on the terminal actually write to the standard output file as a stream of characters and not directly to the terminal.

This output file may connect to

- The terminal, the default destination

Vivekanand College of BCA

- A file using the redirection symbols > and >> (it means the output will store in file rather then display on terminal)
- As input to another program using pipeline (Output of first command become input of next command using piping mechanism)

The terminal, the default destination

```
$ cat sample.txt
Hi
Good bye
```

In above command default output source is terminal so sample.txt display on terminal.

A file using redirection symbol

```
$ cat sample.txt > newfile.txt
```

In above command the output of cat command is not display on terminal but is redirected to newfile.txt (store in newfile.txt)

As input to another program using pipline

```
$ cat sample.txt | wc –c
```

In the above command the output of cat command become input of wc command.

Standard Error

The three standard files is represented by a number, called a file descriptive. A file is opened by referring to its pathname, but subsequent read and write operations identify the file by this file descriptor. The kernel maintains a table of file descriptors for every process running in the system. The first three slots are generally, allocated to the three standard streams:

0 – Standard input

1 – Standard output

2 – Standard error

Vivekanand College of BCA

These descriptors are implicitly prefixed to the redirection symbols. For example, > and 1> mean the same thing to the shell, while < and 0< also are identical. If you opens a file, in all probability, the file will be allocated to descriptor 3.

When you enter an incorrect command or try to open a nonexistent file, certain error messages show up on the screen. This is the standard error stream whose default destination is the terminal.

Examples:

To open a nonexistent file with cat command.

```
$ cat f1

cat: cannot open f1
```

cat fails to open the file and writes to standard error. You can redirect this stream to a file.

It is not possible with following command using > or >>.

```
$ cat f1 > errorfile
```

The error will not store in the errorfile file.

Redirecting standard error requires the use of the 2> symbols.

```
$ cat f1  2>errorfile

$ cat errorfile

cat: cannot open f1
```

you can also append other similar error by appending standard output.

```
$ cat f2  2>>errorfile
```

Vivekanand College of BCA

/dev/null and /dev/tty (two special files)

/dev/null

/dev/null is a special file that simply accepts any stream without growing in size. It is used when you do not want to see the output of any command. Simply redirect the output of that command into /dev/null file.

```
$ cmp f1 f2 > /dev/null

$ cat /dev/null

$ _                                                    size is always zero
```

The size of /dev/null file is always zero. /dev/null simply discard all output written to in it, whether you direct or append output to this file, its size always remain zero. This facility is useful in redirecting error messages away from the terminal so they don't appear on the screen. /dev/null is actually a pseudo-device because, unlike all other device files, it is not associated with any physical device.

/dev/tty

The second special file in the UNIX system is the one indicating one's terminal - /dev/tty. You can redirect some statements to this file.

For example Amit is currently using terminal /dev/pts/1. Sumit is working on terminal /dev/pts/2.

So if Amit issues the command

who > /dev/tty

The list of current user is sent to the terminal he is currently using /dev/pts/1. Similarly Sumit can use an identical command to see the output on his terminal /dev/pts/2. Like /dev/null, /dev/tty can be accessed independently by several users without conflict.

Vivekanand College of BCA

Pipes

Pipe is a special operator in UNIX, which pass the output of one command to another for further processing. The standard output of the one command on the left of the pipe becomes standard input of the command to the right of a pipe.

```
$ cat f1
Hi all
Hello
$ cat f1 | wc
       2       3       13
```

The output of cat f1 is not sent to terminal, but it is sent to wc command as standard input by pipe. so now wc command will count the character, word and line of f1 file.

Creating a tee (tee)

tee is a external command and not a feature of the shell. It saves one copy in a file and writes the other to standard output (terminal).

```
$ who | tee users.txt
amit        pts/3        Jul   7  08:20   (pc.heavens.com)
sumit       pts/5        Jul   7  09:12   (mercury.heavens.com)
```

The tee command displays the output of who command on the terminal and save this output in a file as well.

Command Substitution

Pipe enables a command to obtain its standard input from the standard output of another command.

The shell enables one or more command arguments to be obtained from the standard output of another command. This feature is called command substitution.

Vivekanand College of BCA

For example you need to embed date in another statement.

```
$ echo The date today is `date`
```

When scanning the command line, the ` (backquote or backtick) is the another meta character that the shell looks for. The shell executes the command enclosed in backquote and replaces the enclosed command line with the output of the command.

So the output will be

```
The date today is Sat Aug 8 13:01:40 IST 2014
```

```
$ echo "There are `ls | wc –l` files in the current directory"

There are 6 files in the current directory

$
```

Command substitution is enabled when backquotes are used within double quotes. If you use single quotes, it is not.

```
$ echo 'There are `ls | wc –l` files in the current directory'

There are `ls | wc –l` files in the current directory
```

Shell Variables

The shell supports variables that are useful both in the command line and shell scripts.

A variable assignment is of the form variable=value (no space around =), but its evaluation requires the $ as prefix to the variable name.

```
$ count=5

$ echo $count

5
```

Vivekanand College of BCA

A Variable name begin with a letter but can contain numerals and the _ as the other characters. Names are case-sensitive. All shell variable are initialized to null strings by default. While explicit assignment of null strings with x="" or x='' is possible. You can also use this as a short hand:

```
$ x=
```

This will assigned null value to x.

A variable can be removed with unset and protected from reassignment by readonly. Both are shell internal commands.

$ unset x	x is now undefined
$ readonly x	x can't be reassigned

Vivekanand College of BCA