THE SHELL

Introduction

Shell acts as both a command interpreter as well as a programming facility.

The shell and its interpretive cycle

The shell sits between you and the operating system, acting as a command interpreter. It reads your terminal input and translates the commands into actions taken by the system. The shell is analogous to ***command.com*** in DOS. When you log into the system you are given a default shell. When the shell starts up it reads its startup files and may set environment variables, command search paths, and command aliases, and executes any commands specified in these files. The original shell was the Bourne shell, sh. Every Unix platform will either have the Bourne shell, or a Bourne compatible shell available.

Numerous other shells are available. Some of the more well known of these may be on your Unix system: the Korn shell, ksh, by David Korn, C shell, csh, by Bill Joy and the Bourne Again SHell, bash, from the Free Software Foundations GNU project, both based on sh, the T-C shell, tcsh, and the extended C shell, cshe, both based on csh.

Even though the shell appears not to be doing anything meaningful when there is no activity at the terminal, it swings into action the moment you key in something.

The following activities are 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 line.
- 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 the command execution is complete, the prompt reappears and the shell returns to its waiting role to start the next cycle. You are free to enter another command.

Pattern Matching – The Wild-Cards

A pattern is framed using ordinary characters and a meta character (like *) using well- defined rules. The pattern can then be used as an argument to the command, and the shell will expand it suitably before the command is executed.

The meta characters that are used to construct the generalized pattern for matching filenames belong to a category called wild-cards. The following table lists them:

Wild-card	Matches
*	Any number of characters including none

?	A single character
[ijk]	A single character – either an i, j or k
[x-z]	A single character that is within the ASCII range of characters x and z
[!ijk]	A single character that is not an i, j or k (Not in C shell)
[!x-z]	A single character that is not within the ASCII range of the characters x and z (Not in C Shell)
{pat1,pat2...}	Pat1, pat2, etc. (Not in Bourne shell)

Examples:

To list all files that begin with **chap**, use, **$ls chap***

To list all files whose filenames are six character long and start with chap, use , **$ls chap??**

Note: Both * and ? operate with some restrictions. for example, the * doesn't match all files beginning with a . (dot) or the / of a pathname. If you wish to list all hidden filenames in your directory having at least three characters after the dot, the dot must be matched explicitly.

$ ls .???*

However, if the filename contains a dot anywhere but at the beginning, it need not be matched explicitly.

Similarly, these characters don't match the / in a pathname. So, you cannot use, **$cd /usr?local** to change to **/usr/local**.

The character class

You can frame more restrictive patterns with the character class. The character class comprises a set of characters enclosed by the rectangular brackets, [and], but it matches a single character in the class. The pattern [abd] is character class, and it matches a single character – an a,b or d.

Examples:

$ls chap0[124] - Matches chap01, chap02, chap04 and lists if found.

$ls chap[x-z] - Matches chapx, chapy, chapz and lists if found.

You can negate a character class to reverse matching criteria. For example,

- To match all filenames with a single-character extension but not the .c ot .o files, use ***.[!co]**

- To match all filenames that don't begin with an alphabetic character, use **[!a-zA-Z]***

Escaping and Quoting

Escaping is providing a \ (backslash) before the wild-card to remove (escape) its special meaning.

For instance, if we have a file whose filename is **chap*** (Remember a file in UNIX can be names with virtually any character except the / and null), to remove the file, it is dangerous to give command as **rm chap***, as it will remove all files beginning with chap. Hence to suppress the special meaning of *, use the command **rm chap\***

To list the contents of the file **chap0[1-3]**, use ,

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

A filename can contain a whitespace character also. Hence to remove a file named **My Documend.doc**, which has a space embedded, a similar reasoning should be followed:

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

Quoting is enclosing the wild-card, or even the entire pattern, within quotes. Anything within these quotes (barring a few exceptions) are left alone by the shell and not interpreted. When a command argument is enclosed in quotes, the meanings of all enclosed special characters are turned off.

Examples:

$rm 'chap*'	Removes files chap*
$rm "My Document.doc"	Removes file My Document.doc

Redirection : The three standard files

The shell associates three files with the terminal – two for display and one for the keyboard. These files are streams of characters which many commands see as input and output. When a user logs in, the shell makes available three files representing three streams. Each stream is associated with a default device: -

Standard input: The file (stream) representing input, connected to the keyboard.

Standard output: The file (stream) representing output, connected to the display.

Standard error: The file (stream) representing error messages that emanate from the command or shell, connected to the display.

The standard input can represent three input sources:

The keyboard, the default source.

A file using redirection with the < symbol.

Another program using a pipeline.

The standard output can represent three possible destinations:

The terminal, the default destination.

A file using the redirection symbols > and >>.

As input to another program using a pipeline.

A file is opened by referring to its pathname, but subsequent read and write operations identify the file by a unique number called a 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 as,

0 – Standard input

1 – Standard output

2 – Standard error

These descriptors are implicitly prefixed to the redirection symbols.

Examples:

Assuming file2 doesn’t exist, the following command redirects the standard output to file myOutput and the standard error to file myError.

$ls –l file1 file2 1>myOutput 2>myError

Filters: Using both standard input and standard output

UNIX commands can be grouped into four categories viz.,

1. Directory-oriented commands like mkdir, rmdir and cd, and basic file handling commands like cp, mv and rm use neither standard input nor standard output.
2. Commands like ls, pwd, who etc. don't read standard input but they write to standard output.
3. Commands like lp that read standard input but don't write to standard output.
4. Commands like cat, wc, cmp etc. that use both standard input and standard output.

Commands in the fourth category are called filters. Note that filters can also read directly from files whose names are provided as arguments.

Example:

To perform arithmetic calculations that are specified as expressions in input file calc.txt and redirect the output to a file result.txt, use

$bc < calc.txt > result.txt6.

Pipes: Connecting Commands

With piping, the output of a command can be used as input (piped) to a subsequent command.

$ command1 | command2

Output from command1 is piped into input for command2.

This is equivalent to, but more efficient than:

$ command1 > temp

$ command2 < temp

$ rm temp

Examples

**$ ls -l	wc –l**	**Displays number of file in current directory**
**$ who	wc –l**	**Displays number of currently logged in users**

Creating a tee

tee is an external command that handles a character stream by duplicating its input. It saves one copy in a file and writes the other to standard output. It is also a filter and hence can be placed anywhere in a pipeline. Example:-

The following command sequence uses tee to display the output of who and saves this output in a file as well.

```
$who | tee users.lst
```

Above command displays currently logged in users on standard output and writes a copy to users.lst

Command substitution

The shell enables the connecting of two commands in yet another way. While a 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.

Example:

```
$echo Current date and time is `date`
```

Observe the use of **backquotes** around date in the above command. Here the output of the command execution of date is taken as argument of echo. The shell executes the enclosed command and replaces the enclosed command line with the output of the command. Similarly the following command displays the total number of files in the working directory.

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

Observe the use of double quotes around the argument of echo. If you use single quotes, the backquote is not interpreted by the shell if enclosed in single quotes.

grep: Searching for a pattern

You often need to search a file for a pattern, either to see the lines containing (or not containing) it or to have it replaced with something else. This chapter discusses two important filters that are specially suited for these tasks- grep and sed. This chapter also takes up one of the fascinating features of UNIX – regular expressions (RE).

To discuss all the examples in this chapter we use following **emp.lst** as the reference file.

```
$ cat emp.lst

2233 | a. k. shukla | g. m. | sales | 12/12/52 | 6000

9876 | jai sharma | director | production | 12/03/50 | 7000

5678 | sumit chakrobarty | d. g. m. | marketing | 19/04/43 | 6000

2365 | barun sengupta | director | personnel |11/05/47 | 7800

5423 | n. k. gupta | chairman | admin | 30/08/56 | 5400

1006 | chanchal singhvi | director | sales | 03/09/38 | 6700

6213 | karuna ganguly | g. m. | accounts | 05/06/62 | 6300

1265 | s. n. dasgupta | manager | sales | 12/09/63 | 5600

4290 | jayant choudhury | executive | production | 07/09/50 | 6000

2476 | anil aggarwal | manager | sales | 01/05/59 | 5000

6521 | lalit chowdury | director | marketing | 26/09/45 | 8200

3212 | shyam saksena |d. g. m. | accounts | 12/12/55 | 6000

3564 | sudhir Agarwal | executive | personnel | 06/07/47 | 7500

2345 | j. b. saxena | g. m. | marketing | 12/03/45 | 8000

0110 | v. k. agrawal | g. m. | marketing | 31/12/40 | 9000
```

grep scans its input for a pattern displays lines containing the pattern, the line numbers or filenames where the pattern occurs. The command uses the following syntax:

$grep options pattern filename(s)

grep searches for pattern in one or more filename(s), or the standard input if no filename is specified.

The first argument (except the options) is the pattern and the remaining arguments are filenames.

Examples:

$ grep “sales” emp.lst

```
2233|a. k. shukla |g. m. |sales |12/12/52|6000

1006|chanchal singhvi |director |sales |03/09/38|6700

1265|s. n. dasgupta |manager |sales |12/09/63|5600

2476|anil aggarwal |manager |sales |01/05/59|5000
```

here, grep displays 4 lines containing pattern as “sales” from the file emp.lst.

```
$ grep president emp.lst                    #No quoting necessary here

$ _                                         #No pattern (president) found
```

here, grep silently returns the prompt because no pattern as “president” found in file emp.lst.

$ grep “director” emp1.lst emp2.lst

```
emp1.lst: 9876|jai sharma |director |production |12/03/50|7000

emp1.lst: 2365|barun sengupta |director |personnel |11/05/47|7800

emp1.lst: 1006|chanchal singhvi |director |sales |03/09/38|6700

emp2.lst: 6521|lalit chowdury |director |marketing |26/09/45|8200
```

Here, first column shows file name.

when grep is used with multiple filenames, it displays the filenames along with the output.

grep options:

The below table shows all the options used by grep.

Option	Significance
-i	Ignores case for matching
-v	Doesn't display lines matching expression
-n	Displays line numbers along with lines
-c	Displays count of number of occurrences
-l	Displays list of filenames only
-e exp	Matches multiple patterns
-f filename	Takes patterns from file, one per line
-E	Treats patterns as an ERE
-F	Matches multiple fixed strings

Examples:

- **Ignoring case (-i):**

  When you look for a name but are not sure of the case, use the -i (ignore) option.

  ```
  $ grep -i 'agarwal' emp.lst
  3564|sudhir Agarwal |executive |personnel |06/07/47|7500
  ```

  This locates the name Agarwal using the pattern agarwal.

- **Deleting Lines (-v):**

  The -v option selects all the lines except those containing the pattern.

  It can play an inverse role by selecting lines that does not containing the pattern.

  ```
  $ grep -v 'director' empl.lst
  2233|a. k. shukla |g. m. |sales |12/12/52|6000
  5678|sumit chakrobarty |d. g. m. |marketing |19/04/43|6000
  5423|n. k. gupta |chairman |admin |30/08/56|5400
  ```

```
6213|karuna ganguly |g. m. |accounts |05/06/62|6300

1265|s. n. dasgupta |manager |sales |12/09/63|5600

4290|jayant choudhury |executive |production |07/09/50|6000

2476|anil aggarwal |manager |sales |01/05/59|5000

3212|shyam saksena |d. g. m. |accounts |12/12/55|6000

3564|sudhir Agarwal |executive |personnel |06/07/47|7500

2345|j. b. saxena |g. m. |marketing |12/03/45|8000

0110|v. k. agrawal |g. m. |marketing |31/12/40|9000
```

- **Displaying Line Numbers (-n):**

  The -n(number) option displays the line numbers containing the pattern, along with the lines.

```
$ grep -n 'marketing' emp.lst

3: 5678|sumit chakrobarty |d. g. m. |marketing |19/04/43|6000

11: 6521|lalit chowdury |director |marketing |26/09/45|8200

14: 2345|j. b. saxena |g. m. |marketing |12/03/45|8000

15: 0110|v. k. agrawal |g. m. |marketing |31/12/40|9000
```

  here, first column displays the line number in emp.lst where pattern is found

- **Counting lines containing Pattern (-c):**

  How many directors are there in the file emp.lst?

  The -c(count) option counts the number of lines containing the pattern.

```
$ grep -c 'director' emp.lst

4
```

- **Matching Multiple Patterns (-e):**

With the -e option, you can match the three agarwals by using the grep like this:

```
$ grep -e “Agarwal” -e “aggarwal” -e “agrawal” emp.lst
2476|anil aggarwal |manager |sales |01/05/59|5000
3564|sudhir Agarwal |executive |personnel |06/07/47|7500
0110|v. k. agrawal |g. m. |marketing |31/12/40|9000
```

- **Taking patterns from a file (-f):**

You can place all the patterns in a separate file, one pattern per line.

Grep uses -f option to take patterns from a file:

```
$ cat patterns.lst
director
manager
chairman
$ grep -f patterns.lst emp.lst
9876|jai sharma |director |production |12/03/50|7000
2365|barun sengupta |director |personnel |11/05/47|7800
5423|n. k. gupta |chairman |admin |30/08/56|5400
1006|chanchal singhvi |director |sales |03/09/38|6700
1265|s. n. dasgupta |manager |sales |12/09/63|5600
2476|anil aggarwal |manager |sales |01/05/59|5000
6521|lalit chowdury |director |marketing |26/09/45|8200
```

BASIC REGULAR EXPRESSION (BRE)

Like the shell's wild-cards which matches similar filenames with a single expression, grep uses an expression of a different type to match a group of similar patterns. Unlike shell's wild-cards, grep uses following set of meta-characters to design an expression that matches different patterns.

If an expression uses any of these meta-characters, it is termed as **Regular Expression (RE).**

The below table shows the BASIC REGULAR EXPRESSION(BRE) character set-

Symbols or Expression	**Matches**
*	Zero or more occurrences of the previous character
g*	Nothing or g, gg, ggg, gggg, etc.
.	A single character
.*	Nothing or any number of characters
[pqr]	A single character p, q or r
[c1-c2]	A single character withing ASCII range shown by c1 and c2
[0-9]	A digit between 0 and 9
[^pqr]	A single character which is not a p, q or r
[^a-zA-Z]	A non-alphabetic character
^pat	Pattern pat at beginning of line
pat$	Pattern pat at end of line
^bash$	A bash as the only word in line
^$	Lines containing nothing

Examples:

- ### The character class

  A RE lets you specify a group of characters enclosed within a pair of rectangular brackets, [ ], in which case the match is performed for a single character in the group.

  ```
  $ grep '[aA]g[ar][ar]wal' emp.lst

  3564|sudhir Agarwal |executive |personnel |06/07/47|7500

  0110|v. k. agrawal |g. m. |marketing |31/12/40|9000
  ```

- **The ***

  The * (asterisk) refers to the immediately preceding character.

  Here, it indicates that the previous character can occur many times, or not at all.

```
$ grep '[aA]gg*[ar][ar]wal' emp.lst

2476|anil aggarwal |manager |sales |01/05/59|5000

3564|sudhir Agarwal |executive |personnel |06/07/47|7500

0110|v. k. agrawal |g. m. |marketing |31/12/40|9000
```

- **The Dot**

  A . matches a single character.

  The pattern 2... matches a four-character patten beginning with a 2.

  The pattern .* matches any number of characters, or none.

```
$ grep 'j.*saxena' emp.lst

2345|j. b. saxena |g. m. |marketing |12/03/45|8000
```

- **Specifying pattern locations (^ and $)**

  ^ (carat) – For matching at the beginning of a line

  $ (dollar) – For matching at the end of a line

```
$ grep '^2' emp.lst

2233|a. k. shukla |g. m. |sales |12/12/52|6000

2365|barun sengupta |director |personnel |11/05/47|7800

2476|anil aggarwal |manager |sales |01/05/59|5000

2345|j. b. saxena |g. m. |marketing |12/03/45|8000
```

$ grep '7...$' emp.lst

```
9876|jai sharma |director |production |12/03/50|7000

2365|barun sengupta |director |personnel |11/05/47|7800

3564|sudhir Agarwal |executive |personnel |06/07/47|7500
```

- **To display all lines that don't begins with a 2**

$ grep '^[^2]' emp.lst

```
9876|jai sharma |director |production |12/03/50|7000

5678|sumit chakrobarty |d. g. m. |marketing |19/04/43|6000

5423|n. k. gupta |chairman |admin |30/08/56|5400

1006|chanchal singhvi |director |sales |03/09/38|6700

6213|karuna ganguly |g. m. |accounts |05/06/62|6300

1265|s. n. dasgupta |manager |sales |12/09/63|5600

4290|jayant choudhury |executive |production |07/09/50|6000

6521|lalit chowdury |director |marketing |26/09/45|8200

3212|shyam saksena |d. g. m. |accounts |12/12/55|6000

3564|sudhir Agarwal |executive |personnel |06/07/47|7500

0110|v. k. agrawal |g. m. |marketing |31/12/40|9000
```

- **To display only directories using ls -l and grep**

$ ls -l | grep '^d'

```
drwxr-xr-x 2 chandrakant chandrakant 4096 Jan 16 12:18 Desktop

drwxr-xr-x 2 chandrakant chandrakant 4096 Jan 13 07:54 Documents

drwxr-xr-x 7 chandrakant chandrakant 4096 Jan 16 09:41 Downloads
```

EXTENDED REGULAR EXPRESSION (ERE) AND egrep

ERE make it possible to match dissimilar patterns with a single expression.

grep uses ERE characters with -E option.

egrep is another alternative to use all the ERE characters without -E option.

This **ERE** uses some additional characters set shown in below table-

Expression	**Significance**
ch+	Matches one or more occurrences of character ch
ch?	Matches zero or one occurrence of character ch
exp1 | exp2	Matches exp1 or exp2
GIF | JPEG	Matches GIF or JPEG
(x1|x2)x3	Matches x1x3 or x2x3
(hard|soft)ware	Matches hardware or software

Examples:

- **The + and ?**

  + - Matches one or more occurrences of the previous character

  ? - Matches zero or one occurrence of the previous character.

  ```
  $ grep -E “[aA]gg?arwal” emp.lst

  2476|anil aggarwal |manager |sales |01/05/59|5000

  3564|sudhir Agarwal |executive |personnel |06/07/47|7500
  ```

- **Matching Multiple Patterns(|, (and))**

  ```
  $ grep -E 'sengupta|dasgupta' emp.lst

  2365|barun sengupta |director |personnel |11/05/47|7800

  1265|s. n. dasgupta |manager |sales |12/09/63|5600

  $ grep -E '(sen|das)gupta' emp.lst

  2365|barun sengupta |director |personnel |11/05/47|7800

  1265|s. n. dasgupta |manager |sales |12/09/63|5600
  ```