The Shell

Definitions

- **Bash** - **Bash** is an **sh**-compatible command ***language interpreter*** that executes commands read from the standard input or from a file. **Bash** also incorporates useful features from the Korn and C shells (**ksh** and **csh**).
- **blank -** A space or tab.
- **word -** A sequence of characters considered as a single unit by the shell. Also known as a **token**.

- **name -** A word consisting only of alphanumeric characters and underscores, and beginning with an alphabetic character or an underscore. Also referred to as an **identifier**.
- **metacharacter -** A character that, when unquoted, separates words. One of the following:

  | & ; ( ) < > space tab
- **control operator -** A token that performs a control function. It is one of the following symbols:

  || & && ; ;; ( ) | |& <newline>
- **Special Pattern and Sub Pattern characters:**

  *** ? […] + @ Note: - ! Are used inside[]**

Topics

- The Shell's Interpretive Cycle.
- Wild cards and file name generation.
- Removing the special meanings of wild cards.
- Three standard files and redirection.
- Connecting commands: Pipe.
- Splitting the output: tee.
- Command substitution.
- Basic and Extended regular expressions.
- The grep, egrep.
- Typical examples involving different regular expressions.

The Shell's Interpretive Cycle

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

1. **Issues the prompt** and waits for the user to enter a command.
2. Scans the command line for metacharacters and expands the abbreviations(like *,? associated with filenames) to **recreate a simplified command line**.
3. **Pass command** line to kernel for execution.
4. **Wait** for the command to complete. While command is running shell can't do any work.
5. The **prompt reappears** after the command execution is complete. The above steps are repeated.

Wild cards and filename generation

- In Unix wildcards are commonly used in
  - ***shell* commands** - To **match files and directories names** and is *interpreted by shell*.
  - ***regular expressions(RE)*** - *Used by command* like "grep, sed, awk " to search **for *matching a RE against a text*** in a file, multiple files, or a stream of i/p.
  - ***Shell programming language/Shell script*** - *Used in **case statement** to match strings to match the i/p string to a switch statement.*

The Shell Wild cards and filename generation

- A ***wildcard*** is a character that can ***stand for all members of some class of characters***. When you use a wildcard the shell substitutes the members of the class for the wildcard character.
- A ***wildcard*** is a character that can be used as a **substitute** for any of a ***class of characters*** in a search, thereby greatly increasing the ***flexibility and efficiency of searches***.

The Unix shell wildcard characters: A number of characters are interpreted by the Unix shell before any other action takes place. These characters are known as wildcard characters. Usually these characters are used in place of filenames or directory names.

- ***** → An asterisk matches any number of characters in a filename, including none.
- **?** →The question mark matches any single character.
- **[ ]**→ Brackets enclose a set of characters, any one of which may match a single character at that position.
  - **-** →A hyphen used within [ ] denotes a range of characters.
  - ! → Negation indicates excluding the range of characters.

- **~** →A tilde at the beginning of a word expands to the name of your home directory.
- **[ijk]** → A single character - either I or j or k.
- **[x-z]** →A single character that is within ASCII range of characters x and z.
- **[!ijk]**→ A single character that is not an I or j or k.
- **[!x-z]** →A single character that is not within ASCII range of characters x and z.
- **{pat1,pat2,…}** →pat1, pat2, etc..

To display multiple files listing with similar names using * and ? wildcard characters

- E.g. chap chap01 chap02 chap03 chapx chapxy chapz be the filenames in the current working directory.
- ***Frame the pattern with the ordinary characters like chap and wildcard character * as chap**** that represents all filenames beginning with chap.
- This pattern can be now an argument to command like ls.(Now the command is ls chap*.)

- The shell will expand * suitably before the command is executed.
- The shell looks in the current directory and recreates the command line as ls chap chap01 chap02 chap03 chapx chapxy chapz and hands over this command line to kernel.
- The kernel creates the process for the above to run the command.

The ? Wildcard character matches a single character.

- E.g. chap chap01 chap02 chap03 chapx chapxy chapz be the filenames in the current working directory.
- For the command ls chap?, the shell looks in the current directory and matches all five characters filenames as chapx, chapy, chapz.
- For the command ls chap??, the shell looks in the current directory and matches all six characters filenames as chap01, chap02, chap03.

What * and ? Cannot match?

- The filenames that begin with a . (dot)
  - E.g .bashrc
- Don’t match the / in a pathname.
  - ls /home?administrat*hjr

Matching all files beginning with a . (dot)

- Generally .file_names/.dir_names are hidden files/ hidden directories.
- To display the above use ls -a

The character class- []

- Using * and ? wildcard characters the ***restrictive patterns cannot be framed***.
- Using character class the ***restrictive patterns can be framed.***
- Using character class
  - Can match a single character in the class. E.g. ls chap0[123] , lists chap0, chap01, chap03.
  - Can specify the range of characters in the increasing order of ASCII value. E.g ls chap[x-z], lists chapx, chapy, chapz.

- ***Negating the character class (!):*** Reverses the matching criteria.
- E.g.ls *.[!co], matches all filenames with a single character extension but not .c and .o files.
- E.g. ls [!a-zA-Z]*, matches all filenames that don’t begin with alphabetic character.
- To match all files except those ending with .o, a ***character class should be used as a dummy class.*** E.g. ls *.[!o], dummy class because in the character class only one character is used.

Matching Totally Dissimilar Patterns

- E.g. to copy all C and Java source files from another directory: cp ../*.{c,java} .
- Here in the above example:
  - The patterns are delimited with a comma, and
  - Curly braces are placed around them.

Summary

- Wildcard characters have different meanings depending on where they are placed in the pattern.
- The * and ? Lose their meaning when used inside the character class. They are matched literally.
- The - and ! Also lose their significance when placed outside the class.

Write the output of the following

- ls *.c
- mv * ../bin
- cp foo foo*
- cp ?????? Progs
- lp note[0-1][0-9]
- rm *.[!l][!o][!g]
- cp –r /home/hjr/{include,bin,lib} .

Removing the special meanings of wild cards

- Two Techniques:
  - ***Escaping:*** Providing a \ (back slash) before the wildcard character to remove (escape) its special meaning.
  - ***Quoting:*** Enclosing the wildcard, or even the entire pattern, within quotes (single or double) its special meaning is turned off.

- Single quotes protect all special characters except the single quote.
- Double quote are more permissive; they don't protect double quotes, the $ (the variable prefix) and the ` (backquote, the command substitution character).

Note: Double quotes protect single quotes and vice-versa is true.

Redirection: The Three Standard Files

- Definitions:
  - **Terminal:** In the context of redirection, the terminal is a generic name that represents
    - Screen/ Display
    - Keyboard
  - **Streams:** a *stream* is a sequence of data elements made available over time or a sequence of bytes.

- **Standard Streams** are *pre-connected input and output communication channels between a computer program and its environment when it begins execution*.
- The three I/O connections are called **standard input** (**stdin**), **standard output** (**stdout**) and **standard error** (**stderr**).

NOTE: A child process will inherit the standard streams of its parent process.

Standard Streams associated with default devices

- **Standard Input** – The file/ stream representing input, which is connected to the keyboard.
- **Standard Output** – The file/ stream representing output, which is connected to the display.
- **Standard Error** – The file/ stream representing error messages that emanate from the command or shell. This is also connected to the display.

Standard Input

- Different sources
  - The **keyboard**, the default source.
  - A **File using Redirection with < symbol** (a metacharacter).
  - **Another program using a pipeline** (**|** symbol, a metacharacter).
  - **Here document(<< is the symbol)**: commands using standard i/p can take i/p from a here document. Signifies data is here rather than in a separate file.

- ***Taking input both from file and standard input:***
  - When a command take input from standard i/p, the -(hyphen) symbol must be used to indicate the sequence of taking i/p.
  - E.g. cat – foo, cat foo – bar, where – is i/p from keyboard, foo and bar are file names.

UNIX **vi** Editor

(**vi**sual editor)

Editor(s)

- Editor is a utility that facilitates the editing task – creation and modifications of text files quickly and efficiently.
- Types of editors
  - Screen editors, e.g. vi, vim, pico, joe, emacs ,ed.
    - Provides a whole screen of text at a time.
  - Line editors, e.g. sed, ex.
    - Are useful to make global changes over a(line or) group of lines.

vi - screen editor

- Is a full screen editor and has **three modes** of operation:
- ***Command mode***
- ***Insert mode / Text mode/ Input Mode***
- ***Last line mode/ ex mode***

Note: the first two are basic modes.

- ***Command mode*** - commands which cause immediate action or command execution over a texts of a open file.
- Actions like save, quit, cursor movements , cut, copy, paste, search, replace, substitution and many others.
- Two aspect of commands:
  - **Commands are not echoed** *on the screen* **except colon(:), Forward slash(/) and question mark(?).**
  - **Most commands should not be followed by Enter key *except :, /, and ?.***

- ***Insert mode / Text mode / Input Mode*** - in which entered text is inserted into the file.
- ***Last line /ex Mode mode*** - The cursor will jump to the last line of the screen and wait for a command.

Command Category

Entering and Replacing text	Undoing
Saving text and quitting	Repeating
Navigation	Pattern searching
Editing Text	Substitution – Search and Replace

Input Mode Commands

Command	Function/ Action
i	Inserts text to left of cursor.
a	Appends text to right of cursor.
I	Inserts text at beginning of line.
A	Appends text at end of line.
o	Open line below.
O	Open line above.
r*ch*	Replace single character under cursor with ch. (no ESC required).
R	Replace text from cursor to right. (Existing text overwritten).
s	Replace single character under cursor with a string.
S	Replace entire line.

ex Mode commands (shown are file save and exit commands only)

Command	Action
:w	Write to file and remain in vi editor.
:x	Write to file and quit vi editor.
:wq	Write to file and quit vi editor.
:q	Quit vi editor when no changes are made to file.
:q!	Quit vi editor abandon all changes to a file.
:sh	Create a shell within the vi editor to execute shell commands. To return back to vi editor **type exit or ctrl-d**.
:!cmd	Execute shell command without leaving vi editor.
:recover	Recover file from crash.
:n1,n2w abc.txt	Write lines n1 to n2 to file abc.txt.
:.w abc.txt	Write current line to file abc.txt.
:$w abc.txt	Write last line to file abc.txt.

Command mode commands

Command Category	
Navigation	
Title/ Name	**Commands**
Cursor Movements	**k, j** (up, down) **h, l** (left, right)
Word Navigation	**b** – move back to beginning of word. **e** - move forward to end of word. **w**- move forward to beginning of word.
Moving to Line Extremes	**0(ZERO)** – move to beginning of the current line. **$**- move to the end of the current line. **|** - position the cursor on a particular column. E.g. 30**|**

Command mode commands

Command Category	
Navigation	
Title/ Name	**Commands**
Scrolling	**Ctrl-f**← scrolls forward. **Ctrl-b** ← scroll backward. **Ctrl-d** ← scroll half page forward. **Ctrl-u** ←scroll half page backward.
Absolute Movements	**G, nG.**
Note: ex mode equivalents of absolute movements	**:number, :$**

Command mode commands

Command Category	
Editing Text	
Title/ Name	**Commands**
Deleting Text	**x**- delete character under cursor. **dd-** delete entire line.
Moving Text (from buffer)	**p**- put the text after cursor. **P**-put the text before cursor.
Copying Text	**yy** – Yank current line
Joining Lines	**J**- removes newline character between the two lines to pull up the line below it.

Command mode commands

Command Category	
Undoing Last Editing	
Title/ Name	**Commands**
Undoing Last Editing	**u**- undo last change. **U**- undo all changes on line.

Command mode commands

Command Category	
Repeating the last command	
Title/ Name	**Commands**
Repeating the last command	• (read dot) – can be applied to repeat the last editing instruction like insertion, deletion, or other actions that modifies the buffer.

Command mode commands

Command Category	
Searching for a pattern	
Title/ Name	**Commands**
Searching for a pattern	**/pat** – search forward for pattern pat.
	?pat – search backward for pattern pat.
	n- Repeat search in same direction of original search.
	N- Repeat search in direction opposite to previous search made.

TRIO of search

- To be carried out at a number of places in combination of
  - / (search command)
  - n (repeat search command)
  - . (repeat last editing command)

Substitution-Search and Replace (:s)

- The general form:

<:><address><s/source_pattern/target_pattern/flags>

Flag – g (global)

Address: .,$,n1..n2

Vi Editor Cheat Sheet

Movement Commands

Character

h, **j**, **k**, **l**	Left, down, up, right

Text

w, **W**, **b**, **B**	Forward, backward by word
e, **E**	End of word
(, **)**	Beginning of next, previous sentence
{, **}**	Beginning of next, previous paragraph
[[, **]]**	Beginning of next, previous section

Lines

0, **$**	First, last position of current line
^	First non-blank character of current line
+, **-**	First character of next, previous line
H	Top line of screen
M	Middle line of screen
L	Last line of screen
*n***H**, *n***L**	Line *n* from top, bottom of screen

Scrolling

[Ctrl]**F**, [Ctrl]**B**	Scroll forward, backward one screen
[Ctrl]**D**, [Ctrl]**U**	Scroll down, up one-half screen
[Ctrl]**E**, [Ctrl]**Y**	Show one more line at bottom, top of window
z[Enter]	Scroll until line with cursor is at top of screen
z.	Scroll until line with cursor is at middle of screen
z-	Scroll until line with cursor is at bottom of screen

Searches

/*pattern*	Search forward for *pattern*
?*pattern*	Search backward for *pattern*
n, **N**	Repeat last search in same, opposite direction
/, **?**	Repeat previous search forward, backward
f*x*	search forward for character *x* in current line
F*x*	search backward for character *x* in current line
t*x*	search forward for character before *x* in current line
T*x*	search backward for character after *x* in current line
;	Repeat previous current-line search
,	Repeat previous current-line search in opposite direction

Line Number

[Ctrl]**G**	Display current line number
*n***G**	Move to line number *n*
G	Move to last line in file
:*n*	move to line number *n*

Marking Position

m*x*	Mark current position as *x*
`*x*	Move cursor to *x*
``	Return to previous mark or context
'*x*	Move to beginning of line containing mark *x*
''	Return to beggining of line containing previous mark

Editing Commands

Insert

i, **a**	Insert text before, after cursor
I, **A**	Insert text at beginning, end of line
o, **O**	Open new line for text below, above cursor

Change

r	Replace with next typed characer

~	Change between uppercase and lowercase
c*m*	Change text block defined by movement command *m* (e.g., cw changes next word)
cc	Change current line
C	Change to end of line
R	Type over characters
s	Delete character and continue typing
S	Delete current line and continue typing

Delete, Move

x	Delete character
X	Delete character to the left of the cursor
d*m*	Delete text block defined by movement command *m* (e.g., dw deletes next word)
dd	Delete current line
D	Delete to end of line
p, **P**	Put deleted text before, after cursor
"*n***p**	Put text from delete buffer number *n* after cursor (for last nine deletions)

Yank (copy)

y*m*	Yank (copy) text block defined by movement command *m* (e.g., yw yanks next word)
yy, **Y**	Yank current line
"*a***yy**	Yank current line into named buffer *a*
p, **P**	Put yanked text before, after cursor
"*a***P**	Put text from buffer *a* before cursor

Other Commands

.	Repeat last edit command
u	Undo last edit
U	Undo changes to current line
J	Join two lines
[Ctrl]**L**, [Ctrl]**R**	Redraw screen

Invoking vi

vi *file*	Invoke vi editor on *file*
vi *file1 file2*	Invoke vi editor on files sequentially
view *file*	Invoke vi editor on *file* in read-only mode
vi -R *file*	Invoke vi editor on *file* in read-only mode
vi -r *file*	Recover *file* and recent edits after system crash
vi + *file*	Open *file* at last line
vi +*n file*	Open *file* at line number *n*
vi +/*pattern file*	Open *file* at *pattern*

Exit and Save Commands

ZZ	Save file and quit
:x	Save file and quit
:wq	Save ("write") file and quit
:w	Save file
:w!	Save file (overriding protection)
:30,60w *newfile*	Save lines 30 through 60 as file *newfile*
:30,60w>> *file*	Append lines 30 through 60 to file *file*
:w %.*new*	Save current buffer named *file* as *file.new*
:q	Quit
:q!	Quit, discarding any changes
Q	Quit vi and invoke ex
:e *file2*	Edit *file2* without leaving vi
:e! *file2*	Discard changes to current file, then edit *file2* without leaving vi
:n	Edit next file
:e!	Discard all changes since last save
:e#	Edit alternate file