**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE - PILANI**

K.K. BIRLA GOA CAMPUS

Second Semester 2016-2017

**Lab-sheet 2**

----------------------------------------------------------------------------------------------------------------------------- ---------------

**Objectives:**

1. File Editing in vi
2. I/O Redirection
3. Some Unix Commands
4. Filters & Pipes
5. Pattern Matching

----------------------------------------------------------------------------------------------------------------------------- ---------------

**INTRODUCTION**

You have learned about vi editor in the previous week. In this week, we will explore a little more on vi editor. In UNIX, the keyboard is defined as a standard input device and the computer monitor is defined as a standard output device. If a command is defined to take input from the standard input, it means it takes its input from the keyboard. Similarly, if a command gives its output to the standard output, it means it displays the output to the monitor. UNIX allows us to temporarily change the standard input and standard output by means of what is called as Indirection & Piping.

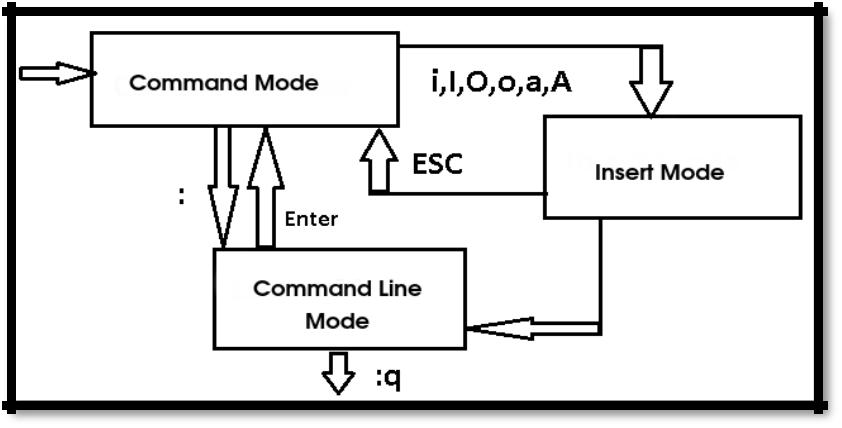
**PART – 1 (VI EDITOR)**

**Basic Operating Modes in vi**

We mentioned while in the process of creating file, that we are switching between two operating modes. The vi Editor at any instant can be found to be in one of the following 3 modes:

* Command Mode
* Command Line Mode
* Insert mode

The navigation among the modes is best illustrated by the following figure:



The technical details of each of the modes are as follows:

**Command Mode:**

Once you finished entering or changing the text, you can move to command mode by pressing **ESC** key. Unlike in Insertion mode, when a key is pressed in the command mode, it doesn’t show up on the screen but it performs some function based on the key pressed.

Refer to the **Table-1** for the basic cursor movement commands and **Table-2** for commands for editing a text.

|  |  |  |
| --- | --- | --- |
|  | **Table-1 Basic Cursor Movement Commands** | |
| **h or left arrow key** |  | Move cursor to the left one character. |
|  |  |  |
| **l or right arrow key** |  | Move cursor to the right one character. |
|  |  |  |
| **j or down arrow key** |  | Move cursor down one line. |
|  |  |  |
| **k or up arrow key** |  | Move cursor up one line. |
|  |  |  |
| **^** |  | Move cursor to the beginning of the line. |
|  |  |  |
| **$** |  | Move cursor to the end of the current line. |
|  |  |  |
| **G** |  | Move cursor to the last line of your file. |
|  |  |  |
| **w** |  | Move to first character of next word. |
|  |  |  |
| **b** |  | Move to first character of previous word. |
|  |  |  |

**Table-2 Commands for editing text**

**i** Inserts text to the left of the cursor.

**I** Inserts text at the beginning of the line, no matter where the cursor is positioned on the current line.

**a** Begins inserting after the character (append) on which the cursor is positioned.

**A** Begins inserting at the end of the current line, no matter where the cursor is positioned on that line.

1. Begins inserting text on a new line below the current line, no matter where the cursor is positioned on that line.

**O** Begins inserting text on a new line above the current line.

**INS** Toggle between insert mode and replace mode. If not currently in Insert Mode, works as same askey pressing **i**.

**Insert Mode:**

vi editor opens in command mode. To perform the said activities on the file, we should change the current working mode to insert mode.

Insert mode allows editing the contents of the current file, making a new file and other manipulations with the file content. This can be done by pressing the keys **i, a, o, O** at command mode **(Table-2).** Once we are changed to insert mode, any character of the keyboard can be typed and written on the screen.

***Task****:*Practice to use**i, a, o, O**with the file you have created i.e. will. While working with it, try to keeptrack of the current mode you are in.

**Command Line Mode:**

The bottom line of the vi editor is called command line. This mode allows giving command at the command line. (This mode is also called as ex command mode, as the commands issued at the command line are compatible with ex editor.) All commands issued in command line mode are displayed in the command line.

Let us introduce a minimal set of commands issued at command line mode here.

|  |  |
| --- | --- |
| **Command** | **Purpose** |
|  |  |
| **:w** | To save your file but not quit *vi.* |
|  |  |
| **:w** <file-name> | To save the contents in the screen (buffer) into the file named *filename*. |
|  |  |
| **:q** | To quit if you haven't made any edits. |
|  |  |
| **:q!** | To quit without saving the changes made in the current edit. |
|  |  |
| **:wq** | To quit and save your edits. |
|  |  |

**Editing Commands in vi**

**1.** **Commands for Copying and Pasting Text**

The commands for copy/paste are issued in command mode. Make sure that you are in command line mode. If not, first switch to command mode by pressing Esc key.

|  |  |
| --- | --- |
| **Command** | **Explanation** |
|  |  |
| **yy** | Copy (yank) the current line. |
|  |  |
| **6yy** | Copy Six lines, beginning with the current line. |
|  |  |
| **yw** | Copy (yank) the current word |
|  |  |
| **:n1,n2y** | Copies line n1 to n2 (both inclusive) |
|  |  |
| **p** | Put the copied text in next line. |
|  |  |
| **P** | Put the copied text above the current line. |
|  |  |
| **:put** | Put copied text after cursor position |
|  |  |

Once you issue the copy command (**yy, 2yy, 4yy** etc), the copied text goes into a temporary memory area (buffer) that is replaced each time when you copy (or delete) more text. Only the current contents of the temporary buffer can be put back into your file. As a result, when you use copy (**yy**), use the put (**p**) command immediately.

**2.** **Commands for Deleting the file contents**

To delete a character/word/line, it is necessary first to position the cursor at the appropriate place. For example, when you are in command mode, pressing x will cause deleting the character over the cursor. Similarly, from the current cursor position, you want to delete 3 characters. This can be done by 2 ways:

1. Bring the cursor over the first of n characters you want to delete. And in command mode, press the key x, n times. That is for deleting 3 characters from the current position; you will press the key x, 3 times.
2. Alternatively you may bring the cursor to the first of n characters you want to delete and issue the command nx. That is, to delete you will press 3x. This command will delete 3 characters from the current cursor position.

The following table (Table-3) shows the commands issued at command mode to perform deletion.

|  |  |
| --- | --- |
| **Command** | **Explanation** |
|  |  |
| **x** | Deletes the character at the current cursor position |
|  |  |
| **X** | Deletes the character left to the cursor |
|  |  |
| **dw** | Deletes a word from the cursor to the next space, or to the next punctuation |
|  |  |
| **dd** | Deletes the current line |
|  |  |
| **nx, ndw, ndd** | Deletes n characters, n words, n lines respectively |
|  |  |
| **d0** | Deletes the current line from the cursor to the beginning of the line |
|  |  |
| **d$** | Deletes the current line from the cursor to the end of the line |
|  |  |
| **3. Showing Line number:** | |

|  |  |
| --- | --- |
| **Command** | **Explanation** |
|  |  |
| **:set number** | Shows line numbers. |
|  |  |
| **:se nu** | Shows line numbers (shortcut for set number). |
|  |  |
| **:set nonumber** | Hides line numbers. |
|  |  |
| **:se nonum** | Hides line numbers (shortcut). |
|  |  |
| **:set number!** | Toggle between showing and hiding line numbers. |
|  |  |
| **:se nu!** | Toggle between showing and hiding line numbers. |
|  |  |

**4.** **Searching for a Text**

Search for a string or a character is possible with vi in command mode. For a string search, the **/** and **?** commands are used. When you start these commands, the command just typed will be shown on the bottom line where you type the particular string to look for. These two commands differ only in the direction where the search takes place. The **/** command searches forwards (downwards) in the file, while the **?** command searches backwards (upwards) in the file. List of commands are as follows:

|  |  |
| --- | --- |
| **Command** | **Explanation** |
|  |  |
| **/**text | Search forward (down) for text. |
|  |  |
| **?**text | Search backward (up) for text. |
|  |  |
| **f<char>** | Search forward for a character on current line (e.g. fb). |
|  |  |
| **F<char>** | Search backward for a character on current line. |
|  |  |
| **n** | Repeat last search in the same direction |
|  |  |
| **N** | Repeat last search in the opposite direction |
|  |  |
| **;** | Repeat last character search in the same direction |
|  |  |
| **%** | Find matching ( ), [ ], or { } |
|  |  |

**5.** **Substituting the Text**

If you want to do substitutions over a range of lines, or throughout the file, the **s** command is used as follows:

**:n1,n2s/old/new/gc** **-** oldtext is replaced by newtext from lines n1 to n2

**:%s/old/new/g** -Substitutes old with new throughout the line.

**:.,$s/old/new/g** -Substitutes old with new from the current cursor position to the end of the line.

**:1,.s/old/new/g** -Substitutes old with new from beginning of the file to the current cursor position.

where,

**n1** - is the beginning line number.

**n2 -** is the ending line number.

**s -** Means to substitute text matching the pattern (**old**) with text specified by (**new**).

**g -** Stands for global (optional). Indicates you want to substitute all occurrences on the indicatedlines. If you don’t use **g**, the editor substitutes only the first occurrence on the indicated lines.

**c -** stands for confirm (optional). It indicates you want to confirm each substitution before vicompletes it.

**6.** **Miscellaneous vi Commands**

|  |  |
| --- | --- |
| **Command** | **Explanation** |
| **R** | Replaces the content of the file with whatever you type from the position where the cursor is |
|  | initially placed. This command brings you to insert mode, where the only editing operation |
|  | possible is to replacing the contents over the cursor |
| **.** | Repeats the action performed by the last command |
| **u** | Undoes the effect of the last command |
| **U** | Restores the changes to the current line since you moved the cursor to this line |
| **w** | Move to the next word |
| **b** | Move to the previous word |
| **J** | Joins the line immediately below the current line with the current line |
| **~** | Changes the character at current position from uppercase to lower case or vice versa |
|  |  |

**Exercises**

Q 1) Prepare a file containing the given 6 lines below and perform the operations stated below (each line in the following text should be a line ending with an enter key in your file).

**I had no where to turn, had no where to go. this is just something I think you need to know I don't know what made me trust you**

**I still remember the day when I told you what I've been through I thought I should run away, go hide in a hole**

**but then you really brought out my true soul**

1. Copy (Yank) the first 2 lines in your file and paste it after line 4 in your file using Nyy and p commands.
2. Copy (Yank) the lines 4 to 5(both inclusive) in your file and paste it before line 1 in your file using :n1,n2y and P commands.
3. Copy (Yank) the lines 3 and 4 in your file and paste it after line 5 in your file using Nyy and :put commands.
4. Copy (Yank) the first 3 lines in your file and paste it after the last line your file using :n1,n2yand :put commands.

Q 2) Create a file named exer1 with following lines (each line in the following text should be a line ending with an enter key in your file) and perform the operations stated below.

**Raise the door or just take off the cover, there she sits just waiting for us to hover.**

**Our first glance we know she waits, just as our mind anticipates.**

**As we sit on the seat, a turn of the key,**

**we know for sure this is the way it should be.**

1. Copy (Yank) the last 2 lines in your file and paste it after line 2 in your file using Nyy and p commands
2. Copy (Yank) the lines 3 to 5 (both inclusive) in your file and paste it before line 2 in your file using :n1,n2y and P commands
3. Copy (Yank) the lines 4 and 5 in your file and paste it after line 4 in your file using Nyy and :put commands
4. Copy (Yank) the first 3 lines in your file and paste it after the last line your file using :n1,n2yand :put commands.
5. 3)
   1. Create a file named MyFile using VI editor. Write the following two lines into the file

**Spades take up leaves**

**No better than spoons,**

Save the file and exit the VI editor.

1. Open MyFile and insert the following two lines after the last line of the file and save the file as MyFile2.

**But the mountains I raise Elude my embrace,**

1. Add the following statement in the last line [After comma] and save the file as MyFile3

**Flowing over my arms**

d) Insert the following lines after line 2 [before line 3] and save the file as MyFile4

**And bags full of leaves Are light as balloons.**

**I make a great noise Of rustling all day**

e) Insert the following line after line 5, save MyFile4 without exiting from VI editor.

**And into my face.**

1. Copy (Yank) the current line and paste it in a new line after the last line. Save the file as MyFile5.
2. Delete the 5th line from the last. Save the file as MyFile6 and exit from the VI editor.
3. Copy the first two lines of MyFile6 file and paste them thrice after the 4th line. Save the file as MyFile7 and exit the VI editor.
4. Cut the 5th to 6th lines from the file and paste them before the 4th line of your file. Save the file as MyFile8.
5. Join the 5th and 6th line using vi editor commands. Join 6th line to 8th line by using only one join operation [using vi editor command] and save the file as MyFile9.
6. Delete lines 5 to 8. Save the file as FinalFile and exit the VI editor.
7. 4)
   1. Create a file named ExFile using VI editor. Write the following three lines into the file

**Next to nothing for weight, And since they grew duller Next to nothing for use.**

Save the file and exit the VI editor.

b) Open ExFile and insert the following two lines after line 2 of the file and save the file as ExFile2

**From contact with earth, Next to nothing for color.**

1. Move the cursor to the 5th line of the file [without using arrow keys] and write the following lines there. Save the file as ExFile3

**And who's to say where The harvest shall stop?**

1. Delete last character of the first line and repeat the same action for all other lines [from command mode] and save the file as ExFile4.
2. Copy the 3rd to 6th lines of the ExFile4 and paste them twice after line 4. Save the file as ExFile5.
3. Convert the first character of the first word [first line] into uppercase and repeat the action for each word. Save the file as ExFile6.
4. Join the 2nd and 3rd line by using vi editor command. Save the file as LastFile and exit the vi editor.
5. 5)
   1. Create a file named CsFile using VI editor. Write the following three lines into the file

**As long as it takes to pass A ship keeps raising its hull; The wetter ground like glass**

Save the file and exit the VI editor.

1. Open CsFile and insert the following three lines in the beginning of the file and save the file as CsFile2

**The people along the sand All turn and look one way.**

**They turn their back on the land.**

1. Move the cursor to the last line [without using arrow keys]. Add the following line as the second last line and save the file as CsFile3.

**Reflects a standing gull**

1. Delete the last two words of first line and repeat the action for each line. Save the file as CsFile4.
2. Replace the last word of the first line with “###”. Repeat this action for last word of each line. Save the file as FinFile.
3. 6)
   1. Create a file named CpFile using VI editor. Write the following three lines into the file

**The water comes ashore,**

**And the people look at the sea. They cannot look out far.**

Save the file and exit the VI editor.

1. Copy (Yank) last two lines of the file and paste it after the last line. Delete the last two words of the 4th line and save the file as CpFile2
2. Write the string “**in deep**” from the current cursor position in **append mode**. Delete the current line in command mode. Save the file as CpFile3.
3. Insert the following lines before the first line and save the file as CpFile4.

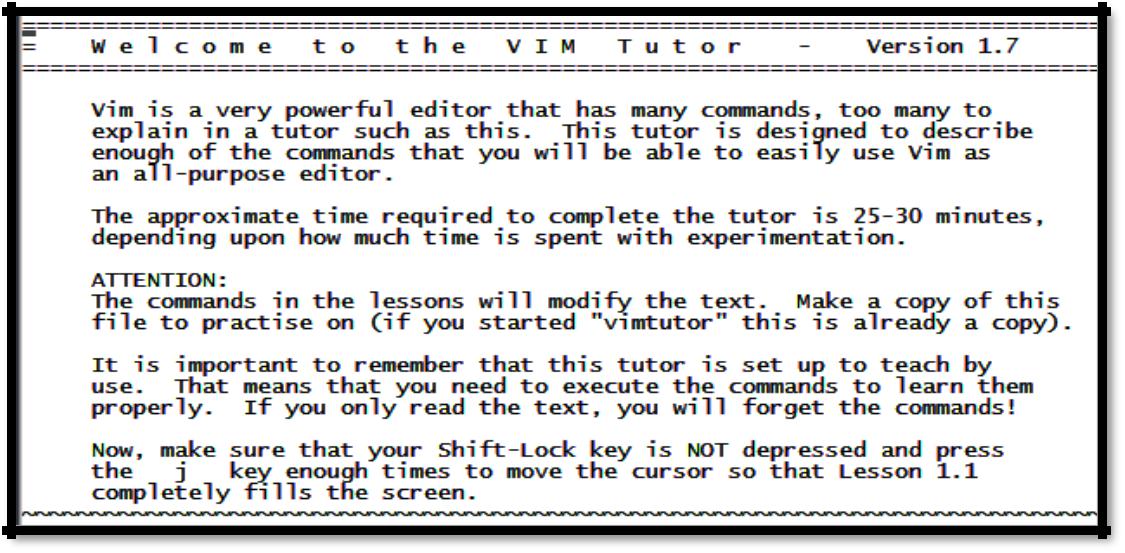
**The land may vary more;**

**But wherever the truth may be -**

1. Copy the 4th line to 6th line and paste it after the last line (use **put**). Copy the current line and paste it after the 2nd line. Save this as CpFile5.
2. Open CpFile5 in VI editor and delete the first two words of each line using **d0**. Save the file as CpFile6.
3. Replace the first word of the first line with the word “First” using 'Replace Command' and repeat this action for next two lines. Save the file as CpFile7 without exiting from VI editor.
4. Copy (Yank) the current line and paste it before 3rd line without moving the cursor from its current location. Save file as CpFile8 and exit VI editor.

**Additional Reading**

Most of Linux systems have command “vimtutor”, which is basically small program allowing user to practice basic vi commands. To use it, type *vimtutor* and press Enter. Following dialogue appears



The tutorial is self-explanatory and consists of practice examples which can be completed while reading the document simultaneously. Go through the document to complete the exercises as per given instructions.

**How to recover your work if something goes wrong!!!**

The vi editor edits a temporary copy of your file, and after the editing is complete, or when you tell it to save, it puts the contents of the temporary copy into the original file. If something goes wrong while you are editing your file, the vi editor will attempt to save whatever work you had in progress, and store it for later recovery. If you were editing the file, and you accidentally got logged out, then the **-r** option of the vi editor helps. Use the following command to open the file:

**vi -r first**

It will show you the temporary file options for recovery. The **-r** option stands for recovery. After using the **-r** option once, you MUST save what you have recovered to the actual file.

**PART – 2 (I/O REDIRECTION)**

**Indirection of Output**

* Create a file named as ***file1*** using **vi editor**, write the following contents in the ***file1***:

*A programmer for whom computing is its own reward; may enjoy the challenge of breaking into other computers but does no harm; true hacker subscribe to a code of ethics and look down upon crackers.*

* Create another file with a name ***file2*** using **vi editor**, write the following contents in the ***file2***:

*I am a white hat hacker. I have my own code of ethics*

* Execute the following two commands:

***[f2012999@ prithvi ~] $ cat file1 [f2012999@ prithvi ~] $ cat file1 > file3***

Observe the difference.

The first command displays the contents of ***file1*** onto the standard output, i.e. the monitor. Instead, the second command redirected that output to another file, named ***file3***. Observe that this file did not exist but execution of the second command created it. You can look at the contents of ***file3*** using vi editor or cat command.

Now execute the following command:

***[f2012999@ prithvi ~] $ cat file1 > file2***

Look at the contents of ***file2*** and observe that by executing the above command you have declared ***file2*** as the temporary standard output causing the contents of ***file1*** to be redirected to ***file2***. Sincethe file ***file2*** is not empty, it will be overwritten.

To avoid overwriting the contents of ***file2***, instead of using **>** use **>>**. This will append the contents of ***file1*** to the contents of ***file2.*** Check it out yourself.

* Execute the following commands and observe the result

***[f2007999@ prithvi ~] $ cal 1 2010 > cal1***

Look at the contents of ***cal1***

***[f2007999@ prithvi ~] $ cal 2 2010 > cal2***

Look at the contents of ***cal2***

***[f2007999@ prithvi ~] $ cat cal2 >> cal1***

Again look at the contents of ***cal1***. Make a note of your observations

**Indirection of Input**

* Execute the following two command and observe the result:

***[f2012999@ prithvi ~] $ cat [f2012999@ prithvi ~] $ cat file2***

In the first case, the cat command takes its input from standard input device, i.e. keyboard. It displays whatever you type on the keyboard. Press ctrl+c to terminate its execution.

In the second case, the **cat** command takes its input from the file named ***file2*** and sends the result to the standard output (i.e. your monitor). You should note that this command does not take its input from standard input (keyboard); rather it takes the input from a file.

* Execute the following command and observe the result:

***[f2012999@ prithvi ~] $ cat < file2***

Do you find any difference between this command and the previous command? No difference, you are reading the contents of the file ***file2*** and outputting on the standard output (monitor), but the file ***file2*** doesn’t becomestandard input for always, after the execution of the command UNIX willautomatically make the standard input as keyboard.

* Execute the following command

***[f2012999@ prithvi ~] $ cat < cal1 > testCal1***

List the contents of your directory (use ***ls***), you will find a new file ***testCal1****.* Look at the contents of file ***testCal1***.

Now execute the following command

***[f2012999@prithvi ~] $ cat cal2 > testCal2***

List the contents of your directory (use ***ls***), again you will find a new file ***testCal2****.* Look at the contents of file ***testCal2***. So what difference do you find in previous two commands?

* The **indirection operators** and usage

**> filename**- make file with a name ***filename*** as the standard output

**< file** - make file with a name ***filename*** as the standard input

**>> file** - make file with a name ***filename*** as the standard output, append to it if it exists

**Exercises**

**Execute the following commands and record the observations**

****

**Important *–*** In place of whole prompt ***[f2012999@ prithvi ~] $*** only ***$*** symbol is used

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Commands** | |  | **What does the command do?** |
|  | ***$ls >*** | ***filelist*** |  | ???? |
|  |  |  | |  |
|  | ***$ ls –l*** | ***> longFileList*** | | ???? |
|  |  | |  |  |
|  | ***$date; cal 2*** | | ***2010*** | ???? |
|  |  | | |  |
|  | ***$( date ; ls ) > complex*** | | | ???? |
|  |  |  |  |  |
|  | ***$date;*** | ***cal 2*** | ***2010; cal 3 10*** | ???? |
|  |  |  |  |  |

**PART – 3 (SOME MORE UNIX COMMANDS)**

1. **wc (word count)**

This command is used to count the number of lines, words and characters in a given file. Execute the command as shown below and observe the result.

***[f2012999@prithvi ~] $wc file2***

You will get the following output:

***1 35 197 file2*** (This means the file ***file2*** has 1 lines, 35 words and 197 characters)

This command can use options like –l, -w, -c to get the number of lines, words characters individually.

Execute the following commands and observe the result:

***[f2012999@prithvi ~] $wc –l file2 [f2012999@prithvi ~] $wc –c file2 [f2012999@prithvi ~] $wc –w file2***

1. **sort**

The **sort** program arranges lines of text alphabetically or numerically. Create a file with a name

***food.txt*** and write the following lines into that file:Afghani Cuisine

Bangkok Wok

Big Apple Deli Isle of Java Mandalay

Execute the following command and observe the result

**[f2012999@prithvi:~]$**sort food.txt

**sort** doesn't modify the file itself; it reads the file and writes the sorted text to the standard output

Now redirect the output of **sort** command to a file ***sortedFood.txt***, execute the following command:

**[f2012999@prithvi:~]$**sort food.txt > sortedFood.txtSee the contents of file **sortedFood.txt**

There are many options that control the sorting. **Use man command to see these options.** Two of these options are given below:

|  |  |
| --- | --- |
| **Option** | **Description** |
|  |  |
| -n | Sort numerically and ignore blanks and tabs |
|  |  |
| -r | Reverse the order of sort |
|  |  |

**Exercise:**

Create a file ***student.txt*** and write the following contents into the file: 2005A7PS634P : DENNIS Peter

2007A3PS222P : Reedam Dwivedi 2008A4PS212G : KIRTI Singh 2007B4A7896G : Deepak SHARMA 2005A2PS343H : Gaurav SINGH

Now look at the contents of file student.txt (use ***cat***)

|  |  |
| --- | --- |
| **Command** | **Observation** |

**$**sort–n student.txt

**$**sort–r student.txt

**$**cat student.txt | grep "200" | sort

**$**cat student.txt | grep "A7" | sort -r

**(c) head**

The head command reads the first few lines of any file given to it as an input and writes them to the display. By default, head returns the first ten lines of each file name that is provided to it. For example, the following will display the first ten lines of the file named ***myFile.txt***

***[f2012999@prithvi ~] $head myFile.txt***

If more than one input file is provided, head will return the first ten lines from each.

Below is an example:

***[f2012999@prithvi ~] $head myFile.txt file1***

You can specify the number of lines that you want to display from the file by using an option ***–n***. This ***-n*** option is used followed by an integer indicating the number of lines desired. For example, the above example could be modified to display the first 2 lines from the file ***myFile.txt.***

***[f2012999@prithvi ~] $head -n2 myFile.txt***

We can have more files in the command, in which case the specified number of lines will be displayed from each file

***[f2012999@prithvi ~] $head –n2 myFile.txt file1***

1. **tail**

The tail command is similar to the head command except that it reads the lines from the end of file.

***[f2012999@prithvi ~] $tail –n3 myFile.txt***

**PART – 4 (PIPES AND FILTERS)**

The indirection operator ( **|** ) is called as pipe symbol, which helps to join two commands. Basically it feeds the output of one command as an input to other command. Two or more commands connected in this way forms a **pipe**.

When a command takes its input from another command, performs some operation on that input, and writes the result to the standard output (which may be piped to yet another program), it is referred to as a **filter**.

Hence the commands that make up the **pipe** are called **filters**. One of the most common uses of filters is to modify output. Just as a common filter cuts unwanted items, the UNIX filters can be used to restructure output. Putting a vertical bar ( | ) on the command line between two commands makes a **pipe**.

When a **pipe** is set up between two commands, the standard output of the command to the left of the pipe symbol becomes the standard input of the command to the right of the pipe symbol.

Solve the following exercise and the concept of pipe will become clearer to you

|  |  |  |
| --- | --- | --- |
| **Commands** |  | **What does the command do?** |
| ***$ ls | wc myFile.txt*** |  |  |
|  |  |  |
| ***$ ls > myFile.txt | wc*** |  |  |
|  |  |  |
| ***$ ls | wc > myFile.txt*** |  |  |
|  |  |  |
| ***$ cat myFile.txt file1 | sort*** |  |  |
|  |  |  |
| ***$ cat file1 | sort | head -1 | wc*** | ***-*** |  |
|  |  |  |

**GREP COMMAND**

Now, let us make the concept of pipes and filters more clear to us. For this, understand one more important command in Unix, i.e. Grep command.

The **grep** command searches a file(s) for lines that contains certain pattern. It can be used in a pipe so that only those lines of the input files containing a given pattern are sent to the standard output.

* Create three file with a name ***gp1.dat***, ***gp2.doc***, and ***gp3.txt***
* Suppose you want to search for files whose name contains these two characters **gp**. Execute the following command and see the output it should come something like what follows:

**[f2012999@prithvi:~]$**ls -l | grep "gp"

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | -rw-r--r-- 1 f2009777 ug2009 | | 0 | 2010-02-04 | 15:18 gp1.dat |  |
|  | -rw-r--r-- 1 f2009777 ug2009 | | 0 | 2010-02-04 | 15:19 gp2.txt |  |
|  | -rw-r--r-- 1 f2009777 ug2009 | | 0 | 2010-02-04 | 15:19 gp3.doc |  |
|  |  |  |  | |  |  |
|  |  |  |  |  |  |  |
|  | **Command** |  | **Observation** | |  |  |
|  |  |  |  |  |  |  |
|  | **$**ls -l | grep "dat" |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | **$**ls -l | grep "txt" |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | **$**ls -l | grep "doc" |  |  |  |  |  |
|  |  |  |  |  |  |  |



*Note: In place of* ***[f2012999@prithvi:~]$*** *only* ***$*** *is used here to save space, so please don’t panic*

* **grep** command has several options which let you modify the search. Below is a list of some of theoptions.

|  |  |
| --- | --- |
| **Option** | **Description** |
|  |  |
| -v | Print all lines that do not match pattern |
|  |  |
| -n | Print the matched line and its line number |
|  |  |
| -c | Print only the count of matching lines |
|  |  |
| -i | Match for both upper or lower case letters |
|  |  |

**To understand** grep **command options in detail solve exercise provided below:**

**$**ls -l | grep–v "gp"

**$**ls -l | grep–n "gp"

**$**ls -l | grep–c "gp"

**$**ls -l | grep–i "GP"

**PART – 5 (PATTERN MATCHING)**

A UNIX ***shell*** is a program that understands the commands given by the user and allows the user to execute commands by typing them manually at a terminal. It also allows the users to write scripts which can execute these commands automatically; such scripts are known as ***shell scripts***. We will cover shell scripts in next lab.

Anything written to the right of the UNIX command is an argument passed to that command i.e. filenames or a piped command.

Look at the following examples:

* **$ls main.c**

In this example **main.c** is the filename which goes as an argument to **ls** which does a long listing for this file name if at all it is present in the working directory.

The shell provides a mechanism for generating a list of file names that match a pattern. For example,

 **$ls gp\***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ---rw-r | --r | 1 f2009777 ug2009 | | | 0 2010-02-04 15:18 gp1.dat | | | |
| -rw-r-- | r-- | 1 | f2009777 | ug2009 | 0 | 2010-02-04 | 15:19 | gp2.txt |
| -rw-r-- | r-- | 1 | f2009777 | ug2009 | 0 | 2010-02-04 | 15:19 | gp3.doc |
|  |  |  |  |  |  |  |  |  |



This command does a listing for all the file names in the current directory that start with **gp***.* The character **\*** also know a **wildcard** finds all the string patterns that are prefixed/suffixed with it.

*Note: If there is a directory with a name* ***gprog (i.e. directory name starts with gp)*** *then above command will display its contents.*

 **$ls \*.txt**

****-rw-r--r-- 1 f2009777 ug2009 0 2010-02-04 15:19 gp2.txt

This command does a listing for all the file names in the current directory which ends with **.txt** extension.



The file name in UNIX has usually two parts separated by a dot (**.**) called as first name and extension. Let us consider a file name as ***filen.ext.*** Here ***filen*** is the first name of the file and ***ext*** is the extension which tells about the type of the file. Say **txt** (for text file), **doc** (document file), **c**

Following are the special characters interpreted by the shell called as *Wild cards*

|  |  |
| --- | --- |
| **Wildcards** | **Meaning** |
|  |  |
| \* | Matches any number of characters including null character |
|  |  |
| ? | Matches a single character |
|  |  |
| [ijk] | Matches a single character either i,j or k. |
|  |  |
| [!ijk] | Matches a single character which is neither of i, j or k |
|  |  |
| [x-z] | Matches for a single character within this ASCII range of characters |
|  |  |
| [!x-z] | Matches for a single character not within this ASCII range of |
|  | characters |

**(i) \* stands for zero or more characters.**

**$ls gp** (outputs file with a name **gp**, if exists)

**$ls gp\*** (outputs all the files with a name that start with **gp**) **$ls \*** (outputs all the files in the working directory)

**(ii) ? matches a single character**

**$ls a?t**

Matches all those files of three characters with **a** as the first and **t** as the third character (and any character in between)

**$ls ?le**

Matches all three character files and those that end with **le**)

**(iii) [ijk] matches a single character out of i, j, or k. $ls [ijk].\***

Matches all those files of which has first character as i, or, k.

**Exercise:** What will be output of the following commands?

|  |  |
| --- | --- |
| **Command** | **Observation** |
|  |  |
| **ls ??i\*** | Displays all files with name starting with any first two characters, i |
|  | as the third character, and followed by any number of characters |
| **ls ?** |  |
|  |  |
| **ls \*?** |  |
|  |  |
| **ls ?\*** |  |
|  |  |
| **ls “\*”** |  |
|  |  |

**QUOTING**

Characters that have a special meaning to the shell, such as **< > \* ? | &** ( ) are called **metacharacters**. Any character preceded by a **\** is *quoted* and loses its special meaning, if any. The **\** is elided so that

$ **echo \?** will echo a single **?,** and

$ **echo \\** will echo a single **\.**

The **\** is convenient for quoting single character. When more than one character needs quoting, the above mechanism is clumsy and error prone.

A string of characters may be quoted by enclosing the string between single quotes. For example, $ **echo xx'\*\*\*\*'xx ;** *Is equivalent to $ echo xx\\*\\*\\*\\*xx*

will echo

**xx\*\*\*\*xx**