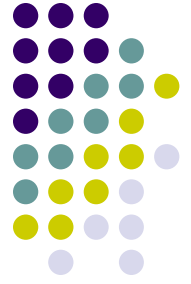




Linux / Unix

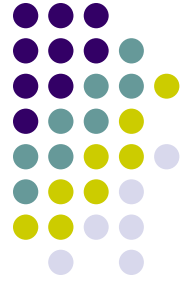
Date: 15 -10 -2010

Introduction

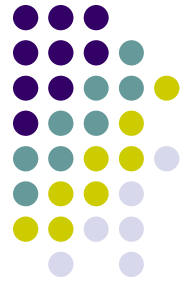


- Linus Torvalds – Creator of Linux
- Open Source Operating System
- Free Software
- Source Code Available
- Kernel can be customized to user's needs

File structure



- /root , /home/users → Home directories
- /bin , /usr/bin , /usr/local/bin → user executables
- /media , /mnt → mount points
- /etc → configuration files
- /tmp → Temporary files
- /boot → Kernel , boot loaders
- /var , /srv, /usr → server data
- /proc , /sys → system information
- /lib, /lib64, /usr/lib , /usr/local/lib → shared libraries
- More info: http://www.comptechdoc.org/os/linux/commands/linux_crfilest.html

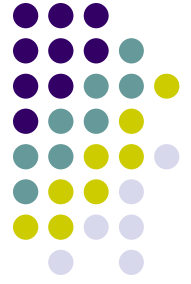


File system commands

- `pwd` - report your current directory
- `cd <to where>` - change your current directory
- `ls <directory>` -list contents of directory
- `cp <old file> <new file>` - copy
- `mv <old file> <new file>` - move (or rename)
- `rm <file>` -delete a file
- `mkdir <new directory name>` -make a directory
- `rmdir <directory>` -remove an empty directory

\$ `man command` gives you help on that command.

Getting Recursive



- remove a directory and its contents:

```
$ rm -r <directory>
```

- copy a directory and its contents:

```
$ cp -r <directory>
```

File permissions.



- There are 3 kinds of users in linux : you (**user**), your friends (**group**) and everyone else (**others**).

r - Read permissions

w - Write permissions

x - execute permissions

d - Directory

- File

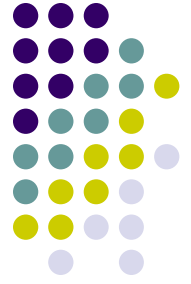
\$ ls -l

-rwxrw-r-- 1 santoshk santoshk 224 Oct 14 17:57 display_time.sh

drwxrwxr-x 2 santoshk santoshk 4096 Oct 14 19:19 test_dir

- For a file if x is set that user can execute the file
- For a directory if x is set that user can that user can enter in that directory.

Changing File Permissions and Ownership



- Make a file readable to your friends:

```
$ chmod 765 <filename>
```

```
7 -> 111 -> rwx
```

```
6 -> 110 -> rw-
```

```
5 -> 101 -> r-x
```

```
-rwx rw- r-x 1 santoshk santoshk 224 Oct 14 17:57 <filename>
```

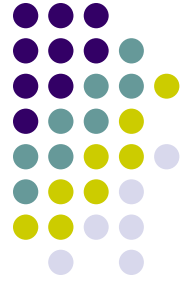
- Change who owns a file:

```
$ chown <user> <filename>
```

- Change to which group the file belongs:

```
$ chgrp <group> <filename>
```

touch



- Look at the full listing again:

```
⌘ ls -l .forward
```

```
-rw-r--r--  1 darin  csua  23 Jan 23  2009 .forward
```

- Each file has a date stamp of when it was modified.
- Use touch to set the timestamp to the current clock.

```
⌘ touch <filename>
```
- Touch creates the file if it didn't exist.
- You can only touch a file to which you can write.

Symbolic Links



- Reference to another file or directory
- use `ln -s <old file> <second name>` to create a symbolic link to a file.

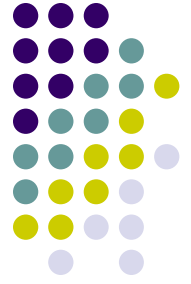
```
$ ln -s nfs.txt link.txt
```

```
$ ls -l
```

```
-rw-rw-r-- 1 santoshk santoshk 26823 Oct 14 19:01 nfs.txt  
lrwxrwxrwx 1 santoshk santoshk    7 Oct 14 19:54 link.txt -> nfs.txt
```

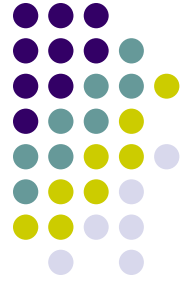
- The first “l” tells you that it’s a symbolic link.
- Symbolic links can be used as if it were its target.

Working on multiple files



- some commands can work on many files at once:
\$ **rm file1 file2 file27**
- Use * to match any number of unknown characters
\$ **rm file***
- Use ? to match one unknown character.
\$ **rm file?**

(un)aliasing



- create shortcuts for yourself

```
$ alias ll='ls -la'
```

- Use alias with no arguments to discover current aliases

```
$ alias
```

```
alias rm='rm -l'
```

```
alias ll='ls -l --color=tty'
```

Type “**unalias rm**” to remove alias.

PATH: a very important shell variable

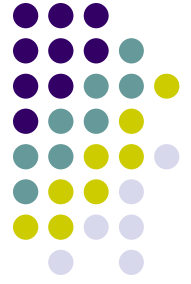


```
$ echo $PATH
```

```
/usr/lib/qt-s.3/bin :/usr/kerberos/bin :/usr/local/bin: /bin:/usr/bin  
:/home/webteam/santoshk/bin
```

- If a program (like `ls`) is in one directory found in your path, then typing it (`~>ls <enter>`) will execute it.
- Otherwise you can type the full absolute address to execute a program (`~>/usr/bin/ls <enter>`)

Finding things in your PATH.



- Type “which <command>” to find the location of the program which would run when you type <command>.

```
$ which grep
```

```
/bin/grep
```

- If you don't remember a command name if it was grep or grepdiff, type “gre<TAB>” to get a list of commands that starts with gre.

```
grefer      grep-changelog  grepjar
```

```
grep       grepdiff
```

- when all else fails, use “find” to find a file.

```
$ find <start dir> -name “*.txt”
```

Other useful pre-defined shell variables



- **HOSTNAME** Name of the computer
- **HOME** Home directory of the user
- **USER** your user login
- **PWD** current directory
- **PATH** defines list of directories to search through when looking for a command to execute.

```
$ echo $HOSTNAME
```

```
cc1.tifr.res.in
```

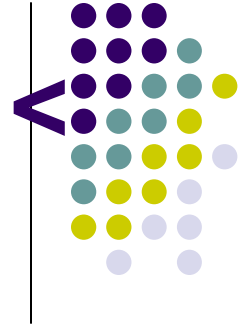
Commands to see all the variables: **env**, **set**

Redirect output to a file with >



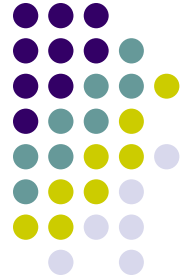
- If you type `who` at the prompt, you will get a list of who is logged into the system.
- If you type `who >f`, a file named `f` will be created and the standard output of `who` will be placed in that file instead of to your screen.
- By default, `who >f` will overwrite the file `f`.
- Use `who >>f` to append to `f` rather than overwriting it.

redirecting input from a file with



- The program `sort` will sort its standard input and then print it on standard out.
- To sort the lines of file1 and display:
`sort < file1`
- To sort the lines of file1 and save in file2:
`sort < file1 > file2`

Piping in unix |



- The output of a command can be piped to another command for further processing

```
$ ls -l | wc -l
```

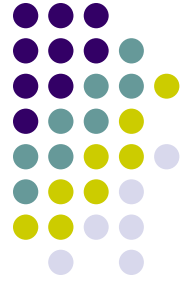
```
$ cat nfs.txt | more
```



shell and shell scripts.

- shell :- A shell is a piece of software that provides an interface for users of an operating system which provides access to the services of a kernel.
 - To see current shell `$ echo $SHELL`
 - To change or use different shell `$ /bin/sh` or `/bin/bash`
- shell script :- Bunch of commands you'd like to automate. You can put them on separate lines of a file. Then type "`shell_name <filename>`" to run the script.
 - `$ sh myscript.sh`
- To make a script executable without giving shell name, the script should have executable file permissions and first line of script should be `#!/<path/shell name>`
 - `$./myscript.sh` or `$ path/myscript.sh`

Simple shell script



```
#!/bin/sh
```

```
#Script to display date and time after every one second
```

```
#alias DSTAMP='date "\"+%d/%b/%Y %H:%M:%S\"'
```

```
alias DSTAMP='date'
```

```
for N in `seq 1 8`
```

```
do
```

```
    echo "Count $N: Now Date and Time is $(DSTAMP)"
```

```
    sleep 1
```

```
done
```

Copy to remote machine : scp



- copy local to remote

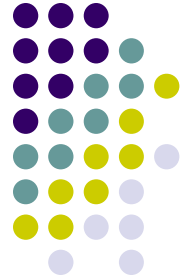
```
⌘ scp <source file> user@machine:<path>
```

- copy remote to local

```
⌘ scp user@machine:<path> <source file>
```

- p Preserves mode, time stamps
- r Recursively copy entire directories.
- v Verbose mode.

Login using ssh



- ssh – remote login program

```
$ ssh -l santoshk cc1.tifr.res.in
```

ssh client in windows is putty. Download from
<http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe>



Date and Time : date

- **date** command prints or sets the system date and time

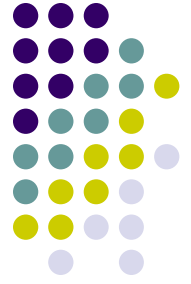
\$ date

Wed Oct 13 17:23:56 IST 2010

\$ date '+%d/%b/%Y %H:%M:%S'

13/Oct/2010 17:22:01

Pattern extraction : grep



- grep is global / regular expression / print

```
$ grep <pattern> <filename>
```

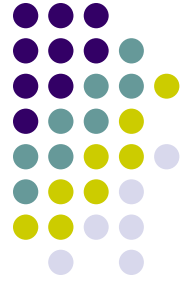
```
$ grep apple fruitlist.txt
```

```
$ grep -i apple fruitlist.txt
```

-i Ignore case

-v Invert the sense of matching

Cutting the fields in a text file



- Cut is for extraction of line segments

```
$ cut -f 2,3 <filename>
```

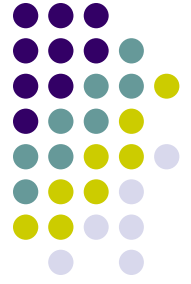
```
$ cut -f 2,3 -d ":" <filename>
```

- awk is for processing text-based data

```
$ awk {'print $2,$5'} <filename>
```

```
$ awk -F":" {' print $2,$5'} /etc/passwd
```


Stream editor : sed



- Sed utility parses text files and can apply textual transformations
- special editor for modifying files automatically

```
$ sed -n '/Start_pattern/,/Stop_pattern /p' <filename>
```

```
$ sed -n '/<!--/,/-->/!p' test2.html
```

```
$ sed 's!Santosh Kyadari!Anil Naik!ig' <filename>
```

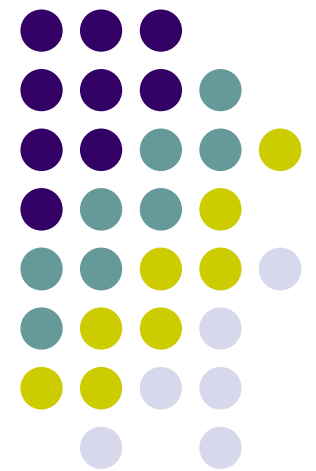
```
$ sed -i 's!Santosh Kyadari!Anil Naik!ig' <filename>
```



More commands

- `sort <filename>` - sort lines of text files
- `uniq <filename>` - report uniq lines
- `tee` - read from standard input and write to standard output and files
- `tar` – backup / archiving utility
- `head` - output the first part of files
- `tail` - output the last part of files
- `cat` - concatenate files and print on the standard output
- `more` – view the contents of a text file one screen at a time
- `echo` - display a line of text

vi editor

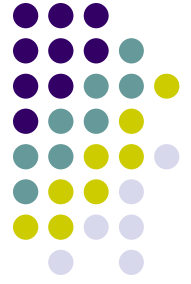


Introduction



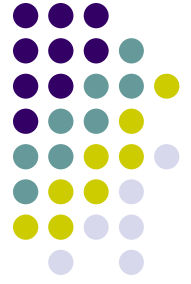
- **vi** is text editor
- Original vi program was written by Bill Joy in 1976
- Use vi editor to:
 - create text files
 - edit text files
- The vi editor is not a text formatter like MS Word
- The current iteration of **vi** for Linux is called **vim**
Vi Improved

Starting vi



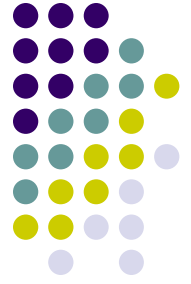
- Type **vi** **<filename>** at the shell prompt
- After pressing enter the command prompt disappears and you see tilde(**~**) characters on all the lines
- These tilde characters indicate that the line is blank

Vi modes



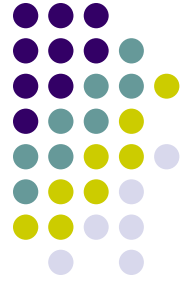
- There are two modes in vi
 - Command mode
 - Input mode
- When you start vi by default it is in command mode
- You enter the input mode through various commands
- You exit the input mode by pressing the Esc key to get back to the command mode

How to exit from vi



- First go to command mode
 - press **Esc** There is no harm in pressing **Esc** even if you are in command mode. Your terminal will just beep and/or or flash if you press **Esc** in command mode
- There are different ways to exit when you are in the command mode

How to exit from vi (command mode)



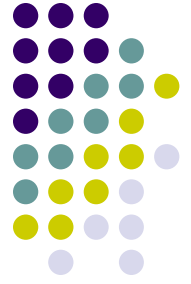
- **:q** <enter> is to exit, if you have not made any changes to the file
- **:q!** <enter> is the forced quit, it will discard the changes and quit
- **:wq** <enter> is for save and Exit
- **:x** <enter> is same as above command
- The **!** Character forces over writes, etc.
:wq!

Moving Around



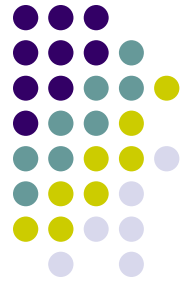
- You can move around only when you are in the command mode
- Arrow keys usually works (but may not)
- The standard keys for moving cursor are:
 - **h** - for left
 - **l** - for right
 - **j** - for down
 - **k** - for up

Moving Around



- **w** - to move one word forward
- **b** - to move one word backward
- **\$** - takes you to the end of line
- **<enter>** takes the cursor to the beginning of next line

Moving Around



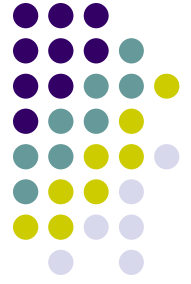
- - - (minus) moves the cursor to the first character in the current line
- **H** - takes the cursor to the beginning of the current screen(Home position)
- **L** - moves to the Lower last line
- **M** - moves to the middle line on the current screen

Moving Around



- **f** - (find) is used to move cursor to a particular character on the current line
 - For example, **fa** moves the cursor from the current position to next occurrence of 'a'
- **F** - finds in the reverse direction

Moving Around



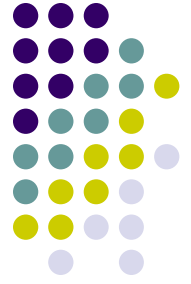
- **)** - moves cursor to the next sentence
- **}** - move the cursor to the beginning of next paragraph
- **(** - moves the cursor backward to the beginning of the current sentence
- **{** - moves the cursor backward to the beginning of the current paragraph

Moving Around



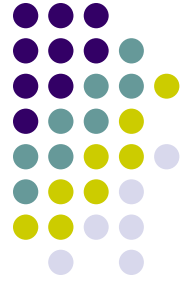
- **Control-d** scrolls the screen down (half screen)
- **Control-u** scrolls the screen up (half screen)
- **Control-f** scrolls the screen forward (full screen)
- **Control-b** scrolls the screen backward (full screen).
- **xG-** to go at x line
- **G-** takes you to bottom line of file
- **gg-** takes you to first line

Entering text



- To enter the text in vi you should first switch to **input mode**
 - To switch to input mode there are several different commands
 - **a** - Append mode places the insertion point after the current character
 - **i** - Insert mode places the insertion point before the current character

Entering text



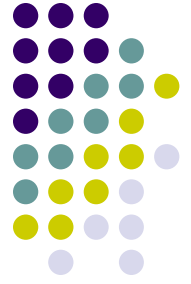
- **I** - places the insertion point at the beginning of current line
- **o** - is for open mode and places the insertion point after the current line
- **O** - places the insertion point before the current line
- **R** - starts the replace (overwrite) mode



Editing text

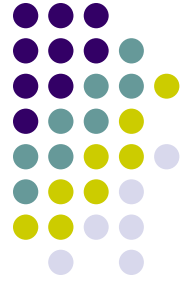
- **x** - deletes the current character
- **d** - is the delete command but pressing only d will not delete anything you need to press a second key
 - **dw** - deletes to end of word
 - **dd** - deletes the current line
 - **d0** - deletes to beginning of line

The change command



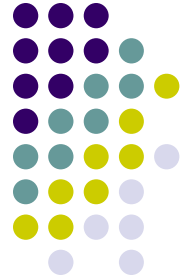
- **c** - this command deletes the text specified and changes the vi to input mode. Once finished typing you should press **<Esc>** to go back to command mode
- **cw** - Change to end of word
- **cc** - Change the current line
- There are many more options

Structure of vi command



- The vi commands can be used followed by a number such as **n<command key(s)>**
 - For example **dd** deletes a line **5dd** will delete five lines.
- This applies to almost all vi commands
- This how you can accidentally insert a number of characters into your document

Undo and repeat command



- **u** - undo the changes made by editing commands
- **.** (dot or period) repeats the last edit command



Copy, cut and paste

- **yy** - (yank) copy current line to buffer
- **nyy** - Where **n** is number of lines
- **p** - Paste the yanked lines from buffer to the line below
- **P** - Paste the yanked lines from buffer to the line above

(the paste commands will also work after the **dd** or **n dd** command)



vi Tricks

- Indent four lines: `4>>`
- Will delete the character under the cursor, and put it afterwards. In other words, it swaps the location of two characters: `xp`
- Similar to `xp`, but swapping lines: `ddp`

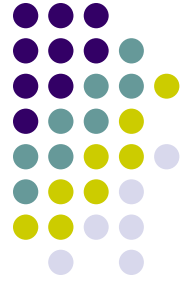
Creating a shell script using vi



- Create a directory **class**
- Change into **class**
- **vi myscript.sh**
- inside the file enter following commands

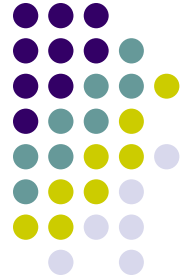
```
clear
echo "======"
echo "Hello World"
echo "======"
sleep 3
clear
echo Host is $HOSTNAME
echo User is $USER
```

Creating a shell script using vi



- Save the file
- Change the permissions on myscript.sh
chmod 700 myscript.sh <enter>
- Now execute myscript.sh
myscript.sh <enter>
- Did the script run?
- Why not?
 - Hint, think about absolute vs relative path
 - Type **echo \$PATH** to see your PATH variable
 - Try this **./myscript.sh** <enter>
 - The **./** mean right here in this directory!

References



- Unix shell programming -by Yashwant Kanetkar
- Unix Concepts and Applications –by Sumitabha Das
- <http://www.grymoire.com/Unix/Sed.html>
- <http://www.grymoire.com/Unix/Awk.html>
- <http://www.grymoire.com/Unix/Quote.html>
- <http://www.grymoire.com/Unix/Find.html>