LEHIGH UNIVERSITY

# Introduction to Linux

## File Permission, Process Management & Editors

Alexander B. Pacheco
LTS Research Computing
September 22, 2015

# Outline

## Relative & Absolute Path

- *Path* means a position in the directory tree.
- You can use either the *relative path* or *absolute path*
- In *relative path* expression
  - . (one dot or period) is the current working directory
  - .. (two dots or periods) is one directory up
  - You can combine . and .. to navigate the file system hierarchy.
  - the path is not defined uniquely and does depend on the current path.
  - `../../tmp` is unique only if your current working directory is your home directory.
- In *absolute path* expression
  - the path is defined uniquely and does not depend on the current path
  - `/tmp` is unique since /tmp is the *abolute path*

## Variables

## Variables I

- *nix also permits the use of variables, similar to any programming language such as C, C++, Fortran etc
- A variable is a named object that contains data used by one or more applications.
- There are two types of variables, Environment and User Defined and can contain a number, character or a string of characters.
- Environment Variables provides a simple way to share configuration settings between multiple applications and processes in Linux.
- By Convention, enviromental variables are often named using all uppercase letters

e.g. `PATH, LD_LIBRARY_PATH, LD_INCLUDE_PATH, TEXINPUTS,` etc

- To reference a variable (environment or user defined) prepend $ to the name of the variable

e.g. `$PATH, $LD_LIBRARY_PATH`

## Variables II

- The command `printenv` list the current environmental variables.

★ Type `printenv` on your command prompt to list all environment variables in your current session.

- The command `env` is used to either print a list of environment variables or run another utility in an altered environment without having to modify the currently existing environment.

★ Type `env SHELL=/bin/tcsh xterm` to start an xterm session in tcsh

♦ To execute the above command successfully, you need to be in GUI mode on the virtual OS or logged into a remote systems with X-Forwarding enabled.

## Variables III

|  |  |
|---|---|
| PATH: | A list of directory paths. |
| HOME: | indicate where a user's home directory is located in the file system. |
| PWD: | contains path to current working directory. |
| OLDPWD: | contains path to previous working directory. |
| TERM: | specifies the type of computer terminal or terminal emulator being used |
| SHELL: | contains name of the running, interactive shell. |
| PS1: | default command prompt |
| PS2: | secondary command prompt |
| LD_LIBRARY_PATH: | colon-separated set of directories where libraries should be searched for first |
| HOSTNAME: | The systems host name |
| USER: | Current logged in user's name |
| DISPLAY: | Network name of the X11 display to connect to, if available. |

## Variables IV

- You can edit the environment variables.
- Command to do this depends on the shell

★ To add your bin directory to the `PATH` variable

  `sh/ksh/bash`: `export PATH=${HOME}/bin:${PATH}`

  `csh/tcsh`: `setenv PATH ${HOME}/bin:${PATH}`

★ Note the syntax for the above commands

★ `sh/ksh/bash:` no spaces except between `export` and `PATH`

★ `csh,tcsh:` no = sign, just a space between `PATH` and the absolute path

★ `all shells:` colon(:) to separate different paths and the variable that is appended to

- **Yes, the order matters.** If you have a customized version of a software say perl in your home directory, if you append the perl path to `PATH` at the end, your program will use the system wide perl not your locally installed version.

## Variables V

- Rules for Variable Names
  1. Variable names must start with a letter or underscore
  2. Number can be used anywhere else
  3. DO NOT USE special characters such as `@`, `#`, `%`, `$`
  4. Case sensitive
  5. Examples
     - Allowed: VARIABLE, VAR1234able, var_name, _VAR
     - Not Allowed: 1VARIABLE, %NAME, $myvar, VAR@NAME
- Assigning value to a variable

| Type | sh,ksh,bash | csh,tcsh |
|---|---|---|
| Shell | name=value | set name = value |
| Environment | export name=value | setenv name value |

- `sh,ksh,bash` THERE IS NO SPACE ON EITHER SIDE OF =
- `csh,tcsh` space on either side of = is allowed for the `set` command
- `csh,tcsh` There is no = in the `setenv` command

## Variables VI

### Exercise

- Create two shell variables containing
  1. your name
     e.g. MYNAME=Alex
  2. a standard greeting
     e.g. Greet=Hello
- We'll make use of this variables in a few slides when we learn some basic commands.

## Basic Commands

# Basic Commands

## What is a command and how do you use it?

- **command** is a directive to a computer program acting as an interpreter of some kind, in order to perform a specific task.
- **command prompt** (or just **prompt**) is a sequence of (one or more) characters used in a command-line interface to indicate readiness to accept commands.
- Its intent is to literally prompt the user to take action.
- A prompt usually ends with one of the characters $, %, #, :, > and often includes other information, such as the path of the current working directory.

★ Virtual Image: `[user@localhost ~]$`

★ Mac OSX in `tcsh`: `[c8-bc-c8-ee-b8-9e:~] apacheco%`

- Each **command** consists of three parts: name, options, arguments

```
[user@localhost ~]$ command options arguments
```

## How to get more information with Linux

- `man` shows the manual for a command or program.
- The manual is a file that shows you how to use the command and list the different options for the command in question.
- Usage: `man [command]`
- Example: `man ls` Enter
- `apropos` shows you all of the man pages that may shed some light on a certain command.
- Usage: `appropos [keyword]`
- Example: `appropos editor` Enter

## Input & Output Commands I

- The basis I/O statements are `echo` for displaying output to screen and `read` for reading input from screen/keyboard/prompt
- The `read` statement takes all characters typed until the Enter key is pressed and stores them into a variable.
- Usage: `read <variable name>`
- Example: `read name` Enter

  *Alex Pacheco* Enter
- In the above example, the name that you enter in stored in the variable `name`.

- The `echo arguments` command will print `arguments` to screen or standard output.
- `arguments` can be a (single or multiple) variable, string of characters or numbers.

## Input & Output Commands II

- Examples:
  1. `echo $LD_LIBRARY_PATH $LD_INCLUDE_PATH` Enter
  2. `echo Welcome to HPC    Training` Enter
- By default, `echo` eliminates redundant whitespace (multiple spaces and tabs) and replaces it with a single whitespace between arguments.
- To include redundant whitespace, enclose the arguments within double quotes

e.g. `echo "Welcome to HPC    Training"` Enter

## Input & Output Commands III

### Exercise

- Print out the variable you created a few slides back

  `echo $MYNAME` Enter

  `echo $Greet` Enter

- Read a variable for greeting message

  `read message` Enter

  *`Welcome to HPC`* Enter

- Combine and print your name, the greeting and the message

  `echo $Greet $MYNAME $message` Enter

- What is the output of the following command?

  `echo $Greet $MYNAME, $message Training` Enter

## Commands: pwd & cd

- `pwd` command prints the current working directory.
- Usage: `pwd`
- Example: `pwd` Enter

- `cd` command allows one to change directory
- argument is the path (relative or absolute) of the directory you want to change to
- Usage: `cd [destination]`
- Example: `cd /tmp` Enter
- The default destination directory is your home directory.
- i.e. If you type `cd` Enter , you will end up in your home directory.
- If you want to go back to the previous directory, type `cd -` Enter

## Command: ls

- `ls` command lists the contents of a directory.
- Usage: `ls <options> <path>`
- Example: `ls` Enter
- The current working directory is the default path.
- To list contents of another directory specify the path, relative or absolute
- Common options to the `ls` command

  `-l`: show long listing format

  `-a`: show hidden files

  `-r`: reverse order while sorting

  `-t`: show modification times

  `-h`: use file sizes in SI units (bytes, kilobytes, megabytes etc ) default is bytes

## Command: alias

- `alias` is a command to create a shortcut to another command or name to execute a long string.
- Usage

  `bash/sh/ksh`: `alias <name>="<actual command>"`

  `csh/tcsh`: `alias <name> "<actual command>"`
- Example:

  `bash/sh/ksh`: `alias lla="ls -al"`

  `csh/tcsh`: `alias lls "ls -al"`
- The `alias` command is very useful tool to create shortcuts to other commands and is most often used by paranoid users to prevent accidental deletion of files.
- `unalias` is a command to remove an alias.
- Usage: `unalias <name>`
- Example: `unalias lla` will remove the shortcut to `ls -al`

## Command: mkdir

- `mkdir` is a command to create a directory
- Usage: `mkdir <options> <directoryname>`
- Example: `mkdir -p $HOME/test/testagain` Enter
- By default, the directory is created in the current directory or in a path relative to the current directory
- The `-p` option will create intermediate directories if they do not exist.

e.g. If the directory `test` does not exist in `$HOME`, then

`mkdir $HOME/test/testagain` will fail.

The `-p` option will create the `test` directory within `$HOME` and then create `testagain` within the newly created `test` directory

## Command: cp

- `cp` is a command to copy a file or directory
- Usage: `cp <options> <source(s)> <destination>`
- Example: `cp $HOME/.bashrc ../../tmp` Enter
- Common options to `cp` command:

  `-r`: copy recursively, required when copying directories.

  `-i`: prompt if file exists on destination and can be copied over.

  `-p`: preserve file access times, ownership etc.
- If there are more than one source files, then the destination (i.e. last entry or file) must be a directory.
- If the source(s) is(are) a file(s) and the destination is a directory, then the file(s) will be copied into the directory

e.g. `cp file1 file2 dir1` Enter

`dir1` will contain the files `file1` and `file2`

If `dir1` is a file, then the above command will fail

## Command: rm

- `rm` command removes or deletes a file or directory
- Usage: `rm <options> <file or directory>`
- Example: `rm $HOME/tmpfile` Enter
- Common options to `rm` command:

  `-r`: remove recursively, required when copying directories.

  `-i`: prompt if file really needs to be deleted

  `-f`: force remove overrides the `-i` option
- BE CAREFUL WHILE USING THE **rm** COMMAND, DELETED FILES CANNOT BE RECOVERED
- To be on the safe side, create an `alias` to the `rm` command and only use the `-f` option only if you are sure you want to delete the file or directory

  `sh/ksh/bash`: `alias rm="rm -i"`

  `csh/tcsh` : `alias rm 'rm -i'`
- delete empty directories using the `rmdir` command.

## Command: mv

- `mv` command moves or renames a file or directory
- Usage: `mv <options> <source> <destination>`
- Example: `mv test test1`
- If there are more than one source file, then the last file is the destination and must be a directory.
- Use the `-i` option to prompt if a file or directory will be overwritten.
- If the source(s) is(are) a file(s) and the destination is a directory, then the file(s) will be copied into the directory.

e.g. `mv file1 file2 dir1` Enter

`dir1` will contain the files `file1` and `file2`

If `dir1` is a file, then the above command will fail

## Pager Commands

- To display a file to screen, *nix provides three commands at your disposal
- `cat`: Show contents of a file.
- `more`: Display contents one page at a time.
- `less`: Display contents one page at a time but allow forward/backward scrolling

  `less > more` or `less is more, more or less`
- Usage: `cat/more/less <options> <filename>`
- Example: `cat .bashrc`
- To scroll forward in `more` or `less`, use the space bar, `CNTRL-f/d` or "Page Down" key.
- To scroll backwards in `less` use `CNTRL-b/u` or "Page Up".
- A rarely used command, `tac` does the opposite of `cat` i.e. show contents of a file in reverse.

## Other Commands I

| | |
|---|---|
| passwd: | change password |
| chsh: | change default shell |
| df: | report disk space usage by filesystem |
| du: | estimate file space usage - space used under a particular directory or files on a file system. |
| sudo: | run command as root (only if you have access) |
| mount: | mount file system (root only) |
| umount: | unmount file system (root only) |
| shutdown: | reboot or turn off machine (root only) |
| top: | Produces an ordered list of running processes |
| free: | Display amount of free and used memory in the system |
| file: | Determine file type |
| touch: | change file timestamps or create file if not present |
| date: | display or set date and time |
| find | : Find a file |

```
find /dir/to/search -name file-to-search
```

## Other Commands II

| | |
|---|---|
| wc: | Count words, lines and characters in a file<br>`wc -l .bashrc` |
| grep: | Find patterns in a file<br>`grep alias .bashrc` |
| awk: | File processing and report generating<br>`awk '{print $1}' file1` |
| sed: | Stream Editor<br>`sed 's/home/HOME/g' .bashrc` |
| set: | manipulate environment variables<br>`set -o emacs` |
| ln: | Link a file to another file<br>`ln -s file1 file2` |
| wait: | wait until all backgrounded jobs have completed |
| which: | shows the full path of (shell) commands |
| whatis: | display manual page descriptions |

## Other Commands III

!name: rerun previously executed command with the same arguments as before, `name <args>`.

Note that you do not always have to type the full command `name`, just the minimum unique characters (no spaces) of `name` need to be entered.

If you had entered two commands `name <args>` and `nbme <args>`, then to rerun `name`, use the command `!na` Enter .

history: display a list of last executed commands. Optional argument `m` will list the last m commands.

All previously executed commands will be listed with a number `n`.

To rerun a command from history which has number `n`, run the command `!n` Enter

To learn more about these commands, type `man command` on the command prompt

## Basic *nix Utilities

## Filename Completion

- Filename or Tab completion is a default feature in `bash` and `tcsh`.
- It allows to a user to automatically complete the file, directory or command name you are typing upto the next unique characters using the TAB key.
- Example: Your home directory contains directories `Desktop`, `Documents` and `Downloads`.

  If you enter the command `ls D` ⇆ , you will be prompted with above the three directory names.

```
[user@localhost ~]$ ls D ⇆
Desktop/ Documents/ Downloads/
[user@localhost ~]$ ls Do ⇆
Documents/ Downloads/
[user@localhost ~]$ ls Do
```

# Wildcards

- *nix shells have the ability to refer to more than one file by name using special characters called Wildcards.
- Wildcards can be used with *nix utilities such as ls, cp, mv, rm, tar and g(un)zip.

| | |
|---|---|
| ? | match a single character |
| * | match zero or more characters |
| [ ] | match list of characters in the list specified |
| [! ] | match characters not in the list specified |

- Examples:
  1. `ls */*`
     list contents of all subdirectories
  2. `cp [a-z]* lower/`
     copy all files with names that begin with lowercase letters to a directory called lower
  3. `cp [!a-z]* upper_digit/`
     copy all files with names that do not begin with lowercase letters to a directory called lower

## How to Login to Remote Systems?

- Most Linux/UNIX systems allow secure shell connections from other systems.

e.g. You need to login using `ssh` to the LTS HPC clusters.

- Usage: `ssh <username>@<remote host>`
- Example: `ssh alp514@polaris.cc.lehigh.edu`
- If your local machine is a UNIX-like system i.e. Linux, Mac OSX, BSD, AIX, Solaris etc and your username on the local machine is the same as that of the remote machine, then

  you can omit the `<username>@` part of the argument.

  i.e. `ssh <remote host>`
- If the remote machine is listening to ssh connections on a non default port (i.e. different from port 22) add `-p <port number>` option

i.e. `ssh -p <port number> <user>@<remote host>`

- If you need to forward the display of an application from the remote system to your local system, add the `-X` option to `ssh`

  Example: `ssh -X alp514@ssh.cc.lehigh.edu`

## File Transfer between two systems I

- `scp` is a command to copy files/directories between two *nix hosts over the SSH protocol.
- Usage:
```
scp <options> <user>@<host>:/path/to/source/file \
    <user>@<host>:/path/to/destination/file/or/directory
```

e.g. You want to copy files between Polaris Cluster and your Linux Desktop/Laptop

```
scp alp514@polaris.cc.lehigh.edu:/home/alp514/octave-tutorial.tar.gz .
```

```
scp -r Public apacheco@polaris.cc.lehigh.edu:~/
```

- You can omit the `<user>@` part of the argument if the username is the same on both systems.
- You can omit the `<user>@<host>:` for your local machine.
- Common options are `-r` and `-p`, same meaning as `cp`.
- add `-P <port number>` option for non default ports.

## File Transfer between two systems II

- `rsync` is another utility that can be used to copy files locally and remotely.
- Usage: `rsync <option> <source> <destination>`
- It is famous for its delta-transfer algorithm

i.e. sending only the differences between the source files and the existing files in the destination.

- Rsync is widely used for backups and mirroring and as an improved copy command for everyday use.
- Common options:

  `-a`: archive mode

  `-r`: recurse into directories

  `-v`: increase verbosity

  `-z`: compress file data during the transfer

  `-u`: skip files that are newer on the receiver

  `-t`: preserve modification times

  `-n`: dry-run, perform a trial run with no changes made
- Example: `rsync -avtzu corona.cc.lehigh.edu:~/* .`
- If you are a user on National Supercomputing resource such as XSEDE, NERSC, OSG, etc, there are other transfer tools such as globus toolkit (gridftp) and bbcp which provide higher bandwidth and parallel file transfers.

## Compressing and Archiving Files I

- Quite often you need to compress and uncompress files to reduce storage usage or bandwidth while transferring files.
- *nix systems have built-in utilities to compress/uncompress files

**Compress**

```
gzip, zip, bzip2
gzip README Enter
```

**Uncompress**

```
gunzip, unzip, bunzip2
gunzip README.gz Enter
```

- Gzipped files have an extension `.gz,.z` or `.Z`
- zipped files have an extension `.Zip` or `.zip`
- Bzipped files have an extension `.bz2, .bz`
- To compress/uncompress files recursively, use the `-r` option.
- To overwrite files while compressing/uncompressing, use the `-f` option.

## Compressing and Archiving Files II

- *nix provides the `tar` package to create and manipulate streaming archive of files.
- Usage: `tar <options> <file> <patterns>`

  `file` is the name of the tar archive file, usually with extension `.tar`

  `patterns` are pathnames for files/directories being archived
- Common options

  `-c`: create an archive file

  `-x`: extract to disk from archive

  `-z`: filter the archive through gzip (adds/requires extension .gz)

  `-j`: filter the archive through bzip2 (adds/requires extension .bz2)

  `-t`: list contents of archive

  `-v`: verbosely list files processed

e.g. `tar -cvzf myhome.tar.gz ${HOME}/*`

- This becomes useful for creating a backup of your files and directories that you can store at some storage facility e.g. external disk

## Redirection

## I/O Redirection

- There are three file descriptors for I/O streams

|  |  |
|---|---|
| STDIN | : Standard Input |
| STDOUT | : Standard Output |
| STDERR | : Standard Error |

- 1 represents STDOUT and 2 represents STDERR
- I/O redirection allows users to connect applications

|  |  |
|---|---|
| < | : connects a file to STDIN of an application |
| > | : connects STDOUT of an application to a file |
| >> | : connects STDOUT of an application by appending to a file |
| \| | : connects the STDOUT of an application to STDIN of another application. |

- Examples:
  1. write STDOUT to file: `ls -l > ls-l.out`
  2. write STDERR to file: `ls -l 2> ls-l.err`
  3. write STDOUT to STDERR: `ls -l 1>&2`
  4. write STDERR to STDOUT: `ls -l 2>&1`
  5. send STDOUT as STDIN: `ls -l | wc -l`

## File Permissions

## File Permissions I

- Since *NIX OS's are designed for multi user environment, it is necessary to restrict access of files to other users on the system.
- In *NIX OS's, you have three types of file permissions
    1. read (r)
    2. write (w)
    3. execute (x)
- for three types of users
    1. user (u)
    2. group (g)
    3. world (o) i.e. everyone else who has access to the system

## File Permissions II

```
[user@localhost ~]$ ls -l
total 44
drwxr-xr-x. 2 user user 4096 Jan 28  2013 Desktop
drwxr-xr-x. 2 user user 4096 Jan 28  2013 Documents
drwxr-xr-x. 2 user user 4096 Jan 28  2013 Downloads
-rwxr-xr-x. 1 user user   32 Sep 11 11:57 hello
drwxr-xr-x. 2 user user 4096 Jan 28  2013 Music
drwxr-xr-x. 2 user user 4096 Jan 28  2013 Pictures
drwxr-xr-x. 2 user user 4096 Jan 28  2013 Public
-rw-rw-r--. 1 user user 3047 Sep 11 11:48 README
drwxr-xr-x. 1 root root 4216 Jan 22 16:17 Shared
drwxr-xr-x. 2 user user 4096 Jan 28  2013 Templates
lrwxrwxrwx. 1 user user    5 Jan 23 08:17 test -> hello
drwxr-xr-x. 2 user user 4096 Jan 28  2013 Videos
[user@localhost ~]$
```

- The first character signifies the type of the file

  `d` for directory

  `l` for symbolic link

  `-` for normal file
- The next three characters of first triad signifies what the owner can do
- The second triad signifies what group member can do

## File Permissions III

- The third triad signifies what everyone else can do

$$d\underbrace{rwx}_{u}\overbrace{r-x}^{g}\underbrace{r-x}_{o}$$

- Read carries a weight of 4
- Write carries a weight of 2
- Execute carries a weight of 1
- The weights are added to give a value of 7 (rwx), 6(rw), 5(rx) or 3(wx) permissions.
- `chmod` is a *NIX command to change permissions on a file

  Usage: `chmod <option> <permissions> <file or directory name>`
- To give user rwx, group rx and world x permission, the command is

  `chmod 751 filename`

## File Permissions IV

- Instead of using numerical permissions you can also use symbolic mode

|  |  |
|---|---|
| u/g/o or a | user/group/world or all i.e. ugo |
| +/- | Add/remove permission |
| r/w/x | read/write/execute |

- Give everyone execute permission:

```
chmod a+x hello.sh
```

```
chmod ugo+x hello.sh
```

- Remove group and world read & write permission:

```
chmod go-rw hello.sh
```

- To change permissions recursively in a directory, use the option `-R` (can also be used in the following two commands)

```
chmod -R 755 ${HOME}/*
```

  What is the permission on ${HOME}?

## File Permissions V

- The `chgrp` command is used to change the group ownership between two groups that you are a member of.

  Usage: `chgrp <option> <new group> <file or directory name>`

- You can use the `chgrp` command to change the ownership of your files from the `users` group to `abc` group.

  Example: `chgrp -R abc collaborative-work-dir`

- The `chown` command is used to change the owner of a file.
- `chown` can only be executed by the superuser, to prevent users simply changing ownership of files that aren't theirs to access.

  Usage: `chown <new owner>[:<group name>] <file or directory name>`

## Process Management

## Processes and Jobs I

- A process is an executing program identified by a unique PID
- ★ To see information about your running processes and their PID and status,

  `ps` Enter
- A process may be in foreground, background or be suspended.
- Processes running in foreground, the command prompt is not returned until the current process has finished executing.
- If a job takes a long time to run, put the job in background in order to obtain the command prompt back to do some other useful work
- There are two ways to send a job into the background:
  1. Add an ampersand `&` to the end of your command to send it into background directly.

     `firefox &` Enter
  2. First suspend the job using Ctrl Z and then type `bg` at the command prompt.
  3. If you type `fg` then the job will run in foreground and you will lose the command prompt.

## Processes and Jobs II

- When a process is running, background or suspended, it will be entered onto a list along with a job number (not PID)

  `jobs` Enter

- To restart a suspended job in foreground or background, type

  `fg %jobnumber` where `jobnumber` is a number greater than 1, or,

  `bg %jobnumber`

- To kill or terminate a process:
  1. Job running in foreground: enter Ctrl `C`
  2. Job whose PID you know
     `kill PID` Enter
  3. Job whose `jobnumber` you know (from `jobs` command)
     `kill %jobnumber` Enter

- The `kill` command can take options specific to UNIX signals
- The most common option is `-9` for the `SIGKILL` signal
- `pstree`: display a tree of processes
- `pkill`: kill process by its name, user name, group name, terminal, UID, EUID, and GID.

## Editors

## File Editing

- The two most commonly used editors on Linux/Unix systems are:
    1. `vi` or `vim` (vi improved)
    2. `emacs`
- `vi/vim` is installed by default on Linux/Unix systems and has only a command line interface (CLI).
- `emacs` has both a CLI and a graphical user interface (GUI).
- If `emacs` GUI is installed then use `emacs -nw` to open file in console.
- Other editors that you may come across on *nix systems

  `kate`: default editor for KDE.

  `gedit`: default text editor for GNOME desktop environment.

  `gvim`: GUI version of `vim`

  `pico`: console based plain text editor

  `nano`: GNU.org clone of `pico`

  `kwrite`: editor by KDE.

## Editor Cheatsheets I

- **vi/vim** and **emacs** are the two most popular *nix file editors.
- Which one to use is up to you.
- **vi/vim** has two modes:
  1. Editing mode
  2. Command mode
- **emacs** has only one mode as in any editor that you use.

| Insert/Appending Text | vi |
|---|---|
| insert at cursor | `i` |
| insert at beginning of line | `I` |
| append after cursor | `a` |
| append at end of line | `A` |
| newline after cursor in insert mode | `o` |
| newline before cursor in insert mode | `O` |
| append at end of line | `ea` |
| exit insert mode | `ESC` |

## Editor Cheatsheets II

| Cursor Movement | vi | emacs |
|---|---|---|
| move left | `h` | `C-b` |
| move down | `j` | `C-n` |
| move up | `k` | `C-p` |
| move right | `l` | `C-f` |
| jump to beginning of line | `^` | `C-a` |
| jump to end of line | `$` | `C-e` |
| goto line `n` | `nG` | `M-x goto-line` $\hookleftarrow$ `n` |
| goto top of file | `1G` | `M-<` |
| goto end of file | `G` | `M->` |
| move one page up | `C-u` | `M-v` |
| move one page down | `C-d` | `C-v` |

C : Control Key

M : Meta or ESCAPE (ESC) Key

$\hookleftarrow$ : Enter Key

## Editor Cheatsheets III

| File Manipulation | vi | emacs |
|---|---|---|
| save file | :w | C-x C-s |
| save file and exit | :wq, ZZ | |
| quit | :q | C-x C-c |
| quit without saving | :q! | |
| delete a line | dd | C-a C-k |
| delete *n* lines | *n*dd | C-a M-*n* C-k |
| paste deleted line after cursor | p | C-y |
| paste before cursor | P | |
| undo edit | u | C-_ |
| delete from cursor to end of line | D | C-k |
| search forward for *patt* | \*patt* | C-s *patt* |
| search backward for *patt* | ?*patt* | C-r *patt* |
| search again forward (backward) | n | C-s(r) |

## Editor Cheatsheets IV

| File Manipulation (contd) | vi | emacs |
|---|---|---|
| replace a character | `r` | |
| join next line to current | `J` | |
| change a line | `cc` | |
| change a word | `cw` | |
| change to end of line | `c$` | |
| delete a character | `x` | `C-d` |
| delete a word | `dw` | `M-d` |
| edit/open file *file* | `:e file` | `C-x C-f` *file* |
| insert file *file* | `:r file` | `C-x i` *file* |
| split window horizontally | `:split` or `C-ws` | `C-x 2` |
| split window vertically | `:vsplit` or `C-wv` | `C-x 3` |
| switch windows | `C-ww` | `C-x o` |

## Editor Cheatsheets V

- Do a google search for more detailed cheatsheets

| | |
|---|---|
| vi | `https://www.google.com/search?q=vi+cheatsheet` |
| emacs | `https://www.google.com/search?q=emacs+cheatsheet` |

### More on the **set -o** command

- The **set -o** command can be used to change the command line editor mode among other things (Do **man set** Enter to find out more)
  1. **set -o emacs**: emacs style in-line editor for command entry, this is the default
  2. **set -o vi**: vi style in-line editor for command entry.

## Start Up Scripts

- When you login to a *NIX computer, shell scripts are automatically loaded depending on your default **shell**
- **sh,ksh**
  1. `/etc/profile`
  2. `$HOME/.profile`
- **bash**
  1. `/etc/profile`, login terminal only
  2. `/etc/bashrc` or `/etc/bash/bashrc`
  3. `$HOME/.bash_profile`, login terminal only
  4. `$HOME/.bashrc`
- **csh,tcsh**
  1. `/etc/csh.cshrc`
  2. `$HOME/.tcshrc`
  3. `$HOME/.cshrc` if .tcshrc is not present
- The `.bashrc, .tcshrc, .cshrc, .bash_profile` are script files where users can define their own aliases, environment variables, modify paths etc.
- e.g. the **alias** command covered earlier can be put in one of these script files depending on your **shell**

## Examples I

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific aliases and functions
alias c="clear"
alias rm="/bin/rm -i"
alias psu="ps -u apacheco"
alias em="emacs -nw"
alias ll="ls -lF"
alias la="ls -al"
export PATH=/home/apacheco/bin:${PATH}
export g09root=/home/apacheco/Software/Gaussian09
export GAUSS_SCRDIR=/home/apacheco/Software/scratch
source $g09root/g09/bsd/g09.profile

export TEXINPUTS=.:/usr/share/texmf//:/home/apacheco/LaTeX//:${TEXINPUTS}
export BIBINPUTS=.:/home/apacheco/TeX//:${BIBINPUTS}
```

## Examples II

```
# .tcshrc

# User specific aliases and functions
alias c clear
alias rm "/bin/rm -i"
alias psu "ps -u apacheco"
alias em "emacs -nw"
alias ll "ls -lF"
alias la "ls -al"
setenv PATH "/home/apacheco/bin:${PATH}"
setenv g09root "/home/apacheco/Software/Gaussian09"
setenv GAUSS_SCRDIR "/home/apacheco/Software/scratch"
source $g09root/g09/bsd/g09.login

setenv TEXINPUTS ".:/usr/share/texmf//:/home/apacheco/LaTeX//:${TEXINPUTS}"
setenv BIBINPUTS ".:/home/apacheco/TeX//:${BIBINPUTS}"
```