

Introduction to Linux

Alexander B. Pacheco
LTS Research Computing
February 19 & 26, 2015



Outline

- 1 What is Linux?
- 2 Variables
- 3 Basic Commands
- 4 Redirection
- 5 File Permissions
- 6 Process Management
- 7 Editors

Installing Linux on VirtualBox

1. Download and install Oracle VirtualBox (and the extension pack) from [here](#)
2. Download the CentOS virtual image from [here](#). (you need to be logged into Lehigh Google to access the image name CentOS.ova. Its about 2.6GB.)
3. Install the image by double clicking on it. If it doesn't work, open virtualbox software that you just installed,
 - 3.1 From the menu, click File>Import Appliance
 - 3.2 Choose the ova file that you just downloaded and click the next button (this instruction may differ on Windows and Mac systems)
 - 3.3 Click the import button
4. Whole process should take a few minutes.
5. Once the process is complete, you should see CentOS listed in the left sidebar.
6. Select CentOS and click the start button or double click CentOS
7. After a minute or two you should see a login prompt
8. Type user and hit enter
9. You should now see a prompt such as `[user@localhost ~]$`
10. Create a password by typing `passwd` and hit enter. You will be prompted to enter a password twice, you will not see any characters on the screen as you type.
11. Create a password for admin user by first logging in as root: type `su -` at the prompt and hit enter. Follow the previous step

Logging into a remote Linux server

► Mac OSX

1. Open the Terminal App
2. At the command prompt enter `ssh user@remotehost`
`user` is your username on the remote Linux server
`remotehost` is the hostname or ip address of the remote Linux server
e.g To log into polaris `ssh alp514@polaris.cc.lehigh.edu`

► Windows

1. Download and install a ssh client such as putty or MobaXterm
2. Open the client
3. Putty: Enter the hostname or ip address of the remote Linux server (make sure the SSH radio button is selected) > Click open
4. MobaXterm: Click New Session > Select SSH tab > Enter hostname and username in the field provided > Click on OK
5. When you are prompted for your password, you may not see any characters on the screen.

What is Linux?

History I

- ▶ Unix was conceived and implemented in 1969 at AT&T Bell labs by Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna.
- ▶ First released in 1971 and was written in assembler.
- ▶ In 1973, Unix was re-written in the programming language C by Dennis Ritchie (with exceptions to the kernel and I/O).
- ▶ The availability of an operating system written in a high-level language allowed easier portability to different computer platforms.
- ▶ The GNU Project, started in 1983 by Richard Stallman, had the goal of creating a “complete Unix-compatible software system” composed entirely of free software.
- ▶ 386BSD released in 1992 and written by Berkeley alumni Lynne Jolitz and William Jolitz. FreeBSD, NetBSD, OpenBSD and NextStep (Mac OSX) descended from this
- ▶ Andrew S. Tanenbaum wrote and released MINIX, an inexpensive minimal Unix-like operating system, designed for education in computer science
- ▶ Frustrated with licensing issues with MINIX, Linus Torvalds, a student at University of Helsinki began working on his own operating system which eventually became the “Linux Kernel”
- ▶ Linus released his kernel for anyone to download and help further development.

Linus's message to comp.os.minix on Aug 26, 1991

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

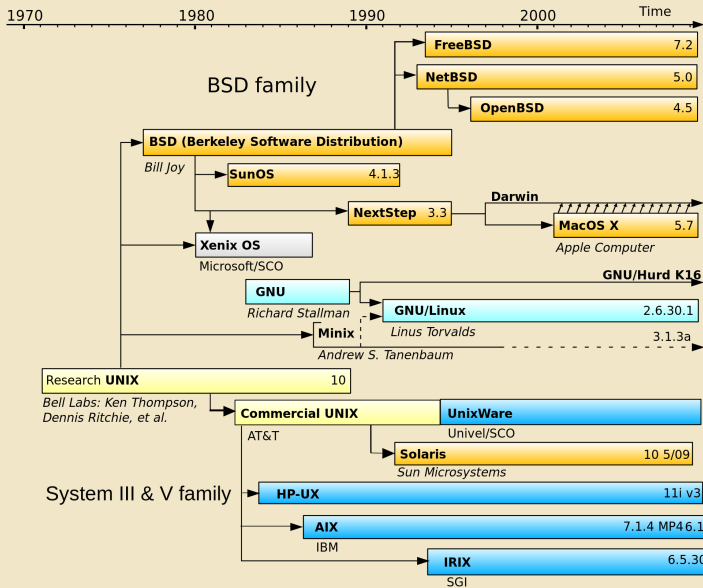
I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (email address)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-).

<https://groups.google.com/forum/?fromgroups=#!msg/comp.os.minix/dlNtH7RRrGA/SwRavCzVE7gJ>

- ▶ Linux is only the kernel, an Operating System also requires applications that users can use.
- ▶ combined with free software available from the GNU project gave birth to a new Operating System known as "GNU/Linux"
- ▶ GNU/Linux or simply Linux is released under the GNU Public License: Free to use, modify and distribute provided you distribute under the GNU Public License.



What is Linux?

- ▶ Linux is an operating system that evolved from a kernel created by Linus Torvalds when he was a student at the University of Helsinki.
- ▶ It's meant to be used as an alternative to other operating systems, Windows, Mac OS, MS-DOS, Solaris and others.
- ▶ Linux is the most popular OS used in a Supercomputer

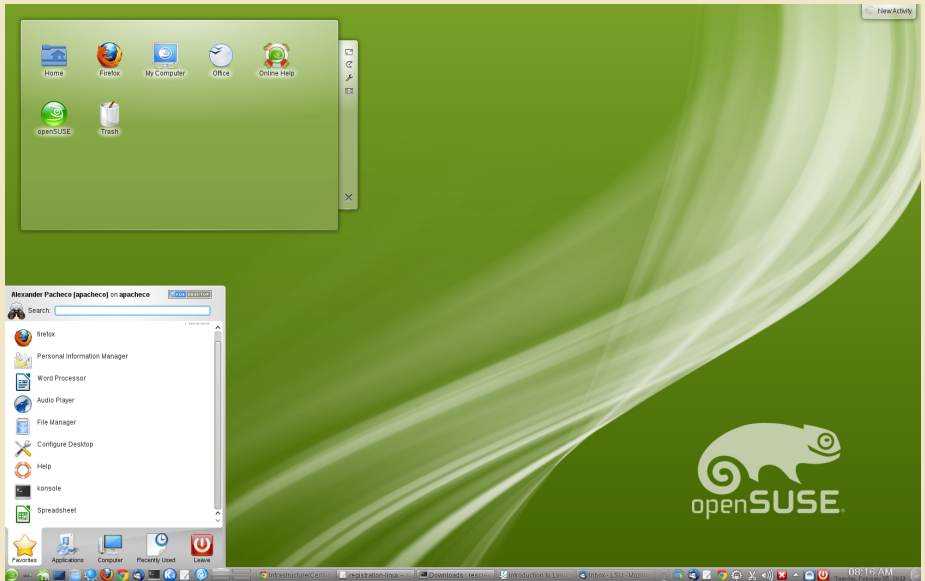
OS Family	Count	Share %
Linux	485	97
Unix	13	2.6
Mixed	1	0.2
Windows	1	0.2

- ▶ If you are using a Supercomputer for your research, it will most likely be based on a *nix OS.

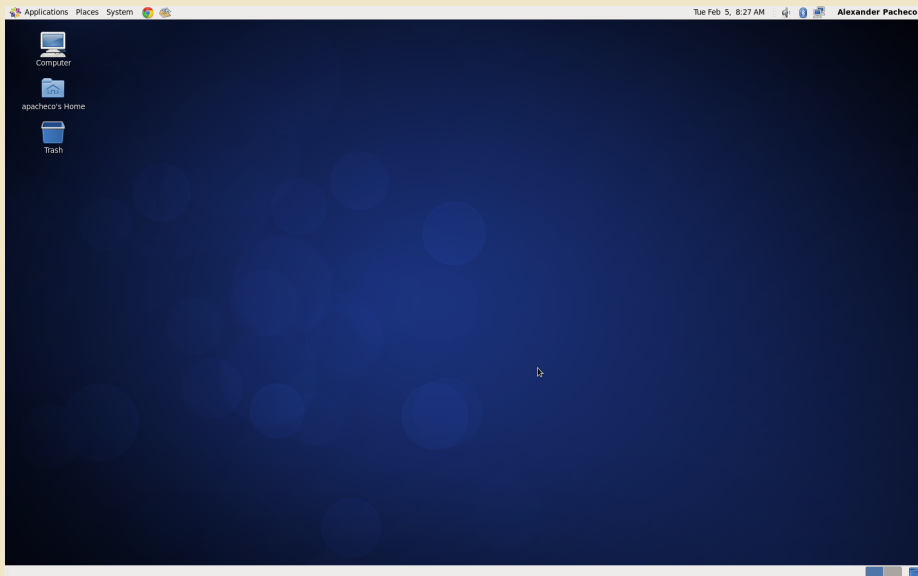
What is Linux?

- ▶ Many software vendors release their own packaged Linux OS (kernel, applications) known as distribution
- ▶ Linux distribution = Linux kernel + GNU system utilities and libraries + Installation scripts + Management utilities etc.
 1. Debian, Ubuntu, Mint
 2. Red Hat, Fedora, CentOS
 3. Slackware, openSUSE, SLES, SLED
 4. Gentoo
- ▶ Application packages on Linux can be installed from source or from customized packages
 1. deb: Debian based distros e.g. Debian, Ubuntu, Mint
 2. rpm: Red Hat based distros, Slackware based distros.
- ▶ Linux distributions offer a variety of desktop environment.
 1. K Desktop Environment (KDE)
 2. GNOME
 3. Xfce
 4. Lightweight X11 Desktop Environment (LXDE)
 5. Cinnamon
 6. MATE

openSUSE KDE Desktop



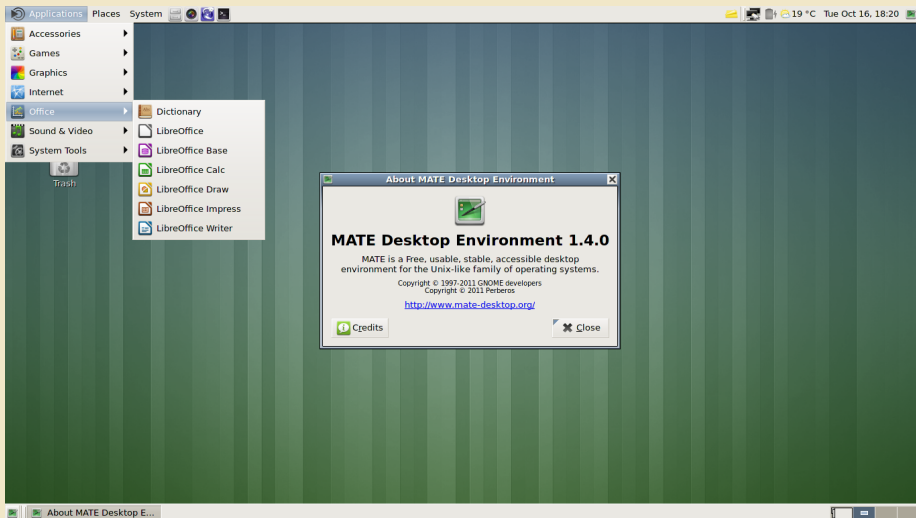
CentOS GNOME Desktop



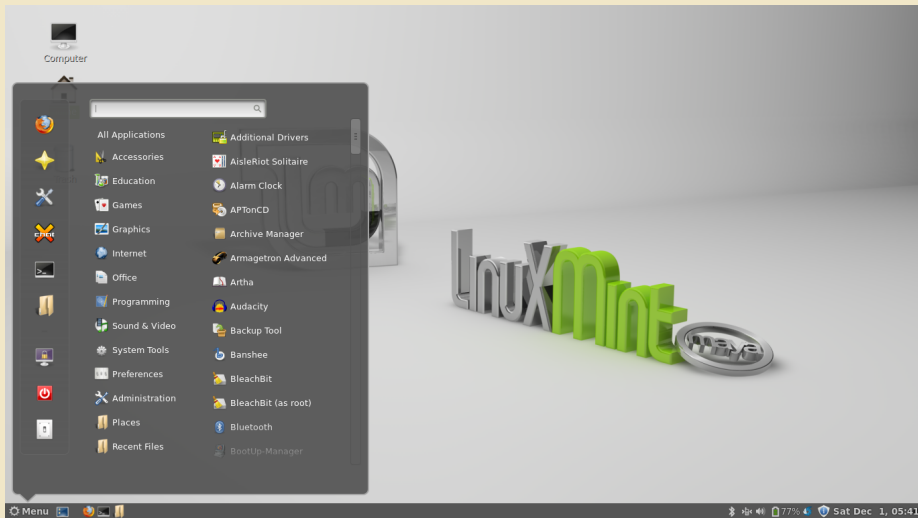
LXDE Desktop



Debian MATE Desktop



Linux Mint Cinnamon Desktop



What is Linux?

- ▶ Linux distributions are tailored to different requirements such as
 1. Server
 2. Desktop
 3. Workstation
 4. Routers
 5. Embedded devices
 6. Mobile devices (Android is a Linux-based OS)
 - ▶ Almost any software that you use on windows has a roughly equivalent software on Linux, most often multiple equivalent software
- e.g. Microsoft Office equivalents are OpenOffice.org, LibreOffice, KOffice
- ▶ For complete list, visit http://wiki.linuxquestions.org/wiki/Linux_software_equivalent_to_Windows_software
 - ▶ Linux offers you freedom, to choose your desktop environment, software.

Popularity of Linux Distributions

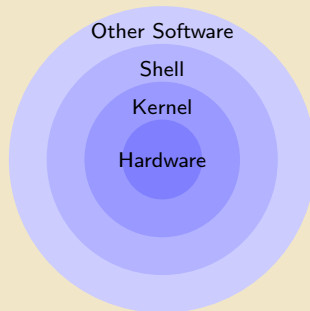
- [DistroWatch](#) provides news, popularity rankings, and other general information about:
 1. various Linux distributions,
 2. free software/open source Unix-like operating systems such as OpenSolaris, MINIX and BSD.
- DistroWatch is NOT an indication of market-share or quality nor is it an indication of how many users but it is clearly an indication of what users are looking at.

Rank	Distribution	Hits	
1	Mint	2427	▲
2	Ubuntu	1830	▼
3	Debian	1597	▲
4	openSUSE	1420	▲
5	Fedora	1322	▲
6	Mageia	1138	▲
7	CentOS	1116	▲
8	Arch	1035	▼
9	elementary	969	▲
10	Android-x86	832	▲

Linux Components I

- ▶ Linux is made up of two (three) parts:

1. Kernel
2. Shell
3. Applications/Programs



Linux Components II

What is a kernel

- ▶ The kernel is the main component of most computer operating systems
- ▶ It is a bridge between applications and the actual data processing done at the hardware level.
- ▶ The kernel's responsibilities include managing the system's resources (the communication between hardware and software components).
- ▶ provides the lowest-level abstraction layer for the resources (especially processors and I/O devices) that application software must control to perform its function.
- ▶ It typically makes these facilities available to application processes through inter-process communication mechanisms and system calls.

What is a SHELL

- ▶ The command line interface is the primary interface to Linux/Unix operating systems.
- ▶ Shells are how command-line interfaces are implemented in Linux/Unix.
- ▶ Each shell has varying capabilities and features and the user should choose the shell that best suits their needs.
- ▶ The shell is simply an application running on top of the kernel and provides a powerful interface to the system.

Types of Shell

sh : Bourne Shell

- ◆ Developed by Stephen Bourne at AT&T Bell Labs

csch : C Shell

- ◆ Developed by Bill Joy at University of California, Berkeley

ksh : Korn Shell

- ◆ Developed by David Korn at AT&T Bell Labs
- ◆ backward-compatible with the Bourne shell and includes many features of the C shell

bash : Bourne Again Shell

- ◆ Developed by Brian Fox for the GNU Project as a free software replacement for the Bourne shell (sh).
- ◆ Default Shell on Linux and Mac OSX
- ◆ The name is also descriptive of what it did, bashing together the features of sh, csch and ksh

tsch : TENEX C Shell

- ◆ Developed by Ken Greer at Carnegie Mellon University
- ◆ It is essentially the C shell with programmable command line completion, command-line editing, and a few other features.

Shell Comparison

Software	sh	csH	ksh	bash	tcsh
Programming Language	✓	✓	✓	✓	✓
Shell Variables	✓	✓	✓	✓	✓
Command alias	X	✓	✓	✓	✓
Command history	X	✓	✓	✓	✓
Filename completion	X	★	★	✓	✓
Command line editing	X	X	★	✓	✓
Job control	X	✓	✓	✓	✓

✓ : Yes

X : No

★ : Yes, not set by default

<http://www.cis.rit.edu/class/simg211/unixintro/Shell.html>

- ▶ 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.

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

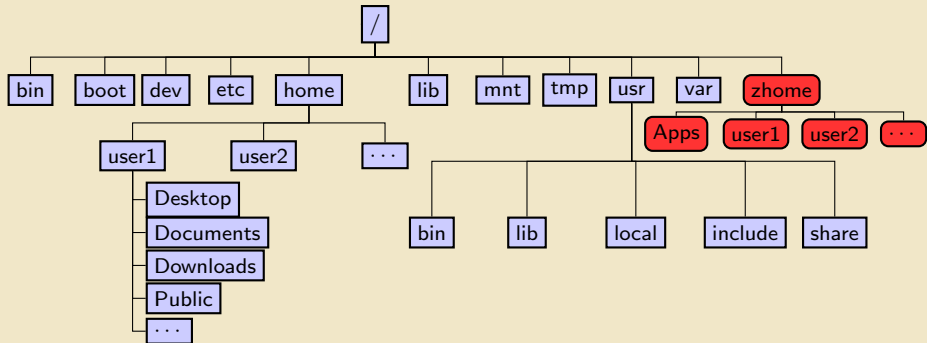
setenv TEXINPUTS ".:usr/share/texmf//:/home/apacheco/LaTeX//:${TEXINPUTS}"
setenv BIBINPUTS ".:/home/apacheco/TeX//:${BIBINPUTS}"
```

Files and Processes

- ▶ Everything in Linux/UNIX is either a file or a process
- ▶ A File is a collection of data, created by users using text editors, running compilers, etc.
- ▶ Examples of Files:
 1. document such as collection of ascii text as in report, essay, etc.
 2. program written in some high level programming language
 3. instructions comprehensible to machine but not a casual user such as executable, binary file
 4. directory containing information about its contents such as subdirectories or other files
- ▶ A process is an executing program identified by a unique process identifier or PID.

Directory Structure

- All files are arranged in a hierarchical structure, like an inverted tree.
- The top of the hierarchy is traditionally called **root** (written as a slash /)



Important Directories

/bin: contains files that are essential for system operation, available for use by all users.

/lib,/lib64: contains libraries that are essential for system operation, available for use by all users.

/var: used to store files which change frequently (system level not user level)

/etc: contains various system configurations

/dev: contains various devices such as hard disk, CD-ROM drive etc

/sbin: same as bin but only accessible by **root**


/tmp: temporary file storage

/boot: contains bootable kernel and bootloader

/usr: contains user documentations, binaries, libraries etc

/home: contains home directories of all users. This is the directory where you are at when you login to a Linux/UNIX system.

- Installing your own OS: **/bin,/lib{64},/etc,/dev** and **/sbin** must be on the same partition.

- ▶ UNIX like OS's are designed for multi user environments i.e. multiple users can exist on the system.
- ▶ Special user called **root** is the administrator and has access to all files in the system.
- ▶ In *nix, users are organized into groups.
- ▶ Each user is in atleast one group.
- ▶ Group membership makes it easier to share files with members of your group.
Type **groups**  to find your group membership.
- ▶ All files are *case sensitive*,
- ◆ myfile.txt, Myfile.txt and myfile.TXT are three different files and can exist in the same directory simultaneously.

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 *absolute 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

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 these variables in a few slides when we learn some 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` 
- ▶ `apropos` shows you all of the man pages that may shed some light on a certain command.
- ▶ Usage: `apropos [keyword]`
- ▶ Example: `apropos editor` 

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  key is pressed and stores them into a variable.
- ▶ Usage: `read <variable name>`
- ▶ Example: `read name` 
Alex Pacheco 
- ▶ In the above example, the name that you enter is 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` 

2. `echo Welcome to HPC Training` 

- 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"` 

Input & Output Commands III

Exercise

- Print out the variable you created a few slides back

```
echo $MYNAME 
```

```
echo $Greet 
```

- Read a variable for greeting message

```
read message 
```

```
Welcome to HPC 
```





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

```
echo $Greet $MYNAME $message 
```


- What is the output of the following command?

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

Commands: `pwd` & `cd`

- ▶ `pwd` command prints the current working directory.
- ▶ Usage: `pwd`
- ▶ Example: `pwd` 
- ▶ `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` 
- ▶ The default destination directory is your home directory.
- ▶ i.e. If you type `cd`  , you will end up in your home directory.
- ▶ If you want to go back to the previous directory, type `cd -` 

Command: ls

- ▶ `ls` command lists the contents of a directory.
- ▶ Usage: `ls <options> <path>`
- ▶ Example: `ls` 
- ▶ 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` 
- ▶ 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` 
- ▶ 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` 

`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` 
- ▶ Common options to `rm` command:
 - r: remove recursively, required when deleting 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` 

`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

```
wc -l .bashrc
```

grep: Find patterns in a file

```
grep alias .bashrc
```

awk: File processing and report generating

```
awk '{print $1}' file1
```

sed: Stream Editor

```
sed 's/home/HOME/g' .bashrc
```

set: manipulate environment variables

```
set -o emacs
```

ln: Link a file to another file

```
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` .

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` .

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

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 
```

Uncompress

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

- ▶ 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

- 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

- ▶ 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

- ▶ The third triad signifies what everyone else can do

$$\begin{array}{ccccccc} & & & g & & & \\ & & & \text{---} & & & \\ d & rwx & r & - & x & r & - & x \\ & u & & & & o & & \end{array}$$

- ▶ 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`

- 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}`?

- ▶ 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

- ▶ A process is an executing program identified by a unique PID
- ★ To see information about your running processes and their PID and status,
ps 
- ▶ 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 &` 
 2. First suspend the job using ` 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.


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

```
jobs 
```


- ▶ 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  C

2. Job whose PID you know

```
kill PID 
```

3. Job whose `jobnumber` you know (from `jobs` command)

```
kill %jobnumber 
```

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

Editors

- ▶ 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.

- ▶ **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

- ▶ insert at cursor
- ▶ insert at beginning of line
- ▶ append after cursor
- ▶ append at end of line
- ▶ newline after cursor in insert mode
- ▶ newline before cursor in insert mode
- ▶ append at end of line
- ▶ exit insert mode

vi

- ▶ i
- ▶ I
- ▶ a
- ▶ A
- ▶ o
- ▶ O
- ▶ ea
- ▶ ESC


Cursor Movement

- ▶ move left
- ▶ move down
- ▶ move up
- ▶ move right
- ▶ jump to beginning of line
- ▶ jump to end of line
- ▶ goto line *n*
- ▶ goto top of file
- ▶ goto end of file
- ▶ move one page up
- ▶ move one page down

vi


- ▶ h
- ▶ j
- ▶ k
- ▶ l
- ▶ ^
- ▶ \$
- ▶ nG
- ▶ 1G
- ▶ G
- ▶ C-u
- ▶ C-d

emacs

- ▶ C-b
- ▶ C-n
- ▶ C-p
- ▶ C-f
- ▶ C-a
- ▶ C-e
- ▶ M-x goto-line  *n*
- ▶ M-<
- ▶ M->
- ▶ M-v
- ▶ C-v

C : Control Key

M : Meta or ESCAPE (ESC) Key

 : Enter Key

File Manipulation

- ▶ save file
- ▶ save file and exit
- ▶ quit
- ▶ quit without saving
- ▶ delete a line
- ▶ delete n lines
- ▶ paste deleted line after cursor
- ▶ paste before cursor
- ▶ undo edit
- ▶ delete from cursor to end of line
- ▶ search forward for *patt*
- ▶ search backward for *patt*
- ▶ search again forward (backward)

vi

- ▶ :w
- ▶ :wq, ZZ
- ▶ :q
- ▶ :q!
- ▶ dd
- ▶ n dd
- ▶ p
- ▶ P
- ▶ u
- ▶ D
- ▶ \i

patt
- ▶ ?*patt*
- ▶ n

emacs

- ▶ C-x C-s
- ▶
- ▶ C-x C-c
- ▶
- ▶ C-a C-k
- ▶ C-a M- n C-k
- ▶ C-y
- ▶
- ▶ C-_
- ▶ C-k
- ▶ C-s *patt*
- ▶ C-r *patt*
- ▶ C-s(r)

File Manipulation (contd)

- ▶ replace a character
- ▶ join next line to current
- ▶ change a line
- ▶ change a word
- ▶ change to end of line
- ▶ delete a character
- ▶ delete a word
- ▶ edit/open file *file*
- ▶ insert file *file*
- ▶ split window horizontally
- ▶ split window vertically
- ▶ switch windows

vi

- ▶ r
- ▶ J
- ▶ cc
- ▶ cw
- ▶ c\$
- ▶ x
- ▶ dw
- ▶ :e *file*
- ▶ :r *file*
- ▶ :split or C-ws
- ▶ :vsplit or C-wv
- ▶ C-ww

emacs


- ▶
- ▶
- ▶
- ▶
- ▶
- ▶ C-d
- ▶ M-d
- ▶ C-x C-f *file*
- ▶ C-x i *file*
- ▶ C-x 2
- ▶ C-x 3
- ▶ C-x o

- 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**  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}"
```

What is a Scripting Language?

- ▶ A **scripting language** or **script language** is a *programming language* that supports the writing of **scripts**.
- ▶ **Scripting Languages** provide a higher level of abstraction than standard programming languages.
- ▶ Compared to programming languages, scripting languages do not distinguish between data types: integers, real values, strings, etc.
- ▶ Scripting Languages tend to be good for automating the execution of other programs.
 - ◆ analyzing data
 - ◆ running daily backups
- ▶ They are also good for writing a program that is going to be used only once and then discarded.

What is a script?

- ▶ A **script** is a program written for a software environment that automate the execution of tasks which could alternatively be executed one-by-one by a human operator.
- ▶ The majority of script programs are “quick and dirty”, where the main goal is to get the program written quickly.

Writing your first script

1. Write a script

- A shell script is a file that contains ASCII text.
- Create a file, `hello.sh` with the following lines

```
#!/bin/bash
# My First Script
echo "Hello World!"
```

2. Set permissions

```
apacheco@apacheco:~/Tutorials/BASH/scripts> chmod 755 hello.sh
```

3. Execute the script

```
apacheco@apacheco:~/Tutorials/BASH/scripts> ./hello.sh
Hello World!
```

Description of the script

- ▶ My First Script

```
#!/bin/bash
# My First Script
echo "Hello World!"
```

- ▶ The first line is called the "SheBang" line. It tells the OS which interpreter to use. In the current example, bash

- ▶ Other options are:

```
sh : #!/bin/sh
ksh : #!/bin/ksh
csh : #!/bin/csh
tcsh : #!/bin/tcsh
```

- ▶ The second line is a comment. All comments begin with "#".
- ▶ The third line tells the OS to print "Hello World!" to the screen.

Special Characters

`#`: starts a comment.

`$`: indicates the name of a variable.

`\`: escape character to display next character literally.

`{ }`: used to enclose name of variable.

`;` Command separator [semicolon]. Permits putting two or more commands on the same line.

`::` Terminator in a case option [double semicolon].

`.` "dot" command [period]. Equivalent to source. This is a bash builtin.

`$?` exit status variable.

`$$` process ID variable.

`[]` test expression









`[[]]` test expression, more flexible than `[]`

`$[]`, `(())` integer expansion.

`||`, `&&`, `!` Logical OR, AND and NOT

Quotation

- ▶ Double Quotation " "
 - Enclosed string is expanded ("\$", "/" and "'")
 - Example: `echo "$myvar"` prints the value of `myvar`
- ▶ Single Quotation ' '
 - Enclosed string is read literally
 - Example: `echo '$myvar'` prints `$myvar`
- ▶ Back Quotation ` `
 - Enclosed string is executed as a command
 - Example: `echo `pwd`` prints the output of the `pwd` command i.e. print the current working directory

- ▶ Login to a Linux machine and open a terminal
- ▶ Enter the following commands or carry out operations asked for.
- ▶ Understand what you are doing and ask for help if unsure. Some commands are incorrect or will fail, enter the correct
 1. `echo hello world` 
 2. `pwd` 
 3. `whoami` 
 4. `cd /tmp` 
 5. `cd -` 
 6. `mkdir test/testagain` 
 7. `cd test/testagain` 
 8. `touch file` 
 9. Go back to your home directory.
 10. Which shell are you using?
 11. Review the commands you have just entered.
 12. create an alias for removing files which prompt for confirmation and delete the file that you created.
 13. From your home directory get a list of files and directory in long format in reverse order with file sizes listed in human readable format.

Exercises II

14. Find out the location of vi, emacs, firefox, google-chrome, thunderbird, latex, pdflatex, gnuplot, python, perl and matlab.
15. Change the permission of the testagain directory to be world writable.
16. open a few applications of choice in foreground one by one and then suspend them,
17. get a list of suspended jobsr,
18. foreground job 1 and close it,
19. background job 2,
20. kill job 3,
21. put job 2 in foreground and close it,
22. check if you still have any jobs running.

Exercises III

1. Exercise courtesy <http://www.doc.ic.ac.uk/~wjk/UnixIntro/Exercise6.html>
2. Copy the file mole.txt
`wget http://www.doc.ic.ac.uk/~wjk/UnixIntro/mole.txt`
3. Go to the end of the document and type in the following paragraph:
Joined the library. Got Care of the Skin, Origin of the Species, and a book by a woman my mother is always going on about. It is called Pride and Prejudice, by a woman called Jane Austen. I could tell the librarian was impressed. Perhaps she is an intellectual like me. She didn't look at my spot, so perhaps it is getting smaller.
4. Correct the three spelling errors in the first three lines of the first paragraph (one error per line) and remove the extra "Geography" in the 3rd line of the first paragraph.
5. Add the words "About time!" to the end of the second paragraph.
6. Delete the sentence "Time flies like an arrow but fruit flies like a banana" and re-form the paragraph.
7. Replace all occurrences of "is" with "was".
8. Swap the two paragraphs.
9. Save the file and quit.

Wednesday January 14th

Joined the library. Got *Care of the Skin*, *Origin of the Species*, and a book by a woman my mother was always going on about. It was called *Pride and Prejudice*, by a woman called Jane Austen. I could tell the librarian was impressed. Perhaps she was an intellectual like me. She didn't look at my spot, so perhaps it was getting smaller.

None of the teachers at school have noticed that I am an intellectual. They will be sorry when I am famous. There was a new girl in our class. She sits next to me in Geography. She was all right. Her name was Pandora, but she likes being called "Box". Don't ask me why. I might fall in love with her. It's time I fell in love, after all I am 13 $\frac{3}{4}$ years old. About time!