Introduction to Linux

Alexander B. Pacheco

User Services Consultant
LSU HPC & LONI
sys-help@loni.org

HPC Training Spring 2013
Louisiana State University
Baton Rouge
Februrary 6, 2013

- 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

- Frustated 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).

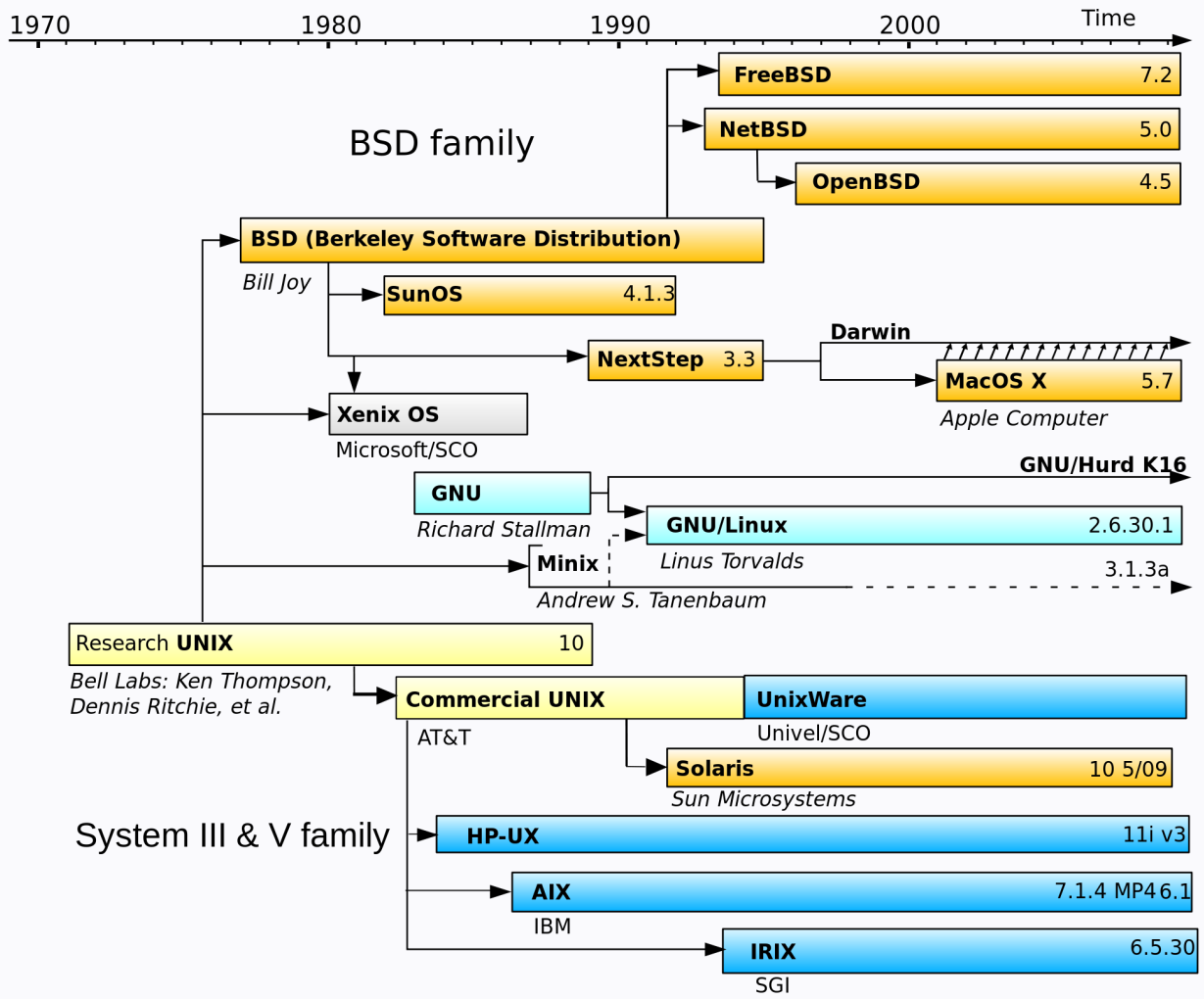
I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

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.



http://en.wikipedia.org/wiki/Linux

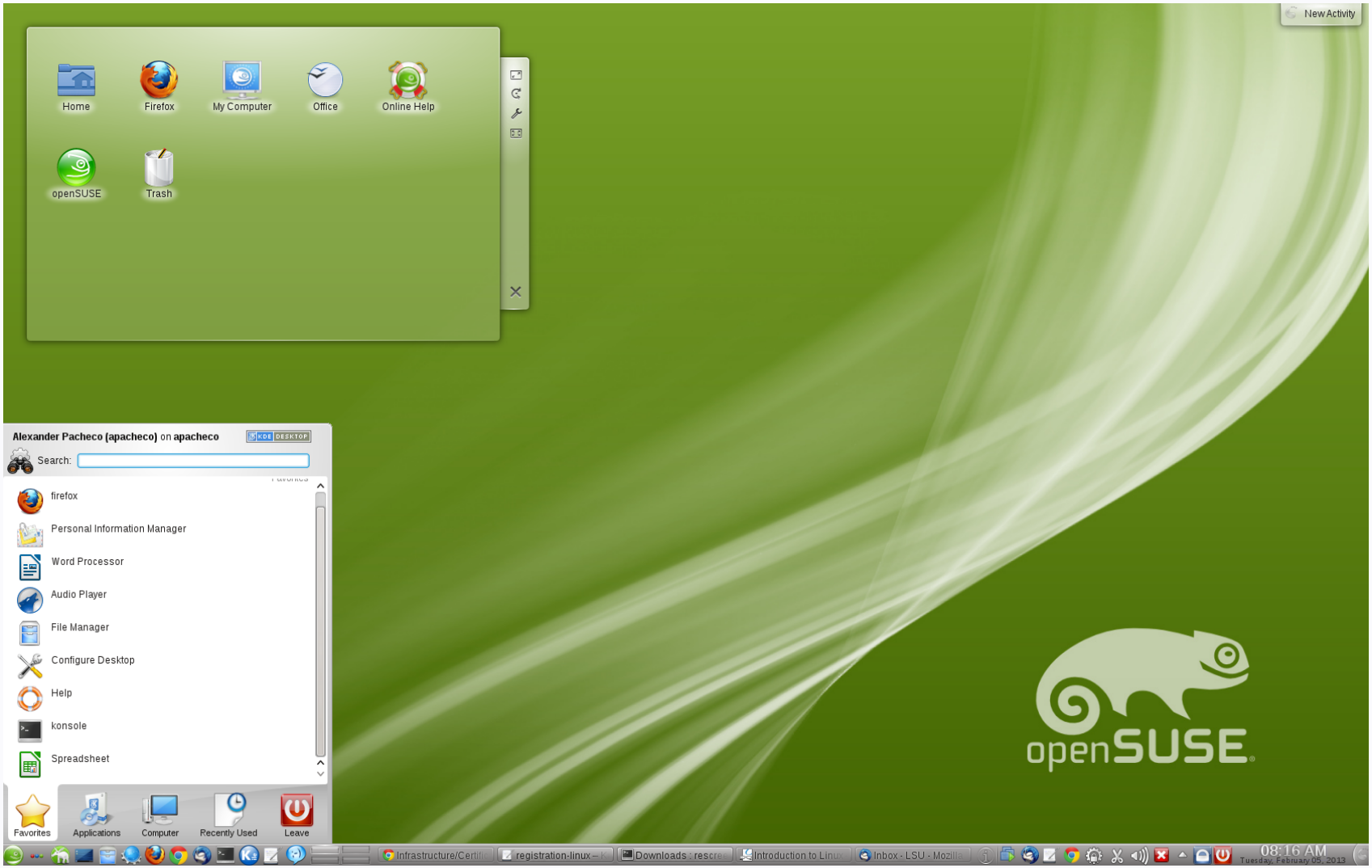
- Linux is an operating system that evolved from a kernel created by Linus Torvalds when he was a student at the University of Helsinki.
- It's meant to be used as an alternative to other operating systems, Windows, Mac OS, MS-DOS, Solaris and others.
- Linux is the most popular OS used in a Supercomputer

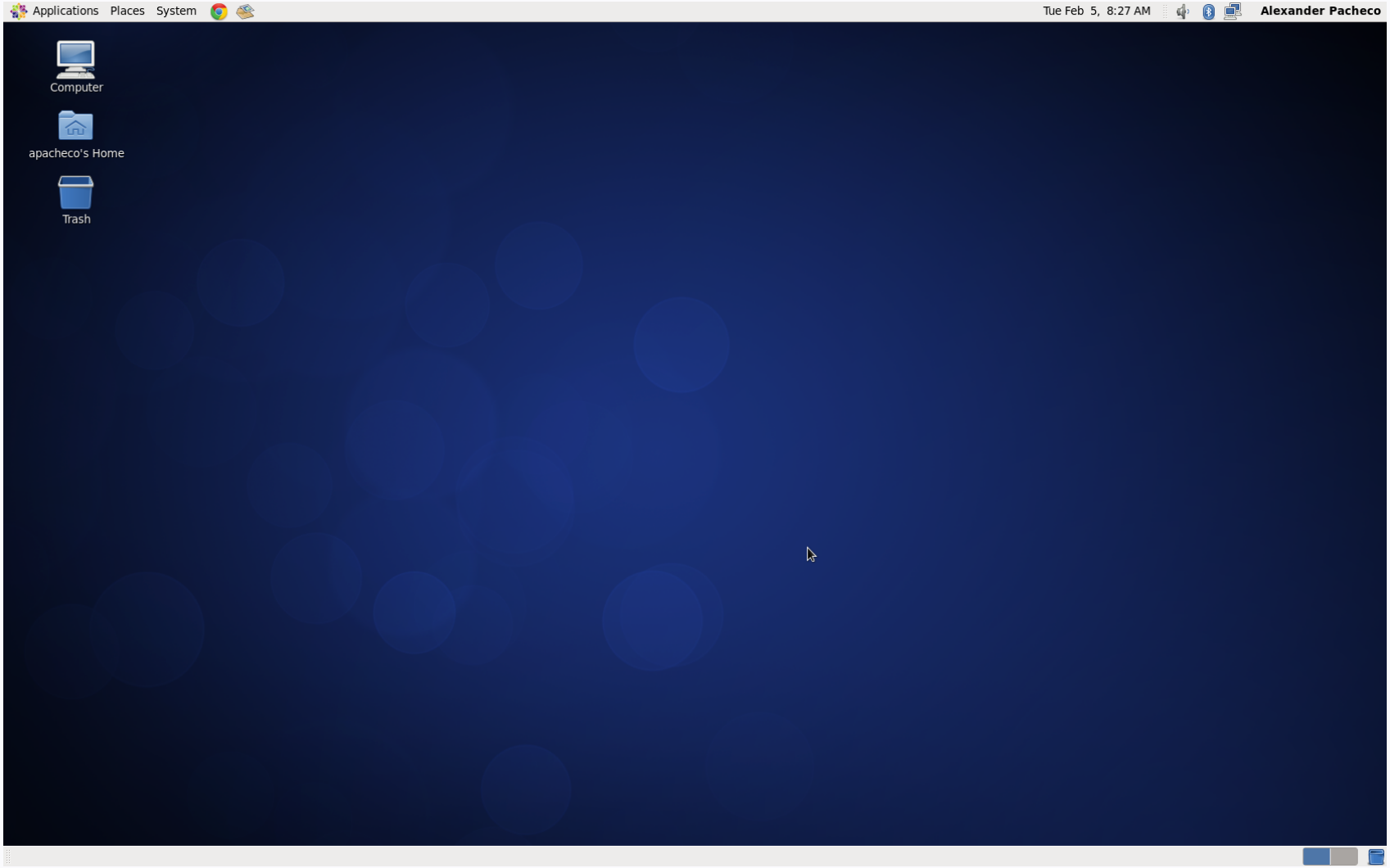
OS Family	Count	Share %
Linux	469	93.8
Unix	20	4
Mixed	7	1.4
Windows	3	0.6
BSD Based	1	0.2

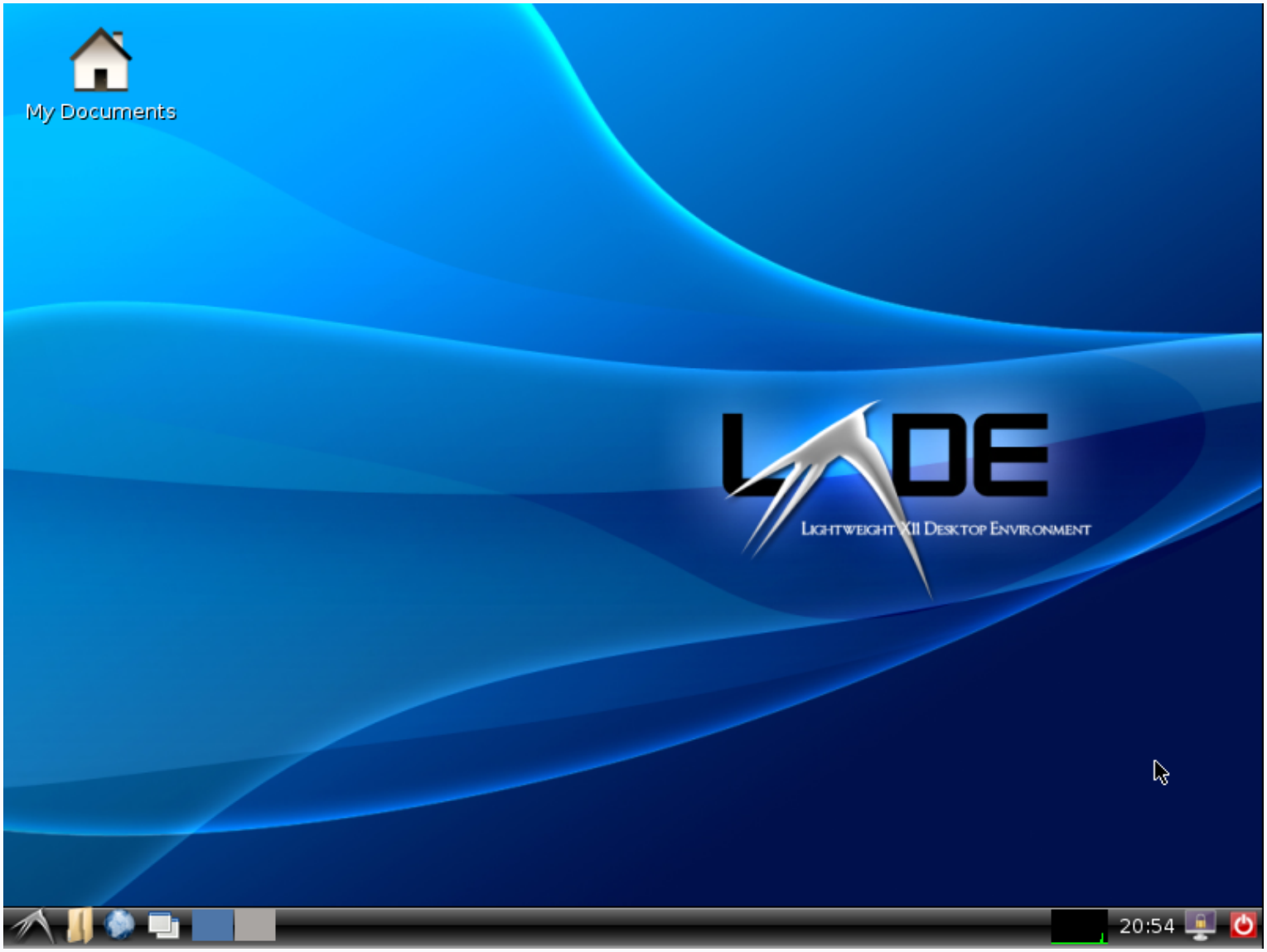
- If you are using a Supercomputer for your research, there is a 98% probability that it will be based on a *nix OS.

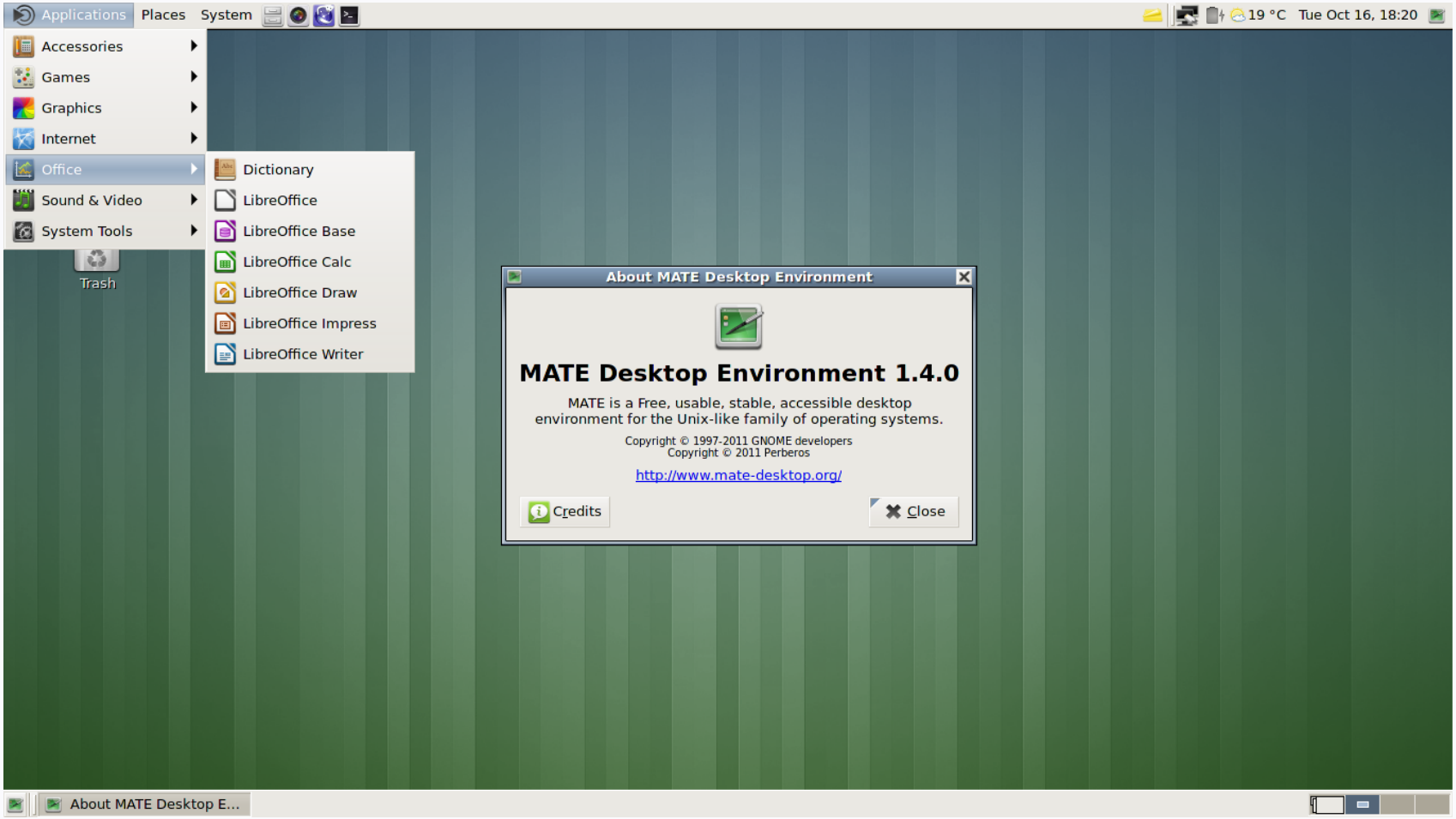
http://www.top500.org/statistics/list/

- 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


New Activity
Home
Firefox
My Computer
Office
Online Help
openSUSE
Trash
Alexander Pacheco (apacheco) on apacheco
KDE DESKTOP
Search:
firefox
Personal Information Manager
Word Processor
Audio Player
File Manager
Configure Desktop
Help
konsole
Spreadsheet
Favorites
Applications
Computer
Recently Used
Leave
openSUSE
08:16 AM
Tuesday, February 05, 2013


Applications
Places
System
Tue Feb 5, 8:27 AM
Alexander Pacheco
Computer
apacheco's Home
Trash


My Documents
LXDE
Lightweight X11 Desktop Environment
20:54


Applications
Places
System
19 °C
Tue Oct 16, 18:20
Accessories
Games
Graphics
Internet
Office
Sound & Video
System Tools
Dictionary
LibreOffice
LibreOffice Base
LibreOffice Calc
LibreOffice Draw
LibreOffice Impress
LibreOffice Writer
Trash
About MATE Desktop Environment
MATE Desktop Environment 1.4.0
MATE is a Free, usable, stable, accessible desktop environment for the Unix-like family of operating systems.
Copyright © 1997-2011 GNOME developers
Copyright © 2011 Perberos
http://www.mate-desktop.org/
Credits
Close
About MATE Desktop E...


Computer
All Applications
Accessories
Education
Games
Graphics
Internet
Office
Programming
Sound & Video
System Tools
Preferences
Administration
Places
Recent Files
Additional Drivers
AisleRiot Solitaire
Alarm Clock
APTonCD
Archive Manager
Armagetron Advanced
Artha
Audacity
Backup Tool
Banshee
BleachBit
BleachBit (as root)
Bluetooth
BootUp-Manager
Menu
77%
Sat Dec 1, 05:41

- 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

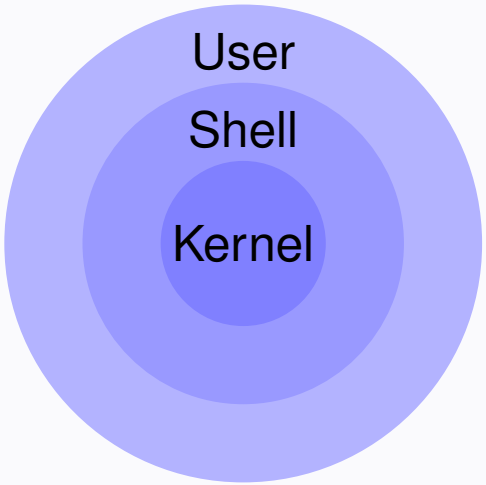
e.g. Microsoft Office equivalents are OpenOffice.org, LibreOffice, KOffice

- 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.

- 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	3614	▲
2	Mageia	2418	▼
3	Ubuntu	1906	▲
4	Fedora	1558	▲
5	OpenSUSE	1349	▼
6	Debian	1345	▲
7	Arch	1211	▼
8	PCLinuxOS	1186	▲
9	Snowlinux	877	▲
10	Zorin	861	–

- Linux is made up of two (three) parts:
  1. Kernel
  2. Shell
  3. Applications/Programs



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.

`sh` : Bourne Shell

- Developed by Stephen Bourne at AT&T Bell Labs

`csh` : 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, csh and ksh

`tcsh` : 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.

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

✓ : Yes

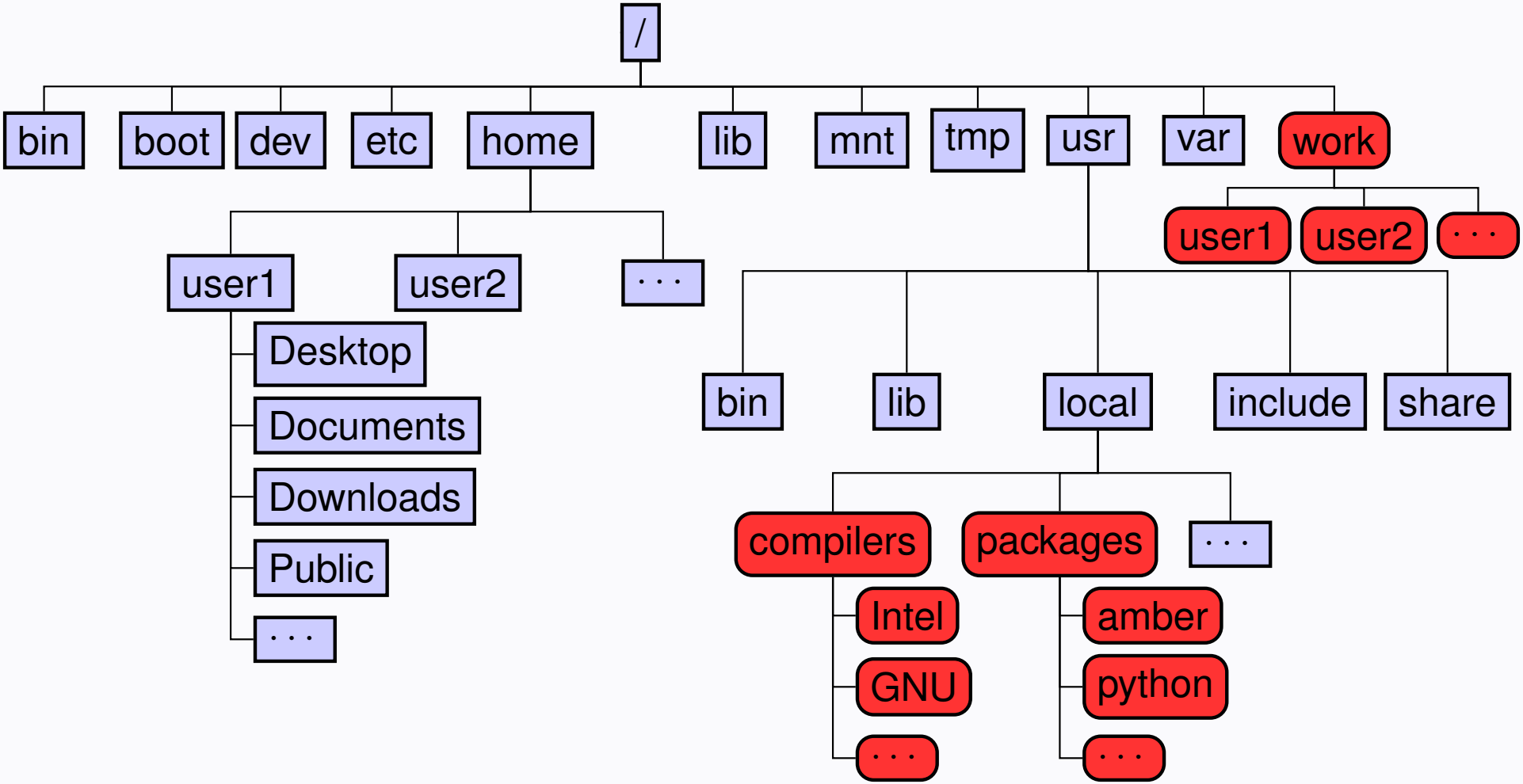
✗ : No

★ : Yes, not set by default

Ref : `http://www.cis.rit.edu/class/simg211/unixintro/Shell.html`

- 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.

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



/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.

- 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 alteast one group.
- ★ On LONI systems, you are in one of the following groups: `lsuusers`, `latechusers`, `unousers`, `ullusers`, `susers`, `tulaneusers`, `loni` or `xavierusers`
- Due to software licensing, you cannot be in more than one of the above groups.
- Group membership makes it easier to share files with members of your group or in the case of LONI systems, with researchers at your university.

  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.

- *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*

- *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

- 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.

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.

- 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.

- 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 this variables in a few slides when we learn some 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` show 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`⟵┘

Print to screen

- `echo arguments` will print `arguments` to screen or standard output.
- `arguments` can be a variable or string of characters including numbers.
- Examples:
  1. `echo $LD_INCLUDE_PATH⟵┘`
  2. `echo Welcome to HPC Training⟵┘`

Exercise

- Print out the variable you created a few slides back

  `echo $MYNAME⟵┘`

  `echo $Greet⟵┘`
- Combine and print your name and the greeting

  `echo $Greet $MYNAME⟵┘`
- What is the output of the following command?

  `echo $Greet $MYNAME, Welcome to HPC Training⟵┘`

Where are you at?

- **pwd**: prints working directory.
- Usage: `pwd`
- Example: `pwd`⟵┘

How to move around the file system

- **cd**: allows one to change directory i.e.
- 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 -` ⟵┘

Listing Directory Contents

- **ls**: list contents of the current or another 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
  1. `-l`: show long listing format
  2. `-a`: show hidden files
  3. `-r`: reverse order while sorting
  4. `-t`: show modification times
  5. `-h`: use file sizes in SI units (bytes, kilobytes, megabytes etc) default is bytes

Create a Directory

- **mkdir**: 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

Copying Files and Directory

- **cp**: copy a file or directory
- Usage: `cp <options> <source(s)> <destination>`
- Example: `cp $HOME/.bashrc ../../tmp`⟵┘
- Common options to **cp** command:
  1. `-r`: copy recursively, required when copying directories.
  2. `-i`: prompt if file exists on destination and can be copied over.
  3. `-p`: preserve file access times, ownership etc.
- If there are more than one source file, then the last file is the destination.
- 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 `dir1` will be the same as `file2`

Removing Files and Directory

- **rm**: remove/delete a file or directory
- Usage: `rm <options> <file or directory>`
- Example: `rm ../../tmp/.bashrc`⟵┘
- Common options to **cp** command:
  1. `-r`: remove recursively, required when copying directories.
  2. `-i`: prompt if file really needs to be deleted
  3. `-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.

Moving or Renaming Files/Directories

- **mv**: move or rename 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.
- 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 `dir1` will be the same as `file2` and `file1` will be deleted.

Display Contents of a file

- **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".

passwd: change password (does not work on LSU HPC and LONI systems)

chsh: change default shell (does not work on LSU HPC and LONI systems)

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

find : Find a file

alias: enables replacement of a word by another string

vi: Edit a file using VI/VIM

emacs: Edit a file using Emacs

file: Determine file type

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`

touch: change file timestamps or create file if not present

date: display or set date and time

ln: Link a file to another file

- `ln -s file1 file2`

which: shows the full path of (shell) commands

who: show who is logged on

whoami: print effective userid

finger: user information lookup program

whatis: display manual page descriptions

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

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 LSU HPC and LONI clusters.

- Usage: `ssh <username>@<remote host>`
- Example: `ssh apacheco@eric.loni.org`
- 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>`

Copy files over the Internet

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

  `<user>@<host>:/path_to_destination`

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

```
scp apacheco@eric.loni.org:/work/apacheco/somefile .
scp -r Public apacheco@eric.loni.org:/work/apacheco/
```

- 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.

- **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:
  1. `-a`: archive mode
  2. `-r`: recurse into directories
  3. `-v`: increase verbosity
  4. `-z`: compress file data during the transfer
  5. `-u`: skip files that are newer on the receiver
  6. `-t`: preserve modification times
  7. `-n`: dry-run, perform a trial run with no changes made
- Example: `rsync -avtzu eric.loni.org:~/* .`

File Compression

- 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.

Tape Archive

- *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
  1. `-c`: create an archive file
  2. `-x`: extract to disk from archive
  3. `-z`: filter the archive through gzip (adds/requires extension .gz)
  4. `-j`: filter the archive through bzip2 (adds/requires extension .bz2)
  5. `-t`: list contents of archive
  6. `-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

- There are three file descriptors for I/O streams
  1. STDIN: Standard Input
  2. STDOUT: Standard Output
  3. 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`

- 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
  2. group
  3. world i.e. everyone else who has access to the system

```
drwxr-xr-x.  2  user  user  4096  Jan  28  08:27  Public
-rw-rw-r--.  1  user  user  3047  Jan  28  09:34  README
```

- 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
- 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
- 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
```

- Use the `-R` flag to change permissions recursively, all files and directories and their contents.

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

What is the permission on ${HOME}?

- 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.
- Unprivileged (regular) users who wish to change the group of a file that they own may use **chgrp**.

e.g. Your default group on LSU HPC is `users` and your advisor requested sysadmins to create a group `abc` for collaborative research among say 10 researchers.

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

- 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 in stored in the variable `name`.
- Use the **echo** command to print the variable `name` to the screen
- `echo $name`⟵┘
- The **echo** statement can print multiple arguments.
- 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`↵ (more than one space between HPC and Training

`echo "Welcome to HPC  Training"`↵

- `read name`↵

  Alex Pacheco↵

- `echo $name`↵
- `echo "$name"`↵

- 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 `CNTRL Z` or `^ 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, type

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

- To kill or terminate a process:
  1. Job running in foreground: enter `CNTRL C` or `^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 optiopn 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.

- 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	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`

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 ⟵ 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

⟵ : Enter Key

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)`

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`

- 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.

- 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 rm="rm -i"` command will modify all `rm` commands that you type as `rm -i`

.bashrc

```
# .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}
```

.tcshrc

```
# .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.

Three things to do to write and execute a 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!
```

- 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.

#:	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

- 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 working directory

```
apacheco@apacheco:~/Tutorials/BASH/scripts> cat quotes.sh
#!/bin/bash

HI=Hello

echo HI             # displays HI
echo $HI            # displays Hello
echo \$HI           # displays $HI
echo "$HI"          # displays Hello
echo '$HI'          # displays $HI
echo "$HIAlex"      # displays nothing
echo "${HI}Alex"    # displays HelloAlex
echo `pwd`          # displays working directory
apacheco@apacheco:~/Tutorials/BASH/scripts> ./quotes.sh
HI
Hello
$HI
Hello
$HI

HelloAlex
/home/apacheco/Tutorials/BASH/scripts
apacheco@apacheco:~/Tutorials/BASH/scripts>
```

- Create shell scripts using either vi or emacs to do the following
  1. Write a simple hello world script
  2. Modify the above script to use a variable
  3. Modify the above script to prompt you for your name and then display your name with a greeting.
- The goal of this exercises is three fold
  1. Get you comfortable with using vi or emacs
  2. Get you started with writing shell scripts
  3. Let you play around with the **chmod** command
- Everything that goes into the scripts should have already been done by you on the command prompt.

Exercise 1

```
#!/bin/bash

# My First Script

echo "Hello World!"
```

Exercise 2

```
#!/bin/bash

# Hello World script using a variable
STR="Hello World!"
echo $STR
```

Exercise 3

```
#!/bin/bash

# My Second Script

echo Please Enter your name:
read name
echo Please Enter a standard Greeting
read greet
echo "$greet $name"
```

- Recall: All variables are stored in memory as strings and converted to numbers when needed
- You can carry out numeric operations on variables
- Arithmetic operations in `bash` can be done within the `$((···))` or `$[···]` commands
  - ★ Add two numbers: `$((1+2))`
  - ★ Multiply two numbers: `$[$a*$b]`
  - ★ You can also use the `let` command: `let c=$a-$b`
- In `tcsh`,
  - ★ Add two numbers: `@ x = 1 + 2`
  - ★ Divide two numbers: `@ x = $a / $b`

Exercise

Write a script to add/subtract/multiply/divide two numbers.

```
apacheco@apacheco:~/Tutorials/BASH/scripts> cat dosum.sh
#!/bin/bash

FIVE=5
SEVEN=7
echo "5 + 7 = " $FIVE + $SEVEN
echo "5 + 7 = " $(($FIVE + $SEVEN))
let SUM=$FIVE+$SEVEN
echo "sum of 5 & 7 is " $SUM
exit
apacheco@apacheco:~/Tutorials/BASH/scripts> ./dosum.sh
5 + 7 = 5 + 7
5 + 7 =  12
sum of 5 & 7 is  12
apacheco@apacheco:~/Tutorials/BASH/scripts> cat dosum.csh
#!/bin/tcsh

set FIVE=5
set SEVEN=7
echo "5 + 7 = " $FIVE + $SEVEN
@ SUM = $FIVE + $SEVEN
echo "sum of 5 & 7 is " $SUM
exit
apacheco@apacheco:~/Tutorials/BASH/scripts> ./dosum.csh
5 + 7 =  5 + 7
sum of 5 & 7 is  12
```

```
apacheco@apacheco:~/Tutorials/BASH/scripts> cat backups.sh
#!/bin/bash

BACKUPDIR=$(pwd)
OF=$BACKUPDIR/$(date +%Y-%m-%d).tgz
tar -czf ${OF} ./*sh
apacheco@apacheco:~/Tutorials/BASH/scripts> ./backups.sh
apacheco@apacheco:~/Tutorials/BASH/scripts> ls *gz
2012-09-18.tgz
```

- User Guides
  - LSU HPC: `http://www.hpc.lsu.edu/docs/guides.php#hpc`
  - LONI: `http://www.hpc.lsu.edu/docs/guides.php#loni`
- Documentation: `https://docs.loni.org`
- Online Courses: `https://docs.loni.org/moodle`
- Contact us
  - Email ticket system: sys-help@loni.org
  - Telephone Help Desk: 225-578-0900
  - Instant Messenger (AIM, Yahoo Messenger, Google Talk)
    - Add "lsuhpchelp"

The End

Any Questions?

Feb 20: Regular Expressions

Feb 27: Advanced Shell Scripting

Survey:
http://www.hpc.lsu.edu/survey