Welcome to Linux

- Welcome
- Survey Of Class
- Logging in
- What is a shell?
- Using a shell
- How to use a command line

Welcome

- Instructor: Tom Duffy
- CA: Hendra Wijaya
- Course Materials
  - Course Outline
  - Chapter 3, Linux Tutorial
  - VI Tutorial
  - "In The Beginning Was The Command Line"
  - "The Cathedral And The Bazaar"

Welcome (cont.)

- Class should be interactive
- Ask Questions at any time
- Let me know if I am going too fast or too slow
  - One topic required for next topic, so it is really important for every one to keep up
- 15 minute break in the middle of class

Survey Of Class

- Computer Experience (show of hands)
  - Windows/Mac GUI (I assume all have good to excellent experience)
  - Inside a computer (Harddrive, CPU, memory, etc.)
  - Command Line Experience (DOS?)
  - UNIX or Linux Experience

Logging In

- Multitasking, Multiuser operating system
  - Can run many programs at once
  - Many people can be on same machine at same time
  - Unlike DOS, where only one person can run one program at one time
  - Windows NT can run multiple programs at same time, but only one person can be logged in

Logging In (part 2)

- Each *user* on a system has a *login name*
  - Normally between 2 and 8 characters
- Each *login name* has a *password*
  - Only the *user* should know their *password*
  - Normally should be something hard to guess
- Please press [Ctrl]-[Alt]-[F1]

Logging In (part 3)

- Each machine has a *hostname*
- You should see a screen with

  hostname login:
- Here, the *login name* is "student" and the *password* is "student"
- When you enter your *password*, nothing will echo on the screen (so an on looker will not see your *password*)

Superuser account

- Normally, you login as a regular user
- Regular users are restricted on the computer as to what they can do
  - Cannot add/delete accounts
  - Cannot modify system settings
  - Can only read and modify their own files and settings

Superuser account (part 2)

- Superuser account is called "root"
- Similar to "Administrator" on Windows NT
- "root" has privilege to do anything on the system
  - Modify system settings
  - Add/Delete user accounts
  - Read/Write anyone's files

Virtual Consoles

- Console is a monitor and keyboard
- Normally, there is one login on the console
- Linux provides multiple login's using "virtual consoles"
- We have 6 virtual consoles, but you can have more
- Press [Alt]-[F2] to get to another console
- You can login again...press [Alt]-[F1] to return

What is a shell?

- The shell is the command line
  - Similar to COMMAND.COM in DOS
- A small program that allows you to interact with the operating system
- The system will start a shell automatically when you log in
- When you log in, you get a shell *prompt*

  [student@hostname student]$ _

What is a command?

- A command consists of two parts
  - Name of the command
  - Its *arguments*
- First item on command line is the command; the remainder are its *arguments*

[student@hostname stdudent]$ cp foo bar

- *cp* is the name of the command
- *foo* and *bar* are the *arguments*

Different types of commands

- Shell builtins
  - Commands that the shell itself knows how to do
- Aliases
  - User defined commands (normally a substitute name for another command)
- Programs on the hard drive
- When you type a command, it looks for all three of these

Types of commands (cont.)

- If the shell finds a *builtin*, an *alias*, or a *program* on disk
  - The shell starts that command and tells the command the *arguments* on the command line

[student@hostname student]$ cp foo bar

- Shell starts *cp* and passes *foo* and *bar* to it
- cp is a program that coppies a file to another file
- cp will copy file *foo* to a new file *bar*

Example commands

[student@hostname student]$ eat dirt

bash: eat: command not found

[student@hostname student]$ make love

make: *** No way to make target `love'. Stop.

Logging out

- To log out simply type "exit"
- When you are done using a Linux system, always log out

[student@hostname student]$ exit

- Log back in using *login name* "student" and *password* "student"

Changing your password

- You should change your password when you get an account on a system to something that is hard for someone else to guess
  - Use numbers and symbols
  - Avoid dictionary words

```
[student@hostname student]$ passwd
Changing password for tduffy
(current) UNIX password:
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```

<- Note the spelling

Change password back

- It won't let us change to student as normal user
  - "student" is a dictionary word
- Must login a root

hostname login: root

Password: sfsuitp

[root@hostname root]# passwd student
Changing password for user student
New UNIX password: student
Retype new UNIX password: student
[root@hostname root]# exit

Files

- A *file* is a collection of data or information that sits on your computer (normally on a hard disk or floppy disk)
- You access this file by using its *filename*
- Different types of files include:
  - History term paper
  - Email message
  - A program that you run

Directories

- A *directory* is a collection of files and other *directories*
- Directories have names like files
- Directories are organized in a "tree" structure
  - *directories* contain *sub-directories*

Files and Directories (cont.)

- To refer to a file or directory, you call it by its *path name*
- A *path name* is a list of *directories* and *sub-directories* to map where your *file* is
- Each directory is separated by a forward slash (/)
  - Different from windows's back slash (\)

papers/english-lit

  - Papers is a directory containing english-lit

Files and Directories (cont.)

papers/notes/cheat-sheet

- The directory "papers" contains another directory "notes" which contains a file "cheat-sheet"
- The directory that contains a *sub-directory* is called a *parent directory*
- "papers" is the *parent directory* of "notes"

The directory tree

- The filesystem on Linux contains a standard directory tree
- The *parent directory* of all directories is called the *root directory* (not to be confused with the superuser "root")
- The *root directory* is refered to with a single forward slash (/)

Home directories

- Each user on a system has a *home directory*
- Normally, this is in /home/username, where *username* is the *login name* of the *user*
- So, our user "student" has their *home directory* in /home/student
- "home" is a *sub-directory* of the *root directory* and "student" is a *sub-directory* of the "home" *directory*

Typical Linux directory tree

/ - bin

- dev
- etc
- home - student

\- larry

- usr - local - bin

\- bin

Current working directory

- At any given moment, you are in one directory
- This directory is called your *current working directory*
- When you login, Linux starts you in your *home directory*
- When you run programs and reference files, they are in relationship to you *current working directory*

CWD (cont.)

- Example:
- We need to create a file to play with

```
[student@hosname student]$ cp /etc/printcap /home/student
```

- The "more" command simply displays a file to the screen

```
[student@hosname student]$ more /home/student/printcap
[student@hosname student]$ more printcap
```

- In the first example, we refer to the file using the *absolute path name* (starts with root dir, /)
- In the second example, we use the *relative path name* (based on the *current working directory*)

Referring to home directories

- There is a shortcut to refer to your home directory
- You can use the character ~ to refer to your *home directory* instead of typing out the *absolute path name*

[student@hosname student]$ more ~/printcap

[student@hosname student]$ more /home/student/printcap

- Are the same

Linux is case sensative

- All files, programs, directories, etc. on Linux are case sensative
- Lower case and Upper case characters are different
- "make", "Make", and "MAKE" are all different names
- Most of the time, Linux uses lower case

Moving around

- To change your *current working directory*, use the command "**cd**" which means change directory
- The *argument* to "cd" is the directory to change to

[student@hosname student]$ cd /etc

- Will change your *current working directory* to the "etc" directory which is a sub-directory of the root directory

Special directories

- There are two special directories
  - .. refers to the *parent directory* of your *current working directory*
  - . refers to your *current working directory*

[student@hosname etc]$ cd ..

[student@hosname /]$

- Now, we are in the *root directory*
- Notice how the *prompt* changes to reflect the *current working directory*

Looking in a directory

- To see the contents of a directory, use the "**ls**" command for list

[student@hostname /]$ ls

bin core etc initrd misc mp3 proc sbin tmp var

boot dev home lib mnt opt root tftpboot usr

- You will see all the directories and files in that directory

Giving options to commands

- Most commands can be changed from their default behaviour by giving them options
- Using a "-" in front of a letter is how you pass an option to a command

[student@hosname /]$ ls -l

- Will change the bahaviour of **ls** to list the entries in their "long" format

Looking at files (cont.)

- **ls** can also be told to look at a directory that is not your current working directory

[student@hosname /]$ ls /home/student

- Will list the contents on your home directory

Creating new directories

- Using the "mkdir" command, you can create new directories
- Lets first move back into our home directory

[student@hosname /]$ cd /home/student

- Now, we can create a directory called "foo"

[student@hostname student]$ mkdir foo

[student@hostname student]$ ls

- Should show a new directory in the list name foo

Copying files

- You can copy files using the "cp" command

```
[student@hostname student]$ cd foo
[student@hostname foo]$ ls
[student@hostname foo]$ cp /etc/termcap .
[student@hostname foo]$ cp /etc/printcap .
[student@hostname foo]$ ls
printcap termcap
```

Moving files

- To move files (or rename them), use the "mv" command
- If the second argument is a directory, it will move the file
- If the second argument is a file (or non-existant), it will rename the file

[student@hostname foo]$ mv termcap bar

- An "ls" should show a termcap is now bar

Deleting files

- To delete a file, use the "rm" command for remove

[student@hostname foo]$ rm bar

- When you ls, bar is no longer there
- Note that when you delete a file on Linux, it is gone. FOREVER.
- There is no trash can or recycle bin. The file is toast. BE CAREFUL :)

Removing a directory

- The "rmdir" command will remove a directory
- The directory must be empty before Linux will allow you to remove it
- So, you must rm every file and rmdir ever sub-directory before rmdir'ing a directory

Looking at files

- As metioned earlier, you can look at files using the "more" command
- [Space] will go to the next page
- [b] will go back a page
- [q] will quit it
- "cat" also allows you to look at files, but it will not look at them one at a time -- the whole file will all scroll really quickly past the screen

Getting help

- To get help on a given command, you can use the command "man" for manual
- To get help on the ls command, I would type

[student@hostname student]$ man ls

- And I could read about how ls worked, what options I could give it, etc.

Types of shells

- Linux has many different types of shells, but there are some standard ones
- Normally, "bash" is the default shell on Linux
- But, you can use "sh", "csh", "tcsh", "zsh", "ksh"
- Each shell has slightly different command line semantics

History of shells

- sh -came first. Called the Bourne shell cause it was written by S. R. Bourne. Not very friendly for interactive use.
- csh -came from UCB. Known as the C Shell. Added aliasing and job control. Not very good at scripting.
- tcsh -added command line editing and TENEX-style completion to C shell
- ksh -David Korn from AT&T decided to use the language of Bourne shell w/ the added features of tcsh (but it wasn't free)
- bash -Bourn again shell. Same idea as ksh using Bourne shell language and C friendly interactivity. (but FREE)
- zsh -written by Paul Falstad while he was a student a Princeton. The kitchen sink of shells. Has creeping featurism.

Exploring the filesystem

- /bin - short for binaries. Location of essential system programs
- /dev - device files. Location of special files that correspond to hardware in your computer.
- /etc - system configuration files
- /sbin - static binaries. essential tools for system administration
- /home - location of users home directories
- /lib - essential system shared libraries. Libraries contain code that many programs use.
- /proc - process filesystem. This contains "virtual" files that allow you to look at the state of processes running on your system as well as memory and cpu information.

Filesystem (cont.)

- /tmp - location of temporary files. These get deleted every time you reboot.
- /usr - location of most of the programs and libraries on your system.
- /opt - optional programs installed by third party software vendors
- /var - area for variable size files like system logs, print queues, mail spools. Files that tend to grow or change often are put here.

Wildcards

- Allows you to specify many files without having to type all their names
- "*" refers to any number of characters or letters (including no characters)

Wildcards (cont.)

```
[student@hostname student]$ touch frog
[student@hostname student]$ touch joe
[student@hostname student]$ touch stuff
[student@hostname student]$ ls
frog      joe         stuff
[student@hostname student]$ ls *o*
from      joe
```

- Notice that only files with an "o" in it are listed

Wildcards (cont.)

- A "*" by itself will substitute all characters

```
[student@hostname student]$ ls *
frog     joe       stuff
[student@hostname student]$ ls f*
frog
[student@hostname student]$ ls *ff
stuff
[student@hostname student]$ ls *f*
frog    stuff
```

Wildcards (cont.)

- When a "*" is substituted for letters and characters, this is called *wildcard expansion*
- This is done before the arguments are passed to the program (i.e. The shell takes care of it)

[student@hostname student]$ ls *o*

- the shell turns into

[student@hostname student]$ ls frog joe

Wildcards (cont.)

- The "?" wildcard is substituted for one letter or character

[student@hostname student]$ ls ?

- Will only show files that are one character long – we don't have any right now

[student@hostname student]$ ls j?o

joe

[student@hostname student]$ ls f??g

frog

[student@hostname student]$ ls ????f

stuff

Using Wildcards (in the wild)

- You can use wildcards to copy (cp) or move (mv) multiple files

[student@hostname student]$ cp /etc/s* ~

- Will copy all files starting with "s" to your home directory (remember that ~ is a shortcut to your home directory)

Hidden Files

- Linux "hides" files that start with a period (.)
- Wildcards do *not* show hidden files
- In order to see hidden files, you must present ls with the -a option

[student@hostname student]$ ls -a

. .. frog joe stuff

Linux Plumbing

- Linux programs get their input from *standard input* (stdin) and send their output to *standard output* (stdout)
- The shell automatically (by default) sets it up so that *standard input* comes from your keyboard and *startdard output* goes to your monitor
- "cat" reads from the files on the command line and outputs to *stdout*

[student@hostname student]$ cat /etc/termcap

<output spews to screen>

Plumbing (cont.)

- If you don't specify a file to "cat", it will take input from *stdin* (your keyboard)

[student@hostname student]$ cat

Hello there.

Hello there.

Bye.

Bye.

[Ctrl-D] quits

- Notice that it echo's each line you type. That is because it is taking input from stdin (keyboard) and output to stdout (monitor)

Plumbing (cont.)

- The "sort" command takes input from *stdin*, sorts it in alphabetical order, and sends the sorted list to *stdout*

[student@hostname student]$ sort

bananas

carrots

apples

[Ctrl-D]

apples

bananas

carrots

Redirecting Input and Output

- The shell allows you to *redirect* stdout to a file by using the “>” directive

[student@hostname student]$ sort > shopping-list

bananas

carrots

apples

[Ctrl-D]

[student@hostname student]$ cat shopping-list

- Should see a sorted list of items now saved in file shopping-list

Redirected I/O (cont.)

```
[student@hostname student]$ cat > items
bananas
carrots
apples
[student@hostname student]$ sort items > shopping-list
[student@hostname student]$ cat shopping-list
apples
bananas
carrots
```

Redirecting Input

- We can also redirect input using the "<" directive

[student@hostname student]$ sort < items

apples

bananas

carrots

- "items" is sent as stdin to sort, but since stdout is not redirected, it goes to the monitor

Filters

- A *filter* is a program that takes input from *stdin*, processes it in some way, and sends the output to *stdout*
- "sort" is a filter

Using Pipes

- You can use these redirects for any command line
- For instance, you can redirect ls to a file

[student@hostname student]$ ls > file-list

- All the files in the directory are now in a new file called file-list

[student@hostname student]$ sort -r file-list

- The "-r" option to sort will do a reverse sort

Pipes (cont.)

- In order to get the output of ls and run it through sort, we needed to use a file in the process
- There is a shortcut using the "|" directive (this is a [shift]-[\]

[student@hostname student]$ ls | sort -r

- Will take the stdout of ls and put it into the stdin of sort

[student@hostname student]$ ls /usr/bin

- Will spew across the screen, so we can use the more command to fix this

[student@hostname student]$ ls /usr/bin | more

Pipes (cont.)

- You can pipe as many commands together as you want

[student@hostname student]$ ls | sort -r | head -1

- "head" is a command that takes the first -x lines and throws aways the rest; "-1" says take the first 1 line and throw away the rest

Non-destructive output

- There is one more pipe worth mentioning
- You can append the output of a command to a file using the ">>" directive

[student@hostname student]$ ls >> file-list

- Will stick another copy of the ls at the bottom of file-list
- REMEMBER: ">","<",">>", and "|" are features of the shell

File Permissions

- All files in Linux have *file permissions*
  - Prevent unauthorized users from modifying system settings
  - Protect users files from other users on system
- Every file is "owned" by a user
- When "student" creates a file in their home directory, it is owned by student
- By default, the permissions allow others to read your file, but not edit or delete it

File Permissions (cont.)

- Not only is a file owned by a user, it is also owned by a *group*
- A *group* is a set of users on a system
- Every user is placed into at least one *group*
- A user can be a member of many *groups*
- If there were a university system, there might be four groups: student, staff, faculty, guest

File Permissions (cont.)

- There are three different types of permissions:
  - *read*
  - *write*
  - *execute*
- There are three clases of users who can have these permissions:
  - the owner of the file
  - a group to which the file belongs
  - all users on the system regardless of their group.

File Permissions (cont.)

- Both files and directories have permissions
- Read permission allows
  - A user to read the contents of a file
  - List the contents of a directory
- Write permission allows
  - A user to modify the contents of a file
  - Create new files or delete files within a directory
- Execute permission allows
  - A user to run the file as a program or shell script
  - For directories, allows the user to cd into that directory

Understanding File Permssions

- Using the "ls" command with the "-l" option, we can look at a files permissions

[student@hostname student]$ ls -l stuff

-rw-r--r--1 student student 505 Mar 13 19:05 stuff

- The first field is the file permssion array
- The third field is the owner of the file
- The fourth field is the group that owns the file

Understanding File Permssions (cont.)

-rw-r--r--1 student student 505 Mar 13 19:05 stuff

- The "-rw-r--r--" is interpreted as follows
  - The first character, "-" is the type of file. "-" means regaral file, "d" means directory, etc.
  - The next three characters are the owner's permssions
  - Then three characters are the group's permissions
  - The last three are everybody else's permissions

Understanding File Permssions (cont.)

-rw-r--r--1 student student 505 Mar 13 19:05 stuff

- "r" stands for read permission
- "w" stands for write permission
- And "x" stands for execute permssion
- If a given owner, group, or everybody else does **not** have permssion, the letter is substituted using the "-" character

Understanding File Permssions (cont.)

-rw-r--r--1 student student 505 Mar 13 19:05 stuff

- This file, the owner has read and write permission ("rw-")
- The group only has read permission ("r--")
- And everybody else only has read permission ("r--"
- Nobody has execute ("x") permssion because this file is not a program or shell script

Understanding File Permssions (cont.)

-rwxr-xr-x

- Here, the owner has read, write, and execute permission whereas both the group and everybody else has read and exectute, but does not have write permission

-rw-------

- Here, only the user had read and write permission. Nobody (except root) can access this file on the system

-rwxrwxrwx

- Here, everybody has read, write, and execute permission on this file

Changing Permissions

- The command to change a file permission is "chmod" for change mode
- *chmod's* first argument is a combination of these three sets
  - First character is a member of {a,u,g,o}
  - Second is a member of {+,-}
  - Third is a member of {r,w,x}

Changing Permissions

- The command to change a file permission is "chmod" for change mode
- In the group {a,u,g,o}
  - "a" stands for all threee of the ones below
  - "u" stands for the owner (or user) of the file
  - "g" stands for the group which owns the file
  - "o" stands for everybody else

Changing Permissions

- The command to change a file permission is "chmod" for change mode
- In the group {+,-}
  - "+" means to give a permission
  - "-" means to take a permission away
- In the group {r,w,x}
  - "r" means read permission
  - "w" means write permission
  - "x" means execute permission

Changing Permissions

- The command to change a file permission is *chmod* for change mode
- With a combination of the first group, second group and third group, you can grant or revoke permissions. Examples:

chmod a+r - gives read access to the file

chmod +r - same as above, if no {a,u,g,o} specified, "a" is assumed

chmod og-x - removes execute permission from other and group

chmod u+rwx - gives the owner read, write, and execute the file

chmod o-rwx - revokes read, write, and execute for others

File Links

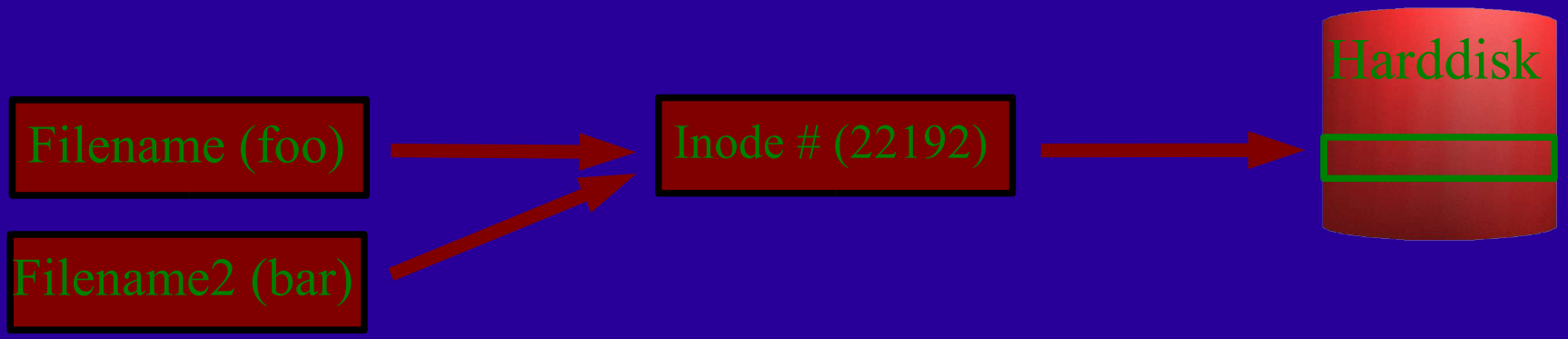
- There are two types of file links
  - Hard links
  - Symbolic links
- Originally, hard links were the only type of link

Hard Links

- Every file on the system has a special number associated with it (*inode number*)
- Each directory contains a list of *inode numbers* that it contains
- A *filename* is a *hard link* to an *inode number*

Hard Links (cont.)

- The "ln" command allows you the create a *hard link* to an *inode number*
- A file can have multiple *hard link*s pointing to it



Hard Links (cont.)

[student@hostname student]$ ls -i

- Will give you a liting of the inode numbers associate with each filename in the diretory

[student@hostname student]$ touch foo

[student@hostname student]$ ls -i foo

22192 foo

[student@hostname student]$ ln foo bar

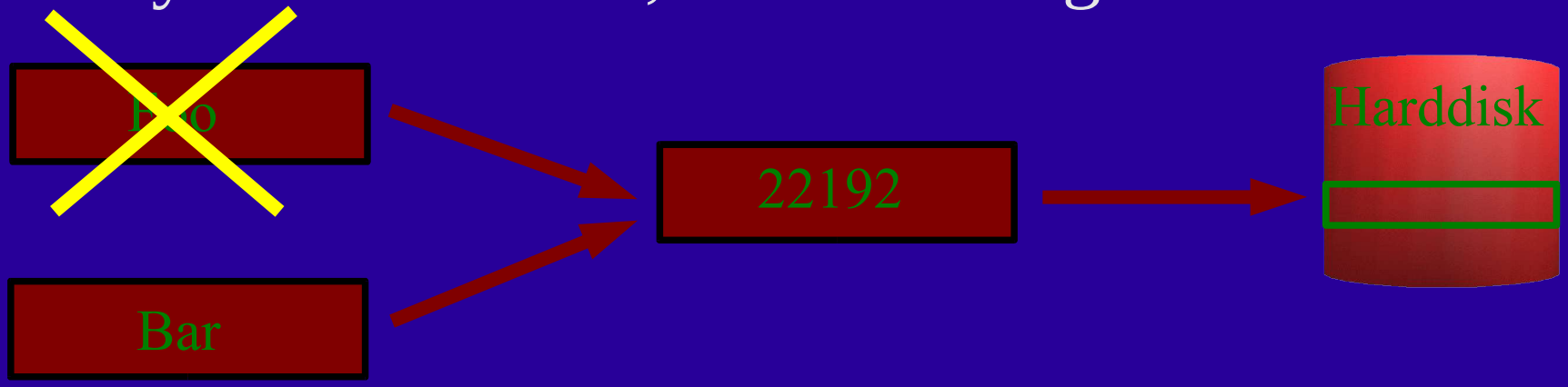
[student@hostname student]$ ls -i foo bar

22192 bar 22192 foo

- Note that your inode numbers will be different

Hard Links (cont.)

- Now, accessing foo or bar will access the same file on the hard drive
- If you modify foo, bar will change as well
- If you remove foo, bar will *not* get removed



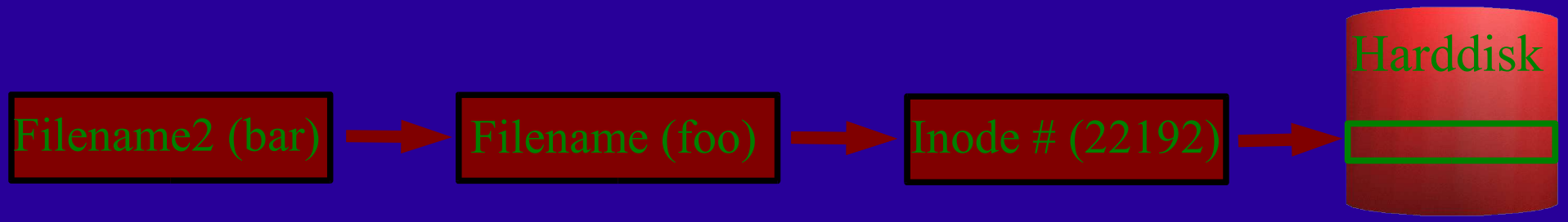
Hard Links (cont.)

[student@hostname student]$ ls -l foo bar

- Now, look at the second item in the list. This is a count of the number of hard links (or references) to the file
- "foo" and "bar" both have 2 references, whereas other files on the system have only 1

Symbolic Links

- The second type of link is the symbolic link
- This is similar to the shortcut in Windows
- Often called a *symlink*
- Does **not** link by *inode number*
- Links to another *filename*



Symlinks (cont.)

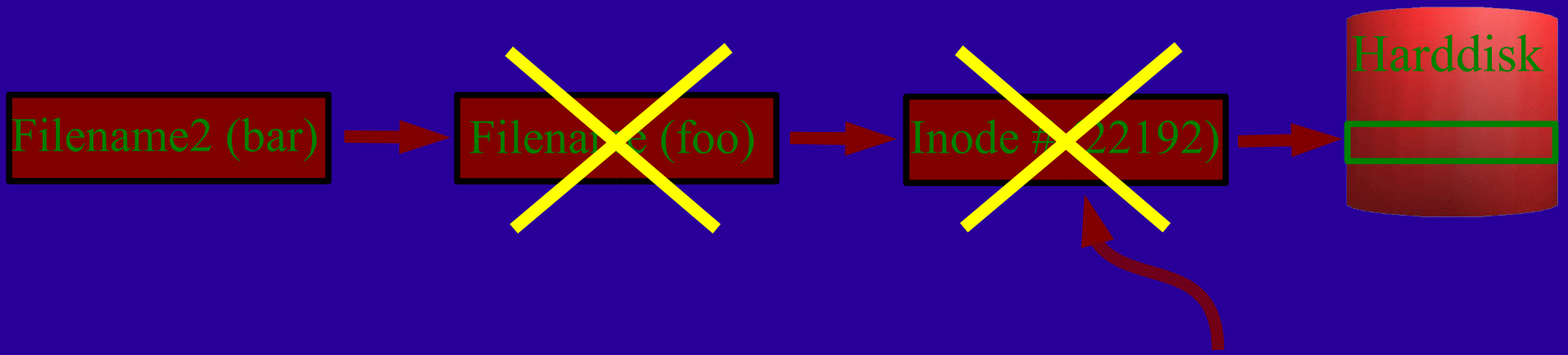
- Use "ln -s" to create a *symlink*

```
[student@hostname student]$ ln -s foo bar
[student@hostname student]$ ls -i foo bar
22195 bar       22192 foo
[student@hostname student]$ ls -l foo bar
lrwxrwxrwx    1 student    student       3 May 8 18:21 bar -> foo
-rw-r--r--    1 student    student      12 May 8 18:20 foo
```

- File permissions are not used on *symlinks*
- Notice the "l" in front indicating this is a *symlink*

Symlinks (cont.)

- If you delete the linked file, the symlink is "broken"



- Now, "bar" is no longer valid
- "ls" will show "bar" blinking red

Linux automatically deletes inodes with no hardlink

Link Differences

- *Hard links* and *Symbolic links* are functionally the same in many respects
- Differences include:
  - Can create *symlinks* to files that don't exist
  - Processed differently by the OS
  - *Symbolic links* identify the file they point to
  - No easy way to tell what *hard links* link to the same *inode number*

Processes and Jobs

- When you run a command or program on Linux, the running program is called a *process*
- To view the running *processes*, use the command "ps"

```
[student@hostname student]$ ps
  PID TTY          TIME CMD
13310 pts/4    00:00:00 bash
6680 pts/4    00:00:00 ps
```

Processes (cont.)

13310 pts/4 00:00:00 bash

- The first filed is the *PID* or *process ID*
  - Unique number to identify the *proccess* in the system
- The last field is the *command* that was run
- We only see the *processes* that student ran from this shell

Processes (cont.)

[student@hostname student]$ ps -aux

- Will list all the *processes* running on the system

Job Control

- Every time you run a command in a shell, you issue a *job*
- A *job* is a *process* that is run from a shell
- In order to manipulate these *jobs*, you need *job control*
- *Job control* is a feature provided by the shell
- Allows you to switch between independent *jobs*

Job Control (cont.)

- Most of the time, you run one *job* at once
- With *job control*, you can run many things at once from one shell
- Example:
  - If you are editing a text file and want to run another command, you can suspend the text editor, issue the command, and come back without having to quit the editor

Job Control (cont.)

- A running *job* is either in the *Foreground* or *Background*
- Only one *job* can be in the *foreground* at once
- The *job* in the *foreground* is the one that you interact with
  - Takes input from keyboard and puts output to screen
  - Unless you have redirected input or output (remember pipes)

Job Control (cont.)

- *Jobs* in the *background* run without interaction
  - No user intervention required
  - Run for a long time
- A *job* can be *suspended*
- A *suspended* job is temporarily stopped
- You can resume a suspended job either in *foreground* or the *background*
- To *suspend* a job, issue a [Ctrl-Z] while it is in the *foreground*

Job Control (cont.)

- Instead of *suspending* a job, you can *inturrupt* it
- This will kill off the job
- Once killed, you **cannot** resume the job
- Use [Ctrl-C] to *inturrupt* a job

Job Control (cont.)

- Examples of job control
- The command "yes" simply prints "y" forever until it is *interrupted*

```
[student@hostname student]$ yes
y
y
y
y
y
```

Job Control (cont.)

- Using [Ctrl-C], *inturrupt* "yes"
- Instead of seeing all the "y"s, let's redirected *standard output* of "yes"
- There is a special file on the system called /dev/null
- /dev/null is a "black hole"
- If you send output to /dev/null, it disappears

Job Control (cont.)

[student@hostname student]$ yes > /dev/null

- This redirects the "y"s to the black-hole. We don't see any more on the screen
- "yes" is still running in the foreground
- [Ctrl-C] will kill it

Job Control (cont.)

- We can make a *process* run in the *background* by using the special character "&"

[student@hostname student]$ yes > /dev/null &

[1] 164

- You are now returned to the shell
- The shell also prints out "[1] 164" or something similar
- The "164" refers to the *process ID* or *PID*
- The "[1]" refers the *job number*
  - The shell asigns a *job number* to all running *jobs*

Job Control (cont.)

- The program "yes" is running in the background
- It is continuously sending a stream of "y"s to /dev/null
- We can see the status of all our *jobs* by running the command "jobs"

```
[student@hostname student]$ jobs
[1]+       Running              yes > /dev/null &
```

- "ps" will also show it running

Job Control (cont.)

- To *terminate* the *job*, use the "kill" command
- "kill" takes either a *process ID* or a *job number* as the first argument
  - Use the "%" in front of the number to refer to a *job number*
  - No "%" means *PID*

```
[student@hostname student]$ kill %1
```

- This will kill off job 1

```
[student@hostname student]$ jobs
[1]+       Terminated          yes > /dev/null
```

Job Control (cont.)

- You could have used the *PID*

[student@hostname student]$ kill 164

- Which is the same as

[student@hostname student]$ kill %1

- Since they both refer to the same *process*

Stoping and Starting Jobs

- There is another way to put a job into the background
- Once you start a process, suspend the job with [Ctrl-Z]

```
[student@hostname student]$ yes > /dev/null
[Ctrl-Z]
[1]+  Stopped                 yes > /dev/null
```

Starting and Stopping Jobs (cont.)

- To bring the job back into the foreground, use the "fg" command
- To put the job into the background, use the "bg" command

[student@hostname student]$ fg

yes > /dev/null

[Ctrl-Z]

[student@hostname student]$ bg

[1]+ yes > /dev/null &

Starting and Stopping Jobs (cont.)

- You can use "jobs" to see it running in the background
- If you want to stop the job again, you must bring it back to the foreground
- Use the "fg" command to do that
- The shell will redisplay the name of the command so you know which one you are restarting

Starting and Stopping Jobs (cont.)

- Note: difference between a backgrounded job and suspended job
  - Background: still running using CPU time and memory
  - Suspended: not running, not using CPU, not doing any work
- A job in the background can still try to display output to screen

```
[student@hostname student]$ yes &
y
y
```

Starting and Stopping Jobs (cont.)

- You can't [Ctrl-C] because the job is in the background
- In order to stop it, you must first bring to foreground
- You will need to type even though stuff is streaming past the screen

[student@hostname student]$ fg

[Ctrl-C]

Starting and Stopping Jobs (cont.)

- "fg" and "bg" only affect the last job that was stopped
- When you type "jobs", the one with a "+" next to it is the job that "fg" and "bg" will affect
- If you want to affect another job, you can use the "%" <number> to address it

```
[student@hostname student]$ fg %s
```

- This will bring job 2 to the foreground

Job Control (final notes)

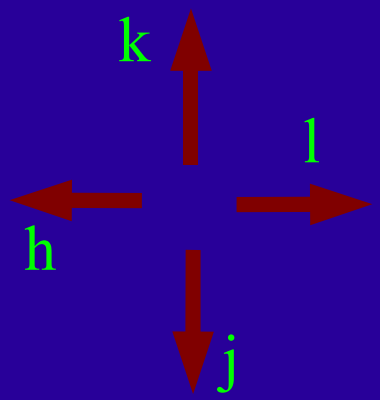
- Job control is a feature of the shell
- There are differences between bash and tcsh
- Some shells do not have job control
- The main one's on Linux do

The VI Editor

- VI is the main editor in UNIX and Linux
  - VI stands for "visual editor"
  - Available on all *NIX systems
  - Not easy to use
  - Not self-explanitory
  - Many, many commands
  - Have I sold you yet? :-)
- Linux uses an enhanced version of VI called VIM (VI Improved)

VI (cont.)

- Normally to start, run "vi"
- Today, run "vimtutor" to start tutorial
- Read along in "VIM Tutor" handout
- h,j,k,l keys used to move around



- Alternatively, use the arrow keys (not in all vi's)

VI (cont.)

- VI is a modal editor
  - While in a mode, VI can only do things allowed in that mode
  - At any given time, vi is in a mode
    - Normal mode
    - Insert mode
    - Replace mode
    - Last line mode
- Hit [Esc] to get to Normal mode

VI Modes

- Normal Mode
  - Moving around the text
  - Entering *normal mode* commands
  - Issuing commands to get into other modes
- Insert Mode
  - Can enter text into your document
- Replace Mode
  - Overwrites text in your document

VI Modes (cont.)

- Last Line Mode
  - Issue filesystem commands
    - Save file
    - Open file
    - Instert file
  - Quit VI
  - Run commands to manipulate your text

VIM 1.2

- Quiting VI
  - Make sure you are in *Normal mode* (hit [Esc])
  - Enter the *Last line mode* by hitting ":"
  - Notice that the cursor is now at the bottom of the screen
  - In last line, "q" [Enter] quits
  - Use "q!" [Enter] to quit without saving
- OK, now type "vimtutor" again to get back in

VIM 1.3

- Delete a character under the cursor, type "x" in *Normal mode*
- Move down to the line marked with the "--->"
- Fix up the sentence "The ccow jumpedd ovverr thhe mooon."

VIM 1.4

- To enter *Insert Mode*, type "i"
- Once in *Insert Mode*, you must hit [Esc] to get back to *Normal Mode* (to move around, etc.)
- Move down to the first line marked "--->"
- Correct the first sentence to match the second

---> There is text misng this .

---> There is some text missing from this line.

VIM 1.4 (cont.)

- Remember, you can always correct a mistake by using the "x" command from *Normal Mode*
- Also, when in *Insert Mode*, you can erase the inserts you just made by using the [Backspace] key

VIM 2.1

- Make sure you are back in *Normal Mode* (hit [Esc])
- The command "dw" will delete a word
- Move down the sentence and delete the words that don't belong
- Must put cursor on the first letter of the word to delete it properly

---> There are a some words fun that don't belong paper in this sentence.

VIM 2.2

- Make sure you are back in *Normal Mode* (hit [Esc])
- The command "d$" will delete the rest of the line to the end
- Move down the sentence and delete the double ending.
- Must put cursor on the space (" ") to delete that character as well as the rest of the line

---> Somebody typed the end of this line twice. end of this line twice.

VIM 2.3

- The "d" command followed by another character (object) will delete something
- The something is determined by the second character (object)
  - w – from cursor to end of word including space
  - e – from cursor to end of word, NOT including space
  - $ - from cursor to end of line
- SYNTAX: [number] d object OR d [number] object
- [number] will modify "d" to do it that many times
- [number] is optional, and will default to "1"

VIM 2.4

- There is an exception to the "d" rule
- "dd" will delete an entire line
- [number] d d will delete [number] of lines
- Edit the poem in the tutorial to delete the bogus lines

1) Roses are red,
2) Mud is fun,
3) Violets are blue,
4) I have a car,
5) Clocks tell time,
6) Sugar is sweet
7) And so are you.

VIM 2.5

- To undo something you have just done incorrectly, use the "u" command
- Use "x" to delete an unwanted character
- Now use "u" to undo it
- Now fix all the errors in this sentence

---> Fiix the errors oon thhis line and reeplace them witth undo.

- Now, use the command capital "U" to undo the whole line

VIM 2.5 (cont.)

- Use lowercase "u" a few times to undo the "U"
- [Ctrl-R] will redo an undo
- Now, type [Ctrl-R] to redo the command

VIM 3.1

- Using the "p" command, you can Put or Paste the last thing deleted
- Rearrange these lines into the correct order by using "dd" to delete a line
- Then move the cursor and use "p" to put it back in the correct place

d) Can you learn too?

b) Violets are blue,

c) Intelligence is learned,

a) Roses are red,

VIM 3.2

- Using the "r" command, vi will replace a character under the cursor
- Move the cursor over the incorrect letters
- Then hit the "r" key followed by the correct letter
- Edit the first line to match the second line

---> Whan this lime was tuoed in, someone presswd some wrojg keys!

---> When this line was typed in, someone pressed some wrong keys!

VIM 3.3

- Using the "cw" command, we can correct entire words
- Place the cursor in the "u" in "lubw"
- Use "cw" to correct work and type "ine"
- Hit [Esc] and move onto the next work until the sentences match

---> This lubw has a few wptfd that mrrf changing usf the change command.

---> This line has a few words that need changing using the change command.

- "cw" will not only replace a word, but it puts you in *Insert Mode*

VIM 3.4

- The "c" command has similar syntax to the "d" command
- SYNTAX: [number] c object OR c [number] object
- Objects:
  - w (word)
  - $ (end of line)
- Fix the following by "correcting" the rest of the first line with the second using "c$"

---> The end of this line needs some help to make it like the second.

---> The end of this line needs to be corrected using the c$ command.

VIM 4.1

- [Ctrl-G] will print print out the filename you are editing and the line number you are currently at

"/tmp/tutorZrieCm" [Modified] line 392 of 785 --49%-- col 15-29

- Remember your line number
- [Shift-G] will bring you to the end of the file
- Now, type in the line number and then hit [Shift-G] (NOTE: line number will **NOT** appear)
- This will bring you to your line number

VIM 4.2

- Using the "/" command, we can search the document
- A "/" should appear at the bottom of the screen like *Last Line Mode*
- Type "errroor" to search for this word (followed by [Enter])
- To repeat the search, use the "n" command
- To search backwards, use the "N" command
- If you want a backward search from the beginning, use the "?" instead of the "/" command
- Note that when search reaches end, it will continue at top

VIM 4.3

- Using the "%" command, vi will match parentheses (, [, or {
- Place the cursor over either the open or close parentheses in the following sentence
- Hit the "%" to see the cursor move to the opposite one

---> This (is a test line with ('s, ['s] and {'s } in it.))

- This is used extensively in programming

VIM 4.4

- Using a *Last Line mode* command, we can perform a search and replace
- Type ":s/old/new/g" to substitute "old" for "new" in a line
- Move down to the line and type ":s/thee/the" to replace the first occurance of "thee" with "the"
- Now, type ":s/thee/the/g" to substitute all occurances of "thee" with "the" on that line

---> thee best time to see thee flowers is in thee spring.

VIM 4.4 (cont.)

- Advanced Search and Replace
  - :#,#s/old/new/g -- will replace all the occurances of old with new between the two line numbers represented by "#"
  - :%s/old/new/g – will replace all the occurances of old with new in the whole file
  - :%s/old/new/gc – will replace all the occurances of old with new in the whole file asking your for confirmation before each sub.

VIM 5.1

- To run a command from within vi, enter the *Last Line Mode* by typing ":"
- Then type "!" followed by a shell command
- Type ":!ls[Enter]" to run "ls"
- The output will come to your screen and you can hit [Enter] to get back to your vi session

VIM 5.2

- To save a file, enter the Last Line Mode by hitting ":"
- Then type "w FILENAME [Enter]" (note the space in between w and FILENAME)
- Make sure the file does not already exist before doing this
- Type ":w TEST[Enter]"
- This saves the tutor file under a new file TEST

VIM 5.3

- You can save part of file instead of a whole file in vi
- Move to a line and type [Ctrl-G] to see that line number
- Now move to a line below and hit [Ctrl-G] as well
- Remember both these numbers
- Now to save the file from the first line to the second line, type ":#,# w TEST2" where #,# is the first number followed by the second number

VIM 5.4

- To merge another file into your current file, you can use the "r" command from the *Last Line Mode*
- Type ":r TEST2" to insert the file TEST2 into your file
- Note the space between the "r" and "TEST2" in the *Last Line* command

VIM 6.1

- You can use the "o" command to open a new line *below* in *Insert Mode*
- Make sure to hit [Esc] to be in *Normal Mode*
- Move the cursor to the line marked with the "--->"
- Hit "o" and copy the line. Hit [Esc] when done to put back into *Normal Mode*

---> After typing o the cursor is placed on the open line in Insert mode.

- To open the line *above* the cursor, you can use the capital "O" command

VIM 6.2

- Use the "a" command to append text *after* the character under the cursor
  - as apposed to "i" which inserts *before* the character
- Move the cursor down to the first "--->" line
- Use the "$" by itself in Normal Mode to go to the end of the line
- Now type "a" to start appending to the line

---> This line will allow you to practice

---> This line will allow you to practice appending text to the end of a line.

VIM 6.2 (cont.)

- You can use the capital "A" command to automagically append to end of a line
  - Instead of having to type "$" and then "a"
- Note that when you append, you go into *Insert Mode*, just that it starts *after* the character rather than *before*

VIM 6.3

- In order to get into *Replace Mode*, you type the command capital "R" from *Normal Mode*
- Move the cursor to the before the word "last" in the first sentence
- Hit "R" and replace the remainder of the sentence with the one below it

---> To make the first line the same as the last on this page use the keys.

---> To make the first line the same as the second, type R and the new text.

- Hit [Esc] when done to get back to *Normal Mode*

VIM 6.4

- VI's default behaviour can be modified by using the "set" command from within *Last Line Mode*
- Normally, when you search, the search is case sensitive.
- We can turn this off by typing ":set ic" for Ignore case
- Now, a search for ignore ("/ignore") will return all matches, disregarding case
- Hit "n" to get to the next match
- Type in ":set hls is" to tell search to highlight all matching items

VIM Help

- If you need help with VI, it has online docs you can access by
  - Hitting [F1] from within vi
  - Typing ":help [Enter]"
- Use ":q[Enter]" to quit the help
- You can get help on specific topics by typing ":help <topic>[Enter]"

Second Field in Directory listings

- Last class, somebody asked what the second field in directories for "ls -l" was
- I incorrectly said it was the number of files in that directory
- It is actuallyis the number of sub directories in that directory
- Will always have at least 2 since "." and ".." are in every directory

The root Account

- System defined account
- No restrictions
  - read, modify, or delete any file on system
  - change permissions, ownership of any file
  - run special programs
    - ones that modify partitions on the hard drive
    - create filesystems
- easy to make accidental mistakes
  - sit on your hands before doing something as root

The root Account (cont.)

- Normally, root has a different character for a prompt
  - users have a $
  - root has a #
- Log in a root only when necessary
- When you are done as root, log out

The root Account (cont.)

- Use the "su" command to swith to root
- "su" stands for switch user
- You can change to user without having to log out
- "su" can take any user as an argument, but root is the only one who can switch to any user
- "su" will prompt you for the root password
- use a single dash after su to load root's environment (otherwise, it will keep your environment)

$ su -

Password:

Booting Linux

- When Linux boots, it runs the kernel
- Mounts the root device (/dev/hda# normally)
- LILO is the older boot program for linux
  - configuration file found at /etc/lilo.conf
- Grub is newer
  - configuration file found at /etc/grub.conf

Shutting Down Linux

- Never just "turn off" a Linux system
- as root
  - # shutdown 20:00 - will shutdown at 8:00pm
  - # shutdown -t 10 - will shudown in 10 seconds
  - # shutdown now - will shotdown now
  - # halt - same as "shutdown now"
- When system says "The system is halted", you can turn the power off
- [Ctrl]-[Alt]-[Del] will reboot nicely on Linux

System Bootup

- When the system boots up, it will start either in graphical or text-only modes
- When system first boots, it runs "init"
  - responsible for running all startup scripts
  - brings system to "multiuser" mode
- init program uses configuration file /etc/inittab
- NOTE: copy /etc/inittab before modifying it -- editing incorrectly can cause system to not boot correctly

/etc/inittab

- By default, /etc/inittab starts 6 virtual consoles
- It starts a default *runlevel*
  - example line looks like "id:3:initdefault:"
  - 1 is single user mode
  - 3 is text-only mode
  - 5 is graphical mode

Mounting File Systems

- Before a filesystem is accessible to the system, it must be mounted
- Includes hard drives as well as floppy drives, cdrom drives, etc.
  - /mnt/floppy is normally where you mount floppies
  - /mnt/cdrom for cdroms
  - /mnt/zip for zip drives
- Before mounting the file system, the directory, or *mount point*, will be empty

Mounting File Systems (cont.)

- System automagically mounts the "root file system" on bootup
- Also mounts /usr, /var, /home, etc. if they are on different partitions or hard drives
- "mount" used to mount file systems
- "umount" (note the lack of a "n") used to unmount file systems

Mounting File Systems (cont.)

- /etc/fstab contains default file system *mount points*
- First field is the device (normally from /dev directory)
- Second is the *mount point*
- Third is the file system type (ext2, msdos, etc.)
- Fourth are mount options
- Fifth field is used by dump (don't worry about it)
- Sixth field is what order to check the filesystems for errors on bootup

Mounting File Systems (cont.)

- /proc needs to be mounted as it is a "virtual file system" with no real hard drive underneath it
- swap is for all swap partitions (called virtual memory in Windoze world...incorrectly)
- Must be root to use mount

mount -t <type> /dev/<device> /<mountpoint>

umount /dev/<device> -- same as

umount /<mountpoint>

Managing Users

- You should at least create one user account for yourself to use on a regular basis
- Every person using the computer should have their own account
- User accounts are stored in /etc/passwd

Managing Users

- Every user has
  - user name -- normally 2-8 characters
  - user ID -- unique number for the user
  - group ID -- the user's default group
  - password -- users encrypted password
  - full name -- users real name
  - home directory -- where the users files go and where the user starts when logging in
  - login shell -- the shell that gets run when the user logs in

Managing Users

- /etc/passwd contains these values separated by ":"
- stored as:

user name:encrypted password:UID:GID:full name: home directory:login shell

- an example:

tduffy:x:1057:1502:Tom Duffy:/home/tduffy:/bin/bash

- Newer versions of Linux don't store the encrypted password in /etc/passwd (that is why there is an "x" there)
- Now, they are stored in /etc/shadow only readable by root

/etc/group

- /etc/group contains a list of Group ID's or GID's
- stored as:

group name:password:GID:other members

- an example:

instructors:x:1502:tduffy,rgreen,rms

- Use "addgroup" or "groupadd" to add a new group to the system

Adding Users

- A user needs an entry in /etc/passwd
  - unique UID
  - specify GID, full name, etc.
- home directory should be created and permissions set on that directory so that the user owns it
- Default "dotfiles" should be copied to user's home directory
- Other stuff such as email, etc should be setup

Adding Users (cont.)

- Normally, not all done by hand
- Can use "adduser" or "useradd" (the same program)
- uses config file /etc/default/useradd
- "userdel" or "deluser" will delete a user
- You can temporarily disable a user account by putting a "*" in the password field of /etc/passwd

Adding Users (cont.)

- After adding a user, change his/her passwd

passwd larry

- The user can change their passwd when they log in
- Can also change their shell with "chsh"
- Can change their full name with "chfn"

Software Management

- Software in most versions of Linux is managed with "rpm"
- "rpm" means RedHat Pacakage Manager, but is on many other distributions of Linux

# rpm -qa	-- list all packages installed
# rpm -i <pacakge>	-- install a package
# rpm -U <package>	-- upgrade a package
# rpm -e <package>	-- remove a package

Device Drivers

- Most device drivers under linux are *kernel modules*
  - Loaded into kernel while running
- Use "lsmod" to list the drivers installed in system
- Many are extra features added into kernel, but some are drivers
- "insmod" will load a driver
- "rmmod" will remove a driver

Device Drivers (cont.)

- /etc/modules.conf contains information about what drivers are used for what pieces of the system
- "eth0" is the first ethernet (network) card in the system
- "sound-slot-0" is the first sound card in your system

alias <device> <driver>

- file also contains driver options

Emergency Recovery

- DON'T PANIC :)
- Always have a "recovery disk"
  - created when installing Red Hat
- Alternatively, boot Red Hat CD 1 and type "linux rescue" at first prompt
- Keep backups of important files (always good advice)
- When all else fails, search the web or email linux mailing lists for help

Emergency Recovery (cont.)

- If you forget your root passwd
  - reboot system
  - before system starts linux
  - type [Ctrl]-[r] to get to lilo prompt and type "linux 1"
  - will come up in single user mode and log you in as root automagically
  - use "passwd" to change root's password

Emergency Recovery (cont.)

- If your file system gets damaged and the regular check cannot fix it
  - normally, if the boot up cannot fix your file system automatically, it will error and drop to a root shell
  - type "fsck" for file system check followed by the *root file system* device
  - eg, # fsck /dev/hda5
  - This will ask many questions. It is ok to answer "yes" to everything
  - Can do this automatically, by typing "# fsck -y /dev/hda5" where /dev/hda5 is your *root file system*

Emacs

- Alternative to VI
- Lost of commands, but has meny system
- Kitchen sink of programs

Other Useful Commands

- The "find" command will search the file system

$ fine . -name <filetolookfor> -print

- will search the current directory and all subdirs for the file matching exactly what you put in
- if you want to use wildcards, you must escape them
  - escaping characters means putting a "\" in front of it

$ find . -name *<string>* -print

Other Useful Commands (cont.)

- The simpler "locate" command will search the whole hard drive
- Actually searches a database that gets built over night
- New files will not be found
- A LOT fater than "find"

Other Useful Commands (cont.)

- The "grep" command will search through a file for a string

$ grep <string_to_look_for> <filename>

- Will print out every line that the <string_to_look_for> appears on in <filename>

Other Useful Commands (cont.)

- The "uname" command will tell you what version of UNIX you are running
- Give it the "-a" option to print out more info

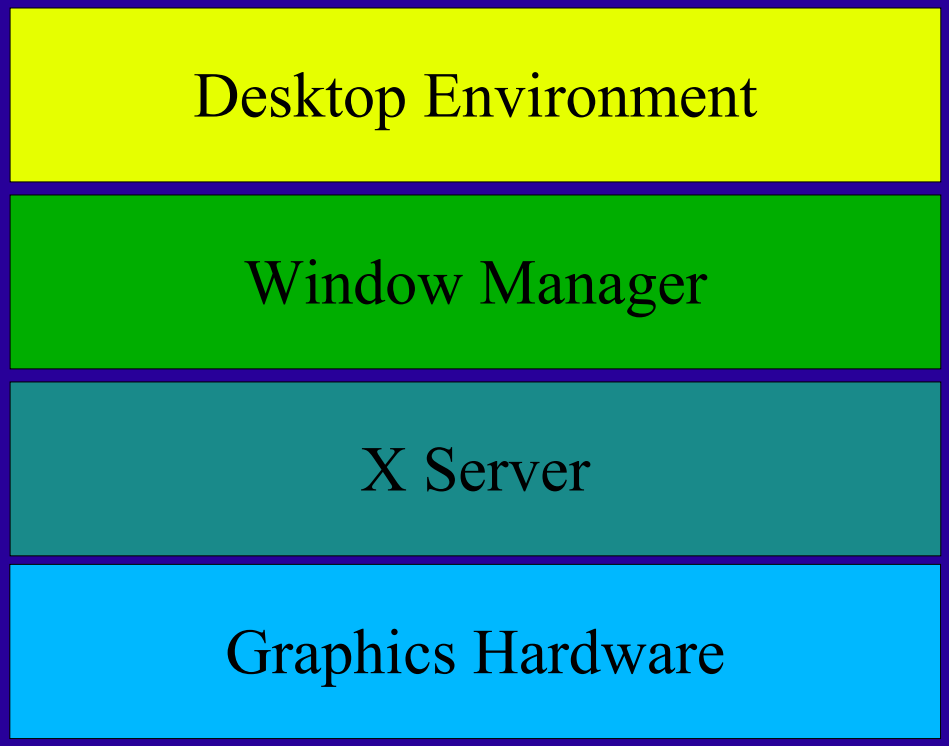
$ uname -a

Linux localhost.localdomain 2.4.9-31SGI_XFS_1.1_PR4 #1 Sun Apr 7 23:06:07 CDT 2002 i686 unknown

- Tells you the type of UNIX, the name of the machine, the version of the kernel and what type of processor you are running on

Linux Graphics Stack

- Graphics Stack

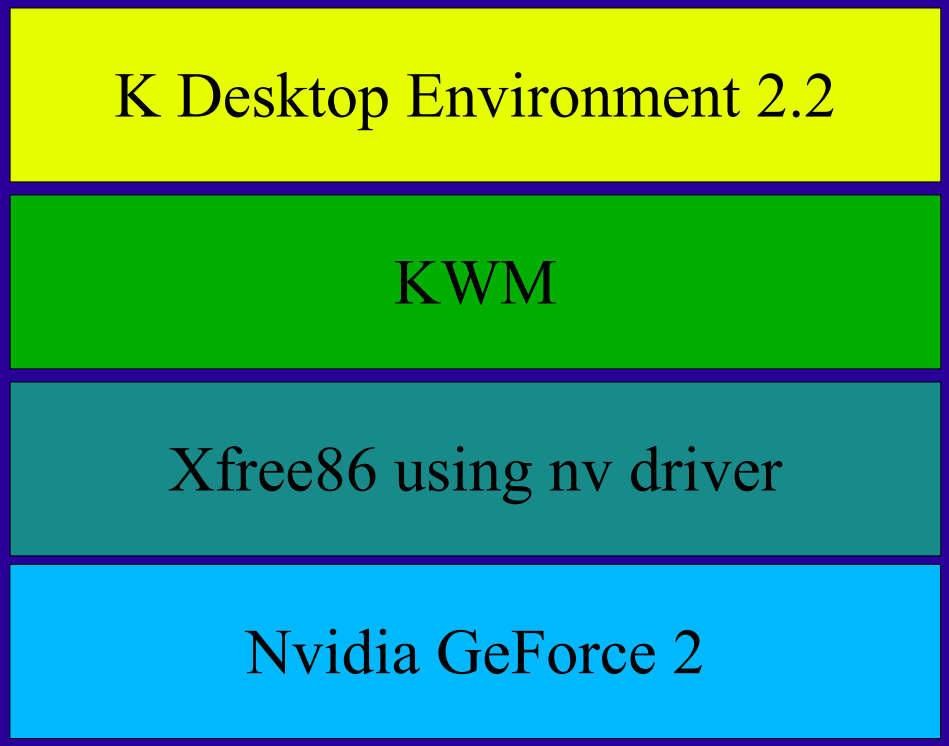


Linux Graphics Stack

- Specific Graphic Stack (GNOME)

Linux Graphics Stack

- Specific Graphic Stack (KDE)



Network Tranparency

- Linux is a multiuser, multitasking environment
- Multiple people can be on a machine at one time
- Traditionally, use "telnet" to get to remote machine
- Nowadays, use "ssh"
  - passwords are encrypted
  - all commands sent over are encrypted
- Must have an account on remote system

Network Tranparency

- In order to telnet or ssh to a machine, you must know the IP address or hostname

$ ifconfig -a

- will print the IP address of your machine

$ ssh -l student 130.212.56.166

- type "student" as Password:
- Now, you are on the instructor computer
- All commands you type affect instructor machine

Your Environment

- In the shell, a way to configure applications and commands is by using environment variables
- Use "env" to print out environment variables in bash
- Use "printenv" in tcsh
- To set environment variables, use

$ export VARIABLE=value

- for bash, sh, ksh

$ setenv VARIABLE value

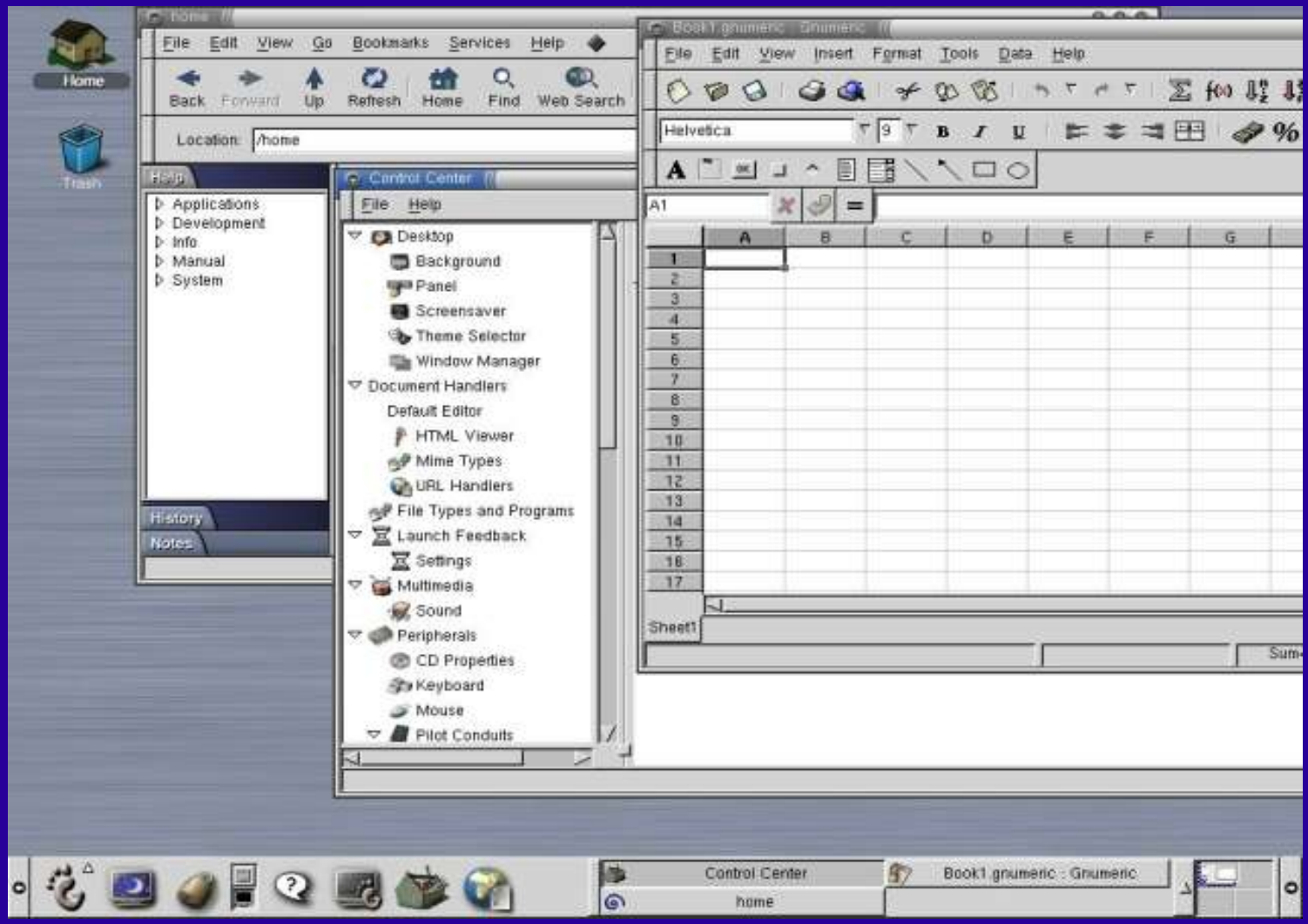
- for tcsh, csh

DISPLAY environment

- A special environment variable called DISPLAY tells all graphical applications where to display
- normaly set to ":0" or "localhost:0"
- if you want to display to another machine, you must set DISPLAY to <ipaddr>:0
- We use the ":0" to denote the first X server running on that machine (normally the case)

$ export DISPLAY=130.212.56.xxx:0

Gnome Screenshot



GNOME

- GNOME stands for Gnu Network Object Model Environment
  - Historic name...don't worry about it really
- Started by Miguel de Icaza
- Wanted to create a Free desktop for UNIX
  - KDE had proprietary tool kit (at the time)

Gnome (cont.)

- Gnome Panel -- long bar at bottom of screen
  - collection of menus
  - panel applets
  - application launchers
- Main Menu -- one with foot on it
  - similar to "Start" menu in Windoze
  - contains launchers for many pre-loaded applications
  - normally item to logout or restart your computer

Gnome (cont.)

- Panel Applets
  - small programs that run in the panel
  - eg. Desk Guide allows you to switch virtual desktops
- Application Launchers
  - buttons that start programs
  - toolbox starts the "Control Center"
  - question mark starts "Help Browser"
  - monitor starts a terminal (command line from within Gnome)
- Arrow on each side of panel hide the panel

Gnome (cont.)

- Desktop is area outside of panel
  - supports drag-and-drop
  - double click on icon (with left mouse button) to launch it
    - If it is a program, application will start
    - If it is data, the appropriate program will start and load the data
    - If it is a folder, the Nautilus file manager will start in that folder

Nautilus

- Nautilus is the file manager for Gnome
  - Manipulate your files (graphically)
  - Left bar has Help, History, Notes, etc.
  - To move, simply drag and drop your folder or file
  - To copy, hold [Ctrl] while dragging and dropping
  - Run a program by double clicking that program or a data file associated with a program
  - Use the right mouse button to do things like "rename", "delete", etc.

Nautilus (cont.)

- To select many items, hold down [Ctrl] and single click additional items
- To move between folders, open another window of Nautilus
- You can drag stuff to Desktop as well

Gnome (cont.)

- Gnome is very configurable
  - can have multiple panels (horizontal or vertical)
  - can have the panels hide automagically
  - many applets that can go into the panel
- Gnome follows several X windows conventions
  - left mouse is used to select and drag items
  - right mouse brings up a menu for a selected item
  - middle mouse button is used to paste text (if in a text area) or to move things
    - middle mouse can be "emulated" by pressing right and left at the same time (if you only have a two button mouse)

Gnome (cont.)

- To cut and paste using the mouse,
  - use the left mouse button to drag across some text to copy
  - move the mouse cursor to the place where you want to paste the text
  - click the middle mouse button (or mouse wheel if you have that)

Gnome (cont.)

- Each window has some buttons on the border to control thewindow.
  - minimize
  - maximize
  - close the window
  - (sometimes) windowshade the window
  - can be configured in the Control Panel component "Window Mananger"

Gnome (cont.)

- Window Manager
  - Gno me uses sawfish as the default window manager
  - Not dependent on any one window manager
  - can use others, but sawfish is most compatible -- A Gnome Compilant window manager
    - sawfish
    - IceWM
    - Enlightenment
    - FVWM2
    - WindowMaker

Gnome Control Panel Walkthrough

- Let's walk through the Gnome Control Panel

KDE

- Originally based on a proprietary widget set
- Now 100% Free as in Speech
- More similar to Windoze
- Gnome and KDE are compatible
  - drag-and-drop works between them
  - docklets work interchangably
  - cut & paste unified
- Your choice...

KDE (cont.)

- Developed in Germany (mostly)
- Works with QT Embedded (Linux PDA OS)
  - applications developed for KDE can be put on PDA as well as other way around
- Uses "K" as the "start" menu
- Koffice is a free office sweet that is integrated with KDE

Bibliography Resources

- J.T.S. Moore, "Revolution OS"

  http://www.revolution-os.com/
- Michael C. Pierce & Robert K. Ware, "VIM Tutor"

  http://cosoft.org.cn/html/documents/english/vim/tutor.html
- Eric Raymond, "The Cathedral and the Bazaar"

  http://www.tuxedo.org/~esr/writings/cathedral-bazaar/
- Neal Stephenson, "In the Beginning was the Command Line"

  http://www.cryptonomicon.com/beginning.html
- Matt Welsh et. al., "Installation and Getting Started Guide"

  http://www.tldp.org/LDP/gs/gs.html