INTRO TO LINUX

Basic Linux proficiency in 24 hours

Rev: 2014-02-24

TECH SPECS

- 24 hours, lecture/lab format
- Hours: 8:30 - 5:00
- Lunch: 11:45 - 1:00
- Breaks every hour or so.. :)

ABOUT THE INSTRUCTOR

- Nathan Isburgh
  - instructor@edgecloud.com
- Unix user 15+ years, teaching it 10+ years
- RHCE, CISSP
- Forgetful, goofy, patient :)

ABOUT THE COLLEGE

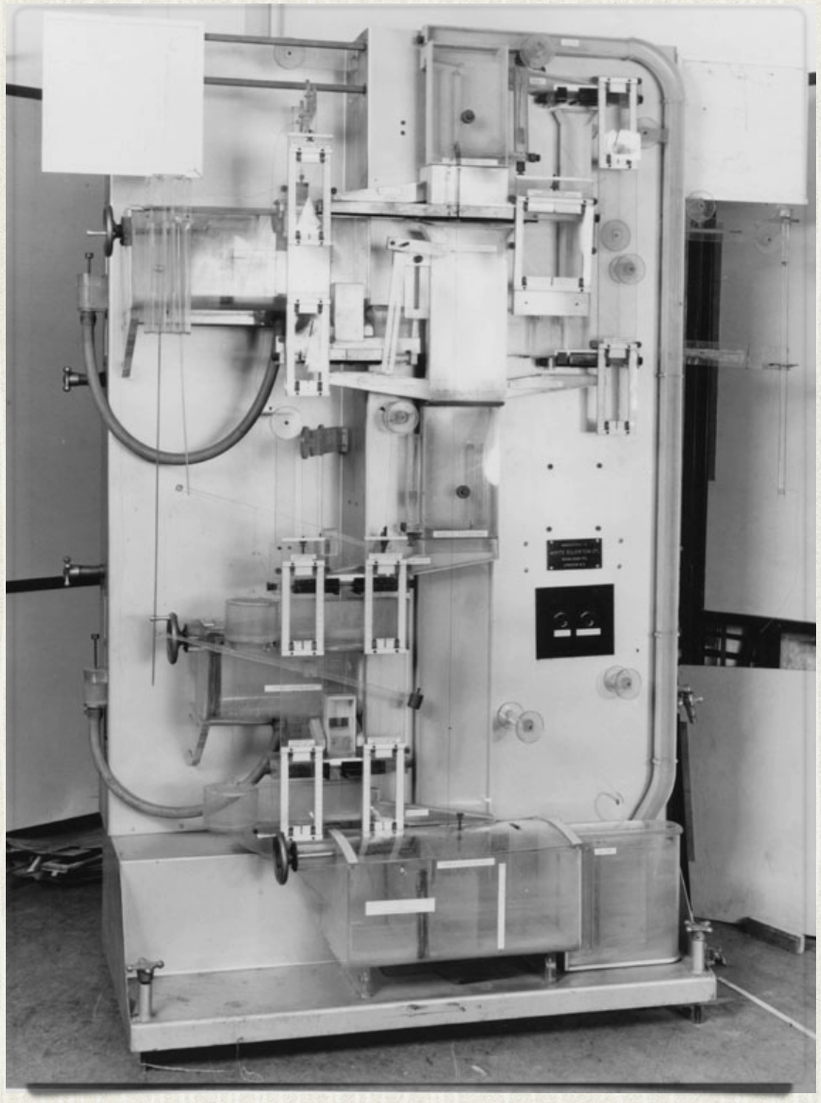
- Rackspace Parking Sticker = good to go
- Breakroom downstairs - labeled “Laundry”
- Sodas - bottles in machine ($1.25) or cans in mini-fridge ($0.50)
- Cafeteria
- Do not speed!
- No smoking anywhere

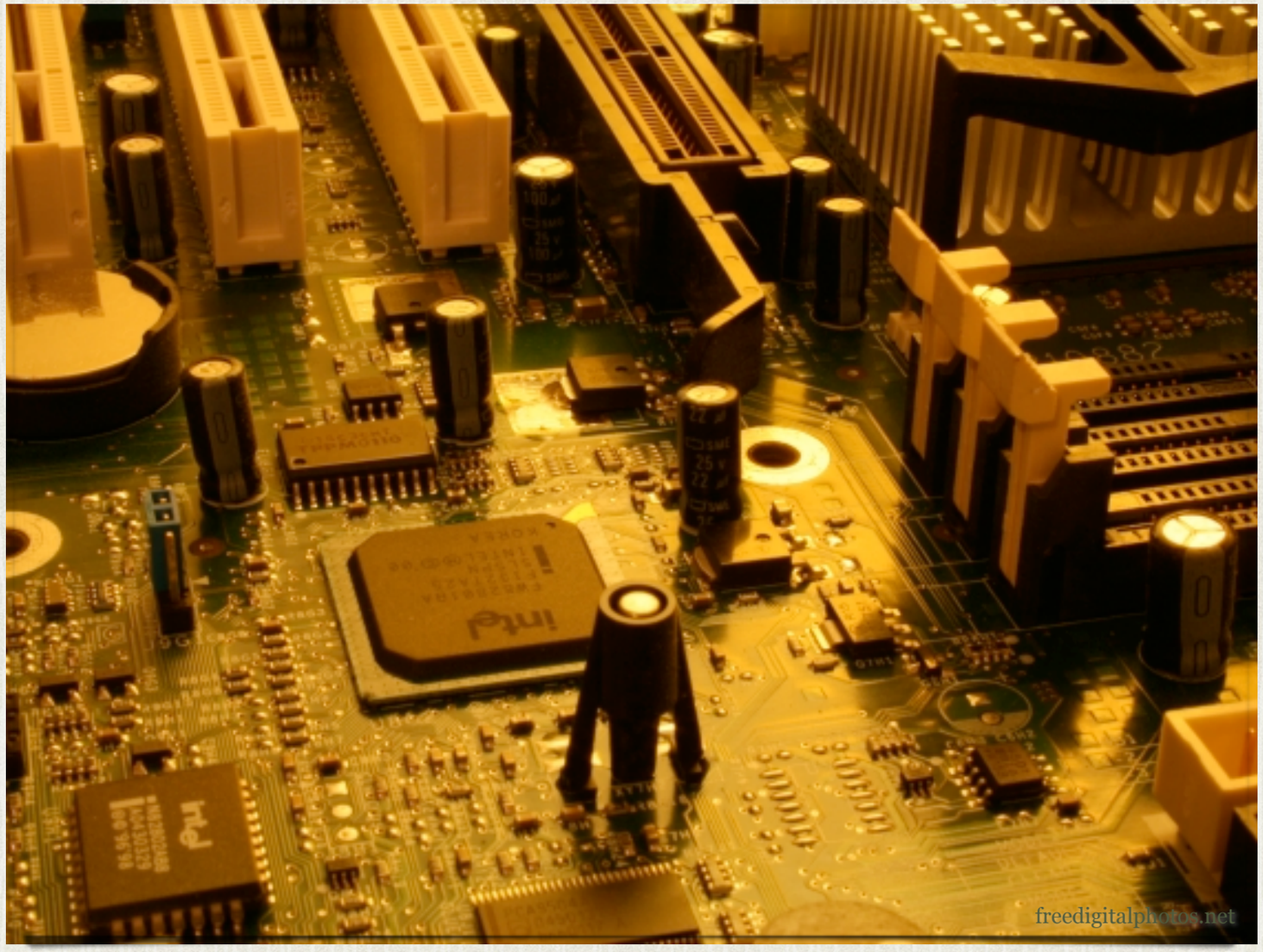
ABOUT THE STUDENTS

- Name?
- Time served, I mean employed, at Rackspace?
- Department?
- Unix/Linux skill level?
- What most interests you about Linux?

EXPECTATIONS OF STUDENTS

- Basic foundation in computer use
- Ask Questions!
- Complete the labs
- Email if you’re going to be late/miss class
- Have fun
- Learn something

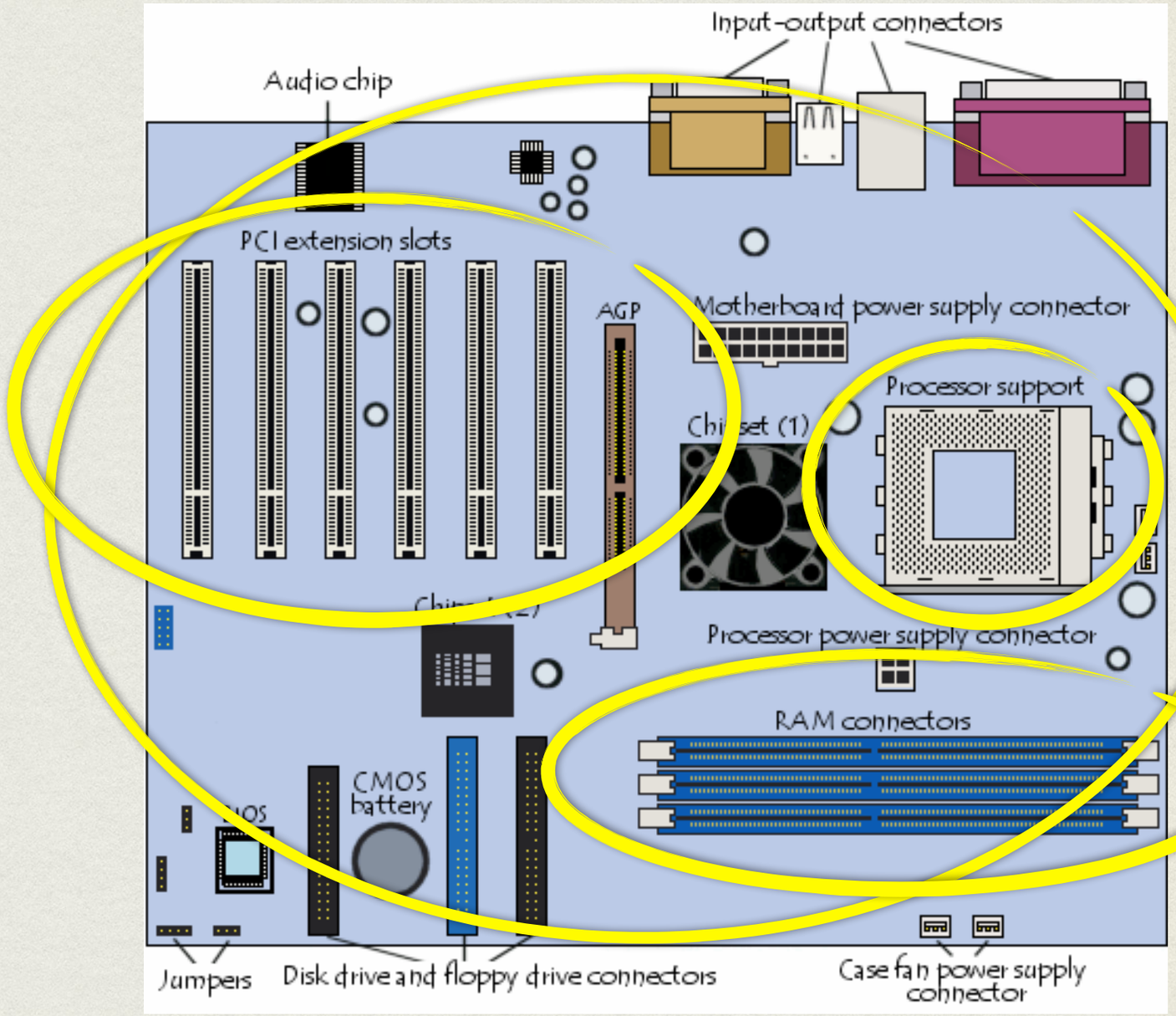




HARDWARE

CORE COMPONENTS

- Motherboard
- CPU
- RAM
- Expansion slots



http://en.kioskea.net/contents/pc/carte-mere.php3

CORE COMPONENTS

- Hard drive
- Removable media drives
- Power supply
- Case

PERIPHERALS

- Keyboard
- Mouse
- Monitor/Video
- Sound
- Printer

RAID ARRAYS

- Redundant Array of Inexpensive Disks
- Stringing together two or more drives
- Provides mix of performance and reliability improvements
- Configured by level…

RAID LEVELS

- 0 (Spanning): Drives simply combined, one after another, to form one large, continuous storage space. No performance or reliability advantages. Used to get large amounts of storage space for cheap.

- 0 (Striping): Drives are combined into one large storage space, but the data is split up and striped across the disks. Provides improved read and write performance through parallel operations. Still no reliability benefit.

RAID LEVELS

- 1 (Mirroring): Each drive in the set is a complete copy of the data. Read performance benefit through parallel read operations. Exceptional reliability benefit through redundancy. Storage limited to size of smallest member.

- 5 (Stripe w/ parity): Most common. Similar to Striped RAID 0, but adds *parity* information, allowing for improved reliability. Minimum 3 members to operate, but can tolerate a drive failure without data loss! Improved performance through parallel operations.

RAID LEVELS

- 6 (Stripe w/ double parity): Same as RAID5, but with doubled parity information, tolerating up to two drive failures in set.

- Levels are often combined (nested) to get the best of different levels: 01, 10, 15, 50, 51, 16, 60, 61

- Nested levels are expensive to implement, but can provide extremely high reliability and performance numbers.

- Common nested levels include…

RAID LEVELS

- 10 (Stripe Set across mirrors): A set that stripes data across two or more RAID1 mirrors.

- 50 (Striped Stripe with Parity Set): Data is striped across two or more RAID5 sets.

- 51 (Mirrored Strip with Parity Set): Data is mirrored across two or more RAID5 sets.

BACKUP MEDIA

- Optical discs: Simple, tough, cheap, small. Limited size. Easy to use.

- Hard drives: Expensive, sensitive. Rapid restore times. Still fairly limited size. Easy to use - often a mirror of the data.

- Tapes: Cheap, reliable, tough. Huge sizes available. Most common backup media for any serious need. Generally requires backup software for managements.

http://www.gnu.org/graphics/gnu-slash-linux.html

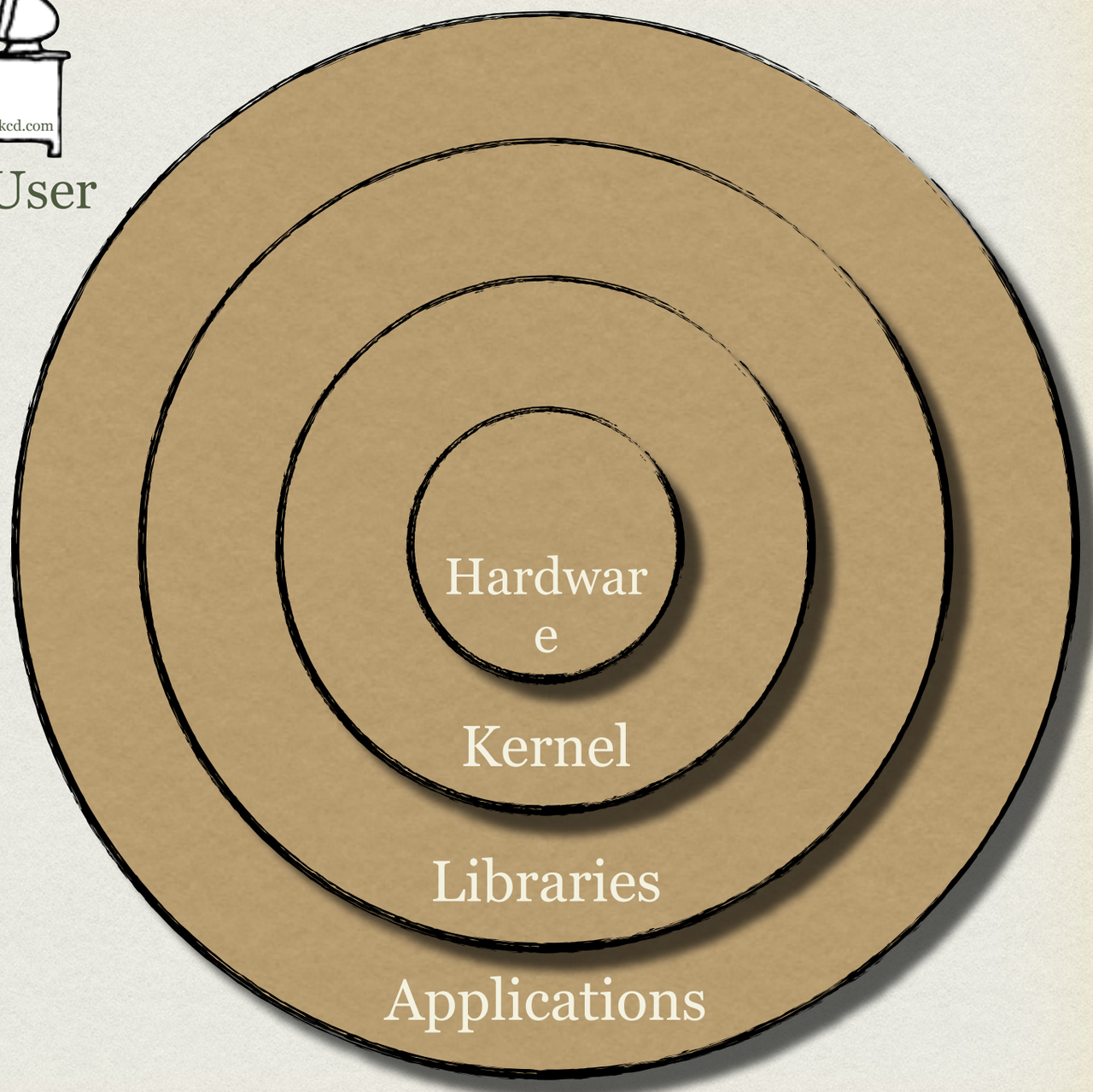
LINUX

The Big Picture

OVERVIEW

- Center of machine
- Scheduler, memory manager, device drivers
- Shared software routines, system calls
- User level software

End User



DISTRIBUTIONS

- The “Linux” part of Linux is the kernel and supporting drivers. By itself, it does not represent a complete operating system.

- Thousands of open source projects combine their powers to form the One True Operating System we know as Linux. :)

- *Distributors* pick and choose from all of this software, combine it with a Linux kernel and package it up into something called a distribution. Common ones include…

DISTRIBUTIONS

- Redhat: One of the oldest and most popular. Originally offered two levels: personal and enterprise. Decided to focus on enterprise offerings, so dropped Red Hat Personal and created the Fedora Project, a community driven entity to produce a personal distribution of Linux.

- Fedora: Aims to release quarterly “Core” distributions. Focuses on up to date software packages and kernels.

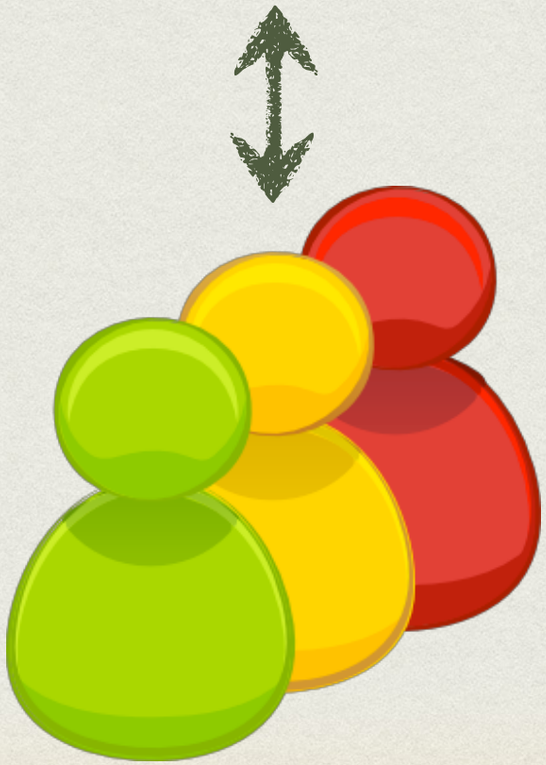
- CentOS: Takes Redhat Enterprise Linux, strips the branding and provides free version.

DISTRIBUTIONS

- Debian: Popular, flexible, apt packaging system
- Ubuntu: Popular for desktops, easy to use, based on Debian
- Gentoo: Focus on performance through targeted, on-the-fly compilation. Unique, advanced, powerful.
- Slackware: One of the first distributions. Meant for advanced users - focus on stability and simplicity.
- 100's of distributions! See http://www.linux.org/dist/

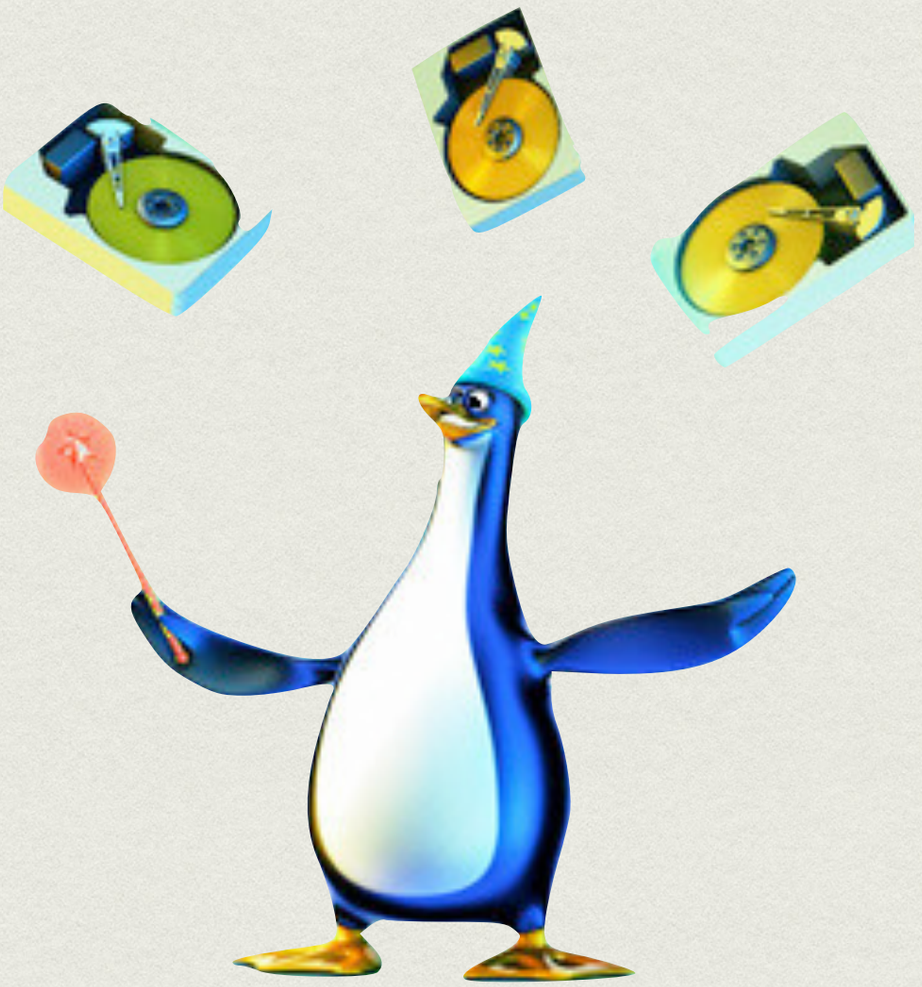
LINUX IS…

- Multiuser
- One of the primary goals of UNIX was to maximize the utilization of the computer (they weren't cheap then!)
- The concept allows multiple users to perform tasks at the same time



LINUX IS…

- Multitasking
- Allowing multiple users necessitates the ability to do multiple things at once.
- Implemented through a complex scheduling system



```
[root@dev1 ~]# ps
  PID TTY          TIME CMD
15844 pts/0    00:00:00 bash
15868 pts/0    00:00:00 ps
[root@dev1 ~]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             9.4G  7.1G  1.8G  80% /
none                  129M     0  129M   0% /dev/shm
[root@dev1 ~]# who
root     pts/0        Nov 28 11:13 (cpe-173-172-107-98.austin.res.rr.com)
[root@dev1 ~]# ls /etc
DIR_COLORS         dev.d          inputrc          makedev.d       php.d          rc4.d          shells
DIR_COLORS.xterm   environment    iproute2         man.config      php.ini        rc5.d          skel
X11                exports        issue            mime.types      pki            rc6.d          smart
adjtime            filesystems    issue.net        mke2fs.conf     pm             redhat-release ssh
aliases            fstab          krb5.conf        modprobe.conf   postfix        resolv.conf    subversion
aliases.db         gpm-root.conf  krb5.conf.rpmnew modprobe.d      ppp            rpc            sudoers
alternatives       group          ld.so.cache      motd            prelink.conf   rpm            sysconfig
apt                group-         ld.so.conf       mtab            printcap       rwtab          sysctl.conf
bashrc             gshadow        ld.so.conf.d     multipath.conf  profile        rwtab.d        syslog.conf
blkid              gshadow-       ld.so.conf.rpmnew my.cnf         profile.d      sasl2          termcap
cron.d             hal            libaudit.conf    netplug         protocols      screenrc       udev
cron.daily         host.conf      libuser.conf     netplug.d       rc             scsi_id.config updatedb.conf
cron.deny          hosts          localtime        nsswitch.conf   rc.d           securetty      vimrc
cron.weekly        hosts.allow    localtime.rpmnew nsswitch.conf.rpmnew rc.local  security       virc
csh.cshrc          hosts.deny     login.defs       openldap        rc.sysinit     selinux        wgetrc
csh.login          httpd          logrotate.d      opt             rc0.d          services       xinetd.d
dbus-1             init.d         lvm              pam.d           rc1.d          sestatus.conf  yum
default            initlog.conf   mail.rc          passwd          rc2.d          shadow         yum.conf
depmod.d           inittab        mailcap          passwd-         rc3.d          shadow-        yum.repos.d
[root@dev1 ~]#
```

SHELLS

Yeah, the hard part of Linux

THE BIG LOOP

- In order to master the shell, you have to understand it's inner workings

- The first concept is *The Big Loop*

THE BIG LOOP

1. Print prompt, await user input
2. Parse and verify input; on error, loop
3. Perform requested operation (execute command, built-in)
4. Loop

MORE ON STEP 2

- Step 2: parse and verify input
- Very important step, includes:
- Syntax checking, command identification, metacharacter substitutions and operations

SYNTAX

- `<command> [options] [arguments]`
- Everything is separated with white space
- The *command* says **what** to do, and has a default behavior
- The *options* say **how** to do it, when behavior beyond the default is necessary
  - Regular options are generally prefixed with a hyphen
  - POSIX options (or long options) use a double hyphen prefix, and often spell out the option with a word rather than just a letter (`--verbose` instead of `-v`)
- The *arguments* specify **what** to act upon

QUOTING

- Generally, arguments are separated with whitespace, but sometimes whitespace needs to be part of the argument itself (spaces in filenames, for example). Consider:
  - `command filename with spaces`
  - Without any guidance, the shell will interpret this input as a command with 3 arguments.
- Quoting is the easiest way to guide the shell in this matter. There are two forms...

SINGLE QUOTES

- Single quotes are the simplest to use:
  - `command 'filename with spaces'`
- The quotes let the shell know where an argument starts and stops (quotes not included), and it doesn't bother with what's between the markers - it is interpreted strictly as data
- Hence, this line would be interpreted as a command with one argument, `filename with spaces`

DOUBLE QUOTES

- Double quotes follow single syntax, but interpret differently:
  - `command “filename with spaces”`
- The quotes let the shell know where an argument starts and stops, but the data in between is loosely examined for *metacharacters*. More on that in a minute.
- So, this line would also be interpreted as a command with one argument, `filename with spaces`

METACHARACTERS

- A metacharacter is any character that has more than one meaning or interpretation.

- For example, you just learned about two of them: the single and double quotes. In normal context, they denote endpoints for arguments, not *actual* quote characters

- But what if you need a quote in your argument value, say a filename with a single quote like: `smith's`

ESCAPING

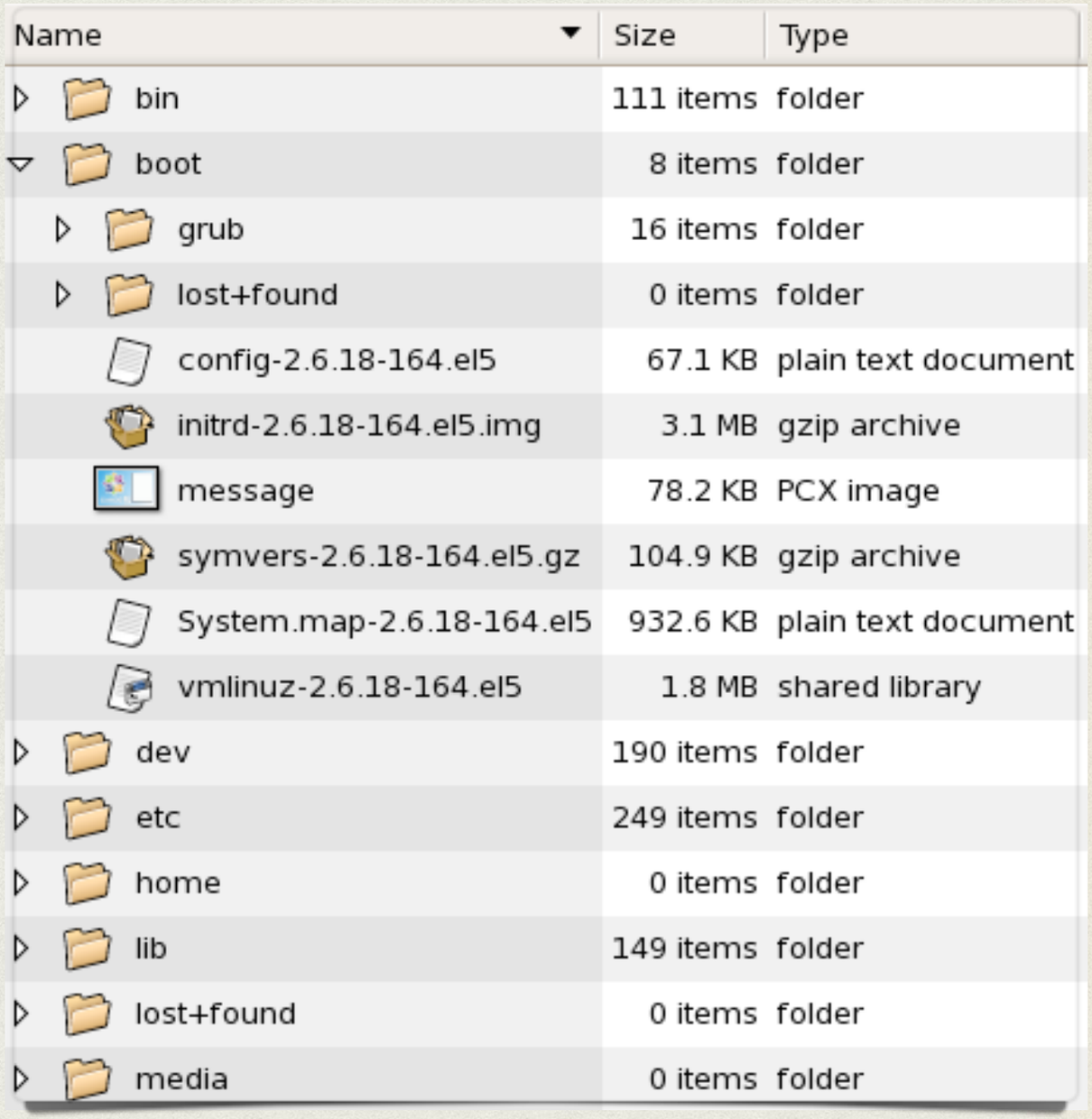
- The quick and simple way to do that is with the escape metacharacter, the backslash: \
  - `command smith\'s`
- The escape character tells the shell to interpret the character following the backslash as a normal character, rather than a metacharacter
- This allows you to use metacharacters as regular characters

BASIC COMMANDS

- `who`: Lists currently logged in users
- `uptime`: Statistics about machine usage and run time
- `echo`: Prints the given arguments to the screen
- `date`: Print current date and time
- `exit`: Terminate current shell session
- `reset`: Reset terminal state to default settings

HIERARCHIES

- Data is stored in files
- Files are grouped and organized in Directories, creating a tree structure
- The filesystem begins at root, represented as: /
- The Standard Hierarchy provides basic organization



WORKING DIRECTORY

- Operations within the shell generally gather input from files and output information to files, so the shell tracks a "working directory" to ease the file specifications, and have a default location to output files if one is not provided

- `pwd`: Print Working Directory

- `cd`: Change [working] Directory

PATHNAMES

- A pathname specifies the exact location of a file or directory within the filesystem.
- Understanding pathnames is critical to a happy shell life
- There are two types of pathnames: absolute and relative

ABSOLUTE PATHNAME

- An absolute pathname uses the root of the filesystem to fix the starting location for the path search.
  - `/etc/passwd`
- Starting from `/`, descend into the `etc` folder, then locate the file named `passwd`
- The key is the leading slash - exactly fixing the starting point

RELATIVE PATHNAME

- Relative pathnames only specify a file's location with respect to a working directory. The path is *relative* to the current working directory. Relative pathnames never start with a /.
  - `memos/january.txt`
- From within the current directory (see? the starting point is the current directory - not always / like for absolute), descend into the `memos` folder and locate the file `january.txt`

COMPARISON

Absolute Pathnames

- *Always start with a /*
- Search starts from /
- Always refers to exactly one file

Relative Pathnames

- *Never start with a /*
- Search starts from CWD
- Can refer to any number of files (dependent on CWD)

BASIC COMMANDS

- `mkdir`: Create a new directory
- `touch`: Update modification and access times of given file
- `spell`: Spell check given file (or input on stdin)
- `mv`: Move a file from one location to another (rename)
- `cp`: Copy a file to another location
- `rm`: Remove (delete) a file
- `ls`: Display listing (contents) of a directory

WILDCARDS

- Wildcards are another set of metacharacters which provide a shorthand notation for specifying large groups of files

- There are 3 basic pathname wildcards:
  - *
  - ? See manpage for details
  - [set] See manpage for details

WILDCARD: *

- The * wildcard is the easiest to understand, and most common
- Definition: Match 0 or more characters. Any characters.
- Examples:
  - `*`
  - `a*`
  - `*.txt`

ENVIRONMENTS

- Every piece of running software (a process - more on that later) has it's own environment
- The environment is simply a collection of key->value pairs
- The key is [traditionally capitalized] letters, numbers and symbols to uniquely identify the variable
- The value is a string

ENVIRONMENTS

- Examples:
  - `PATH=/usr/local/bin:/usr/bin:/bin:/sbin`
  - `HOME=/home/bob`
  - `TOTAL=348`

ENVIRONMENTS

- To create a new variable (or change an existing one):
  - `TOTAL=100`
- You type the name of the variable, an equals sign, and the value. Don’t forget about quoting if needed!

ENVIRONMENTS

- Once a variable is created, you can view it's value with the $ metacharacter. The easiest way is to use `echo`:
  - `echo $TOTAL`
- The $ metacharacter asks the shell to look up the value for the named variable, and replace everything with that value.
- So after parsing, the above command becomes:
  - `echo 100`

ENVIRONMENTS

- Environment variables are local to the containing process, but you can mark variables as “exported”, which allows them to be passed down to subprocesses (child processes)

- Once a variable is created, to mark it exported:
  - `export TOTAL`
- Note the **lack** of the $ metacharacter!
- To stop exporting: `export -n TOTAL`

ENVIRONMENTS

- `set`: Displays all environment variables and values
- `env`: Displays exported environment variables and values
- To remove a variable completely:
  - `unset TOTAL`
- A note about the $ metacharacter: if the variable does not exist, the entire statement evaluates to the empty string

MAN PAGES

- Man pages, short for Manual Pages, represent the online help system in the Linux environment
- Simple interface:
  - `man <command>`
  - `man <library>`
  - `man <function>`
  - `man <file>`

MAN COMMAND

- The `man` command locates the requested manpage and formats it for display

- Manpages can be written to cover any topic, but generally are available for commands, libraries, function calls, kernel modules and configuration files.

- For example, to learn more about the `who` command:

  - `man who`

MANPAGES

- Follow fairly standard format: Name, synopsis, description, examples, see also. Additional parts include author, copyright, bugs and more.
- Manpages are organized into “sections”, grouping user commands into one section, system libraries in another, and so forth.
- The See Also section is invaluable!

INFOPAGES

- There is some movement to convert the aging manpage system into a newer format, the infopage system.
- The info system provides a more advanced interface, supporting links, split windows and more. Accessing infopages is the same:
  - `info <topic>`
- Once within the info system, type ? for help on the interface
- The conversion is still in it's infancy

EXERCISES

- In your home directory, create a directory called ‘test’.
- Read the man page on man.
- List all files in your home directory that start with an ‘a’ (note there might not be any – create a few with the touch command)
- Display your PATH environment variable and explain it’s purpose.
- Open a file browser and use it to explore the filesystem. At the same time, explore the same locations from the command line.
  - Use this time to get comfortable with pathnames
  - They are incredibly important!

INPUT AND OUTPUT

- This is the “normal” flow of data

REDIRECTION

- Changing the standard flow of input and output
- Output redirection sends one or more of the output streams to files on disk
- Input redirection feeds a file from disk as the input to a process

OUTPUT REDIRECTION

```
who > who.out
```

- Simple output redirection. Creates/overwrites file.

OUTPUT REDIRECTION

```
who 2> who.err
```

- Simple stderr output redirection. Creates/overwrites file.

OUTPUT REDIRECTION

```
who > who.out 2> who.err
```

- Combined out & err redirection. Creates/overwrites files.
- File names must be different!

OUTPUT REDIRECTION

```
who > who.all 2>&1
```

- Combined out & err redirection. Creates/overwrites files.
- Only one file name, used for both output streams

OUTPUT REDIRECTION

- All of the previous examples would create the output file if it did not exist, and if it did, would completely overwrite the existing file with the output of the command.

- Adding an extra > would turn the redirection functions into appending mode:

  - who >> who.out

  - who 2>> who.err

  - who >> who.all 2>&1

OUTPUT REDIRECTION SUMMARY

`> file`	• capture stdout to `file` • overwrites • > is equivalent to `1>`
`2> file`	• capture stderr to `file` • overwrites
`> file 2> file2`	• capture stdout to `file` • capture stderr to `file2` • overwrites

OUTPUT REDIRECTION SUMMARY

`>> file`	• capture stdout to `file` • appends • >> is equivalent to `1>>`
`2>> file`	• capture stderr to `file` • appends
`>> file 2>> file2`	• capture stdout to `file` • capture stderr to `file2` • appends

OUTPUT REDIRECTION SUMMARY

`> file 2>&1`	- capture stdout to `file` - capture stderr to `file` - overwrites
`>> file 2>&1`	- capture stdout to `file` - capture stderr to `file` - appends

INPUT REDIRECTION

```
cat < who.all
```

- Simple input redirection

REDIRECTION

- Input redirection isn't common anymore, now that most commands can handle their own file I/O
- Input and output redirection can be combined:
  - `cat < who.all > cat.who.all`
  - `cat < who.all 2> cat.who.all.err`
  - `cat < who.all > cat.who.all.all 2>&1`

EXERCISES

- From your home directory, use echo and output redirection to create a file in the ‘test’ folder called ‘file1’ with the contents ‘helllo’. Use a **relative** pathname.

- Use input redirection and the spell command to spell check ‘file1’.

- Spell check ‘file1’ again, saving the output to a file using redirection.

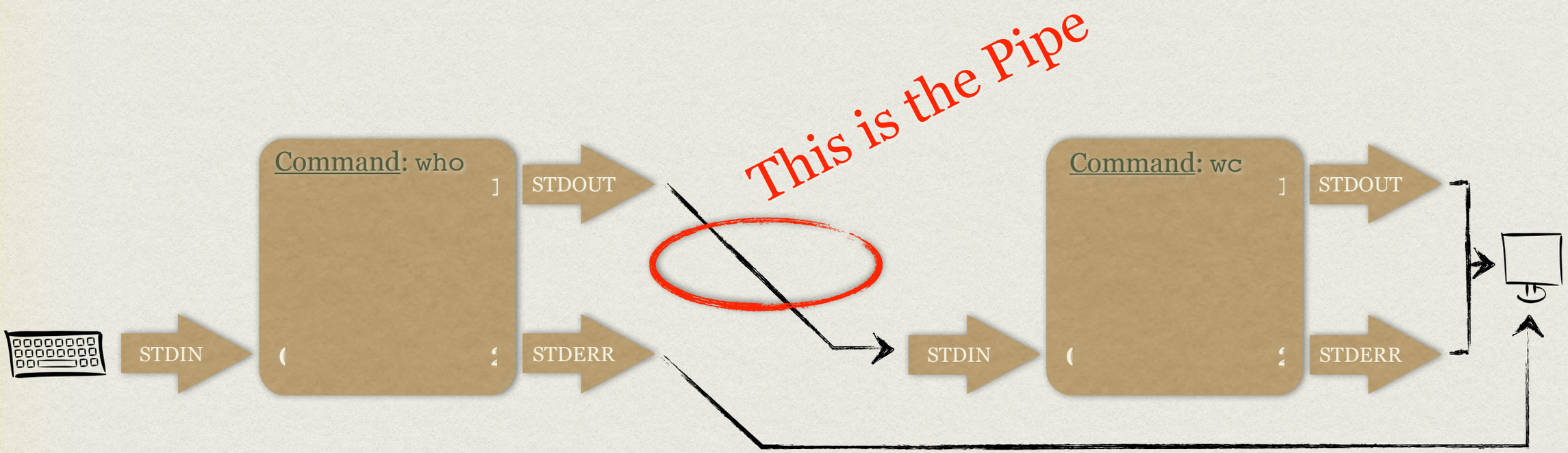
- What is the absolute pathname for ‘file1’?

PIPES

- Sweet, beautiful, powerful pipes! My favorite shell feature!
- In concept, pipes are very, very simple
- A pipe operates on two commands, connecting stdout of the command on the left to stdin of the command on the right
  - `who | wc -l`
- Let's look at a picture of this…

PIPES

```
who | wc -l
```



- The output of `who` is piped into the input of `wc -l`
- This produces a count of the current user sessions

PIPES

- Pipes can be chained as long as needed, and can also be combined with redirection:
  - `who | fgrep bob | wc -l > bob.sessions`
- It's even possible to intermix pipes and redirection! Just keep your streams straight in your head:
  - `who 2> who.errors | fgrep bob 2>&1 | wc -l`
- Try to diagram the previous command!

TEE

- A very useful tool when working with pipes is `tee`
- `tee` takes one argument, a filename, and will feed all input from stdin to the file, *while simultaneously feeding the output to stdout*
- In effect, `tee` forks its input stream, sending one copy to a file on disk, and another copy to stdout
- Very useful tool!

EXERCISES

- Spell check ‘file1’ and, using tee, output the results to the screen and a file on disk.

- Read the man page on wc. Use this information to count the number of misspelled words in /etc/nsswitch.conf

- Use echo and redirection to append a few more lines to ‘file1’ with information about yourself.

FILESYSTEMS

Mmmm crunchy

PURPOSE

- So all this data…
- How to organize? Whose job?
- Filesystems!

PERMISSIONS

- Linux supports 3 main types of access on a file:
  - read: View the contents
  - write: Modify the contents and metadata
  - execute: “Run” the contents
- Actually, it’s slightly more complex because it’s different for files and directories…

PERMISSIONS

	Files	Directories
Read	View the contents	List contents
Write	Change the contents/ metadata	Create/delete entries, change metadata
Execute	“Run” the contents	Operate with directory as CWD

AWESOME… SO?

- Combining these permissions allows for the most common access levels:
  - Read only
  - Read/Write
  - Execute
  - etc
- Now to add a little more granularity, users and groups…

OWNERSHIP

- All files are associated with one user and one group. This creates the foundation for the main meat of the security infrastructure in the Linux (and Unix) operating system.

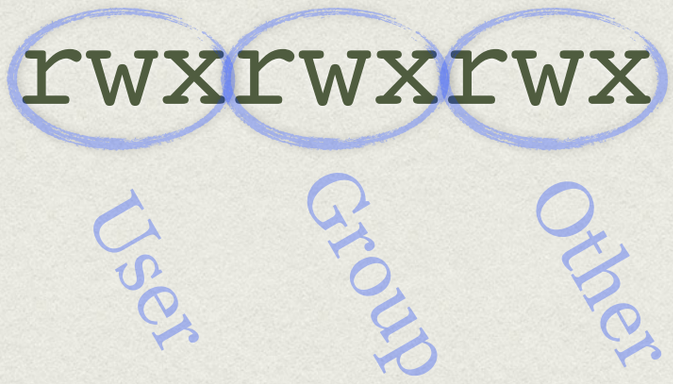
- When a process attempts an operation on a file, the user and group of the process (because every process is associated with one user and one group! surprise!) are compared with the user and group of the file, which determines what level of permissions is granted or denied on the file...

PUTTING IT ALL TOGETHER…

- Every file has 3 levels of permissions:
  - User
  - Group
  - Other
- When a process seeks access, the process user is compared to the file user - if they match, the process gets the User permissions. Next Group. If no match, Other level access

THE TRIPLE OF TRIPLES

- All of the permission information is neatly summarized with 9 characters:
  - `rwxrwxrwx`



- The presence of the letter indicates the permission is granted, a hyphen in it's place indicates the permission is denied. Read only: `r--r--r--`

CHANGING OWNERSHIP

- Two commands are available for changing the ownership of a file:
  - `chown`: Change Owner - changes the user owner of a file
    - `chown bob memo.txt`
  - `chgrp`: Change Group - changes group owner of file
    - `chgrp mgmt memo.txt`

CHOWN IT UP

- `chown` can actually change the group owner as well, so you don't need to bother messing with `chgrp`
  - `chown :mgmt memo.txt`
- You can do both at once, in fact!
  - `chown bob:mgmt memo.txt`

CHANGING PERMISSIONS

- Changing permissions is slightly more involved. The command is `chmod` (change mode)
- There are two basic ways to represent the permissions:
  - human friendly
  - octal

HUMAN FRIENDLY CHMOD

- When using human friendly permission specification, you just need to specify what *level* permission you want to change, *how* you want to change it, and *what* the permissions are..

- A table will clear up the mud...

HUMAN FRIENDLY CHMOD

	Who?	How?	What?
Symbols	u, g, o	+, -, =	r, w, x, s, t
Explanation	user, group, other	add, subtract, set	read, write, execute, set id, sticky

SO…

- Examples:
  - `chmod u+x file`
  - `chmod go-r file`
  - `chmod u=rw,go= file`
- Yes, you can combine “equations” to make different changes by separating them with commas, as in the last example

OCTAL?

- Octal refer to a *base* for a *numbering system*. Namely, base 8. Humans think and count in base 10, decimal. Computers work in base 2 (binary) and sometimes base 16 (hexadecimal). Octal is just another one, useful for permissions

- Short of a long, grueling discussion of numbering systems, you're going to have to just do some memorization here...

OCTAL!

Octal	Binary	Permissions
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

OCTAL

- Each octal digit fully represents all three primary permissions, so to specify all the basic permission levels for a file, all you need are 3 octal digits (user, group, other)!

- `chmod 777 file`
- `chmod 755 file`
- `chmod 644 file`
- `chmod 000 file`

EXERCISES

- Add write permissions for everyone to ‘file1’. Change the owner to ‘user’ and the group to ‘user’. (It won’t change, but if you did it right you won’t get an error message)

- Explain the following permissions: rw-r-----

- Explain the permissions represented by 644

LINKS

- Linux filesystems support two types of links, hard and soft
- Soft links are the easiest to understand, and have cousins in most operating systems, which makes them familiar
- Hard links are best explored later in your Linux career

SOFT LINKS

- A soft (or symbolic) link is like a shortcut in windows: it's a file that simply "points" to another file.
- In Linux, the pathname "pointed to" (source) is stored in the data blocks of the soft link (target)
- A soft link is an actual file, consuming an inode and using data blocks to store whatever pathname it's pointing to

SOFT LINKS

- To create a soft link, use the `ln` command with the -s option:
  - `ln -s memo.txt link-to-memo.txt`
- In this example, `memo.txt` is the source and `link-to-memo.txt` is the target
- This command **creates a new file**, `link-to-memo.txt`, of type link, which points to `memo.txt`

SOFT LINK TRIVIA

- Since soft links merely store a pathname (absolute or relative), they can link to anything, anywhere. Local filesystem, other filesystems, network filesystems, removable media filesystems. They can even point to invalid pathnames! The kernel cares not!
- Removing a soft link does not remove the file pointed to, only the link file.
- Soft links do not have permissions themselves (no need!)

EDITING FILES

- Time for a Nerd Holy War
- Editor of choice, anyone? (TUI only - if anyone throws down with a GUI editor, you've failed the class already!)
- In my opinion, `vi` (or `vim`) wins =)
- `emacs` is great, powerful and fast, but it's just not *common* enough. Plus, the control-x madness is, well, madness! ;)
- For now, you can use `nano`, but learning `vi` will be critical if you intend to further your Linux pursuits

EXERCISES

- In your home directory, create a soft link to ‘file1’. Verify the link by cat-ing the contents out. Compare the inode numbers.

- Use nano to edit file1 with some of your observations about Linux so far

PROCESSES

At least they’re not ISO-9001 processes

STRUCTURE

- In Linux, a Process wraps up everything that is needed to know about a running piece of software

- The meta information not only includes the machine code for the software, but also things like what user/group pair is running the process, when it was started, what the command line was, etc.

- In fact, here's a short list of the pertinent parts of a process:

STRUCTURE

- PID
- PPID
- UID/GID
- Command
- Start Time
- CPU Time
- CWD
- State
- TTY
- Environment
- Priority
- Nice Level

PID

- Process ID
- Linux uses this number to uniquely identify every process on the computer
- Number from 1-32768 (default - can change the maximum)
- Assigns new PIDs incrementally by 1, 2 or 4
- Loops back to 1 after hitting the maximum

PPID

- Parent Process ID
- PID of the process that started this one

UID/GID

- The User and Group running the process
- Very important! Defines access and permissions to file system and operating system.
- Inherited from Parent process unless:
  - SetUID/SetGID bits on executable
- Completes the Circle of Security

COMMAND

- The command (and arguments) for the process
- Identifies the executable running, as well as the arguments passed at invocation

START & CPU TIME

- Start Time tracks when the process was started
- CPU Time tracks time the process actually spends **running on** the CPU

CWD

- Current Working Directory
- ‘nuf said
- Inherited from parent process

STATE

- State of the process:
  - Runnable
  - Stopped
  - Blocked - Interruptible
  - Blocked - Non-interruptible
  - Zombie

TTY

- Connected terminal
- Mostly informational
- Inherited from parent process

ENVIRONMENT

- Every process has it’s own Environment
- Inherited from parent process

PRIORITY

- The priority is a read-only value showing the current priority assigned by the scheduler
- Ranges from 0-99, with higher values representing higher priorities
- The scheduler constantly adjusts priorities to balance efficiency, performance and responsiveness

NICE LEVEL

- The nice level represents one influence on the calculations the kernel uses when assigning priorities
- Originally designed and named to allow users to be “nice” to other users of the system by assigning a higher nice value to an intensive process, which in turn lowers it’s priority
- Ranges from -20 to 19. Default nice level is 0
- Only root can assign negative nice values
- See `nice` and `renice` commands

LISTING PROCESSES

- `ps`: List of current processes
- `pstree`: Generate hierarchical view of processes
- Examples:
  - `ps` *View all processes started by logged in user*
  - `ps aux` *View details of all processes on system*
  - `pstree` *View tree of all processes on system*

PROCESS STATES

- There are 5 basic process states:
  - Runnable
  - Stopped
  - Blocked/Sleeping - interrutible
  - Blocked/Sleeping - non-interrutible
  - Zombie/Defunct

RUNNABLE

- This means the process is running, or is set to run
- Remember: Linux is a multi-tasking operating system, so it's hard to see exactly when processes are running (switched so quickly), so the state is **runnable**, indicating that the scheduler will provide CPU time when it's available

STOPPED

- Opposite of Runnable - the process will not get CPU time
- Nothing happens to the process - it’s still in memory, poised, ready to go. But when it’s put in the stopped state, the scheduler will not put it on the CPU
- Files/network connections remain open, but network connections may drop after a time (timeout)

INTERRUPTIBLE SLEEP

- The process is waiting for some event - perhaps an alarm from a sleep system call, perhaps a signal or other external event

- Interruptible means that other processes/events can break the sleep

NON-INTERRUPTIBLE SLEEP

- This sleep state is generally caused by IO operations - accessing a drive, communicating with the network, etc.
- Non-interruptible means that other processes/events can not break this sleep.
- This process is unable to respond to signals.

ZOMBIE/DEFUNCT

- Braaaaaaiiiiiiinnnnnnssss.. Wait, no, not that kind of zombie.
- An exited process whose parent did not `wait()` on the child
- Does not consume resources beyond a PID and meta information storage (< 1k generally)
- Generally caused by two situations:
  - Bug in software
  - Overly taxed machine

SIGNALS

- First form of Interprocess Communication (IPC)
- A signal is a message sent to a process to indicate events or other conditions. The signal itself is the message - there around three dozen defined signals...

COMMON SIGNALS

- **HUP** - *Hangup*
- **INT** - *Interrupt*
- **QUIT** - *Quit*
- **ILL** - *Illegal Instruction*
- **ABRT** - *Abort*
- **KILL** - *Kill*
- **SEGV** - *Segmentation Fault*
- **ALRM** - *Alarm*
- **TERM** - *Terminate*
- **STOP** - *Stop*
- **CONT** - *Continue*
- **FPE** - *Floating Point Exception*

SENDING SIGNALS

- `kill`: Send a signal to a process. Default signal: TERM
- Examples:
  - `kill 457`
  - `kill -9 2359`
  - `kill -CONT 1350`

USERS & GROUPS, BACKUPS

Basic System Administration

USERS AND GROUPS

- Users and Groups define access to the operating system through the file permission scheme.
- Root is the super user, and the only user with special permissions
- Every user is a member of at least one group, which is called their primary group. The main purpose of this primary relationship is to define group owner of created files.
- Users can have a secondary group membership in as many groups as needed. These secondary relationships exist to broaden a user's access to the files on the system.

CONFIG FILES

- User information is stored in two files:
  - `/etc/passwd`
  - `/etc/shadow`
- Group information is stored in one file:
  - `/etc/group`

/ETC/PASSWD

- List of user records, one per line, with columns separated by colons. Format:
- `login:x:userid:groupid:gecos:homedir:shell`
- Examples:
  - `root:x:0:0:root:/root:/bin/bash`
  - `mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash`

/ETC/SHADOW

- Similar colon-separated-column list of records:
- `login:password:`*password aging fields*
- Aging fields track dates for password resets, locks, etc
- Examples:
  - root:pB8msP1fCbCqc:13904:0:99999:7:::
  - nisburgh:vRoPw6a/jQsp.:14466:0:99999:7:::

/ETC/GROUP

- Same colon-separated-column list of records format
- `groupname:grouppassword:groupid:secondarymembers`
- Group passwords allow temporary access to a group, are rarely used and not set up by default
- Examples:
  - `daemon:x:2:root,bin,daemon`
  - `apache:x:48:jack,nisburgh`

MANAGEMENT

- While it is possible to edit the three files directly, it's easier and safer to use the management commands to create, modify and delete users and groups:

- `useradd, usermod, userdel`

- `groupadd, groupmod, groupdel`

USERADD

- `useradd`: Add a new user to the system

- Accepts various arguments to control the settings on the user account. Most common is the -g option to specify the primary group of the user, and the -G option to list secondary group memberships. Examples:

  - `useradd lisa`

  - `useradd -g clowns -G trouble,simpson bart`

USERMOD, USERDEL

- `usermod`: Modify a user's settings. Example:
  - `usermod -G detention bart`
- `userdel`: Remove a user from the system. Main option to consider is `-r`, which tells `userdel` to remove the user's home and spool directories. Example:
  - `userdel moe`

GROUP COMMANDS

- `groupadd`: Adds a new group to the system. Example:
  - `groupadd bullies`
- `groupmod`: Mainly used to rename a group. Example:
  - `groupmod -n mktg mkg`
- `groupdel`: Remove a group. Example:
  - `groupdel microsoft`

PASSWORDS

- `passwd`: Change login password.
- Root can change the password for any user on the system
- Root can also setup password aging, allowing for timed password resets and account disabling
- `passwd` is also the preferred way to lock a user account:
  - `passwd -l mary`

PASSWORD AGING

- To set the maximum lifetime for a user's password:
  - `passwd -x days login`
- When a user's password has expired, you can set the number of days it can remain expired before disabling the account completely:
  - `passwd -i days login`

SIDE NOTE: SU AND SUDO

- Best practice states that a user should always log in as a regular user, then switch to the root user when necessary for a system administration task. There are two tools available to do this:

- `su`: switch user. As a regular user, this allows you to switch to the root account if you know the root password.

- `sudo`: “su do”. Perform an action as root or another user. If configured for access, you only need your password. Use `visudo` to edit configuration.

EXERCISES

- Create a new group ‘dev’. Create a new user ‘alice’ as a member of the ‘dev’ group, with a description of “Alice from Dev” and a default shell of ‘/bin/csh’. Use the passwd command to set a password for alice, then log in as alice and verify her access.

- Lock alice’s account and verify she can’t log in anymore. Unlock her account and verify access once more. Add alice as a secondary member of the ‘gdm’ group.

- Set a maximum password lifetime of 4 weeks for the alice account. Look at the passwd, shadow and group files.

BACKUPS

- Why backup?
  - Hardware failures
  - Software failures
  - [Epic] User failures
  - Disasters

WHAT TO BACKUP?

- At minimum, all user data and intellectual property
- At maximum, entire systems, OS and all
- In reality, many factors drive what gets backed up:
  - budget
  - time
  - resources
  - need

WHERE TO BACKUP?

- Good question - many, many places
- Local online copies
- Remote online copies
- Offline copies - Disk, Tape

HOW TO BACKUP?

- Small scenario:
  - rsync, tar, burning software, tape drive
- Large scenario:
  - rsync, tar, enterprise backup software, tape libraries

FLATTENING HIERARCHIES

- How to backup a *directory*? The directory represents an entire *tree* of files and directories? How can you put all of the information necessary to recreate the tree into one file?

- tar!

- Originally the Tape Archive tool. Used to backup directory trees to tape. Nowadays more commonly used to “flatten” a tree into one file.

CREATING A TAR ARCHIVE

- To create a tar archive:
  - `tar cf <tarfile.tar> <file> [file]...`

- The `c` option tells `tar` to create an archive. The `f` option is critical - it tells tar to put the archive in a file on disk, rather than on a tape device.

- You can add the `v` option (`tar cvf`) to get verbose output. Tar will report every file added to the archive.

VIEWING AN ARCHIVE

- To view an archive (a table of contents):
  - `tar tf tarfile.tar`
- The `t` option asks `tar` to print a table of contents of the archive. If you add the verbose flag (`tvf`), tar will report detailed information on each file, similar to the long output of the `ls` command.

EXTRACTING AN ARCHIVE

- This is the tricky part of `tar`, and getting it right requires an understanding of how `tar` stores file in the archive.

- When an archive is created, the pathnames are stored into the archive. When you view the table of contents, you're viewing the relative pathnames as they would be created on extraction.

- This can sometimes confuse the user, and is best illustrated with an example…

EXTRACTING AN ARCHIVE

- If `tar tf file.tar` reports:
  - `memo.txt`
  - `report/`
  - `report/data`
- Then when the archive is extracted, the resulting files will be:
  - `CWD/memo.txt`
  - `CWD/report/`
  - `CWD/report/data`
- Where `CWD` represents the current working directory

EXTRACTING AN ARCHIVE

- To extract an archive:
  - `tar xf tarfile.tar`
- `tarfile.tar` will be extracted to the current working directory, so be careful! Make sure you understand the contents of the tar file to be sure you don't accidentally overwrite existing files.

TAR EXAMPLES

Help remove the mud

EXERCISES

- From your home directory, create a tar backup of the test folder. Name the tar file ‘test.tar’. Verify it is correct by viewing the table of contents.

- Create a new directory in your home folder called ‘temp’. Change into this directory and extract your test.tar backup file. Can you see the ‘test’ folder and it’s contents?

- Browse through the man page for ‘diff’. Use ‘diff -r’ to compare the original ‘test’ folder with the newly extracted ‘test’ folder. Are there any differences?

COMPRESSION

- Tar files can get quite large, and storing/sharing them uncompressed wastes a large amount of storage space and bandwidth.

- Enter: compression.

- Compression uses complex algorithms to rewrite the contents of a file in a way that takes up less space, but can be reversed back to the original contents

COMPRESS

- One of the original compression algorithms: the Adaptive Lempel-Ziv. Not used very much any more, especially in Linux environment
- Achieves 40-50% compression on average
- Extension: `.z`
- Compress: `compress`
- Decompress: `decompress`

GZIP

- Updated algorithm: Limpel-Ziv 77 (LZ77)
- Achieves 60-70% compression on average
- Extension: `.gz`
- Compress: `gzip`
- Decompress: `gunzip`

BZIP2

- Powerful algorithm: Burrows-Wheeler Block Sorting Huffman Coding
- Achieves 50-75% compression on average
- Extension: `.bz2`
- Compress: `bzip2`
- Decompress: `bunzip2`

TAR + COMPRESSION

- Once a tarball has been created, it's generally compressed with `gzip` or `bzip2`:
  - `gzip -9 tarfile.tar`
  - `bzip2 -9 tarfile.tar`
- The `-9` option tells the compression tool to maximize compression efficiency (taking longer). 1-9 are acceptable values, with `-1` indicating minimal efficiency and maximum speed.

ZIP FILES

- Zip files, originally put forward in the DOS/Windows world via the pkzip tools, and now winzip, are actually a combination of hierarchy archiving and compression.

- Basically, zip files include the features of tar and compression in one format! Advantages and disadvantages, of course.

- There are open source tools which allow access to creating, viewing and extracting zip files in the Linux environment.

ZIP

- Lots of algorithms implemented
- Varying compression ratio depending on algorithms used
- Extension: `.zip`
- Compress: `zip`
- Decompress: `unzip`

ZIP

- Remember, zip files are not just compressed files. The zip archive actually contains files and directories within it, so the interface is closer to `tar` than `gzip` or `bzip2`.

- Generally, zip files are only encountered in the Linux world when interacting with the Windows world. Within Linux, everything is a compressed tarball.

EXERCISES

- Make several copies of test.tar and use gzip to compress them. Try once with compression level 9 and once with compression level 2. Check the sizes of each.

- Use bzip2 to compress one of the copies. Compare it's size with the gzip sizes.

TROUBLESHOOTING

Or, what to do when the $h1t hits the fan

OVERVIEW

- Troubleshooting is a thorough methodology used to track down the cause of problem.
- Keywords: **thorough** and **methodology**
- Without a thorough and exhaustive approach, the issue might be overlooked
- Without a strong and methodical approach, the issue may be misdiagnosed

TROUBLESHOOTING KEYS

- Most Important: Only change one thing at a time
- Check #1 most likely cause: You
- Check logs for error messages
- After that, check configuration and permissions
- If all else fails, slowly, piece by piece, start removing complexity from the system to narrow down the problem area.
- DOCUMENT EVERYTHING

LOGS

- One of the easiest places to find the cause of a problem is in the log files.

- Log files store informational messages from software. The types of messages include debug information, status information, warnings, errors and more.

- Some applications manage their own log files. Others use the system-wide logging package...

SYSLOG

- syslog - The system logger. A framework consisting of a library, a daemon, a configuration file and logs.
- Any application can use the library and log messages through syslog with simple function calls.
- Log messages consist of 3 parts:
  - Facility
  - Level
  - Message

SYSLOG

- The facility describes what part of the operating system generated the message, and is selected by the software:
  - auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, security, syslog, user, uucp, local0-local7
- The level represents the importance of the message, and is also chosen by the software:
  - emergency, alert, critical, error, warning, notice, info, debug

/ETC/SYSLOG.CONF

- `/etc/syslog.conf` defines where all of the log messages should go. Destinations include files, screens of logged in users, console, other syslog servers.
- Basic file format:
  - `facility.level  destination`
- Examples:
  - `*.err  /dev/console`
  - `mail.*  /var/log/maillog`
  - `*.info;mail.none;authpriv.none  /var/log/messages`

/VAR/LOG

- `maillog`: messages from the email subsystem
- `secure`: authentication and security messages
- `cron`: cron messages
- `boot.log`: boot messages
- `messages`: catch-all

EXERCISES

- Take a few minutes to browse through the various logs in /var/log. Familiarize yourself with the kinds of information available.

- Browse the man page for syslog.conf

WHEN LOGS FAIL...

- Looking through logs is all fine and dandy, but really that's a best case scenario. Your software and hardware rarely come out and announce problems and solutions in the log files. No, it's not that easy!

- More often, users will encounter symptoms of a problem, and you, as the BOFH (hopefully not yet!), will be tasked with finding and fixing the issue.

TROUBLESHOOTING TOOLS

- Troubleshooting can be a mystical art, and fully exploring it's details is best left to a class in it's own right.
- For now, a discussion of several tools to help the process of troubleshooting will have to suffice.
- This list does not include network troubleshooting tools. Those tools will be covered in the networking lectures.

UPTIME

- `uptime`: Reports system uptime along with load averages.
  - Load Average: Average number of processes in run queue that are blocked.
- `uptime` reports three values: the load averaged over the last 1 minute, 5 minutes and 15 minutes. This is useful to get an idea of the load trend on the system.
- Example:

```
[root@dev1 ~]# uptime
 16:09:55 up 682 days, 10:11,  1 user,  load average: 0.00, 0.01, 0.00
[root@dev1 ~]#
```

FREE

- `free`: reports on memory and swap usage
  - buffers: I/O buffers, directory cache
  - cached: filesystem cache (data)
- Example:

```
[root@dev1 ~]# free
             total       used       free     shared    buffers     cached
Mem:        262316     214228      48088          0       1168      41728
-/+ buffers/cache:     171332      90984
Swap:       524280      74564     449716
[root@dev1 ~]#
```

W

- w: Displays an uptime report, followed by a breakdown of all logged-in users and what process they are running
  - JCPU: Combined CPU time of all processes attached to the terminal (foreground and background)
- PCPU: CPU time of foreground process, listed in “what” column
- Example:

```
[root@dev1 ~]# w
 16:26:42 up 682 days, 10:28,  2 users,  load average: 0.02, 0.05, 0.02
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    216-110-93-126.s 16:00    3:57   0.01s  0.01s -bash
root     pts/9    216-110-93-126.s 16:22    0.00s  0.01s  0.00s w
[root@dev1 ~]#
```

VMSTAT

- `vmstat`: Snapshot report covering several primary statistics.
  - procs: number of running and blocked processes
  - swap: swapped in and swapped out blocks of memory, per second
  - io: blocks in and blocks out read/written per second
  - system: interrupts and context switches per second
  - cpu: user, system, idle, wait and time-stolen from a VM

```
[root@dev1 ~]# vmstat
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0  74564   3608   4456  70156    0    0     0     2    0    0  0  0 100  0  0
[root@dev1 ~]#
```

TOP

- `top`: Self-updating tool displays combination summary at top, followed by ordered list of processes. Fully customizable.
  - The summary includes uptime information, memory breakdowns, CPU utilization and process state summaries
  - The process display can be customized and sorted to suit need

```
top - 16:39:32 up 682 days, 10:41,  2 users,  load average: 0.01, 0.00, 0.00
Tasks: 118 total,   1 running, 116 sleeping,   1 stopped,   0 zombie
Cpu(s):  0.1%us,  0.0%sy,  0.0%ni, 99.8%id,  0.0%wa,  0.0%hi,  0.0%si,  0.1%st
Mem:    262316k total,   258024k used,     4292k free,     7380k buffers
Swap:   524280k total,    74564k used,   449716k free,    67808k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      15   0 10316  648  592 S    0  0.2   0:06.24 init
    2 root      RT   0     0    0    0 S    0  0.0   0:04.88 migration/0
    3 root      34  19     0    0    0 S    0  0.0   0:00.19 ksoftirqd/0
```

DF

- `df`: lists filesystem utilization
  - Breaks down size and use information for each mounted filesystem
  - -h is useful option to display in “human-friendly” format

```
[root@dev1 ~]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             9.4G  7.2G  1.8G  81% /
none                  129M     0  129M   0% /dev/shm
[root@dev1 ~]#
```

ULIMIT

- ulimit: Sets resource limits
  - Can limit open files, memory use, cpu time, subprocesses and more.

```
[root@dev1 ~]# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
max nice                        (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 2112
max locked memory       (kbytes, -l) 32
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
max rt priority                 (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 2112
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
[root@dev1 ~]#
```

IOSTAT

- `iostat`: IO statistics report
  - Part of the `sysstat` package; not always installed
  - Allows for drilldown into the IO system to view real time metrics on IO operations per filesystem

```
[root@dev1 ~]# iostat -x
Linux 2.6.18-xen (dev1)   12/10/09

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.05    0.00    0.00    0.03    0.07   99.84

Device:         rrqm/s   wrqm/s   r/s   w/s   rsec/s   wsec/s avgrq-sz avgqu-sz   await  svctm  %util
sda1              0.00     1.68  0.01  0.55     0.14    17.83    32.12     0.03   54.01   2.89   0.16
sda2              0.00     0.00  0.00  0.00     0.01     0.01    35.26     0.00   80.51   4.95   0.00

[root@dev1 ~]#
```

EXERCISES

- Spend a few minutes playing with the various troubleshooting commands covered previously:

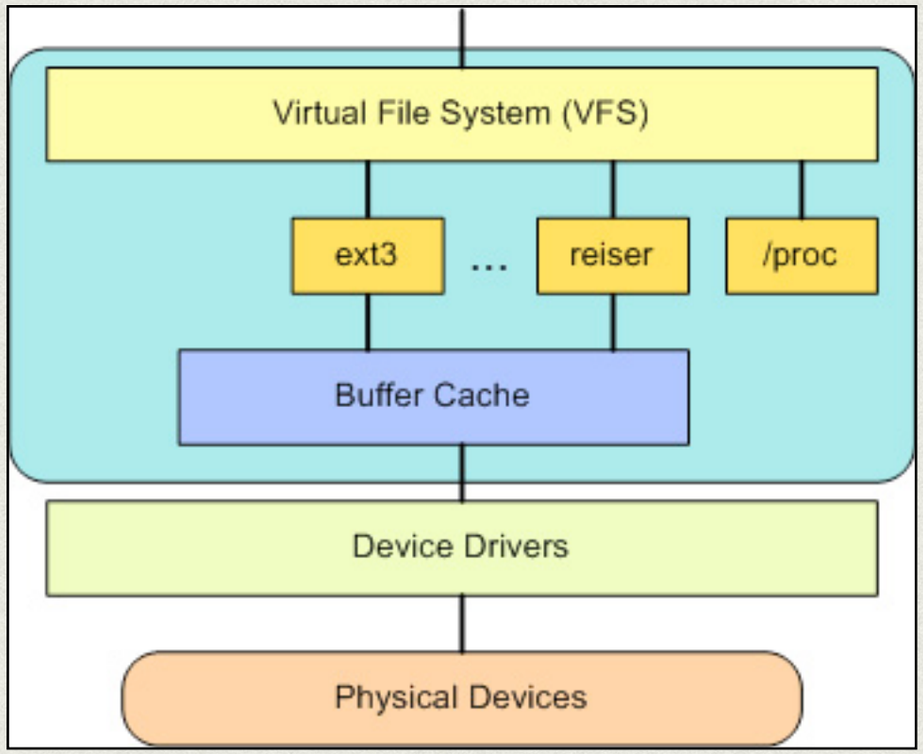
- top, df, free, iostat, vmstat, uptime, w, ulimit

FILESYSTEM ADMINISTRATION

mount? umount? mkfs? fsck?

KERNEL VFS LAYER

- VFS: Virtual File System
- One layer of the kernel is the VFS Abstraction layer. This layer defines a basic interface that all filesystem drivers at minimum must implement.



http://www.ibm.com/developerworks/linux/library/l-linux-kernel/

VFS

- From the user's perspective, the filesystem is simply a hierarchy of directories and files.

- But in reality, some branches might reside on a networked file server, some might be on an optical disc, some on internal drives..

- VFS allows the kernel to stitch all of these disparate storage systems into one cohesive interface!

/ AGAIN

- / is the root of the filesystem, forming the foundation upon which all access is provided.
- When additional filesystems need to be accessible, all that needs to be decided is the pathname to a directory where users can see the filesystem.
- This is known as the mount point.
- The mount point is how the kernel tracks thresholds between filesystems.

LET'S SEE THIS ON THE WHITEBOARD

MOUNT

- `mount`: Attach a filesystem to a given mount point
- Creates the “detour” sign
- Linux supports dozens of different filesystem types, available by the simple `-t` option to the mount command:
- `mount -t smbfs //windoze/share /windoze-share`

UMOUNT

- `umount`: detach mounted filesystem
  - Simply removes the “detour” sign
  - Mount point becomes a simple directory again
  - Generally only need to pass mount point as argument:
  - `umount /windoze-share`

PARTITIONING

- What is partitioning?
  - Splitting up a hard drive into organizable chunks
- Why?
  - Isolates filesystem corruption
  - Simplifies/speeds backups
  - Allows optimizing filesystems to tasks

PARTITIONING TOOLS

- `fdisk`: Works on one disk at a time, allows for viewing and manipulating partition table at a very low level
- `parted`: Excellent partitioning tool with features to restore corrupted partitions and more
- `Gnome Disk Utility`: Graphical partitioning tool with support for logical volumes, encrypted volumes and more

MKFS

- `mkfs`: format a device to create a new filesystem
  - “Paints the parking stripes” for the filesystem structure
- Creates superblock, block groups, superblock copies, bitmaps and inode tables and creates basic structure on disk
- Through `-t` option, mkfs can create different types of filesystems

FILESYSTEM INTEGRITY CHECKS

- `fsck`: Filesystem Check
  - Generally only run when a filesystem needs it:
    - Mount count
    - Last check
    - Dirty
- Checks all of the filesystem structures for accuracy and completeness

DEMONSTRATIONS

- Now for some demonstrations of the various filesystem administration tasks and tools

EXERCISES

- Un-mount the /lab filesystem.

- Rebuild the /lab filesystem (better figure out the right device name!) using ext3, a blocksize of 1k, and a reserve space of 2%. Confirm settings with tune2fs. Mount the /lab filesystem when complete.

- Un-mount the /lab filesystem and force an integrity check. Re-mount the /lab filesystem. Use e2label to set the filesystem label on /lab to '/lab'.

CROND, ATD

CROND

Scheduled fun

OVERVIEW

- `crond` is the cron daemon. Cron provides for the ability to execute commands on a regular basis.
- Generally used to run hourly, daily and weekly type system maintenance scripts.
- Also useful to run reports, cleanup jobs and much, much more.

USING CRON

- Cron is controlled through crontab files.
- There are system-wide crons, accessible under /etc/cron.*
- Every user has their own crontab, accessible through the crontab command

SYSTEM CRONS

- /etc/crontab defines the system cron jobs.
  - Many distributions use the run-parts script to execute all scripts found in /etc/cron.hourly, /etc/cron.daily, etc on the appropriate schedule.
- /etc/crontab defines the times for each schedule: hourly, daily, weekly, monthly
- Some systems (RHEL 6) use anacron as well

CRONTAB

- `crontab`: View, edit or remove crontabs
  - The `-l` option prints the crontab. The `-e` option opens the crontab for editing. The `-r` option removes the crontab.
- Root can work with the crontab for any user by specifying the username on the command line:
  - `crontab -e -u bob`

CRONTAB SYNTAX

- There are two main components to a crontab entry:
  - The timespec specifies when the command should be run
  - The command is what gets executed every time the timespec is matched

CRONTAB TIMESPECS

- The timespec is broken down into 5 fields, separated by spaces:
  - `minute hour day-of-month month day-of-week`
- Each field can contain a number, a range of numbers, a comma-separated list of numbers, an asterisk or a number slash division rate
- Mostly self-explanatory - some examples will help...

TIMESPEC EXAMPLES

- `0 23 * * *` *11pm every day*
- `30 * * * 1-5` *30 minutes after every hour, M-F*
- `0 7 1 * *` *7am, first of every month*
- `* * * * *` *Every single minute*
- `0,10,20,30,40,50 * * * *` *Every 10 minutes*
- `*/5 8-17 * * 1-5` *Every 5 minutes, 8am-5pm, M-F*

EXAMPLE CRONTAB

```
01 4 * * * /usr/local/bin/restart-webserver
00 8 1 * * /usr/bin/mail-report boss@mycompany.com
*/5 * * * * /monitor/bin/check-site -e admin@mycompany.com -o /var/log/check.log
```

- There are various additional options and features available to the cron system. Check the man pages for reference:
  - `cron`, `crontab` (sections 1 and 5)

ATD

ATD OVERVIEW

- `atd` is a simple daemon that executes one-off jobs at a certain time.

- To create an at job:

  - `at <time>`

  - Then you enter all of the commands you want run at the given time, and finish by typing ctrl-d

ATD

- atd is not commonly used these days, but if it's there is can be useful in some situations..
  - If editing the firewall on a machine over the network, it's sometimes nice to put a simple "reset" so if you lock yourself out, you'll be able to get back in the machine:

```
[root@localhost ~]# at now + 10 minutes
at> iptables-save > /iptables.backup
at> iptables -F
at> <EOT>
job 1 at 2009-11-30 10:44 a root
[root@localhost ~]#
```

ATD

- Some additional commands to use with the at system:
- `atq`: Displays list of at jobs
- `atrm`: Removes given at job from queue

SOFTWARE INSTALLATION

Gotta have it

DELIVERY!

- Software is delivered in one of two manners:
  - Source form - requires compiling
  - Binary form - generally wrapped up in a package

WHICH IS BEST?

- Both formats have their advantages and disadvantages..
  - Compiling from source can provide higher performing machine code, plus it gives the option of selecting features and configurations only available at compile time.
- Pre-compiled software is easier - it alleviates the [possible] headaches of compiling, and if distributed in a package format, provides built-in management functionality.

PACKAGES

- Installing a software package is pretty straight forward.
- There are a few different package formats out there. The two most popular are:
  - `rpm`: Redhat Package Manager
  - `deb`: Debian package
- In this course, we'll only be focusing on rpm's. Deb's have similar functionality and capability, so learning the command syntax is about all that is needed for proficiency.

RPM

- RPM's provide full software packaging features: pre-install scripts, post-install scripts, dependencies, meta information, and an installed software database to name a few.

- The RPM system maintains a database of all installed software on a machine - this is useful for tracking and updating reasons, as well as dependency verification and software management.

RPM

- `rpm`: The Redhat Package Manager tool. Provides interface to RPM system, performing queries, installs, upgrades, uninstalls and general database maintenance operations.
- `-i` option: install the given package
- `-q` option: query the database
- `-e` option: erase the given package from the system

YUM

- Not yum as in “This is yummy!”
- yum: Yellowdog Updater Modified
  - Supports package installation over the network through repositories.
  - RPM backend
  - Simple interface

EXERCISES

- Browse through the manpage for ‘rpm’. Study the “Query” section.

- Use your new knowledge to produce an alphabetized listing of the names for every installed package on your system.

- To what package does ‘/usr/bin/time’ belong?

- Browse the manpage for ‘yum’

NETWORK CONFIGURATION AND SERVICES

route add default gw 192.168.0.1

NETWORK CONFIGURATION

- There are two main approaches to configuring a machine for network access:
  - Static configuration
  - Dynamic configuration
- Static configuration uses set parameters for the configuration, which is known by the machine and the network and never changes. Generally used with servers.
- Dynamic configuration configures network machines on the fly, where a service on the network provides all configuration parameters to a machine when it joins the network. Generally used with workstations.

DYNAMIC CONFIGURATION

- Dynamic configuration is the easiest to use.
- The machine just needs to set up it's interfaces with the DHCP protocol.
- DHCP: Dynamic Host Configuration Protocol.
- A lease is obtained from the DHCP server, providing all network configuration details for the client. The lease expires after some amount of time and is renewed by the client to maintain network access.

STATIC CONFIGURATION

- Static configuration requires four configuration parameters in order to allow full network functionality:
  - IP Address
  - Netmask
  - Default Gateway or Router
  - DNS Server(s)

DNS?

- Domain Name Service: This is the glue between network names and IP addresses.

- Remember: Humans like names, computers like numbers. DNS is a service like so many others, mapping names to numbers and numbers to names. Mostly a convenience.

- Also provides for email functionality, geographic load balancing and limited service failover capabilities.

STATIC CONFIGURATION

- The first two components of static configuration are IP address and netmask.
- These provide LAN-level access
- `ifconfig`: Original network Interface configuration tool - being replaced by `ip`
- Basic idea:
  - `ifconfig eth0 192.168.0.100 netmask 255.255.255.0`

GATEWAYS

- The third configuration parameter is the default gateway.
- Provides access to *inter-networking*, or moving from just the local LAN to other LAN's
- `route`: Original kernel routing table tool - being replaced by `ip`
  - Displays and manipulates network routing table
  - `route add default gw 192.168.0.1`

DNS SERVERS

- Final piece of configuration information.
- List of one or more IP addresses which provide the DNS service, allowing name to IP address mapping
- Very simple to configure. Add `nameserver` lines to `/etc/resolv.conf`:
  - `nameserver 192.168.7.15`

STATIC CONFIGURATION

- Once all four pieces of information are configured on the system, full network service will be available.
- Best practice:
  - Configure IP address and netmask. Check LAN connectivity.
  - Configure default gateway. Check intra-LAN connectivity.
  - Configure DNS: Check name resolution.

ONE MORE THING...

- `ifconfig`, `route` and `ip` directly manipulate the running kernel, and are not permanent changes to the system. After a reboot, changes will be lost.
- To make IP address, netmask and gateway changes permanent, you have to edit two configuration files:
  - `/etc/sysconfig/network-scripts/ifcfg-eth0`
  - `/etc/sysconfig/network`

SO…

Demonstrations are Good

EXERCISES

- Check your current IP address, default route and DNS servers.

- Restart networking services using the proper init script:

```
service network restart
```

NETWORK TROUBLESHOOTING

ping!

RESPONSIBILITIES

- Networking systems together is often a difficult task, further complicated by large networks and special requirements.
- For this reason, networking is it's own area of expertise
- The network engineer is responsible for everything up to and including the cable and plug connecting to the server
- The systems engineer is responsible for everything within the server, up to and including the network card interfacing to the cable.

BASICS

- Basic network troubleshooting boils down to verifying three aspects of network performance:
  - LAN access
  - Inter-LAN access
  - DNS service
- Notice the parallels to the last lecture? Indeed!

LAN ACCESS

- LAN access means being able to at least talk to another machine *on your subnet*.
- Obtaining at least this level of access indicates that everything is working fine with the network card, the device drivers, the cable and initial point of access to the network
- This also verifies the IP address and subnet mask
- So how to test? First tool of network troubleshooting!

PING

- `ping`: “Packet Internet Groper”
  - Using IP/ICMP echo requests and echo replies, times the response time between two machines.
- `ping 192.168.0.1`
- Times reported are Round Trip Times (RTT) and represent the time between sending a request and receiving a response.

LAN ACCESS

- Using `ping`, one can verify LAN connectivity by simply pinging a machine on the LAN.
- But what should you ping?
- The gateway is a great start! Always on the subnet, and [should] always be online.

INTER-LAN ACCESS

- Checking inter-LAN access verifies the gateway in two ways:
  - It tests that the gateway itself is working correctly
  - It also tests that the gateway is correctly configured in the system.
- To test, simply ping an IP address in another subnet.
- But what to ping?
  - DNS servers - they’re often times not on the same subnet
  - Memorize another IP in your network, or a public one: 8.8.8.8

DNS

- Checking DNS verifies name to IP mapping
- Simple to test: ping a server by name
- Pick any server: yahoo.com, google.com, mycompany.com
- So long as it’s a name, the DNS system will be tested

MORE TOOLS

- Besides ping, there are other network troubleshooting tools available for more advanced diagnostics:

- `traceroute`: Traces the route a message takes to get from the source machine to the destination.

- `netstat`: Network statistics - details on open and recently closed network connections

- `iptraf`: network statistics tool

MORE TOOLS

- `nmap`: Network mapper - useful for seeing what services are showing on a particular machine
- `tcpdump`: A tool to dump raw network traffic for analysis
- `wireshark`: GUI interface to a tcpdump-like tool
- `ntop`: Top-like command for network connections
- `ngrep`: grep for network connections! :)

EXERCISES

- Use ping to check connectivity to rackspace.com.
- Traceroute a few sites and review the output.