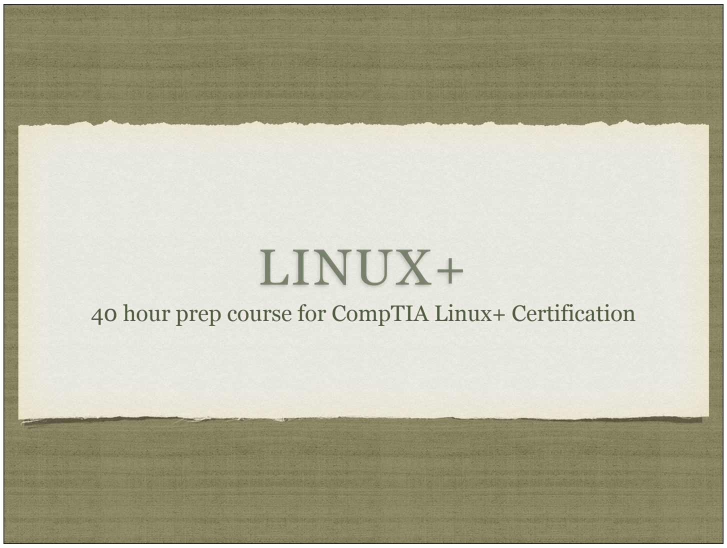
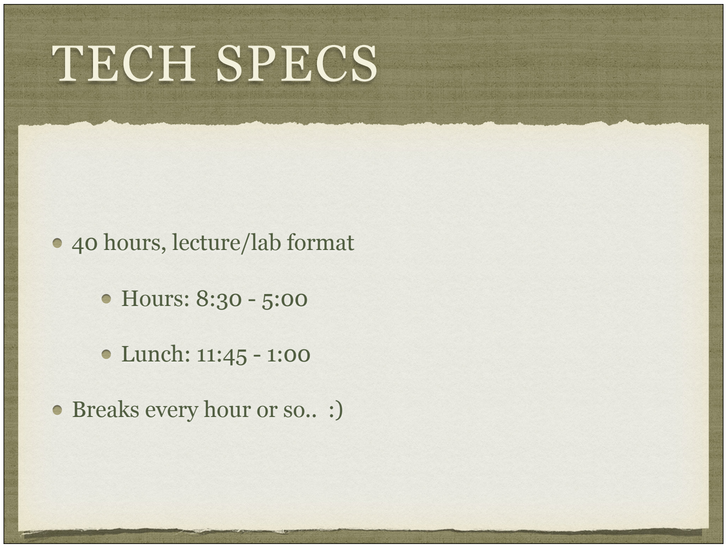


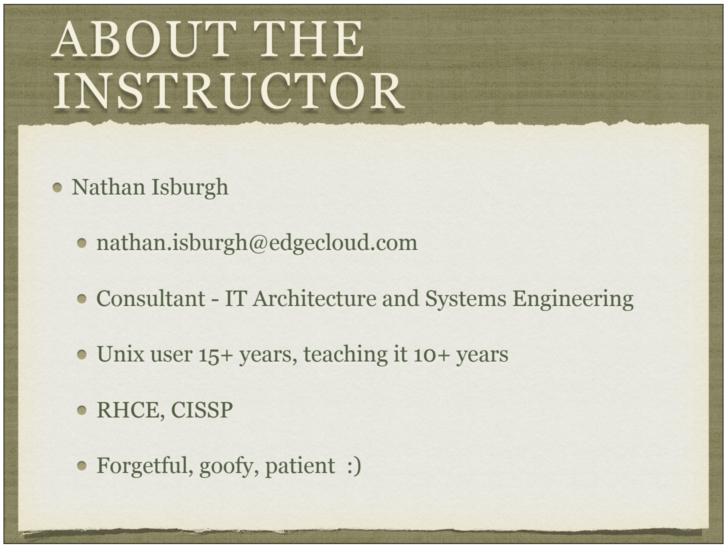
# LINUX+

40 hour prep course for CompTIA Linux+ Certification



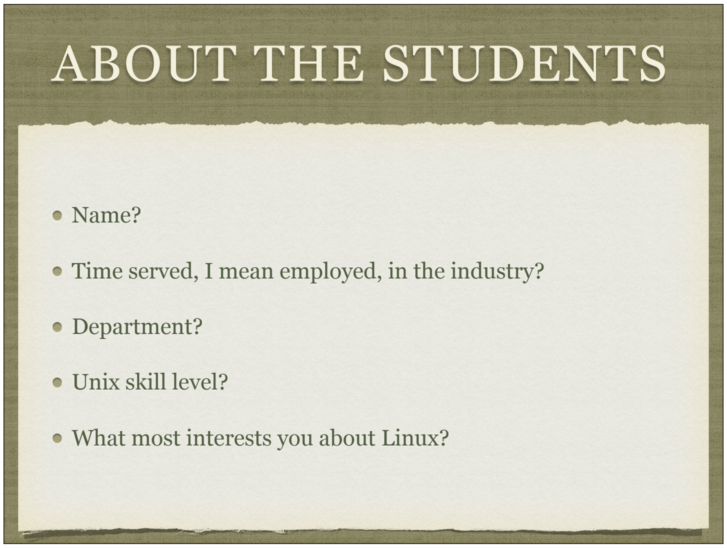
## TECH SPECS

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



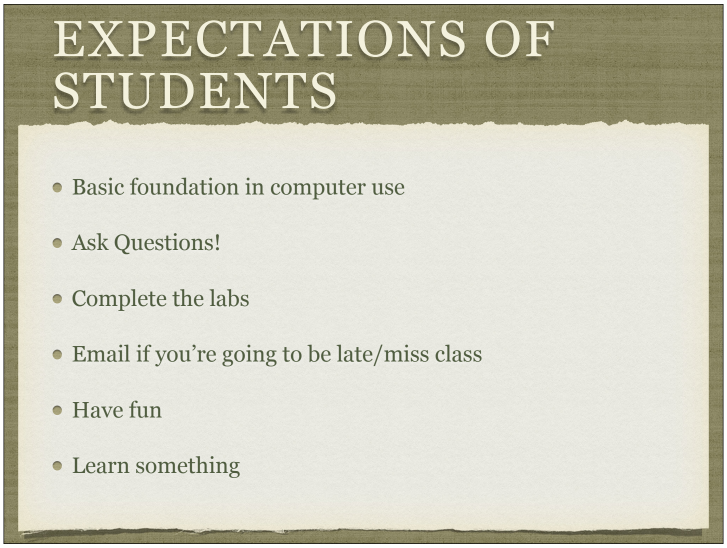
# ABOUT THE INSTRUCTOR

- Nathan Isburgh
  - nathan.isburgh@edgecloud.com
  - Consultant - IT Architecture and Systems Engineering
  - Unix user 15+ years, teaching it 10+ years
  - RHCE, CISSP
  - Forgetful, goofy, patient :)



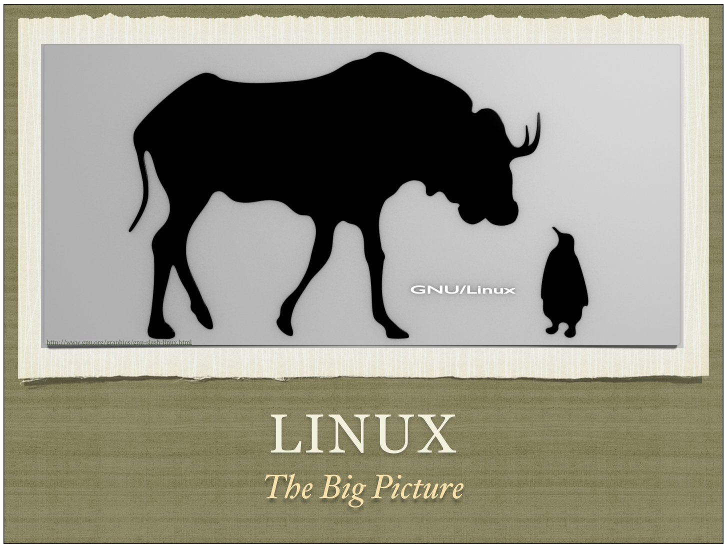
# ABOUT THE STUDENTS

- Name?
- Time served, I mean employed, in the industry?
- Department?
- Unix skill level?
- What most interests you about Linux?



# EXPECTATIONS OF STUDENTS

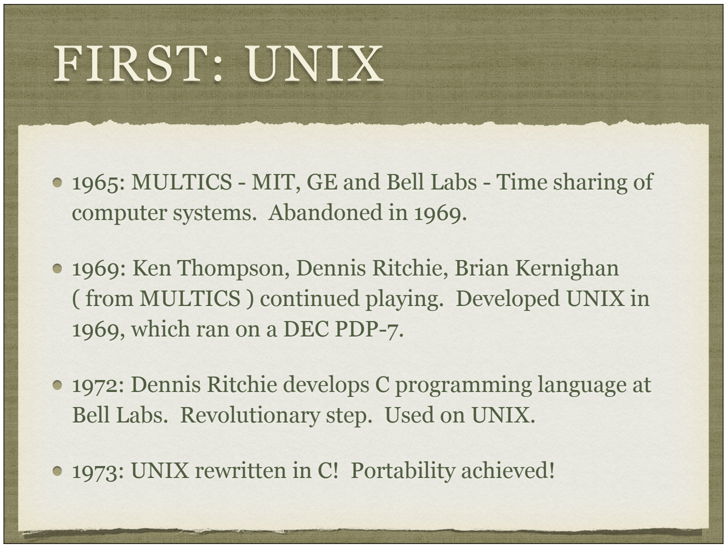
- Basic foundation in computer use
- Ask Questions!
- Complete the labs
- Email if you’re going to be late/miss class
- Have fun
- Learn something



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

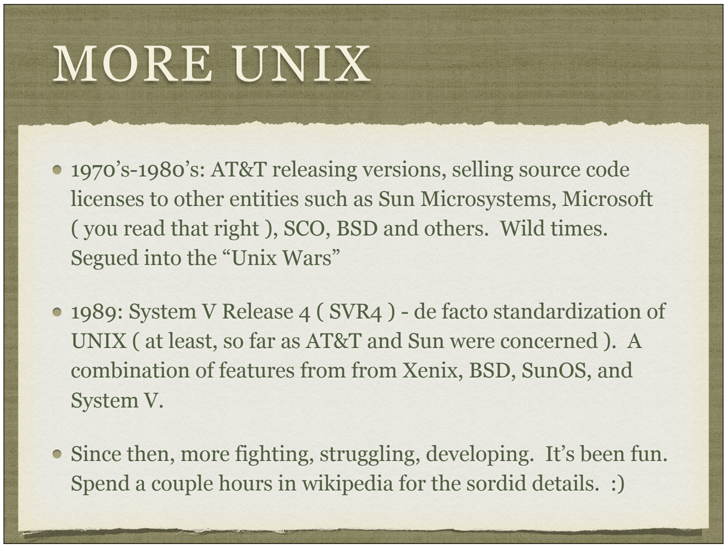
# LINUX

*The Big Picture*



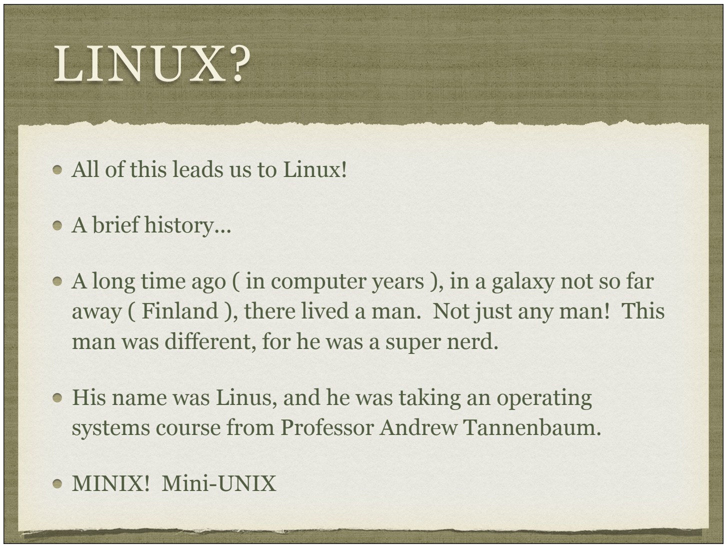
## FIRST: UNIX

- 1965: MULTICS - MIT, GE and Bell Labs - Time sharing of computer systems. Abandoned in 1969.
- 1969: Ken Thompson, Dennis Ritchie, Brian Kernighan ( from MULTICS ) continued playing. Developed UNIX in 1969, which ran on a DEC PDP-7.
- 1972: Dennis Ritchie develops C programming language at Bell Labs. Revolutionary step. Used on UNIX.
- 1973: UNIX rewritten in C! Portability achieved!



# MORE UNIX

- 1970's-1980's: AT&T releasing versions, selling source code licenses to other entities such as Sun Microsystems, Microsoft ( you read that right ), SCO, BSD and others. Wild times. Segued into the "Unix Wars"
- 1989: System V Release 4 ( SVR4 ) - de facto standardization of UNIX ( at least, so far as AT&T and Sun were concerned ). A combination of features from from Xenix, BSD, SunOS, and System V.
- Since then, more fighting, struggling, developing. It's been fun. Spend a couple hours in wikipedia for the sordid details. :)



# LINUX?

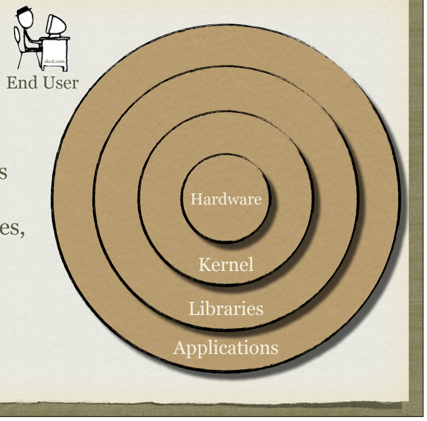
- All of this leads us to Linux!
- A brief history...
- A long time ago ( in computer years ), in a galaxy not so far away ( Finland ), there lived a man. Not just any man! This man was different, for he was a super nerd.
- His name was Linus, and he was taking an operating systems course from Professor Andrew Tannenbaum.
- MINIX! Mini-UNIX

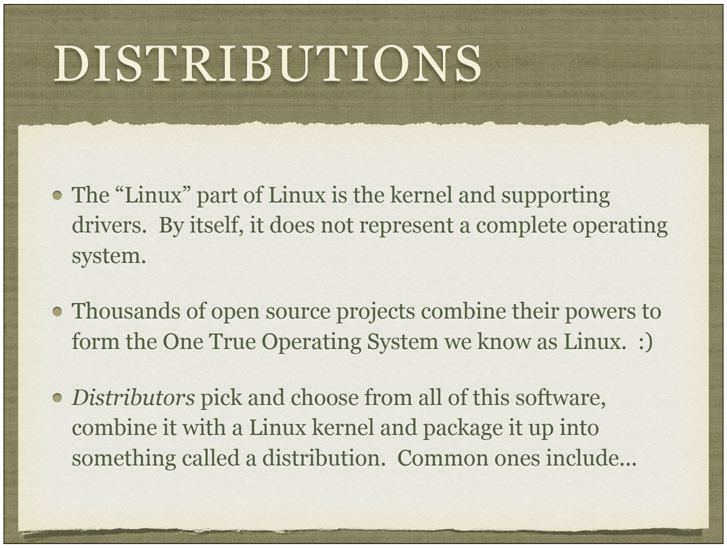
# LINUX

- UNIX not cheap or readily available. Certainly not on this cheap new hardware, the Intel x86 family.
- Linus learned a lot from MINIX, and using x86 hardware, eventually developed Linux. Released first version, *including the source code*, in 1991.
- What? Source code? And why'd he do that?
- Richard Stallman! The Open Source Movement!
  - 1983: GNU
  - 1985: Free Software Foundation.

# OVERVIEW

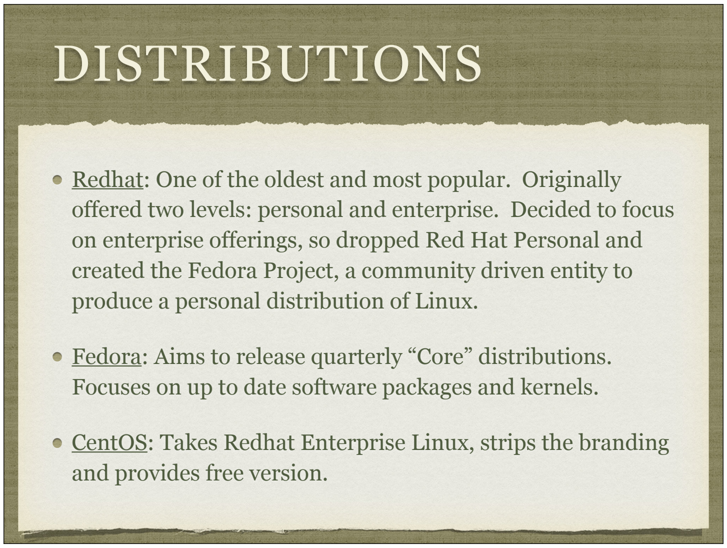
- Center of machine
- Scheduler, memory manager, device drivers
- Shared software routines, system calls
- User level software





# 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

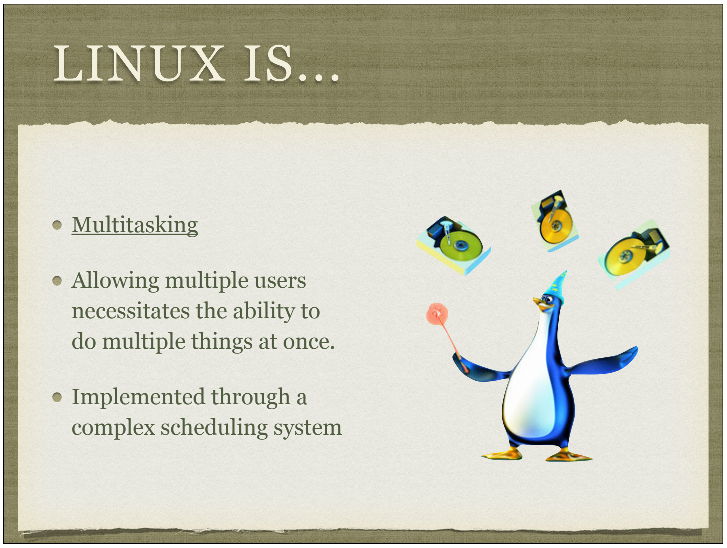
- Redhat: One of the oldest and most popular.  Originally offered two levels: personal and enterprise.  Decided to focus on enterprise offerings, so dropped Red Hat Personal and created the Fedora Project, a community driven entity to produce a personal distribution of Linux.
- Fedora: Aims to release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

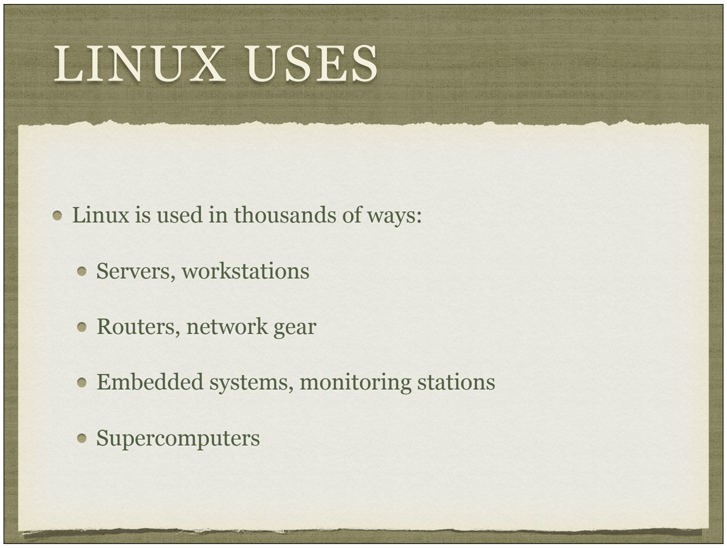
# LINUX IS...

- 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



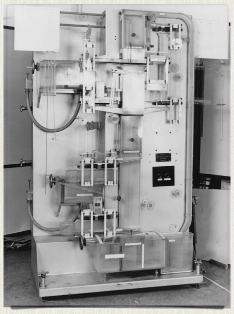
# LINUX IS…

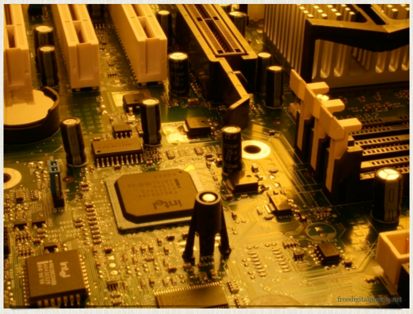
- Multitasking
- Allowing multiple users necessitates the ability to do multiple things at once.
- Implemented through a complex scheduling system



# LINUX USES

- Linux is used in thousands of ways:
  - Servers, workstations
  - Routers, network gear
  - Embedded systems, monitoring stations
  - Supercomputers



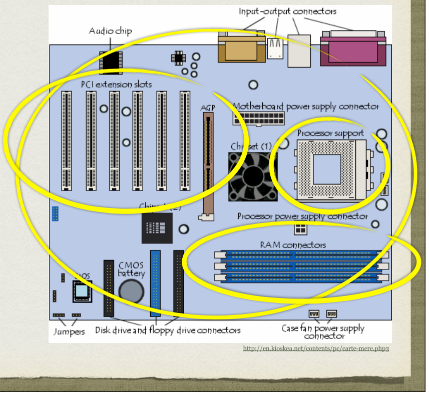


freedigitalphotos.net

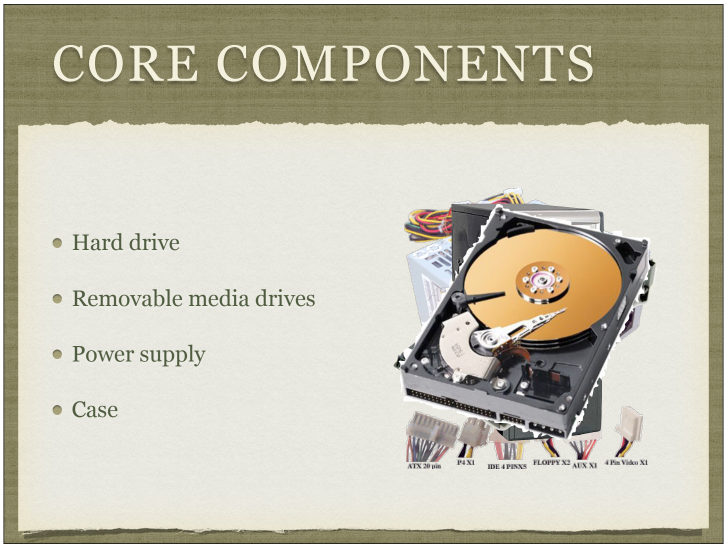
# HARDWARE

# CORE COMPONENTS

- Motherboard
- CPU
- RAM
- Expansion slots


Input-output connectors
Audio chip
PCI extension slots
AGP
Motherboard power supply connector
Processor support
Chipset (1)
Chipset (2)
Processor power supply connector
RAM connectors
CMOS battery
BIOS
Jumpers
Disk drive and floppy drive connectors
Case fan power supply connector

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

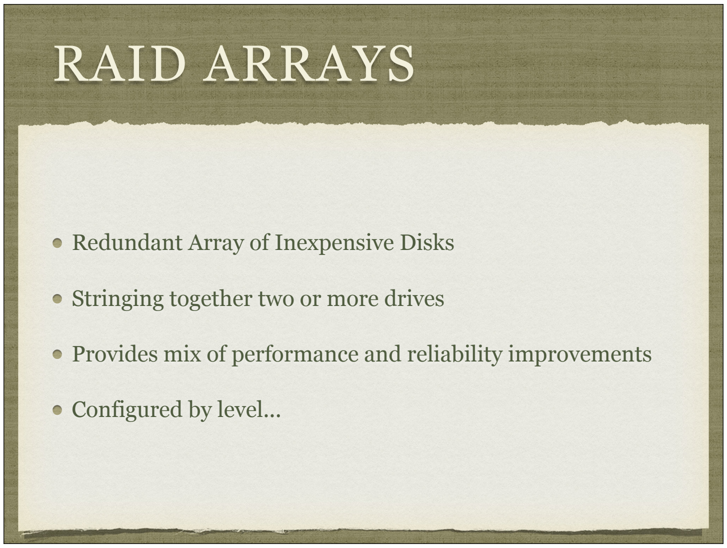


# CORE COMPONENTS

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

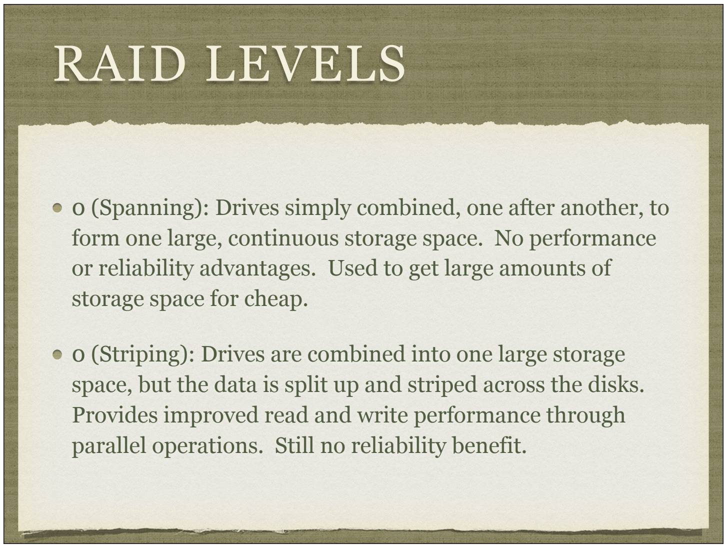
# PERIPHERALS

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



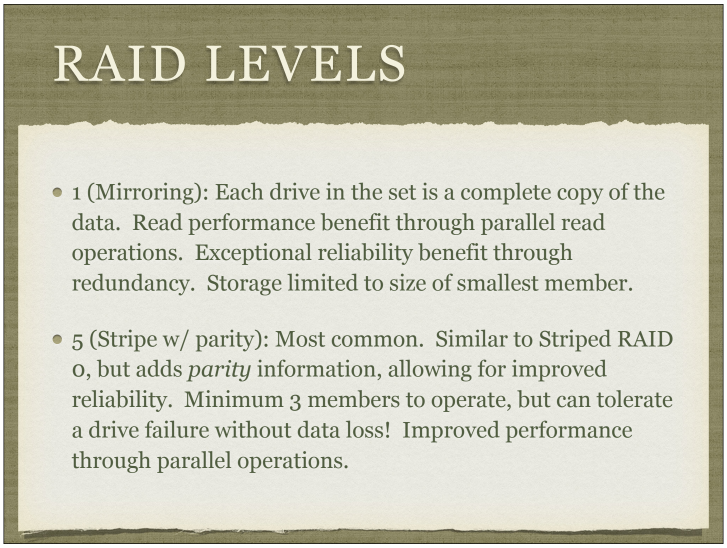
# RAID ARRAYS

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



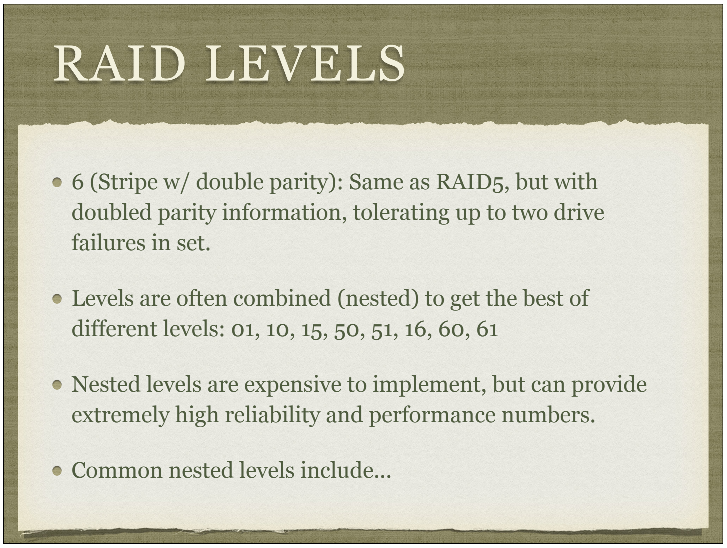
# RAID LEVELS

- 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.



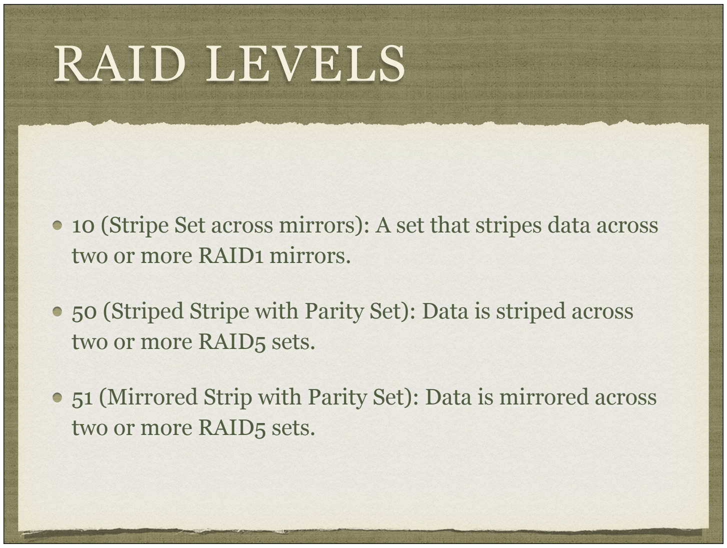
# RAID LEVELS

- 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.



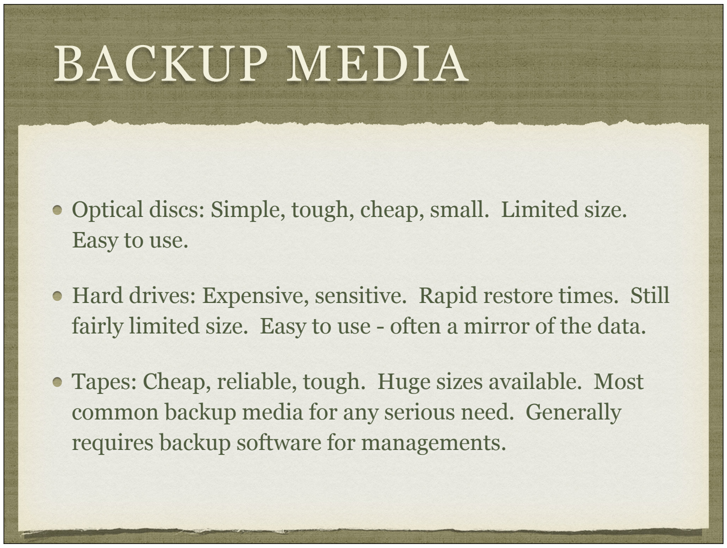
# RAID LEVELS

- 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.

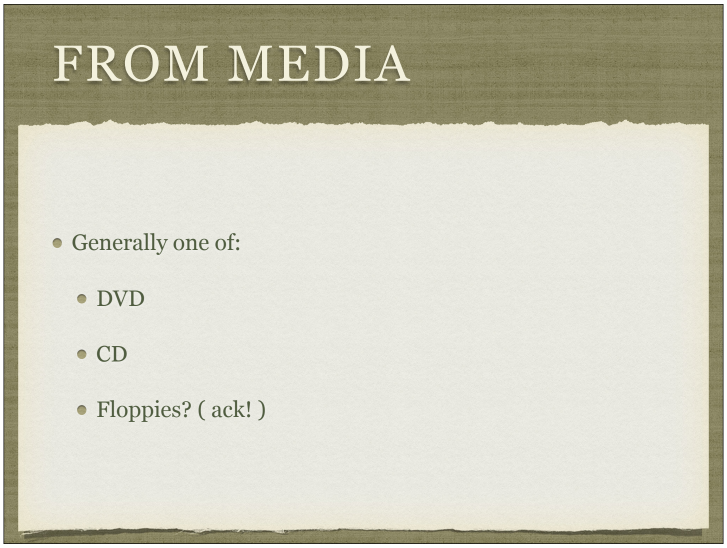


# BACKUP MEDIA

- 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.

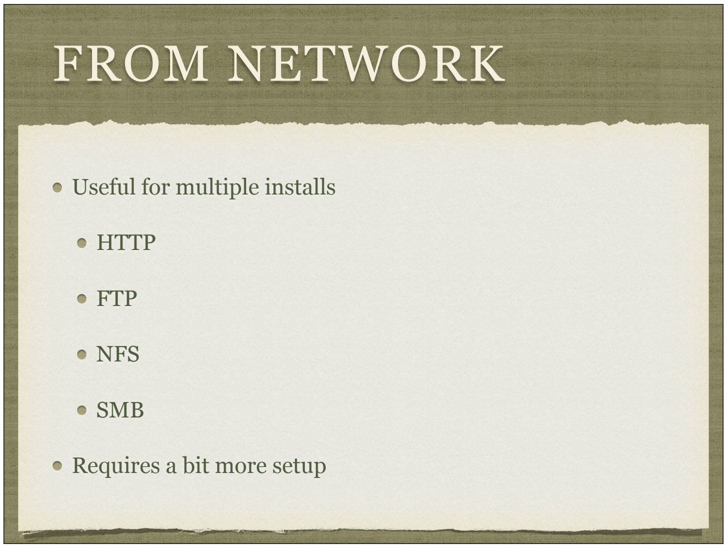
# LINUX INSTALLATION

Joy!



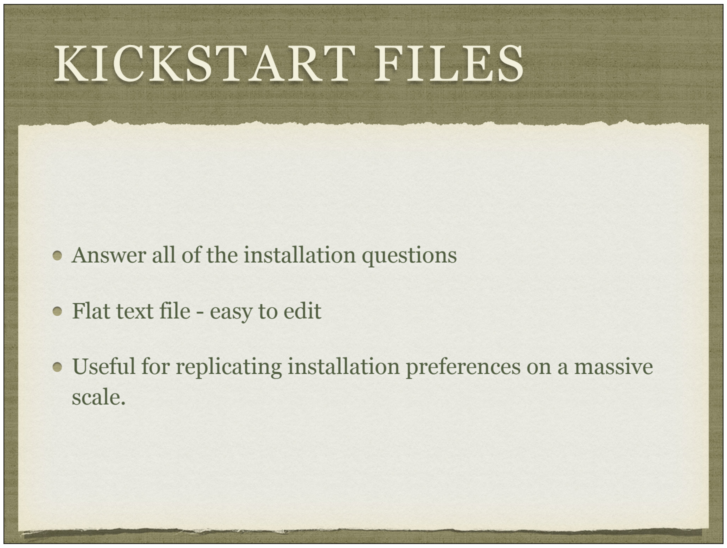
## FROM MEDIA

- Generally one of:
  - DVD
  - CD
  - Floppies? ( ack! )



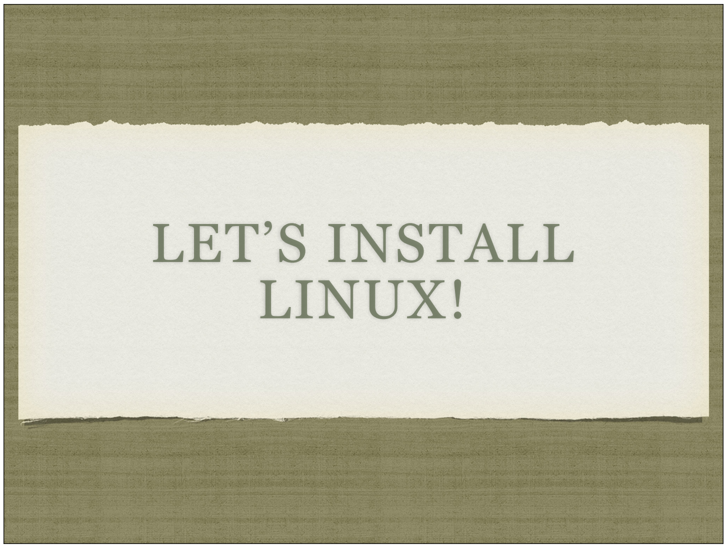
## FROM NETWORK

- Useful for multiple installs
  - HTTP
  - FTP
  - NFS
  - SMB
- Requires a bit more setup

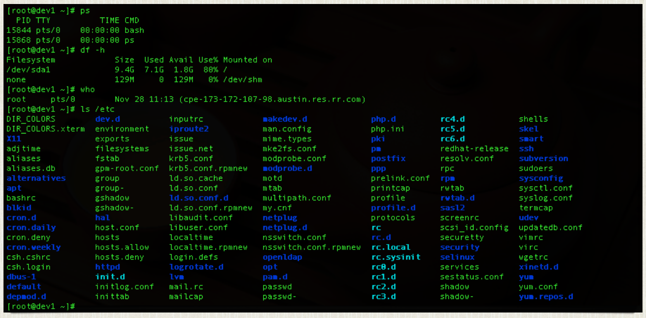


## KICKSTART FILES

- Answer all of the installation questions
- Flat text file - easy to edit
- Useful for replicating installation preferences on a massive scale.



# LET’S INSTALL LINUX!



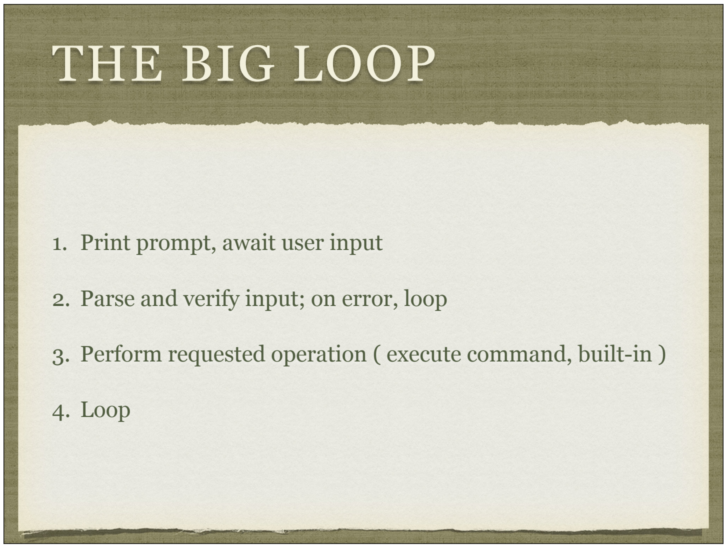
```
[root@dev1 ~]# ps
  PID TTY          TIME CMD
15844 pts/0    00:00:00 bash
15868 pts/0    00:00:00 ps
[root@dev1 ~]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             9.4G  7.1G  1.8G  80% /
none                  129M     0  129M   0% /dev/shm
[root@dev1 ~]# who
root     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*

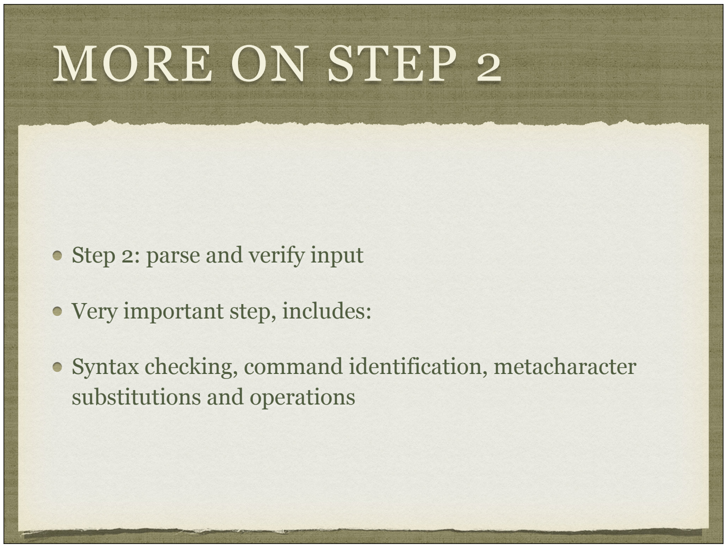
# THE BIG LOOP

- In order to master the shell, you have to understand it’s inner workings
- The first concept is *The Big Loop*



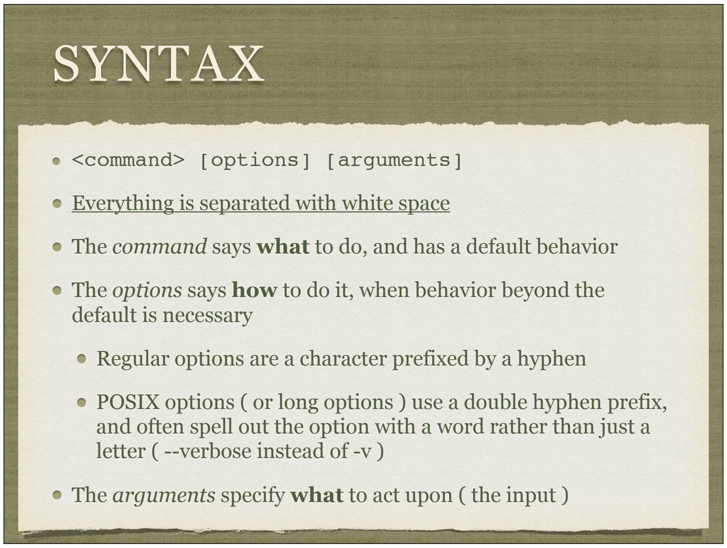
## THE BIG LOOP

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



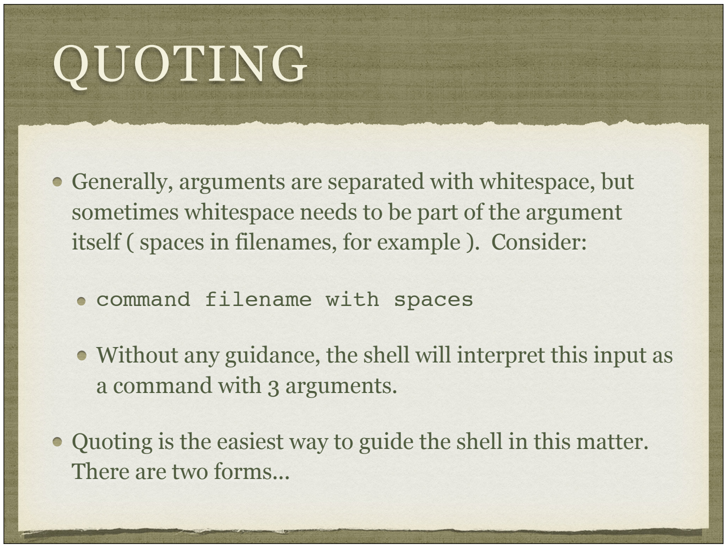
## MORE ON STEP 2

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



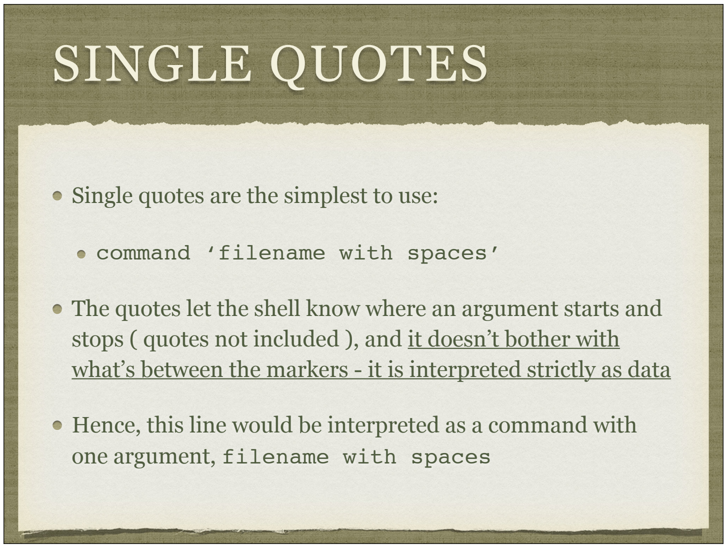
# SYNTAX

- `<command> [options] [arguments]`
- Everything is separated with white space
- The *command* says **what** to do, and has a default behavior
- The *options* says **how** to do it, when behavior beyond the default is necessary
  - Regular options are a character prefixed by 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 ( the input )



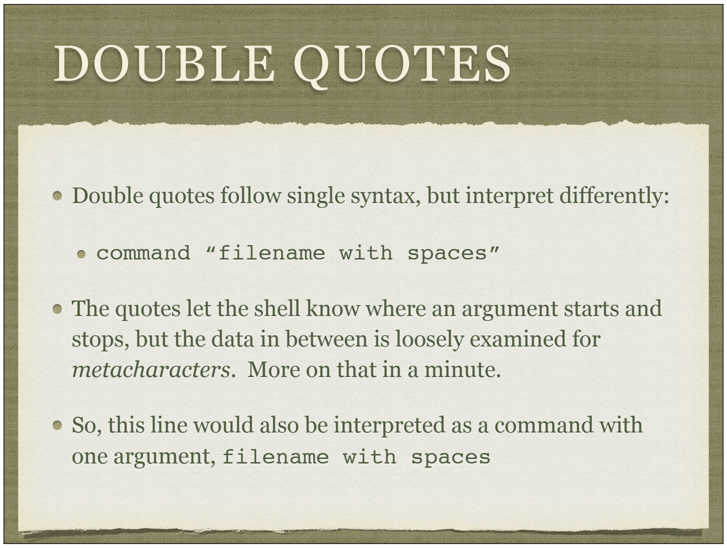
# QUOTING

- 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...



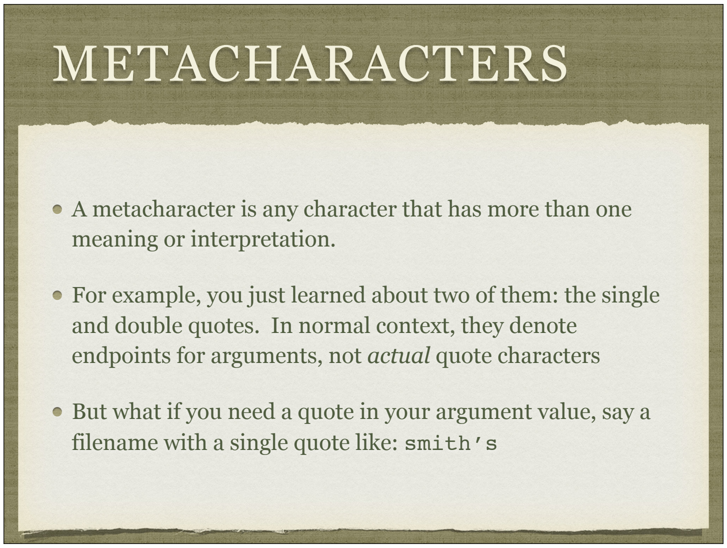
# SINGLE QUOTES

- 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`



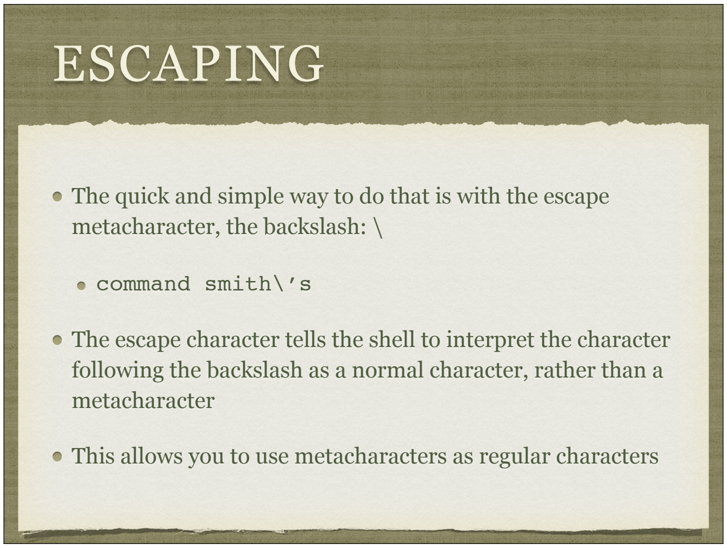
# DOUBLE QUOTES

- 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

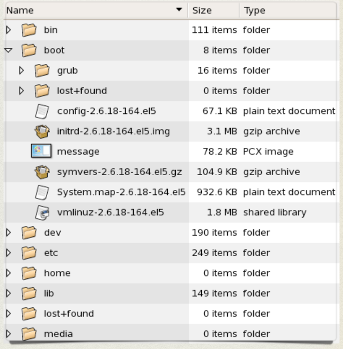
- The quick and simple way to do that is with the escape metacharacter, the backslash: \
  - `command smith\'s`
- The escape character tells the shell to interpret the character following the backslash as a normal character, rather than a metacharacter
- This allows you to use metacharacters as regular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

# BASIC COMMANDS

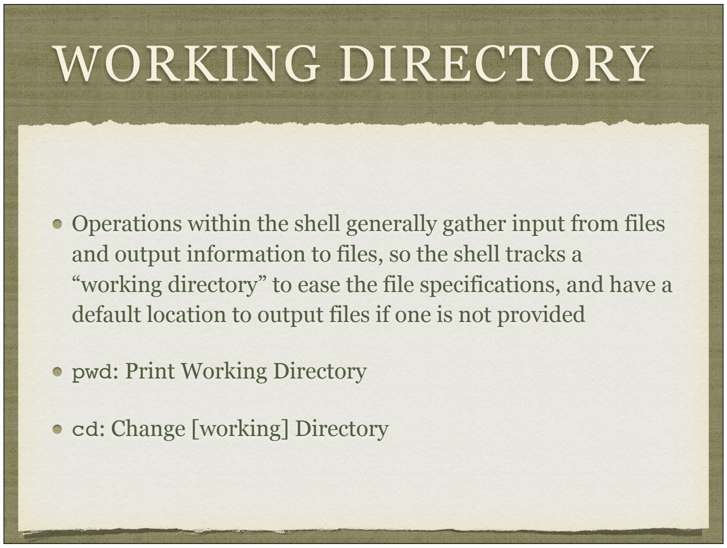
- `who`: Lists currently logged in users
- `uptime`: Statistics about machine usage and run time
- `echo`: Prints the given arguments to the screen
- `date`: Print current date and time
- `exit`: Terminate current shell session
- `reset`: Reset terminal state to default settings

# 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

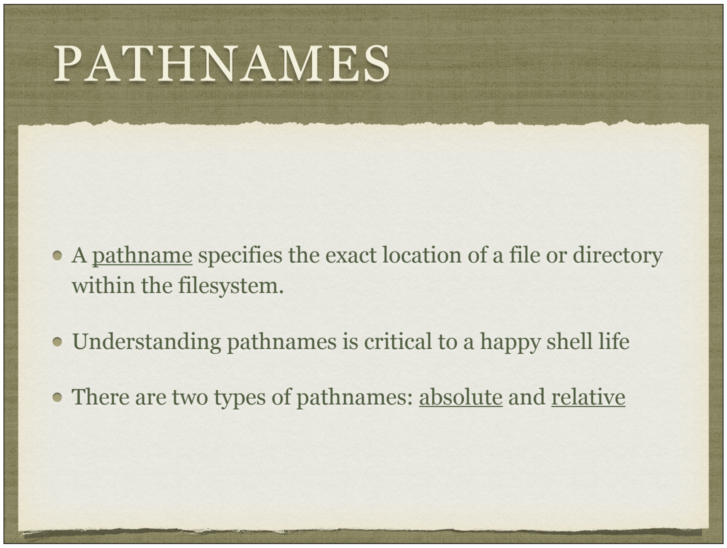


| Name | Size | Type |
|---|---|---|
| bin | 111 items | folder |
| boot | 8 items | folder |
| grub | 16 items | folder |
| lost+found | 0 items | folder |
| config-2.6.18-164.el5 | 67.1 KB | plain text document |
| initrd-2.6.18-164.el5.img | 3.1 MB | gzip archive |
| message | 78.2 KB | PCX image |
| symvers-2.6.18-164.el5.gz | 104.9 KB | gzip archive |
| System.map-2.6.18-164.el5 | 932.6 KB | plain text document |
| vmlinuz-2.6.18-164.el5 | 1.8 MB | shared library |
| dev | 190 items | folder |
| etc | 249 items | folder |
| home | 0 items | folder |
| lib | 149 items | folder |
| lost+found | 0 items | folder |
| media | 0 items | folder |



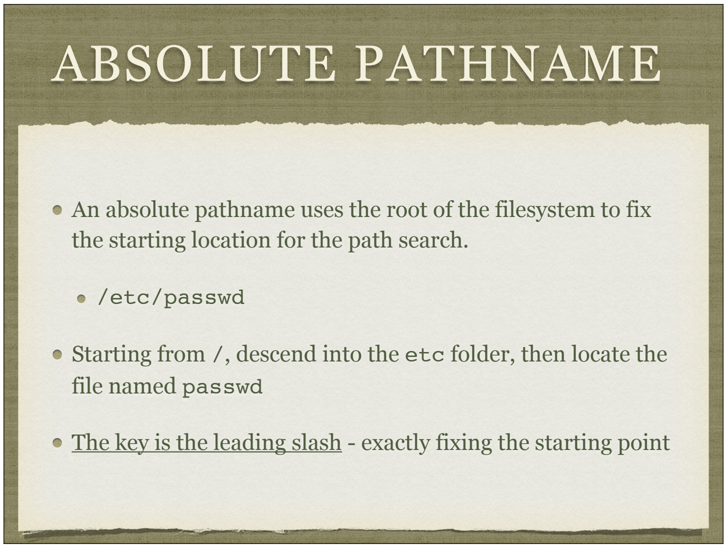
## WORKING DIRECTORY

- 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



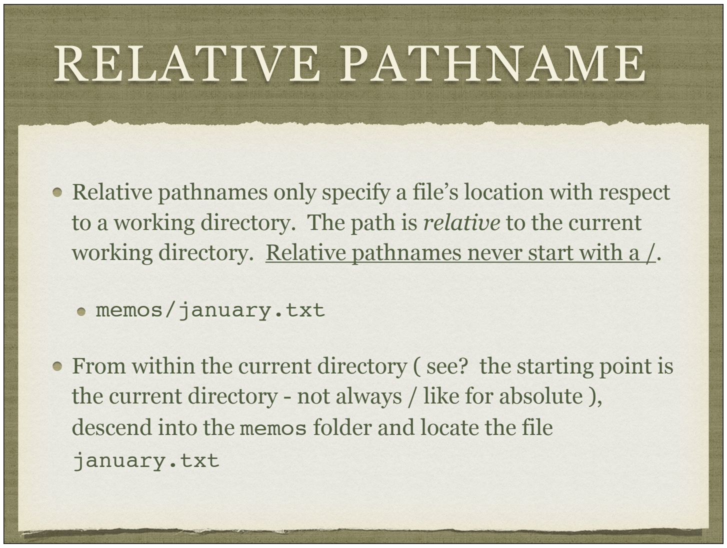
## PATHNAMES

- 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



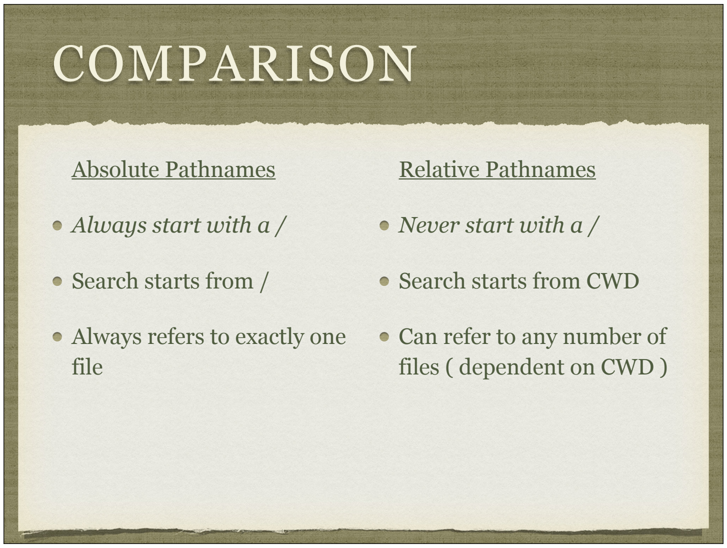
# ABSOLUTE PATHNAME

- 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



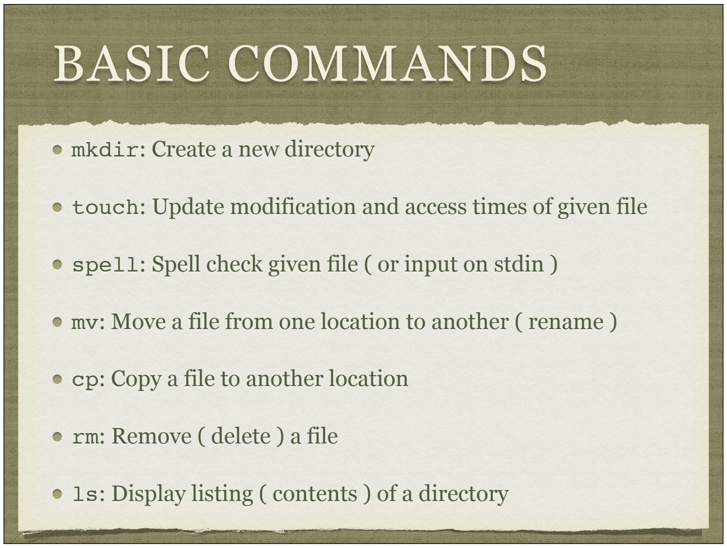
# RELATIVE PATHNAME

- 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`



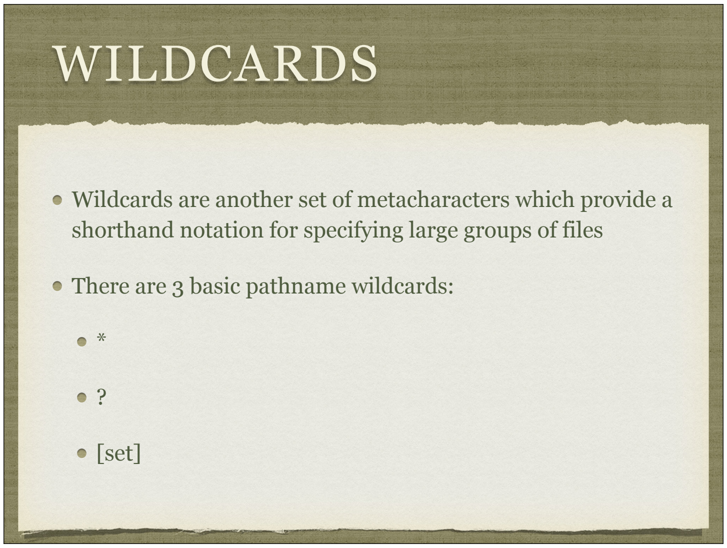
# COMPARISON

| Absolute Pathnames | Relative Pathnames |
|---|---|
| *Always start with a /* | *Never start with a /* |
| Search starts from / | Search starts from CWD |
| Always refers to exactly one file | Can refer to any number of files ( dependent on CWD ) |



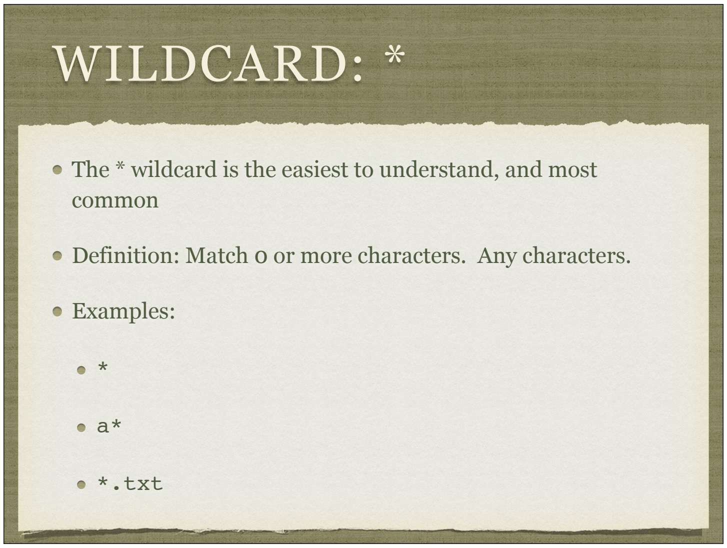
# BASIC COMMANDS

- `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



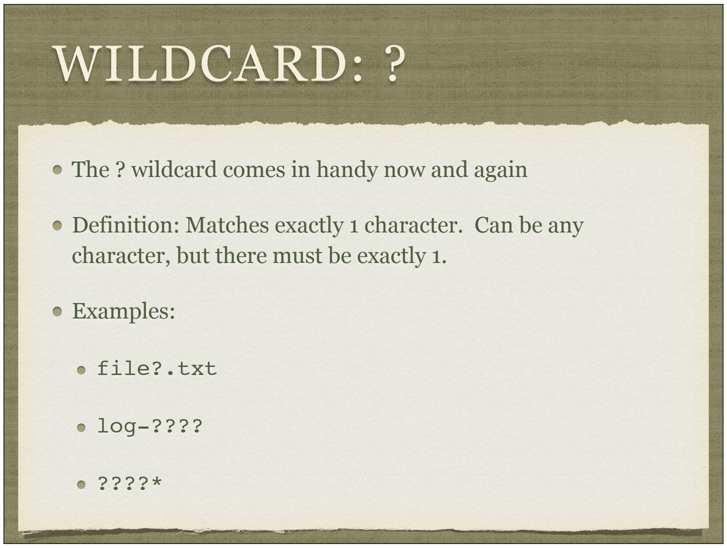
# WILDCARDS

- Wildcards are another set of metacharacters which provide a shorthand notation for specifying large groups of files
- There are 3 basic pathname wildcards:
  - *
  - ?
  - [set]



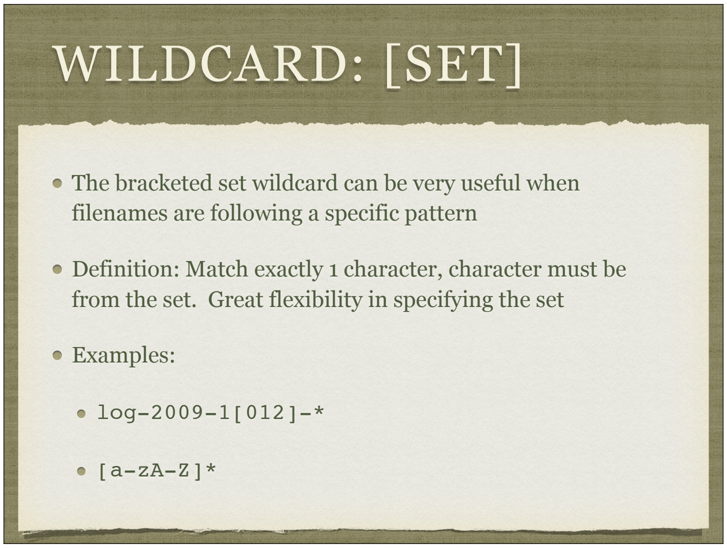
# WILDCARD: *

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



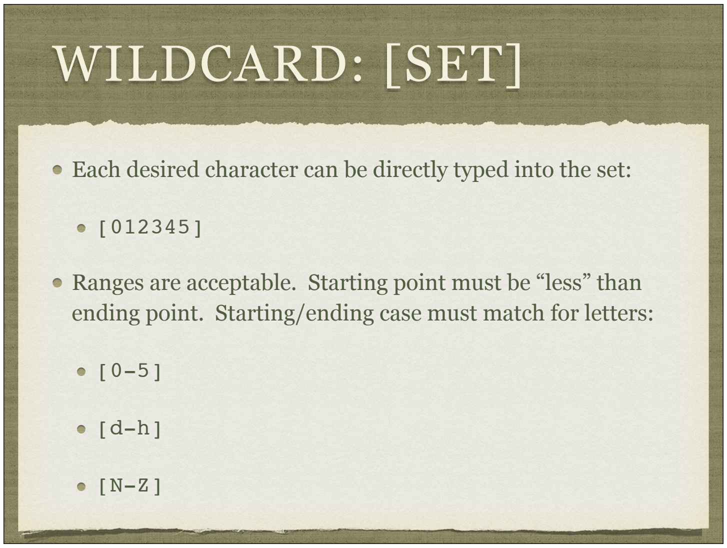
## WILDCARD: ?

- The ? wildcard comes in handy now and again
- Definition: Matches exactly 1 character. Can be any character, but there must be exactly 1.
- Examples:
  - `file?.txt`
  - `log-????`
  - `????*`



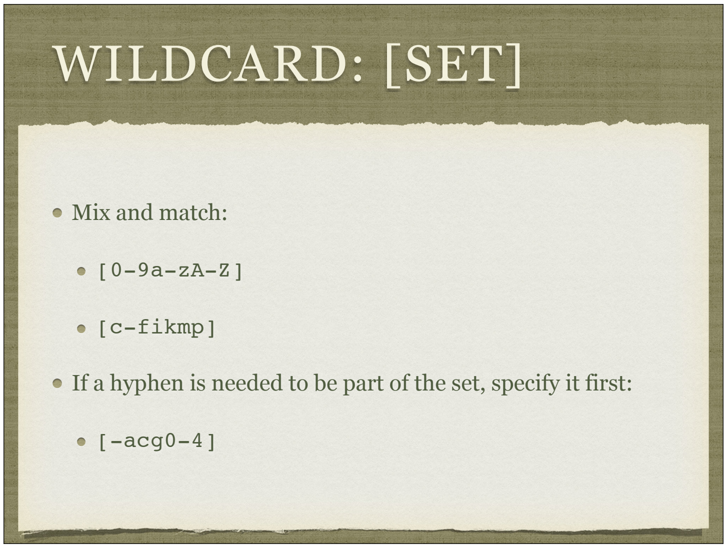
## WILDCARD: [SET]

- The bracketed set wildcard can be very useful when filenames are following a specific pattern
- Definition: Match exactly 1 character, character must be from the set. Great flexibility in specifying the set
- Examples:
  - `log-2009-1[012]-*`
  - `[a-zA-Z]*`



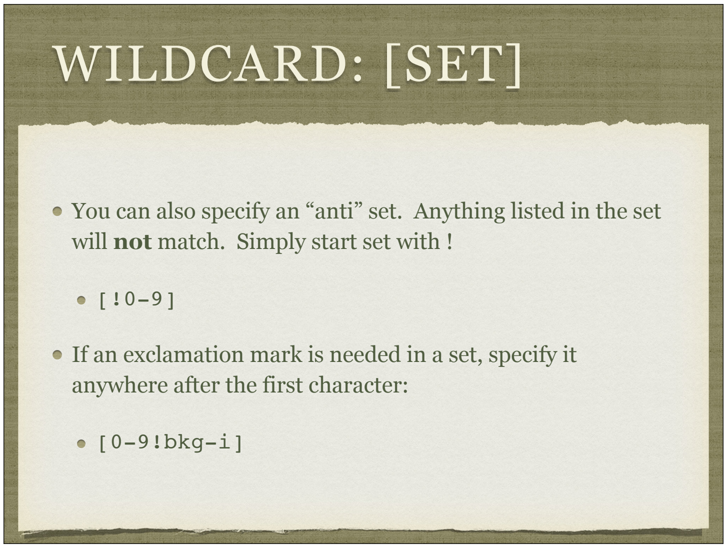
## WILDCARD: [SET]

- Each desired character can be directly typed into the set:
  - `[012345]`
- Ranges are acceptable.  Starting point must be “less” than ending point.  Starting/ending case must match for letters:
  - `[0-5]`
  - `[d-h]`
  - `[N-Z]`



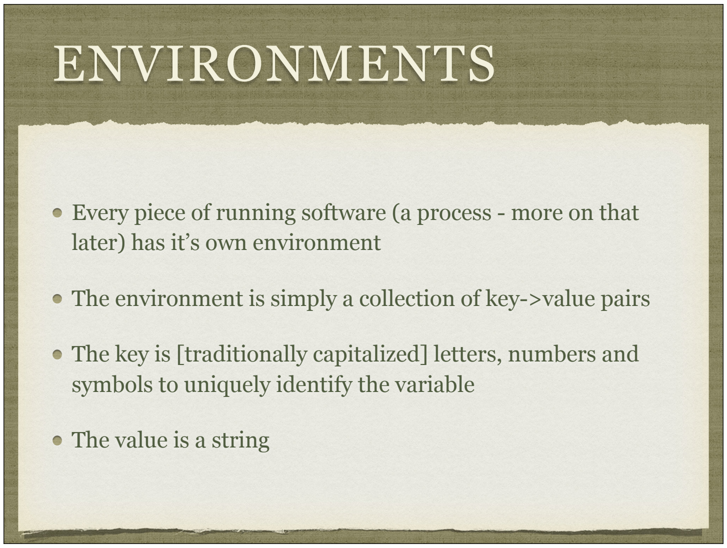
## WILDCARD: [SET]

- Mix and match:
  - `[0-9a-zA-Z]`
  - `[c-fikmp]`
- If a hyphen is needed to be part of the set, specify it first:
  - `[-acg0-4]`



# WILDCARD: [SET]

- You can also specify an “anti” set. Anything listed in the set will **not** match. Simply start set with !
  - `[!0-9]`
- If an exclamation mark is needed in a set, specify it anywhere after the first character:
  - `[0-9!bkg-i]`

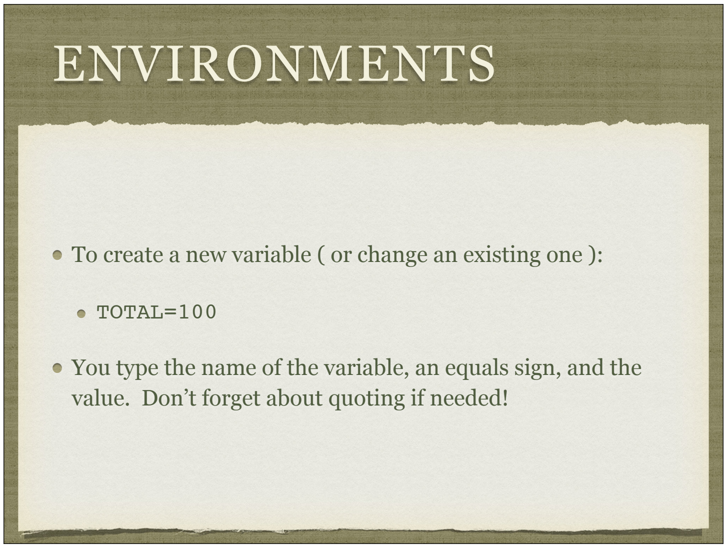


# ENVIRONMENTS

- 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

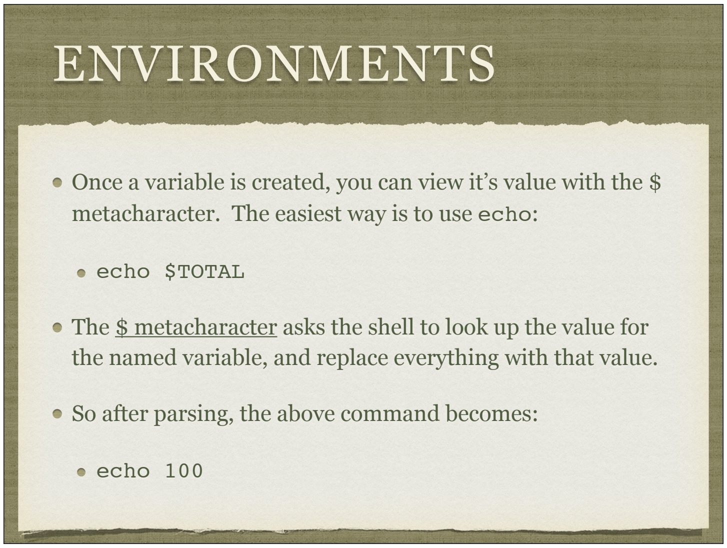
# ENVIRONMENTS

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



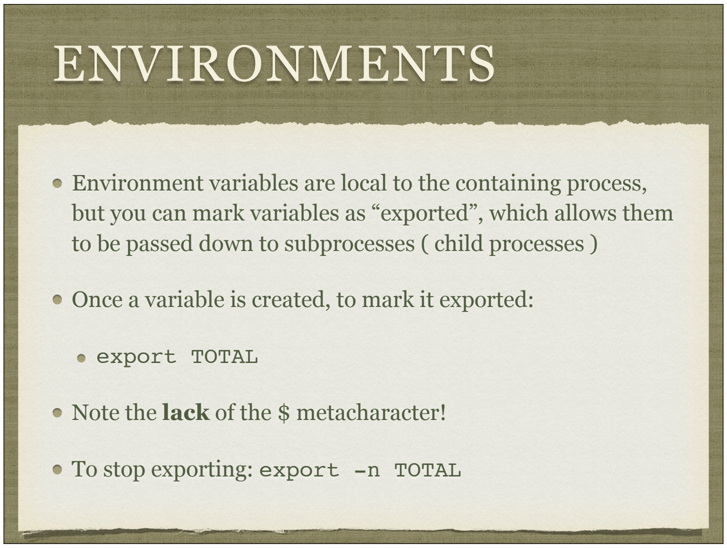
# ENVIRONMENTS

- 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!



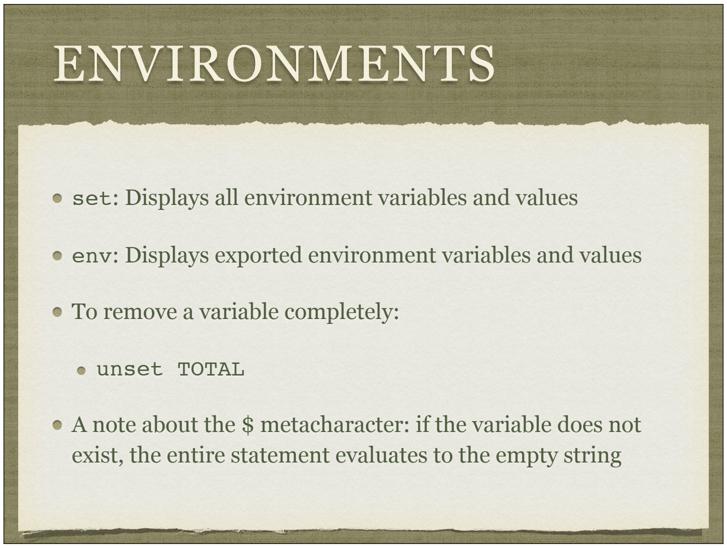
## ENVIRONMENTS

- 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`



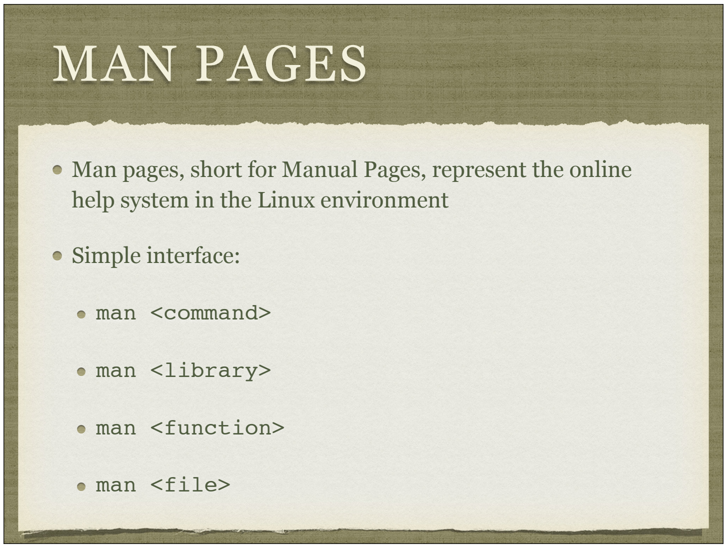
## ENVIRONMENTS

- 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`



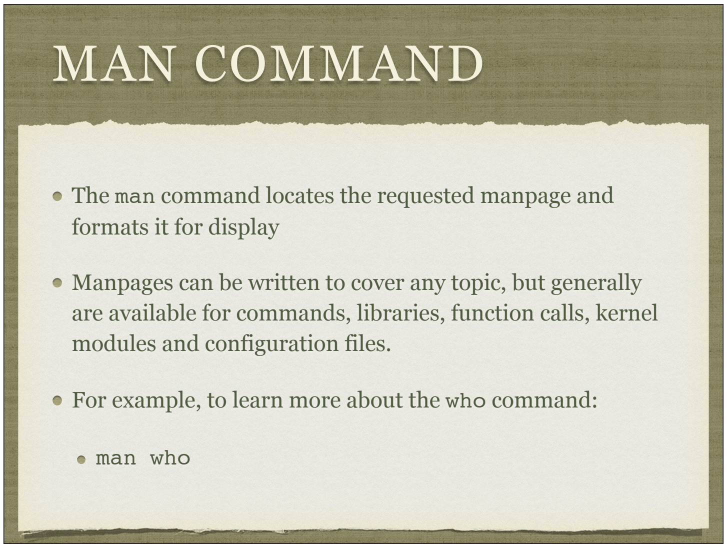
# ENVIRONMENTS

- `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



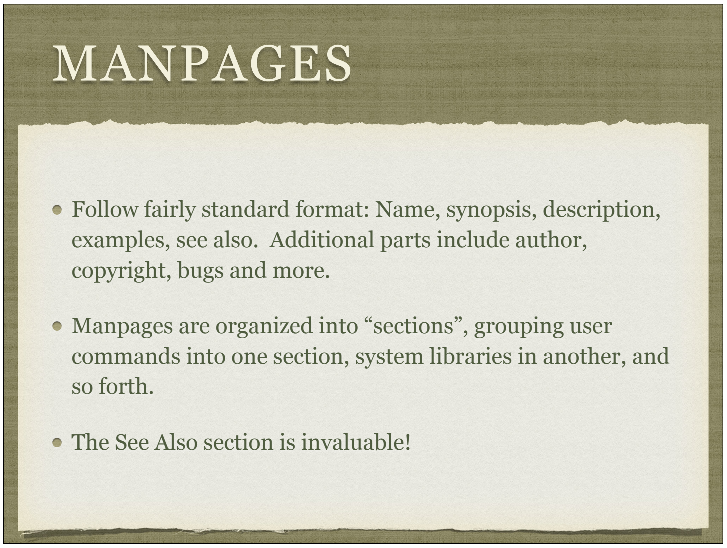
# MAN PAGES

- 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>`



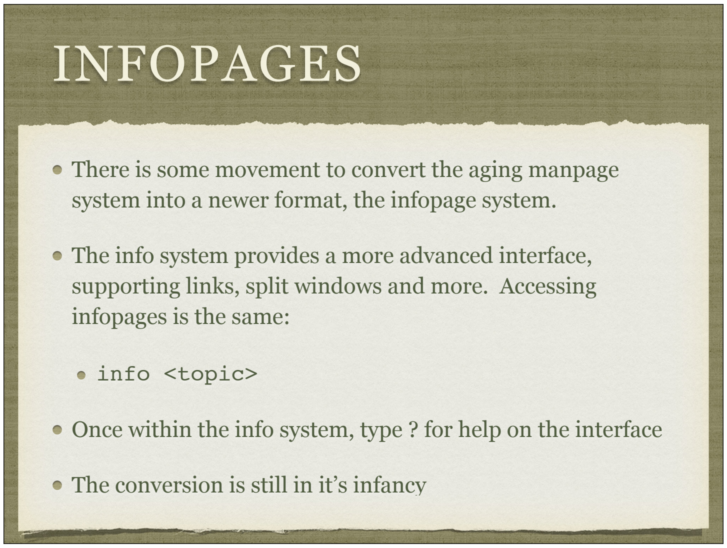
# MAN COMMAND

- 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`



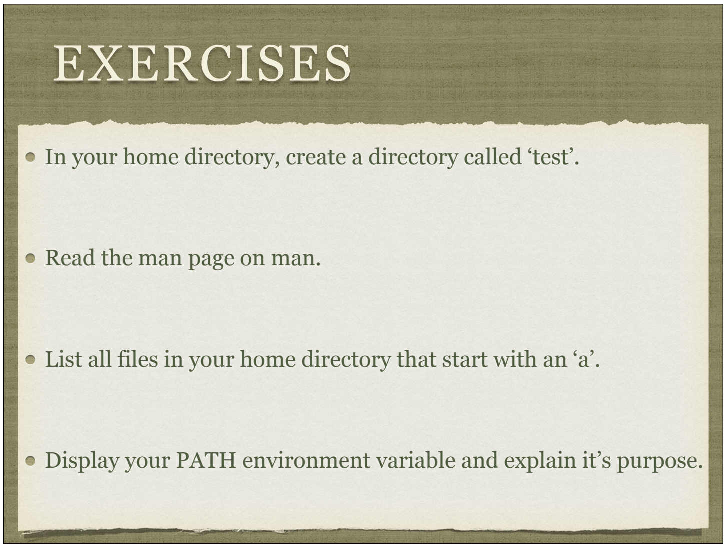
# MANPAGES

- 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!



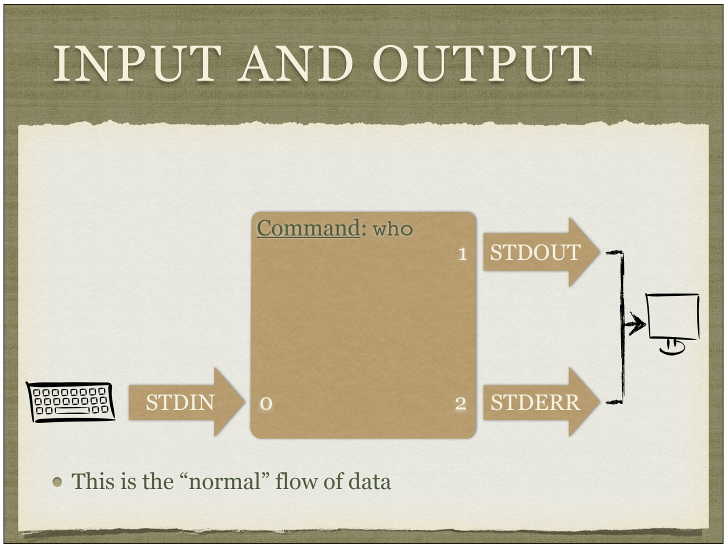
# INFOPAGES

- 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'.
- Display your PATH environment variable and explain it's purpose.



# INPUT AND OUTPUT

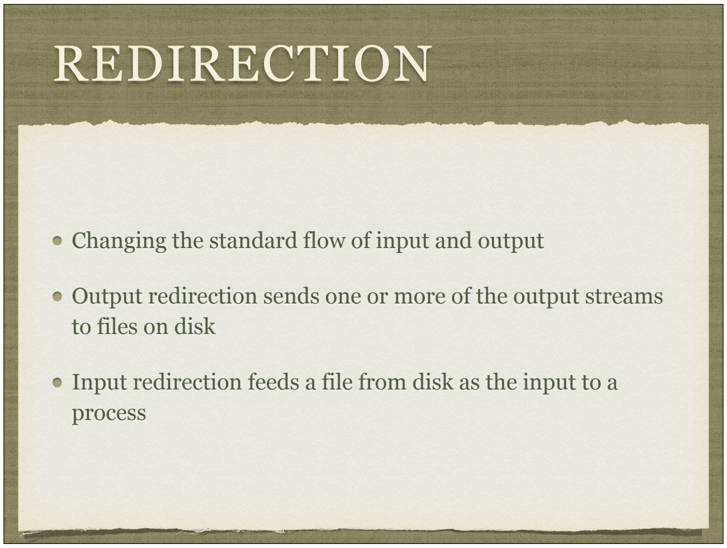
Command: who

1 STDOUT

STDIN 0

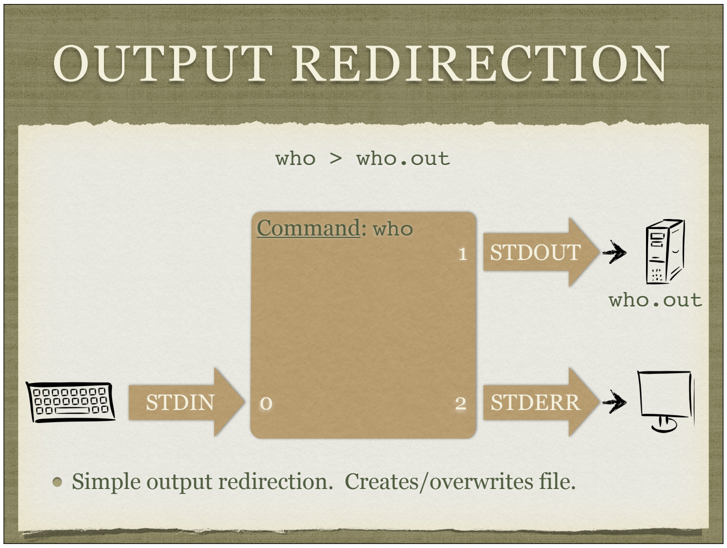
2 STDERR

- 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
```

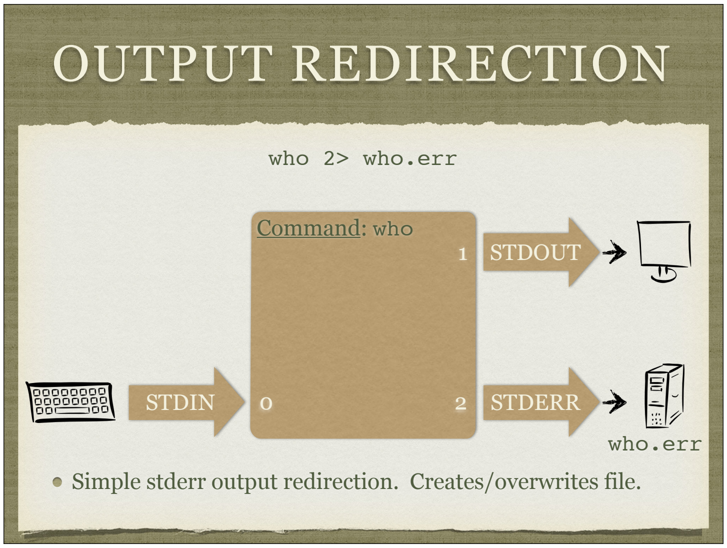
Command: who

STDIN → 0

1 STDOUT → who.out

2 STDERR →

- Simple output redirection. Creates/overwrites file.



# OUTPUT REDIRECTION

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

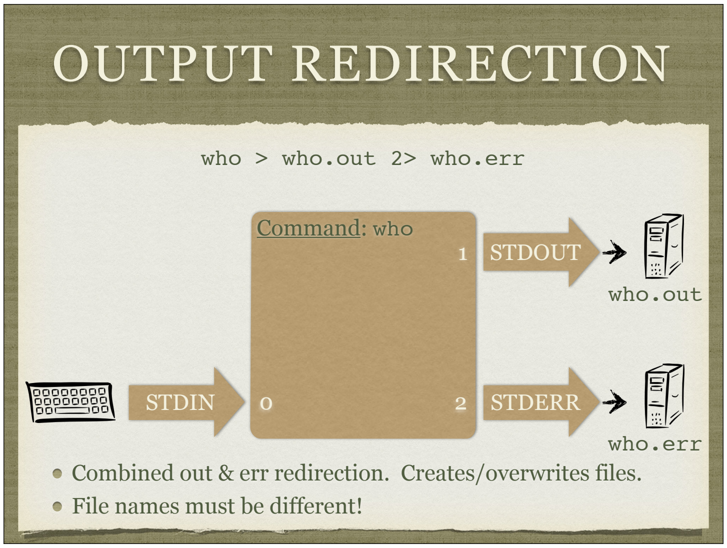
Command: who

STDIN → 0

1 STDOUT →

2 STDERR → who.err

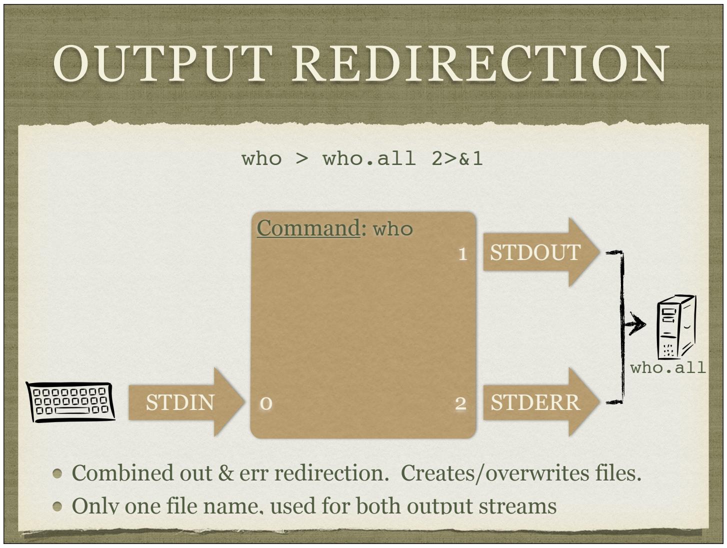
- Simple stderr output redirection. Creates/overwrites file.



# OUTPUT REDIRECTION

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

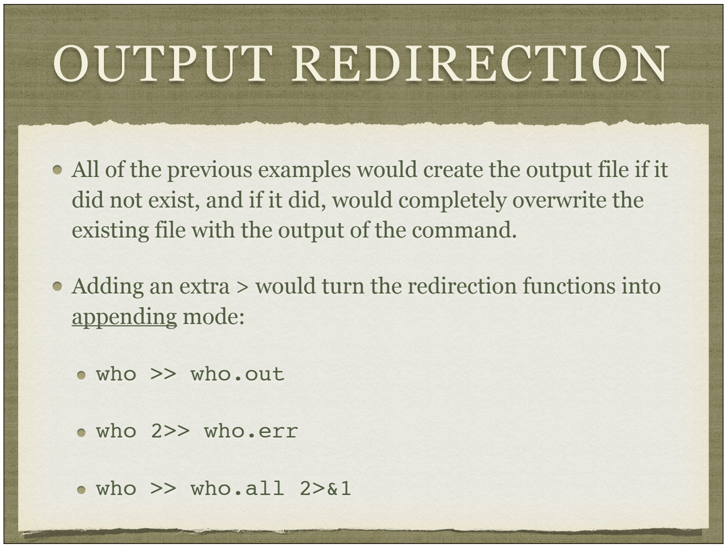
- Combined out & err redirection.  Creates/overwrites files.
- File names must be different!



# OUTPUT REDIRECTION

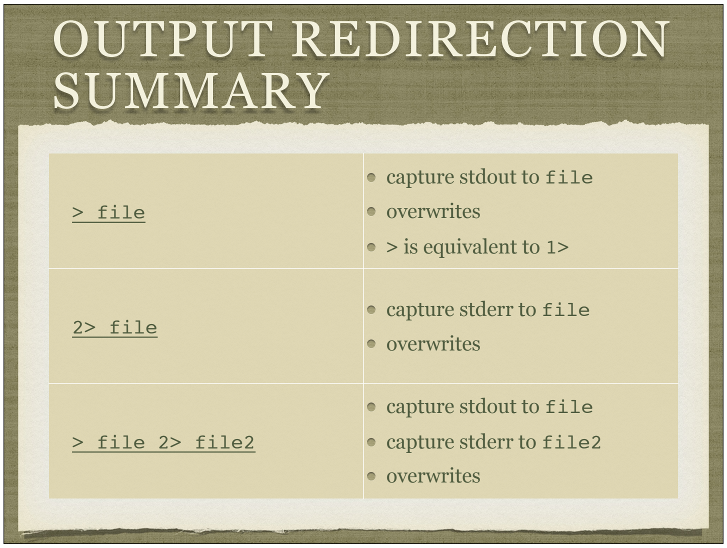
```
who > who.all 2>&1
```

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



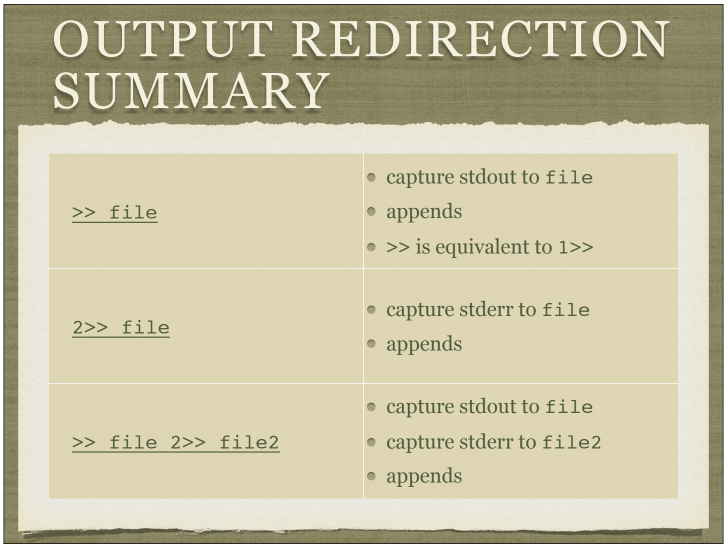
# OUTPUT REDIRECTION

- 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`



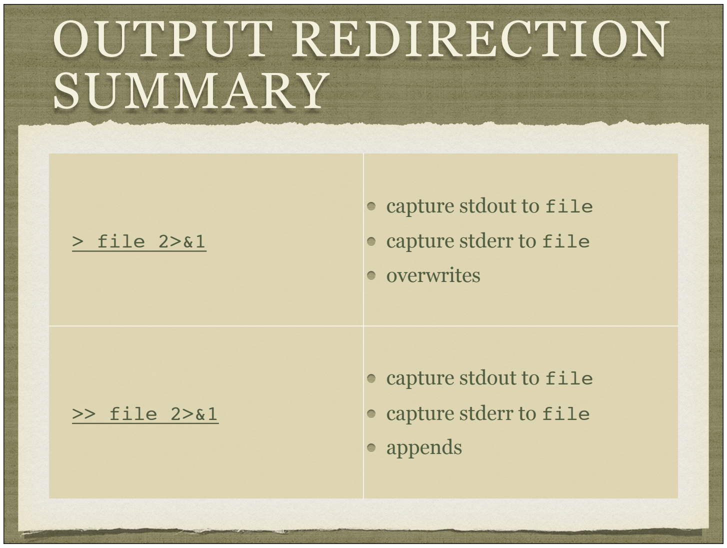
# OUTPUT REDIRECTION SUMMARY

| | |
|---|---|
| `> file` | capture stdout to `file`<br>overwrites<br>> is equivalent to `1>` |
| `2> file` | capture stderr to `file`<br>overwrites |
| `> file 2> file2` | capture stdout to `file`<br>capture stderr to `file2`<br>overwrites |



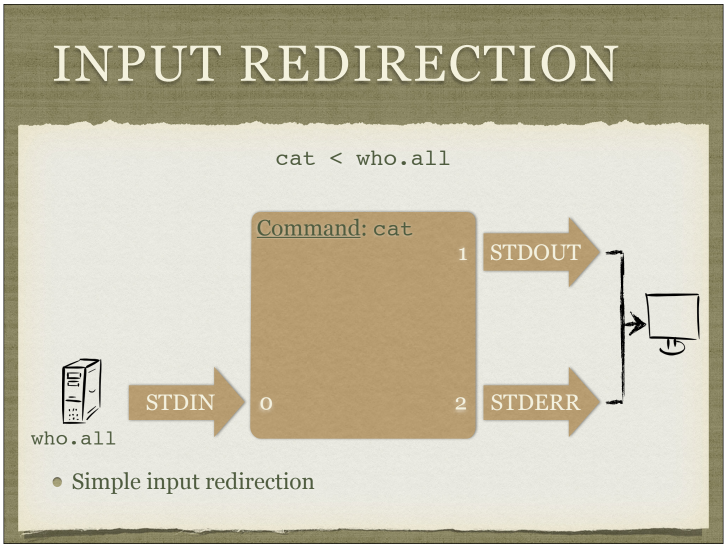
# OUTPUT REDIRECTION SUMMARY

| | |
|---|---|
| `>> file` | - capture stdout to `file`<br>- appends<br>- >> is equivalent to `1>>` |
| `2>> file` | - capture stderr to `file`<br>- appends |
| `>> file 2>> file2` | - capture stdout to `file`<br>- capture stderr to `file2`<br>- appends |



# OUTPUT REDIRECTION SUMMARY

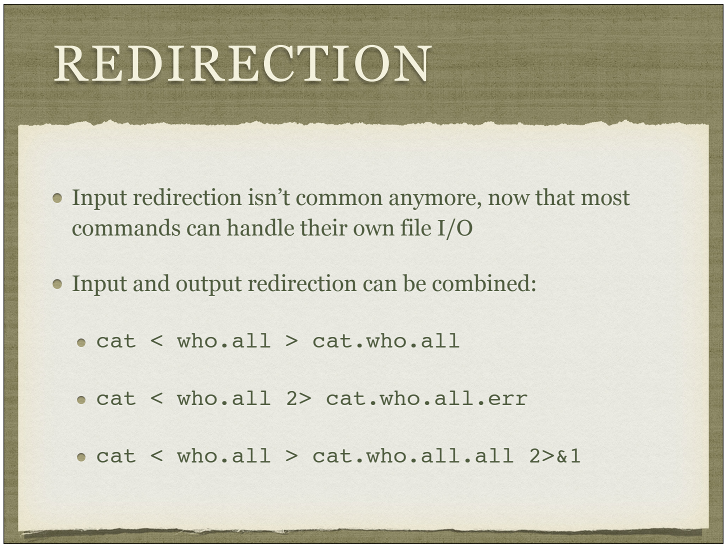
| | |
|---|---|
| `> file 2>&1` | - capture stdout to `file`<br>- capture stderr to `file`<br>- overwrites |
| `>> file 2>&1` | - capture stdout to `file`<br>- capture stderr to `file`<br>- appends |



# INPUT REDIRECTION

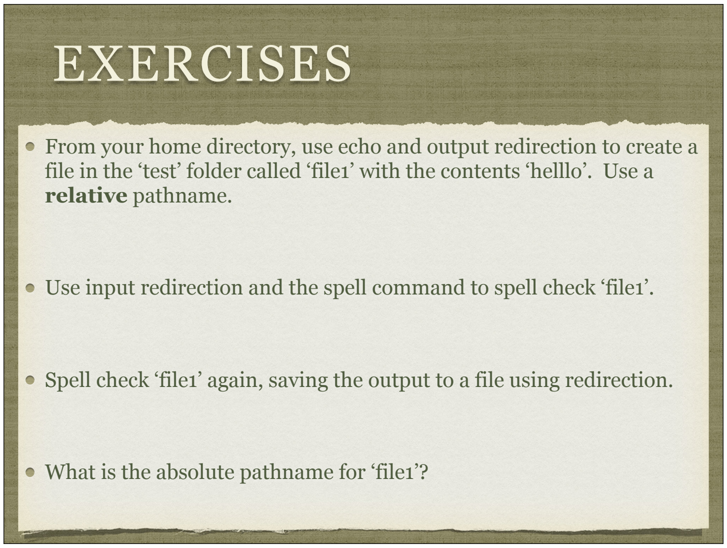
`cat < who.all`

- Simple input redirection



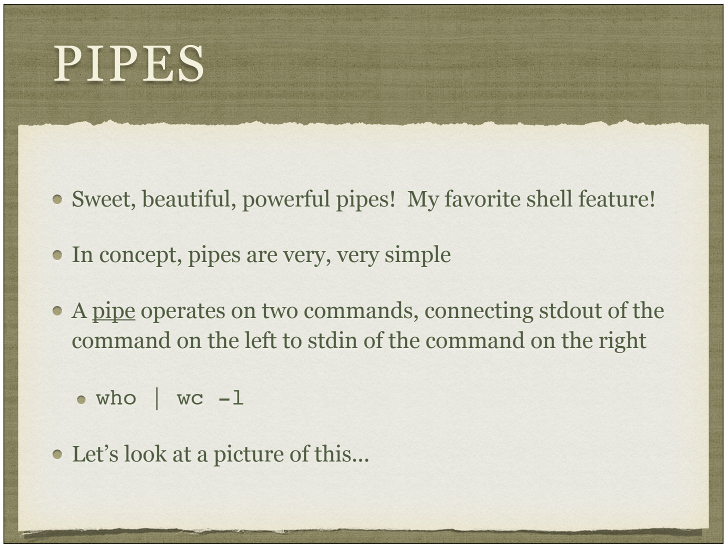
# REDIRECTION

- 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’?

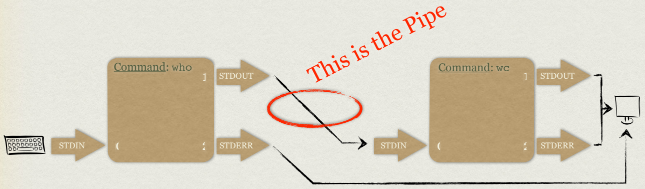


# PIPES

- 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

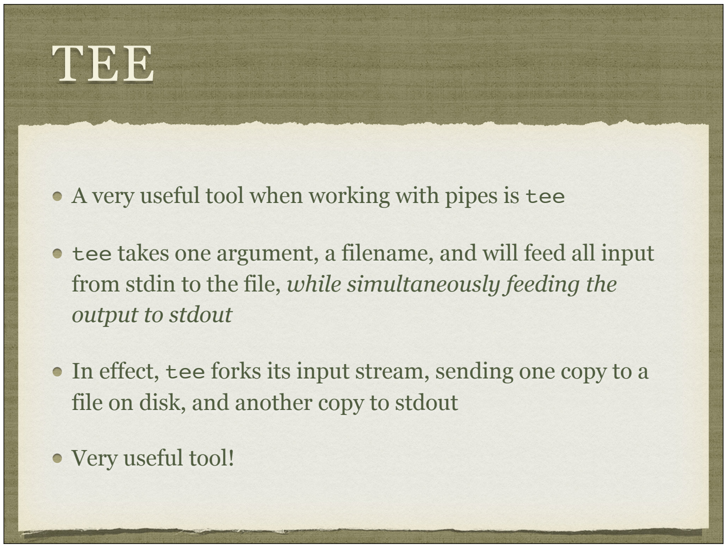
```
who | wc -l
```



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

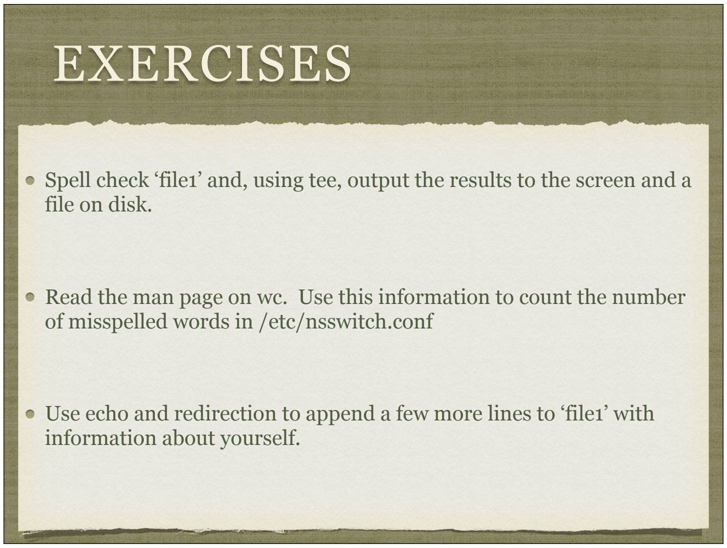
# PIPES

- 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!



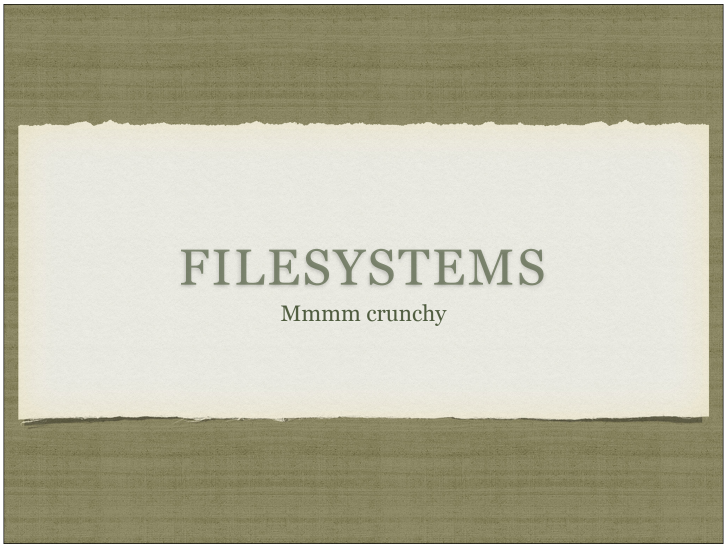
# TEE

- 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!



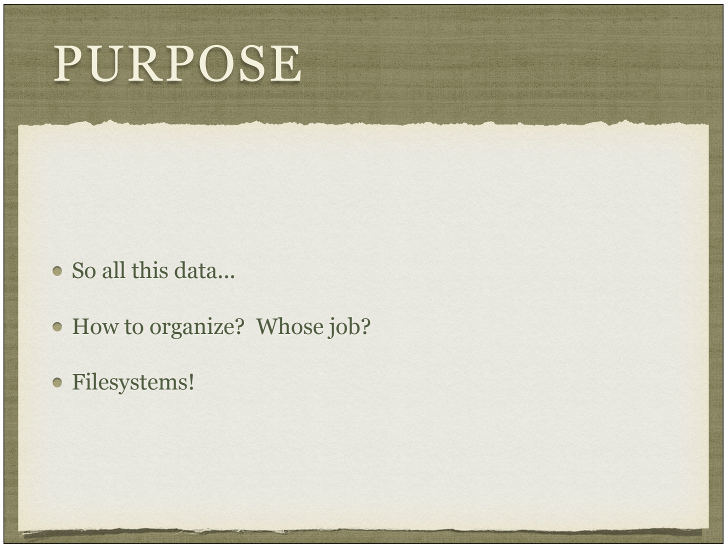
# EXERCISES

- 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.



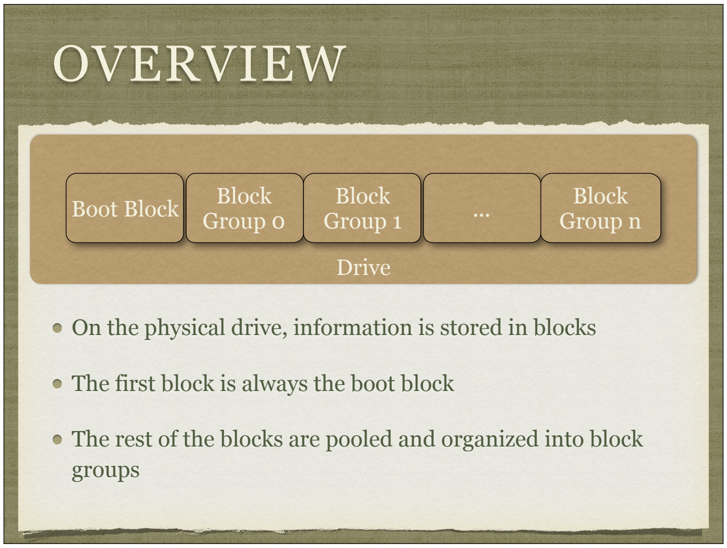
# FILESYSTEMS

Mmmm crunchy



# PURPOSE

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

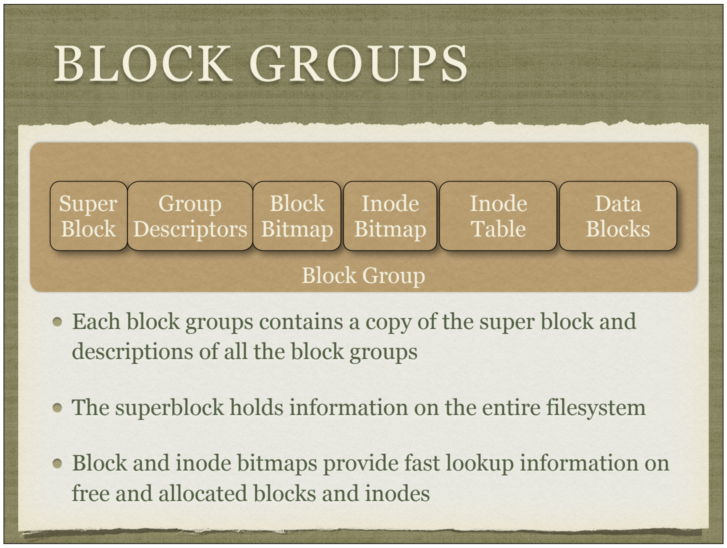


# OVERVIEW

| | | | | |
|---|---|---|---|---|
| Boot Block | Block Group 0 | Block Group 1 | ... | Block Group n |

Drive

- On the physical drive, information is stored in blocks
- The first block is always the boot block
- The rest of the blocks are pooled and organized into block groups

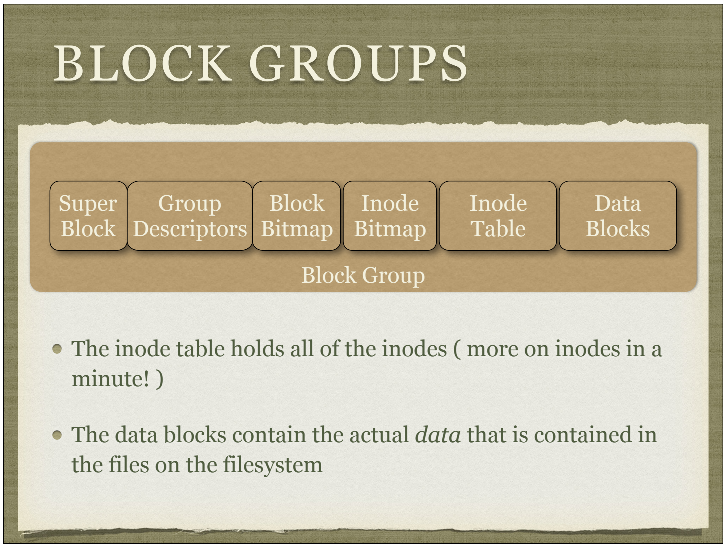


# BLOCK GROUPS

| | | | | | |
|---|---|---|---|---|---|
| Super Block | Group Descriptors | Block Bitmap | Inode Bitmap | Inode Table | Data Blocks |

Block Group

- Each block groups contains a copy of the super block and descriptions of all the block groups
- The superblock holds information on the entire filesystem
- Block and inode bitmaps provide fast lookup information on free and allocated blocks and inodes

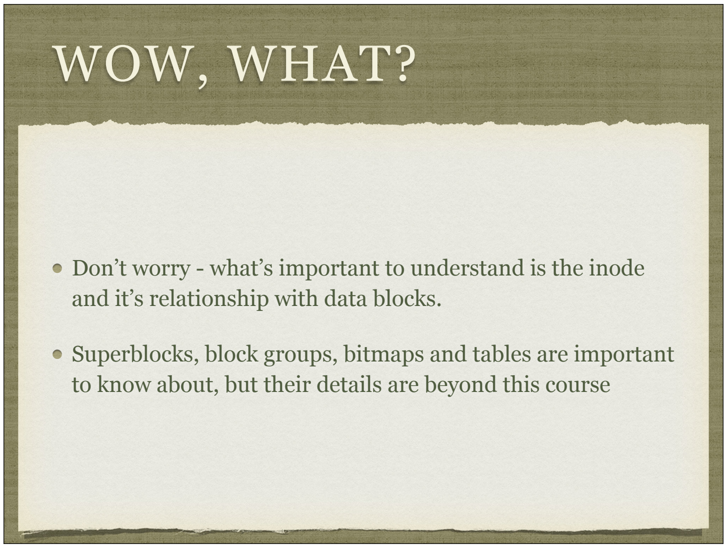


## BLOCK GROUPS

Super Block | Group Descriptors | Block Bitmap | Inode Bitmap | Inode Table | Data Blocks

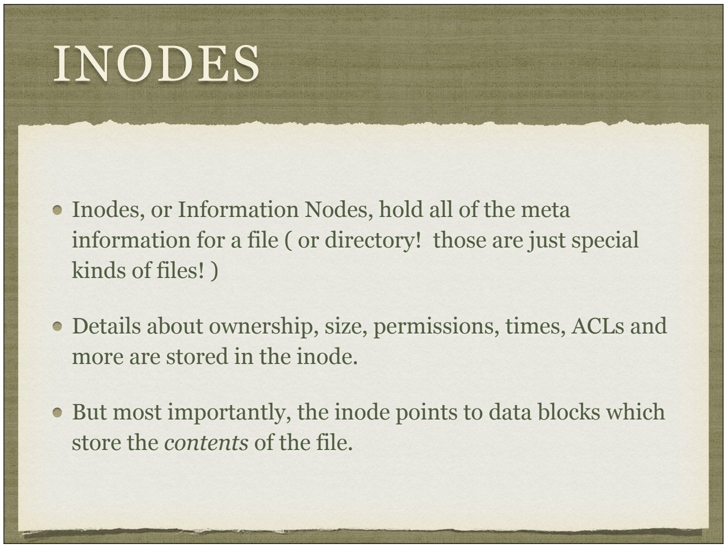
Block Group

- The inode table holds all of the inodes ( more on inodes in a minute! )
- The data blocks contain the actual *data* that is contained in the files on the filesystem



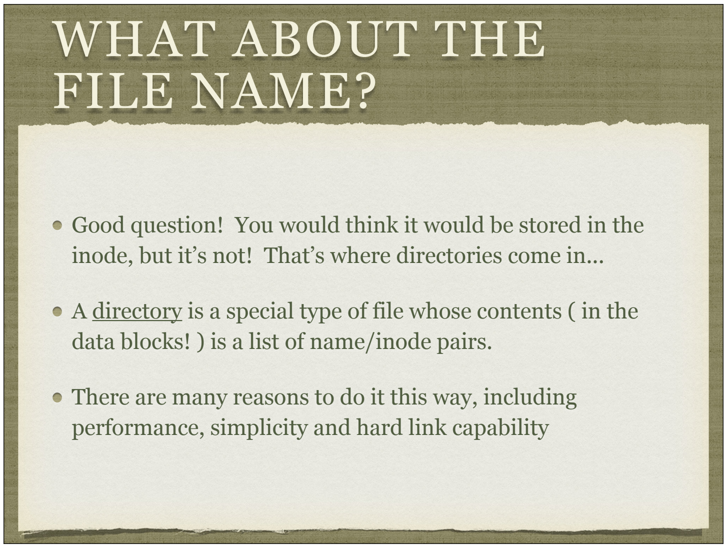
## WOW, WHAT?

- Don't worry - what's important to understand is the inode and it's relationship with data blocks.
- Superblocks, block groups, bitmaps and tables are important to know about, but their details are beyond this course



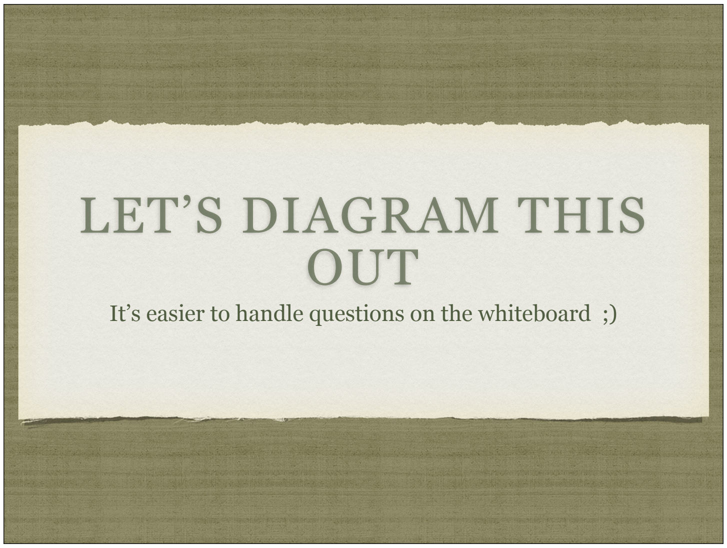
# INODES

- Inodes, or Information Nodes, hold all of the meta information for a file ( or directory!  those are just special kinds of files! )
- Details about ownership, size, permissions, times, ACLs and more are stored in the inode.
- But most importantly, the inode points to data blocks which store the *contents* of the file.



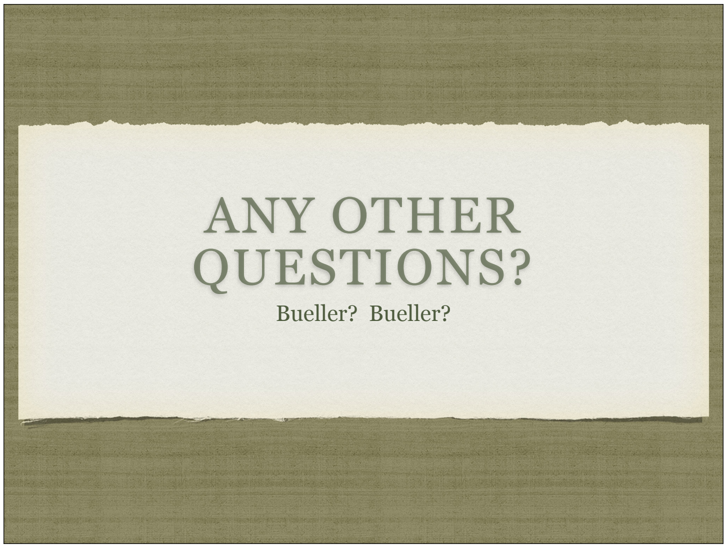
# WHAT ABOUT THE FILE NAME?

- Good question!  You would think it would be stored in the inode, but it's not!  That's where directories come in...
- A <u>directory</u> is a special type of file whose contents ( in the data blocks! ) is a list of name/inode pairs.
- There are many reasons to do it this way, including performance, simplicity and hard link capability



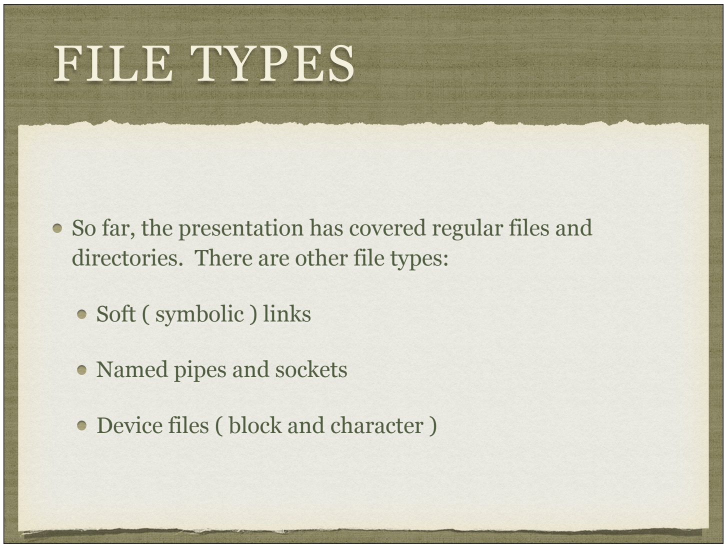
## LET’S DIAGRAM THIS OUT

It’s easier to handle questions on the whiteboard ;)



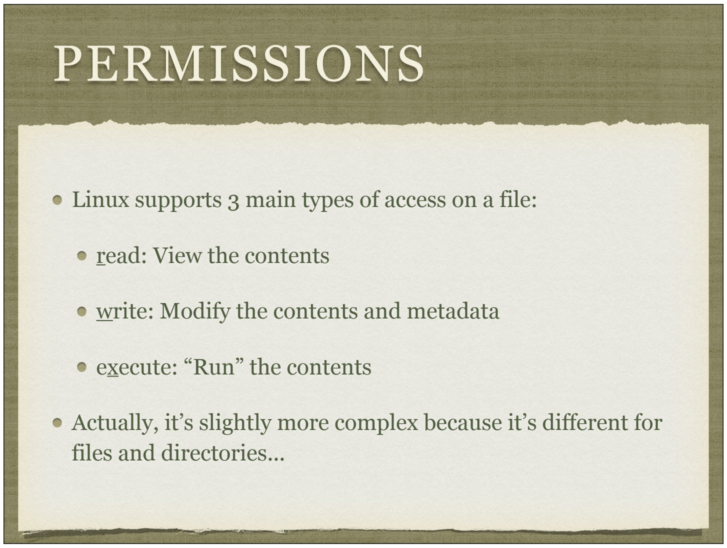
## ANY OTHER QUESTIONS?

Bueller? Bueller?



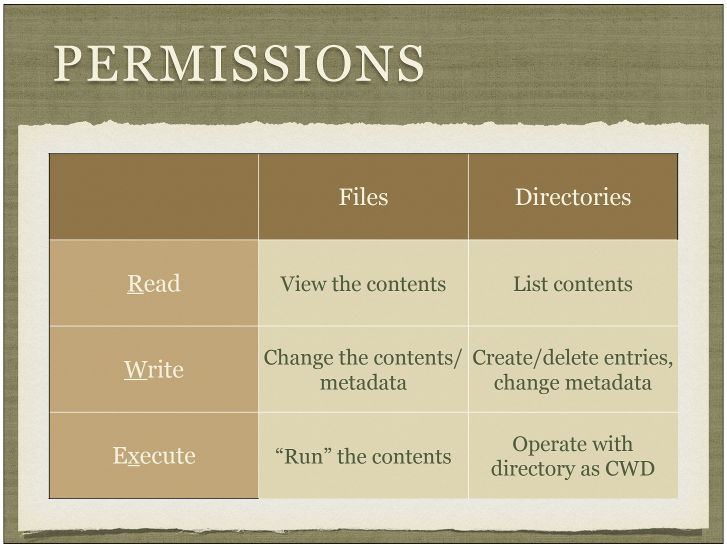
# FILE TYPES

- So far, the presentation has covered regular files and directories.  There are other file types:
  - Soft ( symbolic ) links
  - Named pipes and sockets
  - Device files ( block and character )



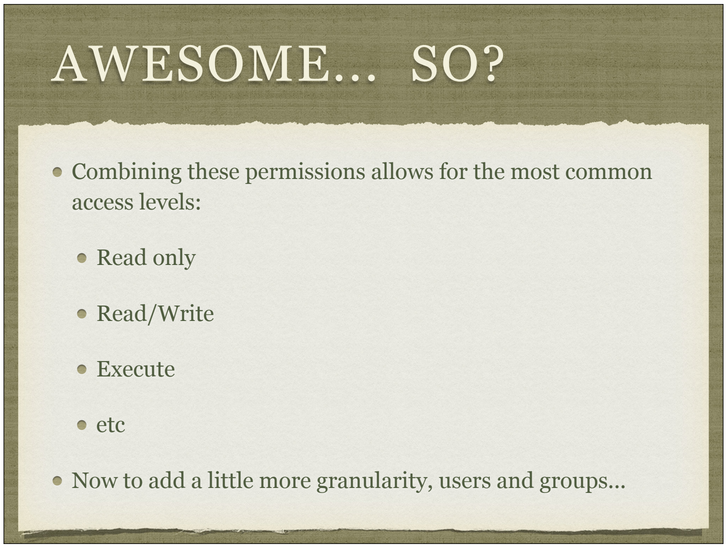
# PERMISSIONS

- 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...



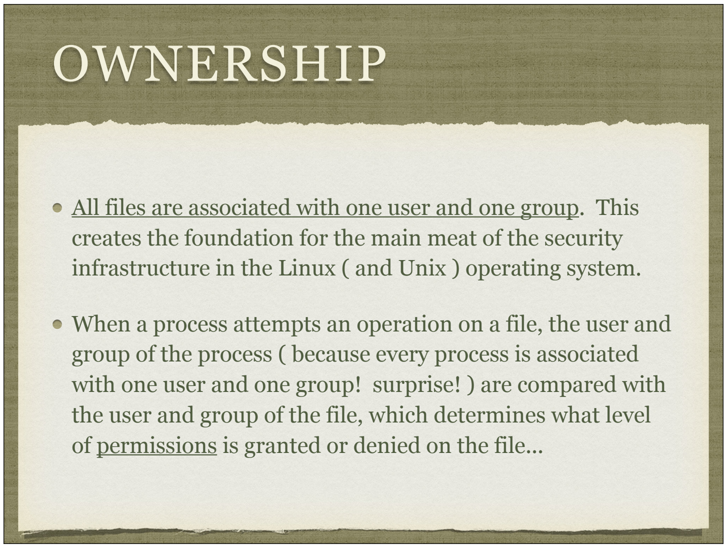
## PERMISSIONS

| | 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 |



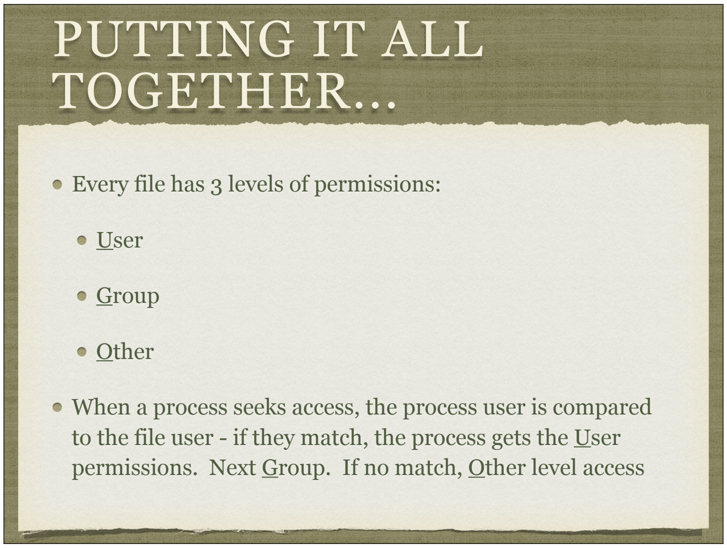
## AWESOME... SO?

- 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...



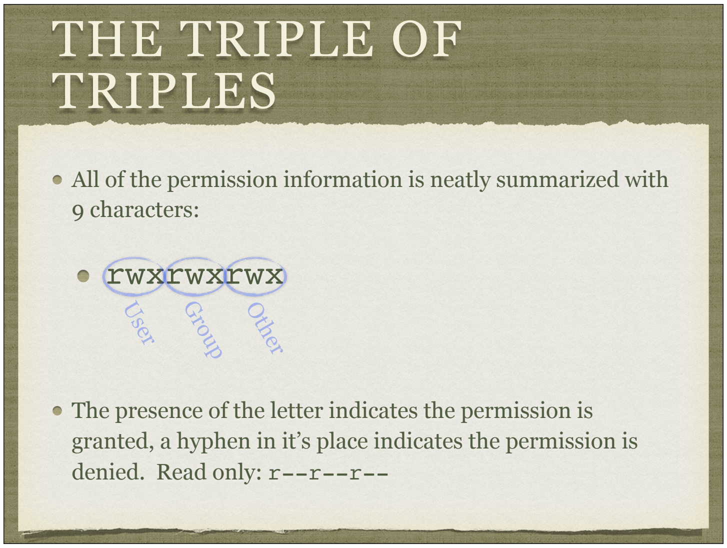
## OWNERSHIP

- 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

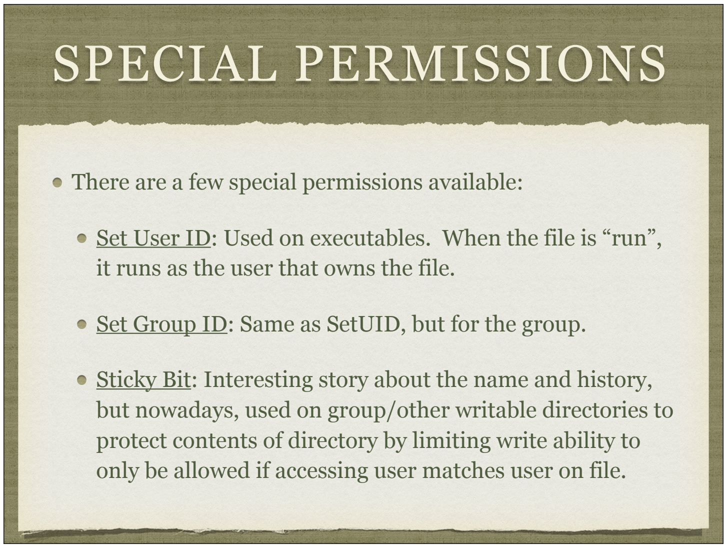


# THE TRIPLE OF TRIPLES

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

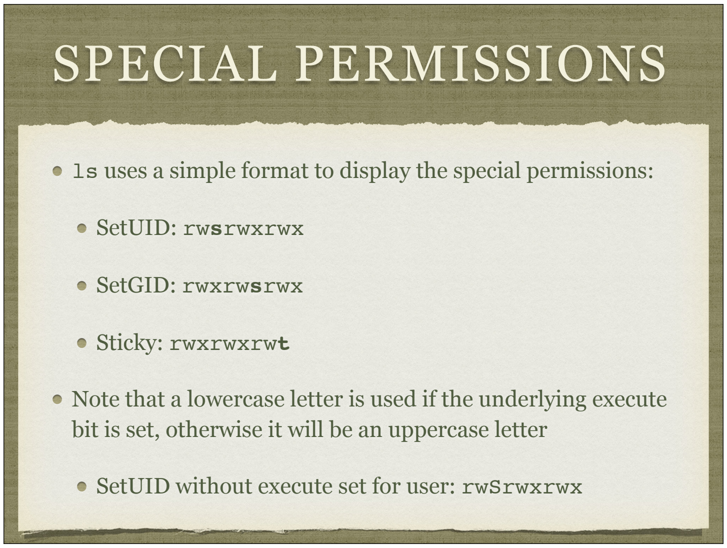
User
Group
Other

- 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--`



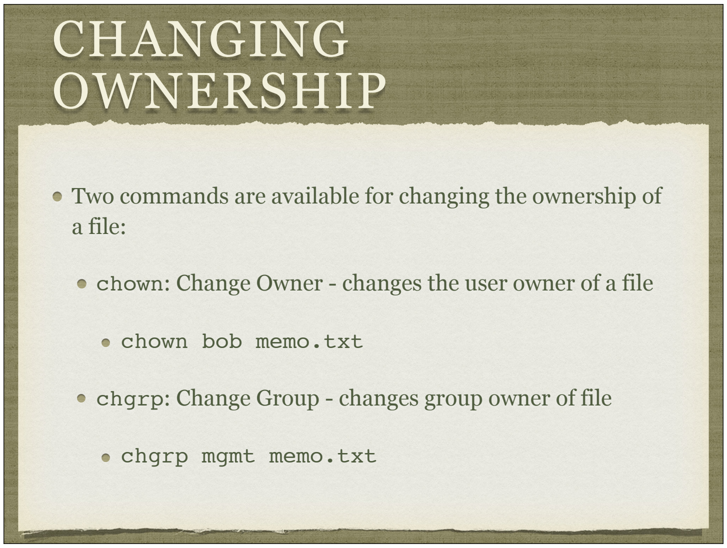
# SPECIAL PERMISSIONS

- There are a few special permissions available:
  - Set User ID: Used on executables. When the file is "run", it runs as the user that owns the file.
  - Set Group ID: Same as SetUID, but for the group.
  - Sticky Bit: Interesting story about the name and history, but nowadays, used on group/other writable directories to protect contents of directory by limiting write ability to only be allowed if accessing user matches user on file.



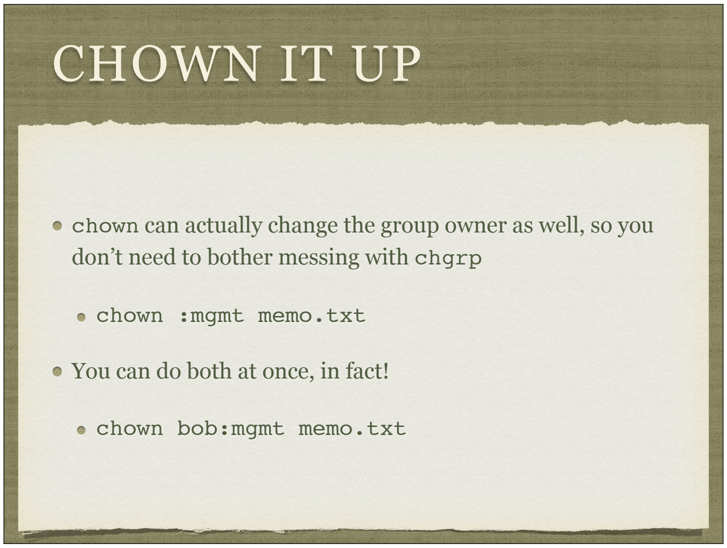
# SPECIAL PERMISSIONS

- `ls` uses a simple format to display the special permissions:
  - SetUID: rw**s**rwxrwx
  - SetGID: rwxrw**s**rwx
  - Sticky: rwxrwxrw**t**
- Note that a lowercase letter is used if the underlying execute bit is set, otherwise it will be an uppercase letter
  - SetUID without execute set for user: `rwSrwxrwx`



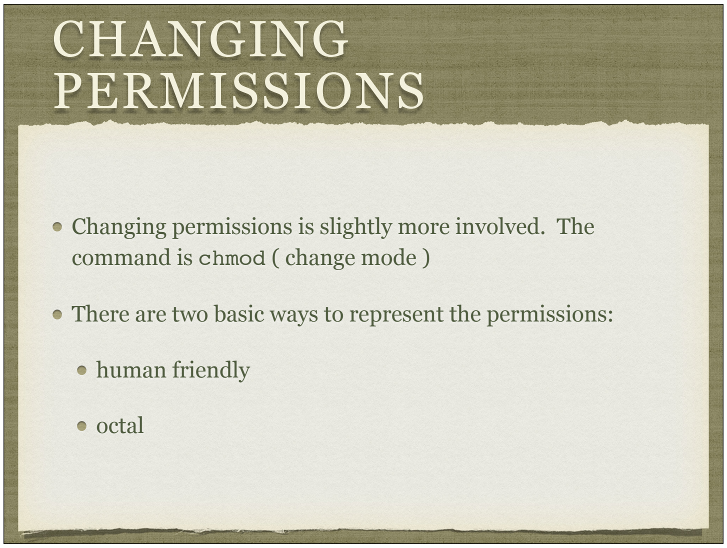
# CHANGING OWNERSHIP

- 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`



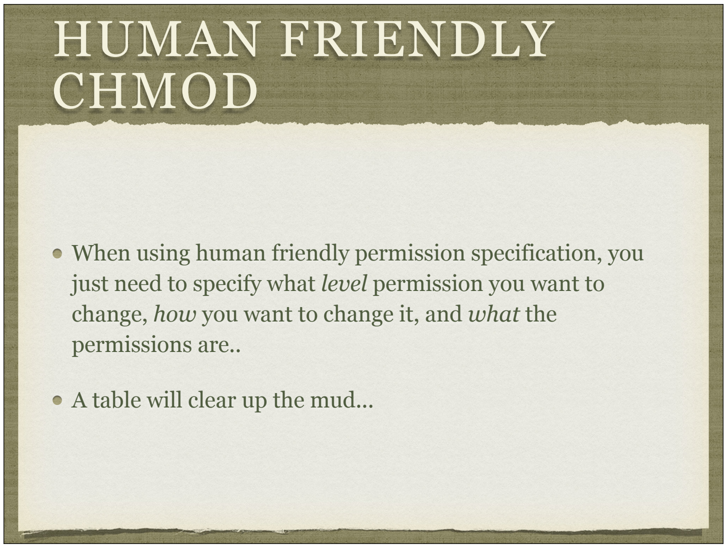
# CHOWN IT UP

- `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`



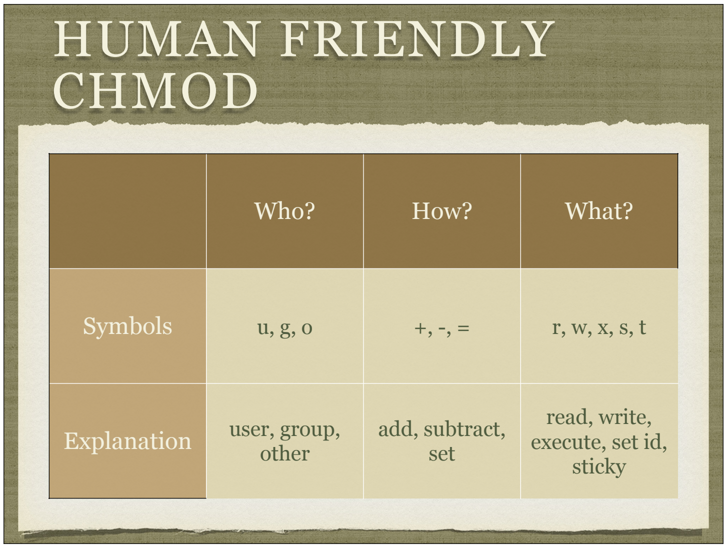
# CHANGING PERMISSIONS

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



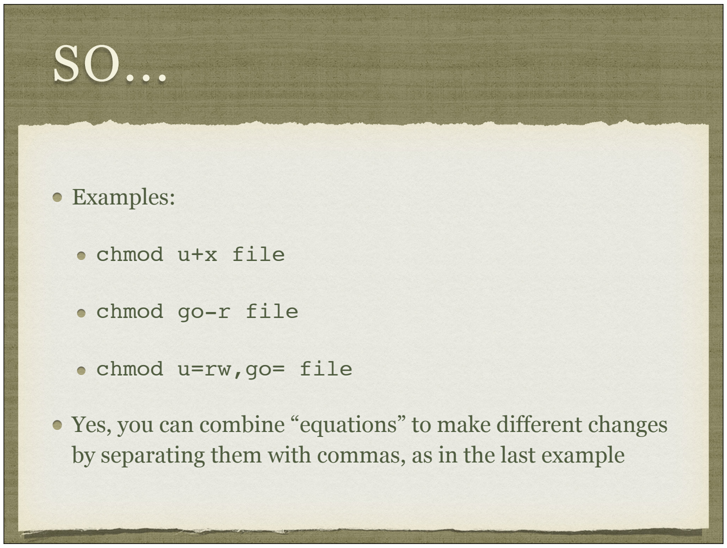
# HUMAN FRIENDLY CHMOD

- 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...



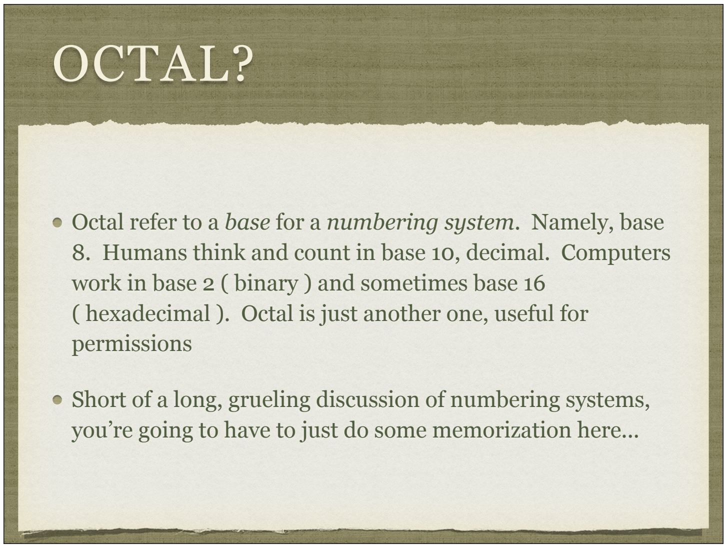
# HUMAN FRIENDLY CHMOD

| | 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 |



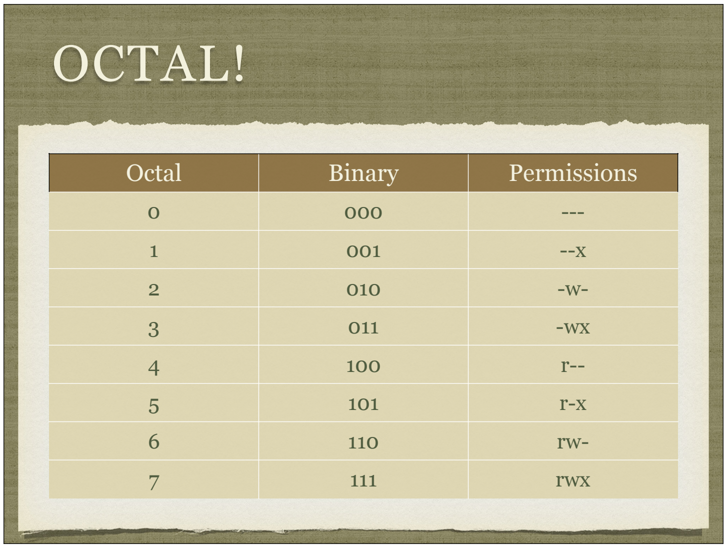
# SO...

- 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



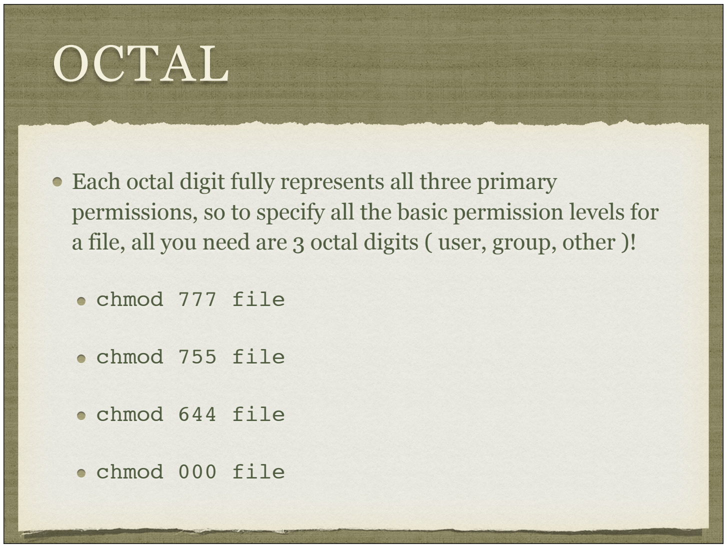
# OCTAL?

- 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...



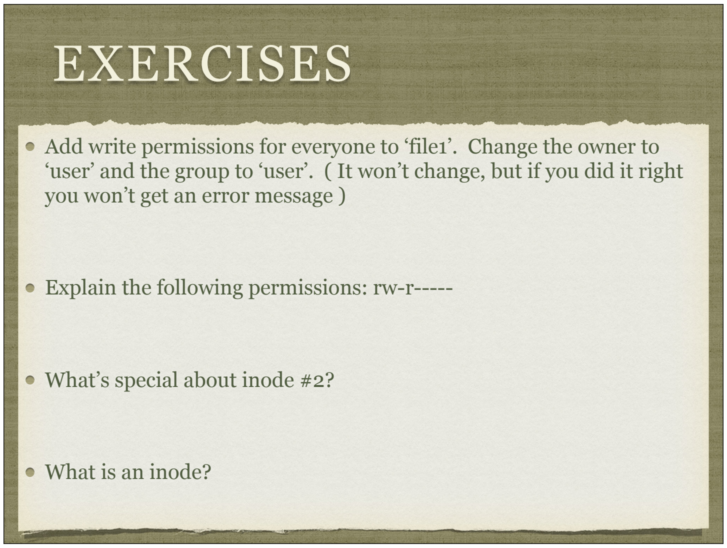
# OCTAL!

| 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 |



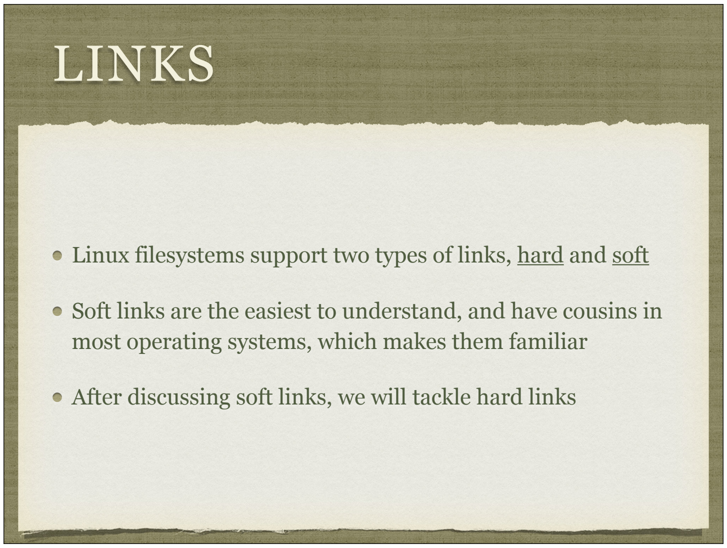
# OCTAL

- 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`



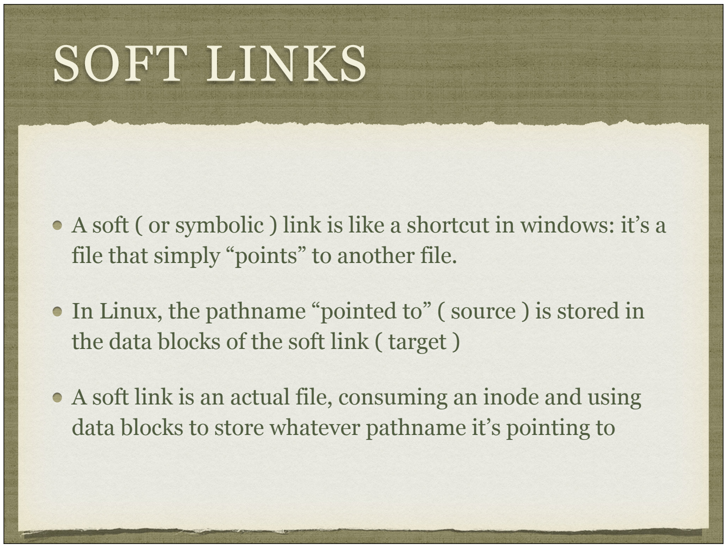
# EXERCISES

- 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-----
- What’s special about inode #2?
- What is an inode?



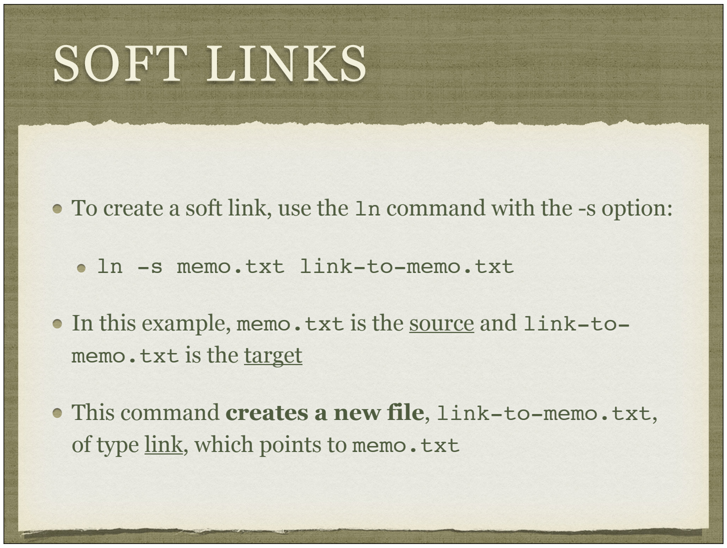
# LINKS

- 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
- After discussing soft links, we will tackle hard links



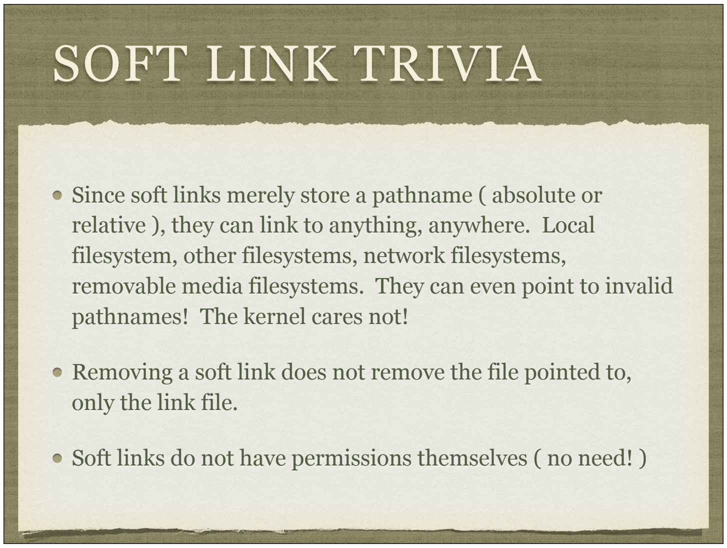
# SOFT LINKS

- 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



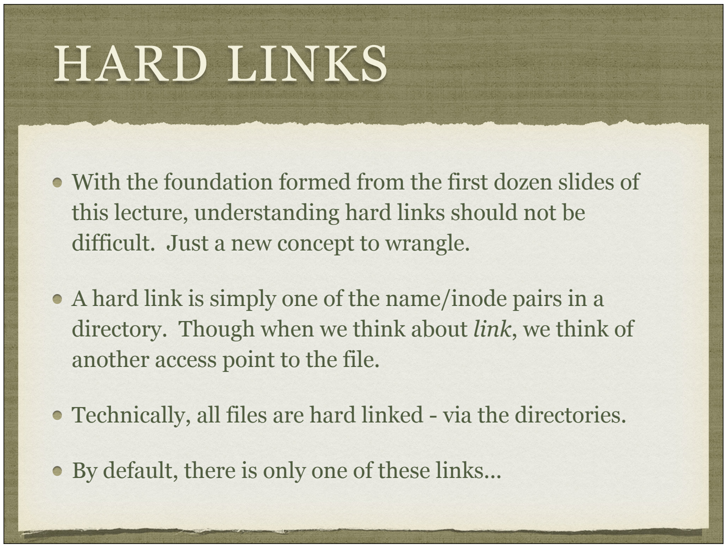
# SOFT LINKS

- 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`



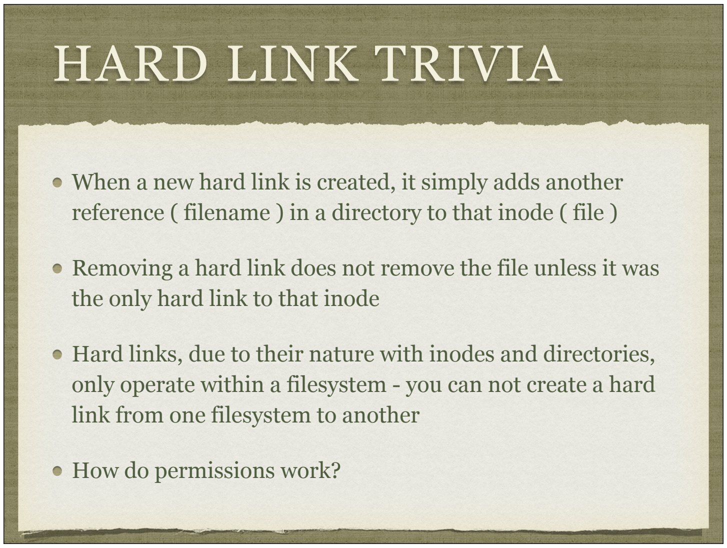
# SOFT LINK TRIVIA

- 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! )



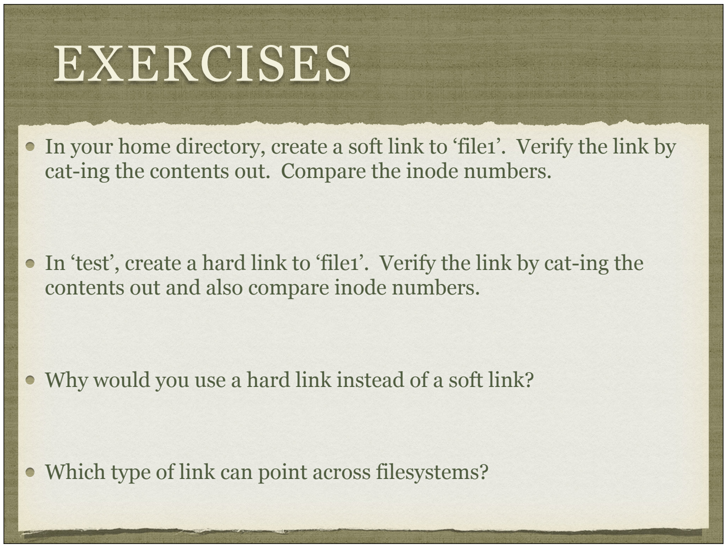
# HARD LINKS

- With the foundation formed from the first dozen slides of this lecture, understanding hard links should not be difficult. Just a new concept to wrangle.
- A hard link is simply one of the name/inode pairs in a directory. Though when we think about *link*, we think of another access point to the file.
- Technically, all files are hard linked - via the directories.
- By default, there is only one of these links...



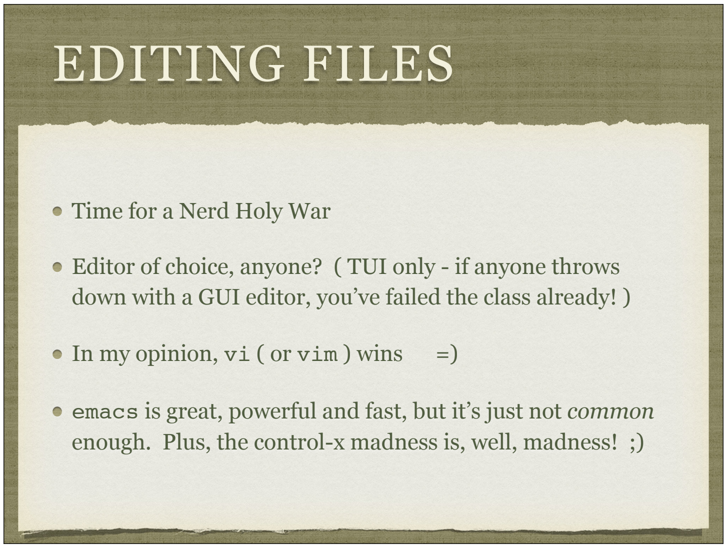
# HARD LINK TRIVIA

- When a new hard link is created, it simply adds another reference ( filename ) in a directory to that inode ( file )
- Removing a hard link does not remove the file unless it was the only hard link to that inode
- Hard links, due to their nature with inodes and directories, only operate within a filesystem - you can not create a hard link from one filesystem to another
- How do permissions work?



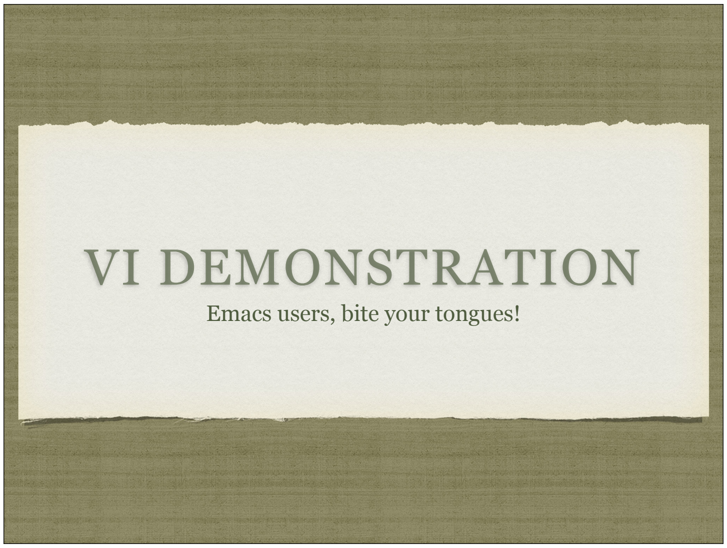
# EXERCISES

- In your home directory, create a soft link to 'file1'. Verify the link by cat-ing the contents out. Compare the inode numbers.
- In 'test', create a hard link to 'file1'. Verify the link by cat-ing the contents out and also compare inode numbers.
- Why would you use a hard link instead of a soft link?
- Which type of link can point across filesystems?



## 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!  ;)



## VI DEMONSTRATION

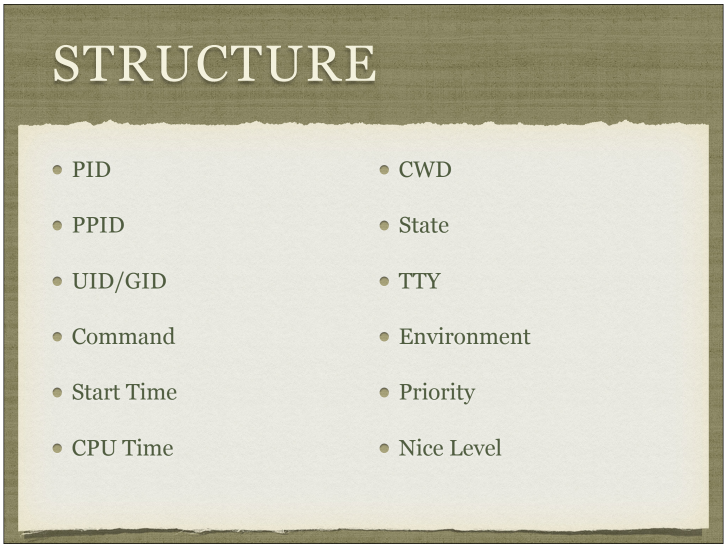
Emacs users, bite your tongues!

# PROCESSES

At least they're not ISO-9001 processes

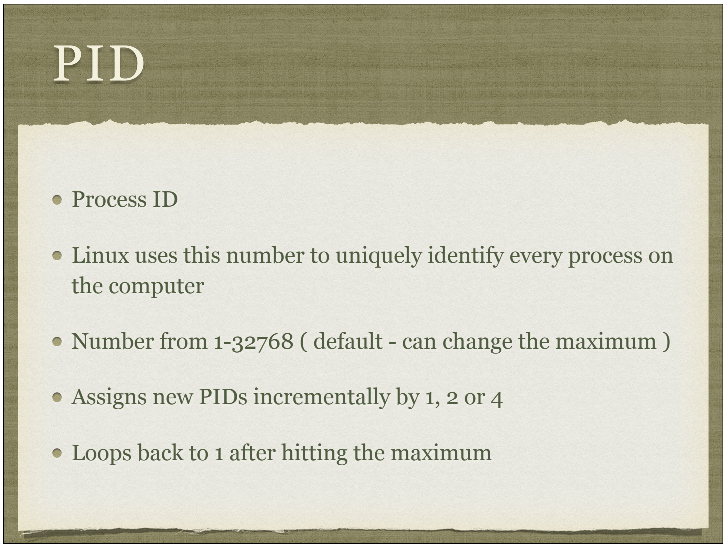
# STRUCTURE

- 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:



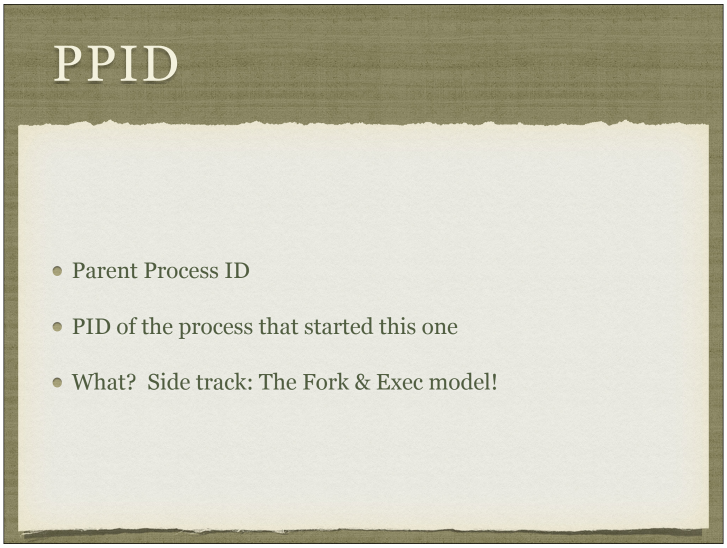
# STRUCTURE

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



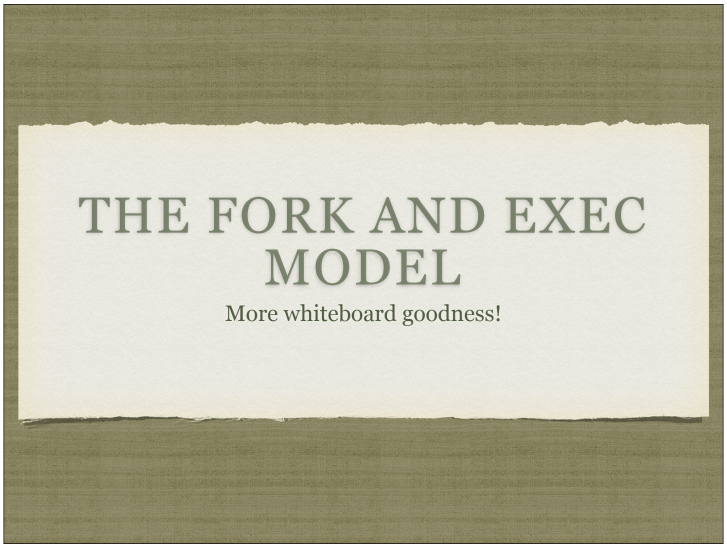
# PID

- 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



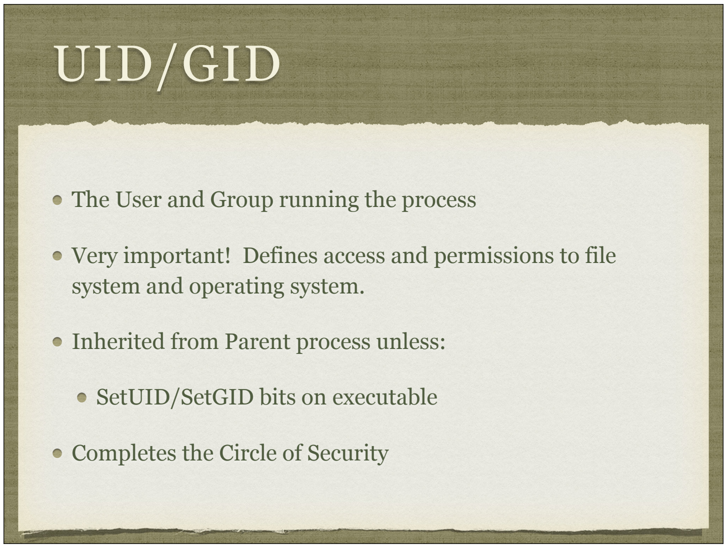
# PPID

- Parent Process ID
- PID of the process that started this one
- What?  Side track: The Fork & Exec model!



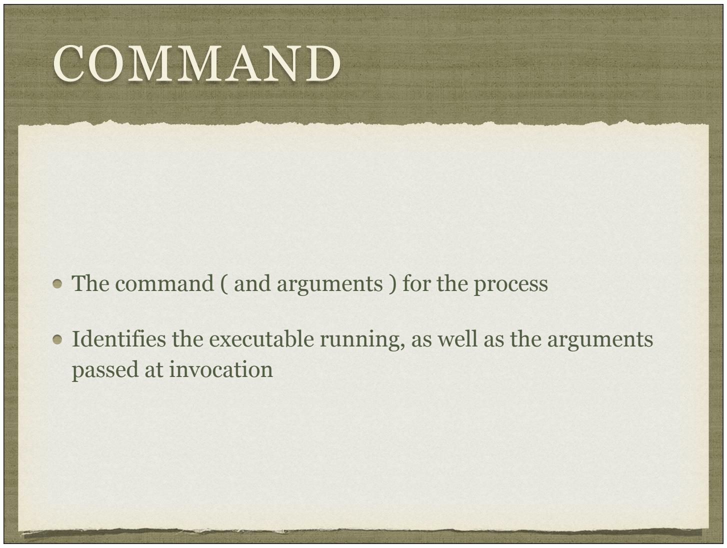
# THE FORK AND EXEC MODEL

More whiteboard goodness!



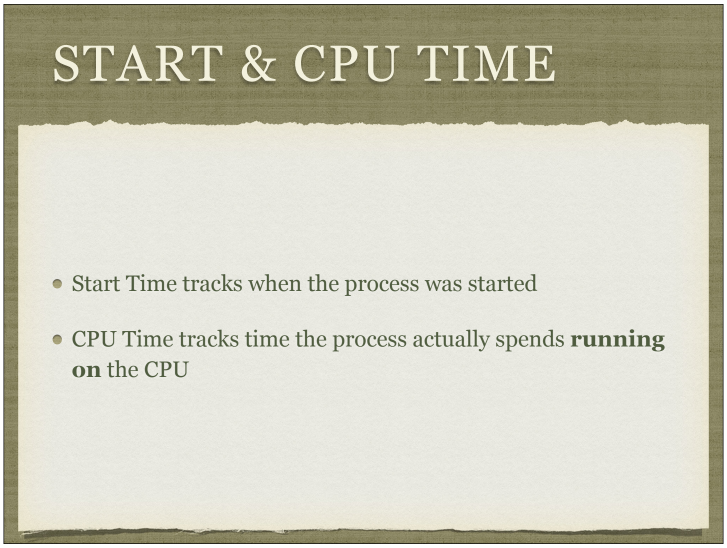
# UID/GID

- 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



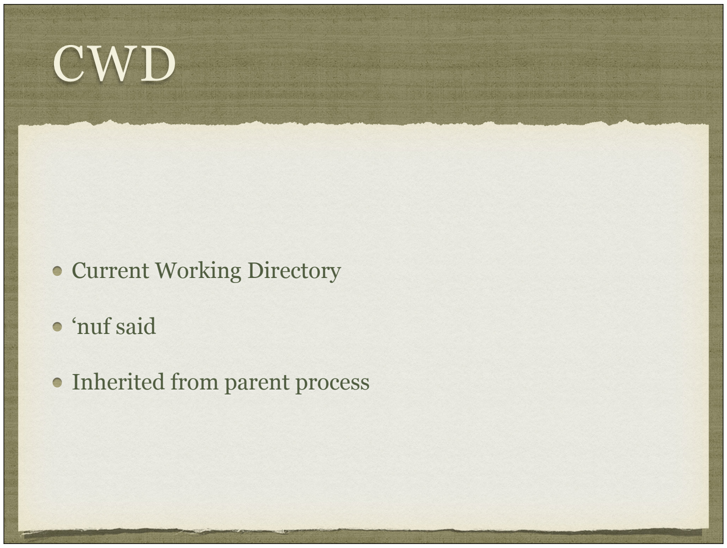
# COMMAND

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



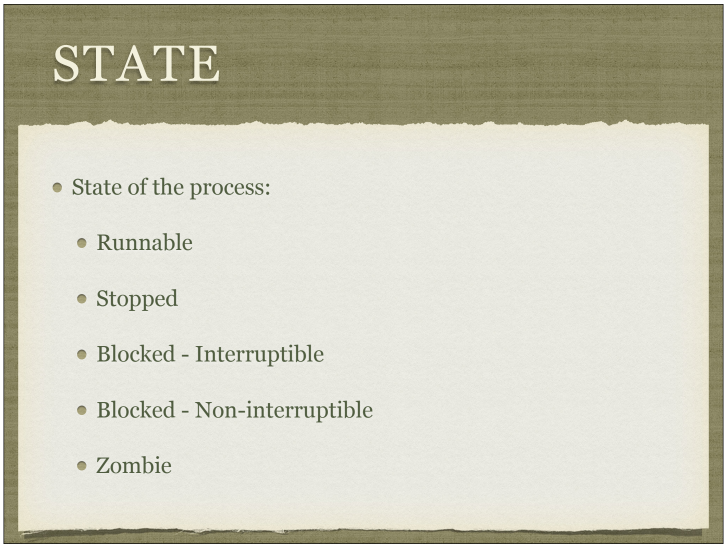
## START & CPU TIME

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



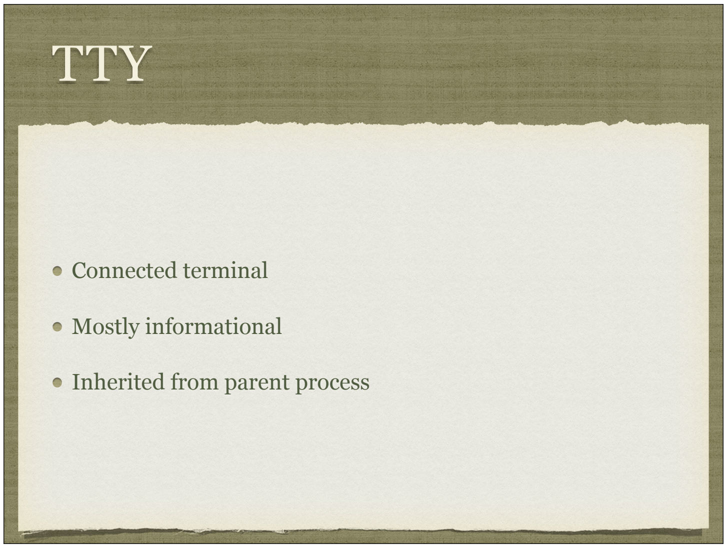
## CWD

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



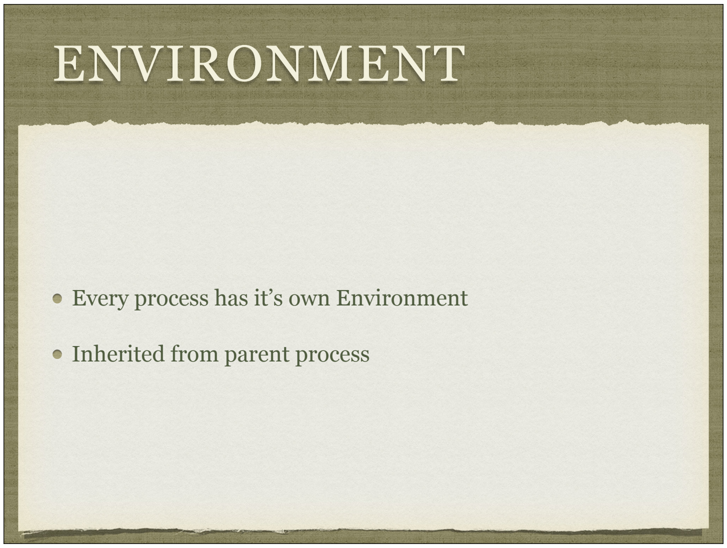
# STATE

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



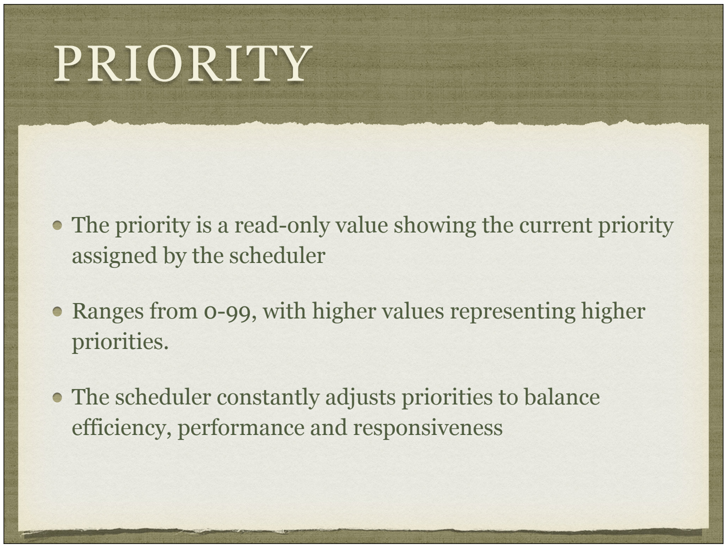
# TTY

- Connected terminal
- Mostly informational
- Inherited from parent process



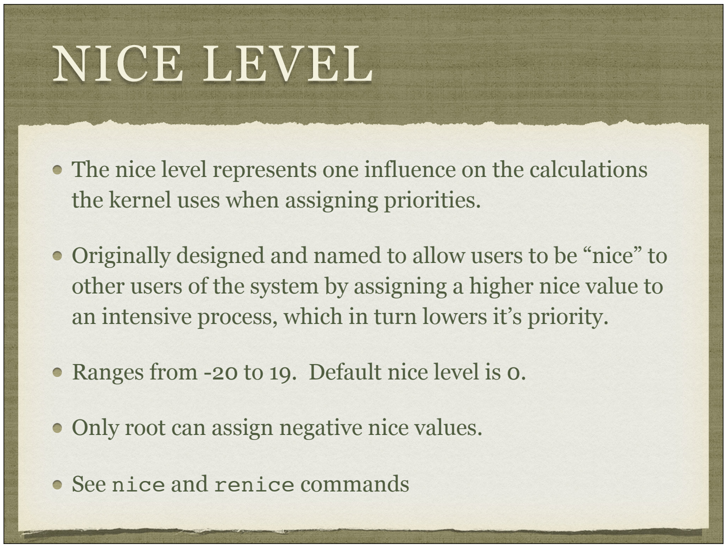
# ENVIRONMENT

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



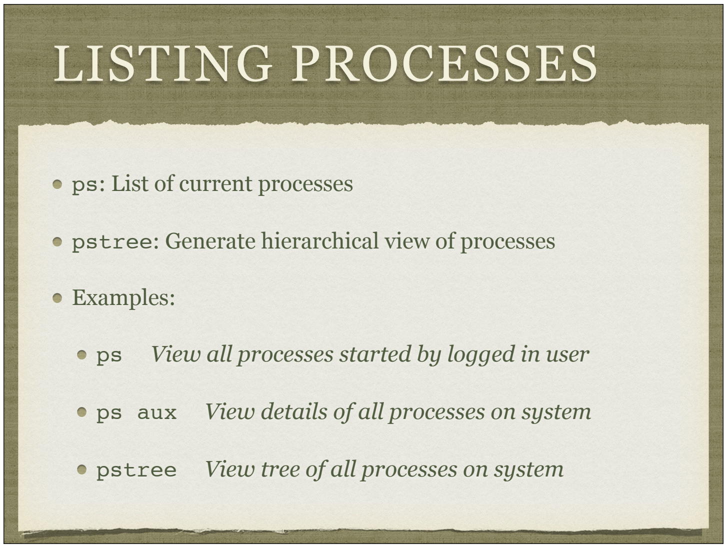
# PRIORITY

- 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



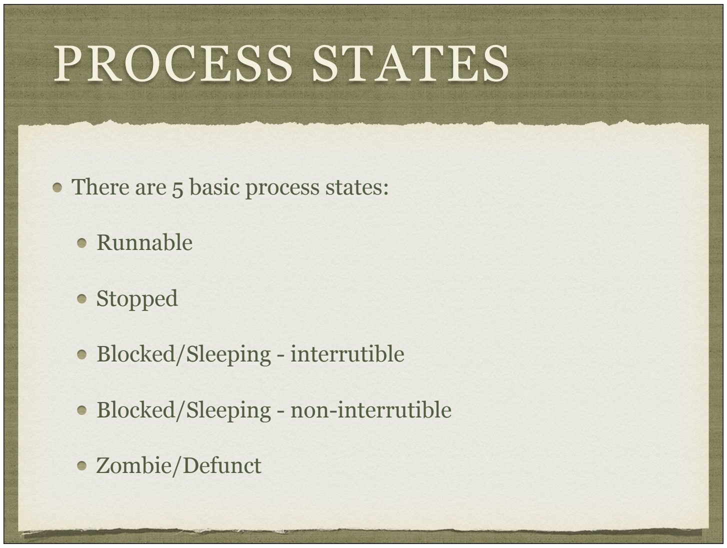
# NICE LEVEL

- 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



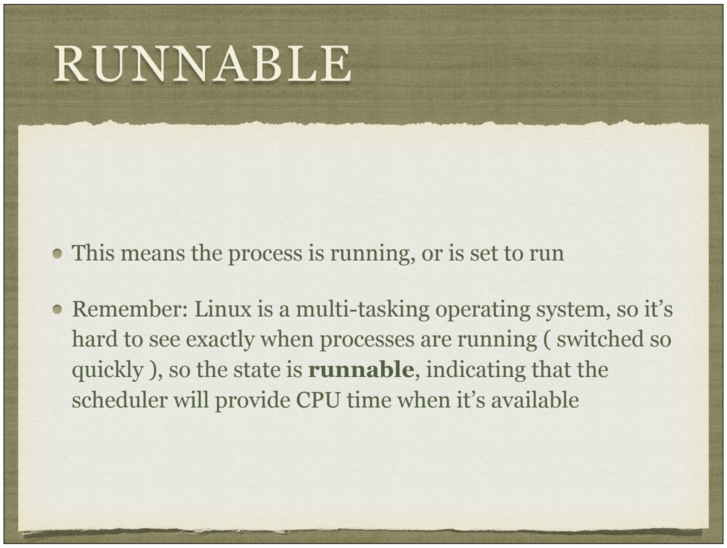
# LISTING PROCESSES

- `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*



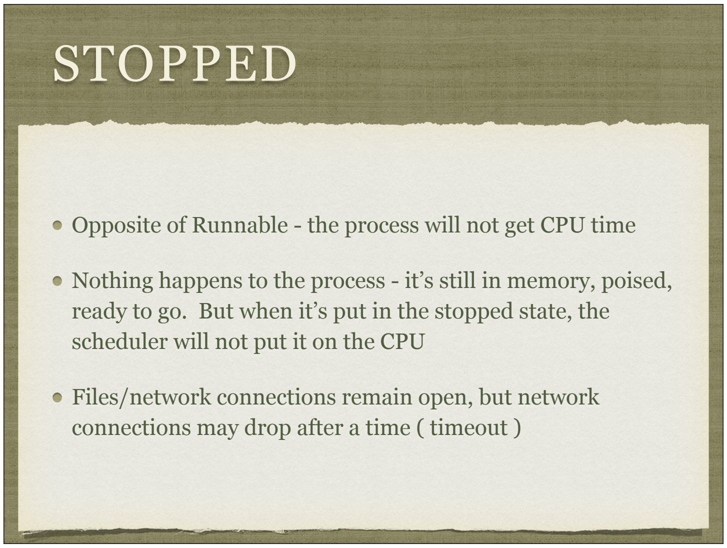
# PROCESS STATES

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



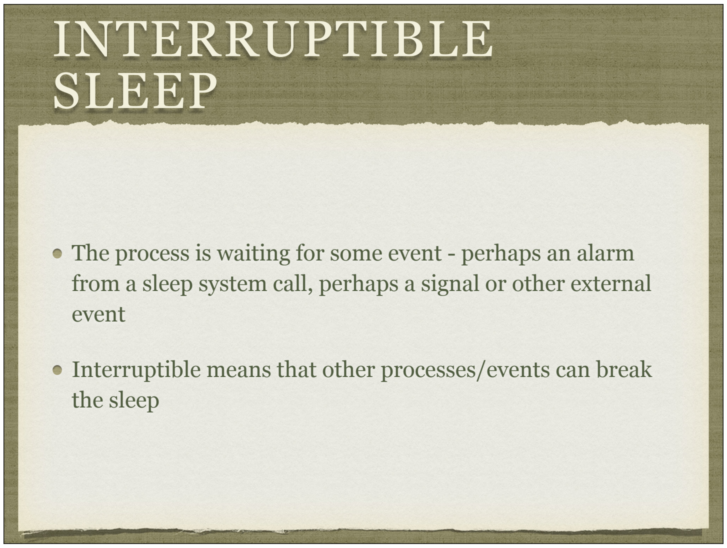
# RUNNABLE

- 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



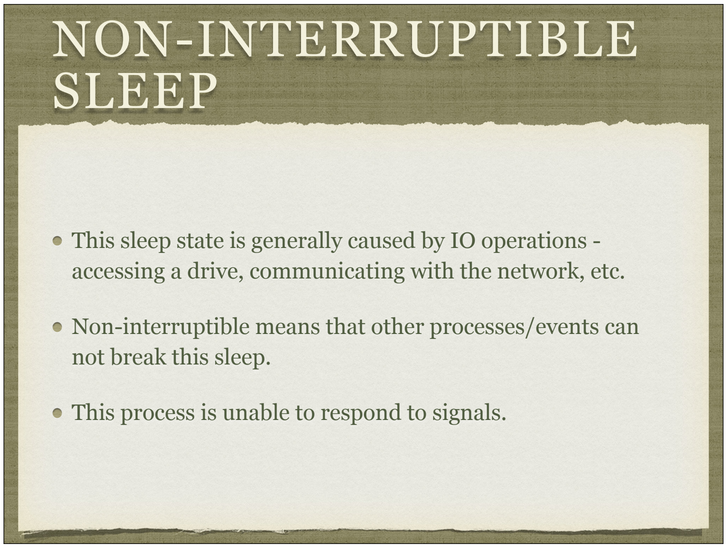
# STOPPED

- 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 )



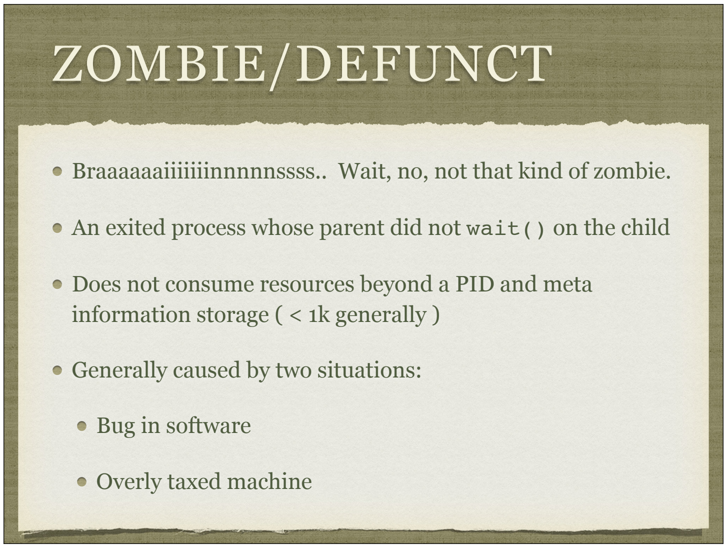
# INTERRUPTIBLE SLEEP

- 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



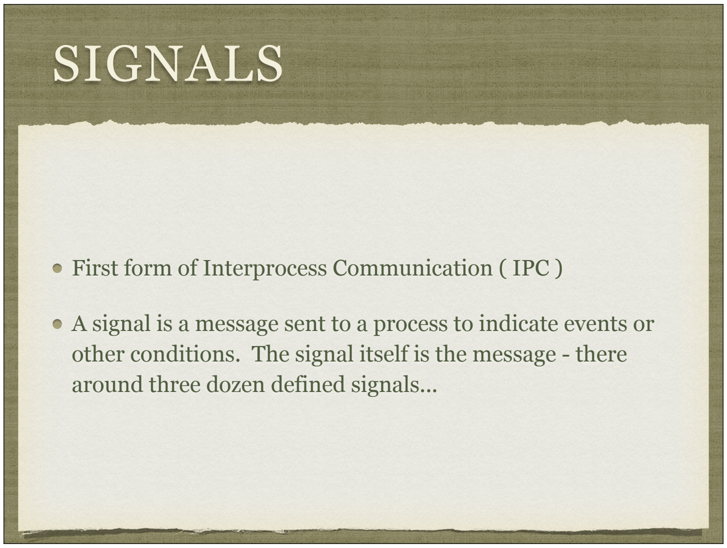
# NON-INTERRUPTIBLE SLEEP

- 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.



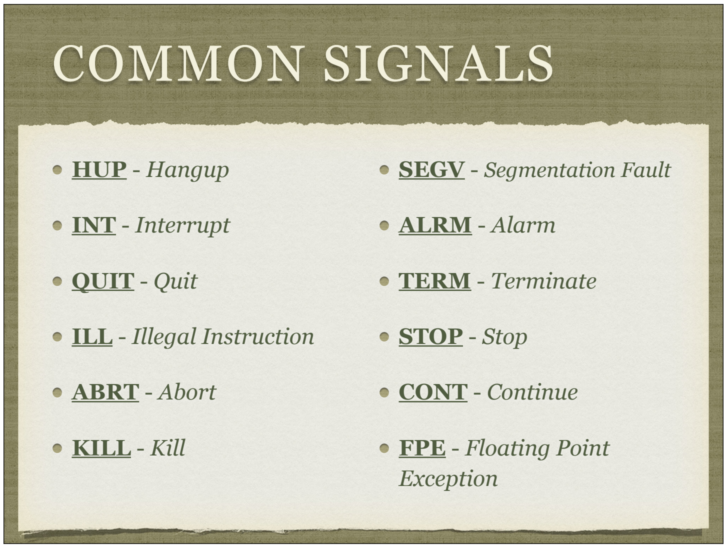
# ZOMBIE/DEFUNCT

- 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



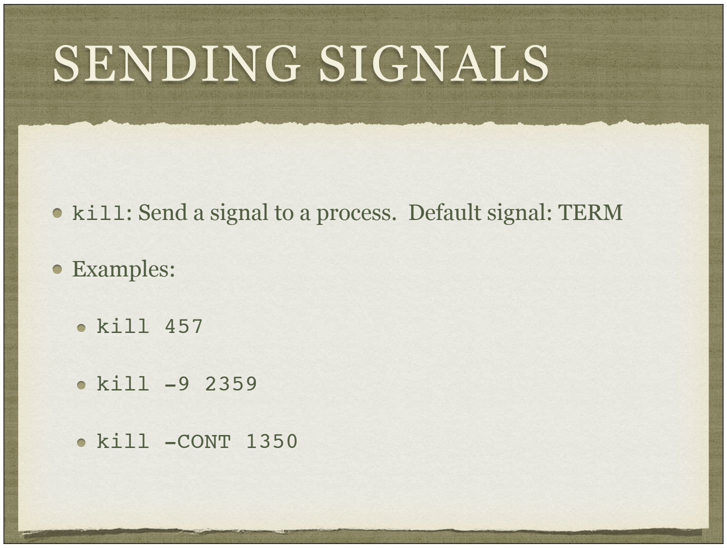
# SIGNALS

- 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...



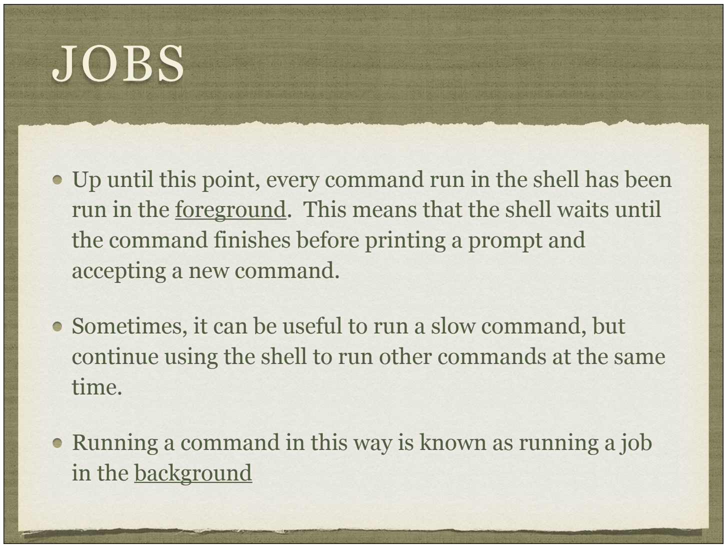
# COMMON SIGNALS

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



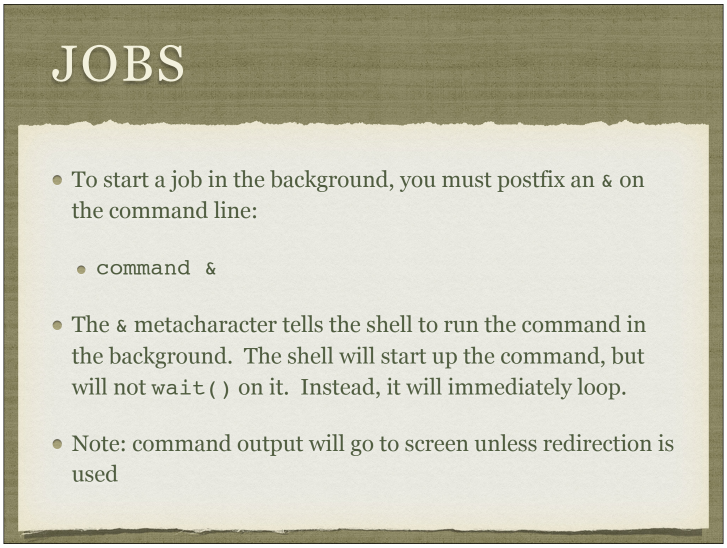
## SENDING SIGNALS

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



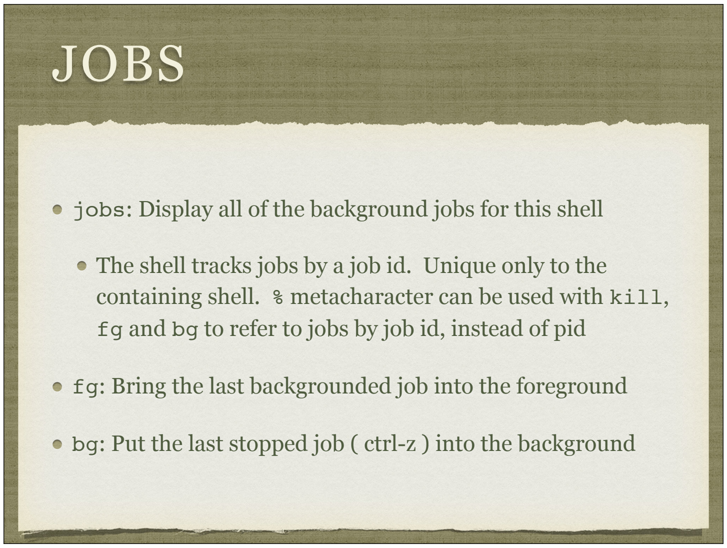
## JOBS

- Up until this point, every command run in the shell has been run in the foreground.  This means that the shell waits until the command finishes before printing a prompt and accepting a new command.
- Sometimes, it can be useful to run a slow command, but continue using the shell to run other commands at the same time.
- Running a command in this way is known as running a job in the background



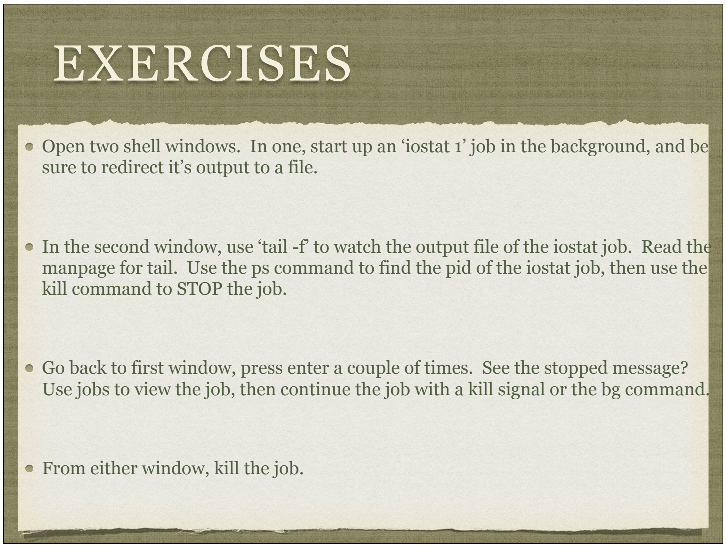
# JOBS

- To start a job in the background, you must postfix an `&` on the command line:
  - `command &`
- The `&` metacharacter tells the shell to run the command in the background. The shell will start up the command, but will not `wait()` on it. Instead, it will immediately loop.
- Note: command output will go to screen unless redirection is used



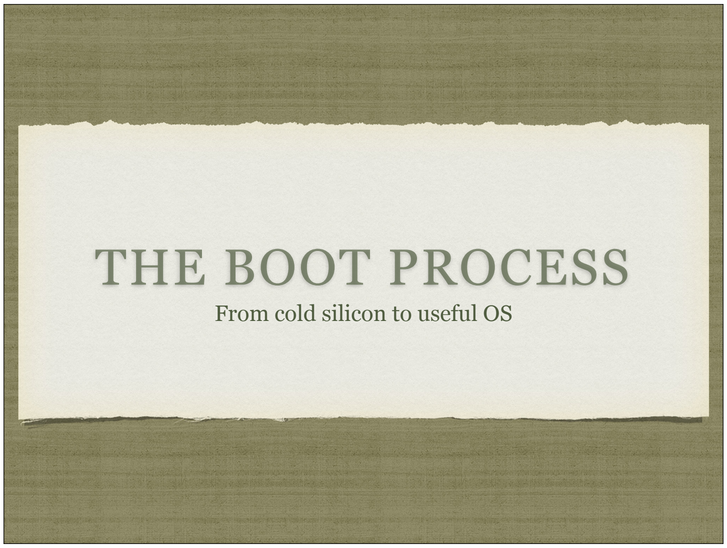
# JOBS

- `jobs`: Display all of the background jobs for this shell
  - The shell tracks jobs by a job id. Unique only to the containing shell. `%` metacharacter can be used with `kill`, `fg` and `bg` to refer to jobs by job id, instead of pid
- `fg`: Bring the last backgrounded job into the foreground
- `bg`: Put the last stopped job ( ctrl-z ) into the background



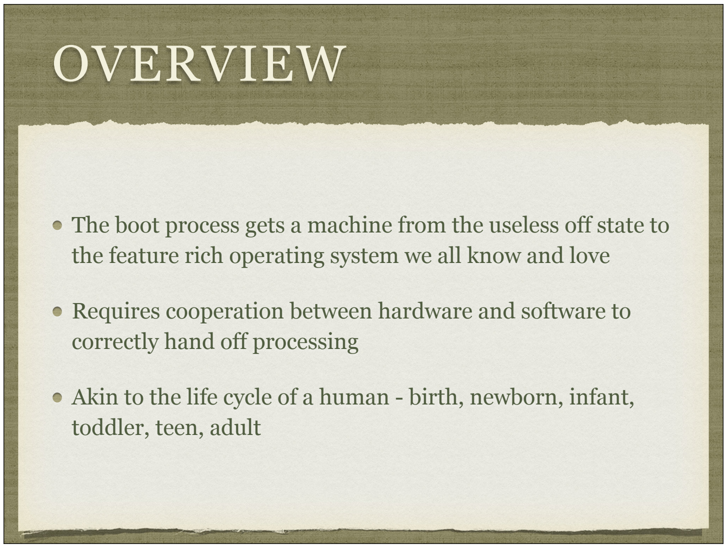
# EXERCISES

- Open two shell windows.  In one, start up an 'iostat 1' job in the background, and be sure to redirect it's output to a file.
- In the second window, use 'tail -f' to watch the output file of the iostat job.  Read the manpage for tail.  Use the ps command to find the pid of the iostat job, then use the kill command to STOP the job.
- Go back to first window, press enter a couple of times.  See the stopped message? Use jobs to view the job, then continue the job with a kill signal or the bg command.
- From either window, kill the job.



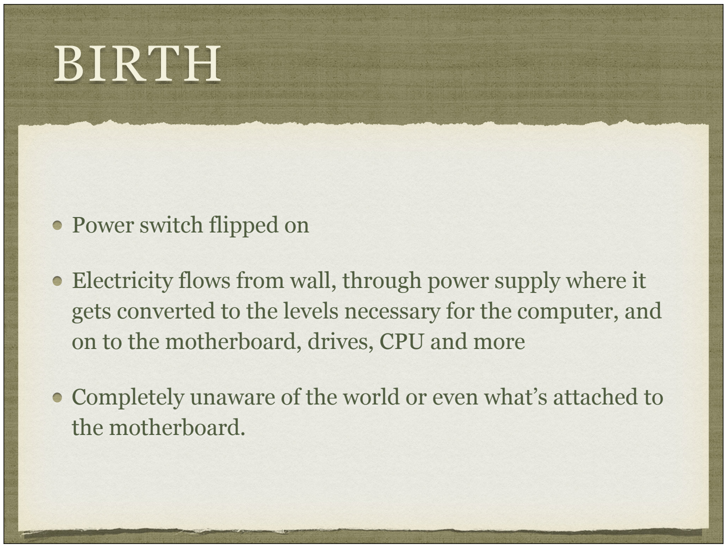
# THE BOOT PROCESS

From cold silicon to useful OS



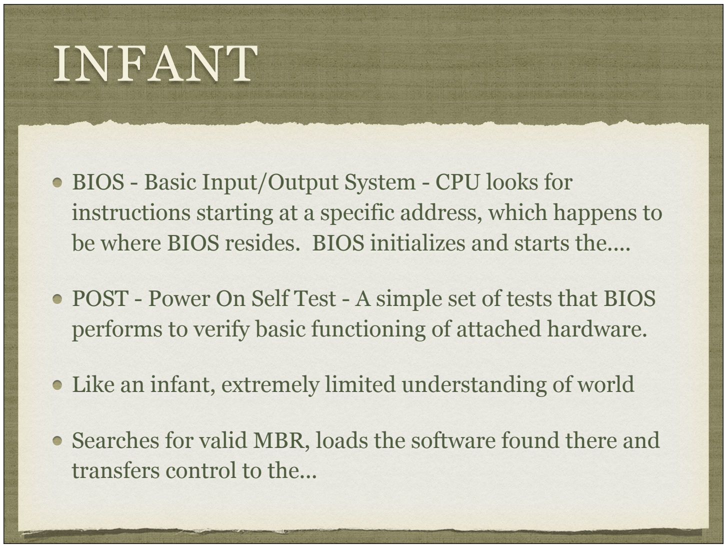
## OVERVIEW

- The boot process gets a machine from the useless off state to the feature rich operating system we all know and love
- Requires cooperation between hardware and software to correctly hand off processing
- Akin to the life cycle of a human - birth, newborn, infant, toddler, teen, adult



## BIRTH

- Power switch flipped on
- Electricity flows from wall, through power supply where it gets converted to the levels necessary for the computer, and on to the motherboard, drives, CPU and more
- Completely unaware of the world or even what’s attached to the motherboard.



## INFANT

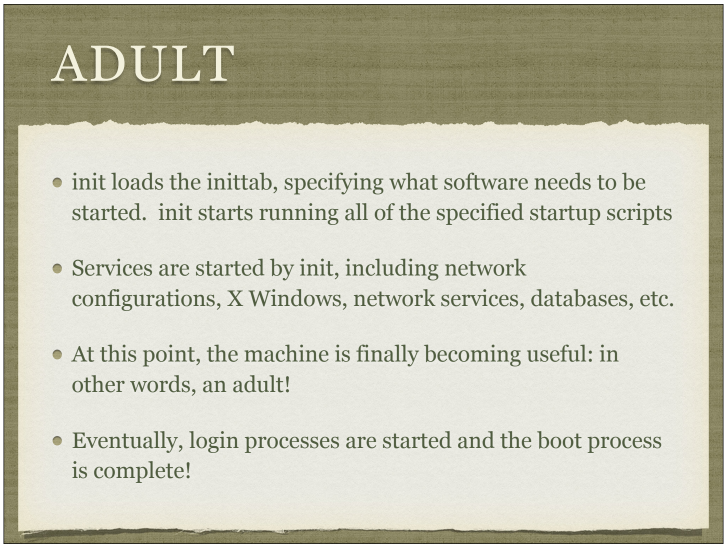
- BIOS - Basic Input/Output System - CPU looks for instructions starting at a specific address, which happens to be where BIOS resides.  BIOS initializes and starts the....
- POST - Power On Self Test - A simple set of tests that BIOS performs to verify basic functioning of attached hardware.
- Like an infant, extremely limited understanding of world
- Searches for valid MBR, loads the software found there and transfers control to the...

# TODDLER

- Boot Loader - Special software installed to the MBR of the boot partition which selects and loads the kernel.
- Can be configured to immediately load the default OS, or can offer choice to user
- Slightly better understanding of world - can read linux filesystems, sometimes includes powerful debugging and configuration support.
- Main job: select and load kernel, transfer control to kernel

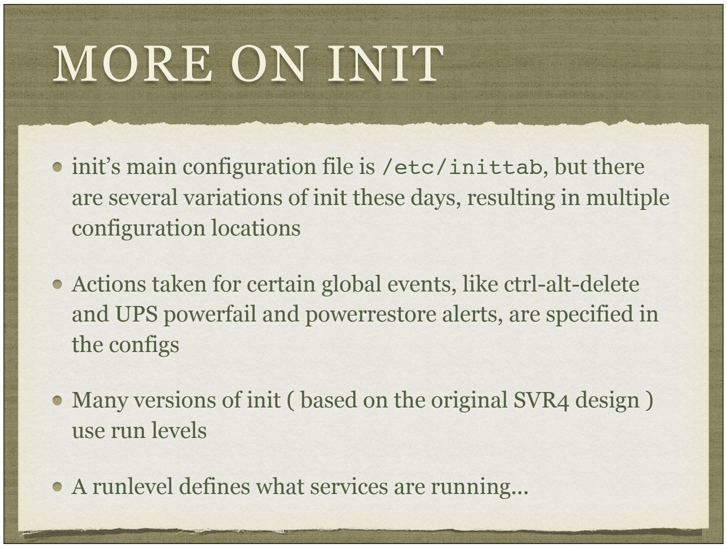
# TEENAGER

- Dreaded teenager age: knows a lot about the world, but doesn't contribute a thing.  Still pretty useless.
- Kernel loads and initializes.  Device drivers are loaded and initialized.  Basic hardware checks performed.
- The First Process is *created from nothing*: `init`



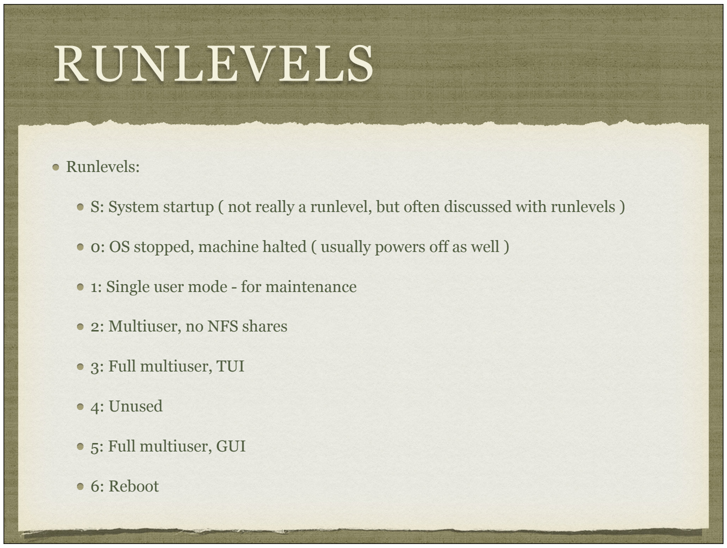
# ADULT

- init loads the inittab, specifying what software needs to be started. init starts running all of the specified startup scripts
- Services are started by init, including network configurations, X Windows, network services, databases, etc.
- At this point, the machine is finally becoming useful: in other words, an adult!
- Eventually, login processes are started and the boot process is complete!



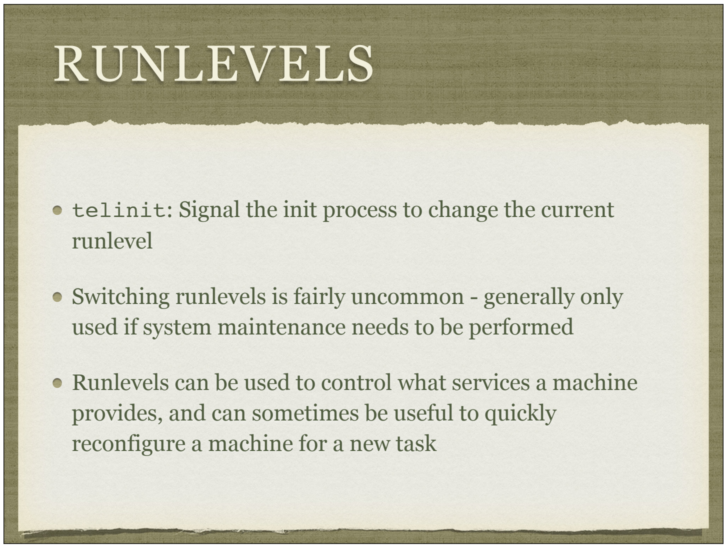
# MORE ON INIT

- init's main configuration file is `/etc/inittab`, but there are several variations of init these days, resulting in multiple configuration locations
- Actions taken for certain global events, like ctrl-alt-delete and UPS powerfail and powerrestore alerts, are specified in the configs
- Many versions of init ( based on the original SVR4 design ) use run levels
- A runlevel defines what services are running...



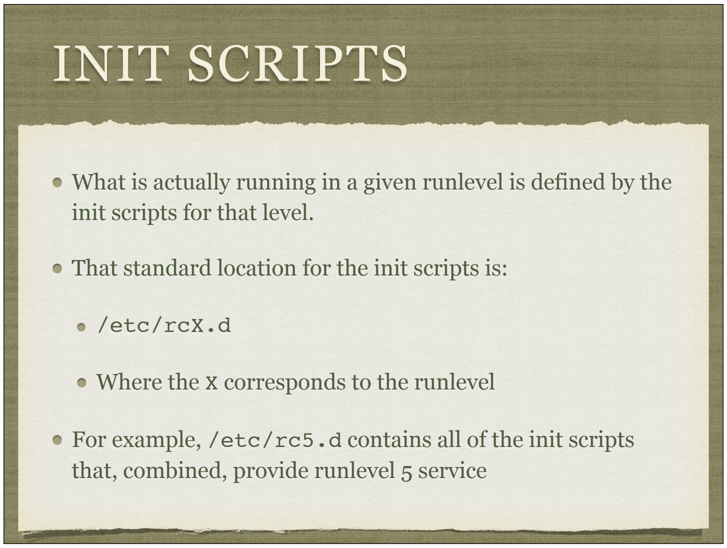
# RUNLEVELS

- Runlevels:
  - S: System startup ( not really a runlevel, but often discussed with runlevels )
  - 0: OS stopped, machine halted ( usually powers off as well )
  - 1: Single user mode - for maintenance
  - 2: Multiuser, no NFS shares
  - 3: Full multiuser, TUI
  - 4: Unused
  - 5: Full multiuser, GUI
  - 6: Reboot



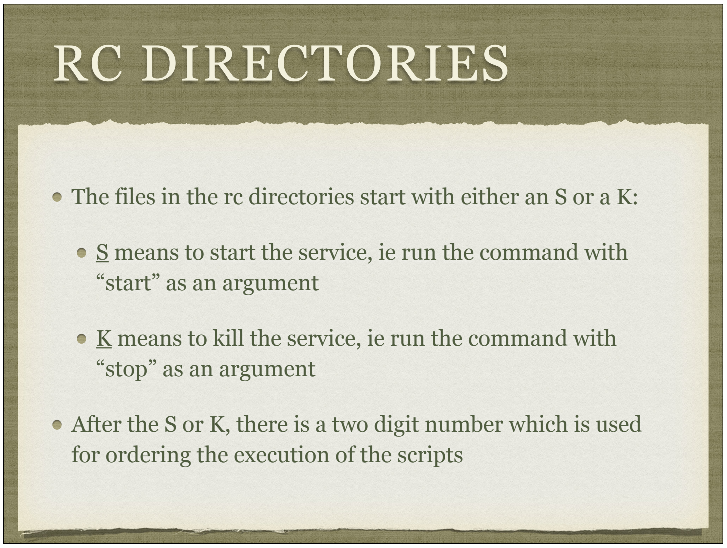
# RUNLEVELS

- `telinit`: Signal the init process to change the current runlevel
- Switching runlevels is fairly uncommon - generally only used if system maintenance needs to be performed
- Runlevels can be used to control what services a machine provides, and can sometimes be useful to quickly reconfigure a machine for a new task



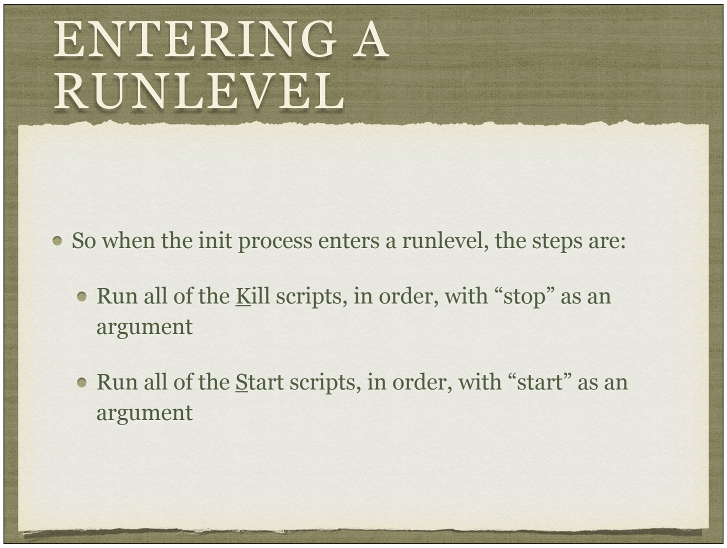
# INIT SCRIPTS

- What is actually running in a given runlevel is defined by the init scripts for that level.
- That standard location for the init scripts is:
  - `/etc/rcX.d`
  - Where the `X` corresponds to the runlevel
- For example, `/etc/rc5.d` contains all of the init scripts that, combined, provide runlevel 5 service



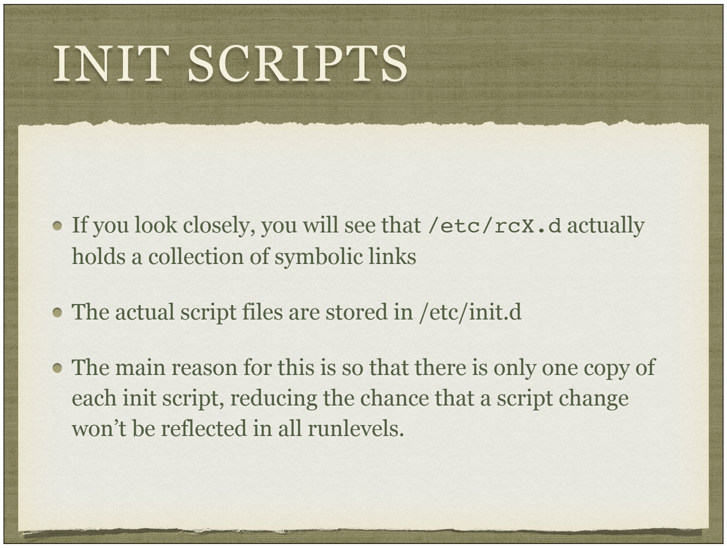
# RC DIRECTORIES

- The files in the rc directories start with either an S or a K:
  - S means to start the service, ie run the command with "start" as an argument
  - K means to kill the service, ie run the command with "stop" as an argument
- After the S or K, there is a two digit number which is used for ordering the execution of the scripts



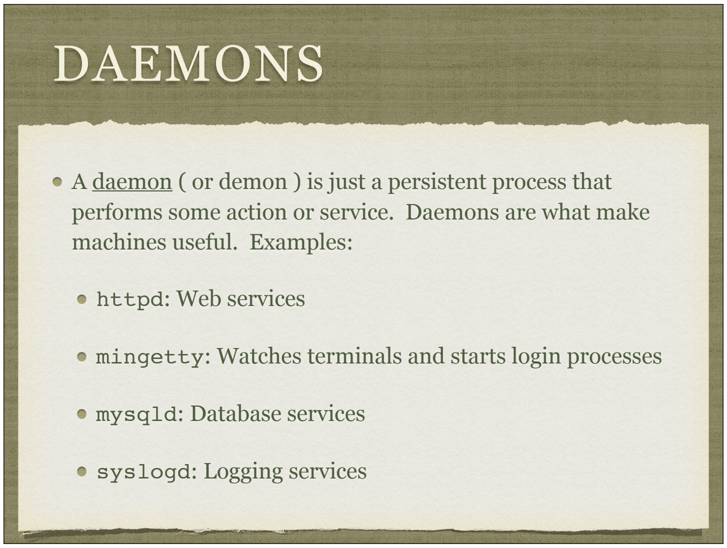
# ENTERING A RUNLEVEL

- So when the init process enters a runlevel, the steps are:
  - Run all of the Kill scripts, in order, with “stop” as an argument
  - Run all of the Start scripts, in order, with “start” as an argument



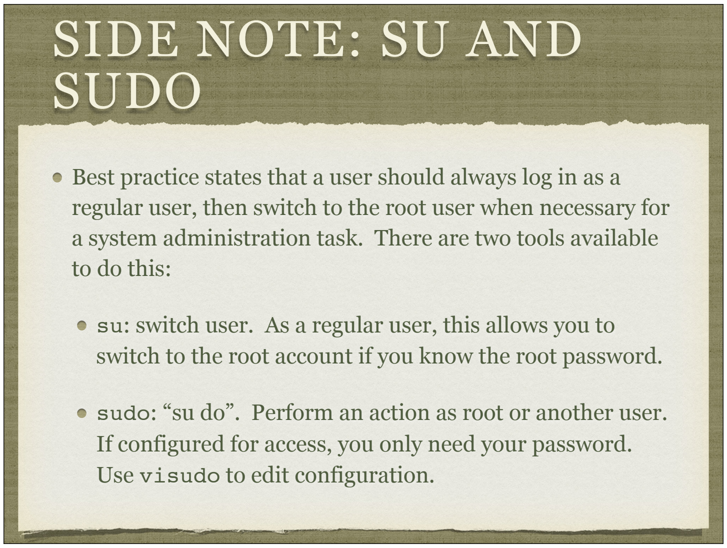
# INIT SCRIPTS

- If you look closely, you will see that `/etc/rcX.d` actually holds a collection of symbolic links
- The actual script files are stored in /etc/init.d
- The main reason for this is so that there is only one copy of each init script, reducing the chance that a script change won’t be reflected in all runlevels.



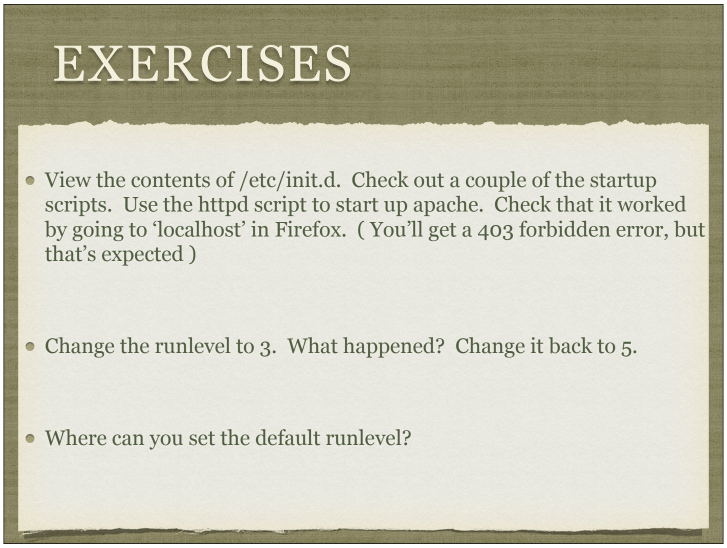
# DAEMONS

- A daemon ( or demon ) is just a persistent process that performs some action or service. Daemons are what make machines useful. Examples:
  - `httpd`: Web services
  - `mingetty`: Watches terminals and starts login processes
  - `mysqld`: Database services
  - `syslogd`: Logging services



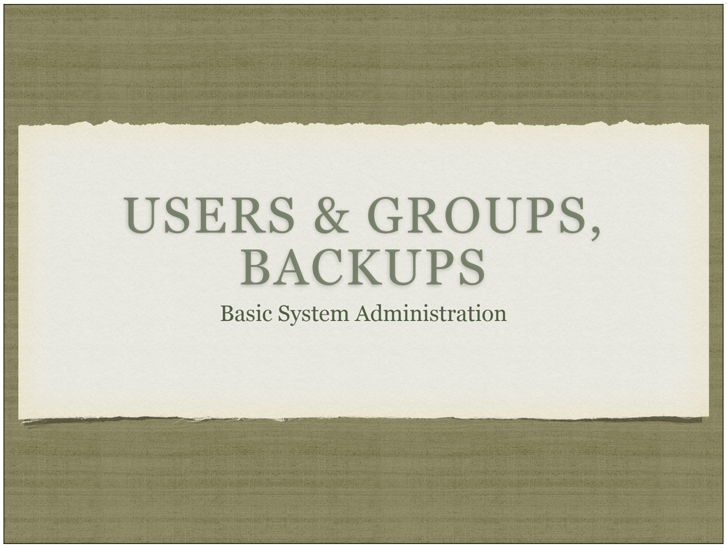
# SIDE NOTE: SU AND SUDO

- 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.



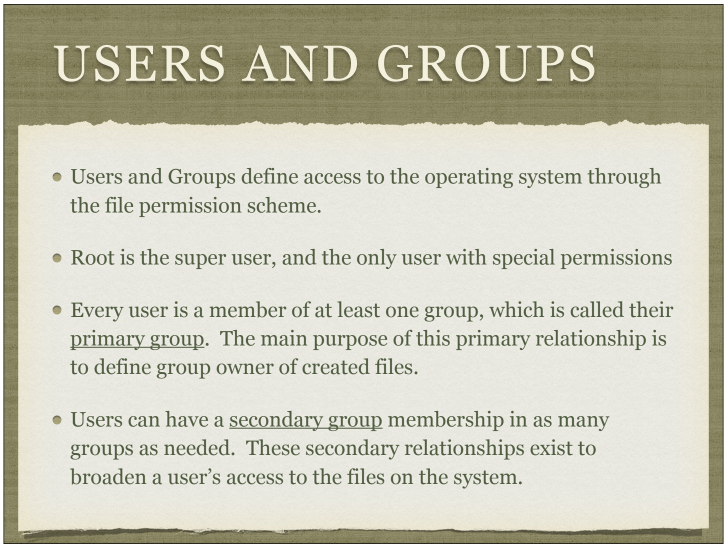
# EXERCISES

- View the contents of /etc/init.d.  Check out a couple of the startup scripts.  Use the httpd script to start up apache.  Check that it worked by going to 'localhost' in Firefox.  ( You'll get a 403 forbidden error, but that's expected )
- Change the runlevel to 3.  What happened?  Change it back to 5.
- Where can you set the default runlevel?



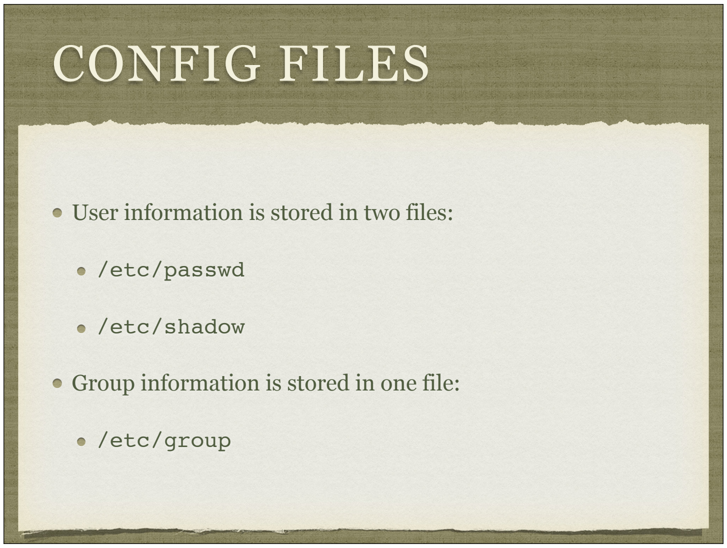
# USERS & GROUPS, BACKUPS

Basic System Administration



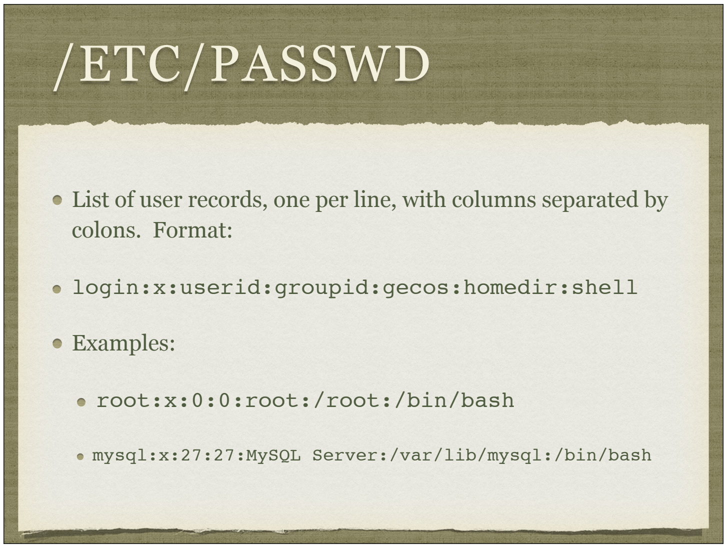
## USERS AND GROUPS

- 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.



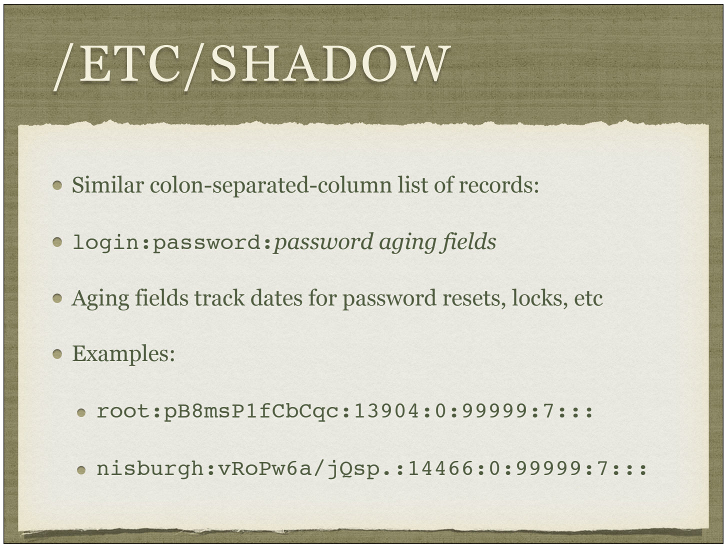
# CONFIG FILES

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



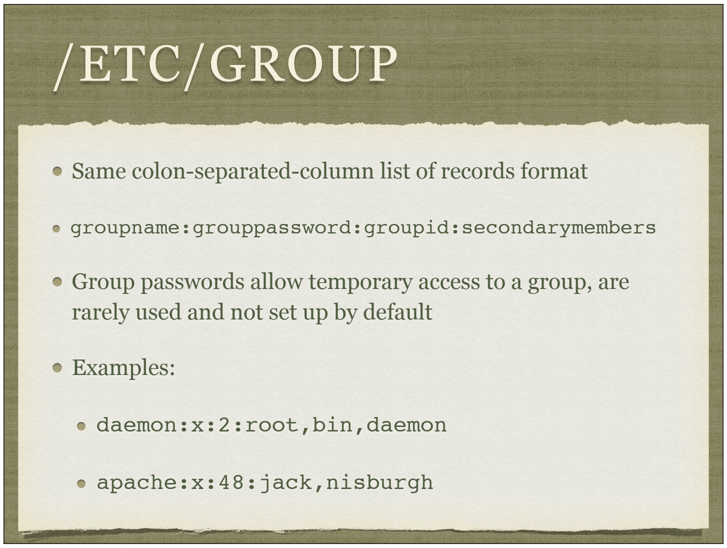
# /ETC/PASSWD

- 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`



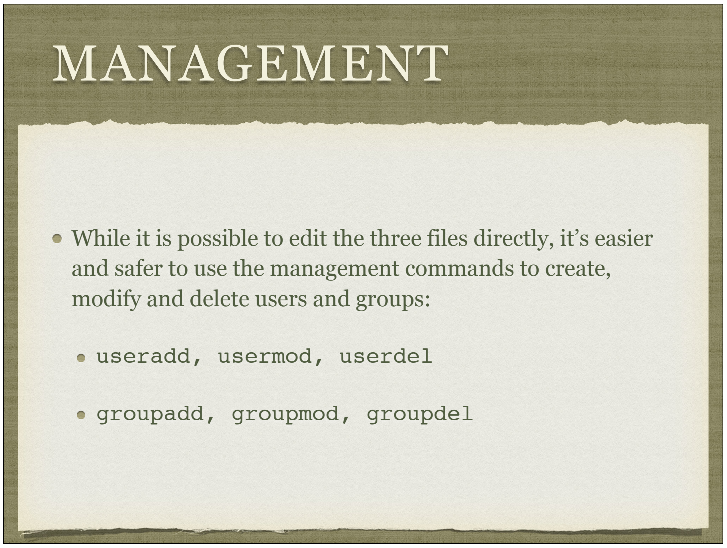
# /ETC/SHADOW

- 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:::`



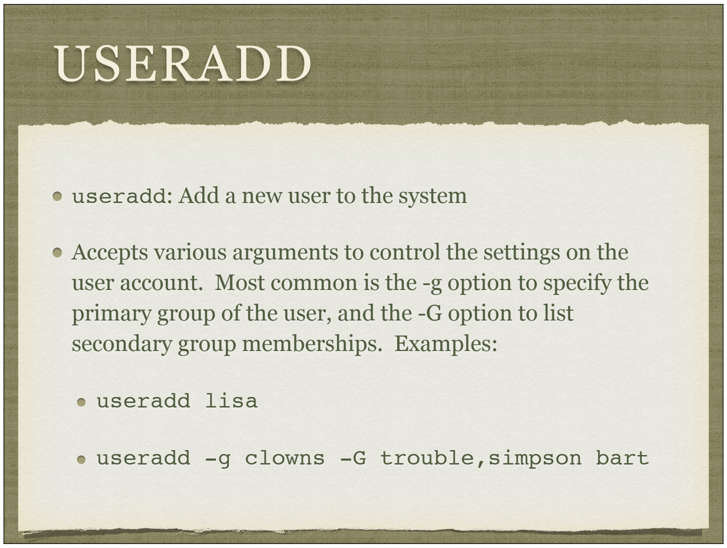
# /ETC/GROUP

- 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`



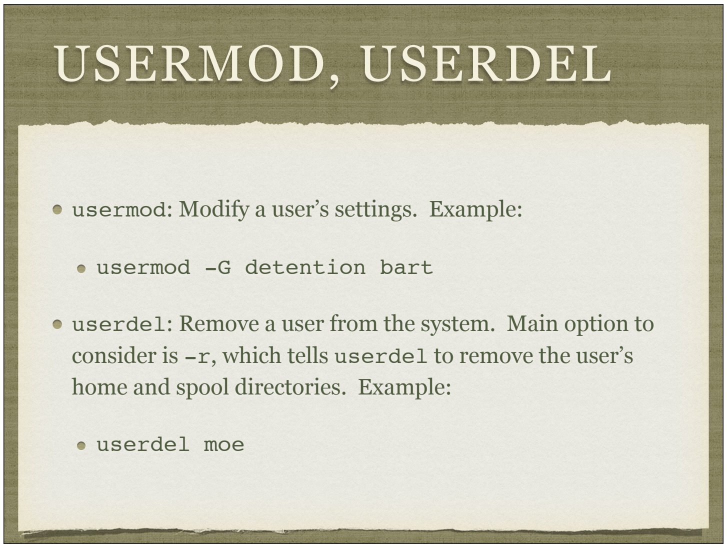
# MANAGEMENT

- 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`



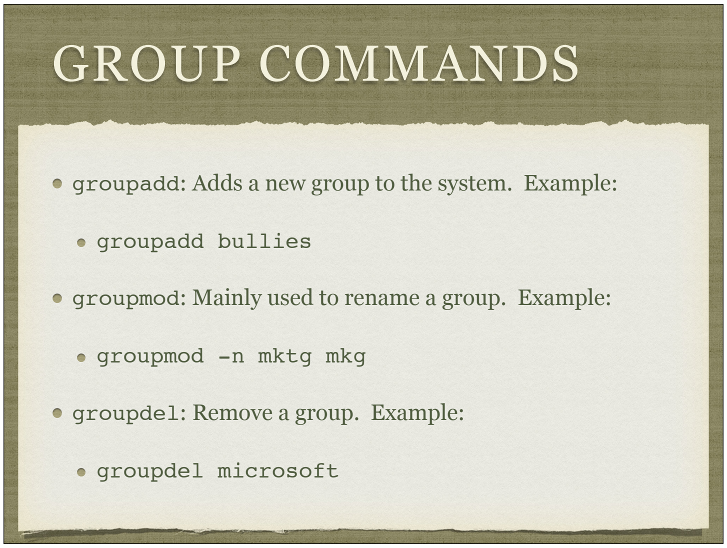
# USERADD

- `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`



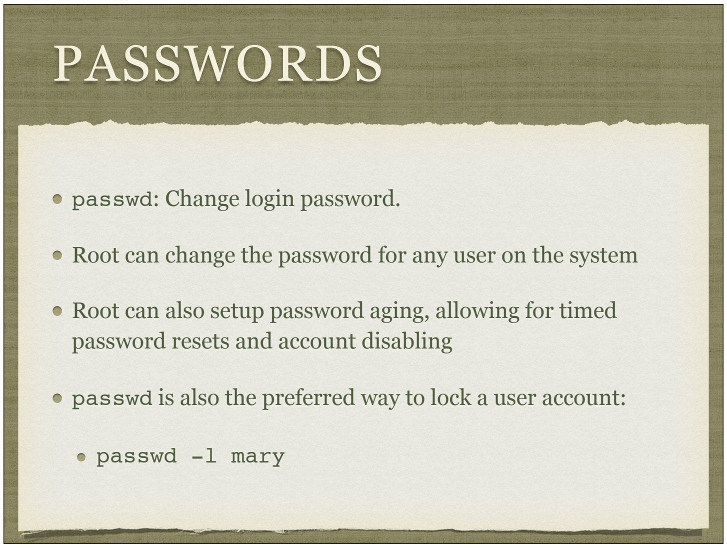
# USERMOD, USERDEL

- `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`



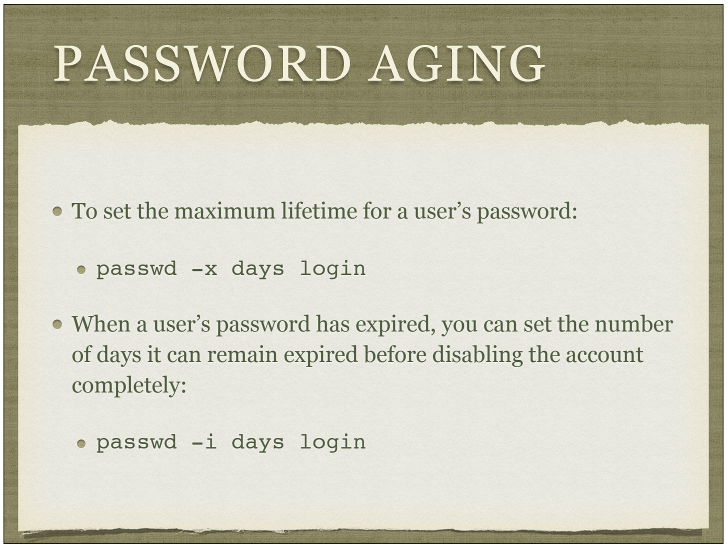
# GROUP COMMANDS

- `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`



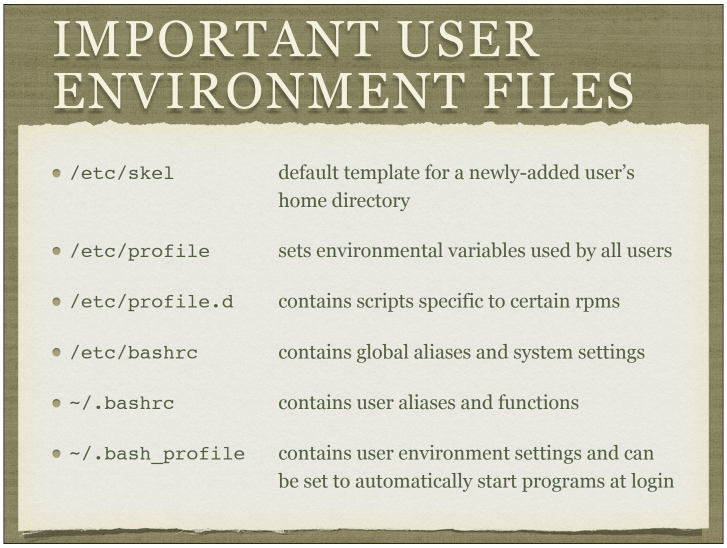
## PASSWORDS

- `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`



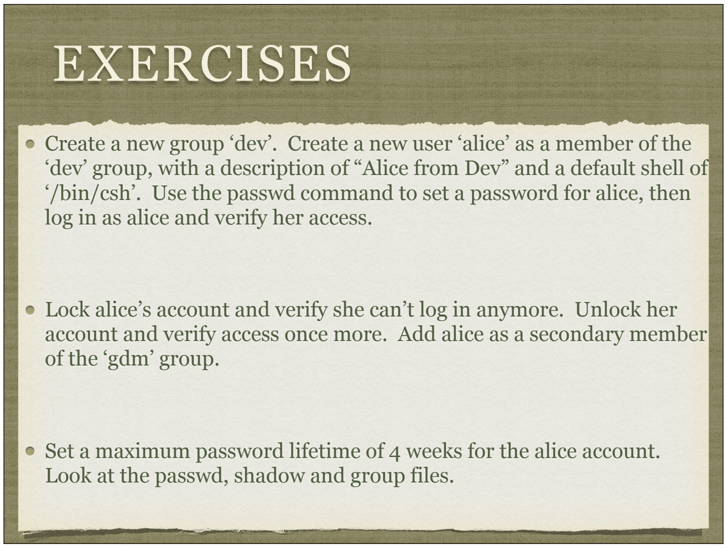
## PASSWORD AGING

- 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`



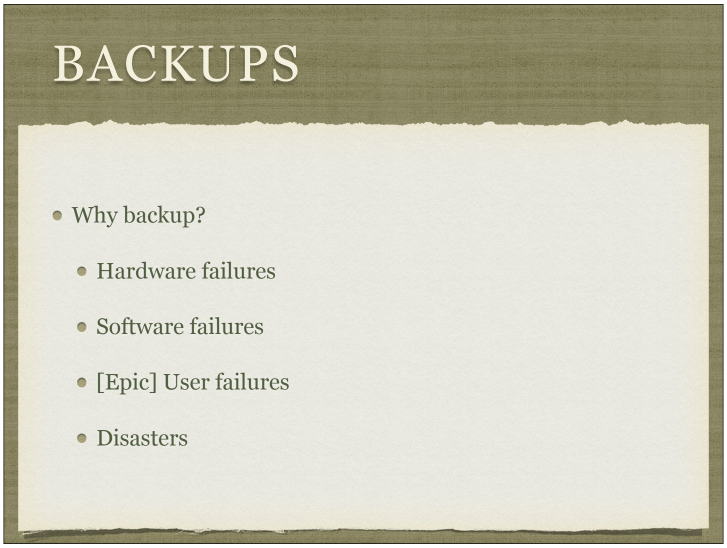
# IMPORTANT USER ENVIRONMENT FILES

| | |
|---|---|
| `/etc/skel` | default template for a newly-added user's home directory |
| `/etc/profile` | sets environmental variables used by all users |
| `/etc/profile.d` | contains scripts specific to certain rpms |
| `/etc/bashrc` | contains global aliases and system settings |
| `~/.bashrc` | contains user aliases and functions |
| `~/.bash_profile` | contains user environment settings and can be set to automatically start programs at login |



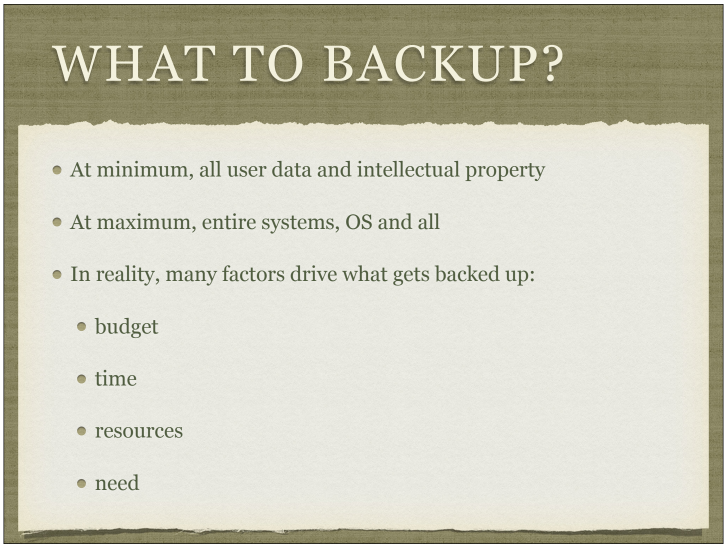
# EXERCISES

- 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.



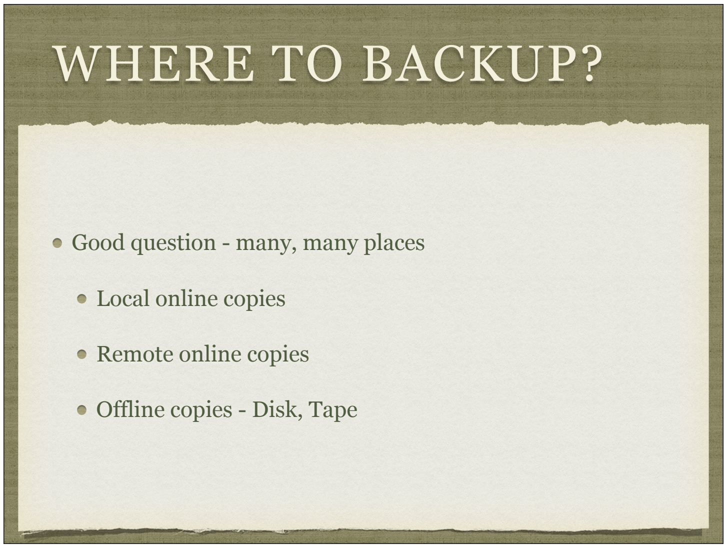
# BACKUPS

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



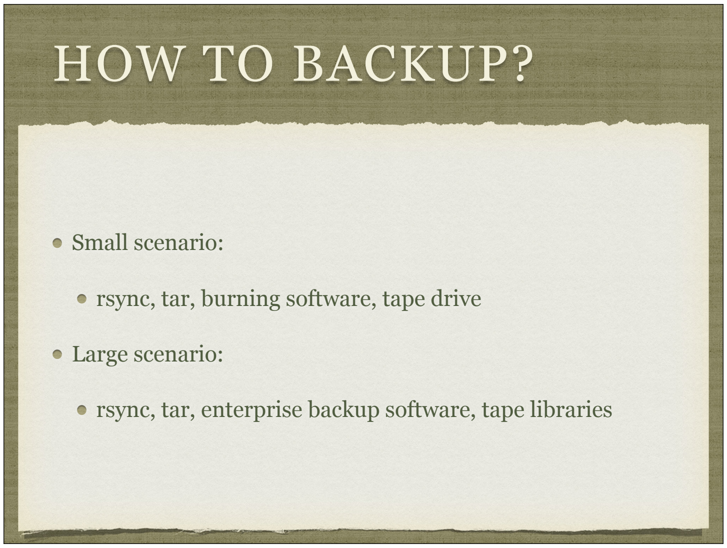
# WHAT TO BACKUP?

- 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



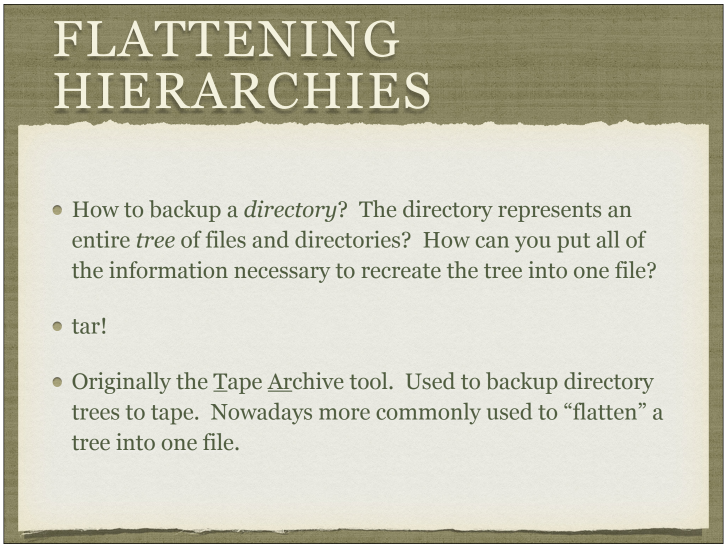
# WHERE TO BACKUP?

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



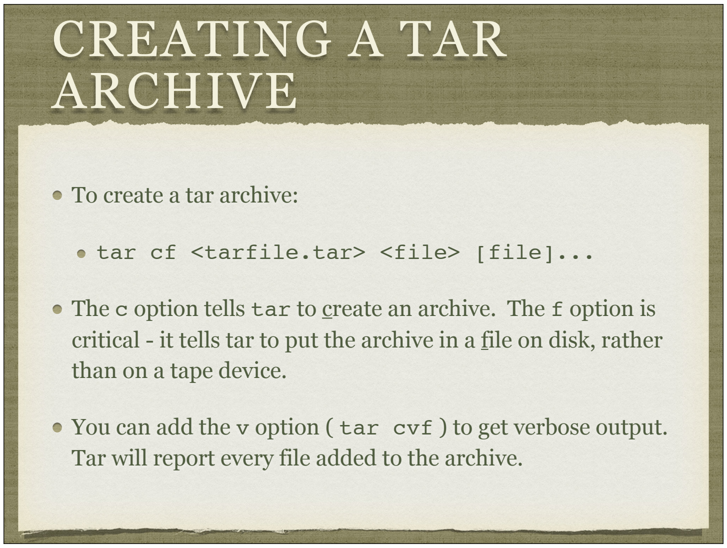
# HOW TO BACKUP?

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



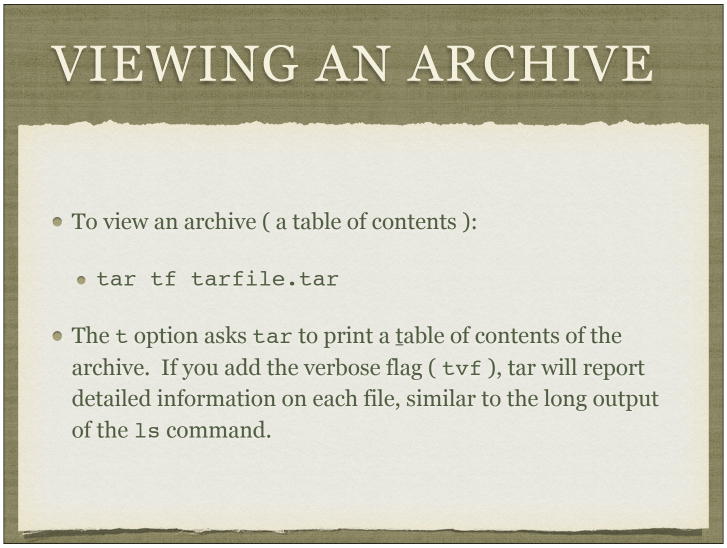
# FLATTENING HIERARCHIES

- 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.



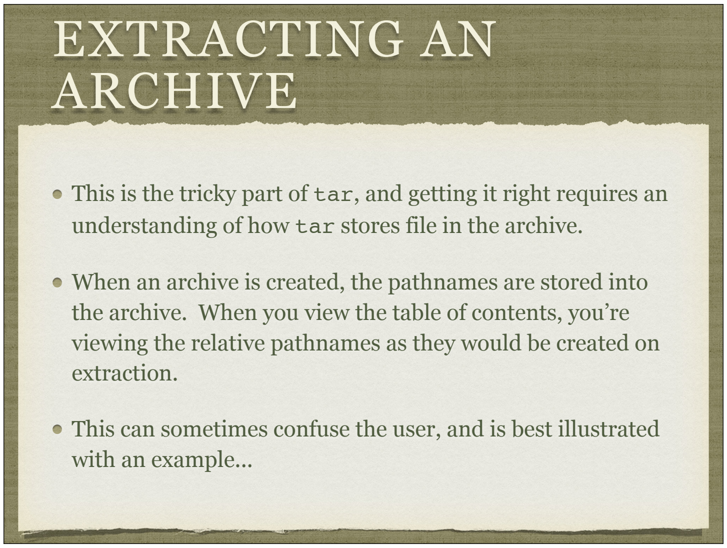
# CREATING A TAR ARCHIVE

- 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.



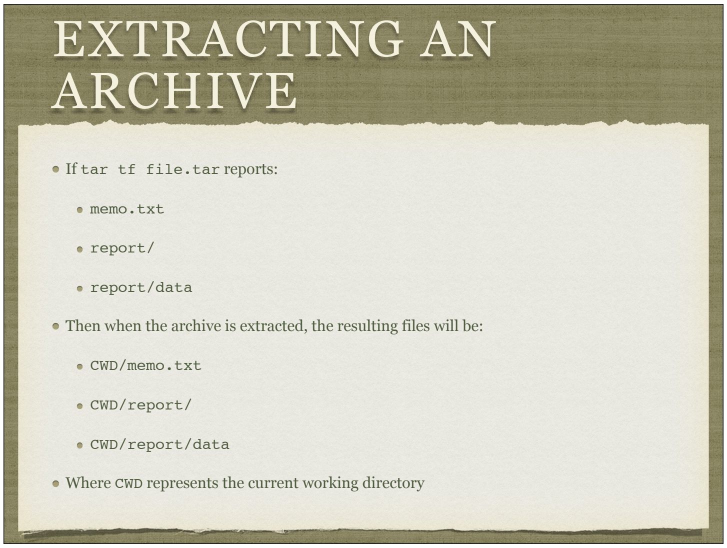
# VIEWING AN ARCHIVE

- 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.



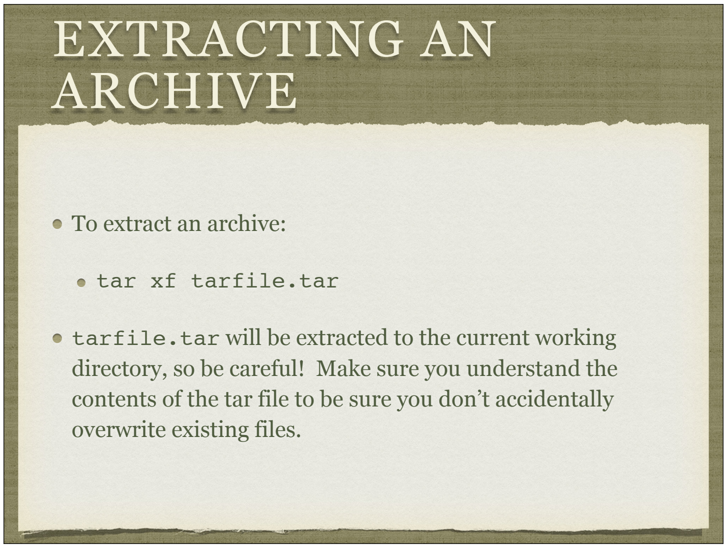
# EXTRACTING AN ARCHIVE

- 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

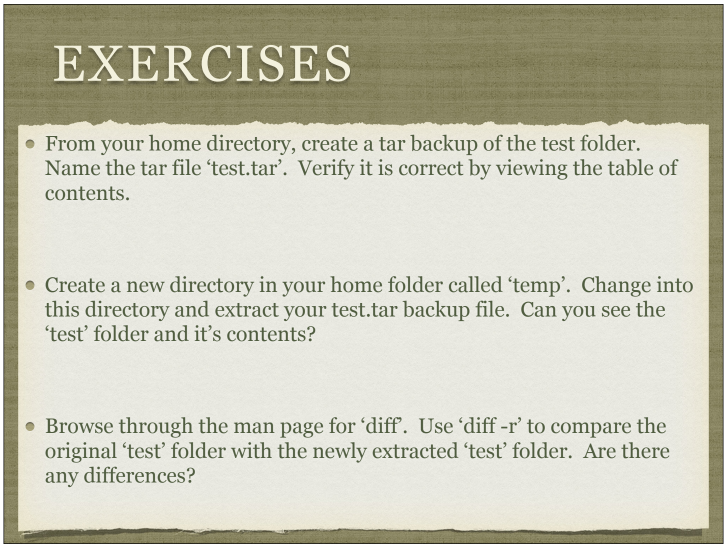


# EXTRACTING AN ARCHIVE

- 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.

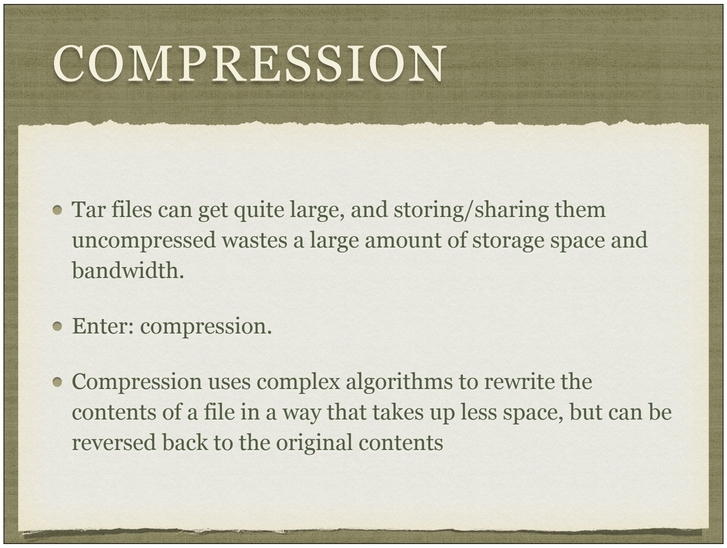
# TAR EXAMPLES

Help remove the mud



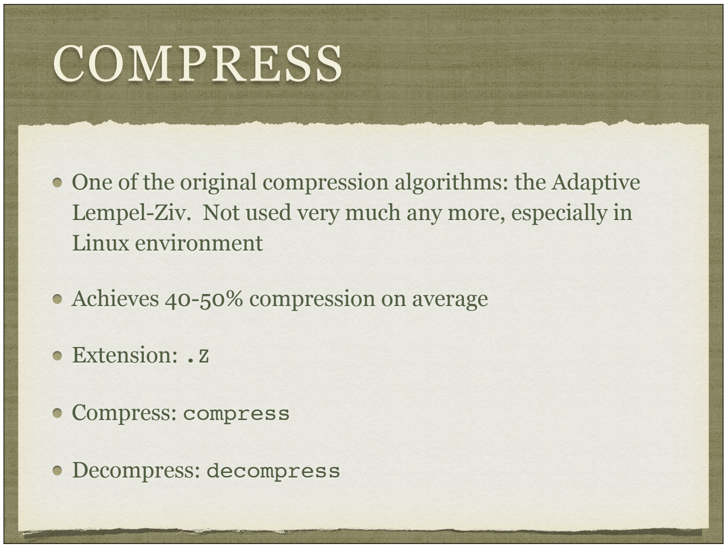
## EXERCISES

- 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?



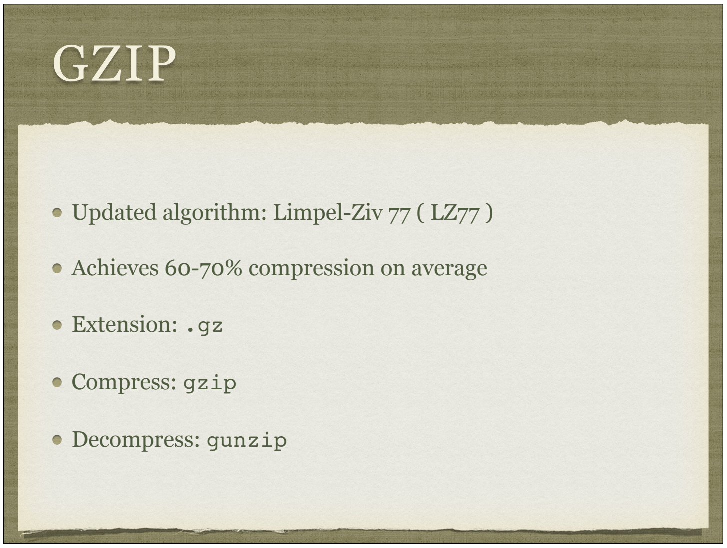
# COMPRESSION

- 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



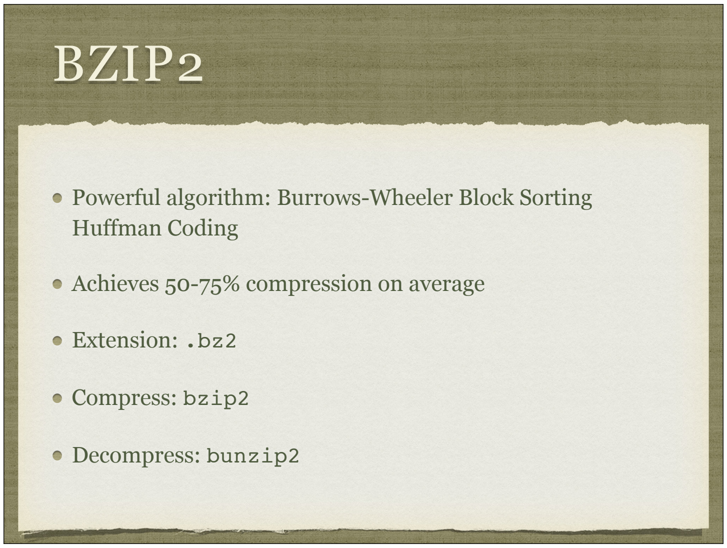
# COMPRESS

- 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`



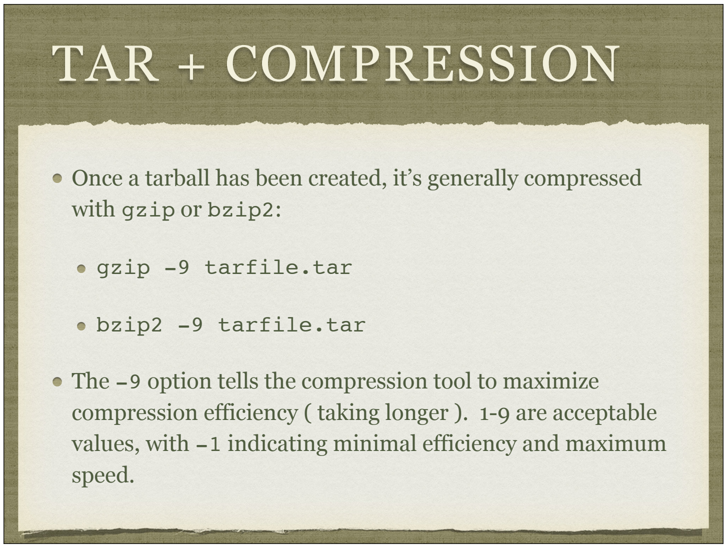
# GZIP

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



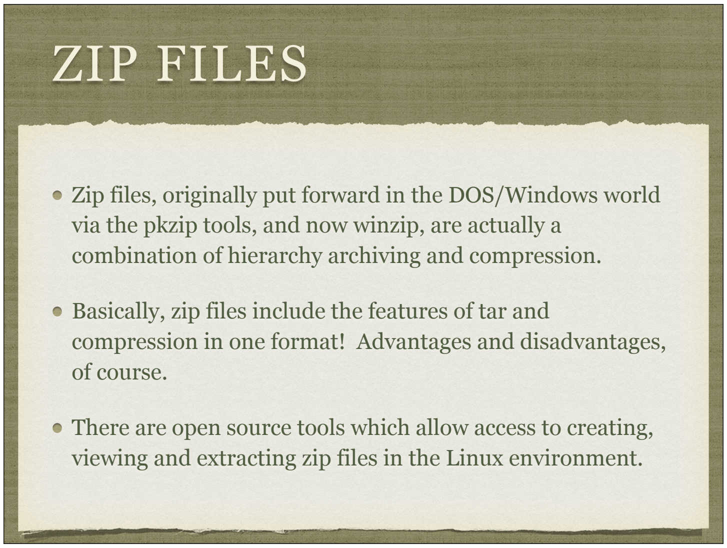
# BZIP2

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



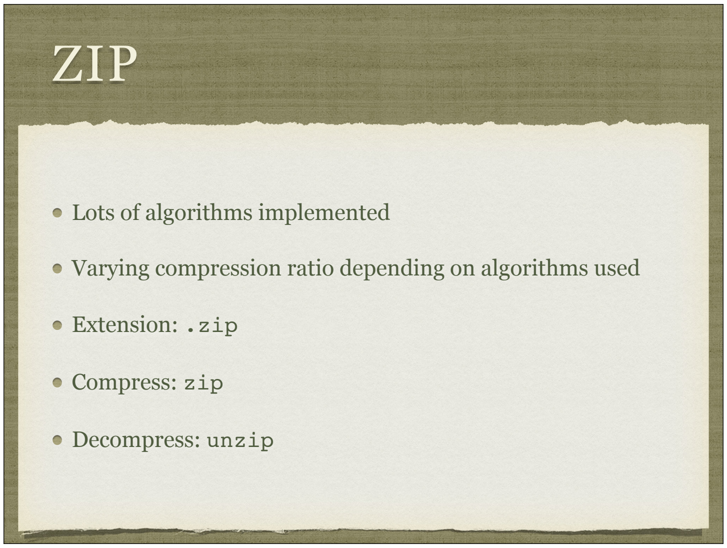
# TAR + COMPRESSION

- 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.



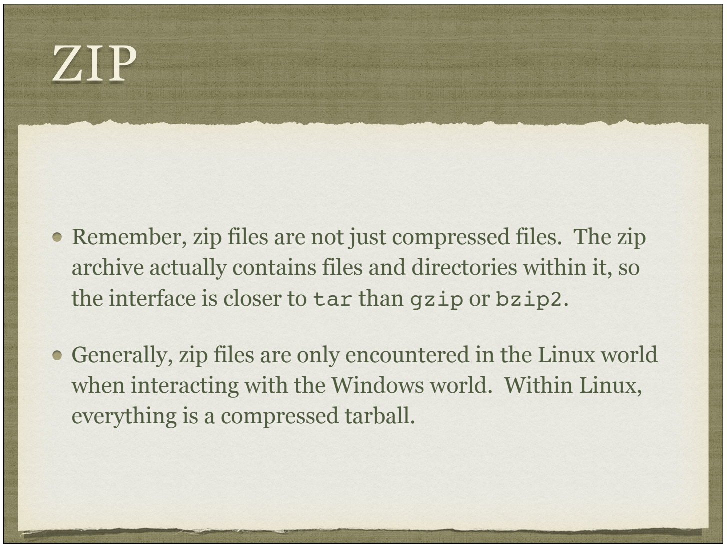
# ZIP FILES

- 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.



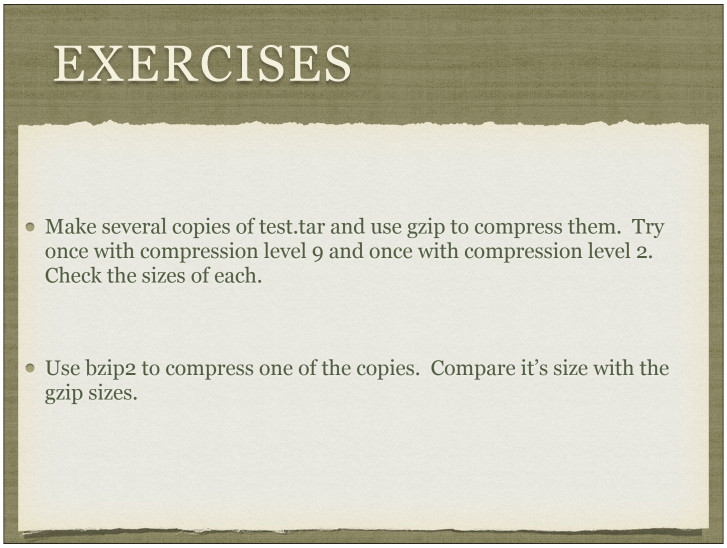
# ZIP

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



# ZIP

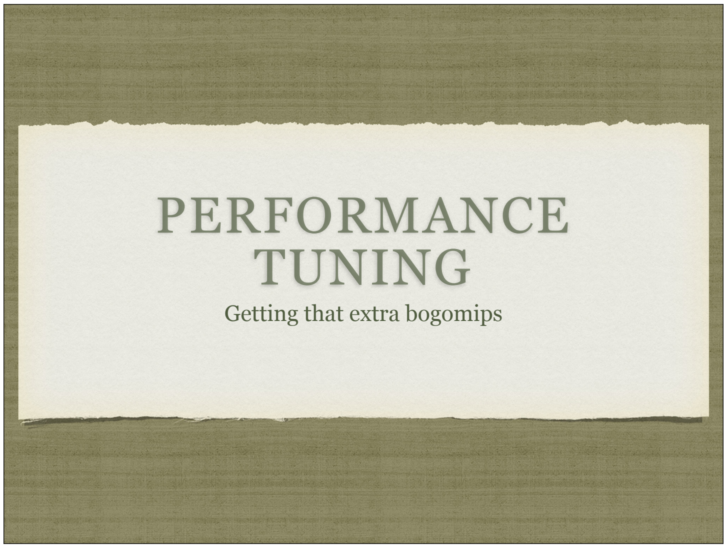
- Remember, zip files are not just compressed files. The zip archive actually contains files and directories within it, so the interface is closer to `tar` than `gzip` or `bzip2`.
- Generally, zip files are only encountered in the Linux world when interacting with the Windows world. Within Linux, everything is a compressed tarball.



# EXERCISES

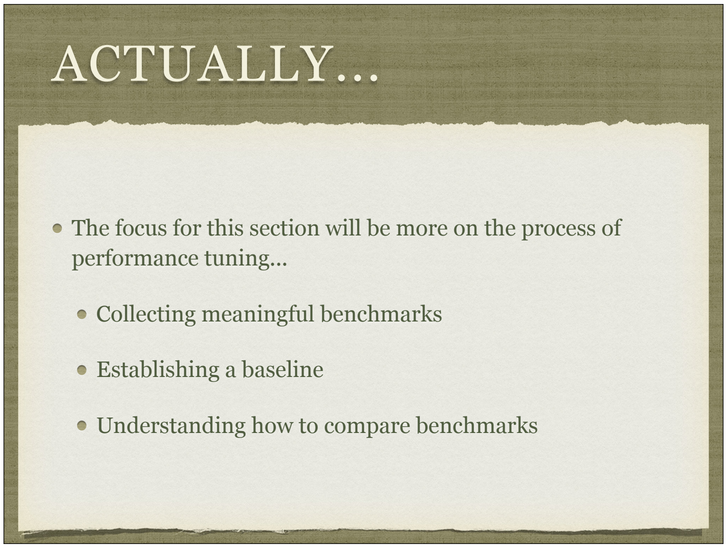
- Make several copies of test.tar and use gzip to compress them.  Try once with compression level 9 and once with compression level 2.  Check the sizes of each.

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



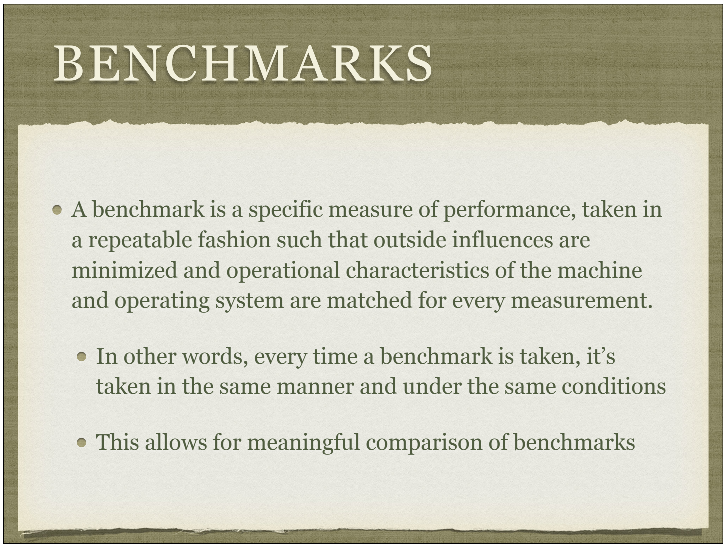
# PERFORMANCE TUNING

Getting that extra bogomips



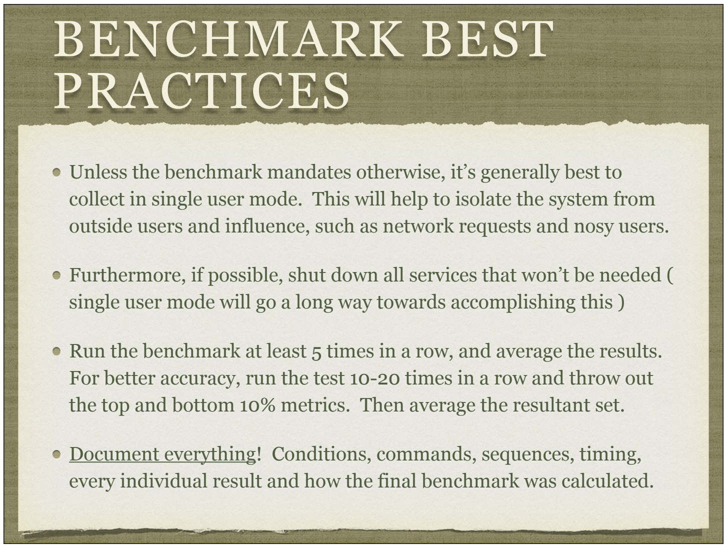
## ACTUALLY...

- The focus for this section will be more on the process of performance tuning...
  - Collecting meaningful benchmarks
  - Establishing a baseline
  - Understanding how to compare benchmarks



# BENCHMARKS

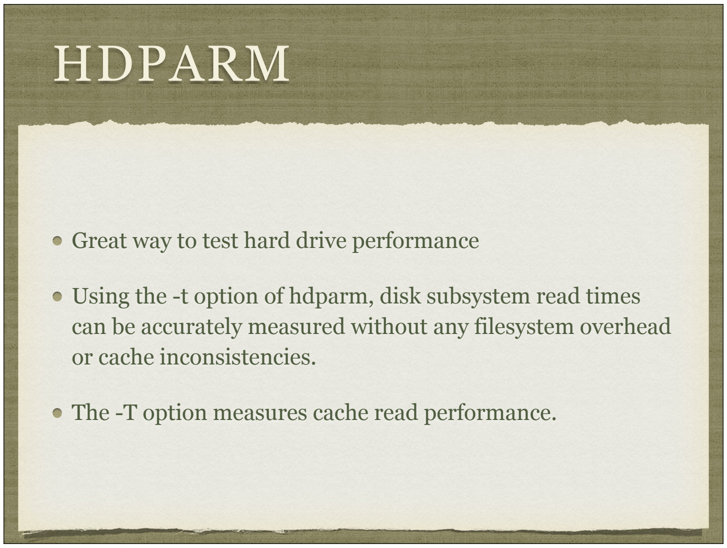
- A benchmark is a specific measure of performance, taken in a repeatable fashion such that outside influences are minimized and operational characteristics of the machine and operating system are matched for every measurement.
  - In other words, every time a benchmark is taken, it's taken in the same manner and under the same conditions
  - This allows for meaningful comparison of benchmarks



# BENCHMARK BEST PRACTICES

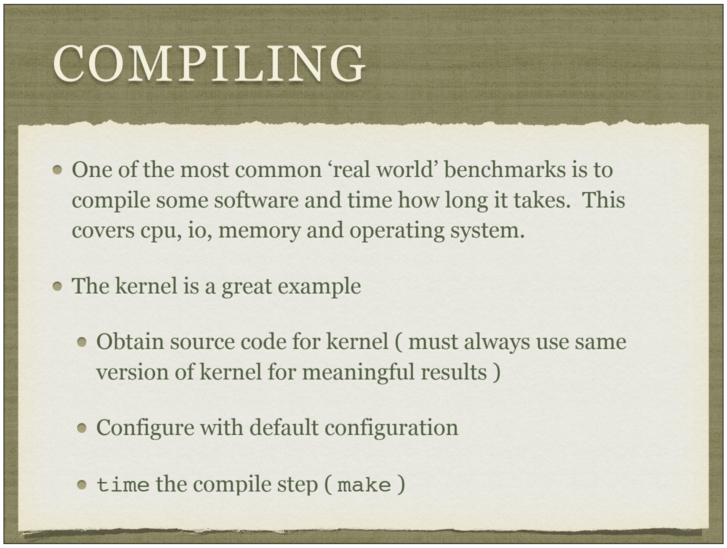
- Unless the benchmark mandates otherwise, it's generally best to collect in single user mode. This will help to isolate the system from outside users and influence, such as network requests and nosy users.
- Furthermore, if possible, shut down all services that won't be needed ( single user mode will go a long way towards accomplishing this )
- Run the benchmark at least 5 times in a row, and average the results. For better accuracy, run the test 10-20 times in a row and throw out the top and bottom 10% metrics. Then average the resultant set.
- Document everything! Conditions, commands, sequences, timing, every individual result and how the final benchmark was calculated.

# BENCHMARKING TOOLS



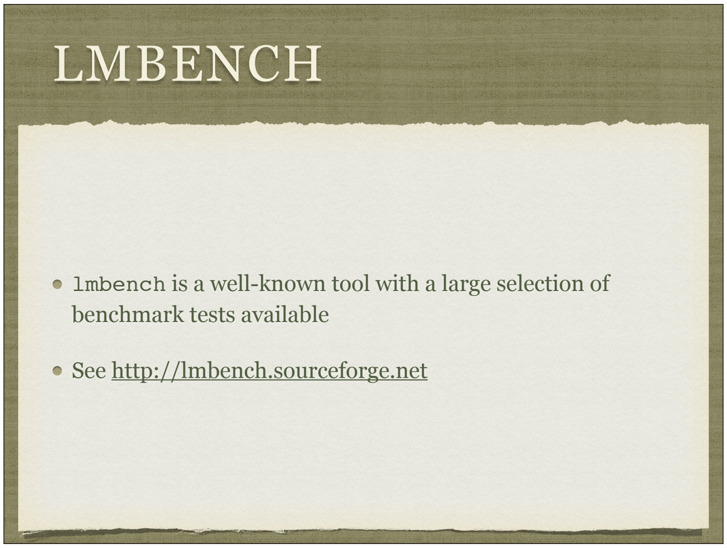
## HDPARM

- Great way to test hard drive performance
- Using the -t option of hdparm, disk subsystem read times can be accurately measured without any filesystem overhead or cache inconsistencies.
- The -T option measures cache read performance.



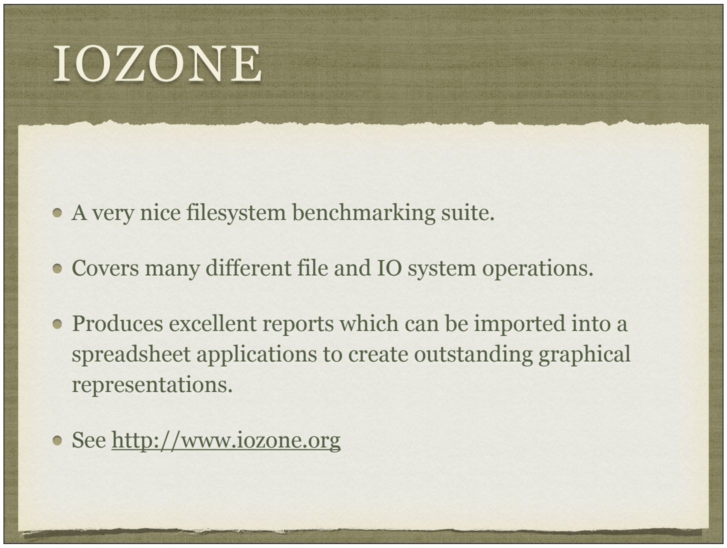
# COMPILING

- One of the most common ‘real world’ benchmarks is to compile some software and time how long it takes.  This covers cpu, io, memory and operating system.
- The kernel is a great example
  - Obtain source code for kernel ( must always use same version of kernel for meaningful results )
  - Configure with default configuration
  - `time` the compile step ( `make` )



# LMBENCH

- `lmbench` is a well-known tool with a large selection of benchmark tests available
- See http://lmbench.sourceforge.net

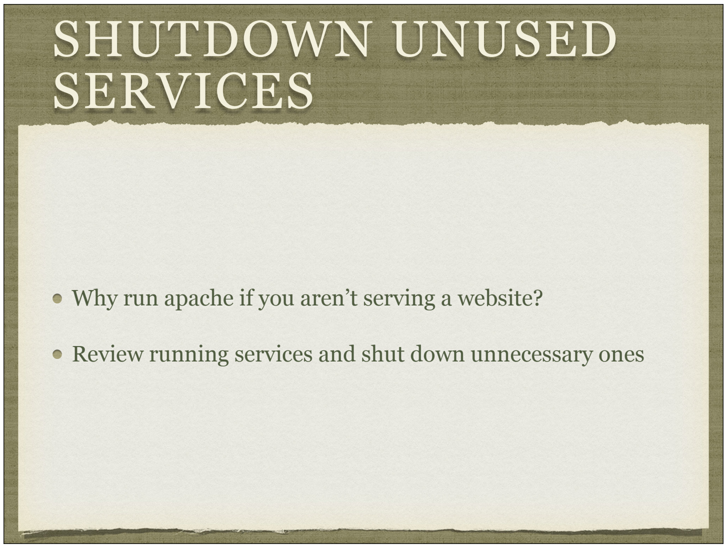


# IOZONE

- A very nice filesystem benchmarking suite.
- Covers many different file and IO system operations.
- Produces excellent reports which can be imported into a spreadsheet applications to create outstanding graphical representations.
- See http://www.iozone.org

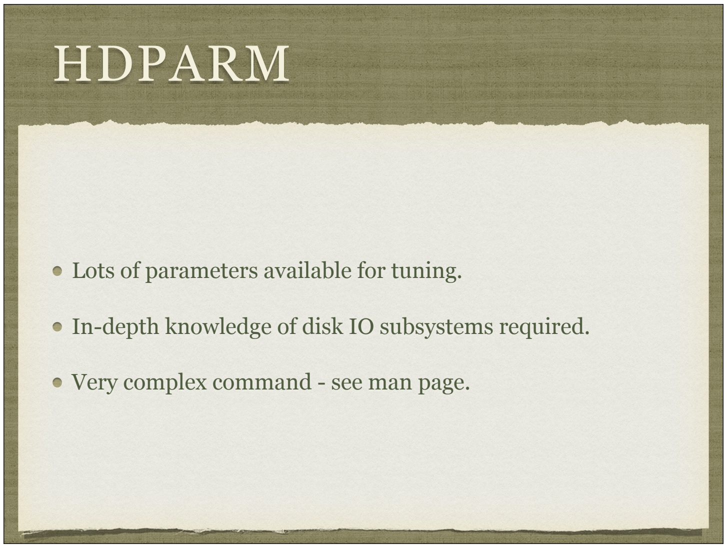
# SOME TUNABLE FEATURES

Play with caution



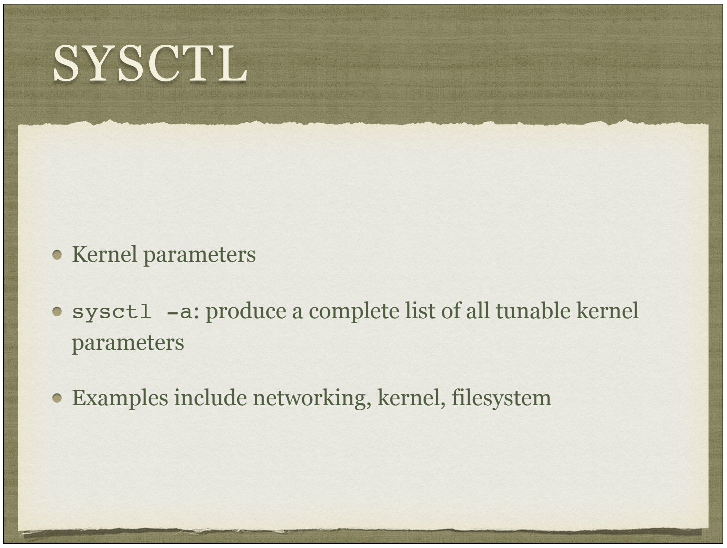
# SHUTDOWN UNUSED SERVICES

- Why run apache if you aren’t serving a website?
- Review running services and shut down unnecessary ones



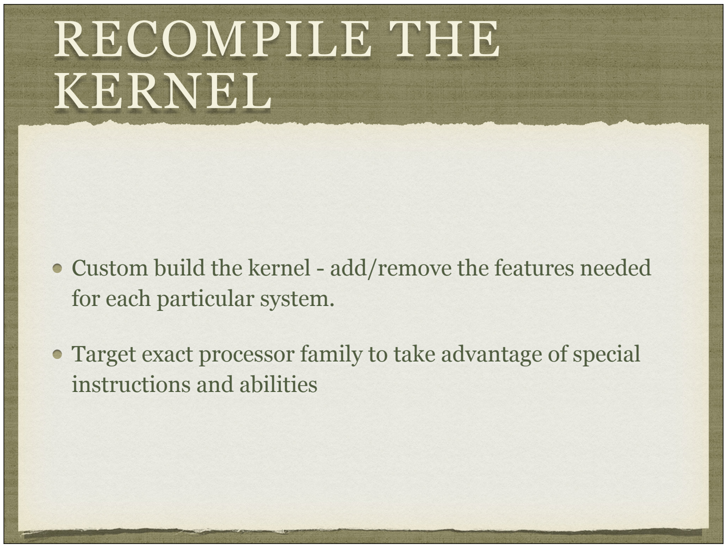
# HDPARM

- Lots of parameters available for tuning.
- In-depth knowledge of disk IO subsystems required.
- Very complex command - see man page.



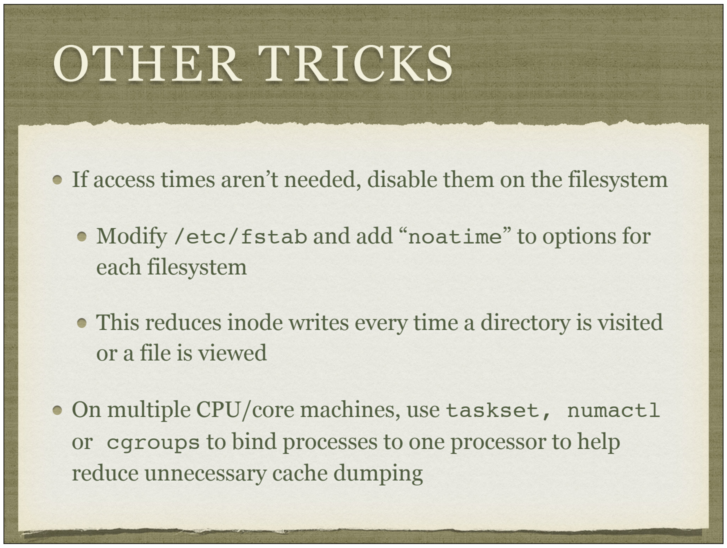
# SYSCTL

- Kernel parameters
- `sysctl -a`: produce a complete list of all tunable kernel parameters
- Examples include networking, kernel, filesystem



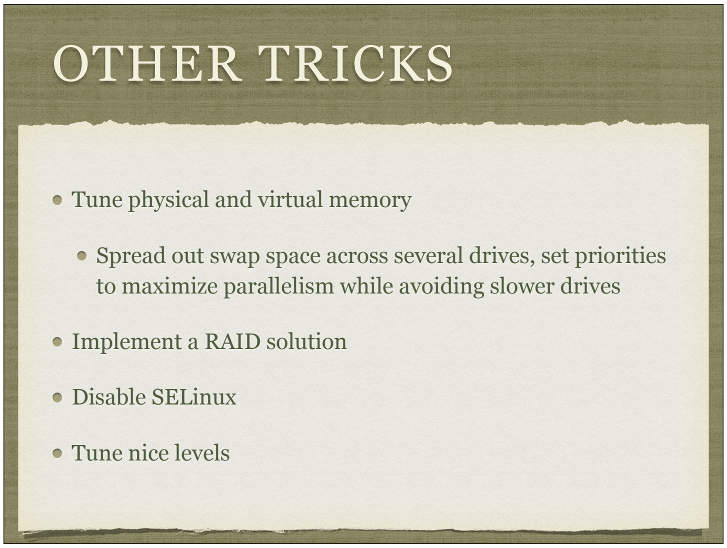
# RECOMPILE THE KERNEL

- Custom build the kernel - add/remove the features needed for each particular system.
- Target exact processor family to take advantage of special instructions and abilities



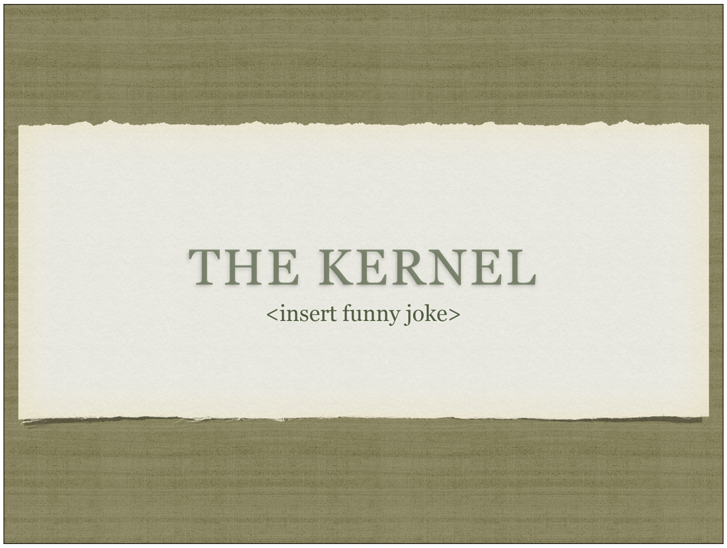
# OTHER TRICKS

- If access times aren’t needed, disable them on the filesystem
  - Modify `/etc/fstab` and add “`noatime`” to options for each filesystem
  - This reduces inode writes every time a directory is visited or a file is viewed
- On multiple CPU/core machines, use `taskset, numactl` or `cgroups` to bind processes to one processor to help reduce unnecessary cache dumping



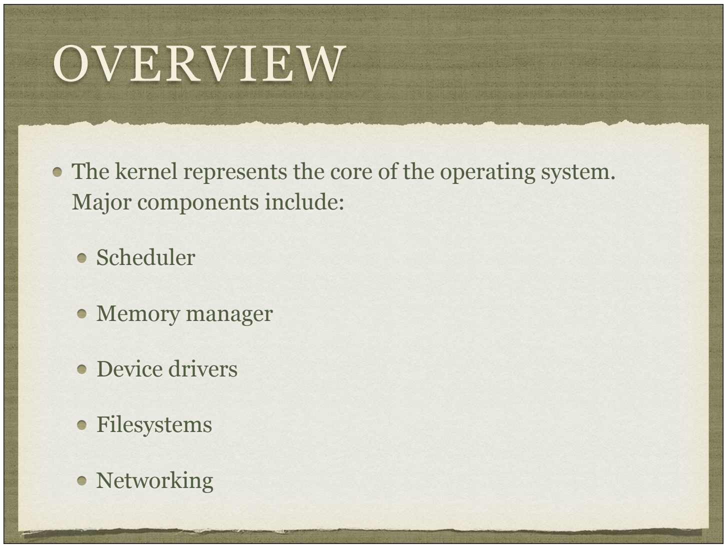
# OTHER TRICKS

- Tune physical and virtual memory
  - Spread out swap space across several drives, set priorities to maximize parallelism while avoiding slower drives
- Implement a RAID solution
- Disable SELinux
- Tune nice levels



# THE KERNEL

<insert funny joke>



# OVERVIEW

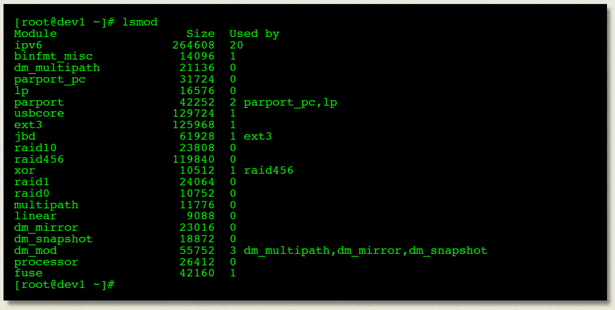
- The kernel represents the core of the operating system. Major components include:
  - Scheduler
  - Memory manager
  - Device drivers
  - Filesystems
  - Networking

# MODULAR

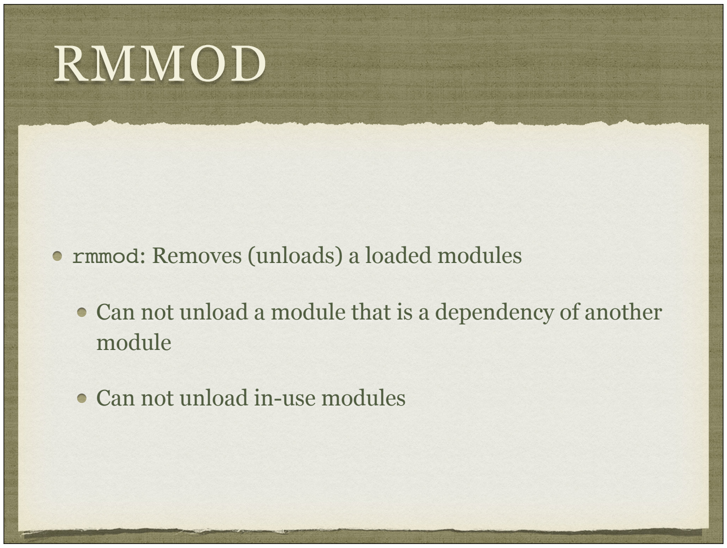
- The Linux kernel is modular, allowing functional blocks of software to be added and removed on the fly via the modules mechanism.
- Modules encompass functions such as:
  - Device drivers
  - Kernel features - firewalls, RAID, LVM
  - Filesystems

# LSMOD

- `lsmod`: Prints all of the currently loaded modules

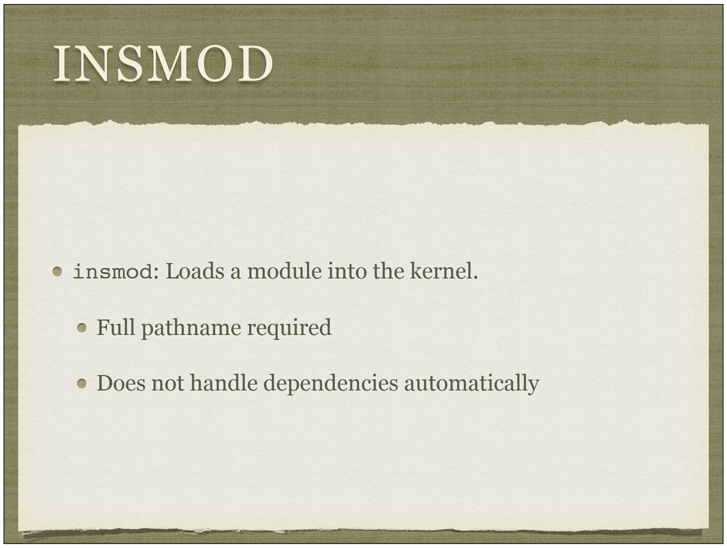


```
[root@dev1 ~]# lsmod
Module                  Size  Used by
ipv6                  264608  20
binfmt_misc            14096  1
dm_multipath           21136  0
parport_pc             31724  0
lp                     16576  0
parport                42252  2 parport_pc,lp
usbcore               129724  1
ext3                  125968  1
jbd                    61928  1 ext3
raid10                 23808  0
raid456               119840  0
xor                    10512  1 raid456
raid1                  24064  0
raid0                  10752  0
multipath              11776  0
linear                  9088  0
dm_mirror              23016  0
dm_snapshot            18872  0
dm_mod                 55752  3 dm_multipath,dm_mirror,dm_snapshot
processor              26412  0
fuse                   42160  1
[root@dev1 ~]#
```



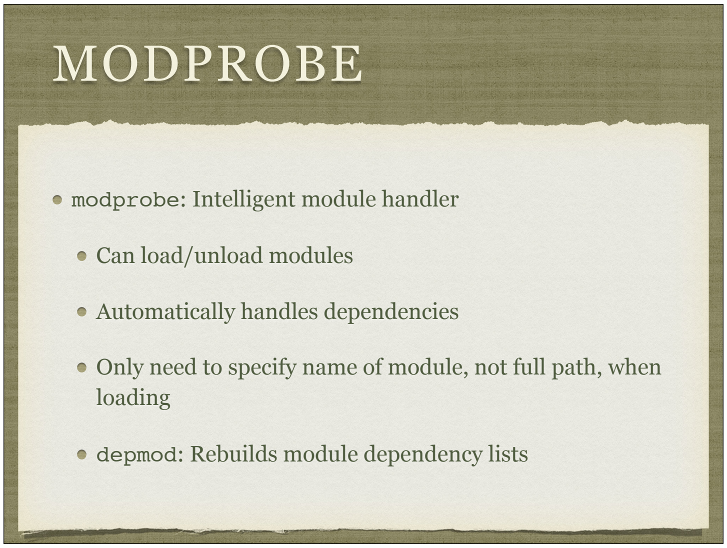
# RMMOD

- `rmmod`: Removes (unloads) a loaded modules
  - Can not unload a module that is a dependency of another module
  - Can not unload in-use modules



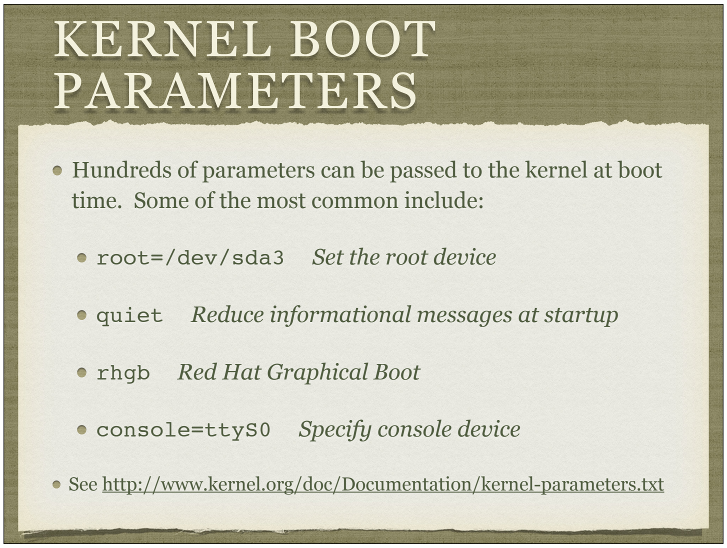
# INSMOD

- `insmod`: Loads a module into the kernel.
  - Full pathname required
  - Does not handle dependencies automatically



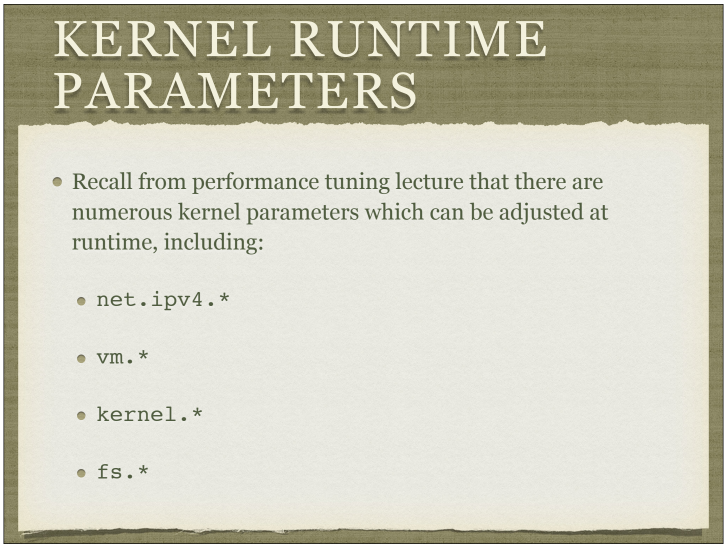
# MODPROBE

- `modprobe`: Intelligent module handler
  - Can load/unload modules
  - Automatically handles dependencies
  - Only need to specify name of module, not full path, when loading
  - `depmod`: Rebuilds module dependency lists



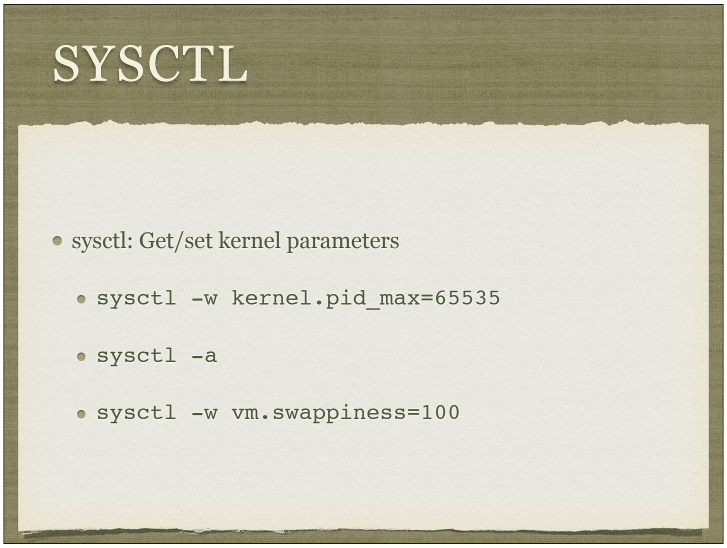
# KERNEL BOOT PARAMETERS

- Hundreds of parameters can be passed to the kernel at boot time. Some of the most common include:
  - `root=/dev/sda3` *Set the root device*
  - `quiet` *Reduce informational messages at startup*
  - `rhgb` *Red Hat Graphical Boot*
  - `console=ttyS0` *Specify console device*
- See http://www.kernel.org/doc/Documentation/kernel-parameters.txt



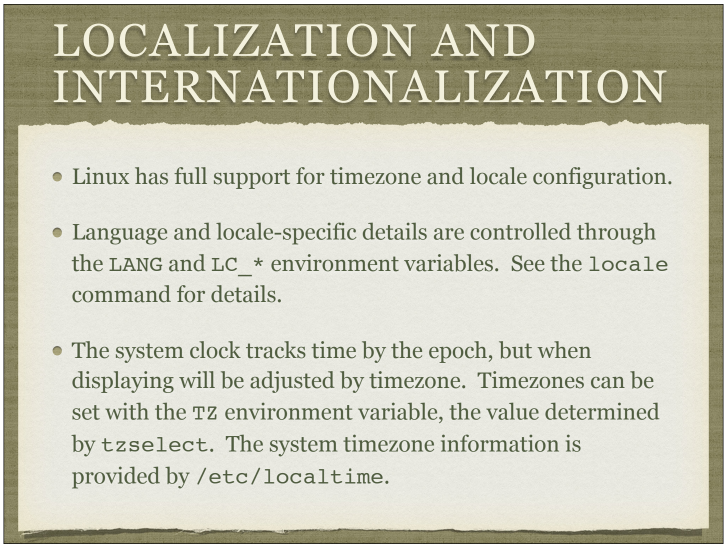
# KERNEL RUNTIME PARAMETERS

- Recall from performance tuning lecture that there are numerous kernel parameters which can be adjusted at runtime, including:
  - `net.ipv4.*`
  - `vm.*`
  - `kernel.*`
  - `fs.*`



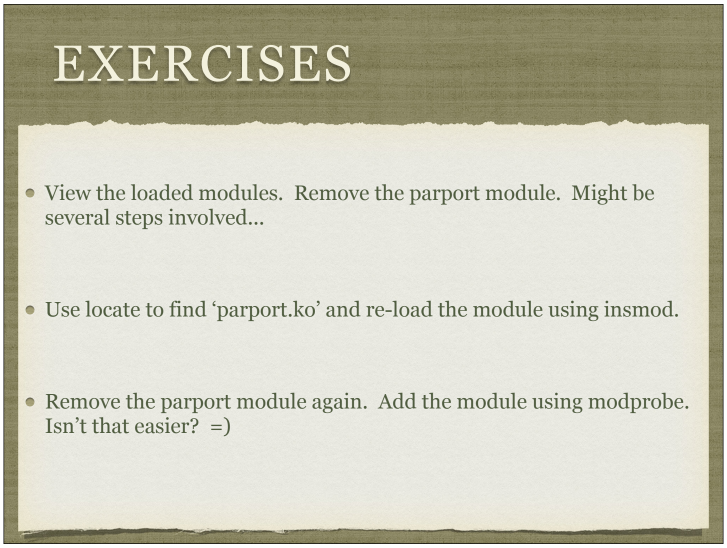
# SYSCTL

- sysctl: Get/set kernel parameters
  - `sysctl -w kernel.pid_max=65535`
  - `sysctl -a`
  - `sysctl -w vm.swappiness=100`



# LOCALIZATION AND INTERNATIONALIZATION

- Linux has full support for timezone and locale configuration.
- Language and locale-specific details are controlled through the `LANG` and `LC_*` environment variables. See the `locale` command for details.
- The system clock tracks time by the epoch, but when displaying will be adjusted by timezone. Timezones can be set with the `TZ` environment variable, the value determined by `tzselect`. The system timezone information is provided by `/etc/localtime`.



# EXERCISES

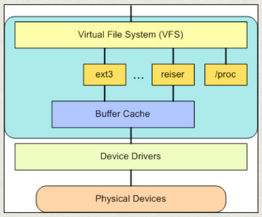
- View the loaded modules. Remove the parport module. Might be several steps involved...
- Use locate to find ‘parport.ko’ and re-load the module using insmod.
- Remove the parport module again. Add the module using modprobe. Isn’t that easier? =)

# FILESYSTEM ADMINISTRATION

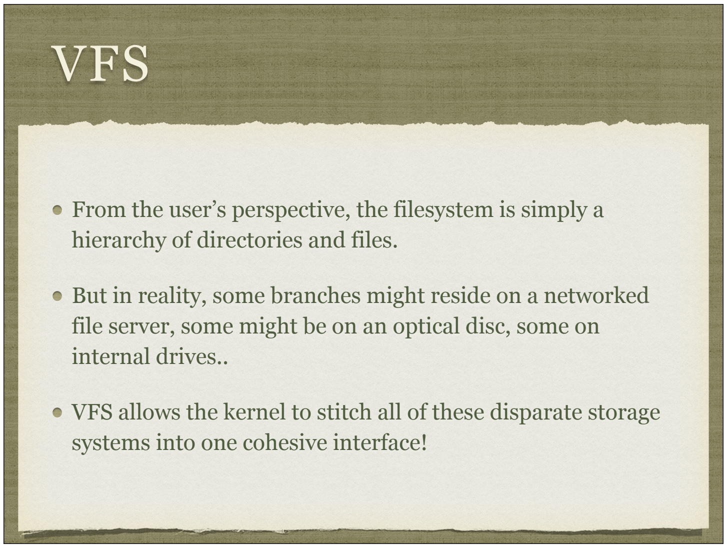
mount? umount? mkfs? fsck?

# 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.

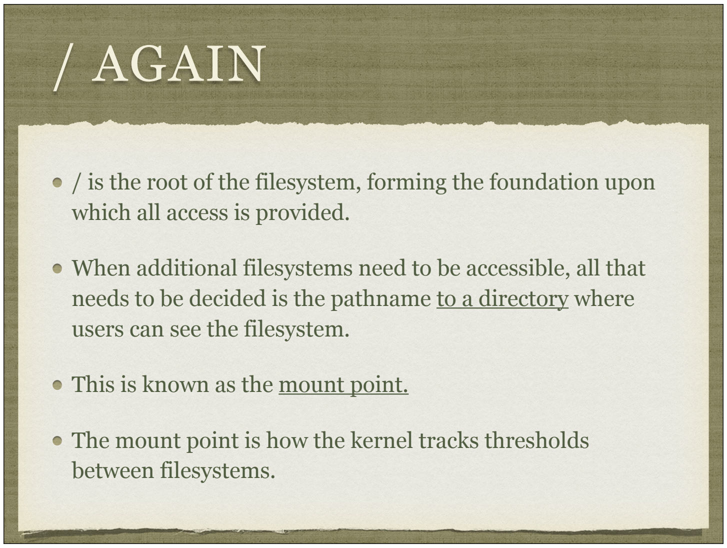


http://www.ibm.com/developerworks/linux/library/l-linux-kernel/



# VFS

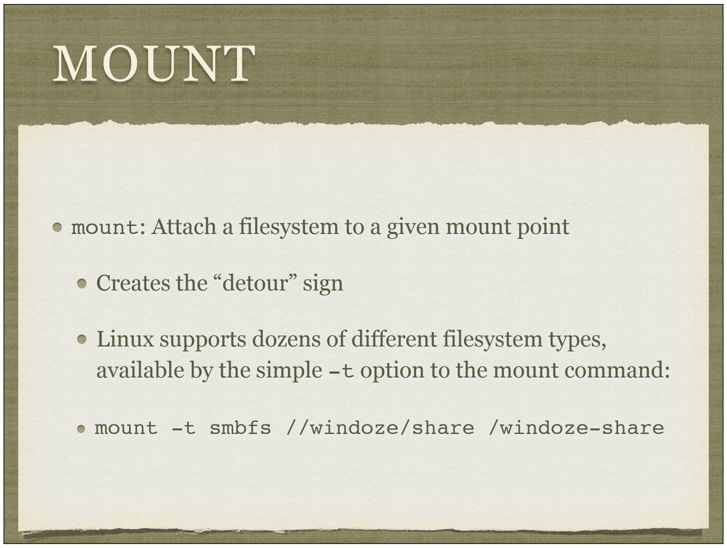
- From the user’s perspective, the filesystem is simply a hierarchy of directories and files.
- But in reality, some branches might reside on a networked file server, some might be on an optical disc, some on internal drives..
- VFS allows the kernel to stitch all of these disparate storage systems into one cohesive interface!



# / AGAIN

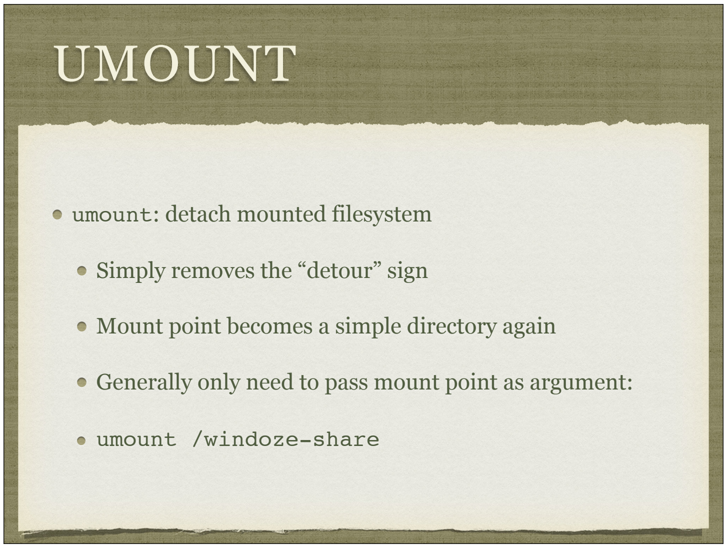
- / is the root of the filesystem, forming the foundation upon which all access is provided.
- When additional filesystems need to be accessible, all that needs to be decided is the pathname to a directory where users can see the filesystem.
- This is known as the mount point.
- The mount point is how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

# LET’S SEE THIS ON THE WHITEBOARD



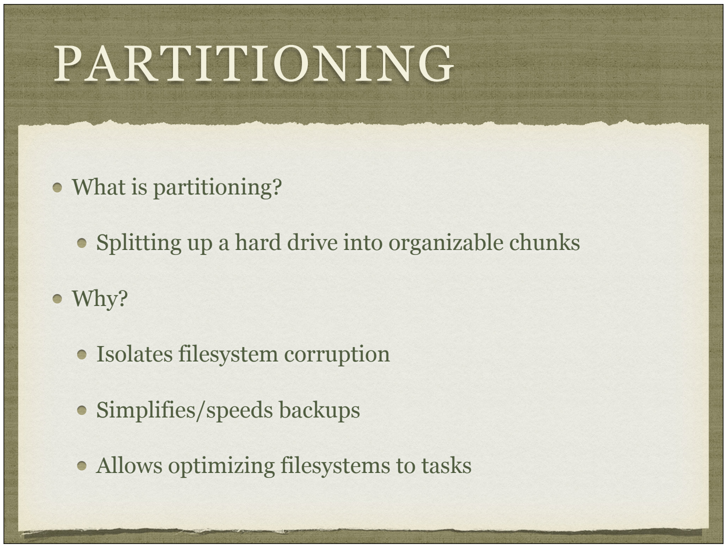
# MOUNT

- `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`



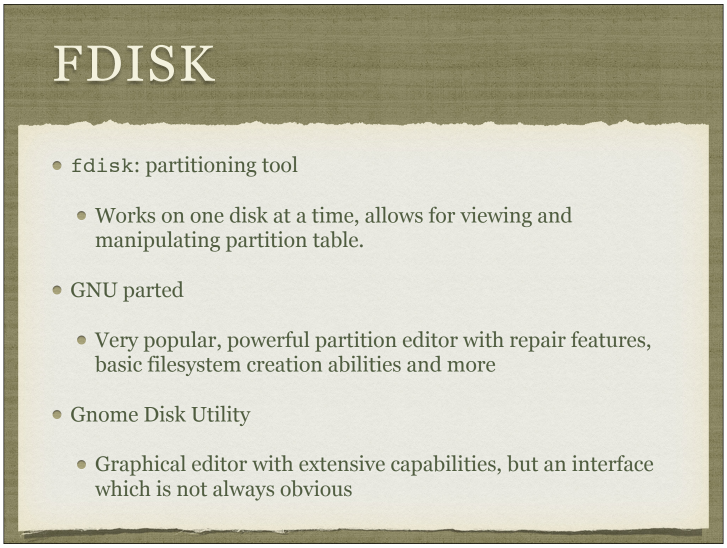
# UMOUNT

- `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`



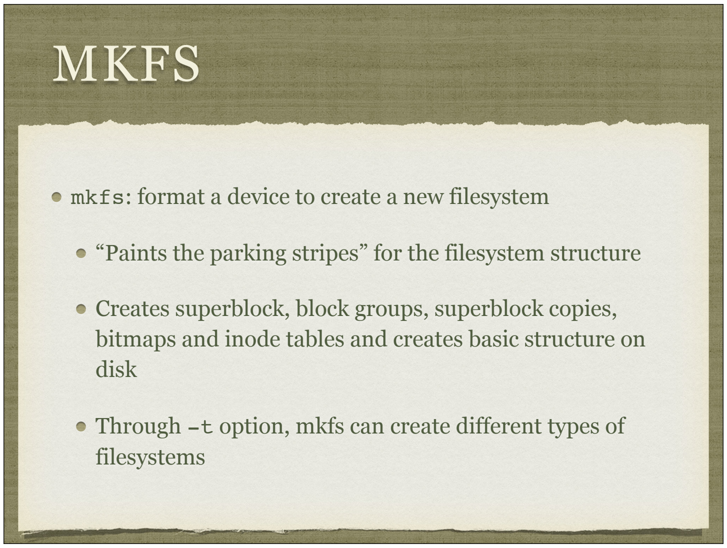
# PARTITIONING

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



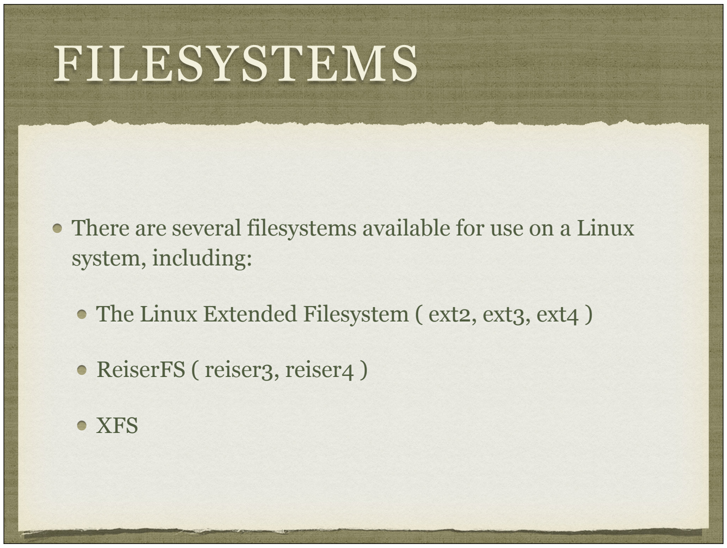
# FDISK

- `fdisk`: partitioning tool
  - Works on one disk at a time, allows for viewing and manipulating partition table.
- GNU parted
  - Very popular, powerful partition editor with repair features, basic filesystem creation abilities and more
- Gnome Disk Utility
  - Graphical editor with extensive capabilities, but an interface which is not always obvious



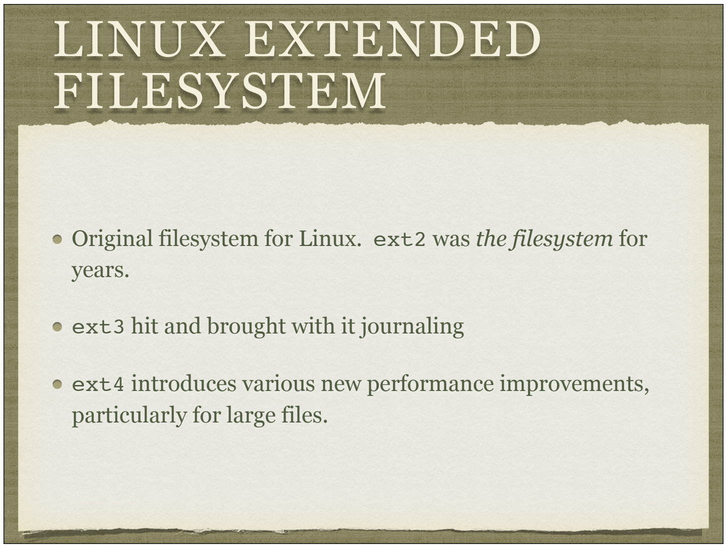
# MKFS

- `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



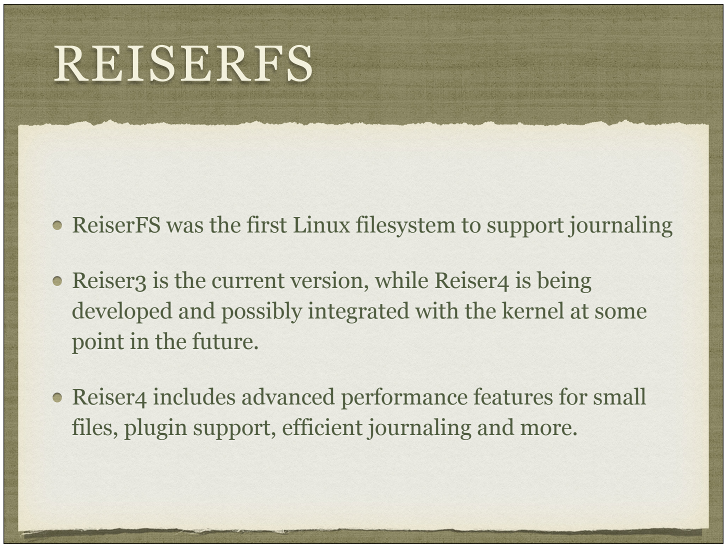
# FILESYSTEMS

- There are several filesystems available for use on a Linux system, including:
  - The Linux Extended Filesystem ( ext2, ext3, ext4 )
  - ReiserFS ( reiser3, reiser4 )
  - XFS



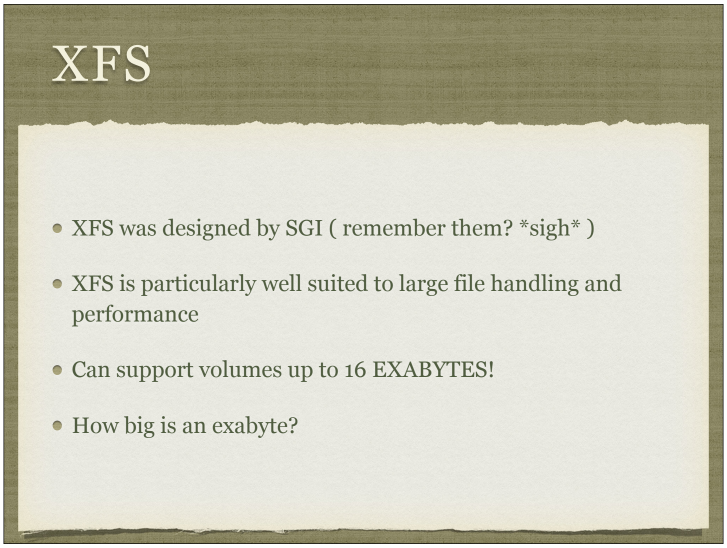
# LINUX EXTENDED FILESYSTEM

- Original filesystem for Linux. `ext2` was *the filesystem* for years.
- `ext3` hit and brought with it journaling
- `ext4` introduces various new performance improvements, particularly for large files.



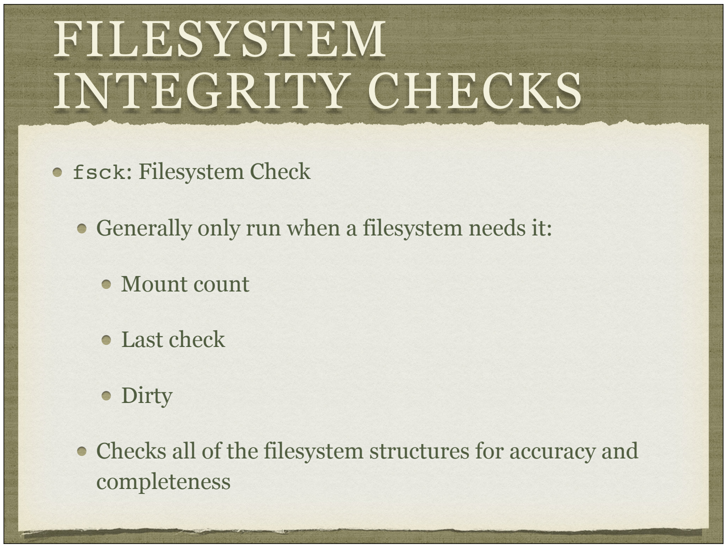
# REISERFS

- ReiserFS was the first Linux filesystem to support journaling
- Reiser3 is the current version, while Reiser4 is being developed and possibly integrated with the kernel at some point in the future.
- Reiser4 includes advanced performance features for small files, plugin support, efficient journaling and more.



# XFS

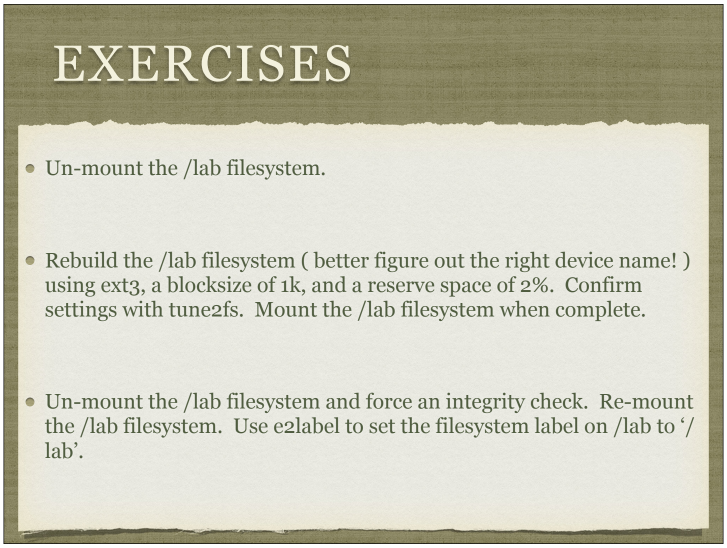
- XFS was designed by SGI ( remember them? *sigh* )
- XFS is particularly well suited to large file handling and performance
- Can support volumes up to 16 EXABYTES!
- How big is an exabyte?



# 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

# EXAMPLES

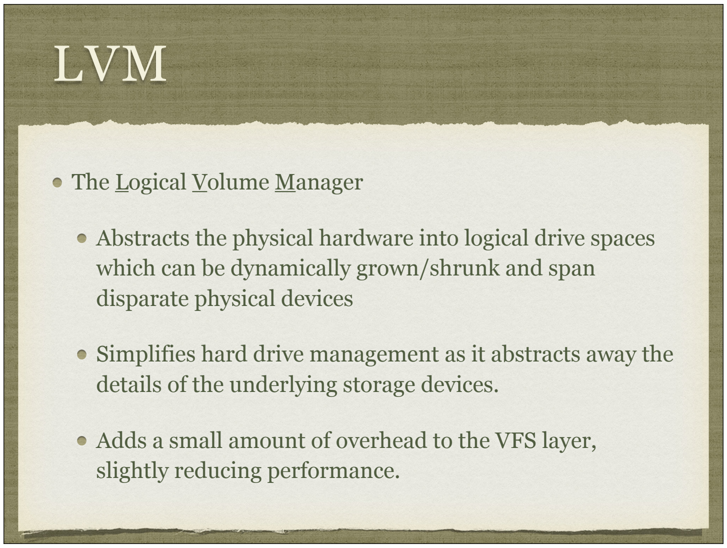


## EXERCISES

- Un-mount the /lab filesystem.

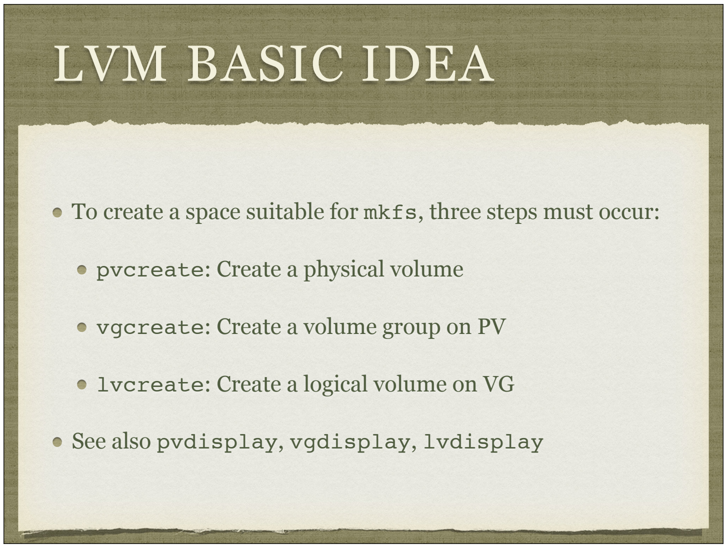
- Rebuild the /lab filesystem ( better figure out the right device name! ) using ext3, a blocksize of 1k, and a reserve space of 2%. Confirm settings with tune2fs. Mount the /lab filesystem when complete.

- 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'.



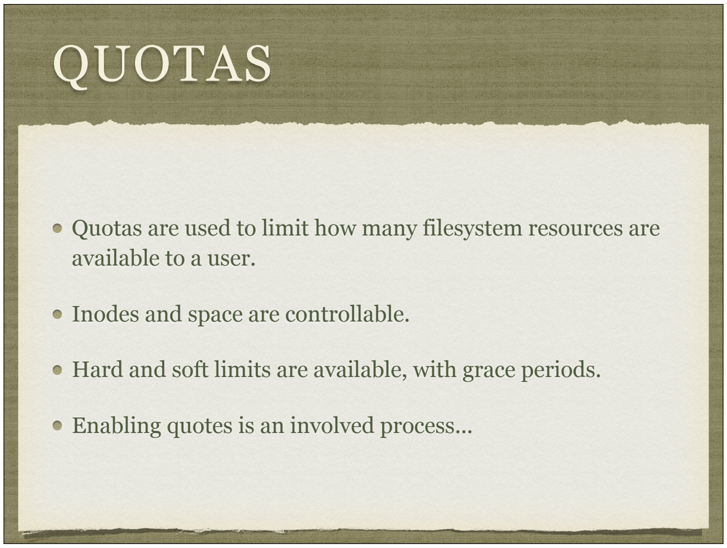
## LVM

- The Logical Volume Manager
  - Abstracts the physical hardware into logical drive spaces which can be dynamically grown/shrunk and span disparate physical devices
  - Simplifies hard drive management as it abstracts away the details of the underlying storage devices.
  - Adds a small amount of overhead to the VFS layer, slightly reducing performance.



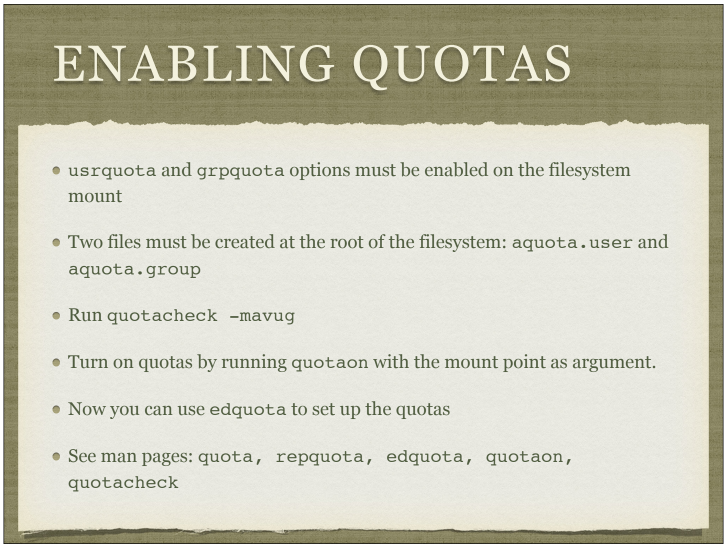
# LVM BASIC IDEA

- To create a space suitable for `mkfs`, three steps must occur:
  - `pvcreate`: Create a physical volume
  - `vgcreate`: Create a volume group on PV
  - `lvcreate`: Create a logical volume on VG
- See also `pvdisplay`, `vgdisplay`, `lvdisplay`



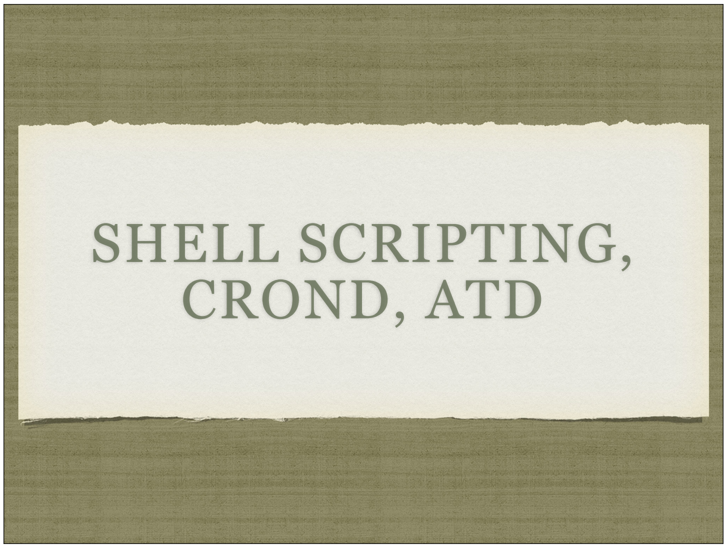
# QUOTAS

- Quotas are used to limit how many filesystem resources are available to a user.
- Inodes and space are controllable.
- Hard and soft limits are available, with grace periods.
- Enabling quotes is an involved process...

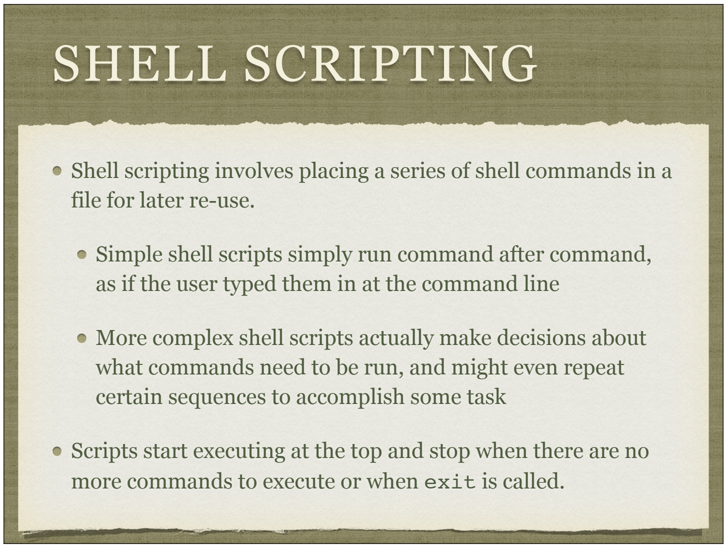


# ENABLING QUOTAS

- `usrquota` and `grpquota` options must be enabled on the filesystem mount
- Two files must be created at the root of the filesystem: `aquota.user` and `aquota.group`
- Run `quotacheck -mavug`
- Turn on quotas by running `quotaon` with the mount point as argument.
- Now you can use `edquota` to set up the quotas
- See man pages: `quota, repquota, edquota, quotaon, quotacheck`



# SHELL SCRIPTING, CROND, ATD



## SHELL SCRIPTING

- Shell scripting involves placing a series of shell commands in a file for later re-use.
  - Simple shell scripts simply run command after command, as if the user typed them in at the command line
  - More complex shell scripts actually make decisions about what commands need to be run, and might even repeat certain sequences to accomplish some task
- Scripts start executing at the top and stop when there are no more commands to execute or when `exit` is called.

# EXAMPLE SHELL SCRIPT

- Here is an example of a very simple shell script:

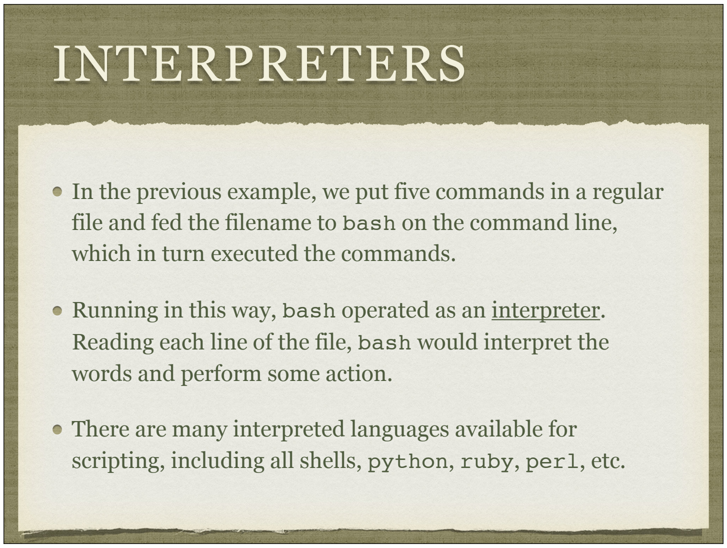
```
echo “Hello, what is your name?”
read NAME
echo “Hello $NAME, it’s nice to meet you!”
echo -n “The current time is: “
date
```

- Using the `echo` command, this script asks a question.
- The `read` command accepts input from the user and stores it in the environment variable `NAME`
- The script finishes up with a couple more echo statements, greeting the user and announcing today’s date

# SHELL SCRIPTING

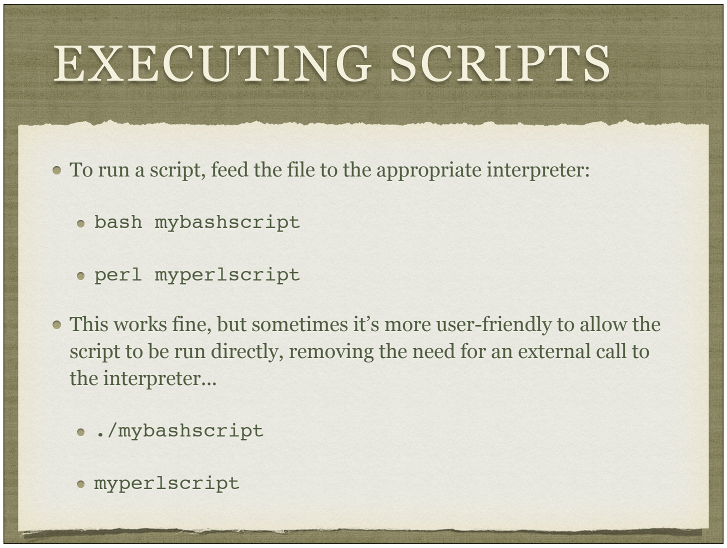
- If we put the example in a file called `myscript`, we can execute the script as:
  - `bash myscript`
- `bash` will open `myscript` and execute each line as if the user had typed it in manually.

```
[root@localhost ~]# bash myscript
Hello, what is your name?
Linus
Hello Linus, it’s nice to meet you!
The current time is: Sun Nov 29 09:39:33 CST 2009
[root@localhost ~]#
```



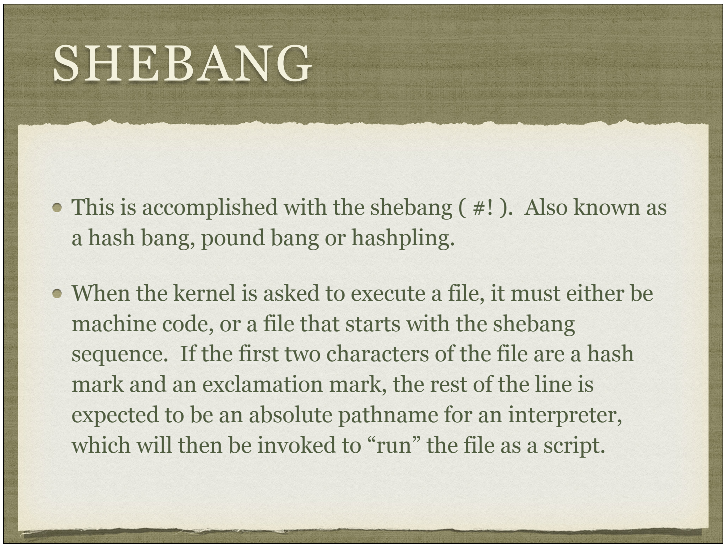
# INTERPRETERS

- In the previous example, we put five commands in a regular file and fed the filename to `bash` on the command line, which in turn executed the commands.
- Running in this way, `bash` operated as an interpreter. Reading each line of the file, `bash` would interpret the words and perform some action.
- There are many interpreted languages available for scripting, including all shells, `python`, `ruby`, `perl`, etc.



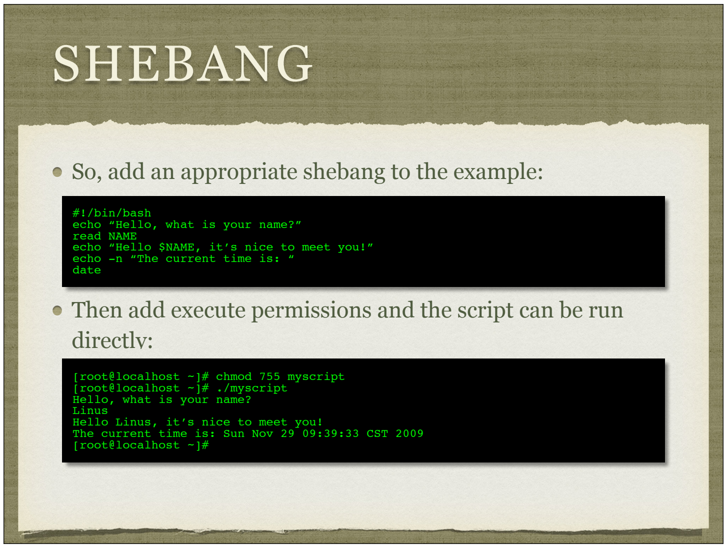
# EXECUTING SCRIPTS

- To run a script, feed the file to the appropriate interpreter:
  - `bash mybashscript`
  - `perl myperlscript`
- This works fine, but sometimes it's more user-friendly to allow the script to be run directly, removing the need for an external call to the interpreter...
  - `./mybashscript`
  - `myperlscript`



# SHEBANG

- This is accomplished with the shebang ( #! ).  Also known as a hash bang, pound bang or hashpling.
- When the kernel is asked to execute a file, it must either be machine code, or a file that starts with the shebang sequence.  If the first two characters of the file are a hash mark and an exclamation mark, the rest of the line is expected to be an absolute pathname for an interpreter, which will then be invoked to “run” the file as a script.



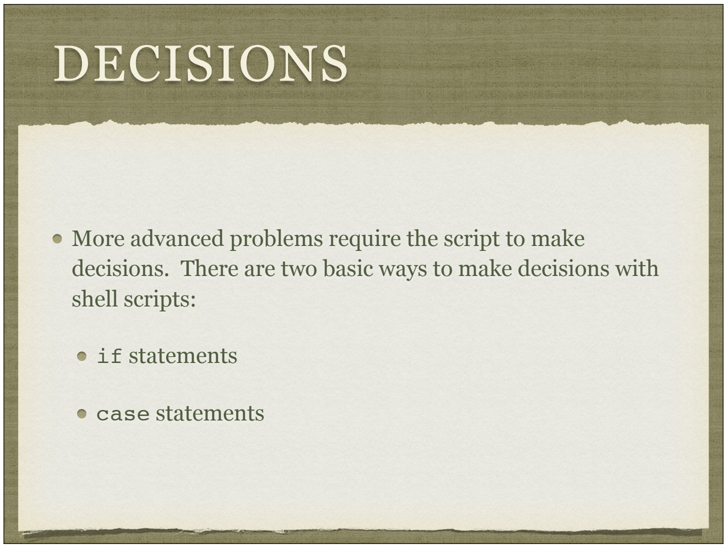
# SHEBANG

- So, add an appropriate shebang to the example:

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
echo “Hello $NAME, it’s nice to meet you!”
echo -n “The current time is: “
date
```

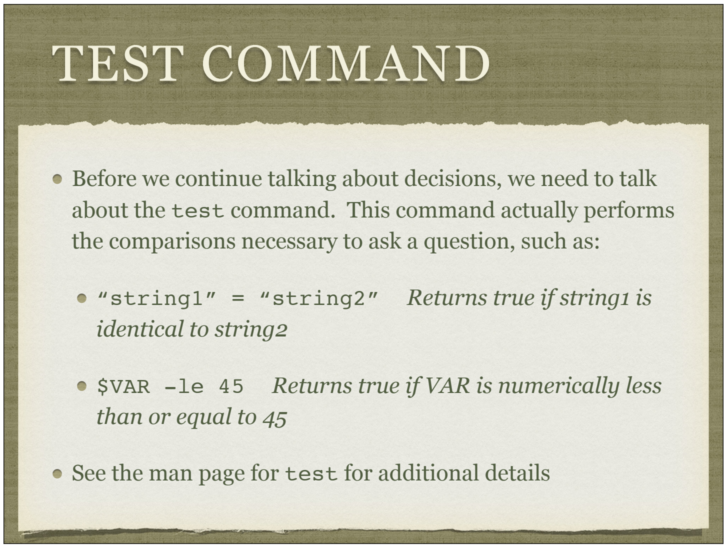
- Then add execute permissions and the script can be run directly:

```
[root@localhost ~]# chmod 755 myscript
[root@localhost ~]# ./myscript
Hello, what is your name?
Linus
Hello Linus, it’s nice to meet you!
The current time is: Sun Nov 29 09:39:33 CST 2009
[root@localhost ~]#
```



# DECISIONS

- More advanced problems require the script to make decisions. There are two basic ways to make decisions with shell scripts:
  - `if` statements
  - `case` statements



# TEST COMMAND

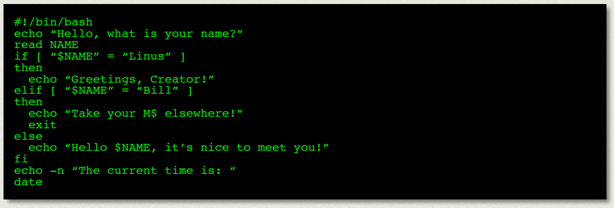
- Before we continue talking about decisions, we need to talk about the `test` command. This command actually performs the comparisons necessary to ask a question, such as:
  - `“string1” = “string2”` *Returns true if string1 is identical to string2*
  - `$VAR -le 45` *Returns true if VAR is numerically less than or equal to 45*
- See the man page for `test` for additional details

# IF STATEMENTS

- Basic syntax:

```
if list;
  then list;
  [ elif list; then list; ]
  ...
  [ else list; ]
fi
```

# IF EXAMPLE



```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
if [ “$NAME” = “Linus” ]
then
  echo “Greetings, Creator!”
elif [ “$NAME” = “Bill” ]
then
  echo “Take your M$ elsewhere!”
  exit
else
  echo “Hello $NAME, it’s nice to meet you!”
fi
echo -n “The current time is: “
date
```

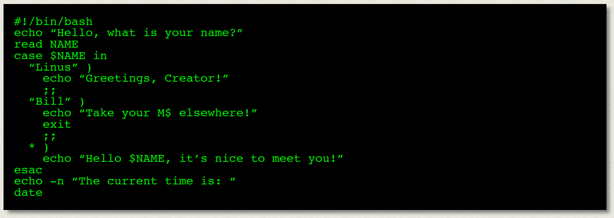
- This script will now base it’s response based on what name the user provides

# CASE STATEMENTS

- Basic syntax:

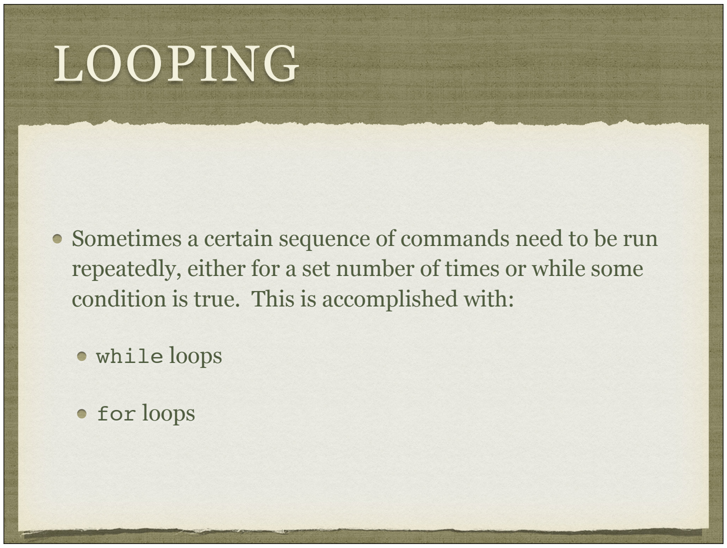
```
case word in

  pattern [| pattern] ) list;;

  ...

esac
```

# CASE EXAMPLE



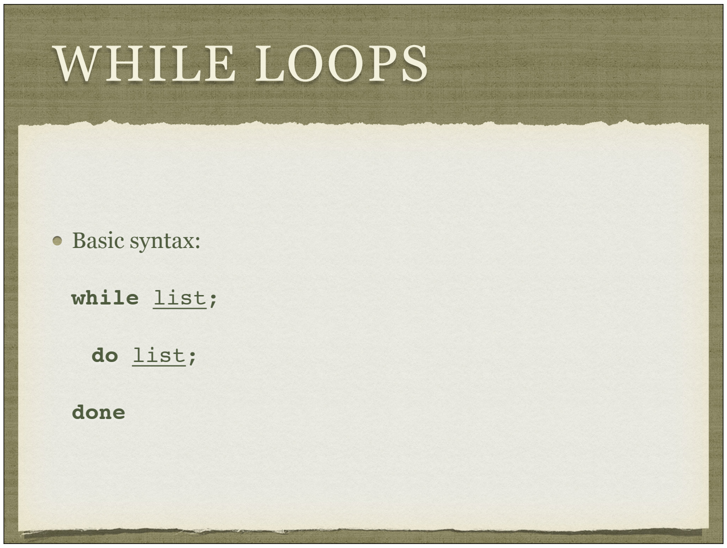
```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
case $NAME in
  “Linus” )
    echo “Greetings, Creator!”
    ;;
  “Bill” )
    echo “Take your M$ elsewhere!”
    exit
    ;;
  * )
    echo “Hello $NAME, it’s nice to meet you!”
esac
echo -n “The current time is: “
date
```

- This script also bases it’s response based on what name the user provides, but does so using a case statement instead of a large if statement



# LOOPING

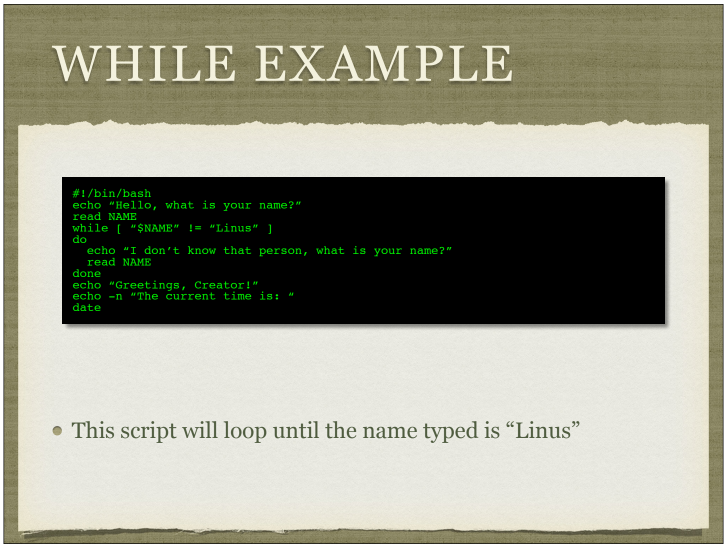
- Sometimes a certain sequence of commands need to be run repeatedly, either for a set number of times or while some condition is true. This is accomplished with:
  - `while` loops
  - `for` loops



# WHILE LOOPS

- Basic syntax:

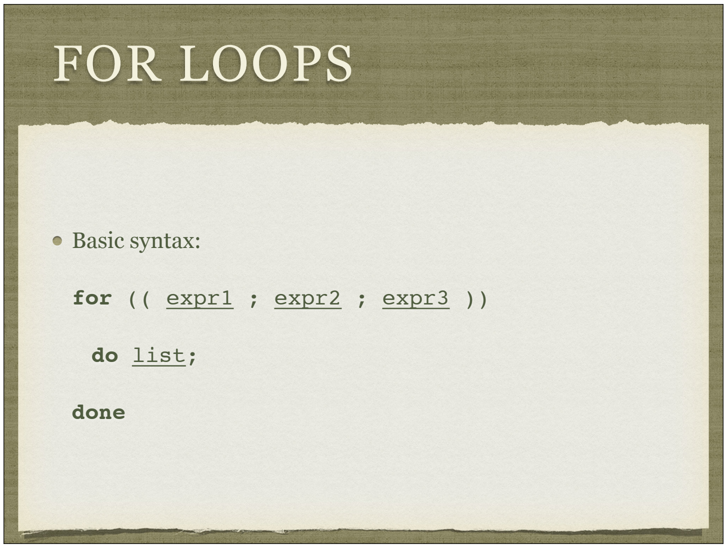
```
while list;
  do list;
done
```



# WHILE EXAMPLE

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
while [ “$NAME” != “Linus” ]
do
  echo “I don’t know that person, what is your name?”
  read NAME
done
echo “Greetings, Creator!”
echo -n “The current time is: “
date
```

- This script will loop until the name typed is “Linus”

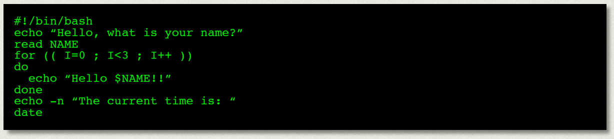


# FOR LOOPS

- Basic syntax:

```
for (( expr1 ; expr2 ; expr3 ))
  do list;
done
```

# FOR EXAMPLE

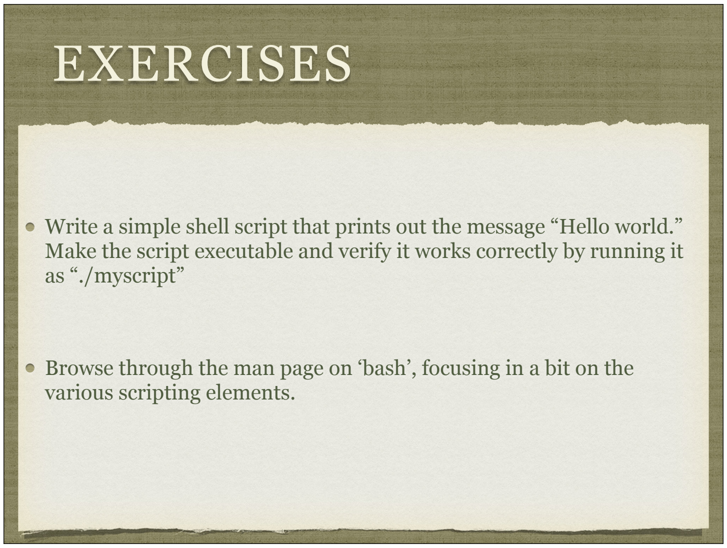


```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
for (( I=0 ; I<3 ; I++ ))
do
  echo “Hello $NAME!!”
done
echo -n “The current time is: “
date
```

- This excitable script repeats your name 3 times before giving you the date and time

# SCRIPTING

- There is of course quite a bit more to shell scripting than can be covered in this course.  There are a few more structures you can use for looping, and dozens of special metacharacters for achieving all kinds of results.
- With this introduction, though, you should be able to read through light shell scripts and have a handle on what’s going on, as well as be able to write simple ones on your own.

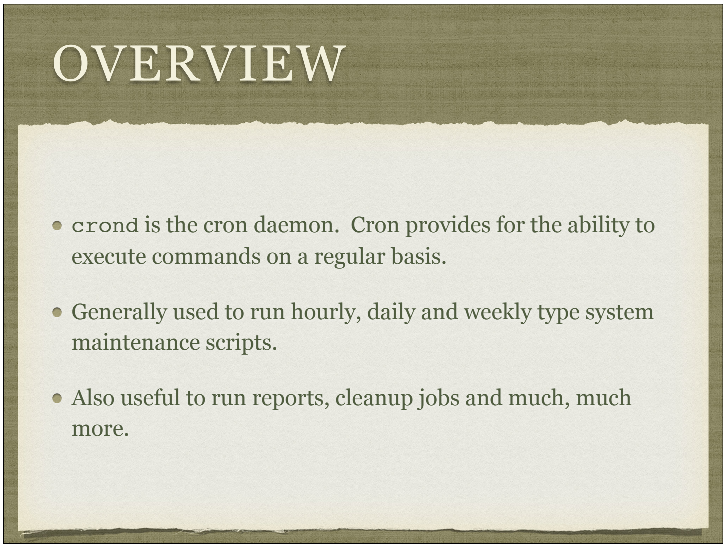


# EXERCISES

- Write a simple shell script that prints out the message “Hello world.” Make the script executable and verify it works correctly by running it as “./myscript”
- Browse through the man page on ‘bash’, focusing in a bit on the various scripting elements.

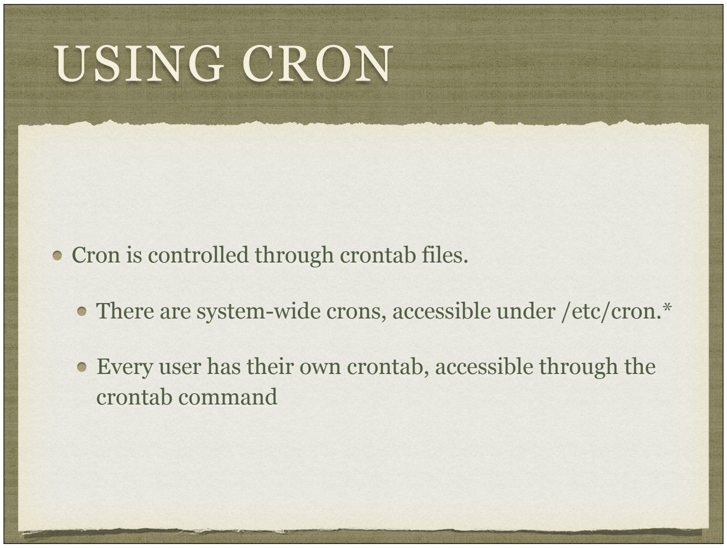
# CROND

Scheduled fun



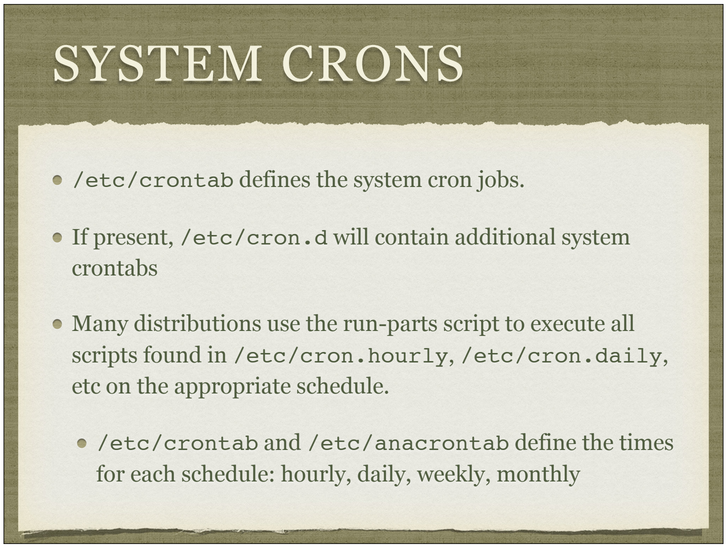
## OVERVIEW

- `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.



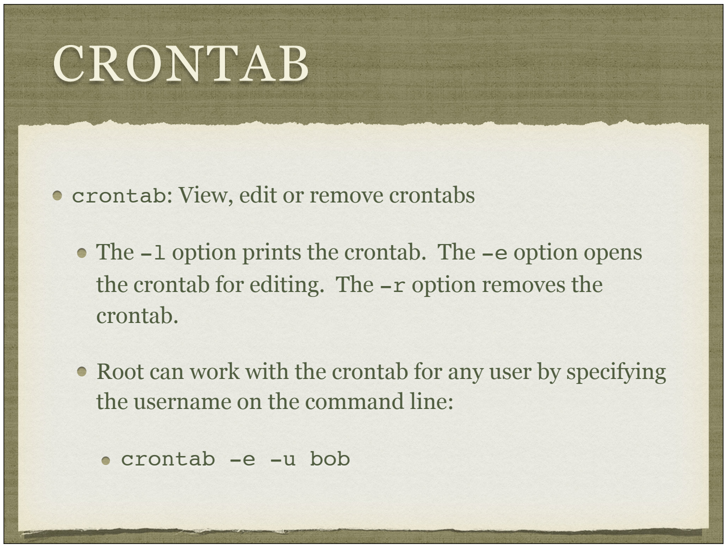
## USING CRON

- 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.
- If present, `/etc/cron.d` will contain additional system crontabs
- 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` and `/etc/anacrontab` define the times for each schedule: hourly, daily, weekly, monthly



# 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

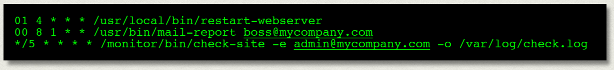
# CRONTAB TIMESPECS

- 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

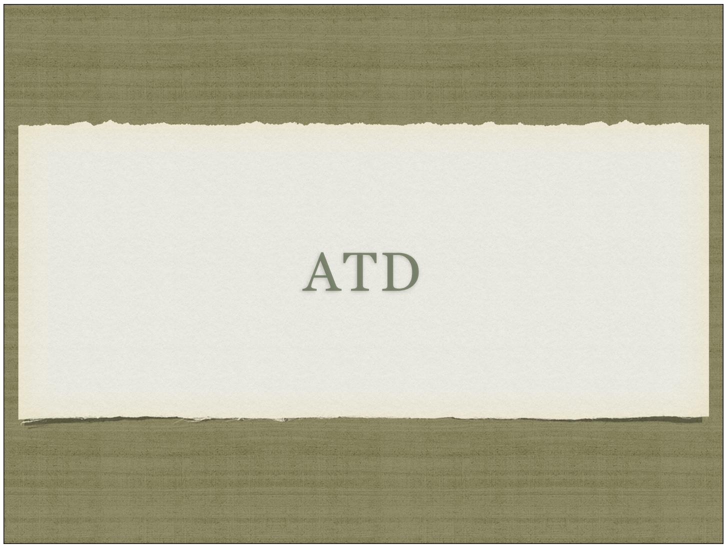
| | |
|---|---|
| `0 23 * * *` | *11pm every day* |
| `30 * * * 1-5` | *30 minutes past every hour, M-F* |
| `0 7 1 * *` | *7am, first of every month* |
| `* * * * *` | *Every single minute* |
| `0,10,20,30,40,50 * * * *` | *Every 10 minutes* |
| `*/5 8-17 * * 1-5` | *Every 5 minutes, 8am-5pm, M-F* |

# EXAMPLE CRONTAB

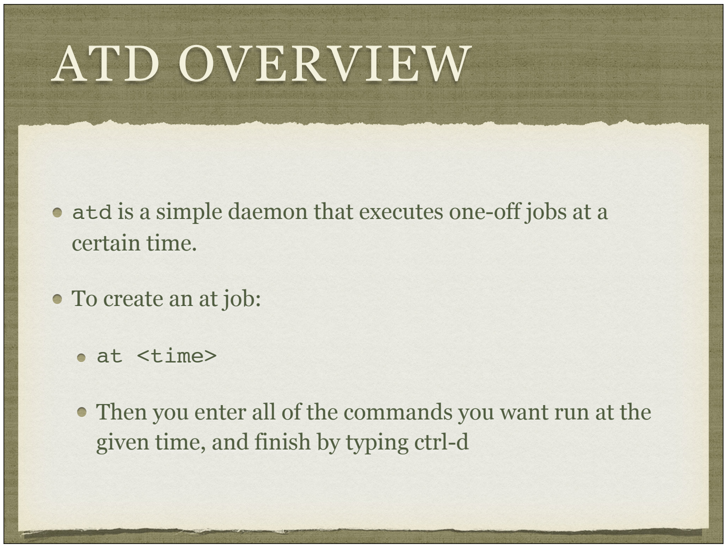


```
01 4 * * * /usr/local/bin/restart-webserver
00 8 1 * * /usr/bin/mail-report boss@mycompany.com
*/5 * * * * /monitor/bin/check-site -e admin@mycompany.com -o /var/log/check.log
```

- 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

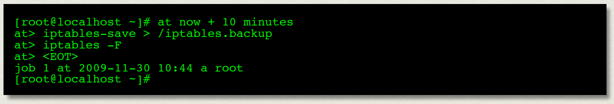


## ATD OVERVIEW

- `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:



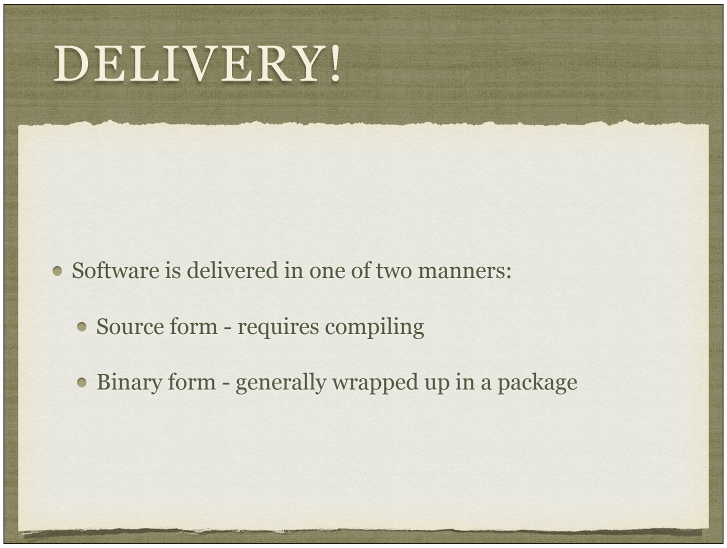
```
[root@localhost ~]# at now + 10 minutes
at> iptables-save > /iptables.backup
at> iptables -F
at> <EOT>
job 1 at 2009-11-30 10:44 a root
[root@localhost ~]#
```

# ATD

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

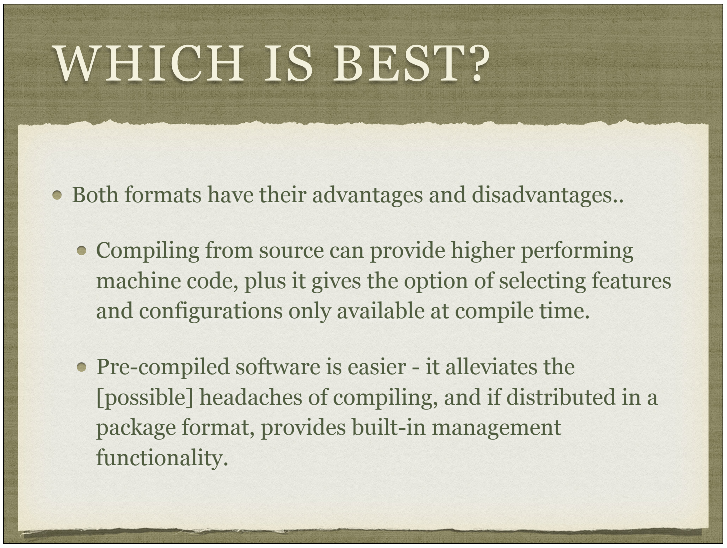
# SOFTWARE INSTALLATION

Gotta have it



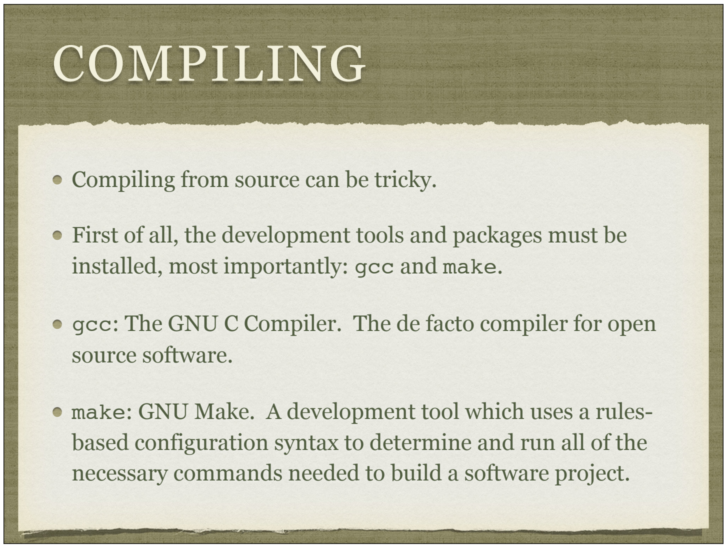
## DELIVERY!

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



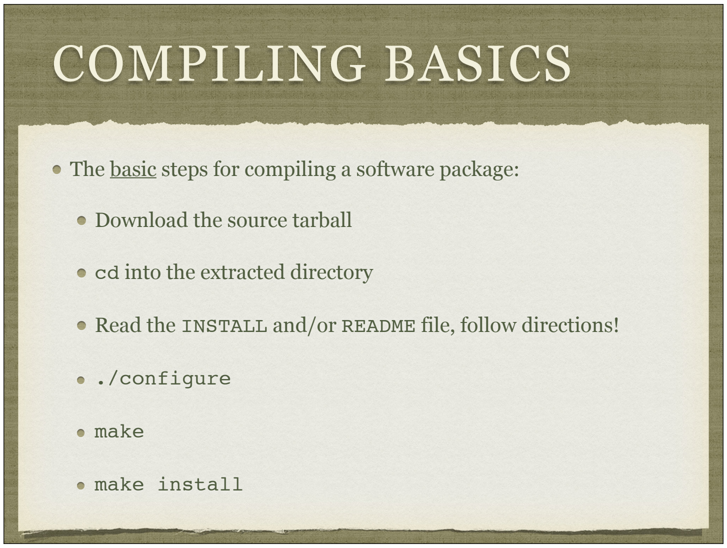
## WHICH IS BEST?

- 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.



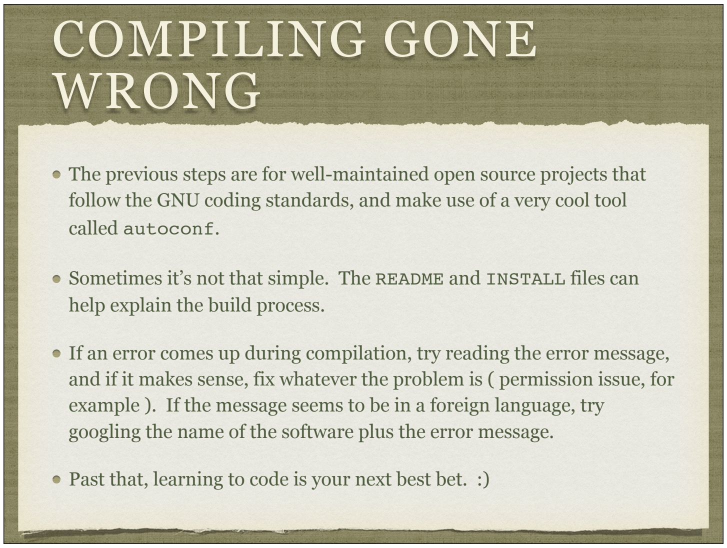
## COMPILING

- Compiling from source can be tricky.
- First of all, the development tools and packages must be installed, most importantly: `gcc` and `make`.
- `gcc`: The GNU C Compiler. The de facto compiler for open source software.
- `make`: GNU Make. A development tool which uses a rules-based configuration syntax to determine and run all of the necessary commands needed to build a software project.



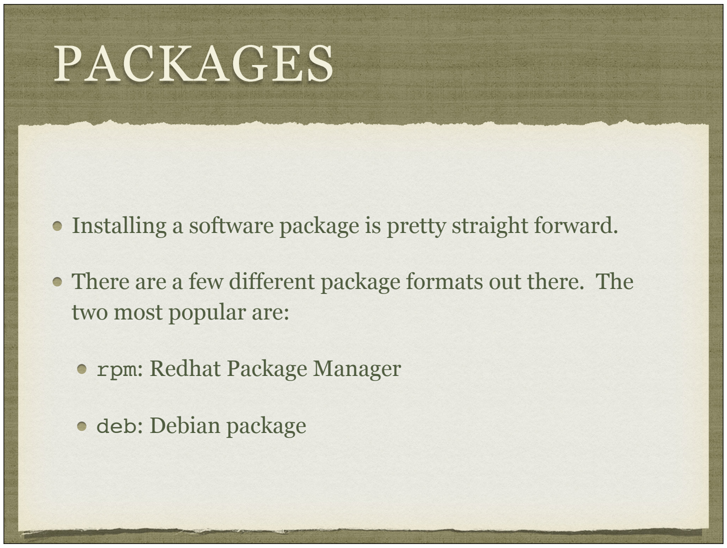
# COMPILING BASICS

- The basic steps for compiling a software package:
  - Download the source tarball
  - `cd` into the extracted directory
  - Read the `INSTALL` and/or `README` file, follow directions!
  - `./configure`
  - `make`
  - `make install`



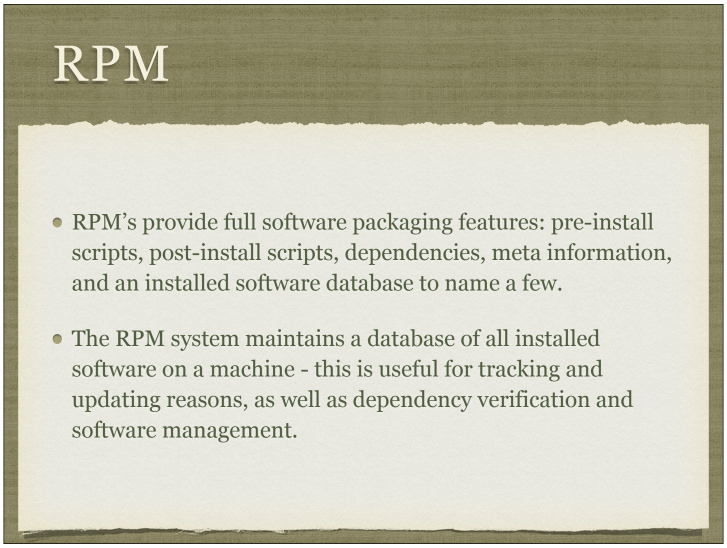
# COMPILING GONE WRONG

- The previous steps are for well-maintained open source projects that follow the GNU coding standards, and make use of a very cool tool called `autoconf`.
- Sometimes it's not that simple. The `README` and `INSTALL` files can help explain the build process.
- If an error comes up during compilation, try reading the error message, and if it makes sense, fix whatever the problem is ( permission issue, for example ). If the message seems to be in a foreign language, try googling the name of the software plus the error message.
- Past that, learning to code is your next best bet. :)



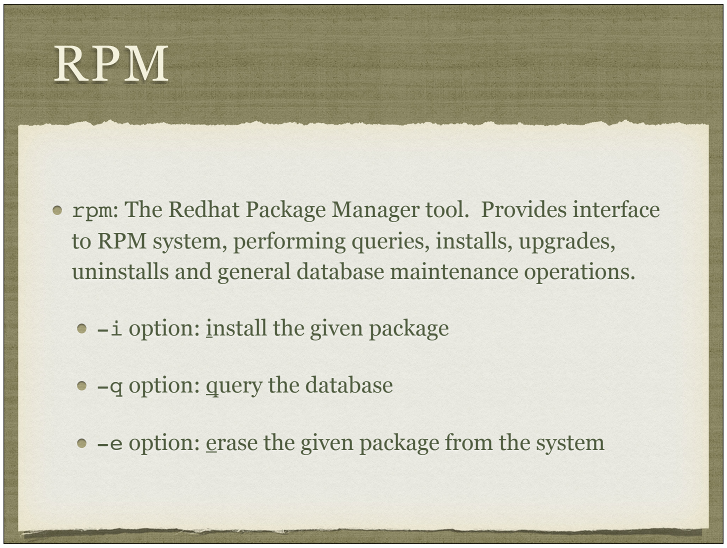
# PACKAGES

- 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



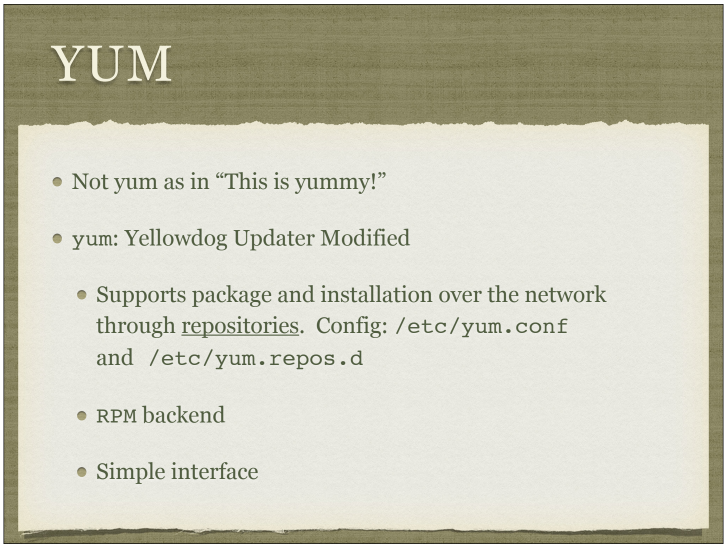
# RPM

- 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.



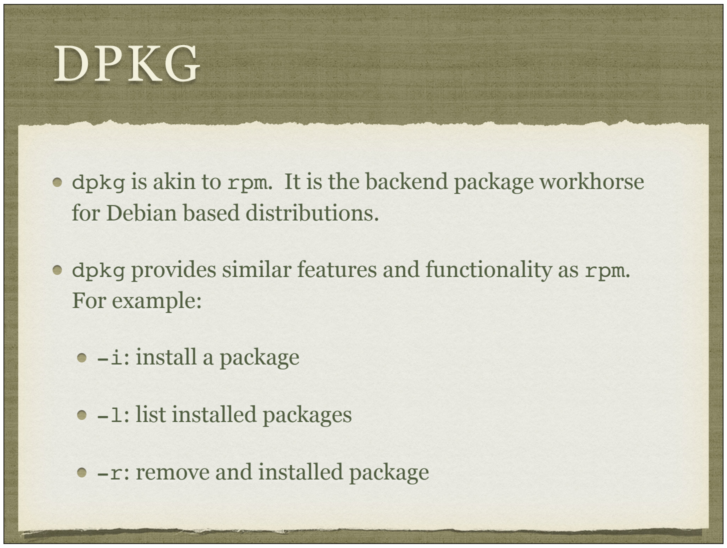
# RPM

- `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



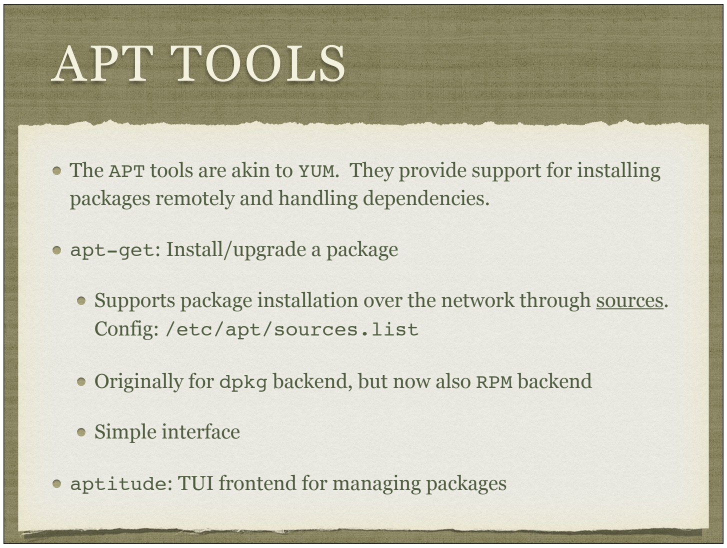
# YUM

- Not yum as in “This is yummy!”
- `yum`: Yellowdog Updater Modified
    - Supports package and installation over the network through repositories. Config: `/etc/yum.conf` and `/etc/yum.repos.d`
    - `RPM` backend
    - Simple interface



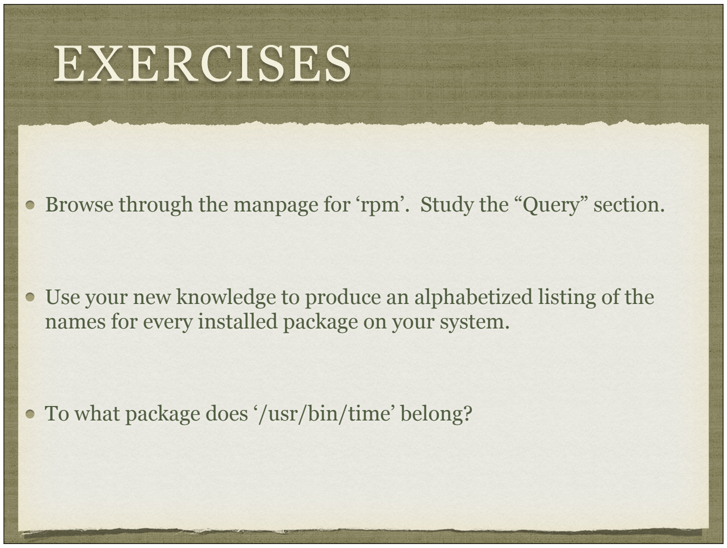
# DPKG

- `dpkg` is akin to `rpm`. It is the backend package workhorse for Debian based distributions.
- `dpkg` provides similar features and functionality as `rpm`. For example:
  - `-i`: install a package
  - `-l`: list installed packages
  - `-r`: remove and installed package



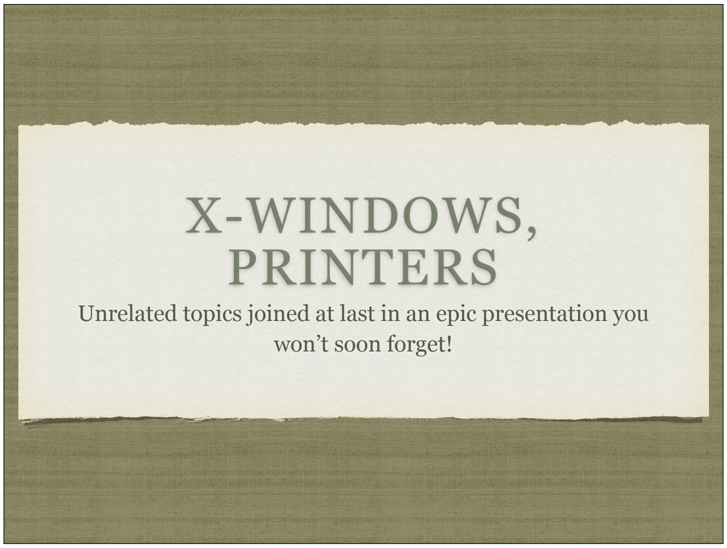
# APT TOOLS

- The `APT` tools are akin to `YUM`. They provide support for installing packages remotely and handling dependencies.
- `apt-get`: Install/upgrade a package
  - Supports package installation over the network through <u>sources</u>. Config: `/etc/apt/sources.list`
  - Originally for `dpkg` backend, but now also `RPM` backend
  - Simple interface
- `aptitude`: TUI frontend for managing packages



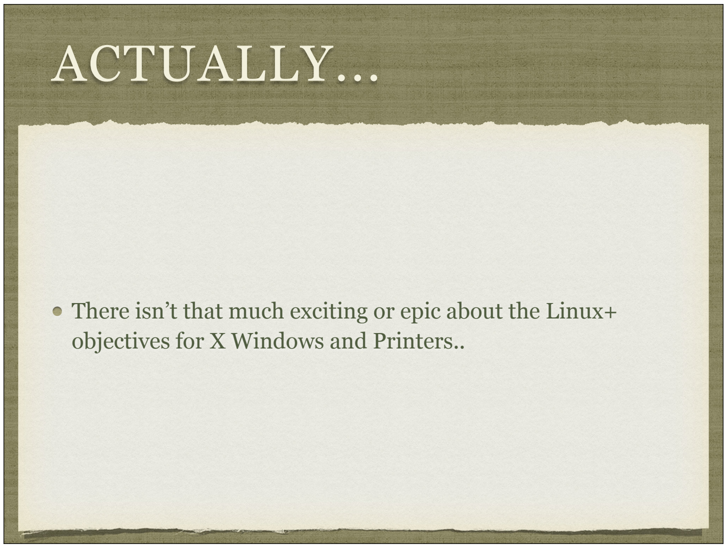
# EXERCISES

- 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?



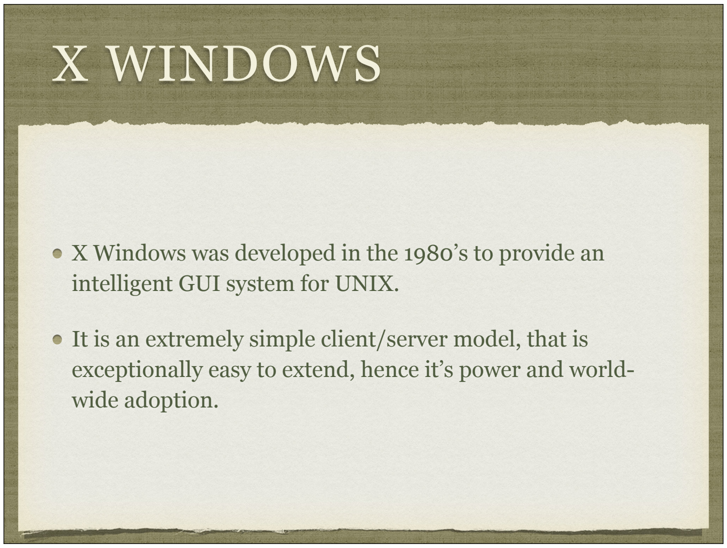
# X-WINDOWS, PRINTERS

Unrelated topics joined at last in an epic presentation you won’t soon forget!



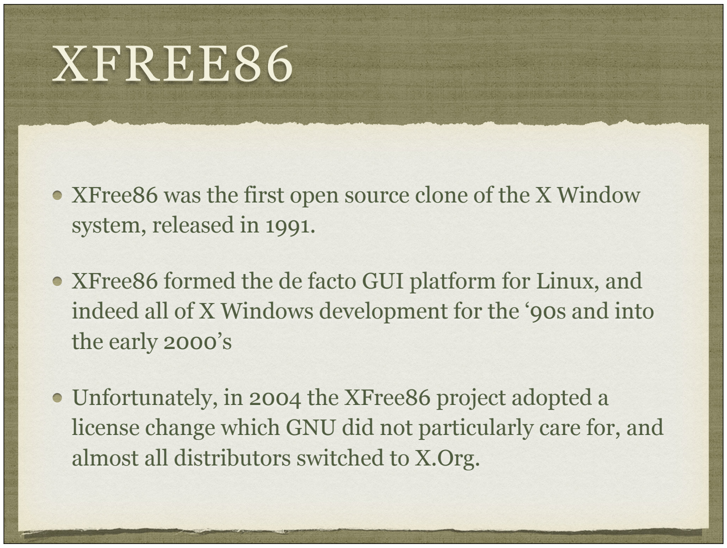
## ACTUALLY…

- There isn’t that much exciting or epic about the Linux+ objectives for X Windows and Printers..



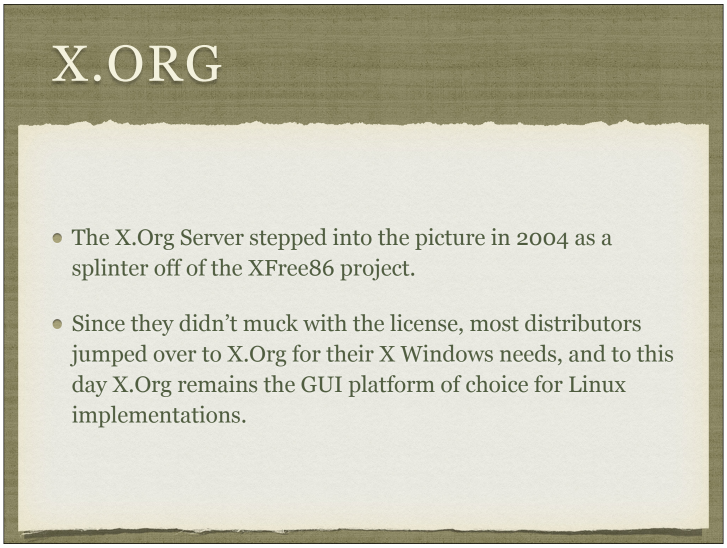
# X WINDOWS

- X Windows was developed in the 1980's to provide an intelligent GUI system for UNIX.
- It is an extremely simple client/server model, that is exceptionally easy to extend, hence it's power and world-wide adoption.



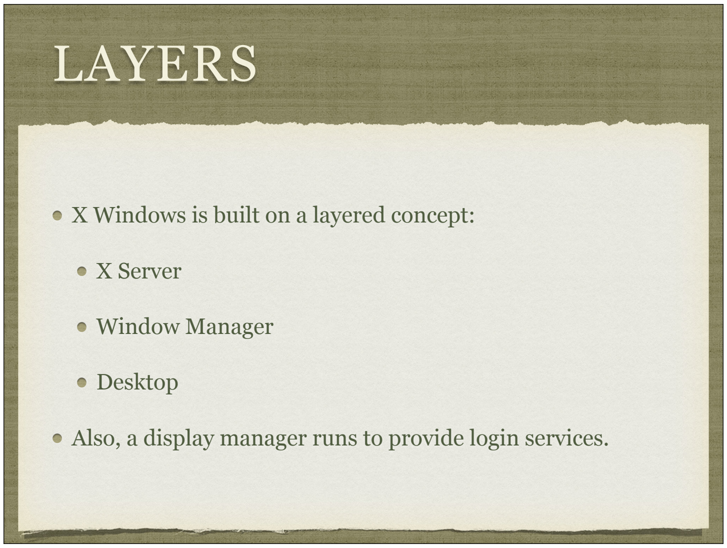
# XFREE86

- XFree86 was the first open source clone of the X Window system, released in 1991.
- XFree86 formed the de facto GUI platform for Linux, and indeed all of X Windows development for the '90s and into the early 2000's
- Unfortunately, in 2004 the XFree86 project adopted a license change which GNU did not particularly care for, and almost all distributors switched to X.Org.



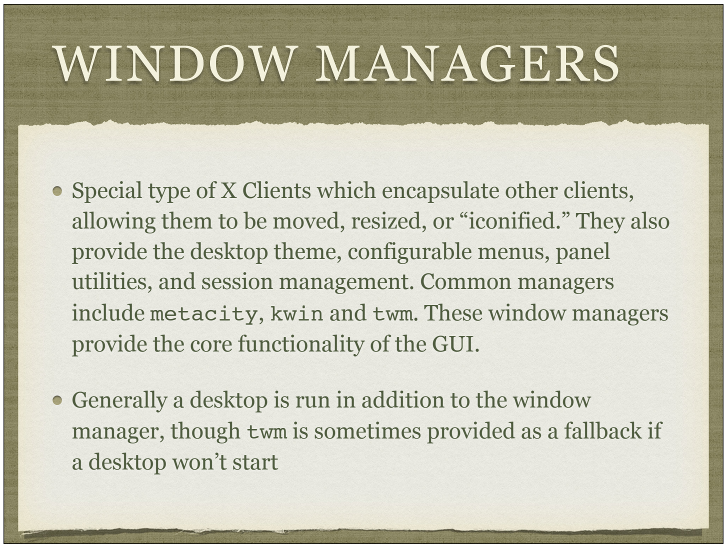
# X.ORG

- The X.Org Server stepped into the picture in 2004 as a splinter off of the XFree86 project.
- Since they didn't muck with the license, most distributors jumped over to X.Org for their X Windows needs, and to this day X.Org remains the GUI platform of choice for Linux implementations.



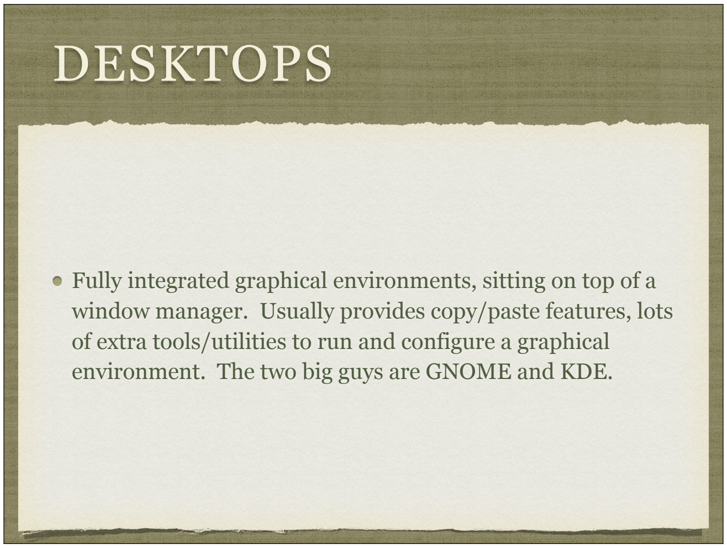
# LAYERS

- X Windows is built on a layered concept:
  - X Server
  - Window Manager
  - Desktop
- Also, a display manager runs to provide login services.



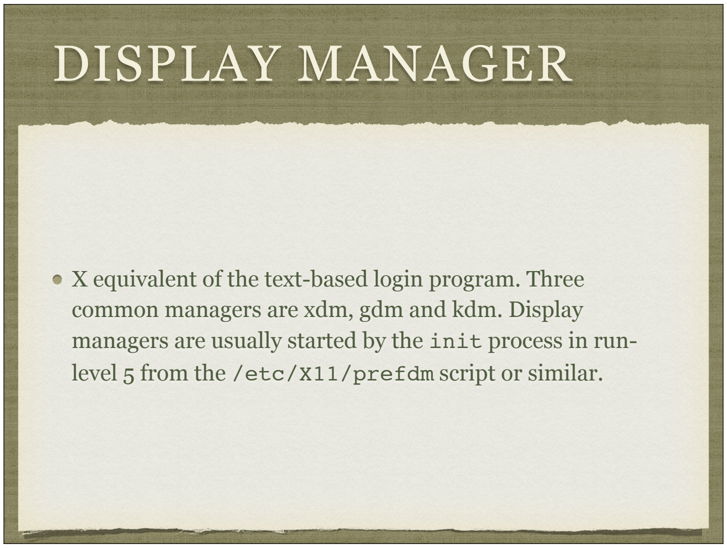
# WINDOW MANAGERS

- Special type of X Clients which encapsulate other clients, allowing them to be moved, resized, or “iconified.” They also provide the desktop theme, configurable menus, panel utilities, and session management. Common managers include `metacity`, `kwin` and `twm`. These window managers provide the core functionality of the GUI.
- Generally a desktop is run in addition to the window manager, though `twm` is sometimes provided as a fallback if a desktop won’t start



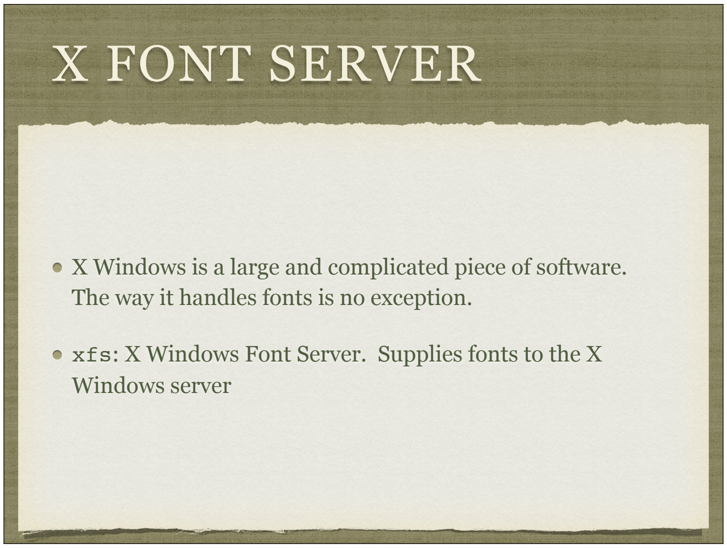
# DESKTOPS

- Fully integrated graphical environments, sitting on top of a window manager. Usually provides copy/paste features, lots of extra tools/utilities to run and configure a graphical environment. The two big guys are GNOME and KDE.



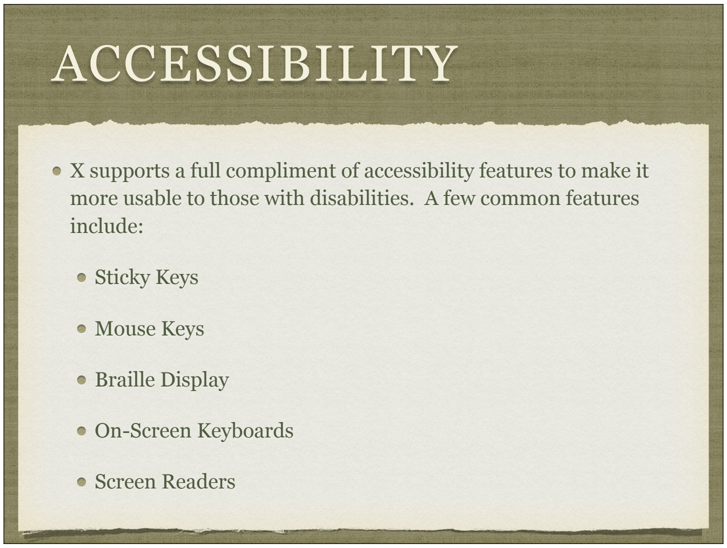
## DISPLAY MANAGER

- X equivalent of the text-based login program. Three common managers are xdm, gdm and kdm. Display managers are usually started by the `init` process in run-level 5 from the `/etc/X11/prefdm` script or similar.



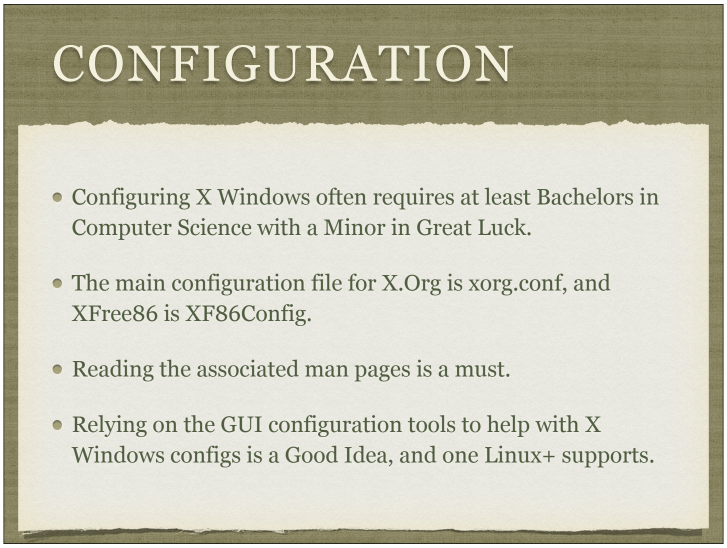
## X FONT SERVER

- X Windows is a large and complicated piece of software. The way it handles fonts is no exception.
- `xfs`: X Windows Font Server. Supplies fonts to the X Windows server



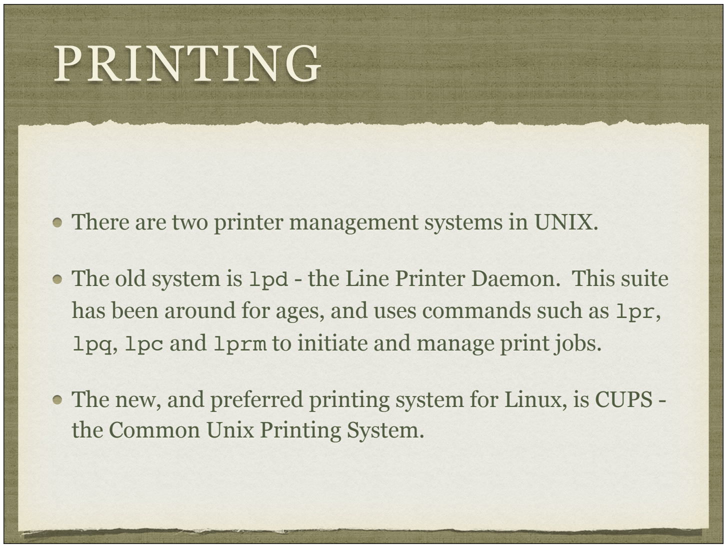
## ACCESSIBILITY

- X supports a full compliment of accessibility features to make it more usable to those with disabilities. A few common features include:
  - Sticky Keys
  - Mouse Keys
  - Braille Display
  - On-Screen Keyboards
  - Screen Readers



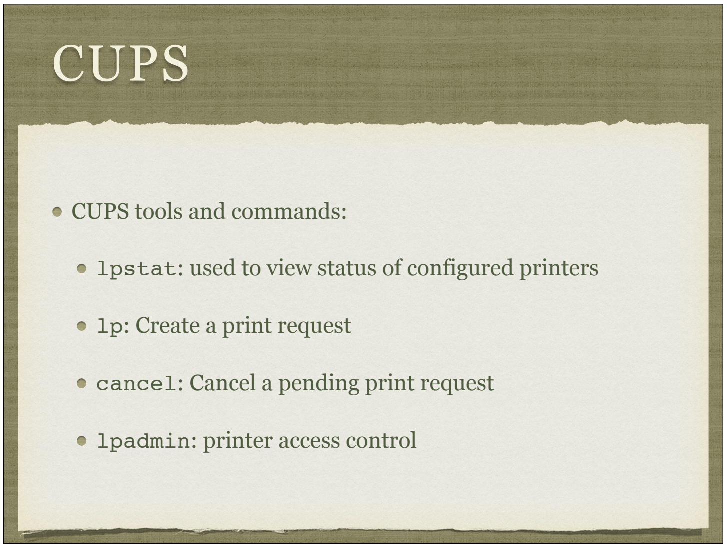
## CONFIGURATION

- Configuring X Windows often requires at least Bachelors in Computer Science with a Minor in Great Luck.
- The main configuration file for X.Org is xorg.conf, and XFree86 is XF86Config.
- Reading the associated man pages is a must.
- Relying on the GUI configuration tools to help with X Windows configs is a Good Idea, and one Linux+ supports.



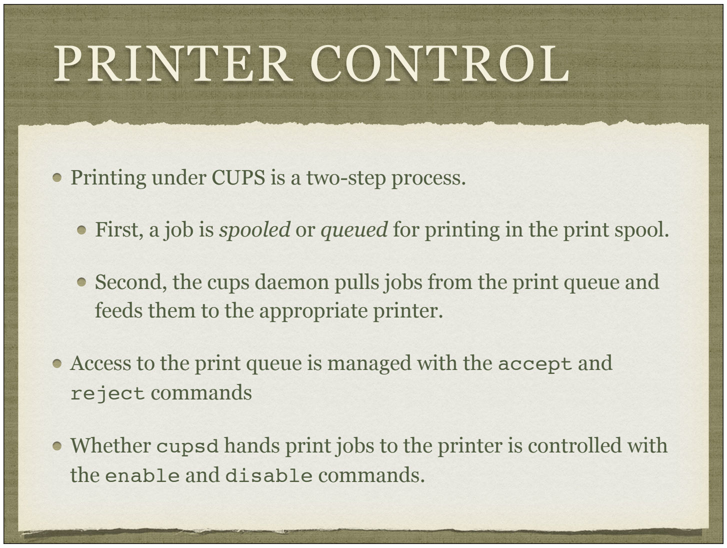
## PRINTING

- There are two printer management systems in UNIX.
- The old system is `lpd` - the Line Printer Daemon.  This suite has been around for ages, and uses commands such as `lpr`, `lpq`, `lpc` and `lprm` to initiate and manage print jobs.
- The new, and preferred printing system for Linux, is CUPS - the Common Unix Printing System.



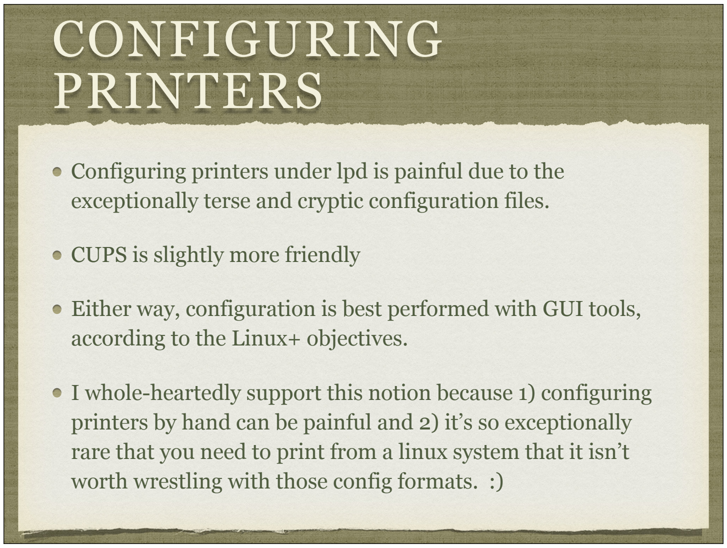
## CUPS

- CUPS tools and commands:
  - `lpstat`: used to view status of configured printers
  - `lp`: Create a print request
  - `cancel`: Cancel a pending print request
  - `lpadmin`: printer access control



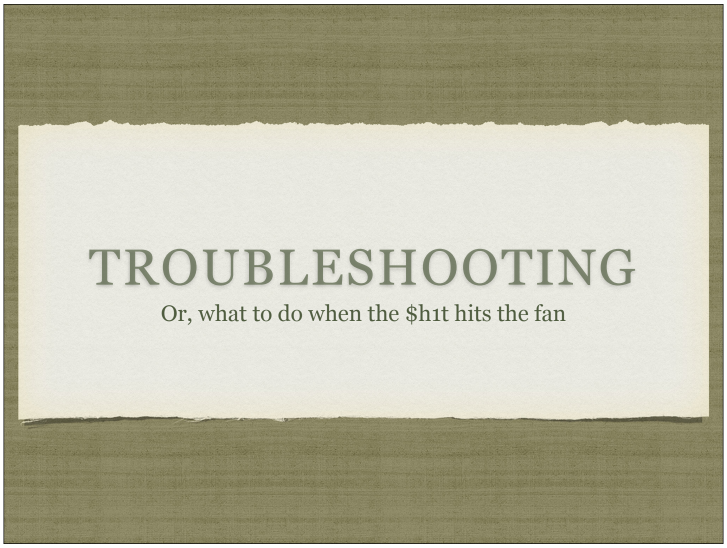
# PRINTER CONTROL

- Printing under CUPS is a two-step process.
  - First, a job is *spooled* or *queued* for printing in the print spool.
  - Second, the cups daemon pulls jobs from the print queue and feeds them to the appropriate printer.
- Access to the print queue is managed with the `accept` and `reject` commands
- Whether `cupsd` hands print jobs to the printer is controlled with the `enable` and `disable` commands.



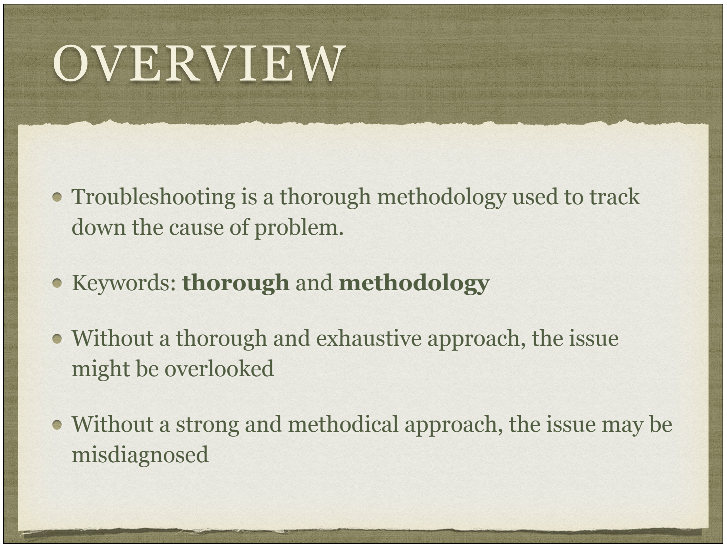
# CONFIGURING PRINTERS

- Configuring printers under lpd is painful due to the exceptionally terse and cryptic configuration files.
- CUPS is slightly more friendly
- Either way, configuration is best performed with GUI tools, according to the Linux+ objectives.
- I whole-heartedly support this notion because 1) configuring printers by hand can be painful and 2) it's so exceptionally rare that you need to print from a linux system that it isn't worth wrestling with those config formats.  :)



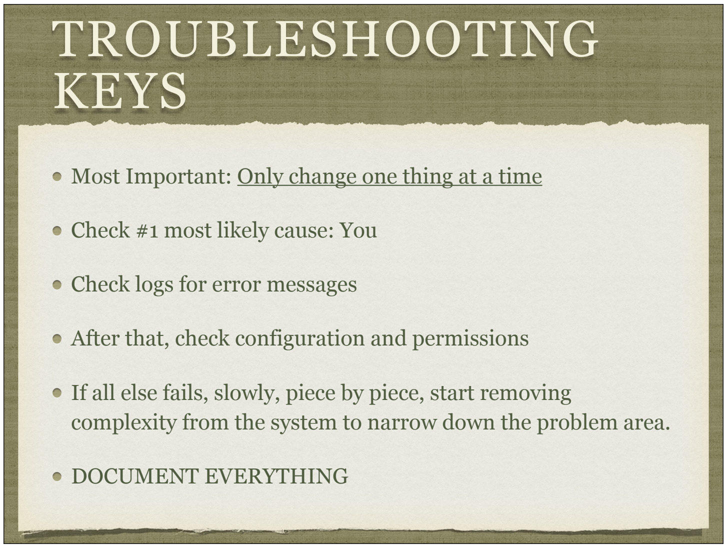
# TROUBLESHOOTING

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



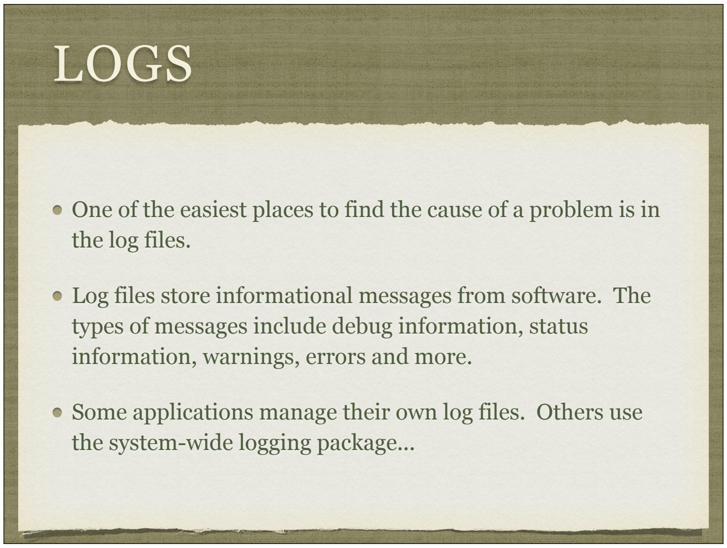
# OVERVIEW

- 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



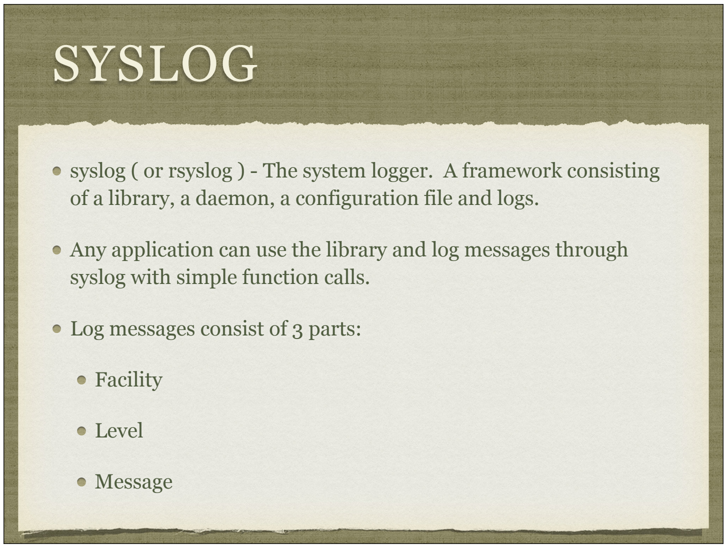
# TROUBLESHOOTING KEYS

- 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



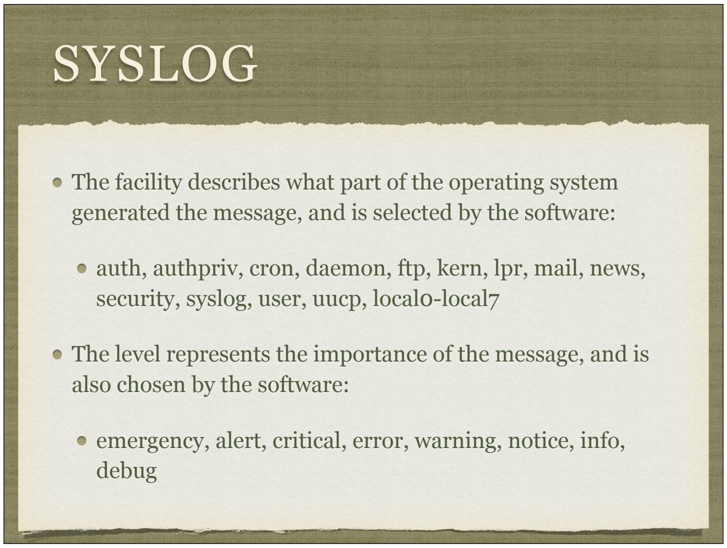
# LOGS

- 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...



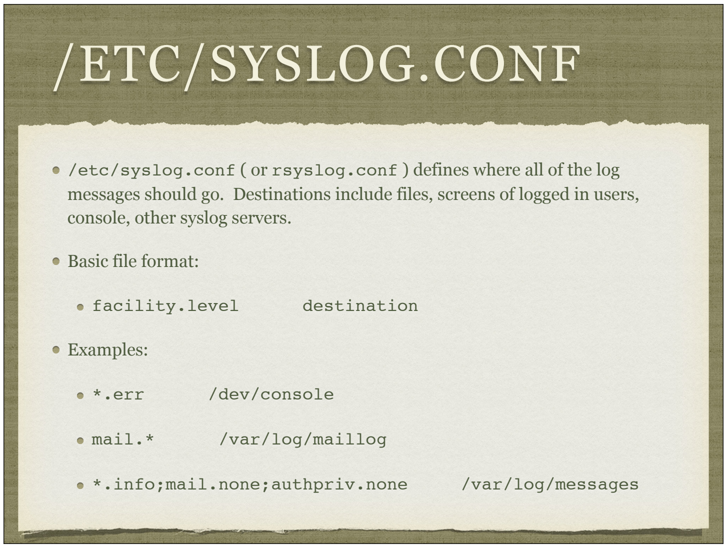
# SYSLOG

- syslog ( or rsyslog ) - 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



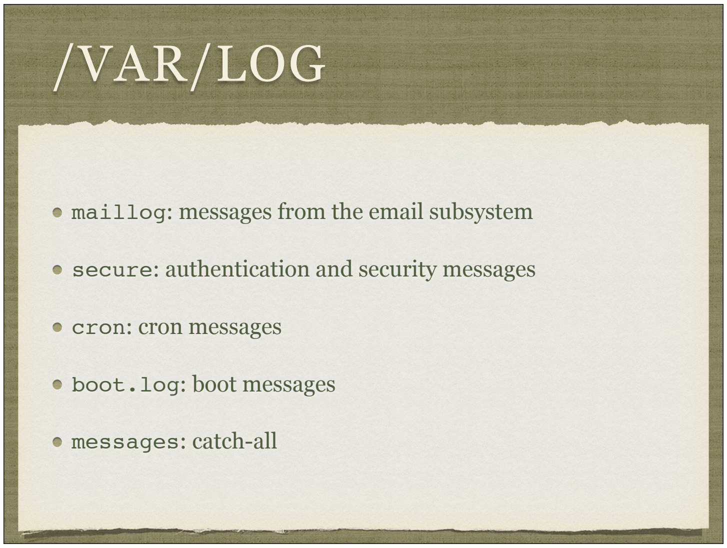
# SYSLOG

- 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



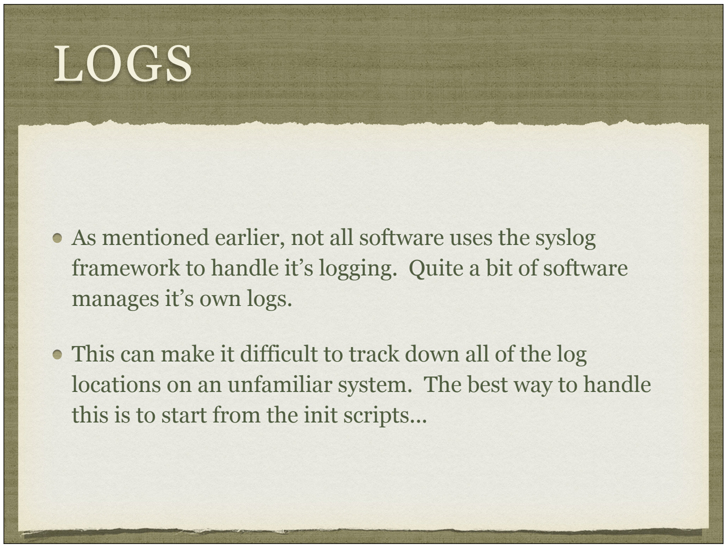
# /ETC/SYSLOG.CONF

- /etc/syslog.conf ( or rsyslog.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



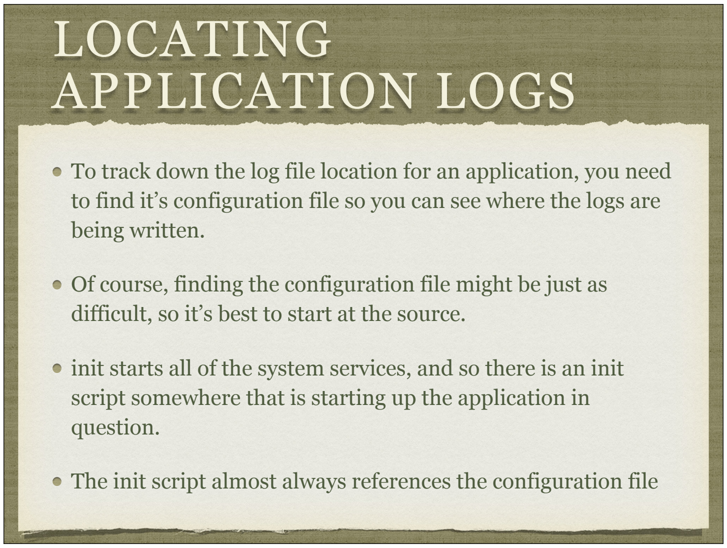
# /VAR/LOG

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



# LOGS

- As mentioned earlier, not all software uses the syslog framework to handle it's logging.  Quite a bit of software manages it's own logs.
- This can make it difficult to track down all of the log locations on an unfamiliar system.  The best way to handle this is to start from the init scripts...



# LOCATING APPLICATION LOGS

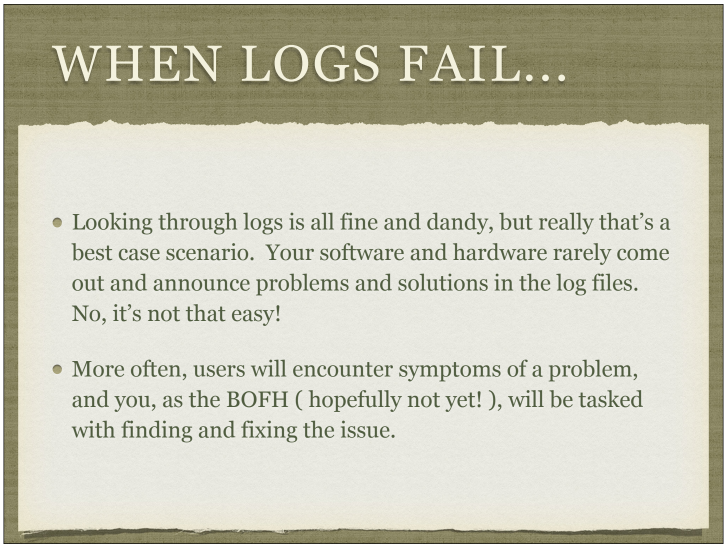
- To track down the log file location for an application, you need to find it's configuration file so you can see where the logs are being written.
- Of course, finding the configuration file might be just as difficult, so it's best to start at the source.
- init starts all of the system services, and so there is an init script somewhere that is starting up the application in question.
- The init script almost always references the configuration file

# LOCATING APPLICATION LOGS

- Now that the configuration file location is known, it only takes a few moments to scan through it and find out where logs are being written.
- As for the format of the log file, that's completely dependent on the application. Some will be similar to syslog, others, like Apache or Qmail, will be completely foreign looking.
- Fortunately, a little common sense and judicious application of Google Ointment will get the information you seek.

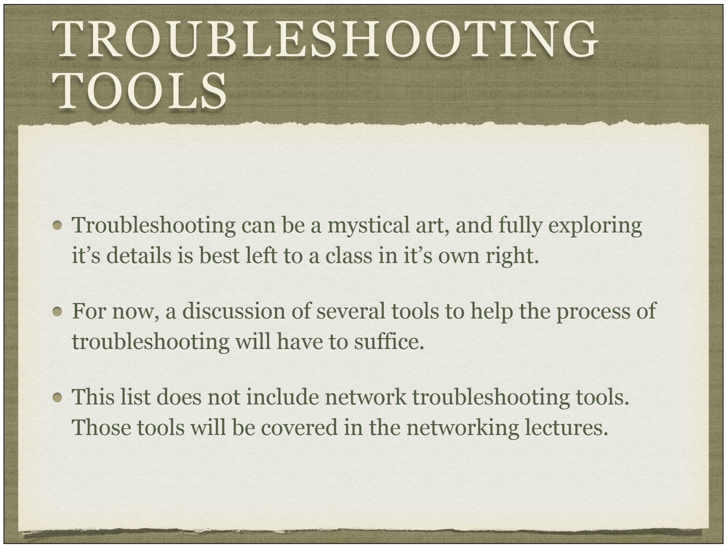
# EXERCISES

- 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.

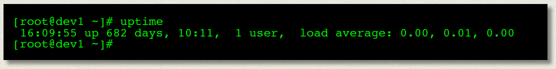


## TROUBLESHOOTING TOOLS

- 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

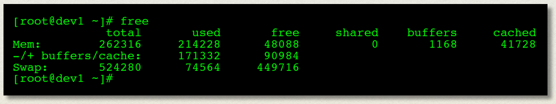
- uptime: Reports system uptime along with load averages.
  - Load Average: Average number of processes in run queue that are blocked.
  - uptime reports three values: the load averaged over the last 1 minute, 5 minutes and 15 minutes.  This is useful to get an idea of the load trend on the system.
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

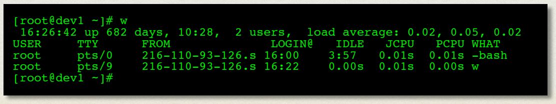
- free: reports on memory and swap usage
  - buffers: I/O buffers, directory cache
  - cached: filesystem cache ( data )
  - Example:



```
[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

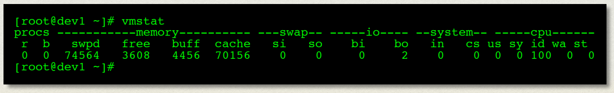
- w: Displays an uptime report, followed by a breakdown of all logged-in users and what process they are running
  - JCPU: Combined CPU time of all processes attached to the terminal ( foreground and background )
  - PCPU: CPU time of foreground process, listed in “what” column
  - Example:



```
[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

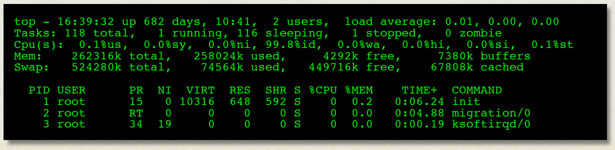
- vmstat: Snapshot report covering several primary statistics.
  - procs: number of running and blocked processes
  - swap: swapped in and swapped out blocks of memory, per second
  - io: blocks in and blocks out read/written per second
  - system: interrupts and context switches per second
  - cpu: 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

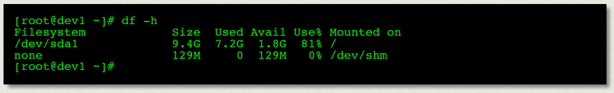
- `top`: Self-updating tool displays combination summary at top, followed by ordered list of processes.  Fully customizable.
  - The summary includes uptime information, memory breakdowns, CPU utilization and process state summaries
  - The process display can be customized and sorted to suit need



```
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

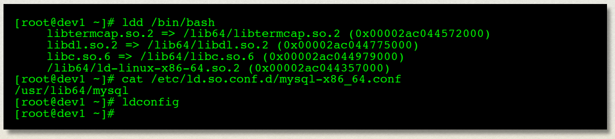
- `df`: lists filesystem utilization
  - Breaks down size and use information for each mounted filesystem
  - -h is useful option to display in “human-friendly” format



```
[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 ~]#
```

# LDD, LDCONFIG

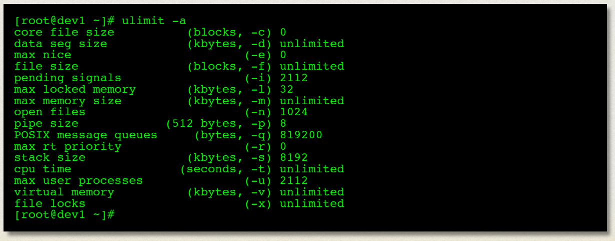
- `ldd`: List library dependencies
- `ldconfig`: Update library location database
  - `/etc/ld.so.conf` and `/etc/ld.so.conf.d/*.conf` for list of pathnames to search for libraries, creates database for dynamic linker



```
[root@dev1 ~]# ldd /bin/bash
     libtermcap.so.2 => /lib64/libtermcap.so.2 (0x00002ac044572000)
     libdl.so.2 => /lib64/libdl.so.2 (0x00002ac044775000)
     libc.so.6 => /lib64/libc.so.6 (0x00002ac044979000)
     /lib64/ld-linux-x86-64.so.2 (0x00002ac044357000)
[root@dev1 ~]# cat /etc/ld.so.conf.d/mysql-x86_64.conf
/usr/lib64/mysql
[root@dev1 ~]# ldconfig
[root@dev1 ~]#
```

# ULIMIT

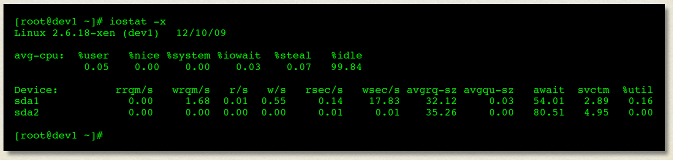
- ulimit: Sets resource limits
  - Can limit open files, memory use, cpu time, subprocesses and more.



```
[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

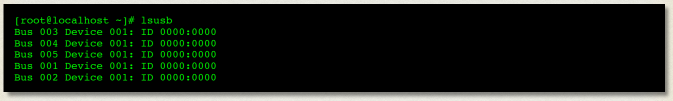
- `iostat`: IO statistics report
  - Part of the `sysstat` package; not always installed
  - Allows for drilldown into the IO system to view real time metrics on IO operations per filesystem



```
[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 ~]#
```

# LSUSB

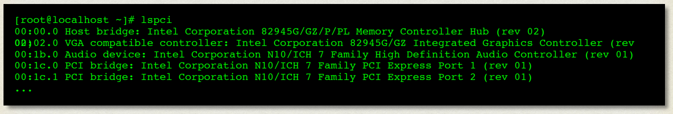
- `lsusb`: List USB bus
  - Generates a listing of devices on the USB bus
  - Consider `-v` option for *detailed* information



```
[root@localhost ~]# lsusb
Bus 003 Device 001: ID 0000:0000
Bus 004 Device 001: ID 0000:0000
Bus 005 Device 001: ID 0000:0000
Bus 001 Device 001: ID 0000:0000
Bus 002 Device 001: ID 0000:0000
```

# LSPCI

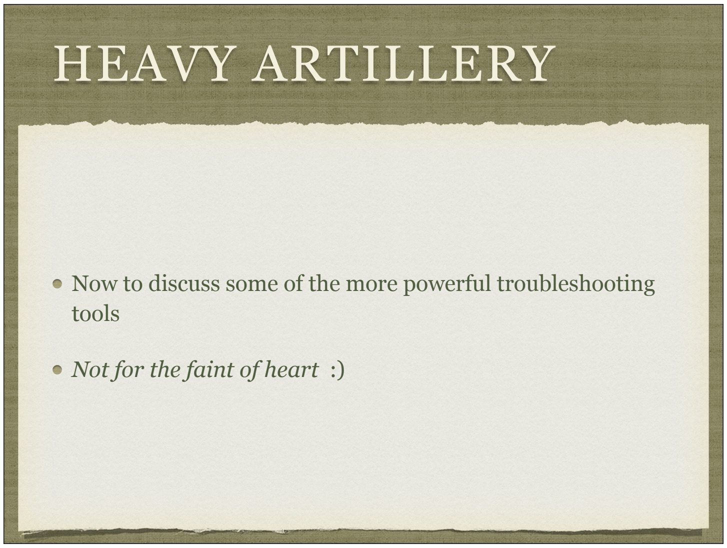
- `lspci`: List PCI bus
  - Generates a listing of devices on the PCI bus
  - Consider `-v` option for *detailed* information



```
[root@localhost ~]# lspci
00:00.0 Host bridge: Intel Corporation 82945G/GZ/P/PL Memory Controller Hub (rev 02)
00:02.0 VGA compatible controller: Intel Corporation 82945G/GZ Integrated Graphics Controller (rev
00:1b.0 Audio device: Intel Corporation N10/ICH 7 Family High Definition Audio Controller (rev 01)
00:1c.0 PCI bridge: Intel Corporation N10/ICH 7 Family PCI Express Port 1 (rev 01)
00:1c.1 PCI bridge: Intel Corporation N10/ICH 7 Family PCI Express Port 2 (rev 01)
...
```

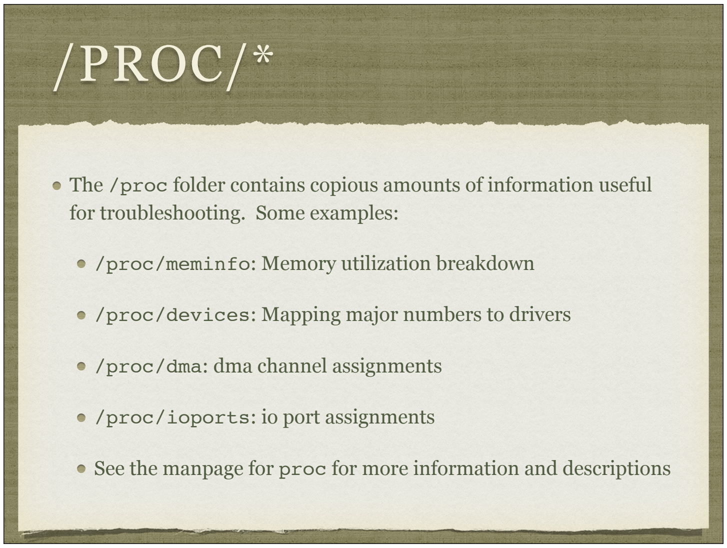
# EXERCISES

- Spend a few minutes playing with the various troubleshooting commands covered previously:
  - top, df, free, iostat, vmstat, uptime, w, ulimit



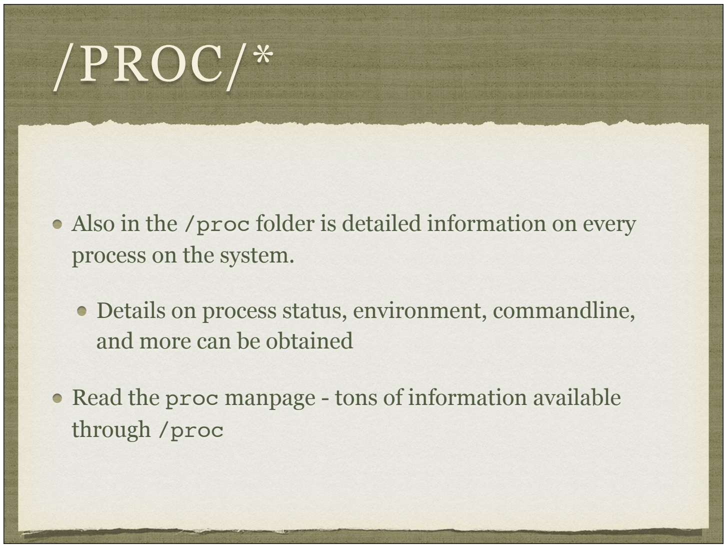
# HEAVY ARTILLERY

- Now to discuss some of the more powerful troubleshooting tools
- *Not for the faint of heart* :)



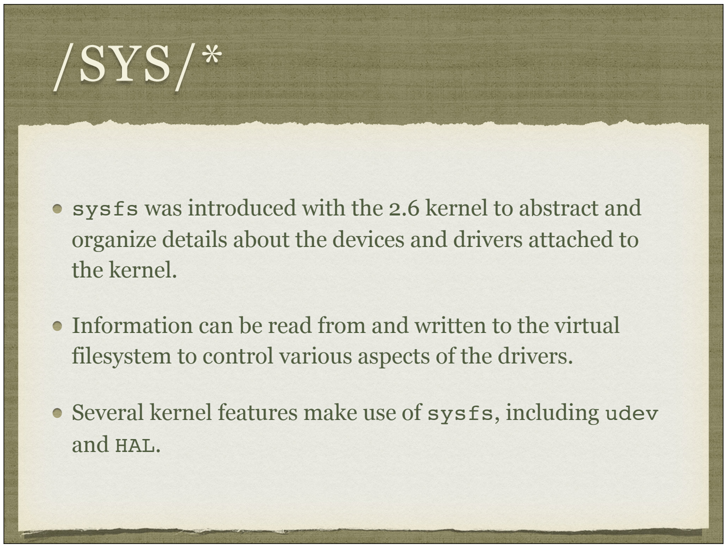
# /PROC/*

- The `/proc` folder contains copious amounts of information useful for troubleshooting. Some examples:
  - `/proc/meminfo`: Memory utilization breakdown
  - `/proc/devices`: Mapping major numbers to drivers
  - `/proc/dma`: dma channel assignments
  - `/proc/ioports`: io port assignments
  - See the manpage for `proc` for more information and descriptions



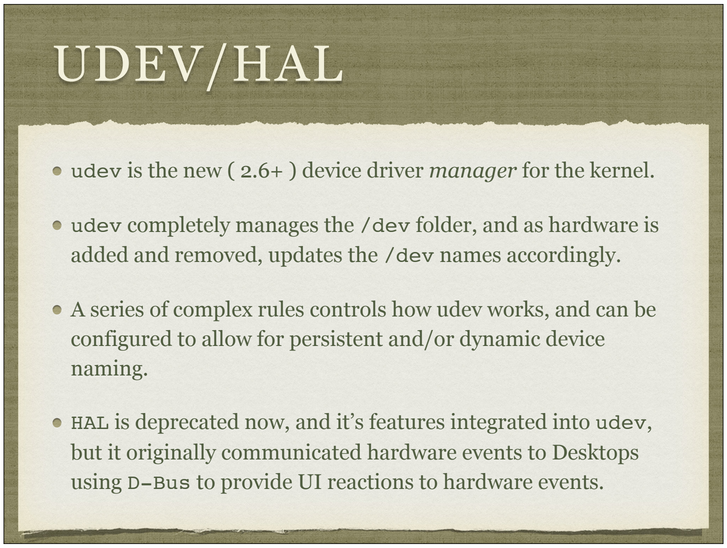
# /PROC/*

- Also in the `/proc` folder is detailed information on every process on the system.
  - Details on process status, environment, commandline, and more can be obtained
- Read the `proc` manpage - tons of information available through `/proc`



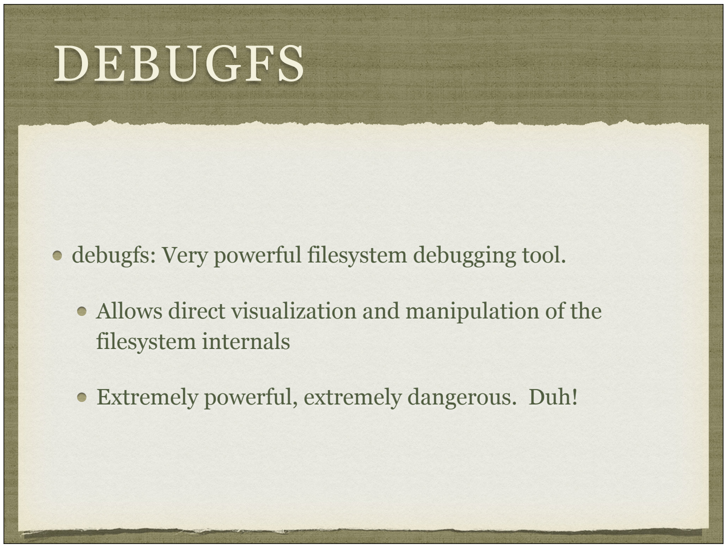
# /SYS/*

- `sysfs` was introduced with the 2.6 kernel to abstract and organize details about the devices and drivers attached to the kernel.
- Information can be read from and written to the virtual filesystem to control various aspects of the drivers.
- Several kernel features make use of `sysfs`, including `udev` and `HAL`.



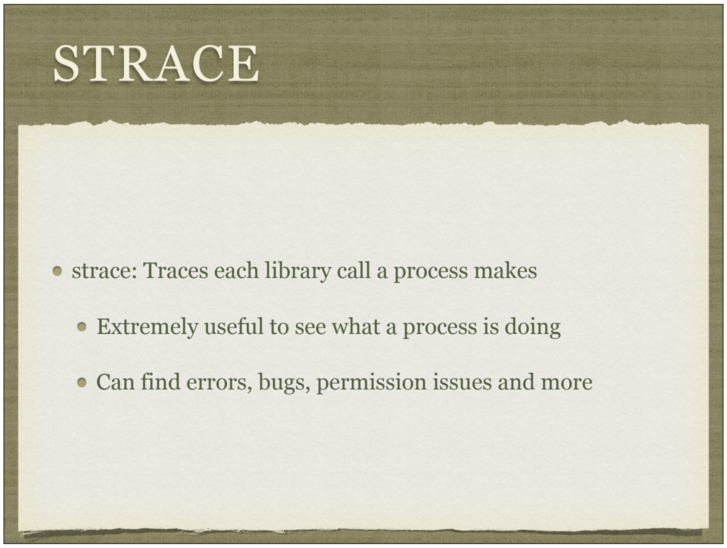
# UDEV/HAL

- `udev` is the new ( `2.6+` ) device driver *manager* for the kernel.
- `udev` completely manages the `/dev` folder, and as hardware is added and removed, updates the `/dev` names accordingly.
- A series of complex rules controls how udev works, and can be configured to allow for persistent and/or dynamic device naming.
- `HAL` is deprecated now, and it's features integrated into `udev`, but it originally communicated hardware events to Desktops using `D-Bus` to provide UI reactions to hardware events.



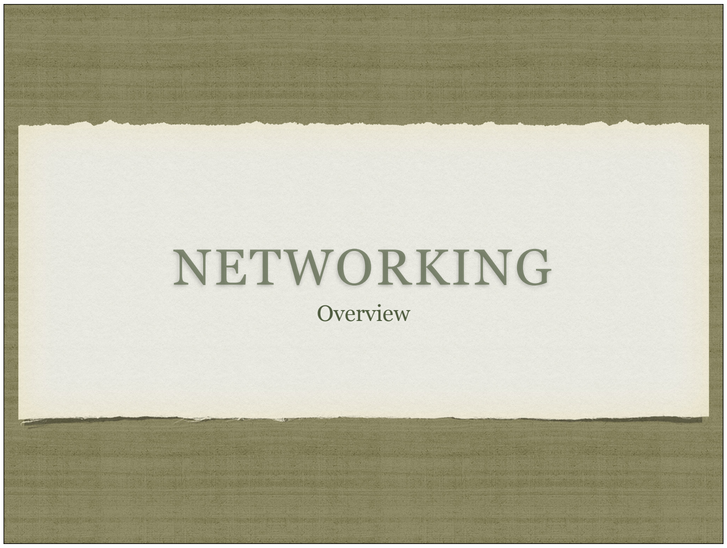
# DEBUGFS

- debugfs: Very powerful filesystem debugging tool.
  - Allows direct visualization and manipulation of the filesystem internals
  - Extremely powerful, extremely dangerous. Duh!



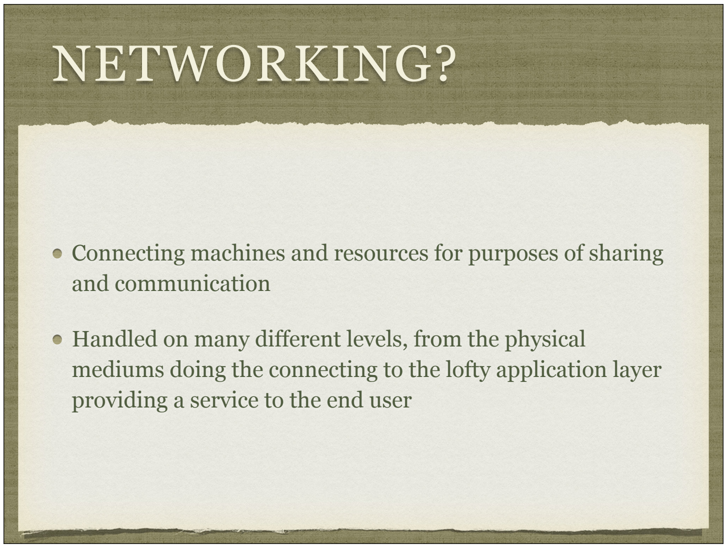
# STRACE

- strace: Traces each library call a process makes
  - Extremely useful to see what a process is doing
  - Can find errors, bugs, permission issues and more



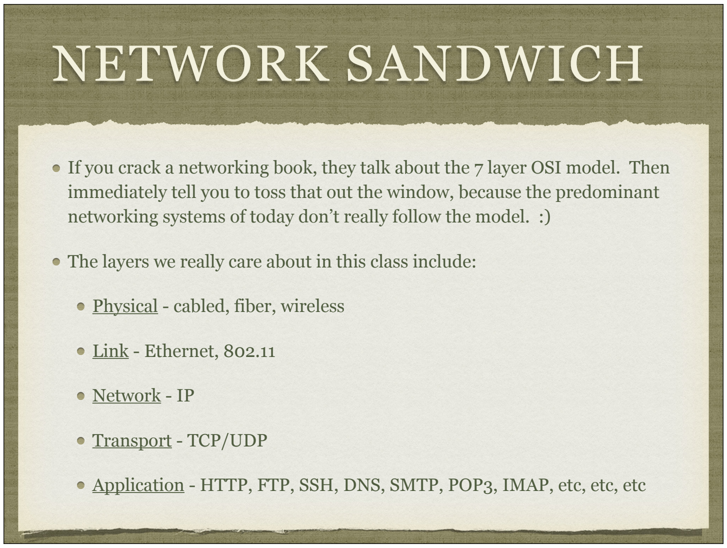
# NETWORKING

Overview



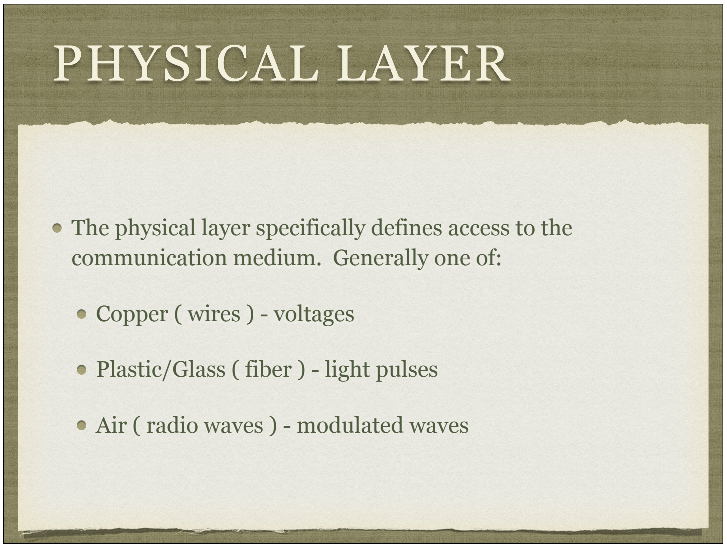
## NETWORKING?

- Connecting machines and resources for purposes of sharing and communication
- Handled on many different levels, from the physical mediums doing the connecting to the lofty application layer providing a service to the end user



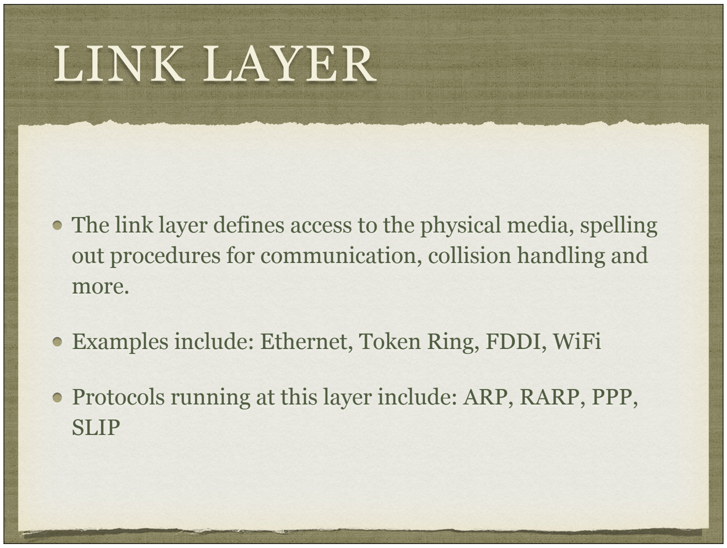
# NETWORK SANDWICH

- If you crack a networking book, they talk about the 7 layer OSI model. Then immediately tell you to toss that out the window, because the predominant networking systems of today don't really follow the model. :)
- The layers we really care about in this class include:
  - Physical - cabled, fiber, wireless
  - Link - Ethernet, 802.11
  - Network - IP
  - Transport - TCP/UDP
  - Application - HTTP, FTP, SSH, DNS, SMTP, POP3, IMAP, etc, etc, etc



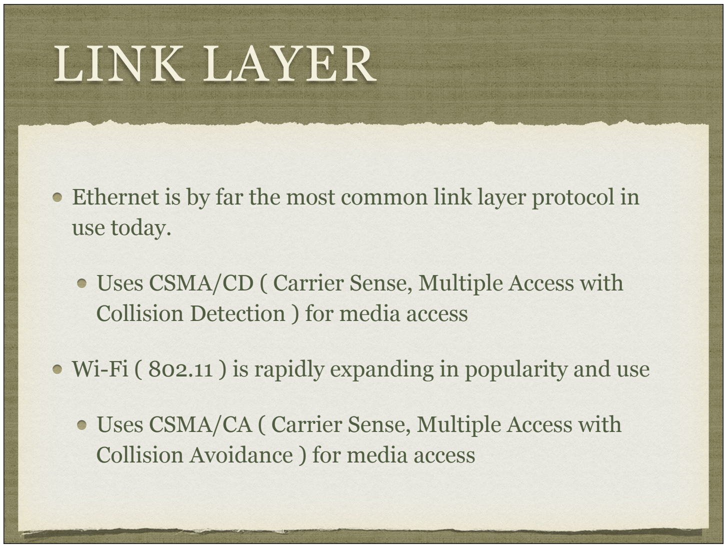
# PHYSICAL LAYER

- The physical layer specifically defines access to the communication medium. Generally one of:
  - Copper ( wires ) - voltages
  - Plastic/Glass ( fiber ) - light pulses
  - Air ( radio waves ) - modulated waves



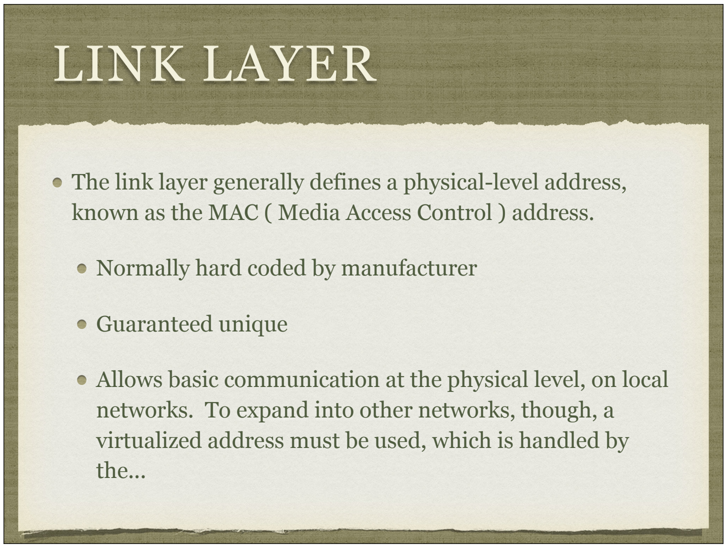
# LINK LAYER

- The link layer defines access to the physical media, spelling out procedures for communication, collision handling and more.
- Examples include: Ethernet, Token Ring, FDDI, WiFi
- Protocols running at this layer include: ARP, RARP, PPP, SLIP



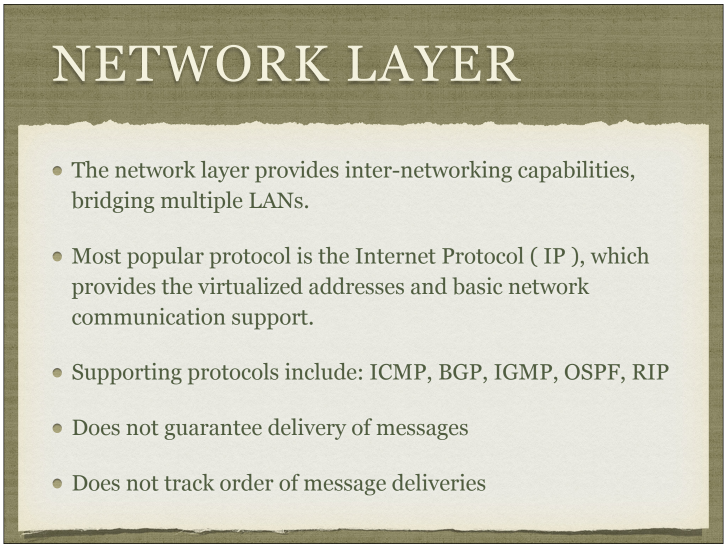
# LINK LAYER

- Ethernet is by far the most common link layer protocol in use today.
  - Uses CSMA/CD ( Carrier Sense, Multiple Access with Collision Detection ) for media access
- Wi-Fi ( 802.11 ) is rapidly expanding in popularity and use
  - Uses CSMA/CA ( Carrier Sense, Multiple Access with Collision Avoidance ) for media access



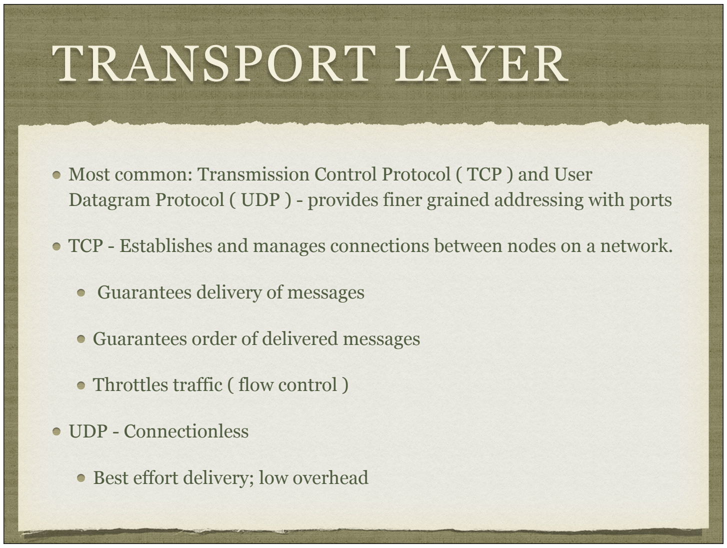
# LINK LAYER

- The link layer generally defines a physical-level address, known as the MAC ( Media Access Control ) address.
  - Normally hard coded by manufacturer
  - Guaranteed unique
  - Allows basic communication at the physical level, on local networks.  To expand into other networks, though, a virtualized address must be used, which is handled by the...



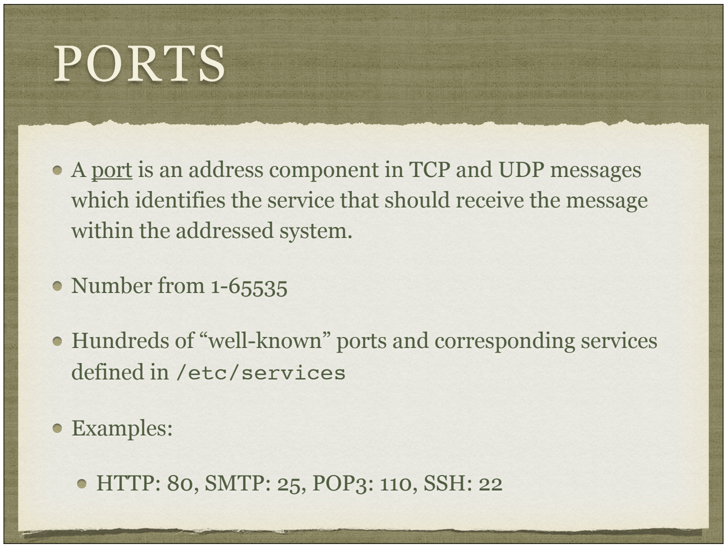
# NETWORK LAYER

- The network layer provides inter-networking capabilities, bridging multiple LANs.
- Most popular protocol is the Internet Protocol ( IP ), which provides the virtualized addresses and basic network communication support.
- Supporting protocols include: ICMP, BGP, IGMP, OSPF, RIP
- Does not guarantee delivery of messages
- Does not track order of message deliveries



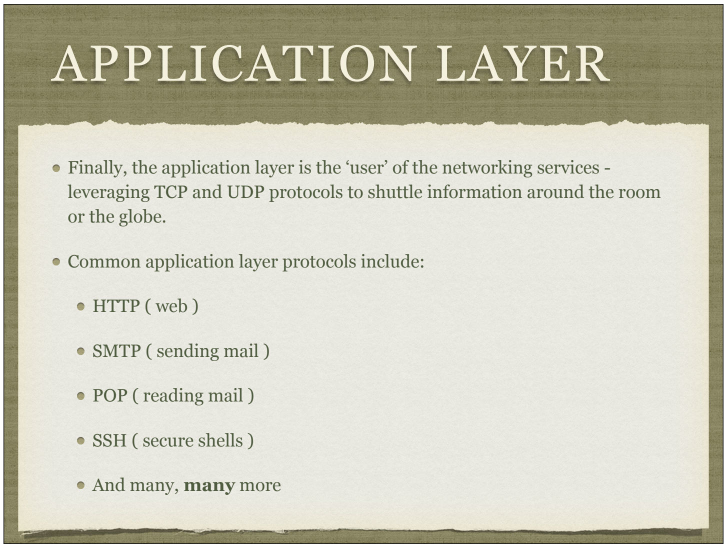
# TRANSPORT LAYER

- Most common: Transmission Control Protocol ( TCP ) and User Datagram Protocol ( UDP ) - provides finer grained addressing with ports
- TCP - Establishes and manages connections between nodes on a network.
  - Guarantees delivery of messages
  - Guarantees order of delivered messages
  - Throttles traffic ( flow control )
- UDP - Connectionless
  - Best effort delivery; low overhead



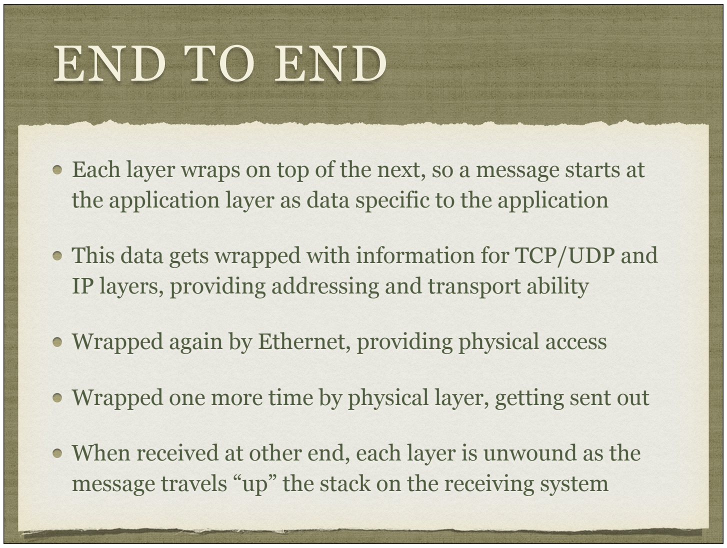
# PORTS

- A port is an address component in TCP and UDP messages which identifies the service that should receive the message within the addressed system.
- Number from 1-65535
- Hundreds of “well-known” ports and corresponding services defined in `/etc/services`
- Examples:
  - HTTP: 80, SMTP: 25, POP3: 110, SSH: 22



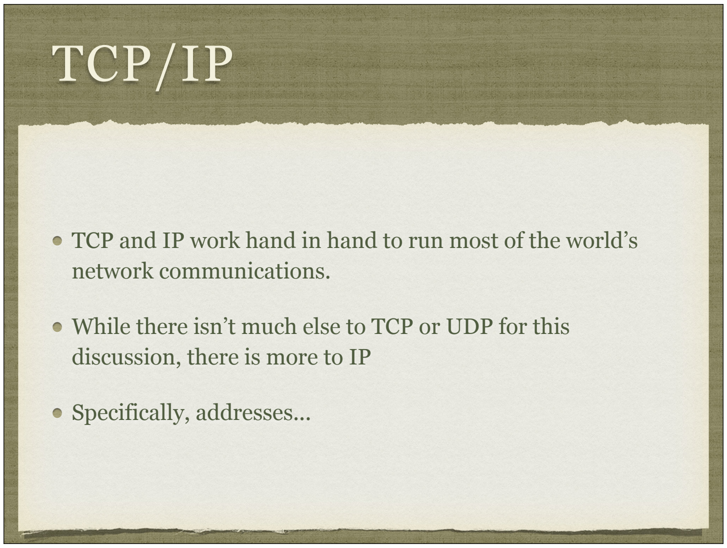
# APPLICATION LAYER

- Finally, the application layer is the ‘user’ of the networking services - leveraging TCP and UDP protocols to shuttle information around the room or the globe.
- Common application layer protocols include:
  - HTTP ( web )
  - SMTP ( sending mail )
  - POP ( reading mail )
  - SSH ( secure shells )
  - And many, **many** more



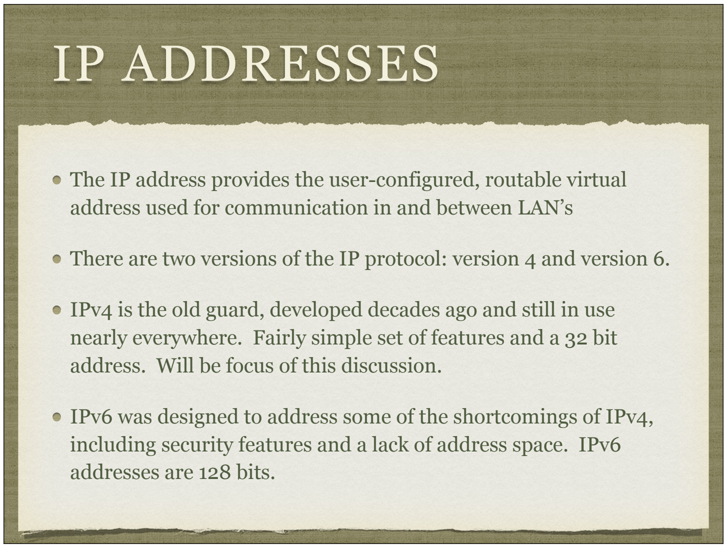
# END TO END

- Each layer wraps on top of the next, so a message starts at the application layer as data specific to the application
- This data gets wrapped with information for TCP/UDP and IP layers, providing addressing and transport ability
- Wrapped again by Ethernet, providing physical access
- Wrapped one more time by physical layer, getting sent out
- When received at other end, each layer is unwound as the message travels “up” the stack on the receiving system



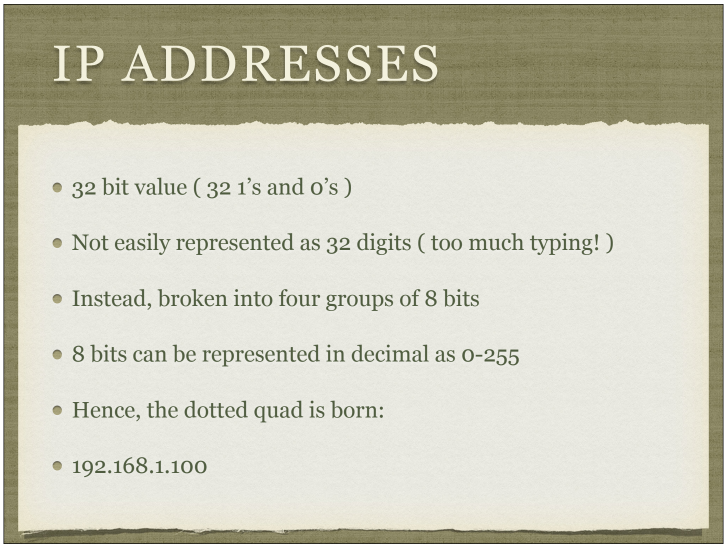
# TCP/IP

- TCP and IP work hand in hand to run most of the world's network communications.
- While there isn't much else to TCP or UDP for this discussion, there is more to IP
- Specifically, addresses...



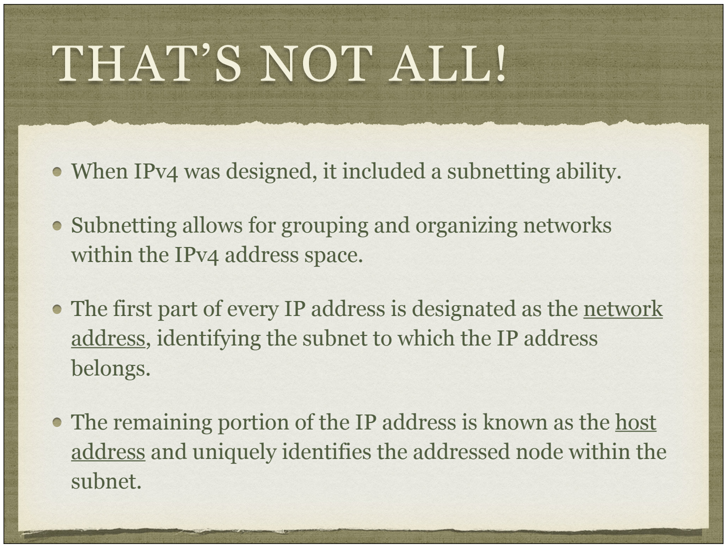
# IP ADDRESSES

- The IP address provides the user-configured, routable virtual address used for communication in and between LAN's
- There are two versions of the IP protocol: version 4 and version 6.
- IPv4 is the old guard, developed decades ago and still in use nearly everywhere. Fairly simple set of features and a 32 bit address. Will be focus of this discussion.
- IPv6 was designed to address some of the shortcomings of IPv4, including security features and a lack of address space. IPv6 addresses are 128 bits.



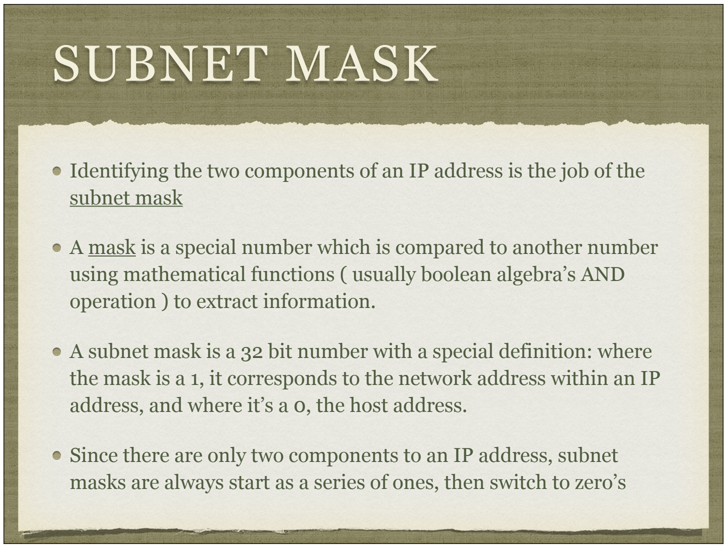
# IP ADDRESSES

- 32 bit value ( 32 1's and 0's )
- Not easily represented as 32 digits ( too much typing! )
- Instead, broken into four groups of 8 bits
- 8 bits can be represented in decimal as 0-255
- Hence, the dotted quad is born:
- 192.168.1.100



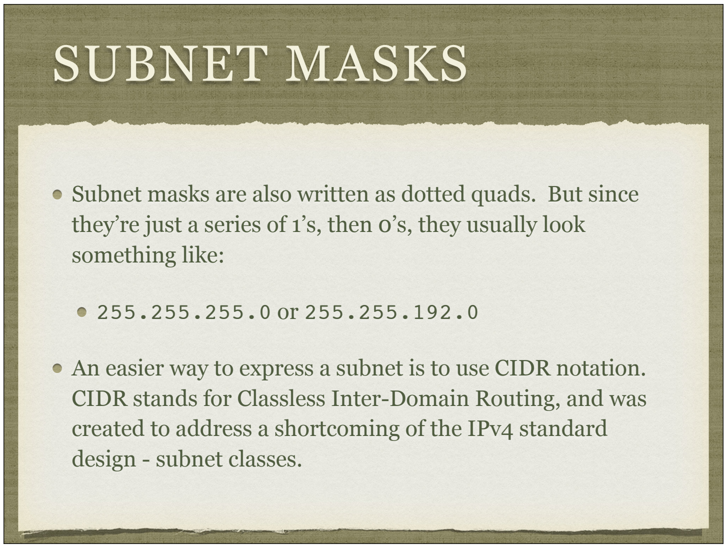
# THAT'S NOT ALL!

- When IPv4 was designed, it included a subnetting ability.
- Subnetting allows for grouping and organizing networks within the IPv4 address space.
- The first part of every IP address is designated as the network address, identifying the subnet to which the IP address belongs.
- The remaining portion of the IP address is known as the host address and uniquely identifies the addressed node within the subnet.



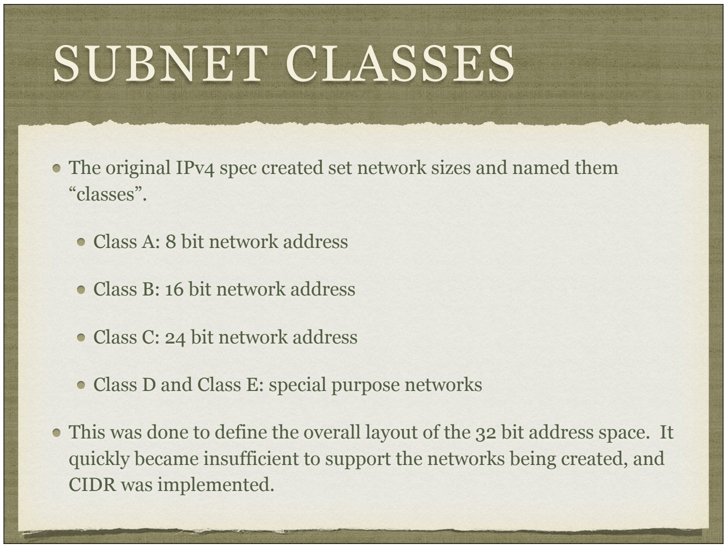
# SUBNET MASK

- Identifying the two components of an IP address is the job of the subnet mask
- A mask is a special number which is compared to another number using mathematical functions ( usually boolean algebra's AND operation ) to extract information.
- A subnet mask is a 32 bit number with a special definition: where the mask is a 1, it corresponds to the network address within an IP address, and where it's a 0, the host address.
- Since there are only two components to an IP address, subnet masks are always start as a series of ones, then switch to zero's



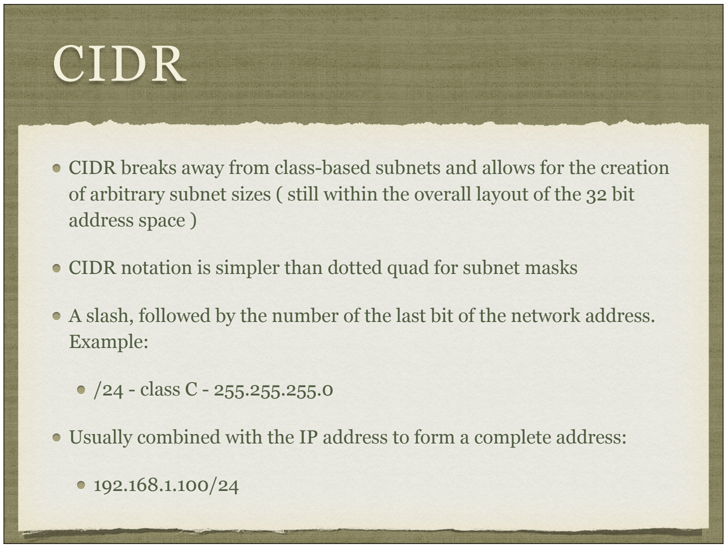
# SUBNET MASKS

- Subnet masks are also written as dotted quads. But since they're just a series of 1's, then 0's, they usually look something like:
  - `255.255.255.0` or `255.255.192.0`
- An easier way to express a subnet is to use CIDR notation. CIDR stands for Classless Inter-Domain Routing, and was created to address a shortcoming of the IPv4 standard design - subnet classes.



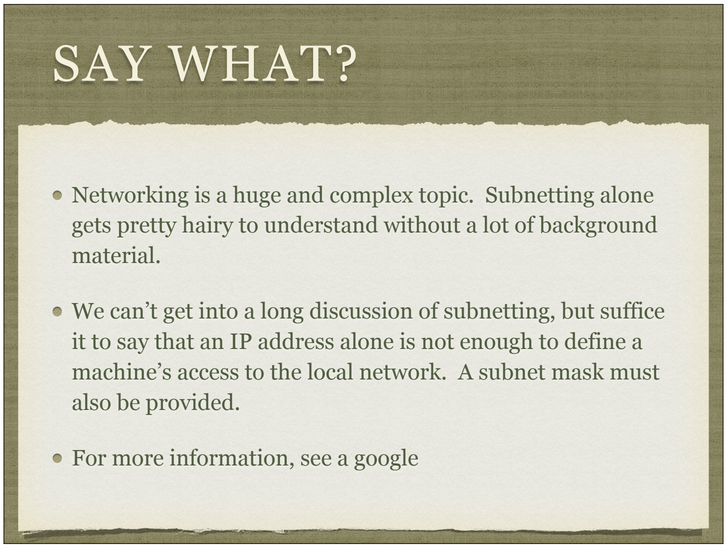
# SUBNET CLASSES

- The original IPv4 spec created set network sizes and named them “classes”.
    - Class A: 8 bit network address
    - Class B: 16 bit network address
    - Class C: 24 bit network address
    - Class D and Class E: special purpose networks
- This was done to define the overall layout of the 32 bit address space. It quickly became insufficient to support the networks being created, and CIDR was implemented.



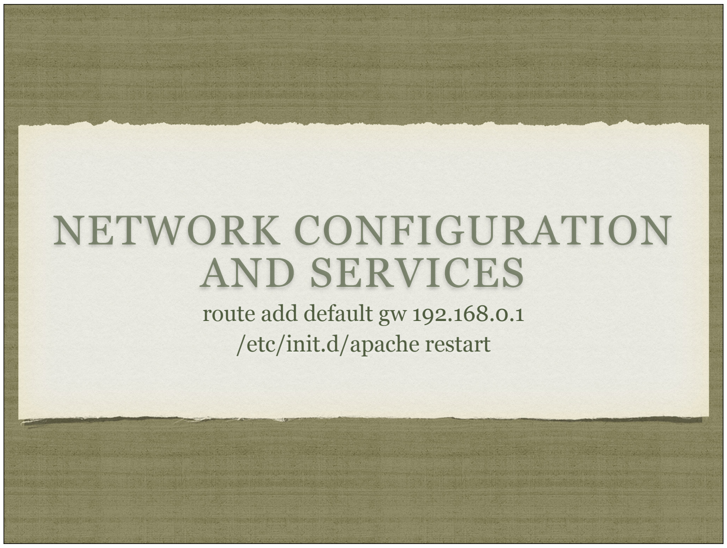
# CIDR

- CIDR breaks away from class-based subnets and allows for the creation of arbitrary subnet sizes ( still within the overall layout of the 32 bit address space )
- CIDR notation is simpler than dotted quad for subnet masks
- A slash, followed by the number of the last bit of the network address. Example:
    - /24 - class C - 255.255.255.0
- Usually combined with the IP address to form a complete address:
    - 192.168.1.100/24



# SAY WHAT?

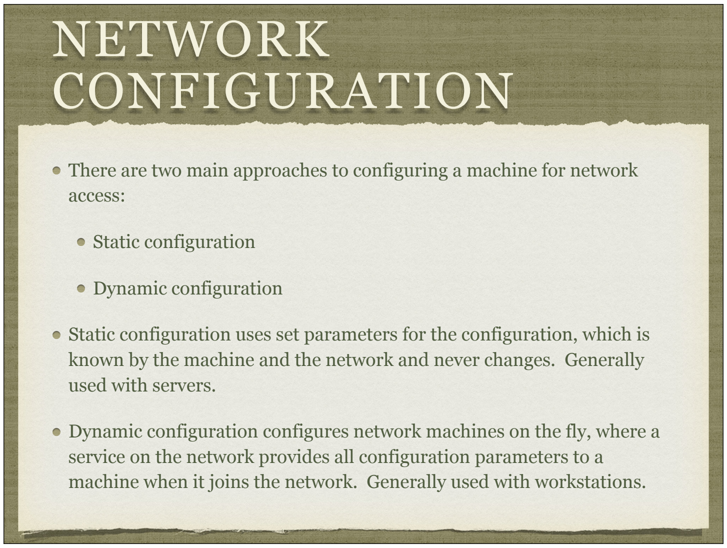
- Networking is a huge and complex topic.  Subnetting alone gets pretty hairy to understand without a lot of background material.
- We can’t get into a long discussion of subnetting, but suffice it to say that an IP address alone is not enough to define a machine’s access to the local network.  A subnet mask must also be provided.
- For more information, see a google



# NETWORK CONFIGURATION AND SERVICES

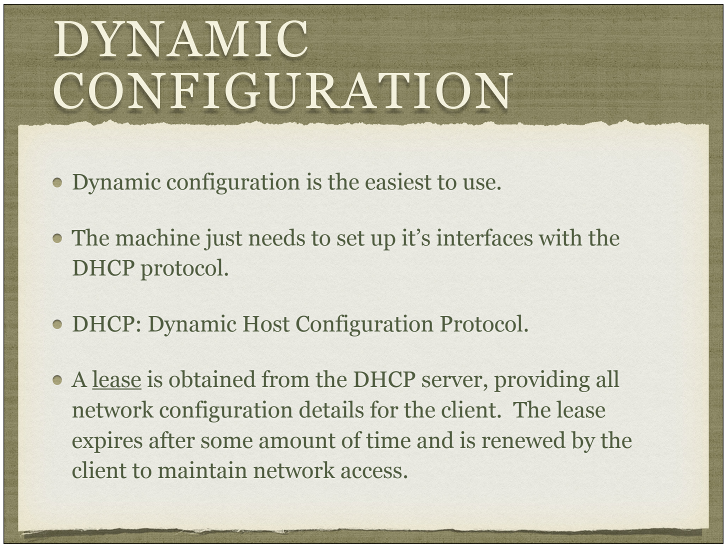
route add default gw 192.168.0.1

/etc/init.d/apache restart



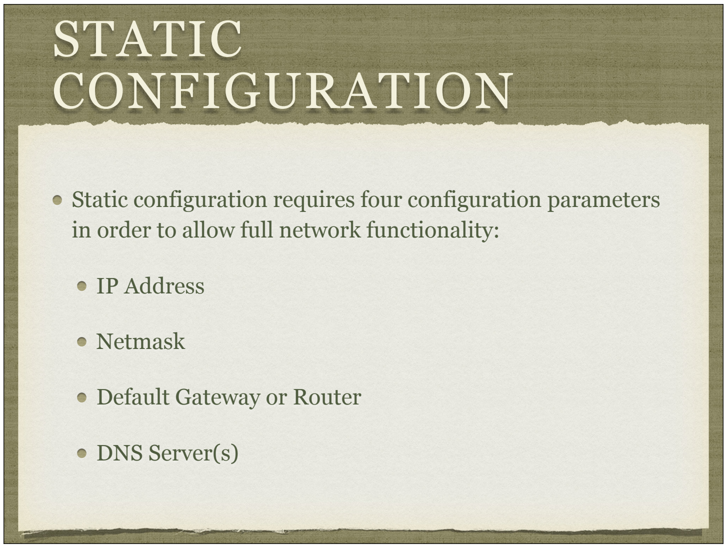
## NETWORK CONFIGURATION

- 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.



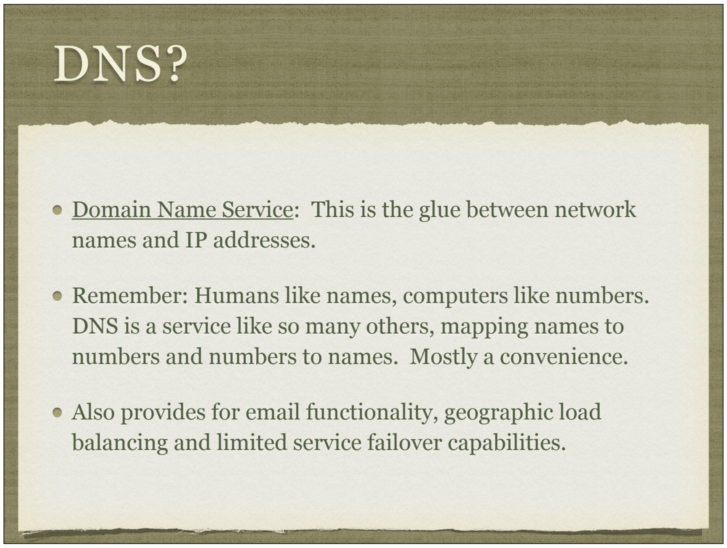
## DYNAMIC CONFIGURATION

- 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.



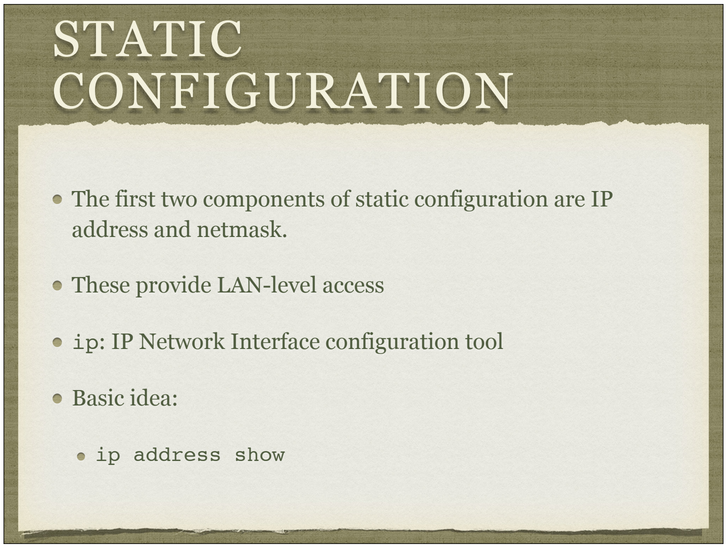
## STATIC CONFIGURATION

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



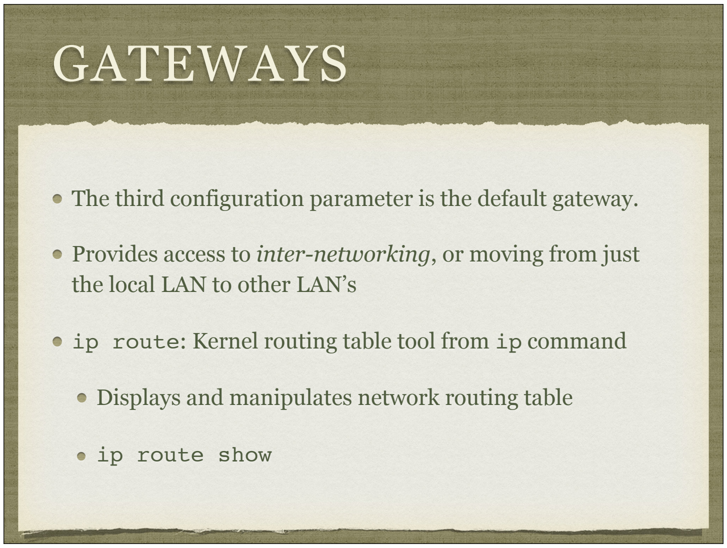
# DNS?

- 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.



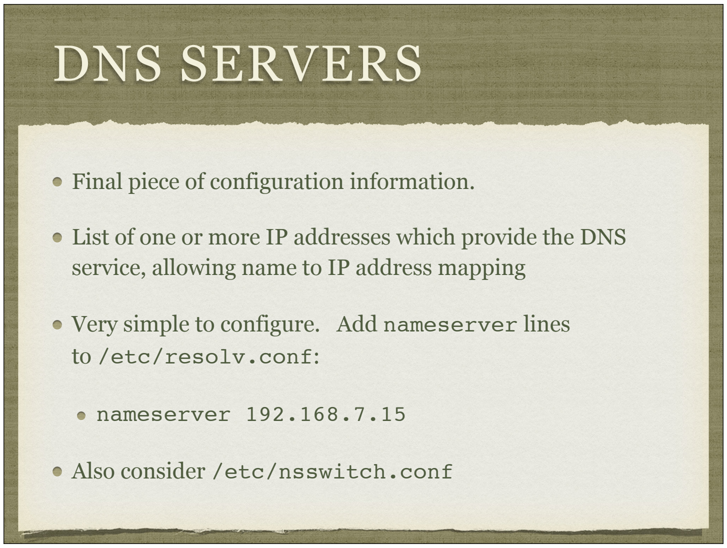
# STATIC CONFIGURATION

- The first two components of static configuration are IP address and netmask.
- These provide LAN-level access
- `ip`: IP Network Interface configuration tool
- Basic idea:
  - `ip address show`



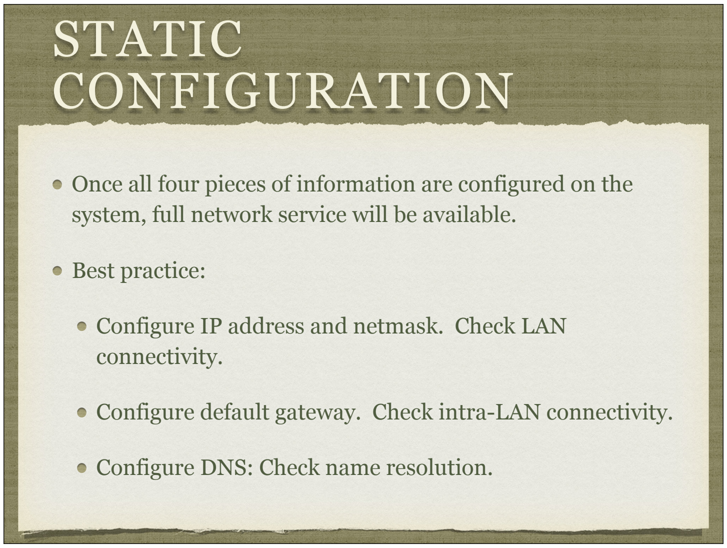
# GATEWAYS

- The third configuration parameter is the default gateway.
- Provides access to *inter-networking*, or moving from just the local LAN to other LAN's
- `ip route`: Kernel routing table tool from `ip` command
  - Displays and manipulates network routing table
  - `ip route show`



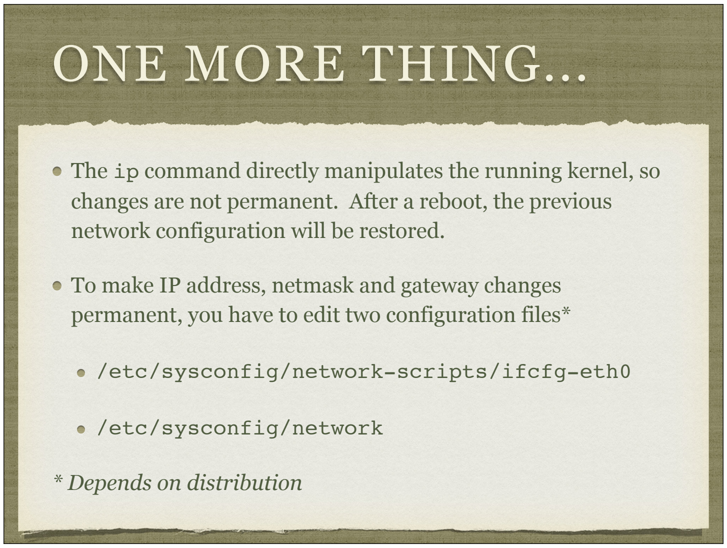
# DNS SERVERS

- 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`
- Also consider `/etc/nsswitch.conf`



# STATIC CONFIGURATION

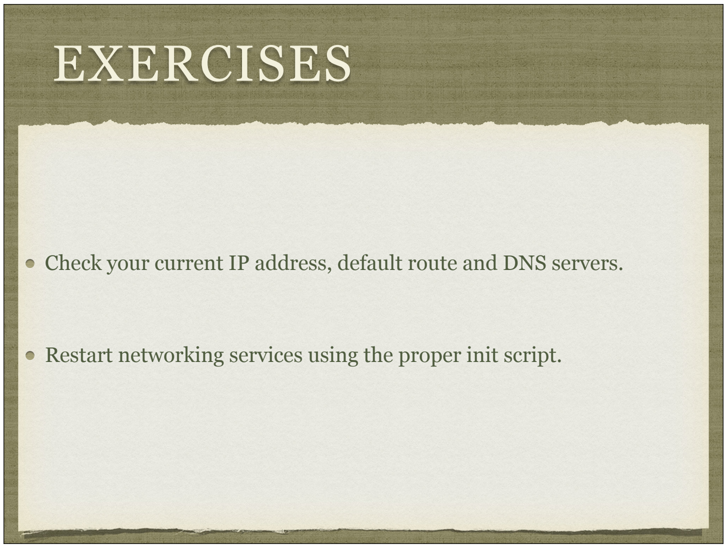
- Once all four pieces of information are configured on the system, full network service will be available.
- Best practice:
  - Configure IP address and netmask. Check LAN connectivity.
  - Configure default gateway. Check intra-LAN connectivity.
  - Configure DNS: Check name resolution.



# ONE MORE THING...

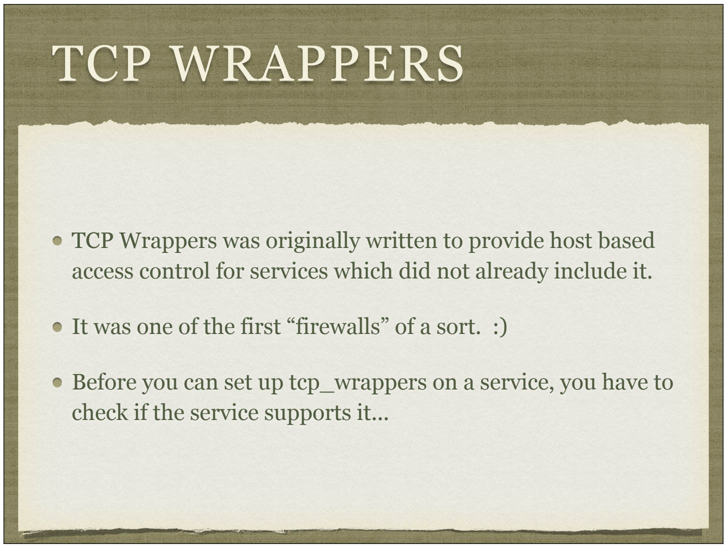
- The `ip` command directly manipulates the running kernel, so changes are not permanent. After a reboot, the previous network configuration will be restored.
- To make IP address, netmask and gateway changes permanent, you have to edit two configuration files*
  - `/etc/sysconfig/network-scripts/ifcfg-eth0`
  - `/etc/sysconfig/network`

** Depends on distribution*



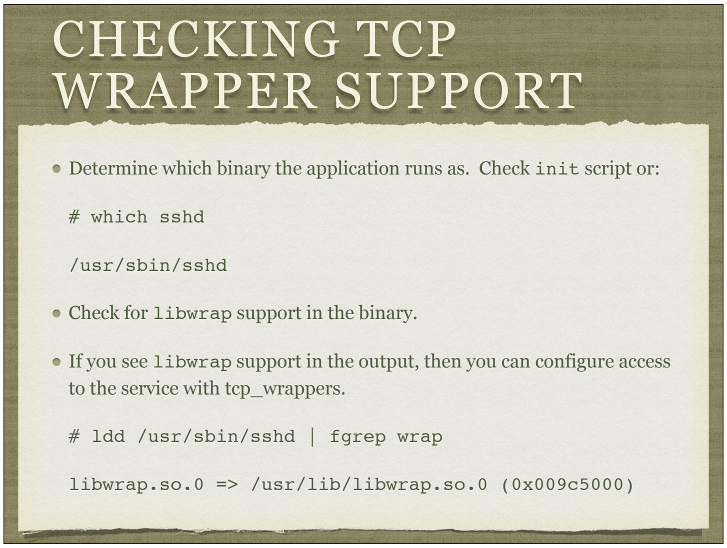
## EXERCISES

- Check your current IP address, default route and DNS servers.
- Restart networking services using the proper init script.



## TCP WRAPPERS

- TCP Wrappers was originally written to provide host based access control for services which did not already include it.
- It was one of the first “firewalls” of a sort.  :)
- Before you can set up tcp_wrappers on a service, you have to check if the service supports it...



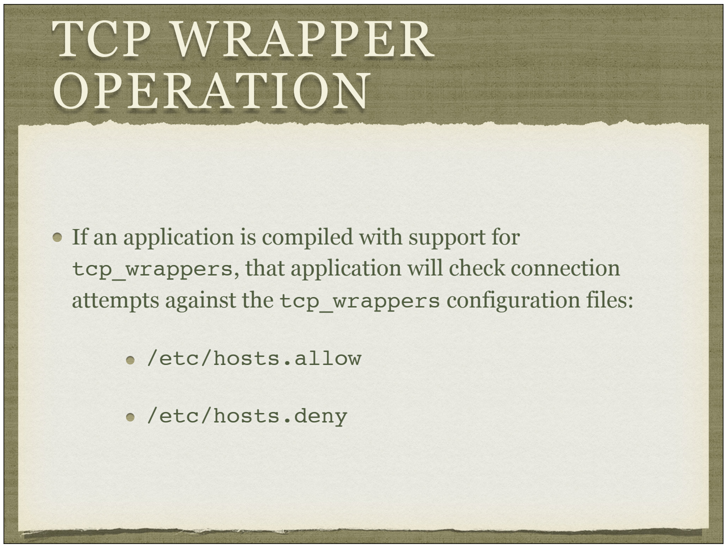
# CHECKING TCP WRAPPER SUPPORT

- Determine which binary the application runs as.  Check `init` script or:

```
# which sshd

/usr/sbin/sshd
```

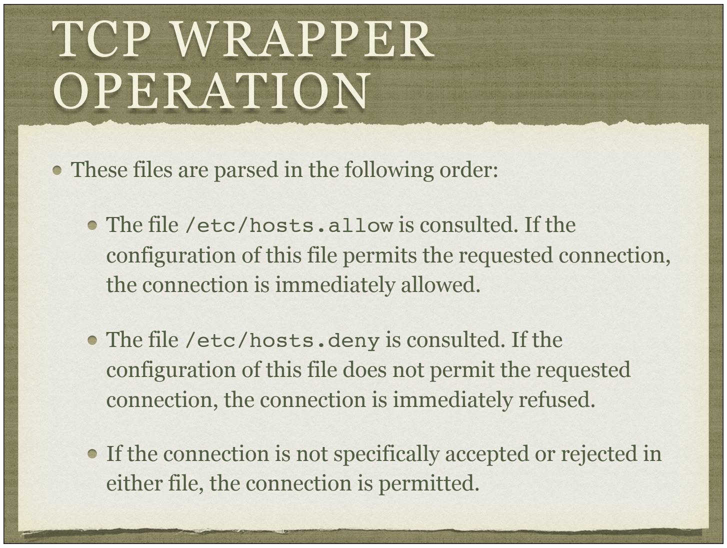
- Check for `libwrap` support in the binary.
- If you see `libwrap` support in the output, then you can configure access to the service with tcp_wrappers.

```
# ldd /usr/sbin/sshd | fgrep wrap

libwrap.so.0 => /usr/lib/libwrap.so.0 (0x009c5000)
```



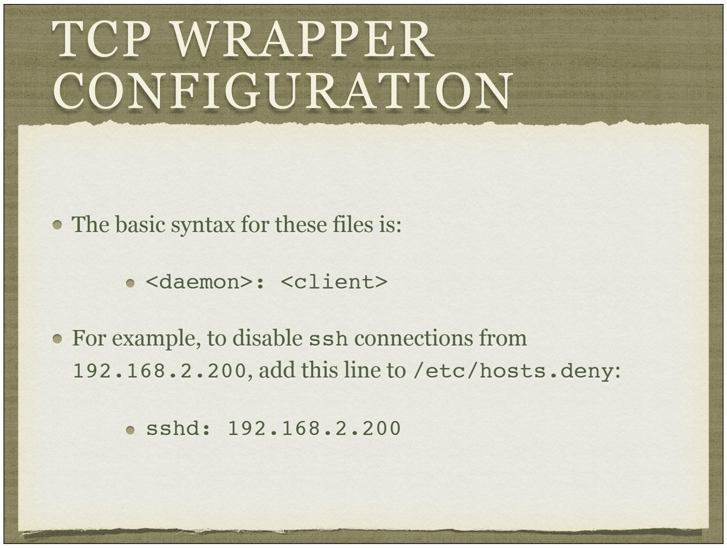
# TCP WRAPPER OPERATION

- If an application is compiled with support for `tcp_wrappers`, that application will check connection attempts against the `tcp_wrappers` configuration files:
  - `/etc/hosts.allow`
  - `/etc/hosts.deny`



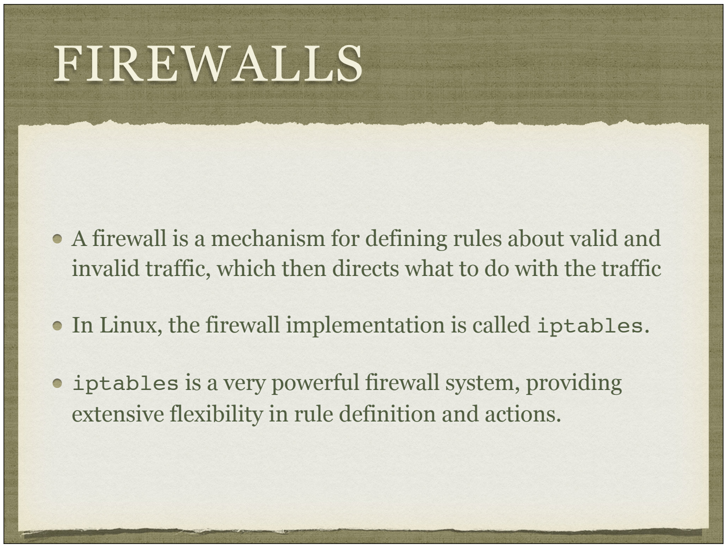
## TCP WRAPPER OPERATION

- These files are parsed in the following order:
  - The file `/etc/hosts.allow` is consulted. If the configuration of this file permits the requested connection, the connection is immediately allowed.
  - The file `/etc/hosts.deny` is consulted. If the configuration of this file does not permit the requested connection, the connection is immediately refused.
  - If the connection is not specifically accepted or rejected in either file, the connection is permitted.



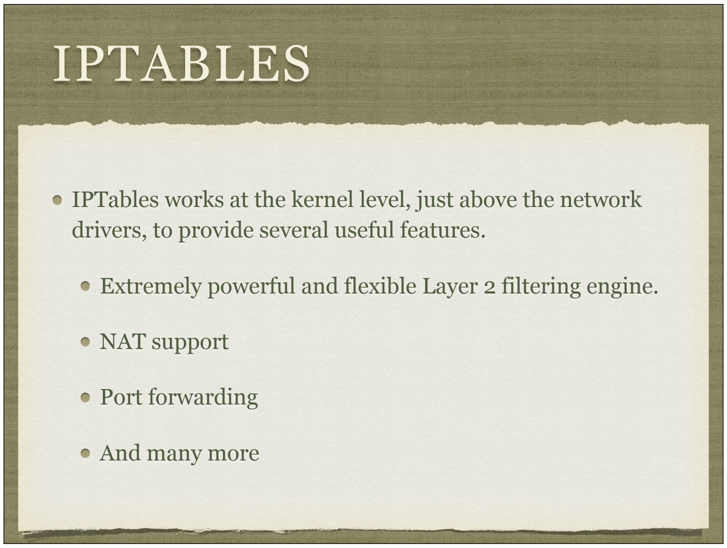
## TCP WRAPPER CONFIGURATION

- The basic syntax for these files is:
  - `<daemon>: <client>`
- For example, to disable `ssh` connections from `192.168.2.200`, add this line to `/etc/hosts.deny`:
  - `sshd: 192.168.2.200`



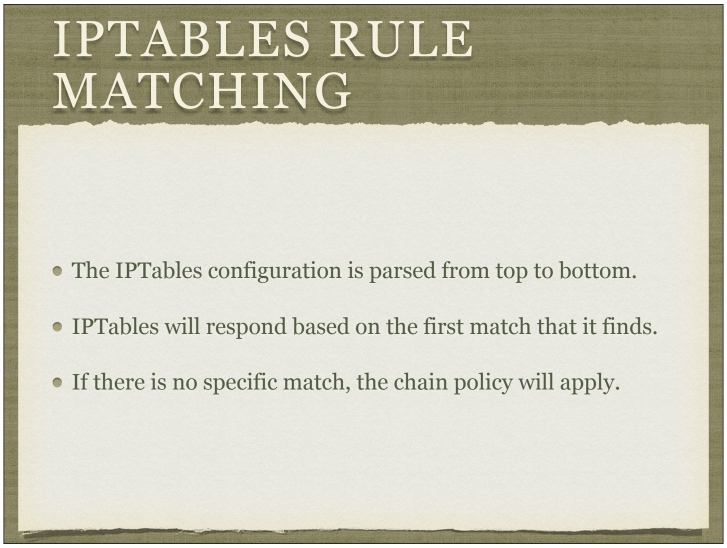
# FIREWALLS

- A firewall is a mechanism for defining rules about valid and invalid traffic, which then directs what to do with the traffic
- In Linux, the firewall implementation is called `iptables`.
- `iptables` is a very powerful firewall system, providing extensive flexibility in rule definition and actions.



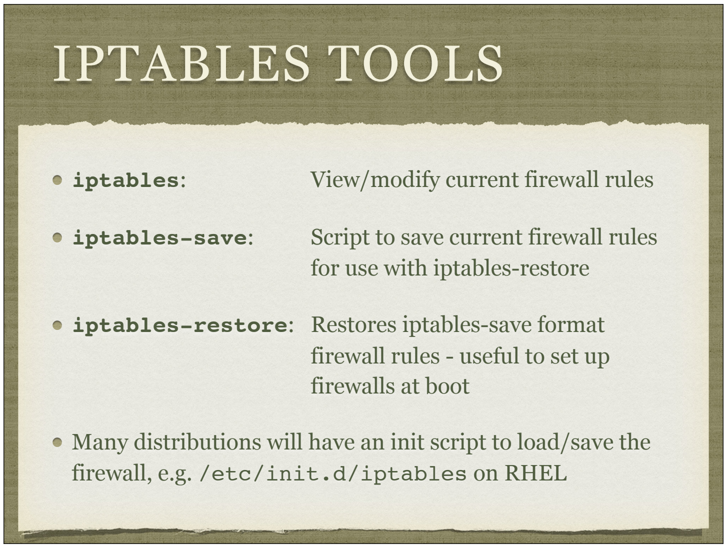
# IPTABLES

- IPTables works at the kernel level, just above the network drivers, to provide several useful features.
  - Extremely powerful and flexible Layer 2 filtering engine.
  - NAT support
  - Port forwarding
  - And many more



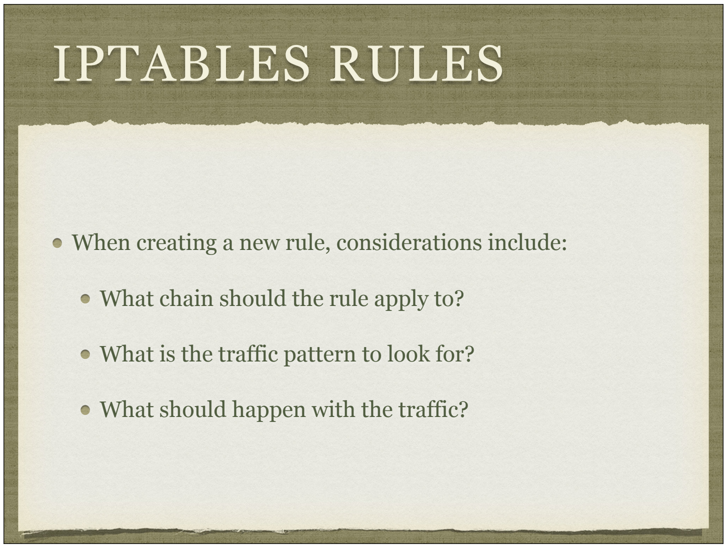
# IPTABLES RULE MATCHING

- The IPTables configuration is parsed from top to bottom.
- IPTables will respond based on the first match that it finds.
- If there is no specific match, the chain policy will apply.



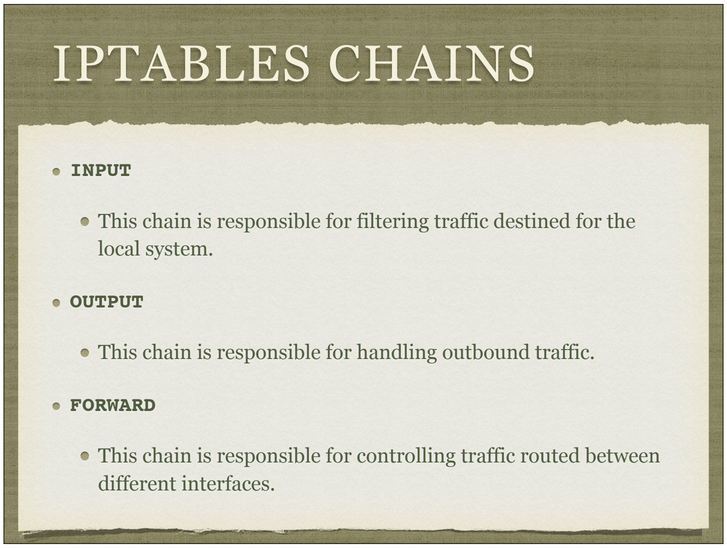
# IPTABLES TOOLS

- **`iptables`**: View/modify current firewall rules
- **`iptables-save`**: Script to save current firewall rules for use with iptables-restore
- **`iptables-restore`**: Restores iptables-save format firewall rules - useful to set up firewalls at boot
- Many distributions will have an init script to load/save the firewall, e.g. `/etc/init.d/iptables` on RHEL



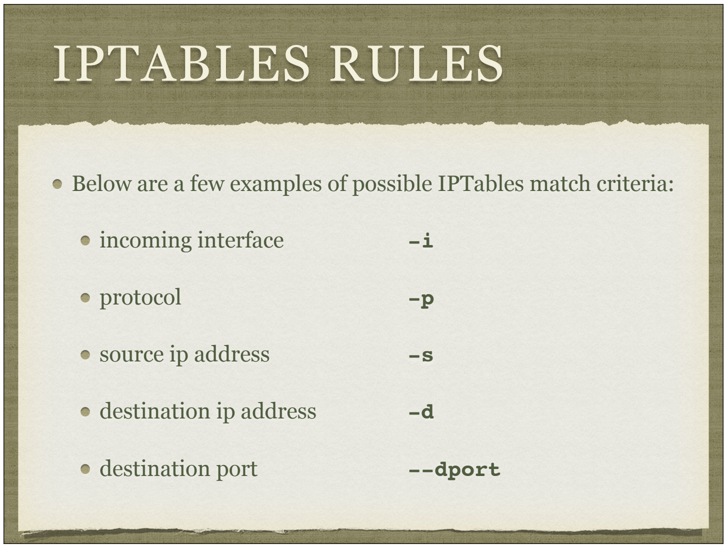
# IPTABLES RULES

- When creating a new rule, considerations include:
  - What chain should the rule apply to?
  - What is the traffic pattern to look for?
  - What should happen with the traffic?



# IPTABLES CHAINS

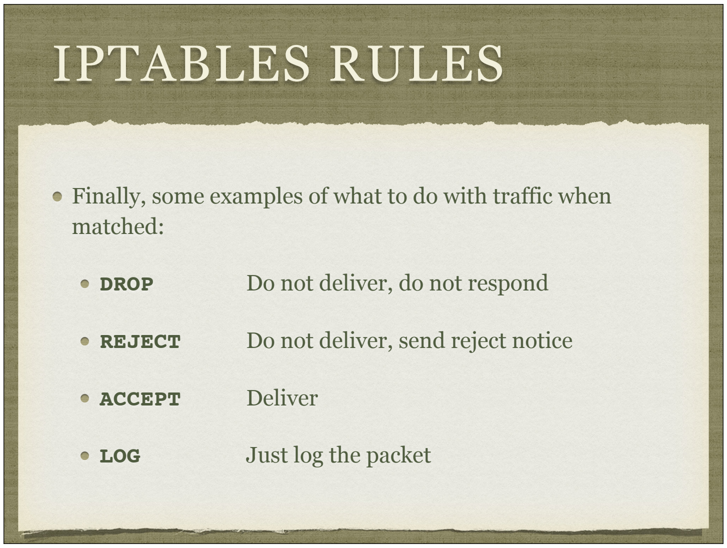
- **`INPUT`**
  - This chain is responsible for filtering traffic destined for the local system.
- **`OUTPUT`**
  - This chain is responsible for handling outbound traffic.
- **`FORWARD`**
  - This chain is responsible for controlling traffic routed between different interfaces.



# IPTABLES RULES

- Below are a few examples of possible IPTables match criteria:

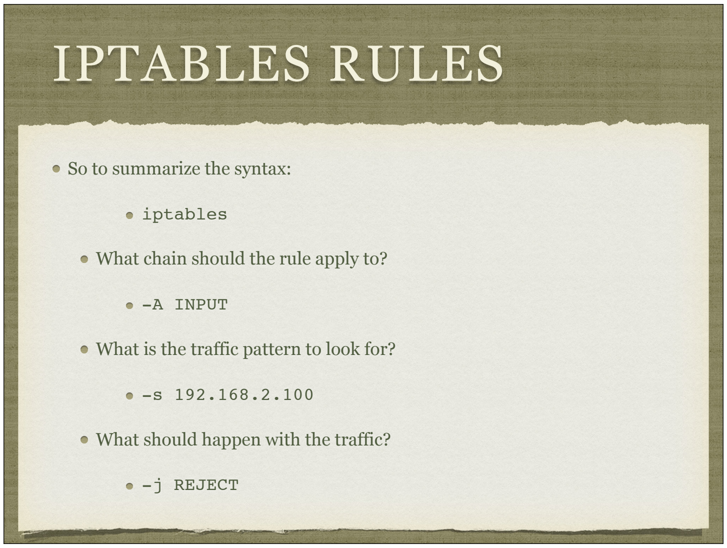
| | |
|---|---|
| incoming interface | `-i` |
| protocol | `-p` |
| source ip address | `-s` |
| destination ip address | `-d` |
| destination port | `--dport` |



# IPTABLES RULES

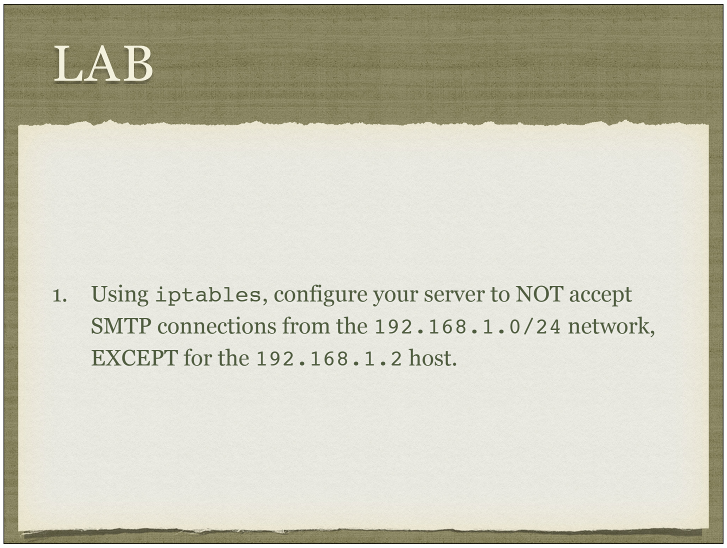
- Finally, some examples of what to do with traffic when matched:

| | |
|---|---|
| `DROP` | Do not deliver, do not respond |
| `REJECT` | Do not deliver, send reject notice |
| `ACCEPT` | Deliver |
| `LOG` | Just log the packet |



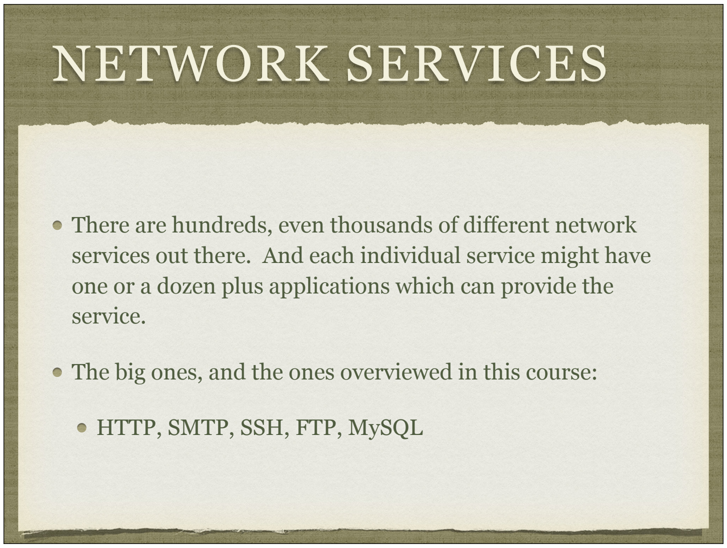
# IPTABLES RULES

- So to summarize the syntax:
    - `iptables`
  - What chain should the rule apply to?
    - `-A INPUT`
  - What is the traffic pattern to look for?
    - `-s 192.168.2.100`
  - What should happen with the traffic?
    - `-j REJECT`



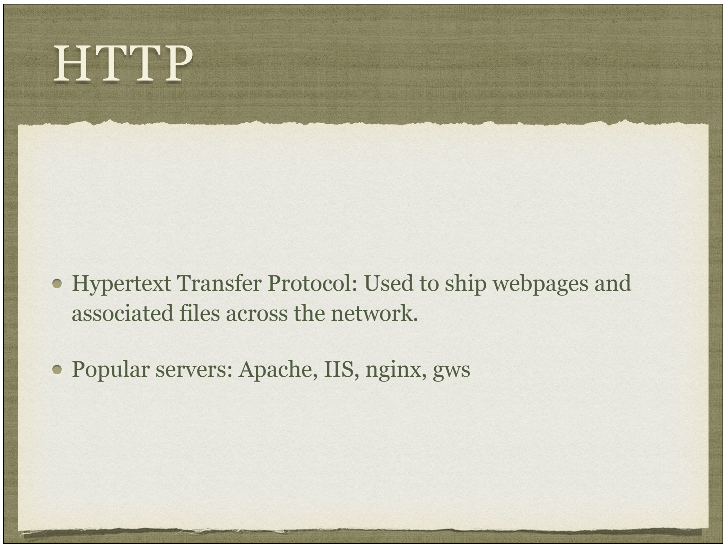
# LAB

1. Using `iptables`, configure your server to NOT accept SMTP connections from the `192.168.1.0/24` network, EXCEPT for the `192.168.1.2` host.



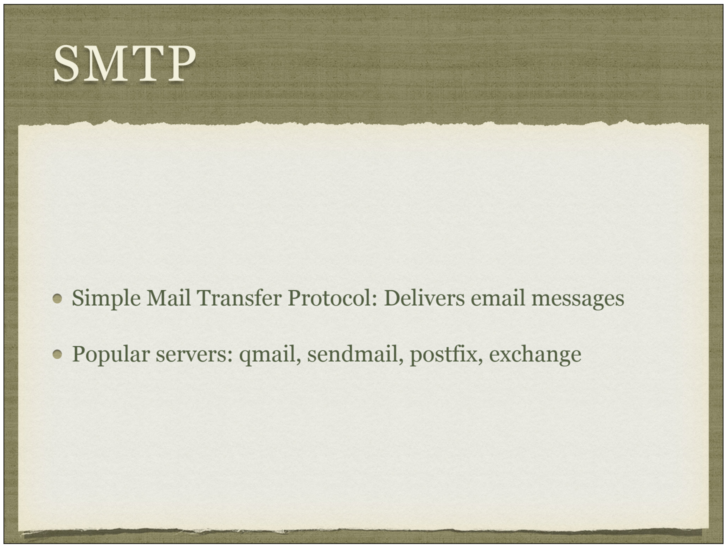
# NETWORK SERVICES

- There are hundreds, even thousands of different network services out there.  And each individual service might have one or a dozen plus applications which can provide the service.
- The big ones, and the ones overviewed in this course:
  - HTTP, SMTP, SSH, FTP, MySQL



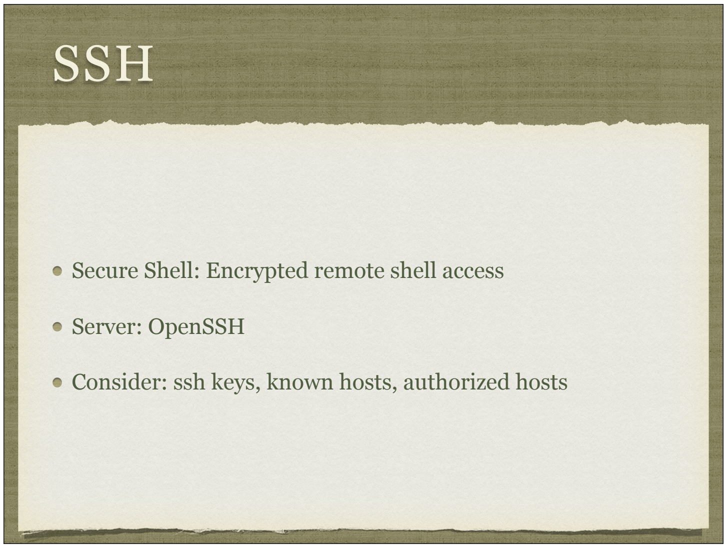
# HTTP

- Hypertext Transfer Protocol: Used to ship webpages and associated files across the network.
- Popular servers: Apache, IIS, nginx, gws



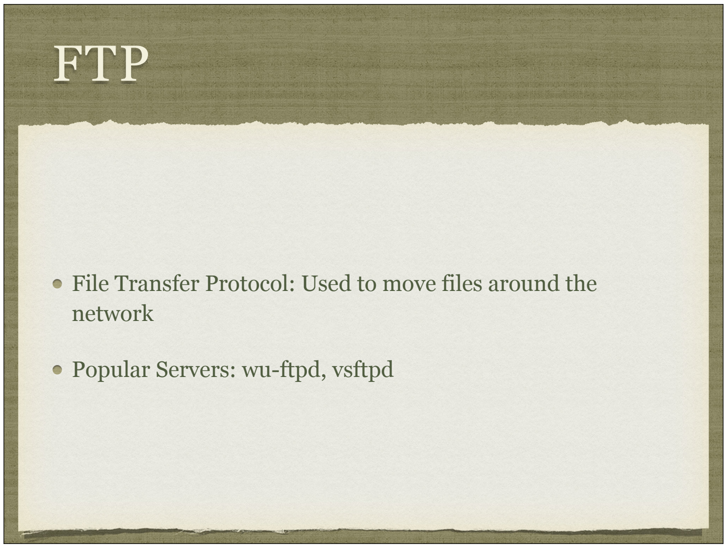
## SMTP

- Simple Mail Transfer Protocol: Delivers email messages
- Popular servers: qmail, sendmail, postfix, exchange



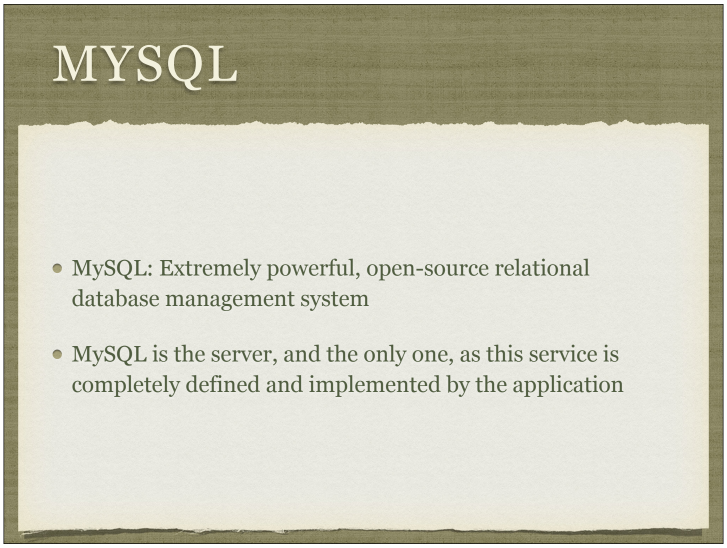
## SSH

- Secure Shell: Encrypted remote shell access
- Server: OpenSSH
- Consider: ssh keys, known hosts, authorized hosts



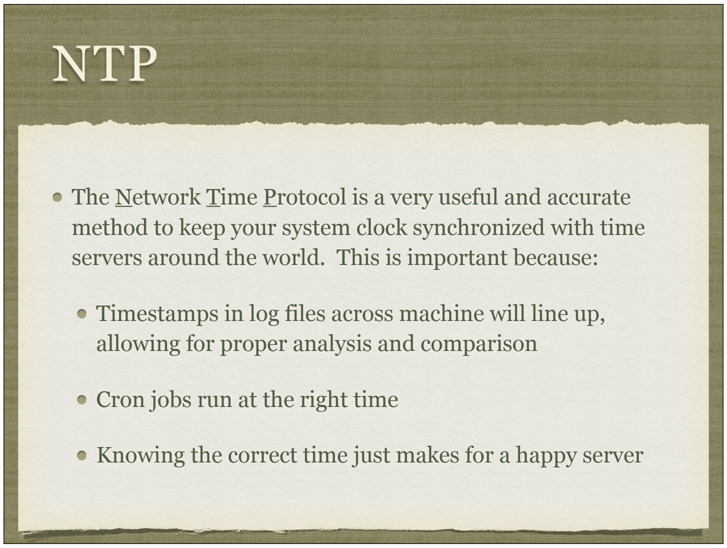
# FTP

- File Transfer Protocol: Used to move files around the network
- Popular Servers: wu-ftpd, vsftpd



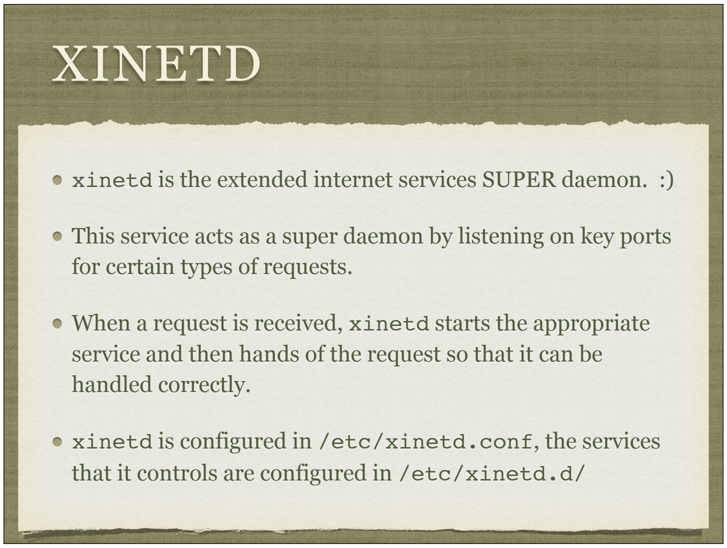
# MYSQL

- MySQL: Extremely powerful, open-source relational database management system
- MySQL is the server, and the only one, as this service is completely defined and implemented by the application



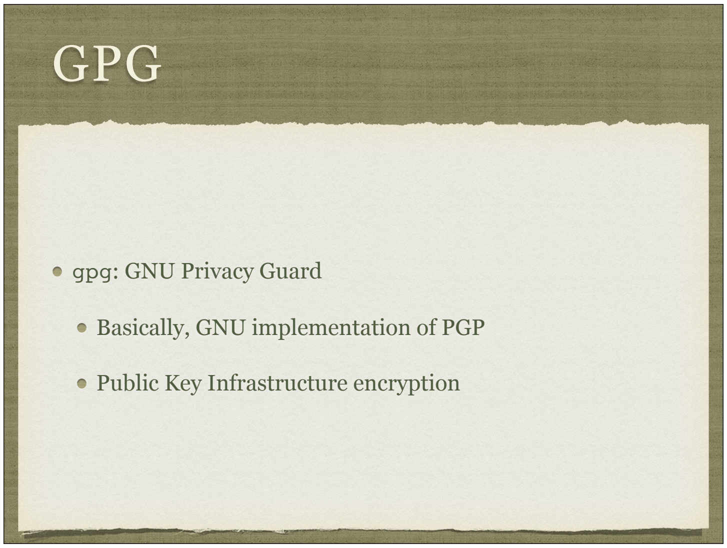
## NTP

- The Network Time Protocol is a very useful and accurate method to keep your system clock synchronized with time servers around the world. This is important because:
  - Timestamps in log files across machine will line up, allowing for proper analysis and comparison
  - Cron jobs run at the right time
  - Knowing the correct time just makes for a happy server



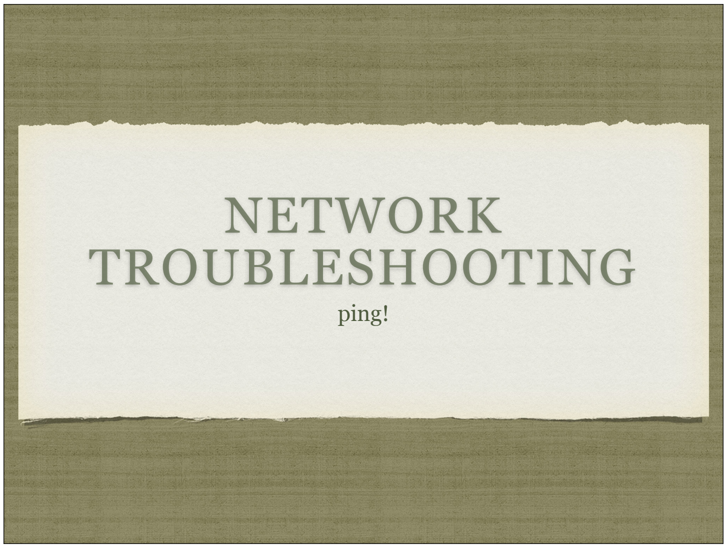
## XINETD

- `xinetd` is the extended internet services SUPER daemon. :)
- This service acts as a super daemon by listening on key ports for certain types of requests.
- When a request is received, `xinetd` starts the appropriate service and then hands of the request so that it can be handled correctly.
- `xinetd` is configured in `/etc/xinetd.conf`, the services that it controls are configured in `/etc/xinetd.d/`



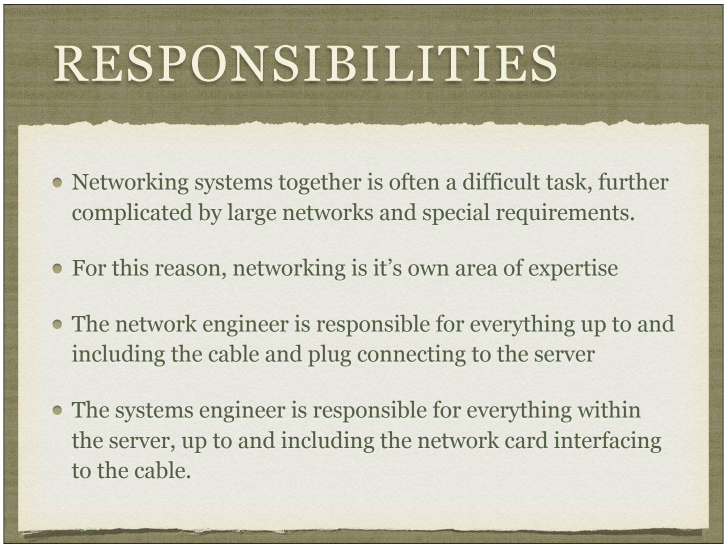
# GPG

- gpg: GNU Privacy Guard
  - Basically, GNU implementation of PGP
  - Public Key Infrastructure encryption



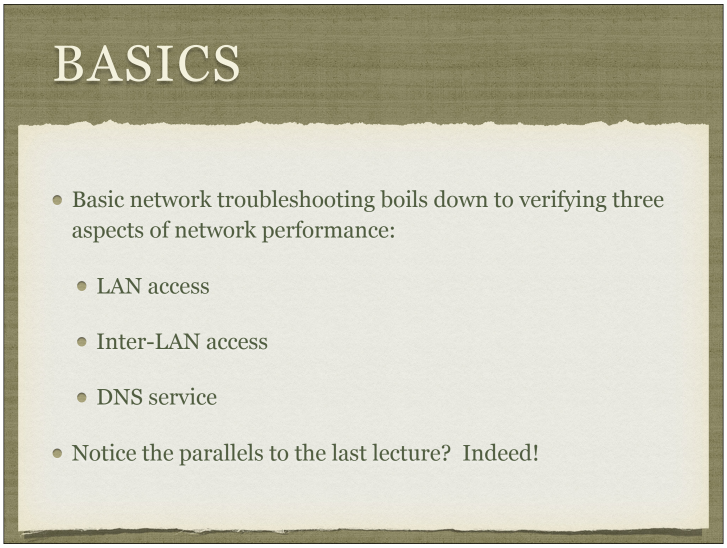
# NETWORK TROUBLESHOOTING

ping!



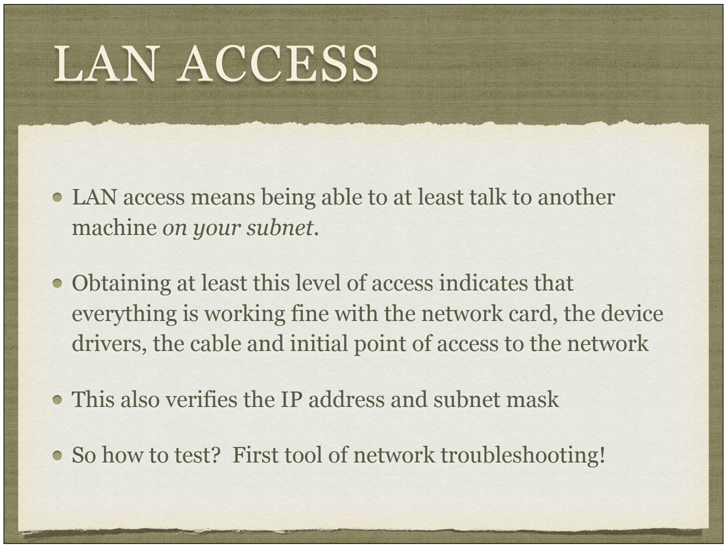
## RESPONSIBILITIES

- 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.



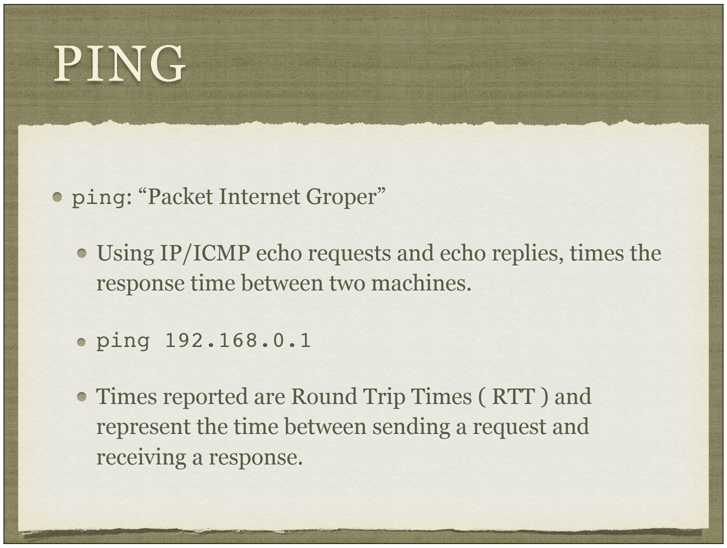
## BASICS

- 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!



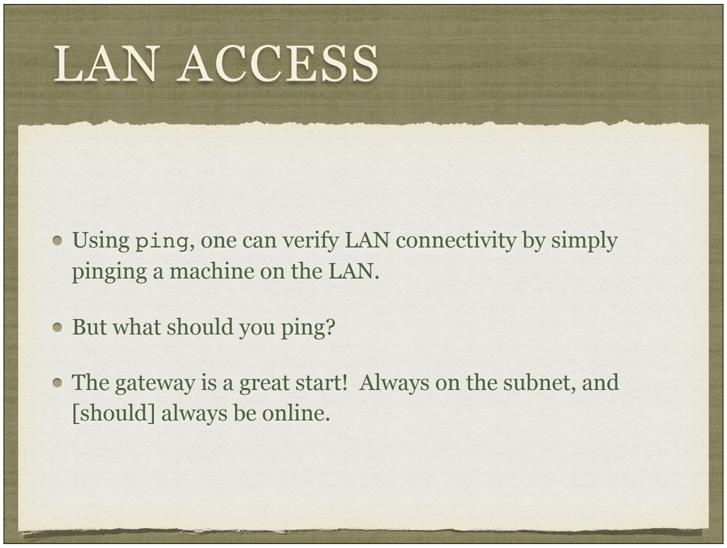
## LAN ACCESS

- 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!



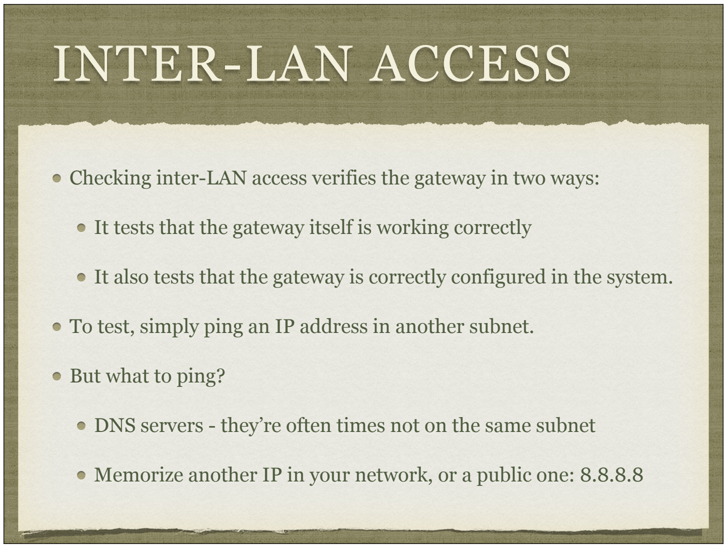
# PING

- `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.



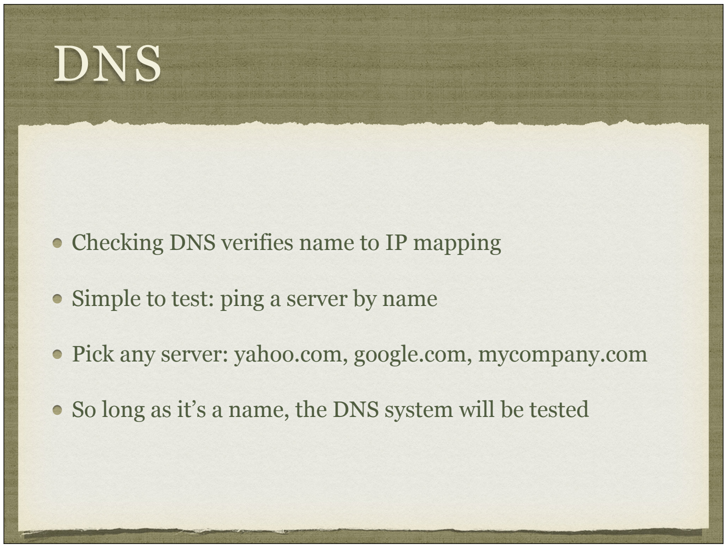
# LAN ACCESS

- 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.



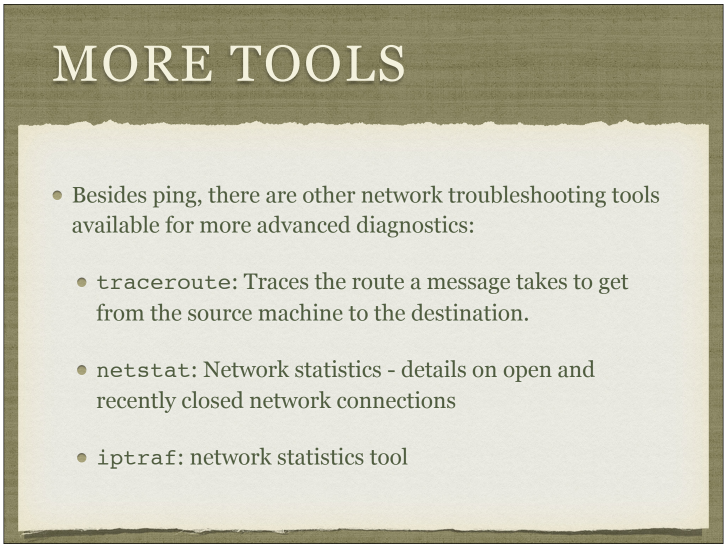
# INTER-LAN ACCESS

- 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



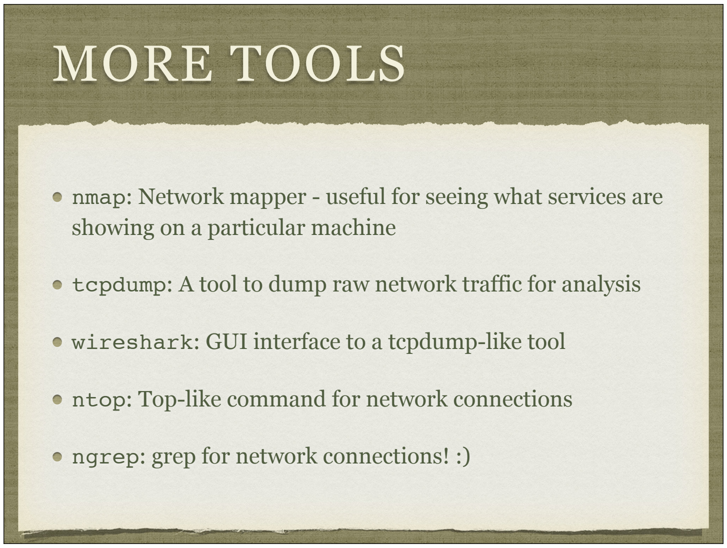
# DNS

- 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



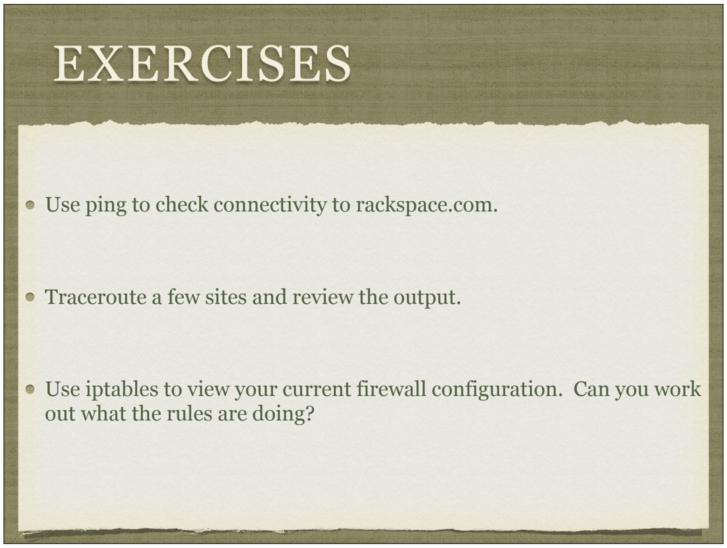
# MORE TOOLS

- 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.
- Use iptables to view your current firewall configuration. Can you work out what the rules are doing?