1 Introduction to the Course

- The cold hard truth
  - this course is NOT sponsored by the CS dept.
  - you will not receive any credit at all

- introduce ourselves to the students
  - Should introduce myself and Marshall
  - are seniors in CS and CS/Math

- And what do we think we’re doing...
  - here to teach you fundamentals of Unix
  - to teach you the student to teach yourself about unix
  - to give you the necessary basics to go out and teach yourself whatever you might need to know to get work done on unix
  - catch a fish for a man vs teach a man to fish
  - trying to follow Mudd philosophy - teach you to teach yourself

- who this course is targeted at
  - those students who have never worked with unix before and would to get some experience doing
  - those students who have had some experience but would like to learn more

- Some reasons to learn about Unix
  - linux and OS X
  - solid and basic understanding of unix
  - not enough time in the regular school year to teach you this and teach you everything else you need to know

  - and also because the CS department doesn’t have the resources, or want to offer this course on its own - maybe they will in the future
  - we needed something to do with our summer vacation
  - the goodness of our hearts, etc.

- To make this perfectly clear...
  - will not teach you to program
  - will not teach you emacs tricks; will provide a reference sheet
  - not about learning to use a single application
  - will not solve world hunger, peace, or to be a better person

2 Computers on Campus

- Available computing resources
  - graphics lab
  - terminal room - printer
  - AC and LAC labs - printer and scanner
  - Engineering lab - to be avoided

- Servers
  - odin, thuban, banshee
  - turing, wilkes, muddcs, arc, cortana

- People to know
  - Tim Buchheim and Roger Wiechman
  - Bring them milk, cookies and be nice to them, they make things work
3 Philosophy and History of UNIX

• Origin of Unix
  • computers were mainframes that people worked at, much like turing
  • teletypes - basically a typewriter plugged into a computer
    · Ken Thomson implemented the first UNIX environment
      have been working at Bell Labs on a project called Multics (Multiplexed ...)
  • Unix was kind of a castrated Multics - much smaller because invented on an underpowered system - PDP-7
  • became popular mainly because it was distributed to Universities
  • will want to display Unix timeline

• Unix Today
  • programmers environment - because of this is extremely powerful
    · still very much a tinkerer’s environment
    · unix encourages and support ”hacking” the system
    · Fundamentally multi-user and multitasking
  • designed in the days of mainframe environments when many people would all work on one computer system
  • we still use the same model in many ways with turing
  • clean, simple interface
  • multi-layered design: user — (shell, other programs) — OS — hardware
  • at top user at bottom hardware, what’s in-between?
  • scales very well - from supercomputers to microwaves (embedded systems)
  • two different flavors of unix - BSD (FreeBSD, OSX) and System V (Solaris, Linux)
  • about as different as mango, peach, kiwi, strawberry ice-cream and peach, strawberry, kiwi, mango ice-cream with bits of vanilla
  • especially that way in linux, its all just annoying

• What is Unix?
  • unix in some sense is not an operating system but rather a programming language (C)/philosophy/interface (POSIX)
  • the development of C (and the internet) are all connected
  • unix is (kind of) C, the Standard C Library, and POSIX (Portable Operating System Standard) which is the API that programmers use to interact with the system kernel
  • the design and implementation of unix also follows a certain philosophy about how an OS should be designed, how programs should be written, etc.
  • everything else is just historical baggage
  • makes unix extremely flexible - can start from scratch with a new implementation but still same ”design201d
  • what would be some good philosophical ideas to follow when building an OS?
  • what are some good rules of thumb? what trade-offs do you want to make?

• Unix philosophy
  • programs should be designed to operate together
  • Programs are viewed as simple tools that combined will do complex tasks
  • LEGO blocks - simple tools but can create very complex systems
  • another analogy - hammer, wrench, saw
  • Avoid the ginormous programs that do everything
  • want to keep everything as simple as possible
  • separate mechanism from policy
  • walk up to some one and try to scare them
    · principle of least surprise
  • there is no ”one true way201d - unix is not perfect!
  • ”Unix *is* user-friendly. It2019s just picky about its friends.201d
very true - have to be able to understand the system at some intuitive level to be able to work with it
also if you don’t like something you can fix it, maybe not easily, but you can fix it
What makes a system secure? What needs to be protected?
What is a good way of protecting things, like property? Ownership

• Unix security
  • concept of ownership - everything on the system is owned by someone
  • ownership implies responsibility - so whoever owns these files is in the end responsible for them
  • protect users from one another and from doing dangerous things
  • protect programs and files from users - or rather allow only certain users to do certain things
  • security: not perfect, can break, and will cause you to scream a few months down the line

4 Types of users

• what powers is each user going to have?
• do you need to have a class of users that manage the system?
• what kinds of users are there going to be?
  • administrative users
    • root - is the best known account of this type - aka The SuperUser
    • complete control over everything on the system
    • can do anything that they want (mostly) - has to respect file perms.
    • can become any user on the system
    • loaded gun with the safety off, easy to shoot yourself in the foot
    • is a standard account on every UNIX/Linux/BSD machine
    • used for system administration/maintenance
    • any other user however can be granted admin privileges, so root isn’t strictly necessary to run a system
• why would you want? what if you tried to delete all the files on the system? shouldn’t be able to do
• should there be different types of normal users?
  • normal users
    • have a home directory - directory to keep all your files to yourself
    • cannot generally see others home directories, Odin had this problem and Charlie still does
    • can use most of the programs on the system
    • any special configuration files are generally stored in your home directory
    • on turing you have about 60 Mb for all your files
    • on new turing will have much more
    • on your own system can have as much as you want ;-)  
    • the CS department does daily backups so if you lose a file you REALLY want then you can ask them to restore it
• for what jobs do you need a third type of user?
• what jobs do you not want root doing, or a normal user? webserver
  • other users: nobody, ssh, ftp, etc.
    • used to run daemons in the background
    • for security reasons do not want to have a web server running as root
    • what if someone owned your web daemon - would own your system...is bad
    • unless you’re setting up your own system don’t have to worry about this
    • and even then still don’t really have to worry about this since most of these users will have been created by default - rarely have to create a new system user
    • now that we have users, how do we manage them?
    • should all users be lumped in to one category?
• groups
  • every user on the system belongs to one or more groups
these are sets of users which all share some characteristic
→ on the board draw a Venn diagram
→ one group, on turing at least, is students
→ every student that has an account belongs to this group
→ consultants are additionally members of the operator group
→ however since consultants are also students, we draw another distinction
→ every user has a primary group - this is a group to which file which you create will be assigned
→ can generally be a member to as many groups as the admin wants you to be
→ some groups serve special purposes along with the "special" users
· exists to allow easier management of a system by administrative users

5 Logging In

• Login
→ two parts: username and password
→ username can be anything, though some are reserved for special purposes
→ on turing it will be the first character of your first name, followed by your last name (hopefully) on new turing the username limit is 16 characters
→ so on turing I am: mkegel; Marshall is mpierce, Erik Shimshock is eshimsho

• password should be strong
→ what makes a strong password?
→ min 8 characters in length
→ special characters (Numbers and @ ! # $ % ^ & . , ( ) }
   , even spaces)
→ uppercase characters
→ should not look like a dictionary word
→ pass-phrases are good things
→ "I took CS60201d could become -\$1 t00k CS 6o#"
   (thats a one with two zeros in took and a lower case 201802019)
→ notice that no password characters show up when typing

6 Introducing the shell

→ what you see when starting
→ the default turing setup, easiest and most familiar setup
→ components of the prompt: username, time, command number, and directory

• Sessions of the Heart
→ unix is at its heart a cmd line system - in some respects this is much more powerful than having just a graphical interface
→ it was created in the days of the teletype, like a telegraph with a keyboard only uppercase letters though - so very limited
→ so the teletype would be hooked up to the giant room filling computer
→ the teletype would send the input, and the output would be printed out on a sheet of paper, this meant that the teletype and the computer could be located either next to each other or across the country, the computer couldn’t tell
→ later the sheet of paper changed to a CRT monitor - would just print the text to the screen
→ over time the teletype changed to a full keyboard but the fundamental relation between the teletype and computer (input and output) for most things hasn’t really changed
→ obviously today you have both upper and lower case characters
→ most of you probably come from either a windows or mac world where most things can be done with a graphical interface, for certain things a GUI (graphical user interface) can be very good (ex. web browsing), however, certain other things are made much simpler by the use of the cmd line.
→ for you, however, the best reason to learn the command line is that you have to
→ the CS department has chosen to have students do most of their work on unix systems, therefore to make your life easier you should learn how to use them
→ Why should you learn to use the command line? What does it offer?

• The Blinking Cursor of Doom
· simple, quick, and powerful
   ➤ what does that mean?
   ➤ can do an enormous amount of work, with very few commands
   ➤ for certain tasks are much superior to a graphic interface
   · becoming a unix power user involves becoming familiar with the cmd line
   ➤ we will focus this course on using the command line
   ➤ for now we want a simple prompt, don’t want to distract you

• Home sweet Home
   · when you log in you will automatically be at your home directory
   · home directories on turing, and many other systems (OS X is different) are located at /home/username
   · on OSX home directories are at /Users/username
   ➤ for me: /home/mkegel (turing, farmboy)
   /Users/mkegel (shadow)
   · directories are denoted with /, windows uses
   · there are other directories that we will get to later, but on turing you should never have to worry about them

• Unix directory tree
   ➤ what do i mean the home is /home/mkegel? no c:?
   · file systems can generally be seen as a tree
   · a cdrom or hard-drive can generally be located anywhere in the tree
   · no preset structure
   · working with devices we’ll get to much later so just sit tight for now
   ➤ there is no c, d, a: like on windows, everything starts at / (root)
   · is filesystem independent

• Introducing the Shell
   ➤ what is the shell? talked about multi-layered design but what is it?
   · The shell is the highest level interaction layer between you and the OS
   · it’s what sits between the teletype and the computer - makes the computer easier to use
   · it runs whatever commands you give it, with whatever arguments you give it
   · the most important thing in learning to use unix in a productive manner is to become comfortable with using a command line
   ➤ that’s not a computer fan, that’s the shell purring...

• syntax of a command
   · will use
   · things in brackets optional, angle brackets not optional
   ➤ that is how commands will appear and be used on the command line

• switches
   ➤ What is an argument and what is a switch?
   · switches are arguments that you pass into a program that modify the programs behavior - switch the behavior of the program
   · sometimes called flags, we’ll use the term switch though
   · different than what is generally referred to as an argument
   · a switch will generally have the form ”-?d where ? is some character
   ➤ most programs take multiple switches and also accept multiple forms for what is the same action
   · other style, GNU style, is ”–switch”
   · most programs accept many switches
   ➤ gpg (encryption key generator) accepts 360!
   · can have multiple switches, example tar
   · tar (tape archive) is an archiving program from the days of tape drives
   · file archives are given the extension ”.tar” (gee big surprise!)
   · to uncompress will most often type
   · ”-xvf” could have been typed ”-x -v -f” but easier to write the first way

• Arguments
   · data that you want to program to operate on
what is an argument? what would you pass in?
· filenames, text strings, man pages
· all of this stuff is sensitive to spaces, so "ab201d is not "a b201d
· not every command takes arguments; some require them
· some basic commands to get started with
  - these commands are more complicated than what we are actually showing here, you2019ll be able to find out more stuff about them after the end of this lecture
· these are also the commands that you2019ll be using the most often
· students are going to have to write all of this down, will probably want to put on the chalkboard
· cd [directory] :: changes the current directory
· pwd :: prints the current directory you are in
· mv [source] [target] :: moves the 2018source2019 file to 2018target2019 file/directory
· cp [source] [target] :: moves the 2018source2019 file to the 2018target2019 directory
· rm [file1] [file2] ... [filen] :: removes files
· ls :: gives you a listing of the files in the directory
· mkdir [name] :: creates a new directory
· rmdir [name] :: removes an empty directory
· man [command name] :: prints the ”man page201d for various system commands
· some commands are interactive
· all these programs just run, do their thing, and then quit
· what we generally refer to as an application is interactive
· emacs and the shell are both interactive programs
· some things to know
  · note: unix is case sensitive - AAAA is different than aaaa - this shouldn2019t be too surprising
  · also most unix commands are done in lower-case as you2019ve just seen
· most switches to programs will also be in lower case, although some need to use both
  - pay careful attention to both the switch you2019re using and the case of the letter(s) you may be passing in
  - there are also special characters will in different contexts will be interpreted differently
  · Ctrl+World = Mwahahaa!
  · discuss C-? and M-? notation
  · you see this everywhere, particularly in documentation, so is very useful to know
  · is incredibly simple: C == Ctrl; M = Alt (but stands for meta)
  · therefore C-x is Ctrl and lowercase c held together
  · are what might be called shortcut keys under windows
  · one other notation is caret notation; ^ would mean C-h
· quitting programs
  · most programs can be quit through either Ctrl-c or Ctrl-d
  · C-d sends an EOF character
  · C-c sends an interrupt signal that causes the program to clean up and then to exit
  · if C-c doesn2019t work and you have to quit the program, then C- will cause the program to exit without cleaning up first (as well as dumping core)
· tab completion
  · wouldn2019t it be nice if say you have some long directory name, ./thisdirectorynameis-tooolong/ but there was some way that you didn2019t have to type it out every time, hmmm...
  · most shells (and any that you would normally use) provide whats called tab-completion, allowing you to complete partially written statements into their full form
  · so given partial data the shell will determine what sequences of characters will appropriately complete what you2019ve written on the cmd line
  · this is accomplished by hitting the tab key
once you’ve typed enough to make the rest of the string unique (that is there is only one possible thing that you could be typing) the shell will automatically complete what you’re typing when you hit tab

since directory names have to be unique on the same level, tab-completion work really, really well

but what if you want to tab complete other things?

like say the arguments to a functions, or switches...

luckily this is setup by default with the shell you’re using, so you can tab complete any number of things, man pages, arguments, etc.

• files and directories

  up arrow gives old commands, continue pressing to work back through your entire command history, useful for when you have to type the same commands again and again - like

  the down arrow with of course cycle forward in your history

• command line editing

  the left and right arrow keys move the cursor allowing you to edit the cmd you’ve typed on the screen - that way you can go back and correct something without having to retype the entire command

  other useful key commands are (assuming they aren’t already used):

• special characters

  the shell interprets a number of characters as being special

  that is some characters have meanings to the shell other than what you may have typed

  so which characters are special?

  what do you need special characters for?

  for example the semi-colon ; separates commands on the cmd line

  but what if you want to pass a special character to a program?

what is it you want to pass these characters into a program?

• have to escape them;

  ?, where ? is the special character you want to use

  echo semi; versus echo semi ;

  so when you type ; the special meaning of ; is disabled

  you can also disable the special meaning by surrounding the character in quotes, either ” or 2018 (double or single quotes)

  demo echo ”; 201d

  how quoting is more powerful since all the special characters inside of the quotes get disabled - well mostly, there are some rules

  pg. 527 UPT

  the character has another meaning that you have to worry about

  if you have a at the end of a line then the new line will be removed and the shell will treat the two lines as one long continuous line

7 Finding help

• Finding Help

  our goal is not to teach you unix, but to give you the basics, then have you be able to teach yourselves what you need to know

  but sometimes you have to ask for help, and often times this is the best way to learn something new, don’t beat your head against the wall if someone you know already has the answer and can help you figure it out!

  so part of teaching yourself is knowing when to ask questions

  there is one resource that is the best and it is? ...Google!

  whenever you have a question about something the first thing that you should always do is google for the answer - most likely someone else has already taken the time to solve and write up whatever solution you are looking for

  lots of very good resources
8 How to use man

- how man is organized - sections - explain what each section covers
  - 1 - Commands available to users
  - 2 - Unix and C system calls
  - 3 - C library routines for C programs
  - 4 - Special file names (Devices and Device Drivers)
  - 5 - File formats, protocols, and conventions for files used by Unix
  - 6 - Games
  - 7 - Conventions, Macro packages, Word processing packages and Misc.
  - 8 - System administration commands and procedures

- always need to keep in mind that one command could be in two+ different sections at the same time
- allows for a better organization of documentation

- some switches
  - so then "man -k" allows you to search for man pages
  - "-s" specify the section to search
  - "-a" show all man pages that match the search, not just the first

- how to read man pages - the parts of a man page
  - at the very top is the is the name and section number
  - NAME - just a quick one line description
  - SYNONYMS - all the arguments that the program accepts
  - DESCRIPTION - a paragraph "summary" of the program
  - OPTIONS - what each of the switches to the program do
  - ENVIRONMENT - what environment variables the program expects (something we'll cover later on)
  - SEE ALSO - other related man pages to see
  - BUGS - bugs the program may have
  - FILES - files that the program may use (usually configuration files)
  - AUTHOR - who wrote the program

- navigating a man page
  - man by default uses more (more is a pager that sucks)
  - on turing this is fixed, but be aware!!
  - so learning to use man is really like learning to use less
    - arrows move man page up/down left/right
    - space bar moves a screen at a time
    - to search would type /pattern, then press enter
    - demo this feature
    - typing / or n again moves to the next instance of pattern
    - N moves to the previous instance
    - b : back - like page up
    - f : forward - page down
  - man almost always uses the less program as its pager, so learn to use less!
  - you’ll also use less for lots of other text viewing

9 Making programs work together on the cmd line

- Working Together
  - unix is about more than just running one program at a time (that's windows job); in unix programs work together, so we'll show you how to do that
  - How do you get programs to talk? Files? Pass in a program as an argument? What should handle this mechanism? What about basic program I/O?
Already know that can accept arguments and print stuff to the terminal.

Text streams are a universal I/O format

- Working Together
  - unix runs more than one program simultaneously
  - programs need to work together
  - programs use stdin, stdout, stderr for general I/O purposes
  - by default I/O comes from cmd line

- What stdout, etc do
  - you should already know about stdin, stdout and stderr from CS70 or CS60
  - they allow a single program to get input from the cmd line or print output to the cmd line
  - stdin - represents the characters you are inputting
  - stdout - the standard place for the program to print output
  - stderr - like stdout but for error output
  - in C++ the three are accessed through cout, cin, and cerr
  - can accessed in other ways through other languages
  - so we have some idea of what stdin and stdout are, they represent the input we get from the terminal and the output we send to the screen, but how do you think that they actually work?

- Everything in Unix is a file
  - devices (keyboard, mice, monitor)
  - directories (yep even directories)
  - links (we2019ll cover what these are)
  - and what you normally think of as files
  - what does it mean to be a file? anything used with read() and write()?
  - put quotes around file -¿ "file", aren’t really files, just treated like files
  - a file is just a stream of data, in some cases with limitations as to how that stream is handled

- Are they file?
  - so stdin, stdout and stderr are files? no, but are treated as files
  - are actually file descriptors, so being treated like files
  - why so surprising? you read() from stdin, write() to stdout and stderr
  - each program has its own set stdout, stdin and stderr that is managed by the Operating system
  - how do you combine simple programs? how do you get two programs to talk?
  - we know that two programs can share the same file - can have the same
  - so to communicate one program could write to a file, the other could read
  - [program one] −¿ [file descriptor/pipe] −¿ [program two]
  - can use the stdin, stdout, and stderr files to have programs talk to each other
  - [program one] −¿ [stdout/stdin] −¿ [program two]
  - the stdout of one program IS the stdin of another
  - very flexible and powerful solution

- Using pipes
  - stdin, stdout, and stderr are actually pipes
  - instead of the program printing its output to the screen it is /piped/ to the second program
  - keyword: pipe
  - you can do this from the shell
  - the syntax is
  - the 2018—2019 character is known as the pipe and is located on the key, so Shift-
  - programs used in this way are called 2018filters2019
  - filters are the tools that you use to make tasks easier, and you need them to be good
  - if you can’t use filters you’re never going to be able to exploit the power of the system
  - cause you won’t be able to leverage the work of other people
  - how many people think that they could implement ls? how about a good implementation?
  - to make them good you have to follow a certain set of standards and conventions
if you don’t the filter will be difficult to use
so what are these conventions? we discussed many of them at the beginning of the class
KISS, principle of least surprise, you’ll learn most of these just by using a UNIX system

• Redirecting Output
  can do more than just piping between programs
  • you can redirect the output of a program to a file
  • you can redirect a program to read stdin from a file
  keyword: redirection
  • you can have whatever file you want to be stdin, or stdout/stderr, using the shell
  • redirect file to stdin
  • You use the j to have file be stdin for the program
  • redirect stdout to file
  • So you use ; to redirect the output
  • to send stderr to a file do
  Some notes on redirecting output
  • You can append to the end of a file (add lines of standard output at the end without writing over what is already there, useful for log files) by doing the following

• Working with stderr
  • word of warning: this is very shell dependent, but since you are working with zsh, or you ought to be the syntax provided here should work
  • send both stdin and stdout with
  • can also append stderr to a file - better for log files since you don’t generally want to send error messages along a pipeline, only want real output
  • to append stderr very much like redirecting stderr
  • to redirect just stderr to a file
  • redirecting just stderr along a pipe is a bit more difficult

• Complex Redirection
  • can modify stdin, etc, file descriptors from the cmd line
  stdin=0, stdout=1, stderr=2 : are file descriptors (integers)
  file descriptors 3 to 9 are free for you to use
  examples:
  • not every program takes things from stdin or send output to stdout
  • for example can’t do
  • instead have to send myfile in as an argument to emacs

10 Another way to combine programs

• The Back-Tick
  • you may remember the ‘ as being a special character
  • ‘ is located on the key above tab and left of 1
  • is called a back-tick or back-quote
  • anytime you surround a string with back-ticks the shell will treat it as a command and then replace the string with the output of the program
  • this will restart your ssh server, since ‘cat..’ returns the PID of sshd
  • another example
  • this will let you edit all of the files that have the word error in them
  • lots of other uses, mainly in shell scripting

11 You Ought to Know this by now

• Review
  • Philosophy and History of Unix
  • Piping, Redirection, Back-Ticks
  • Finding Help
  • Users and Groups
  • Some Basic Commands
  • What the Shell is and why its cool
  • Few Other tidbits
  • if you don’t know what we’re talking about here, then you’ll want to review before going on
12 Processes and the Kernel - Actually running programs

- some of the commands we’ve been running haven’t actually been ”programs” in the example: cd is a builtin, its a part of the shell and not a stand alone
- need to have some sense of the basic, underlying abstractions of how the computer works - not on a bits and byte levels, but higher up
- exercise: back to the layer thing, how the computer is organized
- again is a layer thing - bottom up
- hardware - the machine itself - just electrical signals
- kernel - a mediator between the hardware and the user - provides a set (library) of functions for controlling the hardware that every program you would write would need (ex. drawing text to the screen, writing to disk)
- applications - do the things that you want the computer to do
- the user - running applications and making sense of the electrical signals that the computer processes - only here do things have any ”meaning”
- in understanding unix you don’t so much about the hardware - leave that to the engineers
- really only care about the applications layer - how to make programs do what you want
- but in order to fully appreciate and use unix you have to understand at some level how applications interact with the kernel, and what a kernel is

• What is a process, program
  - what is a program and what is a process?
  - ask what a program/executable is - just the file containing instructions
  - is very broad: does it matter what kind of instructions?
  - what is process?
  - how many copies of each do you normally have?
  - why would you have more than one process of a single program?

• The Kernel?
  - how do the processes get run?
  - does the kernel just run processes? basically, and do what processes need done
  - take Operating Systems if you want to write a kernel
  - take CS105 to learn more about how Computer Systems work
  - as users all we care about is running programs, leave the other stuff to the CS geeks
  - and unix is very good at running programs
  - so what does this have to do with anything?
  - not a whole lot other than making you understand that you can run multiple programs with fear of them crashing the system like on some other systems we could mention

• Making Forrest Run
  - unix is a multitasking environment
  - unix is a multiuser environment
  - so if you are going to run multiple programs from the shell(s) it would be helpful if you knew how to manage them
  - we’ll assume one shell and that you need to run multiple programs
  - is called ”job control”
  - why would you need this even though we have GUIs?

• Making Forrest Run
  - three types of processes: foreground, background, suspended
  - foreground process is the one that receives keyboard input (if interactive) and prints to the terminal
  - sometimes you want to run other programs
  - say stop reading a man page to copy a file, or read email
  - background process don’t receive input, but can still print to the terminal
  - suspended processes just aren’t doing anything - but are not terminating

• Sending the Right Signals
  - recall that C-c (SIGINT) kills the program you are currently running
· so does C-
(SIGQUIT)

→ sometimes its C-d (sends an EOF – usually for interactive programs)
· to suspend a running process use C-z
(SIGTSTP)

→ common mistake to think that the program has just suddenly died
→ people unfamiliar with the system will just start over, being angry and frustrated at the work they "lost"

• Background Processes
  · when we type
    · to run a program in the background we type
      · this causes the program to run, but not in the foreground - so it won't be receiving any input that you type unless you switch to that program
  
  → just because a process is backgrounded does not mean it isn't running, and that suspended process aren't running
  → you could have a copy operation running in the background
  
  → music program in the background
  
  → also note that the system has to run all these, so no free lunch!

→ remember: limited by RAM, hard drive speed (swapping), and cpu power
· to switch back to the process you just suspended you use the fg command
· can of course suspend multiple processes
· demo two other man pages, also suspend
· will have three jobs in the background with cmd line open
· open emacs in background and switch to it

→ this is one of the most useful things to know, so remember it!
→ why this is useful - give this example scenario

→ really only need to remember C-z to suspend and fg to bring the process back

• Mid-level management
  · use the jobs command to list all the jobs that you have running
  · the number identifies each job

→ we use the fg command to switch back to which ever job we want using the number
→ the rest of the output of jobs is obvious
→ example to go back to man perl we would type

→ nifty trick: if we know the name of the process, say we have a single emacs process along with other can quickly switch back to emacs by using a pattern matching scheme for fg
→ instead of
→ so returning to emacs could be as simple as typing

→ don't have to use jobs and look there, just do a single fg command
→ can also do some of the same tricks with bg
→ continues job n in the background

→ now that we can manage a few processes what about the processes on the rest of the machine?

13 Understanding process ownership

• Process ownership
  · every process on the machine has a parent/child status
  · so when you log in and get a shell that process will act as the parent to all the other processes you create on the machine
  · there is one master process on the machine, init
  · every process is a descendent of init
  · every orphaned process is a child of init
  · a zombie is a process which has exited but hasn’t been reaped by its parent
  · draw out a parent/child diagram from init down to the shell

→ parents can query for the exit state of a process

• Process Ownership
  · each user on the system can own processes, so when you log in the shell that you see is owned by you
  · to identify all the processes on the machine each process is given a PID or process id that uniquely identifies the process
you use the ps command to see the processes that are running
- run just by itself (no switches) will just print out your process that are being run from the terminal you logged in on
- to see all processes
  - can see what everyone is running with this
  - so why is this useful?

- How to be like Charlie Manson
  - what if we want to kill a runaway process or any other process
  - imagine you have a java application in an infinite loop
  - don’t want it running forever, so want to kill that process
  - we use the kill cmd to do this
  - two ways to kill
  - also demo
  - we can use ps to find the process id
  - but how did we know that the process was out-of-control in the first place?
  - nifty utility on unix called top
  - will list out the processes in order of cpu time
  - can use this to see if any of your processes are out of control
  - top is made more useful by seeing only the processes that you own
  - first start top
  - then press the up key; then type your username, or another username if you want to see their processes
  - then enter
  - to get help on the other commands top offers use the key when running top
  - to quit top press the q key or C-c
  - top is really nifty, so spend a few minutes trying to figure it out
  - digression: what would happen if say, we were root and we typed
  - what would happen? is that even referring to a process?
  - in general killing a parent does not kill child, child is orphaned and becomes a child of init

- Process priority
  - what is process priority?
    - priority defines how much time the particular process should get from the machine
    - processes with a low priority gets run less often than a process with a high priority
    - as a user you can actually change the priority, only root and the scheduler can do that
    - what you can do is set the *niceness* of the process
      - a higher nice value means that the process is nicer to other processes on the system, giving them more time to get stuff done
    - so a high nice value leads to a low priority and a low nice value leads to a "normal" priority
  - to set the niceness of a process you use the nice cmd
  - if you are going to be running a long and fairly intensive job on turing please use, and be, nice!
  - nothing worse than locking the system up for fifty people while you try to compute prime numbers ;-)
14 Some programs to remember

→ this is a list of programs that we consider to be the most useful or that are basic to being able to use a unix system

→ *locate, echo, cat, clear, less, head, tail*

→ locate
  → allows you to find files on the system
  → has to search a database so not always up-to-date
  → usage
  → accepts wildcards for files which we’ll get to later but which you may already be familiar with...

→ echo
  → prints whatever arguments that you give it to standard output
  → useful for when you want to write/append some string to a file
  → or just when you want to write to stdout
  → usage
  → before using you should really read the man page (it has some very interesting options)

→ cat
  → like echo, instead takes file names as arguments
  → prints the contexts of each file in order to standard out
  → one of the more useful commands, especially if you want to look at a small text file
  → cat is short for concatenate - you may think they should have used "cons2019d"

→ usage
  → if no file is given will take its input from stdin
  → end the text stream with C-d
  → will then print back to stdout
  → most useful when combined with redirection and piping

→ clear
  → clears the screen (also C-l) - useful for when your terminal has filled up with junk and you’d like to make it go away

→ more
  → is a pager
  → sometimes you have more text than can be fit on a single window, so need to way to see the entire text of a document
  → more lets you view the entire text of a document, but is limited in that you can’t scroll up
  → use the space bar to advance to the next screen

→ less
  → less is also a pager
  → less is a better pager than more however
  → everyone knows the old adage "less is more", there is also another pager called most
  → most ¿ less ¿ more
  → have already had exposure to this program through the man command
  → man opens up its man pages in less (not by default but generally)
  → but of course can use less to view more than just man pages

→ usage
  → and some of the key commands to make less do what you want - interactive
  → arrows move man page up/down left/right
  → space bar moves a screen at a time
  → to search would type /pattern, then press enter
  → typing / or n again moves to the next instance of pattern

→ head
  → N moves to the previous instance

→ f : forward - page down

→ b : back - like page up

→ prints out the first n lines of a file to stdout

→ useful for when you have a sorted list and want only the first n

→ by default (if n is not specified) will print out the first 10

→ usage

→ tail

→ like head, in that it will print out the last n lines instead of the first
more useful in that files, especially log files, are often appended to

can then follow these files for as long as they are open and see what is being added to them

usage

"-n" : gives last XX lines of file

"-f" : keeps file open

"+NUM" : shows all but the first NUM lines of a file

the next two are both searching commands - one is for searching through stdin, the other for searching through a directory tree

grep

allows you to search for a regular expression within a file

often though you just want to find a string within a file, not everything matching a certain pattern

this is really just a copout since we don’t want to show you regex until later

so to search through a single file

grep has some useful switches that make it nicer to work with

"-c" : instead of printing lines that match will instead count the number of lines that match and print that instead

"-n" : print the line number on which the pattern was found

"-H" : prints which file the pattern was found in

"-r" : if a directory name is given will recursively descend through the hierarchy looking for matches in all the files in that tree

there are also different versions of grep, so you’ll want to check out the grep man page, as usual

find

at its most basic level find just descends through a directory tree and prints out the names of whatever files it finds there

so you can replicate (in functionality but not speed) the locate command with

this is generally much slower than locate, but has to go through the entire file system starting at ./ (which is where ever you currently are)

but find is much more complex than this and can do locate all on its own

the ",-name" switch will search for the glob, and "-print" will print

find can of course do much more interesting things

try to find out what this does, and why you would need it

w

prints out who is on the system

which

prints out which command exactly will be executed when you just type cmd

useful for when you want to figure out either which program you are using or where the program you are using is

du

displays the amount of space that a file (or directory) is taking up on disk

to get the output to be a bit more useful, will want the "-h" switch

if file is actually a directory, du will recurse through the directory printing out the size of each file as its gets them finally printing out how large the directory is in total at the end

this is not very useful, so pass in the "-s" switch to suppress the extra output

this creates a useful "sizeof" command

du -sh file1 file2 file3 ...

df

def

shows a detailed listing of the amount of space being used on each file system that is mounted

made nicer by the "-h" option

15 Files, programs and file, and the filesystem

what is a file? some review from before

a file is a stream of bytes

traditionally think of a file as something like a text file or mp3

in unix though we know that everything is a file: directories, devices, etc.

review the unix security model
users have a home directory

users not allowed to do random (dangerous) things - only allowed to do those things that they have permission to do

there is a superuser that can do anything

users belong to groups, and have a primary group

• Files
  ➤ how should an OS handle security for files? how do you tell who owns a file? what’s a good way to manage sensitive files?
    • files have their own security model
    • files have an owner and a group that they belong to
      • user is you
      • group is the group the file belongs to
      • so when you create a file, the file is created with you as the owner, your group
      • so we’ve defined two categories that a file belongs to, what else is left?
      • what about the other people on the system?
      • so in all a file is concerned with three sets of people: user, group, others
      • others are by definition everyone else on the system
      • besides owner, group, etc., files have permissions associated with what people inside of these classifications can do with the file
  ➤ Files Permissions
    ➤ based on the security model above what kinds of permissions should we have? what kind do we have?
    ➤ so just because you own the file doesn’t necessarily mean that you can write to it, or execute it if its a shell script
      • so what things can you do to a file?
      • well unix defines only three (there are more that could be useful, but unix is old and the spec. has changed with the times)
      • these three are: read, write and execute
      • can set these three permissions for any categories: user, group, others
      • so the permissions look like: rwx—rwx—rwx
      • the first rwx is the user, the second group and the third others
  ➤ now on to octal permissions
  ➤ so rather than writing out a bitmap each time you want to say what the permissions of a file are, what would be a more compact representation?
  ➤ use permissions base 8
    ➤ they are permissions base 8, because each permission group (user, group, others) can be represented by a number from 0 to 7
  ➤ if we write out 0 to 7 in binary we see that we require 3 bits (hmmm!)
    ➤ - - -
      • so these bits have value 4 2 1 and are mapped to r w x
  ➤ r w x : draw out something like this
    ➤ 4 2 1
    ➤ so 7 looks like 1 1 1
    ➤ and 4 looks like 1 0 0
  ➤ combining this we can write permissions out as a group of three numbers 0 to 7
  ➤ permissions look like XXX
    ➤ the first X is user, the second is group, and the third is others
  ➤ examples of common permissions settings
  • Changing Ownership
    • chmod - will also change ALL aspects of a file if wanted
      ➤ switches u, g, o and the += syntax, with perms rwx
      ➤ also cover -R for recursive
  • special files - directories
    • directories are files too, though can’t read/write to it as you would a regular file
    • but because are files, they have to have the same permissions that other files on the system has
    • unfortunately the permissions on a directory act a bit differently (which you might expect)
      • r can list with cd
      • x can access the file within
      • w means can write a file to that directory
  • dot-files
    ➤ so when you type the ls cmd, do you see every file?
obviously not since I’m asking this question
- .a201d switch will show ALL files in the directory
- this is useful because dot-files are generally used to hide configuration information for programs away from the user
- also keeps the configuration information in your home directory so that each user can have their own settings as they like
- some examples of .files
- one common one you may care about is your .zshrc, this controls some of the settings you have in your shell as you login
- another is .emacs - controls the settings for emacs
- you can also have .directories
- these are directories that are hidden
- do the same thing as .files (hide config info), but for programs that may have many configuration files that need to be hidden
- so is it a nice thing that just ls .a201d show all the files in a directory by default? Yes, but is still historical crap
- a better solution might be possible
- so when you do a ls .a201d, do you recall seeing two files one named . .a201d and the other .. .a201d?
  - probably not but these are two special files
  - the file named . is this directory
  - the file named .. is the parent directory
  - other special files
    - tilde ( ) denotes your home directory
    - will take you to your home directory, a plain cd will do the same
    - username denotes the home directory of that user
  - relative and absolute paths
    - can refer to files in two different ways
    - what if you have written a program and want to access a certain file - config.cfg
      - two ways: relative and absolute path
      - absolute paths never change and describe in "absolute201d terms where a file is in the tree
    - absolute paths will always start with /
    - you are starting at the root and working down the tree
    - relative paths start with a ./ - the directory tree before the ./ is not known
- don’t necessarily know where the program will be stored, but always know where the config.cfg file will be in relation to the program
- or should at least know where it will be
- can refer to it then by saying
- the ./ is filled in by the OS or the shell - depending on context

16 Links - symbolic and hardlinks

- so now that we’ve confused you this much what if you wanted to have two files have the same name?
- why would you want this?
- how about when you upgrade software - could have the program named emacs, but the /real/ name might be emacs-21.2, an older version could be emacs-20.7
- you want to continue to access this program through the same name, but have it always refer to the new version
  - File Links
    - unix uses a thing called links
    - links are a special type of file which transparently redirect you to a different file
    - so you can type emacs, but the OS will know that you really mean emacs-21.2
    - there are two types of links: soft and hard links
  - Types of links
    - a hard link, which was what we created above, is literally a different name for the same file
    - the new name is stored in the directory it’s supposed to be in so takes up virtually no memory
    - the hard link literally points to the same data on disk
a hard link cannot cross file-systems (say you have two different hard drives and want to have one file refer to another)
you cannot hard link to a directory
a soft-link is different and more flexible
it is a separate file (not just an entry in a directory file) that the OS interprets as the real file, basically just a type of pointer
soft-links can go across file systems and can point to directories
in practice you should probably almost always create soft-links
can link both directories and regular files

Creating Links
the basic syntax is backwards, so to "link201d emacs to emacs-21.2 you would type
to create one pass the "-s 201d switch to ln
this will create a soft-link
symlinks don't always work, for example when trying to get to things through ftp - have to mount instead

17 File globbing
most people are probably familiar with file globs, but for now, what is a file glob?
why might you want file globbing?
many times will want to work with multiple files
say you want to mv or cp or rm all the files in a directory
how do you do that?
you use a thing called file globbing
is very similar to UNIX regular expressions, but should not confuse them
globbing is provided by the shell
this is key - is one of the many complaints the people have about unix
the shell is responsible for expanding globs
programs generally only take file names and don't generally do pattern matching
why should is be the responsibility for the shell to expand globs? is this a good thing or a bad one?
some people see this as a bad thing - for you you don't have to care
some standard globs - see page 658 U.P.T.
* - matches all files in a directory (not dot-files though) - match anything
? - match one character
[a-z] - any character a to z - character classes
(x—y—...) - match either x or y - this is not pipe!
a,b,... - expands to "a” ”b"
a[a-z]*
?[c]*c
can put these anywhere in the expression
by far the most common is to do
ex1*ex2
*ex3
ex4*
to match dot-files have to put a dot explicitly
to match all dot-files in a directory would do .*
note that this also lists " ..201d and ..201d
so what would happen say if we did this...
as root from /home?
much badness is what...since more than just the down directory will be changed
.. follows to its parent and so you2019ll also change all the files in all the directories at the same level and below in the tree
draw out to illustrate

18 The tree of life - How the Unix directory tree is organized

Unix Directories
now that you understand how to work with files, how to work with programs should know a little bit more about the environment you are working in
unix is in some way a very structured environment
35 years of tradition have specified how you do certain things and why they are the right things to do
and then we come to the way the directory structure is organized
what do I mean by this?
so with windows you were probably comfortable having all your programs be installed to C:
Program Files
Program; so everything you installed was in a common place
what if some one decided one day to instead install everything to C:
Your Programs
Program and c:
My Programs
Program; but all the old ones you had would (by default) install to the old directory (Program Files)
unix has a "unified hierarchy"
this would be quite a pain
in unix the same thing exists, but across the different unices that exist
of course unix is supposed to be independent of the directory structure on which its working, just makes it a pain for people in real life
this is an historical artifact
in order to gain market share one thing they teach you in business school is to differentiate your product
so when unix began to come out in various commercial flavors the companies changed things a bit to reflect how their product was different
the most obvious and easy way to change an OS is to change where you put things
so you end up with apache (a web daemon) being in four different places in four different flavors of unix
luckily the madness has continued!!!!!!
linux suffers from the same problem though perhaps not to the same extent
where things are put in debian is different from gentoo, is different from...
FreeBSD has some resemblance to the linux directory structure similar
to just limit the confusion we'll just go over the basics which apply to most unix systems, except for OS X which is very different
- *now the real material*
so we know that everything in unix starts at /
so how do we decide what goes under this?
we don't get to so much, though if you want create your own distro...
so lets start with something you already know /home

- General Directories
  - /home is where the 661 is
  - in some sense this is true since this is where the home directories of all the users on the system are stored
  - the home directories generally have the form /home/username, though this isn't strictly necessary, just useful
  - rest of this is taken from
  - see also pg 822 UPT
  - /lib - shared libraries and kernel modules
  - /opt - larger static packages - where you would install KDE for example
  - /boot - kernel image and other boot files
  - /mnt - temporary/permanent mount point for devices

19 Some special devices
- Special files
  - /dev/null
  - /dev/zero
  - /dev/random and /dev/urandom

20 Environment Variables and Shell Variables
- Variables
  - these are special variables that all programs know about
  - they exist in the background and let programs know about certain things
  - variable naming convention
• CAPS are environment
• lower-case are shell
• please follow convention!!!

⇒ you generally only ever care about environment variables
  • how do you see what environment variables you have?
  • simple use the printenv command
  • what about shell variables?
  • pg 705 has a good listing of pre-defined environment variables

• Common Variables
  • good ones to know about - will be all caps

21 The Different Shells

⇒ now is a good time to talk about what shell you are using
⇒ there are in fact many different shells!
⇒ you’ve probably been using zsh, the turing default, all this time with ever knowing it
⇒ there are a multitude of shells, each with different strengths and features
⇒ most fall into two different categories
  ⇒ those based on sh, which was the first shell
  ⇒ those based on csh
⇒ the default on turing is zsh, which is a hybrid of sh and csh
⇒ other shells include bash (probably the most widely used shell), tcsh, ksh (Korn-Shell)
⇒ other random shells: ash, fish, pdksh (free version of Korn-shell)
⇒ probably the best (certainly the most fully featured) shell is zsh
⇒ we recommend that you use either zsh or, if you have to, bash
⇒ why use zsh?
  ⇒ its just plain nifty
  ⇒ will tab complete many very cool things, and even not so cool ones
  ⇒ e.g can tab complete kill and man for example
⇒ also we are only going to demonstrate for this shell, you have the tools to figure out how to do the same things for yourself on your own shell
⇒ so how are they different? what really distinguishes a shell?
⇒ in terms of user interface, shells are almost identical
⇒ shells provide the ability to run more one command at a time
⇒ if you string commands together in a file, with some control structures like if, while, and case statements: you get a language
⇒ called shell scripting
⇒ fundamental different between csh and sh - csh is more based on C syntax
⇒ sh is just funky and has a weird syntax, but for shell scripting is better (better control over output and input)
⇒ so when we say a shell is based on another, what we really mean is that zsh or bash can run scripts written for sh
⇒ and that zsh and bash offer scripting options that weren’t available with sh
⇒ the same for tcsh
⇒ we won’t show you shell scripting yet, but you will learn about it

22 Setting Environment Variables

• Setting Env. Variables
  ⇒ to set environment variables in zsh
  ⇒ can also do this
  ⇒ you can see the output of a variable by using the echo cmd
  ⇒ have to remember the
  $ 
  ⇒ the most common thing to do is to add paths to PATH
  ⇒ from the cmd line for zsh we would type
    • or -
  ⇒ the two are different!
first puts the new path at the end
the second puts the new path at the front
is important because when searching for the cmd to run, the shell will take the first one that matches, so you want to make sure its matching the right cmd
unfortunately you would have to do this every-time you logged in
would get boring

• Setting Env. Variables
  → maybe there is an easy way fix this problem?
  → real answer: this is why you have dot-files
  → we know dot-files hold configuration information
  → zsh is no exception to this rule
  → zsh has several files that hold login information
  → the one that you care about is .zshrc
  → once you change your .zshrc it won’t immediately take effect
  → either you have to log out and restart your shell or you can apply the change from the cmd line
  → to apply from the cmd line use the source cmd
  → will cause the shell to read the configuration information, and apply the changes
  → but this doesn’t cause the shell to reset its old configuration, it just overwrites what it changes
  → so your PATH will be two copies of your old path followed by the added directory assuming you put it at the end
  → not always what you want, and shouldn’t cause any problems unless you are doing some funky things in your .zshrc or .zlogin
  → *changing the prompt*
  → well if you can change your PATH what other things can you change?
  → lots of things, but the most fun to change is the prompt
  → demonstrate some nifty prompts in zsh and how to change them
  → *other shells options*
  → shells are complex beasts and offer loads of options, one of the most annoying is the shell beep
  → to alert you to an error when you mistype something, the shell will cause the computer to emit a beep
  → this is the most annoying thing ever
  → to turn it off we would use the setopt command that zsh has
  → we would put this line in our zshrc; by default it should already be there
  → there are lots of options so check them out!

23 Using alias in the shell

• ALIAS
  → it is a builtin cmd that allows you to "alias" one set of commands to another
  → so you can refer to a complicated, or repetitive, command with a smaller, easier to type alias
  → you always want to use these when you are typing some long string over and over again that you cannot just tab complete
  → is very much like a macro
  → what is one common thing that you might be typing?
  → how about "ssh username@turing.cs.hmc.edu"
  → so to create an alias we need a name: how about sshtoturing
  → from the cmd line would type: (spacing is important)
  → this alias is, of course, only good for so long as the shell is open
  → once you quit the alias is gone
  → the basic syntax is
  → but what would be a good name? something better than sshtoturing
  → the whole point of the alias (for users) is to make your life easier
  → so want to choose a simple name that is easy to remember
  → how about: turing
  → says what you want, is simple and short
  → what would have happened though if we aliased this to ssh?
question of precedence...and the alias wins
in zsh and bash when you type something
on the cmd line that you want to be taken
as a cmd, then if that text string has been
aliased to something else
is this a good thing or bad thing?
used in the appropriate places is OK
in general, however, is a bad thing because
then you get used to a potentially broken
or misleading command action.
almost always want to use a new name
one example is ls
by default what you get on turing is not
/usr/local/bin/ls
instead you get "/usr/local/bin/ls -F"
so how did I know that all of you had this?
for certain programs you’ll want the
same behavior every-time you login
so to make an alias more or less permanent
you would put it in your zshrc
so if you wanted the alias we talked about
before you would add this line
spacing is important so follow closely

24 More about zshrc
so the zshrc can set options and aliases for you,
what else can it do at login?
the at login is important since you can do this
all from the cmd line; just gets old if you want
it every time
creating files and the umask
by default every permission is ON, so its
your job to turn them OFF
the umask is the thing that does this
written out in octal permissions (remember
theses?)

25 UNIX programs for Windows geeks
there are many different tools that unix offers
you should now know how to make them work
together and some of why they work as they do,
but don’t exactly know what tools are out
there

26 su and sudo
up to this point have been fairly general, inform-
ation applies to almost all users of a *nix sys-
tem
this information is really only useful to those
who will using Linux on their own system, not
so much for those people who are just going to
log into turing
sudo and su
discussed before how use
su allows you to assume the role of the super-
ruser (aka root; is default behavior) without
having to directly log in as root
su \username\
allows you to become that user (changes
your UID and GID), but have to know their
password
if you have first become root, you can do su
\username\ without a password
sudo allows you superuser powers without having to become root - more fine grained control than with su

- sudo is flexible, you can have limited powers, or have the whole shebang
- sudo privileges are controlled in /etc/sudoers
  - is a regular text file, like almost all configuration files on unix
  - can add powers for users, or groups
  - generally only modifiable by root (kind of a security risk otherwise)

→ can also get a shell out of sudo by using the -s switch

• editing /etc/sudoers
  - the basic format is: user/
  → see page 992 of unix power tools
  → would you ever want to let a person run emacs (or vi) with sudo? no: they would have access to a root shell...bad thing need to watch out for? reason why you would want: let them edit system files
  → why might you need to use sudo and su? what kinds of things should be be an administrator to do?

27 Mounting partitions

→ this information is really only useful to those who will using Linux on their own system, not so much for those people who are just going to log into turing

• mounting partitions
  → when managing a system you will almost always be modifying "files" that "normal" users should never touch
  → these files are generally configuration files
  → one of the most basic things to configure on a newly installed linux system are the partitions and how the hard disk will be managed
  → in windows separate partitions would just magically show up (not always, but most times)
  → more too it than this, but the management of the disks was done mostly by the operating system
  → also in windows the most the you would really ever do is put the disk/partition at G:, H:, or so on
  → other devices were located at D: (generally a cdrom or dvd drive)
  → A: - floppy drive
  → C: - your main disk - the disk you booted off of
    - these "locations" are known as mount points
    - in unix you don't have a:, c:
    - instead disks and other devices can be "mounted" most anywhere
  → that is if you have a newly installed hard drive you have to decide where in the filesystem tree that drive will be accessed
  → this is what makes the file system(s) transparent to both users and programs
  → generally mount stuff in /mnt, but can be anywhere
  → can also mount what you might consider important directories from other drives
  → for example: /home on my system is another partition that I created for the express purpose of holding everything that I would want in my home directory
  → / holds everything else I need for my system, but is separate from what I have in my home directory
  → this is a nice because it is very flexible and completely transparent to you and to any program that you might run
  → one example of this being done is with each of your, and mine, home directories on turing
  → your home directories are mounted locally on turing when you log in, but where they were before that doesn't really matter
  → obviously turing can't have enough physical space to hold everything that we might want - home directories, staff accounts, projects, random data, backups
  → all this disk space is provided transparently across a network through something called NFS or network file system (or No Fucking Security, nfs assumes you are on a trusted network)
  → so your home directories are located on one machine, while you are working on turing and doing things to your files, editing, compiling, etc.
→ however that was just a setup for what you’ll likely do on your own machine

• mounting devices

  • in linux, we won’t worry about bsd or other *nix, devices are located in a /dev like we talked about before
  • a device is of course just a file - is actually a special file called a device node - tells the system where in the system the device can be found (for PCI devices major and minor number of device)

→ device node != regular file or directory

→ one thing to note: the linux 2.4 kernel is different from the 2.6 kernel in one major way - the 2.6 kernel does use the /dev filesystem approach to devices; 2.6 is just more intelligent and doesn’t enumerate every possible device

→ generally a device will be owned by root and in one of several groups: disk, video, audio, root, etc.

→ hard drives are generally given the designation hdXN, where X is [a-f] and is the drive and N is [0-9]
  • scsi drives (like what you’d have under firewire) are denoted the same as hard drives except with sd instead of hd notation
  • another would be /dev/input/mouse0 (on my system at least) for the mouse

→ most often though will deal with hard drives and partitions or other comparable (NFS) things

→ so then how do you get a partition mounted?

→ easy with the mount cmd

→ mount is fairly smart, can usually figure out which file system is on the device

→ note to windows users: Most any computer with XP will use NTFS which is a filesystem that linux can only read (trouble writing)

→ FAT32 will work just fine

→ to use mount you need a directory on which to mount, so if we were to mount /dev/hda4 on /mnt/windows, then windows would need to be a directory under /mnt

• mount command

→ also to use mount you have to be root or have root powers (sudo)!

→ to specify the file system use the -t switch like such

→ to un-mount the device use the umount command like such

→ also cover –bind (GNU, different on BSD) - can bind one filesystem to two points - useful for when symlinks won’t work (networking stuff)

→ when you are setting up a system its quite helpful to know which tools are available

→ most systems you get will either come with windows installed, in which case the entire hard drive is one windows partition

→ or you got a new hard drive

→ (Windows Case)

→ You’ll want to get your hands on a program called Partition Magic

→ another good program is GNU Parted

→ if you know where to look on the network (ffs.muddstudents.com *ahem*) you might be able to find the program

→ it can do most anything that you’ll want, including converting, moving and resizing Windows partitions

→ you probably shouldn’t let Partition Magic touch any linux partitions

→ (New HD Case)

→ you’ll want to use cfdisk or fdisk to create new partitions

→ use mkfs to then format the partition

→ (General Case)

→ the cmd df will show how much space is available on each mounted filesystem

→ the cmd du will show you how much space something is taking up

→ useful alias

• /etc/fstab

→ So I mentioned that I have /home mounted from another partition

→ this happens on boot, but how?

→ /etc/fstab is a configuration file that lets the system know which devices are to be mounted, where, who can mount, and with what other options (executable flag being one)

→ there are other things in things in fstab than just hard drives, but you can ignore them
to modify fstab you have to be root (wouldn’t want random users messing with this)

- fstab tells the system which filesystems are to be mounted at boot time
- also lets the system know which mount points the system has
- so say I wanted to mount windows then all I would have to type would be
- and the system would auto-magically know what to do
- you also use fstab to mount NFS exports, but we won’t cover that here
- do not fubar this file, will make you unhappy

28 Using cron

- cron is a daemon that runs in the background and schedules tasks

- such tasks might be backing up a hard drive, updating the locate database

- this configuration information is stored in a crontab file

- generally you just give crontab a file containing what you want done and when - this can be done as a regular user

- also, edit the crontab with crontab -e

- mysched will then contain single line entries for each task that you

- each line is formatted like this

- the username is only there in BSD systems, but check your local docs first

- you can give a single number (7), range (3-8), list (5,7,9), or * to indicate all valid numbers for a given entry

- mins: 0 to 59; hrs: 0 to 23 (military time); day: 0 to 31; month: 1 to 12 weekday: 0 (Sunday) to 6 (on BSD it goes from 1-7 with 1=Sunday)

- cron can also make your life easier by just running things on a consistent schedule - say you need to run things on a hourly, daily, weekly, or monthly basis

- cron will run scripts that you put into /etc/cron.xxxx/

- where xxxx is daily, hourly, etc

- on those intervals

- you’ll have to be root to put stuff in these directories though

29 Wine - Windows ”emulation” for linux

- use wine!!

- there isn’t much else to tell you

- install wine

- do this from the cmd line

- hope for the best; does not work for every application

- if you want to run games (or itunes, or quicktime) then you’ll need a thing called cedega

- there is good documentation, you just need to find it

30 Cygwin

- can get lots of GNU programs

- can get X11

- runs on Windows

- get a poorly implemented version of a terminal

- but can do unix-y type stuff on windows if you absolutely have to

31 Sharing files under linux

- use a package of programs called samba

- allows you to share and access windows network shares

- really easy to use but has issues with some of the newer versions of the windows sharing protocol - may not be able to access all files

- better yet why don’t you just use ftp?

- ftp is much easier to setup, and you don’t have to dirty yourself with Windows slime
32 Emacs and vi

- please use emacs, or if you have a graphical environment something like jedit
- vi should only ever be used in the most desperate of circumstances
- emacs and vi
- not our goal to teach students to use these
- and besides by this point in the semester you should already know how to use emacs

33 More topics to talk about

- shell scripting
  - go read Unix Power Tools
- perl and python
  - both powerful interpreted languages
  - easy to learn and use; generally only need to know one but should learn at least one
  - checkout these sites:
  - cannot stress how useful knowing a language like one of these is
  - learn it!
- why ftp is a bad idea - any password send in the clear

34 Things I left out

- speaking of other frustrating things also mention the C-q and C-s control flow "bug"
- C-q resume console I/O
- C-s stop console I/O

35 Regex Info

- metacharacters:
  - . * + ? [ ] ^ $ \ { } | |
- POSIX classes: [alnum:] [digit:] [blank:] etc
- two main commands that we’ll cover that use regular expressions: sed and grep
- grep just searches for the given pattern
- sed can do that too, but often sed is used to replace things
- Note that grep 'fo+>' is equivalent to grep 'foo*'
- sed is generally used for substitutions. There are two really handy commands for sed: s(substitute) and d(delete). Coupled with addresses, you can do some really cool stuff.
- By default, sed applies the commands to every line. You can change this by specifying an address pattern to use. Patterns can be line numbers, regexes, or things like
- $%

- No address -, apply commands to every line
- 1 address -, only on lines that match the address string
- 2 addresses -, only on lines between the first line that matches the first address and the first line that matches the second address
- You can negate addresses by appending a !.
- So, just 'd', without an address, will delete every line.
- 1d will delete just the first line.
- 1,5d will delete lines 1-5.
- $d will delete the last line. This is not the same as
- $ = end of line.
- /\$/d will delete every blank line.
- 1,/^$/d will delete from the first line until the first blank line.
- All the addressing stuff also applies to the 's' command, but to clarify exactly what 's' does, we won’t mix addressing and s.
36 tr info

> tr is very simple. It takes stuff it finds in the first argument and replaces it with the stuff it finds in the second argument. It does take position into account: something found in the 17th position in the first argument will be replaced by the thing in the 17th position of the second argument.

> It’s also handy for deleting newlines or some other character you may have a particular hatred of: tr -d ‘n’

> This is what you would do if you really hated your grader

> strip out all comments

> remove all newlines

> get a failing grade on that assignment

37 screen info

> screen is basically a way to have multiple terminals inside one standard terminal window. (Think job control, but instead of pausing one job to go back to the shell, you could simply have a bunch of shells running, only one of which was in the foreground at any given time.) In the days of large resolutions and good terminal emulators and window manager, screen isn’t that handy for doing stuff on a local session. It is, however, extremely useful for managing shells on remote sessions.

> Make a new (default) session:

> Resume the default screen:

> Make a new screen session with a title:

> Resume foo:

> If you’ve got a un-resumable screen session, tidy it up with:

> Show available sessions:

> The screen commands I use a lot:

> The screen commands I use not so much: