1 Introduction to UNIX (i.e. Unix for Windows Users)

- 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
  - someone who is lost, confused, and looking for spiritual enlightenment
    - mashall has a great religion for you to join
    - we sin so jesus didn’t die in vain
    - ask around, see who has what experience
- 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
  - knuth, knuth, wilkes, muddcs, arc, cortana
  - knuth is the one you care about
  - main server everyone will be working on - give specs.
- 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 knuth
  → teletypes - basically a typewriter plugged into a computer
    · Ken Thomson implemented the first UNIX environment
    · had 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 kmuth
  → clean, simple interface

→ Chalkboard exercise
  → unix is also multi-layered in design, meaning that functionality separated into levels going from the hardware to the user, stuff is also abstracted aways at each layer
  → user | (shell, other programs) | OS(kernel) | 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
→ it doesn’t matter what you run, what we are going to teach you here will apply across most unixes though not all

• 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 ”design”
  → 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
    · output of one could be used as the input to another
      → text streams ⇒ universal input
  · Programs are viewed as simple tools that combined will do complex tasks
    · each program should do one thing extremely well
  → LEGO blocks - simple tools but can create very complex systems
  → another analogy - hammer, wrench, saw
  → Avoid the ginormous programs that do everything
Windows has this problem (think of Office)

Windows programs also don’t integrate very nicely - no pipeline

want to keep everything as simple as possible

KISS - keep it simple stupid

this applies everywhere - both in OS interaction (system calls and library calls) and in program creation

separate mechanism from policy

Unix (and programs that run on Unix) provide a mechanism to do something but do not lock you down to one policy

policy - how a program should look and behave

users have to decide on policy

games and GUIs are generally a good examples - mods and skins

walk up to some one and try to scare them

principle of least surprise

applications will generally produce no output if successful

applications that fail will make it evident (sirens if necessary)

programs should do the least surprising thing

do not try to feed emacs a stray cat

there is no ”one true way” - unix is not perfect!

unix has things wrong with it, and things that unless you understand will cause you a great deal of pain

The Unix Hater’s Handbook - written to expose the flaws in unix

from a stanford site: http://pangea.stanford.edu/computerinfo/unix/overview/advantages.html

"The traditional command line shell interface is user hostile

designed for the programmer, not the casual user.

Commands often have cryptic names and give very little response to tell the user what they are doing. Much use of special keyboard characters - little typos have unexpected results.

To use Unix well, you need to understand some of the main design features. Its power comes from knowing how to make commands and programs interact with each other, not just from treating each as a fixed black box.

Richness of utilities (over 400 standard ones) often overwhelms novices. Documentation is short on examples and tutorials to help you figure out how to use the many tools provided to accomplish various kinds of tasks.”

"Unix *is* user-friendly. It’s just picky about its friends.”

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 privacy -> data

protect work -> programs

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
  ➔ what this is will cover later
• 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? is there a better way to organize responsibility?
  • "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 knuth you have about 120 Mb for all your files
    ➔ normal users have to obey the security policies of the system, superusers can ignore most of them
    ➔ on your own system can have as much as you want ;-) or if you’re CS staff
    ➔ 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
  ➔ run ps -aux | grep daemon
  ➔ 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
  ➔ are the unix solution to separating responsibility among users
    ➔ 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 knuth 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
    ➔ can have groups ftp, www, etc.
  ➔ exists to allow easier management of a system by administrative users

5 Logging In
• Loggin In
  ➔ two parts: username and password
  ➔ username can be anything, though some are reserved for special purposes (root, bin, etc.)
on knuth it will be the first character of your first name, followed by your last name (hopefully); on knuth the username limit is 16 characters

so on knuth I am: mkegel; Marshall is mpierce

- password should be strong
  - what makes a strong password?
  - min 8 characters in length (turing only pays attention to the first 8 characters of a password)
  - special characters (Numbers and @ ! # $ % ^ & . , ( ), even spaces)
  - uppercase characters
  - should not look like a dictionary word
  - pass-phrases are good things
  - my took CS60# could become -> 1 t00k CS 6o# (thats a one with two zeros in took and a lower case O)

Demonstrate logging in to knuth

notice that no password characters show up when typing

6 Introducing the shell

- log into knuth with the default knuth setup
  - what you see when starting
    - the default knuth setup, easiest and most familiar setup
  - components of the prompt: username, time, command number, and directory
  - change the prompt to something much simpler, like %

- 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

- change prompt to something simple - probably use a special account on farmboy

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 knuth, 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 (knuth, farmboy)
  ➔ /Users/mkegel (shadow)
  · directories are denoted with /, windows uses ¥
  · there are other directories that we will get to later, but on knuth 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
  ➔ cmd [switches] [arg1] ... [argn]
  ➔ 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 ”-?” 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!
  ➔ demo the tar stuff from the command line
  ➔ 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
  ➔ tar -xvf [tar file]
  ➔ ”-xvf” could have been typed ”-x -v -f” but easier to write the first way
  ➔ on old archaeic systems this may not actually be the case but on new systems it most often is
• 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 ”ab” is not ”a b”
  ➔ 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, you'll be able to find out more stuff about them after the end of this lecture

these are also the commands that you’ll 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 ‘source’ file to ‘target’ file/directory
- cp <source> <target> :: moves the ‘source’ file to the ‘target’ 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 page” for various system commands

- 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 x held together
    - are what might be called shortcut keys under windows
    - one other notation is caret keys and; “h 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 doesn’t 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
  - wouldn’t it be nice if say you have some long directory name, ./thisdirectorynameis-toolong/ but there was some way that you didn’t have to type it out every time, hmmm...
  - most shells (and any that you would normally use) provide what’s called tab-completion, allowing you to complete partially written statements into their full form
    - demo this feature
    - so given partial data the shell will determine what sequences of characters will appropriately complete what you’ve written on the cmd line
    - this is accomplished by hitting the tab key
    - hit tab key to show different completions
    - 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
demonstrate on the long directory name
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 en-
tire command history, useful for when you
have to type the same commands again and
again - like
make program
the down arrow with of course cycle forward
in your history

command line editing
• the left and right arrow keys move the cur-
sor 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):
  • C-? will delete back
  • C-d will delete forward
  • C-u will delete an entire line
  • C-k will delete all of the line forward
  from the cursor
  • C-l clears the terminal
  • C-y will paste the characters you have
deleted
  • C-a will move the cursor to the begin-
ning of the line
  • C-e will move to the end of a line
  • M-b moves back a word
  • M-f forward a word
  • C-_ will undo the last thing typed
  • C-r will do a history search
  • there are many more…and most impor-
tantly you can create your own!

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?
  • # $ * ? [ ] ( ) = |
  • ; < > ‘ $ " ’ \n• what do you need special characters for?
• for example the semi-colon ‘:’ separates
commands on the cmd line
  • cmd1; cmd2; cmd3;...
• but what if you want to pass a special char-
acter to a program?

Escape From "The Shell"
• what is you want to pass these characters
into a program?
  • have to escape them: \?, where ? is the
special character you want to use
  • demo 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 ’ (double or single quotes)
  • demo echo ";"
• 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
  • ’xxx’ - everything get disabled
  • "xxx" - everything but $ ’ and \
  • 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
  • echo Who \\n  stole \\n  the cookie, \\n  from the \\n  cookie jar \?


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
    - websites (particularly forums), books, documentation, and people
    - some good websites to check out are: linuxdoc.org, freebsd.org
    - some good books are: UNIX Power Tools, FreeBSD Handbook...
    - people: Marshall, Mark, Mac, and many others!!
  - Computer Science Department Staff and Consultants
    - they have people hired to help you out!! You should use them as a resource!
    - Staff take care of the various boxen the department runs
      - give a listing of the boxes, their purpose, and their addresses
        - knuth.cs.hmc.edu – shell server
      - find new mail server
        - muddcs.hmc.edu – web
      - Consultants sit in the terminal room and answer Unix and CS related questions
        - give a list of who are the consultants, emails, and schedule
    - The CS department website
    - QREF’s - documentation written specifically for YOU
    - is a great resource for general information, system policy, etc.
      - give the URL for the QREF’s
        - http://www.cs.hmc.edu/qref/
      - the professors will also help you out as much as they can
        - some good mailing addresses for questions
          - linux-l@cs.hmc.edu - linux mailing list for hmc students and others
          - give out others (?), maybe forums.muddstudents.org
  - Built In Docs
    - man pages
      - purpose is to document how each of the various programs/commands on the system work, and how to get them to do what you want
      - generally short on examples, but in general well worth reading
      - can document other things
        - get in the habit of using man pages - saves time and effort!
      - what man pages are not
        - general reference, programming guides
        - what man pages are
          - very specific references on a limited set of system commands
          - essentially they document how to use the programs on the system
        - some man pages break these rules but in general they seem to be followed
          - demo man
    - Other systems
      - info
        - is mainly about emacs
        - the 'help' command for bash
    - Most programs come with more documentation than just a man page
      - check out: /usr/doc and /usr/share/doc
      - slackware uses /usr/doc and is really good

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 plus different sections at the same time
allows for a better organization of documentation

some switches
- man -k perl
- 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
- man -a sync (demonstrate this)

how to read man pages - the parts of a man page
- at the very top is the is the name and section number; for man it is man(1)
- NAME - just a quick one line description
- SYNOPSIS - all the arguments that the program accepts
  - how to tell if something is optional or required
  - optional things are surrounding in brackets '[xxx]'
  - required things are not
  - show example - man requires at least one argument
- man man
- DESCRIPTION - a paragraph "summary" of the program
- OPTIONS - what each of the switches to the program do
- ENVIRONMENT - what environment variables the program expects

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 knuth man is setup to use less, 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
- exercise - now for a man page hunt

9 Making programs work together on the cmd line

Working Together
- ask question before showing slide
- 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
    ➔ general goes to the terminal you ran the program from
  • stderr - like stdout but for error output
  ➔ need to show distinction between stderr and stdout
  ➔ 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 (we’ll 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?

  ➔ good time to use the chalk board
  ➔ ask group for ideas
  ➔ we know that two programs can share the same file - can have the same file open two different times
  ➔ 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
  ➔ cmd1 | cmd2
  ➔ the ’ | ’ character is known as the pipe and is located on the key, so Shft- is |
  ➔ example of this in action - first run who and tell what wc does, then run this
  ➔ who | wc -l
  ➔ (gives the number of terminals open on the system)
  ➔ get a longer piping example
  • programs used in this way are called ‘filters’
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
demo piping on terminal

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
	cmd1 < file1
	You use the < to have file be stdin for the program
	redirect stdout to file
	cmd2 > file2
	So you use > to redirect the output
Some notes on redirecting output
the file cannot already exist (if you have NOCLOBBER set)
won’t execute if it does
since this is set by default on knuth you need to be aware of it
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
	cmd4 >> file

demo piping on terminal

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
	cmd1 |& cmd2
	cmd >& file
spacing matters so pay attention to it
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
	cmd5 >>& file
to redirect just stderr to a file
	cmd 2> file
redirecting just stderr along a pipe is a bit more difficult
demo this concept

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:
	2>&1
	1>/dev/null
	1>file is same as >file
	2>&1 1>&2 (this won’t actually swap stderr and stdout)
	3>&2 2>&1 1>&3 (this will swap the two)
demo this concept

not every program takes things from stdin or sends output to stdout
for example can’t do
emacs < myfile
emacs < myfile

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
  - 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

  ➔ kill -HUP ‘cat /var/run/sshd.pid’
  ➔ this will restart your ssh server, since ‘cat..’ returns the PID of sshd
  - another example

  ➔ emacs ‘grep -l error *.c’
  ➔ 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

  ➔ do most of this on the board

  ➔ some of the commands we’ve been running haven’t actually been "programs" in the sense normally described

  ➔ example: cd is a builtin, its a part of the shell and not a stand alone program like say emacs is

  ➔ 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

  ➔ develop on the board a better layered model

  ➔ user | shell - applications - libraries | kernel - modules | hardware - remote systems

  ➔ kernel - a mediator between the hardware (and remote systems) 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

  ➔ libraries contain useful code that you’d rather not reimplement

  ➔ 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?

  ➔ the instruction could be machine code

  ➔ could be byte code like in java or compiled python
or could be an interpreted scripting language

what is process?

a program in action

an abstraction provided by the kernel

every process thinks it's the only one running - runs in a self-contained environment with the kernel being responsible for enforcing this abstraction

how is a program different from a process?

how many copies of each do you normally have?

1 for executable (one file containing the binary code)

can have different versions but those don’t count

n for processes

why would you have more than one process of a single program?

need them to be doing different things - two xterm windows for example

• The Kernel?

how do the processes get run?

• the job of the kernel is to manage the many processes on the machine and run them in some sane order so that the system is both responsive and productive

• the kernel also provides an interface to all the hardware on the system

• this interface is uniform across architectures and hardware

• so you always call read() though the disk might actually be just a network share somewhere in asia

does the kernel just run processes? basically, and do what processes need done

what might be the other jobs of the kernel?

manage devices

reading/writing files

handling system calls

enforcing system policies - security for example

allows for multiple users on a single system

abstracts a common interface for different hardware - creates an abstraction

• kernel does more than this

• take Operating Systems if you want to write a kernel

• take CS105 to learn more about how Computer Systems work

as users we care about 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

• can run multiple processes at the same time

• no fear of them interacting

• each process runs in its own little environment isolated from others

• unless you WANT them to interact! very key!

• unix is a multiuser environment

• can handle multiple users using the system at the same time

• so if you are going to run multiple programs from the shell(s) it would be helpful if you knew how to manage them

• with GUIs most of these things are not really an issue, but it's still very useful to have some idea in case you mess things up

• can open as many xterms as you want, but at some level you need to know this...

• from a single shell session can run about as many programs as you would ever want to - constrained by memory and processing speed

• normally though you are only ever running one program at once in a shell - this is referred to as the foreground process

• 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?
  ➔ because you won’t always have a GUI, may end up working at just a terminal

- 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
    ➔ mv ..
    ➔ ls ..
    ➔ rm ..
  · sometimes you want to run other programs
    ➔ say stop reading a man page to copy a file, or read email
      ➔ could open another terminal - do you really want to do this for every program? what if you only have a limited number of terminals?
      · or could ”suspend” the process in which case it stops executing
      · or put it in the ”background”, and continue to let it run
    · background process don’t recieve 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”
  ➔ do a demo of suspending a process
  ➔ demo reading ”man page”, suspending, and using pine
  · will want to demonstrate that pine can not be suspended by default

- Background Processes
  · when we type
    ➔ cmd
      · this program is the one we’ll be running in the foreground
      · to run a program in the background we type
        ➔ cmd &
          · 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
        ➔ emacs &
        ➔ do some thing on the cmd line
        ➔ fg
          · this is one of the most useful things to know, so remember it!
        ➔ why this is useful - give this example scenario
          ➔ are logged in remotely with no X session
          ➔ have an emacs window open and change a bit of code
          ➔ too slow to exit emacs and restart
          ➔ so press C-z to suspend
          ➔ then can rebuild from the cmd line
          ➔ then can switch back to emacs to continue working
really only need to remember C-z to suspend and fg to bring the process back
draw on chalk board what is happening when you suspend or background a process
shell -> process -> shell
shell -> (splits) -> shell (-> process)
and more

• Mid-level management
  • use the jobs command to list all the jobs that you have running

  jobs
  [1] + Suspended man page
  [2] Suspended man perl
  [3] - Suspended man printf
  the number identifies each job
  we use the fg command to switch back to which ever job we want using the number
  by default fg will bring back the job with the plus sign
  this is the current job
  the minus sign was the job that used to be the current job
  the rest of the output of jobs is obvious
  Stopped means it doesn’t receive any cpu time
  running means just that
  example to go back to man perl we would type
  fg %2
  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 %n, you have &?xxxx where xxxx is the first part of the name of the running process
  obviously you need enough of the name to disambiguate the process
  so returning to emacs could be as simple as typing
  fg %em
  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
  bg %n

continues job n in the background
also cover %+ and %-
what happens if we background emacs?
what happens if we suspend emacs and let it continue in the background with bg?
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
  • init -> getty -> your shell (zsh) -> your programs
  • init -> sshd -> your shell (zsh) -> your programs
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
  • ps -aux
• can see what everyone is running with this
  ➔ point out which column is the process id
    ➔ pids are not always 5 digits
    ➔ init for example will always have its
      PID equal to 1
    ➔ on OS X pids are generally 3 or 4 digits
    ➔ linux tends to be 5 digits
    ➔ will most always between 2 and 32000
  ➔ 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
    ➔ this would be a good time for CS60 students to pay attention
  ➔ don’t want it running forever, so want to
    kill that process
  ➔ we use the kill cmd to do this
  ➔ kill [PID | %n]
  ➔ two ways to kill
    ➔ (1) if the process is running in the back-
      ground, can use background number
    ➔ (2) can use the process id (PID)
  ➔ also demo
  ➔ killall <proc name>
  ➔ we can use ps to find the process id
  ➔ demonstrate using ps to find the id and kill
    the process
  ➔ 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
  ➔ demonstrate top - point out the PID column
  ➔ can use this to see if any of your processes
    are out of control
  ➔ should be able to point out several aspects of
    top
    ➔ memory usage, cpu time, state and pri-
      ority
  ➔ top is made more useful by seeing only the
    processes that you own
  ➔ first start top
  ➔ top
  ➔ then press the ‘u’ key; then type your user-
    name, or another username if you want to
    see their processes
  ➔ then enter
  ➔ demo this feature also
  ➔ to get help on the other commands top of-
    fers use the ‘h’ 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
  ➔ kill 1
  ➔ 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
  ➔ init is the exception
  ➔ yes it is - init (or launchd on Tiger) is given
    PID of 1
  ➔ so typing kill 1 while root would kill you
    entire system - basically all processes stop
    and the system has to be rebooted
  ➔ this is a bad thing and should be avoided

• Process priority
  ➔ what is process priority?
    ➔ priority defines how much time the partic-
      ular 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’t actually change the pri-
      ority, 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
  ➔ nice cmd
if you are going to be running a long and fairly intensive job on knuth please use, and be, nice!
nothing worse than locking the system up for fifty people while you try to compute prime numbers ;-)
> show how mprime has been nice on odin

to use nice run it like such
> nice [-n NUM] cmd args... NUM=0 to 19 (lowest) (root for 0 to -20 (highest))

find a demo for nice
check out the man page for more info
does nice have an effect?
nice will have an effect if the process is *cpu bound* meaning that it requires the processor more than anything, doesn’t really need any input
on *IO bound* processes not so much since they wait for input anyway, and while waiting the system will do other things, won’t have much effect
now know all about managing processes and how the system handles the

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
> locate <filename>
> 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
> echo "text string"
> or
> echo text string and more text
> 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 ”cons”
> usage
> cat <file1> <file2>
> 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
> clear

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 > 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
less <file>
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
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
head
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
head [-n XX] <file>
less <file>
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
tail [-n XX | -f | +NUM] <file>
"-n" : gives last XX lines of file
"-f" : keeps file open
"+NUM" : shows all but the first NUM lines of a file
now on to some other commands
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’s until later
so to search through a single file
grep [switches] "string" file1 ... fileN
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
find ./ | grep pattern
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
find ./ -name glob -print
the "-name" switch will search for the glob, and "-print" will print it to standard out (note that glob != regex)

find can of course do much more interesting things

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

```
find ./ -name ".foo" -perm +444
-exec chmod -R a+r {} \;
```

these are other commands which are useful, but not exactly necessary to know

```
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
```

```
which cmd
```

```
du
```

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

```
-du file1 file2 file3 ...
```
to get the output to be a bit more useful, will want the "-h" switch

demonstrate this

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

show the output of and explain

made nicer by the "-h" option

show this output

at the end of this would be a good time to poll to figure out what other programs people would like to know about;

questions like "what is the program that does ....?" sorting, calculator, etc.

15 Files, programs and file, and the filesystem

what is a file? some review from before

a file is a stream of bytes; a file isn’t always necessarily on disk

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 primary group as the group the file belongs to, and all the permissions on

• 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

• File 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. hasn’t 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
  ➔ have an exercise to figure out what the permissions of several files mean
  ➔ so when you see rw-rw-rw what does this mean? etc.

• 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
  ➔ 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
  ➔ 666: everyone read, write
  ➔ 777: do anything to file
  ➔ 022: user can’t do anything, group w, other w
  ➔ 744: user do everything, group read, other read
  ➔ as an exercise write out the octal permissions in the rwxrwxrwx form

• Changing Ownership
  ➔ do an ls -l on some files
  ➔ demonstrate the two following programs
    • chgrp - group
    • chown - owner

• Changing permissions
  • chmod - will also change ALL aspects of a file if wanted
  ➔ demonstrate some standard examples - make file world readable
    • switches u, g, o and the += syntax, with perms r,w,x
  ➔ also cover -R for recursive
    ➔ since may want to change all files in a directory

• 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)
  ➔ list how permissions affect a directory file
    • 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
  ➔ do an example of a hidden file .hidden or “.you don’t see me”
  ➔ is that all the files in the directory?
not necessarily...

ls -a

"-a" 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

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 doesn’t show all the files in a directory by default? yes, but is still historical crap

a better solution might be possible

show a gftp listing of my home on farmboy

so when you do a ls -a, do you recall seeing two files one named "." and the other ".."?

probably not but these are two special files

the file named . is this directory

do a literal translation for whatever directory in on board

so if you type

cd .

you stay in the same place

what if you see a dot in the full path name of a file?

ex. /foo/bar/../baz/box/

where is this referring?

to /foo/bar/baz/box/

so it means nothing, should just ignore it

./ is the same thing as .

the file named .. is the parent directory

you should already know that if you want to go up in the directory hierarchy that you type

therefore .. is the parent directory of the current directory

../ is the same as ..
cd ..

so can probably guess that is you see /foo/bar/../baz/box/

you know where this refers to? where does this go?

answer: /foo/baz/box/

so you can’t ignore these, but can be useful for when you need to craft a path to a file

other special files

• tilde (~) denotes your home directory

cd ~

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

how do you refer to this file?

two ways: relative and absolute path

absolute paths never change and describe in "absolute" terms where a file is in the tree

emacs /home/mkegel/school/fall05/graphics.txt

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

cmd ./one/two/config.cfg

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

→ how might you do this?

• 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 its 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
  → ln [OPTIONS] target link_name
  · the basic syntax is backwards, so to ”link” emacs to emacs-21.2 you would type
  → ln emacs-21.2 emacs
  · to create one pass the ”-s” switch to ln
  → ln -s emacs-21.2 emacs
  · this will create a soft-link
  → show output of ls -l
  · 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
      · will not match dot-files, however
    · ? - 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"

→ give some examples and ask what they would match
· a[a-zA-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 .*
→ show listing of ls .*
→ note that this also lists "." and ".."
→ so what would happen say if we did this...
→ chmod -R a+r ./
→ as root from /home?
→ much badness is what...since more than just the down directory will be changed
→ .. follows to its parent and so you’ll 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 unixes 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
→ just to 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*
→ draw a /
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  
  → cd into each directory as they are being discussed  
  → /home is where the heart (and your homework) 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  
  → man hier (on OSX, also present on FreeBSD, haven’t seen it on Linux)  
  → see also pg 822 UPT  
  → /bin/ user utilities fundamental to both single-user and multi-user environments  
  → /dev/ block and character device files fd/file descriptor files; see fd(4)  
  → /etc/ system configuration files and scripts  
    → X11/ configuration information for X  
      · NFS: exports  
      · system: group, passwd, and shadow  
      · shell: profile  
      · sudo: sudoers  
      · network and inet: hosts, hosts.allow, hosts.deny, hosts.equiv  
      · lots of other stuff  
  → /sbin/ system programs and administration utilities fundamental to both single-user and multi-user environments  
  → /tmp/ temporary files  
  → /usr/ contains the majority of user utilities and applications  
    → bin/ common utilities, programming tools, and applications  
    → games/ Unix games, also fortune  
    → include/ standard C include files  
    → lib/ archive libraries  
  → libexec/ system daemons and system utilities (executed by other programs)  
  → local/ executables, libraries, etc. not included by the basic operating system  
  → sbin/ system daemons and system utilities (executed by users)  
  → share/ architecture-independent data files  
  → share/man/ manual pages  
  → src/ storage for kernel/other source code under linux  
  → /var/ multi-purpose log, temporary, transient, and spool files  
    → log/ misc. system log files  
  → /lib/ shared libraries and kernel modules  
  → /opt - larger static packages - where you would install KDE for example  
  → /boot - kernel image and other boot files  
    → this is where the linux kernel gets put  
    → is sometimes a separate partition - Red Hat does this  
    → probably a good idea since if your main one gets fried can still boot  
  → /mnt - temporary/permanent mount point for devices  
    → devices can be mounted anywhere, so one solution, if you have lots of hard drives, optical drives, cameras, scanners, etc. is to make lots of mount points under /mnt, so /mnt/dvd/, /mnt/ipod, etc.  
    → is this a good thing?  
    → all up to your own preferences  
    → /mnt is meant for (at least in FreeBSD) a single device that won’t be mounted for any length of time  
    → would mount to a different point, say /ipod, so mounted right from root  
    → are multiple solutions, could also have a /vol (/Volumes)  

19 Some special devices  
• Special files  
  · /dev/null  
    · this is a sink for input, you can send anything here and it will just go into the void
useful for when you need a place to get rid of output, but don’t want to waste space on the disk
• if read from will give you an EOF

/dev/zero
• produces an infinite stream of zeros if read from, good for when you need to null something out
• if written to will just give a permission error (?? Is this always true ??)

/dev/random and /dev/urandom
• when read produces a random stream of bytes
• permission error when try to write to

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
    • where to find programs to run, which editor to use, etc
    • serves the same purpose as the registry (sort of) in windows

→ cover the distinction between the two
  • environment variables are inherited across all shells you may have and all programs that you run
  • shell variables are local to each instance of a shell that you may have
  • each shell gets a clean slate of shell variables when it starts
  • analogy - programming: global versus local scope
  • variable naming convention
  • CAPS are environment
  • lower-case are shell
  • please follow convention!!!
  → you generally only ever care about environment variables

→ show listings of the variables
  • how do you see what environment variables you have?
  • simple use the printenv command

→ printenv
  • what about shell variables?

→ set (lists everything: functions, shell, and env. variables)
→ pg 705 has a good listing of pre-defined environment variables

• Common Variables
  • good ones to know about - will be all caps
    • PATH - when you type in a command these are the directories that the shell will search for the cmd you have typed
    • HOME - where your home directory is located
    • HOSTNAME - name of the system you are on
    • SHELL - full path name of shell you are using
    • USER - who you are logged in as
    • UID - your user id
    • TERM - type of terminal you are using

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 knuth 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

→ give proper pronunciation lesson
→ the default on knuth is zsh, which is a hybrid of sh and csh
→ other shells include bash (probably the most widely used shell), tsh, 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
export NAME=value
can also do this
NAME=value; export NAME
you can see the output of a variable by using the echo cmd
echo $ENV
have to remember the $
demonstrate this for the students
the most common thing to do is to add paths to PATH
may have to do this for a class
may want to do this depending on which box you are logged into
from the cmd line for zsh we would type
echo $PATH (to see path)
export PATH=$PATH:/newpath
or -
export PATH=/newpath:\$PATH
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

maybe there is an easy way fix this problem?
ask crowd for suggestion
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
/.zprofile
/.zlogin
/.zshrc
and more
depending on default setup will want to show how to do this add path
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
source .file
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’re 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
setopt nobeep
we would put this line in our zshrc; by default it should already be there
there are lots of options so check them out!
may want to read through the default zshrc to see what’s set

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 user@knuth.cs.hmc.edu"
  – so to create an alias we need a name: how about sshtoknuth
  – from the cmd line would type: (spacing is important)
    alias sshtoknuth='ssh user@knuth.cs.hmc.edu'
can also use " when creating alias’ (good if you want to reference env. variables)
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
alias name='cmd arg1 arg2'
but what would be a good name? something better than sshtoknuth
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: knuth
says what you want, is simple and short
what would have happened though if we’d 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 it will be replaced
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 knuth is not /usr/local/bin/ls
→ instead you get "/usr/local/bin/ls -F"
→ do an example of what each outputs
→ discuss why this might be a bad thing, tell about -G on GNU ls (hide groups)
→ should watch which options you pass by default
→ so how did I know that all of you had this?
→ demonstrate the defaults alias’ that people have in zshrc
→ 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
→ alias knuth=’ssh mkegel@knuth.cs.hmc.edu’
→ 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
→ show in zshrc
→ written out in octal permissions (remember those?)
→ write out the permissions of the umask
→ ask what the permissions of a newly created file would look like
→ ask if the results are good or not
→ what might be a good umask; what would be a bad one?

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

• Useful Applications
→ these are almost all graphical tools and are essentially replacements for programs that students would be using on either their Mac or Win box
  · web browser - firefox and mozilla, konqueror
  · text editing - jedit, emacs, vi
  · word processing - abiword, open office
  · document creation - latex
  · spreadsheet - gnnumeric, open office
  · music - xmms, amarok (my favorite)
  · movies - totem, xine, mplayer, vlc
  · graphical terminals - eterm, aterm, xterm, rxvt, gnome-terminal, konsole
  · mail - thunderbird, evolution, pine/mutt through xterm
  · pdf viewers - xpdf, kpdf, gpdf
  · ftp - ncftp or lftp (cmd line), gftp
  · cd burning - k3b
  · aim - gain, kopete
  · bit torrent - azureus
  · file manager - konqueror, nautilus
  · spell checkers - ispell and aspell
  · Windows like desktop - GNOME, KDE
  · window managers - fluxbox, xfe, window maker, enlightenment

26 su and sudo
→ up to this point have been fairly general, information applies to almost all users of a *nix system
→ 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 knuth
• sudo and su
→ discussed before how use
  · su allows you to assume the role of the superuser (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
   → demonstrate adding
   → mkegel ALL=(ALL) ALL
   → %wheel ALL=(ALL) ALL
   → %special ALL=(root) NOPASSWD:/usr/bin/myprog
   → joe user local_host=/sbin/prog
     · the basic format is:
       USER HOST=[(RUN_AS)] CMD
→ 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 knuth
• 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
→ draw out a diagram of how multiple drives/cdroms/devices can be accessed on a filesystem tree
→ 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 knuth
your home directories are mounted locally on knuth when you log in, but where they were before that doesn’t really matter

obviously knuth 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 knuth 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

  in the 2.4 kernel every device that you could possibly have attached to the system was enumerated and given a device node in /dev
  in 2.6 the kernel is a little bit smarter about devices now and can handle them with some intelligence
  for you this won’t matter too much unless you’re trying to hook in some exotic device
  or if you have sound/video/usb problems (or want to use these devices) then you’ll get to know /dev pretty well
  generally a device will be owned by root and in one of several groups: disk, video, audio, root, etc.

cr hard drives are generally given the designation hdXN, where X is [a-f] and is the drive and N is [0-9]

  → hdal would be the first partition on the master hard drive
  → hdal2 would be the second partition
  → hdb would be the second hard drive
  → drives have up to 4 primary partitions, and can have more logical partitions
  → generally if you just say hda, or hdb you mean the entire device and not any one partition

  ∙ 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)!
  → mount /dev/device /mnt/mntpoint
  → to specify the file system use the -t switch like such
  → mount -t vfat /dev/hda4 /mnt/windows
  → to un-mount the device use the umount command like such
  → umount /dev/windows
  → 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 it's 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

df

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

du <filename>

useful alias

alias sizeof='du -sh'

/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)

show my /etc/fstab - point out what each column is

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

mount /mnt/windows

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

crontab mysched

also, edit the crontab with crontab -e

mysched will then contain single line entries for each task that you want cron to do

each line is formatted like this

mins hrs day-of-month month weekday [username] cmd

too fubar this, will make you un-happy

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)

show a random cron example
→ something scary for Friday the 13th, and something cheerful for Christmas, and school starting every Monday

→ 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

→ wine WinApp.exe

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

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

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

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

→ hand out reference cards

→ demonstrate some nifty tricks (M-q for one)

→ how to exit and save

→ F10 is menu in emacs

→ vi is a modal editor - some people ”just get this” way of editing

→ 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

→ 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:

→ diveintopython.org and python.org

→ perl.org and perl.com
cannot stress how useful knowing a language like one of these is

learn it!

logging in through ssh

cannot stress how useful knowing a language like one of these is

learn it!

logging in through ssh

how to X forward with ssh (-X | -Y)

how to use ftp/sftp and scp

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

3 slides for this topic

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

grep ‘foo’ file1

returns every line that contains the string ‘foo’

grep ‘^foo’ file1

returns every line that starts with foo

grep ‘foo$’ file1

returns every line that ends with foo

grep ‘^foo$’ file1

only lines that are exactly ‘foo’

grep ‘fo*’

would match f, fo, fooooooooooooooooo, etc

grep ‘fo?’

would match f, fo

grep ‘f[oe]*’ file1

would match f, fo, fe, fee, etc, but not fie

grep ‘^f[oe]{1,2}$’ file1

would match fo, fe, feo, fee, foo, but not f or fo00

grep ‘f[^aeiou]*’

would match f followed by 0 or more non-vowels

grep ‘(fa|e)o*’

would match fa, faoooo, e, eoooo, but not fae

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(lete). 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.

$ 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.

`s/tree/cat/` replaces the first instance of 'tree' with 'cat'.

`s/tree/cat/g` replaces every instance of tree with cat (g for global).

`s/tree/(k)/g` replaces every instance of tree with (tree). & references the entire match.

`s/tr\(ee\)/\1/` replaces 'tree' with 'ee'.

`s/\(tr\)\(ee\)/\2\1/` replaces 'tree' with 'eetr'.

`s/\(tr\)\(ee\)/\2\2\1/` replaces 'tree' with 'eeeeeetr'.

`s/<\(\[^>\]\)\(.*\)<\(/\1>/\[\1\]\2\\[\/\1\]/` replaces '<b>foo</b>' with '[b]foo[/b]'.
→ "a a send a literal "a
→ "a ? (shift /) show help
→ "a " (shift ') show current shells in a list