The shell

The shell: Why we (I?) teach it

Many of the students in my classes know the computer as a Desktop and a handful of applications (a browser, Word, Excel, maybe R); to introduce the shell means having **a discussion about the structure of the computer**, about operating systems, about filesystems, about history

The shell offers **programmatic access to a computer's constituent parts,** allow students to "do" data analysis on directories, on processes, on their network (and I realize that the "data analysis" metaphor is getting stretched a bit thin)

Given that there are so many flavors of shell, it is also the first time we can talk about **choosing tools** (that the choice is theirs to make!), about evaluating which shell is best for them, and about the shifting terrain of software development (maybe a point best left for the graduate students)

The shell: Why we teach it

As a practical matter, shell tools are an indispensable part of my own practice, for data “cleaning,” for preliminary data processing, for exploratory analysis; by design, they let you deal with data on a scale that can be difficult from within R

The shell also becomes important as they start to make use of shared course resources (data, software, hardware)

The shell: How I teach it

First, **all the students in the class need access to a shell**; as we mentioned yesterday, this means having to download something like Cygwin for students with Windows machines (next page)

In my case, lectures can take place in a lab with uniform hardware and software (iMacs); when I'm not in the lab, easily 50% of the students bring their laptops to class and (after preparing them in discussion or lab session), we can have a **"find a buddy" interactive session**

The tools are taught **in the context of some data task** (although by this point in our workshop the first couple of data sets that I use have been discussed numerous times and the sense of "discovery" is gone)


Cygwin Information and Installation
http://www.cygwin.com/
Google
Bank Deals ...s and Deals
Google Maps
Wikipedia
News (378)
GNU + Cygnus + Windows = cygwin™
Cygwin Home
Cygwin/X Home
Red Hat Cygwin Product
Community
Reporting Problems
Mailing Lists
Newsgroups
Gold Stars
Mirror Sites
Donations
Documentation
FAQ
User's Guide
API Reference
Acronyms
Contributing
Snapshots
Source in CVS
Cygwin Packages
Software
Setup Package Search
Related Sites
Licensing Terms
sourceware.org
What Is Cygwin?
Cygwin is a Linux-like environment for Windows. It consists of two parts:
• A DLL (cygwin1.dll) which acts as a Linux API emulation layer providing substantial Linux API functionality.
• A collection of tools which provide Linux look and feel.
The Cygwin DLL currently works with all recent, commercially released x86 32 bit and 64 bit versions of Windows, with the exception of Windows CE.
Note that the official support for Windows 95, Windows 98, and Windows Me will be discontinued with the next major version (1.7.0) of Cygwin.
What Isn't Cygwin?
• Cygwin is not a way to run native linux apps on Windows. You have to rebuild your application *from source* if you want it to run on Windows.
• Cygwin is not a way to magically make native Windows apps aware of UNIX ® functionality, like signals, ptys, etc. Again, you need to build your apps *from source* if you want to take advantage of Cygwin functionality.
Help, contact, web page, other info...
Install Cygwin now
PayPal DONATE
Install or update now! (using setup.exe)
or
get help on using setup.exe.
or
find where a package or file lives in the Cygwin release.
Latest Cygwin DLL release version is 1.5.25-14

The shell: What I teach

The next few slides are samples of what I teach; I hadn't intended to teach the material here, but instead review the kinds of things I talk about

In a couple places, we'll have an opportunity to look at some data and i'll point you to that; we'll slow down for those less standard parts of what I cover

They are a work in progress and each year I encounter things that would make them smoother...

Oh and this GAP icon will indicate where my lecture notes have been diced up to focus on just the shell...

For the rest of the session

We will look at the Unix operating system, the philosophy underlying its design and some basic tools

We will use as our case study the last week of traffic across the department’s website `www.stat.ucla.edu`

You will have a chance to kick the tires on these tools and address some simple web site usage statistics

Operating systems

Most devices that contain a computer of some kind will have an OS; they tend to emerge when the appliance will have to deal with new applications, complex user-input and possibly changing requirements on its function

Your Tivo, Treo and (soon) Peugeot will all have operating systems

Operating systems

An operating system is a piece of software (code) that organizes and controls hardware and other software so your computer behaves in a flexible but predictable way

For home computers, Windows, MacOS and Linux are among the most commonly used operating systems

Some history

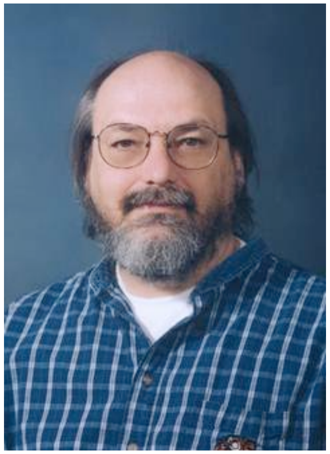
In 1964, Bell Labs partnered with MIT and GE to create Multics (for Multiplexed Information and Computing Service)

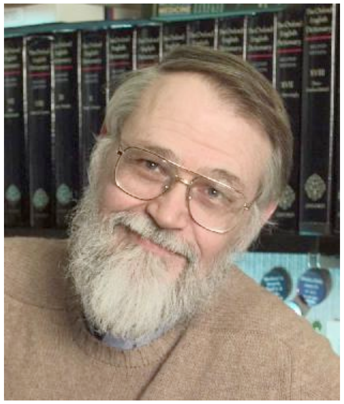
> "Such systems must run continuously and reliably 7 days a week, 24 hours a day in a way similar to telephone or power systems, and must be capable of meeting wide service demands: from multiple man-machine interaction to the sequential processing of absentee-user jobs; from the use of the system with dedicated languages and subsystems to the programming of the system itself"

Some history

Bell Labs pulled out of the Multics project in 1969, a group of researchers at Bell Labs started work on Unics (Uniplexed information and computing system) because initially it could only support one user; as the system matured, it was renamed Unix, which isn't an acronym for anything

Richie simply says that Unix is a “somewhat treacherous pun on Multics”





The Unix filesystem

In Multics, we find the first notion of a hierarchical *file system*; files were arranged in a tree structure allowing users to have control of their own areas

Unix began (more or less) as a file system and then an interactive *shell* emerged to let you examine its contents and perform basic operations

The kernel and the shell

The Unix *kernel* is the part of the operating system that carries out basic functions like accessing files, handling communication, and other functions will discuss shortly

The Unix *shell* is a user interface to the kernel (keep in mind that Unix was designed by computer scientists for computer scientists and the interface is not optimized for novices)

Unix shells

A shell is a type of program called an interpreter; think of it as a text-based interface to the kernel

It operates in a simple loop: It accepts a command, interprets it, executes the command and waits for another

The shell displays a prompt to tell you that it is ready to accept a command

* A boring slide, but full of potential!

Unix shells

The shell is itself a program that the Unix operating system runs for you (a program is referred to as a *process* when it is running)

The kernel manages many processes at once, many of which are the result of user commands (others provide services that keep the computer running)

Some commands are built into the shell, others have been added by users

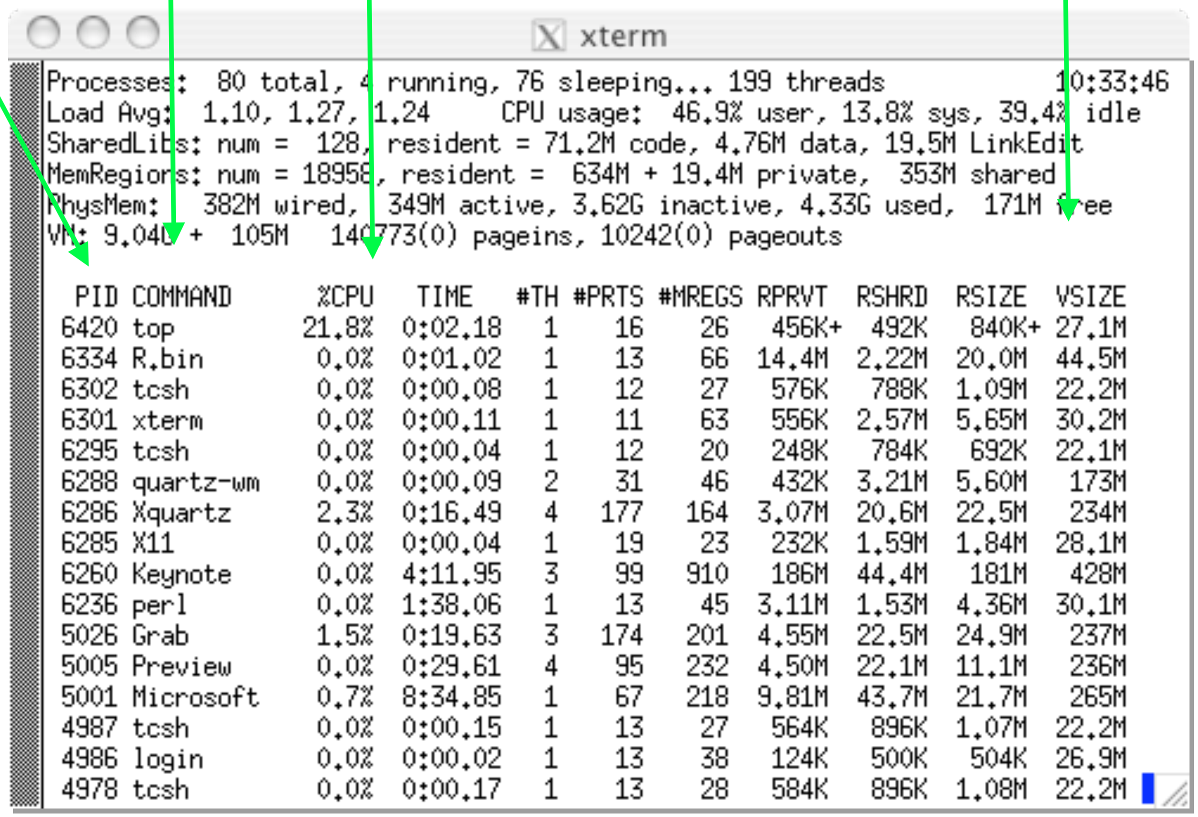
Either way, the shell waits until the command is executed

Name of the command

Process ID

How hard the computer is thinking about it

How much memory is being used



```
Processes:  80 total, 4 running, 76 sleeping... 199 threads            10:33:46
Load Avg:  1.10, 1.27, 1.24     CPU usage:  46.9% user, 13.8% sys, 39.4% idle
SharedLibs: num =  128, resident = 71.2M code, 4.76M data, 19.5M LinkEdit
MemRegions: num = 18958, resident =  634M + 19.4M private,  353M shared
PhysMem:  382M wired,  349M active, 3.62G inactive, 4.33G used,  171M free
VM: 9.04G +  105M   140773(0) pageins, 10242(0) pageouts

  PID COMMAND      %CPU   TIME    #TH #PRTS #MREGS RPRVT  RSHRD  RSIZE  VSIZE
 6420 top         21.8%  0:02.18   1    16     26  456K+  492K   840K+ 27.1M
 6334 R.bin        0.0%  0:01.02   1    13     66  14.4M  2.22M  20.0M  44.5M
 6302 tcsh         0.0%  0:00.08   1    12     27   576K   788K  1.09M  22.2M
 6301 xterm        0.0%  0:00.11   1    11     63   556K  2.57M  5.65M  30.2M
 6295 tcsh         0.0%  0:00.04   1    12     20   248K   784K   692K  22.1M
 6288 quartz-wm    0.0%  0:00.09   2    31     46   432K  3.21M  5.60M   173M
 6286 Xquartz      2.3%  0:16.49   4   177    164  3.07M  20.6M  22.5M   234M
 6285 X11          0.0%  0:00.04   1    19     23   232K  1.59M  1.84M  28.1M
 6260 Keynote      0.0%  4:11.95   3    99    910   186M  44.4M   181M   428M
 6236 perl         0.0%  1:38.06   1    13     45  3.11M  1.53M  4.36M  30.1M
 5026 Grab         1.5%  0:19.63   3   174    201  4.55M  22.5M  24.9M   237M
 5005 Preview      0.0%  0:29.61   4    95    232  4.50M  22.1M  11.1M   236M
 5001 Microsoft    0.7%  8:34.85   1    67    218  9.81M  43.7M  21.7M   265M
 4987 tcsh         0.0%  0:00.15   1    13     27   564K   896K  1.07M  22.2M
 4986 login        0.0%  0:00.02   1    13     38   124K   500K   504K  26.9M
 4978 tcsh         0.0%  0:00.17   1    13     28   584K   896K  1.08M  22.2M
```

The result of typing in the command `top`; a printout of all the processes running on your computer

Operating systems

Processor management

Schedules *jobs* (formally referred to as *processes*) to be executed by the computer

Memory and storage management

Allocate space required for each running process in main memory (RAM) or in some other temporary location if space is tight; and supervise the storage of data onto disk

Operating systems

Device management

A program called a *driver* translates data (files from the filesystem) into signals that devices like printers can understand; an operating system manages the communication between devices and the CPU, for example

Application interface

An API (application programming interface) let programmers use functions of the computer and the operating system without having to know *how* something is done

User interface

Finally, the operating system turns and looks at you; the UI is a program that defines how users interact with the computer -- some are graphical (Windows is a GUI) and some are text-based (your Unix shell)

Unix shell(s)

There are, in fact, many different kinds of Unix shells

The table on the right lists a few of the most popular; your default shell is `tcsh`

Bourne shell `/bin/sh`
The oldest and most standardized shell. Widely used for system startup files (scripts run during system startup). Installed in Mac OS X.

Bash (Bourne Again SHell) `/bin/bash`
Bash is an improved version of sh. Combines features from csh, sh, and ksh. Very widely used, especially on Linux systems. Installed in Mac OS X.
http://www.gnu.org/manual/bash/

C shell `/bin/csh`
Provides scripting features that have a syntax similar to that of the C programming language (originally written by Bill Joy). Installed in Mac OS X.

Korn shell `/bin/ksh`
Developed at AT&T by David Korn in the early 1980s. Ksh is widely used for programming. It is now open-source software, although you must agree to AT&T's license to install it. http://www.kornshell.com

TC Shell `/bin/tcsh`
An improved version of csh. The *t* in tcsh comes from the TENEX and TOPS-20 operating systems, which provided a command-completion feature that the creator (Ken Greer) of tcsh included in his new shell. Wilfredo Sanchez, formerly lead engineer on Mac OS X for Apple, worked on tcsh in the early 1990s at the Massachusetts Institute of Technology.

Z shell `/bin/zsh`
Created in 1990, zsh combines features from tcsh, bash, and ksh, and adds many of its own. Installed in Mac OS X.
http://zsh.sourceforge.net

The original UNIX system shell was a simple program written by Ken Thompson at Bell Laboratories, as the interface to the new UNIX operating system. It allowed the user to invoke single commands, or to connect commands together by having the output of one command pass through a special file called a pipe and become input for the next command. The Thompson shell was designed as a command interpreter, not a programming language. While one could put a sequence of commands in a file and run them, i.e., create a shell script, there was no support for traditional language facilities such as flow control, variables, and functions. When the need for some flow control surfaced, the commands /bin/if and /bin/goto were created as separate commands. The /bin/if command evaluated its first argument and, if true, executed the remainder of the line. The /bin/goto command read the script from its standard input, looked for the given label, and set the seek position at that location. When the shell returned from invoking /bin/goto, it read the next line from standard input from the location set by /bin/goto.

Unlike most earlier systems, the Thompson shell command language was a user-level program that did not have any special privileges. This meant that new shells could be created by any user, which led to a succession of improved shells. In the mid-1970s, John Mashey at Bell Laboratories extended the Thompson shell by adding commands so that it could be used as a primitive programming language. He made commands such as if and goto built-ins for improved performance, and also added shell variables.

At the same time, Steve Bourne at Bell Laboratories wrote a version of the shell which included programming language techniques. A rich set of structured flow control primitives was part of the language; the shell processed commands by building a parse tree and then evaluating the tree. Because of the rich flow control primitives, there was no need for a goto command. Bourne introduced the "here-document" whereby the contents of a file are inserted directly into the script. One of the often overlooked contributions of the Bourne shell is that it helped to eliminate the distinction between programs and shell scripts. Earlier versions of the shell read input from standard input, making it impossible to use shell scripts as part of a pipeline.

By the late 1970s, each of these shells had sizable followings within Bell Laboratories. The two shells were not compatible, leading to a division as to which should become the standard shell. Steve Bourne and John Mashey argued their respective cases at three successive UNIX user group meetings. Between meetings, each enhanced their shell to have the functionality available in the other. A committee was set up to choose a standard shell. It chose the Bourne shell as the standard.

At the time of these so-called ``shell wars'', I worked on a project at Bell Laboratories that needed a form entry system. We decided to build a form interpreter, rather than writing a separate program for each form. Instead of inventing a new script language, we built a form entry system by modifying the Bourne shell, adding built-in commands as necessary. The application was coded as shell scripts. We added a built-in to read form template description files and create shell variables, and a built-in to output shell variables through a form mask. We also added a built-in named let to do arithmetic using a small subset of the C language expression syntax. An array facility was added to handle columns of data on the screen. Shell functions were added to make it easier to write modular code, since our shell scripts tended to be larger than most shell scripts at that time. Since the Bourne shell was written in an Algol-like variant of C, we converted our version of it to a more standard K&R version of C. We removed the restriction that disallowed I/O redirection of built-in commands, and added echo, pwd, and test built-in commands for improved performance. Finally, we added a capability to run a command as a coprocess so that the command that processed the user-entered data and accessed the database could be written as a separate process.

At the same time, at the University of California at Berkeley, Bill Joy put together a new shell called the C shell. Like the Mashey shell, it was implemented as a command interpreter, not a programming language. While the C shell contained flow control constructs, shell variables, and an arithmetic facility, its primary contribution was a better command interface. It introduced the idea of a history list and an editing facility, so that users didn't have to retype commands that they had entered incorrectly.

I created the first version of ksh soon after I moved to a research position at Bell Laboratories. Starting with the form scripting language, I removed some of the form-specific code, and added useful features from the C shell such as history, aliases, and job control.

In 1982, the UNIX System V shell was converted to K&R C, echo and pwd were made built-in commands, and the ability to define and use shell functions was added. Unfortunately, the System V syntax for function definitions was different from that of ksh. In order to maintain compatibility with the System V shell and preserve backward compatibility, I modified ksh to accept either syntax.

The popular inline editing features (vi and emacs mode) of ksh were created by software developers at Bell Laboratories; the vi line editing mode by Pat Sullivan, and the emacs line editing mode by Mike Veach. Each had independently modified the Bourne shell to add these features, and both were in organizations that wanted to use ksh only if ksh had their respective inline editor. Originally the idea of adding command line editing to ksh was rejected in the hope that line editing would move into the terminal driver. However, when it became clear that this was not likely to happen soon, both line editing modes were integrated into ksh and made optional so that they could be disabled on systems that provided editing as part of the terminal interface.

As use of ksh grew, the need for more functionality became apparent. Like the original shell, ksh was first used primarily for setting up processes and handling I/O redirection. Newer uses required more string handling capabilities to reduce the number of process creations. The 1988 version of ksh, the one most widely distributed at the time this is written, extended the pattern matching capability of ksh to be comparable to that of the regular expression matching found in sed and grep.

In spite of its wide availability, ksh source is not in the public domain. This has led to the creation of bash, the ``Bourne again shell'', by the Free Software Foundation; and pdksh, a public domain version of ksh. Unfortunately, neither is compatible with ksh.

In 1992, the IEEE POSIX 1003.2 and ISO/IEC 9945-2 shell and utilities standards were ratified. These standards describe a shell language that was based on the UNIX System V shell and the 1988 version of ksh. The 1993 version of ksh is a version of ksh which is a superset of the POSIX and ISO/IEC shell standards. With few exceptions, it is backward compatible with the 1988 version of ksh.

The awk command was developed in the late 1970s by Al Aho, Brian Kernighan, and Peter Weinberger of Bell Laboratories as a report generation language. A second-generation awk developed in the early 1980s was a more general-purpose scripting language, but lacked some shell features. It became very common to combine the shell and awk to write script applications. For many applications, this had the disadvantage of being slow because of the time consumed in each invocation of awk. The perl language, developed by Larry Wall in the mid-1980s, is an attempt to combine the capabilities of the shell and awk into a single language. Because perl is freely available and performs better than combined shell and awk, perl has a large user community, primarily at universities.

From "ksh - An Extensible High Level Language" by *David G. Korn*

Why the choices?

A shell program was originally meant to take commands, interpret them and then execute some operation

Inevitably, one wants to collect a number of these operations into programs that execute compound tasks; at the same time you want to make interaction on the command line as easy as possible (a history mechanism, editing capabilities and so on)

The original Bourne shell is ideal for programming; the C-shell and its variants are good for interactive use; the Korn shell is a combination of both

Steve Bourne, creator of sh, in 2005

And while we are at it...

Unix itself comes in different flavors; the 1980s saw an incredible proliferation of Unix versions, somewhere around 100 (System V, AIX, Berkeley BSD, SunOS, Linux, ...)

Vendors provided (diverging) version of Unix, optimized for their own computer architectures and supporting different features

Despite the diversity, it was still easier to “port” applications between versions of Unix than it was between different proprietary OS

In the 90s, some consolidation took place; today Linux dominates the low-end market, while Solaris, AIX and HP-UX are leaders in the mid-to-high end

A few common commands

First, commands to explore your file system; walk through *directories* and list files

`pwd, ls, cd`

`mkdir, rmdir`

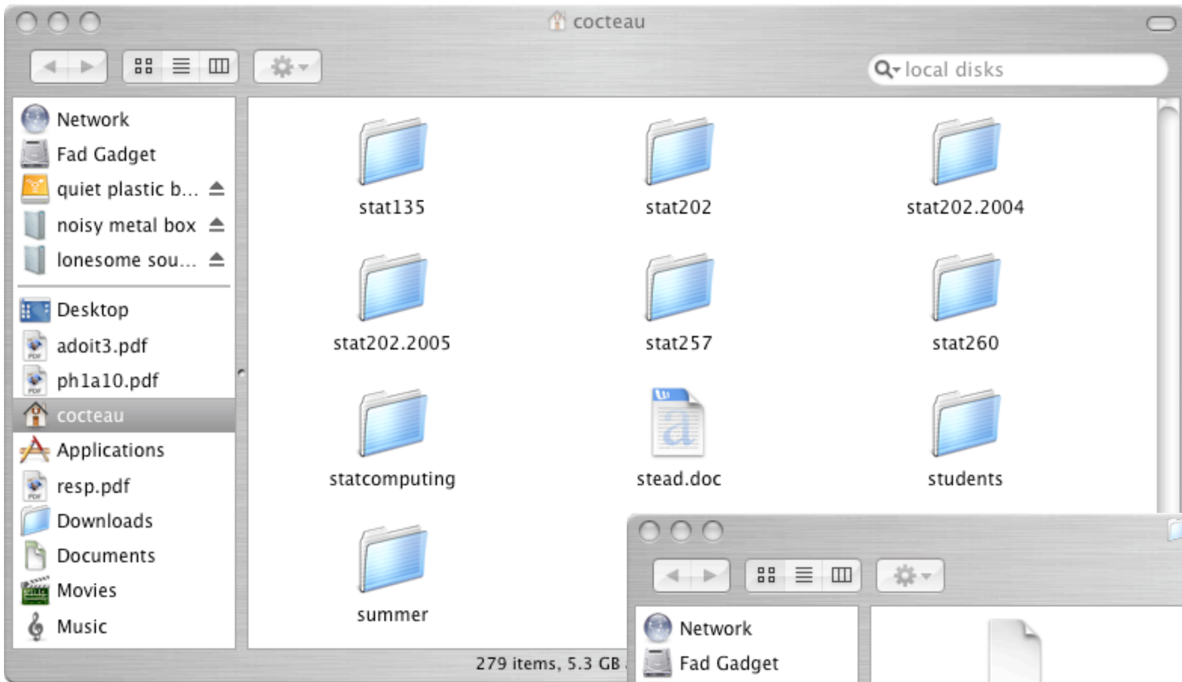
`cp, mv, rm`

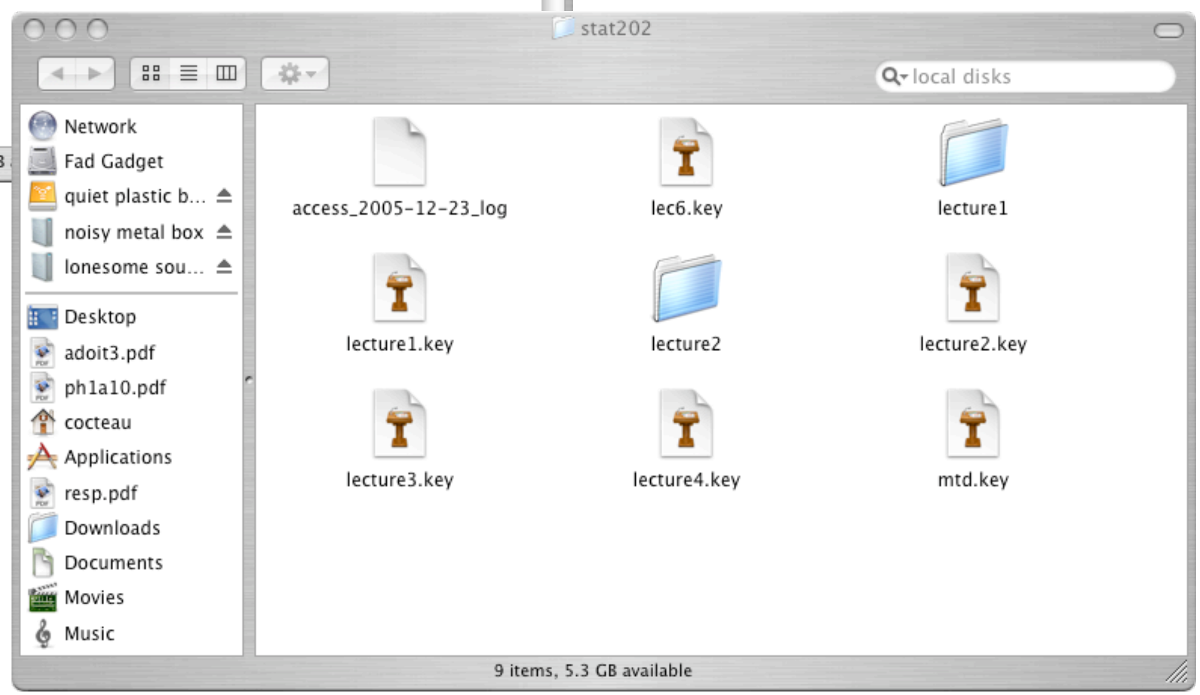
xterm

```
[fad-gadget ~] pwd
/Users/cocteau

[fad-gadget ~] cd stat202
[fad-gadget ~/stat202] pwd
/Users/cocteau/stat202

[fad-gadget ~/stat202] ls
access_2005-12-23_log   lecture1.key            lecture3.key
lec6.key                lecture2                lecture4.key
lecture1                lecture2.key            mtd.key
[fad-gadget ~/stat202]
[fad-gadget ~/stat202] ls -l
total 5197144
-rw-r--r--    1 cocteau  staff  2660935974  4 Oct 08:46 access_2005-12-23_log
drwxr-xr-x   33 cocteau  staff        1122  2 Oct 15:15 lec6.key
drwxr-xr-x   11 cocteau  staff         374  2 Oct 15:49 lecture1
drwxr-xr-x   86 cocteau  staff        2924  2 Oct 15:57 lecture1.key
drwxr-xr-x    5 cocteau  staff         170  4 Oct 10:33 lecture2
drwxr-xr-x   43 cocteau  staff        1462  4 Oct 10:43 lecture2.key
drwxr-xr-x   35 cocteau  staff        1190  4 Oct 08:44 lecture3.key
drwxr-xr-x   30 cocteau  staff        1020  4 Oct 10:10 lecture4.key
drwxr-xr-x  152 cocteau  staff        5168  2 Oct 15:08 mtd.key
[fad-gadget ~/stat202]
```





Another view of the filesystem; here, your Mac will display directories as folders and you navigate by clicking rather than typing commands

An example

Shorthand for your present working directory (where you're at)

Shorthand for the directory one level above

```
[fad-gadget ~/stat202] ls -l

total 5197144
drwxr-xr-x   11 cocteau  staff         374  4 Oct 10:10 .
drwx------  315 cocteau  staff       10710  4 Oct 08:59 ..
-rw-r--r--    1 cocteau  staff  2660935974  4 Oct 08:46 access__2005...
drwxr-xr-x   33 cocteau  staff        1122  2 Oct 15:15 lec6.key
drwxr-xr-x   11 cocteau  staff         374  2 Oct 15:49 lecture1
drwxr-xr-x   86 cocteau  staff        2924  2 Oct 15:57 lecture1.key
drwxr-xr-x    4 cocteau  staff         136  4 Oct 09:41 lecture2
drwxr-xr-x   32 cocteau  staff        1088  4 Oct 10:10 lecture2.key
drwxr-xr-x   35 cocteau  staff        1190  4 Oct 08:44 lecture3.key
drwxr-xr-x  152 cocteau  staff        5168  2 Oct 15:08 mtd.key
```

Your user name

The group you belong to that owns the file

The file's size in bytes

The file's creation date

The file's name

Kinds of files

What you'll notice right away is that there are different types of files having different permissions

Unix filesystem conventions places (shared, commonly used) executable files in places like `/usr/bin` or `/usr/local/bin`

Different files are opened by different kinds of programs; in OSX, there is a beautiful command called `open` that decides which program to use

Kinds of files

Filenames which contain special characters like * and ~ are candidates for *filename substitution*

~ refers to your home directory and * is a wildcard for any number of characters

Other special characters like {, [and ? can also be expanded, but we'll get to them when we learn a bit more about regular expressions

Unix shells

There are many flavors of Unix Shells; that is, there are many kinds of programs that operate as shells

```
sh, csh, tcsh, bash, ksh
```

They are all programs and can be found on the file system

```
which sh
```

An example: HTTP access logs

`www.stat.ucla.edu`

The department runs an Apache Web server running on `taia.stat.ucla.edu`

Each request, each click by a user out on the Internet browsing our site, is logged

There are standards for these files, but in general, they can be a bit hairy to “parse”

Data

The students (if I'm in the lab) will have a data set (`access_log.txt`) pre-loaded for them; otherwise I point them to the location of the data and we (inevitably) have to talk about downloading, etc.

`http://www.stat.ucla.edu/~cocteau/stat202a/data`

HTTP access logs

A bit of digging...

Commands

`pwd, ls, cd`

`more/less, tail, wc`

`cut, sort, uniq`

```
% head access_log.txt

134.226.32.57 - - [20/Sep/2007:07:54:29 -0700] "GET /~sczhu/icons/daught.gif HTTP/1.0" 200 1898 "http://www.stat.ucla.edu/~sczhu/" "Mozilla/
134.226.32.57 - - [20/Sep/2007:07:54:29 -0700] "GET /~sczhu/icons/bio.gif HTTP/1.0" 200 1681 "http://www.stat.ucla.edu/~sczhu/" "Mozilla/5.0
134.226.32.57 - - [20/Sep/2007:07:54:30 -0700] "GET /~sczhu/Zhu_LA_sm.gif HTTP/1.0" 200 39313 "http://www.stat.ucla.edu/~sczhu/" "Mozilla/5.
134.226.32.57 - - [20/Sep/2007:07:54:30 -0700] "GET /favicon.ico HTTP/1.0" 200 318 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1
74.6.28.138 - - [20/Sep/2007:07:54:50 -0700] "GET /~nchristo/statistics100B/syllabus100b.pdf HTTP/1.0" 200 47206 "-" "Mozilla/5.0 (compatibl
164.67.132.219 - - [20/Sep/2007:07:54:55 -0700] "GET /robots.txt HTTP/1.0" 200 559 "-" "gsa-crawler%20%28gsa1%2C%20contact%3A%20jhuang%40ais
164.67.132.219 - - [20/Sep/2007:07:54:55 -0700] "GET /rss/feed.php?unit=uclastat HTTP/1.0" 200 1739 "-" "gsa-crawler%20%28gsa1%2C%20contact%
134.226.32.57 - - [20/Sep/2007:07:55:03 -0700] "GET /%7Esczhu/talks.html HTTP/1.0" 200 9489 "http://www.stat.ucla.edu/~sczhu/" "Mozilla/5.0
134.226.32.57 - - [20/Sep/2007:07:55:03 -0700] "GET /%7Esczhu/icons/back2.gif HTTP/1.0" 200 17061 "http://www.stat.ucla.edu/%7Esczhu/talks.h
134.226.32.57 - - [20/Sep/2007:07:55:03 -0700] "GET /%7Esczhu/icons/bio.gif HTTP/1.0" 200 1681 "http://www.stat.ucla.edu/%7Esczhu/talks.html


% wc access_log.txt

200000 3890201 46321543 access.txt


% tail access_log.txt

76.169.68.146 - - [26/Sep/2007:19:31:57 -0700] "GET /graphics/rss20.gif HTTP/1.1" 200 219 "http://www.stat.ucla.edu/" "Mozilla/4.0 (compatib
76.168.75.194 - - [26/Sep/2007:19:31:58 -0700] "GET /favicon.ico HTTP/1.1" 304 - "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.12) G
217.212.224.159 - - [26/Sep/2007:19:31:58 -0700] "GET /~dinov/courses_students.dir/PIC20_Summer00.dir/docs/appenda/?M=D HTTP/1.0" 200 814 "-
68.180.251.16 - - [26/Sep/2007:19:32:12 -0700] "GET /~dinov/courses_students.dir/Applets.dir/Normal_T_Chi2_F_Tables.htm HTTP/1.0" 200 11205
68.180.251.16 - - [26/Sep/2007:19:32:12 -0700] "GET /index.php HTTP/1.0" 200 17447 "-" "Wget/1.10.2 (Red Hat modified)"
74.6.24.11 - - [26/Sep/2007:19:32:27 -0700] "GET /~ktranbar/deptphotos-Pages/Image35.html HTTP/1.0" 200 480 "-" "Mozilla/5.0 (compatible; Ya
216.125.49.252 - - [26/Sep/2007:19:32:39 -0700] "GET / HTTP/1.1" 200 17335 "http://www.mhhe.com/biosci/cellmicro/lewis4e/student/casehist.mh
216.125.49.252 - - [26/Sep/2007:19:32:39 -0700] "GET /index.css HTTP/1.1" 200 5869 "http://www.stat.ucla.edu/" "Mozilla/4.0 (compatible; MSI
216.125.49.252 - - [26/Sep/2007:19:32:39 -0700] "GET /css/uclastat/site.css HTTP/1.1" 200 4822 "http://www.stat.ucla.edu/" "Mozilla/4.0 (com
216.125.49.252 - - [26/Sep/2007:19:32:39 -0700] "GET /graphics/rss20.gif HTTP/1.1" 200 219 "http://www.stat.ucla.edu/" "Mozilla/4.0 (compati
```

Combined log format

- IP address
- Identity
- Userid
- date
- Request
- Status
- Bytes
- Referrer
- Agent

Unix pipes

Programs usually take some kind of input and generate some kind of output

Unix tools often take input from the user and print the output to the screen

“Redirection” of data to and from files or programs is controlled by pipes

Redirecting output with “|”

Takes output from one command and submits it as input to the next command

Examples

```
cut -d” “ -f1,10 access_log.txt
```

```
cut -d” “ -f9 access_log.txt
```

```
% cut -d" " -f9 access_log.txt | head

200
200
200
200
302
200
304
200
200
200
200
200
200
200
200
200
200
200
302
200
200
200
200
200 ...
```

In general...

```
cut -d” “ -f1,5,9   select the first, fifth and ninth
cut -d” “ -f1-5     select the first through the fifth
cut -d” “ -f1       select just the first
```

Sending output to a file with “>”

With this form of redirection, we take a stream of processed data and store it in a file

Example

```
cut -d” “ -f1 access_log.txt > ips.txt
```

Taking input from a file with “<”

With this form of redirection, we create an input stream from a file

Example

```
wc < access_log.txt
```

The pipeline

As the name might imply, you can connect pipes and have data stream from process to process

Example

```
cut -d" " -f1 access_log.txt | sort | uniq -c | sort -rn

cut -d" " -f9 access_log.txt | sort | uniq -c

cut -d" " -f1 access_log.txt | sort | uniq | wc
```

```
% cut -d “ “ -f1 access_log.txt | sort | uniq | wc

17128   17128  238213


% cut -d” “ -f9 access_log.txt | sort | uniq -c | sort -rn


158760 200
 16161 304
  9690 206
  6794 404
  6043 301
  1652 403
   836 302
    25 401
    19 405
    12 500
     4 400
     3 501
     1 416
```

What are these numbers?

A fast Google search gives us a list of possible errors

Note that Error 200 actually means a success

Error 206 means that only part of the file was delivered; the user cancelled the request before it could be delivered

Error 304 is “not modified”; sometimes clients perform conditional GET requests

HTTP Error 101
Switching Protocols. Again, not really an "error", this HTTP Status Code means everything is working fine.

HTTP Error 200
Success. This HTTP Status Code means everything is working fine. However, if you receive this message on screen, obviously something is not right... Please contact the server's administrator if this problem persists. Typically, this status code (as well as most other 200 Range codes) will only be written to your server logs.

HTTP Error 201
Created. A new resource has been created successfully on the server.

HTTP Error 202
Accepted. Request accepted but not completed yet, it will continue asynchronously.

HTTP Error 203
Non-Authoritative Information. Request probably completed successfully but can't tell from original server.

HTTP Error 204
No Content. The requested completed successfully but the resource requested is empty (has zero length).

HTTP Error 205
Reset Content. The requested completed successfully but the client should clear down any cached information as it may now be invalid.

HTTP Error 206
Partial Content. The request was canceled before it could be fulfilled. Typically the user gave up waiting for data and went to another page. Some download accelerator programs produce this error as they submit multiple requests to download a file at the same time.

HTTP Error 300
Multiple Choices. The request is ambiguous and needs clarification as to which resource was requested.

HTTP Error 301
Moved Permanently. The resource has permanently moved elsewhere, the response indicates where it has gone to.

HTTP Error 302
Moved Temporarily. The resource has temporarily moved elsewhere, the response indicates where it is at present.

HTTP Error 303
See Other/Redirect. A preferred alternative source should be used at present.

```
% cut -d" " -f1 access_log.txt | sort | uniq -c | sort -rn | more

13050 70.184.223.117
 8086 164.67.132.219
 4227 164.67.132.220
 2304 128.97.86.248
 1661 128.97.55.194
 1360 66.249.73.99
 1161 128.97.55.208
 1081 208.68.136.250
 1064 207.46.98.57
  956 76.167.214.187
  808 207.46.98.56
  763 87.237.114.11
  757 207.46.98.58
  720 63.241.61.68
  668 61.149.63.50
  569 164.67.134.26
  548 69.12.181.75
  518 196.1.114.240
  513 65.55.209.79
  505 76.167.183.169
  503 65.55.209.83
  497 217.212.224.159
  496 76.168.72.146
  487 65.55.209.82
  478 65.55.209.78
  473 65.55.209.80
```

The pipeline

In 1972, pipes appear in Unix, and with them a philosophy, albeit after some struggle for the syntax; should it be

```
more(sort(cut)))
```

[Remember this; S/R has this kind of functional syntax]

The development of pipes led to the concept of tools -- software programs that would be in a tool box, available when you need them

"And that's, I think, when we started to think consciously about tools, because then you could compose things together... compose them at the keyboard and get them right every time."

from an interview with Doug McIlroy

Read the man pages!

If the command `uniq` is unfamiliar, you can look up its usage

Example

```
man uniq
man host
```

xterm

```
UNIQ(1)                  BSD General Commands Manual                 UNIQ(1)

NAME
     uniq - report or filter out repeated lines in a file

SYNOPSIS
     uniq [-c | -d | -u] [-f fields] [-s chars] [input_file [output_file]]

DESCRIPTION
     The uniq utility reads the standard input comparing adjacent lines, and
     writes a copy of each unique input line to the standard output.  The sec-
     ond and succeeding copies of identical adjacent input lines are not writ-
     ten.  Repeated lines in the input will not be detected if they are not
     adjacent, so it may be necessary to sort the files first.

     The following options are available:

     -c      Precede each output line with the count of the number of times
             the line occurred in the input, followed by a single space.

     -d      Don't output lines that are not repeated in the input.

     -f fields
:
```

```
% host 70.184.223.117
117.223.184.70.in-addr.arpa domain name pointer
wsip-70-184-223-117.om.om.cox.net.

% host 164.67.132.219
219.132.67.164.in-addr.arpa domain name pointer
gsa1.ais.ucla.edu.

% host 66.249.73.99
99.73.249.66.in-addr.arpa domain name pointer
crawl-66-249-73-99.googlebot.com..
```

```
whois  70.184.223.117
Cox Communications Inc. NETBLK-COX-ATLANTA-10 (NET-70-160-0-0-1)
                                  70.160.0.0 - 70.191.255.255
Cox Communications NETBLK-OM-CBS-70-184-208-0 (NET-70-184-208-0-1)
                                  70.184.208.0 - 70.184.223.255

# ARIN WHOIS database, last updated 2007-09-26 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
blowtorch:~ cocteau$ whois 66.249.73.99

OrgName:    Google Inc.
OrgID:      GOGL
Address:    1600 Amphitheatre Parkway
City:       Mountain View
StateProv:  CA
PostalCode: 94043
Country:    US

NetRange:   66.249.64.0 - 66.249.95.255
CIDR:       66.249.64.0/19
NetName:    GOOGLE
NetHandle:  NET-66-249-64-0-1
Parent:     NET-66-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.GOOGLE.COM
NameServer: NS2.GOOGLE.COM
NameServer: NS3.GOOGLE.COM
NameServer: NS4.GOOGLE.COM
Comment:
RegDate:    2004-03-05
Updated:    2007-04-10

OrgTechHandle: ZG39-ARIN
OrgTechName:   Google Inc.
OrgTechPhone:  +1-650-318-0200
OrgTechEmail:  arin-contact@google.com

# ARIN WHOIS database, last updated 2007-09-26 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

But what are we really after?

Rather than splitting up the data in access_log.txt by day, we might consider dividing it by IP

Once we have such a thing, we can use the command `wc` to tell us about the number of accesses from each user

We can also start to “fit” user-level models that can be used to predict navigation

Rudimentary pattern matching

`grep` can be used to skim lines from a file that have (or don’t have) a particular pattern

Patterns are specified via regular expressions, something we will learn more about later

The name comes from an editing operation on Unix: g/re/p

Example

```
grep 85.249.135.15 access_log.txt
grep /~dinov access_log.txt
```

```
% grep 70.184.223.117 access_log.txt| more
```

```
70.184.223.117 - - [20/Sep/2007:13:10:16 -0700] "GET / HTTP/1.1" 200 16974 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:20 -0700] "GET /graphics/rss20.gif HTTP/1.1" 200 219 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:21 -0700] "GET /index.css HTTP/1.1" 200 5869 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:21 -0700] "GET /css/uclastat/site.css HTTP/1.1" 200 4822 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:22 -0700] "GET /rss/feed.php?unit=uclastat HTTP/1.1" 200 1751 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:23 -0700] "GET /centers HTTP/1.1" 301 323 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:24 -0700] "GET /centers/ HTTP/1.1" 200 6509 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:27 -0700] "GET /program/faq.php HTTP/1.1" 200 18662 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:27 -0700] "GET /graphics/point.gif HTTP/1.1" 200 2397 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:27 -0700] "GET /research HTTP/1.1" 301 324 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:28 -0700] "GET /research/ HTTP/1.1" 200 552 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:28 -0700] "GET /research/index_head.php HTTP/1.1" 200 690 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:29 -0700] "GET /research/index_body.php HTTP/1.1" 200 3712 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:30 -0700] "GET /visitors HTTP/1.1" 301 324 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:30 -0700] "GET /visitors/ HTTP/1.1" 200 2016 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:31 -0700] "GET /library HTTP/1.1" 301 323 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:31 -0700] "GET /library/ HTTP/1.1" 200 550 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:31 -0700] "GET /library/index_head.php HTTP/1.1" 200 927 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:32 -0700] "GET /library/index_body.php HTTP/1.1" 200 3261 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:33 -0700] "GET /noteworthy HTTP/1.1" 301 326 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:33 -0700] "GET /noteworthy/ HTTP/1.1" 200 556 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:33 -0700] "GET /noteworthy/index_head.php HTTP/1.1" 200 659 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:34 -0700] "GET /noteworthy/index_body.php HTTP/1.1" 200 3336 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:35 -0700] "GET /alumni HTTP/1.1" 301 322 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:35 -0700] "GET /alumni/ HTTP/1.1" 200 548 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:35 -0700] "GET /alumni/index_head.php HTTP/1.1" 200 870 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:36 -0700] "GET /alumni/index_body.php HTTP/1.1" 200 2369 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:37 -0700] "GET /cases HTTP/1.1" 301 321 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:37 -0700] "GET /cases/ HTTP/1.1" 200 546 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:37 -0700] "GET /cases/index_head.php HTTP/1.1" 200 639 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:38 -0700] "GET /cases/index_body.php HTTP/1.1" 200 4366 "-" "Mozilla/4.0 (compatible)"
70.184.223.117 - - [20/Sep/2007:13:10:39 -0700] "GET /data HTTP/1.1" 301 320 "-" "Mozilla/4.0 (compatible)"
```

```
% grep 70.184.223.117 access_log.txt | grep -v library
```

Quien es mas macho?

In online marketing, hits rule the roost; the more raw traffic you attract, the greater your opportunities for making a sale

immense HITS

Alright, so it’s not as simple as that, but let’s see what we can learn about centers of activity on our website

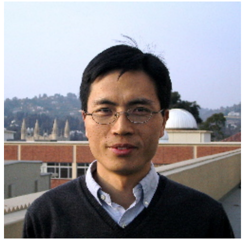
Quien es mas macho?

Compute the number of hits to the portions of the site owned by Song-Chun Zhu, Vivian Lew, Brian Kriegler, Debbie Barrera and Ivo Dinov

Who received the most hits last week?

What can you say about the kinds of files that were downloaded?

What was the most popular portion of each site?



Then...

Pull back a little and tell me about the site and the habits of its visitors; specifically, think about

When is the site active? When is it quiet?

Do the visitors stay for very long? Do they download any of our papers or software? What applications do they run?

On the balance, is our traffic “real” or mostly the result of robots or automated processes?

A second data set

We have assembled a list of all the bylines associated with articles appearing in the New York Times in 1950

Some of this was entered by hand when the archive was scanned into digital form, but that doesn't mean the data are clean!

We have a simple task: Provide me with a list of journalists and the number of items they wrote in 1950

```
% head 1950.txt

By PAUL CROWELL
The New York Times (by Edward Hausner)
By LEE E.COOPER
By JOHN D. MORRIS Special to THE NEW YORK TIMES.
By KALMAN SEIGEL
By HAROLD FABER
The New York Times
By FELIX BELAIR Jr. Special to THE NEW YORK TIMES.
By LINDESAY PARROTT Special to THE NEW YORK TIMES.
Special to THE NEW YORK TIMES.


% wc 1950.txt

53203  317055 1703966 1950.txt


% sort 1950.txt | uniq -c | sort -rn | head

26194 Special to THE NEW YORK TIMES.
 1452 The New York Times
  392 By The Associated Press.
  369 Special to THE NEW YORK TIMES
  263 By THOMAS F. BRADY Special to THE NEW YORK TIMES.
  258 By ARTHUR DALEY
  223 Bradford Bachrach
  203 By LINDESAY PARROTT Special to THE NEW YORK TIMES.
  202 The New York Times Studio
  199 The New York Times (Washington Bureau)
  188 By RAYMOND R. CAMP
  188 By ORVILLE PRESCOTT
  187 By ARTHUR KROCK
 183 By WILLIAM S. WHITE Special to THE NEW YORK TIMES.
 178 By BOSLEY CROWTHER
 172 By DREW MIDDLETON Special to THE NEW YORK TIMES.
 169 By HAROLD CALLENDER Special to THE NEW YORK TIMES.
```

NYT Data set

We have a simple task: Provide me with a list of journalists and the number of items they wrote in 1950

For this week, simply have a look at the data and anticipate complications that you might encounter when taking on the somewhat simply-stated accounting operation


GAP

From machine to machine

Before we get on with today's lecture, there are a couple dangling topics we should mention

In your other courses (and in the bootcamp a week ago) you will run/ran R on the computer on your desk

The Statistics Department has a number of computers available to you that are more powerful (memory, speed) than those on your desk

You can navigate between machines by invoking `ssh` and move files using `scp` (example shortly)

xterm

```
[fad-gadget ~] ssh otter.berkeley.edu
Password:
Linux Debian/Ubuntu 2.6.12-1-amd64-k8-smp #1 SMP Wed Sep 28 CEST 2005 x86_64

  Commands and programs that run under Solaris may not be available
  or may behave differently under GNU/Linux.

You have new mail.
Last login: Mon Oct  9 09:54:42 2006 from 169.232.148.1

   Please visit http://www.stat.berkeley.edu/trouble when reporting problems.

   NOTICE: The SCF modems are temporarily out of service.

>> SYSTEM DOWNTIME: The server will be rebooted at noon on Monday, Oct 9.

otter.berkeley.edu 1> ls *.txt
abstractraphael.txt  dj.txt  fri.txt        infl.txt  resp.txt    trace.txt
cmu.abs.txt          fl.txt  ignorable.txt  list.txt  source.txt  tt.txt
otter.berkeley.edu 2> 
```

```
% ssh otter.berkeley.edu
```

Running jobs

Last time we discussed some basic facts about operating systems; a large part of their functioning is devoted to managing jobs (or programs) run by different users

We used the `top` command to give a dynamic display of what was running, how much of the computer's resources it was using up, etc.

```
Processes:  52 total, 2 running, 50 sleeping... 161 threads          09:25:40
Load Avg:  0.00, 0.00, 0.00     CPU usage:  0.9% user, 3.6% sys, 95.5% idle
SharedLibs: num =  146, resident = 27.9M code, 3.59M data, 8.14M LinkEdit
MemRegions: num =  9586, resident =  130M + 5.73M private, 45.4M shared
PhysMem:   201M wired, 82.1M active,  161M inactive,  445M used, 2.07G free
VM: 4.08G + 91.4M   31653(0) pageins, 0(0) pageouts

  PID COMMAND      %CPU   TIME   #TH #PRTS #MREGS RPRVT  RSHRD  RSIZE  VSIZE
27487 top          9.2%  0:02.50   1    19     22   424K  3.91M  2.29M  26.9M
27482 tcsh         0.0%  0:00.16   1    15     22   580K  4.16M  1.09M  31.1M
27481 sshd         0.0%  0:00.01   1    11     40   108K  4.78M   496K  29.9M
27477 sshd         0.0%  0:00.13   1    18     40   108K  4.78M  1.19M  30.0M
26416 tcsh         0.0%  0:00.32   1    15     22   580K  4.16M  1.10M  31.1M
26415 sshd         0.0%  0:00.42   1    11     40   112K  4.78M   496K  29.9M
26409 sshd         0.0%  0:00.19   1    18     40   100K  4.78M  1.15M  30.0M
19378 httpd        0.0%  0:00.01   1    12    196   256K  16.3M  1.45M  62.3M
19328 httpd        0.0%  0:00.01   1    12    196   388K  16.3M  1.62M  62.3M
18444 check_afp    0.0%  0:00.01   2    24     21   160K  4.21M  1.98M  27.1M
  382 AppleVNCSe   0.0%  3:18.57   7   >>>     49  1.27M  4.32M  2.15M   159M
  381 ARDAgent     0.0%  0:02.73   6    95     78   984K  4.85M  2.16M   192M
  377 SecurityAg   0.0%  0:06.17   1    71    115  3.05M  12.6M  10.1M   216M
  376 authorizat   0.0%  0:00.68   1    22     24   260K  4.35M   968K  27.3M
  375 mcxd         0.0%  2:24.04   1    43     55  17.0M  5.71M  17.6M   179M
  371 WindowServ   0.0%  3:32.31   3   109    108  1.35M  11.4M  6.91M   195M
```

Running top on lab-compute.stat.ucla.edu

How many processes are running?

How much RAM is available?

cocteau@taia.stat.ucla.edu

```
Processes:  163 total, 1 zombie, 2 running, 1 stuck, 159 sleeping... 3 09:30:47
Load Avg:  0.15, 0.05, 0.01     CPU usage:  3.2% user, 7.3% sys, 89.5% idle
SharedLibs: num =  123, resident = 30.6M code, 2.02M data, 9.75M LinkEdit
MemRegions: num = 19151, resident =  256M + 5.95M private, 31.8M shared
PhysMem:   187M wired,  692M active, 1.06G inactive, 1.92G used, 83.4M free
VM: 6.82G +  100M   46013057(0) pageins, 913985(0) pageouts

  PID COMMAND      %CPU   TIME   #TH #PRTS #MREGS RPRVT  RSHRD  RSIZE  VSIZE
29028 rotatelogs   0.0%  0:00.01   1    11    15    80K   376K   280K  17.6M
25342 httpd        0.0%  8:51.55   1    10   503  27.0M  8.61M  29.2M  86.7M
25335 httpd        0.0%  8:28.11   1    10   468  22.2M  11.4M  27.2M  86.7M
25333 httpd        0.0%  9:12.30   1    10   477  27.1M  8.61M  30.2M  86.7M
25331 httpd        0.0%  8:53.57   1    10   417  24.0M  8.61M  21.2M  86.7M
25330 rotatelogs   0.0%  0:02.89   1    12    16   108K   380K   316K  17.6M
25329 rotatelogs   0.0%  0:00.83   1    12    16   104K   380K   312K  17.6M
25328 rotatelogs   0.0%  0:00.03   1    12    16   104K   380K   312K  17.6M
25327 rotatelogs   0.0%  0:00.03   1    12    16   104K   380K   312K  17.6M
25326 rotatelogs   0.0%  0:00.22   1    12    16   104K   380K   312K  17.6M
25325 rotatelogs   0.0%  0:00.00   1    12    16   104K   380K   312K  17.6M
25324 rotatelogs   0.0%  0:00.03   1    11    15    80K   376K   280K  17.6M
25323 rotatelogs   0.0%  0:00.45   1    12    16   104K   380K   312K  17.6M
25322 rotatelogs   0.0%  0:00.03   1    12    16   104K   380K   312K  17.6M
25321 rotatelogs   0.0%  0:00.01   1    12    16   104K   380K   312K  17.6M
25320 rotatelogs   0.0%  0:00.03   1    11    15    80K   376K   280K  17.6M
```

Running `top` on `taia.stat.ucla.edu`

How many processes are running?

How much RAM is available?

What do you reckon this computer does?

Another way to get at processes

While `top` gives you a dynamic, um constantly updating, view of what the processor is doing, you can use the command `ps` to give you a snapshot

The command `ps` has lots and lots of options; it lets you look at all users, just a specific user and control the format of the output

Just typing `ps` will give you the processes that you started (or were started on your behalf); we can also see what others are up to (ah, the joys of a multi-user system) cocteau@login

```
[login:~] cocteau% ps -aux | grep -v root | grep -v postfix
USER       PID %CPU %MEM      VSZ    RSS  TT  STAT STARTED      TIME COMMAND
cocteau    557   1.1 -0.0   30740    464  ??  S      9:29AM   0:00.09 /usr/sbin/sshd -i
daemon     121   0.0 -0.0   18084    444  ??  Ss     6Aug06   0:01.07 portmap
nobody     197   0.0 -0.1   28364   1096  ??  Ss     6Aug06   9:21.11 /usr/sbin/mDNSResponder
chea     19976   0.0 -0.0   28420    504  p7- S     11Sep06   0:06.55 ssh cvs.programmers.ucla.edu -l chea cvs server
rosario   8192   0.0 -0.0   30740    256  ??  S     20Sep06   0:00.99 /usr/sbin/sshd -i
rosario   8194   0.0 -0.0   22684    816  p0  Ss+   20Sep06   0:00.50 -tcsh
rrojas   23080   0.0 -0.0   30740    116  ??  S     23Sep06   0:00.01 /usr/sbin/sshd -i
arno     21414   0.0 -0.0   30740    440  ??  S     Thu11AM   0:00.12 /usr/sbin/sshd -i
arno     21415   0.0 -0.1   22684   1096  p8  Ss    Thu11AM   0:00.21 -tcsh
www      10870   0.0 -1.3   72164  27336  ??  S     Sat04AM   4:16.93 /usr/sbin/httpd
www      11227   0.0 -1.3   72164  27352  ??  S     Sat05AM   4:11.04 /usr/sbin/httpd
www      11228   0.0 -1.2   70076  25084  ??  S     Sat05AM   4:05.80 /usr/sbin/httpd
www      11527   0.0 -1.3   72164  27412  ??  S     Sat05AM   4:11.38 /usr/sbin/httpd
www      16196   0.0 -1.3   72164  27328  ??  S     Sat10AM   3:42.20 /usr/sbin/httpd
zzsi     10625   0.0 -0.2   58120   3156  ??  S     12:53PM   2:23.36 /usr/sbin/smbd -D
arno     27118   0.0 -0.1   34884   2424  p8  S+     6:19AM   0:00.24 pine
erickson   188   0.0 -0.0   30740    456  ??  S      9:23AM   0:00.21 /usr/sbin/sshd -i
erickson   189   0.0 -0.1   22684   1092  p1  Ss+    9:23AM   0:00.19 -tcsh
cocteau    558   0.0 -0.1   22684   1076 std  Ss     9:29AM   0:00.16 -tcsh
[login:~] cocteau%
```

Running “`ps -aux`” on `login.stat.ucla.edu`

What can we see?

`Who are the users?`

* the option -a gives you information on all users, -u gives you a popular view (fields) of the processes, and -x gives you processes that aren’t necessarily associated with a terminal -- this output has been edited slightly with some “grep -v”’s

Job control

Unix allows you to run several processes at once; each process is given a number which you can use to change the status of the process

Because many jobs are running on the computer, the amount of “attention” they get from the central processing unit is controlled by their priorities (-20 to 20, with the higher the number meaning the lower the priority)

`nice` and `renice` lets you lower the priority on a job that you know will run for a long time, freeing system resources for others; `kill` can be used to end processes (politely or with a greater sense of urgency)

xterm

```
[fad-gadget ~] R

R : Copyright 2004, The R Foundation for Statistical Computing
Version 1.9.1  (2004-06-21), ISBN 3-900051-00-3

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> 
```

xterm

```
[fad-gadget ~] ps -o user,pid,nice,command | grep R
USER       PID NI COMMAND
cocteau 10236  0 /Library/Frameworks/R.framework/Resources/bin/R.bin
cocteau 10406  0 grep R

[fad-gadget ~] renice 15 -p 10236
10236: old priority 0, new priority 15

[fad-gadget ~] ps -o user,pid,nice,command | grep R
USER       PID NI COMMAND
cocteau 10236 15 /Library/Frameworks/R.framework/Resources/bin/R.bin
cocteau 10412  0 grep R
[fad-gadget ~] 
```

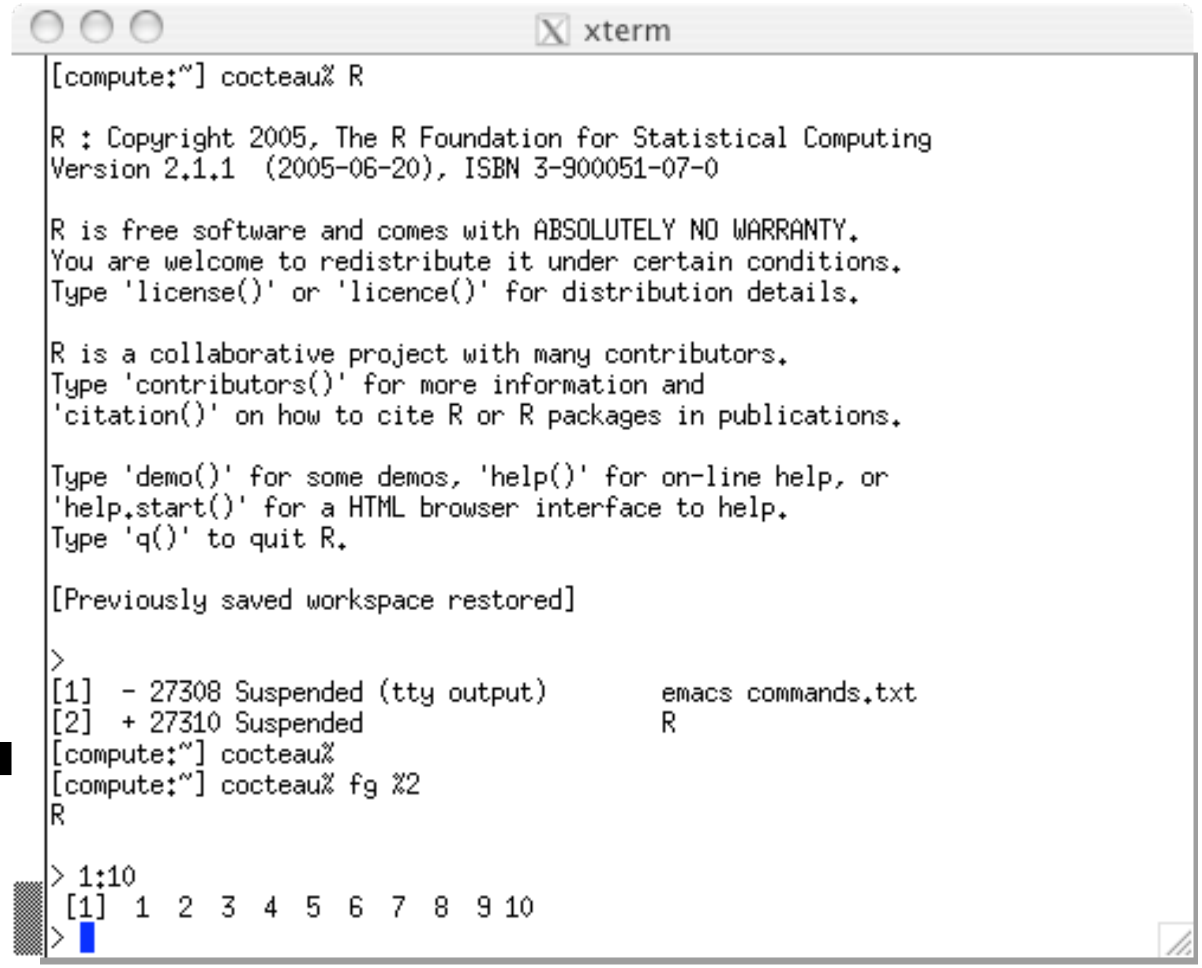
Job control

C-z stops jobs, C-c kills them, and C-d kills your shell

You can also set jobs to run in the *background* (which means your prompt returns)

The command `jobs` lets you see what jobs you have running

If you stop a job, you can restart it or restart it in the background using the commands `bg` and `fg`



Cntl-z hit here

In response, Unix stops the job and gives you a list of other jobs you have stopped or are running in the background

GAP

We get to running R on a file of commands, running R in batch mode and so on later in the quarter...

Back to your “assignments” from last time

Who had the most hits and how did you compute it?

What about the open-ended questions... What are the active periods of the day? What did you learn?

Take about a half an hour to discuss what you found

Also...

I might not have emphasized enough that each of the commands we worked with can either take input from a file as in

```
sort access_log.txt
```

or from a pipe (so-called standard input), as in

```
grep dinov access_log.txt | sort
```

New data

We are now going to look at a series of log files containing data from chat sessions recorded last year; in a two-hour recording session last year we captured 46,000 lines from 4,500 people

The data are stored on `lab-compute.stat.ucla.edu` and you can bring them to your desktop with the command

```
scp -r lab-compute.stat.ucla.edu:/Data/chat .
```

copy recursively (a directory and all its contents)

copy the directory `/Data/chat` from the computer `lab-compute.stat.ucla.edu`

copy it to a directory of the same name (in this case "`chat`") on your local machine

xterm

```
[fad-gadget ~] cd chat
[fad-gadget ~/chat] ls -l
total 11640
-rw-r--r--  1 cocteau  staff  507715  9 Oct 12:40 now.1159914021.txt
-rw-r--r--  1 cocteau  staff  555894  9 Oct 12:40 now.1159917841.txt
-rw-r--r--  1 cocteau  staff  454160  9 Oct 12:40 now.1159920765.txt
-rw-r--r--  1 cocteau  staff  447638  9 Oct 12:40 now.1159924151.txt
-rw-r--r--  1 cocteau  staff  450546  9 Oct 12:40 now.1159927582.txt
-rw-r--r--  1 cocteau  staff  529160  9 Oct 12:40 now.1159992088.txt
-rw-r--r--  1 cocteau  staff  479573  9 Oct 12:40 now.1159995531.txt
-rw-r--r--  1 cocteau  staff  407640  9 Oct 12:40 now.1159998154.txt
-rw-r--r--  1 cocteau  staff  613252  9 Oct 12:40 now.1160410547.txt
-rw-r--r--  1 cocteau  staff  540785  9 Oct 12:40 now.1160413861.txt
-rw-r--r--  1 cocteau  staff  494676  9 Oct 12:40 now.1160416555.txt
-rw-r--r--  1 cocteau  staff  456485  9 Oct 12:40 now.1160419326.txt
[fad-gadget ~/chat] wc *
   10000   92622  507715 now.1159914021.txt
   10000  101173  555894 now.1159917841.txt
   10000   82973  454160 now.1159920765.txt
   10000   81708  447638 now.1159924151.txt
   10000   82399  450546 now.1159927582.txt
   10000   96956  529160 now.1159992088.txt
   10000   88012  479573 now.1159995531.txt
   10000   74815  407640 now.1159998154.txt
   10000  111328  613252 now.1160410547.txt
   10000   98653  540785 now.1160413861.txt
   10000   89669  494676 now.1160416555.txt
   10000   83455  456485 now.1160419326.txt
  120000 1083763 5937524 total
[fad-gadget ~/chat] 
```

Rudimentary pattern matching

We have already seen some basic pattern matching notions; recall the command "`wc *.txt`"

In this expression "`*`" acts as a *wildcard* and matches anything

The files `now.1159914021.txt`, `now.1159927582.txt`, `now.1160410547.txt`, `now.1159917841.txt`, `now.1159992088.txt`, `now.1160413861.txt`, `now.1159920765.txt`, `now.1159995531.txt`, `now.1160416555.txt`, `now.1159924151.txt`, and `now.1159998154.txt` will all be returned by this command

Rudimentary pattern matching

In the expression "`*.txt`" we can name two kinds of characters

The "`.txt`" is made up of *literal* or normal text characters

The "`*`" is a *metacharacter*

GAP

We get to running R on a file of commands, running R in batch mode and so on later in the quarter...

Before the break...

We discussed your experiments with the Web server data

How would we get a time series of hits per day?
Who had the largest number of hits?
What can you say about the files being accessed?

The command `grep` let us extract lines from a file that contained a string of characters; as we started digging into the data, we wanted a more expressive tool for defining patterns

Before the break, we discussed so-called *regular expressions,* a language for describing patterns in text data

And in this session...

Sidestepping the issue of a text corpus somewhat, we are now in a good position to start looking at the Enron email data set

Initially, we have to understand the structure of these data before we dig a bit into the social networking “analysis” that is to come

It will also give us an opportunity to consider simple shell scripts; a mechanism by which we collect commands into reusable programs

Enron

Today we are going to start our work on a set of data related to the Enron corporation

Some relevant links are

http://www.chron.com/news/specials/enron/timeline.html

http://www.cs.cmu.edu/~enron/

http://www.stat.ucla.edu/~cocteau/klimt-ecml04-1.pdf

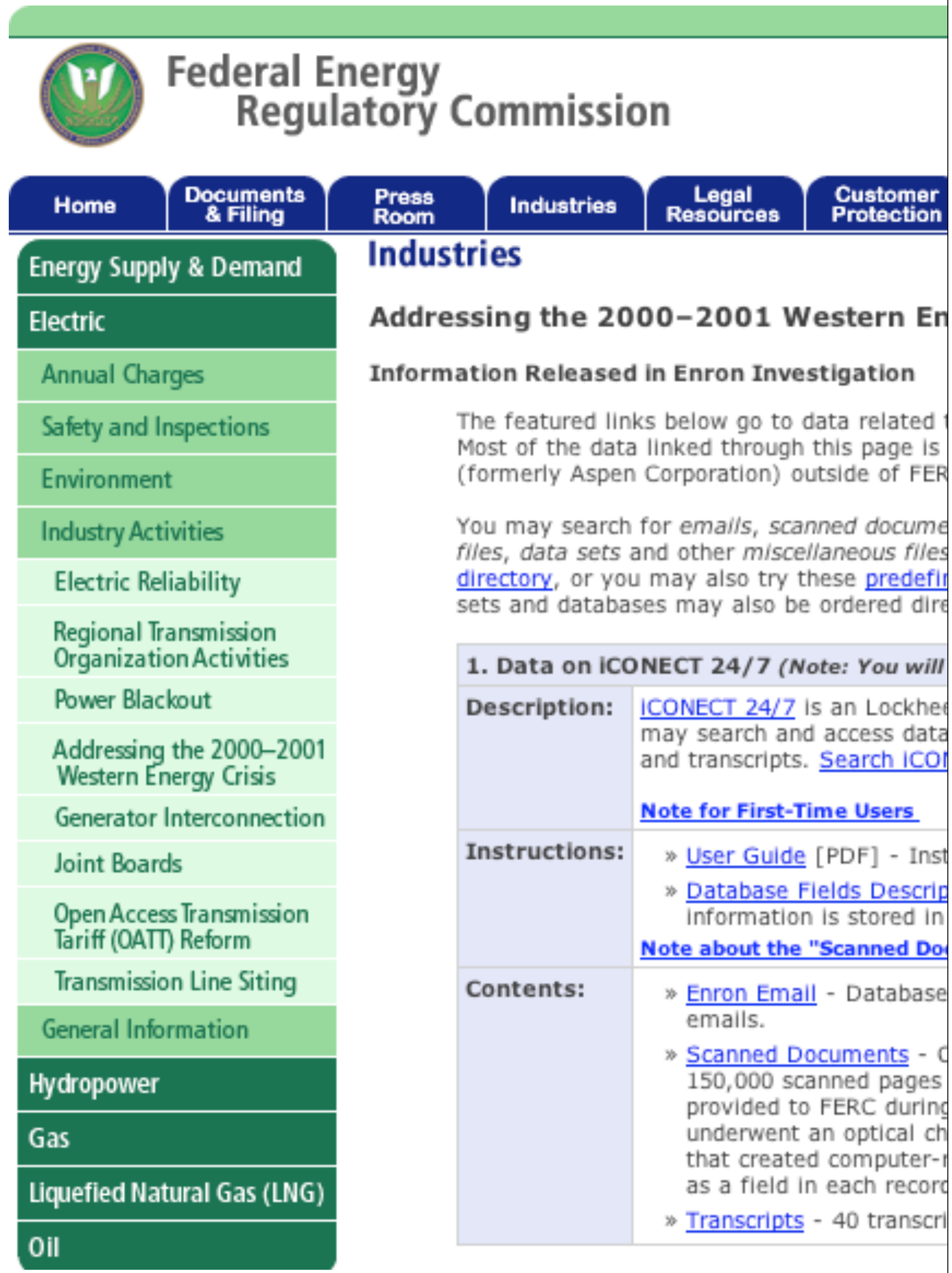
http://www.stat.ucla.edu/~cocteau/Enron_Employee_Status.htm

Enron emails

As part of its investigation into Enron, the Federal Energy Regulatory Commission released the emails of about 150 of its top executives

These data were then cleaned up by groups at MIT and SRI and are now publicly available through the CMU CS Department

To respect the privacy of the individuals involved, I have replaced the body of each email with x's; our interest is not in what was said but who sent email to whom



Organization of the data

The data itself is organized into a series of directories, each named after an executive

Under each directory, you will find possibly more directories, each representing a different mail folder

At the lowest level, you have a series of email messages, one per file; the files in each directory are named 1., 2., 3., etc.

The files we will work with are in `/Data/mailfiles` on `lab-compute`

xterm

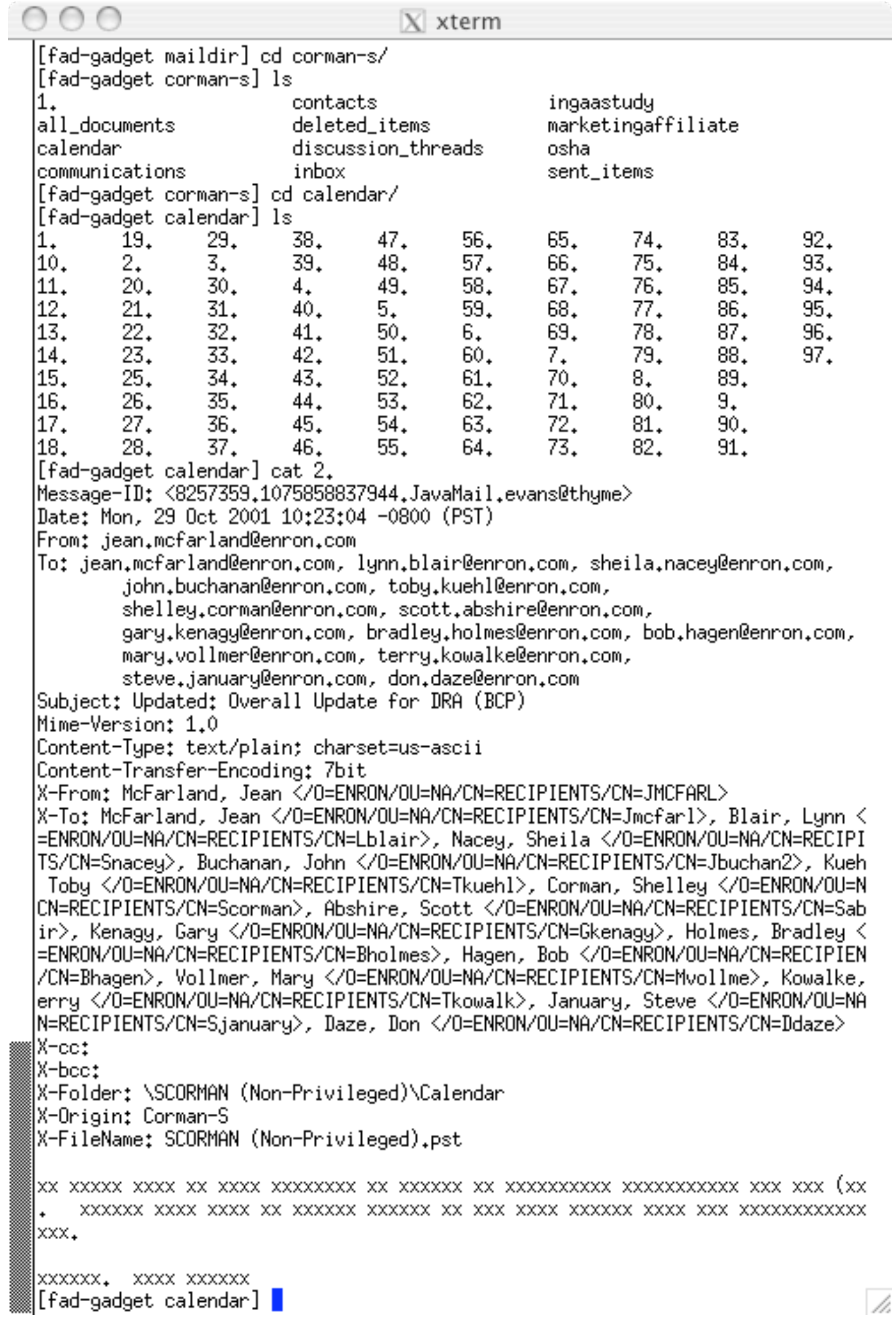
```
[fad-gadget maildir] ls
allen-p         fischer-m       kitchen-l       phanis-s        smith-m
arnold-j        forney-j        kuykendall-t    pimenov-v       solberg-g
arora-h         fossum-d        lavorato-j      platter-p       south-s
badeer-r        gang-l          lay-k           presto-k        staab-t
bailey-s        gay-r           lenhart-m       quenet-j        stclair-c
bass-e          geaccone-t      lewis-a         quigley-d       steffes-j
baughman-d      germany-c       linder-e        rapp-b          stepenovitch-j
beck-s          gilbertsmith-d  lokay-m         reitmeyer-j     stokley-c
benson-r        giron-d         lokey-t         richey-c        storey-g
blair-l         griffith-j      love-p          ring-a          sturm-f
brawner-s       grigsby-m       lucci-p         ring-r          swerzbin-m
buy-r           guzman-m        maggi-m         rodrique-r      symes-k
campbell-l      haedicke-m      mann-k          rogers-b        taylor-m
carson-m        hain-m          martin-t        ruscitti-k      tholt-j
cash-m          harris-s        may-l           sager-e         thomas-p
causholli-m     hayslett-r      mccarty-d       saibi-e         townsend-j
corman-s        heard-m         mcconnell-m     salisbury-h     tycholiz-b
crandell-s      hendrickson-s   mckay-b         sanchez-m       ward-k
cuilla-m        hernandez-j     mckay-j         sanders-r       watson-k
dasovich-j      hodge-j         mclaughlin-e    scholtes-d      weldon-c
davis-d         holst-k         merriss-s       schoolcraft-d   whalley-g
dean-c          horton-s        meyers-a        schwieger-j     whalley-l
delainey-d      hyatt-k         mims-thurston-p scott-s         white-s
derrick-j       hyvl-d          motley-m        semperger-c     whitt-m
dickson-s       jones-t         neal-s          shackleton-s    williams-j
donoho-l        kaminski-v      nemec-g         shankman-j      williams-w3
donohoe-t       kean-s          panus-s         shapiro-r       wolfe-j
dorland-c       keavey-p        parks-j         shively-h       ybarbo-p
ermis-f         keiser-k        pereira-s       skilling-j      zipper-a
farmer-d        king-j          perlingiere-d   slinger-r       zufferli-j
[fad-gadget maildir]
```

An example

Here we select the ex-Vice President for Regulatory Affairs, Shelley Corman

We see the 11 mail folders; selecting the calendar folder, we exhibit the content of mail 2.

Note again, that all textual content has been replaced by x's; we are only interested in (at best) the pattern of communication



```
[fad-gadget maildir] cd corman-s/
[fad-gadget corman-s] ls
1.                     contacts               ingaastudy
all_documents          deleted_items          marketingaffiliate
calendar               discussion_threads     osha
communications         inbox                  sent_items
[fad-gadget corman-s] cd calendar/
[fad-gadget calendar] ls
1.      19.     29.     38.     47.     56.     65.     74.     83.     92.
10.     2.      3.      39.     48.     57.     66.     75.     84.     93.
11.     20.     30.     4.      49.     58.     67.     76.     85.     94.
12.     21.     31.     40.     5.      59.     68.     77.     86.     95.
13.     22.     32.     41.     50.     6.      69.     78.     87.     96.
14.     23.     33.     42.     51.     60.     7.      79.     88.     97.
15.     25.     34.     43.     52.     61.     70.     8.      89.
16.     26.     35.     44.     53.     62.     71.     80.     9.
17.     27.     36.     45.     54.     63.     72.     81.     90.
18.     28.     37.     46.     55.     64.     73.     82.     91.
[fad-gadget calendar] cat 2.
Message-ID: <8257359.1075858837944.JavaMail.evans@thyme>
Date: Mon, 29 Oct 2001 10:23:04 -0800 (PST)
From: jean.mcfarland@enron.com
To: jean.mcfarland@enron.com, lynn.blair@enron.com, sheila.nacey@enron.com,
        john.buchanan@enron.com, toby.kuehl@enron.com,
        shelley.corman@enron.com, scott.abshire@enron.com,
        gary.kenagy@enron.com, bradley.holmes@enron.com, bob.hagen@enron.com,
        mary.vollmer@enron.com, terry.kowalke@enron.com,
        steve.january@enron.com, don.daze@enron.com
Subject: Updated: Overall Update for DRA (BCP)
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-From: McFarland, Jean </O=ENRON/OU=NA/CN=RECIPIENTS/CN=JMCFARL>
X-To: McFarland, Jean </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Jmcfarl>, Blair, Lynn <
=ENRON/OU=NA/CN=RECIPIENTS/CN=Lblair>, Nacey, Sheila </O=ENRON/OU=NA/CN=RECIPI
TS/CN=Snacey>, Buchanan, John </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Jbuchan2>, Kueh
 Toby </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Tkuehl>, Corman, Shelley </O=ENRON/OU=N
CN=RECIPIENTS/CN=Scorman>, Abshire, Scott </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Sab
ir>, Kenagy, Gary </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Gkenagy>, Holmes, Bradley <
=ENRON/OU=NA/CN=RECIPIENTS/CN=Bholmes>, Hagen, Bob </O=ENRON/OU=NA/CN=RECIPIEN
/CN=Bhagen>, Vollmer, Mary </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Mvollme>, Kowalke,
erry </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Tkowalk>, January, Steve </O=ENRON/OU=NA
N=RECIPIENTS/CN=Sjanuary>, Daze, Don </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Ddaze>
X-cc:
X-bcc:
X-Folder: \SCORMAN (Non-Privileged)\Calendar
X-Origin: Corman-S
X-FileName: SCORMAN (Non-Privileged).pst

xx xxxxx xxxx xx xxxx xxxxxxxx xx xxxxxx xx xxxxxxxxxx xxxxxxxxxxx xxx xxx (xx
.   xxxxxx xxxx xxxx xx xxxxxx xxxxxx xx xxx xxxx xxxxxx xxxx xxx xxxxxxxxxxxx
xxx.

xxxxxx.  xxxx xxxxxx
[fad-gadget calendar]
```

Some questions

What is the distribution of numbers of emails per user?

Are the users organizing their email into folders?

Are certain folders common to all users?

What is the distribution of emails per folder?

Hint: One more helpful command

The Unix command `find` traverses a directory tree and returns the files and directories it finds; you can limit the search with various options

For example:

Consider only those email messages numbered `404.`

```
find corman-s -name 404.
```

Consider only those entries that don’t end in a period (.)

```
find corman-s regex ‘[^.]$’
```

```
[fad-gadget maildir] find    corman-s/ -name 404.
corman-s//deleted_items/404.
corman-s//inbox/archives/404.
corman-s//sent_items/404.
[fad-gadget maildir]
[fad-gadget maildir] find    corman-s -regex '.*[^.]$'
corman-s
corman-s/all_documents
corman-s/calendar
corman-s/communications
corman-s/contacts
corman-s/deleted_items
corman-s/discussion_threads
corman-s/inbox
corman-s/inbox/archives
corman-s/inbox/archives/old_messages
corman-s/inbox/archives___post_revised_order
corman-s/inbox/bankruptcy
corman-s/inbox/budget
corman-s/inbox/kidsave
corman-s/inbox/measurement
corman-s/inbox/naesb
corman-s/inbox/oneok
corman-s/inbox/tw_neg_rates
corman-s/inbox/twdatarequests
corman-s/inbox/vacation_schedules
corman-s/ingaastudy
corman-s/marketingaffiliate
corman-s/osha
corman-s/sent_items
[fad-gadget maildir]
```

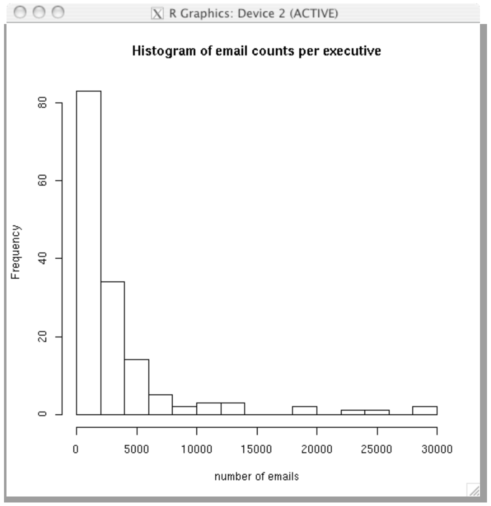
Putting this to work

We can now answer some of the questions about folder usage with calls to `find`, `cut` and `sort`; first, emails per user and folders per user

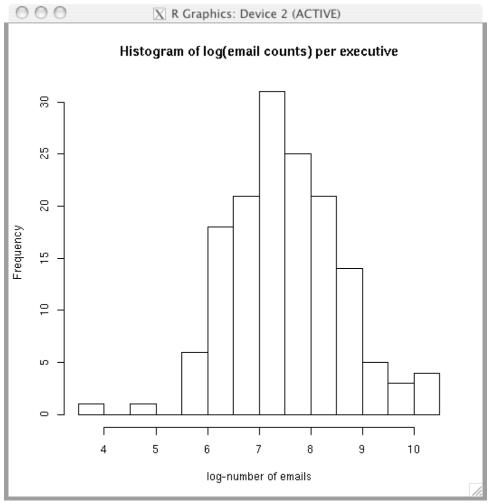
```
[fad-gadget Data] find maildir -type f| cut -d"/" -f2| uniq -c| sort -rn | head
28465 kaminski-v
28234 dasovich-j
25351 kean-s
23381 mann-k
19950 jones-t
18687 shackleton-s
13875 taylor-m
13032 farmer-d
12436 germany-c
11830 beck-s
[fad-gadget Data] find maildir -type d | cut -d"/" -f2 | uniq -c | sort -rn | he
ad
 199 kean-s
 138 beck-s
 117 shapiro-r
 112 shackleton-s
  89 taylor-m
  82 mcconnell-m
  78 griffith-j
  72 germany-c
  70 watson-k
  70 blair-l
[fad-gadget Data]
```

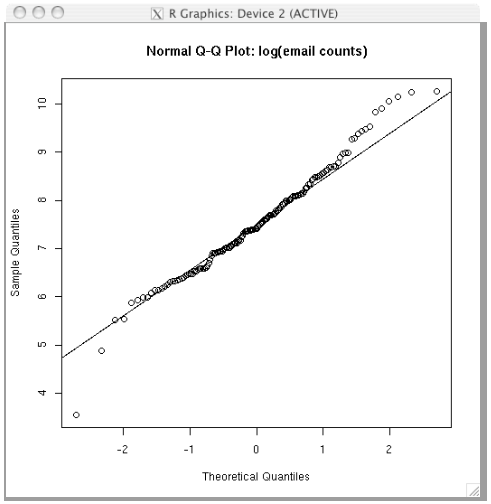
Counts per user

As was the case for hit counts per IP address, we see a very skewed distribution (what Malcolm Gladwell would call a "hockey stick" distribution)



In the bottom figures we present a histogram and a Q-Q plot for the logarithm of the counts





Identifying common folders

The email from 150 executives are included in this file; some folders have similar structures

```
[fad-gadget Data] find maildir -maxdepth 2 -mindepth 2| cut -d"/" -f3 > out.txt
[fad-gadget Data] sort out.txt | uniq -c | sort -rn | head -20
 137 inbox
 136 sent_items
 135 deleted_items
 110 all_documents
  93 discussion_threads
  89 sent
  82 notes_inbox
  78 _sent_mail
  71 calendar
  46 contacts
  39 personal
  35 tasks
  34 to_do
  11 eol
  10 prc
   9 private_folders
   7 presentations
   7 ees
   7 canada
   7 california
```

Looking inside

In a previous version of these slides, we considered a unique message ID tag; instead, let’s consider a time series of the number of emails by day

If we look at the structure of the email *header* we see that a message’s date is kept in a field called `Date:`

xterm

```
Message-ID: <4493790.1075858840249.JavaMail.evans@thyme>
Date: Mon, 29 Oct 2001 09:33:30 -0800 (PST)
From: forrester@forrester.com
To: weekly_research@frstrelay001.forrester.com
Subject: New Research From Forrester -- 10/29/2001
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-From: Forrester Research, Inc. <forrester@FORRESTER.COM>
X-To: WEEKLY_RESEARCH@frstrelay001.forrester.com
X-cc:
X-bcc:
X-Folder: \SCORMAN (Non-Privileged)\Inbox
X-Origin: Corman-S
X-FileName: SCORMAN (Non-Privileged).pst

===========================================================================
xxx xxxxxxxx xx xxx.xxxxxxxxx.xxx -- xx/xx/xxxx
===========================================================================

xxxx xx xxxx xxxxxxxxxx xxx

xxx xxxxxxx xxxxxxxxxx xxxxxxxx xx xxx 'xxx xx xxxx xxx xx xxxxxxx xxxx
xxx xxxxxxx xx xxxxx xxxxxxxxxxx. xxxxx xxxxxx xxx xxxxx xxxx xxxxxxxxx
xxxxx? xxx xxx xxx xxxxx xxxxx? xx xxxxxx xxxxxxx xxxxx xxxx xxxxx?
xxxxxxxxx'x xxxxxxxxxx xxxxxxxxxx xxxxx xxxx xxxxxxx xxxxxxxxx xxx xxxxxxx
xxxxxxxxxxxx xxx xxxx xxx xxxxxx xxxx xxxxxxx'x xxxxxxxxxx xxxxxxxxx.
xxxxxxx xxxxxxxx xxxxxxx:

     xxxxxx x. xxxxx, xxxx xxxxxxxx, xxxxxxxxx, xxx xxx, xxxxxxxx xxx
1. 14%
```

Timing is everything

The dates in Corman's `inbox` folder can be extracted with a simple call to `grep`

```
[fad-gadget inbox] egrep '^Date:\ ' *. | head -20
1.:Date: Mon, 29 Oct 2001 09:33:30 -0800 (PST)
10.:Date: Sun, 28 Oct 2001 07:34:20 -0800 (PST)
11.:Date: Sat, 27 Oct 2001 11:15:21 -0700 (PDT)
12.:Date: Fri, 26 Oct 2001 14:21:09 -0700 (PDT)
13.:Date: Wed, 26 Oct 0001 09:46:23 -0800 (PST)
14.:Date: Fri, 26 Oct 2001 07:53:46 -0700 (PDT)
15.:Date: Thu, 25 Oct 2001 16:32:35 -0700 (PDT)
16.:Date: Tue, 25 Oct 0001 12:43:42 -0800 (PST)
17.:Date: Thu, 25 Oct 2001 08:20:20 -0700 (PDT)
18.:Date: Thu, 25 Oct 2001 08:04:21 -0700 (PDT)
19.:Date: Thu, 25 Oct 2001 06:11:56 -0700 (PDT)
2.:Date: Sat, 29 Oct 0001 21:16:57 -0800 (PST)
20.:Date: Thu, 25 Oct 2001 05:14:08 -0700 (PDT)
21.:Date: Wed, 24 Oct 2001 19:47:56 -0700 (PDT)
22.:Date: Wed, 24 Oct 2001 18:29:59 -0700 (PDT)
23.:Date: Wed, 24 Oct 2001 18:30:48 -0700 (PDT)
24.:Date: Wed, 24 Oct 2001 17:08:15 -0700 (PDT)
25.:Date: Tue, 23 Oct 2001 16:38:07 -0700 (PDT)
26.:Date: Tue, 23 Oct 2001 15:55:15 -0700 (PDT)
27.:Date: Tue, 23 Oct 2001 12:05:06 -0700 (PDT)
[fad-gadget inbox]
```

Shell programs

There are over 150 different directories and it will be hard to extract all the information we are after by hand

Technically, we can use the `find` command to execute a program on each file or directory it encounters*; for the moment, we will ignore this and use date extraction as an application of *shell scripting*

You can collect a series of Unix commands into a shell program; this allows you to repeat commands over different inputs

* The command would look something like

```
find maildir -type f -exec egrep '^Date:' {} ';'
```

Consult the web site below for more information on `find`

`http://www.gnu.org/software/findutils/manual/html_mono/find.html`

Not really a gap, but a good time to split...

From here, I teach a bit about shell programming; I do this because I want the students to see that the commands they've been using can be assembled into programs that can repeat their operations

This will be, of course, a theme in the class; moving from exploratory computing to program-writing; it also lets me talk a bit about permission bits and some trailing filesystem facts

A simple shell program

At the right we have a short program contained in a file `dates.sh`

OK, it isn't much of a program, but it's a reasonably good place to start

The `$1` here refers to the first argument we use to call the program

```
egrep '^Date:' $1
```

Running a shell script

There are two ways to run a shell script; you can either execute it within a new shell (recall that the shell `sh` is just another command)

```
% sh dates.sh
```

This should explain the funny suffix we used for our filename; this kind of naming convention will help you (and others) recognize this file as a shell script (program)

The second way to run this script is to make the file *executable;* that is, it becomes just like any command Unix knows about

Let’s see how this is done; it requires looking a little into how the filesystem specifies *permissions,* who can do what to a file

Permission bits

Unix can support many users on a single system and each user can belong to one or more groups

Every file in a Unix filesystem is owned by some user and one of that user’s groups; each file also has a set of permissions specifying which users can

- r: read
- w: write (modify) or
- x: execute

the file; these are specified with three “bits” and we need three sets of bits to define what the user can do, what their group (that owns the file) can do and what others can do

Shorthand for your present working directory (where you're at)

Shorthand for the directory one level above

An example

```
[fad-gadget marketingaffiliate] ls -al
total 72
drwxr-xr-x   9 cocteau  staff   306 11 Oct 13:33 .
drwxr-xr-x  14 cocteau  staff   476 11 Oct 13:33 ..
-rw-r--r--   1 cocteau  staff  2191 11 Oct 14:14 1.
-rw-r--r--   1 cocteau  staff  4587 11 Oct 14:14 2.
-rw-r--r--   1 cocteau  staff  1061 11 Oct 14:14 3.
-rw-r--r--   1 cocteau  staff  1845 11 Oct 14:14 4.
-rw-r--r--   1 cocteau  staff  2220 11 Oct 14:14 5.
-rw-r--r--   1 cocteau  staff  4163 11 Oct 14:14 6.
-rw-r--r--   1 cocteau  staff  2101 11 Oct 14:14 7.
```

Your user name

The file's name

The group you belong to that owns the file

The file's creation date

The file's size in bytes

An example

The type of file

```
drwxr-xr-x    9 cocteau   staff
drwxr-xr-x   14 cocteau   staff
-rw-r--r--    1 cocteau   staff
-rw-r--r--    1 cocteau   staff
-rw-r--r--    1 cocteau   staff
-rw-r--r--    1 cocteau   staff
```

What you can do to the file

What the owning group can do

What others can do

In general...

The command chmod changes the permissions on a file; here are some examples

```
% chomd g+x dates.sh

% chmod ug-x dates.sh

% chmod a+w dates.sh

% chmod go-w dates.sh
```

You can also use binary to express the permissions; so if we think of bits ordered as rwx, then

$$\begin{array}{l|l} \texttt{r-x} & 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5 \\ \texttt{rwx} & 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 7 \\ \texttt{r--} & 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 4 \end{array}$$

and we can specify permissions with these values

```
% chmod 755 dates.sh
```

A simple shell program

In addition to making our program executable, we need to give the operating some help in figuring out what interpreter to use

That is, we need to tell Unix that the following lines are to be executed in the shell

We start the file with the location of the shell command; if we were working in Python (we'll see this next time), we'd have `usr/bin/python`

```
#!/bin/sh

egrep '^Date:' $1
```

A simple shell program

After that long detour, we could do the following; note that to call the program, we have to tell Unix where to find it

```
[fad-gadget Data] ls
dates.sh        maildir
[fad-gadget Data] chmod +x dates.sh
[fad-gadget Data]
[fad-gadget Data] ls -l
total 8
-rwxr-xr-x    1 cocteau  staff   224 16 Oct 13:28 dates.sh
drwxr-xr-x  152 cocteau  staff  5168 11 Oct 11:19 maildir
[fad-gadget Data]
[fad-gadget Data] cd maildir/corman-s/inbox
[fad-gadget inbox]
[fad-gadget inbox] /Data/dates.sh 2.
Date: Sat, 29 Oct 0001 21:16:57 -0800 (PST)
[fad-gadget inbox]
[fad-gadget inbox] /Data/dates.sh 1.
Date: Mon, 29 Oct 2001 09:33:30 -0800 (PST)
[fad-gadget inbox]
[fad-gadget inbox] /Data/dates.sh 30.
Date: Mon, 22 Oct 2001 14:48:40 -0700 (PDT)
[fad-gadget inbox]
```

A simple shell program

Arguably, we haven't done much in terms of easing our workload

Instead, we could consider looping over all the files in a directory; the slight elaboration of our original program is given at the right

Here we see our basic command to find dates, but it's in the body of a *for loop*

```
#!/bin/sh

for i in `ls`; do

  egrep '^Date:' $i
done
```

For loops

ttp://www.ooblick.com/text/sh/

The basic structure of this construction is given at the right

```
for var in list; do
        commands
done
```

If you have done any programming, this loop will function as you expect; each pass through the loop assigns one value in *`list`* to *`var`*

This is one of several constructions that control the operation or flow of your running program

In our script, the variable `i` takes the output from the command `ls`; note that when we want the value of `i` we use `$i`

```
#!/bin/sh

for i in `ls`; do

  egrep '^Date:' $i
done
```

A simple shell program

In our script, the symbol `i` is a variable; it takes the output from the command `ls`

When we want the value of `i` in our script we refer to `$i`

Variables are used by the shell to remember information; for example, when you start a shell, a number of variables get set by default

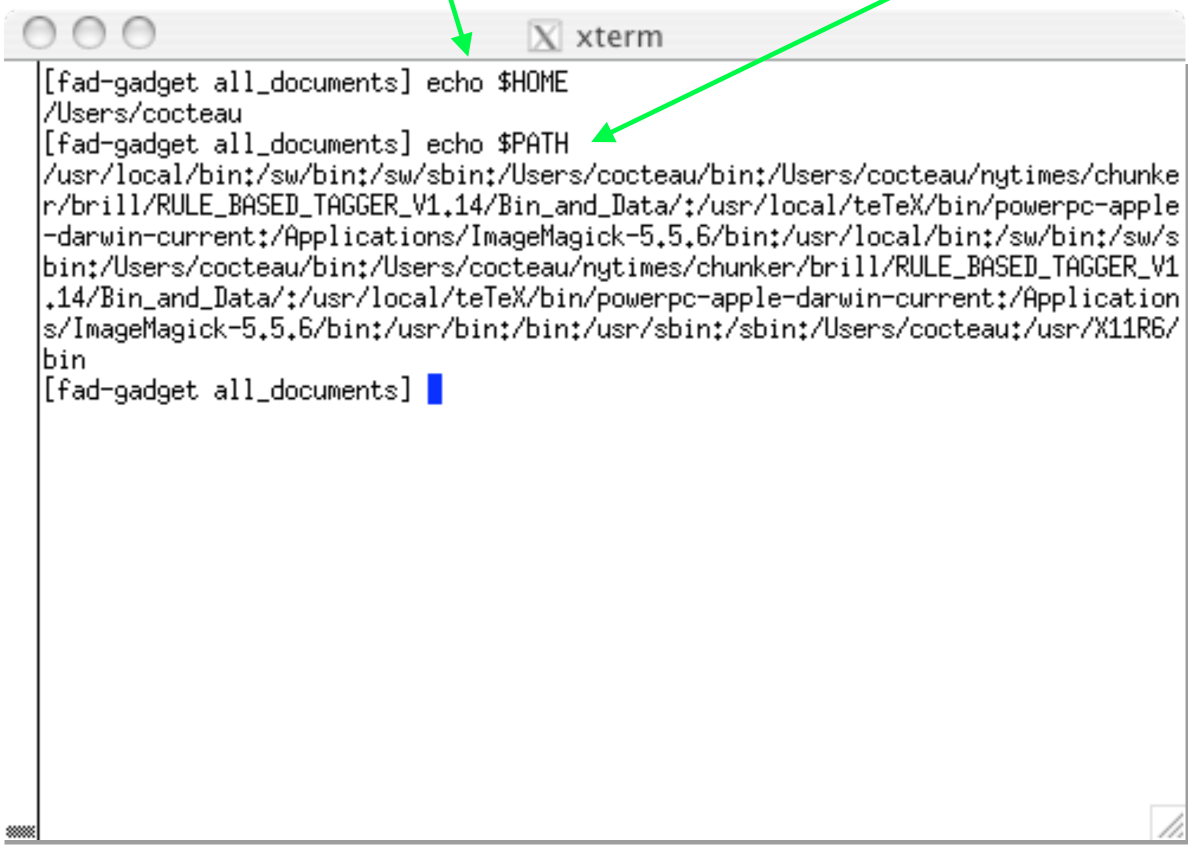
```
#!/bin/sh

for i in `ls`; do

  egrep '^Date:' $i
done
```

Your home directory

The path your shell searches for programs; remember `which`?



A simple shell program

In this short program, we see two different kinds of quotation marks; there are, in fact, three different such constructions

1. `` `command` `` : Backquotes execute the enclosed command and catch the output; here it is assigned in turn to `i`

2. “string“ : Double quotes allow us to slip in special characters that are expanded; so “echo $1” would print the first argument

3. ‘string’ : Single quotes aren’t very fancy; everything inside is as it appears

```
#!/bin/sh

for i in `ls`; do

  egrep '^Date:' $i
done
```

A simple shell program

Running this program generates a single date line for every file in the directory; it scrolls by rather quickly and then...

```
xterm
[fad-gadget inbox] /Data/dates.sh
Date: Mon, 29 Oct 2001 09:33:30 -0800 (PST)
Date: Sun, 28 Oct 2001 07:34:20 -0800 (PST)
Date: Sat, 27 Oct 2001 11:15:21 -0700 (PDT)
Date: Fri, 26 Oct 2001 14:21:09 -0700 (PDT)
Date: Wed, 26 Oct 0001 09:46:23 -0800 (PST)
Date: Fri, 26 Oct 2001 07:53:46 -0700 (PDT)
Date: Thu, 25 Oct 2001 16:32:35 -0700 (PDT)
Date: Tue, 25 Oct 0001 12:43:42 -0800 (PST)
Date: Thu, 25 Oct 2001 08:20:20 -0700 (PDT)
Date: Thu, 25 Oct 2001 08:04:21 -0700 (PDT)
Date: Thu, 25 Oct 2001 06:11:56 -0700 (PDT)
Date: Sat, 29 Oct 0001 21:16:57 -0800 (PST)
Date: Thu, 25 Oct 2001 05:14:08 -0700 (PDT)
Date: Wed, 24 Oct 2001 19:47:56 -0700 (PDT)
Date: Wed, 24 Oct 2001 18:29:59 -0700 (PDT)
Date: Wed, 24 Oct 2001 18:30:48 -0700 (PDT)
Date: Wed, 24 Oct 2001 17:08:15 -0700 (PDT)
Date: Tue, 23 Oct 2001 16:38:07 -0700 (PDT)
Date: Tue, 23 Oct 2001 15:55:15 -0700 (PDT)
Date: Tue, 23 Oct 2001 12:05:06 -0700 (PDT)
Date: Tue, 23 Oct 2001 19:02:55 -0700 (PDT)
```

A simple shell program

...we find a series of errors; what do these mean?

```
xterm
Date: Mon, 25 Mar 2002 13:05:22 -0800 (PST)
Date: Mon, 25 Mar 2002 13:28:31 -0800 (PST)
Date: Mon, 25 Mar 2002 14:09:13 -0800 (PST)
Date: Mon, 25 Mar 2002 15:12:01 -0800 (PST)
Date: Mon, 29 Oct 2001 13:44:53 -0800 (PST)
Date: Mon, 25 Mar 2002 16:03:51 -0800 (PST)
Date: Mon, 25 Mar 2002 17:07:27 -0800 (PST)
Date: Mon, 25 Mar 2002 21:38:28 -0800 (PST)
Date: Mon, 29 Oct 2001 11:08:43 -0800 (PST)
Date: Mon, 29 Oct 2001 06:22:07 -0800 (PST)
egrep: archives: Operation not permitted
egrep: archives___post_revised_order: Operation not permitted
egrep: bankruptcy: Operation not permitted
egrep: budget: Operation not permitted
egrep: kidsave: Operation not permitted
egrep: measurement: Operation not permitted
egrep: naesb: Operation not permitted
egrep: oneok: Operation not permitted
egrep: tw_neg_rates: Operation not permitted
egrep: twdatarequests: Operation not permitted
egrep: vacation_schedules: Operation not permitted
[fad-gadget inbox]
[fad-gadget inbox]
```

ttp://www.ooblick.com/text/sh/

Conditional execution

```
if condition ; then
     commands
[elif condition; then
      commands]
[else
      commands]
fi
```

The problem is that we cannot call egrep on a directory

Therefore, we'd like to assess what kind of file we are dealing with, and only execute the command where we should

Unix provides a conditional evaluation utility called `test`; in addition to performing simple numerical comparisons, it also provides facilities for interrogating files

Here the flag `-f` returns true if `$i` is a regular file

```
#!/bin/sh

for i in `ls`; do

  if test -f $i; then

     egrep '^Date:' $i
  fi
done
```

ttp://www.ooblick.com/text/sh/

Conditional execution

Often, rather than explicitly using the `test` function, programmers will use a shorthand construction

The `[ ]`'s are an implicit call to `test`; to be precise, there's a command called `/bin/[`

```
if condition ; then
     commands
[elif condition; then
      commands]
[else
      commands]
fi
```

```
#!/bin/sh

for i in `ls`; do

  if [-f $i ]; then

     egrep '^Date:' $i
  fi
done
```

Finally...

So far, all we've done is execute a `egrep` command in the directory where we call our program

The commands at the right recurse through the directory provided as an argument to the program

```
#!/bin/sh

for i in `find $1`; do

    if [ -f $i ]; then

        egrep '^Date:' $i
    fi
done
```

A simple shell program

Now we can call the program from any directory; here we call it from `/Data` (note the `./` telling Unix where to find the file)

```
[fad-gadget Data] ./dates.sh maildir/corman-s/inbox
Date: Mon, 29 Oct 2001 09:33:30 -0800 (PST)
Date: Sun, 28 Oct 2001 07:34:20 -0800 (PST)
Date: Sat, 27 Oct 2001 11:15:21 -0700 (PDT)
Date: Fri, 26 Oct 2001 14:21:09 -0700 (PDT)
Date: Wed, 26 Oct 0001 09:46:23 -0800 (PST)
Date: Fri, 26 Oct 2001 07:53:46 -0700 (PDT)
Date: Thu, 25 Oct 2001 16:32:35 -0700 (PDT)
Date: Tue, 25 Oct 0001 12:43:42 -0800 (PST)
Date: Thu, 25 Oct 2001 08:20:20 -0700 (PDT)
Date: Thu, 25 Oct 2001 08:04:21 -0700 (PDT)
Date: Thu, 25 Oct 2001 06:11:56 -0700 (PDT)
Date: Sat, 29 Oct 0001 21:16:57 -0800 (PST)
Date: Thu, 25 Oct 2001 05:14:08 -0700 (PDT)
Date: Wed, 24 Oct 2001 19:47:56 -0700 (PDT)
Date: Wed, 24 Oct 2001 18:29:59 -0700 (PDT)
Date: Wed, 24 Oct 2001 18:30:48 -0700 (PDT)
Date: Wed, 24 Oct 2001 17:08:15 -0700 (PDT)
Date: Tue, 23 Oct 2001 16:38:07 -0700 (PDT)
Date: Tue, 23 Oct 2001 15:55:15 -0700 (PDT)
Date: Tue, 23 Oct 2001 12:05:06 -0700 (PDT)
Date: Tue, 23 Oct 2001 19:02:55 -0700 (PDT)
Date: Tue, 23 Oct 2001 10:02:55 -0700 (PDT)
Date: Mon, 29 Oct 2001 18:59:07 -0800 (PST)
```

Overview

ttp://www.ooblick.com/text/sh/

My goal here is to have you realize that the commands we have been working with can be collected into programs or scripts; along the way, we learned something about file permissions

The scripting facilities in Unix let you do standard things like loop over values, execute commands conditionally, and so on

There are other structures like *while loops* and *case statements* that further redirect the control of your program

```
while condition; do
    commands
done
```

```
case expression in

   pattern)
      commands
   ;;

esac
```

Overview

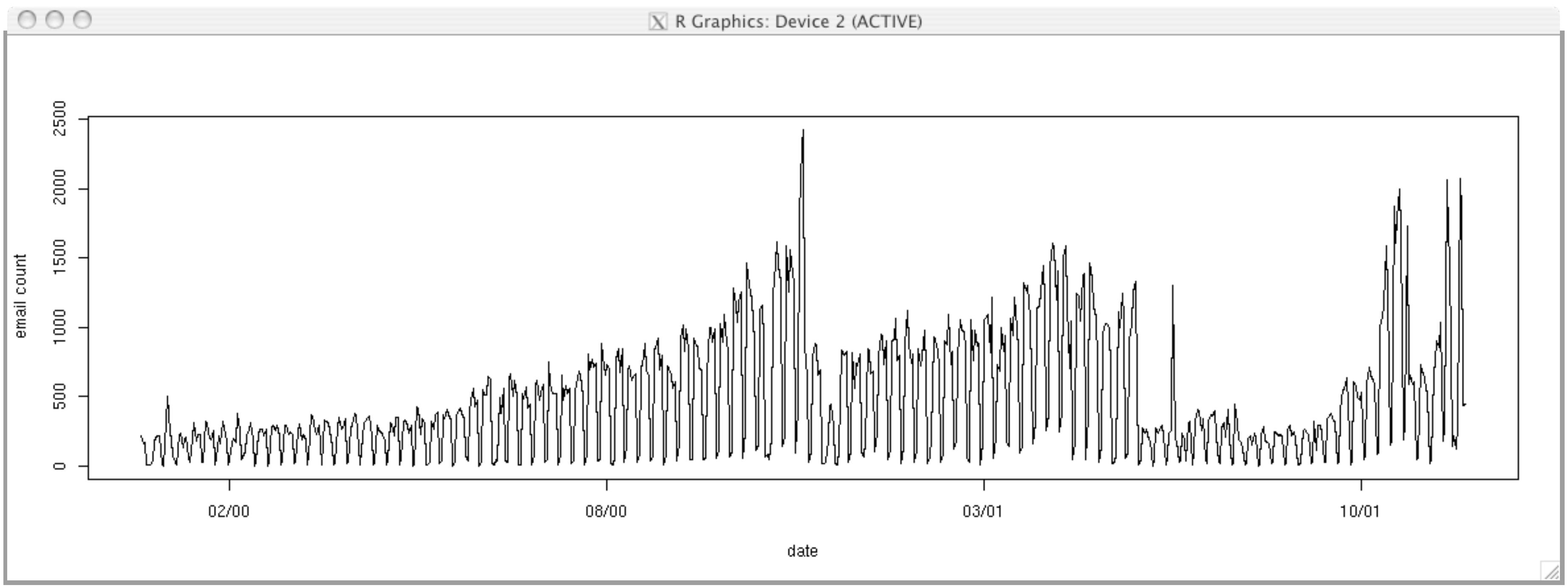
Again, much of the recursion over directories can be done with the `find` command

This presentation was meant to walk you through the structure of a shell script, illustrating what programming tools are at your disposal

This material will come up again as we consider distributing code, say through an R package

Oh, and what were we after?

Here we have a plot of the total number of emails recorded during this period; what patterns do we see?



The gist

I teach the shell for both practical as well as pedagogical reasons; it is structurally similar to other "shells" they will encounter (Python, R), but the vocabulary is fairly limited

With a few commands they can do some pretty powerful analysis, whether or not you want to call that statistics is another question, and you can quickly motivate the need for making programs...

What do y'all think?