JOHNS HOPKINS UNIVERSITY

Zanvyl Krieger School of Arts and Sciences
Advanced Academic Programs

410.734.81
Practical Introduction to Metagenomics

Topic: Introduction to Unix/Linux

Instructor: Joshua Orvis

Why do I need to know this?

Some of you are taking this course because you want to be on the computational side of bioinformatics and, if so, Unix/Linux is the *de facto* primary operating system for the field.

If you see your career following more the role of the biological analyst and not a programmer, you'll most likely still need to learn this. Just as you expect to work with programmers who have some idea about the biology of a project, they'll expect you to have some clue about their side of the work.

Throughout my career it has been my experience that the best analysts are those who can do at least rudimentary shell/Perl/Python coding and it's becoming more common that analysts positions require it.

For this specific class, all work will be performed on the DIAG compute cluster and to keep up you'll need to know these basics.

UNIX introduction

If you're attending this course it is presumed you have already taken 410.634 or have equivalent experience.

While there are many tools available on all platforms, bioinformatics is primarily a UNIX-based field, so it is important for your careers that you become an expert in using at least one UNIX-based operating system.

Aside from code editing, nearly all of this class will consist of performing operations using the Unix/Linux command line, so you **must** get comfortable using this.

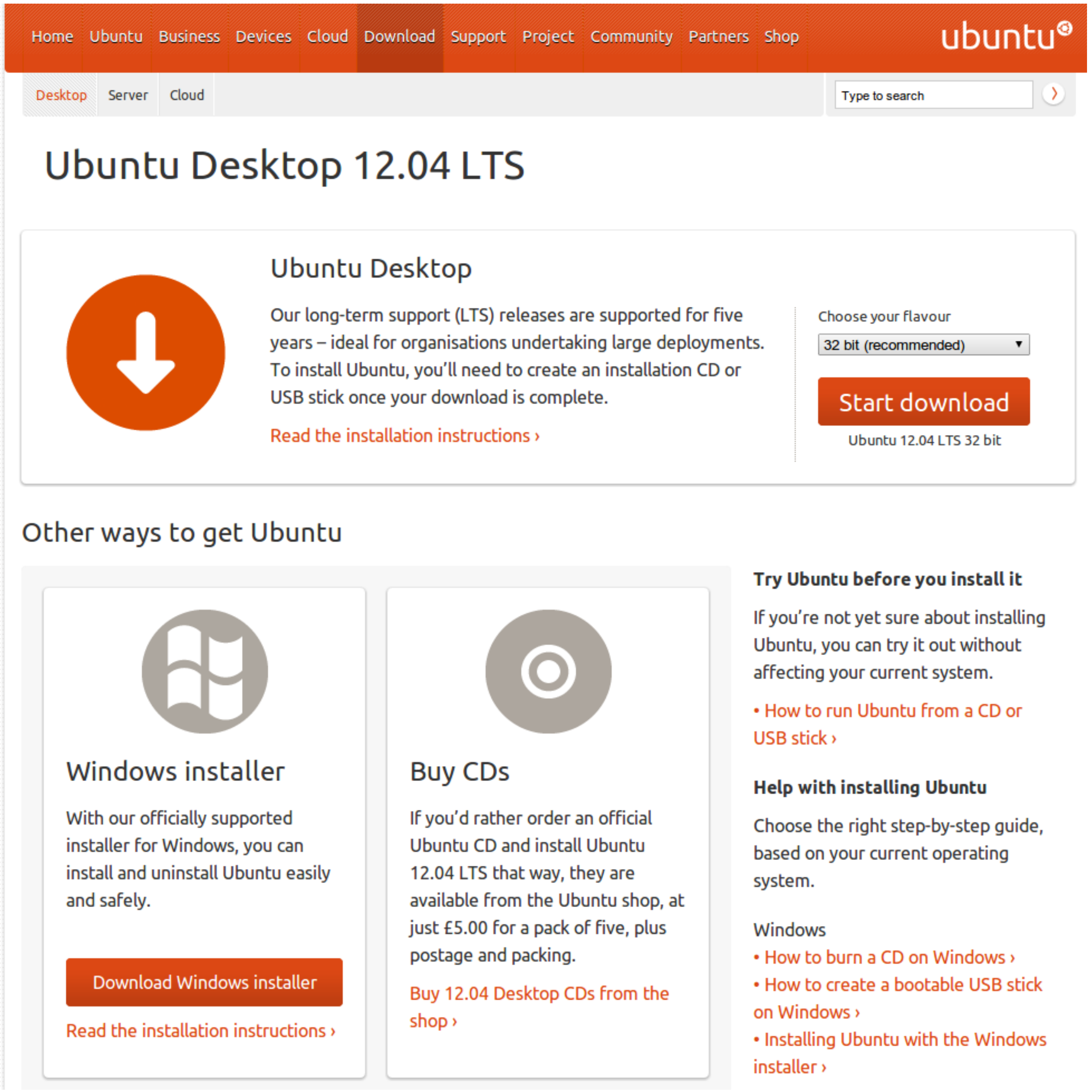
If you are not already using Linux, Mac OS X or another UNIX-based operating system it is assumed that you will follow the “Running Ubuntu” guide on the next slide to either install Linux on your computer or boot from a LiveCD for a temporary working environment.

Getting Linux

If you're using Windows and want to try Linux you can do so and keep your Windows installation at the same time.

If you use the "Download Windows Installer" option near the bottom-left, it will install Windows and Linux apart from each other, and you'll have a menu to choose which one you want to boot into each time you turn your computer on.

http://www.ubuntu.com/download/desktop



Ubuntu Desktop 12.04 LTS

Ubuntu Desktop

Our long-term support (LTS) releases are supported for five years – ideal for organisations undertaking large deployments. To install Ubuntu, you'll need to create an installation CD or USB stick once your download is complete.

Read the installation instructions ›

Choose your flavour: 32 bit (recommended)

Start download

Ubuntu 12.04 LTS 32 bit

Other ways to get Ubuntu

Windows installer

With our officially supported installer for Windows, you can install and uninstall Ubuntu easily and safely.

Download Windows installer

Read the installation instructions ›

Buy CDs

If you'd rather order an official Ubuntu CD and install Ubuntu 12.04 LTS that way, they are available from the Ubuntu shop, at just £5.00 for a pack of five, plus postage and packing.

Buy 12.04 Desktop CDs from the shop ›

Try Ubuntu before you install it

If you're not yet sure about installing Ubuntu, you can try it out without affecting your current system.

• How to run Ubuntu from a CD or USB stick ›

Help with installing Ubuntu

Choose the right step-by-step guide, based on your current operating system.

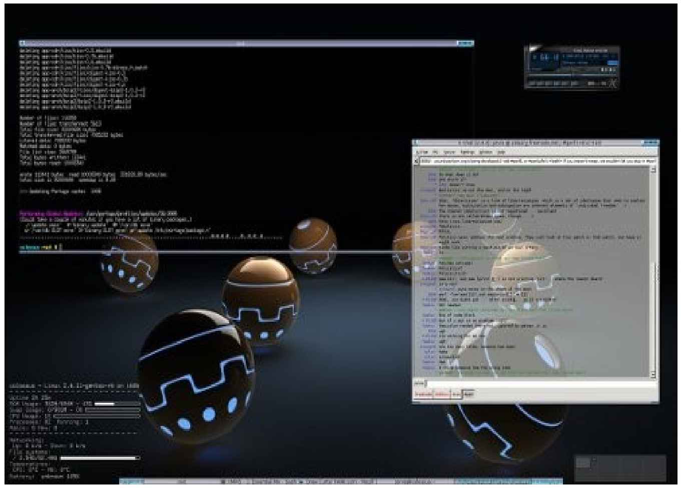
Windows
• How to burn a CD on Windows ›
• How to create a bootable USB stick on Windows ›
• Installing Ubuntu with the Windows installer ›

What is UNIX?

"*Unix is simple. It just takes a genius to understand its simplicity.*" – Dennis Ritchie

Historically, the term "UNIX" evokes images of green/black monochrome screens with a blinking cursor, ready to accept commands from a bearded fat man on a clicky tan keyboard.

Modern UNIX-based operating systems, such as Ubuntu Linux, have very appealing, highly-customizable interfaces that can be used in a familiar point-and-click way but still provide terminal/shell access that is far more powerful for some applications.



What is UNIX?

"*UNIX was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things.*" – Dennis Ritchie

In modern usage, when we say 'UNIX' we are referring to a wide family of operating systems that include BSD, OS X, and numerous distributions Linux.

Specifically, they are mostly all portable, multi-user operating systems with hierarchical directory structures, text-based configuration files, access to command-line terminals and descendent from "Unics", developed by AT&T Bell Labs in the 1970s.

In this class we'll focus on Linux, and specifically our examples will use a *distribution* called Ubuntu Linux. There are hundreds of competing distributions, but this is one of the most popular and is easy to get started with. While I might often say just "Linux" during these lecture slides, you should consider most of this to be true with any UNIX variant.

Open-source software

The Linux *kernel* (the core part of the operating system) is open-source which, at its purest, just means that you have the ability to download and view the source code for it. In practice this is also true for nearly all of the other programs within a given Linux distribution, but not always. Often this is also referred to as free software, but “free” is more difficult to define than you might suspect:

Free as in Beer – First coined by software activist Richard Stallman, meaning that something doesn't cost any money. (I don't know what bars he goes to.)

Free as in Speech – This sort of software is free in the sense of having no restrictions. You can download it, change it, redistribute it, etc.

Free as in Pretzels – Free to acquire, and it may whet the appetite, but you have to pay to get additional, often ultimately necessary, functionality. iPhone users should be familiar with this.

Open-source software - licenses

Corporate lawyers for companies who use (and write) open-source software do not run around courtrooms talking about pretzels and beer. There are a host of specific open-source licenses to choose from when you want to make your project open-source, saving you the hassle of legal document preparation. The Open-Source Initiative (OSI) provides these along with guides for applying them to your project.

www.opensource.org

Choose your license carefully, as they all specify different combinations of rules governing whether users can do things like redistribute, modify or sell your code. Popular examples include:

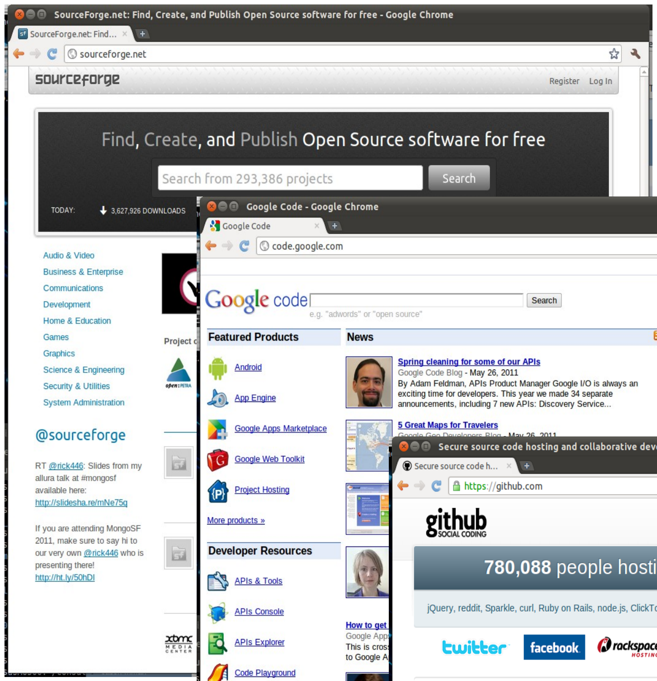
GNU General Public License – popular but “viral” license that requires any programs that link or use GPL code to also be released under the same license.

Artistic License – Written by Larry Wall, the author of Perl, this license has been successfully tested in court.

BSD License – Relatively open and permissive set of licenses that are approved by the Free Software Foundation.

Finding open-source software

There are several websites dedicated to hosting your code and providing it to others. Each of these examples provides free source code management access as well as project organization tools, bug trackers, mailing lists, etc. for your user community.



SourceForge.net: Find, Create, and Publish Open Source software for free - Google Chrome

sourceforge.net

sourceforge Register Log In

Find, Create, and Publish Open Source software for free

Search from 293,386 projects Search

TODAY: 3,627,926 DOWNLOADS

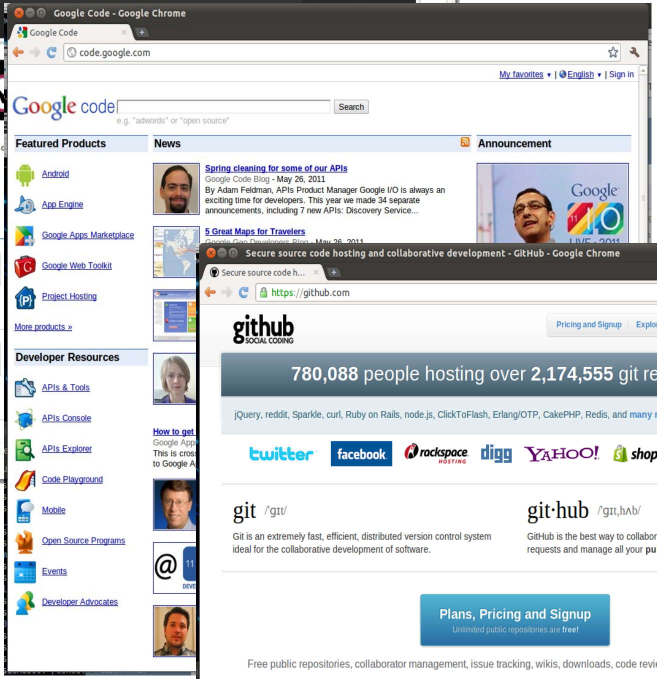
Audio & Video
Business & Enterprise
Communications
Development
Home & Education
Games
Graphics
Science & Engineering
Security & Utilities
System Administration

@sourceforge

RT @rick446: Slides from my allura talk at #mongosf available here: http://slidesha.re/mNe75q

If you are attending MongoSF 2011, make sure to say hi to our very own @rick446 who is presenting there! http://ht.ly/50hDI

SourceForge (sf.net): SVN and CVS code management. Historically the most popular, though losing ground to newer resources.



Google Code - Google Chrome

code.google.com

My favorites ▾ | English ▾ | Sign in

Google code Search
e.g. "adwords" or "open source"

Featured Products

Android
App Engine
Google Apps Marketplace
Google Web Toolkit
Project Hosting
More products »

Developer Resources

APIs & Tools
APIs Console
APIs Explorer
Code Playground
Mobile
Open Source Programs
Events
Developer Advocates

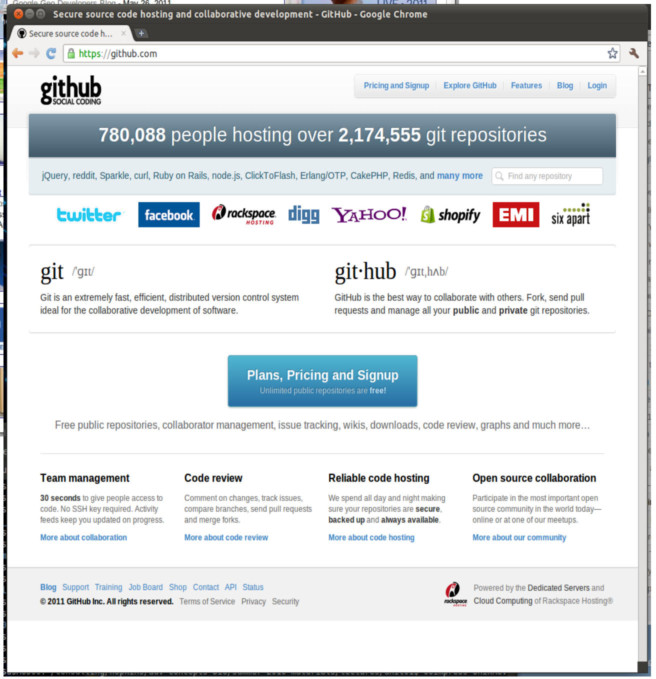
News

Spring cleaning for some of our APIs
Google Code Blog - May 26, 2011
By Adam Feldman, APIs Product Manager Google I/O is always an exciting time for developers. This year we made 34 separate announcements, including 7 new APIs: Discovery Service...

5 Great Maps for Travelers

Announcement

Google Code (code.google.com): SVN or Mercurial repositories. Offering from Google – A distant 3rd place when compared with these other two, but is well integrated into Google's other services.



Secure source code hosting and collaborative development - GitHub - Google Chrome

https://github.com

github SOCIAL CODING

Pricing and Signup | Explore GitHub | Features | Blog | Login

780,088 people hosting over **2,174,555** git repositories

jQuery, reddit, Sparkle, curl, Ruby on Rails, node.js, ClickToFlash, Erlang/OTP, CakePHP, Redis, and many more

Find any repository

git /'gɪt/
Git is an extremely fast, efficient, distributed version control system ideal for the collaborative development of software.

git·hub /'gɪt,hʌb/
GitHub is the best way to collaborate with others. Fork, send pull requests and manage all your **public** and **private** git repositories.

Plans, Pricing and Signup
Unlimited public repositories are **free!**

Free public repositories, collaborator management, issue tracking, wikis, downloads, code review, graphs and much more...

Team management
30 seconds to give people access to code. No SSH key required. Activity feeds keep you updated on progress.
More about collaboration

Code review
Comment on changes, track issues, compare branches, send pull requests and merge forks.
More about code review

Reliable code hosting
We spend all day and night making sure your repositories are **secure**, **backed up** and **always available**.
More about code hosting

Open source collaboration
Participate in the most important open source community in the world today—online or at one of our meetups.
More about our community

Blog Support Training Job Board Shop Contact API Status
© 2011 GitHub Inc. All rights reserved. Terms of Service Privacy Security
Powered by the Dedicated Servers and Cloud Computing of Rackspace Hosting®

GitHub (www.github.com): Git repositories only. The fastest rising and currently most popular of these, GitHub uses a distributed code management system called 'Git' that encourages software forks and merges. This is the home of projects like Rails and JQuery.

Opening a terminal

The terminal is your primary method for accessing your operating system and telling it what to do. Embrace it. Love it. Learn it. As bioinformatics developers you will learn that it is an indispensable tool for getting things done quickly.

To open a terminal in Ubuntu hit “CTRL + Alt + t” or click Applications → Accessories → Terminal

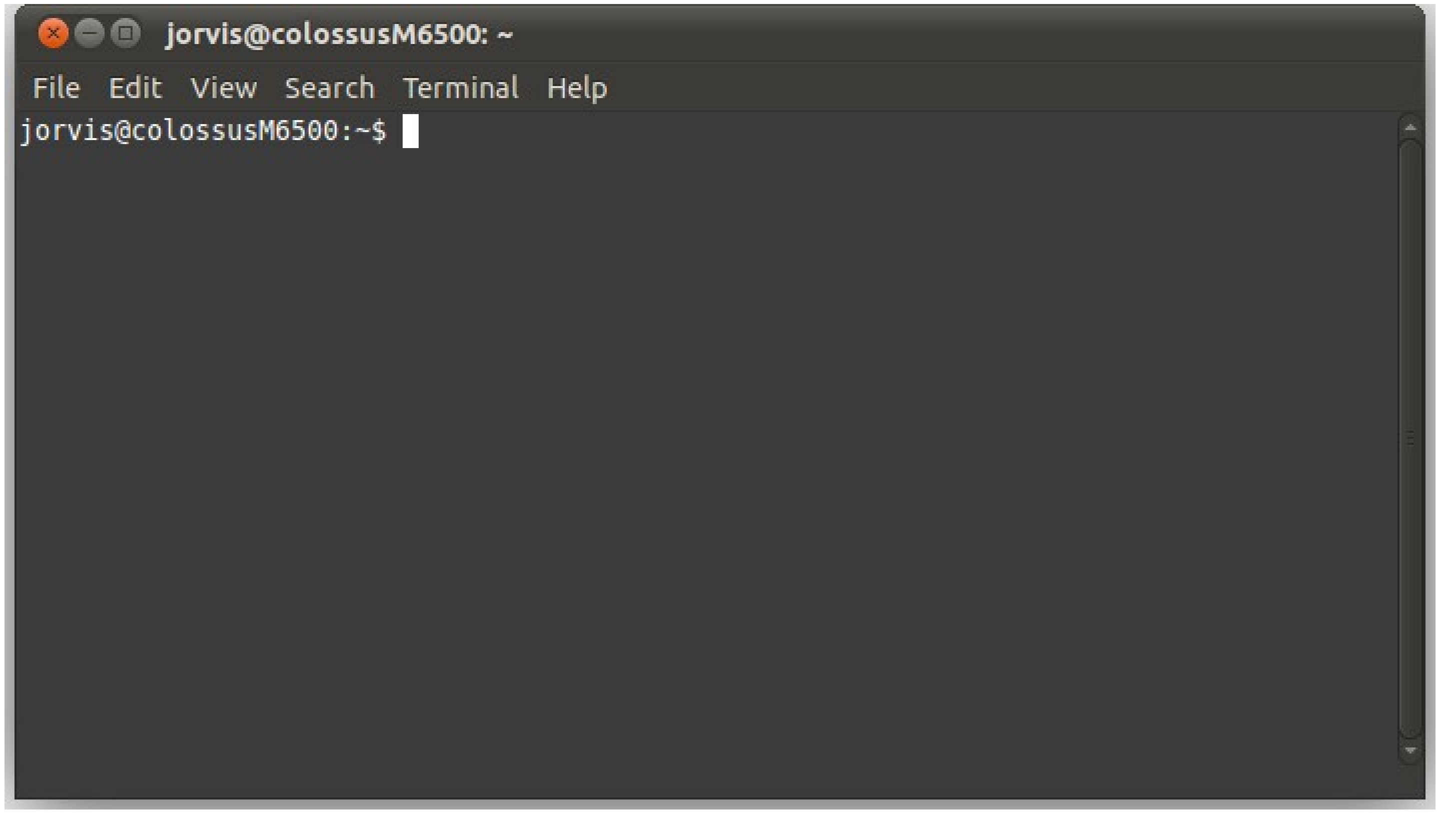
On a Mac open the Finder and search for Terminal.

This is a called a terminal *prompt*. It usually has this format:

user@*host*:*path*$

user: who you are logged in as
host: computer's name
path: position in the filesystem
$: symbol of the *shell* in use

(Why is my host called colossusM6500? Because it's a 17” behemoth of a laptop and is nearly unusable on planes. The M6500 is just the model #)



```
jorvis@colossusM6500:~$
```

Using the terminal - Shells

When using a terminal you are issuing commands within a *shell* and, naturally, there are a variety of different shells to choose from. Each has slightly different syntax when you really get into the details of them, but the general commands we'll use in class will be the same.

The default shell on Ubuntu and recent versions of OS X is *bash*, which stands for Bourne-again shell. Here are some example commands so you can get a feel for the syntax:

```
$ pwd                                  (command only)
$ ls -l                                (command with an option)
$ head -n 5 class.notes                (command with an option with an argument (5) and file)
$ ssh -l jsmith -XY someserver.com     (command with multiple options)
```

We'll cover these commands in detail on upcoming slides, so just focus on the elements of each command here. (Also remember that the '$' symbol just shows the start of a command for our bash shell, and you don't type it.)

In each of these examples the *command* is first. The first ('pwd') doesn't have any *arguments* or *options* passed to it. The second has a '-l' option alone. The next has an option '-n' that requires an argument (5) and a file to operate on. The ssh command has multiple options and arguments.

Let's learn how to log in to a server next and then see what these and other commands actually do.

Logging in and first commands

The **ssh** command allows you to remotely log into other servers, such as the one set up for this class. You must pass it a server name or IP address. (You'll need to replace the IP address here with the one you were sent for this class.)

```
$ ssh -l jsmith 192.168.1.200
```

You will be prompted for a password (again, use the one you were given) and, once successful, will find a shell waiting to accept your commands. Some initial commands to try from here are:

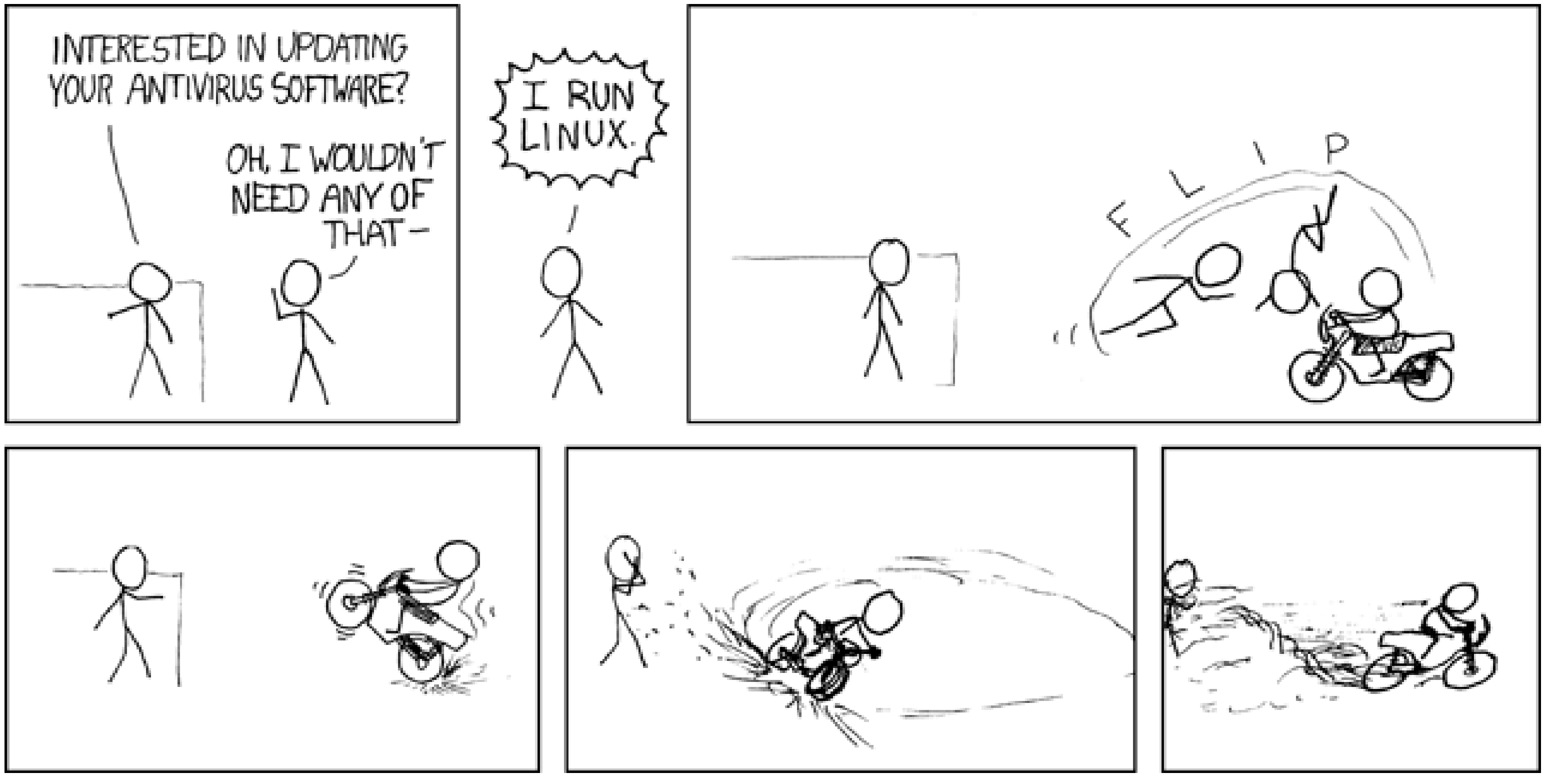
```
$ whoami                    (returns the name of your logged-in user)
$ hostname                  (gives the name of your server)
$ pwd                       (prints your working directory)
$ ls -l                     (lists the files in your working directory, with details)
```

This set of simple commands can help orient you when you're on a new server. The next thing you should do if this is your first time is change your password with *passwd* :

```
$ passwd
```

You will be prompted to enter your current password before entering a new one. You can change this as often as you like. When you are finished with a terminal type 'exit'.

“Linux user at Best Buy”


INTERESTED IN UPDATING YOUR ANTIVIRUS SOFTWARE?
OH, I WOULDN'T NEED ANY OF THAT–
I RUN LINUX.
FLIP

xkcd.com/272/

(There is no point to this at all except to break up slides with lots of text. If you are anti-humor, carry on to the next slide.)

Directory structure and navigation

Most of you are already familiar with the hierarchical way in which files and directories are stored in most operating systems. (Note that 'directories' and 'folders' are generally interchangeable terms.) You can create files and folders, and place them all inside any other folders, creating a tree-like structure. In UNIX file systems the beginning of this tree is called the *root* and has a very simple path:

```
/
```

Most UNIX file systems have several directories here. I'll highlight some important Ubuntu ones for this class that will show the multi-level structure of the hierarchy:

/home/jsmith/	(each user has a home area)
/opt/	(where custom-installed software will go)
/tmp/	(anyone can write temporary files here)
/var/lib/mysql/	(MySQL data file locations)
/var/log/apache2/	(apache writes error and access logs here)
/var/www/	(web application files will go here)

Windows users will notice that there are no drive letters (C:, D:, etc.) In UNIX there is a single directory hierarchy and other drives can be *mounted* to show up as a directory in almost any location.

Directory structure and navigation

When moving around the file system (command: *cd*) you need to know where you are (*pwd*) and see what's there (*ls*). Here are some examples:

```
$ pwd
/home/jsmith

$ ls
bin  docs  lib   music   notes   resume.pdf

$ cd docs                          (note: I could have also typed the full path of /home/jsmith/docs)

$ pwd
/home/jsmith/docs

$ cd ..

$ pwd
/home/jsmith
```

In this series of commands we check our location, see what's there, change directories into the 'docs' folder, show our path, cd back 'up' one directory (using '..'), then show our position again.

“Stand back!”

store.xkcd.com/xkcd/#StandBackScience

Creating/removing files and folders

When you log in you'll be placed in your own area called your *home directory*.

```
$ pwd
/home/jsmith
```

Your home area is your own personal work space to create, upload and organize files and directories in any way you like. The quickest way to create a file is with the *touch* command:

```
$ touch test.file
```

This creates a file called 'test.file' that is initially empty. We'll learn how to populate it soon. You can create a directory like this:

```
$ mkdir notes
```

This creates a directory called 'notes' that you could place other files in. The 'rm' command is used to remove files and directories, though it requires special options when removing a directory as slight protection from accidentally removing more than you intended to. You can even *pass* multiple arguments to rm if you want to delete multiple things:

```
$ rm -rf test.file notes
```

The '-rf' options tell rm to remove the 'notes' directory recursively and forcefully, meaning that it will also remove the contents of 'notes' without validation prompts. Notice that there are no warnings when deleting files you own, so take care when using 'rm'.

Copying and moving files

Two simple commands are used to copy (cp) or move (mv) files on the UNIX file system. The mv command is also used to rename files. Both commands have the same basic syntax:

```
$ command   source    destination
```

For example, to make a backup copy of a file you might do this:

```
$ cp journal.doc journal_backup.doc
```

If this journal file was instead no longer current and you wanted to rename it, you could do:

```
$ mv journal.doc journal.doc.old
```

If you want to copy a directory instead of a single file you have to pass an option to the cp command to tell it to copy *recursively*. This would copy my docs directory into the temp area:

```
$ cp -r /home/jorvis/docs/ /tmp/
```

Be warned that both commands will overwrite any existing files with the destination you specify without warning first. This makes it very easy to lose data if you aren't careful!

Permissions

You can pass an option to the *ls* command to get a 'long listing' of files and directories that provides a lot more information, including the permissions allowed for each.

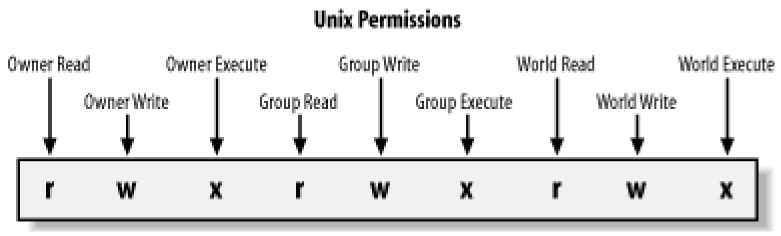
```
$ ls -l
drwxr-xr-x 2 jorvis adm  4096 2011-05-30 20:56 bin
drwxr-x--- 2 jorvis adm  4096 2011-05-30 20:56 docs
drwxr-xr-x 2 jorvis adm  4096 2011-05-30 20:56 lib
drwxr-xr-x 2 jorvis adm  4096 2011-05-30 20:56 music
drwxr-xr-x 2 jorvis adm  4096 2011-05-30 20:56 notes
-rw-r--r-- 1 jorvis adm 62656 2011-05-30 21:45 resume.pdf
drwxrwx--- 2 jorvis adm  4096 2011-05-30 21:45 share
```

There is a lot of useful information here. The columns returned are:

1. Object permission block (see next slide for detailed info here)
2. Number of links to the object (not commonly used)
3. Object user owner
4. Object group owner
5. Object size
6. Last modified date
7. Last modified time
8. Object name

Permissions

Let's return to the first column in the 'ls -l' output on the previous slide. It is a concise shorthand – the first character tells you what type of object it is (file, directory, symlink, etc.) After that are three triplet characters that describe who should be able to read, write and execute that object. For directories, 'execute' specifies whether the user can cd into it. See the graphic below:



Looking back on our previous slide, we see that our 'docs' directory has 'drwxr-x---'. This means the user can do anything to the file, group members can only read or execute, and the rest of the world can't do anything at all.

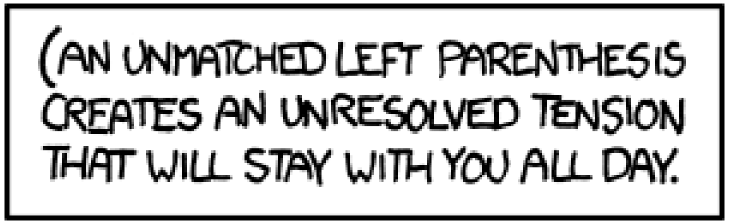
This shorthand can (and most commonly is) shortened even more using octal notation. For each triplet r=4, w=2, and x=1. These values are summed within each triplet then concatenated with the other two. For our 'docs' directory the triplets are:

```
User   rwx = 4+2+1 = 7
Group  r-x = 4+0+1 = 5      == 750
World  --- = 0+0+0 = 0
```

Using this notation you can set all permissions on any object with a single chmod command:

```
$ chmod 755 docs
```
(opens permissions to give world readability)

(


(AN UNMATCHED LEFT PARENTHESIS CREATES AN UNRESOLVED TENSION THAT WILL STAY WITH YOU ALL DAY.

xkcd.com/859

Transferring files

You will often need to transfer files between computers, either downloading data from a server or uploading code/data from your workstation to a server. UNIX provides many ways to do this, but two simple ones you can use throughout this class are:

wget – If you have the URL for a file you would like to download, such as a large DNA database, this is an appropriate command-line choice. It has options to do much more, such as full site mirroring, but here is an example of downloading a single file (please don't do this on the class server).

```
$ wget ftp://ftp.ncbi.nih.gov/blast/db/FASTA/nr.gz        (get the non-redundant protein db)
```

scp – The secure copy tool provides a way to transfer files between computers using syntax similar to native UNIX 'cp' commands. The main difference is that the source or destination can contain the user and host information of a remote server. For example, to transfer a file from your home directory locally to your home directory on a remote server you can do this:

```
$ scp resume.pdf jsmith@192.168.1.200:/home/jsmith/docs/
```

This is the best way to transfer files you work on to the class server, so you should practice this.

Reading file contents

There are several ways to read the contents of a file using the terminal and others that invoke a graphical code editor, which are discussed on the next slide. Using the terminal, the most common way to inspect the contents of a file is to use *less*:

```
$ less dna.fna
```

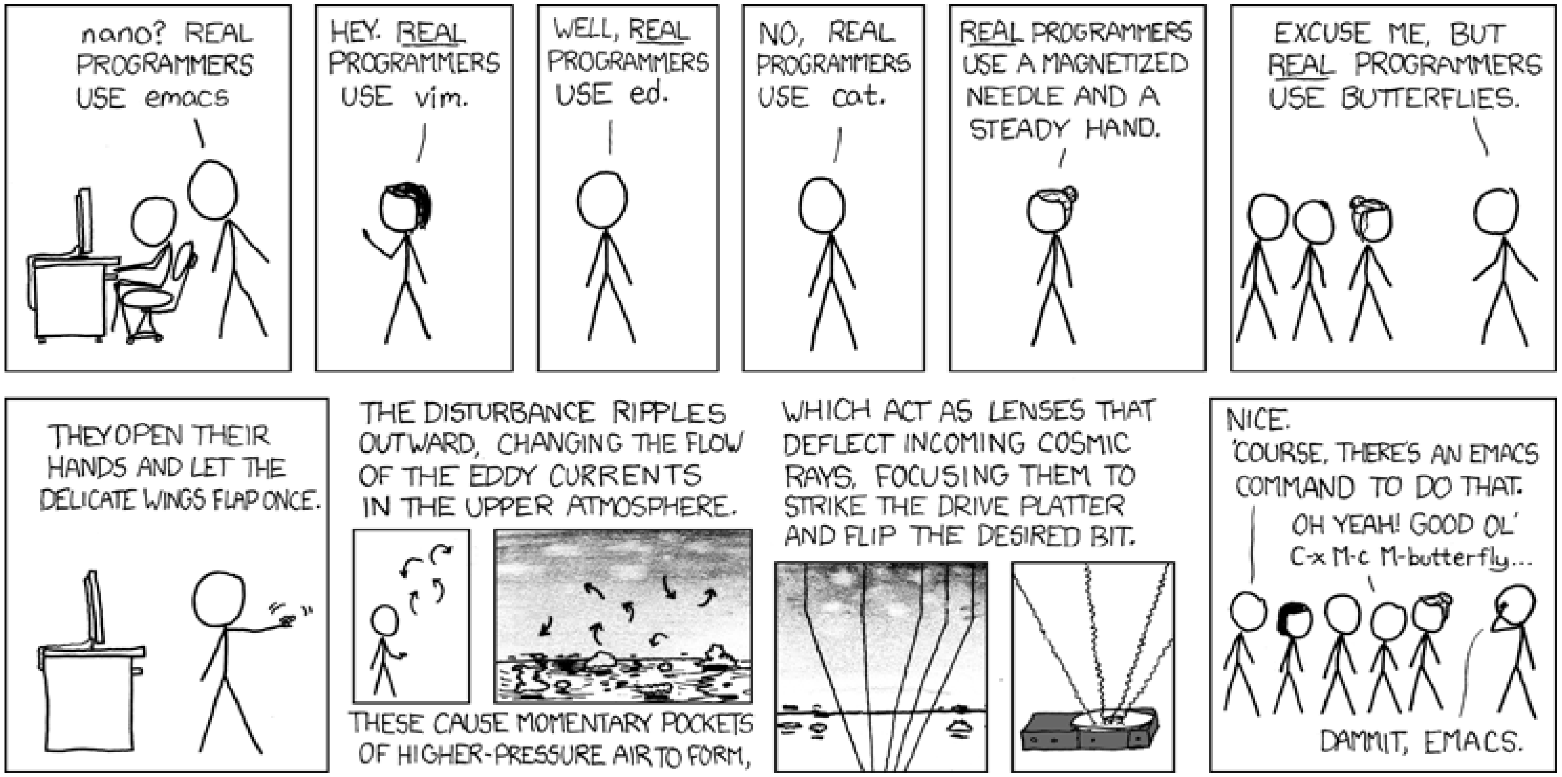
The less program reads the size of your terminal and displays one 'page' at a time. You can use the space bar to scroll down or your keyboard arrows to go either direction. If you want to search the file you can type '/', the string you want to find, then enter to search. Finally, to quit just press 'q'.

If you know you only want to see the first or last number of lines in a file, the *head* and *tail* commands are useful.

```
$ head -n 10 dna.fna          (shows the first 10 lines of the dna.fna file)
$ tail -n 20 dna.fna          (shows the last 20 lines of the dna.fna file)
```

Next we'll cover more advanced editors to view and modify files.

Emacs vs. vi vs. eclipse vs. nedit vs. ...


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

xkcd.com/378

Code editors

I am not going to join the incessant, almost religious, debates about the superiority of one code editor over another. It is much more important that you choose one and spend a lot of time learning all of its features than choose the 'right' one. Some editors run within your terminal window (like vi, and older emacs) while others open a new window within your graphical environment. How do you know where to start? I'll give you my (mostly) unbiased short summaries of some of the most popular, but you should try them all out yourself.

eclipse – This is a full-fledged, professional-level graphical IDE. It is the *de facto* standard for Java developers and those in structured, team-driven development environments.

emacs or xemacs – This uses keystrokes for operations you might normally use a mouse for. It has a very steep learning curve but those developers I've seen master it seem almost godlike in their ability to fling code around.

nedit – A simple, graphical editor with a very low learning curve and surprising amount of features including languages modes, syntax highlighting and keyboard macros.

vi – This is similar to emacs in that it uses keystrokes to perform most operations rather than a mouse, but works only within your terminal windows. This is the most common editor I use when I want to quickly modify a single part of the file, save, and close it.

This is in no way an exhaustive list but includes enough of the most popular alternatives to get started.

Output redirection

Many commands that you run generate output, and UNIX allows you to redirect that output to a file or other programs instead of just displaying it on the screen. There are a few basic tenets of core UNIX programs that make this very useful:

Programs should do one thing, well.
They should be written to connect to other programs via input/output streams.

This allows you to both redirect the output of these tools either to the screen, files on disk or directly as input to another tool. Powerful operations can be performed on the command-line by streaming these tools together. For example, the following series of commands would count the number of entries in a directory full of FASTA files:

```
$ cat *.fna | grep “>” | wc -l
```

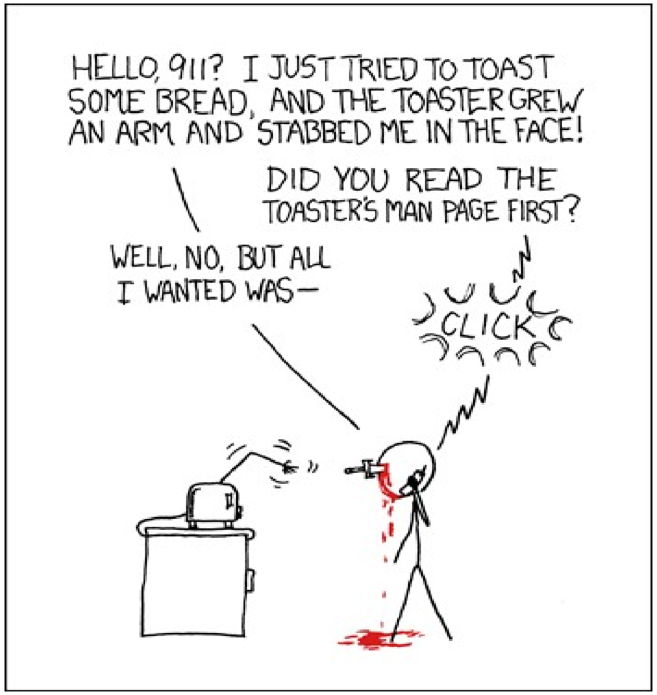
There is a lot of new material here. The *cat* command spews out the contents of the passed files, and in this case a wildcard '*' character is used to say “anything followed by .fna”. So this same command would work if there were 10 or 10,000 files in the directory. This output is then streamed to the *grep* command using the '|' (pipe) symbol. Grep is used to search for strings or patterns within an input file or stream, and will return only those lines of the input that match the search pattern. In this case, we use grep to filter the input to match the FASTA header lines. This output is then piped to *wc*, which is used to count words, lines or characters. The '-l' option to wc specifies that we want a count of the lines returned by grep.

Help with these commands

We have only scratched the surface of the functionality of many of these commands. The best way to learn more, or get clarification, is to use some of the built-in resources and documentation. The most common way is to use the *man pages*. For any command, such as the scp command you just learned, you can pull up that command's <u>man</u>ual like this:

```
$ man scp
```

This will provide often extensive documentation from the tool's author. This should always be your first resource when you need help with a command.


HELLO, 911? I JUST TRIED TO TOAST SOME BREAD, AND THE TOASTER GREW AN ARM AND STABBED ME IN THE FACE!
DID YOU READ THE TOASTER'S MAN PAGE FIRST?
WELL, NO, BUT ALL I WANTED WAS—
CLICK

xkcd.com/293

Other great resources (besides Google) when learning Linux and its tools include:

linuxquestions.org
help.ubuntu.com/community/UsingTheTerminal

Also, IRC is very old but is still very active and one of the best ways to get immediate answers from dozens of other users. There are channels for thousands of topics using a live chat mechanism. Here's a useful getting started guide:

www.linux.com/archive/feed/61439

More commands

In this series of slides we have gone through many common UNIX commands and learned how to get information on new ones we encounter. You should use these resources to find information on these other commands we will use throughout the class.

```
sort          (sorts input)
cut           (filter sections of input)
find          (searches for files/directories)
ps            (view the processes on a computer)
kill          (kill a running process)
tar           (creates an archive of files/directories as a single file)
gzip          (compresses a file)
```

If you are newer to the command line you might find it helpful to refer to a 'cheat sheet' that you keep printed nearby. There are many of these online, but here's an example:

http://fosswire.com/post/2007/08/unixlinux-command-cheat-sheet/

Further reading

Unix commands tutorial and reference:
www.unixtutorial.org/commands

Comparison of open-source licenses
en.wikipedia.org/wiki/Comparison_of_free_software_licenses