

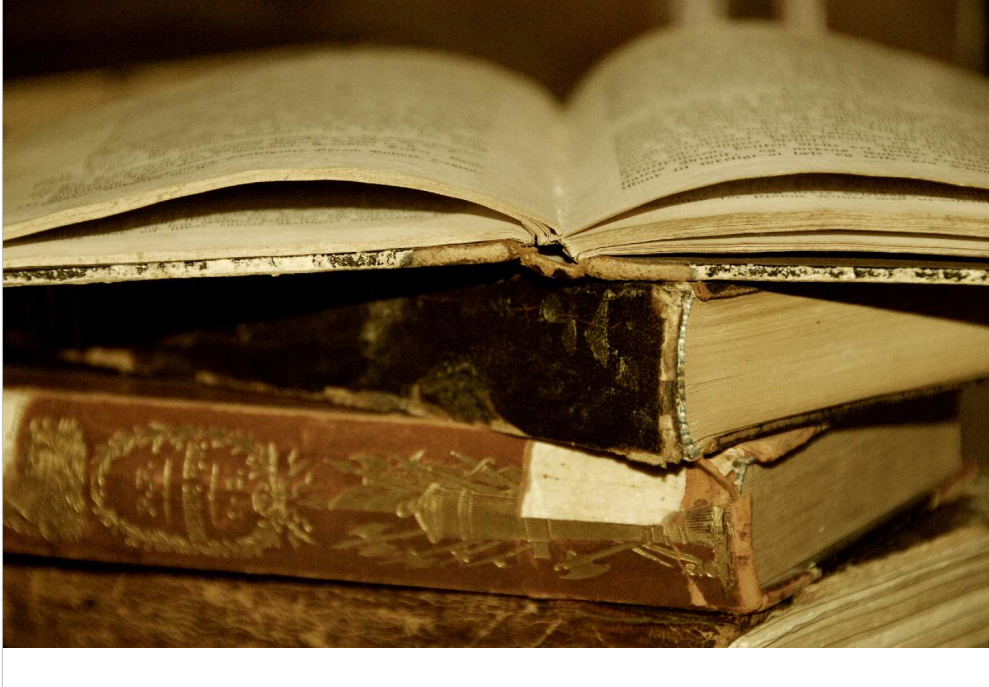
Physics 2660

Fundamentals of Computational Physics

Instructor: Bryan Wright (bryan@virginia.edu)
TA: Sachith Dissanayake (sed5ta@virginia.edu)

Welcome to Physics 2660!

Part 1: About this Course



Who is this class for?

Everybody is welcome, but this class is designed for beginning programmers in the sciences.

It's required for physics and astronomy majors.

Some of the Topics We'll Cover:

- **Introduction to computer programming**
 - Using the Linux operating system,
 - Using editors and compilers,
 - C programming basics (and some C++),
 - Programming style and techniques,
 - Use and construction of code libraries,
 - Debugging,
 - Optimization
- **Data analysis and simulation**
 - Numerical methods for differentiation, integration, iterative solutions, root/maxima/minima finding,
 - Visualization of data distributions,
 - Fitting models to data,
 - Simulation and Monte Carlo modeling of physical processes,
 - Error analysis and hypothesis testing

Building a strong foundation for your future computing projects!

3

In other words, we'll cover the basics of programming, and then look at some specific uses for programming in Physics (and other scientific/engineering fields).

Class Times and Homework:

Tuesdays: Lecture in Physics 205, 2:00-3:15

Thursdays: Labs in Physics 22, 12:30-2:00, 2:15-3:45
(register for one section)

Homework: Sent out after Labs.

Typically due at 10 am the following Thursday.
(electronic submission – don't be late).

Paper copies due at lab meeting.

4

Your TA will be setting up some times each week, outside of our regular lab times, when he'll open up the lab room for your use, and be available to answer questions there.

If you need to see me outside of class times, just send me an e-mail and we can arrange an appointment. You're also welcome to stop in and see me any time in my office (Physics 315).

Approximate Grading Breakdown:

Labs/quizzes: 20%

Homework: 30%

Midterm: 20%

Final Project: 30%

Occasional quizzes may be given before the class/labs. These are mainly to check progress on material in class and reading assignments.

5

In lieu of a final exam, we'll have a final project. This will be a programming assignment that will be due at the time we'd otherwise have a final exam. It will essentially be a long homework assignment.

Class Web Page:

faculty.virginia.edu/comp-phys/phys2660

Announcements area →

start [Phys2660 Spring11] - Mozilla Firefox

http://faculty.virginia.edu/comp-phys/phys2660/

start [Phys2660 Spring11]

PHYS2660 SPRING11

Trace: » labs_start » start

PHYS 2660: Fundamentals of Computational Physics 2011

The course formally known as Physics 254.

Class Links:

Syllabus	Meeting Times	Instructor/Staff
Labs	Homework	Code Library
Reading Assignments/PreLabs		
Class Notes		

Getting Started / Logging on	5 minute Linux guide	FAQ
Text Editors	Online resources	Style guide
Texts / Reserves	Software for home	
Getting Help	Discussion Group	Email Archive

[C/C++ Reference Guide](#)

Announcements

The first class meeting will be on January 25, 2011 at 2pm in Physics 205. See you then! (There will be no Lab meeting during the preceding week.) --Bryan

[About this website.](#)

Done

The web page includes:

- Class notes,
- Program examples and tips,
- Assignment solutions,
- Documentation,
- etc...

After each lecture I'll put up that week's lecture notes.
The format will be a PDF file with annotated slides.

I'll also leave the lecture notes from the previous semester on the site, in case you find them useful.
These were created by Bob Hirosky, who taught this course previously.

Textbooks:

- *C Programming: The Essentials for Engineers and Scientists*, by David R. Brooks
(Main text, covers introduction to C language, many programming examples.)
- *Statistics for Nuclear and Particle Physicists*, by Louis Lyons

Online resources:

- *C/C++ Programmer's Reference*, by Herbert Schildt
(Very useful programming language reference. You'll find a link to it on the class web page.)

7

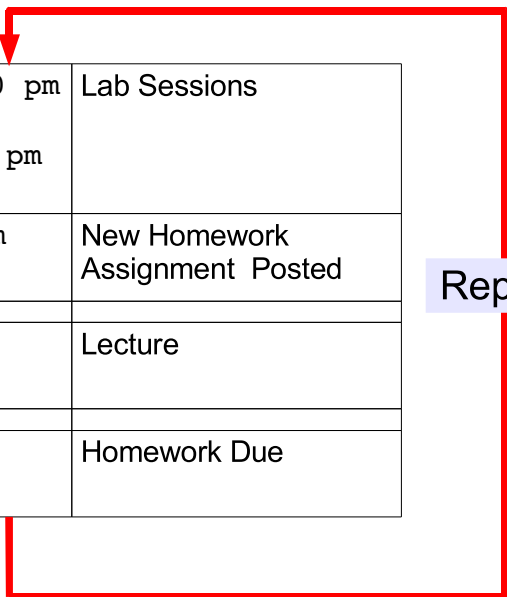
There are other books you may find useful, but they're not required. In particular, the reading assignments for the Lyons book will also point you to equivalent sections of another good book:

“Data Reduction and Error Analysis for the Physical Sciences”, by Philip R. Bevington.

You'll find many other books on the reserve shelves of the Physics Library (3rd floor of the Physics Building).

Assignment Schedule:

The schedule for a typical week will look like this:



Thursday	12:30-2:00 pm or 2:15-3:45 pm	Lab Sessions
	by 5:00 pm	New Homework Assignment Posted
Tuesday	2:00 pm	Lecture
Thursday	10:00 am	Homework Due

Repeat

8

Note that the homework assignments will be posted on the web site on Thursdays, after the lab sessions. I'll send out an e-mail each Thursday letting you know when they're there, and sometimes giving you some extra tips about that week's assignment.

You'll be turning in the homework in two formats: Electronically, through a submission system you'll try out in the first pre-lab exercise, and in printed form, with a signature indicating that the assignment is pledged. Electronic submissions are due by 10am on the following Thursday, and paper versions should be brought to your lab section.

See the "Homework" section of the web page for more information.

Warning and Reassurance:

If you've never programmed before, it can seem very confusing at first. Don't give up! The best way to learn is to ask lots of questions!



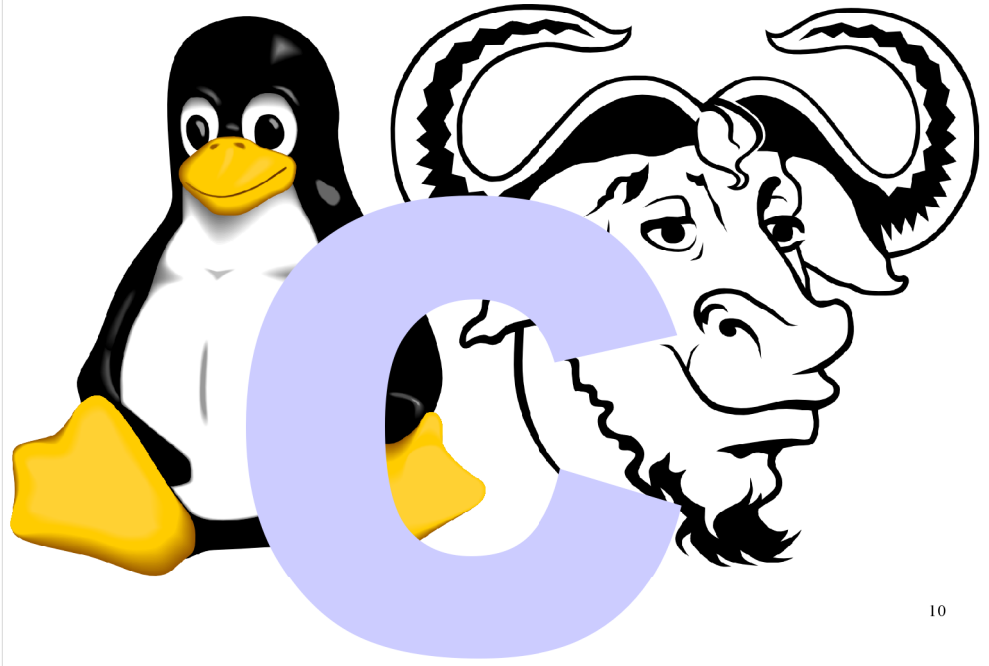
Help is available:

- TA Supervised Lab hours
- E-mail
- Discussion Group in Collab
- etc...

There will be lots of opportunities to get help. I want each of you to come out of this semester feeling like you've mastered something new. Your TA and I are there to help.

And don't forget the web site.

Part 2: Linux, GNU Software and C



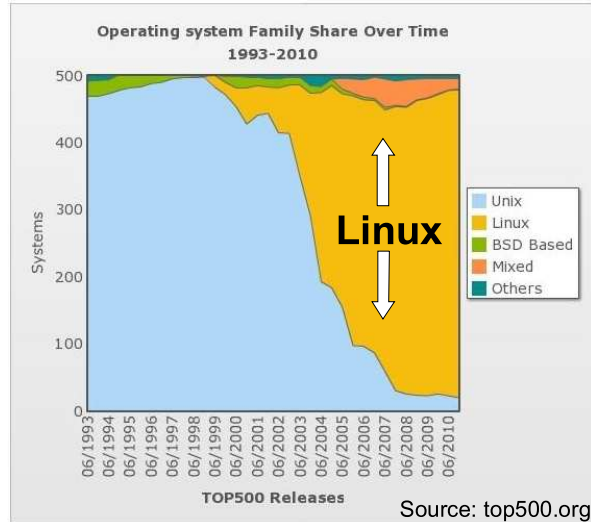
So, let's talk about some of the software we'll be using this semester. I'd like to start out by giving you some context for understanding the part of the computing world we'll be working in.

Why Linux?:

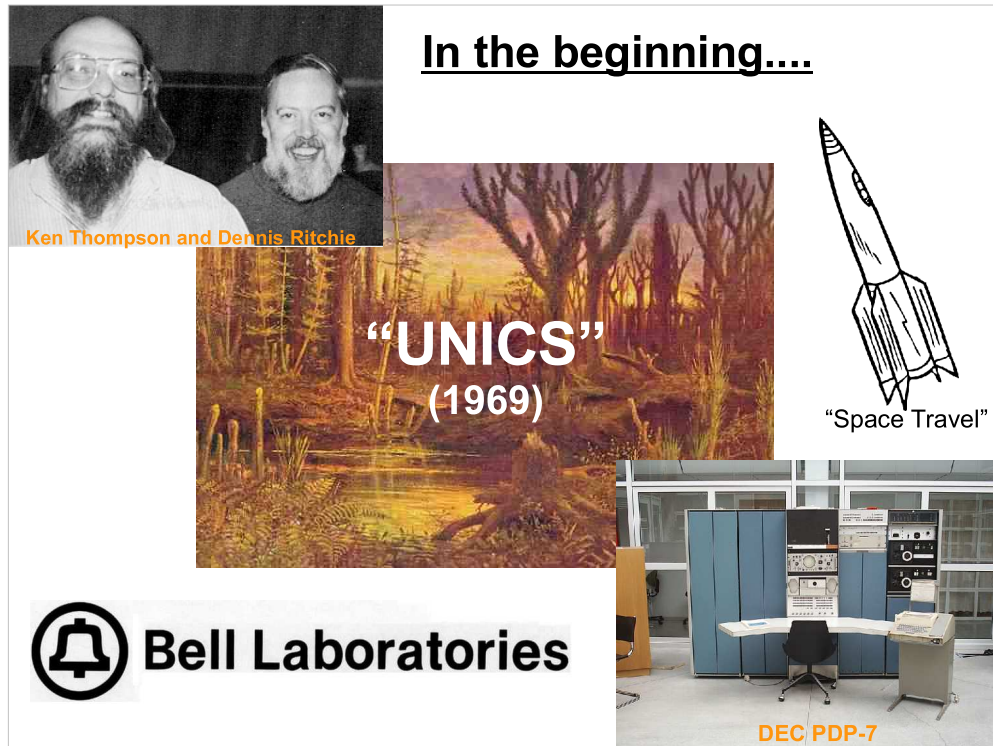
Linux-based computer systems are a mainstay in the world of scientific computing. In any laboratory setting where the research requires large amounts of data processing or computationally intensive calculations, you will routinely find a Linux/Unix cluster of computers handling the workload.

As of Nov 2010 **92%** (96%, counting Unix) of the world's top performing computer systems operate on Linux. Linux is the overwhelming choice for building world-wide high performance computing Grids.

How did Linux become so popular?



(Twice a year top500.org posts benchmark measurements for the top 500 fastest computers on earth.)



In 1969, something called “UNICS” crawled out of the primordial ooze. Ken Thompson and Dennis Ritchie of AT&T Bell Labs were working on a GE mainframe under an operating system called MULTICS.

Thompson had written a game called “space travel” that ran very slowly on the GE machine. The lab had recently acquired a DEC PDP-7 computer, and Thompson began re-writing his game in assembly code for the PDP-7, where he hoped it would run faster.

This effort eventually grew into an entire operating system called UNICS, in contrast to MULTICS. AT&T originally took no interest, but eventually saw the value of the new operating system and began selling it under its new name “UNIX”. It was pretty successful....

<http://www.levenez.com/unix/>



Richard M. Stallman (RMS)



GNU Project: "GNU's Not Unix"
(1983)

"The Four Freedoms:"

0. The freedom to run a program for any purpose
1. The freedom to study and adapt a program
2. The freedom to redistribute
3. The freedom to improve and release improvements



Xerox 9700 Laser Printer



A few years later we meet a very bright student named Richard M. Stallman (RMS). RMS graduated magna cum laude from Harvard and entered grad school (in physics) at MIT. As an undergrad and a grad, he worked in MIT's Artificial Intelligence Lab, and made significant contributions to the field of AI.

When he began his work, he was immersed in the hacker culture of the time, which valued the free exchange of ideas. Things were beginning to change, though. One example involved the first laser printer, made by Xerox. RMS had been able to get source code from Xerox for their earlier printers, so he could modify the printers' behavior. (Xerox was, after all, in the business of selling printers, not software). But Xerox refused to give him the code for their new laser printer, saying that it was proprietary.

As time went on, MIT spawned off many companies around proprietary software, and even began selling Stallman's own code. All of this upset RMS greatly. He believed that users and developers of software should have what he called "four freedoms", shown above. Toward that end, in 1983, he began developing software for what he hoped would eventually be a complete, free (in terms of the "four freedoms") operating system, which he called "GNU", for "GNU's Not Unix" (the first of many recursive acronyms).



Richard M. Stallman (RMS)



The GNU General Public License (GPL)

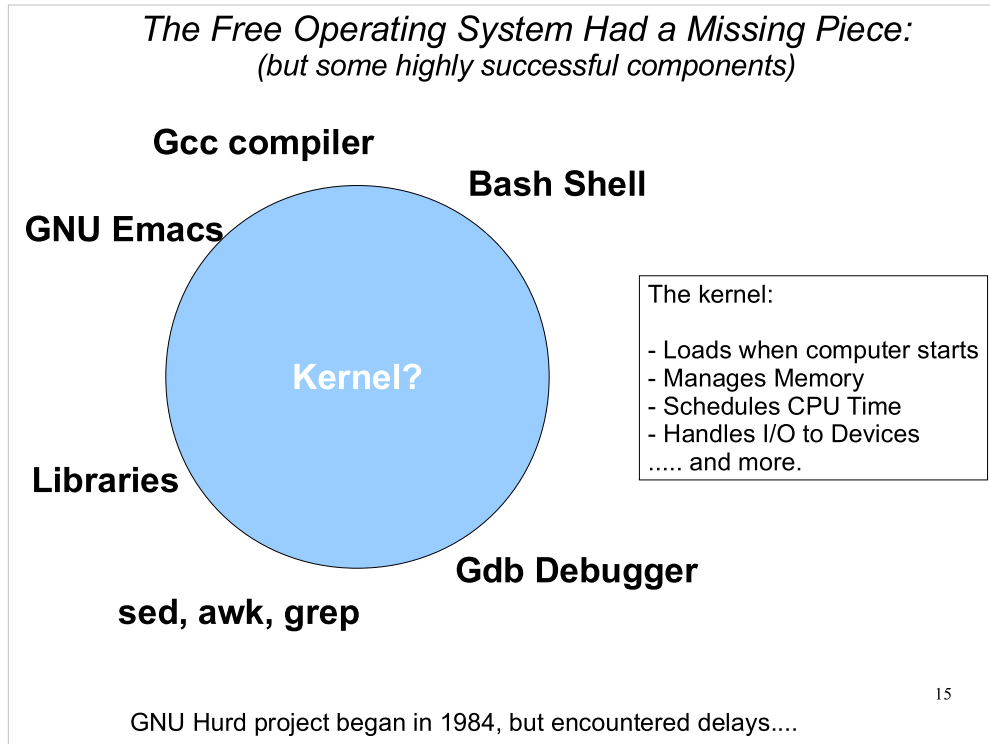
"...if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights."

14

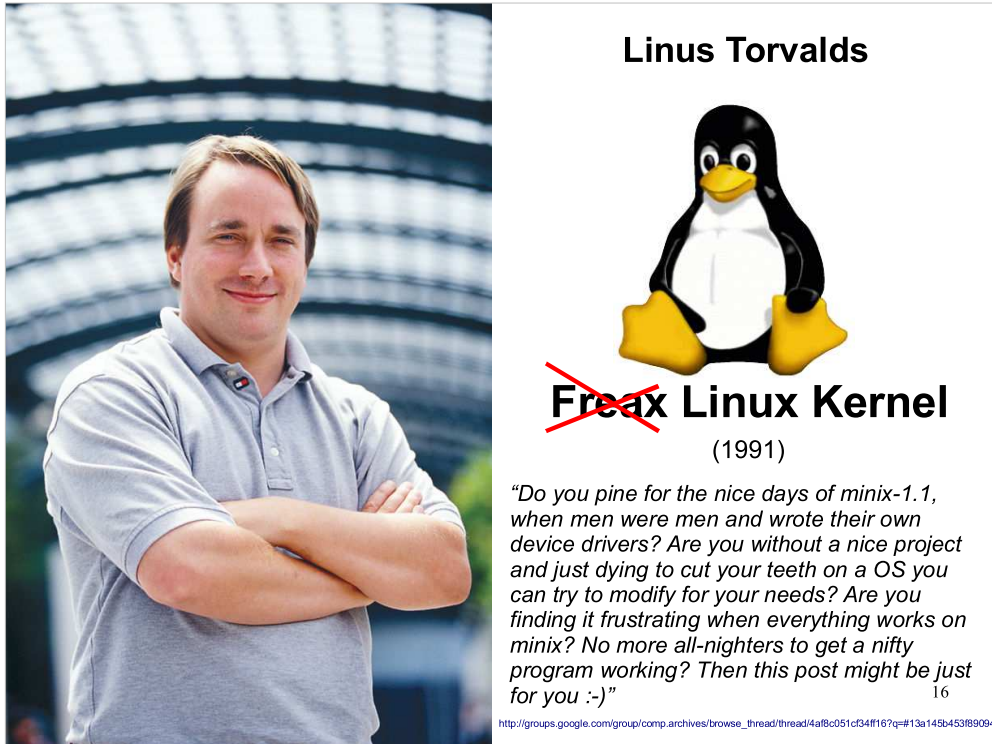
To foster the development of free software, RMS founded the “Free Software Foundation” in 1985. His plan was that the FSF would hold copyrights on GNU software, and would license the use of the software under the terms of the GNU “General Public License”, which would ensure that the software always remained free.

<http://www.fsf.org/>

<http://gplv3.fsf.org/>



By the late 80s, the GNU project had produced an impressive suite of tools, many of which were widely used (gcc and emacs, for example). These were largely independent of each other, and could be used on a wide range of common operating systems, replacing or supplementing tools already found in those operating systems. But a missing piece was still necessary to make GNU a complete operating system on its own. That piece was the kernel. In 1984 RMS had started the Hurd project, with the goal of making a GNU kernel, but it wasn't getting anywhere fast.



Then, in 1991, we meet another bright student. Linus Torvalds was a Finnish engineering student. He'd been inspired by an Operating Systems course he'd taken that used Andrew Tanenbaum's “Minix”, a toy implementation of Unix intended for teaching purposes.

Minix was simple enough to understand, and it ran on cheap, readily available PCs, but it wasn't free. Linus decided to try writing his own Unix-like kernel, to be released under the GPL, and in 1991 posted the message above to the comp.os.minix newsgroup, inviting other developers to download his code and help him work on it.

He originally called the kernel “FREAX”, for “Free Unix”, but the administrator of the download site named the directory “linux”, for “Linus's Unix”, and the name stuck.

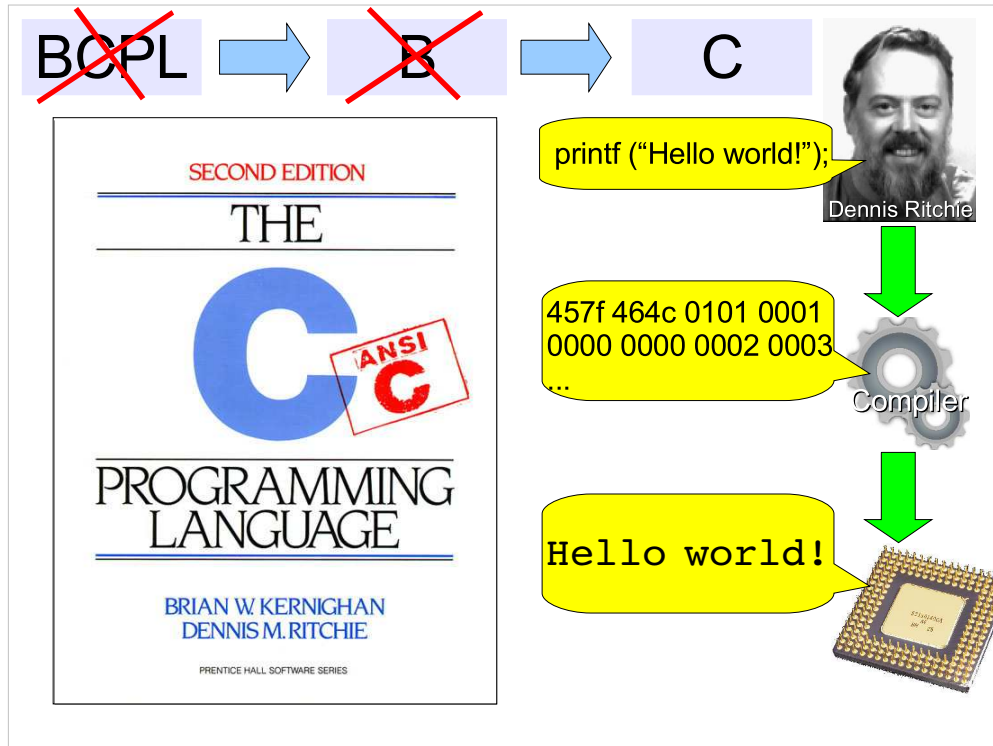


Our working definition of “Linux” for this class:

A complete operating system based on the Linux kernel and including the tools and utilities necessary for running applications.

17

Strictly speaking “Linux” just refers to the kernel that Linus Torvalds began developing. But when people say “Linux” these days they often mean a complete operating system that uses the Linux kernel. That's the definition we'll use for this class.

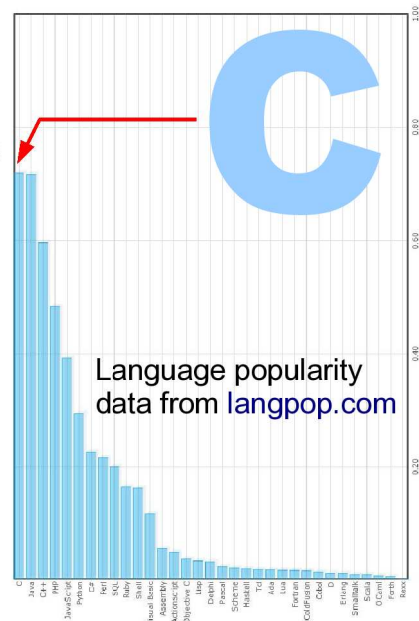
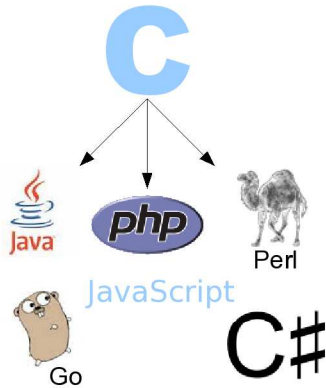


While working on the “Unics” operating system, Ken Thompson and Dennis Ritchie realized that they needed a more convenient way of writing machine code. They needed a **compiler** that could take high-level, readable commands in a specialized **computer language** and translate these into machine code.

At first, they used an existing language called **BCPL** (“Basic Combined Programming Language”). Ken Thompson developed his own version of this and called it “**B**”. Dennis Ritchie improved on this, and called his new language “**C**”.

Why Learn C?

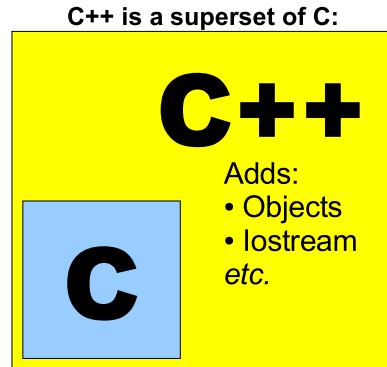
- Simple
- Widely-Used
- Exposes low-level concepts
- Basis for many other languages:



C or C++?

Both! Most of the code we'll write in this class will be C, but we'll use the C++ compiler for everything, because it also understands C. The advantages of this are:

- This will help you learn to **avoid habits** that break C++.
- C++ is **pickier** about sloppy coding (even when writing C-style code).
- It lets us **introduce some C++** programming without requiring you to change the way you compile and run your programs.



20

In this course we'll show you how to use an alternative system for reading/writing data (called “iostream”) that's offered in C++, and we'll take a brief look at something called “classes”, which introduces a whole new programming philosophy called “object oriented programming”. We won't make much use of these tools, but I want you to recognize them when you encounter them later in your career.

Part 3: Using Linux



This is a teletype terminal of the type I used back in the 1970s, when all we had was the command line, and We Liked IT!

Linux is still maybe a little more weighted toward the command line than Windows. Looking at the Linux World, you could say that almost anything you can do at the command line can also be done graphically. In the Windows world, I think it's fair to say that almost anything you can do graphically can also be done from the command line.

The two worlds are converging: Linux's graphical interfaces are continually improving, and Windows keeps improving its command-line interface.

We can all appreciate the value of a graphical interface. It's intuitive (if it's well-designed) and it's "discoverable", in that you can browse around a graphical program's menus and find out what the program can do. But what's the value of a command-line interface? If graphical interfaces are good, why do all major operating systems continue to improve their command-line interfaces?

The Command Line:

The diagram illustrates the components of a terminal command and its output. It shows a terminal prompt `~/demo>` followed by the command `ls`. The output of the `ls` command is listed as follows:

```
clus.pdf      data-for-everybody.1.dat  phase2
cluster.pdf   ForYourEyesOnly.dat      readme.txt
cpuinfo.dat   phase1                    ReadMe.txt
```

Arrows point from the labels 'Prompt', 'Command', and 'Results' to their respective parts in the terminal output.

Output of the "ls" command, which lists the files in the current directory.

Why should you do things from the command line?:

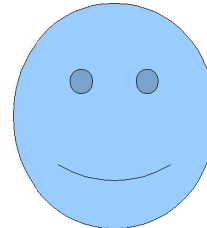
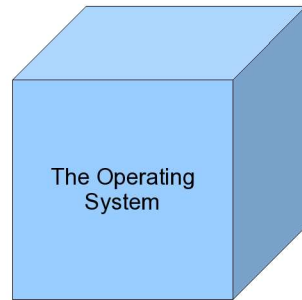
- * In Linux, graphical tools provide a front-end to help you do tasks, but you can **do more** from the command line.
- * There are **several sets** of graphical tools available for Linux, so if you learn one of them you may find that it's not available on the next computer you use.
- * There's no guarantee that a given computer will **have graphical tools installed**, or even a monitor.
- * Text commands are **easily reproduced**. It's easy to document what you've done, or to tell someone else how to do it, or to automate what you've done.

22

The answer is that the command line has its own advantage. Here are some of the things that might make someone choose to use the command line under Linux.

The last item is the most important, I think. This is true for all operating systems, not just Linux, and it's why all major operating systems still have a command-line interface and continue to improve it.

What's a Shell?



The User

The Shell

The Shell intermediates between the user and the operating system. It accepts commands from the user, uses the operating system to execute them, and then returns results to the user.

There are graphical shells and command-line shells. When you log in to Microsoft Windows, you're using a graphical shell. In that case, you communicate with the shell by pointing and clicking. Today we'll be talking about command-line shells, where you communicate by typing commands.

A Few Useful Linux Commands:

ls	List the contents of a directory.
pwd	Show the name of the current directory.
cd	Change the current directory.
less or more	Show the contents of a file, one page at a time.
cp	Copy a file.
mv	Move (rename, relocate or both) a file.
rm	Delete (remove) a file.
mkdir	Make a new directory.
rmdir	Delete (remove) a directory.
man	Show docs (manual pages) for a command.
ln	Make a link to a file.
cat	Spit out the concatenated contents of one or more files, without paging.
touch	Change the timestamp on a file, or create an empty file.
which	Find a command in the search path.

24

This is just a list to get you started.

As you can see, the commands are typically terse.

Command Syntax:

```
~/demo> ls -l
total 60
lrwxrwxrwx 1 bkw1a bkw1a    11 Jan 18 11:39 clus.pdf -> cluster.pdf
-rw-r----- 1 bkw1a bkw1a 20601 Jan 18 10:51 cluster.pdf
-rw-r----- 1 bkw1a demo   983 Jan 18 10:53 cpuinfo.dat
-rw-r--r--  1 bkw1a bkw1a   29 Jan 18 10:59 data-for-everybody.1.dat
-rw-----  1 bkw1a bkw1a   41 Jan 18 10:56 ForYourEyesOnly.dat
drwxr-x---  3 bkw1a bkw1a  4096 Jan 18 11:35 phase1
drwxr-x---  2 bkw1a bkw1a  4096 Jan 18 10:55 phase2
-rw-r----- 1 bkw1a demo    72 Jan 18 10:52 readme.txt
-rw-r----- 1 bkw1a bkw1a  9552 Jan 18 10:52 ReadMe.txt
```

Linux commands are often modified by the addition of switches or qualifiers like the “-l”, for “long”, switch used in the ls command above. These modifiers will often take one of these forms:

- * A dash followed by a letter or number, optionally followed by an argument
- * Two dashes followed by a word, optionally followed by an argument.

For ls, some useful switches are:

- l Gives more information about the files.
- T Combined with -l, sorts the files in reverse time order.
- S Combined with -l, sorts the files in order of descending size.
- a Lists all files, including hidden files.

25

Multiple single-letter switches can often be combined, like “ls -lT” instead of “ls -l -T”

In this case, we can change the behavior of the “ls” command by adding the “-l” switch.

Note, though, that Linux commands were all developed independently, and they have a long history. Syntax conventions have evolved over time, and different developers have used different conventions.

We'll see examples of some odd command syntax with commands like “tar” and “ps”.

Case Sensitivity:

Important Note: when typing commands, file names, etc...

Linux and C/C++ are

CaSe SeNsItIvE

So, for example:

This is not the same as **this**,
VELOCITY is not the same as **Velocity** or **velocity**,
MyFile.dat is not the same as **MyFile.DAT**

For best results, stick to all lower-case unless there's a good reason to do otherwise.

Part 4: Remote Shells



27

So, how do we give commands to a remote computer?
Computer networks have been around for a long time now. One of the first uses of these networks was the execution of commands on remote machines.

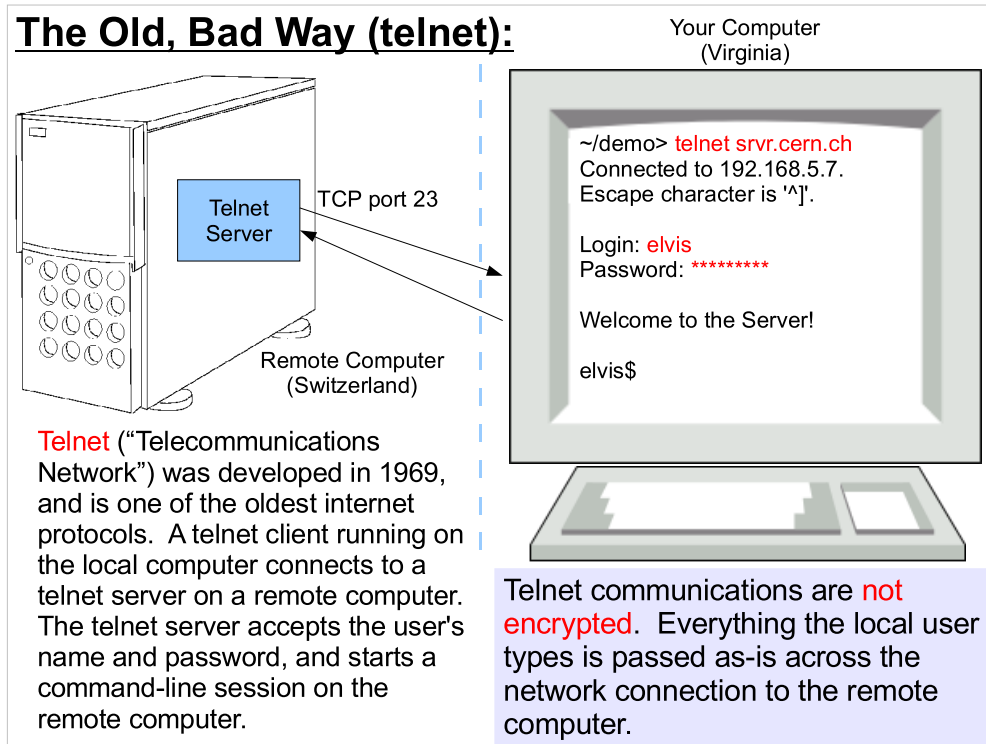
The “Galileo” Cluster:

In this class, we'll do our programming on a Linux cluster called “Galileo”. Galileo doesn't have any keyboards or monitors connected to it. So how do we use it?

Galileo Web Page →



Again, I'll approach the answer asymptotically, starting with a little history...



Note that nothing telnet transmits is encrypted, including your username and password when you log in.

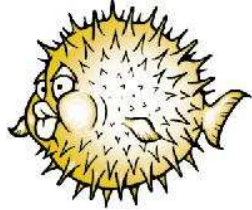
Telnet was written at a time when computers were behemoths, and few people had access to them (or the networks that connected them). Just getting access to a remote computer was a Big Deal. Security was of little or no concern.

Although some network-connected printers and home network routers still accept telnet connections for the purpose of configuring them, there's little need for telnet any more, and it should be avoided whenever possible.

The New Way (ssh):

In 1995, Finnish researcher **Tatu Ylönen** wrote “ssh” (“Secure Shell”) as a more secure replacement for telnet. This soon evolved into a proprietary commercial product.

In 1999, **Björn Grönvall** began with earlier, free versions of Ylönen's code and began writing what was to become OpenSSH, a completely open-source re-implementation of ssh. This is now the de facto standard ssh implementation, and is included in many operating systems.



OpenSSH uses a blowfish as its logo because “blowfish” is the name of the strong, public-domain encryption algorithm it uses (by default) to encrypt network traffic. See: [http://en.wikipedia.org/wiki/Blowfish_\(cipher\)](http://en.wikipedia.org/wiki/Blowfish_(cipher))

MS Windows doesn't have ssh software, but you can install it for free:

- **Putty** Is a free ssh client for Windows. It's easy to download and install.
- **SecureCRT** Is a commercial product, but UVa has a site license. ³⁰

From 1969 to the 1990s, telnet was the standard way of getting a command-line connection to a remote computer. It still had little security, but the world had changed. Now many people had access to computers and the network. Following a breakin at his University, Tatu Ylönen finally got fed up with the lack of security in telnet, and wrote “ssh” as a replacement.

In this class, we'll use ssh to talk to Galileo. It's already installed on the computers in the lab, and in ITC's public computer labs, and you can install it on your own computer if you like, using the links at the bottom of the slide above.

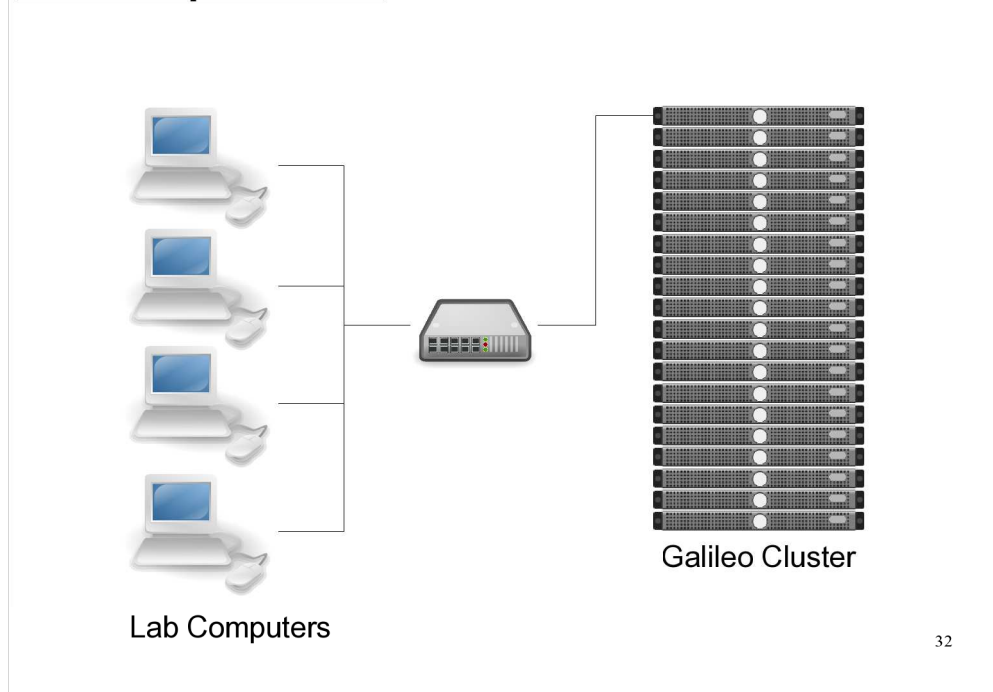
Part 5: Using the Computer Lab



31

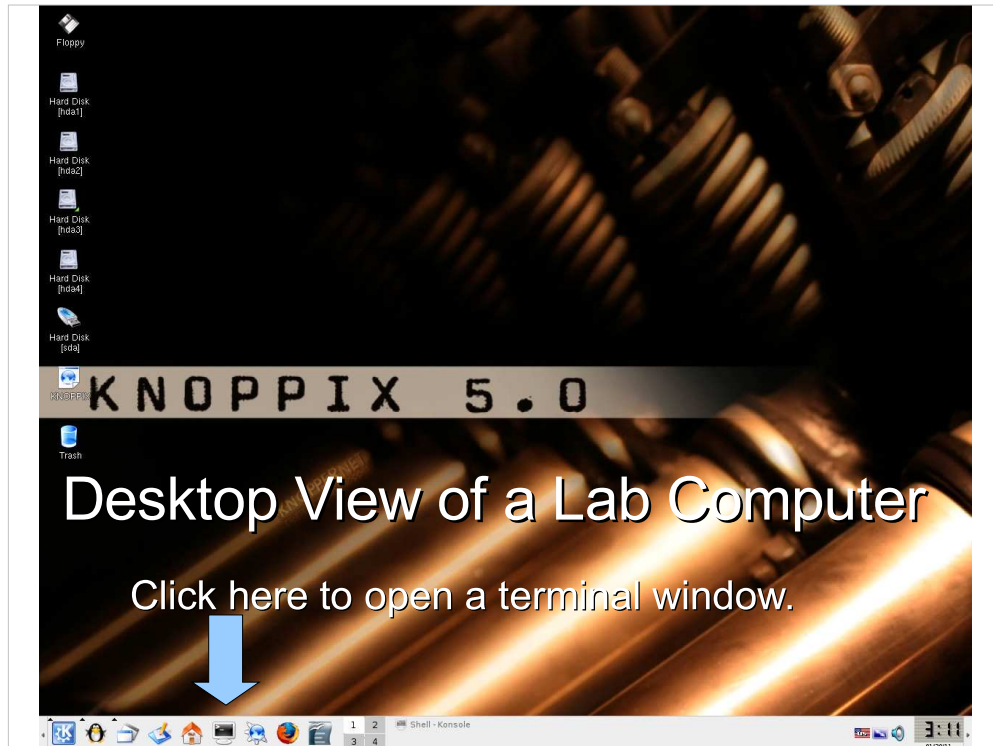
Our computer lab is in room 22, in the basement of the Physics Building. It's just inside the back door closest to the Ed school. (It doesn't really look like the picture above.)

Our Computer Lab:



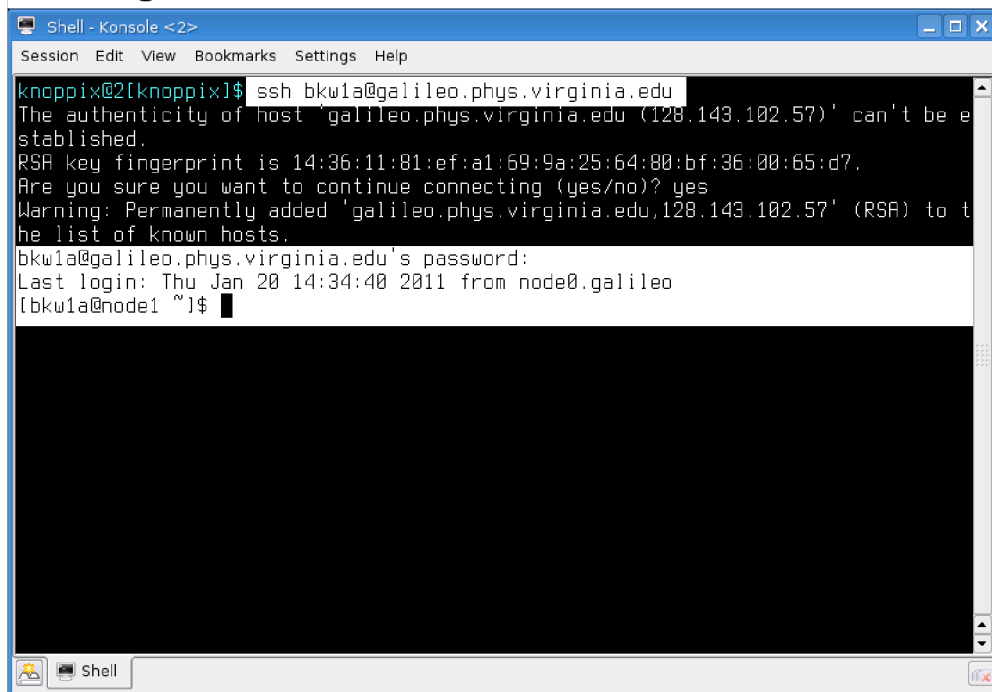
The computer lab we'll be using (room 22) contains several desktop computers running a Linux distribution called KNOPPIX.

How can we use the keyboards, monitors and mice (mouses?) on these lab computers to communicate with Galileo?



The KNOPPIX desktop will probably look similar to the desktop of other computers you've used. At the bottom left, there's a button that will pop up a menu of programs, and along the bottom of the screen there are a few icons for commonly-used programs, like the terminal emulator we'll be using.

SShing to Galileo from the Lab:

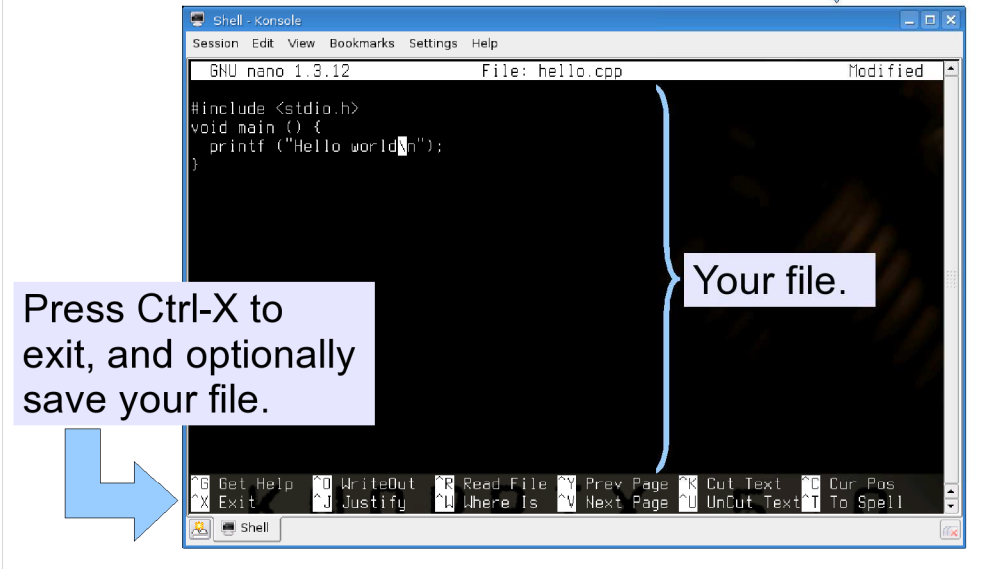


```
Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
knoppix@2[knoppix]$ ssh bkwl1a@galileo.phys.virginia.edu
The authenticity of host 'galileo.phys.virginia.edu (128.143.102.57)' can't be e
established.
RSA key fingerprint is 14:36:11:81:ef:a1:69:9a:25:64:80:bf:36:00:65:d7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'galileo.phys.virginia.edu,128.143.102.57' (RSA) to t
he list of known hosts.
bkwl1a@galileo.phys.virginia.edu's password:
Last login: Thu Jan 20 14:34:40 2011 from node0.galileo
[bkwl1a@node1 ~]$
```

This shows what it looks like when you use SSH to connect to Galileo from one of the lab computers. The first time you do it, SSH will ask you to confirm that this is really the computer you want to talk to. It displays a “fingerprint” that you could verify really belongs to Galileo. (Trust me, it really does.) You'll need to accept this by typing “yes”, and then you'll be asked for your password.

Using the “nano” Text Editor:

```
[bkw1a@node1 ~] nano hello.cpp
```



There are many text editors on Galileo, but I recommend you start out with a simple editor called “nano”. To use nano to edit a file, just type “nano” followed by the file name. Once in nano, you can type normally, or you can use special control keystrokes to do things like saving your file or searching. These are listed at the bottom of the window.

Compiling and Running a Program:

The `g++` command invokes the GNU C++ compiler and tells it to translate our file `hello.cpp` into machine code, which it will write into a new file called just `hello`.

```
[bkw1a@node1 ~] g++ -o hello hello.cpp  
  
[bkw1a@node1 ~] hello  
Hello world!
```

Now that the output file has been created, we've **created a new command** that we can use! Typing `hello` (the name of the output file) runs our program.

36

As we'll see later, you'll usually develop your program by repeatedly editing, then compiling, then running it.

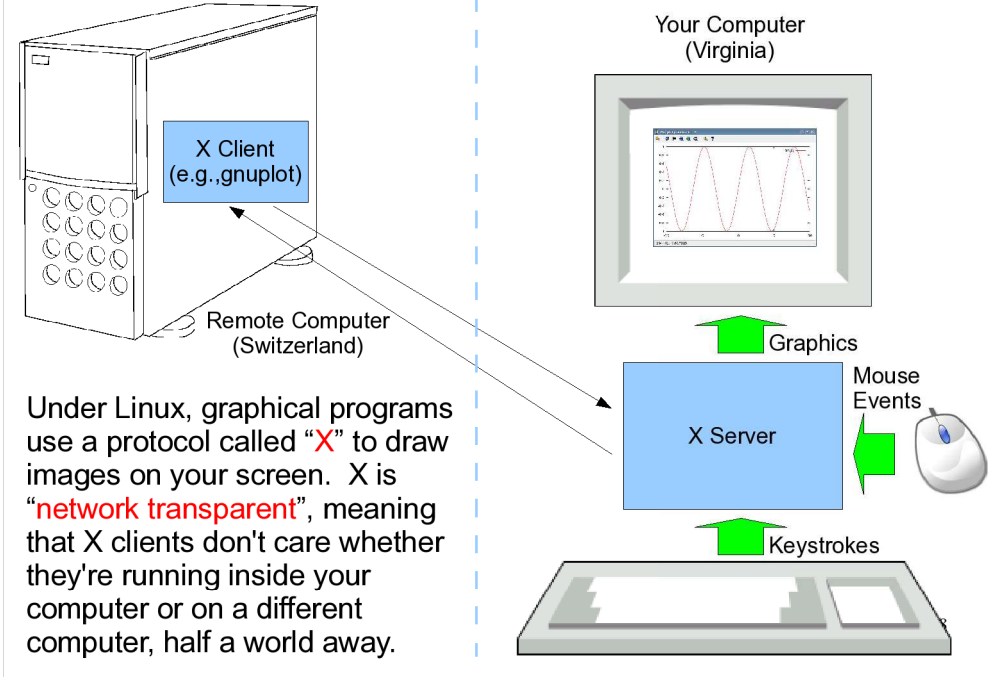
Part 6: Remote Graphics



OK, so ssh is great for typing commands and seeing text, but what about drawing graphics?

In the Linux world, graphics is done through a system called, simply, "X".

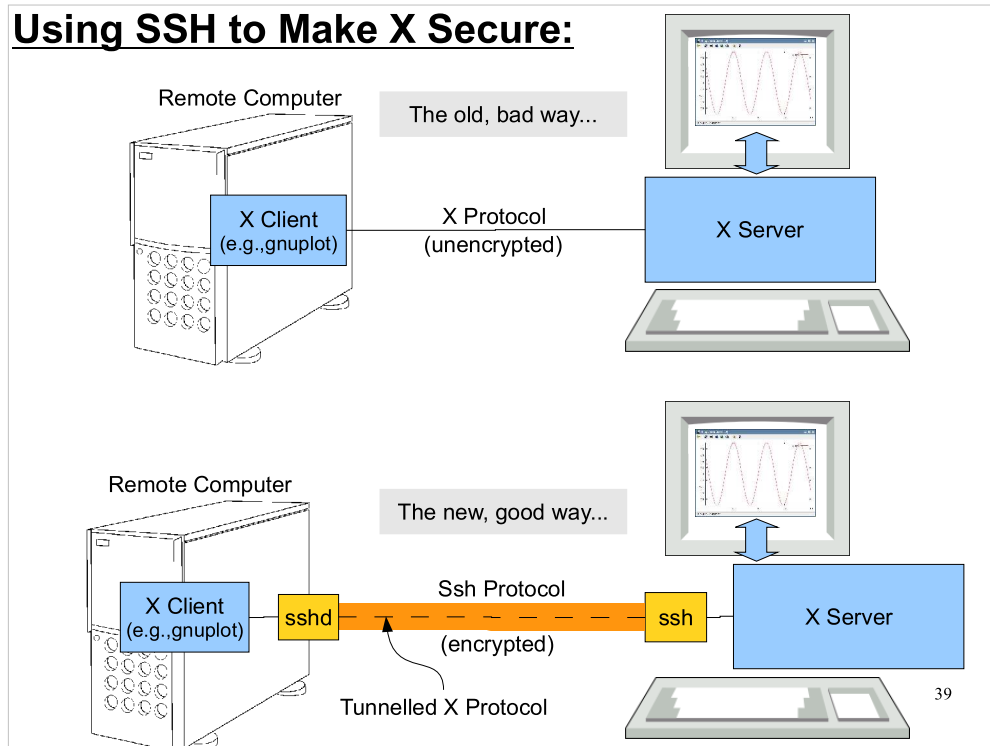
Remote Graphics:



X programs are called “clients”. They send their graphical output to an “X server” program running on your local computer. The X server sends keystrokes and mouse clicks back to the client.

X clients can also be run on the same computer as the X server. This is how locally-running applications display graphical data on a Linux computer. As far as the user is concerned, things look just the same no matter whether the client program is running locally or on a remote computer.

Using SSH to Make X Secure:



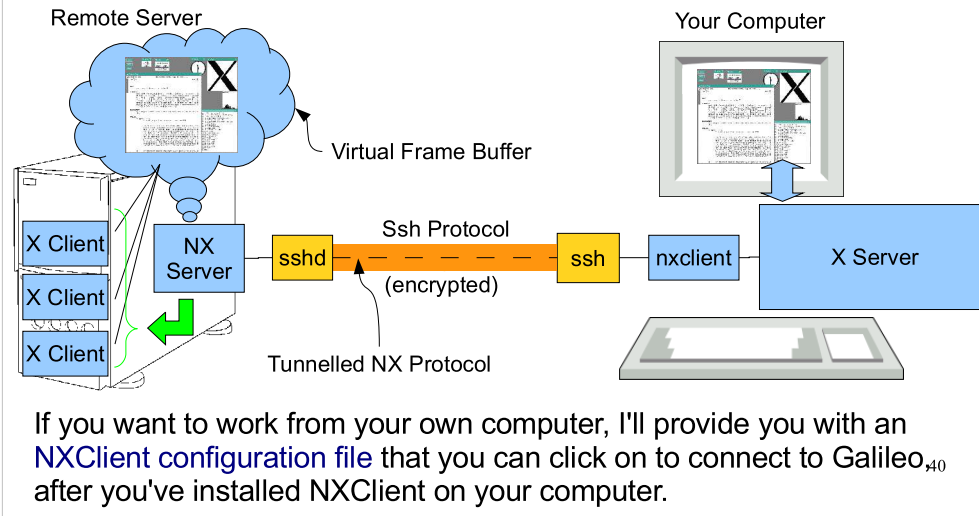
In the bad old days we used to use X in the way shown in the top picture. We'd allow a specific remote computer access to our X server, and then on the remote computer we'd point the remote X applications to our local computer's X display.

This is insecure for a couple of reasons: (1) the X protocol connection between the computers is unencrypted, and (2) any program running on the remote computer (which may not be completely trustworthy) is allowed to read our local keystrokes.

The better way to do it is shown in the bottom illustration. Just let ssh set up an encrypted, tunnelled X connection between the machines. This is easy to do, and it Just Works.

Working Outside the Lab:

The X protocol can be slow, especially over a slow network link. There are ways to greatly compress the X protocol, though. One of these is a free commercial product called **NXClient**. NXClient automatically sets up a secure connection to a remote computer and sends highly compressed X traffic through it. You can even use it over a dialup line!



You can download NXClient from here:

<http://www.nomachine.com/download.php>

After installing it, you'll also need to download the configuration file here:

<http://galileo.phys.virginia.edu/Galileo-xterm.nxs>

(Note that you **don't** need to install a separate X server on your MS Windows computer in order to use NXClient. Everything you need – except the configuration file – is included in NXClient itself.)

The NXClient program is available for Windows, Linux and OS X. It talks to a remote “NX server” that maintains a local virtual display in memory, called a “Virtual Frame Buffer” (VFB). The NX server starts up applications which send their display information to the VFB. The VFB can then be displayed locally by the nxclient application. All traffic between server and client is tunnelled through an encrypted SSH connection.

NX has the added benefit that its NX protocol is a highly compressed version of the X protocol, and will give fast performance even over a slow network connection.



The End

41

Thanks! We'll pick up here next week.