PHYS 27 Scientific Computing Tutorial Documentation

Release 1

James E. Hetrick

November 17, 2015

1 Introduction to Scientific Computing **3**
1.1 What is *Scientific Computing* . . . 3
1.2 The Scientific Computing Environment . . . 4
1.3 The OS and Unix . . . 8
1.4 The Scientific Software Stack . . . 12
1.5 Homework . . . 15

2 Installing your Scientific Computing Environment **17**
2.1 Overview . . . 17
2.2 Prepare to Install . . . 18
2.3 Install VirtualBox . . . 21
2.4 Creating a Virtual Machine (VM) . . . 27
2.5 Defining the VM . . . 27
2.6 Turn on Virtualization in your BIOS? . . . 29
2.7 Configuring the VM . . . 31
2.8 Attach the Fedora Scientific ISO Disk . . . 37
2.9 Installing Fedora Scientific on your VM . . . 42
2.10 Booting Linux from the DVD . . . 42
2.11 Install to Your Hard Drive . . . 44
2.12 Create Root and User Accounts . . . 49
2.13 Boot Linux from Your Hard Drive . . . 52
2.14 Installing ” Guest Additions ” on your VM . . . 55
2.15 First Look at Linux . . . 55
2.16 Prepare for Guest Additions . . . 62
2.17 Upgrade your Kernel . . . 65
2.18 Configuring the Look-and-Feel . . . 72
2.19 Shared Folders . . . 72
2.20 Terminal/Shell Startup Configuration: the `.bashrc` file . . . 75
2.21 Examine `.bashrc` . . . 77
2.22 Fixing the Shared Folders . . . 80
2.23 Fonts and Colors . . . 83
2.24 Homework . . . 84

3 Basic Unix Skills **85**
3.1 Moving Around in the Unix World . . . 85
3.2 Get Some Files . . . 85
3.3 The Unix File System . . . 85
3.4 The Directories `.` and `..` . . . 88
3.5 Pathnames . . . 89
3.6 Unix commands for files and directories . . . 90
3.7 Understanding pathnames . . . 93

3.8 Command Summary . . . 95
3.9 Copy, Move, Delete, and Examine Unix Files . . . 95
3.10 Video for this section . . . 95
3.11 Copying Files and Directories . . . 95
3.12 Moving and Renaming Files and Directories . . . 97
3.13 Removing Files and Directories . . . 100
3.14 Manipulating the contents of files . . . 101
3.15 Searching the contents of a file . . . 102
3.16 Sort the contents of a file . . . 104
3.17 Command Summary . . . 106
3.18 Controlling Data Flow . . . 107
3.19 The Pipe: | . . . 111
3.20 Command Summary . . . 113
3.21 Homework . . . 113

4 The Text Editor **115**
4.1 The Text Editor . . . 115
4.2 Learning More About Emacs . . . 122
4.3 Intermediate Emacs . . . 137
4.4 Homework . . . 151

5 Scientific Visualization **153**
5.1 Scientific Visualization . . . 153
5.2 More fun with Gnuplot . . . 168
5.3 3-D Plots with Gnuplot . . . 172
5.4 Plotting Data from a File . . . 178
5.5 Homework . . . 182

6 Data Analysis **183**
6.1 User Defined Functions . . . 183
6.2 Fitting Functions to Data . . . 186
6.3 Fitting Data II . . . 189
6.4 Saving and Printing . . . 206
6.5 Homework . . . 210
6.6 A Worked Example: Theory meets Experiment . . . 210

7 Scientific Publication **219**
7.1 Scientific Publications . . . 219
7.2 Latex Document Environments and Layout . . . 225
7.3 Error Messages from Latex . . . 245
7.4 Homework . . . 249

8 Intermediate LaTeX **253**
8.1 LaTeX Error Messages . . . 253
8.2 Physics Department LaTeX Templates . . . 253

9 Scientific Drawings **255**
9.1 Using Xfig for Drawings . . . 255
9.2 Starting Xfig . . . 259
9.3 Using Xfig . . . 260
9.4 Homework . . . 266

10 Remote Login and File Transfer **267**
10.1 Remote Login and File Transfer . . . 267
10.2 Get Connected with SSH . . . 269

10.3 SFTP and SCP 272
10.4 Homework 277

11 Symbolic Math and Numerical Linear Algebra **279**
11.1 Symbolic Math and Numerical Linear Algebra 279
11.2 Linear Algebra 288
11.3 Homework 295

Contents:

CHAPTER

ONE

INTRODUCTION TO SCIENTIFIC COMPUTING

1.1 What is *Scientific Computing*

In this course we will learn some of the basic techiniques involved in using your computer to do scientific tasks. Since our course has students with different experience using computers, some tasks will be familiar (I hope to start out that way), and some will not. Along the way, I hope to add a few simple applications to your computer that will help you be a better scientist, engineer, mathematician, or philosopher, astronomer, accountant, of whatever it is that you do. The tools (software) and techniques that you use when doing science with your computer, are somewhat different than those that you use when you use your computer for doing other, non-scientific, tasks. Let's examine the differences between these two ways of using your computer. That will help us outline the things that we will study in this course.

1.1.1 Non-scientific computing:

While you might do several of the following things in the course of some scientific project, these computer activities would not be considered scientific computing

- Email
- Surfing the Internet
- Word Processing (non-scientific)
- Organizing and saving information in documents (files).
- Using a spreadsheet for bookeeping or tabulating things.
- Listening to / downloading / making music
- Editing movies
- Storing pictures
- Gaming

1.1.2 Scientific Computing

When using your computer for science, you do many different things. These might include

Scientific Computing Activities

- **Analyzing data, for example:**
  - doing **statistics** on data to compare with theories
  - **curve fitting** to find mathematical relations in data
- Graphing data and mathematical functions: **scientific visualization**
- Doing **algebra and calculus using the computer**: symbolic mathematics
- **Numerical calculations:**
  - computing **numerical approximations** to mathematical quantites like definate integrals (often called numerical quadrature)
  - **linear algebra** (vector and matrix computations)
  - solving systems of linear equations
  - solving **non-linear equations**
  - and much more... (like computational fluid dynamics and lattice quantum chromodynamics)
- **Scientific publication**
- **Scientific programming**

There are lots more, not covered in this course, such as:

- *Machine Learning*
- *Recording data from experiments*
- *Solving differential equations*
- *Computational Chemistry*
- *Computational Fluid Dynamaics*
- *Lattice Quantum Chromodynamics*

It is these types of scientific tasks that we will cover in this course (except the last item). Many of these topics are the subject of Ph.D. theses, and one could devote decades to learning how to do them well. Our goal is much simpler.

Our Goal in this course

To introduce you to some of these topics in order to get you started, and
To outfit your computer with the free tools needed to do so.

With just this, and a little diligence (i.e. webhunting for examples and tutorials) you'll be surprised how much more useful your computer can be.

First, we must setup a scientific computing environment on your computer. This simply means getting applications to do these things.

1.2 The Scientific Computing Environment

By scientific computing environment , we mean the hardware, software, operating system, and customary tools with which you will carry out the computations described in the previous page. Thus the environment can be quite extensive. However for our purposes it is possible to set up a very nice scientific computing environment on a personal computer.

1.2.1 Scientific Computing Hardware

There is a wide range of machines upon which good science is done. These range from graphing calculators up to multi-million dollar computing facilities which house the fastest and most advanced computers in the world.

Let's first look at perhaps the simplest scientific computing environment, the handheld graphing calculator. If you have a TI-8X (a Texas Instruments 82, 83, 84, or 86, etc.) calculator, you can do a significant number of the items described under scientific computing tasks on the previous page. As an example, and particularly if you yourself have one of these little pocket pals, visit

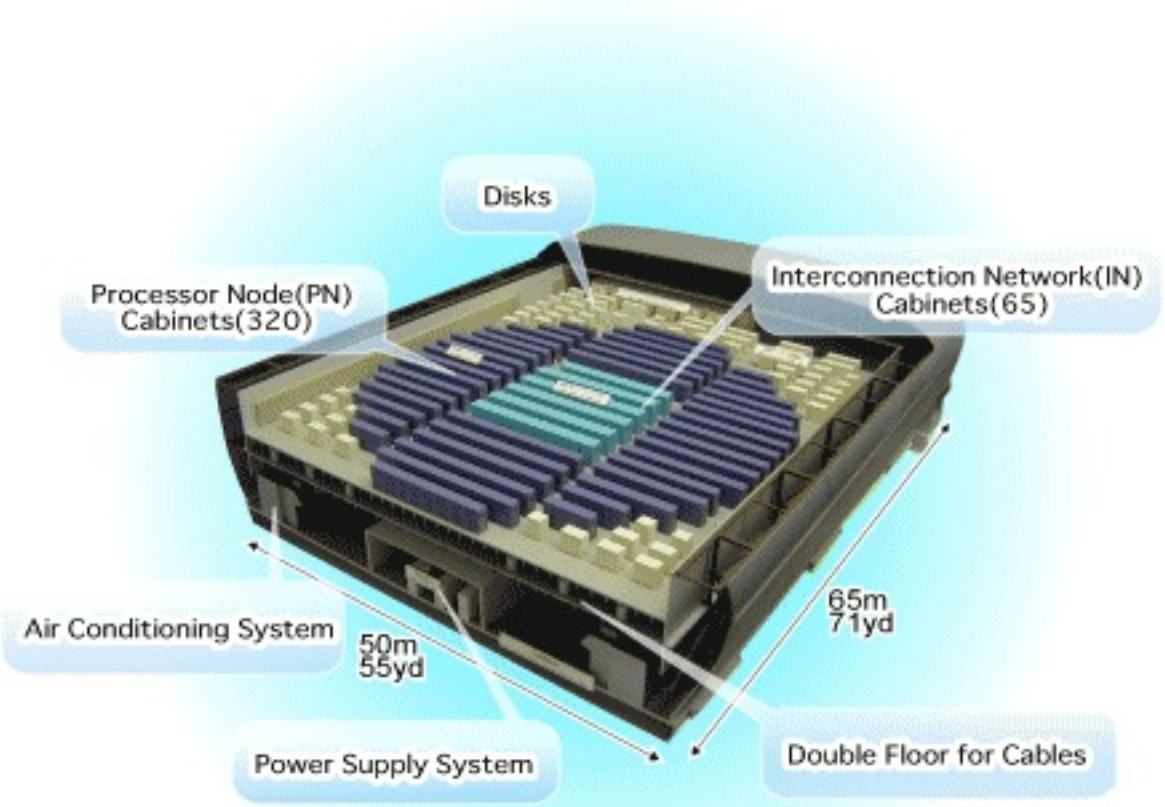
The Quick and Dirty TI Guide to see a list of these features. For example, my TI-86 does:

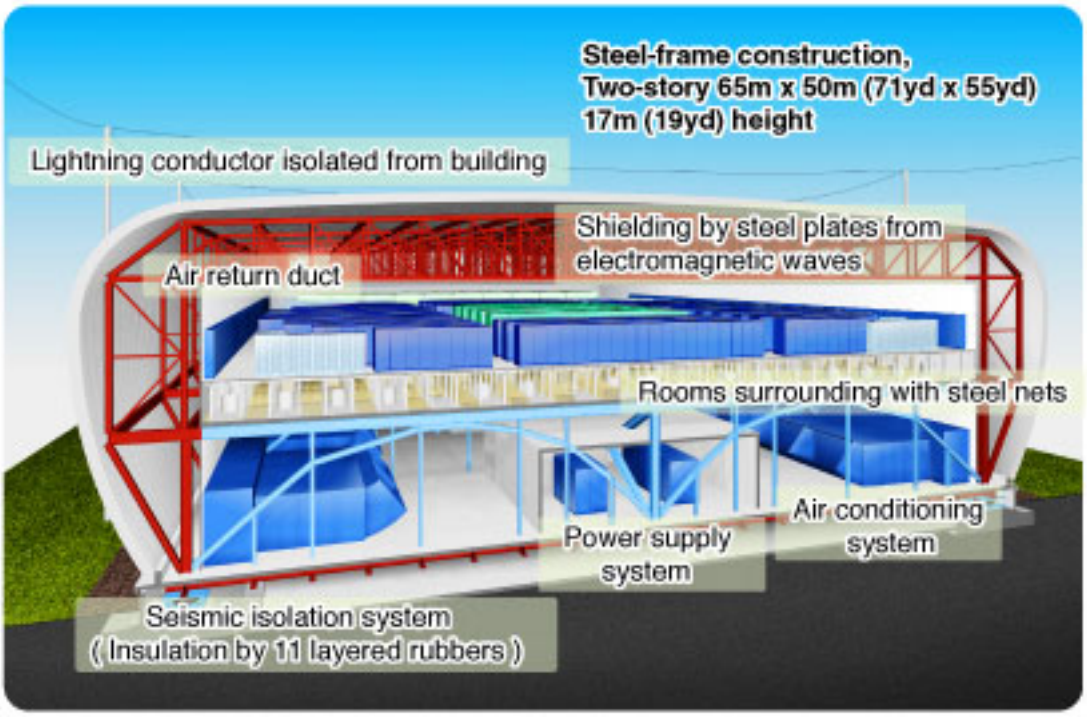
- Data visualization
- Curve fitting
- Linear algebra (solving several linear equations)
- Statistical computations
- Scientific programming

This is pretty good for something that fits in your pocket or hangs from your belt! Now, of course, you can get apps for your phone that do much of this too.

High Performance Computing

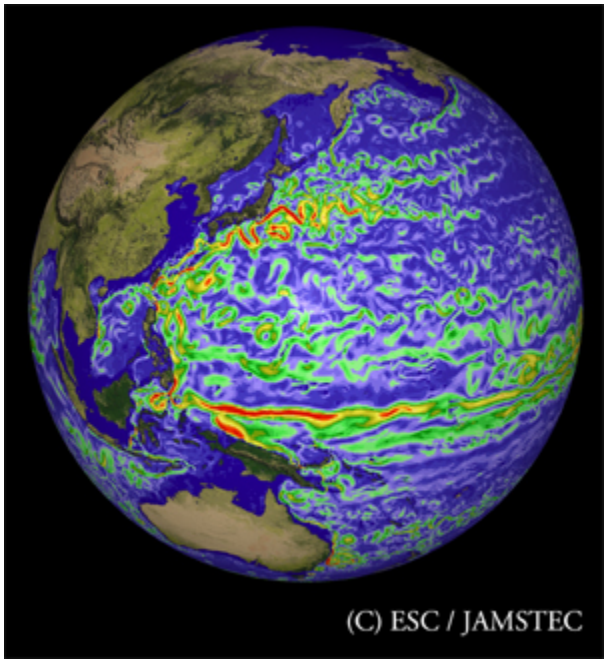
At the other end, consider the Earth Simulator, one of the world's fastest computers (back in 2005) located in Japan.


Disks
Processor Node(PN)
Cabinets(320)
Interconnection Network(IN)
Cabinets(65)
Air Conditioning System
50m
55yd
65m
71yd
Power Supply System
Double Floor for Cables

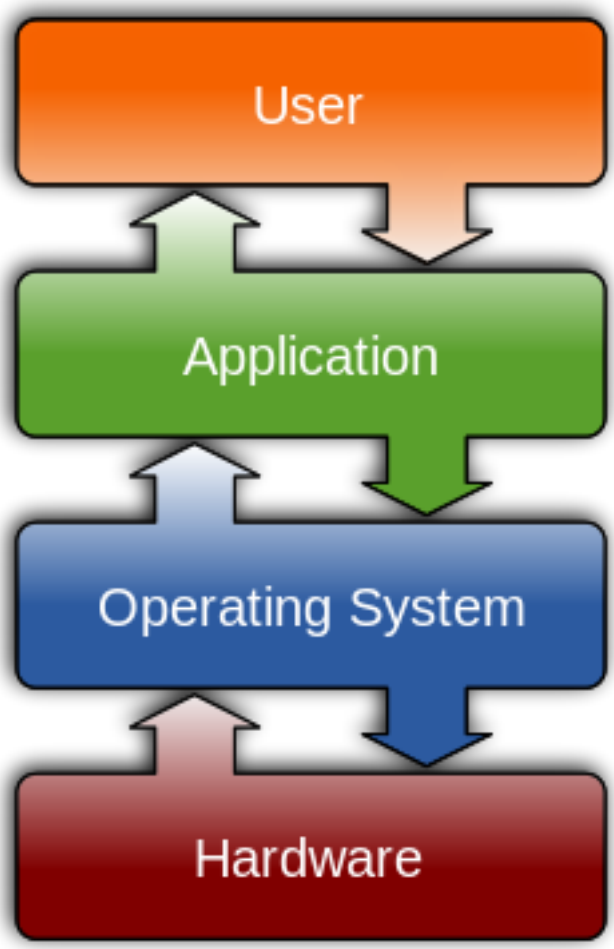

Steel-frame construction,
Two-story 65m x 50m (71yd x 55yd)
17m (19yd) height
Lightning conductor isolated from building
Shielding by steel plates from
electromagnetic waves
Air return duct
Rooms surrounding with steel nets
Air conditioning
system
Power supply
system
Seismic isolation system
(Insulation by 11 layered rubbers)

This computer takes up an entire two story building that is a bit smaller than a football field. The bottom floor is cooling, power, and cabling, and the top floor houses thousands of CPUs wired together to work as one single system. As its name implies, the Earth Simulator was built for global climate modeling.

An example visualization of some of its output is shown below, an El Nino event:



1.3 The OS and Unix



The **Operating System** is the system of interacting programs which provides the interface between the " appications " such as Word, Excel, Safari, iTunes, etc. (also programs) that the user wants to run, and the hardware of the computer, as shown in the figure below. The hardware encompasses many different parts, of course. There are the user interface devices: the mouse, keyboard, screen, and sometimes others, as well as the computer's memory, CPU, data buses, etc. We don't need to worry about these complicated low level hardware devices, because the OS controls them in a convenient way for us. For example, if you have an iPad or iPhone, you probably use the **iOS** OS. That OS has no mouse interface, but does rely heavily on touch- screen input.

While there are many different OS's, you are likely familiar with the top three: *Microsoft*'s **Windows** , *Apple*'s **OSX** , and possibly you have heard of **Linux**–a version of *Unix* .

1.3.1 Unix: The Scientific Operating System

THE operating system used at **ALL** supercomputing centers around the world is some variant of Unix . While there are some recent experimental machines at certain supercomputing centers which are testing new ideas (like clusters of NVidia Graphics/Gaming Cards– called GPUs), all high performance computer scientific systems run Unix. Why?

Advantages and Disadvantages of Unix From: **'http://pangea.stanford.edu/computerinfo/unix/overview/advantages.html'_**

Advantages

- Full multitasking with protected memory. Multiple users can run

multiple programs, each at the same time without interfering with each other or crashing the system. + Very efficient virtual memory; many programs can run with a modest amount of physical memory. + Access controls and security. All users must be authenticated by a valid account and password to use the system at all. All files are owned by particular accounts. The owner can decide whether others have read or write access to her files. + A rich set of small commands and utilities that do specific tasks well – not cluttered up with lots of special options. Unix is a well- stocked toolbox (with amazing gadgets), not a giant menu-driven Swiss Army Knife. + Ability to string commands and utilities together in unlimited ways to accomplish even more complicated tasks – not limited to preconfigured combinations or menus, as in personal computer systems. + A powerfully unified file system. Everything is a file: data, programs, and all physical devices. Entire file system appears as a single large tree of nested directories, regardless of how many different physical devices (disks) are included. + A lean kernel that does the basics for you but doesn't get in the way when you try to do the unusual. + Available on a very wide variety of machines - the most portable operating system. + **Optimized for program development and file manipulation, and thus

for the unusual and new circumstances that are the rule in scientific research.**

Disadvantages

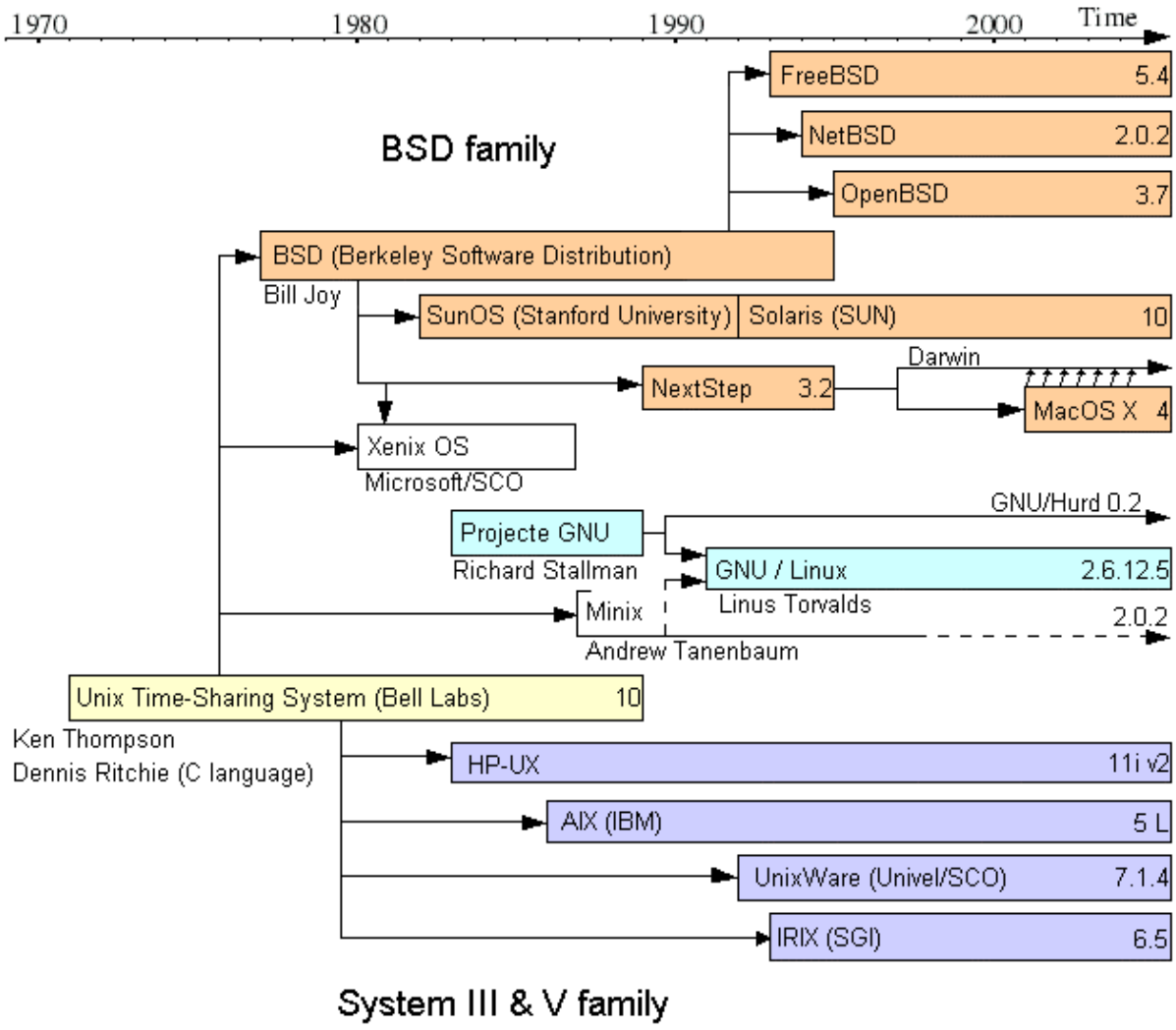
- The traditional command line shell interface is designed for the

programmer, not the casual user. + Commands often have cryptic names and give very little response to tell the user what they are doing. Much use of special keyboard characters - little typos have unexpected results. + To use Unix well, you need to understand some of the main design features. Its power comes from knowing how to make commands and programs interact with each other, not just from treating each as a fixed black box. + Richness of utilities (over 400 standard ones) often overwhelms

novices. Documentation is short on examples and tutorials to help you figure out how to use the many tools provided to accomplish various kinds of tasks.

So, What is Unix? The unix operating system was developed before both Windows and Mac OS for large computers called "Mainframes" of the 1970s (i.e. " back in the Day!). One interacted with these machines by "logging on" via a text terminal, usually in another office, at a place like a bank or an airport where there were {it many people working simultaneously}. These mainframes managed the simultaneous users–they were the original " servers ". Thus the unix operating system, developed at Bell Laboratories, was originally designed to be **"'multitasking"'**_ from the beginning, something that personal computers only developed in the very late 1980s, after the appearance of the 80386 Intel processor (this is the precursor to the Pentium, the 80586). A cpu that can multi-task requires some advanced architecture. It must be able to be interupted from it's current task, save the state of that task, and start working on someone else's job. Also in the 1980s, unix workstations became quite popular among scientists and engineers, and a number of tools were developed for to aid scientists in their work. Many different computer vendors such as IBM, Sun Microsystems, Silicon Graphics, and Hewlett-Packard have developed their own forms of unix to run on their specific hardware. These versions of unix go by the names: AIX, Solaris [1], Ultrix, and HPUX. The development history, evolution, and relationship of these different versions is shown below.

[1] I've always thought "Solaris" was an odd name for an operating system. It originates in the classic scifi novel "Solaris" by Stanislaw Lem, where a sentient ocean covering a distant planet drives explorers mad who are sent to study it.

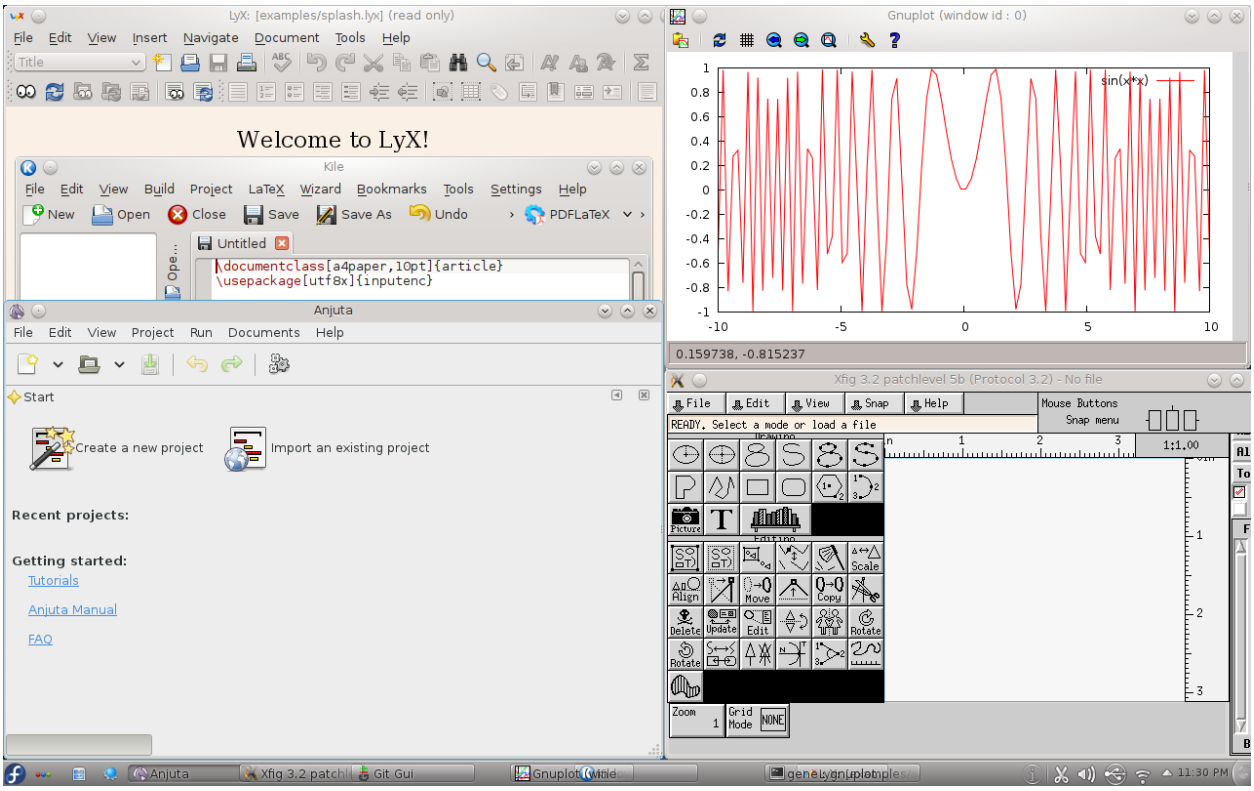


1.3.2 Unix running natively on a PC: = *Linux*

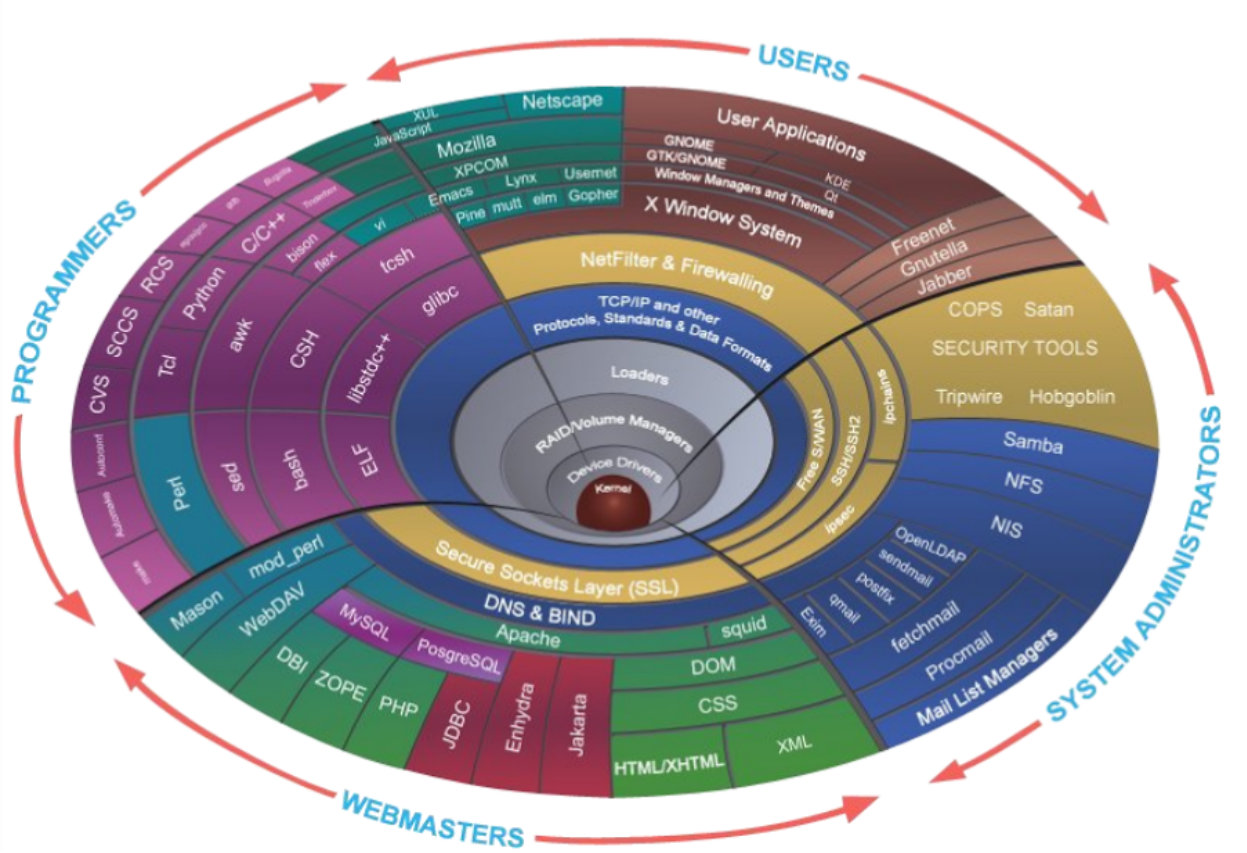
With the introduction of the 80386 CPU chip by Intel in the 1980s, PCs became able to multitask allowing the development of a form of unix which would run on the PC architecture. This version was developed by Linus Torvalds, a Finnish graduate student and the GNU Free Software Foundation developed by Richard Stallman. The name of this version is Linux which is extremely popular, powerful, and *FREE*

For this course, I am recommending the Fedora Scientific Spin of Linux, running on a Virtual Machine on your computer.

A screenshot is below.


LyX: [examples/splash.lyx] (read only)
File Edit View Insert Navigate Document Tools Help
Welcome to LyX!
Kile
File Edit View Build Project LaTeX Wizard Bookmarks Tools Settings Help
New Open Close Save Save As Undo PDFLaTeX
Untitled
\documentclass[a4paper,10pt]{article}
\usepackage[utf8x]{inputenc}
Anjuta
File Edit View Project Run Documents Help
Start
Create a new project
Import an existing project
Recent projects:
Getting started:
Tutorials
Anjuta Manual
FAQ
Gnuplot (window id : 0)
sin(x*x)
0.159738, -0.815237
Xfig 3.2 patchlevel 5b (Protocol 3.2) - No file
File Edit View Snap Help
Mouse Buttons
Snap menu
READY, Select a mode or load a file
Zoom
Grid Mode
NONE

1.4 The Scientific Software Stack



Our Scientific Computing Environment will consist of a number of programs which work together to allow us to carry out the activities **'Scientific Computing'_** in Section 1. You might download them separately (I call this the "à la carte" method), or as a large package. In addition, the list usually changes from year to year as new packages appear that do the job better than what we used previously.

In some Environments–*like at supercomputing centers*–the stack includes several packages to do a specific task and users can choose the one that suits their needs the best, or the one which they are most familiar with. Also, supercomputing centers typically have a variety of machines designed for specific tasks. You use one for high performance number crunching, and another system for 3-D data visualization.

For this course I have chosen a standard set of applications/programs, which I list below. I have set the color of the cells in this table to light red, like this: to show items that are **Built-in** to the OS or come with the installation.

	Fedora Scientific	Windows PC	Mac OSX
Unix Shell	***Built-in***: • Bash	• Cygwin • MSYS • Git Bash	***Built-in***: • Zsh
Text Editor	***Built-in***: • Vim • Emacs • Eclipse • SciTE	• Notepad++ • Others...	• Emacs via MacPorts • Kate via MacPorts • TextWrangler
Data Visualization	***Built-in***: • Gnuplot • MatPlotlib (in Python) • Mayavi	• Gnuplot • Matlab (U. Pacific Lic.) • AVS ($) • Others... ($)	• Gnuplot (via MacPorts)
Scientific Publishing	***Built-in:** • LaTeX • Kile Editor	• MikTex	• MacTex
Drawing/CAD	***Built-in***: • Xfig • Inkscape • Dia • GIMP	• WinFig ($) • Inkscape • Dia • GIMP • SolidWorks ($)	• WinFig ($) • Inkscape • Dia • GIMP
Remote Computing	***Built-in***: • SSH • SFTP • SCP • CURL	• Putty • WinSCP	***Built-in***: • SSH • SFTP • SCP
Math / Science	***Built-in***: • Octave • Python • Sage • R • SciPy • Root • Units • Maxima • Mathematic (by remote login at U. Pacific)	• Mathematica ($) or by remote login • Python tools via Anaconda • R • Octave	• Mathematica ($) or by remote login • Python tools via Anaconda • R • Octave
Programming	***Built-in***: • C/C++ • Fortran • Java • Python • iPython • PERL **Distributed Computing** • OpenMP	• C/C++/Python via Cygwin • Python via Anaconda • PERL via ActivePerl	• C/C++ via gcc/MacPorts • Python via Anaconda

Most of these tools are available a la carte i.e. as individual applications; you have to install each separately. While that is certainly possible, it takes some effort and the individual applications are not setup to work together effectively.

If you use a PC, the **'Cygwin'_** application installs a suite of programs which emulate a Unix environment under the Windows OS. However, it is not a real Unix operating system, only a set of Windows programs which make it possible to build other unix programs and allow them to run under Windows.

If you use a Mac, the underlying OS actually IS Unix. This is why the first item, the Unix Shell, is built-in. You can access it by opening Finder->Applications->Utilities, and clicking on Terminal . This opens a text window, with a prompt (more on this in the next chapter), waiting for you to enter Unix commands.

Notice all the *Built-in*'s in the first column under Fedora Linux/Scientific Spin! The simplest way to get all the programs needed for a robust scientific computing environment would be to get a real Unix system running the Scientific Spin of the Fedora Linux OS. Otherwise, you have to install all of the programs individually (" à la carte "). There are a number of ways to get Linux on your computer.

- **Erase your current Windows or Mac OS**, Reformat your hard drive,

and install Fedora Linux as your operating system. Yes–pretty extreme. But if you are really committed... Linux gets your whole computer; all your hard drive space, and when running all your memory and CPU speed.

- **Dual Boot** This method installs the Linux OS on a separate

partition of your hard drive. Thus your Windows (or Mac) OS looses some hard drive space to the new partition. When you turn on your computer, you are asked if you want to boot into Windows (or OSX), or Linux. Once chosen, the OS gets all of the CPU and memory of the machine, and the whole of the disk partition belonging to that OS.

- **A Virtual Machine running Linux** The Best of Both Worlds *AT THE SAME TIME*!

From the Wikipedia page on Virtual Machines (please read the introduction):

```
"a virtual machine (VM) is an emulation of a particular computer system"
```

The Virtual Machine software interfaces with your computer's hardware at a low level, and then provides an environment which looks like a clean and blank computer. The software can create several such VM's, each blank and ready to be loaded with whatever software you like–AND they can all run at the same time. You configure the size of the disk drive for each, the amount of memory, and several other configuration options. When you run the VM software, you can start each of the virtual machines you have configured.

Thus on your Windows machine, you could have two VM's running (each in a window of their own), one a complete Linux computer, and the other running OSX. It is important to note that these running machines will take their resources (CPU speed, memory, and disk) from the actual physical resources in your computer. Thus if your Windows computer has 8 Gigabytes of memory, and you asign 4 Gb or memory to your "guest" Linux VM, then your "host" Windows computer will have only 4 Gb for itself. However, unless you are pushing either of your computers to their limit (such as gaming, streaming video, or doing significant numerical computation), then you won't really notice the fact that your physical computer is being shared by your native OS and your VM. Magic! There are several popular Virtual Machine programs (see: this'article'_).

One of the most popular and FREE Virtual Machine applications is'VirtualBox'_ from Oracle. VirtualBox runs on Windows, Macs, and Linux "hosts". This means, you could even take the first option–erase your OS, install Linux as the primary OS, then run Windows in a VM as a guest. However, most of you will be running either Windows on a PC or OSX on a Mac. Thus, for this class, I am recommending that you install the appropriate VirtualBox package for your computer, then you will configure a VM and install the Scientific Spin of Fedora Linux on it. You can then run Linux and the scientific tools on your computer.

Whether you have a Mac or a PC won't matter, since in either case you will be running Linux. VirtualBox even comes with some cool features, like Seamless Mode which runs the Unix programs in individual windows so that they look and feel just like other native programs. However, with this method, you have a real Unix kernel running, so the Unix applications are full-featured. In the next chapter, we will assemble these tools and install them.

1.5 Homework

Homework 1 is here

CHAPTER

TWO

INSTALLING YOUR SCIENTIFIC COMPUTING ENVIRONMENT

2.1 Overview

After about 8 years of experience teaching this course with Cygwin as the primary environment for PC users and *fink*, then Macports for Mac users, I have finally decided to move to a unified platform.

This has been made possible by the emergence of **Virtualization Technology**, which allows one to use the hardware on a single machine and make several *Virtual Machines*. Each Virtual Machine (**VM**), while it is running, shares the CPU, memory, hard drive, and other components of the *real* hardware.

Since there is a robust *FREE* Virtual Machine application, and a *FREE* Unix operating system which comes with all the scientific tools (open-source, of course) we need, *AND* this system looks the same whether you have a PC or a Mac, it's like finding the Unified Field Theory that incorporates general relativity and quantum field theory. Well, almost...

So, we have a simple set of steps ahead of us.

- First we install **VirtualBox** from *Oracle*

  This is pretty easy; download the package and click *Next* a bunch of times as it installs.

- Then we *create a Virtual Machine* on which we will do our science

  This is easy too, although you might have to go into your BIOS and make a couple adjustments if you want to use the full power of your 64-bit computer.

  This leaves you with a new-born baby (virtual) machine. No operating system, no files, nothing.

- Now we install **Fedora Linux–Scientific Spin**

  Bang! We download the Fedora ISO, put the virtual DVD in the virtual DVD slot, and reboot. Answer a couple questions, make *root* and *user* accounts, click *Install*, go have coffee. When you finish your cup-a-joe, you reboot your VM and you now have a brand new computer asking you to login.

- Login and look around. Then **Configure a few helpful items**.

  We will add the *Guest Additions*, programs which integrate your new VM with your "*host*" OS (i.e. your PC or your Mac). This gives you "*Seamless*" integration of the two operating systems, as if all features of both were built into one.

You will also

- Set up **Shared Folders**

  so that you can easily access files across your two machines: your original computer that your Mom and Dad bought you <–> and your new VM that you just built from stuff you downloaded from the Internet.

Ready?

2.2 Prepare to Install

Before we actually install our Scientific Computing Environment and start learning to use it, it's very helpful to get an overview of what we are going to do.

Here are the steps. You are going to do the first 2 steps now .

2.2.1 Find out what kind of hardware you have

32 or 64 bit

You will need to know if your computer is a Mac or a PC (DUH!), and if it is a **32- or 64-bit** machine. Most computers purchased since 2012 will be 64-bit, but you will want to check to be sure.

How much Memory?

Next you will need to know how much memory (usually between 2 and 16 Gigabytes) your computer has. You can see both of these by doing the following.

On a PC, *Click the Start globe* at the lower left, on the menu that opens *Right- Click Computer* and select *Properties*. That is: *Start->(Right Click) Computer->Properties*. This opens an info window, where you can see your data width 32- or 64-bit, and the amount of installed memory.

On a Mac: *Click the Apple logo* at the upper left. On the menu that drops down, select About this Mac . From the info window, you can see how much memory you have. Unless you have a very old Mac, you have a 64-bit data bus.

Finally, you need to know how much **free diskspace** you have. You won't need a whole lot, but you should have at least 10 Gb free. Linux will only use what it needs.

On a PC: Click *Start->Computer*. Look at your `C:` drive; You will see a bar graph showing the total size and amount used. Below this, the amount of free space is shown, 125 Gb free out of 297 Gb, in the case displayed below.

On a Mac: Click *Apple->About this Mac*. Next select *More Info...*. On the window that pops up, select *Storage*. Find your `Macintosh HD`, and notice the amount of free disk space above the bar graph.

Again, here is the list of the information you need. Write down what you find.

- Data bus width: 32- or 64-bit?
- Amount of Memory
- Disk size and free space

2.2.2 Downloading the Appropriate Software Items

- You will need to download the **VirtualBox Installer** appropriate for your computer

  Visit Oracle: Download VirtualBox , select the appropriate one, and save it to your computer.

  - If you have a **PC**, Download: VirtualBox for **Windows** Hosts (click the *x86/amd64* link)
  - If you have a **Mac**, Download: VirtualBox for **OS X** Hosts (click the *x86/amd64* link)

- Next download The **Fedora Linux - Scientific** ISO file This is a large file (about 3.5 Gb), and thus it will take some time to download.

Note: If you know you have an old 32-bit PC, or are comfortable with Torrent Downloads use one of the other links

But for most people, you will want to get the **64-bit PC Direct Download** file from: https://labs.fedoraproject.org/scientific/download/index.html

If you have trouble, let me know. I want you to get the latest version, but I can put a copy on the Physics Department server which may download faster for you.

Again, see me if you have trouble getting one of these. I can burn you a copy.

That's it! All the scientific software you need is Built-in to Fedora Scientific Linux.

Do these steps now!

Before your go to the next section: *Install...*, you need to have done the steps above:

- Check and record your computer hardware info
- Download VirtualBox ,
- Download https://labs.fedoraproject.org/scientific/download/index.html

2.2.3 Next Steps

After these first 2 steps, you will carry out the following steps. Read through these so that you understand what is ahead.

1. **Install VirtualBox**
2. **Create and configure a virtual machine (VM)**

   on which you will install Linux.
3. **Start the VM with the Fedora ISO in its virtual DVD drive**.
4. **Go through the Linux Installation proceedure on the VM**.

   You will do some minor configuration steps, such as selecting your language preference, then you will choose a root password and make a standard user account for yourself.

5. **Eject the Fedora Installation ISO** file,

   and reboot the VM**. This will start your fresh new Linux VM. You can login to your user account and look around.

6. **Add "Guest Addition" features** to your Linux VM.

   By adding the included " Guest Additions " to your Linux VM, you can have features such as Seamless Mode (your Linux windows appear on your Windows or Mac desktop just like other programs) and more. Other features include:

   - Shared Folders and Clipboard, between host and guest machines
   - "Seamless" Desktop integration with the Host OS

7. **Reboot with Guest Additions activated**
8. **Configure the Unix behavior and Desktop** "*Look-and-Feel*" to your liking.
9. **Start using your Scientific Computing system**!

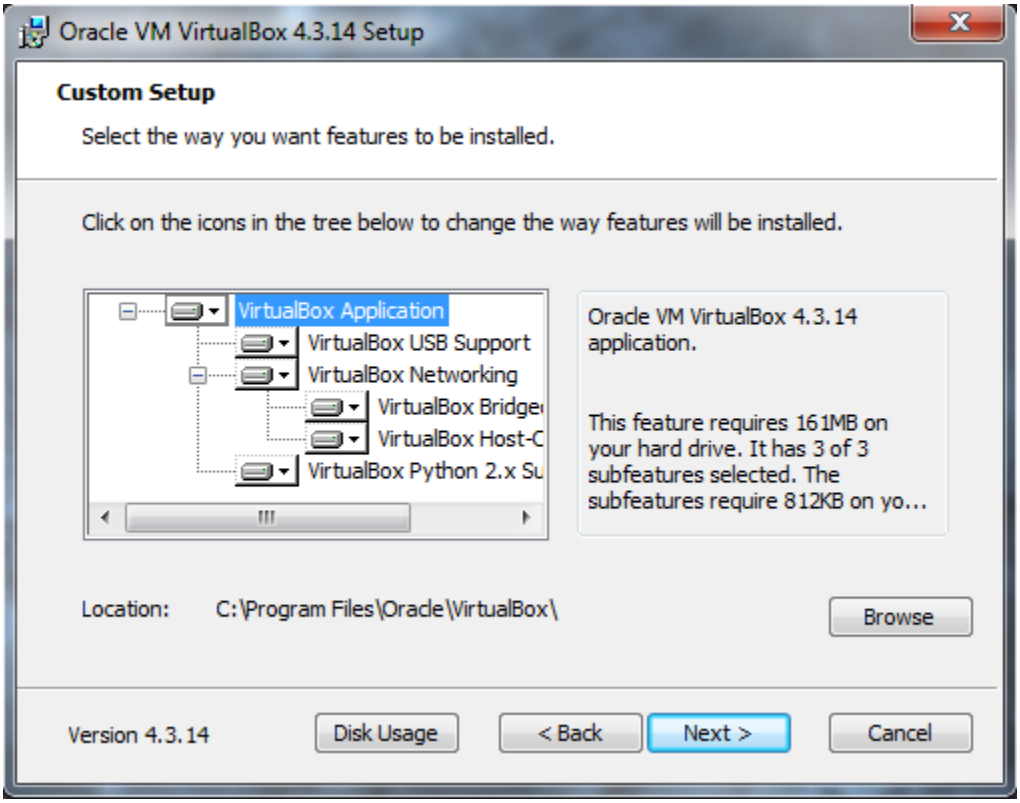
2.3 Install VirtualBox

Having gathered the information about your computer in the last section *Find out what kind of hardware you have*, and downloaded the software files *software*, we are now ready to start installing our scientific system.

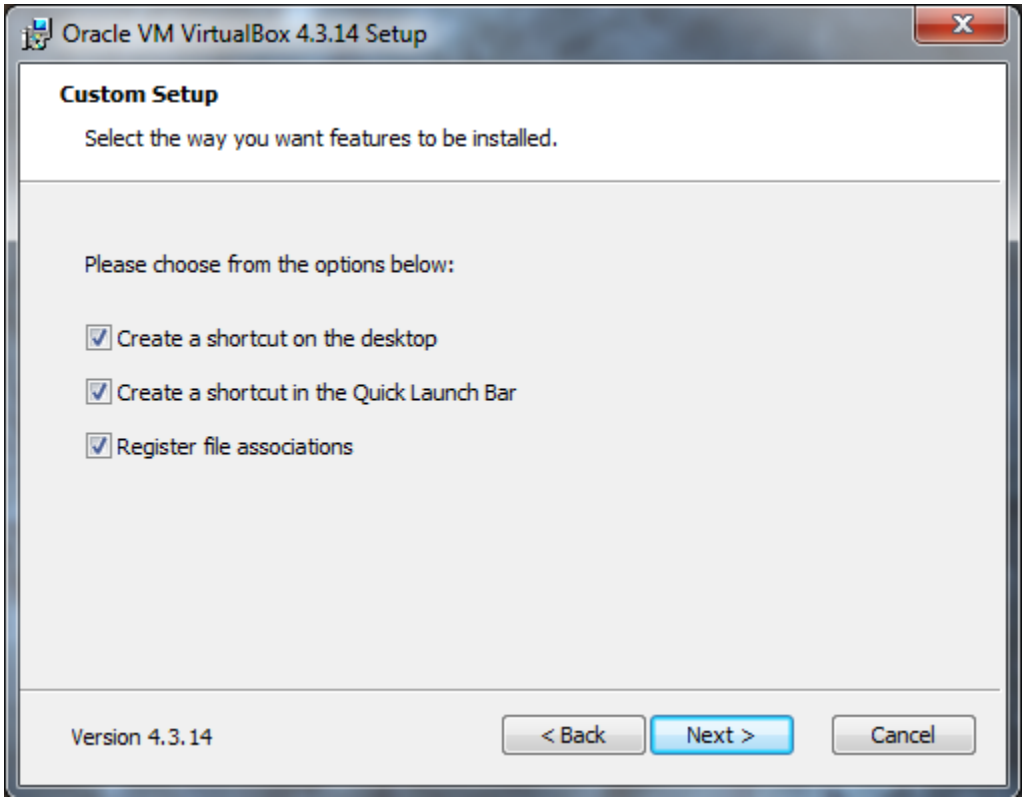
To begin, locate the **VirtualBox Installer** you downloaded, and **Click** (maybe twice?) to start it.

This will start the installer.

Naturally, you will click **Next**.



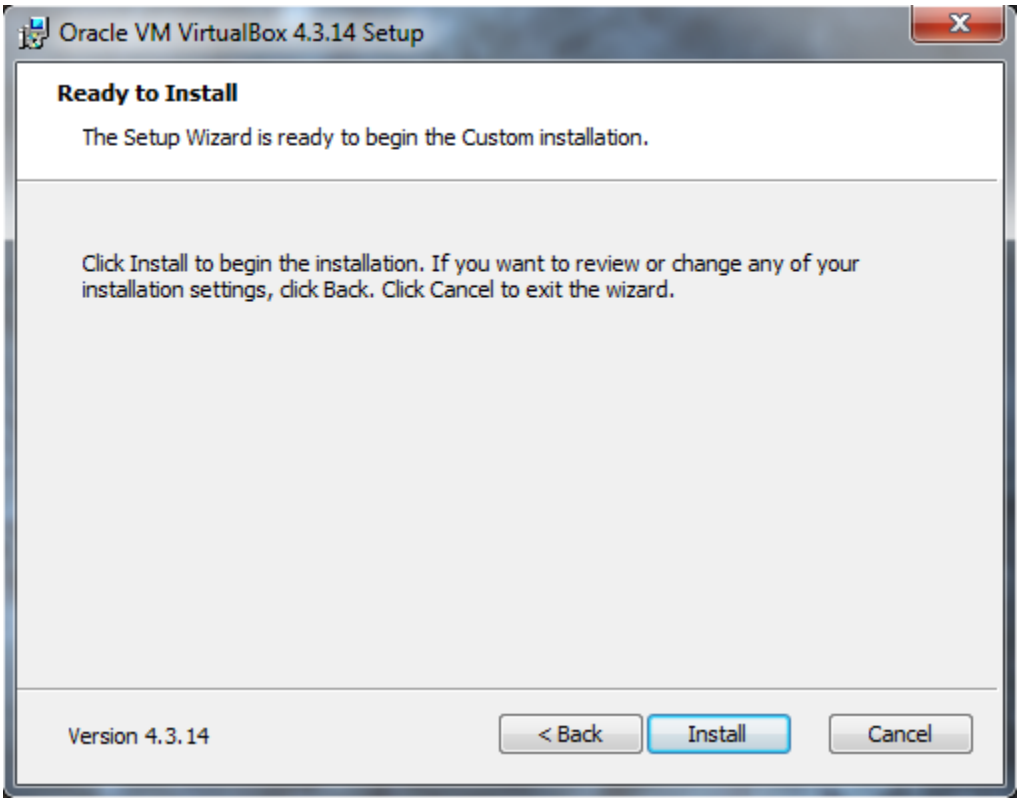
Then **Next** again, to accept the installation of the standard package



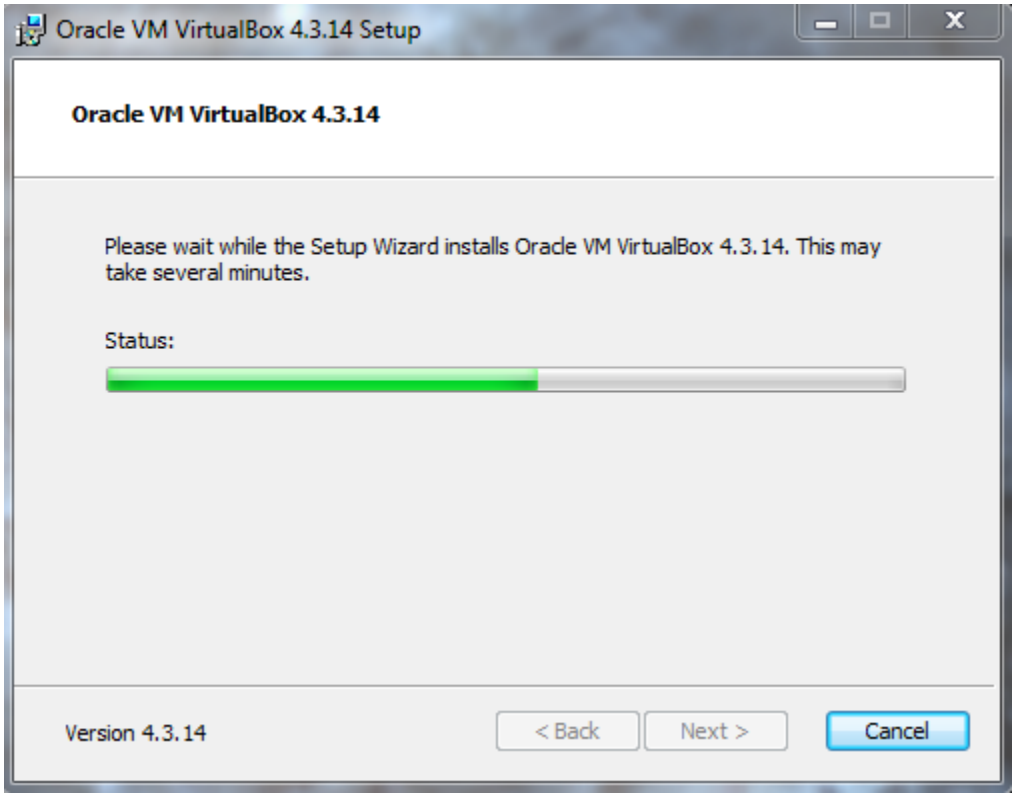
Here you can choose which things you like: an Icon on your desktop, an Icon on your launch bar, or not.

Note: It is important to leave "*Register file associations*" **CHECKED**.

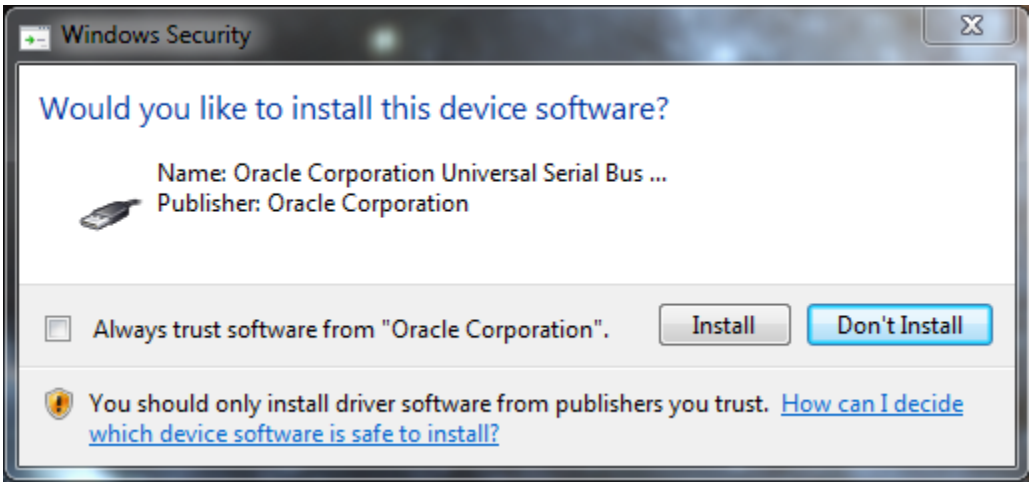
The installation will temporarily disconnect your network connection, but will then reconnect it. Click **Yes**.



Now we are ready to install. So click **Install** !

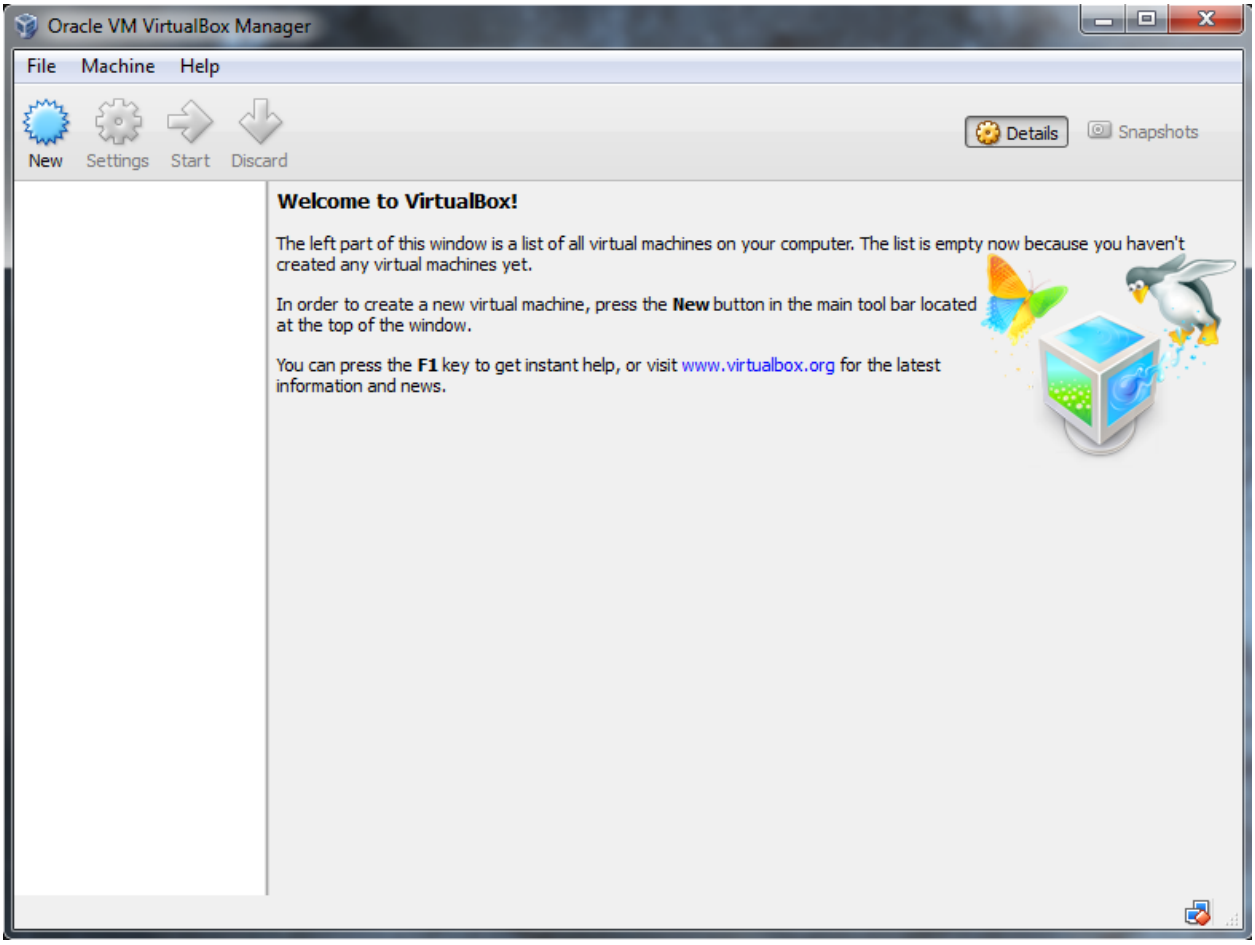


VirtualBox will show you the status of the installations.



It MAY pop-up a screen like this asking if you Trust Content from Oracle (perhaps a few times). If so, click **Install** and continue. If you want to, you can check the Always trust software from "Oracle Corporation" box.

Success! Now click **Finish**. VirtualBox will launch, and you should see the screen below.



Warning: **If you do not ** this screen, somthing went wrong. Probably it's a little thing. Contact Prof. Hetrick. If you need to do so, please include as much information about the error as you can. Screenshots would be appropriate!

Next we will **create a VM** and **install Fedora Scientific Linux** on it.

2.4 Creating a Virtual Machine (VM)

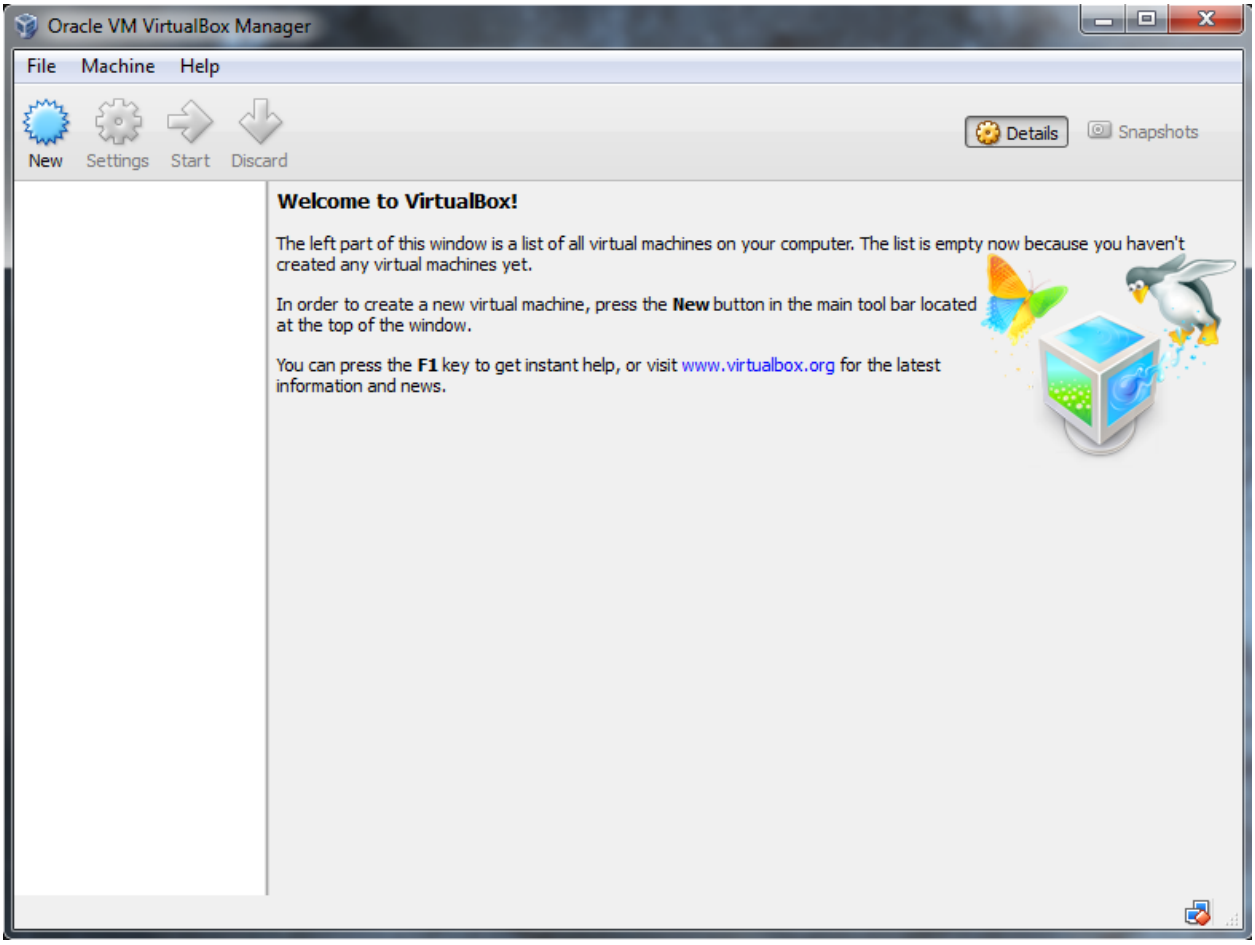
2.4.1 Video for this section

The video for this section is here (YouTube).

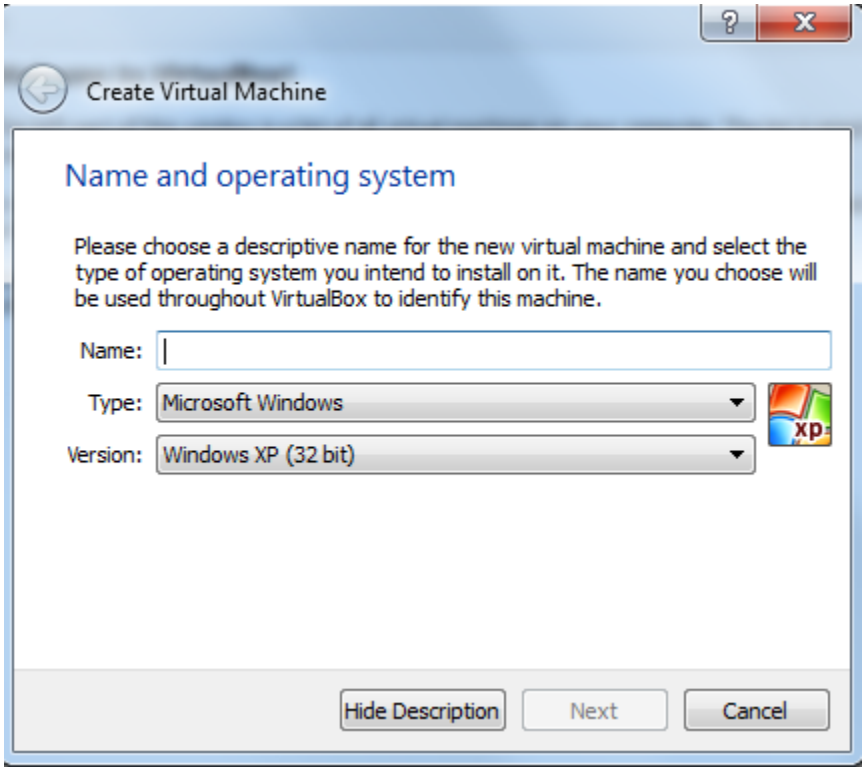
2.5 Defining the VM

Now you are ready to create a virtual clone of your computer hardware and install a Linux Operating System on the clone. Pretty trippy if you think about it.

- First, **start VirtualBox**. You should see this window.



- Click the ” **New**” button.



This brings up the VM building window:

You can see

- a text box for you to give a name to your VM (say, "*Fedora Scientific*")
- a dropdown menu for the "*Type*" of OS: Windows, Linux, Mac, etc.
- a dropdown menu for the "*Version*" of the OS: Windows XP, Win7, Fedora Linux, Ubuntu Linux, etc.

Before proceeding, let's check something first.

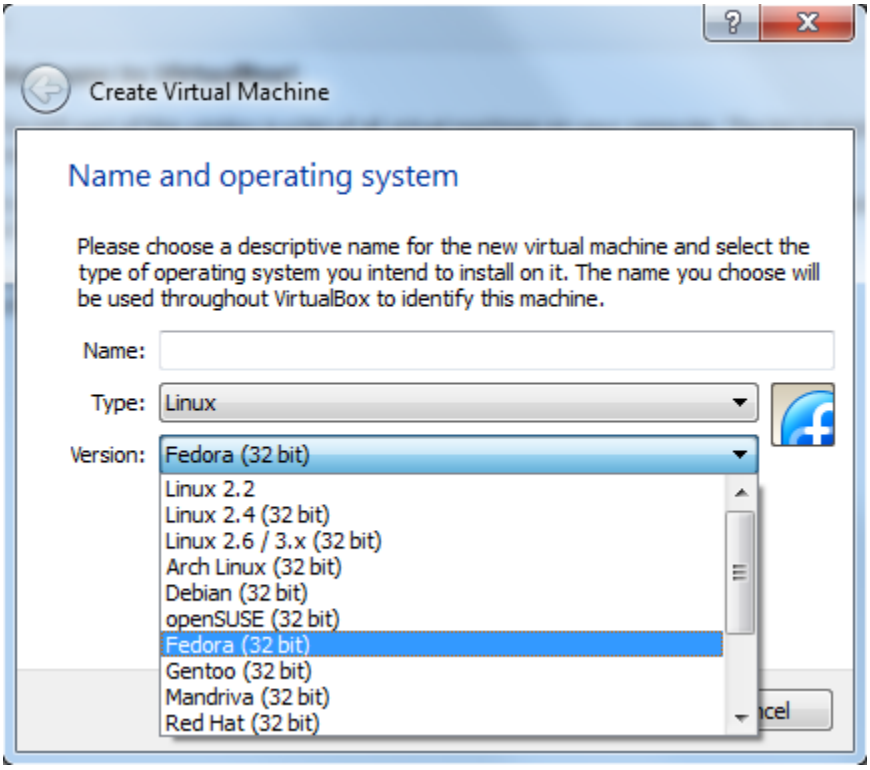
Remember your *Preperations* *Find out what kind of hardware you have*? You found if you have a *32-bit* or *64-bit* computer.

If you have a *32-bit* machine, then you can skip the following BIOS alterations, and just jump to here: *config-the-vm*.

However, if you have a 64-bit machine, you will want to take full advantage of that big fat data bus. Check the following.

2.6 Turn on Virtualization in your BIOS?

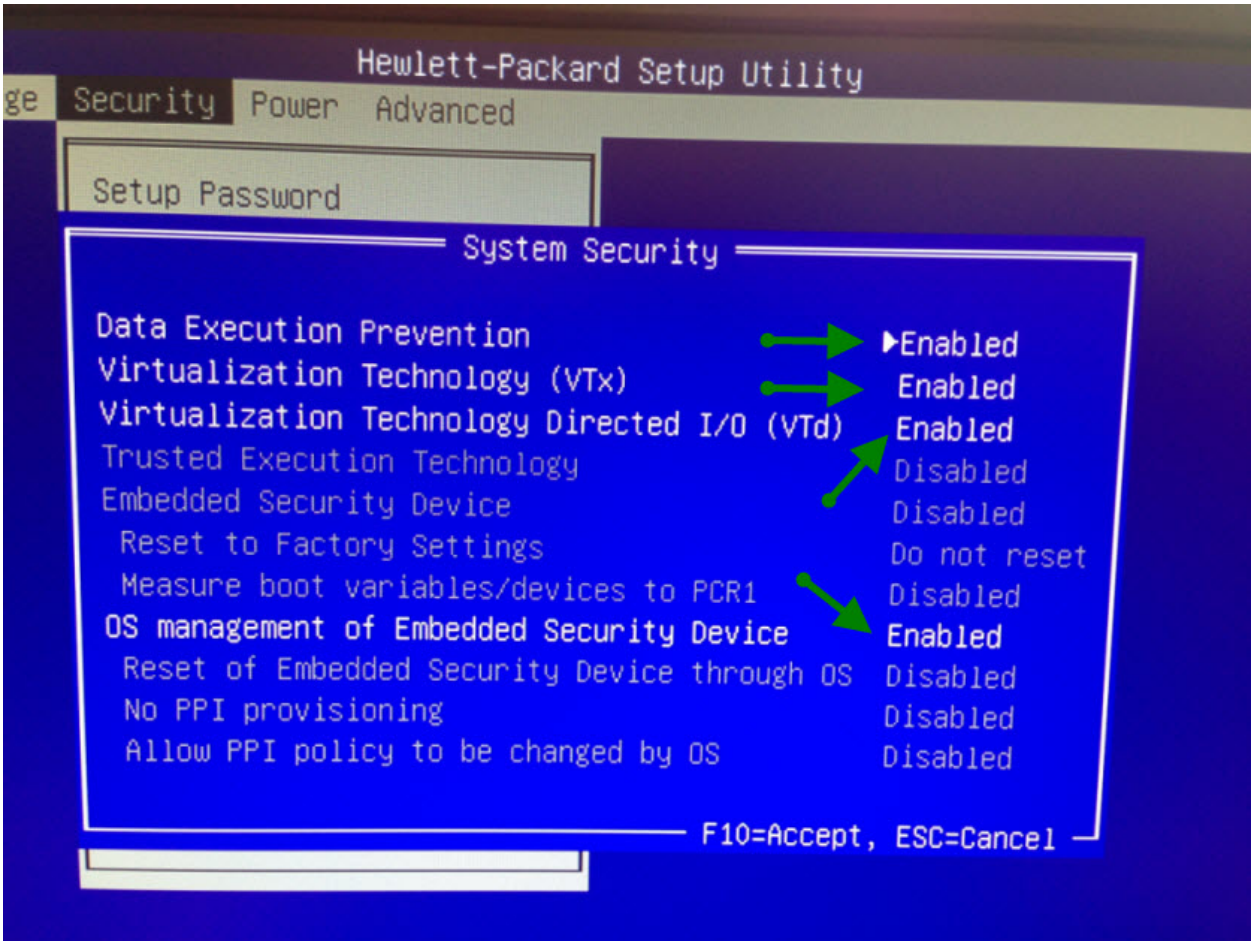
- Click on " **Type**", and select " **Linux**"
- Click on the " **Version**" menu, and look at the options.
- Do you see only ** 32 bit ** choices, i.e. no **64 bit** choices? Like this:



If so, then your computer has " Hardware Virtualization " turned OFF. You can turn it ON (or try to, at least), by doing the following:

- **Reboot** your computer
- **Halt the boot process and enter your BIOS system**

  This is usually done by hitting an **F-Key** during the boot process (Often your system says somthing like "*Hit* **F10** *to enter BIOS options*" during the boot process. If so, *hit the key mentioned* (**F10, F2, ESC**, etc.), or search the internet for "*your computer manufacturer*" "*model*" and "*BIOS Key*", on the internet.
- **Enter the BIOS system**, choose the "*Security*" tab, look through the options and find:
  - **VT/x** (or VT-x, or Virtualization Technology)
  - **VT/d** (or VT-d, or Virtualization Technology Directed I/O)
- **"Enable" VT-x**, and VT-d if it is there.. Below is a screen shot of my BIOS page–yours may look different, but should be similar. In particular, you should be able to find the VT-x and VT-d items. If you can only find VT-x, that is fine–turn that ON (Enabled).

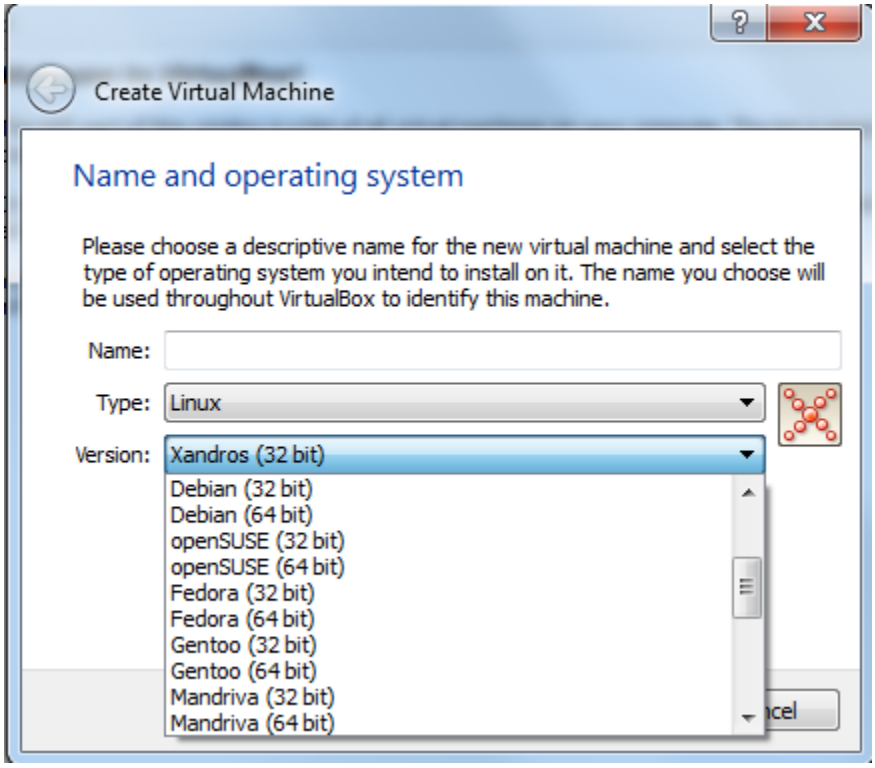


- Once these things are enabled, **Reboot**.
- Start **VirtualBox** again.

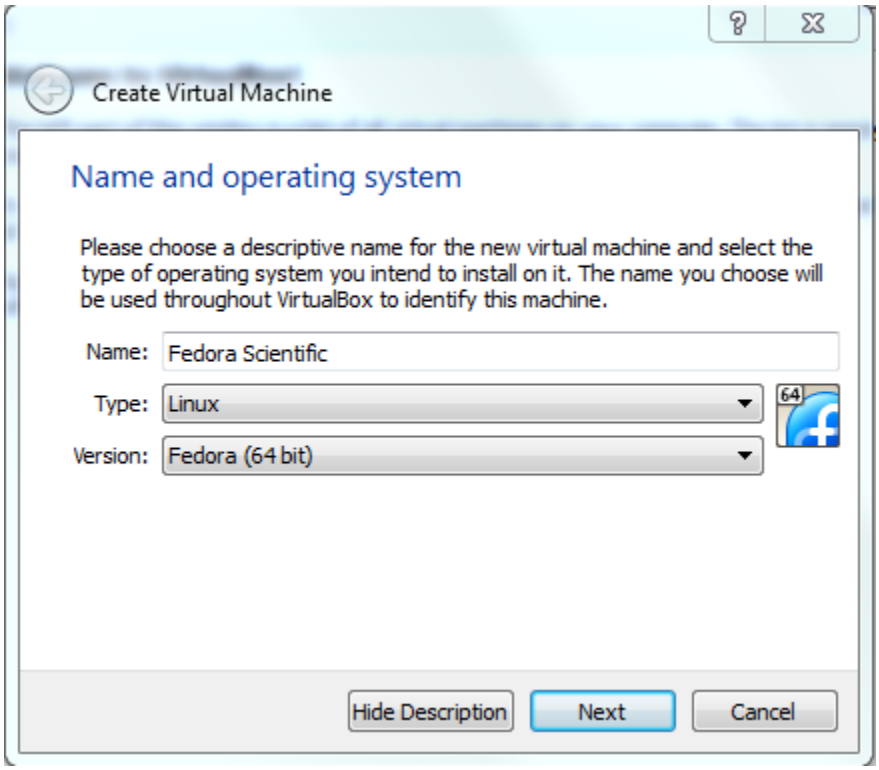
If you have trouble with this, let me know and I'll try to help. However, if we can't get Hardware Virtualization enabled, you can still proceed and install a *32-bit* version of Fedora Scientific (and probably won't even notice the difference).

2.7 Configuring the VM

Now, with Hardware Virtualization enabled, you should see both 32 bit and 64 bit OS versions versions like this:

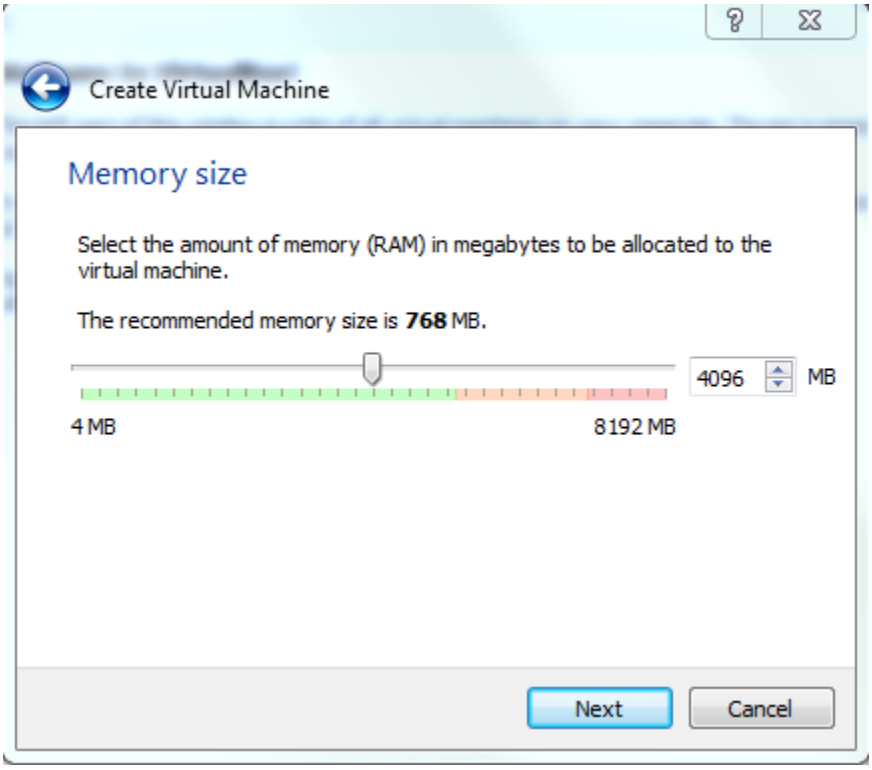


Having selected "**Linux**" as the **Type** of OS, Select "**Fedora (64 bit)** or **(32 bit)** depending on your data bus width. And give your VM a name like: **Fedora Scientific**.



Click **Next**.

You will get this screen, where you will assign the memory for your VM.



I typically assign **Half** of the memory of my machine to the VM. So, if your computer has 4 Gb or memory (also called *RAM–Random Access Memory*), I would assign 2 Gb to my VM. The window will show you the **minimum** recommended memory. You will want more than that, but not so much that you are moving the slider into the RED region.

This memory will only be assigned to the VM when it is running, when presumably you will want it! **As a minimum , don't use less than 1 Gb**. Note that you can type a number in the box too–you don't have to use the slider or Up/Dn arrow buttons.

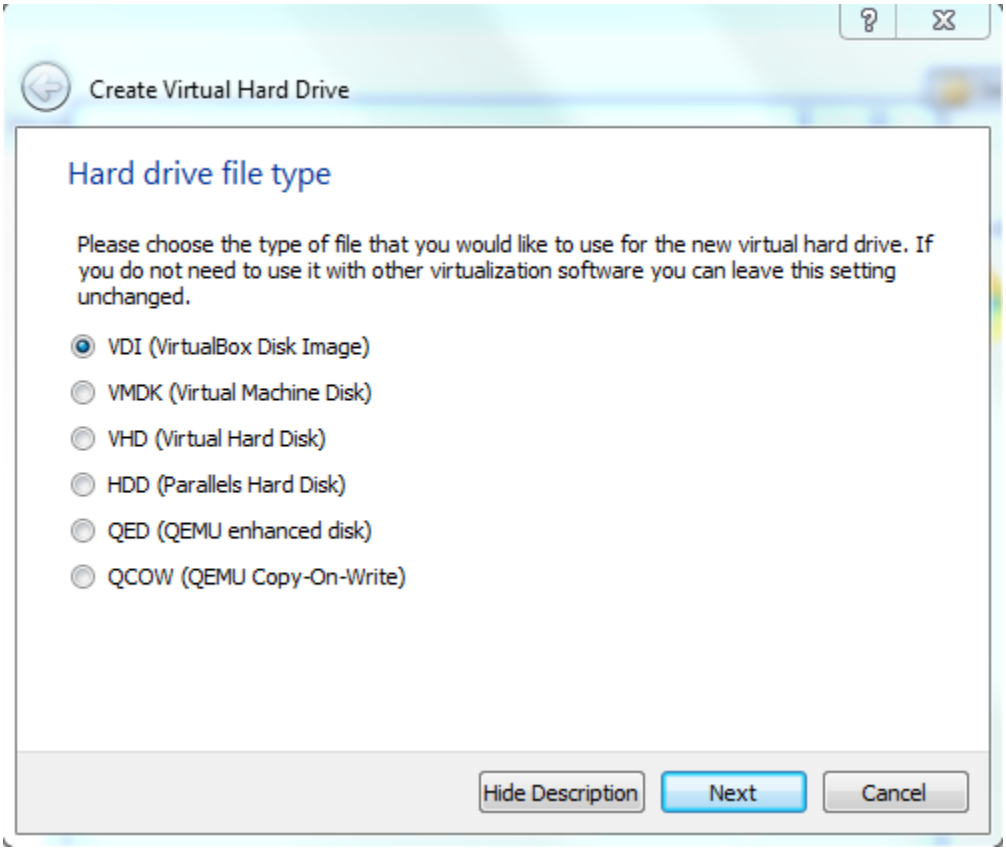
If you have trouble deciding on the amount, let me know–but you might as well try half of your physical memory for starters.

Click **Next** when you have set the memory size.

Now, you will setup the *virtual disk* system for the VM.

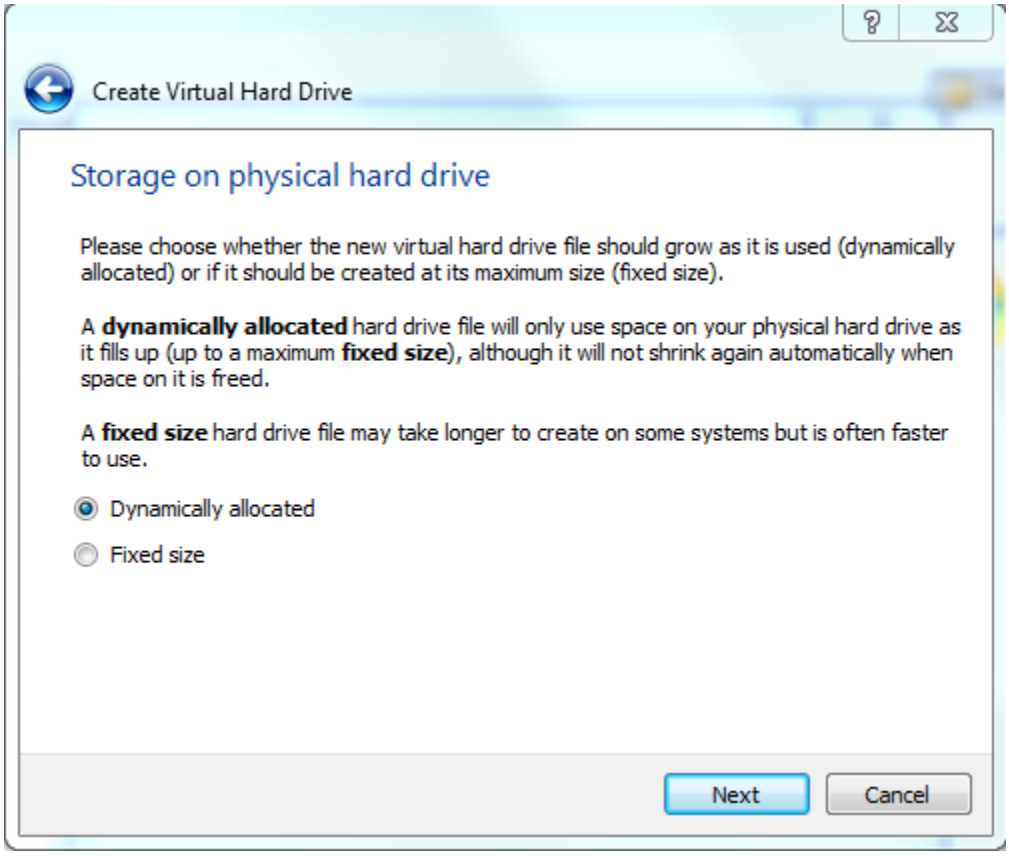
Select "**Create a virtual hard drive now**", then click **Next**.

Click the default type: "**VDI VirtualBox Disk Image**". Then click **Next**.



You will want one that is " **Dynamically Allocated**". This means that even though you set a maximum disk size of, say 40 Gb (which you will do shortly), you will not instantly see this much less free space on your hard drive. The VM will only use the amount of diskspace required to hold the files you create on your VM.

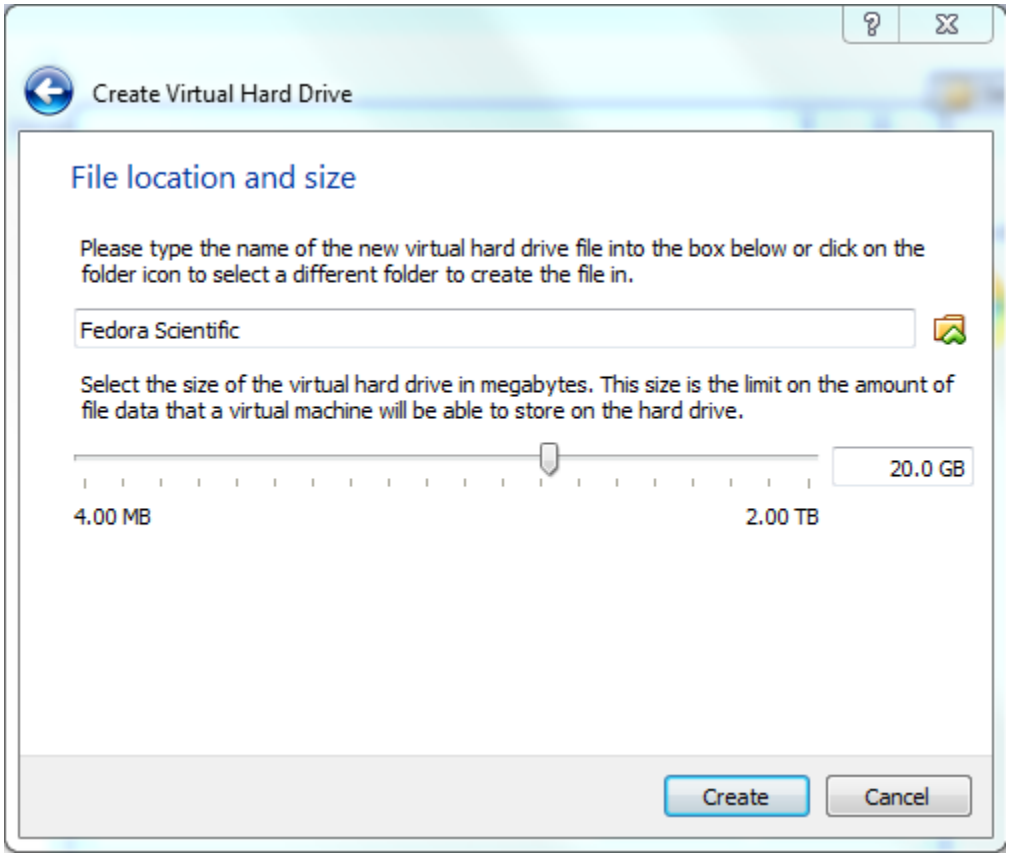
So, check "**Dynamically Allocated**", an click **Next**.



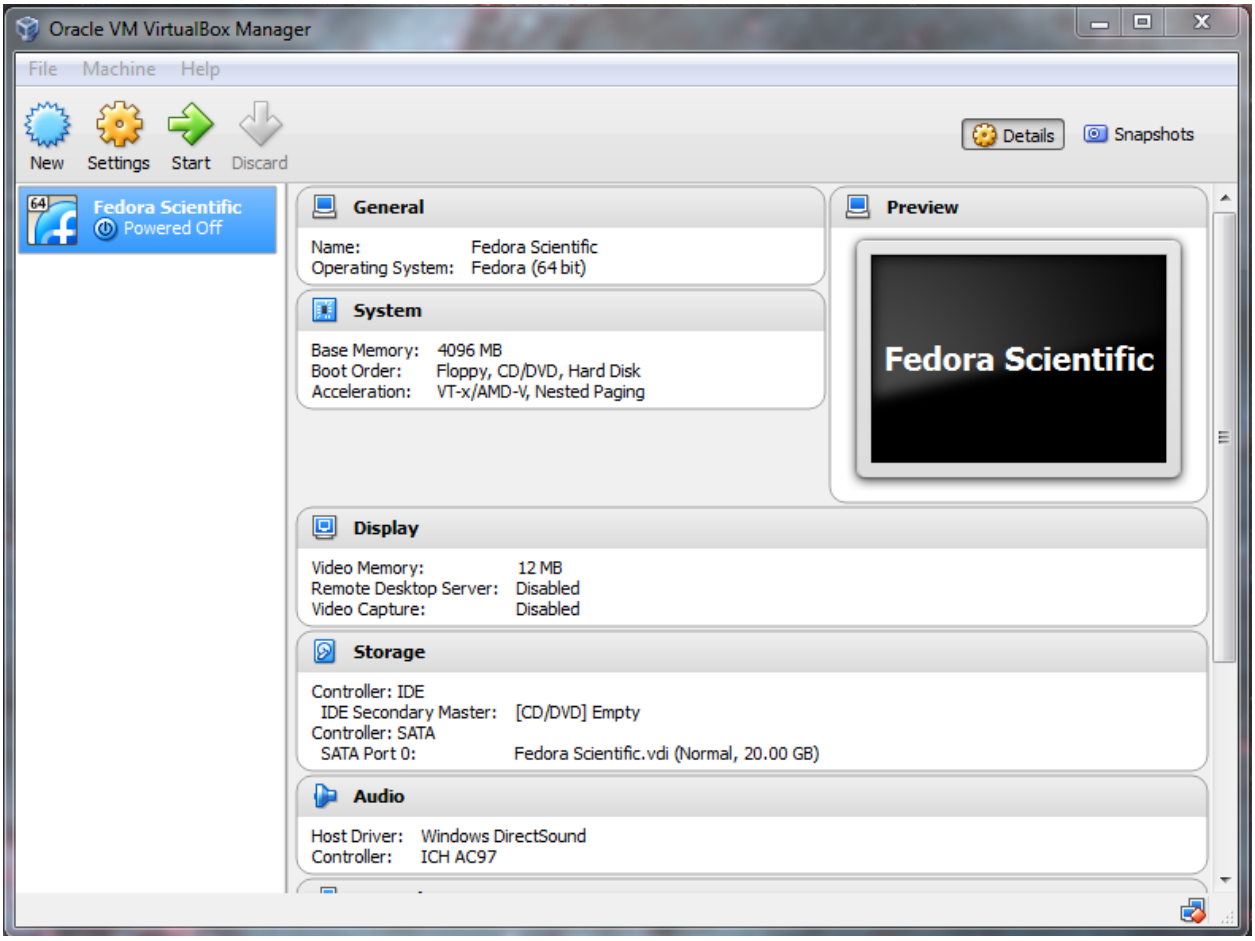
Now you have to choose the maximum size of the disk for your VM. You could go a small as a couple Gb's, however I would suggest between **10-50 Gb** for your Fedora Scientific VM.

You will be able to change this later if you wish–so you can add more space if you start to use your Fedora Scientific VM for a lot of your scientific work (I hope that is the case!).

Choose your disk size, and click **Create**.



You've just configured your Fedora Linux Virtual Machine ! You should be back in the main VirtualBox application window, but now there will be one VM icon on the left panel, with the name of your VM. I will assume you named it "*Fedora Scientific*".



Don't Start it Yet!

Before you start your new VM, you have put the Fedora Scientific DVD (the .iso file) into the DVD drive.

2.8 Attach the Fedora Scientific ISO Disk

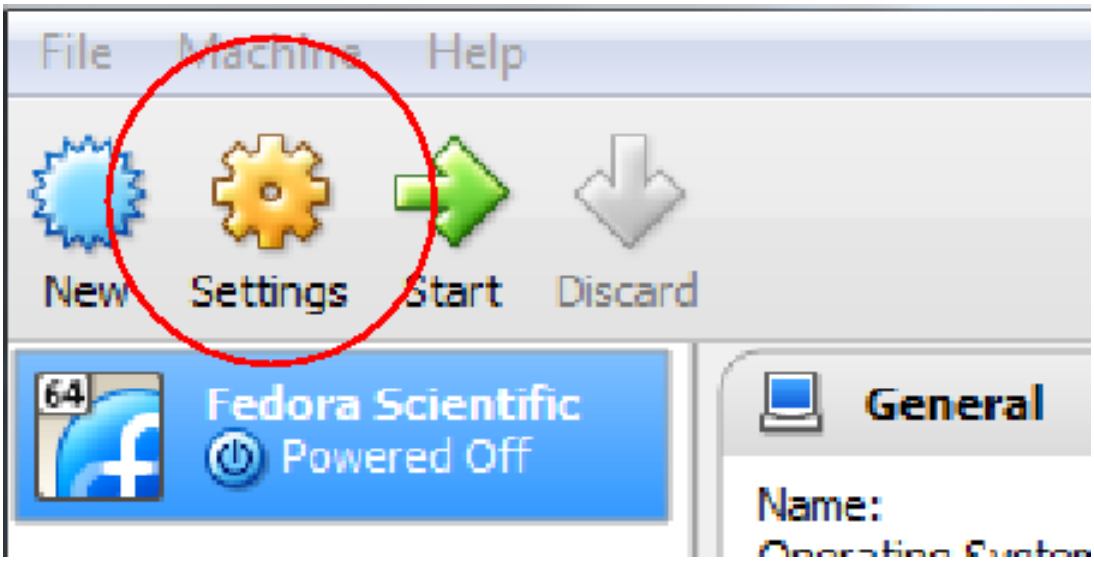
Your new VM is ready to start, but it has no OS on its hard drive (like your computer does). So if you were to start it, it would just turn on and wait.

When you build a new computer from scratch, or install a new OS on a computer, typically you buy (or download) a DVD (or ISO file) with the new OS on it, load this in the DVD tray, and start the machine. When it boots, it will first look in the DVD/CD tray to see if there is a bootable disk there. If so, it boots an OS from that disk. Once that OS is loaded into memory, the computer can carry out instructions and has basic functionality–often just enough to read the new OS from the DVD and write it to the hard drive.

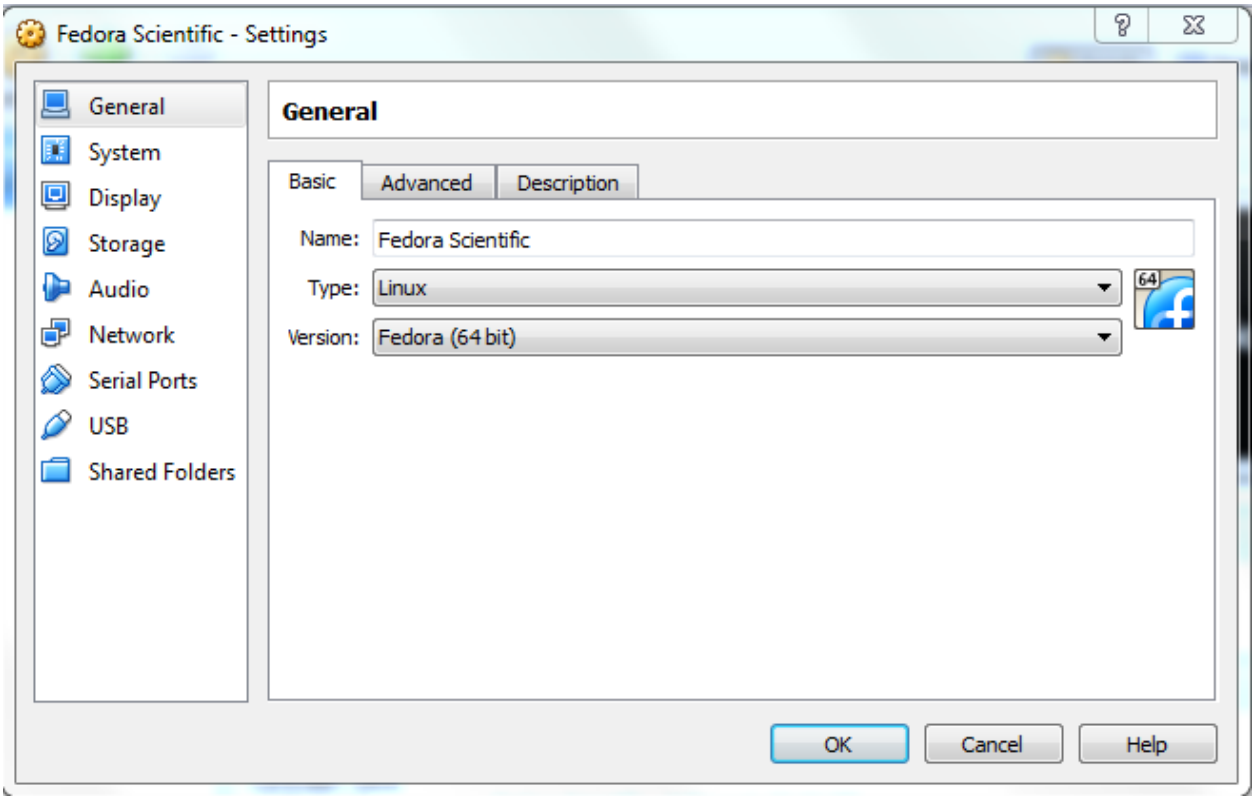
That's what we're going to do here. Except that the OS we want is in the form of the "*Fedora Scientific Linux*" **ISO** file that we downloaded in the last section *software*. **ISO** files are disk images of a CD or DVD. We are going to put this file in a Virtual DVD Drive (How cool is that!).

To do this, make sure that your *Fedora Scientific* VM is highlighted in the left panel. It should be, since it is probably the only one there.

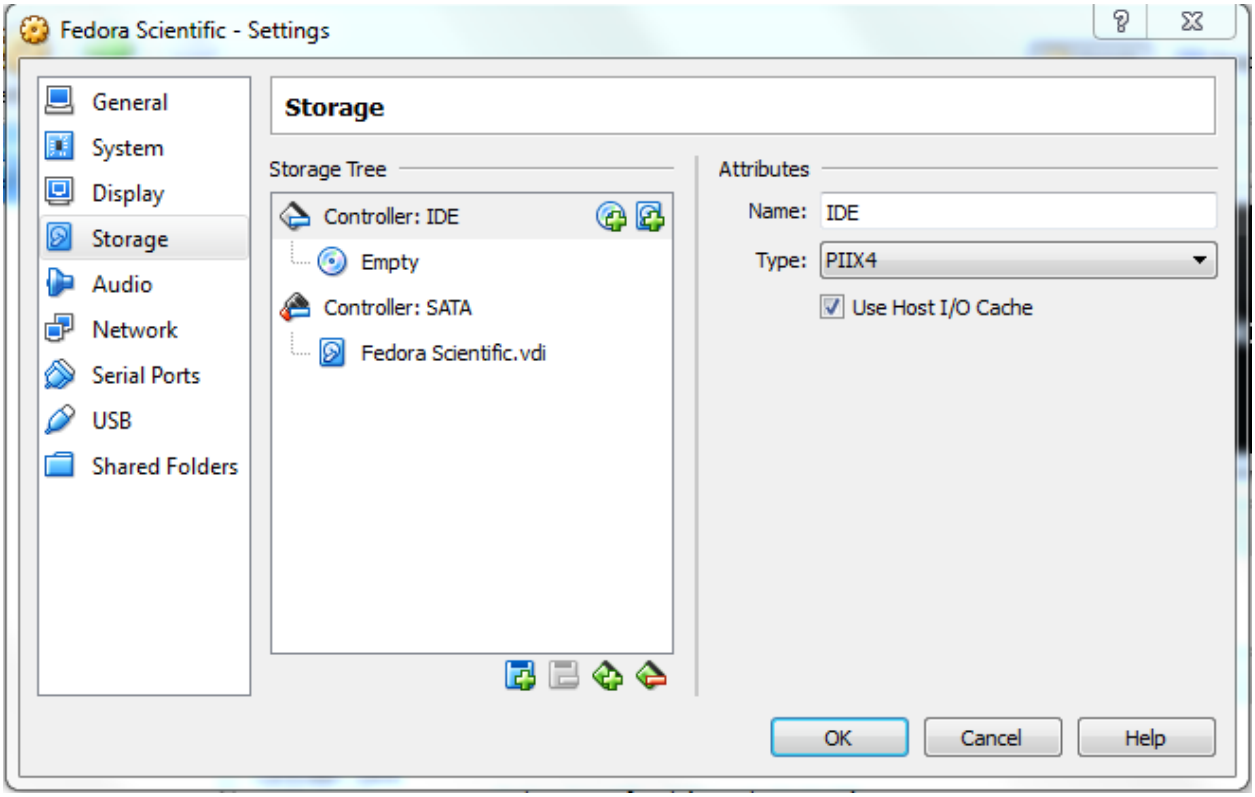
Click "**Settings**" on the top panel.



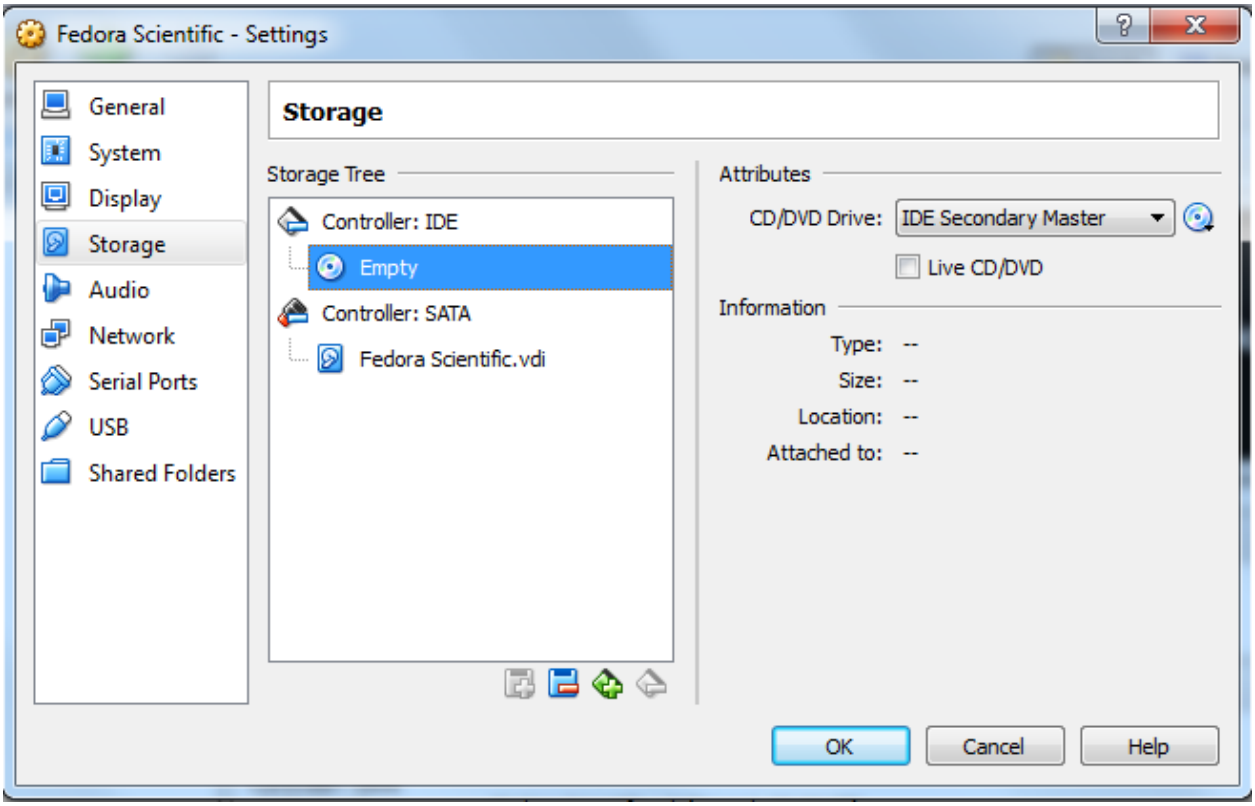
You will see this window.



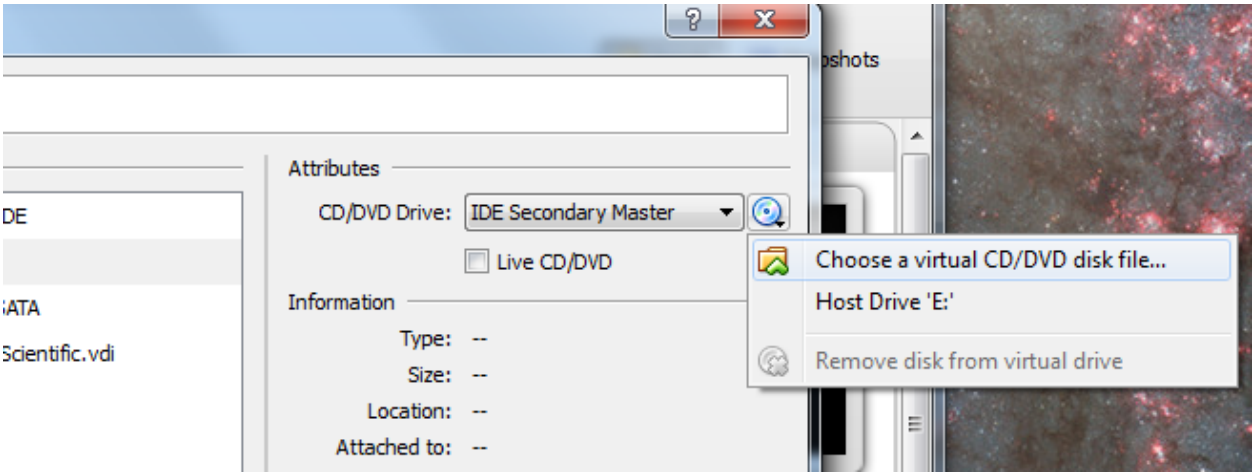
Click "**Storage**" on the left menu, to bring this window forward.



Click the "**Empty**" CD icon under "**Controller: IDE**".

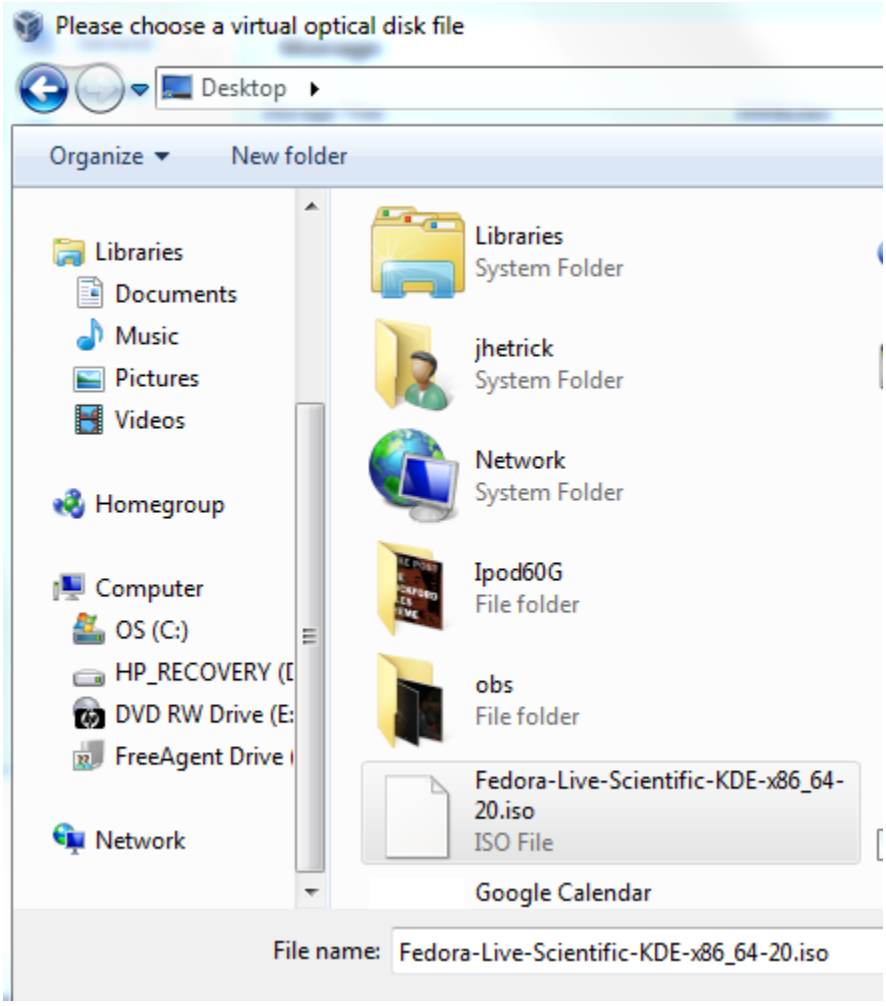


Then, with *Empty* highlighted, **click the CD icon** beside the line that says: "*CD/DVD Drive: Secondary IDE Master*". This drops a menu box like this:

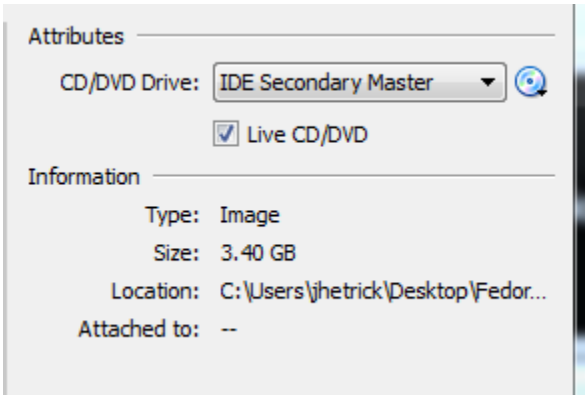


Select "**Choose a virtual CD/DVD disk file...**

Browse your computer to find the `Fedora-Live-Scientific-KDE...-20.iso` file that you downloaded during your *Preparations* in the last section *software*.

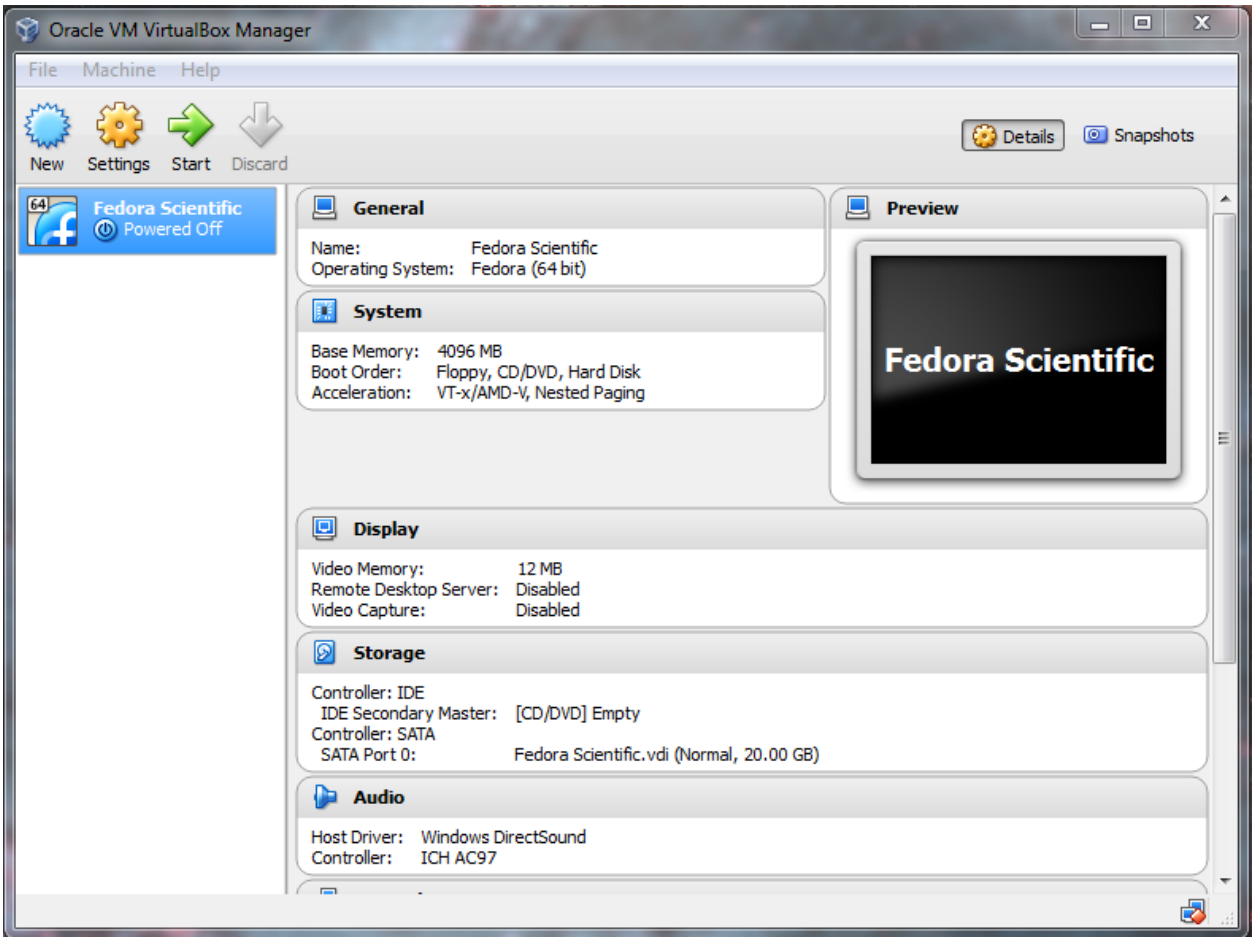


Once you have selected this file, you should see its properties in the Information section of that window, below the CD/DVD Drive. **Also check "Live CD/DVD"**.



At this point, you have virtually loaded a virtual DVD into your Virtual Machine's virtual DVD drive. Thus when you turn on the machine, it will boot the installation sequence from this virtual Fedora DVD file.

Click " **OK**" and return to the main VirtualBox window.



You are now ready to power up your Virtual Machine for the first time

Go ahead and click the " **Start**" **-> button** and move on to the next section.

2.9 Installing Fedora Scientific on your VM

2.9.1 Video for this Section

Video for this section is here.

The first part of the video is the VirtualBox VM creation. Fedora Scientific Linux installation is in the second half of the video.

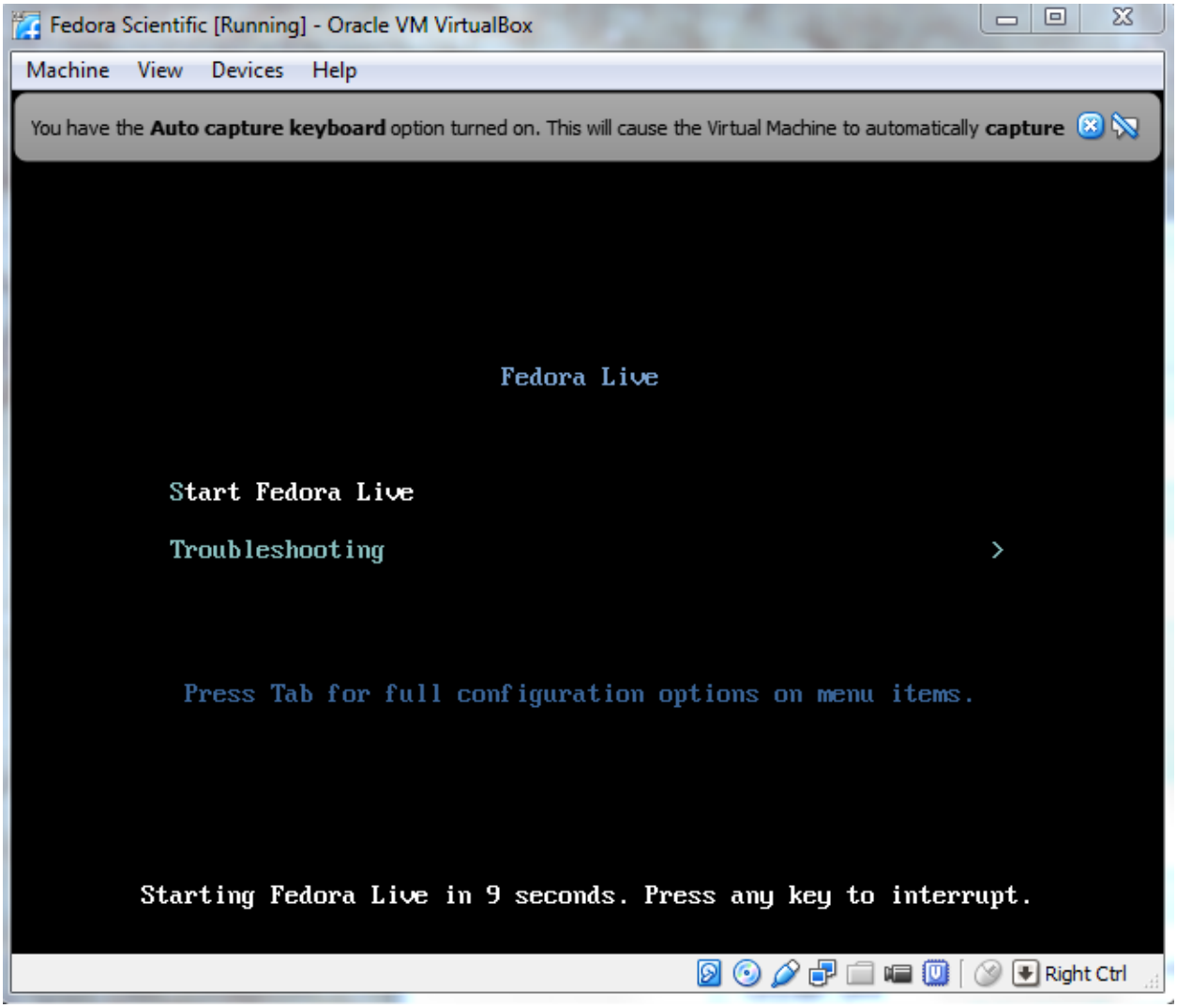
2.10 Booting Linux from the DVD

If you haven't done so,

- Start **VirtualBox**.
- Highlight your **Fedora Scientific** VM.
- and press the **Start (->)** button at the top of the window.

You may have already done this while reading the previous section, and your machine is starting up.

Either way, you should see the display of your new VM starting up, and after a couple spits, it should look like the following.



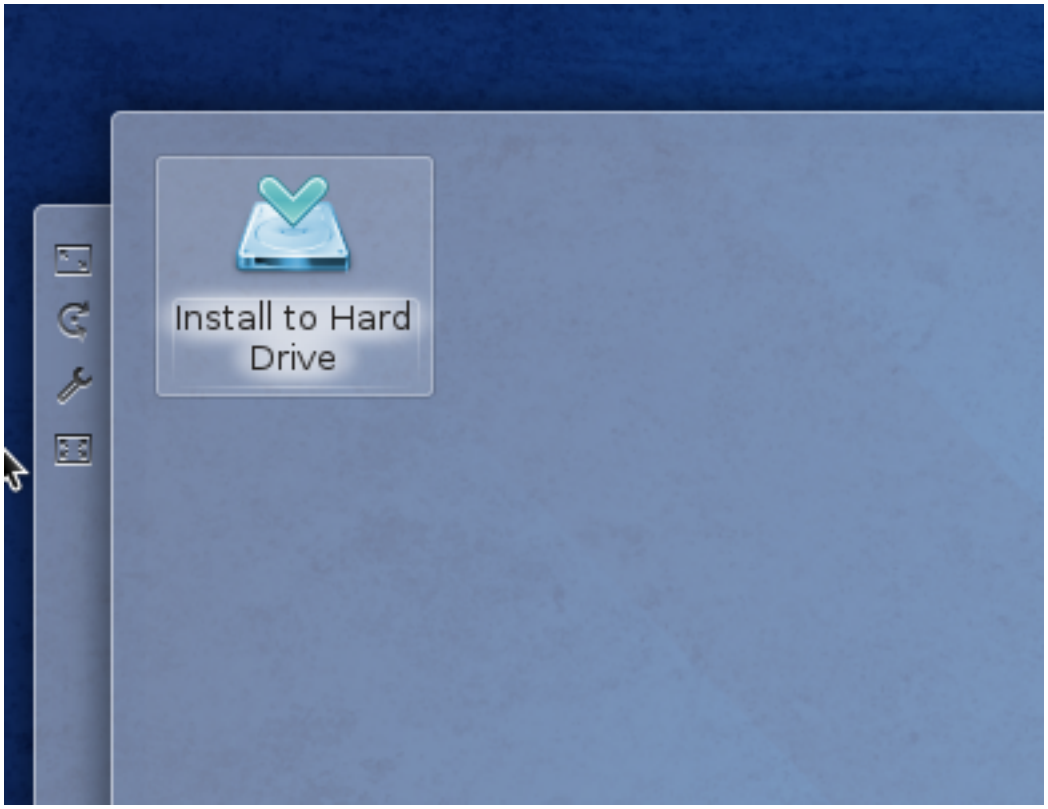
just as it would if you had loaded the Fedora Scientific Install DVD in the DVD drive of a new desktop computer.

You can do nothing for 10 seconds, or you can hit the ENTER button. This will start the Linux boot process and the screen switches to this one, with a blue line building across the bottom of the screen.

You can also click the little blue **[x]** on the info bubble-bars that popped up at the top of the window.

In a short time, the bottom icons will all be there and the file viewer is there.

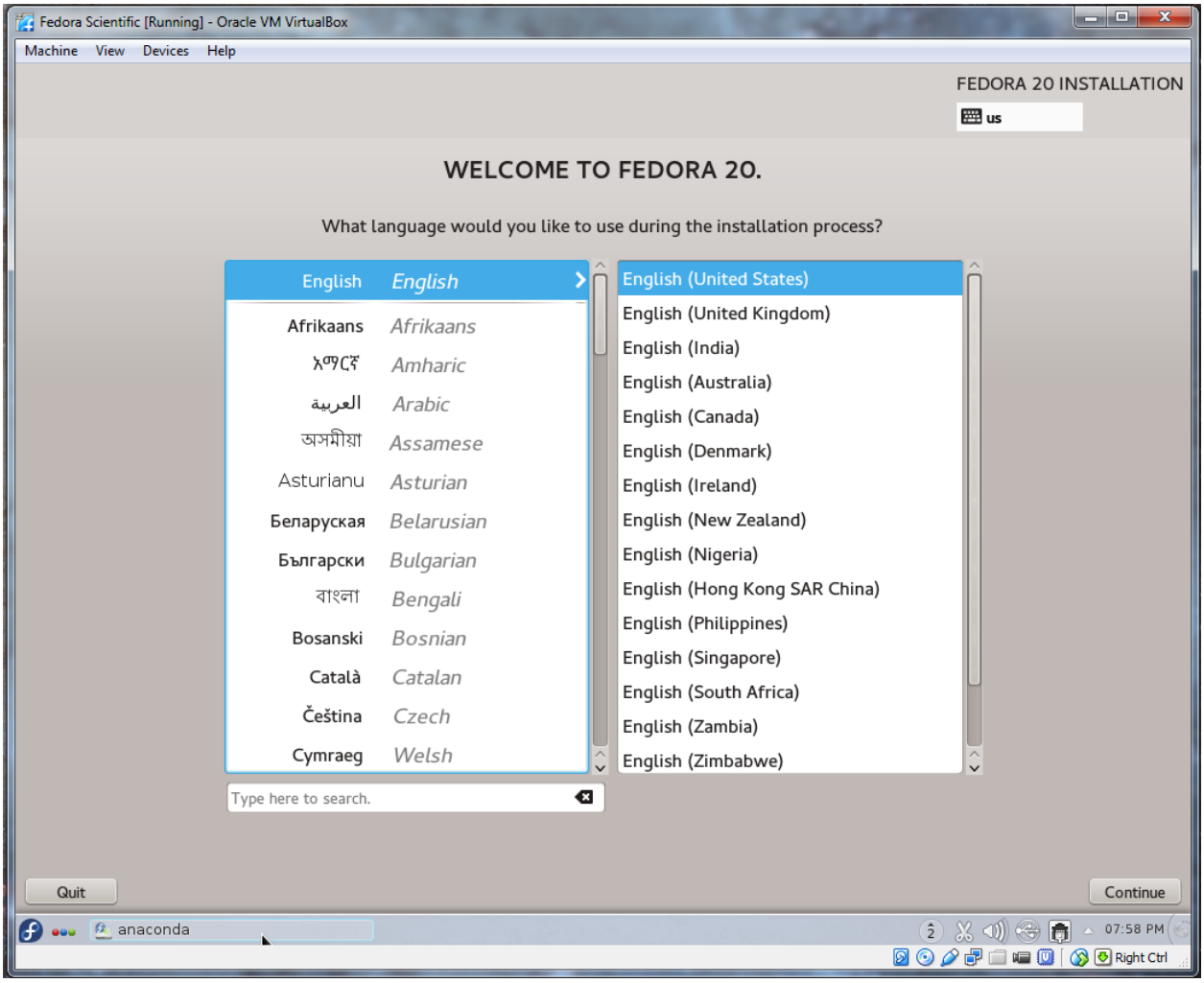
An icon saying " **Install to Hard Drive**" should be there.



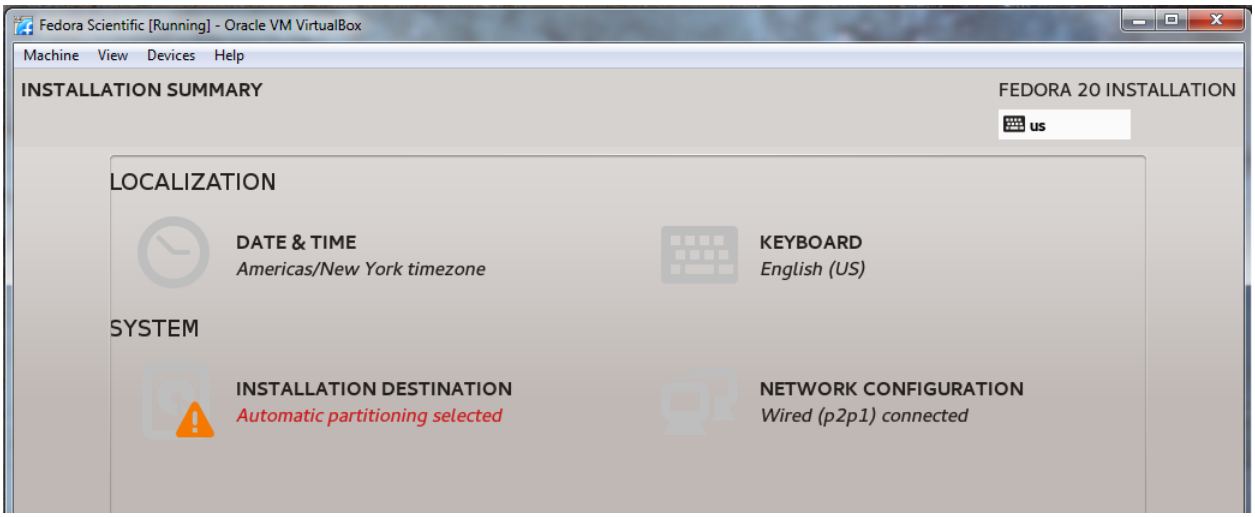
2.11 Install to Your Hard Drive

To start the Fedora Scientific Installation onto the virtual hard drive of your VM, **double click that icon**.

This will start the Installation sequence and you will be prompted for a few configuration items, like your choice of **Language**. I chose *US English*, but you suit yourself.

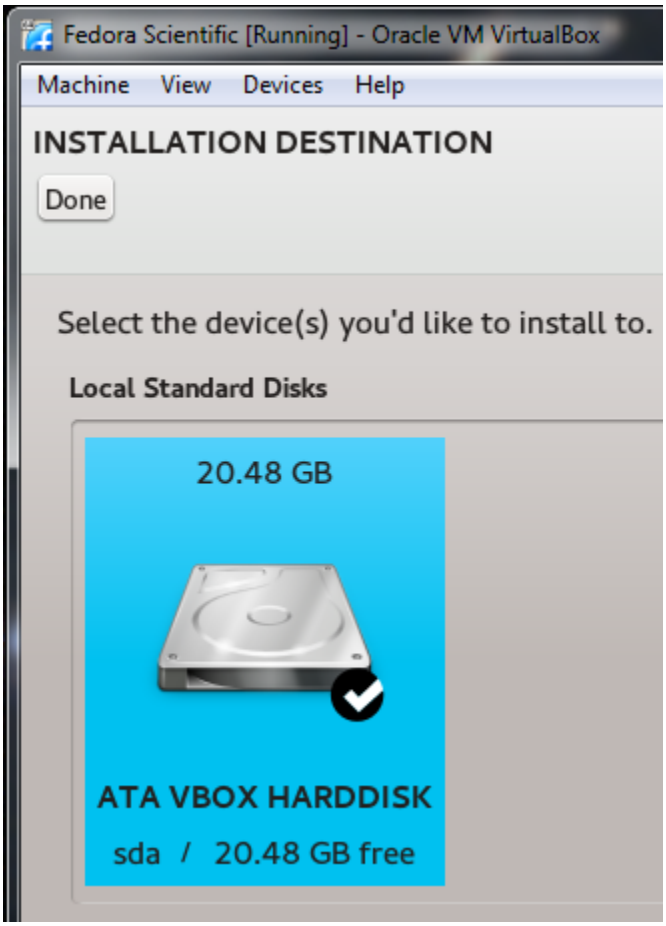


Next you get this window:



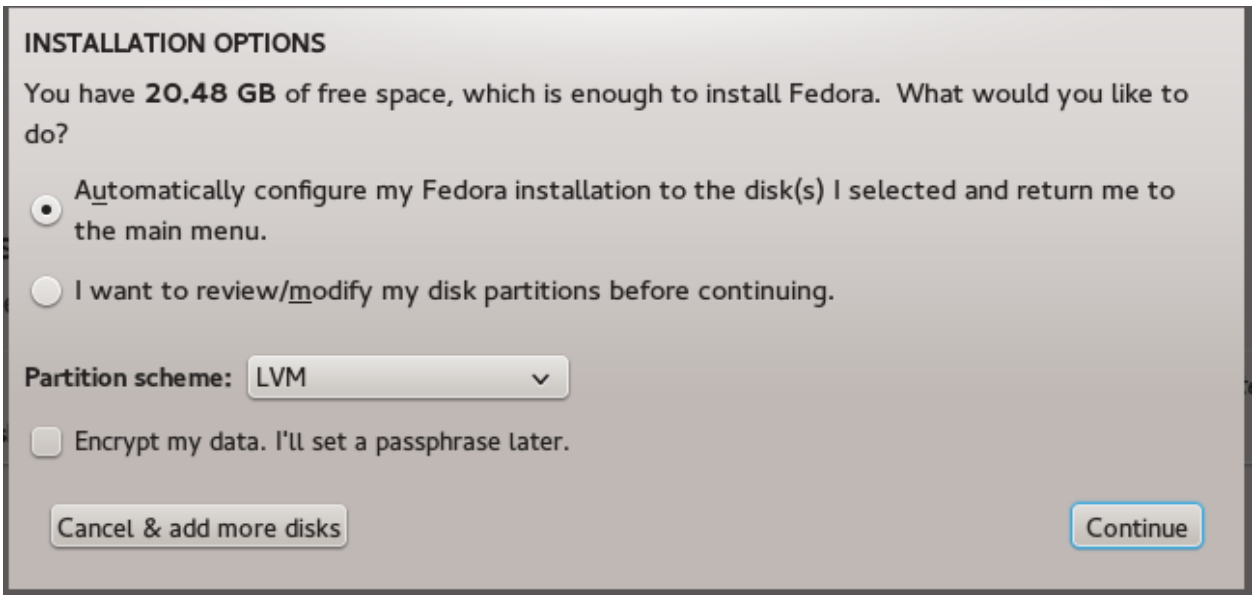
Chances are good that everything is ready except the "**SYSTEM**". The installer would like your permission to partition the virtual disk.

Click **SYSTEM** and you should see this window.



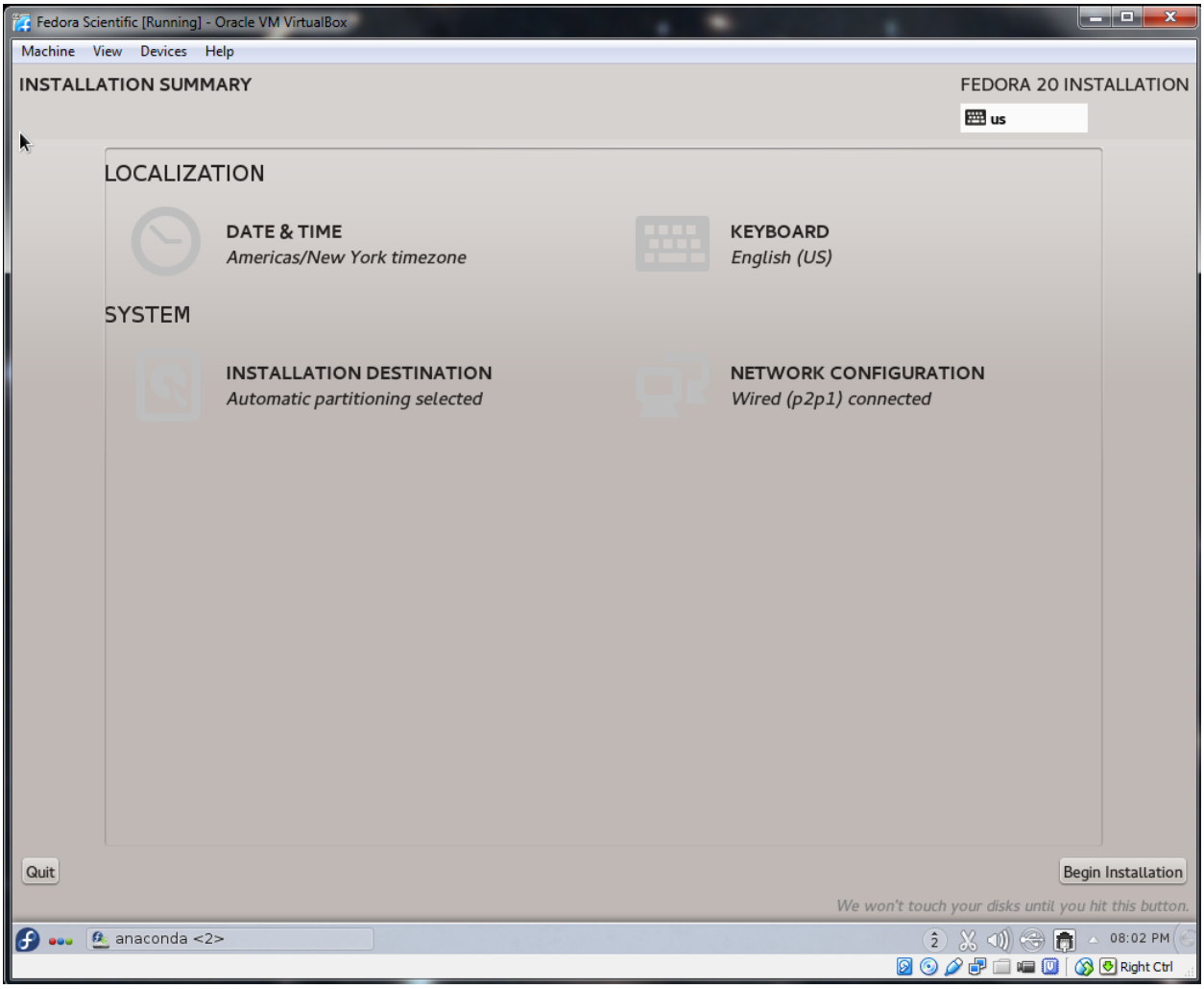
The "**ATA VBOX HARDDISK**" is probably not highlighted (as it is here). Click Once on the hard drive icon. It should turn teal (like below) and a little check mark should appear.

Once yours looks like mine here, Click "**Done**". A window should pop up that looks like this:



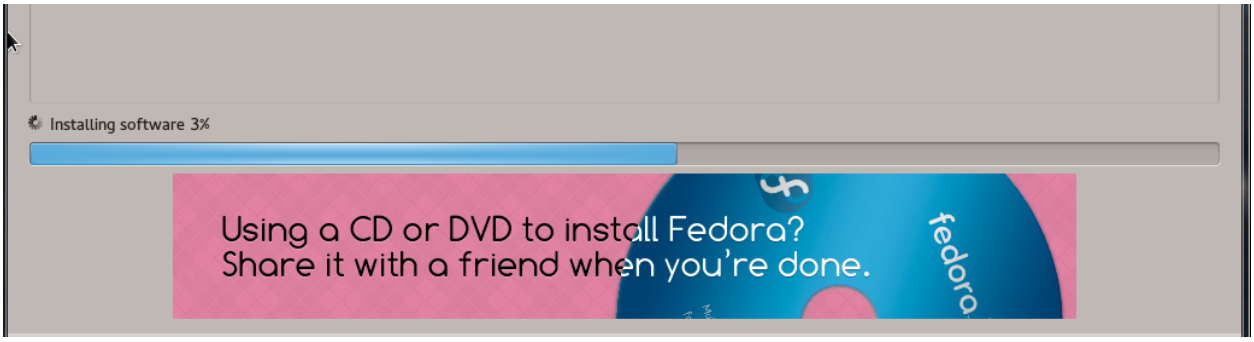
Keep these defaults and Click **Continue**.

Then you should have everything ready to go (i.e. no **!** Error Messages), and your screen will look like this.



Click "**Begin Installation**", in the lower right, to...begin the installation of Fedora Scientific onto your VM's hard drive .

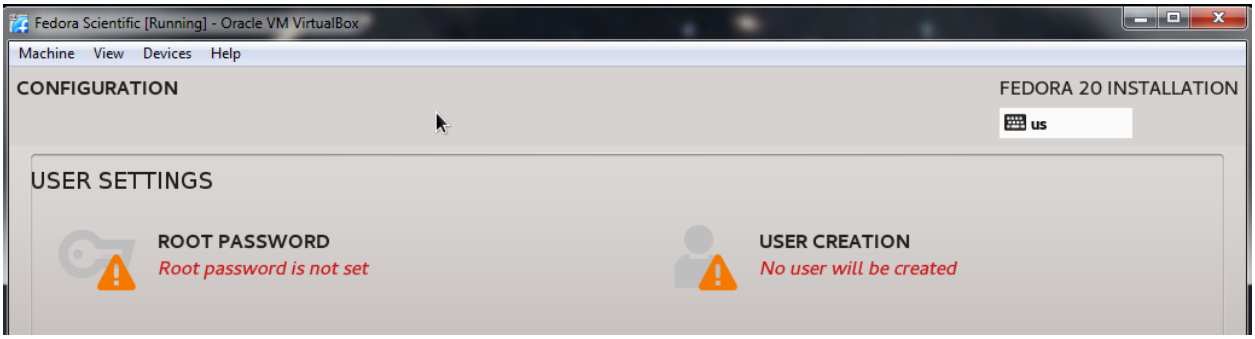
The status bar at the bottom will begin to fill rightward,



and the installer will ask you to do a couple things while it copies and unpacks files.

2.12 Create Root and User Accounts

The installer will ask you to set up two accounts and give them passwords.



One is the **Root** account. This is the system adminstrator account and has complete access to the system. Root can open any file, and more importantly delete any file. So be careful! You will use this account occasionally to make changes to things on your Fedora system. Click **Done** to set the Password.

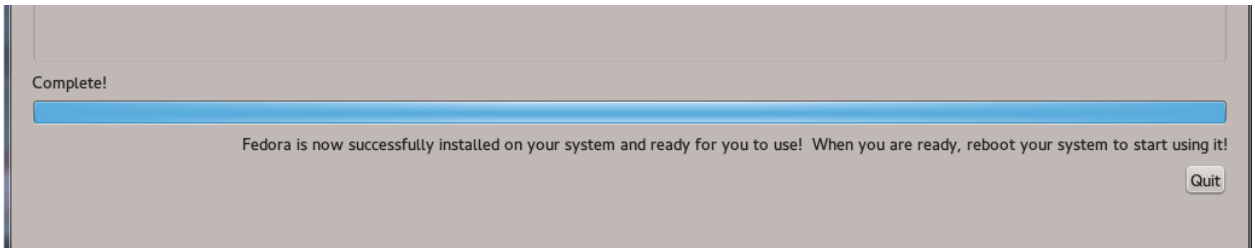
The other is the **User** account you will use the rest of the time (i.e. your normal account). I find that it is easiest to make the **username** of this account the same as your **PacificNET ID**.

> **Warning:** *Save these passwords somewhere*, or better, use ones that you will remember. If you forget your *User* pasword, you can use to *Root* account to reset it. But if you forget the *Root* password, you may have to reinstall everything.

Click **Done** , like before, to create the account and set the Password.

This should bring you back to the main Installation window, with the status bar increasing, followed by post-installation steps.

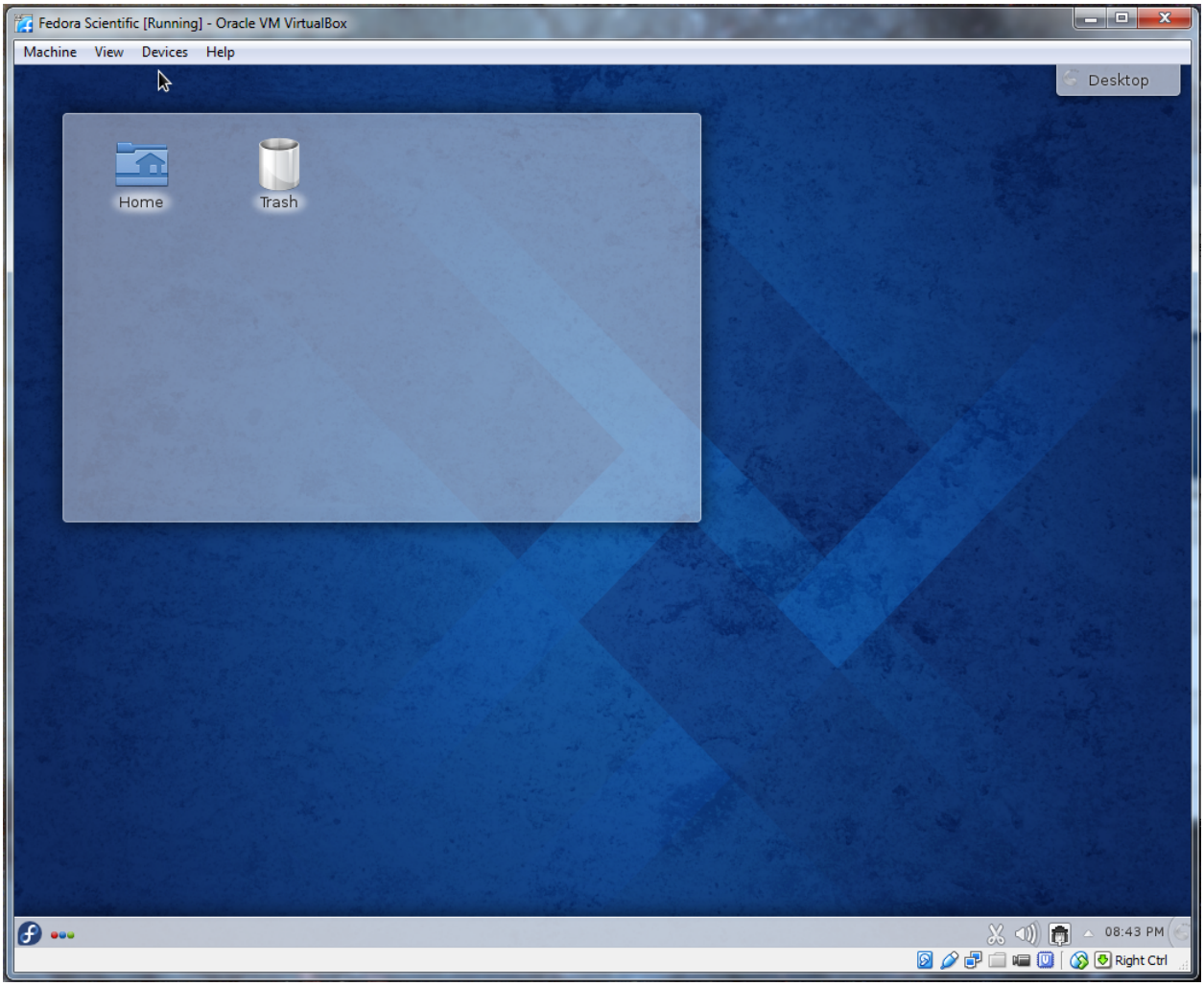
Finally,



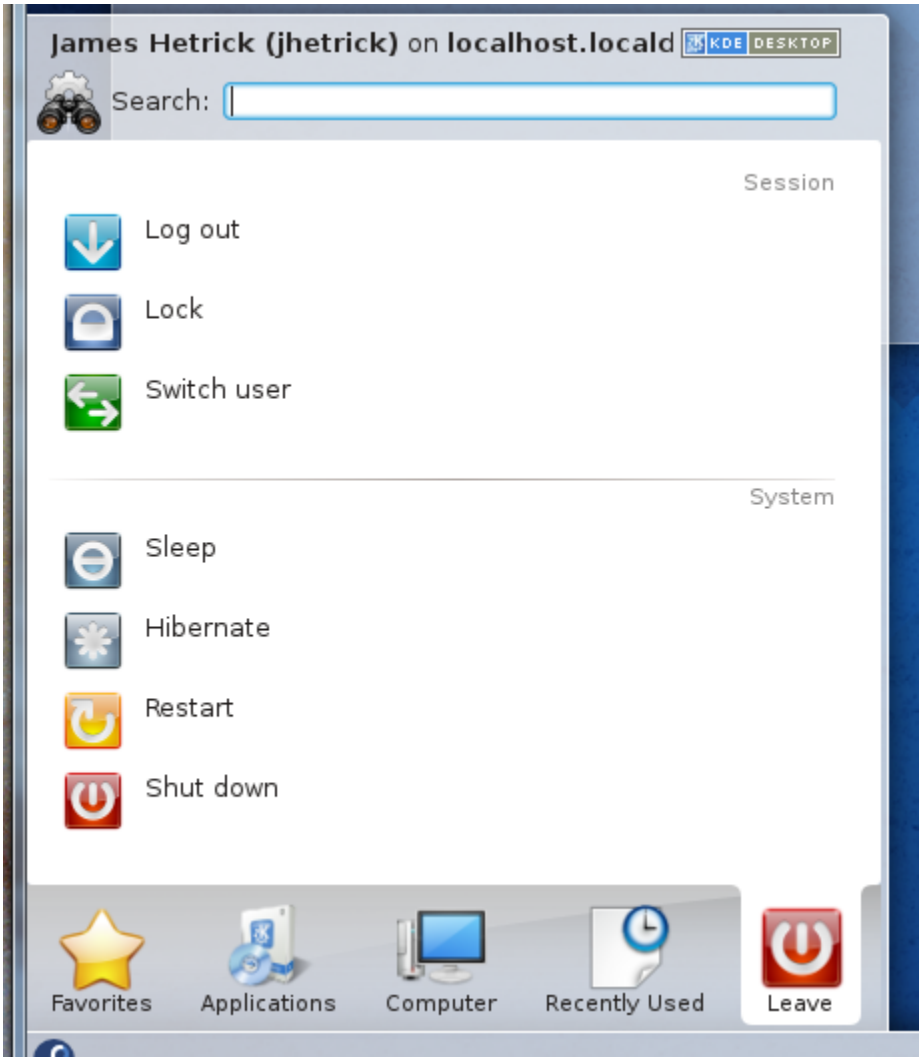
you will see the "**Success!**" banner. Congratulations. If you see this, you most likely have a working installation of Fedora Scientific Linux.

Click **Quit**.

This will end the Installation to Disk process, and bring you back to the Fedora Desktop .

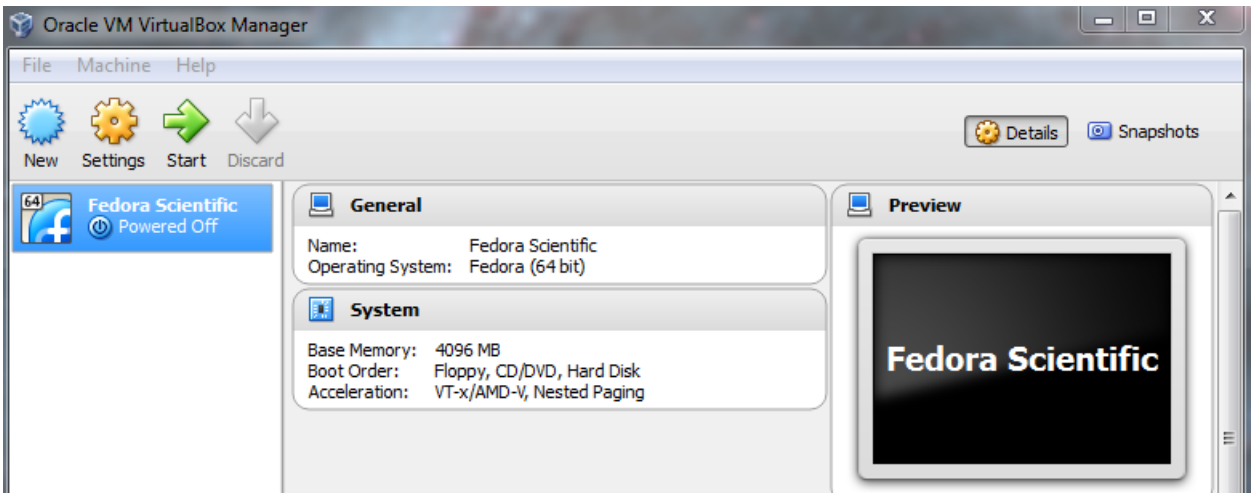


Next, power down the VM by clicking the Fedora "**Start**" **(f)** button, in the lower left corner. Then Click "**Leave**", followed by "**Shutdown**".



The VM will ask you one more time. Click "**Turn of Computer**"

After a couple of system flickers, the VM display should turn **black** and close. Your Oracle VirtualBox window should be back to looking like this, meaning the VM is OFF (and waiting to be awakened again).



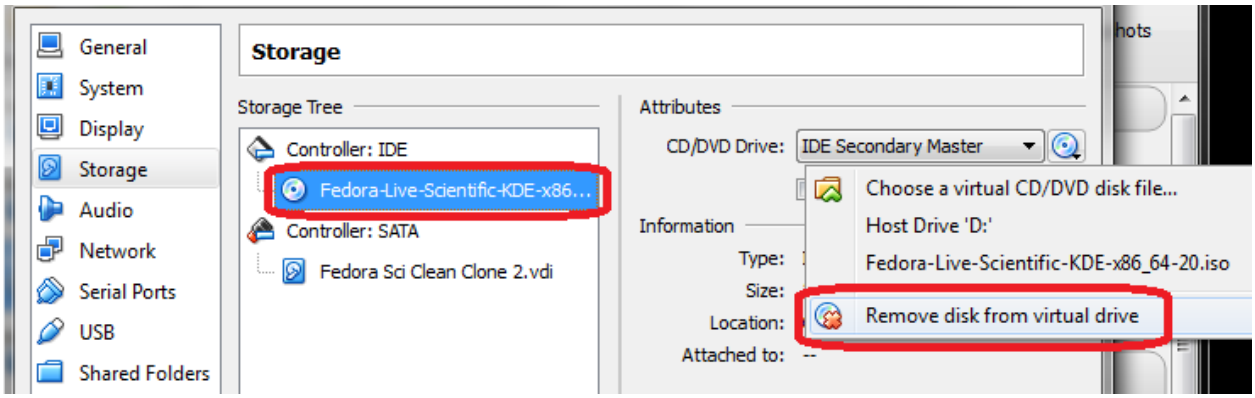
2.13 Boot Linux from Your Hard Drive

Since we still have the `Fedora-Live-Scientific-...-KDE.iso` file loaded in the virtural DVD tray of the VM, if we were to power up the machine again, it would boot from DVD and present us with the "**Install to Hard Drive**" icon again.

We don't want that. So we need to Eject the ISO file from the DVD drive. But the VM has no eject button...

To eject the DVD file, (or in the language of Unix: *unmount* the ISO file), do the following.

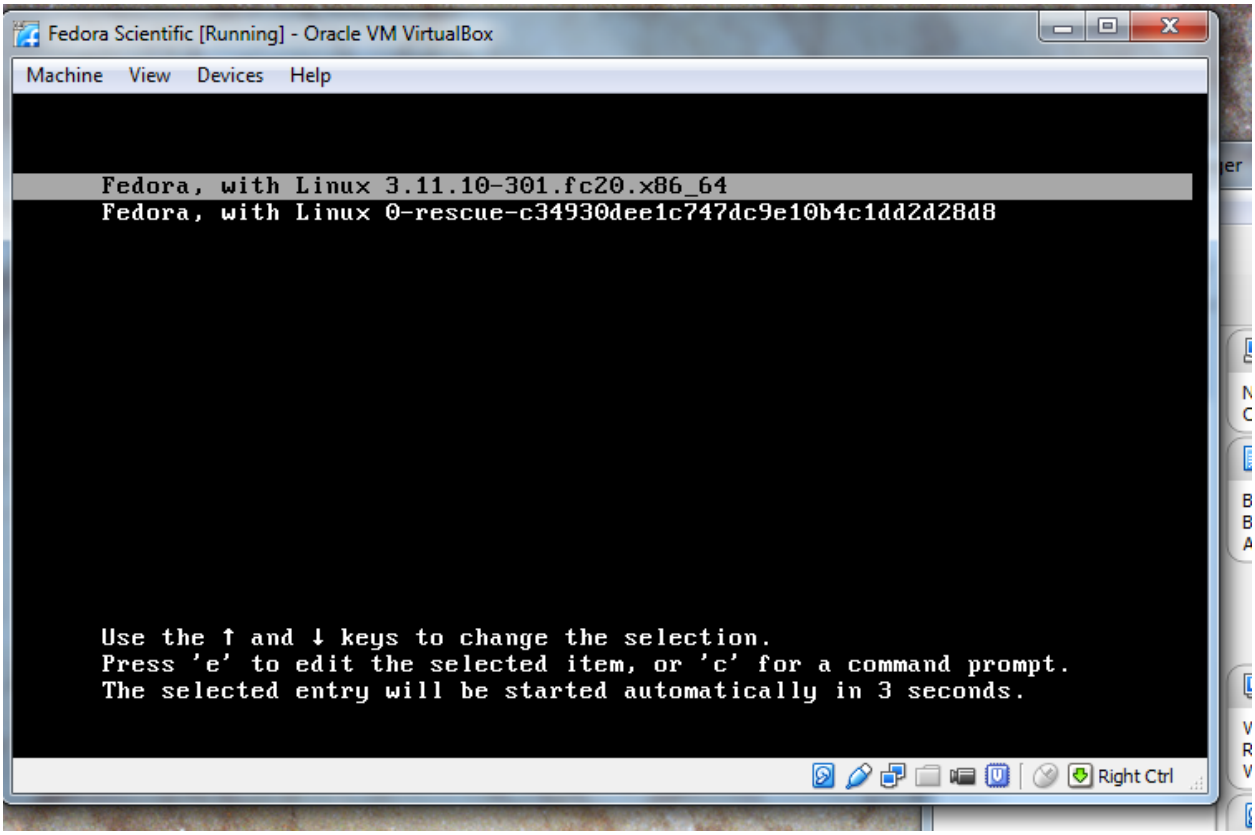
- In the VirtualBox window, Click the Blue icon on the left for your "**Fedora Scientific**" VM,
- then Click the "**Settings**" button at the top.
- Click "**Storage**" on the left, and then under **Controller: IDE**, Click the `Fedora-Live-Scientific...` DVD icon to highlight it (so it turns blue),



- Then, on the far right, Click the DVD icon to the right of "*CD/DVD Drive: IDE Secondary Master*".
- This drops a menu: Select "**Remove disk from virtual drive**"
- The DVD icon below *Controller: IDE* (in the center of the window) should say: "**Empty**".
- Then Click "**OK**" at the bottom.

Your VM is now ready to boot from its own hard drive, *without* the installation DVD.

Go ahead and "**Start**" (**->**) your Fedora Scientific VM again. This time, you will briefly see a new screen:



Showing two "kernel" options which you can boot. One is the main Linux system, and the second is a bare-bones Rescue system in case something went wrong. Within about 3 seconds, the main Linux kernel boots and you are presented with a login screen.

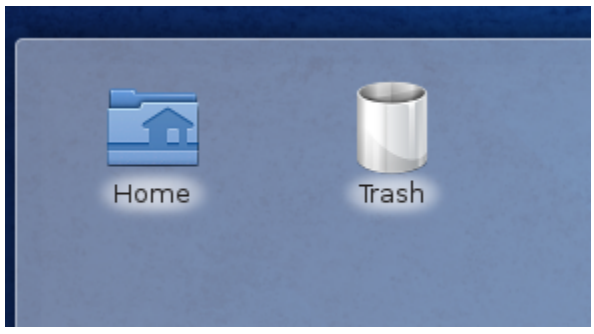
Login with the **Standard Username** (not Root) in the top box, and the **password** in the lower box, which you created earlier.

Shazzam!

Your new Fedora Scientific system should have booted, and you are looking at the **Fedora/KDE** Desktop.

Fedora is the version of Linux we are running. Others are: Ubuntu , Debian , etc., and KDE is the version of the Desktop environment. The other popular Linux Desktop is Gnome .

Notice the two icons:



just like you would see on those " Other " common OS's. From here you can probably start to intuit your way around. You have

- a **Home** and **Trash**
- a **Start Orb** with an **f** at the lower left.
- a **Panel** along the bottom of the window, with various icons.

Not unlike a Windows or Mac desktop, eh? But without all those 3rd party add-ons. No "*1-month free trial of McAffrey*", or other things fighting for your attention.

Go ahead and look around. It's your new Linux system.

2.14 Installing " Guest Additions " on your VM

2.14.1 Video for this Section

Video for this section is here.

If you haven't done so,

- Start **VirtualBox**.
- Highlight your **Fedora Scientific** VM.
- and press the **Start (->)** button at the top of the window.

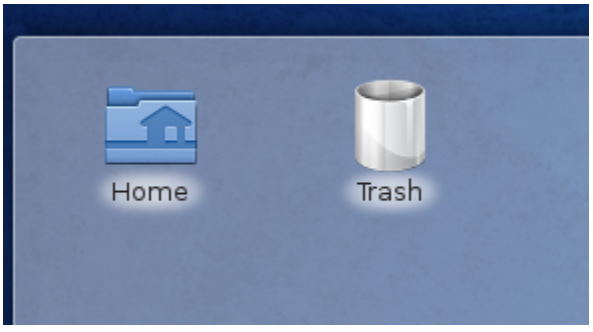
2.15 First Look at Linux

Once you have the login page for Fedora, login as the Standard User. You should be looking at your Desktop , with Home and Trash icons.

In the previous section, you installed Fedora Scientific Linux onto a VM.

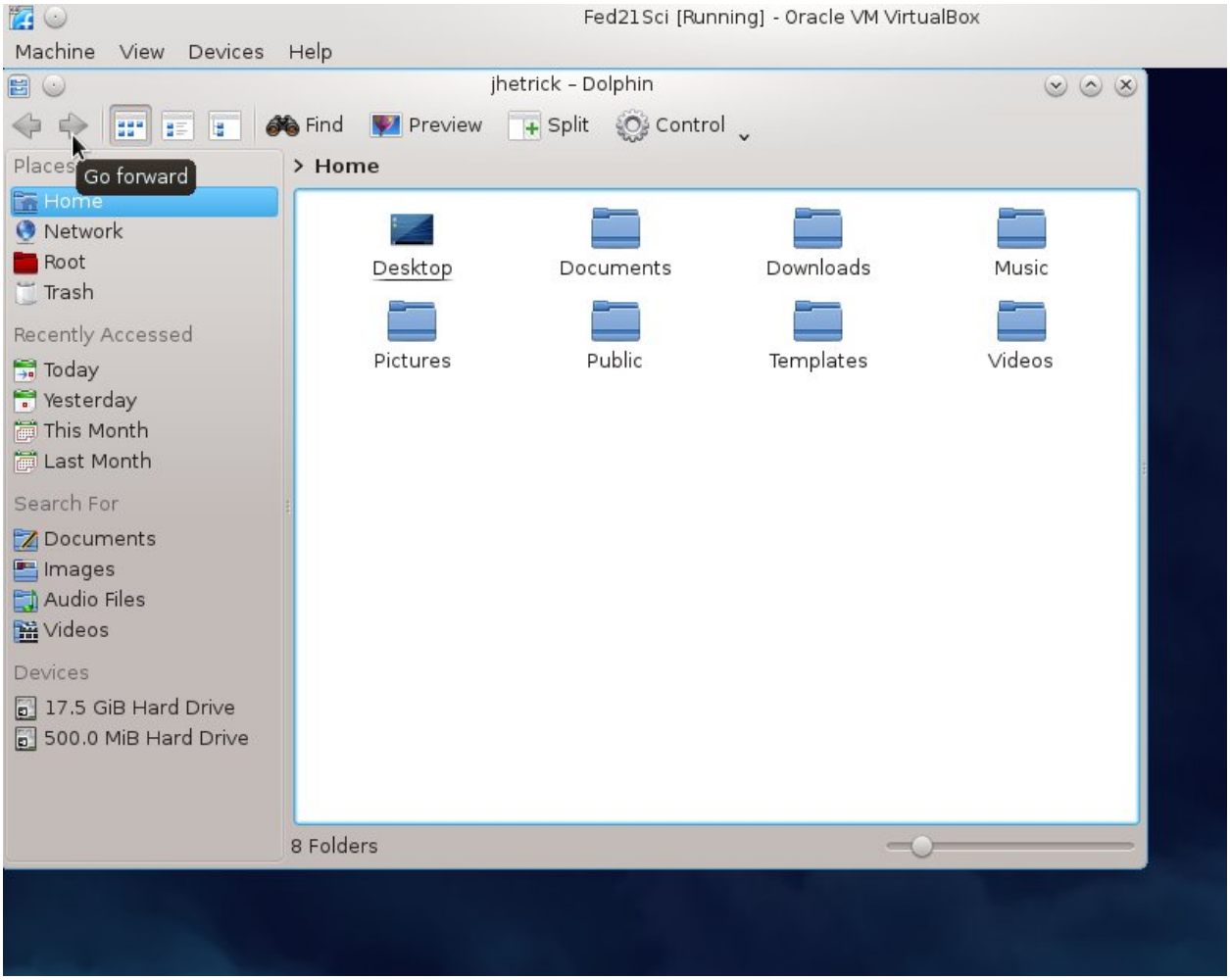
At the end of that section, you were left with the Desktop and encouraged to "look around".

Let's start with your "`Home`" folder.



Double Click the `Home` icon.

This opens the Dolphin File Viewer . Dolphin is the name for the File Viewer/Navigator system of the KDE desktop. It's similar to Window's Explorer or the Mac's Finder .

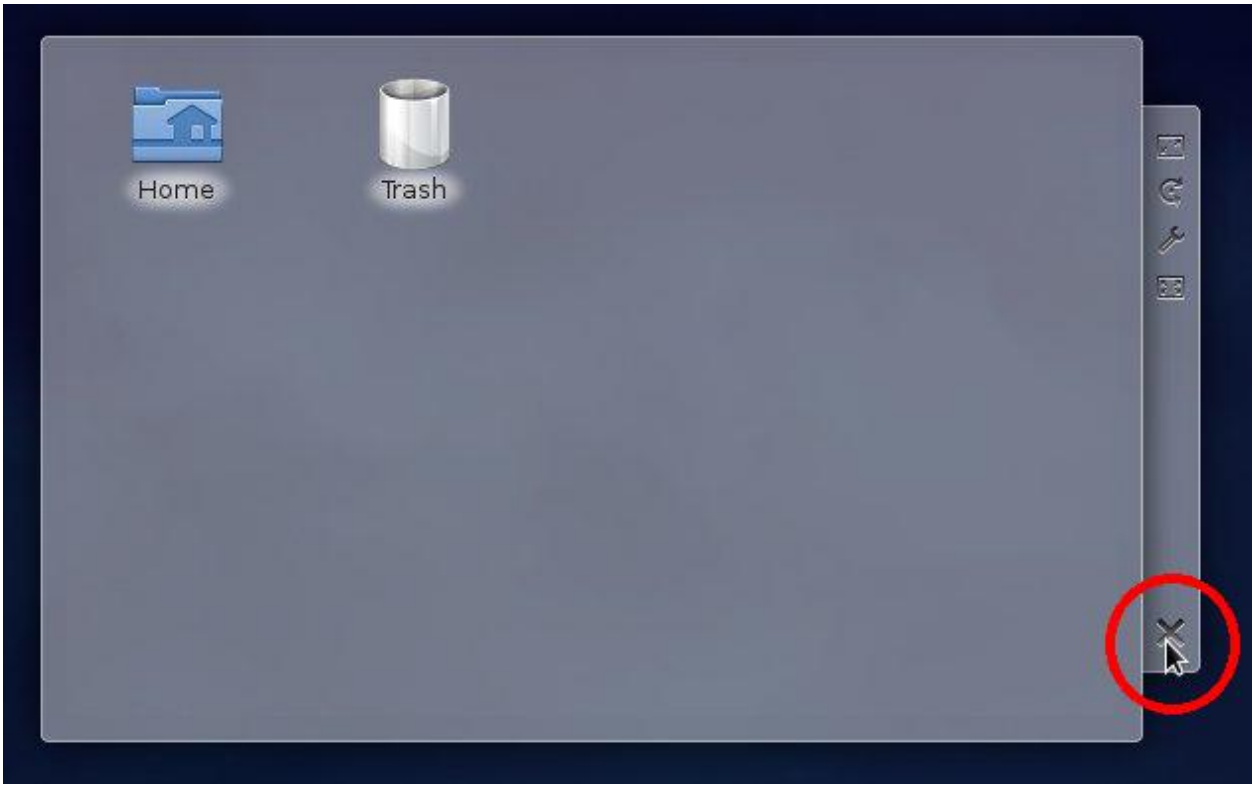


You can see the *Sub-folders* under your "`Home`" folder. They too should look familiar–`Documents`, `Downloads`, `Pictures`, `Music`,... In fact, so far this could just be a sexy blue Windows machine.

The real power of Unix however, is going to shine when you get used to using the *Unix Terminal* and its simple command line for controlling your computer. This is the first thing a hacker wants.

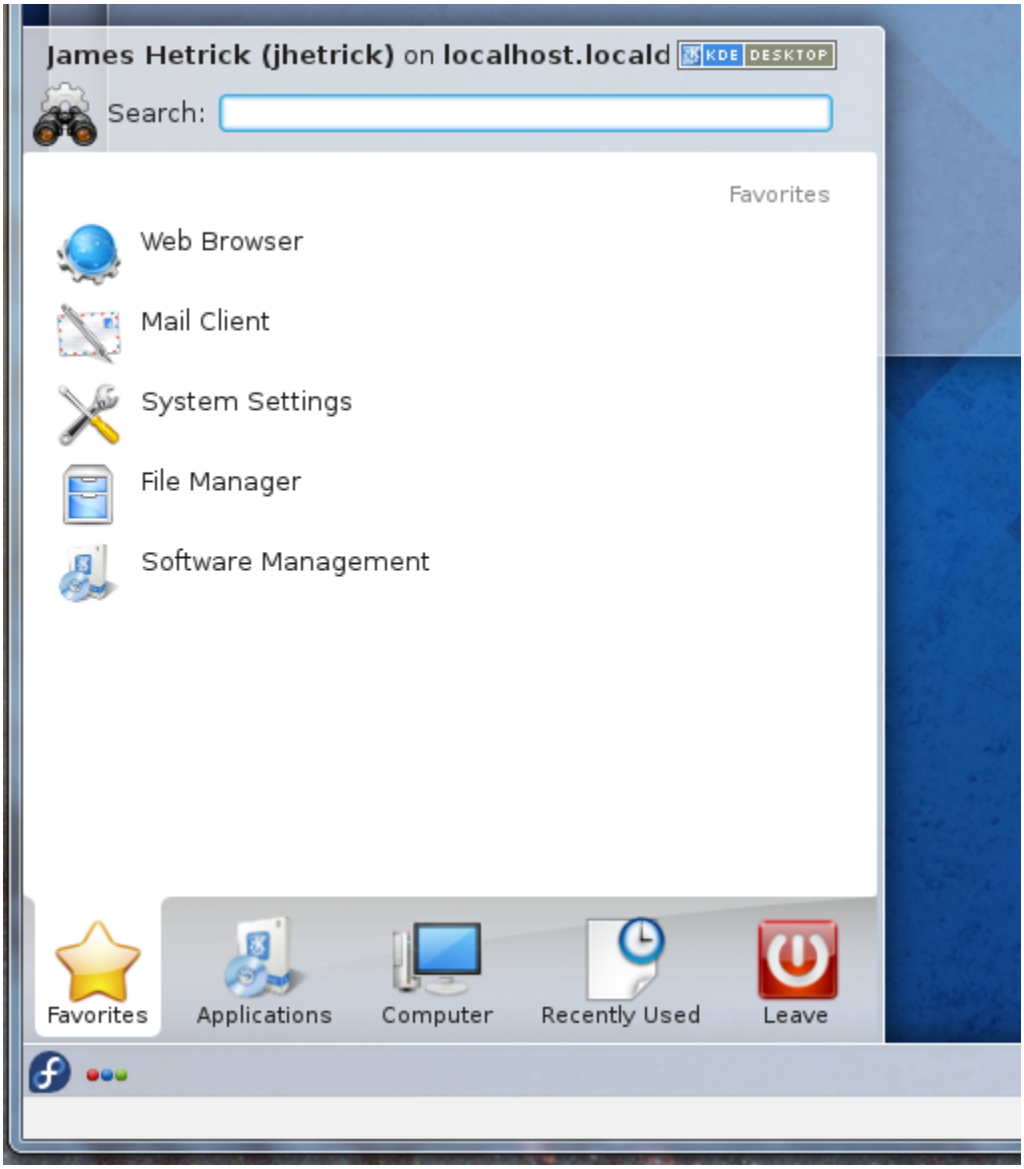
Close your *Dolphin* file explorer, by clicking the `(x)` in the upper right corner on the window.

I don't use the *GUI File Explorer* much with Linux; I find it is much easier to use the *text-based "Terminal"*, which I will describe below. So, I recommend that you **Close the Home/Trash Window** by finding and **Clicking on the X** that pops out to the right of the window:



Next: **Click** on the "*Kickoff Application Launcher*"

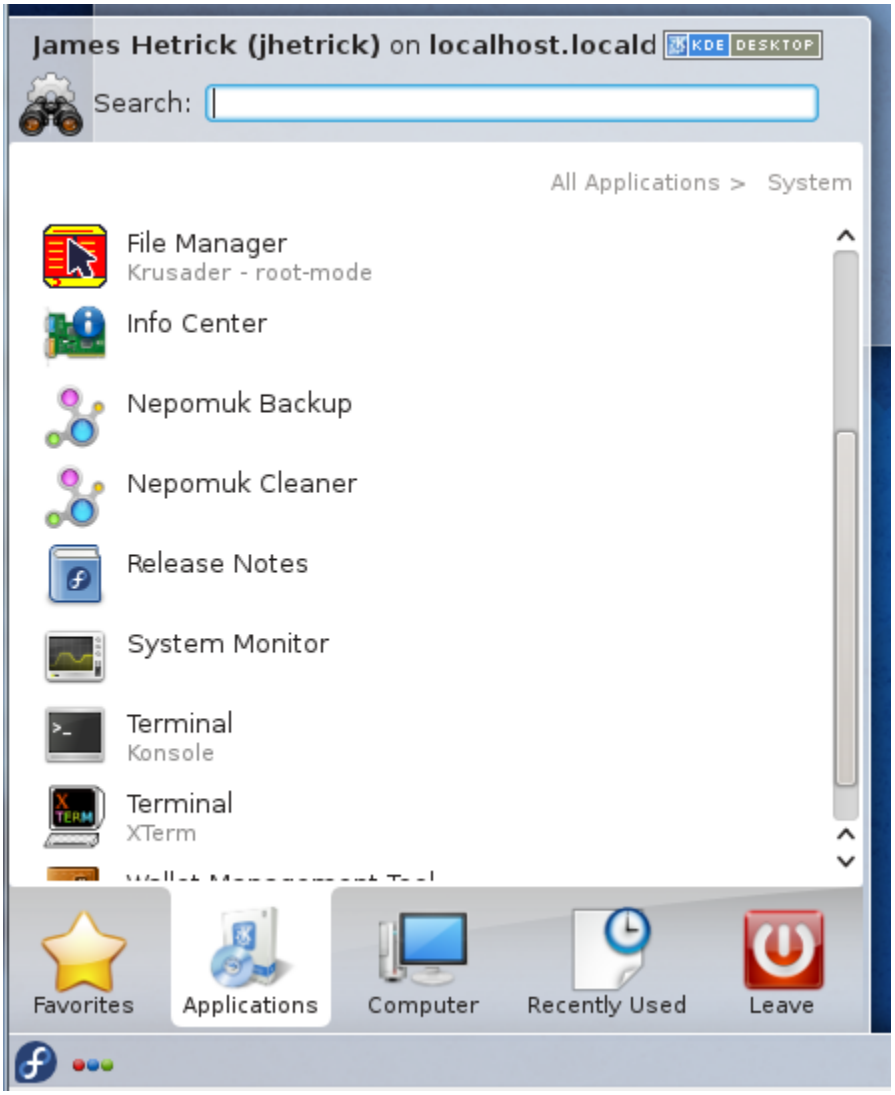
or "*Start Orb*"—as I called it earlier. It's the little `(f)` button in the lower left. This brings up a multi-tabbed menu interface.



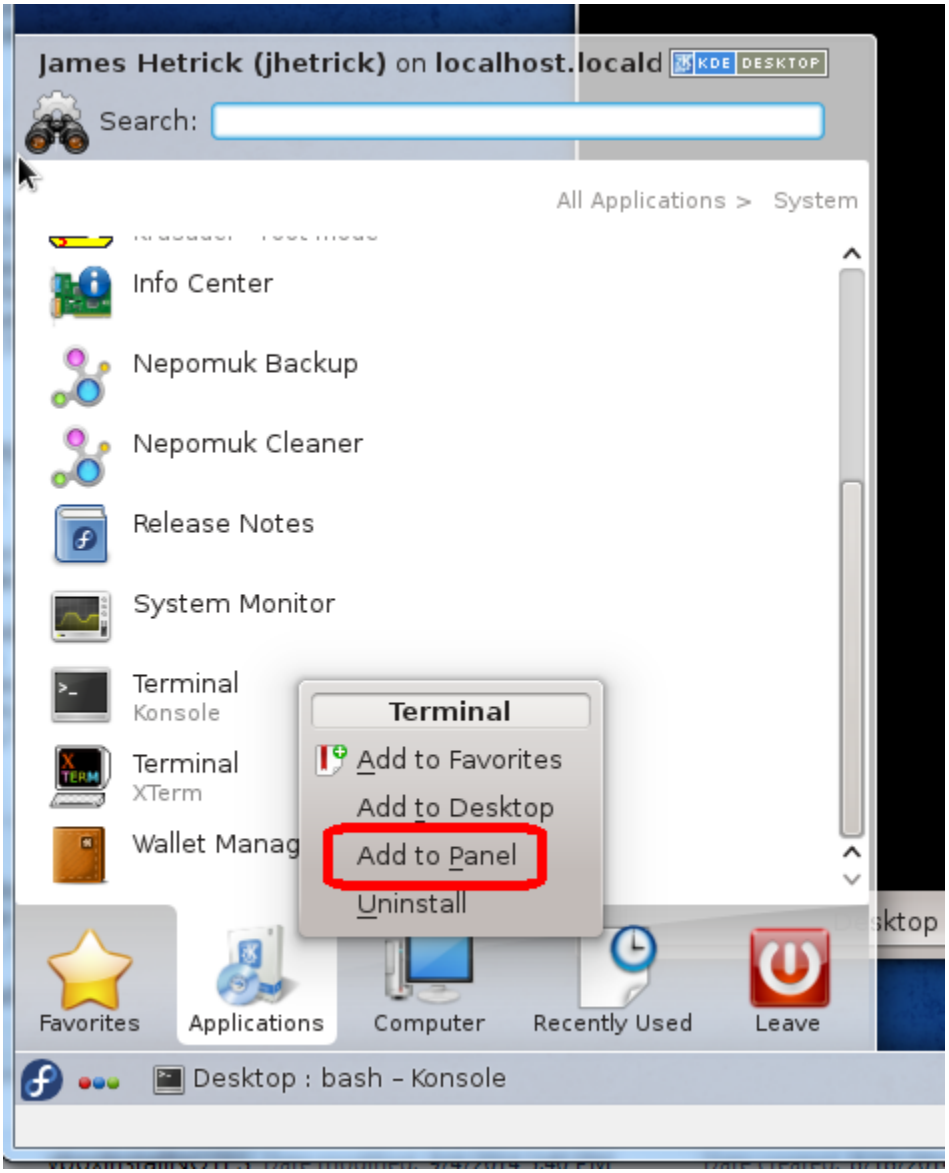
The Default tab is "`Favorites`" with some common Applications like a Web Browser and Mail Client . Later I'll show you how to configure your favorites here.

Now click on the "`Applications`" tab. You can see a large number of Categories: *Multimedia*, *Office*, *Science* and *Math*, etc. Feel free to explore—this list has all the programs and applications that came free with your Fedora Scientific installation.

In particular, click the "`System`" Category, and scroll down to "`Terminal / Konsole`". There are a couple different Terminal programs—I'm going to use the "`Konsole`" one (the name is in small print underneath the icon.

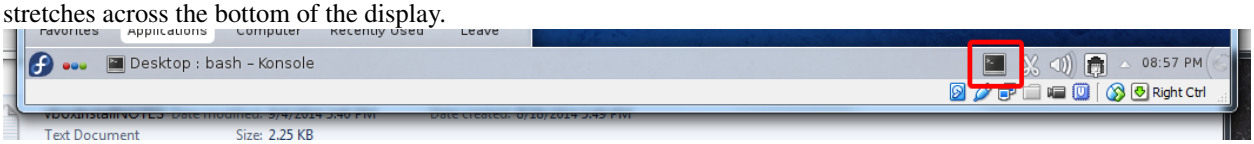


Before you Click to start it (or find it again if you already did), **Right-Click** on the "`Terminal`" icon to open the menu below.

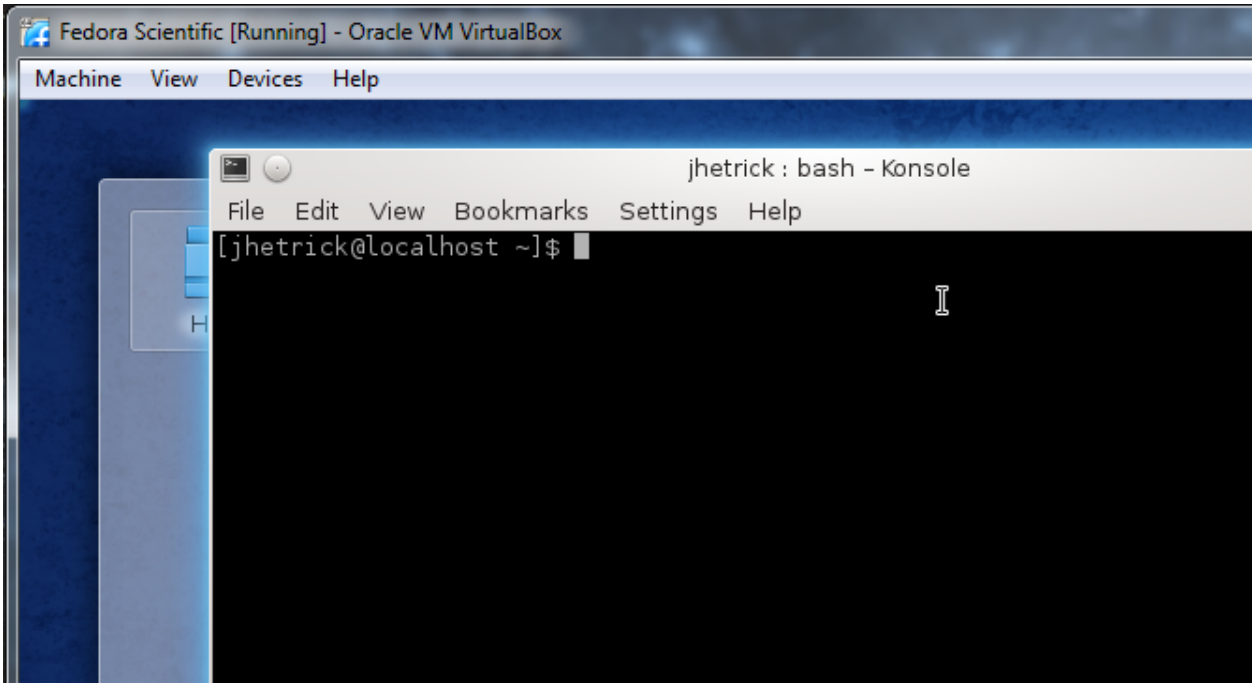


From this menu, **Click** "`Add to Panel`".

Notice that you now have a little TV/Display Terminal icon on the lower right of the "*Panel*", which the bar that stretches across the bottom of the display.



You now have a handy "*Launcher*" for a Terminal . Click the new little `Terminal` now. You will see a black window pop up, and some text ending with a "`$`".



That's what I'm talkin' about!

Welcome to the *Heart of the Machine*, my young dudes.

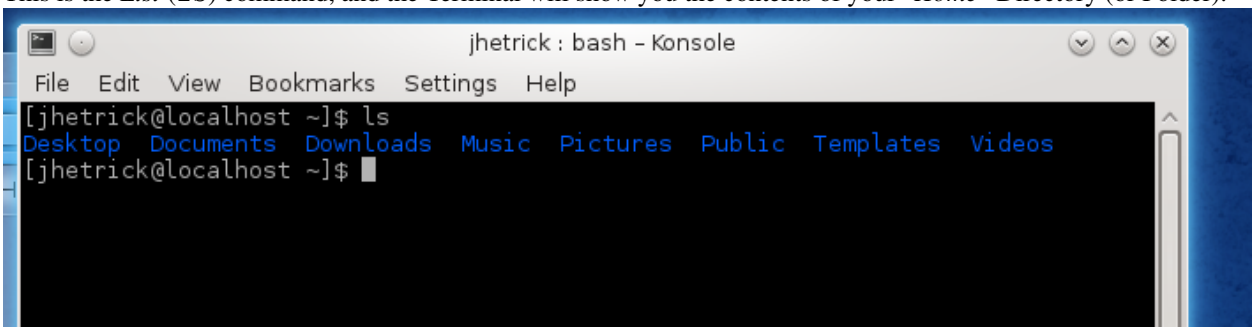
With this *Command Line Prompt* and the *Root* password, you can do anything on this machine.

Let's issue our first Unix command. Type:

ls

in the new Terminal window, followed by ENTER.

This is the *List* (`ls`) command, and the Terminal will show you the contents of your "*Home*" Directory (or Folder).



Folders (or "Directories" as folders are called in Unix) are shown in Blue . If you still have your " Home " folder open in the Dolphin File Explorer, compare the two. The " **ls**" listing should show the same directories (or folders) as you see with the graphical Dolphin explorer. Move the windows around and confirm this. Compare the output above, from the Terminal and **ls**, to the files below, from the Dolphin File Explorer window. http://dirac.physics.pacific.edu/phys/jhetrick/www/phys27/rev2014/filelist.html .. image:: imgs/finst28crp.PNG

2.16 Prepare for Guest Additions

We'll come back to learning all about the Terminal (and more importantly the "*Shell*" sometimes). For now however, we want to use the Terminal to do some administration of the system and to install the "Guest Additions". **Guest Additions** are a number of auxiliary programs and features which you add to your Linux/Desktop/VirtualBox system that make life easier.

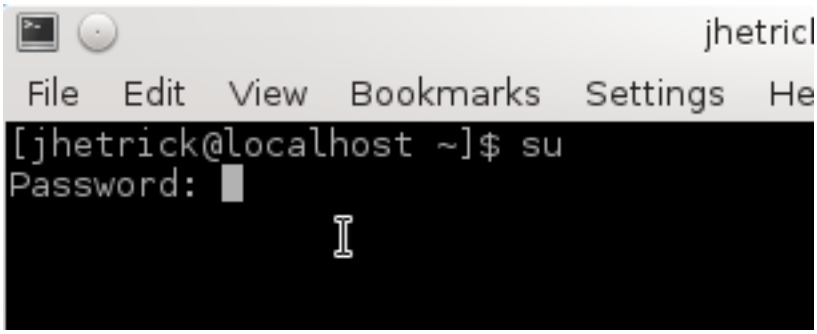
To do this, **Locate or Start a Terminal** window. If the one you had open from the above explorations is still open, use it.

At the prompt (which means, Click the Terminal and enter the following commands after the `$` sign), type:

su

Followed by ENTER. This is the *Set User* or *Super User* command (su). With no username after it, `su` changes the user to the Root (Super) user.

The terminal will respond like this: `Password:`

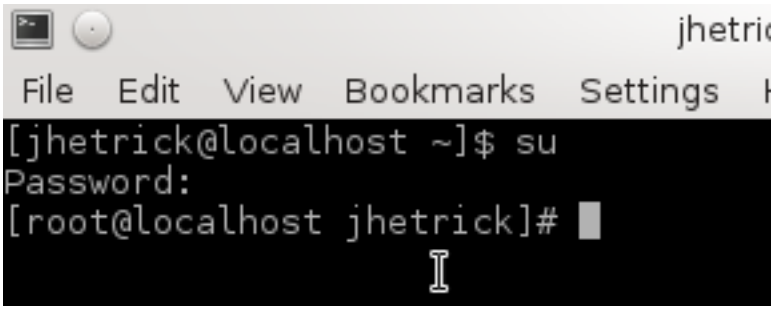


asking for the *Root password*.

Enter the password for the Root account that you created earlier.

(You did remember it, right?)

Once you have entered the password—and notice that the terminal does not echo `**********` as you type the password–the prompt will return. It looks almost the same except that the first word is now `root` instead of the `standard user`, and the last character is a `#` instead of a `$`.



You are now **root** .

If you are a fan of the movie "*Guardians of the Galaxy*", you may like to say *I am Root*, over and over.

As *root* , you do have superpowers. You can delete the entire file system with one command. So be careful. We are not going to do much.

2.16.1 Update your system

First, you really should *update* your system. This will take about 15-30 minutes, so do it when you can read a book while the update happens.

We will use `yum` (Yellowdog Updater, Modified). `Yum` allows you to very easily install and maintain packages on you linux system. There are many variants of Linux: *Fedora*, *Ubuntu*, *Suse*, *Debian*, etc. `Yum` is the updater used by *Fedora* Linux, while `apt-get` is the one used by *Ubuntu*, just in case you run across there terms out there somewhere.

In your Root Terminal window, type:

```
yum update
```

at the # prompt. Then hit **ENTER**.

Your terminal will show some text, something like this:

```
root@localhost jhetrick]# yum update
Loaded plugins: langpacks, refresh-packagekit
updates/20/x86_64/metalink                               |  15 kB  00:00:00
```

Then a huge number of Packages to be updates, like this:

```
Resolving Dependencies
--> Running transaction check
---> Package 4ti2.x86_64 0:1.6-1.fc20 will be updated
---> Package 4ti2.x86_64 0:1.6.2-3.fc20 will be an update
---> Package 4ti2-libs.x86_64 0:1.6-1.fc20 will be updated
---> Package 4ti2-libs.x86_64 0:1.6.2-3.fc20 will be an update
---> Package Cython.x86_64 0:0.19-2.fc20 will be updated
---> Package Cython.x86_64 0:0.21.1-1.fc20 will be an update
---> Package GitPython.noarch 0:0.3.2-0.5.RC1.fc20 will be updated
---> Package GitPython.noarch 0:0.3.2-0.6.RC1.fc20 will be an update
---> Package GraphicsMagick.x86_64 0:1.3.18-4.fc20 will be obsoleted
---> Package GraphicsMagick.x86_64 0:1.3.20-3.fc20 will be obsoleting
---> Package GraphicsMagick-c++.x86_64 0:1.3.18-4.fc20 will be updated
---> Package GraphicsMagick-c++.x86_64 0:1.3.20-3.fc20 will be an update
---> Package GraphicsMagick-doc.noarch 0:1.3.20-3.fc20 will be obsoleting
---> Package ImageMagick.x86_64 0:6.8.6.3-3.fc20 will be updated
---> Package ImageMagick.x86_64 0:6.8.6.3-4.fc20 will be an update
---> Package ImageMagick-c++.x86_64 0:6.8.6.3-3.fc20 will be updated
---> Package ImageMagick-c++.x86_64 0:6.8.6.3-4.fc20 will be an update
---> Package ImageMagick-libs.x86_64 0:6.8.6.3-3.fc20 will be updated
...
```

Don't be surprized if it's hundreds (even in the 1000s). After the packages stop streaming past the window, the system will pause with a message like this:

```
Transaction Summary
================================================================================
Install   20 Packages (+ 147 Dependent packages)
Upgrade  514 Packages (+1047 Dependent packages)

Total download size: 2.6 G
Is this ok [y/d/N]:
```

Yeah; I just did this on my system and it says "`Total download size:  2.6 G`". Don't worry—It won't take up that much space on your disk. Most of the packages will replace currently installed packages. Then it will erase the downloaded files.

The system is waiting for you to type: **y** to tell it to begin the *Update* process. So,

type **y**.

Go get some coffee, or your favorite beverage.

`Yum` will start downloading the packages.

```
Downloading packages:
updates/20/x86_64/prestodelta                                                | 3.0 MB
Delta RPMs reduced 2.1 G of updates to 448 M (79% saved)
(1/1728): GitPython-0.3.2-0.5.RC1.fc20_0.3.2-0.6.RC1.fc20.noarch.drpm        |  41 kB
(2/1728): 4ti2-1.6-1.fc20_1.6.2-3.fc20.x86_64.drpm                           |  20 kB
(3/1728): GraphicsMagick-c++-1.3.18-4.fc20_1.3.20-3.fc20.x86_64.drpm         |  34 kB
(4/1728): ImageMagick-6.8.6.3-3.fc20_6.8.6.3-4.fc20.x86_64.drpm              |  44 kB
(5/1728): ImageMagick-c++-6.8.6.3-3.fc20_6.8.6.3-4.fc20.x86_64.drpm          |  29 kB
(6/1728): ImageMagick-perl-6.8.6.3-3.fc20_6.8.6.3-4.fc20.x86_64.drpm         |  55 kB
...
```

You can see—I have 1728 packages to install! This will take about half an hour.

Finally, my terminal says, simply:

```
Complete!
[root@localhost jhetrick]#
```

The update is done, and you can move on to *Upgrading the Kernel*.

Warning: If you run into problems of disk space, you can do a simpler update, of just your *kernel*. To do this, type:

- **yum clean all** (to get rid of any old files taking up space)
- **yum update kernel**
- then, **shut down/restart** your VM

2.17 Upgrade your Kernel

Now, we are going to upgrade the Linux kernel to make it ready for the *Guest Additions*. To do so, type:

```
yum install dkms kernel-devel kernel-headers
```

and hit ENTER.

Some stuff will happen. Resulting in a line that says:

```
Is this ok [y/d/N]:
```

Yum is the Fedora software updater–it is waiting for you to give it the OK to install the things we typed above.

Hit **y** (ENTER).

Yum will do some more stuff (getting the software, checking dependencies, installing, updating, etc. Finally, you will see the word **Complete!**, which means that the kernel upgrade worked.

Time to reboot the machine.

Close all your open windows by Clicking on the `(x)` 's on the upper right corner of the windows. Then Click "Start" **(f)** , and **Leave** -> **Shutdown**.

Let the machine power down.

2.17.1 Run the Guest Additions Script

If you haven't done so,

- Start **VirtualBox**.
- Highlight your **Fedora Scientific** VM.
- and press the **Start (->)** button at the top to launch your Fedora VM
- **Login** as the *Standard User*

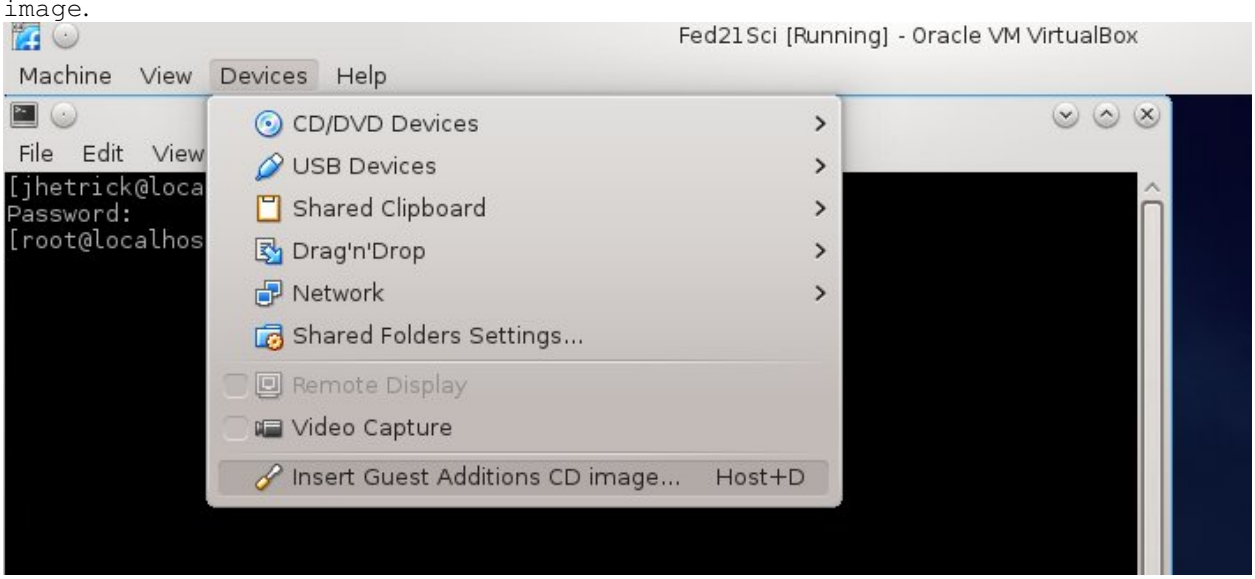
Open a Terminal window by *Clicking on the Terminal** icon on the *Panel*.

At the prompt in the terminal, become *root* by typing:

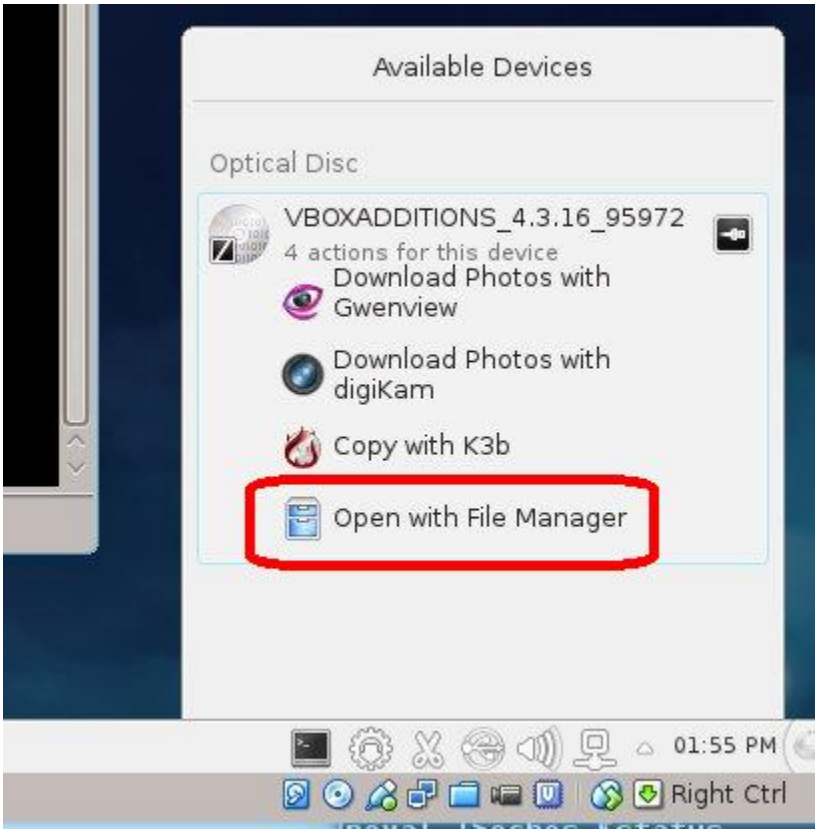
su

followed by the *root password*.

Now, At the very top of your VM window, **Click "Devices"**, and select: `Insert Guest Additions CD image`.



In a few seconds, a *Message Window* will pop up.



Click **Open with File Manager**

This will open a Dolphin File Explorer, and you can see the files contained on the Guest Addition CD .

```
If you missed the pop-up window, don't worry! (It seems to disappear after a few seconds).

Just **click** on the (f) "Start" (called the Kickoff Launcher) button,
then click the **Computer Tab** on the lower left of the screen. Find the CD name at the bottom
``VBOXADDITIONS_4.3.x_nnnnnnn``. **Click on that.

.. image:: imgs/guestAddsComputerFileMngr.jpg

This will open the Dolphin File Explorer also.
```

For some strange reason, the system will only link the `VBOXADDITIONS..` CD to the system when you open it in the *Dolphin file explorer*. So we have to do the step above.

Once the *Dolphin file browser* has opened—which links the CD to the appropriate location (as we'll see next)—you can **close** the *Dolphin* explorer, by **clicking** on the `(x)` on its upper right.

Now, return your attention to the *Terminal* window you opened. The one that has the *root* prompt: ‘‘[root@localhost jhetrick]# ‘‘.

Change directories to the location of the VBOXADDITIONS CD, but typing:

cd /run/media/jhetrick (substitute YOURUSERNAME for `jhetrick`)

then do a listing of the directory, by typing **ls** (ell ess).

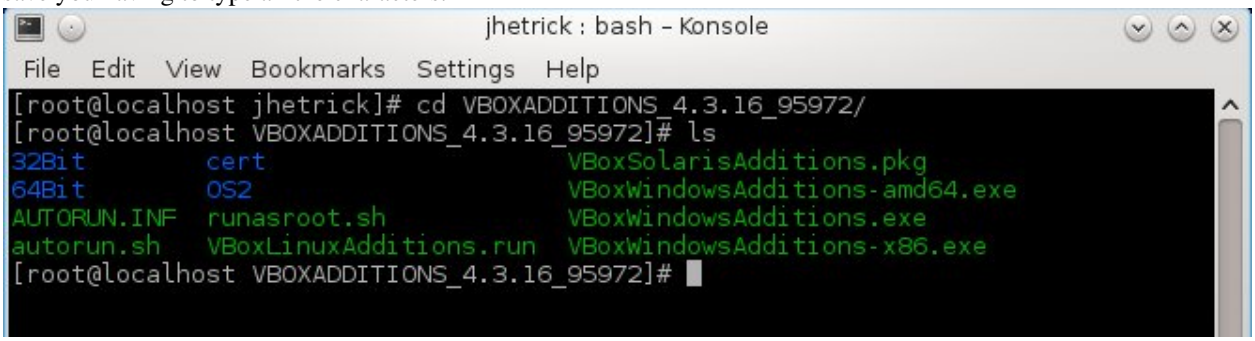
You should see the CD there as a subdirectory:

```
[root@localhost jhetric]# ls
VBOXADDTITIONS_4.3.16_95972
```

Type:

cd V(TAB)

i.e. type **cd V**, then (without hitting ENTER) hit the **TAB** key. This will *auto-complete the rest of the file name, and save you having to type all the characters.



Now type:

./VBoxLinuxAdditions.run

The shell will reply with:

```
[root@localhost VBOXADDITIONS_4.3.16_95972]# ./VBoxLinuxAdditions.run

Verifying archive integrity... All good.
Uncompressing VirtualBox 4.3.16 Guest Additions for Linux............
VirtualBox Guest Additions installer
Removing installed version 4.3.16 of VirtualBox Guest Additions...
Copying additional installer modules ...
Installing additional modules ...
Removing existing VirtualBox DKMS kernel modules           [  OK  ]
Removing existing VirtualBox non-DKMS kernel modules       [  OK  ]
Building the VirtualBox Guest Additions kernel modules
Building the main Guest Additions module                   [  OK  ]
Building the shared folder support module                  [  OK  ]
Building the OpenGL support module                         [FAILED]
(Look at /var/log/vboxadd-install.log to find out what went wrong)
```

```
Doing non-kernel setup of the Guest Additions             [  OK  ]
Installing the Window System drivers
Installing X.Org Server 1.16 modules                      [  OK  ]
Setting up the Window System to use the Guest Additions   [  OK  ]
You may need to restart the hal service and the Window System (or just restart
the guest system) to enable the Guest Additions.

Installing graphics libraries and desktop services componen[  OK  ]
[root@localhost VBOXADDITIONS_4.3.16_95972]#
```

Don't worry if you get: *Building the OpenGL support module [FAILED]*.

Now **Reboot** again...

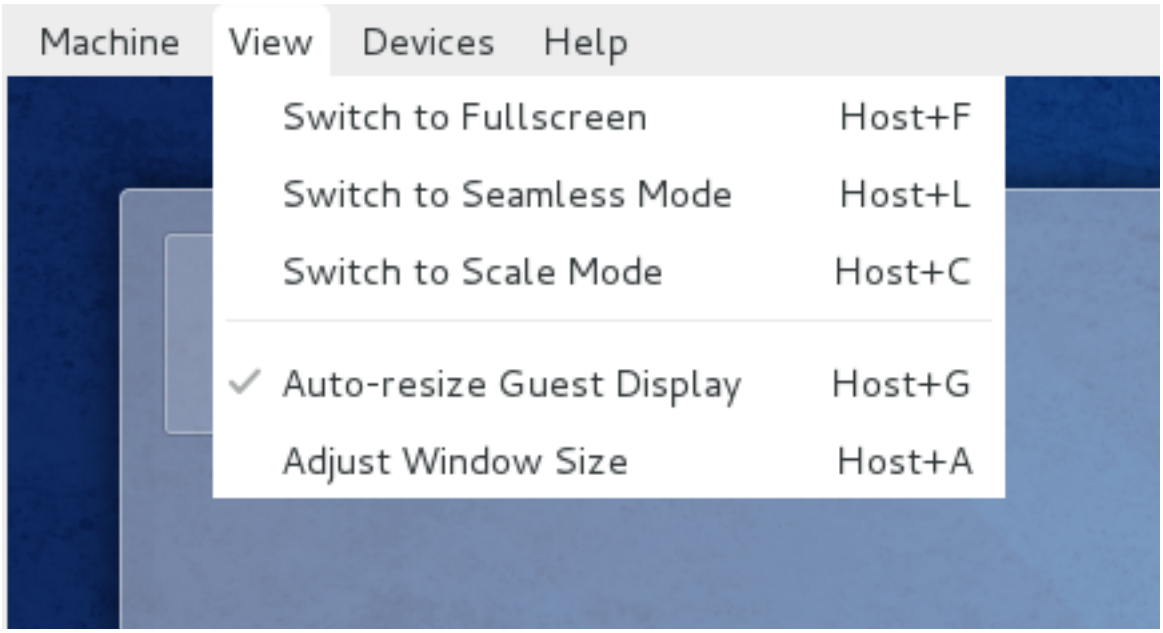
By now you should be proficient at shutting down the VM via the Start (f) Orb -> Leave -> Shutdown , and restarting it from the Oracle VirtualBox window using the **Start**(->) button.

When you see the login screen, **login as yourself** (i.e. the standard user).

Now the magic begins...

At the top of the window, you should see a row of menu items like this:

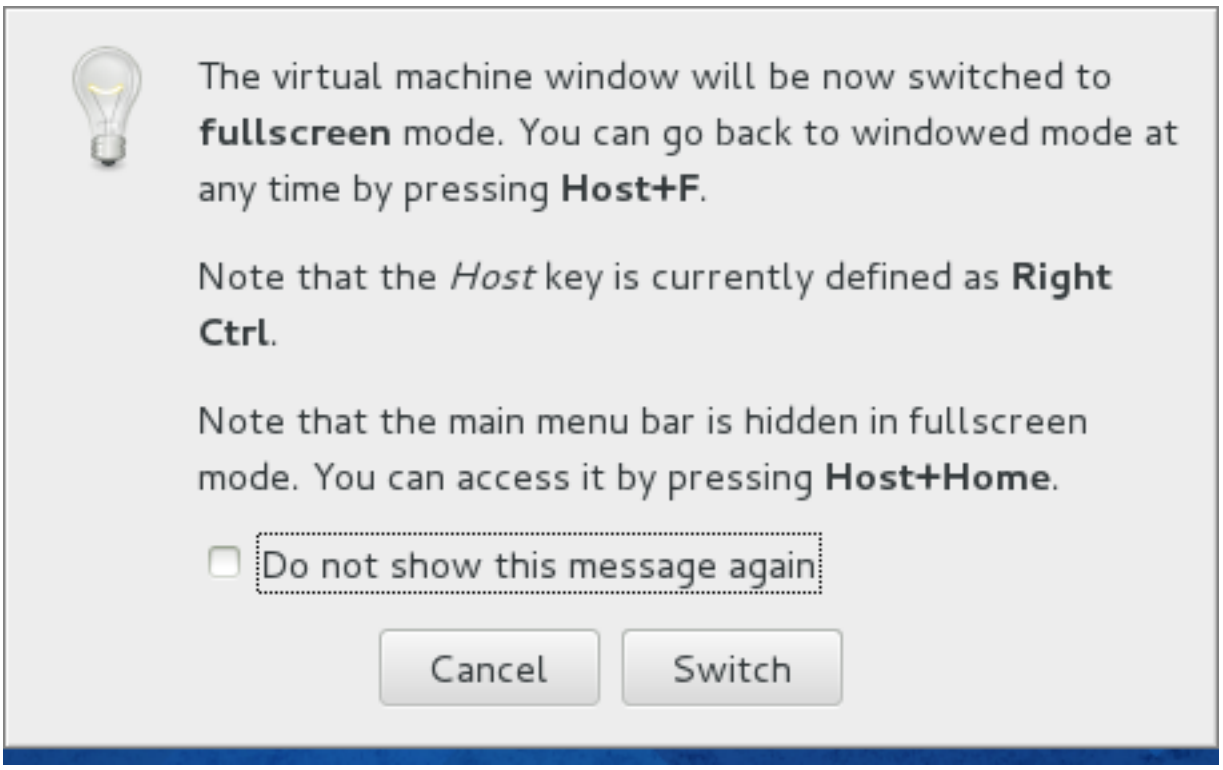
Click the `View` menu, to show these options:



One of them is obvious: **Fullscreen** mode. Let's try it out.

Click "`Switch to Fullscreen`".

This will pop up a window telling us how to recover from this little adventure.



Note: Since your VM wants the keyboard, and your "*Host*" machine (PC or Mac) wants the keyboard we have to define a "*Safe*"-key, which is called the "*Host Key*".

Of course, you can redefine this to something else if you like, but for now the default is

- On a PC: **Host Key = Right-CTRL**, i.e. the Control key on the

Right side of the keyboard.

- On a Mac: **Host Key = Left-Command** Key, i.e. the Command key on the Left side of the keyboard (the Command Key on a Mac keyboard looks like a 4-leaf clover).

Once you go *Fullscreen* there are two ways to get back to "*normal*".

- One way is to press "**Host + F**", which translates to the *Right-Ctrl* + *F* keys together.
- The other way is to hit the **Host + Home** keys together (*Right-Ctrl* + *Home*).

This pops up a little menu with Machine, View, Devices, etc., lets you select View and **uncheck** Fullscreen .

So, let's try the Guest Addition of `Fullscreen View`.

Get ready. READ THIS FIRST.

You are going to: Click **View**->**Switch to Fullscreen** at the top of your VM window.

BUT REMEMBER: To switch back from Fullscreen, Type "**HOST Key + HOME**". What's your *HOST Key* again? (**PC**: *Right-Ctrl* **Mac**: *Left-Command*)

Then CLick **View** and Uncheck "**Switch to Fullscreen**" to come back.

You probably won't be able to see this page once you go Fullscreen with your VM.

Did it work?

Next try `Seamless Mode`. This is a really cool Addition. It allows you to basically have the functionality of both OS's—your native computer and Linux—running simultaneously.

So, again: Be ready to hit the **Host + Home** keys together to come back, if you need to. At the top of the VM window, Click **View->Switch to Seamless Mode**. You should have a Panel (a long "Taskbar") at the bottom of your desktop.

Click the **Terminal** icon on the Panel.

This should open a Terminal on your Desktop.

How cool is that?!

You are now ready to start doing science with your Linux system. In the next section we will make a couple more simple alterations which will make your environment even easier to use.

2.18 Configuring the Look-and-Feel

In this section, we are going to finish setting up your scientific computing environment with a couple final items.

2.18.1 Video for this section

Video for the section is here

Note that once you start the video, a Table-of-Contents icon appears on the bottom right control bar (between the Volume and Fullscreen icons), allowing your to jump to sections in the video.

2.19 Shared Folders

Your scientific computing environment runs in its own world–a VirtualBox VM (Virtual Machine). While this system shares many things with your *host* computer at the hardare level, it doesn't have access to your Windows or Mac files.

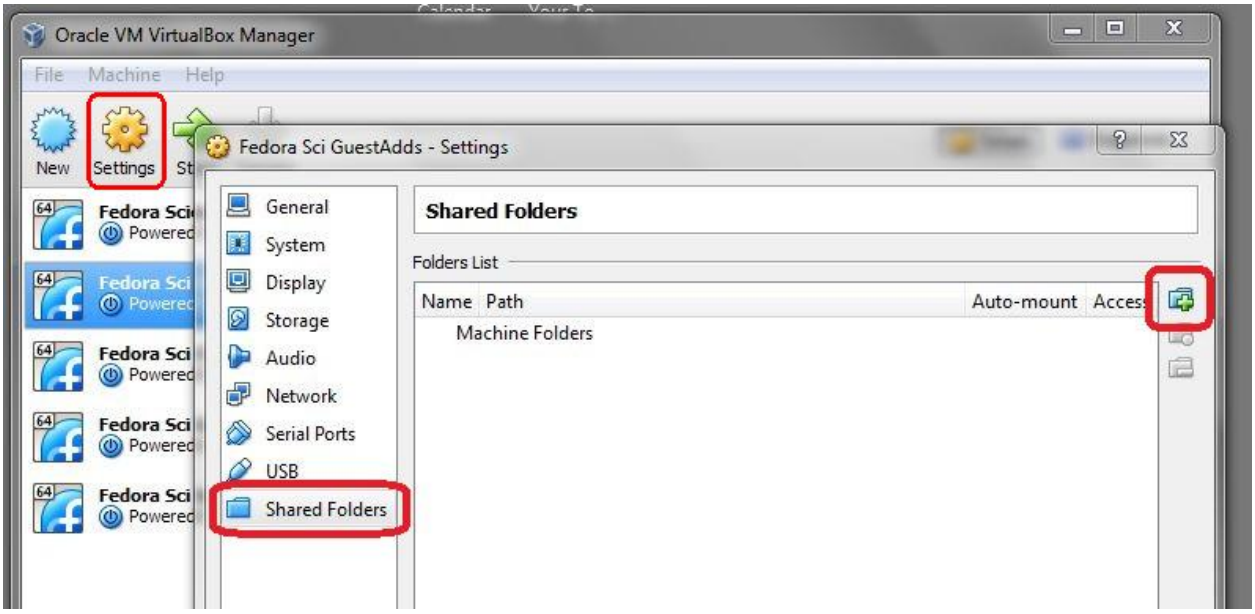
One of the nice features of installing the VBox Guest Additions is that you can set up shared folders between you VM and your Host OS. That way, you can use your computer as you normally would, say, read an email from your lab partner with an attached data file from the experiment you did together, save it to your Desktop, and then access that file in the VM and unleash your scientific unix tools on it.

The following instructions will show you how to set up a shared folders linking your "Desktop" and "My Documents" folders to your unix environment.

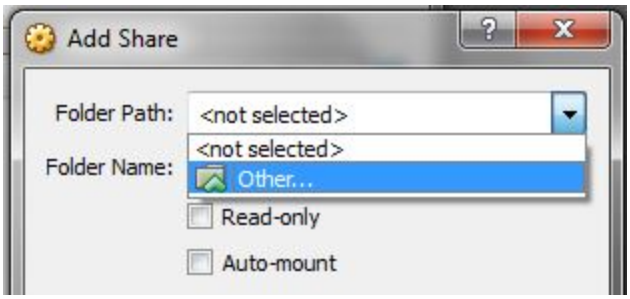
- Start the *VirtualBox* program, but **Don't Start your Fedora Scientific VM**.

Before you start your VM, highlight it, and click **Settings**, then click **Shared Folders** at the bottom of the left side menu.

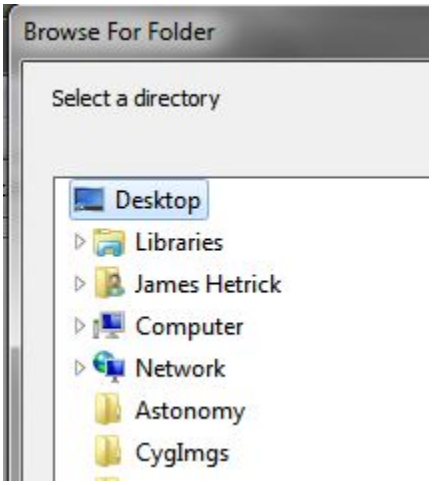
In the dialog box that appears, you should see a small blue folder and green plus on the right side; **Click the +/Folder**.



A dialog appears called "Add Share". Click the dropdown arrow at the right side of the "Folder Path:" textbox. Click the "**Other**" folder icon.

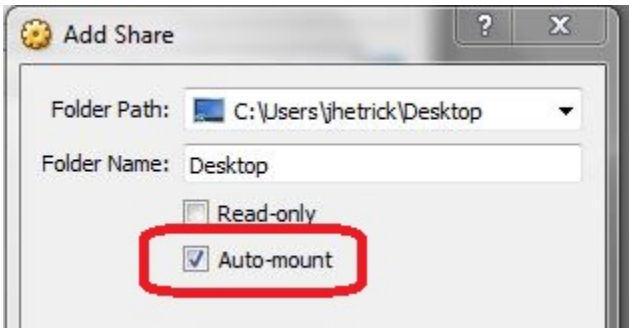


In the box that pops up, "**Desktop**" should be at the top (browse to find it, if Desktop is not at the top of this list). Highlight "Desktop" to choose it, then click "**OK**" at the bottom of the dialog box.

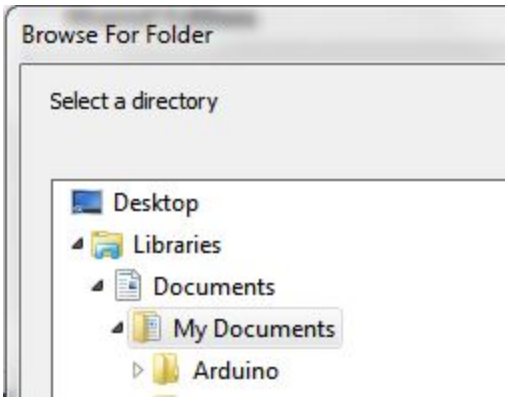


This will fill in the fields of the "**Add Share**" dialog box. Be sure that you check the **Auto-mount** check box, and click "**OK**".

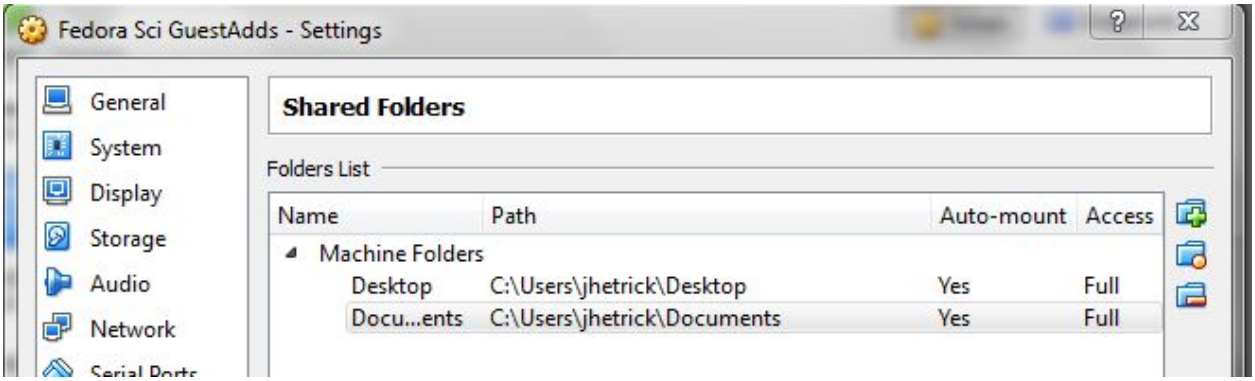
You should see the "Path" to your Desktop folder (`C:/Users/jhetrick/Desktop` for me), Yes for Auto-mount, and Full under Access.



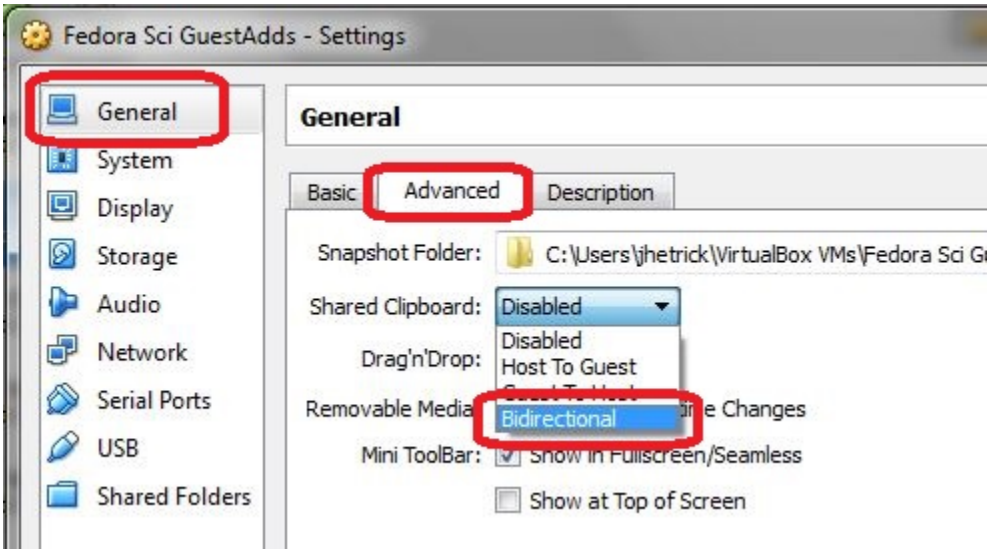
Repeat the above process exactly, but instead of choosing "Desktop", this time choose "my Documents".



Remember to check "Auto-mount". At the end, the Shared Folders dialog window should look like this:



While you still have the **Settings** dialog window open, click **General** on the left menu, then click the **Advanced** tab. On this tab, find **Shared Clipboard**, and choose **Bidirectional** from the dropdown list.



Do the same with **Drag and Drop**; choose **Bidirectional**.

This will allow you to use the Edit->Copy/Paste feature on each application window to copy/paste material from your Host system to your Linux system.

Now click OK until you are are back to the main VirtualBox program window.

- **Start your Fedora Scientific VM**, and login normally.

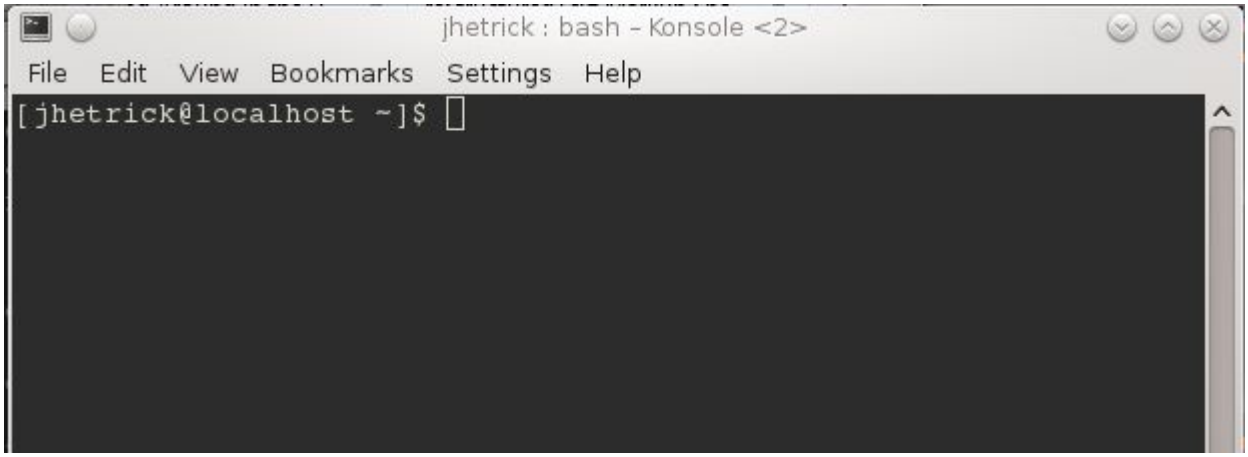
2.20 Terminal/Shell Startup Configuration: the `.bashrc` file

Click the **Terminal** icon you added to the Panel in the last section. It's the little black "TV set"  on the right of the Panel (the "Task Bar") which should be at the top of your screen.

We met the *Terminal* in the previous section. When you start it up, it opens a text-based window (i.e. not a button and menu *GUI* application; google "GUI" if you don't know what it means).

The Terminal is ancient. It comes from time before we could talk with computers, before you could touch the screen to make your choice, before we carried computers in our pockets, even before you you could point at things on the screen with a mouse.

The Terminal comes from the *Time of the Mainframe*, when the computer was an enormous Machine housed in a large room which was modified to handle the huge power consumption and wiring. On every user's desk was a *Terminal*, a little TV and a keyboard. The Terminal allowed users to sent text to the Machine, and receive text back, which was displayed on the screen of the little TV. We still have such Machines, now called *supercomputers*.



The Terminal is just the TV. The program that runs in the Terminal is called "The **Shell**". There are many different unix shells; please read http://en.wikipedia.org/wiki/Unix_shell .

In this course we will be using the *Bash* shell. The shell is a program that reads the text you enter at the *prompt*, and interprests the commands, and carries them out. In the last section, we met the shell, the prompt, and a couple of commands (among them: *ls*, *pwd*, *cd*, and *su*).

The *Prompt* is the small line of text that looks like this:

[jhetrick@localhost ~]$

This is the shell telling you that it's waiting for you to type a command and hit ENTER. In fact, we'll change this rather generic prompt soon. For now, we have a few commands to issue in order to complete the customizations, before we study the shell in more detail.

When you first start the Terminal, which then starts the *Shell*, it looks for a special, *hidden* file, named **.bashrc**. This is a simple text file where you place configuration commands that will customize the behavior of the shell for you, as well as extend its functionality. For example, most people put *aliases* (shorthand names) for common commands.

I have prepared a simple *.bashrc* file for you with a few configuration lines. Once you feel proficient at unix, you can edit and change this file to your liking. Searching the web, you can find many (complex!) examples of *.bashrc* files that people use and have shared.

To use my *.bashrc* file, you must first *get it*.

We first have to get this file onto your VM. So, of course, we'll just download it, but *not* through a browser.

I'm assuming you have have your VM up and running, and you are logged in. If not, do so.

Now, let's see an example of how powerful the shell is.

Open a Terminal and at the prompt, copy/paste (or type) the following text (exactly):

```
curl -O dirac.physics.pacific.edu/phys/jhetrick/www/PHYS27/.bashrc
```

Note: If you set up *Bidirectional Clipboard Sharing* (above), you should be able to simply cut and paste the link above into the Firefox browser textbox.

Here we are using the program **curl** (Command line URL), which when you hit ENTER, goes out into the internet and grabs the URL you asked for. It is insanely powerful. The **-O** switch (more about *switches* later) lets you configure curl's behavior at the command line. If you want you can tell curl to get a webpage , and then read that webpage for URLs, get those, and then read those...etc. Yeah, that's pretty powerful.

If things worked, `curl` should have responded with something like this:

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1716  100  1716    0     0  83958      0 --:--:-- --:--:-- --:--:-- 86052
```

In particular you should see the Total Downloaded Size as 1716 bytes (second number).

Click the [X] button on the Terminal window to close it. Then, click the Terminal icon on the panel to open a new Terminal.

Now when the shell starts, it reads and executes the `.bashrc` file, and you should see a little message and a new prompt, like this:

```
Initializing from ~/.bashrc
     >- path
     >- prompt
     >- aliases
User jhetrick - Welcome to the Machine
sci[~]>
```

Now we are ready to to do some science!

2.21 Examine `.bashrc`

First, let's look at the file you just grabbed. To do this, we'll use **less**, a file pager/viewer. We'll learn more about this and other commands in the next chapter, but for now, just type

```
less .bashrc
```

at the prompt in your shell window, then ENTER (from now on, if I say "*type something at the prompt*", you will know that you follow that with ENTER, right?).

This will display the contents of the file `.bashrc`, which should look like the following.

To "*page through*" the file, go **forward** to the next page of text by hitting the **SPACE** bar. To go back and view the **previous** page, hit the **b** key.

```
#
# .bashrc initialization file
#
# This file created by Dr. J. Hetrick
# for PHYS 27/193 Introduction to
# Scientific Computing
#
# v. 4.0 Nov 2014

# Source global definitions
if [ -f /etc/bashrc ]; then
   . /etc/bashrc
fi


echo "Initializing from ~/.bashrc"

# --- setup PATH to executables
# if you want to add a directory to the
# PATH you add it at the end, with a :

export PATH=".:$PATH:$HOME/bin"

# ----------------------------
echo "     >- path"
```

```
# --- setup BASH SHELL --------
# You can set the prompt to something useful by
# setting the variable PS1.
# \h means "Hostname" (the name of your computer
# \w means "working directory"

# This prompt has your hostname in it, useful
# when you are on different machines and want to
# know which one you are working on. However I've
# "commented it out" of this file by putting #
# in front of it.
#PS1="\h[\w]>"

# We can use this prompt for our Scientific VM
# It will have the working directory inside the []
# and I like a > for a prompt.

PS1="sci[\w]>"

# ----------------------------
echo "     >- prompt"


# --- Put aliases here -------

alias l='ls -F'
alias la='ls -aF'
alias ll='ls -lF'
alias lla='ls -alF'
alias lsd='ls -lF | grep "^d"' # only directories
alias m=less
# this function lists the 20 most recently changed files
function lt() { ls -lFt "$@" | head -20 ;}

# ----------------------------
echo "     >- aliases"


# ----------------------------
# comment this out if too cheezy for you :)
# by putting a # at the front. Notice it
# echos the value of the USER environment variable

echo "User $USER - Welcome to the Machine"
```

First, every line that begins with a hash (**#**) is a comment. It will not be read as a command, but rather is there solely for the reader.

After the top comment block describing the file, there is an **if** statement:

```
if [ -f /etc/bashrc ]; then
   . /etc/bashrc
fi
```

This command effectively says "*if there is a system bashrc (``/etc/bashrc``) file, read and execute its commands first*".

Next comes some comments about the *PATH*, and this command:

```
export PATH=".:$PATH:$HOME/bin"
```

This command adds the current folder (`.`) and a `bin` subfolder to the *PATH*–a colon (:) separated list of folders in which the shell should look for executable programs.

Next, we change the prompt to something friendlier. The default prompt is: `[user@localhost ~]$`.

I prefer the following:

sci[~]>

What this does is let you know that you are currently logged into your Scientific environment (which we're going to call **sci**). Since I often login to other machines around the world via the terminal (and start a shell on that machine), I always put the same `.bashrc` file on the remote machine, except that I change the name in the prompt to the name or location of that machine, like `dirac[~]>` or `hawaii[~]>`.

Additionally, this prompt shows that you are currently working in your **Home** *directory* (or folder), which has a nickname: **~** (the tilde).*Folders* (where files are stored) in Unix are called **Directories**. I will use the two terms interchangably, and you will get used to it. Just remember:

Note: Directory = Folder

If you *cd* to another directory, such as your *Pictures* directory (do this now, by typing `cd Pictures`), you should see that the prompt changes to

sci[Pictures]>

Did it? To move back to your *Home* directory, all you have to do is type `cd` (+ ENTER), and your prompt should return to

sci[~]>

In this little exercise, in addition to learning how to move around a little, you've learned that the tilde–in the context of directories–is a shorthand name for *Your Home (or Default) Directory*.

Note: **~** = your Home (default) directory

After setting the prompt (by defining the variable *PS1="sci[w]>"*), the `.bashrc` file contains **alias** definitions.

```
alias l='ls -F'
alias la='ls -aF'
alias ll='ls -lF;
...
```

These are short hand names for common commands. As you already learned the `ls` command lists the contents of a directory. You can change it's behavior by adding **switches**, various letters prepended by a minus sign (like -F, -alqF, -O, etc.).

For example, at your prompt, type `ls`.

Now type a single `l` (*ell*). It should show roughly the same thing, except that in the latter case, you typed the *alias* for `ls -F`. The `-F` switch tells `ls` to "format" the output, by adding a trailing slash to directories, so that it is easier to distinguish them from simple files.

Now type `ll` (*ell ell*). This version gives much more detail: permissions, owner, group, size, date, etc. This is called a "long listing" and hence the alias `ll`.

Similarly, `la` will show *ALL* files, including hidden ones that start with a dot. Try it: type

```
la
```

2.22 Fixing the Shared Folders

When you booted your VM, the Shared Folders we created at the beginning of this document were "*mounted* (attached to the file system) in a special system directory, `/media` to be specific, and they were created by the root user. We have a couple things to do to make life easier.

First, in your Terminal, type

```
cd /media
```

then

`ll` (*ell ell*) to do a long listing the contents of above directory.

You should see

```
sci[/media]>ll
total 56
drwxrwx---. 1 root vboxsf  8192 Nov 10 10:19 sf_Desktop/
drwxrwx---. 1 root vboxsf 49152 Nov 10 16:18 sf_Documents/
```

This tells us that there are two directories: `sf_Desktop/` and `sf_Documents/`. I like to use the trailing `/` to help you get used to that meaning these are directories, not files.

the first part of each line for these directories shows the type of object (file, directory, or other) and the permissions.

```
drwxrwx---
```

means that this thing is a directory (**d**), and that the *Owner* (denoted by the first *rwx*) has **r**ead, **w**rite, and e**x**ecute privledges. The second *rwx* says that the *Group* has r,w,x, privledges. This means that, like the Owner of the object, anyone in the file/directory's *Group* can read, write (i.e. modify), or execute (in the case of a program) that file. For Directories, *x* permission means that that group or person can *cd* into the Directory.

We then see that the *Owner* is **root**, and the *Group* for the file is the **vboxsf** Group.

```
drwxrwx---. 1 root vboxsf  8192 Nov 10 10:19 sf_Desktop/
```

Try to *cd* into the `sf_Desktop` directory. Do:

```
cd sf_Desktop
```

You should get an error telling you that you don't have the correct permission to do so.

```
sci[/media]>cd sf_Desktop
bash: cd: sf_Desktop: Permission denied
```

Ouch!

One solution:

You could simply become the *root* user everytime you want to *cd* into this directory. You would first do `su` (change to superuser account), issue the root password, then as root, `cd sf_Desktop`, but this is way cumbersome.

A better solution:

It's far easier just to add you to the **vboxsf** Group. Then you would have the correct privledges to *cd* into `sf_Desktop`.

However, only *root* has high enough privledges to edit your Group affiliations on the Machine. So, to make this change, you *do* have to become *root*. But just this once.

The best solution: Do this:

In some previous exercises, you became root by typing the `su` command, and giving the root password. This basically stops your *shell* program (the one that started with you as the user when you clicked the *Terminal* icon), and starts a new *shell* in the same Terminal, but this time logged in as *root*. You get a prompt like this: `[root@localhost /media]#`, and you have complete control of the machine.

There is a better way to do administrative activities which require *root* access, by using the **sudo** (super user do) command. The difference is that with the root shell (and a prompt `[root...]#`) anything you type will be carried out as the root user.

Using *sudo*, you are only able to issue one command at a time. When we set up your user account, we checked "*Make this user an Administrator*" (didn't we...??).

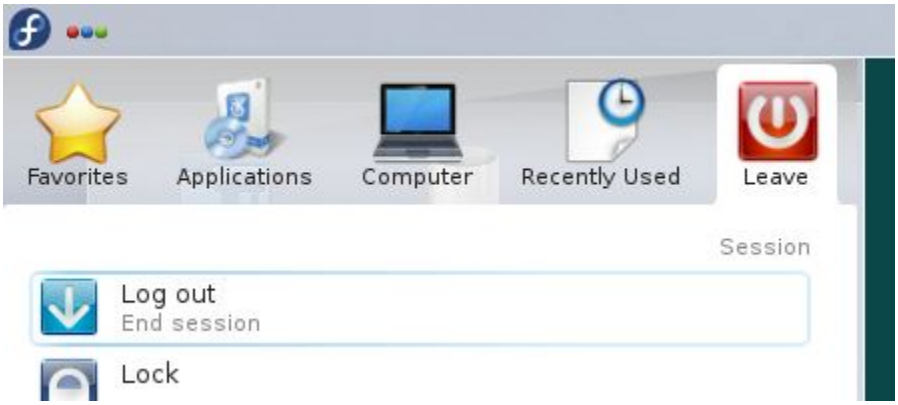
Type the following, substituting your username for YOURUSERNAME (my YOURUSERNAME = jhetrick, so I would type `sudo usermod -aG vboxsf jhetrick`):

```
sudo usermod -aG vboxsf YOURUSERNAME
```

The **sudo** command will first ask to be sure that you are you: **it will ask for YOUR password** (not the *root* password). This is so that, while you were out in the kitchen grabbing another beer, your evil roomate does not come in, and seeing that you have an open Terminal and Shell running, issue a *sudo* command to erase your hard drive (it *could* happen).

This command then adds you to the Group **vboxsf**, but the change doesn't take effect until you login again

Instead of shutting down your whole VM machine and rebooting, you only need to **Logout**, then *Login* again.



Go ahead and Logout/Login. Then open a Terminal.

In your fresh Terminal window, *cd* to the `/media` directory

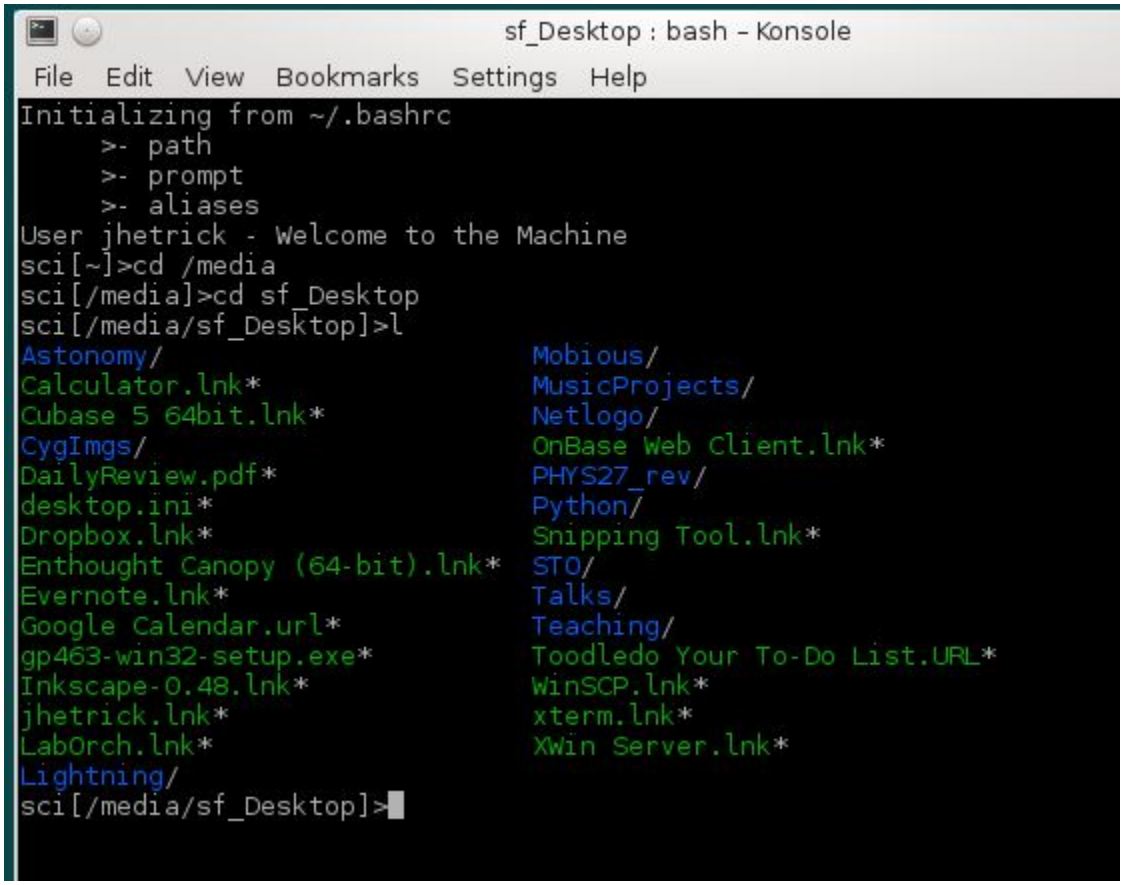
```
cd /media
```

List the contents of the directory with an `l`.

Now try to cd into the `sf_Desktop` directory.

Do a listing there by typing `l`

Do you see the things on your Desktop in that list? Here's mine.



While you have your Terminal running, type a lone `cd` (no target directory after it) to get back to your Home directory.

Here we want to make two "*Symbolic Links*" to the Shared Folders we made. Then you won't have to cd to the /media directory every time. There will be shortcuts in your Home directory. To make these, type the following commands in the terminal, in your Home directory.

```
ln -s /media/sf_Desktop winDesktop
```

```
ln -s /media/sf_Documents winDocs
```

(if you are on a Mac, you can use "macDesktop" and "macDocs")

These lines start with `ln` (lowercase L N, for link), and the command makes shortcuts called `winDesktop` and `winDocs`. Do an `l` to list the contents of your Home directory. Do you see these links? How are they designated so you can tell them from normal files?

Do `cd winDesktop`

Then `l`

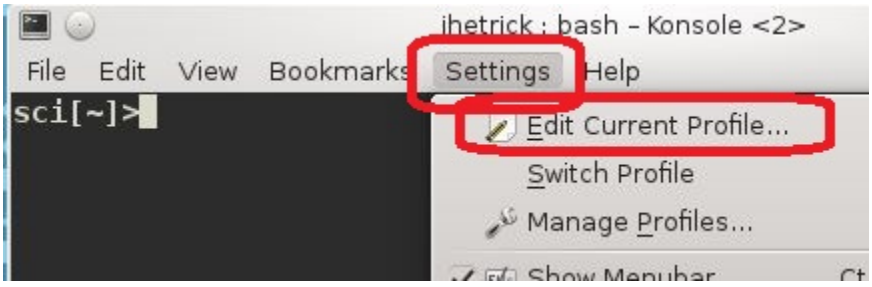
Do you see your Windows (or Mac) Desktop contents?

```
sci[/media/sf_Desktop]>cd
sci[~]>ln -s /media/sf_Desktop winDesktop
sci[~]>ln -s /media/sf_Documents winDocs
sci[~]>l
Desktop/    Downloads/  Pictures/  Templates/  winDesktop@
Documents/  Music/      Public/    Videos/     winDocs@
sci[~]>cd winDesktop
sci[~/winDesktop]>l
Astonomy/                       Mobious/
Calculator.lnk*                 MusicProjects/
Cubase 5 64bit.lnk*             Netlogo/
CygImgs/                        OnBase Web Client.lnk*
DailyReview.pdf*                PHYS27_rev/
desktop.ini*                    Python/
Dropbox.lnk*                    Snipping Tool.lnk*
Enthought Canopy (64-bit).lnk*  STO/
Evernote.lnk*                   Talks/
Google Calendar.url*            Teaching/
gp463-win32-setup.exe*          Toodledo Your To-Do List.URL*
Inkscape-0.48.lnk*              WinSCP.lnk*
jhetrick.lnk*                   xterm.lnk*
LabOrch.lnk*                    XWin Server.lnk*
Lightning/
sci[~/winDesktop]>
```

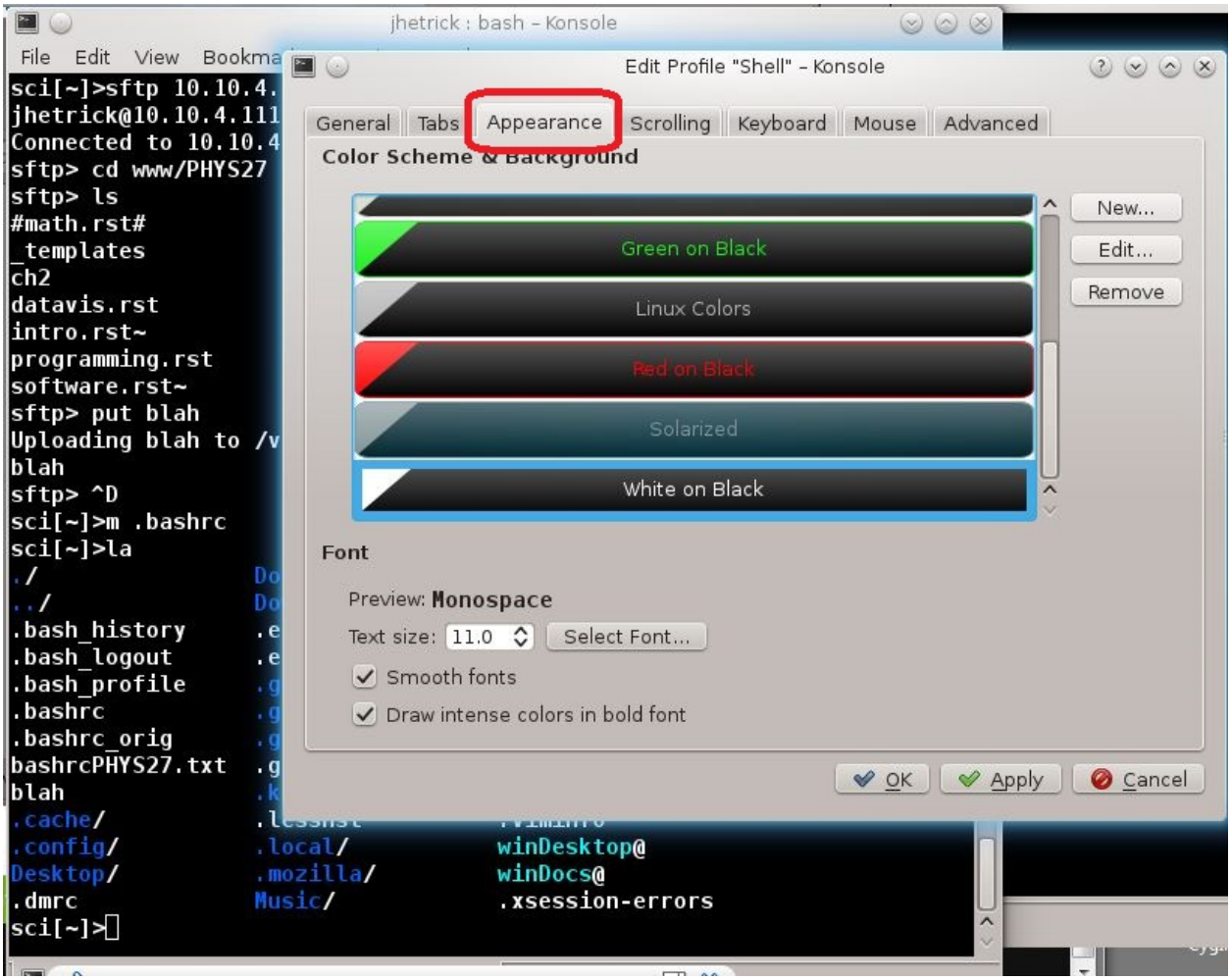
2.23 Fonts and Colors

Finally, I like to play with the font and color a little. This customization is completely up to you.

To change the font that the Terminal uses, go to the very top of the window and click "*Settings*", then click "*Edit Current Profile*".



You can change almost everything about the Terminal, and I recommend staying away from everything but font and color.



Click the "*Appearance*" tab and play around with the options. You can always click "Cancel" to go back to what it was before. If you move the "Edit Profile" window to the side, you can see the effect of your choices on the Terminal. Personally, I like a **bold**, *11 point, Monospace font*, in a stark White-on-Black color scheme. But my eyes are old and need the print to pop.

You are now ready to start using your scientific computing environment.

2.24 Homework

Homework 2 is here

CHAPTER

THREE

BASIC UNIX SKILLS

3.1 Moving Around in the Unix World

3.1.1 Video for this Section

Video for this section is not available yet.

3.2 Get Some Files

In order to learn about unix files and directories, let's get some files and directories that we can explore together.

To do this we will use the `tar` command. `Tar` is an archiving tool for packing up a set of files *and* directories for portable "shipment", and you will often see a, so-called, *tarball* (an archive file in the *tar* format) for download on the web.

A **tar** archive is really just all the files "glued" together, one after the other, with some header information. Usually, this file is then compressed in size using the **gzip** method. A similar way to do both of these tasks (archiving several files into one and compressing the archive) is the **zip** format, which creates files that end in `.zip`.

You can download the tar file for our course by copying this link, pasting it into a shell terminal window at the prompt, and hitting ENTER.

```
curl -O dirac.physics.pacific.edu/phys/jhetrick/www/PHYS27/unixplay.tar.gz
```

Once you have downloaded the file, do a directory listing and see if you see the file **unixplay.tar.gz**.

If so, you can unpack the archive by typing (or cutting and pasting) the following command at the prompt:

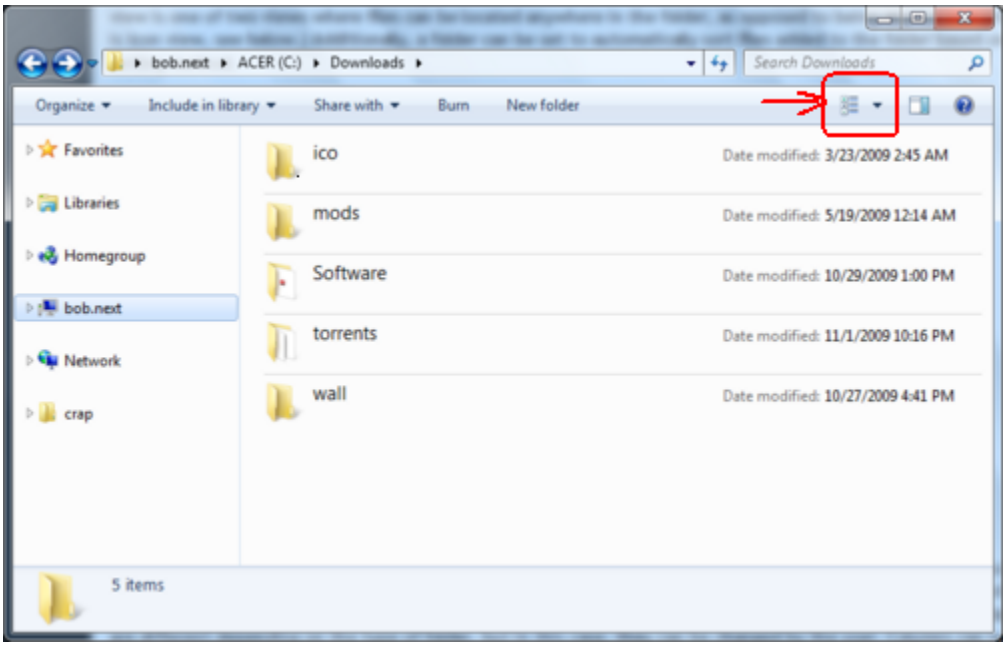
```
tar zxvf unixplay.tar.gz
```

Now do a listing and verify that this command has created a new directory called `unixplay/`. We'll come back to these files shortly.

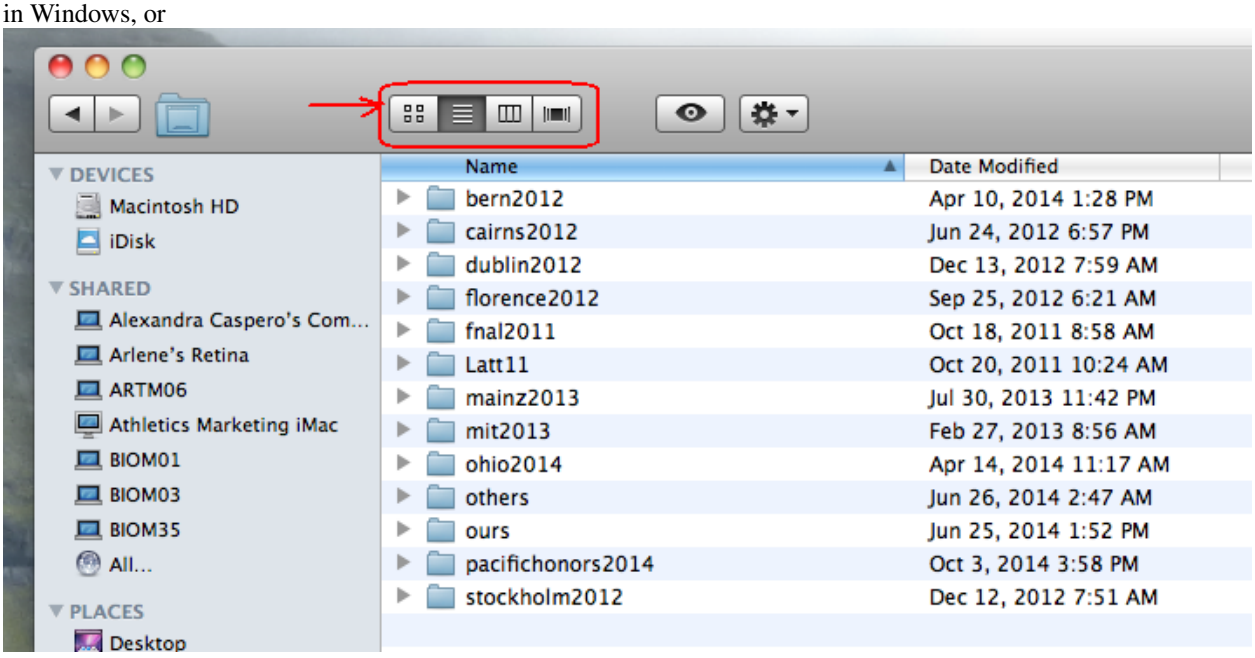
3.3 The Unix File System

Now that you have your Fedora system running and have some files to explore, let's learn to move around a bit. We've got to get used to doing things through a text based shell window. While that might seem extremely limiting at first, it's actually *much* more efficient, especially when you are connecting to another computer (like a supercomputer on the other side of the country). A shell terminal uses very little bandwidth since it only needs to transfer characters instead of pictures and mouse movements.

You are probably used to the **Point-and-Click** (GUI, or *G*raphical *U*ser *I*nterface) environment of Windows and Macs: when you want to do something, you (double) click on its **icon*, possibly enter some parameters (a file to edit for example), and then work within that appliction. At the end you might save your work to a file, or print it. The other big difference is that to find your files, you use a Graphical File Browser , like the one below. (We'll come back to the boxes marked with an arrow in just a minute)



in Windows, or



on a Mac.

To find files and folders, you navigate your way around by clicking on icons, like this

Note: In Unix, *folders* are called **directories**.

That's why the Unix command to move to a new directory is called **cd**, for *Change Directory* . This command moves you to another folder–which I will henceforth call by its proper name, a **directory**.

Using a Graphical File Browser, you usually see all the files and subfolders as little icons which often represent the type of file. You probably know how to change this behavior.

In your **Host** OS–either *Windows* (PC) or *OSX* (Mac), open some Folder on your computer with many files.

Click the red box marked on the images above, on the top menu bar.

Choose to simply *List* the files, or show the *Details*. Also you see that Folders are shown with a small Folder icon so that you can distinguish these from regular files.

Notice that with *Details* you get size information, type, and date of last modification.

Now change the setting back to what it was before (or a new one if you like).

We will do something very similar to this in a Unix shell window, listing files from the prompt in a text-based shell window. You will soon find it is almost as easy as the GUI way, and in many cases more powerful.

Open an shell/terminal window.

There is a default directory for each user, called the **Home** directory.

You may recall from the last chapter that the "full name" of your **Home** directory is ''/home/YOURUSERNAME" .

When you start a shell , or if you login to a computer where you have a unix account, the shell will give you a *prompt*, and place you in your **Home** directory. This is where your init files are, like ''.bashrc" and others.

In fact, the way I have set up your system, the prompt tells you this.

A short hand symbol for your HOME dir is the symbol "**~**" (the tilda). Notice that in your prompt there is the following: **[~]**, meaning that you are working in your HOME dir (I will often say "*dir*" for directory).

3.3.1 Changing to a different directory

We've already learned about this, but mention it here again for completeness.

cd (change directory)

The command **cd** means change the *current working directory* to 'directory'. The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree.

Let's change to the `unixplay/` directory that you have just made by unpacking the tar file.

Note: I'm referring to the directory as `unixplay/` with a `/` at the end. This is a unix-guru way of saying that *unixplay* is a directory (by writing it with an ending `/`. However you don't have to add that `/` when typing it in the the `cd` command.

Type the following:

```
sci[~]>cd unixplay
sci[unixplay]>
```

Type `ls` (or better, the *alias* `l`) to see the contents. You should see

```
sci[~/unixplay]>l
anotherFile  dirTWO/            gammaData/  scifi_list.txt  yetanother
dirONE/      file_unixplay.txt  ints.dat    testfile
```

3.4 The Directories . and . .

Still in the `unixplay/` directory, type

`la` (*alias* for `ls -a`)

As you can now see, in the unixplay/ directory (and in all other directories), there are two special directories called (`.`) and (`..`)

In UNIX, "`.`" means the current directory, so typing

```
cd .
```

means stay where you are (the unixplay/ directory).

This may not seem very useful at first, but using "`.`" as the name of the current directory will save a lot of typing, as we shall see later. We also need the `.` when trying to copy or move files to the current directory.

"`..`" means the parent of the current directory, so typing

```
cd ..
```

will take you one up one level in the directory tree.

Try it now.

In your `unixplay/` directory, type:

```
cd ..
```

This should move you back to your home directory and you prompt should show this by showing **[~]**.

Now `cd` back down to the `unixplay` directory.

You should see **[unixplay]** in the prompt, helping you to remember where you are.

3.4.1 Going Home

Typing `cd` with no directory argument always returns you to your *Home* directory. This is very useful if you get lost in the file system.

Try it

If you are in your `unixplay/` dir, `cd` down one more level into `dirONE/`.

Note: In Unix, names are *case-sensitive*, meaning that the directory `dirONE` and `dirone` are different.

Now, type `cd` by itself. This takes you back to your *home* directory.

Note: Wherever you are, typing a bare `cd` command will take you *Home*.

Let's see how far up in the unix file system we can go.

Type `cd` to start from your home directory. Does your prompt say **[~]** ?

Now type: `cd ..` to move up one level. Your prompt should have **[home]** in it. All users' *home* directories are subdirectories of a dir called `home/`.

Type `cd ..` to go up from there. You should see **[/]** in the prompt.

Type `cd ..` again. Did anything happen? No–you should still see the same **[/]** in the prompt.

That's because you have gone up as far as you can go. The top level directory in a Unix file system is called, simply, `/`. Below that are a number of dirs, one of which is `home/`, and below that is your home dir, *e.g.* `/home/jhetrick/`.

Now type `cd` to go *home*.

3.5 Pathnames

Type the following command at the prompt.

`pwd` (print working directory)

Pathnames enable you to work out where you are in relation to the whole file-system. For example, to find out the full pathname of your `dirONE/` directory in the `unixplay/` directory.

navigate to `dirONE/` directory using `cd`, then type

`pwd`

```
sci[~]>cd unixplay
sci[~/unixplay]>cd dirONE
sci[~/unixplay/dirONE]>pwd
/home/jhetrick/unixplay/dirONE
```

The full pathname to the `dirONE` directory is

```
/home/jhetrick/unixplay/dirONE
```

3.6 Unix commands for files and directories

First let's review the `ls` command that we've already seen.

Let's review:

When you first login, your prompt is placed in your home directory. Your home directory has the same name as your user-name, for example, **jhetrick** (or " **~**" for short).

To find out what files and directories are in your current directory, you type

```
ls
```

The **ls** command *lists* the contents of your current working directory.

```
sci[~]>ls
Desktop    Downloads  Pictures  Templates  winDesktop
Documents  Music      Public    Videos     winDocs
```

If there were no files visible in your home directory, the prompt would simply be returned.

3.6.1 Command Options (aka Switches)

As we have seen, it turns out that `ls` does not, actually, list **all** the files in your home directory only *non-hidden* files. In unix it's easy to make a hidden file, simply start its name with a dot (.)

Files beginning with a dot (such as `.bashrc`, `.emacs`, etc.) are known as *hidden* files and usually contain program configuration information.

However you can list *ALL* the files in your home dir by giving *ls* a **switch** to modify its behavior.

To list all files in your home directory including those whose names begin with a dot, type

```
ls -a
```

This now produces

```
sci[~]>ls -a
.              Downloads        Pictures
..             .emacs           Public
.bash_history  .esd_auth        Templates
.bash_logout   .gnome2          .vboxclient-clipboard.pid
```

```
.bash_profile  .gnome2_private  .vboxclient-display.pid
.bashrc        .gnupg           .vboxclient-draganddrop.pid
.cache         .gtkrc-2.0-kde4  .vboxclient-seamless.pid
.config        .kde             Videos
Desktop        .local           winDesktop
.dmrc          .mozilla         winDocs
Documents      Music            .xsession-errors
```

As you can see, `ls -a` lists many files that are normally hidden.

`-a` is an example of an option (sometimes called a *switch*). The options change the behaviour of the command. There are online manual pages that tell you which options a particular command can take, and how each option modifies the behaviour of the command.

Note: We setup several *aliases* for `ls` with various switches in your `.bashrc` file in the last chapter. You have an *alias* for `ls -aF` in your `.bashrc` file. The alias is `la`, so typing `la` is exactly the same as typing `ls -aF`.

`ls -F`

Alias: `l`

Another option I like to add to your toolkit is the *-F* (Format) switch for *ls*. This option puts "/"s at the end of each directory, which makes it easier to distinguish files from subdirectories. It also puts a "*" at the end of each file which is *executable*. These files are programs or "*scripts* that can be run (like a *.exe* file in Windows), and an "@" at the end of *links*.

```
sci[~]>ls -F
Desktop/    Downloads/  Pictures/  Templates/  winDesktop@
Documents/  Music/      Public/    Videos/     winDocs@
```

As you can see this option makes it easier to see which files are just files and which are subdirectories. It's particularley useful when you are working in a terminal which doesn't support color coding.

Question

Which files are directories? What type are the other files?

3.6.2 `Man` (manual) pages

A wonderful feature of the unix shell are the *man pages*. At the prompt, type

```
man ls
```

This will bring up the *Manual Page(s)* for the `ls` command. The man page is a text file which is opened in the `less` pager that we met in the last chapter. `less` show you one terminal window full of the file at a time, and lets you move forward and backward through the manual with PageUp and PageDown (or SpaceBar and "b").

```
LS(1)                          User Commands                          LS(1)

NAME
     ls - list directory contents
```

```
SYNOPSIS
     ls [OPTION]... [FILE]...

DESCRIPTION
     List  information  about  the FILEs (the current directory by default).  Sort
     entries alphabetically if none of -cftuvSUX nor --sort is specified.

     Mandatory arguments to long options are mandatory for short options too.

     -a, --all
            do not ignore entries starting with .

     -A, --almost-all
            do not list implied . and ..

     --author
            with -l, print the author of each file

     -b, --escape
            print C-style escapes for nongraphic characters
 [More]
```

The **man** page is *GREAT* for learning about all the possible switches and is useful when you want to do something and think `ls` *might* have a switch that does what you want.

For example, scrolling through the man page for `ls`, you can see that one of the switches is `-1` (minus one). This switch lists one file per line. I have used that switch many times when I need to produce a single column list of files to pass to another program.

Apropos (man -k)

If you don't know the name of a command, but want to search all possible shell commands, you can type:

`apropos` SEARCH-WORD

For example, suppose you didn't know that there was a unix command to list the contents of your directory (but you did know about the `apropos` command). You could type:

```
apropos list
```

This produces:

```
sci[~]>apropos list
uffixes (7)          - list of file suffixes
acl (5)              - Access Control Lists
appres (1)           - list X application resource database
argz (3)             - functions to handle an argz list
argz_add (3)         - functions to handle an argz list
argz_add_sep (3)     - functions to handle an argz list
...
ls (1)               - list directory contents
...
XwcTextPropertyToTextList (3) - convert text lists and text property structures
zipinfo (1)          - list detailed information about a ZIP archive
zonetab2pot.py (1)   - Converts a timezone list to a PO file template.
```

showing all the commands and Manual entries with the word "list" in the description. From this list you would learn that there is the `ls` command, and could learn more about it by typing `man ls`.

Try doing: `man man`

3.6.3 Making Directories

`mkdir` (make directory)

We will often need to make a subdirectory in your current directory to hold new files, or particularly, when you start a new project.

> **TIP:**
>
> Keep different projects in separate directories and subdirectories, and give files names that help identify them, like: `comptonExptTheta35deg.dat`

To make a subdirectory called **newdir** in your home directory type the following:

```
mkdir newdir
```

```
sci[~]>mkdir newdir
sci[~]>
```

The shell returns the prompt. However it *has* just created the **newdir** directory.

To see the directory you have just created, type `l`

```
sci[~]>mkdir newdir
sci[~]>l
Desktop/    Downloads/  newdir/    Public/     unixplay/        Videos/      winDocs@
Documents/  Music/      Pictures/  Templates/  unixplay.tar.gz  winDesktop@
```

3.7 Understanding pathnames

First type `cd` to get back to your home-directory, then type

```
l newdir
```

to list the conents of your newdir directory.

Now type

```
l subdir
```

You will get a message like this -

subdir: No such file or directory

The reason is, subdir is not in your current working directory. To use a command on a file (or directory) not in the current working directory (the directory you are currently in), you must either `cd` to the correct directory, or specify its "**full pathname**". To list the contents of your subdir directory, you must type

```
l newdir/subdir
```

Home directories can also be referred to by the tilde ~ character. It can be used to specify paths starting at your home directory. So typing

```
ls ~/newdir
```

will list the contents of your newdir directory, no matter where you currently are in the file system.

Try this:

`cd` down into `subdir/`

Try to list the contents of `newdir/` by doing `ls newdir`.

This fails since newdir/ is not in subdir.

Now use the full pathname with the ~ shortcut:

```
ls ~/newdir
```

This time it works.

What do you think

```
ls ~
```

would list?

What do you think

```
ls ~/..
```

would list?

A handy command to know about is the `tree` command. Try it in your home directory. Type:

```
tree
```

at the prompt. It should show you a "*tree*" (with branches) of your directories and files.

If you just want to see *directories*, type:

```
tree -d
```

with the `-d` switch.

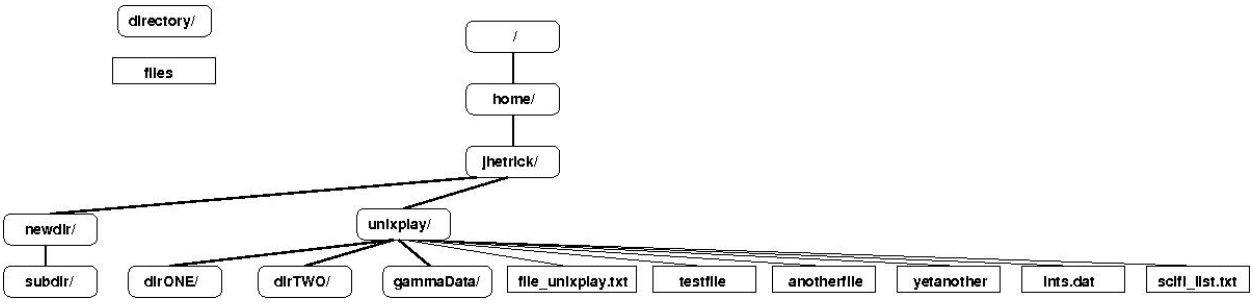
If you want to see *sizes* of files and directories, use the `-h` switch (**Try it**).

3.8 Command Summary

Command	Meaning
ls	list files and directories
ls -a	list all files and directories
mkdir dir	make a directory called dir
cd dir	directory change to named directory
cd	change to home directory
cd ~	change to home directory
cd ..	change to parent directory
pwd	display the path of the current directory
man cmd	page through manual entries for cmd
tar zxvf file.tar.gz	unpack a .tar.gz file
tree	show directory tree structure

aliases	equivalent to	meaning
l	ls -F	list (formatted)
la	ls -aF	list all (incl. hidden)
ll	ls -lF	long list
lt	ls -t | head -20	list most recent
m	less	file pager
fin file	find . -name file	find a file

3.9 Copy, Move, Delete, and Examine Unix Files



3.10 Video for this section

Video for this section is not available yet.

3.11 Copying Files and Directories

3.11.1 The cp command

```
cp file1 file2
```

is the command which makes a copy of `file1` and called `file2` . In this case, both files would be in the current working directory.

That's pretty simply. You might do

```
cp mycode.py mycode.backup23sept
```

to make a backup copy of your paper before you editted it

Other forms are:

```
cp file1 dir
```

This form places a copy of `file1` in *directory* `dir`.

Notice that it has the same form as copying a file and giving it a new name, but the second **argument** to `cp` is a *directory* destination. This usage copies the *file* `file1` to the *location* `dir`, but uses the same name as the original.

Also, you can do:

```
cp file1 file2 file3 dir
```

which places copies of the three files `file1`, `file2`, and `file3` in directory `dir`. Notice that if the last argument to cp is an existing directory , all preceding files in the argument list will be placed there. Both files and directories can have full pathnames if we want to copy files to different places in the file system.

If I were copying the three files above (named `file1`, `file2`, and `file3`), I would use a **Wildcard** match with `*`, and do it like this

```
cp file* dir
```

This form matchs all files whose names begin with `file`.

Time for an example.

In your `unixplay/` dir, in one of the subdirectories created when you unpacked the tar file, you should find a file named `testfile`.

Verify this by listing the directory `unixplay/`.

Now `cd` to your home directory, and type:

```
fin testfile
```

I want to show you how your *find alias* works. Try it again by this time do:

```
fin test\*
```

(You NEED the `\` in front of the `*`). That's how you give a *wildcard* match to the `fin` alias.

Ok, let's play. `cd` into `unixplay/`.

Make a copy of **testfile** called **testfile2**, by typing

```
cp testfile testfile2
```

Verify by listing that you now have 2 files: **testfile** and **testfile2**.

Now let's take a file stored in `unixplay/` and use the `cp` command to put a copy in your `newdir/` directory.

First, `cd` into your `newdir/` directory. It's below your home directory. Wherever you are, you can get there by typing the full pathname (using the ~ alias for your home dir):

```
cd ~/newdir
```

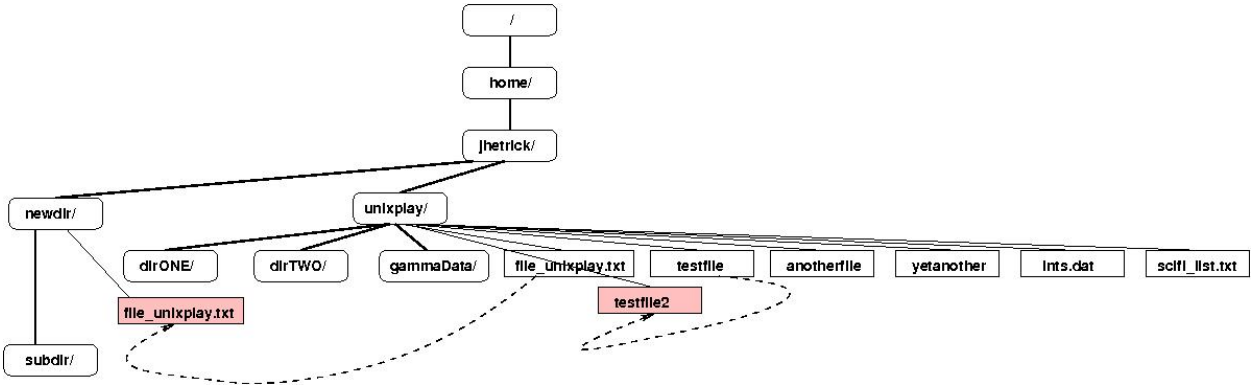
Then at the prompt, type,

```
cp ~/unixplay/file_unixplay.txt .
```

Note : Don't forget the dot `.` at the end! Remember, in Unix, the dot means the current directory. Also notice that we used full pathname of the original (source) file including the shortcut " ~ " for your home directory.

The above command means copy the file `file_unixplay.txt` , which is in `unixplay/` (itself a sub-directory of your *home* directory) to the current directory ("`.`"), keeping the name the same.

The effect of these commands is as follows:



Note: You can *clone* a whole directory tree and its contents (i.e. including subdirectories) by using the "*Recursive*" copy switch, `-r`, with `cp`.

Doing `cp -r dir1 dir2` will place a copy of `dir1/` *and ALL its contents and subdir-contents* in `dir2/`.

3.12 Moving and Renaming Files and Directories

3.12.1 The `mv` command

The `mv` command can be used to "*move*" files *and whole directories* as well as to

```
mv file1 file2
```

moves, or really **renames**, file1 to file2

To move a file from one place to another, you also use the `mv` command. This has the effect of moving rather than copying the file, so you end up with only one file rather than two.

In this usage you would do:

```
mv file1 dir1
```

This would place `file1` in `dir1`, and erase it from its current location.

Let's practice.

Use `mv` to rename a file

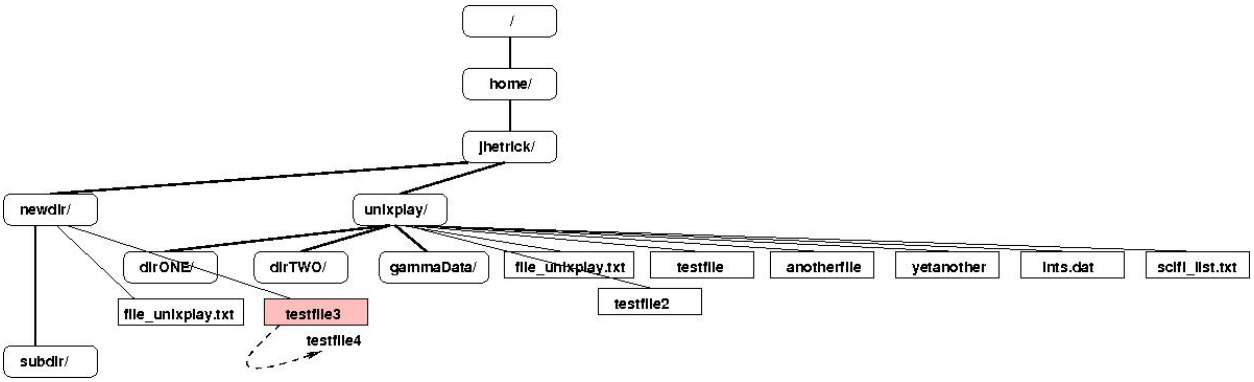
We are now going to rename the file `testfile3` to `testfile4`.

First, change directories to your newdir/ directory (remember how?) . Then type `l` to see what files are there. Do you see `testfile3` that you made by copying `testfile2` above?

Type

```
mv testfile3 testfile4
```

Now do a listing to see if `testfile3` is gone and replaced by `testfile4` (i.e. that `testfile3` was renamed to `testfile4`).



Use `mv` to move a file to somewhere else

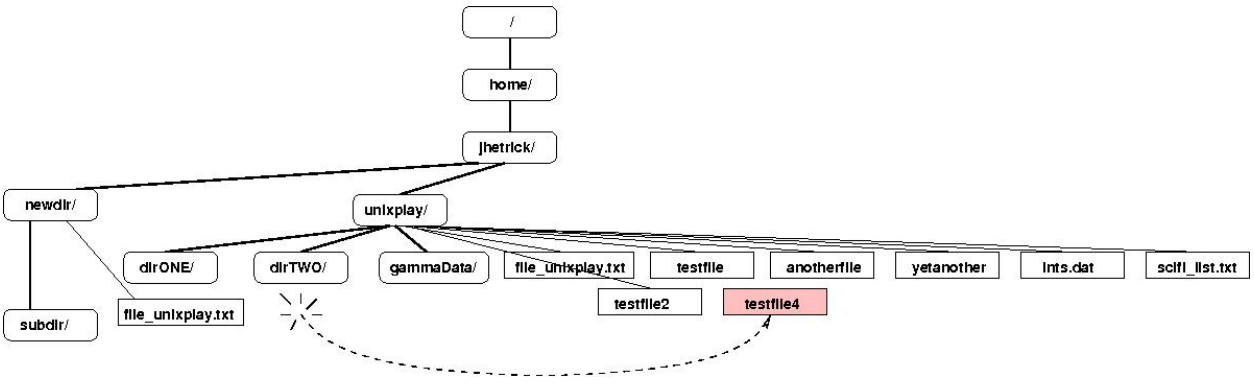
We will now move the file `testfile4` back into your `unixplay/` directory.

Do a listing to verify that `testfile4` is still there in your `newdir/` directory.

Now do

```
mv testfile4 ../unixplay
```

You can see in this command that the destination directory is up one level (`..`) and then down into the `unixplay/` directory. Verify that the move worked–that `testfile4` is not in the current directory (`newdir/`) and is now in `unixplay/` .



Use mv on a whole directory

Amazingly, you can `mv` a directory–either in the sense of renaming or moving it.

`cd` to your home directory .

Do a listing to see that the directories `unixplay/` and `newdir/` are there.

Now do

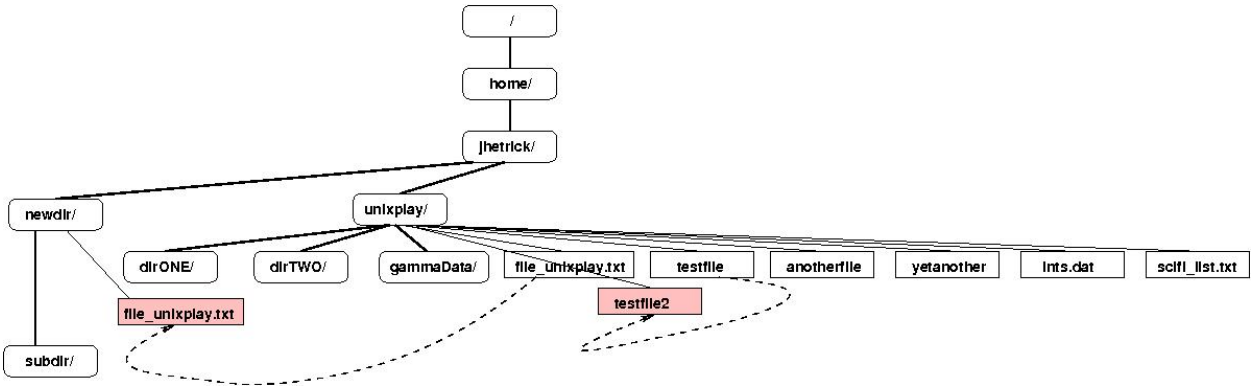
```
mv newdir dirTHREE
```

Do you see that this is using **mv** to rename the directory? Verify that it worked. Was `newdir/` been renamed to `dirTHREE/` ?

Still in you home dir, do

```
mv dirTHREE unixplay
```

This will move the newly renamed `dirTHREE/` into your `unixplay/` directory.



Verify that `dirTHREE/'' is now a subdirectory of ``unixplay/`. There should also be a `dirONE/` and `dirTWO/` there as well.

3.13 Removing Files and Directories

Warning: The `rm` and `rmdir` commands erases your files and dirs. Unix doesn't have a *Recycle Bin* where you can find the files you accidentally deleted. This Unix OS considers you to be God. If you say "*delete this file*, Unix complies with no wimpering; No "Do you really want to do that" warnings, or other protestations. So, be aware when you delete (`rm`) files and directories.

3.13.1 rm and rmdir

To delete (or remove) a file, use the `rm` command. As an example, we are going to create a copy of the `pope.txt` file then delete it.

Do a listing inside your `unixplay/` directory

Is the file `file_unixplay.txt` there? We don't need this one anymore.

Do

```
rm file_unixplay.txt
```

List the directory's contents. The `file_unixplay.txt` should have been listed before you did the `rm` command, but then be gone in the second listing, since you `rm`'ed it.

You can use the `rmdir` command to remove a directory (make sure it is empty first). Try to remove the `unixplay/` directory. You will not be able to do so, since Unix will not let you remove a non-empty directory with `rmdir` (without taking extra steps).

You have two options for removing a directory that has files in it (*assuming you don't need the files!!*).

1. You can `cd` into the directory, *delete all* the files by typing `rm *`, then `cd` back up to the parent directory (`cd ..`), and typing `rmdir dir`.
2. You can use the (*very long*) switch, `--ignore-fail-on-non-empty`, as in `rmdir --ignore-fail-on-non-empty dir`.
3. You can actually use: `rm`, the same command you use for files. Just give it the `-rf` switch. These are **force** (`-f`) and **recursive** (`-r`). This will travel down the directory and its subdirs, removing everything in its path.

Warning: This last way: `rm -rf dir` is convenient, but note that it is *very powerful*! If you cd to `/` (the top of the unix file system), and type `rm -rf` you will erase your *entire* disk, with no way to recover (unless you are making backups). That would probably be bad. Similarly, people have been known to do `rm -rf *` thinking they were down in a subdirectory and wanted to prune everything below. Then they realized that they were actually in their *home* directory. Usually they swear a lot when they realized what's happened. So–BEWARE. With great power comes great responsibility!

Create a directory called `tempstuff` using `mkdir`, `cd` into `tempstuff/` and do `touch zap`. The `touch` command will create a file of zero size named `zap` in the `tempstuff/` directory.

Now try to delete the `tempstuff/'' directoty using either the ``rmdir` or `rm -rf` commands. You might try the experiment with both commands.

3.14 Manipulating the contents of files

3.14.1 `cat` (Concatenate)

The command `cat` (for *concatenate* or to join) can be used to display the contents of a file on the screen. Type:

```
cat testfile
```

This is fine since the file contains a small amount of text which all fits on the screen.

If the file is bigger than will fit in an xterm window, we will use the **less** (pager) command, which you have already met before.

You can also do `cat file1 file2 file3`. This will concatenate (add together) all three files into one long file and spew it to the screen. Unless the file is small enough to fit whole on the screen, cat is most often used to pass the contents of one file to another program.

Try it with the three files in the **unixplay/** dir. Do

```
cat testfile anotherFile yetanother
```

This should produce

```
This is a text file, called testfile.
This is another file.
This is yet another file.
```

The `cat` command printed the contents of each file, one after the other.

3.14.2 `less` (Page through a file)

The command `less` displays the contents of a file onto the screen one page at a time, which is much more suitable for actually reading a file. In your `unixplay/` dir, you will find a a subdir called *dirONE‘* which contains the file “`KublaKhan.txt`”.

`cd` to `dirONE/`, then do

```
less KublaKhan.txt
```

Press the **SPACE-BAR** (or PgDn) if you want to see the next page, and “**b**” (or PgUp) to go back. Type “**q**” when you want to quit reading. As you can see, `less` is much better than `cat` for reading long files.

`less` is a modern replacement to the original Unix paging file viewer called “`more`”. The `more` program printed a page at a time, but offered little other functionality. It’s a bit of Unix geek humor that the replacement for the `more` program would be named "``less”.

`less` offers many features for searching and moving. More on this below. You can see more of the features and how to use them by typing “**h**” while viewing a file with `less` (type “**q**” to “Quit” `Help`)..

3.14.3 `head` (the first lines of a file)

The `head` command writes the first *ten* lines of a file to the screen.

First clear your screen (by typing `clear`, of course!), then `cd` to your `unixdir/` directory. First find the file `ints.dat` and view it with `less`. It contains the first 100 integers. Now type

`head ints.dat`

Then type

`head -5 ints.dat`

What difference did the -5 option make to the head command? This generalizes to `-n`, where **n** is the number of lines you want to print.

3.14.4 `tail` (the last lines of a file)

The `tail` command writes the *last* ten lines of a file to the screen.

`clear` the screen and type

`tail ints.dat`

Question: How can you view the last 15 lines of the file?

3.15 Searching the contents of a file

3.15.1 Simple searching using less

Using `less`, you can search though a text file for a keyword (or text pattern). For example, to search through `KublaKhan.txt` for the word **pleasure**, type

`less KublaKhan.txt`

(You should be able to find this file by now, even if you forgot where it is).

Now, *while still viewing the file in* `less`, type a *forward slash* (*/*) followed by the word you wish to search for, like this:

`/pleasure`

As you can see, less finds and highlights the keyword. Type the " **n**" key to search for the next occurrence of the word.

3.15.2 `grep` (search multiple files for a word)

This is a *REALLY* useful command. (don't ask why it is called *grep*. Ok, if you must know, the answer's here)

`grep` is one of many standard UNIX utilities. It searches files for specified words or patterns.

First `clear` the screen, `cd` into `unixplay/`, do a listing and find the file `scifi_list.txt`. This is a (somewhat outdated) list of the top Science Fiction books of all time, in random order. Have a look with `less`. Then type

`grep time scifi_list.txt`

As you can see, grep has printed out each line containg the word **time** (in color too!)

Or has it ????

Try typing

```
grep Time scifi_list.txt
```

The `grep` command, by default, is case sensitive; it distinguishes between **time** and **Time**.

To ignore upper/lower case distinctions, use the `-i` switch, i.e. type

```
grep -i time scifi_list.txt
```

Try it.

To search for a phrase or pattern, you must enclose it in **single** quotes (the apostrophe symbol). For example to search for the phrase **Time Machine** , type

```
grep 'Time Machine' scifi_list.txt
```

Some of the other options of grep are:

`-v` display only those lines that do NOT match the expression

`-n` precede each matching line with the line number

`-c` print only the total count of matched lines

Try some of them and see the different results. Don't forget, you can use more than one option at a time. For example, the *number* of lines *without* the words **time** or **Time** is

```
grep -ivc time scifi_list.txt
```

3.15.3 `wc` (word count)

A handy little utility is the `wc` command, short for *Word Count*. To do a word count on `scifi_list.txt` , type

```
wc -w scifi_list.txt
```

To find out how many lines the file has, type

```
wc -l scifi_list.txt
```

With no options

```
wc scifi_list.txt
```

wc prints:

```
100 746 4173 scifi_list.txt
```

which shows the number of: *lines words bytes filename*.

3.16 Sort the contents of a file

The file `scifi_list.txt` is nice; it contains the top 100 Science Fiction books. However, it would be nicer if they weren't in *random* order!

No problem–use `sort`!

First have a look at the file using `less`. The columns are

```
1995    53      Stephenson, Neal        The Diamond Age
1956    36      Bester, Alfred  The Stars My Destination
2000    89      Reynolds, Alastair      Revelation Space
...
```

or *Year Rank Author Title*.

If you simply do

```
sort scifi_list.txt
```

you will order the file **alphabetically on the first column**:

```
1818    52      Shelley, Mary   Frankenstein
1864    57      Verne, Jules    Journey to the Center of the Earth
1870    32      Verne, Jules    20,000 Leagues Under the Sea
1884    92      Abbott, Edwin A Flatland
1889    86      Twain, Mark     A Connecticut Yankee in KA's Court
1895    16      Wells, H G      The Time Machine
1897    77      Wells, H G      The Invisible Man
1898    18      Wells, H G      The War of the Worlds
...
```

Notice that "*alphabetical*" on the year, ends up ordering the years.

This is because alphabetical order is

0123...89ABCD...XYZabcd...xyz.

Nice! Now we see the top books, ordered by year.

How about sorting by *rank* (in **column two**)?

Do:

```
sort -k 2 scifi_list.txt
```

which means "*sort alphabetically on column 2*". You should see:

```
1965    1       Herbert, Frank  Dune
1968    10      Clarke, Arthur C        2001: A Space Odyssey
1974    100     Lem, Stanislaw   The Cyberiad
1970    11      Niven, Larry    Ringworld
1959    12      Heinlein, Robert A      Starship Troopers
1968    13      Dick, Philip K  Do Androids Dream of Electric Sheep?
1932    14      Huxley, Aldous  Brave New World
1973    15      Clarke, Arthur C        Rendezvous With Rama
1895    16      Wells, H G      The Time Machine
1966    17      Heinlein, Robert A      The Moon is a Harsh Mistress
1898    18      Wells, H G      The War of the Worlds
1989    19      Simmons, Dan    Hyperion
1985    2       Card, Orson Scott       Ender's Game
1954    20      Clarke, Arthur C        Childhood's End
```

```
1950    21      Bradbury, Ray    The Martian Chronicles
...
```

You can see that the books are ordered by the second column now, but the order is *wrong*!. It's alphabetical, but what we really wanted was "**numerical**" ordering. This time do:

```
sort -k 2 -n scifi_list.txt
```

This time we get it the way we wanted:

```
1965    1       Herbert, Frank  Dune
1985    2       Card, Orson Scott       Ender's Game
1951    3       Asimov, Isaac   Foundation
1979    4       Adams, Douglas  Hitch Hiker's Guide to the Galaxy
1949    5       Orwell, George  1984
1961    6       Heinlein, Robert A      Stranger in a Strange Land
1954    7       Bradbury, Ray   Fahrenheit 451
1984    8       Gibson, William Neuromancer
1950    9       Asimov, Isaac    I, Robot
1968    10      Clarke, Arthur C        2001: A Space Odyssey
1970    11      Niven, Larry    Ringworld
1959    12      Heinlein, Robert A      Starship Troopers
1968    13      Dick, Philip K  Do Androids Dream of Electric Sheep?
1932    14      Huxley, Aldous  Brave New World
1973    15      Clarke, Arthur C        Rendezvous With Rama
...
```

We could also alphabetize the file on the author's names:

```
sort -k 3 scifi_list.txt
```

which gives us:

```
1884    92      Abbott, Edwin A Flatland
1979    4       Adams, Douglas  Hitch Hiker's Guide to the Galaxy
1950    9       Asimov, Isaac    I, Robot
1951    3       Asimov, Isaac   Foundation
1954    29      Asimov, Isaac   The Caves of Steel
1955    49      Asimov, Isaac   The End Of Eternity
1972    43      Asimov, Isaac   The Gods Themselves
1985    68      Atwood, Margaret        The Handmaid's Tale
1988    56      Banks, Iain M   Player Of Games
1990    65      Banks, Iain M   Use of Weapons
1985    97      Bear, Greg      Blood Music
1985    60      Bear, Greg      Eon
1953    64      Bester, Alfred  The Demolished Man
1956    36      Bester, Alfred  The Stars My Destination
...
```

Try:

```
sort -k 3 -r scifi_list.txt
```

What did the `-r` option do? `sort` is *VERY* handy!

3.17 Command Summary

Command	Meaning
`cp` *file1 file2*	make a copy of file1 called file2
cp file1 dir	put a copy of file1 in dir
cp f1 f2 f3 dir	copy files f1 f2 and f3 to dir
cp fi* dir	copy all files which match "fi"... to dir
fin file	find file, by searching . and all subdirs
fin fi*	find files matching "fi" (wildcard: use *, not *)
mv file1 file2	move or rename file1 to file2
mv file1 dir	move file1 into dir
rm file	remove a file or files
rmdir dir	remove an empty directory
rm -rf dir	forcibly delete directory, files, and subdirs
cat file	print a file
cat f1 f2 f3	print a files f1, f2, and f3 all together
less file	display a file a page at a time
head file	print the first few lines of a file
head -30 file	print first 30 lines of a file (works for any n)
tail -n file	print the last n lines of a file
grep 'pattern' file*	check file(s) for 'pattern'
grep -i 'pattern' file*	check file(s)... ignore CASE of match
grep -v 'pattern' file*	check file(s)... report if DOES NOT contain pattern
grep -n 'pattern' file*	check file(s)... report LINE NUMBER of match
grep -c 'pattern' file*	check files(s)... report NUMBER OF MATCHING LINES
wc file	count number of LINES / WORDS / CHARACTERS in file
sort file	sort lines of a file, ALPHABETICALLY on 1st column
sort -k N file	sort lines of a file, on N-th COLUMN
sort -n -k N file	sort lines of a file, NUMERICALLY, on N-th column
sort -r file	sort lines of a file, report in REVERSE order

3.18 Controlling Data Flow

3.18.1 Video for this section

Video for this section is not available yet.

3.18.2 Stdout and Stdin

Most processes initiated by Unix commands write their output to the **standard output** channel (offically called `stdout`), that is, they write to the terminal screen. Similarly most programs take their input from the **standard input**, (`stdin`) i.e., they read input from the keyboard. There is also the standard error , where processes write their error messages, which is by default, also the terminal screen.

Here is one rather abstract example.

We have already seen how to use the ‘‘cat‘ command to write the contents of a file to the screen.

This time however, type `cat` without specifing a file to read. Just type:

```
cat
```

Without specifying a file, the `cat` program has no content to print to the screen.

After typing `cat` and ENTER, the cursor goes to the beginning of the next line and waits.

```
sci[~]>cat
```

Type a few words on the keyboard and press the **ENTER** key. You should see your words as you type, then each time you hit ENTER, the word(s) you typed should be printed again.

```
sci[~]>cat
Here is some text[ENTER]
Here is some text
and some more[ENTER]
and some more
you get the idea[ENTER]
you get the idea
```

Finally, type `Ctrl-d` (the `CTRL` key *and* the "**d**" key, simultaneously).

What happened?

If you run the `cat` command without specifing a file to read, then instead of reading from a file, it reads from **stdin** (*standard input*, i.e. the **keyboard**), then writes to **stdout** (*standard output*, the **screen**), until receiving the **End of File** signal (`CTRL-D`).

In Unix, we can **redirect both the input and the output of commands**, which can be VERY powerful.

3.18.3 Redirecting the Output: >

We use the **>** symbol to redirect the output of a command to a file. In your `unixplay/` directory is a little program called `squares.py` which simply prints out integers and their squares.

`cd` to your `unixplay/` directory, and do a *long list* (use your alias''ll'', or do `ls -lF`). Notice that `squares.py` has is shown with a trailing `\*`

```
sci[unixplay]>ll
total 44
-rw-rw-r--. 1 jhetrick jhetrick   22 Sep 24 15:58 anotherFile
drwxrwxr-x. 2 jhetrick jhetrick 4096 Sep 24 15:58 dirONE/
drwxrwxr-x. 3 jhetrick jhetrick 4096 Sep 24 15:58 dirTWO/
-rw-rw-r--. 1 jhetrick jhetrick   53 Sep 24 15:58 file_unixplay.txt
drwxrwxr-x. 2 jhetrick jhetrick 4096 Sep 24 15:58 gammaData/
-rw-rw-r--. 1 jhetrick jhetrick  292 Sep 24 15:58 ints.dat
-rw-rw-r--. 1 jhetrick jhetrick 4273 Sep 24 15:58 scifi_list.txt

-rwxrw-r--. 1 jhetrick jhetrick  468 Nov 21 10:07 squares.py*

-rw-rw-r--. 1 jhetrick jhetrick   38 Sep 24 15:58 testfile
-rw-rw-r--. 1 jhetrick jhetrick   26 Sep 24 15:58 yetanother
```

We've alread met the trailing `/` decoration, indicating a *directory*, as well as the trailing `\@` showing *links* (shortcuts). Now you are seeing the `\*` which indicates that the file is **executable**–it's a program or *script* that can be run, and it will do something.

Go ahead and type

```
squares.py
```

If called with no *arguments* (i.e. just by itself), it prints the first 10 integers and their squares. (**Did it?**)

If called with a **min** and **max**, like this:

```
squares.py 20 30
```

it prints the integers and squares between the **min** and **max** (20 and 30 in this case) numbers, as shown above. These numbers are called the **arguments** to the command. The output is shown below.

```
20      400
21      441
22      484
23      529
24      576
25      625
26      676
27      729
28      784
29      841
30      900
```

Now let's **redirect the output to a file**. Type this:

```
squares.py > squares.out
```

Verify that the output file is there by listing (`l`) and then have a look at the file `squares.out` with `less`. Remember, you have an alias for `less` in your `.bashrc` file. You can use the alias `m` for `less`, so typing

```
m squares.out
```

should show you the first 10 integers and their squares, in your newly created data file, `squares.out`.

When doing scientific computing we will often use this method to catch the output of a program in a file. We can then use other tools on the file, such as `sort` or plot the data with `gnuplot`.

Try it again, but give `squares.py` some arguments. Catch the output in the same file as above, again:

```
squares.py 20 30 > squares.out
```

Check the output file again with `less` (or `m`).

Notice that the *previous* `squares.out` file, with integers 1 through 10, has been overwritten by the second use of the redirect >.

Note: It is important to remember that when using > by itself, the redirection output file is first cleared before the output is collected. Your previous file "squares.out" has been lost and replaced with the new output.

3.18.4 Append Output to a File: >>

If instead of using > for redirection, we can use the *double* redirect symbol: >> to **append** data to an existing file, as opposed to overwritting it. This is very handy if we do a computation over and over and want to add the result to the end of the output file after each iteration.

Look at your `squares.out` file with `less`. It should contain the squares of the integers from 20 to 30. Now, do

```
squares.py 31 40 >> squares.out
```

Verify that the file `squares.out` now contains the integers and squares from 20 to 40. The last command **appended** the results from 31 to 40 to the existing file `squares.out` which already had 20 through 30.

Note: Using >> does NOT erase, then rewrite the output file. It preserves the output file, and adds to it.

3.18.5 Redirecting the Input: <

We can use the < symbol to redirect the input to a command *from* somewhere else, usually a file.

In the `unixplay/` directory is another executable file called: `cubes.py`

Run this program by typing `cubes.py`.

This time, the program will *prompt you for input*. Instead of taking *command line arguments* like `squares.py` above, it *asks* you to input the numbers:

```
sci[unixplay]>cubes.py
Enter MIN:
```

Enter a small integer, like 3, and then ENTER.

```
sci[unixplay]>cubes.py
Enter MIN: 3
Enter MAX:
```

Do the same for the maximum integer (enter a number greater than the MIN integer).

```
sci[tmp]>cubes.py
Enter MIN: 3
Enter MAX: 8

3       27
4       64
5       125
6       216
7       343
8       512
```

The program outputs the cubes of the integers from MIN to MAX.

This program is a little different than `squares.py`. `squares.py` used "*command line arguments*": items given on the command line after the name of the program that are passed to the program to modify its behavior.

`cubes.py` prompts you for input, asking you questions, to which you input data. So it is expecting YOU to give it data, via *stdin*—the keyboard

We can put this input data intp a file, then *redirect the input* to `cubes.py` from that file.

First, let's put the input numbers in a file. Since we haven't learned how to edit a text file yet, we can use `cat` as we did above. Without specifying an output file, `cat`, by itself, will read from *stdin* (the keybord) and write to *stdout* (the screen). We can use output redirection, >, to redirect that output to a file!

When we run `cubes.py`, it wants us to input two numbers, the MIN and MAX integers.

So, we want to have a file that contains the two numbers that we would input to `cubes.py` if we were typing those inputs by hand from the keyboad. In our example above I used **3** and **8**.

We type `cat > in.cubes` with no filename specified for *cat* to read, *then* redirect the output of *cat* to a file called `in.cubes`.

``sci[unixplay]> cat > in.cubes`` [ENTER]

The cursor will go the beginning of the next line, like it did above, as `cat` waits for you to type something.

So, type

```
3
8
[CTRL-D]
```

Now, do an listing to see that your new file, `in.cubes` is there.

If you want to see what the last file produced was (it should be `in.cubes`), use the `lt` alias I put in your .bashrc file. Do

```
lt
```

and you should see `in.cubes` at the top of the list.

Have a look at `in.cubes` with `less`. It should contain two lines, with: **3** and **8**.

Now use this file *as the input* to `cubes.py` by redirecting input using <

```
sci[unixplay]>cubes.pl < in.cubes
Enter MIN: Enter MAX:
3       27
4       64
5       125
6       216
7       343
8       512
```

This should produce the integers from 0 to 20 and their cubes. **Does it?**

Notice that when `cubes.py` runs, it prints the prompts "*Enter MIN:*" and "*Enter MAX:*", but now reads from the file `in.cubes` instead of *stdin* (the keyboard).

We used the < to redirect the input to `cubes.py` from a file instead of from *stdin*.

3.18.6 < and >

You can use both input and output redirection at the same time! (*I can hear the sound of your mind being blown*).

```
command < in.file > out.file
```

will make **command** read input from the file **in.file** and catch the output in the file **out.file**.

Try this yourself. Use input and output redirection as described above to save the cubes of integers from 0 to 2 to a file called `cubes_0_20.out`.

3.19 The Pipe: |

Previously we learned a few really useful commands: **grep**, **sort**, and **wc**. Be assured, there are many more.

It turns out that we can string these together so that the output of one command serves as the input to the next, using the " **pipe**". This is kind of like redirecting the output and input to a file, but instead of using files on the hard drive, we can pass all the data between programs in memory (which is much faster and more convenient).

The "*pipe*" is the vertical bar "|" on your keyboad, above the "\" backslash.

Recall our file, `scifi_list.txt`, which contains the year of publication, rank, author, and title of 100 of the best science fiction works. Let's ask some questions:

Suppose you want to know the rank in popularity (column 2 in `scifi_list.txt`) of the books by Robert Heinlein . Have a look at the file to remind yourself what's there. Heinlein's books are mixed in with the rest–randomly. **grep** comes to the rescue: `grep Heinlein scifi_list.txt` produces

```
1961    6       Heinlein, Robert A      Stranger in a Strange Land
1973    41      Heinlein, Robert A      Time Enough For Love
1958    93      Heinlein, Robert A      Have Space-Suit - Will Travel
1957    84      Heinlein, Robert A      Citizen Of the Galaxy
1951    88      Heinlein, Robert A      The Puppet Masters
1956    80      Heinlein, Robert A      The Door Into Summer
1966    17      Heinlein, Robert A      The Moon is a Harsh Mistress
1959    12      Heinlein, Robert A      Starship Troopers
```

Then, to get these ordered by popularity rank, you could redirect the output of the `grep` command to a file, then sort the file in a two step process, like this:

```
grep Heinlein scifi_list.txt > heinlein.out
sort -k 2 -n heinlein.out
```

However, the pipe allows you to do this in one step:

```
grep Heinlein scifi_list.txt | sort -k 2 -n
```

which yields

```
1961    6       Heinlein, Robert A      Stranger in a Strange Land
1959    12      Heinlein, Robert A      Starship Troopers
1966    17      Heinlein, Robert A      The Moon is a Harsh Mistress
1973    41      Heinlein, Robert A      Time Enough For Love
1956    80      Heinlein, Robert A      The Door Into Summer
1957    84      Heinlein, Robert A      Citizen Of the Galaxy
1951    88      Heinlein, Robert A      The Puppet Masters
1958    93      Heinlein, Robert A      Have Space-Suit - Will Travel
```

We "*piped*" the output of the **grep** command to the **sort** command, (where we used the switches "**-k 2 -n**" in order to sort on the second "**k**"olumn in **n**umerical order).

We can even chain pipes together:

```
grep Heinlein scifi_list.txt | sort -k 2 -n | wc
```

produces simply

```
8      73     402
```

The output of the sorted grep contains 8 lines, 73 words, 402 bytes.

3.20 Command Summary

Data Flow	Meaning
command > file	redirect output to a file (catch/save output of command)
command >> file	catch and *append* output of command to the end of a file
command < file	read input for a command from a file
cmd1 | cmd2	pipe the output of cmd1 to the input of cmd2

3.21 Homework

Homework 3 is here

CHAPTER

FOUR

THE TEXT EDITOR

4.1 The Text Editor

The first thing we will need to build our scientific computing environment is a **text editor**.

In fact you already have one on your computer–all computers have a default editor that will save files as basic ASCII text. ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange), pronounced [ass' key], is a simple character encoding format based on the English alphabet. It basically lets you encode the keyboard characters plus some special characters like a command to jump to the beginning of a new line (Enter or Return on your keyboad).

The printed ASCII characters (those that get printed to your screen, as opposed to control keys like *CTRL-D* or *ESC*), are:

```
!"#$%&'()\*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

The binary equivalents are of the alphabetical characters are here:

Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000

```
  y         121       01111001      Y         089       01011001
  z         122       01111010      Z         090       01011010
```

There are a few others for a *TAB*, *New Line*, and "*Alarm*" (tell the computer to make a noise or flash). Here are a few of those:

```
ASCII Code    Binary     Character Description
    0        00000000      NUL  null
    1        00000001      SOH  start of header
    2        00000010      STX  start of text
    3        00000011      ETX  end of text
    4        00000100      EOT  end of transmission
    5        00000101      ENQ  enquiry
    6        00000110      ACK  acknowledge
    7        00000111      BEL  bell
    8        00001000       BS  backspace
    9        00001001       HT  horizontal tab
  ...
```

When interacting with your computer, you will need to create text files containing data, programs, scripts (short programs that are not compiled), and LaTeX files. We'll learn more about each of these later.

A **Text editor** is very different from a **WYSIWYG** editor (pronounced [WIZ' ee wig]; short for **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et) like *Microsoft Word*. WYSIWYG editors are for the creation of documents that will be printed (or faxed or emailed) where layout, font, and formatting is essential–as in business documents. WYSIWYG files are huge, even if they contain only a few lines of text, since they keep formatting info, change history, and many other things.

ASCII (also called "*plain text*") files will often be read by another program–**and this is their true beauty** This could be a list of numbers to be plotted, or the source code (the program text) to be compiled into an executable program file (like an .exe file). You might use your computer very professionally in a business office for many years without using a text editor, but for scientific computing and programming, we can't do much without a text editor.

4.1.1 The Default Text Editor

If you are working on a computer with the **Windows** operating system, you already have a default text editor: **Notepad** which you can find from *Start->Programs->Accessories->Notepad*. This program will create ASCII text files. By default, Notepad saves files with the extension `.txt`. You can change this by adding your own extension when saving your file (call it something like `filename.dat`).

If you are using **OSX** on a Mac, the default editor is **TextEdit**. You should be able to find it in Applications.

Try Notepad/TextEdit now.

- Open the *Notepad* (PC) or *TextEdit* (Mac) program, as described above
- Type some text into the editor
- Save the file (onto your desktop or in your home dir. Name it anything you like.)

4.1.2 The Unix Default Editor

Under Unix the default editor is **vi**, although most unix systems also have **emacs** and others (Linux often includes **nano**, **KWrite**, and others).

The first one mentioned, **vi**, is short for **Vi**sual editor and dates from 1976. This editor has two different modes, one for adding text and another for moving around, copying/pasting, search/replace, etc. Since it was developed at a time when using Unix meant interacting with a mainframe *only* via a terminal window and keyboard (*there were no*

computer mice in 1976!), both of these modes had to use characters on the keypad. In 1976, there wasn't even that much uniformity as to the binary sequence sent by an arrow key–if it even existed on your computer.

In the 1980s another text editor became popular: **Emacs**. Emacs is extremely powerful with many features, although some say it has a steep learning curve. The devotion of Emacs users to Emacs and Vi users to Vi has spawned a fundamental rivalry in the unix community. Vi and Emacs are the two major sides in the Editor Wars which raged for some 35 years between unix geeks, and has only subsided when memory and bandwidth became so very cheap.

I myself, use *The One True Editor*: **Emacs**. To get a sense of the features and comparisons of the most popular text editors, open the following link:

- Comparison of Text Editor Features

Impressive, eh? As you can see, there are many... This site lists the editors by popularity.

The top two TEXT editors are: **Vim** (a modern version of **vi**)and **Emacs**. Over the years, I've used many, and they are all good. Let's look at some features that we will need.

An essential feature for remote computing (working on a computer that is not in you lap or under your desk) is that they can be used in text mode over a network connection, such as a Terminal window that is connected to a computer on the other side of the world. If you want to edit a file on that remote machine, you will want to use an efficient editor which has hi functionality using only text based commands. Both **Vim** and **Emacs** are text based. The alternative to this is using a GUI (**G**raphical **U**ser **I**nterface) editor that has little icons for buttons and opens its own window(s).

While these editors are very nice and have lots of features, they use a lot of computing power to update the windows, respond to the mouse clicks, highlight text, etc. Transmitting these features over the network can be quite demanding. Consider that anytime a character is written to the window, the entire window is usually repainted on the screen. When the editor window is running on you own PC, you don't notice this. But when each update has to go across the country and back, you do.

Windows users may like **Notepad++** (**'http://notepad-plus-plus.org/'_**) is such an editor. It runs only in GUI mode, and only under Windows, but is quite nice. **TextWrangler** (**'http://www.barebones.com/products/textwrangler/'_**) is a similarly good GUI editor for Mac users.

Note that if you *REALLY must have* a GUI editor, you can set these editors to retrieve the remote file you wish to edit via the network, edit it locally on your laptop/desktop, then put the new version of the file back on the remote machine. But that takes a bit more work...

Other nice features we expect in a powerful editor are syntax-color-coded text, modes for different types of files (C++, LaTeX, Python, HTML, etc.), and language-specific keyboard shortcuts for common tasks (like commenting regions, or inserting URLs).

4.1.3 Vi

- Open a Terminal window
- At the prompt, type: **vi**

You will see this ("splash") screen:

```
~
~
~
~                                   VIM - Vi IMproved
~
~                                    version 7.2.411
~                               by Bram Moolenaar et al.
~                            Modified by <bugzilla@redhat.com>
~                          Vim is open source and freely distributable
~
```

```
~                               Help poor children in Uganda!
~                         type  :help iccf<Enter>       for information
~
~                         type  :q<Enter>               to exit
~                         type  :help<Enter>  or  <F1>  for on-line help
~                         type  :help version7<Enter>   for version info
~
~
```

Recall that there are **two modes** that *vi* can be in:

- *Command* mode, and
- *Insert* mode

Vi opens in *Command* mode. So, if you type some key strokes they probably won't do anything–**until**–you type one (of several keys) that puts you into *INSERT* mode. Then the keystroke following that will put text in your document.

Try this:

In the open *vi* program, *type* **hello**

Nothing will happen until you type "*o*", at which point the cursor drops one line (and the splash screen disappears), and the word "**– INSERT –**" appears at the bottom of the screen.

In **COMMAND** mode, without any text in the document, the first four characters: *h-e-l-l* don't do anything. (if there *was* text in the window, **h** moves *left*, **l** moves *right*, and **e** would move to the *end of the current word*).

However, the "*o*" character in *h-e-l-l-o* tells *vi* to open a new line for text *below* the current line, and go into **INSERT** mode (That's why it drops down one line). In **INSERT** mode, what you type is recorded.

Type your name and address.

Now type, **ESC** to exit **INSERT** mode and go back to **COMMAND** mode.

In **COMMAND** mode, with the UP arrow, move the cursor up to the first line.

Type **i** to go into **INSERT** mode. Now what ever you type will be inserted into your document.

Type your name on the first line, and your home address on the next 2 (or 3) lines. While you are in **INSERT** mode, vi acts like a normal text editor, like Notepad. You type and text appears, you can use the arrow keys, backspace, and delete to move around.

To *Save* (or "Write") your newly written text file, you must "*Exit* **INSERT** mode" by typing **ESC**, then hit the colon key "**:**". A colon will appear at the bottom left of the window. *Vi* is now waiting for you to enter some command keystrokes.

Type **w myaddress.txt** (note the SPACE between the "w" and the filename "myaddress.txt")

This means "**w**rite the contents of the screen buffer to a file called "myaddress.txt".

Now, to Quit *vi*, we must be in **COMMAND** mode, which we still are. We then type: **:q**. Since we just wrote our file to disk, there are no unsaved changes and *vi* will quit. If we need to Quit *without* saving, we can do a *forced quit* by adding a **!** after the **q**, like this: **:q!**

Now that you have quit *vi*, list your directory contents and see if your new file `myaddress.txt` is there.

If so, type **vi myaddress.txt** at the prompt, and open your file for further editting. Use what you have learned above to change your home address to your campus address. Remember the fundamental *vi* rule: type **i** to go into *INSERT* mode, and **ESC** to go back to **COMMAND** mode. When you are done, save your file.

This ends your introduction to *Vi*. If you want to be a hacker you will have to learn how to use *Vi*, which you can do at many sites online. Here is a good one, for example.

Note: If you typed something in *vi*, or get confused about the modes and don't know what's going on, you can always **EXIT** *vi* by typing these keys: **ESC : q !**.

- **ESC** always puts you back into **COMMAND** mode,
- **:** prepares *vi* to accept commands,
- **q** means *Quit*,
- and **!** means "*force quit even if changes have not been saved*".

4.1.4 Emacs

As I said, I use Emacs as my text editor, partly because I am old and have had too many years of practice with emacs to switch. Another reason I am recommend it is that it's **Free** ! If you look at the comparison list of editors, above, you will notice that many of them have a cost. I have tried to only use free software for this course. In fact **Emacs** was one of the first applications developed by'The Free Software Foundation <http://www.fsf.org/>'_, the people behind GNU software, which you may have heard of. Cygwin, Linux, and much of the Unix world is built on the work of the FSF. Check them out!

So, let's fire up **Emacs**.

Open a Terminal

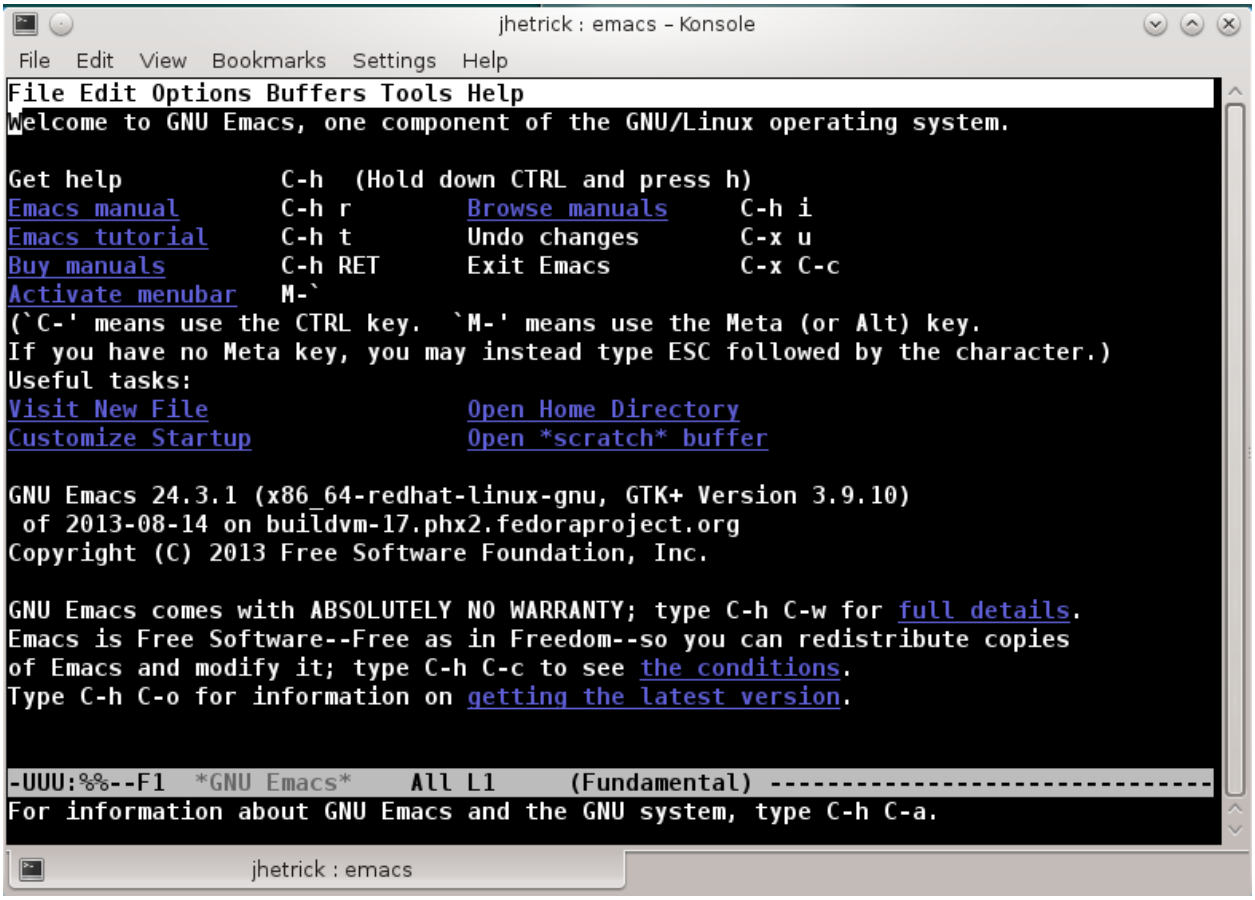
There are two ways to start Emacs.

When running emacs over a network connection, or for a quick edit, use the *Terminal (text) mode* of emacs; at the prompt, type:

emacs -nw

The **-nw** stands for "*No Window*".

You should see something like this:



There are **four areas** to the emacs screen:

- The **Tool Bar** at the top:

```
File   Edit   Options   Buffers   Tools   Help
```

It's not used in terminal mode , but is still there.

- The **Buffer Area**, the large window area in the middle of the screen.

```
Welcome to GNU Emacs, one component of the GNU/Linux operating system.

Get help           C-h  (Hold down CTRL and press h)
Emacs manual       C-h r        Browse manuals     C-h i
Emacs tutorial     C-h t        Undo changes       C-x u
Buy manuals        C-h RET      Exit Emacs         C-x C-c
Activate menubar   M-`
(`C-' means use the CTRL key.  `M-' means use the Meta (or Alt) key.
If you have no Meta key, you may instead type ESC followed by the character.)
Useful tasks:
Visit New File                  Open Home Directory
Customize Startup               Open *scratch* buffer

GNU Emacs 24.3.1 (x86_64-redhat-linux-gnu, GTK+ Version 3.9.10)
 of 2013-08-14 on buildvm-17.phx2.fedoraproject.org
Copyright (C) 2013 Free Software Foundation, Inc.

GNU Emacs comes with ABSOLUTELY NO WARRANTY; type C-h C-w for full details.
Emacs is Free Software--Free as in Freedom--so you can redistribute copies
```

```
of Emacs and modify it; type C-h C-c to see the conditions.
Type C-h C-o for information on getting the latest version.
```

The buffer has some help info at this point. This is where you edit your files. Each file that you are editing is called a **buffer** (it's really a part of memory, which becomes a file on your disk only when you write (or Save) the Buffer. You can have many buffers open at once, i.e. you can be editing many files in one emacs session.

- The **Mode Line**:

```
-UUU:%%--F1  *GNU Emacs*    All L1     (Fundamental) ------------------------------
```

This is a **highlighted bar** of text that tells you *information* about the buffer you are currently editing. We'll learn more about what these phrases mean shortly. Finally, there is a line of text below the Mode Line

- The **Mini-Buffer**

```
For information about GNU Emacs and the GNU system, type C-h C-a.
```

This is where you communiate with emacs. Emacs will put error or other messages here for you (such as the " For more information... " phrase when emacs starts up). You also can enter commands here.

Ususally you would start *emacs* (and *vi* for that matter) by giving it the name of the file you want to edit, which is opened in the *buffer* area. If there is no file with the name you gave it, emacs will open a new buffer with the name of that file.

So, let's **Quit** emacs by typing:

CTRL-x CTRL-c

In other words, *hold the *CTRL* key down and type *x* then *c*. Yup, we're going to be doing a lot of stuff like that.

Emacs uses CTRL and ESC keys as modifiers to the rest of the alphabet keys, in order to give a very rich command set available to the user through a simple text based Terminal. Later, we'll use the GUI version of emacs, and you can go back to *File->Save*, etc. But for now, we are going to learn how powerful the keyboard can be.

So, let's start emacs again with a filename. At the prompt in your Terminal, type:

emacs -nw fox.txt

Now, the **buffer** region is empty, the **mode-line** says *mothers.txt*, and the **mini-buffer** says *(New File)*. Emacs is ready for you to type something profound.

Type:

```
The quick brown fox
jumps over the lazy dog.
```

Notice also that two **'s appear in the **Mode Line**:

```
----: ******-F1  *scratch*        (Lisp Interaction)--L1--All----------------------
```

These two asterics signify that the buffer has been **modified** . In order to save these modifications, you will need to save the buffer to a file.

Emacs commands, such as *save-buffer*, *kill-emacs* (quit), are entered by typing either:

- a **CTRL**-character sequence (abreviated **C-char**), or
- a Meta (or **ESC**)-character sequence (abreviated **M-char**).

Let's finish the example we have open.

So save this buffer to a file, we type the the keyboard shortcut for the "**save-buffer**" command:

C-x C-s

Type this: **CTRL-x** then **CTRL-s**. I've done this so often, I just hold the **CTRL** button with my left pinky finkger down and press "**x**" then "**s**".

This is equivalent the usual GUI menu **File->Save** on most editors.

The *Mini-buffer* respondes with

```
Wrote /home/jhetrick/fox.txt
```

and the **Mode-line** changes to

```
----:---F1  fox.txt       (Text)--L1--All----------------------------------
```

The **Mode-line** now displays the name of the file which is connected to this buffer (fox.txt), and the "modified" asterisks (**) are gone since our changes have been saved.

We can continue to edit the file. Go ahead and add your address to the buffer . Once you have your address in the buffer, **move around with the arrow keys**. **Try using your " Delete, Backspace, Home, and End " keys** . You can see that **emacs** acts pretty much like Notepad . Aside from the text based commands like **C-x C-s**, it's pretty intuitive (at least to me!). Once your have your correct address back, save your changes with " **C-x C-s** " again.

To **Quit** emacs, remember that we have to type a text sequence. Do: **C-x C-c** By now I'm assuming you understand what " **C-x C-c**" means (CTRL-x then CTRL-c). You should have your xterm prompt back. Do an **ls** to see that your file is there. Go ahead and **cat** it which will print the contents of your file to the screen so you can see that you indeed did just edit a text file. Two more things to try: At your xterm prompt, type **emacs -nw YOURLASTNAME.txt** Notice that this opens your file for editing. Quit Emacs with **C-x C-c** .

Now, open emacs without telling it which file to open . **emacs -nw** Now you can tell emacs to **open your YOURLASTNAME.txt ** file by issuing the command: **C-x C-f** Emacs responds by asking you the name for the file you wish it to find in the **mini-buffer**:

```
Find file: ~/
```

Type **YOURLASTNAME.txt** and hit ENTER. Emacs will open your file. Quit emacs again with **C-x C-c**.

You now know the most basic skills of using **emacs**.

You can open a file, edit it, and save it, pretty much the same as you can do with Notepad.

Let's summarize:

Command	Meaning
At Terminal prompt: **emacs -nw file**	Open *file* in *emacs*
In Emacs: **C-x C-f**	Find (Open) File for editting
In Emacs: **C-x C-s**	Save Buffer to file on disk
In Emacs: **C-x C-c**	Quit Emacs and Exit

4.2 Learning More About Emacs

Even though we just saw these commands on the previous page (that may have been a few days ago), it's worth repeating these most basic commands here. With these you can write/edit a file and save it. In addition you would use the *arrow* keys to move around, plus the *Backspace* and *Delete* keys.

Command	Meaning
At Terminal prompt: **emacs -nw file**	Open *file* in *emacs*
In Emacs: **C-x C-f**	Find (Open) File for editting
In Emacs: **C-x C-s**	Save Buffer to file on disk
In Emacs: **C-x C-c**	Quit Emacs and Exit

Of course we'd like a little more functionality.

4.2.1 The GUI version of Emacs

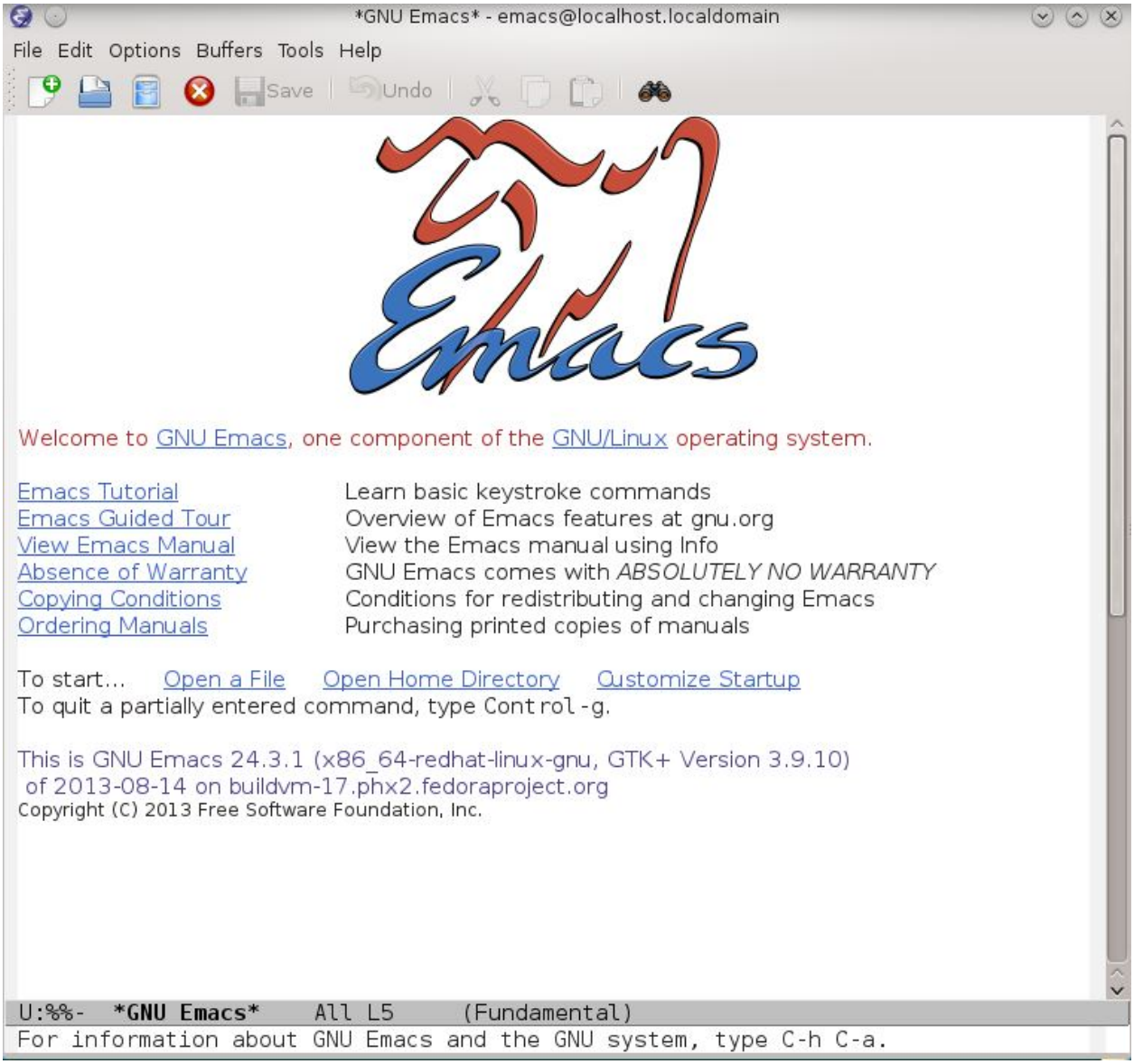
Let's bring Emacs into the 1990's.

On the previous pages I had you start emacs with the command: **emacs -nw** which opens emacs in a *text-mode* terminal window (not a GUI window). Now let's open emacs in *GUI-mode*. Open a terminal and at the prompt type:

emacs

Just **emacs**: NO **-nw** switch!

The naked `emacs` command should open an emacs window on your screen that looks like this:



Notice that GUI version has the same four areas as the text mode version:

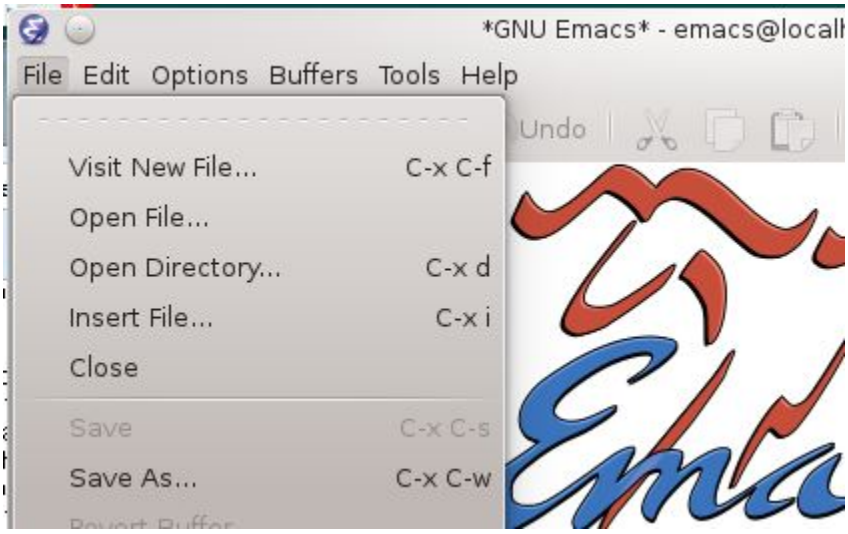
- the Toolbar
- the Buffer Area
- the Mode Line
- the Mini-Buffer

Can you identify them all? In addition you have *icons* in the Toolbar.

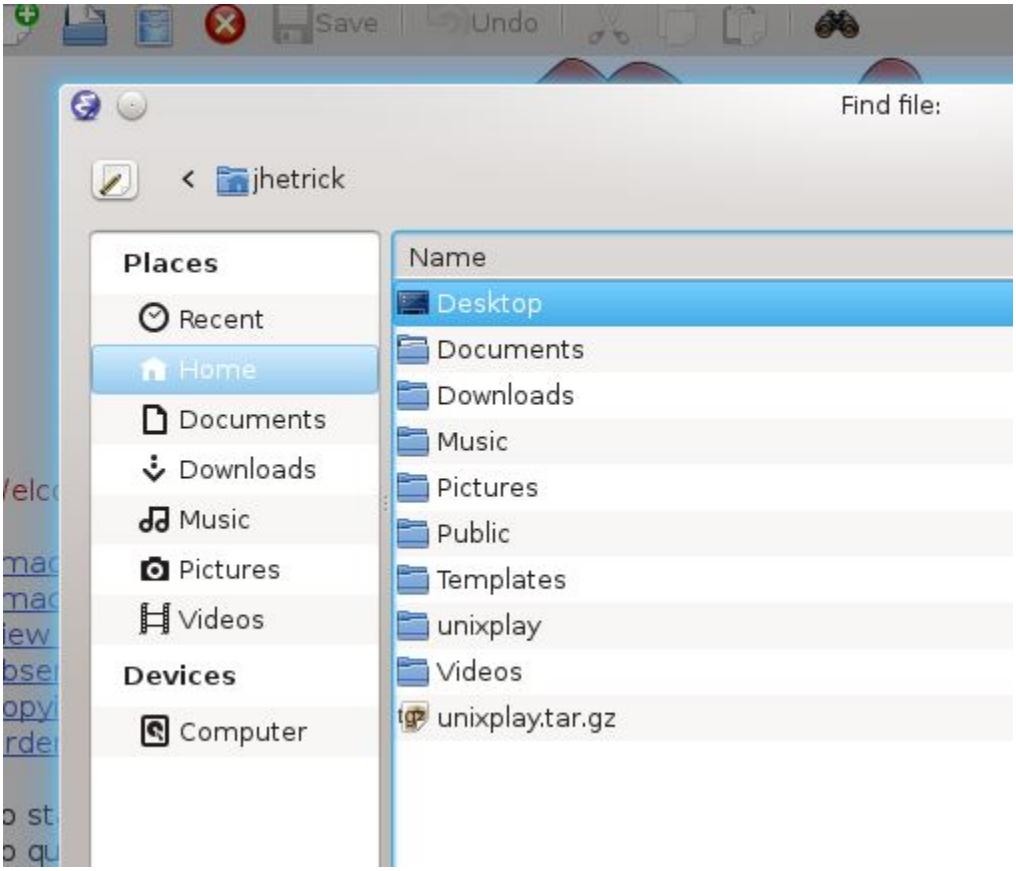
Let's open a file to play with. We're going to open it a couple of different ways.

First, click on the **File** menu in the **Toolbar**.

Click on " **Open File...**".



This opens the File browser window.

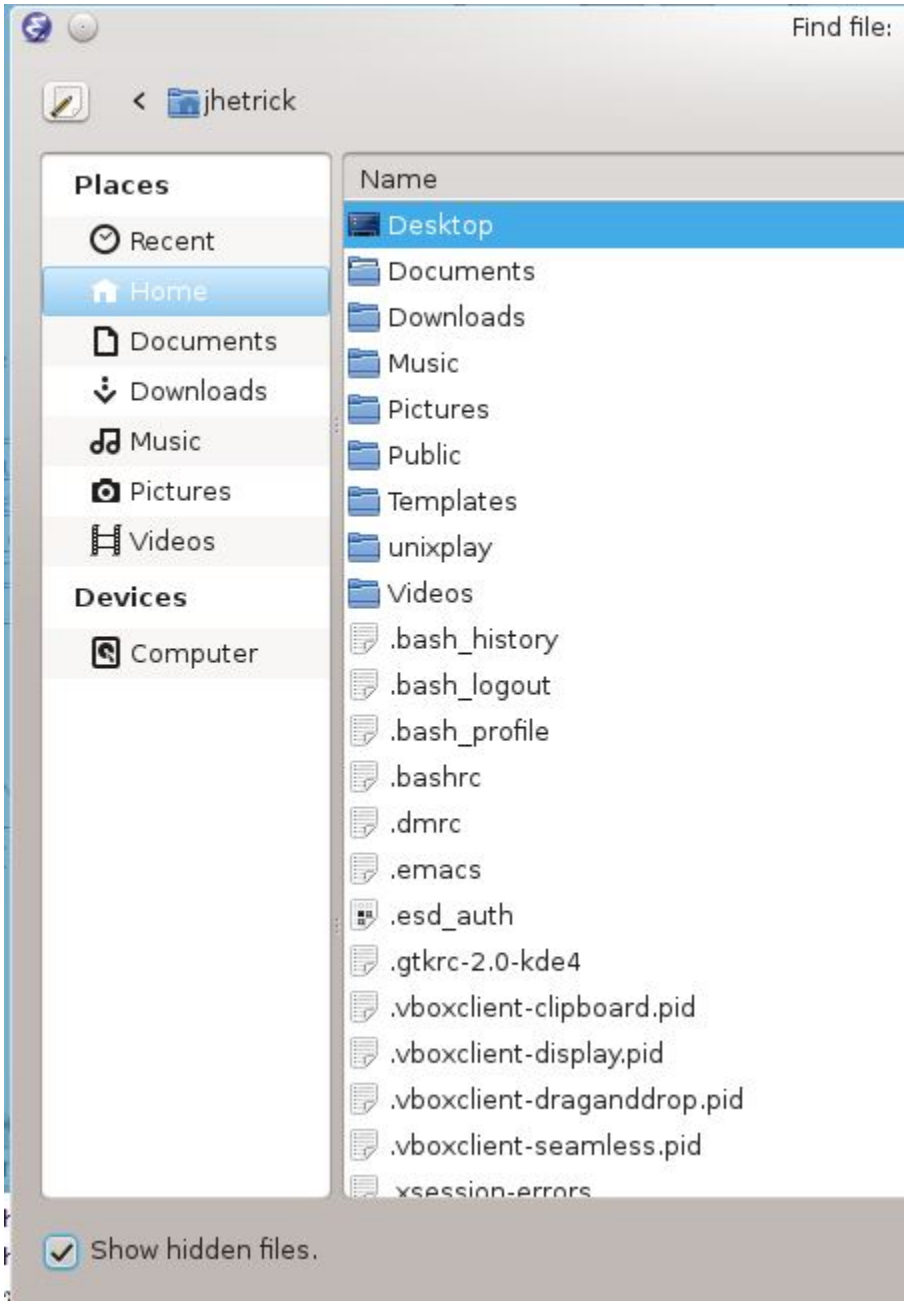


Compare this with the *command line* directory listing, using your "**l**" (ell) alias for `ls -F`.

```
sci[~]>l
Desktop/    Downloads/  Pictures/  Templates/  unixplay.tar.gz
Documents/  Music/      Public/    unixplay/   Videos/
sci[~]>
```

Going back to the *Open File* window, you see your directories and any normal files there.

But suppose you needed to edit your `.bashrc` file, which is there but not listed because its a *Hidden* unix file–it starts with a **.** (dot). If you need to see all the files in a directory, including *Hidden* files, ...simply click the *Hidden Files* button on the *Open File* explorer window.



For the moment, Click *Cancel* and close the *Open File* explorer window. We'll come back to it in a minute.

The *Emacs* window should still be open. This time (with the mouse somewhere in the *emacs* window), type:

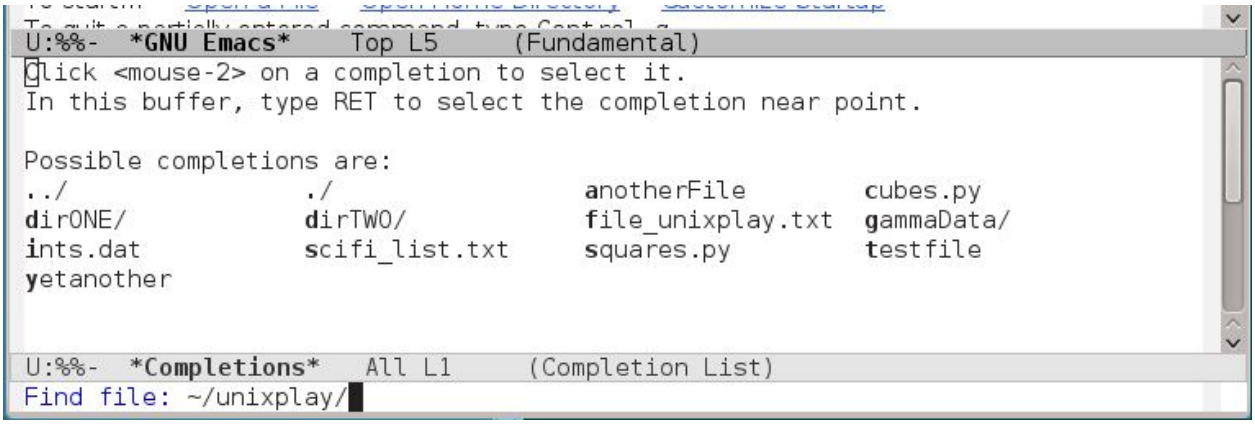
C-x C-f (Ctrl-x Ctrl-f).

In *mini-buffer* at the bottom, you will see: `Find file:  ~/`

Recall `~/` is an *alias* for `YOUR HOME DIRECTORY`, which in my case is `/home/jhetrick/`.

Notice also that the cursor is waiting for you to type a *file name* that would be in your *home directory*. You could, for example, type the path to a file that is in a *subdirectory* by typing the *path* to the file.

For example, with the `Find file: ~/` prompt still showing in the *mini-buffer*, type **unixplay/** and hit the **TAB** button, twice. The TAB tells emacs to try to complete the file name for you. Since there are many options in the `~/unixplay/` directory, emacs displays them by opening another buffer:



You can click on the file you want in that window, and emacs will open it. You can even click on one of the *subdirectories* in the `~/unixplay/` directory.

For example, **Click** on `dirTWO/`, emacs will add `dirTWO/` to the path in the *mini-buffer*.

`Find file: ~/unixplay/dirTWO/`

If you hit **TAB** again (twice), you can play the same game in the `~/unixplay/dirTWO/` directory. Emacs will show you all the files (and subdirectories) there and you can choose one of those to edit.

At this point, if you've been following along, the *mini-buffer* should be waiting for you to choose a file in the `~/unixplay/dirTWO/` directory.

Before we open a file, let's **Abort** this *Find File* process. This is a very useful thing to know how to do. Suppose you type the wrong command sequence, or decide to do something different, you can always **ABORT** the current command by typing **C-g** (Ctrl-g).

In the Emacs window, type **C-g** (Ctrl-g).

The mini-buffer says "`Quit`". This is because we **Abort**ed the command to *find* a file.

In general, you can always abort whatever you are doing (or were trying to do), by typing **C-g**.

You can also *Abort* an action by clicking on the *Red (X)* in the *Toolbar*.

Note: To **Abort** a command, type: **C-g** (Ctrl-g), or the *Red (X)* in the *Toolbar*.

Finally, you can open the file using the method you already know!–the text sequence mode we used before. This is the method to use when connected to a computer over the network. In the emacs window, type:

C-x C-f

Same thing: you get "`Find File:`" in the mini-buffer.

This time let's open a file.

If you type the name of a file which does not exist, the buffer is renamed to this name, and the words **(New File)** appears in the mini-buffer.

Go ahead and type the name of a new file, something like:

zzz

Do you see the `(New File)` text in the mini-buffer?

Now type some characters in the buffer window. Anything will do.

OK. Let's *NOT* save this file. We can *Kill* the buffer by clicking on the **(X)** in the Tool Icons.

This brings up a dialog window that asks you: `Buffer ... modified; kill anyway? [Yes]/[No]`

Click **[No]** (i.e. don't *kill* it)

Now let's do the same action: *kill* the buffer, using text-based commands. In the emacs window, where you still have the new **zzz** buffer, type:

C-x k (Ctrl-x, followed by typing the k key)

The mini-buffer should say: `Kill buffer: (default zzz)`

type **ENTER**

The *mini-buffer* will respond with: `Buffer zzz modified; kill anyway? (yes or no)`

With the cursor at the right end of the mini-buffer (click on the end of this text if it's not there already).

type: *yes**

The `zzz` buffer should disappear.

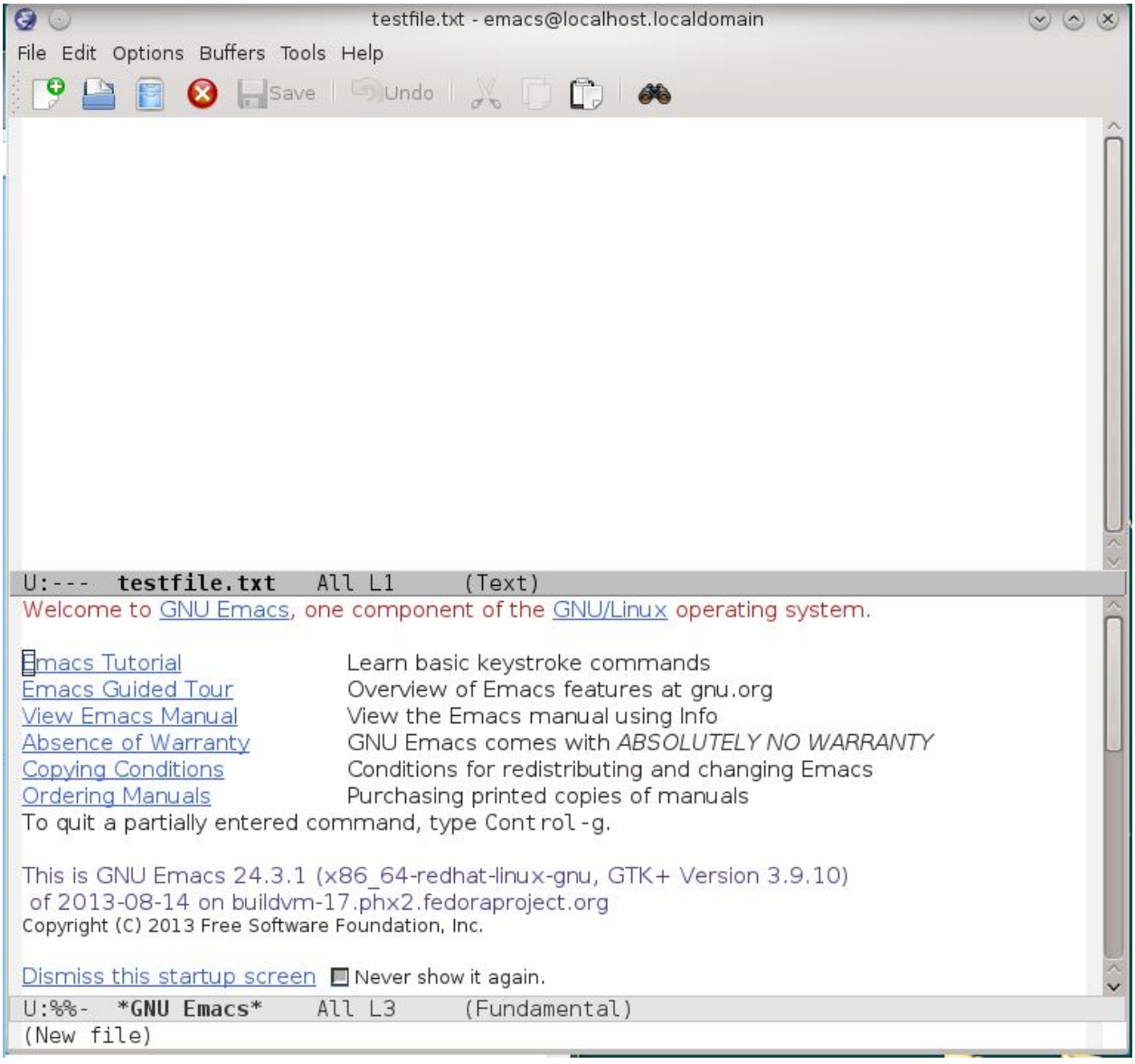
Last thing: let's open a file, edit, and save it.

At the terminal prompt, type

emacs testfile.txt

(I always give my files a suffix `.txt` when they are just text)

This should open emacs with two screens. In the top is your new file, `testfile.txt`. In the lower window is the *Welcome to *EMACS* screen.

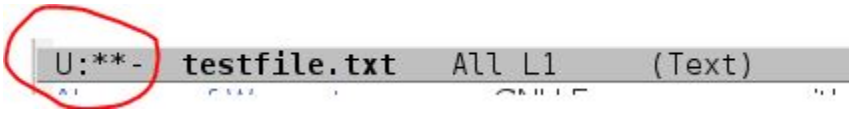


You don't need the *Welcome Screen* any more, so **click** on the *Checkbox* which says "`Never show it again`",

then *click* the line of text to its left, which says "*Dismiss this startup screen*". The *Welcome* screen will go away, and never come back.

Click in the upper window, and **type your name**.

Before you *Save the ''testfile.txt'' file, notice that after you typed your name, the *mode-line*, where emacs shows you useful information, there are two new `\*\*`'s. This means that the *buffer* is modified–there are changes that need to be saved.



Now, to *Save* this "*buffer*" to a file, you surely know to **Click** `File->Save` in the *Tool Bar*.

If you do so, nothing happens, except those `\*\*`'s disappear.

Recall: To do the same with a *Text Mode* command, the command is: **C-x C-s** (Ctrl-x Ctrl-s).

At this point you can use either the intuitive GUI (menu/mouse) mode to click on buttons, or the Text mode keyboard commands, to open, modify, and save

Since there are senior Computer Science students taking this course along side freshmen Physics students, let me through out a little candy for those already familiar with *remote logins* and *ssh* (don't worry freshmen, we'll learn about those in Chapter 10).

Edit a file on a remote unix machine

Suppose you have an account on another machine called: `secrets.nca.gov`. From your laptop, you can do:
C-x C-f (to open a file)
Then, erase (backspace over) the **~/** in the *mini-buffer* and put for the filename:
/secrets.nca.gov:missilecodes.txt
Note the beginning "**/**" (without the usual ~), and the "**:**" separating the filename.
This tells emacs to open the *remote file* for editting *locally*. Emacs will open an *sftp* connection, grab your file (asking for your remote password, of course) and open it in a buffer on your laptop here. When you save it, with: **C-x C-s** it goes out through the network and puts the new version back "over there".

4.2.2 GUI v. Text

Let's summarize, the *GUI* and *Text-mode* methods.

To open a file (`find-file`)	
GUI (menu/mouse) Mode	Click: *File->Open File*
Text Mode	type: **C-x C-f**

To kill a buffer (`kill-buffer`)	
GUI (menu/mouse) Mode	Click: *File->Close*
Text Mode	type **C-x C-f**

To save a buffer (`save-buffer`)	
GUI (menu/mouse) Mode	Click: *File->Save*
Text Mode	type **C-x C-s**

So now let's step up the pace a bit.

Your emacs window should still be open. Go ahead and close it (quit). Remember how?

Now **cd** to your *home dir* and make a *new subdirectory* called **editplay** then cd into it.

cd editplay.

Now save the following file into this directory: stairway.txt

Remember, you can download the file with the `curl` command, by copying and pasting this text to the command line:

curl -O http://dirac.physics.pacific.edu/phys/jhetrick/www/phys27/download/stairway.txt

(I know I could have included this file in the `unixplay.tar.gz` bundle, but I wanted to remind you about the very useful `curl` command. Just remember you need the **-O** (minus capital Oh) switch to save the file to your current directory)

Open it in emacs (use the GUI version of emacs), by typing

emacs stairway.txt

4.2.3 Movement

Let's start by summarizing movement commands.

In graphical GUI mode, things work pretty much as you would expect from any other word processors like MS Word, Notepad, etc.

Most normal movement cammands work:

- Right, Left, Up, Down Arrows
- Page Up, Page Down
- Home, End
- and the Scroll Bar on the left side

Go ahead and try these.

In *Text-mode*, movement can be achieved with the same keys usually, but sometimes on a remote machine the terminal emulation can be strange. For this reason, emacs has some text-based key strokes which implement movement commands. These are (recall: *C-* means *Ctrl-*, and *M-* means *ESC* then *key*)

Emacs *Text-mode* **Movement** commands	
C-a	Beginning of line (Home)
C-e	End of line (End)
M-f	Forward one whole *word*
M-b	Back one word
M-e	Next sentence
M-a	Previous sentence
C-v	Next screen
M-v	Previous screen
M-<	Beginning of buffer
M->	End of buffer

Notice that the GUI version of Emacs (the window you have open) recognizes all the Text-Based commands as well. In fact, when I am in GUI mode, I still use the text-based versions of most commands. It's a lot faster–your hands don't have to leave the keyboard to grab the mouse, so you can really learn to type documents fast.

Try all of the text-based movement commands above now on the `stairway.txt` file.

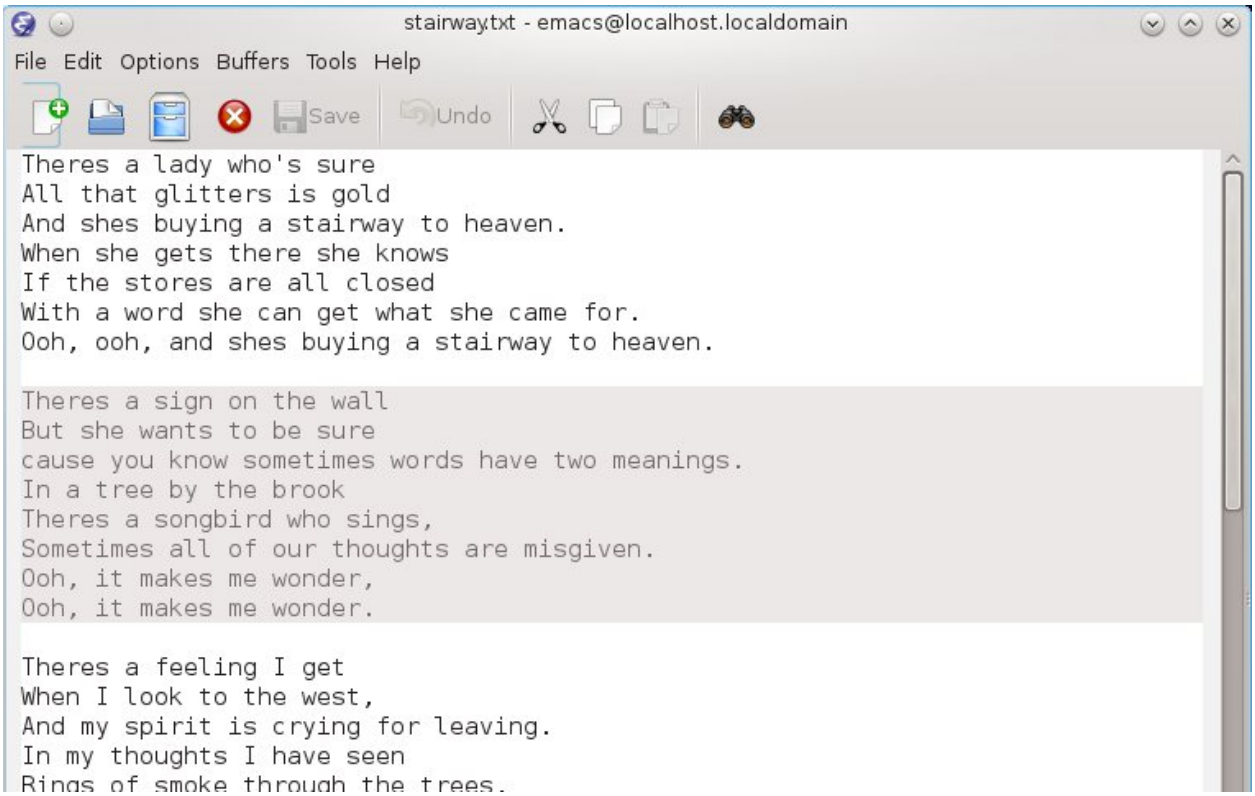
4.2.4 Cutting and Pasting

In GUI-mode

In your open "stairway.txt" buffer,

Click/drag your mouse in the text to highlight a region of text as shown below.

(pick your favorite stanza of the song)



In *Emacs lingo* this highlighted area is called the *Region*. We'll come back to this concept in a minute.

With the region still highlighted, click the **Edit** menu item on the Toolbar, then click **Cut**

The highlighted region should disappear. This is the same as typing **CTRL-x** in MS Word. You have "*Cut*" the region of text, and at the same time *copied* the *region* to the *Clipboard*.

Now go to the Toolbar and click **Edit->Paste**. This should put back the text you just "Cut". It will be Pasted at the place where the cursor is located. Notice too that under **Edit** you also have the option to **Undo** your changes.

Try this: Highlight some text again Cut the text by doing: **Edit->Cut** Undo your "Cut" by clicking: **Edit->Undo** This should put back the text you Cut.

In Text-mode

Now let's try this in *Text mode*.

This is how you cut/copy text when you are editing using emacs on a remote machine via a text terminal, or sometimes in GUI mode when you don't want to grab the mouse.

In your open emacs window, *using the arrow keys*, **move the cursor** to the beginning of a stanza of the song. Now, *Set the Mark*, by typing

C-spacebar (Ctrl-SPACEBAR).

The words `Mark set` will appear in the *mini-buffer*. The *Region* is now the area between the *Mark* and the *Cursor* .

Having set the *Mark* at the beginning of the stanza (when you typed *C-spacebar*), **move the cursor to end of the stanza**, using the arrow keys. Now type

C-w (`cut-region`)

This is the text-mode key sequence that does the "*Cut*".

Similarly, the text-mode key sequence to "*Copy*" is

M-w (`copy-region`).

The text-mode key sequence to **Undo** is

C-x u (`undo`)

Try this now.

Does the text you "Cut" come back (i.e. was your "Cut" undone)?

Pasting

In GUI mode, pasting the text you just cut is easy. Highlight the text, click *Edit->Cut* or *Edit->Copy*, put the cursor where you want to put the text, and click *Edit->Paste*. Pretty easy.

In Text-Mode , the `Paste` command is

C-y ("y" for yank, as in "yank it out of the clipboard, and put it here).

Go back and "Cut" a stanza again:

To "Paste" the text, move the cursor to the place you want to paste it, then type:

C-y

This should "Paste" the text you just "Cut" at that point.

Kill a Whole Line of Text

Another very useful thing to do is to *kill a single line* of text.

Move your cursor to the beginning of a line, although you actually can do this anyhwere in the buffer. Type:

C-k

This will *kill* all the text from the cursor to the end of the line. Try it on the `stairway.txt` file.

Put the cursor at the beginning (or the middle) of a line of text.

```
Theres a sign on the wall
But she wants to be sure
cause you know sometimes words have two meanings.
In a tree by the brook
Theres a songbird who sings,
```

Type:

C-k

```
Theres a sign on the wall
But she wants to be sure

In a tree by the brook
Theres a songbird who sings,
```

Notice that this *does not delete the empty line*, but merely removes the text. If you also want to also delete the blank line, hit **C-k** again. So, typing **C-k C-k** *kills* both the line of text, *and* moves the text below, up by one line.

The text that was removed is now in the *Clipboard* (or "yank buffer"), and so if you want to place it somewhere in your document, you simply move the cursor to that point, and type **C-y**.

If you did the *double* **C-k C-k**, then the blank line (actually the trailing *newline* character) is also saved in the Clipboard. So, if you *paste* (yank) the text into a new location, it will now move the rest of the text *down* one line.

You will need to practice a little, so go ahead and hack up the `stairway.txt` file by cutting and pasting lines, parts of lines, etc. Use both the *single* **C-k** and the *double* **C-k C-k** to get used to the difference.

Let's summarize Cutting and Pasting:

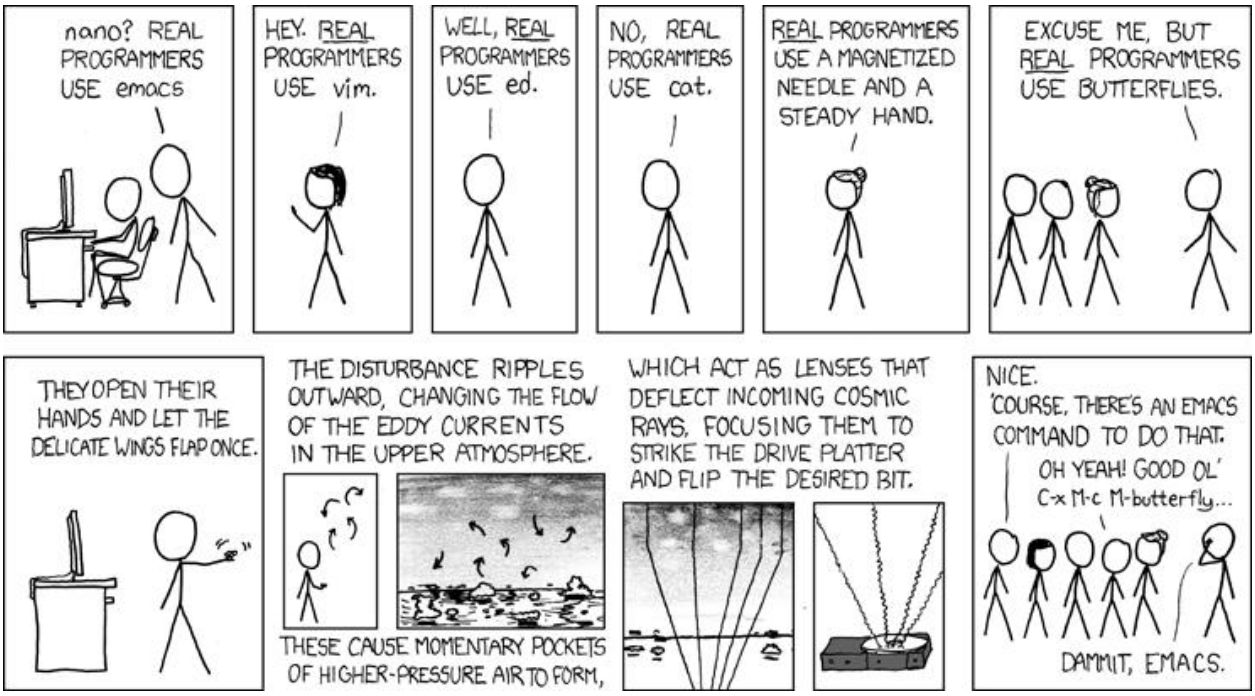
Cut Text (`kill-region`)	
GUI (menu/mouse) Mode	Highlight text; *Edit->Cut*
Text Mode	Set *Region* with *Mark* and *Cursor*; type: **C-w**

Copy Text (`copy-region`)	
GUI (menu/mouse) Mode	Highlight text; *Edit->Copy*
Text Mode	Set *Region* with *Mark* and *Cursor*; type: **M-w**

Paste Text (`yank`)	
GUI (menu/mouse) Mode	Highlight text; *Edit->Paste*
Text Mode	Place *Cursor* at insertion point; type: **C-y**

4.3 Intermediate Emacs

Now that you've had a little introduction to Emacs and some of its keystrokes, you may be able to appreciate this comic.



You haven't gotten much chance to experience the true power of emacs yet, though.

Unfortunately we won't really be firing up the warp engines; there's plenty online if you want to delve deeper. Please have a look at this page:

A Tour of Emacs

You'll notice that we've talked about several of the commands you see there. But there is MUCH more. Emacs has a built-in programming language which let's you write functions and programs which you can bind to keystrokes.

Let's look at just a few more features before we leave our introduction to Emacs.

4.3.1 Customization: The `.emacs` Startup File

When Emacs starts, it looks for a file in your home directory called *hidden* file called: `.emacs`. This file contains commands written in the Emacs programming language called *Lisp* (more specifically it is *Emacs Lisp* –a variation of *Lisp*).

You can customize Emacs to alter its behavior to your liking. For example, when Emacs started up before, we clicked on the line that said

```
Dismiss this startup screen
```

Actually that text was linked to a Lisp function which wrote the following text in your `.emacs` file.

```
'(inhibit-startup-message t)
```

Open a shell window (if you don't have one open). Type **ls -a** (or better you *alias* **la**) for a listing of *all files. You should see `.emacs` in the list.

Have a look at it with **less**. You should see something like this:

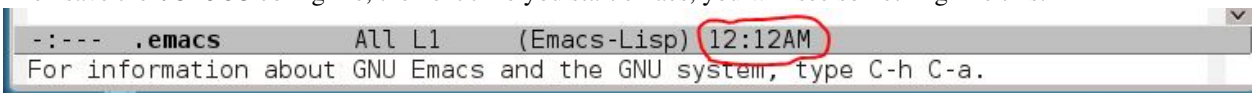
```
;; .emacs

(custom-set-variables
 ;; custom-set-variables was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(diff-switches "-u")
 '(inhibit-startup-screen t))
```

This may look a bit terrifying, but here's a quick example of the kind of thing you can do.

Try this: Edit your `.emacs` file in Emacs (type: **emacs .emacs**), and add the following line to the bottom

```
(display-time)
```

Then save the `.emacs` config file, the next time you start emacs, you will see something like this:



The time is now displayed in the *mode-line*. Not going to change your life, but at least you see how the customizations work. Consult the web for more customization ideas.

4.3.2 Find and Replace

Let's do a simple task which might come up while you are editing. It might seem stupid, but it will lead us into a more advanced topic.

Open Emacs by typing: **emacs** at the prompt.

With no filename (new or existing) give, Emacs start on a buffer called `*scratch*` that looks like this:

```
;; This buffer is for notes you don't want to save, and for Lisp evaluation.
;; If you want to create a file, visit that file with C-x C-f,
;; then enter the text in that file's own buffer.


-UUU:----F1  *scratch*      All L5     (Lisp Interaction)----10:32AM 0.51--------
For information about GNU Emacs and the GNU system, type <f1> C-a.
```

If you want a clean new file in which to put text, you either need to supply a *filename* on the command line when you start emacs, or "*visit*" the file by asking emacs to *open a file* with **C-x C-f**, and giving it a name.

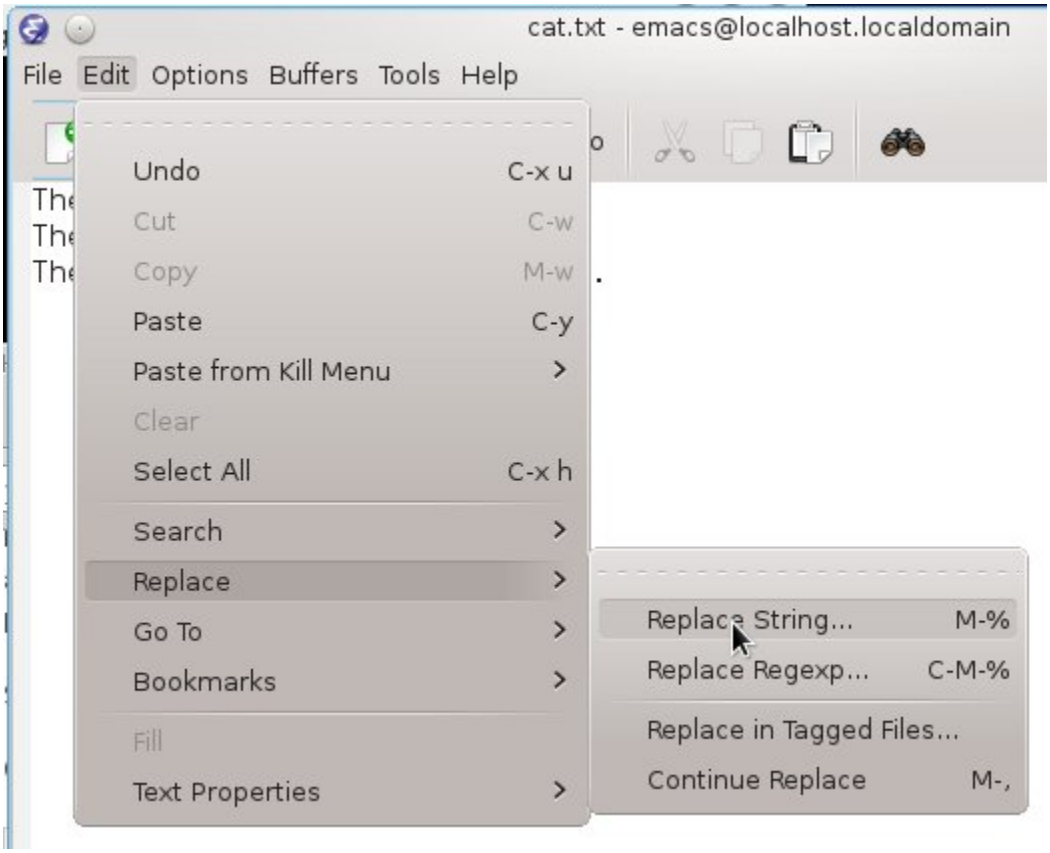
So, let's call this file, `cat.txt`. Copy the text fragment below into the buffer

```
The brown cat is fat
The fat cat is old
Therefore, the brown cat is old.
```

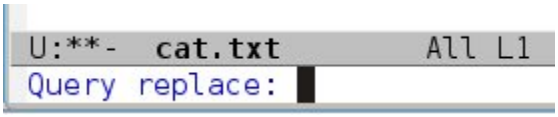
Let's substitute "`dog`" for "`cat`" in this file. Easy enough.

Make sure the cursor is at the beginning of the file (on the first character of the first line). You do this so that everything below will be subject to the Search/Replace.

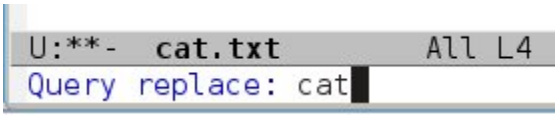
Click on " **Edit->Replace->Replace String** in the *Toolbar*.



This will make Emacs prompt you in the mini-buffer with the words: ‘‘Query replace: ‘‘.

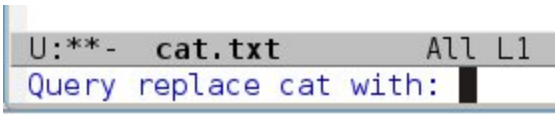


In the mini-buffer type: **cat** then **RETURN**.

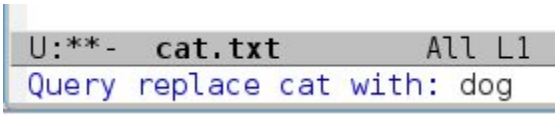


(Remember if anything goes wrong–you can always "Abort" with **C-g** or the Toolbar->Red (X), and then start over).

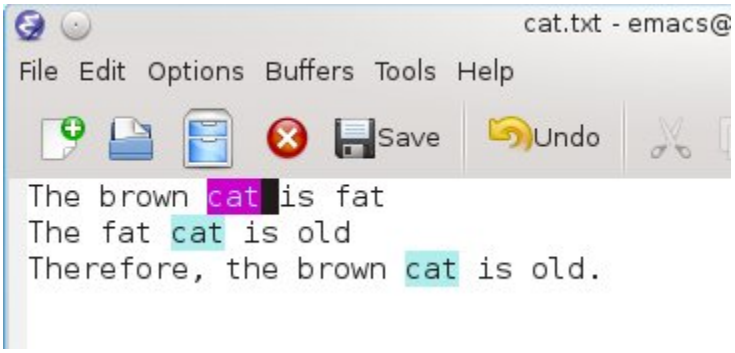
The mini-buffer responds with: ‘‘Query replace cat with: ‘‘



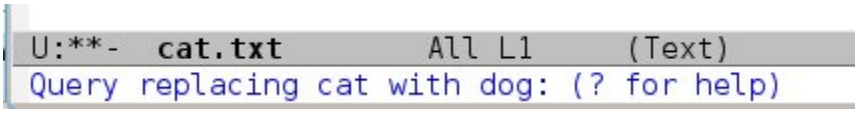
Type: **dog** then **RETURN**



Emacs highlights the next instance of `cat`, and waits for your input.

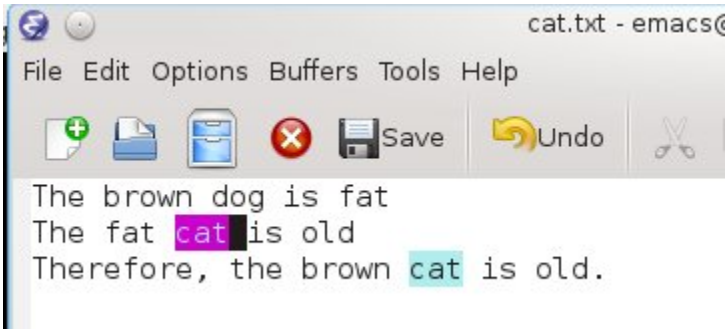


Also the mini-buffer has changed to: `Query replacing cat with dog: (? for help)`



To *accept* this Replacement (i.e. *cat->dog*), just hit the **Spacebar** .

Emacs *replaces* `cat` with `dog`, jumps to the next `cat` and waits for your choice on the next occurance of `cat`:



This time, hit the "**n**" key (for *No*, do not replace this occurance) instead of *Spacebar*. Emacs will skip that replacement and jump to the next `cat`.

Hit the **Spacebar** on this next one.

Emacs says: "`Replaced 2 occurances`", which is what it did.

Now you should have the following text in your emacs buffer:

```
The brown dog is fat
The fat cat is old
Therefore, the brown dog is old.
```

The Text-mode way

Let's change these "dogs" back to "cats". But this time we'll do it the Text-mode way.

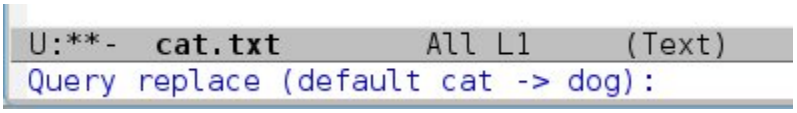
Put the cursor at the beginning of the file.

You may have noticed when you clicked *Edit-Replace-Replace String* in the Toolbar, that to the right of the menu item was the character string: *M-%*. This is the *Text-mode key binding* for the *Query/Replace* command (`query-replace`).

Type: **M-%** (Yeah: **ESC** followed by "**%**". Understand the XKCD comic better now?)

This initiates a "Query/Replace" command, just like you did above with your mouse in the GUI Toolbar.

This time, since you just previously did a *Query/Replace*, Emacs offers the same replacement strings as the "*Default*".



If you just hit **RETURN**, you will accept the *Default*: replacing `cat` with `dog` again.

Since we skipped one `cat` (the "fat cat") in the previous execise, there should be one `cat` left to transform to a `dog`.

Go ahead and hit **RETURN**. Emacs will jump to that occurance of `cat` and wait for you to hit **Spacebar** to *accept* the swap.

Now, the text should be all about *dogs*.

Let's practice one more time, and change all the *dogs* back to *cats*.

Put the cursor at the beginning of the text.

Type: **M-%**

In the mini-buffer, after "‘‘Query replace (Default: cat - >dog): ‘‘", type **dog**

Then after "`with:`", type **cat**

Emacs jumps to the first instance of "`dog`".

This time, let's just do them all at once–we don't want to go through each instance of "`dog`" and answer with "**Spacebar**" or "**n**".

Simply type **!** (an exclaimation symbol).

Emacs will change all occurances of "`dog`" to "`cat`" throughout the file.

Let's summarize Query/Replace:

Query (Search) / Replace (`query-replace`)	
GUI (menu/mouse) Mode	*Edit->Replace->Replace String*
Text Mode	type: **M-%**

Replacement Choices	
Spacebar	to accept replacement
n	to regect the change
!	to accept all replacements
?	for help

4.3.3 Regular Expression Matching

Try this; in Emacs, open a new file `regexp.txt` and copy the following text into it.

```
zzz1234
zyx2345
zzz5555
uuu9845

u6638v
v3948w
w8395x
x7730y
y2349z
```

Let's suppose this is data from some log file, and you need to extract the numbers from each line.

You *could* do it *by hand*, i.e. move around and use *Backspace* and *Delete*. Yuk.

You *could* do *six Query/Replace* operations: replacing the characters `u,v,w,x,y,z` with empty replacement strings (""), i.e. just hit RETURN when asked for the Replace string:

```
"u" -> ""
"v" -> ""
...
"y" -> ""
"z" -> ""
```

That would do it, but it's a lot of different Query/Replace operations. Yuk, again.

There's a really cool way to do it using *Regular Expressions*: a match *pattern* rather than an *exact* match, like `cat`.

What we want to do is to remove all the alphabet characters (a-z) and leave the numbers (0-9). So we need a pattern that will only match alphabet characters.

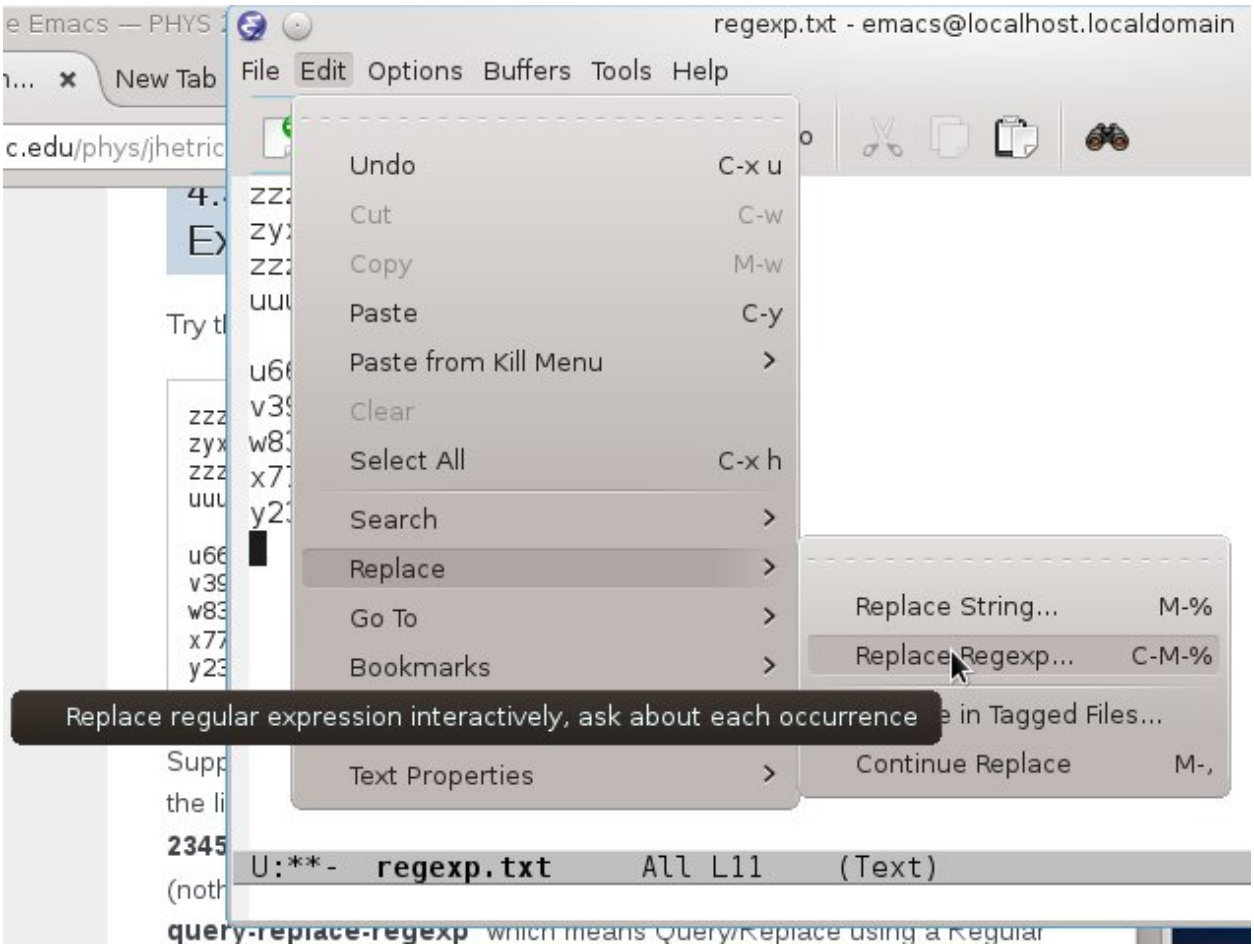
Make sure your cursor is at the top of the text window. Emacs will (for now) only search *Forward* for matches. So, if you start the Query/Replace with the cursor at the end of the text, it will not find anything.

You can quickly move the cursor to the beginning of the buffer by typing: **C-HOME** (Ctrl-HOME). Similarly, **C-END** takes you to the end of the buffer.

To invoke a **Query/Replace using a Regular Expression** (`query-replace-regexp`), we can either use

- the GUI method: *Edit->Replace-Replace Regexp...*
- or the Text-mode keyboard sequence: **M-C-%**

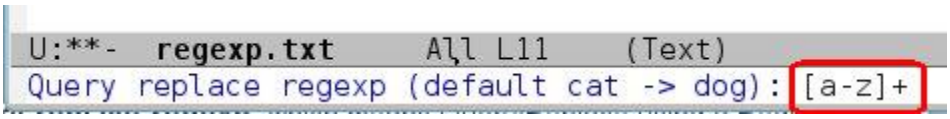
Yup: that's **ESC** followed by **Ctrl-%**, including the *SHIFT* key needed to get the %. Unless you're pretty dextrous, just use the GUI way:



In the *mini-buffer* you see the prompt for the `regexp`, the *Regular Expression* (and because I didn't close my Emacs window since the last Query/Replace, you see the `default: cat -> dog` from before).

Emacs is now asking for a `regexp`–a *PATTERN* to use for matching.

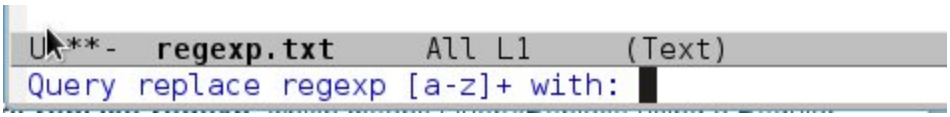
Type: **[a-z]+** then **RETURN**



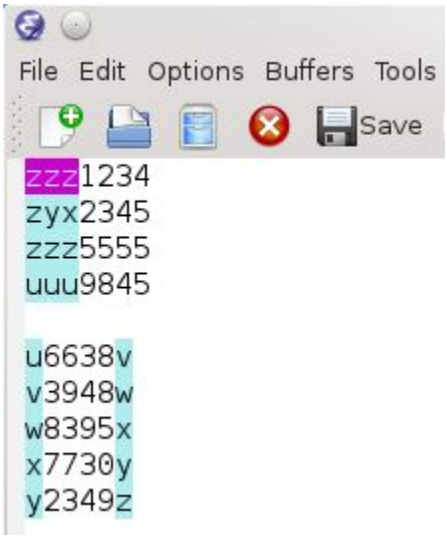
The *pattern* is `[a-z]+`. Here's what the parts mean:

- `[ ]` says to match *any* of the character inside the brackets
- `a-z` means any letter between `a` and `z` (if you wanted ANY letter, including CAPs you would do `[A-Za-z]`)
- `+` means match *one or more* of "these"

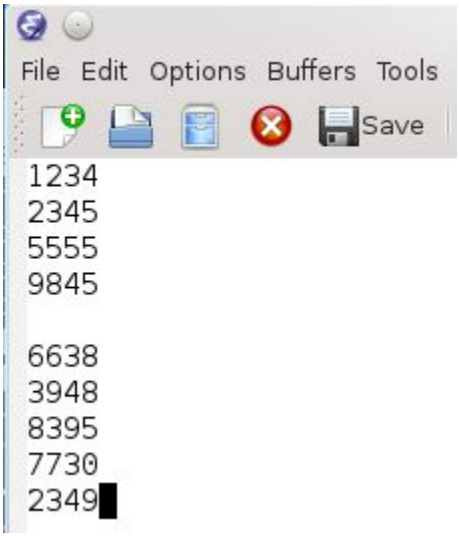
When you hit *RETURN* above, Emacs is now waiting for you to supply the text to replace the thing matched (zzz, zxy, etc.)



If you simply hit **RETURN** again, meaning replace with *nothing* (the empty string), Emacs highlights *everything* that matches the *regular expression pattern* `[a-z]+`.



Hit the **!** key, and *voila!*, all the letter characters are removed and your data is scrubbed clean.



Like many topics in this course, this example is just to let you know that *Regular Expression Substitution* can be very useful. In addition to match a *pattern*, you can also *save* the *match data* and put it in the substituion string. For example, with one Query/Replace/Regexp command you could make the following substitution:

```
zzz\ **1**\ abc -> filename\ **1**\ .dat
zyx\ **2**\ cba -> filename\ **2**\ .dat
qxz\ **3**\ bac -> filename\ **3**\ .dat
```

There are many online resources to learn about *Emacs Regular Expressions*. Try:

- http://www.emacswiki.org/emacs/RegularExpression
- http://www.gnu.org/software/emacs/tour/

Regular Expressions are used in most advanced programming languages, Python, PERL, Java, Javascript, PHP, etc. If you continue to develop your scientific programming skills, you will encounter them again.

In the above discussion, I have tried to always include the *emacs command* for each action. If you look back at the summaries, you see things like:

Action	Emacs Command	Keybinding
To open a file	`(find-file)`	**C-x C-f**
To kill a buffer	`(kill-buffer)`	**C-x C-k**
Paste Text	`(yank)`	**C-y**
...hundreds more		

Here's a small sample of some of the Emacs commands

Emacs: execute-extended-command (M-x)

If you happen to know the *Emacs command* for an *action*, you can have Emacs initiate that action by typing **M-x**, then typing the command in the mini-buffer.

Let's try it.

If you don't have an open Emacs window running, start Emacs.

Now *visit* a file ("visit" is the Emacs lingo for openning a file in a buffer), using the usual keyboard way: type **C-x C-f**

Emacs responds with

```
Find file: ~/
```

in the mini-buffer. You could continue by typing the name of a file to edit, like `stairway.txt`.

However, we don't need to edit the file. So, *abort* the action by typing **C-g** (or clicking the Red (X) in the toolbar.

Once you see `Quit`, indicating that the action was aborted, type:

M-x (ESC-x)

The mini-buffer shows simply

```
M-x
```

Emacs is waiting for you to type a command.

Type

find-file and hit **RETURN**

Emacs response with:

```
Find file: ~/
```

the same as it did above, when you typed **C-x C-f**. You've just learned *yet another way* to make Emacs do things, by using the `(execute-extended-command)` command, which is bound to the keystrokes **M-x**.

A command I use often is the `(goto-line)` command. Suppose you programming and the compiler says you have a bug in your code at line 972.

You can open the program file in Emacs and type **M-x**. Emacs responds with `M-x` in the mini-buffer. To this you type **goto-line** and **RETURN**.

Emacs responds with: ‘‘Goto line: ‘‘

You type: **972 RETURN**

and Emacs jumps to line 972 in your program file so that you can edit out your syntax error.

This would be written in "Emacs lingo" as **M-x goto-line**.

Of course there is a keyboard shortcut way to do this too: **M-g g** (ESC g g), but I find it easier to remember **M-x goto-line**.

4.3.4 Emacs Windows and Buffers

Like most windowed (GUI) applications, you can resize the Emacs window by dragging the corner to make it bigger. You can also open several files/buffers in one emacs session. This is useful when you want to cut and paste across buffers, compare them, or use a host of other

Emacs "modes" |

Here's an example of a useful mode, do this:

M-x calendar

This should pop up a second buffer in a smaller window at the bottom of your emacs window. This buffer will be in "Calendar mode" and thus responds to various keystrokes differently than a normal buffer. For example, if you click on a date that is not today's, the cursor moves there. Typing a period "**.**" takes the cursor back to today's date. Just a very small example of the behavior of Emacs modes.

Now how do you get rid of the calendar?

Well, you know how to kill a buffer: **C-x k**

If your cursor is in the calendar window and you type the above keystroke **C-x k**, you will delete the calendar buffer. Go ahead and try it .

You may see another buffer in it's place, like ***messages*** or another emacs' "operating buffers". You can kill that one too with **C-x k**.

But what about the *Window*?

Even though you killed the buffer the window is still there.

To manipulate windows:

If your cursor is in the window you want to keep , you type

C-x 1 (think "*keep this ONE*").

If you want to delete the window your cursor is in , you type

C-x 0 (think "*this buffer is a ZERO*").

If you have two windows open, you can jump to the other window, by typing

C-x o (*that's an "oh" for "other"*).

Or you can just click in it, which will take your cursor there as well. If you have only one window and want to split your screen, type

C-x 2.

Note that in each window you can type **C-x C-f** to open a separate file in each. Of course you can also use the Toolbar. Finally, you can drag the *mode-line* (the info-bar between buffers) up or down to make the windows bigger/smaller.

Let's practice:

In Emacs, type **C-x 2** to split the screen into two windows.

Drag the mode-line in the middle of the screen to resize the windows.

Type **C-x C-f** to visit a file. Any file will do. In fact, when you get ‘‘Find File: ‘‘, hit the **TAB** key, *twice*.

This will list all the files in the current directory. You can click on one with the mouse. Hitting **TAB** again will scroll to the next page if there are more files than fit on one screen.

Window control summary:

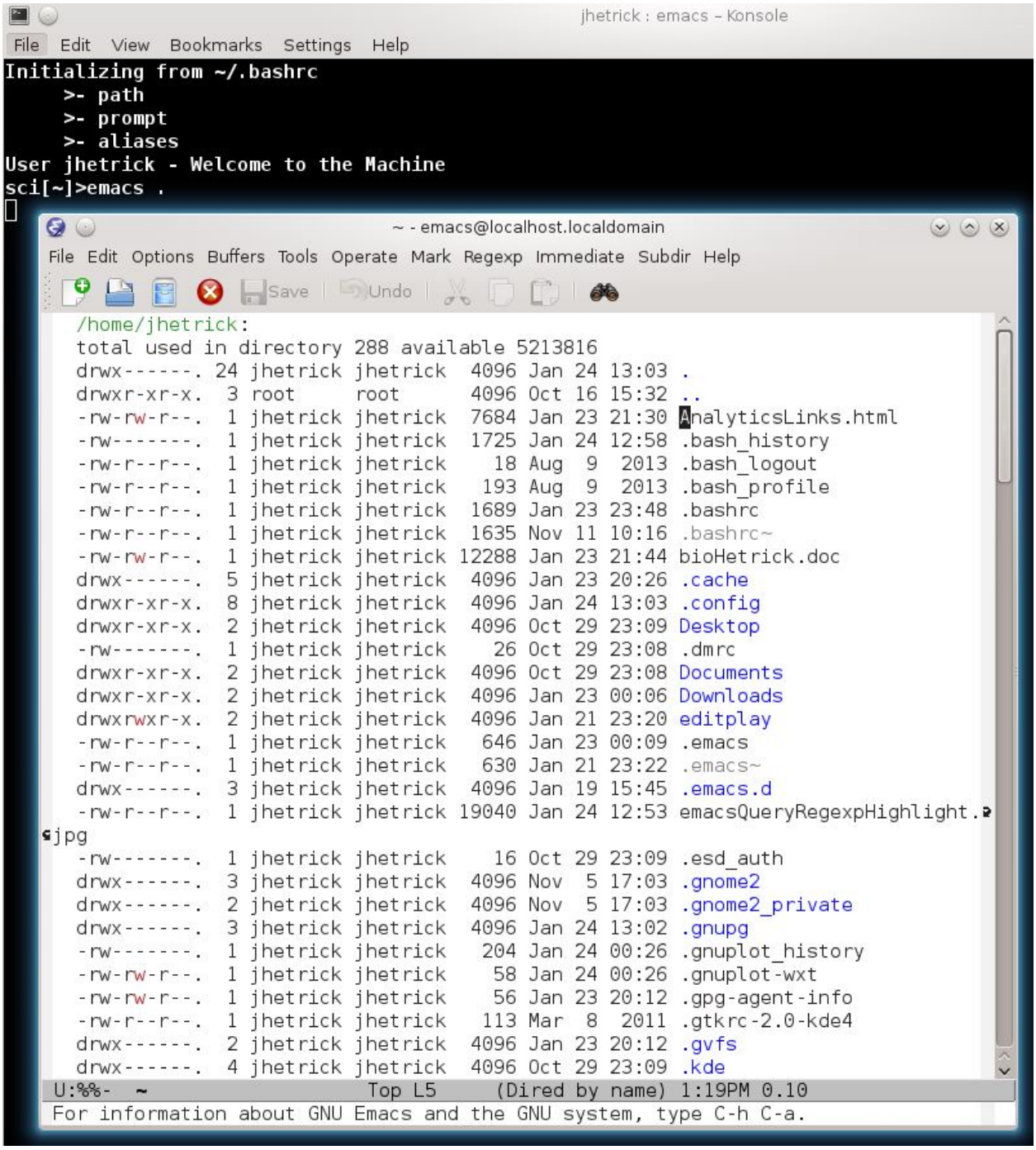
Action	Keystrokes
Close this window	**C-x 0**
Close *all but* this window	**C-x 1**
Split current window into two	**C-x 2**
Split window *horizontally*	**C-x 3**
Jump to other window	**C-x o**
Compare text in two windows	**M-x compare-windows**

4.3.5 Dired

One last useful trick. You can open a **directory** in Emacs:

In an shell terminal, type **emacs .** (i.e. open “the current directory”).

This will open Emacs with a long listing of your files.



For more info on dired mode see: GNU Emacs Manual/Dired

This is `dired mode` (*DIR*ectory *ED*it). There are many handy things you can do in dired mode.

For example, you can **mark files for deletion** by typing **d** with the cursor on the line for a each file. After marking a number of files, typing **x** will "*expunge*" (delete) all marked files. Similarly, you can mark files to be *moved, copied*, etc. in bulk. This is handy if you need to do operations on a large number of files.

Sure, you can do this easily by *Ctrl-Clicking* to mark a number of files in the GUI File Explorer, but if you are logged into a machine on the other side of the world through a terminal window (i.e. you DON'T have GUI access), this can be a very powerful way to do things that you take for granted in your desktop environment.

4.3.6 Some Emacs Fun

People with too much time on their hands have written some pretty silly Emacs modes.

Open Emacs and try these (remember **C-x k** will kill the buffer):

M-x tetris (do you guys know this ancient game?)

M-x doctor

Read: Conway's Game of Life. Then do

M-x life

This has given you a little taste of what Emacs can do.

4.4 Homework

Homework 4 is here

CHAPTER

FIVE

SCIENTIFIC VISUALIZATION

5.1 Scientific Visualization

When doing science, you will often need to "*look at some data*".

In fact, I almost always have a quick "*Exploratory*" look at whatever data I have for a project or experiment, so that I can make sure that everything is making sense. If the temperature of your sample should go up as you increase the magnetic field, then a quick look at your temperature v. magnetic field data will tell you if you have a loose wire, before you spend a lot of time doing a complete experimental run.

Usually, you will be comparing recorded data with a theoretical *prediction* from a *model* in mathematical form.

For this you will need software to do scientific visualization. You are certainly already familiar with the simplest form of data visualization, i.e. plotting or graphing x and y values on a Cartesian grid.

Let's examine this idea of scientific visualization. Here is a definition taken from a tutorial on the website of the Georgia Tech Scientific Visualization Laboratory an entire institute set up to help scientists and engineers with visualization at *GT*.

```
*Scientific visualization, sometimes referred to in shorthand as*
**SciVis**, *is the representation of data graphically as a means of
gaining understanding and insight into the data. It is sometimes
referred to as visual data analysis. This allows the researcher to
gain insight into the system that is studied in ways previously
impossible.*

*It is important to differentiate between scientific visualization and
presentation graphics. Presentation graphics is primarily concerned
with the communication of information and results in ways that are
easily understood. In scientific visualization, we seek to understand
the data. However, often the two methods are intertwined.
Thus scientific visualization is a means of using graphical techniques
to allow the incredibly powerful, massively parallel processing, high
performance, neuro-optical wetware (your eyes and your brain), to look
for patterns in the data.*
```

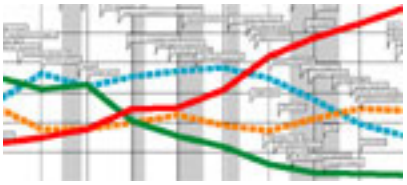
I would also argue that once you have understood your data, the very next step is to present your results to others! To do that, you take the graphics of your visualization studies, clean them up, add helpful guides and text, and make presentation graphics for publication or oral presentations. So, the two are very closely related.

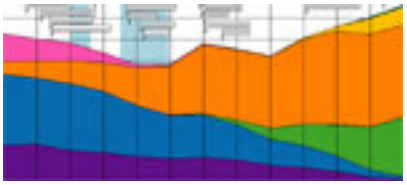
Shortly, we will get a tool which will serve us for both aspects.

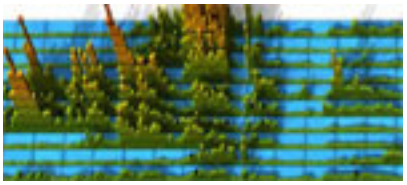
5.1.1 Some Examples

First however, follow the link below to look at a few very impressive visualizations of some interesting data from the Wall Street Journal's Classroom page.

Look at these graphs







prepared by Karl Hartig, for the from the Wall Street Journal Classroom Edition.

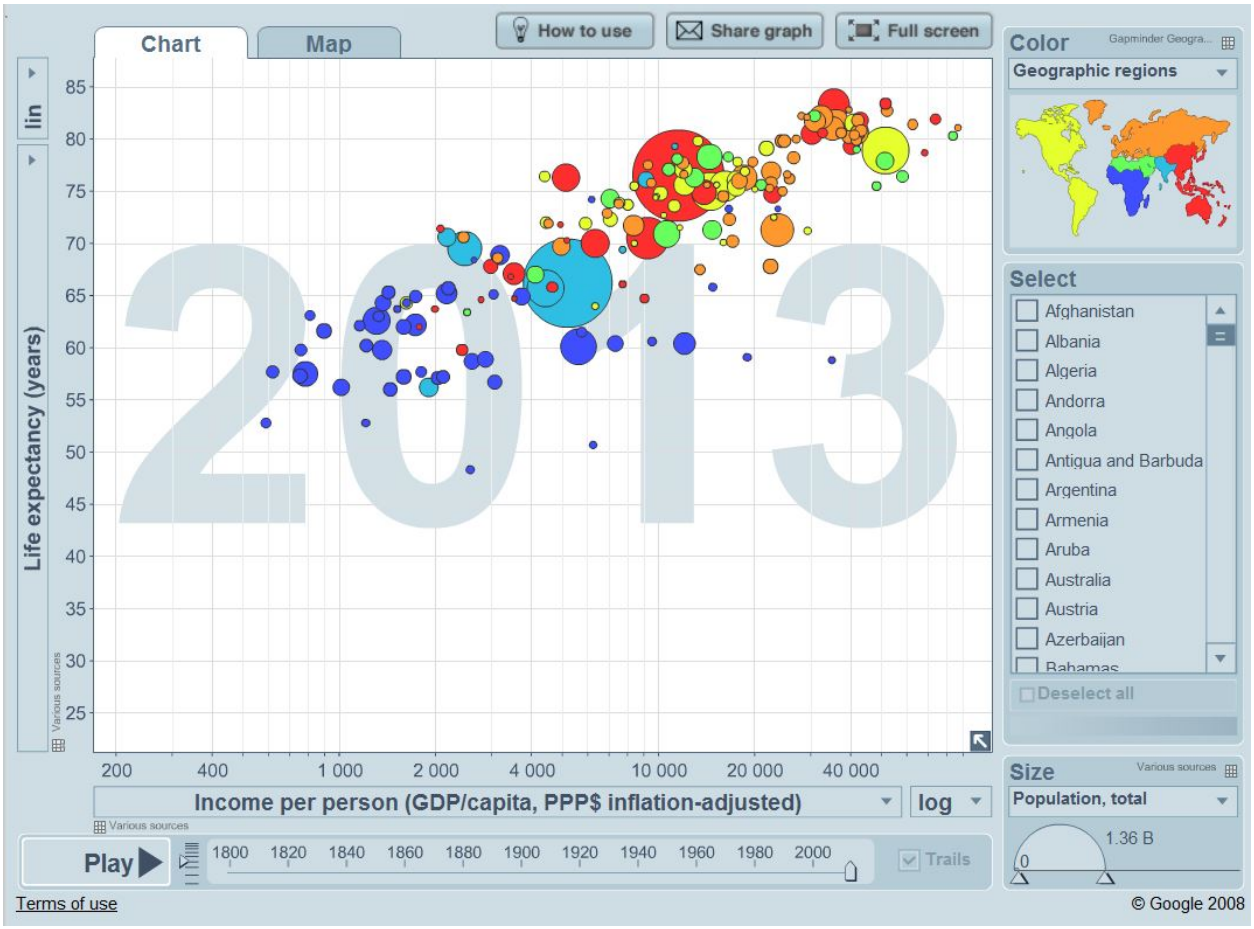
These are best viewed in their PDF versions (because you can zoom in deeply), reached by clicking on the PDF icon below the image on each page.

In particular, be sure to look at

- The Computer Power chart. This one documents the exponential rise in the processing power of computers in the last decade (the so called, Moore's Law). However, I find many of these presentations very thought provoking.
- The 3-D US Population graph shows the age people living in the last century, and how surges, like the Baby Boomers born after WWII, move forward in time so that the overall age distribution of the population seems to have "ridges" along diagonal lines.
- The 3-D plot of imigration shows that the current imigration wave is not unlike the one which occured in the 1800's.
- Energy Production/Flow is quite important as well. Understanding how to make this more efficient is going to become an essential part of 21st century engineering-physics.

One last example. Have a look at www.gapminder.org/world.

It plots *average Life Expectancy* versus *average Income per Capita*, for each country, for each year since 1800. In addition the *Population* of each country is also shown by the size of the marker. You can run the graph as a movie, letting time (the fourth dimension) run from 1800 to 2013.



Run the movie, by clicking **Play** .

Notice the worldwide downward vertical bumps in Life Expectancy in many countries during *World War I* from 1914-1918, and from *World War II* from 1939-1945.

This *data visualization* tells an incredible story. The tool that presents it has been well crafted, allowing you to choose several different quantities to plot so that you can explore many relationships.

Data visualization is a fundamental aspect of doing science.

5.1.2 SciVis Apps

There are many software applications for doing SciVis. A non-authoritative list is here. **Have a look**, and scroll to the bottom of the page, to the *See Also* section.

Many of these are very powerful, and many are very expensive. Your computer may already have a software program to graph data (Excel, for example, and other spreadsheets used in business computing will make simple graphs of data).

Also, you may have had experience with other scientific analysis programs, such as *Matlab*, *Maple*, and others. These are powerful and very good applications.

If you have experience with these, it will come in useful, since the more applications you are familiar with, the more versitile you will be as a scientist or engineer. Furthermore, your career is likely to involve several employers as you begin to build expertise in you area. Different employers will have different resources; Lawrence Livermore National Laboratory, will have many different software packages, and your boss may not blink if you ask to buy Matlab at $10,000 because that is what you are familiar with.

My philosophy in this course is to, as often as possible, provide you with *free* tools, which you can "*take with you*" to your next position, and the one after that.

Linux provides a platform for many of these tools, and *Gnuplot* is one of the easiest to use.

Another popular choice is Matplotlib, which is part of the Python programming language. In a follow-up course to this one, I plan to include a couple chapters on *Matplotlib* and other Python Scientific Visualization tools like Mayavi. For now, though, we will use **'Gnuplot >http://matplotlib.org>'_**.

5.1.3 Gnuplot

Note: "*I can't believe this is free!!!*"——`Ben Rosenkrantz, B.S. Physics 2006`

That pretty much sums it up. *Gnuplot* is an amazingly powerful plotting and data fitting program, capable of making 2 and 3-D plots such as y = f(x), and z = f(x,y). I use it for most of the data visualization in my research.

Getting Started with Gnuplot

First, let's keep our gnuplot exercises in a separate directory.

Open an shell terminal, and make a subdirectory below your home dir called:

gp (you can give it a different name if you like).

Now **cd** to **gp** .

We'll save *Gnuplot* files and data here.

We'll use *Gnuplot* to display three primary types of plots, often on the same graph for comparison. These are:

- *Builtin Mathematical Functions*
- *Numerical Data from a File*
- *User Defined Functions*

We will also do

- *data modelling and statistics*
- *publication/presentation graphics*

Now at the shell prompt, type

gnuplot

This will start the *gnuplot* program and you should see something like this on your screen:

```
     G N U P L O T
     Version 4.6 patchlevel 3    last modified 2013-04-12
     Build System: Linux x86_64

     Copyright (C) 1986-1993, 1998, 2004, 2007-2013
     Thomas Williams, Colin Kelley and many others

     gnuplot home:     http://www.gnuplot.info
     faq, bugs, etc:   type "help FAQ"
     immediate help:   type "help"  (plot window: hit 'h')

Terminal type set to 'wxt'
gnuplot>
```

This last line:

```
gnuplot>
```

is the *Gnuplot* prompt which is where you will type gnuplot commands.

Step One: Builtin Math Functions

In the examples below, be sure to type things exactly as they appear.

Any quotes (**“**) and commas (**,**), etc. are important!

At the **gnuplot>** prompt, type

```
plot x
```

and hit **Enter**.

You should see a graph window with a diagonal red line. This is a plot of the function **f(x) = x**.

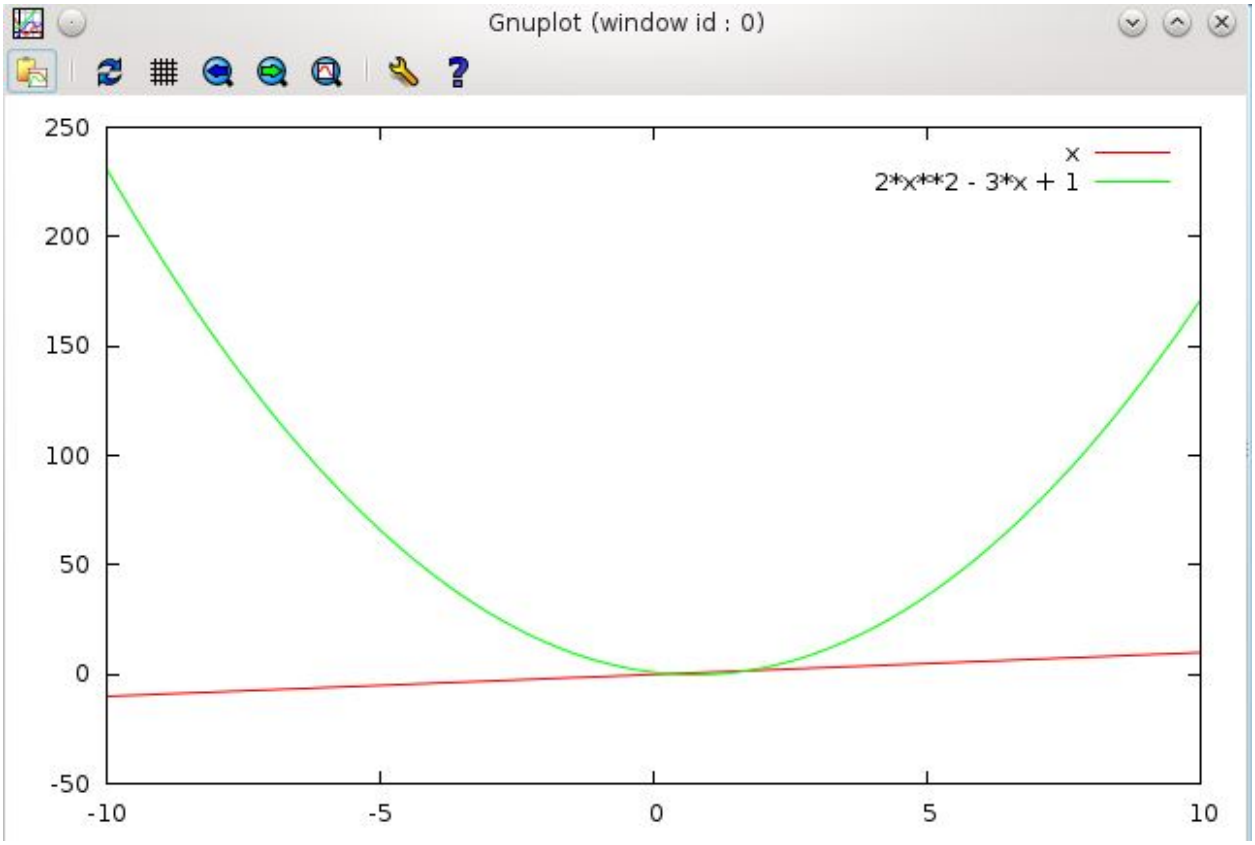
Gnuplot's default plotting range for x is [-10:10] (the range between x = -10 and x = +10).

You just plotted your first function.

Next, let's add a quadratic function: $f(x) = 2x^2 - 3x + 1$

To do this, hit the **UP** arrow to bring back the previously entered command and now add the following, then press **Enter**.

```
plot x, 2*x**2 - 3*x + 1
```



Does your plot look the same? (There should be a line and a parabola).

Notice how we entered

$2x^2 - 3x + 1$

as

```
2*x**2 - 3*x + 1.
```

We must explicitly tell gnuplot about multiplication with the `*` operator.

For the quadratic term, x^2 , we use the notation `x**2`, where the "*double star*" notation: `**` represents exponentiation.

For example, we could do `x**0.5` for $\sqrt(x)$ (the square root of *x*), however gnuplot has a builtin function: `sqrt(x)` for this.

Also, we have two functions plotted, `f(x) = x`, a linear function $f(x) = x$, and $f(x) = 2x^2 - 3x + 1$, a quadratic function.

They are both plotted on the same graph with one `plot` command, separating the two functions `x` and `2*x**2 - 3*x + 1` with a comma **,**

```
plot x, 2*x**2 - 3*x + 1
```

We can plot several more functions on this same graph, we simply put commas between each function to be plotted.

Note: When plotting 2-dimensional plots, the *independent* variable is *(almost) always x*.

I say "*almost*" because we can do *parametric plots* (a more advanced concept) where the independent variable is *t*.

But until we get there, remember: Gnuplot functions are always functions of *x*. `sin(x/pi)`, `exp(-x**2/4)`, `x**3 - x`, etc.

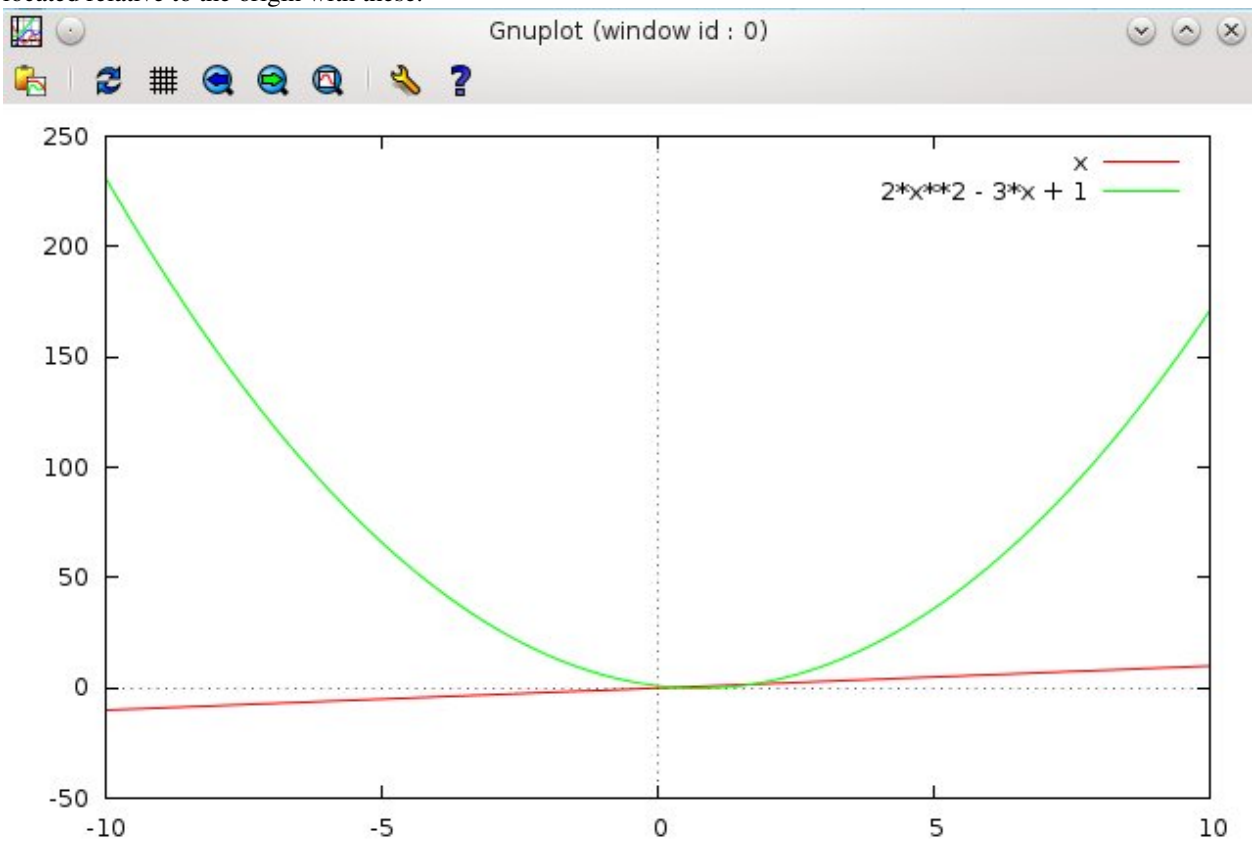
Step 2: Plot Range

Let's make the plot a bit easier to read. At the prompt, type:

```
set zeroaxis
```

Then use the **UP** arrow to recall your last `plot` command. Press **Enter**.

Now you should have dotted lines along the x = 0 and y = 0 axes. It's a little easier to see where your functions are located relative to the origin with these.



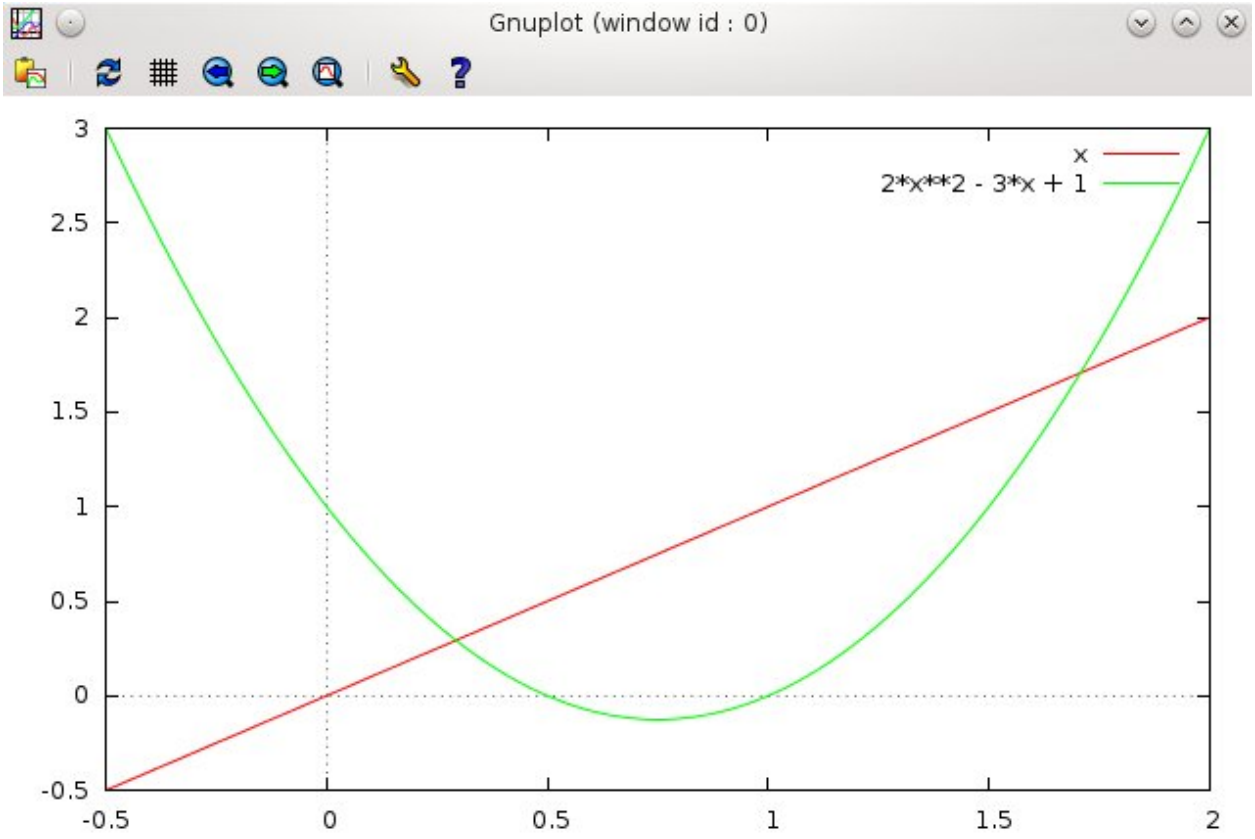
Now, let's zoom in a bit. The interesting area of the graph is where the two functions intersect. Let's narrow the plot range.

Using the **UP** arrow, recall your previous `plot` command.

Now use the **LEFT** arrow to move the cursor to just after the **plot** command, and insert a range specifier **[-0.5:2]** as shown below:

```
plot [-0.5:2] x, 2*x**2 - 3*x + 1
```

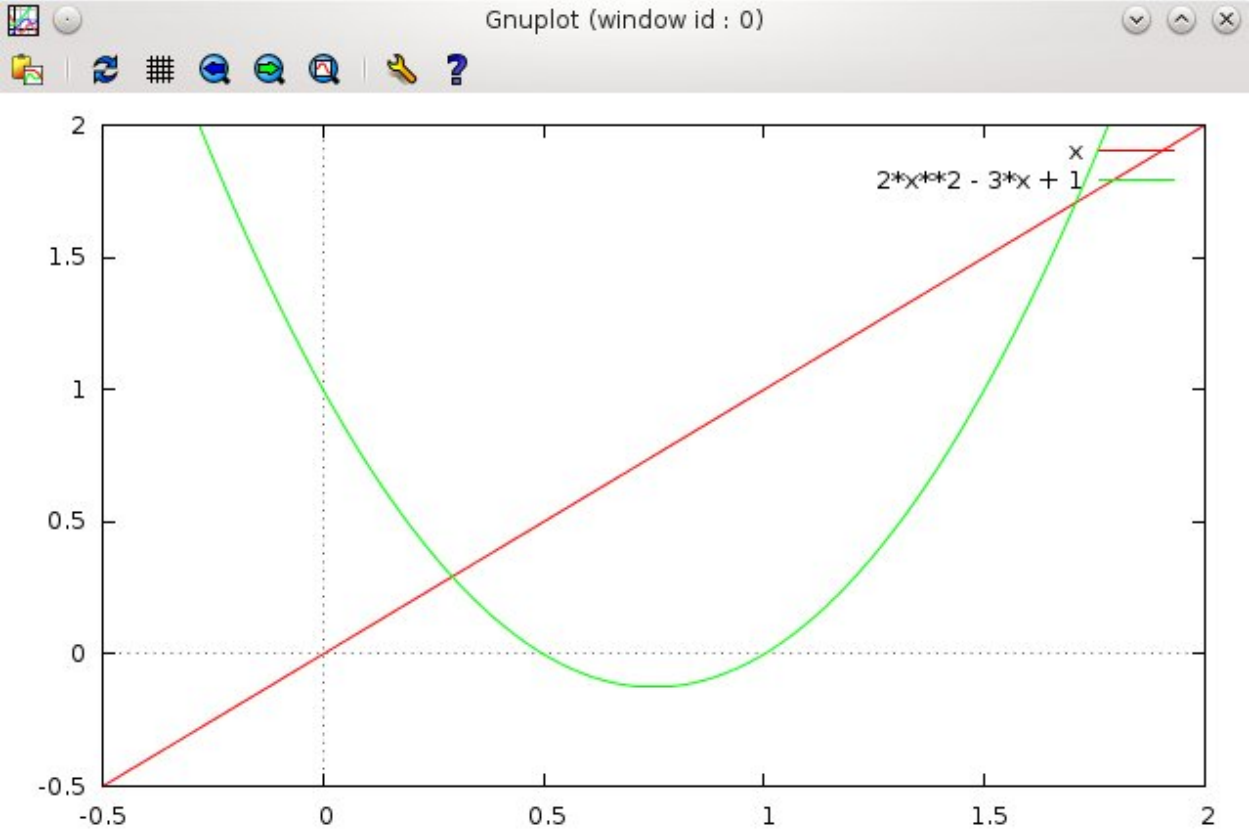
Notice what this does (press **Enter** to execute the `plot` command if you haven't already). The graph should be restricted to the area between x >= -0.5 and x <= 2.0 .



You can also limit the y range by adding a second range specifier for the y axis.

```
plot [-0.5:2][-0.5:2] x, 2*x**2 - 3*x + 1
```

which also limits the y range of the plot to lie between particular values.



The format for the range is

`[Xmin :Xmax][Ymin :Ymax]`

One can allow the x range to be free while specifying the y range with

`[:][Ymin :Ymax]`

and similarly

`[Xmin :Xmax][:]` to leave the y range free.

This however is equivalent to simply `[Xmin :Xmax]` by itself

Notice that even though the *x*-range and *y*-range are the same `[-0.5:2]`, the plot is still rectangular. This is the default for this system.

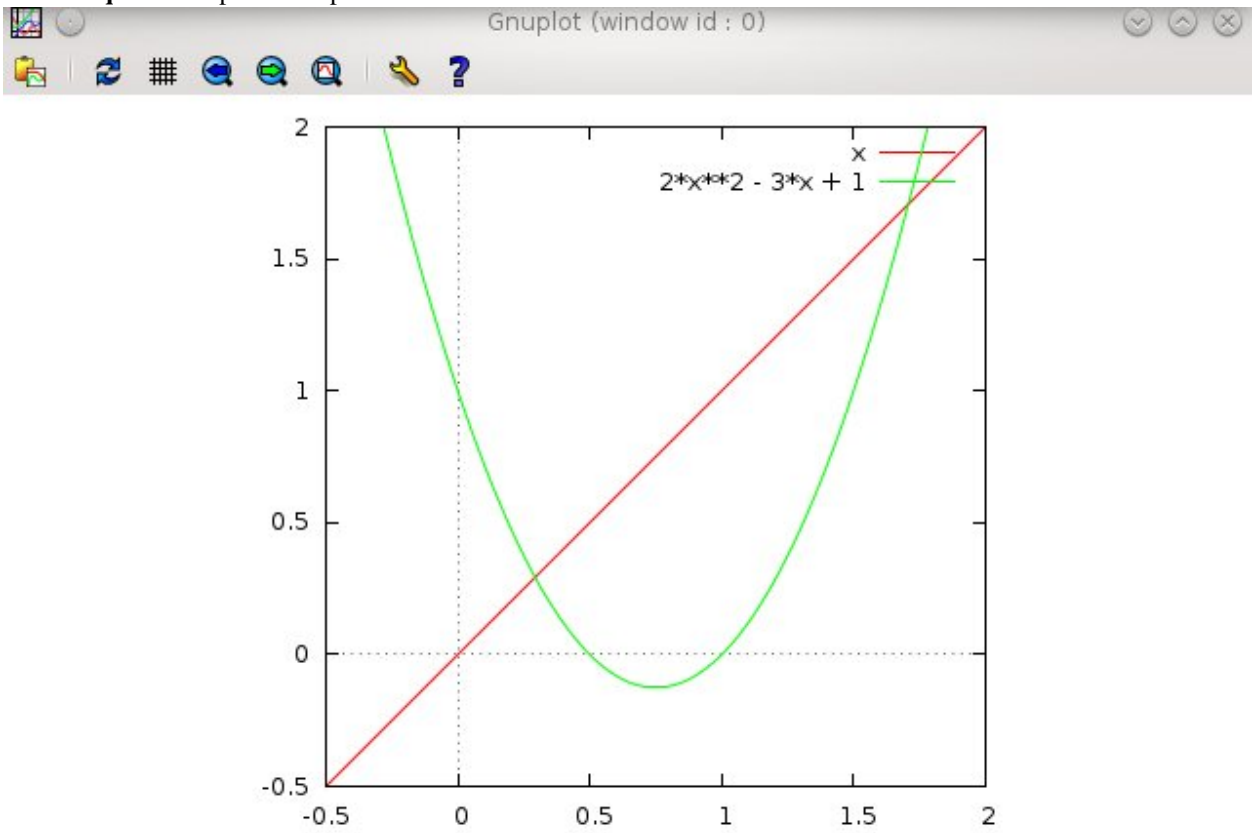
In order to get a *Square* plot, you type this:

set size square (RETURN)

then type **replot** (RETURN).

Replot always reproduces the last plot—it's the same as using the UP arrow key to recall the last plot command typed, and hitting RETURN.

set size square will produce a plot that looks like this.



To go back to your original *aspect ratio*, type:

set nosquare

then **RETURN**, then **replot** (then RETURN).

Step 3: Label the Graph

Still the graph needs more. Let's notice some things.

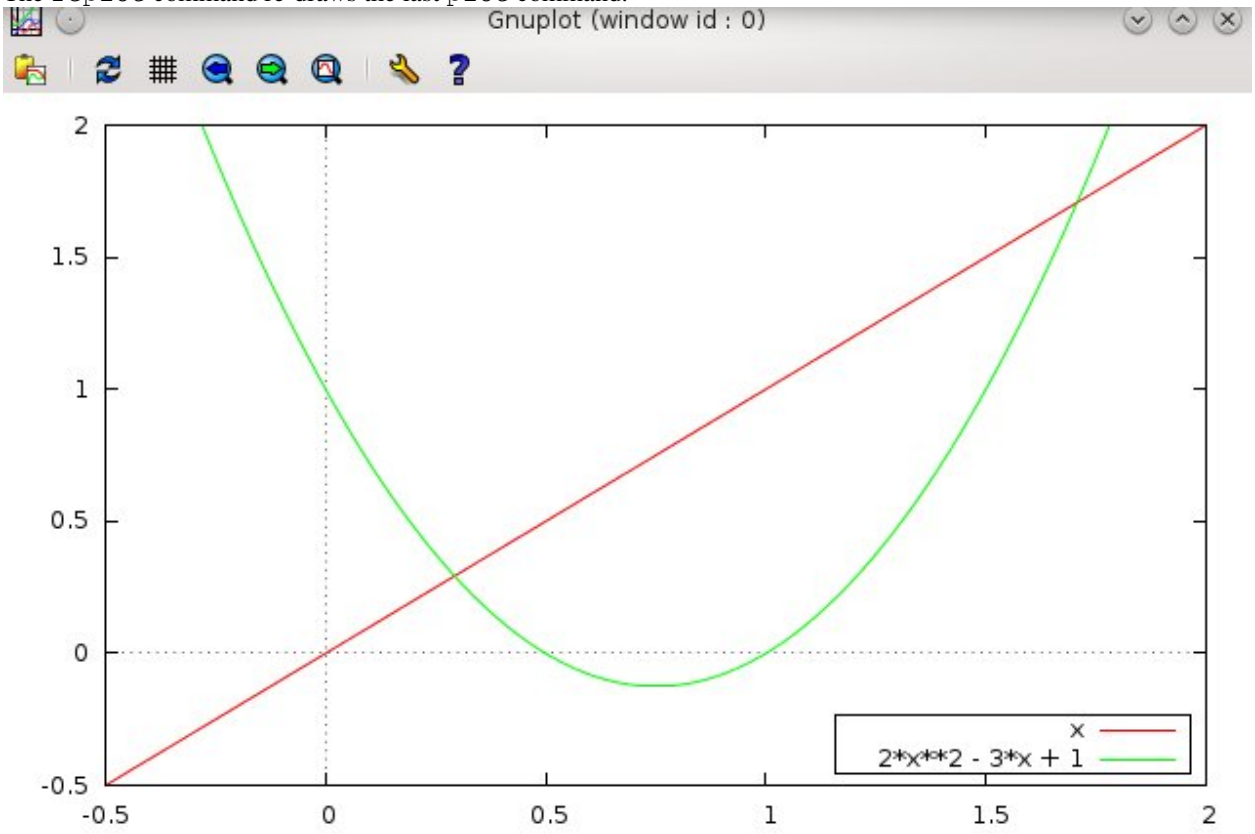
There is a *Key* telling us which plotted lines correspond to which function.

The default Key is to draw short examples of the lines used and copy the function that you typed to plot beside the associated line.

Notice that the key is run over by the plot lines. You can move the key to another place by typing

```
set key bottom right box
replot
```

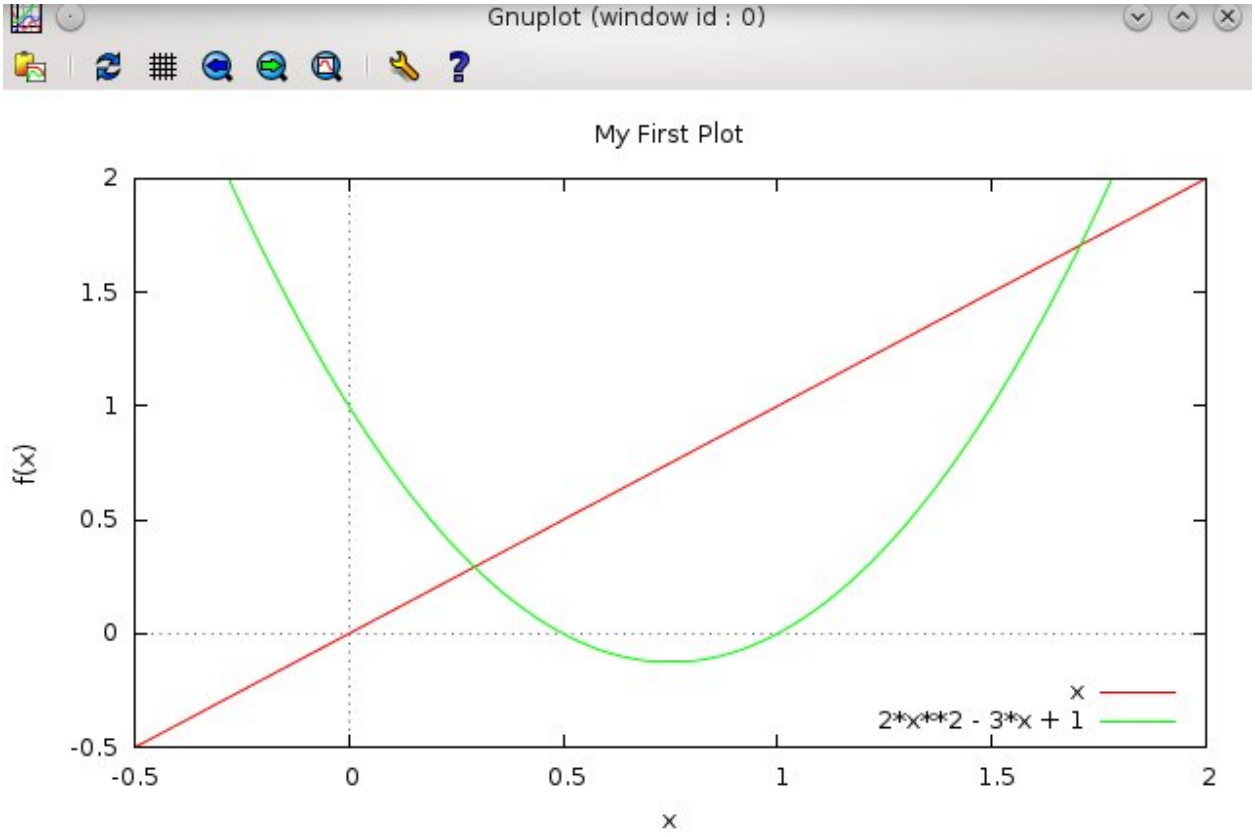
The `replot` command re-draws the last `plot` command.



Notice where the *Key* is now, in a more conveniently viewable location, and it has a nice box around it.

You can get rid of the box if you don't like it by typing

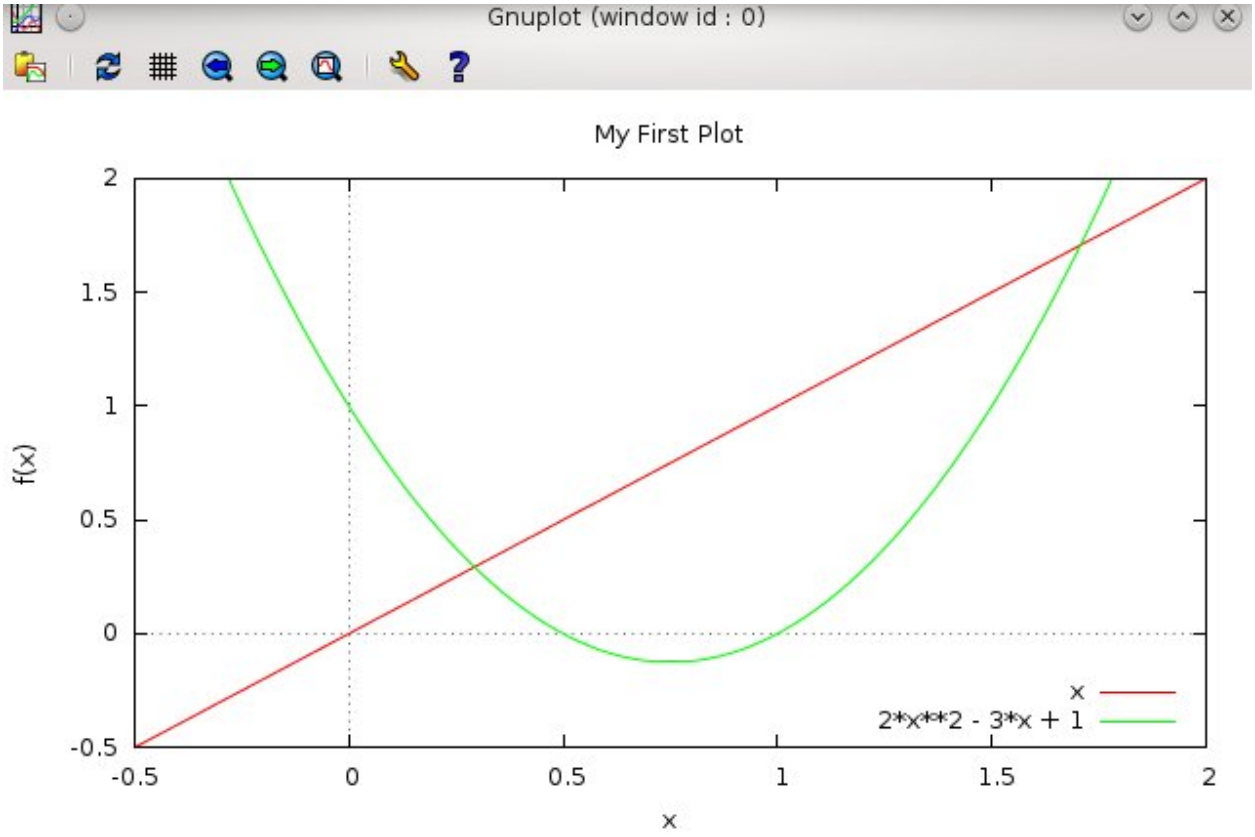
```
set key nobox
replot
```



A graph also needs labels! If your high school teacher did not hammer this into you, let me do so. We can add these with

```
set xlabel "x"
set ylabel "f(x)"
set title "My First Plot"
replot
```

This should label the x and y axes, and put a title above the the graph, as shown above. Verify that it does so.



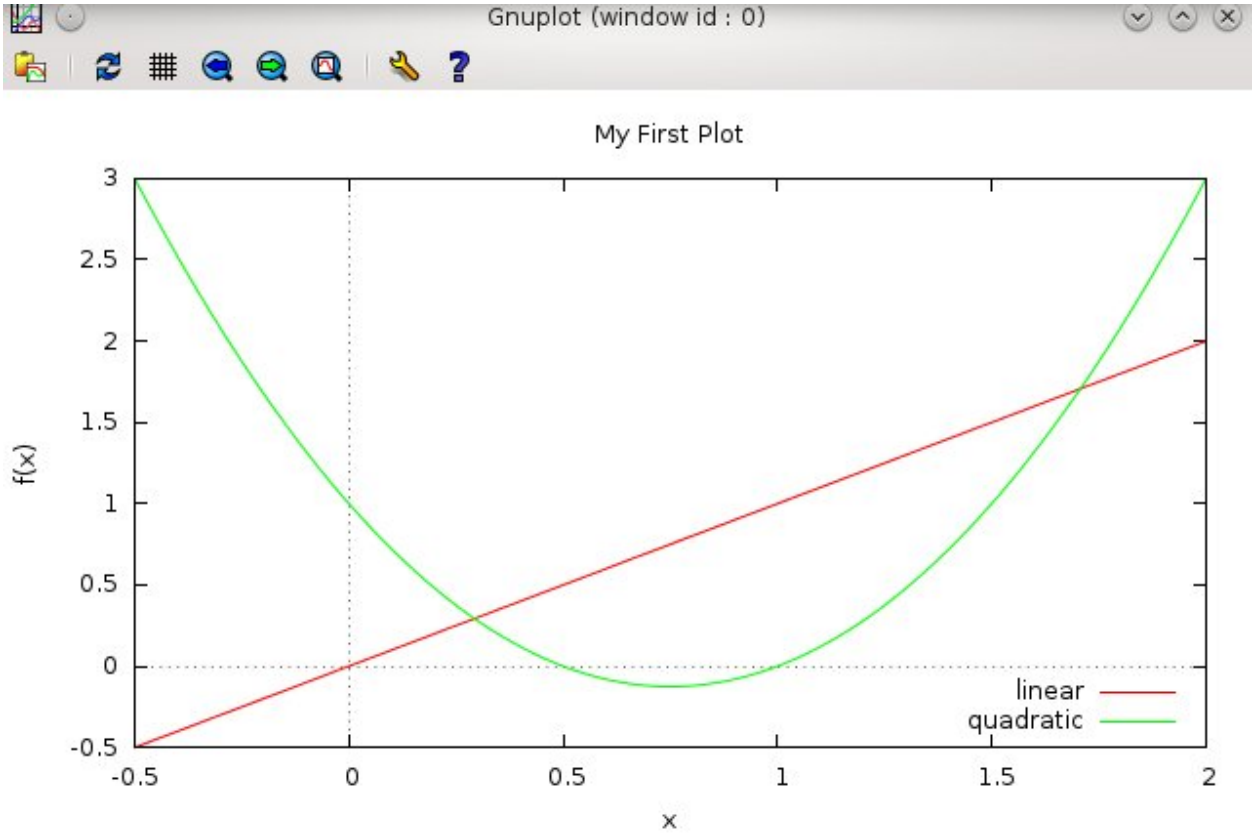
Also, notice that the words you want to appear in the labels must be enclosed in **“”**s.

Words such as these in labels, etc. are called *strings* (short for *a string of characters*), and all “*strings*” must be enclosed in **“”**s. You can even clean up the Key a bit, by giving labels to the lines. Look at the Key as it is now in your plot.

Call back your previous plot command with the **UP** arrow, and add the following **title** option to each function

```
plot [-0.5:2][-0.5:3] x title "linear", 2*x**2 - 3*x + 1 title "quadratic"
```

Do you see what that did to the Key?



You can learn more about the possible settings for the Key by typing

```
help set key
```

This will display Help contents on the options and usage of the `set key` command.

```
gnuplot> help set key
 The `set key` command enables a key (or legend) describing plots on a plot.

 The contents of the key, i.e., the names given to each plotted data set and
 function and samples of the lines and/or symbols used to represent them, are
 determined by the `title` and `with` options of the {`s`}`plot` command.
 Please see `plot title` and `plot with` for more information.

 Syntax:
       set key {on|off} {default}
               {{inside | outside} | {lmargin | rmargin | tmargin | bmargin}
                 | {at <position>}}
               {vertical | horizontal} {Left | Right}
               {{no}opaque}
               {{no}reverse} {{no}invert}
               {samplen <sample_length>} {spacing <vertical_spacing>}
               {width <width_increment>}
               {height <height_increment>}
               {{no}autotitle {columnheader}}
```

```
                {title "<text>"} {{no}enhanced}
                {font "<face>,<size>"} {textcolor <colorspec>}
                {{no}box { {linestyle | ls <line_style>}
Press return for more:
```

Keep hitting the **RETURN** until you get the `gnuplot>` prompt back.

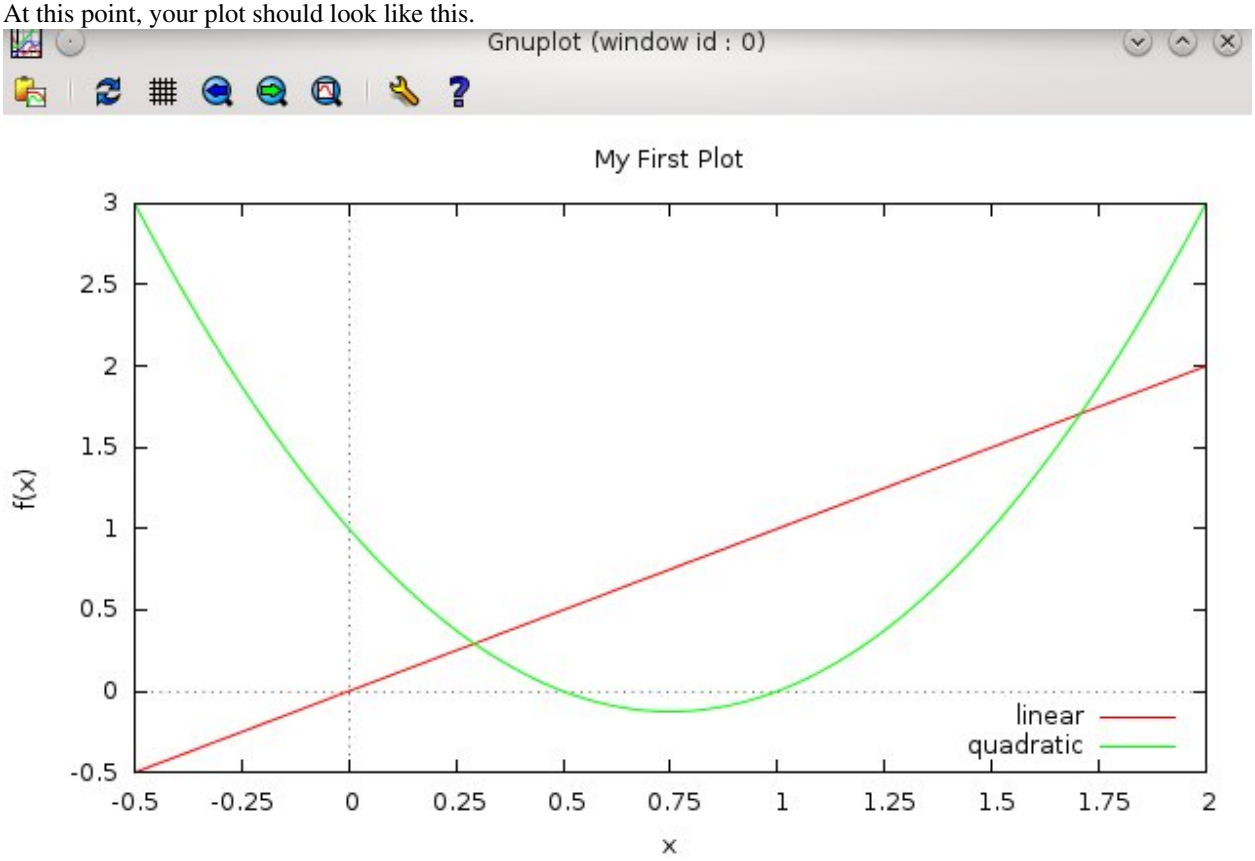
One more thing. Suppose we want a bit finer resolution on the x axis than small tics in steps of 0.5.

Suppose we wanted them every 0.25 instead. We can accomplish this by doing

```
set xtics 0.25
```

How do you think you would adjust the y tics?

At this point, your plot should look like this.



Summary

Let's summarize what we've done so far. If you need to, go back though the above exercise and make sure you know how to do each of these things.

- Gnuplot plots functions of **x**. In other words, our functions are

made from polynomials or builtin functions in which an **x** must appear. To see what I mean: Try **plot y**2** , then try **plot x**2**

- We plotted two different functions on the same graph . We could

easily add more by separating each function by a comma.

- We learned the simplest mathematical operators: ***** for multiply,

and the exponent operator ******, as well as **+** and **-**. Division is done with the forward slash: **/** as in **1/x**2** for

- We learned how to adjust the range of the plot, and add a zeroaxis .
- We learned how to move the Key and change its labels.
- We learned a little about the **help** command.
- We learned how to set labels for axes and give a title to the graph.
- We changed the resolution of the axes tics with **set xtics 0.25** (we could set the y resolution with **set ytics 0.1** for example).

5.2 More fun with Gnuplot

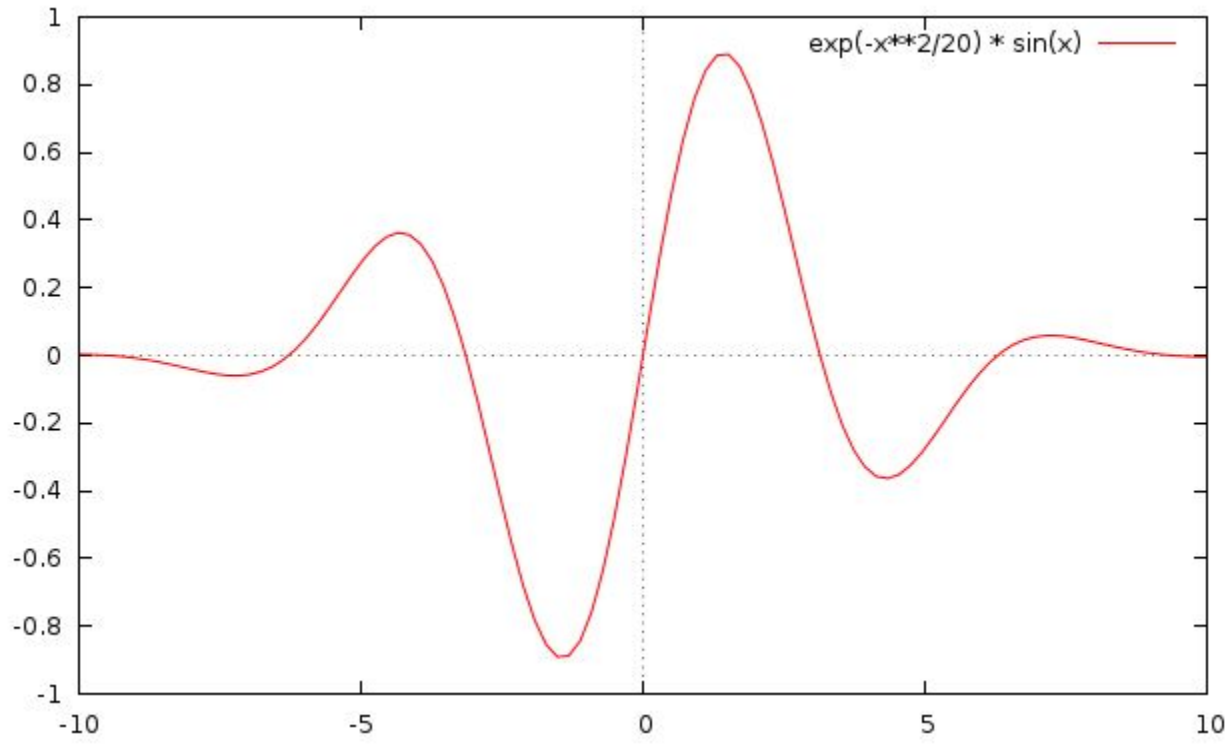
Let's use gnuplot to plot some more mathematical functions. Certainly we'll need more than just polynomials in x as we used on the last page.

Gnuplot has many builtin math function which you can plot. Try the following plot.

If gnuplot is running on your computer, end it by typing quit at the prompt. Then start it again by typing **gnuplot**. This is just to clear out the previous settings, and to show you how to get out when you are done.

Now that it's running again, type:

```
gnuplot> plot exp(-x**2/20) * sin(x)
```



You may want to turn on the zero axis (remember how?). Later we'll learn how to set that as the default behavior at startup. The plot you just made is of the function

where you used the common *exp()* and *sin()* functions. A complete list of functions is here:

- Gnuplot Builtin Math Functions

Please have a look at these.

For example, gnuplot will plot the two lowest *Bessel functions*, *J0(x)* and *J1(x)*. From these two, the higher order Bessel functions can be constructed. Bessel functions are used in the solution of certain differential equations, particularly in cylindrical coordinates, and turn up often in the physics of musical instruments.

Gnuplot also knows about imaginary numbers and certain constants like π.

Try this:

Start gnuplot, then type the following:

```
gnuplot> print pi
gnuplot> w = 5.8
gnuplot> v = 3.0
gnuplot> print w/v
gnuplot> a = {2.0, 3.0}
gnuplot> b = {1.0, -2.0}
```

In the first line, you asked gnuplot to print the value of " **pi**".

You can also use **pi** as a value in a range limit, as in **plot [-pi:pi]**.

In the last two lines you defined two variables, a and b , as complex numbers. The first number in the bracket is the **real** part and the second of course, is the **imaginary** part.

What is the product *ab* ? You should remember from junior high:

$$\begin{aligned} ab &= (a_{\text{real}} + ia_{\text{imag}})(b_{\text{real}} + ib_{\text{imag}}) \\ &= a_{\text{real}}b_{\text{real}} - a_{\text{imag}}b_{\text{imag}} + i(a_{\text{real}}b_{\text{imag}} + a_{\text{imag}}b_{\text{real}}) \\ &= 2 * 1 - 3 * (-2) + i * (2 * (-2) + 3 * 1) \\ &= 8 - i \end{aligned}$$

Now in gnuplot, do:

*print a*b*

Does gnuplot confirm your complex arithmetic?

Here $i = \sqrt{-1}$.

Note: For some incomprehensible reason, engineers tend to use the letter *j* for the square root of -1. While this is completely ridiculous, I point it out so that you will be aware of the pitfall. Physicists tend to use *i*. If you are a bit rusty on complex numbers and how to manipulate them, here are a couple links to brush up.

- www.sosmath.com/complex/number/basic
- www.mathcentre.ac.uk/.../all_subjects/complexnumbers/...

5.2.1 A Simple Calculator

Since gnuplot has many builtin math functions, will store things in variables, and will do calculations and print the result, you can use it as a simple scientific calculator. Often, when I'm working on my computer and don't want to grab my calculator (which I actually don't enjoy using that much), I just fire up gnuplot and do a little calculation there. Here's an example: Suppose I want to compute the following quantity

when *r = 2.5* and *R = 3.9*

You could simply start gnuplot and enter the following (Please perform the steps below:

```
gnuplot> r = 2.5
gnuplot> R = 3.9
gnuplot> result = 4*pi**2 * r**2/sqrt(R**2 - r**2)
gnuplot> print result
82.4300852943585
gnuplot>
```

Much easier! . Really.

Try it–have your mom type in the above while you compute the same thing on your caculator.

I bet your mom wins.

5.2.2 Solving Equations Graphically

If it's not already running, start gnuplot.

Plot the two equations you used on the previous webpage, [Step One: Build in Math Functions]

$$y(x) = x$$
$$y(x) = 2x^2 - 3x + 1$$

Set the x range to **[-0.5:2]** [Step Two: Plot Range.]

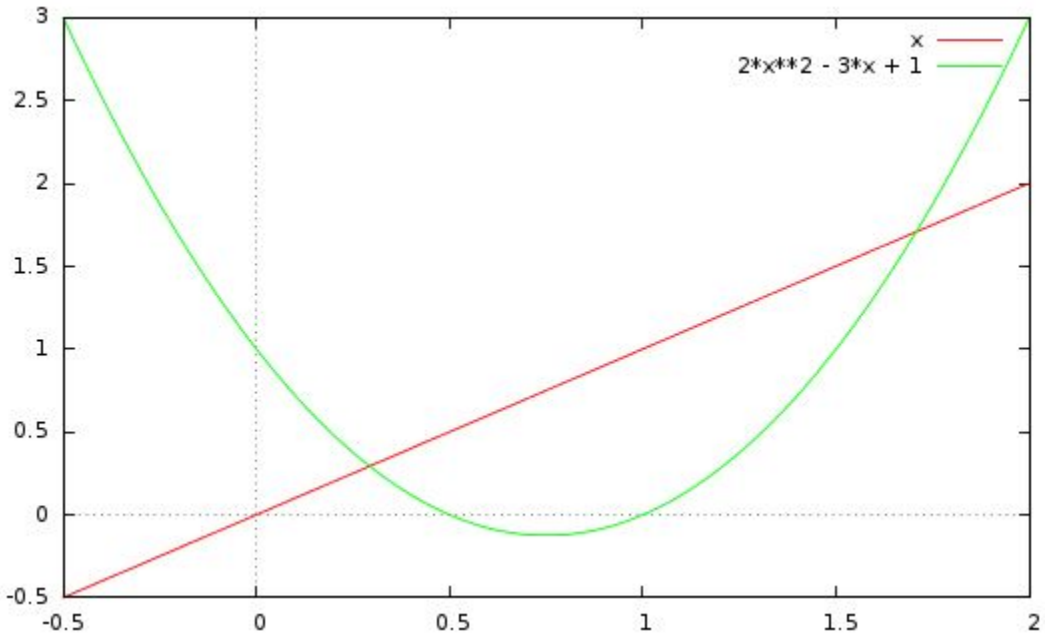
Turn on the *zeroaxis* [also in section: Step Two: Plot Range].

Notice that $2x^2 - 3x + 1$ factors into $2(x - 0.5)(x - 1)$.

Note: Verify this if you don't see it immediately (i.e. write out the second expression and show that it's equal to the first).

Thus the exression $2x^2 - 3x + 1$ has *roots* at $x = 0.5$ and $x = 1$. Remember, a *root* is the *x* value, x_{root}, where the $y(x_{\text{root}}) = 0$

Is this corroborated by your plot? In other words, you can see on your plot the places where the equation $2x^2 - 3x + 1 = 0$ is satisfied, and do you get the values: *0.5* and *1.0*.



Now let's do something a little more interesting.

The *x* values where the line intersects the parabola are given by the solutions to the equation:

$$y(x)_{\text{parabola}} = y(x)_{\text{line}}$$
$$2x^2 - 3x + 1 = x$$

This equation is slightly more non-trivial to solve; you have to use (*OMG!*) **The Quadratic Formula**, and the solutions are

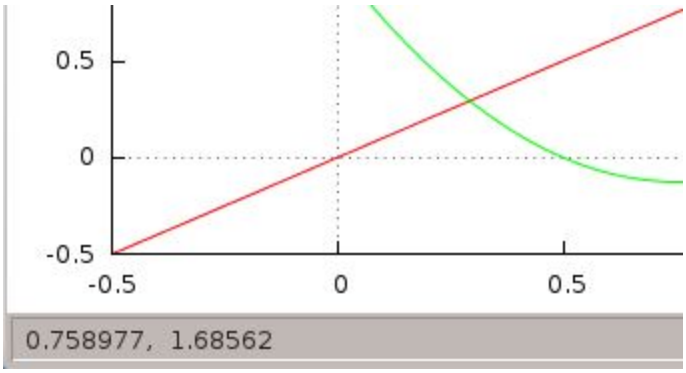
$$x = \pm\frac{\sqrt{2}}{2}$$

Note: Verify this.

Numerically , these solutions are at *x = 0.29289* and *x = 1.70711*,

Does this look correct from your graph?

You can get a rough anwser by using the mouse. As you move the mouse cursor around in the plot area, notice that the **x, y** values of where the mouse cursor is, are printed in the bottom left corner of the gnuplot window.

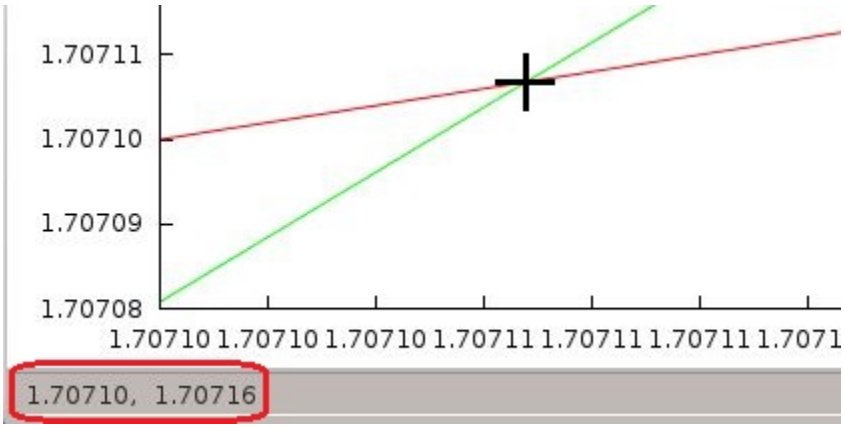


This can give you a pretty good idea of the value of something in your graph.

Here's another method to solve the equation above, instead of the quadratic formula.

We can zoom in and find where exactly the intersection lies. If you successively narrow the plot range, you can "zero in" in the intersection point. Do the following (use the UP arrow to recall each previous line):

```
gnuplot> plot [1.5:2] x, 2*x**2 - 3*x + 1
gnuplot> plot [1.7:1.71] x, 2*x**2 - 3*x + 1
gnuplot> plot [1.706:1.708] x, 2*x**2 - 3*x + 1
gnuplot> plot [1.707:1.7072] x, 2*x**2 - 3*x + 1
gnuplot> plot [1.7071:1.70712] x, 2*x**2 - 3*x + 1
```



Notice how the successive narrowing of the x range allows us to find the x value of the intersection of the two equations (represented with red and green plots) by reading it from the x axis.

Try the same thing by narrowing the x range around the other root near $x = 0.293$

This probably seems pretty simple, and not very useful since we already know the solution of the above equation from the quadratic formula.

What about a situation where you don't know the solution of the equation by analytic (like the quadratic formula) methods? In this case the only way to get the result is to use **numerical methods**, whihc means that we use a computer to find an approximate (and hopefully very accurate) solution.

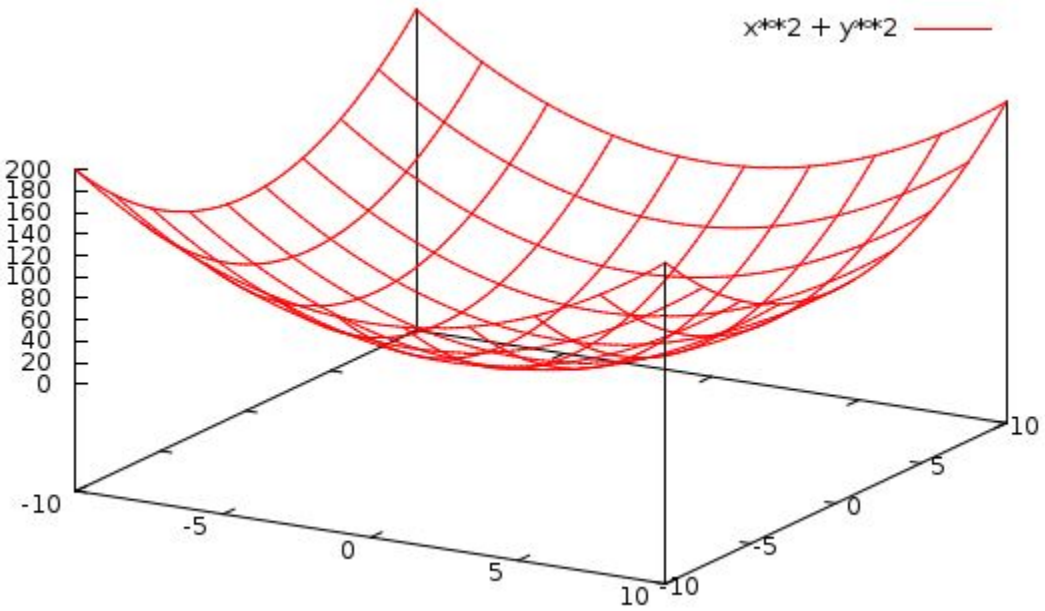
5.3 3-D Plots with Gnuplot

You can plot many interesting mathematical functions *f(x)* with gnuplot such as you have been doing in the previous exercises.

Another powerful feature of gnuplot is the ability to plot functions of two variables as a *surface z = f(x,y)*. There are some examples of what gnuplot can do at the top of this page.

To see what I mean, start gnuplot and type the following at the prompt:

```
gnuplot> splot x**2 + y**2
```



This will produce a simple rectangular parabolic "bowl" shaped surface (hence the command s plot), which represents the function $z = x^2 + y^2$.

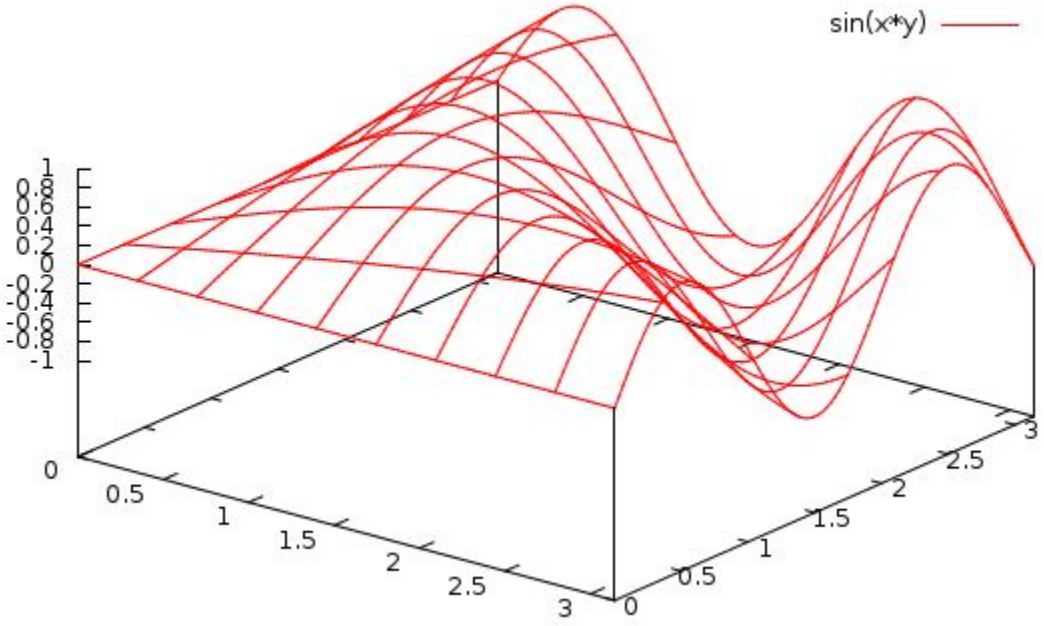
Here we used **splot** (surface plot) instead of the previous **plot** command which is reserved for 2-d f(x) plots.

You should be able to "*grab*" the splot with your mouse , click and hold down the right mouse button, and move the mouse.

This will rotate the graph so that you can see it from different angles, allowing you to explore the 3-dimensionality of the surface.

Try another plot:

```
gnuplot> splot [0:pi][0:pi] sin(x*y)
```



You can see the interesting shape (can you grab it and look at it from different angles?).

Notice the plot ranges *[0:pi]* for both *x* and *y* . (Remember that gnuplot recognizes the word "*pi*")

If you wanted to fix the plot ranges, so that you didn't have to keep typing them, you could type:

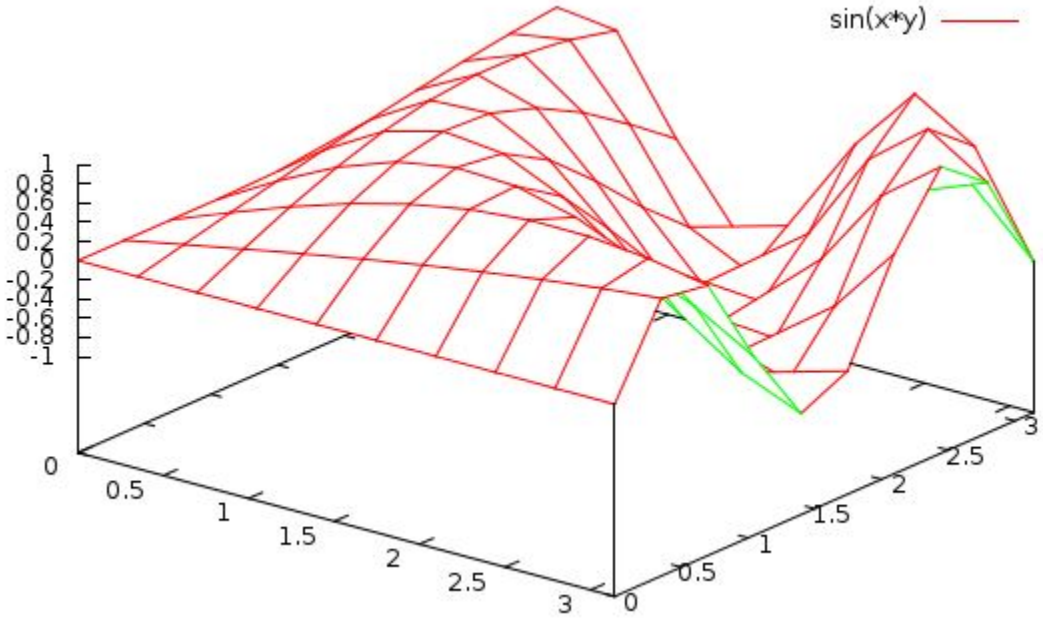
```
gnuplot> set xrange [0:pi]
gnuplot> set yrange [0:pi]
then do:
gnuplot> splot sin(x*y)
```

This will set the default ranges to **[0:pi]** (of course you could set any ranges you want. Sometimes this is a time saver.

Let's try to make the plot better.

First, let's make the surface non-transparent. Type this:

```
gnuplot> set hidden
gnuplot> replot
```

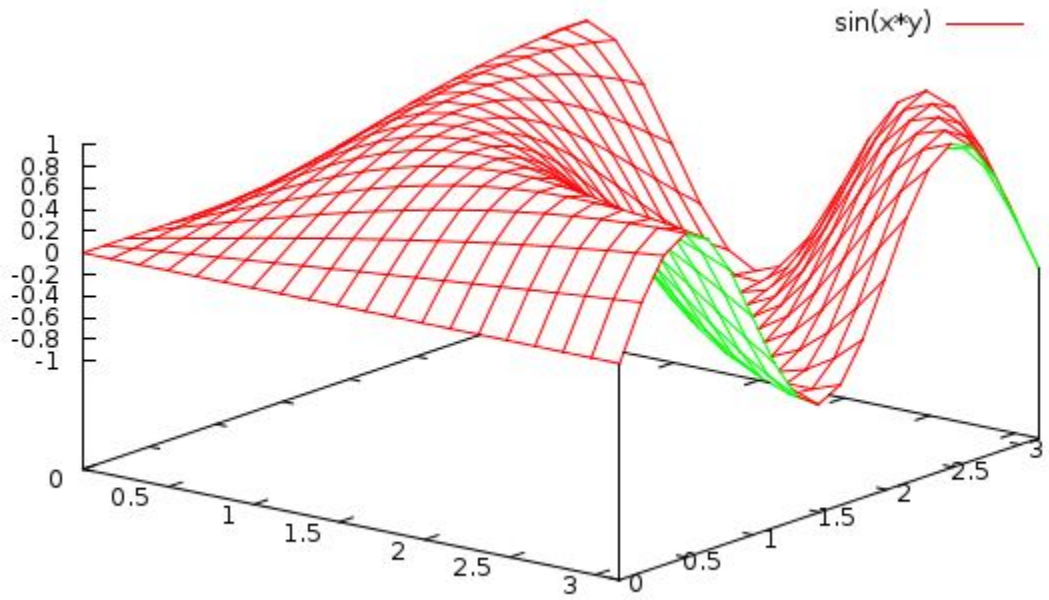


Now the surface is opaque, and the underside is a different color. Notice too that **replot** recalls the last plot.

Still the surface is a litle clunky–it's made of large plane "panels".

If these were smaller, the surface would be more smooth. Type:

```
gnuplot> set isosamples 20
gnuplot> replot
```



Much better. You could also have done **set isosamples 35** or any other number.

The **isosamples** are the number of "samples" in each direction (x and y).

Note that if you are doing a 2-D plot , you would **set samples** to increase the sampling resolution, not *isosamples*– which are only for 3-D splots.

That is:

- 2-D: **plot**: *set samples n* (default is 100)
- 3-D: **splot**: *set isosamples n* (default is 10 in each direction)

If you ever want to see what value a variable is currently set to, do

```
gnuplot> show isosamples
        iso sampling rate is 20, 20
```

(notice that you could set the isosample in x and y separately. If you give only one number, both x and y are set to the same value.)

It is generally true for any variable, that you can see what it's value is by typing **show var**

where *var* is something with a value like *isosamples* or *xtics*, etc..

You can see *all* the **variables** (and classes of variables) that you are able to set by typing just **show** by itself.

```
gnuplot> show
             ^
         valid set options:  [] = choose one, {} means optional

        'all', 'angles', 'arrow', 'autoscale', 'bars', 'border', 'boxwidth', 'clip',
        'cntrparam', 'colorbox', 'contour', 'datafile', 'decimalsign','dgrid3d',
        'dummy', 'encoding', 'fit', 'fontpath', 'format', 'functions', 'grid',
        'hidden', 'isosamples', 'key', 'label', 'loadpath', 'locale', 'logscale',
        'mapping', 'margin', 'offsets', 'origin', 'output', 'plot',
        'palette', 'parametric', 'pm3d', 'pointsize', 'polar', 'print', '[rtuv]range',
        'samples', 'size', 'style', 'terminal', 'tics', 'timestamp',
```

```
'timefmt', 'title', 'variables', 'version', 'view',
'[xyz,cb]{2}label', '[xyz,cb]{2}range', '{m}[xyz,cb]{2}tics',
'[xyz,cb]{2}[md]tics', '{[xyz]{2}}zeroaxis', '[xyz,cb]data', 'zero'
```

This confuses gnuplot into asking you which variable you want to show, and it shows you all the options.

You can also do: **help set**, followed by one of the options listed.

If you want, you can type **show all** which dumps everything.

We won't be doing anything with **splot** for a while, but I wanted to introduce you to it since it produces some cool plots.

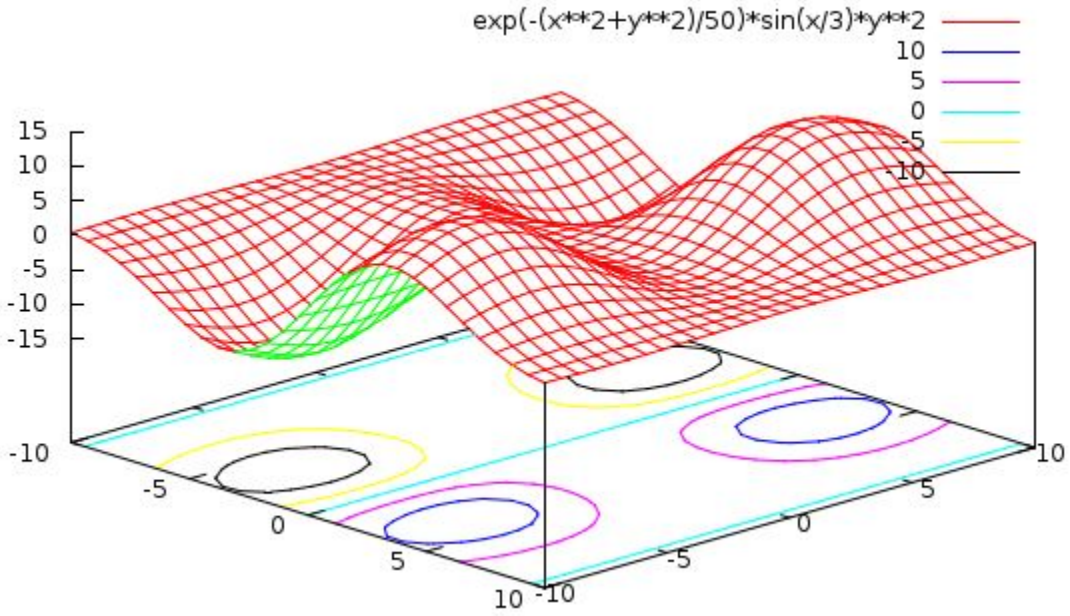
For a bit more eye candy, have a look at some of the plots on the Surfaces and 3-D Plots on the Demo page on the Gunplot site: Gnuplot Demos

You can make any of these plots with Gnuplot on your computer now! (you may need the data file from the Gnuplot site to reproduce some of them).

A really nice feature of the demo page is that the commands to produce the plots is also listed. Unfortunately, some of the plots require data files, and those are not so easy to get (I have them though, if you are really interested in reproducing one of the plots). Some of the plots use advanced features that we won't cover in this course, however you can always learn about these from the online documentation.

One more fun thing to do. Start gnuplot, type:

```
gnuplot> set isosamples 30
gnuplot> set hidden3d
gnuplot> set  **contour**
gnuplot> splot exp(-(x**2+y**2)/50)*sin(x/3)*y**2
```



Do you see what happened? You produced " contour " lines on the "floor" of the plot. This can be useful when your surface is complicated and you want to understand its structure. The contour plot is like a "topo" map that you may have used when hiking.

If you are alert, you will notice that even though the surface is quite bumpy, there is a particular straight path which is at constant "height". This would be very useful to know if you were a Civil Engineer building a road through this terrain.

If you just want the contour map, you can do this:

```
gnuplot> set size square
gnuplot> unset surface
```

Then grab the plot and look at it "from the top" (as if you are looking down the z axis.

You can get your surface back by doing

```
gnuplot> set surface
```

Next we must learn to plot our own data from a file , which we might have taken from an experiment for example.

Before moving on to plotting data files however, try experimenting with surface plotting of some functions of your own.

Make up some functions, f(x,y), and plot them.

5.4 Plotting Data from a File

Now we are getting down to it.

What we have done so is good for learning mathematics, i.e. plotting beautiful functions and seeing what they look like. We even learned to solve an equation "*Graphically*".

But as scientists we are often trying to learn something about physics from an experiment that we have done. In other words, we have some data and we what to understand what theoretical model or mathematical relationship is indicated

by this data. We can often tell that by just looking at a plot of the data, or in fancier language: *visualizing the data*. That's what we'll do now.

First, we need some data.

Open Emacs and enter the following data. You should be able to cut and paste it from your browser window) using the *Edit->Copy/Paste* tab in the window frame.

It is important that you have columns of ASCII (or text) numbers with "white space" in between them.

"White space" means either a single or more SPACEs (as when you hit the space bar), or a TAB.

```
0.0 0.24
0.5 0.66
1.0 1.56
1.5 2.05
2.0 2.80
2.5 1.74
3.0 2.54
3.5 3.36
4.0 4.61
4.5 4.44
5.0 5.61
5.5 4.68
6.0 6.72
6.5 5.97
7.0 7.07
7.5 8.38
8.0 7.44
8.5 9.34
9.0 8.12
9.5 9.97
10.0 10.54
```

Cd to your gp/ dir and save this file as *data1.dat*.

5.4.1 Plotting your first data file

Now that you have some data (and have " **cd**ed" to the place where gnuplot can find it)

start gnuplot.

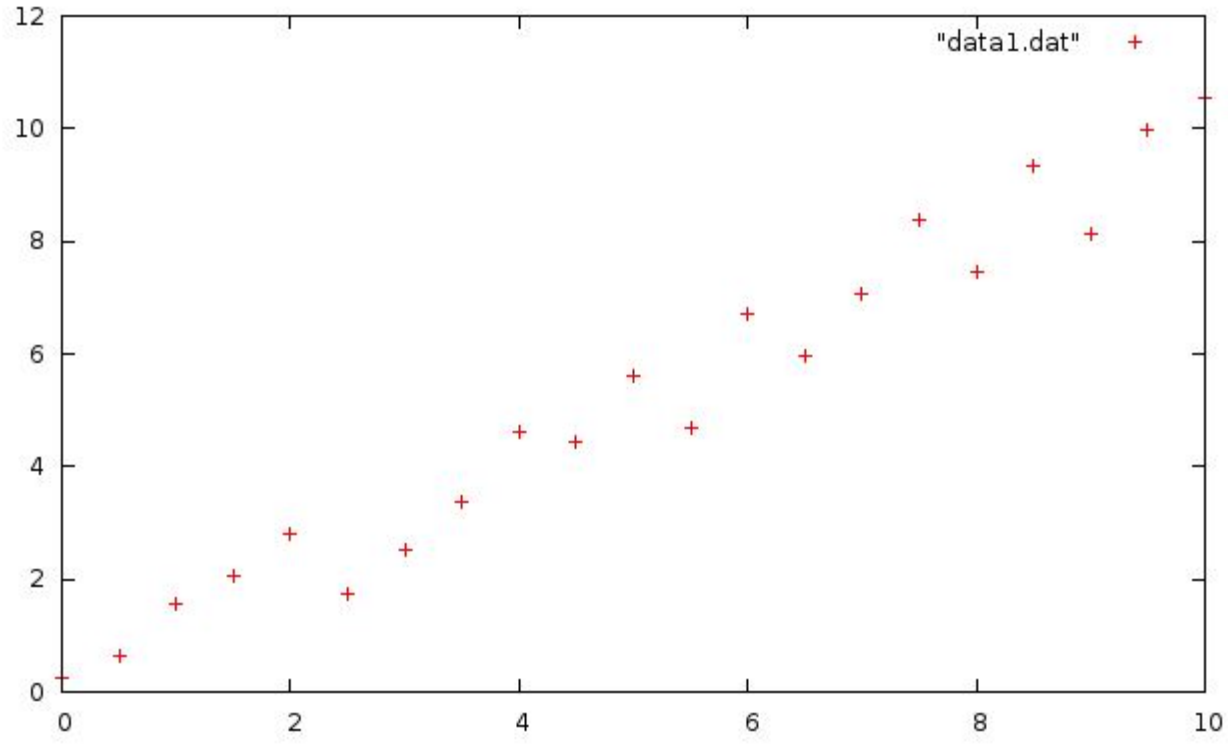
Type:

```
gnuplot> plot "data1.dat"
```

Be sure to include the " " s.

You need to do this anytime you refer to a **"file"** or **"folder"** on your disk, such as a data file, as I have shown in this sentence.

Your plot should look like this:



5.4.2 Plotting Data, Step 2

By this point you show have a plot of some little red +'s representing the data shown above and saved in **data1.dat**. Let's play around with this plot a bit.

One of the first things I notice is that the + in the uppermost right corner is not one of the data point in the file! That's because it is the plot symbol in the key, which in this case, is confusing since it is near other data points.

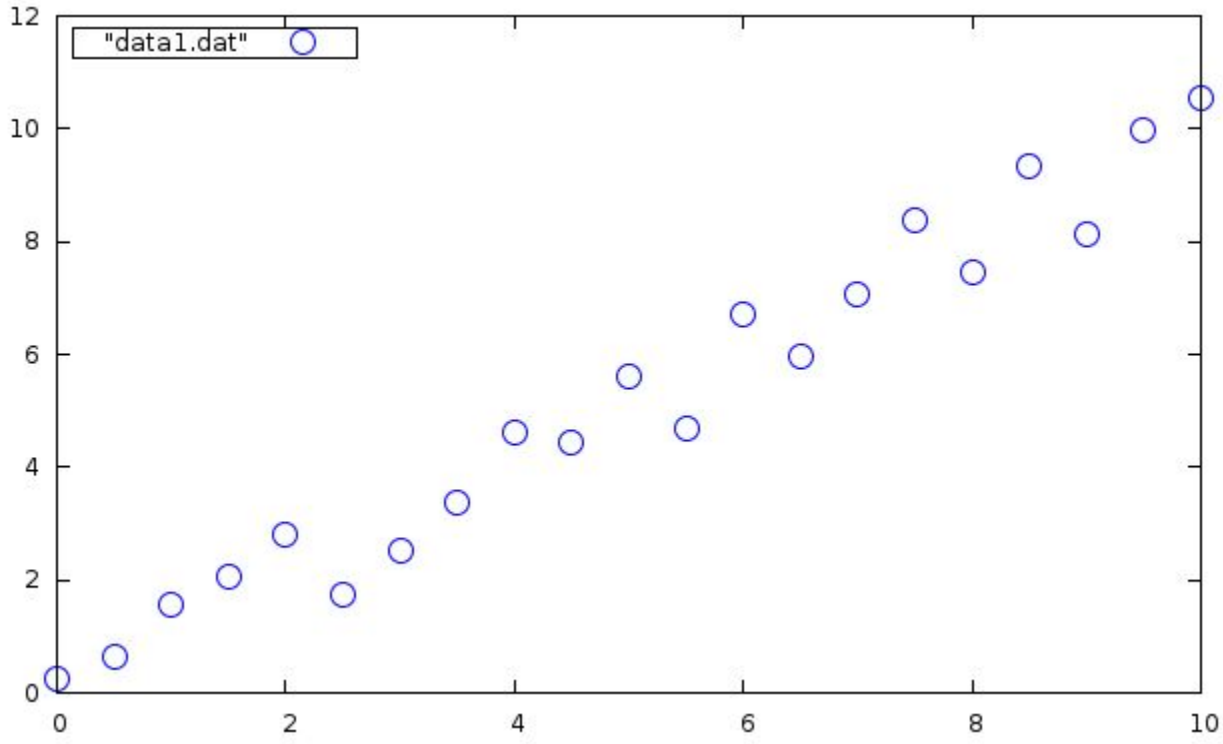
There are several ways to change this. The simplest is to just move the key:

```
gnuplot> set key left
gnuplot> plot "data1.dat"
```

Ah. Much better. You could also out a box around the key; remember how? (see Step Three: Label the Graph:).

Those points are a little small. Let's change them. This time do the plot with the "**with**" option:

```
gnuplot> plot "data1.dat" with points pointtype 6 pointsize 2 linecolor 3
```



Now we have larger blue circles for data points! Nice. Also, you can abreviate that long line, as you can with most gnuplot command options and variables, by typing:

```
gnuplot> plot "data1.dat" w p pt 6 ps 2 lc 3
```

Isn't that a whole lot easier? Try changing the " **pointtype** (pt), the " **pointsize**" (ps), and " **linecolor**" (lc), and see how your plot changes. You can see the various line/point types and colors by typing **test**. This will send a test pattern to the screen and show the things available to your screen.

```
gnuplot> test
```

To see the options available to the *with* command, type *help plot with* at the *gnuplot>* prompt. You'll have to hit *[SPACEBAR]* to page through the help pages and get back your *gnuplot>* prompt.

Summary

So far we have learned the following:

- What is *Scientific Visualization*.
- How to plot a few **mathematical functions** (or two, or three, of them...)
- How to customize the plot with **labels, a title, x and y plot ranges, xtics*, etc.
- How to **solve an equation graphically**
- How to **surface plot** functions **f(x,y)** of x and y in 3-dimensions, using **splot**.
- How to **customize plots** (move/modify the Key , set isosamples and set hidden3d)
- How to turn on **contour plotting**.
- How to **Plot data from a file**.
- How to customize the **point size, shape**, and **color**

You've learned a fair amount already!

5.5 Homework

Homework 5 is here

CHAPTER

SIX

DATA ANALYSIS

6.1 User Defined Functions

Let's learn how to define our own functions in gnuplot.

As you know, we can plot all kinds of mathematical functions (see Builtin Functions) with gnuplot. However, an important class of functions are ones that we can make up ourselves out of other functions. If it's not already running, start *gnuplot*.

We already learned how to use gnuplot as a calculator to define our own variables, as you did in Homework 5, where you calculated the power output of the sun. There, you defined constants made of numbers and builtin constants, such as

```
gnuplot> T = 5.5e3              # set temp. of sun's surface
gnuplot> r = 7e8                # solar radius
gnuplot> A = 4 * pi * r**2      # surface area
gnuplot> print A
6.15752160103599e+018           # in m**2
gnuplot> P = flux * T**4 * A    # solar power output
gnuplot> print P
3.27/19391028086952e+026           # in watts
```

You can also define functions.

Try this:

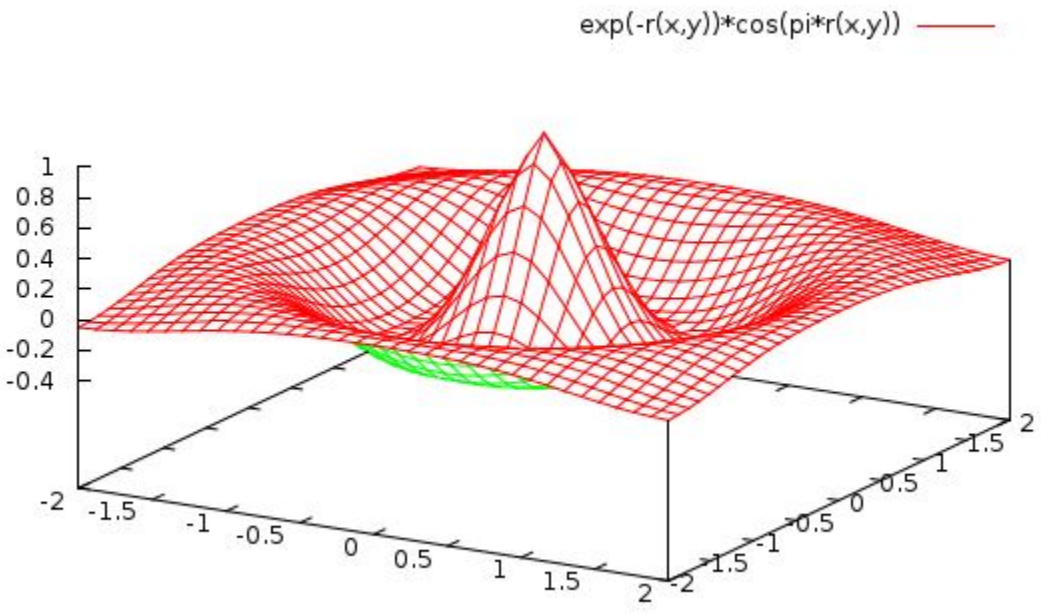
```
gnuplot> f(x) = 3*x**2 - 2*x + 4
gnuplot> plot f(x)
```

You should have gotten a parabola. You can also define a function of several variables:

```
gnuplot> f(x,a,b) = a*x - b
gnuplot> plot f(x,1,2), f(x,3,-2)
```

You can use functions in other functions

```
gnuplot> set hidden
gnuplot> set isosamples 30
gnuplot> r(x,y) = sqrt(x*x + y*y)
gnuplot> splot [-2:2][-2:2] exp(-r(x,y))*cos(pi*r(x,y))
```



You can build quite complicated functions with gnuplot. Remember however, that the variable that is used for plotting in a 2-D plot is always *x*.

Note: Actually what I said above about *x* is not 100% true; you can do:

set polar

which will make the default plot variable *t* (for "**t**heta ").

This makes gnuplot plot functions as distance from the origin (radius) vs. angle t . The default range is 2*pi radians).

6.1.1 The Conditional Function

One very useful construct that I sometimes use is the C programming **if/then** *shorthand* notation which gnuplot recognizes:

'(conditional phrase) ? (if TRUE value) : (if FALSE value)'

The *conditional phrase* is a question such as

```
x == 0            # is x equal to zero?
x != 0            # is x NOT equal to zero?
X > 4 || x < 3    # is x greater than 4  OR less than 3?
x >= 0 && x <= y  # is x greater to or equal to zero  AND less than or equal to y?
```

where we have used the programming logical conditions:

- ==

- !=
- >
- <
- >=
- <=
- &&
- ||

The last two, '&&' and '||', if you haven't figured them out already, are **&&**–meaning logical **AND** (both things must be true, for the whole phrase to be true), and ||–meaning logical **OR** (only one part needs to be true for the phrase to evaluate as true).

If the question part is TRUE, the result is the thing between the **?** and **:** If the question is FALSE, the result is the thing after the **:**

Here's an example. Plot a function whose value is f(x) = 5 if x is between 0 and 1, but f(x) = 0 otherwise.

```
gnuplot> f(x) = (x>0 && x<1) ? 5 : 0
gnuplot> set xzeroaxis
gnuplot> plot [-1:2][-1:6] f(x)
```

Try it! Such constructs could be handy in Digital Signal Processing classes.

You can even define functions **recursively** , such as the factorial function, **n! = n*(n-1)*(n-2)...3*2*1**.

```
gnuplot> fac(n) = (n != 0) ? n*fac(n-1) : 1
gnuplot> print fac(3)
6
gnuplot> print fac(5)
120
gnuplot> print fac(8)
40320
gnuplot> print fac(10)
3628800
```

6.1.2 Functions with Constants

One last thing. Notice that I can define a function with a variable, c, in it:

```
gnuplot> f(x) = c/sqrt(1 + x)
```

However when I try to plot the function, gnuplot complains that the variable c is not defined, hence it, rightly, doesn't know how to plot the function.

```
gnuplot> plot f(x)
        undefined variable: c
```

If I go ahead and define c

```
gnuplot> c = 5
gnuplot> plot f(x)
```

gnuplot can now plot the function, since it knows how to evaluate it.

I can update the value of c by typing

'c = 10' , and 'replot',

and I'll get a new function 'f(x) = 10/sqrt(1 + x)'.

Notice too that gnuplot stopped plotting for x <= 1 , since the function becomes complex for x < 1. In fact it's singular at x = -1. It will however, still tell you the complex value if you ask:

```
gnuplot> print f(-2)
{0.0, -5.0}
```

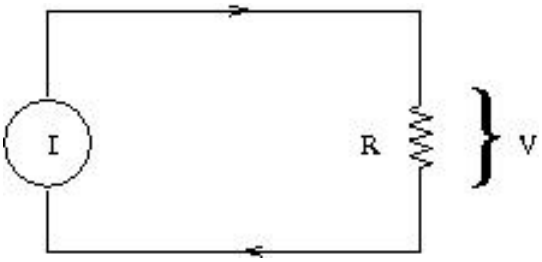
Next, we'll use the ability to define functions with constants to let gnuplot find the "*Best*" constants for the function to "*fit*" our data.

6.2 Fitting Functions to Data

So, now we can do science, i.e. discover something about the world.

We'll start with a trivial example.

Suppose you and a friend are confronted with the following circuit in lab (likely you did this in the 5rd grade, or later in PHYS 55).



You have a variable (i.e. at your control) current source, *I*, in a simple circuit with a resistor, *R*, **whose value you do not know**.

As your friend varies the current, you are able to measure the voltage, *V*, across the resistor. As you know from your studies of circuits, the voltage should be related to the current by *Ohm's Law*:

$V = IR$

Thus, if you plot your measurements of *V* versus *I*, you could find the resistance, *R*, from the slope of the line that goes through your data.

Here's the data. You can also download the file iv.dat to save you typing it in by hand (right-click, Save Link).

```
# I    V_R
#amps volts

0.00  0.00
0.10  6.77
0.20  33.84
0.30  42.38
0.40  53.07
0.50  69.87
0.60  81.45
0.70  96.91
0.80  103.90
```

```
0.90  111.99
1.00  125.11
```

Before we begin, notice that the data file has comments at the top, lines which begin with a '#' . You can put comments anywhere in your data file; everything on the line after a '#' will be ignored by gnuplot.

Start gnuplot and plot this data.

You can see that the voltage rises sort of linearly with increasing current.

Estimate the slope by eye.

What is your estimate? (write it down or remember it for later)

Let's try to fit a line to the data "by hand".

Define a linear function: B

```
f(x) = a*x + b
```

(that's a line, right?)

Since the first data point is (0,0), and since Ohm's law has no constant term, let's set:

```
gnuplot> b = 0
```

(*Knowing this, we could just as easily defined f(x) = a*x.*

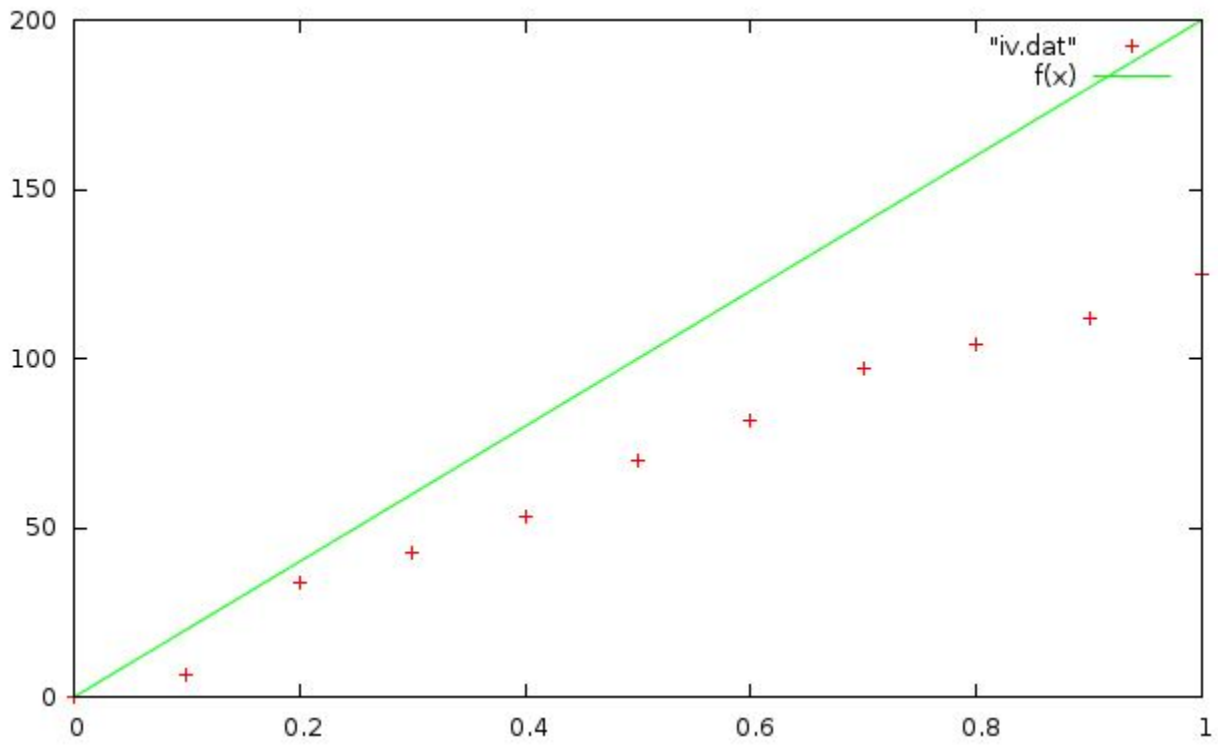
To set *a*, take your in initial guess from above:

```
gnuplot> a = your estimate from above
```

This is a number! You will be typing something like 'a = 200'.

Now, plot the data and your function together.

```
gnuplot> plot "iv.dat", f(x)
```



How well does your guess match?

You can refine your guess by typing a new value for *a*, and doing 'replot'. If the 'f(x)' line is too steep, set *a* lower.

Try it.

Do you see how the line changes as you change the slope, *a*? Try to zero in on the best line that "fits" your data.

What is your best estimate of *a*?

6.2.1 Least Squares Fit

Gnuplot has a wonderful builtin command to automate this process, in a very powerful and flexible way.

The command is called, not surprizingly, 'fit'. You use it like this:

```
gnuplot> fit f(x) "iv.dat" via a
gnuplot> plot "iv.dat", f(x)
```

The syntax of the 'fit' command is:

'gnuplot> fit function "datafile" via var1, va2, var3,...'

Here

- **fit** is the fitting command (this is always the same).
- **function** is a function which you have defined, including some "*variables*" within it, such as 'f(x) = a3*(x**3) + a2*(x**2) + a1*x + a0'.
- **"datafile"** is the file where your data is located. Remember that since this is a file , you must refer to it enclosed in **""**s.
- **via var1, var2, var3,...** are the parameters (the constants) you want gnuplot to tweak until it gets the best fit. You must have at least one parameter **var1**, and the keyword **via**.

When you run the 'fit' command, it will spew a fair amount of information as it changes each of the variables you specified after 'via'.

Finally, it will converge on the " Best " values, and print a summary of the results, which looks like this:

```
gnuplot> fit f(x) "iv.dat" via a
...

After 3 iterations the fit converged.
final sum of squares of residuals : 231.793
rel. change during last iteration : -8.79516e-08

degrees of freedom    (FIT_NDF)                       : 10
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 4.81449
variance of residuals (reduced chisquare) = WSSR/ndf   : 23.1793

Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 130.403          +/- 2.454        (1.882%)


correlation matrix of the fit parameters:

               a
a               1.000
```

This contains some statistical information about your data and the level of its randomness, most of which is beyond the scope of this course. However, it's nice to know that it's there as you learn more about statistics in subsequent courses. We'll look at these things a little bit soon.

The important line is:

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 130.403          +/- 2.454        (1.882%)
```

This is your result: you have just **analyzed your data** and found the resistance, R, to be about **130 Ohms, plus or minus 2.5 Ohms.**

Whoa...

6.3 Fitting Data II

In the previous lecture, we did a very simple fit of a straight line to some data, extracting the value of slope of the data.

But how does gnuplot define the "*Best*" fit?

This goes under a very old branch of mathematics called "*Linear Regression*" (when fitting linear functions), or "Least Squares*" fitting, or sometimes "*Data Modelling*".

In it's simplest form, the procedure works like this. We'll consider a very simple linear fit to three data points.

Suppose your experiment gives you these three data points:

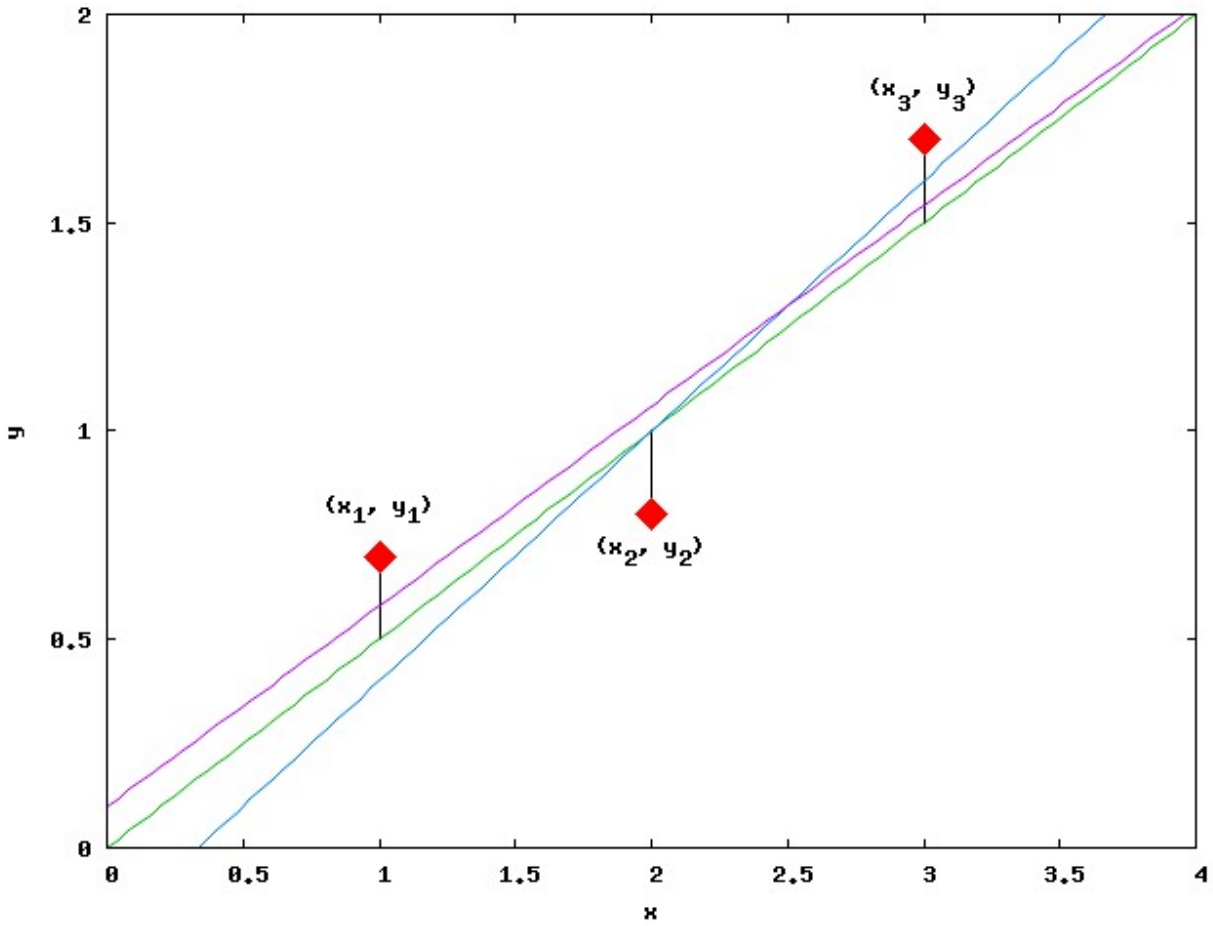
```
1.0   0.7    # x1, y1
2.0   0.8    # x2, y2
3.0   1.7    # x3, y3
```

You define a function *f(x) = a*x + b* which has some parameters that you can change, **a** and **b** in this case.

Each different value of *a* and *b* gives a different line; three different possible line examples are shown below.

But again, how do you know which line is the *best* fit (*i.e.* comes closest to the data)?

For each data point (labeled by *i = 1,2,3*), we compute the **difference** between our guess function $f(x_i) = ax_i + b$ (at x_i) , and the y_i value of the real data.



This difference is called the *Residual* at x_i . The residuals of each data point are shown as the little vertical lines (|) in the figure above.

Since these can be either positive or negative, we *square them* to get a positive number irregardless of whether the data is above or below the line.

Then we sum all the squares of residuals.

$$\begin{aligned} R &= \sum_i [y_i - f(x_i)]^2 \\ &= \sum_i [y_i - (ax_i + b)]^2 \end{aligned}$$

Then: the **fit** program changes the parameters ('a' and 'b' in our case, but there could be more) to get the *lowest sum of the Squares* of the Residuals , *min{R}*, hence the name "**Least Squares Fit**".

For those who know calculus, we are really doing is taking the derivative of *R* with respect to *a* and *b* and setting those expressions to zero; this finds where *R* has a minimum as we vary *a* and *b*.

This gives two equations that we have to solve, sumultaneously.

$$\frac{dR}{da} = -2\sum_i \left[y_i - (ax_i + b)\right] x_i = 0$$
$$\frac{dR}{db} = -2\sum_i \left[y_i - (ax_i + b)\right] = 0$$

These equations are actually easy to solve (email me if you need help). They are really just a *system of two coupled linear* equations, which you could use *Linear Algebra* to solve!

$$a\alpha_1 + b\beta_1 = \gamma_1$$
$$a\alpha_2 + b\beta_2 = \gamma_2$$

where

$$\alpha_1 = \sum_i x_i^2$$
$$\beta_1 = \sum_i x_i$$
$$\gamma_1 = \sum_i x_i y_i$$
$$\alpha_2 = \sum_i x_i$$
$$\beta_2 = \sum_i = N$$
$$\gamma_2 = \sum_i y_i$$

It's easy to solve the equations above for a and b:

$$a = \frac{\gamma_1\beta_2 - \gamma_2\beta_1}{\alpha_1\beta_2 - \alpha_2\beta_1} = \frac{(\sum_i x_i y_i)(N) - (\sum_i y_i)(\sum_i x_i)}{(\sum_i x_i^2)(N) - (\sum_i x_i)(\sum_i x_i)}$$

$$b = \frac{\gamma_2\alpha_1 - \gamma_1\alpha_2}{\alpha_1\beta_2 - \alpha_2\beta_1} = \frac{(\sum_i y_i)(\sum_i x_i^2) - (\sum_i x_i y_i)(\sum_i x_i)}{(\sum_i x_i^2)(N) - (\sum_i x_i)(\sum_i x_i)}$$

However, gnuplot doesn't actually solve them!

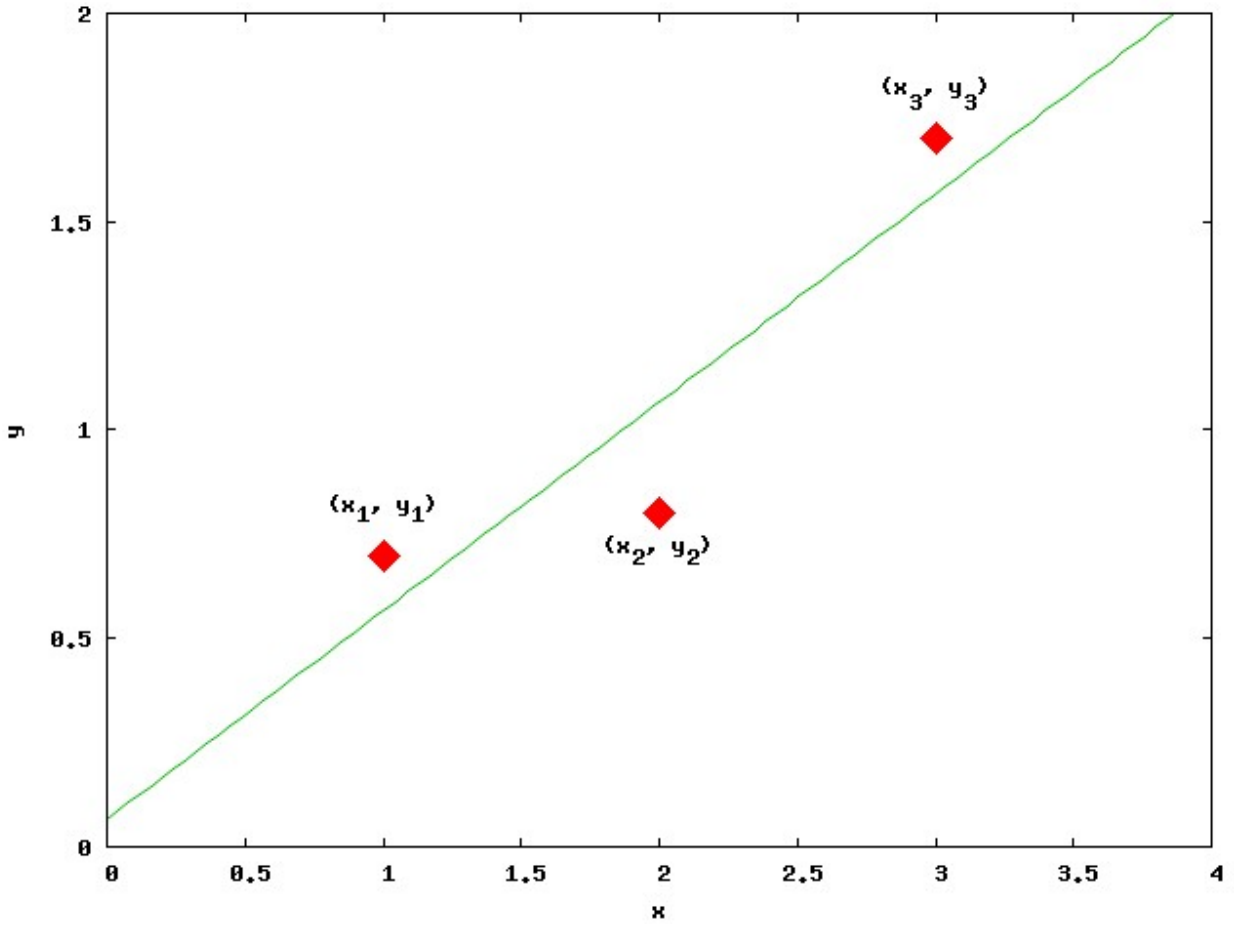
It just varies *a* and *b* and sees how much *R* changes. If increasing *a* decreases *R*, then gnuplot increases *a* a little more.

Once changing the parameters by a little amount always increases *R*, **fit** stops and prints out the resulting parameters:

```
After 4 iterations the fit converged.
final sum of squares of residuals : 0.106667
rel. change during last iteration : -3.71111e-09

degrees of freedom    (FIT_NDF)                       : 1
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.326599
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.106667

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
 **
a               = 0.5              +/- 0.2309       (46.19%)
b               = 0.0666667        +/- 0.4989       (748.3%)
**
```

This gives the paramerters of the line that is the "*Best*" fit to the data, as well as some information about how good the fit is (how big was the sum of the squares of the residuals).

This *Best* line looks like this. It's the one that's closest to the data.

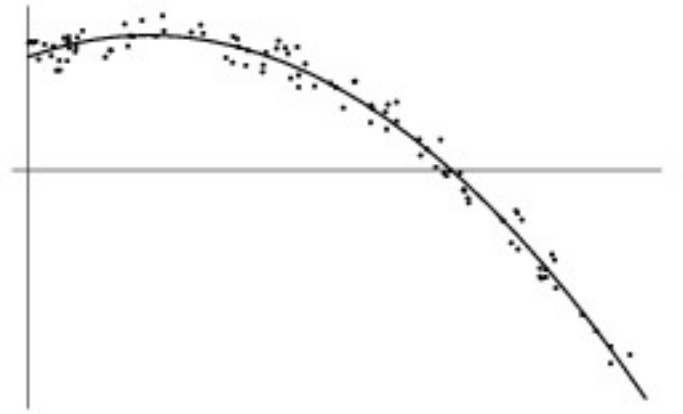


For more information on the *Least Squares* method can be found here:

- http://mathworld.wolfram.com/LeastSquaresFitting.html
- http://www.mathwords.com/l/least_squares_regression_line.htm

as well as any good math book on statistics.

As you can see from the figure below, we need not limit ourselves to linear functions–you could do the same with almost any function and data.



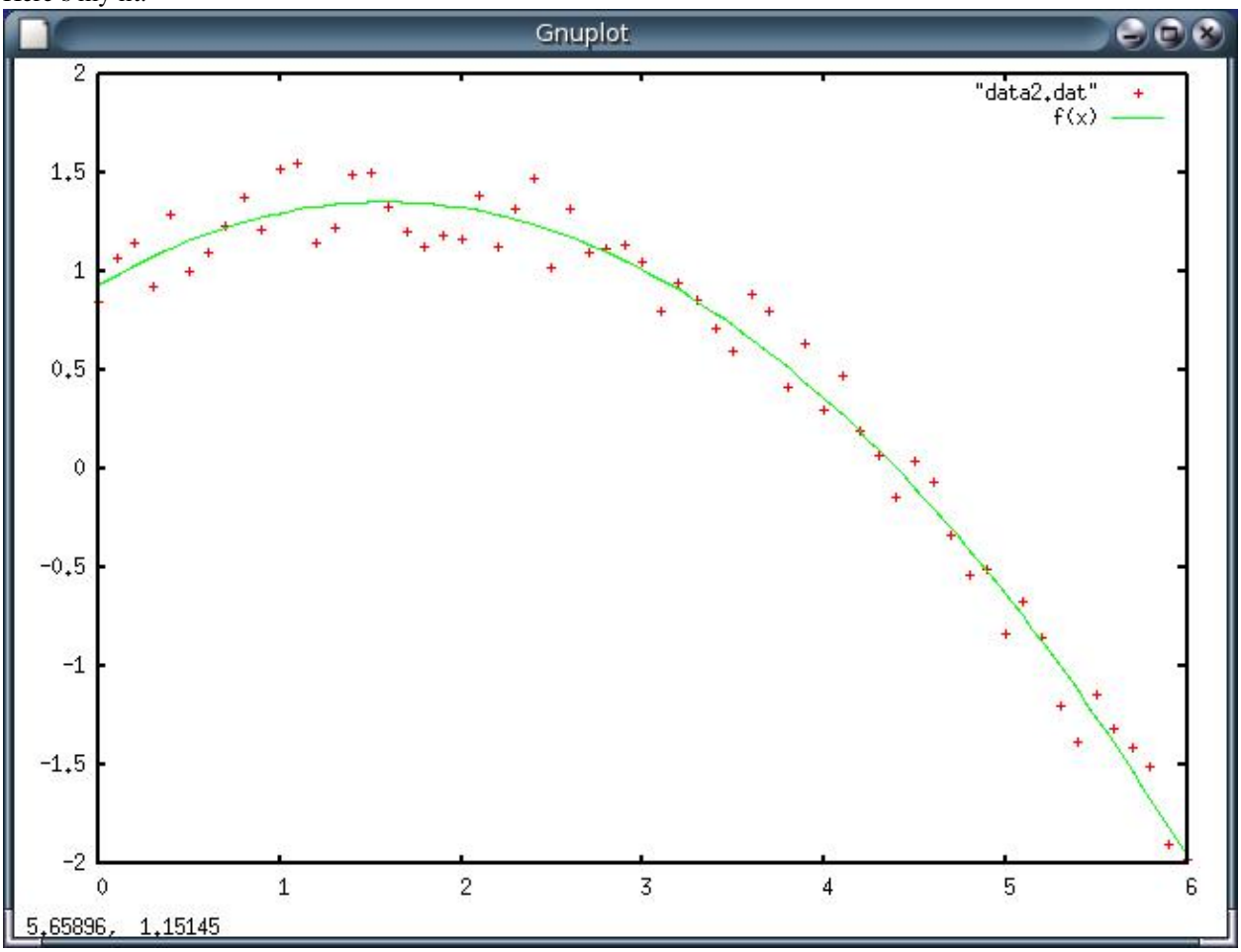
Your job as a scientist is to have insight (usually from *theory*) as to which function is correct.

In this case, you want a quadratic function. You would do the following

```
gnuplot> f(x) = a*x**2 + b*x + c
gnuplot> fit f(x) "data2.dat" via a ,b, c
gnuplot> plot "data2.dat", f(x)
```

Do you see that we've defined a quadratic function: :math"*a x^2 + b x + c* above? Try this yourself . Here's the data: data2.dat

Here's my fit.



Operator Binding or Precedence

Above, when writing about how to enter into gnuplot the fit function for a parabola, I wrote *f(x) = a*x**2 + b*x + c*
without parentheses. This probably bothers several of you, who would rather see the function written as:
f(x) = a(x**2) + b*x + c*
which is "*safer*" . As with most computer math operators (like * and /), the "exponent" operator ** acts only on the previous thing.
Clearly
*(a*x)**2*
is much different than
a(x**2)*.
I have been a bit lazy and used the fact that the exponent operator "*binds tighter*" than the multiplication operator.
When gnuplot reads the line
*a*x**2 + b*x + c*
it first squares *x* before anything else, because ** has the highest "*operator precedence*" or "*tightest binding*".
Then it does the multiplications *a*(x**2)* and *b*x*
Finally, it does the additions because + has lowest precedence.
But we could also use:
a(x**2) + b*x + c*

6.3.1 Using the "*Using*" command

A very *VERY* important feature of gunplot's *plot* and *fit* functions is the **using** argument. It has *MANY* uses, some of which we'll look at below.

Plotting data from different columns

Suppose your data file has separate columns of data that are all measured at the same time. You have an example of this in your "*YOURNAMEcalpop.dat*" file.

Go ahead and find it now.

Before we look at it in gnuplot however, we will have to edit it slightly. It will look something like this (right?):

```
Year       Total US            Total California
1900       76212168          1485053
1910       92228496          2377549
...
```

Start emacs and Open *YOURNAMEcalpop.dat*.

We have to make the first line (or lines) into a comment, ortherwise gnuplot will bail trying to plot the number "*Year*".

Remember how to make a line into a comment? If not see here: Plotting Data from a File. (Simply put a # at the beginning of the line.)

Now our file looks like this:

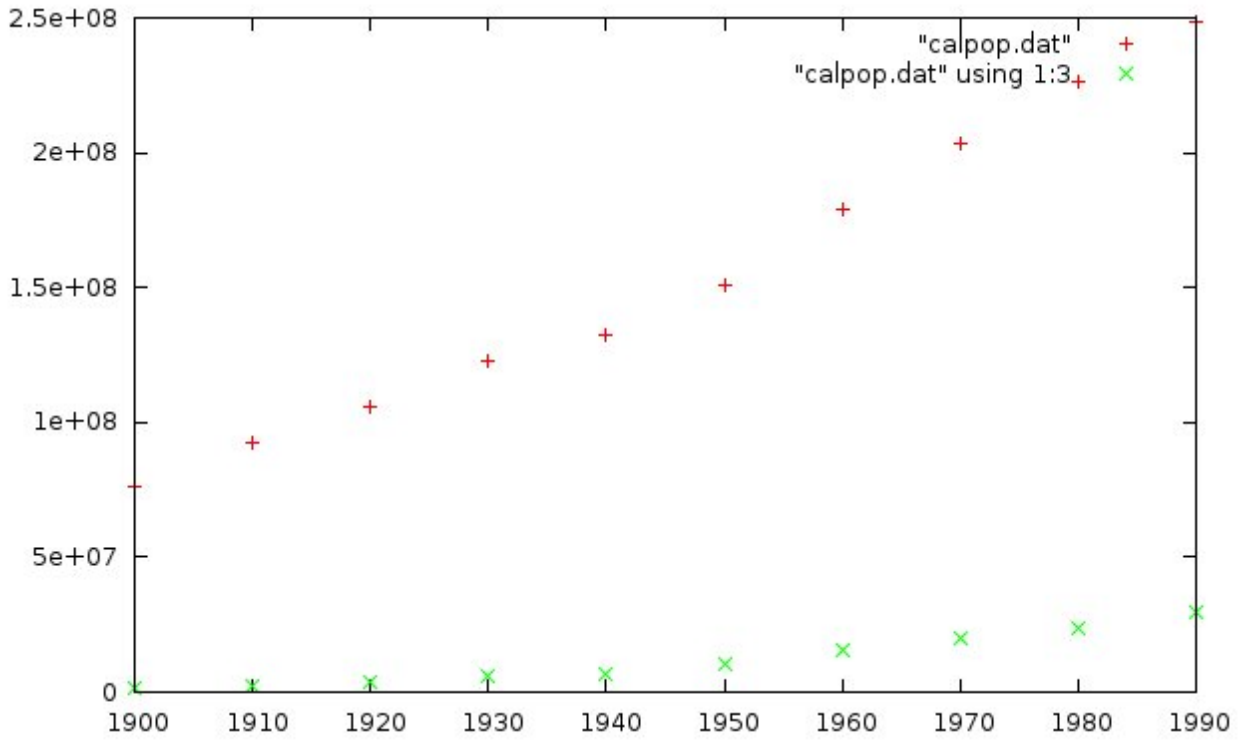
```
#Year       Total US            Total California
1900       76212168          1485053
1910       92228496          2377549
...
```

We can do a normal plot:

```
plot "calpop.dat"`
```

which plots column 1 and 2 on the x and y axes. However look at what this command does

```
gnuplot> plot "calpop.dat", "calpop.dat" using 1:3
```



While the first plot is the normal default one–columns 1:2 on x : y , the second part, *calpop.dat" using 1:3*, plots the data in *calpop.dat* again, but this time *using* columns 1 and 3.

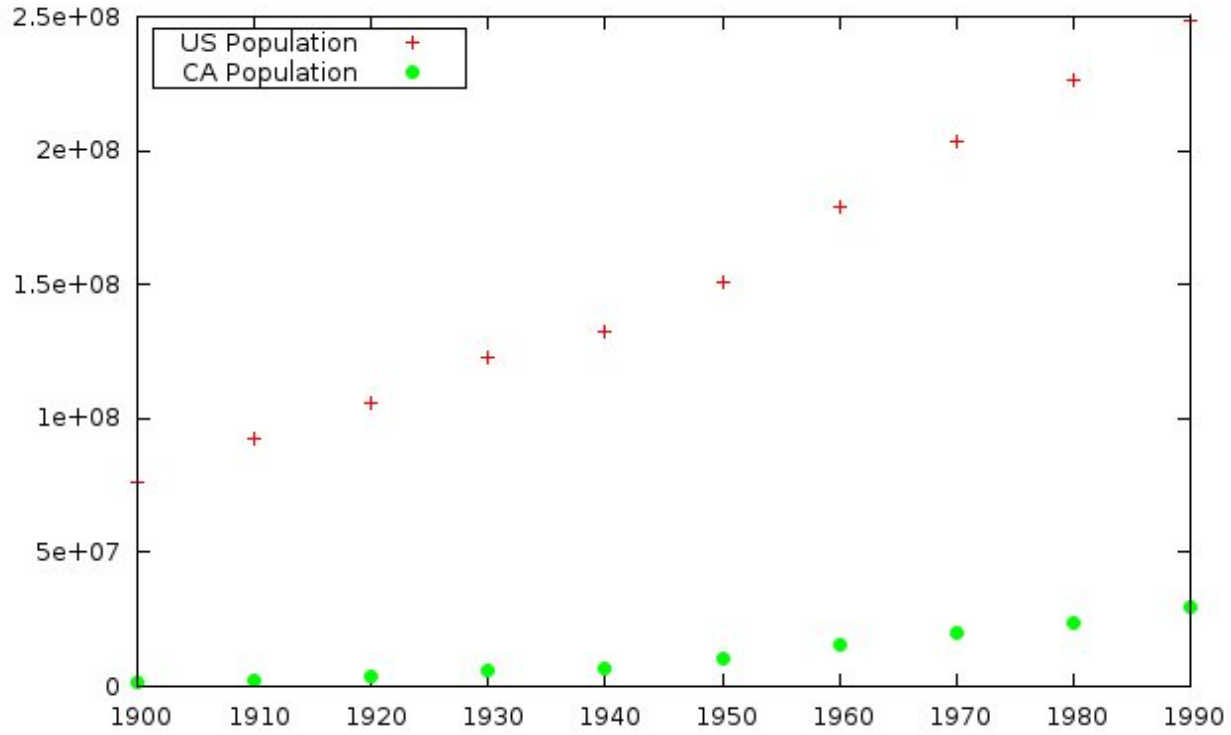
You can simplify this with the abreviated version:

```
gnuplot> plot "calpop.dat", "" u 1:3
```

The empty *""* means use the same datafile as before (with older versions of gnuplot you will have to supply the datafile name again), and *u* is short for *using*.

Also, since the *Key* is in the way of the data in the upper right, let's move it, and give the key items *title*s other than the default names for the *plot* command.

```
gnuplot> set key box left
gnuplot> plot "calpop.dat" title "US Population", "" u 1:3 title "CA Population" pointtype 7
```



This looks very nice. Notice that we:

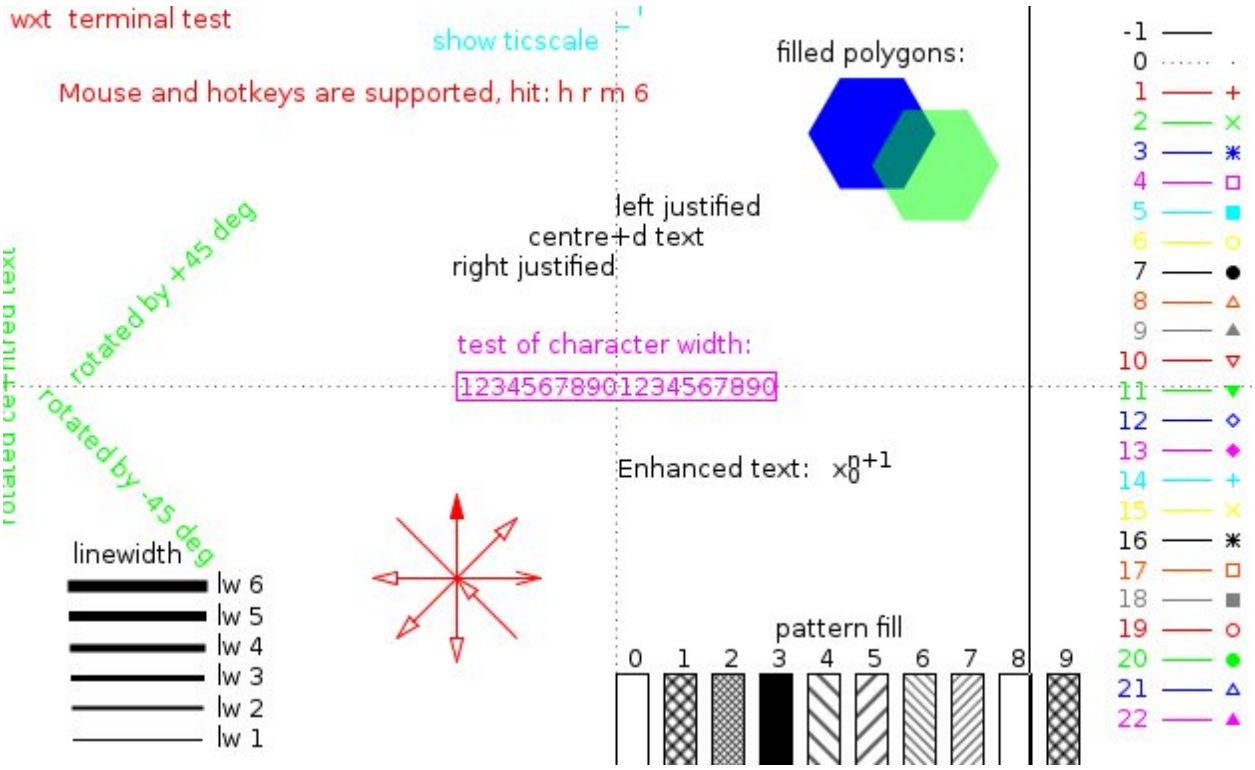
- moved the *Key* and put it in a box, with the single line: *set key box left*
- added *titles* to the key items with *title "string"*. The title character string must be enclosed in quotes.
- changed the *point type* used for displaying the data by specifying *pointtype 7*

This long command to gnuplot can be considerably simplified by abreviating *title* and *pointtype* to

```
gnuplot> plot "calpop.dat" t "US Population", "" u 1:3 t "CA Population" pt 7
```

You can see the different ‘pointtype‘s, and some other info, by typing

```
gnuplot> test
```



The *pointtypes* (and *linetypes*) are shown on the right side).

Of course you could do more if you had more columns, such as

plot "datafile" u 1:4, "" u 1:3, "" u 3:2

etc. Notice that the last instance, *u 3:2*, plots the 3rd column as the *x* values and 2nd column as *y* values of the points. Very nice, huh? But now–*Check this out*!

You can modify the data on the fly with *using*!

Try these examples:

```
gnuplot> plot "calpop.dat", "" u 1:(10*$3)
gnuplot> plot "calpop.dat" u 1:(1.0/$2)
gnuplot> plot "calpop.dat" u 1:($3/$2)
```

In the above examples we did

```
u 1:(10*$3)  -> plots 10 times the -value- in column 3
u 1:(1.0/$2) -> plots 1.0/y for column 2 values
u 1:($2/$3)  -> plots the ratio of the values in columns 3 and 2
```

Here's the syntax for modifying data with *using*:

Simple *i.e.* just plot different columns:

- The *using* command takes the form *u 1:4* to plot columns 1=x and 4=y, for example. No parentheses needed to just rearrange the columns used for *x* and *y*.

If you want to refer to the **value** in a particular column, so that you can multiply, divide, or act with other math functions on it, etc.

Manipulating column values in the plot:

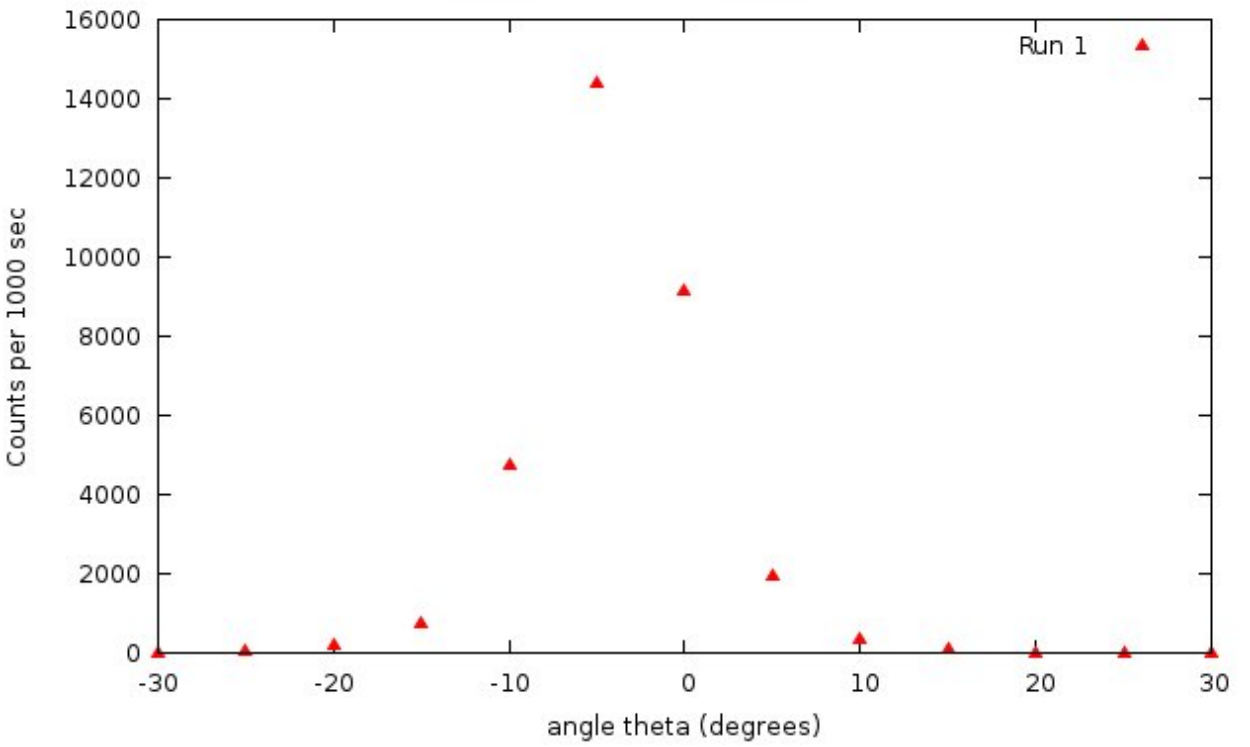
- instead of a column number [example *u 1:4*], you must enclose the column placeholder in ()s.
- *inside the ()*s, you must refer to **value** of a datum in column *n* as **$n**. Like this: *u 1:($4)*
- then, you can treat **$n** as the **value** of the datum for that point.

Here's an example. Our *Rutherford α -particle scattering* experiment produces the following data:

```
file: ruth.dat:

#theta  flux
-30     27.6
-25     65.4
-20     195.2
-15     762.4
-10     4740.8
-5      14392.6
0       9139.8
5       1957.6
10      361.6
15      113.6
20      35.8
25      23.2
30      12.2
```

and the following gnuplot commands create this plot:

```
gnuplot> set xlabel "angle theta (degrees)"
gnuplot> set ylabel "Counts per 1000 sec"
gnuplot> plot "ruth.dat" t "Run 1" pt 5
```



According to theory, these data should follow a curve that looks like this, where A and θ_o are constants (to be solved for):

$$N(\theta) = \frac{A}{sin^4[(\theta - \theta_o)/2)}$$

It turns out that the *fit* procedure has a hard time with this because of the singularity when θ is zero.

This is a common issue for curve fitting algorithms; if the fit function has a singularity, they often fail.

However, if we were to plot *the inverse* of our data: $1/N(\theta)$, our fit function to those data would be

$$\frac{1}{N}(\theta) = \frac{1}{A} sin^4[(\theta - \theta_o)]/2)$$

This function simply goes to zero as $\theta \rightarrow 0$, instead of blowing up. So the *fit* command is stable. It's really simple to plot the *inverse* of our data from the same file:

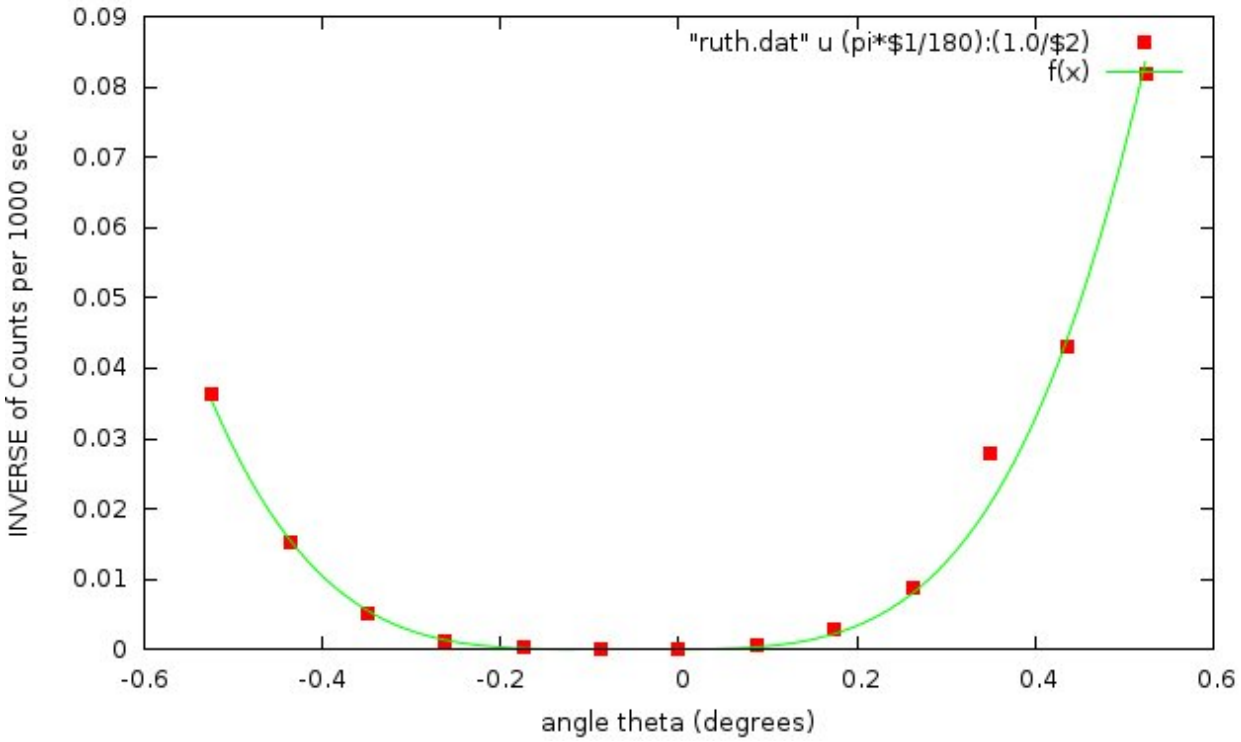
```
gnuplot> plot "ruth.dat" **u (pi*$1/180):(1.0/$2)** pt 5
```

Notice that I also converted the angle (in column 1) to radians.

Now it's easy to fit this data:

```
gnuplot> fit f(x) "ruth.dat" u (pi*$1/180):(1.0/$2) via a,x0
...
After 12 iterations the fit converged.
final sum of squares of residuals : 5.97834e-05
```

```
rel. change during last iteration : -4.08902e-07

degrees of freedom    (FIT_NDF)                        : 11
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00233128
variance of residuals (reduced chisquare) = WSSR/ndf   : 5.43486e-06

Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 12.394           +/- 0.3675       (2.965%)
x0              = -0.0577061       +/- 0.003981     (6.899%)
```



Shazzam!

Mathematical Manipulations of Data with *Using*

With the *using* command, you can do simple or complicated mathematical transformations of your data when you plot it.

A very simple example is this. Suppose you you have these two data sets:

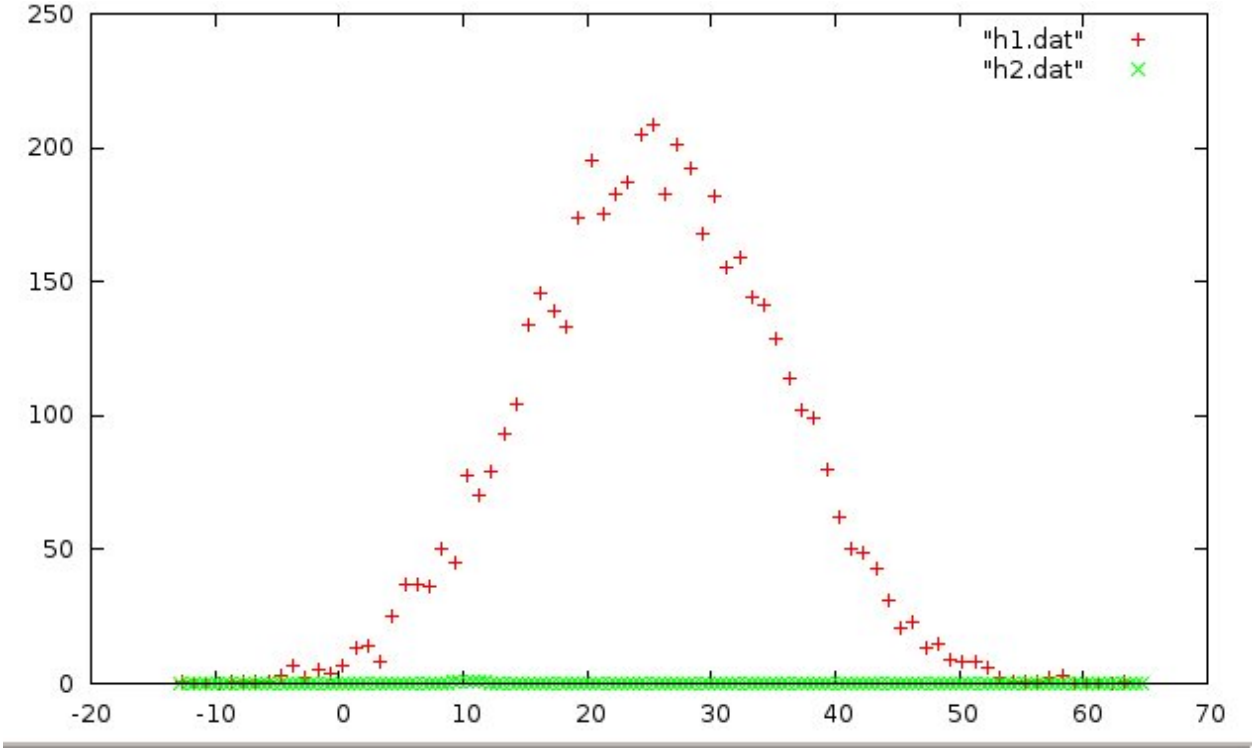
- h1.dat

- h2.dat

If you plot them,

```
gnuplot> plot "h1.dat", "h2.dat"
```

they look like this:



The *h1* data shows some interesting structure: clearly there is something happening at around $x = 25$, but it's pretty hard to see what's going on with *h2*. Have a look at the data files by scrolling throuh them with *less*:

less h1.dat

```
...
1.130134e+01    70
1.230134e+01    79
1.330134e+01    93
1.430134e+01    104
1.530134e+01    134
1.630134e+01    146
1.730134e+01    139
...
```

and

less h2.dat

```
...
1.275000e+01 0.426
1.325000e+01 0.346
1.375000e+01 0.324
```

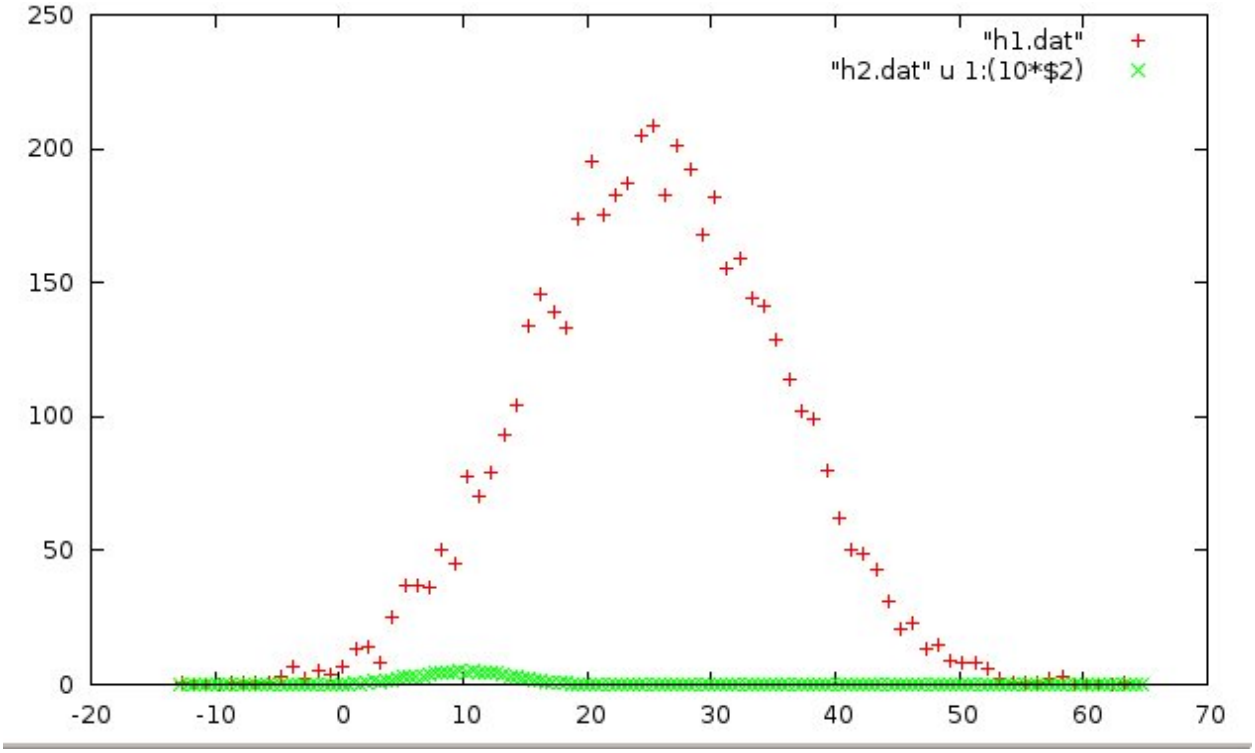
```
1.425000e+01 0.292
1.475000e+01 0.252
1.525000e+01 0.198
...
```

Ah. Now we see what's going on. Where *h1.dat* has values around ~100, *h2.dat*'s values are around ~01. Perhaps we are looking at the energy spectra from two different sources, one much more intense than the other.

Let's adjust this with the *using* command and scale the *h2* data.

Plot the data this time with:

```
gnuplot> plot "h1.dat", "h2.dat" u 1:(10*$2)
```

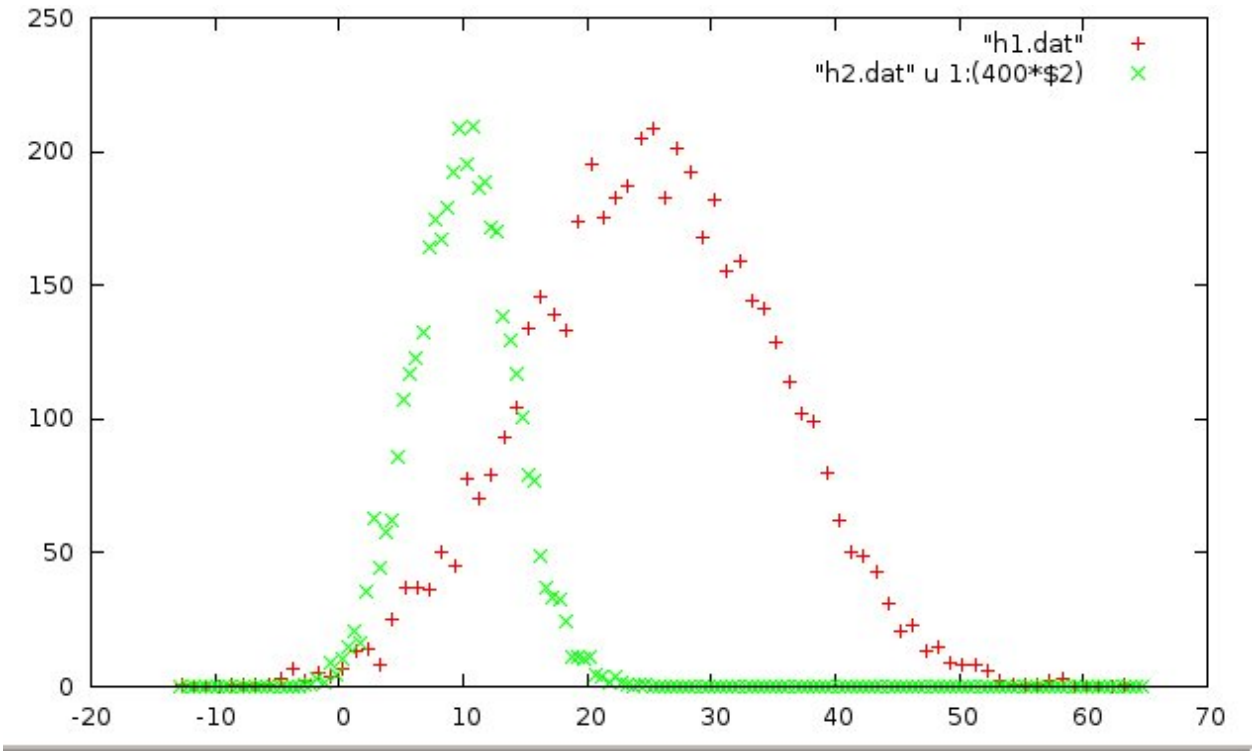


Notice the syntax to the *using* (*u*) command: **u 1:(10*$2)**. This means "use the column 1 values as they are for *x*, then take the value of the column 2 entry ($2), multiply it by 10, and use it for the *y* value (10*$2) of the point to be plotted.

Now we see something emerging–a little bump at ~*x = 10*.

We can blow this up even more by increasing our "scale factor" in the *using* command. Try:

```
gnuplot> plot "h1.dat", "h2.dat" u 1:(100*$2)
gnuplot> plot "h1.dat", "h2.dat" u 1:(400*$2)
```



Now we can see something important. The *h1* data has a peak around ~25, whereas the *h2* data peaks around ~10. This is a simple use of the *using* feature.

While you need to know what you are doing, know that you can use any of the built-in math functions with *using*. In fact you can build quite complicated functions of your own and apply them.

Here's a very simple example:

Suppose your file looked like this:

```
# x   y

  1   1
  2   10
  3   100
  4   1000
```

the *using* command

```
u 1:(2*log10($2)+1)`
```

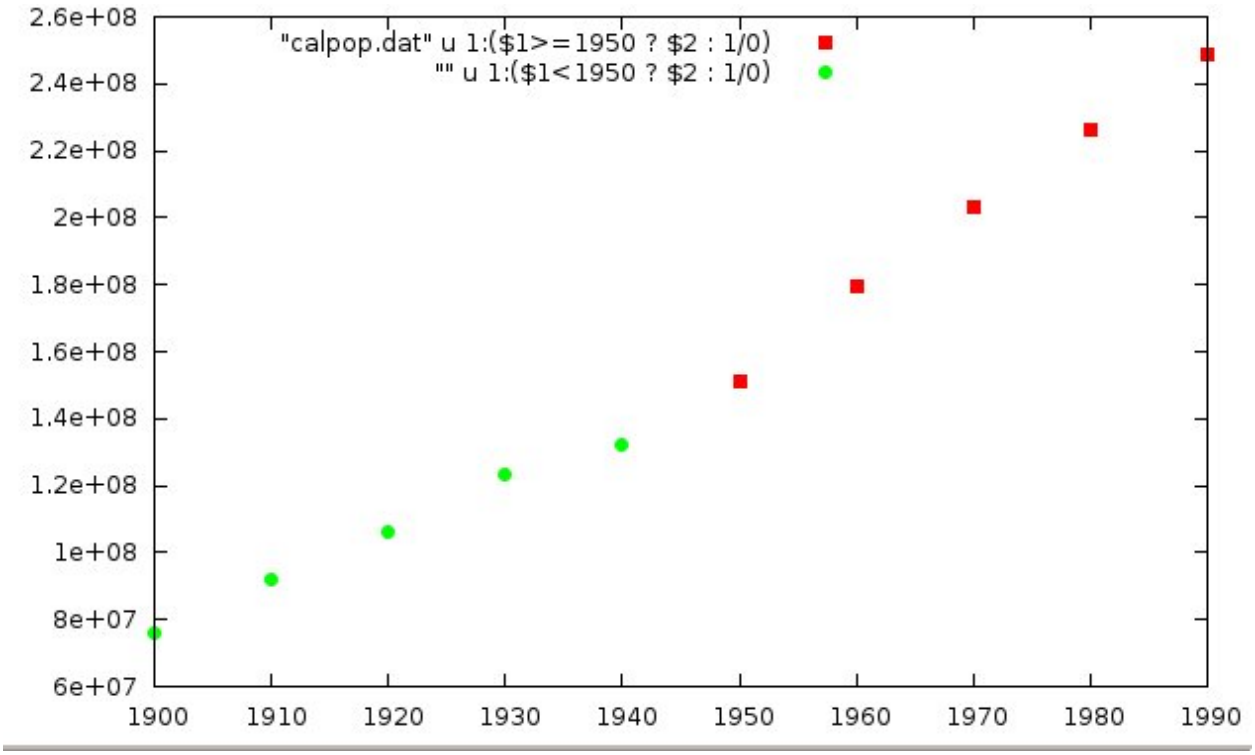
would plot these data as if they were

```
# x             new y

  1   2*0+1     =  1
  2   2*1+1     =  3
  3   2*2+1     =  5
```

Using and If/Then

Here is another of my favorite handy *using* examples. Make sure you understand what it does.

```
plot "calpop.dat" u 1:($1>=1950 ? $2 : 1/0) pt 5, "" u 1:($1<1950 ? $2 : 1/0) pt 7
```



This uses the fact that *1/0*, *division by zero*, evaluates to undefined , so gnuplot ignores that point.

The IF/THEN (actually called the "*ternary*") operator works like this:

```
y = Conditional ? Value if True : Value if False
```

Ususally, when we write **y = x** we are set **y** to the *value* of **x**. So if the value of $x = 5$, then after the statement $y = x$, the *value of y* is now 5.

In the *ternary* operation, in order to determine the value that *y* should be set to, the *Conditional* statement is evaluated. A *Conditional* statement is either **True** or **False**, like $x > 0$ or $x <= 10$. If the *Conditional* is *True* then the value used for *y* is the first value–the one between the **?** and the **:**. If the *Conditional* is *False*, then the second value is used, the one after the **:**.

In the example above,

```
u 1:($1>=1950 ? $2 : 1/0)
```

the first part of the *using* operator *u 1:*, takes the first column of the data file (Year) for the *x* value of the plotted point, as per normal. However the second part *($1>=1950 ? $2 : 1/0* runs the *Conditional* first: *$1>=1950*, and checks if the

year ($1 is the value in the *1st*-column) is less than 1950; if YES it is, the *Conditional* returns **True** and the "normal" value as listed in the file: *$2* meaning the value from column 2 is used for the *y* coordinate of the point.

If the *Conditional* is *NOT* satisfied (evaluates as **False**), then the *y* coordinate is set to 1/0, which is undefined. So gnuplot ignores this point and does not plot it. The result of the above *using* expression is that only points whose year is greater or equal to 1950 are plotted.

To make the plot shown above, with red squares and green circles, plotted two versions of the data, each with its own conditional.

For the first data set in the plot, the *u* command asks if the *x* value (the year) is greater or equal to 1950. If so, it is plotted as is, with *pointtype (pt)* 7.

In the next data set in the plot, if *x* (the year) is less than 1950, the *u* command returns that point's normal *y* value, and it is plotted (this time with *pt 7*). In the second case, points for years > 1950 are *not* plotted.

This is a handy way to suppress plotting or fitting of some part of your data.

More fun with the Ternary

Finally, one last example. This one goes out to all of you who have to take *Digital Signal Processing*.

First, let me introduce a couple interesting math functions.

```
floor(x) returns the largest integer less than x.

 Thus: floor(4.73) = 4
       floor(9.99999) = 9
       floor(pi) = 3
       etc.
```

and the **mod** operation, for which we use the percent symbol: %

```
For two integers a and b:

a % b returns the remainder of integer division.

Thus: 7%3 = 1
      12%6 = 0
      24%5 = 4
       etc.
```

A convenient *Conditional* to check if an integer *a* is EVEN, is

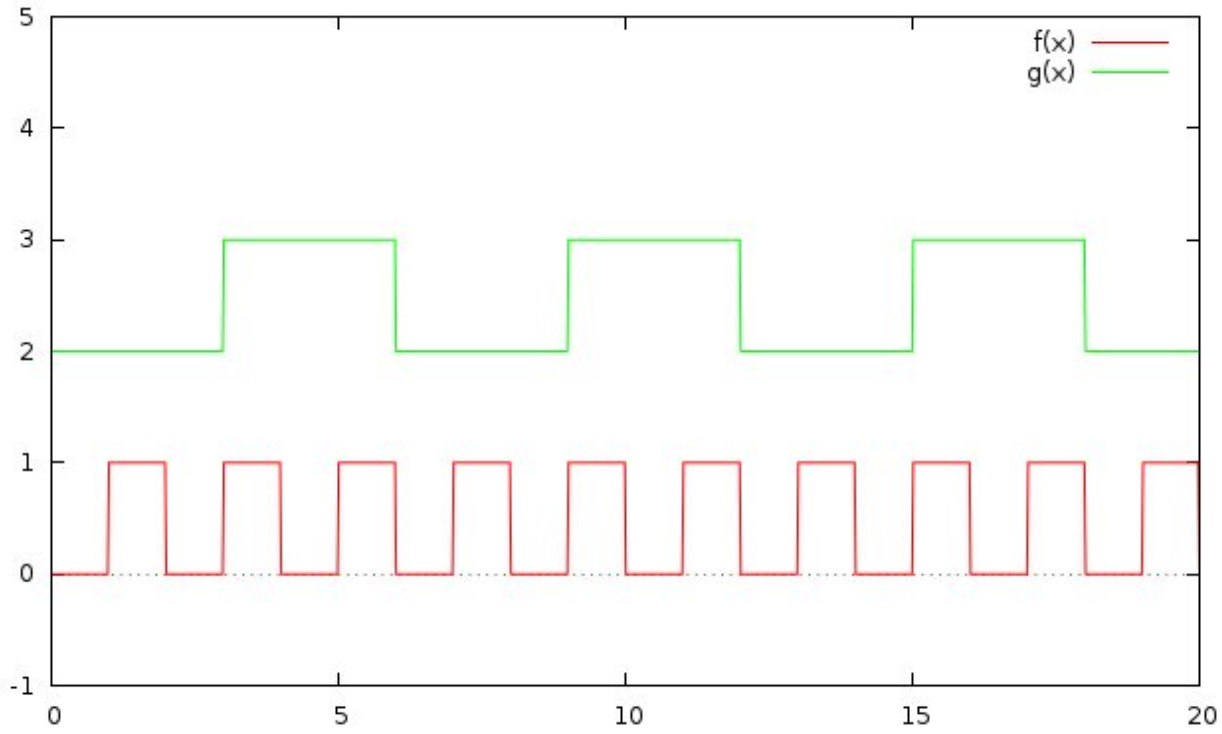
```
a%2 == 0
```

If this is **True**, then *a* must be even (divisible by 2 with no remainder). If it's **False**, then the remainder is 1, and the number is ODD.

The *mod* operation only works for *integers* however, but you can combine the *floor* operation to extract the integer part of a number, then use that in a *mod* expression.

Now consider this graph. I have *set samples 1000* to get a finer plot with 1000 sample points, rather than the default 100 points. That way, my "edges" are more vertical. If you *set samples 100* back to the default, you'll see what I mean.

```
gnuplot> f(x) =  floor(x)%2 == 0 ? 0 : 1
gnuplot> g(x) =  floor(x/3)%2 == 0 ? 2: 3
gnuplot> set zeroaxis
gnuplot> set samples 1000
gnuplot> plot [0:20][-1:5]  f(x), g(x)
```



6.4 Saving and Printing

Often you need to stop your work where you are (maybe to Sleep!) and save it, so you can come back to the project later. There are two different aspects to saving.

6.4.1 Saving a Plot to re-Load it later

If you want to save a plot so that you easily reproduce it later, you use the **save** command. Once you have a plot that you like, you simply do:

```
gnuplot> save "myplot.gp"
```

This will create a new text file named "**myplot.gp**" in the present working directory (remember *pwd*).

Since it's a text file, you can easily look at it, and even edit it if you wanted.

Try it. Make a plot of your favorite function, then **save** it as shown above .

Have a look at this file with **less**.

An easy way to do this is to " CTRL-z " out of gnuplot. This is another bit of Unix magic.

At the gnuplot prompt, type CTRL-z

This should pop you out of gnuplot (actually it suspends gnuplot and gives you back your Terminal shell prompt). At the shell prompt, you can do

less myplot.gp

and have a look at your file. Notice in particular the last lines which have the plot command that you used to make your graph.

When you're done looking at your file, your can bring gnuplot back by putting it in the *foreground*. At the shell prompt, type

fg

then ENTER . You should now have your *gnuplot>* prompt back. (You may have to hit ENTER again to refresh it).

Now, you can shut down your computer, go to a concert, and come back to your scientific work tomorrow. To see your plot again, *cd* to the folder where "*myplot.gp*" is, start gnuplot , and do:

```
gnuplot> load "myplot.gp"
```

This should popup the graph window with your previous plot showing.

Remember that all file or directory names must be enclosed in ‘“”‘s.

This method is used when you wish to save to gnuplot commands which produced an individual plot. If you have done a fit, the function which you have defined will be in the saved file, along with the "*best fit*" for the parameters that were found. Note that when you do a *fit*, gnuplot saves a complete log in the file "*fit.log*". Only the log of the last *fit* is saved however.

6.4.2 Saving your entire gnuplot session

As we did in HW 5, if you want to save a complete record of all the commands you have typed during your session, you can do the following:

```
gnuplot> history "mysession.gp"
```

This will create a (possibly long) file called "*mysession.gp*" with all the commands you have typed into gnuplot. If you view this file in *less*, you can read what you had been doing, and cut/paste commands into gnuplot to redo the things you want.

6.4.3 Saving a plot as an image file

Another very useful *save* method is to write the plot in a different format , for example as a **JPEG graphics file**.

Gnuplot has many different *Terminals* for which it can produce plots. Since gnuplot has been around for 30 years, several of these are outdated. However, a few are quite useful; particularly, the **PDF** "terminal" is one of these, for example.

To see all the terminal types, do

```
gnuplot>set term
```

as if you were going to set the terminal to some type. But just type ENTER instead.

This will list all the possible terminal types. I've highlighted common ones below:

```
Available terminal types:
           aed512  AED 512 Terminal
           aed767  AED 767 Terminal
             aifm  Adobe Illustrator 3.0 Format
         bitgraph  BBN Bitgraph Terminal
              cgm  Computer Graphics Metafile
            corel  EPS format for CorelDRAW
             dumb  ascii art for anything that prints text
              dxf  dxf-file for AutoCad (default size 120x80)
            eepic  EEPIC -- extended LaTeX picture environment
              emf  Enhanced Metafile format
            emtex  LaTeX picture environment with emTeX specials
         epslatex  LaTeX picture environment using graphicx package
     epson_180dpi  Epson LQ-style 180-dot per inch (24 pin) printers
      epson_60dpi  Epson-style 60-dot per inch printers
      epson_lx800  Epson LX-800, Star NL-10, NX-1000, PROPRINTER ...
              fig  FIG graphics language for XFIG graphics editor
            **gif  GIF images using libgd and TrueType fonts**
             gpic  GPIC -- Produce graphs in groff using the gpic preprocessor
          hp2623A  HP2623A and maybe others
           hp2648  HP2648 and HP2647
           hp500c  HP DeskJet 500c, [75 100 150 300] [rle tiff]
             hpdj  HP DeskJet 500, [75 100 150 300]
             hpgl  HP7475 and relatives [number of pens] [eject]
           hpljii  HP Laserjet series II, [75 100 150 300]
             hppj  HP PaintJet and HP3630 [FNT5X9 FNT9X17 FNT13X25]
           imagen  Imagen laser printer
            **jpeg  JPEG images using libgd and TrueType fonts**
        kc_tek40xx  MS-DOS Kermit Tek4010 terminal emulator - color
        km_tek40xx  MS-DOS Kermit Tek4010 terminal emulator - monochrome
             latex  LaTeX picture environment
                mf  Metafont plotting standard
               mif  Frame maker MIF 3.00 format
                mp  MetaPost plotting standard
           nec_cp6  NEC printer CP6, Epson LQ-800 [monochrome color draft]
           okidata  OKIDATA 320/321 Standard
               pbm  Portable bitmap [small medium large] [monochrome gray color]
             pcl5  HP Designjet 750C, HP Laserjet III/IV, etc. (many options)
            **pdf  PDF (Portable Document File) file driver**
```

```
      **png  PNG images using libgd and TrueType fonts**
 postscript  PostScript graphics, including EPSF embedded files (*.eps)
    pslatex  LaTeX picture environment with PostScript \specials
      pstex  plain TeX with PostScript \specials
   pstricks  LaTeX picture environment with PSTricks macros
        qms  QMS/QUIC Laser printer (also Talaris 1200 and others)
      regis  REGIS graphics language
       rgip  RGIP metafile (Uniplex). Option: fontsize (1-8)
    selanar  Selanar
      starc  Star Color Printer
        svg  W3C Scalable Vector Graphics driver
tandy_60dpi  Tandy DMP-130 series 60-dot per inch graphics
    tek40xx  Tektronix 4010 and others; most TEK emulators
    tek410x  Tektronix 4106, 4107, 4109 and 420X terminals
    texdraw  LaTeX texdraw environment
       tgif  TGIF X11 [mode] [x,y] [dashed] ["font" [fontsize]]
   tkcanvas  Tk/Tcl canvas widget [perltk] [interactive]
       tpic  TPIC -- LaTeX picture environment with tpic \specials
    uniplex  RGIP metafile (Uniplex). Option: fontsize (1-8) (Same as rgip)
    unknown  Unknown terminal type - not a plotting device
      vttek  VT-like tek40xx terminal emulator
      **x11  X11 Window System**
       xlib  X11 Window System (gnulib_x11 dump)
```

This is what you get by typing the *set term* command. To use one of these terminal types, you start gnuplot and get plot what you want.

After you have the plot looing the way you want it to, you can produce a *JPEGF image** of your plot by doing the following:

```
gnuplot> set term jpeg            # sets the terminal to JPEG
gnuplot> set output "myplot.jpg"  # sets the output to a file called "myplot.jpg"
gnuplot> replot                   # plots the JPG file to "myplot.jpg"
gnuplot> unset output             # unset file output
gnuplot> set term pop             # return terminal type to default
```

(of course you don't need to type the #comments shown above)

The last line above, **set term pop**, returns the terminal type to the "standard" one that was set when gnuplot stated.

Do this for a plot of your choice (say, *sin(x)* if you can't think of a more interesting one).

Use the *CTRL-z* trick to suspend gnuplot, and get your shell prompt back. Then use the Fedora Linux (KDE Desktop) Image Viewer, *gwenview*, to look at the image you made. At the shell prompt, type:

```
sci[gp]> gwenview myplot.jpg
```

You can now include this image in a document or webpage, or send it to a collaborator via email.

PNG and JPEG files are useful for including other documents such as HTML web pages or MS Word documents.

Another very popular file format at the moment is **PDF**.

You can make a PDF file of your plot by following the steps above, but using *pdf* instead of *jpeg*.

```
gnuplot> set term pdf
gnuplot> set output "myplot.pdf"
gnuplot> replot
gnuplot> unset output
gnuplot> set term pop
```

Be sure you can make a plot in PDF format. Then find it an view it with your PDF viewer.

You'll need to send me a PDF plot later, and we'll use JPEG plots in LaTeX.

6.4.4 Printing your Plots

The methods above are almost enough to do everything, but there are times when you just need to send your Mom a copy of some really interesting work that you've recently done, so she can put it on the 'fridge to display for her friends.

There simplest method to **Print a Plot** is to just produce a **PDF** file as described above, find the file with a PDF Reader, and print the file from the Reader.

The PDF Viewer that is built into Fedora Scientific is **okular**. At the prompt, you can type:

```
sci[gp]> okular myplot.pdf
```

to view the PDF file.

Alternatively, you can copy the file to one of your *Shared Folders*:

```
sci[gp]> cp myplopt.pdf ~/winDesktop
```

for example, to copy the file to your computer's Desktop. Then you can open the PDF file with, say *Adobe Reader* as you do with other PDF files. From there it is easy to print them.

Since *Adobe Reader* is most likely already set up to print to your default printer, this is the easiest way to get a hardcopy plot to put on your 'fridge.

6.5 Homework

Homework 6 is here

6.6 A Worked Example: Theory meets Experiment

In Homework 2 you visited Johnny Utah, a BASE jumping video, photography, and stunt company who have data collected from a long freefall. You should all have a file with this data.

I decided to check out how good this data is.

Also, the distance data is in feet and speed data in miles/hour . It would be easier to convert this to meters and m/s . We can do this by taking advantage of the **using** feature when plotting data.

To find how to convert, I will use the Unix **units** program, a wonderful little text based program that gives you conversion factors for one set of units to MANY others.

In an xterm, you should be able to type " **units**" at the prompt and get:

```
sci[~]|>units

2438 units, 71 prefixes, 32 nonlinear units

You have: feet
You want: meters
        * 0.3048
        / 3.2808399
You have: mph
You want: m/s
       * 0.44704
        / 2.2369363
```

Now, I can set two conversion factor constants in gnuplot that I will use later.

```
gnuplot> f2m = 0.3048
gnuplot> mph2mps = 0.44704
```

6.6.1 Theoretical Expectations from Physics

You probably know that for an object in free fall in the Earth's atmosphere, it begins to drop as if there is no air resistance, so the distance from the starting point increases as

$$y(t) = \frac{1}{2}gt^2$$

where g = 9.8 m/s 2 .

At first, before the object is moving very fast, this will be a good approximation. So let's check it.

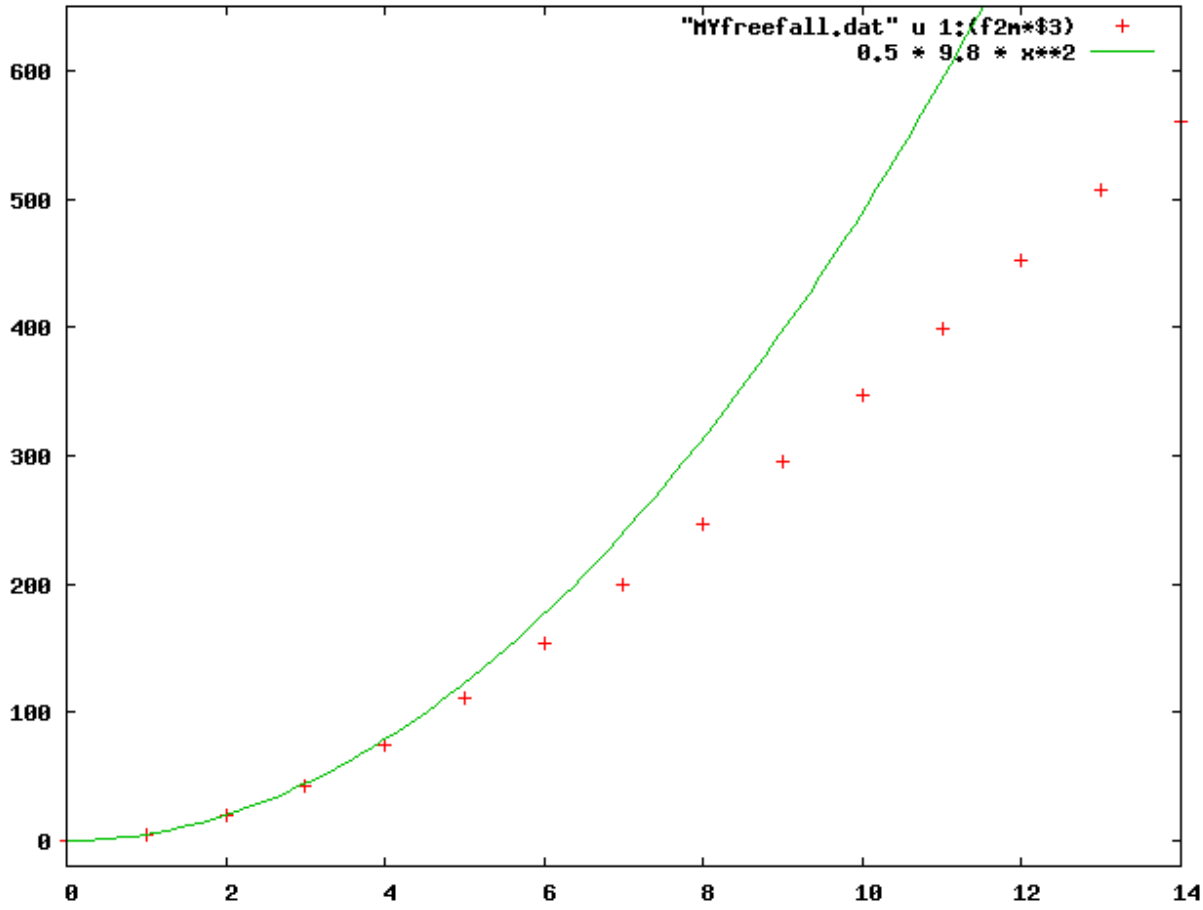
Here's your data from Homework 4.

Ok. Here's the data file:

```
1   22   16    16
2   42   46    62
3   61   76    138
4   72   104   242
5   87   124   366
6   96   138   504
7   103  148   652
8   107  156   808
9   111  163   971
10  114  167   1138
11  117  171   1309
12  119  174   1483
13  119  174   1657
14  119  174   1831
15  119  174   2005
16  119  174   2179
17  119  174   2353
18  119  174   2527
19  119  174   2701
20  119  174   2875
```

Plot it.

```
gnuplot>plot "MYfreefall.dat" u 1:(f2m*$3), 0.5 * 9.8 * x**2
```



At the beginning of the graph (for small *t*), does the function match the data? Do you understand how I converted it to meters by simply using the **using** command and the constant *f2m*? Make sure you do.

Fairly quickly, the force of air resistance becomes apparent and once the force of gravity and the force of air resistance are equal, the object will no longer accelerate (right?

$$F_{\text{gravity}} - F_{\text{air}} = 0$$

So,

$$F_{\text{total}} = 0$$

With no net force on the falling object, it will no longer *accelerate*, since $F = ma$, for those who haven't had Physics in a while).

If the *acceleration* is zero, it's speed will be constant. This speed is called *terminal velocity*. We can see that on our plot, after about 8 seconds–the position increases *linearly* (*i.e.* like a line).

Looking at your data file, the position data in column 3 seems to make sense: *quadratic* ($\sim t^2$) at the beginnig, and *linear* (:math:'sim t) at the end, as expected. We'll deal with the velocity data later.

We can estimate the terminal velocity by plotting a line along with the data and varying its slope until we get it to match the linear part of the data, after 8 sec. It helps to offset the line with a negative *b* constant (*a*x - b*).

Try to estimate the terminal velocity by plotting the data, AND a line *a*x + b*. Vary the constant *a* and *b* until the line goes though (as best you can) the points from *t >= 10*.

Now for the theory.

The governing equation here is Newton's: $F = ma$

We all know that one, right? However, What it really means is this.

where *F* is the *total force* $F_{\text{gravity}} - F_{\text{air}}$, *m* is the mass, and the acceleration, *a*, is the second derivative of position x(t) with respect to time.

$$F = ma = m\frac{d^2y}{dt^2}$$

I am being pedantic here for those who have not had calculus for some time. I will shortly get to *differential equations*– if you have not had *DiffEQs* , don't worry–this is just an example. Nonetheless, follow as much as you can.

Recall that the *force* on a falling object due to gravity is constant

$$F_{\text{gravity}} = mg$$

Now, it turns out that except for rather slow speeds, the force of air resistance can be approximated by a force which grows as the square of velocity

$$F_{\text{air}} = -kv^2$$

The constant k depends on a number of things

$$k = \frac{1}{2}\rho C A$$

where:

$$\begin{aligned}
\rho &= \text{massdensity} \\
C &=\sim 1 \quad \text{a constant related to the shape of the object} \\
A &= \text{cross sectional area perpendicular to velocity}
\end{aligned}$$

Let's simplify the differential equation:

$$F = ma$$

or

$$\begin{aligned}
ma &= F \\
m\frac{d^2y}{dt^2} &= mg - kv^2 \\
\frac{d^2y}{dt^2} &= g - \frac{k}{m}\left(\frac{dy}{dt}\right)^2 \\
\frac{d^2y}{dt^2} + \frac{k}{m}\left(\frac{dy}{dt}\right)^2 - g &= 0 \\
\frac{d^2y}{dt^2} + b\left(\frac{dy}{dt}\right)^2 - g &= 0
\end{aligned}$$

In the last differential equation, we have defined $b = k/m$. This is the one we have to solve for the function y(t) .

I will skip the solution of this equation (it is done in many books on Differential Equations, as well as many jr. level Physics texts. There is a good review that I found:Auburn Engineering Course).

The function $y(t)$ is

$$y(t) = \frac{1}{b}\log\left[\cosh(\sqrt{bg}\,t)\right]$$

where *g* is the acceleration of gravity, and *b = k/m* is as defined above. *cosh()* is the hyperbolic cosine.

See if you can check that this equation $y(t)$ satisfies the differential equation in the last line above!

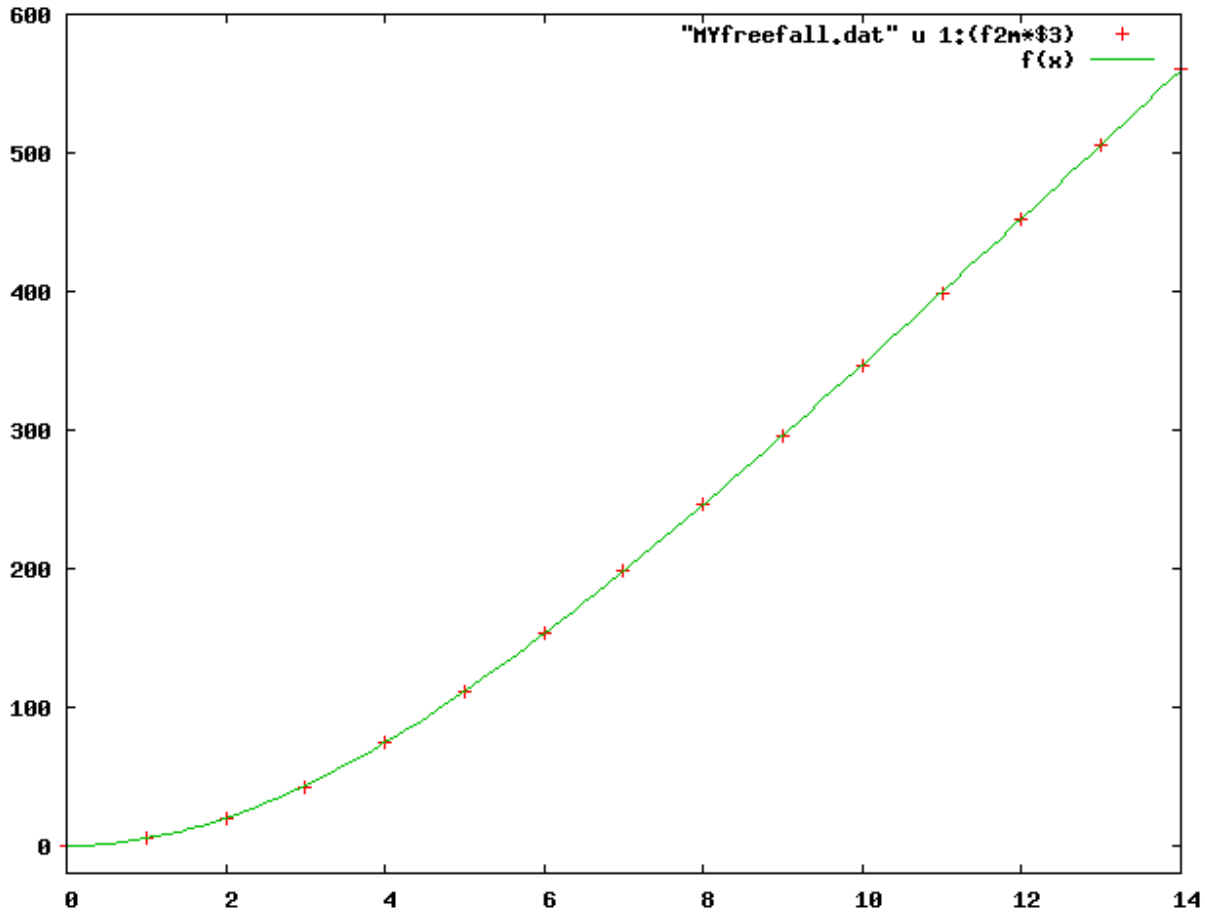
6.6.2 Theory meets Experiment

If you visit the webpage above to look at the method of solution, or even if you just verify that the solution satisfies the DiffEQ, you can see that this was a non-trivial result. How well does it match the experimental data?

All we have to do to check is to fit the solution to the data. Remember that gnuplot (almost) always uses x for the horizontal axis even though we are calling it t in our theoretical discussion. Similarly, we have called the position x(t) even though it's been plotted on the y axis. ...Just wanted to be clear.

```
gnuplot>f(x) = 1/b*log(cosh(sqrt(b*g)*x))
gnuplot> fit f(x) "MYfreefall.dat" u 1:(f2m*$3) via g, b
...
Final set of parameters            Asymptotic Standard Error
=======================            ==========================

g               = 10.0539          +/- 0.0284       (0.2824%)
b               = 0.00338874       +/- 2.414e-05    (0.7123%)
...
```

```
gnuplot>plot "MYfreefall.dat" u 1:(f2m*$3), f(x)
```



BANG!

The theoretical function is really good fit to the data. Also the fitted value for g is rather close to the known value (9.8):

```
gnuplot> print (g-9.8)/9.8 * 100
2.59118938657317
```

within about 2.6%. Notice that in the fit report above, the error on the value for g is 0.284 = 2.8%. Thus the data is consistent with the known value of g=9.8 m/s.

That's not too bad for pulling some data from an extreme sports webpage and applying the laws of physics!

6.6.3 Velocity

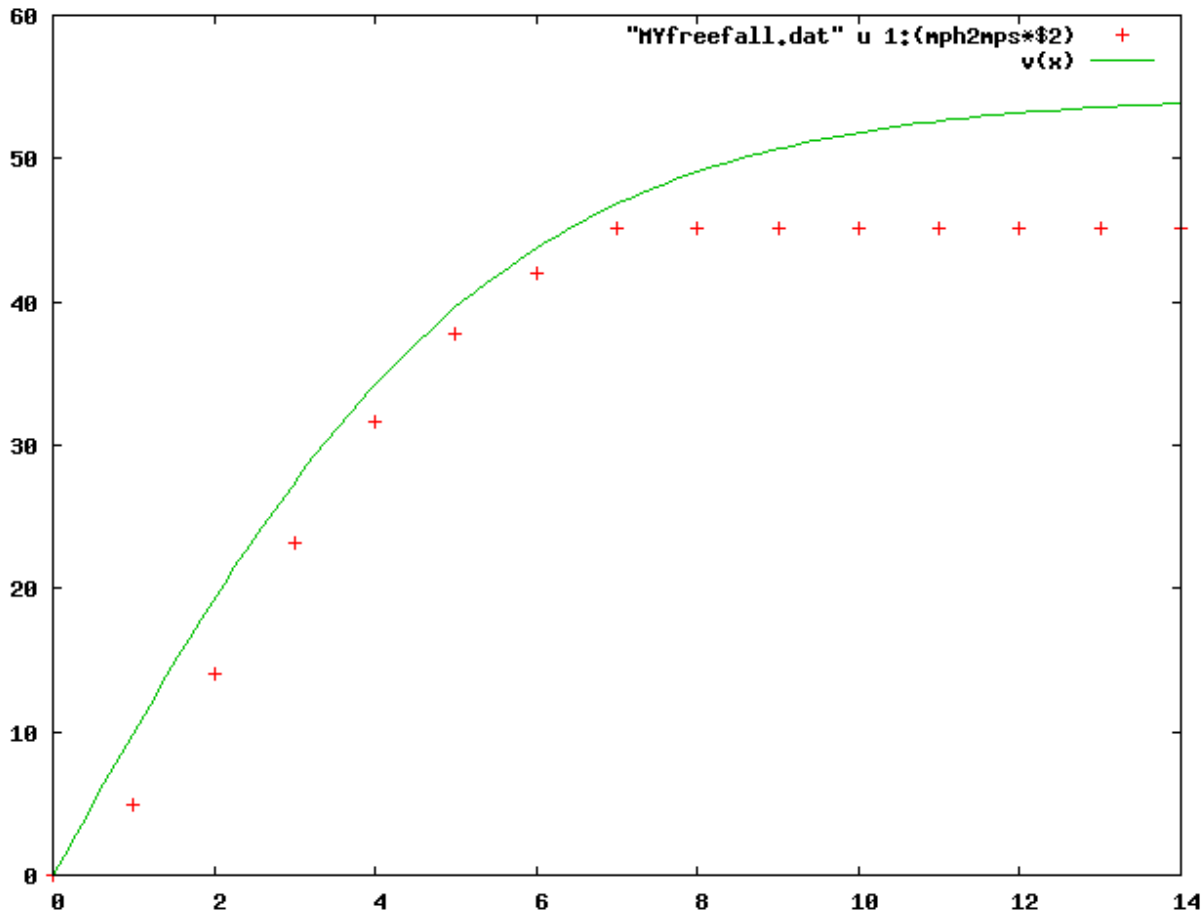
Next lets look at the velocity. Given the position, $y(t)$, the velocity is just the time derivative:

$$\begin{aligned} v(t) &= \frac{dy}{dt} \\ &= \sqrt{\frac{g}{b}} \tanh\left(\sqrt{bg}\, t\right) \end{aligned}$$

Check this. You may have to consult your math book to recall the derivative of cosh(x) .

Now we can check the velocity data against the data. Recall that the velocity data is in the stupid units: mph = miles per hour . However, our constant **mph2mps** takes care of that. So, define the function v(x) , and plot the data and theory.

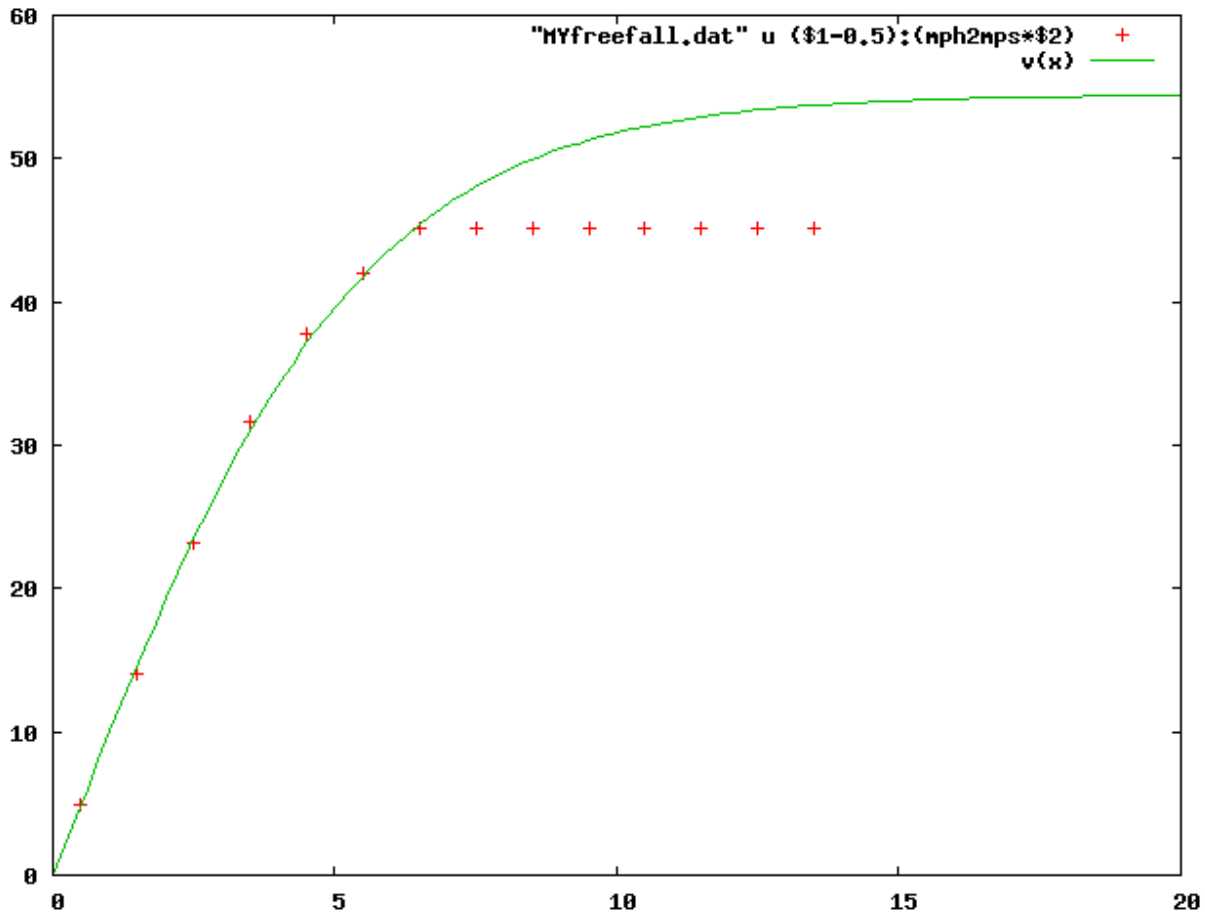
```
gnuplot>v(x) = sqrt(g/b)*sinh(sqrt(b*g)*x)/cosh(sqrt(b*g)*x)
gnuplot>plot "MYfreefall.dat" u 1:(mph2mps*$2), v(x)
```



The match between theory and experiment is not good.

The key thing to realize here is that velocities are usually measured by taking the initial and final positions over some time interval Δt, then dividing the change in position Δy by Δt to get the average speed over that interval. Thus the velocity really represents the speed closer to the middle of the time interval. The velocity recorded at, say, 3 seconds, really should be reported as the speed at 2.5 seconds , since the last measurement of position was made at 2 seconds. Does this make sense to you? In other words, the velocity data should really be shifted back (left) by 0.5 second in time. That's easy to do with **using**!

```
gnuplot>plot [0:20] "MYfreefall.dat" u ($1-0.5):(mph2mps*$2), v(x)
```



Much better.

I also added an extended plot range to the graph so we could see the approach to terminal velocity, and how long it takes to get to a constant speed.

We also see that the " **Terminal Speed**" reported in the original data is not quite correct.

The falling object has not yet obtained constant speed. You might want to plot thus using a " **using** " trick like:

```
gnuplot> u 1:($1>7 ? $2 : 1.0/0)
```

which will block plotting any points after 7 seconds. Do you see how it works?

6.6.4 Forensic Data Analysis

When you think of *forensics*, you probably think of **CSI** of a crime show. Forensic data analysis, is about reconstructing what you can about something from the data you have.

Let's see if we can figure out the **mass of the jumper**!

From our fit, we know that **b**, the coefficient of air resistance, is

$b = 0.0033$

Hence, we can estimate the parameters in this equation. The density of air, ρ, is about $1\ kg/m\ 3$. The "Shape Coefficient", C , for a prone falling human is around 1 (it's probably slightly higher but to our level of accuracy, 1 will do; see the article on Drag Coefficients at Wikipedia).

The surface of a falling person is also hard to estimate, but let's say a 6 foot person (~1.8 m) times a width of half a meter. That's $0.9m^2$.

That means that the mass of our falling person is about

$$
\begin{aligned}
b = k/m &= 0.0033 \\
\frac{k}{0.0033} &= m \\
m &= \frac{1}{0.0033}\frac{1}{2}\rho C A \\
m &= 136 \text{ kg} = 300 \text{ lbs}
\end{aligned}
$$

That seems a bit high, but it could easily be

- a 220 lb man*, with a 50 lb parachute + reserve chute + 30 lbs of

camera and position/speed measuring gear. or + a 180 lb instructor + 100 lb student + 20 lb of gear

3: :///home/jhetrick/www/phys27/pages/gnuplot/solnHW3.html .. _Auburn Engineering Course: http://www.eng.auburn.edu/department/me/courses/nmadsen/egr182a/drag01.html .. _Wikipedia: http://en.wikipedia.org/wiki/Drag_coefficient#Cd_in_other_shapes .. _Vertical Visions: http://www.vertical-visions.com/ .. _Homework 2: :///home/jhetrick/www/phys27/pages/gnuplot/../editor/hw2.html

CHAPTER

SEVEN

SCIENTIFIC PUBLICATION

7.1 Scientific Publications

While you have probably learned how to use *Microsoft's Office Suite*, including *MS Word* and *Powerpoint*, these tools are not very well suited to presentations which involve a significant mathematical exposition. Not only is it difficult to create equations, they look lame in *MS Word*.

That's why professional mathemeticians, physicists, chemists, computer scientists, economists, engineers, and others use **Latex** (pronounced " LAY' tek , not " Laytecs " like the gloves).

An example of this powerful desktop publishing software is shown here:

$$\left\langle \Phi_s(\theta';\varphi'_1,\cdots,\varphi'_{N-1})\middle|H_{\rm mass}\middle|\Phi_s(\theta;\varphi_1,\cdots,\varphi_{N-1})\right\rangle$$

$$= -\delta_{2\pi}(\theta'-\theta)\delta_{2\pi}(\varphi'_1-\varphi_1)\cdots\delta_{2\pi}(\varphi'_{N-1}-\varphi_{N-1})$$

$$\times\sum_{a=1}^{N} m_a A_a\Big\{\ e^{-i\hat{\varphi}_a+i\delta_a}+e^{+i\hat{\varphi}_a-i\delta_a}\ \Big\}$$

$$A_a = e^{-2\pi/N^2\mu L}\prod_{\alpha=1}^{N} B(\mu_\alpha L)^{U_{\alpha a}{}^2}\quad ,$$

This is an equation from a paper of mine.

Have a look at this paper by clicking on the preceeding link. Once on the abstract page, click on the PDF link which is in the upper right corner of that page. Scroll through the paper. This paper was produced with **Latex** and **Gnuplot** and sent to the publisher in this form, ready for publication. Imagine how impressive your lab report would be if it looked like this!

You can have a look at some other examples of Latex in action by visiting the " Preprint Archive " at Cornell University. Started by particle physicists in the late 1980s, this repository is where we send copies of our papers at the same time we send them off to the publisher. That way the paper, and more importantly its results, are available to everyone with internet access.

The archive is here: http://www.arxiv.org/

For some examples of Latex output, scroll down to my field: **High Energy Physics - Lattice** which is where papers on large scale simulations of particle theory go. (the name " Lattice " comes from the fact that we represent spacetime as a 4-dimenional "lattice" of points instead of the usual "spacetime continuum". That way we can simulate the quantum behavior of fields). Have a look at the New listings to see the latest papers posted. This is where almost all new results in physics appear first. Click on any abstract that interests you, then on the PDF link. That should pop the paper up on your screen. Look at the gorgeous typesetting of the document.

As long as you are still visiting the archive, have a look around. The new papers in " String Theory " usually appear on **High Energy: Theory** while new experimental results show up in **High Energy: Phenomenology**. There's an **Astrophysics** branch that includes both new theoy and experimental results, as well as a whole separate branch for **Mathematics**. Or, check out Artificial Intelligence in the **Computer Science** branch. If you are a scientist, start getting used to reading!

Won't it be cool to hand in your next lab report looking polished and professional like one of these research papers?

7.1.1 LaTeX

Again, **latex** is pronounced: LAY' tek , (not lay . tecs like the material). You can read why by googling latex. It was invented by Donald Knuth in the 1980s.

One of the reasons that I have decided to switch to using *Fedora Scientific* is that **latex** comes completely installed if you choose it as a package. That saves a lot of hassle installing it as a standalone package. You can verify that latex was correctly installed by openning a *Terminal* window and typing

sci[~]> latex -v

(-v stands for "show version").

This should display something like this:

```
sci[~]>latex -v
TeX (Web2C 7.4.5) 3.14159
kpathsea version 3.4.5
Copyright (C) 1997-2003 D.E. Knuth.
Kpathsea is copyright (C) 1997-2003 Free Software Foundation, Inc.
There is NO warranty.  Redistribution of this software is
covered by the terms of both the TeX copyright and
the GNU General Public License.
For more information about these matters, see the files
named COPYING and the TeX source.
Primary author of TeX: D.E. Knuth.
Kpathsea written by Karl Berry and others.
```

If you see an error message, email me and we'll sort it out.

7.1.2 Getting Started

Latex is a kind of programming language. What you actually do is write a little program (using your text editor) which gets compiled into a graphical format–we'll use PDF, but another format that was common is DVI, DeVice Independent format.

The program that you write is mostly composed of the text of your document, but in addition contains key words which tell latex to make certain symbols, and take certain lexigraphical actions (superscripting, or bolding, or creating a new section in your paper).

So, you first need a basic latex document.

Open an *Terminal Shell* window .

Make a new subdirectory (below your **home/**) called **latex/** .

Now cd to this directory .

I want to show you a unix "*trick*" (it's actually just a standard unix command).

In the terminal window type *emacs*. This should open an emacs editor on your desktop. Now try to type something in the xterm window . It doesn't respond, right?

That's because when the **emacs** application starts, it takes control of the xterm's prompt–the xterm will wait until the emacs application exits before responding to input.

Exit emacs

Now, start emacs this way:

emacs &

by adding an ampersand "**&**" after the command. This tells the xterm to " start emacs in the background ", detaching it from the current xterm operation. Notice that emacs starts, but you now have your xterm prompt back and can issue commands. Make sure that you can now type commands (like **ls**) at the xterm prompt .

In the open emacs editor, copy the text below and save this file as **tiny.tex** .

```
\documentclass{article}
\begin{document}
This is my first latex document.

Oh yeah,
\em{Physics is the Mother of all Sciences}

\end{document}
```

At your terminal shell prompt (the one that is now active because we detached emacs with the " **&**" above), type:

pdflatex tiny.tex

This will create a file called ** tiny.pdf ** in your **latex** folder. Do an **ls** to see if it's there.

The *built-in PDF Document* in the Fedora Scientific environment is called: **okular**

While still in your **latex** directory, do an listing, and see if **tiny.tex** and **tiny.pdf** files are there.

Now type

okular tiny.pdf

The viewer should open it up so that you can see your new document.

It's tiny, but it's latex document.

7.1.3 Basic Latex Document Structure

Notice what you did above.

- You **wrote tiny.tex in plain text** using emacs (or any text editor) which is something like a program containing your text, plus special commands for formating (the words enclosed in *em{ ...}* are em phasized, for example). You can also use the form: *{em ...}*. Later we'll use special commands for math symbols and more impressive formatting.

- Then you **compiled** the document into a particular output format: PDF in this case, by running **pdflatex tiny.tex**.
- Finally, you **used the PDF viewer**, *okular*, to review your work. At this point you have a document that you could transmit to anyone with a PDF viewer

Let's look at the program part:

The Latex Document

Latex commands begin with a **backslash: **, followed by a keyword.

Any other word in a latex document is considered *text to be typeset*.

```
\documentclass{article}
\begin{document}
This is my first latex document.

Oh yeah,
\em{Physics is the Mother of all Sciences}

\end{document}
```

7.1.4 The (Minimal) Structure of a LateX Document

At the very least, a latex document must have the following features. It must start with the keyword

\documentclass

followed by arguments. We will usually use the *article* class, the document begins

\documentclass{article}

```
\documentclass{article}
\begin{document}
This is my first latex document.

Oh yeah,
\em{Physics is the Mother of all Sciences}

\end{document}
```

but there are **book** and other classes that we could have used instead of *article*.

Following this, we might add some further things to set up the document (including packages, making definitions, for instance). These are "setup" things, not part of the document text itself.

\begin{document}

```
\documentclass{article}
\begin{document}
This is my first latex document.

Oh yeah,
\em{Physics is the Mother of all Sciences}

\end{document}
```

Simple enough, right? From here, everything either gets printed or actions are taken on latex keywords (like \em{} below).

```
\documentclass{article}
\begin{document}

This is my first latex document.

Oh yeah,
\em{Physics is the Mother of all Sciences}

\end{document}
```

Finally, at the end we put

\end{document}

```
\documentclass{article}
\begin{document}
This is my first latex document.

Oh yeah,
\em{Physics is the Mother of all Sciences}

 \end{document}
```

A bit more interesting document

If all went well above, try the file below. If things DIDN'T go well, email your file to me and tell me what happened.

Let's make a file that is a bit more interesting. First, a useful thing to remember is that most of us who know latex, learned it by copying things out of other files we were shown (or found).

So, you can do the same with these examples I'm giving you. You'll need the text below.

You can also just download this file first.tex and save it in your **latex/** directory, or copy and paste it into a file (called **first.tex**) using emacs.

```
% first.tex
% My first real Latex document

\documentclass[12pt]{article}

\begin{document}
```

```
\title{My First LaTeX Document}

\author{
Jane R. Feynman \\
Department of Physics\\
University of the Pacific \\
3601 Pacific Ave., Stockton, CA 95211
}
\maketitle


\begin{abstract}
In the abstract section you would put a brief (one paragraph)
description of your project or experiment, and the main results.
\end{abstract}


\section{Introduction}
% LaTeX comments begin with a percent sign, and are not printed.

In the Introduction you would put your introductory remarks. You will
introduce the project and give the reader some context for your study.

\subsection{Subsection: Math}

Including mathematics is easy.
$$
\int_{-a}^{\infty} f(x) dx= F(a)
$$
or you can do it like this for a numbered equation:

\begin{equation}
\oint_C \frac{dz}{z-z_0}=2\pi i
\end{equation}

\begin{equation}
\sum_{i=1}^\infty \frac{1}{n^2} = \frac{\pi^2}{6}
\end{equation}

A numbered equation can be referred to or cited by giving it a label.
\begin{equation}\label{eq:pot}
V(r)={\hbar^2 \ell(\ell+1)\over 2mr^2}-{\hbar c \alpha\over r}
\end{equation}

Somewhere in your text you may wish to refer to this equation. You don't
need to know what Eq. number it is, just its label. Refer to it with
the command Eq.~(\ref{eq:pot}) and LaTeX automatically uses the correct
equation number, even if you go back and add more equations later. You
will need to run latex twice, once to compute the equation numbers.


\section{Tables}

Suppose that you want to create a table showing your data. The syntax
for a multicolumn table is found below.
\vspace{1cm}

\begin{tabular}{|c|c|c|c|}
```

```
\hline
Run 1 & Output & Run 2 & Output \\
\hline
1 & 37 & 1 & 28 \\
\hline
2 & 42 & 2 & 35 \\
\hline
3 & 73 & 3 & 19 \\
\hline
4 & 15 & 4 & 22 \\
\hline
\end{tabular}


\section{Conclusions}

In this section draw your conclusions and summarize your findings.
Certainly from this short and simple document, we can see that LaTeX
is tremendously powerful, easy, and FUN! There are many online
tutorials and help pages on LaTeX. Simply enter {\tt latex tutorial}
or {\tt introduction to latex} into Google and you will find lots of
help.

\end{document}
```

Try to compile and view this file in the same way as the one above . You should create the file on your computer, but if you have problems and want to see the correct version, the PDF output is here.

Have a look at the PDF file. Notice the different types of things contained:

- A title page and abstract
- Sections and subsections
- Mathematical typesetting, including numbered equations
- Tables
- Different fonts and text sizes

Now edit the file.

- Change the author and address your name and address .
- Change the title to **My Own Latex Document** .
- Change the Abstract to say: In this report, I will prove that Physics is the Mother of all Sciences. .

7.2 Latex Document Environments and Layout

By now you should be able to create PDF files of the sample latex documents shown in the last section. If not, contact me immediately for help.

Next we need to learn about common structures that you will want to use in your latex documents. There are many of these, for which there is much freely available literature on the web. However We will focus on the following set since these are plenty to get you started making really nice reports. The basic structures are:

- Environments or Modes, of which the following are special examples
  - Tables

  - Math modes
  - Figures
  - Verbatim mode
- Layout
- Bibliography

Finally, we need to learn how to troubleshoot (i.e. Fix our files) from the output of **pdflatex**. This can be kind of obscure, but is common to anyone who has done some programming.

Rather than transcribe all of this to HTML for display in your browser, I've put the discussion in a Latex file, which you can read here. You'll likely want to open the PDF document, **second.pdf** ("Open Link in New Window") and simultaneously follow along looking at the source file, **second.tex** the text document that made latex turned into second.pdf.

- second.pdf
- The LateX source code: second.tex).
- s8.png Graphics file needed to compile **second.tex**

Please read **second.pdf** and have a look at the Latex source, then do Homework 7

For convenience, second.pdf is reproduced below. However the PDF file is much clearer.

Our Second Latex Document

Capt. James T. Kirk

Physics Department
3601 Pacific Ave.

University of the Pacific
Stockton, CA 95211

Abstract

The purpose of this report is to introduce you to a few more aspects of Latex and get you ready to produce high quality scientific documents.

Introduction

Now that you are familiar with Latex, we will do some more fancy things.

First, notice the format of this document. The two column format was made by adding the option `twocolumn` in the brackets in the `documentclass` declaration. There are four main document classes: *article, report, letter*, and *book*. Each has special commands (lots for book!). In addition publishers oftern specify a class for their journal and supply you with a `.sty` file which formats your document for their journal. One day, we'll have a Pacific Physics .sty file for our lab reports!

Also, if you are following along in the Latex source file `second.tex`, then you will see many lines with % at the beginning. These are latex *comments*. Similar to the gnuplot comment # character, latex uses the percent sign: %. To put a real percent sign in your document, use backslash percent: \%.

You may have noticed that I can change the font to emphasize words, like `this` or *that*, or even this and this!. All we need to do is enclose the word in {}'s and include the appropriate font type at the beginning. You can change

	{\em this}	*this*
	{\bf this}	**this**
fonts like this	{\tt this}	`this`
	{\sf this}	this
	{\underline {this}}	this

You can also do \textbf{this} to get **this**, etc.

Environments and Modes

To do special kinds of presentation, like *tables*, or *mathematics* (among many others), Latex uses *environments* (which we sometimes call *modes*, as in "math mode").

The way this works is that at some point in your document, you enter a new mode by typing

```
\begin{figure}
...stuff for figure mode goes here...
\end{figure}
```

as shown here for the `figure` mode. This is the environment you would use to set aside some space for inclusion of a figure. Another common mode is the `tabular` mode for making tables, which we'll look at below.

Tables

Let's look at one of the common modes: `tabular` for making tables.

In the previous document, `first.tex`, we made a simple table with several rows and columns. You can certainly make much more complicated tables, and below we make a simple change with some entries that span two columns.

Suppose that you want to create the following table showing your data, but you need some "cells" to span 2 columns (Trial 1 and Trial 2 below).

Trial 1		**Trial 2**	
Bin	Number	Bin	Number
1	3	5	9
2	4	6	3
3	7	7	1
4	1	8	2

This table was created with the following code:

```
\begin{center}
```

```
\begin{tabular}{|c|c|c|c|}\hline
\multicolumn{2}{|c|}{\bf Trial 1}
  & \multicolumn{2}{|c|}{\bf Trial 2}\\
\hline
Bin & Number & Bin & Number \\
\hline
1 & 3 & 5 & 9 \\
\hline
2 & 4 & 6 & 3 \\
\hline
3 & 7 & 7 & 1 \\
\hline
4 & 1 & 8 & 2 \\
\hline
\end{tabular}
\end{center}
```

The line:

```
\begin\{tabular\}\{|c|c|c|c|\}\hline}
```

enters *tabular mode* and sets up the table, with 4 columns, separated by vertical lines, and column entries center justified: this is what `|c|c|c|c|` does.

The `\hline` makes a horizontal line at the top of the table.

Below this line is the first table entry with some special commands:

```
\multicolumn{2}{|c|}{\bf Trial} &
      \multicolumn{2}{|c|}{\bf Trial 2}}\\
```

This defines a row with 2 columns that each span 2 of the previously defined columns. The argument 2 to `multicolumn` tells Latex how many columns to span by this entry. An `\hline` causes a horizontal line to be added to the table after this line.

Following this come the "normal" 4-column table entries:

```
1 & 3 & 5 & 9 \\
\hline
```

Notice that an ampersand (&) is used to separate the table fields and a LaTeX newline, \\, ends each line in the table. The \hline puts a horizontal line between table rows. Take it out and see the difference.

Try changing the \begin{table} line and see what happens. For example, what if you remove the vertical bars from the |c|c|c|c|? What if you only have some: |cc|cc|? How about |l|c|c|r|?

Also, I put a \begin{center} \end{center} pair around the table so that it will be centered in the text column.

Doing the Math

There are several ways to handle mathematics[1] in LaTeX.

You can slip in a few mathematical characters **in a line of text**, such as "it follows that $\int \cos(x)\ dx = \sin(x) + C$ ", or you can focus on multiple equations or formulas that you want to stand out, such as

$$\begin{aligned}
\int_0^{\pi/3} \cos(x)\ dx &= \sin(x)\big|_{x=0}^{\pi/3} \\
&= \sin\left(\frac{\pi}{3}\right) - \sin(0) \\
&= \frac{\sqrt{3}}{2} - 0 \\
&= \frac{\sqrt{3}}{2}
\end{aligned}$$

In either condition, you must be in **Math Mode** in LaTeX to accomplish these tasks.

To enter math mode within the text of a paragraph (sometimes referred to as "inline" or "in-text"), you could use:

\begin{math} ···your math here··· \end{math}

However it's usually much easier to use the shorthand notation of two $ signs (the first $ begins math more and the second $ leaves math mode),

[1] Adapted from notes by Dr. Erin McNelis, Penn State University.

which is completely equivalent to the above longer method, as shown below:

$ ···your math here··· $

The other math mode is its own environment sometimes called "display mode" since it sets the mathematics apart, centered, on the page. There are several ways again to enter math display mode:

For a *single* equation

```
\begin{displaymath} ··· \end{displaymath}
\begin{equation} ··· \end{equation}
\begin{eqnarray} ··· \end{eqnarray}
$$ ··· $$
```

are all equivalent. The last is my favorite since it's the fewest characters to type. An example is:

```
$$
\int x^n e^{ax^{n+1}} dx
     = \frac{ e^{ax^{n+1}} }{ a(n+1) }
$$
```

which gives this

$$\int x^n e^{ax^{n+1}} dx = \frac{e^{ax^{n+1}}}{a(n+1)}$$

in your document.

Note that you can use `eqnarray` even if you have only one equation (more on this mode below).

Common Structures in Math Mode

Subscripts and Superscripts

Subscripts and superscripts are indicated by use of the _ and ^ commands immediately following the item with this sub- or superscript in LaTeX. If the elements in the subscript or superscript are *more than one character long*, they must be placed in curly braces, {···}. If something has both a subscript and a superscript, the order in which you use the commands does

not matter. Here are some examples of the displayed text and the LaTeX code to generate them:

x^2	`x^2`
x^{5y}	`x^{5y}`
x^{5y^2}	`x^{5y^{2}}`
x_2	`x_2`
x_2^{5y}	`x^{5y}_{2}`
x_{5y}	`x_{5y}`
x^{5y_2}	`x^{5y_{2}}`
$x_1^{5y_2}$	`x^{5y_2}_{1}`

NOTE: It does not hurt, and it is good practice, to put single character sub- or superscripts inside the curly braces $\{\cdots\}$ themselves.

Fractions

You may still write fractions as a/b using the / key, but most fractions are better presented in a vertical format, with the numerator physically over the denominator. To do this in LaTeX, use the `\frac{}{}` command (this command ONLY works when you are IN MATH MODE). This command has two arguments, each in its own set of curly braces, $\{\cdots\}$. First is the numerator, the second is the denominator. Here are some examples using the `\frac` command.

This one is in display math mode:

$$f(x) = \frac{x^2 + x - 2}{\sqrt{x^2 + y^2}}$$

The code which produced it is here.

```
$$
f(x) = \frac{x^2 + x - 2}{\sqrt{x^2 + y^2}}
$$
```

Using the `\frac` command from *inline* math mode as shown here
"here is an inline fraction: `$\frac{a}{2\pi}$` in the middle of a sentence",
produces this: "*here is an inline fraction:* $\frac{a}{2\pi}$ *in the middle of a sentence*".

Roots

The \sqrt command is used to get square roots as well as other roots. To get a square root, simply use the \sqrt{stuff under the radical} syntax. To have an n^{th} root, put the value of the n in square brackets, [n], between the \sqrt and the {···}. Some examples:

Here is the square root of 7: $\sqrt{7}$, and here is the 5^{th} root of x^2+7: $\sqrt[5]{x^2+7}$

This was produced with

```
Here is the square root of 7:~ $\sqrt{7}$, and here is the $5^{th}$
root of $x^2 + 7$:~ $\sqrt[5]{x^2+7}$
```

Notice in the "source file: second.tex", that I've also used the ~ character in these lines. Putting a ~ in your latex file inserts about one character's worth of blank space at that location.

Ellipsis

This could also be entitled the "dot dot dot" section. There are two types of horizontal ellipsis, the "low" ellipsis made by \ldots command and the "centered" ellipsis made by the \cdots command. The ellipsis can be vertical as well as diagonal too. See:

$x_1, \ldots, x_n$	$x_1, \ldots, x_n$
$a + \cdots + d$	$a + \cdots d$
$\vdots$	\vdots
$\ddots$	\ddots

The mbox Environment in Math Mode

All text in math mode is italicized, unless it is a special math term such as $\cos(x)$ or $\det(A)$, were these functions have their own LaTeX commands (see next section). If you want to have text presented in the normal typeset while you're in math mode, simply include that text within the mbox environment as such \mbox{ ··· }. Here's an example. The following snippet:

$$\det A = \sum_{j=1}^{n} a_{ij} A_{ij} \qquad \text{for any } i = 1, 2, \cdots n$$

is generated by typing

```
\begin{eqnarray*}
\det A = \sum_{j=1}^n a_{ij} A_{ij}
      \hspace{0.25in}
\mbox{ for any } i = 1, 2, \cdots n
\end{eqnarray*}
```

Note that the * at the end of the word `eqnarray*` turns OFF equation numbering (making it equivalent to $$), and `\hspace{0.25in}` moves things 1/4 of an inch horizontally.

Special Mathematical Formulas in LaTeX

Here is a small list of some useful special mathematical formulas that can be helpful in Math mode:

$\cos(x)$	`\cos(x)`
$\sin(x)$	`\sin(x)`
$\tan(x)$	`\tan(x)`
$\cot(x)$	`\cot(x)`
$\arccos(x)$	`\arccos(x)`
$\det(A)$	`\det(A)`
$\lim_{x \to 0}$	`\lim_{x\rightarrow 0}`
$\ln(x)$	`\ln(x)`
$\log(x)$	`\log(x)`
$\max_{t \in [a,b]}$	`\max_{t\in[a,b]}`
$\min_{t \in [a,b]}$	`\min_{t\in[a,b]}`

Special Mathematical Symbols in LaTeX

There are MANY math symbols such as α, $\int$, ζ and many more.

As these are too numerous to type at the moment, please see the handout of LaTeX math for the most commonly used mathematical symbols and formulas.

Look in Course Materials—Latex for the file `mathcheatsheet.pdf` to find a list of common math symbols.

The Eqnarray Environment

The `eqnarray` environment is similar to a table in the fact that it defaults to having three columns whose entries are to be separated by &'s. One column is for the left hand side of an equation, one column for the equal or relation sign, and the last column for the right hand side of the equation. Remember, that if you do NOT want to produce numbers next to each equation, you simply put a * after the `eqnarray`. Here are a couple of examples:

```
\begin{eqnarray*}
\int_0^{\pi/3} \cos(x) \; dx & = &
    \sin(x) |_{x=0}^{\pi/3}\\
 & = & \sin \left(\frac{\pi}{3}\right)
        - \sin(0) \\
 & = & \frac{\sqrt{3}}{2} - 0 \\
 & = & \frac{\sqrt{3}}{2}
 \end{eqnarray*}
```

$$\begin{aligned}
\int_0^{\pi/3} \cos(x)\, dx &= \sin(x)|_{x=0}^{\pi/3} \\
&= \sin\left(\frac{\pi}{3}\right) - \sin(0) \\
&= \frac{\sqrt{3}}{2} - 0 \\
&= \frac{\sqrt{3}}{2}
\end{aligned}$$

Notice in this example, that we are carrying down the operations on the right, so there is no need for a left hand side of the equation after the first line. Thus, no text preceded the & that separated the first from the second column. As in the `tabular` mode, each line ends with \\ (a newline) except the last one.

The Array Environment

On occasions when you would like to align an equation or elements in an equation with more than the three allotted columns of the `eqnarray` environment, you may use the `array` environment within any of the centered display math mode environments. This essentially gives you a table setting for mathematical terms.

Just as with the tables, you must indicate the number of columns and the alignment of the columns immediately after starting the `array` environment. Here are a few samples.

```
\begin{eqnarray*}
\begin{array}{rcrcrcr}
2x_1 & + & 3x_2 & - & 5x_3 & = & 7\\
3x_1 &   &      & - & x_3 & = & -2\\
     &   & 4x_2 & + & 2x_3 & = & 5\\
\end{array}
\end{eqnarray*}
```

$$\begin{array}{rcrcrcr}
2x_1 & + & 3x_2 & - & 5x_3 & = & 7\\
3x_1 &   &   & - & x_3 & = & -2\\
     &   & 4x_2 & + & 2x_3 & = & 5\\
\end{array}$$

This is a good way to display matrices:

```
\begin{eqnarray*}
A = \left[
\begin{array}{rrrr}
 3 &  4 & -1 &  2 \\
-2 &  3 &  5 &  7 \\
 6 & -5 & -3 & 11 \\
 1 &  2 &  6 & -3 \\
\end{array}
\right]
\end{eqnarray*}
```

$$A = \begin{bmatrix} 3 & 4 & -1 & 2 \\ -2 & 3 & 5 & 7 \\ 6 & -5 & -3 & 11 \\ 1 & 2 & 6 & -3 \end{bmatrix}$$

Use of \left and \right in Math Mode

You may have noticed that we introduced some new notation in the previous example as well as the definite integral example earlier. In order to make the square brackets for our matrix (or our parenthesis for our tall fraction) match the size of the thing they were delimiting, we needed the use of the `\left` and `\right` commands. Essentially, if you need delimiters such as the ( )'s, [ ]'s, or { }'s to stretch to fit around the object they're delimiting, you need to put a `\left` immediately before the opening delimiter and a `\right` immediately before the closing delimiter. This is also how we handle piecewise defined functions. There is a catch though. The `\left` and `\right` come in pairs and may never be stranded alone. With piecewise defined functions we want only one of the { } pairing, so there's essentially no right delimiter. In these cases, simply use a period as the missing delimiter. See the following example for help:

```
\begin{eqnarray*}
f(x) = \left\{
\begin{array}{cl}
x^2 - 2x & \mbox{ if } x < 2 \\
x - 2 & \mbox{ if } 2 \leq x \leq 6 \\
\sqrt{4x} & \mbox{ if } x > 6
\end{array}
\right.
\end{eqnarray*}
```

which produces this:

$$f(x) = \begin{cases} x^2 - 2x & \text{if } x < 2 \\ x - 2 & \text{if } 2 \leq x \leq 6 \\ \sqrt{4x} & \text{if } x > 6 \end{cases}$$

Figures

Often you will want to include a figure such as a graph of data showing theoretical fits. First creat the figure using gnuplot or MATLAB. Then include the graph in your report.

To do so, you need to "*use the package* `graphicx`". You do this by including the line:

```
\usepackage{graphicx}
```

somewhere between

`\documentclass{...}`

and

`\begin{document}`.

Look at the source file for this document. You will find the `\usepackage{graphicx}` line near the beginning. This is an example of one of many packages for latex.

Graphs are now easy to include by using the `\includegraphics` command anywhere in the document. This keyword takes an argument to set the size and then the filename of the image. The particular image used here was of course created in Gnuplot, by setting the terminal type to type PNG: (`set term png`).

The `\includegraphics` command can display images which are in any of these formats:

- PDF
- PNG
- JPEG
- GIF (only with the latest versions of pdflatex)

Latex will put your figure where, in its own calculations, it can best fit your figure. Sometimes this is on the next page or the bottom or top of the current page. It can be frustrating... Notice the line `\begin{figure}[htb]`.

The options `[htb]` indicate to Latex to try to put the figure *Here*, then the *Top*, or the *Bottom* of the page, in that order.

The figure below was produced with the code

```
\begin{figure}[htb]
\includegraphics[width=12cm]{s8.png}
\caption{This is the graph of the Hydrogen 1420 MHz
line from {\bf S8}, recorded by the Pacific Physics
Dept's Radio Telescope}
\label{fig:S8}
\end{figure}
\end{center}
```

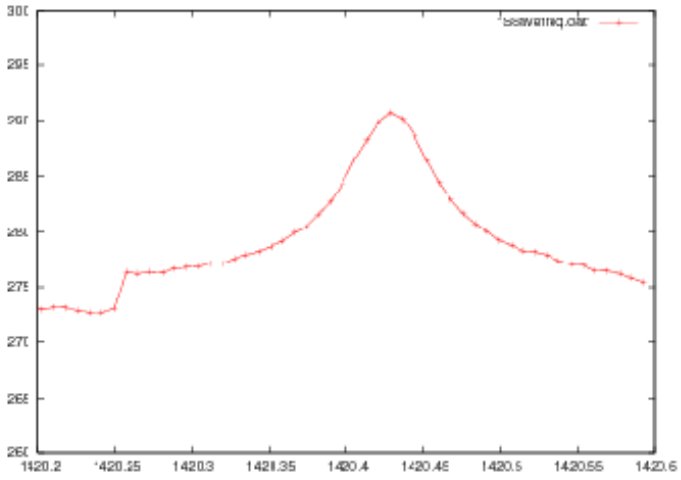


Figure 1: This is the graph of the Hydrogen 1420 MHz line from S8 recorded by the Pacific Physics Dept's Radio Telescope

Notice that `\includegraphics` command can take optional arguments such as `[width=12cm]` which sets the width of the figure on the page. You can

use centimeters (cm), millimeters (mm) or inches (in), as well as several other units. Separate options are separated by a comma as in:

```
\includegraphics[angle=45,width=52mm,scale=0.75]{myfig.jpg}
```

followed by the name of the image file enclosed in curly {} braces.

The `\includegraphics` command is usually encased in a `\figure` environment which sets off the figure by itself and provides the `\caption{}` command and the `\label{}` command.

The `\label{}` command allows you to give the figure an easy to remember nickname, which can be referred to anywhere in your document (like equation references). Just put the following line

```
...in Figure \ref{fig:S8}, we see...
```

which produces:

...in Figure 1, we see...

If you look back to the page where the figure was placed, you see that latex automatically labelled it as **Figure 1**.

The label can be set of characters. Here I used `fig:S8`; I like the `fig:` in front because it helps me remember that it's a figure. You could similarly use `eq:Newton` for your equation labels. Recall that we learned about equation labels in the last tutorial.

Remember that with all references, you have to run `pdflatex` twice to get figure and equation references correct. This is because on the first pass, pdflatex produces a file called `myfile.aux` with all the reference page numbers and locations. The second time `pdflatex` reads this file and uses it for the final version.

Verbatim Environment

Another useful environment is the `verbatim` mode, in which latex types things as they appear, without reformating spaces, etc. An example is below, where I have included a code snippet.

```
int update()  {
  int step, iters=0;
  double startaction,endaction,d_action();
  /* refresh the momenta */
  ranmom();
  /* do "steps" microcanonical steps"  */
  for(step=1; step <= steps; step++){
    update_u(epsilon*(0.5-nflavors1/8.0));
    clear_latvec( F_OFFSET(xxx1));
    grsource_imp( F_OFFSET(phi1), mass);
  }
  ...
```

As you can see, the text inside the `verbatim` environment is printed in `teletype` font so that it stands out, and text spacing is preserved. You enter `verbatim` mode as you do with other modes:

```
\begin{verbatim}
...your verbatim stuff here...
\end{verbatim}
```

For small amounts of verbatim text the `\verb` command is very useful. In the expression `\verb#some text#` the `#` character acts as a switch. All the text that comes between the two `#` characters is output verbatim. For example, latex commands are not interpreted (so you can put $'s &'s %'s, etc. between the #'s). Any character, which does not occur in the string `some text`, other than the `*` character, can be used as the switch. The equals sign = and the pipe | are commonly used as switches.

There is a star version `\verb*` of the of the `\verb` command. It behaves exactly the same as the `\verb` command except spaces are rendered visible. They are replaced by the ␣ character. Thus the command

```
\verb*#Here is some sample text.#
```

produces the output

```
Here␣is␣some␣sample␣text.
```

Running pdflatex

To produce this document, you run `pdflatex second.tex`. You may be surprized at some of the output:

```
Underfull \hbox (badness 6141) in paragraph at lines 422--422
[]\OT1/cmr/bx/n/10 Special Math-e-mat-i-cal For-mu-las in

Underfull \hbox (badness 10000) in paragraph at lines 445--445
[]\OT1/cmr/bx/n/10 Special Math-e-mat-i-cal Sym-bols in
[4]
```

These are warnings from pdflatex saying that it is a bit difficult to layout your document. Usually these errors are not a problem, as long as you get

```
Output written on second.pdf (7 pages, 170908 bytes).
```

at the end. If you *DO* get errors, latex will stop and say something like this:

```
! Undefined control sequence.
l.710 \ssection
                {Conclusions.}
?
```

This means that latex encountered a semi-fatal error on line 710 of your document. It is trying to tell you what the offending command is (in this case it doesn't recognize my misspelling of `ssection` with 2 "s"s.

At the `?` prompt, type `x` to exit latex and fix the error, or `r` to continue and ignore the error as best it can.

We'll learn more about deciphering latex errors later.

Odds and Ends

If you read tutorials about Latex on the web, you'll find that there are other routes to produce a final latex document. In this course, I've streamlined our discussion and only taught you the easiest:

- `pdflatex myfile.tex` → `myfile.pdf`

16

However, the original latex package used the command:

- `latex myfile.tex` → `myfile.dvi`

One would then use a *DVI Viewer*, an application that can display the DVI file (such as `xdvi`).

Finally, one ran the `dvips` program:

- `dvips myfile.dvi` → `myfile.ps`

which produced a *Postscript* file that could be sent to the printer (Postscript is the predecessor to PDF). There are also programs like `dvi2pdf` and `ps2pdf` which convert Postscript to PDF. (you could also use `convert`—see below!) If you use this route, with the `latex` command, then `\includegraphics` will only be compatible with Postscript (or PS or EPS) figures. If this all sounds confusing, don't worry about it. Just use `pdflatex` as I've shown you.

Since PDF has become a very popular standard for documents, and `pdflatex` does everything in one go (or maybe two if you have references), I've decided to teach you that method.

Convert

As part of the installation, I had you get the *ImageMagick* program `convert`. If you have a file in a non-pdflatex-compliant format, you can easily convert it to one of the above, by typing

```
convert myimage.booya myimage.jpg
```

at the xterm prompt. The first filename, `myimage.booya`, is the name of the image you have. Simply give the second filename the suffix of the format you would like, and hit enter. *Convert* will translate it for you. Since it converts to and from PDF, it can be handy to send an image file to someone who is having trouble opening it.

Conclusion

As you can see, Latex is rather powerful, especially for producing documents which contain a significant amount of mathematical formulae. Here we have just scratched the surface, but I hope you have learned enough so that you

can produce a basic, but sophisticated looking paper. From this start, you should be able to teach yourself more. There is a lot of documentation online about Latex, as well as several books.

One really good introduction to Latex is available at

`http://www.ctan.org/tex-archive/info/lshort/english/lshort.pdf`

called *A Not-So-Short Introduction to Latex2e*. I've put a link to it on our course webpage.

Be sure to get the `mathcheatsheet.pdf` that I've also put up on the course webpage. showing many of the math special symbols for Greek letters and other math symbols.

7.3 Error Messages from Latex

Latex is a wonderful typesetting program, however beginners often find it difficult because when something goes wrong, it takes some practice to learn how to read the error message. Once you get some practice, you'll quickly learn that most errors from from misspelling Latex commands or forgetting a **$** sign or **{** somewhere.

You can follow along with the discussion of latex error messages by downloading this file: fixme.tex.

Run

pdflatex fixme.tex

and you will get the errors described here.

7.3.1 An Example of a Latex Error

When you run pdflatex it outputs lots of stuff as it processes your document. Also long as it keeps going, you're probably in good shape.

If all goes well, you will get something like the following lines at the end of the processing output, indicating that you have a readable PDF document. It might still have mistakes, but at least they are not fatal to pdflatex.

```
Output written on second.pdf (8 pages, 153968 bytes).
Transcript written on second.log.
```

7.3.2 Errors

When it encounters an error that it can't understand, it will print out something like this, then stop with a **?**, waiting for input from you as to what to do.

```
Runaway argument?
{article)
! Paragraph ended before \@fileswith@ptions was complete.
<to be read again>
                   \par
l.9

?
```

What has *pdflatex* told you?

The first line is Latex's guess at what's wrong:

```
Runaway argument?
```

In this case, it thinks the problem is a "*runaway argument*".

This usually means that you forgot the matching brace ")" or **]** or **}** at the end of some option. Arguments are things that go into functions or commands, as in: "the arguments to the function **f(x,y,z)** are **x, y**, and **z**.

This is followed by the offending line of text or latex code, in this case at *line 9*: In Emacs, the *line number* should be shown in the *mode-line*, the little info bar at the bottom of your emacs window.

```
Runaway argument?
{article)
```

Yup. It's a runaway argument. Do you see it?

The opening **{** does not have a matching **}** closing it—it has a) instead.

Next comes some stuff about why it got an error which even I, after many years of using Latex, often don't really understand!

```
Runaway argument?
{article)
! Paragraph ended before \@fileswith@ptions was complete.
<to be read again>
                   \par

l.9

?
```

Then **the most important part of the error message**, the line number of the offense :

```
Runaway argument?
{article)
! Paragraph ended before \@fileswith@ptions was complete.
<to be read again>
                   \par
l.9

?
```

Even if you don't understand what the problem is, you can begin to look at this line (**l.9** or **l**ine. **9**) in your file.

Finally, at the end of the error message is the lonely **?**. This is **pdflatex** waiting for you to tell it what to do. You can see what the possible responses are by typing a ? followed by RETURN at the **?** prompt:

```
??
Type <return> to proceed, S to scroll future error messages,
R to run without stopping, Q to run quietly,
I to insert something, E to edit your file,
1 or ... or 9 to ignore the next 1 to 9 tokens of input,
H for help, X to quit.
?
```

Let's go through these:

- There's one command that's not listed here: hit **X** then RETURN. This will simply Exit pdflatex and give you back your prompt.
- If you just hit **RETURN** without any letter, pdflatex will ignore this error and proceed to the next one. Sometimes this is useful if you have an error that you know you can skip until later. Note however, that sometimes the current error will cause errors to occur in things that are not wrong. This could happen for example if you mess up a begin{itemize}. If so, any subsequent item will cause an error.
- Hit **S** or **R** (upper or lower case, followed by RETURN). This will be as if you kept hitting RETURN at each error.
- Hit **Q**. This will do the same as the above, except it won't produce any output.
- Type **H**. This will give you some more information about the error. For example in this case, typing **H** gives you the following

```
? h
I suspect you've forgotten a `}', causing me to apply this
control sequence to too much text. How can we recover?
My plan is to forget the whole thing and hope for the best.

?
```

This actually is pretty useful information.

At this point, I usually edit my file and see if I can fix the error. Below, I remind you how to do this

You can edit your latex file in emacs (remember: use the " **emacs &**" command to detach emacs. That way you can run **pdflatex** at the shell prompt).

To jump to **l.9** (line 9), type

```
M-x goto-line
```

Remember that in emacs, **M-x** means " ESC-x " followed by typing out the text " goto-line " in the minibuffer, the last line at the bottom of the emacs window.

Emacs will then prompt you for the line number. Type the number **9** and hit enter.

Emacs will jump to line 9.

Emacs Tip

The **goto-line** command is very useful, but it's a hassle to have to hit **M-x** and type "goto-line" each time. What I do is define a simple key sequence and bind that the goto-line command. You do this by putting the following line in your .emacs file. Open your **.emacs** file (in your home directory), and put this at the following text at the end:
(define-key global-map "C-xC-l" 'goto-line)
Save your **.emacs** file. The next time you open a file in emacs you can simply type **C-x C-l**, CTRL-x CTRL-l (ell), and emacs will ask you for the line number. Much easier!

Ok, back to your Latex file. Now just fix the problem—in this case, make the braces match.

Often when you are learning, you need to look back at one of your examples, like the source files **first.tex** or **second.tex** to remember the correct syntax. In fact, most of us learned Latex by copying sections of working files.

The issue in this file was a "Runaway Argument", because there was no matching end **}**.

So just fix that, save the file, and recompile (run **pdflatex** on) the file.

```
\documentclass[12pt]{article } <--------
```

This will take you to the next error message, or perhaps to the happy result of a readable PDF file.

7.3.3 Other Common Latex Errors

Here are some other common errors that you might encounter.

1. **Missing $ inserted** This is a very common error once you start to have many equations

   or mathematical environments. You left out a $ or [or] somewhere when you were writing text that needs to be in math mode. It can also occur when you use a math command, such as a sub or superscript: x_i or y^2 outside of a math environment.

1. **Extra alignment tab has been changed to cr.** You have too many or not enough &'s in a row of a table or array or eqnarray.

2. **LaTeX Error: begin{** *something*} **on input line** *line number* **ended by end{document}**. You've left out the end{ *whatever*}. Perhaps you're building a table or an eqnarray that has a bunch of lines and you've forgotten the end statement.

3. **LaTeX Warning: Reference ... undefined on ...**

4. **LaTeX Warning: There were undefined references.** You have **'references in your document'_**, and something is wrong. Usually this happens because you have used to a ref but didn't define the label that goes with it, or have a typo in your ref. You need to compile again so LaTeX will be able to get the references right. NOTE:

   These errors are not fatal and will not usually stop pdflatex. However, your document will have something like "...in equation (**??**) we see...".

5. **Paragraph ended before end was complete.** You are probably missing an } at the end of an end{ *something*} statement.

6. **Undefined control sequence.** You've tried to use a command that doesn't exist. Usually this is the result of a typo. Go to the line number mentioned in the error and look at all your commands. You'll probably find something like rfac where you wanted frac.

7. When I try to compile, it starts, but then just hangs. Nothing happens, and I can't do anything. Then when I try to save the file, it won't let me save it with the same name.** You probably have tried to use a package that you haven't installed. For example, if you try usepackage{fancyhdr} but haven't **'installed the package'_**, then you'll hang pdflatex so badly that you should just kill the xterm window (or, if you know how, kill the latex process you've started). **

8. Something like:

```
! Undefined control sequence.
l.668 \includegraphics
                       [width=8cm]{s8.png}


Even though you have spelled the latex command \includegraphics
correctly, pdflatex still balks here. That's because you forgot to
tell latex that you want to **\usepackage{graphicx}** Notice too, that
in showing you the undefined control sequence error, pdflatex skips a
line right after the word that it doesn't recognize. In this case
that's **includegraphics**.
```

1. When referencing figures or equations, the page numbers or equation numbers don't come out right. **Remember** that when using references, you need to run pdflatex twice in order to get the references right.

Now it's your turn

7.4 Homework

Download the following file: fixme.tex.

This file has a number of errors, which are shown at the end of this page.

Your job is to correct the errors in this file so that you get a readable PDF document.

This is Homework 8

Note, that there are non-fatal errors, i.e. there are still a couple problems with the document even once pdflatex exits successfully. Can you spot and fix them?

What I do is iterate the following steps:

Open a terminal and **pdflatex** the *fixme.tex* file. If there are errors, open the file in emacs by typing **emacs fixme.tex &** so that you have an emacs window and your xterm prompt. Now,

1. In emacs, jump to the line where the error is by with goto-line (did you make the C-x C-l key sequence in your .emacs file?)
2. Fix the problem.
3. Save the file.
4. Run pdflatex on the file at the terminal prompt
5. If there is a new error, return to step 1.
6. Iterate these steps until pdflatex ends successfully.

The completely fixed fixme file is here: fixed.pdf.

A list of the errors that *pdflatex* will generate is shown below.

Latex Error Messages from fixme.tex

```
Runaway argument?
{article)
! Paragraph ended before \@fileswith@ptions was complete.

                   \par
l.9

.....................................................................

! LaTeX Error: \begin{abstract} on input line 14 ended by \end{abs}.
```

```
See the LaTeX manual or LaTeX Companion for explanation.
Type  H   for immediate help.
 ...

l.17 \end{abs}

.....................................................................

! Undefined control sequence.
l.41 \ssection
              {Math Errors}

.....................................................................

! Missing $ inserted.

                $
l.48 ...For example, using a sub or superscript x_
                                                   i or y^2 without

.....................................................................

! Missing $ inserted.

                $
l.53

.....................................................................
! Missing $ inserted.

                $
l.56 the $\backslash
                    $. This makes it an inline math mode switch.

.....................................................................

! Undefined control sequence.
l.64 \integrate
               _{0}^{\infinity} \frac{dx}{1 + x^2} = \rfac{\pi}{2}

.....................................................................

! Undefined control sequence.
l.64 \int_{0}^{\infinity
                       } \frac{dx}{1 + x^2} = \rfac{\pi}{2}

.....................................................................

! Undefined control sequence.
l.64 \int_{0}^{\infty} \frac{dx}{1 + x^2} = \rfac
                                                 {\pi}{2}

.....................................................................

LaTeX Warning: Reference `eq:pot' on page 2 undefined on input line 76.

.....................................................................
```

```
! Extra alignment tab has been changed to \cr.
 \endtemplate

l.118 2 & 42 & 2 & &
                     35 \\

.....................................................................

The table doesn't look right.

.....................................................................
```

Warning: This chapter is not complete. We will use this week to get caught up, and begin Chapter 9 next week. Prof. Hetrick (14 March 2015)

CHAPTER

EIGHT

INTERMEDIATE LATEX

8.1 LaTeX Error Messages

8.1.1 Dissecting the Error Message

8.1.2 Common Errors

8.2 Physics Department LaTeX Templates

8.2.1 Basic Lab Report (suitable for PHYS 57)

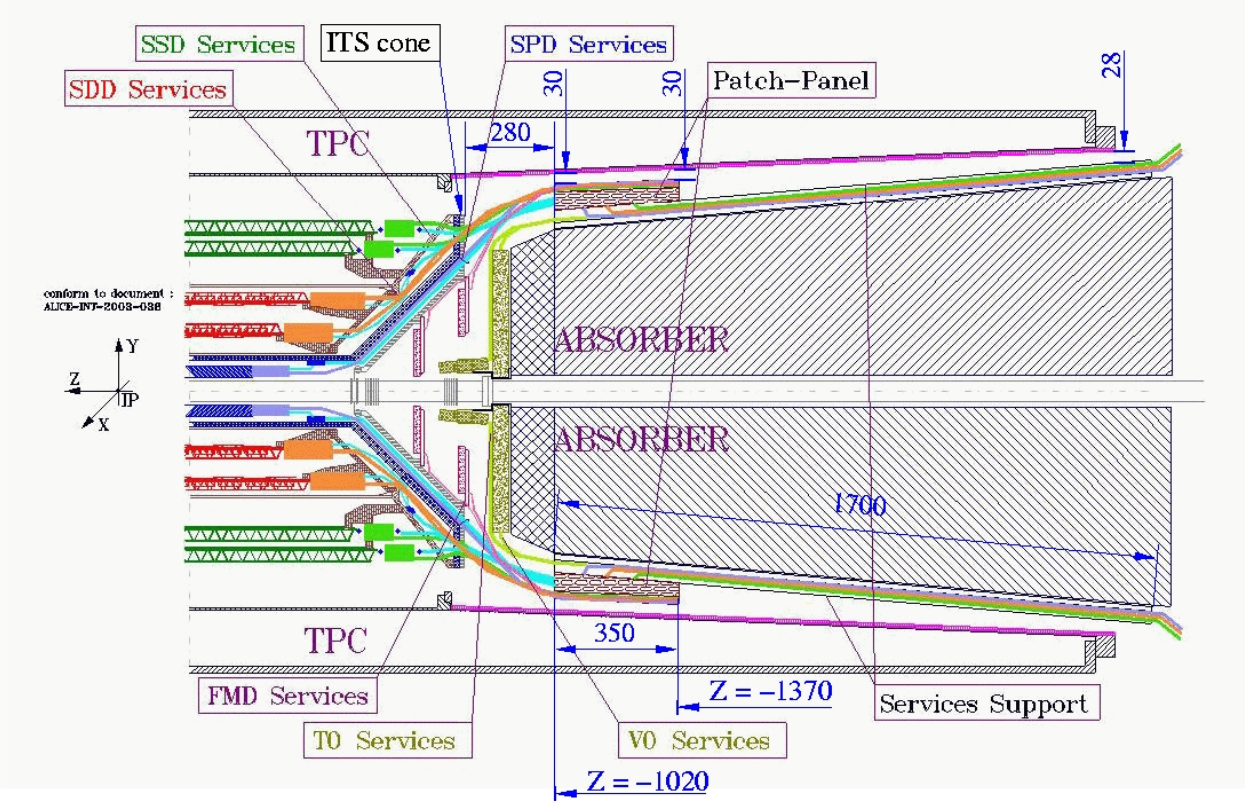
8.2.2 A Professional RevTEX Journal Article (PHYS 151)

8.2.3 The Beamer Class for Presentations

CHAPTER

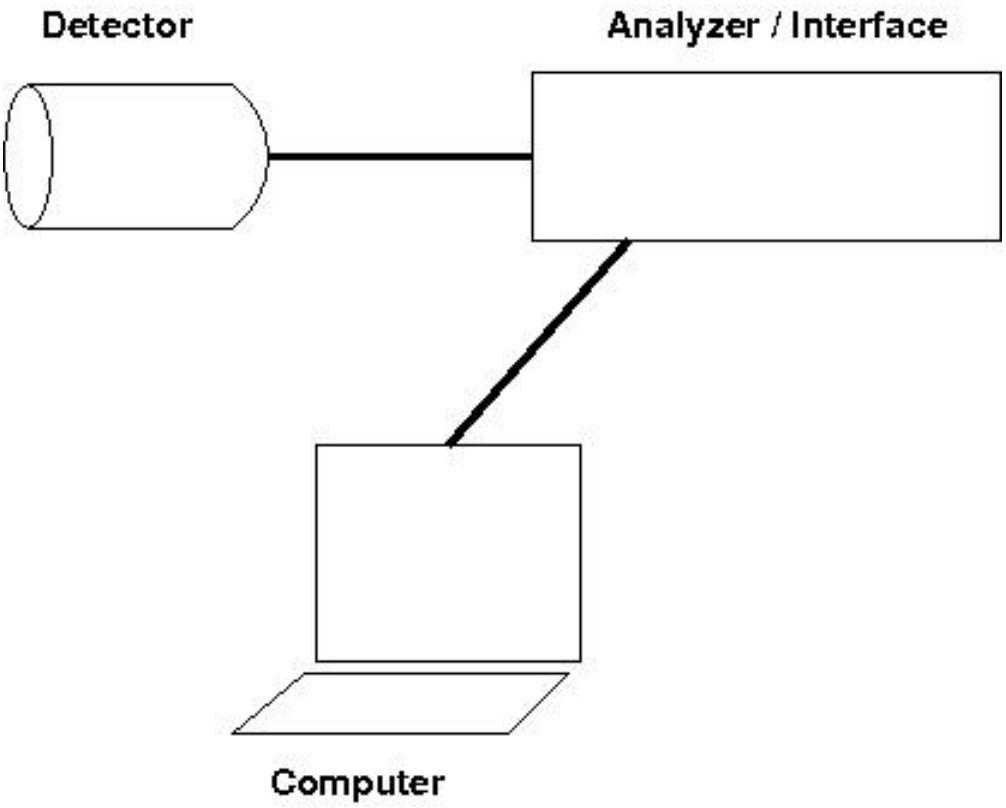
NINE

SCIENTIFIC DRAWINGS

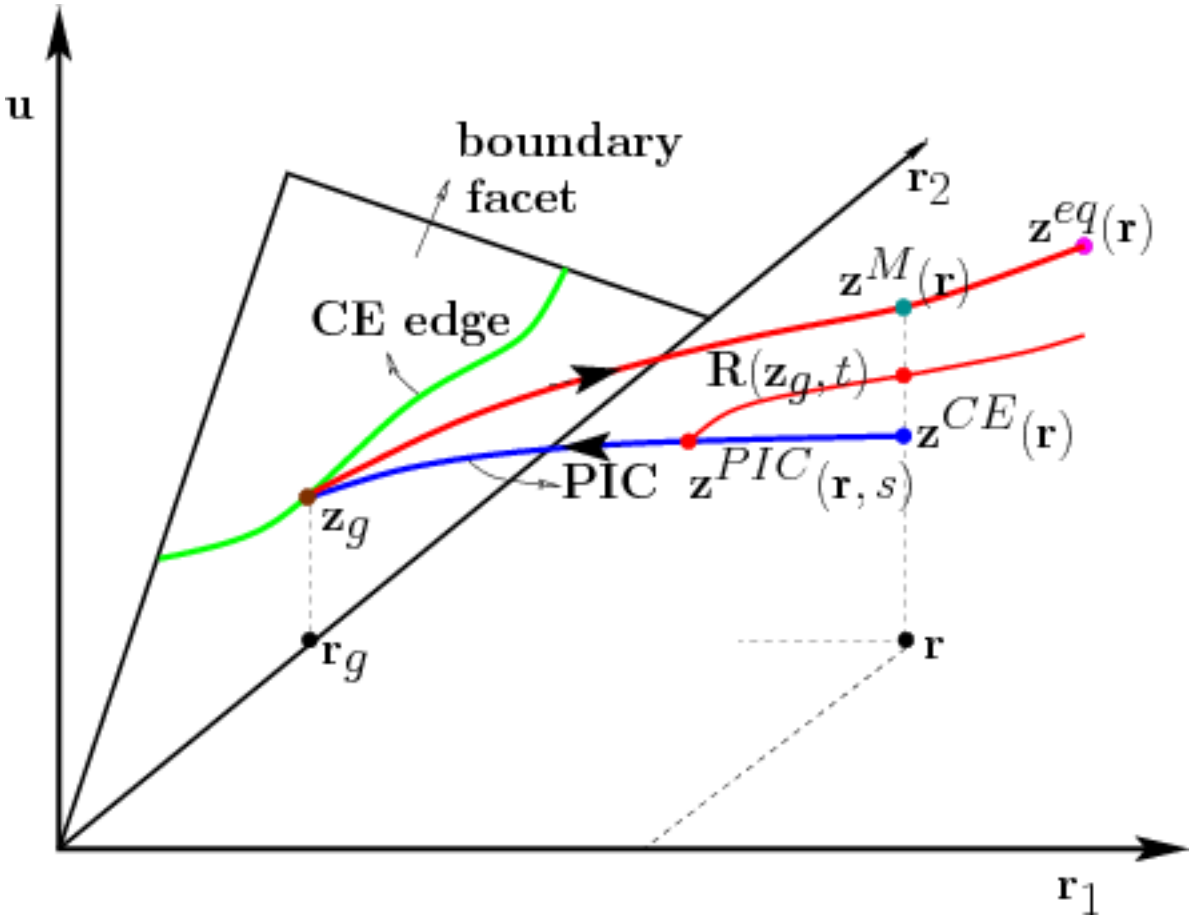


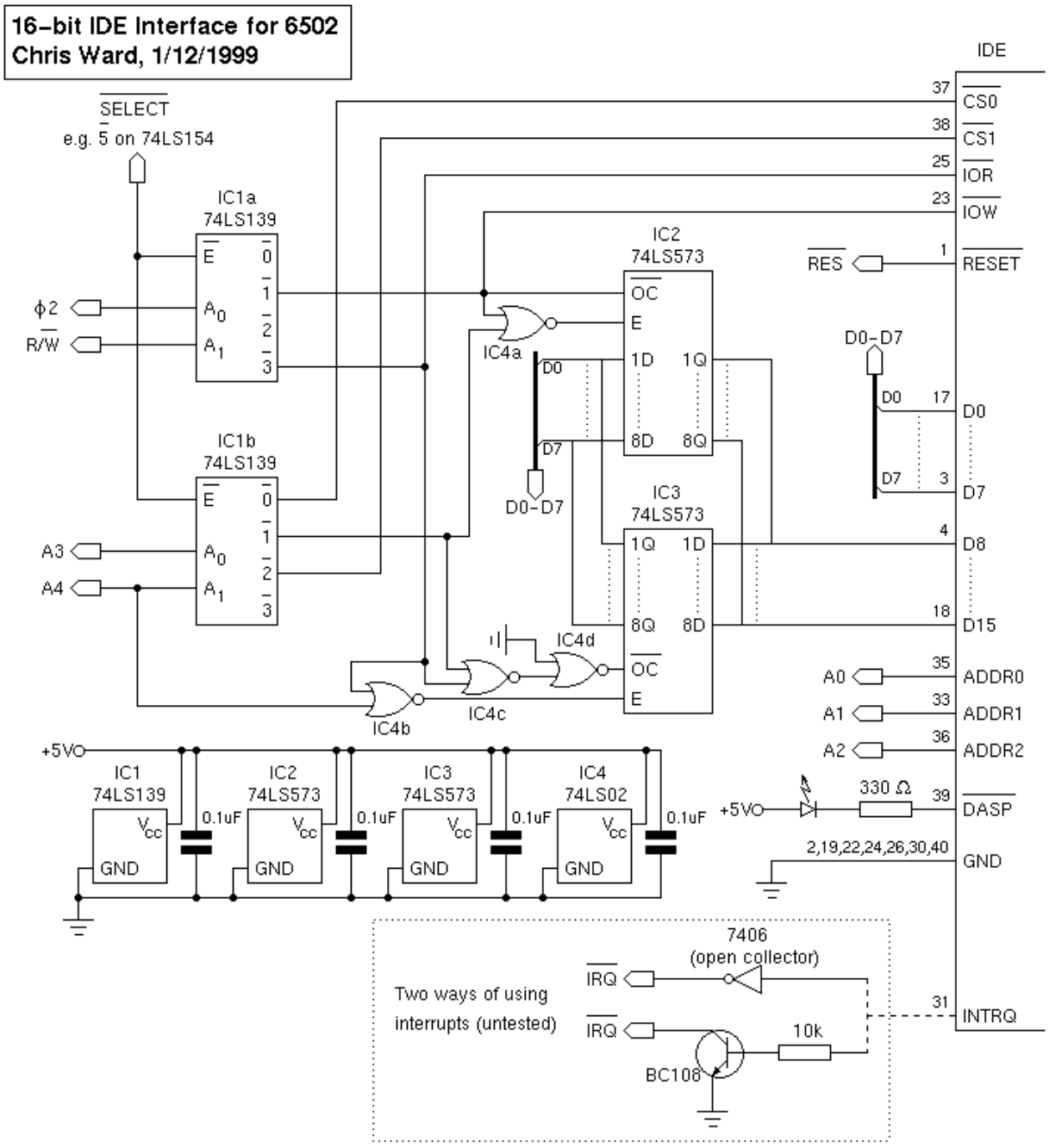
9.1 Using Xfig for Drawings

Sometimes in scientific work you need to make professional looking drawings. These can be simple block diagrams like this, in which you display the basic layout of an experiment:



or a more complex figures and scematics, such as these examples:

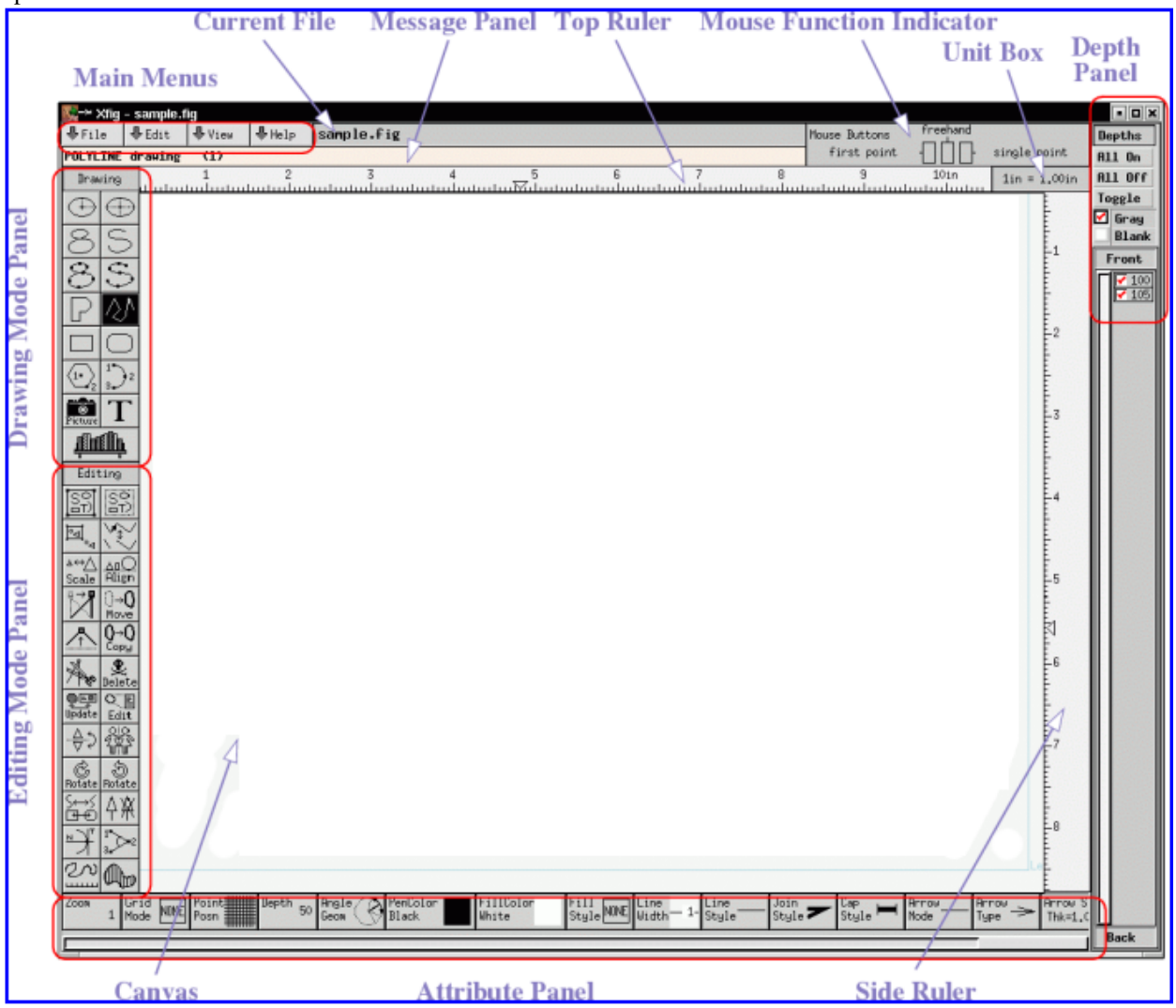

u
boundary
facet
CE edge
$\mathbf{r}_2$
$\mathbf{z}^{eq}(\mathbf{r})$
$\mathbf{z}^{M}(\mathbf{r})$
$\mathbf{R}(\mathbf{z}_g, t)$
$\mathbf{z}^{CE}(\mathbf{r})$
PIC
$\mathbf{z}^{PIC}(\mathbf{r}, s)$
$\mathbf{z}_g$
$\mathbf{r}_g$
$\mathbf{r}$
$\mathbf{r}_1$



Both of these images, and the one at the beginning of this chapter, were made using **Xfig**.

9.2 Starting Xfig

To start the program, open a shell and type: **xfig &** at the prompt. Do so now, so that you can follow along. This will open a new window that looks like this.



Shown in the figure are the various areas of the program. The two main tool panels are the

- **Drawing Mode Panel** This selects the kind of object you wish to draw: Circles, Ellipses, Arcs, Boxes, Lines, Parallelograms, Text, and Images. We'll talk about these in more detail below.
- **Editting Mode Panel** These tools let you manipulate items that you have drawn already. You can Delete, Move, Align, Scale, Rotate/Flip, and group objects into a single Compound Object .
- **Attribute Panel** At the bottom of the screen are buttons which determine the attributes of whatever tool has been chosen. For example, if you are using the Line Drawing Tool, attributes would be line width, line style (solid, dashed, etc.), arrow heads , etc.

9.2.1 Xfig and the Unix Mouse

Look at the " **Mouse Function Indicator**" in the picture above (upper right part of the Xfig window). There are rectangles which represent the mouse buttons; notice that there are three . This is because the Unix/Xwindows world

uses 3 button mice.

On a PC

The **Center** button should be emulated by pressing both mice buttons at once. Thus your mouse buttons are:

- Unix **Left** = PC **Left**
- Unix **Center** = PC **Both**
- Unix **Right** = PC **Right**

On a Mac

Apparently, the recent versions of Fedora Linux/VirtualBox running on Mac OS *does not* include a way for Mac users to emulate a 3-button mouse, as described for PC users above. *However*, if you simply plug in a 3-button (USB) mouse, or a more commonly found 2-button mouse, *with a clickable center scroll-wheel*, it will work fine.

Office Depot carries mice like this for about $10. Unfortunately, I don't know any other way to use Xfig with a Mac.

Note: Copy and Paste in Unix Windows

One of the nice features of Unix/Xwindows is that when you highlight text in an Xwindow, it is automatically **copied** to the clipboard .

Compare this to the PC method: first hightlight text, then click the Edit->Copy OR do CTRL-C. In Unix, the *copy* is automatic whenever you highlight something.

To **Paste** text, you simply position the cursor and click with the Middle mouse button. On a PC that means click both mouse buttons.

Try this:

Open a terminal (if one is not open), and type some text at the prompt. Anything will do

salkfjdhla

Now use the **Left** mouse button to highlight this text.

Next, in the same xterm window, click the **Center** mouse button—i.e. **push** both PC mouse buttons simultaneously .

This should **Paste** the text that you automatically copied when you highlighted the previous text.

9.3 Using Xfig

Now that you have an **xfig** window up and running, let's start to play with it. It's fairly intuitive I think—just remember that some actions are done with the **Center** mouse button.

9.3.1 Drawing

- **Make a circle**: Click the Circle Drawing tool at the top left of the Drawing tools. Now, **Left**-click and **drag** to adjust the size of the circle. When you have a circle of about 1-3 cm in radius, **Left**-click to fix the size and attach the circle to your canvas.

- **Make a Line**: Now, click the Line Drawing Tool (the line segments–4th down on the right column). **Left** click on the canvas; this starts a line. Move the mouse and **Left** click again. Each Left click gives the line a new vertex. Now, **Right**-click. Did your line disappear? Right clicking, with the Line Drawing Tool, means Cancel . Restart a line segment sequence as shown at the beginning of this paragraph. After a few vertices, **Middle**-click (BOTH buttons together). This should attach the line to your canvas.

Ok. Do you see how it works? It's pretty easy; you just have to get used to the tools.

- **Try each of the Drawing Tools**.
- What kind of Object does it create?
- Understand how it works—what do Right, Center, and Left mouse buttons do?
- What Attributes does it have? What do the different options do?

Now let's do a simple exercise using the **Edit Tools**.

9.3.2 Editting

- Click the **Delete** Tool button: 7th up from the bottom on the left column in the **Edit Tool Menu**. A number of little squares should appear on your drawing, and your cursor should become a "Skull and Crossbones".
- *Left-click* on one of the items in your drawing. Did it disappear? It should have. That's what the Delete tool does!
- Now click the **Move** Tool button. Your curson should become a pointing hand.
- *Left-Click* on one of the little squares on one of the objects in your diagram. You should be able to Click-drag it around your canvas, and leave it at a new location.

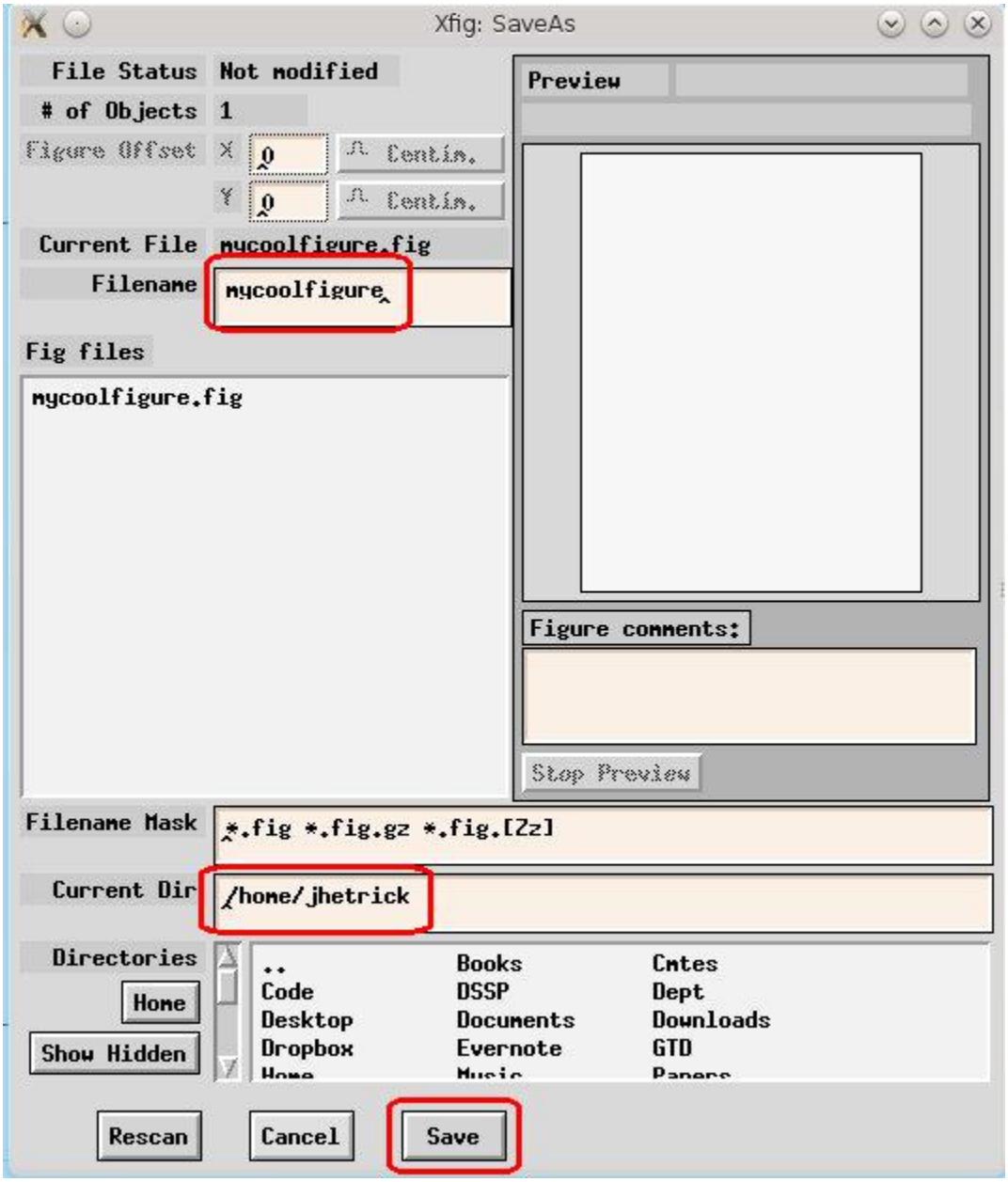
9.3.3 Saving and Exporting

Open **xfig** and make some simple figure using the techniques you learned above. Anything will do, even just a line or circle.

Once you have a figure, there are two ways to store it on disk..

Save/Save As...

To Save your figure in *Native Xfig Format* (as a **.fig** file), you click *File->Save* on the upper left menu bar. This opens the *Save* dialog box:



The essential items are circled in red above:

- The *filename*. Without an extension, your file will be saved with the xfig extension: *.fig*.
- The directory in which your file will be saved: */home/jhetrick/* in the screenshot.
- The *Save* button.

This will save your figure *in Xfig* format, so that you can read it back into **xfig** and continue editing it. *However, this format is only readable by xfig*.

I usually save my figures in this format *and* export them to an image format. That way, if I decide later to change the figure, I can re-open the **.fig* file and make changes. You can't do that with the exported image file.

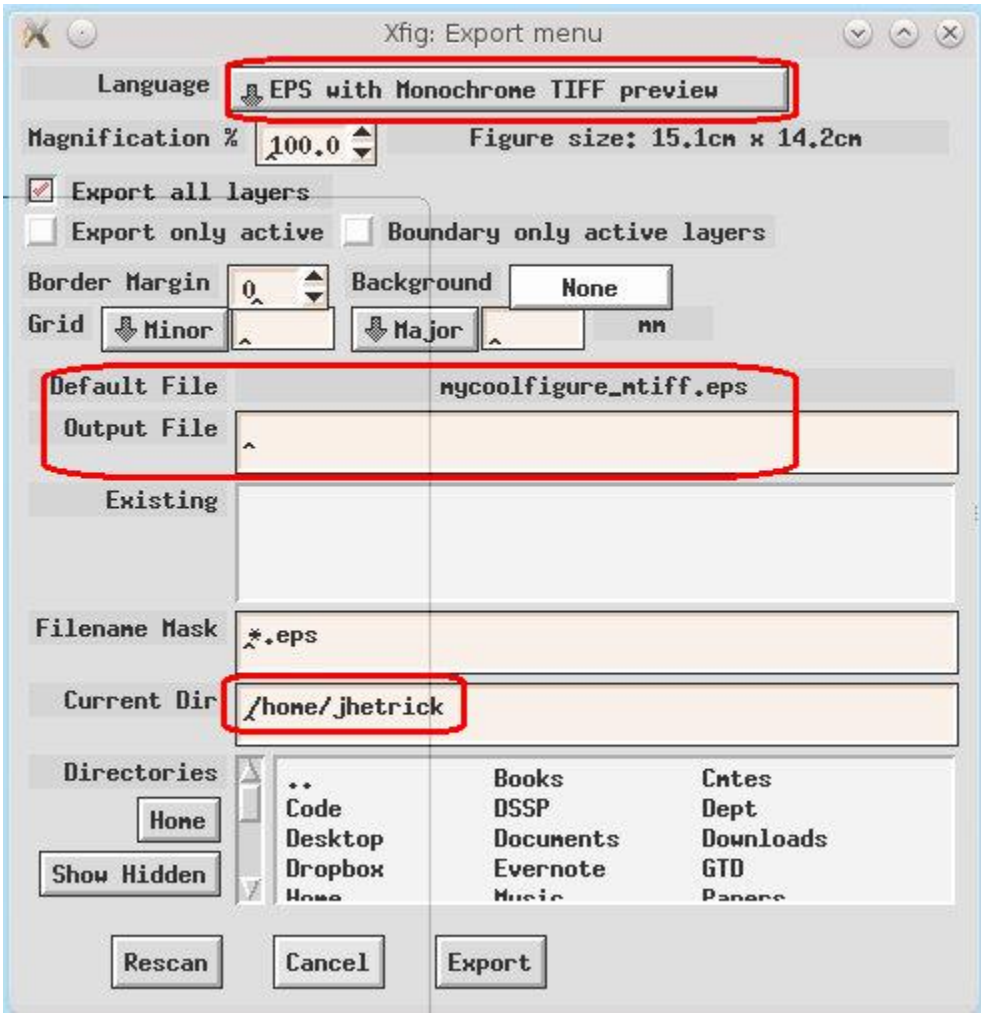
Exporting to an image

When you have the figure in its final form, you are ready to *Export* it to an image file, suitable for inclusion in another document–either a Latex or Word document, or perhaps an HTML page on the web.

To *Export* your figure to an image file:

Click File->Export

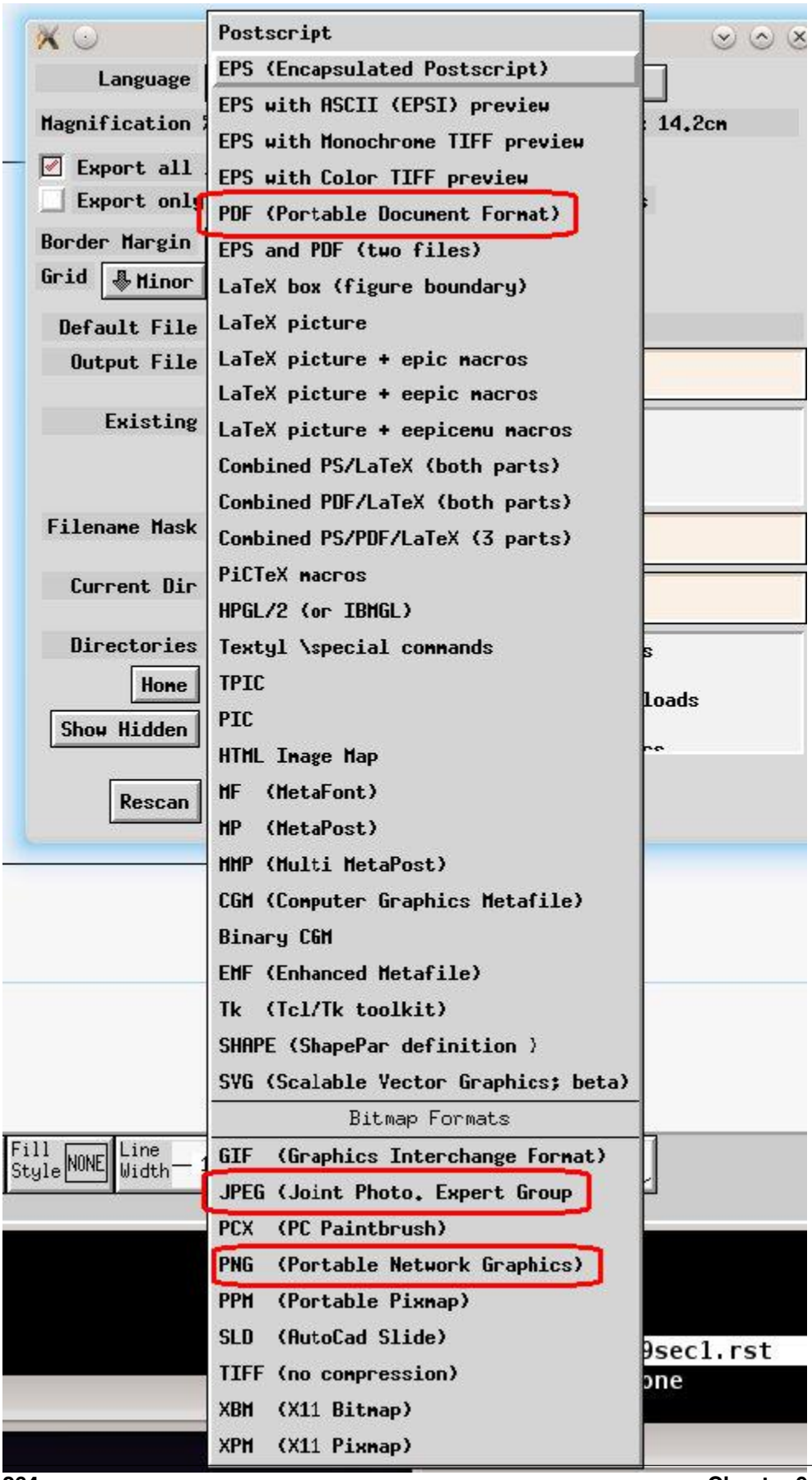
This brings up the *Export* dialog box:



Notice:

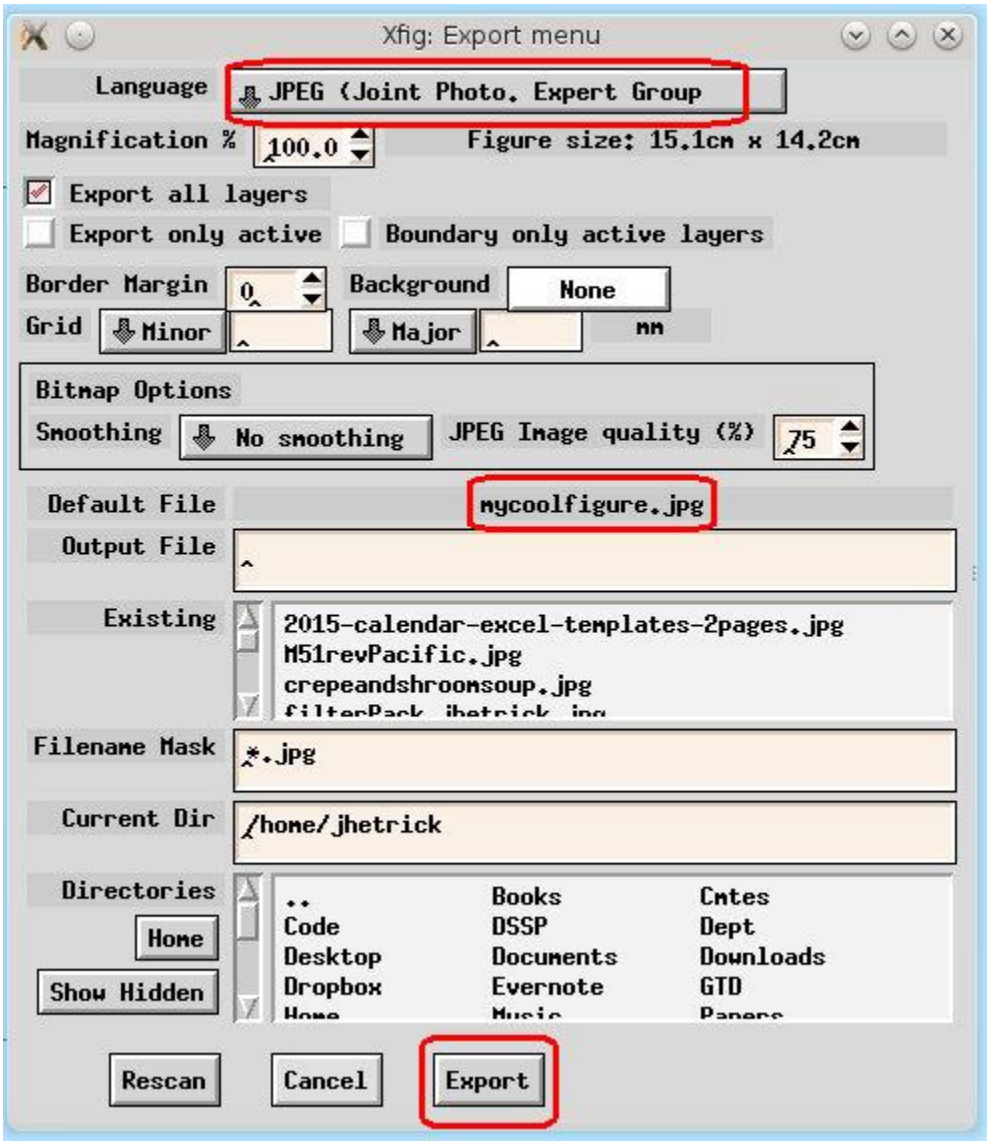
- The "*Language*" menu–this really means the *image format*: EPS, JPEG, TIFF, PDF, etc.
- The *Output File*. This is the name of the exported file. If you already saved your figure (as *mycoolfiure.fig* in this case), then there will be a "*Default File*" with the same name and an extension corresponding to the *Language* of file type you chose.
- *Current Dir*ectory, where the image file will be saved.

If you *Click* on the "Language" dropdown, you can see all the possible export formats:


Language
Magnification
Export all
Export only
Border Margin
Grid
Minor
Default File
Output File
Existing
Filename Mask
Current Dir
Directories
Home
Show Hidden
Rescan
Fill Style
NONE
Line Width
14.2cm
Postscript
EPS (Encapsulated Postscript)
EPS with ASCII (EPSI) preview
EPS with Monochrome TIFF preview
EPS with Color TIFF preview
PDF (Portable Document Format)
EPS and PDF (two files)
LaTeX box (figure boundary)
LaTeX picture
LaTeX picture + epic macros
LaTeX picture + eepic macros
LaTeX picture + eepicemu macros
Combined PS/LaTeX (both parts)
Combined PDF/LaTeX (both parts)
Combined PS/PDF/LaTeX (3 parts)
PiCTeX macros
HPGL/2 (or IBMGL)
Textyl \special commands
TPIC
PIC
HTML Image Map
MF (MetaFont)
MP (MetaPost)
MMP (Multi MetaPost)
CGM (Computer Graphics Metafile)
Binary CGM
EMF (Enhanced Metafile)
Tk (Tcl/Tk toolkit)
SHAPE (ShapePar definition)
SVG (Scalable Vector Graphics; beta)
Bitmap Formats
GIF (Graphics Interchange Format)
JPEG (Joint Photo. Expert Group
PCX (PC Paintbrush)
PNG (Portable Network Graphics)
PPM (Portable Pixmap)
SLD (AutoCad Slide)
TIFF (no compression)
XBM (X11 Bitmap)
XPM (X11 Pixmap)

I've highlighted some of the more popular: PDF, JPEG, and PNG. Note that each of these is a format that can be easily included into a *Latex* document.

Once you have chosen your export format (JPEG shown below), you can see that the *Default File* name shows the appropriate suffix, and you are ready to click the *Export* button at the bottom of the dialog box.



A Handy Tutorial

Below is a link to a nice tutorial I found on the web if you are interested in delving further into Xfig.

- Xfig Tutorial by Peter Hickocks

9.4 Homework

Homework for Chapter 9 is here

CHAPTER

TEN

REMOTE LOGIN AND FILE TRANSFER

10.1 Remote Login and File Transfer

10.1.1 A Little History

The 1970's

Before the days of Personal Computers (PCs), computers were giant room sized machines called Main Frames as we discussed in the **'Introduction'_** to this course. These computers were large machines running in an isolated and temperature controlled place (usually called The Machine Room) and users logged on to the Main Frame. These computers supported many users, logged in simultaneously.

The early 1980's

Then came the PC. With a personal computer, the programs and files, etc. were all physically located in the box on the user's desk, where everything is self contained. Typically, only one user would be logged on to the computer at any time.

The mid 1980's

About the same time as the rise of the PC, the Internet (but not The Web which uses the Internet) became an important part of the worldwide cyberstructure allowing computers to transmit packets of data addressed to a unique computer somewhere else on the internet. Still The Web as you know it know did not exist. In the late 1980's we had three basic forms of using the Internet:

- **Email**
- **Telnet** This is a way of logging into a remote computer, as if you were sitting at a terminal connected to that computer. Once logged in, you got a shell prompt on that machice, just like you get when you open an xterm on your computer. You could issue unix commands and move around the file tree from the prompt.
- **FTP** (File Transfer Protocol). This was a separate program from Telnet which allows users to transfer files to/from a remote computer.

In the days before the World Wide Web (WWW), this was the way we got things from the internet. There were archives of open files that one could access by using FTP protocol to login as "*Anonymous*" with password "*your@email.address*". Once in, you could browse and download files—usually through a 9600 baud modem...

1990

Then came the World Wide Web, a system that linked files on different computers through " Browsers that could interpret files written in "HyperText Markup Language" (HTML). These files could contain "Links" to files, or other HTML pages. This system was developed at the European Particle Accelerator, CERN. Recently the CERN Courier, the monthly news publication from CERN, published an article marking the 20th anniversary of the Web. It featured this document:

Vague but exciting ...

CERN DD/OC — Tim Berners-Lee, CERN/DD

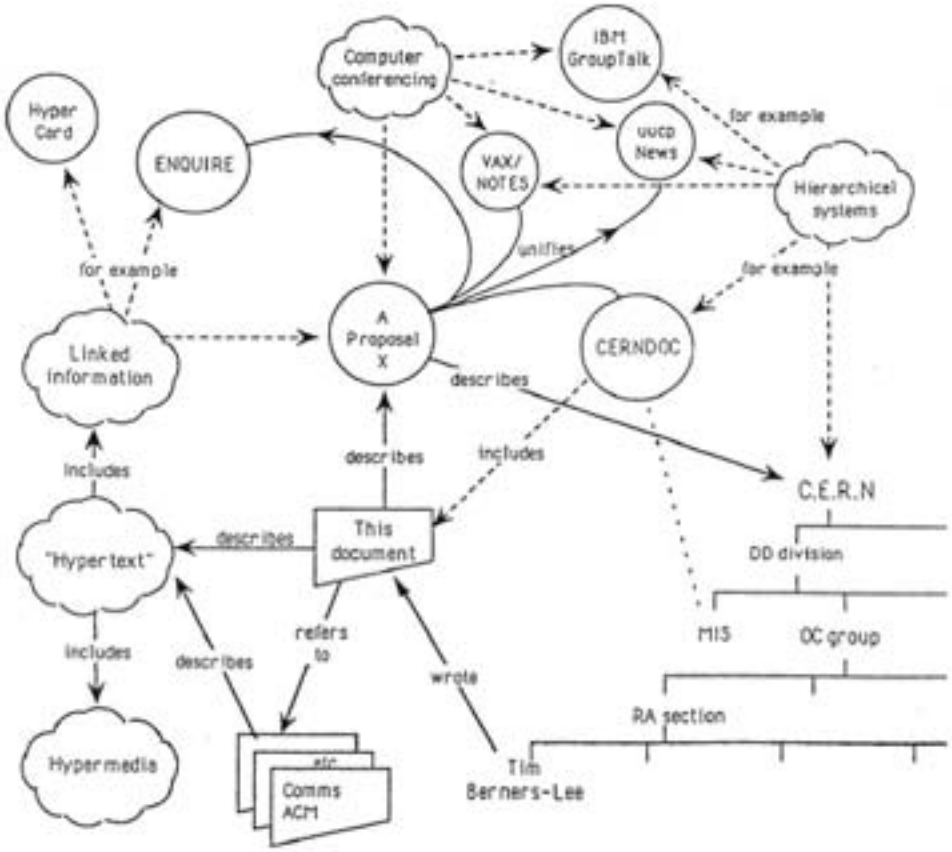
Information Management: A Proposal — March 1989

Information Management: A Proposal

Abstract

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext sytstem.

Keywords: Hypertext, Computer conferencing, Document retrieval, Information management, Project control



which is a copy of Tim Berners-Lee's original memo to his boss outlining an idea for a system of servers, browsers, and interactions. Notice the handwritten note his boss wrote at the top: "*Vague but exciting...*", scribbled by Berner-Lee's boss.

The idea went on to transform the world and lead to The Information Age of global connectivity. The complete article for the CERN Courier is here if you are interested in reading it. While we can now cruise the Web and find a truly amazing amount of information in a variety of forms, there are still times when it is necessary to login to a remote machine and perform tasks from a non- graphical (no mouse or menus) prompt.

10.2 Get Connected with SSH

In the scientific world you usually remote login to other computers because

- the other computer has special resources that are not available on your computer. For example, the remote machine might be one of those supercomputers you looked at in Homework 1. It is super-fast and has a ridiculously large disk.
- your files are there. In this case you might use (S)FTP to get them
- Often certain computers at an institution or lab have certain software applications on them. One computer might have data visualization packages, another might have the statistical software. Another may be a computing cluster for running programs. Yet another could be the storage server connected to a large disk.

Thus we need to learn the basics of remote login.

10.2.1 SSH: Secure Shell

Telnet, which I mentioned above, is a program that you run at the command line of an xterm shell window. It opens a connection to a remote machine which allows access via telnet, then presents the user a login prompt (asks for your username and password) on the remote machine. If the user can provide the correct password, a shell prompt awaiting commands was returned.

The **problem** is... Telnet transactions are sent over the internet as **plain text**, i.e. as simple ASCII characters. These packets bounce from one router to the next to the next and so on, until they finally arrive at their destination. Anywhere along the way, an unscrupulous person could record packets traveling through nearby routers and read all the data. It is very simple to look for the characters " Username: " and " Password: " in the stream, then read the characters following this (up to the RETURN). The hacker would then have the person's username and password.

Note: In about 2001 I discovered a breach of Pacific's network by finding a hacker logged into one of the machines here. After killing his login and looking around a bit, I found that he had set up a program to log router traffic which left a file which contained the usernames and passwords of all the people on the south campus who accessed our school database via telnet on a router in the Psych building.

Because of this big flaw in Telnet, not many computers allow telnet access these days. The replacement for Telnet, is **Secure Shell** (SSH). Basically SSH is the same as Telnet–offering the user a shell prompt on the remote machine–however it encrypts all the data that is transmitted between the machines. Even if a hacker recorded the entire transmitted session, it would only look like giberish. The transmission can only be decoded by public key decryption, which is quite strong if the keys are sufficiently large.

10.2.2 Using SSH

First, we need a machine that we can ssh into.

I've made an account for us to use on **physics-nix.stk.pacific.edu**.

The *username* (*for all of us*) is *phystu*

The *password* for this account is given under the *Resources* course tool menu item on the Sakai PHYS 27 site (look for *Resources* on the left hand tool menu). Open the Resourses Folder and look for the document on the SSH Accounts. The reason for putting this here is that I don't want to post the password to this account on the internet as a webpage. I only want those who have PHYS 27/193 access to Sakai to be able to view it.

Using what you learned above, **ssh to physics-nix.stk.pacific.edu**.

Remember that your USERNAME on this machine will be **phystu** (we will be sharing the account) and the domain name is given above (*physics-nix.stk.pacific.edu*).

Also,

Warning: You *must* be on campus to login to *physics-nix*. If you live off campus (outside the Pacific firewall), you will need to contact me about getting VPN (Virtual Private Network) access. Or you can just do the work of this chapter and the next by bringing your laptop on campus.

Exercise

Open a terminal shell, and type:
ssh phystu@physics-nix.stk.pacific.edu

Once you are logged into **physics-nix.stk.pacific.edu** you will get a prompt that looks like this:

```
phystu@physics-nix[~]>
```

I've preloaded the *.bashrc* file for you.
Do a directory listing
ls
to see what files are there (not much).

10.2.3 Emacs on a remote machine

Now that you have a shell prompt on the remote machine (**physics-nix**), you can issue commands there.

For example, the `which` commmand tells you if a command or program is available, and the location of the program file.

Type:

which gnuplot

to see the location of the **gnuplot** executable file on this machine.

Suppose we wanted to edit a file on **physics-nix**. You could type **emacs**, and in a little while an Emacs window will pop up on your screen. The reason it takes a little while is that the full X-Windowed data is being sent through the network. **physics-nix** must get information about your graphics capability, the location of other windows, the status of your mouse, etc., then send the information for an Emacs window though the internet to the machine you are currently working from (the laptop or desktop on which you are now working). *AND–all this gets encrypted*.

This is a fairly bulky process involving a lot of network traffic. Sometimes it's necessary–say, if you are using gnuplot; you have to be able to look at the graph.

However, if you can save on bandwidth, your connection to the remote computer will be faster.

This is why it's useful to be able to use emacs in "*Text Mode*" as we discussed when we learned in the Emacs: GUI v. Text.

You can start emacs in text mode on **physics-nix** by typing

emacs -nw

Remember this command? The **-nw** stands for "No Window".

Now you can run programs and edit files on the remote machine. What more do you need!

You could be in a cyber cafe in Paris, *ssh'ed* into a computer here in the US, and–as long as you are familiar with moving around in a shell at the command prompt–have pretty much complete access and control of the computer here in the US.

In my research, I use supercomputers around the world, like Hopper shown here at NERSC

Once I am granted an account on one of these machines (usually by a grant writing procedure, where I get access for a certain amount of time), I can login via *ssh* from anywhere, just as you have done in this tutorial.

I then use *emacs* to create and modify files, such as simulation programs which I then run on that particular computer, making use of the fact that *it is INCREDIBLY powerful and has a HUGE amount of disk space*.

Once these programs have finished running, I can analyze the output on the supercomputer, or even better, reduce and *transfer* the data file to my local computer for analysis here (more on *file transfer* shortly).

Exercise

Before you log out from **physics-nix**, use emacs to create a file called "**YOURNAME_nix.txt**". You will need this file below.
where YOURNAME is (duh), *your name*. In the file, put the text

```
This file was created by USERNAME on physics-nix.stk.pacific.edu.
Date: today's date
I'm feeling: --how are you feeling today?--
```

10.2.4 Getting Back Out

When you are finished with your work on the remote computer, you logout by typing:

logout

oddly enough.

Equivalently, you can type *exit* or even just: *CTRL-d* (this is my fave).

All of these do the same thing: exit your session on the remote machine and log you out.

Logout from *physics-nix.stk.pacific.edu*, if you haven't already.

This should leave you at the prompt in your terminal shell on your laptop/computer.

The reason I set up your *.bashrc* file to display:

sci[~]>

in the propmt is so that you will know that you entering commands back on your local laptop. For the same reason, I put *phystu@physics-nix[~]>* in the .bashrc file for user *phystu* on *physics-nix*. It helps keep track of which machine you are logged into when typing commands at the prompt.

10.3 SFTP and SCP

Sometimes you don't want a prompt on the remote machine to run commands,, but rather you need to *get* (or *put*) some files from (or to) the remote machine.

That's when you call on **sftp**: *Secure File Transfer Protocol*. The syntax is very much like **ssh**.

10.3.1 SFTP

SFTP stands for *Secure File Transfer Protocol*.

At the prompt on your local machine (your laptop, say), type:

sftp phystu@physics-nix.stk.pacific.edu

In a short while, **physics-nix.stk.pacific.edu** will ask you for the *password* for the phystu account (this is the same one as you just used to ssh to **physics-nix** above. Finally, you will get a prompt that looks like this:

```
sftp>
```

You can find all the commands that **sftp** accepts by typing **?** .

```
sftp> ?
Available commands:
cd path                       Change remote directory to 'path'
lcd path                      Change local directory to 'path'
chgrp grp path                Change group of file 'path' to 'grp'
chmod mode path               Change permissions of file 'path' to 'mode'
chown own path                Change owner of file 'path' to 'own'
help                          Display this help text
get remote-path [local-path]  Download file
lls [ls-options [path]]       Display local directory listing
ln oldpath newpath            Symlink remote file
lmkdir path                   Create local directory
lpwd                          Print local working directory
ls [path]                     Display remote directory listing
lumask umask                  Set local umask to 'umask'
mkdir path                    Create remote directory
put local-path [remote-path]  Upload file
pwd                           Display remote working directory
exit                          Quit sftp
quit                          Quit sftp
rename oldpath newpath        Rename remote file
rmdir path                    Remove remote directory
rm path                       Delete remote file
symlink oldpath newpath       Symlink remote file
version                       Show SFTP version
!command                      Execute 'command' in local shell
!                             Escape to local shell
?                             Synonym for help
```

You can intuit most of these. The most commonly used commands are:

- **cd path** change directory on the remote machine
- **lcd path** change directory on the local machine
- **get filename** download a file from the remote to local machine
- **put filename** upload a file from the local to remote machine
- **mget filen*** download many files (* matches any characters)
- **mput filen*** upload many files (* matches any characters)
- **quit** or **exit** exit sftp

Exercise

Make sure you have opened an `sftp` session on `physics-nix` as described above.
Now, at the `sftp>` prompt, type
get USERNAME_nix.txt
You should see something like this:

```
sftp> get jhetrick_nix.txt
Fetching /home/jhetrick/jhetrick_nix.txt to /jhetrick.txt
/home/jhetrick//jhetrick_nix.txt                                    100%  444     0.4KB/s   00:00
```

Now **quit** the *SFTP* session, by typing.... `quit`.

Once you have your `sci[~]>` prompt back, indicating you are in the shell on your local computer, do an *ls* to see that you indeed downloaded the file **USERNAME.txt** *from* **physics-nix** to your laptop.
Is it there?
View the contents of the file (remember how?) and check that its contents say that it was created by you on **physics-nix**.

This is how you get file to and from remote computers.

To upload files, you do pretty much the same thing, except that you use the **put** command.

Exercise

On **your local computer** (i.e. your laptop), create a file called "**USERNAME_local.txt**". Add the text "*This file was created on my super-deluxe Dell-o-tron.*" (or something similar that describes your laptop/computer) to the file, so that you know that this was the locally created file.
Upload this file to **physics-nix**.

I'll be able to see this file on physics-nix so I'll be able to tell if you have done it correctly.

This is part of Homework 10.

10.3.2 SCP

Another way to transfer files is to use *Secure CP*. This tool is a mashup of SFTP and the usual unix CP command.

Recall how we copy a file from one place to another, by using the **cp** command:

```
cp file1 file2
cp file1 dir/
cp file1 dir/newname
```

to:

- make a *copy* of *file1* called *file2*, in the same directory
- make a *copy* of *file1* in the directory *dir/*
- make a *copy* of *file1* called *newname* in directory *dir/*

In general, the `cp` command takes the form:

cp source target

meaning that the first thing (the *source*) is copied to the second thing (*the target*).

SCP allows us to use essentially the same one-liner command, but include reference to files and directories on remote machines. The syntax goes like this:

```
sci[]> scp file1 janedoe@remote.mach.ine:dir1/dir2/file2
```

This command will

- open a connection to the machine called *remote.mach.ine* and login as user **janedoe** and ask for her password on the remote machine.

  This is what is meant by the text: **janedoe@remote.mach.ine:**

- put a copy of *file1* in the *sub-subdirectory dir1/dir2* beneath *janedoe's* HOME directory on the remote machine, called *file2*.

Note: if your username on the local machine AND the remote machine is the same, you don't have to include the username (janedoe, in this example). You could just give the remote machine name. In that case, you would type **scp file1 remote.mach.ine:** to put `file1` in your HOME directory on `remote.mach.ine`).

Exercise

To try this out for yourself, open a terminal shell on your laptop, and make a test file (*emacs, "blah blah", save->* `mynametest.txt`).
Now scp this file to *physics-nix*. If the file you just made is in the current directory, you would do this:
`scp mynametest.txt phystu@physics-nix.serv.pacific.edu:`
Rememeber the password for *phystu* on physics-nix; you'll be asked for it.
This should put a copy of the file `mynametest.txt`, in the home directory of user: *phystu*, on the remote machine: **physics-nix.serv.pacific.edu**.

Now, logon to *physics-nix.serv.pacific.edu* using **ssh**:

```
ssh phystu@physics-nix.serv.pacific.edu
```

At the prompt on *physics-nix*, do a directory listing and verify that the file `mynametest.txt` is there.

Edit it with emacs (*remotely, using text-mode emacs* as discussed above), and add a sentence to the end:

"*This file was successfully copied to physics-nix with scp*"

Now, logout from physics-nix (type exit, or CTRL-d). You should have the `sci[~]>` prompt back in your local terminal shell.

Delete the *local* `mynametest.txt` file (*rm mynametest.txt*). We are going to grab back the copy we just put on physics-nix.

Do an `ls` to verify it is gone.

Now, get a copy from physics-nix:

```
scp phystu@physics-nix.serv.pacific.edu:mynametest.txt .
```

Notice this time the *source* is on the remote machine, and the *target* location/name is just **.** . Remember, **.**, a single period, means "**here**" in unix. It stands for "*the current directory*".

This should prompt you for your password on the remote machine, then copy the requested file to the current directory on the local machine.

If you were successful, you should have a file on your local machine that has the added sentence: "*This file was successfully copied to physics-nix with scp*". That's how you *know* you have pulled the copy from physics-nix.serv.pacific.edu, *after* you put it there in the first place.

10.3.3 What's the difference between *sftp* and *scp*?

Basically, *sftp* and *scp* do the same thing–move files from one computer to another. However, *sftp* is a bigger program, whereas *scp* is a oneliner that allows you to grab a single (or multiple) file(s), if you know where on the remote machine they are. You have to give scp "the full path/name" information to the file in order to find it.

With *sftp*, you get an `sftp>` prompt. SFTP really gives you a "*file transfer shell*–an environment in which you have a prompt and a number of commands that you can type, such as `help`, `ls`, `cd ..`, etc. This `sftp` shell allows you to move around within the *sftp* environment (on the remote machine). So, if you can't remember where the file is located on the remote machine, *sftp* would be the way to go. You can login to the remote machine, `cd` around, `ls` to view the directory contents, and then grab the files you need with the `get` command.

10.4 Homework

If you did the above exercises, you are ready for Homework 10, which is here:

Homework 10.

CHAPTER

ELEVEN

SYMBOLIC MATH AND NUMERICAL LINEAR ALGEBRA

11.1 Symbolic Math and Numerical Linear Algebra

Now that you know how to logon to the Physics department's utility computer, **physics-nix.stk.pacific.edu**, you can use some of its specialized software. Actually this machine is VERY old! There is a new replacement for it sitting in a box beside it, but that will have to wait for time later in the summer to install. Nonetheless, you can get started using the old machine. So far, we've learned about

- The Scientific Environment
- Unix
- Gnuplot
- Latex
- Xfig
- SSH/SFTP

The only time we got to do anything like real science was using *gnuplot* where we plotted some data, did some fits, and made a prediction about the population.

Let's return to tools for doing science and math.

11.1.1 Doing Symbolic Calculations (algebra and calculus) with Mathematica

Much of scientific computing is about manipulating numbers: computing them, graphing them, interpreting them , etc. But math is the language of science, and so it would be nice if we can also use the computer to do math.

Of course we can do math—and have. In our Gnuplot investigations we solved the equation $2x^2 - 3x + 1 = x$, graphically . This however involved a numerical method, successively limiting the **x** range to find where the left and right sides were equal. The solution was good to about 6 decimal places (since that's the limit of our x-range accuracy in those plots).

Also using Gnuplot, we can plot interesting mathematical functions, such as the *ArcTangent* $tan^{-1}(x)$ and the *Bessel function* $j_1(x)$. However, what gnuplot does is generate tables of numbers to plot for these functions.

We can't really do **algebra, calculus**, or **differential equations** with gnuplot—these require **Symbolic Math Calculations**, the manipulation of abstract variables, *a, b, g* and operations like $\frac{d}{dx}$. For this we use tools like

- Mathematica (Wolfram Inc.)
- Maple (Maplesoft Inc.)

Another tool you may have used or heard about is: Matlab. Matlab is a *numerical linear algebra tool*. We'll talk about it in the next section.

The two above are the leading packages (as of now) of the tools for symbolic math on the computer, and as you can see they are commercial applications meaning they are not free. An enormous amount of research and work has gone into creating software environments that can do math the way you do it in math class: manipulating variables according to the rules of algebra, trigonometry, calculus, differential geometry, combinatorics, etc. Thus these packages are not cheap: both Mathematica and Maple sell at about $1000 for the basic package (however there are student versions for each at $100-150.

While my philosophy is strongly geared toward providing you with Free software, there is nothing in the Free realm that compares to these systems. There *is* an open-source Python project called Sage, which is being developed. But it is not as powerful as these other tools (yet).

We can, however, give you access to some of these commercial tools because the university has site licenses and these packages are available on certain machines. For the moment, we will explore **Mathematica**. This was part of the purpose of teaching you to login on remote machines. Now the university (and you!) can save money by putting special software on certain machines and then giving students and faculty access to those machines. This is one of the purposes of our machine: **physics-nix.stk.pacific.edu**.

Let's play with Mathematica

We won't be able to delve deeply into using *Mathematica*–for that Wolfram Inc. offers courses, online seminars, trainers and consultants. However, once I give you access to *Mathematica*, one of the easiest ways for you to get deeper into it is to Google "Mathematica Tutorial" or "Introduction to Mathematica" and look at the many (excellent) things that exist on the Web.

My purpose here is to just wet your appetite by showing you how to get started, how to do a couple basic things, and then give you pointers to more documentation.

First, you need to be logged onto **physics-nix.stk.pacific.edu** as user *phystu*. You remember how to do that, right? .

At the prompt, type:

```
phystu@physics-nix[~]> math
```

You should get the following response:

```
Mathematica 7.0 for Linux x86 (32-bit)
Copyright 1988-2009 Wolfram Research, Inc.

In[1]:=
```

The Mathematica prompt is **In[1]:=** and this indicates that it is waiting for input.

If you DO NOT get this response, but instead get :

```
phystu@physics-nix[~]>math
Mathematica 9.0 for Linux x86 (32-bit)
Copyright 1988-2012 Wolfram Research, Inc.

/home/phystu/.Mathematica/Licensing/mathpass:2:
        The Mathematica license you are using has expired.
        Please contact Wolfram Research or an authorized
        Mathematica distributor to extend your license and
        obtain a new password.

/home/phystu/.Mathematica/Licensing/mathpass:3:
        The Mathematica license you are using has expired.
        Please contact Wolfram Research or an authorized
        Mathematica distributor to extend your license and
        obtain a new password.

/home/phystu/.Mathematica/Licensing/mathpass:4:
password file "/home/phystu/.Mathematica/Licensing/mathpass".
        Invalid password.

/home/phystu/.Mathematica/Licensing/mathpass:5:
password file "/home/phystu/.Mathematica/Licensing/mathpass".
        Invalid password.

Mathematica cannot find a valid password.

For automatic Web Activation enter your activation key
(enter return to skip Web Activation):
```

Just type **CTRL-d** to exit.

This indicates that someone else is using *Mathematica* on **physics- nix** at the moment.

I've been trying to save money by just using a 2-seat licencse for *Mma*.

If you get this response, logout by hitting **CTRL-d** , wait some time, log back in, and see if your colleague has finished using *Mathematica*

If you are still having trouble getting a free session, email me and I'll set up a schedule.

If you run into different problems, email me as usual.

Mma is a common abreviation for *Mathematic* and I will use it below.

Let's assume now you have the *Mma* prompt, **In[1]:=**, waiting for you to tell it something.

Try the simplest thing: type **1+1** ENTER

```
In[1]:= 1+1

Out[1]=  2

In[2]:=
```

Mma responds to your input, by computing the sum of 1+1. Bet you knew that one...

Derivatives

Let's try something more interesting. Most of you will have completed Differential Calculus (MATH 051 at Pacific).

I opened my freshman Calculus book and pulled out one of the problems I was assigned. Compute the following derivative:

$$\frac{d}{dx}\left(\frac{\sin x + \cos x}{\sin x - \cos x}\right)$$

In *Mma* we would use the *Derivative* operation: **D[...]** like this. At the *Mma* prompt, **In[2]:=** (it might have another number than 2 on your computer).

```
In[2]:= D[(Sin[x] + Cos[x])/(Sin[x] - Cos[x]), x]
```

Then type, ENTER.

Mma responds with:

```
                                                    2
        Cos[x] - Sin[x]     (Cos[x] + Sin[x])
Out[2]= ---------------- - --------------------
        -Cos[x] + Sin[x]                      2
                            (-Cos[x] + Sin[x])
```

In the **In[2]:=** command, do you see that the **D** (Derivative) operator acts on the function

(Sin[x]+Cos[x])/(Sin[x]-Cos[x]),

and takes the derivative with respect to **x**, which is the last argument to D[..., x]?

Let's look closer at the *Mma* syntax.

- *Mma* functions, operators, etc. (almost) *ALWAYS* begin with Capital letters: **D**[...], ***S**in[x], **C**os[Pi x/2], and so forth.

- Arguments to functions and operators are enclosed in *square* brackets: **Sin[**x**]**, **Exp[**x^2/a **]**, and so on.

  This is special to Mma. Gnuplot, and most programming languages, by contrast, use lower case functions with parentheses like this: **sin(x)**. So, it can sometimes be confusing when switching back and forth between programs.

Just be aware: *Mma* uses **CAPS** and **[]'s**

The answer given in **Out[2]=** is correct. *Mma* has applied the *Chain Rule* with the appropriate Trig derivatives, but it's not the simplest form for the result. For this, there is a handy function: **Simplify[]**

```
In[3]:= Simplify[%]

                2
Out[3]= -------------
        -1 + Sin[2 x]
```

Notice that I used the **%** (percent) here. The **%** symbol in *Mma* means "*the result of the last command*".

Note that sometimes the *Simplify* command helps and sometimes not. But it's worth a try.

Solving (non-linear) Equations

Let's see what *Mma* does with the equation we solved using gnuplot: $2x^2 - 3x + 1 = x$.

For this we use *Mma*'s **Solve[]** function. We have to give it the equation we wish to solve.

Mma's syntax for *an equation* is:

2x^2 - 3x + 1 == x

Notice I've used the **^** (caret) to mean exponentiation of a power.

Also, the **==** in the above statement indicates a relation of equality between two things. This is how we tell *Mma* that the thing above is an *equation*, as opposed to *an assignment*.

If we wrote **2x^2 -3x + 1 = x**, with **=** instead of **==**, we would be telling *Mma* to assign **x** (the RHS) to **2x^2 -3x +1** (the LHS), in the same way you assign a variable **y** the value **5**, by typing **y = 5**.

Next we pass this *equation* to the **Solve[]** function, and tell it to solve the *equation* for the variable **x**.

```
In[7]:= Solve[2x^2 - 3x + 1 == x, x]

                  2 - Sqrt[2]          2 + Sqrt[2]
Out[7]= {{x -> -----------}, {x -> -----------}}
                       2                    2
```

Very good. This is a *quadratic equation*, there are two solutions, and there they are. You can check by hand if you like.

Root Finding

How about another more interesting equation, say, the one from the end of Homework 5: using $V_0 = 50$?

$$\tan(2\sqrt{2E}) = \sqrt{\frac{V_0 - E}{E}}$$

If we just plug it into the above syntax, we get this:

```
In[8]:= Solve[Tan[2Sqrt[2E]]==Sqrt[(50-E)/E], E]

General::ivar: E is not a valid variable.

Out[8]= Solve[False, E]
```

Oops! In *Mma*, **E** is a protected character (like **Pi**). It's *Euler's* number: $E = e^1$.

Try this: Use *x* for **E** in the above equation:

```
In[9]:= Solve[Tan[2Sqrt[2x]]==Sqrt[(50-x)/x], x]

Solve::tdep: The equations appear to involve transcendental functions of the
    variables in an essentially non-algebraic way.
```

Still... no joy. But definately an interesting reply from *Mma*.

Mma is saying that this equation is *transcendental* and can't be solved in closed (algebraic or symbolic) form . Still we can ask it to *Numerically* solve the equation–that is, put numbers in the equation for *x*, and change *x* until we get a *numerical* answer that satisfies the equation to the accuracy we need. As long as there is only one variable, **x**, this will work.

We do this with the function: **FindRoot[eqn, {var, starting_guess }]**

Recall that a *Root* of an expression is an *x* value where the expression = 0. However *Mma* is smart enough to know that the *x* value that solves $\tan(2\sqrt{2x}) = \sqrt{\frac{V_0 - x}{x}}$ is a *Root* of the expression: $\tan(2\sqrt{2x}) - \sqrt{\frac{V_0 - x}{x}} = 0$.

We have to give *Mma* a little help with an initial guess. Let's start at *x = 0*.

```
In[12]:= FindRoot[Tan[2Sqrt[2x]]==Sqrt[(50-x)/x], {x,0}]

                                1
Power::infy: Infinite expression -- encountered.
                                0.
```

Duh. Should have seen that coming. The RHS is infinite at x=0. So, start with a small positive guess: 0.00001

```
In[13]:= FindRoot[Tan[2Sqrt[2x]]==Sqrt[(50-x)/x], {x,0.00001}]

Out[13]= {x ->  0.279726}
```

Booyah!

This should be close to what you got using the *graphical method* (zero-ing in on the place where the two sides are equal) with Gnuplot. My solutions using gnuplot are shown here.

The other solutions from *Mma* are:

```
In[15]:= FindRoot[Tan[2Sqrt[2x]]==Sqrt[(50-x)/x], {x,0.5}]
Out[15]= {x ->  2.5157}
In[17]:= FindRoot[Tan[2Sqrt[2x]]==Sqrt[(50-x)/x], {x,3}]
Out[17]= {x ->  6.97724}
In[19]:= FindRoot[Tan[2Sqrt[2x]]==Sqrt[(50-x)/x], {x,8}]
Out[19]= {x ->  13.6399}
In[22]:= FindRoot[Tan[2Sqrt[2x]]==Sqrt[(50-x)/x], {x,20}]
Out[22]= {x ->  22.4543}
In[23]:= FindRoot[Tan[2Sqrt[2x]]==Sqrt[(50-x)/x], {x,30}]
Out[23]= {x ->  33.309}
In[24]:= FindRoot[Tan[2Sqrt[2x]]==Sqrt[(50-x)/x], {x,45}]
Out[24]= {x ->  45.8083}
```

We get these by giving starting points near the expected answer, and we can find those expected answers by using *Gnuplot* to look at graphs of the functions.

Compare the *Mma* results above with my graphical solutions from HW 5.:

```
# State   Energy
 1         0.280
 2         2.502
 3         6.976
 4        13.643
 5        22.455
 6        33.307
 7        45.811
```

Integration

Before we leave *Mma*, let's do some integrals!

Start with the simplest one: $\int x dx$

```
In[1]:= Integrate[x,x]

         2
        x
Out[1]= --
        2
```

Nice. It's good to know things you learned in grade school are still true.

The syntax of the **Integrate** function in *Mma* is:

- *Indefinate integrals*: **Integrate[integrand , variable]** or
- *Definate integrals*: **Integrate[integrand , { variable , lowerLimit, upperLimit}]**

In the first case, like the trivial case above, we did an *indefinate integral* which returns a function (the anti-derivative).

The second version of the **Integrate[]** function is for a *definate integral*, when the integral has a lower and an upper bound, and the answer is a number.

Indefinate Integrals

Here's a more interesting integral:

$$\int \frac{x^2}{(1+x^2)^{\frac{3}{2}}} dx$$

This one is a little more complicated, and I bet you don't know the answer by looking at it.

Mma gives:

```
In[8]:= Integrate[x^2/(1 + x^2)^(3/2), x]

                x
Out[8]= -(------------) + ArcSinh[x]
                     2
          Sqrt[1 + x ]
```

Now, that would have taken a little work on paper..

Definate Integrals

On to *Definate Integrals*.

What is this one: $\int_0^1 x dx$

```
In[2]:= Integrate[x, {x, 0, 1}]

        1
Out[2]= -
        2
```

OK, let's be serious. Here's another one from my freshman calc book:

$$\int_0^\infty \frac{dx}{(1+x)\sqrt{x}}$$

We code this in *Mma* as follows:

```
In[4]:= Integrate[1/((1+x)Sqrt[x]), {x, 0, Infinity}]

Out[4]= Pi
```

Notice a couple things here:

Unlike Gnuplot, where we *MUST* explicitly put the ***** operator in for multiplication, as in **x*y**, *Mma* understands our usual algebraic expression: *xy* or *x y* (with just a space) as *x "times" y*.

Notice that the *Mma* code, **with the extra ()'s**

1/((1+x)Sqrt[x])

as we typed above, would be very different from this

1/(1+x)Sqrt[x].

without the extra **()'s**.

The first version evaluates to

$$\texttt{1/((1+x)Sqrt[x])} \quad \Rightarrow \quad \frac{1}{(1+x)\sqrt{x}}$$

while the second, **without** the extra set of **()'s** would be:

$$\texttt{1/(1+x)Sqrt[x]} \quad \Rightarrow \quad \frac{1}{(1+x)}\sqrt{x}$$

> **Warning:** Make sure you understand this suble difference!

Also, notice that we used one of the built in *Mma* constants above: **Infintity**. *Mma* also understands (Capital) **Pi**, as shown below.

$$\int_0^{\sqrt{\pi}} x \sin(x) dx$$

```
In[6]:= Integrate[ x Sin[x^2], {x, 0, Sqrt[Pi]}]

Out[6]= 1
```

I think you are getting the picture.

Have a look at this document

- Introduction to Mathematica (with emphasis on the Tricky points) by David Roberts

Then, if you want to learn more look at:

- Mathematica Tutorials at Wolfram.com

Also, a very nice, free website is Wolfram Inc.'s Integrals.

Check it out! Now that you know *Mma* syntax for functions (Sin[], etc.), you enter almost any function whose integral is known, and this site will return the answer to you.

That could come in handy, eh?

This gives you a taste of *symbolic math* on computers. I would love to go further and show you, for example, the *Mma* packages for doing *Riemann Tensor Calculus* in General Relativity, but your head would explode. Hopefully, this taste of *Mma* will get you started, at least enough for your freshman physics and math needs.

11.2 Linear Algebra

There is another commercial package for doing math which is very popular, particularly among engineers, that you should know about: MATLAB

Matlab stands for "*Matrix Laboratory*", which gives you some indication of its purpose. **Linear Algebra** is the mathematics of matrices and vectors, but that's almost the simplest way to think about it. It's the mathematics of *systems of linear equations* (which can be written in matrix form, no matter how many, and *many* other related topics.

In fact, we will be not be using *Matlab*, but rather a free "clone" with much of *Matlab*'s functionality, called *Octave*. More on that below.

Back to *Linear Algebra*.

Believe me, linear algebra is a *FREEKING DEEP* subject. It should be required of all physics and engineering students, and included in PACS. Almost everything you have learned in advanced math has a linear algebra connection.

This is very useful for many things:

- Many data sets can be thought of as a long vector. Calculations on these data can be done by MATLAB. An example might be Stock Market data over some period of time.
- Systems of linear equations. Many other problems, such as partial differential equations, statics problems, fluid dynamics, etc. can be given a matrix/linear algebra form. See: Finite Element Methods
- Many things, like the set of all periodic functions $f(x) = f(x + L)$ can be represented as a set of vectors in an *infinite dimensional vector space*. That's *Fourier Analysis*...
- Signal Processing. MATLAB has many tools for signal analysis, Fourier transforms, audio and image processing (after all an image is just a matrix of pixel values–see what I mean about most problems having a linear algebra representation?).
- Representing *rotations*, *tranlations* (both from one language to another as well as the physics meaning: movement from place to another in space. Actually, if you think about Heisenberg's Matrix Formulation of Quantum Mechanics, in fact *Change* in general can be represented by an infinite dimensional matrix in the space of all possibilities...
- Representing *symmetry*.

...just to name a few.

We are fortunately going to take baby steps into linear algebra.

Whereas Mathematica was a tool whose primary use is for **symbolic** (algebraic) calculations, MATLAB is fundamentally **numerical**. That means that it manipulates matrices and vectors as tables and arrays of numbers (like **[1, -1, 3]**), not as algebraic symbols (like *a, b, g*, and *x*) as Mathematica did.

11.2.1 Matrix Algebra

First, let's make sure you remember what *matrix* × *matrix*, and *matrix* × *vector multiplication* is.

If you are rusty on multiplying *matrices* (the plural of *matrix*), have a look at the Wikipedia page on Matrix Multiplication. Like many Wikipedia pages, it is rather dense. But it's pretty complete.

Anothere place to brush up on your *matrix* × *matrix multiplication* is Kahn Academy's page on matrix algebra.

Warning: Make sure you understand the basics of matrix algebra before going on.

In particular, you need to be able to understand the problems in this quick little quiz. You can check your answers below.

If

$$\mathbf{M} = \begin{pmatrix} 2 & -1 & 0 \\ 3 & -2 & 4 \\ -1 & 1 & 2 \end{pmatrix}$$

What is $\mathbf{M}^2 = ?$

i.e., what is the *matrix* × *matrix* product?

$$\mathbf{M}^2 = \begin{pmatrix} 2 & -1 & 0 \\ 3 & -2 & 4 \\ -1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} 2 & -1 & 0 \\ 3 & -2 & 4 \\ -1 & 1 & 2 \end{pmatrix} = ?$$

Go that one? Good.

What about this *matrix* $\times$ *vector* product?

We will use a *single-column matrix* for the *vector* $\vec{v}$ is

$$\vec{v} = \begin{pmatrix} 2 \\ 1 \\ -1 \end{pmatrix}$$

What is $\mathbf{M} \times \vec{v}$?

(Also, we will usually leave the $\times$ symbol out)

$$\mathbf{M}\vec{v} = \begin{pmatrix} 2 & -1 & 0 \\ 3 & -2 & 4 \\ -1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ -1 \end{pmatrix} = ?$$

Check your answers here before going on.

11.2.2 A Linear System of Equations

Here's a typical linear algebra problem. Let's say you need to solve these three coupled *linear* equations for the variables x, y, and z.

$$\begin{array}{rrrcr} x & +2y & -z & = & 4 \\ 2x & & -2z & = & 1 \\ x & +y & & = & -4 \end{array}$$

These are *linear* equations because the variables x, y, and z appear in linear form, *i.e.* raised to the 1st power, x^1, y^1, z^1, as opposed to quadratic x^2, cubic y^3, or other *non-linear* ($\neq = 1$) powers.

You could also write this in matrix form.

Let

$$\mathbf{A} = \begin{pmatrix} 1 & +2 & -1 \\ 2 & 0 & -2 \\ 1 & +1 & 0 \end{pmatrix}$$

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

and

$$\vec{b} = \begin{pmatrix} 4 \\ 1 \\ -4 \end{pmatrix}$$

Remembering the rules of *matrix* $\times$ *vector* multiplication.

We can write our three linear equations *very* succinctly as $\mathbf{A}\vec{r} = \vec{b}$.

$$\mathbf{A}\vec{r} = \begin{pmatrix} 1 & +2 & -1 \\ 2 & 0 & -2 \\ 1 & +1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x & +2y & -z \\ 2x & & -2z \\ x & +y & \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ -4 \end{pmatrix} = \vec{b}$$

where $\mathbf{A}$ is the matrix on the left, $\vec{r}$ is the (column-matrix) vector of unknowns [x, y, z], and $\vec{b}$ is the vector on the right side.

Let's now define another (unknown) matrix: $\mathbf{A}^{-1}$ which will be the **inverse of A**.

If $\mathbf{A}$ was a normal *scalar* variable (like 5, or π), let's call it a, then the *inverse* of a would be $a^{-1} = \frac{1}{a}$. Then $a^{-1}a = 1$

For *matrices*, given $\mathbf{A}$, we want to find another matrix $\mathbf{A}^{-1}$, defined such that if you multiply $\mathbf{A}^{-1} \times \mathbf{A}$ you get something like $\mathbf{1}$ (One).

In matrix algebra, $\mathbf{1}$ (One) is called **The Identity Matrix**. It is the matrix that you can multiply times *ANY* matrix $\mathbf{M}$, and get the same matrix $\mathbf{M}$ as the result.

That's what 1 (One) does for *scalar* numbers: for any a, $1 \times a = a$.

The *Identity Matrix*, $\mathbf{I}$, looks like this (in 3-dimensions):

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Try it yourself:

Multiply

$$\mathbf{I} \times \mathbf{M} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & -1 & 0 \\ 3 & -2 & 4 \\ -1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & -1 & 0 \\ 3 & -2 & 4 \\ -1 & 1 & 2 \end{pmatrix} = \mathbf{M}$$

Please check that this is true.

The **Inverse of matrix A** is another matrix, which we write as $\mathbf{A}^{-1}$, such that if you multiply $\mathbf{A}^{-1} \times \mathbf{A}$ you get $\mathbf{I}$, the Identity matrix.

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

Now, if we knew the *inverse*, $\mathbf{A}^{-1}$, of $\mathbf{A}$, we could use this to find the vector of unknowns $\vec{r}$ in our original problem—the linear set of equatoins we started with.

Because, if

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

then, multiply both sides of the equation above, $\mathbf{A}\vec{r} = \vec{b}$, by $\mathbf{A}^{-1}$

$$\begin{aligned} \mathbf{A}\vec{r} &= \vec{b} \\ \mathbf{A}^{-1}\mathbf{A}\vec{r} &= \mathbf{A}^{-1}\vec{b} \\ \mathbf{I}\vec{r} &= \mathbf{A}^{-1}\vec{b} \\ \vec{r} &= \mathbf{A}^{-1}\vec{b} \end{aligned}$$

Wow! All we have to do is find the matrix $\mathbf{A}^{-1}$ and we can use it to find our *unknown* vector $\vec{r}$, and *solve the set of linear equations*. The solution is just

$$\vec{r} = \mathbf{A}^{-1}\vec{b}$$

On to Octave.

11.2.3 A Taste of Octave

We will use a free "clone" of *Matlab*, called **Octave**. It doesn't have all the fanciest features of *Matlab*–it actually uses *Gnuplot* for its graphics, but it does have much of the numerical linear algrebra capability. Octave comes standard on *Fedora Scientific*., so you can run it on your own computer; you don't have to log on to another computer.

To start **Octave**, simply open a terminal and type:

`octave`.

You should see this:

```
GNU Octave, version 3.0.3
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying conditions.
...

octave:1>
```

Once again, you get a prompt:

octave:1>

waiting for input.

At the **octave:1>** prompt, type:

```
octave:1> A = [1, 2, -1; 2, 0, -2; 1, 1, 0]
```

When you hit ENTER, *octave* responds with

```
A =

   1   2  -1
   2   0  -2
   1   1   0
```

You have just entered the *matrix form of **A**.

Notice the *octave syntax* for matrices:

- elements in a *row* are separated by *commas* [1 **,** 2 **,** -1; 2 **,** 0...]
- *rows* are separated by semicolons; [1, 2, -1 **;** 2, 0, -2 **;** 1, 1, 0]

Next enter the $\vec{b}$ *vector*. $\vec{b}$ is a *single-column matrix*, with only one element per *row*, so each entry is separated by a semicolon "**;**"

```
octave:2> b = [4; 1; -4]
b =

   4
   1
  -4
```

Now, to find $\mathbf{A}^{-1}$, the *inverse of $\mathbf{A}$, type

```
octave:3> Ainv = inv(A)
Ainv =

  -0.50000   0.25000   1.00000
   0.50000  -0.25000  -0.00000
  -0.50000  -0.25000   1.00000
```

We have defined a new matrix named **Ainv** and called octave's *matrix inverse* function: **inv()**.

Octave uses the star * for matrix multiplication.

Let's check that $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$.

Mathematically, we are asking if

$$\mathbf{A}^{-1}\mathbf{A} = \begin{pmatrix} -0.5 & 0.25 & 1.0 \\ 0.5 & -0.25 & 0.0 \\ -0.5 & -0.25 & 1.0 \end{pmatrix} \begin{pmatrix} 1 & +2 & -1 \\ 2 & 0 & -2 \\ 1 & +1 & 0 \end{pmatrix} = \mathbf{I} \quad ?$$

You can check this "*by hand*", or use *Octave*.

At the **octave:>** prompt type

```
octave:4> Ainv * A
ans =

   1   0   0
   0   1   0
   0   0   1
```

We have shown that indeed, **Ainv** = $\mathbf{A}^{-1}$.

Now let's solve those coupled linear equations. Type:

```
octave:5> Ainv * b
ans =

  -5.7500
   1.7500
  -6.2500
```

There you go:

The unknowns, x, y, and z are

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \mathbf{A}^{-1}\vec{b} = \begin{pmatrix} -0.5 & 0.25 & 1.0 \\ 0.5 & -0.25 & 0.0 \\ -0.5 & -0.25 & 1.0 \end{pmatrix} \begin{pmatrix} 4 \\ 1 \\ -4 \end{pmatrix} = \begin{pmatrix} -5.75 \\ 1.75 \\ -6.25 \end{pmatrix}$$

In other words, the solution of the original set of equations is

$$\begin{pmatrix} x & = & -5.75 \\ y & = & 1.75 \\ z & = & -6.25 \end{pmatrix}$$

Pretty easy. You can verify that these solve the above equations.

$$\begin{array}{cccc} x & +2y & -z & = & 4 \\ 2x & & -2z & = & 1 \\ x & +y & & = & -4 \end{array}$$

Consider the last equation of the set:

$x + y = -4$

Indeed: $-5.75 + 1.75 = -4$.

Probably this doesn't seem like we did rocket science, but imagine if we had a system of 500 coupled linear equations! It would be just as easy, except for typing in the matrix and vector, which you probably would program Octave to construct the vector for you.

At least now, you have a little taste of what Octave can do.

11.3 Homework

Homework 11 is here

$$\mathbf{M}^2 = \mathbf{MM} = \begin{pmatrix} 2 & -1 & 0 \\ 3 & -2 & 4 \\ -1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 2 & -1 & 0 \\ 3 & -2 & 4 \\ -1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -4 \\ -4 & 5 & 0 \\ -1 & 1 & 8 \end{pmatrix}$$

And

$$\mathbf{M}\vec{v} = \begin{pmatrix} 2 & -1 & 0 \\ 3 & -2 & 4 \\ -1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \\ -3 \end{pmatrix}$$

Click here to jump back to the quiz