COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

High Performance Computing Group

http://hpc.mines.edu

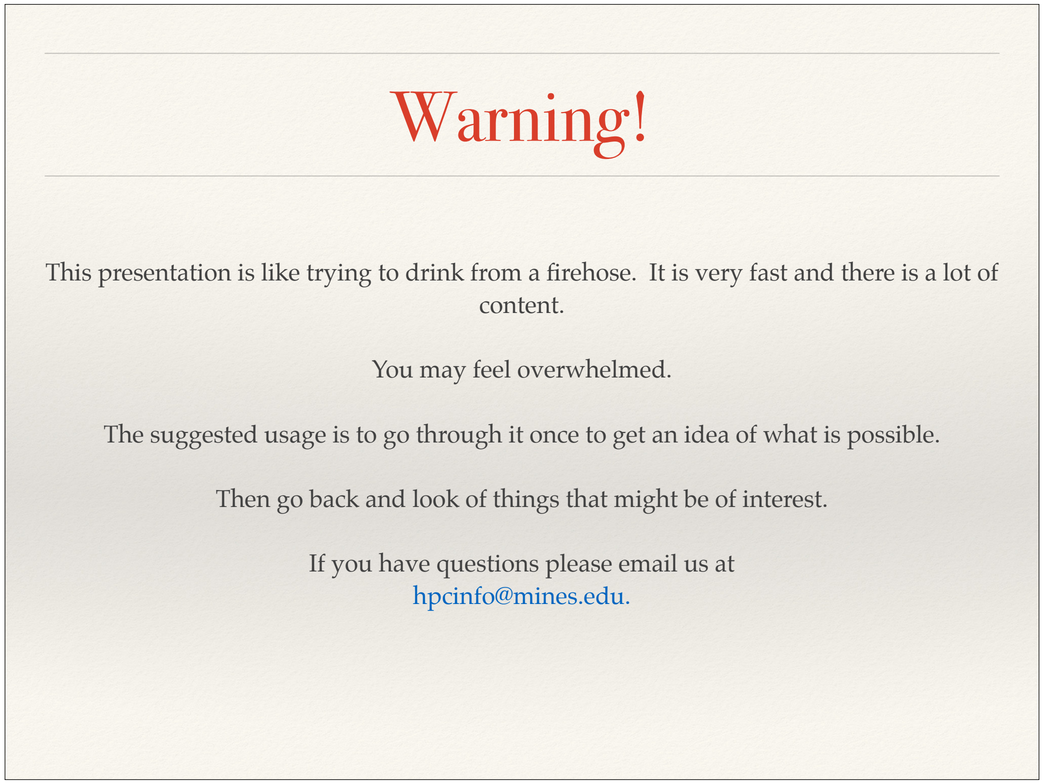
Timothy H. Kaiser, Ph.D.
tkaiser@mines.edu

This tutorial is presented by the High Performance Computing Group at the Colorado School of Mines.

Linux for HPC

Timothy H. Kaiser, Ph.D.
Spring 2014

Welcome to this talk on Linux for High Performance Computing. While our primary interest is in getting people up to speed to be able to use HPC platforms, this talk will help people become familiar with this important and common type of operating system. In particular, it will be helpful for people coming from the Windows or Macintosh systems.



Warning!

This presentation is like trying to drink from a firehose. It is very fast and there is a lot of content.

You may feel overwhelmed.

The suggested usage is to go through it once to get an idea of what is possible.

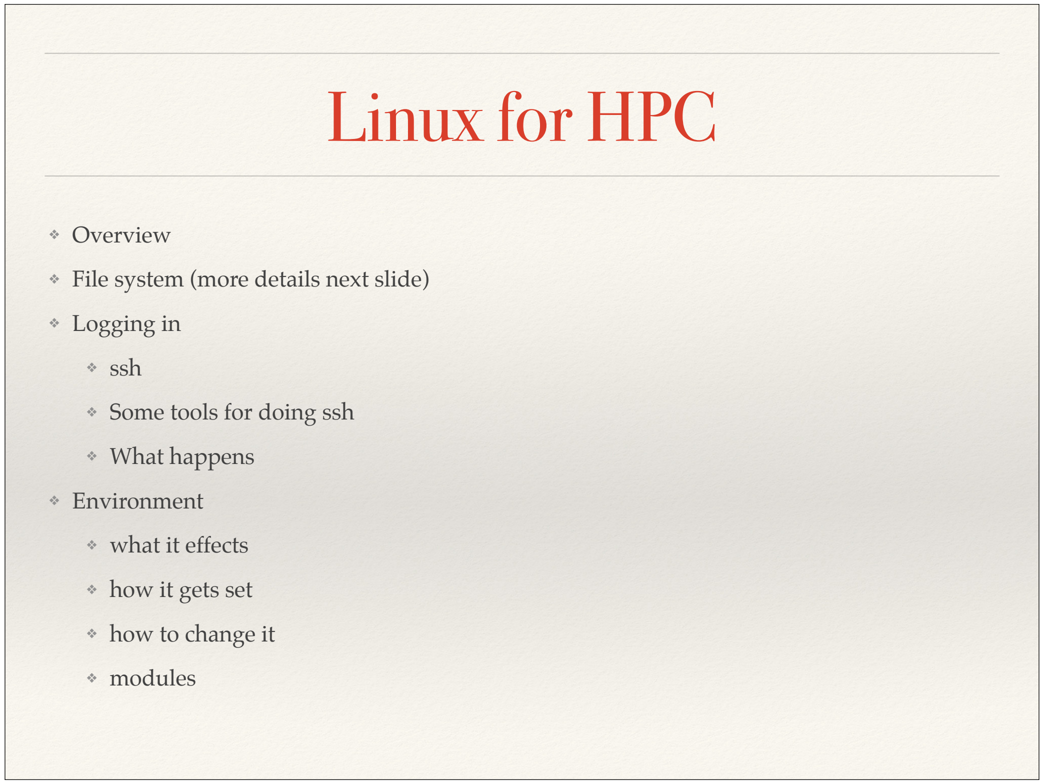
Then go back and look of things that might be of interest.

If you have questions please email us at
hpcinfo@mines.edu.

This presentation is like trying to drink from a firehose. It is very fast and there is a lot of content. You may feel overwhelmed.

The suggested usage is to go through it once to get an idea of what is possible. Then go back and look of things that might be of interest.

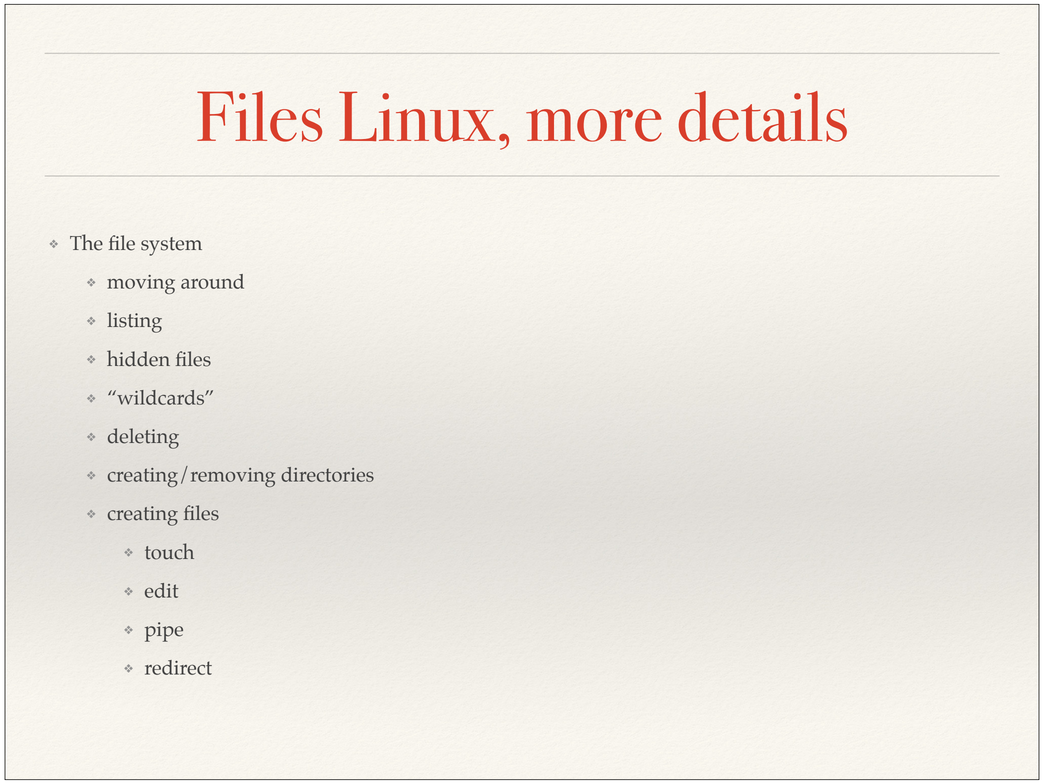
If you have questions please email us at
hpcinfo@mines.edu.



Linux for HPC

- Overview
- File system (more details next slide)
- Logging in
  - ssh
  - Some tools for doing ssh
  - What happens
- Environment
  - what it effects
  - how it gets set
  - how to change it
  - modules

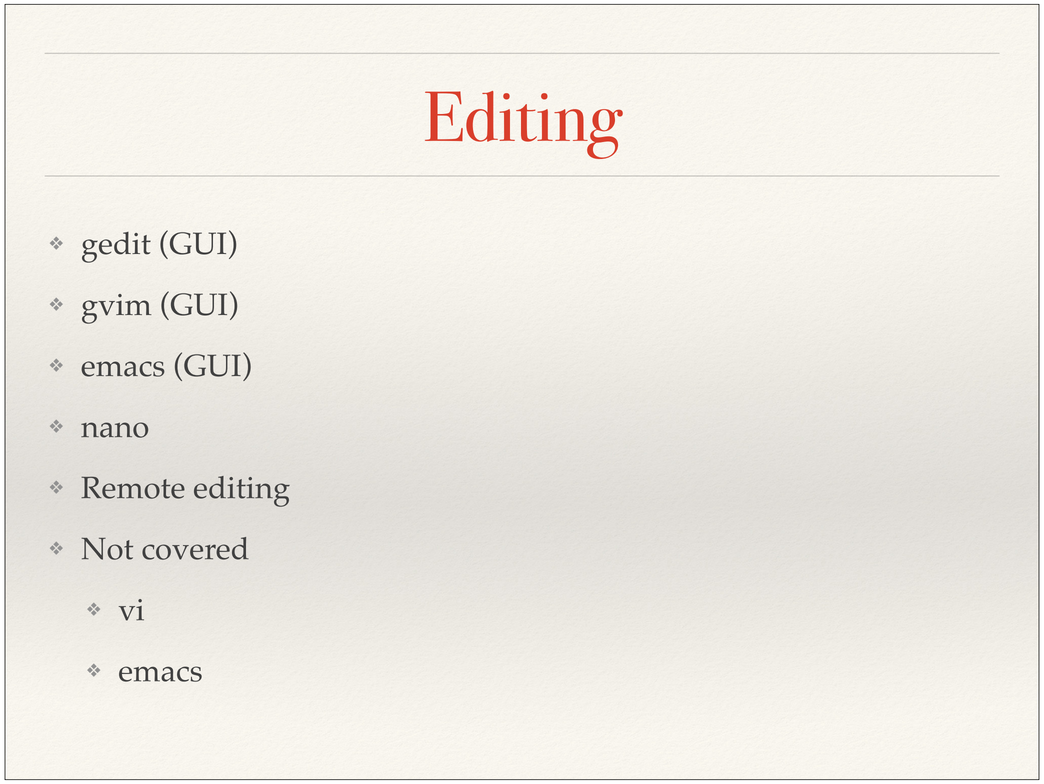
We have a lot to cover. We will start with an overview talking first about the file system. Then we will talk about how you get on to a Linux system using ssh. The environment in which you work determines how you interact with the system so we will talk about how it is set up and how it can be customized.



Files Linux, more details

- The file system
  - moving around
  - listing
  - hidden files
  - “wildcards”
  - deleting
  - creating/removing directories
  - creating files
    - touch
    - edit
    - pipe
    - redirect

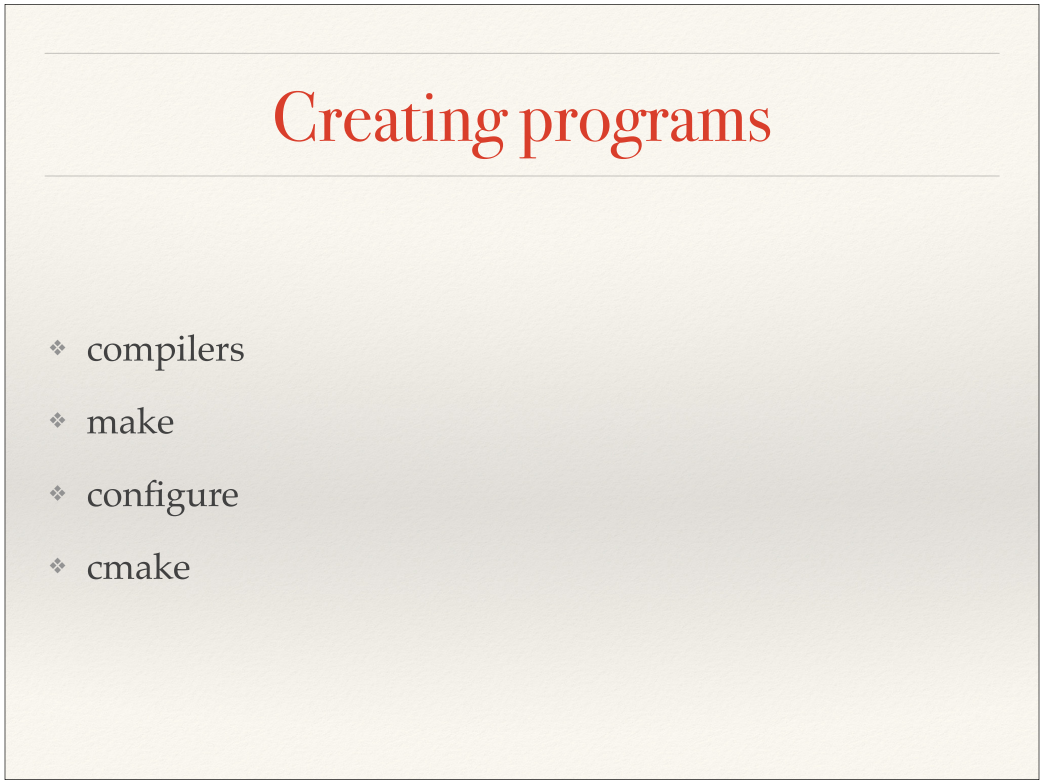
Obviously, the file system is where you store your information. It has a particular structure and We will talk about how to navigate around in it, see what you have, manipulate files and some useful tricks for creating files.



Editing

- gedit (GUI)
- gvim (GUI)
- emacs (GUI)
- nano
- Remote editing
- Not covered
  - vi
  - emacs

Editors are used to change or view files. We will talk about several editors and look at methods for editing files remotely. We will look at several easy to use editors. The V I editor is available on all Linux platforms but we will not be covering it because, while it is powerful, it is not that easy to learn.



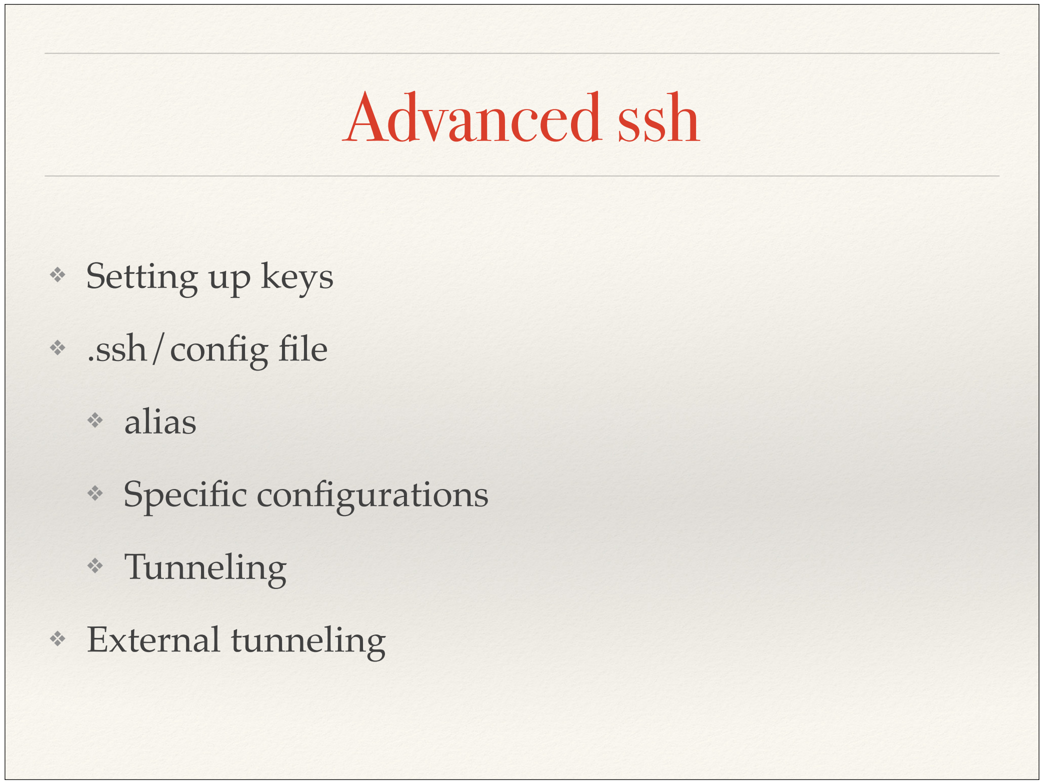
Creating programs

- compilers
- make
- configure
- cmake

If you are writing your own programs or build programs other people provide to you, you will be using compilers. Compilers take human readable source code and convert it into binary executables that can be run on the computer.

The make program provides a method to pass instructions to compilers in a way that allows for easier program builds.

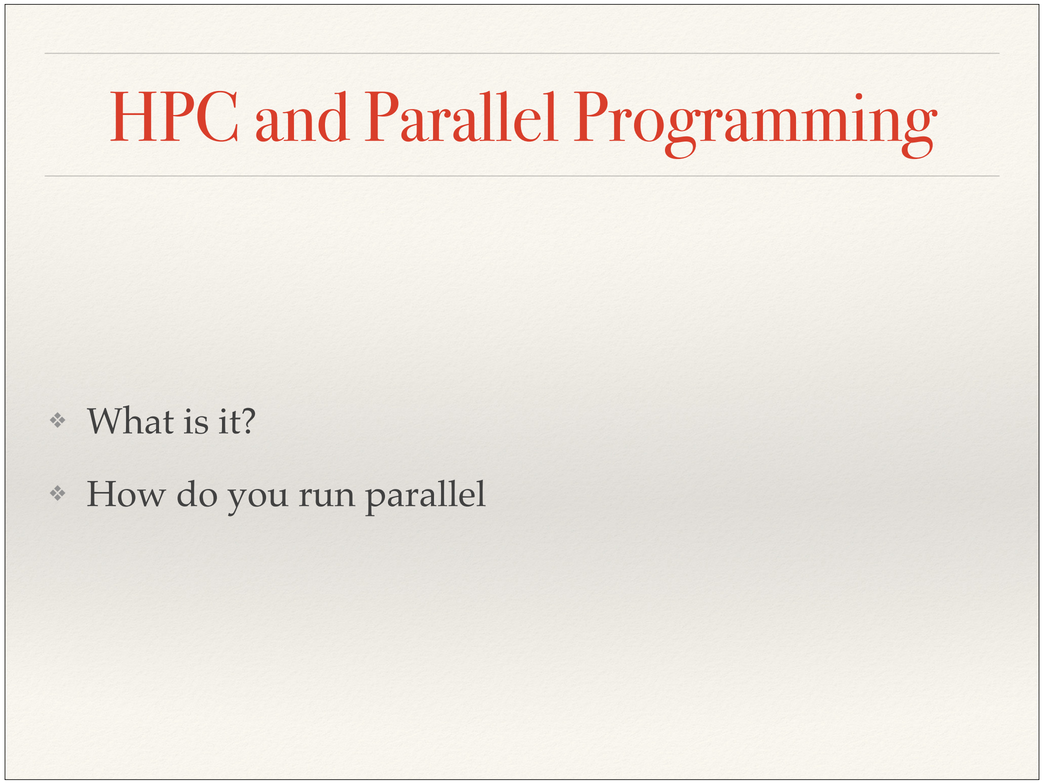
The make program takes a makefile as input. Configure and cmake are programs that allow for creating makefiles semiautomatically.



Advanced ssh

- Setting up keys
- .ssh/config file
  - alias
  - Specific configurations
  - Tunneling
- External tunneling

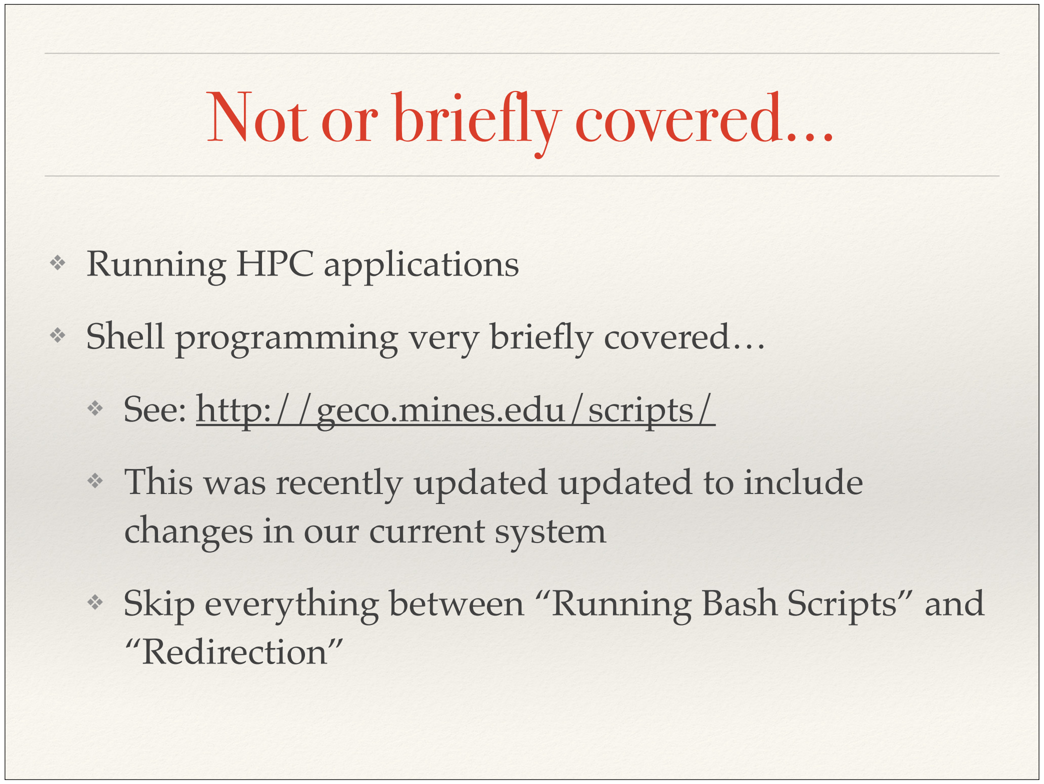
Ssh is the program and protocol that you use to connect to our machines from external boxes. There are a number of ways you can set up ssh to make your life easier. We will show you how to do things like passwordless access and how to access a machine behind a fire wall without using VPN.



HPC and Parallel Programming

- What is it?
- How do you run parallel

We will briefly talk about parallel programming. We will describe the concept and show how you run in parallel. Scripting for parallel applications and writing parallel applications is a subject for other talks.



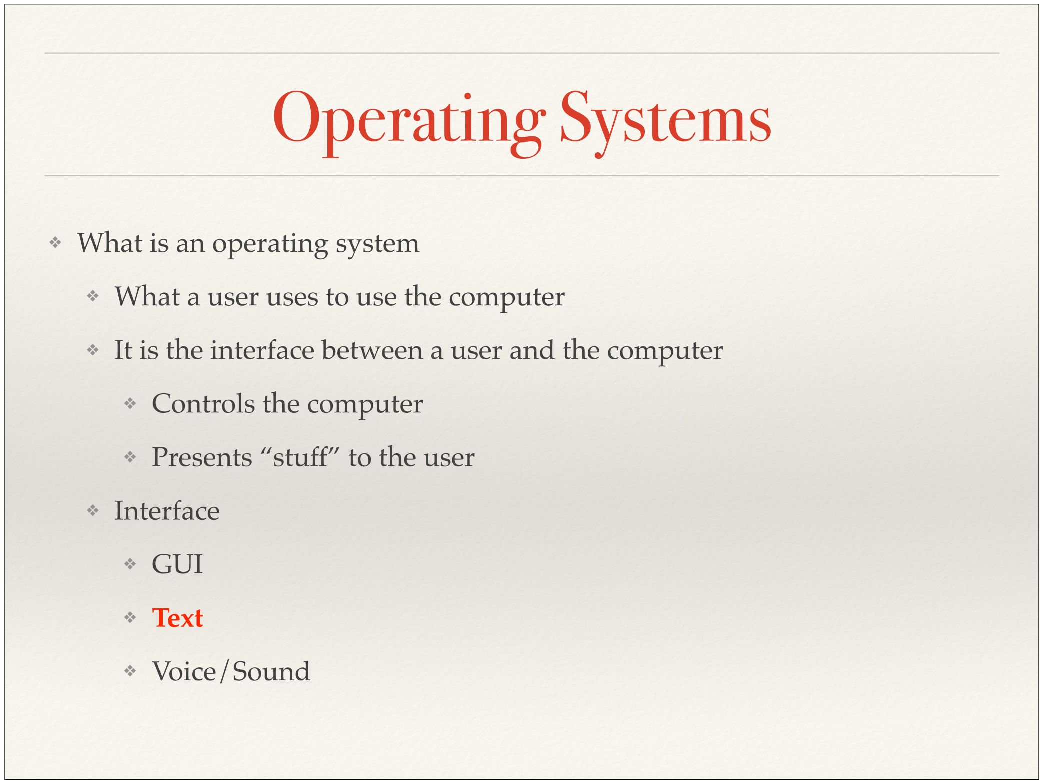
Not or briefly covered…

- Running HPC applications
- Shell programming very briefly covered…
  - See: http://geco.mines.edu/scripts/
  - This was recently updated updated to include changes in our current system
  - Skip everything between “Running Bash Scripts” and “Redirection”

As we said, we will not cover High Performance computing, that is parallel applications in detail. We will also not talk in detail about shell programming.

Let's get going…

Let's get to it!



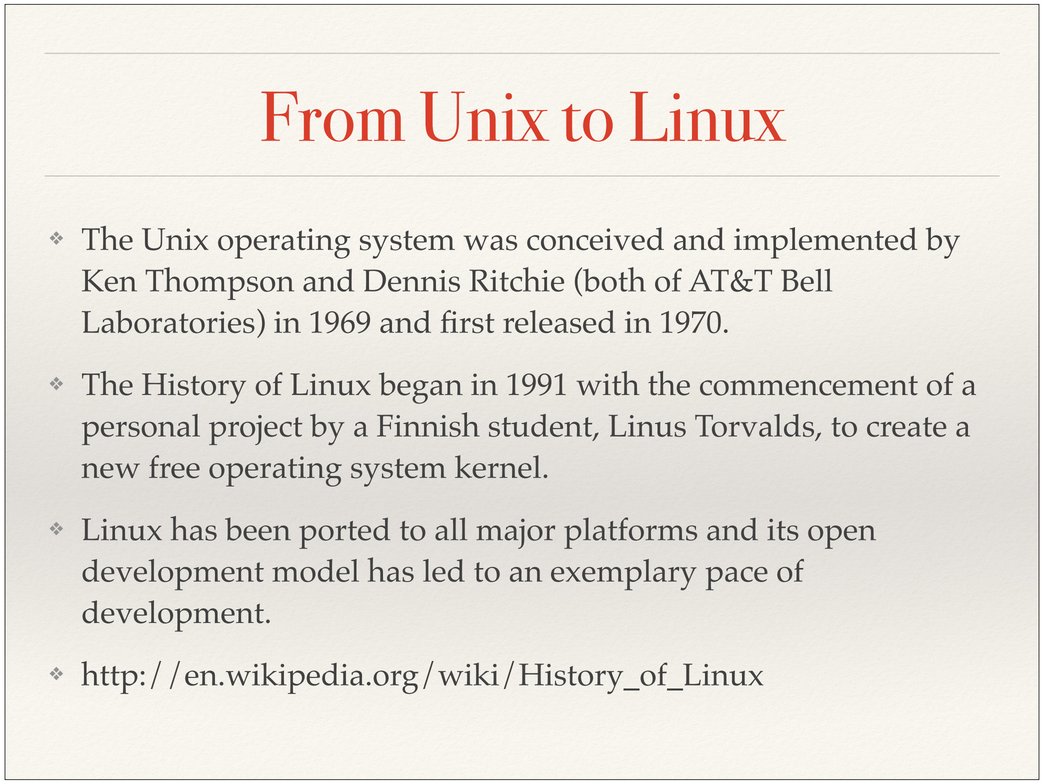
Operating Systems

- What is an operating system
  - What a user uses to use the computer
  - It is the interface between a user and the computer
    - Controls the computer
    - Presents “stuff” to the user
  - Interface
    - GUI
    - **Text**
    - Voice/Sound

A computer operating system is a program, or rather a collection of programs that are running on a computer that allows people to interact with it. It allows the computer to be controlled ,it presents information to the user, and allows the user to pass information and issue instructions.

We will be talking mainly about interacting via a text based interface. Today, many computer OS’s present a G U I or graphical user interface, for example the Mac OS is for the most part GUI based but you can interface using text commands also. Siri is an example of a voice interface for a computer.

By the way, underneath what the user sees the operating system on the iPhone is very Linux like. Also, there are I O S apps that allow text based interaction with Linux systems.

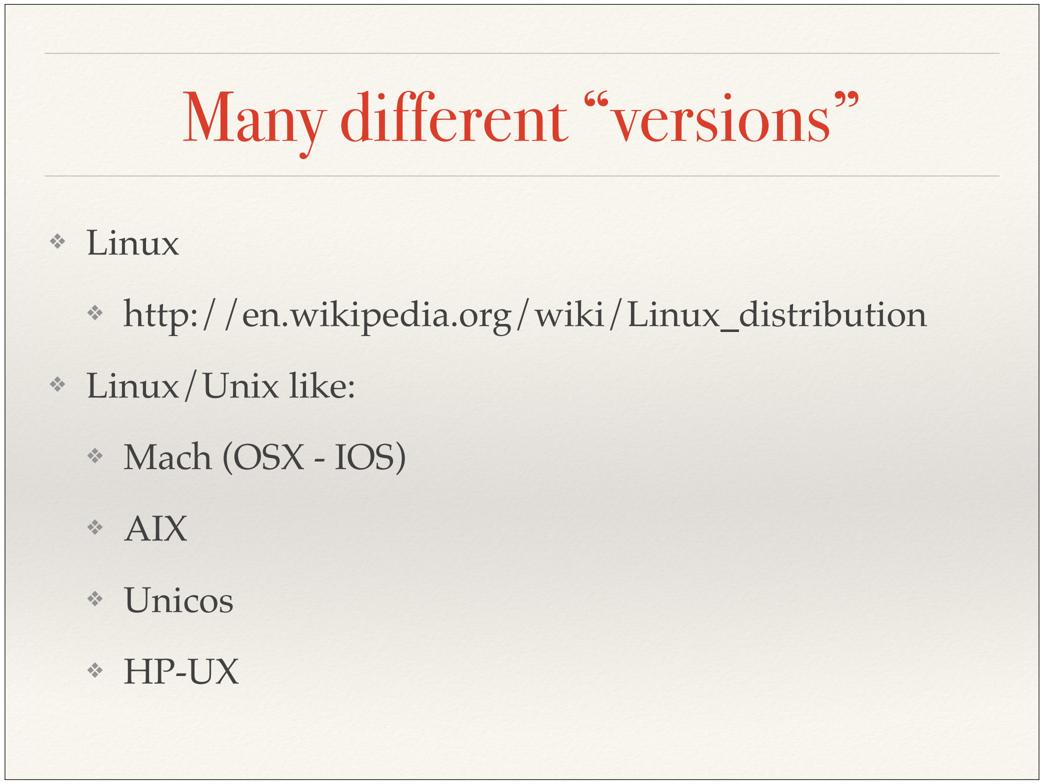


From Unix to Linux

- The Unix operating system was conceived and implemented by Ken Thompson and Dennis Ritchie (both of AT&T Bell Laboratories) in 1969 and first released in 1970.
- The History of Linux began in 1991 with the commencement of a personal project by a Finnish student, Linus Torvalds, to create a new free operating system kernel.
- Linux has been ported to all major platforms and its open development model has led to an exemplary pace of development.
- http://en.wikipedia.org/wiki/History_of_Linux

The Unix operating system has been around for a long time and was originally a product of AT&T. It was written by the creators of the C programming language. C and Unix have grown up together. Linux was first developed in 1991 when Linux Torvalds created the core or kernel of the the OS.

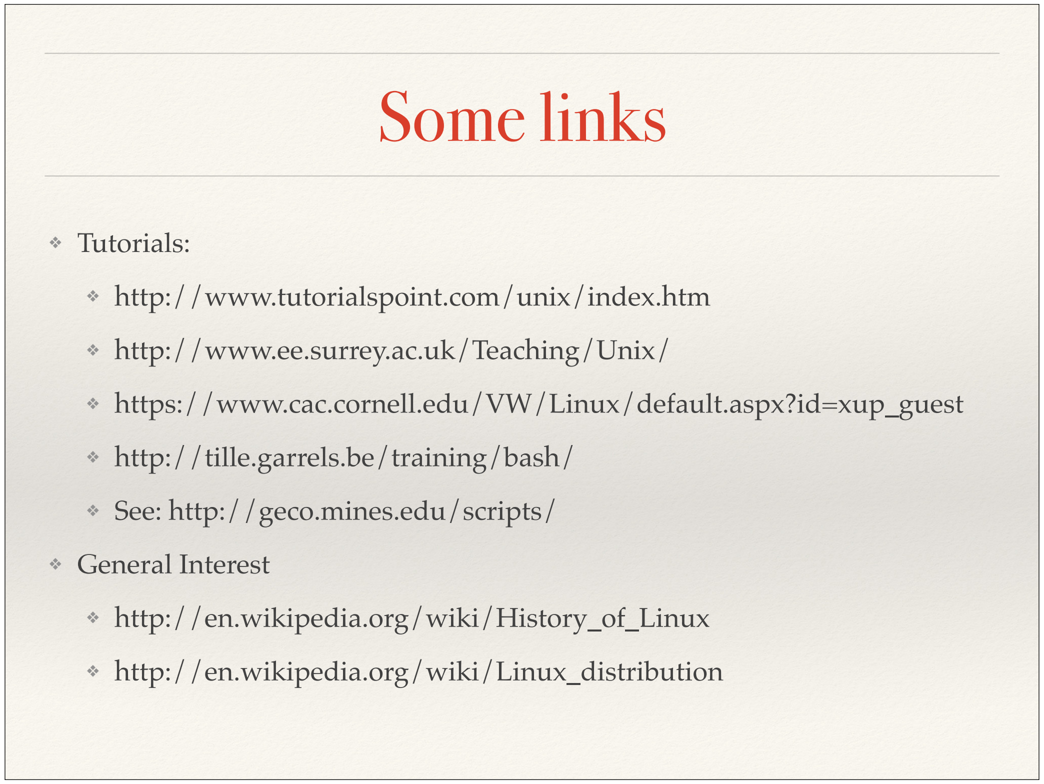
The wikipedia article sighted here has a very good history. One important point is that it has been ported to many many different platforms and is the inspiration of many more operating systems.



Many different "versions"

- Linux
  - http://en.wikipedia.org/wiki/Linux_distribution
- Linux/Unix like:
  - Mach (OSX - IOS)
  - AIX
  - Unicos
  - HP-UX

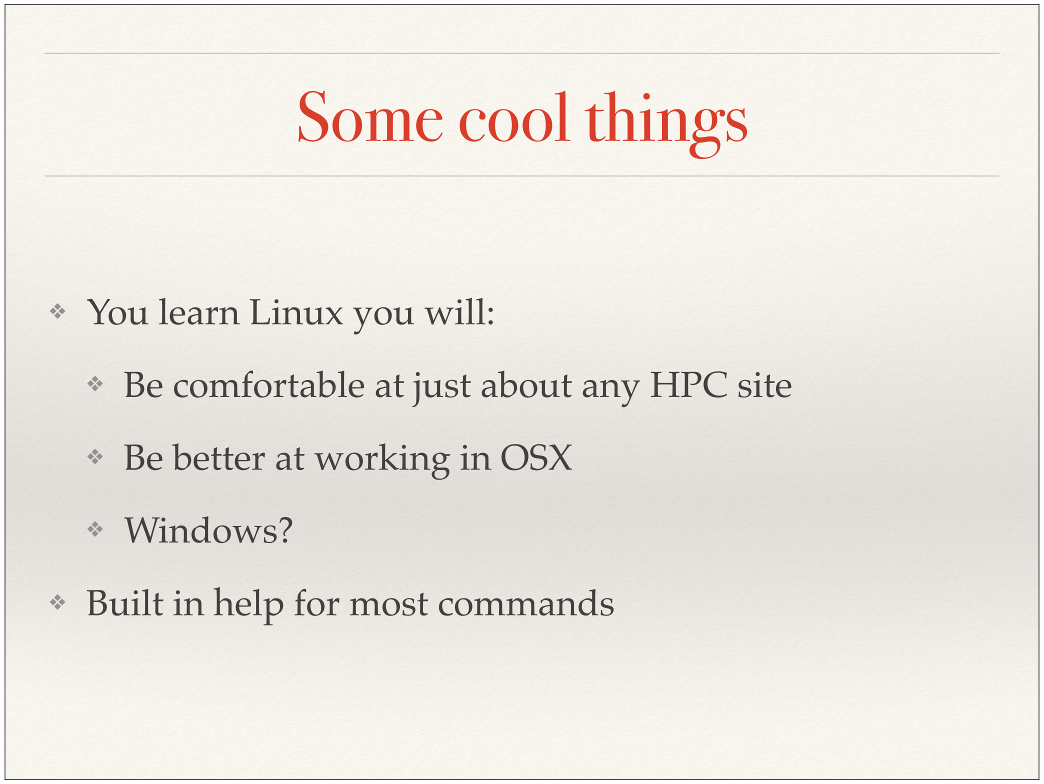
There are many different versions of Linux and Linux like operating systems. This wikipedia article talks about Linux distribution and versions. A distribution is just a collection of the software that makes up the operating system. There is a interesting graphic on this web page that shows a timeline representing the development of various Linux distributions. Some of the important distributions include: Cent, RedHat, Ubuntu, and Sūsé.



Some links

- Tutorials:
  - http://www.tutorialspoint.com/unix/index.htm
  - http://www.ee.surrey.ac.uk/Teaching/Unix/
  - https://www.cac.cornell.edu/VW/Linux/default.aspx?id=xup_guest
  - http://tille.garrels.be/training/bash/
  - See: http://geco.mines.edu/scripts/
- General Interest
  - http://en.wikipedia.org/wiki/History_of_Linux
  - http://en.wikipedia.org/wiki/Linux_distribution

There are many tutorials on using Linux. Some are very high level and some go into great detail. Here are a few we have found that look pretty good.



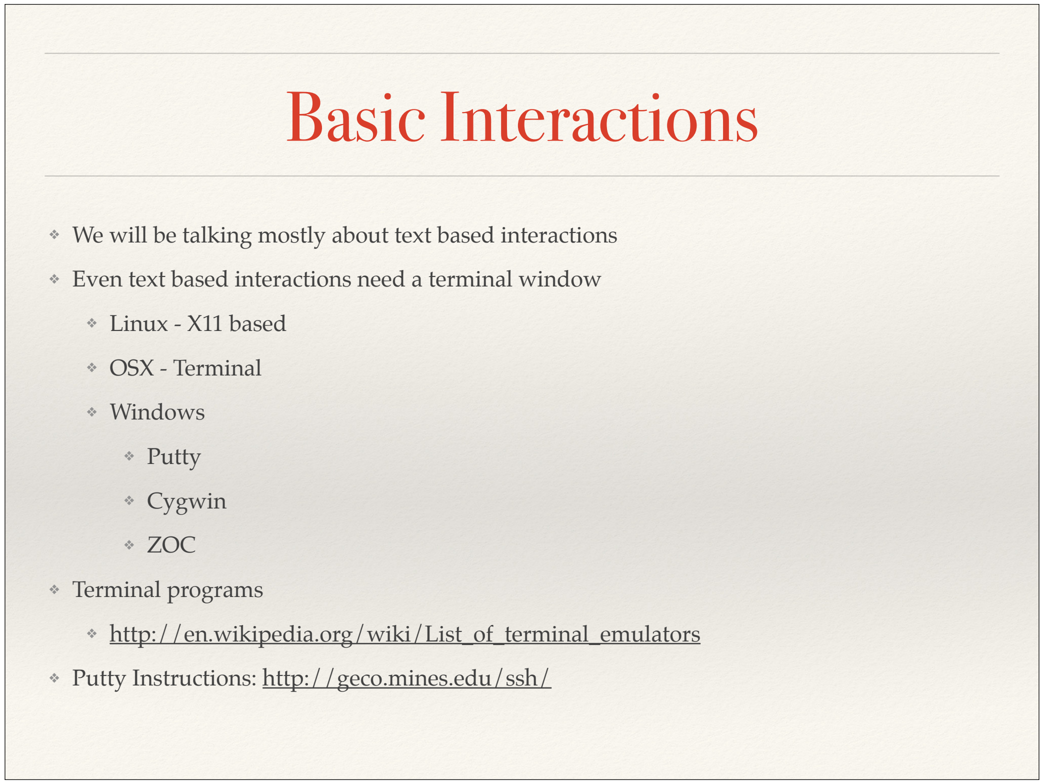
Some cool things

- You learn Linux you will:
  - Be comfortable at just about any HPC site
  - Be better at working in OSX
  - Windows?
- Built in help for most commands

After you get a feel for Linux you will be comfortable at just about any high performance computing site, simply because most HPC platforms run Linux or a Linux like operating system.

You will be surprised that you will feel more comfortable using the lower level features of the Mac’s OS X.

As far as windows, you may feel a bit more comfortable or you may want to start using Linux even on your laptop.



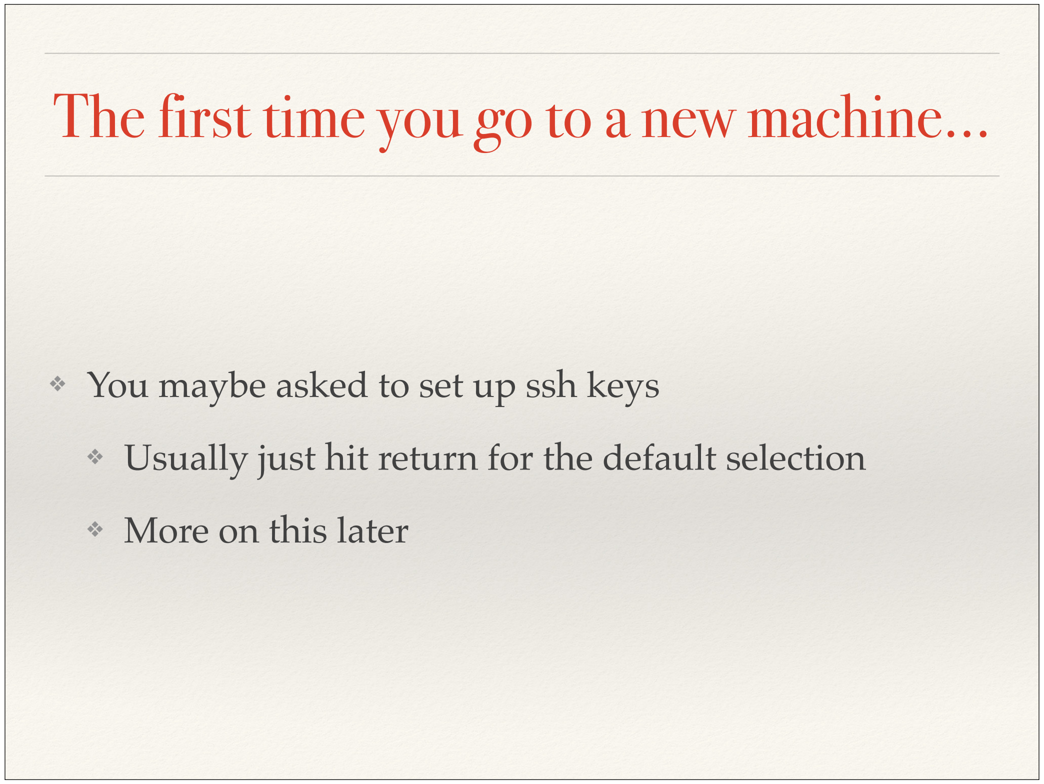
Basic Interactions

- We will be talking mostly about text based interactions
- Even text based interactions need a terminal window
  - Linux - X11 based
  - OSX - Terminal
  - Windows
    - Putty
    - Cygwin
    - ZOC
- Terminal programs
  - http://en.wikipedia.org/wiki/List_of_terminal_emulators
- Putty Instructions: http://geco.mines.edu/ssh/

As we said, we will be talking mostly about text base interactions. But even with text based interactions you need somewhere to type your text and for output to be shown. You need some type of terminal window. The type of terminal window and the program you will run to get a window is dependent on where you will be coming from, that is your machine. If you are coming from another Linux system you will most likely have an X11 based terminal. On OS X there is a terminal program. It's in the Utilities folder inside of the Applications folder.

There are a number of programs available under windows for creating a terminal. Putty is nice because it is free, small and contains all that is required. Cygwin tries to put a Linux like head on Windows and it can be very large if you install the whole package. ZOC is a commercial package that contains many features. It is also available for OS X.

Although it is a bit dated, we do have a web page describing using Putty under windows and the Terminal window under OS X



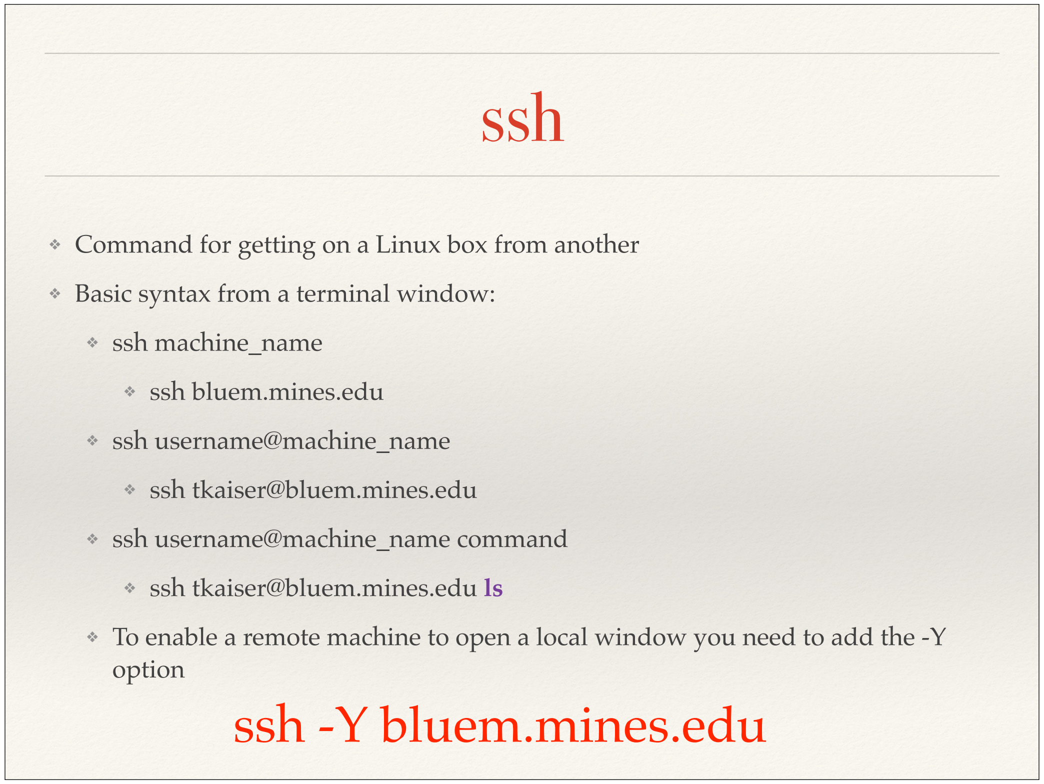
The first time you go to a new machine…

- You maybe asked to set up ssh keys
  - Usually just hit return for the default selection
  - More on this later

The first time you get onto a machine you maybe asked to set up ssh keys. Usually you can just hit return for the default selection.

Recently, we have changed the way we set up accounts on Blue M. You will not need to do this. It will have been done for you.

Also, we have also recently automated access from Blue M to A U N and Mc2. You can now access these other machines from Blue M without entering a password.



ssh

- Command for getting on a Linux box from another
- Basic syntax from a terminal window:
  - ssh machine_name
    - ssh bluem.mines.edu
  - ssh username@machine_name
    - ssh tkaiser@bluem.mines.edu
  - ssh username@machine_name command
    - ssh tkaiser@bluem.mines.edu **ls**
  - To enable a remote machine to open a local window you need to add the -Y option

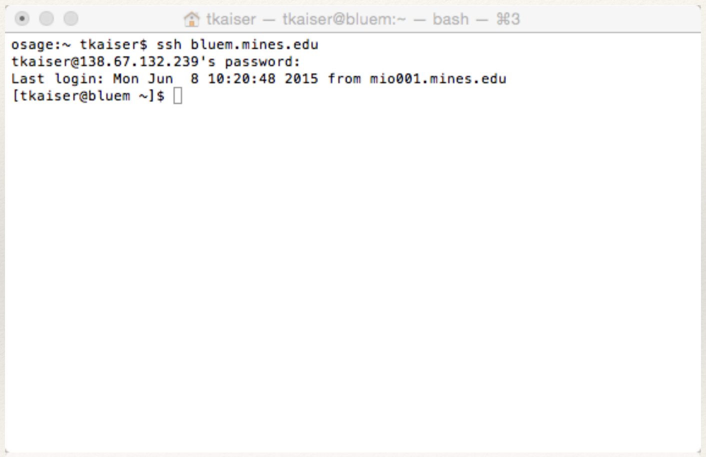
```
ssh -Y bluem.mines.edu
```

The normal command for getting logged into a linux box is ssh. In the simplest form you just type ssh followed by the machine name. If you have a different username on the remote machine you can specify that along with the machine name separated with the "at" symbol.

Another option is to add a command to run when you log in, such as L S. If you do have an extra command on the ssh command line you will be logged in, the command will run, and the session will then close.

The minus Y is important if you want to have a session in which the remote machine will launch a window.

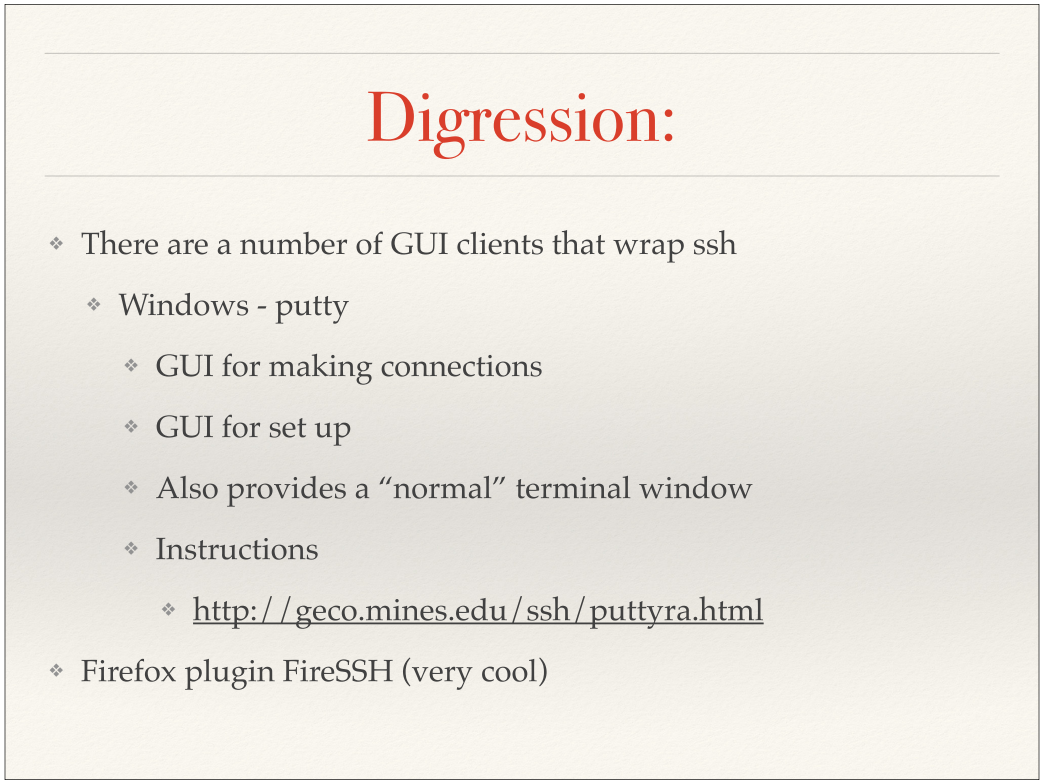
ssh from a terminal window



```
osage:~ tkaiser$ ssh bluem.mines.edu
tkaiser@138.67.132.239's password:
Last login: Mon Jun  8 10:20:48 2015 from mio001.mines.edu
[tkaiser@bluem ~]$ 
```

This is what it looks like to logon to a machine from a terminal window. We type ssh followed by the name of the machine to which we are going. Here we are asked for our password which is not echoed when we type it.

The screen dump shown here was done on a Macintosh running the Terminal application. Terminal can be found in your the — utility — folder inside of the — applications — folder.

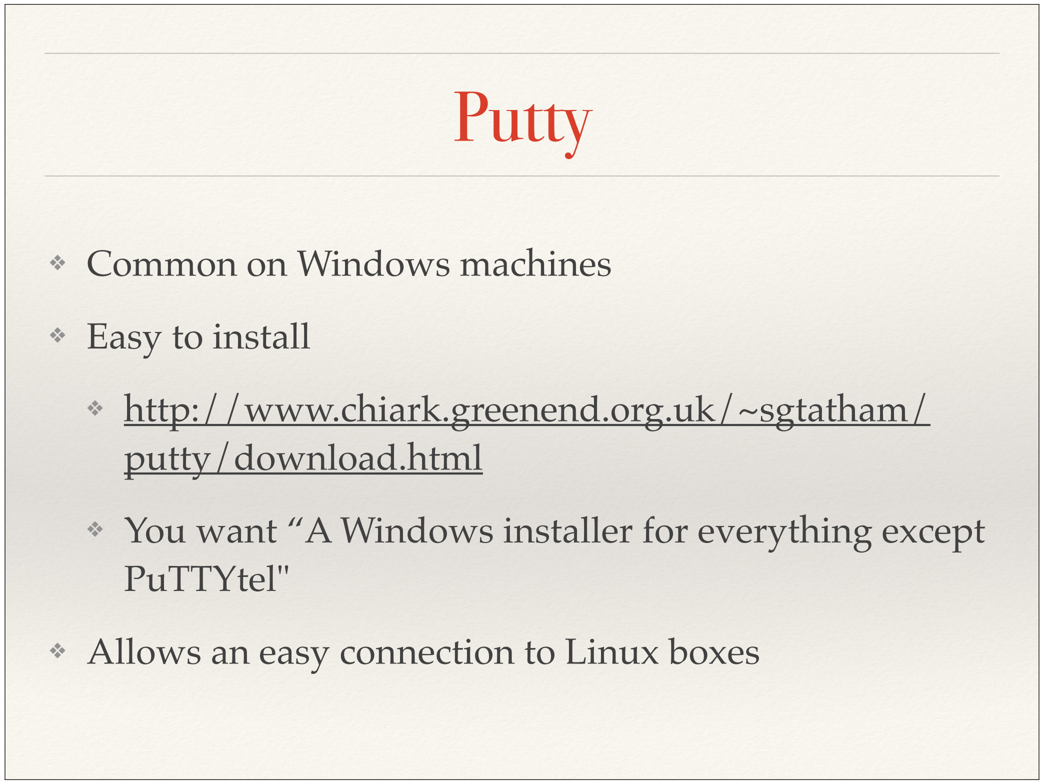


Digression:

- There are a number of GUI clients that wrap ssh
  - Windows - putty
    - GUI for making connections
    - GUI for set up
    - Also provides a “normal” terminal window
    - Instructions
      - http://geco.mines.edu/ssh/puttyra.html
- Firefox plugin FireSSH (very cool)

There are a number of GUI based programs that support ssh . The there is the putty program that is commonly used in the windows world. We actually have a separate web page that shows how to set up putty.

Also, there is a plugin available for the Firefox web browser. It supports doing ssh sessions from inside of web browser. You just enter the name of the machine you are going to instead of a web page.

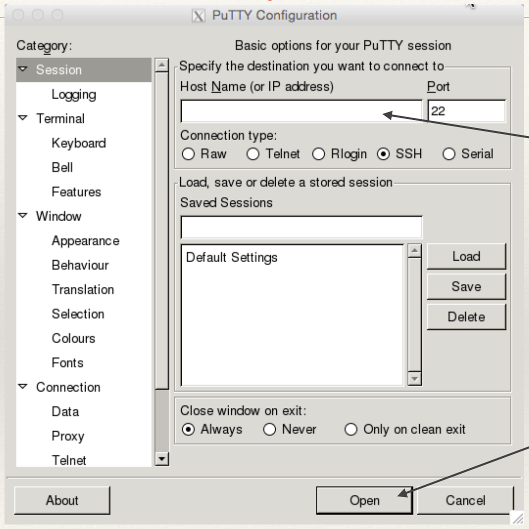


Putty

- Common on Windows machines
- Easy to install
  - http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
  - You want “A Windows installer for everything except PuTTYtel"
- Allows an easy connection to Linux boxes

The Putty application is a light weight application that can be used from windows boxes to connect to linux machines using ssh. It is easy to install and easy to use for basic operations. Also, it has a large number of options and also comes will an application for doing copies of files between machines.

Putty setup Window

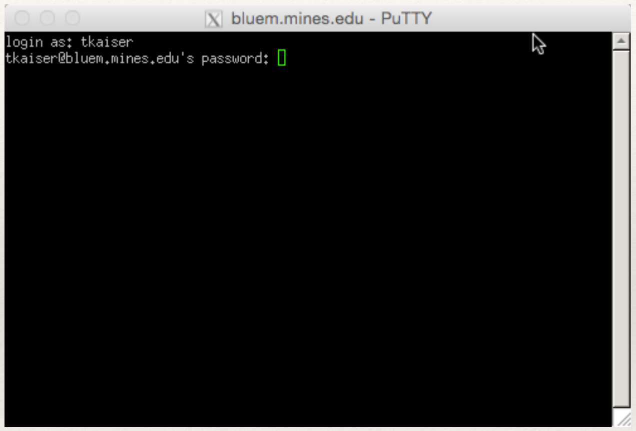


Enter the name of the machine to which you want to connect here.

Then click open

Here is a screen shot of the basic setup window. All you need to do is enter the name of the machine to which you want to connect. Then hit the open button.

Putty Terminal Window



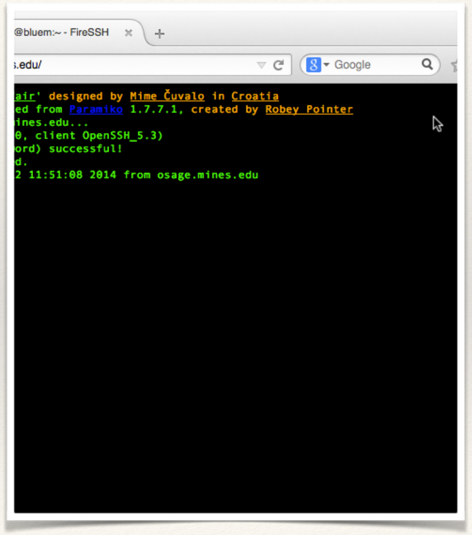
```
login as: tkaiser
tkaiser@bluem.mines.edu's password:
```

- Lots more options, see:
  - http://geco.mines.edu/ssh/puttyra.html

You will be connected to the machine and you will be asked for your user name and password. After that you should be in. Have a look at the local putty page for instructions on advanced options that will, in the end, make your life easier.

Firefox web based interactions

- There is a extension for the Firefox browser called **FireSSH** that gives you a terminal window within Firefox
- Very cool and works well!
- To launch it you just enter the name of the machine if the form ssh://bluem.mines.edu
- By the way…
  - If you try this in Safari it will open up the Terminal Application



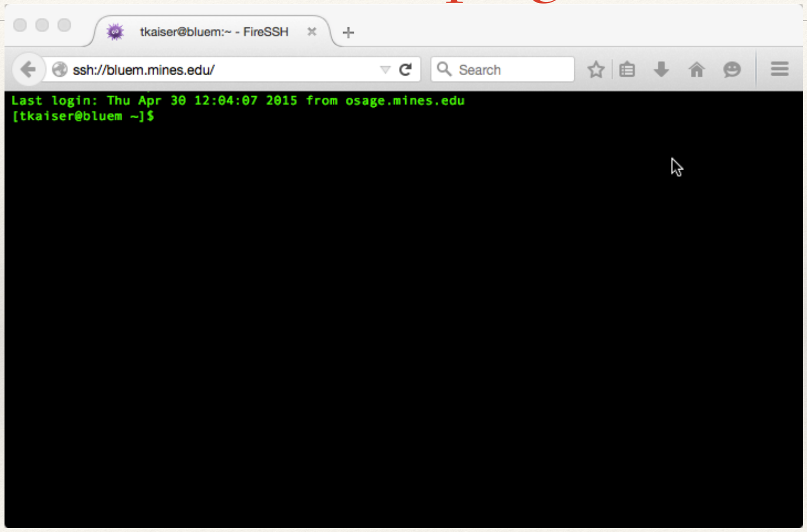
```
air' designed by Mime Čuvalo in Croatia
ed from Paramiko 1.7.7.1, created by Robey Pointer
ines.edu...
0, client OpenSSH_5.3)
ord) successful!
d.
2 11:51:08 2014 from osage.mines.edu
```

This is very cool. Another option for interacting with a Linux machine is to do it from the web browser Firefox. There is an extension called FireSSH that allows for interactive terminal session with remote machines. After the extension is installed you just enter the name of the machine in the address bar. You precede the address withssh colon double slash as show in the screen dump.

The extension should work on any platform that supports Firefox, including Windows. This might be the easiest way to get going in Windows.

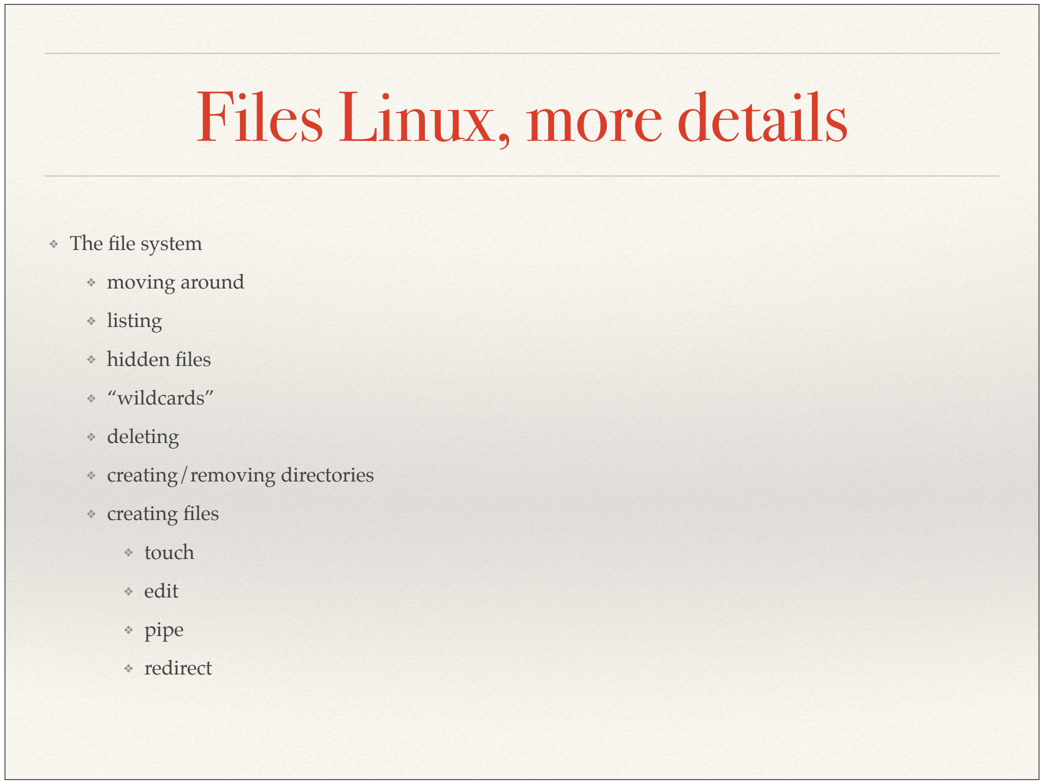
An interesting side note: if you try this in Safari it will open up the Terminal Application. You an even set up bookmarks to open machines both in FireFox and Safari.

FireSSH plugin



```
Last login: Thu Apr 30 12:04:07 2015 from osage.mines.edu
[tkaiser@bluem ~]$
```

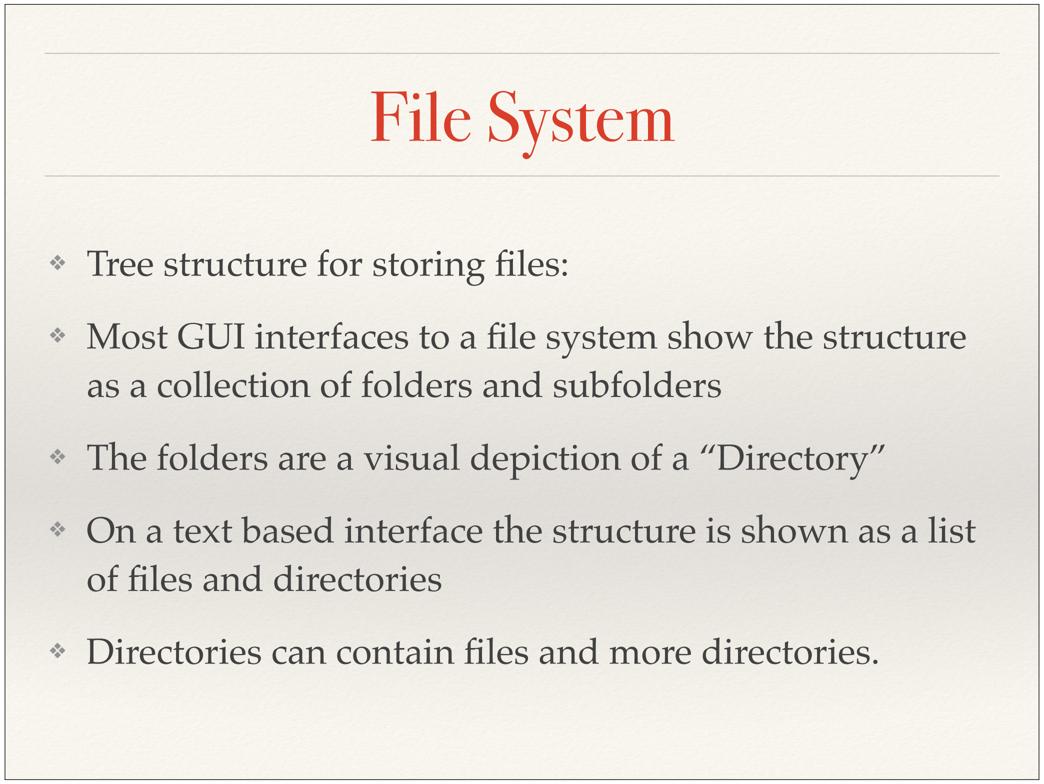
Here is a screen dump of using the Fire ssh plugin with the Firefox browser. To connect to Blue M we enter ssh followed by the full name of the machine. We are then asked for our user name and password.



Files Linux, more details

- The file system
  - moving around
  - listing
  - hidden files
  - “wildcards”
  - deleting
  - creating/removing directories
  - creating files
    - touch
    - edit
    - pipe
    - redirect

We want to talk about the file system in Linux.   We’ll talk about getting around and seeing what you have. Hidden files are very important in Linux. Wildcards can be used to apply an operation to groups of files. Well look at directories and finally we’ll see different ways of creating files including using editors.

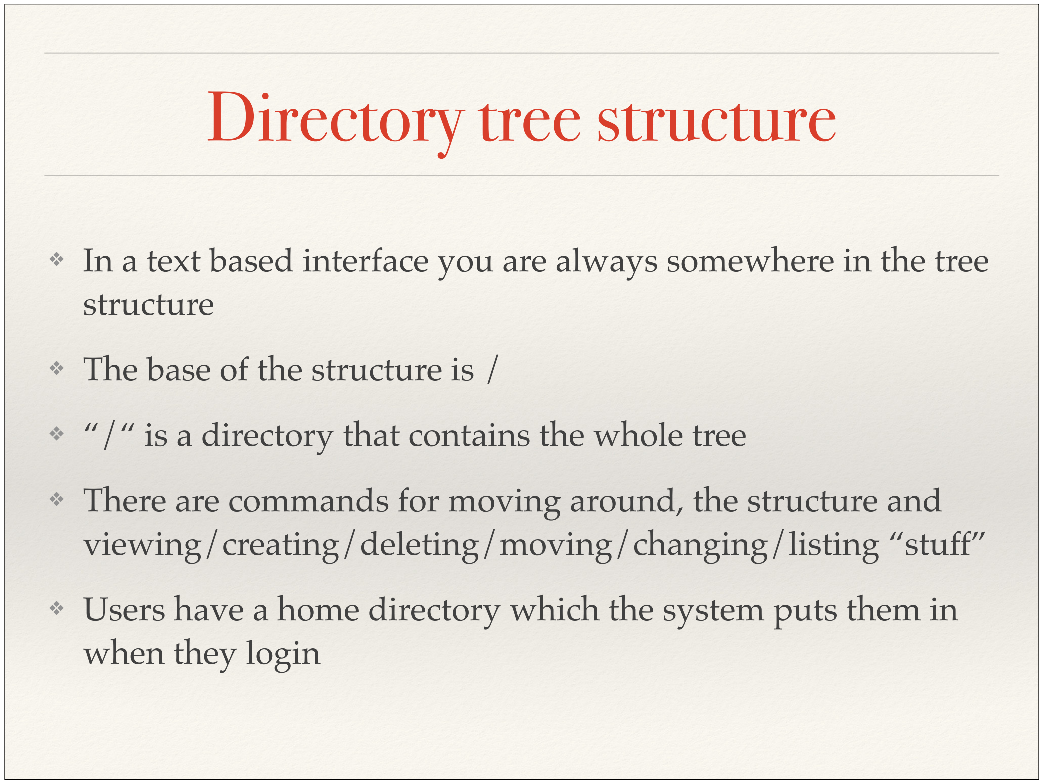


File System

- Tree structure for storing files:
- Most GUI interfaces to a file system show the structure as a collection of folders and subfolders
- The folders are a visual depiction of a "Directory"
- On a text based interface the structure is shown as a list of files and directories
- Directories can contain files and more directories.

The file system is a tree structure that is used to store files. On GUI based systems the file system is shown as a collection of folders and subfolders and files. The folders are directories. Directories can contain regular files or other directories.

On a terminal window, that is, on a text based interface the structure is shown as a list of files and directories.



Directory tree structure

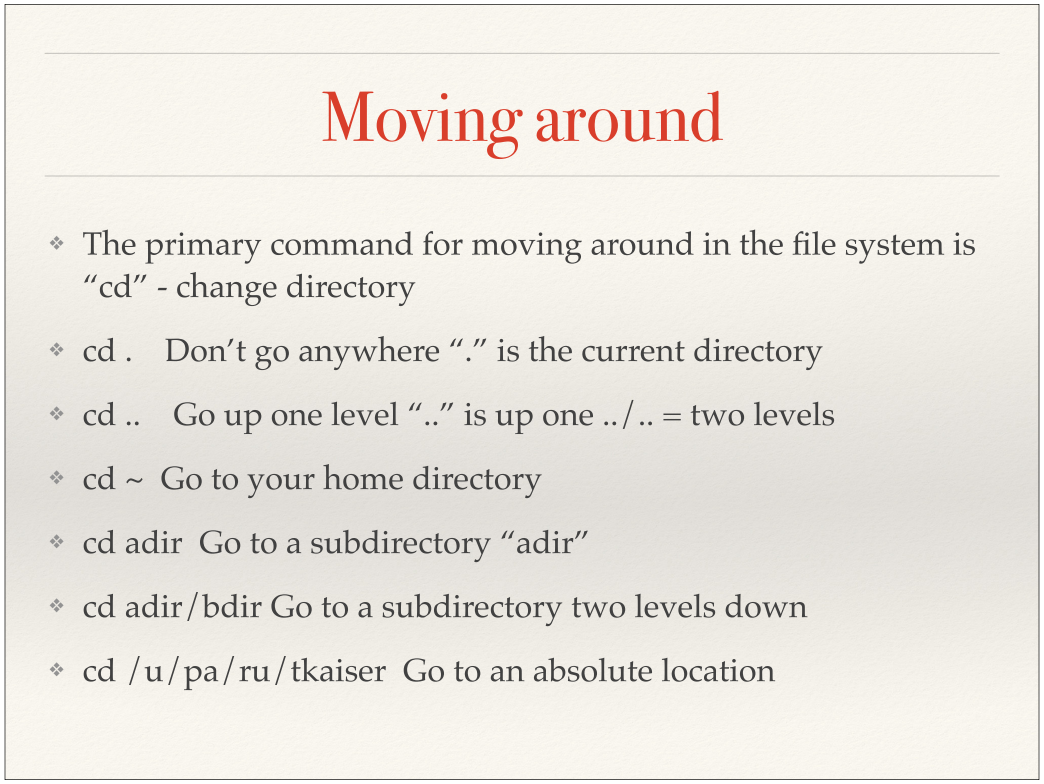
- In a text based interface you are always somewhere in the tree structure
- The base of the structure is /
- "/" is a directory that contains the whole tree
- There are commands for moving around, the structure and viewing/creating/deleting/moving/changing/listing "stuff"
- Users have a home directory which the system puts them in when they login

In a text based interface you are always somewhere in the tree structure. You may hear of your location being described as the current working directory. If you have multiple terminal windows open each one can have a different current working directory.

The top of the tree structure is indicated as a slash. Every sub directory is under this top level directory. When you see a full path to a file it will start with slash and every subdirectory adds another slash to the path.

We'll look as commands for viewing/creating/deleting/moving/changing/listing files and directories.

When you first login to a system you will be put in your home directory. On CSM's HPC platforms you also have a scratch and bins directory where you will do most of your work.

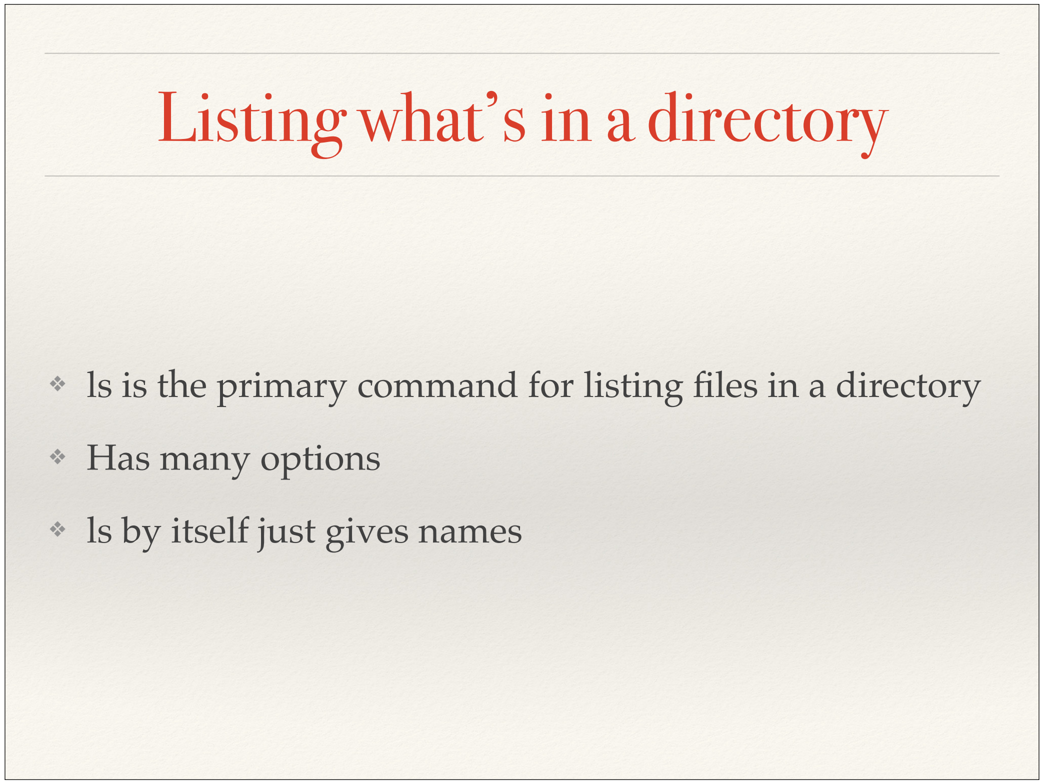


Moving around

- The primary command for moving around in the file system is “cd” - change directory
- cd .   Don’t go anywhere “.” is the current directory
- cd ..   Go up one level “..” is up one ../.. = two levels
- cd ~ Go to your home directory
- cd adir Go to a subdirectory “adir”
- cd adir/bdir Go to a subdirectory two levels down
- cd /u/pa/ru/tkaiser Go to an absolute location

The primary command for moving around in the file system is “CD” or change directory. The period represents the current working directory so CD to period does nothing. CD period period goes up one level CD tilda goes to your home directory. CD followed by a name goes to the named subdirectory, assuming that it exists under the current working directory.

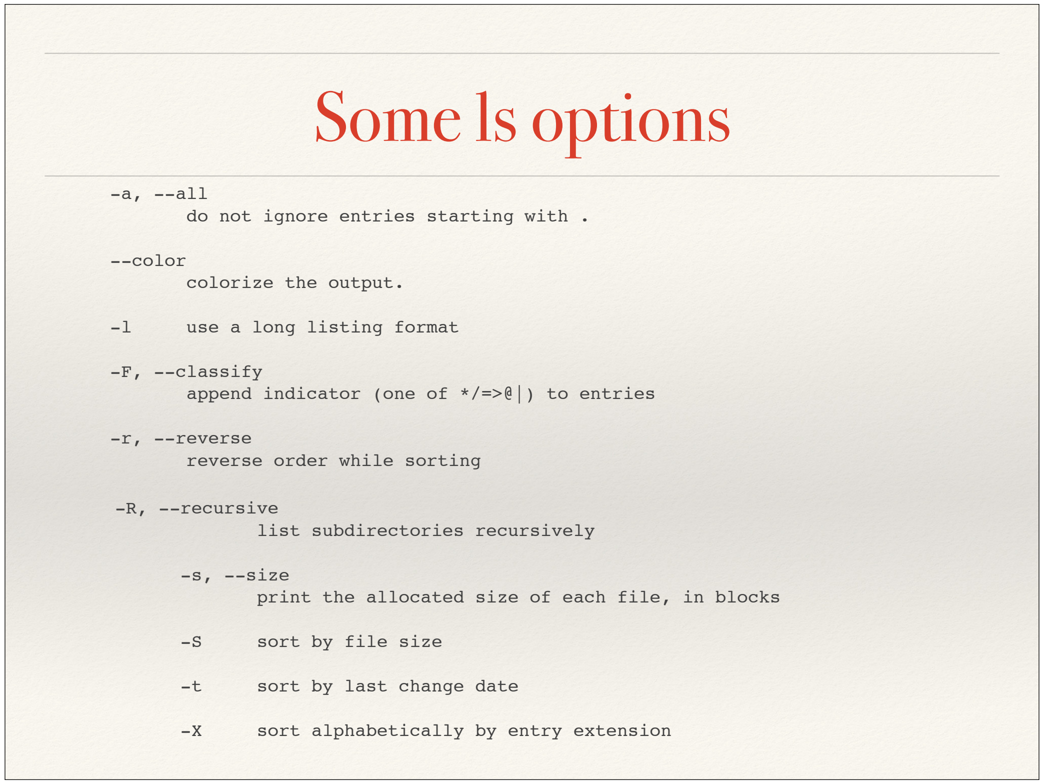
You can go down multiple levels by using the slash character. If you start the path with a slash this is called an absolute path and it will take you to the top level and then down to the named directory.



Listing what's in a directory

- ls is the primary command for listing files in a directory
- Has many options
- ls by itself just gives names

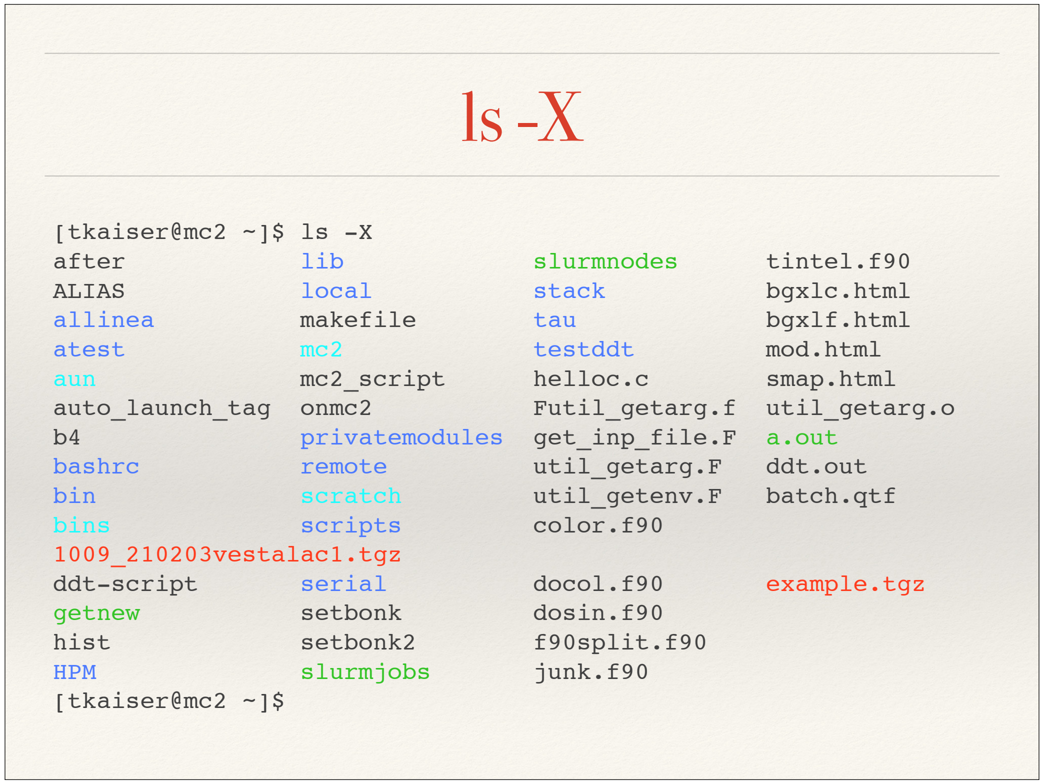
Typing the command "LS" will show you a list of files and directories. Just typing the command you will get a simple list. But we have many options.



Some ls options

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

--color
       colorize the output.

-l     use a long listing format

-F, --classify
       append indicator (one of */=>@|) to entries

-r, --reverse
       reverse order while sorting

 -R, --recursive
              list subdirectories recursively

       -s, --size
              print the allocated size of each file, in blocks

       -S     sort by file size

       -t     sort by last change date

       -X     sort alphabetically by entry extension
```

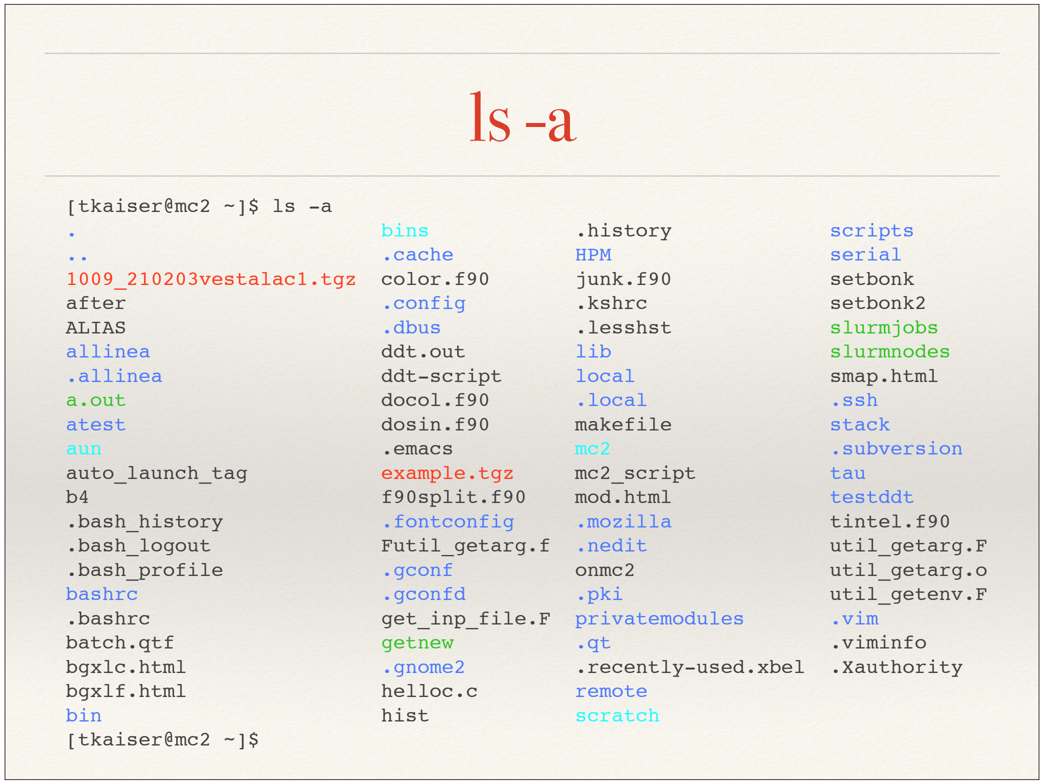
Here are some of the options for LS. As mentioned earlier Linux has hidden files. A file is hidden if it starts with a period. The dash A option will show hidden files. The color and minus F option are related. color assigns a different color to the different types of files for example runnable programs, text files, or directories. The minus F option will do the same thing except by adding a character to the end of the file name. Minus L gives a long listing which contains sizes, dates and protections. We will talk about protections shortly. You can sort the list by size or date and use minus r to reverse the sort. The minus X option is not available on all machines but can be useful.



ls -X

```
[tkaiser@mc2 ~]$ ls -X
after             lib               slurmnodes        tintel.f90
ALIAS             local             stack             bgxlc.html
allinea           makefile          tau               bgxlf.html
atest             mc2               testddt           mod.html
aun               mc2_script        helloc.c          smap.html
auto_launch_tag   onmc2             Futil_getarg.f    util_getarg.o
b4                privatemodules    get_inp_file.F    a.out
bashrc            remote            util_getarg.F     ddt.out
bin               scratch           util_getenv.F     batch.qtf
bins              scripts           color.f90
1009_210203vestalac1.tgz
ddt-script        serial            docol.f90         example.tgz
getnew            setbonk           dosin.f90
hist              setbonk2          f90split.f90
HPM               slurmjobs         junk.f90
[tkaiser@mc2 ~]$
```

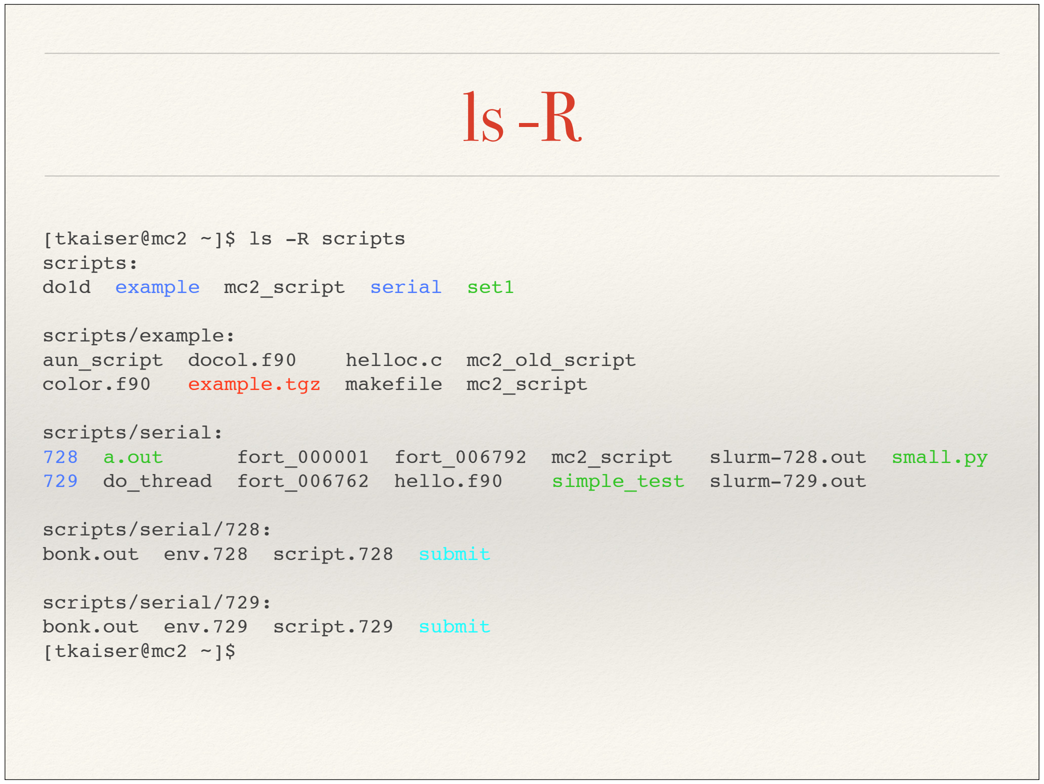
Here we have used the minus X option to sort by file name extension; nothing, C, F, F 90, html, O, out and so on.



ls -a

```
[tkaiser@mc2 ~]$ ls -a
.                         bins            .history             scripts
..                        .cache          HPM                  serial
1009_210203vestalac1.tgz  color.f90       junk.f90             setbonk
after                     .config         .kshrc               setbonk2
ALIAS                     .dbus           .lesshst             slurmjobs
allinea                   ddt.out         lib                  slurmnodes
.allinea                  ddt-script      local                smap.html
a.out                     docol.f90       .local               .ssh
atest                     dosin.f90       makefile             stack
aun                       .emacs          mc2                  .subversion
auto_launch_tag           example.tgz     mc2_script           tau
b4                        f90split.f90    mod.html             testddt
.bash_history             .fontconfig     .mozilla             tintel.f90
.bash_logout              Futil_getarg.f  .nedit               util_getarg.F
.bash_profile             .gconf          onmc2                util_getarg.o
bashrc                    .gconfd         .pki                 util_getenv.F
.bashrc                   get_inp_file.F  privatemodules       .vim
batch.qtf                 getnew          .qt                  .viminfo
bgxlc.html                .gnome2         .recently-used.xbel  .Xauthority
bgxlf.html                helloc.c        remote
bin                       hist            scratch
[tkaiser@mc2 ~]$
```

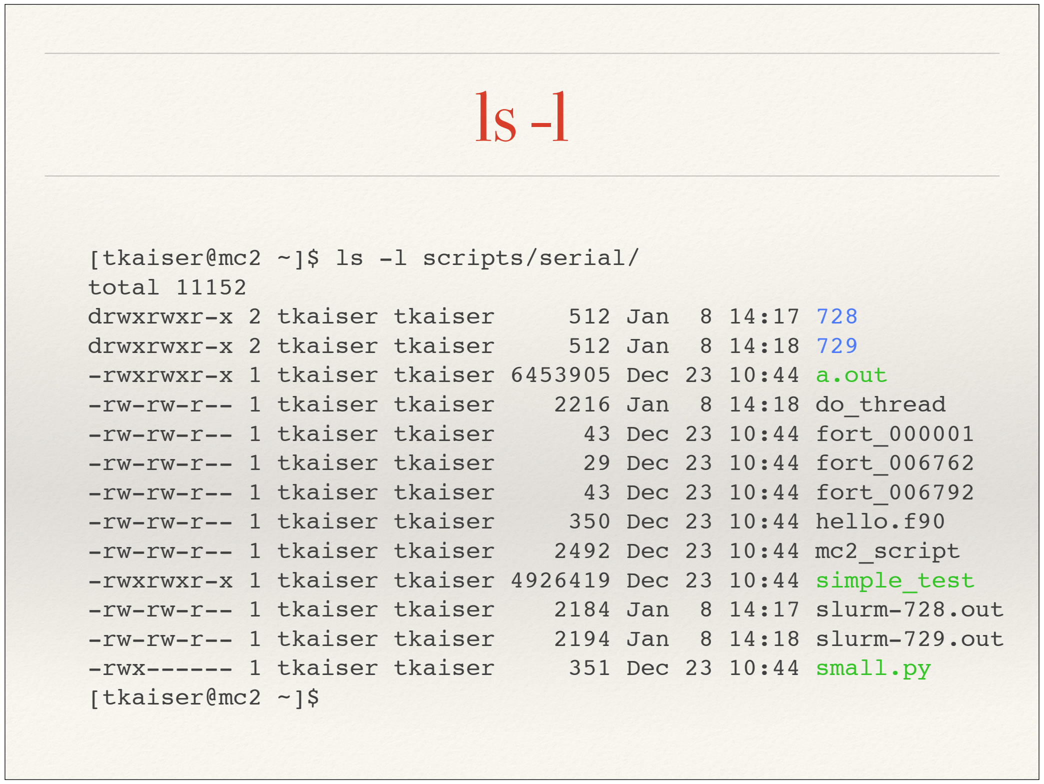
We have a listing showing hidden files. That is files with names that start with a period.



ls -R

```
[tkaiser@mc2 ~]$ ls -R scripts
scripts:
do1d  example  mc2_script  serial  set1

scripts/example:
aun_script  docol.f90    helloc.c  mc2_old_script
color.f90   example.tgz  makefile  mc2_script

scripts/serial:
728  a.out       fort_000001  fort_006792  mc2_script   slurm-728.out  small.py
729  do_thread   fort_006762  hello.f90    simple_test  slurm-729.out

scripts/serial/728:
bonk.out  env.728  script.728  submit

scripts/serial/729:
bonk.out  env.729  script.729  submit
[tkaiser@mc2 ~]$
```

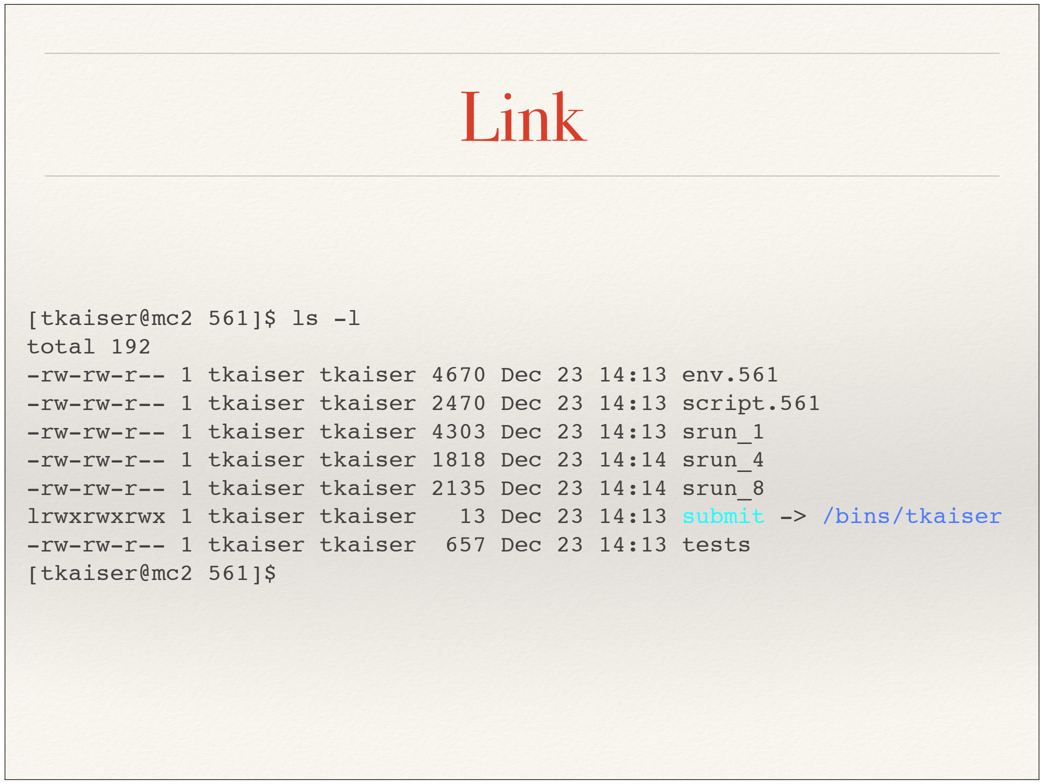
We have a recursive listing showing files, subdirectories and files and within the subdirectories



ls -l

```
[tkaiser@mc2 ~]$ ls -l scripts/serial/
total 11152
drwxrwxr-x 2 tkaiser tkaiser     512 Jan  8 14:17 728
drwxrwxr-x 2 tkaiser tkaiser     512 Jan  8 14:18 729
-rwxrwxr-x 1 tkaiser tkaiser 6453905 Dec 23 10:44 a.out
-rw-rw-r-- 1 tkaiser tkaiser    2216 Jan  8 14:18 do_thread
-rw-rw-r-- 1 tkaiser tkaiser      43 Dec 23 10:44 fort_000001
-rw-rw-r-- 1 tkaiser tkaiser      29 Dec 23 10:44 fort_006762
-rw-rw-r-- 1 tkaiser tkaiser      43 Dec 23 10:44 fort_006792
-rw-rw-r-- 1 tkaiser tkaiser     350 Dec 23 10:44 hello.f90
-rw-rw-r-- 1 tkaiser tkaiser    2492 Dec 23 10:44 mc2_script
-rwxrwxr-x 1 tkaiser tkaiser 4926419 Dec 23 10:44 simple_test
-rw-rw-r-- 1 tkaiser tkaiser    2184 Jan  8 14:17 slurm-728.out
-rw-rw-r-- 1 tkaiser tkaiser    2194 Jan  8 14:18 slurm-729.out
-rwx------ 1 tkaiser tkaiser     351 Dec 23 10:44 small.py
[tkaiser@mc2 ~]$
```

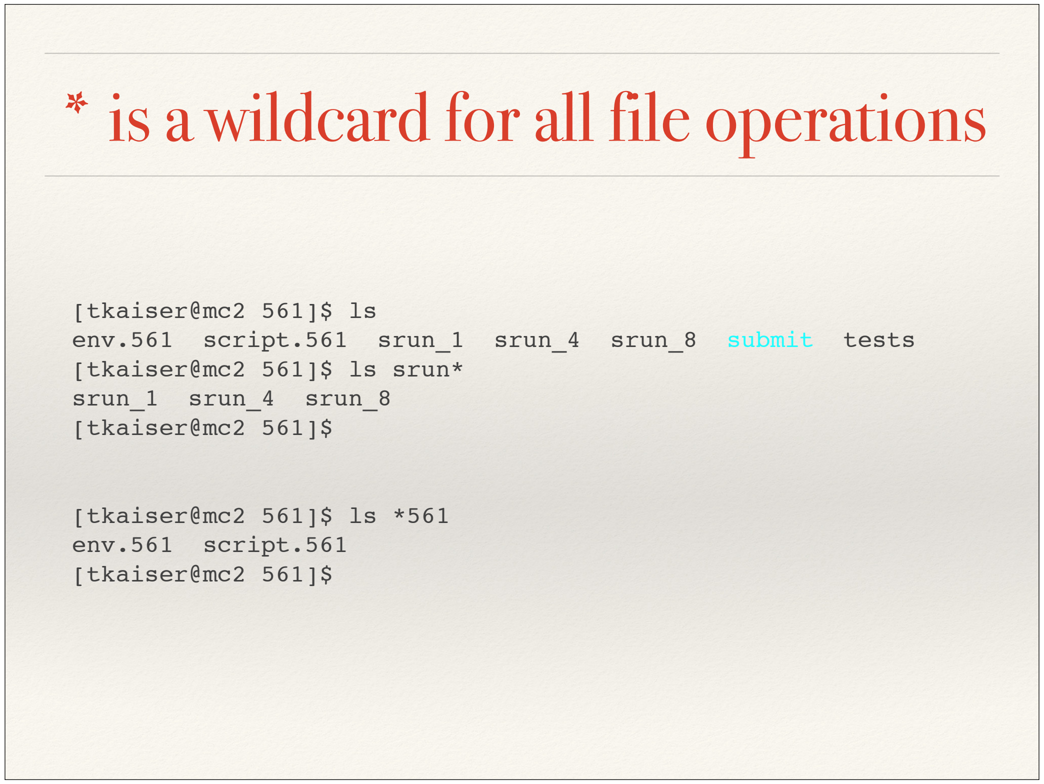
We have a long listing showing file protections or accessibility, owner, size, and date of last change along with the name.



Link

```
[tkaiser@mc2 561]$ ls -l
total 192
-rw-rw-r-- 1 tkaiser tkaiser 4670 Dec 23 14:13 env.561
-rw-rw-r-- 1 tkaiser tkaiser 2470 Dec 23 14:13 script.561
-rw-rw-r-- 1 tkaiser tkaiser 4303 Dec 23 14:13 srun_1
-rw-rw-r-- 1 tkaiser tkaiser 1818 Dec 23 14:14 srun_4
-rw-rw-r-- 1 tkaiser tkaiser 2135 Dec 23 14:14 srun_8
lrwxrwxrwx 1 tkaiser tkaiser   13 Dec 23 14:13 submit -> /bins/tkaiser
-rw-rw-r-- 1 tkaiser tkaiser  657 Dec 23 14:13 tests
[tkaiser@mc2 561]$
```

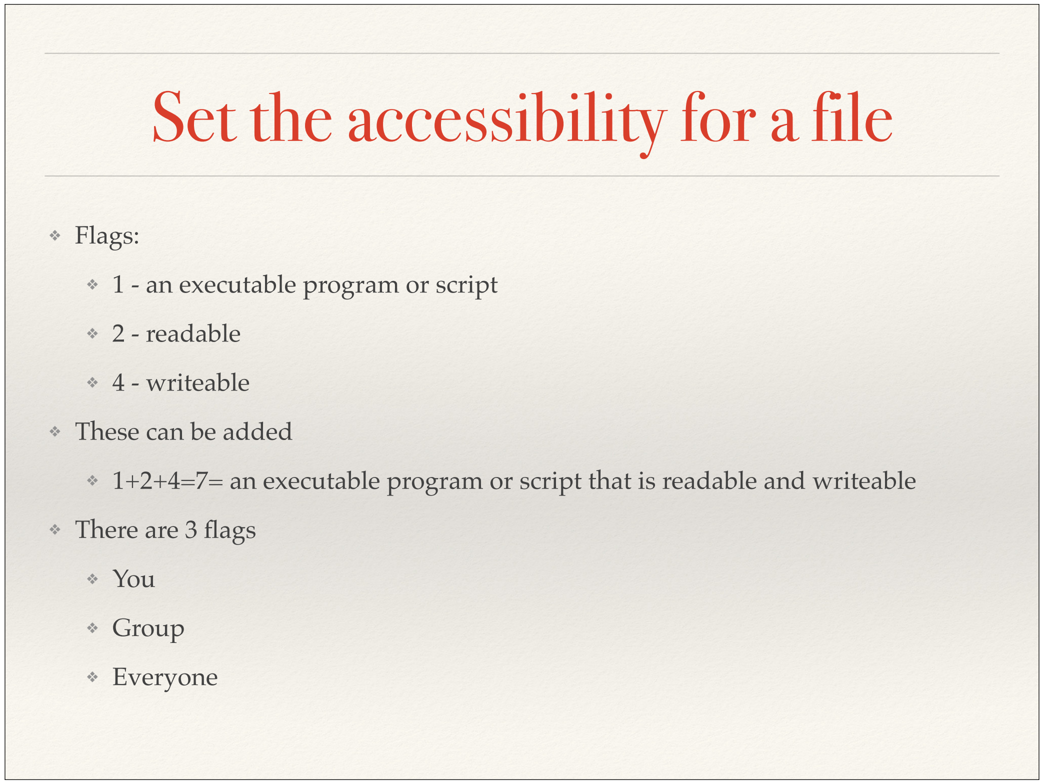
A file link is basically an alias for a file name. It can point to a file in a different directory. Here we use the minus l option to show that a file is a link and the path to the real file.



* is a wildcard for all file operations

```
[tkaiser@mc2 561]$ ls
env.561  script.561  srun_1  srun_4  srun_8  submit  tests
[tkaiser@mc2 561]$ ls srun*
srun_1  srun_4  srun_8
[tkaiser@mc2 561]$


[tkaiser@mc2 561]$ ls *561
env.561  script.561
[tkaiser@mc2 561]$
```

The asterisk is a wildcard that can be used for file operations, for example LS. Here we list files that start with srun and files files that end in 561.



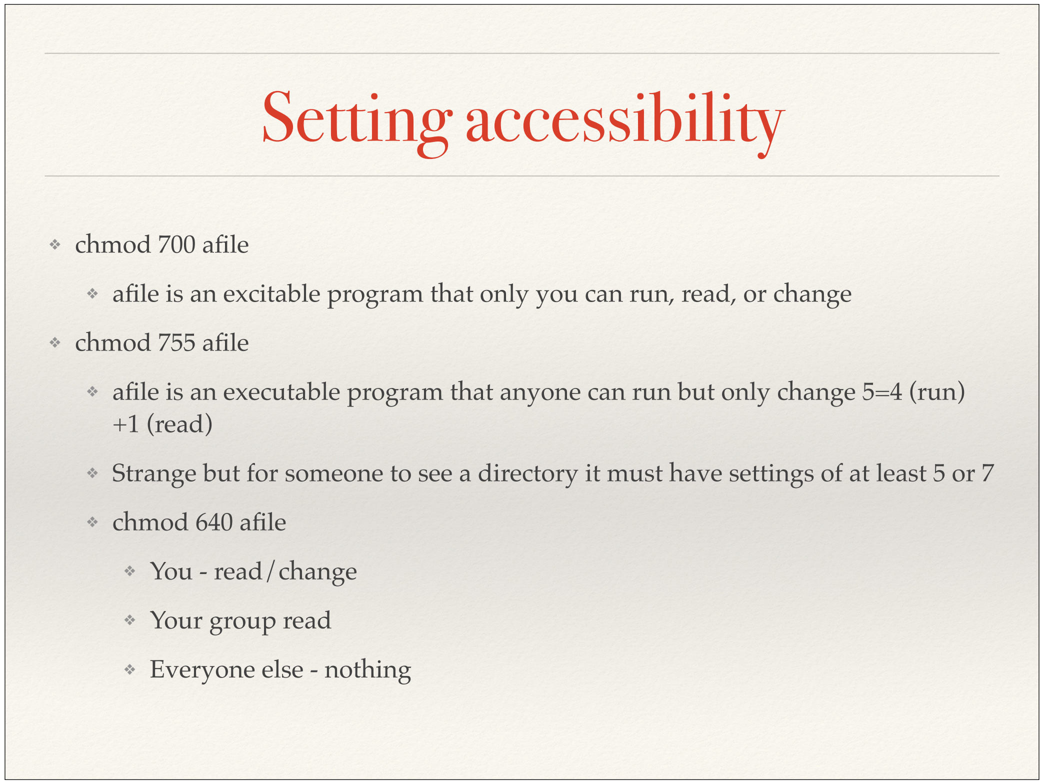
Set the accessibility for a file

- Flags:
  - 1 - an executable program or script
  - 2 - readable
  - 4 - writeable
- These can be added
  - 1+2+4=7= an executable program or script that is readable and writeable
- There are 3 flags
  - You
  - Group
  - Everyone

It is possible to set protections or accessibilities for a file. These settings determine what people can do with a file or directory. For the purpose of accessibility there are three classifications of users related to a file. The first is the file owner. The second is the owners group. A person can belong to several groups. On most systems there is a group with the same name as the owners name. A person will be part of their own group but might also be part of a shared research group that contains several people. While a person can be part of several groups there can only be one group associated with a file.

There are three types of access, executable, that is the file contains a runnable program and can be run by a person in the particular class. Readable implies that the file can be read, writeable implies that the file can be changed or even deleted. These access levels are indicated by setting a flag using the values 1 which implies executable, 2 which implies readable, and 4 which implies writeable. The values can be added so for example 1+2+4=7 implies an executable program or script that is readable and writeable

You can set different protections for different classes. You can set it so that you have full access to a file, the group can only read it and everyone outside of the group has no access.



Setting accessibility

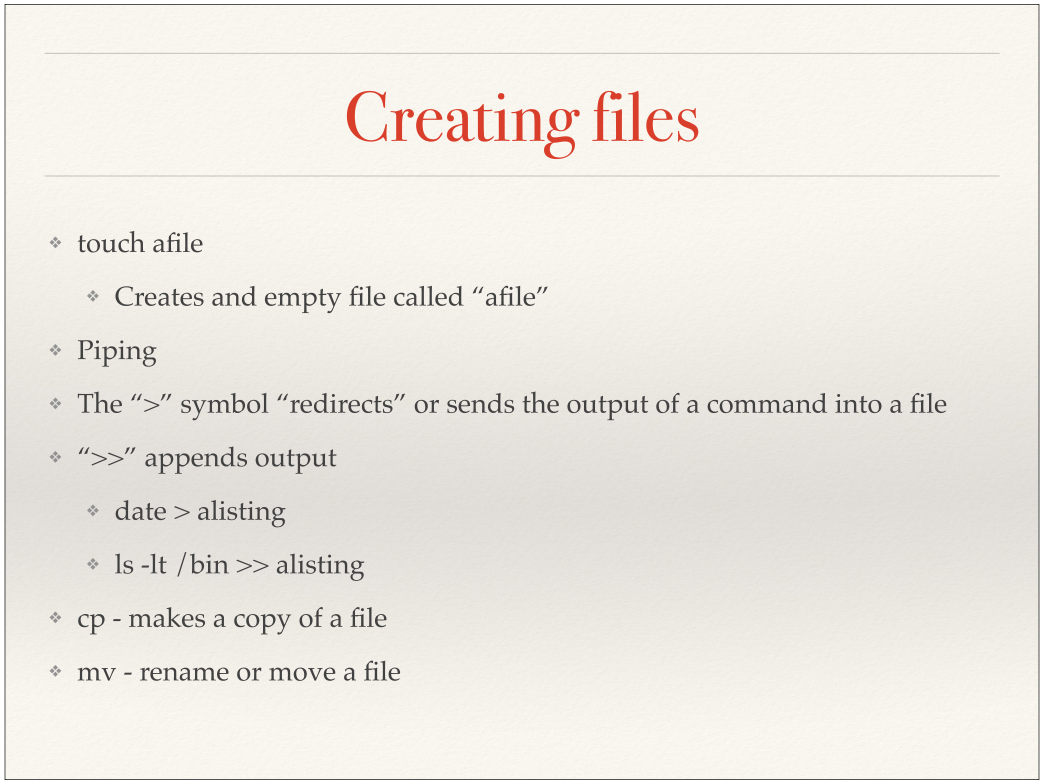
- chmod 700 afile
    - afile is an excitable program that only you can run, read, or change
- chmod 755 afile
    - afile is an executable program that anyone can run but only change 5=4 (run) +1 (read)
    - Strange but for someone to see a directory it must have settings of at least 5 or 7
    - chmod 640 afile
        - You - read / change
        - Your group read
        - Everyone else - nothing

The command to set accessibility is chmod. The easiest syntax for chmod is to give the accessibility level as a three digit number followed by the file name. For example chmod 700 afile sets afile as an executable program that only you can run, read, or change.

chmod 755 afile gives an executable program that anyone can run but only change. The 5 comes from adding 4, the run flag and 1 the read flag.

For someone to see a directory it must have settings of at least 5 or 7

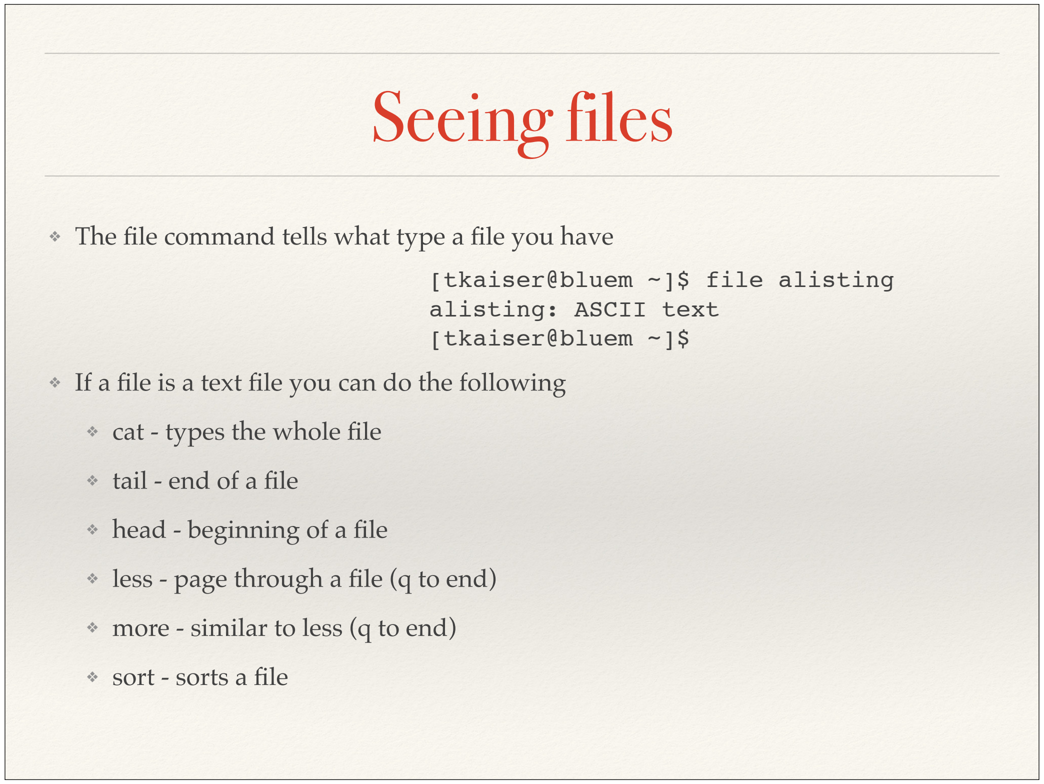
Here is one more example. chmod 640 afile. You can read it and change it. The setting 4 allows people in your group can read it. The 0 states that people outside of your group have no access.



Creating files

- touch afile
  - Creates and empty file called “afile”
- Piping
- The “>” symbol “redirects” or sends the output of a command into a file
- “>>” appends output
  - date > alisting
  - ls -lt /bin >> alisting
- cp - makes a copy of a file
- mv - rename or move a file

There are many ways to create files. Obviously you could use an editor but here are some other ways. The touch command will create an empty file of a given name if it does note exist. If it does exist it updates the latest modification time as shown in LS. If you run a command you can pipe or redirect the output to a file using the greater than symbol. If you use two of them together the output from the command is appended to the file. The CP command copies a file and the MV command renames a file or it can also be used to move the file to a new directory.



Seeing files

- The file command tells what type a file you have

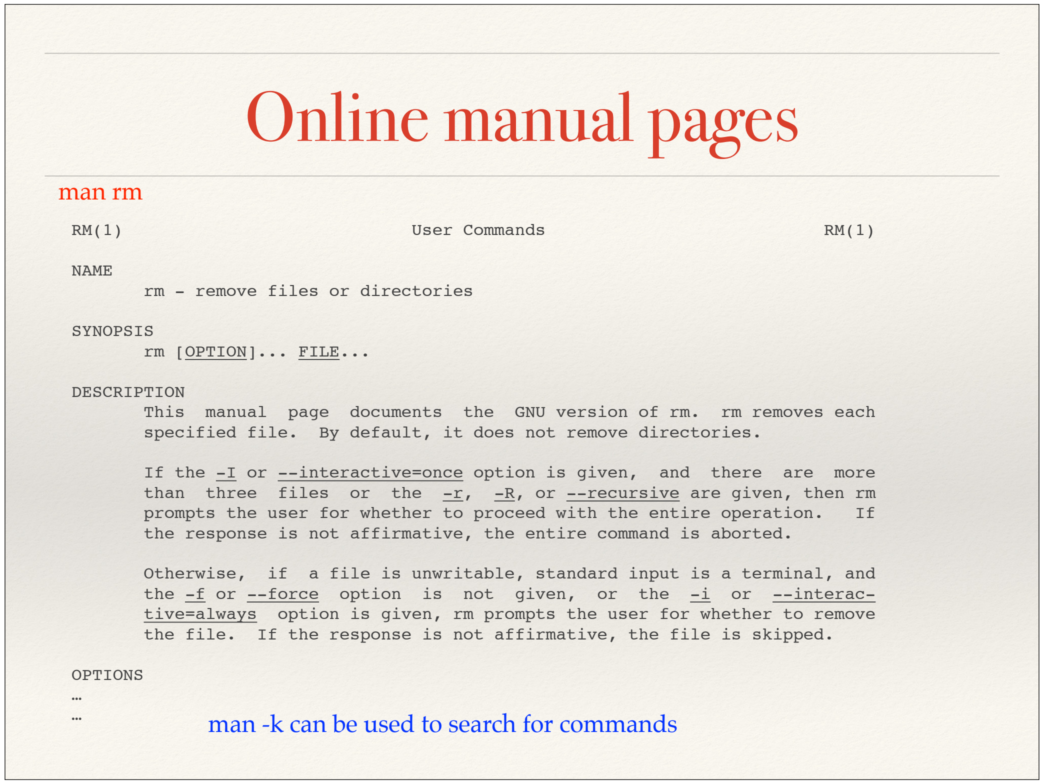
```
[tkaiser@bluem ~]$ file alisting
alisting: ASCII text
[tkaiser@bluem ~]$
```

- If a file is a text file you can do the following
  - cat - types the whole file
  - tail - end of a file
  - head - beginning of a file
  - less - page through a file (q to end)
  - more - similar to less (q to end)
  - sort - sorts a file

There is an interesting command in linux called —— “File”. The purpose of the command is to show the type of contents for a file. It will report if a file is a text file or in binary. File will try to given details. For example it will try to guess the programming language for a file that it determines is the source for a program. For image files it may give you the size.

The commands shown here work for text files. If you try to “cat” or type a image or some other binary file you will get garbage and it could mess up the settings for your terminal window.

The rest of the commands are other ways to view text files. All of them have lots of options. To see the options enter man — followed by the command. The — man — command stands for manual.



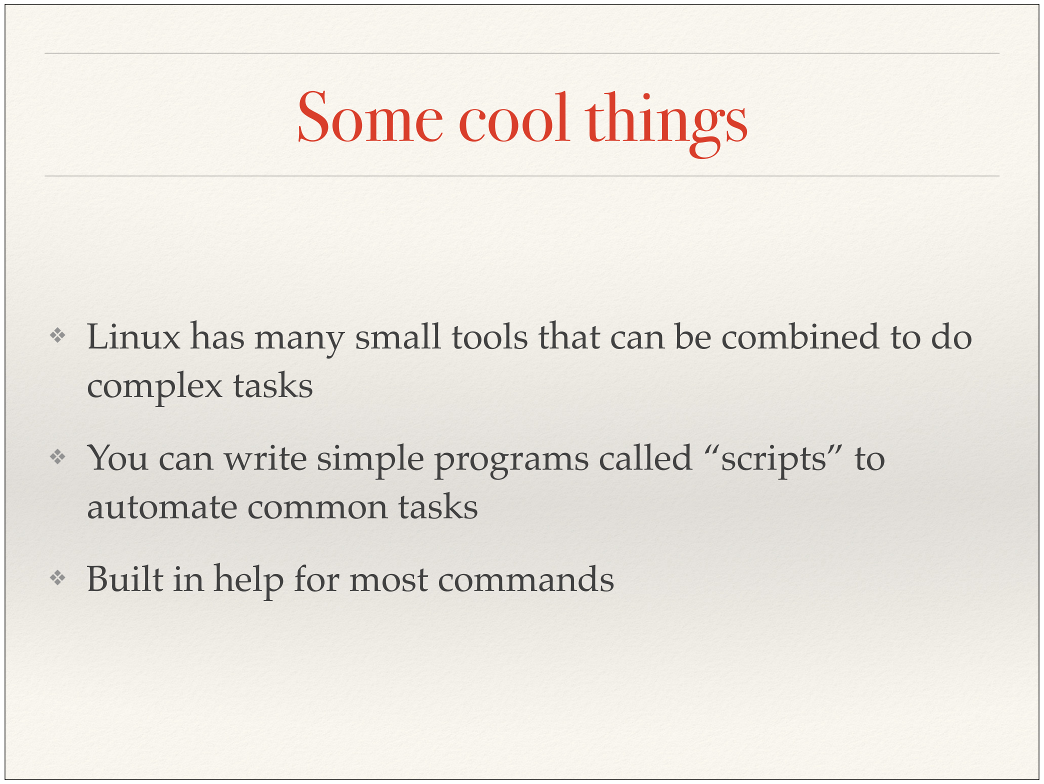
Online manual pages

man rm

```
RM(1)                            User Commands                           RM(1)

NAME
       rm - remove files or directories

SYNOPSIS
       rm [OPTION]... FILE...

DESCRIPTION
       This  manual  page  documents  the  GNU version of rm.  rm removes each
       specified file.  By default, it does not remove directories.

       If the -I or --interactive=once option is given,  and  there  are  more
       than  three  files  or  the  -r,  -R, or --recursive are given, then rm
       prompts the user for whether to proceed with the entire operation.   If
       the response is not affirmative, the entire command is aborted.

       Otherwise,  if  a file is unwritable, standard input is a terminal, and
       the -f or --force  option  is  not  given,  or  the  -i  or  --interac-
       tive=always  option is given, rm prompts the user for whether to remove
       the file.  If the response is not affirmative, the file is skipped.

OPTIONS
…
…
```

man -k can be used to search for commands

Linux has a built in manual. You can type — man — followed by the command name and you will be shown a description and list of options a screen at a time. Here we have the man page for the RM command which we will talk about shortly. If you enter man minus k followed by a key word you will be shown commands related to the key word.

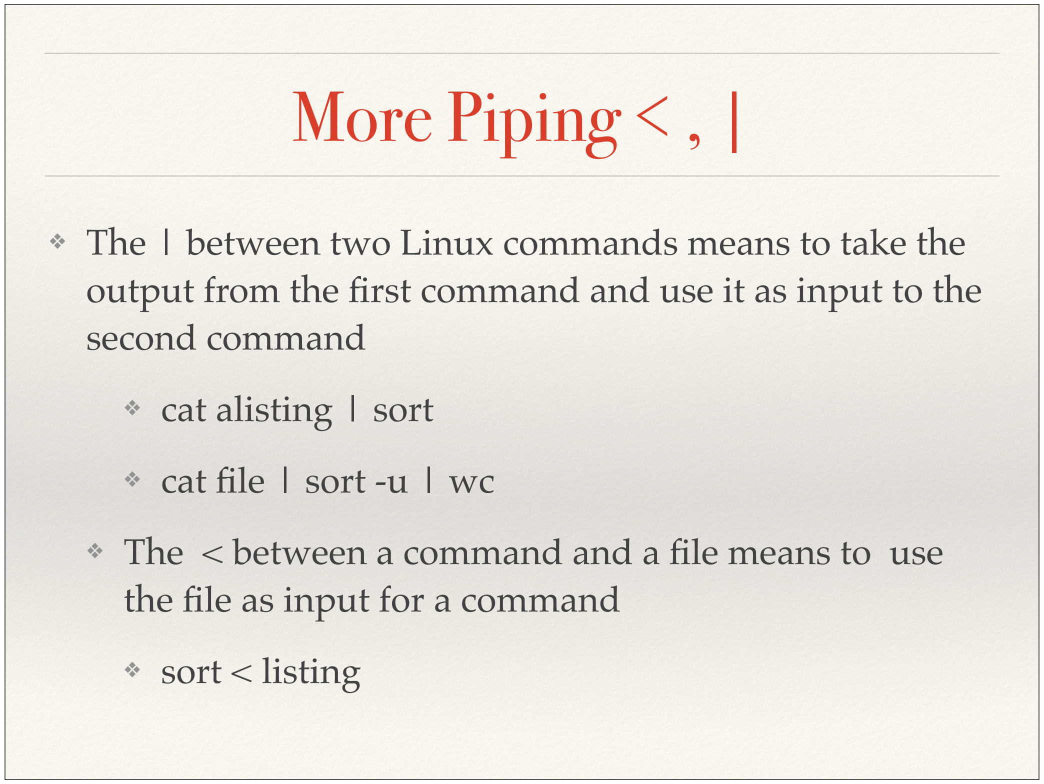


Some cool things

- Linux has many small tools that can be combined to do complex tasks
- You can write simple programs called "scripts" to automate common tasks
- Built in help for most commands

Linux has a lot of cool features. We have talked about the man pages or build in help system. We are not going to talk much about writing scripts but a script can be as simple as a collection of commands that you use on a regular basis that are saved into a file.

One of the nice things about Linux is that you can combine many small tools or commands to do complex tasks even without writing a script. This is done with pipes.

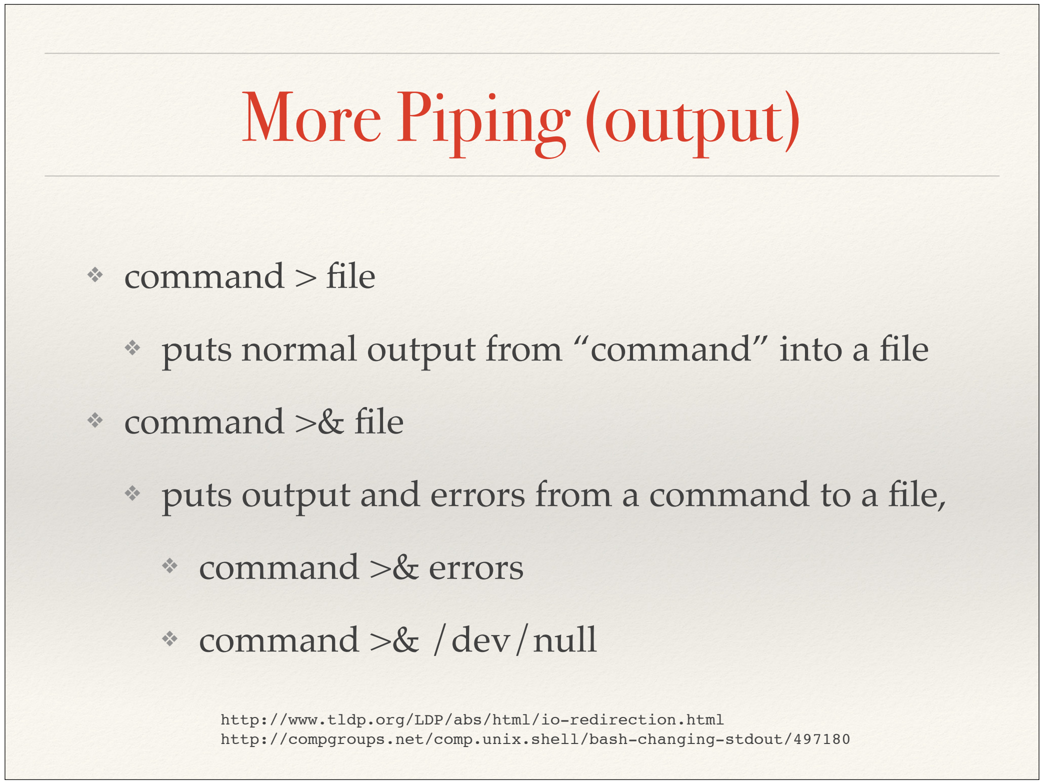


More Piping < , |

- The | between two Linux commands means to take the output from the first command and use it as input to the second command
    - cat alisting | sort
    - cat file | sort -u | wc
    - The < between a command and a file means to use the file as input for a command
    - sort < listing

Pipes can be used to send the output of a command somewhere. The vertical bar is one type of pipe. It causes the output of one command to be used as the input to a second command. You can have multiple vertical bar pipes on a line. For example if you want to count the number of unique lines in a file you can — cat — a file into the sort command with the minus U flag to give you a uniquely sorted list and then pipe that into the W C or word count command.

The less than symbol says that you want send the input from a file into a command. The sort command is like many Linux commands that can use either of the two styles of input piping.



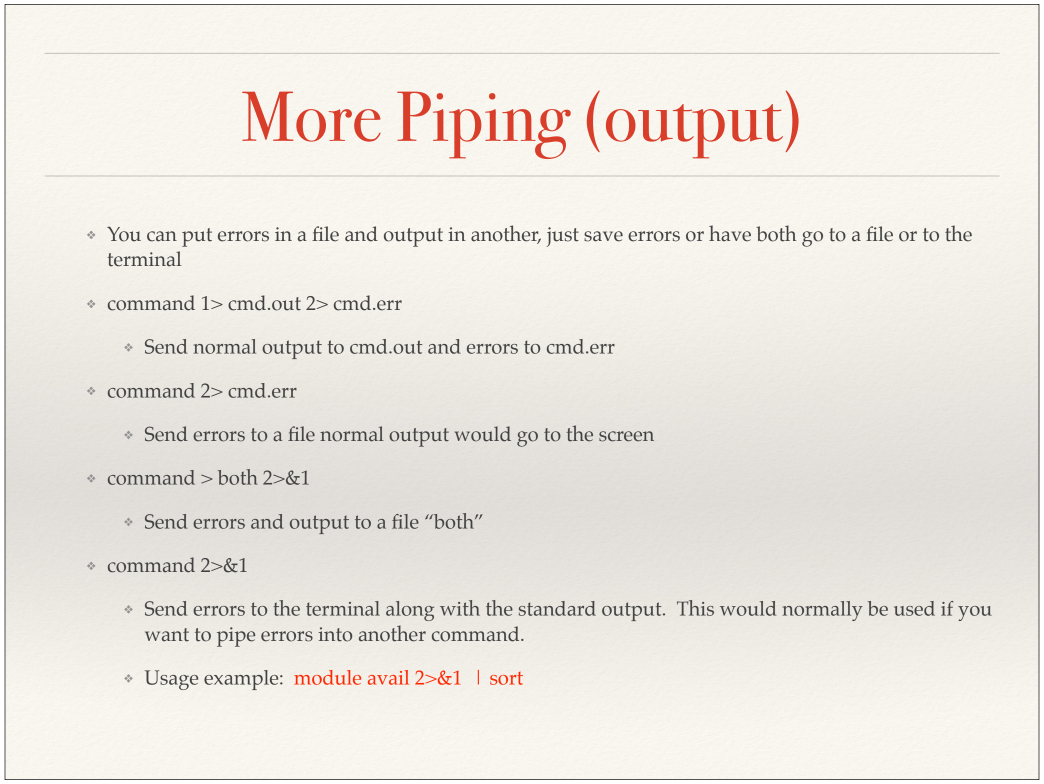
More Piping (output)

- command > file
  - puts normal output from “command” into a file
- command >& file
  - puts output and errors from a command to a file,
    - command >& errors
    - command >& /dev/null

http://www.tldp.org/LDP/abs/html/io-redirection.html
http://compgroups.net/comp.unix.shell/bash-changing-stdout/497180

The greater than symbol can be used to send output to files. Used by itself you can put standard output, that is normal program output into a file as shown here. You can also use the greater than symbol and ampersand symbols together to send both standard output and errors to a file. The file dev null is a special file. Anything sent to dev null just disappears. This is useful if you are not interested in seeing errors from a program.

Note we have some web pages listed here that talk about piping and redirecting.

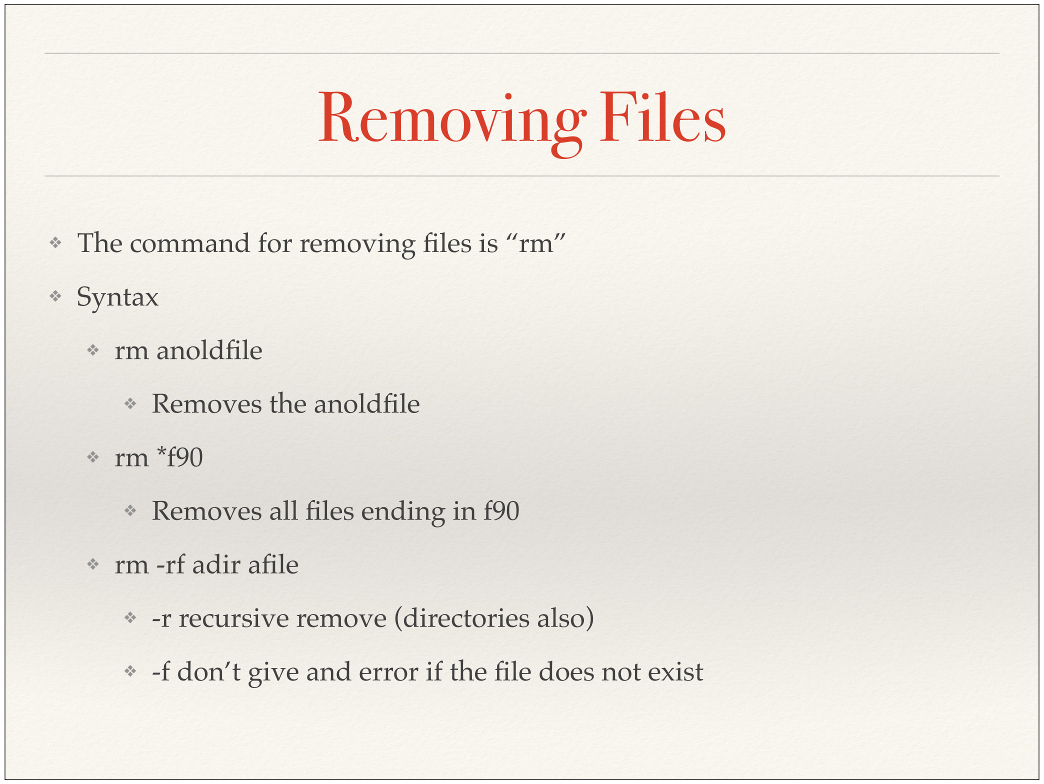


More Piping (output)

- You can put errors in a file and output in another, just save errors or have both go to a file or to the terminal
- command 1> cmd.out 2> cmd.err
  - Send normal output to cmd.out and errors to cmd.err
- command 2> cmd.err
  - Send errors to a file normal output would go to the screen
- command > both 2>&1
  - Send errors and output to a file "both"
- command 2>&1
  - Send errors to the terminal along with the standard output. This would normally be used if you want to pipe errors into another command.
  - Usage example: module avail 2>&1 | sort

Here are some more examples of sending errors and output to different places. In general the number 1 and a greater than symbol will redirect normal output. The number 2 and a greater than symbol will redirect error output.

The last form is used for oddball commands that send their normal output as errors. The — module avail — command is one such command. The only way you can use the vertical bar pipe option with such commands is to first redirect the error output to standard out. For example we show what you would need to do to sort the output from the module avail command.

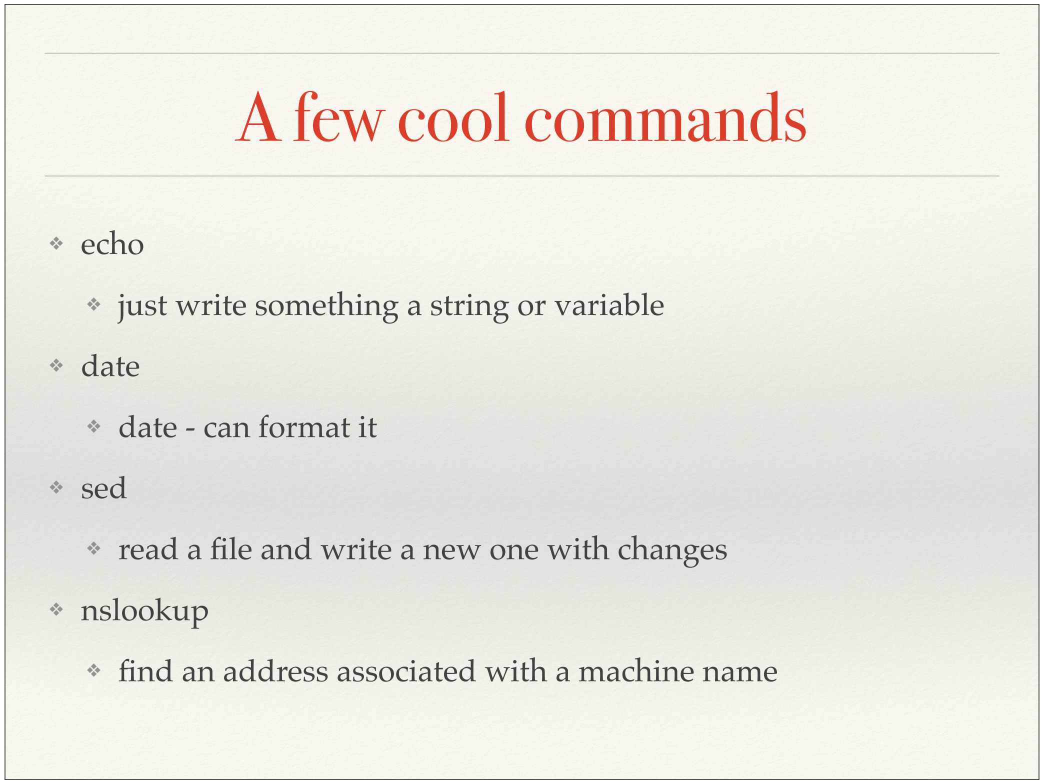


Removing Files

- The command for removing files is “rm”
- Syntax
  - rm anoldfile
    - Removes the anoldfile
  - rm *f90
    - Removes all files ending in f90
  - rm -rf adir afile
    - -r recursive remove (directories also)
    - -f don’t give and error if the file does not exist

R M is the command to remove files. You can name files to remove or again use wild cards. The minus R option will do a recursive remove. That is it will remove directories. This is very dangerous to use with wild cards in that it is possible to remove all of your files.

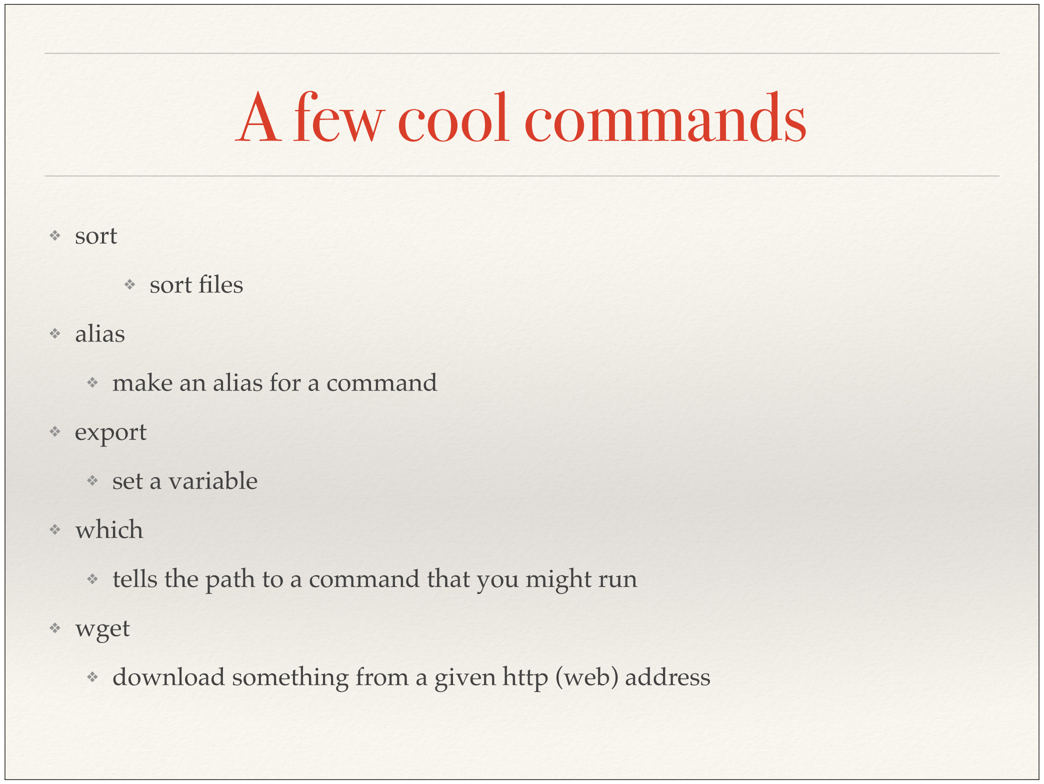
P. S. we do not do backups. we do not do backups. we do not do backups. we do not do backups.



A few cool commands

- echo
  - just write something a string or variable
- date
  - date - can format it
- sed
  - read a file and write a new one with changes
- nslookup
  - find an address associated with a machine name

Here are some interesting commands. The echo command just write something to the screen, technically standard out. The sed command can be used to make changes in files without invoking an editor.



A few cool commands

- sort
    - sort files
- alias
    - make an alias for a command
- export
    - set a variable
- which
    - tells the path to a command that you might run
- wget
    - download something from a given http (web) address

The sort command will sort the text in a file and output it to the screen. There are many options including being able to sort by fields in a file or sort numerically.

The alias command allows you to set up easy to remember substitutions for commands.

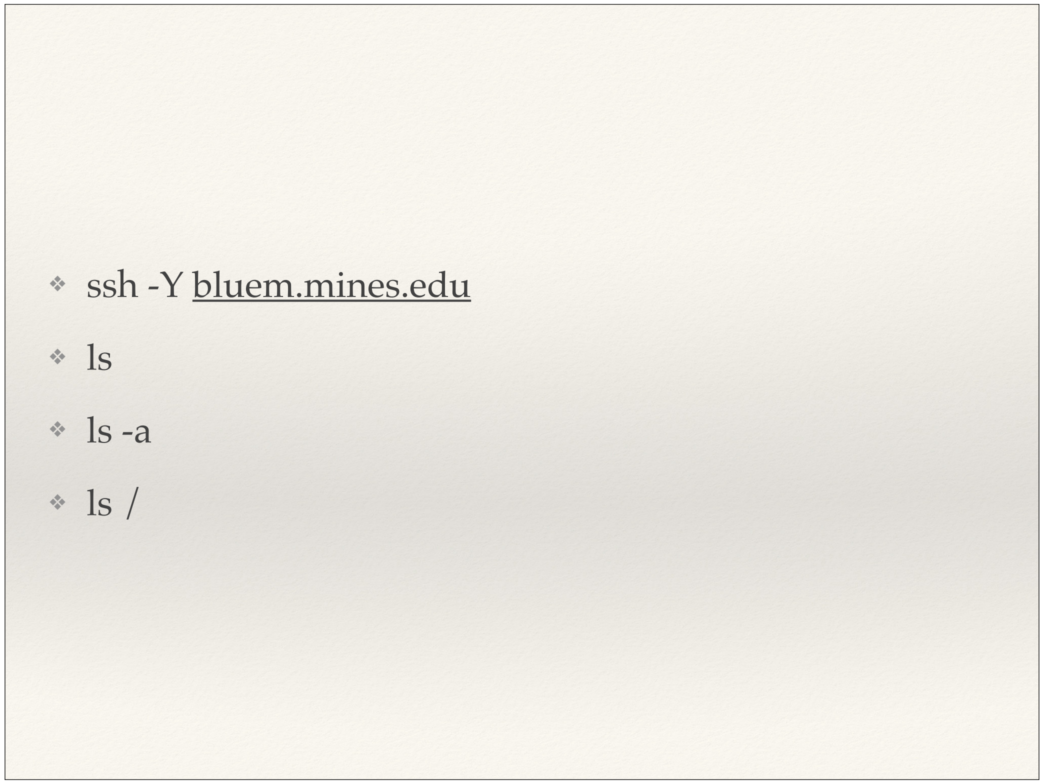
Export sets a variable. We will see more on this shortly.

Which tells the path to a command. This is useful if there might be multiple versions of a command.

W get allows you to download something from a web page. This is used most often when downloading programs.

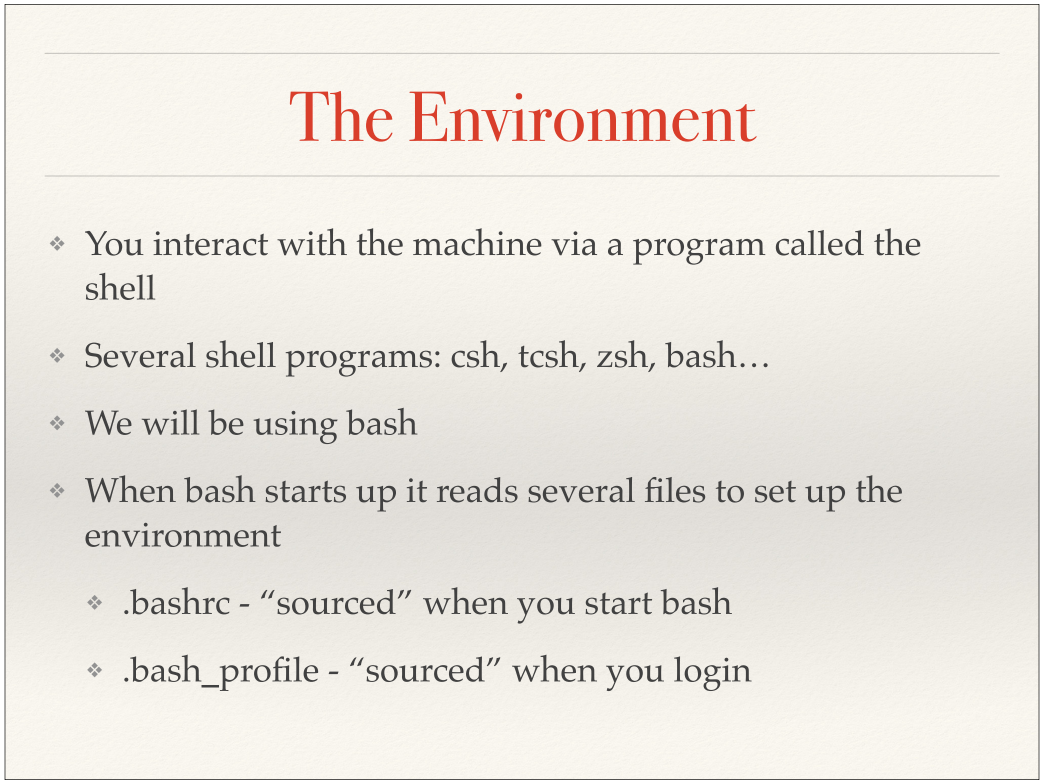
Let's Do It

Here are some things to try.



- ssh -Y bluem.mines.edu
- ls
- ls -a
- ls /

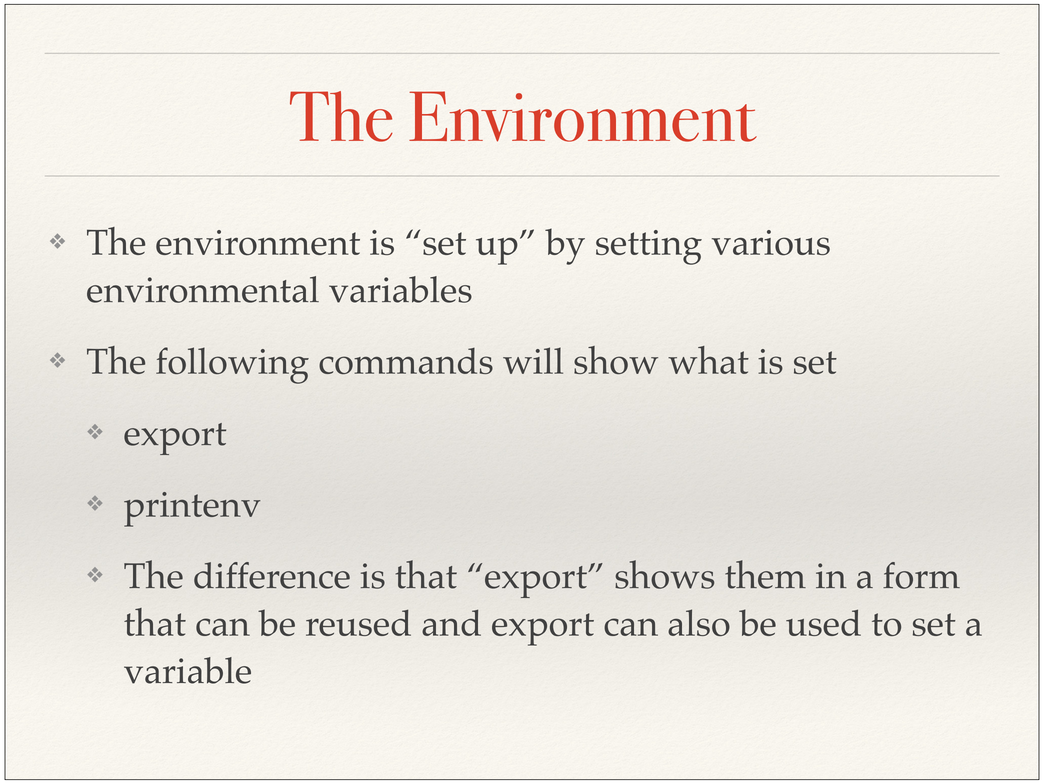
If you logon to blue m from your local box and do an l s you will see the files in your home directory. l s minus a show all of the hidden files and directories. Notice that you have a dot ssh directory that we discussed previously. If you do an l s slash you will see the root of the file system.



The Environment

- You interact with the machine via a program called the shell
- Several shell programs: csh, tcsh, zsh, bash…
- We will be using bash
- When bash starts up it reads several files to set up the environment
    - .bashrc - “sourced” when you start bash
    - .bash_profile - “sourced” when you login

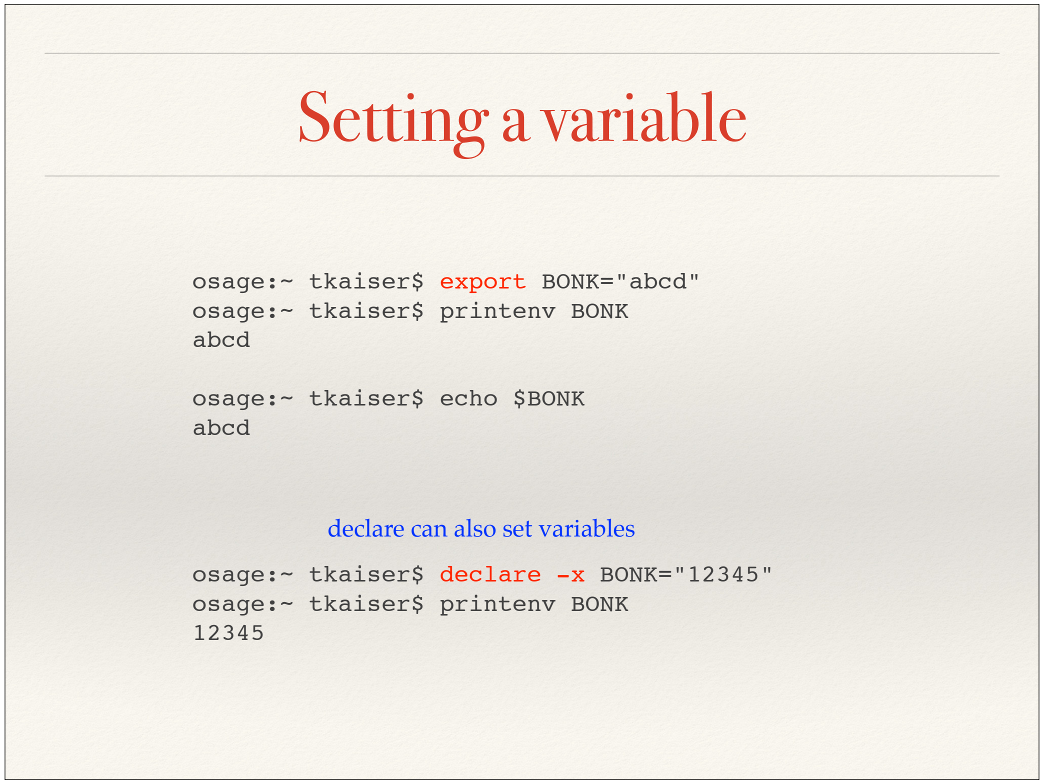
When you login to a linux box there is a program started automatically for you, a shell. The shell is the program that you use to interact with the machine, that is to give it commands. There are several different shell programs, for example C shell, T shell, Z shell, korn, and bash. Bash is fairly common and the shell we will be using.



The Environment

- The environment is “set up” by setting various environmental variables
- The following commands will show what is set
  - export
  - printenv
  - The difference is that “export” shows them in a form that can be reused and export can also be used to set a variable

The environment is “set up” by setting various environmental variables. The commands export and print e n v will show what variables you have set up. You can also use the export command to set variables.



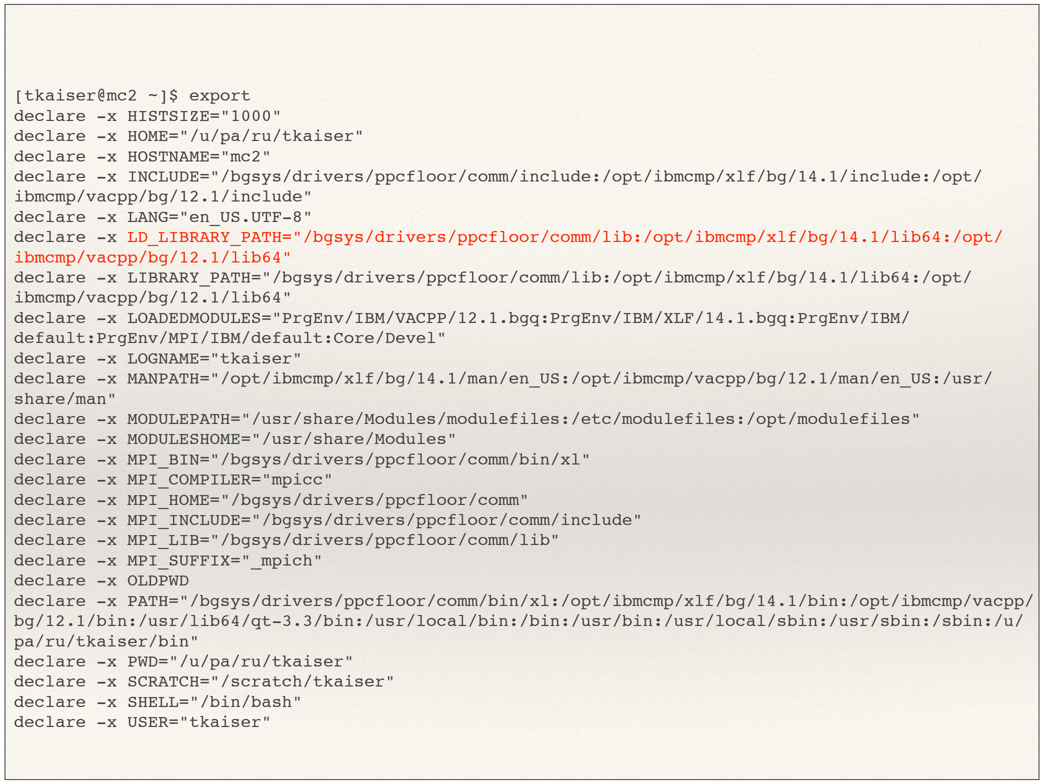
Setting a variable

```
osage:~ tkaiser$ export BONK="abcd"
osage:~ tkaiser$ printenv BONK
abcd

osage:~ tkaiser$ echo $BONK
abcd
```

declare can also set variables

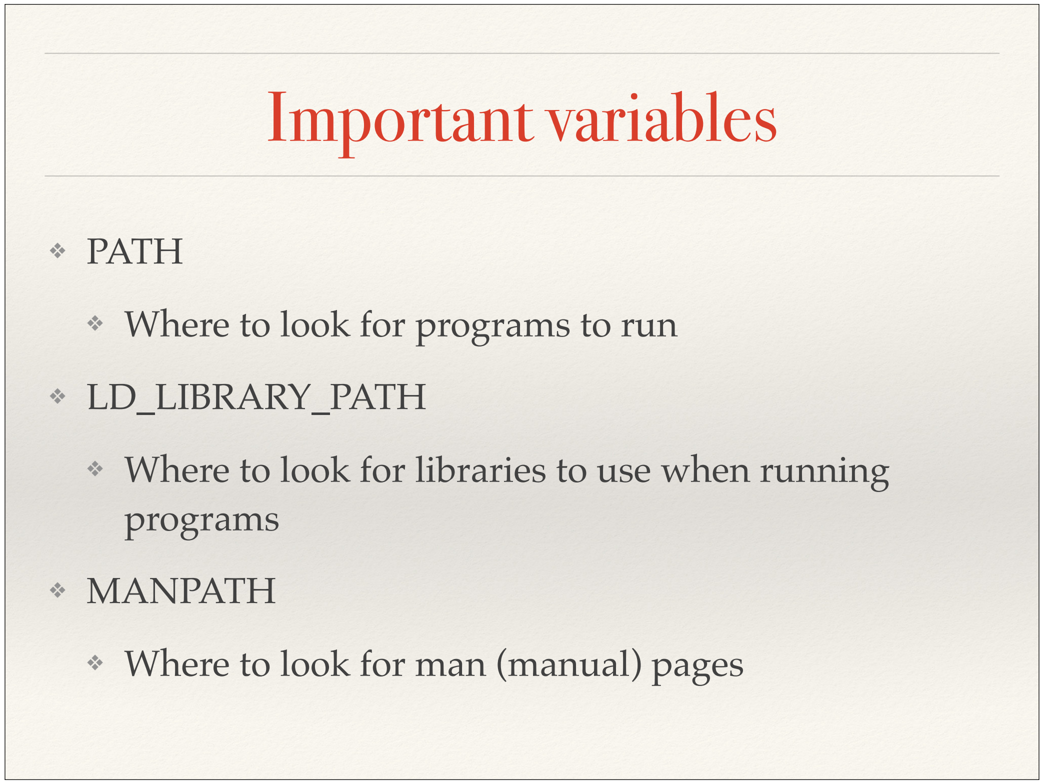
```
osage:~ tkaiser$ declare -x BONK="12345"
osage:~ tkaiser$ printenv BONK
12345
```

We can use the export option to set a variable. Then we can see the value using the print e n v command. Echo can also be used to show a variable. Declare can also be used. It actually has a bunch of options for doing things like locking the value of a variable. For example if you use the minus r option it makes names read only.   These names cannot then be assigned values by subsequent assignment statements or unset.



```
[tkaiser@mc2 ~]$ export
declare -x HISTSIZE="1000"
declare -x HOME="/u/pa/ru/tkaiser"
declare -x HOSTNAME="mc2"
declare -x INCLUDE="/bgsys/drivers/ppcfloor/comm/include:/opt/ibmcmp/xlf/bg/14.1/include:/opt/
ibmcmp/vacpp/bg/12.1/include"
declare -x LANG="en_US.UTF-8"
declare -x LD_LIBRARY_PATH="/bgsys/drivers/ppcfloor/comm/lib:/opt/ibmcmp/xlf/bg/14.1/lib64:/opt/
ibmcmp/vacpp/bg/12.1/lib64"
declare -x LIBRARY_PATH="/bgsys/drivers/ppcfloor/comm/lib:/opt/ibmcmp/xlf/bg/14.1/lib64:/opt/
ibmcmp/vacpp/bg/12.1/lib64"
declare -x LOADEDMODULES="PrgEnv/IBM/VACPP/12.1.bgq:PrgEnv/IBM/XLF/14.1.bgq:PrgEnv/IBM/
default:PrgEnv/MPI/IBM/default:Core/Devel"
declare -x LOGNAME="tkaiser"
declare -x MANPATH="/opt/ibmcmp/xlf/bg/14.1/man/en_US:/opt/ibmcmp/vacpp/bg/12.1/man/en_US:/usr/
share/man"
declare -x MODULEPATH="/usr/share/Modules/modulefiles:/etc/modulefiles:/opt/modulefiles"
declare -x MODULESHOME="/usr/share/Modules"
declare -x MPI_BIN="/bgsys/drivers/ppcfloor/comm/bin/xl"
declare -x MPI_COMPILER="mpicc"
declare -x MPI_HOME="/bgsys/drivers/ppcfloor/comm"
declare -x MPI_INCLUDE="/bgsys/drivers/ppcfloor/comm/include"
declare -x MPI_LIB="/bgsys/drivers/ppcfloor/comm/lib"
declare -x MPI_SUFFIX="_mpich"
declare -x OLDPWD
declare -x PATH="/bgsys/drivers/ppcfloor/comm/bin/xl:/opt/ibmcmp/xlf/bg/14.1/bin:/opt/ibmcmp/vacpp/
bg/12.1/bin:/usr/lib64/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/u/
pa/ru/tkaiser/bin"
declare -x PWD="/u/pa/ru/tkaiser"
declare -x SCRATCH="/scratch/tkaiser"
declare -x SHELL="/bin/bash"
declare -x USER="tkaiser"
```

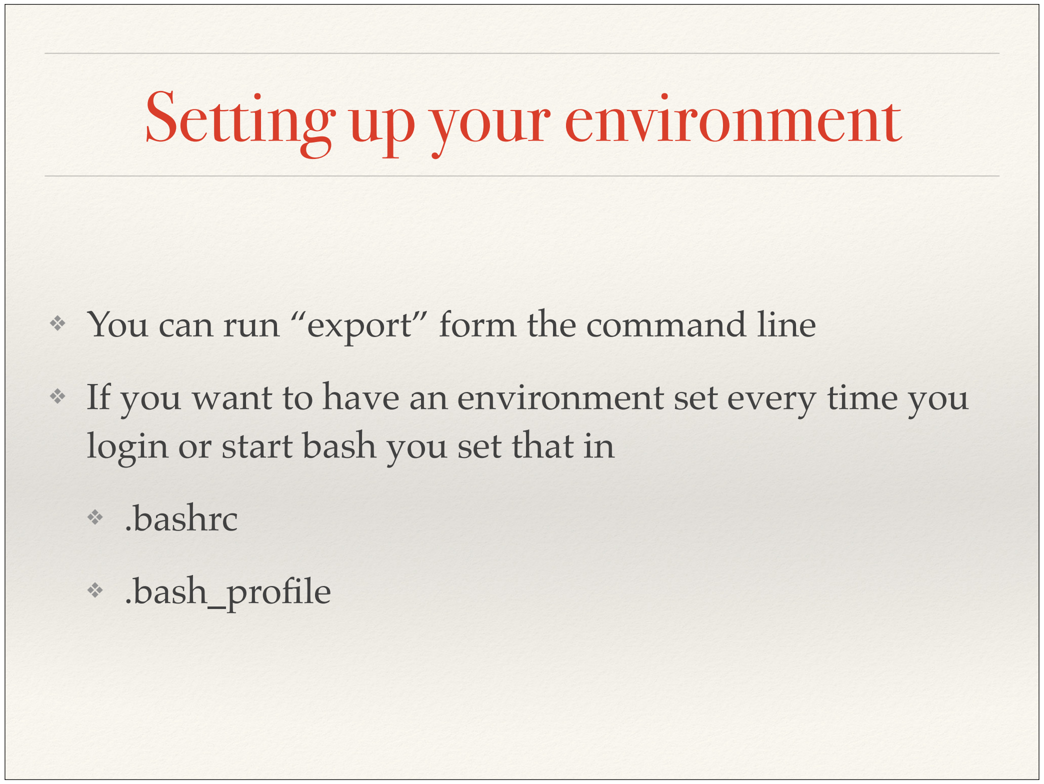
Here is the output from the export command. It shows the declare commands that would be used to set all of the variables.



Important variables

- PATH
  - Where to look for programs to run
- LD_LIBRARY_PATH
  - Where to look for libraries to use when running programs
- MANPATH
  - Where to look for man (manual) pages

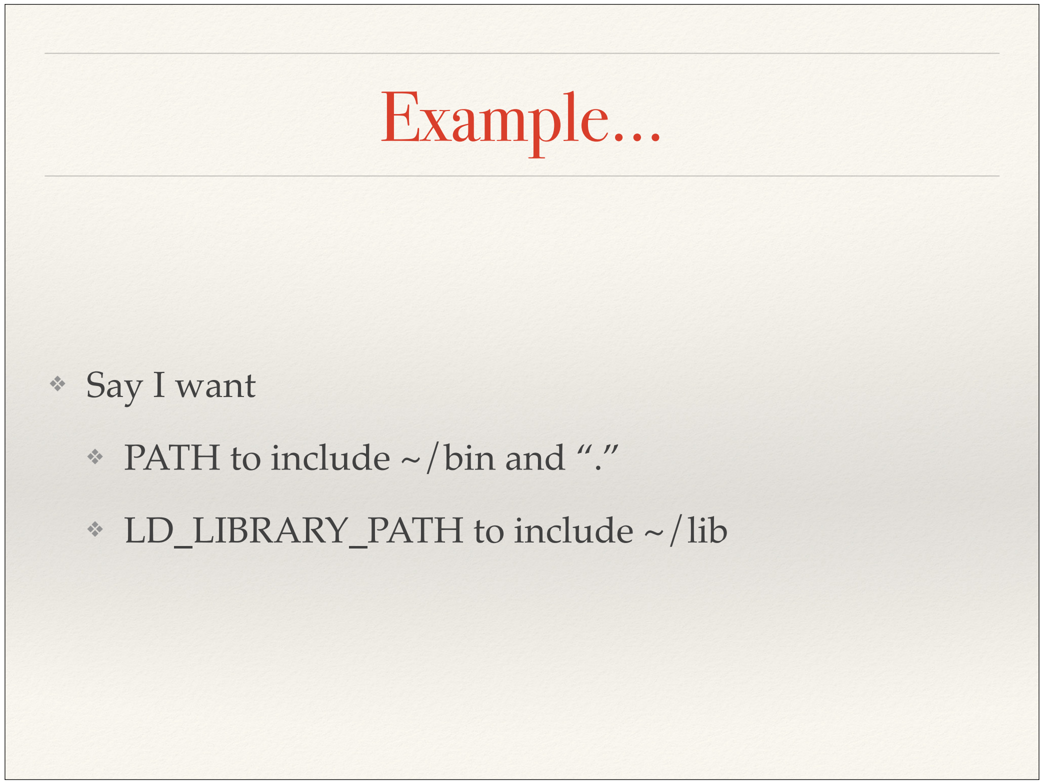
There are some important environmental variables. The PATH variable tells the system where to look for programs. The PATH variable contains a list of directories. If a program you want to run is not in one of the directories on the list then the system will not know how to find it and you must provide the path to where it is located. Some programs need extra libraries to run that are not part of the program. LD_LIBRARY_PATH is similar to PATH except it tells the system where to find libraries. MAN PATH tells the system where to look for manual pages.



Setting up your environment

- You can run “export” form the command line
- If you want to have an environment set every time you login or start bash you set that in
    - .bashrc
    - .bash_profile

You can use the export command to set up your environment. However, if you want the same settings every time you login you can put the export commands in your dot bash rc file or your dot bash profile file.

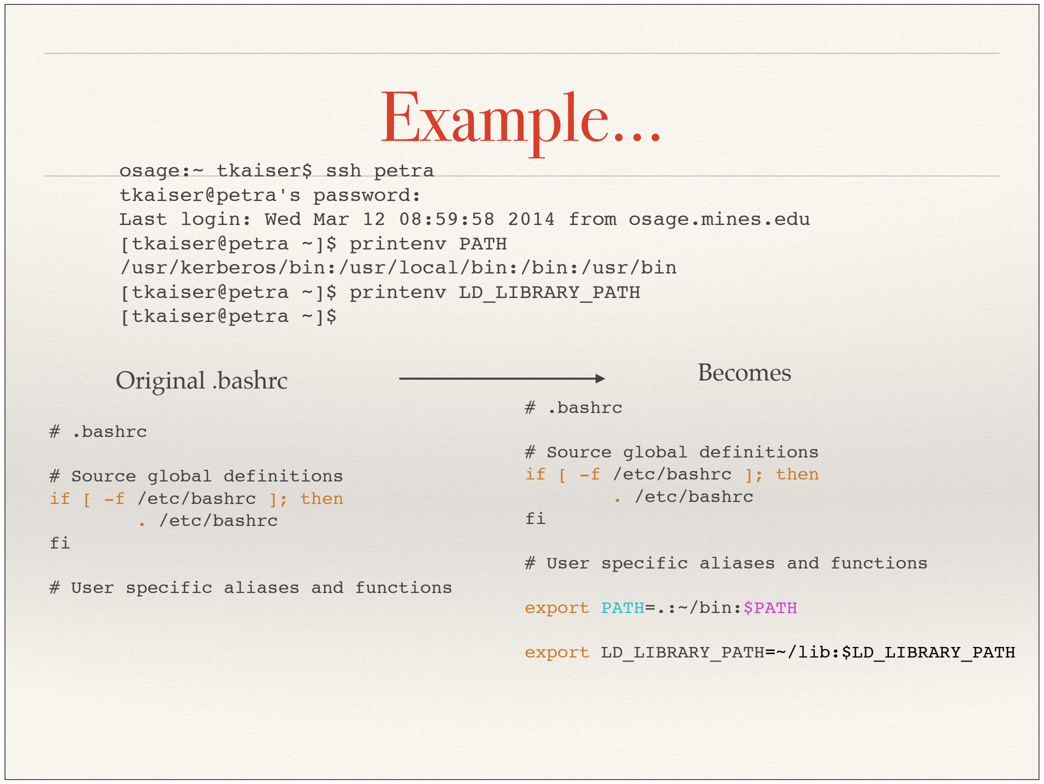


Example…

- Say I want
  - PATH to include ~/bin and “.”
  - LD_LIBRARY_PATH to include ~/lib

Lets say I want my path to include a bin directory off of my home directory. Recall that the tilde character is short hand for home directory. Also we want the system to look for programs in our current directory. The period is short hand for the current directory.

We also want the system to look for libraries in the lib directory below our home directory.



Example…

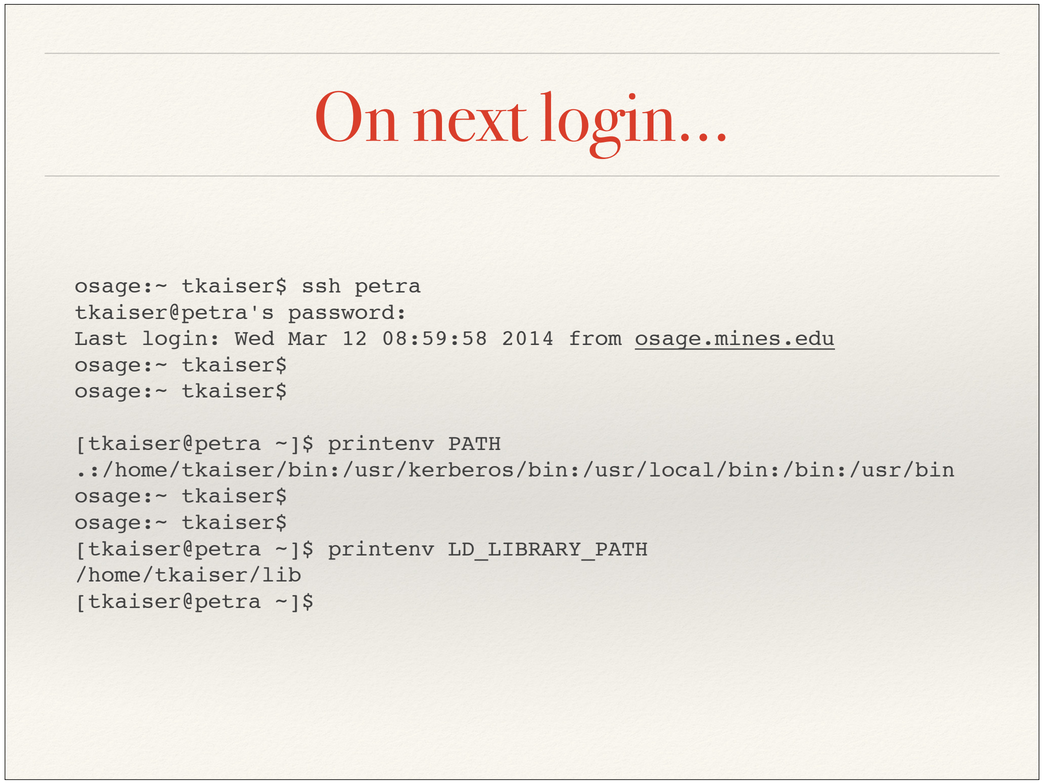
```
osage:~ tkaiser$ ssh petra
tkaiser@petra's password:
Last login: Wed Mar 12 08:59:58 2014 from osage.mines.edu
[tkaiser@petra ~]$ printenv PATH
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin
[tkaiser@petra ~]$ printenv LD_LIBRARY_PATH
[tkaiser@petra ~]$
```

Original .bashrc $\longrightarrow$ Becomes

```
# .bashrc

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

# User specific aliases and functions
```

```
# .bashrc

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

# User specific aliases and functions

export PATH=.:~/bin:$PATH

export LD_LIBRARY_PATH=~/lib:$LD_LIBRARY_PATH
```

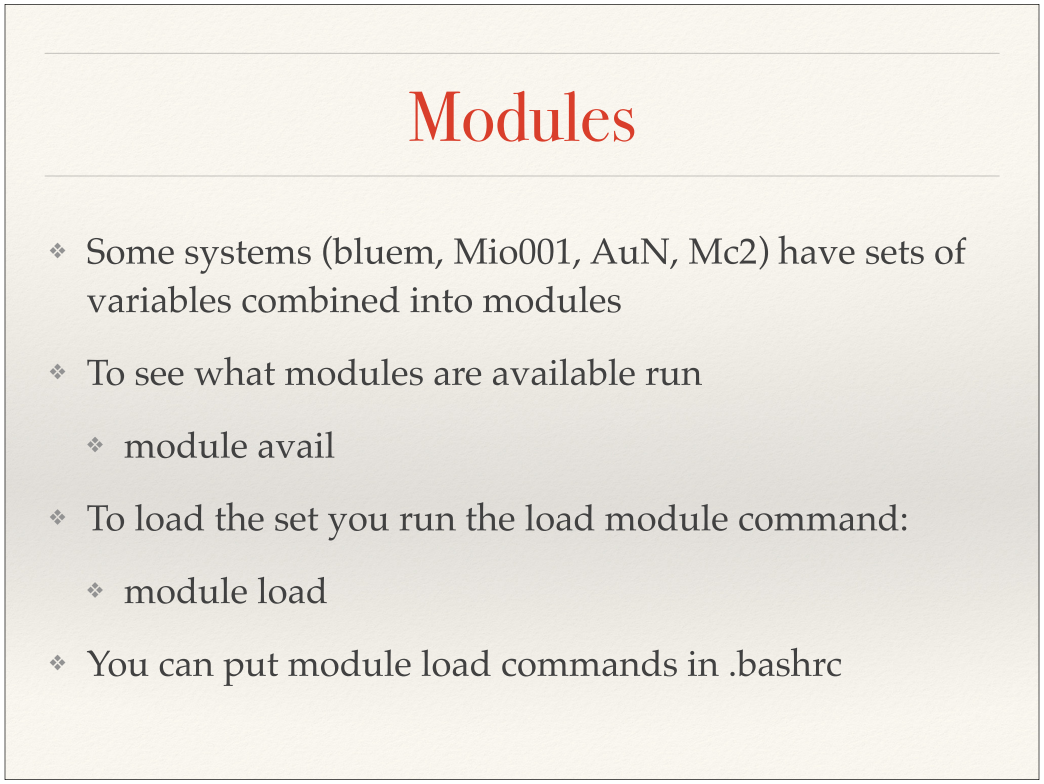
We will want to edit our dot bash RC file and add the lines shown here. The colon character separates directories in the list. The other thing we do here is put $ path and $ LD_LIBRARY_PATH at the end. This causes the new directories to be added at the beginning of our old lists.



On next login…

```
osage:~ tkaiser$ ssh petra
tkaiser@petra's password:
Last login: Wed Mar 12 08:59:58 2014 from osage.mines.edu
osage:~ tkaiser$
osage:~ tkaiser$

[tkaiser@petra ~]$ printenv PATH
.:/home/tkaiser/bin:/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin
osage:~ tkaiser$
osage:~ tkaiser$
[tkaiser@petra ~]$ printenv LD_LIBRARY_PATH
/home/tkaiser/lib
[tkaiser@petra ~]$
```

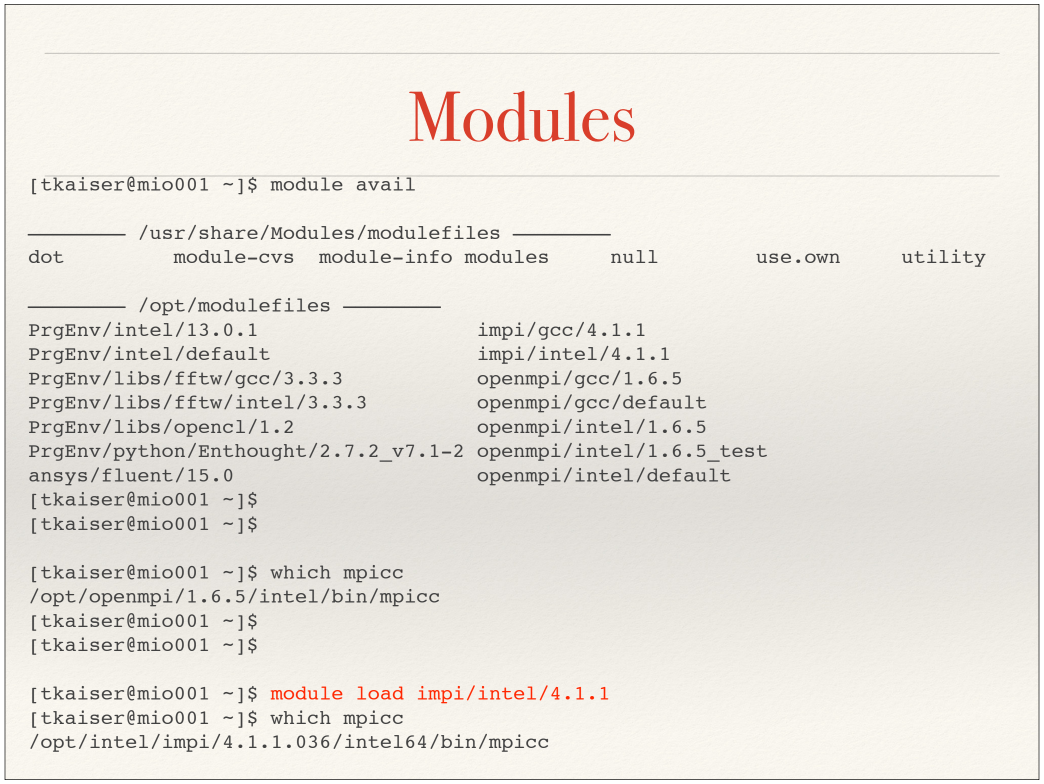
The next time you login the dot bash R C file is read and your environment is set with the new LD Library Path and you new Path.



Modules

- Some systems (bluem, Mio001, AuN, Mc2) have sets of variables combined into modules
- To see what modules are available run
  - module avail
- To load the set you run the load module command:
  - module load
- You can put module load commands in .bashrc

Many computing platforms have a feature called modules. A module is a short file that contains a collection of commands to set environmental variables. You can see what modules are available by running the command: module avail. Modules are often used to set up the environment to run a particular program. That is, it contains all of the setting to enable the program to work properly. The command, module load, followed by the name of the module will load it and import the environmental variable settings. You can, if you like, put module load commands in your dot bash rc file and they will be loaded when you log in so your environment will be set up to run your programs.

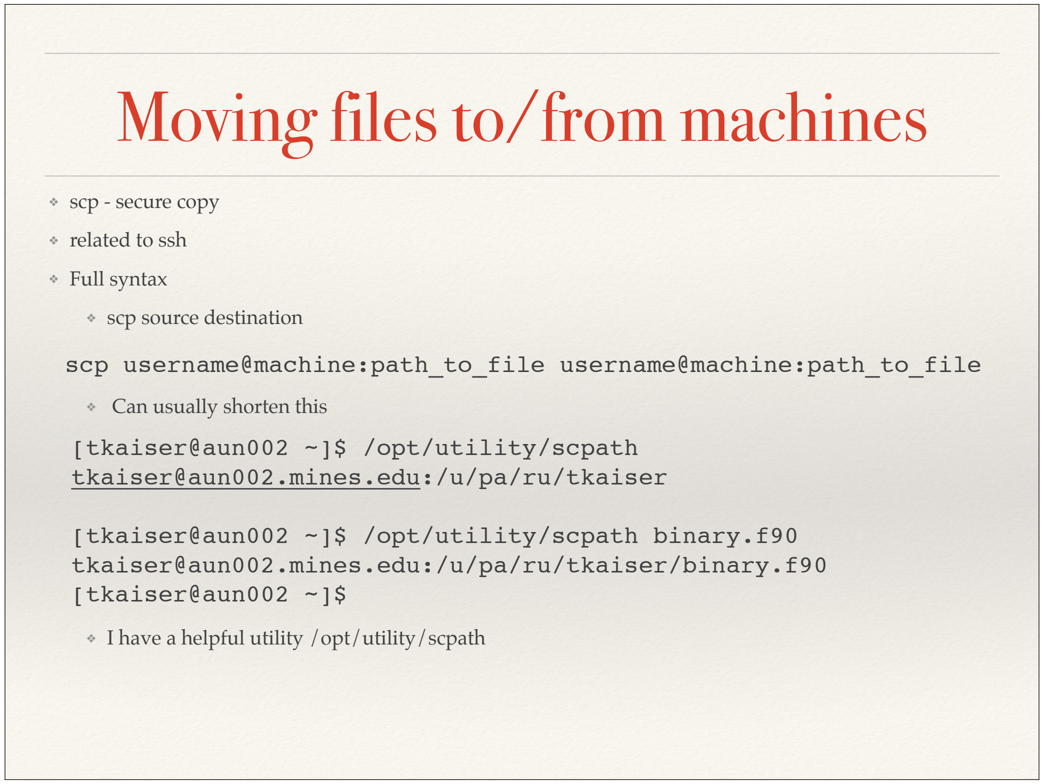


Modules

```
[tkaiser@mio001 ~]$ module avail

———————— /usr/share/Modules/modulefiles ————————
dot          module-cvs  module-info modules     null        use.own     utility

———————— /opt/modulefiles ————————
PrgEnv/intel/13.0.1                  impi/gcc/4.1.1
PrgEnv/intel/default                 impi/intel/4.1.1
PrgEnv/libs/fftw/gcc/3.3.3           openmpi/gcc/1.6.5
PrgEnv/libs/fftw/intel/3.3.3         openmpi/gcc/default
PrgEnv/libs/opencl/1.2               openmpi/intel/1.6.5
PrgEnv/python/Enthought/2.7.2_v7.1-2 openmpi/intel/1.6.5_test
ansys/fluent/15.0                    openmpi/intel/default
[tkaiser@mio001 ~]$
[tkaiser@mio001 ~]$

[tkaiser@mio001 ~]$ which mpicc
/opt/openmpi/1.6.5/intel/bin/mpicc
[tkaiser@mio001 ~]$
[tkaiser@mio001 ~]$

[tkaiser@mio001 ~]$ module load impi/intel/4.1.1
[tkaiser@mio001 ~]$ which mpicc
/opt/intel/impi/4.1.1.036/intel64/bin/mpicc
```

Here we see the modules available on our machine mio by running the module avail command.

Next we run the command which M P I C C. This shows which version of this program is currently our default. Then we load a module. Then we run the which command again and see that the default version of M P I C C has changed.



Moving files to/from machines

- scp - secure copy
- related to ssh
- Full syntax
  - scp source destination

```
scp username@machine:path_to_file username@machine:path_to_file
```

  - Can usually shorten this

```
[tkaiser@aun002 ~]$ /opt/utility/scpath
tkaiser@aun002.mines.edu:/u/pa/ru/tkaiser

[tkaiser@aun002 ~]$ /opt/utility/scpath binary.f90
tkaiser@aun002.mines.edu:/u/pa/ru/tkaiser/binary.f90
[tkaiser@aun002 ~]$
```

  - I have a helpful utility /opt/utility/scpath

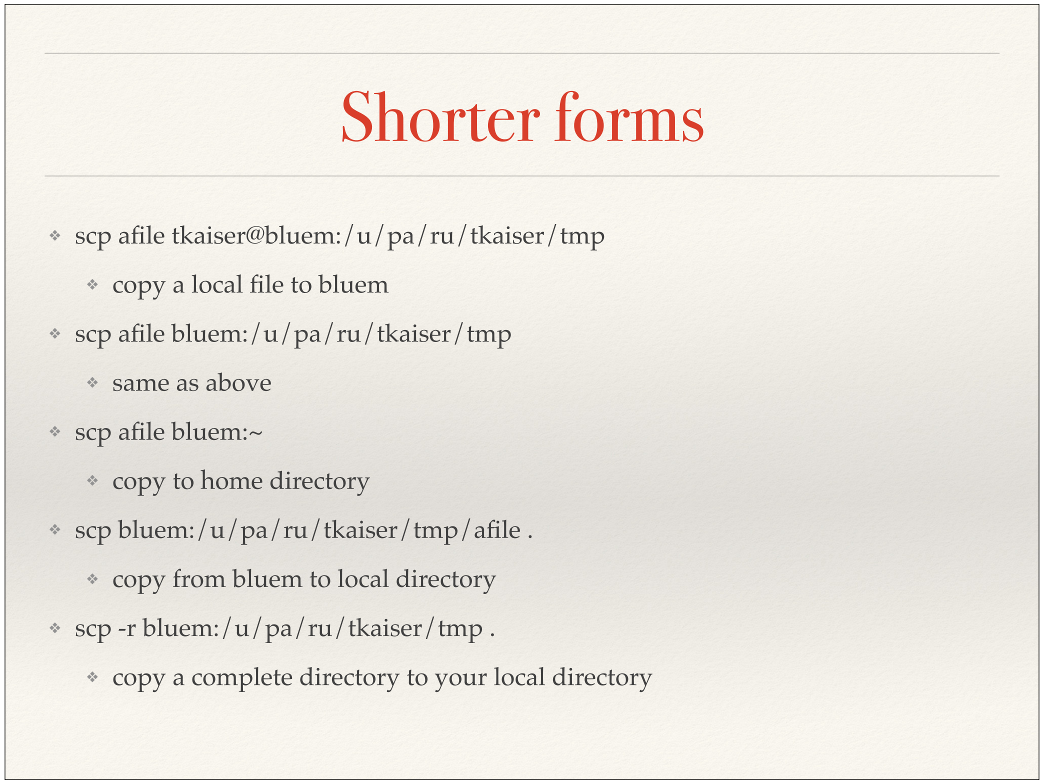
The program that is normally used to copy files between machines is SCP. SCP is related to SSH in that is shares the same security infrastructure core and it uses the same configuration file, dot config.

SCP is very versatile. It can be used to pull a file from a remote machine or push it to the machine. It can actually be used to move a file between two machines, neither of which you are logged into.

The syntax for the command is SCP followed by the file you are copying and then where it is going to. The difficulty here is the long form of the command is rather cumbersome. File names are of the form username, then an “@“ symbol then the machine name, then a colon and then a full path to the location on the machine for the file.

You can usually shorten this. If your username is the same on both machines it is not required. If the file you are moving is on the machine from which you are running the command you can skip the machine name. Finally if the file is in your current directory or is being copied to your current directory you only need a local name. You can also just list the destination directory instead of the directory and file name

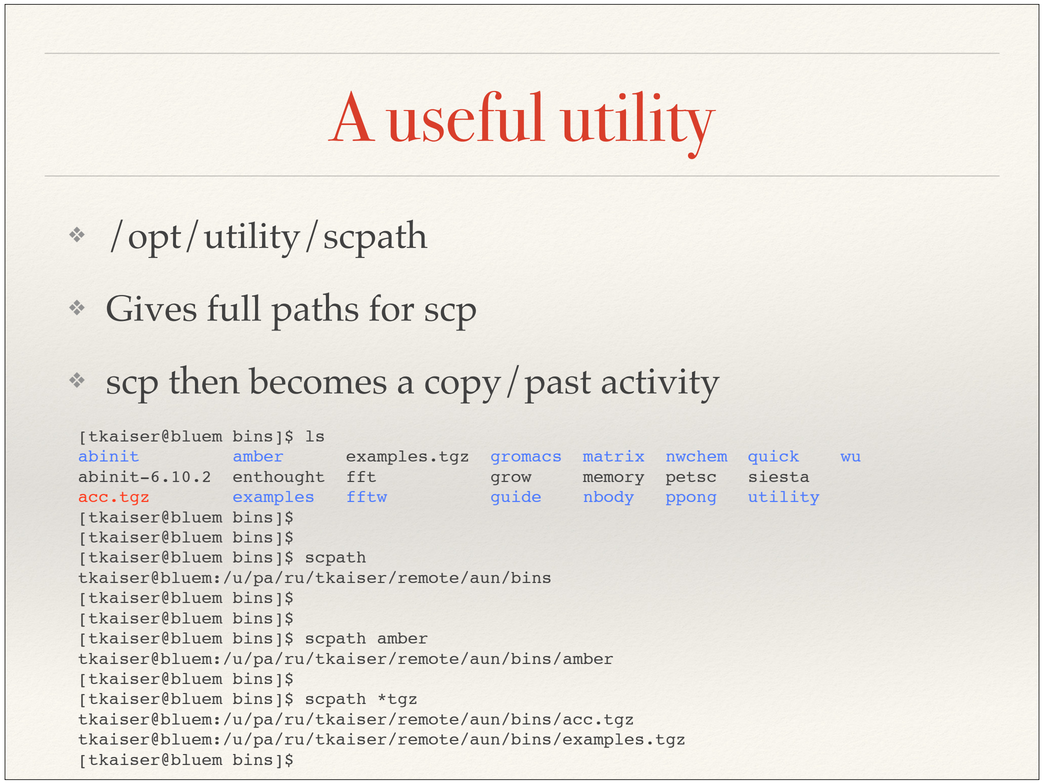
We have a command, opt utility s c path that will give the fully qualified path for a file or a directory. The way it is used is that if



Shorter forms

- `scp afile tkaiser@bluem:/u/pa/ru/tkaiser/tmp`
  - copy a local file to bluem
- `scp afile bluem:/u/pa/ru/tkaiser/tmp`
  - same as above
- `scp afile bluem:~`
  - copy to home directory
- `scp bluem:/u/pa/ru/tkaiser/tmp/afile .`
  - copy from bluem to local directory
- `scp -r bluem:/u/pa/ru/tkaiser/tmp .`
  - copy a complete directory to your local directory

Here are some examples. First we copy a local file to a particular directory on blue m. Because my user name is the same on both machines it can be skipped. In the third example we are doing the copy to our home directory. Next we copy a file from blue m to our local directory. Note the dot at the end of the command. Finally, we use the minus r option to copy a complete directory.

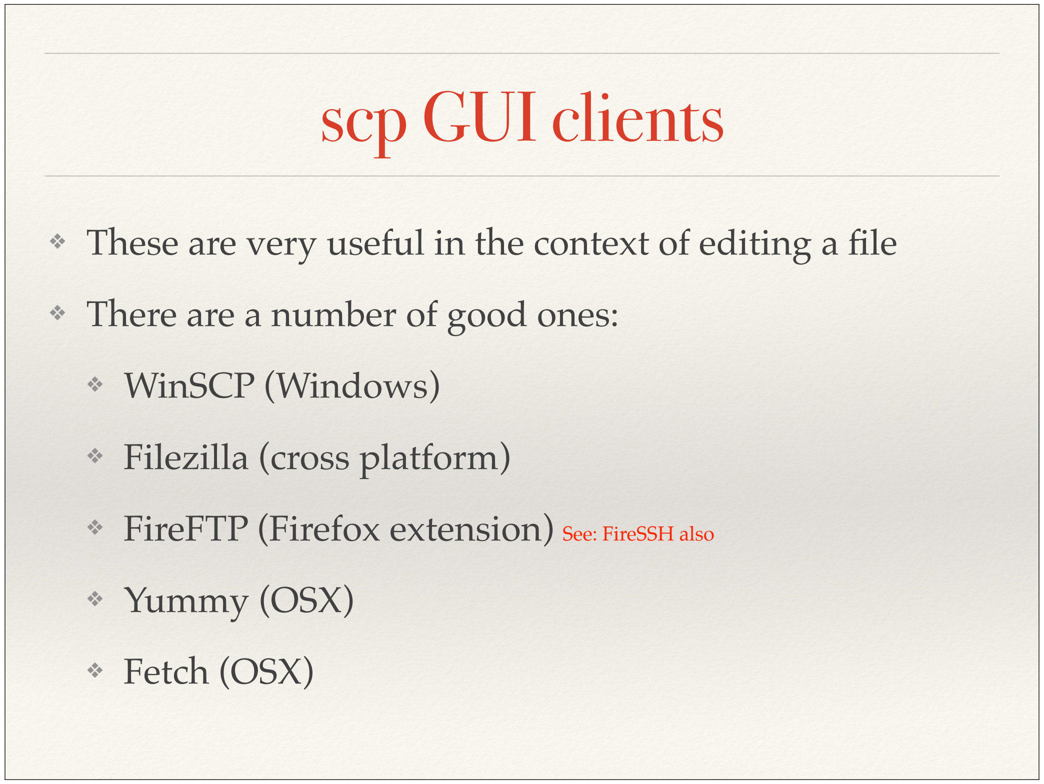


A useful utility

- /opt/utility/scpath
- Gives full paths for scp
- scp then becomes a copy/past activity

```
[tkaiser@bluem bins]$ ls
abinit         amber      examples.tgz  gromacs  matrix  nwchem  quick    wu
abinit-6.10.2  enthought  fft           grow     memory  petsc   siesta
acc.tgz        examples   fftw          guide    nbody   ppong   utility
[tkaiser@bluem bins]$
[tkaiser@bluem bins]$
[tkaiser@bluem bins]$ scpath
tkaiser@bluem:/u/pa/ru/tkaiser/remote/aun/bins
[tkaiser@bluem bins]$
[tkaiser@bluem bins]$
[tkaiser@bluem bins]$ scpath amber
tkaiser@bluem:/u/pa/ru/tkaiser/remote/aun/bins/amber
[tkaiser@bluem bins]$
[tkaiser@bluem bins]$ scpath *tgz
tkaiser@bluem:/u/pa/ru/tkaiser/remote/aun/bins/acc.tgz
tkaiser@bluem:/u/pa/ru/tkaiser/remote/aun/bins/examples.tgz
[tkaiser@bluem bins]$
```

Here are some additional examples of using the S C path utility. In the last example we see that we can use a wild card to get the path for multiple files.

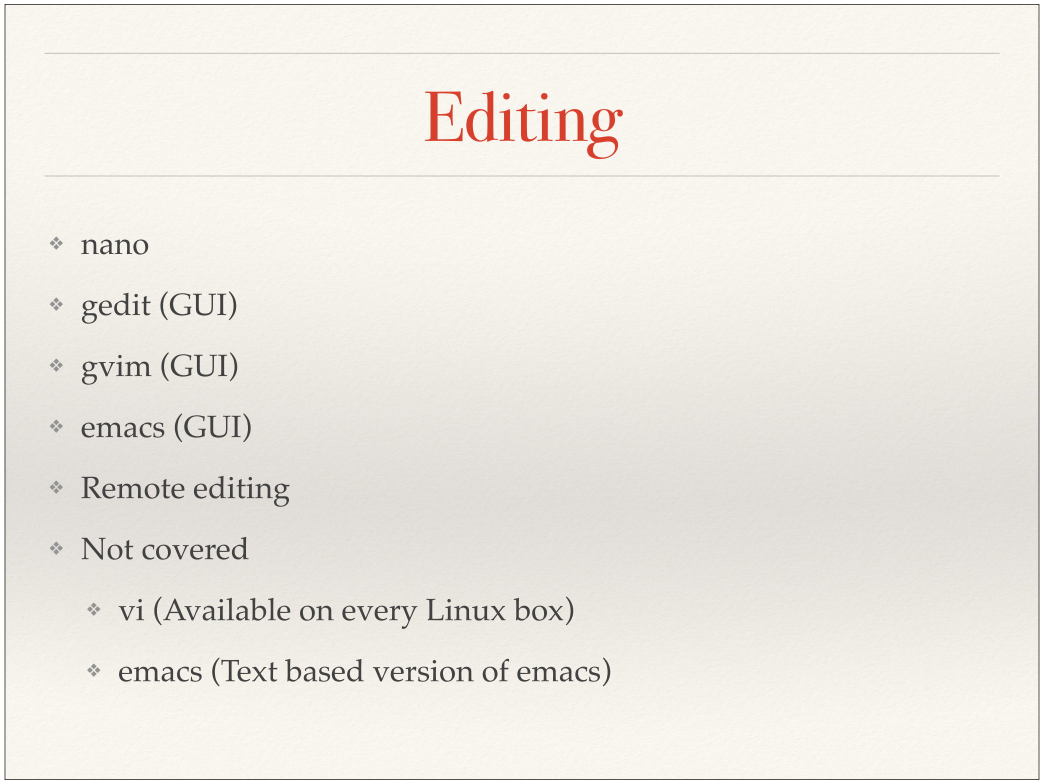


scp GUI clients

- These are very useful in the context of editing a file
- There are a number of good ones:
  - WinSCP (Windows)
  - Filezilla (cross platform)
  - FireFTP (Firefox extension) See: FireSSH also
  - Yummy (OSX)
  - Fetch (OSX)

There are a number of GUI wrappers for scp. Win SCP is common in the windows world. File Zilla is cross platform and works under linux, windows and on the Mac. Fire FTP is similar to Fire SSH and it works as an extension for Fire Fox. That is, you can use fire fox to transfer files. I use yummy on my mac but Fetch is also very good.

These utilities are useful in the context of editing files. What you can do, if you have an editor that you like on your local machine, use can use one of these utilities in connection with the editor. Many editors can be set up to automatically upload via these programs when the file is saved in your local editor.

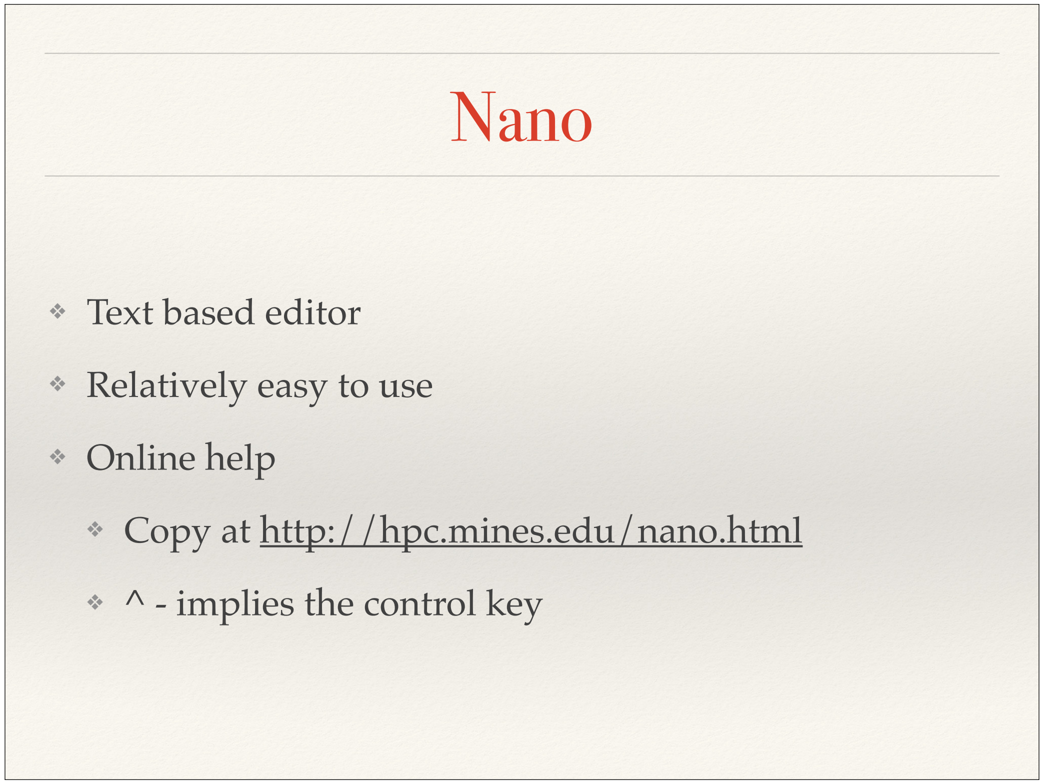


Editing

- nano
- gedit (GUI)
- gvim (GUI)
- emacs (GUI)
- Remote editing
- Not covered
  - vi (Available on every Linux box)
  - emacs (Text based version of emacs)

For now, we are going to talk about editors available on linux boxes. Most linux geeks would say the the VI editor is the standard. However, we are not going to talk about it. It is very power full but a bit difficult to learn to use. The second geek editor is emacs. It is a bit easier but again we'll skip it. There are however GUI versions of VI, called gvim, and emacs.

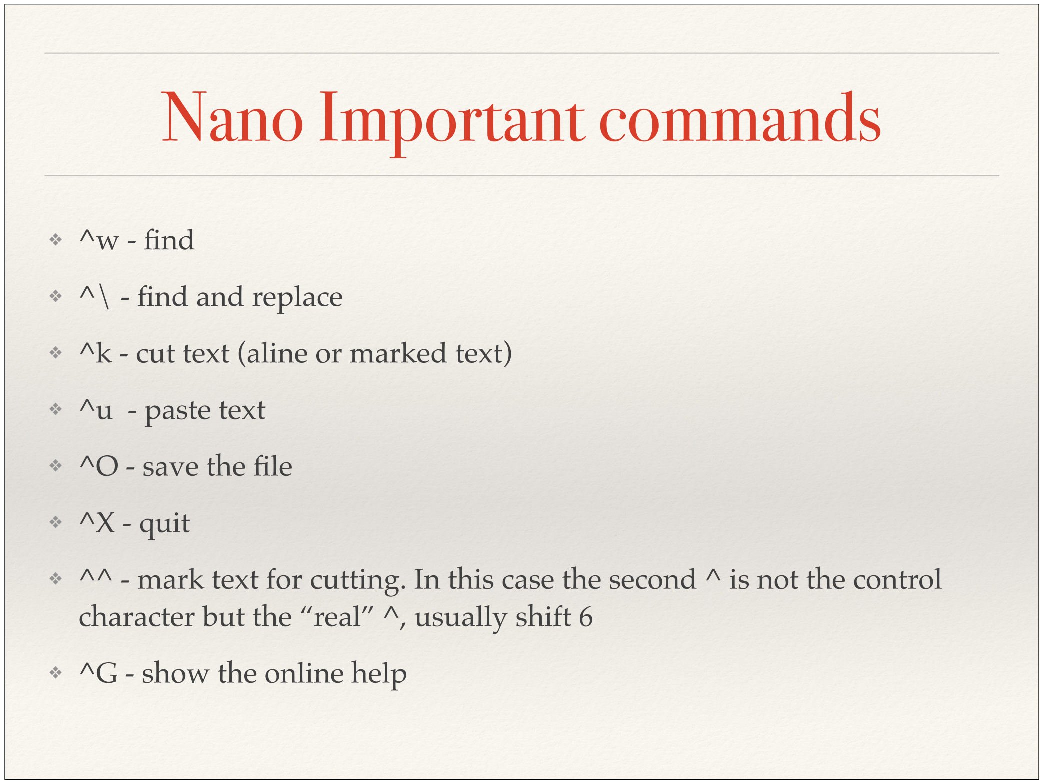
We'll start with nano and then look at some of the GUI editors. We'll also look at what we discussed previously, that is using a local editor along with one of the SCP transfer programs.



Nano

- Text based editor
- Relatively easy to use
- Online help
  - Copy at http://hpc.mines.edu/nano.html
  - ^ - implies the control key

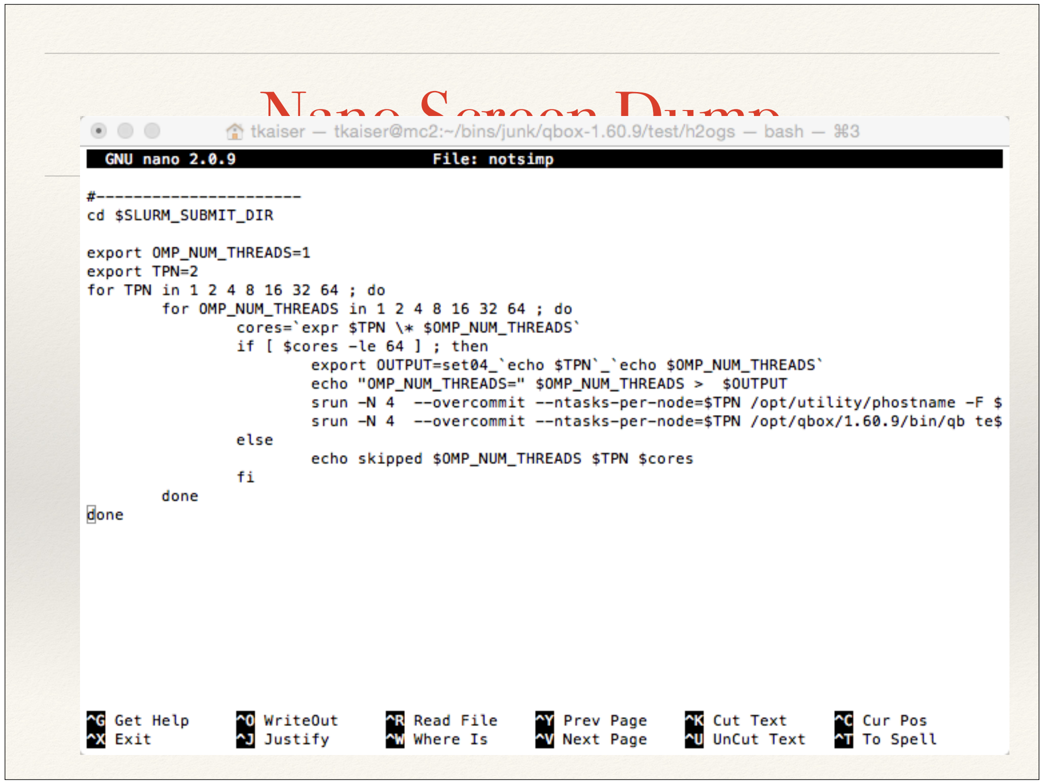
Nano is an easy to use text based editor that should work form many purposes. There is an online help menu that is accessed by entering control G. We have a copy of the help page locally. In the documentation the up carrot implies the control key.



Nano Important commands

- ^w - find
- ^\ - find and replace
- ^k - cut text (aline or marked text)
- ^u - paste text
- ^O - save the file
- ^X - quit
- ^^ - mark text for cutting. In this case the second ^ is not the control character but the “real” ^, usually shift 6
- ^G - show the online help

Here are the important nano commands. If you do a control up carrot this will start to make a block of text for deleting. You then use control k to cut the text and control u to place it.

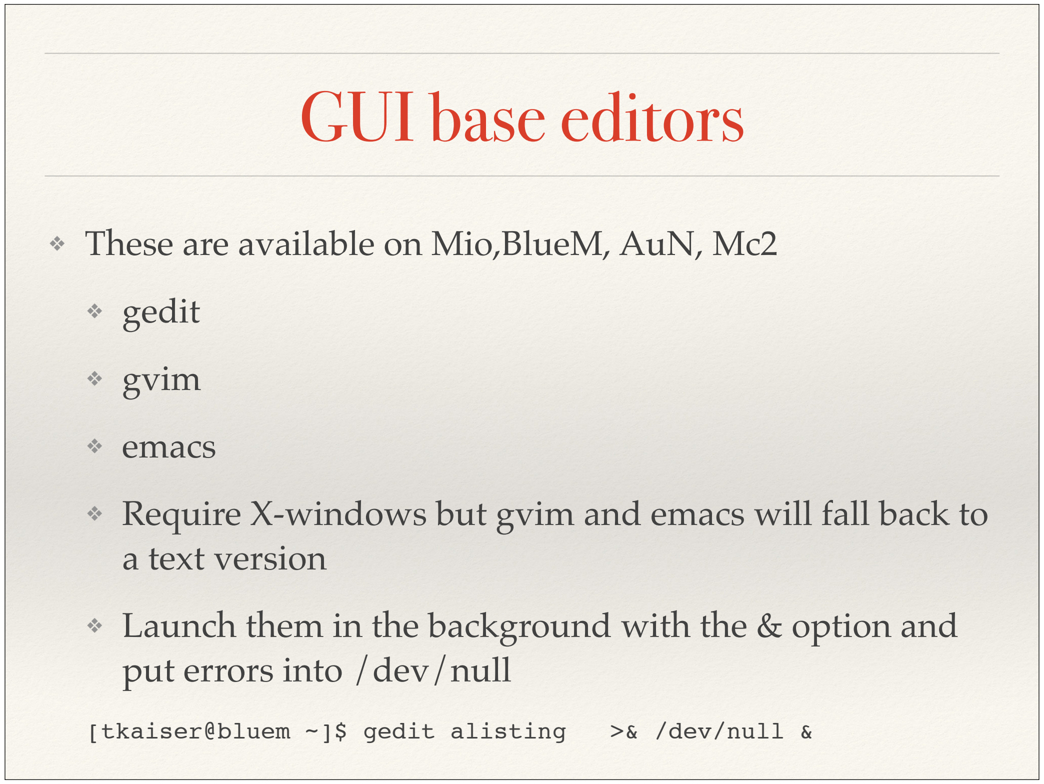


Nano Screen Dump

tkaiser — tkaiser@mc2:~/bins/junk/qbox-1.60.9/test/h2ogs — bash — ⌘3

```
  GNU nano 2.0.9                    File: notsimp

#----------------------
cd $SLURM_SUBMIT_DIR

export OMP_NUM_THREADS=1
export TPN=2
for TPN in 1 2 4 8 16 32 64 ; do
        for OMP_NUM_THREADS in 1 2 4 8 16 32 64 ; do
                cores=`expr $TPN \* $OMP_NUM_THREADS`
                if [ $cores -le 64 ] ; then
                        export OUTPUT=set04_`echo $TPN`_`echo $OMP_NUM_THREADS`
                        echo "OMP_NUM_THREADS=" $OMP_NUM_THREADS >  $OUTPUT
                        srun -N 4  --overcommit --ntasks-per-node=$TPN /opt/utility/phostname -F $
                        srun -N 4  --overcommit --ntasks-per-node=$TPN /opt/qbox/1.60.9/bin/qb te$
                else
                        echo skipped $OMP_NUM_THREADS $TPN $cores
                fi
        done
done








^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Here is a nano screen dump. It shows a number of the important commands at the bottom of the page. This is actually enough to get you started.



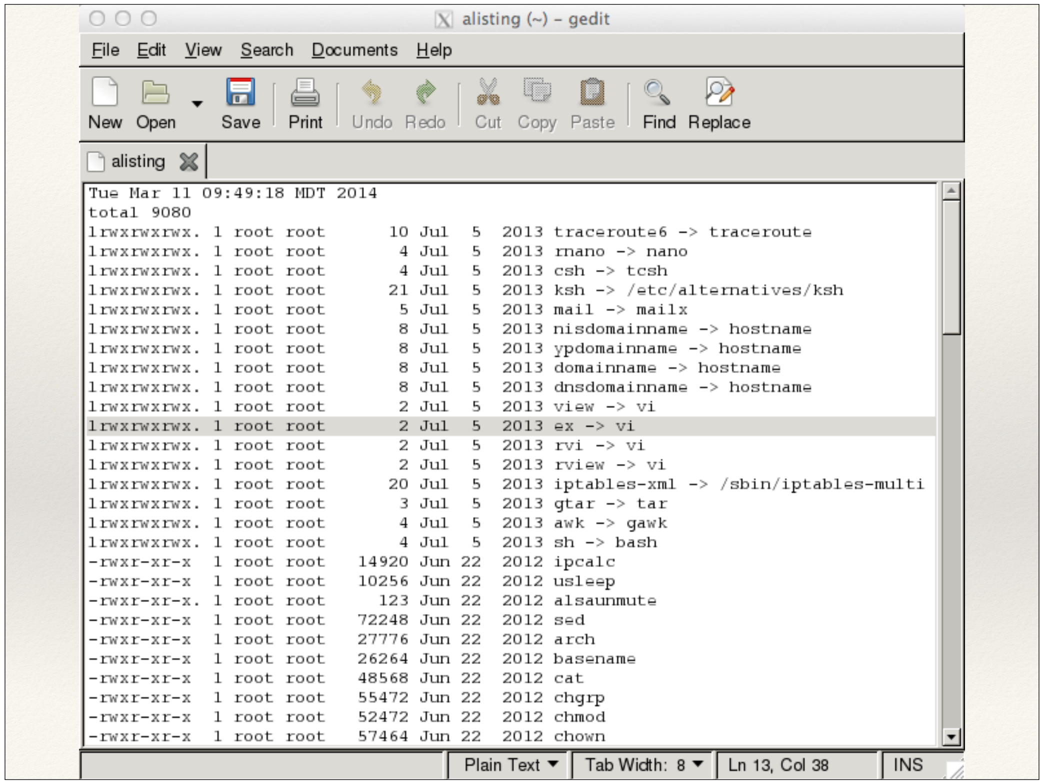
GUI base editors

- These are available on Mio,BlueM, AuN, Mc2
  - gedit
  - gvim
  - emacs
  - Require X-windows but gvim and emacs will fall back to a text version
  - Launch them in the background with the & option and put errors into /dev/null

```
[tkaiser@bluem ~]$ gedit alisting   >& /dev/null &
```

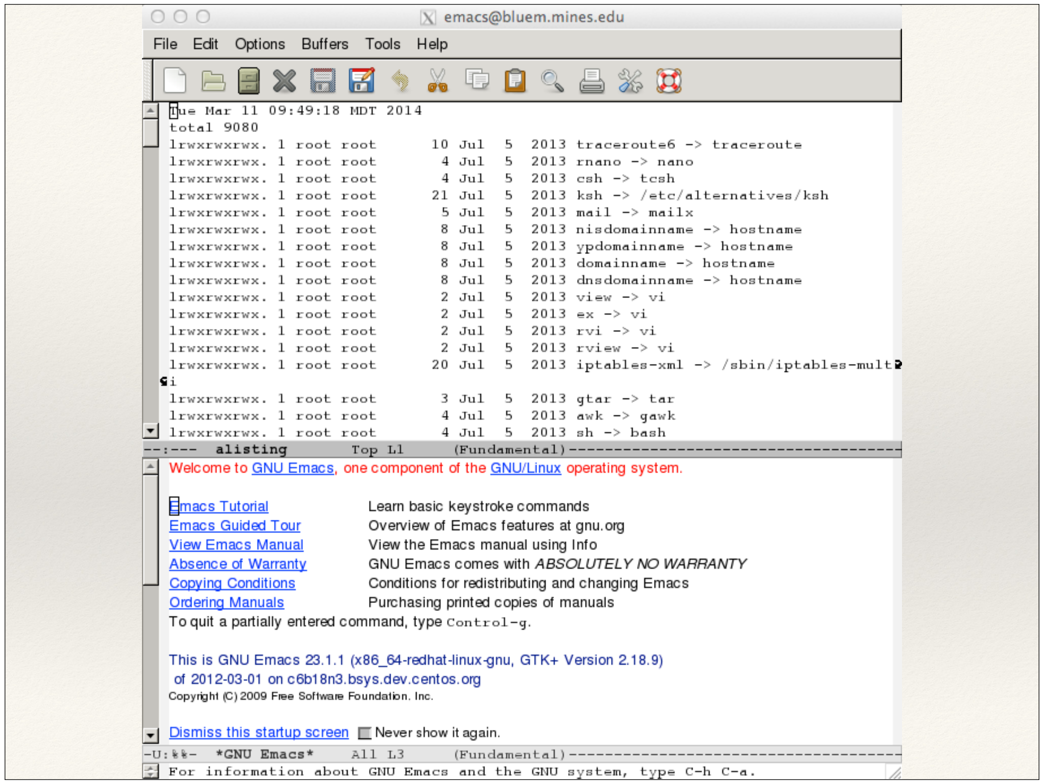
G Edit, g vim, and emacs are available on our local machines. These are GUI based editors and all require your local terminal to support x windows. G vim, and emacs will fall back to the text version if you don't have x windows available.

You will most likely want to launch these in the background. A command is put in the background if the line ends in an ampersand. This will allow you to do other things in the terminal window while the editor is running. These types of editors tend to send unwanted messages to your terminal. This can be prevented by piping the messaged into a special file, dev null, which is like an infinite trash can.


alisting (~) - gedit
File Edit View Search Documents Help
New Open Save Print Undo Redo Cut Copy Paste Find Replace
alisting
Plain Text
Tab Width: 8
Ln 13, Col 38
INS

```
Tue Mar 11 09:49:18 MDT 2014
total 9080
lrwxrwxrwx. 1 root root      10 Jul  5  2013 traceroute6 -> traceroute
lrwxrwxrwx. 1 root root       4 Jul  5  2013 rnano -> nano
lrwxrwxrwx. 1 root root       4 Jul  5  2013 csh -> tcsh
lrwxrwxrwx. 1 root root      21 Jul  5  2013 ksh -> /etc/alternatives/ksh
lrwxrwxrwx. 1 root root       5 Jul  5  2013 mail -> mailx
lrwxrwxrwx. 1 root root       8 Jul  5  2013 nisdomainname -> hostname
lrwxrwxrwx. 1 root root       8 Jul  5  2013 ypdomainname -> hostname
lrwxrwxrwx. 1 root root       8 Jul  5  2013 domainname -> hostname
lrwxrwxrwx. 1 root root       8 Jul  5  2013 dnsdomainname -> hostname
lrwxrwxrwx. 1 root root       2 Jul  5  2013 view -> vi
lrwxrwxrwx. 1 root root       2 Jul  5  2013 ex -> vi
lrwxrwxrwx. 1 root root       2 Jul  5  2013 rvi -> vi
lrwxrwxrwx. 1 root root       2 Jul  5  2013 rview -> vi
lrwxrwxrwx. 1 root root      20 Jul  5  2013 iptables-xml -> /sbin/iptables-multi
lrwxrwxrwx. 1 root root       3 Jul  5  2013 gtar -> tar
lrwxrwxrwx. 1 root root       4 Jul  5  2013 awk -> gawk
lrwxrwxrwx. 1 root root       4 Jul  5  2013 sh -> bash
-rwxr-xr-x  1 root root   14920 Jun 22  2012 ipcalc
-rwxr-xr-x  1 root root   10256 Jun 22  2012 usleep
-rwxr-xr-x. 1 root root     123 Jun 22  2012 alsaunmute
-rwxr-xr-x  1 root root   72248 Jun 22  2012 sed
-rwxr-xr-x  1 root root   27776 Jun 22  2012 arch
-rwxr-xr-x  1 root root   26264 Jun 22  2012 basename
-rwxr-xr-x  1 root root   48568 Jun 22  2012 cat
-rwxr-xr-x  1 root root   55472 Jun 22  2012 chgrp
-rwxr-xr-x  1 root root   52472 Jun 22  2012 chmod
-rwxr-xr-x  1 root root   57464 Jun 22  2012 chown
```

Here is a screen dump of G edit. You should be able to figure it out from here, at least the basics.



```
Tue Mar 11 09:49:18 MDT 2014
total 9080
lrwxrwxrwx. 1 root root      10 Jul  5  2013 traceroute6 -> traceroute
lrwxrwxrwx. 1 root root       4 Jul  5  2013 rnano -> nano
lrwxrwxrwx. 1 root root       4 Jul  5  2013 csh -> tcsh
lrwxrwxrwx. 1 root root      21 Jul  5  2013 ksh -> /etc/alternatives/ksh
lrwxrwxrwx. 1 root root       5 Jul  5  2013 mail -> mailx
lrwxrwxrwx. 1 root root       8 Jul  5  2013 nisdomainname -> hostname
lrwxrwxrwx. 1 root root       8 Jul  5  2013 ypdomainname -> hostname
lrwxrwxrwx. 1 root root       8 Jul  5  2013 domainname -> hostname
lrwxrwxrwx. 1 root root       8 Jul  5  2013 dnsdomainname -> hostname
lrwxrwxrwx. 1 root root       2 Jul  5  2013 view -> vi
lrwxrwxrwx. 1 root root       2 Jul  5  2013 ex -> vi
lrwxrwxrwx. 1 root root       2 Jul  5  2013 rvi -> vi
lrwxrwxrwx. 1 root root       2 Jul  5  2013 rview -> vi
lrwxrwxrwx. 1 root root      20 Jul  5  2013 iptables-xml -> /sbin/iptables-mult
i
lrwxrwxrwx. 1 root root       3 Jul  5  2013 gtar -> tar
lrwxrwxrwx. 1 root root       4 Jul  5  2013 awk -> gawk
lrwxrwxrwx. 1 root root       4 Jul  5  2013 sh -> bash
--:---  alisting       Top L1     (Fundamental)-----------------------------------
```

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

Emacs Tutorial	Learn basic keystroke commands
Emacs Guided Tour	Overview of Emacs features at gnu.org
View Emacs Manual	View the Emacs manual using Info
Absence of Warranty	GNU Emacs comes with *ABSOLUTELY NO WARRANTY*
Copying Conditions	Conditions for redistributing and changing Emacs
Ordering Manuals	Purchasing printed copies of manuals

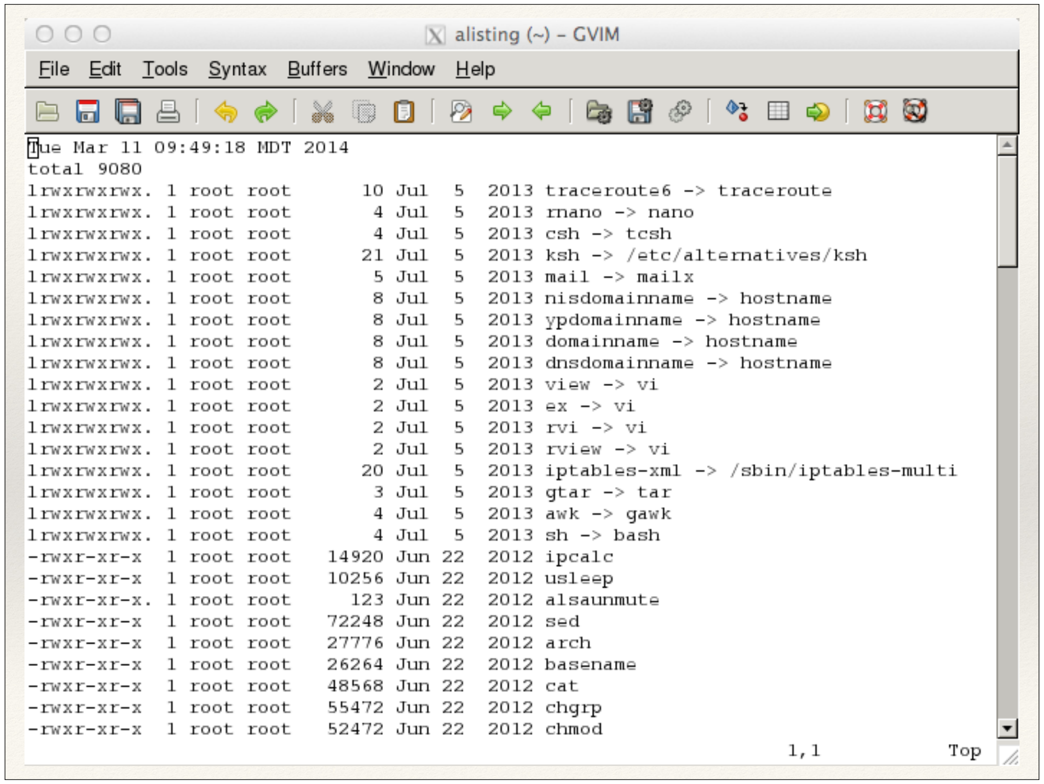
To quit a partially entered command, type `Control-g`.

This is GNU Emacs 23.1.1 (x86_64-redhat-linux-gnu, GTK+ Version 2.18.9)
of 2012-03-01 on c6b18n3.bsys.dev.centos.org
Copyright (C) 2009 Free Software Foundation, Inc.

Dismiss this startup screen ▢ Never show it again.

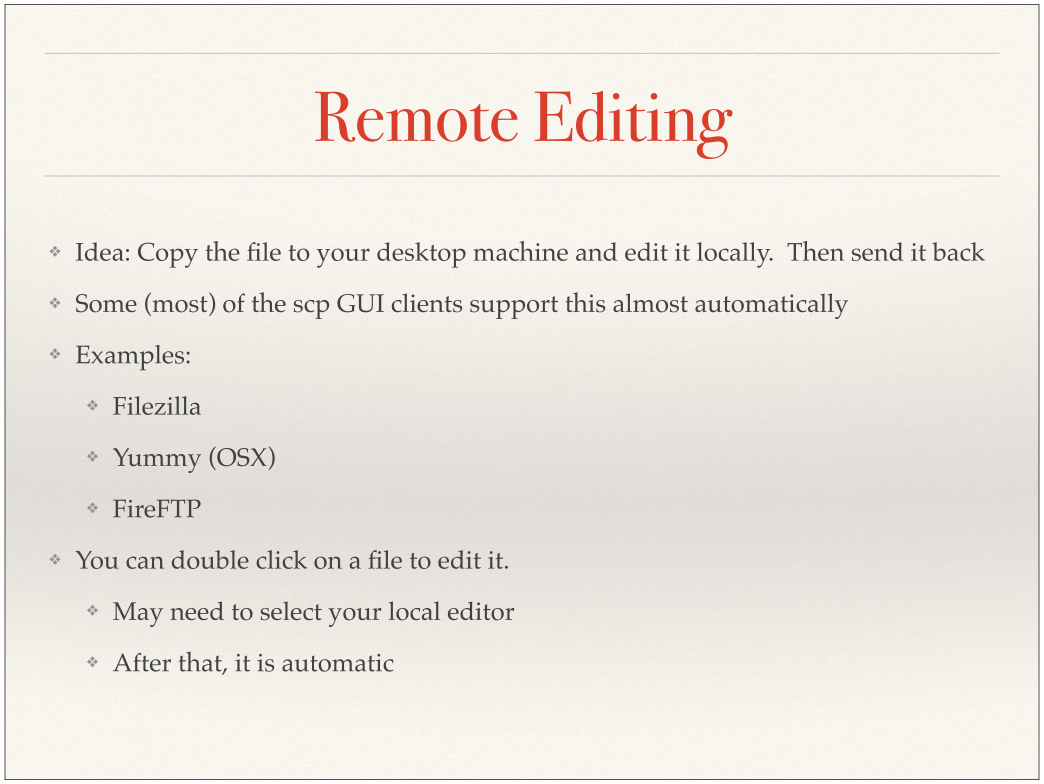
```
-U:%%-  *GNU Emacs*    All L3     (Fundamental)-----------------------------------
For information about GNU Emacs and the GNU system, type C-h C-a.
```

Here is a emacs GUI session.



```
Tue Mar 11 09:49:18 MDT 2014
total 9080
lrwxrwxrwx. 1 root root      10 Jul  5  2013 traceroute6 -> traceroute
lrwxrwxrwx. 1 root root       4 Jul  5  2013 rnano -> nano
lrwxrwxrwx. 1 root root       4 Jul  5  2013 csh -> tcsh
lrwxrwxrwx. 1 root root      21 Jul  5  2013 ksh -> /etc/alternatives/ksh
lrwxrwxrwx. 1 root root       5 Jul  5  2013 mail -> mailx
lrwxrwxrwx. 1 root root       8 Jul  5  2013 nisdomainname -> hostname
lrwxrwxrwx. 1 root root       8 Jul  5  2013 ypdomainname -> hostname
lrwxrwxrwx. 1 root root       8 Jul  5  2013 domainname -> hostname
lrwxrwxrwx. 1 root root       8 Jul  5  2013 dnsdomainname -> hostname
lrwxrwxrwx. 1 root root       2 Jul  5  2013 view -> vi
lrwxrwxrwx. 1 root root       2 Jul  5  2013 ex -> vi
lrwxrwxrwx. 1 root root       2 Jul  5  2013 rvi -> vi
lrwxrwxrwx. 1 root root       2 Jul  5  2013 rview -> vi
lrwxrwxrwx. 1 root root      20 Jul  5  2013 iptables-xml -> /sbin/iptables-multi
lrwxrwxrwx. 1 root root       3 Jul  5  2013 gtar -> tar
lrwxrwxrwx. 1 root root       4 Jul  5  2013 awk -> gawk
lrwxrwxrwx. 1 root root       4 Jul  5  2013 sh -> bash
-rwxr-xr-x  1 root root   14920 Jun 22  2012 ipcalc
-rwxr-xr-x  1 root root   10256 Jun 22  2012 usleep
-rwxr-xr-x. 1 root root     123 Jun 22  2012 alsaunmute
-rwxr-xr-x  1 root root   72248 Jun 22  2012 sed
-rwxr-xr-x  1 root root   27776 Jun 22  2012 arch
-rwxr-xr-x  1 root root   26264 Jun 22  2012 basename
-rwxr-xr-x  1 root root   48568 Jun 22  2012 cat
-rwxr-xr-x  1 root root   55472 Jun 22  2012 chgrp
-rwxr-xr-x  1 root root   52472 Jun 22  2012 chmod
                                                               1,1           Top
```

Finally we have a G vim GUI window.



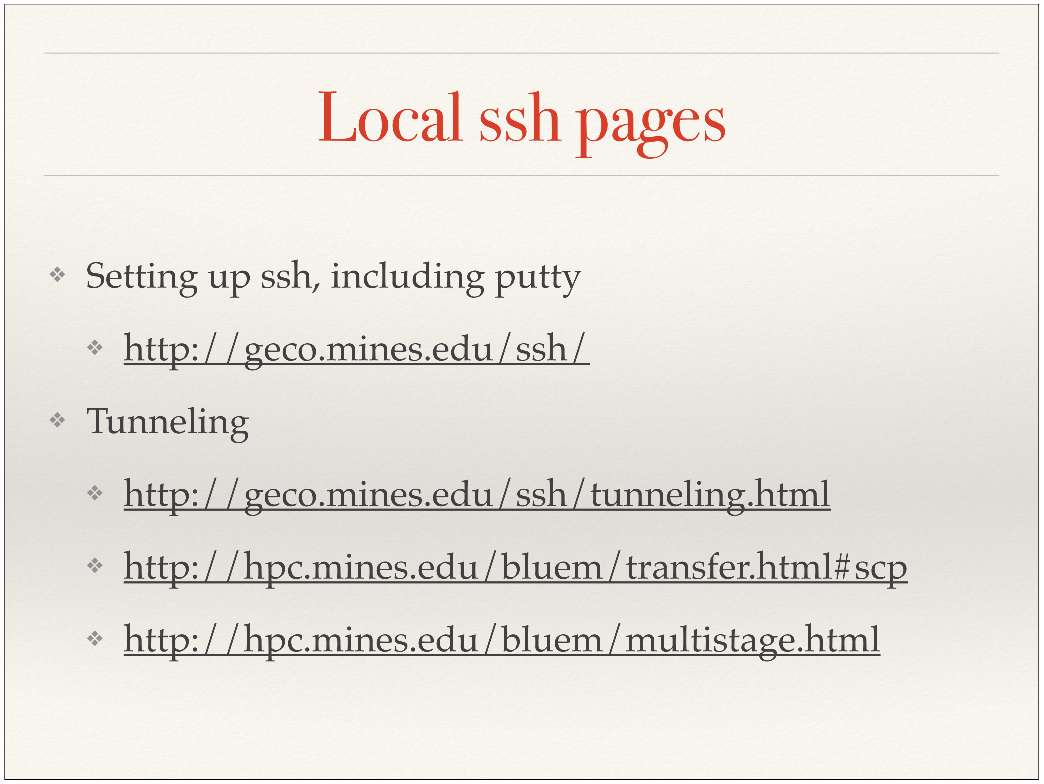
Remote Editing

- Idea: Copy the file to your desktop machine and edit it locally. Then send it back
- Some (most) of the scp GUI clients support this almost automatically
- Examples:
  - Filezilla
  - Yummy (OSX)
  - FireFTP
- You can double click on a file to edit it.
  - May need to select your local editor
  - After that, it is automatic

If you have an editor that you like to use on your local machine you can. In fact many of the SCP GUI clients can act as a communications channel between your local editor and remote machines. What you do is open a remote file browser. Then select a file. There will be an option to open the file in you editor. Then when the file is saved from inside your editor it is automatically saved on the remote machine. In reality what is happening is there is a local copy of the file created. You edit that file and then it is automatically copied back. Very cool

Back to ssh

We want to go back to talk a bit more about SSH. In particular, we want to talk about some features that can make your life much easier.

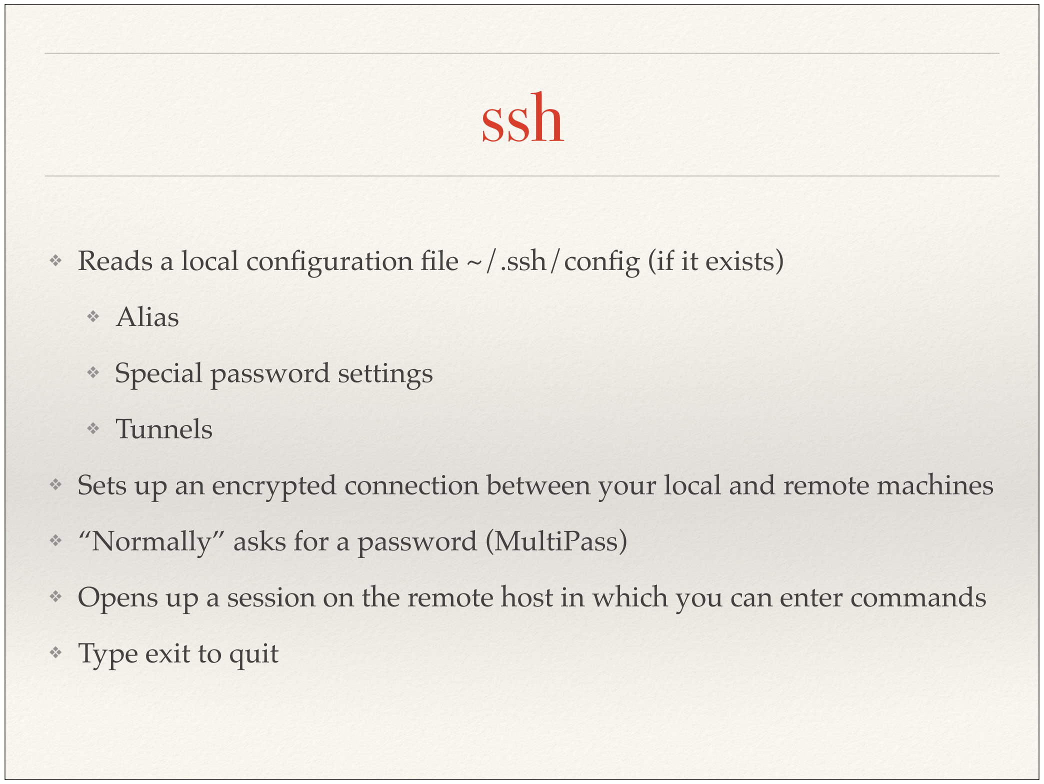


Local ssh pages

- Setting up ssh, including putty
  - http://geco.mines.edu/ssh/
- Tunneling
  - http://geco.mines.edu/ssh/tunneling.html
  - http://hpc.mines.edu/bluem/transfer.html#scp
  - http://hpc.mines.edu/bluem/multistage.html

We have a number of local web pages that describe some of the features. We have a detailed description on how to set the the Windows SSH client putty.

We will be discussing ssh tunneling shortly.



ssh

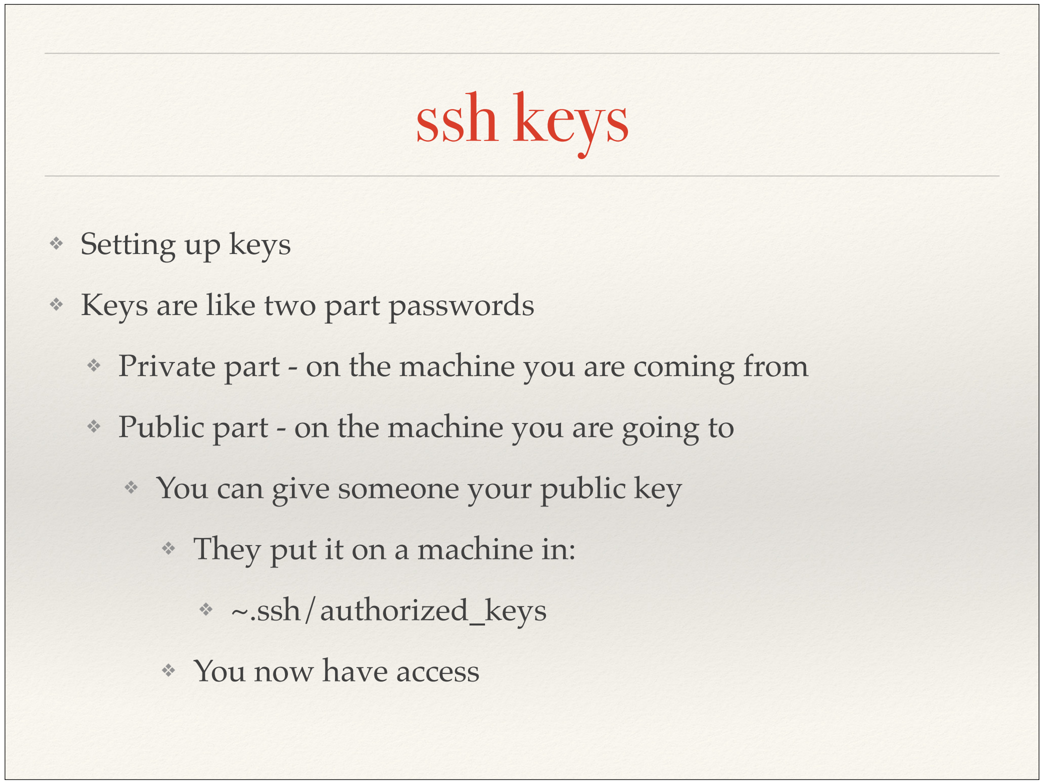
- Reads a local configuration file ~/.ssh/config (if it exists)
    - Alias
    - Special password settings
    - Tunnels
- Sets up an encrypted connection between your local and remote machines
- "Normally" asks for a password (MultiPass)
- Opens up a session on the remote host in which you can enter commands
- Type exit to quit

ssh is the normal command for logging into a linux box from another linux box. Even the GUI programs that connect to a linux box normally use ssh under the wraps. By the way ssh stands fore secure shell. All of what gets passed to a remote box is encrypted, including your password.

Before a connecting is made a configuration file is read off of your home directory. The configuration file is in a hidden directory dot ssh . There are a lot of options that you can set in the file to make your life easier. We will discuss these later.

If there are no special settings in your configuration file then when you start ssh you will be asked for you pass word for the remote machine. Fro the Mines HPC platforms this is your Multi-Pass password.

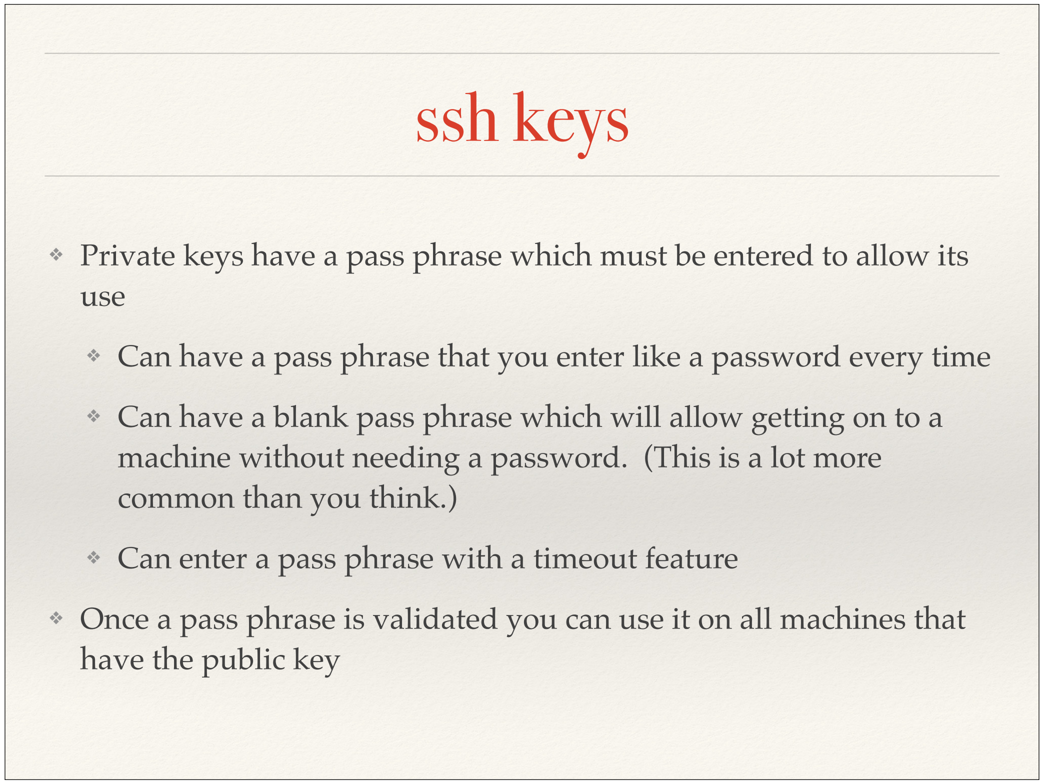
After the remote session starts you can enter commands. To quit type exit.



ssh keys

- Setting up keys
- Keys are like two part passwords
  - Private part - on the machine you are coming from
  - Public part - on the machine you are going to
    - You can give someone your public key
      - They put it on a machine in:
        - ~.ssh/authorized_keys
      - You now have access

We have talked about SSH being the program that you use to communication between machines. There are many features of SSH that can make your life easier. Keys are one of the features of SSH. Keys are like two part passwords. One part is on the machine you are going to and the other is on your local machine. The local part is called the private key. The remote part is called the public key. The public keys are copied to the SSH authorized keys file on the remote machine. Then when you run SSH you can tell it to match your public and private keys. If there is a match then you are allowed access without entering a password.

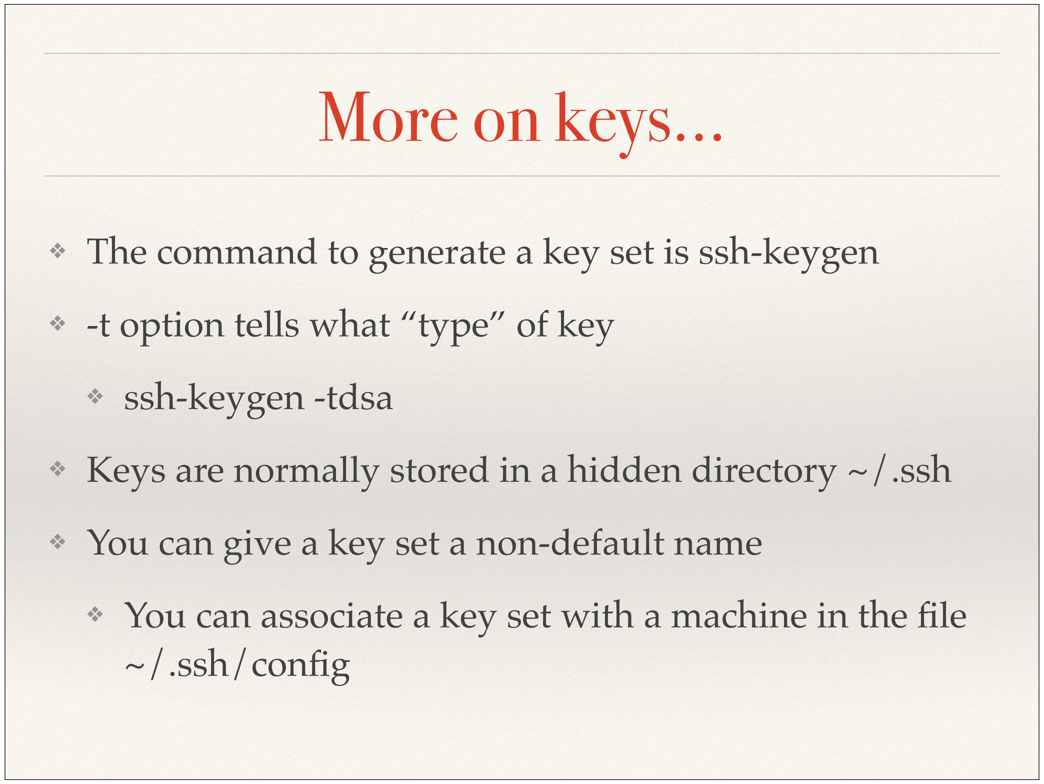


ssh keys

- Private keys have a pass phrase which must be entered to allow its use
  - Can have a pass phrase that you enter like a password every time
  - Can have a blank pass phrase which will allow getting on to a machine without needing a password. (This is a lot more common than you think.)
  - Can enter a pass phrase with a timeout feature
- Once a pass phrase is validated you can use it on all machines that have the public key

Private keys have a pass phrase that must be entered when they are used. So what is the advantage over using a regular password? You can have a blank pass phrase which will allow getting on to a machine without needing a password. One place this is used is on high performance computing clusters. This allows you to freely move between nodes without entering passwords. Some parallel computing software relies on this capability.

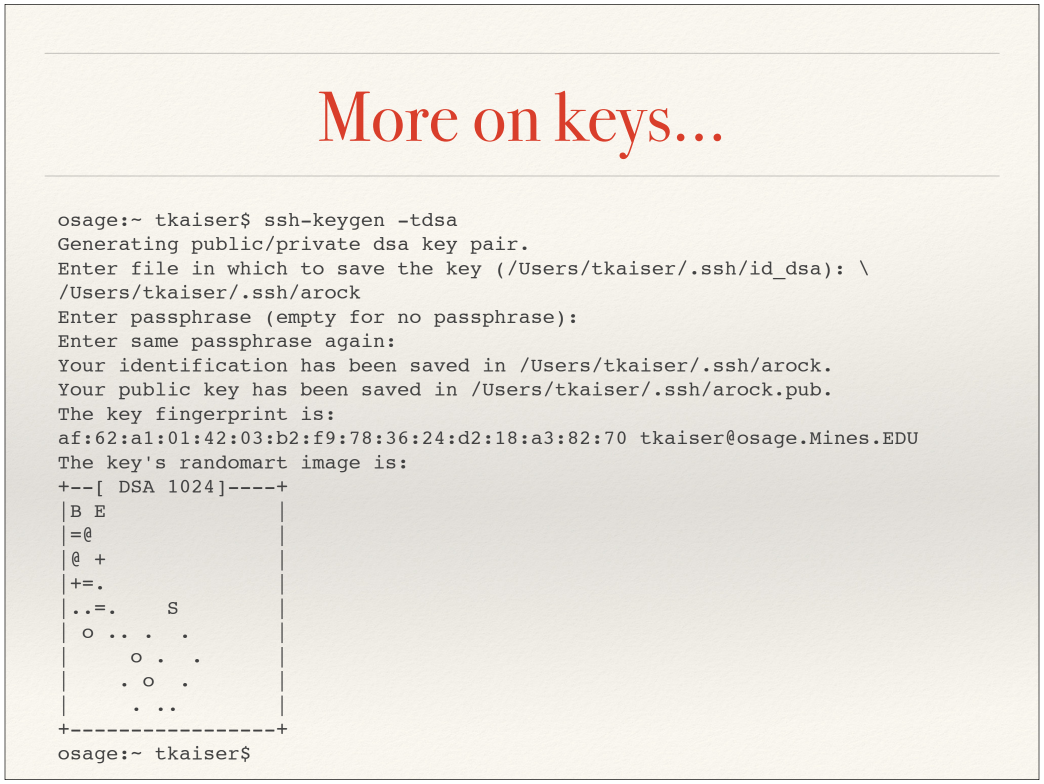
One very cool feature is that you can assign a timeout for a pass phrase. That is, you run a command that takes your pass phrase but you also give it a exploration time. So for some period of time, say 8 hours, you can log into machines without giving a password of pass phrase.



More on keys…

- The command to generate a key set is ssh-keygen
- -t option tells what “type” of key
  - ssh-keygen -tdsa
- Keys are normally stored in a hidden directory ~/.ssh
- You can give a key set a non-default name
  - You can associate a key set with a machine in the file ~/.ssh/config

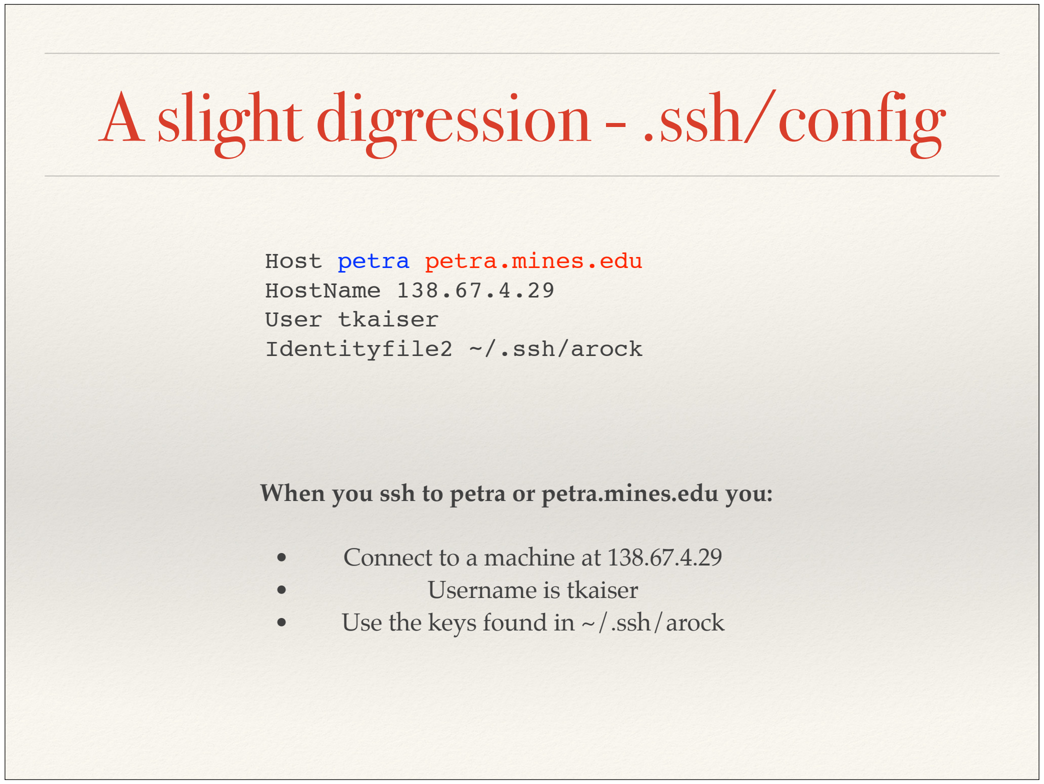
You can use the command ssh key gen to create key pairs, that is, a matching private and public key set. There are various types of keys that can be created corresponding to different protocols within ssh. We typically specify minus t dsa. Keys are normally stored in a hidden directory off of your home directory .ssh. Finally you can give keys a non default name. This is often done to create keys that will be used on some particular machine or a set of remote machines.



More on keys…

```
osage:~ tkaiser$ ssh-keygen -tdsa
Generating public/private dsa key pair.
Enter file in which to save the key (/Users/tkaiser/.ssh/id_dsa): \
/Users/tkaiser/.ssh/arock
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/tkaiser/.ssh/arock.
Your public key has been saved in /Users/tkaiser/.ssh/arock.pub.
The key fingerprint is:
af:62:a1:01:42:03:b2:f9:78:36:24:d2:18:a3:82:70 tkaiser@osage.Mines.EDU
The key's randomart image is:
+--[ DSA 1024]----+
|B E              |
|=@               |
|@ +              |
|+=.              |
|..=.    S        |
| o .. .  .       |
|     o .  .      |
|    . o  .       |
|     . ..        |
+-----------------+
osage:~ tkaiser$
```

Here is a screen dump of creating a new key set. We will give this key set a non default name of — a rock —. We created two files in our ssh directory — a rock — and — a rock — pub. Some versions of ssh key gen produce a image associated with the key as shown here. I have never actually seen a use for this.



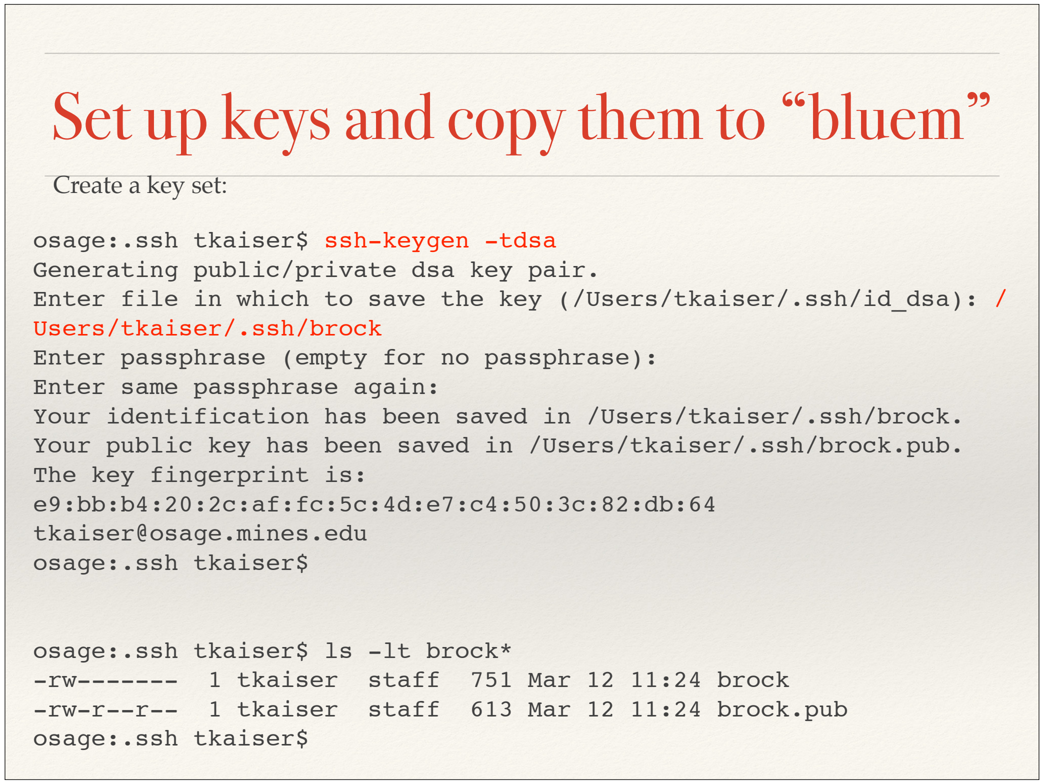
A slight digression - .ssh/config

```
Host petra petra.mines.edu
HostName 138.67.4.29
User tkaiser
Identityfile2 ~/.ssh/arock
```

When you ssh to petra or petra.mines.edu you:

- Connect to a machine at 138.67.4.29
- Username is tkaiser
- Use the keys found in ~/.ssh/arock

Inside your ssh directory you can have a file called config. You can use this file to create mappings between keys and machines. Here is how you could use a key with a non default name, in this case — a rock —. This key will be used to access a machine called petra. After this is set up you can ssh to petra or petra dot mines dot edu using the key — a rock —.

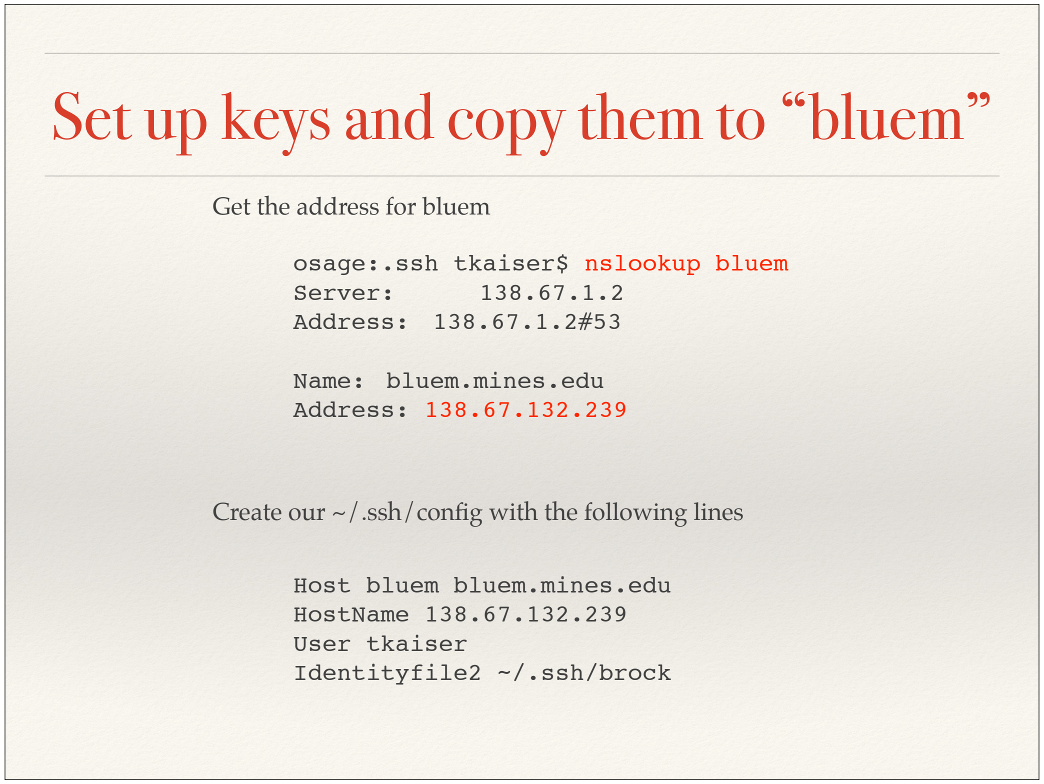


Set up keys and copy them to "bluem"

Create a key set:

```
osage:.ssh tkaiser$ ssh-keygen -tdsa
Generating public/private dsa key pair.
Enter file in which to save the key (/Users/tkaiser/.ssh/id_dsa): /
Users/tkaiser/.ssh/brock
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/tkaiser/.ssh/brock.
Your public key has been saved in /Users/tkaiser/.ssh/brock.pub.
The key fingerprint is:
e9:bb:b4:20:2c:af:fc:5c:4d:e7:c4:50:3c:82:db:64
tkaiser@osage.mines.edu
osage:.ssh tkaiser$


osage:.ssh tkaiser$ ls -lt brock*
-rw-------  1 tkaiser  staff  751 Mar 12 11:24 brock
-rw-r--r--  1 tkaiser  staff  613 Mar 12 11:24 brock.pub
osage:.ssh tkaiser$
```

Here is a screen dump of creating a new key set for bluem. We will give this key set a non default name of — b rock —. This created two files in our ssh directory — b rock — and — b rock — dot pub.



Set up keys and copy them to "bluem"

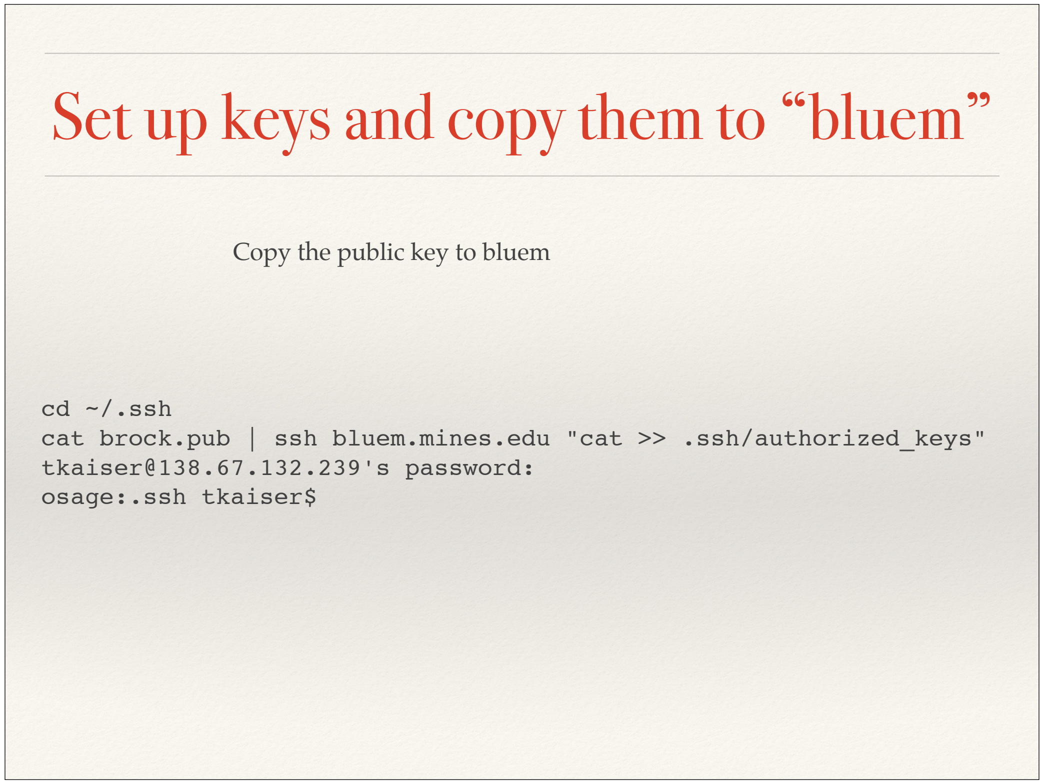
Get the address for bluem

```
osage:.ssh tkaiser$ nslookup bluem
Server:      138.67.1.2
Address:  138.67.1.2#53

Name:  bluem.mines.edu
Address: 138.67.132.239
```

Create our ~/.ssh/config with the following lines

```
Host bluem bluem.mines.edu
HostName 138.67.132.239
User tkaiser
Identityfile2 ~/.ssh/brock
```

Here we show the same basic set up in the config file for our machine blue M. There is however one other step. You must get the public key installed on blue m.



Set up keys and copy them to “bluem”

Copy the public key to bluem

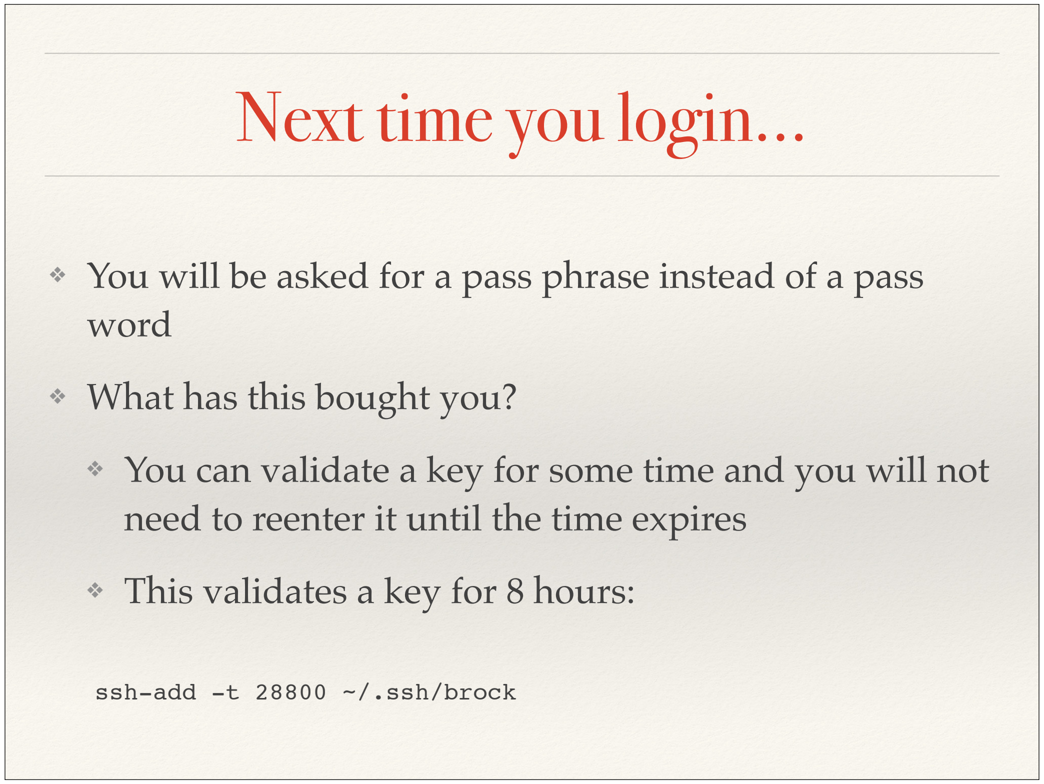
```
cd ~/.ssh
cat brock.pub | ssh bluem.mines.edu "cat >> .ssh/authorized_keys"
tkaiser@138.67.132.239's password:
osage:.ssh tkaiser$
```

As said, the public or pub portion of the key set needs to be placed on the remote machine. In particular a copy of the key needs to go into the file authorized_keys on the remote machine. Here is an easy way to do that. Go to your ssh directory and run the command shown here in red. Note the name of the remote machine, blue m dot mines dot edu and the file containing the public portion of the key “b rock dot pub”.

Recall that the cat command types a file to the terminal and that the vertical bar is pipe. So we have the output for the cat command being set or piped the the ssh command. We are doing an ssh to our remote machine. Also recall that we can give ssh a command to run on the remote machine instead of logging in. Here we are telling ssh to run the cat command on our remote machine and send the output to our authorized_keys file. But what is it putting there? Remember that we are also sending ssh the contents of our key file by using combination of the cat command the the vertical bar or pipe option.

So here is what is happening.

The ssh command is doing three things. It is collecting the contents of the key file via the first cat command and the pipe. It is connecting to our remote machine. Finally, it is putting the contents of the key file into the authorized_keys file.

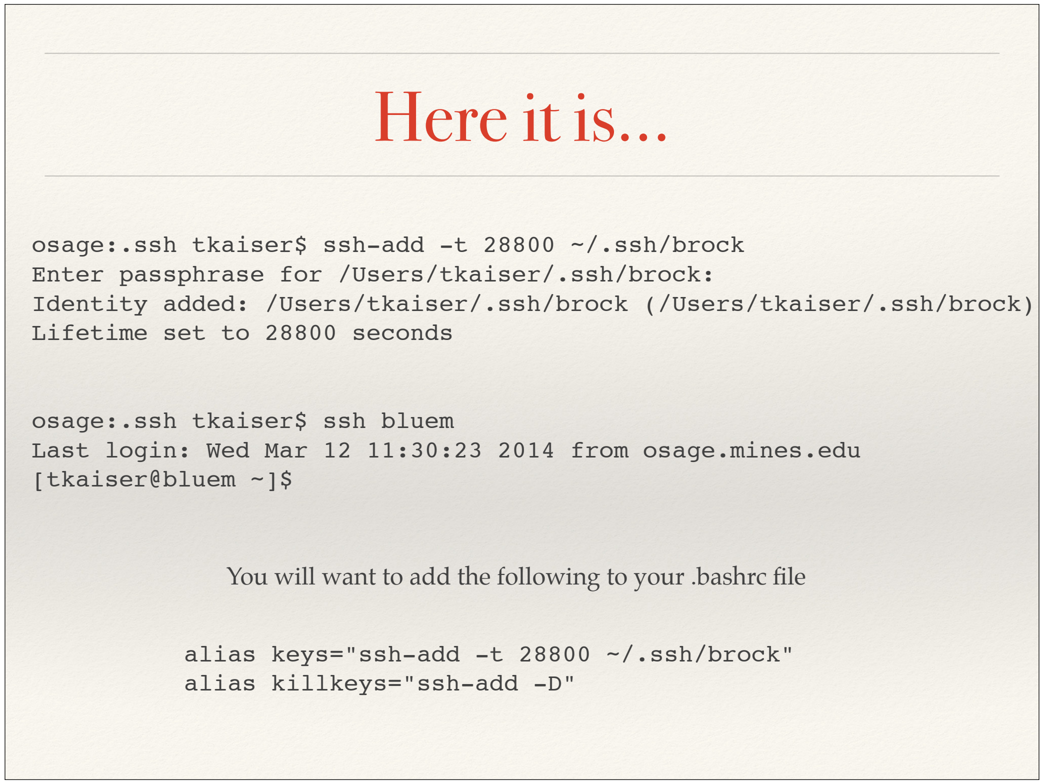


Next time you login…

- You will be asked for a pass phrase instead of a pass word
- What has this bought you?
  - You can validate a key for some time and you will not need to reenter it until the time expires
  - This validates a key for 8 hours:

```
ssh-add -t 28800 ~/.ssh/brock
```

The next time you login to a remote machine you will be asked for a pass phrase instead of a pass word. What has this bought you? It turns out that you can validate a key for some time and you will not need to reenter it until the time expires. The command of interest is ssh dash add. This command takes a time in seconds and the key you want to validate. Here we are validating our — b rock — key for blue m for 28800 seconds or for 8 hours.



Here it is…

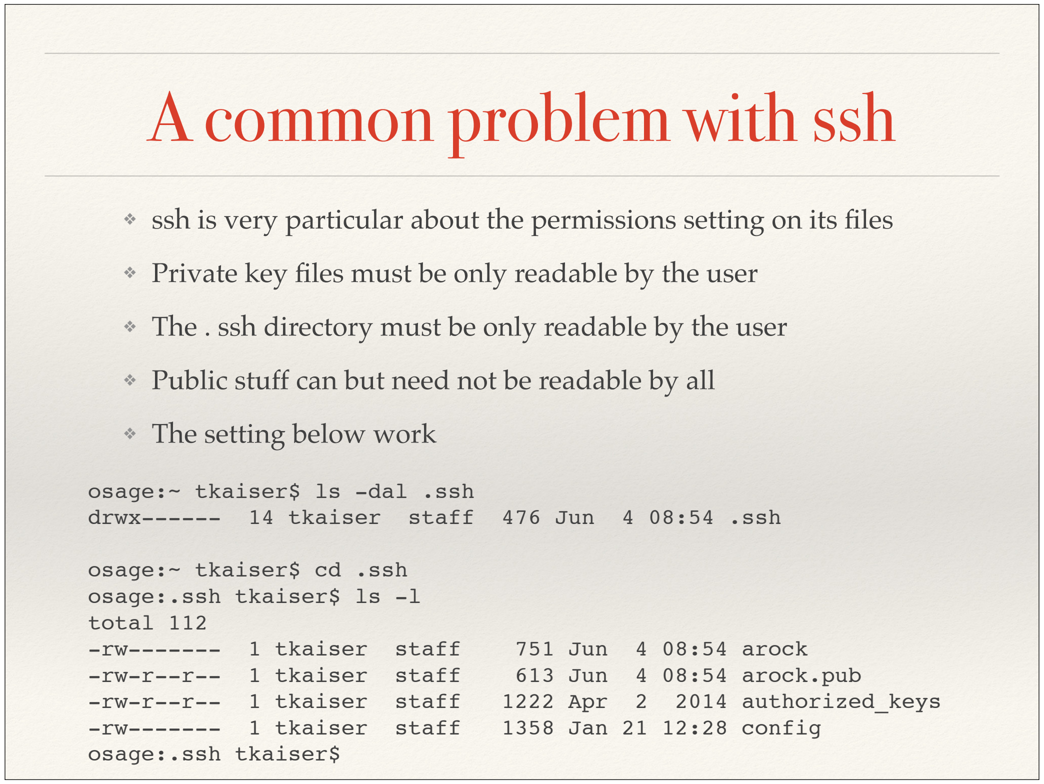
```
osage:.ssh tkaiser$ ssh-add -t 28800 ~/.ssh/brock
Enter passphrase for /Users/tkaiser/.ssh/brock:
Identity added: /Users/tkaiser/.ssh/brock (/Users/tkaiser/.ssh/brock)
Lifetime set to 28800 seconds


osage:.ssh tkaiser$ ssh bluem
Last login: Wed Mar 12 11:30:23 2014 from osage.mines.edu
[tkaiser@bluem ~]$
```

You will want to add the following to your .bashrc file

```
alias keys="ssh-add -t 28800 ~/.ssh/brock"
alias killkeys="ssh-add -D"
```

Here it is in action. We validate the key for 8 hours. We are then free to login to the remote machine without entering a password.

Since this command is rather cumbersome you might want to add an alias for it to your dot bash r c file. Then you will just need to enter the keys command instead of the ssh add command. We also have a command here to unauthorize the keys. By the way, you can have multiple keys authorized with a single ssh add command. If they have the same pass phrase you will only need to enter it once.

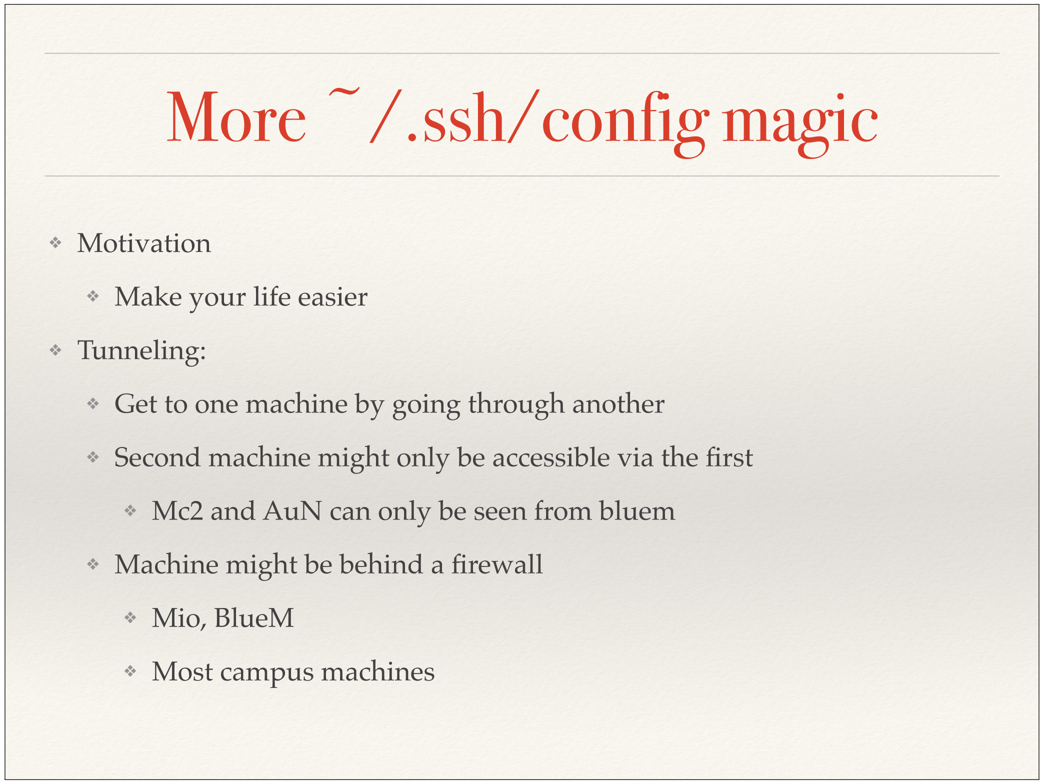


A common problem with ssh

- ssh is very particular about the permissions setting on its files
- Private key files must be only readable by the user
- The . ssh directory must be only readable by the user
- Public stuff can but need not be readable by all
- The setting below work

```
osage:~ tkaiser$ ls -dal .ssh
drwx------   14 tkaiser  staff  476 Jun  4 08:54 .ssh

osage:~ tkaiser$ cd .ssh
osage:.ssh tkaiser$ ls -l
total 112
-rw-------  1 tkaiser  staff   751 Jun  4 08:54 arock
-rw-r--r--  1 tkaiser  staff   613 Jun  4 08:54 arock.pub
-rw-r--r--  1 tkaiser  staff  1222 Apr  2  2014 authorized_keys
-rw-------  1 tkaiser  staff  1358 Jan 21 12:28 config
osage:.ssh tkaiser$
```

When ssh does not work a common problem is that file permissions are not set properly. Ssh is very particular about the permissions setting on its files. Private key files must be only readable by the user and the dot ssh directory must be only readable by the user. Public keys can but need not be readable by all. You can use the chmod command to set file permission as discussed earlier in this presentation. In particular, if permissions get messed up you would want to do a chmod 700 dot ssh in your home directory and then you could do a chmod 600 on all files inside your dot ssh directory.

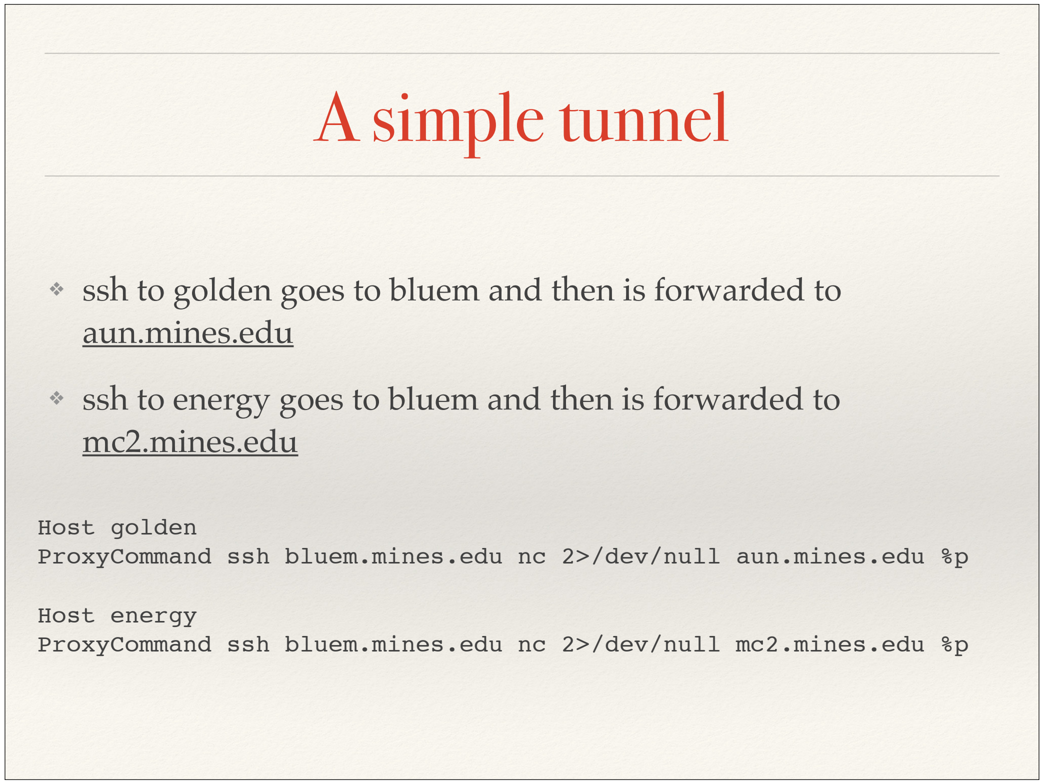


More ~/.ssh/config magic

- Motivation
  - Make your life easier
- Tunneling:
  - Get to one machine by going through another
  - Second machine might only be accessible via the first
    - Mc2 and AuN can only be seen from bluem
  - Machine might be behind a firewall
    - Mio, BlueM
    - Most campus machines

There are other things you can do to make your life easier. Tunnels allow you to log in to a machine and then automatically then be forwarded to a second machine. That is, you get to one machine by going through another. This can be useful if for some reason you don't have direct access to the second machine and you need to go through the first to get to the second.   Our machines Mc2 and A U N can only be seen from blue m. Most machines on campus are behind a firewall so to get to them from off campus you can tunnel through the machine imagine.

Tunnels are set up in your dot ssh config file.

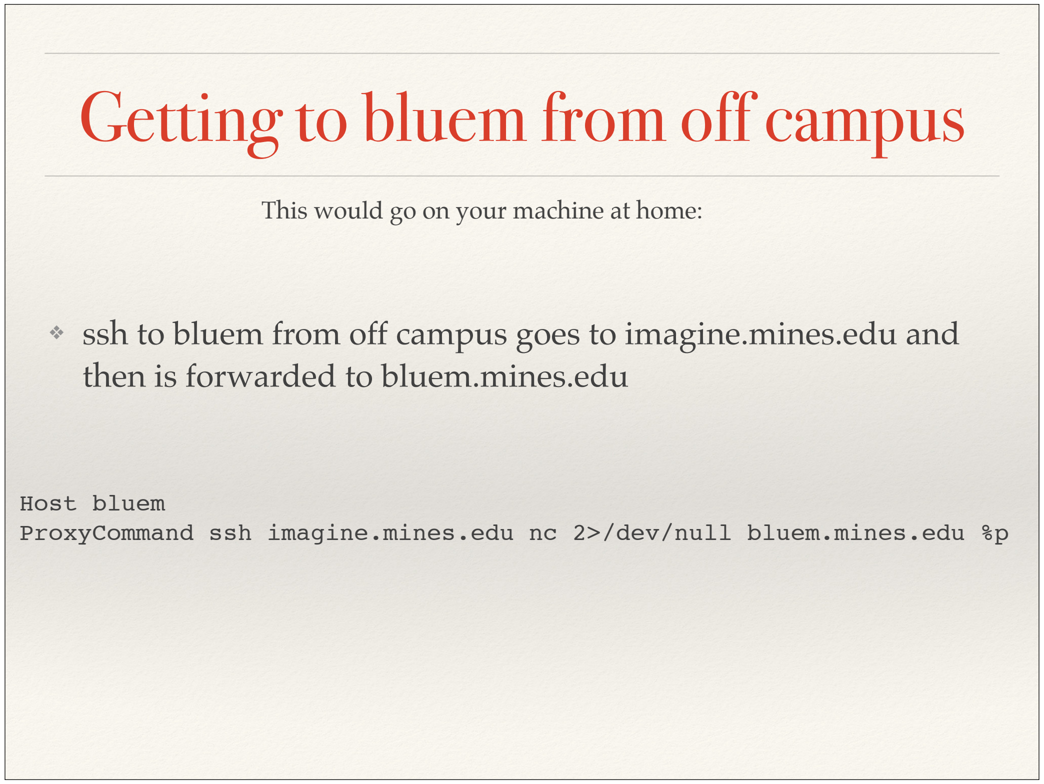


A simple tunnel

- ssh to golden goes to bluem and then is forwarded to aun.mines.edu
- ssh to energy goes to bluem and then is forwarded to mc2.mines.edu

```
Host golden
ProxyCommand ssh bluem.mines.edu nc 2>/dev/null aun.mines.edu %p

Host energy
ProxyCommand ssh bluem.mines.edu nc 2>/dev/null mc2.mines.edu %p
```

On of the neat things about tunnels is that you can rename hosts while doing the connection. Here we have tunnels set up for two of our H P C platform that are accessible only through blue m. With the four lines placed in your config file you can do an ssh to “golden”, that is A U N through blue m and to “energy”, that is M C 2 also going through blue m. From your local machine it looks like you have a direct connection to the two machines hidden behind blue m. The nice thing about this is that it saves you a step.



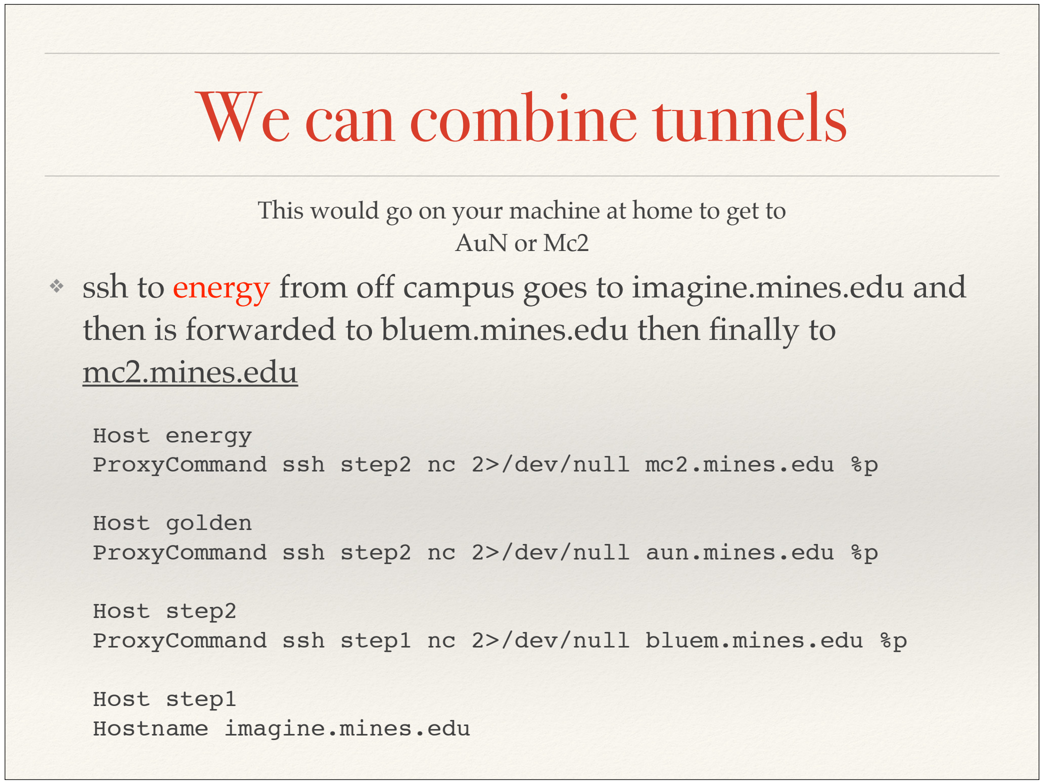
Getting to bluem from off campus

This would go on your machine at home:

- ssh to bluem from off campus goes to imagine.mines.edu and then is forwarded to bluem.mines.edu

```
Host bluem
ProxyCommand ssh imagine.mines.edu nc 2>/dev/null bluem.mines.edu %p
```

Blue m is actually behind the campus firewall so a direct connection to blue m from off campus is not possible. However, if you have a linux or OS x box off campus you can set up a tunnel to get to blue m. The syntax is the same. In this case you would need to put these two lines in your config file on your home machine.



We can combine tunnels

This would go on your machine at home to get to
AuN or Mc2

- ssh to energy from off campus goes to imagine.mines.edu and then is forwarded to bluem.mines.edu then finally to mc2.mines.edu

```
Host energy
ProxyCommand ssh step2 nc 2>/dev/null mc2.mines.edu %p

Host golden
ProxyCommand ssh step2 nc 2>/dev/null aun.mines.edu %p

Host step2
ProxyCommand ssh step1 nc 2>/dev/null bluem.mines.edu %p

Host step1
Hostname imagine.mines.edu
```

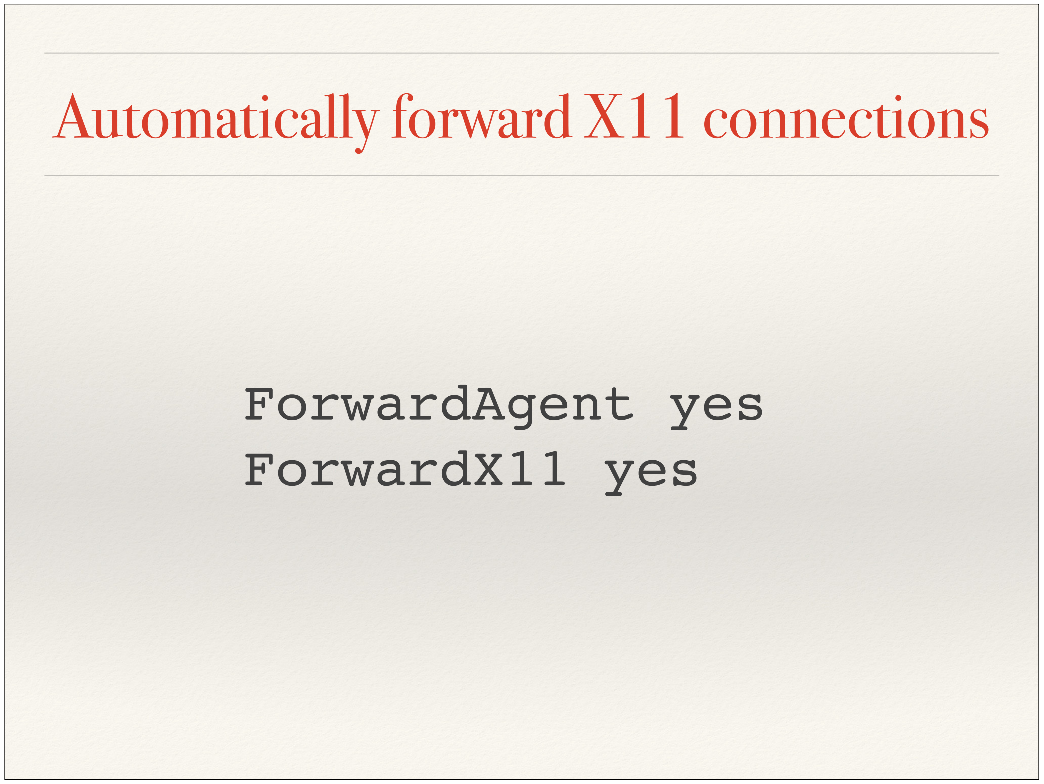
You can combine tunnels. If you put the lines shown here in the config file on your home machine it would allow you to login to energy, that is M C 2, and golden, that is A U N, as if you had a direct connection.

To see what is going on here look at the -- Host -- lines and Proxy command lines that follow.

If you do an ssh to energy you will end up on M c 2 dot mines dot edu. The proxy command line is sort of like an instruction on how this will happen. We see near the end of the
proxy command line the goal which is M c 2 dot mines dot edu. The rest of the line is an explanation on how to get there. In particular, to get to M c 2 dot mines dot edu we first do an ssh to step 2.

Now if we look at the host and proxy command for step 2 we see our goal is to go to blue m dot mines dot edu. But again we are told that to get to blue m dot mines dot edu we need to do an ssh to step 1.

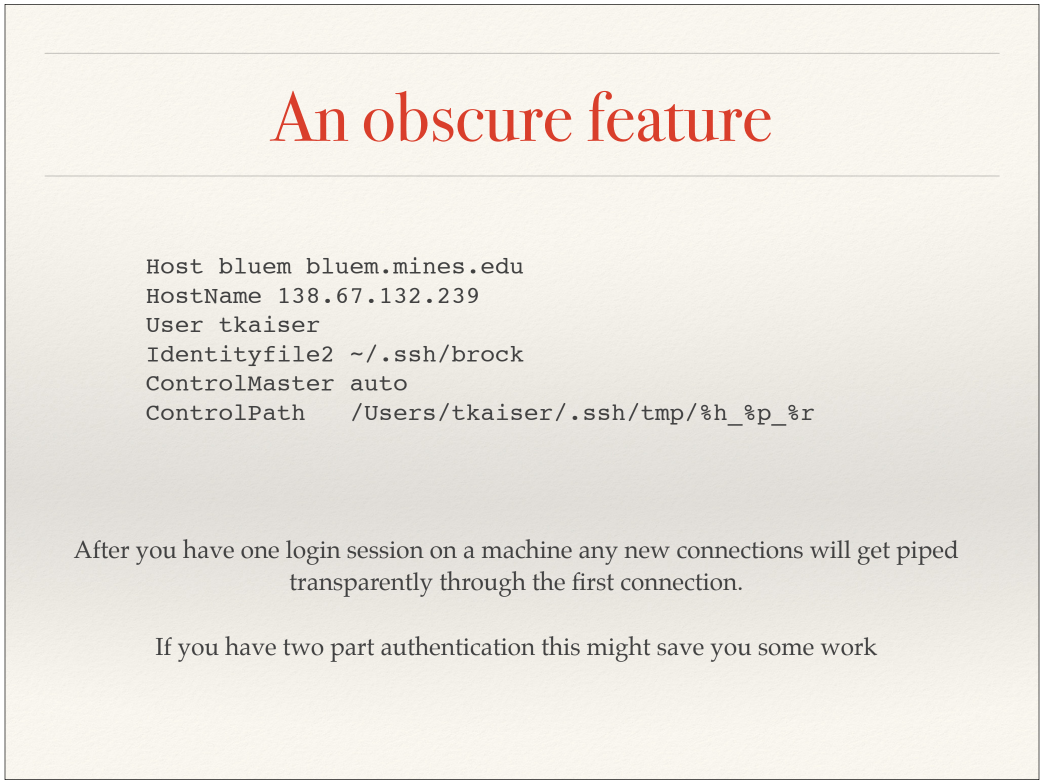
If we look at the host line for step 1 we see that this maps directly to imagine dot mines dot edu.



Automatically forward X11 connections

```
ForwardAgent yes
ForwardX11 yes
```

Here is another thing you might want to add to your config file if you are running any x 11 programs. These lines will forward x 11 sessions between machines.



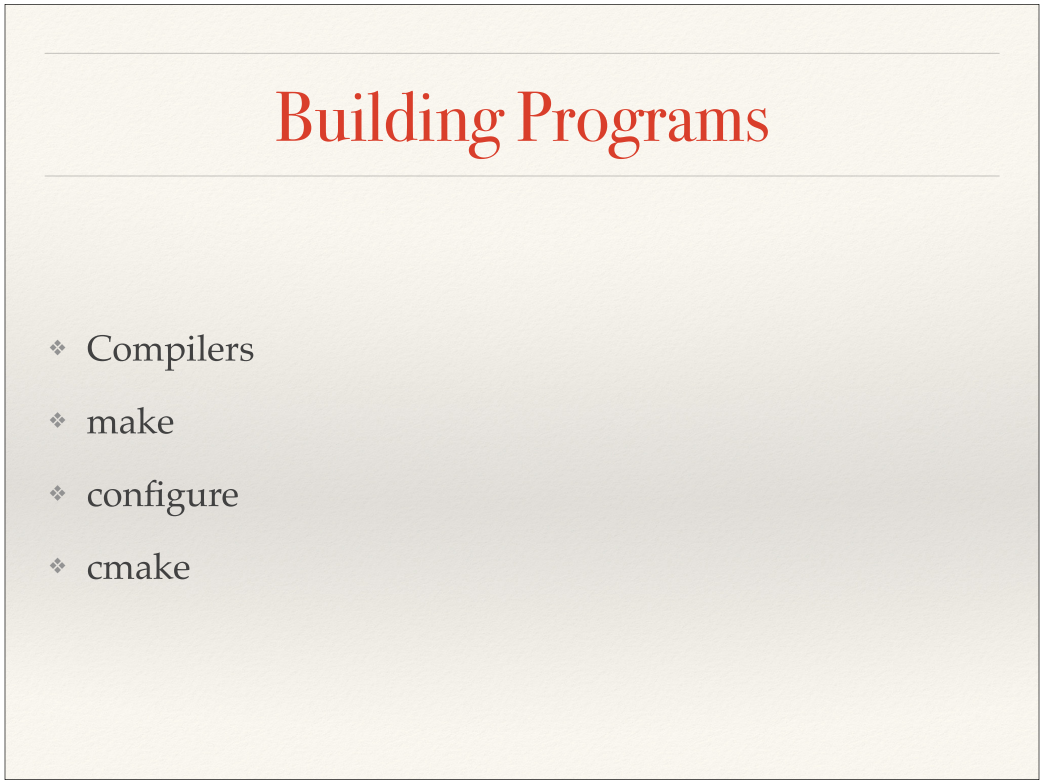
An obscure feature

```
Host bluem bluem.mines.edu
HostName 138.67.132.239
User tkaiser
Identityfile2 ~/.ssh/brock
ControlMaster auto
ControlPath   /Users/tkaiser/.ssh/tmp/%h_%p_%r
```

After you have one login session on a machine any new connections will get piped transparently through the first connection.

If you have two part authentication this might save you some work

This is an obscure feature that might be useful if you need to do a cumbersome two part authentication to get to a machine. A two part authentication can get rather involved and you might not want to do it more than one time a day. After you have one login session on a machine any new connections will get piped transparently through the first connection. Obviously, you will need to change the path to your ssh directory and your identity file line will be different. Also you will need to create a directory tmp inside of your ssh directory.



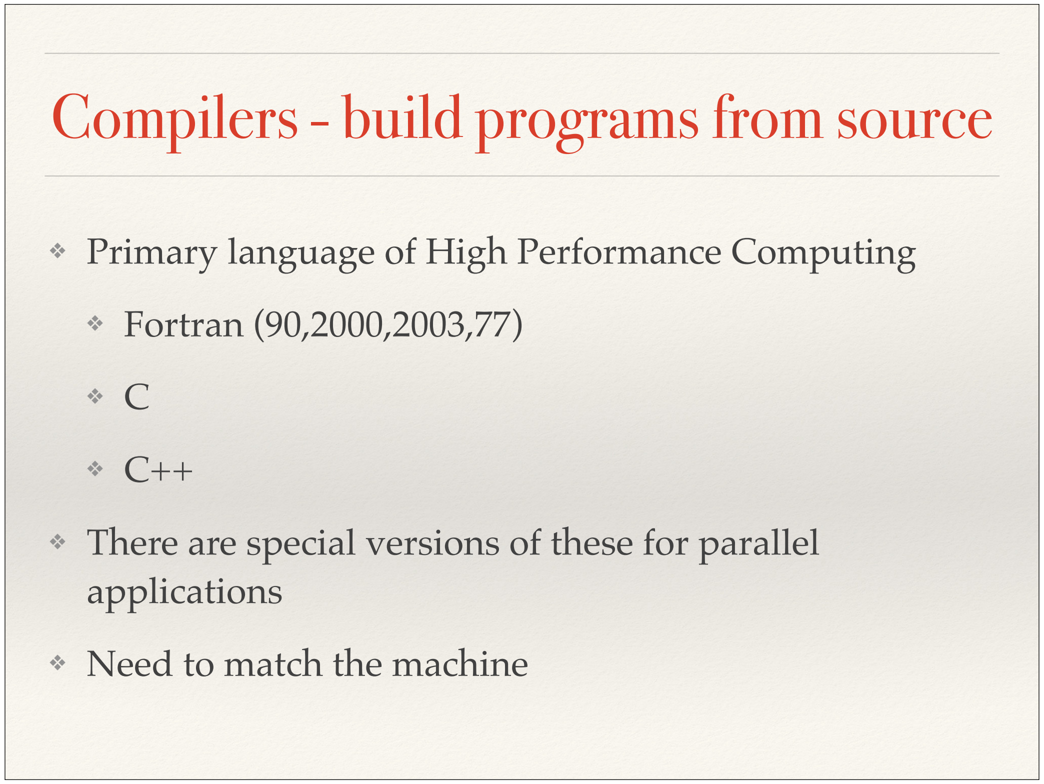
Building Programs

- Compilers
- make
- configure
- cmake

Every command that you run on a compute has at its root a program that someone built. Most programs start off in some language that people can understand. We then use compilers to convert these programs to a language the computer can understand.

The make command is designed to take an input file, called a makefile, of instructions for building programs. The instructions usually have something to do with what source files are used to create a program and how those are to be compiled. The makefile can be rather complicated. Configure and c make are two programs that are designed to make it easier to build makefiles.

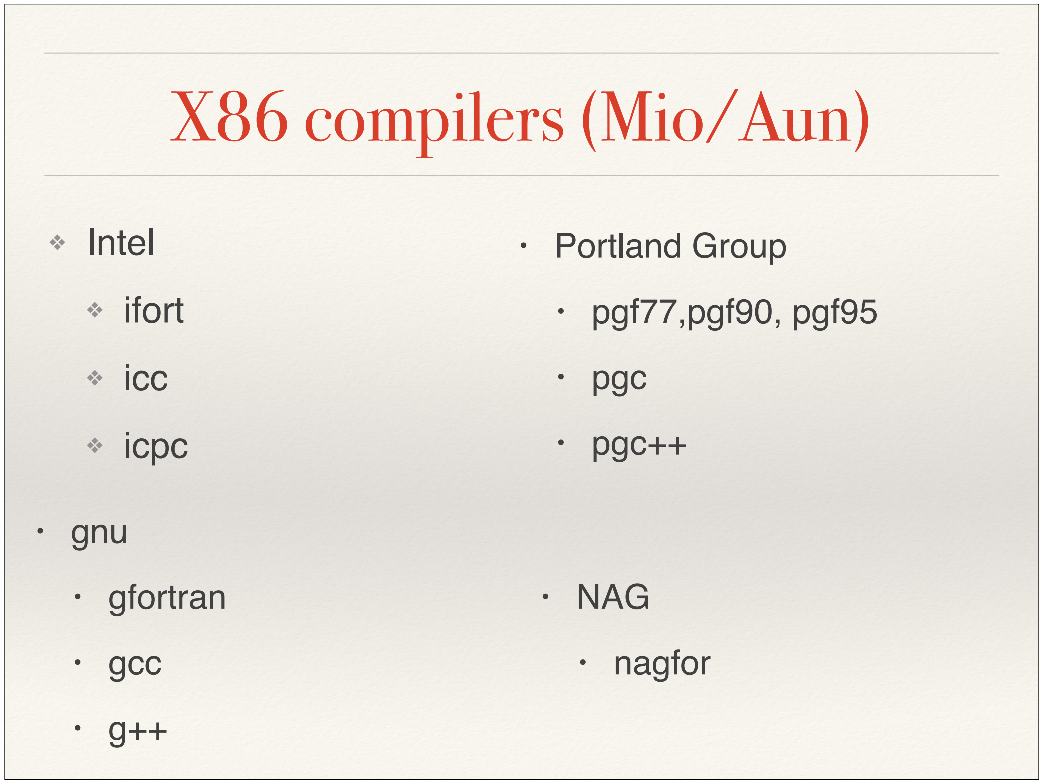
We'll talk first about compilers.



Compilers - build programs from source

- Primary language of High Performance Computing
  - Fortran (90,2000,2003,77)
  - C
  - C++
- There are special versions of these for parallel applications
- Need to match the machine

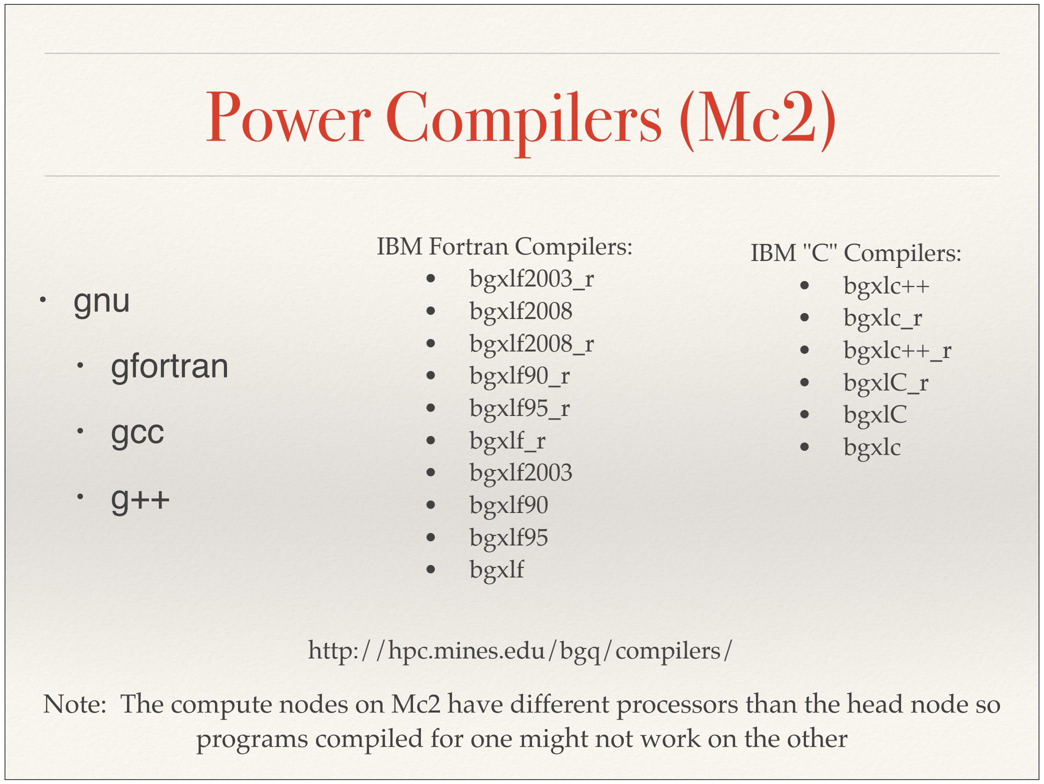
The primary languages of High Performance Computing are Fortran, C and, C++. A person would write in one of these languages and then call a compiler to convert them to machine languages. If you are running on a parallel machine you will be calling special versions of the compilers. Different types of machines will have their own compiler set.



X86 compilers (Mio/Aun)

- Intel
  - ifort
  - icc
  - icpc
- gnu
  - gfortran
  - gcc
  - g++
- Portland Group
  - pgf77,pgf90, pgf95
  - pgc
  - pgc++
- NAG
  - nagfor

On our machines Mio and A u N we have compilers that are designed to run on the Intel X86 chip sets. We have three commercial compiler sets from Intel, Portland Group, and NAG. The fortran compilers are ifort, pgf77, pgf90, pgf95, g fortran, and nagfor. The rest of the compilers listed here are for C and C++ programs.



Power Compilers (Mc2)

- gnu
  - gfortran
  - gcc
  - g++

IBM Fortran Compilers:

- bgxlf2003_r
- bgxlf2008
- bgxlf2008_r
- bgxlf90_r
- bgxlf95_r
- bgxlf_r
- bgxlf2003
- bgxlf90
- bgxlf95
- bgxlf

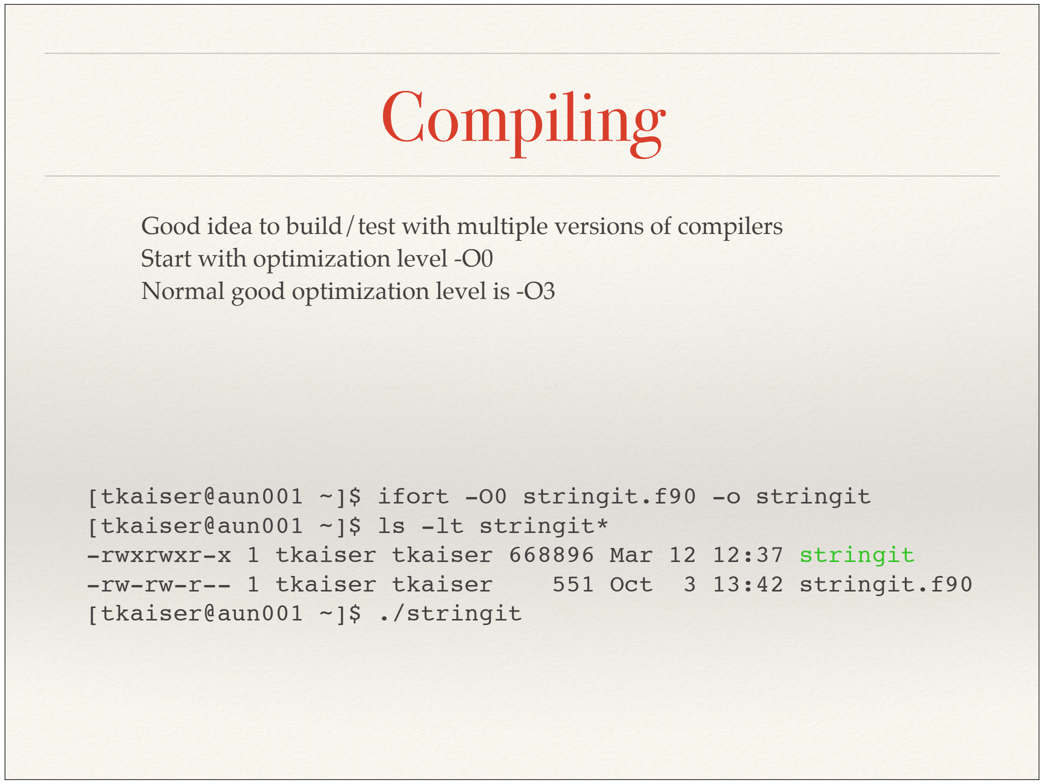
IBM "C" Compilers:

- bgxlc++
- bgxlc_r
- bgxlc++_r
- bgxlC_r
- bgxlC
- bgxlc

http://hpc.mines.edu/bgq/compilers/

Note: The compute nodes on Mc2 have different processors than the head node so programs compiled for one might not work on the other

On our machine M C 2 we have the gnu open source compilers as well as IBM specific compilers. The list of IBM compilers shown here are for serial codes. There is a different set of compilers if you want to run large scale parallel applications. Also, the compute nodes on Mc2 have different processors than the head node. So programs compiled for one might not work on the other.

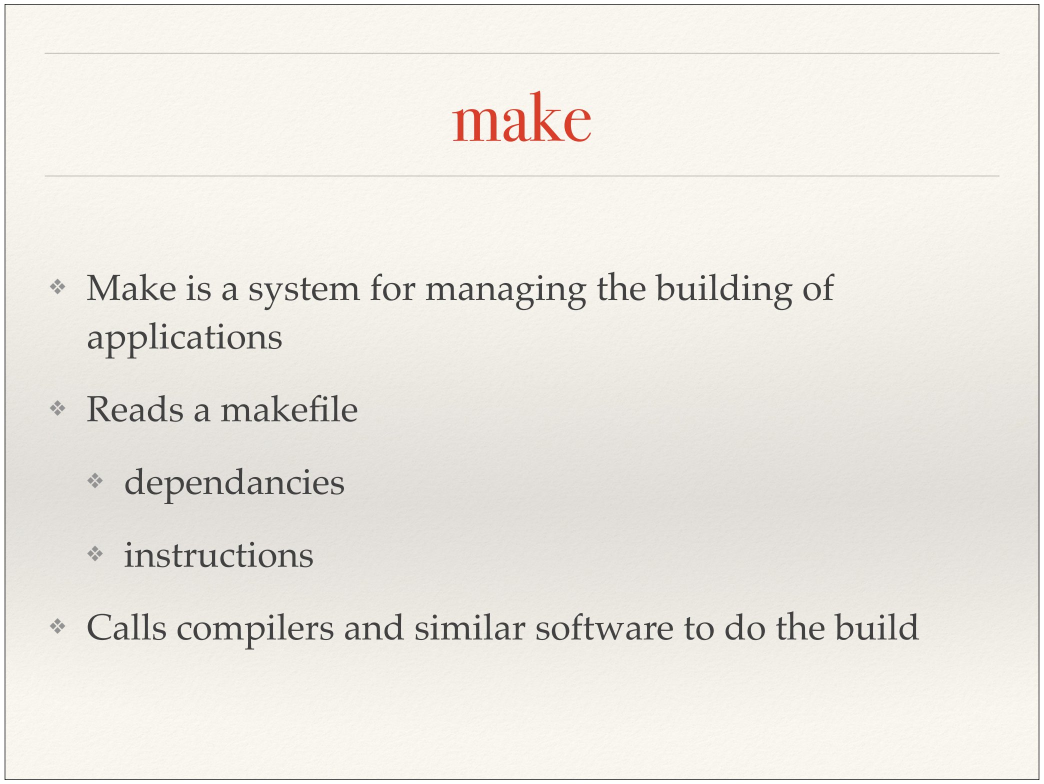


Compiling

Good idea to build / test with multiple versions of compilers
Start with optimization level -O0
Normal good optimization level is -O3

```
[tkaiser@aun001 ~]$ ifort -O0 stringit.f90 -o stringit
[tkaiser@aun001 ~]$ ls -lt stringit*
-rwxrwxr-x 1 tkaiser tkaiser 668896 Mar 12 12:37 stringit
-rw-rw-r-- 1 tkaiser tkaiser    551 Oct  3 13:42 stringit.f90
[tkaiser@aun001 ~]$ ./stringit
```

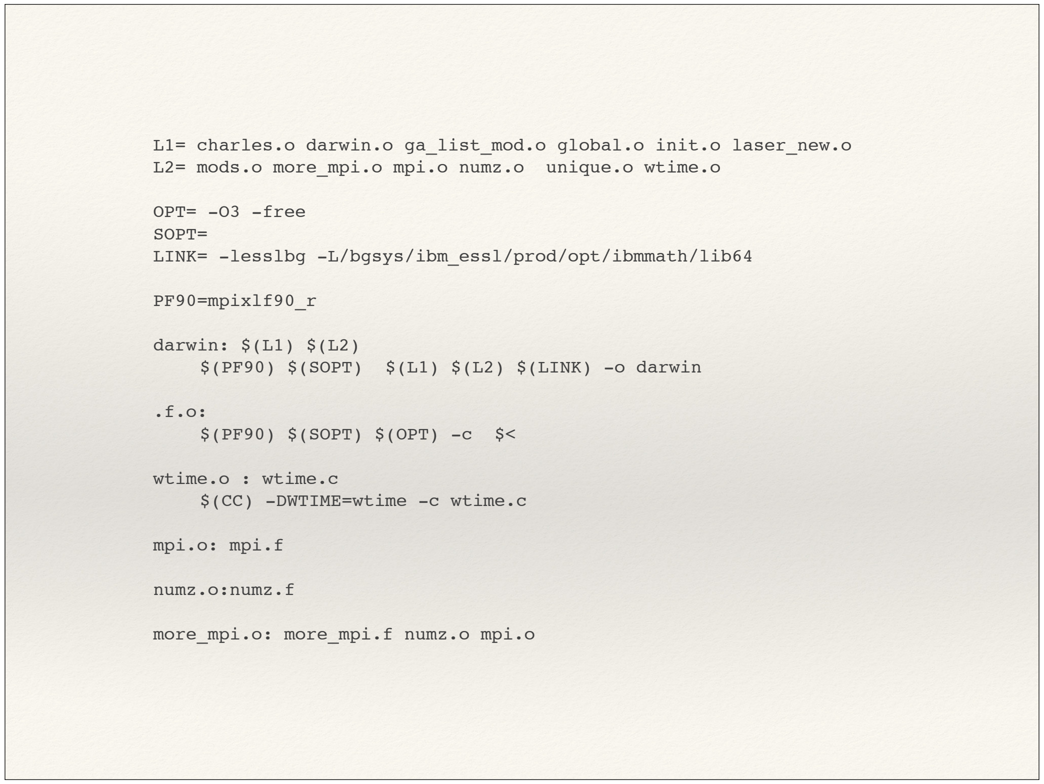
In general it is a good idea to build and test with multiple versions of compilers from different vendors. Also, you want to first run you program at a low level of optimization and then see if you get the same result if you use a higher level of optimization. Here we are building a Fortran program, — string it dot f 90 — into an application — string it — using a low level of optimization.



make

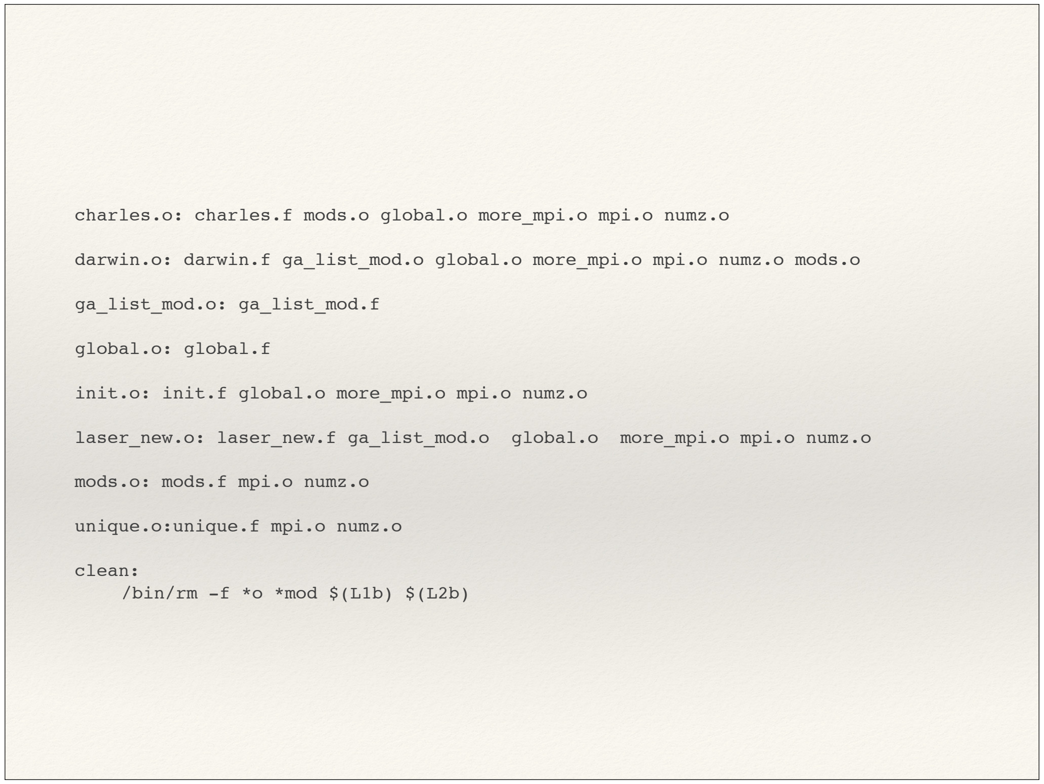
- Make is a system for managing the building of applications
- Reads a makefile
  - dependancies
  - instructions
- Calls compilers and similar software to do the build

As we mentioned, the make utility can be used to help build applications. In particular, make is a system for managing the building of applications. It reads a makefile which contains dependancies, that is the order in which things need to be built, and instructions for doing the build. It then calls compilers and similar software to do the build.



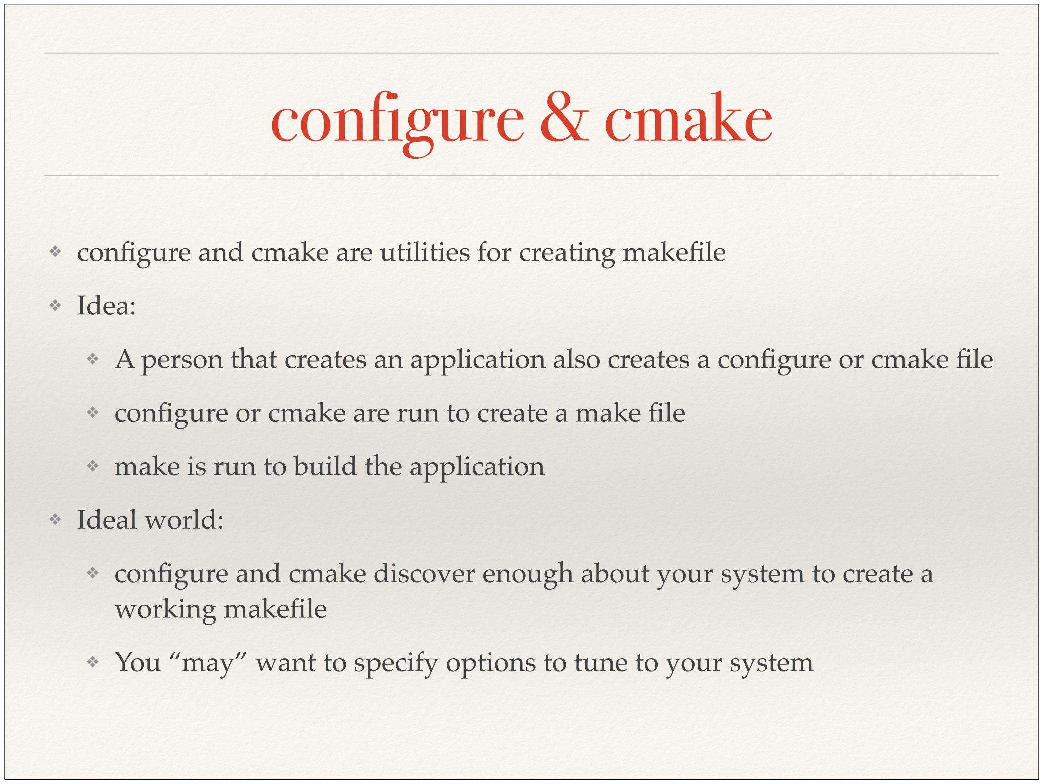
```
L1= charles.o darwin.o ga_list_mod.o global.o init.o laser_new.o
L2= mods.o more_mpi.o mpi.o numz.o  unique.o wtime.o

OPT= -O3 -free
SOPT=
LINK= -lesslbg -L/bgsys/ibm_essl/prod/opt/ibmmath/lib64

PF90=mpixlf90_r

darwin: $(L1) $(L2)
    $(PF90) $(SOPT)  $(L1) $(L2) $(LINK) -o darwin

.f.o:
    $(PF90) $(SOPT) $(OPT) -c  $<

wtime.o : wtime.c
    $(CC) -DWTIME=wtime -c wtime.c

mpi.o: mpi.f

numz.o:numz.f

more_mpi.o: more_mpi.f numz.o mpi.o
```

This is the first half of a slightly complex makefile. The L1 and L2 lines are a list of intermediate files that will be built. The OPT line is the options that will be passed to the compiler. The PF90 line gives the name of the fortran compiler we will be using for the build. The next two lines show the final build command and dependancies. We are building a program called darwin that depends on the the files listed in L1 and L2. The next line gives the command for the final build. Note that it uses a collection of variables defined above for the command. The next set of lines give the default command line for building Fortran source codes. This will be used unless we specifically give instructions for individual files. The following lines have special instructions for some of our files.



```
charles.o: charles.f mods.o global.o more_mpi.o mpi.o numz.o

darwin.o: darwin.f ga_list_mod.o global.o more_mpi.o mpi.o numz.o mods.o

ga_list_mod.o: ga_list_mod.f

global.o: global.f

init.o: init.f global.o more_mpi.o mpi.o numz.o

laser_new.o: laser_new.f ga_list_mod.o  global.o  more_mpi.o mpi.o numz.o

mods.o: mods.f mpi.o numz.o

unique.o:unique.f mpi.o numz.o

clean:
    /bin/rm -f *o *mod $(L1b) $(L2b)
```

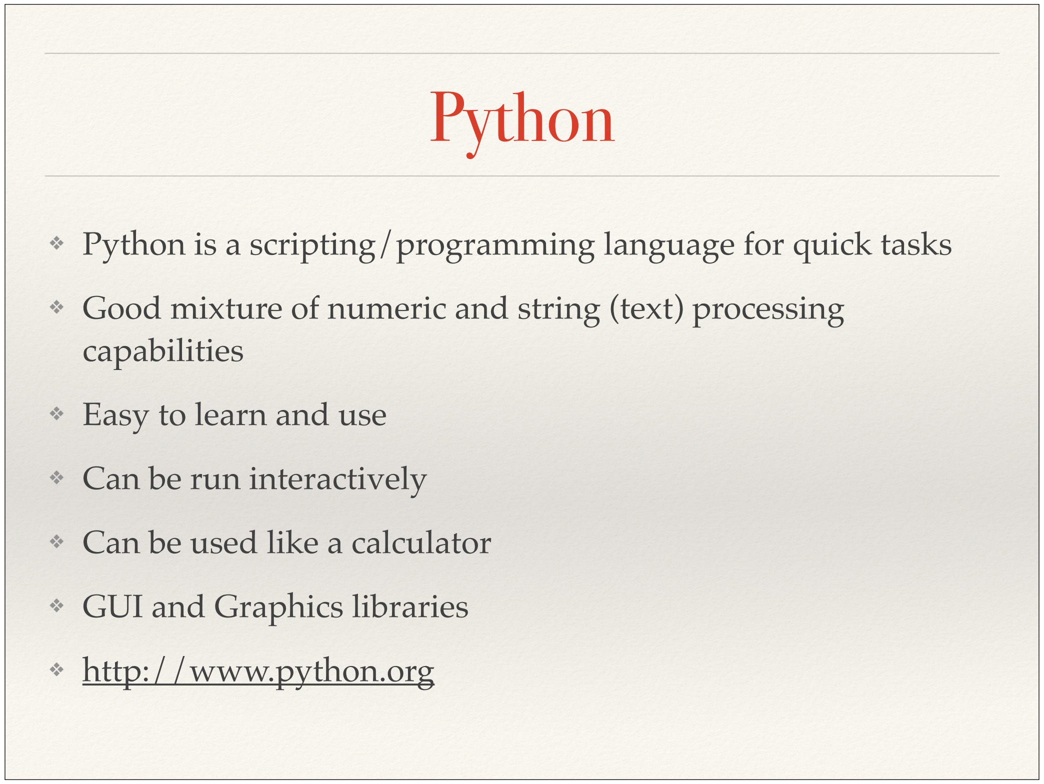
On the second half of the makefile we list the dependencies. For example to build the file charles dot O we need the fortran source file and a collection of other dot O files. So before make tries to build charles dot O it will build the other dot O files. Also, if we change charles . F and rerun make it will rebuild charles dot O.



configure & cmake

- configure and cmake are utilities for creating makefile
- Idea:
  - A person that creates an application also creates a configure or cmake file
  - configure or cmake are run to create a make file
  - make is run to build the application
- Ideal world:
  - configure and cmake discover enough about your system to create a working makefile
  - You “may” want to specify options to tune to your system

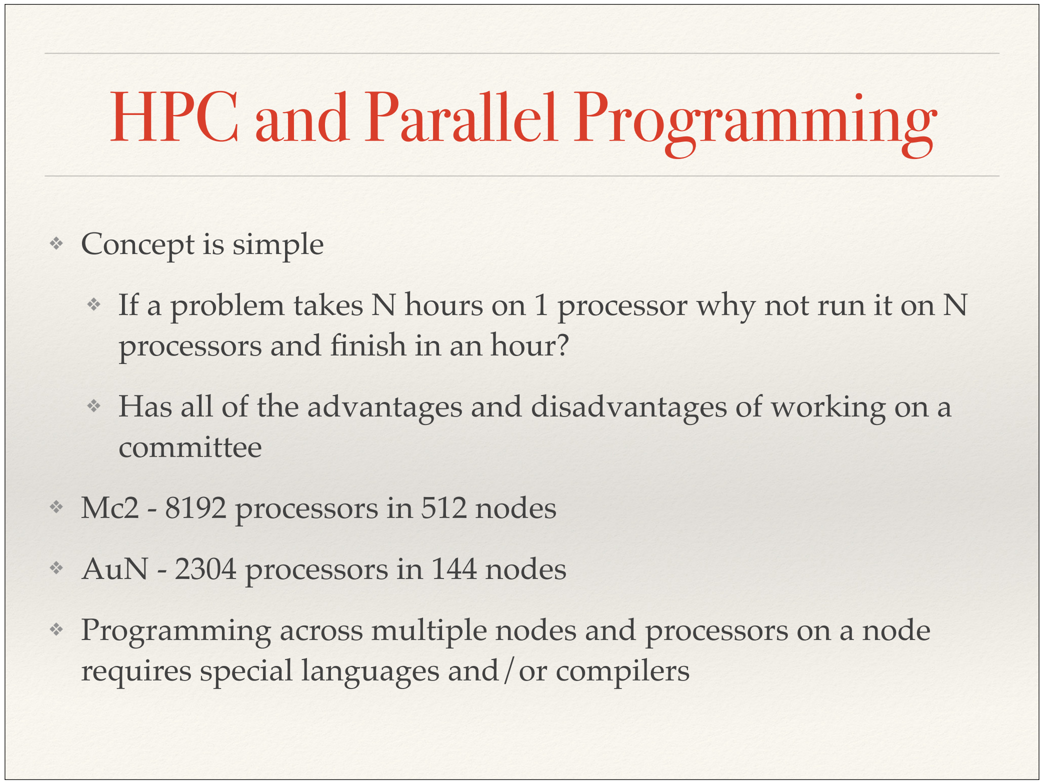
Configure and c make are utilities for creating makefiles. If you down load a large program from the web that you are building on a machine the downloaded package may contain a configure or c make file. These can be very complicated. The idea is that a person that creates an application also creates a configure or c make file that are distributed with the source. You then run configure or c make to create a make file and then finally the utility make is run to build the application. In the ideal world configure and c make will discover enough about your system to create a working makefile. Good luck!



Python

- Python is a scripting/programming language for quick tasks
- Good mixture of numeric and string (text) processing capabilities
- Easy to learn and use
- Can be run interactively
- Can be used like a calculator
- GUI and Graphics libraries
- http://www.python.org

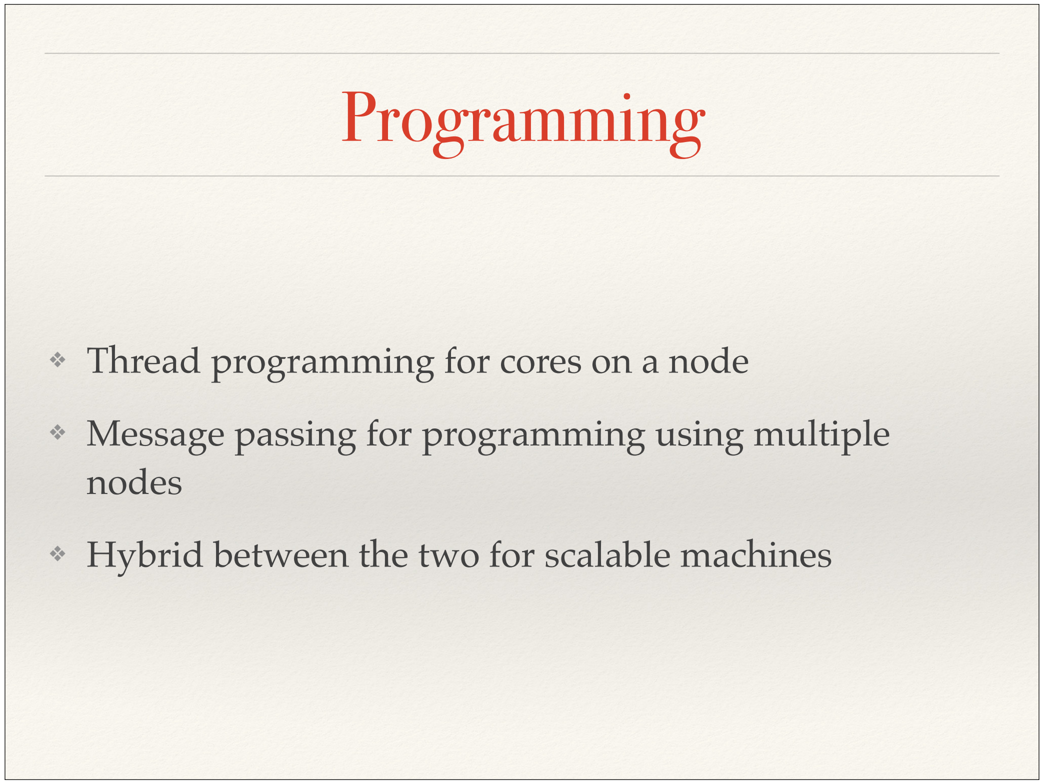
Even if you don’t write Fortran of C programs if you are running on a linux system it might be useful to learn python. Python is a scripting and programming language for quick tasks that can also be used for post processing data from other programs. It has a good mixture of numeric and string (text) processing capabilities. It even supports complex numbers. It is easy to learn and use. You can write complete programs or it can be run interactively. It can be used like a calculator. There are many numeric and graphics libraries available.



HPC and Parallel Programming

- Concept is simple
  - If a problem takes N hours on 1 processor why not run it on N processors and finish in an hour?
  - Has all of the advantages and disadvantages of working on a committee
- Mc2 - 8192 processors in 512 nodes
- AuN - 2304 processors in 144 nodes
- Programming across multiple nodes and processors on a node requires special languages and/or compilers

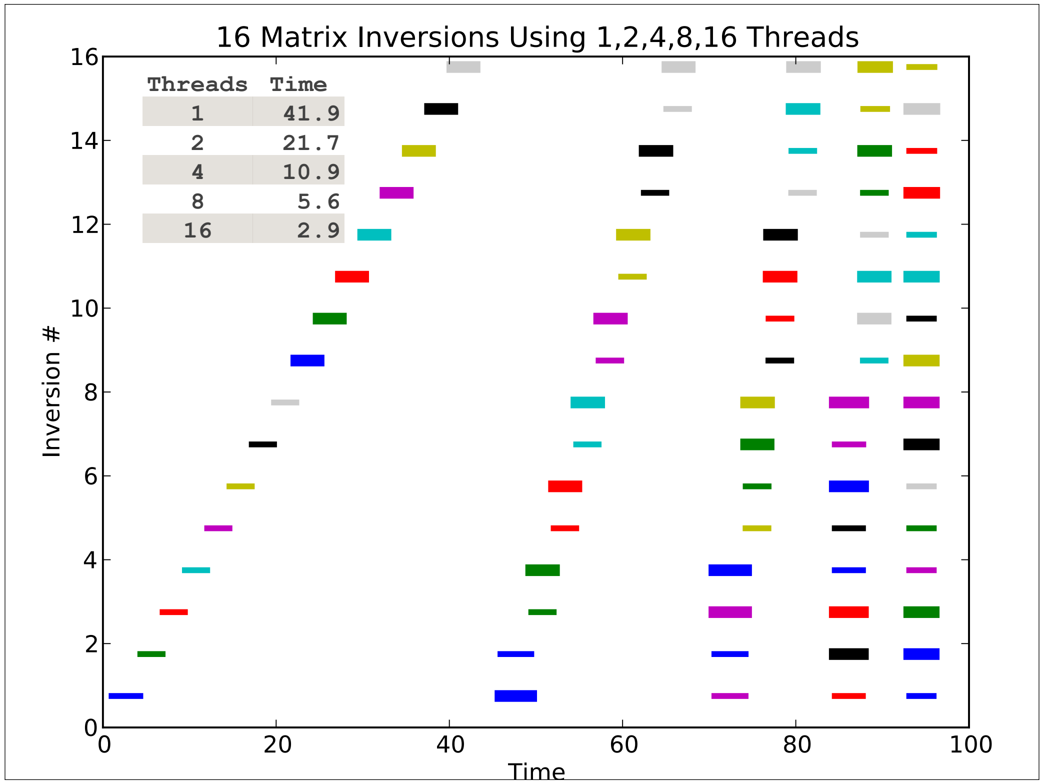
Finally, we will talk about high performance computing and parallel programming. The concept is simple. If a problem takes N hours on 1 processor why not run it on N processors and finish in an hour? Running a Parallel program can be thought of doing computation by committee. It has all of the advantages and disadvantages of working on a committee. That is, if everyone does there work, there is little time spent in communications (meetings), everyone has about the same amount of work to do, no one has to wait too long for input from another, then things will run well. Our machine M C 2 has 8192 processors that can be used on a single task. A U N has 2304 processors in groups of 16 on 144 nodes. Programming to run on multiple processors requires special languages and/or compilers.



Programming

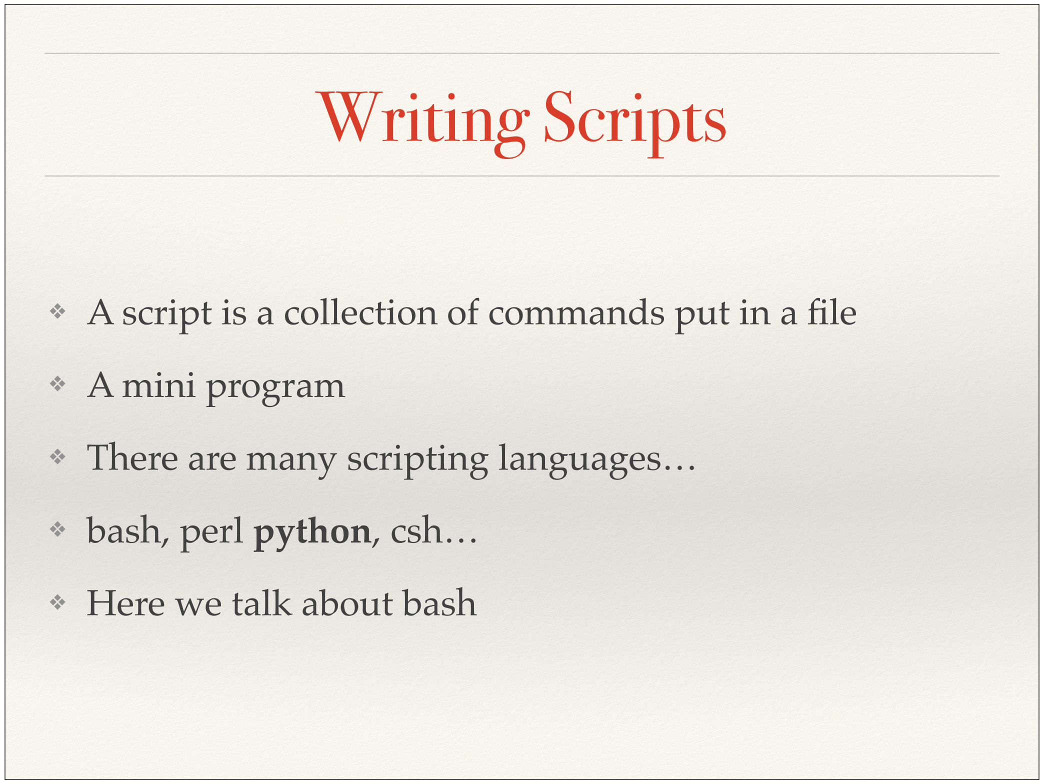
- Thread programming for cores on a node
- Message passing for programming using multiple nodes
- Hybrid between the two for scalable machines

You can do what's called threads based programming if you only want to use all of the cores or processors in a single node. If you want to use multiple nodes together you will need to use software that can pass messages between the nodes. It is also possible to run using both types of parallelism.



Threads	Time
1	41.9
2	21.7
4	10.9
8	5.6
16	2.9

Here we have a chart showing the wall clock time on a program. We are doing 16 matrix inversions 5 times. The first time we are not running in parallel. We do one inversion after the next and it took 41.9 seconds to do them all. The blips on the chart show the start and runtime for each inversion. Then we did them two at a time and we ran in 21.7 seconds. Notice the overlap in the second set of blips. Then we did 4, 8, and 16 at a time with a best runtime of 2.9 seconds doing all 16 inversions together.

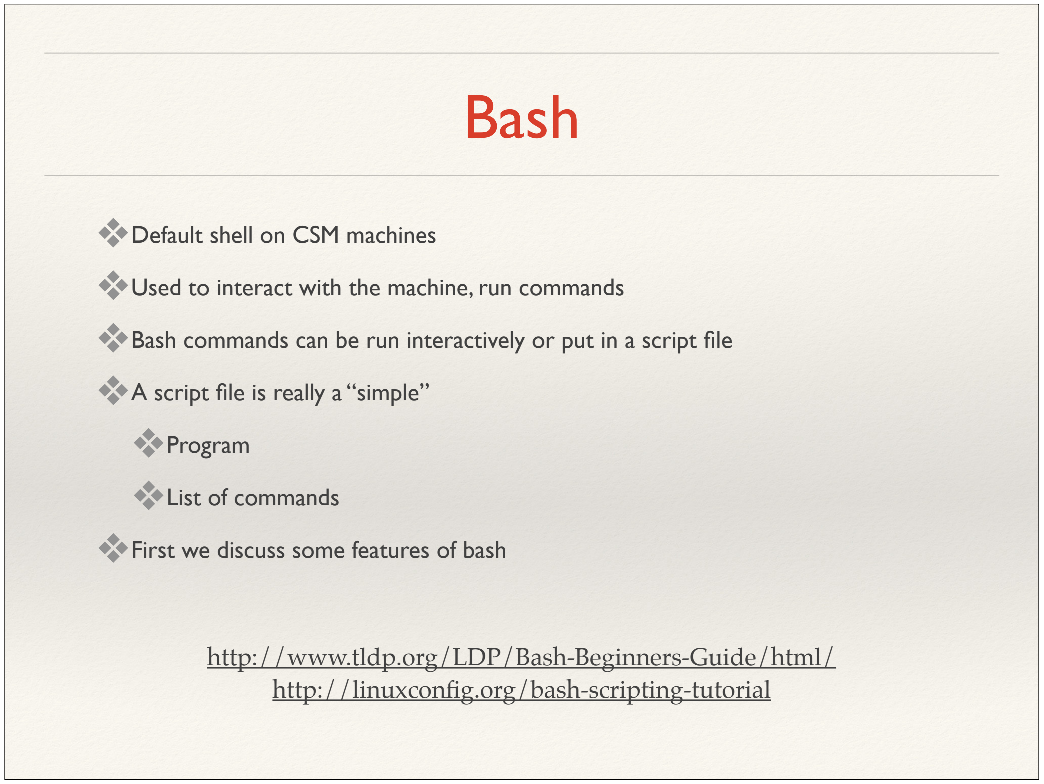


Writing Scripts

- A script is a collection of commands put in a file
- A mini program
- There are many scripting languages…
- bash, perl **python**, csh…
- Here we talk about bash

Next I want to talk briefly about writing scripts. When you are interacting with the computer on the command line the program that takes your input and runs the commands is called the shell. The default shell program on our machines is called bash.

If you have a set of commands you run often you can put them in a script. Then the script can be run just like a regular program. Since it is like a program there are different scripting languages that can be used but since we have already showed you bash commands we will show some scripts that include bash commands.



Bash

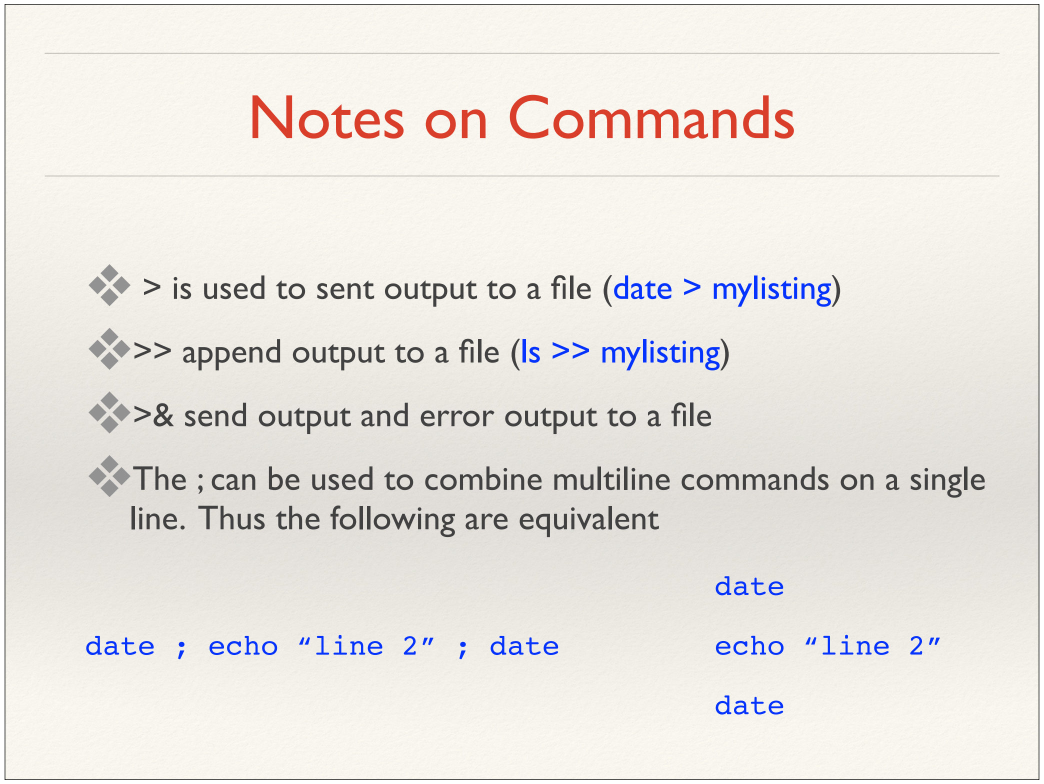
- Default shell on CSM machines
- Used to interact with the machine, run commands
- Bash commands can be run interactively or put in a script file
- A script file is really a "simple"
    - Program
    - List of commands
- First we discuss some features of bash

http://www.tldp.org/LDP/Bash-Beginners-Guide/html/
http://linuxconfig.org/bash-scripting-tutorial

As we said, we will use bash in our examples but other shells and scripting languages have similar capabilities.

Bash commands can be run interactively or put in a script file that is then run.

A script file is really a "simple" list of commands or program. It can the structure of a normal program with tests, branches and loops.



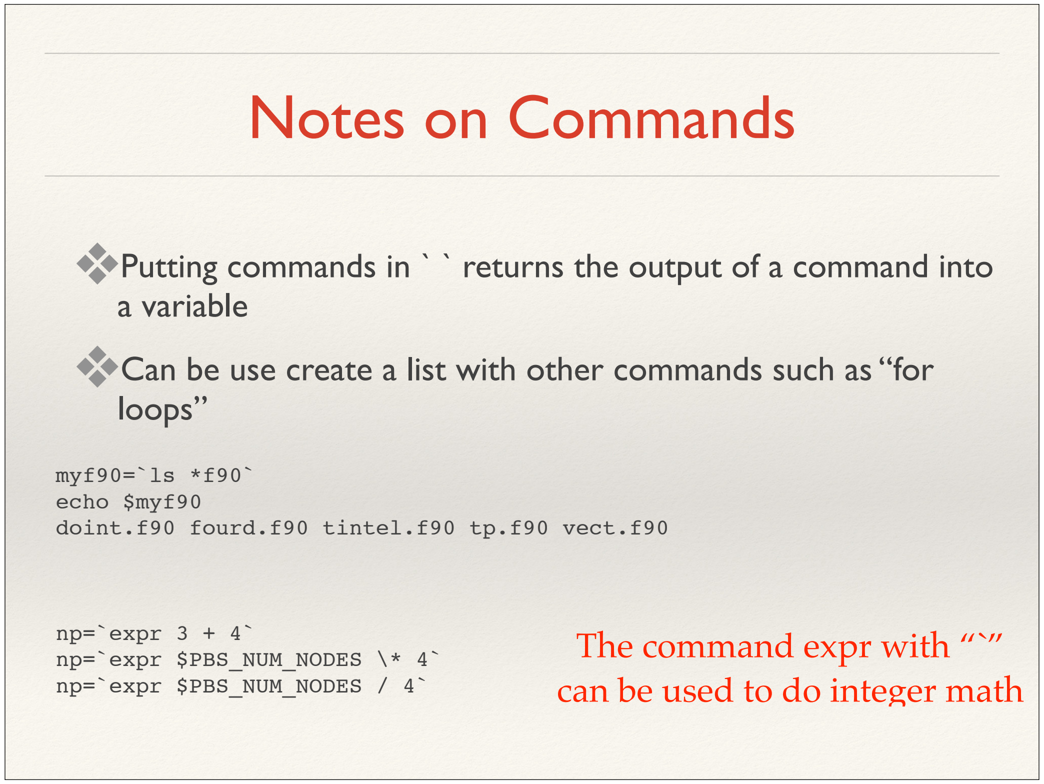
Notes on Commands

- > is used to sent output to a file (date > mylisting)
- >> append output to a file (ls >> mylisting)
- >& send output and error output to a file
- The ; can be used to combine multiline commands on a single line. Thus the following are equivalent

```
date ; echo “line 2” ; date
```

```
date
echo “line 2”
date
```

We want to talk a bit about some of the syntax features in bash that we will be using. The “greater than” symbol can be used to send output from a command to a file. A double “greater than” will append output to an existing file. If you use “greater than” along with an “ampersand” then output and error information are sent to a file. The “semicolon” can be used to combine a collection of commands into a single line.



Notes on Commands

- Putting commands in ` ` returns the output of a command into a variable
- Can be use create a list with other commands such as "for loops"

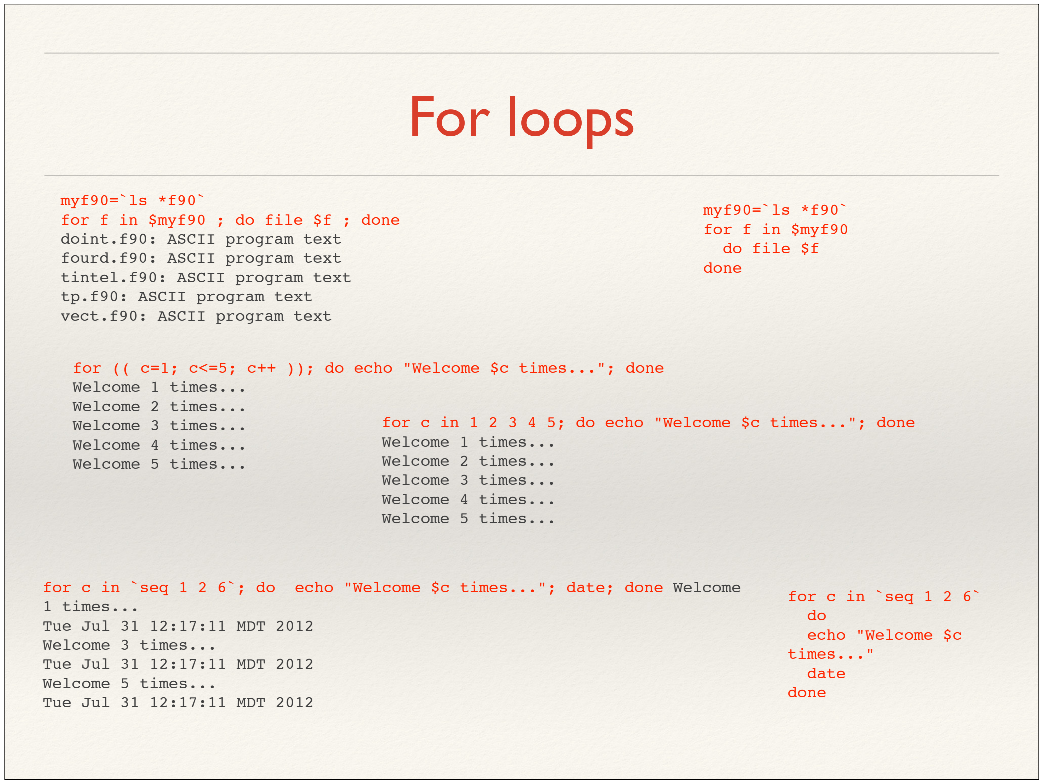
```
myf90=`ls *f90`
echo $myf90
doint.f90 fourd.f90 tintel.f90 tp.f90 vect.f90
```

```
np=`expr 3 + 4`
np=`expr $PBS_NUM_NODES \* 4`
np=`expr $PBS_NUM_NODES / 4`
```

The command expr with "`" can be used to do integer math

Putting commands in back tick marks returns the output of a command into a variable.
One way we used this is to create a list with other commands.
Here for example, we get a list of Fortran 90 programs.
You will see in the next slide this can be used to do loops.

We can use the back tick along with the expr command to do integer math.



For loops

```
myf90=`ls *f90`
for f in $myf90 ; do file $f ; done
doint.f90: ASCII program text
fourd.f90: ASCII program text
tintel.f90: ASCII program text
tp.f90: ASCII program text
vect.f90: ASCII program text
```

```
myf90=`ls *f90`
for f in $myf90
  do file $f
done
```

```
for (( c=1; c<=5; c++ )); do echo "Welcome $c times..."; done
Welcome 1 times...
Welcome 2 times...
Welcome 3 times...
Welcome 4 times...
Welcome 5 times...
```

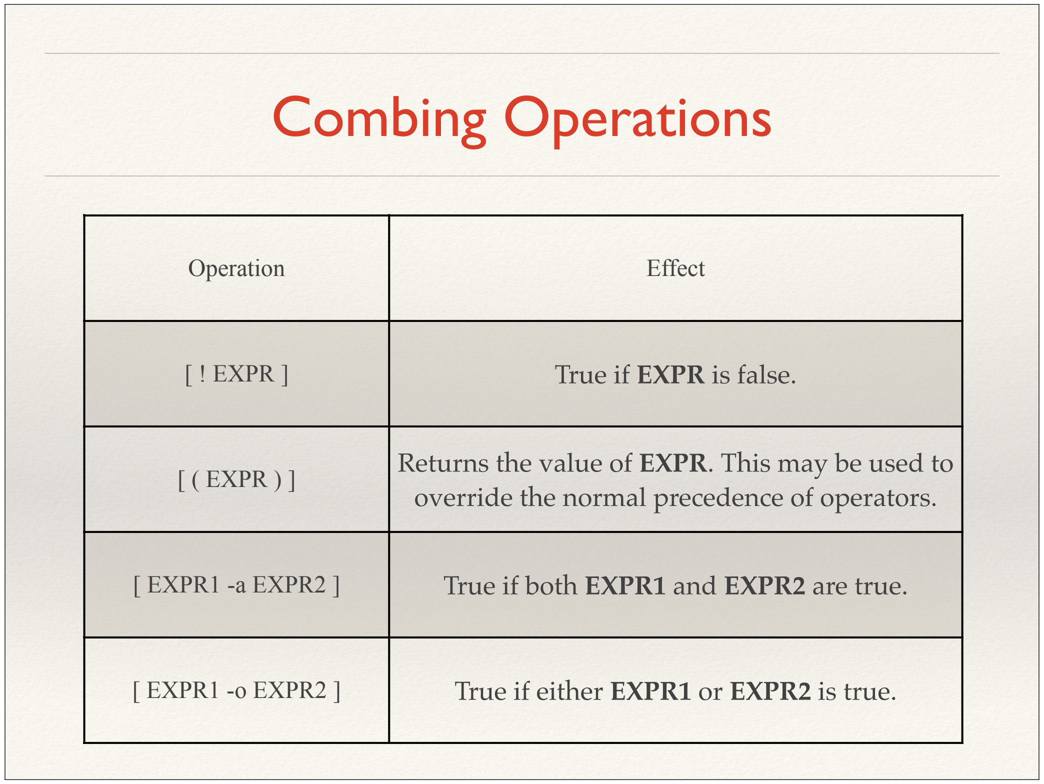
```
for c in 1 2 3 4 5; do echo "Welcome $c times..."; done
Welcome 1 times...
Welcome 2 times...
Welcome 3 times...
Welcome 4 times...
Welcome 5 times...
```

```
for c in `seq 1 2 6`; do  echo "Welcome $c times..."; date; done Welcome
1 times...
Tue Jul 31 12:17:11 MDT 2012
Welcome 3 times...
Tue Jul 31 12:17:11 MDT 2012
Welcome 5 times...
Tue Jul 31 12:17:11 MDT 2012
```

```
for c in `seq 1 2 6`
  do
  echo "Welcome $c
times..."
  date
done
```

Here we have four types of — for — loops. In the first case we are looping over a list of files that was created with the L S command.

In the second case were using C style indexing. For the third case we are actually incrementing over a list of items given on the command line. Note, these do not need to be numbers. Finally, we use the S E Q or sequence command to generate the values for our loop.

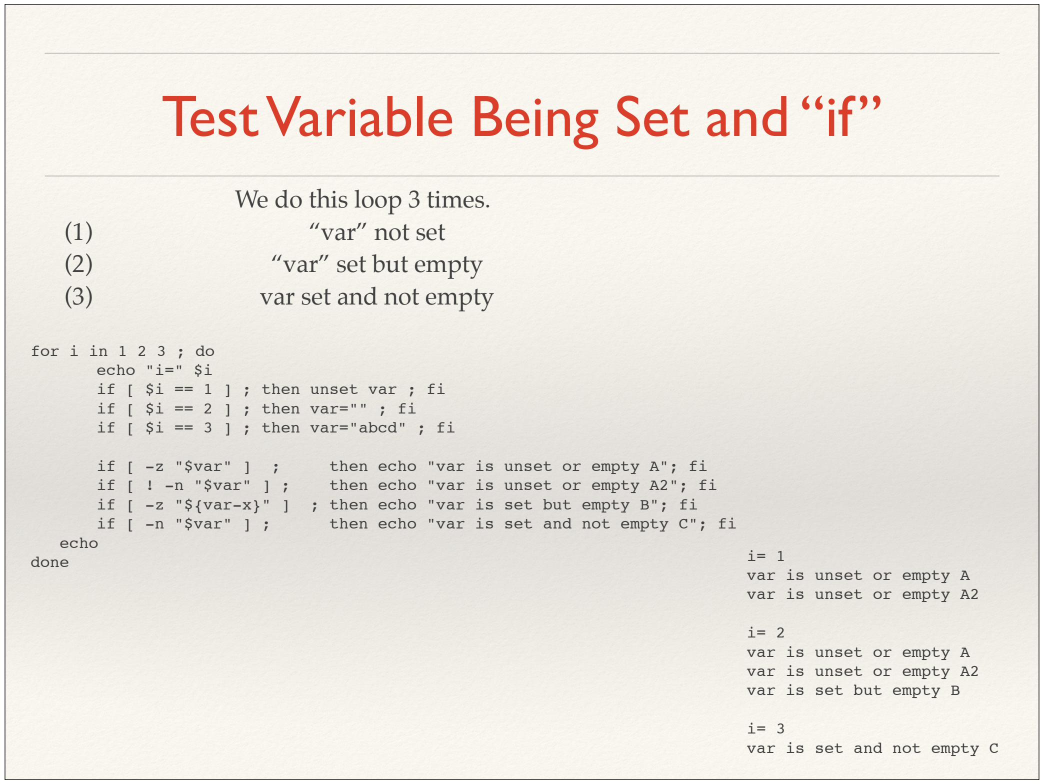


Combing Operations

Operation	Effect
[! EXPR]	True if **EXPR** is false.
[(EXPR)]	Returns the value of **EXPR**. This may be used to override the normal precedence of operators.
[EXPR1 -a EXPR2]	True if both **EXPR1** and **EXPR2** are true.
[EXPR1 -o EXPR2]	True if either **EXPR1** or **EXPR2** is true.

We are going to look at “if” statements and testing variables next.
Before that, we note that expressions can be logically grouped.

Minus A is the and operation.
Minus O allows the or operation.
The explanation mark negates the value of the expression.
The brackets do precedence changes.



Test Variable Being Set and “if”

We do this loop 3 times.

(1) “var” not set

(2) “var” set but empty

(3) var set and not empty

```
for i in 1 2 3 ; do
       echo "i=" $i
       if [ $i == 1 ] ; then unset var ; fi
       if [ $i == 2 ] ; then var="" ; fi
       if [ $i == 3 ] ; then var="abcd" ; fi

       if [ -z "$var" ]  ;     then echo "var is unset or empty A"; fi
       if [ ! -n "$var" ] ;    then echo "var is unset or empty A2"; fi
       if [ -z "${var-x}" ]  ; then echo "var is set but empty B"; fi
       if [ -n "$var" ] ;      then echo "var is set and not empty C"; fi
   echo
done
```

```
i= 1
var is unset or empty A
var is unset or empty A2

i= 2
var is unset or empty A
var is unset or empty A2
var is set but empty B

i= 3
var is set and not empty C
```

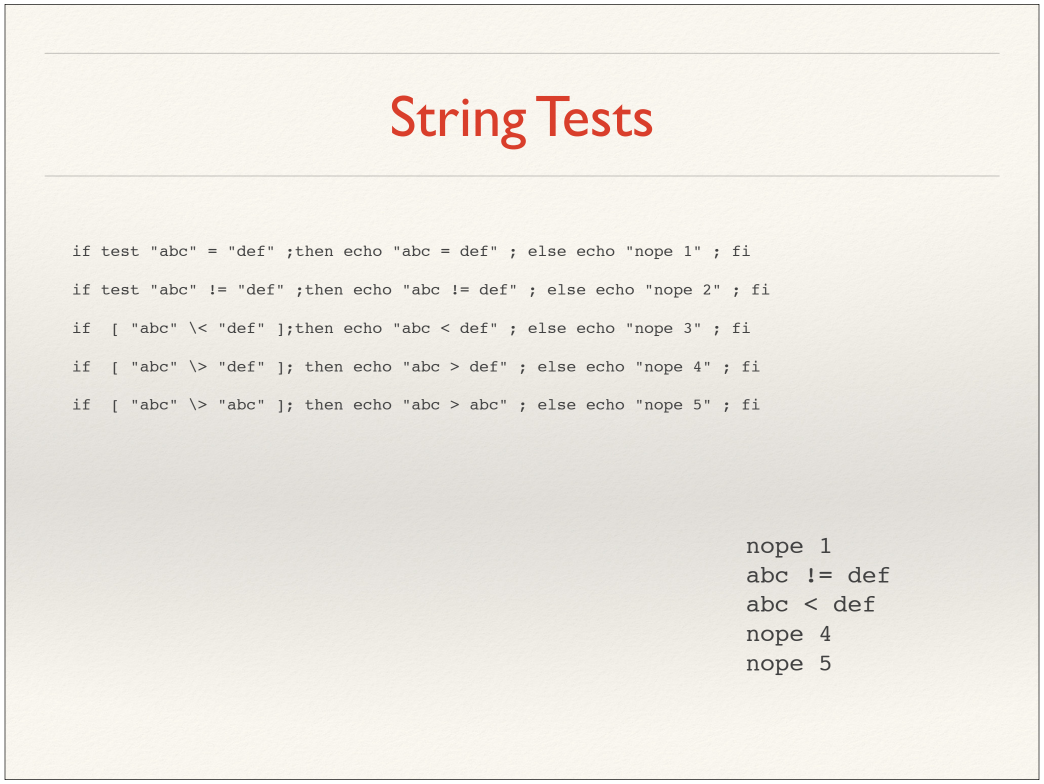
Here we show a few things, including testing to see if a variable is set.

In this for loop we have the values 1, 2, and 3. In the first part of the loop we test I. On the iteration we un set the variable var. In the second iteration we set var to an empty string. Finally in the last iteration we set var to the string A B C D.

— —

In the second part of the loop we do our tests. A minus n test is true if the variable is set to a nonempty string. Minus Z is true if the variable is unset or empty.

We have the results of the tests on the left.

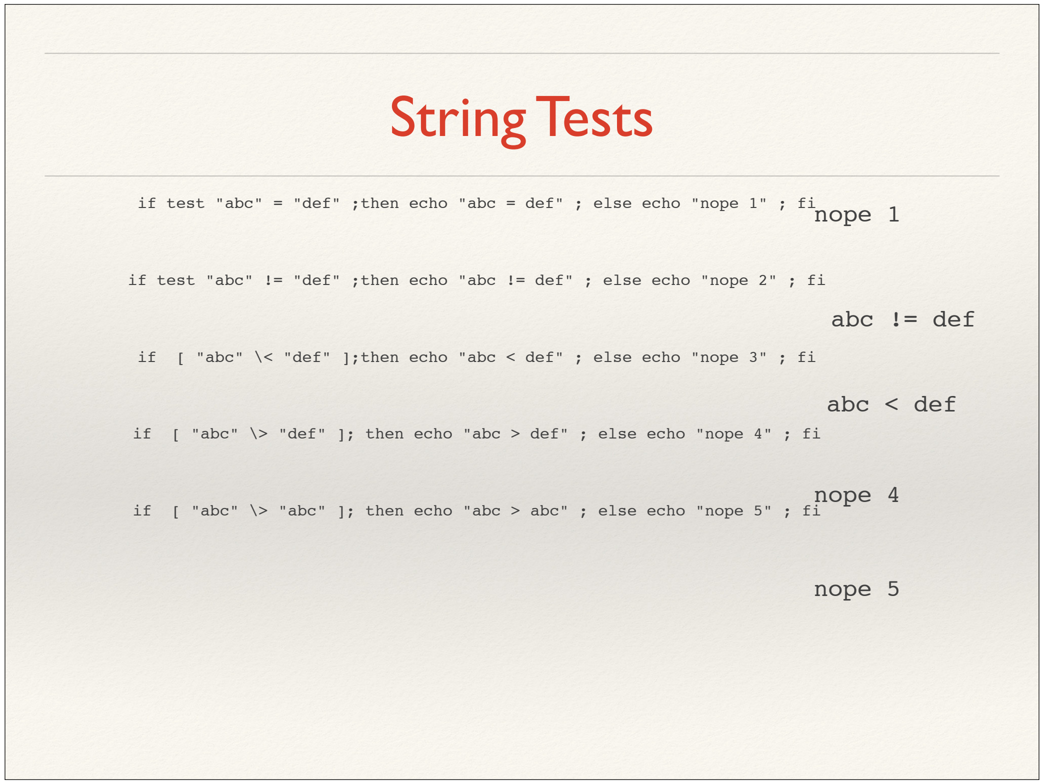


String Tests

```
if test "abc" = "def" ;then echo "abc = def" ; else echo "nope 1" ; fi

if test "abc" != "def" ;then echo "abc != def" ; else echo "nope 2" ; fi

if  [ "abc" \< "def" ];then echo "abc < def" ; else echo "nope 3" ; fi

if  [ "abc" \> "def" ]; then echo "abc > def" ; else echo "nope 4" ; fi

if  [ "abc" \> "abc" ]; then echo "abc > abc" ; else echo "nope 5" ; fi
```

```
nope 1
abc != def
abc < def
nope 4
nope 5
```

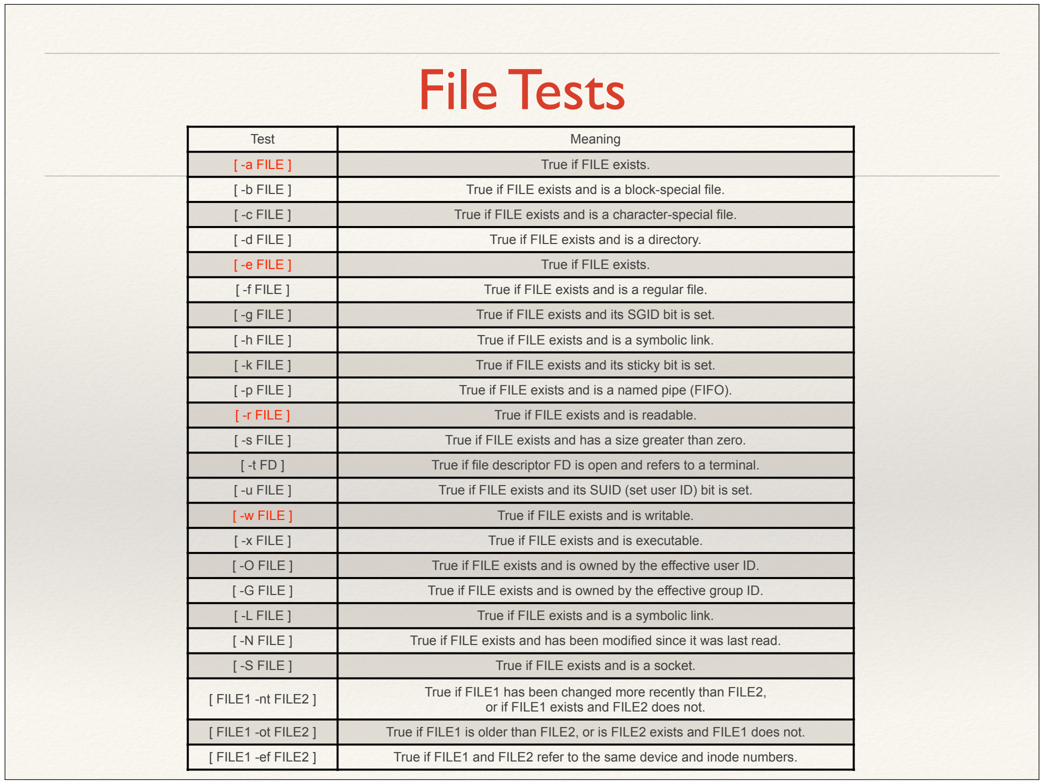
Here we have some string tests. We can test to see if they are the same, different, greater than or less than. Note the back slash in front of the greater than and less that symbols. This is required because these are normally reserved for input and output redirection.



String Tests

```
if test "abc" = "def" ;then echo "abc = def" ; else echo "nope 1" ; fi
nope 1

if test "abc" != "def" ;then echo "abc != def" ; else echo "nope 2" ; fi
abc != def

if  [ "abc" \< "def" ];then echo "abc < def" ; else echo "nope 3" ; fi
abc < def

if  [ "abc" \> "def" ]; then echo "abc > def" ; else echo "nope 4" ; fi
nope 4

if  [ "abc" \> "abc" ]; then echo "abc > abc" ; else echo "nope 5" ; fi
nope 5
```

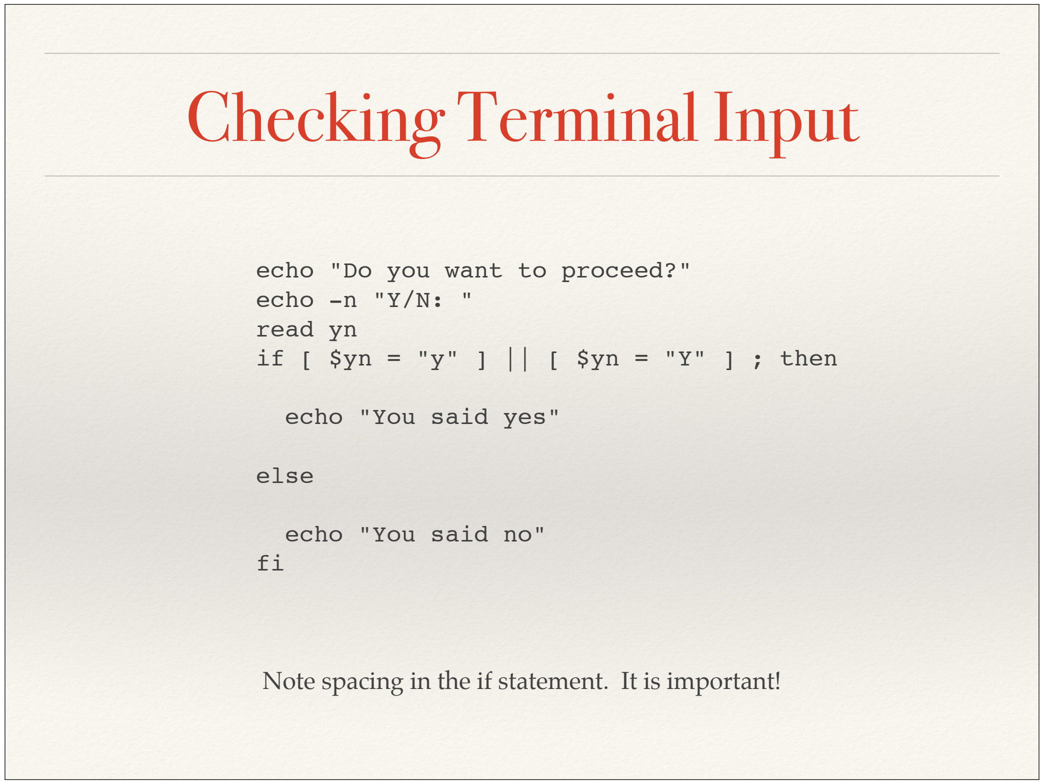
Here we have some string tests. We can test to see if they are the same, different, greater than or less than. Note the back slash in front of the greater than and less that symbols. This is required because these are normally reserved for input and output redirection.



File Tests

Test	Meaning
[-a FILE]	True if FILE exists.
[-b FILE]	True if FILE exists and is a block-special file.
[-c FILE]	True if FILE exists and is a character-special file.
[-d FILE]	True if FILE exists and is a directory.
[-e FILE]	True if FILE exists.
[-f FILE]	True if FILE exists and is a regular file.
[-g FILE]	True if FILE exists and its SGID bit is set.
[-h FILE]	True if FILE exists and is a symbolic link.
[-k FILE]	True if FILE exists and its sticky bit is set.
[-p FILE]	True if FILE exists and is a named pipe (FIFO).
[-r FILE]	True if FILE exists and is readable.
[-s FILE]	True if FILE exists and has a size greater than zero.
[-t FD]	True if file descriptor FD is open and refers to a terminal.
[-u FILE]	True if FILE exists and its SUID (set user ID) bit is set.
[-w FILE]	True if FILE exists and is writable.
[-x FILE]	True if FILE exists and is executable.
[-O FILE]	True if FILE exists and is owned by the effective user ID.
[-G FILE]	True if FILE exists and is owned by the effective group ID.
[-L FILE]	True if FILE exists and is a symbolic link.
[-N FILE]	True if FILE exists and has been modified since it was last read.
[-S FILE]	True if FILE exists and is a socket.
[FILE1 -nt FILE2]	True if FILE1 has been changed more recently than FILE2, or if FILE1 exists and FILE2 does not.
[FILE1 -ot FILE2]	True if FILE1 is older than FILE2, or is FILE2 exists and FILE1 does not.
[FILE1 -ef FILE2]	True if FILE1 and FILE2 refer to the same device and inode numbers.

We can also do file tests. Some of the important ones are to see, if the file exists, see if it’s readable or writable, we can also test using the minus N T, O T, or E F options to see the relative age of files or to see if two files are actually identical.

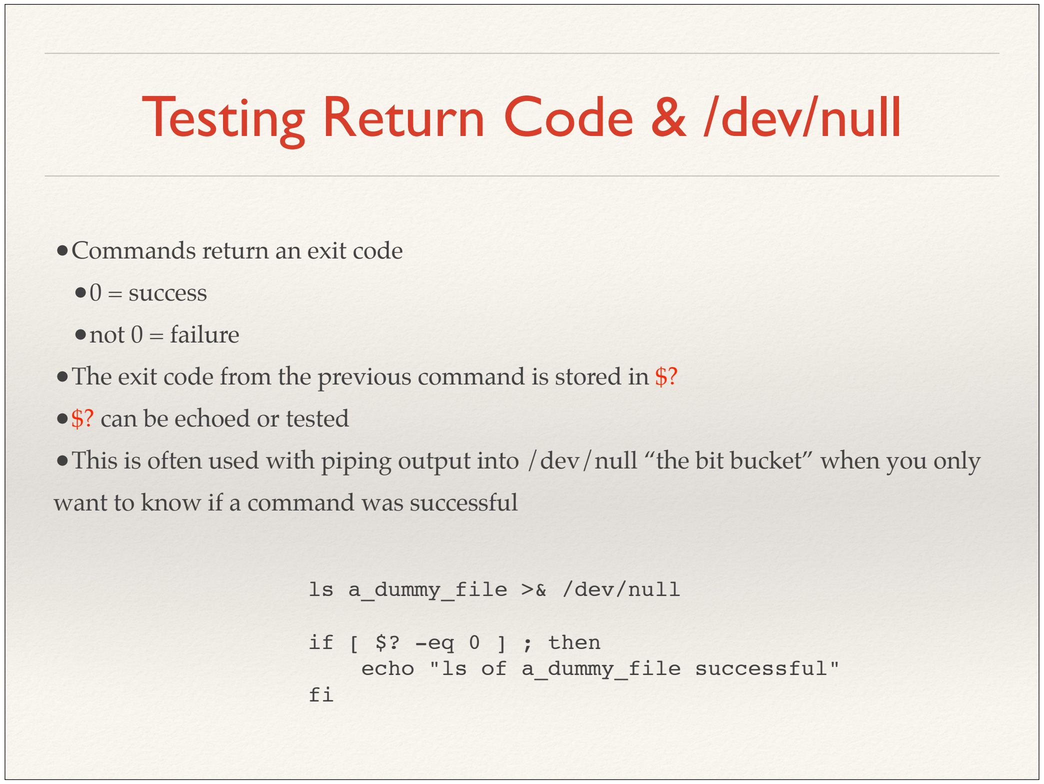


Checking Terminal Input

```
echo "Do you want to proceed?"
echo -n "Y/N: "
read yn
if [ $yn = "y" ] || [ $yn = "Y" ] ; then

  echo "You said yes"

else

  echo "You said no"
fi
```

Note spacing in the if statement. It is important!

If you have an interactive script and you want to check input this is how it is done.



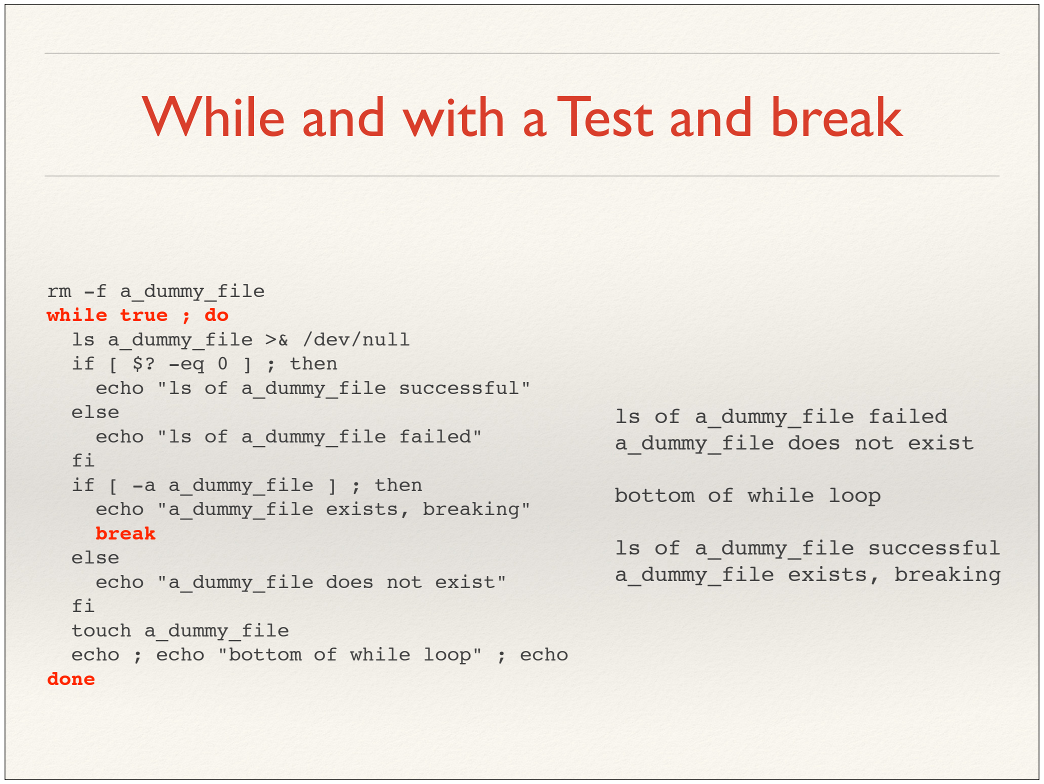
Testing Return Code & /dev/null

- Commands return an exit code
  - 0 = success
  - not 0 = failure
- The exit code from the previous command is stored in $?
- $? can be echoed or tested
- This is often used with piping output into /dev/null "the bit bucket" when you only want to know if a command was successful

```
ls a_dummy_file >& /dev/null

if [ $? -eq 0 ] ; then
    echo "ls of a_dummy_file successful"
fi
```

Unix commands return an exit code. A normal exit or success is 0

The exit code from the previous command is stored in the variable dollar question.
Dollar question can be echoed or tested with an if statement.
This is often used with piping output into /dev/null "the bit bucket" when you
only want to know if a command was successful. This is actually another way to determine
if a files exists. You do an L S of the file and check the exit code from L S.
If it is 0 the file exists.

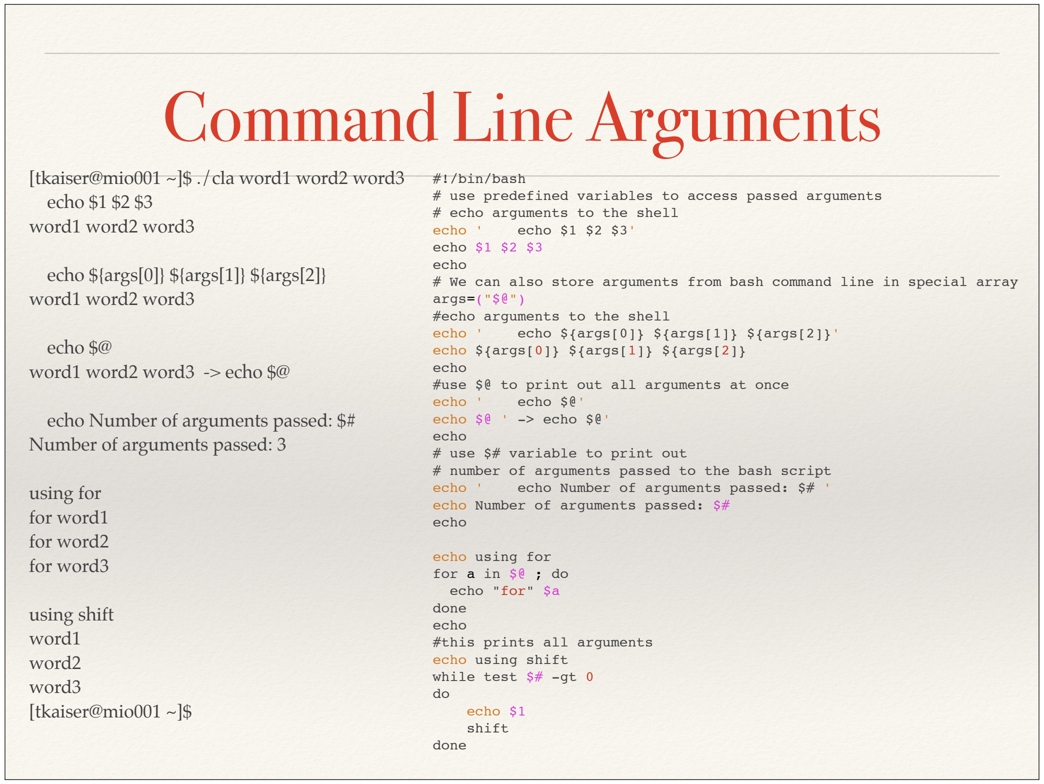


While and with a Test and break

```
rm -f a_dummy_file
while true ; do
  ls a_dummy_file >& /dev/null
  if [ $? -eq 0 ] ; then
    echo "ls of a_dummy_file successful"
  else
    echo "ls of a_dummy_file failed"
  fi
  if [ -a a_dummy_file ] ; then
    echo "a_dummy_file exists, breaking"
    break
  else
    echo "a_dummy_file does not exist"
  fi
  touch a_dummy_file
  echo ; echo "bottom of while loop" ; echo
done
```

```
ls of a_dummy_file failed
a_dummy_file does not exist

bottom of while loop

ls of a_dummy_file successful
a_dummy_file exists, breaking
```

Finally we show the while command. In this case we're first remove the file — a_dummy_file —. We then we use L S command to see if the file exist. If not, we create it. The next time through we break out of the while loop using the break command.

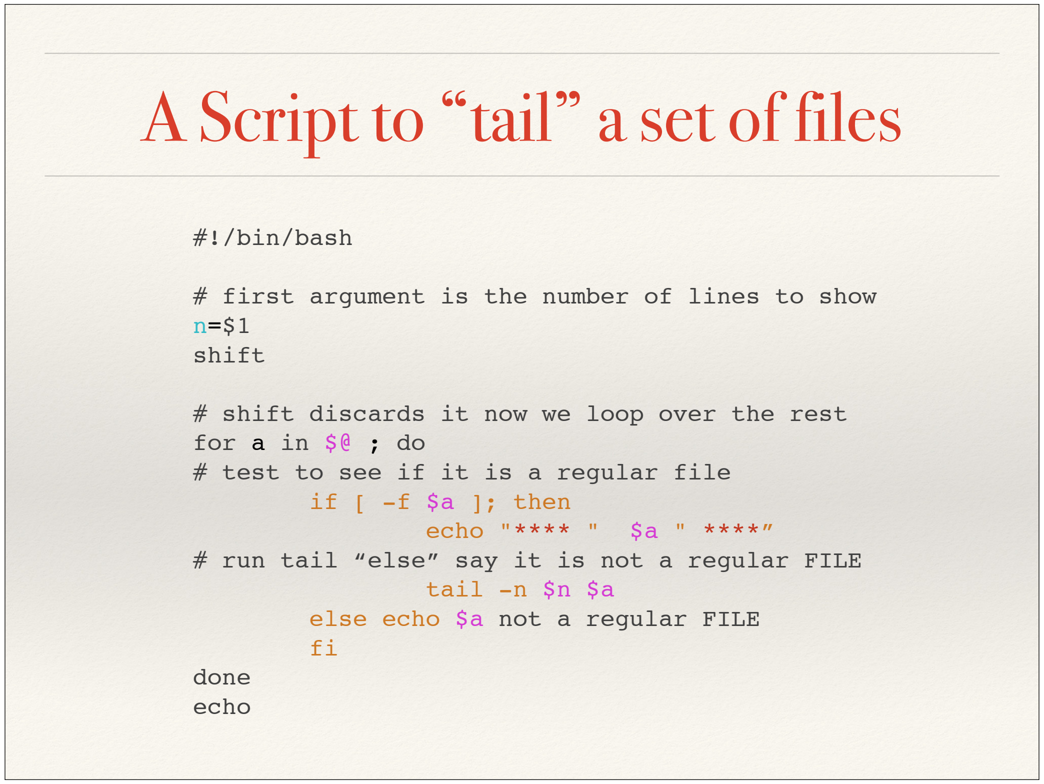


Command Line Arguments

```
[tkaiser@mio001 ~]$ ./cla word1 word2 word3
    echo $1 $2 $3
word1 word2 word3

    echo ${args[0]} ${args[1]} ${args[2]}
word1 word2 word3

    echo $@
word1 word2 word3  -> echo $@

    echo Number of arguments passed: $#
Number of arguments passed: 3

using for
for word1
for word2
for word3

using shift
word1
word2
word3
[tkaiser@mio001 ~]$
```

```bash
#!/bin/bash
# use predefined variables to access passed arguments
# echo arguments to the shell
echo '    echo $1 $2 $3'
echo $1 $2 $3
echo
# We can also store arguments from bash command line in special array
args=("$@")
#echo arguments to the shell
echo '    echo ${args[0]} ${args[1]} ${args[2]}'
echo ${args[0]} ${args[1]} ${args[2]}
echo
#use $@ to print out all arguments at once
echo '    echo $@'
echo $@ ' -> echo $@'
echo
# use $# variable to print out
# number of arguments passed to the bash script
echo '    echo Number of arguments passed: $# '
echo Number of arguments passed: $#
echo

echo using for
for a in $@ ; do
  echo "for" $a
done
echo
#this prints all arguments
echo using shift
while test $# -gt 0
do
    echo $1
    shift
done
```

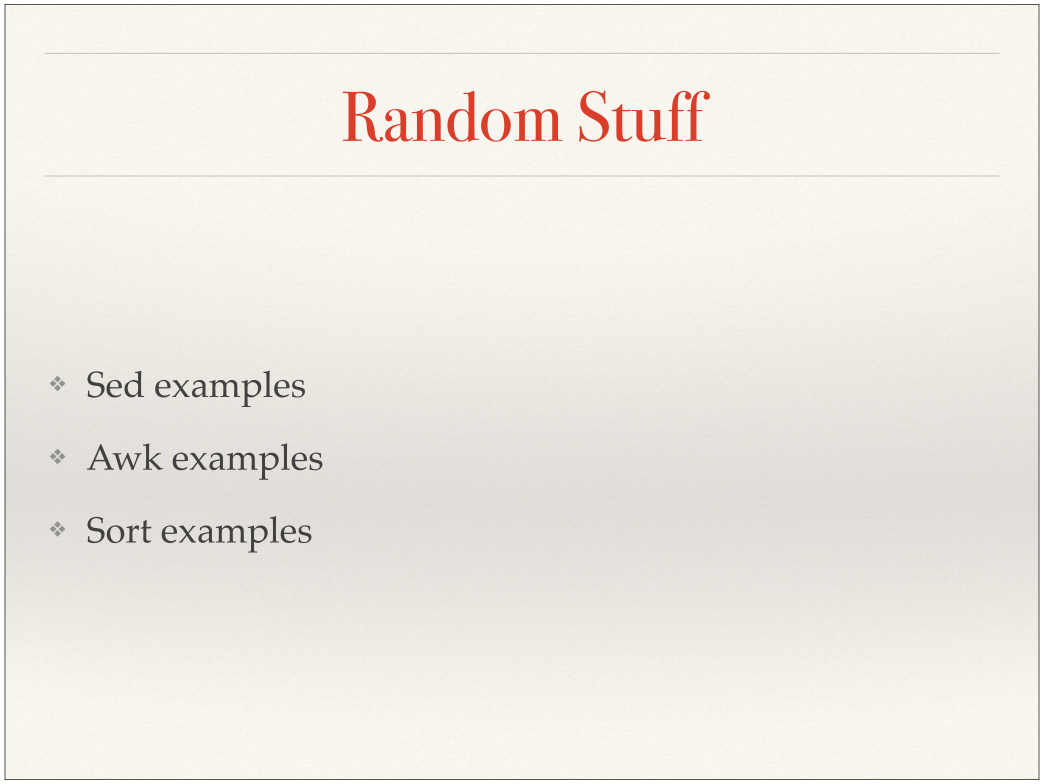
Here we show how to access command line arguments that are passed to a script. We can echo them by giving the number of the argument preceded with a dollar sign. We can store them in and array for later access. We can print all of the arguments by using the dollar sign followed by the — at — symbol. We can get the number of arguments or print tem in a for loop. The shift command strips off the leftmost argument in the command line so we can iterate over the arguments stripping off one at a time.



A Script to “tail” a set of files

```
#!/bin/bash

# first argument is the number of lines to show
n=$1
shift

# shift discards it now we loop over the rest
for a in $@ ; do
# test to see if it is a regular file
        if [ -f $a ]; then
                echo "**** "  $a " ****”
# run tail “else” say it is not a regular FILE
                tail -n $n $a
        else echo $a not a regular FILE
        fi
done
echo
```

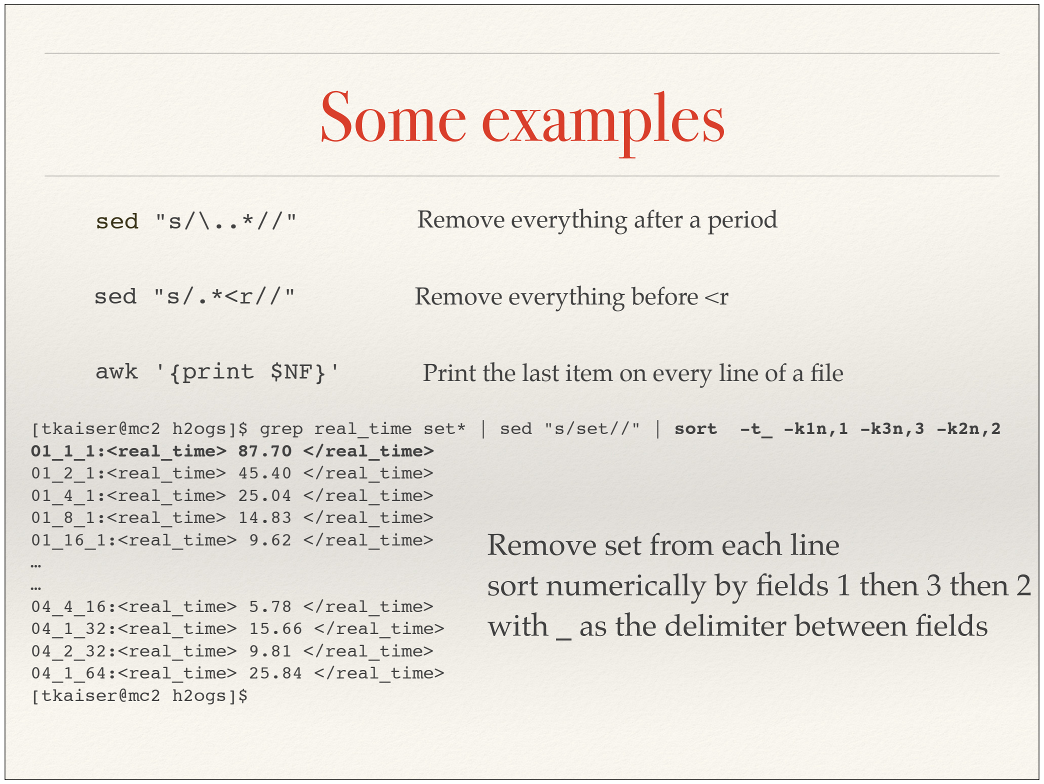
He we have a simple script that uses command line arguments. It does a tail of every file listed on the command line, assuming that the first argument is the number of lines to print. This is stripped off using the shift command and then we loop over the rest of the arguments for the tail command.



Random Stuff

- Sed examples
- Awk examples
- Sort examples

The sed, awk, and sort commands are very useful but the syntax can be challenging . We have some examples that show some common difficult tasks to do with these commands.



Some examples

```
sed "s/\..*//"
```
Remove everything after a period

```
sed "s/.*<r//"
```
Remove everything before <r

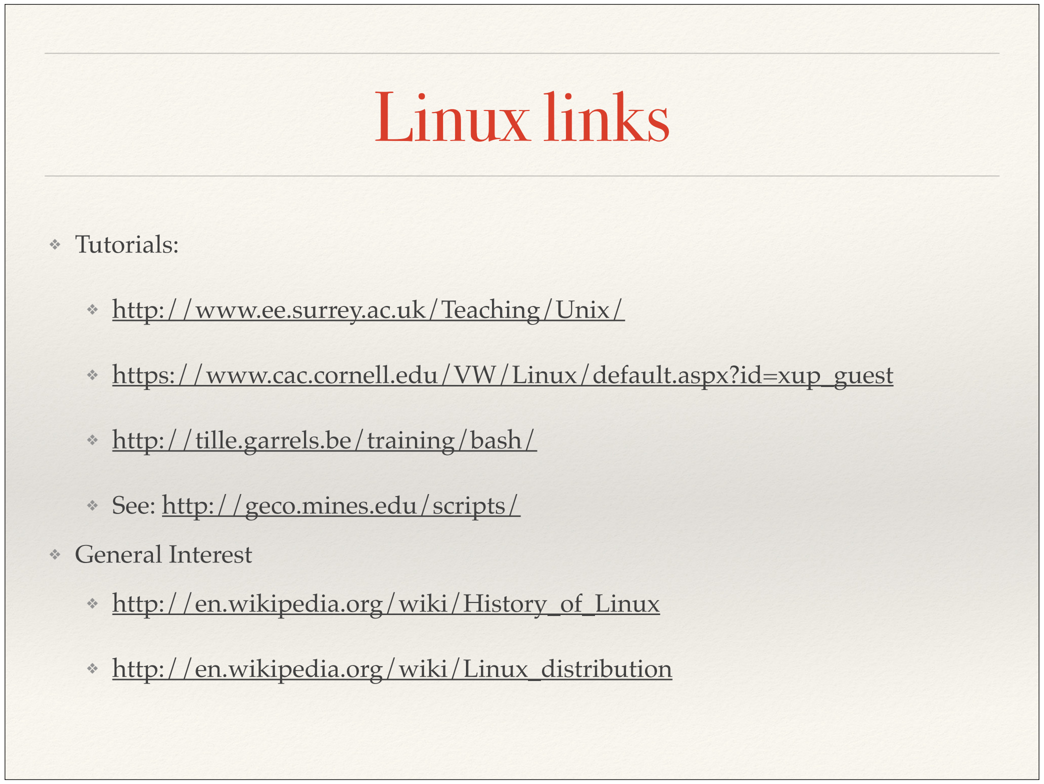
```
awk '{print $NF}'
```
Print the last item on every line of a file

```
[tkaiser@mc2 h2ogs]$ grep real_time set* | sed "s/set//" | sort  -t_ -k1n,1 -k3n,3 -k2n,2
01_1_1:<real_time> 87.70 </real_time>
01_2_1:<real_time> 45.40 </real_time>
01_4_1:<real_time> 25.04 </real_time>
01_8_1:<real_time> 14.83 </real_time>
01_16_1:<real_time> 9.62 </real_time>
…
…
04_4_16:<real_time> 5.78 </real_time>
04_1_32:<real_time> 15.66 </real_time>
04_2_32:<real_time> 9.81 </real_time>
04_1_64:<real_time> 25.84 </real_time>
[tkaiser@mc2 h2ogs]$
```

Remove set from each line
sort numerically by fields 1 then 3 then 2
with _ as the delimiter between fields

With the first three examples we would normally run this command with input being piped in, that is, running some other command first and using there vertical bar between the command and the sed or awk command. The first sed command shows how to remove everything after a period in the input steam. Then we remove everything before the less than symbol followed by r. The awk command show how to print the last item on the line of each file. Finally we have the grep command along with the sed command piped into sort to post process a file.

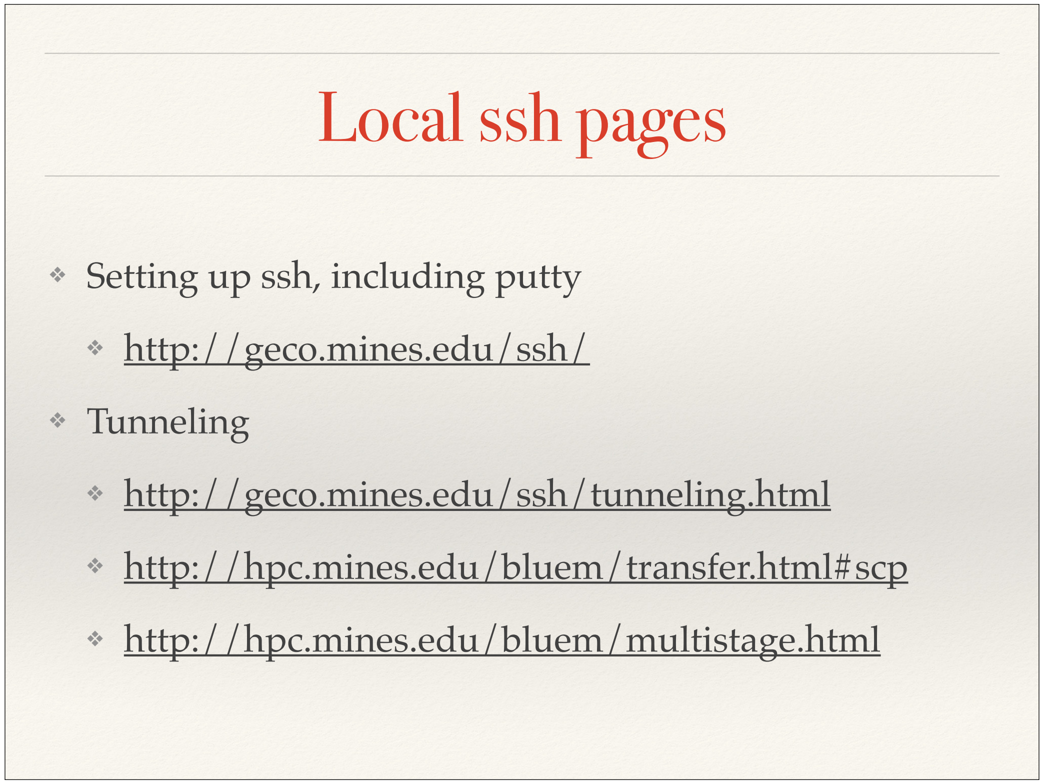
We use grep to pull out every line that contains the word “set” we then use sed to remove “set” from each line. Then we sort the output numerically based on three values. The values are separated by the underscore character as indicated by the minus t option. We sort first by the values in column 1 then by the values in column 3 and finally by the values in column 2.



Linux links

- Tutorials:
  - http://www.ee.surrey.ac.uk/Teaching/Unix/
  - https://www.cac.cornell.edu/VW/Linux/default.aspx?id=xup_guest
  - http://tille.garrels.be/training/bash/
  - See: http://geco.mines.edu/scripts/
- General Interest
  - http://en.wikipedia.org/wiki/History_of_Linux
  - http://en.wikipedia.org/wiki/Linux_distribution

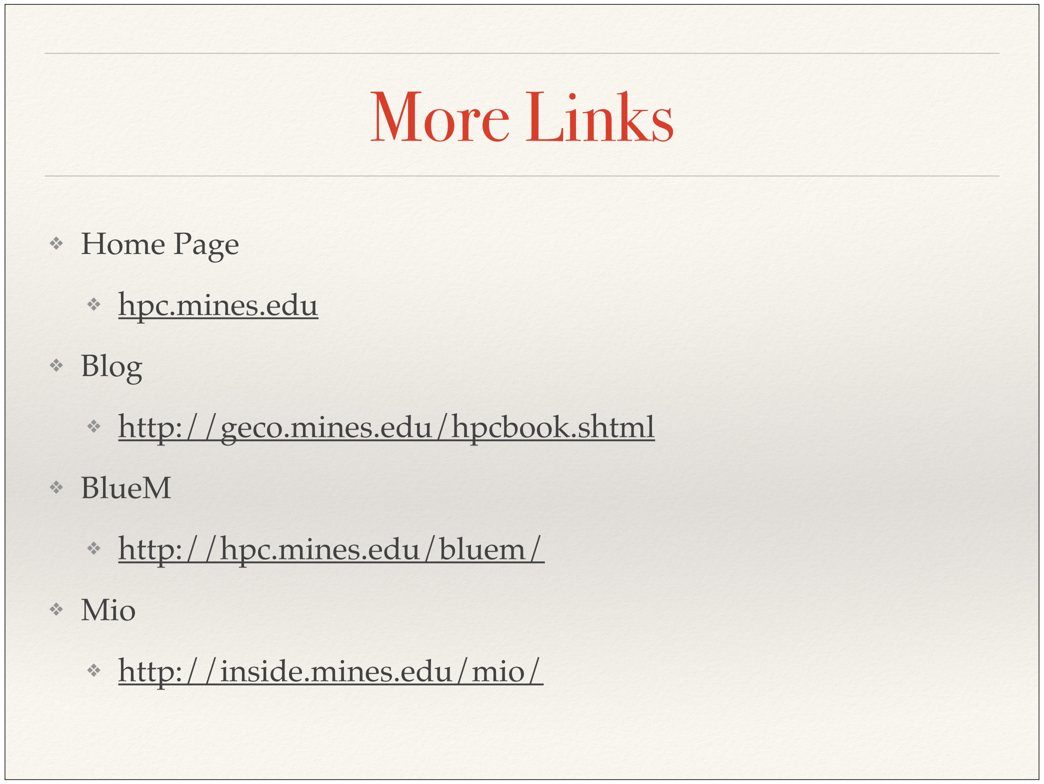
That's about it for toady. There are lots of things to learn. Here are some good web pages to look at.



Local ssh pages

- Setting up ssh, including putty
  - http://geco.mines.edu/ssh/
- Tunneling
  - http://geco.mines.edu/ssh/tunneling.html
  - http://hpc.mines.edu/bluem/transfer.html#scp
  - http://hpc.mines.edu/bluem/multistage.html

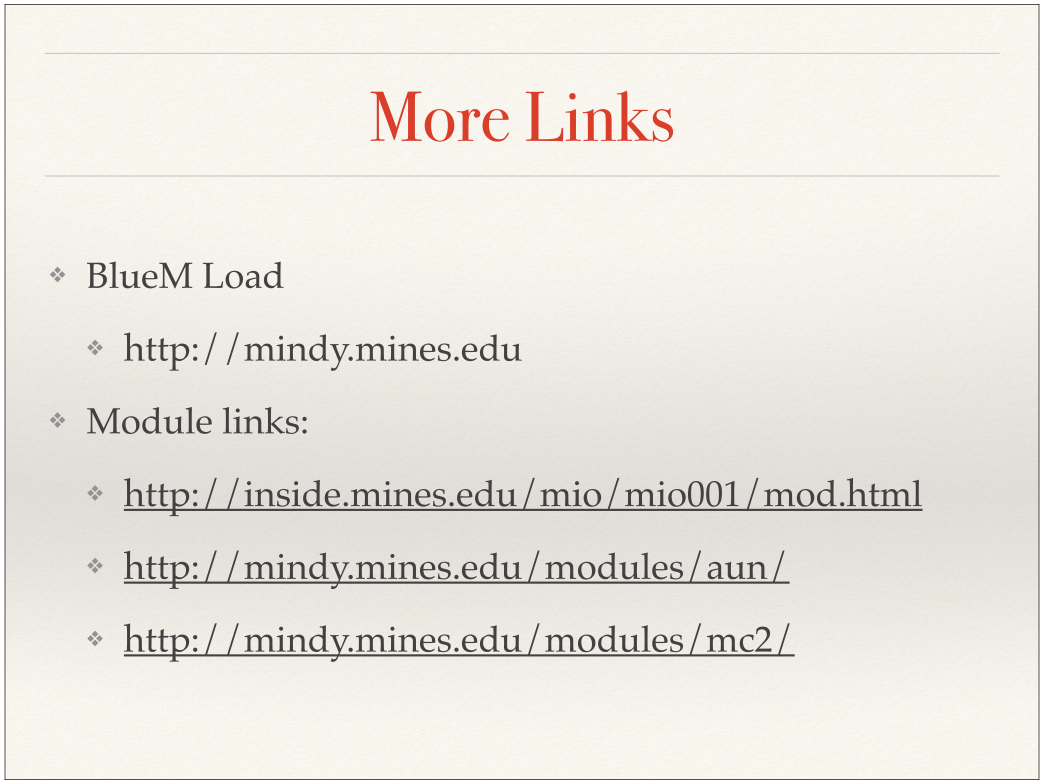
We have some local pages that talk about ssh .



More Links

- Home Page
  - hpc.mines.edu
- Blog
  - http://geco.mines.edu/hpcbook.shtml
- BlueM
  - http://hpc.mines.edu/bluem/
- Mio
  - http://inside.mines.edu/mio/

Here are some links to our HPC pages, including our blog and machine specific pages.



More Links

- BlueM Load
  - http://mindy.mines.edu
- Module links:
  - http://inside.mines.edu/mio/mio001/mod.html
  - http://mindy.mines.edu/modules/aun/
  - http://mindy.mines.edu/modules/mc2/

Here are some more local links.