A crash course in unix

mimi, verica, brian, chris

Introduction

By the end of this tutorial you will know what unix is and how to:

- log in and out of your account
- move through unix's directories
- manipulate files and directories
- use common programs to speed up your work
- print a file
- customize your environment
- log in to a remote computer

This is not a comprehensive course, but a beginning, a glimpse into the world of unix.

What is unix?

Unix is a circa 1970 operating system (OS)

- BUT, it has evolved to stay modern.
- Widely used by technical types (engineers, scientists)
- **OS**: the way computers know what to do AND how users interact with the hardware
- unix is fun!

Unix is used instead of MS Windows.

Linux and Mac OSX are Unix-based

What unix is NOT

Unix is not a programing language.

Unix is not a program. (It is many programs.)

Unix is not hard to understand or use.

You don't open unix.

Logging in

“logging in” starts a session under your username.

- To log in to the system: enter your **username**
- enter your **password**
- WELCOME TO UNIX

open a **terminal** window

- where you’ll enter **commands** using a shell
- **shell**: translates commands for **kernel**
- **kernel**: core of the OS, where the instructions are translated so the hardware can understand what to do.

When you log in and open a terminal, it looks for resource files, more on this later.

Try a command

Copy some stuff for later:

```
$ cp ../verica/dmy/usefulaliases .
$ cp ../verica/dmy/scriptingexample.csh .
```

Not impressed? Confused? Just wait!

The `$` symbol is used to represent the unix prompt.

`courier font` is used when we type unix commands. You should be able to type the same thing at your prompt.

Logging out

learn to leave:

- at the prompt, type `exit`
- That exits from the terminal...
- now either open a new terminal
- or log in again if you are completely out.

Always use `exit` to quit a session. Avoid closing an active window.

Every new terminal window you open starts a new "session." You can have many sessions open at once.

Unix commands

commands tell the computer what to do

for example, `cp` copies a file

- most direct method:

  $ commandname arguments

more generally, unix commands have the structure:

  $ commandname -[options] {arguments}

  commandname -- the command (cp, ls, etc)

  -[options] -- options to change behavior, usually 1-2 characters

  {arguments} -- the input/output/instructions the command needs to run

- example 1:

```
$ cp -r ../verica/dmy/ .
```

  `-r` option

  argument 1: “target files,” ../verica/dmy/

  argument 2: “destination” (dot means pwd)

- example 2:

```
$ cp dmy/test* .
```

Unix commands: getting help

`man`: One way to see the options of a command is to use the manual, or “man page”

- `$ man ls`
- Now we can use a few options with `ls`:

```
$ ls
$ ls -a
$ ls -a test*
$ ls -ar
```

sometimes you can use multiple options

Web searches like Google are great resources for getting help with commands.

Unix commands: redirection

PIPELINES

"pipe" output from one command to another.

- example: send output of ncdump to a **pagination** program (`less`)
- `$ ncdump -c testfile.nc | less`
- the "pipe" symbol redirects the output to less
- multiple pipes can be strung together (pipeline)

Other kinds of redirection

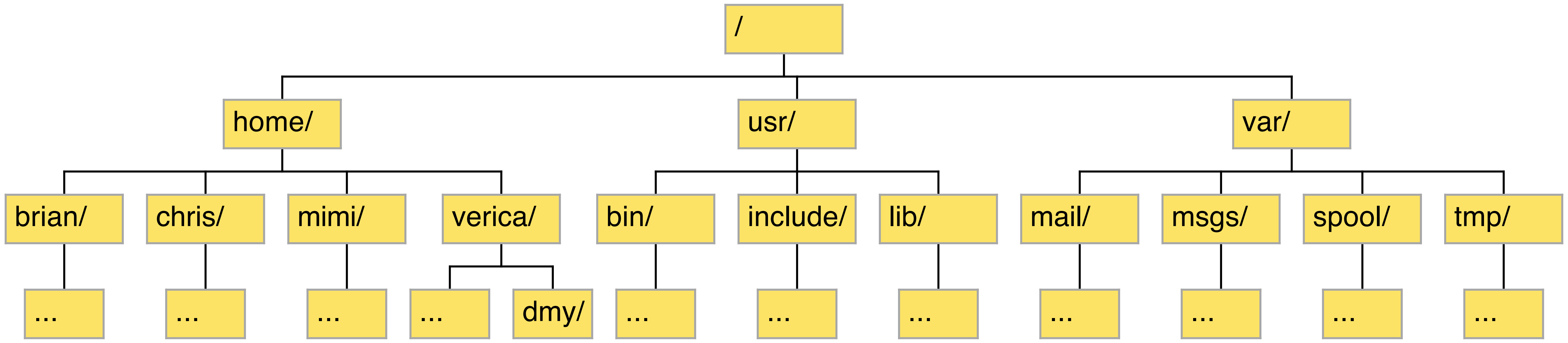
- put output of command into a file: `command > filename`
- append output to end of file: `command >> filename`
- give command input: `command < filename`
- examples will follow later!!

Don't be overwhelmed, redirection is just for convenience!

ADVANCED: Try looking at the `tee` command.

Directory Structure
Directory/File Manipulation

Directory tree



Translate to MS Windows:
directories = folders
cd/ls/grep = Windows Explorer

Commands

`pwd`	path of working directory
`ls`	list
`cd`	change directory
`cp`	copy
`rm`	remove
`mv`	move (rename)
`mkdir`	make directory
`rmdir`	remove directory
`cat`	concatenate and print files
`more / less`	pagination - display text to screen
`head / tail`	display beginning / end of file
`grep`	search for text
`find`	search for path
`chmod`	change mode of permissions
`du`	disk usage

Where? What? Move!

`pwd`: path of working directory

check where you are:

```
$ pwd
```

`ls`: list

see the contents of the current directory:

```
$ ls
```

`cd`: change directory

go to “parent directory:”

```
$ cd ..
```

check where you are now:

```
$ pwd
```

go back to your home directory:

```
$ cd your_user_name
```

Create. Destroy.

`mkdir` - make directory

make directory named temp:

```
$ mkdir temp
```

check what you have in the directory now:

```
$ ls
```

`rmdir` - remove directory

remove directory temp:

```
$ rmdir temp
```

check if temp is there:

```
$ ls
```

Copy directory and move in

`cp` - copy

copy directory named homeworks from my dmy directory to a directory called projects:

```
$ cp -r /home/verica/dmy/homeworks dmy/projects
```

`ls` - list

list what is in the dmy without entering into it

```
$ ls dmy
```

`cd` - change directory

move into tmp in dmy

```
$ cd dmy/tmp
```

`ls` - list

- list what's in the directory dmy/tmp
```
$ ls
```

- list in a long format to get a bunch of information that you might find useful at some point (file mode, number of links, owner name, group name, number of bytes in the file, date and time file was last modified, and the path name)
```
$ ls -l
```

- list also files/directories with names that start with dot (.)
```
$ ls -a
```

- list both in a long format and files with names starting with .
```
$ ls -la
```

Copy, rename, remove files

- list the content of the directory
  ```
  $ ls
  ```

`cp` - copy

- copy file named dmyfile to file named newfile
  ```
  $ cp dmyfile newfile
  ```
- list again
  ```
  $ ls
  ```

`mv` - move/rename

- rename dmyfile into newerfile
  ```
  $ mv dmyfile newerfile
  ```
- list again
  ```
  $ ls
  ```

`rm` - remove (delete)

- delete newfile
  ```
  $ rm newfile
  ```
- list again
  ```
  $ ls
  ```

Wild cards

```
$ cd tmp
```

`*` - exchanges any series of letters or numbers, including nonexisting ones

- list all files/directories with names starting with file

```
$ ls file*
```

`?` - exchange only one letter or number

- list all files/directories with names starting with file and having one more character at the end:

```
$ ls file?
```

`[]` - exchange given range of letters or numbers

- list files with names that start with file, finish with 0 and have 1, 2 or 3 in between

```
$ ls file[1-3]0
```

WARNING: never type `rm *`

Text files

`cat` - concatenate files

```
$ less file1
```
(`q` to quit)

```
$ less file2
```
(`q` to quit)

- concatenate these files to file catfiles:
  ```
  $ cat file[12] > catfiles
  ```

`more` and `less` - display file on the screen, but `less` is more powerful, as it allows going back to the beginning of the file, searching through the file …

- see again what is in the file catfiles:
  ```
  $ less catfiles
  ```

`head` - displays only the beginning of the file

`tail` - displays only the end of the file (`-f` is an useful option, as it keeps displaying the end of the file even when the file is still changing)

To create and modify longer and more meaningful files you need to use text editors, like emacs or vi or one of gazillion others

Search

`grep` - searches for the text either in the files or in the output of some command

- look for files that contain 3 in their text:

  ```
  $ grep 3 ./*
  ```

- look for files that contain 3 in their names:

  ```
  $ ls ./file* | grep 3
  ```

`find` - searches for the location of a file with prescribed parameters

- look for a file in a dmy directory that contains e in its name:

  ```
  $ find ~/dmy/ -name "*e*"
  ```

Ownership, permissions and space

- list the files in current directory to see who is owner and what are the permissions:
  ```
  $ ls -l
  ```

`chmod` - change mode of permissions

- set the your rights as a user to be read, write and execute, your group's rights to be read and execute and for others to only be able to read files named file?:
  ```
  $ chmod u=rwx,g=rx,o=r file?
  ```

`du` - disk usage

- check how much space (in KB) is taken by your current directory:
  ```
  $ du -ks
  ```

Applications & Shells

Processes and Programs

`$ ls`

- what you see is files and directories

`$ top`

- you see the columns 'COMMAND', 'CPU', 'USER' plus a few others. This command tells you what processes and programs are running on the machine you're on (COMMAND), how much of the CPU they're using, and who's running them (USER).

    For example, one of the processes running right now is 'top', and you are the user running it

    To end 'top':

    - `$[ctrl-c]`
    - This stops the process active in a terminal window.

`ps` is similar to `top`, except that it shows less information, it doesn't remain active, and it only shows you what's running in one terminal.

So how do I start programs?

Generally, the program name will be in your path or aliased to a command that will open the program (more on paths and aliases later!)

You can use Firefox to surf the Web. Let's try opening it now...

```
$ firefox &
```

It should open in a window.

You can use `kill` to close a misbehaving program.

`$ ps` shows you the process id (PIDs) of programs

`$ kill [-9] (PID)` to close emacs (fill in PID from prev. step)

Forgot the name?
Try `ctrl-d`!
`$ fir[ctrl-d]`

Forgot the "&"?
Try `ctrl-z/bg/fg`

SOME USEFUL PROGRAMS:

Data processing: Matlab, NCL, IDL

Word processing/presentation software/spreadsheets: LaTeX, Bibtex

Text Editors: vi, emacs, nedit

Imaging: GIMP, ImageMagick

Misc: Acroread (pdfs), Gaim (IM), Firefox

Printing

You can print a text file or a PostScript file (“ps”) by sending it directly to the printer with

```
$ lpr filename.ps
```

- Some useful flags with lpr:
  - `-P printername` sends to specified printer
  - `-# NUM` prints NUM copies of the file
  - `-K2` prints two-sided, book format

You can also use `convert` to change an image file or pdf to a ps, which you can print with `lpr`.

```
$ convert tempfile.pdf tempfile.ps
```

What does “print” really mean?
It means writing output... either to the screen (standard output) or to a printer.

Don’t try:
`lpr file.pdf` it **will print GIBBERISH!!!**
Also, other file exentions (gif, jpg) will not print with lpr.

Let's talk about shells…

What is a shell?

- The shell is what you've been using to issue commands and open programs. It's the user interface of Unix.
- Comes in different varieties: right now you're using TENEX C-shell (tcsh). Other options include bash (Bourne again shell), sh (original Bourne shell). You can switch between shells within a session, or change your default. Syntax of the commands is the primary difference between shells.
- **`$ echo $SHELL`**

  shows you which shell you're using (will say /bin/tcsh)

Try using autocomplete (available on most shells) by pressing TAB after typing a few letters.

Let’s talk about shells… part deux

You can use commands specific to your shell to set up your working environment.

`$ cd` (go back to your home directory)

```
$ ls -a
```

The file ‘.cshrc’ is the configuration file for the tcsh. You can modify this file to add things that you like!

Let’s take a look:

```
$ less .cshrc (q to quit less)
```

The file has commands separated into three categories:

- **environmental settings** set environmental variables and libraries used by some programs
- **command paths** tell the shell where to look for executables
- **aliases** let you make short cuts for commands

Customizing your .cshrc

Let's add a few useful aliases to your .cshrc file.

The file 'usefulaliases' has some useful aliases in it. Take a look:

`$ less usefulaliases` (`q` to quit `less`)

`$ cp .cshrc cshrc.backup`

(make a backup of your original .cshrc file, just in case)

`$ cat usefulaliases >> .cshrc` will append usefulaliases to your .cshrc file.

`$ source .cshrc` updates the commands in your terminal. You only need to do this if you add aliases/paths/etc after you open a shell.

`$ ll` (one of the aliases we just added: now is the same as "`ls -l`": lists the contents of the folder in long format.)

You can check to see if an alias exists and what it's aliased to by typing `which CMD` where CMD is the command you're searching.

Example: `$ which ll`

Shell scripting

One extremely powerful property of the unix environment is the ability to put a list of commands into a file, and run the list like a program.

`$ less scriptingexample.csh` (an example tcsh script)

'scriptingexample.csh' takes two arguments as input, and creates a file called outputfile that includes those arguments in a sentence.

Before you can use this file like a program, you have to make it executable:

`$ chmod a+x scriptingexample.csh`

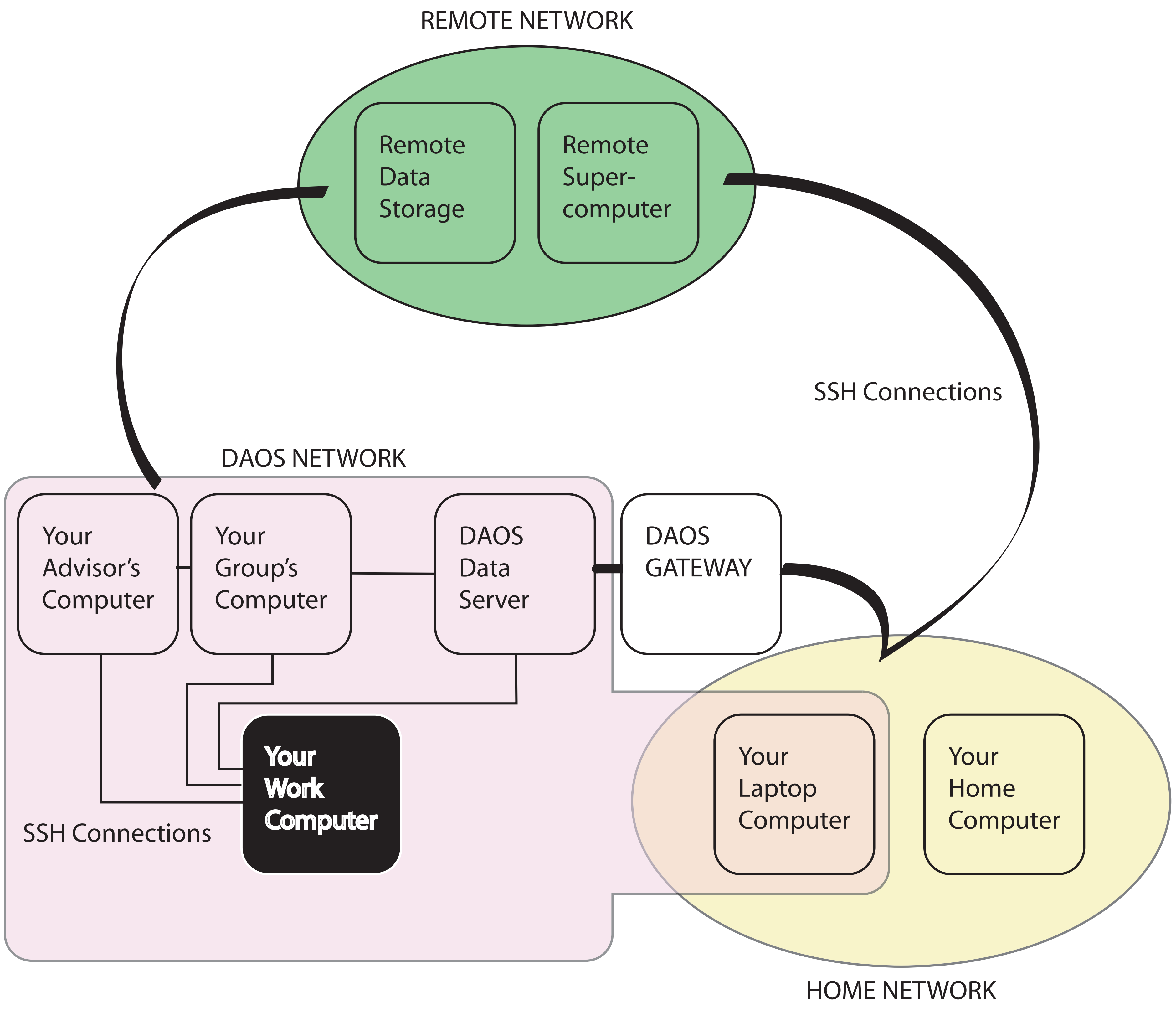
`$ ./scriptingexample.csh unix fantastic` runs this 'program'

`$ less outputfile` to look at the file it created.

Shell scripts are especially useful for repetitive jobs. Examples include: renaming files, repeatedly moving files from one directory to another, simple data analysis jobs, automation of things like conversions (from ps to jpg, e.g,), or for shortcuts for systematically editing files

Remote Sessions

Remote sessions



Remote sessions - why?

Resources (CPU, storage, data, collaboration, etc)

EXAMPLES: *task (resource)*

- access reanalysis data stored in Boulder, CO (storage/data)
- run a model and do analysis simultaneously (CPU)
- You are at a conference:
  - check your email (convenience)
  - give a file to a colleague (collaboration/storage)
  - access data on your work computer (data)
- Industrial espionage. ($$$)

Remote sessions: `ssh`

Always use secure methods!!

- OpenSSH, or as you will know it, `ssh` ("secure shell")

connect to a computer called fifi:

```
$ ssh username@fifi.atmos.ucla.edu
```

- enter password to access fifi
- now you should be at a unix prompt on fifi, it will use the resource file in your home directory on fifi. Try to look at that:

```
fifi$ less .bashrc
```

What if you want a graphical interface? Use the `-X`/`-Y` option:

```
$ ssh -Y username@fifi.atmos.ucla.edu
```

- open a new xterm: `fifi$ xterm &`

ssh is often used as a verb: "just ssh to bluesky to run the job."

Your computer keeps a record of all the hosts you connect to with ssh. Take a look at ~/.ssh/known_hosts

Remote sessions: `scp`

Copy a file from one computer to another:

use "secure copy:"

```
$ scp username@fifi.atmos.ucla.edu:~/../
chollow/.bashrc ~/temporaryitems/chris_bashrc
```

Send a file to a different computer:

```
$ scp testfile.nc
username@fifi.atmos.ucla.edu:~/dmy/tmp/
test_send.nc
```

`scp` is just like `cp`, and will be your primary method, but sometimes you need to send many files, or you have to do more complicated sets of copies, for that use `sftp`...

scp can be used recursively ("-r") to copy a directory structure, or with wildcards to copy all files that match a pattern.

when using wildcards on the remote computer, you need to enclose the argument in quote marks

sometimes you will use "anonymous ftp" sites. Then you do not supply a username. Some sites will use "sftp" (secure version)

Remote sessions: `sftp`

Transfer files between the computers:
use “secure file transfer protocol”

First, log on to the remote machine:

```
$ sftp air.atmos.ucla.edu
```

- use `ls` and `cd` to navigate the remote directories, and `lls` and `lcd` for the local ones.
- use `get` to copy files from the remote machine to the local machine, and `put` to copy from the local machine to the remote one:

```
sftp> get testdata/test_air.txt
sftp> put Pictures/pic_fifi.txt
```

When you are done, log out of the remote machine:

```
sftp> exit
```

you may often use sftp to transfer data to and from the machine “sftp.atmos.ucla.edu” (great name, huh). You can log in to sftp from an outside machine and “put” files there temporarily, and then log in from a local machine to “get” those files.

Wrap-up

We can help you, or Prashant Doma can help you, just ask.

This is available at http://www.atmos.ucla.edu/~xep

There is **A LOT** more to learn. Try looking at Harley Hahn's introductory book.

The internet has lots and lots and lots of unix resources.

GOOD LUCK!