

Data Analysis in Geophysics

ESCI 7205

Class 4

Bob Smalley

Basics of UNIX commands

Modify last command in history list using caret or circumflex accent, “^”, to fix typos or make small changes.

Replaces text inside first two carets with that between second and third.

(can sometimes skip closing caret as shown below in second example.)

```
$ ls trk1.kml
trk1.kml
$ ^1^2^
ls trk2.kml
trk2.kml
$ !!:p
ls trk2.kml
$ ^2^1
ls trk1.kml
trk1.kml
$
```

First it shows it to you and executes the edited command.

Environment (esoteric and essential)

Basics of the UNIX/Linux Environment

The UNIX Environment (general and CERI specific)

Mitch/Bob/Deshone have set up the basic CERI environment on both the Macs (new this year) and Suns so that everyone can access the standard UNIX tools and geophysics packages available on the UNIX systems at CERI.

The UNIX Environment

But what does this mean?

Many UNIX utilities, including the shell, need information about you and what you're doing in order to do a reasonable job.

What kinds of information?

Well, to start with, a lot of programs (particularly editors) need to know what kind of terminal you're using.

Your environment is composed of a number of

environment variables

which provide this important information to the
operating system.

Rather than forcing you to type this (hard to remember, where does one find it?) information with every command

such as (`% mail -editor vi -term aardvark48`)

UNIX uses *environment variables* to store information that you'd rather not worry about.

For example, the *TERM* environment variable tells programs what kind of terminal you're using. Any programs that care about your terminal type know (or ought to know) that they can read this variable, find your terminal type, and act accordingly.

UNIX commands receive information from three potential sources.

- Arguments on the command line

- Data coming down their standard input channel.

- The *environment*. When a command is started, it is sent a list of *environment variables* by the shell.

Since you generally want the computer to behave the same way everyday, these

environment variables

are setup and stored in

configuration files

that are accessed automatically at login.

What are your environment variables?

The commands `env`, or
`setenv` with no parameters,

print the current environment variables to the
Standard Out.

```
141:> env
```

```
USER=rsmalley
```

```
LOGNAME=rsmalley
```

```
HOME=/gaia/home/rsmalley
```

```
PATH=./gaia/home/rsmalley:/gaia/home/rsmalley/bin:/gaia/home/rsmalley/shells:/gaia/home/rsmalley/dem:/gaia/home/rsmalley/
```

```
defm:/gaia/home/rsmalley/defm/src:/gaia/home/rsmalley/
```

```
viscold_pollitz/viscoprogs_rs:/gaia/home/rsmalley/gg:/gaia/home/rsmalley/gg/com:/gaia/home/rsmalley/gg/gamit/bin:/gaia/home/
```

```
rsmalley/gg/kf/bin:/gaia/dunedain/d2/gps/bin:/gaia/smeagol/local/
```

```
passcal.2006/bin:/gaia/smeagol/local/gmt/GMT4.2.1/bin:/usr/sbin:/
```

```
usr/local/tex/bin/sparc-sun-solaris2.8:/gaia/home/rsmalley/
```

```
bin:/opt/local/sbin:/opt/sfw/bin:/usr/bin:/usr/ccs/bin:/usr/
```

```
local/bin:/opt/SUNWspr/SC5.0/bin:/opt/local/bin:/usr/bin:/usr/
```

```
dt/bin:/usr/openwin/bin:/bin:/usr/ucb:/gaia/smeagol/local/bin:/
```

```
net/gps4/d1/Noah/rbh/usr/PROGRAMS.330/bin:/gaia/home/rsmalley/X/
```

```
bin:/gaia/home/rsmalley/X/com:/gaia/home/rsmalley/record_reading/
```

```
bin:/gaia/home/rsmalley/record_reading/scripts
```

```
MAIL=/var/mail//rsmalley
```

```
SHELL=/usr/bin/tcsh
```

```
TZ=US/Central
```

```
LC_CTYPE=en_US.ISO8859-1
```

```
LC_COLLATE=en_US.ISO8859-1
```

```
LC_TIME=en_US.ISO8859-1
LC_NUMERIC=en_US.ISO8859-1
LC_MONETARY=en_US.ISO8859-1
LC_MESSAGES=C
SSH_CLIENT=75.66.47.230 50561 22
SSH_CONNECTION=75.66.47.230 50561 141.225.157.63 22
SSH_TTY=/dev/pts/12
TERM=xterm
HOSTTYPE=sun4
VENDOR=sun
OSTYPE=solaris
MACHTYPE=sparc
SHLVL=1
PWD=/gaia/home/rsmalley
GROUP=user
HOST=alpaca.ceri.memphis.edu
REMOTEHOST=c-75-66-47-230.hsd1.tn.comcast.net
MANPATH=/gaia/smeagol/local/passcal.2006/man:/gaia/smeagol/local/gmt/GMT4.2.1/man:/ceri/local/man:/usr/dt/man:/usr/man:/usr/openwin/share/man:/usr/local/man:/opt/SUNWspro/man:/opt/sfw/man:/usr/local/texmf/man:/gaia/smeagol/local/man
LD_LIBRARY_PATH=/gaia/smeagol/local/gmt/lib:/gaia/opt/SUNWspro/lib:/gaia/opt/SUNWspro/SC5.0/lib:/usr/lib:/usr/openwin/lib
```

```
LM_LICENSE_FILE=/gaia/opt/licenses/licenses_combined  
EDITOR=vi  
AB2_DEFAULTSERVER=http://stilgar.ceri.memphis.edu:8888  
PRINTER=3892
```

You get all the stuff shown so far automatically.

If you can figure it out, you can change it to suit yourself.

(But when you break-it, don't ask [or humbly ask] the system managers for help. If you were smart enough to break it, you're smart enough to fix it.)

```
GMTHOME=/gaia/smeagol/local/gmt/GMT4.2.1
NETCDFHOME=/gaia/smeagol/local/gmt
GMT_GRIDDIR=/gaia/smeagol/local/gmt/GMT4.2.1/share/dbase
GMT_IMGDIR=/gaia/smeagol/local/gmt/GMT4.2.1/DATA/img
GMT_DATADIR=/gaia/smeagol/local/gmt/GMT4.2.1/DATA/misc
CWD=/gaia/home/rsmalley
HELP_DIR=/gaia/home/rsmalley/gg/help/
INSTITUTE=uom
RECORD_READING=/gaia/home/rsmalley/record_reading
RECORD_READING_BIN=/gaia/home/rsmalley/record_reading/bin
RECORD_READING_SCR=/gaia/home/rsmalley/record_reading/scripts
RECORD_READING_SRC=/gaia/home/rsmalley/record_reading/src
latestrtvel=rtvel4_9305_5bv19
LATESTRTVEL=rtvel4_9305_5bv19
ANONFTP=/gaia/midtown/mid4/smalley/public_ftp
ANONFTP_IN=/gaia/midtown/mid4/smalley/public_ftpinbox
SACDIR=/gaia/tesuji/d1/local/sac
SACXWINDOWS=x11
SACAUX=/gaia/tesuji/d1/local/sac/aux
SACSUNWINDOWS=0
GPSHOME=/gaia/dunedain/d2/gps
```

Plus you can add our own stuff (above).

Unless you are running Linux (in which case you are the *system manager*), you can forget about setting up most of this as the *system managers* do it for you.

There are a few environment variables, however, that you need to know about and/or set up yourself.

HOME*

This environment variable controls what UNIX commands consider your (base) home directory.

This is how “cd” and “~” know which directory to refer to

```
% echo $HOME  
/gaia/home/rsmalley
```

To refer to the value of an environment variable put a \$ in front of the name.

*these environment variables should not be changed by the user

The \$ therefore has a special meaning to the shell.

(As do the characters “ ~, !, /, *, ?, ^, \ “
all of which we have already seen.

By the time we are done we will have used up most of the non alpha-numeric characters with special meanings.)

SHELL*

This variable stores your default shell

```
% echo
```

```
/usr/bin/tcsh
```

Seems pretty simple!

How to find your SHELL

```
$ echo $SHELL
```

```
/bin/bash
```

```
$ /bin/csh ←
```

Start csh

```
> echo $SHELL
```

```
/bin/bash ←
```

OOPS!

SHELL

How to really find your shell
Start bash (inside csh, inside bash)

```
$ echo $SHELL
/bin/bash
$ /bin/csh
> echo $SHELL
/bin/bash
> echo $0
/bin/csh
>
```

0 is the shell variable containing the name of the program that is running - the shell. The shell is just another program to UNIX. \$0 is value of shell variable 0).

SHELL

How to really find your shell

```
$ echo $SHELL
/bin/bash
$ /bin/csh
> echo $SHELL
/bin/bash
> echo $0
/bin/csh
> ps -p $$
PID TTY          TIME CMD
91456 ttys003      0:00.02 -bin/csh
> /bin/tcsh
>> echo $SHELL
/bin/bash
> echo $0
/bin/tcsh
> ps -p $$
PID TTY          TIME CMD
91467 ttys003      0:00.03 -bin/tcsh
> exit
exit
> exit
exit
$ echo $0
-bash
```

\$ is the shell variable containing the process id (pid), \$\$ is value of shell variable \$ (very UNIX).

What happens if we enter `$SHELL` all by itself?

`$ $SHELL`

The shell sees

```
$ /bin/bash
```

Since a shell variable is just a character string, it replaces the `$SHELL` with the character string.

So if the shell variable is a command or otherwise interpretable by the shell it will try to do it.

Can also id the shell by the prompts (\$, >, etc., once you know which is which).

These examples also show that the shell is just another program – the only thing special about it is that it is the program that is started automatically for you when you login.

Finally, What is my shell?

This seems to be the best way to find out.

```
% echo $0
```

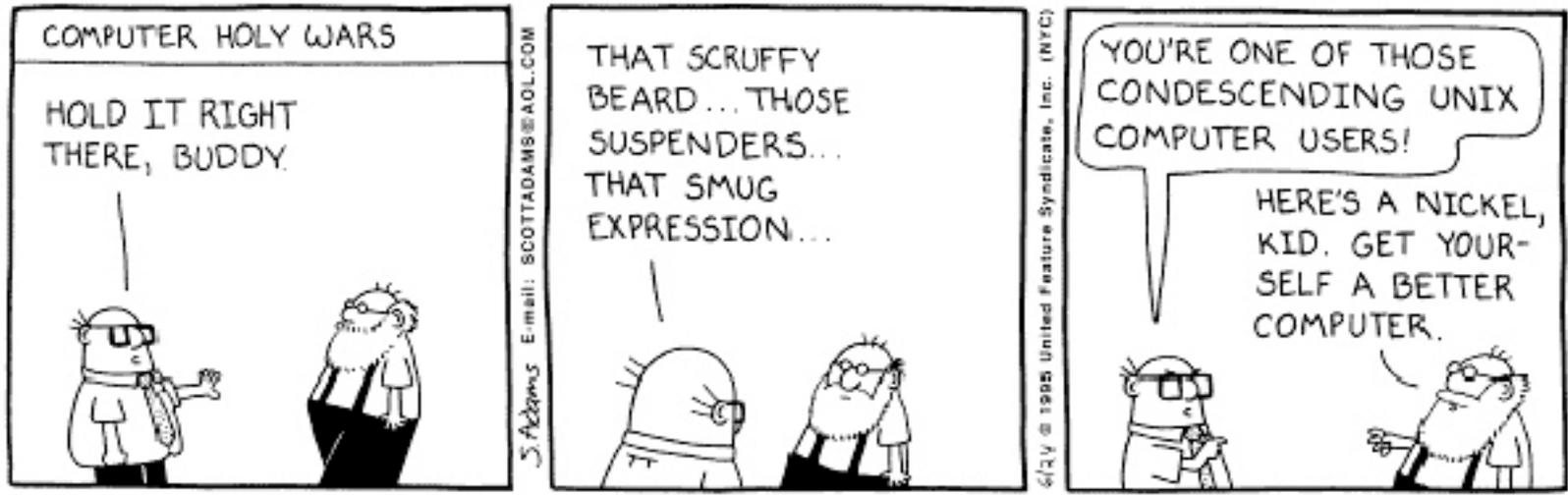
Works for csh, tcsh, sh, and bash.

(\$0 does not refer to the shell in general, this is one of the UNIX “standards” that \$0 is the program you are running!!

[which in this case is the shell – perfect UNIX logic]).

"people who have trouble with typing commands should not be using a computer."

Response of the UNIX community to criticism that UNIX ignored the needs of the unsophisticated user.



Environment variables are managed by your shell.

The difference between
environment variables
and regular
shell variables
is that

a shell variable is local to a particular instance of the shell (such as your current shell or a shell script), while environment variables are "inherited" by any program you start, including another shell.

That is, the new process gets its own copy of these variables, which it can read, modify, and pass on in turn to its own children.

In fact, every UNIX process (not just the shell) passes its environment variables to its child processes.

Example (very important) environment variable, what it is used for, and how to maintain it (you will probably need to do this at some point).

PATH

To see the value of the environment variable PATH, echo it to the screen.

PATH

This environment variable tells the shell where to find executable files

```
%echo $PATH
```

```
./:/gaia/home/rsmalley:/gaia/home/rsmalley/bin:/gaia/home/rsmalley/shells:/gaia/home/rsmalley/dem:/gaia/home/rsmalley/defm:/gaia/home/rsmalley/defm/src:/gaia/home/rsmalley/viscold_pollitz/viscoprogs_rs:/gaia/home/rsmalley/gg:/gaia/home/rsmalley/gg/com:/gaia/home/rsmalley/gg/gamit/bin:/gaia/home/rsmalley/gg/kf/bin:/gaia/dunedain/d2/gps/bin:/gaia/smeagol/local/passcal.2006/bin:/gaia/smeagol/local/gmt/GMT4.2.1/bin:/usr/sbin:/usr/local/tex/bin/sparc-sun-solaris2.8:/gaia/home/rsmalley/bin:/opt/local/sbin:/opt/sfw/bin:/usr/bin:/usr/ccs/bin:/usr/local/bin:/opt/SUNWspro/SC5.0/bin:/opt/local/bin:/usr/bin:/usr/dt/bin:/usr/openwin/bin:/bin:/usr/ucb:/gaia/smeagol/local/bin:/net/gps4/d1/Noah/rbh/usr/PROGRAMS.330/bin:/gaia/home/rsmalley/X/bin:/gaia/home/rsmalley/X/com:/gaia/home/rsmalley/record_reading/bin:/gaia/home/rsmalley/record_reading/scripts
```

The “:” is used to separate each full path name in sh, bash (space for csh, tcsh).

When you run a command (from the terminal or a shell script), your shell looks through each directory in your *PATH* variable, in order, until it finds the first instance of an executable file with the name of the command.

It then runs the command.

```
%echo $PATH
./:/gaia/home/rsmalley:/gaia/home/rsmalley/bin:/gaia/home/
rsmalley/shells:/gaia/home/rsmalley/dem:/gaia/home/rsmalley/
defm:/gaia/home/rsmalley/defm/src:/gaia/home/rsmalley/
viscold_pollitz/viscoprogs_rs: etc.
```

My path starts with dot (“.”).

This is convenient (or else you have to type the relative path `./myprog` to execute programs in the working directory) but this is considered a security weakness.

Next I have a number of my directories where I've written Fortran or C programs and shell scripts.

In the “standard” UNIX organization that you will see in most books, one is supposed to put all your executable programs in your

“~/bin” directory

and all your shell scripts in your

“~/scripts” directory.

You will probably not find many people (or systems) that do this anymore.

So how does this work?

If you are working a program to do least squares analysis and decide to call it "ls" what will happen when you enter the command "ls"?

It depends.

What happens depends on *your* path.

Remember that to UNIX, everything outside the kernel (including the shell) is just a file.

Some of these files are executable (programs).

When the shell goes looking through your path for an executable file (has to have executable set in file permissions) named “ls”, it will run the first one it finds.

If the directory containing your least squares program (executable file), “ls”, is in your path

Before

the directory containing the UNIX list command, “ls”, it will run your program and you will not be able (at least simply) to get a listing of your directory!

(How to solve this? Have to give full path to the system ls, /bin/ls for example. You need to know where the system ls lives.)

If the directory containing your least squares program, “ls”, is in your path

after

the directory containing the UNIX list command, “ls”, it will run the UNIX ls command and you will not be able (at least simply) to run your program!

(How to solve this? Give full path to your program ls, e.g. ~/myprogs/ls or relative path ./ls, ../myprogs/ls etc.

Can't solve this with the “\” we saw before since this undoes an alias. It does not change the path.

which

Command that shows what the shell finds for the command name.

```
$ which ls  
/bin/ls
```

Or if you have redefined `ls` and it is found in your path first

```
$ which ls  
/user/smalleby/in/ls
```

To run a specific executable file – give its full path.

```
$ /bin/ls
!          Public
Adobe SVG 3.0 Installer Log  Sites
Desktop          bin . . .
```

More examples.
Can use all the tricks in specifying paths.

Run from one directory up.

```
$ pwd
/Users/smalley/bin
$ ../hello.sh
Hello
$
```

If the file is in the working directory, and that directory is not in your path, use the dot.

```
$ ./hello.sh
Hello
$
```

This behavior is not a “bug”, it is considered to be desirable and an example of the POWER of UNIX.

(This is also where the security problem comes in when dot is in your path. If someone compromises your system and puts a malicious file in your directory with the name “1s” and you have dot in your path and don’t notice the file “1s” and enter “1s” to get a list, you execute the bad file instead.)

How you make your path is up to you.

Will see how to do it next.

Modifying your environment

Modifying your environment

If you mess up modifying the environment in your current window – you may “break” your current window (shell).

This is generally not a problem on the sun, mac, etc.

The environment is local to that window/shell.

Just close it and open another window.

How to change/set shell and environment variables.

In csh/tcsh use commands

set for regular (local) shell variables and
setenv for environment (global) variables.

```
set term = xterm  
setenv TERM = xterm
```

We already mentioned the difference between regular shell and environment variables.

(you have to know that xterm is something that the shell will understand.)

If you need to deal with this level of UNIX, go
find a wizard
(Bob Debula, Mitch Withers).

This syntax is also specific to csh/tcsh.

```
set term = xterm  
setenv TERM = xterm
```

(note that since UNIX is case sensitive this is two environment variables. A local one “term” and a global one “TERM”)

To do the same thing in bash.

```
term=xterm  
TERM=xterm
```

Note that there are NO SPACES on either side of the equals sign here.

How do we tell the difference between a regular shell variable and an environment variable in bash.

(there is really no difference within an instance of a shell).

(`set` with no parameters lists shell variables, `env` also lists environment variables. In `csh` - `setenv` with no parameters lists environment variables,)

In `sh`/`bash`, when we define a variable, it is a regular shell variable.

To make it an environment variable (one that is inherited) you export it.

```
term=xterm  
TERM=xterm  
EXPORT term  
EXPORT TERM
```

setenv:

The csh/tcsh command to change environment settings.

Can be run on the command line,
from within a local configuration file
(.cshrc or .login),
or in a shell script.

When run it without specifying an environment variable, it will print all environment variables to the screen

How to change/set your path in csh/tcsh.

```
% setenv PATH {$PATH}:/gaia/home/rsmalley/scripts
```

This adds the (text string that is the) name of the directory

```
' /gaia/home/rsmalley/scripts '
```

to the end of the current environment variable PATH associated with the active shell.

When UNIX starts, you automatically get a path environment variable (it may be, but probably is not, empty) and this is the best candidate for the one you will have to change.

The environment variable is just a text string.

The shell interprets it.

setenv:

```
% setenv PATH /gaia/home/rsmalley/scripts:{$PATH}
```

Operationally it adds the the directory

`/gaia/home/rsmalley/scripts`

to the path, this time at the beginning.

What are the braces “{“ “}” for?

They delimit the shell variable used with the \$. They are needed when characters that could be in a variable name follow it without a space.

```
SANJUAN=/volumes/seismicdata/panda/sanjuan
```

```
... ${SANJUAN}_disk1    OR    ...{$SANJUAN}_disk1
```

expands to `/volumes/seismicdata/panda/sanjuan_disk1`

while

```
... $SANJUAN_disk1
```

tries to expand a variable named `SANJUAN_disk1`

Which probably does not exist.

setenv:

```
% setenv PATH /gaia/home/rsmalley/scripts:{$PATH}
```

Note PATH is used twice. On the right with the \$ it refers to the current value of the environment variable.

On the left it refers to the name of the environment variable that is being set to the string on the right (including the old value).

setenv:

```
% setenv PATH /gaia/home/rsmalley/scripts:${SPATH}
```

In this case it will append the current value of PATH to the new information and put everything in a new version of PATH.

(sort of like $\text{vari}=\text{vari}+1$ in Fortran, Matlab, c, etc. This is not a mathematical algebraic equation.)

If you don't write any of your own programs (or always use the path to the program/file) you will not have to change your path from the default.

(The default path at CERl will give you the path to the tcsh (and other) shell(s), and the paths to the tools such as MATLAB, SAC GMT, and some others.)

Modifying your default environment.

We already saw that you can always change things in your current environment [and that of any new child process] using the setenv command.

But it will get old changing everything to the way you want it each time you log in/open a new window/start a new shell.

And this being UNIX, there is a (easy) way to set up your own personal environment.

Modifying your default environment.

The setup of your personal environment (personal changes/preferences for how you want the shell to work for you) in csh and tcsh is stored in the file named

.cshrc

(there is also a file .login, but it is not likely you will have to change it (it get's used when you log in, not each time you start a shell) – so I'll mention it for completeness, but let's ignore it.)

When to make *your own environment variables.*

Anytime *you want a global definition of something.*

```
417:> grep rtvel .cshrc  
setenv latestrtvel rtvel4_9305_5bv19  
setenv LATESTRTVEL $latestrtvel
```

Modifying your default environment variable
PATH using the .cshrc (.bashrc) file.

We are now doing brain surgery on ourselves.

In a mirror.

This is dangerous.

So--

Make a back up of the current, working .cshrc
(.bashrc) file before you change it.

Have a second terminal window open in case you mess your file up so completely and break your active window.

This way you have another window open to delete the offending file and restore things from the backup file. (Unless you run the command to change it in a window, the environment is static once a window is open.)

You want this window open BEFORE you make the change, as any window opened after the file is saved will use the modified, bad, .cshrc (.bashrc) file.

For your path, you will see something like this in your `.cshrc` file.

```
set path = ( . ~ ~/bin ~/shells ~/dem ~/defm ~/defm/src $path )
```

Which uses the set command (local) rather than the setenv command (global).

The man page for set says var = value set assigns value to var, where value is one of:

- word - A single word (or quoted string).
- (wordlist) - A space-separated list of words enclosed in parentheses.

Ex. using the command set with the environment variable path also sets the environment variable PATH (tcsh).

```
265:> set path = ( $path ~/ESCI7205 )
```

Now look at the environment variable PATH (using a script I wrote to put out each entry on a separate line)

```
266:> ExaminePath.sh
```

```
•  
/gaia/home/rsmalley  
/gaia/home/rsmalley/bin  
• • •  
  
/gaia/home/rsmalley/record_reading/scripts  
/gaia/home/rsmalley/ESCI7205
```

```
267:>
```

The ESCI7205 entry was not there before.

When you set path, it also changes PATH.
When you setenv PATH, it also changes path.
They seem to track.

I've not been able to find documentation on how
this works. (I think one is for sh/bash and one for csh/tcsh)

But this is what you will see in both the
universal .cshrc (/etc/.cshrc), and if you make
changes, in your own .cshrc file.

It has been copied down through the ages.

After changing your path in the current shell.

First see what your path is.

```
266:> ExaminePath.sh
```

```
•  
/gaia/home/rsmalley  
/gaia/home/rsmalley/bin  
• • •
```

```
/gaia/home/rsmalley/record_reading/scripts
```

```
267:>
```

.cshrc (csh resource script) configuration file (aka dot file)

```
setenv PATH ./gaia/home/rsmalley/bin:$PATH
setenv PATH ${PATH}:/gaia/home/rsmalley/record_reading/bin
setenv PATH ${PATH}:/gaia/home/rsmalley/record_reading/scripts
setenv PRINTER 3892
alias cd 'cd \!*;echo $cwd'
alias home "cd ~"
alias del 'rm -i'
set history=500
set ignoreeof
set savehist=500
set filec
```

.bashrc (bash resource script)
configuration file (aka dot file)

Slightly different

```
PATH=.:/Users/robertsmalley:/Users/robertsmalley/bin:$PATH
```

Or (usually – so children inherit it)

```
export PATH=.:/Users/robertsmalley:/Users/robertsmalley/bin:$PATH
```

```
export PATH=$PATH:/Users/robertsmalley/gamit_globk_10.4/com:/Users/robertsmalley/gamit_globk_10.4/gamit/bin:/Users/robertsmalley/gamit_globk_10.4/kf/bin
```

Once you have made changes to your `.cshrc` (`.bashrc`) (and saved them), which is just a file, how do you have them activated in your current window/shell?

(at this point they will be activated in any new shell/window/login)

You could log out and then log back in (not very efficient as you lose your history, but it works), or open a new window (ditto) and work there.

Use the source command with the .cshrc (.bashrc) file as input. (don't need the input redirect "<")

```
151:> source .cshrc
```

```
152:>
```

source: executes configuration files

If you change your configuration file, you will need to execute `source` in all open terminal windows for the changes to take effect. The changes automatically will take effect when new terminal windows/shells are opened.

Say you have edited the `.cshrc` file.

```
% nedit ~/.cshrc
```

```
% source ~/.cshrc
```

The default .cshrc file that everyone at CERl gets when they login, open a window, or start a shell is stored (on the SUN) in the file

/etc/.cshrc

And on the Mac in the file

/etc/csh.cshrc

After that the shell looks in your home directory for a .cshrc, which is used to expand upon and/or override the CERl values.

MANPATH*

Tells the shell where to find the manual pages read using the man command

```
%echo $MANPATH
```

```
/gaia/smeagol/local/passcal.2006/man:/gaia:smeagol/local/gmt/GMT4.2.1/man:/opt/local/man:/ceri/local/man:/usr/dt/man:/usr/man:/usr/openwin/share/man:/usr/local/man:/opt/SUNWspro/man:/opt/sfw/man:/usr/local/teTeX/man:/gaia/smeagol/local/man:/opt/csw/man
```

If you do a man on a command and the shell can't find a manual page (and you are sure the man page exists), this environment variable may not be set correctly.

HOST*: environment variable with the name of the machine you are currently logged into.

REMOTEHOST*: environment variable with the name of the machine you are sitting in front of, if different (e.g. you are in the class on a PC and have used the program ssh to log into a sun at CERI.).

```
161:> echo $HOST $REMOTEHOST  
alpaca.ceri.memphis.edu  
162:>
```

`SSH_CLIENT`: the IP (internet protocol) address and port of the `HOST` machine.

`SSH_CONNECTION`: the IP addresses and ports of the `HOST` machine and the `REMOTEHOST` machine.

```
162:> echo $SSH_CLIENT $SSH_CONNECTION
```

```
75.66.47.230 51704 22 75.66.47.230 51704 141.225.157.63 22
```

```
163:>
```

If you want to get as much info as you can about the IP addresses. (Can also put in the name and get the address.)

```
169:> nslookup 141.225.157.63
```

```
Server: dns1.memphis.edu
```

```
Address: 141.225.253.21
```

```
Name: alpaca.ceri.memphis.edu
```

```
Address: 141.225.157.63
```

```
170:> nslookup 75.66.47.230
```

```
Server: dns1.memphis.edu
```

```
Address: 141.225.253.21
```

```
Name: c-75-66-47-230.hsd1.tn.comcast.net
```

```
Address: 75.66.47.230
```

```
171:>
```

Aside ---

How to destroy your input data file and how to prevent it (i.e. accidentally doing it).

First – look at file.

```
262:> more flong.dat
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
6  
7  
8  
9  
10
```

```
263:>
```

Sort it, using the sort command.

```
263:> sort flong.dat
```

```
1  
10  
10  
2  
3  
4  
5  
6  
6  
7  
7  
8  
8  
9  
9
```

```
264:>
```

So far OK.

Say we want to save the sorted output to a file.
Use redirection.

```
264:> sort flong.dat > flong.dat  
265:> more flong.dat  
266:>
```

We just erased our file!

UNIX says we will need an output file, and (unless your sys admin has done the non-UNIX philosophy action of setting “no-clobber”) it has permission to clobber a pre-existing output file – so it does. It then goes looking for the input file, which it cannot find because it just erased it!!

(Notice that this is not consistent with the `PATH=$PATH:/mybin` or `x=x+1` model!)
(consistency is the hob-goblin of little minds)

It sorts nothing (the now empty input file) and puts it into the output file.

It sees no reason to complain, warn you, etc.

You are an adult, this behavior is “obvious” from the operating principles of UNIX.

Say we want to save the sorted output to a file.
Use redirection.

```
264:> sort flong.dat > flong.dat  
flong.dat: File exists.
```

UNIX says we will need an output file but *your sys admin* has done the non-UNIX philosophy action of defining “no-clobber” to protect *you* from yourself so UNIX cannot make the output file (since it cannot erase the pesky file with the same name).

Having no-clobber set prevents you from
inadvertently erasing existing files

It protects you from yourself.

Very non-UNIX philosophy.

Say we want to save the sorted output to a file.
Use redirection.

```
264:> sort flong.dat > flong.dat
flong.dat: File exists.
265:> sort flong.dat >! flong.dat
266:> more flong.dat
267:>
```

But if you insist, you can still erase your input file!

The `>!` says to redirect the output to the file `flong.dat` and clobber a file with that name if you need to (i.e. a file with that name exists).

Tricky distinctions in sh and bash

set is a shell command to set the value of a shell attribute variable; these are internal variables used by the shell program.

env is a program that runs another program with modified environment variables.

Tricky distinctions

The major difference is that the env command will never modify the shell's own environment (only that of the child process), while set will.

set can also change settings like brace expansion within the shell.

You might also want to look at export, which changes the environment variables for all future commands.

Tricky distinctions

Enter the commands set and env and look at the differences (it will help if you sort the output of env)

(e.g. - noclobber is a shell attribute variable, not an environment variable - it is “set” and “unset” using set)

How to set/clear noclobber

```
-bash 532 ~ # set -o | grep noclobber
noclobber          off
-bash 533 ~ # set -o noclobber
-bash 534 ~ # set -o | grep noclobber
noclobber          on
-bash 535 ~ # set +o noclobber
-bash 536 ~ # set -o | grep noclobber
noclobber          off
-bash 537 ~ #
```

Aliases

Basics of the UNIX/Linux Environment

Alias

The alias and unalias commands allow you to rename, or define/undefine “shortcuts” (including mental), for commands.

Their use parallels their name – you are using another name, that is easier to type/remember, for something.

You can set an alias in *your shell interactively* (you will only have it locally and in child processes)

or set in *your configuration files* (.cshrc/.bashrc) so it is available every time you login, start a shell or open a new terminal window (which starts a shell for that terminal window).

Typical UNIX think.

When to make/use aliases.

Anytime you find yourself typing the same command over and over, you could make an alias.

Anytime you prefer to type a command “your way”.

Typical UNIX think.

When to make/use aliases.

Anytime you find yourself mis-typing the same thing over and over, you could make an alias

(“mroe” is usually aliased to the more command for example {why learn to type?}.

The original, interactive spelling corrector!).

Example aliases taken from .cshrc on CERISUN system (so you get these automatically).

```
alias settitlebar 'echo -n "^[ ]2;$CWD^G"'
alias cd 'chdir \!* && cwdcmd && settitlebar'
alias howmuch 'du -sk .'
alias a alias
alias h 'history'
alias u unalias
alias m more
alias mroe more
alias l 'ls -F'
alias c clear
alias src source
```

Example aliases taken from my SUN .cshrc file.

```
alias mjday '/gaia/dunedain/d2/gps/oldbin/mjday'  
alias home "cd ~"  
alias x 'chmod +x'  
alias dir 'ls -lt | more'  
alias hp "lpr -Php_3890 "  
alias tek "lpr -P3904_tek"  
alias nb "lpr -P3892_grad "  
alias nbcolor "lpr -P3892_hpcolor "  
alias DEM "cd $home/dem"  
alias ssh_yang 'ssh -l gps yang.soest.hawaii.edu'  
alias ftp_jpl 'ftp bodhi.jpl.nasa.gov'  
alias matlab_term 'matlab -nodesktop -nosplash'
```

You can find all the aliases that are defined by using the command `alias` without any arguments.

Dealing with file names with special characters

Basics of the UNIX/Linux Environment

Say I have a file named “!”. (this is probably because I used >! at some time while in bash, but this syntax is for tcsh not bash, so I redirected my output to a file called !)

```
$ rm !  
remove !? Y
```

That was easy.

What about a file named “_”

Make a file named “~” with touch command

(use man to see what the touch command does)

```
> touch -
```

```
> ls
```

```
-          f2.dat          HW          hw1a.txt    SCRIPTS
f1.dat     f_1_2_3.dat    hw1.txt     NOTES      SRC
```

Try to remove it.

```
> rm -
```

```
usage: rm [-fiRr] file ...
```

What is the problem? (you tell me.)

We have to let the shell know that the “-” is NOT a switch.

Use the “-” switch all by itself.

```
> rm - -  
rm: remove - (yes/no)? Y  
>
```

Remember that filenames can have any character but the “/” (used to define the path), so sooner or later you are going to get a file name that will be hard or dangerous to reference.

You will have to be especially careful/creative if you get a file named “*” as

`%rm *`

can be disastrous

(and the more privileges you have and the higher up you are in the directory structure, the more disastrous it is.)

File Permissions

Basics of the UNIX/Linux Environment

Every user on a UNIX system has a unique username, and is a member of at least one group (the primary group for that user).

A user can also be a member of one or more other groups.

Only the administrator can create new groups or add/delete group members (one of the shortcomings of the system).

Every file (directories are files) on the system has an owner, and also an associated group.

Every file also has a set of permission flags which specify separate read, write and execute permissions for the

'user' (owner),

'group',

and 'other'

(everyone else with an account on the computer)

Permissions

Read

ability to read the file (r).

Write

ability to write or overwrite the file (w).

Execute

ability to execute or run the file and allow others to view directories (x).

(if a directory is not executable, non-owner's cannot cd into it or see what is in it at all.)

How to view the ownership & permissions of files/directories (review)

ls -l: lists long format

```
> ls -l
total 2201712
-rw-rw-rw- 1 rsmalley user 54847 Mar  7 2009 *CHARGE-2002-107*
-rw-rw-rw- 1 rsmalley user   413 Oct 30 2006 022285A.cmt
-rwxrwxrwx 1 rsmalley user 13092 Aug 13 2007 a.out
Drwxrwxrwx 3 rsmalley user   512 Oct 10 2008 adelitst
Drwxrwxrwx 5 rsmalley user   512 Aug 29 2007 ANT_GMT
```

Permissions

How to view the ownership & permissions of files/directories (review)

ls -l: lists long format

```
> ls -l
```

```
total 2201712
```

```
-rw-rw-rw- 1 rsmalley user 54847 Mar  7 2009 *CHARGE-2002-107*  
-rw-rw-rw- 1 rsmalley user   413 Oct 30 2006 022285A.cmt  
-rwxrwxrwx 1 rsmalley user 13092 Aug 13 2007 a.out  
Drwxrwxrwx 3 rsmalley user   512 Oct 10 2008 adelitst  
Drwxrwxrwx 5 rsmalley user   512 Aug 29 2007 ANT_GMT
```

Owner

How to view the ownership & permissions of files/directories (review)

ls -l: lists long format

```
> ls -l
```

```
total 2201712
```

```
-rw-rw-rw- 1 rsmalley user 54847 Mar  7 2009 *CHARGE-2002-107*  
-rw-rw-rw- 1 rsmalley user   413 Oct 30 2006 022285A.cmt  
-rwxrwxrwx 1 rsmalley user 13092 Aug 13 2007 a.out  
Drwxrwxrwx 3 rsmalley user   512 Oct 10 2008 adelitst  
Drwxrwxrwx 5 rsmalley user   512 Aug 29 2007 ANT_GMT
```

Group

Changing owners and groups.

If you create a file, you are the owner/user.

Mitch and Bob have the SUN and Mac systems set up to automatically set the group to 'user', or all users of the CERI UNIX system.

Default permissions on the SUN are

`rw-r--r--`

(numerically 644)

And on the Mac are (seem to be)

`rwX-----`

(numerically 700)

chmod

Command to change file or directory permissions (change mode in the normal UNIX philosophy of naming commands).

```
%chmod ugo+x hello.sh
```

```
%ls -lF hello.sh
```

```
-rwxr-xr-x    1 rsmalley user      21 Sep 16 08:36 hello.sh*
```

go-x Removes execute privileges from group and other

o+r Adds read privileges to other

Flags (or numerical values) allows you to set the permissions. Using wildcards you can set permissions globally within a directory, and with the `-r` flag all subdirectories.

Changing Permissions

you can also use octal values (numbers) to change ownership

644 represents u=rw; go=r
755 represents u=rwx; go=rx

(using this puts you in a special eunuch class)

Connecting remotely

Basics of the UNIX/Linux Environment

On a Mac running OS-X, from a terminal window
enter

```
ssh -X alpaca.ceri.memphis.edu -l rsmalley
```

The `-X` flag gives us X-windows graphics
capability.

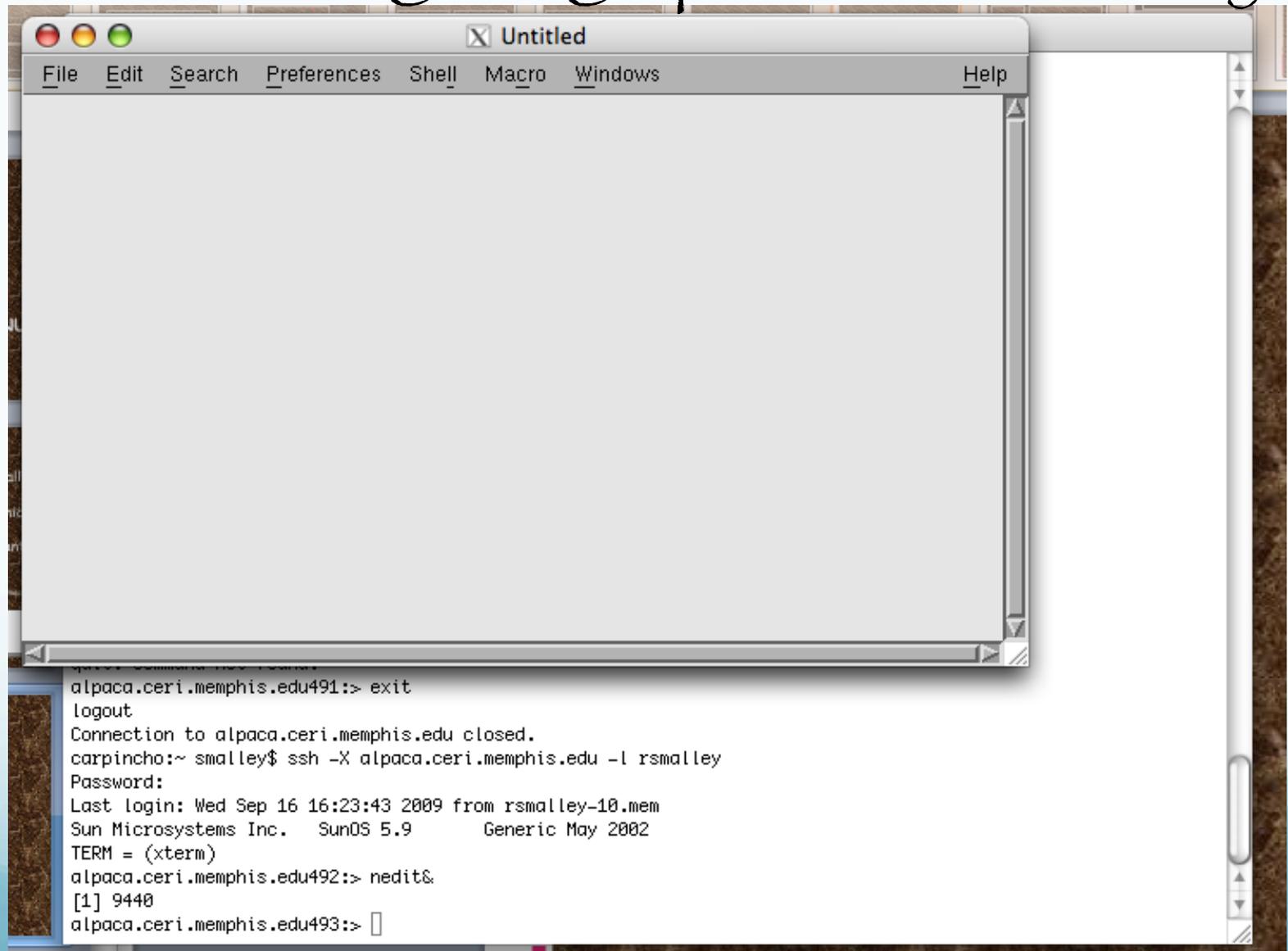
Next is the name of the machine we want to
connect to.

The `-l` flag passes the username.

(Without this flag, it will pass whatever your username is on the mac.)

```
carpincho:~ smalley$  
carpincho:~ smalley$ ssh -X alpaca.ceri.memphis.edu -l rsmalley  
Password:  
Last login: Wed Sep 16 15:56:01 2009 from carpincho.ceri.  
Sun Microsystems Inc. SunOS 5.9 Generic May 2002  
TERM = (xterm)  
alpaca.ceri.memphis.edu489:> █
```

Try running nedit.
On the mac – we get X graphics automatically



The image shows a screenshot of a Mac OS X terminal window. The window title is "Untitled" and the menu bar includes "File", "Edit", "Search", "Preferences", "Shell", "Macro", "Windows", and "Help". The terminal output shows the following sequence of commands and responses:

```
alpaca.ceri.memphis.edu491:> exit
logout
Connection to alpaca.ceri.memphis.edu closed.
carpincho:~ smalley$ ssh -X alpaca.ceri.memphis.edu -l rsmalley
Password:
Last login: Wed Sep 16 16:23:43 2009 from rsmalley-10.mem
Sun Microsystems Inc. SunOS 5.9 Generic May 2002
TERM = (xterm)
alpaca.ceri.memphis.edu492:> nedit&
[1] 9448
alpaca.ceri.memphis.edu493:> 
```

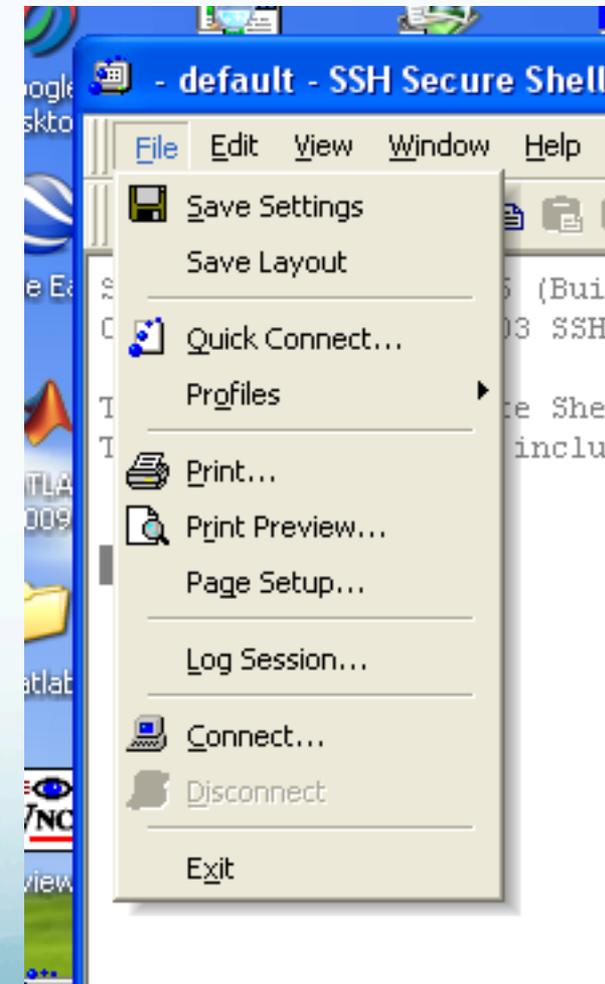
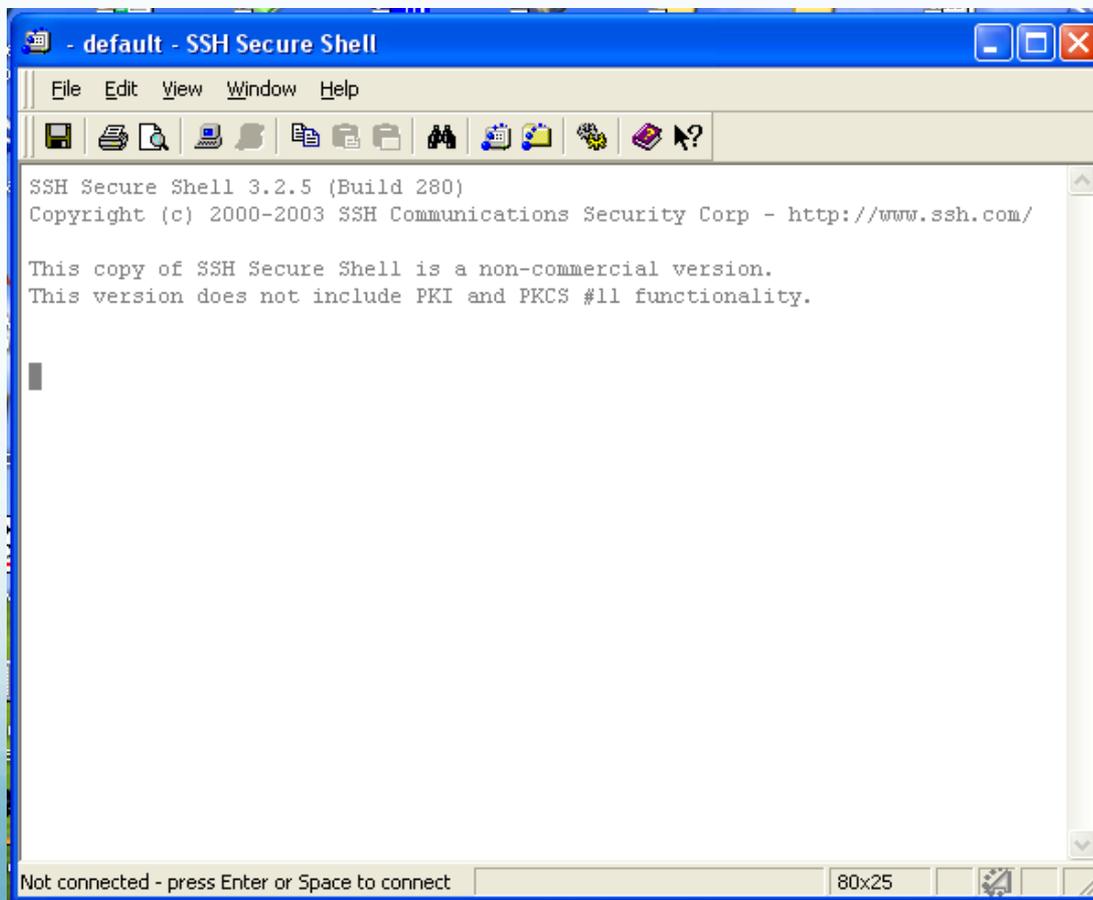
On the PC it is a few more clicks, but first we need (to install) two programs SSH Secure Shell Client and Exceed (part of the Hummingbird package).



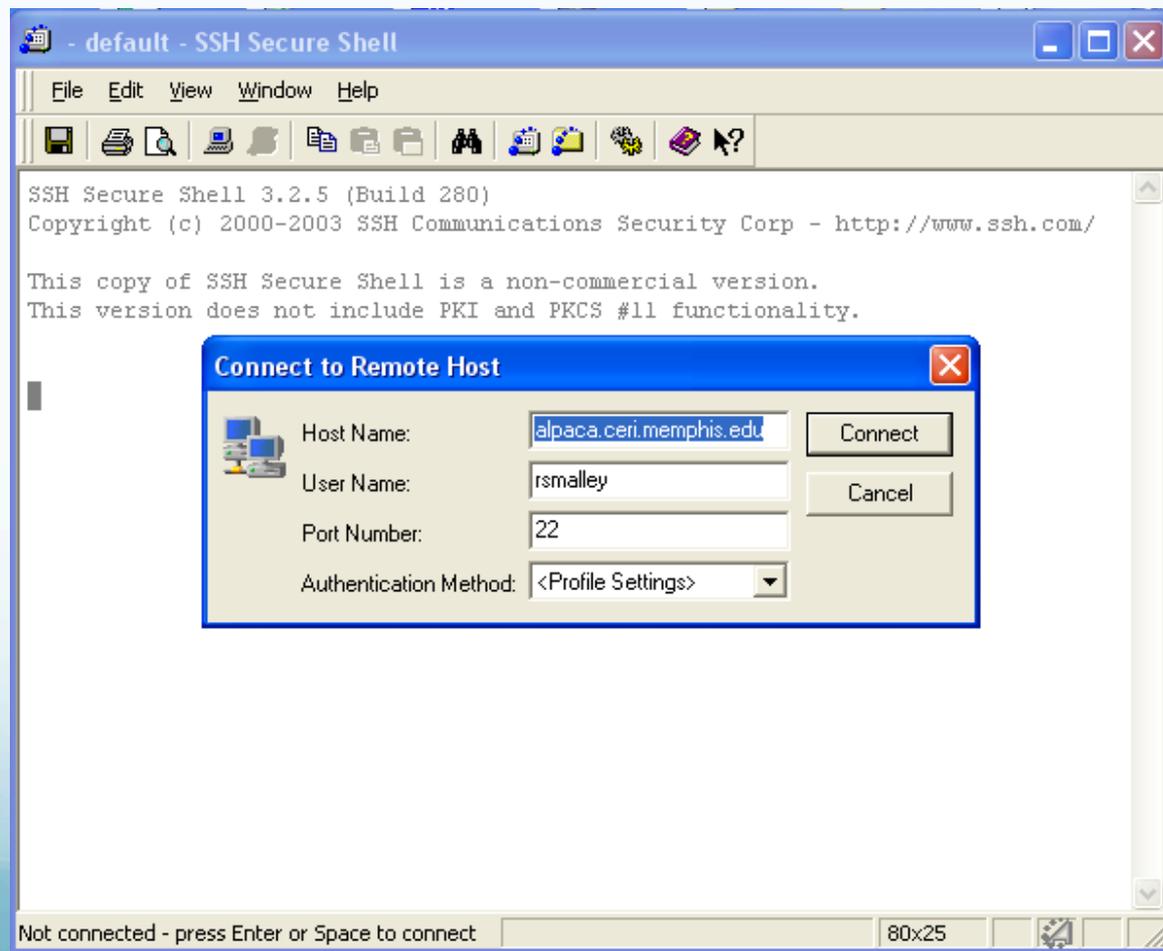
Double click on exceed (it will start up and put an icon in the tray, it does not have a window).

Double click on SSH Secure Shell Client

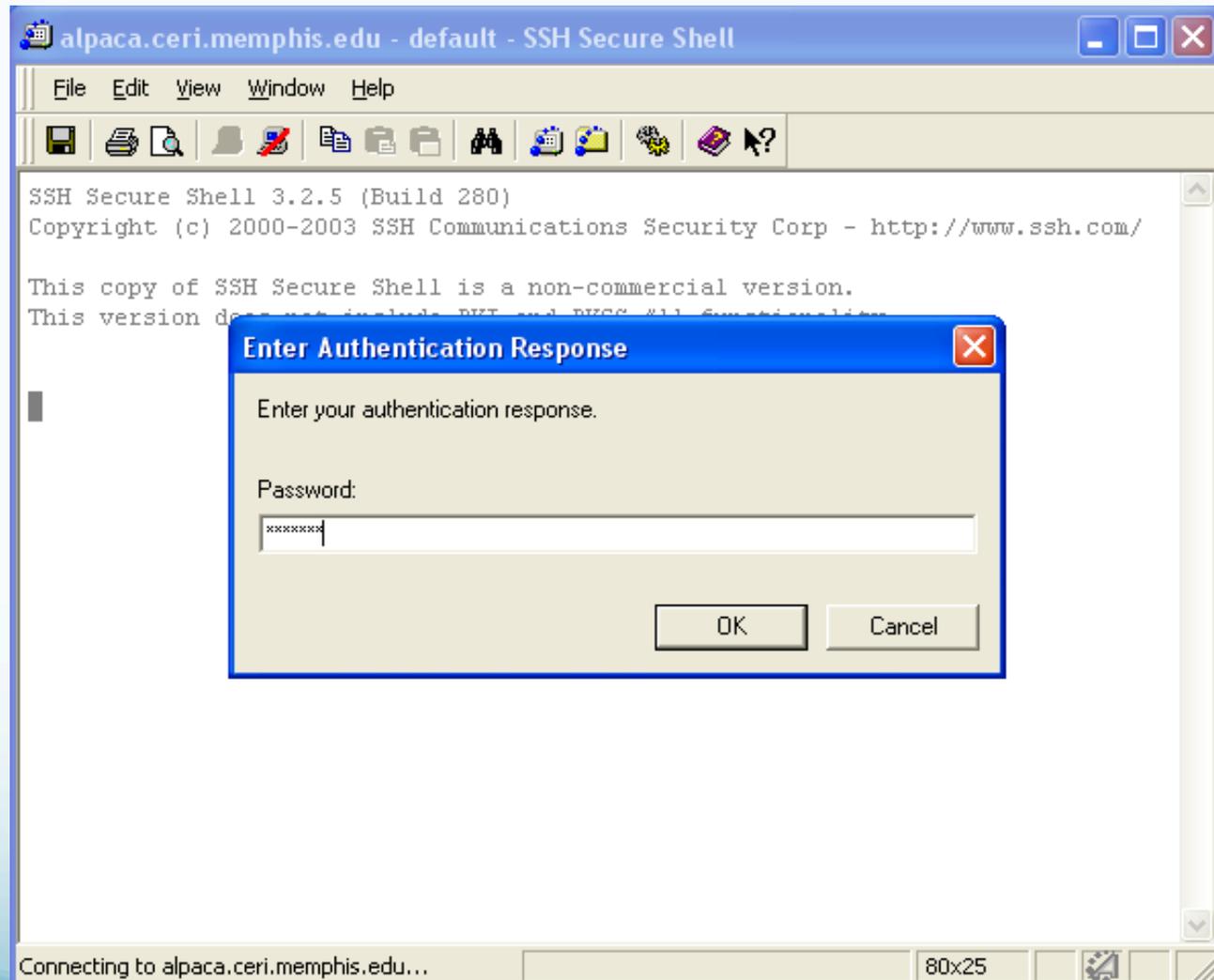
You will get this window (left). Now we have to connect to a machine. Click on File and then connect.



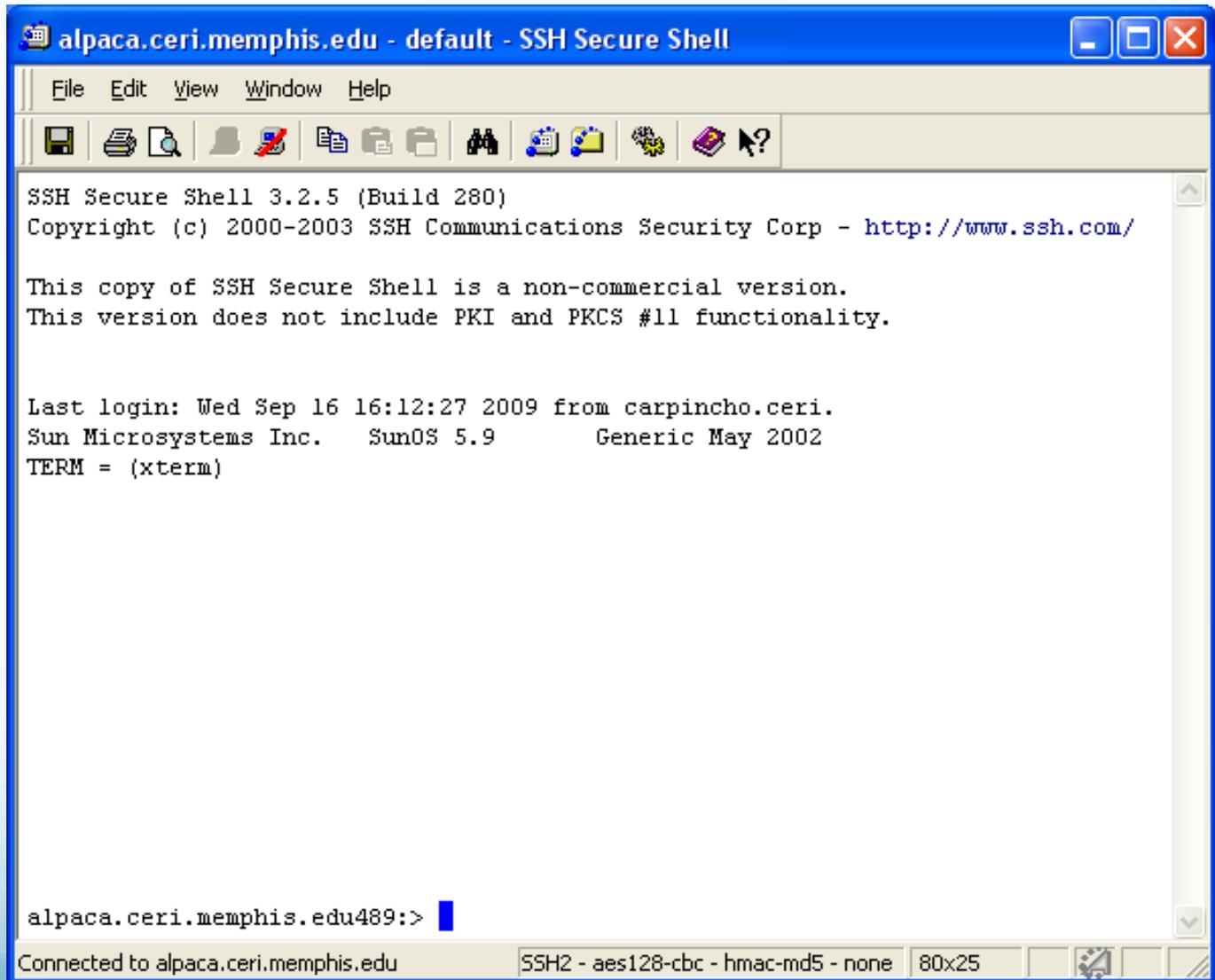
This brings up the connect dialog. Put in the host name you want to connect to and your username. Leave the other stuff alone (default). Click connect.



It will now ask for your password.



And we are finally connected.



The image shows a screenshot of an SSH Secure Shell terminal window. The window title is "alpaca.ceri.memphis.edu - default - SSH Secure Shell". The menu bar includes "File", "Edit", "View", "Window", and "Help". The toolbar contains various icons for file operations and system functions. The terminal text displays the SSH version (3.2.5), copyright information (2000-2003 SSH Communications Security Corp), and a notice that this is a non-commercial version without PKI and PKCS #11 functionality. It also shows the last login time (Wed Sep 16 16:12:27 2009) and system information (Sun Microsystems Inc. SunOS 5.9, Generic May 2002). The terminal prompt is "alpaca.ceri.memphis.edu489:>". The status bar at the bottom indicates the connection details: "Connected to alpaca.ceri.memphis.edu", encryption settings "SSH2 - aes128-cbc - hmac-md5 - none", and window dimensions "80x25".

```
alpaca.ceri.memphis.edu - default - SSH Secure Shell
File Edit View Window Help
SSH Secure Shell 3.2.5 (Build 280)
Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/

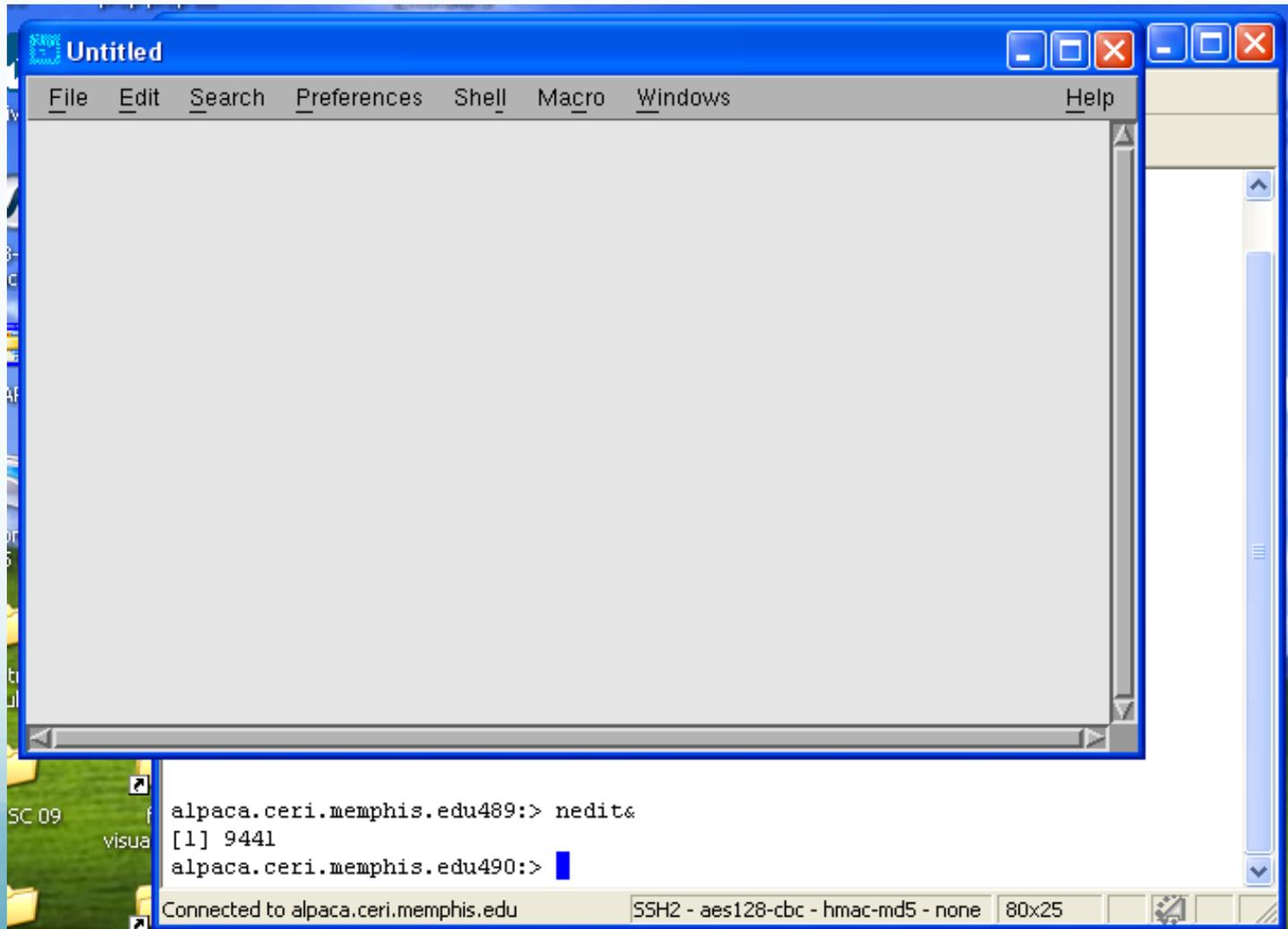
This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.

Last login: Wed Sep 16 16:12:27 2009 from carpincho.ceri.
Sun Microsystems Inc. SunOS 5.9 Generic May 2002
TERM = (xterm)

alpaca.ceri.memphis.edu489:>
```

Connected to alpaca.ceri.memphis.edu SSH2 - aes128-cbc - hmac-md5 - none 80x25

Start nedit in the background (the trailing &).
This permits the terminal to continue accepting
commands.



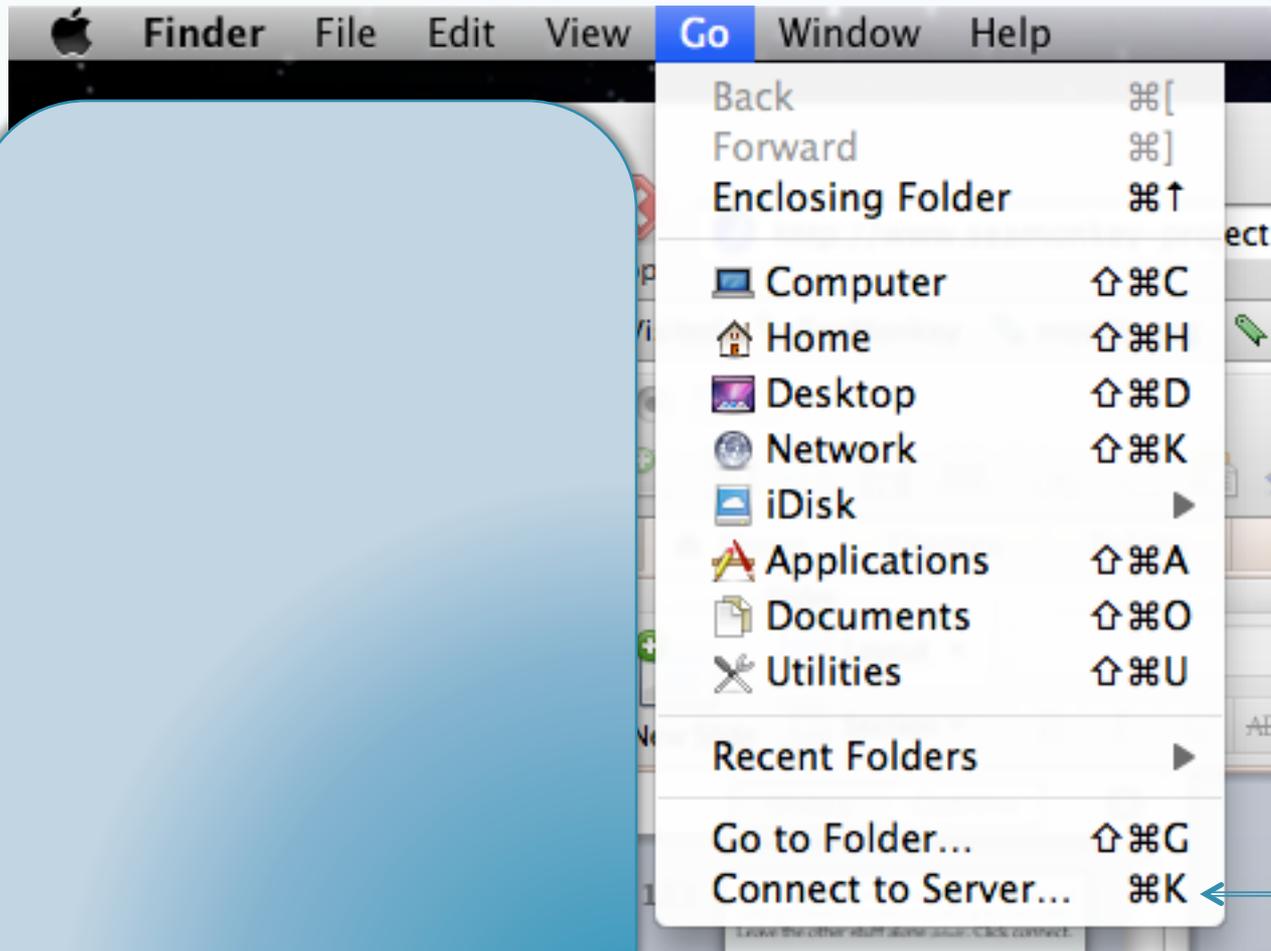
Unfortunately, you cannot ssh into the Student Lab Mac system from off-campus.

You will have to ssh into the sun system (use enigma) and from there ssh into the Student Mac Lab system.

This means that you will probably not be able to do graphics remotely from the Student Mac Lab.

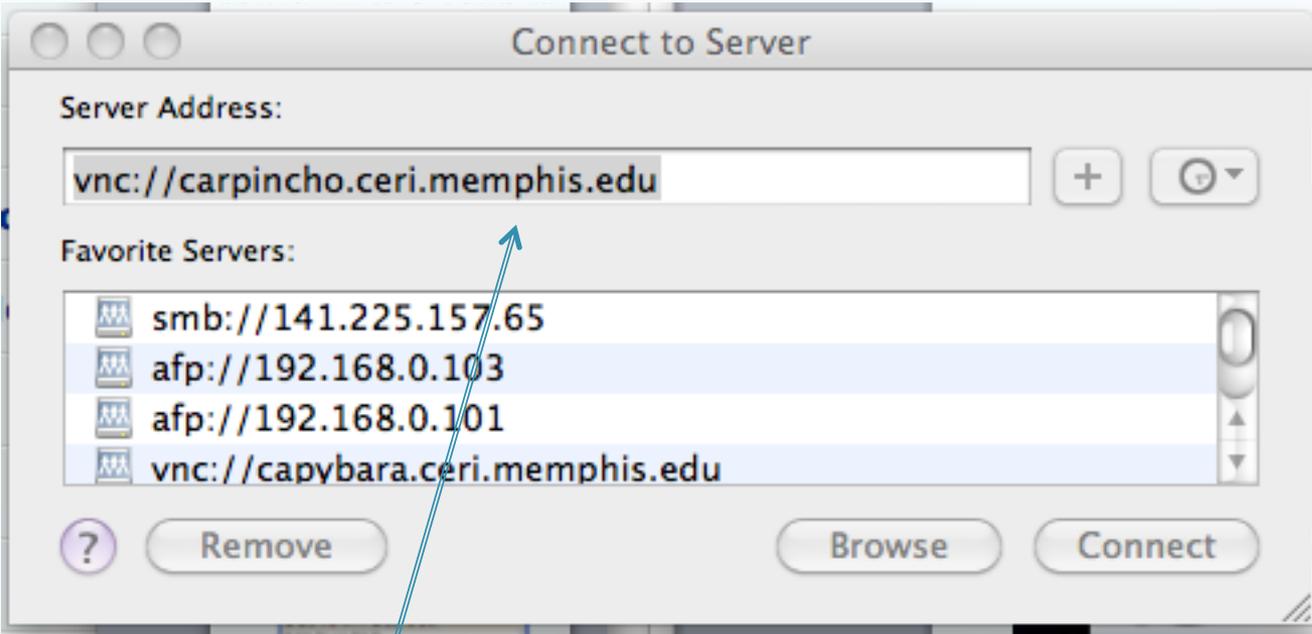
You can only do the terminal interface.

Using Screen Sharing/VNC



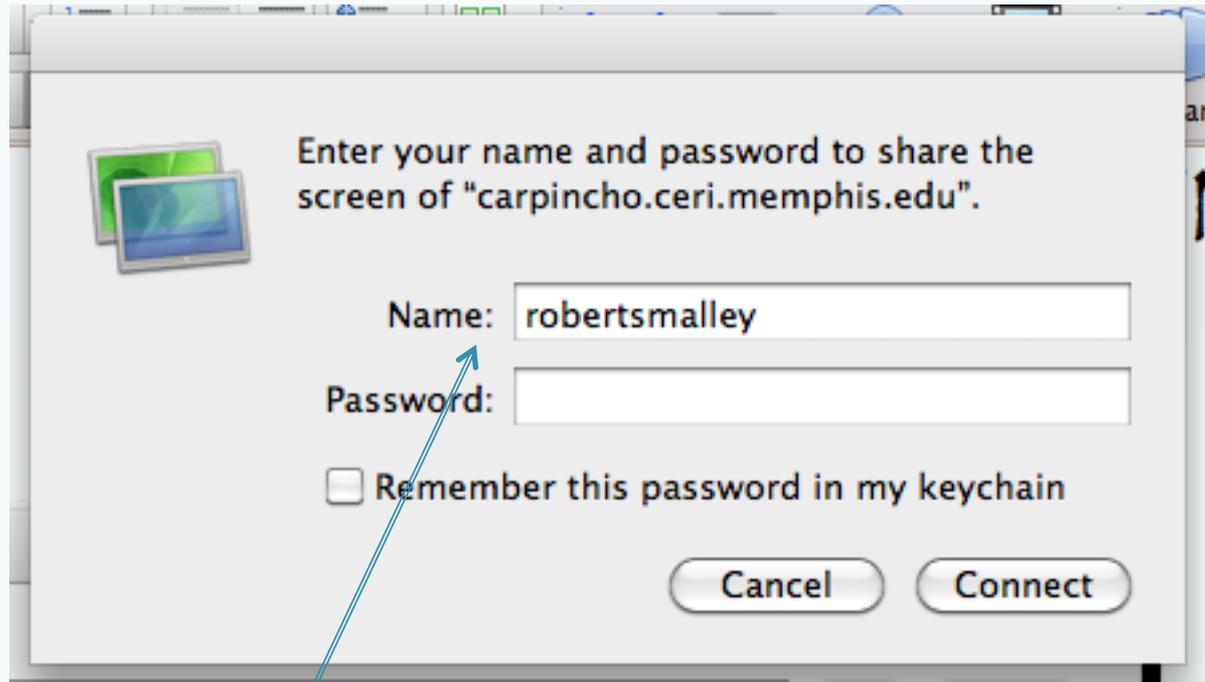
Select this
(will be highlighted –
does not come
through on screen
capture)

Using Screen Sharing/VNC



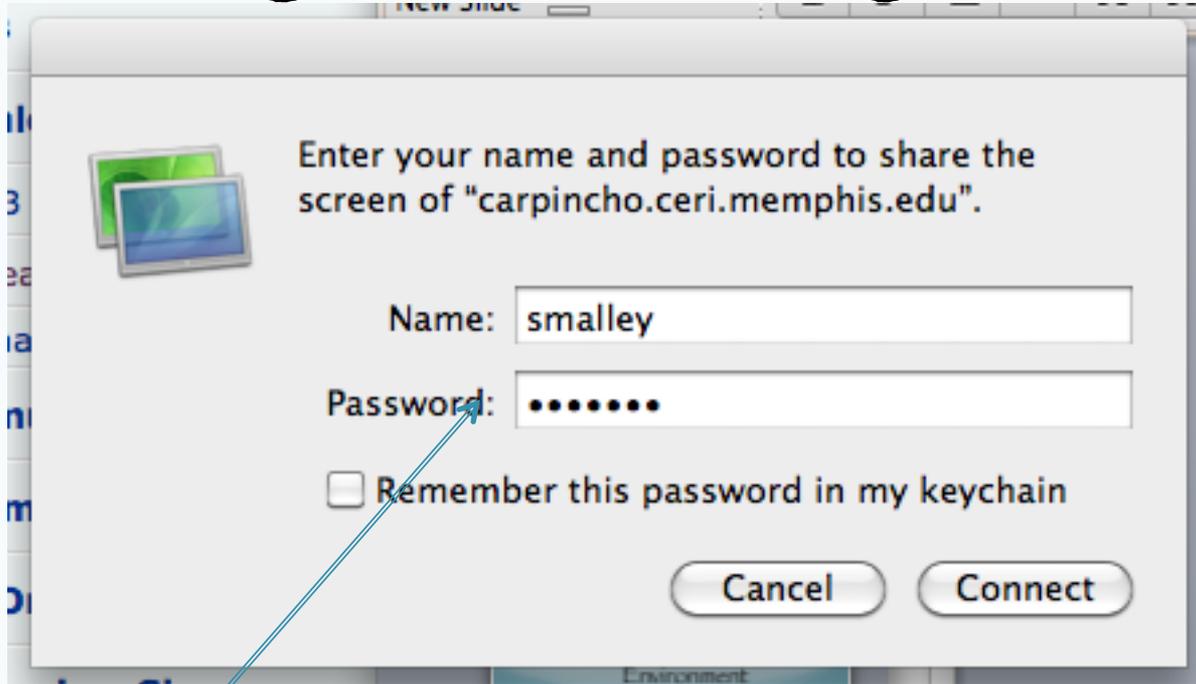
Select address from list, enter it, or browse
(will be highlighted – does not come through on screen capture)
Then click connect

Using Screen Sharing/VNC



Now you get a login screen
It will have automatically put in the username on
the LOCAL machine.

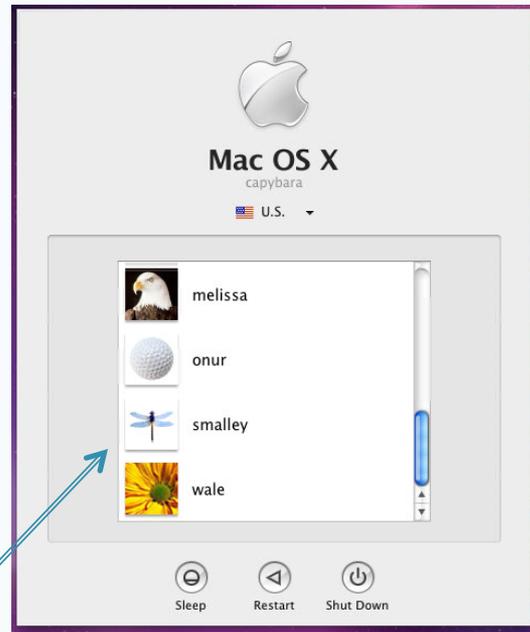
Using Screen Sharing/VNC



You may have to change the username to a different one on the REMOTE machine.
Plus put in your password.

Not a good idea to have the computer remember your password.

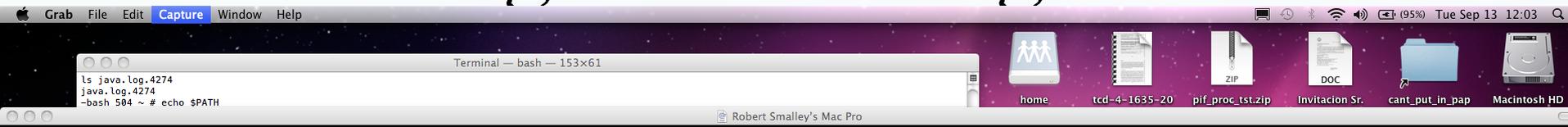
Using Screen Sharing/VNC



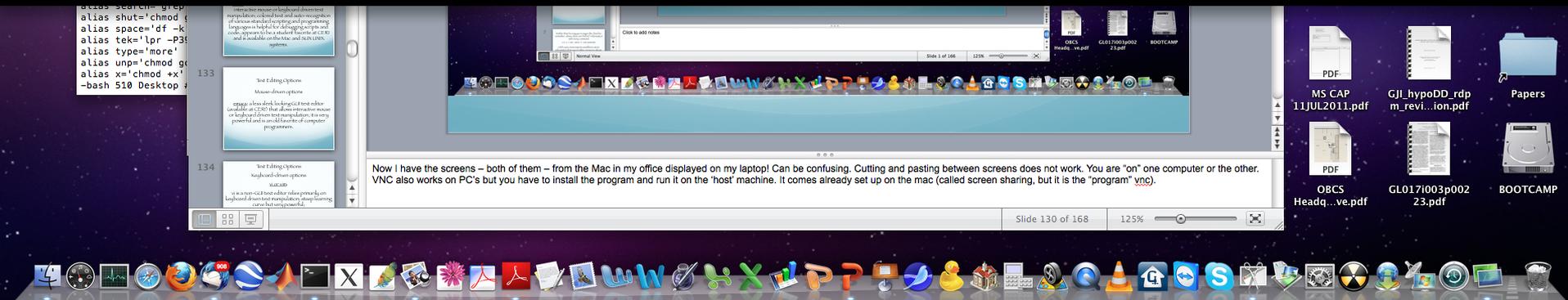
Now you will have to log into the REMOTE machine.

The login process is 2 levels – one to connect to the machine (as an authorized user) and one to login (possibly as another authorized user).

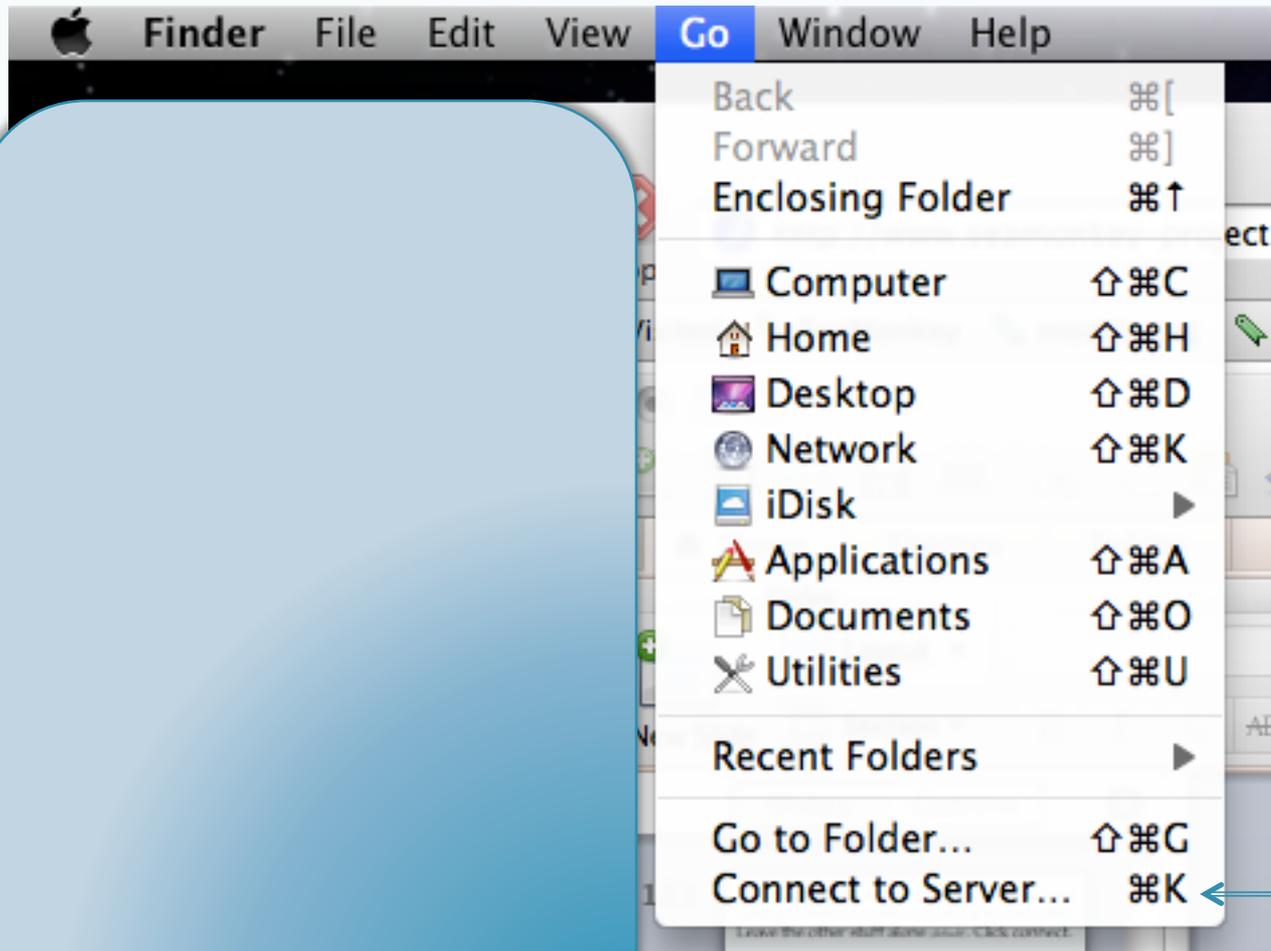
Using Screen Sharing/VNC



In this example someone is logged in and the screen is locked and you need to enter the password

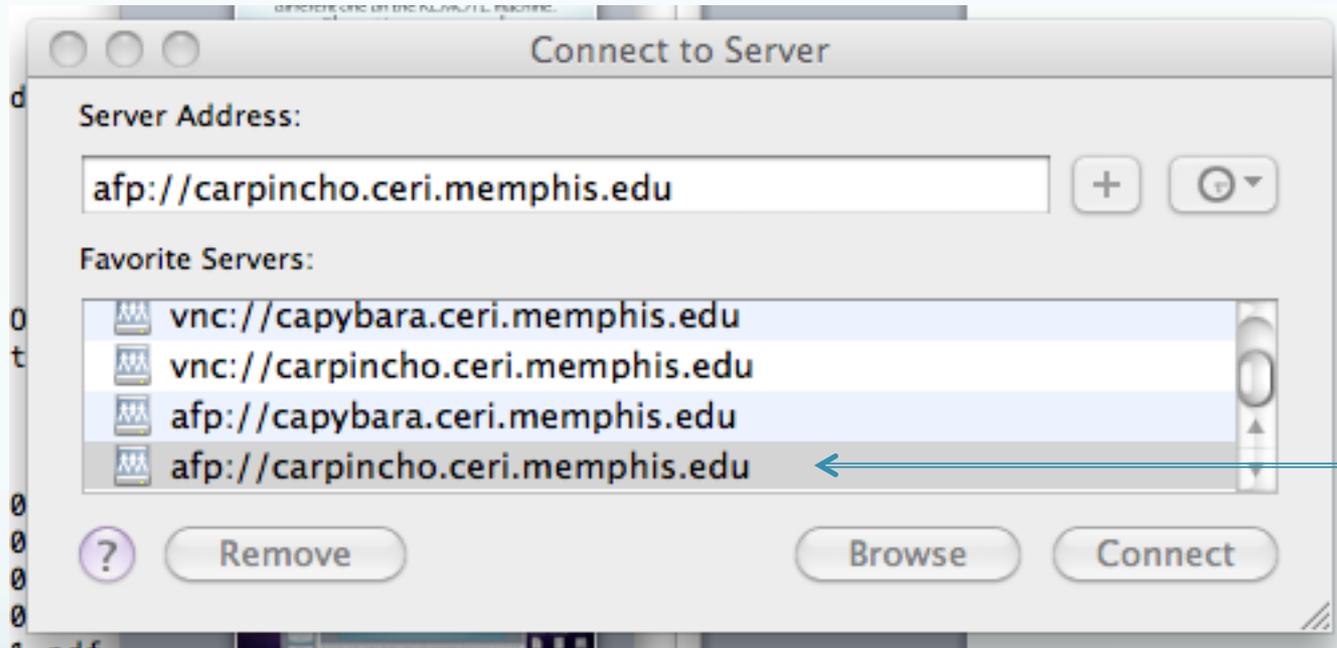


Sharing disks



Select this
(will be highlighted –
does not come
through on screen
capture)

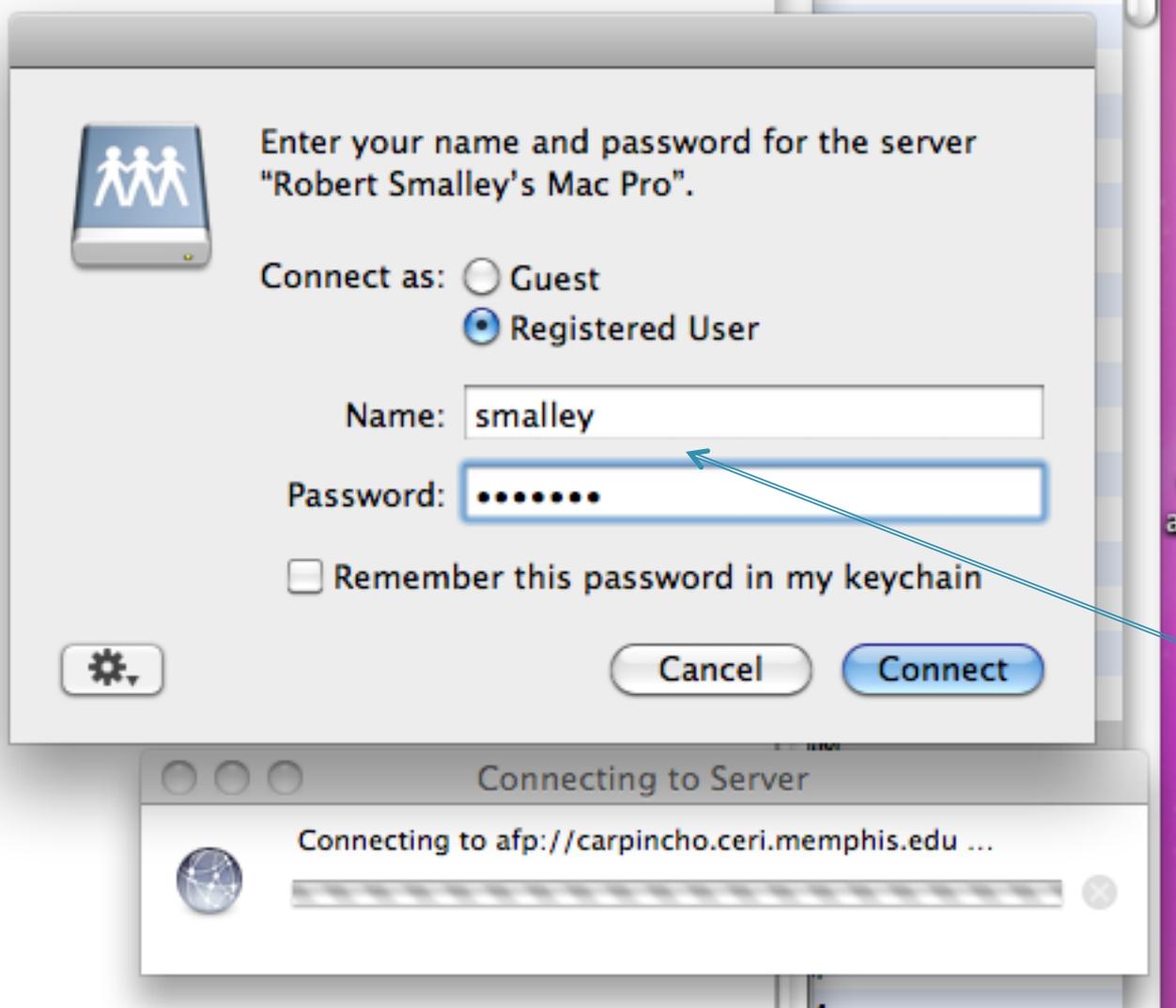
Sharing disks



Now select this instead (will be highlighted – does not come through on screen capture. When you double click it goes up top. Or type it in up top.)

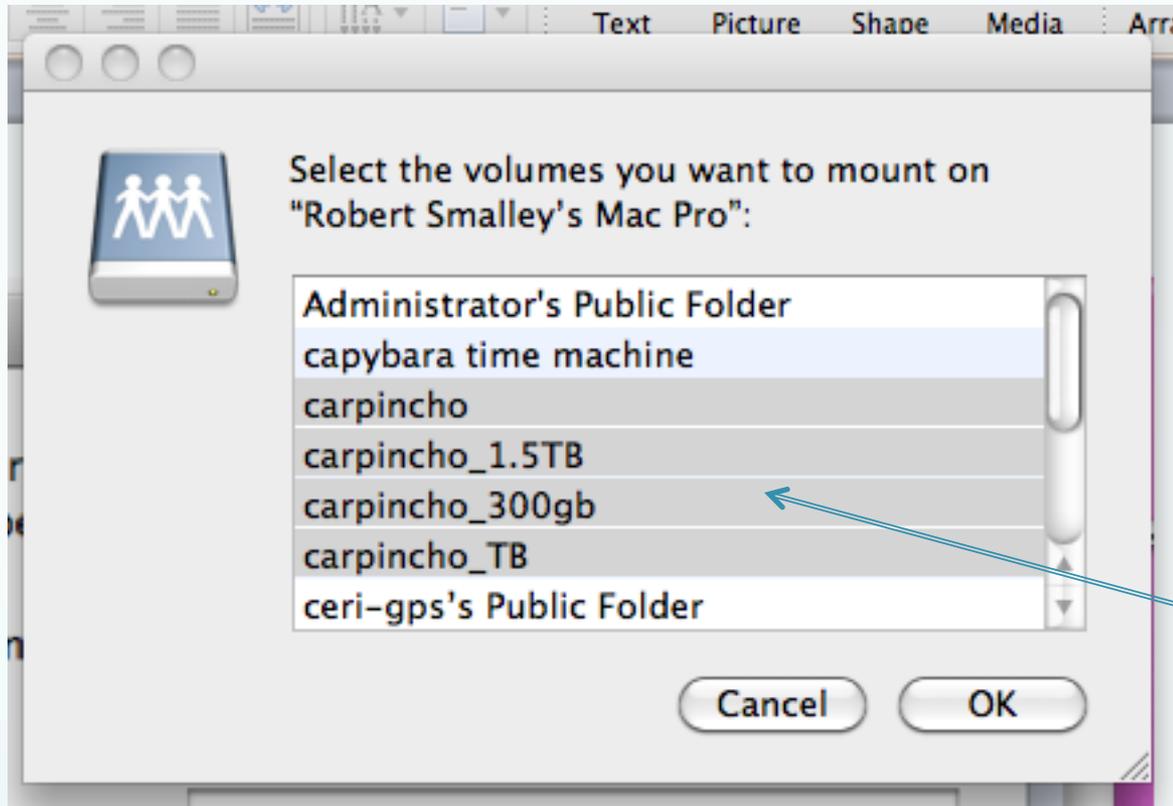
“afp” is apple file protocol – it is how the Mac shares disks with other Macs.

Sharing disks



You have to
log in.

Sharing disks



Select the disks you want to mount. (it will be blue)

Sharing disks



They show up on the desktop and can be accessed under /Volumes in the UNIX file structure.

```
-bash 520 ~ # cd /Volumes
-bash 521 Volumes # ls
BOOTCAMP      Macintosh HD
-bash 522 Volumes #
```

```
carpincho      carpincho_1.5TB carpincho_300gb carpincho_TB
```

Sharing disks

You can also “mount” disks from the UNIX and PC systems using “samba”

smb://TCP/IP address or name