Classroom Introduction to Unix/Linux and Running Jobs on the Ada Cluster

Course Material

https://hprc.tamu.edu/wiki/index.php/HPRC:Classes:Hwang

Open Access Lab Workstations

Log in with NetID + Password (same as howdy.tamu.edu)

Head Start

If you know how, open MobaXterm and connect to Ada

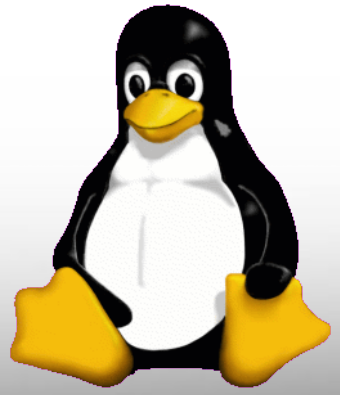
ssh [NetID]@ada.tamu.edu

I. Introduction to Unix/Linux, and II. Running Jobs on the Ada Cluster

Rick McMullen, Ph.D., Associate Director HPRC
mcmullen@tamu.edu

Texas A&M University
High Performance Research Computing

TEXAS A&M UNIVERSITY



HPRC Help Desk

Website:	https://hprc.tamu.edu
Email:	help@hprc.tamu.edu
Telephone:	(979) 845-0219
Visit us in person:	104B Henderson Hall

Appointments are appreciated, but not required

Help us, help you -- we need more info

- Which Cluster
- UserID/NetID
- Job id(s) if any
- Location of your jobfile, input/output files
- Application used if any
- Module(s) loaded if any
- Error messages
- Steps you have taken, so we can reproduce the problem

Additional References

A wide range of information and training content are available through:
HPC University, http://hpcuniversity.org/

Linux/Unix Basics for HPC: October 9, 2014 (with video) [TACC]
https://portal.tacc.utexas.edu/-/linux-unix-basics-for-hpc

Express Linux Tutorial: Learn Basic Commands in an Hour [TACC]
https://portal.tacc.utexas.edu/c/document_library/get_file?uuid=ed6c16e9-bcbc-4b70-9311- 5273b09508b8&groupId=13601

Introduction to Linux for HPC [LSU]
http://www.hpc.lsu.edu/training/weekly-materials/2015-Fall/intro-linux-2015-09-02.pdf

Logistics

Progression: "How do I…?"

Focus: "What's next?"

Goal: "I can use this comfortably!"

Five Sections

Based on how our users have learned the Unix/Linux environment

Each Section

Information + Examples + Checkpoint

General Definitions

Unix/Linux:	Operating system
Distribution:	Operating system + software collection

Local:	The computer in front of you
Remote:	A computer you connect to

Interactive:	A program that stops to ask you for input
GUI:	Graphical User Interface
Terminal:	Text-based interface for launching commands

Documentation: the *man* command

```
$ man cmd_name
```

View man page for *gedit*:

```
$ man gedit
```

View man page for *scp*:

```
$ man scp
```

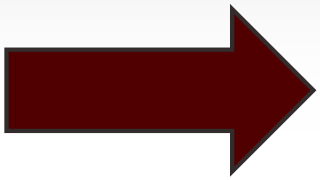
A man page is organized in a standard layout:
NAME, SYNOPSIS, DESCRIPTION, OPTIONS, …

Many users find it easier to use the Internet.
Most man pages are available for viewing in an internet browser.

Press *'q'* to exit a man page.

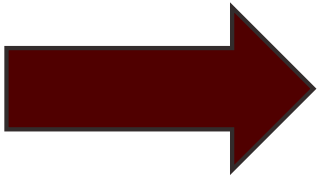
Overview

Section I
Connect
Navigate
View Files



Section II
Directories
Attributes
Edit Files

Section III
Transfer Files
Processes
Signals



Section IV
Bash
Environment
Redirects & Pipes

Section V
Other Topics

Section I Definitions

ssh:	Secure Shell – encrypted network protocol
X11:	Enables GUI over network

Xming:	X11 for Windows
XQuartz:	X11 for OS X

PuTTY:	Tool for SSH and Telnet connection
MobaXterm:	Tool for SSH + X11 + other connections *MobaXterm will replace PuTTY + Xming for this class*

Log In – Remote Access

We use *ssh* to connect and issue commands.

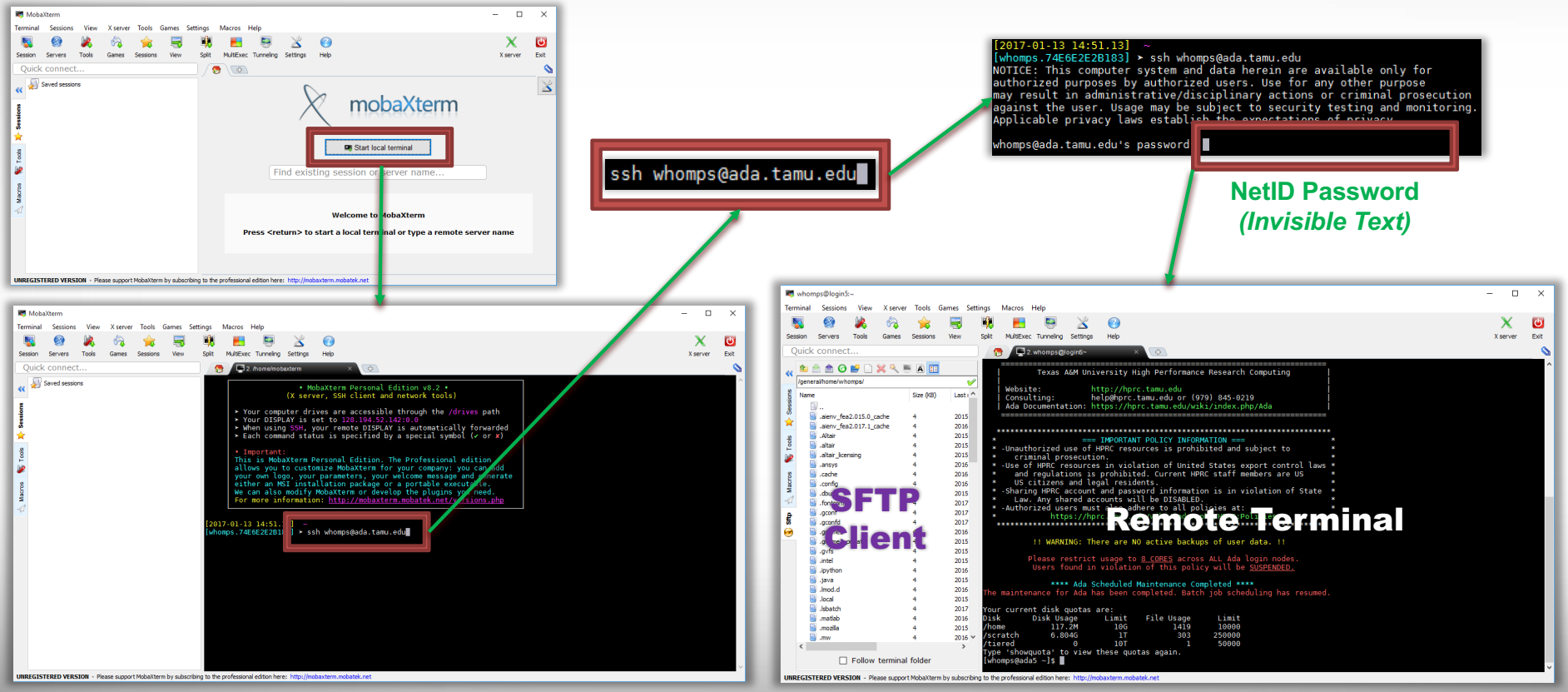
Windows: *MobaXterm*
See also: *https://hprc.tamu.edu/wiki/index.php/HPRC:Access:Windows*

OS X: *Terminal + Xquartz*

Unix/Linux: *Terminal + X11*

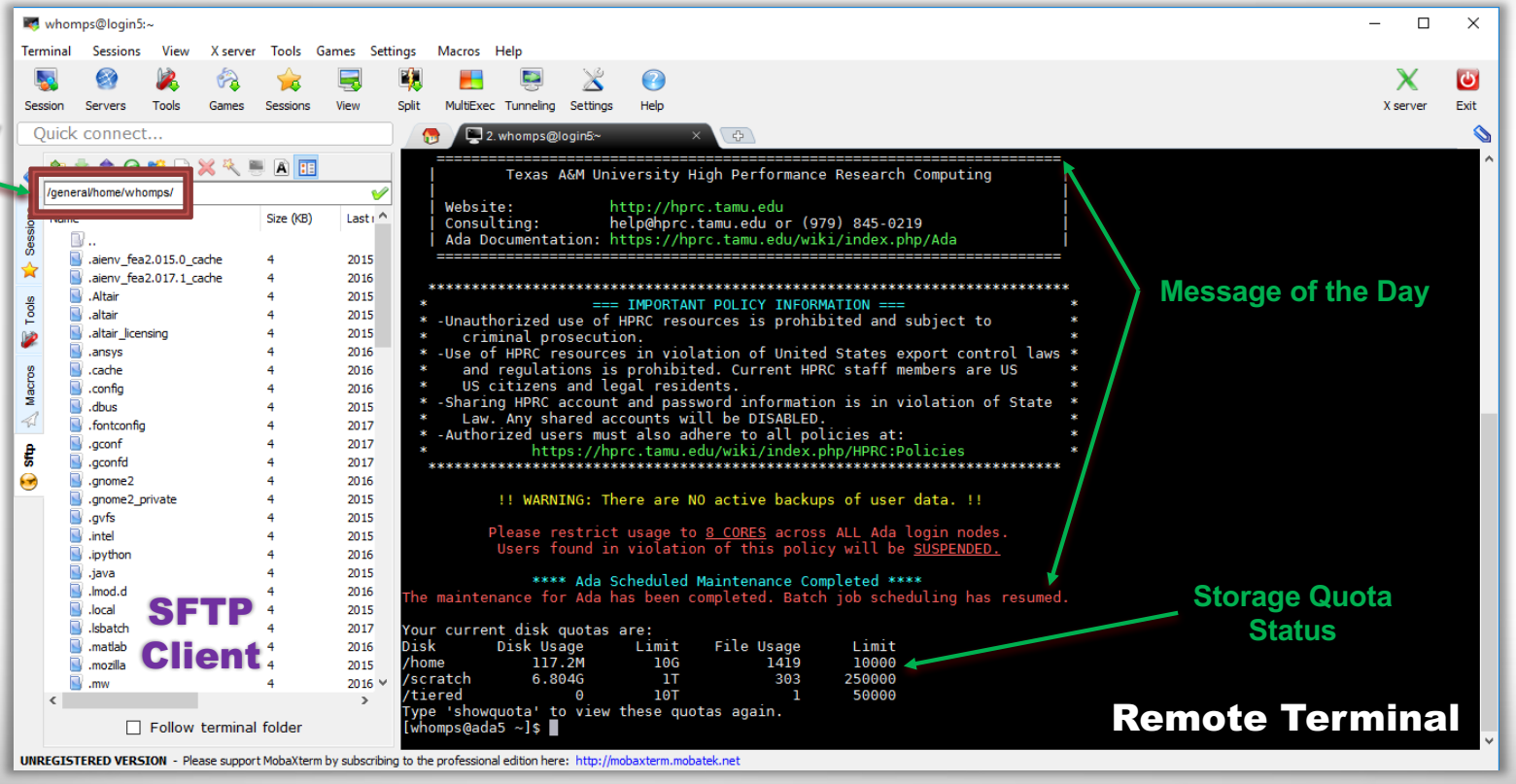
Using SSH - MobaXterm (on Windows)

https://hprc.tamu.edu/wiki/index.php/HPRC:Access:Windows



Section I: Connect

Using SSH - MobaXterm (on Windows)



Using SSH (with a terminal)

https://hprc.tamu.edu/wiki/index.php/Ada:Access

You may see something like the following the first time you connect to the remote machine from your local machine:

```
% ssh -X user_NetID@ada.tamu.edu
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)?
```

Type **yes**. You will then see the following:

```
Host 'ada.tamu.edu' added to the list of known hosts.
user_NetID@ada.tamu.edu's password:
```

You will use the *ssh* command when connecting from OS X, UNIX/Linux, or MobaXterm hosts.

Your Login Password

Both state of Texas law and TAMU regulations prohibit the sharing and/or illegal use of computer passwords and accounts.

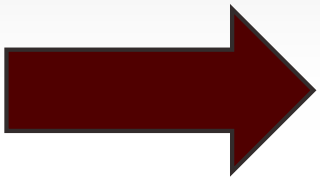
Be responsible with your password:

- Don’t write down passwords.
- Don’t choose easy to guess/crack passwords.
- Change passwords frequently.

TAMU HPRC resources use your NetID Credentials (“Howdy! Password”)

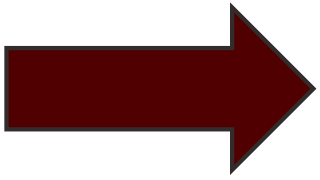
Overview

Section I
Connect
Navigate
View Files



Section II
Directories
Attributes
Edit Files

Section III
Transfer Files
Processes
Signals



Section IV
Bash
Environment
Redirects & Pipes

Section V
Other Topics

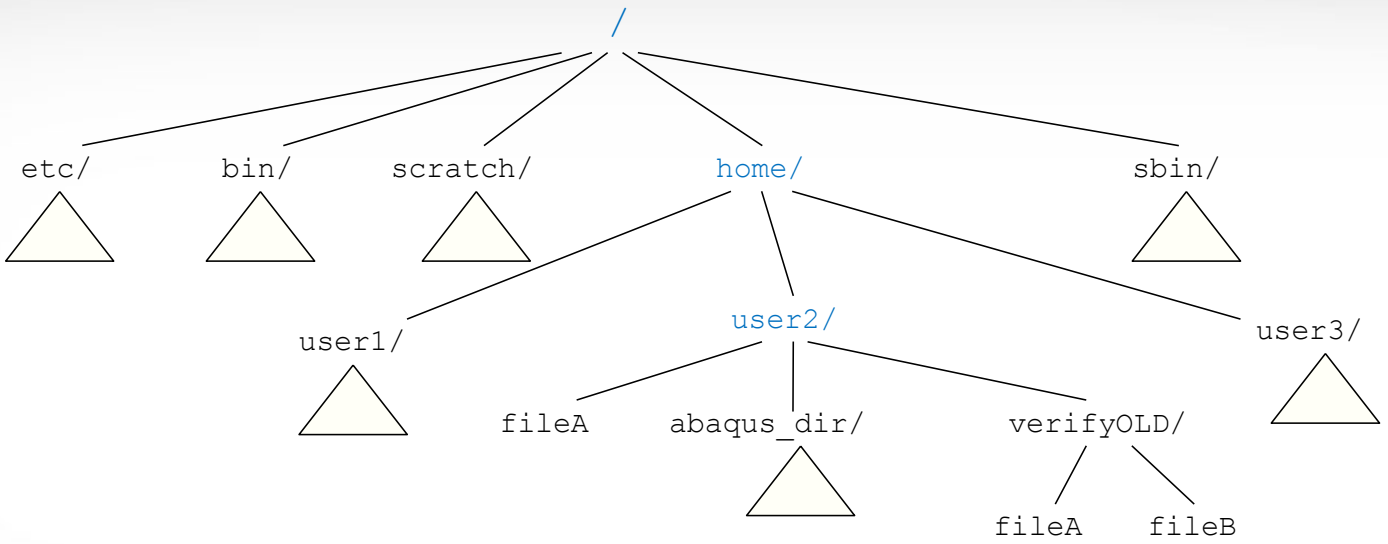
Where are you after you login?

```
$ pwd
```

pwd command (Print Current/Working Directory)

```
$ pwd
/home/user_NetID
```

Directory Paths



Upon login, you are located in your home directory.

In Windows, the home directory is usually `C:\Users\NetID`

On Ada, the home directory is located at `/home/NetID`

Listing Files and Directories: the *ls* command

```
$ ls [options] [directory or file name]
```

Commonly used options

- **-l** display contents in “long” format
- **-a** show all file (including hidden files - those beginning with .)
- **-t** sort listing by modification time
- **-r** reverse sort order
- **-F** append type indicators with each entry (* / = @ |)
- **-h** print sizes in user-friendly format (e.g. 1K, 234M, 2G)

Exercise:

```
$ ls
$ ls -a
```

```
$ touch hello.txt
$ ls
$ ls *.txt
```

The *tree* command

```
$ tree [dir_name]
```

Shows the contents of a directory structure in a hierarchical arrangement.

```
$ tree bin
bin
├── perlsh
└── xtail.pl

0 directories, 2 files
```

Changing Directories: the *cd* command

```
$ cd [directory name]
```

Return to last directory:

```
$ cd -
```

Go to parent directory:

```
$ cd ..
```

Return to home directory:

```
$ cd
```

or

```
$ cd ~
```

Exercise:

```
$ mkdir dir3
$ mkdir dir3/dir4
$ cd dir3
$ pwd
$ cd dir4
$ pwd
```

```
$ cd ..
$ pwd
$ cd dir4
$ pwd
$ cd -
$ pwd
```

```
$ cd
$ pwd
$ cd dir3
$ pwd
$ cd ~
$ pwd
```

**mkdir* means “make directory”

Useful Navigation Tips

Terminal usage involves a lot of memory and typing.
Save time and effort by using shortcuts.

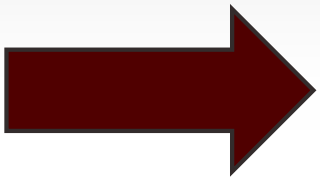
TAB-Completion: Use *TAB key* to complete when typing file, directory or command name

`history` **Command:** Show command history

Arrow Keys: *up arrow* and *down arrow* can browse through the command history

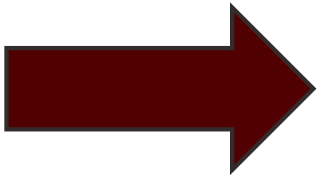
Overview

Section I
Connect
Navigate
View Files



Section II
Directories
Attributes
Edit Files

Section III
Transfer Files
Processes
Signals



Section IV
Bash
Environment
Redirects & Pipes

Section V
Other Topics

Displaying File Contents

Dump the contents of a file to the screen:

```
$ cat [file name]
```

Display a text file one page at a time:

```
$ more [file name]
```

Display a text file one page at a time:

```
$ less [file name]
```

Other related commands:

- ***head***: output the first part of files
- ***tail***: output the last part of files
- ***wc*** (word count) or ***wc -l*** (line count)

Exercise:

```
$ cat /etc/hosts
$ more /etc/hosts
$ less /etc/hosts
$ wc -l /etc/hosts
```

Displaying File Contents

Files can viewed with text editors.

Open a file with *gedit*:

```
$ gedit [file name]
```

Open a file with *nano*:

```
$ nano [file name]
```

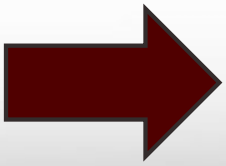
Open a file with *vi*:

```
$ vi [file name]
```

Graphic User Interface (GUI) options require X11 forwarding.

How do I choose?

1) What is installed?
2) What am I comfortable with?



New users usually like:

1) Text: *cat*
2) GUI: *gedit*

Types of File: the ***file*** command

```
$ file [name]
```

Displays a brief description of the contents or other type information for a file.

```
$ file hello.c
hello.c: ASCII C program text
```

file can display when a file has been edited on a Windows/DOS machine.
The **CRLF Line Terminators** will cause interpretation errors on Unix machines.

```
$ file dosText.txt
dosText.txt: […]with CRLF line terminators
$ dos2unix dosText.txt
```

Displaying Image Files

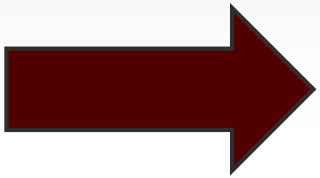
Eye of GNOME is installed on most of our systems.

```
$ eog [name]
```

Displays an image file in a new graphic window.

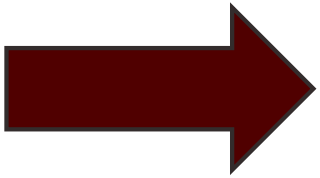
Overview

Section I
Connect
Navigate
View Files



Section II
Directories
Attributes
Edit Files

Section III
Transfer Files
Processes
Signals



Section IV
Bash
Environment
Redirects & Pipes

Section V
Other Topics

Section II Definitions

Directory: A container for files
Equivalent to Windows folders

Attributes: File properties + permissions
Info like "last edited" & "date created" & "owner"

PuTTY: Tool for SSH and Telnet connection

MobaXterm: Tool for SSH + X11 + other connections
MobaXterm will replace PuTTY + Xming for this class

Common Directory Commands

To make a new directory:

```
$ mkdir [directory name]
```

To change to another directory:

```
$ cd [directory name]
```

To remove an empty directory:

```
$ rmdir [directory name]
```

Exercise:

```
$ mkdir dir2
$ touch dir2/f2.txt
$ ls
$ ls dir2
```

```
$ pwd
$ cd dir2
$ pwd
$ cd ..
$ pwd
```

```
$ rmdir dir2
$ ls dir2
$ rm dir2/f2.txt
$ rmdir dir2
$ ls
```

File and Directory Names

Careful selection of characters prevents naming conflicts and errors.

Commonly Used	Do Not Use (Reserved)	Avoid Using
• A-Z • a-z • 0-9 • . *(period)* • - *(hyphen)* • _ *(underscore)*	• / *(forward slash)* • > *(greater than)* • < *(less than)* • | *(pipe)* • : *(colon)* • & *(ampersand)*	• *(white space)* • ( ) *(parentheses)* • ‘ *(quotes)* • ? *(question mark)* • * *(asterisk)* • \ *(backslash)* • $ *(dollar sign)*

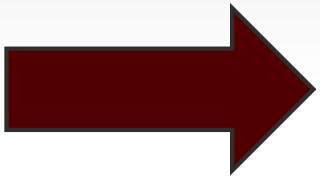
Don’t start or end your filename with a space, period, hyphen, or underscore.

Avoid blank space in the file name: ("*my data file*" vs "*my_data_file.txt*")

Names are case sensitive

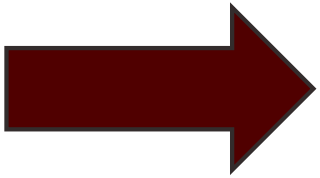
Overview

Section I
Connect
Navigate
View Files



Section II
Directories
Attributes
Edit Files

Section III
Transfer Files
Processes
Signals



Section IV
Bash
Environment
Redirects & Pipes

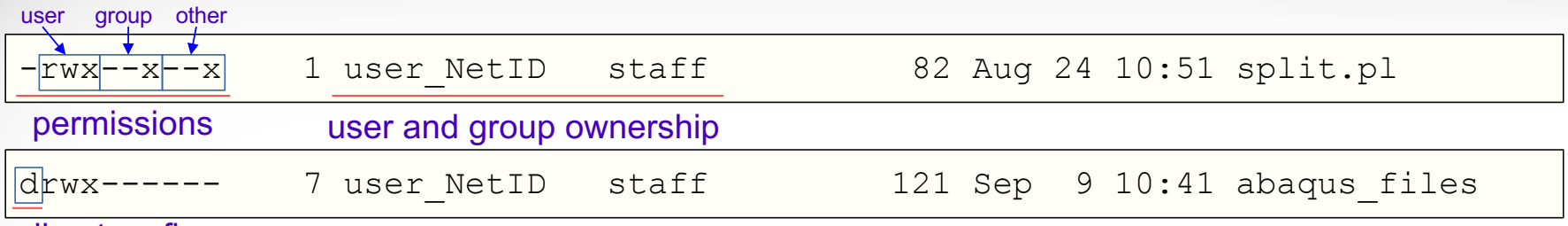
Section V
Other Topics

File Attributes: A look with *ls*

```
[user_NetID@ada ~]$ ls -l
total 37216
drwx------    7 user_NetID    user_NetID         121 Sep  9 10:41 abaqus_files
-rw-------    1 user_NetID    user_NetID        2252 Aug 24 10:47 fluent-unique.txt
-rw-------    1 user_NetID    user_NetID    13393007 Aug 24 10:40 fluent-use1.txt
-rw-------    1 user_NetID    user_NetID         533 Aug 24 11:23 fluent.users
drwxr-xr-x    3 user_NetID    user_NetID          17 May  7 16:56 man
-rw-------    1 user_NetID    user_NetID    24627200 Sep  9 10:49 myHomeDir.tar
lrwxrwxrwx    1 root     root           21 May 28 16:11 README -> /usr/local/etc/README
-rwx------    1 user_NetID    user_NetID         162 Sep  7 12:20 spiros-ex1.bash
-rwx--x--x    1 user_NetID    user_NetID          82 Aug 24 10:51 split.pl
drwxr-xr-x    2 user_NetID    user_NetID           6 May  5 11:32 verifyOLD
```

- file permissions
- hard link count
- user name
- group name
- file size in bytes
- file modification date
- file modification time
- file name

File Ownership and Permissions



Octal	Binary	Permissions
0	000	- - -
1	001	- - x
2	010	- w -
3	011	- w x
4	100	r - -
5	101	r - x
6	110	r w -
7	111	r w x

There are 3 permissions sets for each file:

- 1st set - user (the owner)
- 2nd set - group (to which file owner belongs)
- 3rd set - other (all other users)

For files:

- The *r* indicates read permission
- The *w* indicates writes permission
- The *x* indicates execute permission

For directories:

- The *r* indicates that a user can list contents
- The *w* indicates that a user can add/delete files
- The *x* indicates that a user can cd into directory
- The *x* also indicates that a user can execute programs

Edit File Attributes: the ***chmod*** command

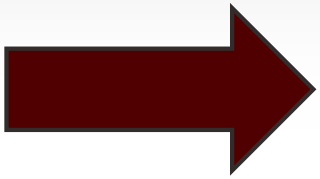
```
$ chmod [options] [permission mode] [target_file]
```

```
$ chmod 777 myFile.txt      ( the permissions will be set to rwxrwxrwx )

$ chmod o-x myFile.txt      ( the permissions will change to rwxrwxrw- )

$ chmod gu-x myFile.txt     ( the permissions will change to rw-rw-rw- )

$ chmod u+x myFile.txt      ( the permitions will change to rwxrw-rw- )
```

The **-R** option recursively applies the specified permissions to all files and directories within target directory

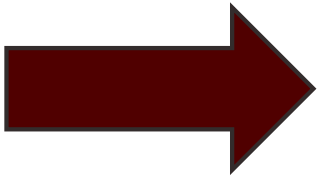
Overview

Section I
Connect
Navigate
View Files



Section II
Directories
Attributes
Edit Files

Section III
Transfer Files
Processes
Signals



Section IV
Bash
Environment
Redirects & Pipes

Section V
Other Topics

Editing File Contents

Files can edited with text editors if you have the correct permissions.

Open a file with *gedit*:

```
$ gedit [file name]
```

Open a file with *nano*:

```
$ nano [file name]
```

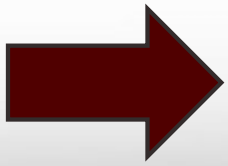
Open a file with *vi*:

```
$ vi [file name]
```

Graphic User Interface (GUI) options require X11 forwarding.

How do I choose?

1) What is installed?
2) What am I comfortable with?



New users usually like:

1) Text: *cat*
2) GUI: *gedit*

Windows to UNIX/Linux

Some users prefer to edit file on their local Windows machine.
Files are then transferred to the UNIX/Linux server.

Considerations:

1) How big are these files?
2) How often do the files update?
3) Is comfort worth inconvenience?

-IMPORTANT-

Text file edited with Windows contain different line terminators (CR/LF vs LF).
Use ***dos2unix*** to convert a DOS/Windows edited text file to UNIX format.

```
$ dos2unix myDOSfile.txt
```

Copying Files: the *cp* command

```
$ cp [options] [source] [target]
```

If source is a file, and…
- *target is a new name:* copy source and call it target
- *target is a directory:* copy source and place it in directory

If source is a directory, the ***-r*** option is used, and…
- *target is a new name:* copy source and contents into directory with new name
- *target is a directory:* copy source and place it in directory

Exercise:

```
$ cp hello.txt world.txt
$ ls
```

```
$ mkdir dir1
$ cp hello.txt dir1/f1.txt
$ ls dir1
```

Moving/Renaming Files: the ***mv*** command

```
$ mv [source] [target]
```

If source is a directory, and…
- *target is an existing dir:* source directory is moved inside target directory
- *target is a new name:* source directory is renamed to new name

If source is file, and…
- *target is an existing dir:* source file is moved inside target directory
- *target is a new name:* source file is renamed to new name

Exercise:

```
$ mv hello.txt save.txt
$ ls
```

```
$ mv save.txt dir1
$ ls
$ ls dir1
```

Deleting Files: the *rm* command

```
$ rm [options] [file name]
```

Commonly used options

- -i prompt user before any deletion
- -r remove the contents of directories recursively
- -f ignore nonexistent files, never prompt

-- BE CAREFUL --
YOU CAN PERMANENTLY DELETE EVERYTHING
"NEVER PROMPT" == NO CONFIRMATION

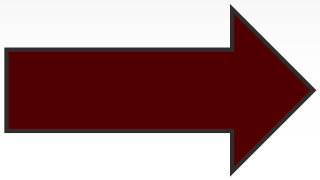
Exercise:

```
$ rm world.txt
$ ls
```

```
$ rm dir1
$ rm -rf dir1
$ ls
```

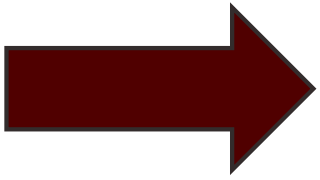
Overview

Section I
Connect
Navigate
View Files



Section II
Directories
Attributes
Edit Files

Section III
Transfer Files
Processes
Signals



Section IV
Bash
Environment
Redirects & Pipes

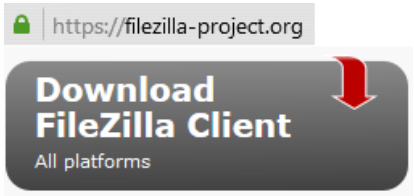
Section V
Other Topics

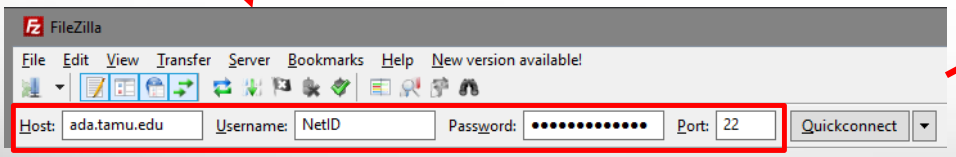
File Transfers Using FileZilla

The FileZilla Client:

1) Available on Windows, OS X, and UNIX/Linux
2) Allows permissions to be preserved or implied
3) Easy to use without previous experience

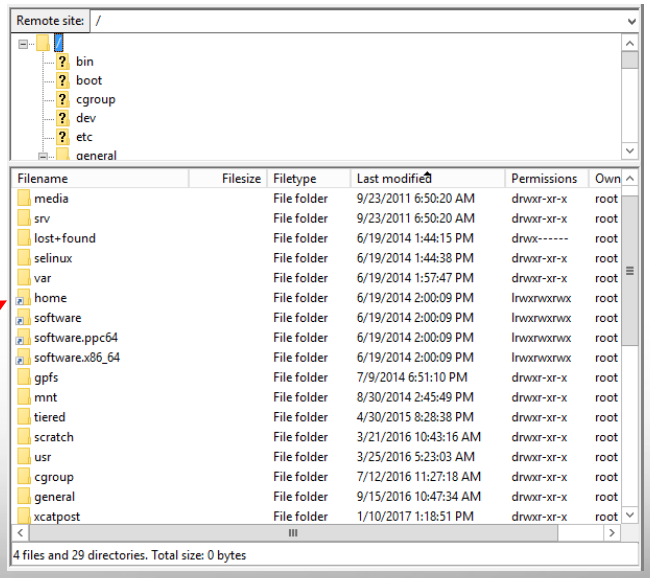
Download from:
https://filezilla-project.org





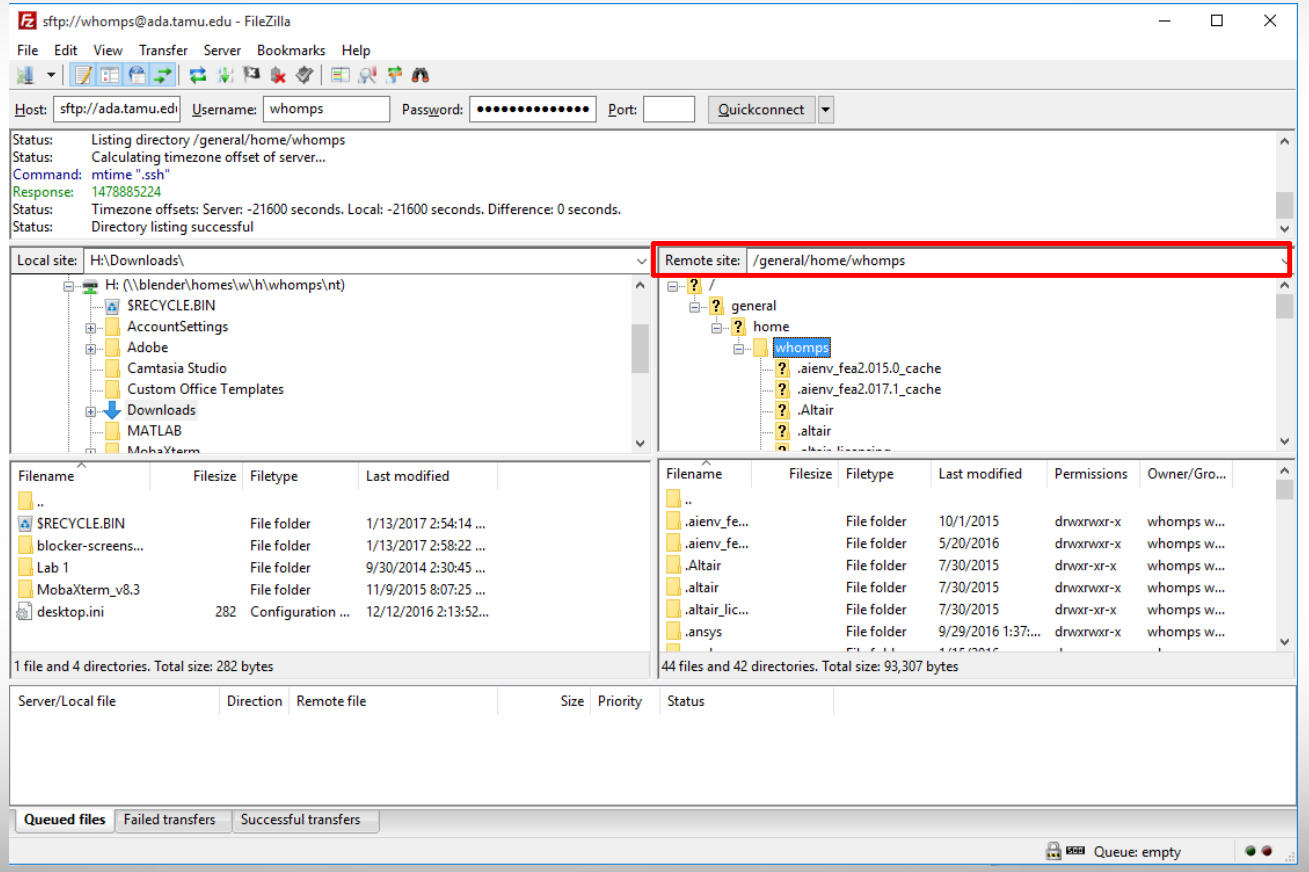
Connect with remote login

Drag and drop files



File Transfers Using FileZilla

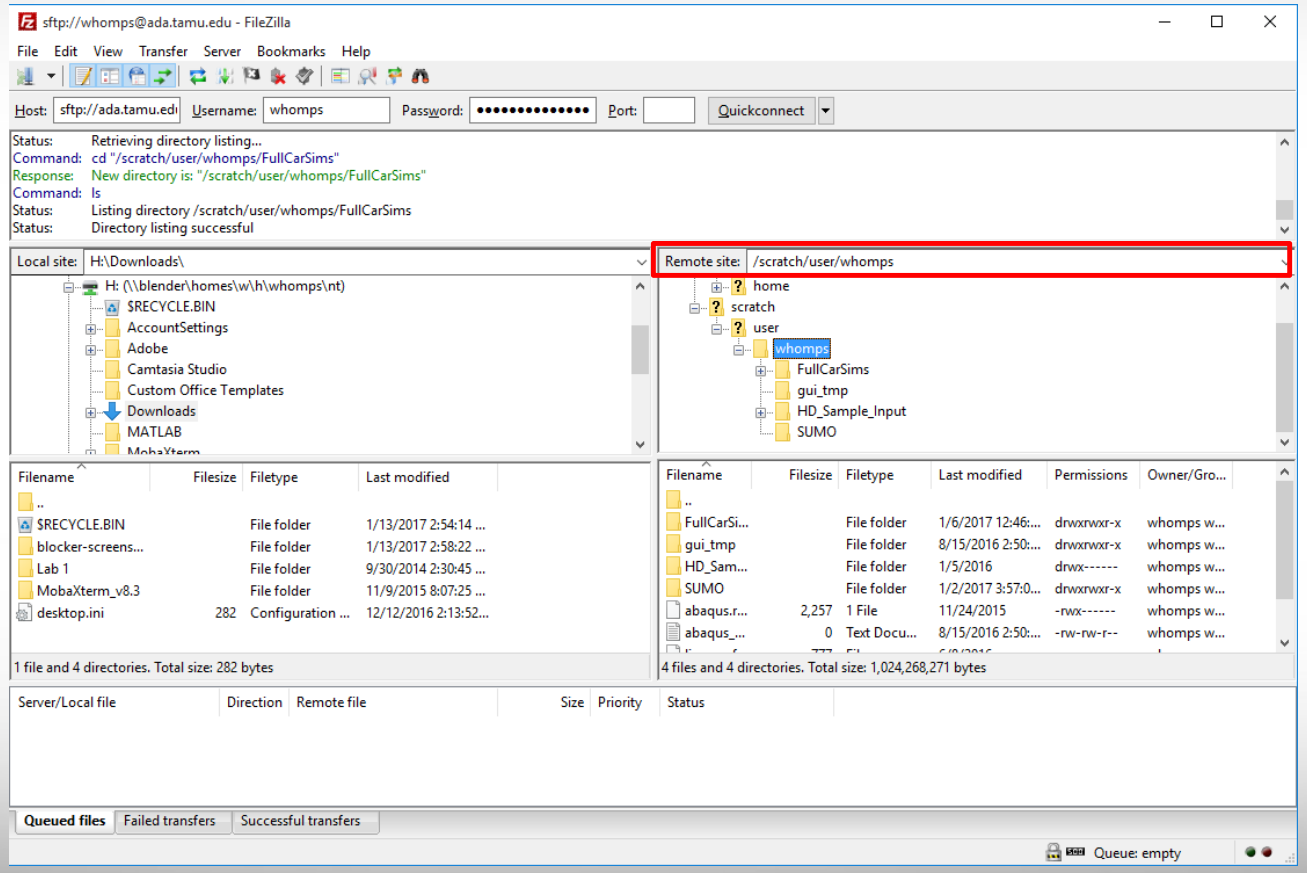
Local Directories *(TAMU H-Drive)*



Remote Directories *(Ada Home)*

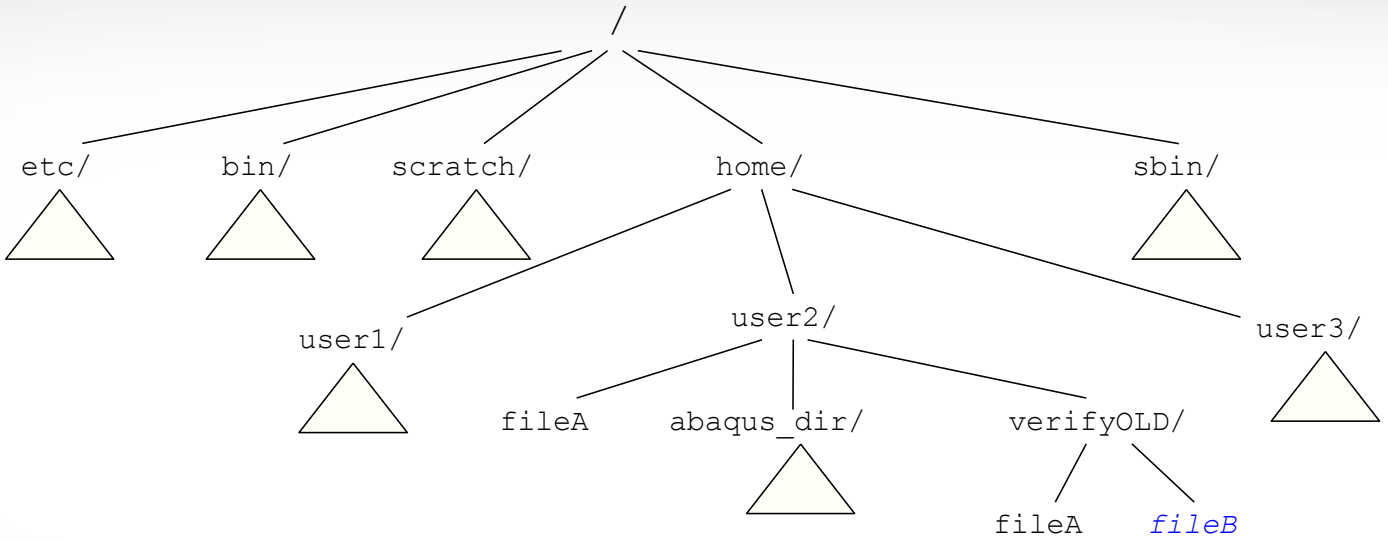
File Transfers Using FileZilla

Local Directories (TAMU H-Drive)



Remote Directories (Ada Scratch)

Absolute vs Relative Path



For file `fileB` under `/home/user2/verifyOLD`:

- The **absolute** (*full)* pathname is: `/home/user2/verifyOLD/fileB`
- The ***relative*** pathname is: `verifyOLD/fileB` if the current working directory is `/home/user2/`

Transfer Files Using *scp*

The ***scp*** command allows transfers to remote locations without using a GUI.

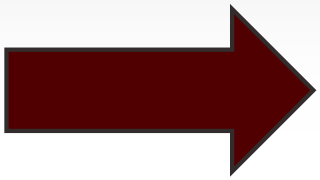
```
$ scp [[user@]host1:]filename1 [[user@]host2:]filena2
```

```
$ scp myfile1 user@ada.tamu.edu
$ scp myfile1 user@ada.tamu.edu:/scratch/user/[NetID]
$ scp user@ada.tamu.edu:myfile2 ~/Desktop/newFileName
$ scp -r user@ada.tamu.edu:dir3 local_dir/  (recursive)
```

Destination must be *addressable*.
A server is addressable – You can connect to it. You know the IP or hostname.
Your laptop might not be – No public IP? Firewall? Router?

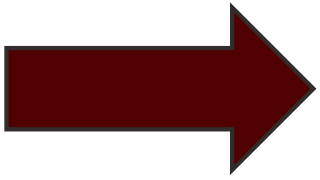
Overview

Section I
Connect
Navigate
View Files



Section II
Directories
Attributes
Edit Files

Section III
Transfer Files
Processes
Signals



Section IV
Bash
Environment
Redirects & Pipes

Section V
Other Topics

Processes, ***ps***, and ***top***

Process: A *program* that is loaded into memory and executed
Program: Machine readable code (binary) that is stored on disk

The ***ps*** command shows currently running processes.

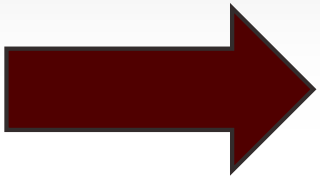
```
$ ps [options]
```

The ***top*** command displays real-time system resources usage.

```
$ top [options]
```

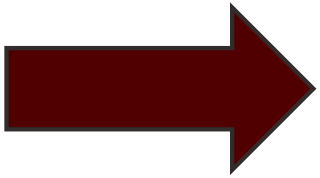
Overview

Part I
Connect
Navigate
View Files



Part II
Directories
Attributes
Edit Files

Part III
Transfer Files
Processes
Signals



Part IV
Bash
Environment
Redirects & Pipes

Part V
Other Topics

Process Communication Using Signals

A *signal* is a notification to a process that some event has occurred.

Various conditions can generate signals. Some of them include:

- The ***`kill`*** command
- Certain terminal characters (e.g. ^C is pressed)
- Certain hardware conditions (e.g. the modem hangs)
- Certain software conditions (e.g. division by zero)

After a process terminates, it returns an *exit status* to the parent process.

The *exit status* is an integer between 0 and 255.

- Exit status 0 usually means successful execution
- Non-zero exit status means some failure
- Exit status 127 usually means “command not found”
- If command dies due to a fatal signal, status is 128 + sig #

The *kill* Command

The ***kill*** command can generate a signal to the process specified by a PID.

```
$ kill [signal name] pid
```

The ***kill -l*** command lists all the signal names available.

```
$ kill -l
```

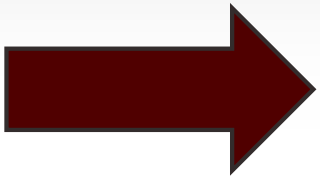
The ***kill -9*** command sends the (un-interruptible) kill signal.

```
$ kill -9 pid
```

kill can generate any type of signal, not just “kill” signals

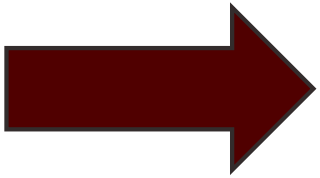
Overview

Part I
Connect
Navigate
View Files



Part II
Directories
Attributes
Edit Files

Part III
Transfer Files
Processes
Signals



Part IV
Bash
Environment
Redirects & Pipes

Part V
Other Topics

What is a Shell?

The *shell* is command language interpreter that executes commands. Commands can be read from stdin (keyboard) or from a file (script).

There are several variants of shell. Our clusters use Bash.

Bash has a number of start-up files that are used to initialize the shell.

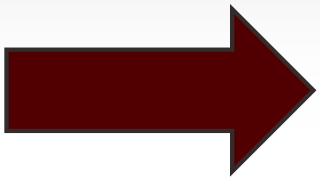
Initialization differs depending on whether the shell is a login shell, an interactive shell, or a non-interactive shell.

In general:

- When a user logs on, `/etc/profile` is sourced
- If it exists, `~/.bash_profile` is sourced
- If `.bash_profile` doesn't exist, but a `.bash_login` file does exist, it is sourced
- If even the `.bash_login` doesn't exist, but a `.profile` does exist, it is sourced

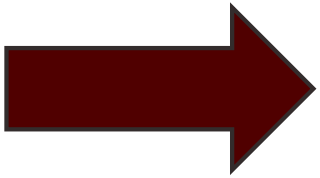
Overview

Part I
Connect
Navigate
View Files



Part II
Directories
Attributes
Edit Files

Part III
Transfer Files
Processes
Signals



Part IV
Bash
Environment
Redirects & Pipes

Part V
Other Topics

Shell Variables

Shell variables are name-value pairs created and maintained by the shell.

```
$ HELLO="Hello World!"
```

Variable values can be extracted by suffixing the name with "$"

```
$ echo $HELLO
```

Variable names must begin with an alphabetic or underscore character. The remaining characters can be alphanumeric or an underscore.

There are two types of variables: *local* and *environment*

- ***Local***: known only to the shell in which they are created
- ***Environment***: available to any child processes spawned from the shell from which they were created

Environment Variables

Environment variables can be thought of as global variables.

The ***export*** command makes variables available to child processes.

```
$ export NAME="user_NetID"
```

Some environment variables are set by the system upon login.

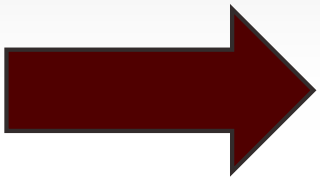
The ***export -p*** and ***env*** commands can be used to see the current variables.

```
$ export -p
```

```
$ env
```

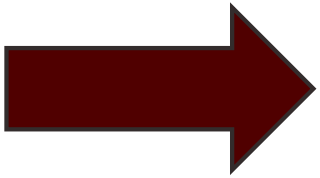
Overview

Part I
Connect
Navigate
View Files



Part II
Directories
Attributes
Edit Files

Part III
Transfer Files
Processes
Signals



Part IV
Bash
Environment
Redirects & Pipes

Part V
Other Topics

I/O Redirection

When an interactive shell starts, it inherits 3 I/O streams from the login program:

- *stdin* normally comes from the keyboard (fd 0)
- *stdout* normally goes to the screen (fd 1)
- *stderr* normally goes to the screen (fd 2)

There are times when the user wants to read input from a source and/or send output to a destination outside these standard channels.

This can be accomplished using I/O redirection.

```
$ echo "Hello!" > myTextFile.txt
```

Redirection Operators

<	redirects input
>	redirects output
>>	appends output
<<	input from *here document*
2>	redirects error
&>	redirects output and error
>&	redirects output and error
2>&1	redirects error to where output is going
1>&2	redirects output to where error is going

Pipes

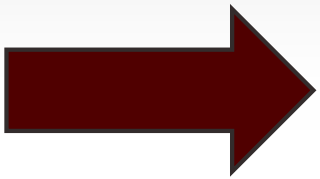
A pipe takes the output of one command and sends it to another.

"Left-Out is sent Right-In"
This can be done multiple times in a "pipeline"

```
$ who > tmp
$ wc -l tmp
38 tmp
$ rm tmp

(using a pipe saves disk space and time)

$ who | wc -l
38
$ du . | sort -n | sed -n '$p'
84480   .
```

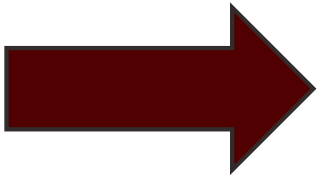
Overview

Part I
Connect
Navigate
View Files



Part II
Directories
Attributes
Edit Files

Part III
Transfer Files
Processes
Signals



Part IV
Bash
Environment
Redirects & Pipes

Part V
Other Topics

Aliases

An alias is a bash user-defined abbreviation for a command.

Aliases help simplify long commands or difficult syntax.

Aliases set at the command line are not inherited by subshells. They are normally set in the ~/*.bashrc* initialization file.

Aliases

The `alias` built-in command lists all aliases that are currently set.

```
$ alias
alias co='compress'
alias cp='cp -i'
alias mroe='more'
```

The `alias` command is also used to set an alias.

```
$ alias co=compress
$ alias cp='cp -i'
$ alias m=more
$ alias mroe='more'
```

The `unalias` command deletes an alias.
The `\` character can be used to temporarily turn off an alias.

```
$ unalias mroe
$ \ls
```

The '***source***' and Dot Commands

The source command is a built-in bash command and the '**.**' is simply another name for it.

Both commands take a script name as an argument. The script will be executed in the context of the current shell. All variables, functions, aliases set in the script will become a part of the current shell's environment.

```
$ source .bash_profile
$ . .bash_profile
```

The *find* Command

```
$ find [target dir] [expression]
```

```
$ find . -name "*.txt" -print

$ find . -newer results4.dat -name "*.dat" -print

$ find /scratch/user_NetID -mtime +2  -print

$ find /scratch/user_NetID -mtime -7  -print

$ find /tmp -user user_NetID -print
```

Comparing Files – *diff* and *cmp*

```
$ diff [options] FILES

# basic example

  $ diff file1 file2

# side by side comparison (long line truncated):

  $ diff -y file1 file2

# side by side comparison with screen width of 180 characters

  $ diff -y -W 180 file1 file2
```

```
$ cmp file1 file2
```

grep – Search pattern(s) in files

```
$ grep [options] PATTERN [FILES ...]
```

basic example

```
$ grep GoodData mydata.txt
```

search multiple matches

```
$ grep -e GoodData -e Important mydata.txt
```

excluding a pattern; show non-matched lines

```
$ grep -v NG mydata.txt
```

```
$ cat mydata.txt | grep GoodData
$ grep -v junk mydata.txt | grep -v NG
$ grep -e "^OUTPUT" mydata.txt
```

The *tar* Command

```
$ tar [options] [tar file] [file or dir name]
```

Used to “package” multiple files (along with directories if any) into one file suffixed with a .tar suffix by convention.

Commonly used options:

x	extract files from a tar
c	create a new tar
t	list the contents of a tar
v	verbosely list files processed
f	use the specified tar file
z	the tar file is compressed

The Backslash

The backslash (\) is used to escape a single character from interpretation.

```
$ echo Where are you going\?
Where are you going?
$ echo \\
\
$ echo '\\'
\\
$ echo '\$5.00'
\$5.00
$ echo "\$5.00"
$5.00
$ echo 'Don\'t you need $5.00?"
>
>'
Don\t you need .00?
```

Single Quotes

Single quotes protect all metacharacters from interpretation. To print a single quote, it must be enclosed in double quotes or escaped with a backslash.

```
$ echo 'hi there
> how are you?
> when will this end?
> when the quote is matched
> oh'
hi there
how are you?
when will this end?
when the quote is matched
oh
$ echo Don\'t you need '$5.00?'
Don't you need $5.00?
$ echo 'Mother yelled, "Time to eat!"'
Mother yelled, "Time to eat!"
```

Double Quotes

Double quotes allow variable and command substitution, and protect any other metacharacters from interpretation by the shell.

```
$ name=user_NetID
$ echo “Hi $name, I’m glad to meet you!”
Hi user_NetID, I’m glad to meet you!
$ echo “Hey $name, the time is $(date)”
Hey user_NetID, the time is Mon Sep 13 12:15:34 CDT 2004
```

References

Here are some slides from TACC and LSU on the similar subject.

Linux/Unix Basics for HPC: October 9, 2014 (with video) [TACC]
https://portal.tacc.utexas.edu/-/linux-unix-basics-for-hpc

Express Linux Tutorial: Learn Basic Commands in an Hour [TACC]
https://portal.tacc.utexas.edu/c/document_library/get_file?uuid=ed6c16e9-bcbc-4b70-9311-5273b09508b8&groupId=13601

Introduction to Linux for HPC [LSU]
http://www.hpc.lsu.edu/training/weekly-materials/2015-Fall/intro-linux-2015-09-02.pdf