Introduction to Linux

Bash and Basic GNU/Linux and Unix Concepts

Expectations!

- UNIX and Linux are gargantuan topics that only come into focus with experience.
- Provide some basic concepts for users assumed to be familiar with MacOS or Windows (computers in general).
- Provide familiarity with Linux commands.
- Get started in understanding command line interface - CLI.

Class outline

- History of Linux
- Kernel and shells
- The bash shell
- Files and directories
- File ownership and permissions
- Essential Linux commands with exercises!
- Processes
- File transfer

History

Late 60’s through 1980’s

- UNIX came about as a result research at Bell Labs (Dennis Ritchie, Brian Kernighan, Ken Thompson, et al). Originally written in assembly language.
- Unics (Unix) was named in contrast to MIT’s Multics operating system.
- Berkeley Software Distribution (BSD), or Berkeley Unix derived from Bell Labs’ work due to government monopoly agreements.
- Unix led to the BSD family of operating systems in the 1990’s.

Bell Laboratories

History

Richard Stallman, in 1983

- Started the GNU (GNU's Not Unix!) project
- Open-sourced versions of standard suite of Unix utilities found in BSD
- GNU is also a software license
- Many of the core utilities used in Linux, BSD-derived UNIX and proprietary UNIX operating systems are from the GNU project
- Just about all commands in this lesson are as a result of the GNU project

What is “Open Source?”

- “Open source” refers to a family of software licenses where the source code is available to the public with little or no copyright restrictions
- BSD-style licenses make source code available to be distributed without restriction.
- GNU GPL (General Public License) style licenses require source code availability. Projects that include GPL code must make any alterations to that code available under the GPL.
- Free and Open Source Software “Free as in Speech, not Beer!”

History

Linus Torvalds, in 1991

- Released the first version of his Linux kernel.
- Started as a study in processor architectures while at the University of Helsinki, Finland, and to this day still has the authority on what gets included in the Linux kernel
- In 1992 adopted the GNU license and rapidly gathered developers
- Combined the GNU suite of utilities with a new operating system kernel (GNU/Linux)

History

- By the late 1990’s/early 2000’s GNU/Linux gained main-stream adoption
- Especially in research and academic circles due to structural similarities with the already established UNIX and BSDs
- Linux steadily increased its market share of commercial servers
- Became gained widespread desktop adoption
- Present on gadgets (e.g. Android and many other embedded devices, Tivo/DVRs etc.)
- Automotive Grade Linux (AGL) on 2018 Toyota Camry

BSD: 1990s – Current
Worthy of Mention

While Linux is great, the BSD family is also a solid choice for scientific use depending on application.

- FreeBSD
- OpenBSD
- NetBSD
- Many Specialized Appliances (pfSense, FreeNAS)
- Others…

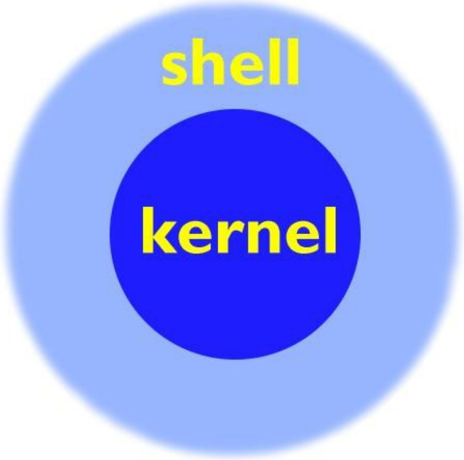
Popular Linux Distributions Include:

- Red Hat Enterprise Linux
- Fedora (upstream project for Red Hat)
- CentOS
- Debian
- Ubuntu
- Suse
- Linux Mint
- Gentoo
- For more distros visit: http://distrowatch.com/

Kernel Concepts

- In the operating system the "**`kernel`**" is the core software used to "*talk*" to computer hardware
- Is a core and modular system of drivers used to create a standardized environment for interfacing with hardware
- Kernel operates in its own memory or "kernel-space"
- Responsible for allocating memory (RAM) and CPU time, to system processes and user processes as well as interacting with files.

Your Shell



- On log-in, the system runs a *shell*
- A shell is the environment within which you will interface with the kernel via commands
- Shell determines the syntax for complex command-line operations and shell scripting
- The shell you're using is called "bash," the successor to the venerable "Bourne Shell" called "sh"
- BASH: "`B`ourne `A`gain `SH`ell" written by Brian Fox in 1989

Various Shells

- sh – the original UNIX shell
- bash – written as a replacement/extension of sh
- csh – C shell based on the C programming language developed in the late 1970s
- tcsh – enhanced version of C shell
- ksh – Korn shell developed in the early 1980’s, backward compatible with sh, but with some features of csh
- zsh – extended version of sh, developed in 1990
- dash – developed as replacement for ash in Debian

Linux accounts: Basic Credentials

- To access a Linux system, you need to have an account
- A Linux account includes the following:

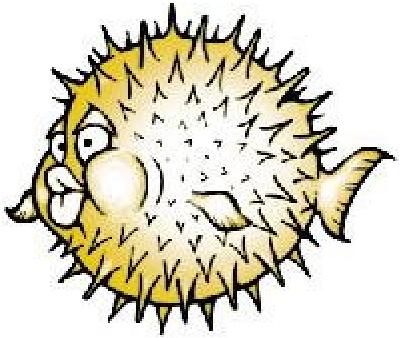
 - username and password

 - uid and gid

 - a home directory, which is where you are placed by default when you log in

 - a default shell

Using SSH to log in:



But First! Introducing OpenSSH:

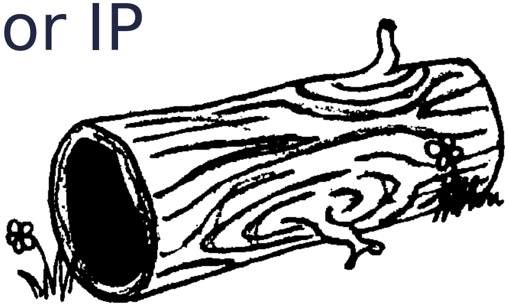
- SSH stands for "**S**ecure **SH**ell"
- Traffic over SSH is encrypted
- Developed as a secure alternative to the older **`RSH`** and **`Telnet`** protocols
- **SSH** supports a file-transfer subsystem - **SCP**
- SSH can act as an encryption layer for arbitrary network connections – SSH Tunnels

Lets get on Helix with our Ned Creds

- MacOS:
  - Finder -> Applications -> Utilities -> Terminal
  - Now type: “`ssh username@helix.nih.gov`”
  - At the prompt, enter the account password

- Windows:
  - Launch PuTTY. Under “Host Name (or IP address), type:

```
username@helix.nih.gov
```



  - …and click “Open”
  - At the prompt, enter the account password

More on shells

- What shell am I in? Typing “`echo $SHELL`” will show you!
- You should see ‘`/bin/bash`’
- Typing “`echo $0`” will also show your shell
- `$SHELL` and `$0` are shell variables (…more about variables later)
- List of available shells on the system can be displayed by typing “`chsh --list-shells`”
- The chsh command can be used to change your default shell as well, but on Helix and Biowulf never change it to a shell that ends in LOCKED – you will lock yourself out of your account!

Shell preferences

- When you login, startup scripts are run to setup your environment
- For bash, you can customize your environment by adding or modifying environment variables and aliases in the .bashrc file in your home directory.
- Examples:

```
alias list=‘ls’
alias rm=‘rm -i’

export PATH=$PATH:/scratch/$USER
export EDITOR=/usr/bin/vim
export PS1="[\u@\h   \w   \# ]"
set -o noclobber
```

Common Linux commands

awk	allows manipulation of text
bg	place suspended job into background
cat	view contents of a file
cd	change directory
chmod	change permissions on a file/directory
cp	copy a file
cut	extract a field of data from text output
echo	output text to the terminal or to a file
emacs	editor
fg	bring suspended job to foreground
file	display file type
find	search for files
grep	search a file or command output for a pattern
head	view beginning of file
history	display list of most recent commands
less	scroll forward or back through a file
ln	create a link to a file
ls	list files in a directory
man	view information about a command
mkdir	make directory
more	scroll through file a page at a time
mv	change the name of a file (move)
nano/pico	text editors
printenv	display shell variables
ps	show current process information
pwd	print current working directory
rm	delete or remove a file
rmdir	delete or remove a directory
sed	stream editor
sleep	pause
sort	perform a sort of text
stat	display file status info
tail	view end of the file
touch	create an empty file or update timestamps
tr	character substitution tool
uniq	remove identical, adjacent lines
vi/vim	text editor
wc	print number of lines, words or characters
which	shows full path of a command
whoami	displays username

Our cast! (of characters)
aka Special/Meta Characters

Character	Name/Location
\	Backslash (above the enter key)
/	Slash (left of right shift key)
`	Back-tick (left of the number 1, above the tab key)
|	Pipe (shift-\)
[and]	Brackets (left of the backslash)
{ and }	Braces or “curly” brackets (shift-[and shift-])
< and >	Angle brackets (left of the right shift key)
~	Tilde (shift-~)
!, @, #, $, %,^, &, *, (,)	(!) Bang/exclamation mark, (@) at sign, (#) hash, ($) dollar/string, (%) percent, (^) caret, (&) ampersand, (*) asterisk/star, and the left and right parenthesis.

Linux Command Basics

- Linux commands are case-sensitive

ls is not the same as LS

- Linux commands may have options that come after the command that start with a “–” and followed by a letter or “- -” followed by a word:

```
$ ls –r
$ ls --reverse
```

- Linux commands may allow for arguments:

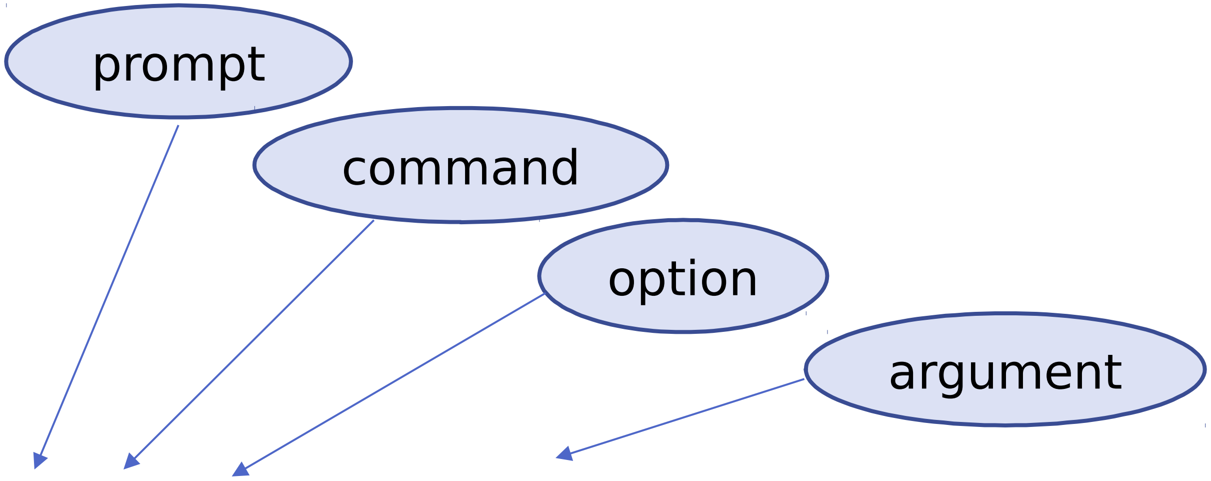
```
$ ls /scratch
```

- You can run more than one command on the same line by separating the commands with a semicolon (;)

```
$ date; ls
```

- Most Linux commands have a manual page or help to describe how they can be used….more about this later!

Linux Command by Parts



prompt

command

option

argument

```
$  ls  –r /home/$USER
```

Exercise #1: **`First commands`**

```
[username@helix ~]$
```

- Type “`whoami`”, press Enter – who are you?
- Type “`pwd`”, press Enter – where are you?

- Type “`echo $HOME`” – what does it show?
- Type “`echo $USER`” – what does it show?
- Type “`echo $PWD`” – what does it show?

- $HOME, $USER and $PWD are three more examples of shell variables as we saw earlier with $SHELL

Key Concepts: Files and Processes

In Unix, and by extension, Linux, everything is either a file or a process. Meaning everything can be interfaced via the file system(s).

- Files: text, data, documents, traditional files
- Directories: directories are special text files that contain a bunch of other files
- Devices: all disks, video hardware, audio hardware, processors, memory, USB ports – all hardware can be interfaced via files (usually in /dev)
- Processes: all running processes can be "seen" via the file system (in /proc) – each has a unique identifier (PID)

More on Files

- Each file (and directory) has a name
- The filename can contain letters, numbers and special characters
- Every file has a unique path to its location

  Example: /home/student2/read-write.txt
- A filename MUST be unique within a directory… though files with the same filename can exist in different directories
- Like Linux commands, filenames are case sensitive so a file named “myfile” and “Myfile” can co-exist in the same directory – the names are different.
- Filenames can be lengthy

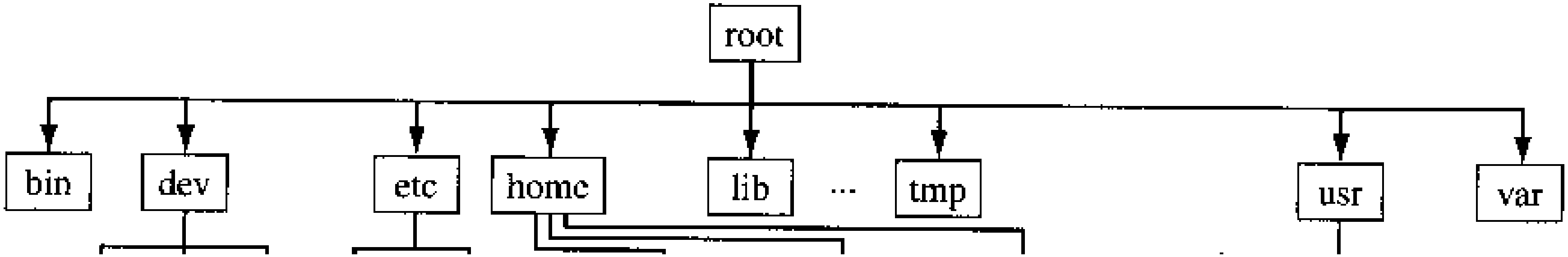
More on Directories

- A directory is a special type of file that can hold other files
- The "working directory" is the directory with which your shell is currently associated…where you currently are! When you first login, you will normally be in your home directory, /home/username

- Use the '**pwd**' command to **p**rint **w**orking **d**irectory

- Special directory notations:
- . refers to the current working directory
- .. refers to the parent directory (one level back – the parent directory of /home/username would be /home)

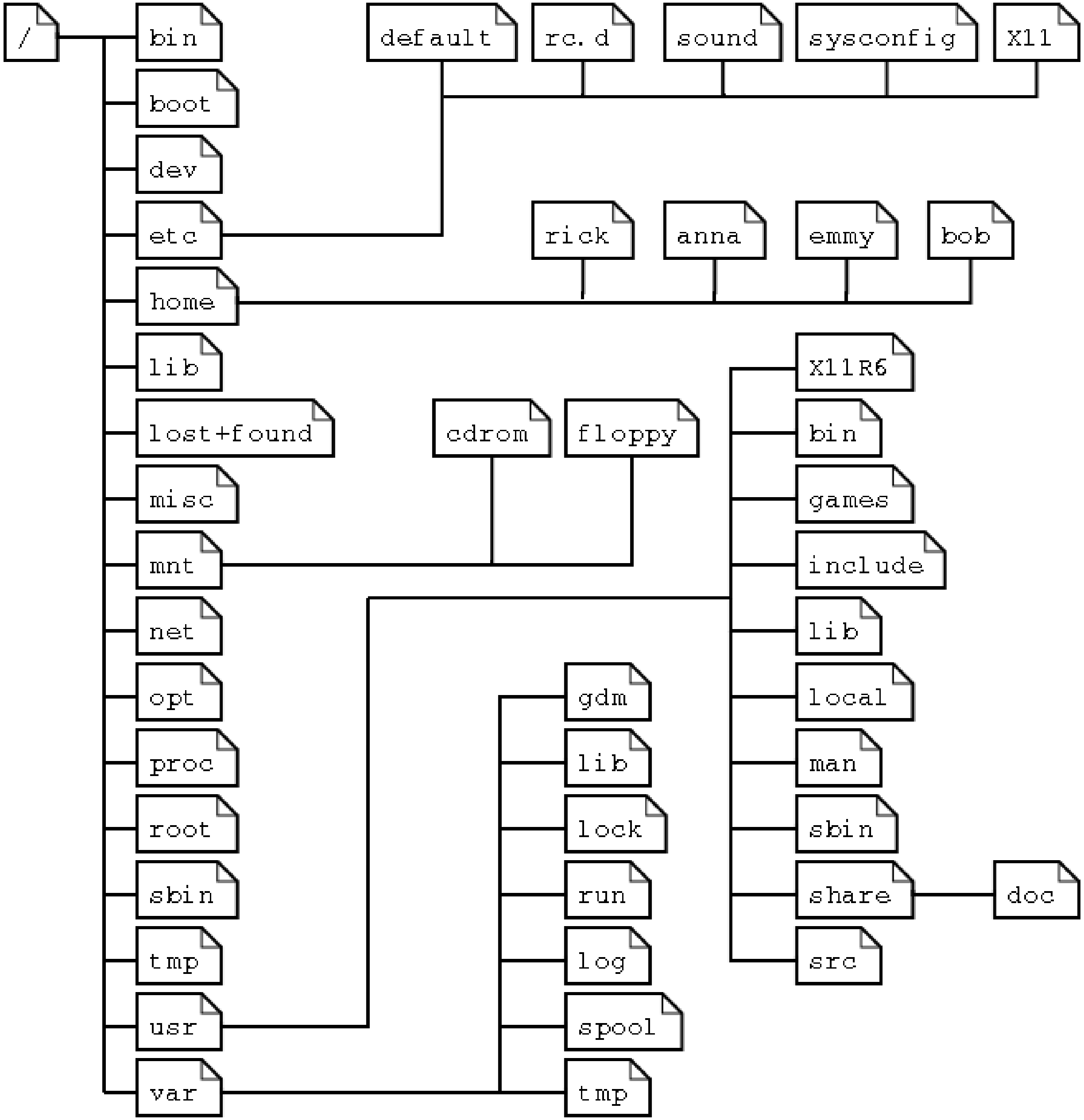
The File System Hierarchy



- Unix and Unix-like file systems are arranged in a tree structure, all with the same bottom level, called **“root” (/)**.
- Unlike Windows there are no drives, drive letters or any separate conceptual “space” for storage hardware.
- New hardware will come in the form of a “file system” attached (mounted) to some arbitrary point in the directory structure.

Directory Structure

- `/` root
- `/bin` bare essential commands
- `/boot` OS Kernels
- `/dev` hardware devices
- `/etc` system files, configuration
- `/home` home directories
- `/lib` Libraries needed by the system
- `/opt` 3rd party applications
- `/proc` Running processes
- `/sbin` administrative commands
- `/tmp` temporary space
- `/usr` operating system applications
- `/var` Logs, databases and other variable length stuff



File system layout

Common directory layout on a Linux system

Note that /var is frequently a mount-point to a separate file system. This is often true of /home, /tmp and /usr as well.

On Helix/Biowulf, /home is on a network file system as are data directories

Limit the effects of resource exhaustion by having separate mount points.

cd and ls commands

- The “cd” command is used to **c**hange **d**irectory location. Without an argument, “**cd**” takes you to your home directory
- The “**ls**” command is used to list the files in a directory. Like many Linux commands, it can take a number of “flags” as options to change the behavior of the command

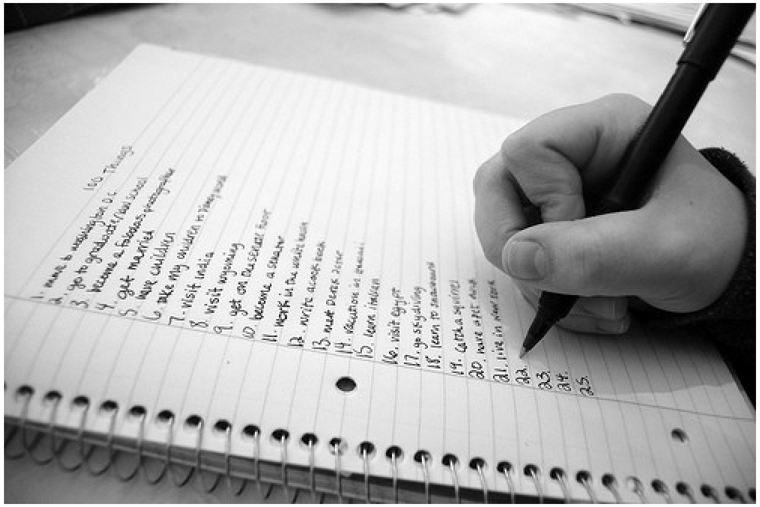
```
$ cd /home/$USER
$ ls
$ cd /etc
$ pwd
$ ls -l
$ cd ..
$ pwd
$ ls -l
$ cd
$ pwd
```

Exercise #2: “ls”, “cd” & “man” commands

- Type “`cd`” to get to your home directory
- Try “`ls -l`”
- Try “`ls -a`”
- Try “`ls -la`”
- How are the above outputs different?
- Try “`ls –l /data/classes/linux`”

The above shows how providing an argument to the ‘ls’ command displays the contents of a directory without first changing to the directory

- Type “`cd /scratch`” to change to the /scratch directory
- Try “`ls -lt`”
- How is the output ordered?
- Now try “`ls --help`”



- “man” displays a ***manual***

for a command

- Now type “`man ls`”
- Scroll with arrow keys
- “q” to quit

Users and Groups

- Users are associated with a unique user identification (UID) number that the system uses internally
- Users can be real people
- Users can be system entities
- Users can be herded via groups
- Groups also are associated with a unique group identification (GID) number by the system
- Groups allow multiple users to access/share the same files

Ownership & Permissions

- Linux systems are inherently multi-user environments that allow user**s** to create files, run programs and share data.
- Files and directories have two types of ownership – the user and group.
- A Linux group consists of one or more users
- Files and directories have three types of access permissions:

read permission (r)

write permission (w)

Symbolic Notation

execute permission (x)

- Every file and directory has permissions for three levels or entities of permissions:

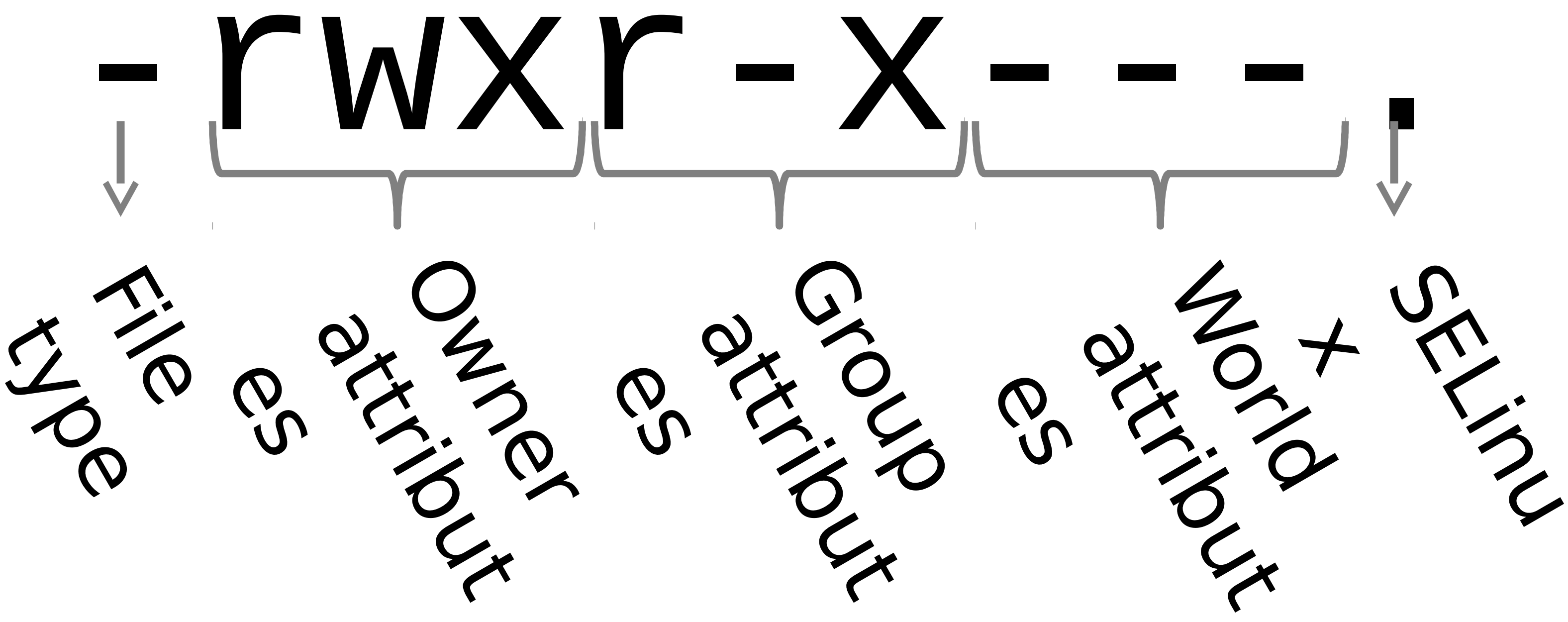
a) user or owner denoted by the letter "**u**"

b) group (one or more users) denoted by the letter "**g**"

c) others or world denoted by the letter "**o**"

Permissions triplets

Each triplet indicates the access permissions for that level – in the example below, the user/owner has read, write & execute permission, other group members only have read and execute permissions and all others have no access permissions.



```
-rwxr-x---.
```

File type

Owner attributes

Group attributes

World attributes

SELinux

Long List Output Explained (a little)

- From left to right:
- Unix permissions
- Hard links
- Owner
- Group ownership
- File size in bytes
- Modification date
- Name of file

```
$ ls -la
drwxrwx---  104 patkus    staff   110592 Jan  6 13:02 .
drwxr-xr-x 2510 root      root    196608 Jan  6 12:58 ..
-rw-r--r--    1 patkus    patkus    1051 May  8  2012 ad-week
-rwxr--r--    1 patkus    staff      239 May 11  2007 alias.pl
-rw-r-----    1 patkus    staff     1185 Jun 22  1998 bp.txt
-rwxr-xr-x    1 root      root     27320 Mar 29  2012 getpass.awk
-rw-rw-r--    1 susanc    staff    20529 Aug  7  2009 httpd.conf
-rwxr--r--    1 root      staff   136236 Sep 10  2013 memcon
drwxr-x---    2 patkus    staff     4096 Jun 24  2010 misc
drwx------    3 patkus    staff     4096 Jun 24  2008 test
-rwx------    1 patkus    staff      493 Feb 10  2009 unlock
-rw-r-----    1 patkus    staff       38 Oct 20  2010 world.c
-rwxr-x---    1 patkus    staff     6703 Jan  8  2013 world.exe
-rwxrwx---    1 patkus    staff     2350 May 22  2009 year.pl
lrwxrwxrwx    1 patkus    staff        7 Jan  6 15:30 year2 -> year.pl
```

Special Directories:

- .   is the current working directory
- ..   is the “parent” directory, one level “back”

Permissions described:

File Type:

- “-” regular file
- “d” directory
- “l” symlink
- “b” block device
- “c” character device
- “p” named pipe
- “s” socket

Permissions

- “r” read
- “w” write
- “x” execute

Special values:

- “x” executable
- “s” or “t”: executable and setuid/setgid/sticky
- “S” or “T”: setuid/setgid or sticky, but not executable.

Changing Permissions and Ownership

- Use ‘**chmod**’ to change the file permissions:

chmod [ugoa][+/-][rwx] filename

where u=user, g=group, o=others or world and a=all three

For example, to provide group read access to a file:

```
$ chmod  g+r  myfile
```

Or to remove file access to another than the owner or group members (in other words, others):

```
$ chmod  o-rwx  myfile
```

- The ‘**`chown`**’ command is used to change file ownership and the ‘**`chgrp`**’ command can change group ownership of a file. As a regular user, you can not change the ownership of a file, but you can change the group ownership if you are a member of the group to which you are changing the group ownership
- You can use the –R argument on any of the above to recursively make changes on a directory of files

Exercise #3 pre-exercise

We’ll talk about some of these commands shortly, but you each need to make a copy of files needed for the rest of class.

- First go to your home directory

```
$ cd  /home/$USER
```

- Make a ‘LinuxClass’ directory for yourself and go into that directory:

```
$ mkdir  LinuxClass
$ cd  LinuxClass
```

- Copy the files from the exercise file to your directory:

```
$ cp   -r   /data/classes/linux/*   .
$ ls  -l
```

- Create a shell variable to your class directory:

```
$ export  class=$PWD
$ cd  $class
```

Exercise #3: File Permissions

Read Permissions

The cat command displays contents of a file and returns your command prompt.

```
$ cd /home/$USER/LinuxClass
$ cat  read-write.txt
```

Change the read permission

```
$ chmod  u-r  read-write.txt
$ cat  read-write.txt
```

What happened? Now restore the read permission

```
$ chmod  u+r  read-write.txt
$ cat  read-write.txt
```

Execute Permissions

```
$ cd /home/$USER/LinuxClass
```

Run the myhostname file to see the system name

```
$ ./myhostname
```

Change the file permissions:

```
$ chmod  u-x  myhostname
$ ./myhostname
$ chmod  u+x  myhostname
$ ./myhostname
```

Change permissions on the directory dir-perms:

```
$ chmod  u-x  dir-perms
$ ls dir-perms
$ ls –l dir-perms
```

What happened and why?

Wildcards

- With many Linux commands, you can use wildcards to match characters
- The ‘*’ can be used to match zero or more characters

Examples:

```
$ ls  bear*
bears  bears7  bears_chicago
$ ls  *bear*
bears bears7 bears_chicago polarbears
```

- The ‘?’ can be used to match EXACTLY one character

```
$ ls  bears?
bears7
```

Special Keys

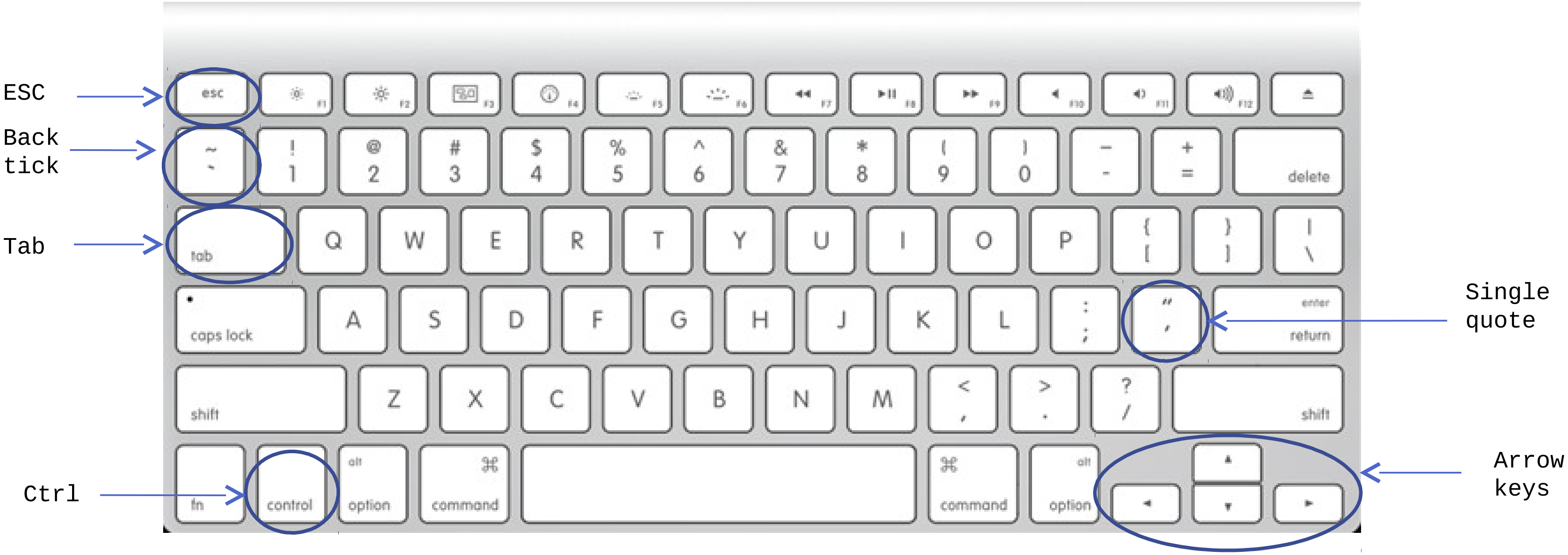
Tab Key allows for command auto-completion

Single Quote vs **Back Tick**

Arrow keys allow you to:

a) Move horizontally along the command to make changes without deleting and retyping everything

b) Move vertically through your history of previously run commands



Ctrl-c and ESC

- If you get into a situation where you can’t seem get back to a command line prompt, try hitting the Ctrl-C combination or the Esc key – often one of those will abort whatever you are currently doing.
- The Ctrl character is often represented by the ^

Let’s go home!

- The “~” is a special character that is short-hand for “/home/username”
- The environmental variable $HOME also stores the path of “/home/username”
- Several ways to get to your home directory

```
$ cd  ~
$ cd  $HOME
$ cd  /home/username
$ cd /home/$USER
$ cd
```

- But you can also use the “~” and $HOME as arguments with other commands:

```
$ ls  ~/tmp
$ ls  $HOME/LinuxClass
```

Absolute and Relative paths

- The starting “/” in the directory argument explicitly spells out a pathname – specifying an **absolute** or **full path**

- No leading “/” means you are specifying a path that is relative to the current working directory.

```
$ cd  /home/username


# Absolute path:
$ cd  /home/username/tmp


# Relative path:
$ cd  tmp


# Using ~
These do the same:
$ cd  ~/tmp
$ cd  /home/username/tmp
```

What is that file?

File

Tells us what type of file it might be – text, executable, PDF, jpg, gzip, tar, directory, etc.

```
$ file read-write.txt

$ file world.exe

$ file examples
```

Apropos

Apropos will search documentation for keywords. If you don’t know what man page to look at, use apropos to search for potentially related material

```
$ apropos  editor

$ apropos  “text editor”
```

Cat and Echo

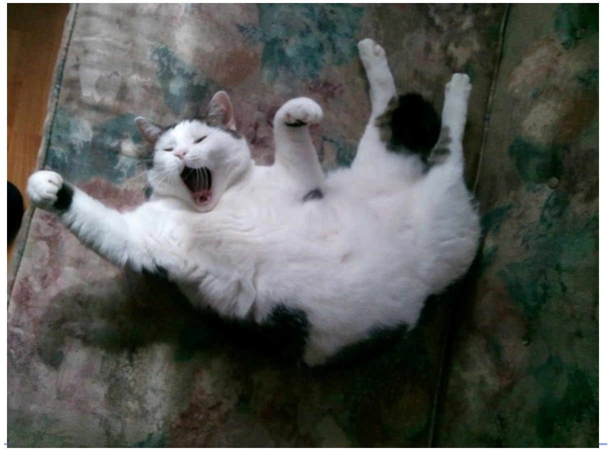
Use cat to display file contents to the terminal:

```
$ cat bears
$ cat bears7
$ cat bears bears7
```

“cat” is short for concatenate. The “cat” command takes one or more files and concatenates their contents to standard output.

Use echo to output arbitrary text to the terminal:

```
$ echo ‘Hello World!’
$ echo without single quotes
$ echo ‘output’ > Myoutput
```



Output Redirection to Files

Redirect output:

```
$ cat bears > Newbears
$ cat Newbears
$ cat bears bears7 > newbears
$ cat newbears

$ echo ‘Hi there!’ > greeting
$ cat greeting
```

Append files:

```
$ echo ‘Hi yourself!’ >> greeting
$ cat newbears >> greeting
$ cat greeting
```

Exercise #4: cat and echo

cat a file to view contents

```
$ pwd
$ cd  /home/$USER/LinuxClass
$ cat  lions
$ cat  tigers
$ cat  bears
$ cat  lions tigers > animals
$ cat  animals
$ cat  bears >> animals
$ cat  animals
```

Using echo

```
$ echo my name is Chris
$ echo “my name is $USER” > myname
$ cat myname
$ echo Hello $USER >> myname
$ cat myname
```

Symbolic links

Allows you to reference same file with different name or path – a symbolic link is a another file type.

```
ln  –s  <existing_file>  <file_link>
$ cat Capitals
$ ln  –s  Capitals  Capslink
$ ls  –l  Cap*
$ cat Capslink
```

Example:

```
$ ln  –s /home/$USER/LinuxClass/examples/tmp/colors color-pairs
$ ls –la color-pairs
$ cat /home/$USER/LinuxClass/examples/tmp/colors
$ cat color-pairs
```

Creating and deleting files/directories

Using touch and mkdir

To create an empty file, use the touch command:

```
$ touch  emptyfile
```

You can also create a file using an editor such as pico, nano, vi or emacs:

```
$ pico  penguinfile
```

To create a directory:

```
$ mkdir  Mydirectory
$ mkdir  2017
$ mkdir  –p  2017/Jan/stats
```

Using rm to remove files and directories

To remove a file:

```
$ rm emptyfile
$ touch myFile
$ chmod  u-rwx  myFile
$ rm myFile
$ rm –f myFile
```

To remove a directory:

```
$ rmdir Mydirectory
$ rm –r 2017
```

rmdir only works if the directory is empty!

Dangerous:

```
$ rm  –rf  *
```

Exercise #5: Creating and deleting files

Creating a file and directory

```
$ cd  /home/$USER/LinuxClass
$ echo  ‘I love genomic research!’  >  science
$ touch  science_project
$ mkdir  scienceclass
$ ls  –ld  science*
```

Deleting a file and directory

```
$ rm  science*
$ ls  –ld  science*
```

What happened?

```
$ rmdir  scienceclass
```

Displaying/Editing Files using the pagers **`more`** or **`less`**

"more" or "less"

"head" or "tail"

```
$ more  colleges.txt


$ less  colleges.txt


$ head  colleges.txt
 - prints out the first 10
lines by default.  Can use
the –n argument to specify
the number of lines


$ tail  –20  colleges.txt
 - prints out the last 20
lines
```

Text editors:

Good simple editors:

- pico (pine composer)
- nano (pico clone)

Good choices for editing in a terminal are:

- "vim" (vi-improved)
- "emacs" (**E**ditor **MAC**ro**S**)

pico – a simple editor

$ pico filename

The essentials:

- Just start typing – can use arrow keys to position, backspace or delete key to delete characters to the left
- Keystrokes for basic commands at bottom of the screen
- ^G – help screen (^C to exit help)
- ^O – save the file
- ^W – search for a string
- ^X – exit pico

```
$ pico  --help
```

Exercise #6: Editing a file using pico

```
$ cd  /home/$USER/LinuxClass
$ pico bashrc
```

1) Using the RIGHT arrow key, position the cursor at the end of the first line,
2) Use the Backspace key to remove the bracketed text from the first line and then simply start typing $HOME after the colon
3) Hit Ctrl-W (to search), type PICOPATH and hit Enter – this should place you on the last line
4) Hit Ctrl-E to get to the end of the line
5) Use the Backspace key to remove everything after the ‘=‘ sign and type ‘/usr/bin/pico’
6) Use the up & the right arrow keys to get to the @ on the 2nd line
7) Backspace to remove <USERNAME> and type your username
8) Use the down arrow key to get to the 3rd line
9) Hit Ctrl-K to cut the 3rd line
10) Hit the Up arrow to get to the 1st line & Ctrl-A to get to the start of the line
11) Hit Ctrl-U to paste the text – the 3rd line should now be the first
12) Hit Ctrl-X to exit – type Yes to save the file when prompted and hit Enter when prompted for the name

```
$ cat bashrc
```

Moving /Copying files

Move (mv)

Can be used to move a file or rename a file (some Linux versions have a 'rename' command, but not all):

```
$ touch football
$ ls -l football
$ mv football volleyball
$ ls -l *ball
$ mkdir sports
$ mv sports sportsNEW
$ ls -ld sports*
$ mv volleyball sportsNEW
$ ls -la sportsNEW
```

Copy (cp)

To copy a file or directory, use the cp command:

```
$ echo 'Goal!!' > soccer
$ cp soccer soccerball
$ ls -la soccer*
$ cp -p soccer soccerball
$ ls -la soccer*
$ cp -p soccer sportsNEW
```

Archival copy:

```
$ cp -a 2012tax 2012save
$ cp -pr 2012tax 2012save
```

Exercise #7: Moving/Copying Files

Move (mv)

```
$ cd  /home/$USER/LinuxClass
$ touch  baseball
$ mv  baseball  hockey
$ ls  -la  baseball
$ ls  -la  hockey
```

Now create a directory path using mkdir with the -p option:

```
$ mkdir  -p  2012/tax/forms
$ mv  2016  2017
$ ls  -Rl  2016
$ ls  -Rl   2017
```

Copy (cp)

```
$ cp  hockey  icehockey
$ mkdir  hockeypuck
$ mv  icehockey  hockeypuck
$ cp -pr hockeypuck hockeystick
$ ls  -la  hockey*
$ ls  -la  *hockey

$ cp  2017  2016
```

What did you see?

```
$ cp  -r  2017  2016
```

Archival copy:

```
$ cp  -a  2017  2018
```

grep – pattern matching search of a file

“grep” – global/ regular expression/ print

```
$ grep  cat  nonsense.txt
$ grep  dog  nonsense.txt
$ grep  -i  dog  nonsense.txt
$ grep  -v  dog  nonsense.*
$ grep  oc  nonsense.txt
$ grep  ^oc  nonsense.txt
$ grep  oc$  nonsense.txt
```

grep is a powerful tool. Use it (as well as egrep...extended grep)

```
$ grep  --help
$ man  grep
```

find – where are my darn files!

find - used to locate files based on various criteria

```
$ find  [path_to_search]  [expression/options]
$ find  .   –name  “*.txt”
$ find  /home/$USER/LinuxClass  –iname  “capital*”
$ find  /home/$USER/LinuxClass  –type  f  -mmin  20
$ find  /home/$USER/LinuxClass  –type  f  -mmin  -20
$ find  /home/$USER/LinuxClass  –type  f  -mmin  +20
$ find  /home/$USER/LinuxClass  –type  f  –mtime  1
$ find  /home/$USER/LinuxClass  –name  “*.bak”  –delete
$ find  .  –name  “*.txt”  –exec  ls  -la  {}  \;
$ man  find
```

Exercise #8: Using find

- Let’s use the find utility to look for files in your home directory
- Make sure you are in your class directory:

```
$ cd ~/LinuxClass
$ pwd
```

- Using ‘find’, locate the file named ‘colors’
- Using ‘find’, locate the files that **contain** the word ‘bear’ where the match is **case insensitive** (hint: -iname option)
- Using ‘find’, locate the files that were modified LESS than 90 minutes ago (hint: -mmin option)
- How many files did you find for each?

wc - what’s in that file?

“wc” (word count)

```
$ wc  nonsense.txt
 19 95 505 nonsense.txt
```

Output shows the number of lines, words and characters in the file

Can use argument to only get one of the three values:

```
$ wc  –l  nonsense.txt
$ wc  –w  nonsense.txt
$ wc  –m  nonsense.txt
```

```
$ wc  -help
```

uniq – show or remove duplicate lines

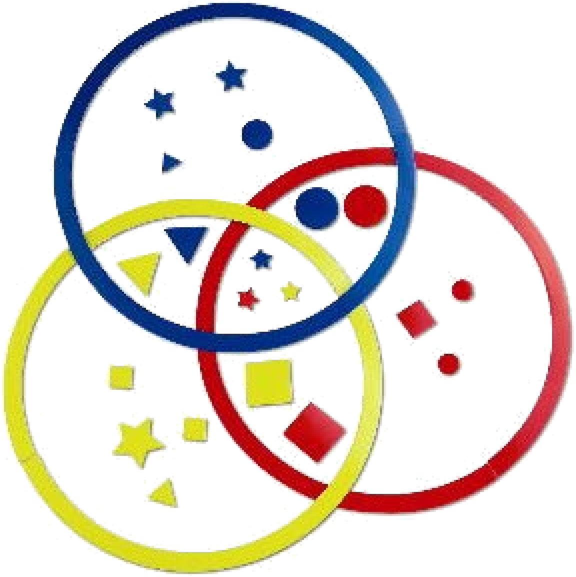
- uniq – show either unique or duplicate **consecutive** lines in a file or output. Default behavior is to merge adjacent matching lines into one, but can be used to print just the matching lines or provide a count of matching lines…most effective with the sort command

```
$ uniq  bears     # will show all unique lines
$ uniq  -d  bears  # show only duplicate lines
$ uniq  -c  bears  # show a count of each unique line
```

Sorting

Sort command

“sort” can be used to read a file, sort the contents and output to the terminal



```
$ cat  baseball.txt

$ sort  baseball.txt

$ sort  –r  baseball.txt

$ sort  –b  -k2
baseball.txt

$ sort  –bn  -k2
baseball.txt

$ sort  –bnr –k2
baseball.txt

$ sort  –help
```

Pipes (redirect to other processes)

Much like you can write output to files, you can write or “pipe” output to other commands using pipes “|”

```
$ cat college1 | sort | uniq
$ cat college2 | sort | uniq
$ cat college1 college2 | sort | uniq  -c
```

Write to a file at the end:

```
$ cat college1 college2 | sort | uniq |grep  ^B > Colleges
```

Exercise #9: sort, grep and redirection

- cd /home/$USER/LinuxClass
- Look at the contents of two files, grocery**1** and grocery2 (use cat command)
- Combine the two files using the cat command and then use the sort and uniq commands to get a list of sorted, unique items for the grocery list
- Now redirect the output to a file named grocery3
- Use the wc command to determine how many unique items are on the list (in the grocery3 file).
- Use grep and wc to determine how many items in the grocery3 list start with the letter ‘c’

Exercise #9 continued

- $ cat grocery**1**
- $ cat grocery2
- $ cat grocery**1** grocery2 | sort | uniq
- $ cat grocery**1** grocery2 | sort | uniq | wc –l

You should have 32 items

- $cat grocery**1**   grocery2 | sort | uniq > grocery3
- $ grep ^c grocery3

7 items start with the letter c

Other useful commands

- history – displays a history of commands – allows an easy way of running a command again without having to type it out again

- alias – list aliases or create a new one for another cmd

```
Example:
$ alias hist=“history 20”
```

- cut – print out selected fields

```
Example:
$ cat famousdogs | cut –f1,4 –d:
```

date & cal

- date – prints the current date and time

```
$ date
Mon Apr 20 14:09:01 EDT 2017
$ date +“%D”
04/20/17
```

- cal – print the calendar for the current month or entire year

```
$ cal -y 2018
```

Input, Output and Error

- Commands can have an input and output
- STDIN or ‘standard input’ is input from the keyboard though we can have redirected input from a file
- STDOUT or ‘standard output’ is output going to the screen. We’ve already seen where we can ‘redirect’ the output of a command to a file or pipe it as the input to another command
- Commands may also produce errors such as ‘Permission denied’
- STDERR or ‘standard error’ is error output that goes to the screen by default

Input, Output and Error (cont)

- STDIN, STDOUT and STDERR have handles or numbers associated with each:

	Handle	
STDIN	0	Standard input
STDOUT	1	Standard output
STDERR	2	Standard error

```
$ chmod  ugo-r  /scratch/Class/exercises/dogfile2
```

- Can redirect the STDERR to a file:

```
$ grep  dog  $HOME/LinuxClass/exercises/dogfile*   2>  errors
$ grep  dog  $HOME/LinuxClass/exercises/dogfile*   2>  /dev/null
```

- Can redirect BOTH the STDOUT and STDERR to a file:

```
$ grep  dog  $HOME/LinuxClass/exercises/dogfile*  >  out_errs  2>&1
```

or

```
$ grep  dog  $HOME/LinuxClass/exercises/dogfile*  &>  out_errs
```

Putting commands together

Read in from a file with input redirection, do some stuff and output to another file:

```
$ sort -r < foodfile | grep -i ^s > s-foods
```

- Program first, then arguments, then any file I/O
- Most programs will read from standard input (stdin) if no file is specified in arguments

awk – text manipulation

- In awk, lines are broken down into fields that are represented by variables $1, $2, $3, etc
- ‘print $1’ will print the first field
- Let’s look at an example:

```
$ cat  hare_tortoise
The hare beat the tortoise handily.
```

- We can change the ordering of words using awk:

```
$ cat hare_tortoise | awk ‘{print $1,$5,$3,$4,$2,$6}’
The tortoise beat the hare handily.
```

- You do NOT have to use all of the fields – can pick and choose as needed

sed – stream editor for pattern matching and modification

- In sed, one can do text pattern matching and modification

```
$ cat  hare_tortoise
The hare beat my tortoise handily.
```

- If we want to change the word ‘beat’ with the word ‘defeated’:

```
$ cat hare_tortoise | sed  ‘s/beat/defeated/g’
The hare defeated my tortoise handily.
```

- The trailing ‘/g’ at the end of that command indicates that the **change is to be done globally**...without it, only the first occurrence of the word in the file will be changed.

- We can use both awk & sed on the same command line:

```
$ cat hare_tortoise | awk ‘{print $1,$5,$3,$4,$2,$6}’ |
sed ‘s/beat/defeated/g’
The tortoise defeated my hare handily.
```

tr – allows character substitution or translation

- With tr, characters can used to translated – often used to change case of letters…for example:

```
$ echo ‘I love Linux!’ | tr “a-z” “A-Z”
I LOVE LINUX!
```

More Linux Command Basics…Quotes

- Linux treats single, double and back quotes in commands differently
- Contents of a set of single quotes are treated as a string:

```
$ echo ‘$USER’
$USER
```

- Contents of a set of double quotes will have any included variables replaced:

```
$ echo “The home directory of $USER is $HOME”
The home directory of username is /home/username
```

- Contents of a set of back quotes or back ticks (on the upper left of the keyboard) are treated as a command and the output can be assigned to a variable:

```
$ NOW=`date`; echo $NOW
$ NOW=$(date); echo $NOW
Tue Oct 11 10:38:13 EDT 2017
```

Shell Variables

Variable assignment

Arbitrary assignment

```
$ MYWORLD=“Hello World”
$ echo $MYWORLD
```

With program output

```
$ RIGHTNOW=`date`
$ echo $RIGHTNOW
Wed Jan 30 14:12:28 EDT 2017
```

From a file

```
$ FILE=`cat nonsense.txt`
$ echo $FILE

$ echo $FILE | tr “ “ “\n” \
|sort | uniq
```

Shell Variables

Show all currently assigned variables

```
$ printenv
HOSTNAME=helix.nih.gov
TERM=xterm
SHELL=/bin/bash
HISTSIZE=500
SSH_CLIENT=165.112.93.227 49886 22
QTDIR=/usr/lib64/qt-3.3
QTINC=/usr/lib64/qt-3.3/include
SSH_TTY=/dev/pts/286
HISTFILESIZE=500
USER=patkus
…
```

Useful predefined and important variables

- `$HOSTNAME` System hostname
- `$USER` Your Username
- `$SHELL` Your shell
- `$HOME` Home directory
- `$PWD` Current directory
- `$PATH` Command paths

Shell Variable Scope

Variables are available only to your immediate shell environment by default.

```
$ MYVAR=“This is my var”
$ echo $MYVAR

$ printenv | grep MYVAR

Is MYVAR listed?
```

Exporting a variable makes the variable available to sub processes

```
$ MYVAR=“exported”
$ printenv | grep MYVAR
$ export MYVAR
$ printenv | grep MYVAR

Or

$ export MYVAR2=“exported2”
$ printenv | grep MYVAR
```

Your PATH

Execution path

In BASH, execution of a program happens when you enter the program name. Your PATH variable keeps you from having to enter the full path to the program

```
$ echo $PATH


$ which date
$ which whoami
$ which perl
```

Setting your PATH

```
$ PATH=/home/$USER:$PATH
$ export PATH
```

Or

```
$ export PATH=/home/$USER:$PATH


$ echo $PATH
```

Loops

“For” loops will traverse space-delimited data

```
$ FILE=`cat foodfile`
$ for n in $FILE
> do
> echo $n
> done

$ for n in `cat foodfile`; do
> echo $n
> done
```

Loop over the output of the seq command:

```
$ seq 1 10

$ t=0
$ for n in `seq 1 10`; do
> t=`expr $t + $n`
> echo $t
> done
$ echo $t
55
$ t=1
$ for n in `seq 1 10`; do
> t=`expr $t \* $n`
> done
```

Loops, part 2

Let’s combine contents of many named files into one

```
$ cd /home/$USER/LinuxClass/loops
$ cat nih1
.
.
$ cat nih9
```

We can combine the contents of the 9 nih named files into one using a loop:

```
$ for i in `seq 1 9`
> do cat nih$i >> NIH-ALL
> done
$ cat NIH-ALL
```

What is truth?

Equality: if, then, else, fi

```
$ BAD=“good”
$ if [ “$BAD” = “good” ]
> then
> echo “true”
> fi

$ if [ “$BAD” = “no good” ]
> then
> echo “true”
> else
> echo “false”
> fi
```

Existence

```
$ if [ -f $HOME/showvar ]
> then
> echo true
> fi

$ if [ -d $HOME ]
> then
> echo true
> fi
```

Processes

Show processes

Show your processes:

```
$ ps
$ ps –f
```

sleep is a delay or pause for specified number of seconds

```
$ sleep 5
```

```
$ sleep 25 &
$ ps -f
$ ps –f --forest
```

Show all processes:

```
$ ps –e
$ ps –ef --forest
$ man ps
```

More on Processes

Background and Foreground processes

- A command/job can be run in the background by adding ‘&’ to end of the command:

```
$ sleep 50 &
[1]+ Done       sleep 50
```

- **ctrl-z** suspends a running job/process
- **bg** allows you to resume a suspended job in the background and returns you to the command prompt
- **fg** allows you to resume a suspended job in the foreground until it completes
- **ctrl-c** interrupts or kills the currently running process

- **Warning:** Backgrounded processes will die when you log out of your session unless you use something like nohup or screen.

More on Processes

Detach and Reattach processes

```
Ctrl-z suspends an active job

$ sleep 300
[ctrl–z] (process is suspended)

$ bg
$ ps –f
$ fg
```

Killing a process

```
$ sleep 300
[ctrl-Z]
$ bg
$ ps
  PID TTY          TIME CMD
 6686 pts/0    00:00:03 bash
 8298 pts/0    00:00:00 sleep
 8299 pts/0    00:00:00 ps
```

(find the PID of the process you want to kill)

```
$ kill 8298
```

Processes: kill them

“kill” only requests that the program exit. Use a signal 9 to force it to exit

```
$ sleep 300
[ctrl-Z]
$ ps
  PID TTY          TIME CMD
 6686 pts/0    00:00:03 bash
 8298 pts/0    00:00:00 sleep
 8299 pts/0    00:00:00 ps
```

(find the PID of the process you want to kill)

```
$ kill -9 8298
```

- The kill command is slightly misnamed, what it actually does is send a *signal* to a process
- Most signals are interpreted by the application being signaled and thus behavior is consistent only by convention
- Using signal 9 is dangerous if used indiscriminately!

Processes: Make them Nicer

“nice” a process. This is useful for log-running, intensive processes that you don’t want to impact the system. Values for “n” can be between 0 (highest priority) and 19 (lowest priority).

```
$ nice –n 10 cat /dev/urandom > /dev/null &
$ top –u `whoami`
…
$ renice –n 15 –p [pid of “cat” command]

Now kill the process:
$ ps –ef|grep urandom
$ kill -9 [pid of urandom process]
```

Exercise #10: Using ‘kill’ on a process

- First start a ‘sleep’ process that will run in the background for 300 seconds:

$ sleep 300

- Type ‘**Ctrl-z**’ (the Ctrl and z keys together) to suspend the ‘sleep’ process
- Type ‘**bg**’ to unsuspend the ‘sleep’ process and have it run in the background.
- Check that the process is running by using the ‘**ps**’ command and note the pid, process identification number
- Using the ‘**kill**’ command with the pid of the sleep process from the previous step, terminate the sleep process
- How can you check that the sleep process is gone?

uptime

- **uptime** shows a summary of the system status

```
$ uptime
 14:39:46 up 14 days,  7:00, 305 users,  load average: 39.18, 40.68, 38.68
```

- Numbers shown from left to right:

- Current time

- Amount of the time the system has been up

- Number of users currently logged on

- The average system load for the past 1, 5 and 15 minutes

- The load is the usage of the system’s CPUs – a load of 1 corresponds to a full load of 1CPU

Who is doing what…using top

By default, top will produce continuous output about running processes

```
$ top
top - 16:19:54 up 28 days,  9:07, 255 users,  load average: 32.18, 32.79, 33.22
Tasks: 4749 total,   8 running, 4733 sleeping,   7 stopped,   1 zombie
Cpu(s):  9.6%us,  5.8%sy,  6.0%ni, 78.2%id,  0.2%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:  1058656848k total, 955041356k used, 103615492k free,    79064k buffers
Swap: 67108856k total,   547376k used, 66561480k free, 89619996k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
120202 johanesb  39  19  235m 180m 1432 R 96.4  0.0 170:21.86 merlin
252158 liqingli  39  19 58496  26m  756 S 95.0  0.0  17141:15 moe
170176 bozser    33  13  407m 117m 2588 S 60.1  0.0  62:30.33 ascp
218983 jrussler  20   0 18532 4704  872 R 22.3  0.0   0:00.38 top
127988 elliottm  39  19  223m 3544 1064 S 16.8  0.0 782:02.42 sshd
198816 wenxiao   20   0  4280  792  416 D 14.0  0.0  24:50.19 gzip
```

Hit ‘q’ to quit out of top

Looking at file system (disk) Space

To see local file system space:

```
$ df –l
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/vg_helix-root
                      51403396   8502228  40289968  18% /
tmpfs                529355640      2216 529353424   1% /dev/shm
/dev/sda2               495844    180879    289365  39% /boot
/dev/sda1               204580     33228    171352  17% /boot/efi
/dev/mapper/vg_helix-tmp
                      51606140    973788  48010912   2% /tmp
/dev/mapper/vg_helix-var
                      32253856  19349996  11265460  64% /var
```

To see numbers in human readable format:

```
$ df –lh
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/vg_helix-lv_root
                       50G  8.9G   38G  19% /
tmpfs                 505G  5.6M  505G   1% /dev/shm
/dev/sda2             485M  142M  318M  31% /boot
/dev/sda1             200M  256K  200M   1% /boot/efi
/dev/mapper/vg_helix-lv_tmp
                       50G  613M   47G   2% /tmp
```

All filesystems, including network file systems:

```
$ df –h
```

Directory size

Estimate file space use (du)

```
$ cd /home/$USER
```

Estimate a file size:

```
$ du LinuxClass/pbs_user_guide.pdf
```

Summary:

```
$ du -s LinuxClass
```

Summary in human-readable format:

```
$ du -sh LinuxClass
```

Default behavior:

```
$ du
```

Checking Quotas on Helix/Biowulf

“checkquota”

- The **checkquota** command will query all network storage devices to find the applicable quota(s) for your user
- This command is specific to Helix and Biowulf and is not available to Linux in general since it relies on information that is site-specific to this infrastructure.

```
$ checkquota
Mount                   Used       Quota   Percent    Files
/data:               94.2 GB    200.0 GB    47.12%    70424
/home:                5.2 GB      8.0 GB    64.71%   133607
mailbox:            347.9 MB      1.0 GB    33.98%
```

File Transfer

SCP, SFTP and clients

- SCP and SFTP are file transfer protocols that run over SSH, the same protocol that you used to log in
- They are very secure and encrypt both the log-in and content of any transfer

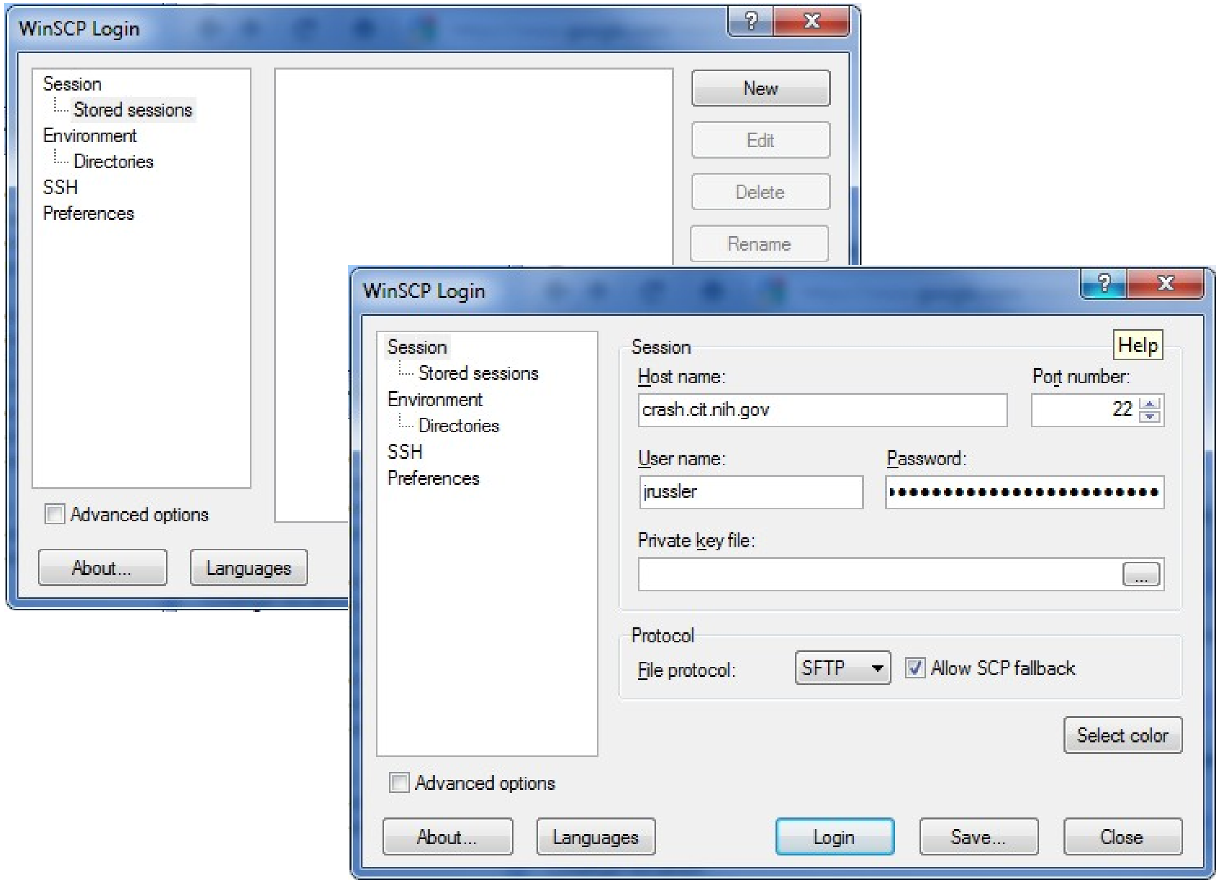
Clients

Linux/MacOS:

- “scp” secure copy
- “sftp” secure FTP
- “fuze-ssh” (Linux only)

Windows:

- WinSCP
- Filezilla
- Swish



Host name: crash.cit.nih.gov

Port number: 22

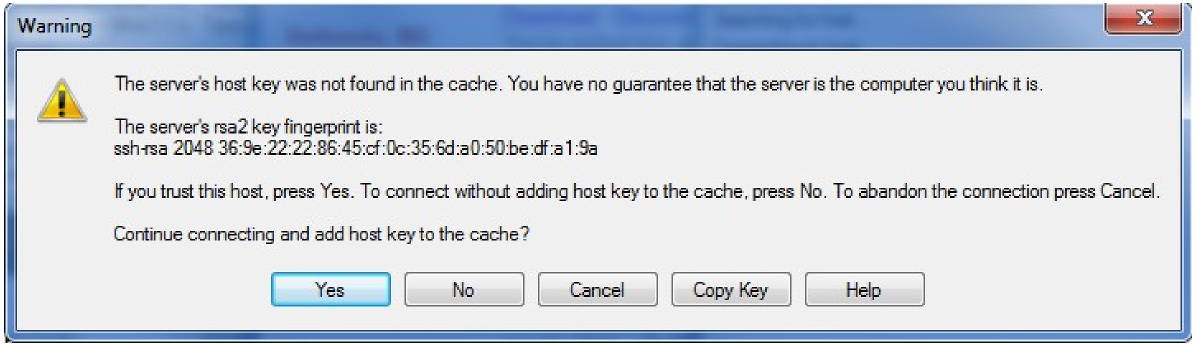
User name: jrussler

Password: ••••••••••••••••••••••••••

Private key file:

File protocol: SFTP

☑ Allow SCP fallback



The server's host key was not found in the cache. You have no guarantee that the server is the computer you think it is.

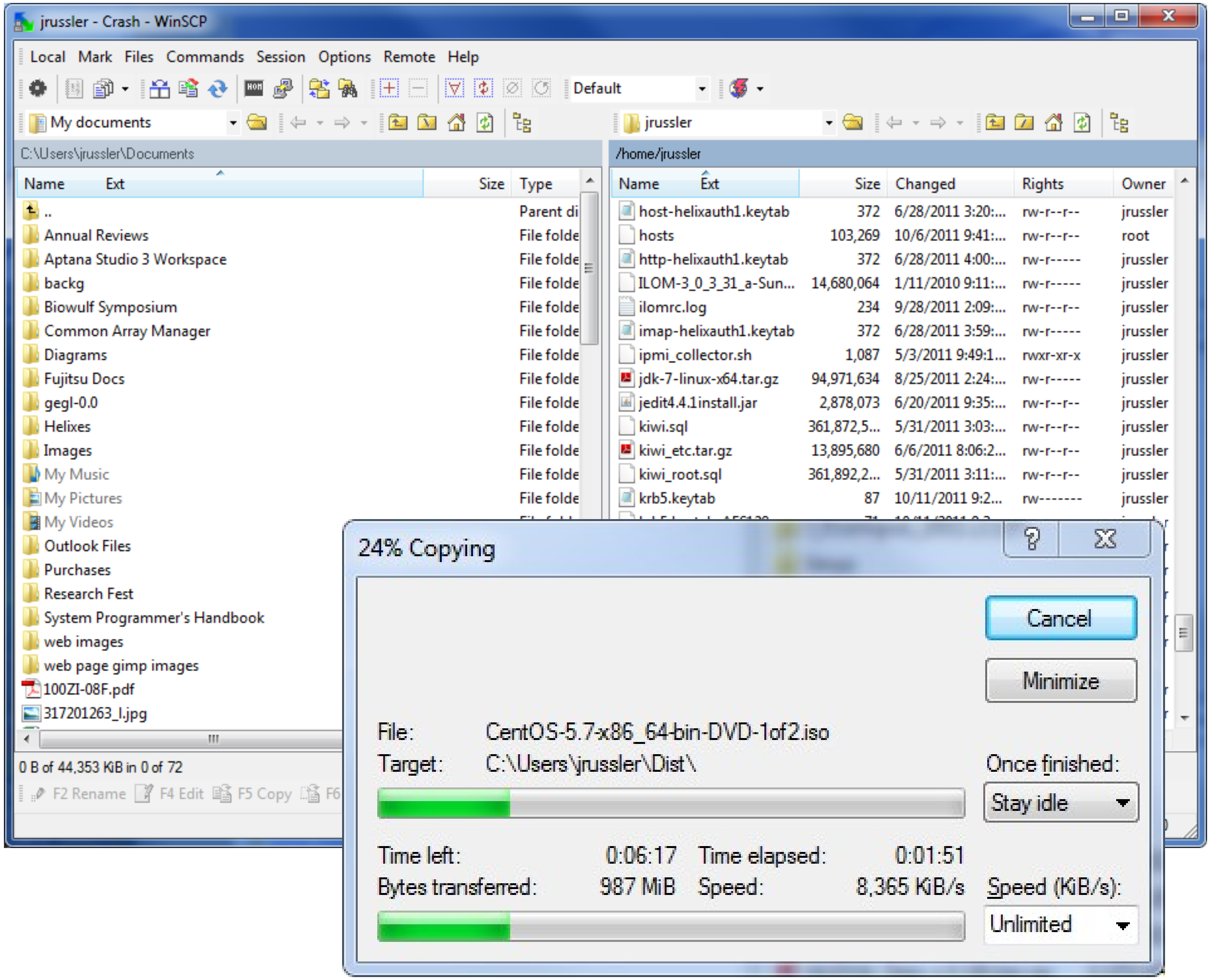
The server's rsa2 key fingerprint is:
ssh-rsa 2048 36:9e:22:22:86:45:cf:0c:35:6d:a0:50:be:df:a1:9a

If you trust this host, press Yes. To connect without adding host key to the cache, press No. To abandon the connection press Cancel.

Continue connecting and add host key to the cache?

WinSCP

- Start WinSCP
- Click “New”
- Enter the host name (i.e.: crash.cit.nih.gov)
- Fill in user name
- Leave password blank
- Click Login
- If this is the first time you’ve connected to this host, you’ll have to accept the host’s key


jrussler - Crash - WinSCP
Local Mark Files Commands Session Options Remote Help
Default
My documents
jrussler
F2 Rename F4 Edit F5 Copy F6
Cancel
Minimize

C:\Users\jrussler\Documents

Name	Ext	Size	Type
..			Parent di
Annual Reviews			File folde
Aptana Studio 3 Workspace			File folde
backg			File folde
Biowulf Symposium			File folde
Common Array Manager			File folde
Diagrams			File folde
Fujitsu Docs			File folde
gegl-0.0			File folde
Helixes			File folde
Images			File folde
My Music			File folde
My Pictures			File folde
My Videos			
Outlook Files			
Purchases			
Research Fest			
System Programmer's Handbook			
web images			
web page gimp images			
100ZI-08F.pdf			
317201263_l.jpg			

0 B of 44,353 KiB in 0 of 72

/home/jrussler

Name	Ext	Size	Changed	Rights	Owner
host-helixauth1.keytab		372	6/28/2011 3:20:...	rw-r--r--	jrussler
hosts		103,269	10/6/2011 9:41:...	rw-r--r--	root
http-helixauth1.keytab		372	6/28/2011 4:00:...	rw-r-----	jrussler
ILOM-3_0_3_31_a-Sun...		14,680,064	1/11/2010 9:11:...	rw-r-----	jrussler
ilomrc.log		234	9/28/2011 2:09:...	rw-r--r--	jrussler
imap-helixauth1.keytab		372	6/28/2011 3:59:...	rw-r-----	jrussler
ipmi_collector.sh		1,087	5/3/2011 9:49:1...	rwxr-xr-x	jrussler
jdk-7-linux-x64.tar.gz		94,971,634	8/25/2011 2:24:...	rw-r-----	jrussler
jedit4.4.1install.jar		2,878,073	6/20/2011 9:35:...	rw-r--r--	jrussler
kiwi.sql		361,872,5...	5/31/2011 3:03:...	rw-r--r--	jrussler
kiwi_etc.tar.gz		13,895,680	6/6/2011 8:06:2...	rw-r--r--	jrussler
kiwi_root.sql		361,892,2...	5/31/2011 3:11:...	rw-r--r--	jrussler
krb5.keytab		87	10/11/2011 9:2...	rw-------	jrussler

24% Copying

File: CentOS-5.7-x86_64-bin-DVD-1of2.iso
Target: C:\Users\jrussler\Dist\
Once finished: Stay idle
Time left: 0:06:17 Time elapsed: 0:01:51
Bytes transferred: 987 MiB Speed: 8,365 KiB/s
Speed (KiB/s): Unlimited

WinSCP Interface

- Left window is your local workstation, right window is the remote host
- Drag and drop files
- Navigate like a traditional explorer interface

Using OpenSSH (Unix/Linux/MacOS)

SCP files via command line:

Transfer a file:

```
$ scp helix.nih.gov:/tmp/file ~
```

Recursive transfer (whole directory)

```
$ scp –r helix.nih.gov:/tmp/dir ~
```

Preserve time stamps of the file being transferred:

```
$ scp –p helix.nih.gov:/tmp/file ~
```

From local host to remote.

```
$ scp ~/file helix.nih.gov:/tmp/
```

As usual

```
$ scp --help
$ man scp
```

Using SFTP

```
$ sftp helix.nih.gov
sftp> cd /tmp
sftp> get file
Fetching /tmp/file to file
/tmp/file  100% 2048KB   2.0MB/s   00:00
sftp> put file newfile
Uploading file to /tmp/file
file  100% 2048KB   2.0MB/s   00:00
sftp> exit

$ man sftp
```

File Transfer via Helixdrive

Network drive (Windows)

Helixdrive is available to users with Helix accounts:

- Open “Computer” from the start menu
- Click “Map Network Drive”
- Folder: helixdrive.nih.gov/username
- Click Finish
- If prompted, enter Helix username and password

Network Drive (MacOS)

- Menu Bar -> Go -> Connect to Server
- Server Address: smb://helixdrive.nih.gov/username
- Click “Connect”
- Check “Registered User”
- If prompted, enter Helix username and password

Exercise #11: Using scp

- Type ‘exit’ or ‘logout’ to log off from Helix and get back to your local system:

```
$ exit
```

- Use scp to copy the file read-write.txt from Helix to your local system:

```
$ scp
username@helix.nih.gov:/data/classes/linux/read-write.txt   .
```

- Advanced exercise….copy a whole directory:

```
$ scp -pr
username@helix.nih.gov:/data/classes/linux/examples  .
```

Cron

Cron: run a job whenever you want

- crontab -l
- crontab –e

```
--
15 3 * * * ~/script.sh >> ~/output 2> ~/error.log–
```

Runs “script.sh” at 3:15AM every day of every week of every month of every year.

```
* * * * *
```

- First number is the minute at which to run (0-59)
- Second is the hour (0-23)
- Third is the day of the month (1-31)
- Fourth is the month (1-12)
- Fifth is the day of the week (0-6), 0 is Sunday

Review

- History – Linux Torvalds, 1991
- Why Linux? Performance, functionality and portability
- Bash shell & shell variables
- Files and directories – permissions & ownership
- Linux file system
- Paths to files/directories
- Basic Linux commands to create & access files & directories
- pico editor
- sort, grep & find
- pipe & file redirection
- processes
- transferring data to and from a Linux system
- cron

Logging out

```
$ exit
```

Resources

- Linux Documentation Project: http://tldp.org/
  - **Introduction to Linux - A Hands on Guide**
  - **Bash Guide for Beginners**

- Helix Web Site: http://helix.nih.gov