Intro to Linux Shell

Bruno Mundim

March 13, 2019

Outline

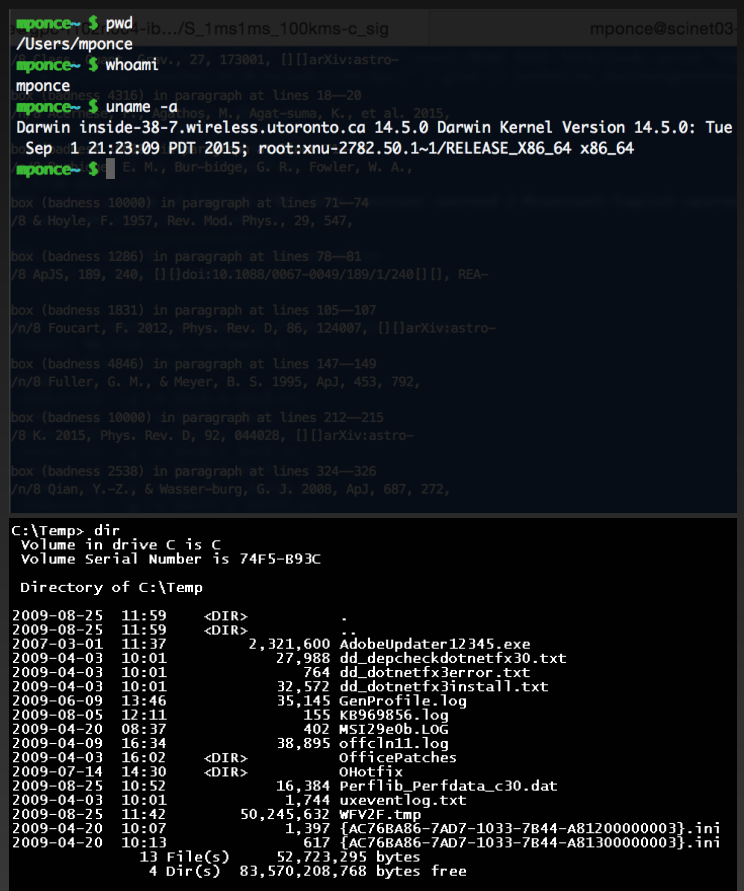
SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

What is a shell?

- A **shell** is a "*super*"-program
- It allows to run other programs
- All general purpose computers have a shell of some sort

What is a shell?

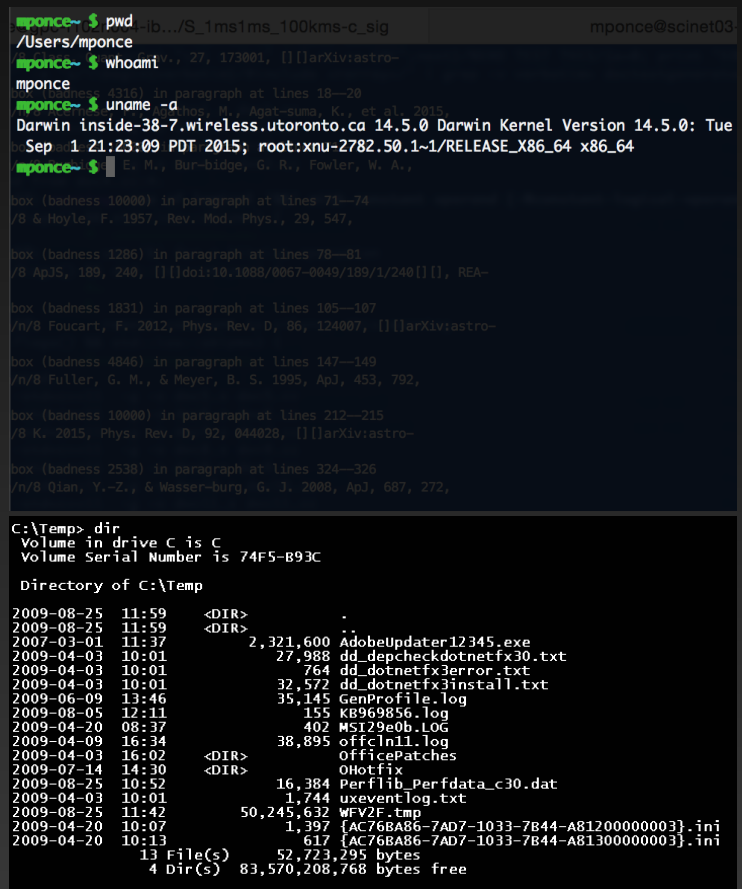
- A **shell** is a "*super*"-program
- It allows to run other programs
- All general purpose computers have a shell of some sort



What is a shell?

- A **shell** is a "*super*"-program
- It allows to run other programs
- All general purpose computers have a shell of some sort

Very well-known example, nowadays...



What is a shell?

- A **shell** is a "*super*"-program
- It allows to run other programs
- All general purpose computers have a shell of some sort

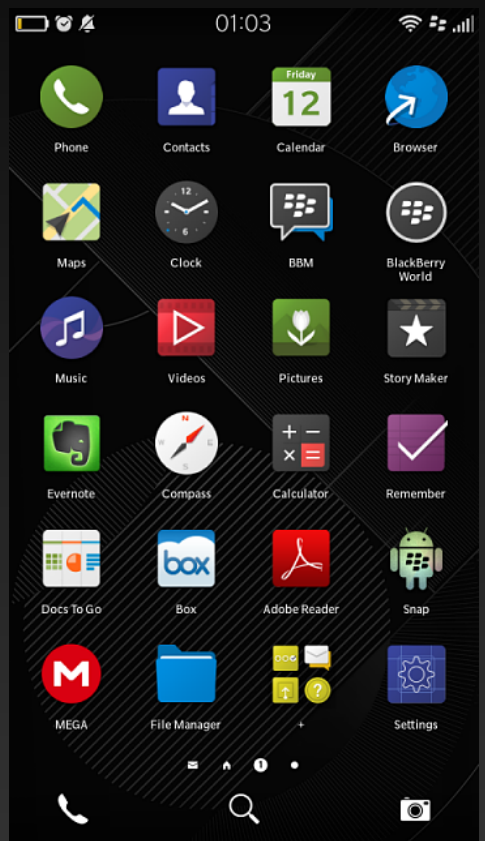
```
mponce~ $ pwd
/Users/mponce
mponce~ $ whoami
mponce
mponce~ $ uname -a
Darwin inside-38-7.wireless.utoronto.ca 14.5.0 Darwin Kernel Version 14.5.0: Tue
 Sep  1 21:23:09 PDT 2015; root:xnu-2782.50.1~1/RESEARCH_X86_64 x86_64
mponce~ $
```

```
C:\Temp> dir
 Volume in drive C is C
 Volume Serial Number is 74F5-B93C

 Directory of C:\Temp

2009-08-25  11:59    <DIR>          .
2009-08-25  11:59    <DIR>          ..
2007-03-01  11:37         2,321,600 AdobeUpdater12345.exe
2009-04-03  10:01            27,988 dd_depcheckdotnetfx30.txt
2009-04-03  10:01               764 dd_dotnetfx3error.txt
2009-04-03  10:01            32,572 dd_dotnetfx3install.txt
2009-06-09  13:46            35,145 GenProfile.log
2009-08-05  12:11               155 KB969856.log
2009-04-20  08:37               402 MSI29e0b.LOG
2009-04-09  16:34            38,895 offcln11.log
2009-04-03  16:02    <DIR>          OfficePatches
2009-07-14  14:30    <DIR>          OHotfix
2009-08-25  10:52            16,384 Perflib_Perfdata_c30.dat
2009-04-03  10:01             1,744 uxeventlog.txt
2009-08-25  11:42        50,245,632 WFV2F.tmp
2009-04-20  10:07             1,397 {AC76BA86-7AD7-1033-7B44-A81200000003}.ini
2009-04-20  10:13               617 {AC76BA86-7AD7-1033-7B44-A81300000003}.ini
              13 File(s)     52,723,295 bytes
               4 Dir(s)  83,570,208,768 bytes free
```

Very well-known example, nowadays...



The Truth about interfaces

- Nobody. Nobody. Nobody, uses a Graphical User Interface (GUI) for High Performance Computing (HPC). Nobody.
- Why? Because HPC is Unix/Linux based, without a GUI.
- Why?
  - Because the earliest mainframe computers were Unix based, and it's always been that way.
  - You can't have hundreds of people logged into a node, and run GUIs for all of them (but you can run a command line interface).
  - GUIs are slow over networks.
- Who cares? Well, if you're going to do real HPC then you're going to need to interface with these computers, and that means learning how to use the command line.
- This is not to suggest that Linux machines don't have GUIs. They do. It's just the HPC machines that don't.

GUIs versus the command line

- Graphical User Interfaces (GUIs) have many strengths.
  - Very good at operating an existing system.
  - Very good at using existing functionality, existing controls.
  - Programs tend to have lots of functionality built into them, but can only do what they've been programmed to do.
  - Can't save a series of commands to replicate functionality.
  - Easy to learn. Hard to use for big tasks.
- The Command Line Interface (CLI) has a different approach.
  - A blank canvas; you get to program what you want to do.
  - Good at creating new things.
  - Commands that do already exist are very good at doing *one* thing.
  - Commands that you create can be saved and re-used.
  - Hard to learn. Easy to use for big tasks.

"The" shell

Open a Terminal:

- Windows: start up MobaXterm.
- Mac: Applications/Utilities/Terminal (drag this to the dock).
- Linux: xterm, eterm, konsole, ...
- The terminal launches a shell. The shell is what you are actually interacting with when you type commands.
- The shell provides access to files, the network, and other programs.
  - You type in commands.
  - The shell interprets them.
  - Performs actions on its own, or launches other programs.
- The most commonly used shell in Linux is bash.
- There are others; mostly the same but some syntax is different.
- Those of you using MobaXterm: go to Settings > Configuration and change your "persistent HOME directory" to a permanent location.

The command line prompt

Now that we've got a terminal open, what do we see? We see the command line prompt!

On MobaXterm, the prompt looks something like this:

```
[USERname.mycomp]
```

Where 'USERname' is your username, and 'mycomp' is the name of your computer. On a Mac the prompt might look like this:

```
mycomp:~ USERname$
```

On a Linux machine, the prompt might look like this:

```
[USERname@mycomp ~]$
```

All of these are customizable, which we won't be covering today. It doesn't matter what it looks like, so long as you're comfortable with the prompt.

Our first shell command

We will be using the 'bash' shell for this class. It is the most commonly used on Linux systems, is widely available, and is the default on SciNet.

```
[USERname.mycomp]
[USERname.mycomp] hello="world"
[USERname.mycomp] echo Hello, world
Hello, world
[USERname.mycomp] echo Hello, $hello
Hello, world
```

Don't forget to hit 'Enter' at the end of each line.

The '=' sign tells the shell to create a variable called 'hello' and assign it the value "world". The value of the variable is accessed using the $.

The 'echo' command prints out whatever the shell gives it.

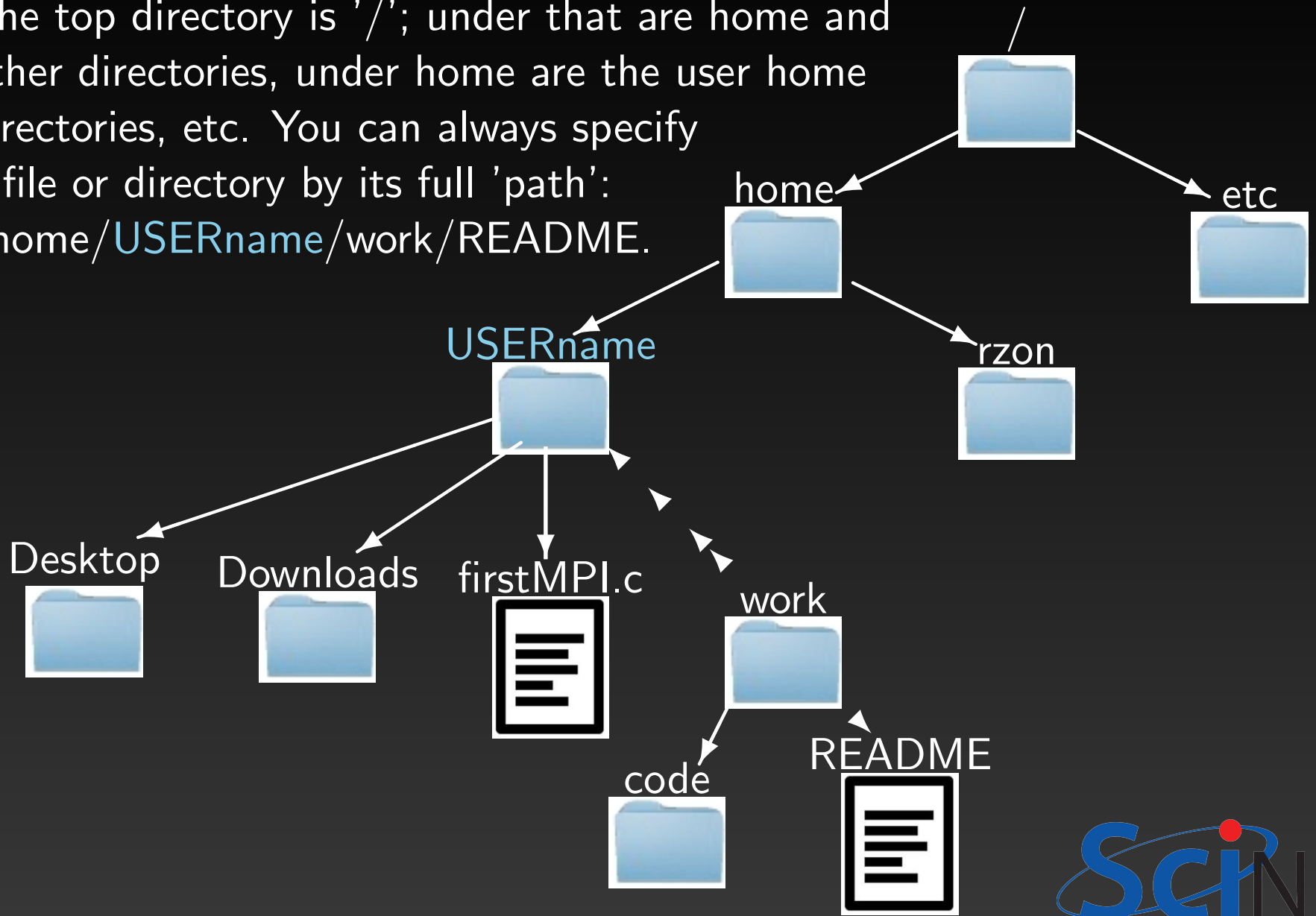
If you get an error message, it's likely you're running a different shell (csh, tcsh, zsh). Type 'bash' to start a bash shell, and try again.

Basics: home sweet home

- When you launch a shell, you start in your home directory, this is the top directory of all of your stuff.
- The home directory is /home/mobaxterm for MobaXterm, /Users/username on Macs, /home/username on Unix/Linux systems, /home/g/group/username on SciNet.
- The home directory is universally represented by the ~ symbol.
- Directories are sometimes called folders because of how they are represented in GUIs. We will call them directories.
- On Unix systems directories are listings of files, including other directories.
- If you are using MobaXterm your home directory will be put in your "persistent HOME directory" location, as set in Settings > Configuration.

A typical Linux directory tree

The top directory is '/'; under that are home and other directories, under home are the user home directories, etc. You can always specify a file or directory by its full 'path': /home/USERname/work/README.



Basics: the file system

I will be assuming I am on a MobaXterm terminal. Your output will likely differ somewhat if you are on a different system.

```
[USERname.mycomp]
[USERname.mycomp] pwd
/home/mobaxterm
[USERname.mycomp] ls
Desktop LauncherFolder MyDocuments
[USERname.mycomp] ls /home
USERname mobaxterm
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`arg`	mandatory argument
`[arg]`	optional argument

- 'pwd' stands for 'present working directory'. It will print the directory you are currently in. As mentioned on the last slide, you begin in your home directory.
- 'ls' stands for 'list'. If no argument is given it lists the contents of the current directory, otherwise it lists the contents of the argument. Some implementations of ls include colour.

Downloading today's data

```
[USERname.mycomp] pwd
/home/mobaxterm
```

```
[USERname.mycomp] ls
Desktop LauncherFolder MyDocuments
```

```
[USERname.mycomp] wget support.scinet.utoronto.ca/~ejspence/commandline.tar.gz
--2013-10-02 10:20:44--
http://support.scinet.utoronto.ca/~ejspence/commandline.tar.gz
Resolving support.scinet.utoronto.ca...  142.150.198.33
Connecting to support.scinet.utoronto.ca|142.150.198.33|:80...  connected.
HTTP request sent, awaiting response...  200 OK
Length: 15927 (16K) [application/x-gzip]
Saving to: ‘commandline.tar.gz’

100%[=======================================>] 15,927 --.-K/s in 0s

2013-10-02 10:20:44 (94.8 MB/s) - ‘commandline.tar.gz’ saved [15927/15927]
```

```
[USERname.mycomp]
```

If you are using a MAC, instead try:

```
curl -O https://support.scinet.utoronto.ca/~ejspence/commandline.tar.gz
```

Downloading today's data, continued

```
[USERname.mycomp] ls
Desktop LauncherFolder MyDocuments commandline.tar.gz
[USERname.mycomp] tar -z -x -f commandline.tar.gz
[USERname.mycomp]
[USERname.mycomp] ls -F
Desktop LauncherFolder MyDocuments commandline.tar.gz data/
```

What happened?

- `wget / curl` downloads the (tarred) data file. A 'tar' file (also called a 'tarball') is a file in which has been bundled a number of other files, for easy of moving around.
- `tar` handles a tar file,
  - `-z` means gunzip it.
  - `-x` means extract the contents of the file.
  - `-f` specifies which file you are applying this command to.

The data is now in the 'data' directory.

Creating directories

```
[USERname.mycomp] pwd
/home/mobaxterm
[USERname.mycomp] ls
Desktop LauncherFolder MyDocuments
commandline.tar.gz data
[USERname.mycomp] mkdir firstdir
[USERname.mycomp] ls -F
Desktop LauncherFolder MyDocuments command-
line.tar.gz data/ firstdir/
[USERname.mycomp] mkdir /home/mobaxterm/2ndir
[USERname.mycomp] ls -F
2ndir/ Desktop LauncherFolder MyDocuments
commandline.tar.gz data/ firstdir/
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`arg`	mandatory argument
`[arg]`	optional argument

- 'mkdir' stands for 'make directory'. It creates a new directory, putting it in the current directory unless a different path is specified.
- 'ls -F' lists the directory, as before, but labels directories with a '/', and links with a '@'.

Moving between directories

```
[USERname.mycomp] ls
2ndir Desktop LauncherFolder MyDocuments
commandline.tar.gz data firstdir
[USERname.mycomp] mkdir firstdir/temp
[USERname.mycomp] cd firstdir
[USERname.mycomp] pwd
/home/mobaxterm/firstdir
[USERname.mycomp] ls
temp
[USERname.mycomp] cd temp
[USERname.mycomp] pwd
/home/mobaxterm/firstdir/temp
[USERname.mycomp] cd ..
[USERname.mycomp] pwd
/home/mobaxterm/firstdir
[USERname.mycomp] cd ~
[USERname.mycomp] pwd
/home/mobaxterm
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`arg`	mandatory argument
`[arg]`	optional argument

'cd' stands for 'change directory'. It moves you to the directory you specify. With no argument it moves you to the home directory.

Tips for getting around

Some common commands for moving around your directories:

- The directory above is represented by the '..' symbol; the current directory is represented by the '.' symbol:
  - 'cd ..' goes up a directory.
  - 'cd ../..' goes up two directories.
  - 'cd ../otherdir' goes up one directory and then down into 'otherdir'.
  - 'cd firstdir/seconddir/../..' goes nowhere.
  - 'cd ./././.' also goes nowhere.
- You can use absolute paths: 'cd /home/mobaxterm/firstdir/temp'.
- ~ is the symbol for your home directory, on whatever system you are using. 'cd ~/work' goes to my ~/work directory (/home/mobaxterm/work).
- 'cd' without any arguments goes to your home directory (~), from no matter where you are.
- 'cd -' goes back to the directory you were in previously.

Tips for using the command line

Some more helpful tips for using the command line:

- Use the 'tab' key, it will 'auto-complete' the available options based on what you've already typed,
  - start typing your command, and then hit 'tab'
  - the shell will fill in the rest, if there is only one option.
  - if nothing happens, there is either no option or more than one option.
  - hit the tab key twice, this will list all available options
  - continue typing to reduce the number of options, then hit tab again to fill in the rest.
- Use 'Ctrl-a' to go to the beginning of the command line, 'Ctrl-e' to go to the end of the line.
- Use the up arrow. This scrolls through the shell's 'history'.

History

```
[USERname.mycomp] HISTTIMEFORMAT="[%F %T] "
[USERname.mycomp] history
.
.
14 [2014-06-05 11:23:42] mkdir firstdir/temp
15 [2014-06-05 11:23:47] cd firstdir
16 [2014-06-05 11:23:49] pwd
17 [2014-06-05 11:23:50] ls
18 [2014-06-05 11:23:53] cd temp
19 [2014-06-05 11:23:55] pwd
20 [2014-06-05 11:23:58] cd ..
21 [2014-06-05 11:23:59] pwd
22 [2014-06-05 11:24:03] cd
23 [2014-06-05 11:24:05] pwd
24 [2014-06-05 11:24:11] history
[USERname.mycomp]
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`arg`	mandatory argument
`[arg]`	optional argument

- The history command prints the commands that you've typed at the command line. "history 10" prints the last 10 commands.
- Use the up arrow to access the entries.

Our commands so far

There are a couple of things to observe about the commands we've seen so far:

- The commands are designed to be fast and easy to use.
- The commands do, essentially, only one specific thing.
- The commands are pretty cryptic. Either you know them or you don't.
- Commands can take options. These are usually indicated with a '-something' flag (such as 'ls -F').

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`arg`	mandatory argument
`[arg]`	optional argument

As you may have hoped, the purpose of this class is to teach you enough commands that you will be able to survive the Unix command line.

Man pages

Know a command but aren't sure how to use the options? Use the man (manual) page!

- Most programs have a man page describing its use and all available options.
- These pages are good for finding out more about a command you already use, but are less good for learning new commands.
- Many programs have gazillions of options.
- No human being who has ever lived has known all the options for 'ls'.
- Over time you will find a few options that you find useful for your favourite commands.
- Unfortunately, MobaXterm dumps all man pages together, so you need to scroll down to find the entry you want. Or "source" the file new_man.

Man pages: help!

Use the man (manual) page for a list of all flags for a command.

```
[USERname.mycomp] man ls
NAME
  ls - list directory contents
SYNOPSIS
  ls [OPTION]...  [FILE]...
DESCRIPTION
  List information about the FILEs (the
  current directory by default).  Sort
  entries alphabetically if none of -
  cftuvSUX nor --sort.

Mandatory arguments to long options are
mandatory for short options too.
-a, --all
  do not ignore entries starting with .
-A, --almost-all
  do not list implied .  and ..
...
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`arg`	mandatory argument
`[arg]`	optional argument

Not sure how to use the command? Not sure what options there are? Check the man page!

Wildcards

Wildcards (*) capture all possible combinations that fit a given description.

```
[USERname.mycomp] pwd
/home/mobaxterm
```

```
[USERname.mycomp] cd data
```

```
[USERname.mycomp] ls
alexander Bert Frank_Richard gerdal jamesm
Lawrence THOMAS
```

```
[USERname.mycomp] ls -d *er*
alexander Bert gerdal
```

```
[USERname.mycomp] ls -d *e
Lawrence
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`arg`	mandatory argument
`[arg]`	optional argument

The shell expands the wildcard into a list of all possible matches, and passes the list to the command.

Manipulating files: copying

```
[USERname.mycomp] ls
Bert           Lawrence alexander jamesm
Frank_Richard THOMAS    gerdal
```

```
[USERname.mycomp] cd gerdal
```

```
[USERname.mycomp] ls
.
Data0413 Data0468 Data0528 Data0558
```

```
[USERname.mycomp] ls *27*
Data0227 Data0279
```

```
[USERname.mycomp] cp Data0227 Data0227-new
```

```
[USERname.mycomp] ls *27*
Data0227 Data0227-new Data0279
```

```
[USERname.mycomp] cp Data0227 ..
```

```
[USERname.mycomp] ls ..
Bert       Frank_Richard THOMAS     gerdal
Data0227   Lawrence       alexander jamesm
```

'cp' stands for 'copy'; it copies a file.

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`arg`	mandatory argument
`[arg]`	optional argument

Wildcards can appear anywhere in the variable you are searching for. They don't need to come at the end.

Manipulating files: moving

```
[USERname.mycomp] pwd
/home/mobaxterm/data/gerdal
[USERname.mycomp] ls *27*
Data0227 Data0227-new Data0279
[USERname.mycomp] mv Data0227-new new.txt
[USERname.mycomp] ls *27*
Data0227 Data0279
[USERname.mycomp] ls *txt
new.txt
[USERname.mycomp] mv new.txt ../Data0227
[USERname.mycomp] ls *txt
ls: *txt: No such file or directory
[USERname.mycomp] cd ..
[USERname.mycomp] ls *27*
Data0227
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`arg`	mandatory argument
`[arg]`	optional argument

- 'mv' stands for 'move'; it moves a file and/or renames it.
- mv can overwrite a file, so be careful when moving things!

Manipulating files: deleting

```
[USERname.mycomp] pwd
/home/mobaxterm/data
[USERname.mycomp] cd ..
[USERname.mycomp] ls
Bert       Frank_Richard THOMAS    gerdal
Data0227   Lawrence      alexander jamesm
[USERname.mycomp] ls *27*
Data0227
[USERname.mycomp] rm Data0227
[USERname.mycomp] ls *227*
ls: *227*: No such file or directory
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`arg`	mandatory argument
`[arg]`	optional argument

- 'rm' stands for 'remove'; it deletes a file. It does not delete directories, by default.
- rm does not 'move the file to the Trash'. It deletes it; it's gone; it's not recoverable. Be sure before using rm.

Copying directories

```
[USERname.mycomp] pwd
/home/mobaxterm/data
[USERname.mycomp] mkdir temp
[USERname.mycomp] cp gerdal/Data0227 temp
[USERname.mycomp] ls temp
Data0227
[USERname.mycomp] cp temp temp2
cp: omitting directory 'temp'
[USERname.mycomp] cp -r temp temp2
[USERname.mycomp] ls
Bert          Lawrence  alexander jamesm
Frank_Richard THOMAS    gerdal    temp
temp2
[USERname.mycomp] ls temp2
Data0227
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`rmdir dir`	delete a directory
`arg`	mandatory argument
`[arg]`	optional argument

'cp' will only copy files by default. To copy directories, including everything within them, use 'cp -r'.

Deleting directories

```
[USERname.mycomp] pwd
/home/mobaxterm/data
[USERname.mycomp] ls temp
Data0227
[USERname.mycomp] rm temp
rm: temp: is a directory
[USERname.mycomp] rmdir temp
rmdir: 'temp': Directory not empty
[USERname.mycomp] rm temp/*
[USERname.mycomp] ls temp
[USERname.mycomp] rmdir temp
[USERname.mycomp] rm temp2/*
[USERname.mycomp] rmdir temp2
[USERname.mycomp]
```

'rmdir' deletes a directory.

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`rmdir dir`	delete a directory
`arg`	mandatory argument
`[arg]`	optional argument

Uncharacteristically for Linux, rmdir protects you. You can't delete a directory with files in it, you must delete the files first.

Looking inside files

```
[USERname.mycomp] cd alexander
[USERname.mycomp] pwd
/home/mobaxterm/data/alexander
[USERname.mycomp] more data_560.DATA
#
Reported: Sat May 7 10:50:03 2011
Subject: georgeSpice437
Year/month of birth: 1997/12
Sex: M
CI type: 20
Volume: 3
Range: 5
Discrimination:
[USERname.mycomp]
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`rmdir dir`	delete a directory
`more file`	scroll through file

`arg`	mandatory argument
`[arg]`	optional argument

'more' lists the contents of the file.

Looking inside files, continued

```
[USERname.mycomp] cat data_560.DATA
#
Reported: Sat May 7 10:50:03 2011
Subject: georgeSpice437
.
.
```

```
[USERname.mycomp] less data_560.DATA
#
Reported: Sat May 7 10:50:03 2011
Subject: georgeSpice437
.
.
```

'more', 'cat', and 'less' all output the contents of the file, but in different ways. Can you tell the differences? Type 'q' to get out of 'more' or 'less'.

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`rmdir dir`	delete a directory
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents

`arg`	mandatory argument
`[arg]`	optional argument

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

cat'ing files together

```
[USERname.mycomp] ls *DATA
.
.
data_379.DATA data_434.DATA data_530.DATA
data_420.DATA data_502.DATA data_560.DATA
data_297.DATA data_357.DATA data_421.DATA
[USERname.mycomp] cat *DATA > all-DATA
[USERname.mycomp] ls *DATA
all-DATA data_297.DATA data_357.DATA
.
.
data_346.DATA data_415.DATA data_498.DATA
data_550.DATA data_292.DATA data_347.DATA
data_420.DATA data_502.DATA data_560.DATA
[USERname.mycomp]
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`rmdir dir`	delete a directory
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents
`cmd > file`	redirect output to file

`arg`	mandatory argument
`[arg]`	optional argument

- 'cat' dumps the input (whatever it is) to the screen.
- '>' redirects the input to a file, instead of the screen.

cat'ing files together, continued

```
[USERname.mycomp] less all-DATA
#
Reported: Wed Aug 17 13:56:38 2011
Subject: madonnaStarr178
Year/month of birth: 1995/02
Sex: N
CI type: 8
Volume: 7
Range: 3
Discrimination: 5
#
Reported: Thu May 19 09:08:14 2011
Subject: paulSpice199
Year/month of birth: 1994/01
Sex: M
CI type: 24
Volume: 4
Range: 9
.
.
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`rmdir dir`	delete a directory
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents
`cmd > file`	redirect output to file

`arg`	mandatory argument
`[arg]`	optional argument

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Redirection fun

- cmd > file takes the output that would have gone to the screeen, creates a new file called file, and redirects (dumps) the output to the file. If the file already exists the previous content of the file is overwritten.
- cmd >> file takes the output that would have gone to the screen, and *appends* it to file. If the file doesn't already exist then it is created.
- cmd < file takes file and uses it as input to cmd.

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`rmdir dir`	delete a directory
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents
`cmd > file`	redirect output to file
`cmd >> file`	append output to file
`cmd < file`	use file as input to cmd

`arg`	mandatory argument
`[arg]`	optional argument

More redirection fun

```
[USERname.mycomp] cat < all-DATA
#
Reported: Wed Aug 17 13:56:38 2011
Subject: madonnaStarr178
Year/month of birth: 1995/02
Sex: N
CI type: 8
Volume: 7
Range: 3
Discrimination: 5
#
Reported: Thu May 19 09:08:14 2011
Subject: paulSpice199
Year/month of birth: 1994/01
Sex: M
CI type: 24
Volume: 4
Range: 9
.
.
```

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`rmdir dir`	delete a directory
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents
`cmd > file`	redirect output to file
`cmd >> file`	append output to file
`cmd < file`	use file as input to cmd

`arg`	mandatory argument
`[arg]`	optional argument

Head/Tail

```
[USERname.mycomp] pwd
/home/mobaxterm/data/alexander
[USERname.mycomp] head -4 all-DATA
#
Reported: Wed Aug 17 13:56:38 2011
Subject: madonnaStarr178
Year/month of birth: 1995/02
[USERname.mycomp] echo "nice" >> all-DATA
[USERname.mycomp] tail -5 all-DATA
CI type: 20
Volume: 3
Range: 5
Discrimination:
nice
[USERname.mycomp]
```

'head'/'tail' prints the first/last 10 lines of the input.

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`rmdir dir`	delete a directory
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents
`cmd > file`	redirect output to file
`cmd >> file`	append output to file
`cmd < file`	use file as input to cmd
`head file`	print first 10 lines of file
`tail file`	print last 10 lines of file

Word count

```
[USERname.mycomp] pwd
/home/mobaxterm/data/alexander
[USERname.mycomp] wc all-DATA
441 1173 7184 all-DATA
[USERname.mycomp] wc -l all-DATA
441 all-DATA
[USERname.mycomp] wc -w all-DATA
1173 all-DATA
[USERname.mycomp] wc -c all-DATA
7184 all-DATA
[USERname.mycomp] wc -w *DATA
.
.
24 data_550.DATA
23 data_560.DATA
2346 total
```

'wc' stands for 'word count'. It counts the number of words/lines/characters in the input.

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`rmdir dir`	delete a directory
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents
`cmd > file`	redirect output to file
`cmd >> file`	append output to file
`cmd < file`	use file as input to cmd
`head file`	print first 10 lines of file
`tail file`	print last 10 lines of file
`wc file`	word count data of file

File-Type & Location

The `file` commands will try to determine the type of a given file.

```
[USERname.mycomp] cd ~/data
```

File-Type & Location

The `file` commands will try to determine the type of a given file.

```
[USERname.mycomp] cd ~/data
[USERname.mycomp] file Lawrence/
```

File-Type & Location

The `file` commands will try to determine the type of a given file.

```
[USERname.mycomp] cd ~/data
[USERname.mycomp] file Lawrence/
Lawrence/:  directory
```

File-Type & Location

The `file` commands will try to determine the type of a given file.

```
[USERname.mycomp] cd ~/data
[USERname.mycomp] file Lawrence/
Lawrence/:  directory
[USERname.mycomp] file Lawrence/Data0554
```

File-Type & Location

The `file` commands will try to determine the type of a given file.

```
[USERname.mycomp] cd ~/data
[USERname.mycomp] file Lawrence/
Lawrence/:  directory
[USERname.mycomp] file Lawrence/Data0554
Lawrence/Data0554:  ASCII text
```

File-Type & Location

The `file` commands will try to determine the type of a given file.

```
[USERname.mycomp] cd ~/data
[USERname.mycomp] file Lawrence/
Lawrence/:  directory
[USERname.mycomp] file Lawrence/Data0554
Lawrence/Data0554:  ASCII text
```

The command `which` will look where and which program is being executed when ran in the command line:

```
[USERname.mycomp] which wget
/bin/wget
```

File-Type & Location

The `file` commands will try to determine the type of a given file.

```
[USERname.mycomp] cd ~/data
[USERname.mycomp] file Lawrence/
Lawrence/:  directory
[USERname.mycomp] file Lawrence/Data0554
Lawrence/Data0554:  ASCII text
```

The command `which` will look where and which program is being executed when ran in the command line:

```
[USERname.mycomp] which wget
/bin/wget
[USERname.mycomp] file /bin/wget
```

File-Type & Location

The `file` commands will try to determine the type of a given file.

```
[USERname.mycomp] cd ~/data
[USERname.mycomp] file Lawrence/
Lawrence/:  directory
[USERname.mycomp] file Lawrence/Data0554
Lawrence/Data0554:  ASCII text
```

The command `which` will look where and which program is being executed when ran in the command line:

```
[USERname.mycomp] which wget
/bin/wget
[USERname.mycomp] file /bin/wget
/bin/wget:  PE32 executable (console) Intel 80386, for MS Windows
```

File-Type & Location

The `file` commands will try to determine the type of a given file.

```
[USERname.mycomp] cd ~/data
[USERname.mycomp] file Lawrence/
Lawrence/:  directory
[USERname.mycomp] file Lawrence/Data0554
Lawrence/Data0554:  ASCII text
```

The command `which` will look where and which program is being executed when ran in the command line:

```
[USERname.mycomp] which wget
/bin/wget
[USERname.mycomp] file /bin/wget
/bin/wget:  PE32 executable (console) Intel 80386, for MS Windows
```

Similar to `which` there is `whereis` to find other files associated with the command (try `man which`, `man whereis`).

Comparing Files

```
[USERname.mycomp] cd ~/data
```

Comparing Files

```
[USERname.mycomp] cd ~/data
[USERname.mycomp] diff alexander/data_216.DATA alexander/data_242.DATA
```

Comparing Files

```
[USERname.mycomp] cd ~/data
[USERname.mycomp] diff alexander/data_216.DATA alexander/data_242.DATA
2,9c2,9
< Reported:  Wed Aug 17 13:56:38 2011
< Subject:  madonnaStarr178
< Year/month of birth:  1995/02
< Sex:  N
< CI type:  8
< Volume:  7
< Range:  3
< Discrimination:  5
---
> Reported:  Thu May 19 09:08:14 2011
> Subject:  paulSpice199
> Year/month of birth:  1994/01
> Sex:  M
> CI type:  24
> Volume:  4
> Range:  9
> Discrimination:  8
```

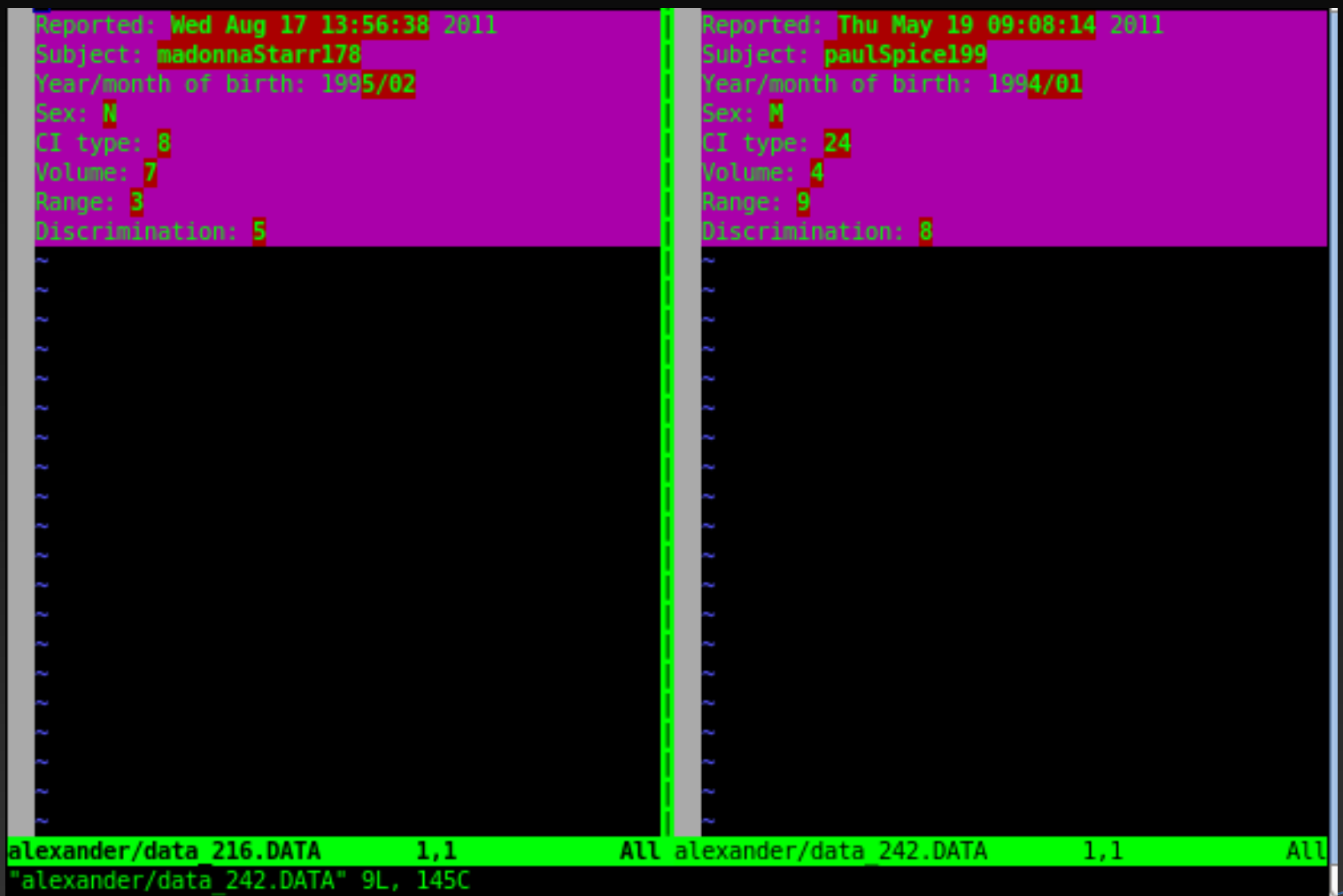
vimdiff: an interactive diff with vim-interface

vimdiff: an interactive diff with vim-interface

```
[USERname.mycomp] vimdiff alexander/data_216.DATA alexander/data_242.DATA
```

vimdiff: an interactive diff with vim-interface

```
[USERname.mycomp] vimdiff alexander/data_216.DATA alexander/data_242.DATA
```



For EXIT, type "**:q**" and Enter **twice**

Comparing File's content

- `diff`. `vimdiff` work fine and nicely for text files

Comparing File's content

- `diff`. `vimdiff` work fine and nicely for text files
- there is even a 3-files comparison tool named, `diff3`

Comparing File's content

- `diff`. `vimdiff` work fine and nicely for text files
- there is even a 3-files comparison tool named, `diff3`
- `vimdiff` can handle three or even more files too!!!

Comparing File's content

- `diff`. `vimdiff` work fine and nicely for text files
- there is even a 3-files comparison tool named, `diff3`
- `vimdiff` can handle three or even more files too!!!
- but they are not useful for dealing with binary files (data or executables)

Comparing File's content

- `diff`. `vimdiff` work fine and nicely for text files
- there is even a 3-files comparison tool named, `diff3`
- `vimdiff` can handle three or even more files too!!!
- but they are not useful for dealing with binary files (data or executables)
- for binary files, use `cmp` instead
- `cmp` compare two files byte by byte...

find

- Wildcards are very powerful.
- from the data directory, type 'ls */*00*'.

```
[USERname.mycomp] pwd
/home/mobaxterm/data/alexander
[USERname.mycomp] cd ..
[USERname.mycomp] ls */*00*
.
.
Bert/audioresult-00330.txt Bert/audioresult-00460.txt Frank_Richard/data_500
Bert/audioresult-00332.txt Bert/audioresult-00466.txt Lawrence/Data0300
Bert/audioresult-00350.txt Bert/audioresult-00470.txt Lawrence/Data0400
```

- This finds files which contain '00' in the name, in any subdirectory one level below this one.
- Similarly for 'echo */*00*'.
- But it can only match the specified levels of directories.
- 'find' is a tool which lets you find files anywhere below a given directory, based on arbitrary criteria.

find, continued

```
[USERname.mycomp] pwd
/home/mobaxterm/data
[USERname.mycomp] find . -print
.
.
./jamesm/data_553.txt
./jamesm/NOTES
./jamesm/data_374.txt
./jamesm/data_280.txt
./jamesm/data_375.txt
./jamesm/data_476.txt
./jamesm/data_264.txt
[USERname.mycomp]
```

'find . -print' tells find to look for files starting in the directory '.', and to print the results.

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`mkdir dir`	create a directory
`rmdir dir`	delete a directory
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents
`cmd > file`	redirect output to file
`cmd >> file`	append output to file
`cmd < file`	use file as input to cmd
`head file`	print first 10 lines of file
`tail file`	print last 10 lines of file
`wc file`	word count data of file
`find dir`	find files

More find options, continued

```
[USERname.mycomp] find . -type f -name "*09*"
./gerdal/Data0409
./alexander/data_309.DATA
./jamesm/data_509.txt
[USERname.mycomp]
```

The '-name' argument specifies the characteristics of the name of the file to be found.

Our commands

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`mkdir dir`	create a directory
`rmdir dir`	delete a directory
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents
`cmd > file`	redirect output to file
`cmd >> file`	append output to file
`cmd < file`	use file as input to cmd
`head file`	print first 10 lines of file
`tail file`	print last 10 lines of file
`wc file`	word count data of file
`find dir`	find files

Pipelines of commands

- So far we've used the following technique to combine commands:

```
[USERname.mycomp] pwd
/home/mobaxterm/data
[USERname.mycomp] cd Lawrence
[USERname.mycomp] wc Data* > all-wcs
[USERname.mycomp] more all-wcs
```

  which creates a temporary file we don't care about; we just want to scroll through the wc results.

- This combination of actions, output of one command goes straight to another, is so common and useful that the shell has a special feature to do this:

```
[USERname.mycomp] wc Data* | more
```

- The 'pipe' allows you to chain together small commands.

The sort command

The sort command can take a number of important flags:

- `-n`: sort by number (not lexicographic; 101 $<$ 30 without `-n`).
- `-k [num]`: sort by the num'th column.
- `-r`: reverses order.

```
[USERname.mycomp] sort -n -k 3 all-wcs
...
9 24 150 Data0515
9 24 153 Data0214
468 1246 7644 total
```

```
[USERname.mycomp] wc Data* | sort -n -k 3
...
9 24 150 Data0515
9 24 153 Data0214
468 1246 7644 total
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`arg`	mandatory argument
`[arg]`	optional argument

Pop quiz!

Modify this code to print only the smallest, and then only the largest, data file.

```
[USERname.mycomp] wc Data* | sort -n -k 3
...
9 24 150 Data0515
9 24 153 Data0214
468 1246 7644 total
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`arg`	mandatory argument
`[arg]`	optional argument

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Pop quiz!

Modify this code to print only the smallest, and then only the largest, data file.

```
[USERname.mycomp] wc Data* | sort -n -k 3
...
9 24 150 Data0515
9 24 153 Data0214
468 1246 7644 total
```

```
[USERname.mycomp] wc Data* | sort -n -k 3
| head -n 1
9 24 144 Data0234
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`arg`	mandatory argument
`[arg]`	optional argument

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Pop quiz!

Modify this code to print only the smallest, and then only the largest, data file.

```
[USERname.mycomp] wc Data* | sort -n -k 3
...
9 24 150 Data0515
9 24 153 Data0214
468 1246 7644 total
```

```
[USERname.mycomp] wc Data* | sort -n -k 3
| head -n 1
9 24 144 Data0234
```

```
[USERname.mycomp] wc Data* | sort -n -k 3
-r | head -n 2
468 1246 7644 total
9 24 153 Data0214
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`arg`	mandatory argument
`[arg]`	optional argument

Pop quiz!

Modify this code to print only the smallest, and then only the largest, data file.

```
[USERname.mycomp] wc Data* | sort -n -k 3
...
9 24 150 Data0515
9 24 153 Data0214
468 1246 7644 total
```

```
[USERname.mycomp] wc Data* | sort -n -k 3
| head -n 1
9 24 144 Data0234
```

```
[USERname.mycomp] wc Data* | sort -n -k 3
-r | head -n 2
468 1246 7644 total
9 24 153 Data0214
```

```
[USERname.mycomp] wc Data* | sort -n -k 3
-r | head -n 2 | tail -n 1
9 24 153 Data0214
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`arg`	mandatory argument
`[arg]`	optional argument

Our first shell script

It's time to write our first shell script. To do this we need to use a text editor. Linux has many text editors, and they're all difficult to use.

In MobaXterm try Tools > MobaTextEditor, or you can use nano, emacs, vi, ... Create a file called 'biggest':

```
#!/bin/bash
wc * | sort -n -k 3 | tail -n 2 | head -n 1
```

And run it by typing

```
[USERname.mycomp] rm all-wcs
[USERname.mycomp] source biggest
9 24 153 Data0214
```

Permissions

Files are not generally executable by default, for security reasons. How do you tell if it's executable?

```
[USERname.mycomp] ls -l biggest
-rw-r--r-- 1 mponce scinet 0 Jan 20 14:56 biggest
```

Skip the first dash. The next three indicate the permissions (readable, writable, executable) for the owner of the file, then for members of the group, then everyone else.

To change the permissions using the 'chmod' command:

```
[USERname.mycomp] ls -l biggest
-rw-r--r-- 1 mponce scinet 0 Jan 20 14:56 biggest
[USERname.mycomp] chmod u+x biggest
[USERname.mycomp] ls -l biggest
-rwxr--r-- 1 mponce scinet 0 Jan 20 14:56 biggest
[USERname.mycomp] ./biggest
9 24 153 Data0214
```

Largest range - grep

Number of characters is probably unimportant for our analysis. What about the data with the largest range?

```
[USERname.mycomp] grep Range Data0352
Range: 2
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`arg`	mandatory argument
`[arg]`	optional argument

grep prints the lines from the input that contain the search argument.

Largest range - grep

Number of characters is probably unimportant for our analysis. What about the data with the largest range?

```
[USERname.mycomp] grep Range Data0352
Range: 2
---------------------------------------
[USERname.mycomp] grep Range *
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`arg`	mandatory argument
`[arg]`	optional argument

grep prints the lines from the input that contain the search argument.

Largest range - grep

Number of characters is probably unimportant for our analysis. What about the data with the largest range?

```
[USERname.mycomp] grep Range Data0352
Range: 2
```

```
[USERname.mycomp] grep Range *
...
Data0544:Range: 6
Data0554:Range: 2
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`arg`	mandatory argument
`[arg]`	optional argument

grep prints the lines from the input that contain the search argument.

Largest range - grep

Number of characters is probably unimportant for our analysis. What about the data with the largest range?

```
[USERname.mycomp] grep Range Data0352
Range: 2
```

```
[USERname.mycomp] grep Range *
...
Data0544:Range: 6
Data0554:Range: 2
```

```
[USERname.mycomp] grep -v Range Data0352
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`arg`	mandatory argument
`[arg]`	optional argument

grep prints the lines from the input that contain the search argument.

Largest range - grep

Number of characters is probably unimportant for our analysis. What about the data with the largest range?

```
[USERname.mycomp] grep Range Data0352
Range: 2
```

```
[USERname.mycomp] grep Range *
...
Data0544:Range: 6
Data0554:Range: 2
```

```
[USERname.mycomp] grep -v Range Data0352
#
Reported: Mon Jul 25 14:01:36 2011
Subject: babyMcCartney281
Year/month of birth: 1991/07
Sex: M
CI type: 14
Volume: 1
Discrimination: 4
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`arg`	mandatory argument
`[arg]`	optional argument

grep prints the lines from the input that contain the search argument.

grep -v prints the lines from the input that *don't* contain the search argument.

Pop quiz!

Create a new script, called 'biggestRange', which prints out the data file which has the biggest Range.

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`arg`	mandatory argument
`[arg]`	optional argument

Pop quiz!

Create a new script, called 'biggestRange', which prints out the data file which has the biggest Range.

```
[USERname.mycomp] cat biggestRange
#!/bin/bash

# a comment!  Use the # sign.

grep Ra * | sort -n -k 2 | tail -n 1
```

```
[USERname.mycomp]
```

```
[USERname.mycomp] chmod u+x biggestRange
```

```
[USERname.mycomp] ./biggestRange
Data0531:Range: 9
```

```
[USERname.mycomp]
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`arg`	mandatory argument
`[arg]`	optional argument

Cutting up the output

How do keep just part of the output?

Suppose we want just the file name?

```
[USERname.mycomp] grep Ra * | sort -n -k 2
| tail -1
Data0531:Range: 9
```

```
[USERname.mycomp] grep Ra * | sort -n -k 2
| tail -1 | cut -c -8
Data0531
```

Suppose we just want the range?

```
[USERname.mycomp] grep Ra * | sort -n -k 2
| tail -1 | cut -c 10-
Range: 9
```

Suppose we just want something else?

```
[USERname.mycomp] grep Ra * | sort -n -k 2
| tail -1 | cut -c 2-5
ata0
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`cut flags output`	cut part of output

`arg`	mandatory argument
`[arg]`	optional argument

- "-c" tells cut to cut characters.
- "-8" means keep up-to-and-including character eight.
- "10-" means keep 10 and higher.

Suppose I need to save information

If I need to save something, I use variables.

```
[USERname.mycomp] grep Ra * | sort -n -k 2
| tail -1 | cut -c -8
Data0531
```

```
[USERname.mycomp] i=‘grep Ra * | sort -n
-k 2 | tail -1 | cut -c -8‘
```

```
[USERname.mycomp]
```

```
[USERname.mycomp] echo i
i
```

```
[USERname.mycomp] echo $i
Data0531
```

NOTE: The ‘ symbol used here is the backward quote in the upper-left corner of your keyboard!! Using this backward quote will execute the command.

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`cut flags output`	cut part of output
`arg`	mandatory argument
`[arg]`	optional argument

What happens if you use the single forward quote?

Arguments for bash scripts

- We'd like to use our script on each directory, but not have a copy in each one.
- Move biggestRange down one directory, and change it so that it works on any directory's files:

```
[USERname.mycomp] pwd
/home/mobaxterm/data/Lawrence
[USERname.mycomp] mv biggestRange ..
[USERname.mycomp] cd ..
[USERname.mycomp] cat biggestRange
#!/bin/bash
grep Range ${1}/* | sort -n -k 2 | tail -1
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`cut flags output`	cut part of output
`arg`	mandatory argument
`[arg]`	optional argument

- When a command is run in the shell, its name is put in ${0}.
- All other arguments are put in ${1}, ${2}...

Arguments for bash scripts

```
[USERname.mycomp] pwd
/home/mobaxterm/data
```

```
[USERname.mycomp] ls
Bert Frank_Richard Lawrence THOMAS
alexander biggestRange gerdal jamesm
```

```
[USERname.mycomp] ./biggestRange Bert
Bert/audioresult-00384.txt:Range: 10
```

```
[USERname.mycomp] ./biggestRange THOMAS
THOMAS/0336:Range: 10
```

```
[USERname.mycomp] ./biggestRange james
grep: james/*: No such file or directory
```

```
[USERname.mycomp] ./biggestRange jamesm
jamesm/data_517.txt:Range: 10
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`cut flags output`	cut part of output
`arg`	mandatory argument
`[arg]`	optional argument

Loops for bash scripts

Bash has loops, just like any language:

```
[USERname.mycomp]
[USERname.mycomp] cat loop_biggestRange
#!/bin/bash
for dir in alexander Bert Frank_Richard
do
echo "The biggest range in " ${dir} " is:"
./biggestRange ${dir}
done
[USERname.mycomp]
[USERname.mycomp] ./loop_biggestRange
The biggest range in alexander is:
alexander/data_462.DATA:Range: 10
The biggest range in Bert is:
Bert/audioresult-00384.txt:Range 10
The biggest range in Frank_Richard is:
Frank_Richard/data_538:Range: 10
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`cut flags output`	cut part of output
`for..do..done`	for loop in bash
`arg`	mandatory argument
`[arg]`	optional argument

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

If statements for bash scripts

Bash has if statements, just like any other language:

```
[USERname.mycomp] pwd
/home/mobaxterm/data
```

```
[USERname.mycomp] cat if_test.sh
#!/bin/bash
cd ${1}
for filename in *
do

if [ $filename == "NOTES" ]
then
echo "I'm a NOTES file."
fi

done
```

```
[USERname.mycomp] ./if_test.sh jamesm
I'm a NOTES file.
```

```
[USERname.mycomp] ./if_test.sh alexander
```

```
[USERname.mycomp]
```

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`cut flags output`	cut part of output
`for..do..done`	for loop in bash
`if..then..fi`	if statement in bash
`arg`	mandatory argument
`[arg]`	optional argument

An easier way to avoid the NOTES file would be

```
cd ${1}
ls * | grep -v NOTES
```

Logout

```
[USERname.mycomp] logout
```

What to do when you're finished? Use the 'logout' command to exit the terminal session cleanly (you don't need to do this now). Ctrl-d also works.

More commands

`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`cut flags output`	cut part of output
`for..do..done`	for loop in bash
`if..then..fi`	if statement in bash
`logout`	close the terminal session

`arg`	mandatory argument
`[arg]`	optional argument

Enough to get started

- These commands are enough to get started with using the command line.
- As you have seen, Linux commands are simple, and are designed to do one specific task very well.
- By combining these commands together we will be able to do more interesting, complex tasks.
- If there is functionality that you think it ought to exist, it probably does. Ask someone what the command is, or google it.

Shell-command cheat sheet

`echo arg`	echo the argument
`pwd`	present working directory
`ls [dir]`	list the directory contents
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`mkdir dir`	create a directory
`rmdir dir`	delete a directory
`file file`	type of file
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents
`cmd > file`	redirect output to file
`cmd >> file`	append output to file
`cmd < file`	use file as input to cmd
`head file`	print first 10 lines of file
`tail file`	print last 10 lines of file
`wc file`	word count data of file

`file filename`	info about the file's type
`locate filename`	locate find files by name
`whereis filename`	find files assoc.w/commands
`which filename`	locate commands (programs)
`diff file1 file2`	compare two (text) files
`vimdiff file1 file2`	vi-interface for `diff`
`cmp file1 file2`	compare two (binary) files
`wget (curl) url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`chmod p file`	change file permissions
`grep arg file`	search for arg in file
`cut flags output`	cut part of output
`for..do..done`	for loop in bash
`if..then..fi`	if statement in bash
`logout`	close the terminal session
`arg`	mandatory argument
`[arg]`	optional argument