

# INTRODUCTION TO BASIC LINUX

# What Will We Learn?

- Common commands in Linux
  - Creating files and folders
  - Moving around to different folders
- Editing and saving text files
  - From the command line!
- File permissions
  - From the command line: Viewing and changing them
- Bash scripts
  - Creating and running common scripts

# What is Linux?

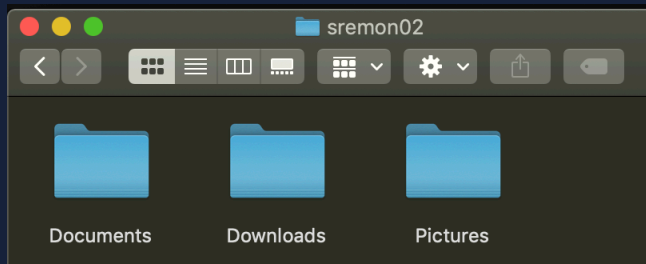
- Operation System, just like Windows or OSX (Macs)
- Free!
- Used by Amazon, Facebook, Google, many phones....
- 



- Using Linux can seem overwhelming at first
  - Don't panic!
  - There are many online resources and cheat sheets

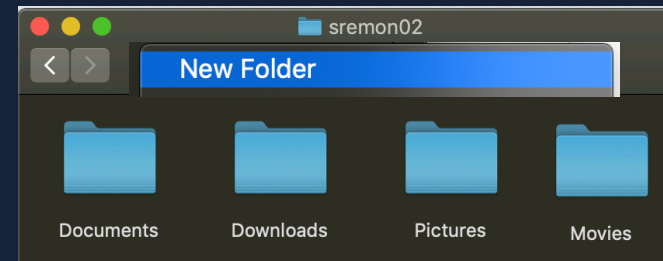
# Shell / Command Line / Terminal / Console

- A place to type in commands to the computer
  - If you can click it, you can type it
- The expected way to navigate Linux



```
[sremon02@login001 ~]$ ls
Documents Downloads Pictures
[sremon02@login001 ~]$
```

*Using the command line to get a list of folders*



```
[sremon02@login001 ~]$ ls
Documents Downloads Pictures
[sremon02@login001 ~]$ mkdir Movies
[sremon02@login001 ~]$ ls
Documents Downloads Movies Pictures
[sremon02@login001 ~]$
```

*Using the command line to make a new folder*

# Aside: Terminology

## CLI:

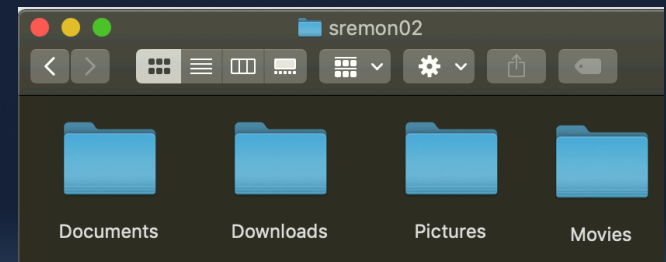
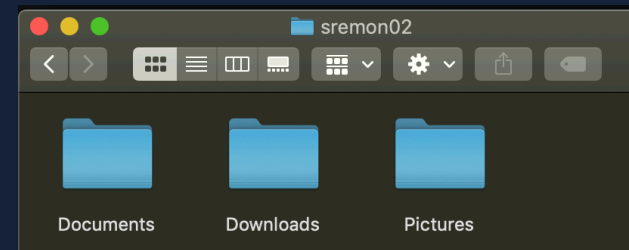
- Command Line Interface
- Typing commands in.
- What we're going to learn!

```
[sremon02@login001 ~]$ ls
Documents Downloads Pictures
[sremon02@login001 ~]$
```

```
[sremon02@login001 ~]$ ls
Documents Downloads Pictures
[sremon02@login001 ~]$ mkdir Movies
[sremon02@login001 ~]$ ls
Documents Downloads Movies Pictures
[sremon02@login001 ~]$
```

## GUI:

- Graphical User Interface
- Clicking graphics.
- What you're probably used to!

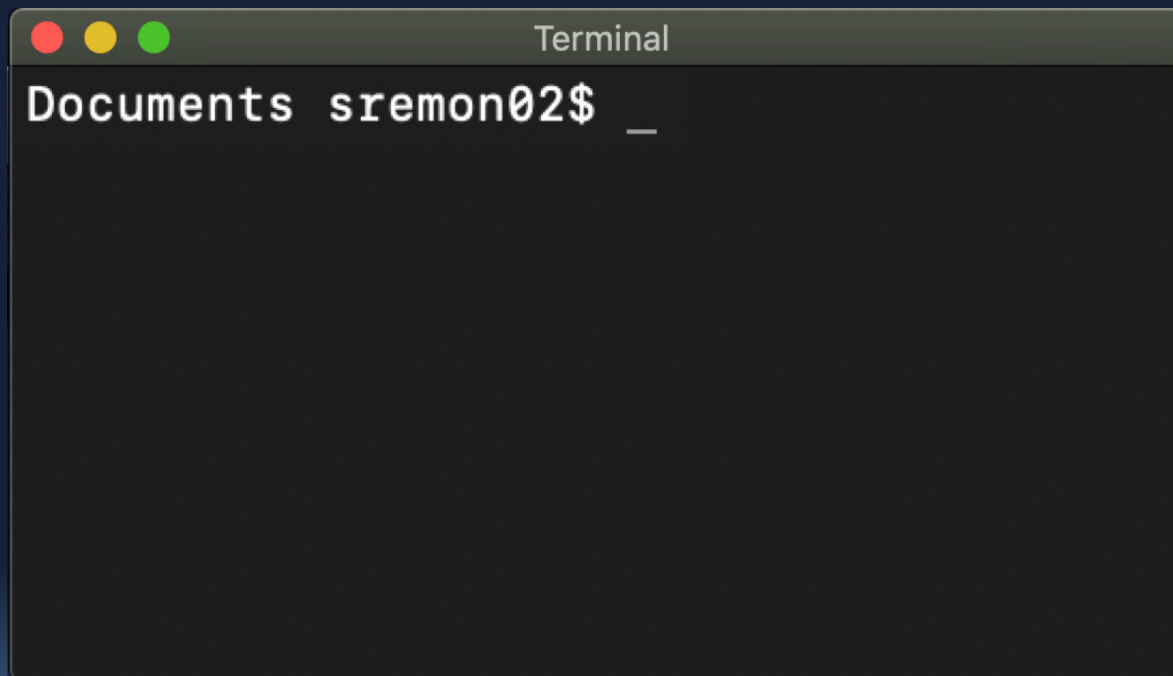


# COMMAND BASICS

# Command Line Breakdown

You'll always see a **prompt** with:

1. What folder you're currently in ("*Documents*")
2. What user is logged in (here, "sremon02")
3. Waiting for input! (\$)

A screenshot of a macOS Terminal window. The title bar at the top is dark gray and contains the word "Terminal" in white. Below the title bar, the terminal content is displayed on a black background. The first line shows the prompt "Documents sremon02\$ \_" in white text. The prompt consists of the current directory "Documents", the username "sremon02", a dollar sign "\$", and a cursor character "\_".

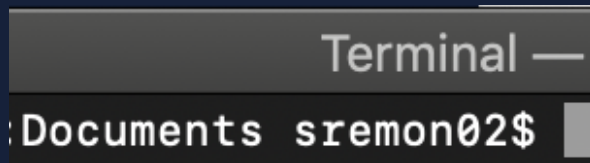
```
Terminal
Documents sremon02$ _
```

# Command Line Breakdown: Ordering

The order might be interchanged:

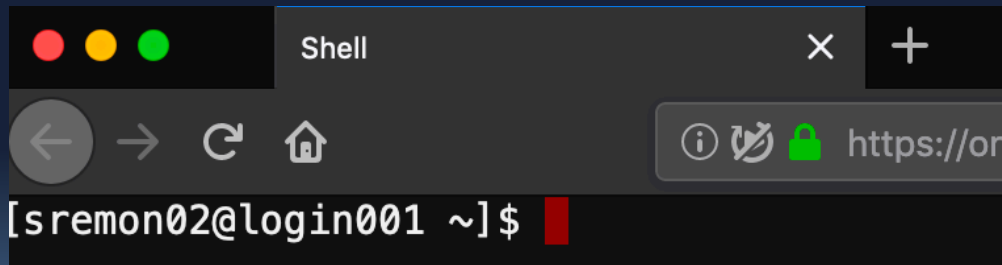
- On my local laptop, the folder is displayed first:

```
Folder username$ <you type commands here>
```



- In the Tufts Research Cluster, the username is displayed first:

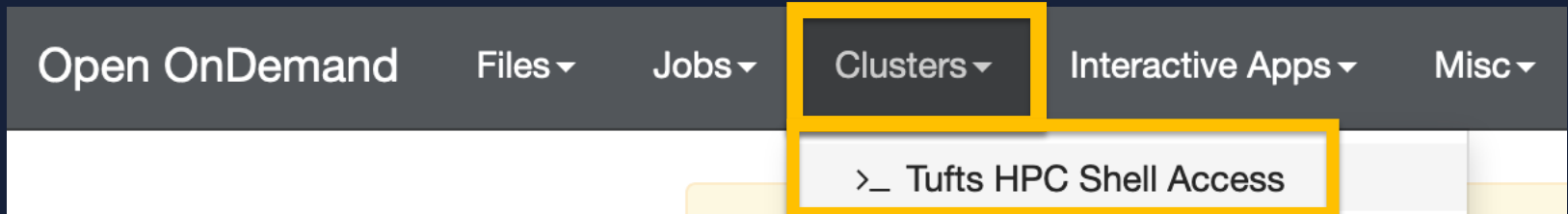
```
[Username ~folder~] $ <you type commands here>
```





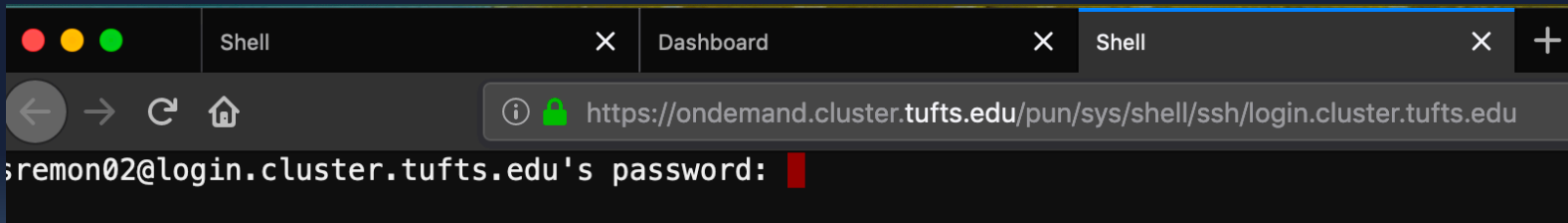
# Log In! This Workshop is Interactive

1. Open a browser and log in to <https://ondemand.cluster.tufts.edu/>
2. Choose **Tufts HPC Shell Access** under **Clusters**.



3. Type in your password at the prompt

*Nothing will appear, but it's being typed! Linux doesn't show passwords.*



4. Hit return/enter when you're done

# What's a Command?

- You can't just type anything!
  - There are predefined commands
- Try typing “Wipe the screen” and hitting enter
- What happens?

```
[sremon02@login001 ~]$ wipe the screen  
-bash: wipe: command not found
```

# First Command: Clear

- Your first command: `clear`
  - Type `clear` and hit enter
  - What happened?

```
[sremon02@login001 ~]$ clear
```

- Commands are case sensitive!
  - Try `CLEAR`
  - Try `Clear`

```
[sremon02@login001 ~]$ CLEAR
-bash: CLEAR: command not found
[sremon02@login001 ~]$ Clear
-bash: Clear: command not found
[sremon02@login001 ~]$
```

# System Commands: Try these out!

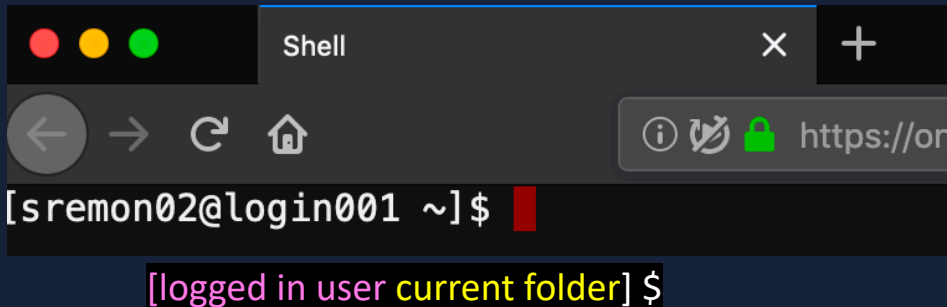
How to get user information:

- `whoami`
  - Display username
- `groups`
  - Display the groups you belong to

```
[sremon02@login001 ~]$ whoami
sremon02
[sremon02@login001 ~]$ groups
facstaff
```

# What is that ~ ?

- ~ is the **home folder**
  - The base folder you'll work out of
  - A command line shorthand



The screenshot shows a terminal window titled "Shell" with a dark background. The prompt is `[sremon02@login001 ~]$`. Below the prompt, a tooltip or help text is visible: `[logged in user current folder] $`.

## Examples:

- On Windows, my home folder is

`C:\Users\susi\`

- `C:\Users\susi\Downloads`
- `C:\Users\susi\Documents`

- On a Mac, my home folder is

`MacintoshHD/Users/susi/`

- `MacintoshHD/Users/susi/Downloads`
- `MacintoshHD/Users/susi/Documents`

It's far easier to type `~\Documents` and

`~\Downloads`

Notice we separate folders with `\` or `/` when writing them out  
– OS dependent

# Quick Terminology

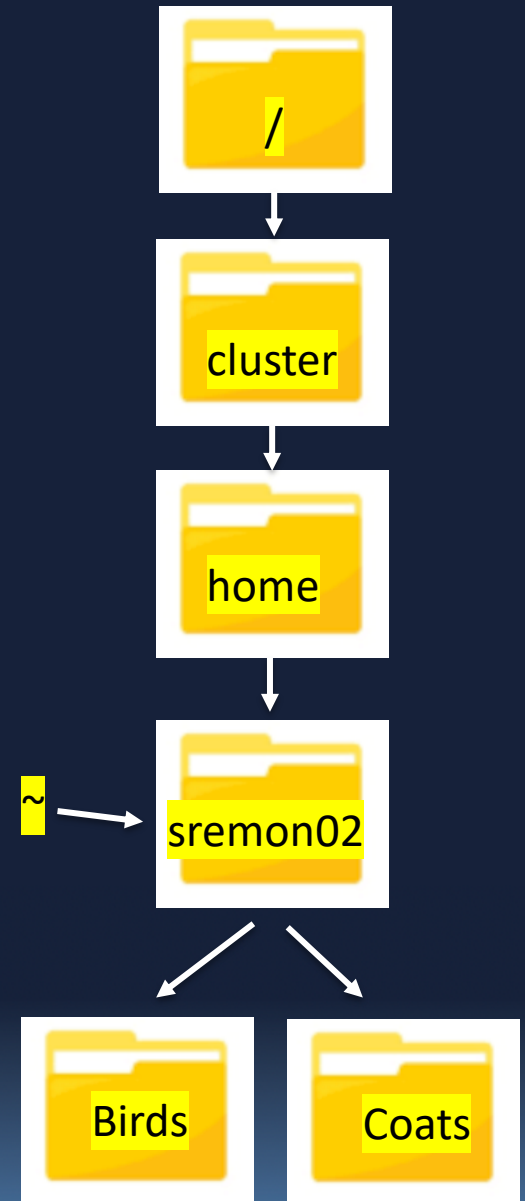
- **Directory**
  - Folder
  - Documents, Downloads, Pictures....
- **Root directory:**
  - The main folder of the computer
  - In Linux: /

## Examples:

- Windows:
  - My home directory (~) is C:\Users\susi\  
C:\Users\susi\
    - The root directory is C:\
- Mac:
  - My home directory (~) is MacintoshHD/Users/susi/  
MacintoshHD/Users/susi/
    - The root directory is Macintosh HD/
- Linux:
  - My home directory (~) is /home/susi  
/home/susi
    - The root directory is /

# Directory Structure

- Here is the directory structure
  - A diagram of the file system
- / is used for two things
  - The **root directory** at the top of a Linux file system
  - When writing a path, separating directories:  
`/cluster/home/sremon02/Birds`



# Common Commands

- Try this out: `pwd`
  - print working directory
  - "Where are you right now?"

```
[sremon02@login001 ~]$ pwd
/cluster/home/sremon02
[sremon02@login001 ~]$
```

- Try this out: `ls`
  - list everything in this directory
  - Blank? The folder is empty.

```
[sremon02@login001 ~]$ ls
Documents Downloads Movies
[sremon02@login001 ~]$
```

- `ls` and `pwd` are two of the most common commands



# DIRECTORY COMMANDS



Tufts Technology  
Services

# What Else Can We Do?

- Many commands are shorthand for the words you'd use
  - `pwd`
    - print **w**orking **d**irectory
  - `ls`
    - **l**ist everything in this directory
- What if you want to make a new folder?
- Or, using Shell terminology, make a new directory?

# Making Directories

- `mkdir <name of directory>`
  - make a new `directory` called `<name of directory>`
- Try it out: `mkdir Birds`
  - It doesn't display anything, so follow that with an `ls`

```
[sremon02@login001 ~]$ ls
[sremon02@login001 ~]$ mkdir Birds
[sremon02@login001 ~]$ ls
Birds
```

*The shell only displays things when we ask it to!*

# Aside: Terminology

- Basic structure of Linux commands:
  - `command`
  - `command argument1 argument2 argument3...`
  - `mkdir Birds`
- Not every command needs arguments
  - `ls`
  - `pwd`
  - `clear`

# Making Directories: Naming

- Try this out: `mkdir Winter Coats`
  - Follow that with an `ls`

```
[sremon02@login001 ~]$ mkdir Winter Coats
[sremon02@login001 ~]$ ls
Birds Coats Winter
[sremon02@login001 ~]$
```

- What happened?

# Terminology: Strings

- Text in a command is called a **string**
  - Spaces separate strings
    - Each Word Here Is A Separate String
  - Quotations tell the shell to ignore the spaces
    - “This is one string”
- **Birds**
  - The shell sees **one** string
- **Winter Coats**
  - The shell sees **two** strings
    - **Winter** is one string
    - **Coats** is another string
- **“Winter Coats”**
  - The shell sees **one** string
    - **“Winter Coats”**

# Commands: Quick Glance

- Basic structure of Linux commands:
  - `command`
  - `command argument1 argument2 argument3...`
- Each argument is entered as a separate string
  - `mkdir Birds`
    - One string = One argument = One new directory
    - Make a directory called Birds
  - `mkdir Winter Coats`
    - Two strings = Two arguments = Two new directories
    - Make a directory called Winter
    - Then make a directory called Coats

# Strings with Quotes

- We want the shell to see one string, not two
- We can use quotes!
  - Try `mkdir "Winter Coats"`
  - Make a directory called "Winter Coats"

```
[sremon02@login001 ~]$ mkdir "Winter Coats"  
[sremon02@login001 ~]$ ls  
Birds  Coats  Winter  Winter Coats  
[sremon02@login001 ~]$
```



# Recap: Common Commands

- `clear`
  - `clear` the screen
- `pwd`
  - print `w`orking `d`irectory
- `ls`
  - list everything in this directory
- `mkdir <name of directory>`
  - make a new `dir`ectory called `<name of directory>`
- `mkdir <name of directory1> <name of directory2>`
  - make multiple new `dir`ectories called `<name of directory1>` and `<name of directory2>`
- “strings go in quotes”
  - The shell treats this as one argument

# Changing to a Different Folder

- Where directory are we in right now?
  - Enter the command `pwd`
- What directories are inside this directory?
  - Enter the command `ls`
- Our goal?
  - Be inside the “Birds” folder.
- In a GUI, we can double click a folder to go into it.
  - But we can’t double click in the shell!

```
[sremon02@login001 ~]$ pwd
/cluster/home/sremon02
[sremon02@login001 ~]$ ls
Birds Coats Winter Winter Coats
[sremon02@login001 ~]$
```

# Changing Directories

- We'll need to use a **command** with an **argument**
  - **Change our current directory to the Bird directory**
- Enter this command: **cd Birds**
  - **change directory to Birds**
  - Then do a **pwd** to see if it worked

```
[sremon02@login001 ~]$ pwd
/cluster/home/sremon02
[sremon02@login001 ~]$ ls
Birds  Coats  Winter  Winter  Coats
[sremon02@login001 ~]$ cd Birds/
[sremon02@login001 Birds]$ pwd
/cluster/home/sremon02/Birds
[sremon02@login001 Birds]$ █
```

*Notice that the prompt changed, too! It always has your current folder.*

```
[sremon02@login001 Birds folder] $
```

# Changing Directories: Navigation

- How did we get here?
  - `cd <where you want to go>`
  - `cd Birds`
    - Change our current directory to the Bird directory
- But.... How do we go back?

```
[sremon02@login001 ~]$ cd Birds/  
[sremon02@login001 Birds]$ pwd  
/cluster/home/sremon02/Birds  
[sremon02@login001 Birds]$ cd ?????????? █
```

- There are several ways!

# Relative vs Absolute Paths

- *Path* is like shell word for *address*.
- Relative paths
  - “From here, Steve’s house is down the road on the left”
  - Birds
  - `cd Birds`
- Absolute paths
  - “Steve’s exact address is 555 Main Street, Somerville MA”
  - `/cluster/home/sremon02/Birds`
  - `cd /cluster/home/sremon02/Birds`

```
[sremon02@login001 Birds]$ pwd
/cluster/home/sremon02/Birds
[sremon02@login001 Birds]$ cd ?????????? █
```

- We’re in the Birds directory. How could we get back to the `/sremon02` directory?

# Changing Directory with Paths

- We can use an absolute path!
  - Try it – **yours will be different.**
  - First, **pwd** to see your path. Then **cd** around.
  - Try different paths! Note that the prompt changes.
- With absolute paths, you can navigate anywhere. For me:

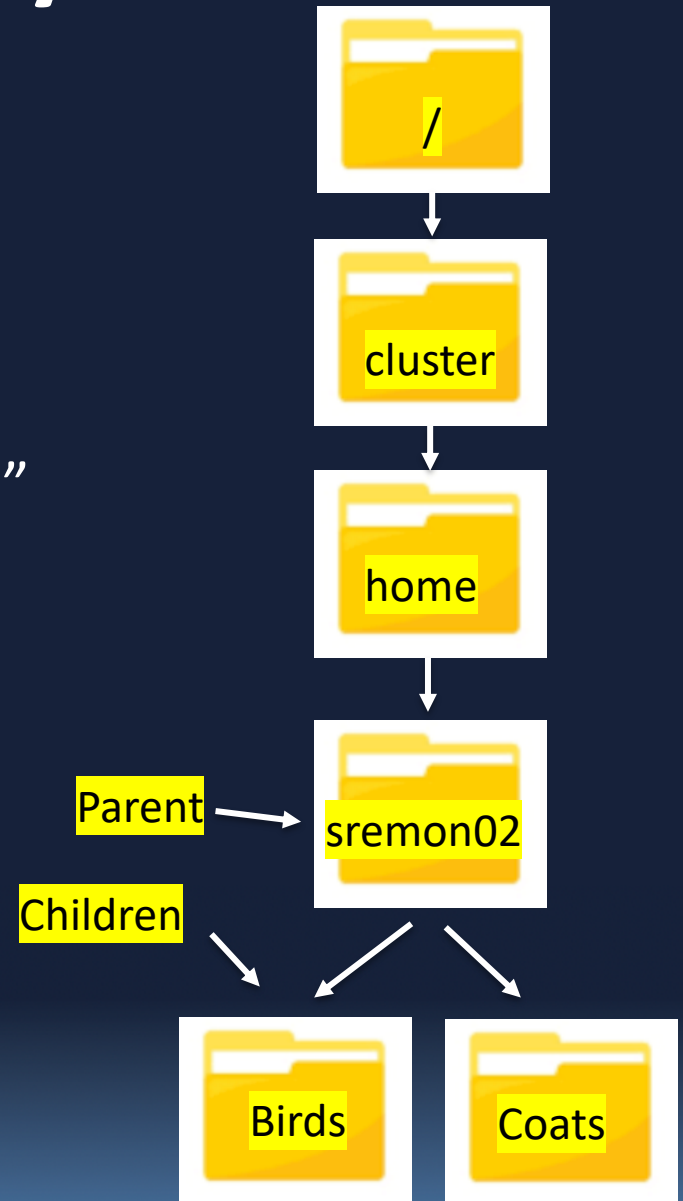
```
[sremon02@login001 ~]$ cd /cluster/home/  
[sremon02@login001 home]$ pwd  
/cluster/home  
[sremon02@login001 home]$ cd /cluster/  
[sremon02@login001 cluster]$ pwd  
/cluster  
[sremon02@login001 cluster]$ cd /cluster/home/sremon02/  
[sremon02@login001 ~]$ pwd  
/cluster/home/sremon02  
[sremon02@login001 ~]$
```

# Changing Directory: Parent

- Here is the directory structure
  - A diagram of the file system
- **Parent directory**
  - “Containing folder”
  - You might also hear “child directory” or “sibling directory”

From `/cluster/home/sremon02/Birds`

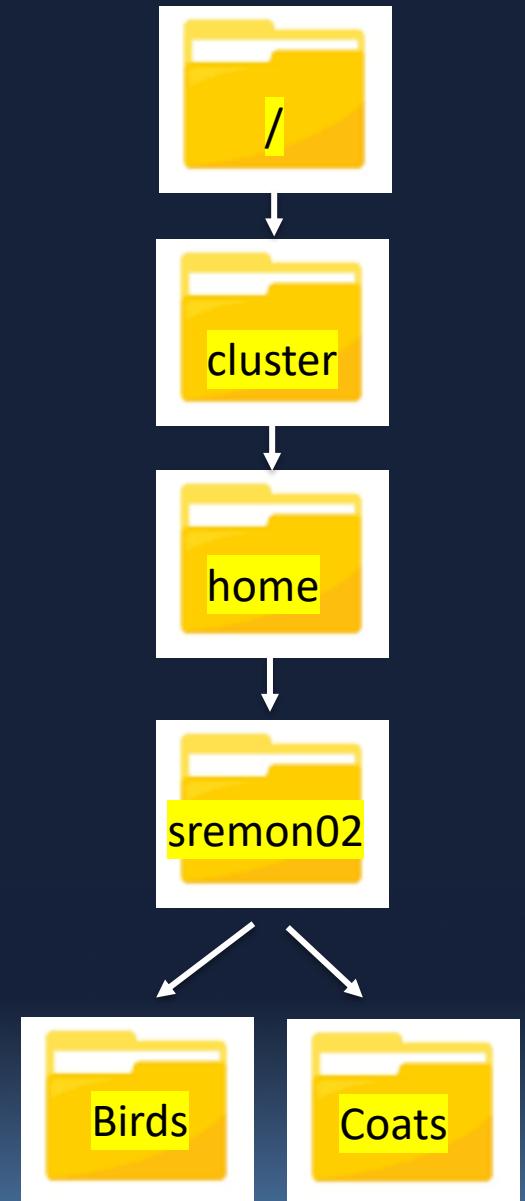
- Parent directory:
  - `/cluster/home/sremon02`



# Parent: ..

- `..` is a shortcut for “parent directory”
- From here:
  - `/cluster/home/sremon02/Birds`
- We *could* enter the command:
  - `cd /cluster/home/sremon02/`
- But it’s a lot easier to enter:
  - `cd ..`
- Try it!

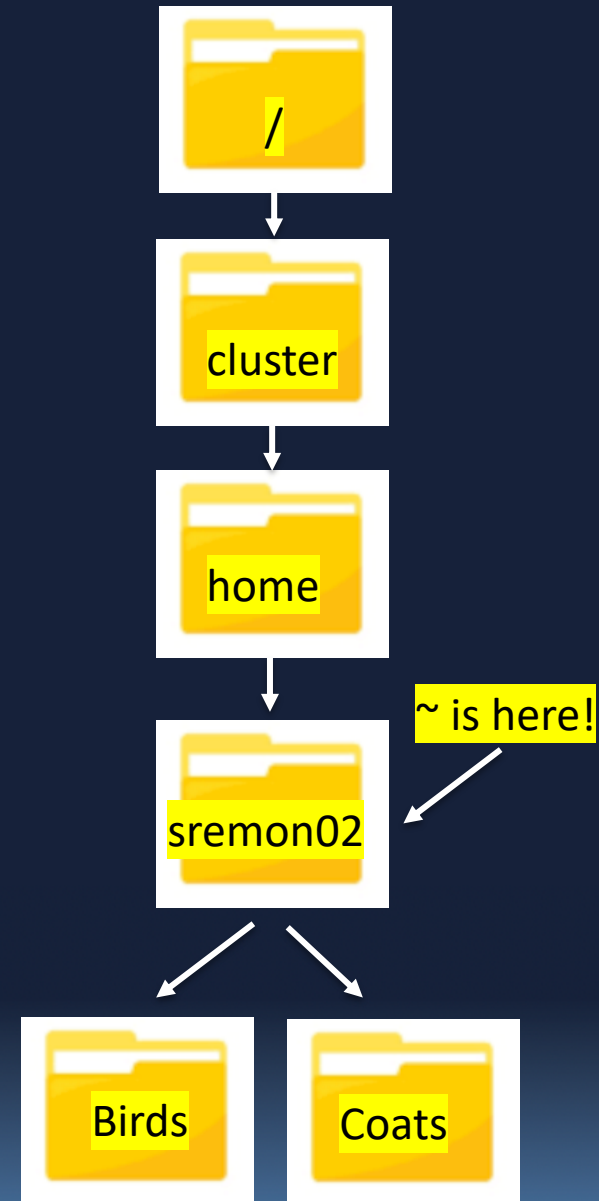
```
[sremon02@login001 Birds]$ pwd
/cluster/home/sremon02/Birds
[sremon02@login001 Birds]$ cd ..
[sremon02@login001 ~]$ pwd
/cluster/home/sremon02
[sremon02@login001 ~]$
```





# Home: ~

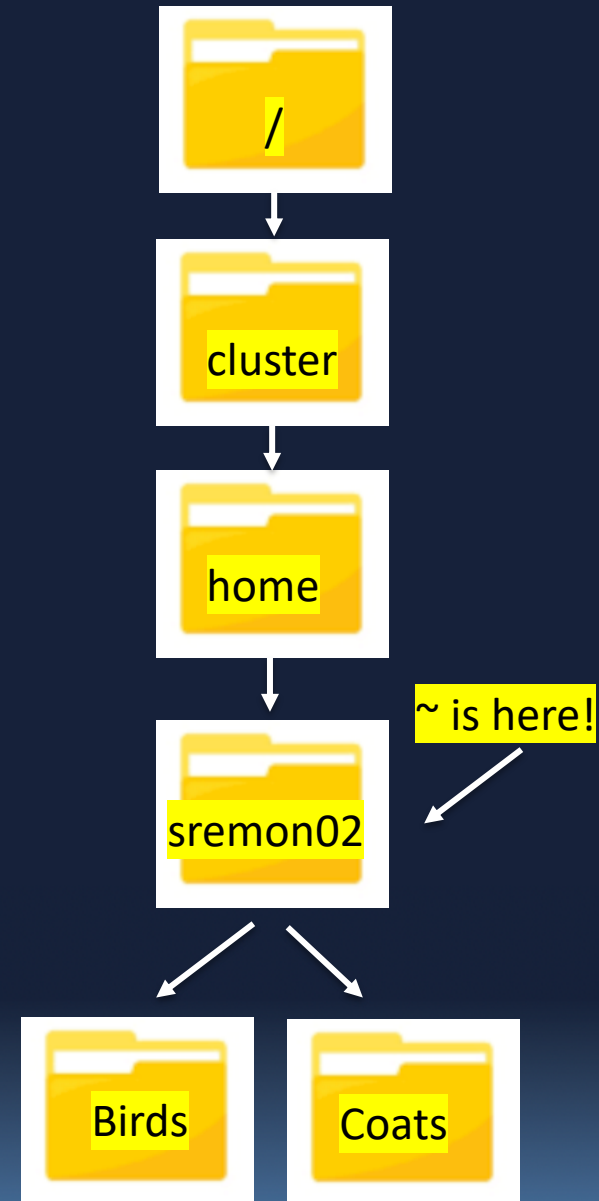
- ~ is shorthand for a user's "home" path
  - Their default location
  - Don't confuse it with folders called "home"!
- My home path is
  - `/cluster/home/sremon02/`
- ~ is like an absolute path
  - I can use it anywhere to get back to `/cluster/home/sremon02/`



# Using ~

- We *could* enter the command:
  - `cd /cluster/home/sremon02/`
- But it's a lot easier to enter:
  - `cd ~`
- This works from anywhere.
- Try it!

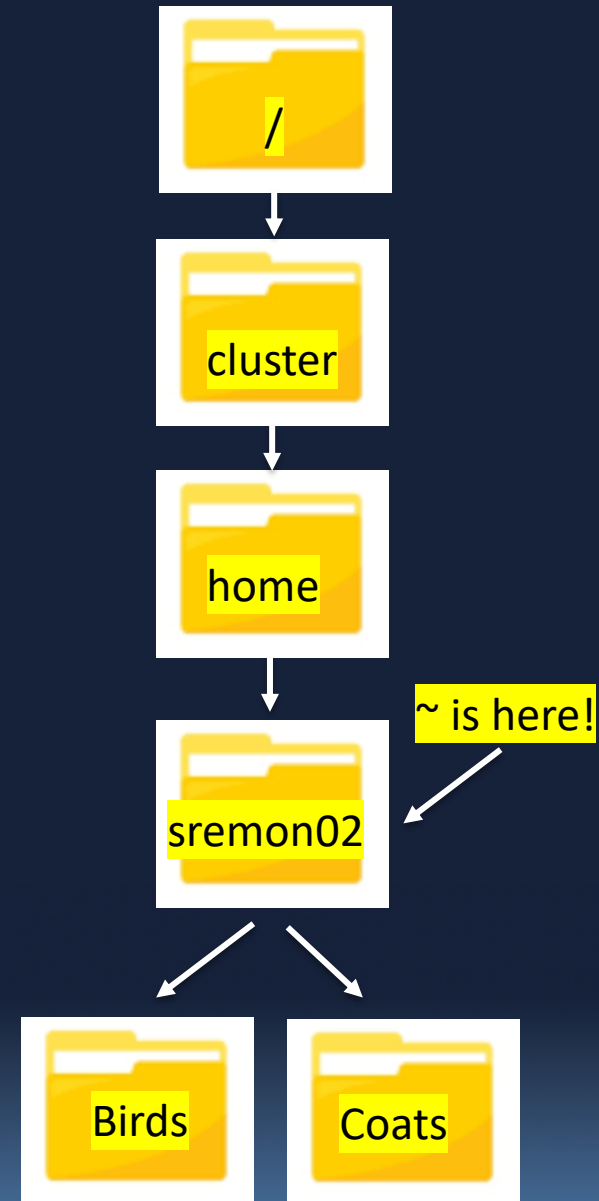
```
[sremon02@login001 Birds]$ pwd
/cluster/home/sremon02/Birds
[sremon02@login001 Birds]$ cd ~
[sremon02@login001 ~]$ pwd
/cluster/home/sremon02
[sremon02@login001 ~]$ cd /cluster
[sremon02@login001 cluster]$ pwd
/cluster
[sremon02@login001 cluster]$ cd ~
[sremon02@login001 ~]$ pwd
/cluster/home/sremon02
[sremon02@login001 ~]$
```



# Combining Shortcuts: ~

- We *could* enter the commands:
  - `cd ~`
  - `cd Birds`
- But it's a lot easier to enter:
  - `cd ~/Birds`
- Because it starts with ~, it works from anywhere.
- Try it!

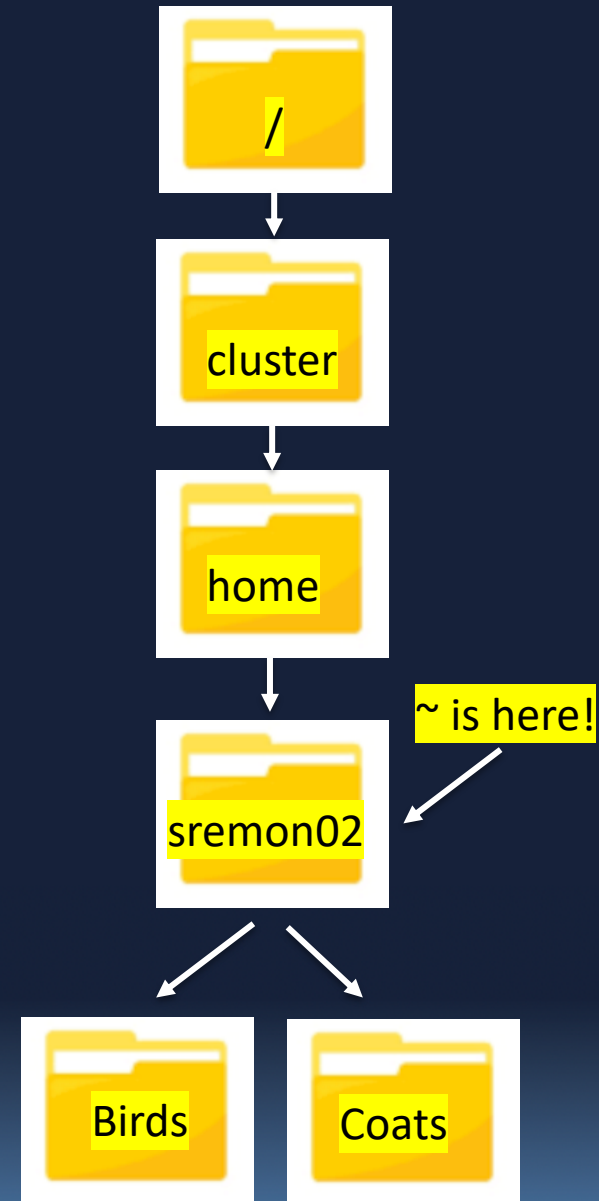
```
[sremon02@login001 cluster]$ cd ~/Birds/  
[sremon02@login001 Birds]$ pwd  
/cluster/home/sremon02/Birds
```



# Combining Shortcuts: ..

- What if we're in "Birds" but want to be in "Coats"?
- We *could* enter the commands:
  - `cd ..`
  - `cd Coats`
- But it's a lot easier to enter:
  - `cd ../Coats`
- Try it!

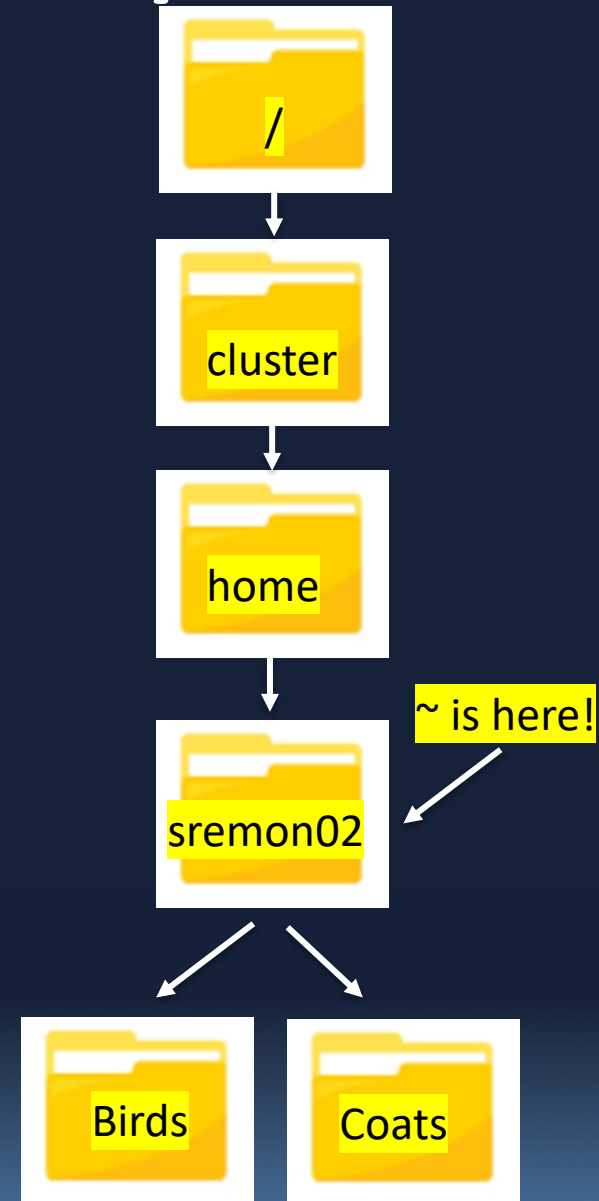
```
[sremon02@login001 Birds]$ pwd
/cluster/home/sremon02/Birds
[sremon02@login001 Birds]$ cd ../Coats
[sremon02@login001 Coats]$ pwd
/cluster/home/sremon02/Coats
[sremon02@login001 Coats]$
```



# Combining Shortcuts: ../..

- What if we're in "Birds" but want to be in "home"?
- We *could* enter the commands:
  - `cd ..`
  - `cd ..`
- But it's a lot easier to enter:
  - `cd ../../`
- Try it!

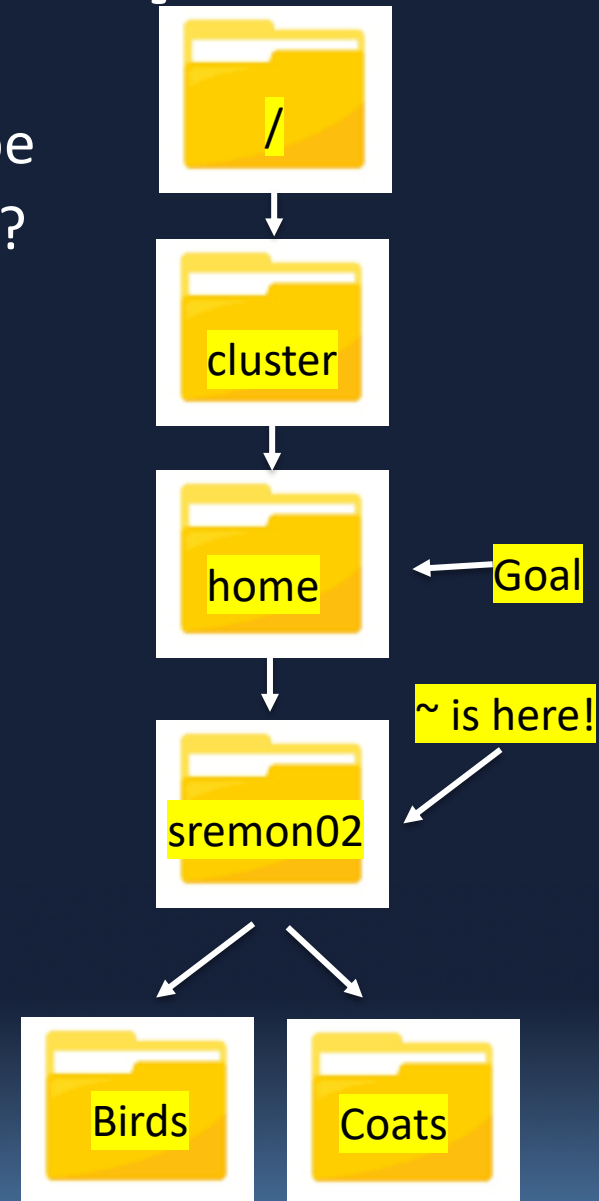
```
[sremon02@login001 Birds]$ pwd
/cluster/home/sremon02/Birds
[sremon02@login001 Birds]$ cd ../../
[sremon02@login001 home]$ pwd
/cluster/home
[sremon02@login001 home]$
```



# Combining Shortcuts: ~/..

- What if we're lost, but we know we want to be in the parent directory of our home directory?
- We *could* enter the commands:
  - `cd ~`
  - `cd ..`
- But it's a lot easier to enter:
  - `cd ~/..`
- Try it!

```
[sremon02@login001 Coats]$ pwd
/cluster/home/sremon02/Coats
[sremon02@login001 Coats]$ cd ~/..
[sremon02@login001 home]$ pwd
/cluster/home
[sremon02@login001 home]$
```



# More Shortcuts

- Directory paths can get long!
  - Shortcuts are great
  - ~ for “home”
  - .. for “parent”
- Next up.... Tab autocomplete!

# Protip: Tab Complete

1. Enter `cd ~` (to get you to home)
2. Now, type `cd B` but don't hit enter:

```
[sremon02@login001 home]$ cd ~  
[sremon02@login001 ~]$ cd B
```

3. Instead, hit tab!  
`Birds` autofills

```
[sremon02@login001 home]$ cd ~  
[sremon02@login001 ~]$ cd Birds/
```

4. Then hit Enter

This can be used on files, directories, anything!

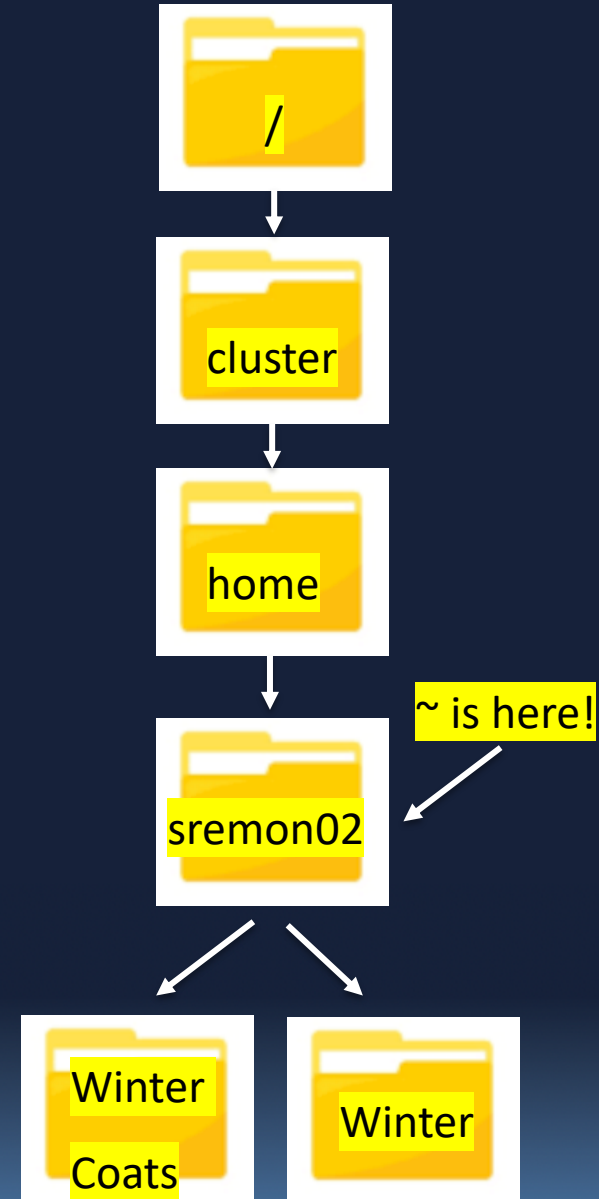


# Tab Complete with Multiple Options

- What if the letter isn't unique?
- Enter
  - `cd ~` (to get you to home)
  - `ls` (so you can see your folders)
- Now, type this but **don't hit enter**:
  - `cd W`
  - Hit tab – twice.
  - You're given the list of possibilities

```
[sremon02@login001 ~]$ cd W
```

```
[sremon02@login001 ~]$ cd Winter  
Winter/      Winter Coats/
```



# Recap: Navigation Commands

- `cd <directory>`
  - change `directory` to `<directory>`
- Relative path: “Down the road on the left”
  - `cd Birds`
- Absolute path: “555 Main Street, Somerville MA”
  - `cd /cluster/home/sremon02/Birds`
- Shorthand:
  - `..`
    - *Parent directory* a.k.a. “Containing folder”
    - `cd ..`
  - `~`
    - *home* a.k.a. always (for me) `/cluster/home/sremon02`
    - `cd ~`

# DIRECTORY MANIPULATION



Tufts Technology  
Services

# Other Directory Actions

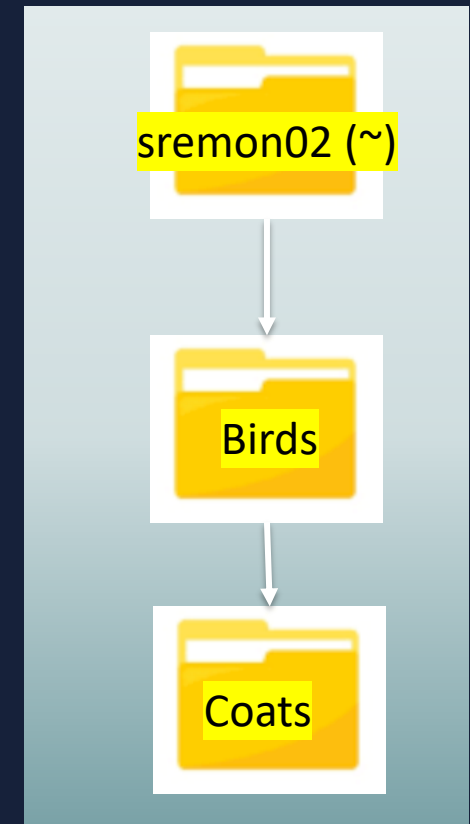
- You've seen this!
- `mkdir <name of directory>`
  - `make` a new `directory` called `<name of directory>`
- What else can we do? Anything!
  - Move directories
  - Copy directories
  - Remove directories – **carefully!**

```
[sremon02@login001 ~]$ ls
[sremon02@login001 ~]$ mkdir Birds
[sremon02@login001 ~]$ ls
Birds
```

# Moving Directories

- Let's move Coats to be inside of Birds

```
[sremon02@login001 ~]$ ls  
Birds Coats Winter Winter Coats  
[sremon02@login001 ~]$
```



# mv: Move Directory

- Let's move Coats to be inside of Birds
- `mv <source> <destination>`
  - The `move` command
  - Takes two `arguments`:
    - What we're moving (Coats)
    - Where it's going to end up (Birds/Coats)
- `mv Coats/ Birds/Coats`

```
[sremon02@login001 ~]$ mv Coats/ Birds/Coats
[sremon02@login001 ~]$ ls
Birds  Winter  Winter  Coats
[sremon02@login001 ~]$
```

- Do you know how to check if it worked?

*Be careful when specifying the target name, since mv will silently overwrite any existing file or directory with the same name!*

# Renaming Directories

- No "rename" command
- Use the "move" command
  - `mv <directory> <new name>`
- Try
  - `mv Birds/ myBird/`
- Check if it worked: `ls`

```
[sremon02@login001 ~]$ ls
Birds  Winter  Winter Coats
[sremon02@login001 ~]$ mv Birds/ myBird
[sremon02@login001 ~]$ ls
myBird Winter  Winter Coats
[sremon02@login001 ~]$
```

# Renaming and Moving Directories

- Can rename while moving
  - `mv myBird/ Winter/Robins`
- Check if it worked
  - `ls`
  - `cd Winter`
  - `ls`
- Then move the directory back:
  - `mv Robins ../Birds`

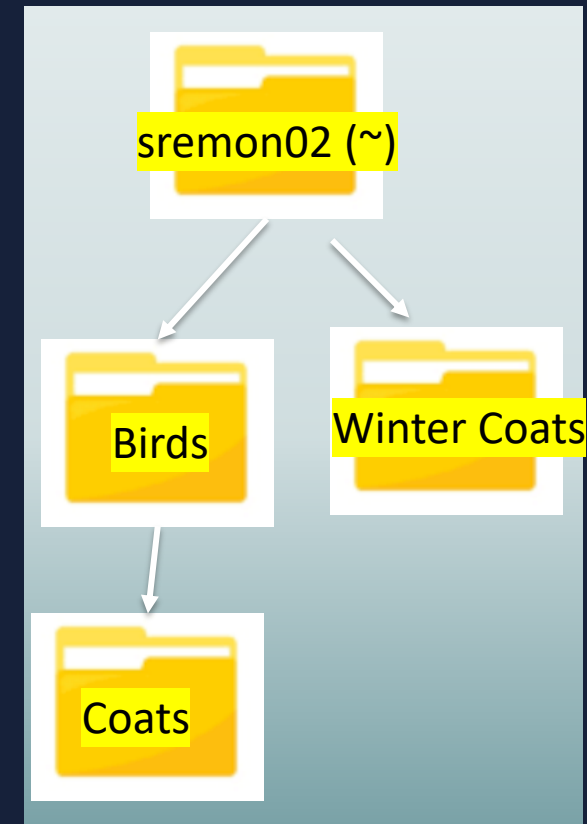
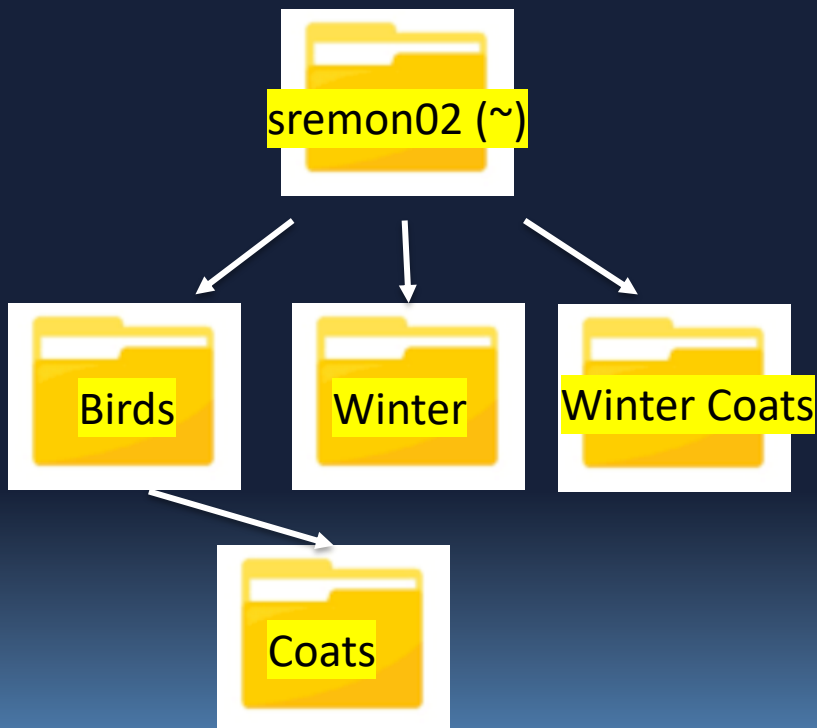
```
[sremon02@login001 ~]$ mv myBird/ Winter/Robins
[sremon02@login001 ~]$ ls
Winter  Winter Coats
[sremon02@login001 ~]$ cd Winter
[sremon02@login001 Winter]$ ls
Robins
[sremon02@login001 Winter]$ mv Robins/ ../Birds
[sremon02@login001 Winter]$ ls
[sremon02@login001 Winter]$ cd ..
[sremon02@login001 ~]$ ls
Birds  Winter  Winter Coats
[sremon02@login001 ~]$
```



# Deleting Directories

- Let's delete the Winter directory
  - a.k.a. **remove** the directory

```
[sremon02@login001 ~]$ ls  
Birds  Winter  Winter Coats
```



# rmdir: Remove Directory

- Let's remove the Winter directory
- `rmdir <directory>`
  - The `remove directory` command
  - Takes one `argument`:
    - What we're removing (Winter)
- `rmdir Winter`

```
[sremon02@login001 ~]$ ls
Birds  Winter  Winter  Coats
[sremon02@login001 ~]$ rmdir Winter
```

- Do you know how to check if it worked?

*Be **INCREDIBLY** careful with `rm` and `rmdir`! Don't **EVER** just enter "`rm`" or "`rmdir`" without an argument, and **always** check your argument! You can easily delete the whole file system – Linux will not ask you if you're sure. It assumes you know what you're doing!*

# Recap: Directory Manipulation

- `mkdir <name of directory>`
  - make a new **directory** called `<name of directory>`
- `mv <source> <destination>`
  - The **move** command
  - Also used to rename things!
- `rmdir <directory>`
  - The **remove directory** command
  - BE CAREFUL
- Check if your commands worked with `ls`

# FILES

# File Extensions

- Define the type of file. E.g.:
  - .txt – text file
    - myRecipe.txt
    - groceryList.txt
  - .mp4– movie file
    - casablanca.mp4
    - batman.mp4
  - .mp3 – music file
    - beatlesYesterday.mp3
    - madonnaVogue.mp3
- You can google them!

# Text Editors

## Windows / Mac

- Microsoft Word
- TextEdit
- Notepad
- Wordpad

## Linux

- nano
- vi
- vim
- emacs

# Nano Overview

- Accessed through the command line
- Installed by default on most Linux machines
- Very basic and beginner friendly
- You can't:
  - Style text – no bold, underline, or italics
  - Change font size or color – only the default!
- You can:
  - Type!
  - Cut, copy, and paste
  - Search a text file

# Starting Nano

- To start a new file:
  - `nano <name of file>`
  - `nano cookieRecipe.txt`
- To open an existing file:
  - `nano <name of file>`
  - `nano cookieRecipe.txt`
- Same command! Nano:
  - Checks if it exists
  - Creates the file if not
  - Opens the file

Try it!

`nano cookieRecipe.txt`

```
[sremon02@login001 ~]$ ls
Birds  Winter Coats
[sremon02@login001 ~]$ nano cookieRecipe.txt
```



# Nano Basics

Nano opens right in the command prompt!

*Current program (nano)*

*Current file (cookieRecipe.txt)*

GNU nano 2.0.9

File: cookieRecipe.txt

*Where you'll type your text!*

**^G** Get Help   **^O** WriteOut   **^R** Read File   **^Y** Prev Page   **^K** Cut Text   **^C** Cur Pos  
**^X** Exit   **^J** Justify   **^W** Where Is   **^V** Next Page   **^U** UnCut Tex   **^T** To Spell

*Keyboard shortcuts. ^ is the Control key (a.k.a. Ctrl)*

Important commands:

- **WriteOut**: Save
- **Exit**: Quit back to command line

# Nano: Editing & Saving

1. Add some text.
2. Hit **Control-O** to save

```
GNU nano 2.0.9      File: cookieRecipe.txt      Modified
First, buy butter.

File Name to Write: cookieRecipe.txt
^G Get Help      ^T To Files      M-M Mac Format      M-P Prepend
^C Cancel        M-D DOS Format   M-A Append        M-B Backup File
```

3. Confirm with **Enter**.

```
GNU nano 2.0.9      File: cookieRecipe.txt
First, buy butter.

[ Wrote 1 line ]
^G Get Help      ^O WriteOut      ^R Read File      ^Y Prev Page      ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is      ^V Next Page      ^U UnCut Text    ^T To Spell
```

# Nano: Quitting & Reopening

1. Hit **Ctrl-X** to quit
2. **ls** to see your new file

```
[sremon02@login001 ~]$ ls  
Birds  cookieRecipe.txt  Winter Coats
```

Colors will be different in different command prompts!

Here:

- White: Files
- Blue: Directories

# Nano: With Tab

1. Reopen your file – try tab autocomplete:

`nano cookie`

```
[sremon02@login001 ~]$ nano cookie
```

2. Hit **Tab**!

```
[sremon02@login001 ~]$ nano cookieRecipe.txt
```

3. Hit **Enter** to open the file.
4. Hit **Ctrl-X** to quit again.

# Nano: Opening a file

- Can use relative and absolute paths
- Depending on what directory you're in, any of these might work:
  - `nano cookieRecipe.txt`
  - `nano ~/cookieRecipe.txt`
  - `nano ../cookieRecipe.txt`
- This will always work:
  - `nano /path/to/filename`
  - `nano /cluster/home/sremon02/cookieRecipe.txt`

Try navigating around opening and closing the file.

- Don't forget about tab autocomplete!

# New Shortcut: Up Arrow

- Shows previously entered command
- Can go back almost indefinitely

```
[sremon02@login001 ~]$ pwd  
/cluster/home/sremon02  
[sremon02@login001 ~]$
```

- Hitting the Up arrow:

```
[sremon02@login001 ~]$ pwd  
/cluster/home/sremon02  
[sremon02@login001 ~]$ pwd
```

# Working with Files

- Just like directories, you can:
  - Move or rename a file (`mv <filename> <destination>` )
  - Carefully delete a file (`rm <filename>` )
  - Copy a file (`cp <filename> <destinationFilename>` )
- Remember, moving or copying will **overwrite** anything with the same name **without warning!**
- Can use relative or absolute paths:
  - `mv cookieRecipe.txt ~/Birds/cookieRecipe.txt`
  - `mv ~/cookieRecipe.txt ../../cookieRecipe.txt`
  - `mv /cluster/home/sremon02/cookieRecipe.txt ../../cookieRecipe.txt`

# New Shortcut: .

- What if I'm inside `/cluster/home/sremon02/Birds` and want `cookieRecipe.txt` here?

```
mv ../cookieRecipe.txt /cluster/home/sremon02/Birds/
```

- That's very long.

- New shortcut!

*. is the current directory*

```
mv ../cookieRecipe.txt .
```

```
[sremon02@login001 Birds]$ mv ../cookieRecipe.txt .  
[sremon02@login001 Birds]$ ls  
Coats  cookieRecipe.txt  
[sremon02@login001 Birds]$
```



# Recap: Files

- nano: Text editor  
`nano cookieRecipe.txt`
- Just like directories, you can:
  - Move a file (`mv <currentLocation> <Destination>` )
  - Carefully delete a file (`rm <filename>` )
  - Copy a file (`cp <filename> <destinationFilename>` )
- The **UP** arrow gives command history
- `.` is *the current directory*  
`mv ../cookieRecipe.txt .`

# SCRIPTS

# Scripts: Overview

- A file with commands in it
- New file extension:
  - .sh – shell file
- Why?
  - Save commands to run later
  - Run commands based on something else

# Writing a Basic Script

- Start a new file:
  - `nano myScript.sh`

```
[sremon02@login001 ~]$ cd ~  
[sremon02@login001 ~]$ ls  
Birds  Winter Coats  
[sremon02@login001 ~]$ nano myScript.sh
```

# Adding a Command

- Add a command to the file:

`pwd`

```
GNU nano 2.0.9           File: myScript.sh           Modified
pwd
[ New File ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Tex ^T To Spell
```

- Save the file (`Ctrl-O`, then `enter`)
- Quit nano (`Ctrl-X`)

# Aside: Permissions

- To run a script from the shell, the command is `./<name of script>`
  - For example, `./myScript.sh`
- However, this won't work by default!
  - Linux is strict about **permissions**
  - Who can do what?
  - By default, the shell doesn't have permission to execute the script.

# Permissions

- Permissions are set with the `chmod` command
  - change file `mode` bits\*
    - You don't need to remember that
- Permissions are set by numbers
  - 775: Read, write, and execute permission
  - 700: Private just to you!
- `chmod <permission number> <file or directory>`
  - `chmod 775 myScript.sh`

# Setting Permissions on a Script

- Let's try to run our script

```
./myScript.sh
```

```
[sremon02@login001 ~]$ ./myScript.sh  
-bash: ./myScript.sh: Permission denied
```

- Bummer. Enter:

```
chmod 775 myScript.sh
```

```
[sremon02@login001 ~]$ ./myScript.sh  
-bash: ./myScript.sh: Permission denied  
[sremon02@login001 ~]$ chmod 775 myScript.sh
```

- Nothing prints – we didn't ask for anything to print.



# Try 2: Running the Script

- Use the **up arrow** twice to get your command back.

```
[sremon02@login001 ~]$ ./myScript.sh
```

- Hit enter to run it
- It should print your working directory!

```
[sremon02@login001 ~]$ ./myScript.sh  
/cluster/home/sremon02
```

# Fun Scripts

- You can put any commands in a script!
- New commands need to go on new lines
- Let's edit the script
  - `nano myScript.sh`
- Add the lines:
  - `ls`
  - `mkdir Test`
  - `ls`
- Save and quit nano

```
GNU nano 2.0.9      File: myScript.sh      Modified
pwd
ls
mkdir Test
ls
[ Read 1 line ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Tex ^T To Spell
```

# Re-Running a Script

- Use the **up arrow** twice to get your command back.

```
[sremon02@login001 ~]$ ./myScript.sh
```

- Hit **Enter** to run it
- Watch the magic!

```
[sremon02@login001 ~]$ ./myScript.sh
/cluster/home/sremon02
Birds  cookieRecipe.txt  myScript.sh  Winter Coats
Birds  cookieRecipe.txt  myScript.sh  Test  Winter Coats
[sremon02@login001 ~]$
```

# New Command: Echo

- `echo` 'Text you want'
  - Single quotes
- Prints (echoes) to the screen
- Try it!

```
[sremon02@login001 ~]$ echo 'Hello!'  
Hello!
```

- Not very useful by itself
- Very useful in scripts!

# Echo in Scripts

- Let's make a new script
  - `nano myEchoScript.sh`
- Fill it with
  - `echo 'We are in the directory'`
  - `pwd`
  - `echo 'These are the contents'`
  - `ls`

```
GNU nano 2.0.9          File: myEchoScript.sh          Modified
echo 'We are in the directory'
pwd
echo 'These are the contents'
ls
```

**^G** Get Help    **^O** WriteOut    **^R** Read File    **^Y** Prev Page    **^K** Cut Text    **^C** Cur Pos  
**^X** Exit        **^J** Justify     **^W** Where Is    **^V** Next Page    **^U** UnCut Text **^T** To Spell

- Save and quit nano

# Running a Script

- Let's try to run our script

```
./myEchoScript.sh
```

```
[sremon02@login001 ~]$ ./myEchoScript.sh  
-bash: ./myEchoScript.sh: Permission denied
```

- Permissions must be set for any new file! Enter:

```
chmod 775 myEchoScript.sh
```

```
[sremon02@login001 ~]$ chmod 775 myEchoScript.sh
```

- And run the script again.

```
[sremon02@login001 ~]$ ./myEchoScript.sh  
We are in the directory  
/cluster/home/sremon02  
These are the contents  
Birds myEchoScript.sh myScript.sh Test Winter Coats  
[sremon02@login001 ~]$
```

chnology

# Recap: Basic Scripts

- `.sh`:
  - Shell script file extension
  - `nano myScript.sh`
- Running a script:
  - `./<name of script>`
  - `./myScript.sh`
- Giving the shell permissions to run the script
  - `chmod <permission number> <file or directory>`
  - `chmod 775 myScript.sh`
- Printing to the screen
  - `echo 'Text you want'`

# VARIABLES



# Eliminating Repetition

- This is a lot to type repeatedly:
  - `echo 'Hi Donna'`
  - `echo 'It is great to see you'`
  - `echo 'Hi Sanjay'`
  - `echo 'It is great to see you'`
  - `echo 'Hi Marco'`
  - `echo 'It is great to see you'`
- What if we could write “`It is great to see you`” once and reuse it?
- A lot of programming is finding quick shortcuts to do things!

# Variables

- Let you hold values for reuse
  - Like a box
- Created with =
  - Can use any word you'd like
  - `greeting='It is great to see you'`
    - No spaces!
- Called with \$
  - `echo $greeting`
  - `echo 'Hi Sanjay'`
  - `echo $greeting`

```
[sremon02@login001 ~]$ greeting='It is great to see you'  
[sremon02@login001 ~]$ echo $greeting  
It is great to see you
```

# Reassigning Variables

- `greeting='It is great to see you'`
- `echo $greeting`
- `echo 'Greta'`
- `echo $greeting`
- `greeting='Thanks for coming'`
- `echo $greeting`

```
[sremon02@login001 ~]$ greeting='It is great to see you'  
[sremon02@login001 ~]$ echo $greeting  
It is great to see you  
[sremon02@login001 ~]$ echo 'Greta'  
Greta  
[sremon02@login001 ~]$ echo $greeting  
It is great to see you  
[sremon02@login001 ~]$ greeting='Thanks for coming'  
[sremon02@login001 ~]$ echo $greeting  
Thanks for coming
```

- *Tips: Variable names ('greeting') autocomplete with tab*
- *Don't forget that you can hit the up arrow to go to previous commands!*

# What Can a Variable Hold?

- Strings
  - `greeting='It is great to see you'`
  - `echo $greeting`
- Numbers
  - `greeting=12345`
  - `echo $greeting`

```
[sremon02@login001 ~]$ greeting='It is great to see you'  
[sremon02@login001 ~]$ echo $greeting  
It is great to see you  
[sremon02@login001 ~]$ greeting=12345  
[sremon02@login001 ~]$ echo $greeting  
12345
```

# Variables and Echo

- echo sees spaces as new arguments
- That works for variables, too!
  - `greeting='It is great to see you'`
  - `animal='Fox'`
  - `echo $greeting $animal`

```
[sremon02@login001 ~]$ greeting='It is great to see you'  
[sremon02@login001 ~]$ animal='Fox'  
[sremon02@login001 ~]$ echo $greeting $animal  
It is great to see you Fox  
[sremon02@login001 ~]$
```

# Variables and Scripts

- Just like any other command

```
[sremon02@login001 ~]$ nano variables.sh
```

```
GNU nano 2.0.9      File: variables.sh      Modified

greeting='It is great to see you'
echo $greeting

greeting=12345
echo $greeting

```

**^G** Get Help **^O** Write Out **^R** Read File **^Y** Prev Page **^K** Cut Text **^C** Cur Pos  
**^X** Exit **^J** Justify **^W** Where I **^V** Next Page **^U** UnCut Text **^T** To Spell

```
[sremon02@login001 ~]$ chmod 775 variables.sh
[sremon02@login001 ~]$ ./variables.sh
It is great to see you
12345
```

# Aside: Comments

- Added to a script for humans to read
  - Ignored by computers
  - Helps with organization and clarity
- Created with #

```
[sremon02@login001 ~]$ nano variables.sh
GNU nano 2.0.9 File: variables.sh

#This is a comment, the computer ignores it

greeting='It is great to see you'
echo $greeting

# Variable as a number

greeting=12345
echo $greeting

^G Get Hel^O WriteOu^R Read Fi^Y Prev Pa^K Cut Tex^C Cur Pos
^X Exit ^J Justify^W Where I^V Next Pa^U UnCut T^T To Spell
```

```
[sremon02@login001 ~]$ ./variables.sh
It is great to see you
12345
```

# Recap: Variables and Comments

- Variables
  - Save values for later reuse
  - Created with =
    - Can use any word you'd like
    - No spaces!
  - Called with \$
- Comments
  - Added to a script for humans to read
  - Ignored by computers
  - Created with #

```
GNU nano 2.0.9 File: variables.sh

#This is a comment, the computer ignores it

greeting='It is great to see you'
echo $greeting

# Variable as a number

greeting=12345
echo $greeting

^G Get Hel ^O WriteOu ^R Read Fi ^Y Prev Pa ^K Cut
^X Exit    ^J Justify ^W Where I ^V Next Pa ^U UnCu
```



# LOOPS

# Finding an Easier Way

- This is a lot to type repeatedly:
  - echo 1
  - echo 2
  - echo 3
  - echo 4
- What if we could write “echo” once and do it 4 times?
- We need a loop
  - For the numbers 1-4
  - echo that number

# For Loops

In the shell, loops look like this:

```
for <a variable> in <a list>  
do  
    <what the loop should do>  
done
```

```
for number in 1 2 3 4  
do  
    echo $number  
done
```

The line breaks are important!

```
[sremon02@login001 ~]$ for number in 1 2 3 4  
> do  
> echo $number  
> done  
1  
2  
3  
4  
[sremon02@login001 ~]$
```

# Aside: Carat & Ctrl-C

- in a shell means “You didn’t finish the command. Waiting for more input”

```
[sremon02@login001 ~]$ for number in 1 2 3 4  
> █
```

- If you didn’t mean to and want to exit the command:
  - **Ctrl-C**
  - Ctrl-C is good to know! It stops any input or runaway programs.

```
[sremon02@login001 ~]$ for number in 1 2 3 4  
> ^C  
[sremon02@login001 ~]$ █
```

# Loops and Scripts

```
[sremon02@login001 ~]$ nano loops.sh
```

These both work – the right one is better!

```
GNU nano 2.0.9 File: loops.sh
# Practicing for loops
for name in 'Bob' 'Alicia' 'Dom'
do
echo $name
echo 'Good morning'
done
^G Get Hel ^O WriteOu ^R Read Fi ^Y Prev Pa
^X Exit ^J Justify ^W Where I ^V Next Pa
```

```
GNU nano 2.0.9 File: loops.sh
# Practicing for loops
for name in 'Bob' 'Alicia' 'Dom'
do
    echo $name
    echo 'Good morning'
done
^G Get Hel ^O WriteOu ^R Read Fi ^Y Prev P
^X Exit ^J Justify ^W Where I ^V Next P
```

```
[sremon02@login001 ~]$ chmod 775 loops.sh
[sremon02@login001 ~]$ ./loops.sh
Bob
Good morning
Alicia
Good morning
Dom
Good morning
[sremon02@login001 ~]$
```



# Nested For Loops

- Anything can go in a loop – even another loop!
- But: Slows your program down

```
GNU nano 2.0.9      File: loops.sh
# Practicing for loops
greeting='Good morning'
for name in 'Bob' 'Alicia' 'Dom'
do
    echo $name
    for animal in 'fox' 'cat'
    do
        echo says $greeting $animal
    done
done
```

**^G** Get He**^O** Write**^R** Read F**^Y** Prev P**^K** Cut Te  
**^X** Exit **^J** Justif**^W** Where **^V** Next P**^U** UnCut

```
[sremon02@login001 ~]$ ./loops.sh
Bob
says Good morning fox
says Good morning cat
Alicia
says Good morning fox
says Good morning cat
Dom
says Good morning fox
says Good morning cat
[sremon02@login001 ~]$
```

# Recap: Loops

- Repeat a command for values in a list
- Can go in scripts
- Nested for loops
  - Combined for loops

```
for <variable> in <list>  
do  
    <what to do>  
done
```

```
GNU nano 2.0.9      File: loops.sh  
  
# Practicing for loops  
greeting='Good morning'  
  
for name in 'Bob' 'Alicia' 'Dom'  
do  
    echo $name  
    for animal in 'fox' 'cat'  
    do  
        echo says $greeting $animal  
    done  
done  
█  
  
^G Get He^O WriteO^R Read F^Y Prev P^K Cut Te  
^X Exit  ^J Justif^W Where ^V Next P^U UnCut
```



# Next Steps

- We didn't cover (in order of difficulty):
  - Flags: `ls -f`
  - Pipes: `|`
  - Passing arguments to scripts
  - `grep` (requires flags and pipes)
- We **strongly recommend** you look into these!
  - Important to know before Bioinformatics
  - Google Linux tutorials
  - Passing arguments to scripts can be found here
    - <https://www.lifewire.com/pass-arguments-to-bash-script-2200571>
  - `grep` can be found here:
    - <https://opensourceforu.com/2012/06/beginners-guide-gnu-grep-basics/>

# Help!

Most commands have a manual, if not, most of them have help:

- `man command` (q to quit)
  - Display command manual
- `man -k keyword`
  - Search all manuals based on keyword
- `command -h` or `--help`
  - Help info

...And Google helps!

# Tips & Tricks

- ❑ Control + A -- Jumps to the beginning of the line
- ❑ Control + E -- Jumps to the end of the line
- ❑ Clear long commands:
  - ❑ Ctrl+U --Clear up to the beginning
  - ❑ Ctrl+C – Cancel
  - ❑ Ctrl+E, then Ctrl+U
- ❑ Put your commonly used commands in scripts
- ❑ Change the permission of your scripts
- ❑ Create aliases for common commands
  - `alias heldweinlab="cd /cluster/tufts/heldweinlab/"`

# More Information + Contact RT-TTS

Wiki:

- [go.tufts.edu/cluster](https://go.tufts.edu/cluster)

Giant list of Linux commands:

- <https://linuxide.com/guide/linux-command-shelf.html>

Google it!

If you have any questions regarding to Tufts HPC cluster access or usage,

Please feel free to contact us at:

**[tts-research@tufts.edu](mailto:tts-research@tufts.edu)**

**MORE USEFUL COMMANDS!**

# Commands: Other Useful Notes

- ❑ Two versions of **options**: Short (-a) and/or Long (--all)
- ❑ Most of the options can **Mix and Match!**
- ❑ The wildcard “\*”
- ❑ Execute “./”
- ❑ Run commands from a file “source” or “.”

# System

To get information about your system:

Memory information:

- `free -g` or `-m` or `-h`
  - Display host memory spec in GB/MB/human readable

# System (Continued, 1)

To get information about your system:

Memory information:

- `free -g` or `-m` or `-h`
  - Display host memory spec in GB/MB/human readable
- `quota`
  - Display storage/memory quota



# System (Continued, 2)

To get information about your system:

Memory information:

- `free -g` or `-m` or `-h`
  - Display host memory spec in GB/MB/human readable
- `quota`
  - Display storage/memory quota
- **`showquota`**
  - Unique to Tufts HPC cluster
  - Display home directory quota and project space quota

# System (Continued, 3)

To get information about your system:

Memory information:

- `free -g` or `-m` or `-h`
  - Display host memory spec in GB/MB/human readable
- `quota`
  - Display storage/memory quota
- `showquota`
  - Unique to Tufts HPC cluster
  - Display home directory quota and project space quota
- `du -sh`
  - Display disc usage summary in human readable form

# System (Continued, 4)

To get information about your system:

System information:

- **hostname**
  - Display system hostname
- **hostname -i**
  - Display IP address of the host

# System (Continued, 5)

To get information about your system:

System information:

- `hostname`
  - Display system hostname
- `hostname -i`
  - Display IP address of the host
- `lscpu`
  - Display CPU information (compact)
- `lscpu -e=[list]`
  - Display selected CPU information

# System (Continued, 6)

To get information about your system:

System information:

- hostname
  - Display system hostname
- hostname -i
  - Display IP address of the host
- lscpu
  - Display CPU information (compact)
- lscpu -e=[list]
  - Display selected CPU information
- **cat /proc/cpuinfo**
  - Display CPU information  
(Individually)

# System (Continued, 7)

Try it out yourself!

# DIRECTORY AND FILES

# Directory & File

Navigate in the system, actions in directory level:

- `pwd`
  - Show current directory path



# Directory & File (Continued, 1)

Navigate in the system, actions in directory level:

- `pwd`
  - Show current directory path
- `cd directory_path`
  - Go the “directory\_path” location
- `cd ~` or `cd`
  - Go to home directory
- `cd ..`
  - Go up a level of the directory
- `cd -`
  - Go back to the last directory you were in

# Directory & File (Continued, 2)

Listing directory contents:

- `ls`
  - Display contents in the current directory

# Directory & File (Continued, 3)

Listing directory contents:

- `ls`
  - Display contents in the current directory
- `ls -a`
  - Display ALL contents in the current directory
- `ls -l`
  - Display long listing of the contents
- `ls -d */`
  - Display directory entries only
- `ls -r or -t or -X or -R or .etc`
  - Display in reverse order/sort by time/sort by extension/recursively

# Directory & File (Continued, 4)

Navigate in the system, editing files, working with directories, .etc.

- `mkdir dir_name`
  - Create a directory

# Directory & File (Continued, 5)

Navigate in the system, editing files, working with directories, .etc.

- `mkdir dir_name`
  - Create a directory
- `rm file_name`
  - Remove a file
- `rm -r` or `-rf dir_name`
  - Remove a directory with its contents
- `rmdir`
  - Remove an empty directory

# Directory & File (Continued, 6)

Navigate in the system, editing files, working with directories, .etc. The object that you work on:

- `cp file1 file2` or `dir`
  - Copy file1 to file2
- `cp -r dir1 dir2`
  - Copy dir1 to dir2 with contents

# Directory & File (Continued, 7)

Navigate in the system, editing files, working with directories, .etc. The object that you work on:

- `cp file1 file2 or dir`
  - Copy file1 to file2
- `cp -r dir1 dir2`
  - Copy dir1 to dir2 with contents
- `mv file1 file2`
  - Rename file1 as file2

# Directory & File (Continued, 8)

Navigate in the system, editing files, working with directories, .etc. The object that you work on:

- `cp file1 file2`
  - Copy file1 to file2
- `cp -r dir1 dir2`
  - Copy dir1 to dir2 with contents
- `mv file1 file2`
  - Rename file1 as file2
- `ln -s dir_path new_dir_path`
  - Create a link to dir\_path



# Directory & File (Continued, 9)

Try it out yourself!

# Directory & File (Continued, 10)

Work with text files:

- **cat file\_name**
  - Display file content
- **less file\_name**
  - Display a long text file per page at a time
- **more file\_name**
  - Display a long text file per page at a time

# Directory & File (Continued, 11)

Work with text files:

- `cat file_name`
  - Display file content
- `less file_name`
  - Display a long text file per page at a time
- `more file_name`
  - Display a long text file per page at a time
- `head file_name`
  - Display first 10 lines of a file
- `tail file_name`
  - Display last 10 lines of a file

# Directory & File (Continued, 12)

Work with text files:

- `cat file_name`
  - Display file content
- `less file_name`
  - Display a long text file per page at a time
- `more file_name`
  - Display a long text file per page at a time
- `head file_name`
  - Display first 10 lines of a file
- `tail file_name`
  - Display last 10 lines of a file
- `diff file1_name file2_name`
  - Difference between two files

# Directory & File (Continued, 13)

Searching for something? Look no further:

- **grep pattern file**
  - Search text for specific pattern
- **grep -i pattern file**
  - Search with case insensitive pattern
- **grep --color pattern file**
  - Highlight pattern in search result
- **grep -r pattern dir**
  - Search pattern recursively.
- **grep -c pattern file**
  - Count the matching case

# Directory & File (Continued, 14)

Searching for something? Look no further:

- **find dir\_name -name file\_name**
  - Find file “file\_name” in directory dir\_name
- **find dir\_name -atime n** or **-mmin n**
  - The file was last accessed more than n days or minutes ago
- **find dir\_name -iname file\_name**
  - Search for file\_name ignore case
- **find dir\_name -executable**
  - Search for executable files
- **which**
  - List the path for a command

# Directory & File (Continued, 15)

- **clear**
  - Clear the terminal screen

# Directory & File (Continued, 16)

- clear
  - Clear the terminal screen
- **Command 1 | command 2**
  - Use output of command 1 as the input of command 2



# Directory & File (Continued, 17)

- clear
  - Clear the terminal screen
- Command 1 | command 2
  - Use output of command 1 as the input of command 2
- **Command > file**
  - Save output to file
- **Command >> file**
  - Append output to file
- **Command 2> file**
  - Redirect error message to file
- **Command < file**
  - Use data from file as input of command

# Directory & File (Continued, 18)

Try it out yourself!

# Directory & File (Continued, 19)

Navigate in the system, editing files, working with directories, .etc. The object that you work on:

- `wget file_url`
  - Download file from web

# Directory & File (Continued, 20)

Navigate in the system, editing files, working with directories, .etc. The object that you work on:

- `wget file_url`
  - Download file from web
- `tar -cf` or `-xf file_list` or `file.tar`
  - Compress or Extract files
- `gzip (-l) file_name`
  - Compress file to .gz (with info)
- `gzip -d` or `gunzip file.gz`
  - Decompress file.gz
- `unzip file.zip`
  - Decompress file.zip

# EDITING TEXT FILES

# Directory & File

Try it out yourself!

# PERMISSIONS

# Permissions Overview

Changing permissions!

- **chmod permissions file\_name**

- permissions:

- u: user(owner) g: group o: others
- r: read (4) w: write (2) x: execute (1)
- No permission (0)

- **chmod u=rwx,g=rx,o=r file\_name**

- **chmod 754 file\_name**

- User have the right to read, write, and execute
- Group members have the right to read and execute
- Others have the right to read

- **chmod +x file\_name**

- Executable for owner and group



# Permissions Commands

Changing owner and group!

- **chown ownername file/dir**
  - Change the owner of a file or directory
  - Use **-R** for recursively change ownership under a directory
- **chown :groupname file/dir**
- **chgrp groupname file/dir**
  - Change the group of a file or directory
  - Use **-R** for recursively change group under a directory

# Permissions: Try It

Try it out yourself!