02/11/2018

OSU Software Carpentry Workshop

Unix Shell - MORNING LESSON BY NGG

Topics to cover:

1. Introduction to Shell
2. Navigating Directories
   Break
3. Working with Files
4. Pipes and Filters
   Break
5. Loops
6. Scripts

1. Introduction to Shell

Slides.

2. Navigating Directories

Start with slides, then move to the terminal in local machine.

THE MOUSE DOES NOT WORK IN THE SHELL!!! USE ONLY YOUR KEYBOARD AND ARROW KEYS FROM NOW ON!!!

```
[nathalia] $  hdjsfn
-bash: hdjsfn: command not found
```

We have to type commands that the computer recognizes...

```
[nathalia] $  pwd
/Users/grachetng  #HOME DIRECTORY.
```

Note: OUR FILESYSTEM IS DIFFERENT! What shows up on my screen will be different from what shows up on your screen, AND our paths to directories like the Desktop might be slight different too.

In this part of the lesson, we'll learn how to navigate the filesystem. We will find out our current directory, change directories, list contents of the directory, and make new directory.

Let's change directory to our Desktop...

But how do I know which path I should take to Desktop?

```
[nathalia] $  ls #LIST THE CONTENT OF YOUR CURRENT DIRECTORY
(...)

[nathalia] $  cd Desktop/ #MOST LIKELY YOUR DESKTOP DIRECTORY WILL BE INSI
DE YOUR HOME DIRECTORY

[nathalia] $  pwd #OUR CURRENT DIRECTORY CHANGED
/Users/grachetng/Desktop

[nathalia] $  ls
(...)
```

`cd` allows us to move anywhere in the filesystem hierarchy (up, down, from side-to-side) by providing a path.

Remember that `Final_draft.doc` file that was in my Desktop/PhD/Dissertation ???

```
[nathalia] $  cd PhD/Dissertation/ #GOING DOWN IN THE HIERARCHY

[nathalia] $  pwd
/Users/grachetng/Desktop/PhD/Dissertation
```

`cd` takes a path, or a shortcut

Point out: relative vs absolute paths

~ represents HOME directory
../ represents PARENT directory
/ in the **beginning** of a path represents the ROOT directory

IF I WANT TO GO BACK TO HOME, THERE ARE A FEW OPTIONS (USE ONLY ONE)

```
[nathalia] $  cd  #GOING UP IN THE HIERARCHY
[nathalia] $  pwd
/Users/grachetng

[nathalia] $  cd ~
[nathalia] $  pwd
/Users/grachetng

[nathalia] $  cd ../../.. #3 PARENT DIRECTORIES DOWN FROM HOME
[nathalia] $  pwd
/Users/grachetng

[nathalia] $  cd /Users/grachetng/
[nathalia] $  pwd
/Users/grachetng
```

I want to be in the Desktop.

```
[nathalia] $  cd Desktop/

[nathalia] $  pwd
/Users/grachetng/Desktop

[nathalia] $  ls
(...)
```

`ls` list current directory contents.

Point out about options or flags

Depending on how much files you have gathered in you Desktop, or in any directory, you might want to customize `ls` so it can be easier for you to find your files or directory. We can customize `ls` by using `flags` or `options`.

Notice that just typing `ls` I cannot recognize immediately what is a directory or a file.

```
[nathalia] $  ls
2-11-2018_OSU_Carpentry_Workshop    FigTree v1.4.3.dmg          Untitled.p
ages
Agronomy Journal            JOBS                      raxmlGUI_v1.5b1.dmg
BIOSTAR Handbook releases       PhD                   ~$Moller.docx
Colton_PLS              Python_for_Biologists_super_bundle  ~$PCRs turf.xl
sx
```

```
[nathalia] $  ls -F
2-11-2018_OSU_Carpentry_Workshop/   FigTree v1.4.3.dmg          Untitled.p
ages
Agronomy Journal/           JOBS/                     raxmlGUI_v1.5b1.dmg
BIOSTAR Handbook releases/      PhD/                      ~$Moller.docx
Colton_PLS/             Python_for_Biologists_super_bundle/ ~$PCRs turf.xl
sx
```

Point out that a / in the beginning of a name refers to the root directory, and a / in the end of a name refers to a directory

Point out the usage of flags (command, space, dash, option), and nesting

```
[nathalia] $  ls -lF
total 44192
drwxr-xr-x   6 grachetng  staff       192 Oct 31 12:15 2-11-2018_OSU_Carpe
ntry_Workshop/
drwxr-xr-x   5 grachetng  staff       160 Sep 25  2017 Agronomy Journal/
drwxr-xr-x   5 grachetng  staff       160 Jun  3 13:36 BIOSTAR Handbook re
leases/
drwxr-xr-x  12 grachetng  staff       384 Oct 30 14:33 Colton_PLS/
-rw-r--r--@  1 grachetng  staff  11787328 Jan 19  2018 FigTree v1.4.3.dmg
drwxr-xr-x   7 grachetng  staff       224 Oct 15 17:25 JOBS/
drwxr-xr-x  41 grachetng  staff      1312 Oct 31 09:00 PhD/
drwx------@ 10 grachetng  staff       320 Mar  8  2018 Python_for_Biologis
ts_super_bundle/
-rw-r--r--@  1 grachetng  staff    145724 Sep 18 16:48 Untitled.pages
-rw-r--r--@  1 grachetng  staff  10231390 Feb  8  2018 raxmlGUI_v1.5b1.dmg
-rw-r--r--@  1 grachetng  staff       162 Feb 28  2018 ~$Moller.docx
-rw-r--r--@  1 grachetng  staff       171 Jul 16  2015 ~$PCRs turf.xlsx

# NOW NOTICE THE CHANGES BETWEEN THESE OPTIONS

[nathalia] $  ls -lhrF
total 44192
-rw-r--r--@  1 grachetng  staff   171B Jul 16  2015 ~$PCRs turf.xlsx
-rw-r--r--@  1 grachetng  staff   162B Feb 28  2018 ~$Moller.docx
-rw-r--r--@  1 grachetng  staff   9.8M Feb  8  2018 raxmlGUI_v1.5b1.dmg
-rw-r--r--@  1 grachetng  staff   142K Sep 18 16:48 Untitled.pages
drwx------@ 10 grachetng  staff   320B Mar  8  2018 Python_for_Biologists_
super_bundle/
drwxr-xr-x  41 grachetng  staff   1.3K Oct 31 09:00 PhD/
drwxr-xr-x   7 grachetng  staff   224B Oct 15 17:25 JOBS/
-rw-r--r--@  1 grachetng  staff    11M Jan 19  2018 FigTree v1.4.3.dmg
drwxr-xr-x  12 grachetng  staff   384B Oct 30 14:33 Colton_PLS/
drwxr-xr-x   5 grachetng  staff   160B Jun  3 13:36 BIOSTAR Handbook relea
ses/
drwxr-xr-x   5 grachetng  staff   160B Sep 25  2017 Agronomy Journal/
drwxr-xr-x   6 grachetng  staff   192B Oct 31 12:15 2-11-2018_OSU_Carpentr
y_Workshop/
```

How do I know all the options I can use?

Let's check the manual pages of these commands! Use `man` to open up the manual page of a command.

```
[nathalia] $  man ls
```

- POP-UP WINDON WILL APPEAR WITH THE MANUAL PAGE
- IN A MANUAL PAGE YOU WILL FIND:
  NAME - THE NAME OF THE COMMAND
  SYNOPSIS - HOW TO USE THE COMMAND, FLAGS AND INPUT FILES
  DESCRIPTION - DESCRIPTION OF THE FLAGS AND OUTPUTS PRODUCED BY EACH ONE.
- SCROW UP-AND-DOWN (IN MAC THE MOUSE WORKS FOR SCROLLING... IN WINDOWS?!)
- NOTICE THE PROMPT IS GONE...

TELL ME what does `l`, `h`, `r`, and `F` mean.

TELL ME is `F` the same as `f`? #lowercase and uppercase letters are considered different characters for the computer.

All commands we will use today have a manual page. **I want to make you aware of the manual pages, and how to find information in it for your troubleshooting. I recommend exploring these man pages in your own time... before going to bed, morning read, when you are watching Netflix...**

PRESS `q` to exit the manual page.

You can also find help by the folllowing flag:

```
[nathalia] $  ls --help
ls: illegal option -- -
usage: ls [-ABCFGHLOPRSTUWabcdefghiklmnopqrstuwx1] [file ...]

[nathalia] $  cd --help
-bash: cd: --: invalid option
cd: usage: cd [-L|-P] [dir]
```

Also, Google has all the answers for your questions... It'll be become your best friend liking or not!

Make sure you are still on Desktop/

We will be working with real data set soon, so I want to make sure from now on we are working from the same relative path (this will avoid confusion with pathnames).

Let's create a new directory to keep our future data set.

Let's use `mkdir` command to make a new directory.

```
[nathalia] $  mkdir unix_workshop_2.11.2018
```

CAUTION ON NAMING directories and files:

- use a name that is informative and meaningful to you
- use a new name (avoid overwritting of files)
- use letters and numbers
- don't use spaces because spaces are used to break arguments
- don't start the name with a dash - because commands treat dash as flags
- use underscores _, period `.`, and dash `-` if to separate words within a name

```
[nathalia] $  cd unix_workshop_2.11.2018/

[nathalia] $  pwd
/Users/grachetng/Desktop/unix_workshop_2.11.2018
```

BREAK, MAYBE

3. Working with Files

BRIEFLY review topics covered before break.

Now, let's start working with files. **In this part, we'll learn how to create a file, copy, rename, move and delete files.**

```
[nathalia] $  pwd
/Users/grachetng/Desktop/unix_workshop_2.11.2018

#CHANGE DIR IF NECESSARY
```

Before we start working with our real data, we should practice a few common commands to create and manipulate files.

Let's create a new text file so we can practice these commands.

We'll use `nano` as out text editor.

```
[nathalia] $  nano test.txt

#POP-UP WINDOWN

#NOTICE:
EMPHASIZE THAT MOUSE DOES NOT WORK HERE! USE ARROW KEYS!
CURSOR IN THE MIDDLE OF THE WINDOWN
UPPER BAR - FILE NAME
LOWER BAR - OPTIONS TO EXIT, SAVE, ETC.

#TYPE:
I AM LEARNING UNIX COMMANDS TODAY. IT IS SOOOOO MUCH FUN!!!! I LOVE IT!!!

#SAVE AND EXIT,
control+x

#NOTICE THE MESSAGE IN THE LOWER BAR
press Y
press Return/Enter to save with the same input name

[nathalia] $  ls
test.txt
```

Check the contents of the file using `less` and `cat`.

```
[nathalia] $  cat test.txt
I AM LEARNING UNIX COMMANDS TODAY. IT IS SOOOOO MUCH FUN!!!! I LOVE IT!!!

[nathalia] $  less test.txt

#Pop-up window
#press q to exit
```

TELL ME What's the difference between them? When should you use one over the other?

If we are not satisfied with the text... let's go back and make changes!

```
[nathalia] $  nano test.txt

#MAKE A CHANGE:
I AM LEARNING UNIX COMMANDS TODAY. IT IS FUN!!!! I LIKE IT!!!

#SAVE AND EXIT
```

```
[nathalia] $  cat test.txt
I AM LEARNING UNIX COMMANDS TODAY. IT IS FUN!!!! I LIKE IT!!!
```

Let's make a copy of this file to keep as an original that we won't modify.

We'll use `cp` to copy a file, and this command requires two inputs.

`cp input1 input2`

input1 is the file we want to copy
input2 is the name of the copy

Must keep the order of these inputs.

```
[nathalia] $  cp test.txt test_original.txt

[nathalia] $  ls
test.txt        test_original.txt
```

Let's create a new directory to keep our original files in

```
[nathalia] $  ls -F
originals/      test.txt        test_original.txt
```

Let's move test_original into this new directory using `mv` command.

`mv` also requires two inputs:

`mv input1 input2`

input1 the file we want to move
input2 the path to where we want to move it into

```
[nathalia] $  mv test_original.txt originals/
```

To see if the file got moved, we can `cd` and `ls` into `originals/`, or simply use `ls` and provide the relative path to `originals/`.

So `ls` allows us to see the contents of directories that are not immediately in our filesystem hierarchy without changing our current directory.

```
[nathalia] $ ls #do not provide a path, lists contents of the current dire
ctory
originals   test.txt

[nathalia] $  ls originals/ #if provide a path, lists contents of the dire
ctory in the path
test_original.txt

[nathalia] $  pwd # we didn't change directories
/Users/grachetng/Desktop/unix_workshop_2.11.2018
```

`mv` is a flexible command. We can use to move or to rename files or directories.

How `mv` will perform these tasks depend on our input.

To move a file, we did:

`mv input1 input2`

input1 was the name of the file we want to move
input2 the path to where we want to move it into

Now, to rename a file, we'll do the same, but the input 2 will be the a new file name:

`mv input1 input2`

input1 is the name of the file we want to rename
input2 is the new file name

Again, must keep the order of the inputs.

Important to use a NEW name otherwise the computer will OVERWRITE the file and not even give you a warning message!!!

```
[nathalia] $  mv test.txt learning_unix.txt

[nathalia] $  ls
learning_unix.txt   originals
```

We can rename a file that is not in our current directory by proving a path to the file as input 1 and the same path but adding a new name in the end as input2.

```
[nathalia] $  mv originals/test_original.txt originals/learning_unix.txt

[nathalia] $  ls originals/
learning_unix.txt
```

IMPORTANT: IF YOU WANT TO RENAME A FILE THAT IS NOT IN YOUR CURRENT DIRECTORY, MAKE SURE YOU PROVIDE THE PATH in INPUT2... OTHERWISE WHAT WILL HAPPEN?!?!?!?!?!?!

```
[nathalia] $  mv originals/learning_unix.txt unix.txt
[nathalia] $  ls
learning_unix.txt   originals       unix.txt
[nathalia] $  ls originals/

#IT MOVES THE FILE TO THE CURRENT DIRECTORY WITH THE NAME YOU PUT IN INPUT
2
```

IMPORTANT: you can move a file and rename it at the same time. You provide the filename in input1, and the path and new file name in input2

```
[nathalia] $  mv unix.txt originals/learning_unix.txt

[nathalia] $  ls
learning_unix.txt   originals

[nathalia] $  ls originals/
learning_unix.txt
```

Emphasis on `mv` to move files, and `mv` to rename files before moving on to the next command

Now, we'll learn how to remove files and directories, but before I demonstrate how to do so... a word of caution!

DELETING FILES AND DIRECTORIES FROM THE SHELL IS FOREVER AND THERE AIN'T NO GOING BACK!!! THERE IS NOT A 'TRASH' IN THE SHELL LIKE OUR COMPUTER HAS TO RECOVER DELETED FILES.

BE CAREFUL!! DON'T GO TO DOWN DELETING THINGS THAT YOU ARE NOT USING NOW, BECAUSE IF YOU NEED THEM LATER THERE IS NO WAY TO RECOVER.

To remove a file, use `rm`

```
[nathalia] $  rm learning_unix.txt

#if you don't believe me, go check your trash folder to see if the file is
there...

# you can use the flag -i to request a confirmation before deleting...

it would look like this: [nathalia] $  rm -i learning_unix.txt

[nathalia] $  ls
originals
```

To remove a directory and its contents, use `rm` with `-r` flag

```
[nathalia] $  rm -r originals/

[nathalia] $  ls
[nathalia] $

# # you also can use the flag -i to request a confirmation before deleting
...
```

So far, we learned a few basic file and directory manipulations so far.

Now we'll learn more useful and powerful commands that work on data sets. To demonstrate these commands, it will be much clearler to use a real data set. Let's download the data from the command line using `wget` and the URL to the file.

Copy the URL in ETHERPAD

When you press Return/Enter, things will be printed to the screen...

```
[nathalia] $  wget https://swcarpentry.github.io/shell-novice/data/data-sh
ell.zip
--2018-10-31 14:36:05--  https://swcarpentry.github.io/shell-novice/data/d
ata-shell.zip
Resolving swcarpentry.github.io (swcarpentry.github.io)... 185.199.111.153
, 185.199.110.153, 185.199.108.153, ...
Connecting to swcarpentry.github.io (swcarpentry.github.io)|185.199.111.15
3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 579150 (566K) [application/zip]
Saving to: ‘data-shell.zip’

data-shell.zip                     100%[==================================
===========================>] 565.58K   600KB/s    in 0.9s

2018-10-31 14:36:07 (600 KB/s) - ‘data-shell.zip’ saved [579150/579150]
```

Notice the `.zip` extension of the file, which means the file is compressed, and we need to decompress it before anything.

```
[nathalia] $  tar -xf data-shell.zip

[nathalia] $  ls
data-shell  data-shell.zip

#GUNZIP didn't work.
```

Explain the data set

This is a data set from Nelle Nemo, who is a marine biologist. She just returned from a 6-month expedition to the North Pacific Gyre where she did a survey of the gelatinous marina life in the Great Pacific Garbage Patch.

GROUP EXERCISE

Give 2 mins for letting the group explore the filesystem on their own.

```
[nathalia] $  ls -R #this will help you explore the filesystem at once
(...)
```

Let me give you more info about the data and what we'll focus for the next part of the lesson this morning.

Nelle collected 1520 samples in the Great Pacific Garbage Patch. She returned to land, and processed the samples in the laboratory. She ran each sample through an protein assay machine, which will take about 2 weeks to finish... So far, she has 17 text files with the relative abundance of 300 proteins in each one.

The files are in `data-shell/north-pacific-gyre/2012-07-03`.

What she needs to do is:

- to finish processing the samples.
- In the meantime, perform a preliminary statistical analysis on these 17 files using the script `goostats`, which her advisor wrote.

Maybe you are thinking... the analysis might take what 1 min to complete, entering a command and pressing enter 17 times, checking instagram in between... Nahhh that isn't too bad... but then...

- Write a statisticl report to her advisor with ALL samples within 2 weeks because after that her advisor is going on sabbatical leave.

Now... it's entering command pressing enter and waiting 1 min 1520 times... No thanks!!

Now we'll help Nelle create a pipeline to:

- **A) check her files prior to analysis**
- **B) run commands to perform the statistical analysis over multiple files in one task**
- **C) write a script that will automate the statistical analysis on all 1520 files after her assay finishes.**

4. Pipes and Filters

So, first of all we need to make sure each file produced by the machine has 300 entries. We'll use pipe and filters to do so.

```
[nathalia] $  cd data-shell/north-pacific-gyre/2012-07-03/

[nathalia] $  ls
NENE01729A.txt  NENE01751A.txt  NENE01843A.txt  NENE01978A.txt  NENE02040A
.txt  NENE02043A.txt  goostats
NENE01729B.txt  NENE01751B.txt  NENE01843B.txt  NENE01978B.txt  NENE02040B
.txt  NENE02043B.txt
NENE01736A.txt  NENE01812A.txt  NENE01971Z.txt  NENE02018B.txt  NENE02040Z
.txt  goodiff

[nathalia] $  less NENE01729A.txt #focus on the first file.
# quick look to see what's inside...
```

Use the command `wc` to display the number of lines, words, and bytes contained in each input file:

```
[nathalia] $  wc NENE01729A.txt
     300     300    4406 NENE01729A.txt

   # lines   words   bytes
```

We are only interested in number of lines, so let's pass `-l` to only display number of lines

```
[nathalia] $  wc -l NENE01729A.txt
     300 NENE01729A.txt
```

We must check all 17 files... doing this over 16 more times, does not sound too bad.. but remember that later she'll need to do it 1519 times... so let's do this in one line of commands.

Notice that all files produced by the machine have a pattern:

- start with NENE
- followed by numbers and one letter
- end with `.txt` file extension

We can take advantage of that by using a wildcard `*` which is a feature of the shell.

The wildcard `*` matches zero or more characters. When the shell sees a wildcard, it expands the wildcard to create a list of matching filenames before running the command that was asked for.

```
[nathalia] $  wc -l NENE*
     300 NENE01729A.txt
     300 NENE01729B.txt
     300 NENE01736A.txt
     300 NENE01751A.txt
     300 NENE01751B.txt
     300 NENE01812A.txt
     300 NENE01843A.txt
     300 NENE01843B.txt
     300 NENE01971Z.txt
     300 NENE01978A.txt
     300 NENE01978B.txt
     240 NENE02018B.txt
     300 NENE02040A.txt
     300 NENE02040B.txt
     300 NENE02040Z.txt
     300 NENE02043A.txt
     300 NENE02043B.txt
    5040 total

# now we have the total number of lines of each file.

Notice one file has 240 lines, perhaps should double check what happened d
uring the processing of that sample...
```

We'll now create a pipe to sort the output of line numbers. The command to sort is `sort`, and we'll pass the flag `-n` that will sort the output numerically by arithmetic value.

A pipe is characterized by a vertical bar | between two commands.

A pipe tells the shell to get the output of the command on the left and use as input to the command on the right.

```
#TIP!!!! USE THE UP-ARROW KEY TO RETRIEVE THE HISTORY OF COMMANDS

[nathalia] $  wc -l NENE* | sort -n
     240 NENE02018B.txt
     300 NENE01729A.txt
     300 NENE01729B.txt
     300 NENE01736A.txt
     300 NENE01751A.txt
     300 NENE01751B.txt
     300 NENE01812A.txt
     300 NENE01843A.txt
     300 NENE01843B.txt
     300 NENE01971Z.txt
     300 NENE01978A.txt
     300 NENE01978B.txt
     300 NENE02040A.txt
     300 NENE02040B.txt
     300 NENE02040Z.txt
     300 NENE02043A.txt
     300 NENE02043B.txt
    5040 total

# NOTICE THAT NOW THE OUTPUT IS SORTED!! YEY!
```

It's important to Nelle to keep a record of these information we are retrieving...

Let me explain to you something the computer does here in very simple words...

`wc` is a command
`-l` is a flag passed to wc
`NENE*` is a standard input or stdin that is being passed to `wc -l`

Everytime you type a command and you add a filename or a path, you are passing a standard input to that command.

Remember that REPL principle of the shell? The computer will process your task and give you an output to the screen. When an output is being generated on your screen, that is called standard output or stdout.

Standard output is not stored anywhere, the output is just printed to the screen. Because Nelle needs to keep a record of the steps she is taking and the results we are retrieving for later, we have to save the output of this command.

To save standard output we have to REDIRECT standard output into a text file. It's simple, we'll type the same command and add > and give it a new filename. The computer will create a file with the name we provide, and write the output to the file. Nothing will be printed to the screen, but the contents of the file will be saved for future reference.

```
[nathalia] $  wc -l NENE* | sort -n > line_numbers_02-11-2018.txt

[nathalia] $  less line_numbers_02-11-2018.txt
```

Nelle also told me that she collected samples in two locations, and that in the file names these locations are A and B.

Then I was like... *Nelle, some files have a Z on their name...* And then she explained that by convention in her lab, 'Z' indicates samples with missing information. Another student took a depth of each sample, but for some reasone depth wasn't recorded for these samples, that's why they have a 'Z' in their name. She also said these samples should be excluded from the analysis...

what do we do? DELETE THE FILES?

Oh gosh, no... neve delete stuff specially if it's data collected...

Let's modify own pipe to only look at files that have A or B in their name.

```
Remember to use the up-arrow key!!!!!!!!!!!!!!!!!!!!!

[nathalia] $  wc -l NENE*[AB].txt | sort -n # don't redirect stdout just n
ow, let's see if the pipe will work
     240 NENE02018B.txt
     300 NENE01729A.txt
     300 NENE01729B.txt
     300 NENE01736A.txt
     300 NENE01751A.txt
     300 NENE01751B.txt
     300 NENE01812A.txt
     300 NENE01843A.txt
     300 NENE01843B.txt
     300 NENE01978A.txt
     300 NENE01978B.txt
     300 NENE02040A.txt
     300 NENE02040B.txt
     300 NENE02043A.txt
     300 NENE02043B.txt
    4440 total

# now, redirect stdout to overwrite that file we created

[nathalia] $  wc -l NENE*[AB].txt | sort -n > line_numbers_02-11-2018.txt

#inspect with less...
```

With more files being added, Nelle can run this pipe, save the stdout and change the dates in the file name to keep a record of the line numbers of each file overtime...

With more files being added, inspecting might be a challenge...

Let's do a quick pipe inspection

```
[nathalia] $  cat line_numbers_02-11-2018.txt | head -n 5 | less

# cat
# head prints the first lines in a file, with -n flag prints the number pr
ovided
# less opens that window with the first 5 lines

#tail is the opposite of head... test it out!
```

BREAK

5. Loops

Now we'll help Nelle write command that will perform the stastistical analysis on her files. We'll build a `for loop` for that.

What's a for loop? A loop is an iteration statement that will be repeatedly executed. Execute a command repetitively. It helps in the automation of tasks, reduces amount of typing

In this case, we'll use a for loop to run the statistical analysis on each file, one at a time. But we'll write one block of code that will run the analysis on all 15 files.

This is a basic for loop syntax:

```
for VARIABLE in SOMEWHERE;
> do command1;
> command2;
> commandN;
> done
```

or,

```
$ for VARIABLE in SOMEWHERE; do command1; command2; commandN; done
```

VARIABLE is an arbitrary name that you choose.
SOMEWHERE can be a file or a directory.
INDENTATION.

> prompt change = indicates that shell knows there are more commands to be entered

IMPORTANT: What is a ‘VARIABLE’?*

A variable is simply a box, which you create, to place values into it. A more technical definition is: a character string that you assign a value. The value could be text, number, filename, path, etc.

You can assing more than one type of value to a variable.

Don’t use !, * or - in variable names because these characters have special meaning for unix...

You call a variable you defined by using `$` in front of the variable name.

This is how you define a variable:

```
$ variable_name=variable_value
$ words="one two three"
$ echo $words
one two three
$ words="flower sun moon and me"
$ echo $words
flower sun moon and me
```

The syntax of a for loop in plain English:

```
$ for variable in collection; do things with variable; done
```

A for loop in action:

```
$ for character in $words; do echo $character; done
flower
sun
moon
and
me

$ for char in $words; do echo $char; done
flower
sun
moon
and
me



[nathalia] $  for file in NENE*[AB].txt;
```

```
> do echo $file;
> done
NENE01729A.txt
NENE01729B.txt
NENE01736A.txt
NENE01751A.txt
NENE01751B.txt
NENE01812A.txt
NENE01843A.txt
NENE01843B.txt
NENE01978A.txt
NENE01978B.txt
NENE02018B.txt
NENE02040A.txt
NENE02040B.txt
NENE02043A.txt
NENE02043B.txt
```

A loop does many tasks with one command block... or does many mistakes if the block contains typos... A good way to check is by using `echo` commands.

So, this part of the block `do echo $file stats-$file` will be the statistics, we'll remove echo and insert a command to run goostast.

DRY RUN Run the goostat script on one file before running it in the loop.

How will we ran goostats?! `goostats` is a shell script, and we'll run it in this way:

```
[nathalia] $  bash goostats
goodstats file1 file2
call goostats with two arguments
```

The message is the usage of the script.

We have to add two arguments:

- `file1` is the file we have
- `file2 is the name of the output or results of the statistical analysis.

Now, let's try:

```
[nathalia] $  bash goostats NENE01729A.txt stats-NENE01729A.txt

[nathalia] $  ls
NENE01729A.txt         NENE01843A.txt         NENE02040A.txt         go
ostats
NENE01729B.txt         NENE01843B.txt         NENE02040B.txt         li
ne_numbers_02-11-2018.txt
NENE01736A.txt         NENE01971Z.txt         NENE02040Z.txt         st
ats-NENE01729A.txt
NENE01751A.txt         NENE01978A.txt         NENE02043A.txt
NENE01751B.txt         NENE01978B.txt         NENE02043B.txt
NENE01812A.txt         NENE02018B.txt         goodiff

[nathalia] $  less stats-NENE01729A.txt
```

Cool! Noticed that the computer worked for a few seconds, and then returned the prompt?!

So let's do a Dry Run again adding the file2 to our for loop, like so:

```
[nathalia] $  for file in NENE*[AB].txt;  do echo $file stats-$file;done
NENE01729A.txt stats-NENE01729A.txt
NENE01729B.txt stats-NENE01729B.txt
NENE01736A.txt stats-NENE01736A.txt
NENE01751A.txt stats-NENE01751A.txt
NENE01751B.txt stats-NENE01751B.txt
NENE01812A.txt stats-NENE01812A.txt
NENE01843A.txt stats-NENE01843A.txt
NENE01843B.txt stats-NENE01843B.txt
NENE01978A.txt stats-NENE01978A.txt
NENE01978B.txt stats-NENE01978B.txt
NENE02018B.txt stats-NENE02018B.txt
NENE02040A.txt stats-NENE02040A.txt
NENE02040B.txt stats-NENE02040B.txt
NENE02043A.txt stats-NENE02043A.txt
NENE02043B.txt stats-NENE02043B.txt
```

Now, we are ready to build the complete for loop:

```
[nathalia] $  for file in NENE*[AB].txt;  do bash goostats $file stats-$fi
le; done
```

Where is the prompt?! Is it analysing anything?! Yes, it is...

-press control+c to kill the command.

```
^C
[nathalia] $  ls
NENE01729A.txt        NENE01843B.txt        NENE02040Z.txt          st
ats-NENE01729B.txt
NENE01729B.txt        NENE01971Z.txt        NENE02043A.txt          st
ats-NENE01736A.txt
NENE01736A.txt        NENE01978A.txt        NENE02043B.txt          st
ats-NENE01751A.txt
NENE01751A.txt        NENE01978B.txt        goodiff             stats-
NENE01751B.txt
NENE01751B.txt        NENE02018B.txt        goostats
NENE01812A.txt        NENE02040A.txt        line_numbers_02-11-2018.tx
t
NENE01843A.txt        NENE02040B.txt        stats-NENE01729A.txt
```

Yeah, so it is working. But it would be nice to which file it is analyzing instead of having nothing being printed to the screen... We can easily add that to the script with `echo` commands:

```
[nathalia] $  for file in NENE*[AB].txt;  do echo "Analyzing $file"; bash
goostats $file stats-$file; echo "Wrote stats-$file" ;done
Analyzing NENE01729A.txt
Wrote stats-NENE01729A.txt
Analyzing NENE01729B.txt
Wrote stats-NENE01729B.txt
Analyzing NENE01736A.txt
Wrote stats-NENE01736A.txt
Analyzing NENE01751A.txt
Wrote stats-NENE01751A.txt
Analyzing NENE01751B.txt
Wrote stats-NENE01751B.txt
Analyzing NENE01812A.txt
Wrote stats-NENE01812A.txt
Analyzing NENE01843A.txt
Wrote stats-NENE01843A.txt
Analyzing NENE01843B.txt
Wrote stats-NENE01843B.txt
Analyzing NENE01978A.txt
Wrote stats-NENE01978A.txt
Analyzing NENE01978B.txt
Wrote stats-NENE01978B.txt
Analyzing NENE02018B.txt
Wrote stats-NENE02018B.txt
Analyzing NENE02040A.txt
```

```
Wrote stats-NENE02040A.txt
Analyzing NENE02040B.txt
Wrote stats-NENE02040B.txt
Analyzing NENE02043A.txt
Wrote stats-NENE02043A.txt
Analyzing NENE02043B.txt
Wrote stats-NENE02043B.txt

[nathalia] $  ls
NENE01729A.txt         NENE01978A.txt          goostats            stats-
NENE01843B.txt
NENE01729B.txt         NENE01978B.txt          line_numbers_02-11-2018.tx
t stats-NENE01978A.txt
NENE01736A.txt         NENE02018B.txt          stats-NENE01729A.txt
stats-NENE01978B.txt
NENE01751A.txt         NENE02040A.txt          stats-NENE01729B.txt
stats-NENE02018B.txt
NENE01751B.txt         NENE02040B.txt          stats-NENE01736A.txt
stats-NENE02040A.txt
NENE01812A.txt         NENE02040Z.txt          stats-NENE01751A.txt
stats-NENE02040B.txt
NENE01843A.txt         NENE02043A.txt          stats-NENE01751B.txt
stats-NENE02043A.txt
NENE01843B.txt         NENE02043B.txt          stats-NENE01812A.txt
stats-NENE02043B.txt
NENE01971Z.txt         goodiff             stats-NENE01843A.txt
```

6. Shell scripts

Now, that we built our code and we know it's running and producing the statistics that we want, we can write our code into a shell script.

Advantages of a script: make analysis more reproducible, easier to re-analyze later.

```
[nathalia] $  nano do-stats.sh
```

```
[nathalia] $  cat do-stats.sh
for file in NENE*[AB].txt;  do echo "Analyzing $file"; bash goostats $file
stats-$file; echo "Wrote stats-$file"; done
```

```
[nathalia] $  bash do-stats.sh
Analyzing NENE01729A.txt
Wrote stats-NENE01729A.txt
Analyzing NENE01729B.txt
Wrote stats-NENE01729B.txt
Analyzing NENE01736A.txt
Wrote stats-NENE01736A.txt
Analyzing NENE01751A.txt
Wrote stats-NENE01751A.txt
Analyzing NENE01751B.txt
Wrote stats-NENE01751B.txt
Analyzing NENE01812A.txt
Wrote stats-NENE01812A.txt
Analyzing NENE01843A.txt
Wrote stats-NENE01843A.txt
Analyzing NENE01843B.txt
Wrote stats-NENE01843B.txt
Analyzing NENE01978A.txt
Wrote stats-NENE01978A.txt
Analyzing NENE01978B.txt
Wrote stats-NENE01978B.txt
Analyzing NENE02018B.txt
Wrote stats-NENE02018B.txt
Analyzing NENE02040A.txt
Wrote stats-NENE02040A.txt
Analyzing NENE02040B.txt
Wrote stats-NENE02040B.txt
Analyzing NENE02043A.txt
Wrote stats-NENE02043A.txt
Analyzing NENE02043B.txt
Wrote stats-NENE02043B.txt
```

GREAT!!!

Advantages: the scripts will take the right files = avoiding the ones that have Z in their name

But there is a disadvantage: if she decides to analyze other data that do not follow the name pattern, the script won't work. We can make an adaptation to work with any file pattern we want.

`"$@"` is a special variable which means **All of the command-line arguments to the shell script**

```
[nathalia] $  nano do-stats.sh

[nathalia] $  cat do-stats.sh
for file in "$@";  do echo "Analyzing $file"; bash goostats $file stats-$f
ile; echo "Wrote stats-$file"; done

[nathalia] $  bash do-stats.sh NENE*[AB].txt
Analyzing NENE01729A.txt
Wrote stats-NENE01729A.txt
Analyzing NENE01729B.txt
Wrote stats-NENE01729B.txt
Analyzing NENE01736A.txt
Wrote stats-NENE01736A.txt
Analyzing NENE01751A.txt
Wrote stats-NENE01751A.txt
Analyzing NENE01751B.txt
Wrote stats-NENE01751B.txt
Analyzing NENE01812A.txt
Wrote stats-NENE01812A.txt
Analyzing NENE01843A.txt
Wrote stats-NENE01843A.txt
Analyzing NENE01843B.txt
Wrote stats-NENE01843B.txt
Analyzing NENE01978A.txt
Wrote stats-NENE01978A.txt
Analyzing NENE01978B.txt
Wrote stats-NENE01978B.txt
Analyzing NENE02018B.txt
Wrote stats-NENE02018B.txt
Analyzing NENE02040A.txt
Wrote stats-NENE02040A.txt
Analyzing NENE02040B.txt
Wrote stats-NENE02040B.txt
Analyzing NENE02043A.txt
Wrote stats-NENE02043A.txt
Analyzing NENE02043B.txt
Wrote stats-NENE02043B.txt
```

Now, let's add a usage information about the script:

```
[nathalia] $  nano do-stats.sh

#Script to calculate stats for protein data
```

```
#For Nelle

if [[ $# -eq 0 ]]
  then
    echo "usage: $ bash do-stats [pattern]"
    echo "[pattern] e.g. NENE*[AB].txt"
    exit 1
fi

for file in "$@";  do echo "Analyzing $file"; bash goostats $file stats-$f
ile; echo "Wrote stats-$file"; done


[nathalia] $  bash do-stats.sh
usage: $ bash do-stats [pattern]
[pattern] e.g. NENE*[AB].txt

[nathalia] $  bash do-stats.sh NENE*[AB].txt
Analyzing NENE01729A.txt
Wrote stats-NENE01729A.txt
Analyzing NENE01729B.txt
Wrote stats-NENE01729B.txt
Analyzing NENE01736A.txt
Wrote stats-NENE01736A.txt
Analyzing NENE01751A.txt
Wrote stats-NENE01751A.txt
Analyzing NENE01751B.txt
Wrote stats-NENE01751B.txt
Analyzing NENE01812A.txt
Wrote stats-NENE01812A.txt
Analyzing NENE01843A.txt
Wrote stats-NENE01843A.txt
Analyzing NENE01843B.txt
Wrote stats-NENE01843B.txt
Analyzing NENE01978A.txt
Wrote stats-NENE01978A.txt
Analyzing NENE01978B.txt
Wrote stats-NENE01978B.txt
Analyzing NENE02018B.txt
Wrote stats-NENE02018B.txt
Analyzing NENE02040A.txt
Wrote stats-NENE02040A.txt
```

```
Analyzing NENE02040B.txt
Wrote stats-NENE02040B.txt
Analyzing NENE02043A.txt
Wrote stats-NENE02043A.txt
Analyzing NENE02043B.txt
Wrote stats-NENE02043B.txt
```

Go back to slides and summarize lesson!