EEB 177 Lecture 6

Michael Alfaro

EEB 177 Lecture 6

Topics

- Permissions
- Scripts
- Python

Office Hours

Tuesday 1:00-2:00

Terasaki 2149

Preliminaries

- Start **nano**: `$ nano` and save the file "classwork-Tuesday-1-28.txt" to your class-assignments directory
- push this to your remote repository
- you can write answers to today's exercises in this file.

Writing a shell script

Lets illustrate some ideas about paths, scripts, and permissions by writing a simple shell script. You are going to write a program in your text editor that will execute a series of shell commands that you have already learned.

open up nano and type the following lines:

open `nano` in your `class-assignments` folder and save the following file as dir.sh:

```
#! /bin/bash
ls -la
echo "Above are the directory listings for this folder"
pwd
echo "right now it is :"
date
```

save this file as **dir.sh**

Paths

there are two standard locations for programs– `/usr/bin` and `/bin`

use `ls` to see what is in them

The shell searches these directories (and others that have been addded to the path) whenever you type a command.

Type `echo $PATH` to see your current path.

`which` will tell you the directory to a command. Try `which cat`

Creating a scripts directory and adding it to the path

we want a single working copy of each program on our machines so we need to make sure the shell searches for our programs. . . .

- go to your home directory
- create a directory called `scripts`
- to add the scripts directory to the path, open the `.profile` file in nano
- add this line (exactly) `export PATH="$PATH:$HOME/scripts"`
- exit and save

Now we have created a program we would like to run and created a path to the scripts directory. What else do we need to do?

hint: where is `dir.sh` right now?

hint: what permissions do we need to execute a script?

the shebang (#!)

`#!` is called the shebang–it means that all following contents of script will be sent to the program following the shebang

`#! /bin/bash` sends it all to bash

remember,new scripts are not executable w/o changing permissions

checking permissions

- cd ~/scripts
- check permission with ls -l
- add permission to execute with chmod u+x

try running your program from different directories. Does it work? Why?

exercise

Add your scripts directory to your remote repository. You will need to

- `git init` in your scripts directory
- add your script
- commit your script
- create a remote repo on github
- copy and paste the command lines from the remote repo after you create it.

exercise

Lets make a shell script of the body mass extraction exercise we did Tuesday

go to your scripts directory and create this file

```
touch ExtractBodyM.sh
```

edit your script

Use nano to open and edit your script

```
nano ExtractBodyM.sh &
```

the “&” character opens the script in the background so yu can keep using the terminal

Now add the pipeline to the script

```
tail -n +2 ../data/Pacifici2013_data.csv | cut -d ';'
-f 2-6 | tr -s ';' ' ' | sort -r -n -k 6 > BodyM.csv
```

Comments

It is a good idea to add comments to your script so that you know what the purpose of the code is when you return to it. Use the # character for comments.

Take a csv file delimited by ';' # Remove the header

Make space separated

Sort according to the 6th (numeric) column in descending order

Redirect to a file

```
tail -n +2 ../data/Pacifici2013_data.csv | cut -d ';'
-f 2-6 | tr -s ';' ' ' | sort -r -n -k 6 > BodyM.csv
```

Running the script

You can run the script using the bash command

```
bash ExtractBodyM.sh
```

How could you make this script run automatically from any prompt without typing **bash**?

solution

Add the shebang line and place the script in your **scripts** directory

#! /bin/bash

Take a csv file delimited by ';' # Remove the header

Make space separated

Sort according to the 6th (numeric) column in descending order

Redirect to a file

```
tail -n +2 ../data/Pacifici2013_data.csv | cut -d ';'
-f 2-6 | tr -s ';' ' ' | sort -r -n -k 6 > BodyM.csv
```

you will need to change permissions to execute the script!

Programming languages

- There are over 2000
- There is no perfect language for all tasks
- You are already learning several: regular expressions, python, R
- This class does not cover fast, compiled languages like C.

dynamic typing

Programming languages like `C` and `Fortran` are statically typed, meaning that you need to define the type of a variable when you create it. Python does not require this and automatically determines type. You can see the type of a variable with the `type` function.

```
In [22]: xx = 2

In [23]: type(xx)
Out[23]: int

In [24]: xx = "two"

In [25]: type  (xx)
Out[25]: str
```

strings

Python is an excellent language for bioinformatics in part because it provides many built-in functions for manipulating strings. You can see these methods by typing the name of a string followed by a

```
 In [1]: astring = "ATGCATG"
# return the length of the string
In [2]: len(astring)
Out[2]: 7
```

You can also use string functions by creating the string on the fly with quotation marks and calling method from the new string.

```
# make upper case
In [12]: "atgc".upper()
Out[12]: 'ATGC'
# make lower case
In[13]: "TGCA".lower()
Out[13]: 'tgca'
```

concatenating strings with + and `join`

```
In [14]: genus = "Rattus"
In [14]: species =
"norvegicus"
In [16]: genus + " " + species
Out[16]: 'Rattus norvegicus'
```

String challenge

Do the following

1. Initialize the string s = "WHEN on board H.M.S. Beagle, as naturalist".
2. Apply a string method to count the number of occurrences of the character b.
3. Modify the command such that it counts both lower and upper case bs.
4. Replace WHEN with When.

Collections

Python has variables that are **collections** of other objects. **lists** are collections of ordered values and are defined by `[]`.

```
# Anything starting with # is a comment
In [26]: MyList = [3,2.44,"green",True]
In [27]: MyList[1]
Out[27]: 2.44
In [28]: MyList[0] # NOTE: FIRST ELEMENT -> 0
```