Monday, May 14, 2018

Shell Scripts

Today's topics: random shell concepts left out of previous lectures, collecting shell commands into a **script**

These topics are mostly optional — very useful things to know if you're going to continue writing scripts and programs — but worth looking at now because they provide background for our next set of Python projects

Reading: *PCfB* Ch 6 ("Scripting with the Shell")

- environment variables
- $PATH
- permissions
- conditional execution
- loops

Environment Variables

Each time you open a terminal window the system starts a new shell

The shell uses a set of **environment variables** to define various user settings

A command to know: **`printenv`** (“print environment”)

- it prints a complete list of variables and their current values
- most of them are very obscure

Here are some that were printed on my desktop system:

```
$ printenv
SHELL=/bin/bash
TERM=xterm-256color
USER=conery
PWD=/Users/conery/Classes/410/units/6_scripts/
lectures
LANG=en_US.UTF-8
HOME=/Users/conery
```

echo

A shell command that doesn't seem like it would be very helpful: **echo**

- all it does is print its command line argument

```
$ echo hello
hello
$ echo 'stop repeating what I say'
stop repeating what I say
```

It is useful, however, if you want it to print the value of an environment variable

```
$ echo $HOME
/Users/conery
$ echo $LANG
en_US.UTF-8
```

Important:

if X *is an environment variable, use* $X *to refer to its value*

$PATH

A very important variable is $PATH

- it is a series of file paths, separated by colons
- tells the shell where to look to find executable files

Example (again from my desktop system, edited to show one directory per line):

```
$ echo $PATH
/Users/conery/miniconda3/bin:
/Users/conery/SysAdmin/Scripts/python:
/Users/conery/SysAdmin/Scripts/sh:
/Users/conery/Applications/Bioinformatics:
/Users/conery/Applications/bin:
/usr/local/bin:
/usr/bin:
/bin:
/usr/sbin:
/sbin
```

miniconda was added to my path by the conda installer

I set up my .bash_profile to include folders where I keep my own programs and scripts

/usr, /usr/local, etc are standard locations in a Unix system

The Shell Uses $PATH to Find Executables

When you type a shell command

- the shell uses the first word as the name of the program you want to run
- it searches the directories in your execution path to find an executable file with that name
- it launches the first program it finds

If no file is found it prints an error message

> *Exceptions: in modern shells (including bash) some of the more common commands are built into the shell*
>
> ```
> ls
> cd
> printenv, …
> ```

Adding Directories to Your Path

If you want to add new folders to your execution path (*e.g.* after installing some new software) the best place to do it is in .bash_profile (a hidden file in your home folder)

- edit the file to modify the definition of PATH
- save the file
- the next time you start a shell the new settings will take effect

Here are some of the settings in my bash profile:

```
PATH=/usr/local/bin:$PATH

PATH=~/Applications/Bioinformatics:$PATH

PATH=~/SysAdmin/Scripts/python:$PATH
```

To make it easier to understand (and to update it in the future) I have one addition per line

Each line adds a new folder to those currently in the path

IMPORTANT:

- *the variable name (no dollar sign) is on the left*
- *the variable value (with dollar sign) is on the right*
- *no space on either side of the equal sign*

Another New Command: `which`

A command named `which` will tell you where the shell finds an executable command

Example:

```
$ which python
/Users/conery/miniconda3/bin/python
```

The output tells me that when I run `python` the shell will start the version I installed with conda

To see all the different versions that are available:

```
$ which -a python
/Users/conery/miniconda3/bin/python
/usr/bin/python
```

> *The file in `/usr/bin` is the version of Python that comes preinstalled with macOS*

Permissions

Here is part of the output of the `ls` command when I asked it to show me the contents of the sim folder:

```
$ ls -l sim
-rw-r--r--   1 conery  329K Jan  9  2016 A_R1.fastq
-rw-r--r--   1 conery  329K Jan  9  2016 A_R2.fastq
drwxr-xr-x  32 conery  1.0K Jan  9  2016 artwork/
drwxr-xr-x   4 conery  128B Jan  9  2016 clusters/
drwxr-xr-x   5 conery  160B Jan 13 11:42 log/
-rw-r--r--   1 conery  1.8K Jan  9  2016 otus.fasta
-rw-r--r--   1 conery  241B Jan  9  2016 sim.cfg
-rw-r--r--   1 conery  1.0M Jan  9  2016 sim.db
```

The 10-character strings at the start of each line are **permissions**

- the first character is a `d` if the name the the name of a directory
- the next 3 characters define what I (as owner of the file) can do:
  `r` = read, `w` = write, `x` = execute
- the next 3 are the permissions for the members of my group, and the last 3 are for everybody else

Permissions (cont'd)

Here are some of the versions of Python installed by Anaconda:

```
$ ls -l ~/miniconda3/bin/python*
lrwxr-xr-x  1 conery  staff     9B Jan 19 10:39
/Users/conery/miniconda3/bin/python@ -> python3.6
-rwxr-xr-x  1 conery  staff   3.3M Jan 19 10:39
/Users/conery/miniconda3/bin/python3.6*
-rwxr-xr-x  1 conery  staff   135B Jan 19 10:39
/Users/conery/miniconda3/bin/pythonw*
```

★ an `x` on a plain file (not a directory) means the file is an executable program that can be run by a shell command

(An `l` at the front of a listing means the name is a "symbolic link")

Turn Your Programs Into Executable Files

PCfB explains how to make your own executable programs that can be run from the command line

Examples:

- collect a set of shell commands into a **shell script**
- turn a Python file into a **new shell command**, *e.g.* to run the 16S analysis pipeline

Shell scripts are useful when you find you're repeating the same steps over and over and want to automate those steps

Example: each time we run the 16S pipeline (the set of programs that produces the `sim` folder) we get subdirectories named `clusters`, `map`, `merged`, *etc*

- suppose you want to save the FASTA files in the `merged` folder and delete all the log files

You can put the shell commands that do those tasks in a file named `save_fasta.sh`. Then each time you want to clean up a project folder cd to that folder and run the command:

```
$ cd sim
$ save_fasta.sh

$ cd ~/meta/newproject
$ save_fasta.sh
```

Executable Script (cont'd)

Putting the commands in a file is simple. These are the four shell commands that do the job:

```
mkdir fastas
mv merged/*.fasta fastas
rm merged/log.*
echo "*.fasta moved to ../fastas" > merged/readme.txt
```

Turning that file into an executable program means you can run it from any directory

- you don't have to remember which folder you put it in
- you don't have to copy the file to each project directory you want to clean up

There are three steps (described on the following slides) to making an executable file:

1. change the file's permissions so it is executable
2. add a "shebang" line
3. put the file in a folder that is in your execution path

1. Add Execute Permissions

Use a shell command named `chmod` (which stands for “change mode”, as in “change read mode” or “change write mode”)

In the folder where you saved your file type this command:

```
chmod +x [filename]
```

Here is an example, using `save_fastas.sh`, showing its permissions before and after the `chmod` command:

```
$ ls -l save_fastas.sh
-rw-r--r--@ 1 conery  Feb 19 09:51 save_fastas.sh

$ chmod +x save_fastas.sh

$ ls -l save_fastas.sh
-rwxr-xr-x@ 1 conery  Feb 19 09:51 save_fastas.sh*
```

2. Add a Shebang Line

The shell now knows `save_fastas.sh` is an executable file, but it doesn't know ***how*** to execute it

- is it a C program? FORTRAN? Python?

That's where the "shebang" line comes in

Make this the ***very first line*** (no blank lines before it!) in your file:

```
#! /usr/bin/env bash
```

- the first two letters (***no indentation!***) are # ("shell comment") and ! ("bang")
- the rest of the line tells the OS to use bash to execute the remaining lines in the file

Here is my program, complete with shebang line and a comment at the start of the file:

```
$ cat save_fastas.sh
#! /usr/bin/env bash
# shell script to save Fasta files...
mkdir fastas
...
```

3. Move the File to a Location in Your Path

Earler we saw this command, which displays the names of the directories in your execution path:

```
$ echo $PATH
/Users/conery/miniconda3/bin:
/Users/conery/SysAdmin/Scripts/python:
/Users/conery/SysAdmin/Scripts/sh:
/Users/conery/Applications/Bioinformatics:
/Users/conery/Applications/bin:
```

Simply choose one of these folders and move your new script there

Since `save_fastas.sh` is a shell script I put it in the folder with my other shell scripts:

```
$ mv save_fastas.sh ~/SysAdmin/Scripts/sh
```

Turn Your Python Programs Into Executables

The same three steps can be used to run a Python program

Example: I have a Python program named `lorem.py` (each time it runs it prints the familiar “lorem ipsum” text)

```
$ lorem.py
Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do eiusmod…
```

I can run this program from anywhere

- I don’t have to remember which folder I put it in
- I don’t have to cd to that folder and type “python lorem.py”

I went through the same three steps to make this program executable:

1. I used chmod to add execute permissions
2. I added a “shebang” line [see next page]
3. the file is in ~/SysAdmin/Scripts/python

The “Lorem Ipsum” Script

The shebang line on a Python script tells the OS that the rest of the file contains Python statements

```
$ cat ../python/lorem.py
#! /usr/bin/env python3
print("""Lorem ipsum dolor sit amet, consectetur ...
... laborum.""")
```

Notes:

- I could use “python” instead of “python3” if I’m sure the default Python on my system is Python 3.x
- the triple quotes are a Python construct that allows us to make very long strings that span several lines.

Optional Topic: Conditional Execution in Shell Scripts

Bash and other Unix shells have if commands

- test some condition ("does a file exist? did I connect from a remote machine? is this file empty?)
- execute certain commands only if the condition is true

An example from my "bashrc" file (executed every time a new shell starts):

```
if [ -n "$SSH_CONNECTION" ]
then
  HOSTCOLOR=35
else
  HOSTCOLOR=34
fi
```

The Boolean expression is true if that environment variable is defined

Unlike Python, Bash doesn't use indentation to define bodies of if statements — mark the boundaries with `then`, `else`, and `fi`

Refer to online documentation (or better, find a book) to learn about the sorts of tests you can perform

Optional Topic: Iteration in Shell Scripts

Shells also have a type of for command for repeating groups of statements

The basic syntax is

```
for x in names
do
  commands…
done
```

Here `names` refers to a set of strings (often file names, but can be anything)

`for` commands can be typed in an interactive session — the shell notices you are typing a command that spans several lines and changes the prompt until you type `done` to end the command

Example: the sim folder has 6 FASTQ files (A_R1, A_R2, B_R1, etc). This loop makes three new folders (A, B, and C) and moves pairs of FASTQ files to the folder with that name:

```
$ for x in A B C
> do
>   mkdir $x
>   mv $x*.fastq $x
> done
```

*Note: x is the **name** of the loop control variable, $x is the **value** of the variable*

Shell Script *vs* Python Script

One of the main goals for this course is developing skills to automate analysis pipelines

- run applications (preprocess FASTQ files, find unique sequences, form clusters, map to reference, …)
- general “housekeeping” (remove log files, rearrange data, …)
- write programs to analyze data and plot results

Most of these these things can be done with shell scripts (but some may require conditionals and iterations)

Conversely, since Python is also a scripting language (a Python program can run another program) we can automate tasks with Python

Rule of thumb:

- ☆ if a task is complex enough to require conditional execution or iteration write a Python program

Our next topic this term: using Python as a scripting language