Tutorial: Examining file contents

Most files we will work with are 'pure text' files. That is, they contain only normal alphanumeric characters (a-z, 0-9) and things you could type on the keyboard. It is frequently useful to view these from the terminal. Say, for example, you want to make sure you entered all of the commands from the last Exercise into your journal file correctly. How would you do that?

Pick a journal file in your home directory (your home directory is `~`, or `/home/astr/ugrad/username/`. you can get there with the command `cd` or `cd ~`). We will examine its contents.

First, we'll use the simplest command available: `cat`, short for `conCATenate`.

```
cat filename.pro
```

You should see something like:

```
; IDL Version 8.1 (linux x86_64 m64)
; Journal File for ginsbura@cosmos.colorado.edu
; Working directory: /home/astr/grad/ginsbura
; Date: Mon Aug 27 13:57:12 2012
```

followed by your commands.

If you've picked a large file, you might not be able to read the whole thing in one screen. You can probably scroll up with the mouse, but there's another way to view the file contents.

```
$ less filename.pro
```

Within the `less` environment, you can use the up and down keys and the spacebar to scroll through the file. When you're done, press "`q`" (and remember this - `q` frequently gets you out of "interactive" windows).

You can also use both commands to show line numbers before the lines. With cat, use:

```
$ cat -n filename.pro
```

With less, use `$ less -N filename.pro`

Remember that UNIX *is* case sensitive! IDL and the Mac OS filesystem are not, but these are rare exceptions in the computing world!

But don't try to remember all of the "-n" and "-t" and "-v" options for each command here. That's a waste of your limited memory. That's what computers are for. There is a UNIX "manual" command `man` - use that.

```
$ man less
```

```
$ man cat
```

The documentation for `less` is long and verbose. Honestly, I only use it to quickly skim files. The documentation for `cat` is short and pretty easy to understand (although there are more details with the extra-verbose `info` command).

There's another handy command for remembering things. Say you want to find out what "text editors" are on the system. Use the `apropos`[1] command to find things related to text editors:

```
$ apropos editor
```

These are all of the editors you can use. Some of them are "stream editors" that we'll discuss another time, but many are text editors with a somewhat familiar feel. I recommend trying `nano`, `gedit`, and `vi` or `vim`.

[1] **apropos**: with reference to; concerning http://oxforddictionaries.com/definition/american_english/apropos?region=us&q=apropos

Tutorial: Tarballs, Zipping, and Backups

This tutorial is about packaging files. You may want to package data, code, or figures. This kind of task is done for you automatically if you attach multiple files to e-mails, but there are some systems that only accept tarballs (admittedly, dated ones).

However, there are many common daily uses for tarballs. We'll cover a straightforward one - backing up data - but the most common use for tarballs is distributing source code.

So, what's a tarball? It's a "folder" that's been compressed into a single file. They'll usually have the suffix `.tar`, `.tar.gz`, `.tgz`, or `.tar.bz`. The `.tar` is for tarball, the other bits of the extension indicate "zipping", or data compression, using either gzip or bzip (they're two algorithms; the first is faster but less efficient. Both make the files much smaller).

First, a reminder: Look at the manual file for `tar`.

```
$ man tar
```

Then, let's create a tarball of our backup directory. From the manual, we see which commands we need:

```
[-]c --create
-f, --file F
```

(the brackets around the dash, `[-]`, mean the dash is optional)

And let's include some options that are helpful but not necessary:

```
-v, --verbose
-z, --gzip
```

So our command will look like:

```
$ tar -czvf backup_20120903.tar.gz backup/
```

(make sure you're in your home directory: `cd`)

Note that the output `.tar.gz` file name comes before the folder name.

OK, we've got a backup copy of our backup directory. Neat. What next? Let's have a look inside first, using the `[-]t --list` command:

```
$ tar -tzvf backup_20120903.tar.gz
```

It lists the files you included along with their creation dates. You can do the same thing with `less`:

```
$ less backup_20120903.tar.gz
```

`git` is a system to automate this style of building "code snapshots". Dropbox, mercurial, and subversion all do different versions of the same thing.

Repeating some of Exercise 1, but in python

Re-do sections of exercise 1, but this time instead of using IDL, you'll use python.

`cd` to your `assignments0` directory. There, open `ipython`.

Redo exercise 1.0. Instead of `^`, use `**`.

Redo exercise 1.1. Instead of `print,y`, use `print y` (i.e., no comma, just a space). Instead of help, just type `whos` (you don't need to specify the variable names).

To get access to the number `pi`, you need to do this:

```
import numpy as np
print np.pi
```

Unlike in IDL, `pi` is not a built-in number. It is part of the `numpy` module, which is a module for numerical operations. `import` means you're loading the module to make it accessible. IDL doesn't have the concept of module importing in the same way as python.

Redo exercise 1.4. Replace all commands (`sqrt`,`sin`, etc) with `np.sqrt`, `np.sin`, etc. Replace `atan` with `np.arctan`, and similarly for others. Replace `alog` with `np.log`.

Look at exercise 1.7, but don't try to do it. Python does not have a ++ operator. Instead, do this (note that `>>>` means python prompt, just like `IDL>` means IDL prompt and `$` means bash prompt):

```
>>> x = 5
>>> x += 1
>>> print x
>>> x += 2
>>> print x
```

Do exercise 1.8. Instead of the `help` commands, use `whos`. Note that `a` will be a *list*, not an array. Replace line 4, which reads `a = [a,10]` with `a.append(10)`.

When you get to the line `b = 3*a`, things will get a little weird. Because `a` is a list and not an array, `3*a` means 'copy `a` 3 times'. So, do this:

```
>>> a = np.array(a)
>>> b = 3*a
... etc ...
```

At this point, quit out of python with the command `exit()`.

You should have a file that looks something like `ipython_log_2013-01-23.py`. Add this to your `git` repository and `commit`. Make sure your commit message notes that this is part of Tutorial 3.

Start up `ipython` again, but this time with an extra option:

```
$ ipython -pylab
```

This is the mode that allows you to plot.

Now, do exercise 1.9. Instead of `x = [...]`, do `x = array([...])`. You do not need the window command. The plot command will look like `plot(x,y)` instead of `plot,x,y`.

Unlike IDL, python does not clear the plot by default, so you never need to use a command like `oplot` - just use `plot` again. If you want to clear the plot, you can do it with the `clf()` command.

Instead of `linestyle=2`, use `linsetyle='--'`.

Instead of `psym=1`, use `marker="+"`

Where to get more information

IDL to python translation: https://www.cfa.harvard.edu/~jbattat/computer/python/science/idl-numpy.html

IDL docs: http://www.exelisvis.com/docs/

numpy docs (these are the documents relevant for scientific coding in python): http://docs.scipy.org/doc/

The numpy book (pdf, free, 378 pages!) http://www.tramy.us/numpybook.pdf