# Command Line Tutorial - Accessing and Organizing NOAA Data

March 8, 2019

## 0.1 # Accessing & Organizing NCEI/NOAA Data

For this tutorial we will be using some U.S. Hourly Precipitation Data, published by NOAA:

**Citation:**

> National Climatic Data Center, NESDIS, NOAA, U.S. Department of Commerce (2016). *U.S. Hourly Precipitation Data [dataset]*. NCEI DSI 3240_01
>
> National Climatic Data Center, NESDIS, NOAA, U.S. Department of Commerce (2016). *U.S. Hourly Precipitation Data [dataset]*. NCEI DSI 3240_02

From the webpage at https://data.nodc.noaa.gov/cgi-bin/iso?id=gov.noaa.ncdc:C00313, we see that data can be downloaded via the FTP server at ftp://ftp.ncdc.noaa.gov/pub/data/hourly_precip-3240/.

For our purposes, we are only going to download a small subset of the available data.

Our objective is to use shell commands to download and organize the data.

We will use the `wget` command line utility to download the data - if for any reason you are unable to install or get `wget` working, the data are available for download from https://unmm-my.sharepoint.com/:u:/g/personal/jwheel01_unm_edu/EQ4cpOBAiHVEhSUdkfQFSAcBgEnWUViyqdIXKvwglz6bWQ?e=flQL41.

### 0.1.1 Create the Project Directory

First we need to create a working directory for the project, using our `home` directory (Mac and Linux) or our user directory (Windows).

Within the shell, take a moment to determine which directory you are in, change to your `home` directory, and verify that you have done so.

Now use the `mkdir` command to create a directory named *precipitation_data*. Note that, similar to `cd`, the `mkdir` command does not print any output. Once done, change into this directory.

```
mkdir precipitation_data
```

### 0.1.2 Download Data

One way to get the data we need is to use a web browser to download the files from the FTP site linked above. This is straight forward but presents two problems:

1. Each directory contains multiple tarfiles of data - unless we use a bulk download manager, the more data we need, the longer it will take to download.
2. Unless we specify the download directory each time, the files will be saved to some other location on our computer and we will have to move them to our *precipitation_data* directory.

This is exactly the type of repetitive and error prone task that benefits from batch processing using shell commands. We're going to use a command line utility called `wget` to download the complete contents of each directory. First, we can use the standard `--help` flag to check the usage and parameters.

```
wget --help
```

More info on `wget` is available at https://www.gnu.org/software/wget/.

Looking at the help info, we see there are a couple of ways we can simplify our download process. One option is to use the `-r` flag, which recursively follows the links found within the specified URL. In our case, this will result in downloading all of the files in whichever FTP directory we specify when we run the command:

```
wget -r ftp://ftp.ncdc.noaa.gov/pub/data/hourly_precip-3240/66/
```

**Quick check:** What's one way we can re-run the command above for two more FTP directories without having to paste in or type the full URLs below:

```
ftp://ftp.ncdc.noaa.gov/pub/data/hourly_precip-3240/67/
ftp://ftp.ncdc.noaa.gov/pub/data/hourly_precip-3240/91/
```

This strategy works fine for a small handful of URLs, but itself becomes repetitive (and error prone) if we run the command manually to download the contents of many more directories. Referring back to the help info, another option is to use the `-i` flag to specify an input file that contains a list of the URLs we want to harvest from.

There are multiple ways to create the file we need for this, including using a text editor in the shell itself. For today, we will instead create the file in the shell using the `touch` command and then use our operating system's default text editor to add some URLs.

```
touch --help
```

After consulting the help info, let's create an empty file called `urls.txt`:

```
touch urls.txt
```

`touch` is another command that doesn't print any output. How can we verify our file was created?

Open `urls.xt` with a text editor and paste in the following lines:

```
ftp://ftp.ncdc.noaa.gov/pub/data/hourly_precip-3240/51/
ftp://ftp.ncdc.noaa.gov/pub/data/hourly_precip-3240/50/
ftp://ftp.ncdc.noaa.gov/pub/data/hourly_precip-3240/48/
```

Keep in mind that we can add as many URLs as we like and will still only have to run a single `wget` command. Even better, this file becomes documentation that we can use to double check or replicate/reproduce our work.

We can now use `wget` with the `-r` and `-i` flags to download the contents of the FTP URLs listed in `urls.txt`.

```
wget -r -i urls.txt
```

The final lines of the output gives us some idea of the effort we saved:

```
FINISHED --2019-02-05 20:32:07--
Total wall clock time: 39s
Downloaded: 42 files, 11M in 21s (521 KB/s)
```

**Exercise** Documentation for the data are available from the FTP site linked above. Using shell commands:

1. Create a *documentation* directory in the *precipitation_data* directory. Change into this directory.
2. Use one of the above methods to download the following two files into the *documentation* directory:
    - dsi3240.pdf
    - readme.txt

### 0.1.3 Access the Data: Unzip and Move Files

So far so good, but now we have 80+ tarfiles that have to be decompressed before we can use them! As with downloading, this is something we could do manually but it would take longer than necessary.

Additionally, if we uncompress the files into the same directories as the compressed tarfiles we downloaded, our workspace becomes cluttered. So to keep our organization simple, we also want to move the decompressed files to a different directory.

We will do one manually just to demonstrate.

We will also use the `tar` command to go through the process for a single file, but first let's create a directory to store our decompressed data. Using one of the methods we discussed for specifying a relative or absolute path, create a new directory *decompressed_data* in the *precipitation_data* directory.

Here's the steps for untarring a single file. Keep in mind that that path specified with the `-C` flag will vary depending on your operating system.

```
tar --help
```

```
tar -C ~/Documents/work/precipitation_data/decompressed_data/ -xf 3240_48_1948-1998.tar.Z
```

Considering how many files we have, this is not much of an improvement over using a GUI to uncompress everything manually. We could try a wildcard, like `tar -C ~/Documents/work/precipitation_data/decompressed_data/ -xf *.tar.Z`, but unfortunately `tar` doesn't allow such use.

**Using `For` Loops** As an interpreter, the shell allows us to do more than execute single commands. For our current purposes, we can use **for**, **while**, and **until** loops to iterate a command over a group or series (in our case, a series of files). We won't be using conditional logic today, but conditional (**if**) statements are also available for processing shell commands.

The basic syntax of a **for** loop is

```
for i in series; do
    some command $i
done
```

Using the `tar` command from above, we can use the following loop to untar all of the files in a directory.

```
for t in *.tar; do
    tar -C ~/Documents/work/precipitation_data/decompressed_data/ xf "$f";
done
```

A further refinement to make the loop easier to read is to set the path to the *decompressed_data* directory as a variable. Using "dataDir" as our variable name:

```
dataDir=~/Documents/work/precipitation_data/decompressed_data/ # Note - no spaces in the varial
for t in *.tar.Z; do
    tar -C $dataDir -xf $t;
done
```

This works! But we have several directories of tarfiles, and again we would find ourselves repeating the process for every directory. No problem with a handful, but if we had downloaded all of the data from the FTP site it would still be a time consuming process.

Fortunately, loops can be nested - we can run a loop in a loop - which gives use the possibility to untar ALL of the downloaded files at once. Making sure to run the following command from the parent directory containing all of the downloaded data directories:

```
dataDir=~/Documents/work/precipitation_data/decompressed_data/
for d in */; do
    for t in $d/*.tar.Z; do
        tar -C $dataDir -xf $t;
    done
done
```

That's it - all of the data have been decompressed into the specified directory.

Before moving on, it's possible we may want to download additional data at some point. Rather than go through the process of rewriting the above loop, we can save it as a script which can be executed at need. All that's needed is to add one line:

```
#!/bin/bash
dataDir=~/Documents/work/precipitation_data/decompressed_data/
for d in */; do
    for t in $d/*.tar.Z; do
        tar -C $dataDir -xf $t;
    done
done
```

**Exercise** Using the `touch` command and a text editor, create a file named `untar_precip_data.sh` and copy the above script into it. Be sure to create the file in the directory which contains all of the download directories.

Before we can run a shell script we have to make it executable by using the `chmod` command's capability to set the *executable* permission for the user, group, or others for the script file.

```
chmod --help
```

```
chmod u+x untar_precip_data.sh
```

As needed, the script can be run as

```
./untar_precip_data.sh
```

Note that in many cases we need to specify the path to the script - here we have used the shorthand for calling the script from within the directory that contains it.

### 0.1.4 Interacting with Files - Move, Copy, Delete

Before we do anything with the data files, we can do a little more cleanup. Along the way, we have created a couple of files - `urls.txt` and `untar_precip_data.sh` in particular - that have helped us batch process certain tasks. In the course of a research project we may create many such files, and they can become hard to keep track of over time. It may be useful to create a special directory for these.

Change to the *precipitation_data* directory and create one more directory named *scripts*. Do not change into the new directory.

Since `urls.txt` is in our working directory, we will move that one first. Because we might want to use other, similar files in the future to download more data, we want to avoid overwriting our existing file by giving it a more specific name that simply `urls.txt`. In this case, we will rename the file by adding today's date.

There is no "rename" command in the shell, but we have a few options. First, we can use the `mv` or "move" command to rename the file in place.

```
mv --help
```

```
mv urls.txt 2019-02-06_urls.txt
```

Again, no output is printed to the screen but we can use `ls` to verify that the file was renamed. We can use a second `mv` command to move the file to the *scripts* directory:

```
mv 2019-02-06_urls.txt scripts/
```

```
ls scripts
```

We could also have done it all with one command. Since we already moved our file, we can demonstrate this by creating a new, empty file that will also be named `urls.txt`.

Go ahead and create this file using the `touch` command.

The command to both rename and move this new file is

```
mv urls.txt scripts/2019-02-07_urls.txt

ls scripts
```

Now let's add the `untar_precip_data.sh` script to the *scripts* directory. That file is pretty deep in our current folder hierarchy, so rather than navigate back and forth, we can use the `find` command to search for it. A successful search will return the path to the file we're looking for, which we can then use to copy or move the file without having to change directories:

```
find --help

find -name "*.sh"
```

Note that we used a wildcard to search recursively for any shell scripts in our working directory and subdirectores. This is useful if we are going to need to copy or move more than one file.

Now that we have the path to the script, we can copy it to our *scripts* directory using the `cp` command. We can also rename it while copying. To copy it from within the *precipitation_data* directory, use the relative path as below:

```
cp --help

cp ./ftp.ncdc.noaa.gov/pub/data/hourly_precip-3240/untar_precip_data.sh scripts/recursive_untar

ls -l scripts/
```

Now we have moved or copies all of our helper files to our *scripts* directory, but the original copy of `untar_precip_data.sh` is still in its original directory. If we decide we don't need it, we can delete it using the `rm` command:

```
rm --help

rm -f ./ftp.ncdc.noaa.gov/pub/data/hourly_precip-3240/untar_precip_data.sh

ls ./ftp.ncdc.noaa.gov/pub/data/hourly_precip-3240/
```

Note that `rm` is more or less forever - files deleted this way do not go into a recycle bin! To demonstrate, let's say we feel that we have documented our process well enough, and since NOAA has the original data we can delete all of the tarfiles we downloaded earlier. We can do this with a single `rm` command from within the *precipitation_data* directory:

```
rm -rf ftp.ncdc.noaa.gov
```

This is a useful way to quickly clean up a work space, but should be used with caution!

**Exercise** The documentation files we downloaded earlier are also deeply nested within the *documentation* directory. Move them to the top of the *documentation* directory and delete the other, empty subdirectories.

### 0.1.5 Getting File Info & Reading Files

A last thing to do for today is to read our data files and find out some basic information about them.

Change to the *decompressed_data* directory. Use `ls -l` to list information about the directory contents. Note that file sizes are provided in bytes - we can also use the `-h` flag to generate more human readable file sizes.

Since these are text files, we could open them in any text editor. However, if we don't need to edit the files a quicker and safer way to view the contents of a text file is using the `cat` command.

```
cat --help

cat 3240_914951_por-1998
```

If a file is large, we can pipe the output of the `cat` command to the `less` command, which will allow us to scroll through the output:

```
less --help

cat 3240_914951_por-1998 | less
```

Another useful command for getting information about files is `wc`, which provides line, word, and character counts for a specified file.

```
wc --help

wc -l 3240_914951_por-1998
```

From this point, we would generally use a shell script or other scripting language to send the data through an analysis pipeline.