Today, you'll be learning bash commends on your laptop. First, download the relevant folder (**Day1_worksheet_files.zip**) from the **data** directory on GitHub and move it to your Desktop.

Open up Terminal (or Command prompt if using Windows[1]). Navigate to the Desktop of your computer.

```
cd Desktop
```

How do you check that your current directory is your Desktop?

```
pwd
```

You'll notice that this folder is compressed (i.e., it's got a **.zip** extension). Unzipping is an important skill on TACC, as many programs/pieces of software are provided as a compressed file. To unzip files on your personal laptop, the commands are slightly different from those we'll use on TACC. Run the following line to unzip the `Day1_worksheet_files.zip` file.

```
unzip Day1_worksheet_files.zip
```

Delete the `Day1_worksheet_files.zip` file (*be careful to not delete the actual folder that you've just unzipped!)*

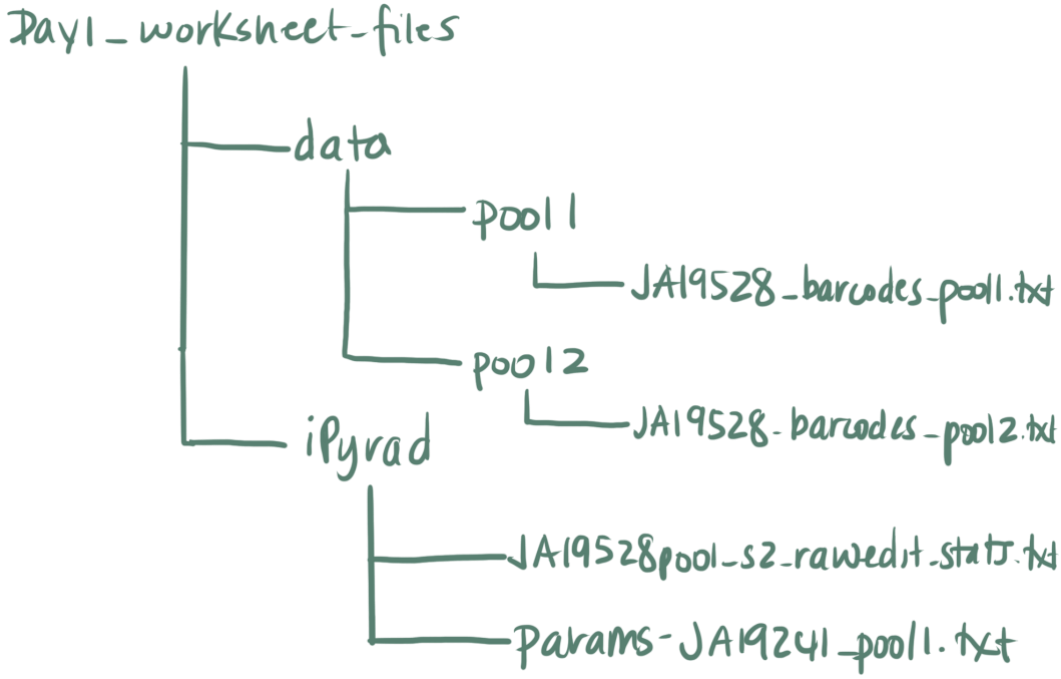
```
rm Day1_worksheet_files.zip
```

Navigate so that you're now in the Day1_worksheet_files directory.

```
cd Day1_worksheet_files
```

Navigating around the file system

The Day1_worksheet_files directory has a series of nested folders within it that has the following structure:



[1] https://www.laptopmag.com/amp/articles/use-bash-shell-windows-10

Move into the `pool1` directory.

```
cd data
cd pool1
```

Move backwards two directory levels, from `pool1` to `Day1_worksheet_files`.

```
cd ../../
```

Now, move from `Day1_worksheet_files` into `pool1` *in a single line of code*.

```
cd data/pool1
```

Getting file characteristics, copying, and renaming files

Navigate back to the `Day1_worksheet_files` directory, and move into the `iPyrad` directory *in a single line of code.*

```
cd ../../iPyrad
```

Make a **copy** of `params-JA19241_pool1.txt` in the same directory and name it `params-JA19528_pool1.txt`.

```
cp params-JA19241_pool1.txt params-JA19528_pool1.txt
```

Navigate back to the `Day1_worksheet_files` directory. *Without moving out of this directory*, make a **copy** of the `params-JA19528_pool1.txt` file you just made and name it `params-JA19528_pool2.txt`.

```
cd ..
cp iPyrad/params-JA19528_pool1.txt iPyrad/params-JA19528_pool2.txt
```

Move the `JA19528pool1_s2_rawedit_stats.txt` file from the `iPyrad` directory into the `data` directory.

```
mv iPyrad/JA19528pool1_s2_rawedit_stats.txt data/
```

Rename the `params-JA19241_pool1.txt` file `params-JA19528_pool3.txt` from your current directory.

```
mv params-JA19241_pool1.txt params-JA19528_pool3.txt
```

Move into the `iPyrad` directory. How **large** are all of the params files?

```
ls -l
```

Move into the `data` directory.

```
cd ../data
```

In a single line of code, what do the contents of `JA19528_barcodes_pool1.txt` (which is within the `pool1` directory) look like?

```
cat pool1/JA19528_barcodes_pool1.txt
```

[1] https://www.laptopmag.com/amp/articles/use-bash-shell-windows-10

Let's check how many samples should be in each of the pools. Are there the same number of **lines** in the `JA19528_barcodes_pool1.txt` as `JA19528_barcodes_pool2.txt`?

```
cat pool1/JA19528_barcodes_pool1.txt | wc -l # 23 lines
cat pool2/JA19528_barcodes_pool2.txt | wc -l # 23 lines
```

Using wildcards

From your current directory (`data`), using a wildcard, list the two barcode files for both pools 1 and 2. *There are multiple ways to do this.*

```
ls pool*/*.txt
ls pool*/*barcodes*
ls */*.txt
ls pool*/JA19528*
```

Move out a directory to `Day1_worksheet_files` and using a wildcard, list *only **barcodes** files containing* "**pool1**" in their file names.

```
ls data/*/*pool*
```

Editing files

Move into the iPyrad directory.

```
cd ../iPyrad
```

Take a look at the contents of the `params-JA19528_pool1.txt` file.

```
cat params-JA19528_pool1.txt
```

This is a typical parameter file from iPyrad. We'll explore the details of this file later, but for now all you need to know is that there is a parameter setting on each line, and that the details (and numbers) of the 30 parameters are provided by the "## [X]" strings on the right. The double hashtag/pound symbol indicates that these are comments and will not be read in by iPyrad. The actual parameter settings – and what iPyrad *will* read – are on the lefthand side of this file, to the left of the comments.

Now, edit the `params-JA19528_pool1.txt` file to do the following:

- Make the assembly_name parameter (parameter 0) be JA19528pool1
- Make the path to pool1 be the current path from Day1_worksheet_files to the pool1 barcodes file (parameters 1, 2, and 3)

```
nano params-JA19528_pool1.txt
JA19528pool1 # for parameter 0
../data/pool1/JA19528_barcodes_pool1.txt # for parameters 1-3
```

[1] https://www.laptopmag.com/amp/articles/use-bash-shell-windows-10

Advanced bash – to do on your own time

To do some advanced bash, download the relevant folder and move the folder to your Desktop, using only bash.

```
cd ~/Desktop

wget https://github.com/eachambers/UNAMtraining/blob/main/data/
Day1_advanced_files.zip

unzip Day1_advanced_files.zip
```

Sequencing data

There are several file types for commonly used sequencing data that you'll get from any NGS dataset, but we're going to focus on fastq files.

FASTQ files

As you already know from lecture, with typical RADseq data such as ddRAD, we usually sequence "paired-end" data. This means that each pool has two reads: a forward read (R1) and a reverse read (R2).

Fastq files are the raw files that are sent to us from the sequencing facility and so are the first step in processing our data. There should be two copies of each fastq file; they'll be labeled *R1_.fastq and *R2_fastq. As mentioned above, R1 and R2 indicate the different directions of sequencing. Importantly, .fastq files tell us not only the actual sequence information, but also information about quality scores for each nucleotide in the sequence.

A typical .fastq file looks something like this:

```
@SIM:1:FCX:1:15:6329:1045 1:N:0:CCGG

TCGCACTCAACGCCCTGCATATGACAAGACAGAATC

+

<>;##=><9=AAAAAAAAAA9#:<#<;<<<????#=
```

Each read always begins with an @ character followed by an identifier (in this case, SIM; keep in mind that most of this is optional). The first line also has information about how many Ns there are (0), and what the adaptor sequence is (CCGG). Line 2 is the actual raw sequence. Line 3 is simply a separator, and is always '+'. Finally, line 4 tells us the quality value of each of the nucleotides in line 2. Here is the code for quality scores, starting from lowest (!) to highest (~) quality.

[1] https://www.laptopmag.com/amp/articles/use-bash-shell-windows-10

There is a single file in this directory, called `alt16w2m_TX_R1_.fastq.gz`. Now, take a look at the *beginning of the* `alt16w2m_TX_R1_.fastq.gz` file.

```
head alt16w2m_TX_R1_.fastq.gz

# Output looks like:
r%{aalt16w2m_TX_R1_.fastq˟›?:,?ۇ?,?'%?qq??d?f3s7????????$(AYY?[}*+S??
?w???????p8_??>????_????????????˛????????z<??x8????????!???e?R??B⁄??J???
uG?&??
K??^ᶲ?zF????????z???Yy???????SL|?????o?c?7???????7???6U???2m?????{;???9??F
t- ??+????_?h??_???B????????/z=+??O??
x??w?c2ó?79???????w?~?5???y???Qt?65???????k?ﻭn?C^=?_??|?V????W]ʋ9?ȠT??ˏジ
??D???n~̋̊?_??Vo??c?Ӭ#???????|m????zvx>)|ןּ?/=?????????_???
歛??/7?ʌ??
?Z??^?IOA???PDd_?>???K??!?????Ҍ???ݛ??d???z?4?^5?2??OOf?\??G+??? ?<???
```

It looks messed up! This is because the file extension ".gz" means this is a compressed file, much like the .zip file we encountered in the in-class worksheet. Because this is a different file extension than .zip, we can't use the same command (unzip) to unzip this file. To unzip .gz files, we use the command gunzip.

We can also view the .gz file contents without actually unzipping it. To do this, we still use the gunzip command, but also add in the -c flag. Do this now to see the first 10 lines of the .gz file.

```
gunzip -c alt16w2m_TX_R1_.fastq.gz | head
```

Print out the first 20 lines of this file. (*Hint: the -n flag, used with head, can reveal the top number of lines of your choosing; just specify the number after the flag*).

```
gunzip -c alt16w2m_TX_R1_.fastq.gz | head -n 20
```

Let's take a look at the first sample contained within `alt16w2m_TX_R1_.fastq.gz`. What does it look like?

```
@NS500358:231:HKHLMBGXC:4:11401:12512:1020 1:N:0:CCGCGT
AATTCGCGACGGGGGTGTTGTGGAGGGTGCAACGTTGGCAAGCCCAGCCCTCCTCCAGC
TCTGCCTCTTCGGGGCTCTTCCCCCCCGAATCCTCAACGGGGCTCTCCATCTGGGGCGTC
TCCTCCTTGCCACTGGCTTTGGGTTC
+
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEAEEEEEEAEEEEEEEE
EEEEEE<EEEEEEEEEEAA6EEA<EEEEEE<EEEEEEEEEEEAAEEAEEEEEEEEEEEEA<EEEE
EEAEEEEEEEAEEAEEEEA/
```

What is the sample name for this read?

```
NS500358
```

What is the adaptor sequence?

```
CCGCGT
```

[1] https://www.laptopmag.com/amp/articles/use-bash-shell-windows-10

How is the quality of this sequence? Quality scores ("Q scores") range from 0 to 40. Check out this chart to answer this question.

```
Very good! Most of the calls have a Q score of 36 (are Es). The lowest
call has a Q score of 21 (6).
```

[1] https://www.laptopmag.com/amp/articles/use-bash-shell-windows-10