**Computer Lab 12 – Assembling RADseq data with pyRAD**

**Conservation Genetics (BIOL 4174 / 5174)**

Installation

ipyrad unfortunately relies on an outdated version of Python. We’ll use conda to set up this older Python environment. First, we tell conda to make a separate Python 2.7 environment:

conda create -n py27 python=2.7 anaconda

This will take a minute to complete. Once its finished, we’ll load this environment:

source activate py27

Next, we’ll call conda (in the py27 environment) to install our software:

conda install -c ipyrad ipyrad

conda install -c bioconda fastq

The original version of pyRAD, as well as tutorials for its use, can be found at: <http://dereneaton.com/software/pyrad/>. Today, we’ll be using its successor, ipyrad. Tutorials and documentation for ipyrad can be found here: <http://ipyrad.readthedocs.io/>

Part I: Examining Illumina Data

First, we’ll use the free program FastQC to examine the quality of our Illumina output, before applying any filtering to the data. Open a Terminal and change directories to the location of your Lab12 files. Then, unzip the raw Illumina data. To speed things up, today you’ll be working with a small subset of a full Illumina run (2,000,000 reads from the original ~500,000,000 reads):

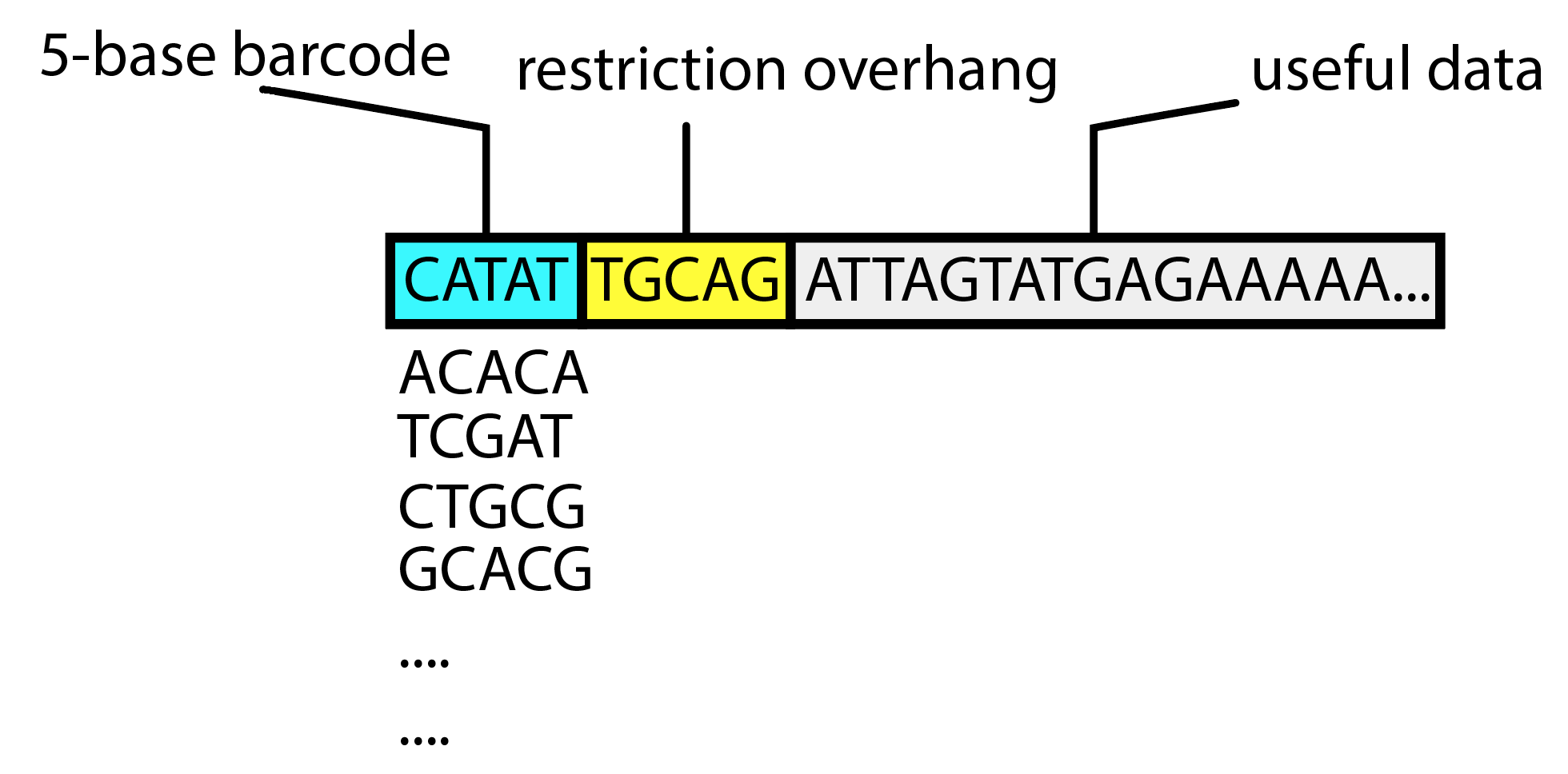
gunzip ddrad\_example.fq.gz

This will create a FastQ file ‘ddrad\_example.fq’ which contains your raw reads. Assuming that you have completed the conda instllation (above), you can run FastQC on the raw Illumina data like so:

fastqc ddrad\_example.fq

FastQC will take a few moments to run. Once it has completed, you will notice a ddrad\_example\_fastqc.html file in your Lab 12 directory. Open your Finder window, and double-click this .html file to view it in your web browser.

The first section is ‘Basic Statistics’. Here, you will see reported the number of reads sequenced, their length, and overall GC-content.



Below this are a series of plots. Here, I’ll briefly describe the ones relevant to today’s lab, along with how you would interpret it. For future reference, the following website is also very helpful: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/3%20Analysis%20Modules/

Per base sequence quality:

This plot shows a series of box plots for each base position in your reads (X-axis) of quality scores (Y-axis). The higher the quality score, the higher confidence in the base call. Typically, the overall quality will decrease along the length of the read, with the ‘most confident’ base-calls early in the read. If qualities dip drastically into the red (<20, or <99% confidence) on either end, you may wish to trim off these lower-quality bases using tools such as Trimmomatic, or the end trimming options in pyRAD. Consistently low scores along the entire length of the read could be symptomatic of problems with the run, such as overclustering, which we will diagnose using the other plots produced by FastQC.

Per tile sequence quality

This plot shows deviation in sequence quality in different tiles, which could be used to find position-specific failures on the flowcell (indicating by streaks of red in the plot), as well as help indicate overclustering.

Per sequence quality scores

This plot shows the distribution of average quality scores across your sequence reads. Most reads should have a fairly high average quality. If a significant portion of reads have low quality, or if this plot is multi-modal, it could indicate some sort of systematic problem- for example with the flowcell, or with mis-loading of your library causing overclustering.

Per base sequence content

This plot shows the distribution of different nucleotides at each position. In a completely random library, we would expect little difference in the frequencies of nucleotides at different positions along the read. However, our reads are not entirely random! **Take a look at the figure above, which shows the components of an unedited RADseq read, and see if you can figure out why the per base sequence content plot looks so wonky.** (Note: There will be no homework for the grad-only labs).

Per sequence GC content

This is the distribution of GC counts per read (red) versus the theoretical expectation of a normal distribution (blue). An unusually shaped distribution, or multimodality, could indicate presence of contamination (e.g. bacterial contamination in DNA extraction), while a shifted distribution could represent some systematic bias.

Overrepresented sequences

This table reports sequences which were excessively sequenced. These could indicate contamination, Ilumina adaptors which failed to trim, or in the case of RADseq, our barcodes and restriction sites (see figure above).

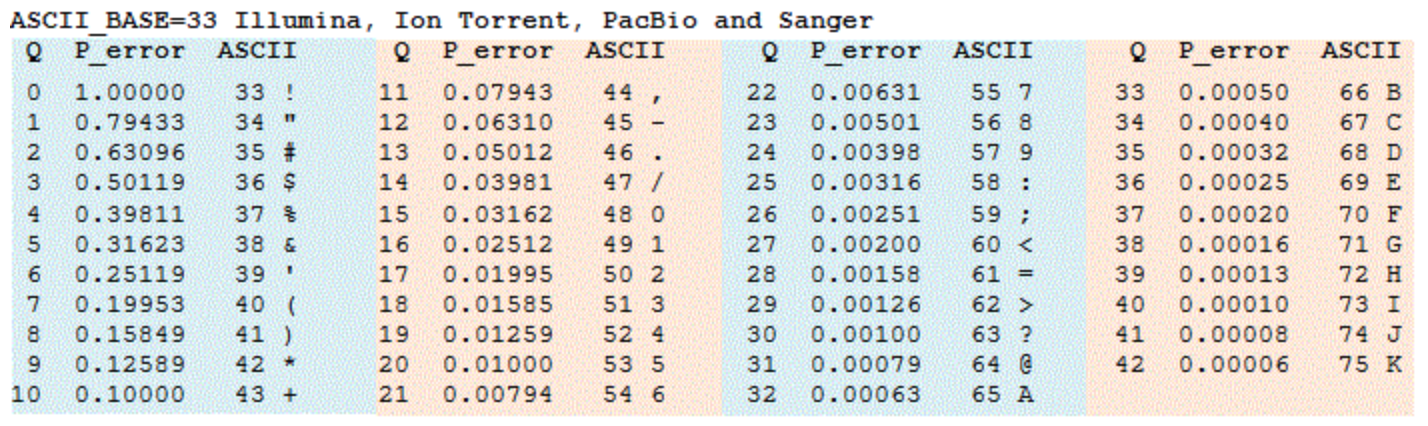
Part II: ipyrad

Before running pyrad, take a look at the FastQ format:

head -4 ddrad\_example.fq

This prints the first 12 lines- each sequence read is represented by a 4-line block: Line 1 is a string of characters beginning with an @ symbol that is used to provide a unique identifier to the sequence. Line 2 is the sequence. Line 3 begins with a + symbol and often repeats the unique identifier. Line 4 is a string of ASCII characters that are used to denote the quality score for each base in the sequence.

* 1. **The first six bases of the sequence represent the barcode. What is the barcode of this sequence?**
  2. **Do you find this barcode in the barcodes.txt file?** 
     1. You can open the file in Textwrangler and look through it to see if you can find this barcode. If there are “N” sequences in the barcode, find the closest match.
  3. **The next five bases after the barcode represent the cut site of the restriction enzyme (sbfI) used in this study. These bases should be TGCAG. Does this match your sequence?**
  4. **Overall, how would you rate the quality of this sequence?** All sequences in this file were scored according to the Phred33 format- see the table below.



Let’s also take a look at the barcodes file. This will be used to sort the raw reads based on the first 5 bases, into files for each sample. You’ll see sample names (left) separated by a tab from their associated barcode sequence (right).

cat ddrad\_example\_barcodes.txt

*Create an ipyrad params file*

Now that we understand what our datafile look like, let’s create the parameters file for ipyrad. This will be a simple text file that will contain all of the running parameters for ipyrad. Assuming you have completed the conda install, call ipyrad to create a params file for the assembly ‘ddrad\_example’:

ipyrad -n ddrad\_example

This should create a new parameters file called “params-ddrad\_example.txt”. Open it in TextWrangler and set the following:

1. Set [2] raw\_fastq\_path to: ddrad\_example.fq
2. Set [3] barcodes\_path to: ddrad\_examples\_barcodes.txt
3. Set [5] assembly\_method to: denovo (this tells pyrad we don’t have a genome to assist assembly)
4. Set [7] datatype to: ddrad
5. Set [8] restriction\_overhang to: TGCAG, CG
6. Set [11] mindepth\_statistical to: 10
7. Set [12] mindepth\_majrule to: 6
8. Set [14] clust\_threshold to: 0.88
9. Set [15] max\_barcode\_mismatch to: 0
10. Set [16] filter\_adaptors to: 1
11. Set [17] filter\_min\_trim\_len to: 50
12. Set [20] max\_Hs\_consens to: 10,10
13. Set [21] min\_samples\_locus to: 12
14. Set [22] max\_shared\_Hs\_locus to: 0.7
15. Set [27] output\_formats to: \*

Save the file and go back to your terminal. Note that some of these have been relaxed a little more than I usually would, to accommodate the subset data wer are working with today. Now we’ll run Step 1 to demultiplex our data, using the barcodes set in the ddrad\_example\_barcodes.txt file. This shouldn’t take long with our small-ish subset file:

ipyrad -p params-ddrad\_example.txt -s 1

At the end of this step, there should be a new directory called ddrad\_example\_fastqs/. This contains a separate FastQ file for each sample, sorted based on those barcodes from our ddrad\_example\_barcodex.txt file. Use **ls -lh** to view the files in this folder. You should see that some of the samples have relatively little data (e.g. popFsample4 has only 3.7Kb of fastq-formatted reads, while PopAsample1 has 3.7Mb!). This could be caused by errors in balancing samples in the library prep, or simply as an artefact of the sub-sampling I did to create today’s input file.

Now, let’s perform the read filtering and editing of Step 2:

ipyrad -p params-ddrad\_example.txt -s 2

You should now have a new directory ddrad\_example\_edits/ which contains the reads passing those filters. We did not apply very stringent filters, and so likely did not see much loss in data. You can check out a summary of the results by looking at the ddrad\_example\_edits/s2\_rawedit\_stats.txt file:

less -S ddrad\_example\_edits/s2\_rawedit\_stats.txt

Now, lets cluster reads within samples. This is one of the most computationally intensive steps. With today’s toy dataset it should complete fairly quickly- with much larger real datasets, expect this to take quite a while longer!

ipyrad -p params-ddrad\_example.txt -s 3

This command will cluster reads within samples to create ‘stacks’ which hopefully all represent sequences from the same place in the genome, or ‘loci’. The goal is to find reads which map to the same location in the genome, under the assumption that the percentage of sequence similarity we have set will not cause multiple locations in the genome to be ‘clumped’ together. This could happen for example with paralogous loci, or those resulting from duplications in the genome. However, we’ll try to identify and remove these cases in later steps.

Once step 3 is complete (on my desktop it took about 3 minutes), you will see a new directory called ddrad\_example\_clust\_0.85. Let’s see a summary of our results so far:

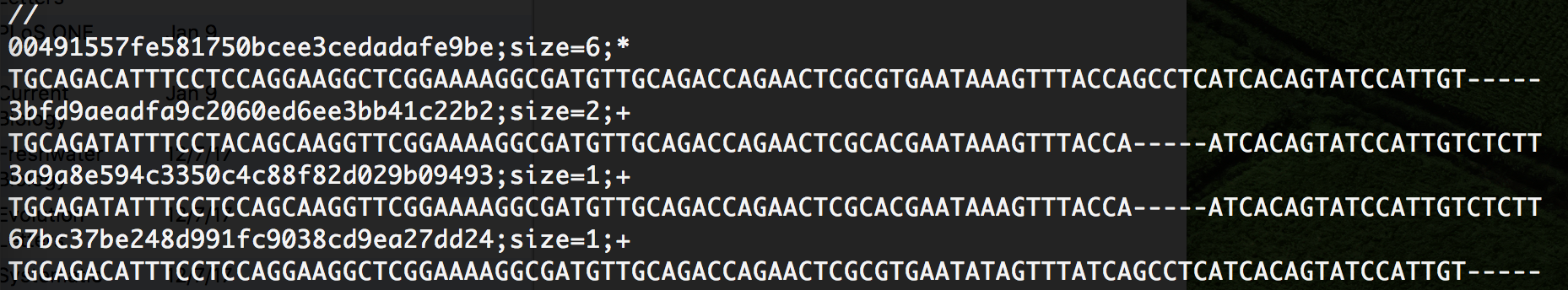
ipyrad -p params-ddrad\_example.txt -r

This should show you rows for each sample, with columns showing how many raw reads were present, how many reads passed filtering, and how many loci were present at a sufficient depth (clusters\_total). The full log file for Step 3 can be viewed by using **cat** or **less** on the file ddrad\_example\_clust\_0.85/s3\_cluster\_stats.txt.

Let’s check out the clusters which were found within the first individual, to get a better feel for what’s going on:

gunzip -c ddrad\_example\_clust\_0.85/popAsample1.clustS.gz | head -n 36

Here, we’ll see reads that are sufficiently similar grouped together in clusters separated by “//”. Reads which are completely identical are collapsed together, and represented using the ‘size=’ field in the header for each sequence. For example, in the following ‘read cluster’, there are 4 variants, with 10 total reads in the cluster:



These variants could represent true alleles, or they could be errors introduced during sequencing or PCR. Luckily, steps 4 and 5 will figure all of this out. Let’s call them both at once:

ipyrad -p params-ddrad\_example.txt -s 45

Here, we are trying to estimate error rates and heterozygosity (4), and using these estimates to try and determine what the ‘true’ sequence is for each cluster (5). These consensus sequences are placed in a new directory ddrad\_example\_consens/. Let’s take a look at the results so far:

ipyrad -p params-ddrad\_example.txt -r

You should now have some new columns in your summary results table- heterozygosity and error estimates for each sample (hetero\_est and error\_est) as well as the total number of observed loci for each individual (read\_consens). Remember, we are working with a small subset of the data- the full run for this dataset identifies between 20,000-30,000 passing loci within each of these samples! Let’s take a look at one of them:

gunzip -c ddrad\_example\_consens/popAsample1.consens.gz | head -10

This should show you the first 5 loci. Not that now they are not represented as ‘read clusters’, but instead as a single consensus sequence for each sample. Neat! Not, run Step 6 to cluster these loci across all samples, to figure out which ones are homologous, based on the similarity threshold we set in our params file:

ipyrad -p params-ddrad\_example.txt -s 6

This should have created a new directory ddrad\_example\_across. Let’s go ahead and run the final step now, to turn this output into something usable for analysis:

ipyrad -p params-ddrad\_example.txt -s 7

Your final output should now be located in ddrad\_examples\_outfiles/. Let’s check them out. First, look at the pyrad-native “.loci” format:

less -S ddrad\_example\_output/ddrad\_example.loci

This shows an alignment for each locus, separated by “//” between loci. Next, take a look at the “snps.phy” file, which shows 1 SNP per locus, all pasted together. The “ddrad\_example.str” file would be used as input to STRUCTURE (which we used last week). Next week, we’ll use some of these inputs for real analyses.

Finally, a summary of all filtering: ddrad\_example\_outfiles/ddrad\_example\_stats.txt. Note that there are not very many loci in the final files- this is because of the fact that we input 1/250 of the total data.