

Exercise 8 - String manipulation with regex in R

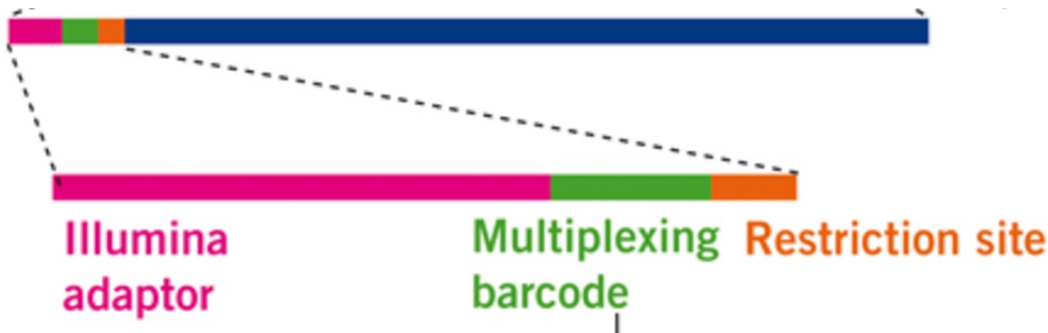
Work with a partner to complete the tasks below and submit your results via a pull request on GitHub by the beginning of tutorial next Friday.

To begin this week, one of the partners should fork the TA's Exercise 8 Github repo and provide collaborative access to the other partner. Clone the forked repo so that you have the required files. Be sure to commit regularly to avoid merge conflicts and demonstrate effort by both partners.

Background

RADseq

The cost of high-throughput sequencing technologies continue to decrease... but sometimes grant funding does, too... Rather than sequencing the whole genome of each sample, many population geneticists stretch their sequencing dollars further with reduced representation techniques. In this tutorial, you will be working with RADseq (restriction site associated DNA sequencing) data generated by me (Meredith) in the Feder Lab! Briefly, genomic DNA from each sample was digested with two restriction enzymes (EcoRI is the important one today), fragmenting the genome. Adapters with restriction enzyme “sticky ends” were then ligated to the genomic DNA fragments. As part of these adapters, a unique sequence of ~8 bp for each individual sample, sits just 5' of the EcoRI cut site (of which the ‘AATTC’ remains in the sequence).



After library preparation, all of these individual samples are pooled together (sometimes 100+) in a single sequencing run. It sounds like a mess, but luckily, we have bioinformatic tools and **regex** to “unpool” or demultiplex individuals, based on the ~8 bp barcode found preceding an ‘AATTC’ in each (well, hopefully most) of the 100s of millions of raw sequences.

Fastq file

Along with the next-generation sequencing and big data revolution, came the **fastq** file format (similar to **fasta**, but with additional lines of sequence quality scores). The raw data files we now receive from sequencing facilities are nearing 100 Gb a piece, necessitating efficient automated processing. You'll want to be familiar with the file format for this tutorial.

```

@FCC638CACXX:5:1105:14309:52971#ATCNCGATC/1
CCAGCTCTAAATTCCAAAGCCTCACC GCATTGACGAAACACAGCCTCGAGGAAGACAAATAGGAATATTTAGGGATTATAAGATTTGTGTGTTTGAAGAC
+
bbbeeeeggggigfhiiiiifiiiiiiiiifgggggeeeededddcc`bccdcccccdcbcccccccccc
@FCC638CACXX:5:1105:14154:53737#ATCNCGATC/1
GCTATTGGACAATTCATTACATGTGGCGGCATGTTACGCATTTGGGATTGAGCGGGTTAGATCGGAAGAGCACACGCTCTGAATCCAGTCACATCACGA
+
[\\^ccccceeeehhh_eaSbKRb^Y^_eeehhhbedeab\\NWbeeehbHV\\aaW^_ZZ\\N^_XXXY^[_WWX^_`GJS]YY]Y`R]^a^B
@FCC638CACXX:5:1105:13834:54314#ATCNCGATC/1
TGCTGCTAAATTCGTTTCGTTTGAATAAACTAAATGCGTTCAACAAAAAATTGATTATTCGTTTATGGAACAACAAAACCGTTTCGAAAATAAAAAATTT
+
bbbeeeeggggigiiihghiiiihiiiiigihiiiiiiiiihfiiiiiiiggggeeeebd`ccccccaccaccaccccccccccc
@FCC638CACXX:5:1105:13834:54314#ATCNCGATC/1
CCTCCAACAATTCGGTTCGATATTTAGGTGGCCCGTTGACATGGAATGCACGATATGTATGTAGTTAGATCGGAAGAGCACACGCTGTAACCTCC
+
bbbeeeecg[bbcefgdeghedffhfhff]b`egghfh_ege]egfdegfbcdgggag]c]bZ^ZacZb]]_]`Z]^_ac^bab^]^ab_Y]^Y_b

```

As you can see above, the `fastq` is organized in 4 line units:

@ Header with sequencing information, but no sample information yet!

sequence data

+

quality scores associated with sequence base pairs

For the purposes of this tutorial, we are going to focus on the first two lines.

VCF file

Biologists handling genomic data, also frequently encounter the `VCF` file format. `VCFs` are used to store SNP (single nucleotide polymorphism) genotypes and associated statistics for many individuals (columns) at many positions (rows) in the genome. Below is a truncated example of SNP data from a RADseq data set. The majority of the header and position statistics have been removed for clarity.

```

##Deleted a large header, all lines starting with ##
#CHROM POS INFO FORMAT CF.A.003 CF.A.004 CF.A.005 CF.A.006 CF.A.007 CF.A.008
Contig23 34 DeletedStuff GT:AD:DP:GQ:PL 0/0:1,0:1:3:0,3,40 0/0:4,0:4:12:0,12,160 0/0:3,0:3:9:0,9,120
Contig150 37 DeletedStuff GT:AD:DP:GQ:PL 0/0:5,0:5:15:0,15,123 0/0:5,0:5:15:0,15,121 0/0:4,0:4:12:0,12,99
Contig234 63 DeletedStuff GT:AD:DP:GQ:PL 0/0:1,0:1:3:0,3,35 0/0:1,0:1:3:0,3,39 0/0:1,0:1:3:0,3,39
Contig286 18 DeletedStuff GT:AD:DP:GQ:PL 0/0:5,0:5:15:0,15,199 0/0:1,0:1:3:0,3,32 0/0:1,0:1:3:0,3,40
Contig286 76 DeletedStuff GT:AD:DP:GQ:PL 0/0:5,0:5:15:0,15,190 0/0:1,0:1:3:0,3,37 0/0:1,0:1:3:0,3,39
Contig286 84 DeletedStuff GT:AD:DP:GQ:PL 0/0:5,0:5:15:0,15,196 0/0:1,0:1:3:0,3,39 0/0:1,0:1:3:0,3,39
Contig319 49 DeletedStuff GT:AD:DP:GQ:PL 0/0:1,0:1:3:0,3,40 0/0:1,0:1:3:0,3,40 ./.:.:.:. 0/0:2
Contig319 64 DeletedStuff GT:AD:DP:GQ:PL 0/0:1,0:1:3:0,3,40 0/0:1,0:1:3:0,3,40 ./.:.:.:. 0/0:2
Contig355 39 DeletedStuff GT:AD:DP:GQ:PL 0/0:2,0:2:6:0,6,55 0/0:2,0:2:6:0,6,55 0/0:3,0:3:9:0,9,83
Contig355 83 DeletedStuff GT:AD:DP:GQ:PL 0/0:2,0:2:6:0,6,55 0/0:2,0:2:6:0,6,55 0/0:3,0:3:9:0,9,83
Contig426 46 DeletedStuff GT:AD:DP:GQ:PL 0/0:2,0:2:6:0,6,55 0/0:2,0:2:6:0,6,55 0/0:1,0:1:3:0,3,28
Contig426 60 DeletedStuff GT:AD:DP:GQ:PL 0/0:2,0:2:6:0,6,55 0/0:2,0:2:6:0,6,55 0/0:1,0:1:3:0,3,28
Contig449 19 DeletedStuff GT:AD:DP:GQ:PL 0/0:5,0:5:15:0,15,194 0/0:3,0:3:9:0,9,120 0/0:4,0:4:12:0,12,160
Contig454 2 DeletedStuff GT:AD:DP:GQ:PL 0/0:1,0:1:3:0,3,37 ./.:.:.:. 0/0:1,0:1:3:0,3,40 0/0:5
Contig454 58 DeletedStuff GT:AD:DP:GQ:PL 0/0:1,0:1:3:0,3,37 ./.:.:.:. 0/0:1,0:1:3:0,3,40 0/0:5
Contig500 27 DeletedStuff GT:AD:DP:GQ:PL 0/0:2,0:2:6:0,6,80 0/0:3,0:3:9:0,9,107 0/0:2,0:2:6:0,6,80
Contig538 36 DeletedStuff GT:AD:DP:GQ:PL ./.:.:.:. 0/0:2,0:2:6:0,6,80 0/0:5,0:5:15:0,15,200 0/0:2
Contig588 70 DeletedStuff GT:AD:DP:GQ:PL 0/0:1,0:1:3:0,3,39 0/0:1,0:1:3:0,3,39 0/0:2,0:2:6:0,6,77
Contig588 78 DeletedStuff GT:AD:DP:GQ:PL 0/0:1,0:1:3:0,3,39 0/0:1,0:1:3:0,3,39 0/0:2,0:2:6:0,6,77
Contig588 84 DeletedStuff GT:AD:DP:GQ:PL 0/0:1,0:1:3:0,3,39 0/0:1,0:1:3:0,3,39 0/0:2,0:2:6:0,6,77

```

This data set included biallelic SNPs only and the data for each individual and position are listed in the following format:

genotype : read count per allele : total read depth : genotype quality : genotype likelihoods

For the purposes of this tutorial, don't worry about understanding each field, just the **pattern** in which they are arranged.

Writing files line by line

When dealing with very large data files, you can easily slow down your program by storing too much information in memory. Therefore, whenever possible, write what you can to an output file frequently. For example:

```
file.create(filename)
for (each step in your loop){
  get or create string
  write(string,file=filename,append=TRUE)}
```

Approximating a dictionary

Other programming languages have a structure called a dictionary (an unordered set of keys and associated values) which is useful for storing and quickly retrieving information. R does not have an analagous tool, but other structures (e.g. lists) can be adapted to perform similar functions. For this tutorial, a two column data frame will be standing in for a dictionary, column 1 as the “key” and 2 as the “value”. You can check whether your “key” is in the dictionary with:

```
key %in% dictionary[,1]
```

This returns TRUE or FALSE. You can retrieve the “value” associated with a “key” by:

```
dictionary[dictionary[,1]==key,2]
```

Challenge

1. For this challenge, you'll be performing two string manipulations on the modified VCF file `Cflorida.vcf`. This file contains the SNP information for two populations of *Rhagoletis* flies collected from flowering dogwood (*Cornus florida*) fruits in Steven F. Austin Forest, TX and Gainesville, FL. Unfortunatley, all collaborators did not follow a standardized naming convention, resulting in individual samples from TX named:

CF.A.XXX

CF.A2.XXX

CF07.A.XXX

cf.a.XXX

and individuals from FL named:

CF.G2.XXX

CF.GAI.XXX

cf.gai.XXX

with XXX representing 3 digit IDs.

Your first objective is to use regular expressions to standardize all TX samples to `Cf.Sfa.XXX` and all FL samples to `Cf.Gai.XXX`.

The VCF file also contains many different fields of information for each individual and SNP, as explained above. Today, we are interested only in the allele counts (number of reference and alternate alleles sequenced for each individual), bolded in the example below:

0/0:**7,0**:7:21:0,21,194

Your second objective is to use regular expressions to extract only the allele counts for each SNP for each individual (e.g. 7,0). If the data are missing, indicated by ., replace with NA.

The goal is to create a new file `CfloridaCounts.txt`, with the same structure as `Cflorida.vcf`, but with corrected sample names and allele counts only.

A couple tips on this one:

Combine these objectives into one for loop over `Cflorida.vcf`.

Work out your regex on paper first.

2. For this challenge, you'll be implementing a simple version of a demultiplexing program for RADseq data (explained above). You will start with two files `indivIDs.txt` and `seqFastq.fq`:

*`indivIDs.txt` is tab-delimited with 8 bp DNA "barcodes" in the left column and the associated sample ID in the right column

*`seqFastq.fq` is a fastq file (described above), containing raw sequence data that lacks associated sample IDs.

Your objective is to write a fasta file, `IDseq.fasta`. The header line should contain the sample ID (e.g. > ID) and the sequence line should contain the associated DNA sequence from the fastq file, with the "barcode" and AATTC cut site removed. Also, plot a histogram of the start position of the AATTC cut site match in each barcoded sequence.

A couple tips on this one:

Break this down into a few chunks for easier management:

- a. Store the sample IDs in a data frame, DNA “barcodes” will be column 1 (“keys”) and the sample IDs column 2 (“values”).
- b. Write a regular expression to match the 8 bp barcode, followed by the AATTC cut site, followed by the rest of the sequence. Note that you will need to reference each of these pieces later on.
- c. Use a while loop to scan the fastq file line by line, searching each sequence line for the regular expression. Then:
 - * Check the sample ID “dictionary” for the 8 bp barcode
 - * Store the position of the cutsite match
 - * Write the header with sample ID to the fasta file
 - * Write the trimmed sequence to the fasta file

Devise a plan for splitting up the work and generating the required code. Do this in parallel, not sequentially. Don’t forget to check and edit each other’s code. Remember to frequently **add-commit** locally and **push-pull** to GitHub to avoid conflicts. Also, remember you don’t have to be in the same place at the same time to work on this collaboratively thanks to GitHub!!!

Turning in your assignment via GitHub

Once you have committed all changes to your local Git repos and pushed all of those commits to the forked repo on GitHub, you can “turn in” your assignment using a **pull request**. This can be done from the GitHub repo website. When viewing the forked repo, select “Pull requests” in the upper middle of the screen, then click the green “New pull request” button in the upper right. You’ll then see a screen with a history of commits for you and your collaborator, select the green “Create pull request button”. In the text box next to your user icon near the top of the page, remove whatever text is there and add “owner’s last name - collaborator’s last name submission”, but obviously substitute your last names. If I and Ann Raiho worked on the project together the text would read “jones-raiho submission”. Then click the green “Create pull request” button. **Only one of you will need to create a pull request.**