Pattern Matching and For Loops

Nick Bartelo

1/28/2021

Q1

1. Extract the barcode sequences for each read in demo.fastq. In this file, the barcode is the last set of alphabetic characters in the sequence identifier line. How many unique barcodes are there? List them. (3pts). Hint: Use egrep, sort, uniq. Read the man pages for egrep, to identify useful options. Use the pipe (|) to chain commands. You can use multiple egrep commands.

To begin, we ssh into aphrodite using `ssh nib4003@aphrodite.med.cornell.edu`. We then ssh into curie using `ssh curie.pbtech`. We have a folder here entitled "ANGSD_2021_hw". Therefore, we use `cd ANGSD_2021_hw` to move into this folder and create a new directory here for this homework using the command `mkdir pattern_matching_and_for_loops`. Next, we cd into this directory using `cd pattern_matching_and_for_loops`. We now copy the demo.fastq file from /home/luce/angsd/demo-data to the new directory using the command `cp /home/luce/angsd/demo-data/demo.fastq .` where cp is the command to copy a document, the path provided leads to the document, and the '.' at the end means to put the file into the current directory.

To figure out how many barcodes there are in the fastq file, we first define what the barcode is. An example of an entry for the fastq is shown below:

@NB501971:154:H3LLMBGX7:1:11101:6765:1063 1:N:0:GAGTGG CGGCTNGTCTTAAACGTCTTTAAG-
GTGCTTGAGAGGATGAAGGTGGTGAAAAAGAGAATACAGGACCAAAAGAGGACATCAGGAG +
AA/AA#EEEE/EEEE//EAA/E6//EAE/EEEEEA/E///E6///EEEE/E66AEEEE/AEEA/AEEEE/E/<EEEEAEEAEEEE

The barcode is identified as the 6 letters at the end of the first line (GAGTGG). We can therefore use the fact that our barcodes are found in lines that begin with '@'. As a result, we can use the command `egrep '@' demo.fastq` to return all files with the barcodes in them.

Next, we need to keep only the barcodes, which represent the last 6 letters of each line resulting from the above command. Therefore, we pipe a new command which keeps only the last 6 letters, representing each barcode, given by `egrep '@' demo.fastq | egrep -o '......$'` where `-o` is used only to print the matching string and `'......$'` represents 6 wildcard characters (can be any character) with the `$` sign specifying to look for the pattern at the end of the line, precisely where the barcode is.

Our next task is to finally find the unique barcodes so that we can count how many there are. In order to do so, we must first sort the barcodes because the unique command looks at adjacent rows. As a result, if there are two of the same barcodes that are not one on top of the other in the file, they will each be considered unique. To remedy this situation, we add the command `sort` to our previous commands which groups all barcodes together. Finally, we can receive all unique barcodes by adding the `uniq` command. Our final code to output all unique barcodes is given by `egrep '@' demo.fastq | egrep -o '......$' | sort | uniq`.

To answer the question we have been asked, we can add `wc -l` to our command to count the number of lines in our above command so that the final command is `egrep '@' demo.fastq | egrep -o '......$' | sort | uniq | wc -l` where `wc -l` counts the number of lines in the output, and therefore the number of unique barcodes, which is 20. For fun, we can also see how many occurrences of each barcode are in the fastq file. The command used for this is `egrep '@' demo.fastq | egrep -o '......$' | sort | uniq`

`-c` where `-c` counts the number of times each unique barcode is found. Below we output a table of the results for each barcode.

Count	Unique Barcode
12	AAGTGG
36	CAGTGG
5	GAATGG
64	GACTGG
45	GAGAGG
6	GAGCGG
50	GAGGGG
6	GAGTAG
59	GAGTCG
27	GAGTGA
76	GAGTGC
24260	GAGTGG
36	GAGTGT
44	GAGTTG
53	GATTGG
39	GCGTGG
73	GGGTGG
50	GTGTGG
20	NAGTGG
39	TAGTGG

Q2

2. Copy the PDB files from /home/luce/angsd/demo-data. Write a for loop that will loop over each file, print the name of the file being processed, print the number of lines in the file, print the number of atoms in the file, and print the source organism(s). (3pts, with bonus points for making the output as concise and readable as possible). Hint: Use echo, cat, cut to include only relevant information.

First, we need to copy all the PDB files. To do this, we make sure we are in our pattern_matching_and_for_loops directory and then use the command `cp /home/luce/angsd/demo-data/*.pdb .` which copies all the files to the directory. The *.pdb represents a wildcard so that all files that end in .pdb are copied from the directory. This is powerful when many files need to be transferred at once.

Next, we look at all files briefly using `less file` to find the source organism(s) in each file. We find that this step is extremely important because if we had only looked at a subset, we may conclude that all organisms could be found by using `egrep 'ORGANISM'` since this is found in all but one file,namely 1AAP.pdb. Therefore, we have demonstrated the necessity to thoroughly inspect the parts of the files that we are interested so as to not make any mistakes when searching for our strings of interest. We notice that we can output all organisms by first using `egrep` to find all rows that start with 'SOURCE', followed by searching 'ORGANISM' for all files but 1AAP.pdb, and '$' for 1AAP.pdb.

The following outputs the name of the file being processed, the number of lines in the file, the number of atoms in the file, and the source organism(s): `for file in *.pdb; do echo $file; cat $file | wc -l; egrep -c '^ATOM' $file; egrep '^SOURCE' $file | egrep '\$' | awk -F"[()]" '{print $2}' | sed 's/\$//'; egrep '^SOURCE' $file | egrep 'ORGANISM' | cut --complement -d ":" -f 1 | sed 's/;//' | sort | uniq; done`. This outputs lines in the order of filename, number of lines in the file, number of atoms in the file, common and scientific names of source organism(s).

Now we dissect all the commands we used in the for loop. The `;` symbol breaks up the specific functions desired in the for loop. `for file in *.pdb` looks at all files with a .pdb extension. `do echo $file` prints the name of each file when it is called upon in the for loop. `cat $file | wc -l` returns the number of lines

in the specific file being examined at that time. `egrep -c '^ATOM' $file` searches for all occurrences in a file where the start of a line is "ATOM" and outputs the number of lines, giving the number of atoms. Next we needed to find the name of the source organism(s).

We did this differently for one of the files, 1AAP.pdb, because it had a different format than all others. `egrep '^SOURCE' $file | egrep '\$' | awk -F"[()]" '{print $2}' | sed 's/\$//'` first looks at all lines in the file that begin with "SOURCE". Next, we noticed by inspection of the file that the source organisms contain a "$" symbol. Therefore, we keep only the lines that contain this symbol using `egrep '\$'`. The resulting lines contained the source organisms in parentheses, and therefore we used the `awk -F"[()]" '{print $2}'` command, which extracts only what is inside of the parentheses of a line. Since the names of the source organisms contained a "$" sign in them, we finally cut out this symbol using `sed 's/\$//'`.

For all other files, we used the commands `egrep '^SOURCE' $file | egrep 'ORGANISM' | cut --complement -d ":" -f 1 | sed 's/;//' | sort | uniq` to extract the sample organism(s). We again looked at the lines that began with "SOURCE" using `egrep '^SOURCE' $file`. For these files we were then able to find all lines containing the source organism(s) by searching for 'ORGANISM' using `egrep 'ORGANISM'`. The following command `cut --complement -d ":" -f 1` cuts a line apart given the delimiter ":" in our case, and `--complement` keeps only the characters following the delimiter. The `-f 1` command keeps the information after the ":" delimiter. Some of the source organisms still contain a ";" symbol at the end of the line. The command `sed 's/;//'` deletes this symbol, leaving only the organisms names. We found that there were some files which repeated an organism. Therefore, we used the `sort` command to order the source organisms in each file and the `uniq` command to drop all duplicate source organisms.

We then have to end the for loop with `done`. After this code is run, we receive the information desired, shown in the table below.

Filename	Number of Lines	Number of Atoms	Scientific Name of Source Organism(s)	Common Name of Source Organism(s)
1A3B.pdb	2983	2325	HIRUDO MEDICINALIS, HOMO SAPIENS	MEDICINAL LEECH, HUMAN
1AAP.pdb	1068	866	HOMO SAPIENS, ESCHERICHIA COLI	NA, NA
1BTH.pdb	6359	5812	BOS TAURUS	BOVINE
1DZI.pdb	2895	1714	HOMO SAPIENS	NA
1M3D.pdb	23352	20797	BOS TAURUS	BOVINE
1NMJ.pdb	543	427	RATTUS NORVEGICUS	RAT
1O91.pdb	4057	3120	MUS MUSCULUS	MOUSE
1TAW.pdb	2520	2044	BOS TAURUS, HOMO SAPIENS	BOVINE, HUMAN
1Y0F.pdb	4259	2800	RATTUS NORVEGICUS	RAT