Programming for Bioinformatics | BIOL 7200

Week 3 Exercise

September 5, 2023

Instructions for submission

- Run a Linux or Mac terminal on your computer
- You may want to create a directory to work in (e.g., ”~/biol7200/class3/ex3”)
- Download "week3_data.tar.gz" and extract the contents into your working directory (e.g., ”~/biol7200/class3/ex3”)
- Note, extracting the provided tar.gz file will take a minute or two as it contains many files.
- Prepare a file containing answers to the below questions. Copy the question and either paste the body of a script, write the correct answers, or provide a screenshot of your working below the question as directed in the instructions provided in the question.
- In cases where you need to paste a screenshot of your terminal, you can do that as follows. Screenshot a selected area with Windows: win+shift+S, Mac cmd+shift+4 and then paste in your submission sheet
- Use the `clear` command to tidy up your terminal so you show only relevant commands and outputs
- Name your submission sheet: ”gtusername.pdf”, or ”gtusername.docx”

Grading Rubric

This assignment will be graded out of 100.

Each question includes the points available in parentheses.

Questions

1. (5 points)

   To acquire a dataset for a later question, I downloaded all the histidine kinase domains of proteins found in Escherichia coli strain K-12 from the Uniprot database. This dataset is dicsussed in the relevant question below. The sequences came with long headers including lots of extranious information. Fortunately, we can quickly tidy them up using regex so you can use them later.

   Using a regex with `sed`, create a copy of HK_domains.faa in which you replace fasta headers of HK_domain.faa file with just the gene name after GN= in the existing headers and the position ranges.

   E.g., the first header is currently

   >sp|P09835|UHPB_ECOLI|311-499 Signal transduction histidine-protein kinase/phosphatase UhpB OS=Escherichia coli (strain K12) OX=83333 GN=uhpB PE=1 SV=3

   the new header should be

   >uhpB_311-499

   Provide a screenshot showing the following commands executed in the same terminal window:

- `head -1 <original file>`
- Your command that converts the headers and outputs to a new file
- `head -1 <new file>`

2. (5 points)

   While the new headers are better than what we had before, the above headers are not good enough because HK_domains.faa contains protein sequence. Bacterial protein names start with an uppercase letter, while gene names start with a lowercase letter. Create a new file the fasta headers with the protein names instead. Protein names are present once on each line and bacterial protein names are of the format AbcD (1 uppercase letter, two lowercase letters, one upppercase letter).

   Create a new copy of HK_domains.faa in which you replace the headers with just the protein name and amino acid position range.

   e.g., the first header is currently

   >sp|P09835|UHPB_ECOLI|311-499 Signal transduction histidine-protein kinase/phosphatase UhpB OS=Escherichia coli (strain K12) OX=83333 GN=uhpB PE=1 SV=3

   the new header should be

   >UhpB_311-499

   Provide a screenshot showing the following executed in the same terminal window:

   - `head -1 <original file>`
   - Your command that converts the headers and outputs to a new file
   - `head -1 <new file>`

3. Extra credit (5 points)

   Make another copy of HK_domains.faa where you remake the headers as in question 1 (i.e., get the gene name from next to "GN="), but modify the regex replace command so that it converts the first character of the gene name to uppercase. i.e., the output of the command should be identical to that of question 2, but you should do it with a regex that matches the same thing that you matched in question 1.

   Provide a screenshot showing the following executed in the same terminal window:

   - `head -1 <original file>`
   - Your command that converts the headers and outputs to a new file
   - `head -1 <new file>`

4. (5 points)

   The next two questions will use a regex to extract information from a dendrogram represented in Newick format. Below is a description of what Newick format is. If you are already familiar with Newick format, you can skip to the **Task** section below.

   Dendrograms are a common way to visualize relationships among samples. They can be used whenever trying to visualize many pairwise comparisons and are commonly used in combination with heatmaps.

Today we will be working with a dendrogram that is used to show the evolutionary history (phylogenetic relationships) of organisms.

When a dendrogram is used to visualize the phylogenetic relationship between organisms, it is referred to as a phylogenetic tree. However, note that not all dendrograms are phylogenetic trees. Only those that represent a hypothesis of phylogenetic relationships among organisms.

A common format in which dendrograms are encoded is Newick format. Newick format is simpler than another dendrogram format, Nexus, which offers the ability to enocde more information about the dendrogram. However, Newick format is able to enocde all the core information you need to draw the encodeded dendrogram.

Newick format is described by one of its formulators here. I also describe it below.

Newick has the following core format to describe a dendrogram.

(A,B),C;

"A", "B", and "C" in the above Newick string are node IDs. The nodes represented by letters in this string are "terminal nodes", which are commonly called leaves or leaf nodes. In addition to leaf nodes there are internal nodes which are the branch points in the dendrgram. These typically don't have node IDs (though they can). Parentheses are used to indicate branching patterns within the tree. Each node within the dendrogram is connected to its parent or child nodes by a line called a branch (or edge). Each branch is separated with a ",". Newick strings are typically ended with a ";" character (although this isn't always enforced)

By reading the above Newick string, one can draw the encoded dendrogram which looks like this:

```
A----\
     |----\
B----/    |
          |
C---------/
```

I encourage you to experiment with made up newick strings on IToL to get familiar with Newick format and with dendrograms. IToL supports copying or typing in a newick string and will visualize the encoded dendrogram. Note for example what difference it makes to remove the parentheses.

In addition to the core format which contains the minimal information required to draw a branching pattern, Newick format can also encode information about branches. Specifically, the length of the branches (which typically indicates the difference between the connected nodes), and support for internal nodes (which typically indicates a measure of confidence in the subtended node - the child nodes and all children of those child nodes.)

Every node can have a branch length, but only internal nodes have support values (you can always be certain that a leaf node should be grouped with itself!) Branch lengths and node support could be encoded in our example Newick as follows

(A:8,B:5)50:2,C:4;

In this example, each node has a branch length which follows a ":" character. Only one node has a support value (50), which is between a ")" and a ":" character. As an aside, it is not possible to interpret support values without knowing how they were calculated (e.g., bootstrapping). However, a value of 50 probably means we aren't too sure about this node and A and C or B and C might go together instead of A and B.

Below is a portion of the newick string from our question's data. It shows the relationship between chickens, humans, chimpanzees, mice, and rats. Make sure you understand what all the components are before trying to answer these questions. You may find it helpful to load this Newick string on IToL.

(Gallus_gallus:0.04596,((Homo_sapiens:0.00957,Pan_troglodytes:0.03864)100:0.01549,
(Rattus_norvegicus:0.03107,Mus_musculus:0.01651)91:0.00398)99:0.01629)

Task

Newick leaf names can contain any characters except "(", ")", ",", ":", and ";". Write a regex that would match **ANY** allowed leaf name in a newick string. Use your regex and `grep` to extract all leaf names from the Tree_of_life.nwk in the data directory.

Note that leaf names can be numbers so consider what about newick format is used to distinguish leaf names from other elements. Your regex should be able to identify the leaf names "0.5" and "0.000123" in the newick string "(0.5:0.4,0.000123:0.123);"

Provide a screenshot showing the following executed in the same terminal window:

- Your command to identify all leaves piped to `wc -l` to count the results (i.e., `<your command> | wc -l`)
- Your command to identify all leaves piped to `head` to show a snippet of the results (i.e., `<your command> | head`)

5. (5 points)

   Use a regex and `grep` to extract all branch lengths from the Tree_of_life.nwk file and then calculate the sum of all branch lengths using `awk`.

   Note again that leaf names can be numbers. Your regex should match only branch lengths and not node support or leaf names.

   Provide a screenshot showing the command and its result in a terminal window

6. (4 points)

   As mentioned above, the naming convention for bacterial gene names is that they begin with a lowercase letter, while protein names begin with an uppercase letter. (quick aside, bacterial gene names are also italicized to increase the visual distinction, but that's not relevant here as all our text is unformatted.) I have provided you with a dataset of badly organized fasta files. Within the week3_data.tar.gz archive is a directory "data/find_data/" which contains most of the genes and proteins of *Escherichia coli* strain K-12. Normally it is good practice to distinguish gene and protein sequences using fasta extensions like .fna (fasta nucleic acid) and .faa (fasta amino acid). However, whoever generated these files used a random selection of the generic extensions .fasta and .fa. As there is no fixed string that is common to all gene files or all protein files, globs are of no use. However, the filenames are named according to the bacterial gene and protein name convention (genes start with lowercase; proteins start with uppercase). Therefore a regex can be used to select just the gene or protein files.

Use `find` with a regular expression and `wc -l` to determine how many gene files are in the directory "data/find_data/". Note use "egrep" regextype as that refers to the GNU ERE covered in class.

Provide a screenshot of a terminal window showing your command and the output

7. (4 points)

   Use `find` with a regular expression and `wc -l` to determine how many protein files are in the directory "data/find_data/".

   Provide a screenshot of a terminal window showing your command and the output

8. (2 points)

   Use `find` with a regular expression and `-exec` to copy all genes to a new directory called "genes/" and another `find` command to copy all proteins to a new directory called "proteins/"

   Provide a screenshot showing the following executed in the same terminal window:

   - Your command to copy all gene files to "genes/"
   - Your command to copy all protein files to "proteins/"
   - The output of the command `ls genes/ | wc -l; ls proteins/ | wc -l`

9. Setting up and forking a git repo (30 points)

   Setup

   For this question we will be creating and using a github account. If you are already familiar with Git and have either a github or gitlab account set up then you can proceed to the "Tasks" section below. Otherwise please follow the instructions here to set up a github account and your local `git` installation.

   1. Create an account on github.com

   2. Create a personal access token by following the instruction on the github help pages section at this link. Set the expiration of the token to be at least 90 days from now so it will last for the duration of this class (you can choose "no expiration" and delete the token manually later if you wish). Set the scope of the token to be "repo" (this should also check all boxes in the repo section of the page). Save this token in a text file for use later in these instructions. This token is like a password so don't share it anywhere public or others will be able to modify your repos. If you ever think a token has been accessed by someone else, delete the token immediately on your github account.

   3. If you don't already have git installed (i.e., `which git` returns nothing) then install it by following the instructions on this page. Note that students using WSL should install using the Ubuntu instructions.

   4. Configure your git installation so that you can interact with your github account repositories by running the following commands (substituting the placeholders with your data):

- `git config --global user.email "<your email>"`
- `git config --global user.name "<your name>"`
- `git config --global credential.helper store`

The last command will tell git to save your token after the first time you use it so you don't have to enter it again. Don't run this if you would prefer to enter your token each time.

Tasks

1. (6 points)

   Make a git repo on your github account called "perfect_hits". OPTIONAL: You may wish to create a README.md when prompted in which you can document the usage of the script in this repo, but your README will not be assessed.

   Clone the repo to a location on your computer (perhaps ~/git_repos/perfect_hits/).

   Provide a screenshot of your terminal in which you execute the clone command.

2. (6 points)

   Copy your `find_perfect_matches.sh` script from last week (Question 4.3) into the new repo.

   Add and commit your script so it is tracked by git.

   Provide a screenshot of your terminal in which you executed the copy, add, and commit commands successfully.

3. (6 points)

   Make and activate a conda/mamba environment with blast installed. Activate an existing one if you already have one.

   Make a hardlink of your `find_perfect_matches.sh` script in the mamba env bin folder for the activated environment.

   Provide a screenshot in which you show you can now call your script with `which <script>` while in the mamba env

4. (6 points)

   Replace the contents of the `find_perfect_matches.sh` script in your repo with `echo "oops!"`. Save the file.

   Run the `find_perfect_matches.sh` that is on your path (not the file that is in your git repo).

   Provide a screenshot of the output of running `find_perfect_matches.sh` from your PATH.

   Is the hardlinked file that you put in the mamba env bin directory also modified? Explain?

5. (6 points)

   Revert the changes to the script in your git repo by running `git checkout .`.

The above command changed every file in the repo to the version at the HEAD of your git history. What does HEAD refer to in this context?

If you wanted to revert to a different point in your git history (e.g., a specific commit) how would you do that?

10. find_homologs.sh (40 points)

It is rare that you will be looking for perfect hits in assemblies. A strength of BLAST is the speed with which it can identify approximate matches. One case in which you may wish to identify approximate matches is when searching for homologous genes in different organisms. See this week's recommended reading "intro_to_sequence_similarity_searching.pdf" for an introduction to using sequence similarity to identify homologous genes.

For this exercise you have been provided with the sequence of histidine kinase (HK) domains from the organism *Escherichia coli* strain K-12. HK domains are perhaps the most widely used of all signalling domains. In bacteria, HK domains are commonly involved in two-component signalling systems. Two-component systems involve two proteins (hence two-component) that together orchestrate a response (usually) to a specific environmental signal.

Signalling is acheived through activation of the "sensor histidine kinase" protein (which contains the HK domain) upon sensing a specific environmental stimulus. The activated sensor histidine kinase then in turn activates its partner protein called a response regulator. The response regulator (as the name suggests) then regulates some sort of response, often by impacting gene expression.

Understanding the environmental signals that an organism is sensetive to can provide a lot of information about the kinds (and number) of environments that organism encounters. When sequencing a novel organism, one of the first steps in analyzing the sequence is to identify the kind of genes that are present. When you have information about the kinds of genes an organism encodes, you can order to infer things about the organisms' biology.

Tasks

1. Create a copy on github of your perfect_hits repo created in question 9.1. To do this you can click the "+" at the top right of the page, choose import repository, and then paste the URL of your perfect_hits repo in the "old repository clone URL" box. Name your copy something like "find_homologs"

2. Clone your new repo to a location on your computer

3. rename the "find_perfect_matches.sh" script to be "find_homologs.sh" using `git mv` and commit the change

4. Using git to track changes to your script (by adding and committing them), modify the script so that it performs the following function:

   - Given a protein sequence query, perform a blast search against a nucleotide subject (This website should help determine which BLAST program to use)
   - Filter hits to keep only hits with >30% sequence identity and >90% match length (90% of the query sequence length)
   - Output matches to a specified file

- Print the number of matches identified

Usage of your script should be `./find_homologs.sh <query file> <subject file> <output file>`

In your submission sheet include the following:

- The content of your find_homologs.sh script
- A screenshot of your terminal showing the last 10 commits with one commit per line (i.e., not the default output) using the command `git log -n 10 --oneline`. Note if you made fewer than 10 commits that is acceptable. You need only show that you made commits.
- A screenshot of your terminal showing the output of your script when used to identify homologs of the "HK_domains.faa" sequences in each of the four bacterial assemblies provided in the "data/" dir of the "week3_data.tar.gz" archive ("Escherichia_coli_K12.fna", "Pseudomonas_aeruginosa_UCBPP-PA14.fna", "Vibrio_cholerae_N16961.fna", and "Wolbachia.fna")

11. Extra credit (10 points)

Preface: This question will be hard as I expect you will solve it using logic that will be unfamiliar to anyone who has not used loops before (there may be non-loop solutions I haven't considered). If you do not see a solution to this problem now, consider revisiting it after we cover loops in class to see if you can see a solution at that point.

Last week's assignment included an exercise in which you wrote a script that identified perfect matches found using BLAST. We used that script to identify which of two bacterial genome assemblies encoded CRISPR repeats. In addition to identifying which of the assemblies encodes CRISPR arrays, the script identified where in the assembly those arrays are located.

CRISPR immunity typically works by sequence-specific degradation of invading DNA or RNA. However, the repeat sequence is not the sequence that is used to identify target sequences to degrade. Instead, the sequence in between each repeat (called spacers because they separate repeats) is the sequence that encodes the immune memory of CRISPR systems.

The find_perfect_matches.sh script identifies the location of repeat sequences in an assembly. In doing so, it also identifies the location of sequences between each repeat - the spacers. A common analysis performed by CRISPR researchers is to extract spacer sequences and BLAST them against a database of viruses and plasmids to predict immune targets.

You task is to create a new branch in your perfect_hits repo (created in question 9.1) and develop a script that builds on find_perfect_matches.sh by adding functionality to retrieve spacer sequences from the provided assembly. The script should have the same usage as the find_perfect_matches.sh script written last week (although you may change the name using `git mv` if you like, perhaps to something like "extract_spacers.sh")

You should commit changes as you develop the script so that you can report the git log as part of your solution. You may find this seqtk function helpful for extracting the spacer sequences from the assembly. That function takes as input an assembly file and a BED file containing the regions you want to extract. You saw a BED file in the first assignment. In addition, you can find a description of the format here. Note that only the 3 required fields are needed by this function. You can use something other than seqtk if you prefer.

Watch out for the start/stop position of repeats vs start/stop position of spacers. Keep in mind that your repeats all have the same sequence but spacers are almost always different sequence. If your spacers all start or end with the same base then you may have made a mistake. Also be careful that you are extracting the whole spacer.

A hint is included below the description of how to submit your response. Don't read the hint if you would rather ponder the problem without.

In your submission sheet include the following:

- The content of your script
- An exported yaml of a mamba environment that can run your script (ideally containing only the dependencies of your script)
- A screenshot of a terminal in which you show the following using issued commands:
  - The branches in your git repo showing that you created and used a new branch
  - the last 10 commits in your git history using the command `git log -n 10 --oneline`
  - The execution of your script on the assembly you determined encodes a CRISPR array
  - The contents of the output file containing the identified spacers

If you are interested to learn more about CRISPR systems, this review offers an excellent overview of the biology as well as a brief history of the discovery of CRISPR systems and the prediction of their function by bioinformaticians. To be clear: this reading is not assigned. It is well outside the scope of the class and is only offered here for anyone who is interested in learning more about the biology behind these exercise questions.

Hint: you can create the bed file using an awk script. Bear in mind that awk code is executed from left to right, so any code to the left of where a variable is assigned will be using the value assigned to that variable on the previous line of the file being processed.