1. Let’s use git!
2. Use the tutorial <https://try.github.io/levels/1/challenges/1> to get a feel for how it works
3. Install git. On a mac, use homebrew. On Linux/Windows 10, use apt-get.
4. Create a profile on github.com. If you’re affiliated with a university, you can get a free upgraded plan. You should do that.
5. Create a repository for this assignment. I’m going to expect assignments to be submitted as github repos for this and all future assignments. If you create the repo online, it gives you all the instructions you need to link your local repo and the online repo.
6. Implement the naïve string searching algorithm
7. Use wget to download a genome from <https://www.dropbox.com/s/u7ws7yavym56emr/my_genome.fa> and some reads from <https://www.dropbox.com/s/gjac55m5metseca/my_reads.fastq>. These are fake data
8. Write a function to read the fasta file. It will be a little more complicated than last time, because each chromosome spans multiple lines (which is common).
9. Write a function to compute the REVERSE COMPLEMENT of the chromosomes, because reads can come from either the + or the – strand
10. Write a function that takes two arguments: a long string (i.e. the sequence of a chromosome) and a short string (i..e. a read) and returns the first position in the long string where the short string occurs. If it doesn’t occur at all, return None (“None” is a specific type in Python, so I mean to literally “return None” if you don’t find the string)
11. Map the reads! Output a file where each line consists of a read name, the chromosome it mapped to, the strand it mapped to (i.e. + if it mapped to the regular strand, - if it mapped to the reverse complement) and the position that it mapped to.
12. When you run the script, use the “time” function in bash to see how long it takes. i.e. “time python my\_script.py” will say how long my\_script.py took to run.
13. Implement the hashing mapping algorithm
14. Use the same dataset as before, but this time do a preprocessing step in which you generate a dictionary that holds all the possible reads that could have come from the genome.
15. Use the dictionary to map the reads
16. Time it!
17. What’s the big O time complexity of this algorithm?
18. Just like time complexity, there’s also space complexity. In this version of the algorithm, you have to create a dictionary with all the possible reads. What’s the big O space complexity of this algorithm?