# Programming for Bioinformatics | BIOL7200

## Week 4 Exercise

September 14, 2021

### Exercises

Regardless of your specific areas of research, file manipulations techniques will always be useful. If there is only one thing that you can learn from this class, let it be this. We’ll be happy to go over any concept that you are still struggling with in Thursday’s discussion session.

#### xargs

1. Passing arguments with xargs
   1. List the files in the directory, and pass them to the head command using xargs and the -I flag

**ANS: ls | xargs -I head**

* 1. Use xargs and touch to make 10 files by the name of: file1.fna, file2.fna, …, file10.fna

**ANS: seq 1 10 | xargs -I X touch fileX.fna**

* 1. Use sed and xargs to change the extension of the all the .fna files to .fasta

ANS: ls \*.fna |sed 's/.fna//' |xargs -I ASH mv ASH.fna ASH.fasta

* 1. Get the first two columns of the UCSC gene file using cut

**ANS: cut -f1,2 knownGene.txt**

1. Multiple arguments with xargs
   1. Print the numbers 1 to 12

**ANS: echo 1 2 3 4 5 6 7 8 9 10 11 12 | xargs**

* 1. Pass these to xargs in multiple of 3 and print them as:

ANS: **seq 1 12 | xargs -n3 sh -c 'echo "First number: $1; second number: $2; third number: $3"' sh**

First number: 1; second number 2; third number 3;

First number: 4; second number 5; third number 6;

First number: 7; second number 8; third number 9;

First number: 10; second number 11; third number 12;

Text

Description automatically generated

#### General programming

1. What is the value of a\_number after each line? (Write the value of a\_number after each statement; these statements are in continuation)
   1. a\_number = 6

ANS: 6

* 1. a\_number -= 12

ANS: -6

* 1. a\_number += 15

ANS: 9

* 1. a\_number /= 3

ANS: 3.0

* 1. a\_number \*= 2

ANS: 6.0

* 1. a\_number = a\_number + a\_number / 2

ANS: 9.0

1. Evaluate these Boolean statements. (Answer is TRUE or FALSE)
   1. 1 > 2

ANS: FALSE

* 1. 2 > 1

ANS: TRUE

* 1. !(2 > 1)

ANS: FALSE

* 1. (2 > 1) || (1 > 2)

ANS: TRUE

* 1. (2 > 1) && (1 > 2)

ANS: FALSE

* 1. !(2 > 1) || !(1 > 2)

ANS: TRUE

* 1. !((2 > 1) || (1 > 2))

ANS: FALSE

* 1. !((2 > 1) && (1 > 2))

ANS: TRUE

#### Biologically-inspired problem

1. Format recognition

A large part of data analysis deals with cleaning and processing data. There are a lot of data formats – some of which might just be different formal standards for storing the same data. Given the large number of formats that exist, it is almost impossible to remember each of them – feel free to look up their specifications when you need to use them.

We will deal with four standard (and common) bioinformatics file formats this week: EMBL, FASTQ, GenBank, and MEGA. Here are their specifications:

Format and link to description

* EMBL  
  [https://www.genomatix.de/online\_help/help/sequence\_formats.html#EMBL](https://www.genomatix.de/online_help/help/sequence_formats.html%23EMBL)
* FASTQ  
  [https://www.genomatix.de/online\_help/help/sequence\_formats.html#FASTQ](https://www.genomatix.de/online_help/help/sequence_formats.html%23FASTQ)
* GenBank  
  <https://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html>
* MEGA  
  <https://www.megasoftware.net/mega1_manual/DataFormat.html>

Let’s write simple awk one liners to check the format for input files. Only the first line of the input file will be piped into your awk one-liner.

Notes:

* You can send the first line using the head command. Formats can often be recognized by using only the first column.
* The solution can be very clean and simple to something extremely elaborate. It all depends on how you think.
* There are files provided on Canvas for you to check your one-liner.

**ANS: head -1 file.fastq | awk '{if ($1 == "ID") {print "This is an EMBL file"} else if ($1 == "LOCUS") {print "This is a Genbank file"} else if ($1~/#/) {print "This is a MEGA file"} else if ($1~/@/) {print "This is a fastq file"} else {print "Type of file not known"}}'**

Text

Description automatically generated

#### Final installation problems

This will be the final installation problem for this course. These two problems will test your understanding of the subject matter. If you succeed here, you will be able to tackle all sorts of installations in the field of bioinformatics. There are three problems here that will be solved using the following mechanisms:

* sudo apt-get
* Simple make
* Version control (git)

For the first problem, we will be using version control. We will talk more about this later in the course. To get you started, version control software help you keep track of changes in your code. This is especially important when you’re working on a large project or when you are working on a shared codebase. cvs and git are two of many utilities which do just that.

1. Install git using apt-get

Keep this one simple and figure out what command(s) you need to run to install apt-get on sudo privileges.

ANS: **sudo apt-get install git**

Text

Description automatically generated

1. Straight-forward install

Next up is SAMtools. SAMtools is a program for manipulating mapping files from the mapping of NGS short read sequence data to a genome. It is very fast and great for file format manipulation. Installation of SAMtools is also one of the easiest installations in bioinformatics.

* 1. Download the latest version of the SAMtools ***github*** source code (here is the homepage: <http://www.htslib.org/>, github repos are linked within this page). **Don’t download the pre-compiled binaries.**
  2. Unpack the source and ensure you can find the Makefile. If it doesn't exist, how can you generate it?

**ANS: tar -xf samtools-1.13.tar.bz2**

**The makefile exsists. If it doesn’t, you can use cmake command to generate it.**

A screenshot of a computer

Description automatically generated with medium confidence

* 1. Build samtools and its associated programs with make.

**ANS: ./configure –prefix=/home/ashlesha/Downloads/samtools**

**sudo apt-get install gcc**

**sudo apt-get install libncurses5-dev libncursesw5-dev zlib1g-dev libbz2-dev liblzma-dev**

**Text

Description automatically generated**

**make**

**Text

Description automatically generated**

**make install**

**Text

Description automatically generated**

**Text

Description automatically generated**

* 1. Place the created binaries somewhere on your PATH, e.g. in your bin directory.

**ANS: export PATH=/home/ashlesha/Downloads/samtools/bin:$PATH**

* 1. Run them to see what happens. You should get some help information or errors as you didn’t give them any input.

**ANS: ./bowtie2sam.pl**

**./export2sam.pl**

**Text

Description automatically generated**

1. Harder install: the Kent source tree

The Kent source tree is a huge bundle of bioinformatics utilities named for James Kent, the brain behind the UCSC Genome Browser. The Source Tree has come in handy many times for many people. Although you can get binaries nowadays, you wouldn’t learn how to install things which have similar dependencies as not everything is available precompiled. So, do not download the precompiled binaries for this exercise.

* 1. Obtain the source code for the Kent source tree using git as described at <http://genome.ucsc.edu/admin/git.html>

Text

Description automatically generated

* 1. The README file, or some variation thereof, is often very useful for installation. Read it to figure out what you need to do to install the source tree. If you aren’t setting up your architecture type, you aren’t doing it right.

**ANS:** **mkdir ~/bin/$MACHTYPE**

**export PATH=”$HOME/bin/$MACHTYPE:$PATH”**

**export PATH=”$HOME/bin/aarch64:$PATH”**

**export MACHTYPE= aarch 64**

**cd kent/src/lib**

**make**

Text

Description automatically generated

**sudo apt-get install libpng-dev libmysqlclient-dev uuid-dev**

**make**

**cd ..**

**cd jkOwnLib**

**make**

Text

Description automatically generated

**cd ..**

**cd htslib**

**make**

Text

Description automatically generated

* 1. Attempt to compile the source tree; it should work. If you get stuck, how can you resolve the issue? (Perhaps look into the environment variables). You don’t need to install the whole thing. Individual programs can be built by going into their directories and using make to compile the programs as you need them.

ANS: **cd kent/src/gfServer**

**make**

**kent/src/gfClient**

**make**

**kent/src/blat**

**make**

**kent/src/utils/faToNib**

**make**

**The blat executable is found in the ~/bin/$MACHTYPE folder. It can be run from anywhere since the folder is added to the path.**