Programming for Bioinformatics | BIOL7200

Week 4 Exercise

September 13, 2022

The goal of these exercises is to get you used to working with some basic UNIX commands and their options. Try and think about what each command does, and where it can be used. Reading the `man` page for each of these is **recommended**.

We're going to cover the following commands:

Command	Explanation
`xargs`	Command building utility
`awk`	A simple programming language for line-by-line operations

Instructions for submission

- Prepare two solution sheets for this exercise - one for submission and another for your own reference. **Only submit the submission sheet**. The solution sheet you create for your reference should help you going forward - make it as detailed/brief as you'd like for your own learning style. For the submission sheet, copy the question and write the correct answers below the question.

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

    ls | xargs -I a head a

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

seq 1 10 | xargs -I b touch fileb.fna

    3. Use `sed` and `xargs` to change the extension of the all the `.fna` files to `.fasta`

    ls *.fna | sed 's/\.fna//' | xargs -I c mv c.fna c.fasta

4. Get the first two columns of the UCSC gene file using **cut**
   cut -f1,2 knownGene.txt

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

seq 1 12 | xargs

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

```
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;
```

seq 1 12 | xargs -n3 bash -c 'echo "First number:$0; Second number $1;third number $2;"'

General programming

3. 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`

6

   2. `a_number -= 12`

-6

   3. `a_number += 15`

   9

   4. `a_number /= 3`

3

   5. `a_number *= 2`

6

   6. `a_number = a_number + a_number / 2`

9

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

FALSE

   2. `2 > 1`

   TRUE

   3. `!(2 > 1)`

FALSE

4. `(2 > 1) || (1 > 2)`

TRUE

5. `(2 > 1) && (1 > 2)`

FALSE

6. `!(2 > 1) || !(1 > 2)`

TRUE

7. `!((2 > 1) || (1 > 2))`

FALSE

8. `!((2 > 1) && (1 > 2))`

TRUE

Biologically-inspired problem

5. 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
   - FASTQ
     https://www.genomatix.de/online_help/help/sequence_formats.html#FASTQ
   - 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.

   head -1 file.embl | awk '{if ($1=="#mega"){print " It is a mega file";} else if ($1=="ID"){print " It is a embl file";} else if ($1=="@"){print " It is a fastq file";} else if ($1=="LOCUS"){print " It is a genbank file";} else {print "It is not a mega or fastq or embl or genbank file";}}'

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.

6. 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.

   sudo apt-get install git

7. 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.**

   wget https://github.com/samtools/samtools/releases/download/1.16.1/samtools-1.16.1.tar.bz2

   2. Unpack the source and ensure you can find the **Makefile**. If it doesn't exist, how can you generate it?

   The makefile is present already, if it not it can be generated by using configure.

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

      cd samtools-1.x
      Make
      Make install

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

export PATH= /bin:$PATH

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

8. 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

   git clone git://genome-source.soe.ucsc.edu/kent.git

cd kent

git checkout -t -b beta origin/beta

2. 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.

cat README

wget --timestamping http://hgdownload.soe.ucsc.edu/admin/jksrc.zip

uname -m

vi .bashrc

Make

sudo apt-get install uuid-dev

export MACHTYPE=x86_64

export PATH=~/bin/$MACHTYPE:$PATH

3. 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.