Programming for Bioinformatics
BIOL 8803 B
September 14th, 2015

This week will be about getting/installing/running new programs. The commands and concepts for today are:

`sudo` – super-user do; do the command as a super user. Use with caution.

`apt-get` – package installation on Debian systems

`rpm`, `yum` – RedHat package manager

`fink` – Mac package manager

`PATH` – the environmental variable that the shell uses to search for executables

`bin` – typically the name for a folder that contains executables. Short for binary.

`cc`, `gcc` and `icc` – C compilers

`configure` – a script which builds the makefile

`make` – build the executables

`install` - put the executables where they belong, usually you don't call it yourself

`git` – a version control program, currently the hottest

`cvs` – older, but still good version control

Exercises

1.) `sudo` and installation, removal, etc.

`sudo` is very powerful, but dangerous. In short, it allows you to do anything on the system. On systems you don't own you probably won't have `sudo` privileges. On those where you do, please use it with care and be respectful of other people's data.

a.) People who have Ubuntu:

> i.) Use `apt-get` to install gcc without `sudo`. What happened?
>
> ii.) Try again with `sudo`.
>
> iii.) Why does it work now?
>
> iv.) A lot of installation on Ubuntu can be handled with `apt-get`. Again: **Don't use it for the exercises here**; not everything has a neat little package and you need to know how to install things that don't.

b.) If you have a Mac and you are feeling bold, finish the other exercises and install `fink`. Try (a) above using `fink` instead of `apt-get`.

2.) The `PATH` to enlightenment

The `PATH` environmental variable tells the shell where to look for executables. When you use the `ls` command, it works because the shell knows to look for it on the `PATH`.

a.) Print your `PATH` environmental variable to the screen. What are the directories in it? Some of them should look familiar at this point.

b.) Make a directory called '`bin`' in your home directory and put a symbolic link to some already existing executable in the newly created `bin` directory. Call the link something different from the actual executable or this is all for naught.

c.) Add your `bin` directory to your `PATH` environmental variable. Keep in mind what character separates the different directories in the `PATH` environmental variable. If you mess up your `PATH`, then just leave the shell and make a new one.

d.) Try running the executable through the symbolic link. Did it work? Use which to see the alias.

e.) Add your new `bin` directory to your `PATH` in your login file (you should know what this means). Make sure you didn't everything. If you did, get it right.

f.) Congratulations! You now have a centralized folder where you can install executables and will be on your `PATH` every time you log in. If you don't have permission to install things, install them here!

g.) If you're on a system with multiple users, what might be an advantage of installing an executable in one of the directories on the default `PATH`?

3.) The C compilers

There are various C compilers available today. They all pretty much do the same thing, turn C code into a program, but they do it with widely varying efficiency.

a.) Find out what your `cc` command really goes to, i.e. what is your default C compiler?

b.) Compile the `haveSomeNumbers.c` file from T-Square and run the result (you're going to need the `–o` flag. Read about it before using it). Add the result to your `PATH` and run it that way.

c.) C is beyond the scope of this course, but try and figure out what the `haveSomeNumbers.c` file does anyway. This program is way simple. Many, many bioinformatics programs are written in C.

d.) Look up comparisons of the `gcc` and `icc` online. What are the advantages and disadvantages of each? What other C compilers exist?

I assume you'll have some basic knowledge on what compilers are and what it means to compile a program no matter, how big or how small, at this point. In the following exercises we are going to do something that will be very pertinent to all of you regardless of the area you work in (or will be working in). Although in the following exercises, you'll be doing a specific task (calling variants and changing the ref version), the basic procedure of how you accomplished the task remains the same across the fields.

Basic Concept: The field is moving from assembling genomes from scratch to mapping to an already assembled genome with high level of annotations. This saves enormous amount of time, computational resources as well as human resources. Is you work with a eukaryotic genome with an assembled reference, you will be working this way. If you work with prokaryotes, at some point you will do this too.

The whole process starts with mapping the sequencing reads to a reference genome. The output of the process is a SAM (Sequence Alignment Map) file. This file is converted to a BAM file (Binary Alignment Map), the compressed brother of SAM. That's your starting point today. The following steps are sorting the BAM file, converting it to MPILEUP format, calling the variants.

The additional wrinkle today is that the version of the reference genome you mapped your file to, is old now and there is a new version out. You already have done variant calling and a set of analysis and would not like to repeat the whole analysis. Is there a way you can translate your coordinates of your initial variant calling to match the new reference genome version? Yes – the tool is part of Kent source, a large repository of tools written primarily by Jim Kent (a.k.a. the god) to do a lot of operations on genomic data. Try googling what you can do with it.

Now let's do the whole thing as an exercise here.

Basic Workflow: In order to achieve this, we'll have to:

1) Sort the BAM file
2) Convert BAM to MPILEUP
3) Call variants
4) Lift the coordinates to a different ref genome version

In order to do this, we have the following tools at our disposals:

1) Sort the BAM file – samtools sort
2) Convert BAM to MPILEUP – samtools mpileup
3) Call variants – bcftools
4) Lift the coordinates to a different ref genome version – liftOver from Kent Source

So we need to install samtools and Kent source. Samtools is straightforward but Kent has multiple dependencies. MySQL is one of them. So we'll start by installing samtools, MySQL and Kent in order, from scratch. For any other dependencies – feel free to use sudo apt-get for this exercise.

4.) Compiling binaries.

Samtools is a program for mapping second-generation short read sequencing data to a genome. It is very, very fast and is great for mapping sequencing data.

a.) Download the latest version of the samtools program source code (http://www.htslib.org/), not the pre-compiled binaries.

b.) Unpack the source browse the `Makefile`, some parts should look vaguely familiar. What does a `Makefile` do?

c.) Build `samtools` and its associated programs with `make`.

d.) Put the created binaries somewhere on your `PATH`, e.g. in your `bin` directory.

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

5.) A harder install with MySQL

MySQL is a relational database management system. If that doesn't mean anything to you right now that's okay, but databases are extremely useful in bioinformatics. I recommend relational databases (taught in CS 4400) for everybody. I have a mountain of SQL books if you are interested. MySQL is also a good example for typical compilation/installation.

a.) Download the latest source code for MySQL (http://www.mysql.com/downloads/mysql/), not the precompiled binaries.

b.) Unpack the source and run `cmake .` in the just created directory. You won't have cmake in your system, get it from apt-get.

c.) Build the MySQL executables with `make`

d.) Try to install them with `make install`

e.) That should have failed. Why?

f.) How would you get around this with `sudo`, and how would you get around this with the `cmake` (hint: you have to tell `cmake` where YOUR `bin` directory is. Run `cmake --help`)?

6.) Version control with `git` and `cvs`

When you're working on a big project you need a way to track changes in your code and share code. `cvs` and `git` are utilities which do just that. It doesn't really matter to me which one you use, though `git` is very popular these days. For right now, we're just going to use them to download existing code.

a.) Install them both with `apt-get`.

7.) Ultimate installation challenge: 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. He looks like a hobo. 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: **don't download the precompiled binaries for this exercise**.

a.) Obtain the source code for the Kent Source Tree using `git` as described at http://genome.ucsc.edu/admin/git.html

b.) 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 your architecture type, you aren't doing it right.

c.) 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, which is a huge pain. Individual programs can be built by going into their directories and using `make` to compile the programs as you need them.

Since this a challenge and a huge pain if you aren't doing it right – come to me once you've exhausted every possibility. It's fun once you get it right.

HINT: This exercise is all about dependencies and environment variables.

8.) Lifting variants from one version to another.

This is a real, bona-fide task that is done quite often by many people. Reference genomes are updated continually. The coordinates that you have today might or might not change in the next release. It doesn't make sense to keep repeating the analysis every time a new version is out. That would be horrible to do. So let's be a little smarter here.

a.) Download (wget/curl) the BAM file from http://jordan.biology.gatech.edu/biol8803b/. This BAM file contains reads mapped to chromosome 21 of human genome version 18 (hg18). Again, a BAM is a binary version of SAM file. Which means you can't read it as it is (unless you are some sort of cyborg). But if you want to know what kind of information is there in the file, check here: http://samtools.github.io/hts-specs/SAMv1.pdf

The chr21 file for hg18 is located on the UCSC genome browser's download page. It can also be reached directly from this link http://hgdownload.soe.ucsc.edu/goldenPath/hg18/chromosomes/

b.) Now let's sort the file. Whenever you map reads to a reference, they will come out in the order they went. But when you call variants, you want the order to be according to the genomic coordinate as you walk through the genome to perform variant calling.

The command is simple, see here: http://samtools.sourceforge.net/samtools.shtml.

c.) Great! Give me some variants now!

```
samtools mpileup -uf ref.fa aln.bam | bcftools view -bvcg - > var.raw.bcf

bcftools view var.raw.bcf | vcfutils.pl varFilter -D100 > var.flt.vcf
```

NOTE: DON'T COPY PASTE. IT WILL NOT WORK. The files ref.fa and aln.bam are your reference genome file and sorted alignment file respectively. The files var.raw.bcf and var.flt.vcf are the output file which can be named anything. Keep the extensions (bcf and vcf) so that you know what they are.

Source: http://samtools.sourceforge.net/mpileup.shtml. Try to understand what happened with the above commands.

NOTE 2: With GATK in picture, things have complicated a bit but in the spirit of not making this exercise harder, we'll stick to the two lines of code in the link

d.) In order to use liftover, we will require to convert the VCF file to BED format. This can be done by the following command:

```
awk '{if($1 !~ /^#/){print $1, $2, $2+1}}' var.flt.vcf > in.bed
```

Of course change var.flt.vcf to whatever name you gave. What did I just do in that command?

e.) Stay with me, we are almost through. Give the var.bed file that you just created to liftover and use appropriate mapping files.

The mapping files can be found here: http://hgdownload.soe.ucsc.edu/downloads.html. Scroll down to hg18 (cause that's what the data was mapped against), click on LiftOver files and select hg18Tohg19.over.chain.gz

To know how to run liftOver program, just run it without any arguments. The usage instructions is in the first 5 lines. That's how it typically is. It's not hard at all.

f.) Congratulations! You learned how to move around in different versions of reference genomes. **Rename** the new bed file you just created as **hg19.bed, and submit** the new bed file on T-square.