A Legofit Tutorial

Alan R. Rogers

July 28, 2022

Contents

1 Introduction

This is a tutorial on Legofit, a package that uses genetic data to estimate the history of population size, subdivision, and admixture [8, 9]. I've designed it for use in the AGAR Workshop on Computational Evolutionary Genetics, which will meet online July 27–29, 2022. There is much more in this document than we will use in the workshop. The extra material is there for those who want to use Legofit on their own, after the workshop is over. For the workshop itself, you will need sections 2, **??** (excluding **??**), and **??**. Further information about Legofit is available online.

Try to install Legofit (sec. 2) on your own before the workshop starts. We'll help with installation problems during the workshop, but that will take less time if you've already made a start.

2 Installing Legofit

This section explains how to get Legofit running on your own computer.

2.1 Install git if necessary

You may have git already. To find out whether you do, type:

```
type git
```

On Mac or Linux, this will print the location of the executable git file. If there is no such file, you'll get an error message saying `type: git: not found`. I recommend using homebrew to install packages on a Mac. To install git using homebrew, type:

```
brew install git
```

Otherwise, visit git's website and look for the download instructions.

2.2 Clone legofit

Use git to clone Legofit onto your machine. This step will create a subdirectory called "legofit," which will contain the source code. Before executing the command below, use the `cd` command to move into the directory where you keep source code that other people have written. I keep such code in a directory called `distrib`, which I originally created by typing `mkdir distrib`. Having created such a directory, move into it by typing

```
cd distrib
```

Then clone legofit by typing:

```
git clone https://github.com/alanrogers/legofit.git legofit
```

This will create a directory called legofit. Use `cd` to move into it.

2.3 Install a C compiler if necessary

To see whether you have a C compiler already, type `type cc`, or `type gcc`, or `type clang`. You should get output like

```
cc is /usr/bin/cc
```

If you need to install a C compiler, there are several alternatives. On a Mac, you can install Xcode from the App Store. Or you can use Homebrew to install clang or gcc:

```
brew install clang
```

or

```
brew install gcc
```

On Linux, the command would be

```
sudo apt-get install clang
```

or

```
sudo apt-get install gcc
```

2.4 Install "make" if necessary

You will also need the program "make," so type "type make." If you need to install, then

```
brew install make
```

2.5 Set up a "bin" directory to hold executable files.

On Unix-like operating systems (MacOS and Linux) it is conventional for each user to maintain a directory named "bin," just below the home directory, which contains executable files. This directory must also be added to the system's PATH variable, which is used for finding executable files.

First create a "bin" directory, if you don't already have one. To do so, first type the command

```
cd
```

at the shell prompt. This moves you into your home directory. Then type

```
mkdir bin
```

This creates a new directory called "bin."

You now need to add this to your shell's PATH variable. I'll assume you're using the bash shell. In your home directory, look for a file called either "`.bash_profile`" or "`.profile`". Because the name begins with ".", it will not show up if you type `ls`. However, it will appear if you type `ls .bash_profile` or `ls .profile`. If only one of these files exists, open it with a text editor (not a word processor). If they both exist, edit "`.bash_profile`". If neither exists, use the editor to create a new file called "`.bash_profile`".

Within this file, you may find existing definitions of the PATH variable. Add the following after the last line that changes this variable:

```
export PATH=$HOME/bin:$PATH
```

Save this change and exit the editor.

The contents of this file are executed by the shell each time you log into your computer. Since you have just created the file, your PATH has not yet been reset. Log out and then log back in again, and your PATH should be set. To check it, type

```
echo $PATH
```

This will print your PATH.

2.6 Compile and install legofit

The details are in the Legofit documentation.

3 Setting up your environment at the CHPC

This section isn't needed for the AGAR workshop, because we aren't using the CHPC.

Utah's CHPC has several "clusters," each with many "nodes." Each node is a powerful computer in its own right. The slurm version of the pipeline submits legofit jobs to multiple nodes so that they can run in parallel. Within each node, legofit parallelizes across all available cores. This greatly reduces the time required for an analysis.

To log into a CHPC cluster, your own computer must support ssh. To find out whether it does, type the command

```
type ssh
```

at the bash prompt. If ssh is on your system, the response will be something like

```
ssh is /usr/bin/ssh
```

If ssh is not on your system, the response will be

```
-bash: type: ssh: not found
```

In the latter case, install openssh by typing

```
brew install openssh
```

Now that you have access to ssh, you can log into one of the CHPC clusters by typing the following at the bash prompt. Either

```
ssh -l <your unid> notchpeak.chpc.utah.edu
```

or

```
ssh -l <your unid> kingspeak.chpc.utah.edu
```

or

```
ssh -l <your unid> lonepeak.chpc.utah.edu
```

Here, `<your unid>` is your own University of Utah id. Each of these commands will log you onto one of the clusters at Utah's CHPC.

In the sections that follow, I will introduce you to several directories and files on the CHPC servers. Each one is described by its "path name" relative to your own "home directory," which is represented by the symbol `~`. (In scripts, this directory is often called `$HOME`.) Before these path names will work, you'll need to define two "soft links" within your own home directory, which point to group storage devices owned by the Rogers lab. After you define these soft links, `~/grp1` will refer to one of these group storage devices and `~/grp2` to the other. You can establish these links by typing

```
ln -s /uufs/chpc.utah.edu/common/home/rogersa-group1 ~/grp1
ln -s /uufs/chpc.utah.edu/common/home/rogersa-group2 ~/grp2
```

If everyone defines these soft links in the same way, then we can all use path names of the form `~/grp1/rogers/data/whatever`.

The Legofit executables are already available in directory `~/grp1/bin`. To make these easily available, use a text editor to edit the file `.bash_profile` in your home directory. Within that file, you will find a line that defines a variable called `PATH`, which is used by bash to find executable programs. We want to add `~/grp1/bin` to the end of `PATH`, so that bash can find the Legofit executables. Edit that line that defines `PATH` so that it looks like this

```
export PATH=$PATH:$HOME/bin:$HOME/.local/bin:$HOME/grp1/bin
```

Also add the following line at the end of the file

```
module --latest load git
```

This last line will give you access to the program "git," which you'll need to access the agar22 repository. After making these changes, save the file and exit the editor. The file `.bash_profile` is normally read only once, when you first log in. Because you've only just edited this file, bash doesn't yet know about the changes you made there. To fix this, you can either type `logout` and then `ssh` back in again, or type `. .bash_profile`, which tells bash to execute the commands in this file.

Finally, clone the agar22 repository into your own home directory on the CHPC.

```
git clone https://github.com/alanrogers/agar22.git agar22
```

This will create a subdirectory called "agar22," which contains files related to the workshop.

4 Making an input file in .raf format

Legofit uses data in ".raf" format. The following data sets are already available on the server at Utah's CHPC.

Altai Neanderthal	`~/grp1/rogers/data/altai/orig2/altai.raf.gz`
Vindija Neanderthal	`~/grp1/rogers/data/vindija/orig/vindija.raf.gz`
Western Europeans (SGDP)	`~/grp1/rogers/data/simons/raf/weur.raf.gz`

If you'd like to make a data set for another modern human population, copy the following slurm script into a directory of your own:

`~/grp1/rogers/data/simons/raf/weur.slr`

Change its name (because "weur" stands for "western European") and edit it to reflect the population or populations that you want to study. You'll find lots of options under

`~/grp1/rogers/data/simons/`

Near the top of this script, you'll find the lines

```
#SBATCH --account=rogersa-np
#SBATCH --partition=rogersa-np
```

These specify the account and partition under which the job will run. The choice "rogersa-np" refers to my (single) node on the notchpeak cluster. To run the job there, log into the notchpeak cluster, use "cd" to move to the directory that contains the script and then type

`sbatch <your script name>`

where `<your script name>` is the name of your version of the script.

If the notchpeak node is busy, try kingspeak. To do so, log into kingspeak and specify `--account=rogersa-kp` and `--partition=rogersa-kp`. To do this, it isn't necessary to edit the file. You can override the values in the file by specifying these options on the command line, like this:

`sbatch -A rogersa-kp -p rogersa-kp <your script name>`

If you're a member of my lab, you can also run under my allocation. To do this on notchpeak, log into that cluster and use

`sbatch -A rogersa -p notchpeak <your script name>`

On kingspeak, the command is

`sbatch -A rogersa -p kingspeak <your script name>`

The lonepeak cluster is available to users who have no allocation. To use it, log into lonepeak and type

`sbatch -A rogersa -p lonepeak <your script name>`

However you run your script, it will take hours to complete. You can check on its status at any time by typing

`squeue -u <your user id>`

where `<your user id>` is the id you use to log onto the CHPC cluster. As a shorthand, I often type this as

`squeue -u ‘whoami‘`

which uses Linux's "whoami" command to fill in your user id. (Note: the ticks in this command are "back ticks.")