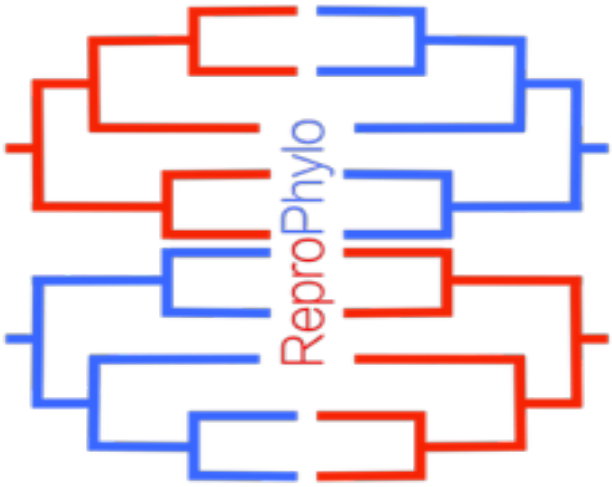


ReproPhylo

reproducible phylogenomics

User Manual

You can edit this document to improve it. Be bold.

Software: https://github.com/HullUni-bioinformatics/ReproPhylo
This manual: http://goo.gl/aZeRXf
Webpage: http://hulluni-bioinformatics.github.io/ReproPhylo @ReproPhylo
Quick links: FAQ; Citation; Contact; Download; Installation

Table of contents

1. Introduction

1.1. What ReproPhylo is

ReproPhylo is a pipeline of modules to implement reproducible workflows for phylogenomic analysis. It is open source software (CC0 public domain) for you to use, modify and distribute as you see fit. This is community software, we would welcome your contributions.

1.2. About this Manual

This document outlines how best to use the reproducible phylogenomics pipeline ReproPhylo. **You can edit this document like you would a wiki and also leave comments.** Be bold. We would welcome your additions, edits, and corrections to this document. You could also create a GitHub issue. This manual has medium level information; its more than basic, but some details are linked to rather than included as they break up the flow.

1.3. Brief overview of reproducibility

Our reproducible phylogenomics ideas are outlined in a series of blog posts part 1, part 2a, part 2b. ReproPhylo allows you to carry out a phylogenomic analysis using pre-written or self-written commands. This programmatic approach ensures that all stages of the analysis are explicitly recorded, and can be exactly replicated, or reproduced with modification, as required. Sequence data, and any generated intermediate data files (e.g. alignments, metadata), are tracked and held in version control- meaning that there can be no doubt which version of which file was used for any analysis. ReproPhylo will write a human-readable detailed graphical report for each experiment. ReproPhylo will create a .zip archive of the entire experiment to upload to FigShare or equivalent repository. ReproPhylo is best deployed as a Docker container, which includes not just the experimental components but also the phylogenetics programs and any dependencies, ensuring that it can be run exactly as it was on the previous phylogeneticist's machine.

1.4. Version control in ReproPhylo

The purpose of version control is to track all changes to your files and to enable you to return to any previous file version if you require. This allows you to recover from mistakes and to change your mind about some of your actions, without having to repeat previous analysis.

In ReproPhylo, the *input data* (sequences) and its *metadata* (eg sampling location), the *output* (eg alignments, trees), its *statistics* (eg, informative alignment positions), *methods information* (eg command lines, supermatrix content) and *environment information* (eg program versions) are all automatically and continuously saved in a single file, termed pickle. This file is automatically and continuously tracked by the version control program Git without you having to do anything (ReproPhylo GIt demo). We are considering an automated push to GitHub, but at the moment, you can push your Git repository manually using the git push command. Tailored instructions for doing so are provided on GitHub every time you start a new repository there.

2. Installation and Launch

The recommended way to use ReproPhylo is in **Jupyter Notebook (**previously called IPython notebooks), with commands and instructions in an interactive GUI environment for you to run or modify. [about Jupyter notebooks aka 'IPython']. Instructions for installation without Docker can be found here.

As a proof of concept, ReproPhylo is also available in **Galaxy**; a web-based GUI, incorporating interactive workflows, and providing its own set of reproducibility tools [about Galaxy]

The latest software can be downloaded from the GitHub ReproPhylo page, but it is not a single-click install. We provide a ReproPhylo Docker environment available from dockerhub, which is a ready-to-go self-contained virtual machine with all dependencies pre-installed and is the easiest way to install ReproPhylo. A Galaxy instance containing ReproPhylo tools is also provided (see section 5).

2.1. ReproPhylo in Docker (Linux machines)

2.1.1. Starting up

The quickest way to deploy ReproPhylo in Jupyter Notebook is by using the Docker container on a Linux machine. Instructions on how to **install on Windows/ OSX** can be found in the next section.

You will first need to install_Docker on your system.

Then, download_startRP.
It is a script which will deploy Jupyter notebook with ReproPhylo, as well as tutorial notebooks.

The startRP script is downloaded as a zip file. Extract it and then copy the startRP shell script anywhere on your machine.

startRP runs as follows:

```
$ cd /path/to/startRP/

$ sh ./startRP /absolute/path/to/your/project/directory
```

Done!

The **project directory** refers to wherever you are going to work, e.g. you could create an empty "my_experiment" directory in your home folder with `mkdir ~/my_experiment` and the absolute path would be `/home/your_user_name/my_experiment`. The directory will be created if it does not already exist, so take care to avoid typos, as a misspelled path will be created and used instead of the one you have intended to point at. The script will allow the Docker container to access your display, and will start up your default browser with the Jupyter notebook login page. If this is your first start-up, getting to this point will take a few minutes as the Docker image will be downloaded. The browser may warn you that the address is untrusted, in which case you should follow the given instruction to permit access. Jupyter notebook will ask for a password, which is `password`. You may change it by editing the startRP shell script, if you intend to access the container remotely.

ReproPhylo should now be working and you can run it through the tutorial Jupyter notebooks (as described in section 3 below) or modify them for your own analyses. The tutorial notebooks should be visible in the Jupyter notebook file browser, once you have logged in, in the `Tutorial_files/Basic` directory.

The same script can be used to start a ReproPhylo container in subsequent runs, provided the program was stopped appropriately (see next paragraph). It won't reinstall everything and will be considerably quicker.

Ins and outs of startRP (Only read if startRP doesn't work)

The startRP script was tested on Ubuntu 14.04 and thus uses `apt-get` frequently. Other commands (such as pip, wget, mkdir, xhost, xdg-open) are universal for GNU/Linux machines. If you are trying to use it on another Linux distribution, some modifications will be required, and we'll be happy to try and assist should you get stuck (see below for Windows and OSX).

startRP is designed for a local installation, and it therefore contains two classes of commands. The first class perceives your local computer as a *server*. It will take care of "serving" the Docker container with ReproPhylo in it. The second class perceives your computer as a *client*. It will take care of running ReproPhylo in your browser. This means that the script as it is, cannot be used directly on a true server, as some commands will make no sense, and there will be some additional security considerations. For convenience, I am listing the two classes here:

Server side:

Creating the working directory on your local machine as a server

As the script is set up at the moment, the working directory is considered to be on the server side. It makes no difference if your server and client are the same machine, but a different solution may be needed if ReproPhylo is truly served remotely (`$1` is the startRP command line argument stating the path to the working directory).

```
echo "checking if path exists: $1"
if [ -d $1 ]; then
  echo "Already exists"
else
  echo "Path was not found, creating."
  mkdir $1
fi
```

If you wish to make this operation manually, the appropriate command would be:

```
mkdir /path/to/workdir
```

Downloading tutorial files to your local machine as a server

The script checks if the tutorial files are in the working directory (which is "server side") and if you want to install them (i.e., if there is no `--xt` flag in the startRP command line). It makes no difference if you serve and use on the same machine, but a different solution may be needed if ReproPhylo is truly served remotely (`$1` is the startRP command line argument stating the path to the working directory).

```
if [ -d $1/Tutorial_files ]; then
   echo "Tutorial_files exists in $1"
elif [ $2 = "--xt" ]; then
   echo "Tutorial files opt out"
elif [ $# -eq 2 ] && [ $2 != "--xt" ] || [ $# -eq 1 ]; then
   echo "Putting tutorial files in $1"
   wget -c  https://github.com/HullUni-bioinformatics/ReproPhylo/archive/master.zip
   unzip -qq master.zip
   cd ReproPhylo-master
   cp -r Tutorial_files $1
   cd ..
   rm -r ReproPhylo-master
   rm master.zip
   echo "Tutorial files are in $1"
fi
```

If you wish to get the tutorial files and extract them in the working directory manually, here are the needed commands:

```
cd /path/to/workdir
wget -c https://github.com/HullUni-bioinformatics/ReproPhylo/archive/master.zip
```

```
unzip -qq master.zip
cp -r ReproPhylo-master/Tutorial_files .
rm -r ReproPhylo-master master.zip
```

Serving the Docker container

```
sudo docker run --net=host --name rpnotebook -d -p 8888:8888 -e "PASSWORD=password" -v /tmp/.X11-unix:/tmp/.X11-unix:ro -v $1:/notebooks -e DISPLAY=$DISPLAY szitenberg/reprophylo /notebook.sh
```

`sudo docker run` -The docker command to (download an image), create and run a container

`--net=host` -Use host to access the internet

`--name rpnotebook` -The container's name

`-d` -Run as daemon (at the background)

`-p 8888:8888` -Couple port 8888 in the container with port 8888 in the server. This is limiting because it does not take advantage of Jupyter notebook's ability to dynamically choose a vacant port. To change the port you'll need to edit both startRP and notebook.sh in the container: Clone the Dockerfile repository and edit the script there, by changing 8888 in the ipython notebook command line to something else. In the Dockerfile itself, the line EXPOSE 8888 will have to be changed accordingly. Then build the Docker image locally.

`-e "PASSWORD=password"` -sets up the password to access Jupyter notebook

`-v /tmp/.X11-unix:/tmp/.X11-unix:ro` -Couple the location of x-server in the container and server in a read only mode (ro).

`-v $1:/notebooks` -Couples the work dir on the server with the one in the container (`/notebooks`)

`-e DISPLAY=$DISPLAY` -Couples the DISPLAY variable between the container and the server

`szitenberg/reprophylo` -The Docker image name

`/notebook.sh` -The container's start up command. Runs the script notebook.sh

Running this step manually will be done the same, just remember to change `$1` to the actual `/path/to/workdir`

Also remember to install Docker beforehand. Note that Docker does have Windows and OSX versions, but ReproPhylo will only work in the Docker terminal (called boot2docker and is a virtual machine) in those OSs. This means you will be able to use all the programs and python modules, ReproPhylo included, in IPython, in itself very useful, but you will not be able to run Jupyter notebook (easily solvable) or produce figures (very difficult).

Client side:

Allowing the served program to access the local x server

```
xhost +local:root
```

Since the container is designed to work locally, the program is ran as root. We then need to allow the "remote" container to access the "local" x-server, local and remote being the same physical machine. A remote Docker container will have to be set up to run the program as a specific user, rather than

root as it is now, for security reasons, and then the local xhost command will have to change accordingly.

Starting up a browser

```
xdg-open https://localhost:8888
```

Start up the default browser at *https://localhost:8888.* localhost should be changed to the server name or IP if the container is served remotely. **You could just open your browser as you normally would and stick the address in instead.**

2.1.2. Stopping a ReproPhylo Docker container

When you use startRP, your files are read from and written to a local directory. Therefore it is safe to stop and remove a container between sessions. You can save the following shell script as `stopRP` and use it to stop your Docker container once you're done:

```
#!/bin/bash
sudo docker rm -f rpnotebook
```

and run as follows:

```
$ cd /path/to/stopRP/

$ sh ./stopRP
```

or you could just run

```
sudo docker rm -f rpnotebook
```

in the terminal

2.2. Linux installation without Docker

Instructions for Linux installation without Docker can be found here.

2.3. Windows and OSX (and also linux)

2.3.1 Vagrant

This option was tested on OSX and Ubuntu 14.04, and should work just as well on Windows. At the background, we'll set a virtual machine to serve Jupyter notebook without a display, and then use the local web browser as GUI. A great thank you to Dr. Steve Moss for his work on this distribution. Here are steps:

Install vagrant on your OS.
Make sure to get the latest version, which is not always what you get with your distro manager.

Install virtualbox for your OS

Download ReproPhyloVagrant, which is a zip file (or git clone)

Extract the zip file, a directory called ReproPhyloVagrant-master should appear.

Start a terminal or a cmdva

Change location into the extracted folder:
`cd /path/to/ReproPhyloVagrant-master`

Run the command **`vagrant up`**
This command will set up the virtual machine and can take half an hour to run the first time.

Run the command: **`vagrant ssh`**
The prompt in the terminal should change and indicate that you are now inside the virtual machine

Run the command: **`startRP.sh`**

In your browser, go to `https://localhost:8888`
You may get a warning about security, follow the instructions to allow the page to load. There are no security issues, it all happens locally.

Type in the password **`reprophylo`**
Done!
Further instructions are in section 3.

Your notebooks and output files will be saved in `ReproPhyloVagrant-master/notebooks`, or any subdirectory you create in there. Data you want to use also has to be placed within `ReproPhyloVagrant-master/notebooks` or its subfolders.

To stop the virtual machine do the following:

cd /path/to/ReproPhyloVagrant-master
`ctrl+C` twice to stop ipython notebook
`exit` to leach the virtual machine
`vagrant halt` to stop it

Next time you `run vagrant` up it will be very quick.

2.3.2. WinPython (deprecated)

A WinPython 64bit version, which includes ReproPhylo and most of the dependencies is available to download here. It is an older version of ReproPhylo and we do not expect to maintain this distribution in the near future. Setting it up should work as follows:

1. Download Git installer for Windows
2. Run it to install. You may need to right-click and run as administrator. When asked, choose to allow Git to work in the Windows command prompt and not only in bash. Otherwise, keep the default settings.
3. *Optional:* Perl is required for codon alignment. If you may need to use this option and Perl is not yet installed, download and run this installer.
4. Download the WinPython+ReproPhylo zip file. Note that this is not the official WinPython distribution, but version 2.7.9.5, Release 2015-04 of May 12th, 2015.
5. Extract anywhere on your machine.
6. To start up Jupyter Notebook, find `IPythoNotebook.exe` and double-click it. Jupyter Notebook will start up in your default browser.

All done!

Notebooks you create will be written within `WinPython-64bit-2.7.9.5/notebooks`. However, you can use data from anywhere on your machine, as long as you specify the path in your scripts. This directory already contains two subdirectories, one with the tutorials discussed in the next section, and

another with WinPython documentation.

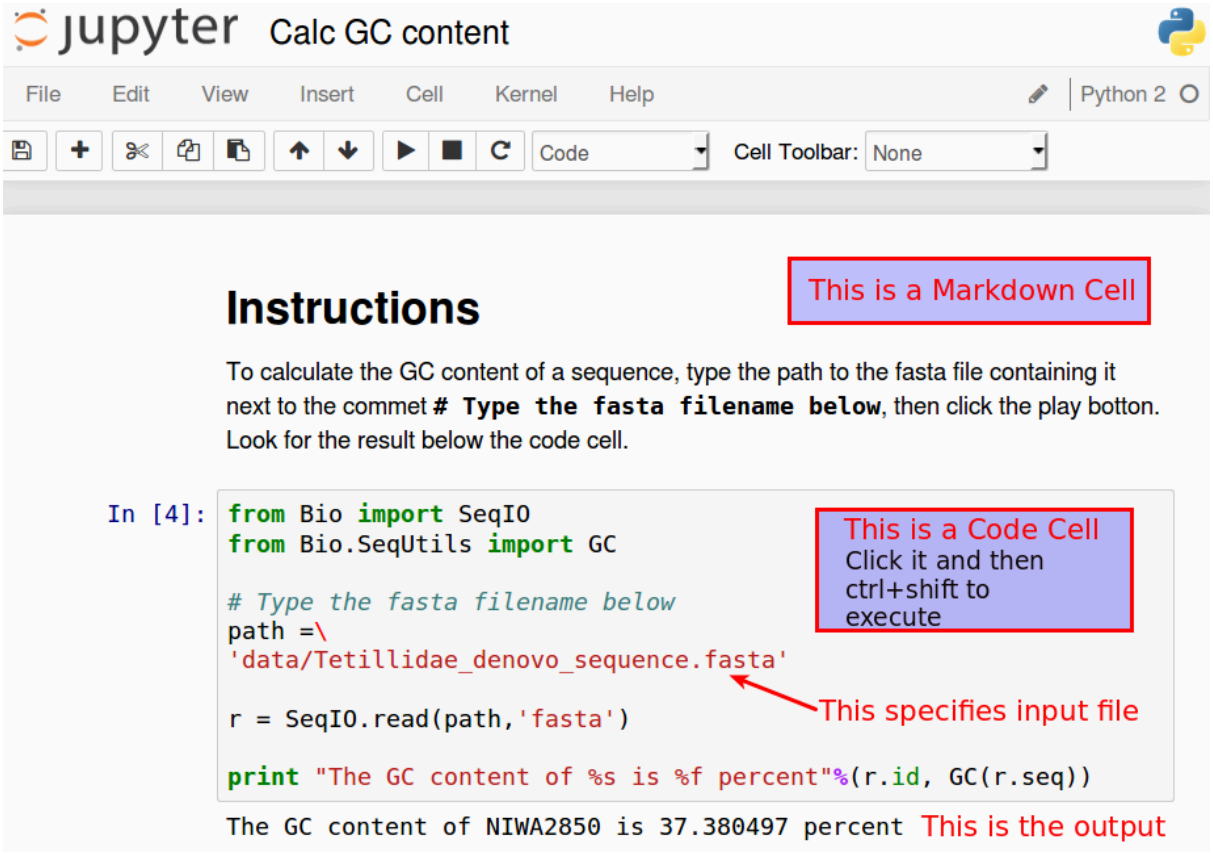
The programs already included within this WinPython distribution are raxmlHPC, raxmlHPC-PTHREAD-SS3, MAFFT, MUSCLE, TrimAl and PAL2NAL. They are accessed by default by ReproPhylo and there is no need to set up paths. You can run any other command line tool that is on your machine by specifying its full path. Since there are ReproPhylo functions that run EXONERATE, you might want to install its Windows/cygwin version. Similarly, Bayestraits is also available for Windows. Any function in ReproPhylo that execute external software will have a cmd keyword where the full path to the executable can be specified. Programs that are not wrapped by ReproPhylo can be executed using the included `subprocess` module or the ! notation in Jupyter Notebook.

3. Tutorial

This tutorial is currently being rewritten, as many options were added since it was first written. Sections 3.1 - 3.4 are new, and the remaining, while informative, is not completely up to date. This notebook illustrates most of the additional functionality.

3.1. Jupyter Notebook Intro

The use cases described in this manual are also provided as functional Jupyter Notebooks which can be downloaded from ReproPhylo github. They are downloaded by default when you use a Docker container with the startRP script. A Jupyter Notebook consists of formatted text (markdown) cells, which contain comments and explanations, but do not affect the program. Actual script is written in code cells, which have a shaded background. The code in the code cells can be executed (run) by placing the cursor anywhere inside a code cell and clicking `shift+enter`.



3.2. Starting a Project

Section quick reference, Full reference

This section in nbviewer

In the beginning of each analysis, the first step is to load ReproPhylo and its dependencies with the command

```
from reprophylo import *
```

Once this is done we can start a Project. A Project contains all the data, metadata, methods and environment information, and it is the unit that is saved as a pickle file, which is version controlled with Git.

Although ReproPhylo is designed to record versions and update the pickle file automatically, this will be opt-out of in this tutorial, and will be introduced after we have covered the basics. Instead, we will manually save a pickle file at the end of each section, and will load it in the next one. You should use the same pickle file name at the end of all the sections. The new content will be added to the one already present in the file.

If you want to jump ahead, there are presaved pickle files (`Tutorial_files/basic/outputs`), numbered according to the section after which they were saved. For example, `outputs/3.6.alignments.pkpj` was saved at the end of section 3.6 and can be loaded at the top of section 3.7, instead of your own file.
To start a `Project`, we have to specify the loci to analyse (not actual sequence data, only some information on the loci) and a pickle file name.

3.2.1 Describing Loci

A `Locus` can be described manually using a command or by providing a file. For each `Locus`, we have to specify the character type (DNA or protein) the feature type (eg, rRNA, CDS or gene), the name of the locus (eg, MT-CO1) and other possible aliases which may come handy if we want to read a genbank file (eg, cox1, coi).

Describe loci using a command

```
coi = Locus(char_type='dna',
            feature_type='CDS',
            name='MT-CO1',
            aliases=['cox1', 'coi'])
```

This is a single `Locus` description (a `Locus` object). We can confirm its content by printing it like this:

```
print coi
```

to get this output:

```
Locus(char_type=dna, feature_type=CDS, name=MT-CO1, aliases=cox1;
coi)
```

Describing loci using a file

Another way of describing loci is to write them in a file. The file has one line for each `Locus`, where each line has at least four items, separated by commas. The items, as above, are the character type, the feature type, the name of the locus and other possible aliases. At least one alias must be specified, but it can be identical to the name. For the MT-CO1 Locus, a file would look like this:

```
dna,CDS,MT-CO1,cox1,coi
```

Deducing a loci file from a genbank file

A third way of describing loci is to run a command that guesses them from a genbank file and writes them into a comma delimited file, as above. This file can be used as is, or it can be edited. The following command will prepare such a loci file from a genbank file containing all the GenBank records belonging to the sponge family Tetillidae. Text starting with a hash (`#`) is a comment which do not affect the command:

```
list_loci_in_genbank('data/Tetillidae.gb', # The input genbank
                                           # file

                     'data/loci.csv',      # The loci file

                     'outputs/loci_counts.txt') # Additional
                                                # output,
                                                # discussed
                                                # below.
```

The command generated the loci file and wrote it in `data/loci.csv`. Here are some excerpts separated by three dots:

```
dna,rRNA,18s,18S ribosomal RNA,18S rRNA
dna,rRNA,28s,28S large subunit ribosomal RNA,28S ribosomal RNA
...
dna,CDS,MT-ATP8,atp8,ATP8
dna,CDS,MT-CO1,coi,COI,cox1,COX1,coxI
...
dna,rRNA,rnl,rnl
dna,rRNA,rns,rns
dna,rRNA,rrnL,rrnL
```

Each line represents a locus that was found in the genbank file `data/Tetillidae.gb`. For some genes, such as 18s, synonyms were recognized and placed as aliases in one line. In other cases, such as for rnl and rrnL, they were not.

Editing the loci file

Possible edits to this file include:

Synonymization. This is done by adding a comma and a shared integer in all the lines that are the same locus. For example the lines

```
dna,rRNA,rnl,rnl
dna,rRNA,rrnL,rrnL
```

will become

```
dna,rRNA,rnl,rnl,9
dna,rRNA,rrnL,rrnL,9
```

Which integer is written is unimportant, as long as it is shared between synonymous lines.

Change of character type. If our data includes translations to protein sequence, we can change dna to prot, as such:

```
prot,CDS,MT-CO1,coi,COI,cox1,COX1,coxI.
```

This will tell the program to use protein sequences instead of DNA sequence. The sequence alignment tutorial explains how to use both protein and DNA sequence of the same locus to conduct codon alignment.

Deletion of loci. It is possible to delete loci we do not want to analyse. They will not be read, even if they exit in our data.

The second file that the command above produced, the `outputs/loci_counts.txt`, contains a list of the loci found in the genbank file, with the number of their occurrences. This can be used as a guide when deciding which loci to delete and which to keep.

3.2.2 Loading loci to a new Project

Loading `Locus` objects

First we'll make another `Locus` object to make a point that more than one can be read:

```
ssu = Locus('dna','rRNA','18S',['ssu','SSU-rRNA'])
```

Regardless of whether we have one or more `Locus` objects, they are read as a list, which means that they are wrapped with square brackets and separated by commas:

```
loci_list = [coi, ssu]
```

This command will start the `Project` and will write it to the pickle file `outputs/dummy.pkpj`:

```
pj = Project(loci_list, pickle='outputs/dummy.pkpj')
```

This following alternative will start a `Project` and will load the loci from a file `data/edited_loci.csv` that looks like this:

```
dna,rRNA,18s,18S ribosomal RNA,18S rRNA
```

```
dna,rRNA,28s,28S large subunit ribosomal RNA
dna,CDS,MT-CO1,coi,COI,cox1,COX1,coxI
```

```
pj = Project('data/edited_loci.csv',
             pickle='outputs/my_project.pkpj', git=False)
```

This will provoke a bunch of Git related messages which will be discussed in the version control section of this tutorial.
If we print the `Project` we'll get this message:

```
print pj
```

```
Project object with the loci 18s,28s,MT-CO1,
```

3.2.3 Modifying the loci of an existing `Project`

As you have seen, when you start a `Project` you pass a list of loci or a csv file name with the loci attributes:

```
pj = Project(loci_list, pickle='filename')
```

Once the `Project` exists, it is possible to modify the `Locus` objects it contains. To add a `Locus`, you need to create it, as you have done:

```
lsu = Locus('dna', 'rRNA', '28S', ['28s','LSU-rRNA'])
```

and then also add it to the `Project`. Loci are stored in a list called `pj.loci`. So the new `Locus` can be appended to it:

```
pj.loci.append(ssu)
```

or if we have a list of new loci to add, for example:

```
new_loci_list = [nd5, lsu]
```

it can be added to the loci list like so:

```
pj.loci += new_loci_list
```

Lastly, we can modify loci that are already in `pj.loci`. For example, change the name and add an alias to the MT-CO1 `Locus` object:

```
for l in pj.loci:                # Find the Locus named MT-CO1
    if l.name == 'MT-CO1':
        l.name = 'COI'           # Rename it to COI
        l.aliases.append('coi')  # Add the alias coi
```

The last step in any of the sections is to update the pickle file.

```
pickle_pj(pj, 'outputs/my_project.pkpj')
```

```
'outputs/my_project.pkpj'
```

3.2.4 Quick reference

```
# A Locus object
coi = Locus(char_type='dna',          # or 'prot'
            feature_type='CDS',       # any string
            name='MT-CO1',            # any string
            aliases=['coi', 'cox1']) # list of strings

# Guess loci.csv file from a genbank file
list_loci_in_genbank('genbank.gb',
                     'loci.csv',
                     'loci_counts.txt')

# Start a Project
# With a Locus object list
pj = Project([coi, ssu], pickle='pickle_filename')

# With a loci.csv file
pj = Project('loci.csv', pickle='pickle_filename')

# Add a Locus to an existing Project
pj.loci.append(coi)
#Or
pj.loci += [coi]

# Modify a Locus existing in a Project
for l in pj.loci:
    if l.name == 'MT-CO1':
        l.name = 'newName'
        l.feature_type = 'newFeatureType'
        l.char_type = 'prot'
        l.aliases.append('newAlias')
        #Or
        l.aliases += ['newAlias1,newAlias2']
```

3.3. Reading Data

Section quick reference, Full reference

This section in nbviewer

This part will show methods by which to read data into the ReproPhylo `Project`

3.3.1 Reading data from GenBank or EMBL files

GenBank or EMBL files should be the prefered way to read data from online databases because ReproPhylo can store all the associated metadata and make it available for steps such as tree annotation or even Bayestraits analysis. When we pass a GenBank file, only loci and feature types that match the loci we have passed upon creating the Project will be retained, and the rest will be ignored. This is handy for multi-featured GenBank entries that contain any number of genes on top of the ones we are interested in. In this example, only *cox1* CDSs will be read from entries of complete mitochondrial genomes. First we read the pickle file from the section 3.2:

```
from reprophylo import *
pj = unpickle_pj('outputs/my_project.pkpj', git=False)
```

Now we can add data to the `Project`, by reading a list of one or more GenBank files:

```
input_filnames = ['data/Tetillidae1.gb', 'data/Tetillidae2.gb']
pj.read_embl_genbank(input_filnames)
```

```
/home/amir/Dropbox/python_modules/reprophylo.py:1015:
UserWarning: Version control off
  warnings.warn('Version control off')
```

3.3.2 Reading other sequence file formats

When GenBank or EMBL files are read, the accession numbers are used as sequence IDs in ReproPhylo. But when other file formats are used, it is difficult to predict whether a unique sequence ID is available in the sequence header. Therefore, ReproPhylo regards data read from other file formats as 'denovo' and creates denovo sequence IDs. For the same reason, there is no mechanism to prevent you from reading the same file twice, at the moment. All the information found in the original sequence header is retained and made available as metadata. ReproPhylo can handle any format that is compatible with the SeqIO module of Biopython. Reading prealigned sequences is done by a different dedicated method which will be discussed below.

Reading files

In this example we read a fasta file with an unpublished sequence. We will specify the data type (`'dna'`) and the file format. This means that DNA and protein files need to be read in two separate actions.

```
# This list can include one or more file names
denovo_sequence_filenames =
['data/Tetillidae_denovo_sequence.fasta']
pj.read_denovo(denovo_sequence_filenames, 'dna', format='fasta')
```

```
1
```

This is how the 'denovo' record looks like if we ask to print it in GenBank format:

```
for r in pj.records:
    if r.id == 'denovo0': print r.format('genbank')
```

```
LOCUS       denovo0                 2092 bp    DNA             UNK 01-JAN-1980
DEFINITION  NIWA2850 Craniella microsigma cox1
ACCESSION   denovo0
VERSION     denovo0
KEYWORDS    .
SOURCE      .
  ORGANISM  .
            .
FEATURES             Location/Qualifiers
     source          1..2092
                     /feature_id="denovo0_source"
                     /original_id="NIWA2850"
                     /original_desc="Craniella microsigma cox1"
ORIGIN
        1 atgataggaa ctggatttag cttgcttatt agattagaac tatccgctcc cggattaatg
       61 ttgggtgacg accatttata caatgttatg gtcacggccc acggtcttat aatggtcttt
      121 ttcttagtta tgccggttat gataggtggg ttcggtaatt gaatggttcc cctttacatc
      181 ggggcaccgg atatggcttt tccaagatta aacaatatta gtttttgagt tttacccccc
      241 tcattaatac tactgctagg ttctgctttt gttgaacaag gggttgggac aggatggacc
      301 ctttatccac cattatcaag tatacaggct cattctgggg gctcagtcga tgcggcaatt
      361 tttagtcttc atttggctgg gatctcttca attttagggg caatgaattt tataactact
      421 atctttaata tgcgggcacc tgggattacc atggatagat tgcctctatt tgtttgatct
      481 attttaataa caacttattt gttattatta gctttgccag tattggctgg tgccataact
      541 atgcttttaa cagatagaaa tttcaataca acgttcttcg atcccgctgg tggtggggac
      601 ccaatattat ttcaacattt attttggttc tttgggcatc cggaagttta tgtactagtt
      661 ctccccgggt ttggaattgt ttctcagatt attccaacat tcgcggctaa aaaacaaata
      721 tttggctatc tagggatggt ctatgctatg gtttctatag gaattttagg ttttatagtt
      781 tgagcttgta gatgggcgtg cgatagagtg atctatcgta gtataacatg actgtatgct
      841 ggaaagccta aaaaaagaaa ttcattaatt actcgtaatg acaggttcga tacagtaaaa
      901 atattaatgg agggtcaatc agcaggcaac ggtatagttt atactggagc ctcagagact
      961 acacgtcatg cccttgagga tgatttatat tgagctattg gtttatttga ggccgaagga
     1021 accttaaaga taagtaaggg tcggatctat attagtgcgt gtcaatcgac tagtaatata
     1081 aaggtccttt accgaataaa gggaatattt tgtttagggg gcgttaaaat aagaaaagac
     1141 ccccgttata gtgattgaaa gttagggagc gatttaaata aaatagtaaa attattagtt
     1201 tattaatgga gactaatcac tagaaagaaa aatatacaat tagtagagtt gataaagttc
     1261 ataaattgta aatattttcc taatttggtt gaatactttg gtttagagta ttgaccccga
     1321 ataatgcttg attaactggg tttgttgagg gagatggaaa cttaaatatt cagataagac
     1381 cacaccagtg gcggcccgaa tttcgattac acaaaaagag agagatgttt agatttaatc
     1441 aatgatattt ttcctgggtc catttgggct tcaggcaatc catcagaaca ctttaaatat
     1501 tcggcggggt caataagaac tcgaagtgac tggataaaat attttactag gtatccattt
     1561 aaggggaata aaaatattca atatgtgcgt tggttgaaat gccataatat tgttattcaa
     1621 ggtctacaca aaaccgagaa ggggttagct caaattaaat caatttggac tcaaggtgaa
     1681 gatatagtcc aatcccctta gtaatagggg ggtataacac gattgagtgt tgtaatttaa
     1741 gcatcacatg tttacagttg gaatggatgc cgactctagg gcatacttta gcgctgcaac
     1801 gatgataatc gccgtaccaa ccggaataaa aatctttagt tggatcgcta cagtagtagg
     1861 gggctcattg agaatagata ctcctatgtt atgggctatg ggatttgttt ttttatttac
     1921 tgtaggagga ttaaccggaa ttgtggtagc aagtaattct ttagatgtgt tgctccacga
     1981 cacatattac gttgttgctc attttcatta tgttctatcc atgggggcta tctttgctat
     2041 ctttggaggg gtttattatt gatttggtaa aattactggt tattgttaca ac
//
```

The record was assigned the ID `'denovo0'`, and a `'source'` feature was created, including the fasta header as the `'original_id'` and `'original_desc'` qualifiers.

However, it has no feature to indicate what locus it is and it will be ignored down the line. It is now up to us to add such a feature. Note that for large scale data, such as Exonerate results, other methods apply and will be discussed later.

Adding features

Here we only have one new sequence and we know its ID - `'denovo0'` so it is easy enough to add a feature:

```
pj.add_feature_to_record('denovo0', 'CDS', qualifiers={'gene':
'cox1'})
```

```
'denovo0_f0'
```

Feature `'denovo0_f0'` was created.
Often we would want to assign gene names to a whole lot of sequences based on one name we recognize in the fasta header. We can create a dictionary that will specify the gene and feature type of each sequence:

```
feature_lookup = {'NIWA2850': ['CDS','cox1'],
                      # If we had more sequences
                      # we would add them here:
                      # 'Seq2': ['18S', 'rRNA'],
                      # ...
                     }
```

Now we can use this dictionary to create the feature:

```
for r in pj.records:

    source = r.features[0]
    quals = source.qualifiers

    if ('original_id' in quals and
        quals['original_id'][0] in feature_lookup):

        original_id = quals['original_id'][0]
        feature_type = feature_lookup[original_id][0]
        gene = feature_lookup[original_id][1]
        pj.add_feature_to_record(r.id,
                                 feature_type,
                                 qualifiers={'gene': gene})
```

The `add_feature_to_record` method allows to limit the feature to just a part of the sequence and to add any number of qualifiers. Look it up in the module reference.
This is how the record looks now, with the new feature added:

```
for r in pj.records:
    if r.id == 'denovo0': print r.format('genbank')
```

```
LOCUS       denovo0                 2092 bp    DNA             UNK 01-JAN-1980
DEFINITION  NIWA2850 Craniella microsigma cox1
ACCESSION   denovo0
VERSION     denovo0
KEYWORDS    .
SOURCE      .
  ORGANISM  .
            .
FEATURES             Location/Qualifiers
     source          1..2092
                     /feature_id="denovo0_source"
                     /original_id="NIWA2850"
                     /original_desc="Craniella microsigma cox1"
     CDS             1..2092
                     /feature_id="denovo0_f0"
                     /GC_content="37.3804971319"
                     /gene="cox1"
                     /nuc_degen_prop="0.0"
ORIGIN
        1 atgataggaa ctggatttag cttgcttatt agattagaac tatccgctcc cggattaatg
       61 ttgggtgacg accatttata caatgttatg gtcacggccc acggtcttat aatggtcttt
      121 ttcttagtta tgccggttat gataggtggg ttcggtaatt gaatggttcc cctttacatc
      181 ggggcaccgg atatggcttt tccaagatta aacaatatta gtttttgagt tttacccccc
      241 tcattaatac tactgctagg ttctgctttt gttgaacaag gggttgggac aggatggacc
      301 ctttatccac cattatcaag tatacaggct cattctgggg gctcagtcga tgcggcaatt
      361 tttagtcttc atttggctgg gatctcttca attttagggg caatgaattt tataactact
      421 atctttaata tgcgggcacc tgggattacc atggatagat tgcctctatt tgtttgatct
      481 attttaataa caacttattt gttattatta gctttgccag tattggctgg tgccataact
      541 atgcttttaa cagatagaaa tttcaataca acgttcttcg atcccgctgg tggtggggac
      601 ccaatattat ttcaacattt attttggttc tttgggcatc cggaagttta tgtactagtt
      661 ctcccccggt ttggaattgt ttctcagatt attccaacat tcgcggctaa aaaacaaata
      721 tttggctatc tagggatggt ctatgctatg gtttctatag gaattttagg ttttatagtt
      781 tgagcttgta gatgggcgtg cgatagagtg atctatcgta gtataacatg actgtatgct
      841 ggaaagccta aaaaaagaaa ttcattaatt actcgtaatg acaggttcga tacagtaaaa
      901 atattaatgg agggtcaatc agcaggcaac ggtatagttt atactggagc ctcagagact
      961 acacgtcatg cccttgagga tgatttatat tgagctattg gtttatttga ggccgaagga
     1021 accttaaaga taagtaaggg tcggatctat attagtgcgt gtcaatcgac tagtaatata
     1081 aaggtccttt accgaataaa gggaatattt tgtttagggg gcgttaaaat aagaaaagac
     1141 ccccgttata gtgattgaaa gttagggagc gatttaaata aaatagtaaa attattagtt
     1201 tattaatgga gactaatcac tagaaagaaa aatatacaat tagtagagtt gataaagttc
     1261 ataaattgta aatattttcc taatttggtt gaatactttg gtttagagta ttgaccccga
     1321 ataatgcttg attaactggg tttgttgagg gagatggaaa cttaaatatt cagataagac
     1381 cacaccagtg gcggcccgaa tttcgattac acaaaaagag agagatgttt agatttaatc
     1441 aatgatattt ttcctgggtc catttgggct tcaggcaatc catcagaaca ctttaaatat
     1501 tcggcggggt caataagaac tcgaagtgac tggataaaat attttactag gtatccattt
     1561 aaggggaata aaaatattca atatgtgcgt tggttgaaat gccataatat tgttattcaa
     1621 ggtctacaca aaaccgagaa ggggttagct caaattaaat caatttggac tcaaggtgaa
     1681 gatatagtcc aatcccctta gtaatagggg ggtataacac gattgagtgt tgtaatttaa
     1741 gcatcacatg tttacagttg gaatggatgc cgactctagg gcatacttta gcgctgcaac
     1801 gatgataatc gccgtaccaa ccggaataaa aatctttagt tggatcgcta cagtagtagg
     1861 gggctcattg agaatagata ctcctatgtt atgggctatg ggatttgttt ttttatttac
     1921 tgtaggagga ttaaccggaa ttgtggtagc aagtaattct ttagatgtgt tgctccacga
     1981 cacatattac gttgttgctc attttcatta tgttctatcc atgggggcta tctttgctat
     2041 ctttggaggg gtttattatt gatttggtaa aattactggt tattgttaca ac
//
```

Through the qualifiers dictionary, we can also attempt to add a translation of the sequence. We can also define a location for the feature, as a subset of the whole sequence :

```
qualifiers={'gene': 'cox1',
            'transl_table': 4,
            'codon_start': 1,
            'organism': 'Craniella microsigma'}


for record in pj.records:
   if 'denovo' in record.id: # New sequences are assigned
                             # with IDs starting
                             # with 'denovo'
       pj.add_feature_to_record(record.id, 'CDS',
                     # The location is specified as a list
                     # of lists. Every sub-list is an exon
                     # and has the start, the end and the strand.
                     # The numbers are "real" positions and not
                     # machine. ie, counting starts from 1.
                               location=[[1,786,1],[1742,2092,1]],
                               qualifiers=qualifiers)
```

`transl_table` is the genetic code to use in order to translate the coding sequence into a protein. The number, 4 in this case, specify the table to use, out of the GenBank genetic code tables.

3.3.3 Reading sequence alignments

ReproPhylo allows to read prealigned sequences in any of the Biopython AlignIO compatible formats, as follows:

```
pj.read_alignment('Another_locus.nex', 'dna', 'CDS', 'ND5',
                  format='nexus')
```

This will place the alignment in the `Project.alignments` attribute (`pj.alignments` in this case) and the unaligned sequences as records in `Project.records`. There must be a `Locus` object in `pj.loci`, that is compatible with the character (dna) feature type (CDS) and the locus name (ND5) specified in the `read_alignmnet` command. The records will be assigned 'denovo' IDs, and the nexus sequence names will be stored in the 'original_id' qualifiers. 'original_desc' qualifier remain empty in this case, because nexus files don't have them.

3.3.4 Reading a Nexus alignment with PAUP commands

Many published datasets are available in nexus format with charset commands that describe the data partitions. ReproPhylo can read such a matrix, split the partitions into individual alignments and place them in `Project.alignments`, and then put each sequence from each partition in `Project.records`. This facilitates experimentation with the data composition. It is even possible to turn such a nexus file directly into a new `Project` instance with all the information set up. To do that use the following command:

```
nexus_filename = 'data/some_supermatrix.nex'

pj = pj_from_nexus_w_charset(nexus_filename,

                             'data',
                             # path to write intermediate fasta file
                             'dna',
                             # Character type ('dna' or 'prot')
                             'CDS',
                             # Feature type (Any)
                             project = True,
                             # Will return a Project instance
                             # instead of a list
                             # of fasta files per partition
                             # if project will save it to this file:
                             pickle = 'new_pickle_name',
                             git = True)
                             # Will start and manage repository
```

To finish the section we'll update the `Project`'s pickle file:

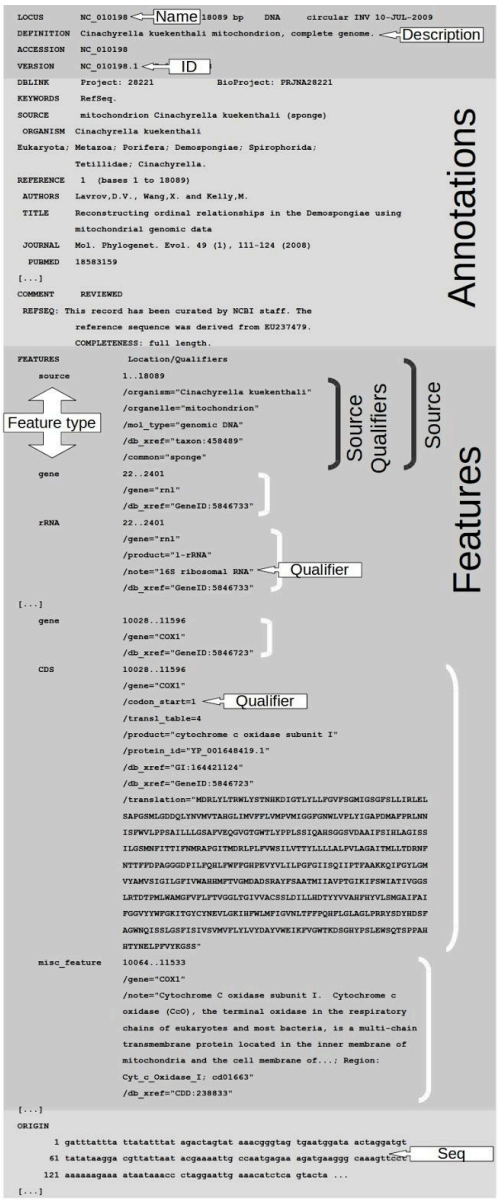
```
pickle_pj(pj, 'outputs/my_project.pkpj')
```

```
'outputs/my_project.pkpj'
```

3.3.5 Quick reference

```
# Read GenBank or embl files
input_filnames = ['file1', 'file2']
pj.read_embl_genbank(input_filnames)

# Read other formats
denovo_sequence_filenames = ['file1.fasta', 'file2.fasta']
pj.read_denovo(denovo_sequence_filenames, 'dna', format='fasta')
#Or
pj.read_denovo(denovo_sequence_filenames, 'prot', format='fasta')

# Add asequence feature to a record
pj.add_feature_to_record('someRecordID', 'CDS',
                         qualifiers={'gene': 'cox1'})
# Or
qualifiers={'gene': 'cox1',
            'transl_table': 4,
            'codon_start': 1,
            'organism': 'Craniella microsigma'}

pj.add_feature_to_record(someRecordID, 'CDS',
                         location=[[1,786,1],[1742,2092,1]],
```

```
                                qualifiers=qualifiers)

# Read a sequence alignment
pj.read_alignment('Another_locus.nex', 'dna', 'CDS', 'ND5',
format='nexus')

# Read a Nexus alignment with a super-matrix and charset commands
nexus_filename = 'data/some_supermatrix.nex'
pj = pj_from_nexus_w_charset(nexus_filename,
                             'data',
                             'dna',
                             'CDS',
                             project = True,
                             pickle = 'new_pickle_name',
                             git = True)
```



3.4. Metadata handling

Section quick reference

This section reviews the various methods for reading and modifying metadata within a ReproPhylo `Project`. Utilizing it will be discussed in later sections.

3.4.1 What is metadata in ReproPhylo?

Within a ReproPhylo `Project`, metadata is tied to sequences and sequence features, in the Biopython sense. Since sequence records in the `Project` are in fact Biopython `SeqRecord` objects, a quick review of the GenBank file format, based on which the `SeqRecord` class is structured, will help understand basic concepts.

When a data file is read, each sequence will be stored as a `SeqRecord` object in the `Project.records` list of `SeqRecord` objects. A `SeqRecord` object has an annotations attribute (`SeqRecord.annotations`) which is a Python dictionary containing information regarding the sequence as a whole. Additional `SeqRecord` attribute is the features Python list (`SeqRecord.features`). Features in the list are stored as `SeqFeature` Biopython objects, and they define the start and end of each locus in the sequence (`SeqFeature.location` attribute. The `SeqFeature` also has the `SeqFeature.qualifiers` dictionary, which holds any additional metadata about this sequence feature. For example, the gene name and product name. This information is used by ReproPhylo to sort loci into their respective bins, (eg, coi, 18S etc.). Note that the values in the

`SeqFeature.qualifiers` dictionary are always stored as Python lists, even if they consist of a single value. For example, a `SeqFeature.qualifiers` dictionary might look like this:

```
{'gene': ['cox1'],
'translation': ['AATRNLLK']}
```

Another important `SeqRecord` attribute is type (`SeqRecord.type`), which is a string stating the feature type, whether it is `'gene'`, `'CDS'`, `'rRNA'` or anything else.
A special `SeqFeature` is the source feature (`SeqFeature.type == 'source'`). It is the first feature in each `SeqRecord.features` list, and is generated automatically by ReproPhylo if you read a file that does not have such features (eg, fasta format). The qualifiers dictionary of this automatically generated source feature will then contain the `original_id` and `original_desc` (description) from the fasta headers. Technically, there is absolutely no difference between the source feature and all the other features. However, conceptually, metadata stored in the source feature applies to all the other features. ReproPhylo knows this and provides tools to access it accordingly (further down).

For a more detailed description of metadata in the `SeqRecord` Biopython object, refer to this section in the Biopython tutorial. Although ReproPhylo provides some `Project` methods for modifying the metadata, and also a method to edit the metadata in a spreadsheet, the most flexible way to do it is by utilizing Biopython code, and mastering it is helpful within ReproPhylo and in life in general!

3.4.2 Modifying the metadata

A Biopython example

With Biopython we can iterate over the records and their features in the `pj.records` list and make changes or additions to the qualifiers of each feature as follows. To get a working example going, first we load our `Project` with its loci and data:

```
from reprophylo import *
pj = unpickle_pj('outputs/my_project.pkpj', git=False)
```

The first record looks like this:

```
print pj.records[0].format('genbank')
```

```
LOCUS       KC902343               1728 bp    DNA              INV 05-SEP-2013
DEFINITION  Cinachyrella cf. paterifera 0M9H2022-P small subunit 18S ribosomal
            RNA gene, partial sequence.
ACCESSION   KC902343
VERSION     KC902343.1  GI:511637204
KEYWORDS    .
SOURCE      Cinachyrella cf. paterifera 0M9H2022-P
  ORGANISM  Cinachyrella cf. paterifera 0M9H2022-P
            Eukaryota; Metazoa; Porifera; Demospongiae; Tetractinomorpha;
            Spirophorida; Tetillidae; Cinachyrella.
REFERENCE   1  (bases 1 to 1728)
  AUTHORS   Redmond,N.E., Morrow,C.C., Thacker,R.W., Diaz,M.C.,
            Boury-Esnault,N., Cardenas,P., Hajdu,E., Lobo-Hajdu,G., Picton,B.E.,
            Pomponi,S.A., Kayal,E. and Collins,A.G.
  TITLE     Phylogeny and Systematics of Demospongiae in Light of New
```

```
            Small-Subunit Ribosomal DNA (18S) Sequences
  JOURNAL   Integr. Comp. Biol. 53 (3), 388-415 (2013)
   PUBMED   23793549
REFERENCE   2  (bases 1 to 1728)
  AUTHORS   Redmond,N. and Collins,A.G.
  TITLE     Direct Submission
  JOURNAL   Submitted (16-APR-2013) National Systematics Lab and Smithsonian,
            10th and Constitution NW, Washington, DC, USA
FEATURES             Location/Qualifiers
     source          1..1728
                     /note="PorToL ID: NCI376"
                     /mol_type="genomic DNA"
                     /country="Australia"
                     /feature_id="KC902343.1_source"
                     /db_xref="taxon:1342549"
                     /specimen_voucher="0M9H2022-P"
                     /organism="Cinachyrella cf. paterifera 0M9H2022-P"
     rRNA            <1..>1728
                     /feature_id="KC902343.1_f0"
                     /product="small subunit 18S ribosomal RNA"
                     /nuc_degen_prop="0.0"
                     /GC_content="52.0833333333"
ORIGIN
        1 gtctcaaaga ctaagccatg catgtccaag tatgaacgct tcgtactgtg aaactgcgaa
       61 tggctcatta aatcagttat agtttatttg atggttgctt gctacatgga taaccgtggt
      121 aattctagag ctaatacatg cacagagtcc cgacttccgg gagggacgta tttattagat
      181 ccaaaaccag cgcgggtgtc ccctcgtggg tgcccggtcc ctgggcgatt catgataact
      241 gctcgaatcg cacggccctg gcgccggcga tggtccattc aaatttctgc cctatcaact
      301 ttcgatggta cggtagtggc ctaccatggt tgcaacgggt gacggagaat tagggttcga
      361 ttccggagag ggagcctgag aaacggctac cacatccaag gaaggcagca ggcgcgcaaa
      421 ttacccaatc ccgactcggg gaggtagtga caataaataa caatgccggg ctctcgcagt
      481 ctggcaattg gaatgagtcc aatctaaacc ccttaacgag gaacaattgg agggcaagtc
      541 tggtgccagc agccgcggta attccagctc caatagcgta tattaaagtt gttgcagtta
      601 aaaagctcgt agttggattt cggggcggcc cggccggtcc gccgcgaggc gagcactggt
      661 cgggcgccct tcctctcgaa ggcttcgact gctcttgatt gcggtggtcg aggagttcgg
      721 gacgtttact ttgaaaaaat tagagtgttc aaggcaggcc gtcgcctgaa tacattagca
      781 tggaataatg gaagaggacc tcggtcctat tttgttggtt tccagggccg aagtaatgat
      841 taagagggac agttgggggc attcgtattc aattgtcaga ggtgaaattc tcggatttat
      901 ggaagacgaa caagtgcgaa agcatttgcc aaggatgttt tcattaatca agaacgaaag
      961 ttgggggttc gaagacgatc agataccgtc gtagtcccaa ccataaacta tgccgactag
     1021 ggatcggcgg atgttagcgt ctgactccgt cggcaccttg cgagaaatca agagtctttg
     1081 ggttccgggg ggagtatggt cgcaaggctg aaacttaaag gaattgacgg aagggcacca
     1141 ccaggagtgg agcctgcggc ttaatttgac tcaacacggg gaaactcacc aggtccggac
     1201 atggtaagga ttgacagatc gagagctctt tcttgattct atgggtggtg gtgcatggcc
     1261 gttcttagtt ggtggagtga tttgtctggt taattccgtt aacgaacgag accttaacct
     1321 gctaactagt cacgccgttc ccgaacggcg ggcgacttct tagagggaca accggccccg
     1381 aagccggcgg aagtctgagg caataacagg tctgtgatgc ccttagatgt tctgggccgc
     1441 acgcgcgcta cactgacgga ggcagcgagc atgtccttcg ccgagaggtg cggggaatct
     1501 tgtgaaactc cgtcgtgctg gggatagatc attgcaattc tcgatcttga acgaggaatt
     1561 cctagtaagc gcgagtcagc agctcgcgtt gattacgtcc ctgccctttg tacacaccgc
     1621 ccgtcgctac taccgattga atggtttagt gagatcttcg gattggagcc gccgtgacgg
     1681 gcgaccgccg cggcggattt cgagaagtcg atcaaacttg atcattta
//
```

The next cell will get the organism name for each record, which is a qualifier in the source feature. It will then get the genus out of this name, and place it in a new qualifier, in all the record's features:

```
for record in pj.records:

    # get the source qualifiers
    source_feature = record.features[0]
```

```
    source_qualifiers = source_feature.qualifiers

    # get the species name
    species = None
    if 'organism' in source_qualifiers:
        species = source_qualifiers['organism'][0]
        # qualifier values are lists

    # place the genus as a qualifier in all the features
    if species:
        genus = species.split()[0]
        for f in record.features:
            f.qualifiers['genus'] = [genus]
```

The genus qualifier was added to all the features.

```
print pj.records[0].format('genbank')
```

```
LOCUS       KC902343               1728 bp    DNA              INV 05-SEP-2013
DEFINITION  Cinachyrella cf. paterifera 0M9H2022-P small subunit 18S ribosomal
            RNA gene, partial sequence.
ACCESSION   KC902343
VERSION     KC902343.1  GI:511637204
KEYWORDS    .
SOURCE      Cinachyrella cf. paterifera 0M9H2022-P
  ORGANISM  Cinachyrella cf. paterifera 0M9H2022-P
            Eukaryota; Metazoa; Porifera; Demospongiae; Tetractinomorpha;
            Spirophorida; Tetillidae; Cinachyrella.
REFERENCE   1  (bases 1 to 1728)
  AUTHORS   Redmond,N.E., Morrow,C.C., Thacker,R.W., Diaz,M.C.,
            Boury-Esnault,N., Cardenas,P., Hajdu,E., Lobo-Hajdu,G., Picton,B.E.,
            Pomponi,S.A., Kayal,E. and Collins,A.G.
  TITLE     Phylogeny and Systematics of Demospongiae in Light of New
            Small-Subunit Ribosomal DNA (18S) Sequences
  JOURNAL   Integr. Comp. Biol. 53 (3), 388-415 (2013)
   PUBMED   23793549
REFERENCE   2  (bases 1 to 1728)
  AUTHORS   Redmond,N. and Collins,A.G.
  TITLE     Direct Submission
  JOURNAL   Submitted (16-APR-2013) National Systematics Lab and Smithsonian,
            10th and Constitution NW, Washington, DC, USA
FEATURES             Location/Qualifiers
     source          1..1728
                     /note="PorToL ID: NCI376"
                     /mol_type="genomic DNA"
                     /country="Australia"
                     /feature_id="KC902343.1_source"
                     /db_xref="taxon:1342549"
                     /specimen_voucher="0M9H2022-P"
                     /genus="Cinachyrella"
                     /organism="Cinachyrella cf. paterifera 0M9H2022-P"
     rRNA            <1..>1728
                     /feature_id="KC902343.1_f0"
                     /product="small subunit 18S ribosomal RNA"
                     /genus="Cinachyrella"
                     /nuc_degen_prop="0.0"
                     /GC_content="52.0833333333"
ORIGIN
        1 gtctcaaaga ctaagccatg catgtccaag tatgaacgct tcgtactgtg aaactgcgaa
```

```
       61 tggctcatta aatcagttat agtttatttg atggttgctt gctacatgga taaccgtggt
      121 aattctagag ctaatacatg cacagagtcc cgacttccgg gagggacgta tttattagat
      181 ccaaaaccag cgcgggtgtc ccctcgtggg tgcccggtcc ctgggcgatt catgataact
      241 gctcgaatcg cacggccctg gcgccggcga tggtccattc aaatttctgc cctatcaact
      301 ttcgatggta cggtagtggc ctaccatggt tgcaacgggt gacggagaat tagggttcga
      361 ttccggagag ggagcctgag aaacggctac cacatccaag gaaggcagca ggcgcgcaaa
      421 ttacccaatc ccgactcggg gaggtagtga caataaataa caatgccggg ctctcgcagt
      481 ctggcaattg gaatgagtcc aatctaaacc ccttaacgag gaacaattgg agggcaagtc
      541 tggtgccagc agccgcggta attccagctc caatagcgta tattaaagtt gttgcagtta
      601 aaaagctcgt agttggattt cggggcggcc cggccggtcc gccgcgaggc gagcactggt
      661 cgggcgccct tcctctcgaa ggcttcgact gctcttgatt gcggtggtcg aggagttcgg
      721 gacgtttact ttgaaaaaat tagagtgttc aaggcaggcc gtcgcctgaa tacattagca
      781 tggaataatg gaagaggacc tcggtcctat tttgttggtt tccagggccg aagtaatgat
      841 taagagggac agttgggggc attcgtattc aattgtcaga ggtgaaattc tcggatttat
      901 ggaagacgaa caagtgcgaa agcatttgcc aaggatgttt tcattaatca agaacgaaag
      961 ttgggggttc gaagacgatc agataccgtc gtagtcccaa ccataaacta tgccgactag
     1021 ggatcggcgg atgttagcgt ctgactccgt cggcaccttg cgagaaatca agagtctttg
     1081 ggttccgggg ggagtatggt cgcaaggctg aaacttaaag gaattgacgg aagggcacca
     1141 ccaggagtgg agcctgcggc ttaatttgac tcaacacggg gaaactcacc aggtccggac
     1201 atggtaagga ttgacagatc gagagctctt tcttgattct atgggtggtg gtgcatggcc
     1261 gttcttagtt ggtggagtga tttgtctggt taattccgtt aacgaacgag accttaacct
     1321 gctaactagt cacgccgttc ccgaacggcg ggcgacttct tagagggaca accggccccg
     1381 aagccggcgg aagtctgagg caataacagg tctgtgatgc ccttagatgt tctgggccgc
     1441 acgcgcgcta cactgacgga ggcagcgagc atgtccttcg ccgagaggtg cggggaatct
     1501 tgtgaaactc cgtcgtgctg gggatagatc attgcaattc tcgatcttga acgaggaatt
     1561 cctagtaagc gcgagtcagc agctcgcgtt gattacgtcc ctgccctttg tacacaccgc
     1621 ccgtcgctac taccgattga atggtttagt gagatcttcg gattggagcc gccgtgacgg
     1681 gcgaccgccg cggcggattt cgagaagtcg atcaaacttg atcattta
//
```

Important:
In addition to the record ID, ReproPhylo assigns a unique *feature ID* for each feature within the record. In a record with a record ID `KC902343`, the ID of the first feature will be `KC902343_source`, for the second and third features the IDs will be `KC902343_f0` and `KC902343_f1` and so on. For a record with a record ID `denovo2`, the features will get the feature IDs `denovo2_source`, `denovo2_f0`, `denovo2_f1` and so on. This is important because it allows to access specific features directly (say, the cox1 features), using their feature ID.

Some ReproPhylo shortcuts

ReproPhylo adds some basic shortcuts for convenience. Here are some examples:

```
pj.if_this_then_that('Cinachyrella', 'genus', 'yes',
                     'porocalices', mode='part')
```

This method iterates over all the records in the `Project` and makes some changes where the rules provided to it apply. If the **value 'Cinachyrella'** is found in the **qualifier 'genus'** it will put the **value 'yes'** in the **qualifier 'porocalices'**, which is a morphological trait of the sponge genus *Cinachyrella*. **mode='part'** means that the match can be partial. The default is mode='whole'.

```
features_to_modify = ['KC902343.1_f0', 'JX177933.1_f0']
pj.add_qualifier(features_to_modify, 'spam?', 'why not')
```

This method will add a qualifier **spam?** with the value **why not** to specific features that have the feature IDs **KC902343.1_f0** and **JX177933.1_f0**. Within each record, ReproPhylo

assigns unique ID for each feature.

```
pj.add_qualifier_from_source('country')
```

We may want to place the source qualifier **'country'** explicitly in each of the other features in the record. The **add_qualifier_from_source** method will take effect in all the records that have a country qualifier in their source feature. It will copy it to all the other features, along with its value in that record.

```
pj.copy_paste_from_features_to_source('eggs?', 'spam?')
```

Or vice-versa, we can make sure that a qualifier that is in only one of the features, is copied as a value of a source feature qualifier and thus apply it to the whole record (and all its features). In this case, the function **copy_paste_from_features_to_source** will take effect in records where at least one non-source record feature has the qualifier **eggs?**, and it will copy the value of **eggs?** to the source qualifier **spam?**.

```
pj.copy_paste_within_feature('GC_content', '%GC')
```

Lastly, we may want to equate qualifiers that have different names in different records, but are essentially the same thing. For example, 'sample' and 'voucher'. This can be done by applying the qualifier name of one of them to the other, using the method **copy_paste_within_feature**. In every record feature that has the qualifier **GC_content**, a new qualifier will be created, **%GC**, and it will contain the value of **GC_content**.
This is the FEATURES section of the above record, with the resulting changes to it. The method which is responsible for each qualifier is indicated next to it (the method names are not a part of the real output):

```
FEATURES             Location/Qualifiers
     source          1..1728
                     /feature_id="KC902343.1_source"
                     /mol_type="genomic DNA"
                     /country="Australia"
                     /eggs?="why not" #### copy_paste_from_features_to_source
                     /note="PorToL ID: NCI376"
                     /db_xref="taxon:1342549"
                     /specimen_voucher="0M9H2022-P"
                     /genus="Cinachyrella" #### Biopython script from section
3.4.2.1
                     /organism="Cinachyrella cf. paterifera 0M9H2022-P"
     rRNA            <1..>1728
                     /porocalices="yes" ####if_this_then_that
                     /product="small subunit 18S ribosomal RNA"
                     /country="Australia" #### add_qualifier_from_source
                     /nuc_degen_prop="0.0"
                     /feature_id="KC902343.1_f0"
                     /spam?="why not" #### add_qualifier
                     /%GC="52.0833333333" #### copy_paste_within_feature
                     /GC_content="52.0833333333"
                     /genus="Cinachyrella" #### Biopython script from section
3.4.2.1
```

Using a spreadsheet

ReproPhylo provides an alternative route for metadata editing that goes through a spreadsheet. This way, the spreadsheet can be routinely edited and the changes read into

the `Project` and propagated to its existing components (eg, trees). The best way to edit this spreadsheet probably goes through pandas, if you are familiar with it. Otherwise, it is possible to edit and save in excel, libreoffice and similar programmes, although beware of errors.
In this section I will give an example using a spreadsheet programme. This example will add the qualifier 'monty' and the value 'python' to each *source* feature, and the qualifier 'holy' with the value 'grail' to each *non-source* feature.
The first step is to write a csv file (the separators are actually tabs and not commas)

```
pj.write('outputs/metadata_example.tsv', format='csv')
```

In the resulting file, each ***feature*** has its own line, and each record has as many lines as non-source features it contains. Source feature qualifiers are included in all the lines, as they apply to all the features in the record. They are indicated in the titles with the prefix source:_. To add a qualifier to the source feature, we will need to use this prefix in its title.
I have opened this file in a spreadsheet programme and added the qualifiers as follows:

feature_id	source:_monty	holy	spam?	%GC	G
KC902343.1_f0	python	grail	why not	52.0833333333	
KC902290.1_f0	python	grail	null	51.9740718916	
KC902265.1_f0	python	grail	null	50.7246376812	
KC902264.1_f0	python	grail	null	51.8818760857	
KC902195.1_f0	python	grail	null	51.566469094	
KC902189.1_f0	python	grail	null	51.9397799653	
KC902108.1_f0	python	grail	null	52.0894643908	
KC902033.1_f0	python	grail	null	52.0396270396	
KC901899.1_f0	python	grail	null	51.9397799653	
JX177987.1_f0	python	grail	null	49.9097472924	
JX177986.1_f0	python	grail	null	52.1767810026	
JX177985.1_f0	python	grail	null	50.487804878	
JX177984.1_f0	python	grail	null	51.8806744488	
JX177983.1_f0	python	grail	null	50.9895227008	
JX177982.1_f0	python	grail	null	51.7354289456	
JX177981.1_f0	python	grail	null	51.9736842105	
JX177980.1_f0	python	grail	null	51.6547696301	
JX177979.1_f0	python	grail	null	51.6547696301	
JX177978.1_f0	python	grail	null	51.6547696301	
JX177976.1_f0	python	grail	null	51.5249837768	
JX177975.1_f0	python	grail	null	51.5249837768	
JX177974.1_f0	python	grail	null	49.7424576895	
JX177973.1_f0	python	grail	null	52.1739130435	
JX177972.1_f0	python	grail	null	51.9050593379	
JX177971.1_f0	python	grail	null	52.123327516	
JX177970.1_f0	python	grail	null	51.9556333917	
JX177969.1_f0	python	grail	null	51.9230769231	
JX177968.1_f0	python	grail	null	52.4441762221	
JX177967.1_f0	python	grail	null	52.0979020979	
JX177966.1_f0	python	grail	null	52.5766871166	

The edited spreadsheet was saved as `outputs/edited_metadata_example.tsv`, and it can now be read back to the `Project`

```
pj.correct_metadata_from_file('outputs/edited_metadata_example.tsv')

# Propagate the changes so they are also updated in tree leaves.
pj.propagate_metadata()
```

If we print the first record again, this is how its FEATURES section looks now:

```
FEATURES             Location/Qualifiers
     source          1..1728
                     /note="PorToL ID: NCI376"
                     /mol_type="genomic DNA"
                     /country="Australia"
                     /organism="Cinachyrella cf. paterifera
0M9H2022-P"
                     /feature_id="KC902343.1_source"
                     /db_xref="taxon:1342549"
                     /specimen_voucher="0M9H2022-P"
                     /genus="Cinachyrella"
                     /eggs?="why not"
                     /monty="python"  #### New source qualifier
     rRNA            <1..>1728
                     /porocalices="yes"
                     /product="small subunit 18S ribosomal RNA"
                     /holy="grail"    #### New non-source qualifier
                     /country="Australia"
                     /nuc_degen_prop="0"
                     /feature_id="KC902343.1_f0"
                     /%GC="52.0833333333"
                     /spam?="why not"
                     /record_id="KC902343.1"
                     /GC_content="52.0833333333"
                     /genus="Cinachyrella"
```

To finish this section, we'll update the pickle file:

```
pickle_pj(pj, 'outputs/my_project.pkpj')
```

```
'outputs/my_project.pkpj'
```

3.4.3 Quick reference

```
## A Biopython example
for record in pj.records:

    # get the source qualifiers
    source_feature = record.features[0]
    source_qualifiers = source_feature.qualifiers
```

```
    # get the species name
    species = None
    if 'organism' in source_qualifiers:
        # qualifier values are lists
        species = source_qualifiers['organism'][0]

    # place the genus as a qualifier in all the features
    if species:
        genus = species.split()[0]
        for f in record.features:
            f.qualifiers['genus'] = [genus]

## Add qualifier based on condition
pj.if_this_then_that('Cinachyrella', 'genus', 'yes', 'porocalices',
                     mode='part')

## Modify qualifier of specific features
features_to_modify = ['KC902343.1_f0', 'JX177933.1_f0']
pj.add_qualifier(features_to_modify, 'spam?', 'why not')

## Copy qualifier from source to features
pj.add_qualifier_from_source('country')
# or vice-versa
pj.copy_paste_from_features_to_source('spam?', 'eggs?')

## Duplicate a qualifier with a new name
pj.copy_paste_within_feature('GC_content', '%GC')

## Write metadata spreadsheet
pj.write('outputs/metadata_example.tsv', format='csv')

# Read a corrected metadata spreadsheet
pj.correct_metadata_from_file('outputs/edited_metadata_example.tsv')

# Propagate the changes
pj.propagate_metadata()
```

3.5. Pre alignment filtering

Section quick reference, Full reference

This section in nbviewer

This section is a walk through the pre-alignment sequence filtering in ReproPhylo. We will start by several preliminaries discussed in the previous sections:

```
from reprophylo import *

pj = unpickle_pj('outputs/my_project.pkpj', git=False)
```

3.5.1 Filtering by sequence length or GC content

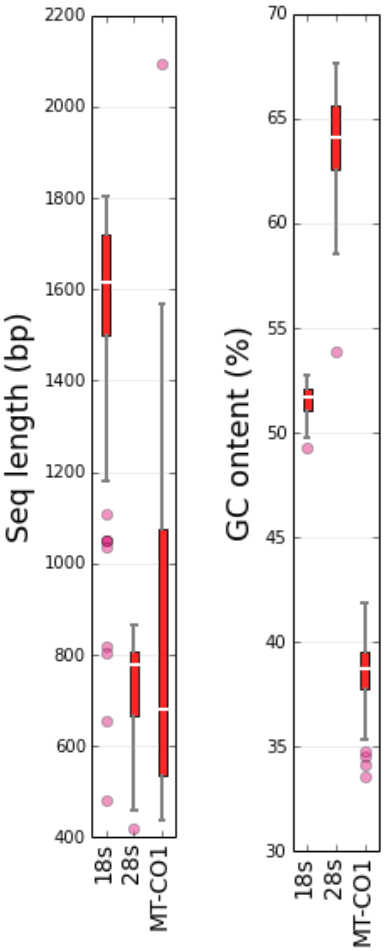
At this point we have record features belonging to the loci in our `Project`. We have to split them by locus:

```
pj.extract_by_locus()
```

With this done, we can display length and %GC distribution for each locus:

```
%matplotlib inline
pj.report_seq_stats()
```

```
Distribution Of Sequence Lengths
Distribution Of Sequence Statistic "Gc_Content"
```



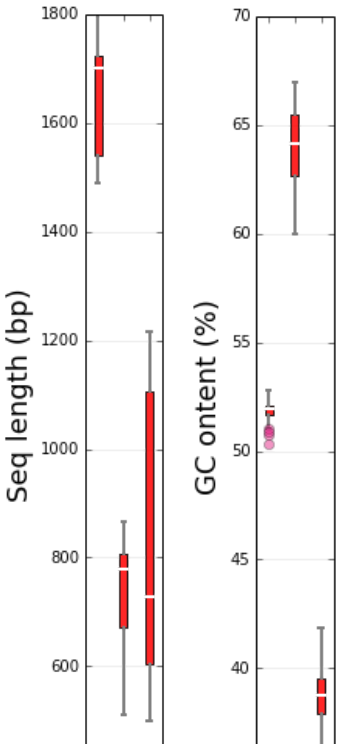
Now we'll exclude all the outliers:

```
# Define minima and maxima
gc_inliers = {
    '18s': [50,54],
    '28s': [57,67],
    'MT-CO1': [35,43]
}
len_inliers = {
    '18s': [1200,1800],
    '28s': [500,900],
    'MT-CO1': [500,1500]
}

# Apply to loci data
for locus in gc_inliers:
    # trim GC outliers
    pj.filter_by_gc_content(locus,
                            min_percent_gc=gc_inliers[locus][0],
                            max_percent_gc=gc_inliers[locus][1])

    # trim length outlier
    pj.filter_by_seq_length(locus,
                            min_length=len_inliers[locus][0],
                            max_length=len_inliers[locus][1])
```

We can now confirm that the filter worked:

```
pj.report_seq_stats()
```

```
Distribution Of Sequence Lengths
Distribution Of Sequence Statistic "Gc_Content"
```



3.5.2 Excluding and including

It is possible to exclude and include sequences by record id.

Excluding

By default, excluding is done by starting with a full bin (all the sequences are included). In this case, since we have already filtered some sequences out, we need to start excluding from the current state and not from a full bin. Starting from a full bin by using the default setting `start_from_max=True` would undo the filtering by GC content and sequence length we have done above. As an example we will exclude JX177918.1 from the MT-CO1 `Locus` bin.

```
exclude = {'MT-CO1': ['JX177918.1']}
```

```
pj.exclude(start_from_max=False, **exclude)
```

The following line confirms that this record id is no longer in the MT-CO1 `Locus` bin.

```
any(['JX177918.1' in feature.id for feature in
pj.records_by_locus['MT-CO1']])
```

```
False
```

Including

By default, including starts from empty bins, however here we want to keep the current state and only add one sequence:

```
include = {'MT-CO1': ['JX177918.1']}
pj.include(start_from_null=False, **include)
```

The following line confirms that this record was added back to the MT-CO1 Locus bin.

```
any(['JX177918.1' in feature.id for feature in
pj.records_by_locus['MT-CO1']])
```

```
True
```

To finish this section:

```
# Update the pickle file
pickle_pj(pj, 'outputs/my_project.pkpj')
```

```
'outputs/my_project.pkpj'
```

3.5.3 Quick reference

```
# Split records to loci bins
pj.extract_by_locus()

# Show length and %GC distributions
%matplotlib inline
pj.report_seq_stats()

# Filter  by GC content
pj.filter_by_locus('LocusName',
                   min_percent_gc = 30,
                   max_percent_gc = 50)
```

```
# Filter  by sequence length
pj.filter_by_seq_length('LocusName',
                        min_length = 200,
                        max_length = 1000)

# Include or exclude records in the loci bins
records = {'LocusName1': ['recordid1','recordid2'],
           'LocusName2': ['recordid3','recordid4']}
pj.exclude(start_from_max=True, **records)

# or
pj.include(start_from_null=True, **records)
```

3.6 Producing and accessing sequence alignment

Section quick reference

```
from reprophylo import *
pj = unpickle_pj('./outputs/my_project.pkpj',
                 git=False)
```

The execution of sequence alignments and accessing them is wrapped with a rich set of functions and methods that make it very convenient to handle many of them. Therefore, it make sense to use ReproPhylo for sequence alignment, even if you do not need a tree as a final output. Although ReproPhylo rejects alignments with less than four sequences because they cannot serve for phylogenetic reconstruction.

3.6.1 Configuring a sequence alignment process

Sequence alignment processes are configured with the `AlnConf` class. An object of this class will generate a command-line and the required input files, but will not execute the process (this is shown below). Once the process has been successfully executed, this `AlnConf` object is stored in `pj.used_methods` and it can be invoked as a report.

The `AlnConf` instance allows control over:

1. The program used (Mafft or Muscle)
2. Whether or not to conduct a codon alignment for CDS loci
3. The genetic code to use for codon alignment
4. The command that invokes the programme (if you want to use a programme that is not in your path)
5. The loci names of the loci to align using this specific approach
6. Custom command line arguments in order to deviate from the programme's default settings

Example 1: codon alignment of CDS loci with the MAFFT L-ins-i algorithm

The next bit of code will construct an `AlnConf` instance that will align only the MT-CO1 CDS

locus, by grabbing the protein sequences from `pj.records`, aligning them using the MAFFT L-ins-i algorithm, and then proceeding with a codon alignment of the CDS sequence with pal2nal, using the protein alignment as reference.

```
mafft_linsi = AlnConf(pj,                                   # The Project

                      method_name='mafftLinsi',             # Any unique method name,
                                                            # 'mafftDefault' by default

                      CDSAlign=True,                        # Use this method to align
                                                            # protein sequences, and then
                                                            # pal2nal to align the CDSs
                                                            # This is the default setting
                                                            # and it is ignored with non-CDS
                                                            # loci.

                      codontable=4,                         # The genetic code that
                                                            # applies to this data,
                                                            # codontable=1 by default

                      program_name='mafft',                 # mafft or muscle.
                                                            # 'mafft' by default

                      cmd='mafft',                          # The command on your machine
                                                            # that invokes the program.
                                                            # 'mafft' by default

                      loci=['MT-CO1'],                      # A list of loci names to align.
                                                            # loci='all' by default, which will
                                                            # align all the loci in the project.
                                                            # If loci=='all', and CDSAlign==True
                                                            # CDS loci will be aligned as proteins
                                                        # (and then at the DNA level with pal2nal)
                                                        # but other DNA loci (e.g. rRNA) will be
                                                        # aligned directly at the DNA level.

                      cline_args={'localpair': True,# Program specific keywords and arguments.
                                  'maxiterate': 1000}# cine_args=={} by default, which will
                     )                                   # execute the program with default settings
```

```
mafft --localpair --maxiterate 1000 933261440758989.85_CDS_proteins_MT-CO1.fasta
```

Example 2: Alignment of rRNA loci with Muscle default algorithm

This is a simpler example where DNA loci will be directly aligned using Muscle with default settings. I am not specifying `CDSAlign=False` because this is not a CDS locus so there will be no attempt to do a codon alignment. the codontable argument is also ignored. I am also not specifying `cmd='muscle'`, because when we set `program='muscle'`, then the default value of `cmd` becomes `'muscle'`.
(hint: by the way, if you have a reference alignment which accounts for the secondary structure on your RNA locus, it can be utilized with the `seed` argument in Mafft).

```
muscle_defaults = AlnConf(pj,
                          method_name="muscleDefault",
                          program_name="muscle",
                          loci=['18s','28s'])
```

```
muscle -in 375991440758992.65_18s.fasta
muscle -in 375991440758992.65_28s.fasta
```

3.6.2 Executing sequence alignment processes

Once we have one or more `AlnConf` objects, we can execute them in one go using the `Project` method `align`. This method accepts a list of `AlnConf` objects and does whatever it is each of them tells it:

```
pj.align([mafft_linsi, muscle_defaults])
```

When the process is done, the `AlnConf` objects will be stored in `pj.used_methods`, which is a dictionary using the method names as keys:

```
pj.used_methods
```

```
{'mafftLinsi': <reprophylo.AlnConf instance at 0x7f103c1f7128>,
 'muscleDefault': <reprophylo.AlnConf instance at
0x7f103ba20050>}
```

if we print one of these `AlnConf` objects as a string, we will get complete details about the process, including programme versions and references:

```
print pj.used_methods['mafftLinsi']
```

```
AlnConf named mafftLinsi with ID 933261440758989.85
Loci: MT-CO1
Created on: Fri Aug 28 11:49:49 2015
Commands:
MT-CO1: mafft --localpair --maxiterate 1000
933261440758989.85_CDS_proteins_MT-CO1.fasta

Environment:
Platform:
```

```
Linux-3.13.0-40-generic-x86_64-with-Ubuntu-14.04-trusty
 Processor: x86_64
 Python build: defaultJun 22 2015 17:58:13
 Python compiler: GCC 4.8.2
 Python implementation: CPython
 Python version: 2.7.6
 ete2 version: 2.2rev1056
 biopython version: 1.64
 dendropy version: 3.12.0
 cloud version: 2.8.5
 reprophylo version 1.0
 User: amir-TECRA-W50-A
 Program and version: MAFFT v7.123b\nPal2Nal v14
 Program reference:Katoh
 Standley 2013 (Molecular Biology and Evolution 30:772-780)
MAFFT multiple sequence alignment software version 7:
improvements in performance and usability.\nMikita Suyama
 David Torrents
 and Peer Bork (2006) PAL2NAL: robust conversion of protein
sequence alignments into the corresponding codon
alignments.Nucleic Acids Res. 34
 W609-W612.
execution time:
2.26954507828


==============================
Core Methods section sentence:
==============================
The dataset(s) MT-CO1 were first aligned at the protein level
using the program MAFFT v7.123b [1].
The resulting alignments served as guides to codon-align the
DNA sequences using Pal2Nal v14 [2].

Reference:
[1]Katoh, Standley 2013 (Molecular Biology and Evolution
30:772-780) MAFFT multiple sequence alignment software version
7: improvements in performance and usability.
[2]Mikita Suyama, David Torrents, and Peer Bork (2006) PAL2NAL:
robust conversion of protein sequence alignments into the
corresponding codon alignments.Nucleic Acids Res. 34,
W609-W612.
```

3.6.3 Accessing sequence alignments

The `pj.alignments` dictionary

The alignments themselves are stored in the `pj.alignments` dictionary, using keys that

follow this pattern: `locus_name@method_name` where `method_name` is the name you have provided to your `AlnConf` object.

```
pj.alignments
```

```
{'18s@muscleDefault': <<class 'Bio.Align.MultipleSeqAlignment'>
instance (52 records of length 1824, IUPACProtein()) at
7f103bba6790>,
 '28s@muscleDefault': <<class 'Bio.Align.MultipleSeqAlignment'>
instance (48 records of length 909, IUPACProtein()) at
7f103be32310>,
 'MT-CO1@mafftLinsi': <<class 'Bio.Align.MultipleSeqAlignment'>
instance (73 records of length 1227, IUPACAmbiguousDNA()) at
7f103bbc1350>}
```

Accessing a MultipleSeqAlignment object

An alignment can be easily accessed and manipulated with any of Biopython's AlignIO tricks using the `Project` method `fa`:

```
print pj.fa('18s@muscleDefault')[:4,410:420].format('phylip-relaxed')
```

```
returning alignment object 18s@muscleDefault
 4 10
KC762720.1_f0  GAGAAACGGC
KC774024.1_f0  GAGAAACGGC
KC762713.1_f0  GAGAAACGGC
KC762708.1_f0  GAGAAACGGC
```

Writing sequence alignment files

Alignment text files can be dumped in any AlignIO format for usage in an external command line or GUI program. When writing to files, you can control the header of the sequence by, for example, adding the organism name of the gene name, or by replacing the feature ID with the record ID:

```
# record_id and source_organism are feature qualifiers in the
SeqRecord object
# See section 3.4
files = pj.write_alns(id=['record_id','source_organism'],
                      format='fasta')
files
```

```
['28s@muscleDefault_aln.fasta',
 '18s@muscleDefault_aln.fasta',
 'MT-CO1@mafftLinsi_aln.fasta']
```

The files will always be written to the current working directory (where the Jupyter notebook

file is), and can immediately be moved programmatically to avoid clutter:

```
# make a new directory for your alignment files:
if not os.path.exists('alignment_files'):
    os.mkdir('alignment_files')

# move the files there
for f in files:
    os.rename(f, "./alignment_files/%s"%f)
```

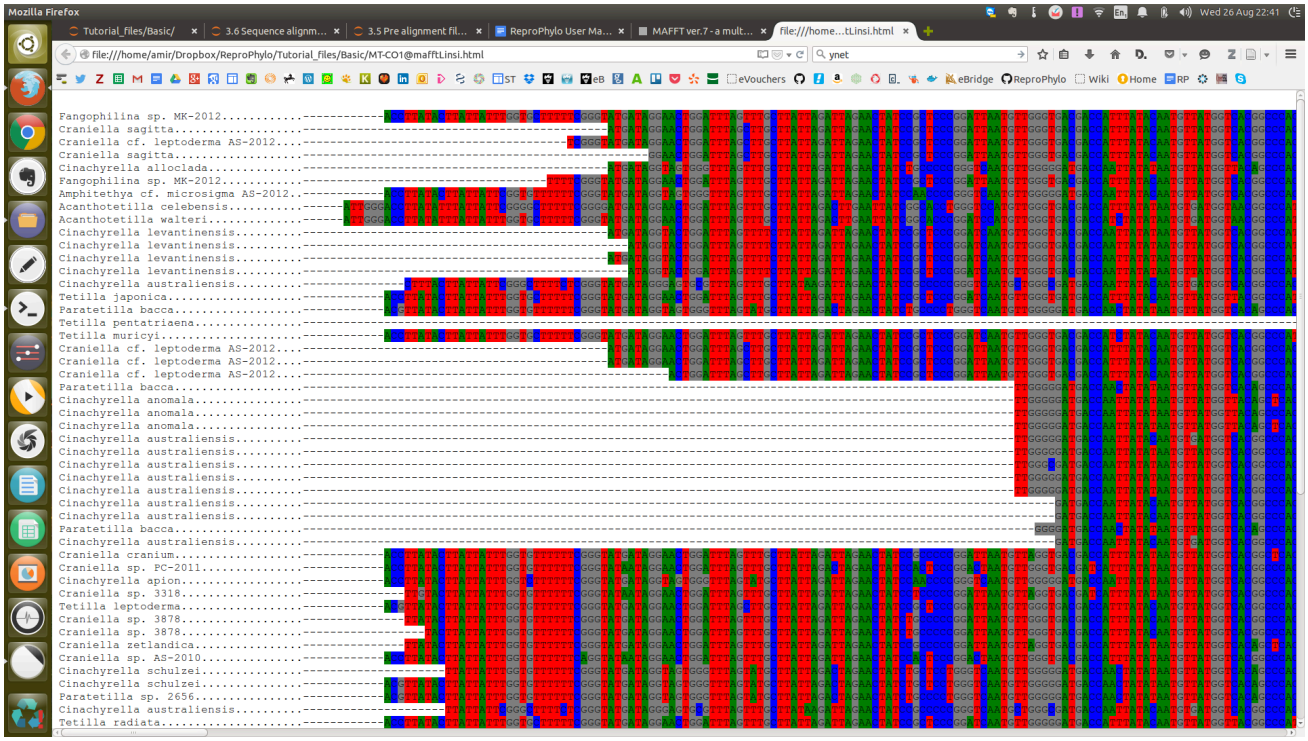
Viewing alignments

ReproPhylo has a `Project` method to view the alignments in the browser. This one also allows you to control the content of sequence headers.

```
pj.show_aln('MT-CO1@mafftLinsi',id=['source_organism'])
# source_organism is a feature qualifier in the SeqRecord object
# See section 3.4
```

As a result of this command, a new browser tab will open, showing the alignment.

Note1: In some cases, the tab will not open automatically, you will need to look for the html file in your working directory and open it manually.
Note2: This is slow with large alignment. A better approach with large files would be to write a text file, as above, and look at the alignment in AliView or any of your preferred alignment viewers.



Pickle the Project to end this section:

```
pickle_pj(pj, 'outputs/my_project.pkpj')
```

```
'outputs/my_project.pkpj'
```

3.6.4 Quick reference

```
# Make a AlnConf object
alnconf = AlnConf(pj, **kwargs)

# Execute alignment process
pj.align([alnconf])

# Show AlnConf description
print pj.used_methods['method_name']

# Fetch a MultipleSeqAlignment object
aln_obj = pj.fa('locus_name@method_name')

# Write alignment text files
pj.write_alns(id=['some_feature_qualifier'], format='fasta')
# the default feature qualifier is 'feature_id'
# 'fasta' is the default format

# View alignment in browser
pj.show_aln('locus_name@method_name',id=['some_feature_qualifier'])
```

3.7 Alignment trimming

This section also starts with a `Project` that already contains alignments:

```
from reprophylo import *
pj = unpickle_pj('./outputs/my_project.pkpj',
                 git=False)
```

If we call the keys of the pj.alignments dictionary, we can see the names of the alignments it contains:

```
pj.alignments.keys()
```

```
['28s@muscleDefault', 'MT-CO1@mafftLinsi',
'18s@muscleDefault']
```

3.7.1 Configuring an alignment trimming process

Like the sequence alignment phase, alignment trimming has its own configuration class, the

`TrimalConf` class. An object of this class will generate a command-line and the required input files for the program TrimAl, but will not execute the process (this is shown below). Once the process has been successfully executed, this `TrimalConf` object is also stored in `pj.used_methods` and it can be invoked as a report.

Example1, the default gappyput algorithm

With `TrimalConf`, instead of specifying loci names, we provide alignment names, as they appear in the keys of `pj.alignments`

```
gappyout = TrimalConf(pj,                       # The Project

                      method_name='gappyout',  # Any unique string
                                               # ('gappyout' is default)

                      program_name='trimal', # No alternatives in
                                               # this ReproPhylo version

                      cmd='default', # the default is trimal.
                                # Change it here
                                # or in pj.defaults['trimal']

                      alns=['MT-CO1@mafftLinsi'],# 'all' by default

                      trimal_commands={'gappyout': True} # By
                         # default, the gappyout algorithm is used.
                      )
```

```
trimal -in 587711440759152.37_MT-CO1@mafftLinsi.fasta
-gappyout
```

List comprehension to subset alignments

In this example, it is easy enough to copy and paste alignment names into a list and pass it to `TrimalConf`. But this is more difficult if we want to fish out a subset of alignments from a very large list of alignments. In such cases, Python's list comprehension is very useful. Below I show two uses of list comprehension, but the more you feel comfortable with this approach, the better.

Getting locus names of rRNA loci

If you read the code line that follows very carefully, you will see it quite literally says "take the name of each Locus found in `pj.loci` if its feature type is rRNA, and put it in a list":

```
rRNA_locus_names = [locus.name for locus in pj.loci if
                    locus.feature_type == 'rRNA']
print rRNA_locus_names
```

```
['18s', '28s']
```

what we get is a list of names of our rRNA loci.

Getting alignment names that have locus names of rRNA loci

The following line says: "take the key of each alignment from the `pj.alignments` dictionary if the first word before the '@' symbol is in the list of rRNA locus names, and put this key in a list":

```
rRNA_alignment_names = [key for key in pj.alignments.keys() if
                        key.split('@')[0] in rRNA_locus_names]
print rRNA_alignment_names
```

```
['28s@muscleDefault', '18s@muscleDefault']
```

We get a list of keys, of the rRNA loci alignments we produced on the previous section, and which are stored in the `pj.alignments` dictionary. We can now pass this list to a new `TrimalConf` instance that will only process rRNA locus alignments:

```
gt50 = TrimalConf(pj,
                  method_name='gt50',
                  alns = rRNA_alignment_names,
                  trimal_commands={'gt': 0.5}  # This will keep
                                               # positions with up to
                                               # 50% gaps.
                  )
```

```
trimal -in 915841440759159.29_28s@muscleDefault.fasta -gt 0.5
trimal -in 915841440759159.29_18s@muscleDefault.fasta -gt 0.5
```

3.7.2 Executing the alignment trimming process

As for the alignment phase, this is done with a `Project` method, which accepts a list of `TrimalConf` objects.

```
pj.trim([gappyout, gt50])
```

Once used, these objects are also placed in the `pj.used_methods` dictionary, and they can be printed out for observation:

```
print pj.used_methods['gappyout']
```

```
TrimalConf named gappyout with ID 587711440759152.37
Alignments: MT-CO1@mafftLinsi
Created on: Fri Aug 28 11:52:32 2015
Commands:
MT-CO1@mafftLinsi@gappyout: trimal -in
```

```
587711440759152.37_MT-CO1@mafftLinsi.fasta -gappyout

Environment:Platform:
Linux-3.13.0-40-generic-x86_64-with-Ubuntu-14.04-trusty
 Processor: x86_64
 Python build: defaultJun 22 2015 17:58:13
 Python compiler: GCC 4.8.2
 Python implementation: CPython
 Python version: 2.7.6
 ete2 version: 2.2rev1056
 biopython version: 1.64
 dendropy version: 3.12.0
 cloud version: 2.8.5
 reprophylo version 1.0
 User: amir-TECRA-W50-A
 Program and version: trimAl 1.2rev59
 Program reference: Salvador Capella-Gutierrez; Jose M.
Silla-Martinez; Toni Gabaldon. trimAl: a tool for automated
alignment trimming in large-scale phylogenetic analyses.
Bioinformatics 2009 25: 1972-1973.
execution time:
0.478782892227

==============================
Core Methods section sentence:
==============================
The alignment(s) MT-CO1@mafftLinsi were trimmed using the
program trimAl 1.2rev59 [1].

Reference:
Salvador Capella-Gutierrez; Jose M. Silla-Martinez; Toni
Gabaldon. trimAl: a tool for automated alignment trimming in
large-scale phylogenetic analyses. Bioinformatics 2009 25:
1972-1973.
```

3.7.3 Accessing trimmed sequence alignments

The `pj.trimmed_alignments` dictionary

The trimmed alignments themselves are stored in the `pj.trimmed_alignments` dictionary, using keys that follow this pattern: `locus_name@alignment_method_name@trimming_method_name` where `alignment_method_name` is the name you have provided to your `AlnConf` object and `trimming_method_name` is the one you provided to your `TrimalConf` object.

```
pj.trimmed_alignments
```

```
{'18s@muscleDefault@gt50': <<class
```

```
'Bio.Align.MultipleSeqAlignment'> instance (52 records of
length 1685, IUPACAmbiguousDNA()) at 7fe542480510>,
 '28s@muscleDefault@gt50': <<class
'Bio.Align.MultipleSeqAlignment'> instance (48 records of
length 798, IUPACAmbiguousDNA()) at 7fe542480550>,
 'MT-CO1@mafftLinsi@gappyout': <<class
'Bio.Align.MultipleSeqAlignment'> instance (73 records of
length 1107, IUPACAmbiguousDNA()) at 7fe5424d6dd0>}
```

Accessing a MultipleSeqAlignment object

A trimmed alignment can be easily accessed and manipulated with any of Biopython's AlignIO tricks using the `fta Project` method:

```
print
pj.fta('18s@muscleDefault@gt50')[:4,410:420].format('phylip-relaxed
')
```

```
returning trimmed alignment object 18s@muscleDefault@gt50
 4 10
KC762720.1_f0  CCAATCCCGA
KC774024.1_f0  CCAATCCCGA
KC762713.1_f0  CCAATCGGGA
KC762708.1_f0  CCAATCCCGA
```

Writing trimmed sequence alignment files

Trimmed alignment text files can be dumped in any AlignIO format for usage in an external command line or GUI program. When writing to files, you can control the header of the sequence by, for example, adding the organism name of the gene name, or by replacing the feature ID with the record ID:

```
# record_id and source_organism are feature qualifiers in the
SeqRecord object
# See section 3.4
files = pj.write_trimmed_alns(id=['record_id','source_organism'],
                                  format='fasta')
files
```

```
['28s@muscleDefault@gt50_trimmed_aln.fasta',
 '18s@muscleDefault@gt50_trimmed_aln.fasta',
 'MT-CO1@mafftLinsi@gappyout_trimmed_aln.fasta']
```

The files will always be written to the current working directory (where this Jupyter notebook file is), and can immediately be moved programmatically to avoid clutter:

```
# make a new directory for your trimmed alignment files:
```

```
if not os.path.exists('trimmed_alignment_files'):
    os.mkdir('trimmed_alignment_files')

# move the files there
for f in files:
    os.rename(f, "./trimmed_alignment_files/%s"%f)
```

Viewing trimmed alignments

Trimmed alignments can be viewed in the same way as alignments, but using this command:

```
pj.show_aln('MT-CO1@mafftLinsi@gappyout',id=['source_organism'])
```

```
# Pickle the Project

pickle_pj(pj, 'outputs/my_project.pkpj')
```

```
'outputs/my_project.pkpj'
```

3.7.4 Quick reference

```
# Make a TrimalConf object
trimconf = TrimalConf(pj, **kwargs)

# Execute alignment process
pj.trim([trimconf])

# Show AlnConf description
print pj.used_methods['method_name']

# Fetch a MultipleSeqAlignment object
trim_aln_obj =
pj.fta('locus_name@aln_method_name@trim_method_name')

# Write alignment text files
pj.write_trimmed_alns(id=['some_feature_qualifier'],
                      format='fasta')
# the default feature qualifier is 'feature_id'
# 'fasta' is the default format

# View alignment in browser
pj.show_aln('locus_name@aln_method_name@trim_method_name',id=['some
_feature_qualifier'])
```

3.8 Building a supermatrix

Section quick reference

This section shows how to build a supermatrix by providing minimal requirements for gene content per taxon (OTU). This approach is more suited for *small scale* analysis, because it relies on manual decisions, whereas *large scale* suprematrices are better constructed with the parameter space and data explorations tools of ReproPhylo. However, these are not addressed in this section. First, lets load our `Project` with the trimmed alignments:

```
from reprophylo import *
pj = unpickle_pj('outputs/my_project.pkpj', git=False)
```

3.8.1 Sorting out the metadata

The main decision to make when building a supermatrix is what metadata will be used to indicate that sequences of several genes belong to the same OTU in the tree. Obvious candidates would be the species name (stored as '`source_organism`' if we read a GenBank file), or sample ID, voucher specimen and so on. Often, we would be required to modify the metadata in our `Project`, in a way that will correctly reflect the relationship between sequences that emerged from the same sample.

In the case of the `Tetillidae.gb` example file, sample IDs are stored either under `'source_specimen_voucher'` or `'source_isolate'`. In addition, identical voucher numbers are sometimes formatted differently for different genes.

In the file '`data/Tetillida_otus_corrected.csv`', I have unified the columns `'source_specimen_voucher'` and `'source_isolate'` in a single column called `'source_otu'` and also made sure to uniformly format all the voucher specimens:

H	I
source:_specimen_voucher	source:_otu
QMG_320216	QMG_320216
QMG_320216	QMG_320216
QMG_320270	QMG_320270
QMG_320270	QMG_320270
QMG_320636	QMG_320636
QMG_320656	QMG_320656
QMG_320656	QMG_320656
QMG_321405	QMG_321405
QMG314224	QMG_314224
QMG315031	QMG_315031
QMG316342	QMG_316342
QMG316372	QMG_316372
QMG316372	QMG_316372
QMG318785	QMG_318785
QMG320143	QMG_320143
QMG320270	QMG_320270

Our Project has to be updated with the recent changes to the spreadsheet:

```
pj.correct_metadata_from_file('data/Tetillida_otus_corrected.csv')
```

Such fixes can also be done programmatically (see section 3.4)

3.8.2 Designing the supermatrix

Supermatrices are configured with objects of the class `Concatenation`. In a `Concatenation` object we can indicate the following:

1. The name of the concatenation
2. The loci it includes (here we pass `Locus` objects rather than just `Locus` names)
3. The qualifier or metadata that stores the relationships among the records
4. What loci all the OTUs must have
5. Groups of loci from which each OTU must have at least one
6. Which trimmed alignment to use, if we have more than one for each locus in our Project

Here is an example:

```
concat = Concatenation('large_concat', # Any unique string

                       pj.loci, # This is a list of Locus objects

                       'source_otu', # The values of this qualifier
                        # flag sequences the belong to the same
                        # sample

                       otu_must_have_all_of=['MT-CO1'], # All the
                       # OTUS must have a cox1 sequence

                       otu_must_have_one_of=[['18s','28s']], # All
                  # the OTUs must have either 18s or 28s or both

                       define_trimmed_alns=[] # We only have one
                                              # alignment per gene
                           # so the list is empty (default value)
                      )
```

If we print this `Concatenation` object we get this message:

```
print concat
```

```
Concatenation named large_concat, with loci 18s,28s,MT-CO1,
of which MT-CO1 must exist for all species
and at least one of each group of [ 18s 28s ] is represented.
Alignments with the following names:  are prefered
```

3.8.3 Building the supermatrix

Building the suprematrix has two steps. First we need to mount the `Concatenation` object

onto the `Project` where it will be stored in the list `pj.concatenations`. Second, we need to construct the `MultipleSeqAlignment` object, which will be stored in the `pj.trimmed_alignments` dictionary, under the key `'large_concat'` in this case:

```
pj.add_concatenation(concat)
pj.make_concatenation_alignments()
```

```
Concatenation large_concat will have the following data
OTU                          18s                 28s                 MT-CO1
NIWA_28507                   JX177975.1_f0  JX177943.1_f0  JX177896.1_f0
ZMBN_85230                                  HM592765.1_f0  HM592717.1_f0
NIWA_28910                   JX177982.1_f0                 JX177865.1_f0
VM_14754                     JX177986.1_f0  JX177960.1_f0  HM032751.1_f0
ZMBN_85239                   JX177987.1_f0  JX177959.1_f0  HM592669.1_f0
ZMBN_81789                                  HM592753.1_f0  HM592667.1_f0
QMG_315031                   JX177974.1_f0  JX177942.1_f0  HM032749.2_f0
NIWA_28617                   JX177980.1_f0                 JX177912.1_f0
RMNH_POR_3206                               JX177925.1_f0  JX177892.1_f0
UFBA_2021_POR                               JX177921.1_f0  JX177907.1_f0
NIWA_28586                   JX177978.1_f0  JX177953.1_f0  JX177918.1_f0
QMG_320270                   JX177963.1_f0  JX177931.1_f0  HM032741.1_f0
QMG_318785                   JX177985.1_f0                 HM032752.3_f0
NIWA_25206                   JX177981.1_f0                 JX177917.1_f0
QMG_320216                   JX177966.1_f0                 JX177902.1_f0
MHNM_16194                   HM629803.1_f0  JX177941.1_f0  JX177905.1_f0
ZMA_POR_16637                               HM592820.1_f0  HM592745.1_f0
SAM_S1189                                   JX177929.1_f0  JX177910.1_f0
TAU_25529                    JX177970.1_f0  JX177939.1_f0  JX177906.1_f0
LB_1756                                     JX177933.1_f0  JX177886.1_f0
MNRJ_576                                    JX177957.1_f0  HM032742.1_f0
NIWA_28877                   JX177977.1_f0  JX177950.1_f0  JX177864.2_f0
NIWA_28524                   JX177976.1_f0  JX177945.1_f0  JX177895.1_f0
QMG_316342                   JX177983.1_f0  JX177955.1_f0  HM032747.2_f0
TAU_25568                    JX177969.1_f0  JX177940.1_f0  JX177904.1_f0
NIWA_28929                                  JX177951.1_f0  JX177863.1_f0
DH_S271                      JX177965.1_f0  JX177935.1_f0  JX177913.1_f0
ZMBN_85240                                  HM592754.1_f0  HM592668.1_f0
QMG_321405                                  JX177930.1_f0  HM032743.1_f0
NIWA_36097                                  JX177944.1_f0  JX177866.1_f0
UFBA_2586_POR                               JX177958.1_f0  JX177898.1_f0
NIWA_52077                                  JX177948.1_f0  JX177916.1_f0
QMG_316372                   HE591469.1_f0                 HM032748.2_f0
QMG_320636                   JX177971.1_f0                 HM032745.1_f0
NIWA_28496                                  JX177946.1_f0  JX177897.1_f0
QMG_314224                                  JX177924.1_f0  HM032744.1_f0
QMG_320143                   JX177973.1_f0  JX177922.1_f0  HM032746.1_f0
DH_S124                                     JX177938.1_f0  JX177903.1_f0
SP_DH_S192                   JX177961.1_f0  JX177956.1_f0  JX177901.1_f0
SP_DH_S193                                  JX177926.1_f0  JX177900.1_f0
NIWA_28957                                  JX177949.1_f0  JX177867.2_f0
RMNH_POR_2877                               JX177920.1_f0  JX177909.1_f0
```

```
# Save the results of this section in the pickle
```

```
pickle_pj(pj, 'outputs/my_project.pkpj')
```

```
'outputs/my_project.pkpj'
```

Now that this supermatrix is stored as a trimmed alignment in the `pj.trimmed_alignments` dictionary, we can write it to a file or fetch the `MultipleSeqAlignment` object, as shown in section 3.7.

3.8.4 Quick reference

```
# Design a supermatrix
concat = Concatenation('concat_name', loci_list, 'otu_qualifier',
                       **kwargs)

# Add it to a project
pj.add_concatenation(concat)

# Build supermatrices based on the Concatenation
# objects in pj.concatenations
pj.make_concatenation_alignments()
```

Tree reconstruction can be done with RAxML or Phylobayes. This section will cover one example in which we will build a supermatrix tree using RAxML, and a single gene tree using Phylobayes.

3.9 Reconstructing trees

Quick reference

```
from reprophylo import *
pj = unpickle_pj('outputs/my_project.pkpj', git=False)
```

3.9.1 Using RAxML

RAxML is configured with the `RaxmlConf` object. This object provides control over the following settings:

1. **`method_name`**: The method name.
2. **`program_name` & `cmd`**:. RAxML binaries exist in several versions. **If you are using the Docker container you can leave this as is.** The versions vary in the number of threads they utilized (PTHREADS or not), and the architecture they are optimized for (AVX or SSE3). raxmlHPC-PTHREADS-SSE3 is the default here, both as the program name and as the cmd. If you do not want to use multiple threads, you have to specify the name and command of the non PTHREADS binary, ie, raxmlHPC.
3. **`keepfiles`**: Whether or not to **keep the output files** in the working directory (the tree is stored in the `Project`)
4. **`preset`**: The RAxML algorithm. `RaxmlConf` has several preset algorithms:
   - `'fa'` - will run a single ML search with rapid bootstrap
   - `'fD_fb'` - will run a single ML tree with relBootstrap (quick and least accurate supports calculation)

- `'fd_b_fb'` - will run one or more ML trees with thorough bootstrap (slow and accurate)
- `'fF_fJ'` - will run a fast ML tree with sh-like supports (quick and dirty)
- `'fd_fJ'` - will run one or more (proper) ML tree(s) with sh-like supports (quick supports calculation).

5. **`alns`**: Alignments to analyze. 'all' by default. It can be modified by passing a list of trimmed alignment names and/or concatenation names.
6. **`model`**: The model of rate heterogeneity. For example, GAMMA (parametric) or CAT (nonparametric). The CAT model is a nonparametric approach to categories the rate variation without calculating the GAMMA distribution, as a fast approximation. It is different than the CAT model in Phylobayes, where the number of parameters increases by categorizing the data to subsets, which differ in their substitution matrices and rate variation categories. The CAT in RAxML is "quick and dirty". The CAT in Phylobayes is "slow and accurate."
7. **`matrix`**: The protein substitution matrix. This parameter is only relevant to protein datasets, and it is ignored for DNA only datasets. By default it is set to 'JTT'. If it is a concatenated analysis, the string specified here will be set as the substitution matrix of each of the protein partitions. **However**, it is possible to pass a dictionary, instead of a string, containing the locus names as keys, and the name of substitution matrix assigned to each of them as values. **Also important**, partition information is taken into account automatically. No need to make a partition file.
8. **`threads`**: The number of threads to use. Using PTHREADS, threads=1 is automatically changed to 2. Using the non PTHREADS version, the threads number is set to one, regardless of the value the user passes.
9. **cline_args**: Other command line arguments, most importantly, the argument `'-N'` should be used to determine the number of ML searches (it is 1 by default and it doesn't work with `fa` or `fF_fJ`), and `'-#'` should be used to set the number of bootstrap replicates (it is 100 by default and it **only** works with `fD_fb` and `fd_b_fb`). `-N` and `-#` are not synonyms. This is different from the RAxML command line.

The RAxML manual is an important read, in order to understand all the analysis modifiers that can be passed, and to become familiar with the full range of models and substitution matrices available.
In this example, comments which specify item numbers, refer to the list just above. It will configure a concatenated analysis of the supermatrix `'large_concat'`, with the GTR GAMMA model for all the partitions, utilizing two threads and with two ML searches. Branch supports will be derived from a relBootstrap analysis.

```
raxml = RaxmlConf(pj,  # The Project
                  method_name='supermatrix', # Any string
                  program_name='raxmlHPC-PTHREADS-SSE3', # item 2
                  keepfiles=False,  # False is default
                  cmd='default',                          # item 2
                  preset='fD_fb',                         # item 4
                  alns=['large_concat'],                  # item 5
                  model='GAMMA',                          # item 6
                  matrix='JTT',                           # item 7
                  threads=4,                              # item 8
                  cline_args={'-N': 2}                    # item 9
```

```
                    )
raxmlHPC-PTHREADS-SSE3 -f D -m PROTGAMMAJTT -n 221101440759352.8_large_concat0 -q
221101440759352.8_large_concat_partfile -p 603 -s 221101440759352.8_large_concat.fasta -T 4 -N 2


raxmlHPC-PTHREADS-SSE3 -f b -m PROTGAMMAJTT -n 221101440759352.8_large_concat1 -q
221101440759352.8_large_concat_partfile -p 369 -s 221101440759352.8_large_concat.fasta -t
RAxML_bestTree.221101440759352.8_large_concat0 -T 4 -z
RAxML_rellBootstrap.221101440759352.8_large_concat0
```

3.9.2 Using Phylobayes

In this example, a `PbConf` object is set to analyse a single trimmed alignment. The `cline_args` here are horrible and set this way for speed. The default settings, however, are sensible. Still, read the manual, at least the bits about `nchain`, burn-in and the proper usage of the GTR and/or CAT models (and others).

```
phylo = PbConf(pj,                                  # Default setting:
               method_name='single_gene',           # 'dna_cat_gtr'
               program_name='phylobayes',
               keepfiles=False,                     # True
               cmd='default',
               alns=['28s@muscleDefault@gt50'],     # 'all'
               cline_args={'gtr': True,
                           'cat': True,
                           'nchain': '2 50 0.9 5',# '2 100 0.1 100'
                           'b': '1'                # '5'
                           }
               )
```

```
pb -d 262531440759355.16_28s@muscleDefault@gt50.phylip -gtr
-nchain 2 50 0.9 5 -b 1 -cat
262531440759355.16_28s@muscleDefault@gt50
```

3.9.3 Executing the tree reconstructions and accessing trees

This is done using the tree Project method:

```
pj.tree([raxml, phylo])
```

The resulting trees are placed in the `pj.trees` dictionary, with keys of the form `'locus_name@aln_method@trim_method@tree_method'`. For trees from supermatrices the key is `'concat_name@mixed@mixed@tree_method'`. The values in this dictionary are lists, each with two values. The first in an ETE `Tree` object, and the second is an NHX string representation of the tree.

```
pj.trees.keys()
```

```
['large_concat@mixed@mixed@supermatrix',
'28s@muscleDefault@gt50@single_gene']
```

And as for alignment and trimming, we can review the approaches that we used:

```
print pj.used_methods['single_gene']
```

```
PbConf named single_gene with ID 262531440759355.16
Alignments: 28s@muscleDefault@gt50
Created on: Fri Aug 28 11:55:55 2015
Commands:
28s@muscleDefault@gt50: ['pb -d
262531440759355.16_28s@muscleDefault@gt50.phylip -gtr -nchain 2
50 0.9 5 -b 1 -cat 262531440759355.16_28s@muscleDefault@gt50']

Environment:
Platform: Linux-3.13.0-40-generic-x86_64-with-Ubuntu-14.04-trusty
 Processor: x86_64
 Python build: defaultJun 22 2015 17:58:13
 Python compiler: GCC 4.8.2
 Python implementation: CPython
 Python version: 2.7.6
 ete2 version: 2.2rev1056
 biopython version: 1.64
 dendropy version: 3.12.0
 cloud version: 2.8.5
 reprophylo version 1.0
 User: amir-TECRA-W50-A
 Program and version: phylobayes version 3.3f
 Program reference: N. Lartillot
 T. Lepage and S. Blanquart
 2009: PhyloBayes 3: a Bayesian software package for phylogenetic
reconstruction and molecular dating. Bioinformatics Vol. 25 no.
17.
execution time:
180.445002794

==============================
Core Methods section sentence:
==============================
Phylogenetic trees were reconstructed from the dataset(s)
28s@muscleDefault@gt50 using the program phylobayes version 3.3f
[1].

Reference:
N. Lartillot, T. Lepage and S. Blanquart, 2009: PhyloBayes 3: a
Bayesian software package for phylogenetic reconstruction and
molecular dating. Bioinformatics Vol. 25 no. 17.
```

`Tree` objects can be fetched easily and manipulated with ETE tricks, using the ft Project

method.

```
t =
pj.ft('28s@muscleDefault@gt50@single_gene').convert_to_ultrametric(10)
```

```
returning tree object 28s@muscleDefault@gt50@single_gene
```

or written to a file in a suitable format

```
pj.ft('28s@muscleDefault@gt50@single_gene').write(features=['source_organism'],
                                                  format=5, outfile="new_tree.nw")
```

```
returning tree object 28s@muscleDefault@gt50@single_gene
```

```
# And now update the Project's pickle:
pickle_pj(pj, 'outputs/my_project.pkpj')
```

```
'outputs/my_project.pkpj'
```

3.9.4 Quick reference

```
# Configure a raxml analysis
raxml = RaxmlConf(pj, **kwargs)

# Configure a phylobayes analysis
phylo = PbConf(pj, **kwargs)

# Execute tree reconstruction
pj.tree([list_of_RaxmlConf_and_or_PbConf_objects])

# Fetch an ETE Tree object
t = pj.ft('locus_name@aln_name@trim_name@tree_name')

# Write newick file
t.write(format=5, outfile="filename.nw")

# Write NHX format with all the qualifiers
t.write(features=[], format=5, outfile="filename.nw")
```

3.10 Tree annotation and report

The last section of this tutorial is about producing annotated tree figures and a human readable report. First we have to load our `Project` again:

```
from reprophylo import *
pj = unpickle_pj('outputs/my_project.pkpj', git=False)
```

3.10.1 Updating the metadata after the tree has been built

Often, we want to display information that did not exist in the `Project` when we first built our trees. This is not an issue. We can add metadata now and propagate it to all the parts of the `Project`, including to our preexisting trees. For example, I add here some morphological information. Some of the species in our data have a morphological structure called porocalyx,

```
genera_with_porocalices = ['Cinachyrella',
                           'Cinachyra',
                           'Amphitethya',
                           'Fangophilina',
                           'Acanthotetilla',
                           'Paratetilla']
```

while others do not:

```
genera_without_porocalices = ['Craniella',
                              'Tetilla',
                              'Astrophorida']
```

The following command will add the value `'present'` to a new qualifier called `'porocalices'` in sequence features of species that belong to `genera_with_porocalices`:

```
for genus in genera_with_porocalices:
    pj.if_this_then_that(genus, 'genus', 'present', 'porocalices')
```

and the following command will add the value `'absent'` to a new qualifier called `'porocalices'` to sequence features of species that belong to `genera_without_porocalices`:

```
for genus in genera_without_porocalices:
    pj.if_this_then_that(genus, 'genus', 'absent', 'porocalices')
```

The new qualifier `porocalices` in now updated in the `SeqRecord` objects within the `pj.records` list (more on this in section 3.4). But in order for it to exist in the `Tree` objects, stored in the `pj.trees` dictionary, we have to run this command:

```
pj.propagate_metadata()
```

Only now the new qualifier is available for tree annotation. Note that qualifiers that existed in the `Project` when we built the trees, will be included in the `Tree` object by default.

3.10.2 Configuring and writing a tree figure

The `annotate Project` method will produce one figure for each tree in the `Project`

according to the settings. Colors can be indicated with X11 color names. The following settings can be controlled:

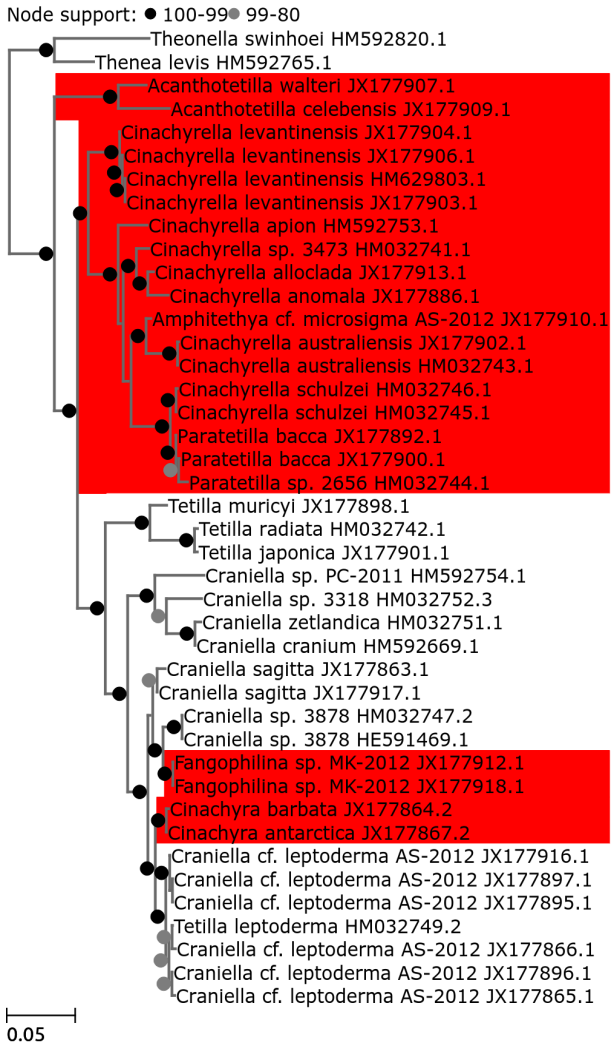
1. **`fig_folder`**: The path for the output figure file
2. **`root_meta`** and **`root_value`**: The qualifier and its value that will indicate the outgroup. It can be `'mid'` and `'mid'` for a midpoint root, or for example, `'source_organism'` and `'Some species binomial'` to set a group of leaves with a shared value as an outgroup (required).
3. **`leaf_labels_txt_meta`**: A list of qualifiers which values will be used as leaf labels, required.
4. **`leaf_node_color_meta`** and **`leaf_label_colors`**: The qualifier that determines clade background colors and a dictionary assigning colors to the qualifier's values (defaults to `None` and `None`).
5. **`ftype`** and **`fsize`**: Leaf label font and font size (default `'verdana'` and 10)
6. **`node_bg_meta`** and **`node_bg_color`**: A qualifier that determines the leaf label colors and a dictionary assigning colors to its values (defaults to `None` and `None`).
7. **`node_support_dict`** and **`support_bullet_size`**: A dictionary assigning support ranges to bullet colors, and the size of the bullets (defaults to `None` and `5`),
8. **`heat_map_meta`** and **`heat_map_colour_scheme`**: A list of qualifiers which will be the heatmap's columns, and the color scheme (defaults to `None` and 2 see ETE for color schemes)
9. **`pic_meta`**, **`pic_paths`**, **`pic_w`** and **`pic_h`**: You can put small images next to leaves. `pic_meta` will determine the qualifier according to which values an image will be assigned. pic_paths is a dictionary assigning image file paths to the qualifier's values. `pic_w` and `pic_h` are the dimensions of the images in pixels (the defaults are `None` for all the four keywords).
10. **`multifurc`**: Branch support cutoff under which to multifurcate nodes (default - `None`).
11. **`branch_width`** and **`branch_color`** (defaults: 2 and `DimGray`)
12. **`scale`**: This will determine the width of the tree (default 1000)
13. **`html`**: The path to which to write an html file with a list of the figures and links to the figure files (default `None`)

Example 1, the metadata determines background colours

```
bg_colors = {'present':'red',
             'absent': 'white'}

supports = {'black': [100,99],
            'gray': [99,80]}

pj.annotate('./images/',                # Path to write figs to

            'genus', 'Astrophorida',    # Set OTUs that have
                                        # 'Astrophorida'
                                        # in their 'genus' qualifier
                                        # as outgroup

            ['source_organism', 'record_id'], # leaf labels

            node_bg_meta='porocalices', # The qualifier that
```

```
                                    # will determine bg colors
    node_bg_color=bg_colors,        # The colors assigned to
                                    # each qualifier value

    node_support_dict=supports,

    html='./images/figs.html'
   )

pj.clear_tree_annotations()
```

In the resulting figure (below), clades of species with porocalices have red background, node with maximal relBootstrap support have black bullets, and nodes with branch support > 80 has gray bullets.

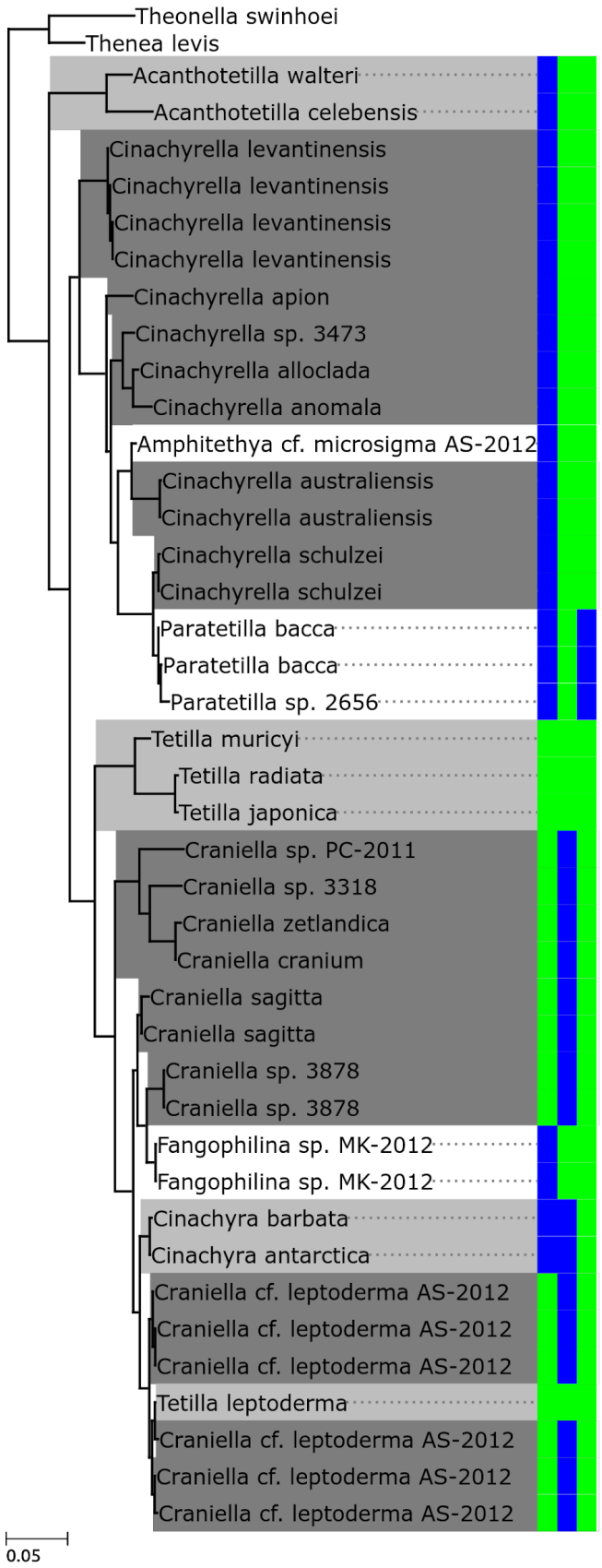


Example 2, the metadata as a heatmap

The second example introduces midpoint rooting and a heatmap. There are three columns in this heatmap, representing numerical values of three qualifiers. In this instance, the values are 0 or 1 for presence and absence. In addition, we change the branch colour to black and assign shades of gray to the genera.

```
bg_colors = {'Cinachyrella': 'gray',
             'Cinachyra': 'silver',
             'Amphitethya': 'white',
             'Fangophilina':'white',
             'Acanthotetilla':'silver',
             'Paratetilla':'white',
             'Craniella': 'gray',
             'Tetilla': 'silver',
             'Astrophorida': 'white'}

pj.clear_tree_annotations()

pj.annotate('./images/',                # Path to write figs to

            'mid', 'mid',               # Set midpoint root

            ['source_organism'],         # leaf labels

            fsize=13,

            node_bg_meta='genus',        # The qualifier that
                                         # will determine bg colors
            node_bg_color=bg_colors,     # The colors assigned to
                                         # each qualifier value

            # heatmap columns
            heat_map_meta=['porocalyx', 'cortex', 'calthrops'],

            heat_map_colour_scheme=0,

            branch_color='black',

            html='./images/figs.html'
           )
```

And this is what it looks like:


Theonella swinhoei
Thenea levis
Acanthotetilla walteri
Acanthotetilla celebensis
Cinachyrella levantinensis
Cinachyrella levantinensis
Cinachyrella levantinensis
Cinachyrella levantinensis
Cinachyrella apion
Cinachyrella sp. 3473
Cinachyrella alloclada
Cinachyrella anomala
Amphitethya cf. microsigma AS-2012
Cinachyrella australiensis
Cinachyrella australiensis
Cinachyrella schulzei
Cinachyrella schulzei
Paratetilla bacca
Paratetilla bacca
Paratetilla sp. 2656
Tetilla muricyi
Tetilla radiata
Tetilla japonica
Craniella sp. PC-2011
Craniella sp. 3318
Craniella zetlandica
Craniella cranium
Craniella sagitta
Craniella sagitta
Craniella sp. 3878
Craniella sp. 3878
Fangophilina sp. MK-2012
Fangophilina sp. MK-2012
Cinachyra barbata
Cinachyra antarctica
Craniella cf. leptoderma AS-2012
Craniella cf. leptoderma AS-2012
Craniella cf. leptoderma AS-2012
Tetilla leptoderma
Craniella cf. leptoderma AS-2012
Craniella cf. leptoderma AS-2012
Craniella cf. leptoderma AS-2012
0.05

3.10.3 Archive the analysis as a zip file

The `publish` function will produce an html human readable report containing a description of the data, alignments, trees, and the methods that created them in various ways. The following options control this function:

1. **`folder_name`**: zip file name or directory name for the report (will be created)
2. **`figures_folder`**: where did you save your tree figures?
3. **`size`**: `'small'` = don't print alignment statistics graph, or `'large'`: print them. If `'large'` is chosen, for each alignment and trimmed alignment, gap scores and conservation scores plots will be printed. (default - `'small'`).
4. **`compare_trees`**: a list of algorithms to use to formally compare trees. The algorithms to choose from are `'topology'`, `'branch-length'` and `'proportional'`. (default, [])
5. **`compare_meta`**: Similar to the OTU qualifier required for data concatenation, we need to say which qualifier identifies a discrete sample, that will allow to compare trees of different genes. By default, it will look for a `Concatenation` object and will use the OTU meta that is specified there. If there are no `Concatenation` objects, and we have not specified a `compare_meta`, it will raise an error.
6. **`trees_to_compare`**: A list of keys from the `pj.trees` dictionary. This allows to control what trees will go into the pairwise comparisons and also control their order of appearance in the results. (default, 'all')
7. **`unrooted_trees`**: `True` or `False` (default). If `True`, the algorithm will minimize the difference before determining it.

This is a minimal example, which does not include tree comparisons. Tree comparisons are shown later.

```
publish(pj, 'my_report', './images/', size='large')
```

```
checking if file exists
reporter was called by publish
starting species table
starting sequence statistics plots
starting concatenations
starting methods
starting alignment statistics
starting RF matrix(ces)
reporting trees
pickling
archiving
report ready
```

```
pickle_pj(pj, 'outputs/my_project.pkpj')
```

```
'outputs/my_project.pkpj'
```

4 Git and Pickle integration in ReproPhylo

This section demonstrates the interaction of ReproPhylo and of pickled ReproPhylo `Project` files with Git. In section 3 we disabled Git and saved the pickle file manually at the end of each sub section. However, ReproPhylo is designed to update the `Project's` pickle file automatically after time consuming steps and also to create a version control repository and record versions in real time. All of this will happen if we start a `Project` using the default setting `git=True`.

Once we start a `Project` this way, it can be the only version controlled `Project` in the current working directory. Any additional `Project` will have to be started with a different pickle name, and with `git=False`. Should it not be the case, helpful error messages will guide you through.

Also, once we started a `Project`, it can only be resumed with the command `unpickle_pj`. If we try to reconstruct the `Project` using the command `pj = Project(...)`, another helpful error message will be raised.

4.1 The long version

4.1.1 Start a `Project`, read data, do alignment, show Git log

Start a Project

As we did in section 3, we start a `Project`, and provide a pickle file name. We do not, however, use `git=False` and therefore git is invoked, as the default behaviour.

```
from reprophylo import *
pj = Project('git_demo_files/loci_edited.csv',
pickle='git_demo_files/git_demo')
```

```
/home/amir/Dropbox/python_modules/rpgit.py:93: UserWarning: Thanks to Stack-Overflow
users Shane Geiger and Billy Jin for the git wrappers code
  warnings.warn('Thanks to Stack-Overflow users Shane Geiger and Billy Jin for the
git wrappers code')
/home/amir/Dropbox/python_modules/rpgit.py:109: UserWarning: A git repository was
created in /home/amir/Dropbox/ReproPhylo/Tutorial_files/Git.
  warnings.warn('A git repository was created in %s.'%repoDir)
/home/amir/Dropbox/python_modules/reprophylo.py:255: UserWarning: The new repository
is called git_demo_files/git_demo.
  warnings.warn('The new repository is called %s.'%open(cwd + '/.git/description',
'r').read().rstrip())
DEBUG:Cloud:Log file (/home/amir/.picloud/cloud.log) opened
```

We get three warnings, which are only information messages.

- The first massage includes credit for some code I got online.
- The second gives us the location in which the repository will be maintained
- The third gives us the name of the repository

Read data

We can move on to reading data and aligning some loci:

```
genbank = './git_demo_files/Tetillidae.gb'
pj.read_embl_genbank([genbank])
```

Do alignment

```
pj.extract_by_locus()
mafft = AlnConf(pj)
pj.align([mafft])
```

```
mafft 217511440955273.78_CDS_proteins_MT-CO1.fasta
```

So our data was split to bins according the the `Locus` objects in the `Project`, and all the loci were aligned with the default settings of Mafft.

Show last Git action (which was to commit the pickle with the alignment)

At this point, let's check what pickle and git did at the background, by asking for git info:

```
pj.last_git_log()
```

```
Sun Aug 30 18:21:15 2015
STDOUT:
[master 09df506] AlnConf named mafftDefault with ID 217511440955273.78 Loci: MT-CO1
Created on: Sun Aug 30 18:21:13 2015 Commands: MT-CO1: mafft
217511440955273.78_CDS_proteins_MT-CO1.fasta
 1 file changed, 0 insertions(+), 0 deletions(-)

STDERR:None
>>>>
```

The last git action was to commit the pickle file, after the sequence alignment was complete. The git message is the report we get when we print the used method (from `pj.used_methods` if you recall).
We can show the full log like this:

```
pj.show_commits()
```

```
commit 09df506f5a5a003f1665d5abf52d11fb66755a90
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:21:15 2015 +0100

    AlnConf named mafftDefault with ID 217511440955273.78
    Loci: MT-CO1
    Created on: Sun Aug 30 18:21:13 2015
    Commands:
    MT-CO1: mafft 217511440955273.78_CDS_proteins_MT-CO1.fasta

    Environment:
    Platform: Linux-3.13.0-40-generic-x86_64-with-Ubuntu-14.04-trusty
     Processor: x86_64
     Python build: defaultJun 22 2015 17:58:13
     Python compiler: GCC 4.8.2
     Python implementation: CPython
```

```
     Python version: 2.7.6
     ete2 version: 2.2rev1056
     biopython version: 1.64
     dendropy version: 3.12.0
     cloud version: 2.8.5
     reprophylo version 1.0
     User: amir-TECRA-W50-A
     Program and version: MAFFT v7.123b\nPal2Nal v14
     Program reference:Katoh
     Standley 2013 (Molecular Biology and Evolution 30:772-780) MAFFT multiple
sequence alignment software version 7: improvements in performance and
usability.\nMikita Suyama
     David Torrents
     and Peer Bork (2006) PAL2NAL: robust conversion of protein sequence alignments
into the corresponding codon alignments.Nucleic Acids Res. 34
     W609-W612.
    execution time:
    1.39940595627

    ==============================
    Core Methods section sentence:
    ==============================
    The dataset(s) MT-CO1 were first aligned at the protein level using the program
MAFFT v7.123b [1].
    The resulting alignments served as guides to codon-align the DNA sequences using
Pal2Nal v14 [2].

    Reference:
    [1]Katoh, Standley 2013 (Molecular Biology and Evolution 30:772-780) MAFFT
multiple sequence alignment software version 7: improvements in performance and
usability.
    [2]Mikita Suyama, David Torrents, and Peer Bork (2006) PAL2NAL: robust
conversion of protein sequence alignments into the corresponding codon
alignments.Nucleic Acids Res. 34, W609-W612.

commit 5d9e94d44f88128374f0470d4006f4e6cb1ed10c
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:20:25 2015 +0100

    1 genbank/embl data file(s) from Sun Aug 30 18:20:25 2015

commit 0423808e2ef5bec77da2fdb482b7466916546da7
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:13:11 2015 +0100

    Project object with the loci MT-CO1, from Sun Aug 30 18:13:11 2015

commit abeaa0b6195f41049a04f0dbabfe87c9bdece320
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:13:11 2015 +0100

    2 script file(s) from Sun Aug 30 18:13:11 2015
```

This output is the complete list of git actions since we first started the `Project`, with the oldest at the bottom. Each action has a **commit hash**, the author of the commit, the time it was made, and an indented commit message. If we look at the messages from bottom to top we can see that so far we have done the following:

1. Saved relevant files that preexisted in the working directory when we started the git repository (2 script files, which are this notebook and its checkpoint)
2. Saved a pickle file of a `Project` with a single gene (MT-CO1)
3. Read a genbank file into the `Project` and updated the pickle file

4. Ran sequence alignment for the MT-CO1 gene using Mafft

4.1.2 Revert to older Project version

In addition to logging our actions, git allows us to 'undo' and 'redo' them by reverting to previous versions of the pickle file.

For example, let's say we want to cancel our latest sequence alignment. Our current `Project` has one alignment in it:

```
pj.alignments.keys()
```

```
['MT-CO1@mafftDefault']
```

To move back to when we had no alignments in the `Project`, we need the '**commit** hash' from our commits log, of the action the preceded the sequence alignment. The hash is the long alphanumeric string at the top of each commit, just a few characters from it's start shoud do it.

When I was writing this notebook, the git hash of the action which preceded the sequence alignment (one before last) was `5d9e94d44f88128374f0470d4006f4e6cb1ed10c`, but it will be something else for you. To move back to it I do:

```
pj = revert_pickle(pj, '5d9e94d4')
```

```
Git STDOUT:
Git STDERR:

/home/amir/Dropbox/python_modules/reprophylo.py:240: UserWarning: Git repository
exists for this Project
  warnings.warn('Git repository exists for this Project')
```

We get no output or errors from git, which is what we expect. When we revert, ReproPhylo restarts the `Project` and it lets us know that a git repository already exists, and it will keep using it.

Lets see how many alignments the `Project` has now:

```
pj.alignments.keys()
```

```
[]
```

Right. No alignments now. But wait, was this reversion a mistake? No problem. We can get our alignment back. The git hash for the alignment step is `09df506f5a5a003f1665d5abf52d11fb66755a90` (will be something else for you). Let's get it back:

```
pj = revert_pickle(pj, '09df506f5')
pj.alignments.keys()
```

```
Git STDOUT:
Git STDERR:
['MT-CO1@mafftDefault']
```

OK! No git error messages, and we have our alignment back in `pj.alignments`.

4.1.3 Recovering from unintentional changes

Now lets do something stupid: We will make a new `AlnConf` object, with different run parameters, but without changing the name of the `AlnConf` object, thus overwriting the resulting alignment of the previous one. For this alignment step, this is not the end of the world, since it is very quick. However, this will work the same for long analyses, such as tree reconstruction or when there is a lot of data.

```
new_mafft = AlnConf(pj, cline_args=dict(localpair=True,
                                        maxiterate=1000))
pj.align([new_mafft])
```

```
mafft --localpair --maxiterate 1000 611281440957509.19_CDS_proteins_MT-CO1.fasta
```

Now, checking the `used_methods` dictionary, we realize the gravity of our mistake, as the new `AlnConf` is stored under the same key as the old one, which is now gone from both the `used_methods` and the `alignment` dictionaries:

```
print 'Alignments:'
print pj.alignments
print
print 'Used Methods:'
print pj.used_methods
```

```
Alignments:
{'MT-CO1@mafftDefault': <<class 'Bio.Align.MultipleSeqAlignment'> instance (92
records of length 1566, IUPACAmbiguousDNA()) at 7f239f2b2950>}

Used Methods:
{'mafftDefault': <reprophylo.AlnConf instance at 0x7f239f52f488>}
```

Checking the string representation of the `AlnConf` object, which has the same name as the old one, will confirm it shows the new command line, rather than the old one:

```
print pj.used_methods['mafftDefault']
```

```
AlnConf named mafftDefault with ID 611281440957509.19
Loci: MT-CO1
Created on: Sun Aug 30 18:58:29 2015
Commands:
MT-CO1: mafft --localpair --maxiterate 1000
611281440957509.19_CDS_proteins_MT-CO1.fasta

Environment:
Platform: Linux-3.13.0-40-generic-x86_64-with-Ubuntu-14.04-trusty
 Processor: x86_64
 Python build: defaultJun 22 2015 17:58:13
 Python compiler: GCC 4.8.2
 Python implementation: CPython
 Python version: 2.7.6
 ete2 version: 2.2rev1056
 biopython version: 1.64
 dendropy version: 3.12.0
```

```
 cloud version: 2.8.5
 reprophylo version 1.0
 User: amir-TECRA-W50-A
 Program and version: MAFFT v7.123b\nPal2Nal v14
 Program reference:Katoh
 Standley 2013 (Molecular Biology and Evolution 30:772-780) MAFFT multiple sequence
alignment software version 7: improvements in performance and usability.\nMikita
Suyama
 David Torrents
 and Peer Bork (2006) PAL2NAL: robust conversion of protein sequence alignments into
the corresponding codon alignments.Nucleic Acids Res. 34
 W609-W612.
execution time:
3.97148609161


==============================
Core Methods section sentence:
==============================
The dataset(s) MT-CO1 were first aligned at the protein level using the program
MAFFT v7.123b [1].
The resulting alignments served as guides to codon-align the DNA sequences using
Pal2Nal v14 [2].

Reference:
[1]Katoh, Standley 2013 (Molecular Biology and Evolution 30:772-780) MAFFT multiple
sequence alignment software version 7: improvements in performance and usability.
[2]Mikita Suyama, David Torrents, and Peer Bork (2006) PAL2NAL: robust conversion of
protein sequence alignments into the corresponding codon alignments.Nucleic Acids
Res. 34, W609-W612.
```

Thanks to the Git repository, it is possible to recover from this blunder. We can spot an old version that contains the original alignment step and revert to it.

```
pj.show_commits()
```

```
commit 1e11023bab07af3882b10bae65053301c0c16997
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:58:33 2015 +0100

    AlnConf named mafftDefault with ID 611281440957509.19
    Loci: MT-CO1
    Created on: Sun Aug 30 18:58:29 2015
    Commands:
    MT-CO1: mafft --localpair --maxiterate 1000
611281440957509.19_CDS_proteins_MT-CO1.fasta

    Environment:
    Platform: Linux-3.13.0-40-generic-x86_64-with-Ubuntu-14.04-trusty
     Processor: x86_64
     Python build: defaultJun 22 2015 17:58:13
     Python compiler: GCC 4.8.2
     Python implementation: CPython
     Python version: 2.7.6
     ete2 version: 2.2rev1056
     biopython version: 1.64
     dendropy version: 3.12.0
     cloud version: 2.8.5
     reprophylo version 1.0
     User: amir-TECRA-W50-A
     Program and version: MAFFT v7.123b\nPal2Nal v14
     Program reference:Katoh
     Standley 2013 (Molecular Biology and Evolution 30:772-780) MAFFT multiple
sequence alignment software version 7: improvements in performance and
```

```
usability.\nMikita Suyama
     David Torrents
     and Peer Bork (2006) PAL2NAL: robust conversion of protein sequence alignments
into the corresponding codon alignments.Nucleic Acids Res. 34
     W609-W612.
    execution time:
    3.97148609161

    ==============================
    Core Methods section sentence:
    ==============================
    The dataset(s) MT-CO1 were first aligned at the protein level using the program
MAFFT v7.123b [1].
    The resulting alignments served as guides to codon-align the DNA sequences using
Pal2Nal v14 [2].

    Reference:
    [1]Katoh, Standley 2013 (Molecular Biology and Evolution 30:772-780) MAFFT
multiple sequence alignment software version 7: improvements in performance and
usability.
    [2]Mikita Suyama, David Torrents, and Peer Bork (2006) PAL2NAL: robust
conversion of protein sequence alignments into the corresponding codon
alignments.Nucleic Acids Res. 34, W609-W612.

commit 809b314e27f5a3303f64a2ecf3a1556b4cd327bd
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:54:19 2015 +0100

    2 script file(s) from Sun Aug 30 18:54:19 2015

commit 39434c1c76a39b9a5f8cda246c714e30817a3138
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:49:46 2015 +0100

    2 script file(s) from Sun Aug 30 18:49:46 2015

commit 09df506f5a5a003f1665d5abf52d11fb66755a90
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:21:15 2015 +0100

    AlnConf named mafftDefault with ID 217511440955273.78
    Loci: MT-CO1
    Created on: Sun Aug 30 18:21:13 2015
    Commands:
    MT-CO1: mafft 217511440955273.78_CDS_proteins_MT-CO1.fasta

    Environment:
    Platform: Linux-3.13.0-40-generic-x86_64-with-Ubuntu-14.04-trusty
     Processor: x86_64
     Python build: defaultJun 22 2015 17:58:13
     Python compiler: GCC 4.8.2
     Python implementation: CPython
     Python version: 2.7.6
     ete2 version: 2.2rev1056
     biopython version: 1.64
     dendropy version: 3.12.0
     cloud version: 2.8.5
     reprophylo version 1.0
     User: amir-TECRA-W50-A
     Program and version: MAFFT v7.123b\nPal2Nal v14
     Program reference:Katoh
     Standley 2013 (Molecular Biology and Evolution 30:772-780) MAFFT multiple
sequence alignment software version 7: improvements in performance and
usability.\nMikita Suyama
     David Torrents
```

```
     and Peer Bork (2006) PAL2NAL: robust conversion of protein sequence alignments
into the corresponding codon alignments.Nucleic Acids Res. 34
     W609-W612.
    execution time:
    1.39940595627

    ==============================
    Core Methods section sentence:
    ==============================
    The dataset(s) MT-CO1 were first aligned at the protein level using the program
MAFFT v7.123b [1].
    The resulting alignments served as guides to codon-align the DNA sequences using
Pal2Nal v14 [2].

    Reference:
    [1]Katoh, Standley 2013 (Molecular Biology and Evolution 30:772-780) MAFFT
multiple sequence alignment software version 7: improvements in performance and
usability.
    [2]Mikita Suyama, David Torrents, and Peer Bork (2006) PAL2NAL: robust
conversion of protein sequence alignments into the corresponding codon
alignments.Nucleic Acids Res. 34, W609-W612.

commit 5d9e94d44f88128374f0470d4006f4e6cb1ed10c
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:20:25 2015 +0100

    1 genbank/embl data file(s) from Sun Aug 30 18:20:25 2015

commit 0423808e2ef5bec77da2fdb482b7466916546da7
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:13:11 2015 +0100

    Project object with the loci MT-CO1, from Sun Aug 30 18:13:11 2015

commit abeaa0b6195f41049a04f0dbabfe87c9bdece320
Author: Amir Szitenberg <szitenberg@gmail.com>
Date:   Sun Aug 30 18:13:11 2015 +0100

    2 script file(s) from Sun Aug 30 18:13:11 2015
```

The git log lists a sequence alignment at the top, the very last alignment we ran. But we want to revert to an earlier sequence alignment. If we scroll down the log we can find this earlier alignment and get its git hash. For me it is `09df506f5a5a003f1665d5abf52d11fb66755a90` but it will be something else for you.

Wait! before we revert, we need to grab hold of the new alignment and its used method, so that we can add them to the `Project` under a different method name, after we revert:

```
latest_alignment_object = pj.alignments['MT-CO1@mafftDefault']
latest_used_method = pj.used_methods['mafftDefault']
```

now we can revert:

```
pj = revert_pickle(pj, '09df506f5a')
```

```
Git STDOUT:
Git STDERR:
```

Good. Last step, we add the latest alignment and used method, **but with a different name**:

```
new_name = 'mafft_linsi'
```

```
# add the alignment to the Project
pj.alignments['MT-CO1@' + new_name] = latest_alignment_object

# Fix the used method name
latest_used_method.method_name = new_name

# Add the latest used method to the used_methods dict:
pj.used_methods[new_name] = latest_used_method
```

How many alignments and used methods are there now?

```
pj.alignments.keys()
```

```
['MT-CO1@mafft_linsi', 'MT-CO1@mafftDefault']
```

```
pj.used_methods.keys()
```

```
['mafftDefault', 'mafft_linsi']
```

Good. Now we have the `Project`, with the two alternative sequence alignments of the MT-CO1 gene. Nothing is lost, nothing had to be rerun, thanks to git.

We're not done!

The Project is automatically pickled when we

- Read data
- Read metadata
- Run alignment, trimming or tree reconstruction
-

We have done nothing of those as our last step, so the pickle is not up to date. Let's save it:

```
pickle_pj(pj, 'git_demo_files/git_demo')
```

```
'git_demo_files/git_demo'
```

OK, now we're done. We can turn the machine off. Next time we'll start as follows and carry on from where we stopped (`git=True` by default):

```
pj = unpickle_pj('git_demo_files/git_demo')
```

4.2 Possible error messages

If you are not using the Docker ReproPhylo distribution, and you are new to Git, you might get the following error when you start a new `Project` with `pj=Project('loci_file',pickle='pikle_filename')`:

```
RuntimeError: Git: set your email with '!git config --global
user.email "your_email@example.com"' or disable git (the ! is needed
```

in Jupyter Notebook. In a terminal, ommit it)

This is because git expects your email to be configured. To configure it, run the following in a terminal:

```
git config --global user.email "your_email@example.com"
```

Another possible error when you start a new `Project` with `pj=Project('loci_file',pickle='pikle_filename')`, as opposed to loading one with `unpickle_pj` or with `revert_pickle`, can arise because `Project` expects pickle to be a file name that does not yet exist. Otherwise, the following error will be raised,

IOError: Pickle git_demo_files/git_demo exists. If you want to keep using it do pj=unpickle_pj('git_demo_files/git_demo') instead.

to protect you from unintentionally deleting existing projects.

ReproPhylo also tries to make sure that an unpickled, reverted or new `Project` can identify its unique Git repository. This connection can be broken if a Git repository already existed in the working directory, which does not belong to the current `Project` or if the pickle file was moved independently from the directory in which it is found. The Git repository is found in a directory called `.git`, which is a hidden directory. To view hidden files and folders in your file browser, click `ctrt+H`. If you want to move the `Project` to another location, the folder containing both the `.git` directory and the pickle file must be moved as one unit. Should the connection between a `Project` and its Git repository be broken, the following error will be show:

RuntimeError: The Git repository in the CWD does not belong to this project. Either the pickle moved, or this is a preexsisting repo. Try one of the following: Delete the local .Git dir if you don't need it, move the pickle and the notebook to a new work dir, or if possible, move them back to their original location. You may also disable Git by with stop_git().

Note that even if the link between a repository and a project was broken, the pickle file still contains the full `Project` and is totally usable, by passing `git=False`, like this:
`pj=unpickle_pj('my_pickle_file', git=False)`

4.2 The short version

```
# Show the last git action
pj.last_git_log()

# Show all the commits in the git repository
pj.show_commits()

# Revert to a previous commit
# Using a hash from the commits list
pj = revert_pickle(pj, '5d9e94d4')
```

5. Jupyter notebooks with use cases

Parameter and data exploration 1

Parameter and data exploration 2

6. Tools in ReproPhylo

Abstract methods for including tools in ReproPhylo are in the queue for development. Currently , the way to include tools go through hands on modification of the ReproPhylo script. To help with this, some hints have been written in as comments, and they are searchable using the phrase `PROGRAM PLUG`. For example, a programme needs to have a default path in the following section of the code:

```
922      self.defaults = {'raxmlHPC': programspath+'raxmlHPC-PTHREADS-SSE3',
923                       'mafft': 'mafft',
924                       'muscle': programspath+'muscle',
925                       'trimal': programspath+'trimal',
926                       'pb': programspath+'pb',
927                       'bpcomp': programspath+'bpcomp',
928                       'tracecomp': programspath+'tracecomp',
929                       'fasttree': programspath+'FastTreeMP',
930                       'pal2nal': programspath+'pal2nal.pl',
931                       # PROGRAM PLUG
932                       # 'program name': programpath+'the basic command'
933                      }
```

where `programspath == ''`, except in the WinPython version where it points to the default programmes directory.

The user runs programmes using `Project` methods. As an example, the `tree` method interacts with the programme in several places. First, it requires version reference information:

```
2606        # PROGRAM PLUG
2607        # NOTE: THIS METHOD SERVES ALL PHYLO PROGRAMS ALTHOUGH THE ITERATOR IS
2608        # CALLED raxml_method
2609        # THIS GETS THE VERSION AND REFERENCE OF THE PROGRAM
2610
2611        # elif isinstance(raxml_method, Conf object name):
2612        #    p = sub.Popen(raxml_method.cmd+" command that writes version",
shell=True, stderr=sub.PIPE, stdout=sub.PIPE)
2613        #    raxml_method.platform.append('Program and version: '+
2614        #                            # 1 for stderr, 0 for stdout
2615        #                            p.communicate()[1].splitlines()[# get the
line and split])
2616        #    raxml_method.platform.append('Program reference: write the
reference here')
```

then it needs to execute the command line stored in the `Conf` object

```
2625                # PROGRAM PLUG
```

```
2626                # THIS RUNS THE PROGRAM
2627                # elif isinstance(raxml_method, Conf Object Name):
2628                #     sub.call(cline, shell=True)
```

finally, it needs to get the output and place it in the `Project`

```
2663        # PROGRAM PLUG
2664        # NOTE: THIS IS SIMPLIFIED. MIGHT WORK WITH SOMETHING LIKE
2665        # FASTTREE. SEE MORE EXAMPLES ABOVE
2666        # THIS SECTION MAKES A Tree OBJECT OUT OF THE OUTPUT FILE
2667
2668        # elif isinstance(raxml_method, Conf object name):
2669        #    base_name = "%s_%s"%(raxml_method.id, trimmed_alignment)
2670        #    tree_file = "the form of the output file with the %s"%base_name
2671        #    t = Tree(tree_file)
```

The user configures the programme execution using `Conf` objects. `PROGRAM PLUG` hints for the

`Conf` objects are included in the `AlnConf` class.

7. ReproPhylo module index

This section provides a detailed index of the objects, object methods and preliminaries if each object. Preliminaries are functions designed to be invoked by the object's methods but may also be useful on their own right and therefore worth mentioning. Each entry will include usage, raised errors and known issues. The index is organised as follows:

7.1. The Locus object

A Locus instance contains a description of a gene locus, required for Project methods to function correctly

7.1.1. Locus

Usage:

```
locus = Locus(char_type=char_type, feature_type=feature_type, name=name,  aliases=aliases)
```

***char_type*: dna or prot.** The molecule type you want to analyses.

***feature_type*: genbank feature type (eg, CDS, gene, tRNA).** Needed also for denovo data, when the actual type is not important, as long as you specify the same type for your data with 'add_feature_to_record'

***name*: any string.** Preferably the gene or product values as they appear in the genbank file, if one is used. No spaces allowed. Preferably, the same name should be designated to your denovo data with 'add_feature_to_record' .

***aliases*: a list of names used to represent the locus in the product or gene qualifiers on genbank.** Must be identical to the way it appears in the genbank file (including whitespaces). Each record feature that you want to include in the analysis must have the *name* or one of the *aliases* in the gene or product qualifier, which appears in the genbank file, or as was specified with 'add_feature_to_record' .

errors will be raised when:

- char_type is anything but dna or prot
- feature_type is not a string
- aliases is not a list
- any value in aliases is not a string

7.1.2. Locus methods

```
__str__(self)
```

If L is a `Locus` object, `str(L)` will return a string representation of it.

7.2. The Concatenation object

A Concatenation instance contains instructions on how to concatenate loci into a super matrix. It determines which loci to include, which loci must exist for all the OTUs in the supermatrix and which are optional.

7.2.1. Concatenation

Usage

```
c = Concatenation(name, loci,otu_meta,otu_must_have_all_of = [], otu_must_have_one_of =
                    'all',define_trimmed_alns = [])
```

***name*: any string**, have to be unique.

***loci*: a list of Locus objects**

***otu_meta*: the name of the qualifier (or column in the CSV file) which specifies the OTU**. This column may have the same value for different record features, indicating that the sequences belong to the same sample and should be concatenated.

***otu_must_have_all_of*: a list of Locus object names representing sequences that must exist for each of the OTUs in the supermatrix.** OTUs that do not have this sequence will be excluded from the supermatrix.

***otu_must_have_one_of*: a list of lists of Locus object names**. The OTUs in the supermatrix must have at least one locus in each list.

***define_trimmed_alns*: list of strings which are partial or complete tokens of trimmed alignments.** Tokens are the names of the trimmed alignment. Trimmed alignment tokens have three values connected by a '@', representing the locus name, the alignment name and the trimming name. For example, "cox1@mafftDefault@trimal". When a locus has a single trimmed alignment, it will be used without checking this list. If there is more than one trimmed alignment for a locus, this list will be checked for clues regarding which alignment should be taken. For example, if all the loci were aligned twice, once with MAFFT and again with Muscle, and we wish to use only the mafft trimmed alignments, assuming the AlnConf.method_name is "mafftDefault", the value of "*define_trimmed_alns*" should be ["mafftDefault"]. If we want to use the MAFFT alignment for cox1 and the Muscle alignment for cytb, the value can be ["cox1@mafftDefault", "cytb@muscleDefault"]

errors will be raised when:

- loci contain values which are not Locus objects
- a Locus.name repeats more than once in loci

known issues:

You are protected from having a locus represented by less than four sequences in the supermatrix, the locus will be dropped. You are NOT protected from having no overlap in the loci, ie, each OTU is represented by different loci with no overlap, if your specifications allow it.

7.2.2. Concatenation methods

```
__str__(self)
```

If C is a `Concatenation` object, `str(C)` will return a string representation of it.

7.2.3. The Project object

The Project contains all the input data and analysis output. It also records the (phylogenetic analysis) methods used. The majority of steps taken are done using Project (python) methods. The Project can be saved as a pickle file which can then be read back in order to add data or modify the analysis. It is the best reproducibility option in ReproPhylo, but there also alternative 'back up' strategies such as keeping the sequence records and metadata in a GenBank format, as well as the trees and the various stages of the alignments in a chosen format.

7.3. Project

Usage

```
project = Project("loci_file.csv")
```

or

```
project = Project(loci_objects_list)
```

Attributes

The attributes are populated as the analysis progresses. The contain the inputs, outputs and information on the analysis. If project is a Project instance, it will have the following attributes.

***project.loci*: a list of Locus objects**

***project.records*: a list of SeqRecord objects of the input data.** Records from genbank files only retain features that fit one of the Locus objects. Empty list by default

***project.starttime*: a formatted string representing the time the project was initiated**

***project.user*: a list of items containing user specified information.** It takes its content from a file named USER, placed in the cwd by the user, which contains lines with the format keyword=value. It is a good place to record any important aspect of the analysis not recorded automatically, such as the search phrase that was used in GenBank and when GenBank was accessed, for example:

```
name=Amir Szitenberg

email=A.Szitenberg@hull.ac.uk

GenBank search phrase = Tetillidae[orgn]

GenBank accessed on = 24/12/2014
```

By default it is an empty list

***project.records_by_locus*: a dictionary with Locus object names as keys and lists of SeqRecord objects as values.** The SeqRecord objects are the precise sequence feature described by the Locus object. For example, if we have a nuclear protein CDS Locus with name X, and we provide a genbank entry of the DNA sequence containing both exons and introns, the SeqRecord object in project.records_by_locus['X'] will contain only the exons, ie the CDS. It will have no metadata associated. The record ID will be the accession number with the suffix "_f0" if it is the first feature from that genbank entry to be used, "_f1" if it is the second and so forth. The metadata associated with the record feature will be accessed using the extended ID (original accession plus the suffix). This attribute is populated using the method project.extract_by_locus(). By default it is an empty dictionary.

***project.concatenations*: a list of Concatenation objects.** Each with a unique name attribute. They can be added to the Project using project.add_concatenation(). The default value is an empty list.

***project.alignments*: a dictionary with alignment tokens as keys and MultipleSeqAlignment objects as values.** Alignment tokens have the form of Locus.name@AlnConf.method_name. For example, "cox1@mufftDefaults". The project.alignments attribute gets populated by passing an AlnConf object list to the project.align method. By default, project.alignments is an empty dictionary.

***project.trimmed_alignments*: a dictionary with trimmed alignment tokens as keys and MultipleSeqAlignment objects as values.** Trimmed alignment tokens have the form of Locus.name@AlnConf.method_name@TrimalConf.method_name. For example, "cox1@mufftDefaults@gappyout". The prject.trimmed_alignments attribute gets populated by passing a TrimalConf object list to the project.trim method. By default, project.trimmed_alignments is an empty dictionary.

***project.trees*: a dictionary with tree tokens as keys and Tree objects as values.** Tree tokens have the form of Locus.name@AlnConf.method_name@RaxmlConf.method_name.

For example, “cox1@mufftDefaults@raxmltree”. The project.trees attribute gets populated by passing a RaxmlConf object list to the project.tree method. By default, project.trees is an empty dictionary.

project.used_methods: AlnConf, TrimalConf and RaxmlConf objects that passed through project.align, project.trim and project.tree respectively are backed up in the project.used methods list for subsequent reporting. In picked Project objects, they are replaced by string representations of the Conf objects. By default, project.used_methods is an empty list.

***project.aln_summaries*: a list of lists of strings**. Each list contains string items providing the following information about the alignments and trimmed alignments in the Project: token, number of columns, number of rows, number of unique sequences, number of completely undetermined (all gaps) sequence, number of variable columns, number of parsimony informative columns and the average gap proportion. It is populated by project.align and project.trim. By default, it is an empty list.

***project.defaults*: a dictionary with program names as keys (red) and command paths as values (green)**. The keys have to be kept as they are. By default, it is assumed that all the programs are executable and are in the path:

```
{'raxmlHPC':
'raxmlHPC-PTHREADS-SSE3','mafft':
'mafft','muscle': 'muscle', 'trimal':
'trimal','pb': 'pb', 'bpcomp': 'bpcomp',
'tracecomp': 'tracecomp', 'pal2nal':
'pal2nal.pl'}
```

7.3.1. Project methods

`project.read_embl_genbank(filenames_list)`

***filenames_list*: a list of strings.** The strings are paths to genbank or embl formatted files. According to the Locus objects in project.loci, the records will be stripped from sequence features which are not needed for the analysis. In addition, the remaining features will be given a feature ID qualifier, as well as sequence length, GC content and the proportion of ambiguity symbols qualifiers. If the sequence is protein or the feature is a coding sequence, protein ambiguity symbols proportion will be added as well. If start_git was used, the input files will be added to the repository, and .ipynb and .py files will be updated.

`project.read_denovo(filenames_list, char_type, format = 'fasta')`

***filenames_list*: a list of strings.** The strings are paths to input files. If start_git was used, the input files will be added to the repository, and .ipynb and .py files will be updated. All the files must have the same character type, either DNA or protein. The method can be used twice consecutively in order to read both DNA and protein data into the same Project. A source feature will be created for each sequence. The record will be given a record ID of the form ‘denovo0'. The record id will be placed in the ‘original_id' qualifier. The record description, if exists, will go into the ‘original_desc' qualifier. The new source feature will be given a feature id of the form denovo0_source, which will be placed in the feature_id qualifier of the source feature. If the sequences are aligned, the gaps will be reset and a warning will be raised (to read alignments as alignments see project.read_alignment). The source feature on its own is insufficient. At least one more feature has to be created with project.add_feature_to_record.

***char_type*: ‘dna' or ‘protein'.** Describes the character type of the sequences in the files read.

***format*: the input files format.** ‘fasta' by default. Can be any Biopython SeqIO or AlignIO

format. All the files must have the same format. To read files with different formats use consecutively.

`project.read_alignment(filename, char_type, feature_type, locus_name, format="fasta", aln_method_name = "ReadDirectly", exclude=[])`

***filename*: a string.** The string is a path to input an input alignment file. If start_git was used, the input file will be added to the repository, and .ipynb and .py files will be updated. The file will be read as a MultipleSeqAlignment and placed into project.alignment with the token "locus_name@aln_method_name" as key. The records will also be treated in the same manner as they would have been if the file was read with project.read_denovo, with two exceptions: first, both a source feature and an additional feature will be created. For the records to be used, char_type, feature_type and locus_name have to fit one of the Locus objects in project.loci. Second, the same feature qualifiers will be added as when a genbank file is read using project.read_embl_genbank.

***char_type*: 'dna' or 'protein'.** Describes the character type of the sequences in the files read.

***feature_type*: a string.** Needs to fit one of the feature types in project.loci.

***format*: the input files format.** 'fasta' by default. Can be any Biopython SeqIO or AlignIO format.

***aln_method_name*: any string.** will be used in tokens serving as keys in project.alignments, project.trimmed_alignments and project.trees. "ReadDirectly" by default.

`project.add_feature_to_record(record_id, feature_type, location='full', qualifiers={})`

***record_id*: string.** The id of the record to which a feature is added. The qualifiers 'GC content', 'nuc_degen_prop' and 'prot_degen_prop' will be added.

***feature_type*: a string.** Type of the added feature.

***location*: a list of three integers:** [start, end, strand]. end has to be larger then start. strand is either 1 or -1. By default, the whole sequence will be included. Biopython will raise an error if the sequence is shorter than the specified location length.

***qualifiers*: dictionary.** keys are feature qualifiers and values are the qualifier values.

`project.add_concatenation(concatenation_object)`

***concatenation_object*: a Concatenation object.** The object will be appended to project.concatenations. Values of used as record ids will be checked and characters that will break downstream analyses will be replaced with "_ro_". The original values will be backed up in a new qualifier.

errors will be raised when:

- concatenation_object is not a Concatenation object
- concatenation_object.name allready exists in project.concatenations (you can reset by doing `project.concatenations = []`)

`project.make_concatenation_alignments()`

Will create a supermatrix for each Concatenation object in project.concatenations and will put them in project.trimmed alignment using the concatenation.name as a key.

errors will be raised when:

- There is more than one trimmed alignment for a locus and no extra definitions are provides
- Cannot guess the prefered trimmed alignment based on the hits supplied via `define_trimmed_aln` in the Concatenation object

`project.write(filename, format = 'genbank')`

Will write the SeqRecord objects from project.records into a file, including their modifications. The modifications can include the exclusion of record features, because they did not match a Locus object. I will also include changes and additions made to feature qualifiers, either automatically or by the user. If start_git was used, filename will be added to the repository and .py and .ipynb files will be updated.

***filename*: output file name.**

***format*: string.** Either 'csv', to produce a tab delimited text file, or any Biopython recognizable format. 'genbank' by default.

`project.correct_metadata_from_file(csv_file)`

A CSV file written with project.write() can be edited manually and then read back into the Project in order to modify and add feature qualifiers. If start_git was used, csv_file will be added to the repository and .py and .ipynb files will be updated.

***csv_file*: CSV file name.** The CSV file needs to be tab delimited. It is easy enough to write one with project.write(), edit it and read it back. It is almost impossible to write one manually from scratch. When editing, be aware of typical errors.

`project.if_this_then_that(IF_THIS, IN_THIS, THEN_THAT, IN_THAT, mode = 'whole')`

Allows to search for a value in a certain qualifier and when found to put another (or the same) value in another (or the same) qualifier, either new or existing one. example

***IF_THIS*: any string.** A search phrase to look for.

***IN_THIS*: a qualifier name.**

***THAN_THAT*: any_string.** A value to introduce to the metadata

***IN_THAT*: a qualifier name.** The qualifier in which to put the new value. Can be a pre-existing or new qualifier.

***mode*: 'whole' or 'part'.** 'whole' means that only an identical match to IF_THIS will be considered a match. 'part' means that even if IF_THIS is a subset of the target, it will be taken as a match. For example, in order to get genera from species names, the 'part' mode needs to be used.

`project.add_qualifier(feature_ids, name, value)`

Add a qualifier and its value to features for which you specify their feature_id. example

***feature_ids*: a list of feature ids.** The feature ids can be retrieved by looking at a file generated with project.write().

***name*: any string.** The name of the new qualifier or existing qualifier.

***value*: any string.** The value of the new or existing qualifier.

`project.add_qualifier_from_source(qualifier)`

For each record in Project.records, this will duplicate the source feature qualifier specified by name and place it in all the other feature in the record. For example, for a genbank record with a source feature and a cox1 gene feature, project.add_qualifier_from_source('organism') will add the organism value to the new organism qualifier in the cox1 gene feature.

***qualifier*: a qualifier name present in source.** If there is no qualifier of this name in the source feature of some or all of the records, there will be no warning and no error.

`project.copy_paste_within_feature(from_qualifier, to_qualifier)`

Duplicate a qualifier within a feature, and give the duplicate a new name. This can be handy if you want to add values to a qualifier in some records, but you want to do it in another field to keep the original as is.

***from_qualifier*: string.** The name of the qualifier you want do duplicate within each record feature.

***to_qualifier*: string.** The name that you want to give to the duplicate. If the name exists, the values will be overwritten without warning.

`project.copy_paste_from_features_to_source(from_feature_qual, to_source_qual)`

***from_feature_qual*: a string.** The name of the non-source feature qualifier you want to copy into the source feature.

***to source_qual*: a string.** The name this qualifier will have in the source feature. Existing source qualifier with the same name will be overwritten.

`project.extract_by_locus()`

Will iterate over records and record features in project.records. For each non-source feature: if both the feature type and gene or product qualifiers fit the Locus.feature_type and Locus.name/Locus.aliases of any of the Locus objects in project.loci, the feature will be placed as a SeqRecord object with feature_id as SeqRecord.id in the list project.records_by_locus[Locus.name].

will raise errors when:

- trying to read a record that only has protein sequence as DNA Locus
- trying to read a DNA record that doesn't have a 'translation' qualifier as a Protein Locus.

`project.exclude(start_from_max=True, **kwargs)`

Will exclude the records specified in kwargs. This method can be used instead of project.extract_by_locus() by leaving start_from_max set to its default value - True. In such case, all the record features will be read to project.records_by_locus, except for the ones specified in kwargs. Alternatively, start_from_max can be set to - False. In this case, the excluded record features passed through kwargs will be removed from the *current* content of project.records_by_locus.

***start_from_max*: True or False.** True means that all the record features in project.records that fit any of the Locus objects will be included, except for the ones specified in kwargs. False

means the excluded features will be removed from the current content of project.records_by_locus

****kwargs*: a dictionary with Locus.name as keys and feautre id lists as values.** This dictionary can look like this, for example:

```
**{cox1: ['denovo0_f0',
 'AM45814_f3'], cytb:
 ['FR784125_f0']}
```

Another way to pass this value is as a list of keywords and arguments. For example:

```
pj.exclude(cox1=['denovo0_f0',  'AM45814_f3'], cytb=['FR784125_f0'])
```

Using this notation, the curly braces are omitted and the locus name is used as a name and not as a string (i.e. no quotation marks). While looking nicer, there is a catch with this approach. Locus names that start with a number (e.g. 18S) cannot be converted from string to name, and **the method will break**. Therefore, the first notation is safer in this case.

Two records will be excluded from the cox1 dataset and one from the 18S dataset.

will warn when:

- not all the specified feature ids are found
- a Locus name, as used in the kwargs keys, is not found in project.loci

`project.include(start_from_null=True, **kwargs)`

Will include the records specified in kwargs. This method can be used instead of project.extract_by_locus() by leaving start_from_null set to its default value - True. In such case, only the record features passed through kwargs will be read to project.records_by_locus. Alternatively, start_from_null can be set to - False. In this case, the included record features passed through kwargs will be added to the *current* content of project.records_by_locus.

***start_from_null*: True or False.** True means that only the record features in project.records that fit any of the Locus objects and are specified in kwargs, will be included. False means the included features will be added to the current content of project.records_by_locus

***kwargs*: a dictionary with Locus.name as keys and feautre id lists as values.** This dictionary can look like this, for example:

```
**{cox1: ['denovo0_f0',
 'AM45814_f3'], 18s:
 ['FR784125_f0']}
```

Two records will be added to the cox1 dataset and one to the 18S dataset.

will warn when:

- not all the specified feature ids are found
- a Locus name, as used in the kwargs keys, is not found in project.loci

`project.filter_by_seq_length(locus_name, min_length=0, max_length=None)`

Will remove sequences from project.records_by_locus if they are shorter than min_length, for the specified locus. If max_length is specified, it will also remove sequences that are longer than max_length. The sequences will not be removed from project.records. If you run project.extract_by_locus() again, the filtering will be undone. The same will happen with project.exclude() with **`start_from_max=True.`**

***min_length*: integer.** The minimum sequence length allowed. Zero by default.

- ***max_length*: integer.** The maximum sequence length allowed. Unlimited by default.

`project.filter_by_gc_content(locus_name, min_percent_gc=0, max_percent_gc=None)`

Will remove sequences from project.records_by_locus if they fall within the GC content range specifiers. The sequences will not be removed from project.records. If you run project.extract_by_locus() again, the filtering will be undone. The same will happen with project.exclude() with **`start_from_max=True.`**

***min_length*: integer.** The minimum sequence length allowed. Zero by default.

***max_length*: integer.** The maximum sequence length allowed. Unlimited by default.

`project.align(alignment_methods=[], pal2nal='defaults')`

Will run sequence alignments to the data of the loci specified in AlnConf objects and the programs and parameters specified. The method will run the command lines in the AlnConf object, will run pal2nal if required, and will produce platform, software and timing information which will be stored in the AlnConf object. It will also store the resulting alignments as a MultipleSeqAlignment in the project.alignments attribute and append the used AlnConf object to project.used_methods. Finaly it will add a list of alignment statistics to project.aln_summaries.

***alignment_methods*: AlnConf object list.**

***pal2nal*: path to executable.** The defaults value 'defaults' means the path will be taken from the project.defaults attribute. It can be overridden *ad-hoc* by providing the path as a string or for the duration of the analysis by changing it in project.defaults. For example, to change the path of pal2nal and using a non executable instance of it you can use the following command:

```
project.defaults['pal2nal'] = 'perl /home/user/program/pal2nal.pl'.
```

will warn when:

- alignments have less than four unique sequences and are therefore dropped.

will raise errors when a codon alignment is attempted and:

- cannot find CDSs for all the protein sequences
- cannot find a protein sequence for all the CDSs
- a CDS and its respective protein sequence are not of compatible lengths (CDS three times longer the protein

`project.trim(list_of_Conf_objects)`

Will run alignment trimming on alignments specified in TrimalConf objects and the programs and parameters specified. The method will run the command lines in the TrimalConf object, and will produce platform, software and timing information which will be stored in the TrimalConf object. It will also store the resulting trimmed alignments as MultipleSeqAlignment objects in the project.trimmed_alignments attribute and append the used TrimalConf object to project.used_methods.

***list_of_Conf_objects*: RaxmlConf object list.**

`project.tree(list_of_Conf_objects)`

Will run tree reconstruction on trimmed alignments specified in RaxmlConf objects and the programs and parameters specified. The method will run the command lines in theRaxmlConf

object, and will produce platform, software and timing information which will be stored in the RaxmlConf object. It will also store the resulting trees as a Tree object in the project.trees attribute and append the used RaxmlConf object to project.used_methods. Finally it will add also add a NHX string representation to project.trees.

***list_of_Conf_objects*: RaxmlConf object list.**

Also see important reporting functions (ie not Project methods) below

`project.species_vs_loci(outfile_name)`

Will print a CSV file with Locus names as columns and source organism qualifiers (species) as rows. The values will be the number of occurrences of each species in each Locus. Records without a source feature, or without an organism qualifier in the source feature will be summarized in a line called 'undef'.

***outfile_name*: output file name** for the CSV table.

`project.write_by_locus(format = 'fasta')`

Will write a sequence file for each Locus in project.records_by_locus. The file name will be Locus.name.format (eg, cox1.fasta). The sequence ids in the files will be the feature ids (eg, denovo0_f0, AM745218_f3).

***format*: string.** Any Biopython recognizable format.

`project.write_alns(id=['feature_id'], format = 'fasta')`

Will write a sequence alignment file for each MultipleSeqAlignment object stored in project.alignments.

***id*: a list of feature qualifier names,** as they appear in a genbank representation of project.records (A genbank file can be produced with project.write()). Names of source qualifiers have to be prefixed with "source_". Annotations have to be prefixed with "annotation_". The values of the qualifiers in the list will be used as sequence headers in the written files.The default value is ['feautre_id'].

***format*: any Biopython AlignIO format** in which the files should be written.

will raise an error when:

- there are no alignments in the Project

`project.write_trimmed_alns(id=['feature_id'], format = 'fasta')`

Will write a sequence alignment file for each MultipleSeqAlignment object stored in project.trimmed_alignments.

***id*: a list of feature qualifier names,** as they appear in a genbank representation of project.records (A genbank file can be produced with project.write()). Names of source qualifiers have to be prefixed with "source_". Annotations have to be prefixed with "annotation_". The values of the qualifiers in the list will be used as sequence headers in the written files.The default value is ['feautre_id'].

***format*: any Biopython AlignIO format** in which the files should be written.

will raise an error when:

- there are no trimmed alignments in the Project

`project.show_aln(token, id=['feature_id'])`

Will write an html file of the alignment or trimmed alignment indicated by the token and will show it in a new browser tab.

***token*: a key in the project.alignments or project.trimmed alignments dictionaries.**

***id*: a list of feature qualifier names,** as they appear in a genbank representation of project.records (A genbank file can be produced with project.write()). Names of source qualifiers have to be prefixed with "source_". Annotations have to be prefixed with "annotation_". The values of the qualifiers in the list will be used as sequence headers in the written files.The default value is ['feautre_id'].

`project.clear_tree_annotations()`

Will clear all node Face objects from all the trees

`project.write_nexml(output_name)`

Will write all the trees to a file in nexml format. The tree leaf attributes will include all the feature qualifiers as well as the aligned sequences and the trimmed-aligned sequence.

***output_name*: any string.**

`project.annotate(fig_folder, root_meta, root_value, leaf_labels_txt_meta, leaf_node_color_meta=None, leaf_label_colors=None, node_bg_meta=None, node_bg_color=None, node_support_dict=None, heat_map_meta = None, heat_map_colour_scheme=2, multifurc=None, scale = 1000, html = None)`

Will produce annotated .png representation of all the trees in the project. The basic operation of the method is described next, but it is not essential to understand in order to use. The method works by storing some Face and NodeStyle objects in the Tree objects, making and adding a TreeStyle and then rendering a png file for each tree using that TreeStyle. The TreeStyle is not retained in the Tree object. The Face and NodeStyle objects can be cleared from the trees by using project.clear_tree_annotations(). The options in project.annotate() represent a very small selection of the total ETE2 capabilities. It is therefore also possible to fetch a Tree objects and use ETE functions and Tree methods directly on it. For example, we can add a customized NodeStyle:

```
t = project.ft("cox1")

nstyle - NodeStyle()

# configure nstyle according to ETE2 manual

for n in t.traverse():

    n.set_style(nstyle)
```

***fig_folder*: a string.** A path in which to write the png files. Required.

***root_meta*: a qualifier name.** This qualifier will be used to look for values that match the values in root_value and will take the record features which have this value to belong to the outgroup. For midpoint rooting use "mid". Required.

***root_value*: a string.** Record features which have this value in the root_meta qualifier will be included in the outgourp. Required.

***leaf_label_text_meta*: a list of strings.** The strings are feature qualifier names which should be included in the leaf labels in the trees. Required.

***leaf_node_color_meta*: a string.** A qualifier name. The values in this qualifier will determine the label colors based on a dictionary passed through leaf_label_colors.

***leaf_label_colors*: a dictionary with qualifier values as keys and color names, RGB codes or a random color generator as values.** Possible examples:

```
# Using color names

colors = {'Tetilla' = 'red',

          'Craniella' = 'blue'}

# Using RGB codes

colors = {'Tetilla' = 'FF0000',

          'Craniella' = '0000CD'}

# Using random

colors = {'Tetilla' = random_color(h=None, l=None, s=None),

          'Craniella' = random_color(h=None, l=None, s=None)}

# You can fix any of the components by specifying h, l or s.
```

***node_bg_meta*: a string.** A qualifier name. The values in this qualifier will determine the background colors of tree clades based on a dictionary passed through node_bg_colors.

***node_bg_colors*: a dictionary with qualifier values as keys and color names, RGB codes or a random color generator as values.** Possible examples:

```
# Using color names

colors = {'Tetilla' = 'red',

          'Craniella' = 'blue'}

# Using RGB codes

colors = {'Tetilla' = 'FF0000',

          'Craniella' = '0000CD'}

# Using random

colors = {'Tetilla' = random_color(h=None, l=None, s=None),

          'Craniella' = random_color(h=None, l=None, s=None)}

# You can fix any of the components by specifying h, l or s.
```

***node_support_dict*: a dictionary of colors as keys and lists of two integers (upper and lower node support limits) as values.** This dictionary will determine the annotation of node supports on the tree, using colored bullets. For example:

```
# When node supports are percents (eg, bootstrap percentage)

supports = {'black':[100,99],

            'dimgray':[99,75],
```

```
                'silver':[75,50]}

        # When node supports are proportion values

        # (eg posterior probabilities)

        supports = {'black':[1,0.99],

                'dimgray':[0.99,0.75],

                'silver':[0.75,0.5]}
```

***heat_map_meta*: a list of feature qualifiers.** The values in all of these qualifiers have to be numeric. They will compose a vector to be used in a profice node feature which will be plotted with using a ProfileFace.

***heat_map_colour_scheme*: colors used to create the gradient from min values to max values.** 0=green & blue; 1=green & red; 2=red & blue (default). In all three cases, missing values are rendered in black and transition color (values=center) is white.

***multifurc*: a numeric value within the range of node supports.** For example, 1 - 100 for bootstarp support, 0 - 1 for posterior probabilities. Nodes with lower support than specified will be multifurcated.

***scale*: integer.** The tree width. It is the same as the ETE2 TreeStyle attribute named scale. The default value is 1000.

***html*: file path.** If specified, an html with links to the figures off all the trees will be written in the path.

Known issues:

- The node support legend orders randomly
- Multifurc is glitchi in some cases

`project.report_seq_stats()`

Will plot four figures, each with a box plot representation of sequence length, GC content, proportion of nucleotide and protein ambiguous positions, for each locus.

In the following methods, ***token*** is a search phrase that will be looked for amongst the keys of the relevant project attribute. For example, if we look for a trimmed alignmnet using the token “cox1”, we will get project.trimmed_alignmnets[“cox1@mafftDefault@gappyout”].

will raise an error when:

- the token is not found
- the token matches more than one key. In this case a more informative token is needed. In the case of out cox1 token, we may get an error message looking like this:

```
The token cox1 was found in more then one tree key:
[“cox1@mafftDefault@gappyout”, “cox1@muscleDefault@gappyout”]
```

we can copy and paste our more informative token from this error message and use cox1@**mafft**Default as our more specific token.

`project.ft(token)`

Fetch a Tree objects from project.trees using a token.

`project.fa(token)`

Fetch a MultipleSeqAlignment objects from project.alignments using a token.

`project.fta(token)`

Fetch a MultipleSeqAlignment objects from project.trimmed_alignments using a token.

`project.fr(locus_name, filter=None)`

Will fetch the SeqRecord objects of the specified locus.

***locus_name*: the name attribute of one of the Locus objects in project.loci.**

***filter*: a list of lists**. Every (sub)list is a pair of qualifier and value. If filter is specified, only records that have all the specified values in the specified qualifiers will be kept. For example, the command:

```
project.fr('cox1', filter = [['genus', 'Cinachyrella'],
['porocalyx','2']])
```

will return a list of cox1 SeqRecord objects which have the value 'Cinachyrella' in their genus qualifier and the value '2' in their porocalyx qualifier (see here for context).

7.4. ReproPhylo functions meant to be used directly

`list_loci_in_genbank(genbank_filename, control_filename, loci_report=None)`

***genbank_filename*: the path to the genbank file.**

***control_filename*: a path to write the loci CSV**

***loci_report*: a path to write the loci counts report.** If None, will be written to stdout.

will warn when

- a gene or product qualifiers are not found

will raise error when

- the file format breaks the Biopython genbank parser

known issues

for now only accept genbank files (not embl)

`pickle_pj(project, pickle_file_name)`

Will create a pickle file of the Project instance. Will commit it if start_git() was used. Will replace the Conf objects in project.used_methods with string representations of these Conf objects.

***project*: a Project instance.**

***pickle_file_name*: a path to write the file.**

known issues

The string representations of the Conf objects are badly formatted but readable.

`unpickle_pj(pickle_file_name)`

Will return a Project instance based on the pickle file passed.

***pickle_file_name*: the path to the pickle file**

usage:

```
project = unpickle_pj(pickle_file_name)

# project is now a Project instance. For example,

# it has the project.loci attribute.
```

known issues

The string representations of the Conf objects are badly formatted but readable.

`publish(project, folder_name, figures_folder)`

Will archive the following file into a zip file: a report produced with report_methods, a Project pickle file produced with pickle_pj, a genbank file containing project.records, a nexml file containing all the trees, with the aligned and trimmed aligned sequences as leaf attributes.

***project*: a Project instance**

***folder_name*: the name of the zip file** (.zip will be added if missing from this value)

***figures_folder*: a path to the tree figure files.** The files will be picked up only if they were generated using project.annotate() .

`calc_rf(project, figs_folder)`

Will calculate the robinson_foulds distance between each pair of trees in project.trees. Will return a list with two values. The first will be a heat map plot of the rf values with numerical labels, the second will be a list of strings forming a legend table which associate the labels in the figure with trees in project.trees.

***project*: a Project instance**

***figs_folder*: a path to write the heatmap plot png file.**

`draw_trimal_scc(project, num_col, figs_folder, trimmed=False, alg = '-scc')`

`view_csv_as_table(csv_filename, delimiter, quotechar='|')`

7.5. The AlnConf object

5.5.1. AlnConf

5.5.2. AlnConf methods

5.5.3. AlnConf preliminaries

7.6. The TrimalConf object

5.6.1. TrimalConf

5.6.2. TrimalConf methods

5.6.3. TrimalConf preliminaries

5.7. The RaxmlConf object

5.7.1. RaxmlConf

5.7.2. RaxmlConf methods

5.7.3. RaxmlConf preliminaries

78. Undocumented functions

The following functions are present in the module but are not yet covered by either the use cases or this command reference. They include the PbConf object, which can be used to run a phylobayes tree reconstruction, a bayestraits function that takes trait information from the project's metadata and a tree from the project's trees and exonerate functions designed to run exonerate and feed the data to a reprophylo project. The LociStats object allowing sorting and subsetting loci based on various statistics is also available.

The three top things which are under development are a more scalable report, faster feature iteration throughout, and multiparanoid functions.

8. A Galaxy workflow - Iguaninae data

Galaxy is an open, web browser based environment designed to provide workflow tools using a graphic user interface. Galaxy takes care of all the aspects of reproducibility by controlling the input and output files and their provenance in recorded histories. The histories record the order of the tools that were used, the input and output files and the choice of parameters. It is possible to export, publish and share Galaxy histories in a way that will allow others to import them directly into galaxy, review the analysis, repeat it and extend it.

The following use case focuses on an exploratory analysis of genbank data of iguanas. Since the ReproPhylo Galaxy tools are powered by the python module, they lag behind in versatility. the Galaxy tools do not allow us to configure the phylogenetic analysis at the moment. This is obviously a drawback which is the focus of the current development. However, they still provide a powerful environment for making sense of your data as well as publicly available data.

In this use case we will perform the following tasks:

- Obtain and install Galaxy with ReproPhylo
    - All inclusive on Ubuntu and similar OSs
    - Add only the ReproPhylo tools to your existing Galaxy
    - Try other methods to get galaxy if these ones are no good
- Get data from GenBank
- Explore and choose the loci to analyse
- Start a Project with the selected loci
- Read the GenBank records
- Explore the available metadata from the genbank file
- Add additional information of our own
- Run a fixed phylogenetic pipeline
- Annotate the resulting trees using the metadata
- Archive the results
- Tools not covered by this use case
- Export the history
- Make a workflow out of the history and edit it

8.1. Getting ReproPhylo in Galaxy

An archived Galaxy distribution with the ReproPhylo tools already set up can be downloaded here.

For any linux distribution which uses apt-get, you can extract the archive file and run the INSTALL.sh file.

```
$ cd ~/Downloads && unzip ReproPhyloGalaxy-master.zip

$ cd ~/Downloads/ReproPhyloGalaxy-master && sudo ./INSTALL.sh
```

It will place a `galaxy-dist` folder and will download dependencies using apt-get and pip. To run Galaxy, do the following:

```
$ cd ~/galaxy-dist && sudo sh run.sh --reload
```

The first start up takes time because Galaxy downloads dependencies as well. Subsequent start ups will be quick. Once it is done, you'll receive the message

```
serving on http://127.0.0.1:8080
```

Go to this address in your browser.

Later on, in order to quit Galaxy when you're done working, use ctrl+c.

If you want to add the ReproPhylo tools to your existing Galaxy download the archive here, and extract:

```
$ cd ~/Downloads && unzip ReproPhyloGalaxy-master.zip
```

Then copy the reprophylo directory to your tools directory:

```
$ cp -r ~/Downloads/ReproPhyloGalaxy-master/galaxy-dist/tools/reprophylo
/path/to/your/galaxy-dist/tools/.
```

Next, you'll have to update the tool_conf file with the new tools (also these instructions for more help). This file can be either

`/your-path/galaxy-dist/tool_conf.xml`,

or

`/your-path/galaxy-dist/config/tool_conf.xml.main`.

copy the text **in bold** into anywhere in the file, according to the place you want ReproPhylo to appear in your Galaxy tools menu. However, make sure not to break existing section blocks:

```
  <section id="some section" name="some section">
      <tool file="directory/some tool.xml" />
  </section>
  <section id="ReproPhylo" name="ReproPhylo">
    <tool file="reprophylo/start_a_project.xml" />
    <tool file="reprophylo/read_data.xml" />
    <tool file="reprophylo/read_revised_metadata.xml" />
    <tool file="reprophylo/run_exploratory_pipeline.xml" />
    <tool file="reprophylo/annotate_trees.xml" />
    <tool file="reprophylo/report.xml" />
    <tool file="reprophylo/manage_project.xml" />
   </section>
  <section id="some other section" name="some other section">
      <tool file="directory/some tool.xml" />
  </section>
```

You'll also need to install some more **dependencies**. On machines with apt-get, it is a good idea to start with

```
$ sudo apt-get update
```

You'll need to make sure you have python 2.7 or later, python-pip. (if you use apt-get you can do `sudo`

`apt-get install python python-dev python-pip`). On OSX homebrew is a good replacement for apt-get (ie `brew install [whatever]`).

Then get some python modules and MAFFT:

Biopython and ETE2 dependencies and modules

```
$ sudo apt-get install python-setuptools python-numpy python-qt4 python-scipy
python-mysqldb python-lxml

$ sudo apt-get install python-biopython

$ sudo pip install ete2
```

Dendropy

```
$ sudo pip install dendropy
```

Cloud

```
$ sudo pip install cloud
```

Pandas

```
$ sudo pip install pandas
```

Matplotlib

```
$ sudo apt-get build-dep python-matplotlib sudo apt-get install
python-matplotlib
```

and Mafft

```
$ sudo apt-get install mafft
```

Once this is all done you can start your Galaxy instance:

```
$ cd ~/galaxy-dist && sudo sh run.sh --reload
```

The first start up takes time because Galaxy downloads dependencies as well. Subsequent start ups will be quick. Once it is done, you'll receive the message

```
serving on http://127.0.0.1:8080
```

Go to this address in your browser.

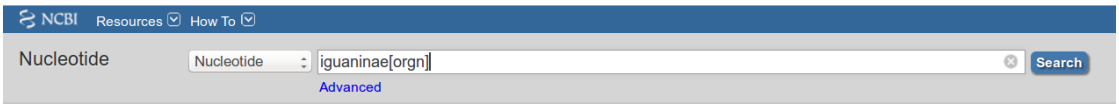
Later on, in order to quit Galaxy when you're done working, use ctrl+c.

Finally, if the above methods do not fit your system, you can check the Galaxy manual for their supported OSs.

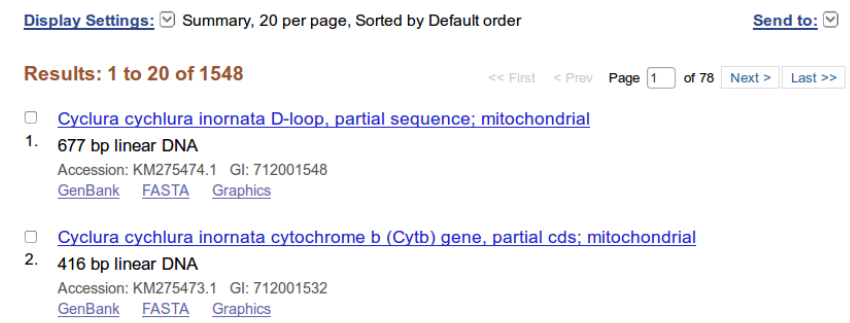
While developing, we target Ubuntu. However, I will be willing to attempt to assist with installation on other systems if all of the above fails. Contact me at A.Szitenberg@Hull.ac.uk.

8.2. Getting data from GenBank

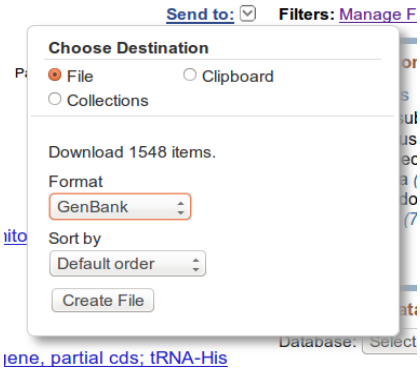
This is probably very obvious for most, but here's a short example, for the sake of completeness. In this use case we'll take all the available Iguaninae data from the Nucleotide database in GenBank. Type in the search phrase `iguaninae[orgn]` to the search box in the Nucleotide database:



The result page should look as follows:



Use the `Send to:` link on the top right hand side to download a genbank file:



The file will be saved in your Downloads directory and will be most likely called *sequence.gb*.

8.3. Uploading your data to Galaxy

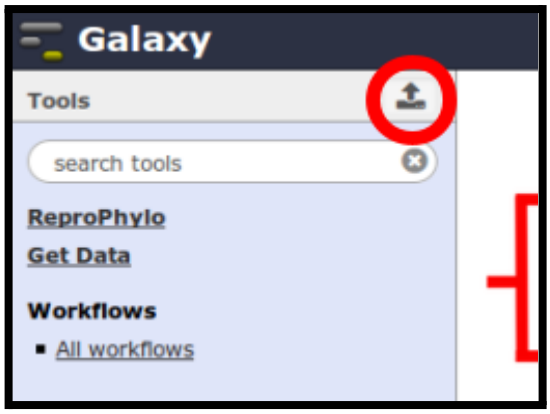
Since Galaxy is in your web browser, we will use the term “uploading” for getting data into Galaxy, even if it is local, just because this is a browser page terminology. If you haven't already started Galaxy, do it as follows. If this is the first start-up, it will take a little time:

```
$ cd /path/to/your/galaxy-dist/

$ sudo sh run.sh --reload
```

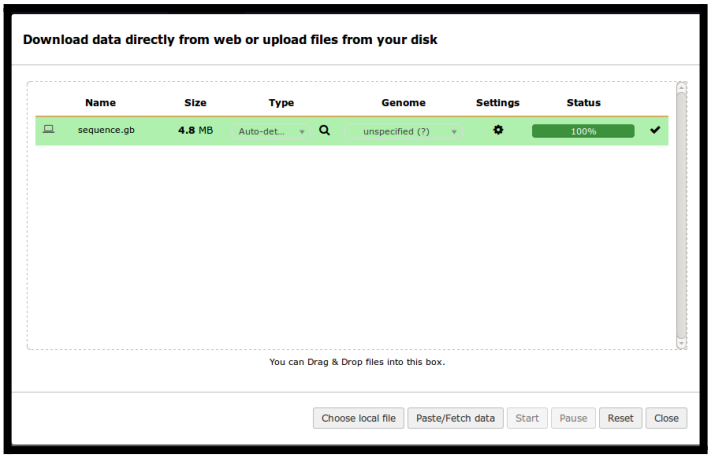
When you get the message:

```
serving on http://127.0.0.1:8080
```

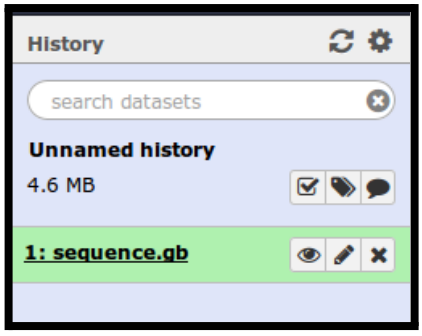
go to this address in your browser. The Galaxy page has the tools panel on the left, the analysis history on the right. Tool forms and output will appear in the middle. To upload your Iguaninae sequences click the icon on the top, right-hand side of the Tools panel:



The upload box will open. You can either drag and drop your file or locate it using the “Choose local file” button. Once dropped/ chosen, click the start button.



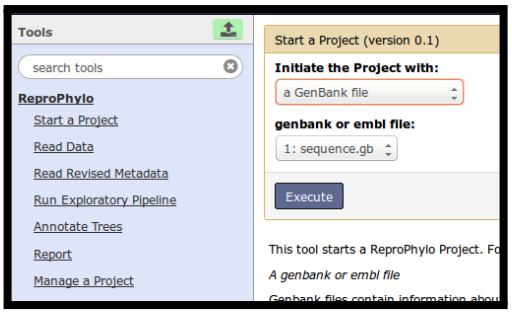
When the file has been uploaded you can close the box. You will then be able to spot your file at the top of the History panel.



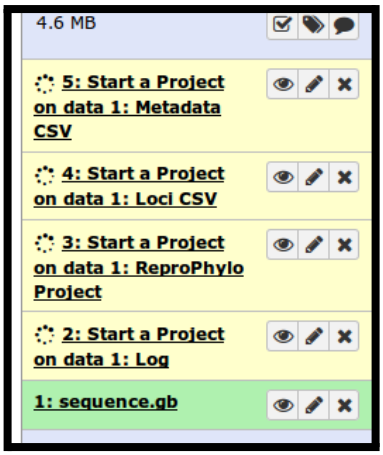
If you wish, you can click “Unnamed history” to edit this history's name.

8.4. Explore and choose the loci to analyse

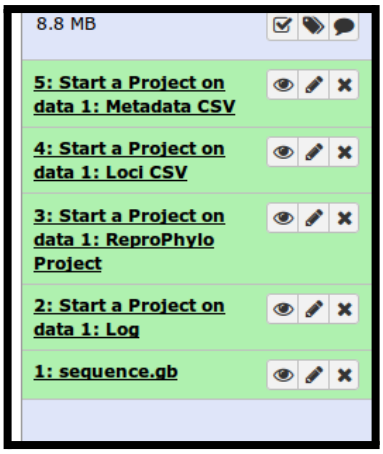
For this we will use the “Start a Project” tool. In the Tools sidebar, click ReproPhylo, then click Start a Project. The tool's gui will appear in the middle panel. From the drop-down menu “Initiate the Project with:”, choose “a GenBank file”. In the drop-down menu “genbank or embl file:” make sure that your uploaded genbank file is chosen. It should be, as it is the only item in the history. Finally click Execute.



Once the tool has started running, four new items will appear in the History panel:



They will first appear grey as the tool is initiating, then they will turn yellow as the tool runs, and green when it's done:



The new items are identified by their number in the queue, the name of the tool that generated them, the index of the input file and the output type. For example, item number two was generated with the tool “Start a Project” using item 1 (sequence.gb) as input, and is a log file of the run. We will look at this file in a moment.

Item number 3 is a pickle file of the Project we have generated. It has all the loci from the genbank file in its project.loci attribute and all the records from the genbank file in its project.records attribute (see Project for full description). In this use case we are going to ignore this output because we are going to customize the included loci instead of taking all of them.

Item number 4 is a CSV file (comma delimited) describing all the loci in the genbank file. This is the

file we are going to download and edit, in order to determine the loci we want to include.

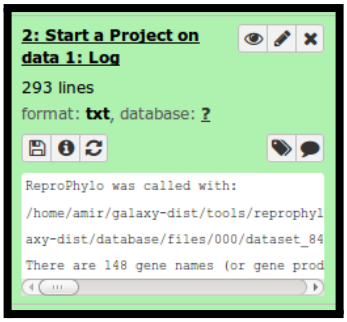
Item number 5 is another CSV file (tab delimited) describing the metadata of all the records in the genbank file. Since we are going to exclude some records by removing some loci, this file is also not relevant for us in this use case.

To download the loci CSV file, click item number 4, which will expand and look like this:

On the top right hand side of window shown above, from left to right, we can see the ‘view' button, the ‘edit' button and the ‘hide' button. The ‘view' button will present this output in the main panel, the ‘edit' button will allow you to edit things like the output's name and its format. The ‘hide' button will hide this output. It can be shown again using the menu that opens when you use the gear button on the top right hand side of the History panel.

On the middle left hand side of the window there are the ‘save', ‘info' and ‘rerun' buttons. Click the ‘save' button to save the loci csv file to your Downloads directory.

In order to choose our loci, we will also need to see the log of this tool. To do that, expand item number 2 from the history “Start a Project on data 1: Log” and click the ‘view' (eye) icon.



The log will appear in the middle panel and will look like this:

```
ReproPhylo was called with:
/home/amir/galaxy-dist/tools/reprophylo/base_reprophylo.p

There are 148 gene names (or gene product names) detected

------------------------------
Gene and count sorted by counts
------------------------------

304 instances of CDS,ND4
229 instances of CDS,cytb
48 instances of CDS,Cytb
47 instances of CDS,COIII
23 instances of rRNA,28S ribosomal RNA
21 instances of CDS,alpha enolase
21 instances of rRNA,5.8S ribosomal RNA
15 instances of CDS,C-mos
15 instances of CDS,NADH dehydrogenase subunit 4
13 instances of CDS,NT3
11 instances of CDS,c-mos
9 instances of CDS,cytochrome B
9 instances of rRNA,12S ribosomal RNA
7 instances of CDS,cytochrome b
6 instances of CDS,BDNF
6 instances of CDS,ND1
6 instances of CDS.ND2                                    .
```

…

The top line is the command-line that was used to call the command-line program that powers this Galaxy tool (it is truncated in this figure). First we are informed that 148 loci were found in this genbank file, loci being unique values found in the gene and product qualifiers of the genbank records. Than, all the loci are listed in descending order based on their count in the file. A locus that appears in four different names in the genbank file will also appear four times in this list. We are going to exclude most of the loci and keep only the abundant ones. To do that, open the loci csv file you have saved to Downloads (item number 4) in any text editor (not in a word processor). Keep only the lines that appear in the image below and delete the rest. Note that for some loci, such as cyt-b, synonyms were identified and placed on one line. In other cases, such as with NT3, synonyms were not identified. To indicate that NT3 and NT-3 are synonyms, add ",9" at the end of both lines- as is shown below. By adding any shared integer at the end of two or more lines, we can indicate that these names belong to the same gene. It can be any integer at all. Go here for more on the structure of the loci CSV file.

edited_loci.csv ×

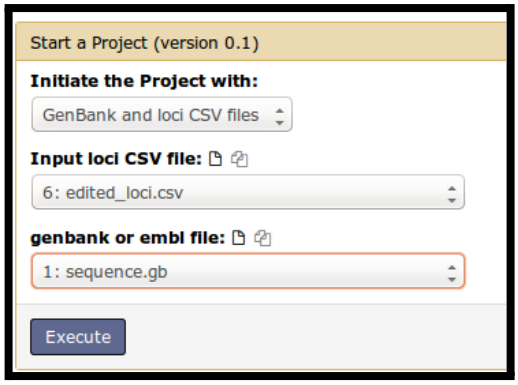
```
1  dna,rRNA,28s,28S ribosomal RNA
2  dna,rRNA,5.8S_ribosomal_RNA,5.8S ribosomal RNA
3  dna,CDS,C_mos,C-mos,c-mos,C-MOS
4  dna,CDS,MT-CO3,COIII,cox III,COX3
5  dna,CDS,MT-CYB,cytb,CYTB,Cytb,cytochrome B,cytochrome b
6  dna,CDS,MT-ND4,NADH dehydrogenase subunit 4,nadh4,ND4
7  dna,CDS,NT-3,NT-3,9
8  dna,CDS,NT3,NT3,9
9  dna,rRNA,rrnS,12S ribosomal RNA,12S rRNA
10 dna,CDS,alpha_enolase,alpha enolase
11
```

When you are done editing, save the file. Use the name 'edited_loci.csv' to match the name used here. Upload the file in the same way we uploaded the genbank file. Finally we can start a project that will include only the loci we are interested in.

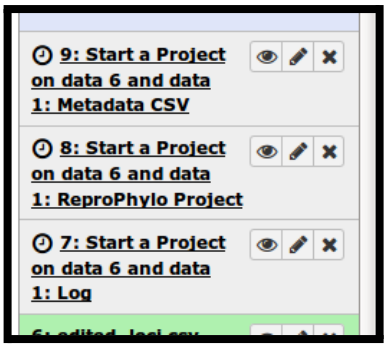
8.5. Start a Project with the selected loci and the relevant records from the genbank files

Go to the "Start a Project" tool again. This time, choose the "GenBank and loci CSV files" option from the "Initiate the Project with:" drop down menu. Now indicate the loci CSV file you have just uploaded

(should be number 6) and the genbank file and execute (It is also possible to start a Project using a loci CSV file only, and to then add sequence from other file formats using another tool. This is not covered here).

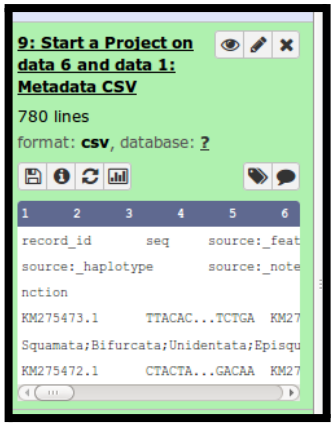


Three new items (7-9) will appear in the history, and will include a new log file (7), a new Project file (8) and a new metadata CSV file (9). The Project and metadata will now only include records that belong to loci we have indicated in our loci CSV file. Note that since we have provided our own loci CSV file, one is not generated and is not part of the output.



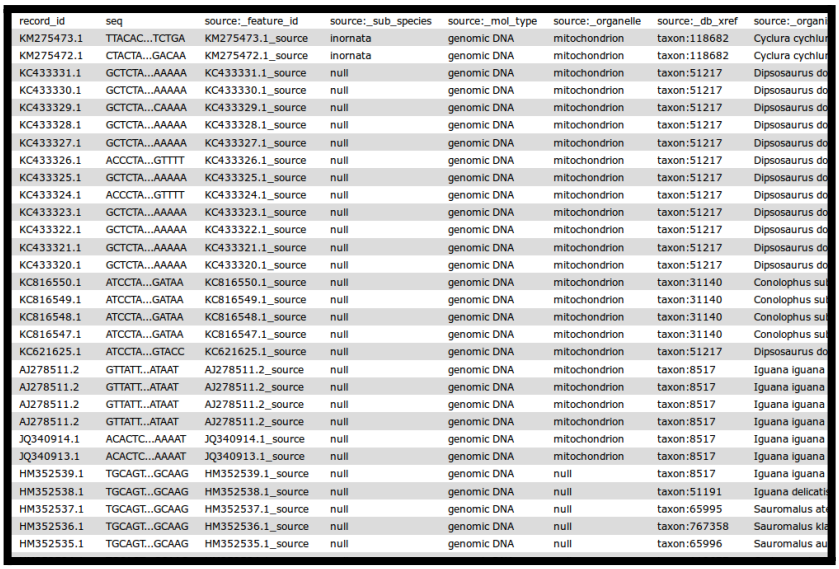
8.6. Explore the available metadata from the genbank file.

The next step in our use case is to edit the metadata. Specifically, we are going to add a genus qualifier that will allow us to color the clades according to genus. The first stage is to expand the new metadata CSV item in the history panel (item 9).



The history panel provides a peek to the content of the file, but we can also view the full table in the

main panel by clicking the eye icon. This is a section of the display that should appear:



record_id	seq	source:_feature_id	source:_sub_species	source:_mol_type	source:_organelle	source:_db_xref	source:_organi
KM275473.1	TTACAC...TCTGA	KM275473.1_source	inornata	genomic DNA	mitochondrion	taxon:118682	Cyclura cychlu
KM275472.1	CTACTA...GACAA	KM275472.1_source	inornata	genomic DNA	mitochondrion	taxon:118682	Cyclura cychlu
KC433331.1	GCTCTA...AAAAA	KC433331.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433330.1	GCTCTA...AAAAA	KC433330.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433329.1	GCTCTA...CAAAA	KC433329.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433328.1	GCTCTA...AAAAA	KC433328.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433327.1	GCTCTA...AAAAA	KC433327.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433326.1	ACCCTA...GTTTT	KC433326.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433325.1	GCTCTA...AAAAA	KC433325.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433324.1	ACCCTA...GTTTT	KC433324.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433323.1	GCTCTA...AAAAA	KC433323.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433322.1	GCTCTA...AAAAA	KC433322.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433321.1	GCTCTA...AAAAA	KC433321.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC433320.1	GCTCTA...AAAAA	KC433320.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
KC816550.1	ATCCTA...GATAA	KC816550.1_source	null	genomic DNA	mitochondrion	taxon:31140	Conolophus sul
KC816549.1	ATCCTA...GATAA	KC816549.1_source	null	genomic DNA	mitochondrion	taxon:31140	Conolophus sul
KC816548.1	ATCCTA...GATAA	KC816548.1_source	null	genomic DNA	mitochondrion	taxon:31140	Conolophus sul
KC816547.1	ATCCTA...GATAA	KC816547.1_source	null	genomic DNA	mitochondrion	taxon:31140	Conolophus sul
KC621625.1	ATCCTA...GTACC	KC621625.1_source	null	genomic DNA	mitochondrion	taxon:51217	Dipsosaurus do
AJ278511.2	GTTATT...ATAAT	AJ278511.2_source	null	genomic DNA	mitochondrion	taxon:8517	Iguana iguana
AJ278511.2	GTTATT...ATAAT	AJ278511.2_source	null	genomic DNA	mitochondrion	taxon:8517	Iguana iguana
AJ278511.2	GTTATT...ATAAT	AJ278511.2_source	null	genomic DNA	mitochondrion	taxon:8517	Iguana iguana
AJ278511.2	GTTATT...ATAAT	AJ278511.2_source	null	genomic DNA	mitochondrion	taxon:8517	Iguana iguana
JQ340914.1	ACACTC...AAAAT	JQ340914.1_source	null	genomic DNA	mitochondrion	taxon:8517	Iguana iguana
JQ340913.1	ACACTC...AAAAT	JQ340913.1_source	null	genomic DNA	mitochondrion	taxon:8517	Iguana iguana
HM352539.1	TGCAGT...GCAAG	HM352539.1_source	null	genomic DNA	null	taxon:8517	Iguana iguana
HM352538.1	TGCAGT...GCAAG	HM352538.1_source	null	genomic DNA	null	taxon:51191	Iguana delicati
HM352537.1	TGCAGT...GCAAG	HM352537.1_source	null	genomic DNA	null	taxon:65995	Sauromalus at
HM352536.1	TGCAGT...GCAAG	HM352536.1_source	null	genomic DNA	null	taxon:767358	Sauromalus kl
HM352535.1	TGCAGT...GCAAG	HM352535.1_source	null	genomic DNA	null	taxon:65996	Sauromalus au

This view allows us to easily explore the metadata and consider the changes we want to make. In order to introduce these changes we will have to download this file, which is a tab delimited text file, and open it in a spreadsheet program such as Excel or Libre Office. Click the diskette icon and look for the file in the Downloads directory on your machine.

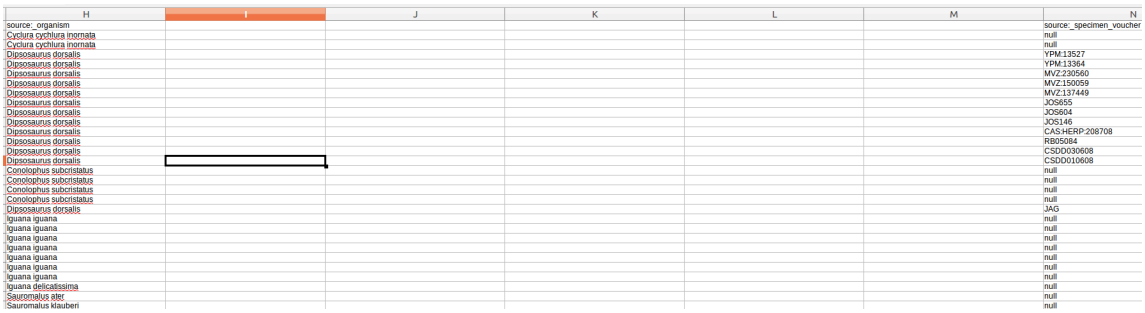
8.7. Add additional information of our own

This section has nothing much to do with Galaxy or ReproPhylo. It is one possible example of how one might edit the metadata using an external program. In this case we will make a ‘genus' column out of the “source:_organism” column:

1. Spot the source:_organism column in the spreadsheet.

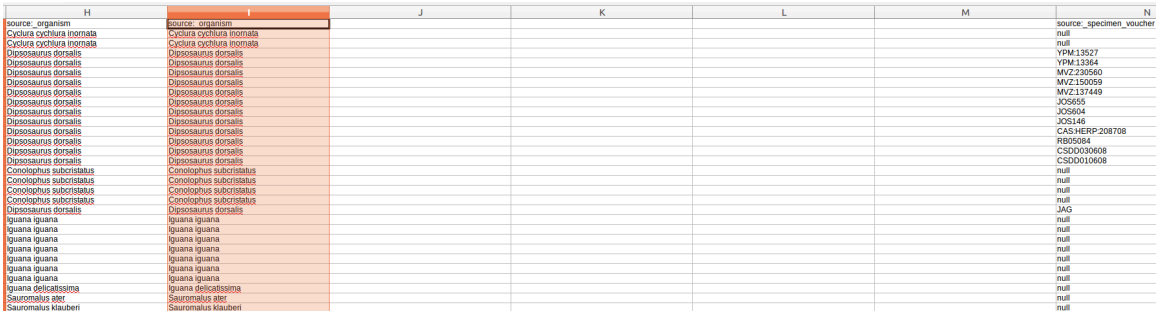
	H	I	
f	source:_organism	source:_specimen_voucher	sou
	Cyclura cychlura inornata	null	null
	Cyclura cychlura inornata	null	null
	Dipsosaurus dorsalis	YPM:13527	35.
	Dipsosaurus dorsalis	YPM:13364	34.
	Dipsosaurus dorsalis	MVZ:230560	33.
	Dipsosaurus dorsalis	MVZ:150059	34.
	Dipsosaurus dorsalis	MVZ:137449	35.
	Dipsosaurus dorsalis	JOS655	33.
	Dipsosaurus dorsalis	JOS604	33.
	Dipsosaurus dorsalis	JOS146	33.
	Dipsosaurus dorsalis	CAS:HERP:208708	33.
	Dipsosaurus dorsalis	RB05084	33.
	Dipsosaurus dorsalis	CSDD030608	34.
	Dipsosaurus dorsalis	CSDD010608	36.
	Conolophus subcristatus	null	null
	Conolophus subcristatus	null	null
	Conolophus subcristatus	null	null
	Conolophus subcristatus	null	null
	Dipsosaurus dorsalis	JAG	null
	Iguana iguana	null	null
	Iguana iguana	null	null
	Iguana iguana	null	null
	Iguana iguana	null	null
	Iguana iguana	null	null
	Iguana iguana	null	null
	Iguana iguana	null	null
	Iguana delicatissima	null	null
	Sauromalus ater	null	null
	Sauromalus klauberi	null	null
	Sauromalus australis	null	null
	Cyclura carinata	null	null
	Cyclura ricordi	null	null
	Conolophus subcristatus	null	null
	Conolophus pallidus	null	null
	Iguana iguana	null	null
	Iguana iguana	null	null

2. Add five columns to the right of the source:_organism column:



H	I	J	K	L	M	N
source:_organism						source:_specimen_voucher
Cyclura cychlura inornata						null
Cyclura cychlura inornata						null
Dipsosaurus dorsalis						YPM:13527
Dipsosaurus dorsalis						YPM:13364
Dipsosaurus dorsalis						MVZ:230560
Dipsosaurus dorsalis						MVZ:150059
Dipsosaurus dorsalis						MVZ:137449
Dipsosaurus dorsalis						JOS655
Dipsosaurus dorsalis						JOS604
Dipsosaurus dorsalis						JOS146
Dipsosaurus dorsalis						CAS:HERP:208708
Dipsosaurus dorsalis						RB05084
Dipsosaurus dorsalis						CSDD030608
Dipsosaurus dorsalis						CSDD010608
Conolophus subcristatus						null
Conolophus subcristatus						null
Conolophus subcristatus						null
Conolophus subcristatus						null
Dipsosaurus dorsalis						JAG
Iguana iguana						null
Iguana iguana						null
Iguana iguana						null
Iguana iguana						null
Iguana iguana						null
Iguana iguana						null
Iguana iguana						null
Iguana delicatissima						null
Sauromalus ater						null
Sauromalus klauberi						null

3. Copy and paste source:_organism to an empty column (make a duplicate):



H	I	J	K	L	M	N
source:_organism	source:_organism					source:_specimen_voucher
Cyclura cychlura inornata	Cyclura cychlura inornata					null
Cyclura cychlura inornata	Cyclura cychlura inornata					null
Dipsosaurus dorsalis	Dipsosaurus dorsalis					YPM:13527
Dipsosaurus dorsalis	Dipsosaurus dorsalis					YPM:13364
Dipsosaurus dorsalis	Dipsosaurus dorsalis					MVZ:230560
Dipsosaurus dorsalis	Dipsosaurus dorsalis					MVZ:150059
Dipsosaurus dorsalis	Dipsosaurus dorsalis					MVZ:137449
Dipsosaurus dorsalis	Dipsosaurus dorsalis					JOS655
Dipsosaurus dorsalis	Dipsosaurus dorsalis					JOS604
Dipsosaurus dorsalis	Dipsosaurus dorsalis					JOS146
Dipsosaurus dorsalis	Dipsosaurus dorsalis					CAS:HERP:208708
Dipsosaurus dorsalis	Dipsosaurus dorsalis					RB05084
Dipsosaurus dorsalis	Dipsosaurus dorsalis					CSDD030608
Dipsosaurus dorsalis	Dipsosaurus dorsalis					CSDD010608
Conolophus subcristatus	Conolophus subcristatus					null
Conolophus subcristatus	Conolophus subcristatus					null
Conolophus subcristatus	Conolophus subcristatus					null
Conolophus subcristatus	Conolophus subcristatus					null
Dipsosaurus dorsalis	Dipsosaurus dorsalis					JAG
Iguana iguana	Iguana iguana					null
Iguana iguana	Iguana iguana					null
Iguana iguana	Iguana iguana					null
Iguana iguana	Iguana iguana					null
Iguana iguana	Iguana iguana					null
Iguana iguana	Iguana iguana					null
Iguana iguana	Iguana iguana					null
Iguana delicatissima	Iguana delicatissima					null
Sauromalus ater	Sauromalus ater					null
Sauromalus klauberi	Sauromalus klauberi					null

4. Select the new source:_organism column and use the menu “data/text to columns” to split the values to columns on whitespaces.

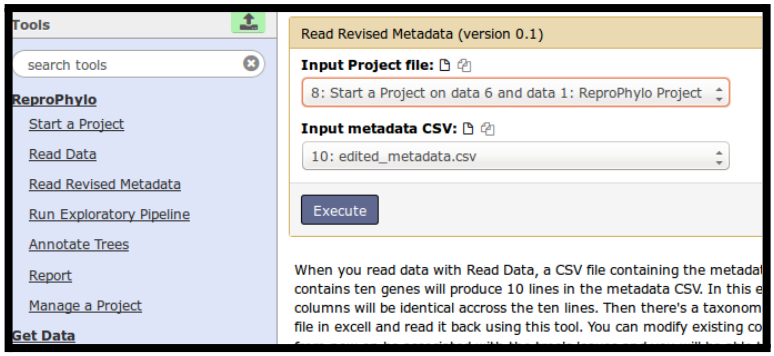
H	I	J	K	L	M	N
source:_organism	source:_organism					source:_specimen_voucher
Cyclura cychlura inornata	Cyclura	cychlura	inornata			null
Cyclura cychlura inornata	Cyclura	cychlura	inornata			null
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				YPM:13527
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				YPM:13364
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				MVZ:230560
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				MVZ:150059
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				MVZ:137449
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				JOS655
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				JOS604
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				JOS146
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				CAS:HERP:208708
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				RB05084
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				CSDD030608
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				CSDD010608
Conolophus subcristatus	Conolophus	subcristatus				null
Conolophus subcristatus	Conolophus	subcristatus				null
Conolophus subcristatus	Conolophus	subcristatus				null
Conolophus subcristatus	Conolophus	subcristatus				null
Dipsosaurus dorsalis	Dipsosaurus	dorsalis				JAG
Iguana iguana	Iguana	iguana				null
Iguana iguana	Iguana	iguana				null
Iguana iguana	Iguana	iguana				null
Iguana iguana	Iguana	iguana				null
Iguana iguana	Iguana	iguana				null
Iguana iguana	Iguana	iguana				null
Iguana iguana	Iguana	iguana				null
Iguana delicatissima	Iguana	delicatissima				null
Sauromalus ater	Sauromalus	ater				null
Sauromalus klauberi	Sauromalus	klauberi				null

5. Rename the duplicate source:_organism column as “genus”. Note that now it only contains the first word of source:_organism, which is the genus. Then delete the other new columns, which contain the other parts of the source:_organism split values.

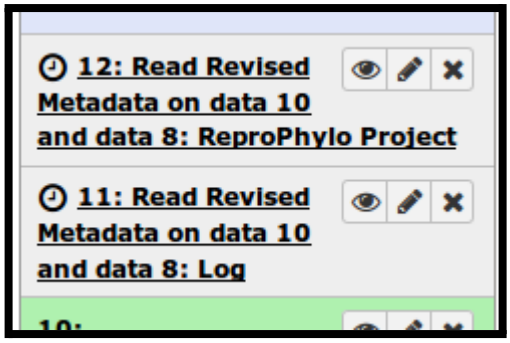
source:_organism	genus	source:_specimen_voucher	so
Cyclura cychlura inornata	Cyclura	null	nu
Cyclura cychlura inornata	Cyclura	null	nu
Dipsosaurus dorsalis	Dipsosaurus	YPM:13527	35
Dipsosaurus dorsalis	Dipsosaurus	YPM:13364	34
Dipsosaurus dorsalis	Dipsosaurus	MVZ:230560	33
Dipsosaurus dorsalis	Dipsosaurus	MVZ:150059	34
Dipsosaurus dorsalis	Dipsosaurus	MVZ:137449	35
Dipsosaurus dorsalis	Dipsosaurus	JOS655	33
Dipsosaurus dorsalis	Dipsosaurus	JOS604	33
Dipsosaurus dorsalis	Dipsosaurus	JOS146	33
Dipsosaurus dorsalis	Dipsosaurus	CAS:HERP:208708	33
Dipsosaurus dorsalis	Dipsosaurus	RB05084	33
Dipsosaurus dorsalis	Dipsosaurus	CSDD030608	34
Dipsosaurus dorsalis	Dipsosaurus	CSDD010608	36
Conolophus subcristatus	Conolophus	null	nu
Conolophus subcristatus	Conolophus	null	nu
Conolophus subcristatus	Conolophus	null	nu
Conolophus subcristatus	Conolophus	null	nu
Dipsosaurus dorsalis	Dipsosaurus	JAG	

6. Save your edited file. Use the name “edited_metadata.csv” to match the name used here. Make sure to save it as a tab delimited CSV file. Now you can upload your edited file to galaxy. Once it is done, you should expect to see it as a new item (most likely number 10) in your history panel.

Revised metadata files are read back using the tool “Read Revised Metadata”. The tool takes a Project file (use the most recent, number 8, which includes only our loci of choice), and the metadata CSV file. Click “Execute” to run.

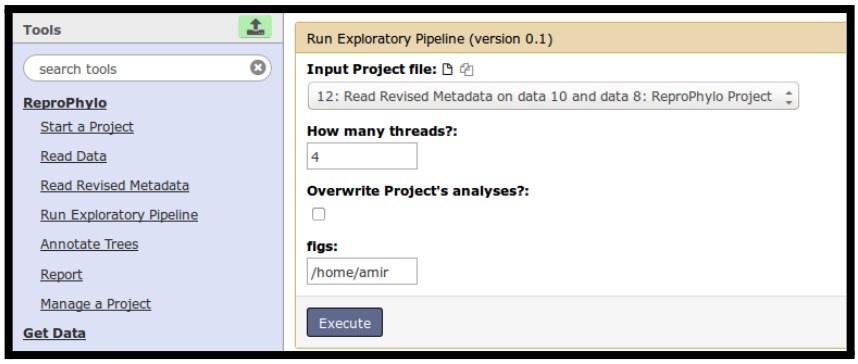


This tool should add a log file (11) and a Project file (12) which will now have the corrected metadata in it.



8.8. Run a fixed phylogenetic pipeline

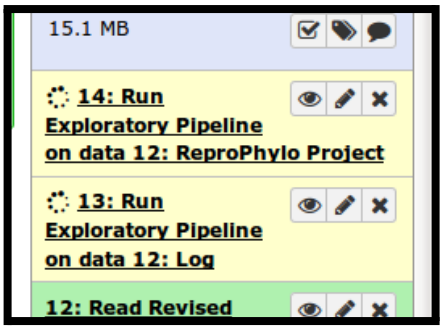
Unlike the python module, the Galaxy tools do not allow us to configure the phylogenetic analysis at the moment. This is obviously a drawback which is the focus of the current development. However, as a first run that allows you to evaluate your data, or the data from GenBank, it is still very useful. To perform this run, click the “Run Exploratory Pipeline” tool.



The tool takes a Project file as input. Specify the most recent, which has the edited metadata. According to your machine specs, specify the number of threads to run (if in doubt 4 is a good choice). You can also select to overwrite existing analyses. This can be useful when you are using a Project which already contains trees, by adding a locus to it with some new data. In this case you may want to run the analysis again for all the loci, by ticking this box, or just for the new locus, by leaving the box unticked. The utility of this option will increase, once it is possible to configure the analysis. Bear in mind that it is possible to analyse the same locus twice, if we would like for example to try several different taxon samplings, by having the same line twice in the loci CSV file, but with a different third value, which is the locus name, in each of the lines.

The last bit of information needed is a path to write .png graphic files showing the trees. The .png files will be written in the path you specify here, and will not be tracked by Galaxy. There are several reasons for this. First, these files are not needed for reproducibility. You can print them at any time using your Project file. Therefore, there is no need to clutter the history with them. Second, if you archive the analysis, as we will when we are done here, the archive will contain these figures and it *will* be a part of the history.

This tool will add two output items to your history. A Project file, which now also contains alignments, trimmed alignments and trees, and a log file:



The log file this tool produces can be viewed in the main panel by clicking the eye icon, and it has the following sections:

The command line used to call ReproPhylo (truncated in the figure) followed by a table showing the loci in the Project, their record content, and some sequence statistics:

```
ReproPhylo was called with:
/home/amir/galaxy-dist/tools/reprophylo/base_reprophylo.py /home/amir/galaxy-dist/database/fil


---------------------------------------------
The Project Now Contains The Following Loci:
---------------------------------------------
Locus                       Records   Sequence length (max min mean)
5.8S_ribosomal_RNA          21        72       72       72.0
MT-ND4                      320       1381     12       551.3
MT-CYB                      294       1140     180      982.8
C_mos                       27        495      227      377.1
rrnS                        10        952      29       385.0
NT-3                        14        510      482      491.4
MT-CO3                      49        786      643      650.7
alpha_enolase               21        88       88       88.0
28s                         23        699      126      169.8
------------------
```

Next, the log shows all the MAFFT command lines that were executed and also information about dropped loci or taxa. The alignment strategy is fixed to default MAFFT settings without codon alignment.

```
------------------
Sequence Alignment
------------------
mafft 197351417472577.26_5.8S_ribosomal_RNA.fasta
mafft 197351417472577.26_MT-ND4.fasta
mafft 197351417472577.26_MT-CYB.fasta
mafft 197351417472577.26_C_mos.fasta
mafft 197351417472577.26_rrnS.fasta
mafft 197351417472577.26_NT-3.fasta
mafft 197351417472577.26_MT-CO3.fasta
mafft 197351417472577.26_alpha_enolase.fasta
mafft 197351417472577.26_28s.fasta
Alignment 5.8S_ribosomal_RNA@MafftDefaults has less than 4 unique sequences and will be dropped
```

The next part shows the TrimAl command lines, which are fixed to the gappyput approach. It also informs us of sequences that were all gap after trimming and were therefore removed.

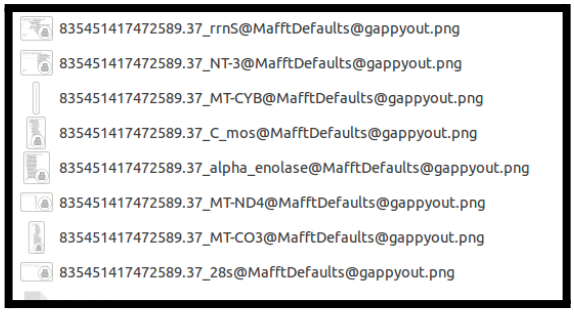
```
------------------
Trimming Alignment
------------------
/home/amir/galaxy-dist/tools/reprophylo/programs/trimal -in 993301417472586.53_MT-CO3@MafftDefaults.fasta -gappyout
/home/amir/galaxy-dist/tools/reprophylo/programs/trimal -in 993301417472586.53_NT-3@MafftDefaults.fasta -gappyout
/home/amir/galaxy-dist/tools/reprophylo/programs/trimal -in 993301417472586.53_MT-ND4@MafftDefaults.fasta -gappyout
/home/amir/galaxy-dist/tools/reprophylo/programs/trimal -in 993301417472586.53_alpha_enolase@MafftDefaults.fasta -gappyout
/home/amir/galaxy-dist/tools/reprophylo/programs/trimal -in 993301417472586.53_C_mos@MafftDefaults.fasta -gappyout
/home/amir/galaxy-dist/tools/reprophylo/programs/trimal -in 993301417472586.53_rrnS@MafftDefaults.fasta -gappyout
/home/amir/galaxy-dist/tools/reprophylo/programs/trimal -in 993301417472586.53_28s@MafftDefaults.fasta -gappyout
/home/amir/galaxy-dist/tools/reprophylo/programs/trimal -in 993301417472586.53_MT-CYB@MafftDefaults.fasta -gappyout
Alignment MT-ND4@MafftDefaults@gappyout has undetermined sequences which will be dropped: ['U66233.1_f0', 'U66232.1_f0', 'U66235.
------------------
```

The log ends with RAxML command lines, html formated links to the figures, and references for all the software used. The tree reconstruction strategy is fixed to the default RAxML rapid hill climbing

algorithm for a single ML search, with relBootstrap branch supports.

```
------------------
Tree Reconstructios
------------------
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f D -m GTRGAMMA -n 835451417472589.37_rrnS@MafftDefaults@gappyout0 -p 14 -s 835451417472589.37_rrnS@M
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f b -m GTRGAMMA -n 835451417472589.37_rrnS@MafftDefaults@gappyout1 -p 354 -s 835451417472589.37_rrnS@
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f D -m GTRGAMMA -n 835451417472589.37_C_mos@MafftDefaults@gappyout0 -p 657 -s 835451417472589.37_C_mo
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f b -m GTRGAMMA -n 835451417472589.37_C_mos@MafftDefaults@gappyout1 -p 278 -s 835451417472589.37_C_mo
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f D -m GTRGAMMA -n 835451417472589.37_alpha_enolase@MafftDefaults@gappyout0 -p 632 -s 835451417472589
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f b -m GTRGAMMA -n 835451417472589.37_alpha_enolase@MafftDefaults@gappyout1 -p 157 -s 835451417472589
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f D -m GTRGAMMA -n 835451417472589.37_MT-CYB@MafftDefaults@gappyout0 -p 32 -s 835451417472589.37_MT-C
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f b -m GTRGAMMA -n 835451417472589.37_MT-CYB@MafftDefaults@gappyout1 -p 675 -s 835451417472589.37_MT-
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f D -m GTRGAMMA -n 835451417472589.37_MT-ND4@MafftDefaults@gappyout0 -p 271 -s 835451417472589.37_MT-
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f b -m GTRGAMMA -n 835451417472589.37_MT-ND4@MafftDefaults@gappyout1 -p 919 -s 835451417472589.37_MT-
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f D -m GTRGAMMA -n 835451417472589.37_MT-CO3@MafftDefaults@gappyout0 -p 986 -s 835451417472589.37_MT-
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f b -m GTRGAMMA -n 835451417472589.37_MT-CO3@MafftDefaults@gappyout1 -p 867 -s 835451417472589.37_MT-
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f D -m GTRGAMMA -n 835451417472589.37_NT-3@MafftDefaults@gappyout0 -p 789 -s 835451417472589.37_NT-3@
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f b -m GTRGAMMA -n 835451417472589.37_NT-3@MafftDefaults@gappyout1 -p 976 -s 835451417472589.37_NT-3@
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f D -m GTRGAMMA -n 835451417472589.37_28s@MafftDefaults@gappyout0 -p 581 -s 835451417472589.37_28s@Ma
/home/amir/galaxy-dist/tools/reprophylo/programs/raxmlHPC-PTHREADS-SSE3 -f b -m GTRGAMMA -n 835451417472589.37_28s@MafftDefaults@gappyout1 -p 696 -s 835451417472589.37_28s@Ma
--------------
Tree Annotation
--------------
<html>
rooting tree 28s@MafftDefaults@gappyout@fD_fb at midpoint
<A href=/home/amir/835451417472589.37_28s@MafftDefaults@gappyout.png>835451417472589.37_28s@MafftDefaults@gappyout</A><BR>
rooting tree MT-CO3@MafftDefaults@gappyout@fD_fb at midpoint
<A href=/home/amir/835451417472589.37_MT-CO3@MafftDefaults@gappyout.png>835451417472589.37_MT-CO3@MafftDefaults@gappyout</A><BR>
rooting tree MT-ND4@MafftDefaults@gappyout@fD_fb at midpoint
<A href=/home/amir/835451417472589.37_MT-ND4@MafftDefaults@gappyout.png>835451417472589.37_MT-ND4@MafftDefaults@gappyout</A><BR>
rooting tree MT-CYB@MafftDefaults@gappyout@fD_fb at midpoint
<A href=/home/amir/835451417472589.37_MT-CYB@MafftDefaults@gappyout.png>835451417472589.37_MT-CYB@MafftDefaults@gappyout</A><BR>
rooting tree alpha_enolase@MafftDefaults@gappyout@fD_fb at midpoint
<A href=/home/amir/835451417472589.37_alpha_enolase@MafftDefaults@gappyout.png>835451417472589.37_alpha_enolase@MafftDefaults@gappyout</A><BR>
rooting tree rrnS@MafftDefaults@gappyout@fD_fb at midpoint
<A href=/home/amir/835451417472589.37_rrnS@MafftDefaults@gappyout.png>835451417472589.37_rrnS@MafftDefaults@gappyout</A><BR>
rooting tree NT-3@MafftDefaults@gappyout@fD_fb at midpoint
<A href=/home/amir/835451417472589.37_NT-3@MafftDefaults@gappyout.png>835451417472589.37_NT-3@MafftDefaults@gappyout</A><BR>
rooting tree C_mos@MafftDefaults@gappyout@fD_fb at midpoint
<A href=/home/amir/835451417472589.37_C_mos@MafftDefaults@gappyout.png>835451417472589.37_C_mos@MafftDefaults@gappyout</A><BR>
</html>
/home/amir
------------------
Program References
------------------
--------------------------------
Exploratory Pipeline References
```

If you check out the path you specified for the png files, you will see a png file for each one of the trees:

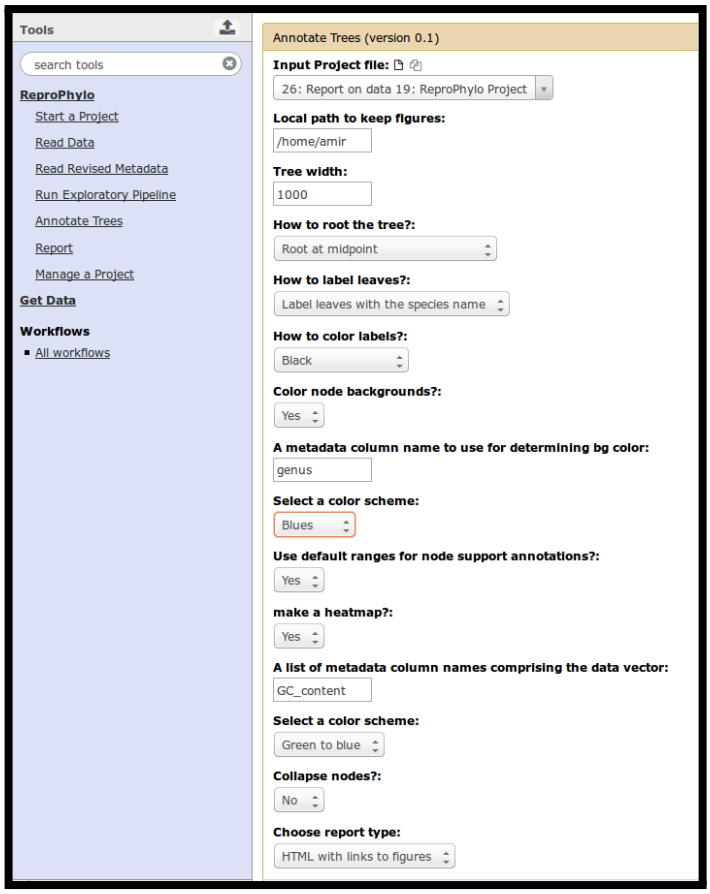


The file names are composed of a unique ReproPhylo process ID (mostly an internal thing, but can be useful to assert which process created which tree by checking the log), the locus name, the AlnConf name and the TrimalConf name. The RaxmlConf name is not included, but if you run several tree strategies on the same trimmed alignment (currently only possible using the python module directly) it will be noted by a different process ID. You will be able to identify the process by checking the log or the archive we will create later.

8.9. Annotate the resulting trees using the metadata

The “Run Exploratory Pipeline” tool provides very basic and somewhat uninformative tree figures, because they have no meaningful annotation. The “Annotate Trees” tool allows you to add all those things that will make the trees easier to interpret. The first value taken in the tool's GUI is the Project file produced by the “Run Exploratory Pipeline” tool. The second is a path to which you wish to write your annotated trees png files. Next comes “Tree width” which controls the width of the trees compared to the figure borders.

By default, rooting is done at midpoint. If you wish to change this, you can choose “Specify outgroup using metadata” from the “How to root” dropdown menu. This will add two new boxes to the form, the first of which will ask you to specify a column name in your metadata. If you wish to use a source qualifier, the format will be source_qualifierName (no colon). In the second new box, put the value specifying an outgroup OTU. In our case we might have put ‘genus' in the first of these boxes and some outgroup genus name in the second of these boxes. For sets of loci that don't have the this genus, the rooting would stay at midpoint.



After sorting out rooting, we need to decide how to label the leaves. By default they are labeled with the species name from the organism qualifier of the source feature. If you wish to change this, choose “specify labels using the metadata” from the dropdown menu. This will open a new box, in which you'll need to provide a whitespace delimited list of all the qualifiers you want to include in the label (eg “source_organism feature_id gene”).

You can also decide whether to leave the labels black or to color them based on some metadata. If you wish to color the labels, choose “According to rules” in the “How to color labels” dropdown menu. As before, this will open a box in which to specify a column name, and a dropdown menu in which to choose a color scheme.

In this use case, we will color clade backgrounds according to genera. Choose yes for “Color node backgrounds?”. In the box that appears, labeled “A metadata column name to use for determining bg color” type in “genus”. In the dropdown menu “select color scheme” choose “blues”.

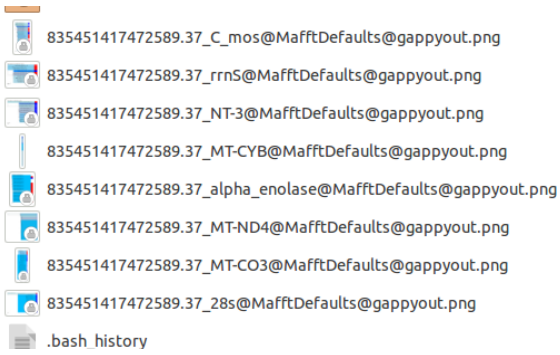
We are going to leave “Use default ranges for node support annotation?” at Yes. If you wish to change it to No, you'll be asked to provide support ranges and corresponding colors. You will need to type in a string similar to this one: “black 100 90 gray 90 80 silver 80 50”. You can see color names here.

Another feature we'll take advantage of now is the heatmap. Our's is going to be a simple one with just one column, but the principle is the same for any size. Change “make heatmap?” to Yes. In the box that appears, type in “GC_content”, which is one of the column names in the metadata spreadsheet. Finally, we can choose a color scheme. Here we'll leave it on Red to blue.

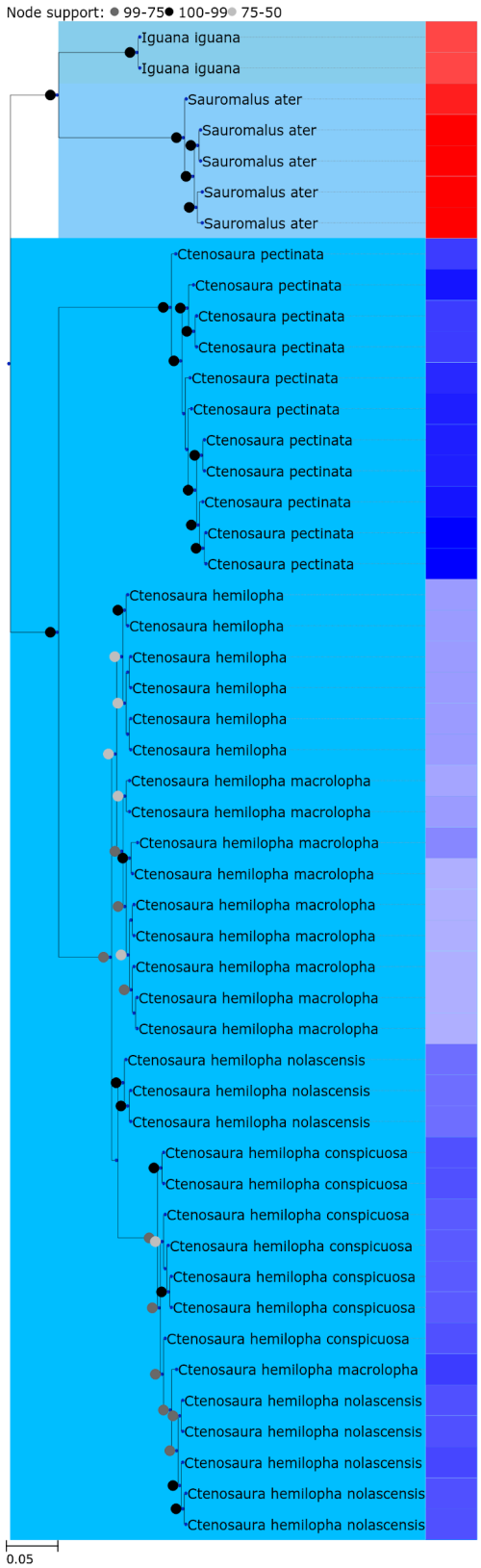
Last option in the form has to do with node collapsing based on node support. You can specify a value, under which nodes will be multifurcated in the figure. We are going to leave as is.

The tool allows you to either make only the figures, with an html file to point at them, or to make a full archive. We are going to do the first, and leave the second for later.

Execute the action. Once it is done, your tree files in the path you specified should look like this:

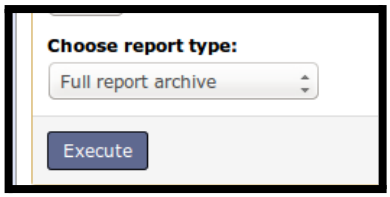


The cox 3 tree (next page) seems to be a nice example of how even this quick analysis provides insight on the data. Each genus in the tree is colored with a different shade of blue, and GC values are color coded on the right hand side. Since the extreme colors (red and blue in this case) are given to the actual lowest and highest values encountered, (which are 42.9% for the species *Ctenosaura pectinata* and 48.99% for the species *Sauromalus ater* in this case) our example is on one hand a bit misleading because it looks like the range of GC values is higher than it really is. On the other hand, using the full range of the color scale allows us to visualize the fit between the GC values and the phylogeny, and to highlight the connection between the tree's midpoint (as it is midpoint rooted) and the GC content of the leaves. This kind of information is valuable while building a phylogenetic tree, and we make it jump out at us with almost no effort at all.

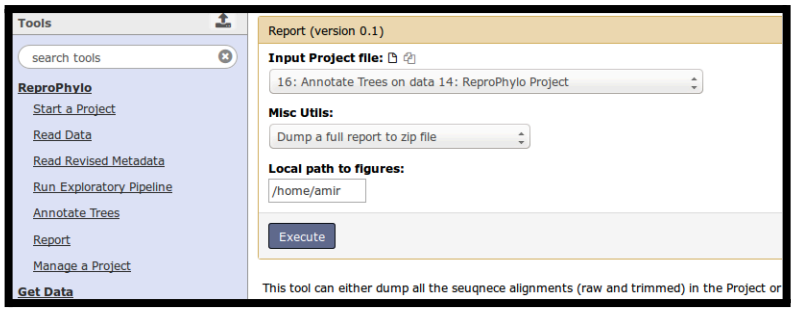

Node support: 99-75 100-99 75-50
Iguana iguana
Iguana iguana
Sauromalus ater
Sauromalus ater
Sauromalus ater
Sauromalus ater
Sauromalus ater
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura hemilopha
Ctenosaura hemilopha
Ctenosaura hemilopha
Ctenosaura hemilopha
Ctenosaura hemilopha
Ctenosaura hemilopha
Ctenosaura hemilopha macrolopha
Ctenosaura hemilopha macrolopha
Ctenosaura hemilopha macrolopha
Ctenosaura hemilopha macrolopha
Ctenosaura hemilopha macrolopha
Ctenosaura hemilopha macrolopha
Ctenosaura hemilopha macrolopha
Ctenosaura hemilopha macrolopha
Ctenosaura hemilopha nolascensis
Ctenosaura hemilopha nolascensis
Ctenosaura hemilopha nolascensis
Ctenosaura hemilopha conspicuosa
Ctenosaura hemilopha conspicuosa
Ctenosaura hemilopha conspicuosa
Ctenosaura hemilopha conspicuosa
Ctenosaura hemilopha conspicuosa
Ctenosaura hemilopha conspicuosa
Ctenosaura hemilopha conspicuosa
Ctenosaura hemilopha macrolopha
Ctenosaura hemilopha nolascensis
Ctenosaura hemilopha nolascensis
Ctenosaura hemilopha nolascensis
Ctenosaura hemilopha nolascensis
Ctenosaura hemilopha nolascensis
0.05

8.10. Archive the results

An archive of our analysis can be produced by the annotation tool, by selecting an archive output at the very bottom of the form.



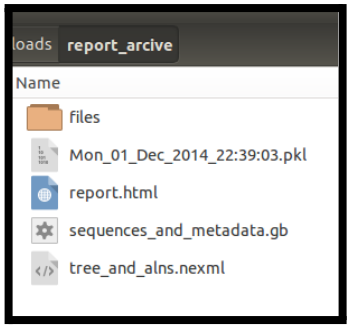
However, typically we would want to archive only after having a look at the figure. We might also want to write files that are not part of the default archive. For these things we can use the “Report” tool. The input for this tool is our latest Project file, then we get to choose between producing a zip file and between producing a GenBank file for the records and alignment files with our choice of format. Here we'll make a zip file. The last input is the path that contains our figure files, the same one we specified in the annotation tool.



Running the tool with the zip file option will produce three outputs: A log file (18) a Project file (19) and a report zip file (20).

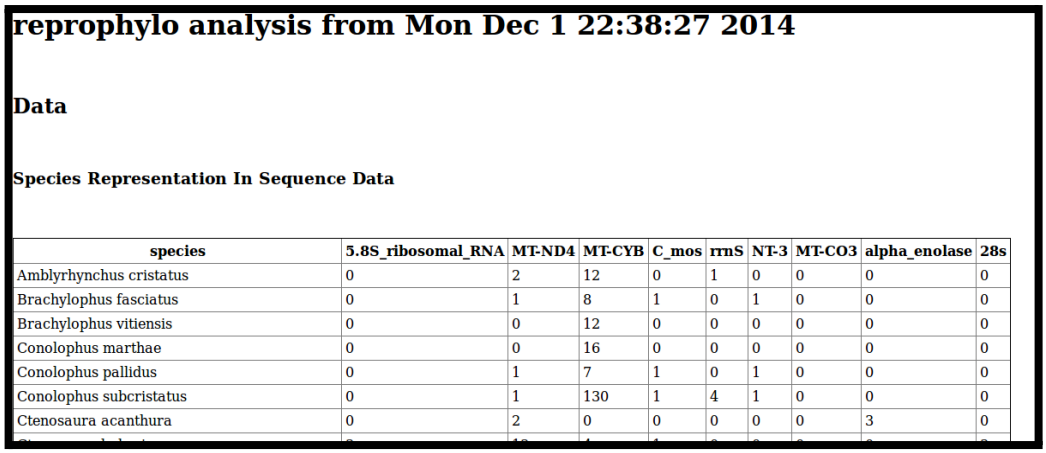
Expand number 20 and click the diskette icon to download the zip file once the analysis is ready.

Extract or mount the zip file and have a look on the files it contains:



Now open the report.html in your web browser. It should have the following sections:

The report begins with a header specifying the date it was made, followed by a table counting the number of sequence per locus per species in the Project's records.



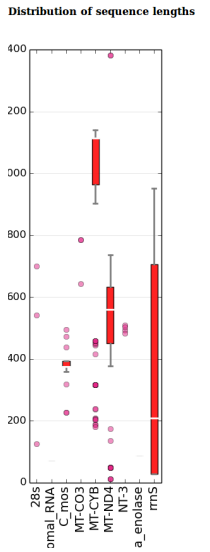
reprophylo analysis from Mon Dec 1 22:38:27 2014

Data

Species Representation In Sequence Data

species	5.8S_ribosomal_RNA	MT-ND4	MT-CYB	C_mos	rrnS	NT-3	MT-CO3	alpha_enolase	28s
Amblyrhynchus cristatus	0	2	12	0	1	0	0	0	0
Brachylophus fasciatus	0	1	8	1	0	1	0	0	0
Brachylophus vitiensis	0	0	12	0	0	0	0	0	0
Conolophus marthae	0	0	16	0	0	0	0	0	0
Conolophus pallidus	0	1	7	1	0	1	0	0	0
Conolophus subcristatus	0	1	130	1	4	1	0	0	0
Ctenosaura acanthura	0	2	0	0	0	0	0	3	0

The second section is a set of plots showing a bunch of sequence statistics, such as sequence length, for each locus.



This is followed by a description of each of the Conf objects that were used to align, trim and reconstruct a tree. These descriptions include the names of the analyzed loci, the time the analysis started, the command lines that were run, environment info and the length of the execution.

AlnConf named MafftDefaults with ID 1973514174725777.26

```
AlnConf named MafftDefaults with ID 1973514174725777.26
Loci: 5.8S_ribosomal_RNA,MT-ND4,MT-CYB,C_mos,rrnS,NT-3,MT-CO3,alpha_enolase,28s
Executed on: Mon Dec  1 22:22:57 2014
Commands:
5.8S_ribosomal_RNA: mafft 1973514174725777.26_5.8S_ribosomal_RNA.fasta
MT-ND4: mafft 1973514174725777.26_MT-ND4.fasta
MT-CYB: mafft 1973514174725777.26_MT-CYB.fasta
C_mos: mafft 1973514174725777.26_C_mos.fasta
rrnS: mafft 1973514174725777.26_rrnS.fasta
NT-3: mafft 1973514174725777.26_NT-3.fasta
MT-CO3: mafft 1973514174725777.26_MT-CO3.fasta
alpha_enolase: mafft 1973514174725777.26_alpha_enolase.fasta
28s: mafft 1973514174725777.26_28s.fasta

Environment:
Platform: Linux-3.13.0-40-generic-x86_64-with-Ubuntu-14.04-trusty
 Processor: x86_64
 Python build: defaultMar 22 2014 22:59:56
 Python compiler: GCC 4.8.2
 Python implementation: CPython
 Python version: 2.7.6
 ete2 version: 2.2rev1056
 biopython version: 1.64
 dendropy version: 3.12.0
 cloud version: 2.8.5
 User: amir-TECRA-W50-A
 Program and version:   MAFFT v7.123b (2013/10/15)
execution time:
9.25475597382
```

The results section of the report include the following alignment statistics table

Global Alignmnet Statistics

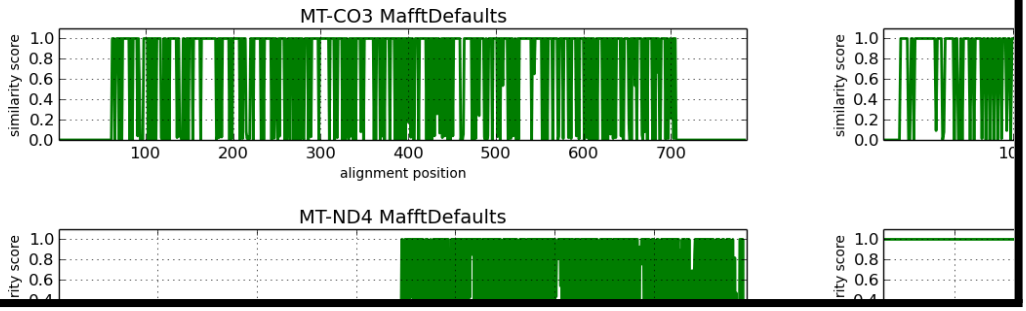
Name=Alignment name
NumPos=Alignment length
NumSeq=Number of sequences
Unique=Number of unique sequences
GapProp=Average gap proportion
VarCols=Total variable positions
ParsInf=Parsimony informative positions
UnSeqs=Completely undetermined sequences (only gaps)

Name	NumPos	NumSeq	Unique	GapProp	VarCols	ParsInf	UnSeqs	
5.8S_ribosomal_RNA@MafftDefaults	72	21	1	0.000000	0	0	0	
MT-ND4@MafftDefaults	1386	320	246	0.602239	373	328	0	
MT-CYB@MafftDefaults	1140	294	161	0.137904	550	468	0	
C_mos@MafftDefaults	504	27	20	0.251690	55	36	0	
rrnS@MafftDefaults	956	10	5	0.597280	85	75	0	
NT-3@MafftDefaults	526	14	10	0.065861	160	16	0	
MT-CO3@MafftDefaults	786	49	30	0.172119	180	171	0	
alpha_enolase@MafftDefaults	88	21	5	0.000000	4	2	0	
28s@MafftDefaults	785	23	5	0.783661	241	14	0	
rrnS@MafftDefaults@gappyout	703	10	5	0.543101	78	75	0	
C_mos@MafftDefaults@gappyout	370	27	19	0.035235	48	36	0	
alpha_enolase@MafftDefaults@gappyout	88	21	5	0.000000	4	2	0	
MT-CYB@MafftDefaults@gappyout	819	294	150	0.064414	403	353	0	
MT-ND4@MafftDefaults@gappyout	361	320	170	0.040313	182	162	10	['U66233.1_f0', 'U66232.1_f0', 'U66235.1_f0', 'U66228.1_f0', 'U66237.1_f0', 'U66229.1_f0', 'U66236.1_f0', 'U66227.1_f0', 'U66238.1_f0', 'U66234.1_f0']
MT-CO3@MafftDefaults@gappyout	643	49	29	0.000000	179	170	0	
NT-3@MafftDefaults@gappyout	487	14	10	0.007480	156	15	0	
28s@MafftDefaults@gappyout	127	23	5	0.003766	86	14	0	

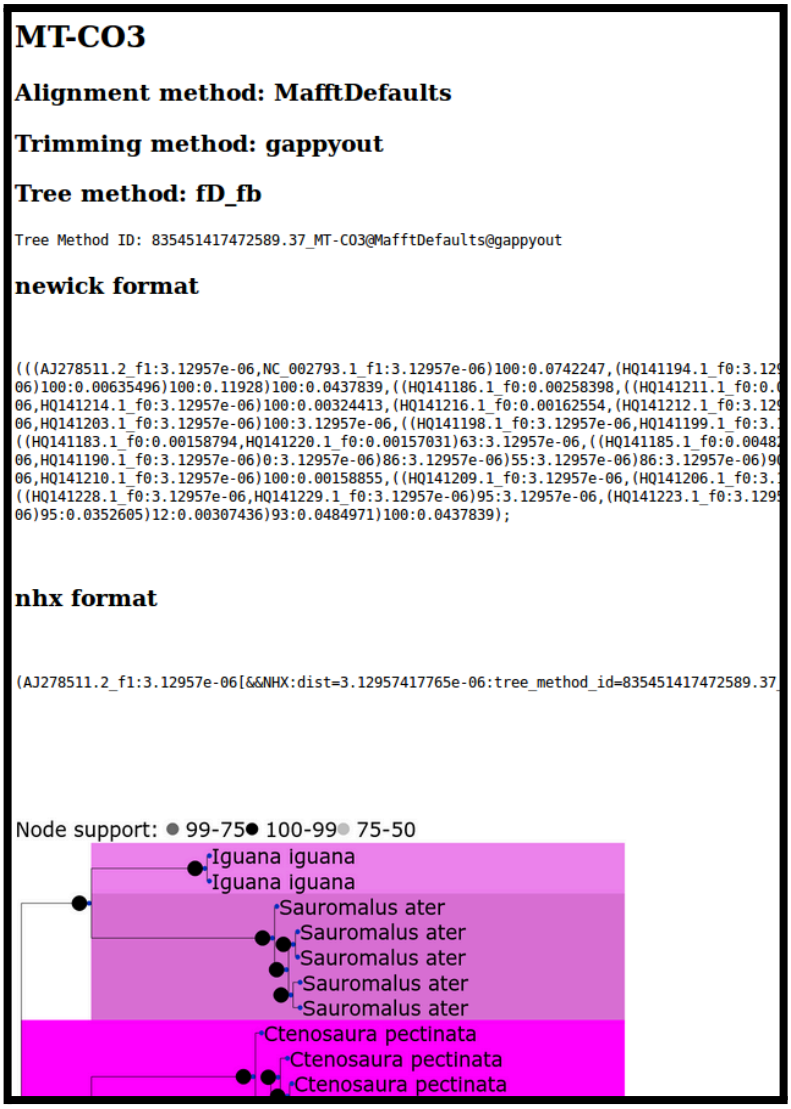
Alignment 5.8S_ribosomal_RNA@MafftDefaults has less than 4 unique sequences and will be dropped

as well as plots showing gap proportions and residue similarity values for each position in each full and trimmed alignment.

Trimal's Residue Similarity Score (-scc)



The final section of the report shows all the trees in the Project, including their newick and nhx representations, and the figure generated.



MT-CO3

Alignment method: MafftDefaults

Trimming method: gappyout

Tree method: fD_fb

Tree Method ID: 835451417472589.37_MT-CO3@MafftDefaults@gappyout

newick format

(((AJ278511.2_f1:3.12957e-06,NC_002793.1_f1:3.12957e-06)100:0.0742247,(HQ141194.1_f0:3.12
06)100:0.00635496)100:0.11928)100:0.0437839,((HQ141186.1_f0:0.00258398,((HQ141211.1_f0:0.
06,HQ141214.1_f0:3.12957e-06)100:0.00324413,(HQ141216.1_f0:0.00162554,(HQ141212.1_f0:3.12
06,HQ141203.1_f0:3.12957e-06)100:3.12957e-06,((HQ141198.1_f0:3.12957e-06,HQ141199.1_f0:3.
((HQ141183.1_f0:0.00158794,HQ141220.1_f0:0.00157031)63:3.12957e-06,((HQ141185.1_f0:0.0048
06,HQ141190.1_f0:3.12957e-06)0:3.12957e-06)86:3.12957e-06)55:3.12957e-06)86:3.12957e-06)90
06,HQ141210.1_f0:3.12957e-06)100:0.00158855,((HQ141209.1_f0:3.12957e-06,(HQ141206.1_f0:3.
((HQ141228.1_f0:3.12957e-06,HQ141229.1_f0:3.12957e-06)95:3.12957e-06,(HQ141223.1_f0:3.12
06)95:0.0352605)12:0.00307436)93:0.0484971)100:0.0437839);

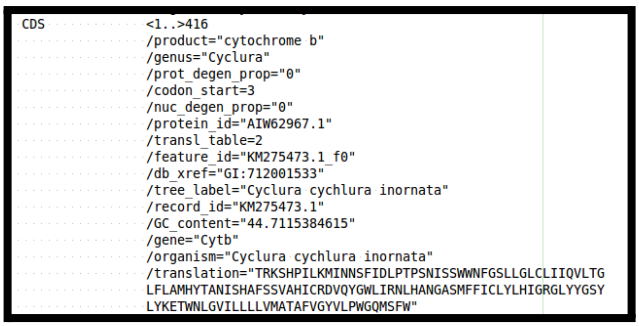
nhx format

(AJ278511.2_f1:3.12957e-06[&&NHX:dist=3.12957417765e-06:tree_method_id=835451417472589.37

Node support: 99-75 100-99 75-50

Iguana iguana
Iguana iguana
Sauromalus ater
Sauromalus ater
Sauromalus ater
Sauromalus ater
Sauromalus ater
Ctenosaura pectinata
Ctenosaura pectinata
Ctenosaura pectinata

The archive includes several additional files. One of them is a genbank file, which is different from the input file in several ways. First of all, it includes any data we included, even sequences we have passed through the Read Data tool (see below). It also includes some automatically included feature qualifiers (such as feature id and GC content) and ones that were manually added (such as genus), as the snippet below shows. We have written this data in genbank file format because it is a standard format that can be opened and analysed easily by many scripts and other programs.



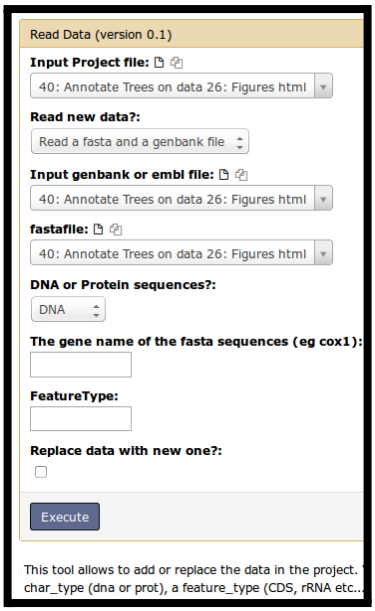
```
CDS             <1..>416
                /product="cytochrome b"
                /genus="Cyclura"
                /prot_degen_prop="0"
                /codon_start=3
                /nuc_degen_prop="0"
                /protein_id="AIW62967.1"
                /transl_table=2
                /feature_id="KM275473.1_f0"
                /db_xref="GI:712001533"
                /tree_label="Cyclura cychlura inornata"
                /record_id="KM275473.1"
                /GC_content="44.7115384615"
                /gene="Cytb"
                /organism="Cyclura cychlura inornata"
                /translation="TRKSHPILKMINNSFIDLPTPSNISSWWNFGSLLGLCLIIQVLTG
                LFLAMHYTANISHAFSSVAHICRDVQYGWLIRNLHANGASMFFICLYLHIGRGLYYGSY
                LYKETWNLGVILLLLVMATAFVGYVLPWGQMSFW"
```

Finally, it includes a PhyloXML file, containing all the trees and alignments format. The leaf attributes in those trees include the aligned and trimmed-aligned sequences, as the snippet below shows. Remember you can produce sequence alignment files in any format using the Report tool.

8.11. Tools not covered by this use case

Earlier we included data from a genbank file in a new project. The Read Data tool allows us to add data to an existing Project. We have the option to read a genbank file, a fasta file or both. Since ReproPhylo is capable to read any file format of unaligned or aligned sequences supported by BioPython, this feature will soon follow in the galaxy tools as well.

In the example below, the choice taken is to add both file formats. We therefore get a dropdown menu from which to choose a genbank file, and another to choose a fasta file. We also need to specify whether the fasta file is DNA or protein, the gene name of the fasta sequences and their feature type. The tool can be used several times consecutively to add several files. The command line tool also allows to use the file name as gene and feature, in order to enable reading multiple files in one go, so this will follow in Galaxy as well.



The Manage Project tool includes most of the functions encountered in other tools, but in a way that allows to break them down to independent operations or to clump them in a different way than they are clumped by the other tools. This should allow more flexibility in designing a workflow.

8.12. Export your history

The History panel allows you to export the history to a file, which can then be imported by others (or your future self) into Galaxy. This file will include all the inputs, outputs and intermediates, as well as a record of all the tools and parameter choice. The history is exported to a file by choosing “Export to file” from the gear button at the top right side of the panel.

8.13. Save and edit a workflow

The history can be saved as a workflow, which will present the tools in the order they were used and will allow you to repeat the analysis with or without changing the parameters. You can also edit the workflow to add or remove tools, as well as split and parallelize the workflow. See this page on how to do all those things.

9. FAQ

9.1. Where can I get ReproPhylo?

The software is available to download from the ReproPhylo GitHub repository. It is under the most permissive licence we could find, CC0, which makes it public domain. Our intention is that you can do anything you wish with this software including re-using, modifying, and incorporating into other software, whether commercial or not. It would be great however if you fed any improvements back in to ReproPhylo.

9.2. How can I cite ReproPhylo?

If you use ReproPhylo in a publication, please cite:

Szitenberg A, John M, Blaxter ML, Lunt DH. ReproPhylo: An Environment for Reproducible Phylogenomics. PLoS Comput Biol. 2015;11: e1004447. doi:10.1371/journal.pcbi.1004447

The programs running within the ReproPhylo pipeline should also be cited appropriately. So, if you align with MAFFT, trim the alignment with TrimAl and create a tree with RAxML or PhyloBayes which you modify with ETE2 you should cite appropriately. All the program references are in the next section.

9.3. I have found an error in the code or manual

If something is generating an error, and you think it is a bug rather than your setup, you should create an issue (bug report) on GitHub. See if you can replicate the error using a standard Docker ReproPhylo installation- that way we will know it is not the environment. If you can help you could fix the code yourself and then issue a pull request on GitHub, otherwise create a GitHub issue.

If the problem is with documentation then you can just directly edit this manual to improve or correct it. Be bold. We consider this manual an evolving community document and actively encourage your contributions. You could also email the authors if you would like to discuss the documentation or the code at A.Szitenberg@hull.ac.uk.

9.4. I would like [my favourite feature] included

There are far too many approaches in phylogenetics for us to attempt to include them all, though we will try to build in the most important. Suggestions could be added to the GitHub issue page. It is not that difficult to ‘wrap' many existing programs so that they will operate within the ReproPhylo pipeline. Have a look at adding tools into ReproPhylo section, we would welcome contributions. If you would like to take a more active role in developing ReproPhylo, welcome, please email us.

10. Program References

RAxML: A. Stamatakis: "RAxML Version 8: A tool for Phylogenetic Analysis and Post-Analysis of Large Phylogenies". In Bioinformatics, 2014

Phylobayes: N. Lartillot, T. Lepage and S. Blanquart, 2009: PhyloBayes 3: a Bayesian software

package for phylogenetic reconstruction and molecular dating. Bioinformatics Vol. 25 no. 17.

MAFFT: Katoh, Standley 2013 (Molecular Biology and Evolution 30:772-780) MAFFT multiple sequence alignment software version 7: improvements in performance and usability.

Muscle: Edgar 2004: MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Research 32(5):1792-1797

Pal2Nal: Mikita Suyama, David Torrents, and Peer Bork (2006) PAL2NAL: robust conversion of protein sequence alignments into the corresponding codon alignments.*Nucleic Acids Res.* **34**, W609-W612.

trimAl: Salvador Capella-Gutierrez; Jose M. Silla-Martinez; Toni Gabaldon. trimAl: a tool for automated alignment trimming in large-scale phylogenetic analyses. Bioinformatics 2009 25: 1972-1973.

ETE: Jaime Huerta-Cepas, Joaquin Dopazo and Toni Gabaldon. ETE: a python Environment for Tree Exploration. BMC Bioinformatics 2010, 11:24.

NumPy: Stefan van der Walt, S. Chris Colbert and Gael Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011)

Matplotlib: John D. Hunter. Matplotlib: A 2D Graphics Environment ,Computing in Science & Engineering, 9, 90-95 (2007)

Pandas: Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

HTML.py: http://www.decalage.info/python/html

Biopython: Cock PJ, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, Friedberg I, Hamelryck T, Kauff F, Wilczynski B, and de Hoon MJ. Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics 2009 Jun 1; 25(11) 1422-3.doi:10.1093/bioinformatics/btp163 pmid:19304878

Cython: Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn and Kurt Smith. Cython: The Best of Both Worlds, Computing in Science and Engineering, 13, 31-39 (2011)

Cloud: https://pypi.python.org/pypi/cloud/2.8.5

11. Contact

For general questions about ReproPhylo contact Amir Szitenberg (szitenberg@gmail.com) or Dave Lunt (dave.lunt@gmail.com). For technical issues, bug reports, and feature suggestions you should create a GitHub issue if possible. If the problem is with documentation then you can just directly edit this manual to improve or correct it. Be bold.