The input data

Navigate to the folder containing the iPyrad output files (`ranaddrad_output_files`), called (see end of 3t_iPyrad_tutorial for details). If you haven't yet downloaded these output files, they're available here. Take a look at the phylip file (containing both SNPs and invariant sites) that we'll use as input into RAxML-ng to build a phylogenetic tree. What does this file look like, and what information does it contain?

```
cd ranaddrad_output_files
# relevant file is ranaddrad.phy
```

Install RAxML-ng

Let's install the RAxML-ng software which will allow us to reconstruct phylogenies with our data. The installation instructions and manual for RAxML-ng is available here. You should be able to download it (as you did with iPyrad) using miniconda with the following command:

```
conda install -c bioconda raxml-ng
```

Check to see that RAxML-ng worked by typing the following. It doesn't matter where you are.

```
raxml-ng -h
# the above line (as with any software help command) outputs a bunch
of information about parameter settings in raxml-ng
```

Running RAxML-ng

Normally, we'd submit the tree as a job to TACC, but given that we don't actually have that much data (in terms of sequence length and number of individuals), we can run this tree in a development node. This should take ~30 minutes to run.

To make sure we don't run out of time, start a development node for the maximum amount of time, two hours.

```
idev -t 2:00:00 -A Phylogenomics
```

Now, run RAxML-ng with the following flags:

`--msa`	Set this to be the *Rana* sequence file (phylip file). *Make sure to give RAxML-ng the full path to this file so it knows where to look!*
`--model`	Set the substitution model to be GTR+G
`--tree`	Set this to `rand` which means random starting tree.
`--bs-trees`	Set this to do `100` bootstraps. Doing 200 is more typical, but we want this tree to run pretty quickly so we'll stick to 100 for now.
`--threads`	Set the threads to `2`. This will speed up the tree-building process.
`--all`	No need to add a value for this flag, but it needs to be included in your line of code because you're telling RAxML-ng to both estimate the ML tree and perform bootstrapping at the same time.

```
./raxml-ng –msa ../ranaddrad_output_files/ranaddrad.phy –model GTR+G –
tree rand –bs-trees 100 –threads 2 –all
```

What gets printed to the screen as RAxML-ng is running the tree? This output is identical to the contents of the `*.log` file.

```
# RAxML-ng prints output as it reads in the input file, tells
parameters we've set (i.e., how many partitions we've set (0), how
many sites there are (97.02% invariant sites), etc.

# RAxML-ng then generates 10 random starting trees
# then we see the bootstraps running
# finally we see the final results and the paths for where the output
files are saved
```

What files are outputted by RAxML-ng once it's done running the tree?

```
ls ranaddrad.phy.raxml*
# there are 8 files:
# .bestModel, .bestTree, .bootstraps, .mlTrees, .rba, .startTree,
.support, and .log
```

What does the `.log` file contain?

```
cat ranaddrad.phy.raxml.log
# this contains exactly the same printout as what RAxML-ng printed out
to the screen while it was running
```

What does the `.bestTree` file contain?

```
cat ranaddrad.phy.raxml.bestTree
# this contains the best tree that RAxML-ng found with branch lengths
(but no support values)
```

What does the `.support` file contain?

```
cat *.support
# this contains the best tree with branch lengths and bootstraps
```

Make a Nexus tree file

For data visualization, we're going to want a Nexus file to import into R. We can do this manually in a text editor. As you learned in lecture, Nexus files are unique in that they have "blocks" of data within them. Let's look at a Nexus file containing a tree:

```
#NEXUS
begin trees;
    tree tree1 = (((A,B),C),(D,E));
end;
```

Which file did you obtain from RAxML that has *both support and branch lengths plotted on the best tree?* This is the tree we'll want to use for subsequent analyses.

```
ranaddrad.phy.raxml.support # has both BLs and BS on the tree
```

Open a text editor such as Atom or BBEdit. Copy the contents of the above file with support and branch lengths and add the required text to make this into a Nexus file. Save the file as `ranaddrad_tree.nexus`. What does your file look like?

```
#NEXUS
begin trees;
      tree tree1 =
(Rneo_T527:0.000230,(((Rber_T1113a:0.000032,Rber_T1113b:0.000024)100:0
.001202,Rber_T1114:0.001314)100:0.001110,(((Rsph_T26064:0.001412,Rsph_
T25870:0.001426)100:0.009780,(Rbla_D2865:0.000280,Rbla_D2864:0.000288)
100:0.009519)86:0.001861,((Rchi_T2034b:0.000154,Rchi_T2034a:0.000027)1
00:0.000136,Rchi_T2049:0.000151)100:0.022596)100:0.008479)100:0.003392
,Rneo_T480:0.000208):0.0;
end;
```

Advanced phylogenetics: to do in your own time

Feel free to try adding/changing parameters from the RAxML-ng run we did above. You could do 200 bootstrap replicates rather than 100, or adjust any other parameters to see how the tree changes. Make sure to change the output file names so they don't overwrite the tree files you just ran.

```
./raxml-ng –msa ../ranaddrad_output_files/ranaddrad.phy –model GTR+G –
tree rand –bs-trees 200 –threads 2 –all
```

Installing FigTree

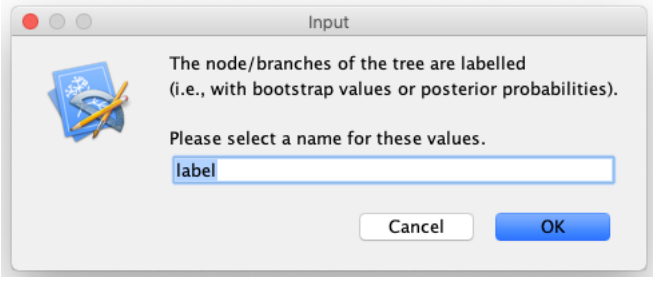
FigTree (made by Andrew Rambaut) is a great program to visualize trees. Download the program according to your OS here. This is a "regular" (i.e., GUI) program so you don't need to do compilation or anything; your computer should handle the installation.

Copy the RAxML-ng output file containing *branch lengths and support values for the best tree* from TACC to your computer. *Recall that this is easiest to do if you copy while in your personal computer.*

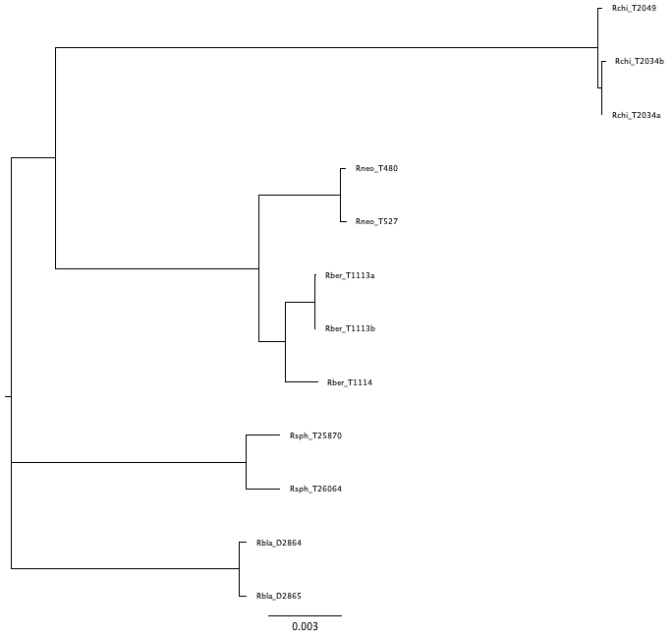
```
scp
eac3496@frontera.tacc.utexas.edu:/scratch1/03123/eac3496/training_cour
se/ranaddrad_raxml/ranaddrad.phy.raxml.support .
```

Visualize the tree

Let's take a look at the phylogeny you just constructed. FigTree only recognizes tree files (i.e., those with the file extension `.tree`). Make a copy of the tree you used above (containing both branch lengths and bootstrap support values). Manually add `.tree` to the extension of the file you just copied and open it up in FigTree. A popup (below) will appear asking you to select a name for the node labels. Change this label to "bootstrap", because that's what the labeling in the tree file is.

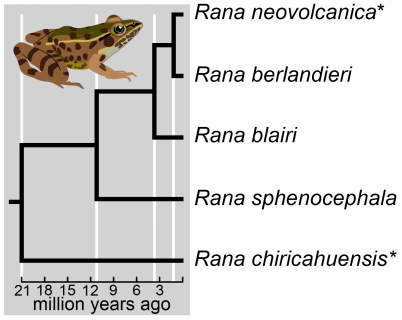


What does the tree look like?

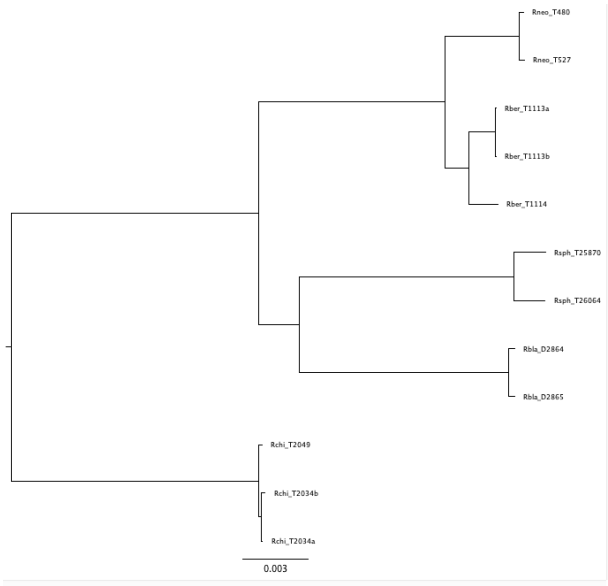


Reroot the tree

You may notice that the relationships seem off from the image I showed in lecture (see below). This is because the tree isn't rooted in the right place.



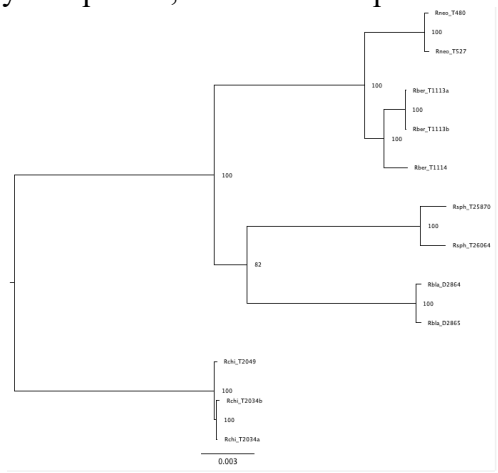
We can reroot the tree in the correct location in FigTree. *R. chiricahuensis* should be the outgroup species. To do this, click on the branch leading to all three *R. chiricahuensis* individuals and you'll see it highlighted in blue. Then, click the icon to "Reroot" (it has a yellow arrow). You should now see that the *R. chiricahuensis* individuals are the outgroup to the remaining individuals on the tree.



Display bootstrap support values

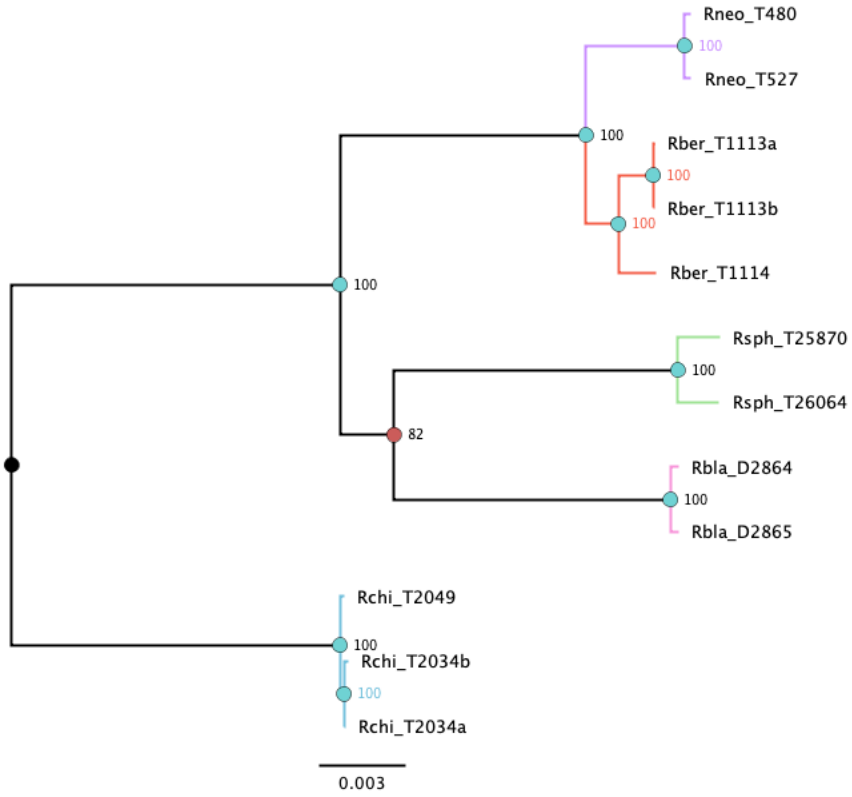
You may have noticed that the bootstrap support values aren't displayed. To do so, check off the "Node Labels" checkbox on the lefthand side of the screen. But wait! It's still not showing bootstraps; it's showing something else.

Click the arrow directly to the left of Node Labels. You can now specify what you want FigTree to display. Under the "Display" dropdown, click "bootstrap". Now the values should appear.



Other fun stuff in FigTree

You can explore FigTree and colorize clades, branches, thicken branches or increase the font size of labels.



Export the tree

To export the tree you've made, click File > Export PDF… and save the file.