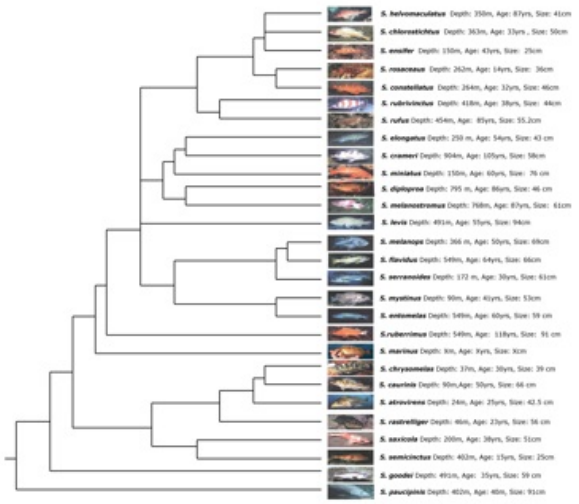
TreeSnatcher Plus

Tutorial 6 Fuzzy images

What we want

The trees depicted in the images are difficult to preprocess in one way or another.
For each image, we will look at the possibilities TreeSnatcher Plus offers you to prepare it adequately.

In the accompanying ZIP-file you will find nearly all images from this tutorial. You should try to reproduce the work steps for each image as detailed below or find another solution.



Suggestions:
Drag a **selection box** around the tree but omit the images.

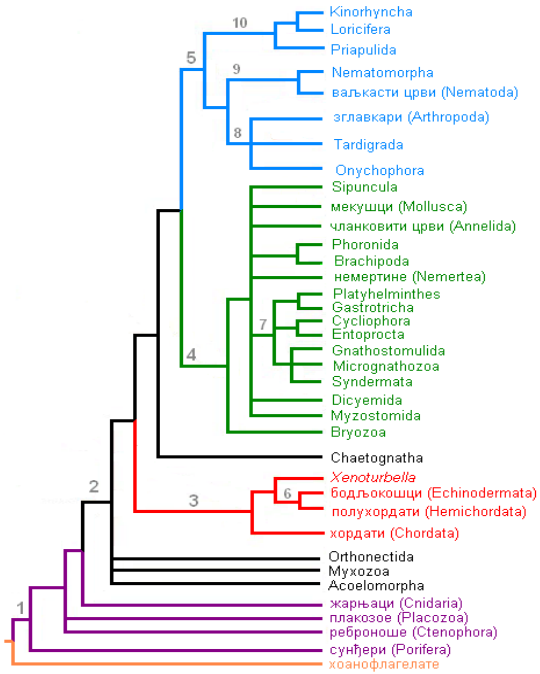
Use **Trim** or **Crop** for a better overview: **Trim** clears the image outside the selection box, **Crop** copies the image portion within the selection box into a fresh image.

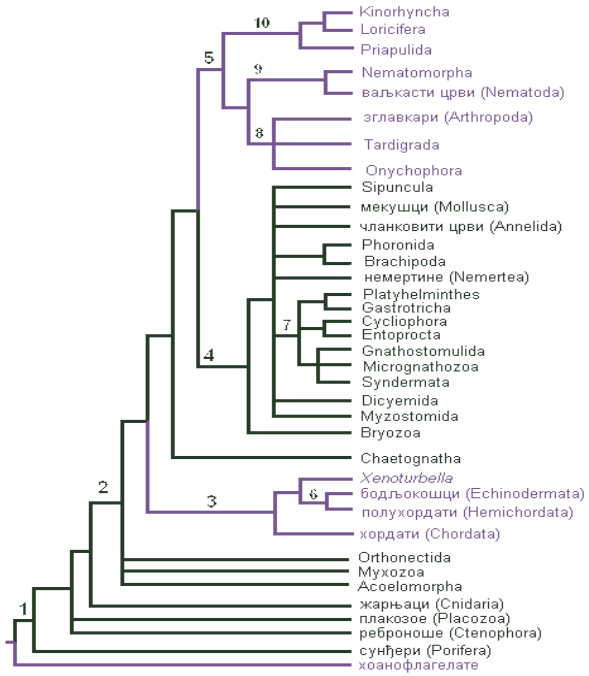
Sharpen the lines, work with different pencil widths to paint over lines that are too rough.

Binarize the image.

Scale it using **Double Size**. Scaling cannot be undone.

If you use the tree type **Rectangular**, check how the program has divided the branches into segments. It is likely that you have to straighten lines and to accentuate some bends.





We can binarize this image with a suitably low threshold value. If there is no universal threshold value for an image, we can try to reduce the colors used in the image prior to binarization.

Here, try **Color Quantization** with 3 as the desired number of colors in the image. This technique seldom produces intuitive results, you should therefore play around with it.

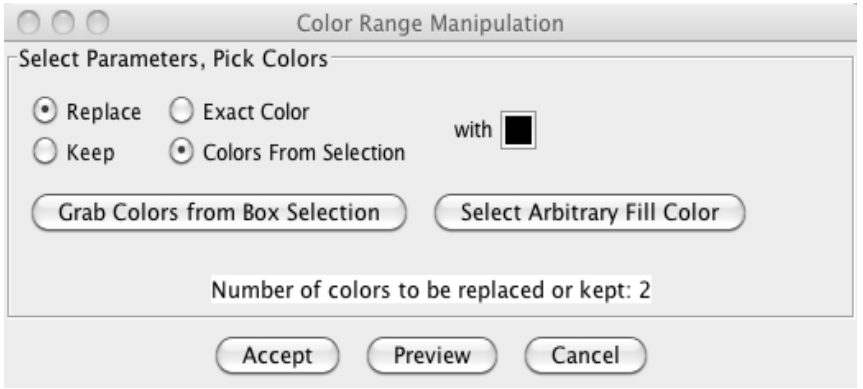
Alternatively, you can get rid of a color by replacing it by another. TreeSnatcher Plus offers two mechanisms: **Pipette/Fill** and the **Color Dialog**.

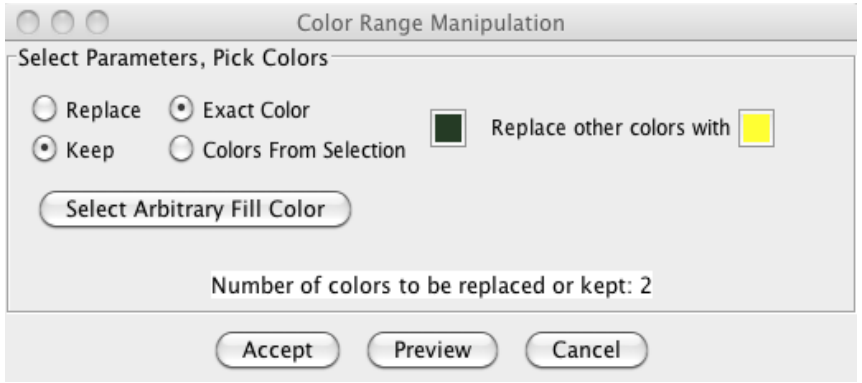
You can select a color from the modified image with the **Pipette** tool. The red, green and blue components at the coordinate beneath the mouse cursor are shown in the bottom left part of the window. The fill color itself is also shown in the left toolbar.

Select **Fill** and click into the image:
The program colors the pixel you clicked with the fill color. Then it moves on to the neighbors and colors all neighbors that have the same color as the first pixel, and so on. Technically spoken, this operation is an iterative flood fill. It is widely used in drawing applications.

The coloring mechanisms offered by the fill dialog work different:
Instead of replacing one color in an enclosed area by another, it either replaces at least one color in the whole image by another, or it preserves at least one color and replaces the remaining colors.

Click **Fill Dialog**. This dialog box appears. Its content varies depending on your choice.



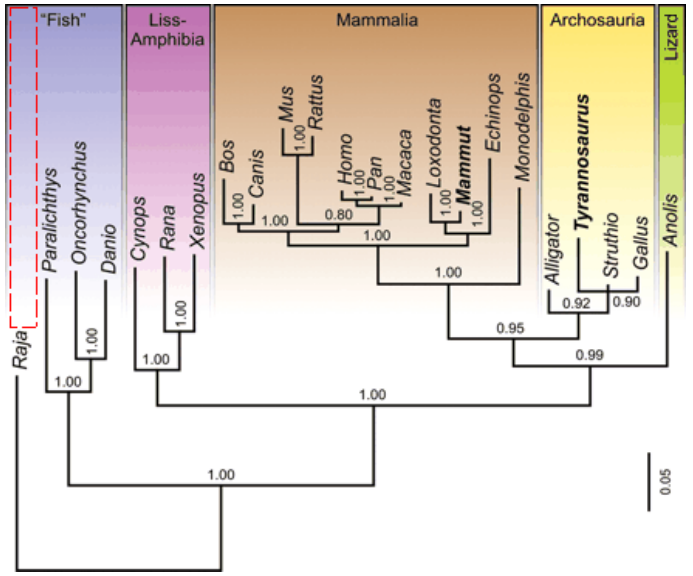


The upper combination of settings allows you to replace all colors within the selection box by another color, currently black. You must place the selection box before you click **Fill Dialog**. The program has found two distinct colors in the selection.

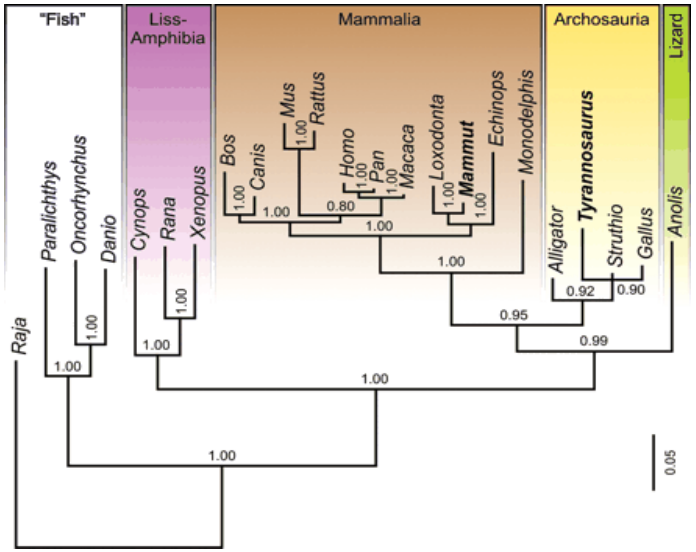
You can replace black with a color from the image: Click into the color box, then into the image. The color in the box changes.

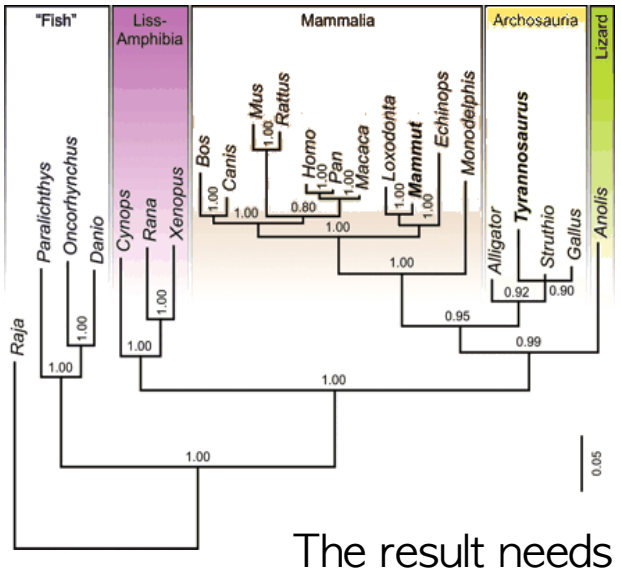
The lower combination of settings allows you to preserve the color green, as shown in the color box, and replace all other colors in the image with yellow. You can also select both colors from the image, or you use a different fill color using **Select Arbitrary Fill Color**.

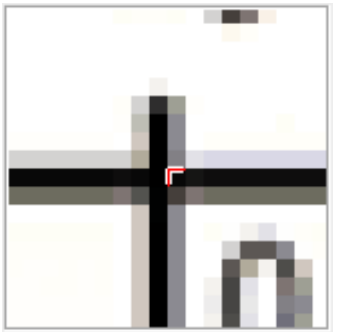
An example for a color range manipulation:



Place the **Box Selection**.
Click **Fill Dialog**.
Select Replace and Colors From Selection.
From the image, select binary white as fill color.
Click **Grab Colors from Box Selection**.
The Box Selection should now encompass the whole image.
Click **Preview**. If you are satisfied with the result, click **Done**.
After some more of these manipulations, this could be the outcome:



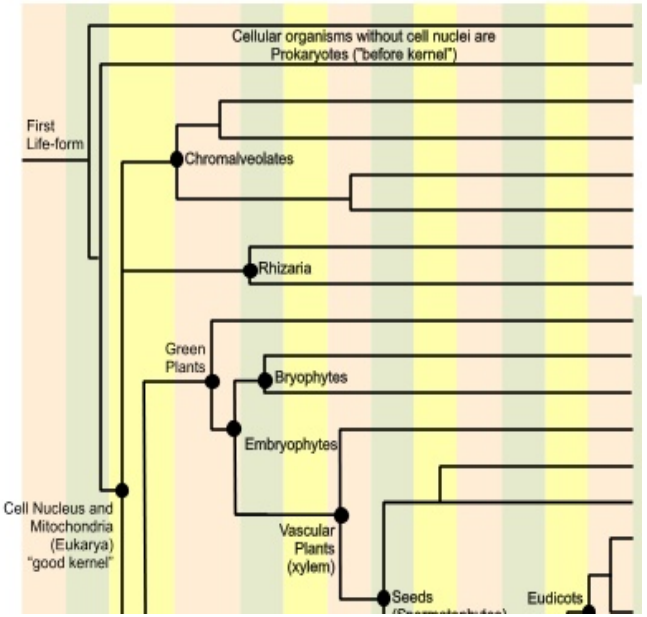




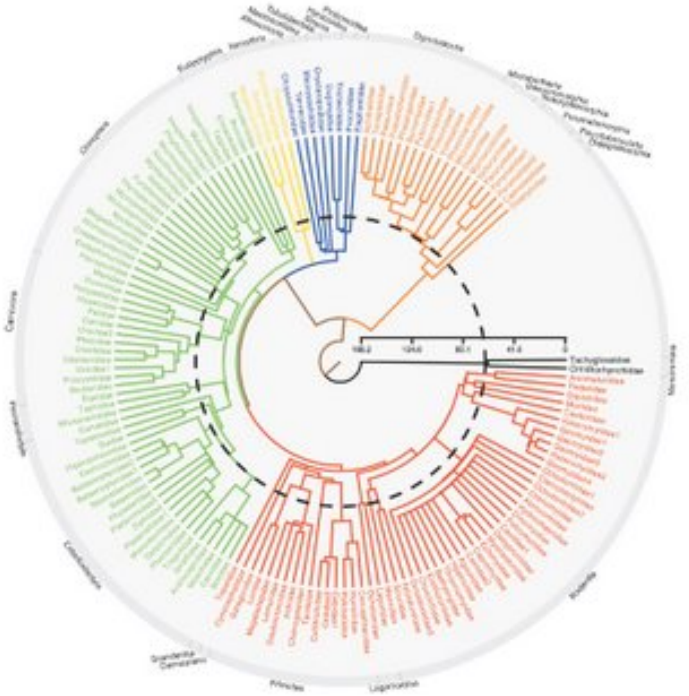
The result needs further refinement.

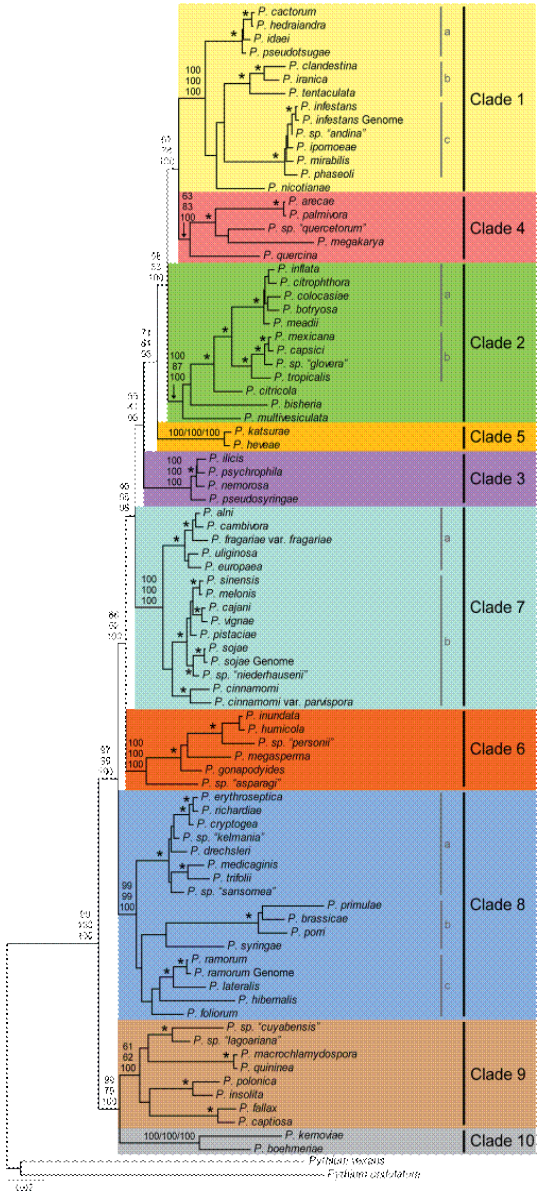
For the preparation of images in TreeSnatcher Plus, it is our top priority to completely isolate the foreground from the background. To get rid of as many colors as possible is always helpful.

This is a portion of another image.
The preparation is fairly easy:
Issue **Local Threshold**, then **Thin**.



I have not successfully tried this so far:





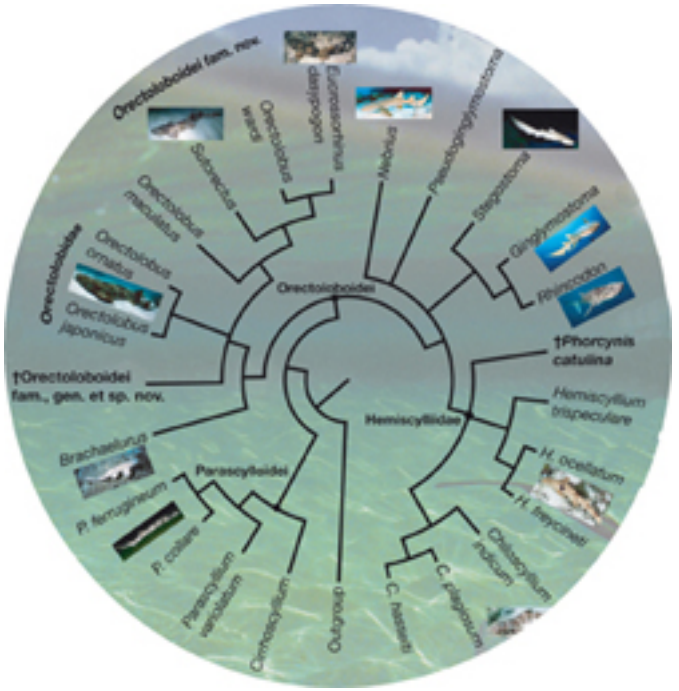
To preprocess this image is very difficult:
The color patches are not uniform.
Some branches are hard to locate.

Some suggestions:
Fill the left part of the image with gray.
You should see the dashed lines now.
Paint fresh lines over the dashed lines.

Select a vertical portion of the image that encompasses as many colors as possible.
Perform a color range manipulation: Replace those colors with white.

Or try to get rid of most colors using **Grayscale Conversion**.

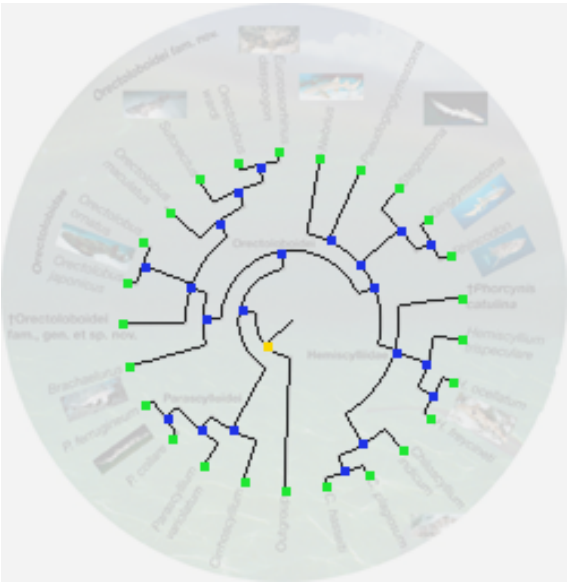
Probably a lot of manual work will remain if the program shall interpret the rectangular topology correctly.

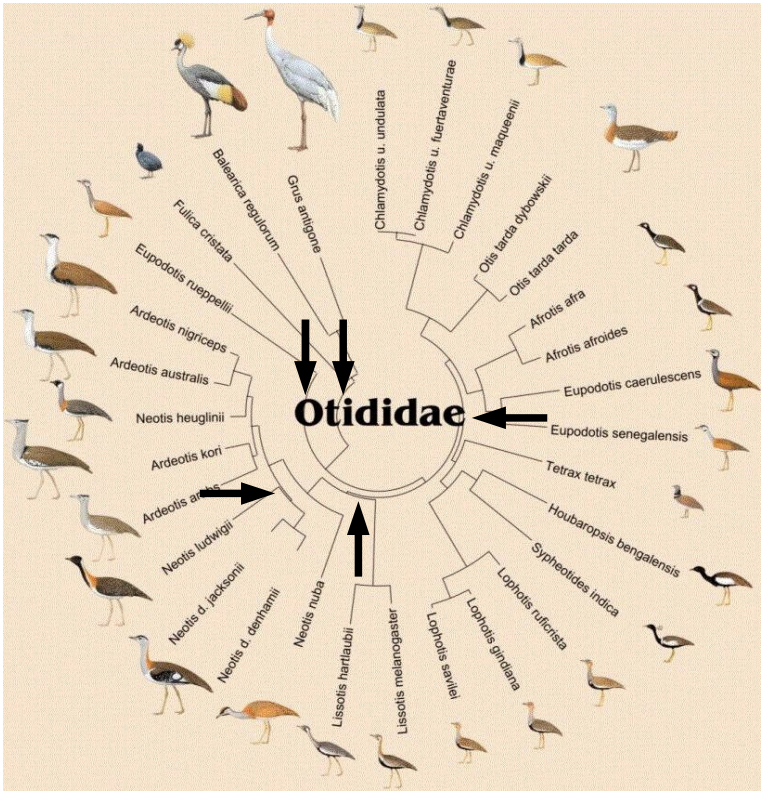


Perform a **Grayscale Conversion** with the red factor in the first, the green factor in the third and the blue factor in the first quarter.
Sharpen with Gauss kernel width 1, Amplification 1x,
Minimum filter with box width around 6,
Local Threshold,
Thin,
Flood Foreground,
Extract Foreground,
Isolate overlapping text.

TreeSnatcher Plus does not yet support length relevant branch segments in round tree topologies. This feature is nearly finished.

Round trees pose a tougher challenge than rectangular trees: It is more difficult to find the branch segments and to decide which are length relevant.





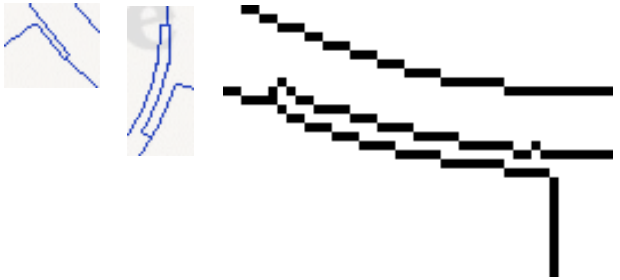
Here, try **Sharpen**, Minimum and **Threshold**. Keep an eye on the highlighted positions.

Activate **Mask User Drawings**, **Thin Freehand Drawings** and **Flood User Drawings**.

Flood the **tree**. You will immediately see where the gaps in the tree are. After you have closed them with the **Pencil**, the flooded area gets larger.

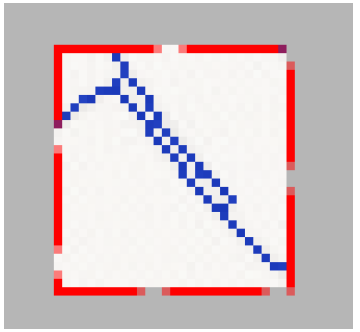
Thin the whole image again after you have corrected the tree with the drawing tools.

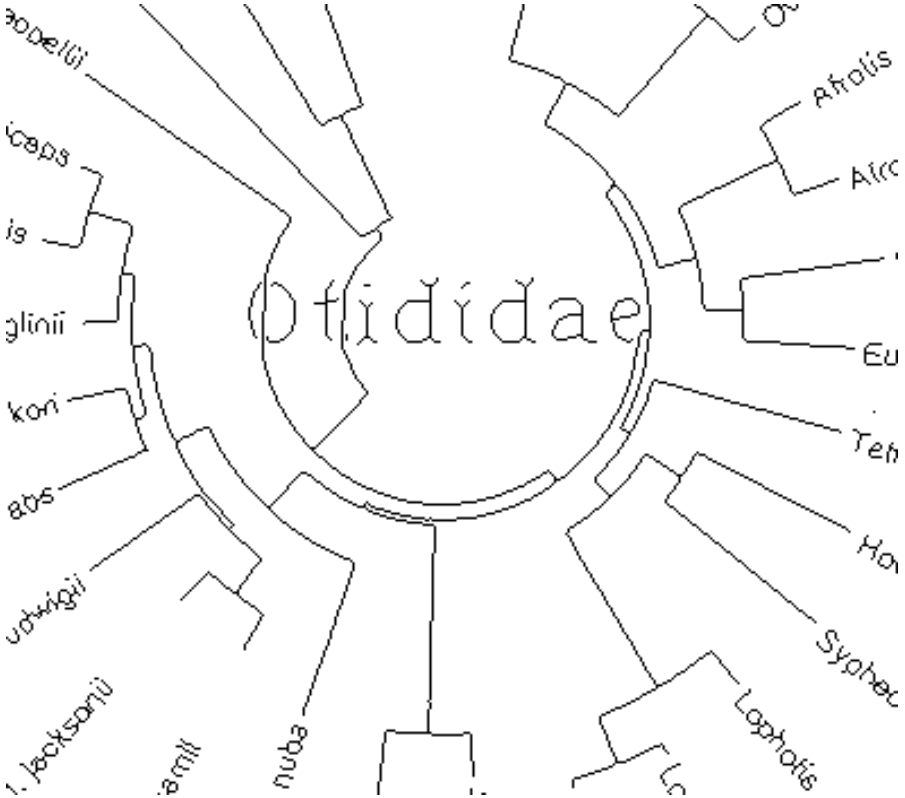
The benefit of Mask User Drawings combined with **Black Line**



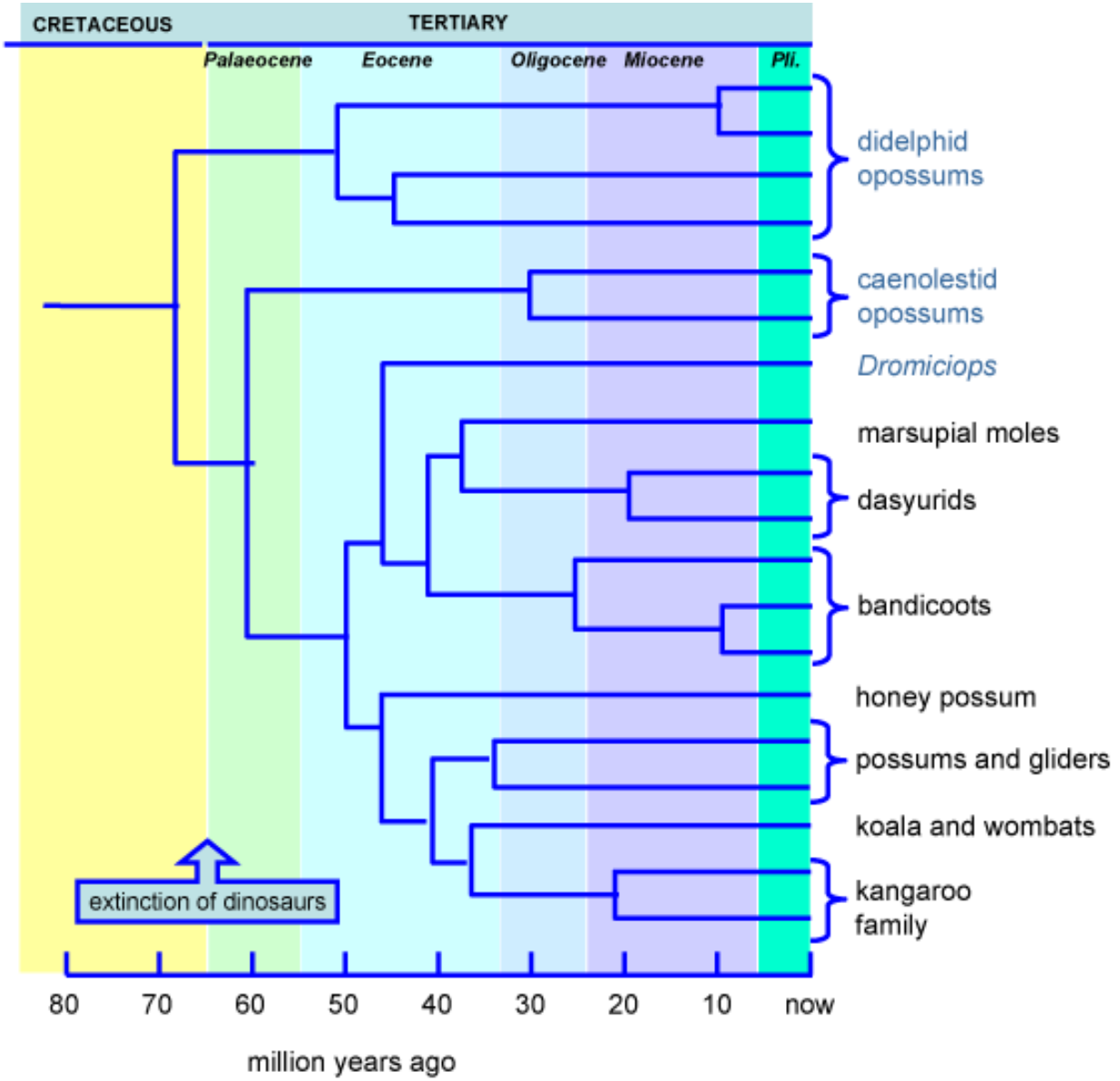
This is the right way to separate lines.

This is the wrong way to separate lines.





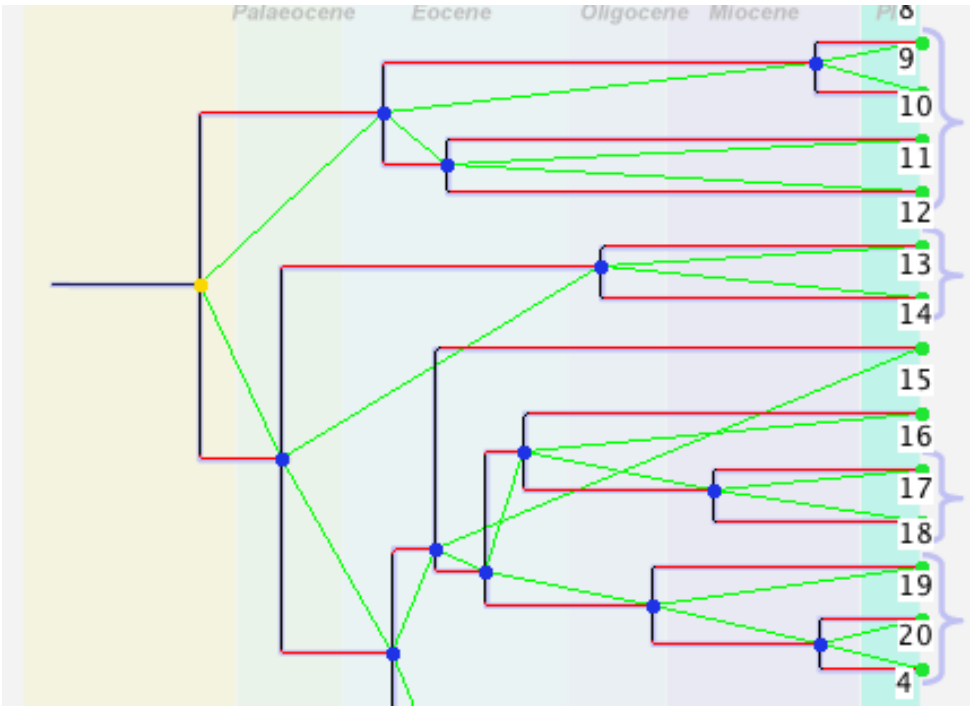
The remaining work steps should be routine by now. Keep in mind that TreeSnatcher Plus cannot yet recognize length relevant branch segments in this sort of tree topology.



At first glance **Local Threshold** appeared efficient here.

Unfortunately, upon closer inspection, we see that the last column is missing.

After noticing that the tree has a uniform color, things are surprisingly easy:
Use the **Fill Dialog**, keep only the color of the tree - blue, replace all other colors with white.
Thin the resulting foreground, correct some positions...



(((12:140, 13:141):141, ((14:213, ((15:174, (16:91, 17:91):84):17, (18:118, (19:44, 20:44):74):74):22):19, (4:214, ((2:156, 3:156):31, (5:168, (6:97, 7:97):72):19):26):18):49):36, ((8:46, 9:46):191, (10:209, 11:209):28):81);

One last comment on branch lengths:
The program uses and displays up to four decimal places for branch lengths,
but the smallest length unit in TreeSnatcher Plus is the diameter of a single pixel,
but we loose accuracy during the image processing,
but an image of a tree is never as exact as the underlying data source.

This means: Do not overestimate the accuracy of the branch lengths in the Newick representation.