
Newick Utilities Tutorial

Version 1.1 – November 3, 2009

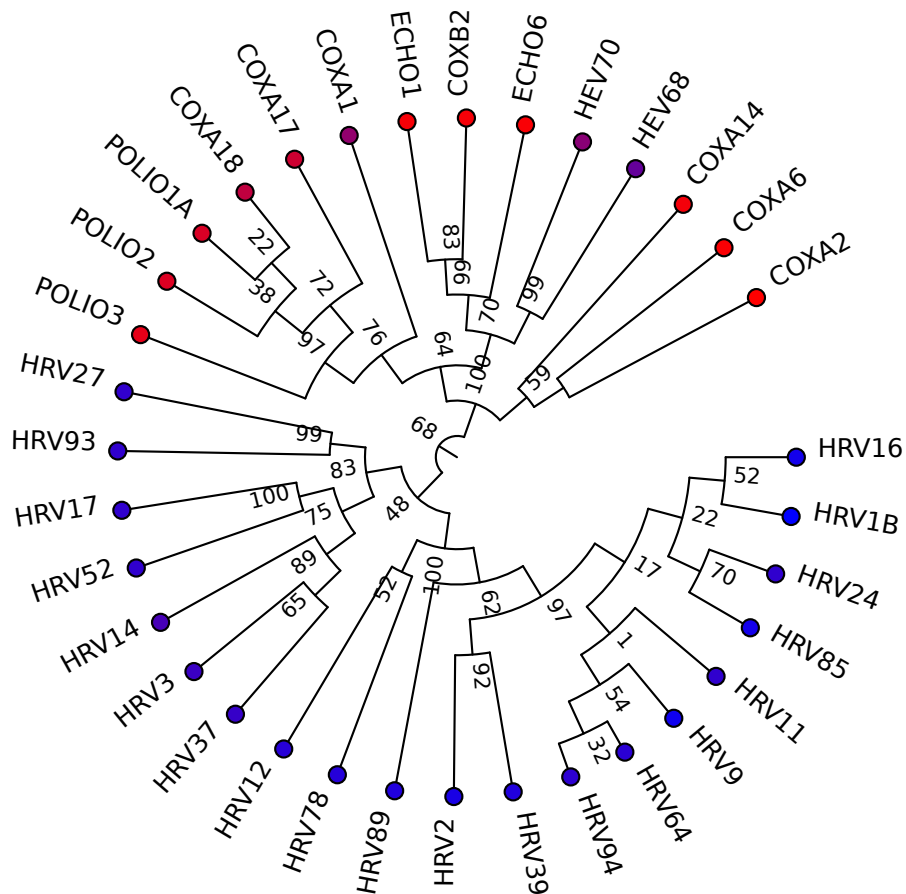
Thomas Junier thomas.junier@unige.ch

Computational Evolutionary Genomics Group

Department of Genetic Medicine and Development

University of Geneva, Switzerland

http://cegg.unige.ch/newick_utils



Contents

| | |
|--|----------|
| Introduction | 3 |
| 1 General Remarks | 5 |
| 1.1 Help | 5 |
| 1.2 Input | 5 |
| 1.2.1 Multiple Input Trees | 6 |
| 1.3 Output | 6 |
| 1.4 Options | 6 |
| 2 Simple Tasks | 7 |
| 2.1 Displaying Trees | 7 |
| 2.1.1 As Text | 7 |
| 2.1.2 As SVG | 8 |
| 2.1.3 Ornaments | 14 |
| 2.1.4 Options not Covered | 18 |
| 2.2 Rooting and Rerooting | 18 |
| 2.2.1 Rerooting on the ingroup | 21 |
| 2.3 Extracting Subtrees | 23 |
| 2.3.1 Monophyly | 25 |
| 2.3.2 Context | 26 |
| 2.3.3 Siblings | 26 |
| 2.3.4 Limits | 27 |
| 2.4 Computing Bootstrap Support | 27 |
| 2.5 Retaining Topology | 29 |
| 2.6 Extracting Distances | 33 |
| 2.6.1 Selection | 35 |
| 2.6.2 Methods | 37 |
| 2.6.3 Alternative formats | 38 |
| 2.7 Finding subtrees in other trees | 38 |
| 2.8 Renaming nodes | 40 |
| 2.8.1 Breaking the 10-character limit in PHYLIP alignments | 40 |
| 2.8.2 Higher-rank trees | 42 |
| 2.9 Condensing | 44 |
| 2.10 Pruning | 45 |
| 2.11 Trimming trees | 47 |
| 2.12 Indenting | 49 |
| 2.13 Extracting Labels | 53 |
| 2.13.1 Counting Leaves in a Tree | 53 |
| 2.14 Ordering Nodes | 54 |
| 2.15 Generating Random Trees | 55 |
| 2.16 Stream editing | 56 |

| | | |
|----------|--|-----------|
| 2.16.1 | Opening Poorly-supported Nodes | 60 |
| 3 | Advanced Tasks | 62 |
| 3.1 | Checking Consistency with other Data | 62 |
| 3.1.1 | By condensing | 62 |
| 3.2 | Bootscanning | 65 |
| 3.3 | Number of nodes vs. Tree Depth | 66 |
| 4 | Python Bindings | 68 |
| 4.1 | API Documentation | 69 |
| A | Defining Clades by their Descendants | 70 |
| A.1 | Why not just use node numbers? | 71 |
| B | Newick order | 72 |
| C | Installing the Newick Utilities | 73 |
| C.1 | From source | 73 |
| C.2 | As binaries | 73 |

Introduction

The Newick Utilities are a set of UNIX (including Mac OS X) shell programs for working with phylogenetic trees. Their main features are:

- they require no user interaction¹
- they can work on any number of trees at a time
- they perform well with large trees
- they are implemented as filters
- they read and write text

They are not tools for *making* phylogenies. Rather, they are for processing existing ones, by which I mean manipulating the tree or extracting information from it: rerooting, simplifying, extracting subtrees, printing branch lengths and distances, etc - see table 1; a glance at the table of contents should also give you an idea.

Each of the programs performs one task (with some variants). For example, here is how you would reroot a series of phylograms contained in file `mytrees.nw` using node `Dmelano` as the outgroup:

```
$ nw_reroot mytrees.nw Dmelano
```

Now, you might want to make cladograms from the rerooted trees. Program `nw_topology` does the job, and since the utilities are filters, you can do it in a single command:

```
$ nw_reroot mytrees.nw Dmelano | nw_topology -
```

As you can see, it is straightforward to pipe Newick Utilities together, and of course they can be mixed freely with any other shell program (see e.g. 2.13.1).

Organization of This Document

This tutorial is organized as follows: chapter 1 discusses common features of the Newick Utilities, chapter 2 shows examples of simple tasks, and chapter 3 has examples of more advanced tasks.

It is not necessary to read this material in order: you can pretty much jump to any section in chapters 2 and 3, they do not require reading previous sections. I would suggest reading chapter 1, then section 2.1 because it explains how all the tree graphics were produced.

The files for all the examples in this tutorial can be found in subdirectory `data`.

¹Why this is a good thing is not the focus of this document: I shall assume that if you are reading this, you already know when a command-line interface is better than an interactive interface.

| Program | Function |
|----------------|--|
| nw_clade | Extracts subtrees specified by node labels |
| nw_condense | Condenses (simplifies) trees |
| nw_display | Shows trees as graphs (ASCII graphics or SVG) |
| nw_distance | Prints distances between nodes, in various ways |
| nw_ed | Stream editor (<i>à la</i> sed or awk) |
| nw_gen | Random tree generator |
| nw_indent | Shows Newick in indented form |
| nw_labels | Prints node labels |
| nw_match | Finds matches of a tree in another one |
| nw_order | Orders tree (preserving topology) |
| nw_prune | Removes branches based on labels |
| nw_rename | Changes node labels according to a mapping |
| nw_reroot | (Re)roots the tree |
| nw_support | Computes bootstrap support of a tree given replicate trees |
| nw_topology | Alters branch properties, preserving topology |
| nw_trim | Trims a tree at a specified depth |

Table 1: The Newick Utilities and their functions

Chapter 1

General Remarks

The following applies to all programs in the Newick Utilities package.

1.1 Help

All programs print a help message if passed option `-h`. Here are the first 20 lines of `nw_indent`'s help:

```
$ nw_indent -h | head -20
```

```
Indents the Newick, making structure more clear.
```

```
Synopsis
```

```
-----
```

```
nw_indent [-cht:] <newick trees filename|->
```

```
Input
```

```
-----
```

```
Argument is the name of a file that contains Newick trees, or '-' (in which case trees are read from standard input).
```

```
Output
```

```
-----
```

```
By default, prints the input tree, with each parenthesis and each leaf on a line of its own, and indented a multiple of ' ' (two spaces) to reflect structure. The default output is valid Newick.
```

The help page describes the program's purpose, its input and output, and its options, in a format reminiscent of UNIX manpages. It also shows a few examples. All examples can be tried out using files in the `data` directory.

1.2 Input

Since the Newick Utilities are for working with trees, it should be no surprise that the main input is a file containing trees. The trees must be in Newick format, which is

one of the most widely used tree formats. Its complete description can be found at <http://evolution.genetics.washington.edu/phylip/newicktree.html>.

The input file is always the first argument to the program (after any options). It may be a file stored in a filesystem, or standard input. In the latter case, the filename is replaced by a '-' (dash):

```
$ nw_display mytrees.nw
```

is the same as

```
$ cat mytrees.nw | nw_display -
```

or

```
$ nw_display - < mytrees.nw
```

Of course the second ("dashed") form is only really useful when chaining several programs into pipelines.

1.2.1 Multiple Input Trees

The input file can contain one or more trees. When there is more than one, I prefer to have one tree per line, but this is not a requirement: they can be separated by any amount of whitespace, including none at all. The task will be performed¹ on each tree in the input. So if you need to reroot 1,000 trees on the same outgroup, you can do it all in a single step (see 2.2).

1.3 Output

All output is printed on standard output (warnings and error messages go to standard error). The output is either trees or information about trees. In the first case, the trees are in Newick format, one per line. In the second case, the format depends on the program, but it is always text (ASCII graphics, SVG, numeric data, textual data, etc.).

1.4 Options

Options change program behaviour and/or allow extra arguments to be passed. They are all passed on the command line, before the mandatory argument(s), using a single letter preceded by a dash, in the usual UNIX way. There are no mandatory control files, although some tasks require additional files (*e.g.* 2.1.2). For example, we saw above that `nw_display` produces graphs. By default the graph is ASCII graphics, but with option `-s`, the program produces SVG:

```
$ nw_display -s sometree.nw
```

All options are described in the program's help page (see 1.1).

¹well, attempted...

Chapter 2

Simple Tasks

The tasks shown in this chapter all involve a single Newick Utilities program (plus possibly `nw_display`), so they can serve as introduction to each individual program.

2.1 Displaying Trees

Perhaps the simplest and most common operation on a Newick tree is just to look at it. But a Newick tree is not very intuitive for us humans, as we can quickly see by looking *e.g.* at a tree of Old World primates:

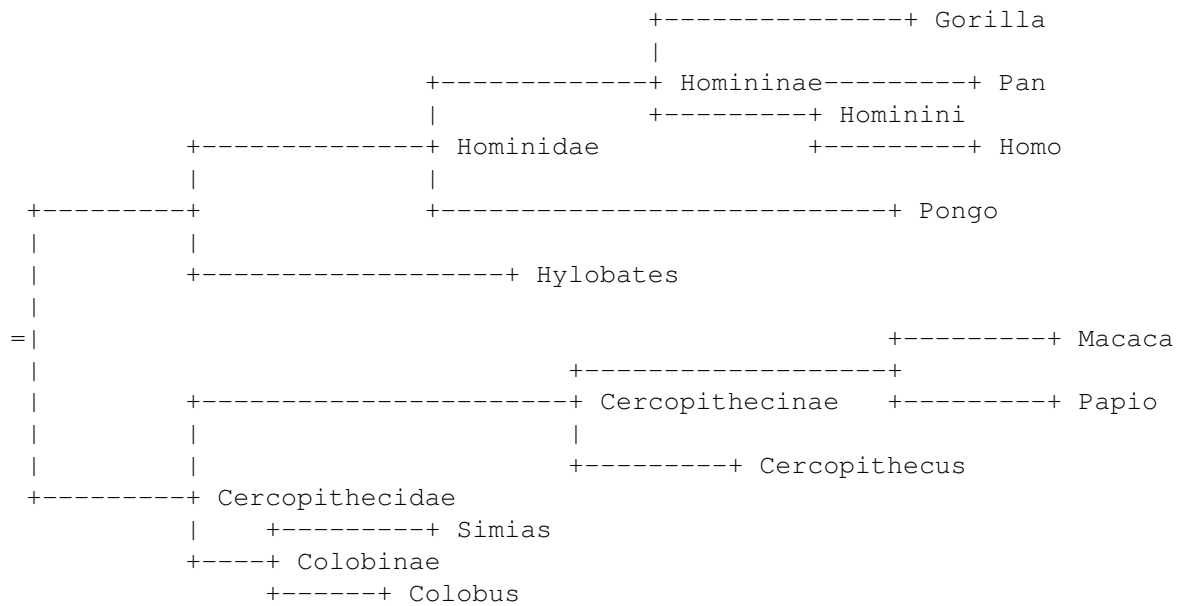
```
$ cat catarrhini  
  
(((Gorilla:16,(Pan:10,Homo:10)Hominini:10)Homininae:15,Pongo:30)  
Hominidae:15,Hylobates:20):10,(((Macaca:10,Papio:10):20,  
Cercopithecus:10)Cercopithecinae:25,(Simias:10,Colobus:7)  
Colobinae:5)Cercopithecidae:10);
```

So we want to make a graphical representation from it. This is the purpose of the `nw_display` program.

2.1.1 As Text

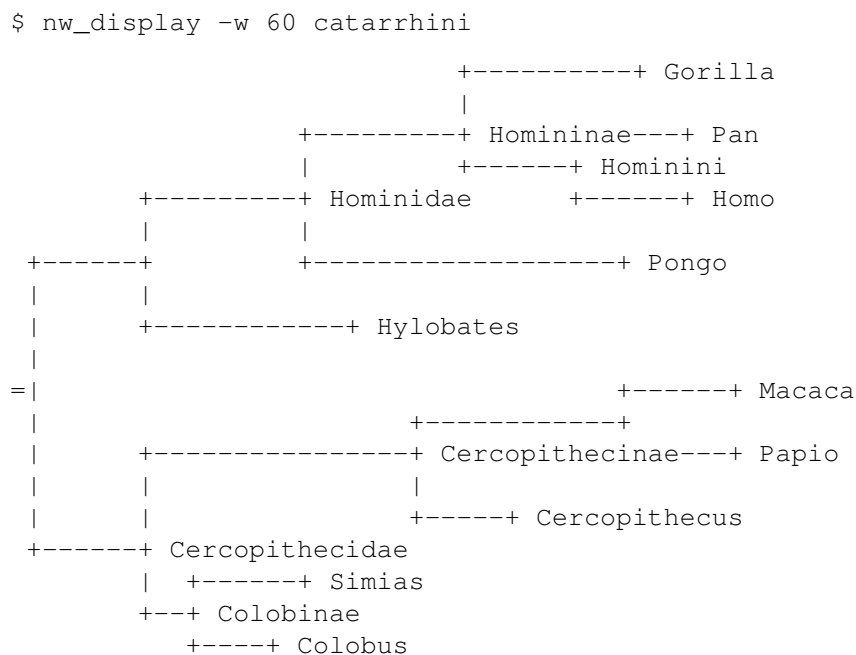
At its simplest, `nw_display` just outputs a text graph. Here is the primates tree, shown with `nw_display`:

```
$ nw_display catarrhini
```



That's pretty low-tech compared to interactive, colorful graphical displays, but if you use the shell a lot (like I do), you may find it useful.

You can use option `-w` to set the number of columns available for display (the default is 80):



2.1.2 As Scalable Vector Graphics

First, a disclaimer: there are dozens of tools for viewing trees out there, and I'm not interested in competing with them. The reasons why I included SVG capabilities (besides automation, etc.) were:

- I wanted to be able to produce reasonably good graphics even if no other tool was at hand
- I wanted to be sure that large trees could be rendered¹

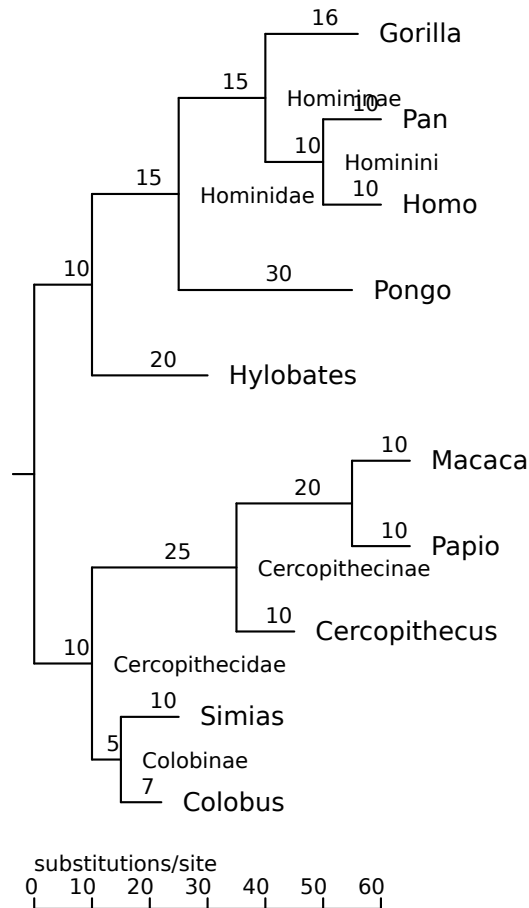
To produce SVG, pass option `-s`:

```
$ nw_display -s catarrhini > catarrhini.svg
```

Now you can visualize the result using any SVG-enabled tool (all good Web browsers can do it), or convert it to another format with, say `rsvg` or Inkscape (<http://www.inkscape.org>). The SVG produced by `nw_display` is designed to be easy to edit in an interactive editor (Inkscape, Adobe Illustrator, etc.): for example, the tree edges are in one group, and the text in another, so it is easy to change the line width of the edges, or the font family of the text (you can also do this from `nw_display` using a CSS map, see 2.1.2).

The following PDF image was produced like this:

```
$ inkscape -f catarrhini.svg -A catarrhini.pdf
```



All SVG images shown in this document were processed in the same way. In the rest of this document we will usually skip the redirection into an SVG file and omit the SVG-to-PDF conversion step.

¹I have had serious problems visualising trees of more than 1,000 leaves using some popular software I will not name here - either it was painfully slow, or it simply crashed, or else the output was unreadable, incomplete, or otherwise unsuitable.

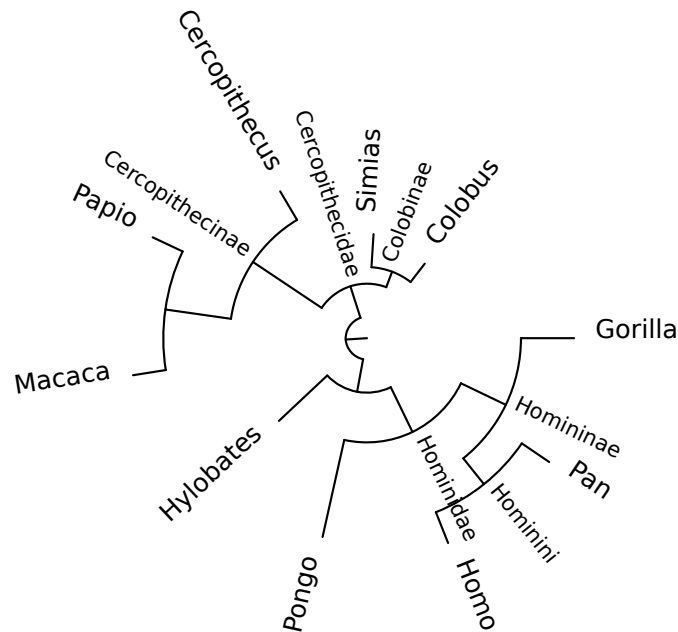
Scale Bar

If the tree is a phylogram, `nw_display` prints a scale bar.² Its units can be specified with option `-u`, the default is substitutions per site. To suppress the scale bar, pass the `-S` switch.

Radial trees

You can make radial trees by passing the `-r` switch:

```
$ nw_display -sr -S -w 450 catarrhini
```



You already know `-w`, except that for SVG the value is in pixels instead of columns.

Using CSS

You can modify node style using CSS. This is done by specifying a CSS *map*, which is just a text file that says which style should be applied to which node. If file `css.map` contains the following

²The positioning of the scale bar is a bit crude, especially for radial trees. This is mainly because of the "SVG string length curse", that is, the impossibility of finding out the length of a text string in SVG. This means it is hard to ensure the scale bar will not overlap with a node label, unless one places it far away in a corner, which is what I do for now. An improvement to this is on my TODO list.

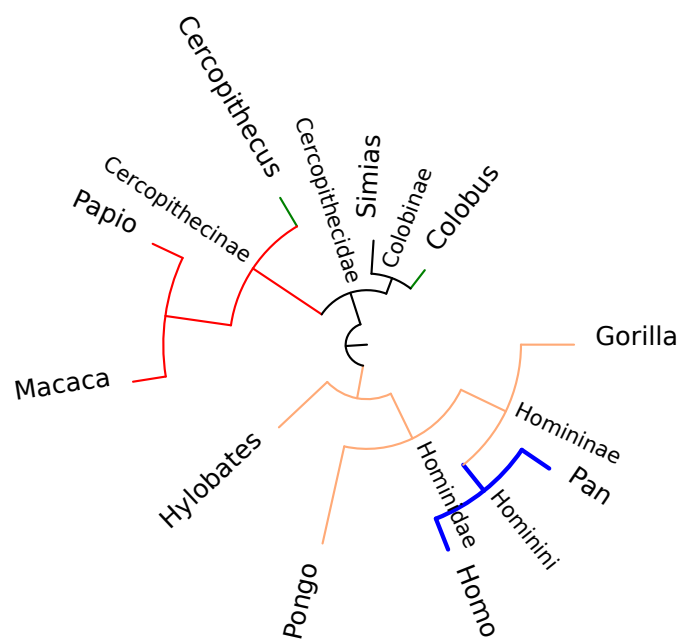
```

stroke:red Clade Macaca Cercopithecus
stroke:#fa7 C Homo Hylobates
stroke:green Individual Colobus Cercopithecus
"stroke-width:2; stroke:blue" Clade Homo Pan

```

we can apply the style map to the tree above by passing `-c`, which takes the name of the CSS file as argument:

```
$ nw_display -sr -S -w 450 -c css.map catarrhini
```



The syntax of the CSS map file is as follows:

- Each line describes one style and the set of nodes to which it applies. A line contains elements separated by whitespace (whitespace between quotes does not count).
- The first element of the line is the style, and it is a snippet of CSS code.
- The second element states whether the following nodes are to be treated individually or as a clade. It is either `Clade` or `Individual` (which can be abbreviated to `C` or `I`, respectively).
- The remaining element(s) are node labels and specify the nodes to which the style must be applied: if the second element was `Clade`, the program finds the

last common ancestor of the nodes and applies the style to that node and all its descendants. If the second element was `Individual`, then the style is only applied to the nodes themselves.

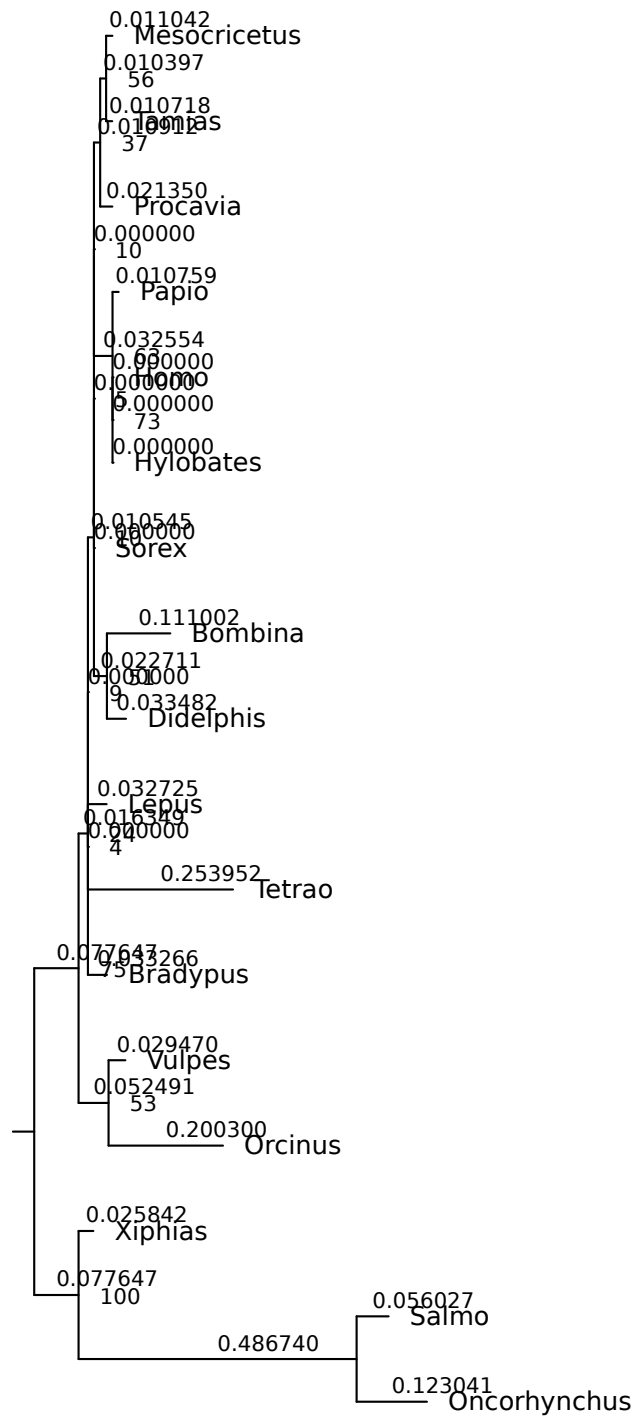
In our example, `css.map`:

- the first line states that the style `stroke:red` must be applied to the Clade defined by `Macaca` and `Cercopithecus`, which consists of these two nodes, their ancestor `Cercopithecinae`, and `Papio`.
- Line 2 prescribes that style `stroke:#fa7` (an SVG hexadecimal color specification) must be applied to the clade defined by `Homo` and `Hylobates`, which consists of these two nodes, their last common ancestor (unlabeled), and all its descendants (that is, `Homo`, `Pan`, `Gorilla`, `Pongo`, and `Hylobates`, as well as the inner nodes `Hominini`, `Homininae` and `Hominidae`).
- Line 3 instructs that the style `stroke:green` be applied individually to nodes `Colobus` and `Cercopithecus`, and only these nodes - not to the clade that they define.
- Line 4 says that style `stroke-width:2; stroke:blue` should be applied to the clade defined by `Homo` and `Pan` - note that the quotes have been removed: they are not part of the style, rather they allow us to improve readability by adding some whitespace.

The style of an inner clade overrides that of an outer clade, *e.g.*, although the `Homo - Pan` clade is nested inside the `Homo - Hylobates` clade, it has its own style (blue, wide lines) which overrides the containing clade's style (pinkish with normal width). Likewise, `Individual` overrides `Clade`, which is why `Cercopithecus` is green even though it belongs to a "red" clade.

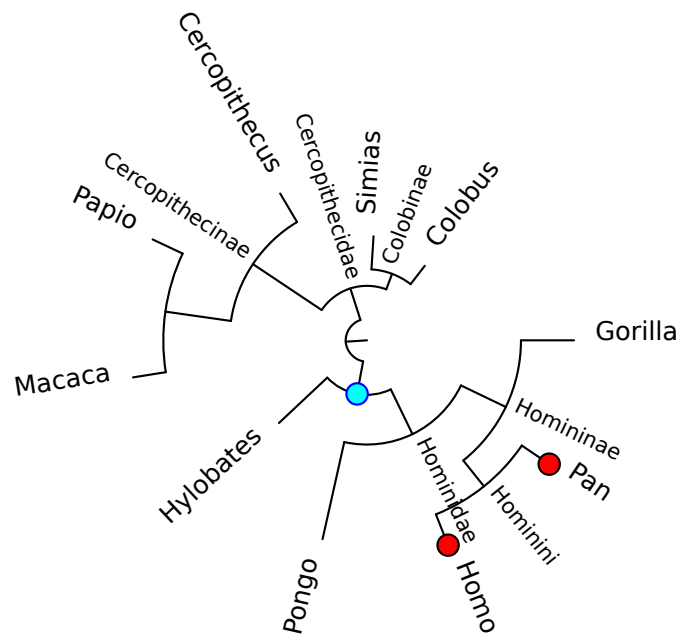
Styles can also be applied to labels. Option `-l` specifies the leaf label style, option `-i` the inner node label style, and option `-b` the branch length style. For example, the following tree, which was produced using defaults, could be improved a bit:

```
$ nw_display -sS vertebrates.nw
```



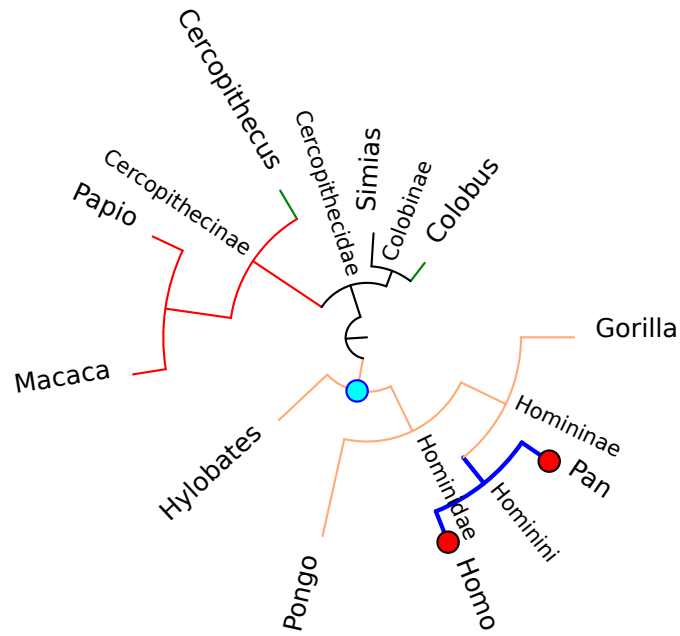
Let's remove the branch length labels, reduce the vertical spacing, reduce the size of inner node labels (bootstrap values), and write the leaf labels in italics, using a font with serifs:

```
$ nw_display -s -S -v 20 -b 'opacity:0' -i 'font-size:8' \
-l 'font-family:serif;font-style:italic' vertebrates.nw
```

Ornaments and CSS can be combined:

```
$ nw_display -sr -S -w 450 -o ornament.map -c css.map catarrhini
```



Example: Mapping GC Content

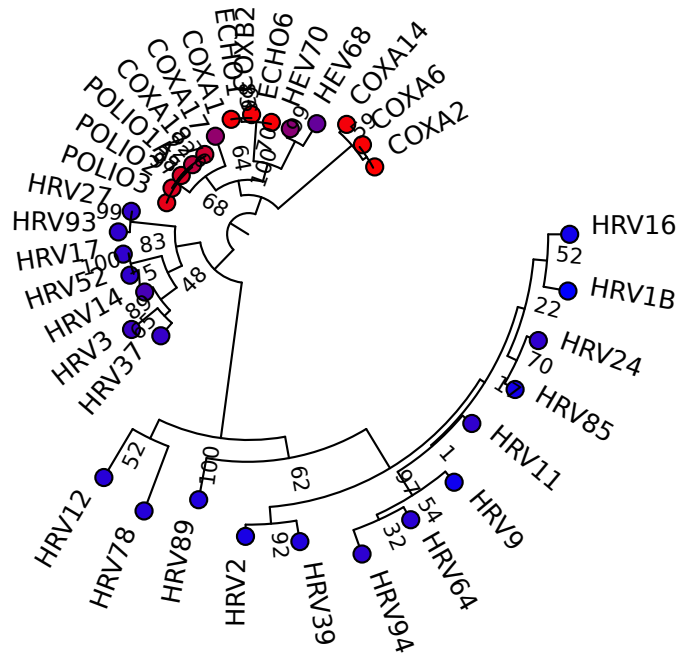
In a study of human rhinoviruses, I have produced a phylogenetic tree, `HRV_ingrp.nw`. I have also computed the GC content of the sequences, and mapped it into a gradient that goes from blue (33.3%) to red (44.5%). I used this gradient to produce a CSS map, `b2r.map`:

```
$ head -5 b2r.map
```

```
"<circle r='4' style='fill:#2500d9;stroke:black' />"      I      HRV78
"<circle r='4' style='fill:#2700d7;stroke:black' />"      I      HRV12
"<circle r='4' style='fill:#2100dd;stroke:black' />"      I      HRV89
"<circle r='4' style='fill:#0000ff;stroke:black' />"      I      HRV1B
"<circle r='4' style='fill:#1300eb;stroke:black' />"      I      HRV16
```

in which the `fill` values are hexadecimal color codes along the gradient. Then:

```
$ nw_display -sr -S -w 450 -o b2r.map HRV_ingrp.nw
```



As we can see, the high-GC sequences are all found in the same main clade.

Multiple SVG Trees

Like all Newick Utilities, `nw_display` can handle multiple trees, even in SVG mode. The best way to do this was not evident: one can generate one file per tree (but then we break the rule that every program is a filter and so writes to standard output), or one can put all the trees in one SVG document (but then we have to impose tiling or some other arrangement), or one can just output one SVG document after another. This is what we do (this may change in the future). So if you have many trees in document `forest.nw`, you can say:

```
$ nw_display -s forest.nw > forest_svg
```

But `forest_svg` isn't valid SVG – it is a concatenation of many SVG documents. You can just extract them into individual files with `csplit`:

```
$ csplit -sz -f tree_ -b '%02d.svg' forest_svg '/<?xml/' {*}
```

This will produce one SVG file per tree in `forest.nw`, named `tree_01.svg`, `tree_02.svg`, etc.

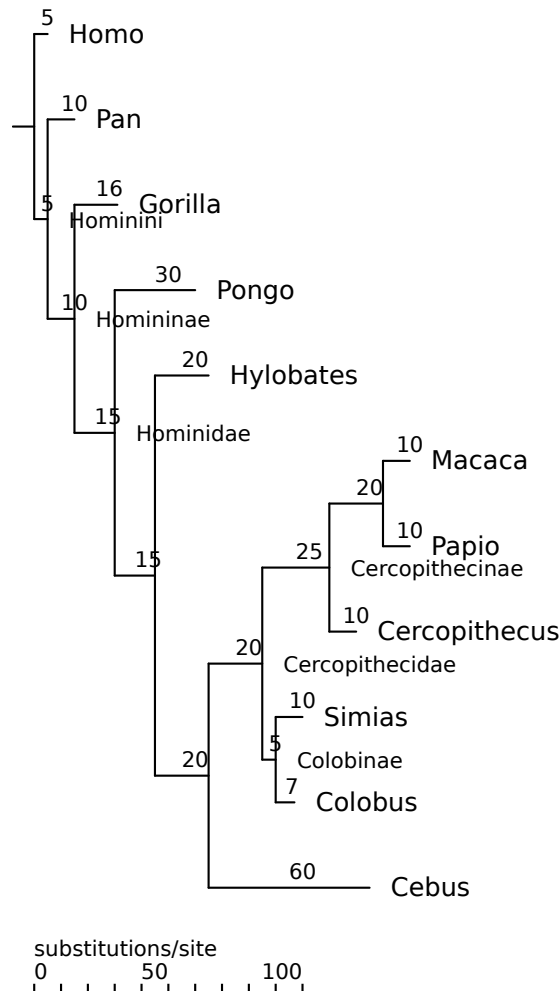
2.1.4 Options not Covered

`nw_display` has many options, and we will not describe them all here - all of them are described when you pass option `-h`. They include support for clickable images (with URLs to arbitrary Web pages), nudging labels, changing the root length, etc.

2.2 Rooting and Rerooting

Rooting transforms an unrooted tree into a rooted one, and rerooting changes a rooted tree's root. Some tree-building methods produce rooted trees (e.g., UPGMA), others produce unrooted ones (neighbor-joining, maximum-likelihood). The Newick format is implicitly rooted, in the sense that there is a 'top' node from which all other nodes descend. Some applications regard a tree with a trifurcation at the top node as unrooted.

Rooting a tree is usually done by specifying an *outgroup*. In the simplest case, this is a single leaf. The root is then placed in such a way that one of its children is the outgroup, while the other child is the rest of the tree (sometimes known as the *ingroup*). Consider the following primate tree, `simiiformes-wrong`:

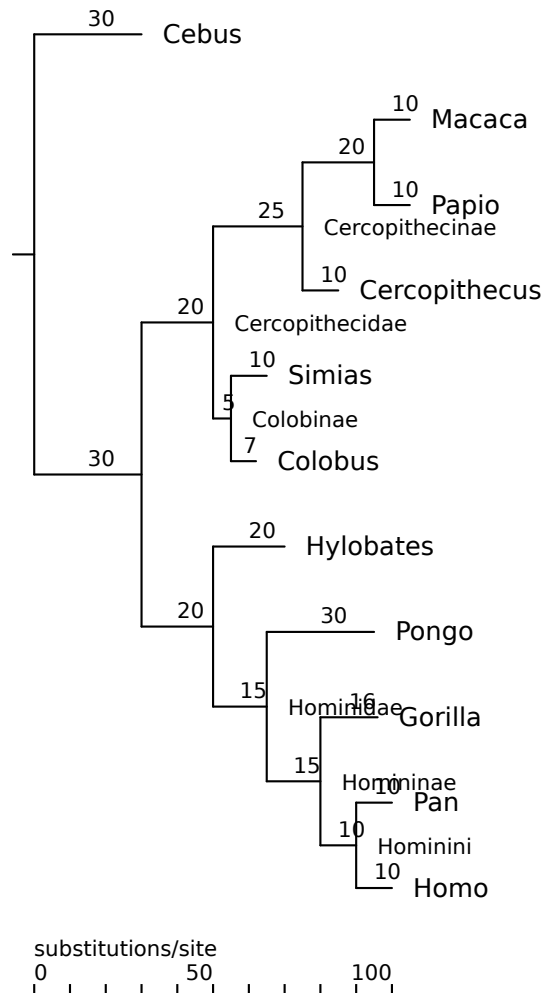


It is wrong because *Cebus*, which is a New World monkey (capuchin), should be the sister group of all the rest (Old World monkeys and apes, technically Catarrhini),

whereas it is shown as the sister group of the macaque-colobus family, Cercopithecoidea. We can correct this by re-rooting the tree using *Cebus* as outgroup:

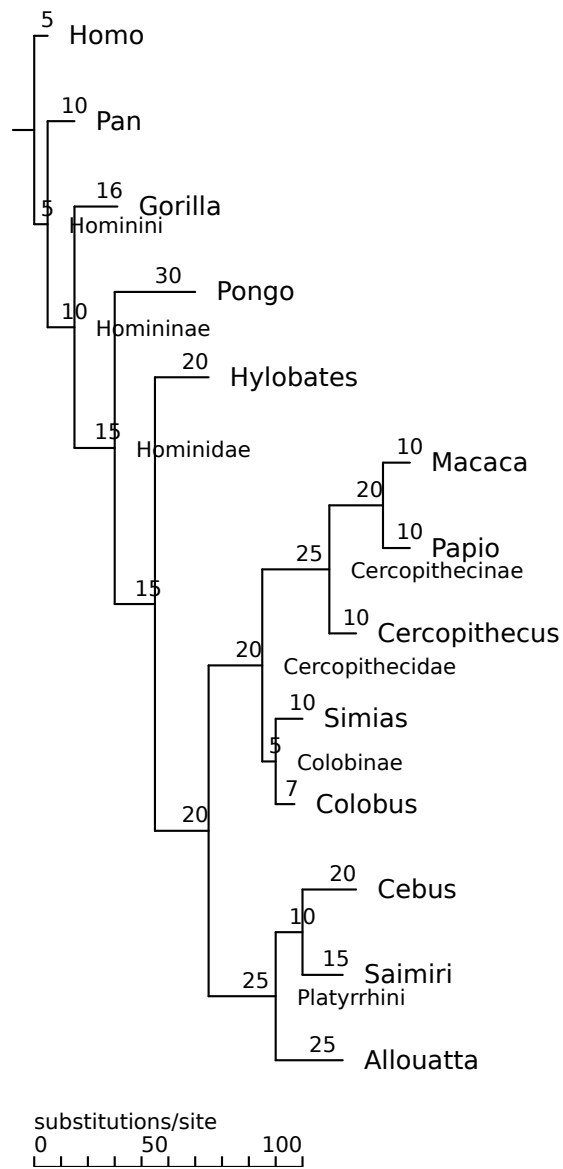
```
$ nw_reroot simiiformes_wrong Cebus | nw_display -s -
```

which produces:



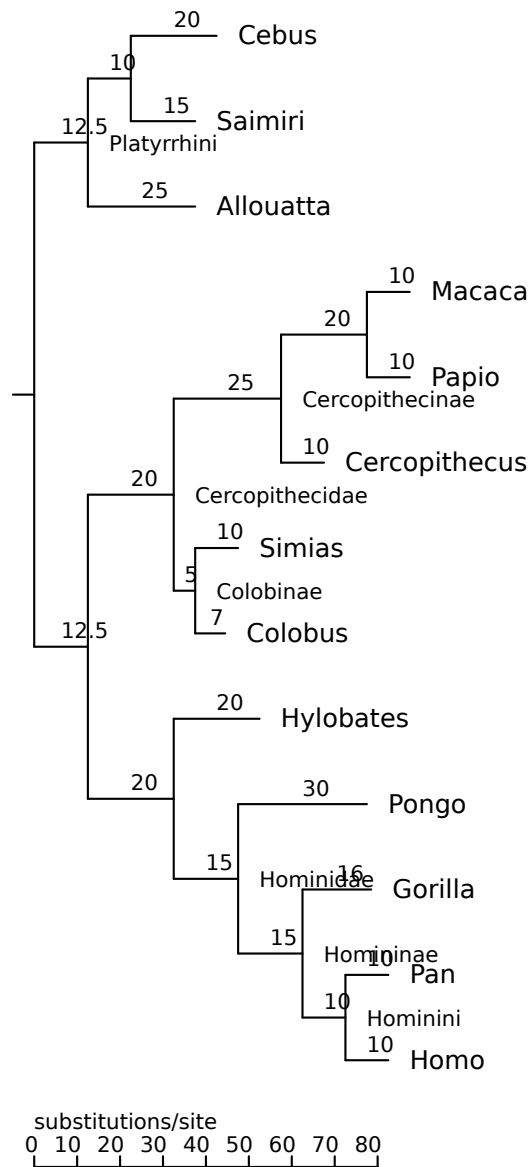
Now the tree is correct. Note that the root is placed in the middle of the ingroup-outgroup edge, and that the other branch lengths are conserved.

The outgroup does not need to be a single leaf. The following tree is wrong for the same reason as the one before, except that it has three New World monkey species instead of one, and they appear as a clade (Platyrrhini) in the wrong place:



We can correct this by specifying the New World monkey clade as outgroup:

```
$ nw_reroot simiiformes_wrong_3og Cebus Allouatta | nw_display -s -
```



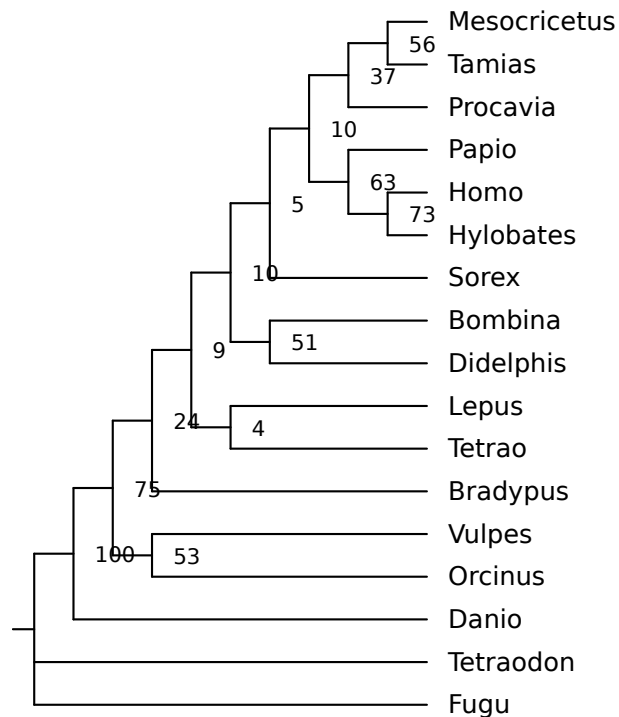
Note that I did not include all three New World monkeys, only *Cebus* and *Allouatta*. This is because it is always possible to define a clade using only two leaves. The result would be the same if I had included all three, though. You can use inner labels too, if there are any:

```
$ nw_reroot simiiformes_wrong_3og Platyrrhini
```

will reroot in the same way (not shown). Beware that inner labels are often used for support values (as in bootstrapping), which are generally not useful for defining clades.

2.2.1 Rerooting on the ingroup

Sometimes the desired species cannot be used for rooting, as their last common ancestor is the tree's root. For example, consider the following tree:



It is wrong because *Danio* (a ray-finned fish) is shown closer to tetrapods than to other ray-finned fishes (*Fugu* and *Tetraodon*). So we should reroot it, specifying that the fishes should form the outgroup. We could try this:

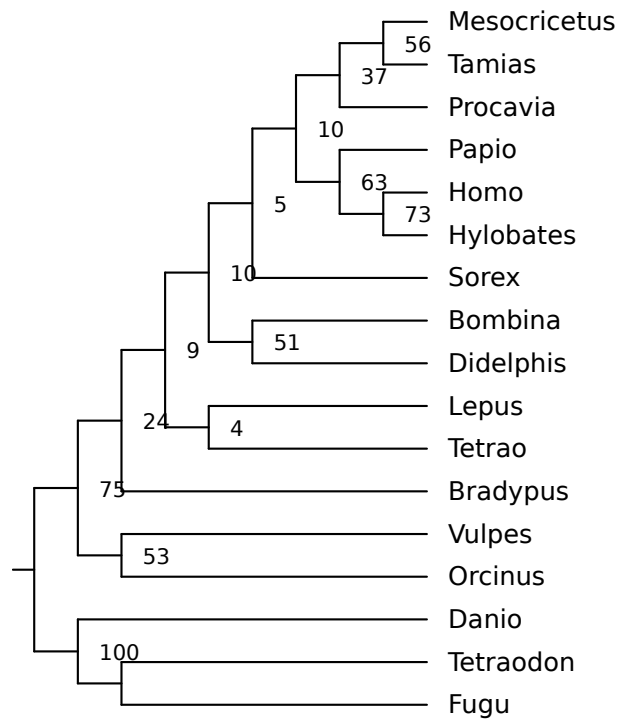
```
$ nw_reroot vrt1cg.nw Fugu Danio
```

But this will fail:

```
Outgroup's LCA is tree's root - cannot reroot. Try -l.
```

This fails because the last common ancestor of the two pufferfish is the root itself. The workaround in this case is to try the ingroup. This is done by passing option `-l` ("lax"), along with *all* species in the outgroup (this is because `nw_reroot` finds the ingroup by complementing the outgroup):

```
$ nw_reroot -l vrt1cg.nw Danio Fugu Tetraodon | nw_display -s -v 20 -
```

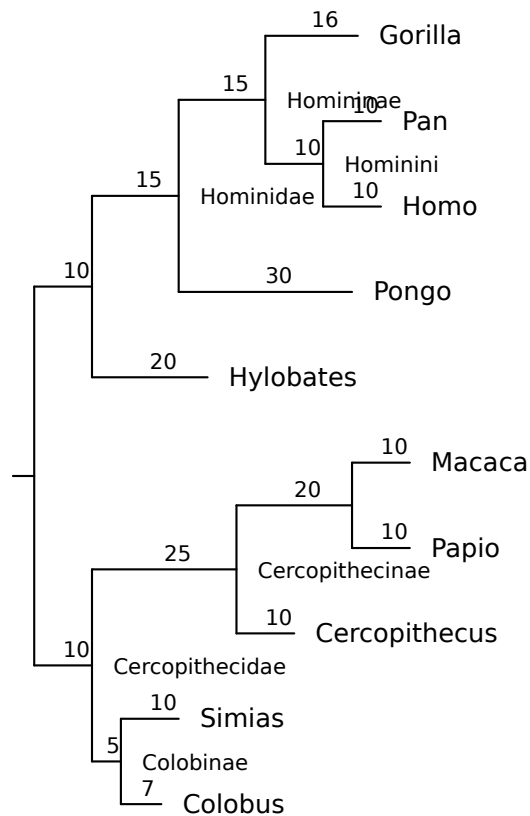


To repeat: all outgroup labels were passed, not just the two normally needed to find the last common ancestor – since, precisely, we can't use the LCA.

2.3 Extracting Subtrees

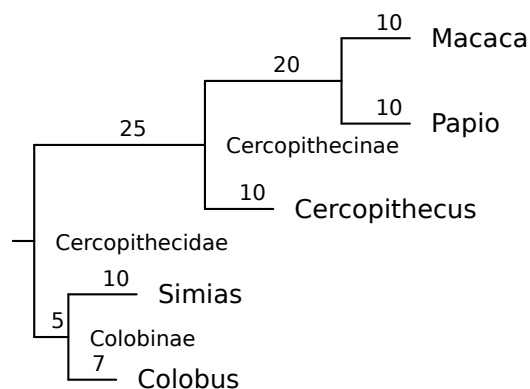
You can extract a clade (AKA subtree) from a tree with `nw_clade`. As usual, a clade is specified by a number of node labels, of which the program finds the last common ancestor, which unequivocally determines the clade (see Appendix A). We'll use the catarrhinian tree again for these examples:

```
$ nw_display -sS catarrhini
```



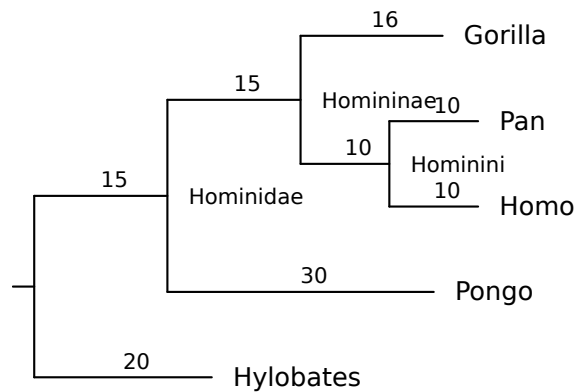
In the simplest case, the clade you want to extract has its own, unique label. This is the case of *Cercopithecoidea*, so you can extract the whole cercopithecoid subtree (Old World monkeys) using just that label:

```
$ nw_clade catarrhini Cercopithecoidea | nw_display -sS -
```



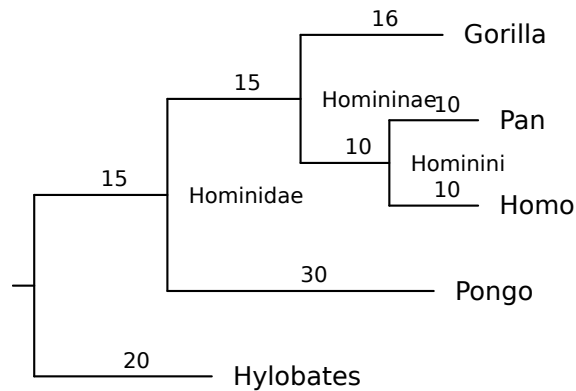
Now suppose I want to extract the apes subtree. These are the Hominidae (“great apes”) plus the gibbons (*Hylobates*). But the corresponding node is unlabeled in our tree (it would be *Hominoidea*), so we need to specify (at least) two descendants:

```
$ nw_clade catarrhini Gorilla Hylobates | nw_display -sS -
```



The descendants do not have to be leaves: here I use Hominidae, an inner node, and the result is the same.

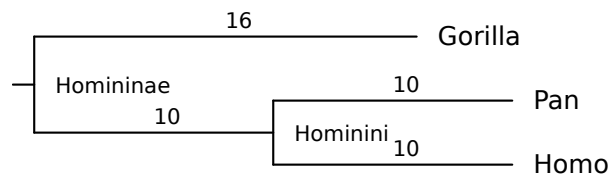
```
$ nw_clade catarrhini Hominidae Hylobates | nw_display -sS -
```



2.3.1 Monophyly

You can check if a set of leaves³ form a monophyletic group by passing option `-m`: `nw_clade` will report the subtree only if the LCA has no descendant leaf other than those specified. For example, we can ask if the African apes (humans, chimp, gorilla) form a monophyletic group:

```
$ nw_clade -m catarrhini Homo Gorilla Pan | nw_display -sS -v 30 -
```



Yes, they do – it's subfamily Homininae. On the other hand, the Asian apes (orangutan and gibbon) do not:

³In future versions I may extend this to inner nodes

```
$ nw_clade -m catarrhini Hylobates Pongo
```

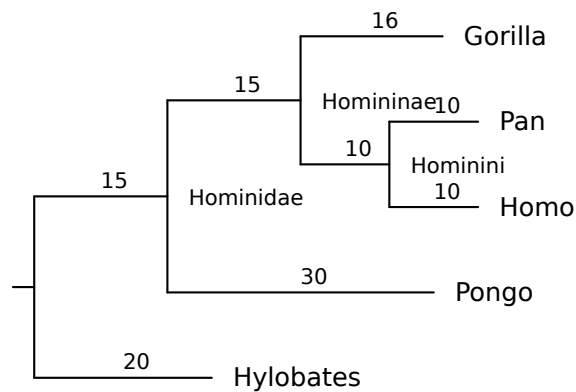
[no output]

Maybe early hominines split from orangs in South Asia before moving to Africa.

2.3.2 Context

You can ask for n levels above the clade by passing option `-c`:

```
$ nw_clade -c 2 catarrhini Gorilla Homo | nw_display -sS -
```

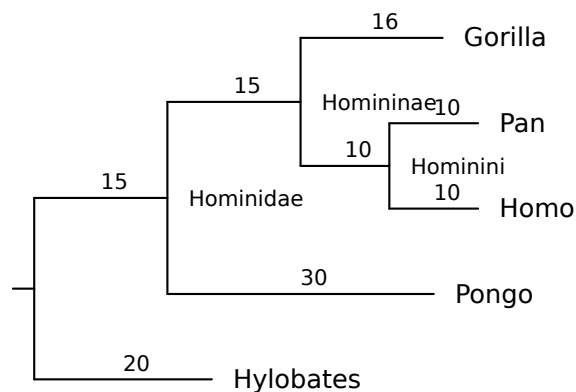


In this case, `nw_clade` computed the LCA of `Gorilla` and `Homo`, “climbed up” two levels, and output the subtree at that point. This is useful when you want to extract a clade with its nearest neighbor(s). I use this when I have several trees in a file and my clade’s nearest neighbors aren’t always the same.

2.3.3 Siblings

You can also ask for the siblings of the specified clade. What, for example, is the sister clade of the cercopithecoidea? Ask for `Cercopithecoidea` and pass option `-s`:

```
$ nw_clade -s catarrhini Cercopithecoidea | nw_display -sS -
```



Why, it's the good old apes, of course. I use this a lot when I want to get rid of the outgroup: specify the outgroup and pass `-s` - behold!, you have the ingroup.

Finally, although we are usually dealing with bifurcating trees, `-s` also applies to multifurcations: if a node has more than one sibling, `nw_clade` reports them all, in Newick order.

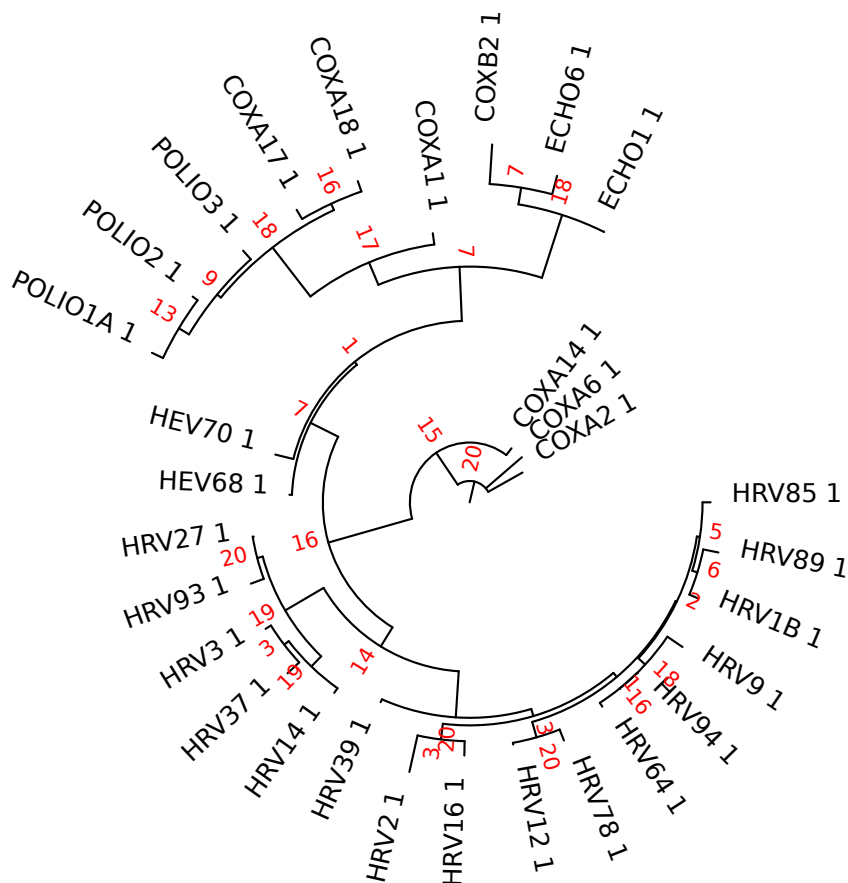
2.3.4 Limits

`nw_clade` assumes that node labels are unique. This should change in the future.

2.4 Computing Bootstrap Support

`nw_support` computes bootstrap support values from a target tree and a file of replicate trees. Say the target tree is in file `HRV.nw` and the replicates (20 of them) are in `HRV_20reps.nw`. You can attribute support values to the target tree like this:

```
$ nw_support HRV.nw HRV_20reps.nw \
| nw_display -sr -S -w 500 -i 'font-size:small;fill:red' -
```



In this case I have colored the support values red. Option `-p` uses percentages instead of absolute counts.

Notes

There are many tree-building programs that compute bootstrap support. For example, PhyML can do it, but for large tasks I typically have to distribute the replicates over several jobs (say, 100 jobs of 10 replicates each). I then collect all replicates files, concatenate them, and use `nw_support` to attribute the values to the target tree.

`nw_support` assumes rooted trees (it may as well, since Newick is implicitly rooted), and the target tree and replicates should be rooted the same way. Use `nw_reroot` to ensure this.

2.5 Retaining Topology

There are cases when one is more interested in the tree's structure than in the branch lengths, maybe because lengths are irrelevant or just because they are so short that they obscure the branching order. Consider the following tree, `vrt1.nw`:

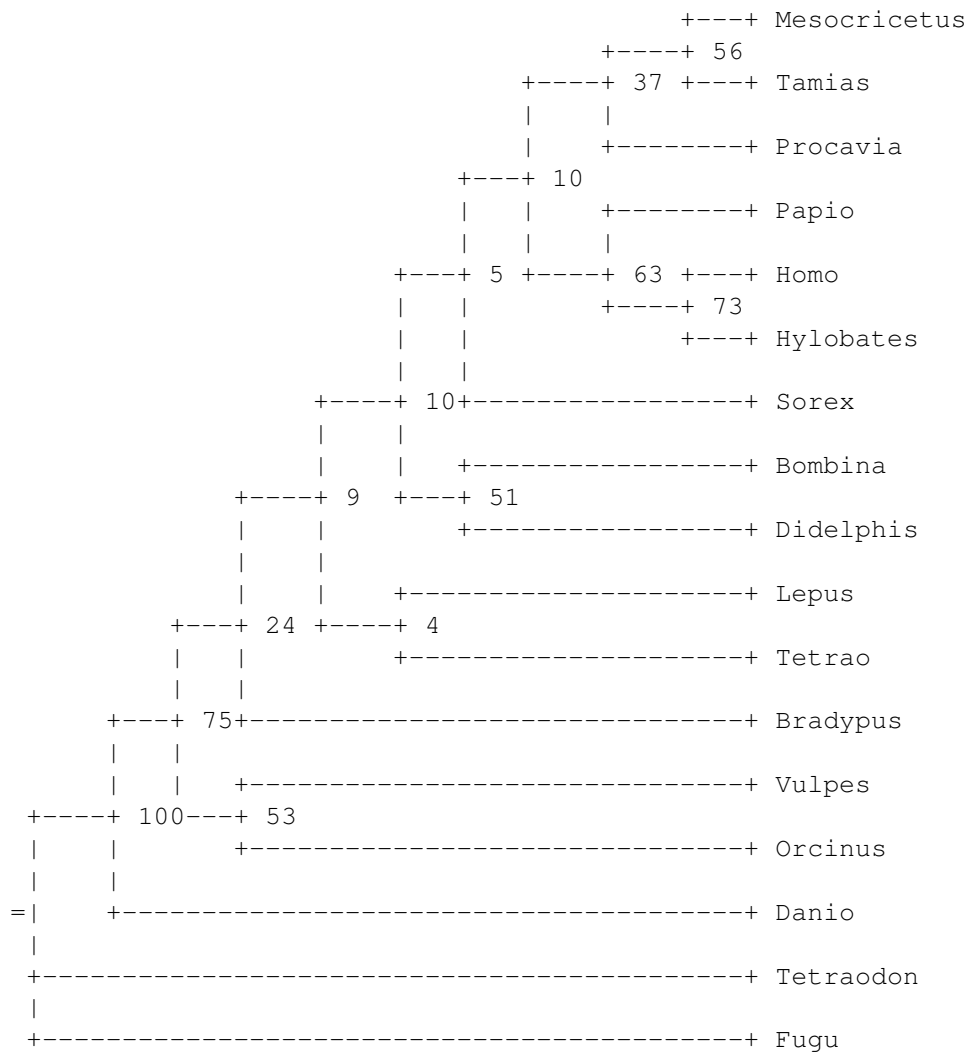
```

                ++ Mesocricetus
                | 56
            +++37amias
            ||
            |++ Procavia
            | 10
            | | Papio
            | |
            |-5 63mo
            | | 73
            | | Hylobates
            |
            ++ 10rex
            ||
            ||+-----+ Bombina
            |+9 51
            | +-+ Didelphis
            |
            |-+ Lepus
            | 24
            |-----+ Tetrao
            |
            +-----+75Bradypus
            |
            | +-+ Vulpes
            +-----+ 100 +-+ 53
            |
            | +-----+ Orcinus
            +-+ Danio
            |
            +---+ Tetraodon
            |
            +-----+ Fugu

```

Its structure is not evident, particularly in the upper half. This is because many branches are short in relation to the depth of the tree, so they are not well resolved. A better-resolved tree can be obtained by discarding branch lengths altogether:

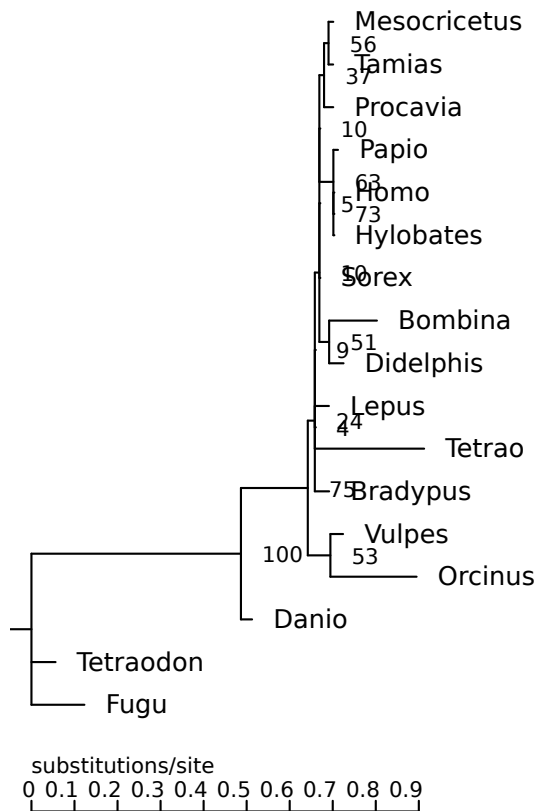
```
$ nw_topology vrt1.nw | nw_display -w 60 -
```



This effectively produces a *cladogram*, that is, a tree that represents ancestry relationships but not amounts of evolutionary change. The inner nodes are evenly spaced over the depth of the tree, and the leaves are aligned, so the branching order is more apparent.

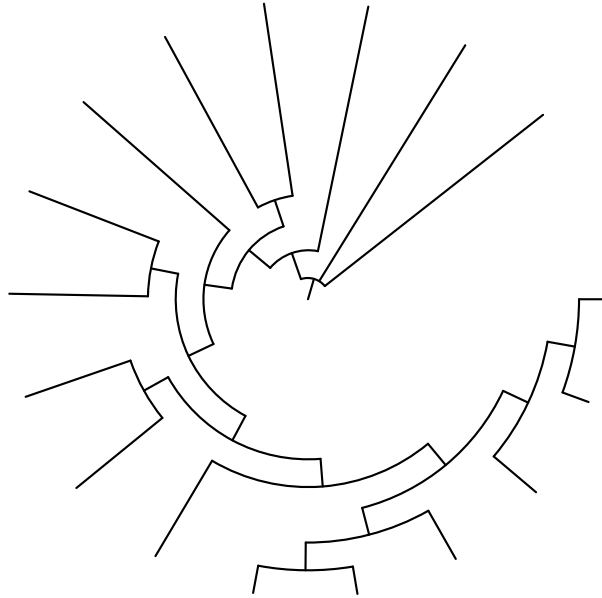
Of course, ASCII trees have low resolution in the first place, so I'll show both trees look in SVG. First the original:

```
$ nw_display -s -v 20 -b "opacity:0" vrt1.nw
```



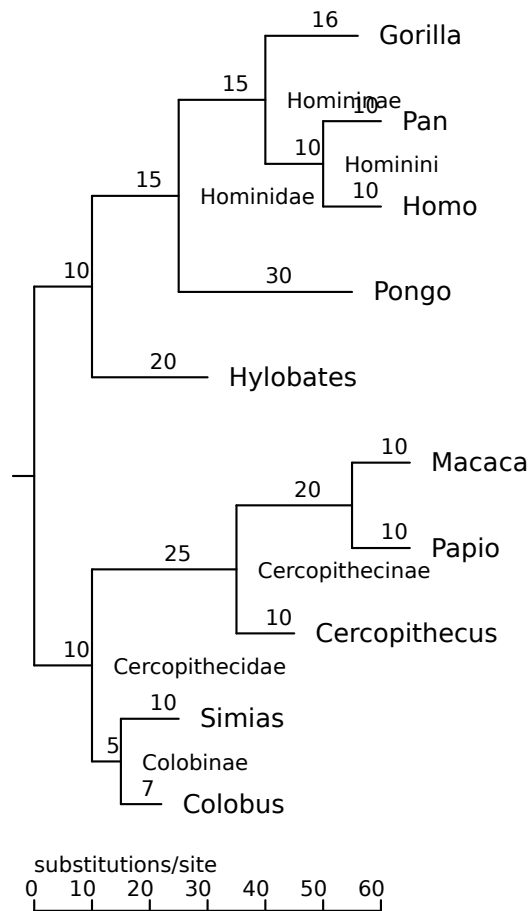
And now as a cladogram:

```
$ nw_topology vrt1.nw | nw_display -s -v20 -
```

2.6 Extracting Distances

`nw_distance` prints distances between nodes, in various ways. By default, it prints the distance from the root of the tree to each labeled leaf, in Newick order. Let's look at distances in the catarrhinian tree:



```
$ nw_distance catarrhini
```

```
56
60
60
55
30
65
65
45
25
22
```

This means that the distance from the root to Gorilla is 56, etc. The distances are in the same units as the tree's branch lengths – usually substitutions per site, but this is not specified in the tree itself. If the tree is a cladogram, the distances are expressed in numbers of ancestors. Option `-n` shows the labels:

```
$ nw_distance -n catarrhini
```

```

Gorilla 56
Pan      60
Homo     60
Pongo    55
Hylobates      30
Macaca  65
Papio    65
Cercopithecus 45
Simias   25
Colobus  22

```

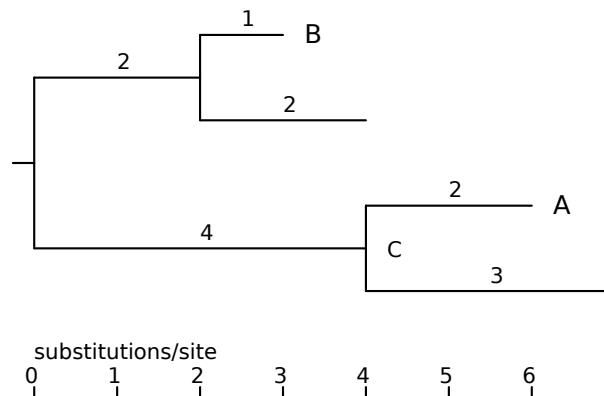
There are two main parameters to `nw_distance`: the *method* and the *selection*. The method determines how to compute the distance (from what node to what node), and the selection determines for which nodes the program is to compute distances. Let's look at examples.

2.6.1 Selection

In this section we will show the different selection types, using the default distance method (*i.e.*, from the tree's root – see below). The selection type is the argument to option `-s`. The nodes appear in the same order as in the Newick tree, except when they are specified on the command line (see below).

To illustrate the selection types, we need a tree that has both labeled and unlabeled leaves and inner nodes. Here it is

```
$ nw_display -s dist_sel_xpl.nw
```



We will use option `-n` to see the node labels.

All labeled leaves

The selection consists of all leaves with a label. This is the default, as leaves will mostly be labeled and we're generally more interested in leaves than inner nodes.

```
$ nw_distance -n dist_sel_xpl.nw
```

```

B      3
A      6

```

All labeled nodes

Option `-s l`. This takes all labeled nodes into account, whether they are leaves or inner nodes.

```
$ nw_distance -n -s l dist_sel_xpl.nw
```

```
B      3
A      6
C      4
```

All leaves

Option `-s f`. Selects all leaves, whether they are labeled or not.

```
$ nw_distance -n -s f dist_sel_xpl.nw
```

```
B      3
      4
A      6
      7
```

All inner nodes

Option `-s i`. Selects the inner nodes, labeled or not.

```
$ nw_distance -n -s i dist_sel_xpl.nw
```

```
      2
C      4
```

All nodes

Option `-s a`. All nodes are selected.

```
$ nw_distance -n -s a dist_sel_xpl.nw
```

```
B      3
      4
      2
A      6
      7
C      4
      0
```

Command line selection

The selection consists of the nodes whose labels are passed as arguments on the command line (after the file name). The distances are printed in the same order.

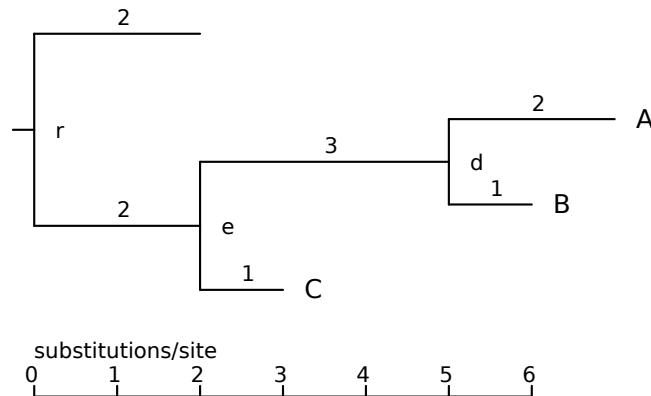
```
$ nw_distance -n dist_sel_xpl.nw A C
```

```
A      6
C      4
```

2.6.2 Methods

In this section we will take the default selection and vary the method. The method is passed as argument to option `-m`. I will also use an *ad hoc* tree to illustrate the methods:

```
$ nw_display -s dist_meth_xpl.nw
```



As explained above, the default selection consists of all labeled leaves – in our case, nodes A, B and C.

Distance from the tree's root

This is the default method: for each node in the selection, the program prints the distance from the tree's root to the node. This was shown above, so I won't repeat it here.

Distance from the last common ancestor

Option `-m l`. The program computes the LCA of all nodes in the selection (in our case, node e), and prints out the distance from that node to all nodes in the selection.

```
$ nw_distance -n -m l dist_meth_xpl.nw
```

```
A      5
B      4
C      1
```

Distance from the parent

Option `-m p`. The program prints the length of each selected node's parent branch.

```
$ nw_distance -n -m p dist_meth_xpl.nw
```

```
A      2
B      1
C      1
```

Matrix

Option `-m m`. Computes the pairwise distances between all nodes in the selection, and prints it out as a matrix.

```
$ nw_distance -n -m m dist_meth_xpl.nw
```

| | A | B | C |
|---|---|---|---|
| A | 0 | 3 | 6 |
| B | 3 | 0 | 5 |
| C | 6 | 5 | 0 |

2.6.3 Alternative formats

Option `-t` changes the output format. For matrix output, (`-m m`), the matrix is triangular.

```
$ nw_distance -t -m m dist_meth_xpl.nw
```

```
3
6      5
```

When labels are printed (option `-n`), the diagonal is shown

```
$ nw_distance -n -t -m m dist_meth_xpl.nw
```

| | | | |
|---|---|---|---|
| A | 0 | | |
| B | 3 | 0 | |
| C | 6 | 5 | 0 |

For all other formats, the values are printed in a line, separated by TABs.

```
$ nw_distance -n -t -m p dist_meth_xpl.nw
```

| | | |
|---|---|---|
| A | B | C |
| 2 | 1 | 1 |

2.7 Finding subtrees in other trees

`nw_match` tries to match a (typically smaller) "pattern" tree to one or more "target" tree(s). If the pattern matches the target, the target tree is printed. Intuitively, a pattern matches a target if one can superimpose it onto the target without "breaking" either. More accurately, the following happens (in both trees):

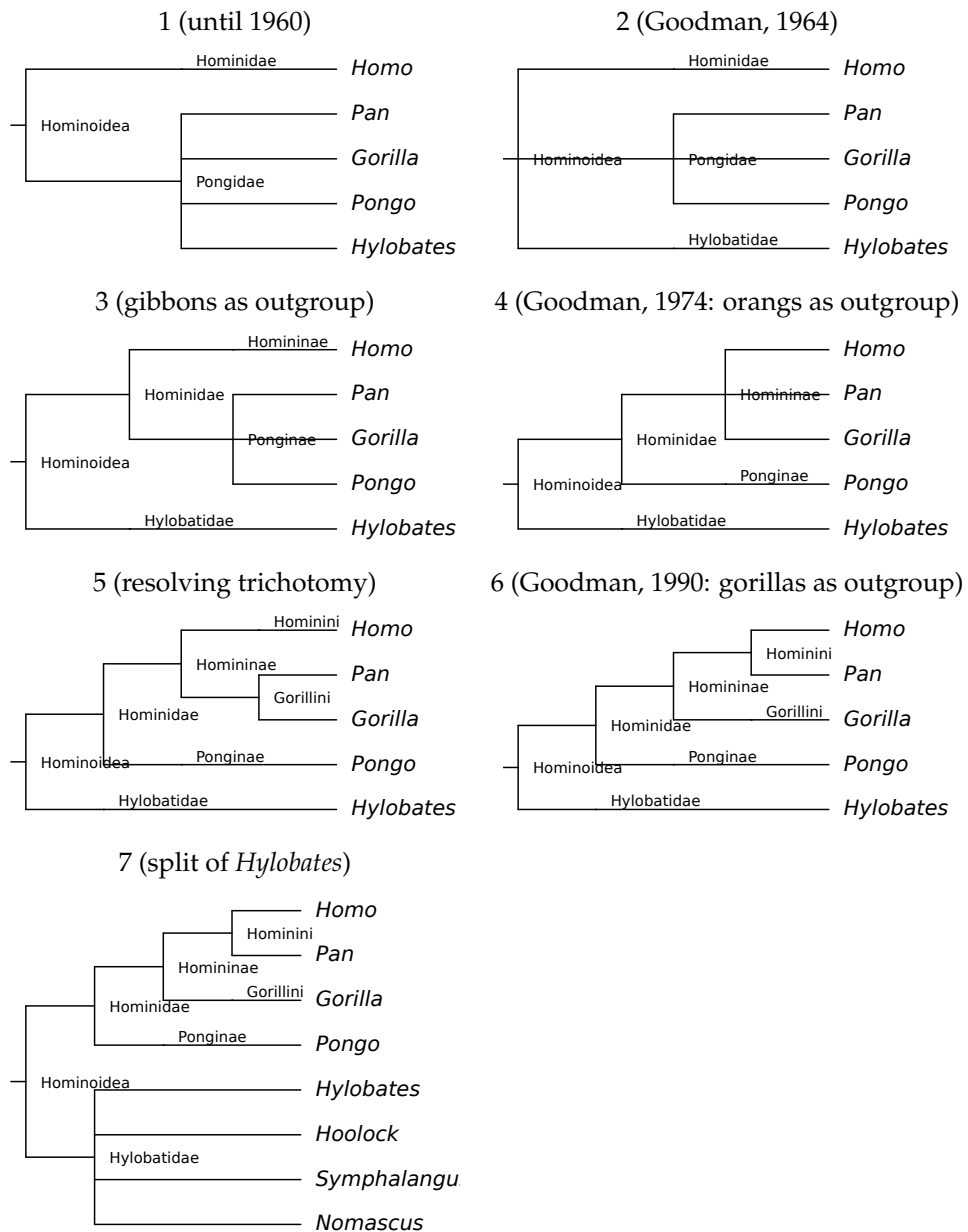
1. leaves with labels found in both trees are kept, the other ones are pruned
2. inner labels are discarded
3. both trees are ordered (as done by `nw_order`, see 2.14)
4. branch lengths are discarded

At this point, the modified pattern tree is compared to the modified target, and if the Newick strings are identical, the match is successful.

Example: finding trees with a specified subtree topology

File `hominoidea.nw` contains seven trees corresponding to successive theories about the phylogeny of apes (these were taken from <http://en.wikipedia.org/wiki/Hominoidea>). Let us see which of them group humans and chimpanzees as a sister clade of gorillas (which is the current hypothesis).

Here are small images of each of the trees in `hominoidea.nw`:



Trees #6 and #7 match our criterion, the rest do not. To look for matching trees in `hominoidea.nw`, we pass the pattern on the command line:

```
$ nw_match hominoidea.nw '(Gorilla, (Pan, Homo));' | nw_display -w 60 -
```

```

                                     +-----+ Homo
                                   +-----+ Hominini
                                +-----+ Homininae +-----+ Pan
                                |
                                |
+-----+ Hominidae +-----Gorillini+ Gorilla
|
|=| Hominoidea+-----Ponginae-----+ Pongo
|
+-----Hylobatidae-----+ Hylobates

                                     +-----+ Homo
                                   +-----+ Hominini
                                +-----+ Homininae+-----+ Pan
                                |
                                |
+-----+ Hominidae +-----Gorillini+ Gorilla
|
|
|
|=| Hominoidea-----+ Hylobates
|
|
|
+-----+ Hylobatidae
          +-----+ Hoolock
          +-----+ Symphalangus
          |
          +-----+ Nomascus

```

Note that only the pattern tree's topology matters: we would get the same results with pattern `((Homo, Pan), Gorilla);`, `((Pan, Homo), Gorilla);`, etc., but not with `((Gorilla, Pan), Homo);` (which would select trees #1, 2, 3, and 5. In future versions I might add an option for strict matching.

The behaviour of `nw_match` can be reversed by passing option `-v` (like `grep -v`): it will print trees that *do not* match the pattern. Finally, note that `nw_match` only works on leaf labels (for now), and assumes that labels are unique in both the pattern and the target tree.

2.8 Renaming nodes

Renaming nodes is the rather boring operation of changing a node's label. It can be done *e.g.* for the following reasons:

- building a higher-level tree (*i.e.*, a families tree from a tree of genera, etc)
- mapping one namespace into another (see 2.8.1)
- correcting wrong names

Renaming is done with `nw_rename`. This takes a *renaming map*, which is just a text file with the old and new names on the same line.

2.8.1 Breaking the 10-character limit in PHYLIP alignments

A technical hurdle with phylogenies is that some programs do not accept names longer than, say, 10 characters in the PHYLIP alignment. But of course, many scientific names

or sequence IDs are longer than that. One solution is to rename the sequences, before constructing the tree, using a numerical scheme, e.g., *Strongylocentrotus purpuratus* → ID_01, etc. This means we have an alignment of the following form:

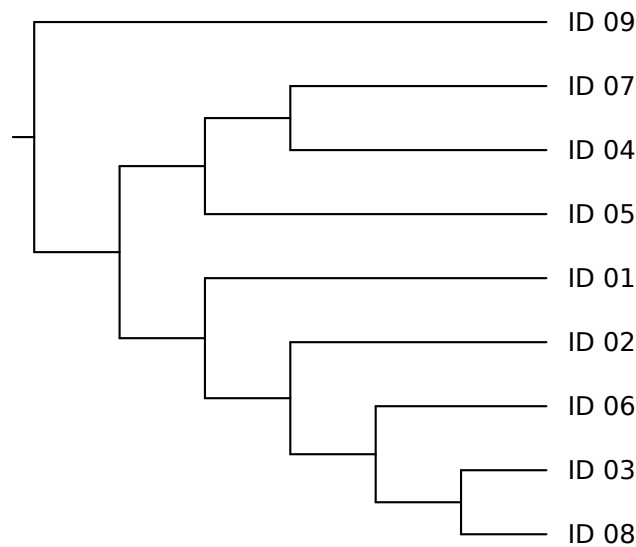
```
154 259
ID_01 PTTNSAPAL DAAETGHTSG ...
ID_02 SVSSHVPAL DAAETGHTSS ...
...
```

together with a renaming map, `id2longname.map`:

```
ID_01 Strongylocentrotus_purpuratus
ID_02 Harpagofututor_volsellorhinus
...
```

The alignment's IDs are now sufficiently short, and we can use it to make a tree. It will look something like this:

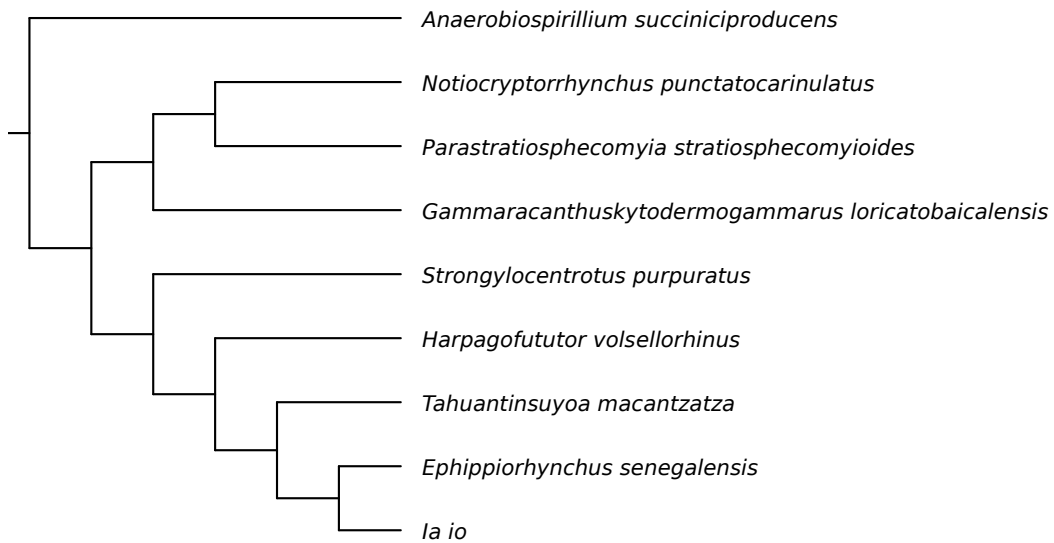
```
$ nw_display -s short_IDs.nw -v 30
```



Not very informative, huh? But we can put back the original, long names :

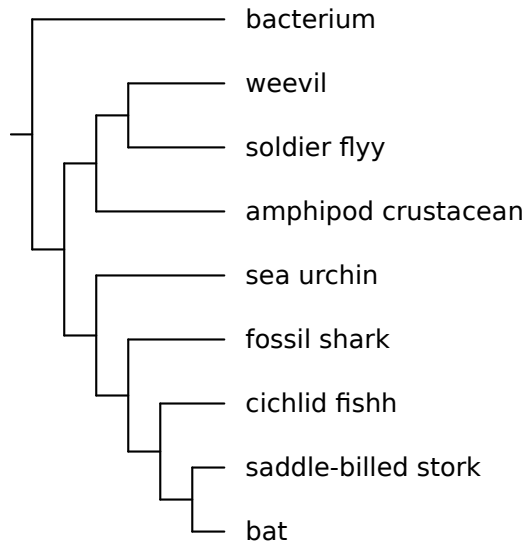
```
$ nw_rename short_IDs.nw id2longname.map \
| nw_display -s -l 'font-size:small;font-style:italic' -w 500 -v 30 -W 6 -
```

(option `-W` specifies the mean width of label characters, in pixels – use it when the default is wrong, as in this case with very long labels and small characters)



Now that's better... although exactly what these critters are might not be evident. Not to worry, I've made another map and I can rename the tree a second time on the fly:

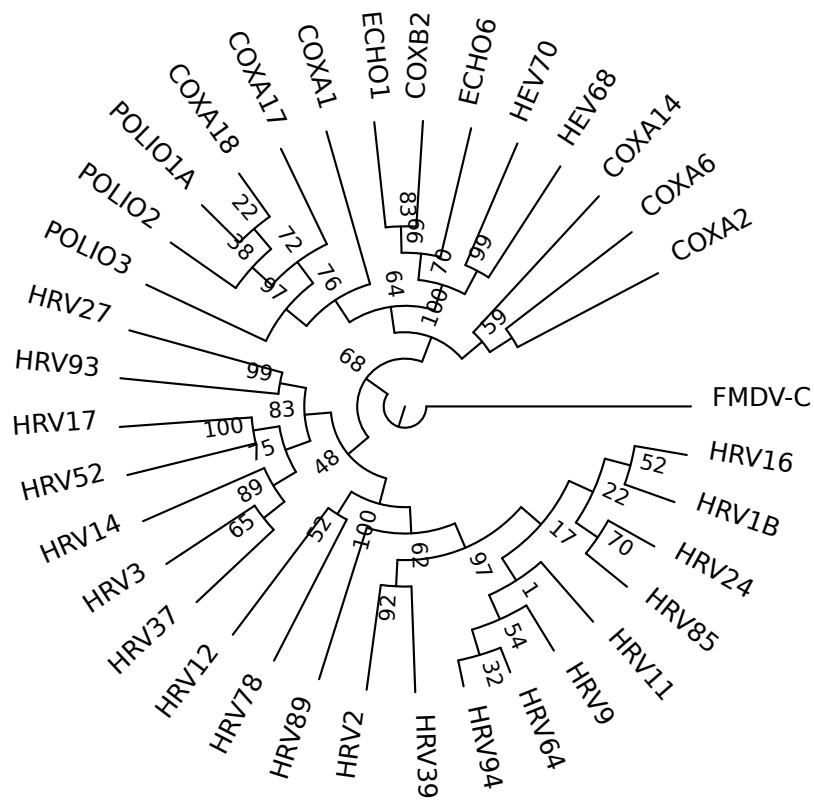
```
$ nw_rename short_IDs.nw id2longname.map \
| nw_rename - longname2english.map \
| nw_display -s -v 30 -W 10 -
```



2.8.2 Higher-rank trees

Here is a tree of a few dozen enterovirus and rhinovirus isolates. I show it as a cladogram (using `nw_topology`, see 2.5) because branch lengths do not matter here. I know that these isolates belong to three species in two genera: human rhinovirus A (`hrv-a`), human rhinovirus B (`hrv-b`), and enterovirus (`hev`).

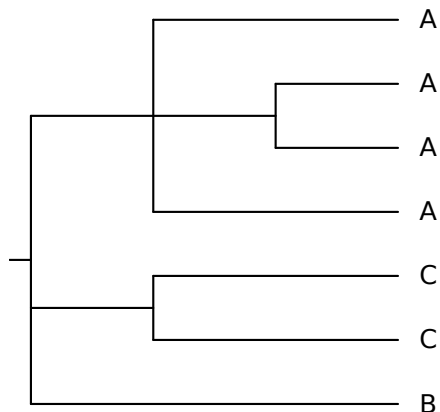
```
$ nw_topology HRV_FMDV.nw | nw_display -sr -w 400 -
```



I want to see if the tree correctly groups isolates of the same species together. So I use a renaming map that maps an isolate name to its species:

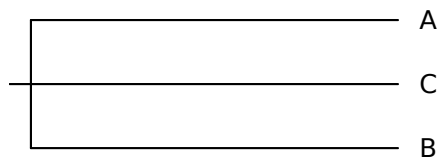
```
HRV16 HRV-A
HRV1B HRV-A
...
HRV37 HRV-B
HRV14 HRV-B
...
POLIO1A HEV
COXA17 HEV
```

```
$ nw_rename HRV_FMDV.nw HRV_FMDV.map \
| nw_topology - | nw_display -srS -w 400 -
```

it has a clade that consists only of A, another of only C, plus a B leaf. Condensing will replace those clades by an A and a C leaf, respectively:

```
$ nw_condense condense1.nw | nw_display -s -w 200 -v 30 -
```

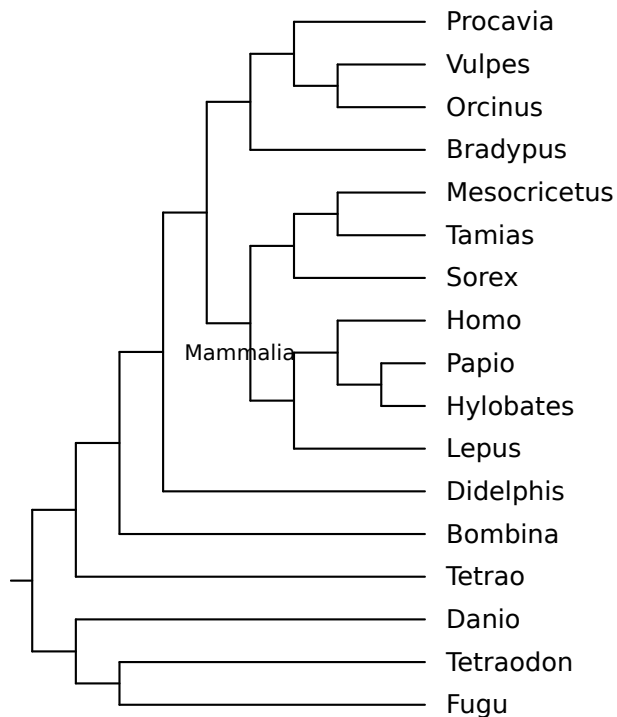


Now the A and B leaves stand for whole clades. The tree is simpler, but the information about the relationships of A, B and C is conserved, while the details of the A and C clades is not. A typical use of this is producing genus trees from species trees (or any higher-level tree from a lower-level one), or checking consistency with other data: For example condensing the virus tree of section 2.8.2 gives this:

The relationships between the species is now evident – as is the fact that the various isolates do cluster within species in the first place. This need not be the case, and renaming-then-condensing is a useful technique for checking this kind of consistency in a tree (see 3.1 for more examples).

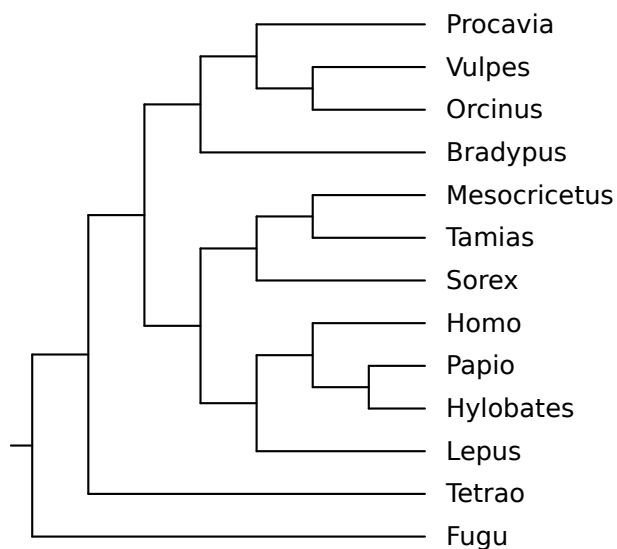
2.10 Pruning

Pruning is simply removing arbitrary nodes. Say you have the following tree:



and say you only need a subset of the species, perhaps because you want to compare this tree to another tree with fewer species. Specifically, let's say you don't need to show *Tetraodon*, *Danio*, *Bombina*, and *Didelphis*. You just pass those labels to `nw_prune`:

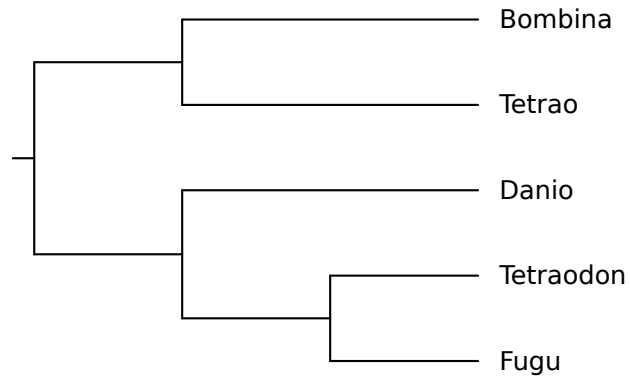
```
$ nw_prune vrt2_top.nw Tetraodon Danio Bombina Didelphis \
| nw_display -s -v 20 -
```



Note that each label is removed individually. The discarding of *Didelphis* is the cause of the disappearance of the node labeled *Mammalia*.

You can also discard internal nodes, if they are labeled (in future versions it will be possible to discard a clade by specifying descendants, just like `nw_clade`). For example, you can discard the whole mammalian clade like this:

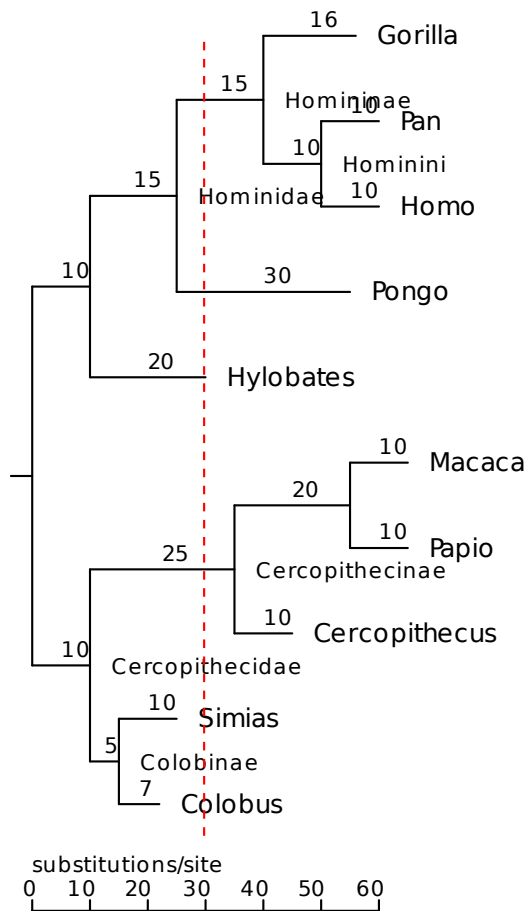
```
$ nw_prune vrt2_top.nw Mammalia | nw_display -s -
```



By the way, *Tetrao* and *Tetraodon* are not the same thing, the first is a bird (grouse), the second is a pufferfish.

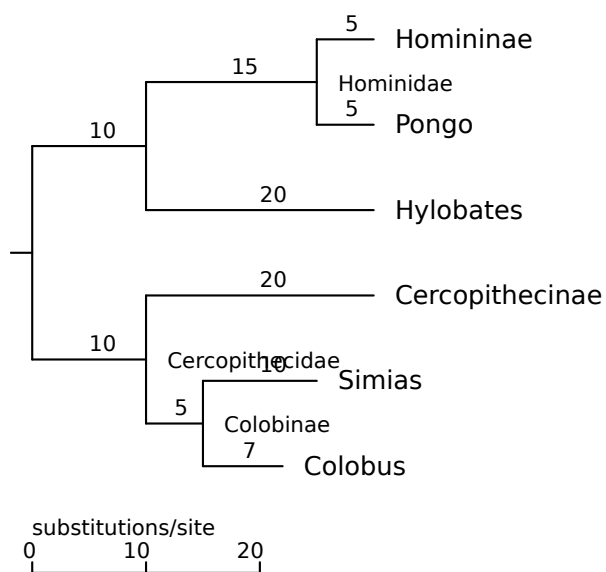
2.11 Trimming trees

Trimming a tree means cutting the nodes whose depth is larger than a specified threshold. Here is what will happen if I cut the catarrhini tree at depth 30:



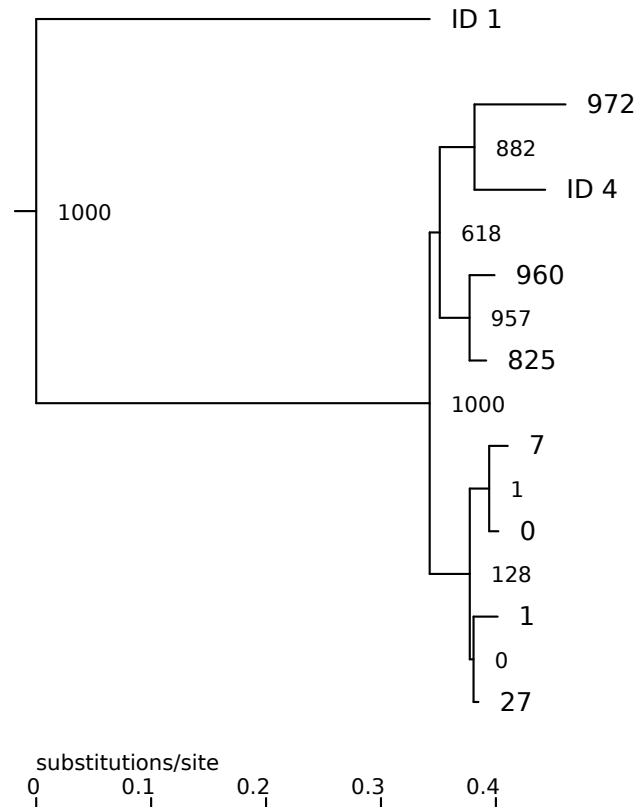
The tree will be "cut" on the red line, and everything right of it will be discarded:

```
$ nw_trim catarrhini 30 | nw_display -s -
```



By default, depth is expressed in branch length units – usually substitutions per site. By passing the `-a` switch, it is measured in number of ancestors, instead. Here are the first four levels of a huge tree (it has more than 1000 leaves):

```
$ nw_trim -a big.rn.nw 4 | nw_display -s -b 'opacity:0' -
```



The leaves with labels of the form `ID *` are also leaves in the original tree, the other leaves are former inner nodes whose children got trimmed. Their labels are the (absolute) bootstrap support values of those nodes. Note that the branch lengths are conserved. It is apparent that the ingroup's lower half has very poor support. This would be harder to see without trimming the tree, due to its huge size.

Trimming cladograms

By definition, cladograms do not have branch lengths, so you need to express depth in numbers of ancestors, and thus you want to pass `-a`.

2.12 Indenting

`nw_indent` reformats Newick on several lines, with one node per line, nodes of the same depth in the same column, and children nodes to the right of their parent. This shows the structure more clearly than the compact form, but since whitespace is ignored in the Newick format⁴, the indented form is still valid. For example, this is a tree in compact form, in file `falconiformes`:

⁴except between quotes

```
(Pandion:7, ((Accipiter:1,Buteo:1):1, (Aquila:1,Haliaeetus:2):1):2,
(Milvus:2,Elanus:3):2):3,Sagittarius:5, ((Micrastur:1,Falco:1):3,
(Polyborus:2,Milvago:1):2):2);
```

And this is the same tree, indented:

```
$ nw_indent falconiformes
```

```
(
  Pandion:7,
  (
    (
      (
        Accipiter:1,
        Buteo:1
      ):1,
      (
        Aquila:1,
        Haliaeetus:2
      ):1
    ):2,
    (
      Milvus:2,
      Elanus:3
    ):2
  ):3,
  Sagittarius:5,
  (
    (
      Micrastur:1,
      Falco:1
    ):3,
    (
      Polyborus:2,
      Milvago:1
    ):2
  ):2
);
```

The structure is much more clear, it is also relatively easy to edit manually in a text editor - while still being valid Newick.

Another advantage of indenting is that it is resistant to certain errors which would cause `nw_display` to fail.⁵ For example, there is an error in this tree:

```
(Pandion:7, ((Buteo:1,Aquila:1,Haliaeetus:2):2, (Milvus:2,
Elanus:3):2):3,Sagittarius:5((Micrastur:1,Falco:1):3,
(Polyborus:2,Milvago:1):2):2);
```

⁵This is because indenting is a purely lexical process, hence it does not need a syntactically correct tree.

yet it is hard to spot, and trying `nw.display` won't help as it will abort with a parse error. With `nw.indent`, however, you can at least look at the tree:

```
(
  Pandion:7,
  (
    (
      Buteo:1,
      Aquila:1,
      Haliaeetus:2
    ):2,
    (
      Milvus:2,
      Elanus:3
    ):2
  ):3,
  Sagittarius:5
  (
    (
      Micrastur:1,
      Falco:1
    ):3,
    (
      Polyborus:2,
      Milvago:1
    ):2
  ):2
);
```

While the error is not exactly obvious, you can at least view the Newick. It turns out there is a comma missing after `Sagittarius:5`.

The indentation can be varied by supplying a string (option `-t`) that will be used instead of the default (which is two spaces). If you want to indent by four spaces instead of two, you could say this:

```
$ nw_indent -t '    ' accipitridae

(
  (
    Buteo:1,
    Aquila:1,
    Haliaeetus:2
  ):2,
  (
    Milvus:2,
    Elanus:3
  ):2
):3;
```

Option `-t` can also be used to highlight indentation:

```
$ nw_indent -t '| ' accipitridae

(
| (
| | Buteo:1,
| | Aquila:1,
| | Haliaeetus:2
| ):2,
| (
| | Milvus:2,
| | Elanus:3
| ):2
):3;
```

Now the indentation levels are easier to see, but at the expense of the tree no longer being valid Newick.

Finally, option `-c` ("compact") does the reverse: it removes all indentation and produces a compact tree. You can use this when you want to produce a compact Newick file after editing. For example, using Vim, after loading a Newick tree I do

```
gg!}nw_indent -
```

to indent the file, then I edit it, then compact it again:

```
gg!}nw_indent -c -
```

2.13 Extracting Labels

To get a list of all labels in a tree, use `nw_labels`:

```
$ nw_labels catarrhini  
  
Gorilla  
Pan  
Homo  
Hominini  
Homininae  
Pongo  
Hominidae  
Hylobates  
Macaca  
Papio  
Cercopithecus  
Cercopithecinae  
Simias  
Colobus  
Colobinae  
Cercopithecidae
```

The labels are printed out in Newick order. To get rid of internal labels, use `-I`:

```
$ nw_labels -I catarrhini  
  
Gorilla  
Pan  
Homo  
Pongo  
Hylobates  
Macaca  
Papio  
Cercopithecus  
Simias  
Colobus
```

Likewise, you can use `-L` to get rid of leaf labels, and with `-t` the labels are printed on a single line, separated by tabs (here the line is folded due to lack of space).

```
$ nw_labels -tL catarrhini  
  
Hominini          Homininae          Hominidae          Cercopithecinae Colobinae  
Cercopithecidae
```

2.13.1 Counting Leaves in a Tree

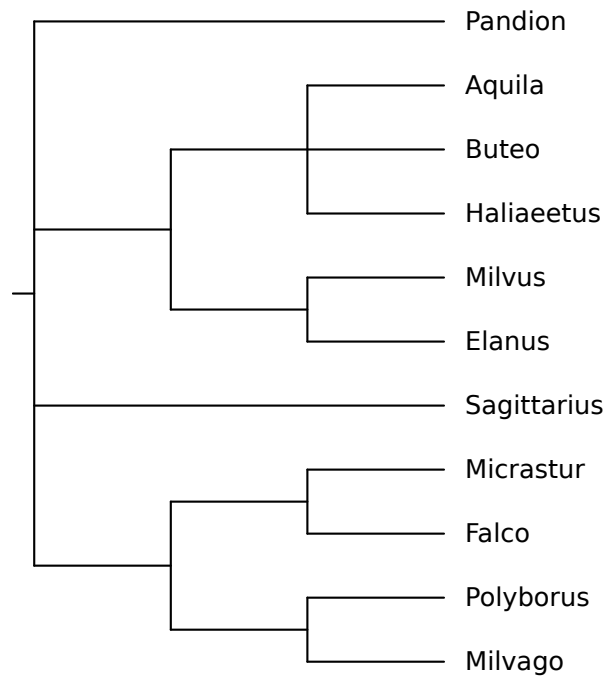
A simple application of `nw_labels` is a leaf count (assuming each leaf is labeled - Newick does not require labels):

```
$ nw_labels -I catarrhini | wc -l  
  
10
```

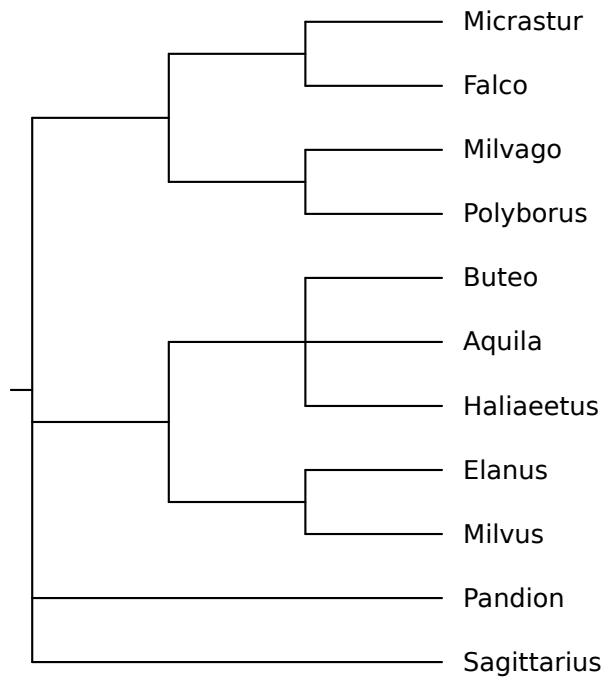
2.14 Ordering Nodes

Two trees that differ only by the order of children nodes within the parent convey the same biological information, even if the text (Newick) and graphical representations differ. For example, files `falconiformes_1` and `falconiformes_2` are different, and they yield different images:

```
$ nw_display -sS -v 30 falconiformes_1
```



```
$ nw_display -sS -v 30 falconiformes_2
```



But do they represent different phylogenies? In other words, do they differ by more than just the ordering of nodes? To check this, we pass them to `nw_order` and use `diff` to compare the results⁶:

```
$ nw_order falconiformes_1 > falconiformes_1.ord.nw ; \
nw_order falconiformes_2 > falconiformes_2.ord.nw ; \
diff -s falconiformes_1.ord.nw falconiformes_2.ord.nw
```

```
Files falconiformes_1.ord.nw and falconiformes_2.ord.nw are identical
```

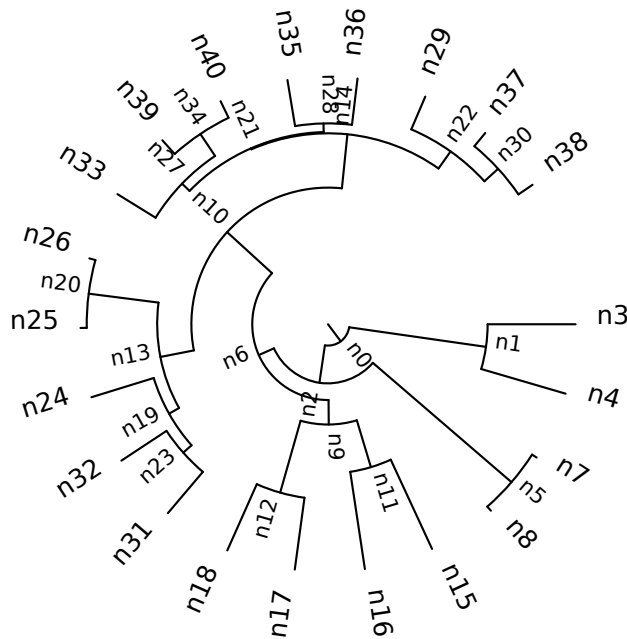
So, after ordering, the trees are the same: they tell the same biological story. Note that these trees are cladograms. If you have trees with branch lengths, this approach will only work if the lengths are identical, which may or may not be what you want. You can get rid of the branch lengths using `nw_topology` (see 2.5).

2.15 Generating Random Trees

`nw_gen` generates clock-like random trees, with exponentially distributed branch lengths. Nodes are sequentially labeled.

```
$ nw_gen -s 0.123 | nw_display -sSr -
```

⁶One could also compute a checksum using `md5sum`, etc



Here I pass option `-s`, whose argument is the pseudorandom number generator's seed, so that I get the same tree every time I produce this document. Normally, you will not use it if you want a different tree every time. Other options are `-d`, which specifies the tree's depth, and `-l`, which sets the average branch length.

I use random trees to test the other applications, and also as a kind of null model to test to what extent a real tree departs from the null hypothesis.

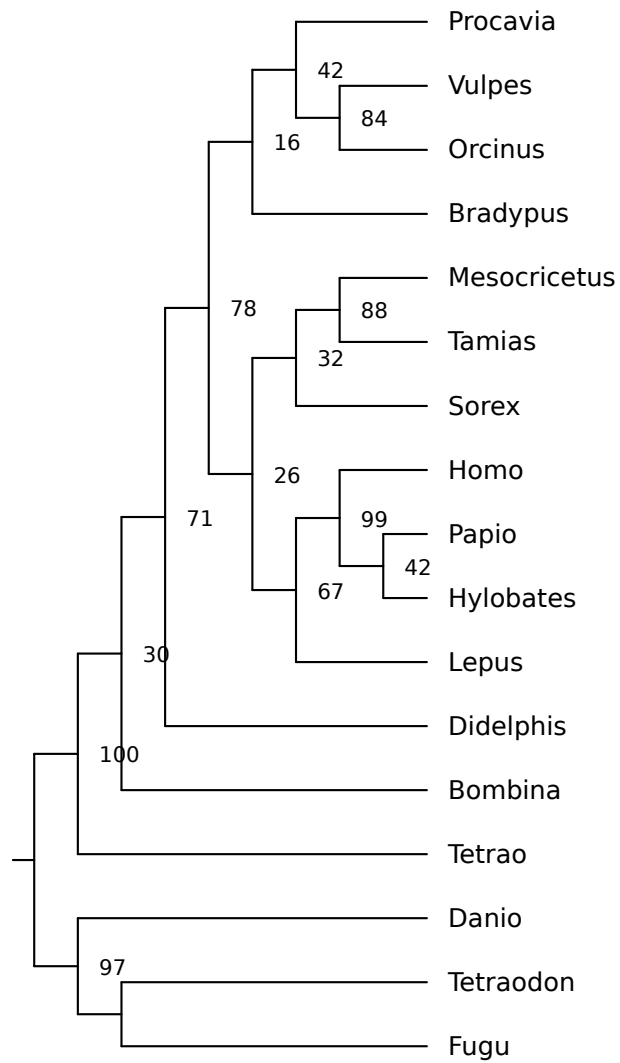
2.16 Stream editing

`nw_ed` is one of the more experimental programs in the Newick Utilities package. It was inspired by UNIX utilities like `sed` and `awk`, which perform an action on the parts of input (usually lines) that meet some condition.

`nw_ed` iterates over the nodes in a specific order, and for each node it evaluates a logical expression provided by the user. If the expression is true, `nw_ed` performs a user-specified action. By default, the (possibly modified) tree is printed at the end of the run.

Let's look at an example before we jump into the details. Here is a tree of vertebrate genera, showing support values:

```
$ nw_display -s -v 30 vrt2cg.nw
```



Let's extract all well-supported clades, using a support value of 95% or more as the criterion for being well-supported:

```
$ nw_ed -n vrt2cg.nw 'b >= 95' s | nw_display -w 65 -
```


Addresses

Currently, addresses are logical expressions involving node properties, and the action is performed on the nodes for which the expression is true. They are composed of numbers, logical operators, and node functions.

The functions have one-letter names, to keep expressions short (after all, they are passed on the command line). There are two types, numeric and boolean.

| name | type | meaning |
|---------|---------|--|
| a | numeric | number of ancestors of node |
| b | numeric | node's support value (or zero) |
| d | numeric | node's depth (distance to root) |
| i | boolean | true iff node is strictly internal (i.e., not root!) |
| l (ell) | boolean | true iff node is a leaf |
| r | boolean | true iff node is the root |

The logical and relational operators work as expected, here is the list, in order of precedence, from tightest to loosest-binding. Anyway, you can use parentheses to override precedence, so don't worry.

| symbol | operator |
|--------|--------------------------|
| ! | logical negation |
| == | equality |
| != | inequality |
| < | greater than |
| > | lesser than |
| >= | greater than or equal to |
| <= | lesser than or equal to |
| & | logical and |
| | logical or |

Here are a few examples:

| expression | selects: |
|---------------|--|
| l | all leaves |
| l & a <= 3 | leaves with 3 ancestors or less |
| i & (b >= 95) | internal nodes with support greater than 95% |
| i & (b < 50) | unsupported nodes (less than 50%) |
| !r | all nodes except the root |

Actions

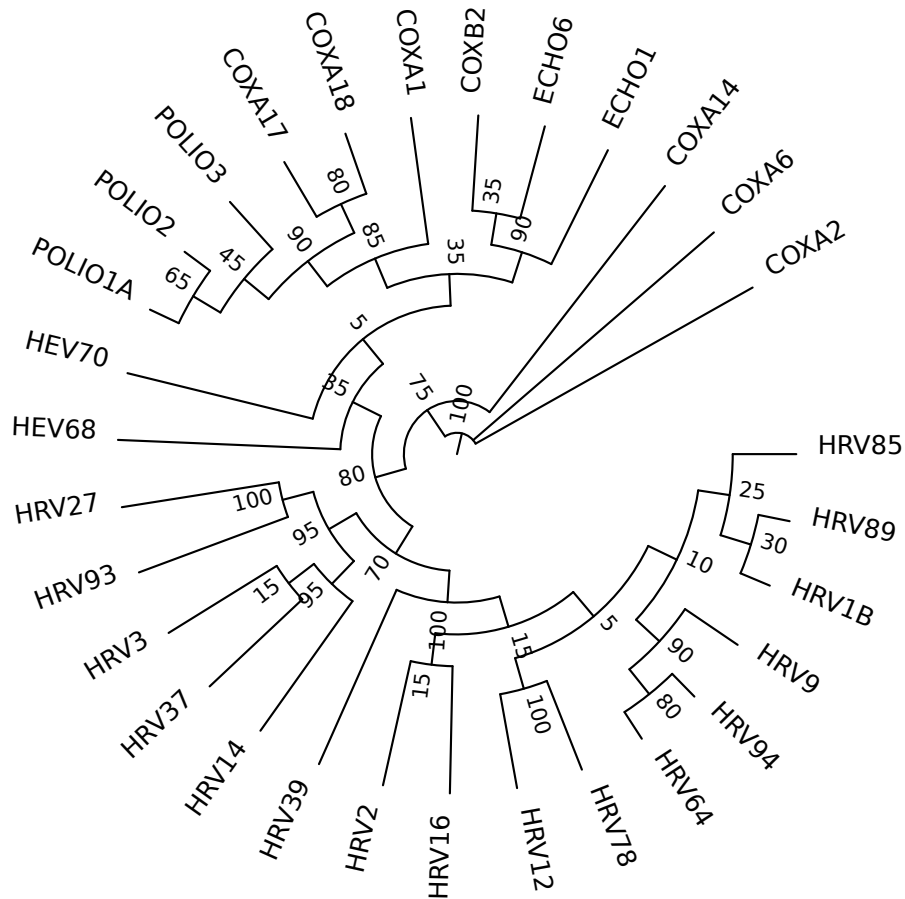
The actions are also coded by a single letter, for the same reason. For now, the following are implemented:

| code | effect | modifies tree? |
|------|---------------------------------------|----------------|
| d | delete the node (and all descendants) | yes |
| l | print the node's label | no |
| o | splice out the node | yes |
| s | print the subtree rooted at the node | no |

`nw.ed` is somewhat experimental; it is also the only program that is not deliberately "orthogonal" to the rest, that is, it can emulate some of the functionality of other utilities.

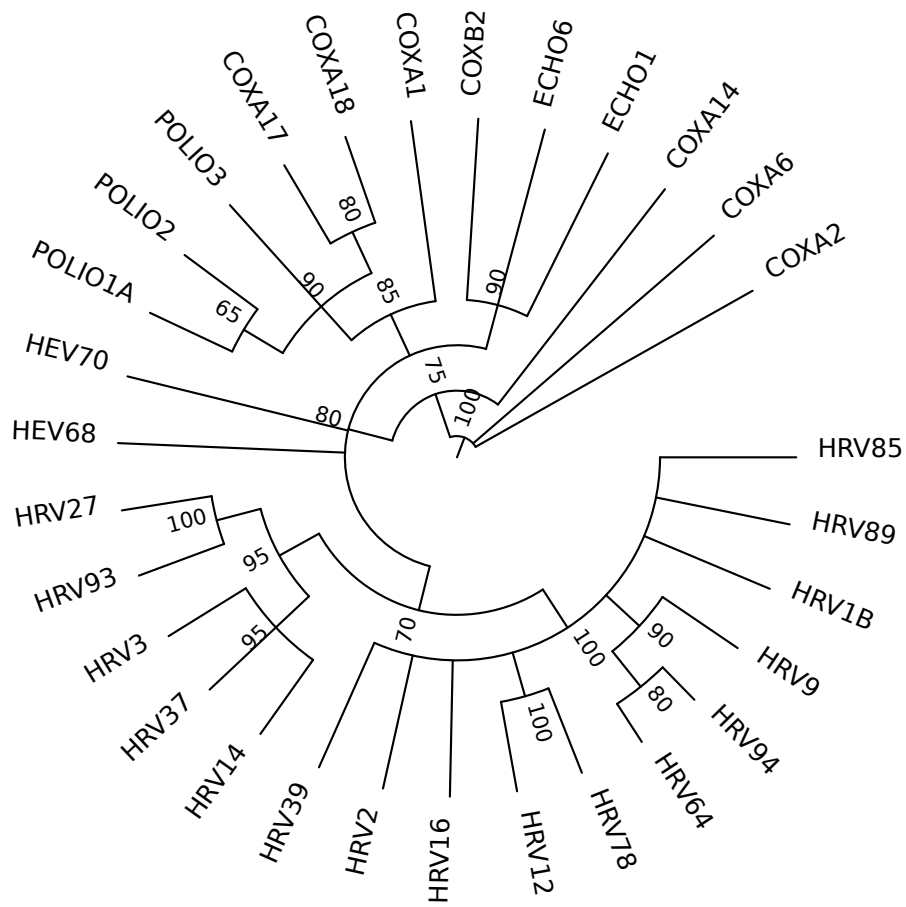
2.16.1 Opening Poorly-supported Nodes

When a node has low support, it may be better to splice it out from the tree, reflecting uncertainty about the true topology. Consider the following tree, `HRV_cg.nw`:



The inner node labels represent bootstrap support, as a percentage of replicates. As we can see, some nodes are much better supported than others. For example, the `(COXB2, ECHO6)` node (top of the figure) has a support of only 35%, and in the lower right of the graph there are many nodes with even poorer support. Let's "open" the nodes with less than 50% support. This means that those nodes will be spliced out, and their children attached to their "grandparent":

```
$ nw_ed HRV_cg.nw 'i & b < 50' o | nw_display -sr -w 450 -
```



Now COXB2 and ECHO6 are siblings of ECHO1, forming a node with 90% support. What this means is that the original tree strongly supports that these three species form a clade, but is much less clear about the relationships *within* the clade. Opening the nodes makes this fact clear by creating multifurcations. Likewise, the lower right of the figure is now occupied by a highly multifurcating (8 children) but perfectly supported (100%) node, none of whose descendants has less than 80% support.

Chapter 3

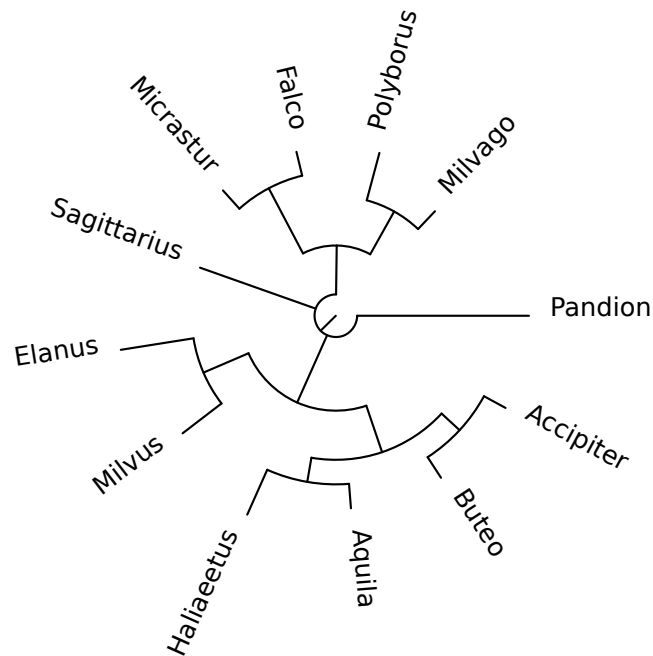
Advanced Tasks

The tasks presented in this chapter are more complex than that of chapter 2, and generally involve many Newick Utilities as well as other programs.

3.1 Checking Consistency with other Data

3.1.1 By condensing

One can check the consistency of a tree with respect to additional information by renaming and condensing. For example, I have the following tree of Falconiformes (diurnal raptors: eagles, falcons, etc):



Now I also have the following information about the family to which each genus belongs:

| Genus | Family |
|--------------------|---------------|
| <i>Accipiter</i> | Accipitridae |
| <i>Aquila</i> | Accipitridae |
| <i>Buteo</i> | Accipitridae |
| <i>Elanus</i> | Accipitridae |
| <i>Falco</i> | Falconidae |
| <i>Haliaeetus</i> | Accipitridae |
| <i>Micrastur</i> | Falconidae |
| <i>Milvago</i> | Falconidae |
| <i>Milvus</i> | Accipitridae |
| <i>Pandion</i> | Pandionidae |
| <i>Polyborus</i> | Falconidae |
| <i>Sagittarius</i> | Sagittariidae |

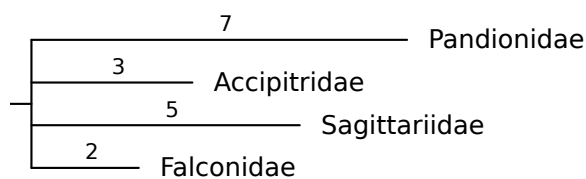
Let's see if the tree is consistent with this information. If it is, all families should form clades. To check this, I will rename each leaf by replacing the genus name by the family name, then condense the tree. If the original tree is consistent, the final tree should have one leaf per family.

First, I create a renaming map (see 2.8) based on the above information (here are the first three lines):

```
$ head -3 falc_map
Accipiter      Accipitridae
Buteo  Accipitridae
Aquila  Accipitridae
```

Then I use it to rename, and then I condense the tree:

```
$ nw_rename falconiformes falc_map \
| nw_condense - | nw_display -s -S -v 20 -
```



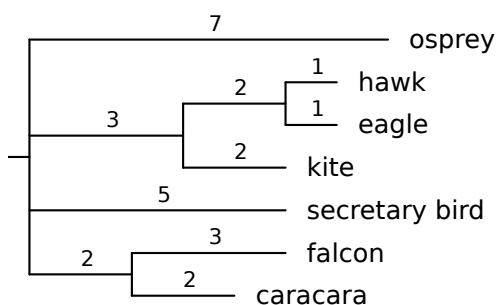
As we can see, there is one leaf per family, so the above information is consistent with the tree.

Let's see if common English names are also consistent with the tree. Here is one possible table of vernacular names of the raptor genera:

| Genus | English name |
|--------------------|----------------------------------|
| <i>Accipiter</i> | hawk (sparrowhawk, goshawk, etc) |
| <i>Aquila</i> | eagle |
| <i>Buteo</i> | hawk |
| <i>Elanus</i> | kite |
| <i>Falco</i> | falcon |
| <i>Haliaeetus</i> | eagle (sea eagle) |
| <i>Micrastur</i> | falcon (forest falcon) |
| <i>Milvago</i> | caracara |
| <i>Milvus</i> | kite |
| <i>Pandion</i> | osprey |
| <i>Polyborus</i> | caracara |
| <i>Sagittarius</i> | secretary bird |

And here is the corresponding tree:

```
$ nw_rename falconiformes falconiformes_vern1.map \
| nw_condense - | nw_display -s -S -v 20 -
```



So the above common names are consistent with the tree. However, some species have many common names. For example, the *Buteo* hawks are often called "buzzards" (in Europe), and two species of falcons have been called "hawks" (in North America): the peregrine falcon (*Falco peregrinus*) was called the "duck hawk", and the American kestrel (*Falco sparverius*) was called the "sparrow hawk".¹ If we map these common names to the tree and condense, we get this:

```
$ nw_rename falconiformes falconiformes_vern2.map \
| nw_condense - | nw_display -s -S -v 20 -
```

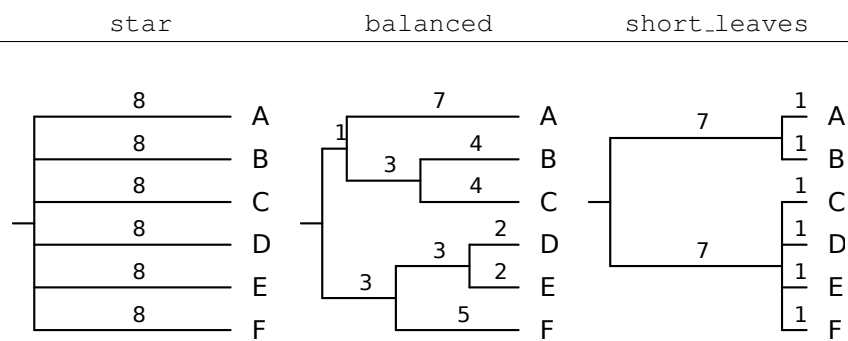
¹This is confusing because there are true hawks called "sparrow hawks", e.g. the Eurasian sparrow hawk *Accipiter nisus*. To add to the confusion, the specific name *sparverius* looks like the English word "sparrow", and also resembles the common name of *Accipiter nisus* in many other languages: *épervier* (fr), *Sperber* (de), *sparviere* (it). Oh well. Let's not drop scientific names just yet!

until position 450 or so, the query sequence's nearest relatives (in terms of substitutions/site) are HRV-36 and HRV-89. After that point, it is HRV-67. This suggests that there is a recombination breakpoint near position 450.

The script uses `nw_reroot` to reroot the trees on the outgroup, `nw_clade` and `nw_labels` to get the labels of the ingroup, `nw_distance` to extract the distance between the query and the other sequences, as well as the usual `sed`, `grep`, etc. The plot is done with `gnuplot`.

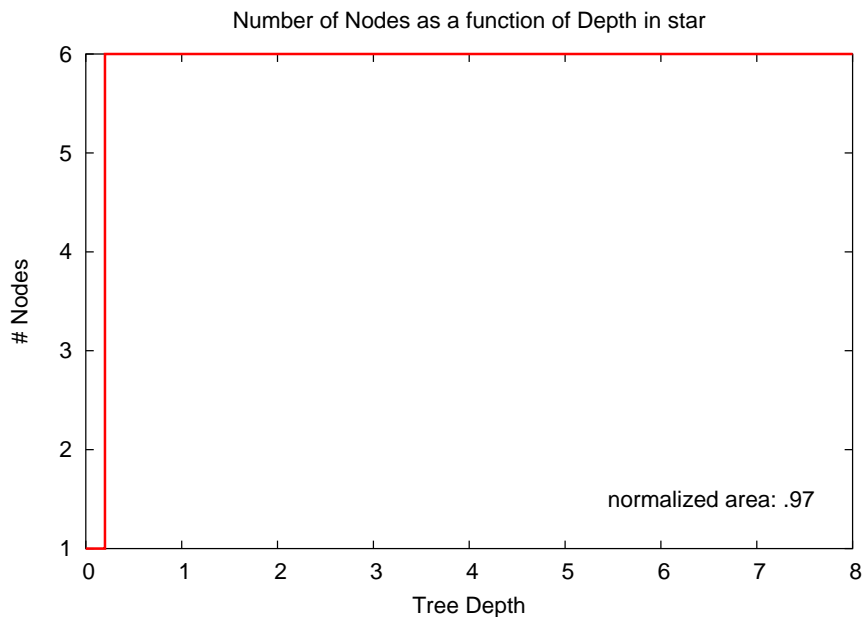
3.3 Number of nodes vs. Tree Depth

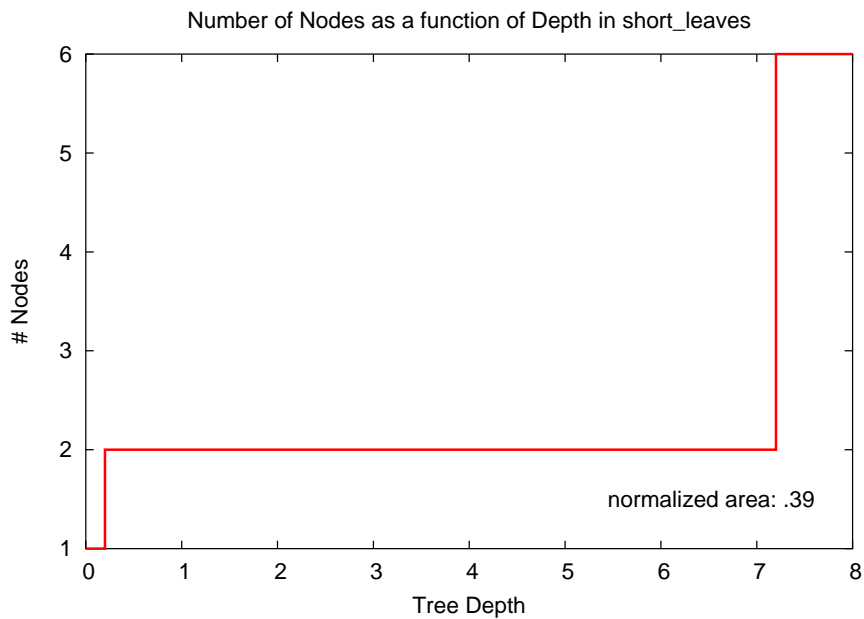
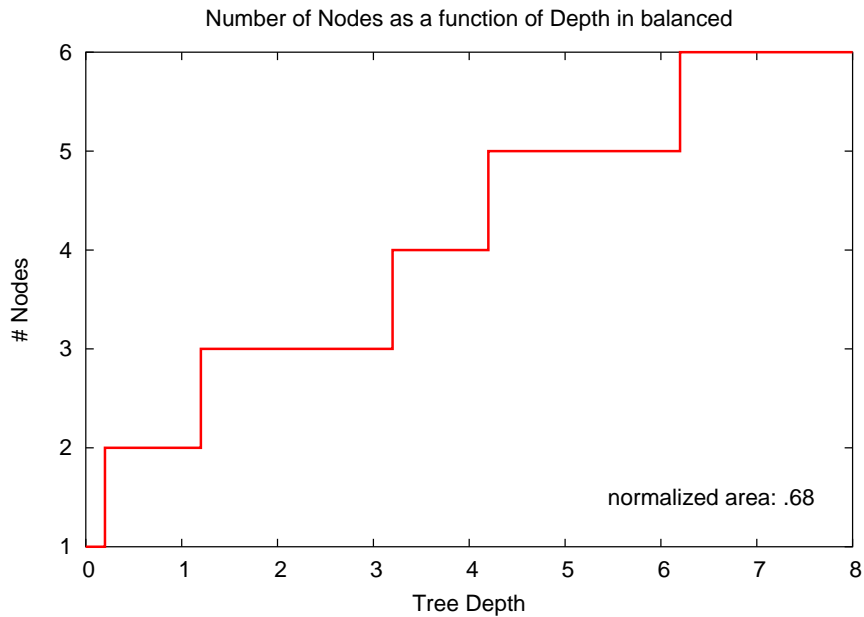
A simple measure of a tree's shape can be obtained by computing the number of nodes as a function of depth. Consider the following trees:



they have the same depth and the same number of leaves. But their shapes are very different, and they tell different biological stories. If we assume that they are clock-like (i.e., that the mutation rate is constant over the whole tree), `star` shows an early radiation, `short_leaves` shows two stable lineages ending in recent branching, while `balanced` shows branching spread over time.

The nodes-vs-depth graphs for these trees are as follows:





The graphs show the (normalized) area under the curve: it is close to 1 for star-like trees, close to 0 for trees with very short leaves, and intermediary for more balanced trees.

The images were made with the `bootscan.sh` script (in directory `src`), in the following way:

```
$ nodes_vs_clades.sh star 40
```

where 40 is just the sampling density (how many points to take on the x axis). The script uses `nw_distance` to get the tree's depth, `nw_ed` to sample the number of nodes at a given depth, and `nw_indent` to count the leaves, plus the usual `awk` and friends. The plot is done with `gnuplot`.

Chapter 4

Python Bindings

Although the Newick Utilities are primarily intended for shell use, it is also possible to use (some of) their functionalities in Python programs¹. While the Newick Utilities are written in C, the `ctypes` module² makes it easy to access them from Python, and the distribution contains a module, `newick_utils.py`, that provides an object-oriented interface to the underlying C code.

Let's say we want to add a utility that prints simple statistics about trees, like the number of nodes, the depth, whether it is a cladogram or a phylogram, etc. We will call it `nw_info.py`, and we'll pass it a Newick file on standard input, so the usage will be something like:

```
$ nw_info.py < data/catarrhini
```

The overall structure of this program is simple: iteratively read all the input trees, and do something with each of them:

```
1 from newick_utils import *
2
3 for tree in Tree.parse_newick_input():
4     pass # process tree here!
```

Line 1 imports definitions from the `newick_utils.py` module. Line 3 is the main loop: the `Tree.parse_newick_input` reads standard input and yields an instance of class `Tree` for each Newick string. We can now work with it, using methods of class `Tree` or adding our own:

```
1 #!/usr/bin/env python
2
3 from newick_utils import *
4
5 def count_polytomies(tree):
6     count = 0
7     for node in tree.get_nodes():
8         if node.children_count() > 2:
9             count += 1
10    return count
11
```

¹Or C programs, for that matter: the app code is well separated from the library code, which can serve as an API

²Available in Python 2.5 and up

```

12 for tree in Tree.parse_newick_input():
13     type = tree.get_type()
14     if type == 'Phylogram':
15         # also sets nodes' depths
16         depth = tree.get_depth()
17     else:
18         depth = None
19     print 'Type:', type
20     print '#Nodes:', len(list(tree.get_nodes()))
21     print '┆┆#leaves:', tree.get_leaf_count()
22     print '#Polytomies:', count_polytomies(tree)
23     print "Depth:", depth

```

When we run the program, we get:

```
$ nw_info.py < catarrhini
```

```

Type: Phylogram
#Nodes: 19
┆┆#leaves: 10
#Polytomies: 0
Depth: 65.0

```

As you can see, most of the work is done by methods called on the `tree` object, such as `get_leaf_count` which (surprise!) returns the number of leaves of a tree. But since there is no method for counting polytomies, we added our own function, `count_polytomies`, which takes a `Tree` object as argument.

As another example, a simple implementation of `nw_reroot` is found in `src/nw_reroot.py`. Its performance is almost as good as `nw_reroot`'s, especially on large input.

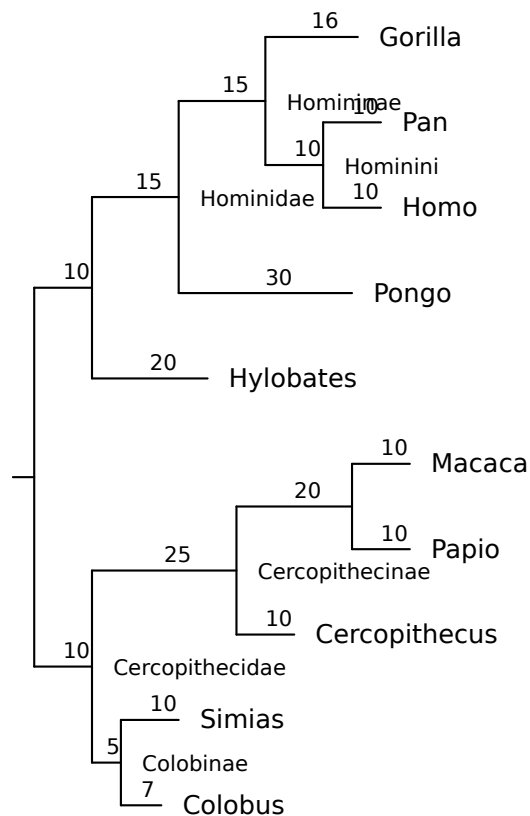
4.1 API Documentation

Detailed information about all classes and methods available for accessing the Newick Utilities library from Python is found in file `newick_utils.py`. Note that the library must be installed on your system, which means that you must compile from source.

Appendix A

Defining Clades by their Descendants

When you need to specify a clade using the Newick Utilities, you either give the label of the clade's root, or the labels of (some of) its descendants. Since inner nodes rarely have labels (or worse, have unuseable labels like bootstrap support values), you will often need to specify clades by their descendants. Consider the following tree:



Suppose we want to specify the Hominoidea clade - the apes. It is the clade that contains *Homo*, *Pan* (chimps), *Gorilla*, *Pongo* (orangutan), and *Hylobates* (gibbons).

The clade is not labeled in this tree, but this list of labels defines it without ambiguity. In fact, we can define it unambiguously using just *Hylobates* and *Homo* - or *Hylobates* and any other label. The point is that *you never need more than two labels to unambiguously define a clade*.

You cannot choose any two nodes, however: the condition is that the last common ancestor of the two nodes be the root of the desired clade. For instance, if you used *Pongo* instead of *Hylobates*, you would define the Hominidae clade, leaving out the gibbons.

A.1 Why not just use node numbers?

Some applications attribute numbers to all inner nodes and allow users to specify clades by referring to this number. Such a scheme is not workable when one has many input trees, however, because there is no guarantee that the same clade (assuming it is present) will have the same number in different trees.

Appendix B

Newick order

There are many ways of visiting a tree. One can start from the root and proceed to the leaves, or the other way around. One can visit a node before its children (if any), or after them. Unless specified otherwise, the Newick Utilities process trees as they appear in the Newick data. That is, for tree $(A, (B, C) d) e$; the order will be A, B, C, d, e.

This means that a child always comes before its parent, and in particular, that the root comes last. This is known as reverse post-order traversal, but we'll just call it "Newick order".

Appendix C

Installing the Newick Utilities

C.1 From source

I have tested the Newick Utilities on various distributions of Linux, as well as on Mac OS X¹. On Linux, chances are you already have development tools preinstalled, but some distributions (*e.g.*, Ubuntu) do not install GCC, etc. by default. Check that you have GCC, Bison, Flex, and the GNU autotools, including Libtool. On MacOS X, you need to install XCode (<http://developer.apple.com/tools/xcode>).

The package uses the GNU autotools, like many other open source software packages. So all you need to do is the usual

```
$ tar xzf newick-utils-x.y.z.tar.gz
$ cd newick-utils-x.y.z
$ ./configure
$ make
$ make check
# make install
```

The `make check` is optional, but you should try it anyway. Note that the `nw_gen` test may fail - this is due to differences in random number generators, as far as I can tell.

C.2 As binaries

Since version 1.1, there is also a binary distribution. The name of the archive matches `newick-utils-<version>-<platform>.tar.gz`. All you need to do is:

```
$ tar xzf newick-utils-<vesion>-<platform>.tar.gz
$ cd newick-utils-<vesion>-<platform>
```

The binaries are in `src`. Testing may be less important than when installing from source, but you can do it like this:

```
$ cd tests
$ for test in test*.sh; do ./$test; done
```

any failure will generate a `FAIL` message (which you could filter with `grep`, etc).

You can then copy/move the binaries wherever it suits you.

¹I use Linux as a main development platform. Although I try my best to get the package to compile on Macs, I don't always succeed.