Guangchuang Yu

Data Integration, Manipulation and Visualization of Phylogenetic Trees

Preface

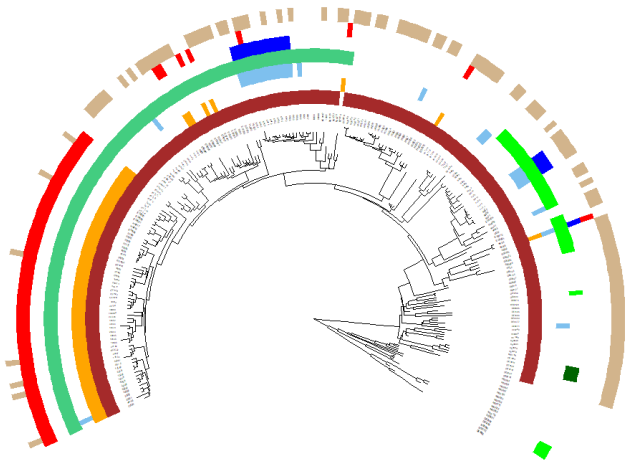
> You can't even begin to understand biology, you can't understand life, unless you understand what it's all there for, how it arose - and that means evolution.
>
> — Richard Dawkins

The book is meant as a guide for data integration, manipulation and visualization of phylogenetic trees using a suite of R packages, tidytree (https://CRAN.R-project.org/package=tidytree), treeio (http://bioconductor.org/packages/treeio) and ggtree (http://bioconductor.org/packages/ggtree).

If you use the software suite in published research, please cite the most appropriate paper(s) from this list:

1. **G Yu**[*], TTY Lam, H Zhu, Y Guan[*]. Two methods for mapping and visualizing associated data on phylogeny using ggtree. ***Molecular Biology and Evolution***, 2018, 35(2):3041-3043. doi: 10.1093/molbev/msy194 (https://doi.org/10.1093/molbev/msy194).
2. **G Yu**, DK Smith, H Zhu, Y Guan, TTY Lam[*]. ggtree: an R package for visualization and annotation of phylogenetic trees with their covariates and other associated data. ***Methods in Ecology and Evolution***. 2017, 8(1):28-36. doi: 10.1111/2041-210X.12628 (https://doi.org/10.1111/2041-210X.12628).

Data Integration, Manipulation and Visualization of Phylogenetic Trees



Guangchuang Yu

School of Basic Medical Sciences
Southern Medical University

The book's source code is hosted on GitHub, at https://github.com/YuLab-SMU/treedata-book. If you notice typos or

other issues, feel free to open an issue on GitHub or send me a pull request. If you do the latter, in your commit message, please add the sentence "I assign the copyright of this contribution to Guangchuang Yu," so that I can maintain the option of publishing this book in other forms.

Part I: Tree data input, output and manipulation

1

Importing Tree with Data

1.1 Overview of Phylogenetic Tree Construction

Phylogenetic trees are used to describe genealogical relationships among a group of organisms, which can be constructed based on the genetic sequences of the organisms. A rooted phylogenetic tree represents a model of evolutionary history depicted by ancestor-descendant relationships between tree nodes and clustering of 'sister' or 'cousin' organisms at different level of relatedness, as illustrated in Figure 1.1. In infectious disease research, phylogenetic trees are usually built from the pathogens' gene or genome sequences to show which pathogen sample is genetically closer to a pathogen sample, providing insights into the underlying unobserved epidemiologic linkage and potential source of an outbreak.

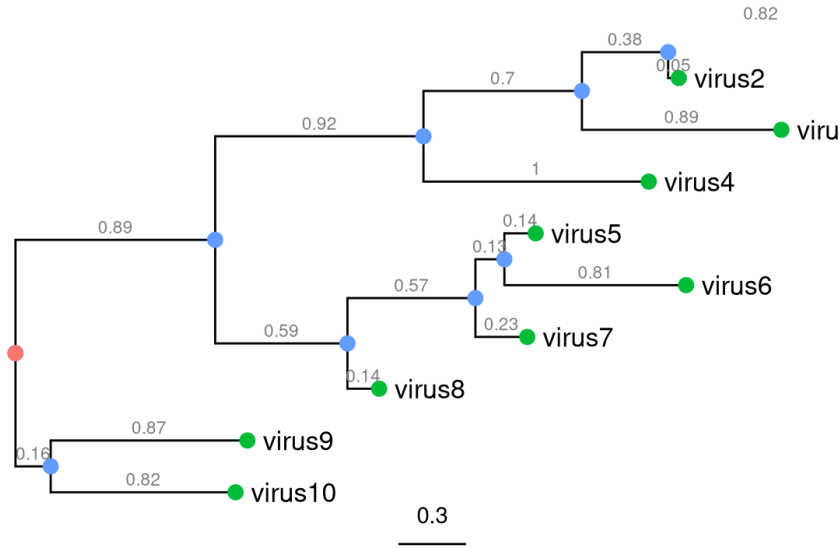


Figure 1.1: **Components of a phylogenetic tree.** External nodes (green circles), also called 'tips', represent actual organisms sampled and sequenced (*e.g.*, virus in infectious disease research). They are the 'taxa' in the terminology of evolutionary biology. The internal nodes (blue circles) represent hypothetical ancestors for the tips. The root (red circle) is the common ancestor of all species in the tree. The horizontal lines are branches and represent evolutionary changes (grey number) measured in unit of time or genetic divergence. The bar at the bottom provides the scale of these branch lengths.

Phylogenetic tree can be constructed from genetic sequences using distance-based methods or character-based methods. Distance-based methods, including unweighted pair group method with arithmetic means (UPGMA) and Neighbor-joining (NJ), are based on the matrix of pairwise genetic distances calculated between sequences. The character-based methods, including maximum parsimony (MP) (Fitch 1971 (page 276)), maximum likelihood (ML) (Felsenstein 1981 (page 276)), and Bayesian Markov Chain Monte Carlo (BMCMC) method (Rannala and Yang 1996 (page 280)), are based on mathematical model that describes the evolution of genetic characters and search for the best phylogenetic tree according to their own optimality criteria.

Maximum Parsimony (MP) method assumes that the evolutionary change is rare and minimizes the amount of character-state changes (*e.g.*, number of DNA substitutions). The criterion is similar to Occam's razor, that the simplest hypothesis that can explains the data is the best hypothesis. Unweighted parsimony assumes mutations across different characters (nucleotides or amino acids) are equally likely while weighted method assume unequal likely of mutations (*e.g.*, the third codon position is more liable than other codon positions; and the transition mutations have higher frequency than transversion). The concept of MP method is straightforward and intuitive, which is a probable reason for its popularity amongst biologists who care more about the research question rather than the computational details of the analysis. However, this method has a number of disadvantages, which in particular the tree inference can be biased by the well-known systematic error called long-branch attraction (LBA) that incorrectly infer distantly related lineages as closely related (Felsenstein 1978 (page 276)). This is because MP method poorly takes into consideration of many sequence evolution factors (*e.g.*, reversals and convergence) that are hardly observable from the existing genetic data.

Maximum likelihood (ML) method and Bayesian Markov Chain Monte Carlo (BMCMC) method are the two most commonly used methods in phylogenetic tree construction and are most often used in scientific publications. ML and BMCMC methods require a substitution model of sequence evolution. Different sequence data have different substitution models to formulate the evolutionary process of DNA, codon and amino acid. There are several models for nucleotide substitution, including JC69, K2P, F81, HKY and GTR (Arenas 2015 (page 275)). These models can be used in conjunction with the rate variation across sites (denoted as +Γ)) (Yang 1994 (page 281)) and the proportion of invariable sites (denoted as +I) (Shoemaker and Fitch 1989 (page 281)). Previous research (Lemmon and Moriarty 2004 (page 278)) had suggested that misspecification of substitution model might bias phylogenetic inference. Procedural testing for the best-fit substitution model is recommended.

The optimal criterion of ML method is to find the tree that maximizes the likelihood given the sequence data. The procedure of ML method is simple: calculating the likelihood of a tree and optimizing its topology and

branches (and the substitution model parameters, if not fixed) until the best tree is found. Heuristic search such as those implemented in *PhyML* and *RAxML*, is often used to find the best tree based on the likelihood criterion. Bayesian method finds the tree that maximizes posterior probability by sampling trees through MCMC based on the given substitution model. One of the advantage of BMCMC is that parameter variance and tree topological uncertainty, included by the posterior clade probability, can be naturally and conveniently obtained from the sampling trees in MCMC process. Moreover, influence of topological uncertainty to other parameter estimates are also naturally integrated in the BMCMC phylogenetic framework.

In a simple phylogenetic tree, data associated with the tree branches/nodes could be the branch lengths (indicating genetic or time divergence) and lineage supports such as bootstrap values estimated from bootstrapping procedure or posterior clade probability summarized from the sampled trees in the BMCMC analysis.

1.2 Phylogenetic Tree Formats

There are several file formats designed to store phylogenetic trees and the data associated with the nodes and branches. The three commonly used formats are Newick[1], NEXUS (Maddison et al. 1997) and Phylip (Felsenstein 1989 (page 276)). Some formats (*e.g.*, NHX) are extended from Newick format. Newick and NEXUS formats are supported as input by most of the software in evolutionary biology, while some of the software tools output newer standard files (*e.g.*, *BEAST* and *MrBayes*) by introducing new rules/data blocks for storing evolutionary inferences. On the other cases (*e.g.*, *PAML* and *r8s*), output log files are only recognized by their own single software.

1.2.1 Newick tree format

The Newick tree format is the standard for representing trees in computer-readable form.

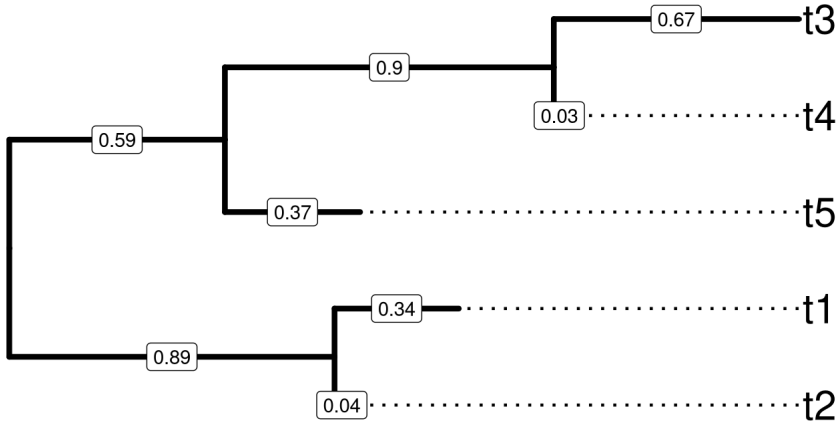


Figure 1.2: **A sample tree for demonstrating Newick text to encode tree structure.** Tips were aligned to right hand side and branch lengths were labelled on the middle of each branch.

The rooted tree shown in Figure 1.2 can be represented by the following sequence of characters as a newick tree text.

```
((t2:0.04,t1:0.34):0.89,(t5:0.37,(t4:0.03,t3:0.67):0.9):0.59);
```

The tree text ends with semicolon. Internal nodes are represented by a pair of matched parentheses. Between the parentheses are descendant nodes of that node. For instance `(t4:0.59, t1:0.37)` represents the parent node of `t4` and `t1` that are the immediate descendants. Sibling nodes are separated by comma and tips are represented by their names. A branch length (from parent node to child node) is represented by a real number after the child node and preceded by a colon. Singular data (*e.g.*, bootstrap values) associated with internal nodes or branches maybe encoded as node label and represented by the simple text/numbers before the colon.

Newick tree format was developed by Meacham in 1984 for the *PHYLIP* (Retief 2000 (page 280)) package. Newick format is now the most widely used tree format and used by *PHYLIP*, *PAUP** (Wilgenbusch and Swofford 2003 (page 281)), *TREE-PUZZLE* (Schmidt et al. 2002), *MrBayes* and many other applications. Phylip tree format contains Phylip multiple sequence

alignment (MSA) with a corresponding Newick tree text that was built based on the MSA sequences in the same file.

1.2.2 NEXUS tree format

The NEXUS format incorporates Newick tree text with related information organized into separated units known as **blocks**. A NEXUS block has the following structure:

```
#NEXUS
...
BEGIN characters;
...
END;
```

For example, the above example tree can be saved as a following NEXUS format:

```
#NEXUS
[R-package APE, Wed Nov  9 11:46:32 2016]

BEGIN TAXA;
    DIMENSIONS NTAX = 5;
    TAXLABELS
        t5
        t4
        t1
        t2
        t3
    ;
END;
BEGIN TREES;
    TRANSLATE
        1   t5,
        2   t4,
        3   t1,
        4   t2,
        5   t3
    ;
```

```
TREE * UNTITLED = [&R] (1:0.89,((2:0.59,3:0.37):0.34,
    (4:0.03,5:0.67):0.9):0.04);
END;
```

Comments can be placed by using square brackets. Some blocks can be recognized by most of the programs including `TAXA` (contains information of taxa), `DATA` (contains data matrix, *e.g.*, sequence alignment) and `TREE` (contains phylogenetic tree, *i.e.*, Newick tree text). Notably, blocks can be very diversed and some of them are only be recognized by one particular program. For example NEXUS file exported by *PAUP** has a **paup** block which contains *PAUP** commands, whereas *FigTree* exports NEXUS file with a **figtree** block that contains visualization settings. NEXUS organizes different types of data into different blocks, whereas programs that support reading NEXUS can parse some blocks they recognized and ignore those they could not. This is a good mechanism to allow different programs to use the same format without crashing when unsupported types of data are present. Notably most of the programs only support parsing `TAXA`, `DATA` and `TREE` blocks, therefore a program/platform that could generically read all kinds of data blocks from the NEXUS would be useful for phylogenetic data integration.

The `DATA` block is widely used to store sequence alignment. For this purpose, user can store tree and sequence data in Phylip format which are essentially Phylip multiple sequence alignment and Newick tree text respectively. It is used in Phylogeny Inference Package (*PHYLIP*).

1.2.3 New Hampshire eXtended format

Newick, NEXUS and phylip are mainly designed to store phylogenetic tree and basic singular data associated with internal nodes or branches. In addition to the singular data annotation at branches and nodes (mentioned above), New Hampshire eXtended format (NHX), which is based on Newick (also called New Hampshire), was developed to introduce tags to associate multiple data fields with the tree nodes (both internal nodes and tips). Tags are placed after branch length and must be wrapped between `[&&NHX` and `]` which makes it possible to compatible with NEXUS format as it defined characters between `[` and `]` as comments. NHX is also the output format of *PHYLDOG* (Boussau et al.

2013 (page 275)) and *RevBayes* (Höhna et al. 2016). A Tree Viewer (*ATV*) (Zmasek and Eddy 2001) is a java tool that supports displaying annotation data stored in NHX format, but this package is no more maintained.

Here is a sample tree from NHX definition document[2]:

```
(((ADH2:0.1[&&NHX:S=human],
ADH1:0.11[&&NHX:S=human]):0.05[&&NHX:S=primates:D=Y:B=100],ADHY:0.1[&&NH
(ADH4:0.09[&&NHX:S=yeast],ADH3:0.13[&&NHX:S=yeast],
ADH2:0.12[&&NHX:S=yeast],ADH1:0.11[&&NHX:S=yeast]):0.1[&&NHX:S=Fungi])
[&&NHX:D=N];
```

1.2.4 Jplace format

In order to store the NGS short reads mapped onto a phylogenetic tree (for the purpose of metagenomic classification), Matsen proposed jplace format for such phylogenetic placements (Matsen et al. 2012 (page 279)). Jplace format is based on JSON and contains four keys: `tree`, `fields`, `placements`, `metadata` and `version`. The `tree` value contains tree text extended from Newick tree format by putting the edge label in brackets (if available) after branch length and putting the edge number in curly braces after edge label. The `fields` value contains header information of placement data. The value of `placements` is a list of `pqueries`. Each `pquery` contains two keys: `p` for placements and `n` for name or `nm` for names with multiplicity. The value of `p` is a list of placement for that `pqueries`.

Here is a jplace sample file:

```
{
    "tree": "(((((((A:4{1},B:4{2}):6{3},C:5{4}):8{5},D:6{6}):
    3{7},E:21{8}):10{9},((F:4{10},G:12{11}):14{12},H:8{13}):
    13{14}):13{15},((I:5{16},J:2{17}):30{18},(K:11{19},
    L:11{20}):2{21}):17{22}):4{23},M:56{24});",
    "placements": [
    {"p":[24, -61371.300778, 0.333344, 0.000003, 0.003887],
     "n":["AA"]
    },
    {"p":[[1, -61312.210786, 0.333335, 0.000001, 0.000003],
```

```
[2, -61322.210823, 0.333322, 0.000003, 0.000003],
          [3, -61352.210823, 0.333322, 0.000961, 0.000003]],
     "n":["BB"]
    },
    {"p":[[8, -61312.229128, 0.200011, 0.000001, 0.000003],
          [9, -61322.229179, 0.200000, 0.000003, 0.000003],
          [10, -61342.229223, 0.199992, 0.000003, 0.000003]],
    "n":["CC"]
    }
    ],
    "metadata": {"info": "a jplace sample file"},
    "version" : 2,
    "fields": ["edge_num", "likelihood", "like_weight_ratio",
    "distal_length", "pendant_length"
    ]
}
```

Jplace is the output format of *pplacer* (Frederick A. Matsen, Kodner, and Armbrust 2010) and Evolutionary Placement Algorithm (*EPA*) (Berger, Krompass, and Stamatakis 2011 (page 275)a (page 275)). But these two programs do not contain tools to visualize placement results. *Pplacer* provides `placeviz` to convert jplace file to phyloXML or Newick formats which can be visualized by *Archaeopteryx*[3].

1.2.5 Software outputs

RAxML (Stamatakis 2014 (page 281)) can output Newick format by storing the bootstrap values as internal node labels. Another way that *RAxML* supported is to place bootstrap value inside square brackets and after branch length. This could not be supported by most of the software that support Newick format where square brackets will be ignored.

BEAST (Bouckaert et al. 2014 (page 275)) output is based on NEXUS and it also introduce square brackets in the tree block to store evolutionary evidences inferred by *BEAST*. Inside brackets, curly braces may also incorporated if feature values have length more than 1 (*e.g.*, HPD or range of substitution rate). These brackets are placed between node and branch length (*i.e.*, after label if exists and before colon). Bracket is not defined in Newick format and is reserve character for NEXUS comment. So these information will be ignored for standard NEXUS parsers.

Here is a sample TREE block of the *BEAST* output:

```
tree TREE1 = [&R]
(((11[&length=9.4]:9.38,14[&length=6.4]:6.385096430786298)
[&length=25.7]:25.43,4[&length=9.1]:8.821663252749829)
[&length=3.0]:3.10,(12[&length=0.6]:0.56,
(10[&length=1.6]:1.56,(7[&length=5.2]:5.19,
((((2[&length=3.3]:3.26,(1[&length=1.3]:1.32,
(6[&length=0.8]:0.83,13[&length=0.8]:0.8311577761397366)
[&length=2.4]:2.48917886025146)
[&length=0.9]:0.9416178372674331)
[&length=0.4]:0.49,9[&length=1.7]:1.757288031101215)
[&length=2.4]:2.35,8[&length=2.1]:2.1125745387283246)
[&length=0.2]:0.23,(3[&length=3.3]:3.31,
(15[&length=5.2]:5.27,5[&length=3.2]:3.2710481368304585)
[&length=1.0]:1.0409443024626412)
[&length=1.9]:2.0372962536780435)
[&length=2.8]:2.8446835614595685)
[&length=5.3]:5.367459711197171)
[&length=2.0]:2.0037467863383043)
[&length=4.3]:4.360909907798238)[&length=0.0];
```

BEAST output can contain many different evolutionary inferences, depending of the analysis models defined in *BEAUTi* for running. For example in molecular clock analysis, it contains `rate`, `length`, `height`, `posterior` and corresponding HPD and range for uncertainty estimation. `Rate` is the estimated evolutionary rate of the branch. `Length` is the length of the branch in years. `Height` is the time from node to root while `posterior` is the Bayesian clade credibility value. The above example is the output tree of clock analysis and should contains these inferences. To save space, I only keep the `length` above.

MrBayes (Huelsenbeck and Ronquist 2001 (page 277)) is a program that uses Markov Chain Monte Carlo method to sample from the posterior probability distributions. Its output file annotates nodes and branches separately by two sets of square brackets. For example below, posterior clade probabilities for the node and branch length estimates for the branch:

```
tree con_all_compat = [&U]
(8[&prob=1.0]:2.94e-1[&length_mean=2.9e-1],10[&prob=1.0]:2.25e-1[&length
((((1[&prob=1.0]:1.43e-1[&length_mean=1.4e-1],2[&prob=1.0]:1.92e-1[&leng
[&prob=1.0]:1.24e-1[&length_mean=1.2e-1],9[&prob=1.0]:2.27e-1[&length_me
[&prob=1.0]:1.72e-1[&length_mean=1.7e-1],12[&prob=1.0]:5.11e-1[&length_m
[&prob=1.0]:1.76e-1[&length_mean=1.7e-1],
(((3[&prob=1.0]:5.46e-2[&length_mean=5.4e-2],
(6[&prob=1.0]:1.03e-2[&length_mean=1.0e-2],7[&prob=1.0]:7.13e-3[&length_
[&prob=1.0]:6.93e-2[&length_mean=6.9e-2])
[&prob=1.0]:6.03e-2[&length_mean=6.0e-2],
(4[&prob=1.0]:6.27e-2[&length_mean=6.2e-2],5[&prob=1.0]:6.31e-2[&length_
[&prob=1.0]:6.07e-2[&length_mean=6.0e-2])
[&prob=1.0]:1.80e-1[&length_mean=1.8e-1],11[&prob=1.0]:2.37e-1[&length_m
[&prob=1.0]:4.05e-1[&length_mean=4.0e-1])
[&prob=1.0]:1.16e+000[&length_mean=1.162699558201079e+000])
[&prob=1.0][&length_mean=0];
```

To save space, most of the inferences were removed and only contains `prob` for clade probability and `length_mean` for mean value of branch length. The full version of this file also contains `prob_stddev`, `prob_range`, `prob(percent)`, `prob+-sd` for probability inferences and `length_median`, `length_95%_HPD` for every branch.

The *BEAST* and *MrBayes* outputs are expected to be parsed without inferences (dropped as comments) by software that support NEXUS. *FigTree* supports parsing *BEAST* and *MrBayes* outputs with inferences that can be used to display or annotate on the tree. But from there extracting these data for further analysis is still challenging.

HyPhy (Pond, Frost, and Muse 2005) could do a number of phylogenetic analysis, including ancestral sequence reconstruction. For ancestral sequence reconstruction, these sequences and the Newick tree text are stored in NEXUS format as the major analysis output. It did not completely follow the NEXUS definition and only put the ancestral node labels in `TAXA` instead of external node label. The `MATRIX` block contains sequence alignment of ancestral nodes which cannot be referred back to the tree stored in `TREES` block since it does not contains node labels. Here is the sample output (to save space, only the first 72bp of alignment are shown):

```
#NEXUS

[
Generated by HYPHY 2.0020110620beta(MP) for MacOS(Universal Bin-
```

```
ary)
    on Tue Dec 23 13:52:34 2014

]

BEGIN TAXA;
    DIMENSIONS NTAX = 13;
    TAXLABELS
        'Node1' 'Node2' 'Node3' 'Node4' 'Node5' 'Node12' 'Node13' 'Node1
            'Node18' 'Node20' 'Node22' 'Node24' 'Node26' ;
END;

BEGIN CHARACTERS;
    DIMENSIONS NCHAR = 2148;
    FORMAT
        DATATYPE = DNA

        GAP=-
        MISSING=?
        NOLABELS
    ;

MATRIX
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATTGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAATGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
 ATGGAAGACTTTGTGCGACAGTGCTTCAATCCAATGATCGTCGAGCTTGCGGAAAAGGCAATGAAAGAATA
END;

BEGIN TREES;
    TREE tree = (K,N,(D,(L,(J,(G,((C,(E,O)),(H,(I,(B,(A,(F,M))))))))));
END;
```

There are other applications that output rich information text that also contains phylogenetic trees with associated data. For example *r8s* (Sanderson 2003) output three trees in its log file, namely TREE, RATE and PHYLO for branches scaled by time, substitution rate, and absolute substitutions respectively.

Phylogenetic Analysis by Maximum Likelihood (*PAML*) (Yang 2007 (page 282)) is a package of programs for phylogenetic analyses of DNA or protein sequences. Two main programs, *BaseML* and *CodeML*, implement a variety of models. *BaseML* estimates tree topology, branch lengths and substitution parameters using a number of nucleotide substitution models available, including JC69, K80, F81, F84, HKY85, T92, TN93 and GTR. *CodeML* estimates synonymous and non-synonymous substitution rates, likelihood ratio test of positive selection under codon substitution models (Goldman and Yang 1994 (page 276)).

BaseML outputs *mlb* file that contains input sequence (taxa) alignment and phylogenetic tree with branch length as well as substitution model and other parameters estimated. The supplementary result file, *rst*, contains sequence alignment (with ancestral sequence if perform reconstruction of ancestral sequences) and Naive Empirical Bayes (NBE) probabilities that each site in the alignment evolved. *CodeML* outputs *mlc* file that contains tree structure and estimation of synonymous and non-synonymous substitution rates. *CodeML* also output supplementary result file, *rst*, that is similar to *BaseML* except that site is defined as codon instead of nucleotide. Parsing these files can be tedious and would oftentimes need a number of post-processing steps and require expertise in programming (e.g. with Python[4] or Perl[5]).

Introducing square brackets is quite common for storing extra information, including *RAxML* to store bootstrap value, NHX format for annotation, jplace for edge label and *BEAST* for evolutionary estimation, *etc.*. But the positions to place square brackets are not consistent in different software and the contents employ different rules for storing associated data, these properties make it difficult to parse associated data. For most of the software, they will just ignore square brackets and only parse the tree structure if the file is compatible. Some of them

contains invalid characters (e.g. curly braces in `tree` field of jplace format) and even the tree structure can't be parsed by standard parsers.

It is difficult to extract useful phylogeny/taxon-related information from the different analysis outputs produced by various evolutionary inference software, for displaying on the same phylogenetic tree and for further analysis. *FigTree* supports *BEAST* output, but not for most of other software outputs that contains evolutionary inferences or associated data. For those output rich text files (e.g. *r8s*, *PAML*, *etc.*), the tree structure cannot be parsed by any tree viewing software and users need expertise to manually extract the phylogenetic tree and other useful result data from the output file. However, such manual operation is slow and error-prone.

It was not easy to retrieve phylogenetic trees with evolutionary data from different analysis outputs of commonly used software in the field. Some of them (*e.g.*, *PAML* output and jplace file) without software or programming library to support parsing file, while others (*e.g.*, *BEAST* and *MrBayes* output) can be parsed without evolutionary inferences as they are stored in square brackets that will be omitted as comment by most of the software. Although *FigTree* support visualizing evolutionary statistics inferred by *BEAST* and *MrBayes*, extracting these data for further analysis is not supported. Different software packages implement different algorithms for different analyses (*e.g.*, *PAML* for d_N/d_S, *HyPhy* for ancestral sequences and *BEAST* for skyline analysis). Therefore, in encountering the genomic sequence data, there is a desire need to efficiently and flexibly integrate different analysis inference results for comprehensive understanding, comparison and further analysis. This motivated us to develop the programming library to parse the phylogenetic trees and data from various sources.

1.3 Getting Tree Data with *treeio*

Phylogenetic trees are commonly used to present evolutionary relationships of species. Information associated with taxon species/strains may be further analyzed in the context of the evolutionary

history depicted by the phylogenetic tree. For example, host information of the influenza virus strains in the tree could be studied to understand host range of a virus linage. Moreover, such meta-data (*e.g.*, isolation host, time, location, *etc.*) directly associated with taxon strains are also often subjected to further evolutionary or comparative phylogenetic models and analyses, to infer their dynamics associated with the evolutionary or transmission processes of the virus. All these meta-data or other phenotypic or experimental data are stored either as the annotation data associated with the nodes or branches, and are often produced in inconsistent format by different analysis programs.

Getting trees into R is still limited. Newick and Nexus can be imported by several packages, including ape (https://CRAN.R-project.org/package=ape), phylobase (https://CRAN.R-project.org/package=phylobase). NeXML format can be parsed by RNeXML (https://CRAN.R-project.org/package=RNeXML). However, analysis results from widely used software packages in this field are not well supported. SIMMAP output can be parsed by phyext2 (https://CRAN.R-project.org/package=phyext2) and phytools (https://CRAN.R-project.org/package=phytools). Although PHYLOCH (http://www.christophheibl.de/Rpackages.html) can import *BEAST* and *MrBayes* output, only internal node attributes were parsed and tip attributes were ignore[6]. Many other software outputs are mainly required programming expertise to import the tree with associated data. Linking external data, including experimental and clinical data, to phylogeny is another obstacle for evolution biologists.

To fill the gap that most of the tree formats or software outputs cannot be parsed within the same software/platform, an R package *treeio* was developed for parsing various tree file formats and outputs from common evolutionary analysis software. *Treeio* is developed with the R programming language (R Core Team 2016 (page 280)). Not only the tree structure can be parsed but also the associated data and evolutionary inferences, including NHX annotation, clock rate inferences (from BEAST (http://beast2.org/) or r8s (http://ginger.ucdavis.edu/r8s) (Sanderson 2003) programs), snynonymous and non-synonymous substitutions (from CodeML), and ancestral sequence construction (from HyPhy (https://veg.github.io/hyphy-site/), BaseML (http://abacus.gene.ucl.ac.uk/software/paml.html) or CodeML (http://abacus.gene.ucl.ac.uk/software/paml.html)), *etc.*. Currently, treeio (https://bioconductor.org/packages/treeio/) is able to read the following

file formats: Newick, Nexus, New Hampshire eXtended format (NHX), jplace and Phylip as well as the data outputs from the following analysis programs: ASTRAL (https://github.com/smirarab/ASTRAL), BEAST (http://beast2.org/), EPA (http://sco.h-its.org/exelixis/web/software/epa/index.html), HyPhy (https://veg.github.io/hyphy-site/), MEGA (https://www.megasoftware.net/), MrBayes (http://nbisweden.github.io/MrBayes/), PAML (http://abacus.gene.ucl.ac.uk/software/paml.html), PHYLDOG (https://pbil.univ-lyon1.fr/software/phyldog/), pplacer (http://matsen.fhcrc.org/pplacer/), r8s (http://ginger.ucdavis.edu/r8s), RAxML (http://evomics.org/learning/phylogenetics/raxml/) and RevBayes (https://revbayes.github.io/intro.html) *etc*. This is made possible with the several parser functions developed in *treeio* (Table 1.1)[7].

Table 1.1: Parser functions defined in treeio

Parser function	Description
read.astral	parsing output of ASTRAL
read.beast	parsing output of BEAST
read.codeml	parsing output of CodeML (rst and mlc files)
read.codeml_mlc	parsing mlc file (output of CodeML)
read.fasta	parsing FASTA format sequence file
read.hyphy	parsing output of HYPHY
read.hyphy.seq	parsing ancestral sequences from HYPHY output
read.iqtree	parsing IQ-Tree newick string, with ability to parse SH-aLRT and UFBoot support values
read.jplace	parsing jplace file including output of EPA and pplacer
read.jtree	parsing jtree (page 82) format
read.mega	parsing MEGA Nexus output
read.mega_tabular	parsing MEGA tabular output
read.mrbayes	parsing output of MrBayes
read.newick	parsing newick string, with ability to parse node label as support values
read.nhx	parsing NHX file including output of PHYLDOG and

RevBayes	
read.paml_rst	parsing rst file (output of BaseML or CodeML)
read.phylip	parsing phylip file (phylip alignment + newick string)
read.phylip.seq	parsing multiple sequence alignment from phylip file
read.phylip.tree	parsing newick string from phylip file
read.r8s	parsing output of r8s
read.raxml	parsing output of RAxML

The treeio (https://bioconductor.org/packages/treeio/) package defines base classes and functions for phylogenetic tree input and output. It is an infrastructure that enables evolutionary evidences that inferred by commonly used software packages to be used in R. For instance, d_N/d_S values or ancestral sequences inferred by CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html) (Yang 2007 (page 282)), clade support values (posterior) inferred by BEAST (http://beast2.org/) (Bouckaert et al. 2014 (page 275)) and short read placement by EPA (http://sco.h-its.org/exelixis/web/software/epa/index.html) (Berger, Krompass, and Stamatakis 2011 (page 275)b (page 275)) and pplacer (http://matsen.fhcrc.org/pplacer/) (Frederick A Matsen, Kodner, and Armbrust 2010 (page 279)). These evolutionary evidences can be further analyzed in R and used to annotate phylogenetic tree using ggtree (https://bioconductor.org/packages/ggtree/) (Yu et al. 2017). The growth of analysis tools and models available introduces a challenge to integrate different varieties of data and analysis results from different sources for an integral analysis on the the same phylogenetic tree background. The treeio (https://bioconductor.org/packages/treeio/) package provides a `merge_tree` function to allow combining tree data (page 49) obtained from different sources. In addition, treeio (https://bioconductor.org/packages/treeio/) also enables external data (page 53) to be linked to phylogenetic tree structure.

After parsing, storage of the tree structure with associated data is made through a S4 class, `treedata`, defined in the tidytree (https://CRAN.R-project.org/package=tidytree) package. These parsed data are mapped to the tree branches and nodes inside `treedata` object, so that they can be efficiently used to visually annotate the tree using ggtree (https://bioconductor.org/packages/ggtree/)

package (Yu et al. 2017) (described in Chapter 4 (page 107) and 5 (page 139)). A programmable platform for phylogenetic data parsing, integration and annotations as such makes us easier to identify the evolutionary dynamics and correlation patterns.

1.3.1 Overview of *treeio*

The treeio (http://bioconductor.org/packages/treeio) package defined S4 classes for storing phylogenetic tree with diverse types of associated data or covariates from different sources including analysis outputs from different software packages. It also defined corresponding parser functions for parsing phylogenetic tree with annotation data and stored as data object in R for further manipulation or analysis (see Table 1.1). Several accessor functions were defined to facilitate accessing the tree annotation data, including `get.fields` for obtaining annotation features available in the tree object, `get.placements` for obtaining the phylogenetic placement results (*i.e.*, output of *pplacer*, *EPA*, *etc.*), `get.subs` for obtaining the genetic substitutions from parent node to child node, and `get.tipseq` for getting the tip sequences.

The S3 class, `phylo`, which was defined in *ape* (Paradis, Claude, and Strimmer 2004 (page 280)) package, is widely used in R community and many packages. As *treeio* uses S4 class, to enable those available R packages to analyze the tree imported by *treeio*, *treeio* provides `as.phylo` function to convert *treeio*-generated tree object to `phylo` object that only contains tree structure without annotation data. In the other way, *treeio* also provides `as.treedata` function to convert `phylo` object with evolutionary analysis result (*e.g.*, bootstrap values calculated by *ape* or ancestral states inferred by *phangorn* (Schliep 2011 (page 280)) *etc*) to be stored as a `treedata` S4 object, making it easy to map the data to the tree structure and to be visualized using ggtree (http://bioconductor.org/packages/ggtree).

To allow integration of different kinds of data in phylogenetic tree, *treeio* provides `merge_tree` function (details in section 2.2.1 (page 49)) for combining evolutionary statistics/evidences imported from different sources including those common tree files and outputs from analysis programs (Table 1.1). There are other information, such as sampling location, taxonomy information, experimental result and evolutionary

traits, *etc.* that are stored in separate files with user-defined format. In treeio (http://bioconductor.org/packages/treeio), we could read in these data from the users' files using standard R *IO* functions, and attach them to the tree object by the full_join (page 53) method defined in our treeio (http://bioconductor.org/packages/treeio) package (see also the %<+% operator (page 172) defined in ggtree (http://bioconductor.org/packages/ggtree)). After attaching, the data will become the attributes associated with nodes or branches, which can be compared with other data incorporated, or can be visually displayed on the tree.

To facilitate storing the merged data into a single file, treeio (http://bioconductor.org/packages/treeio) implemented `write.baset` and `write.jtree` function to export treedata object for storing complex data associated with the phylogenetic tree (see Chapter 3 (page 71)).

1.3.2 Function Demonstration

1.3.2.1 Parsing BEAST output

```
file <- system.file("extdata/BEAST", "beast_mcc.tree",
         package="treeio")
beast <- read.beast(file)
beast
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/BEAST/beast_mcc.tree'.
##
## ...@ phylo:
## Phylogenetic tree with 15 tips and 14 internal nodes.
##
## Tip labels:
##  A_1995, B_1996, C_1995, D_1987, E_1996, F_1997, ...
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'height',   'height_0.95_HPD',  'height_median',
##  'height_range', 'length',   'length_0.95_HPD',
```

```
##  'length_median',    'length_range', 'posterior',    'rate',
##  'rate_0.95_HPD',    'rate_median',  'rate_range'.
```

Since *%* is not a valid character in *names*, all the feature names that contain *x%* will convert to *0.x*. For example, *length_95%_HPD* will be changed to *length_0.95_HPD*.

Not only tree structure but also all the features inferred by *BEAST* will be stored in the S4 object. These features can be used for tree annotation (Figure 5.7).

1.3.2.2 Parsing MEGA output

MEGA (https://www.megasoftware.net/) (Kumar, Stecher, and Tamura 2016 (page 278)) supports exporting trees in three distinct formats: Newick, tabular and Nexus. The Newick file can be parsed using the `read.tree` or `read.newick` functions. MEGA Nexus file is similar to BEAST Nexus and treeio (http://bioconductor.org/packages/treeio) provides `read.mega` function to parse the tree.

```
file <- system.file("extdata/MEGA7", "mtCDNA_timetree.nex",
         package = "treeio")
read.mega(file)
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/MEGA7/mtCDNA_timetree.nex'.
##
## ...@ phylo:
## Phylogenetic tree with 7 tips and 6 internal nodes.
##
## Tip labels:
##  homo_sapiens, chimpanzee, bonobo, gorilla, orangutan, sumatran, ...
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'branch_length',    'data_coverage',    'rate', 'reltime',
##  'reltime_0.95_CI',  'reltime_stderr'.
```

The tabular output contains tree and associated information (divergence time in this example) in a tabular flat text file. The `read.mega_tabular` function can parse the tree with data simultaneously.

```
file <- system.file("extdata/MEGA7",
         "mtCDNA_timetree_tabular.txt", package = "treeio")
read.mega_tabular(file)
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/MEGA7/mtCDNA_timetree_tabular.t
##
## ...@ phylo:
## Phylogenetic tree with 7 tips and 6 internal nodes.
##
## Tip labels:
##  chimpanzee, bonobo, homo sapiens, gorilla, orangutan, sumatran, ...
## Node labels:
## [1] ""           ""           "demoLabel2" ""
## [5] ""           ""
##
## Rooted; no branch lengths.
##
## with the following features available:
##  'RelTime',  'CI_Lower', 'CI_Upper', 'Rate',
##  'Data Coverage'.
```

1.3.2.3 Parsing MrBayes output

```
file <- system.file("extdata/MrBayes", "Gq_nxs.tre",
         package="treeio")
read.mrbayes(file)
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/MrBayes/Gq_nxs.tre'.
##
## ...@ phylo:
```

```
## Phylogenetic tree with 12 tips and 10 internal nodes.
##
## Tip labels:
##  B_h, B_s, G_d, G_k, G_q, G_s, ...
##
## Unrooted; includes branch lengths.
##
## with the following features available:
##  'length_0.95HPD',   'length_mean',  'length_median',    'prob',
##  'prob_range',   'prob_stddev',  'prob_percent', 'prob+-sd'.
```

1.3.2.4 Parsing PAML output

The `read.paml_rst` function can parse *rst* file from BASEML (http://abacus.gene.ucl.ac.uk/software/paml.html) and CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html). The only difference is the space in the sequences. For BASEML (http://abacus.gene.ucl.ac.uk/software/paml.html), each ten bases are separated by one space, while for CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html), each three bases (triplet) are separated by one space.

```
brstfile <- system.file("extdata/PAML_Baseml", "rst",
         package="treeio")
brst <- read.paml_rst(brstfile)
brst
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/PAML_Baseml/rst'.
##
## ...@ phylo:
## Phylogenetic tree with 15 tips and 13 internal nodes.
##
## Tip labels:
##  A, B, C, D, E, F, ...
## Node labels:
##  16, 17, 18, 19, 20, 21, ...
##
## Unrooted; includes branch lengths.
##
```

```
## with the following features available:
##  'subs', 'AA_subs'.
```

Similarly, we can parse the *rst* file from CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html).

```
crstfile <- system.file("extdata/PAML_Codeml", "rst",
        package="treeio")
## type can be one of "Marginal" or "Joint"
crst <- read.paml_rst(crstfile, type = "Joint")
crst
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/PAML_Codeml/rst'.
##
## ...@ phylo:
## Phylogenetic tree with 15 tips and 13 internal nodes.
##
## Tip labels:
##  A, B, C, D, E, F, ...
## Node labels:
##  16, 17, 18, 19, 20, 21, ...
##
## Unrooted; includes branch lengths.
##
## with the following features available:
##  'subs', 'AA_subs'.
```

Ancestral sequences inferred by BASEML (http://abacus.gene.ucl.ac.uk/software/paml.html) or CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html) via marginal or joint ML reconstruction methods will be stored in the S4 object and mapped to tree nodes. treeio (https://bioconductor.org/packages/treeio/) will automatically determine the substitutions between the sequences at the both ends of each branch. Amino acid substitution will also be determined by translating nucleotide sequences to amino acid sequences. These computed substitutions will also be stored in the S4 object for efficient tree annotation later (Figure 5.9).

CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html) infers selection pressure and estimated d_N/d_S, d_N and d_S. These information are stored in output file *mlc*, which can be parsed by `read.codeml_mlc` function.

```
mlcfile <- system.file("extdata/PAML_Codeml", "mlc",
        package="treeio")
mlc <- read.codeml_mlc(mlcfile)
mlc
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/PAML_Codeml/mlc'.
##
## ...@ phylo:
## Phylogenetic tree with 15 tips and 13 internal nodes.
##
## Tip labels:
##  A, B, C, D, E, F, ...
## Node labels:
##  16, 17, 18, 19, 20, 21, ...
##
## Unrooted; includes branch lengths.
##
## with the following features available:
##  't',    'N',    'S',    'dN_vs_dS', 'dN',   'dS',   'N_x_dN',   'S_x
```

In previous session, we separately parsed *rst* and *mlc* files. However, they can also be parsed together using `read.codeml` function.

```
## tree can be one of "rst" or "mlc" to specify
## using tree from which file as base tree in the object
ml <- read.codeml(crstfile, mlcfile, tree = "mlc")
ml
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/PAML_Codeml/rst',
##  '/home/ygc/R/library/treeio/extdata/PAML_Codeml/mlc'.
```

```
##
## ...@ phylo:
## Phylogenetic tree with 15 tips and 13 internal nodes.
##
## Tip labels:
##  A, B, C, D, E, F, ...
## Node labels:
##  16, 17, 18, 19, 20, 21, ...
##
## Unrooted; includes branch lengths.
##
## with the following features available:
##  'subs', 'AA_subs',  't',    'N',    'S',    'dN_vs_dS', 'dN',   'dS'
##  'N_x_dN',   'S_x_dS'.
```

All the features in both *rst* and *mlc* files were imported into a single S4 object and hence are available for further annotation and visualization. For example, we can annotate and display both d_N/d_S (from *mlc* file) and amino acid substitutions (derived from *rst* file) on the same phylogenetic tree (Yu et al. 2017).

1.3.2.5 Parsing HyPhy output

Ancestral sequences inferred by HyPhy (https://veg.github.io/hyphy-site/) are stored in the Nexus output file, which contains the tree topology and ancestral sequences. To parse this data file, users can use the `read.hyphy.seq` function.

```
ancseq <- system.file("extdata/HYPHY", "ancseq.nex",
          package="treeio")
read.hyphy.seq(ancseq)
```

```
## 13 DNA sequences in binary format stored in a list.
##
## All sequences of same length: 2148
##
## Labels:
## Node1
```

```
## Node2
## Node3
## Node4
## Node5
## Node12
## ...
##
## Base composition:
##     a     c     g     t
## 0.335 0.208 0.237 0.220
## (Total: 27.92 kb)
```

To map the sequences on the tree, user shall also provide an internal-node-labelled tree. If users want to determine substitution, they need also provide tip sequences.

```
nwk <- system.file("extdata/HYPHY", "labelledtree.tree",
        package="treeio")
tipfas <- system.file("extdata", "pa.fas", package="treeio")
hy <- read.hyphy(nwk, ancseq, tipfas)
hy
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/HYPHY/labelledtree.tree'.
##
## ...@ phylo:
## Phylogenetic tree with 15 tips and 13 internal nodes.
##
## Tip labels:
##  K, N, D, L, J, G, ...
## Node labels:
##  Node1, Node2, Node3, Node4, Node5, Node12, ...
##
## Unrooted; includes branch lengths.
##
## with the following features available:
##  'subs', 'AA_subs'.
```

1.3.2.6 Parsing r8s output

r8s (http://loco.biosci.arizona.edu/r8s/) uses parametric, semiparametric and nonparametric methods to relax molecular clock to allow better estimations of divergence times and evolution rates (Sanderson 2003). It outputs three trees in log file, namely *TREE*, *RATO* and *PHYLO* for time tree, rate tree and absolute substitution tree respectively.

Time tree is scaled by divergence time, rate tree is scaled by substitution rate and absolute substitution tree is scaled by absolute number of substitution. After parsing the file, all these three trees are stored in a *multiPhylo* object.

```
r8s <- read.r8s(system.file("extdata/r8s",
        "H3_r8s_output.log", package="treeio"))
r8s
```

```
## 3 phylogenetic trees
```

1.3.2.7 Parsing output of RAxML bootstraping analysis

RAxML (http://evomics.org/learning/phylogenetics/raxml/) bootstraping analysis output a Newick tree text that is not standard as it stores bootstrap values inside square brackets after branch lengths. This file usually cannot be parsed by traditional Newick parser, such as `ape::read.tree`. The function `read.raxml` can read such file and stored the bootstrap as an additional features, which can be used to display on the tree or used to color tree branches, *etc.*.

```
raxml_file <- system.file("extdata/RAxML",
        "RAxML_bipartitionsBranchLabels.H3",
        package="treeio")
raxml <- read.raxml(raxml_file)
raxml
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/RAxML/RAxML_bipartitionsBranchLa
##
## ...@ phylo:
## Phylogenetic tree with 64 tips and 62 internal nodes.
##
## Tip labels:
##  A_Hokkaido_M1_2014_H3N2_2014, A_Czech_Republic_1_2014_H3N2_2014, FJ5
##
## Unrooted; includes branch lengths.
##
## with the following features available:
##  'bootstrap'.
```

1.3.2.8 Parsing NHX tree

NHX (New Hampshire eXtended) format is an extension of Newick by introducing NHX tags. NHX is commonly used in phylogenetics software (including PHYLDOG (http://pbil.univ-lyon1.fr/software/phyldog/) (Boussau et al. 2013 (page 275)), RevBayes (http://revbayes.github.io/intro.html) (Höhna et al. 2014 (page 277))) for storing statistical inferences. The following codes imported a NHX tree with associated data inferred by PHYLDOG.

```
nhxfile <- system.file("extdata/NHX", "phyldog.nhx",
        package="treeio")
nhx <- read.nhx(nhxfile)
nhx
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/NHX/phyldog.nhx'.
##
## ...@ phylo:
## Phylogenetic tree with 16 tips and 15 internal nodes.
##
## Tip labels:
##  Prayidae_D27SS7@2825365, Kephyes_ovata@2606431, Chuniphyes_-
```

```
multidentata@1277217, Apolemia_sp_@1353964, Bargmannia_amoena@263997, Ba
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'Ev',   'S',    'ND'.
```

1.3.2.9 Parsing Phylip tree

Phylip format contains multiple sequence alignment of taxa in Phylip sequence format with corresponding Newick tree text that was built from taxon sequences. Multiple sequence alignment can be sorted based on the tree structure and displayed at the right hand side of the tree using ggtree (http://bioconductor.org/packages/ggtree) (see also the msaplot function (page 180)).

```
phyfile <- system.file("extdata", "sample.phy",
        package="treeio")
phylip <- read.phylip(phyfile)
phylip
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/sample.phy'.
##
## ...@ phylo:
## Phylogenetic tree with 15 tips and 13 internal nodes.
##
## Tip labels:
##  K, N, D, L, J, G, ...
##
## Unrooted; no branch lengths.
```

1.3.2.10 Parsing EPA and pplacer output

EPA (http://sco.h-its.org/exelixis/web/software/epa/index.html) (Berger, Krompass, and Stamatakis 2011 (page 275)b (page 275)) and PPLACER (htt

(Frederick A Matsen, Kodner, and Armbrust 2010 (page 279)) have common output file format, `jplace`, which can be parsed by `read.jplace()` function.

```
jpf <- system.file("extdata/EPA.jplace",  package="treeio")
jp <- read.jplace(jpf)
print(jp)
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/EPA.jplace'.
##
## ...@ phylo:
## Phylogenetic tree with 493 tips and 492 internal nodes.
##
## Tip labels:
##  CIR000447A, CIR000479, CIR000078, CIR000083, CIR000070, CIR000060, ...
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'nplace'.
```

The number of evolutionary placement on each branch will be calculated and stored as the *nplace* feature, which can be mapped to line size and/or color using ggtree (https://bioconductor.org/packages/ggtree/) (Yu et al. 2017).

1.3.2.11 Parsing jtree format

The *jtree* is a JSON based format that was defined in this treeio (https://bioconductor.org/packages/treeio/) package to support tree data inter change (see session 3.3 (page 82)). Phylogenetic tree with associated data can be exported to a single *jtree* file using `write.jtree` function. The *jtree* can be easily parsed using any JSON parser. The *jtree* format contains three keys: tree, data and metadata. The tree value contains tree text extended from Newick tree format by putting the edge number in curly braces after branch length. The data value contains node/branch-specific data, while metadata value contains additional meta information.

```
jtree_file <- tempfile(fileext = '.jtree')
write.jtree(beast, file = jtree_file)
read.jtree(file = jtree_file)
```

```
## 'treedata' S4 object that stored information of
##  '/tmp/RtmpVHT6rV/file4d1a12a5c9b4.jtree'.
##
## ...@ phylo:
## Phylogenetic tree with 15 tips and 14 internal nodes.
##
## Tip labels:
##  K_2013, N_2010, D_1987, L_1980, J_1983, G_1992, ...
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'height',   'height_0.95_HPD',  'height_range', 'length',
##  'length_0.95_HPD',  'length_median',    'length_range', 'rate',
##  'rate_0.95_HPD',    'rate_median',  'rate_range',
##  'height_median',    'posterior'.
```

1.3.3 Getting information from *treedata* object

After the tree was imported, users may want to extract information that stored in the `treedata` object. treeio (http://bioconductor.org/packages/treeio) provides several accessor methods to extract tree structure, features/attributes that stored in the object and their corresponding values.

The `get.tree` or `as.phylo` methods can convert the `treedata` object to `phylo` object which is the fundamental tree object in the R community and many packages work with `phylo` object.

```
beast_file <- system.file("examples/MCC_FluA_H3.tree",
         package="ggtree")
beast_tree <- read.beast(beast_file)
```

```
# or get.tree
as.phylo(beast_tree)
```

```
##
## Phylogenetic tree with 76 tips and 75 internal nodes.
##
## Tip labels:
##  A/Hokkaido/30-1-a/2013, A/New_York/334/2004, A/New_York/463/2005, A/
##
## Rooted; includes branch lengths.
```

The `get.fields` method return a vector of features/attributes that stored in the object and associated with the phylogeny.

```
get.fields(beast_tree)
```

```
##  [1] "height"          "height_0.95_HPD" "height_median"
##  [4] "height_range"    "length"          "length_0.95_HPD"
##  [7] "length_median"   "length_range"    "posterior"
## [10] "rate"            "rate_0.95_HPD"   "rate_median"
## [13] "rate_range"
```

The `get.data` method return a tibble of all the associated data.

```
get.data(beast_tree)
```

```
## # A tibble: 151 x 14
##    height height_0.95_HPD height_median height_range length
##     <dbl> <list>                  <dbl> <list>        <dbl>
##  1   19   <dbl [2]>                19   <dbl [2]>     2.34
##  2   17   <dbl [2]>                17   <dbl [2]>     1.18
##  3   14   <dbl [2]>                14   <dbl [2]>     0.966
##  4   12   <dbl [2]>                12   <dbl [2]>     1.87
##  5    9   <dbl [2]>                 9   <dbl [2]>     2.93
##  6   10   <dbl [2]>                10   <dbl [2]>     0.827
##  7   10   <dbl [2]>                10   <dbl [2]>     0.834
```

```
##  8   10.8 <dbl [2]>                10.8 <dbl [2]>     0.233
##  9    9   <dbl [2]>                 9   <dbl [2]>     1.28
## 10    9   <dbl [2]>                 9   <dbl [2]>     0.414
## # … with 141 more rows, and 9 more variables:
## #   length_0.95_HPD <list>, length_median <dbl>,
## #   length_range <list>, posterior <dbl>, rate <dbl>,
## #   rate_0.95_HPD <list>, rate_median <dbl>,
## #   rate_range <list>, node <int>
```

If users are only interesting a subset of the features/attributes return by `get.fields`, they can extract the information from the output of `get.data` or directly subset the data by `[` or `[[`.

```
beast_tree[, c("node", "height")]
```

```
## # A tibble: 151 x 2
##     node height
##    <int>  <dbl>
##  1    10   19
##  2     9   17
##  3    36   14
##  4    31   12
##  5    29    9
##  6    28   10
##  7    39   10
##  8    90   10.8
##  9    16    9
## 10     2    9
## # … with 141 more rows
```

```
head(beast_tree[["height_median"]])
```

```
## [1] 19 17 14 12  9 10
```

1.4 Summary

Software tools for inferring molecular evolution (*e.g.*, ancestral states, molecular dating and selection pressure, *etc.*) are proliferating, but there is no single data format that is used by all different programs and capable to store different types of phylogenetic data. Most of the software packages have their own unique output formats and these formats are not compatible with each other. Parsing software outputs is challenging, which restricts the joint analysis using different tools. The treeio (http://bioconductor.org/packages/treeio) provides a set of functions (Table 1.1) for parsing various types of phylogenetic data files. These phylogenetic data can be integrated together that allows further exploration and comparison. Until now, most software tools in field of molecular evolution are isolated and often not fully compatible with each other for input and output files. All these software tools are designed to do their own analysis and the outputs are often not readable in other software. There is no software tool that was particularly designed to unify the inference data from different analysis programs. An efficient incorporation of the different inference data can compare and verify between methods, and comprehend the understanding of the study target at different aspects and angles, which may discover new systematic patterns and bring new insights to generate new hypothesis.

As phylogenetic trees are growing in its application to identify patterns in evolutionary context, more different disciplines are employing phylogenetic trees in their research. For example, spatial ecologists may map the geographical positions of the organisms to their phylogenetic trees to understand the biogeography of the species (Schön et al. 2015 (page 281)); disease epidemiologists may incorporate the pathogen sampling time and locations into the phylogenetic analysis to infer the disease transmission dynamics in spatiotemporal space (He et al. 2013 (page 277)); microbiologists may determine the pathogenicity of different pathogen strains and map them into their phylogenetic trees to identify the genetic determinants of the pathogenicity (Bosi et al. 2016 (page 275)); genomic scientists may use the phylogenetic trees to help taxonomically

classify their metagenomic sequence data (Gupta and Sharma 2015 (page 277)). A robust tool such as treeio (http://bioconductor.org/packages/treeio) to import and map different types of data into the phylogenetic tree is important to facilitate these phylogenetics-related research, or *a.k.a* 'phylodynamics'. Such tool could also help integrate different metadata (time, geography, genotype, epidemiological information) and analysis results (selective pressure, evolutionary rates) at the highest level and provide a comprehensive understanding of the study organisms. In the field of influenza research, there have been such attemps of studying phylodynamics of the influenza virus by mapping different meta-data and analysis results on the same phylogenetic tree and evolutionary time scale (Lam et al. 2015 (page 278)).

2

Manipulating Tree with Data

2.1 Manipulating tree data using tidy interface

All the tree data parsed/merged by treeio (https://bioconductor.org/packages/treeio/) can be converted to tidy data frame using the tidytree (https://cran.r-project.org/package=tidytree) package. The tidytree (https://cran.r-project.org/package=tidytree) package provides tidy interfaces to manipulate tree with associated data. For instances, external data can be linked to phylogeny or evolutionary data obtained from different sources can be merged using tidyverse verbs. After the tree data was manipulated, it can be converted back to `treedata` object and exported to a single tree file (page 71), further analyzed in R or visualized using ggtree (https://bioconductor.org/packages/ggtree/) (Yu et al. 2017).

2.1.1 The `phylo` object

The `phylo` class defined in ape (https://cran.r-project.org/package=ape) is fundamental for phylogenetic analysis in R. Most of the R packages in this field (https://CRAN.R-project.org/view=Phylogenetics) rely extensively on `phylo` object. The tidytree (https://CRAN.R-project.org/package=tidytree) package provides `as_tibble` method to convert the `phylo` object to tidy data frame, a `tbl_tree` object.

```
library(ape)
set.seed(2017)
tree <- rtree(4)
tree
```

```
##
## Phylogenetic tree with 4 tips and 3 internal nodes.
##
## Tip labels:
## [1] "t4" "t1" "t3" "t2"
##
## Rooted; includes branch lengths.
```

```
x <- as_tibble(tree)
x
```

```
## # A tibble: 7 x 4
##   parent  node branch.length label
##    <int> <int>         <dbl> <chr>
## 1      5     1       0.435   t4
## 2      7     2       0.674   t1
## 3      7     3       0.00202 t3
## 4      6     4       0.0251  t2
## 5      5     5      NA       <NA>
## 6      5     6       0.472   <NA>
## 7      6     7       0.274   <NA>
```

The `tbl_tree` object can be converted back to a `phylo` object.

```
as.phylo(x)
```

```
##
## Phylogenetic tree with 4 tips and 3 internal nodes.
##
## Tip labels:
## [1] "t4" "t1" "t3" "t2"
##
## Rooted; includes branch lengths.
```

Using `tbl_tree` object makes tree and data manipulation more effective and easier. For example, we can link evolutionary trait to phylogeny using dplyr (https://CRAN.R-project.org/package=dplyr) verbs `full_join`:

```
d <- tibble(label = paste0('t', 1:4),
            trait = rnorm(4))

y <- full_join(x, d, by = 'label')
y
```

```
## # A tibble: 7 x 5
##   parent  node branch.length label  trait
##    <int> <int>         <dbl> <chr>  <dbl>
## 1      5     1       0.435   t4     0.943
## 2      7     2       0.674   t1    -0.171
## 3      7     3       0.00202 t3     0.570
## 4      6     4       0.0251  t2    -0.283
## 5      5     5      NA       <NA>  NA
## 6      5     6       0.472   <NA>  NA
## 7      6     7       0.274   <NA>  NA
```

2.1.2 The `treedata` object

The tidytree (https://CRAN.R-project.org/package=tidytree) package defines `treedata` class to store phylogenetic tree with associated data. After mapping external data to the tree structure, the `tbl_tree` object can be converted to a `treedata` object.

```
as.treedata(y)
```

```
## 'treedata' S4 object'.
##
## ...@ phylo:
## Phylogenetic tree with 4 tips and 3 internal nodes.
##
## Tip labels:
```

```
## [1] "t4" "t1" "t3" "t2"
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'trait'.
```

The `treedata` class is also used in treeio (https://bioconductor.org/packages/treeio/) package to store evolutionary evidences inferred by commonly used software (BEAST, EPA, HYPHY, MrBayes, PAML, PHYLDOG, pplacer, r8s, RAxML and RevBayes, etc.) (see details in Chapter 1 (page 7)).

The tidytree (https://CRAN.R-project.org/package=tidytree) package also provides `as_tibble` to convert `treedata` object to a tidy data frame. The phylogentic tree structure and the evolutionary inferences were stored in the `tbl_tree` object, making it consistent and easier for manipulating evolutionary statistics inferred by different software as well as linking external data to the same tree structure.

```
y %>% as.treedata %>% as_tibble
```

```
## # A tibble: 7 x 5
##   parent  node branch.length label  trait
##    <int> <int>         <dbl> <chr>  <dbl>
## 1      5     1       0.435   t4     0.943
## 2      7     2       0.674   t1    -0.171
## 3      7     3       0.00202 t3     0.570
## 4      6     4       0.0251  t2    -0.283
## 5      5     5      NA       <NA>  NA
## 6      5     6       0.472   <NA>  NA
## 7      6     7       0.274   <NA>  NA
```

2.1.3 Access related nodes

dplyr (https://cran.r-project.org/package=dplyr) verbs can be applied to `tbl_tree` directly to manipulate tree data. In addition, tidytree (https://CRAN.R-project.org/package=tidytree)

provides several verbs to filter related nodes, including `child`, `parent`, `offspring`, `ancestor`, `sibling` and `MRCA`.

These verbs accept a `tbl_tree` and a selected node which can be node number or label.

```
child(y, 5)
```

```
## # A tibble: 2 x 5
##   parent  node branch.length label  trait
##    <int> <int>         <dbl> <chr>  <dbl>
## 1      5     1         0.435 t4     0.943
## 2      5     6         0.472 <NA>  NA
```

```
parent(y, 2)
```

```
## # A tibble: 1 x 5
##   parent  node branch.length label trait
##    <int> <int>         <dbl> <chr> <dbl>
## 1      6     7         0.274 <NA>     NA
```

```
offspring(y, 5)
```

```
## # A tibble: 6 x 5
##   parent  node branch.length label  trait
##    <int> <int>         <dbl> <chr>  <dbl>
## 1      5     1       0.435   t4     0.943
## 2      7     2       0.674   t1    -0.171
## 3      7     3       0.00202 t3     0.570
## 4      6     4       0.0251  t2    -0.283
## 5      5     6       0.472   <NA>  NA
## 6      6     7       0.274   <NA>  NA
```

```
ancestor(y, 2)
```

```
## # A tibble: 3 x 5
##   parent  node branch.length label trait
##    <int> <int>         <dbl> <chr> <dbl>
## 1      5     5        NA     <NA>     NA
## 2      5     6         0.472 <NA>     NA
## 3      6     7         0.274 <NA>     NA
```

```
sibling(y, 2)
```

```
## # A tibble: 1 x 5
##   parent  node branch.length label trait
##    <int> <int>         <dbl> <chr> <dbl>
## 1      7     3       0.00202 t3    0.570
```

```
MRCA(y, 2, 3)
```

```
## # A tibble: 1 x 5
##   parent  node branch.length label trait
##    <int> <int>         <dbl> <chr> <dbl>
## 1      6     7         0.274 <NA>     NA
```

All these methods also implemented in treeio (http://bioconductor.org/packages/treeio) for working with `phylo` and `treedata` objects. You can try accessing related nodes using the tree object. For instance, the following command will output child nodes of the selected internal node `5`:

```
child(tree, 5)
```

```
## [1] 1 6
```

Beware that the methods work for tree objects output related node numbers, while the methods implemented for `tbl_tree` object output a `tibble` object that contains related information.

2.2 Data Integration

2.2.1 Combining tree data

The treeio (https://bioconductor.org/packages/treeio/) package serves as an infrastructure that enables various types of phylogenetic data inferred from common analysis programs to be imported and used in R. For instance d_N/d_S or ancestral sequences estimated by CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html), and clade support values (posterior) inferred by BEAST (http://beast2.org/)/MrBayes (http://nbisweden.github.io/MrBayes/). In addition, treeio (https://bioconductor.org/packages/treeio/) package supports linking external data to phylogeny. It brings these external phylogenetic data (either from software output or exteranl sources) to the R community and make it available for further analysis in R. Furthermore, treeio (https://bioconductor.org/packages/treeio/) can combine multiple phylogenetic trees together into one with their node/branch-specific attribute data. Essentially, as a result, one such attribute (*e.g.*, substitution rate) can be mapped to another attribute (*e.g.*, d_N/d_S) of the same node/branch for comparison and further computations (Yu et al. 2017).

A previously published data set, seventy-six H3 hemagglutinin gene sequences of a lineage containing swine and human influenza A viruses (Liang et al. 2014 (page 278)), was here to demonstrate the utilities of comparing evolutionary statistics inferred by different software. The dataset was re-analyzed by BEAST (http://beast2.org/) for timescale estimation and CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html) for synonymous and non-synonymous substitution estimation. In this example, we first parsed the outputs from BEAST (http://beast2.org/) using

`read.beast` and from CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html) using `read.codeml` into two `treedata` objects. Then the two objects containing separate sets of node/branch-specific data were merged via the `merge_tree` function.

```
beast_file <- system.file("examples/MCC_FluA_H3.tree",
         package="ggtree")
rst_file <- system.file("examples/rst", package="ggtree")
mlc_file <- system.file("examples/mlc", package="ggtree")
beast_tree <- read.beast(beast_file)
codeml_tree <- read.codeml(rst_file, mlc_file)

merged_tree <- merge_tree(beast_tree, codeml_tree)
merged_tree
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/ggtree/examples/MCC_FluA_H3.tree',
##  '/home/ygc/R/library/ggtree/examples/rst',
##  '/home/ygc/R/library/ggtree/examples/mlc'.
##
## ...@ phylo:
## Phylogenetic tree with 76 tips and 75 internal nodes.
##
## Tip labels:
##  A/Hokkaido/30-1-a/2013, A/New_York/334/2004, A/New_York/463/2005, A/
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'height',   'height_0.95_HPD',  'height_median',
##  'height_range', 'length',   'length_0.95_HPD',
##  'length_median',    'length_range', 'posterior',    'rate',
##  'rate_0.95_HPD',    'rate_median',  'rate_range',   'subs',
##  'AA_subs',  't',    'N',    'S',    'dN_vs_dS', 'dN',   'dS',   'N_x
##  'S_x_dS'.
```

After merging the `beast_tree` and `codeml_tree` objects, all node/branch-specific data imported from BEAST (http://beast2.org/) and CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html) output files are all available in the

merged_tree object. The tree object was converted to tidy data frame using tidytree (https://cran.r-project.org/package=tidytree) package and visualized as hexbin scatterplot of d_N/d_S, d_N and d_S inferred by CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html) versus *rate* (substitution rate in unit of substitutions/site/year) inferred by BEAST (http://beast2.org/) on the same branches.

```
library(dplyr)
df <- merged_tree %>%
  as_tibble() %>%
  select(dN_vs_dS, dN, dS, rate) %>%
  subset(dN_vs_dS >=0 & dN_vs_dS <= 1.5) %>%
  tidyr::gather(type, value, dN_vs_dS:dS)
df$type[df$type == 'dN_vs_dS'] <- 'dN/dS'
df$type <- factor(df$type, levels=c("dN/dS", "dN", "dS"))
ggplot(df, aes(rate, value)) + geom_hex() +
  facet_wrap(~type, scale='free_y')
```

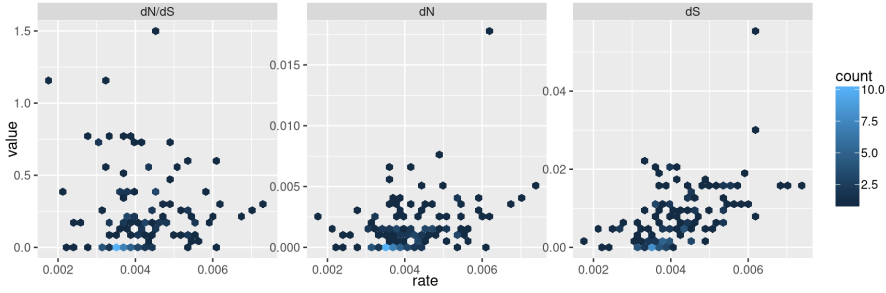


Figure 2.1: **Correlation of d_N/d_S, d_N and d_S versus substitution rate.** After merging the *BEAST* and *CodeML* outputs, the branch-specific estimates (substitution rate, d_N/d_S , d_N and d_S) from the two analysis programs are compared on the same branch basis. The associations of d_N/d_S, d_N and d_S vs. *rate* are visualized in hexbin scatter plots.

The output is illustrated in Fig. 2.1. We can then test the association of these node/branch-specific data using Pearson correlation, which in this case showed that d_N and d_S, but not d_N/d_S are significantly (p-values) associated with *rate*.

Using `merge_tree`, we are able to compare analysis results using identical model from different software packages or different models using different or identical software. It also allows users to integrate different analysis finding from different software packages. Merging tree data is not restricted to software findings, associating external data to analysis findings is also granted. The `merge_tree` function is chainable and allows several tree objects to be merged into one.

```
phylo <- as.phylo(beast_tree)
N <- Nnode2(phylo)
d <- tibble(node = 1:N, fake_trait = rnorm(N), another_trait
        = runif(N))
fake_tree <- treedata(phylo = phylo, data = d)
triple_tree <- merge_tree(merged_tree, fake_tree)
triple_tree
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/ggtree/examples/MCC_FluA_H3.tree',
##  '/home/ygc/R/library/ggtree/examples/rst',
##  '/home/ygc/R/library/ggtree/examples/mlc'.
##
## ...@ phylo:
## Phylogenetic tree with 76 tips and 75 internal nodes.
##
## Tip labels:
##  A/Hokkaido/30-1-a/2013, A/New_York/334/2004, A/New_York/463/2005, A/
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'height',   'height_0.95_HPD',  'height_median',
##  'height_range', 'length',   'length_0.95_HPD',
##  'length_median',    'length_range', 'posterior',    'rate',
##  'rate_0.95_HPD',    'rate_median',  'rate_range',   'subs',
##  'AA_subs',  't',    'N',    'S',    'dN_vs_dS', 'dN',   'dS',   'N_x
##  'S_x_dS',   'fake_trait',   'another_trait'.
```

The `triple_tree` object showed above contains analysis results obtained from BEAST (http://beast2.org/) and CODEML (http://abacus.gene.ucl.ac.uk/software/paml.html), and evolutionary trait from external sources. All these information can be used to annotate the tree using ggtree (https://bioconductor.org/packages/ggtree/) (Yu et al. 2017).

2.2.2 Linking external data to phylogeny

In addition to analysis findings that are associated with the tree as we showed above, there is a wide range of heterogeneous data, including phenotypic data, experimental data and clinical data *etc.*, that need to be integrated and linked to phylogeny. For example, in the study of viral evolution, tree nodes may associated with epidemiological information, such as location, age and subtype. Functional annotations may need to be mapped on gene trees for comparative genomics studies. To facilitate data integration, treeio (https://bioconductor.org/packages/treeio) provides `full_join` method to link external data to phylogeny and stored in either `phylo` or `treedata` object. Beware that linking external data to a `phylo` object will produce a `treedata` object to store the input `phylo` with associated data. The `full_join` methods can also be used at tidy data frame level (*i.e.* `tbl_tree` object described previously) and at `ggtree` level (described in session 7.1 (page 172)) (Yu et al. 2018 (page 282)).

The following example calculated bootstrap values and merging that values with the tree (a `phylo` object) by matching their node numbers.

```
library(ape)
data(woodmouse)
d <- dist.dna(woodmouse)
tr <- nj(d)
bp <- boot.phylo(tr, woodmouse, function(x) nj(dist.dna(x)))
```

```
##
Running bootstraps:       100 / 100
## Calculating bootstrap values... done.
```

```
bp2 <- tibble(node=1:Nnode(tr) + Ntip(tr), bootstrap = bp)
full_join(tr, bp2, by="node")
```

```
## 'treedata' S4 object'.
##
## ...@ phylo:
## Phylogenetic tree with 15 tips and 13 internal nodes.
##
## Tip labels:
##  No305, No304, No306, No0906S, No0908S, No0909S, ...
##
## Unrooted; includes branch lengths.
##
## with the following features available:
##  'bootstrap'.
```

Another example demonstrates merging evolutionary trait with the tree (a `treedata` object) by matching their tip labels.

```
file <- system.file("extdata/BEAST", "beast_mcc.tree",
        package="treeio")
beast <- read.beast(file)
x <- tibble(label = as.phylo(beast)$tip.label, trait =
        rnorm(Ntip(beast)))
full_join(beast, x, by="label")
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/BEAST/beast_mcc.tree'.
##
## ...@ phylo:
## Phylogenetic tree with 15 tips and 14 internal nodes.
##
## Tip labels:
##  A_1995, B_1996, C_1995, D_1987, E_1996, F_1997, ...
##
## Rooted; includes branch lengths.
##
```

```
## with the following features available:
##  'height',   'height_0.95_HPD',  'height_median',
##  'height_range', 'length',   'length_0.95_HPD',
##  'length_median',    'length_range', 'posterior',    'rate',
##  'rate_0.95_HPD',    'rate_median',  'rate_range',   'trait'.
```

Manipulating tree object is frustrated with the fragmented functions available for working with `phylo` object, not to mention linking external data to the phylogeny structure. With treeio (http://bioconductor.org/packages/treeio) package, it is easy to combine tree data from various sources. In addition, with tidytree (https://CRAN.R-project.org/package=tidytree) package (session 2.1 (page 43)), manipulating tree is more easier using tidy data principles (https://www.jstatsoft.org/article/view/v059i10) and consistent with tools already in wide use, including dplyr (https://cran.r-project.org/package=dplyr), tidyr (https://cran.r-project.org/package=tidyr), ggplot2 (https://cran.r-project.org/package=ggplot2) and ggtree (https://bioconductor.org/packages/ggtree/).

2.2.3 Grouping taxa

The `groupOTU` and `groupClade` methods are designed for adding taxa grouping information to the input tree object. The methods were implemented in tidytree (https://CRAN.R-project.org/package=tidytree), treeio (http://bioconductor.org/packages/treeio) and ggtree (http://bioconductor.org/packages/ggtree) respectively to support adding grouping information at `tbl_tree`, `phylo` and `treedata`, and `ggtree` levels. These grouping information can be used directly in tree visualization (*e.g.* coloring tree based on grouping (page 162)) with ggtree (https://bioconductor.org/packages/ggtree/) (Figure 6.5).

2.2.3.1 groupClade

The `groupClade` method accepts an internal node or a vector of internal nodes to add grouping information of clade/clades.

```
nwk <- '(((((((A:4,B:4):6,C:5):8,D:6):3,E:21):10,
         ((F:4,G:12):14,H:8):13):13,((I:5,J:2):30,
         (K:11,L:11):2):17):4,M:56);'
tree <- read.tree(text=nwk)

groupClade(as_tibble(tree), c(17, 21))
```

```
## # A tibble: 25 x 5
##    parent  node branch.length label group
##     <int> <int>         <dbl> <chr> <fct>
##  1     20     1             4 A     1
##  2     20     2             4 B     1
##  3     19     3             5 C     1
##  4     18     4             6 D     1
##  5     17     5            21 E     1
##  6     22     6             4 F     2
##  7     22     7            12 G     2
##  8     21     8             8 H     2
##  9     24     9             5 I     0
## 10     24    10             2 J     0
## # … with 15 more rows
```

2.2.3.2 groupOTU

```
set.seed(2017)
tr <- rtree(4)
x <- as_tibble(tr)
## the input nodes can be node ID or label
groupOTU(x, c('t1', 't4'), group_name = "fake_group")
```

```
## # A tibble: 7 x 5
##   parent  node branch.length label fake_group
##    <int> <int>         <dbl> <chr> <fct>
## 1      5     1       0.435   t4    1
## 2      7     2       0.674   t1    1
## 3      7     3       0.00202 t3    0
```

```
## 4      6     4       0.0251  t2    0
## 5      5     5      NA       <NA>  1
## 6      5     6       0.472   <NA>  1
## 7      6     7       0.274   <NA>  1
```

Both `groupClade()` and `groupOTU()` work with `tbl_tree`, `phylo` and `treedata`, and `ggtree` objects. Here is an example of using `groupOTU()` with `phylo` tree object.

```
groupOTU(tr, c('t1', 't4'), group_name = "fake_group") %>%
  as_tibble
```

```
## # A tibble: 7 x 5
##   parent  node branch.length label fake_group
##    <int> <int>         <dbl> <chr> <fct>
## 1      5     1       0.435   t4    1
## 2      7     2       0.674   t1    1
## 3      7     3       0.00202 t3    0
## 4      6     4       0.0251  t2    0
## 5      5     5      NA       <NA>  1
## 6      5     6       0.472   <NA>  1
## 7      6     7       0.274   <NA>  1
```

Another example of working with `ggtree` object can be found in session 6.5 (page 162).

The `groupOTU` will trace back from input nodes to most recent common ancestor. In this example, nodes 2, 3, 7 and 6 (`2 (t1) -> 7 -> 6` and `3 (t4) -> 6`) are grouping together.

Related OTUs are grouping together and they are not necessarily within a clade. They can be monophyletic (clade), polyphyletic or paraphyletic.

```
cls <- list(c1=c("A", "B", "C", "D", "E"),
            c2=c("F", "G", "H"),
            c3=c("L", "K", "I", "J"),
            c4="M")
```

```
as_tibble(tree) %>% groupOTU(cls)
```

```
## # A tibble: 25 x 5
##    parent  node branch.length label group
##     <int> <int>         <dbl> <chr> <fct>
##  1     20     1             4 A     c1
##  2     20     2             4 B     c1
##  3     19     3             5 C     c1
##  4     18     4             6 D     c1
##  5     17     5            21 E     c1
##  6     22     6             4 F     c2
##  7     22     7            12 G     c2
##  8     21     8             8 H     c2
##  9     24     9             5 I     c3
## 10     24    10             2 J     c3
## # … with 15 more rows
```

If there are conflicts when tracing back to mrca, user can set `overlap` parameter to "origin" (the first one counts), "overwrite" (default, the last one counts) or "abandon" (un-selected for grouping)[8].

2.3 Rescaling Tree Branches

Phylogenetic data can be merged for joint analysis (Figure 2.1). They can be displayed on the same tree structure as more complex annotation to help visually inspection of their evolutionary patterns. All the numerical data stored in `treedata` object can be used to re-scale tree branches. For example, CodeML infers d_N/d_S, d_N and d_S, all these statistics can be used as branch lengths. All these values can also be used to color the tree (session 4.3.4 (page 125)) and can be project to vertical dimension to create two-dimensional tree or phenogram (session 4.2.2 (page 0) and Figure 4.5 and 4.11).

```
p1 <- ggtree(merged_tree) + theme_tree2()
p2 <- ggtree(rescale_tree(merged_tree, 'dN')) +
        theme_tree2()
p3 <- ggtree(rescale_tree(merged_tree, 'rate')) +
        theme_tree2()

cowplot::plot_grid(p1, p2, p3, ncol=3, labels =
        LETTERS[1:3])
```

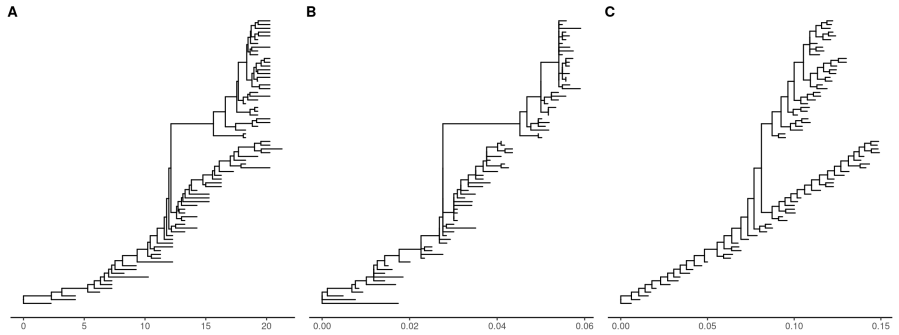


Figure 2.2: **Re-scaling tree branches.** The tree with branches scaled in time (year from the root) (A). The tree was re-scaled using d_N as branch lengths (B). The tree was re-scaled using substitution rates (C).

2.4 Subsetting Tree with Data

2.4.1 Removing tips in a phylogenetic tree

Sometimes we want to remove selected tips from a phylogenetic tree. This is due to several reasons, including low sequence quality, errors in sequence assembly, an alignment error in part of the sequence and an error in phylogenetic inference *etc*.

Let's say that we want to remove three tips (colored by red) from the tree (Figure 2.3A), the `drop.tip()` method removes specified tips and update

tree (Figure 2.3B). All associated data will be maintained in the updated tree.

```
f <- system.file("extdata/NHX", "phyldog.nhx",
        package="treeio")
nhx <- read.nhx(f)
to_drop <- c("Physonect_sp_@2066767",
            "Lychnagalma_utricularia@2253871",
            "Kephyes_ovata@2606431")
p1 <- ggtree(nhx) + geom_tiplab(aes(color = label %in%
        to_drop)) +
  scale_color_manual(values=c("black", "red")) + xlim(0,
        0.8)

nhx_reduced <- drop.tip(nhx, to_drop)
p2 <- ggtree(nhx_reduced) + geom_tiplab() + xlim(0, 0.8)
plot_grid(p1, p2, ncol=2, labels = c("A", "B"))
```

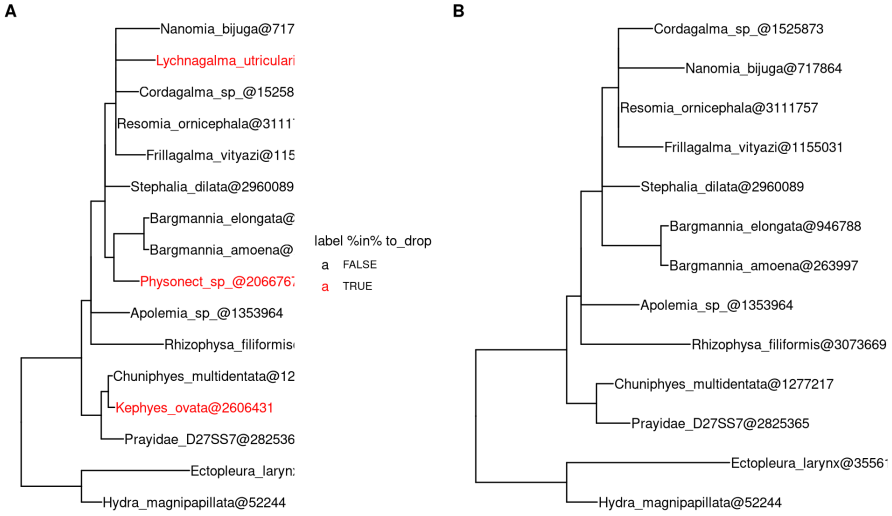


Figure 2.3: **Removing tips from tree.** Original tree with three tips (colored by red) to remove (A). Updated tree that removed selected tips (B).

2.4.2 Subsetting tree by tip label

Tree can be large and difficult to look at only the portions of interest. The `tree_subset()` function was created in treeio (http://bioconductor.org/packages/treeio) package to extract a subset of the tree portion while still maintaining the structure of the tree portion. The `beast_tree` in Figure 2.4A is slighly crowded. Obviously, we can make the figure taller to allow more space for the labels (similar to use "Expansion" slider in `FigTree`) or we can make the text smaller. However, these solutions are not always applicable when you have a lot of tips (*e.g.* hundreds or thousands of tips). In particuarly, when you are only interested in the portion of the tree around a particular tip, you certainly don't want to explore a large tree to find centain species you are interested in.

Let's say you are interested in tip *A/Swine/HK/168/2012* from the tree (Figure 2.4A) and you want to look at the immediate relatives of this tip.

The `tree_subset()` function allows for you to look at the portions of the tree that are of interest. By default, `tree_subset()` will internally call `groupOTU()` (page 56) to assign group specified tip from the rest of other tips (Figure 2.4B). Additionally, the branch lengths and related associated data are maintained after subsetting (Figure 2.4C). The root of the tree is always anchored at zero for the subset tree by default and all the distances are relative to this root. If you want all the distances are relative to the original root, you can specify the root position (by `root.position` parameter) to the root edge of the subset tree, which is the sum of branch lengths from the original root to the root of the subset tree (Figure 2.4D and E).

```
beast_file <- system.file("examples/MCC_FluA_H3.tree",
         package="ggtree")
beast_tree <- read.beast(beast_file)

p1 = ggtree(beast_tree) +
  geom_tiplab() +  xlim(0, 40) + theme_tree2()

tree2 = tree_subset(beast_tree, "A/Swine/HK/168/2012",
         levels_back=4)
```

```
p2 <- ggtree(tree2, aes(color=group)) +
  scale_color_manual(values = c("black", "red")) +
  geom_tiplab() +  xlim(0, 4) + theme_tree2()

p3 <- ggtree(tree2, aes(color=group)) +
  geom_tiplab(hjust = -.1) + xlim(0, 5) +
  geom_point(aes(fill = rate), shape = 21, size = 4) +
  scale_color_manual(values = c("black", "red"), guide =
        FALSE) +
  scale_fill_continuous(low = 'blue', high = 'red') +
  theme_tree2() + theme(legend.position = 'right')


p4 <- ggtree(tree2, aes(color=group),
          root.position = as.phylo(tree2)$root.edge) +
  geom_tiplab() + xlim(18, 24) +
  scale_color_manual(values = c("black", "red")) +
  theme_tree2()

p5 <- ggtree(tree2, aes(color=group),
          root.position = as.phylo(tree2)$root.edge) +
  geom_rootedge() + geom_tiplab() + xlim(0, 40) +
  scale_color_manual(values = c("black", "red")) +
  theme_tree2()

plot_grid(p2, p3, p4, p5, ncol=2, labels=LETTERS[2:5]) %>%
  plot_grid(p1, ., ncol=2, labels=c("A", ""),
        rel_widths=c(.5, 1))
```

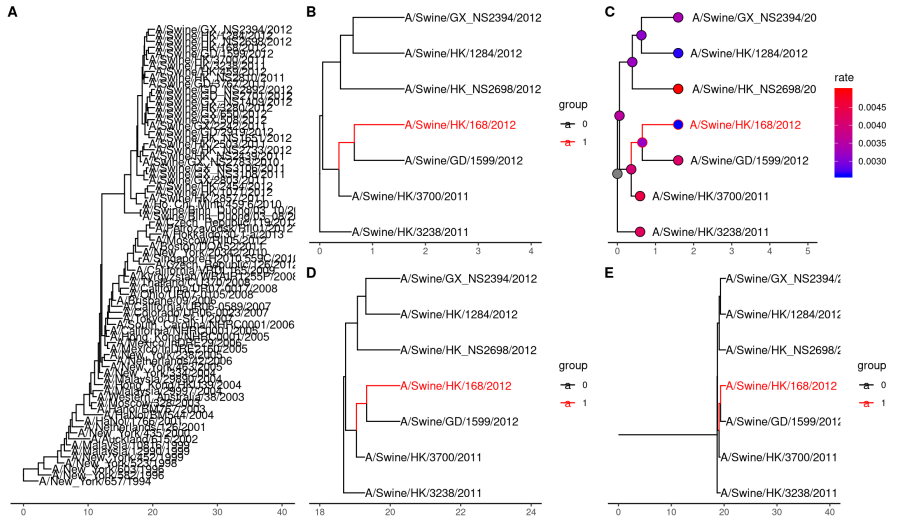


Figure 2.4: **Subsetting tree for specific tip.** The original tree (A). The subset tree (B). Subset tree with data (C). Visualize the subset tree relative to original position, without root edge (D) and with root edge (E).

2.4.3 Subsetting tree by internal node number

If you are interesting at certain clade, you can specify the input node as an internal node number. The `tree_subset()` function will take the clade as a whole and also trace back to particular levels to look at the immediate relatives of the clade (Figure 2.5A and B). We can use `tree_subset` to zoom in selected portions and plot a whole tree with the portion of it, that is similar to the `ape::zoom()` function to explore very large tree (Figure 2.5C and D).

```
clade <- tree_subset(beast_tree, node=121, levels_back=0)
clade2 <- tree_subset(beast_tree, node=121, levels_back=2)
p1 <- ggtree(clade) + geom_tiplab() + xlim(0, 5)
p2 <- ggtree(clade2, aes(color=group)) + geom_tiplab() +
  xlim(0, 8) + scale_color_manual(values=c("black", "red"))


library(ape)
library(tidytree)
```

```
library(treeio)

data(chiroptera)

nodes <- grep("Plecotus", chiroptera$tip.label)
chiroptera <- groupOTU(chiroptera, nodes)

clade <- MRCA(chiroptera, nodes)
x <- tree_subset(chiroptera, clade, levels_back = 0)

p3 <- ggtree(chiroptera, aes(colour = group)) +
  scale_color_manual(values=c("black", "red")) +
  theme(legend.position = "none")
p4 <- ggtree(x) + geom_tiplab() + xlim(0, 5)
plot_grid(p1, p2, p3, p4,
  ncol=2, labels=LETTERS[1:4])
```

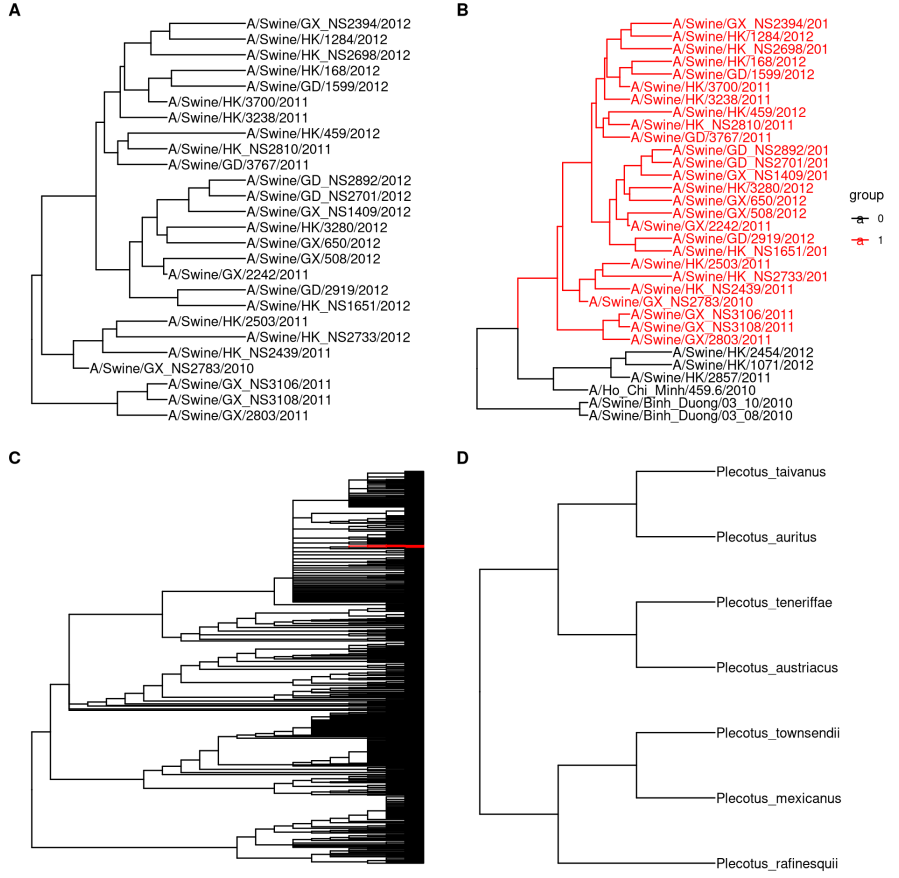


Figure 2.5: **Subsetting tree for specific clade.** Extracting a clade (A). Extracting a clade and trace back to look at its immediate relatives (B). Viewing a very large tree (C) and a selected portion of it (D).

2.5 Manipulating tree data for visualization

Tree visualization is supported by ggtree (page 107). Although *ggtree* implemented several methods for visual exploration of tree with data (page 155), you may want to do something that is not supported directly. In

this case, you need to manipulate tree data with node coordination positions that used for visualization. This is quite easy with *ggtree*. User can use `foritfy()` method which internally call `tidytree::as_tibble()` to convert the tree to tidy data frame and add columns of coordination positions (*i.e.* x, y, branch and angle) that are used to plot the tree. You can also access the data via `ggtree(tree)$data`.

Here is an example to plot two trees face to face that is similar to a `ape::cophyloplot()`.

```
library(dplyr)
library(ggtree)

x <- rtree(30)
y <- rtree(30)
p1 <- ggtree(x)
p2 <- ggtree(y)

d1 <- p1$data
d2 <- p2$data

## reverse x-axis and
## set offset to make the tree in the right hand side of the
        first tree
d2$x <- max(d2$x) - d2$x + max(d1$x) + 1

pp <- p1 + geom_tiplab() + geom_tree(data=d2) +
        geom_tiplab(data = d2, hjust=1)

dd <- bind_rows(d1, d2) %>%
  filter(!is.na(label))

pp + geom_line(aes(x, y, group=label), data=dd,
        color='grey')
```

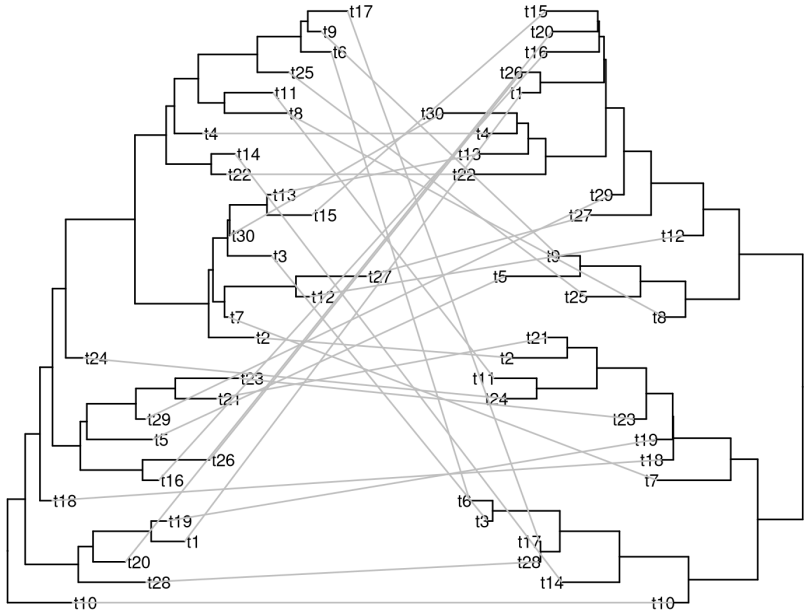


Figure 2.6: **Plot two phylogenetic trees face to face.** Plotting a tree using `ggtree()` (left hand side) and subsequently add another layer of tree by `geom_tree()` (right hand side). The relative positions of the plotted trees can be manual adjusted and adding layers to each of the tree (*e.g.* tip labels) is independent.

It is quite easy to plot multiple trees and connect taxa in one figure. For instance, plotting trees contructed from all internal gene segments of influenza virus and connecting equivalent strans across the trees (Venkatesh et al. 2018 (page 281)).

```
z <- rtree(30)
d2 <- fortify(y)
d3 <- fortify(z)
d2$x <- d2$x + max(d1$x) + 1
d3$x <- d3$x + max(d2$x) + 1

dd = bind_rows(d1, d2, d3) %>%
  filter(!is.na(label))
```

```
p1 + geom_tree(data = d2) + geom_tree(data = d3) +
        geom_tiplab(data=d3) +
  geom_line(aes(x, y, group=label, color=node < 15),
        data=dd, alpha=.3)
```

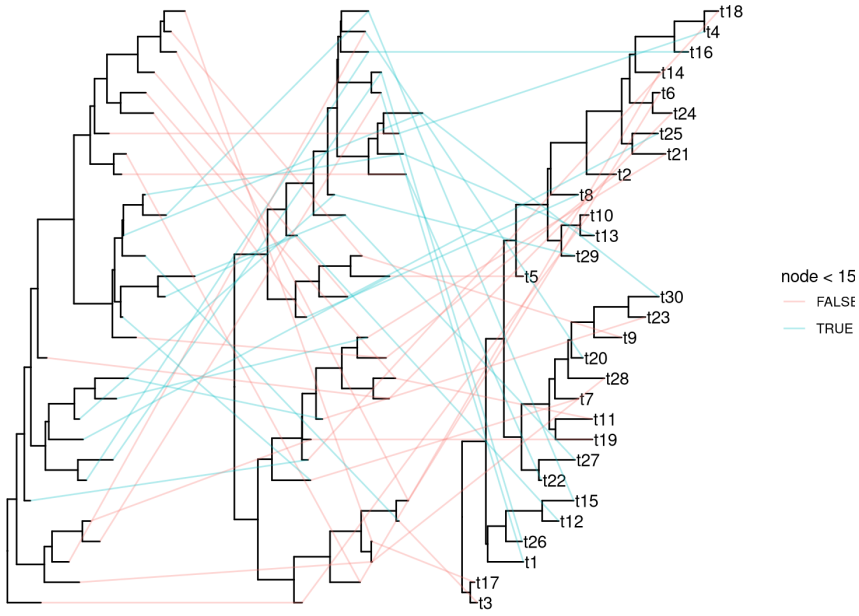


Figure 2.7: **Plot multiple phylogenetic trees side by side.** Plotting a tree using `ggtree()` and subsequently add multiple layers of trees by `geom_tree()`.

2.6 Summary

The treeio (http://bioconductor.org/packages/treeio) package allows us to import diverse phylogeny associated data into R. However, phylogenetic tree is stored in way to facilitate computational processing which is not

human fridenly and need expertise to manipulate and explore tree data. The tidytree (https://CRAN.R-project.org/package=tidytree) package provides tidy interface for exploring tree data, while ggtree (http://bioconductor.org/packages/ggtree) provides a set of utilitise to visualize and explore tree data using grammar of graphics. This full suit of packages make it easy for ordinary users to interact with tree data, and allow us to integrate phylogeny associated data from different sources (*e.g.* experimental result or analysis finding), which creates the possibility of comparative study.

3

Exporting tree with data

3.1 Introduction

The treeio (https://bioconductor.org/packages/treeio/) package supports parsing various phylogenetic tree file formats including software outputs that contain evolutionary evidences. Some of the formats are just log file (*e.g.* PAML (http://abacus.gene.ucl.ac.uk/software/paml.html) and r8s (http://ginger.ucdavis.edu/r8s) outputs), while some of the others are non-standard formats (*e.g.* BEAST (http://beast2.org/) and MrBayes (http://nbisweden.github.io/MrBayes/) outputs that introduce square bracket, which was reserved to store comment in standard Nexus format, to store inferences). With treeio (https://bioconductor.org/packages/treeio/), we are now able to parse these files to extract phylogenetic tree and map associated data on the tree structure. Exporting tree structure is easy, users can use `as.phyo` method defined treeio (https://bioconductor.org/packages/treeio/) to convert `treedata` object to `phylo` object then using `write.tree` or `write.nexus` implemented in ape (https://cran.r-project.org/web/packages/ape/index.html) package (Paradis, Claude, and Strimmer 2004 (page 280)) to export the tree structure as Newick text or Nexus file. This is quite useful for converting non-standard formats to standard format and for extracting tree from software outputs, such as log file.

However, exporting tree with associated data is still challenging. These associated data can be parsed from analysis programs or obtained from external sources (*e.g.* phenotypic data, experimental data and clinical data). The major obstacle here is that there is no standard format that designed for storing tree with data. NeXML (http://www.nexml.org/) (Vos et al. 2012) maybe the most flexible format, however it is currently not widely supported. Most of the analysis programs in this field rely

extensively on Newick string and Nexus format. In my opinion, although BEAST Nexus format[9] may not be the best solution, it is currently a good approach for storing heterogeneous associated data. The beauty of the format is that all the annotate elements are stored within square bracket, which is reserved for comments. So that the file can be parsed as standard Nexus by ignoring annotate elements and existing programs should be able to read them.

3.2 Exporting Tree Data to *BEAST* Nexus Format

3.2.1 Exporting/converting software output

The treeio (https://bioconductor.org/packages/treeio/) package provides `write.beast` to export `treedata` object as BEAST Nexus file (Bouckaert et al. 2014 (page 275)). With treeio (https://bioconductor.org/packages/treeio/), it is easy to convert software output to BEAST format if the output can be parsed by treeio (https://bioconductor.org/packages/treeio/) (see Chapter 1 (page 7)). For example, we can convert NHX file to BEAST file and use NHX tags to color the tree using FigTree[10] (Figure 3.1A) or convert CODEML output and use d_N/d_S, d_N or d_S to color the tree in FigTree (Figure 3.1B).

Here is an example of converting NHX file to BEAST format:

```
nhxfile <- system.file("extdata/NHX", "phyldog.nhx",
        package="treeio")
nhx <- read.nhx(nhxfile)
# write.beast(nhx, file = "phyldog.tree")
write.beast(nhx)
```

```
#NEXUS
[R-package treeio, Tue Oct  1 09:58:54 2019]
```

```
BEGIN TAXA;
    DIMENSIONS NTAX = 16;
    TAXLABELS
        Prayidae_D27SS7@2825365
        Kephyes_ovata@2606431
        Chuniphyes_multidentata@1277217
        Apolemia_sp_@1353964
        Bargmannia_amoena@263997
        Bargmannia_elongata@946788
        Physonect_sp_@2066767
        Stephalia_dilata@2960089
        Frillagalma_vityazi@1155031
        Resomia_ornicephala@3111757
        Lychnagalma_utricularia@2253871
        Nanomia_bijuga@717864
        Cordagalma_sp_@1525873
        Rhizophysa_filiformis@3073669
        Hydra_magnipapillata@52244
        Ectopleura_larynx@3556167
    ;
END;
BEGIN TREES;
    TRANSLATE
        1   Prayidae_D27SS7@2825365,
        2   Kephyes_ovata@2606431,
        3   Chuniphyes_multidentata@1277217,
        4   Apolemia_sp_@1353964,
        5   Bargmannia_amoena@263997,
        6   Bargmannia_elongata@946788,
        7   Physonect_sp_@2066767,
        8   Stephalia_dilata@2960089,
        9   Frillagalma_vityazi@1155031,
        10  Resomia_ornicephala@3111757,
        11  Lychnagalma_utricularia@2253871,
        12  Nanomia_bijuga@717864,
        13  Cordagalma_sp_@1525873,
        14  Rhizophysa_filiformis@3073669,
        15  Hydra_magnipapillata@52244,
        16  Ectopleura_larynx@3556167
    ;
    TREE * UNTITLED = [&R] (((1[&Ev=S,S=58,ND=0]:0.0682841,(2[&Ev=S,S=69
END;
```

Another example of converting CodeML output to BEAST format:

```
mlcfile <- system.file("extdata/PAML_Codeml", "mlc",
         package="treeio")
ml <- read.codeml_mlc(mlcfile)
# write.beast(ml, file = "codeml.tree")
write.beast(ml)
```

```
#NEXUS
[R-package treeio, Tue Oct  1 09:58:54 2019]

BEGIN TAXA;
    DIMENSIONS NTAX = 15;
    TAXLABELS
        A
        B
        C
        D
        E
        F
        G
        H
        I
        J
        K
        L
        M
        N
        O
    ;
END;
BEGIN TREES;
    TRANSLATE
        1   A,
        2   B,
        3   C,
        4   D,
        5   E,
        6   F,
```

```
7   G,
        8   H,
        9   I,
        10  J,
        11  K,
        12  L,
        13  M,
        14  N,
        15  O
    ;
    TREE * UNTITLED = [&U] (11[&t=0.082,N=1514.9,S=633.1,dN_vs_dS=0.0224
END;
```

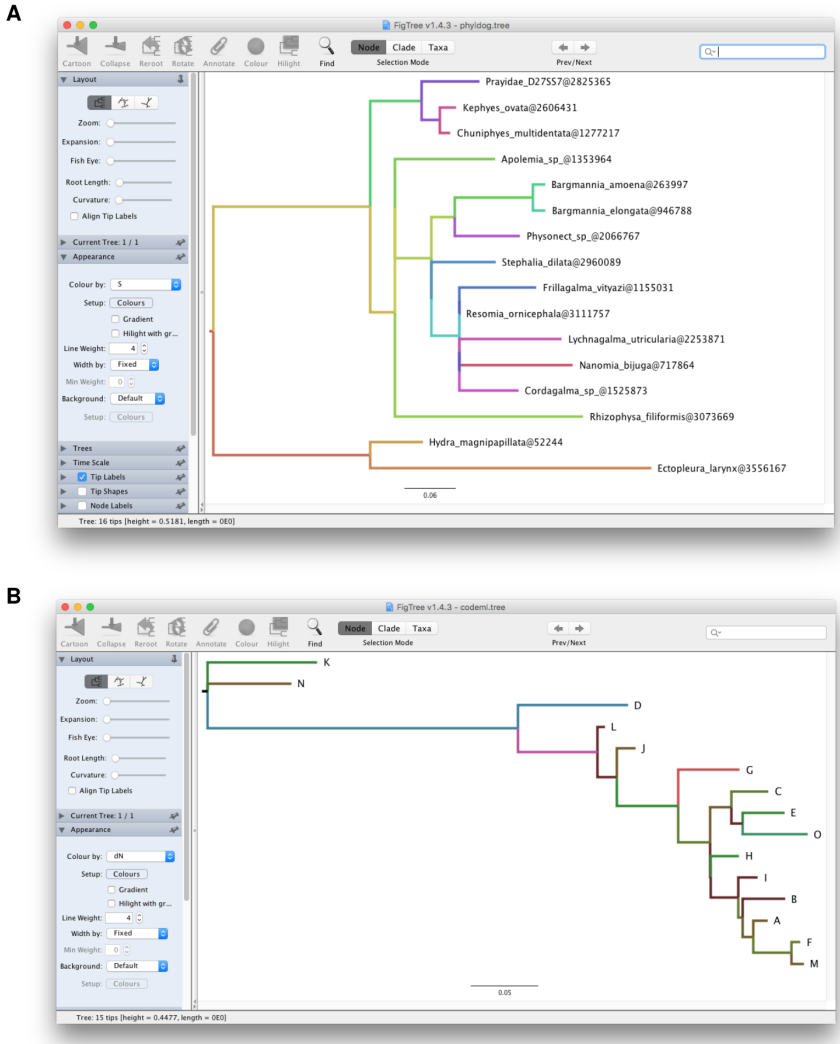


Figure 3.1: **Visualizing BEAST file in FigTree.** Directly visualizing `NHX` file (A) and `CodeML` output (B) in `FigTree` is not supported. `treeio` can convert these files to BEAST compatible NEXUS format which can be directly opened in `FigTree` and visualized annotated data.

3.2.2 Combining tree with external data

Using the utilities provided by tidytree (https://CRAN.R-project.org/package=tidytree) and treeio (https://bioconductor.org/packages/treeio/), it is easy to link external data onto the corresponding phylogeny. The `write.beast` function enable users to export the tree with external data to a single tree file.

```
phylo <- as.phylo(nhx)
## print the newick text
write.tree(phylo)
```

```
[1] "(((Prayidae_D27SS7@2825365:0.0682841,(Kephyes_ovata@2606431:0.0193
```

```
N <- Nnode2(phylo)
fake_data <- tibble(node = 1:N, fake_trait = rnorm(N),
         another_trait = runif(N))
fake_tree <- full_join(phylo, fake_data, by = "node")
write.beast(fake_tree)
```

```
#NEXUS
[R-package treeio, Tue Oct  1 09:58:57 2019]

BEGIN TAXA;
    DIMENSIONS NTAX = 16;
    TAXLABELS
        Prayidae_D27SS7@2825365
        Kephyes_ovata@2606431
        Chuniphyes_multidentata@1277217
        Apolemia_sp_@1353964
        Bargmannia_amoena@263997
        Bargmannia_elongata@946788
        Physonect_sp_@2066767
        Stephalia_dilata@2960089
        Frillagalma_vityazi@1155031
```

```
  Resomia_ornicephala@3111757
          Lychnagalma_utricularia@2253871
          Nanomia_bijuga@717864
          Cordagalma_sp_@1525873
          Rhizophysa_filiformis@3073669
          Hydra_magnipapillata@52244
          Ectopleura_larynx@3556167
      ;
  END;
  BEGIN TREES;
      TRANSLATE
          1   Prayidae_D27SS7@2825365,
          2   Kephyes_ovata@2606431,
          3   Chuniphyes_multidentata@1277217,
          4   Apolemia_sp_@1353964,
          5   Bargmannia_amoena@263997,
          6   Bargmannia_elongata@946788,
          7   Physonect_sp_@2066767,
          8   Stephalia_dilata@2960089,
          9   Frillagalma_vityazi@1155031,
          10  Resomia_ornicephala@3111757,
          11  Lychnagalma_utricularia@2253871,
          12  Nanomia_bijuga@717864,
          13  Cordagalma_sp_@1525873,
          14  Rhizophysa_filiformis@3073669,
          15  Hydra_magnipapillata@52244,
          16  Ectopleura_larynx@3556167
      ;
      TREE * UNTITLED = [&R] (((1[&fake_trait=-0.118140158479532,another_t
  END;
```

After merging, the `fake_trait` and `another_trait` stored in `fake_data` will be linked to the tree, `phylo`, and store in the `treedata` object, the `fake_tree`. The `write.beast()` function export the tree with associated data to a single BEAST format file. The associated data can be used to visualized the tree using ggtree (http://bioconductor.org/packages/ggtree) (Figure 5.7) or `FigTree` (Figure 3.1).

3.2.3 Merging tree data from different sources

Not only Newick tree text can be combined with associated data, but also tree data obtained from software output can be combined with external data, as well as different tree objects can be merged together (for details, see Chapter 2 (page 43)).

```
## combine tree object with data
tree_with_data <- full_join(nhx, fake_data, by = "node")
tree_with_data
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/NHX/phyldog.nhx'.
##
## ...@ phylo:
## Phylogenetic tree with 16 tips and 15 internal nodes.
##
## Tip labels:
##  Prayidae_D27SS7@2825365, Kephyes_ovata@2606431, Chuniphyes_multident
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'Ev',   'S',    'ND',   'fake_trait',   'another_trait'.
```

```
## merge two tree object
tree2 <- merge_tree(nhx, fake_tree)
tree2
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/treeio/extdata/NHX/phyldog.nhx'.
##
## ...@ phylo:
## Phylogenetic tree with 16 tips and 15 internal nodes.
##
## Tip labels:
```

```
##  Prayidae_D27SS7@2825365, Kephyes_ovata@2606431, Chuniphyes_multident
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'Ev',   'S',    'ND',   'fake_trait',   'another_trait'.
```

```
identical(tree_with_data, tree2)
```

```
## [1] TRUE
```

After merging data from different sources, the tree with the associated data can be exported into a single file.

```
write.beast(tree2)
```

```
#NEXUS
[R-package treeio, Tue Oct  1 09:58:57 2019]

BEGIN TAXA;
    DIMENSIONS NTAX = 16;
    TAXLABELS
        Prayidae_D27SS7@2825365
        Kephyes_ovata@2606431
        Chuniphyes_multidentata@1277217
        Apolemia_sp_@1353964
        Bargmannia_amoena@263997
        Bargmannia_elongata@946788
        Physonect_sp_@2066767
        Stephalia_dilata@2960089
        Frillagalma_vityazi@1155031
        Resomia_ornicephala@3111757
        Lychnagalma_utricularia@2253871
        Nanomia_bijuga@717864
        Cordagalma_sp_@1525873
        Rhizophysa_filiformis@3073669
```

```
Hydra_magnipapillata@52244
        Ectopleura_larynx@3556167
    ;
END;
BEGIN TREES;
    TRANSLATE
        1   Prayidae_D27SS7@2825365,
        2   Kephyes_ovata@2606431,
        3   Chuniphyes_multidentata@1277217,
        4   Apolemia_sp_@1353964,
        5   Bargmannia_amoena@263997,
        6   Bargmannia_elongata@946788,
        7   Physonect_sp_@2066767,
        8   Stephalia_dilata@2960089,
        9   Frillagalma_vityazi@1155031,
        10  Resomia_ornicephala@3111757,
        11  Lychnagalma_utricularia@2253871,
        12  Nanomia_bijuga@717864,
        13  Cordagalma_sp_@1525873,
        14  Rhizophysa_filiformis@3073669,
        15  Hydra_magnipapillata@52244,
        16  Ectopleura_larynx@3556167
    ;
    TREE * UNTITLED = [&R] (((1[&Ev=S,S=58,ND=0,fake_trait=-0.118140158
END;
```

The output BEAST Nexus file can be imported into R using the `read.beast` function and all the associated data can be used to annotate the tree using ggtree (https://bioconductor.org/packages/ggtree/) (Yu et al. 2017).

```
outfile <- tempfile(fileext = ".tree")
write.beast(tree2, file = outfile)
read.beast(outfile)
```

```
## 'treedata' S4 object that stored information of
##  '/tmp/RtmpzE0nrH/file7e4a75c3b19c.tree'.
##
## ...@ phylo:
```

```
## Phylogenetic tree with 16 tips and 15 internal nodes.
##
## Tip labels:
##  Prayidae_D27SS7@2825365, Kephyes_ovata@2606431, Chuniphyes_multident
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'another_trait',    'Ev',   'fake_trait',   'ND',   'S'.
```

3.3 Exporting Tree Data to *jtree* Format

The treeio (https://bioconductor.org/packages/treeio/) package provides `write.beast` to export `treedata` to BEAST Nexus file. This is quite useful to convert file format, combine tree with data and merging tree data from different sources as we demonstrated in session 3.2 (page 72). The treeio (https://bioconductor.org/packages/treeio/) package also supplies `read.beast` function to parse output file of `write.beast`. Although with treeio (https://bioconductor.org/packages/treeio/), the R community has the ability to manipulate BEAST Nexus format and process tree data, there is still lacking library/package for parsing BEAST file in other programming language.

JSON (https://www.json.org/) (JavaScript Object Notation) is a lightweight data-interchange format and widely supported in almost all modern programming languages. To make it easy to import tree with data in other programming languages, treeio (https://bioconductor.org/packages/treeio/) supports exporting tree with data in jtree format, which is JSON-based and can be easy to parse using any languages that supports JSON.

```
write.jtree(tree2)
```

```
{
    "tree": "(((Prayidae_D27SS7@2825365:0.0682841{1},(Kephyes_o-
```

```
vata@2606431:0.0193941{2},Chuniphyes_multidentata@1277217:0.012-
```

```
1378{3}):0.0217782{20}):0.0607598{19},((Apolemia_sp_@1353964:0.
```

```
11832{4},(((Bargmannia_amoena@263997:0.0144549{5},Bargmannia_el-
```

ongata@946788:0.0149723{6}):0.0925388{25},Physonect_sp_@2066767-

```
:0.077429{7}):0.0274637{24},(Stephalia_dilata@2960089:0.0761163-
```

{8},((Frillagalma_vityazi@1155031:0.0906068{9},Resomia_orniceph-

```
ala@3111757:1{10}e-06):1{28}e-06,((Lychnagalma_utricularia@2253-
```

```
871:0.120851{11},Nanomia_bijuga@717864:0.133939{12}):1{30}e-06,
```

```
Cordagalma_sp_@1525873:0.0693814{13}):1{29}e-06):0.0333823{27})
```

```
:1{26}e-06):0.0431861{23}):1{22}e-06,Rhizophysa_filiformis@3073-
```

```
669:0.22283{14}):0.0292362{21}):0.185603{18},(Hydra_magnipapill-
```

```
ata@52244:0.0621782{15},Ectopleura_larynx@3556167:0.332505{16})
```

```
:0.185603{31}){17};",
    "data":[
  {
    "edge_num": 1,
    "Ev": "S",
    "S": "58",
    "ND": 0,
    "fake_trait": -0.1181,
    "another_trait": 0.3583
  },
  {
    "edge_num": 2,
    "Ev": "S",
    "S": "69",
    "ND": 1,
    "fake_trait": -1.0231,
    "another_trait": 0.1782
  },
  {
    "edge_num": 3,
    "Ev": "S",
    "S": "70",
    "ND": 2,
    "fake_trait": 0.1036,
    "another_trait": 0.9347
  },
  {
    "edge_num": 4,
    "Ev": "S",
    "S": "31",
    "ND": 9,
    "fake_trait": -0.7655,
    "another_trait": 0.8884
  },
  {
    "edge_num": 5,
    "Ev": "S",
    "S": "37",
    "ND": 10,
    "fake_trait": -0.3576,
    "another_trait": 0.5472
  },
```

```
{
    "edge_num": 6,
    "Ev": "S",
    "S": "38",
    "ND": 11,
    "fake_trait": 0.5826,
    "another_trait": 0.0054
  },
  {
    "edge_num": 7,
    "Ev": "S",
    "S": "61",
    "ND": 13,
    "fake_trait": -0.3069,
    "another_trait": 0.6506
  },
  {
    "edge_num": 8,
    "Ev": "S",
    "S": "52",
    "ND": 15,
    "fake_trait": 1.1091,
    "another_trait": 0.5833
  },
  {
    "edge_num": 9,
    "Ev": "S",
    "S": "53",
    "ND": 16,
    "fake_trait": -0.7074,
    "another_trait": 0.8089
  },
  {
    "edge_num": 10,
    "Ev": "S",
    "S": "54",
    "ND": 17,
    "fake_trait": 0.8381,
    "another_trait": 0.0142
  },
  {
    "edge_num": 11,
```

```
  "Ev": "S",
      "S": "65",
      "ND": 19,
      "fake_trait": 1.4588,
      "another_trait": 0.5168
    },
    {
      "edge_num": 12,
      "Ev": "S",
      "S": "71",
      "ND": 20,
      "fake_trait": 2.4347,
      "another_trait": 0.0047
    },
    {
      "edge_num": 13,
      "Ev": "S",
      "S": "64",
      "ND": 22,
      "fake_trait": 1.5658,
      "another_trait": 0.0455
    },
    {
      "edge_num": 14,
      "Ev": "S",
      "S": "26",
      "ND": 28,
      "fake_trait": 0.5891,
      "another_trait": 0.5658
    },
    {
      "edge_num": 15,
      "Ev": "S",
      "S": "16",
      "ND": 5,
      "fake_trait": 0.7044,
      "another_trait": 0.6631
    },
    {
      "edge_num": 16,
      "Ev": "S",
      "S": "15",
```

```
  "ND": 6,
      "fake_trait": -0.6592,
      "another_trait": 0.5052
    },
    {
      "edge_num": 17,
      "Ev": "S",
      "S": "9",
      "ND": 30,
      "fake_trait": -0.4796,
      "another_trait": 0.5199
    },
    {
      "edge_num": 18,
      "Ev": "D",
      "S": "17",
      "ND": 8,
      "fake_trait": 1.1583,
      "another_trait": 0.6469
    },
    {
      "edge_num": 19,
      "Ev": "S",
      "S": "36",
      "ND": 4,
      "fake_trait": -0.3,
      "another_trait": 0.1197
    },
    {
      "edge_num": 20,
      "Ev": "S",
      "S": "60",
      "ND": 3,
      "fake_trait": -0.5958,
      "another_trait": 0.9221
    },
    {
      "edge_num": 21,
      "Ev": "S",
      "S": "17",
      "ND": 29,
      "fake_trait": 0.4763,
```

```
  "another_trait": 0.2626
    },
    {
      "edge_num": 22,
      "Ev": "S",
      "S": "19",
      "ND": 27,
      "fake_trait": 1.6598,
      "another_trait": 0.1251
    },
    {
      "edge_num": 23,
      "Ev": "D",
      "S": "24",
      "ND": 26,
      "fake_trait": -1.4122,
      "another_trait": 0.9355
    },
    {
      "edge_num": 24,
      "Ev": "S",
      "S": "24",
      "ND": 14,
      "fake_trait": 1.0044,
      "another_trait": 0.4201
    },
    {
      "edge_num": 25,
      "Ev": "S",
      "S": "33",
      "ND": 12,
      "fake_trait": 1.6869,
      "another_trait": 0.6343
    },
    {
      "edge_num": 26,
      "Ev": "S",
      "S": "35",
      "ND": 25,
      "fake_trait": -0.6925,
      "another_trait": 0.5852
    },
```

```
{
    "edge_num": 27,
    "Ev": "S",
    "S": "40",
    "ND": 24,
    "fake_trait": 0.4092,
    "another_trait": 0.5382
  },
  {
    "edge_num": 28,
    "Ev": "S",
    "S": "45",
    "ND": 18,
    "fake_trait": -0.5099,
    "another_trait": 0.939
  },
  {
    "edge_num": 29,
    "Ev": "S",
    "S": "46",
    "ND": 23,
    "fake_trait": -1.4843,
    "another_trait": 0.0444
  },
  {
    "edge_num": 30,
    "Ev": "S",
    "S": "56",
    "ND": 21,
    "fake_trait": -1.0766,
    "another_trait": 0.519
  },
  {
    "edge_num": 31,
    "Ev": "S",
    "S": "12",
    "ND": 7,
    "fake_trait": -1.0428,
    "another_trait": 0.6413
  }
],
    "metadata": {"info": "R-package treeio", "data": "Tue Oct  1 09:58:5
}
```

The *jtree* format is based on JSON and can be parsed using JSON parser.

```
jtree_file <- tempfile(fileext = '.jtree')
write.jtree(tree2, file = jtree_file)
jsonlite::fromJSON(jtree_file)
```

```
$tree
[1] "(((Prayidae_D27SS7@2825365:0.0682841{1},(Kephyes_ovata@260-
```

```
6431:0.0193941{2},Chuniphyes_multidentata@1277217:0.0121378{3}):0.021778

$data
   edge_num Ev  S ND fake_trait another_trait
1         1  S 58  0    -0.1181        0.3583
2         2  S 69  1    -1.0231        0.1782
3         3  S 70  2     0.1036        0.9347
4         4  S 31  9    -0.7655        0.8884
5         5  S 37 10    -0.3576        0.5472
6         6  S 38 11     0.5826        0.0054
7         7  S 61 13    -0.3069        0.6506
8         8  S 52 15     1.1091        0.5833
9         9  S 53 16    -0.7074        0.8089
10       10  S 54 17     0.8381        0.0142
11       11  S 65 19     1.4588        0.5168
12       12  S 71 20     2.4347        0.0047
13       13  S 64 22     1.5658        0.0455
14       14  S 26 28     0.5891        0.5658
15       15  S 16  5     0.7044        0.6631
16       16  S 15  6    -0.6592        0.5052
17       17  S  9 30    -0.4796        0.5199
18       18  D 17  8     1.1583        0.6469
19       19  S 36  4    -0.3000        0.1197
20       20  S 60  3    -0.5958        0.9221
21       21  S 17 29     0.4763        0.2626
22       22  S 19 27     1.6598        0.1251
23       23  D 24 26    -1.4122        0.9355
24       24  S 24 14     1.0044        0.4201
25       25  S 33 12     1.6869        0.6343
26       26  S 35 25    -0.6925        0.5852
27       27  S 40 24     0.4092        0.5382
28       28  S 45 18    -0.5099        0.9390
29       29  S 46 23    -1.4843        0.0444
30       30  S 56 21    -1.0766        0.5190
31       31  S 12  7    -1.0428        0.6413

$metadata
$metadata$info
[1] "R-package treeio"

$metadata$data
[1] "Tue Oct  1 09:58:57 2019"
```

The *jtree* file can be directly imported as a `treedata` object using `read.jtree` provided also in treeio (https://bioconductor.org/packages/treeio/) package (see also session 1.3 (page 37)).

```
read.jtree(jtree_file)
```

```
## 'treedata' S4 object that stored information of
##  '/tmp/RtmpzE0nrH/file7e4a27bb6ba4.jtree'.
##
## ...@ phylo:
## Phylogenetic tree with 16 tips and 15 internal nodes.
##
## Tip labels:
##  Prayidae_D27SS7@2825365, Kephyes_ovata@2606431, Chuniphyes_multident
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'Ev',   'S',    'ND',   'fake_trait',   'another_trait'.
```

3.4 Summary

Phylogenetic tree associated data is often stored in a separate file and need expertise to map the data to the tree structure. Lacking standardization to store and represent phylogeny and associated data makes it difficult for researchers to access and integrate the phylogenetic data into their studies. The treeio (http://bioconductor.org/packages/treeio) package provides functions to import phylogeny with associated data from a number of sources, including analysis finding from commonly used software and external data such as experimental, clinical or meta data. These tree + data can be exported into a single file as a BEAST or jtree formats, and the output file can be parsed back to R by treeio (http://bioconductor.org/packages/treeio) and the data is easy to access. The input

and output utilities supplied by treeio (http://bioconductor.org/packages/treeio) package lay the foundation for phylogenetic data integration for downstream comparative study and visualization.

Part II: Tree data visualization and annotation

4

Phylogenetic Tree Visualization

4.1 Introduction

There are many software packages and webtools that are designed for displaying phylogenetic trees, such as *TreeView* (Page 2002 (page 279)), *FigTree*[11], *TreeDyn* (Chevenet et al. 2006), *Dendroscope* (Huson and Scornavacca 2012 (page 277)), *EvolView* (He et al. 2016 (page 277)) and *iTOL* (Letunic and Bork 2007 (page 278)), *etc.*. Only several of them, such as *FigTree*, *TreeDyn* and *iTOL*, allow users to annotate the trees with coloring branches, highlighted clades and tree features. However, their pre-defined annotating functions are usually limited to some specific phylogenetic data. As phylogenetic trees are becoming more widely used in multidisciplinary studies, there is an increasing need to incorporate various types of the phylogenetic covariates and other associated data from different sources into the trees for visualizations and further analyses. For instance, influenza virus has a wide host range, diverse and dynamic genotypes and characteristic transmission behaviors that are mostly associated with the virus evolution and essentially among themselves. Therefore, in addition to standalone applications that focus on each of the specific analysis and data type, researchers studying molecular evolution need a robust and programmable platform that allows the high levels of integration and visualization of many of these different aspects of data (raw or from other primary analyses) over the phylogenetic trees to identify their associations and patterns.

To fill this gap, we developed ggtree (http://bioconductor.org/packages/ggtree), a package for the R programming language (R Core Team 2016 (page 280)) released under the Bioconductor

project (Gentleman et al. 2004 (page 276)). The ggtree (http://bioconductor.org/packages/ggtree) is built to work with phylogenetic data object (see chapter 1 (page 7) and chapter 9 (page 199)), and display tree graphics with ggplot2 (https://CRAN.R-project.org/package=ggplot2) package (Wickham 2016 (page 281)) that was based on the grammar of graphics (Wilkinson et al. 2005 (page 281)).

The R language is increasingly being used in phylogenetics. However, a comprehensive package, designed for viewing and annotating phylogenetic trees, particularly with complex data integration, is not yet available. Most of the R packages in phylogenetics focus on specific statistical analyses rather than viewing and annotating the trees with more generalized phylogeny-associated data. Some packages, including ape (https://CRAN.R-project.org/package=ape) (Paradis, Claude, and Strimmer 2004 (page 280)) and phytools (https://CRAN.R-project.org/package=phytools) (Revell 2012 (page 280)), which are capable of displaying and annotating trees, are developed using the base graphics system of R. In particular, ape (https://CRAN.R-project.org/package=ape) is one of the fundamental package for phylogenetic analysis and data processing. However, the base graphics system is relatively difficult to extend and limits the complexity of tree figure to be displayed. OutbreakTools (https://CRAN.R-project.org/package=OutbreakTools) (Jombart et al. 2014 (page 277)) and phyloseq (http://bioconductor.org/packages/phyloseq) (McMurdie and Holmes 2013 (page 279)) extended ggplot2 (https://CRAN.R-project.org/package=ggplot2) to plot phylogenetic trees. The ggplot2 (https://CRAN.R-project.org/package=ggplot2) system of graphics allows rapid customization and exploration of design solutions. However these packages were designed for epidemiology and microbiome data respectively and did not aim to provide a general solution for tree visualization and annotation. The ggtree (http://bioconductor.org/packages/ggtree) package also inherits versatile properties of ggplot2 (https://CRAN.R-project.org/package=ggplot2), and more importantly allows constructing complex tree figures by freely combining multiple layers of annotations using the tree associated data imported from different sources (see detailed in Chapter 1 (page 7)).

4.2 Visualizing Phylogenetic Tree with *ggtree*

The ggtree (http://bioconductor.org/packages/ggtree) package is designed for annotating phylogenetic trees with their associated data of different types and from various sources. These data could come from users or analysis programs, and might include evolutionary rates, ancestral sequences, *etc.* that are associated with the taxa from real samples, or with the internal nodes representing hypothetic ancestor strain/species, or with the tree branches indicating evolutionary time courses. For instance, the data could be the geographic positions of the sampled avian influenza viruses (informed by the survey locations) and the ancestral nodes (by phylogeographic inference) in the viral gene tree (Lam et al. 2012 (page 278)).

The ggtree (http://bioconductor.org/packages/ggtree) supports ggplot2 (https://CRAN.R-project.org/package=ggplot2)'s graphical language, which allows high level of customization, is intuitive and flexible. It is notable that ggplot2 (https://CRAN.R-project.org/package=ggplot2) itself does not provide low-level geometric objects or other support for tree-like structures, and hence ggtree (http://bioconductor.org/packages/ggtree) is a useful extension on that regard. Even though the other two phylogenetics-related R packages, OutbreakTools (https://CRAN.R-project.org/package=OutbreakTools) and phyloseq (http://bioconductor.org/packages/phyloseq), are developed based on ggplot2 (https://CRAN.R-project.org/package=ggplot2), the most valuable part of ggplot2 (https://CRAN.R-project.org/package=ggplot2) syntax - adding layers of annotations - is not supported in these packages. For example, if we have plotted a tree without taxa labels, OutbreakTools (https://CRAN.R-project.org/package=OutbreakTools) and phyloseq (http://bioconductor.org/packages/phyloseq) provide no easy way for general `R` users, who have little knowledge about the infrastructures of these packages, to add a layer of taxa labels. The ggtree (http://bioconductor.org/packages/ggtree) extends ggplot2 (https://CRAN.R-project.org/package=ggplot2) to support tree objects by implementing a geometric layer, `geom_tree`, to support visualizing tree structure. In ggtree (http://bioconductor.org/packages/ggtree), viewing a phylogenetic tree is relatively easy, via the command `ggplot(tree_object) + geom_tree() +`

`theme_tree()` or `ggtree(tree_object)` for short. Layers of annotations can be added one by one via the + operator. To facilitate tree visualization, ggtree (http://bioconductor.org/packages/ggtree) provides several geometric layers, including `geom_treescale` for adding legend of tree branch scale (genetic distance, divergence time, *etc.*), `geom_range` for displaying uncertainty of branch lengths (confidence interval or range, *etc.*), `geom_tiplab` for adding taxa label, `geom_tippoint` and `geom_nodepoint` for adding symbols of tips and internal nodes, `geom_hilight` for highlighting a clade with rectangle, and `geom_cladelabel` for annotating a selected clade with a bar and text label, *etc.*. A full list of geometric layers provided by *ggtree* are summarized in Table 5.1.

To view a phylogenetic tree, we first need to parse the tree file into `R`. Treeio (page 7) package parses diverse annotation data from different software outputs into `S4` phylogenetic data objects. The ggtree (http://bioconductor.org/packages/ggtree) mainly utilizes these `S4` objects to display and annotate the tree. There are also other R packages that defined `S3`/`S4` classes to store phylogenetic trees with domain specific associated data, including `phylo4` and `phylo4d` defined in phylobase (https://CRAN.R-project.org/package=phylobase) package, `obkdata` defined in OutbreakTools (https://CRAN.R-project.org/package=OutbreakTools) package, and `phyloseq` defined in phyloseq (http://bioconductor.org/packages/phyloseq) package (see also Chapter 9 (page 199)). All these tree objects are also supported in ggtree (http://bioconductor.org/packages/ggtree) and their specific annotation data can be used to annotate the tree in ggtree (http://bioconductor.org/packages/ggtree). Such compatibility of ggtree (http://bioconductor.org/packages/ggtree) facilitates the integration of data and analysis results. In addition, ggtree (http://bioconductor.org/packages/ggtree) also supports other tree-like structure, including dendrogram (page 208).

4.2.1 Basic Tree Visualization

The ggtree (http://bioconductor.org/packages/ggtree) package extends ggplot2 (https://CRAN.R-project.org/package=ggplot2) (Wickham 2009 (page 281)) package to support viewing phylogenetic tree. It implements `geom_tree` layer for displaying phylogenetic tree, as shown below in Figure 4.1A.

```
library("treeio")
library("ggtree")

nwk <- system.file("extdata", "sample.nwk",
         package="treeio")
tree <- read.tree(nwk)

ggplot(tree, aes(x, y)) + geom_tree() + theme_tree()
```

The function, `ggtree`, was implemented as a short cut to visualize a tree, and it works exactly the same as shown above.

ggtree (http://bioconductor.org/packages/ggtree) takes all the advantages of ggplot2 (https://CRAN.R-project.org/package=ggplot2). For example, we can change the color, size and type of the lines as we do with ggplot2 (https://CRAN.R-project.org/package=ggplot2) (Figure 4.1B).

```
ggtree(tree, color="firebrick", size=1, linetype="dotted")
```

By default, the tree is viewed in ladderize form, user can set the parameter `ladderize = FALSE` to disable it (Figure 4.1C).

```
ggtree(tree, ladderize=FALSE)
```

The `branch.length` is used to scale the edge, user can set the parameter `branch.length = "none"` to only view the tree topology (cladogram, Figure 4.1D) or other numerical variable to scale the tree (*e.g.* d_N/d_S, see also in Chapter 5 (page 139)).

```
ggtree(tree, branch.length="none")
```

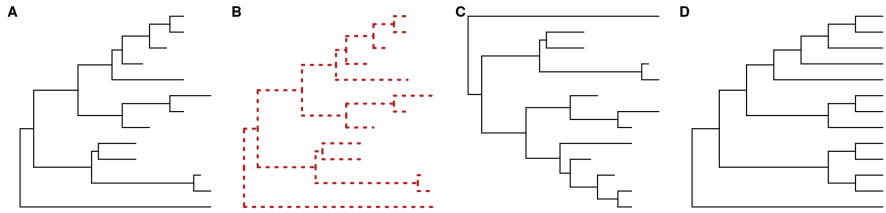


Figure 4.1: **Basic tree visualization.**

4.2.2 Layouts of phylogenetic tree

Viewing phylogenetic with ggtree is quite simple, just pass the tree object to *ggtree* function. We have developed several types of layouts for tree presentation (Figure 4.2), including rectangular (by default), slanted, circular, fan, unrooted (equal angle and daylight methods), time-scaled and 2-dimensional layouts.

Here are examples of visualizing a tree with different layouts:

```
library(ggtree)
set.seed(2017-02-16)
tree <- rtree(50)
ggtree(tree)
ggtree(tree, layout="slanted")
ggtree(tree, layout="circular")
ggtree(tree, layout="fan", open.angle=120)
ggtree(tree, layout="equal_angle")
ggtree(tree, layout="daylight")
ggtree(tree, branch.length='none')
ggtree(tree, branch.length='none', layout='circular')
ggtree(tree, layout="daylight", branch.length = 'none')
```

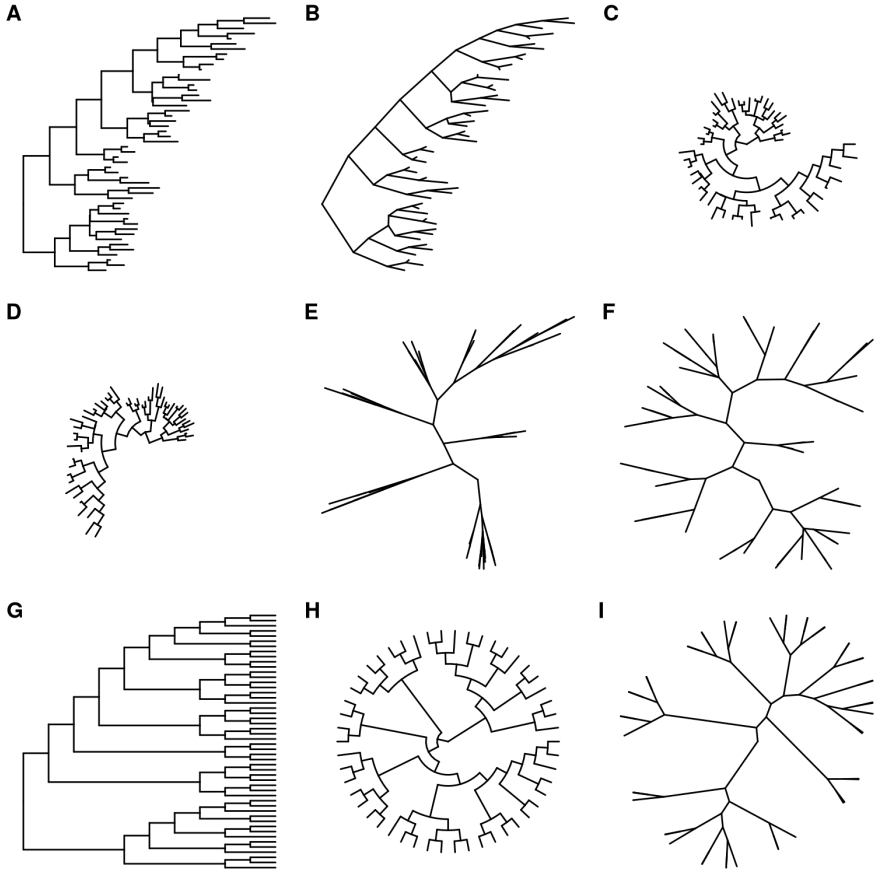


Figure 4.2: **Tree layouts.** Phylogram: rectangular layout (A), slanted layout (B), circular layout (C) and fan layout (D). Unrooted: equal-angle method (E) and daylight method (F). Cladogram: rectangular layout (G), circular layout (H) and unrooted layout (I). Slanted and fan layouts for cladogram are also supported.

There are also other possible layouts that can be drawn by modifying scales/coordination (Figure 4.3).

```
ggtree(tree) + scale_x_reverse()
ggtree(tree) + coord_flip()
ggtree(tree) + layout_dendrogram()
```

```
print(ggtree(tree), newpage=TRUE,
        vp=grid::viewport(angle=-30, width=.9, height=.9))
ggtree(tree, layout='slanted') + coord_flip()
ggtree(tree, layout='slanted', branch.length='none') +
        layout_dendrogram()
ggtree(tree, layout='circular') + xlim(-10, NA)
ggtree(tree) + scale_x_reverse() + coord_polar(theta='y')
ggtree(tree) + scale_x_reverse(limits=c(10, 0)) +
        coord_polar(theta='y')
```

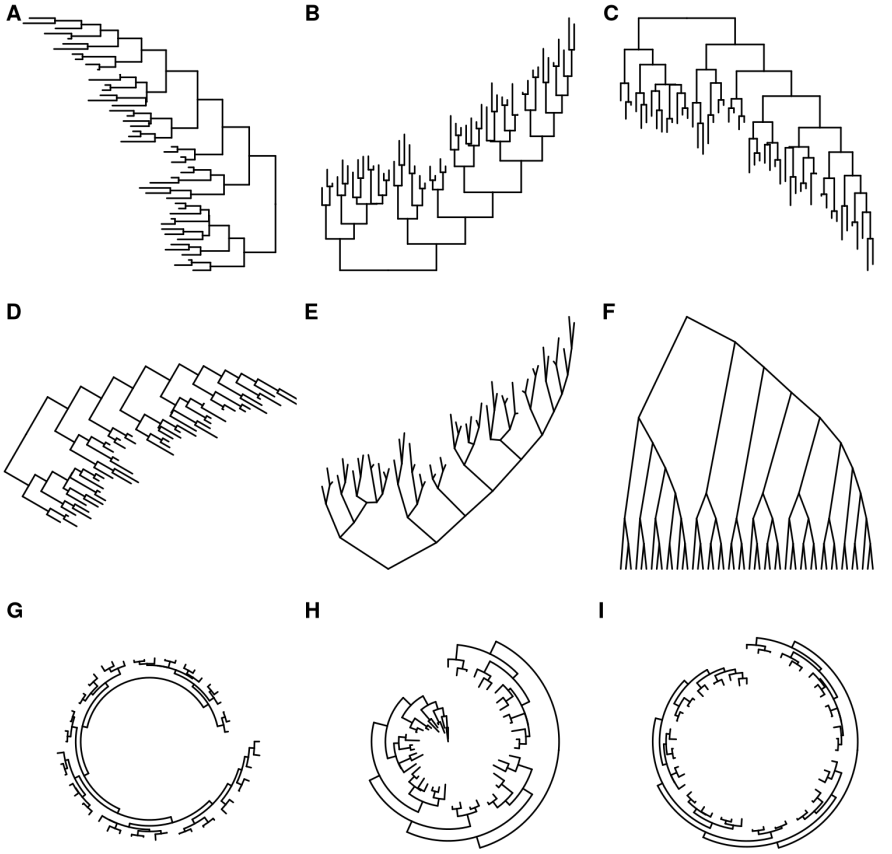


Figure 4.3: **Derived Tree layouts.** right-to-left rectangular layout (A), bottom-up rectangular layout (B), top-down rectangular layout (Dendrogram) (C), rotated rectangular layout (D), bottom-up slanted layout (E), top-down slanted layout (Cladogram) (F), circular layout (G), circular inward layout (H and I).

Phylogram. Layouts of *rectangular*, *slanted*, *circular* and *fan* are supported to visualize phylogram (by default, with branch length scaled) as demonstrated in Figure 4.2A, B, C and D.

Unrooted layout. Unrooted (also called 'radial') layout is supported by equal-angle and daylight algorithms, user can specify unrooted layout algorithm by passing "equal_angle" or "daylight" to `layout` parameter to

visualize the tree. Equal-angle method was proposed by Christopher Meacham in *PLOTREE*, which was incorporated in *PHYLIP* (Retief 2000 (page 280)). This method starts from the root of the tree and allocates arcs of angle to each subtrees proportional to the number of tips in it. It iterates from root to tips and subdivides the angle allocated to a subtree into angles for its dependent subtrees. This method is fast and was implemented in many software packages. As shown in Figure 4.2E, equal angle method has a drawback that tips are tend to be clustered together and leaving many spaces unused. The daylight method starts from an initial tree built by equal angle and iteratively improves it by successively going to each interior node and swinging subtrees so that the arcs of "daylight" are equal (Figure 4.2F). This method was firstly implemented in *PAUP** (Wilgenbusch and Swofford 2003 (page 281)).

Cladogram. To visualize cladogram that without branch length scaling and only display the tree structure, `branch.length` is set to "none" and it works for all types of layouts (Figure 4.2G, H and I).

Time-scaled layout. For time-scaled tree, the most recent sampling date must be specified via the `mrsd` parameter and *ggtree* will scaled the tree by sampling (tip) and divergence (internal node) time, and a time scale axis will be displayed under the tree by default.

```
beast_file <- system.file("examples/MCC_FluA_H3.tree",
                          package="ggtree")
beast_tree <- read.beast(beast_file)
ggtree(beast_tree, mrsd="2013-01-01") + theme_tree2()
```

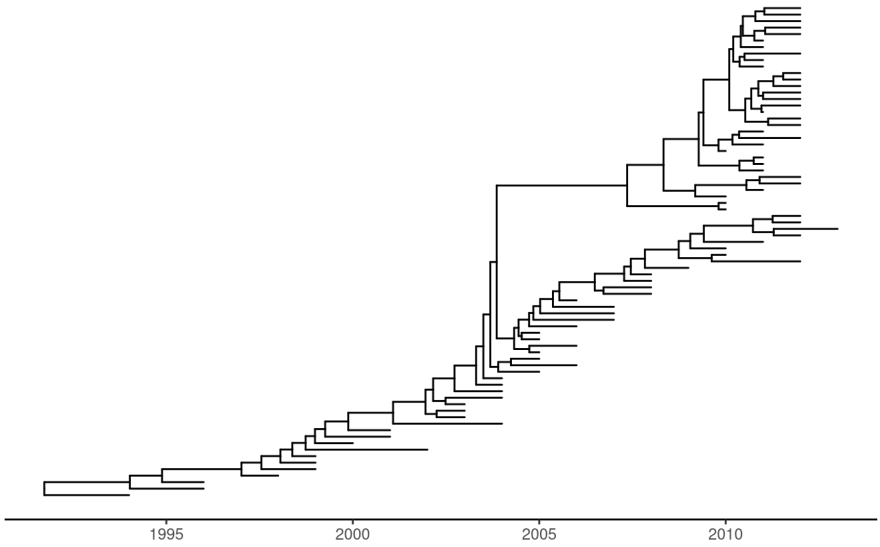


Figure 4.4: **Time-scaled layout.** The *x-axis* is the timescale (in units of year). The divergence time was inferred by *BEAST* using molecular clock model.

Two-dimensional tree layout. A two-dimensional tree is a projection of the phylogenetic tree in a space defined by the associated phenotype (numerical or categorical trait, on the *y*-axis) and tree branch scale (*e.g.*, evolutionary distance, divergent time, on the *x*-axis). The phenotype can be a measure of certain biological characteristics of the taxa and hypothetical ancestors in the tree. This is a new layout we proposed in *ggtree*, which is useful to track the virus phenotypes or other behaviors (*y*-axis) changing with the virus evolution (*x*-axis). In fact, the analysis of phenotypes or genotypes over evolutionary time have been widely used for study influenza virus evolution (Neher et al. 2016 (page 279)), though such analysis diagrams are not tree-like, *i.e.*, no connection between data points, unlike our two-dimensional tree layout that connect data points with the corresponding tree branches. Therefore, this new layout we provided will make such data analysis easier and more scalable for large sequence data sets of influenza viruses.

In this example, we used the previous time-scaled tree of H3 human and swine influenza viruses (Figure 4.4; data published in (Liang et al. 2014

)) and scaled the *y*-axis based on the predicted *N*-linked glycosylation sites (NLG) for each of the taxon and ancestral sequences of hemagglutinin proteins. The NLG sites were predicted using *NetNGlyc* 1.0 Server[12]. To scaled the *y*-axis, the parameter `yscale` in the `ggtree()` function is set to a numerical or categorical variable. If `yscale` is a categorical variable as in this example, users should specify how the categories are to be mapped to numerical values via the `yscale_mapping` variables.

```
NAG_file <- system.file("examples/NAG_inHA1.txt",
         package="ggtree")

NAG.df <- read.table(NAG_file, sep="\t", header=FALSE,
                     stringsAsFactors = FALSE)
NAG <- NAG.df[,2]
names(NAG) <- NAG.df[,1]

## separate the tree by host species
tip <- get.tree(beast_tree)$tip.label
beast_tree <- groupOTU(beast_tree, tip[grep("Swine", tip)],
                       group_name = "host")

p <- ggtree(beast_tree, aes(color=host), mrsd="2013-01-01",
            yscale = "label", yscale_mapping = NAG) +
  theme_classic() + theme(legend.position='none') +
  scale_color_manual(values=c("blue", "red"),
                     labels=c("human", "swine")) +
  ylab("Number of predicted N-linked glycoslyation sites")

## (optional) add more annotations to help interpretation
p + geom_nodepoint(color="grey", size=3, alpha=.8) +
  geom_rootpoint(color="black", size=3) +
  geom_tippoint(size=3, alpha=.5) +
  annotate("point", 1992, 5.6, size=3, color="black") +
  annotate("point", 1992, 5.4, size=3, color="grey") +
  annotate("point", 1991.6, 5.2, size=3, color="blue") +
  annotate("point", 1992, 5.2, size=3, color="red") +
  annotate("text", 1992.3, 5.6, hjust=0, size=4, label="Root
         node") +
  annotate("text", 1992.3, 5.4, hjust=0, size=4,
           label="Internal nodes") +
```

```
    annotate("text", 1992.3, 5.2, hjust=0, size=4,
             label="Tip nodes (blue: human; red: swine)")
```

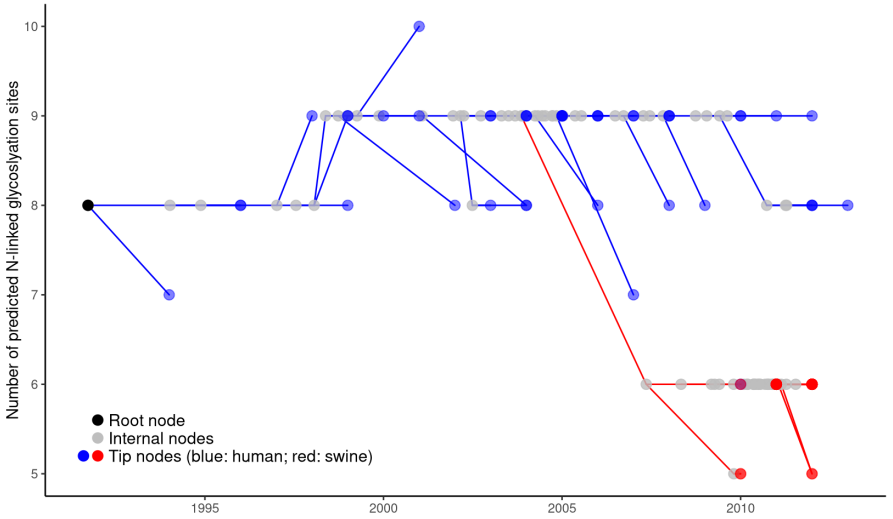


Figure 4.5: **Two-dimensional tree layout.** The trunk and other branches highlighted in red (for swine) and blue (for human). The *x*-axis is scaled to the branch length (in units of year) of the time-scaled tree. The *y*-axis is scaled to the node attribute variable, in this case the number of predicted *N*-linked glycosylation site (NLG) on the hemagglutinin protein. Colored circles indicate the different types of tree nodes. Note that nodes assigned the same *x*- (temporal) and *y*- (NLG) coordinates are superimposed in this representation and appear as one node, which is shaded based on the colors of all the nodes at that point.

As shown in Figure 4.5, two-dimensional tree good at visualizing the change of phenotype over the evolution in the phylogenetic tree. In this example, it is shown that H3 gene of human influenza A virus maintained high level of *N*-linked glycosylation sites (n=8 to 9) over last two decades and dropped significantly to 5 or 6 in a separate viral lineage transmitted to swine populations and established there. It was indeed hypothesized that the human influenza virus with high level of glycosylation on the viral hemagglutinin protein provides better shielding effect to protect the antigenic sites from exposure to the herd immunity, and thus has

selective advantage in human populations that maintain high level of herd immunity against the circulating human influenza virus strains. For the viral lineage that newly jumped across the species barrier and transmitted to swine population, the shielding effect of the high-level surface glycan oppositely impose selective disadvantage because the receptor-binding domain may also be shielded which greatly affect the viral fitness of the lineage that newly adapted to a new host species. Another example of two-dimensional tree can be found on Figure 4.12.

4.3 Displaying Tree Components

4.3.1 Displaying tree scale (evolution distance)

To show tree scale, user can use `geom_treescale()` layer (Figure 4.6A).

```
ggtree(tree) + geom_treescale()
```

`geom_treescale()` supports the following parameters:

- *x* and *y* for tree scale position
- *width* for the length of the tree scale
- *fontsize* for the size of the text
- *linesize* for the size of the line
- *offset* for relative position of the line and the text
- *color* for color of the tree scale

```
ggtree(tree) + geom_treescale(x=0, y=45, width=1,
        color='red')
ggtree(tree) + geom_treescale(fontsize=6, linesize=2,
        offset=1)
```

We can also use `theme_tree2()` to display the tree scale by adding *x axis*.

```
ggtree(tree) + theme_tree2()
```

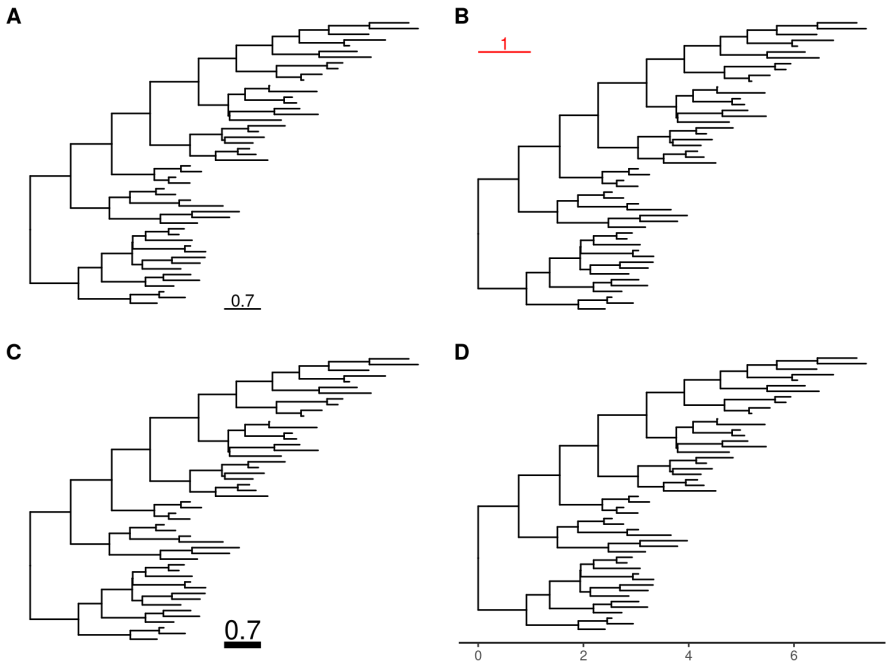


Figure 4.6: **Display tree scale.** `geom_treescale` automatically add a scale bar for evolutionary distance (A). Users can modify color, width and position of the scale (B) as well as size of the scale bar and text and their relative position (C). Another possible solution is to enable x-axis which is useful for time-scale tree (D).

Tree scale is not restricted to evolution distance, treeio (http://bioconductor.org/packages/treeio) can re-scale the tree with other numerical variable (details described in session 2.3 (page 58)).

4.3.2 Displaying nodes/tips

Showing all the internal nodes and tips in the tree can be done by adding a layer of points using `geom_nodepoint`, `geom_tippoint` or `geom_point`.

```
ggtree(tree) + geom_point(aes(shape=isTip, color=isTip),
        size=3)
```

```
p <- ggtree(tree) + geom_nodepoint(color="#b5e521",
        alpha=1/4, size=10)
p + geom_tippoint(color="#FDAC4F", shape=8, size=3)
```

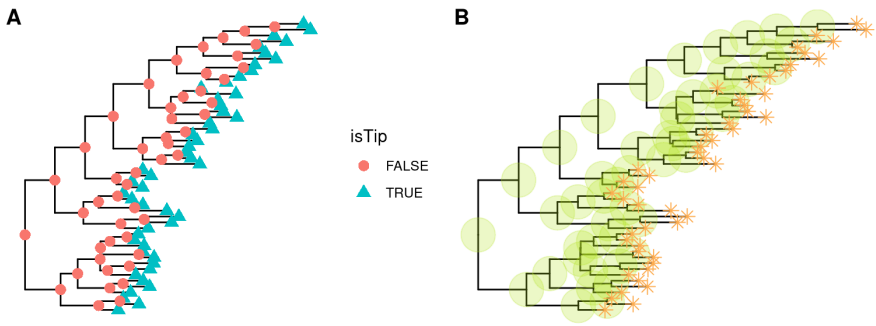


Figure 4.7: **Display external and internal nodes.** `geom_point` automatically add symbolic points of all nodes (A). `geom_nodepoint` adds symbolic points for internal nodes and `geom_tippoint` adds symbolic points for external nodes (B).

4.3.3 Displaying labels

Users can use `geom_text` or `geom_label` to display the node (if available) and tip labels simultaneously or `geom_tiplab` to only display tip labels (Figure 4.8A).

```
p + geom_tiplab(size=3, color="purple")
```

`geom_tiplab` not only supports using *text* or *label* geom to display labels, it also supports *image* geom to label tip with image files (see Chapter 7 (page 171)). A corresponding geom, `geom_nodelab` is also provided for displaying node labels.

For *circular* and *unrooted* layout, ggtree (http://bioconductor.org/packages/ggtree) supports rotating node labels according to the angles of the branches (Figure 4.8B).

```
ggtree(tree, layout="circular") +
        geom_tiplab(aes(angle=angle), color='blue')
```

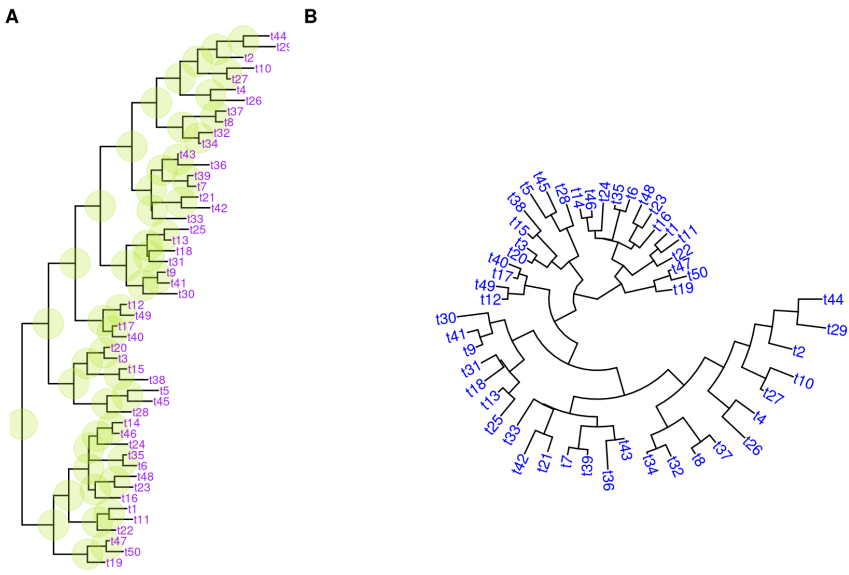


Figure 4.8: **Display external and internal nodes.** `geom_point` automatically add symbolic points of all nodes (A). `geom_nodepoint` adds symbolic points for internal nodes and `geom_tippoint` adds symbolic points for external nodes (B).

By default, the positions to display text are based on the node positions, we can change them to based on the middle of the branch/edge (by setting `aes(x = branch)`), which is very useful when annotating transition from parent node to child node.

4.3.4 Displaying root edge

ggtree (http://bioconductor.org/packages/ggtree) doesn't plot root edge by default. Users can use `geom_rootedge()` to automatically display root edge (Figure 4.9A). If there is no root edge information, `geom_rootedge()` will display nothing by default (Figure 4.9B). Users can set the root edge to the tree (Figure 4.9C) or specify `rootedge` in `geom_rootedge()` (Figure 4.9D).

```
## with root edge = 1
tree1 <- read.tree(text='((A:1,B:2):3,C:2):1;')
ggtree(tree1) + geom_tiplab() + geom_rootedge()

## without root edge
tree2 <- read.tree(text='((A:1,B:2):3,C:2);')
ggtree(tree2) + geom_tiplab() + geom_rootedge()

## setting root edge
tree2$root.edge <- 2
ggtree(tree2) + geom_tiplab() + geom_rootedge()

## specify length of root edge for just plotting
## this will ignore tree$root.edge
ggtree(tree2) + geom_tiplab() + geom_rootedge(rootedge = 3)
```

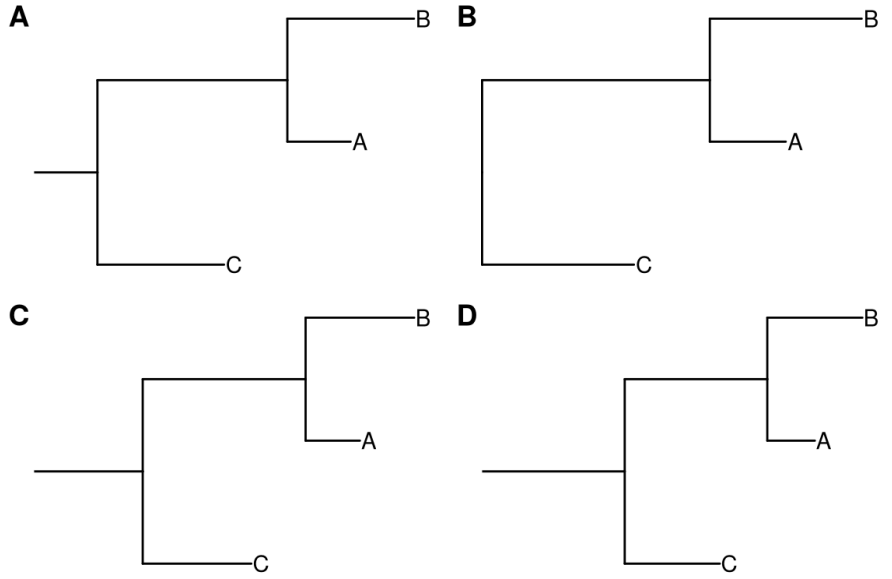


Figure 4.9: **Display root edge.** `geom_rootedge` supports displaying root edge if the root edge was presented (A). It shows nothing if there is no root edge (B). In this case, users can manually setting root edge for the tree (C) or just specify the length of root for plotting (D).

4.3.5 Color tree

In `ggtree`, coloring phylogenetic tree is easy, by using `aes(color=VAR)` to map the color of tree based on a specific variable (numeric and category are both supported).

```
ggtree(beast_tree, aes(color=rate)) +
    scale_color_continuous(low='darkgreen', high='red') +
    theme(legend.position="right")
```

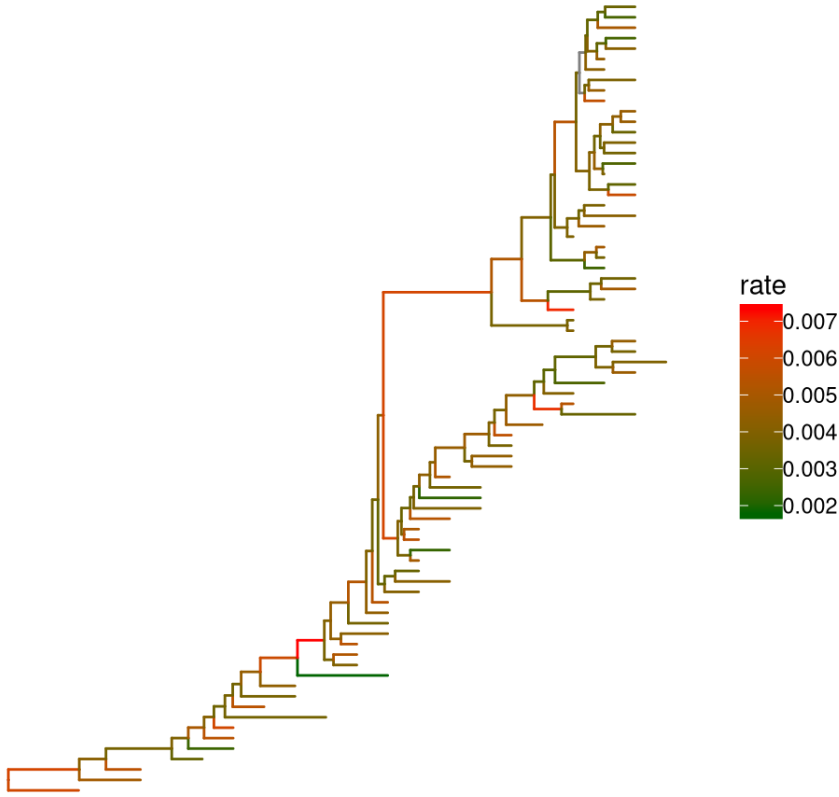


Figure 4.10: **Color tree by continuous or discrete feature.** Edges are colored by values that associated with child node.

User can use any feature (if available), including clade posterior and d_N/d_S *etc.*, to scale the color of the tree. If the feature is continuous numerical value, ggtree (http://bioconductor.org/packages/ggtree) provides a `continuous` parameter to support plotting continuous state transition in edges. Here, we use the example provided in `http://www.phytools.org/eqg2015/asr.html` to demonstrate this functionality.

```
anole.tree<-read.tree("http://www.phytools.org/eqg2015/data/anole.tre"
```

```
        )
svl <-
        read.csv("http://www.phytools.org/eqg2015/data/svl.csv",
    row.names=1)
svl <- as.matrix(svl)[,1]
fit <- phytools::fastAnc(anole.tree,svl,vars=TRUE,CI=TRUE)

td <- data.frame(node = nodeid(anole.tree, names(svl)),
                 trait = svl)
nd <- data.frame(node = names(fit$ace), trait = fit$ace)

d <- rbind(td, nd)
d$node <- as.numeric(d$node)
tree <- full_join(anole.tree, d, by = 'node')

ggtree(tree, aes(color=trait), layout = 'circular',
       ladderize = FALSE, continuous = TRUE, size=2) +
    scale_color_gradientn(colours=c("red", 'orange',
        'green', 'cyan', 'blue')) +
    geom_tiplab(hjust = -.1) + xlim(0, 1.2) +
        theme(legend.position = c(.05, .85))
```

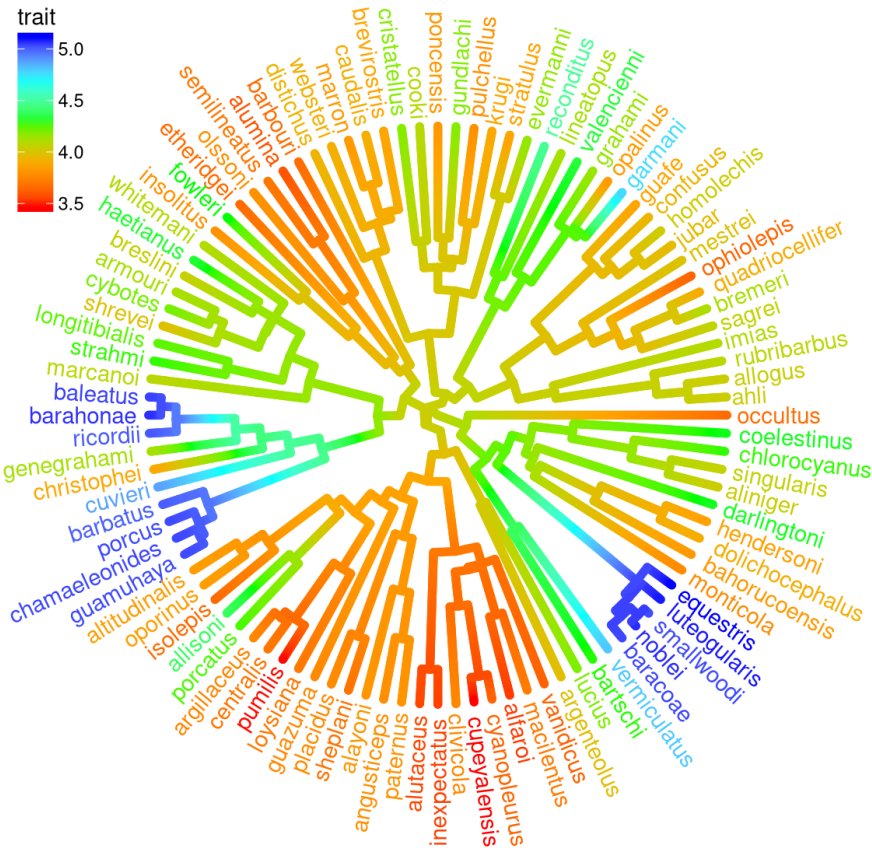


Figure 4.11: **Continuous state transition in edges.** Edges are colored by the values from ancestral trait to offspring.

Besides, we can use two-dimensional tree (as demonstrated in Figure 4.5) to visualize phenotype on the vertical dimension to create the phenogram. We can use ggrepel (https://CRAN.R-project.org/package=ggrepel) package to repel tip labels to avoid overlapping as demonstrated in Figure A.4.

```
ggtree(tree, aes(color=trait), continuous = TRUE, yscale =
        "trait") +
    scale_color_viridis_c() + theme_minimal()
```

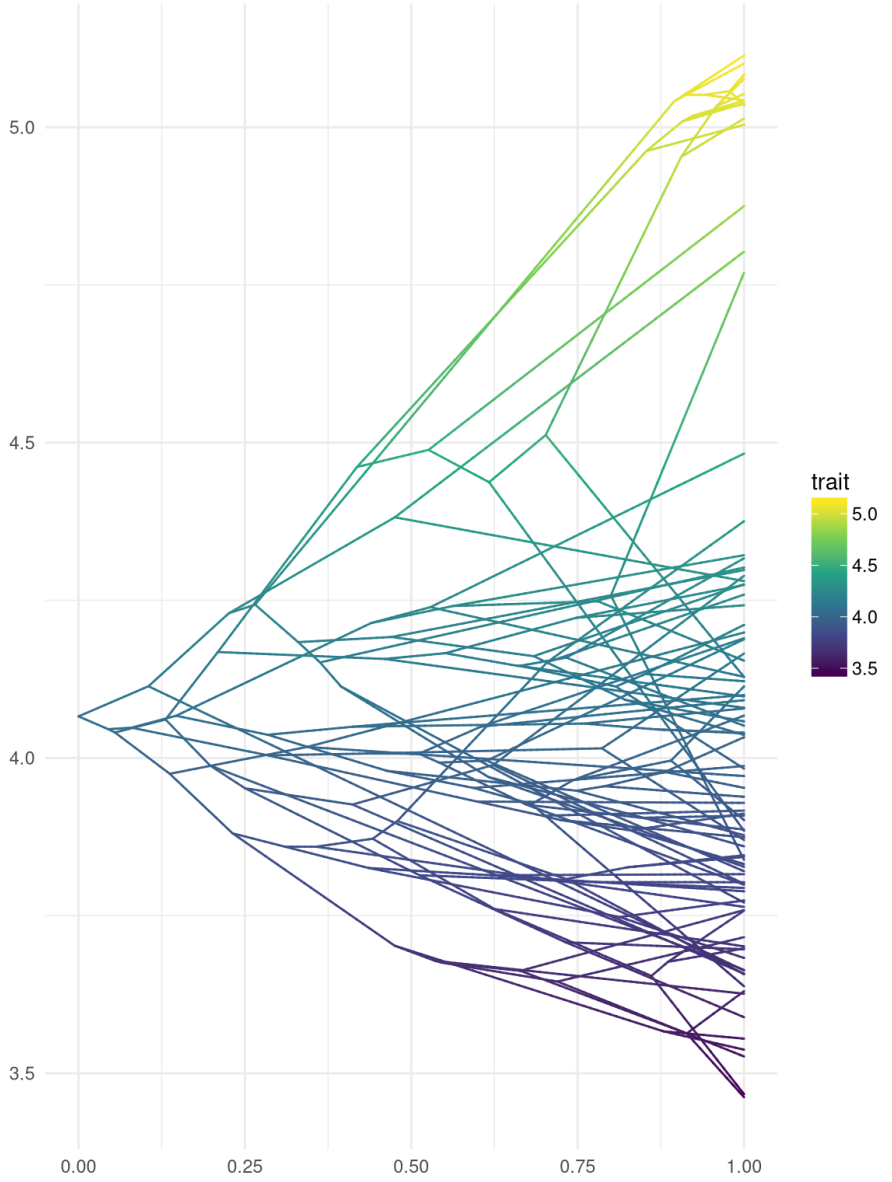


Figure 4.12: **Phenogram.** Projecting the tree into a space defined by time (or genetic distance) on the horizontal axis and phenotype on the vertical dimension.

4.3.6 Rescale tree

Most of the phylogenetic trees are scaled by evolutionary distance (substitution/site). In ggtree, users can re-scale a phylogenetic tree by any numerical variable inferred by evolutionary analysis (*e.g.* d_N/d_S).

This example displays a time tree (Figure 4.13A) and the branches were rescaled by substitution rate inferred by BEAST (Figure 4.13B).

```
library("treeio")
beast_file <- system.file("examples/MCC_FluA_H3.tree",
         package="ggtree")
beast_tree <- read.beast(beast_file)
beast_tree
```

```
## 'treedata' S4 object that stored information of
##  '/home/ygc/R/library/ggtree/examples/MCC_FluA_H3.tree'.
##
## ...@ phylo:
## Phylogenetic tree with 76 tips and 75 internal nodes.
##
## Tip labels:
##  A/Hokkaido/30-1-a/2013, A/New_York/334/2004, A/New_York/463/2005, A/
##
## Rooted; includes branch lengths.
##
## with the following features available:
##  'height',   'height_0.95_HPD',  'height_median',
##  'height_range', 'length',   'length_0.95_HPD',
##  'length_median',    'length_range', 'posterior',    'rate',
##  'rate_0.95_HPD',    'rate_median',  'rate_range'.
```

```
p1 <- ggtree(beast_tree, mrsd='2013-01-01') + theme_tree2()
        +
    labs(caption="Divergence time")
```

```
p2 <- ggtree(beast_tree, branch.length='rate') +
        theme_tree2() +
    labs(caption="Substitution rate")
```

The following example draw a tree inferred by CodeML (Figure 4.13C), and the branches can be rescaled by using d_N/d_S values (Figure 4.13D).

```
mlcfile <- system.file("extdata/PAML_Codeml", "mlc",
        package="treeio")
mlc_tree <- read.codeml_mlc(mlcfile)
p3 <- ggtree(mlc_tree) + theme_tree2() +
    labs(caption="nucleotide substitutions per codon")
p4 <- ggtree(mlc_tree, branch.length='dN_vs_dS') +
        theme_tree2() +
    labs(caption="dN/dS tree")
```

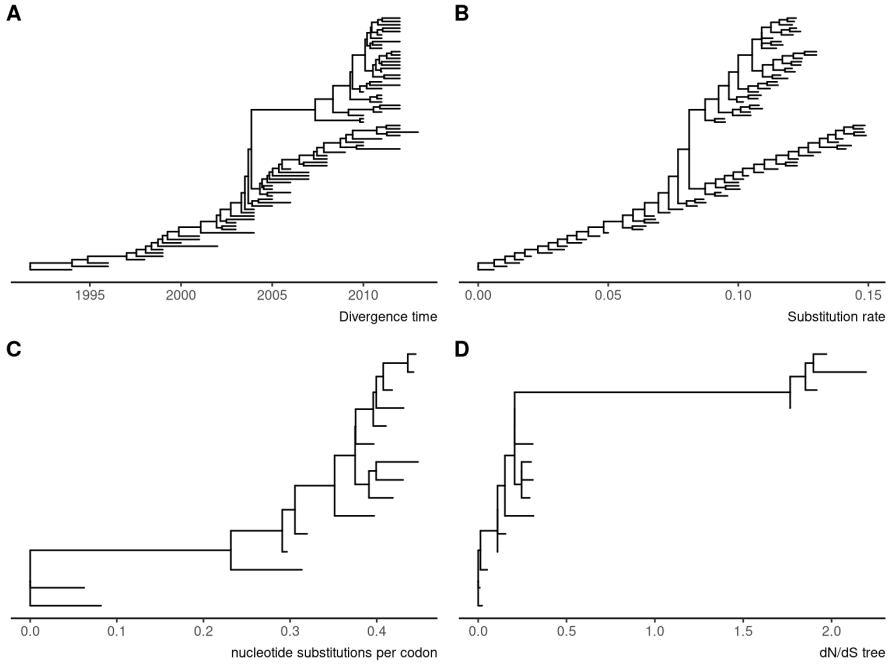


Figure 4.13: **Rescale tree branches.** A time scaled tree inferred by BEAST (A) and its branches were rescaled by substitution rate (B). A tree inferred by CodeML (C) and the branches were rescaled by d_N/d_S values (D).

In addition to specify `branch.length` in tree visualization, users can change branch length stored in tree object by using `rescale_tree` function, and the following command will display a tree that is identical to (Figure 4.13B).

```
beast_tree2 <- rescale_tree(beast_tree,
        branch.length='rate')
ggtree(beast_tree2) + theme_tree2()
```

4.3.7 Modify compenents of a theme

`theme_tree()` defined a totally blank canvas, while *`theme_tree2()`* adds phylogenetic distance (via x-axis). These two themes all accept a parameter of `bgcolor` that defined the background color. Users can use

any theme components (http://ggplot2.tidyverse.org/reference/theme.html) to the `theme_tree()` or `theme_tree2()` functions to modify them.

```
set.seed(2019)
x <- rtree(30)
ggtree(x, color="red") + theme_tree("steelblue")
ggtree(x, color="white") + theme_tree("black")
```

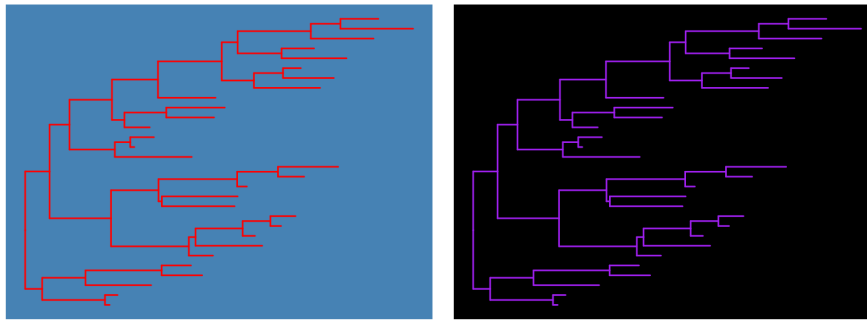


Figure 4.14: **ggtree theme.**

Users can also use image file as tree background, see example in Appendix B (page 256).

4.4 Visualize a list of trees

`ggtree` supports `multiPhylo` object and a list of trees can be viewed simultaneously.

```
trees <- lapply(c(10, 20, 40), rtree)
class(trees) <- "multiPhylo"
ggtree(trees) + facet_wrap(~.id, scale="free") +
        geom_tiplab()
```

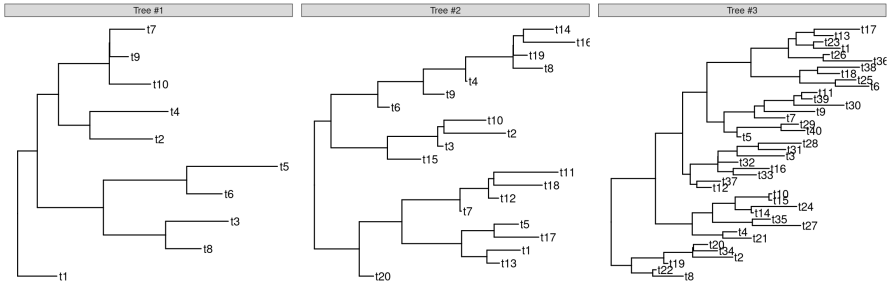


Figure 4.15: **Visuazlie multiPhylo object.**

One hundred bootstrap trees can also be view simultaneously.

```
btrees <- read.tree(system.file("extdata/RAxML",
        "RAxML_bootstrap.H3", package="treeio"))
ggtree(btrees) + facet_wrap(~.id, ncol=10)
```

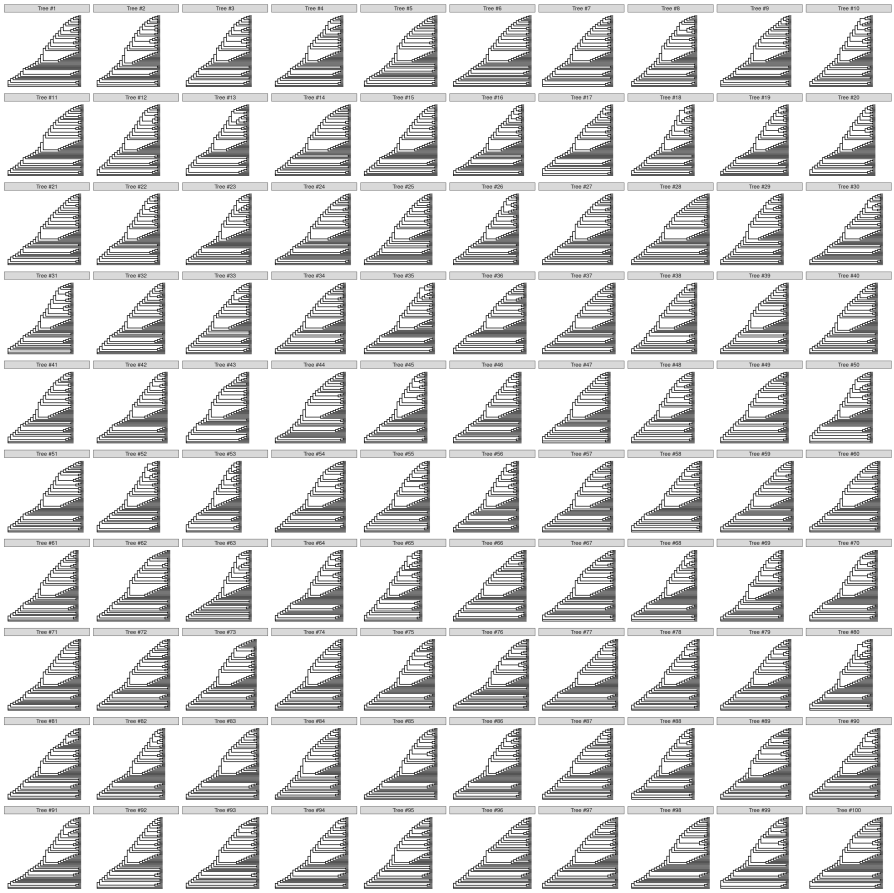


Figure 4.16: **Visualize one hundred bootstrap trees.**

Another way to view the bootstrap trees is to merge them together to form a density tree using `ggdensitree` function.

```
ggdensitree(btrees, alpha=.3, colour='steelblue') +
    geom_tiplab(size=3) + xlim(0, 45)
```

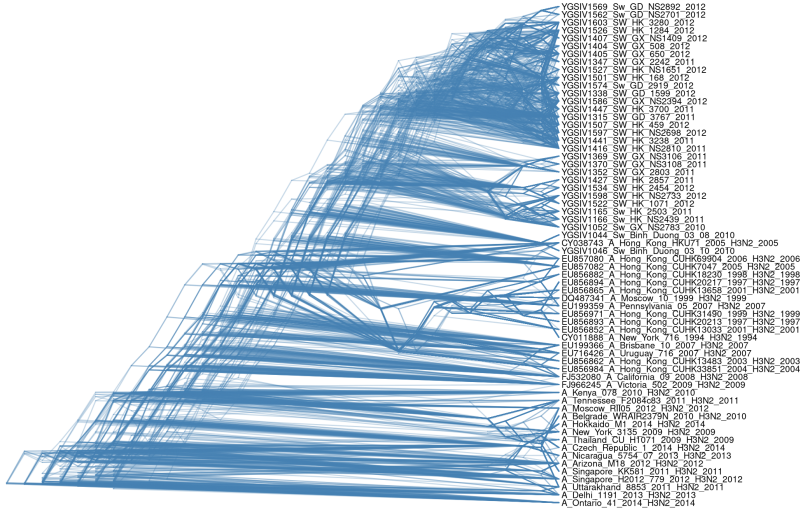


Figure 4.17: **DensiTree.**

4.5 Summary

Visualizing phylogenetic tree using ggtree (http://bioconductor.org/packages/ggtree) is easy by using a single command `ggtree(tree)`. The ggtree (http://bioconductor.org/packages/ggtree) package provides several geometric layers to display tree components such as tip labels, symbolic points for both external and internal nodes, root edge, *etc*. Associated data can be used to rescale branch lengths, color the tree and display on the tree. All these can be done by the ggplot2 grammar of graphic syntax. ggtree (http://bioconductor.org/packages/ggtree) also provides several layers that are specifically designed for tree annotation (see Chapter 5 (page 139)).

5

Phylogenetic Tree Annotation

5.1 Visualizing and Annotating Tree using Grammar of Graphics

The *ggtree* is designed for more general purpose or specific type of tree visualization and annotation. It supports grammar of graphics implemented in *ggplot2* and users can freely visualize/annotate a tree by combining several annotation layers.

```
library(ggtree)
treetext = "(((ADH2:0.1[&&NHX:S=human],
        ADH1:0.11[&&NHX:S=human]):
0.05 [&&NHX:S=primates:D=Y:B=100],ADHY:
0.1[&&NHX:S=nematode],ADHX:0.12 [&&NHX:S=insect]):
0.1[&&NHX:S=metazoa:D=N],(ADH4:0.09[&&NHX:S=yeast],
ADH3:0.13[&&NHX:S=yeast], ADH2:0.12[&&NHX:S=yeast],
ADH1:0.11[&&NHX:S=yeast]):0.1[&&NHX:S=Fungi])[&&NHX:D=N];"
tree <- read.nhx(textConnection(treetext))
ggtree(tree) + geom_tiplab() +
  geom_label(aes(x=branch, label=S), fill='lightgreen') +
  geom_label(aes(label=D), fill='steelblue') +
  geom_text(aes(label=B), hjust=-.5)
```

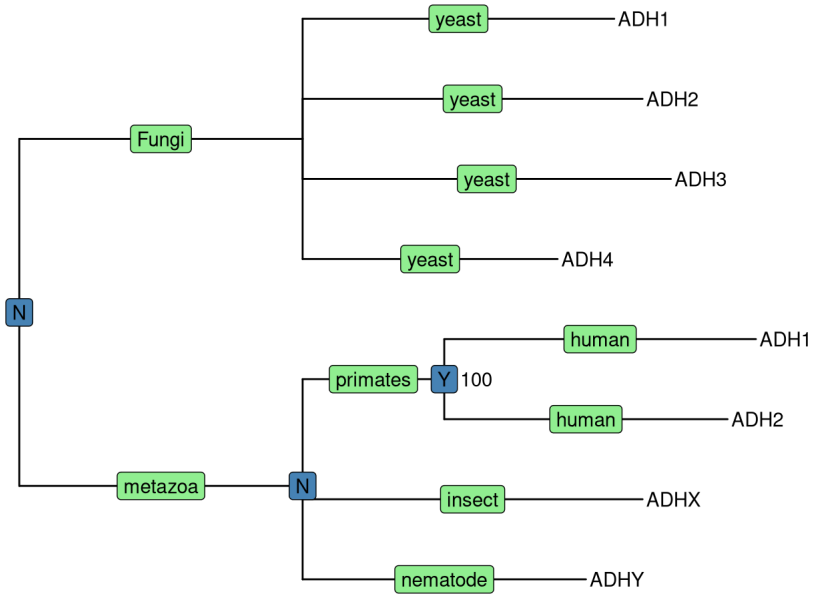


Figure 5.1: **Annotating tree using grammar of graphics.** The NHX tree was annotated using grammar of graphic syntax by combining different layers using + operator. Species information were labelled on the middle of the branches, Duplication events were shown on most recent common ancestor and clade bootstrap value were dispalyed near to it.

Here, as an example, we visualized the tree with several layers to display annotation stored in NHX tags, including a layer of `geom_tiplab` to display tip labels (gene name in this case), a layer using `geom_label` to show species information (`S` tag) colored by lightgreen, a layer of duplication event information (`D` tag) colored by steelblue and another layer using *geom_text* to show bootstrap value (`B` tag).

Layers defined in *ggplot2* can be applied to *ggtree* directly as demonstrated in Figure 5.1 of using *geom_label* and *geom_text*. But *ggplot2* does not provide graphic layers that are specific designed for phylogenetic tree annotation. For instance, layers for tip labels, tree branch scale legend, highlight or labeling clade are all unavailable. To make tree annotation more flexible, a number of layers have been implemented in *ggtree* (Table

5.1), enabling different ways of annotation on various parts/components of a phylogenetic tree.

Table 5.1: Geom layers defined in ggtree.

Layer	Description
geom_balance	highlights the two direct descendant clades of an internal node
geom_cladelabel	annotate a clade with bar and text label
geom_facet	plot associated data in specific panel (facet) and align the plot with the tree
geom_hilight	highlight a clade with rectangle
geom_inset	add insets (subplots) to tree nodes
geom_label2	modified version of geom_label, with subsetting supported
geom_nodepoint	annotate internal nodes with symbolic points
geom_point2	modified version of geom_point, with subsetting supported
geom_range	bar layer to present uncertainty of evolutionary inference
geom_rootpoint	annotate root node with symbolic point
geom_rootedge	add root edge to a tree
geom_segment2	modified version of geom_segment, with subsetting supported
geom_strip	annotate associated taxa with bar and (optional) text label
geom_taxalink	associate two related taxa by linking them with a curve
geom_text2	modified version of geom_text, with subsetting supported
geom_tiplab	layer of tip labels
geom_tippoint	annotate external nodes with symbolic points
geom_tree	tree structure layer, with multiple layout supported

geom_treescale	tree branch scale legend

5.2 Layers for Tree Annotation

5.2.1 Colored strips

ggtree (http://bioconductor.org/packages/ggtree) (Yu et al. 2017) implements *geom_cladelabel* layer to annotate a selected clade with a bar indicating the clade with a corresponding label.

The *geom_cladelabel* layer accepts a selected internal node number and label corresponding clade automatically (Figure 5.2A). To get the internal node number, please refer to Chapter 2 (page 46).

```
set.seed(2015-12-21)
tree <- rtree(30)
p <- ggtree(tree) + xlim(NA, 6)

p + geom_cladelabel(node=45, label="test label") +
    geom_cladelabel(node=34, label="another clade")
```

Users can set the parameter, `align = TRUE`, to align the clade label, `offset`, to adjust the position and color to set the color of bar and label text *etc* (Figure 5.2B).

```
p + geom_cladelabel(node=45, label="test label", align=TRUE,
         offset = .2, color='red') +
    geom_cladelabel(node=34, label="another clade",
         align=TRUE, offset = .2, color='blue')
```

Users can change the `angle` of the clade label text and relative position from text to bar via the parameter `offset.text`. The size of the bar and text can be changed via the parameters `barsize` and `fontsize` respectively (Figure 5.2C).

```
p + geom_cladelabel(node=45, label="test label", align=T,
        angle=270, hjust='center', offset.text=.5,
        barsize=1.5) +
    geom_cladelabel(node=34, label="another clade", align=T,
        angle=45, fontsize=8)
```

Users can also use `geom_label` to label the text and can set the background color by `fill` parameter (Figure 5.2D).

```
p + geom_cladelabel(node=34, label="another clade", align=T,
        geom='label', fill='lightblue')
```

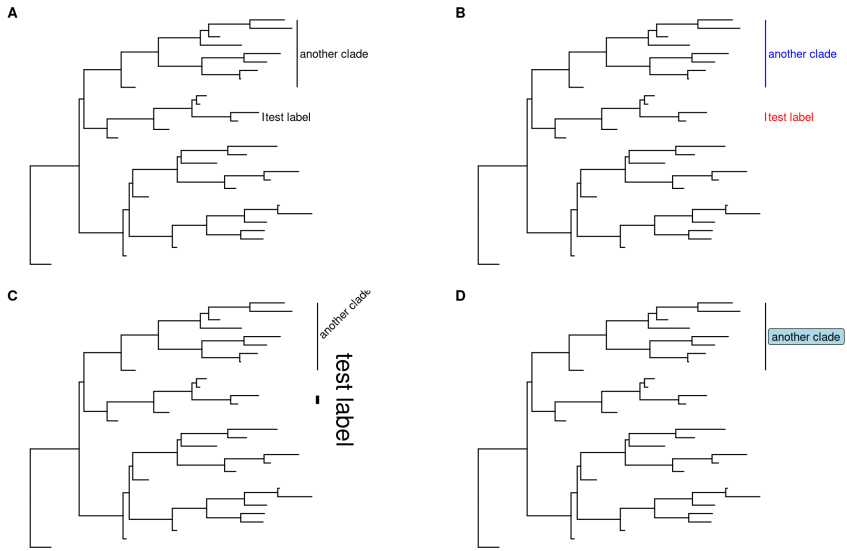


Figure 5.2: **Labeling clades.**

`geom_cladelabel` also supports unrooted tree layouts (Figure 5.3A).

```
ggtree(tree, layout="daylight") +
  geom_cladelabel(node=35, label="test label", angle=0,
                  fontsize=8, offset=.5, vjust=.5)  +
```

```
    geom_cladelabel(node=55, label='another clade',
                    angle=-95, hjust=.5, fontsize=8)
```

`geom_cladelabel` is designed for labeling Monophyletic (Clade) while there are related taxa that are not form a clade. `ggtree` provides `geom_strip` to add a strip/bar to indicate the association with optional label for Polyphyletic or Paraphyletic (Figure 5.3B).

```
p + geom_tiplab() +
  geom_strip('t10', 't30', barsize=2, color='red',
             label="associated taxa", offset.text=.1) +
  geom_strip('t1', 't18', barsize=2, color='blue',
             label = "another label", offset.text=.1)
```

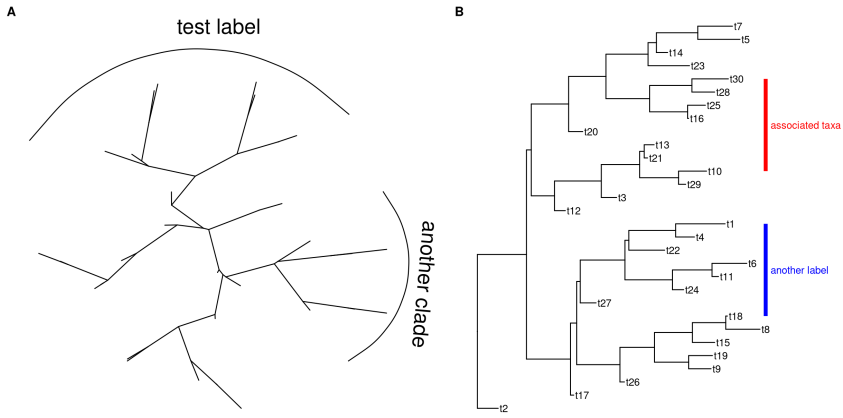


Figure 5.3: **Labeling associated taxa.** `geom_cladelabel` is for labeling Monophyletic and it also supports unrooted layout (A). `geom_strip` is designed for labeling associated taxa (Monophyletic, Polyphyletic or Paraphyletic) (B).

5.2.2 Highlight clades

`ggtree` implements *`geom_hilight`* layer, that accepts an internal node number and add a layer of rectangle to highlight the selected clade Figure (5.4).[13]

```
nwk <- system.file("extdata", "sample.nwk",
         package="treeio")
tree <- read.tree(nwk)
ggtree(tree) + geom_hilight(node=21, fill="steelblue",
         alpha=.6) +
    geom_hilight(node=17, fill="darkgreen", alpha=.6)
```

```
ggtree(tree, layout="circular") + geom_hilight(node=21,
         fill="steelblue", alpha=.6) +
    geom_hilight(node=23, fill="darkgreen", alpha=.6)
```

The `geom_hilight` layer also support highlighting clades for unrooted layout trees (Figure 5.4C).

```
pg + geom_hilight(node=55) + geom_hilight(node=35,
         fill='darkgreen')
```

Another way to highlight selected clades is setting the clades with different colors and/or line types as demonstrated in Figure 6.2.

In addition to `geom_hilight`, `ggtree` also implements `geom_balance` which is designed to highlight neighboring subclades of a given internal node (Figure 5.4D).

```
ggtree(tree) +
  geom_balance(node=16, fill='steelblue', color='white',
         alpha=0.6, extend=1) +
  geom_balance(node=19, fill='darkgreen', color='white',
         alpha=0.6, extend=1)
```

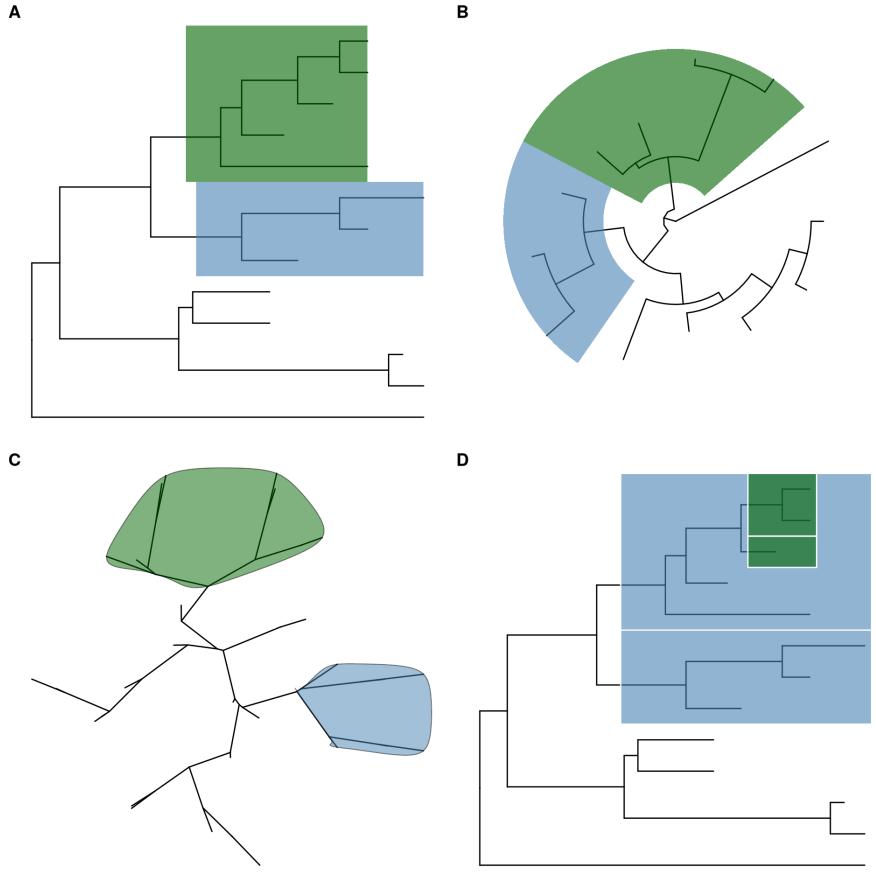


Figure 5.4: **Highlight selected clades.** Rectangular layout (A), circular/fan (B) and unrooted layouts. Highlight neighboring subclades simultaneously (D).

5.2.3 Taxa connection

Some evolutionary events (e.g. reassortment, horizontal gene transfer) cannot be modeled by a simple tree. ggtree (http://bioconductor.org/packages/ggtree) provides `geom_taxalink` layer that allows drawing straight or curved lines between any of two nodes in the tree, allow it to represent evolutionary events by connecting taxa.

```
ggtree(tree) + geom_tiplab() + geom_taxalink('A', 'E') +
  geom_taxalink('F', 'K', color='red', linetype = 'dashed',
    arrow=grid::arrow(length=grid::unit(0.02, "npc")))
```

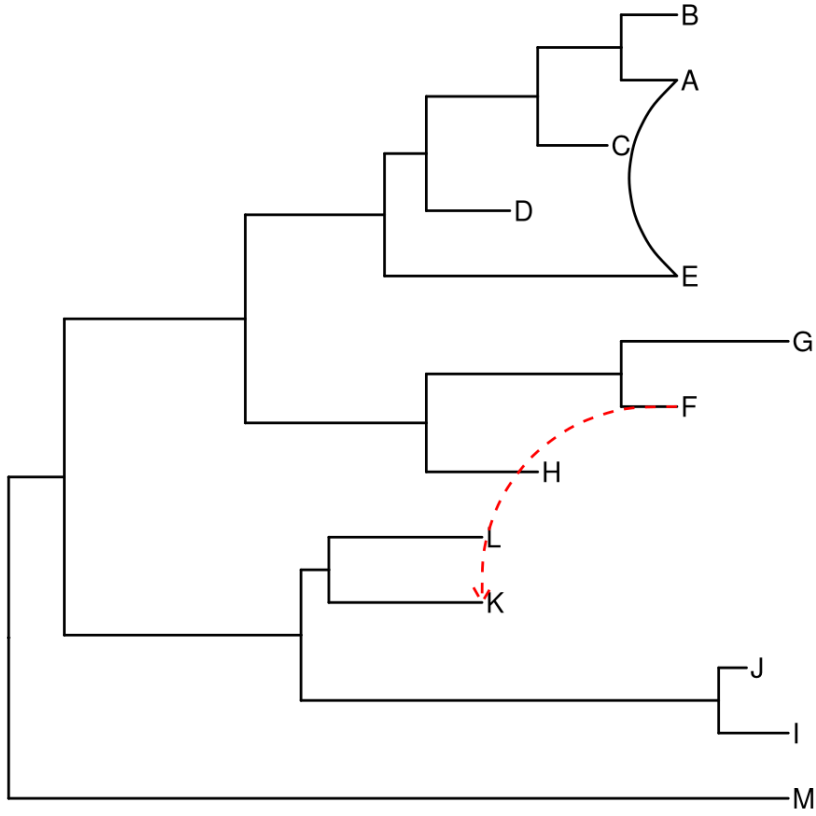


Figure 5.5: **Linking related taxa.** This can be used to indicate evolutionary events such as reassortment and horozontal gene transfer.

5.2.4 Uncertainty of evolutionary inference

The `geom_range` layer supports displaying interval (highest posterior density, confidence interval, range) as horizontal bars on tree nodes. The center of the interval will anchor to corresponding node. The center by default is the mean value of the interval (Figure 5.6A). We can set the `center` to estimated mean or median value (Figure 5.6B), or observed value. As the tree branch and the interval may not be in the same scale, ggtree (http://bioconductor.org/packages/ggtree) provides `scale_x_range` to add second x axis for the range (Figure 5.6C). Note that x axis is disable by default theme and we need to enable it if we want to dispaly it (*e.g.* `theme_tree2`).

```
file <- system.file("extdata/MEGA7", "mtCDNA_timetree.nex", package = "t
x <- read.mega(file)
p1 <- ggtree(x) + geom_range('reltime_0.95_CI', color='red', size=3, alp
p2 <- ggtree(x) + geom_range('reltime_0.95_CI', color='red', size=3, alp
p3 <- p2 + scale_x_range() + theme_tree2()
```

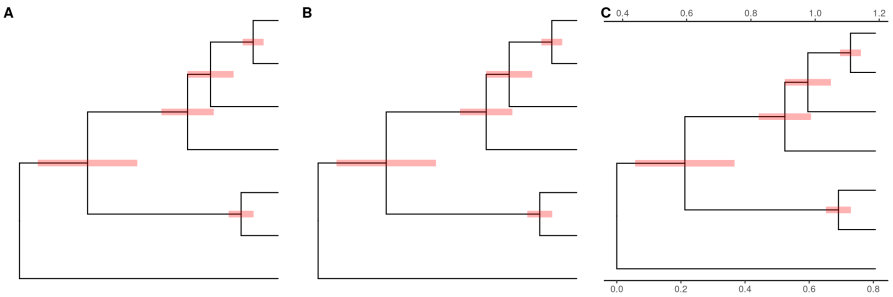


Figure 5.6: **Displaying uncertainty of evolutoinary inference.** The center (mean value of the range (A) or estimated value (B)) is anchor to the tree nodes. A second x axis was used for range scaling (C).

5.3 Tree annotation with output from evolution software

5.3.1 Tree annotation using data from evolutionary analysis software

Chapter 1 (page 0) introduced using treeio (http://bioconductor.org/packages/treeio) packages to parse different tree formats and commonly used software outputs to obtain phylogeny-associated data. These imported data as S4 objects can be visualized directly using ggtree (http://bioconductor.org/packages/ggtree). Figure 5.1 demonstrates a tree annotated using the information (species classification, duplication event and bootstrap value) stored in NHX file. *PHYLDOG* and *RevBayes* output NHX files that can be parsed by treeio (http://bioconductor.org/packages/treeio) and visualized by ggtree (http://bioconductor.org/packages/ggtree) with annotation using their inference data.

Furthermore, the evolutionary data from the inference of *BEAST*, *MrBayes* and *RevBayes*, d_N/d_S values inferred by *CodeML*, ancestral sequences inferred by *HyPhy*, *CodeML* or *BaseML* and short read placement by *EPA* and *pplacer* can be used to annotate the tree directly.

```
file <- system.file("extdata/BEAST", "beast_mcc.tree",
        package="treeio")
beast <- read.beast(file)
ggtree(beast, aes(color=rate))  +
    geom_range(range='length_0.95_HPD', color='red',
        alpha=.6, size=2) +
    geom_nodelab(aes(x=branch, label=round(posterior, 2)),
        vjust=-.5, size=3) +
    scale_color_continuous(low="darkgreen", high="red") +
    theme(legend.position=c(.1, .8))
```

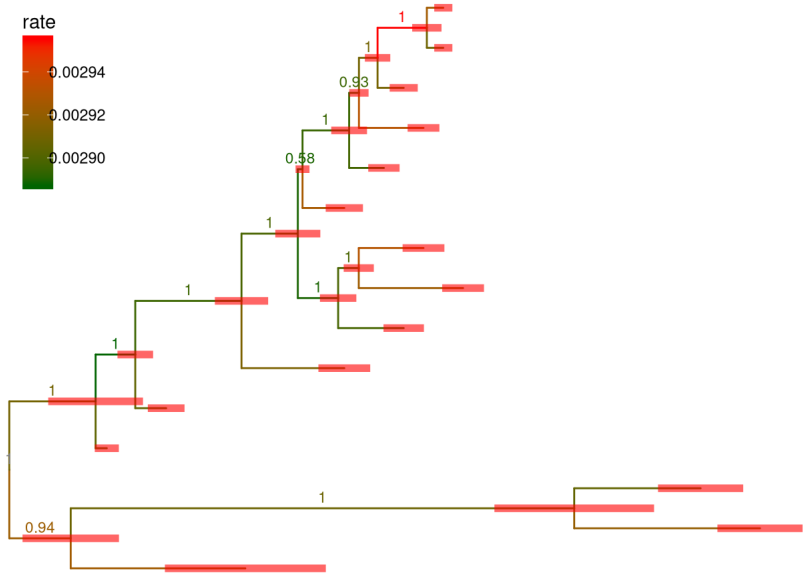


Figure 5.7: **Annotating *BEAST* tree with *length_95%_HPD* and posterior.** Branch length credible intervals (95% HPD) were displayed as red horizontal bars and clade posterior values were shown on the middle of branches.

In Figure 5.7, the tree was visualized and annotated with posterior > 0.9 and demonstrated length uncertainty (95% Highest Posterior Density (HPD) interval).

Ancestral sequences inferred by *HyPhy* can be parsed using *treeio* (page 32), whereas the substitutions along each tree branch was automatically computed and stored inside the phylogenetic tree object (*i.e.*, S4 object). The *ggtree* can utilize this information in the object to annotate the tree, as demonstrated in Figure 5.8.

```
nwk <- system.file("extdata/HYPHY", "labelledtree.tree",
                   package="treeio")
ancseq <- system.file("extdata/HYPHY", "ancseq.nex",
                      package="treeio")
tipfas <- system.file("extdata", "pa.fas", package="treeio")
```

```
hy <- read.hyphy(nwk, ancseq, tipfas)
ggtree(hy) +
  geom_text(aes(x=branch, label=AA_subs), size=2,
            vjust=-.3, color="firebrick")
```

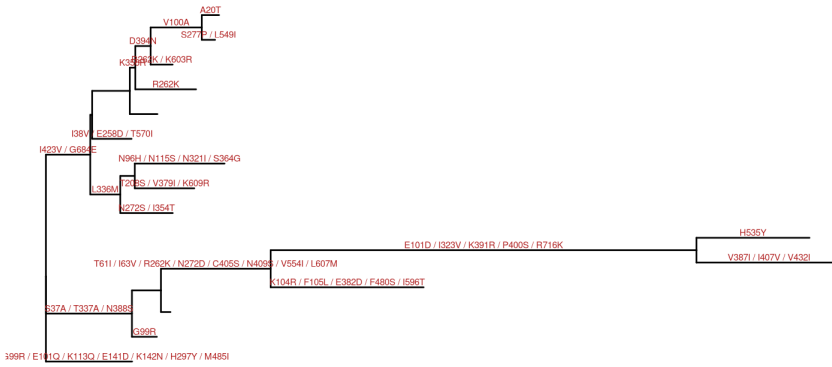


Figure 5.8: **Annotating tree with amino acid substitution determined by ancestral sequences inferred by HYPHY.** Amino acid substitutions were displayed on the middle of branches.

PAML's *BaseML* and *CodeML* can be also used to infer ancestral sequences, whereas *CodeML* can infer selection pressure. After parsing this information using *treeio* (page 29), *ggtree* can integrate this information into the same tree structure and used for annotation as illustrated in Figure 5.9.

```
rstfile <- system.file("extdata/PAML_Codeml", "rst",
                       package="treeio")
mlcfile <- system.file("extdata/PAML_Codeml", "mlc",
                       package="treeio")
ml <- read.codeml(rstfile, mlcfile)
ggtree(ml, aes(color=dN_vs_dS), branch.length='dN_vs_dS') +
  scale_color_continuous(name='dN/dS', limits=c(0, 1.5),
                         oob=scales::squish,
                         low='darkgreen', high='red') +
  geom_text(aes(x=branch, label=AA_subs),
            vjust=-.5, color='steelblue', size=2) +
  theme_tree2(legend.position=c(.9, .3))
```

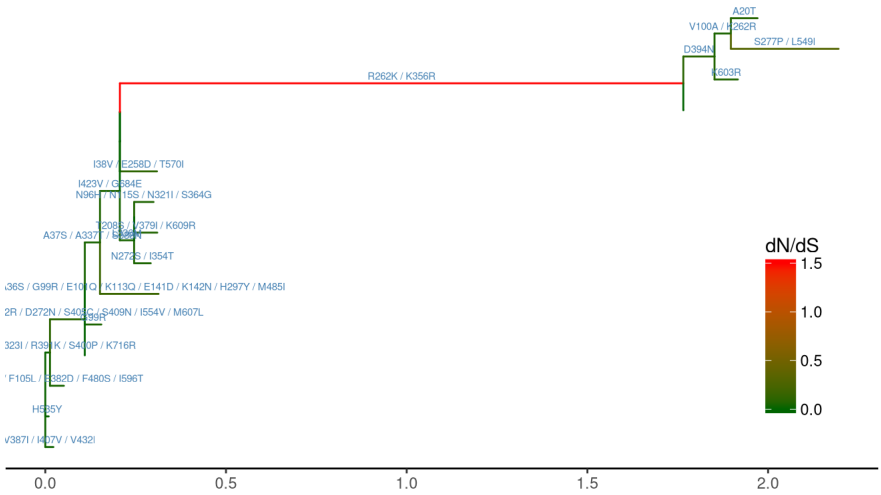


Figure 5.9: **Annotating tree with animo acid substitution and d_N/d_S inferred by *CodeML*.** Branches were rescaled and colored by d_N/d_S values and amino acid substitutions were displayed on the middle of branches.

Not only all the tree data that parsed by treeio (http://bioconductor.org/packages/treeio) can be used to visualize and annotate phylogenetic tree using `ggtree`, but also other tree and tree-like objects defined in R community are supported. The ggtree (http://bioconductor.org/packages/ggtree) plays an unique role in R ecosystem to facilitate phylogenetic analysis and it can be easily integrated into other packages and pipelines. For more details, please refer to chapter 9 (page 199). In addition to direct support of tree objects, ggtree (http://bioconductor.org/packages/ggtree) also allow users to plot tree with different types of external data (see also chapter 7 (page 171) and (Yu et al. 2018 (page 282))).

5.4 Summary

ggtree (http://bioconductor.org/packages/ggtree) implements grammar of graphics for annotating phylogenetic trees. Users can use ggplot2 syntax to combine different annotation layers to produce complex tree

annotation. If you are familiar with ggplot2 (https://CRAN.R-project.org/package=ggplot2), tree annotation with high level of customization can be intuitive and flexible using ggtree (http://bioconductor.org/packages/ggtree).

6

Visual Exploration of Phylogenetic Tree

The ggtree (http://bioconductor.org/packages/ggtree) supports many ways of manipulating the tree visually, including viewing selected clade to explore large tree (Figure 6.1), taxa clustering (Figure 6.5), rotating clade or tree (Figure 6.6B and 6.8), zoom out or collapsing clades (Figure 6.3A and 6.2), *etc.*. Details tree manipulation functions are summarized in Table 6.1.

Table 6.1: Tree manipulation functions.

Function	**Descriptiotn**
collapse	collapse a selecting clade
expand	expand collapsed clade
flip	exchange position of 2 clades that share a parent node
groupClade	grouping clades
groupOTU	grouping OTUs by tracing back to most recent common ancestor
identify	interactive tree manipulation
rotate	rotating a selected clade by 180 degree
rotate_tree	rotating circular layout tree by specific angle
scaleClade	zoom in or zoom out selecting clade
open_tree	convert a tree to fan layout by specific open angle

6.1 Viewing Selected Clade

A clade is a monophyletic group that contains a single ancestor and all of its descendants. We can visualize a specific selected clade via the `viewClade` function as demonstrated in Figure 6.1B. Another similar function is `gzoom` which plots the tree with selected clade side by side. These two functions are developed to explore large tree.

```
library(ggtree)
nwk <- system.file("extdata", "sample.nwk",
         package="treeio")
tree <- read.tree(nwk)
p <- ggtree(tree) + geom_tiplab()
viewClade(p, MRCA(p, "I", "L"))
```

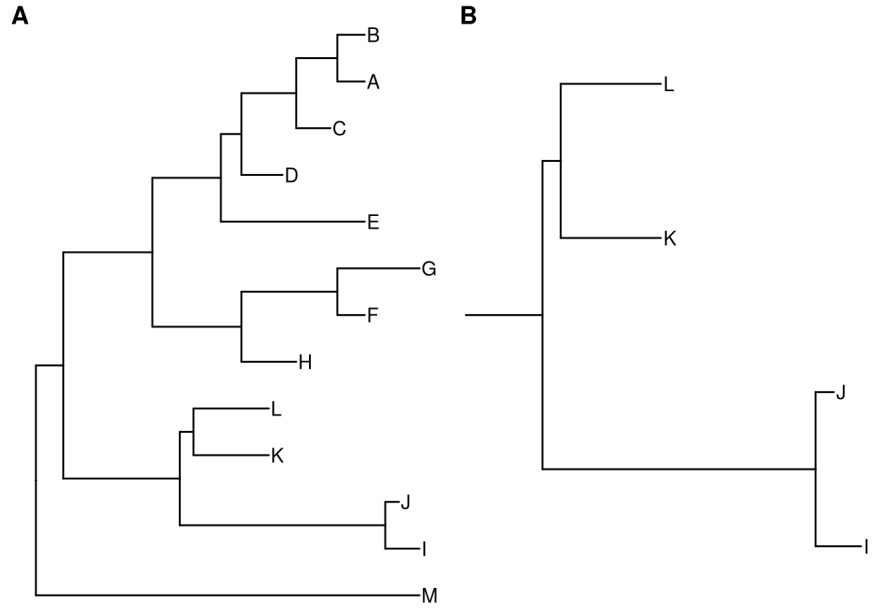


Figure 6.1: **Viewing a selected clade of a tree.** An example tree used to demonstrate how *ggtree* support exploring or manipulating phylogenetic tree visually (A). The *ggtree* supports visualizing selected clade (B). A clade can be selected by specifying a node number or determined by most recent common ancestor of selected tips.

Some of the functions, *e.g.* `viewClade`, work with clade and accept a parameter of internal node number. To get the internal node number, user can use `MRCA()` function (as in Figure 6.1) by providing two taxa names. The function will return node number of input taxa's most recent commond ancestor (MRCA). It works with tree and graphic (*i.e.* `ggtree()` output) object. tidytree (https://CRAN.R-project.org/package=tidytree) also provide `MRCA` function to extract information of MRCA node (see details in session 2.1.3 (page 46)).

6.2 Scaling Selected Clade

The *ggtree* provides another option to zoom out (or compress) these clades via the `scaleClade` function. In this way, we retain the topology and branch lengths of compressed clades. This helps to save the space to highlight those clades of primary interest to the study.

```
tree2 <- groupClade(tree, c(17, 21))
p <- ggtree(tree2, aes(color=group)) +
         theme(legend.position='none') +
  scale_color_manual(values=c("black", "firebrick",
         "steelblue"))
scaleClade(p, node=17, scale=.1)
```

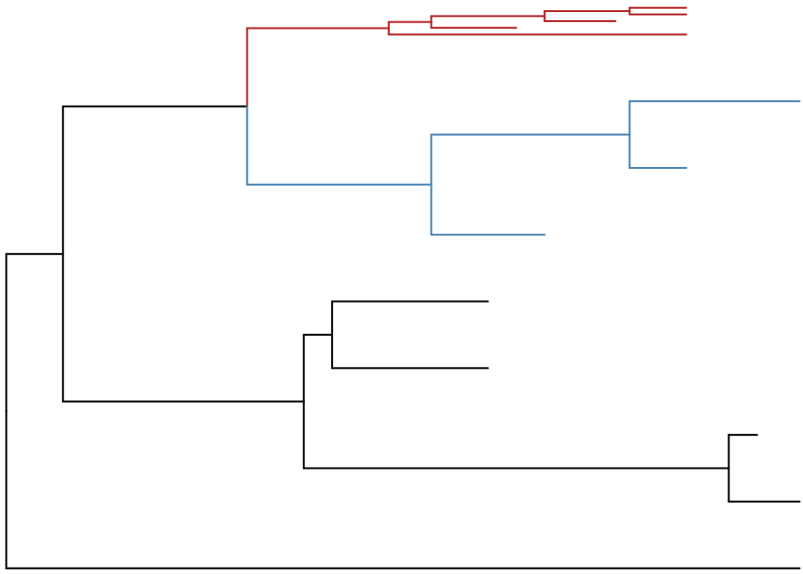


Figure 6.2: **Scaling selected clade.** Clades can be zoom in (if `scale > 1`) to highlight or zoom out to save space.

If users want to emphasize important clades, they can use `scaleClade` function with `scale` parameter larger than 1. Then the selected clade will be zoomed in. Users can also use `groupClade` to select clades and color them with different colors as shown in Figure 6.2.

6.3 Collapsing and Expanding Clade

It is a common practice to prune or collapse clades so that certain aspects of a tree can be emphasized. The *ggtree* supports collapsing selected clades using the `collapse` function as shown in Figure 6.3A.

```
p2 <- p %>% collapse(node=21) +
  geom_point2(aes(subset=(node==21)), shape=21, size=5,
         fill='green')
p2 <- collapse(p2, node=23) +
  geom_point2(aes(subset=(node==23)), shape=23, size=5,
         fill='red')
print(p2)
expand(p2, node=23) %>% expand(node=21)
```

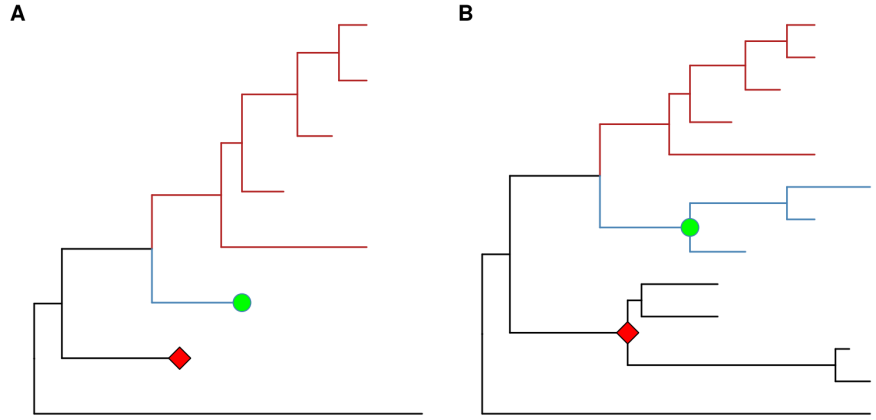


Figure 6.3: **Collapsing selected clades and expanding collapsed clades.** Clades can be selected to collapse (A) and the collapsed clades can be expanded back (B) if necessary as *ggtree* stored all information of species relationships. Green and red symbols were displayed on the tree to indicate the collapsed clades.

Here two clades were collapsed and labelled by green circle and red square symbolic points. Collapsing is a common strategy to collapse clades that are too large for displaying in full or are not primary interest of the study. In *ggtree*, we can expand (*i.e.*, uncollapse) the collapsed branches back with `expand` function to show details of species relationships as demonstrated in Figure 6.3B.

Triangles are often used to represent the collapsed clade and ggtree (http://bioconductor.org/packages/ggtree) also supports it. The `collapse` function provides a "mode" parameter, which by default is "none" and the selected clade was collapsed as a "tip". User can specify `mode` to "max" (Figure 6.4A), "min" (Figure 6.4B) and "mixed" (Figure 6.4C).

```
p2 <- p + geom_tiplab()
node <- 21
collapse(p2, node, 'max') %>% expand(node)
collapse(p2, node, 'min') %>% expand(node)
collapse(p2, node, 'mixed') %>% expand(node)
```

We can pass additional parameter to set the color and transparency of the triangles (Figure 6.4D).

```
collapse(p, 21, 'mixed', fill='steelblue', alpha=.4) %>%
  collapse(23, 'mixed', fill='firebrick', color='blue')
```

We can combine scaleClade (page 157) with `collapse` to zoom in/out of the triangles (Figure 6.4E).

```
scaleClade(p, 23, .2) %>% collapse(23, 'min',
         fill="darkgreen")
```

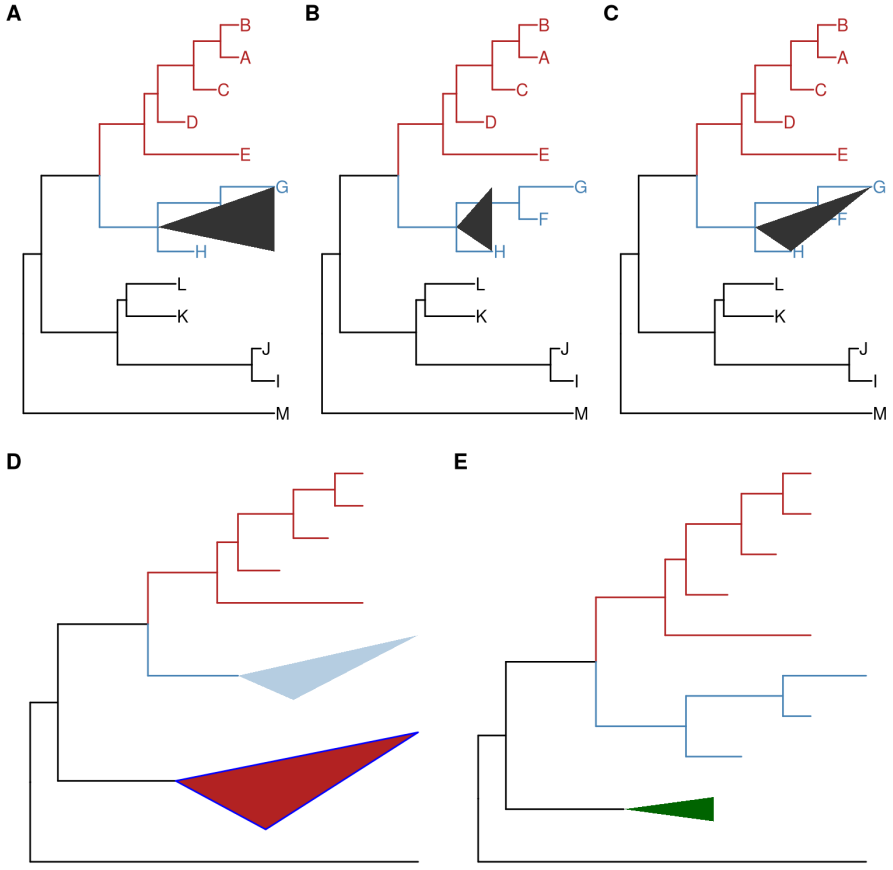


Figure 6.4: **Collapse clade as triangle.** 'max' takes the position of most distant tip (A). 'min' takes the position of closest tip (B). 'mixed' takes the positions of both closest and distant tips (C), which looks more like the shape of the clade. Set color, fill and alpha of the triangles (D). Combine with `scaleClade` to zoom out triangle to save space (E).

6.4 Grouping Taxa

The `groupClade` function assigns the branches and nodes under different clades into different groups. *`groupClade`* accepts an internal node or a vector of internal nodes to cluster clade/clades.

Similarly, `groupOTU` function assigns branches and nodes to different groups based on user-specified groups of operational taxonomic units (OTUs) that are not necessarily within a clade, but can be monophyletic (clade), polyphyletic or paraphyletic. It accepts a vector of OTUs (taxa name) or a list of OTUs and will trace back from OTUs to their most recent common ancestor (MRCA) and cluster them together as demonstrated in Figure 6.5.

A phylogenetic tree can be annotated by mapping different line type, size, color or shape to the branches or nodes that have been assigned to different groups.

```
data(iris)
rn <- paste0(iris[,5], "_", 1:150)
rownames(iris) <- rn
d_iris <- dist(iris[,-5], method="man")

tree_iris <- ape::bionj(d_iris)
grp <- list(setosa     = rn[1:50],
            versicolor = rn[51:100],
            virginica  = rn[101:150])

p_iris <- ggtree(tree_iris, layout = 'circular',
         branch.length='none')
groupOTU(p_iris, grp, 'Species') + aes(color=Species) +
  theme(legend.position="right")
```

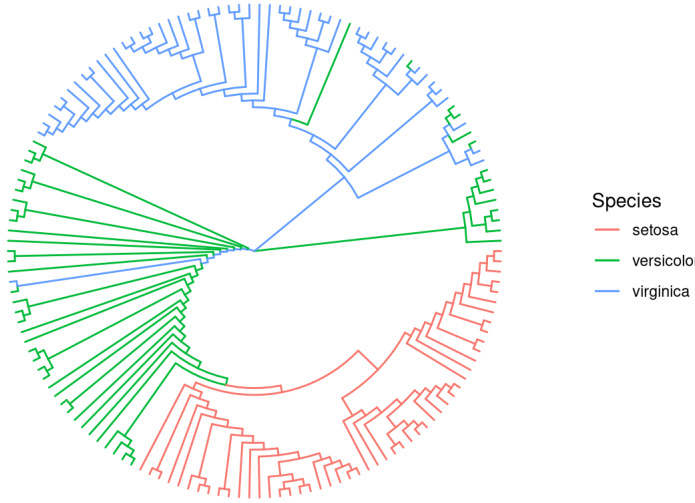


Figure 6.5: **Grouping OTUs.** OTU clustering based on their relationships. Selected OTUs and their ancestors upto MRCA will be clustered together.

We can grouping taxa at tree level. The following code will produce identical figure of Figure 6.5 (see more details described at session 2.2.3 (page 55)).

```
tree_iris <- groupOTU(tree_iris, grp, "Species")
ggtree(tree_iris, aes(color=Species), layout = 'circular',
         branch.length = 'none') +
  theme(legend.position="right")
```

6.5 Exploring tree structure

To facilitate exploring the tree structure, *ggtree* supports rotating selected clade by 180 degree using the `rotate` function (Figure 6.6B). Position of

immediate descendant clades of internal node can be exchanged via `flip` function (Figure 6.6C).

```
p1 <- p + geom_point2(aes(subset=node==16),
          color='darkgreen', size=5)
p2 <- rotate(p1, 17) %>% rotate(21)
flip(p2, 17, 21)
```

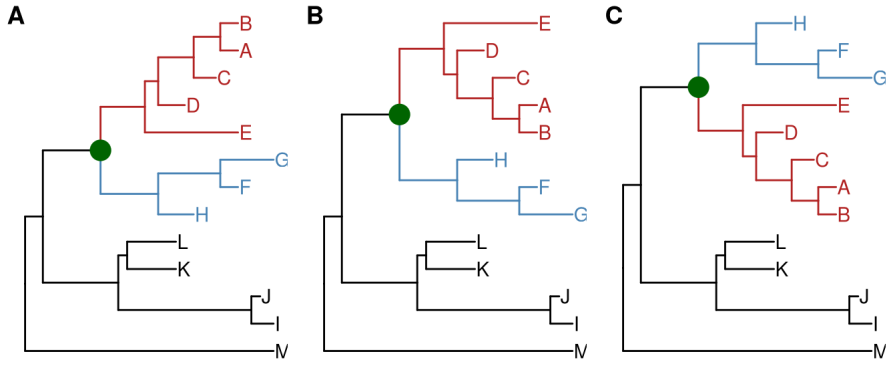


Figure 6.6: **Exploring tree structure.** A clade (indicated by darkgreen circle) in a tree (A) can be rotated by 180° (B) and the positions of its immediate descedant clades (colored by blue and red) can be exchanged (C).

Most of the tree manipulation functions are working on clades, while ggtree (http://bioconductor.org/packages/ggtree) also provides functions to manipulate a tree, including `open_tree` to transform a tree in either rectangular or circular layout to fan layout, and `rotate_tree` function to rotate a tree for specific angle in both circular or fan layouts, as demonstrated in Figure 6.7 and 6.8.

```
p3 <- open_tree(p, 180) + geom_tiplab()
print(p3)
```

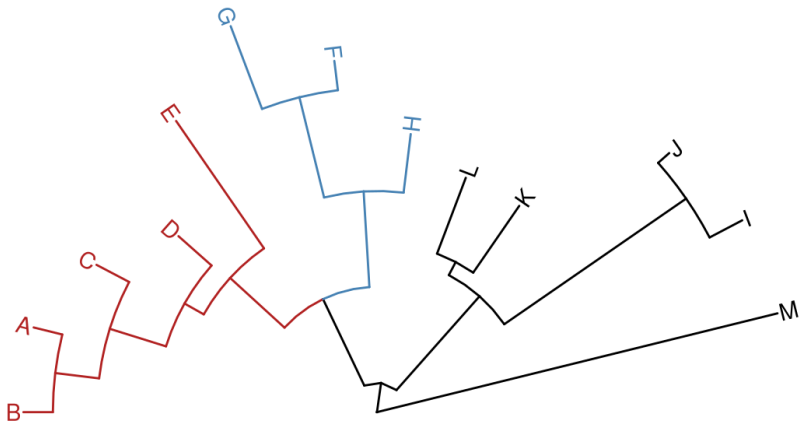


Figure 6.7: **Transforming a tree to fan layout.** A tree can be transformed to fan layout by `open_tree` with specific *angle* parameter.

```
rotate_tree(p3, 180)
```

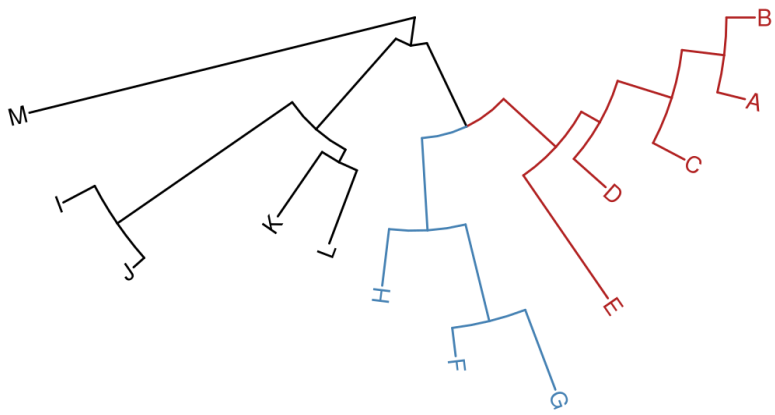


Figure 6.8: **Rotating tree.** A circular/fan layout tree can be rotated by any specific *angle*.

The following example traverse all the internal nodes and rotate them one by one (Figure 6.9).

```
set.seed(2016-05-29)
x <- rtree(50)
p <- ggtree(x) + geom_tiplab()
for (n in reorder(x, 'postorder')$edge[,1] %>% unique) {
    p <- rotate(p, n)
    print(p + geom_point2(aes(subset=(node == n)),
         color='red'))
}
```

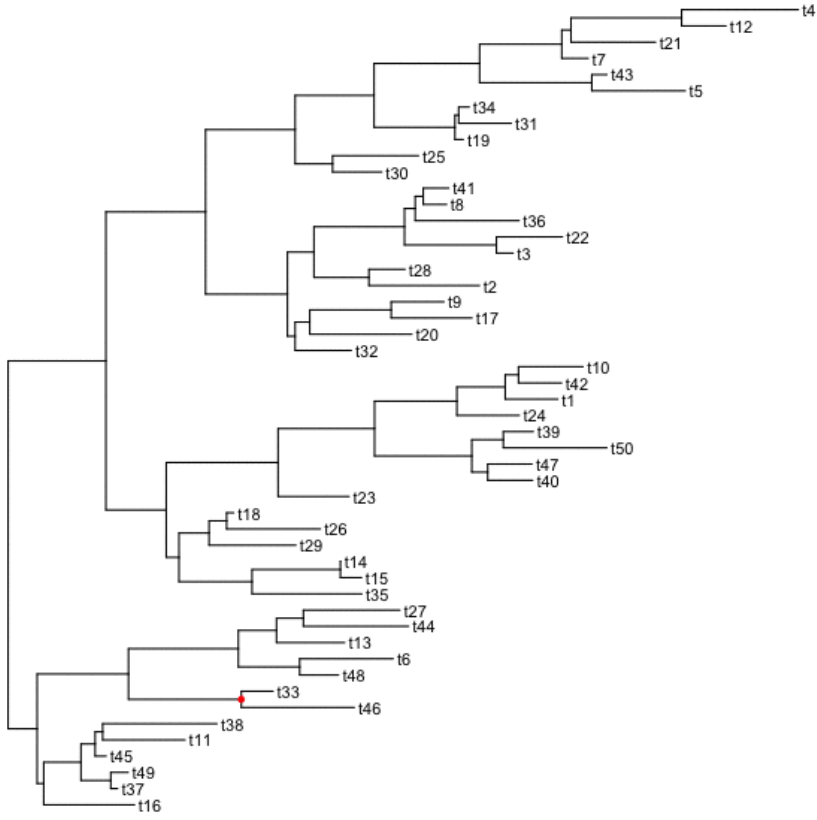


Figure 6.9: **Traverse and rotate all clades.**

```
set.seed(123)
tr <- rtree(50)
p <- ggtree(tr, layout='circular') + geom_tiplab()

for (angle in seq(0, 270, 10)) {
    print(open_tree(p, angle=angle) + ggtitle(paste("open
        angle:", angle)))
}
```

Figure 6.10 demonstrates the usage of `open_tree` with different open angles.

Figure 6.10: **Open tree with different angles.**

Figure 6.11 illustrates rotating tree with different angles.

```
for (angle in seq(0, 270, 10)) {
    print(rotate_tree(p, angle) + ggtitle(paste("rotate
        angle:", angle)))
}
```

Figure 6.11: **Rotate tree with different angles.**

Interactive tree manipulation is also possible via `identify` methods[14].

6.6 Summary

The ggtree (http://bioconductor.org/packages/ggtree) provides a set of functions to allow visually manipulating phylogenetic tree and exploring tree structure with associated data.

7

Plotting tree with data

Integrating user data to annotate phylogenetic tree can be done at different levels. The treeio (http://bioconductor.org/packages/treeio) package implements `full_join` methods to combine tree data to phylogenetic tree object (https://bioconductor.org/packages/devel/bioc/vignettes/treeio/inst/doc/Importer.html). The tidytree (https://CRAN.R-project.org/package=tidytree) package supports linking tree data to phylogeny using tidyverse verbs (https://cran.r-project.org/web/packages/tidytree/vignette/tiytree.html). ggtree (http://bioconductor.org/packages/ggtree) supports mapping external data to phylogeny for visualization and annotation on the fly. Although the feature of linking external data is overlapping among these packages, they have different application scopes. For example, in addition to the `treedata` object, ggtree (http://bioconductor.org/packages/ggtree) also supports several other tree objects (see chapter 9 (page 199)), including `phylo4d`, `phyloseq` and `obkData` that were designed to contain domain specific data. The design of these objects did not consider to support linking external data to the object (it can not be done at tree object level). We can visualize trees from these objects using ggtree (http://bioconductor.org/packages/ggtree) and link external data at visualization level (Yu et al. 2018 (page 282)).

ggtree (http://bioconductor.org/packages/ggtree) provides two general methods for mapping and visualizing associated external data on phylogenies. Method 1 (page 172) allows external data to be mapped on the tree structure and used as visual characteristic in tree and data visualization. Method 2 (page 173) plots the data with the tree side by side using different geometric functions after reordering the data based on the tree structure. These two methods integrate data with phylogeny for further exploration and comparison in the evolutionary biology context.

7.1 Mapping data to the tree structure

In ggtree (http://bioconductor.org/packages/ggtree), we implemented an operator, %<+%, to attach annotation data to a ggtree graphic object. Any data that contains a column of "node" or first column of taxa labels can be integrated using the %<+% operator. Multiple datasets can be attached progressively. When the data are attached, all the information stored in the data serve as numerical/categorical node attributes and can be directly used to visualize the tree by scaling the attributes as different colors or line sizes, label the tree using the original values of the attributes or parsing them as math expression (page 244), emoji (page 193) or silhouette image (page 185). The following example uses the %<+% operator to integrat taxon (tip_data.csv) and internal node (inode_data.csv) information and map the data to different colors or shapes of symbolic points and labels (Figure 7.1). The tip data contains imageURL that links to online figures of the species, which can be parsed and used as tip labels in ggtree (http://bioconductor.org/packages/ggtree) (see chapter 8 (page 185)).

```
library(ggimage)
library(ggtree)
url <- paste0("https://raw.githubusercontent.com/TreeViz/",
              "metastyle/master/design/viz_targets_exercise/")

x <- read.tree(paste0(url, "tree_boots.nwk"))
info <- read.csv(paste0(url, "tip_data.csv"))

p <- ggtree(x) %<+% info + xlim(-.1, 4)
p2 <- p + geom_tiplab(offset = .6, hjust = .5) +
    geom_tippoint(aes(shape = trophic_habit, color =
        trophic_habit, size = mass_in_kg)) +
    theme(legend.position = "right") +
        scale_size_continuous(range = c(3, 10))

d2 <- read.csv(paste0(url, "inode_data.csv"))
```

```
p2 %<+% d2 + geom_label(aes(label = vernacularName.y, fill =
        posterior)) +
    scale_fill_gradientn(colors =
        RColorBrewer::brewer.pal(3, "YlGnBu"))
```

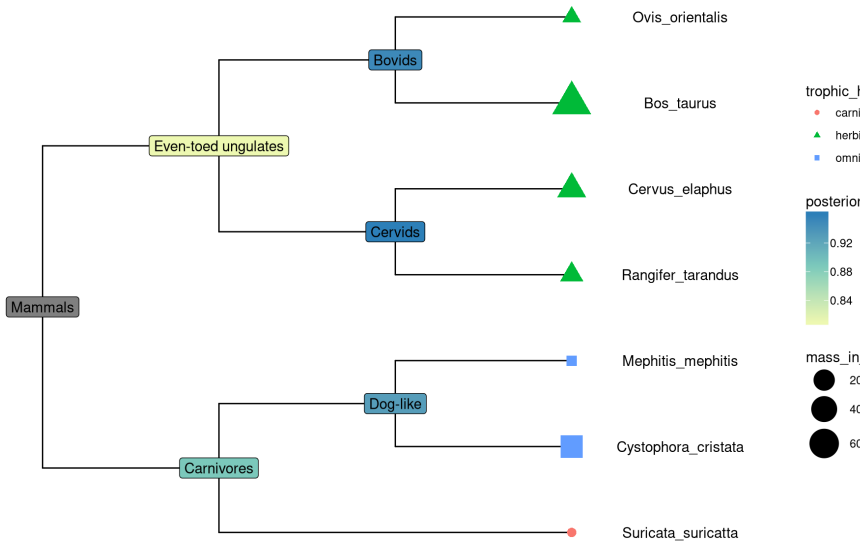


Figure 7.1: **Example of attaching multiple datasets**.

Although the data integrated by the %<+% operator in ggtree (http://bioconductor.org/packages/ggtree) is for tree visualization, the data attached to the ggtree graphic object can be converted to treedata object that contains the tree and the attached data (see session 7.5 (page 181)).

7.2 Alignning graph to the tree based on tree structure

For associating phylogenetic tree with different type of plot produced by user's data, ggtree provides geom_facet layer and facet_plot function which accept an input data.frame and a geom function to draw the input data. The data will be displayed in an additional panel of the plot. The

geom_facet (or facet_plot) is a general solution for linking graphic layer to a tree. The function internally re-orders the input data based on the tree strucutre and visualizes the data at the specific panel by the geometric layer. Users are free to visualize several panels to plot different types of data as demonstrated in Figure 9.4 and to use different geometric layers to plot the same dataset (Figure 11.1) or different datasets on the same panel.

The geom_facet is designed to work with most of the geom layers defined in ggplot2 (https://CRAN.R-project.org/package=ggplot2) and other ggplot2 (https://CRAN.R-project.org/package=ggplot2)-based packages. A list of the geometric layers that work seamlessly with geom_facet and facet_plot can be found in Table G.1. As the ggplot2 (https://CRAN.R-project.org/package=ggplot2) community keeps expanding and more geom layers will be implemented in either ggplot2 (https://CRAN.R-project.org/package=ggplot2) or other extensions, geom_facet and facet_plot will gain more power to present data in future. Note that different geom layers can be combined to present data on the same panel and the combinations of different geom layers create the possibility to present more complex data with phylogeny.

```
library(ggtree)
remote_folder <-
         paste0("https://raw.githubusercontent.com/katholt/",

         "plotTree/master/tree_example_april2015/")

## read the phylogenetic tree
tree <- read.tree(paste0(remote_folder, "tree.nwk"))

## read the sampling information data set
info <- read.csv(paste0(remote_folder,"info.csv"))

## read and process the allele table
snps<-read.csv(paste0(remote_folder, "alleles.csv"), header
         = F,
                row.names = 1, stringsAsFactor = F)
snps_strainCols <- snps[1,]
snps<-snps[-1,] # drop strain names
colnames(snps) <- snps_strainCols
```

```
gapChar <- "?"
snp <- t(snps)
lsnp <- apply(snp, 1, function(x) {
        x != snp[1,] & x != gapChar & snp[1,] != gapChar
    })
lsnp <- as.data.frame(lsnp)
lsnp$pos <- as.numeric(rownames(lsnp))
lsnp <- tidyr::gather(lsnp, name, value, -pos)
snp_data <- lsnp[lsnp$value, c("name", "pos")]

## read the trait data
bar_data <- read.csv(paste0(remote_folder, "bar.csv"))

## visualize the tree
p <- ggtree(tree)

## attach the sampling information data set
## and add symbols colored by location
p <- p %<+% info + geom_tippoint(aes(color=location))

## visualize SNP and Trait data using dot and bar charts,
## and align them based on tree structure
p + geom_facet(panel = "SNP", data = snp_data, geom =
        geom_point,
              mapping=aes(x = pos, color = location), shape
        = '|') +
    geom_facet(panel = "Trait", data = bar_data, geom =
        ggstance::geom_barh,
               aes(x = dummy_bar_value, color = location,
        fill = location),
               stat = "identity", width = .6) +
    theme_tree2(legend.position=c(.05, .85))
```

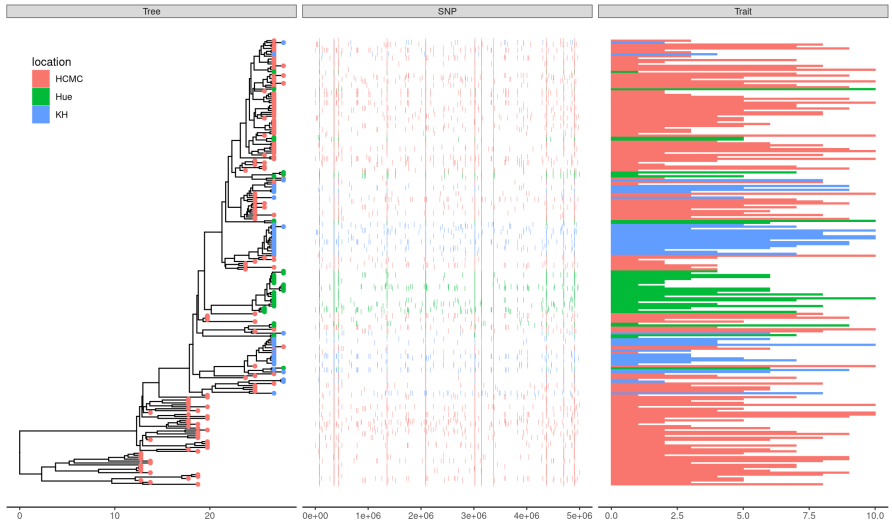


Figure 7.2: **Example of plotting SNP and trait data**.

7.3 Visualize tree with associated matrix

The `gheatmap` function is designed to visualize phylogenetic tree with heatmap of associated matrix (either numerical or categorical). `geom_facet` is a general solution for plotting data with the tree, including heatmap. `gheatmap` is specifically designed for plotting heatmap with tree and provides shortcut for handling column labels and color palette. Another difference is that `geom_facet` only supports rectangular and slanted tree layouts while `gheatmap` supports rectugular, slanted and circular (Figure 7.4) layouts.

In the following example, we visualized a tree of H3 influenza viruses with their associated genotype (Figure 7.3A).

```
beast_file <- system.file("examples/MCC_FluA_H3.tree",
        package="ggtree")
beast_tree <- read.beast(beast_file)
```

```
genotype_file <- system.file("examples/Genotype.txt",
         package="ggtree")
genotype <- read.table(genotype_file, sep="\t",
         stringsAsFactor=F)
colnames(genotype) <- sub("\\.$", "", colnames(genotype))
p <- ggtree(beast_tree, mrsd="2013-01-01") +
    geom_treescale(x=2008, y=1, offset=2) +
    geom_tiplab(size=2)
gheatmap(p, genotype, offset=5, width=0.5, font.size=3,
        colnames_angle=-45, hjust=0) +
    scale_fill_manual(breaks=c("HuH3N2", "pdm", "trig"),
        values=c("steelblue", "firebrick", "darkgreen"),
         name="genotype")
```

The *width* parameter is to control the width of the heatmap. It supports another parameter *offset* for controlling the distance between the tree and the heatmap, for instance to allocate space for tip labels.

For time-scaled tree, as in this example, it's more often to use x axis by using `theme_tree2`. But with this solution, the heatmap is just another layer and will change the x axis. To overcome this issue, we implemented `scale_x_ggtree` to set the x axis more reasonable (Figure 7.3A).

```
p <- ggtree(beast_tree, mrsd="2013-01-01") +
    geom_tiplab(size=2, align=TRUE, linesize=.5) +
    theme_tree2()
gheatmap(p, genotype, offset=8, width=0.6,
        colnames=FALSE, legend_title="genotype") +
    scale_x_ggtree() +
    scale_y_continuous(expand=c(0, 0.3))
```

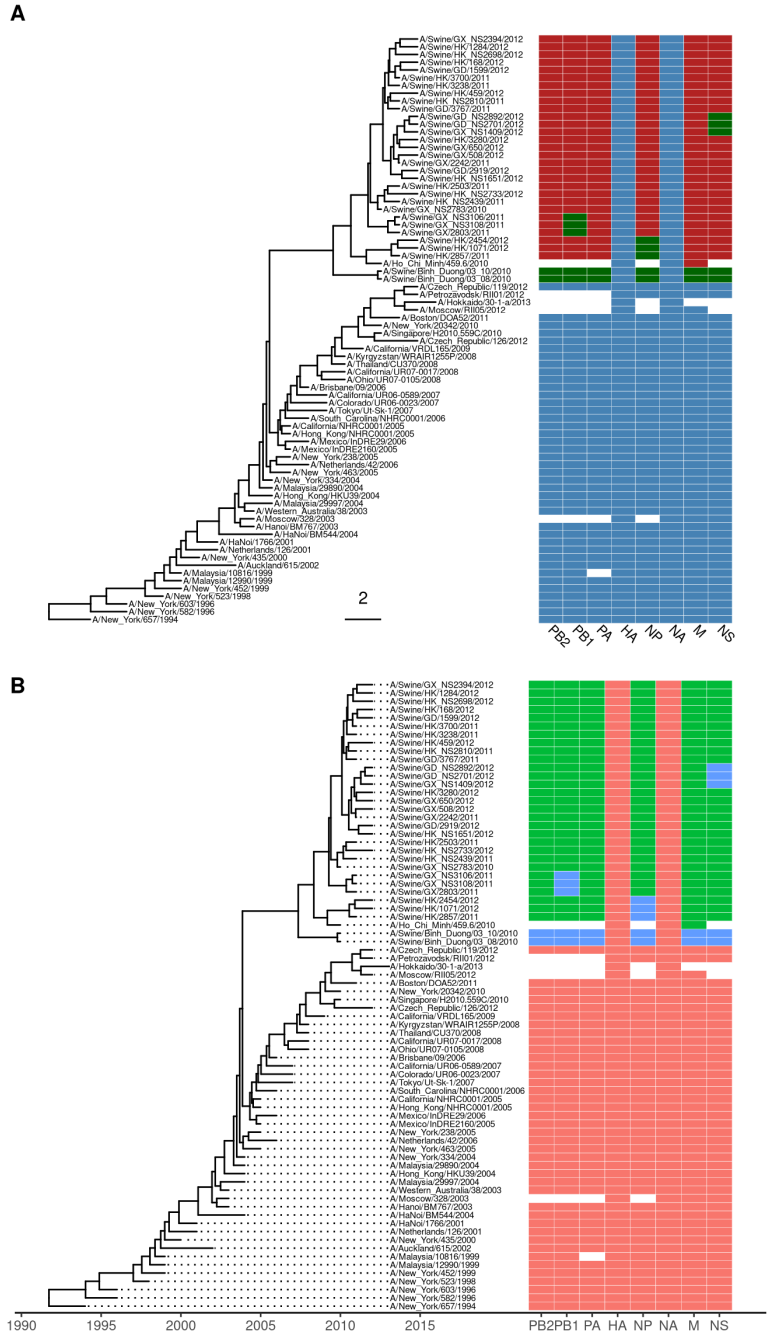

A
A/Swine/GX_NS2394/2012
A/Swine/HK/1284/2012
A/Swine/HK_NS2698/2012
A/Swine/HK/168/2012
A/Swine/GD/1599/2012
A/Swine/HK/3700/2011
A/Swine/HK/3238/2011
A/Swine/HK/459/2012
A/Swine/HK_NS2810/2011
A/Swine/GD/3767/2011
A/Swine/GD_NS2892/2012
A/Swine/GD_NS2701/2012
A/Swine/GX_NS1409/2012
A/Swine/HK/3280/2012
A/Swine/GX/650/2012
A/Swine/GX/508/2012
A/Swine/GX/2242/2011
A/Swine/GD/2919/2012
A/Swine/HK_NS1651/2012
A/Swine/HK/2503/2011
A/Swine/HK_NS2733/2012
A/Swine/HK_NS2439/2011
A/Swine/GX_NS2783/2010
A/Swine/GX_NS3106/2011
A/Swine/GX_NS3108/2011
A/Swine/GX/2803/2011
A/Swine/HK/2454/2012
A/Swine/HK/1071/2012
A/Swine/HK/2857/2011
A/Ho_Chi_Minh/459.6/2010
A/Swine/Binh_Duong/03_10/2010
A/Swine/Binh_Duong/03_08/2010
A/Czech_Republic/119/2012
A/Petrozavodsk/RII01/2012
A/Hokkaido/30-1-a/2013
A/Moscow/RII05/2012
A/Boston/DOA52/2011
A/New_York/20342/2010
A/Singapore/H2010.559C/2010
A/Czech_Republic/126/2012
A/California/VRDL165/2009
A/Kyrgyzstan/WRAIR1255P/2008
A/Thailand/CU370/2008
A/California/UR07-0017/2008
A/Ohio/UR07-0105/2008
A/Brisbane/09/2006
A/California/UR06-0589/2007
A/Colorado/UR06-0023/2007
A/Tokyo/UI-Sk-1/2007
A/South_Carolina/NHRC0001/2006
A/California/NHRC0001/2005
A/Hong_Kong/NHRC0001/2005
A/Mexico/InDRE29/2006
A/Mexico/InDRE2160/2005
A/New_York/238/2005
A/Netherlands/42/2006
A/New_York/463/2005
A/New_York/334/2004
A/Malaysia/29890/2004
A/Hong_Kong/HKU39/2004
A/Malaysia/29997/2004
A/Western_Australia/38/2003
A/Moscow/328/2003
A/Hanoi/BM/767/2003
A/HaNoi/BM544/2004
A/HaNoi/1766/2001
A/Netherlands/126/2001
A/New_York/435/2000
A/Auckland/615/2002
A/Malaysia/10816/1999
A/Malaysia/12990/1999
A/New_York/452/1999
A/New_York/523/1998
A/New_York/603/1996
A/New_York/582/1996
A/New_York/657/1994
2
PB2 PB1 PA HA NP NA M NS
genotype
HuH3N2
pdm
trig
B
1990 1995 2000 2005 2010 2015
PB2 PB1 PA HA NP NA M NS
genotype
HuH3N2
pdm
trig
NA

Figure 7.3: **Example of plotting matrix with** `gheatmap`.

7.3.1 Visualize tree with multiple associated matrix

Of course, we can use multiple `gheatmap` function call to align several associated matrix with the tree, however, ggplot2 (https://CRAN.R-project.org/package=ggplot2) doesn't allow us to use multiple `fill` scales[15].

To solve this issue, we can use ggnewscale (https://github.com/eliocamp/ggnewscale) to create new `fill` scales. Here is an example of using ggnewscale (https://github.com/eliocamp/ggnewscale) with `gheatmap`.

```
nwk <- system.file("extdata", "sample.nwk",
         package="treeio")

tree <- read.tree(nwk)
circ <- ggtree(tree, layout = "circular")

df <- data.frame(first=c("a", "b", "a", "c", "d", "d", "a",
         "b", "e", "e", "f", "c", "f"),
                  second= c("z", "z", "z", "z", "y", "y",
         "y", "y", "x", "x", "x", "a", "a"))
rownames(df) <- tree$tip.label

df2 <- as.data.frame(matrix(rnorm(39), ncol=3))
rownames(df2) <- tree$tip.label
colnames(df2) <- LETTERS[1:3]


p1 <- gheatmap(circ, df, offset=.8, width=.2,
                colnames_angle=95, colnames_offset_y = .25) +
    scale_fill_viridis_d(option="D", name="discrete\nvalue")


library(ggnewscale)
p2 <- p1 + new_scale_fill()
gheatmap(p2, df2, offset=15, width=.3,
          colnames_angle=90, colnames_offset_y = .25) +
    scale_fill_viridis_c(option="A",
         name="continuous\nvalue")
```

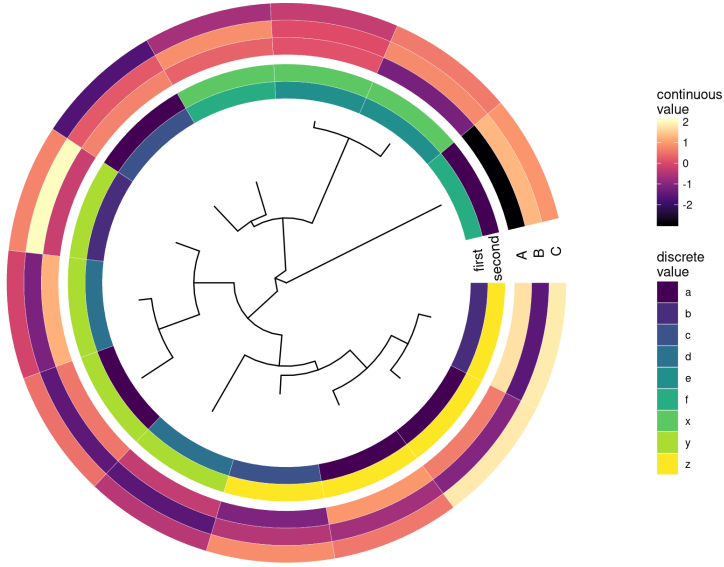


Figure 7.4: **Example of plotting matrix with** `gheatmap`.

7.4 Visualize tree with multiple sequence alignment

The `msaplot` accepts a tree (output of `ggtree`) and a fasta file, then it can visualize the tree with sequence alignment. We can specify the `width` (relative to the tree) of the alignment and adjust relative position by `offset`, that are similar to `gheatmap` function.

```
tree <- read.tree("data/tree.nwk")
p <- ggtree(tree) + geom_tiplab(size=3)
msaplot(p, "data/sequence.fasta", offset=3, width=2)
```

A specific slice of the alignment can also be displayed by specifying `window` parameter.

```
p <- ggtree(tree, layout='circular') +
    geom_tiplab(offset=4, align=TRUE) + xlim(NA, 12)
msaplot(p, "data/sequence.fasta", window=c(120, 200))
```

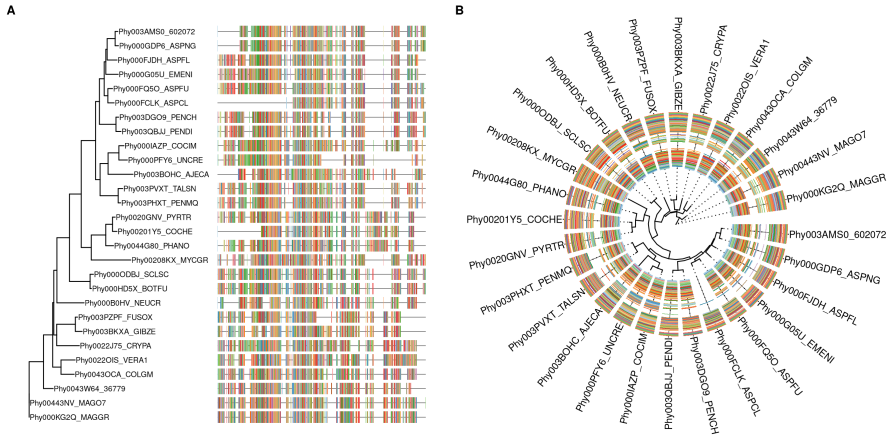


Figure 7.5: **Example of plotting multiple sequence alignment with a tree**.

7.5 The `ggtree` object

7.6 Summary

Although there are many software packages support visualizing phylogenetic tree, plotting tree with data is often missing or with only limited supports. Some of the packages defines S4 classes to store

phylogenetic tree with domain specific data, such as OutbreakTools (https://CRAN.R-project.org/package=OutbreakTools) (Jombart et al. 2014 (page 277)) defined `obkData` for storing tree with epidemiology data and phyloseq (http://bioconductor.org/packages/phyloseq) (McMurdie and Holmes 2013 (page 279)) defines `phyloseq` for storing tree with microbiome data. These packages are capable to present some of the data stored in the object on the tree. However, not all the associated data are supported. For example, species abundance stored in `phyloseq` object is not supported to be visualized using phyloseq (http://bioconductor.org/packages/phyloseq) package. These packages did not provide any utilities to integrate external data for tree visualization. None of these packages support visualizting external data and align the plot to tree based on the tree structure.

ggtree (http://bioconductor.org/packages/ggtree) provides general solutions for integrating data. Method 1, the `%<+%` operator, can integrate external and internal node data and map the data as visual characteristic to visualize the tree and other datasets used in `geom_facet`. Method 2, the `geom_facet` layer, has no restriction of input data as long as there is a `geom` function available to plot the data (*e.g.* species abundance displayed by `geom_density_ridges` as demonstrated in Figure 9.4). Users are free to combine different panels and combine different `geom` layers in the same panel (Figure 11.1). ggtree (http://bioconductor.org/packages/ggtree) has many unique features that cannot be found in other implementations:

1. Integrating node/edge data to the tree can be mapped to visual characteristics of the tree or other datasets (Figure).
2. Capable of parsing expression (math symbols or text formatting), emoji and image files (chapter 8 (page 185)).
3. No predefined of input data types or how the data should be plotted in `geom_facet` (Table G.1).
4. Combining different `geom` functions to visualize associated data is supported (Figure 11.1).
5. Visualizing different datasets on the same panel is supported.
6. Data integrated by `%<+%` can be used in `geom_facet`.
7. Able to add further annotation to specific layers.
8. Modular design by separating tree visualization, data integration (method 1) and graph alignment (method 2).

Modular design is a unique feature for ggtree (http://bioconductor.org/packages/ggtree) to stand out from other packages. The tree can be visualized with data stored in tree object or external data linked by `%<+%` operator, and fully annotated with multiple layers of annotations (Figure 7.1 and 11.1), before passing it to `geom_facet`. `geom_facet` can be called progressively to add multiple panels or multiple layers on the same panels (Figure 11.1). This creates the possiblity of plotting full annotated tree with complex data panels that contains multiple graphic layers.

ggtree (http://bioconductor.org/packages/ggtree) fits the `R` ecosystem and extends the abilities of integrating and presenting data with trees to existing phylogenetic packages. As demonstrated in Figure 9.4, we are able to plot species abundance distributions with `phyloseq` object. This cannot be easily done without ggtree (http://bioconductor.org/packages/ggtree). With ggtree (http://bioconductor.org/packages/ggtree), we are able to attach additional data to tree objects using `%<+%` and align graph to tree using `geom_facet`. Integrating ggtree (http://bioconductor.org/packages/ggtree) to existing workflows will definitely extends the abilities and broadens the applications to present phylogeny-associated data, especially for comparative studies.

8

Annotating Tree with Silhouette Images and Sub-plots

8.1 Annotating Tree with Images

We usually use text to label taxa, *i.e.* displaying taxa names. If the text is image file name (either local or remote), ggtree (http://bioconductor.org/packages/ggtree) can read the image and display the actual image as label of the taxa (Figure 8.1). The `geom_tiplab` and `geom_nodelab` are capable to render silhouette image by internally called ggimage (https://CRAN.R-project.org/package=ggimage) package.

Online tools such as `iTOL` (Letunic and Bork 2007 (page 278)) and `EvolView` (He et al. 2016 (page 277)) support displaying subplots on phylogenetic tree. However only bar and pie charts are supported by these tools. Users may want to visualize node-associated data with other visualization methods, such as violin plot (Grubaugh et al. 2017 (page 276)), venn diagram (Lott et al. 2015), sequence logo *etc*, and display them on the tree. In ggtree (http://bioconductor.org/packages/ggtree), all kinds of subplots are supported as we can export all subplots to image files and use them to label corresponding nodes on the tree.

```
library(ggimage)
library(ggtree)

nwk <- "((((bufonidae, dendrobatidae), ceratophryidae),
         (centrolenidae, leptodactylidae)), hylidae);"

x = read.tree(text = nwk)
ggtree(x) + xlim(NA, 7) + ylim(NA, 6.2) +
    geom_tiplab(aes(image=paste0("img/frogs/", label,
         '.jpg')),
```

```
                geom="image", offset=2, align=2, size=.2)  +
    geom_tiplab(geom='label', offset=1, hjust=.5) +
    geom_image(x=.8, y=5.5, image="img/frogs/frog.jpg",
         size=.2)
```

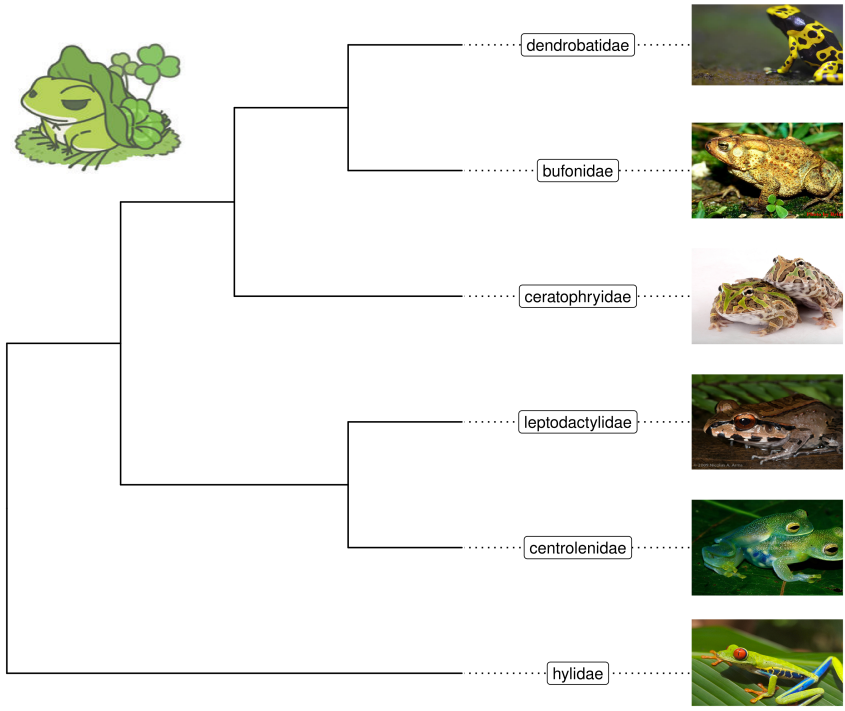


Figure 8.1: **Labelling taxa with images.**

8.2 Annotating Tree with Phylopic

Phylopic contains more than 3200 silhouettes and covers almost all life forms. **ggtree** also supports using phylopic[16] to annotate the tree. Most importantly, we can color the image using numerical/categorical variable.

```
tree <- ape::read.nexus("data/tree.nex")
phylopic_info <- data.frame(node = c(124, 113, 110, 96, 89,
        70),
                            phylopic = c("c4ac9ea6-b465-
        4ed0-a5de-9fe4eebd3ef9",
                                         "5388a472-994a-48e1-
        86de-d988c6019e72",
                                         "615c82cf-fabe-49f8-
        b571-e8117edfa212",
                                         "63ff1b70-6b6e-4412-
        b4d2-b2e2fb51aa95",
                                         "d65d5abd-df8b-4509-
        912c-e42f6c34861f",
                                         "6d54acc7-9e1e-4b79-
        be88-4904cf58fbb5"),
                            species = c("galagoids",
        "lemurs", "tarsiers",
                                         "cebids",
        "hominoids", "cercopithecoids"))
pg <- ggtree(tree)
pg <- pg %<+% phylopic_info +
  geom_nodelab(aes(image=phylopic, color=species,
        subset=!is.na(species)),
             geom="phylopic", alpha = .6) +
  theme(legend.position=c(.2, .8))
```

ggimage (https://CRAN.R-project.org/package=ggimage) package supports querying phylopic uid from scientific name, which is very handy for annotating tree with phylopic.

```
require(ggtree)
newick = "((Pongo_abelii,(Gorilla_gorilla_gorilla,
        (Pan_paniscus,Pan_troglodytes)Pan,Homo_sapiens)Homininae)Homini


tree = read.tree(text=newick)

idlist = ggimage::phylopic_uid(tree$tip.label)
ids <- sapply(idlist, function(x) x[1,1])
```

```
d <- data.frame(label = tree$tip.label, uid = ids)

p <- ggtree(tree) %<+% d +
  geom_tiplab(aes(image=uid, color=label), geom="phylopic",
        offset=2.5) +
  geom_tiplab(aes(label=label), offset = .2) + xlim(NA, 7) +
  theme(legend.position='none')
```

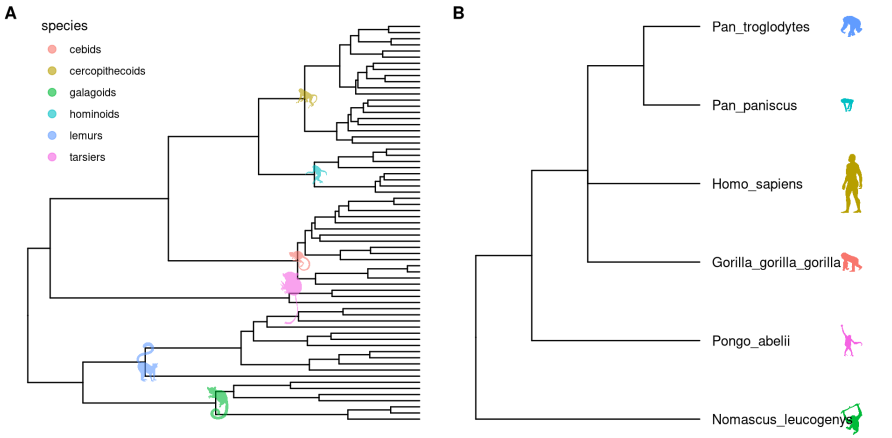


Figure 8.2: **Labelling taxa with phylopic images.** ggtree will automatically download phylopic figures by querying provided UID (A). The UID can be searched using scientific names of the taxa (B). The figures can be colored using on numerical or categorical values.

8.3 Annotating Tree with Sub-plots

ggtree provides a layer, geom_inset, for adding subplots to a phylogenetic tree. The input is a named list of ggplot graphic objects (can be any kind of charts). These objects should named by node numbers. Users can also use ggplotify (https://CRAN.R-project.org/package=ggplotify) to convert plots that generated by other functions (even implemented by base graphics) to ggplot objects, which can then be used in geom_inset. To facilitate adding

bar and pie charts (e.g. summarized stats of results from ancestral reconstruction) to phylogenetic tree, ggtree (http://bioconductor.org/packages/ggtree) provides `nodepie` and `nodebar` functions to create a list of pie or bar charts.

8.3.1 Annotate with bar charts

```
library(phytools)
data(anoletree)
x <- getStates(anoletree,"tips")
tree <- anoletree

cols <- setNames(palette()
        [1:length(unique(x))],sort(unique(x)))
fitER <- ape::ace(x,tree,model="ER",type="discrete")
ancstats <- as.data.frame(fitER$lik.anc)
ancstats$node <- 1:tree$Nnode+Ntip(tree)

## cols parameter indicate which columns store stats
bars <- nodebar(ancstats, cols=1:6)
bars <- lapply(bars, function(g) g+scale_fill_manual(values
        = cols))

tree2 <- full_join(tree, data.frame(label = names(x), stat =
        x ), by = 'label')
p <- ggtree(tree2) + geom_tiplab() +
    geom_tippoint(aes(color = stat)) +
    scale_color_manual(values = cols) +
    theme(legend.position = "right") +
    xlim(NA, 8)
p + geom_inset(bars, width = .08, height = .05, x =
        "branch")
```

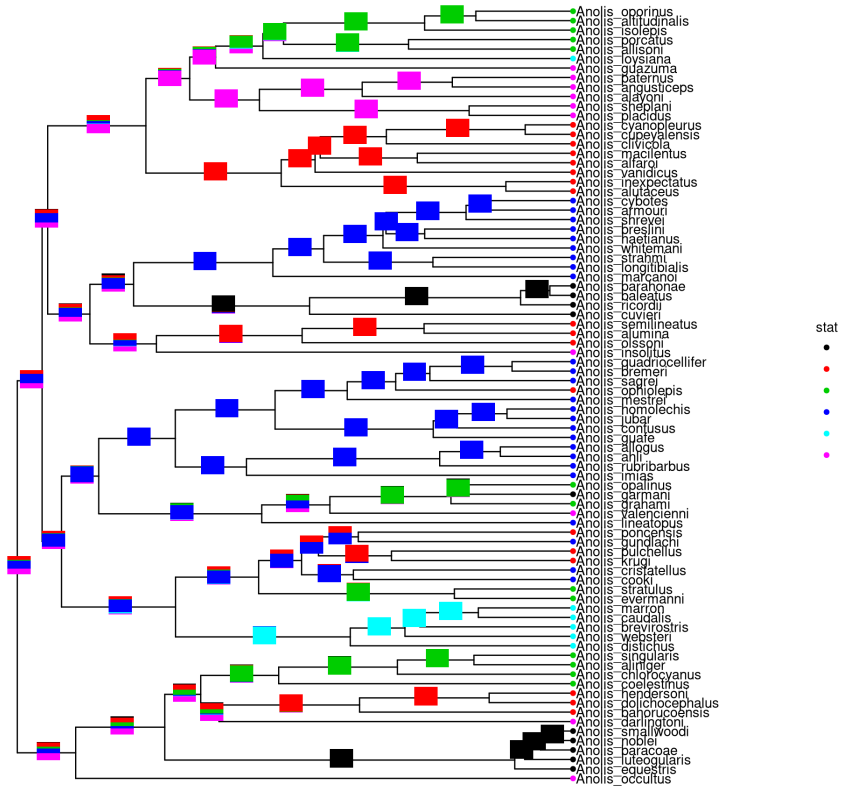


Figure 8.3: **Annotate internal nodes with barplots.**

The *x* position can be one of 'node' or 'branch' and can be adjusted by the parameter *hjust* and *vjust* for horizontal and vertical adjustment respecitvely.

8.3.2 Annotate with pie charts

Similarly, users can use `nodepie` function to generate a list of pie charts and place these charts to annotate corresponding nodes. Both `nodebar` and `nodepie` accepts parameter *alpha* to allow transparency.

```
pies <- nodepie(ancstats, cols = 1:6)
pies <- lapply(pies, function(g) g+scale_fill_manual(values
        = cols))
p + geom_inset(pies, width = .1, height = .1)
```

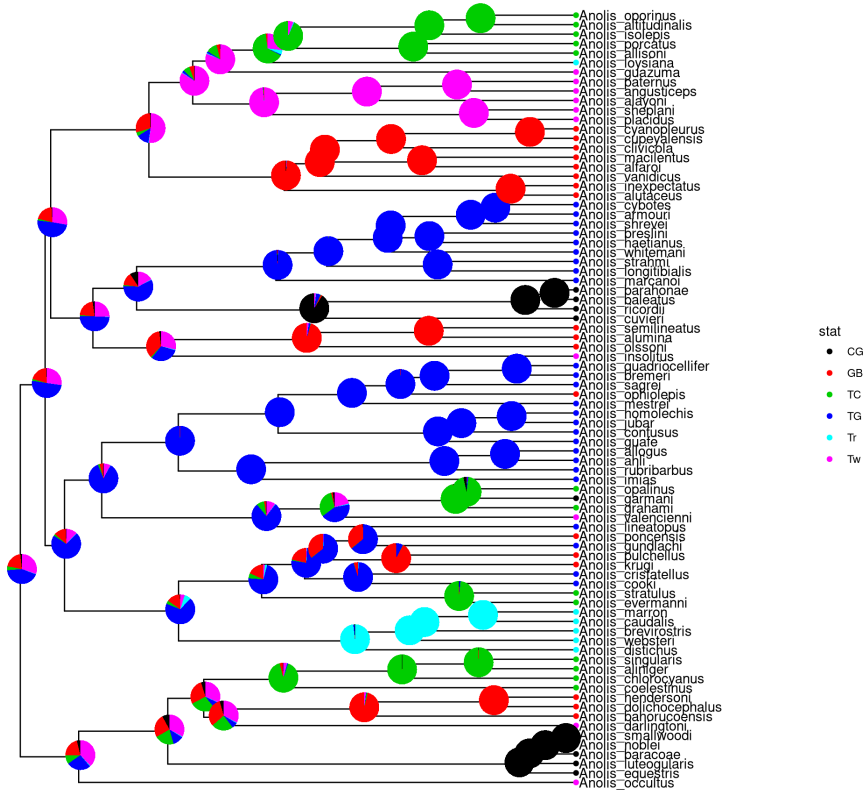


Figure 8.4: **Annotate internal nodes with pie plots.**

8.3.3 Annotate with other types of charts

The `inset` function accepts a list of ggplot graphic objects and these input objects are not restricted to pie or bar charts. They can be any kinds of

charts and hybrid of these charts.

```
pies_and_bars <- pies
i <- sample(length(pies), 20)
pies_and_bars[i] <- bars[i]
p + geom_inset(pies_and_bars, width=.08, height=.05)
```

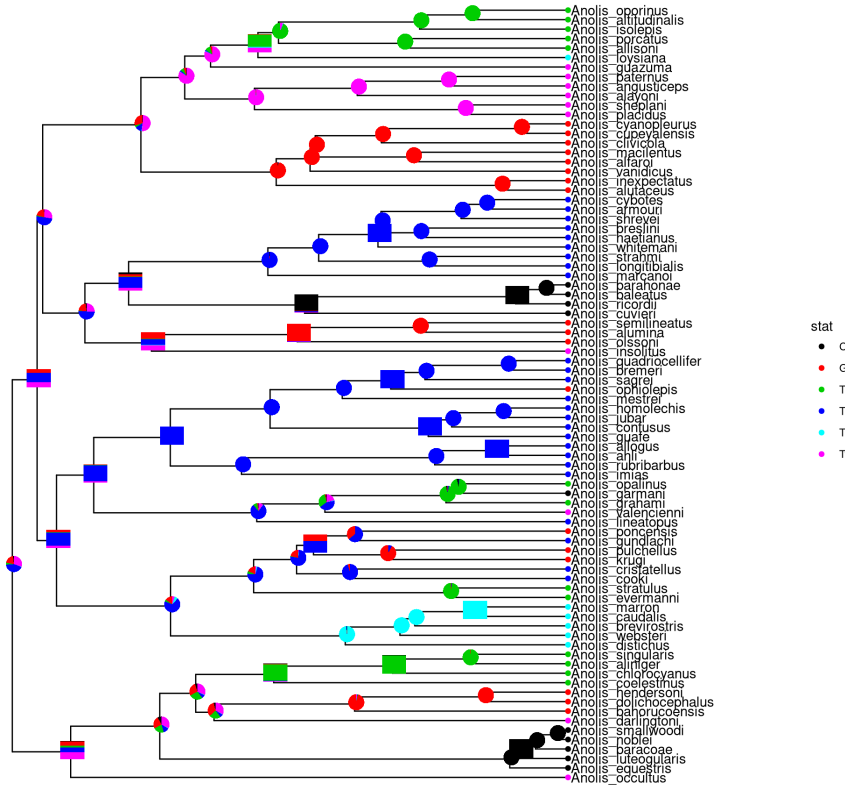
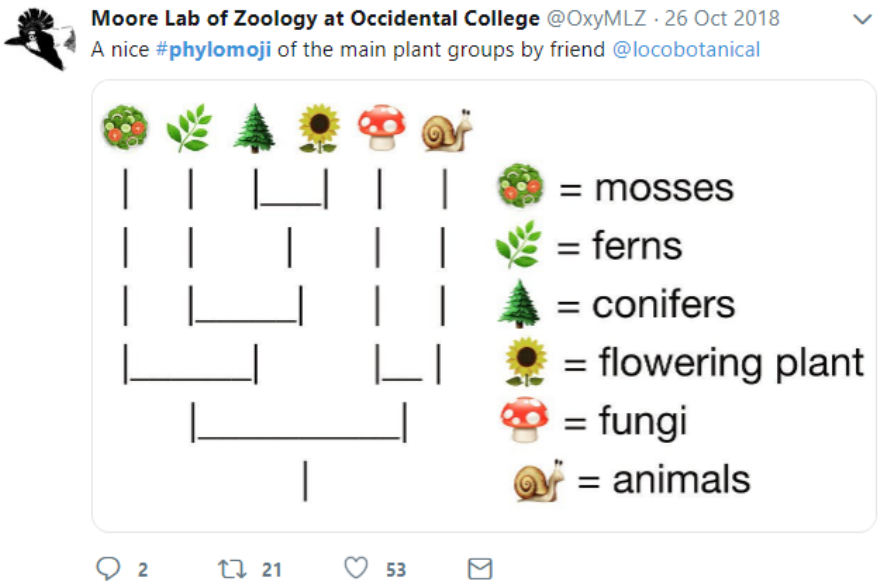


Figure 8.5: **Annotate internal nodes with different types of subplots (bar and pie).**

8.4 Have Fun with Phylomoji

#phylomoji (https://twitter.com/hashtag/phylomoji?src=hash) is fun and it is internally supported by **ggtree** since 2015[17].

Here, we will use **ggtree** to recreate the figure:



(https://twitter.com/OxyMLZ/status/1055586178651451392)

```
library(ggplot2)
library(ggtree)

tt = '((snail,mushroom),
        (((sunflower,evergreen_tree),leaves),green_salad));'
tree = read.tree(text = tt)
d <- data.frame(label = c('snail','mushroom', 'sunflower',
                          'evergreen_tree','leaves',
        'green_salad'),
               group = c('animal', 'fungi', 'flowering
        plant',
```

```
                        'conifers', 'ferns', 'mosses'))

ggtree(tree, linetype = "dashed", size=1, color='firebrick')
        %<+% d +
  xlim(0, 4.5) + ylim(0.5, 6.5) +
  geom_tiplab(parse="emoji", size=15, vjust=.25) +
  geom_tiplab(aes(label = group), geom="label", x=3.5,
        hjust=.5)
```

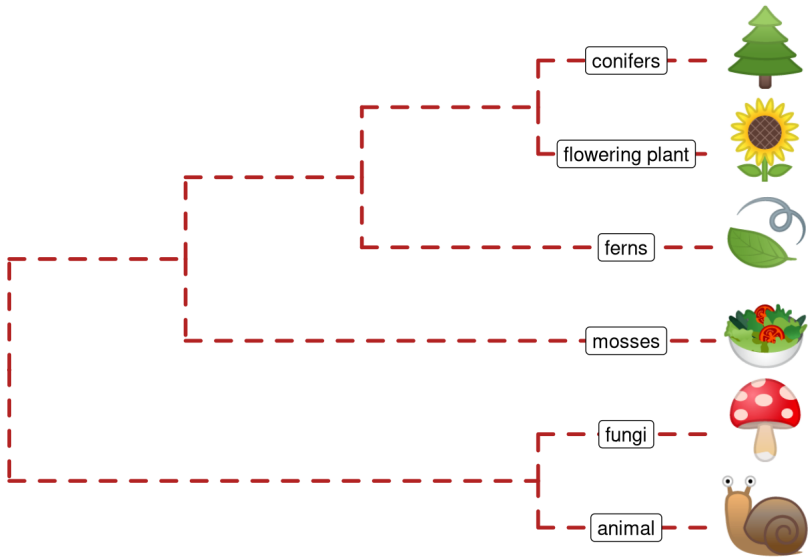


Figure 8.6: **Parsing label as emoji.** Text (*e.g.* node or tip labels) can be parsed as emoji.

Note that the output may depends on what emoji fonts installed in your system[18].

With ggtree (http://bioconductor.org/packages/ggtree), it is easy to generate phylomoji. The emoji is treated as **text**, like abc. We can use emojis to label taxa, clade, color and ratate emoji with any given color and angle. This functionality is internally supported by the emojifont (https://CRAN.R-project.org/package=emojifont) package.

8.4.1 Emoji in cirular/fan layout tree

```
p <- ggtree(tree, layout = "circular", size=1) +
  geom_tiplab(parse="emoji", size=10, vjust=.25)
print(p)

## fan layout
p2 <- open_tree(p, angle=200)
print(p2)

p2 %>% rotate_tree(-90)
```

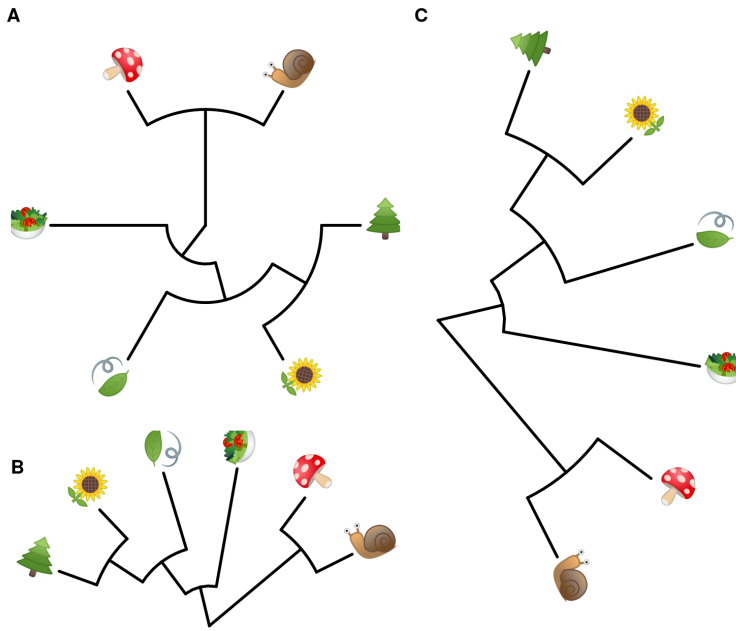


Figure 8.7: **Phylomoji in cirular and fan layouts.**

8.4.2 Emoji to label clades

```
set.seed(123)
tr <- rtree(30)

ggtree(tr) + xlim(NA, 5.2) +
    geom_cladelabel(node=41, label="chicken", parse="emoji",
                    fontsize=12, align=TRUE,
        color="firebrick") +
    geom_cladelabel(node=53, label="duck", parse="emoji",
                    fontsize=12, align=TRUE,
        color="steelblue") +
    geom_cladelabel(node=48, label="family", parse="emoji",
                    fontsize=12, align=TRUE,
        color="darkkhaki")
```

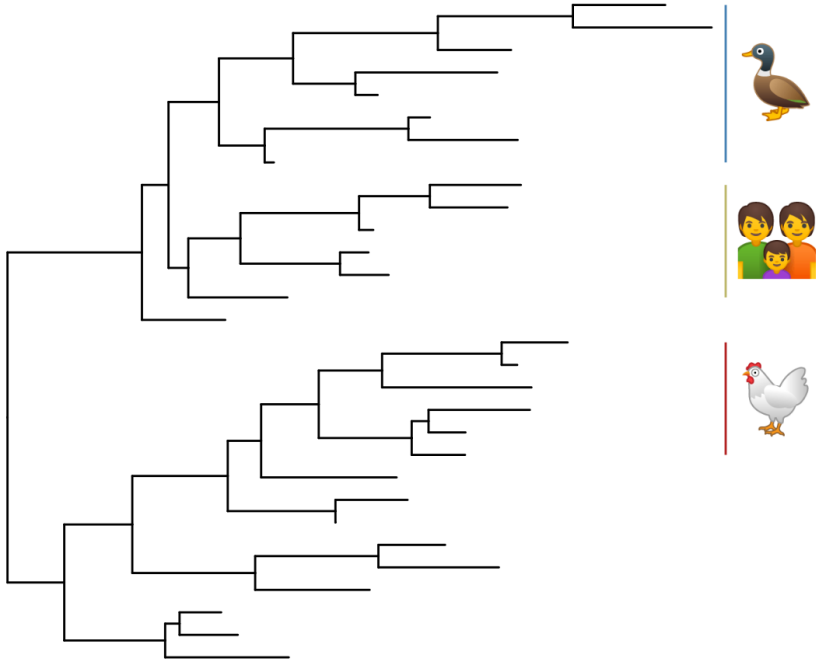


Figure 8.8: **Emoji to label clades.**

8.4.3 Apple Color Emoji

Although R's graphical devices don't support `AppleColorEmoji` font, it's still possible to use it. We can export the `plot` to `svg` file and render it in `Safari`.

```
library(ggtree)
tree_text <- "(((((cow, (whale, dolphin)), (pig2, boar)),
        camel), fish), seedling);"
x <- read.tree(text=tree_text)
library(ggimage)
library(gridSVG)
p <-  ggtree(x, size=2) + geom_tiplab(size=20,
        parse='emoji') +
    xlim(NA, 7) + ylim(NA, 8.5) +
```

```
    geom_phylopic(image="79ad5f09-cf21-4c89-8e7d-
        0c82a00ce728",
                  color="firebrick", alpha = .3, size=Inf)

p
ps = grid.export("emoji.svg", addClass=T)
```

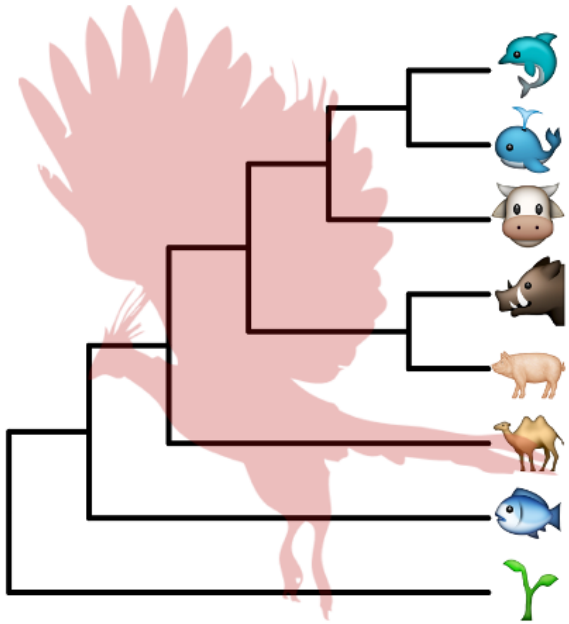


Figure 8.9: **Use Apple Color Emoji in ggtree.**

8.4.4 Phylomoji in ASCII art

This is also possible, please refer to Appendix D (page 261) for details.

9

ggtree for other tree-like objects

9.1 ggtree for phylogenetic tree objects

The treeio (http://bioconductor.org/packages/treeio) packages allows parsing evolutionary inferences from a number of software outputs and linking external data to the tree structure. It serves as an infrastructure to bring evolutionary data to the R community. The ggtree (http://bioconductor.org/packages/ggtree) package works seamlessly with treeio (http://bioconductor.org/packages/treeio) to visualize tree associated data to annotate the tree. The ggtree (http://bioconductor.org/packages/ggtree) package is general tool for tree visualization and annotation and it fits the ecosystem of R packages. Most of the S3/S4 tree objects defined by other R packages are also supported by ggtree (http://bioconductor.org/packages/ggtree), including `phylo`(session 4.2 (page 109)), `multiPhylo` (session 4.4 (page 134)), `phylo4`, `phylo4d`, `phyloseq` and `obkData`. With ggtree (http://bioconductor.org/packages/ggtree), we are able to generate more complex tree graph which is not possible or easy to do with other packages. For example, the visualization of the `phyloseq` object in Figure 9.4 is not supported by the phyloseq (http://bioconductor.org/packages/phyloseq) package. The ggtree (http://bioconductor.org/packages/ggtree) package also extend the possibility of linking external data to these tree object (Yu et al. 2018 (page 282)).

9.1.1 The phylo4 and phylo4d objects

The `phylo4` and `phylo4d` are defined in the phylobase (https://CRAN.R-project.org/package=phylobase) package. The `phylo4` object is a S4 version of `phylo`, while `phylo4d` extends `phylo4` with a data frame that contains trait data. The phylobase (https://CRAN.R-project.org/package=phylobase) package provides `plot` method, which internally calls the `treePlot` function, to

display the tree with the data. However there are some restrictions of the `plot` method, it can only plot numeric values for tree-associated data as bubbles and cannot generate figure legend. `Phylobase` doesn't implement visualization method to display categorical values. Using associated data as visual characteristics such as color, size and shape, is also not supported. Although it is possible to color the tree using associated data, it requires users to extract the data and map them to color vector manually follow by passing the color vector to the `plot` method. This is tedious and error-prone since the order of the color vector needs to be consistent with the edge list stored in the object.

The ggtree (http://bioconductor.org/packages/ggtree) package supports `phylo4d` object and all the associated data stored in the `phylo4d` object can be used directly to annotate the tree (Fig.).

```
library(phylobase)
data(geospiza_raw)
g1 <- as(geospiza_raw$tree, "phylo4")
g2 <- phylo4d(g1, geospiza_raw$data, missing.data="warn")

d1 <- data.frame(x = seq(0.93, 1.15, length.out = 5),
                 lab = names(geospiza_raw$data))

ggtree(g2) + geom_tippoint(aes(size = wingL), x = d1$x[1],
          shape = 1) +
    geom_tippoint(aes(size = tarsusL), x = d1$x[2], shape =
          1) +
    geom_tippoint(aes(size = culmenL), x = d1$x[3], shape =
          1) +
    geom_tippoint(aes(size = beakD),   x = d1$x[4], shape =
          1) +
    geom_tippoint(aes(size = gonysW),  x = d1$x[5], shape =
          1) +
    scale_size_continuous(range = c(3,12), name="") +
    geom_text(aes(x = x, y = 0, label = lab), data = d1,
          angle = 90) +
    geom_tiplab(offset = .3) + xlim(0, 1.3) +
    theme(legend.position = c(.1, .75))
```

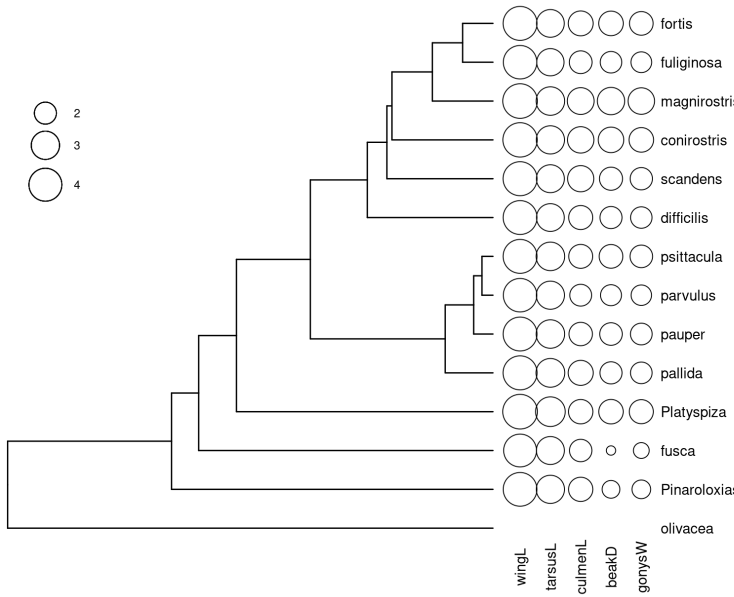


Figure 9.1: **Visualizing phylo4d data using ggtree.**

9.1.2 The phylog object

The `phylog` is defined in ade4 (https://CRAN.R-project.org/package=ade4) package. The package is designed for analyzing ecological data and provides `newick2phylog`, `hclust2phylog` and `taxo2phylog` functions to create phylogeny from Newick string, hierachical clustering result or a taxonomy. The `phylog` object is also supported by `ggtree` as demonstrated in Figure 9.2.

```
library(ade4)
data(taxo.eg)
tax <- as.taxo(taxo.eg[[1]])
print(tax)
```

```
##       genre famille ordre
## esp3     g1    fam1  ORD1
## esp1     g1    fam1  ORD1
```

```
## esp2     g1    fam1  ORD1
## esp4     g1    fam1  ORD1
## esp5     g1    fam1  ORD1
## esp6     g1    fam1  ORD1
## esp7     g1    fam1  ORD1
## esp8     g2    fam2  ORD2
## esp9     g3    fam2  ORD2
## esp10    g4    fam3  ORD2
## esp11    g5    fam3  ORD2
## esp12    g6    fam4  ORD2
## esp13    g7    fam4  ORD2
## esp14    g8    fam5  ORD2
## esp15    g8    fam5  ORD2
```

```
tax.phy <- taxo2phylog(as.taxo(taxo.eg[[1]]))
print(tax.phy)
```

```
## Phylogenetic tree with 15 leaves and 16 nodes
## $class: phylog
## $call: taxo2phylog(taxo = as.taxo(taxo.eg[[1]]))
## $tre: ((((esp3,esp1,esp2,esp4,e...15)l1g8)l2fam5)l3ORD2)Root;
##
##         class   length
## $leaves numeric 15
## $nodes  numeric 16
## $parts  list    16
## $paths  list    31
## $droot  numeric 31
##         content
## $leaves length of the first preceeding adjacent edge
## $nodes  length of the first preceeding adjacent edge
## $parts  subsets of descendant nodes
## $paths  path from root to node or leave
## $droot  distance to root
```

```
ggtree(tax.phy) + geom_tiplab() + geom_nodelab(geom='label')
```

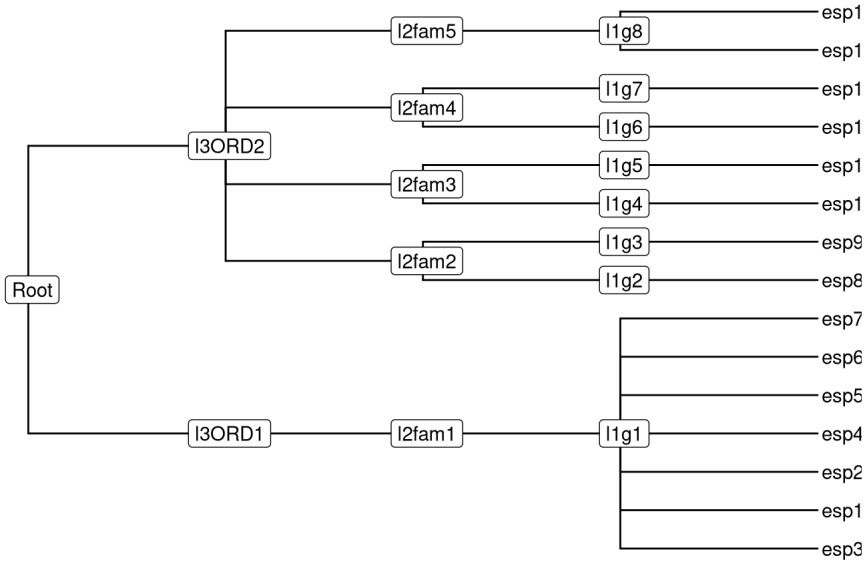


Figure 9.2: **Visualizing phylog tree object.**

9.1.3 The phyloseq object

The `phyloseq` class that defined in the phyloseq (http://bioconductor.org/packages/phyloseq) package was designed for storing microbiome data, including phylogenetic tree, associated sample data and taxonomy assignment. It can import data from popular pipelines, such as *QIIME* (Kuczynski et al. 2011 (page 278)), *mothur* (Schloss et al. 2009 (page 280)), *DADA2* (Callahan et al. 2016 (page 276)) and *PyroTagger* (Kunin and Hugenholtz 2010 (page 278)), *etc.*. The *ggtree* supports visualizing the phylogenetic tree stored in `phyloseq` object and related data can be used to annotate the tree as demonstrated in Figure 9.3 and 9.4.

```
library(phyloseq)
library(scales)

data(GlobalPatterns)
GP <- prune_taxa(taxa_sums(GlobalPatterns) > 0,
        GlobalPatterns)
```

```
GP.chl <- subset_taxa(GP, Phylum=="Chlamydiae")

ggtree(GP.chl) + geom_text2(aes(subset=!isTip, label=label),
        hjust=-.2, size=4) +
  geom_tiplab(aes(label=Genus), hjust=-.3) +
  geom_point(aes(x=x+hjust, color=SampleType, shape=Family,
        size=Abundance),na.rm=TRUE) +
  scale_size_continuous(trans=log_trans(5)) +
  theme(legend.position="right")
```

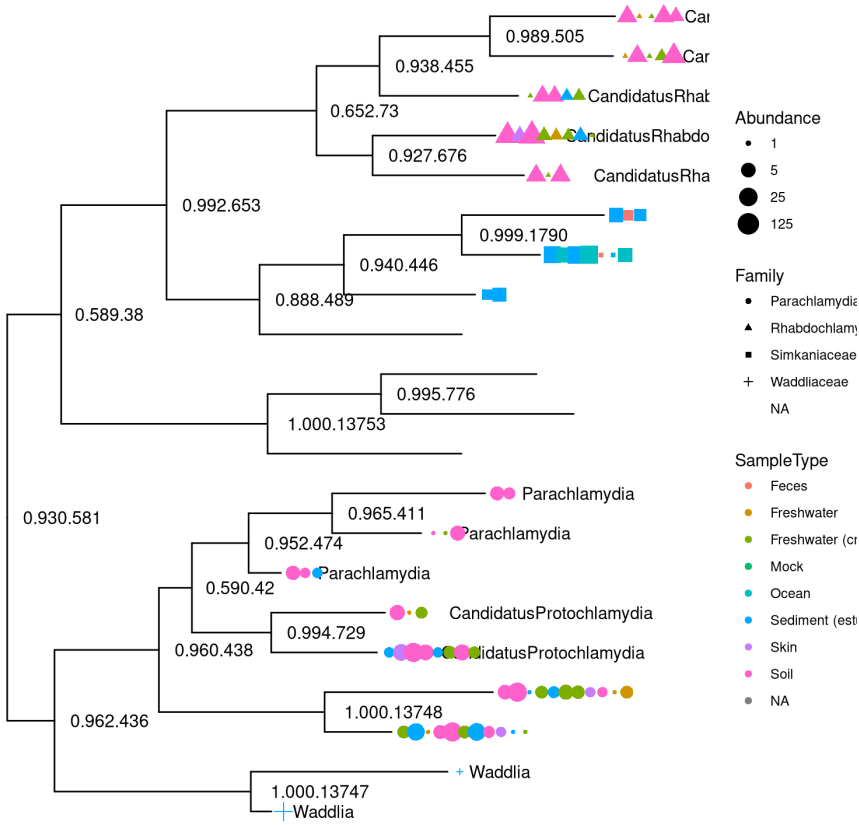


Figure 9.3: **Visualizing phyloseq tree object.**

Figure 9.3 reproduce output of `phyloseq::plot_tree()`. Users of phyloseq (http://bioconductor.org/packages/phyloseq) will find ggtree (http://bioconductor.org/packages/ggtree) useful for visualizing microbiome data and for further annotation, since ggtree (http://bioconductor.org/packages/ggtree) supports high-level of annotation using grammar of graphics and can add tree data layers that are not available in phyloseq (http://bioconductor.org/packages/phyloseq).

```
library(ggridges)

data("GlobalPatterns")
GP <- GlobalPatterns
GP <- prune_taxa(taxa_sums(GP) > 600, GP)
sample_data(GP)$human <- get_variable(GP, "SampleType") %in%
  c("Feces", "Skin")

mergedGP <- merge_samples(GP, "SampleType")
mergedGP <- rarefy_even_depth(mergedGP,rngseed=394582)
mergedGP <- tax_glom(mergedGP,"Order")

melt_simple <- psmelt(mergedGP) %>%
  filter(Abundance < 120) %>%
  select(OTU, val=Abundance)

ggtree(mergedGP) +
  geom_tippoint(aes(color=Phylum), size=1.5) +
  geom_facet(mapping = aes(x=val,group=label,
                           fill=Phylum),
             data = melt_simple,
             geom = geom_density_ridges,
             panel="Abundance",
             color='grey80', lwd=.3)
```

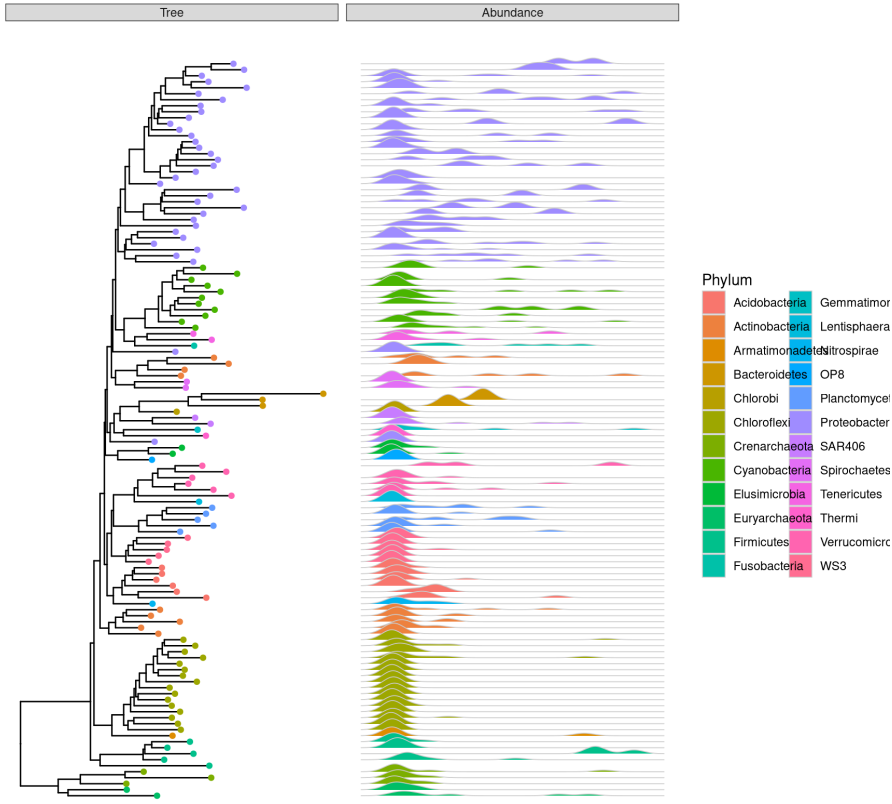


Figure 9.4: **Phylogenetic tree with OTU abundance densities.** Tips were colored by Phylum and corresponding abundance across different samples were visualized as density ridgelines and sorted according to the tree structure.

This example uses microbiome data that provided in `phyloseq` package and density ridgeline is employed to visualize species abundance data. The `geom_facet` layer automatically re-arranges the abundance data according to the tree structure, visualizes the data using the specify `geom` function, *i.e.* `geom_density_ridges`, and aligns the density curves with the tree as demonstrated in Fig. . Note that data stored in the `phyloseq` object is visible to `ggtree` and can be used directly in tree visualization (`Phylum` was used to color tips and density ridgelines in this example). The source

code of this example was firstly published in Supplemental File of (Yu et al. 2018 (page 282)).

9.1.4 The obkData object

The `okbData` is defined in the OutbreakTools (https://CRAN.R-project.org/package=OutbreakTools) package to store incidence-based outbreak data, including meta data of sampling and information of infected individuals such as age and onset of symptoms. The `ggtree` supports the `obkData` object and the information can be used to annotate the tree as shown in Figure 9.5.

```
library(OutbreakTools)
data(FluH1N1pdm2009)
attach(FluH1N1pdm2009)

x <- new("obkData",
         individuals = individuals,
         dna = dna,
         dna.individualID = samples$individualID,
         dna.date = samples$date,
         trees = FluH1N1pdm2009$trees)
ggtree(x, mrsd="2009-09-30", as.Date=TRUE, right=TRUE) +
    geom_tippoint(aes(color=location), size=3, alpha=.75) +
    scale_color_brewer("location", palette="Spectral") +
    theme_tree2(legend.position='right')
```

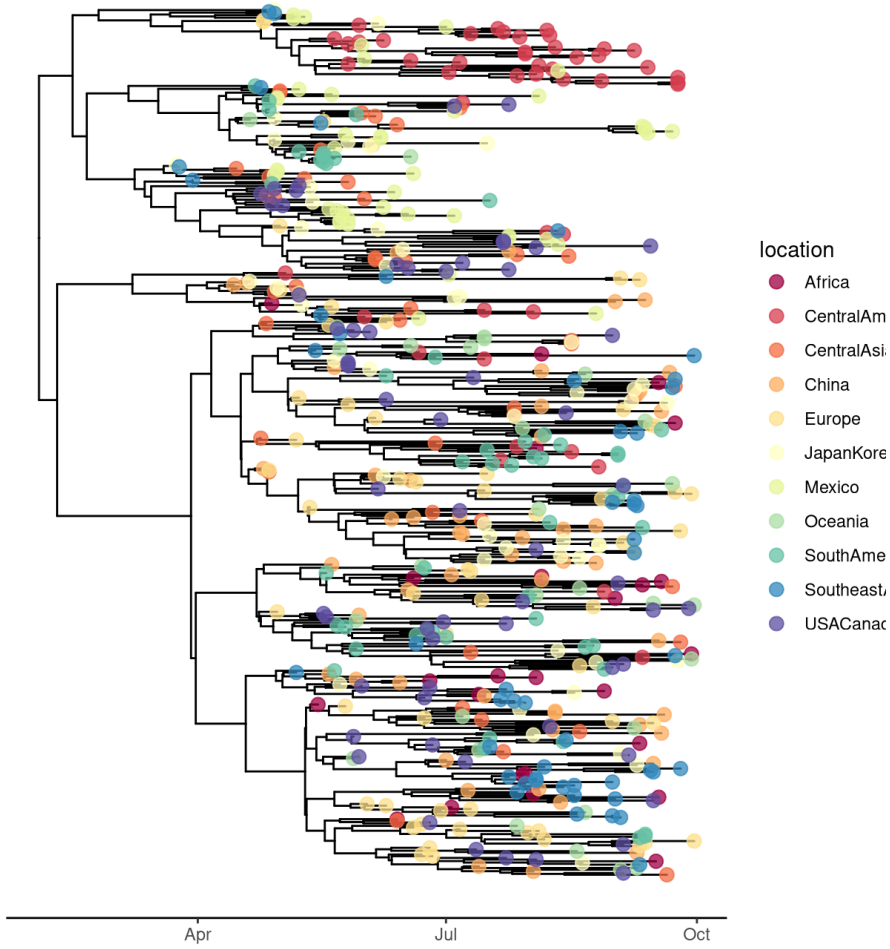


Figure 9.5: **Visualizing obkData tree object.** *x*-axis was scaled by timeline of the outbreak and tips were colored by location of different individuals.

9.2 ggtree for dendrograms

A dendrogram is a tree diagram to display hierachical clustering and classification/regression trees.

We can calculate a hierachical clustering using the function `hclust`

```
hc <- hclust(dist(mtcars))
hc
```

```
##
## Call:
## hclust(d = dist(mtcars))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 32
```

The `hclust` object describes the tree produced by the clustering process. It can be converted to `dendrogram` object, which stores the tree as deeply-nested lists.

```
den <- as.dendrogram(hc)
den
```

```
## 'dendrogram' with 2 branches and 32 members total, at height 425.3447
```

The ggtree (http://bioconductor.org/packages/ggtree) package supports most of the hierarchical clustering objects defined in the R community, including `hclust` and `dendrogram` as well as `agnes`, `diana` and `twins` that defined in the cluster (https://CRAN.R-project.org/package=cluster) package. Users can use `ggtree(object)` to display its tree structure, and user other

layers and utilities to customize the graph and of course add annotation to the tree.

ggtree (http://bioconductor.org/packages/ggtree) provides `layout_dendrogram` to layout the tree top down, and `theme_dendrogram` to display tree height (similar to `theme_tree2` for phylogenetic tree) as demonstrated in Figure 9.6.

```
clus <- cutree(hc, 4)
g <- split(names(clus), clus)

p <- ggtree(hc, linetype='dashed')
clades <- sapply(g, function(n) MRCA(p, n))

p <- groupClade(p, clades, group_name='subtree') +
        aes(color=subtree)

d <- data.frame(label = names(clus),
                  cyl = mtcars[names(clus), "cyl"])

p %<+% d +
  layout_dendrogram() +
  geom_tippoint(size=5, shape=21, aes(fill=factor(cyl),
        x=x+.5), color='black') +
  geom_tiplab(aes(label=cyl), size=3, hjust=.5,
        color='black') +
  geom_tiplab(angle=90, hjust=1, offset=-10, show.legend=F)
        +
  scale_color_brewer(palette='Set1', breaks=1:4) +
  theme_dendrogram(plot.margin=margin(6,6,80,6)) +
  theme(legend.position=c(.9, .6))
```

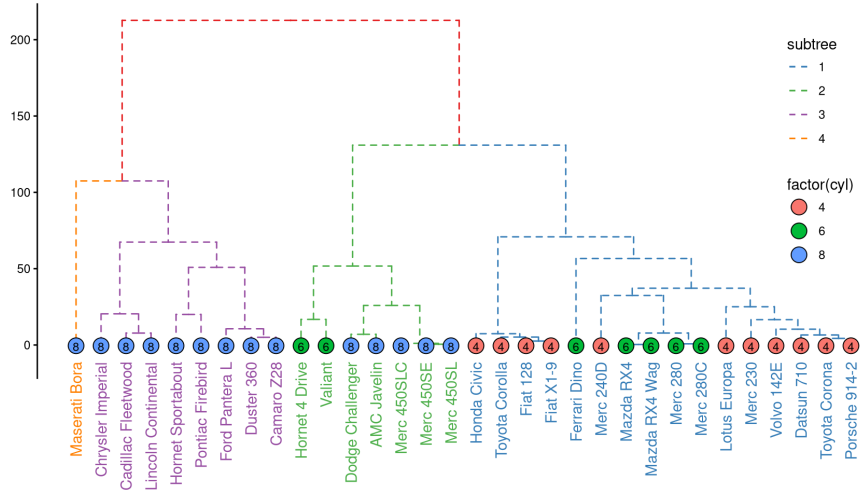


Figure 9.6: **Visualizing dendrogram.**

9.3 ggtree for tree graph

treeio (http://bioconductor.org/packages/treeio) supports converting tree graph (as an `igraph` object) to `phylo` object and ggtree (http://bioconductor.org/packages/ggtree) supports directly visualizing tree graph as demonstrated in Figure 9.7.

```
library(igraph)
g <- graph.tree(40, 3)
arrow_size <- unit(rep(c(0, 3), times = c(27, 13)), "mm")
ggtree(g, layout='slanted', arrow =
         arrow(length=arrow_size)) +
  geom_point(size=5, color='steelblue', alpha=.6) +
  geom_tiplab(hjust=.5,vjust=2) + layout_dendrogram()
```

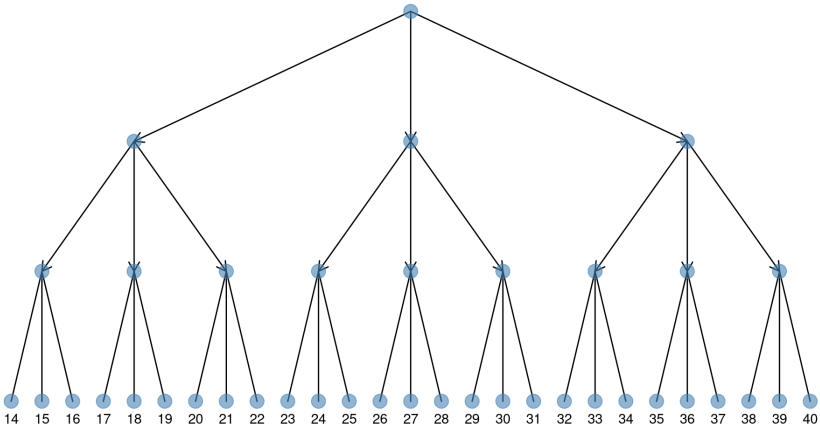


Figure 9.7: **Visualizing tree graph.**

Part III: Miscellaneous topics

10

ggtree utilities

10.1 facet utilities

10.1.1 facet_widths

```
library(ggplot2)
library(ggstance)
library(ggtree)
library(reshape2)

set.seed(123)
tree <- rtree(30)

p <- ggtree(tree, branch.length = "none") +
    geom_tiplab() + theme(legend.position='none')

a <- runif(30, 0,1)
b <- 1 - a
df <- data.frame(tree$tip.label, a, b)
df <- melt(df, id = "tree.tip.label")

p2 <- facet_plot(p + xlim_tree(8), panel = 'bar', data = df,
         geom = geom_barh,
                 mapping = aes(x = value, fill =
         as.factor(variable)),
                 width = 0.8, stat='identity') +
         xlim_tree(9)

facet_widths(p2, widths = c(1, 2))
```

It also supports using name vector to set the widths of specific panels. The following code will display identical figure to Figure 10.1A.

```
facet_widths(p2, c(Tree = .5))
```

The `facet_widths` function also work with other `ggplot` object as demonstrated in Figure 10.1B.

```
p <- ggplot(iris, aes(Sepal.Width, Petal.Length)) +
  geom_point() + facet_grid(.~Species)
facet_widths(p, c(setosa = .5))
```

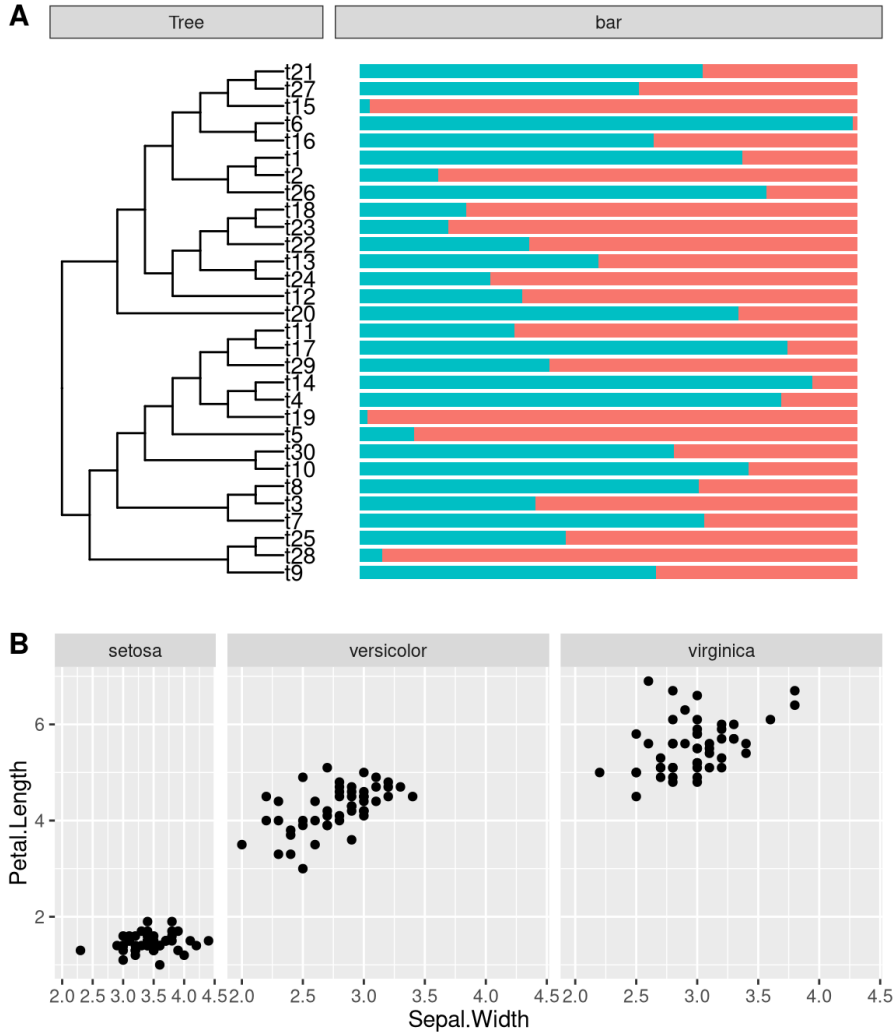


Figure 10.1: **Adjust relative widths of ggplot facets.** The `facet_widths` function works with `ggtree` (A) as well as `ggplot` (B).

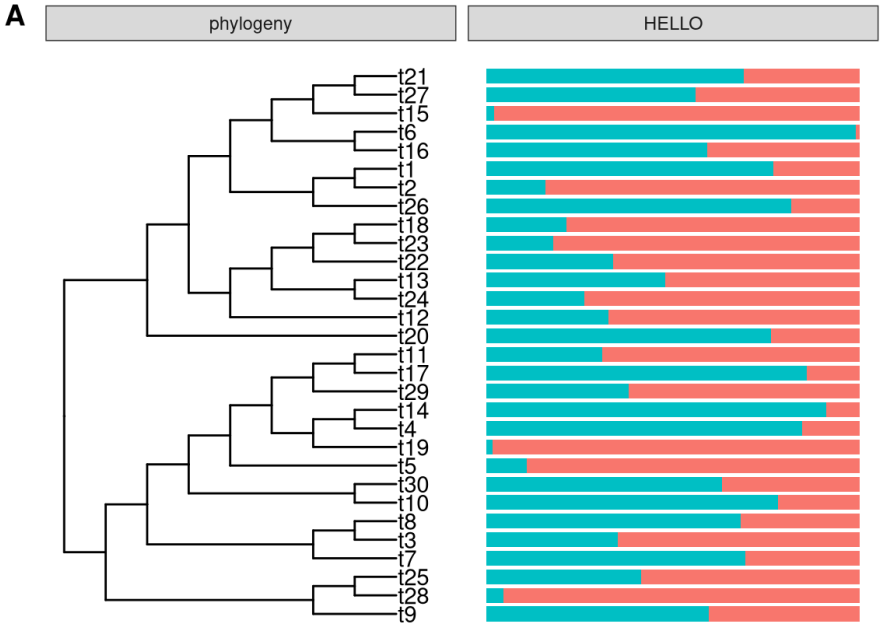
10.1.2 facet_labeller

The `facet_labeller` function was designed to re-label selected panels, and it currently only works with `ggtree` object (*i.e.* `facet_plot` output).

```
facet_labeller(p2, c(Tree = "phylogeny", bar = "HELLO"))
```

If you want to combine `facet_widths` with `facet_labeller`, you need to call `facet_labeller` to re-label the panels before using `facet_widths` to set the relative widths of each panels. Otherwise it wont work since the output of `facet_widths` is re-drawn from `grid` object.

```
facet_labeller(p2, c(Tree = "phylogeny")) %>%
        facet_widths(c(Tree = .4))
```


A
phylogeny
HELLO
t21
t27
t15
t6
t16
t1
t2
t26
t18
t23
t22
t13
t24
t12
t20
t11
t17
t29
t14
t4
t19
t5
t30
t10
t8
t3
t7
t25
t28
t9

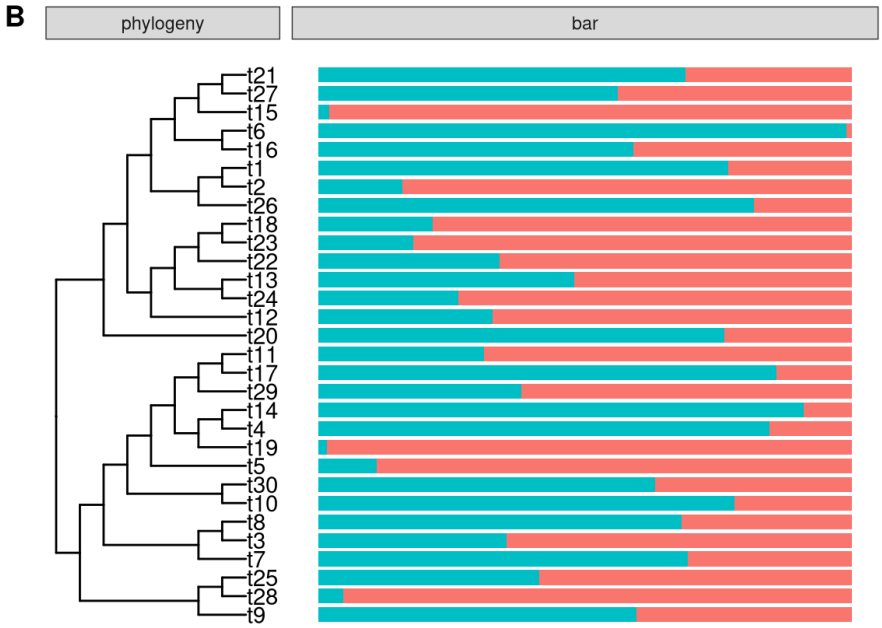

B
phylogeny
bar
t21
t27
t15
t6
t16
t1
t2
t26
t18
t23
t22
t13
t24
t12
t20
t11
t17
t29
t14
t4
t19
t5
t30
t10
t8
t3
t7
t25
t28
t9

Figure 10.2: **Rename facet labels.** Rename multiple labels simultaneously (A) or only for specific one (B) are all supported. `facet_labeller` can combine with `facet_widths` to rename facet label and then adjust relative widths (B).

10.2 layout utilities

In session 4.2.2 (page 112), we introduce several layouts that supported by ggtree (http://bioconductor.org/packages/ggtree). The ggtree (http://bioconductor.org/packages/ggtree) package also provide several layout functions that can transfrom from one to another. Note that not all layouts are supported (see 10.1).

Table 10.1: Layout layers.

Layout	Description
layout_circular	transform rectangular layout to circular layout
layout_dendrogram	transform rectangular layout to dendrogram layout
layout_fan	transform rectangular/circular layout to fan layout
layout_rectangular	transform circular/fan layout to rectangular layout

```
set.seed(2019)
x <- rtree(20)
p <- ggtree(x)
p + layout_dendrogram()
ggtree(x, layout = "circular") + layout_rectangular()
p + layout_circular()
p + layout_fan(angle=90)
```

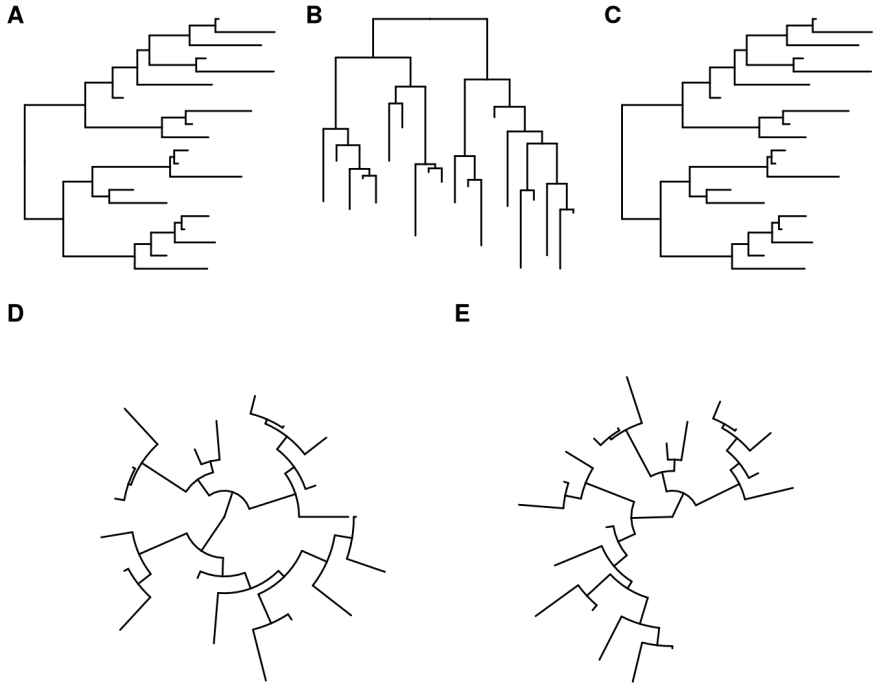


Figure 10.3: **Layout layers for transforming among different layouts**. Default rectangular layout (A); transform rectangular to dendrogram layout (B); transform circular to rectangular layout (C); transform rectangular to circular layout (D); transform rectangular to fan layout (E).

10.3 legend utilities

10.4 scale utilities

10.4.1 xlim_expand

Sometimes we need to set `xlim` for specific panel (*e.g.* allocate more space for long tip labels (page 241) at `Tree` panel). However, the `ggplot2::xlim()` function applies to all the panels. ggtree (http://bioconductor.org/packages/ggtree) provides `xlim_expand()` to adjust `xlim` for user specific panel. It accepts two parameters, `xlim` and `panel`, and can adjust all individual panels as demonstrated in Figure 10.4A. If you only want to adjust `xlim` of the `Tree` panel, you can use `xlim_tree()` as a shortcut.

```
set.seed(2019-05-02)
x <- rtree(30)
p <- ggtree(x) + geom_tiplab()
d <- data.frame(label = x$tip.label,
                value = rnorm(30))
p2 <- facet_plot(p, panel = "Dot", data = d,
            geom = geom_point, mapping = aes(x = value))
p2 + xlim_tree(6) + xlim_expand(c(-10, 10), 'Dot')
```

The `xlim_expand()` function also works with `ggplot2::facet_grid()`. As demonstrating in Figure 10.4B, only the `xlim` of *virginica* panel was adjusted by `xlim_expand()`.

```
g <- ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
    geom_point() + facet_grid(. ~ Species, scales =
        "free_x")
g + xlim_expand(c(0, 15), 'virginica')
```

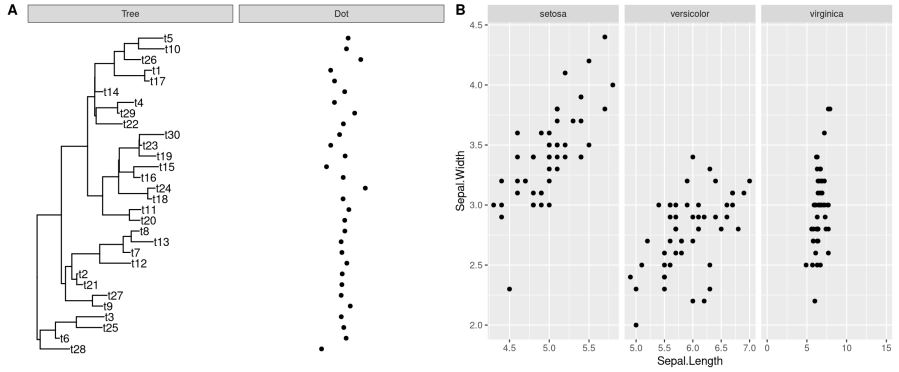


Figure 10.4: **Setting xlim for user specific panel.** xlim for `ggtree::facet_plot` (A, Tree and Dot panels), and `ggplot2::facet_grid` (B, virginica panel).

11

Gallery of reproducible examples

11.1 Visualizing pairwise nucleotide sequence distance with phylogenetic tree

This example reproduces Fig. 1 of (Chen et al. 2017 (page 276)). It extracts accession numbers from tip labels of the HPV58 tree and calculates pairwise nucleotide sequence distances. The distance matrix is visualized as dot and line plots. This example demonstrates the abilities of adding multiple layers to a specific panel. As illustrated on Figure 11.1, the `facet_plot` function displays sequence distances as a dot plot and then adds a layer of line plot to the same panel, *i.e.* Sequence Distance. In addition, the tree in `facet_plot` can be fully annotated with multiple layers (clade labels, bootstrap support values, *etc*). The source code of this example was firstly published in Supplemental File of (Yu et al. 2018 (page 282)).

```
library(tibble)
library(tidyr)
library(Biostrings)
library(treeio)
library(ggplot2)
library(ggtree)

hpvtree <-
        paste0("https://raw.githubusercontent.com/GuangchuangYu/"
```

```
        ,

        "plotting_tree_with_data/master/HPV58.tree")
tree <- read.tree(hpvtree)

clade <- c(A3 = 92, A1 = 94, A2 = 108, B1 = 156, B2 = 159, C
        = 163, D1 = 173, D2 = 176)
tree <- groupClade(tree, clade)
cols <- c(A1 = "#EC762F", A2 = "#CA6629", A3 = "#894418", B1
        = "#0923FA",
         B2 = "#020D87", C = "#000000", D1 = "#9ACD32",D2 =
        "#08630A")

## visualize the tree with tip labels and tree scale
p <- ggtree(tree, aes(color = group), ladderize = FALSE) %>%
        rotate(rootnode(tree)) +
    geom_tiplab(aes(label = paste0("italic('", label,
        "')")), parse = TRUE, size = 2.5) +
    geom_treescale(x = 0, y = 1, width = 0.002) +
    scale_color_manual(values = c(cols, "black"), na.value =
        "black", name = "Lineage",
                       breaks = c("A1", "A2", "A3", "B1",
        "B2", "C", "D1", "D2")) +
    guides(color = guide_legend(override.aes = list(size =
        5, shape = 15))) +
    theme_tree2(legend.position = c(.1, .88))
## Optional
## add labels for monophyletic (A, C and D) and paraphyletic
        (B) groups
p <- p + geom_cladelabel(94, "italic(A1)", color =
        cols[["A1"]], offset = .003, align = TRUE,
                     offset.text = -.001, barsize = 1.2,
        extend = c(0, 0.5), parse = TRUE) +
    geom_cladelabel(108, "italic(A2)", color = cols[["A2"]],
        offset = .003, align = TRUE,
                    offset.text = -.001, barsize = 1.2,
        extend = 0.5, parse = TRUE) +
    geom_cladelabel(131, "italic(A3)", color = cols[["A3"]],
        offset = .003, align = TRUE,
                    offset.text = -.001, barsize = 1.2,
        extend = c(0.5, 0), parse = TRUE) +
```

```
    geom_cladelabel(92, "italic(A)", color = "darkgrey",
        offset = .00315, align = TRUE,
                  offset.text = 0.0002, barsize = 2,
        fontsize = 5, parse = TRUE) +
    geom_cladelabel(156, "italic(B1)", color = cols[["B1"]],
        offset = .003, align = TRUE,
                  offset.text = -.001, barsize = 1.2,
        extend = c(0, 0.5), parse = TRUE) +
    geom_cladelabel(159, "italic(B2)", color = cols[["B2"]],
        offset = .003, align = TRUE,
                  offset.text = -.001, barsize = 1.2,
        extend = c(0.5, 0), parse = TRUE) +
    geom_strip(65, 71, "italic(B)", color = "darkgrey",
        offset = 0.00315, align = TRUE,
                  offset.text = 0.0002, barsize = 2,
        fontsize = 5, parse = TRUE) +
    geom_cladelabel(163, "italic(C)", color = "darkgrey",
        offset = .0031, align = TRUE,
                  offset.text = 0.0002, barsize = 3.2,
        fontsize = 5, parse = TRUE) +
    geom_cladelabel(173, "italic(D1)", color = cols[["D1"]],
        offset = .003, align = TRUE,
                  offset.text = -.001, barsize = 1.2,
        extend = c(0, 0.5), parse = TRUE) +
    geom_cladelabel(176, "italic(D2)", color = cols[["D2"]],
        offset = .003, align = TRUE,
                  offset.text = -.001, barsize = 1.2,
        extend = c(0.5, 0), parse = TRUE) +
    geom_cladelabel(172, "italic(D)", color = "darkgrey",
        offset = .00315, align = TRUE,
                  offset.text = 0.0002, barsize = 2,
        fontsize = 5, parse = TRUE)
## Optional
## display support values
p <- p + geom_nodelab(aes(subset = (node == 92), label =
        "*"),
                   color = "black", nudge_x = -.001,
        nudge_y = 1) +
    geom_nodelab(aes(subset = (node == 155), label = "*"),
                   color = "black", nudge_x = -.0003,
        nudge_y = -1) +
```

```
    geom_nodelab(aes(subset = (node == 158), label =
        "95/92/1.00"),
                     color = "black", nudge_x = -0.0001,
        nudge_y = -1, hjust = 1) +
    geom_nodelab(aes(subset = (node == 162), label =
        "98/97/1.00"),
                     color = "black", nudge_x = -0.0001,
        nudge_y = -1, hjust = 1) +
    geom_nodelab(aes(subset = (node == 172), label = "*"),
                     color = "black", nudge_x = -.0003,
        nudge_y = -1)
```

```
## extract accession numbers from tip labels
tl <- tree$tip.label
acc <- sub("\\w+\\|", "", tl)
names(tl) <- acc

## read sequences from GenBank directly into R
## and convert the object to DNAStringSet
tipseq <- ape::read.GenBank(acc) %>% as.character %>%
    lapply(., paste0, collapse = "") %>% unlist %>%
    DNAStringSet
## align the sequences using muscle
tipseq_aln <- muscle::muscle(tipseq)
tipseq_aln <- DNAStringSet(tipseq_aln)
```

```
## calculate pairwise hamming distances among sequences
tipseq_dist <- stringDist(tipseq_aln, method = "hamming")

## calculate percentage of differences
tipseq_d <- as.matrix(tipseq_dist) / width(tipseq_aln[1]) *
        100

## convert the matrix to tidy data frame for facet_plot
dd <- as_tibble(tipseq_d)
dd$seq1 <- rownames(tipseq_d)
td <- gather(dd,seq2, dist, -seq1)
td$seq1 <- tl[td$seq1]
td$seq2 <- tl[td$seq2]
```

```
g <- p$data$group
names(g) <- p$data$label
td$clade <- g[td$seq2]

## visualize the sequence differences using dot plot and
        line plot
## and align the sequence difference plot to the tree using
        facet_plot
p2 <- facet_plot(p, panel = "Sequence Distance", data = td,
        geom_point,
            mapping = aes(x = dist, color = clade, shape =
        clade), alpha = .6) %>%
    facet_plot(panel = "Sequence Distance", data = td, geom
        = geom_path,
            mapping=aes(x = dist, group = seq2, color =
        clade), alpha = .6) +
    scale_shape_manual(values = 1:8, guide = FALSE)

print(p2)
```

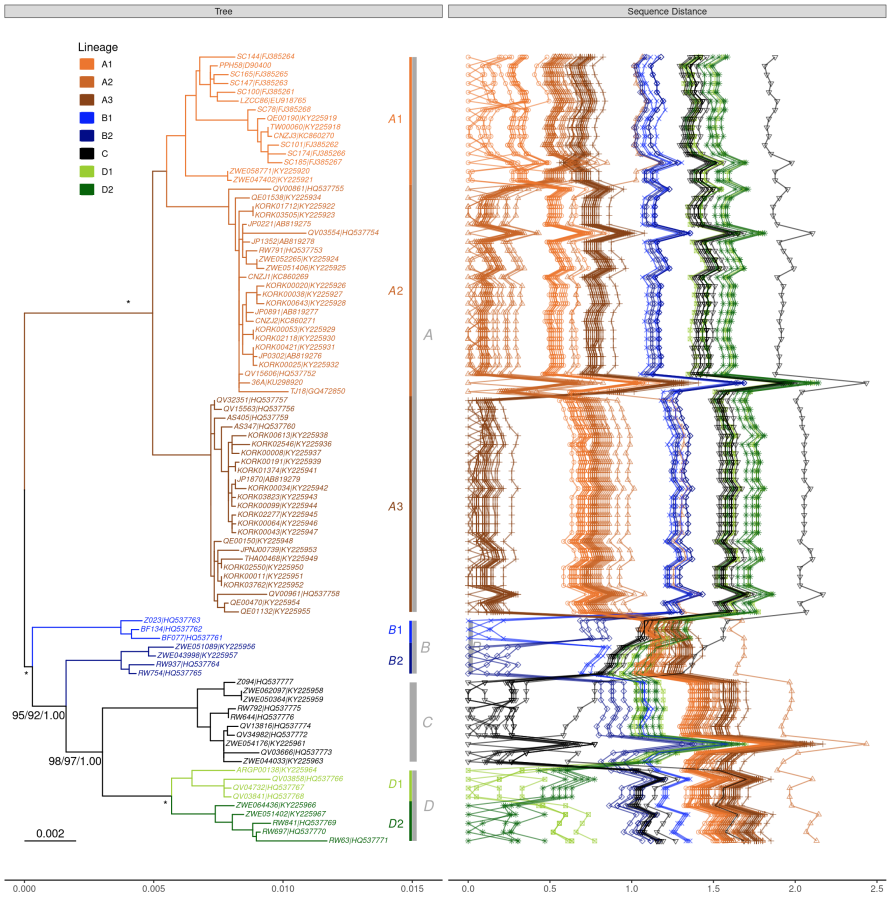


Figure 11.1: **Phylogeny of HPV58 complete genomes with dot and line plots of pairwise nucleotide sequence distances**.

11.2 Displaying different symbolic points for bootstrap values.

We can cut the bootstrap values into several intervals, *e.g.* to indicate whether the clade is high, moderate or low support. Then we can use these intervals as categorical variable to set different color or shape of symbolic points to indicate the bootstrap values belong to which category.

```
library(treeio)
library(ggplot2)
library(ggtree)

tree <-
         read.newick("data/RMI.phy_phyml_tree_rooted_labeled.txt",
          node.label='support')
root <- rootnode(tree)
ggtree(tree, color="black", size=1.5, linetype=1,
         right=TRUE) +
    geom_tiplab(size=4.5, hjust = -0.060, fontface="bold") +
         xlim(0, 0.09) +
    geom_point2(aes(subset=!isTip & node != root,
                    fill=cut(support, c(0, 700, 900,
         1000))),
                    shape=21, size=4) +
    theme_tree(legend.position=c(0.2, 0.2)) +
    scale_fill_manual(values=c("white", "grey", "black"),
         guide='legend',
                    name='Bootstrap Percentage(BP)',
                    breaks=c('(900,1e+03]', '(700,900]',
         '(0,700]'),
                    labels=expression(BP>=90,70 <= BP * " <
         90", BP < 70))
```

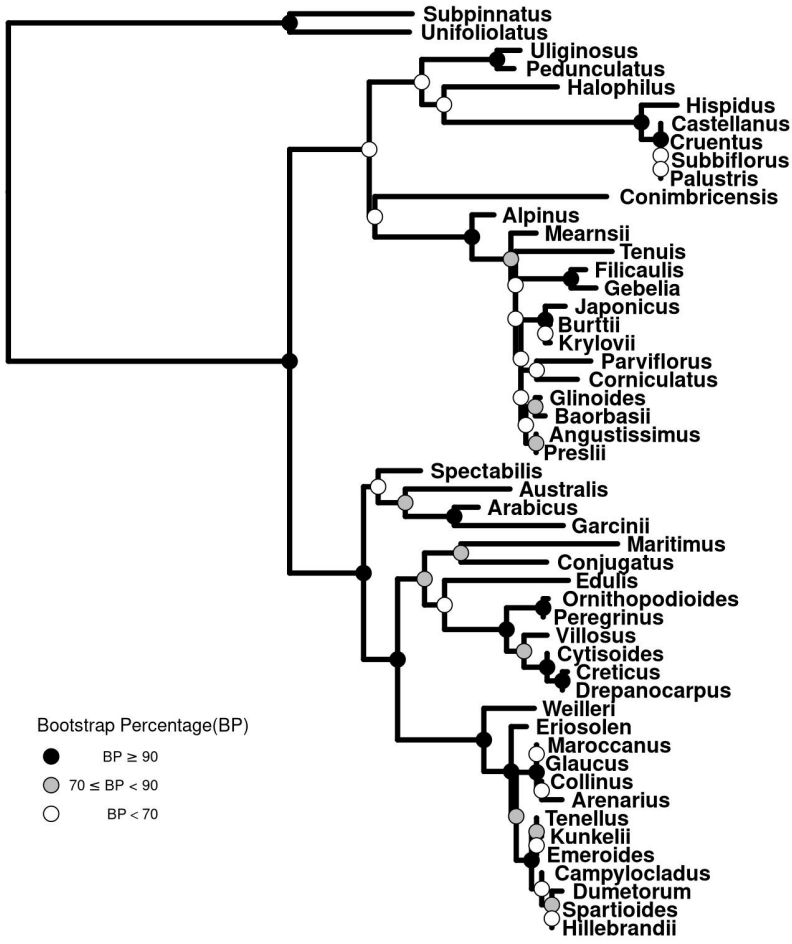


Figure 11.2: **Partitioning bootstrap values**. Bootstrap values were divided into three categories and this information was used to color circle points.

11.3 Phylogenetic tree with genome locus structure

The `geom_motif` is defined in `ggtree` and it is a wrapper layer of `gggenes::geom_gene_arrow`. The `geom_motif` can automatically adjust genomic alignment by selective gene (via the `on` parameter) and can label genes via the `label` parameter. In the following example, we use `example_genes` dataset provided by gggenes (https://CRAN.R-project.org/package=gggenes). As the dataset only provide genomic coordinations of a set of genes, a phylogeny for the genomes need to be firstly constructed. We calculate jaccard similarity based on the ratio of overlapping genes among genomes and correspondingly determine genome distance. BioNJ algorithm was applied to construct the tree (Figure 11.3).

```
library(dplyr)
library(ggplot2)
library(gggenes)
library(ggtree)

get_genes <- function(data, genome) {
    filter(data, molecule == genome) %>% pull(gene)
}

g <- unique(example_genes[,1])
n <- length(g)
d <- matrix(nrow = n, ncol = n)
rownames(d) <- colnames(d) <- g
genes <- lapply(g, get_genes, data = example_genes)

for (i in 1:n) {
    for (j in 1:i) {
        jaccard_sim <- length(intersect(genes[[i]],
         genes[[j]])) /
                       length(union(genes[[i]], genes[[j]]))
        d[j, i] <- d[i, j] <- 1 - jaccard_sim
    }
}

tree <- ape::bionj(d)
```

```
p <- ggtree(tree, branch.length='none') +
    geom_tiplab() + xlim_tree(5.5) +
    geom_facet(mapping = aes(xmin = start, xmax = end, fill
        = gene),
               data = example_genes, geom = geom_motif,
        panel = 'Alignment',
               on = 'genE', label = 'gene', align = 'left')
        +
    scale_fill_brewer(palette = "Set3") +
    scale_x_continuous(expand=c(0,0)) +
    theme(strip.text=element_blank(),
        panel.spacing=unit(0, 'cm'))

facet_widths(p, widths=c(1,2))
```

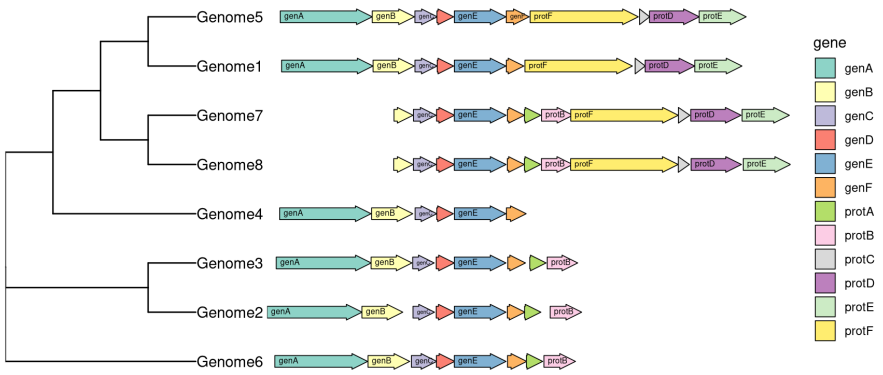


Figure 11.3: **Genomic features with phylogenetic tree.**

12

ggtree extensions

```
rp <- BiocManager::repositories()
db <- utils::available.packages(repo=rp)
x <- tools::package_dependencies('ggtree', db=db,
                                 which = c("Depends",
        "Imports"),
                                 reverse=TRUE)
print(x)
```

```
## $ggtree
## [1] "LINC"           "LymphoSeq"      "philr"
## [4] "singleCellTK"   "universalmotif" "genBaRcode"
## [7] "harrietr"       "STraTUS"
```

There are 8 packages in CRAN or Bioconductor that depend or import ggtree (http://bioconductor.org/packages/ggtree) and several packages on github that extends ggtree.

- deeptime (https://github.com/willgearty/deeptime): Plotting Tools for Anyone Working in Deep Time
- ggnetworx (https://github.com/KlausVigo/ggnetworx): phylogenetic networks using ggplot2 and ggtree
- microbiomeViz (https://github.com/lch14forever/microbiomeViz): Visualize microbiome data with black magic ggtree

Appendix

A

Frequently asked questions

A.1 Installation

ggtree (http://bioconductor.org/packages/ggtree) is released within the Bioconductor project, you need to use BiocManager (https://CRAN.R-project.org/package=BiocManager) to install it.

```
## you need to install BiocManager before using it
## install.packages("BiocManager")
library(BiocManager)
install("ggtree")
```

Bioconductor release is adhere to specific R version. Please make sure you are using latest version of R if you want to install the latest release of Bioconductor packages, including `ggtree`. Beware that bugs will only be fixed in current release and develop branches. If you find a bug, please follow the guide[19] to report it.

A.2 Basic R related

A.2.1 Use your local file

If you are new to `R` and want to use `ggtree` for tree visualization, please do learn some basic `R` and `ggplot2`.

A very common issue is that users always copy-paste command without looking at the function's behavior. `system.file()` was used in the `treeio` and `ggtree` package documentation to find files in the packages.

```
system.file                package:base                R Documentation

Find Names of R System Files

Description:

     Finds the full file names of files in packages etc.

Usage:

     system.file(..., package = "base", lib.loc = NULL,
                 mustWork = FALSE)
```

For users who want to use their own files, please just use relative or absolute file path (*e.g.* `f = "your/folder/filename"`).

A.3 Aesthetic mapping

A.3.1 Inherit *aes*

```
ggtree(rtree(30)) + geom_point()
```

For example, we can add symbolic points to nodes with `geom_point()` directly. The magic here is we don't need to map `x` and `y` position of the points by providing `aes(x, y)` to `geom_point()` since it was already mapped by `ggtree` function and it serves as a global mapping for all layers.

But what if we provide a `dataset` in a layer and the `dataset` doesn't contain column of `x` and/or `y`, the layer function also try to map `x` and `y`

and also others if you map them in ggtree function. As these variable is not available in your dataset, you will get the following error:

```
Error in eval(expr, envir, enclos) : object 'x' not found
```

This can be fixed by using parameter inherit.aes=FALSE which will disable inheriting mapping from ggtree function.

A.3.2 Never use $ in aes

NEVER DO THIS[20].

See the explaination in the ggplot2 book 2ed (https://github.com/hadley/ggplot2-book/blob/master/layers.rmd#aesthetic-mappings-secaes):

> Never refer to a variable with $ (e.g., diamonds$carat) in aes(). This breaks containment, so that the plot no longer contains everything it needs, and causes problems if ggplot2 changes the order of the rows, as it does when facetting.

A.4 Text & Label

A.4.1 Tip label truncated

ggplot2 (https://CRAN.R-project.org/package=ggplot2) can't auto adjust xlim based on added text[21].

```
library(ggtree)
## example tree from
        https://support.bioconductor.org/p/72398/
tree <- read.tree(text= paste("
        (Organism1.006G249400.1:0.03977,
        (Organism2.022118m:0.01337,",
            "
        (Organism3.J34265.1:0.00284,Organism4.G02633.1:0.00468)0.51:0.0
```

```
        ))
p <- ggtree(tree) + geom_tiplab()
```

In this example, the tip labels displayed on Figure A.1A are truncated. This is because the units are in two different spaces (data and pixel). Users can use `xlim` to allocate more spaces for tip labels (Figure A.1B).

```
p + xlim(0, 0.08)
```

Another solution is to set `clip = "off"` to allow drawing outside of the plot panel. We may also need to set `plot.margin` to allocate more spaces for margin (Figure A.1C).

```
p + coord_cartesian(clip = 'off') +
  theme_tree2(plot.margin=margin(6, 120, 6, 6))
```

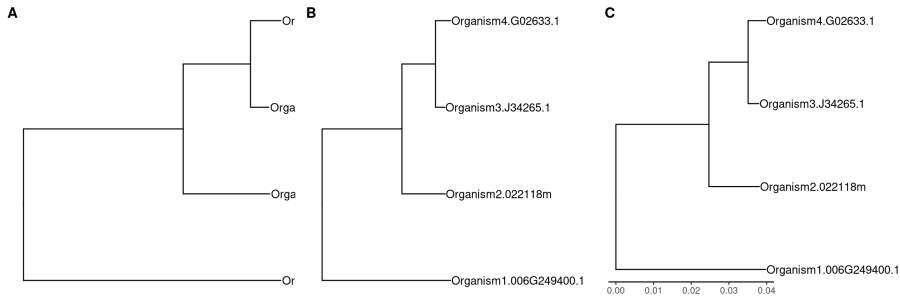


Figure A.1: **Allocating more spaces for truncated tip lables.** Long tip lables may be truncated (A). One solution is to allocate more spaces for plot panel (B) and another solution is to allow plotting labels outside the plot panel (C).

A.4.2 Modify (tip) labels

If you want to modify tip labels of the tree, you can use `treeio::rename_taxa()` to rename a `phylo` or `treedata` object.

```
tree <- read.tree(text = "((A, B), (C, D));")
d <- data.frame(label = LETTERS[1:4],
                label2 = c("sunflower", "tree", "snail",
        "mushroom"))

## rename_taxa use 1st column as key and 2nd column as value
        by default
## rename_taxa(tree, d)
rename_taxa(tree, d, label, label2) %>% write.tree
```

```
## [1] "((sunflower,tree),(snail,mushroom));"
```

If the input tree object is a `treedata` instance, you can use `write.beast()` to export the tree with with associated data to a BEAST compatible NEXUS file.

Renaming phylogeny tip labels seems not be a good idea, since it may introduce problems when mapping the original sequence alignment to the tree. Personally, I recommend to store the new labels as a tip annotation in `treedata` object.

```
tree2 <- full_join(tree, d, by = "label")
tree2
```

```
## 'treedata' S4 object'.
##
## ...@ phylo:
## Phylogenetic tree with 4 tips and 3 internal nodes.
##
## Tip labels:
## [1] "A" "B" "C" "D"
##
## Rooted; no branch lengths.
##
## with the following features available:
##  'label2'.
```

If you just want to show different or additional information when plotting the tree, you don't need to modify tip labels. This could be easily done via the `%<+%` operator to attach the modified version of the labels and than use `geom_tiplab` to display the modified version (Figure A.2).

```
p <- ggtree(tree) + xlim(NA, 3)
p1 <- p + geom_tiplab()

## the following command will produce identical figure of p2
## ggtree(tree2) + geom_tiplab(aes(label = label2))
p2 <- p %<+% d + geom_tiplab(aes(label=label2))
cowplot::plot_grid(p1, p2, ncol=2, labels = c("A", "B"))
```

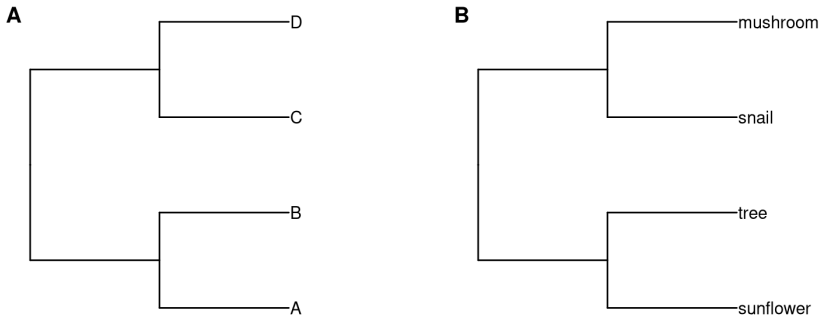


Figure A.2: **Alternative tip labels.** Original tip lables (A) and modified version (B).

A.4.3 Formatting (tip) labels

If you want to format labels, you need to set `parse=TRUE` in `geom_text`/`geom_tiplab` and the `label` should be string that can be parsed into expression and displayed as described in `?plotmath`.

For example, the tip labels contains two parts, species name and accession number and we want to display species name in *italic*, we can use command like this to format specific tip/node label (Figure A.3A):

```
set.seed(2019-06-24)
tree <- rtree(30)
p1 <- ggtree(tree) +
  geom_tiplab(aes(subset=node==35),
                  label='paste(italic("species name"),
                                " accession number")',
              parse=T) + xlim(0, 6)
```

Another example for formating all tip labels is demonstrated in Figure A.3B:

```
p2 <- ggtree(tree) +
  geom_tiplab(aes(label=paste0('bold(', label,
                  ')~italic(', node, ')')),
              parse=TRUE) + xlim(0, 5)
```

The `label` can be provided by a `data.frame` that contains related information of the taxa (Figure A.3C).

```
tree <- read.tree(text = "((a,(b,c)),d);")
genus <- c("Gorilla", "Pan", "Homo", "Pongo")
species <- c("gorilla", "spp.", "sapiens", "pygmaeus")
geo <- c("Africa", "Africa", "World", "Asia")
d <- data.frame(label = tree$tip.label, genus = genus,
                species = species, geo = geo)

p3 <- ggtree(tree) %<+% d + xlim(NA, 6) +
    geom_tiplab(aes(label=paste0('italic(', genus,
                  ')~bolditalic(', species, ')~', geo)),
                parse=T)
cowplot::plot_grid(p1, p2, p3, ncol=3, labels =
         LETTERS[1:3])
```

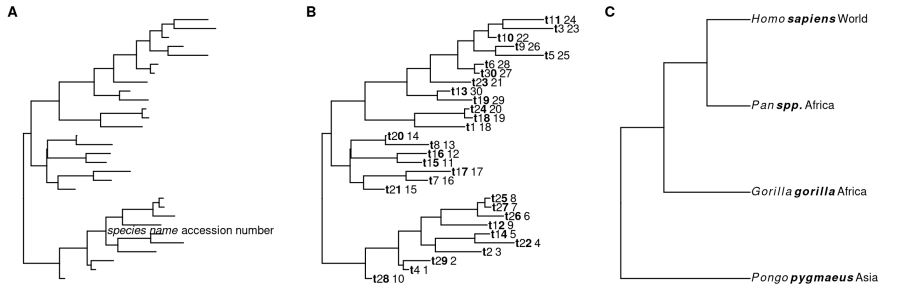


Figure A.3: **Formatting labels.** Formatting specific tip/node label (A), all tip labels (B & C).

A.4.4 Avoid overlapping text labels

User can use ggrepel (https://cran.r-project.org/web/packages/ggrepel/) package to repel overlapping text labels[22]. .

For example:

```
library(ggrepel)
library(ggtree)
raxml_file <- system.file("extdata/RAxML",
        "RAxML_bipartitionsBranchLabels.H3",
        package="treeio")
raxml <- read.raxml(raxml_file)
ggtree(raxml) + geom_label_repel(aes(label=bootstrap,
        fill=bootstrap)) +
  theme(legend.position = c(.1, .8)) +
        scale_fill_viridis_c()
```

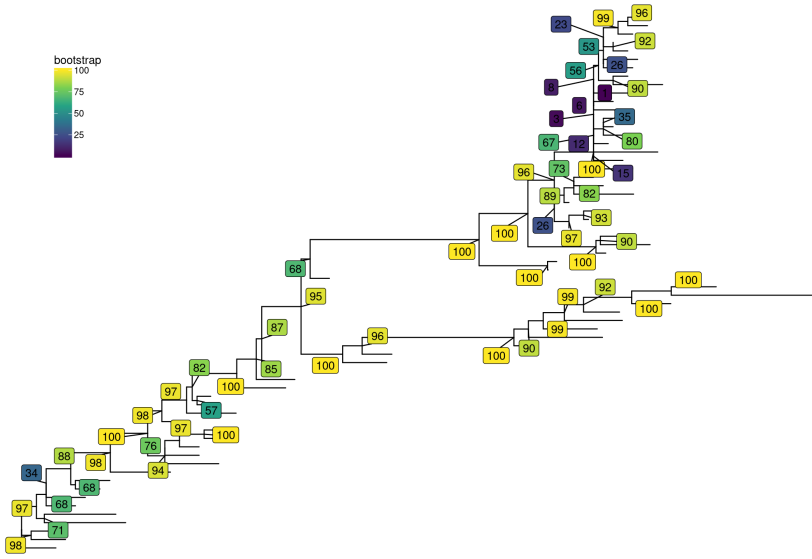


Figure A.4: **Repel labels.** Repel labels to avoid overlapping.

A.4.5 Bootstrap values from newick format

It's quite command to store `bootstrap` value as node label in `newick` format. Visualizing node label is easy using `geom_text2(aes(subset = !isTip, label=label))`.

If you want to only display a subset of `bootstrap` (e.g. bootstrap > 80), you can't simply using `geom_text2(subset= (label > 80), label=label)` since `label` is a character vector, which contains node label (bootstrap value) and tip label (taxa name). If we use `geom_text2(subset= (as.numeric(label) > 80), label=label)`, it will also fail since `NAs` were introduced by coercion. We need to convert `NAs` to logical `FALSE`, this can be done by the following code:

```
nwk <- system.file("extdata/RAxML","RAxML_bipartitions.H3",
        package='ggtree')
tr <- read.tree(nwk)
ggtree(tr) + geom_text2(aes(label=label, subset =
        !is.na(as.numeric(label)) & as.numeric(label) > 80))
```

Another solution is converting the bootstrap value outside ggtree.

```
q <- ggtree(tr)
d <- q$data
d <- d[!d$isTip,]
d$label <- as.numeric(d$label)
d <- d[d$label > 80,]

q + geom_text(data=d, aes(label=label))
```

A.5 Different x labels for different facet panels

This is not supported by ggplot2 (https://CRAN.R-project.org/package=ggplot2) in general. However, we can just draw text labels for each panels and put the labels beyond the plot panels as demonstrated in Figure A.5.

```
library(ggtree)
library(ggplot2)
set.seed(2019-05-02)
x <- rtree(30)
p <- ggtree(x) + geom_tiplab()
d <- data.frame(label = x$tip.label,
                value = rnorm(30))
p2 <- facet_plot(p, panel = "Dot", data = d,
            geom = geom_point, mapping = aes(x = value))

p2 <- p2 + theme_bw() +
    xlim_tree(5) + xlim_expand(c(-5, 5), 'Dot')

d = data.frame(.panel = c('Tree', 'Dot'),
               lab = c("Distance", "Dot Units"),
               x=c(2.5,0), y=-2)

p2 + scale_y_continuous(limits=c(0, 31),
                        expand=c(0,0),
                        oob=function(x, ...) x) +
```

```
    geom_text(aes(label=lab), data=d) +
    coord_cartesian(clip='off')  +
    theme(plot.margin=margin(6, 6, 40, 6))
```

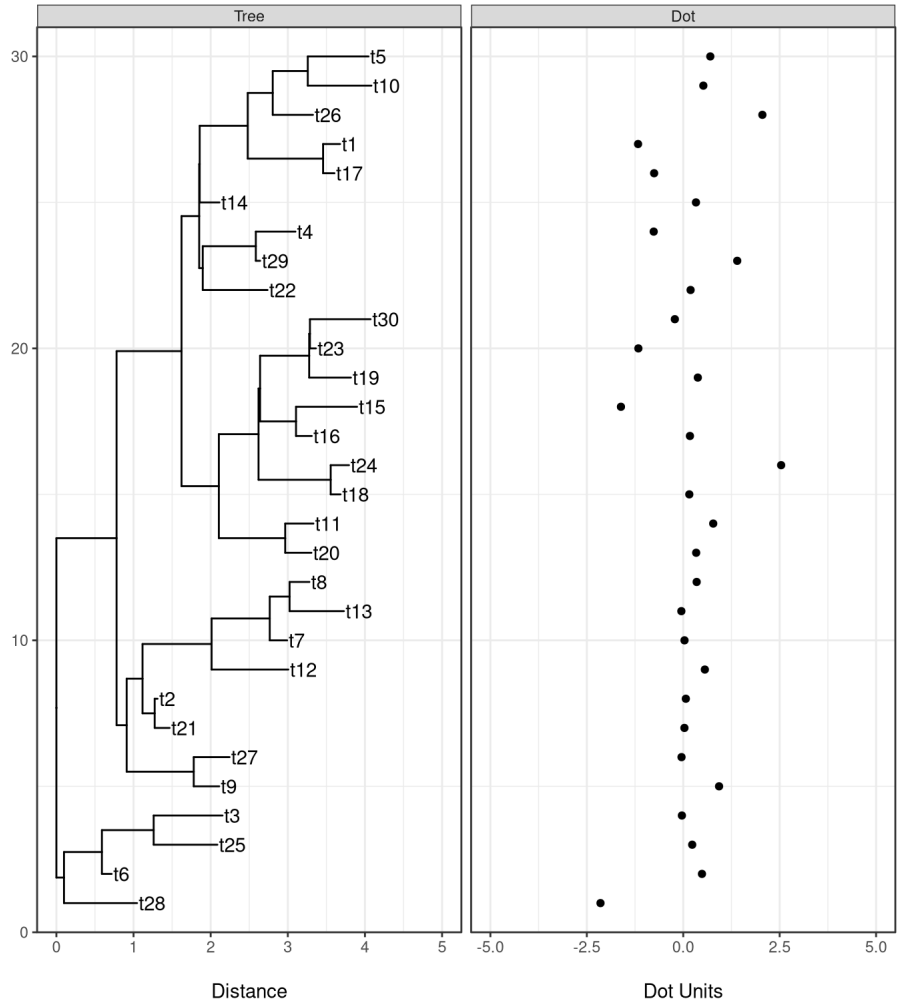


Figure A.5: **X-axis titles for different facet panels.**

A.6 Plot something behind the phylogeny

The ggtree function plot the tree structure and normally we add layers on top of the tree.

```
set.seed(1982)
x <- rtree(5)
p <- ggtree(x) + geom_hilight(7, alpha=1)
```

If we want the layers behind the tree layer, we can reverse the order of all the layers.

```
p$layers <- rev(p$layers)
```

Another solution is to use ggplot() instead of ggtree() and + geom_tree() to add the layer of tree structure at the correct position of layer stack.

```
ggplot(x) + geom_hilight(7, alpha=1) + geom_tree() +
        theme_tree()
```

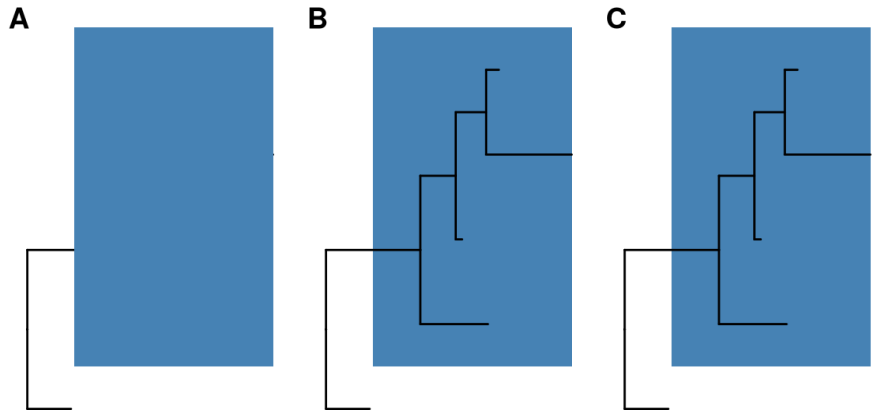


Figure A.6: **Add layers behind tree structure.** A layer on top of the tree structure (A). Reverse layer order of A (B). Add layer behind the tree layer (C).

A.7 Enlarge center space in circular/fan layout tree

This question was asked several times[23], and a published example can be found in https://www.ncbi.nlm.nih.gov/pubmed/27605062. Increasing percentage of center white space in circular tree is useful to avoid overlapping tip labels and to increase readibility of the tree by moving all nodes and branches further out. This can be done simply by using `+xlim()` to allocate more space, just like in Figure 4.3G, or assign a long root branch that is similar to the "Root Length" parameter in FigTree.

```
set.seed(1982)
tree <- rtree(30)
plot_grid(
  ggtree(tree, layout='circular') + xlim(-10, NA),
  ggtree(tree, layout='circular') + geom_rootedge(5),
  labels = c("A", "B", ncol=2)
)
```

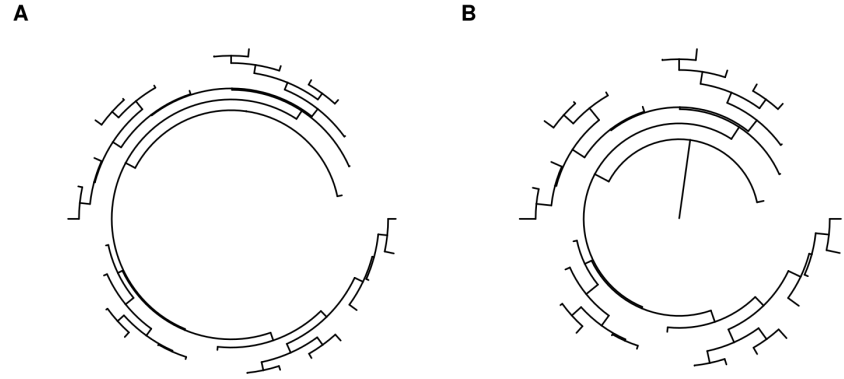


Figure A.7: **Enlarge center space in circular tree.** Allocate more space by `xlim` (A) or long root branch (B).

A.8 Use the most distant tip from the root as the origin of the time scale

The `revts` will reverse the x-axis by setting the most recent tip to 0. We can use `scale_x_continuous(labels=abs)` to label x-axis using absolute values.

```
tr <- rtree(10)
p <- ggtree(tr) + theme_tree2()
p2 <- revts(p) + scale_x_continuous(labels=abs)
plot_grid(p, p2, ncol=2, labels=c("A", "B"))
```

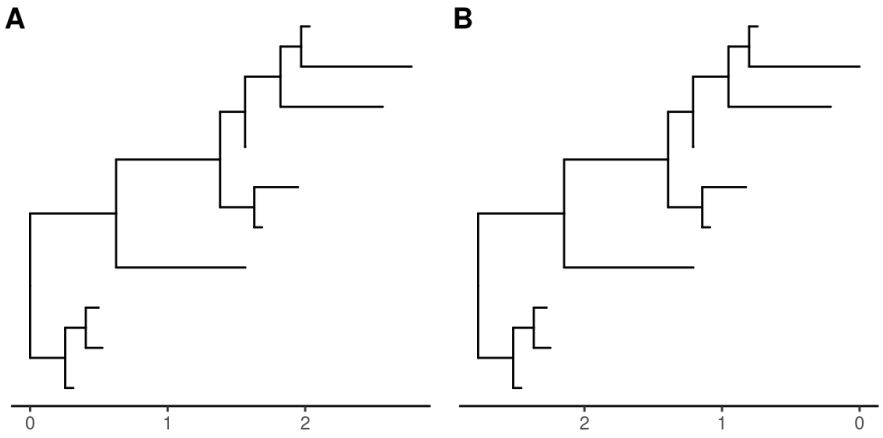


Figure A.8: **Origin of the time scale.** Forward: from the root to tips (A). Backward: from the most distant tip to the root (B).

A.9 Changing branch length of outgroup

When outgroups are on a very long branch length (Figure A.9A), we would like to keep the out groups in the tree but ignore their branch lengths (Figure A.9B)[24]. This can be easily done by modifying coordination of the out groups.

```
x <- read.tree("data/long-branch-example.newick")
m <- MRCA(x, 75, 76)
y <- groupClade(x, m)
p <- p1 <- ggtree(y, aes(linetype = group)) +
  geom_tiplab(size = 2) +
  theme(legend.position = 'none')
p$data[p$data$node %in% c(75, 76), "x"] <- mean(p$data$x)
plot_grid(p1, p, ncol=2)
```

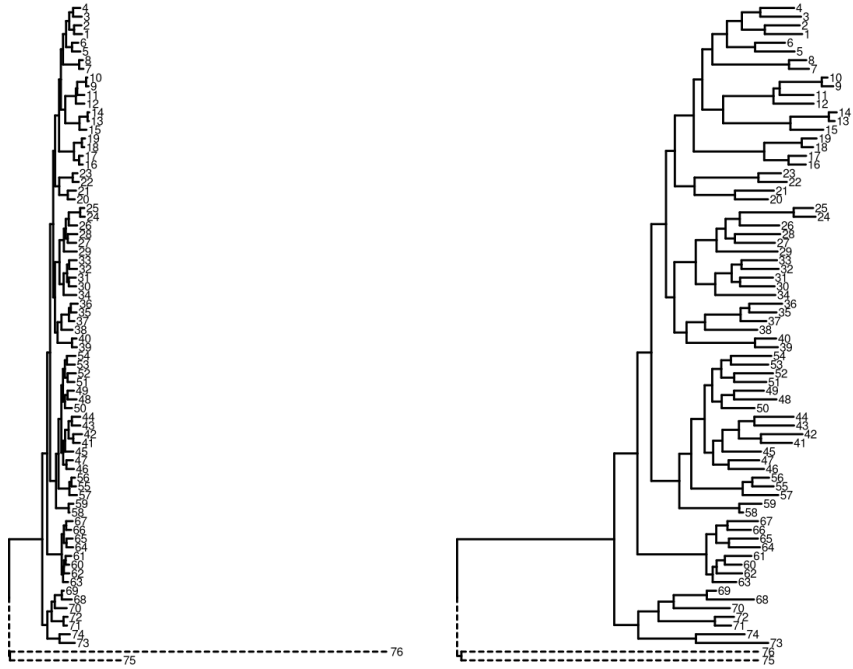


Figure A.9: **Changing branch length of outgroup.** Original tree (A) and reduced outgroup branch length version (B).

A.10 Edit tree graphic details

It can be hard to modify plot details for ordinary users using ggplot2 (https://CRAN.R-project.org/package=ggplot2)/ggtree (http://bioconductor.org/packages/ggtree). We recommend using the export (https://CRAN.R-project.org/package=export) package to export ggtree (http://bioconductor.org/packages/ggtree) output to 'Microsoft Office' Document and edit the tree graphic in 'PowerPoint'.

B

Tips for using `ggtree` *with* `ggimage`

ggtree (http://bioconductor.org/packages/ggtree) supports annotating tree with silhouette images via the ggimage (https://CRAN.R-project.org/package=ggimage) package. ggimage (https://CRAN.R-project.org/package=ggimage) provides grammar of graphic syntax to work with image files. It allows processing images on the fly via the `image_fun` parameter, which accepts a function to process `magick-image` object. The magick (https://CRAN.R-project.org/package=magick) package provides several functions and these functions can be combined to perform a particular task.

B.1 Example 1: Remove background of images

```
set.seed(1982)
x <- rtree(5)
p <- ggtree(x) + theme_grey()
p1 <- p + geom_nodelab(image="img/frogs/frog.jpg",
         geom="image", size=.12) +
           ggtitle("original image")
p2 <- p + geom_nodelab(image="img/frogs/frog.jpg",
         geom="image", size=.12,
             image_fun= function(.)
         magick::image_transparent(., "white")) +
             ggtitle("image with background removed")
plot_grid(p1, p2, ncol=2)
```

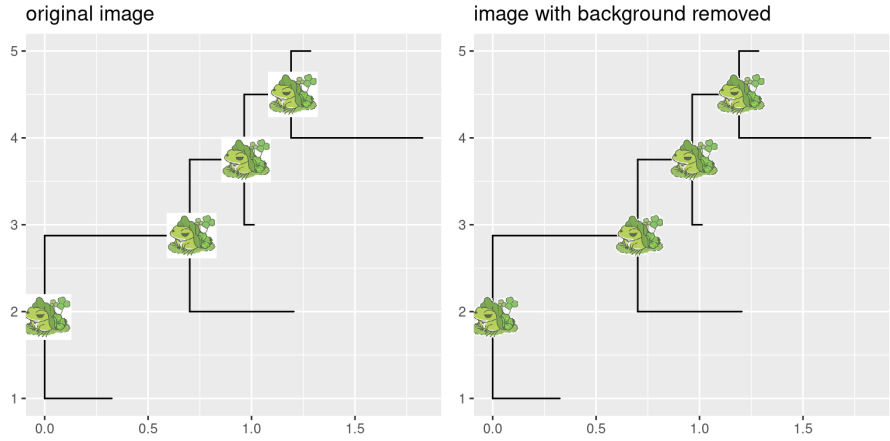


Figure B.1: **Remove image background.** Plotting silhouette images on phylogenetic tree without (A) and with (B) background remove.

B.2 Example 2: Plot tree on a background image

The `geom_bgimage` add a layer of the image and put the layer to the bottom of the layer stack. It is a normal layer and doesn't change the structure of the output `ggtree` object. Users can add annotation layers as without the background image layer.

```
ggtree(rtree(20), size=1.5, color="white") +
  geom_bgimage('img/blackboard.jpg') +
  geom_tiplab(color="white", size=5, family='xkcd')
```

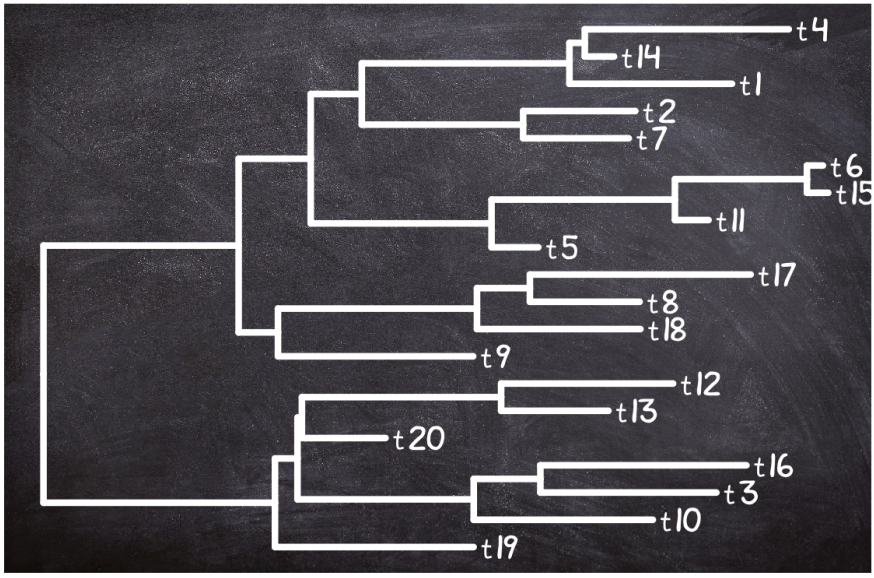


Figure B.2: **Use image file as tree background.**

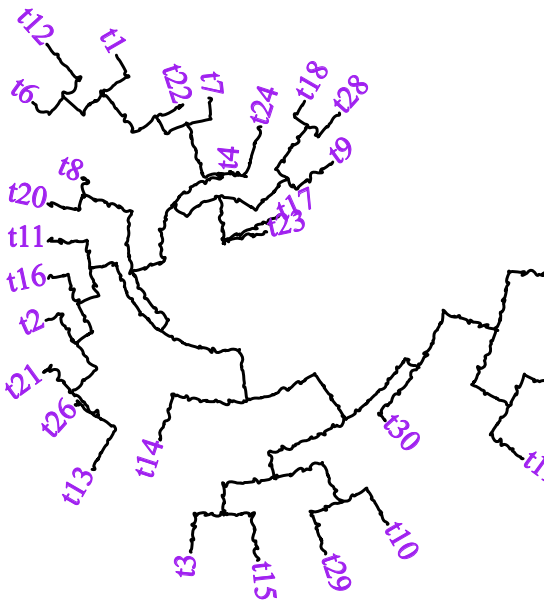
C

Comic (xkcd-like) phylogenetic tree

```
library(htmltools)
library(XML)
library(gridSVG)
library(ggplot2)
library(ggtree)
library(comicR)

p <- ggtree(rtree(30), layout="circular")  +
    geom_tiplab(aes(label=label), color="purple")
print(p)
svg <- grid.export(name="", res=100)$svg
```

```
tagList(
    tags$div(
              id = "ggtree_comic",
              tags$style("#ggtree_comic text {font-
         family:Chalkduster;}"),
              HTML(saveXML(svg)),
              comicR("#ggtree_comic", ff=5)
          )
) # %>% html_print
```


t12
t1
t6
t22
t7
t24
t18
t28
t4
t9
t8
t20
t17
t23
t11
t16
t2
t21
t26
t13
t14
t30
t3
t15
t29
t10

D

Print ASCII-art rooted tree

```
library(data.tree)
tree <- rtree(10)
d <- as.data.frame(as.Node(tree))
names(d) <- NULL
print(d, row.names=FALSE)
```

```
 11
  ¦--12
  ¦   ¦--13
  ¦   ¦   ¦--t4
  ¦   ¦   °--t6
  ¦   °--t10
  °--14
      ¦--15
      ¦   ¦--16
      ¦   ¦   ¦--17
      ¦   ¦   ¦   ¦--t5
      ¦   ¦   ¦   °--t7
      ¦   ¦   °--t2
      ¦   °--18
      ¦       ¦--t1
      ¦       °--t9
      °--19
          ¦--t3
          °--t8
```

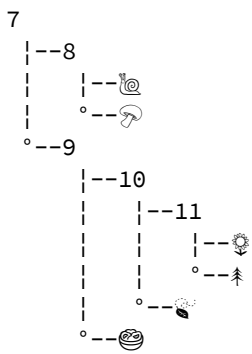
It is neat to print ASCII-art of phylogeny. Sometimes we don't want to plot the tree, but just take a glance at the tree structure without leaving the

focus from R console. However, it is not a good idea to print the whole tree as ASCII text if the tree is large. Sometimes we just want to look at a specific portion of the tree and their immediate relatives. At this scenario, we can use `treeio::tree_subset()` function (see session 2.4) to extract selected portion of a tree. Then we can print ASCII-art of the tree subset to explore evolutionary relationship of interested species in R console.

ggtree (http://bioconductor.org/packages/ggtree) supports parsing tip labels as emoji to create phylomoji (page 193). With the data.tree (https://CRAN.R-project.org/package=data.tree) and emojifont (https://CRAN.R-project.org/package=emojifont) packages, we can also print phylomoji as ASCII text.

```
library(emojifont)

tt <- '((snail,mushroom),
        (((sunflower,evergreen_tree),leaves),green_salad));'
tree <- read.tree(text = tt)
tree$tip.label <- emoji(tree$tip.label)
d <- as.data.frame(as.Node(tree))
names(d) <- NULL
print(d, row.names=FALSE)
```



E

Zoom in selected portion

```
set.seed(2019-08-05)
x <- rtree(30)
nn <- tidytree::offspring(x, 43, self_include=TRUE)
ggtree(x) + ggforce::facet_zoom(xy = node %in% nn)
```

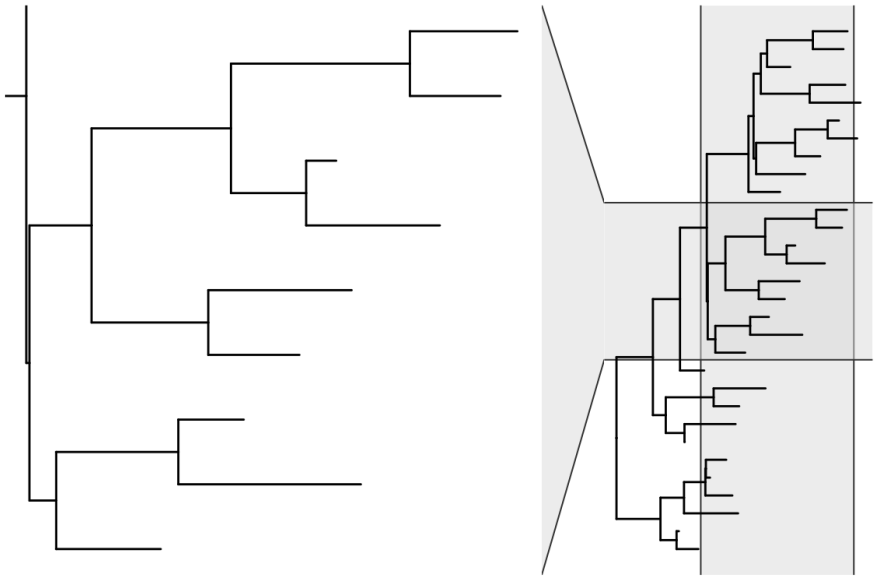


Figure E.1: **Zoom in selected clade.**

F

Run ggtree in Jupyter notebook

If you have Jupyter notebook (https://jupyter.org/) installed on your system, you can install IRkernel (https://irkernel.github.io/) with the following command in R:

```
install.packages("IRkernel")
IRkernel::installspec()
```

Then you can use ggtree and other R packages in Jupyter notebook. Here is a screenshot of recreating Figure 8.6 in Jupyter notebook.

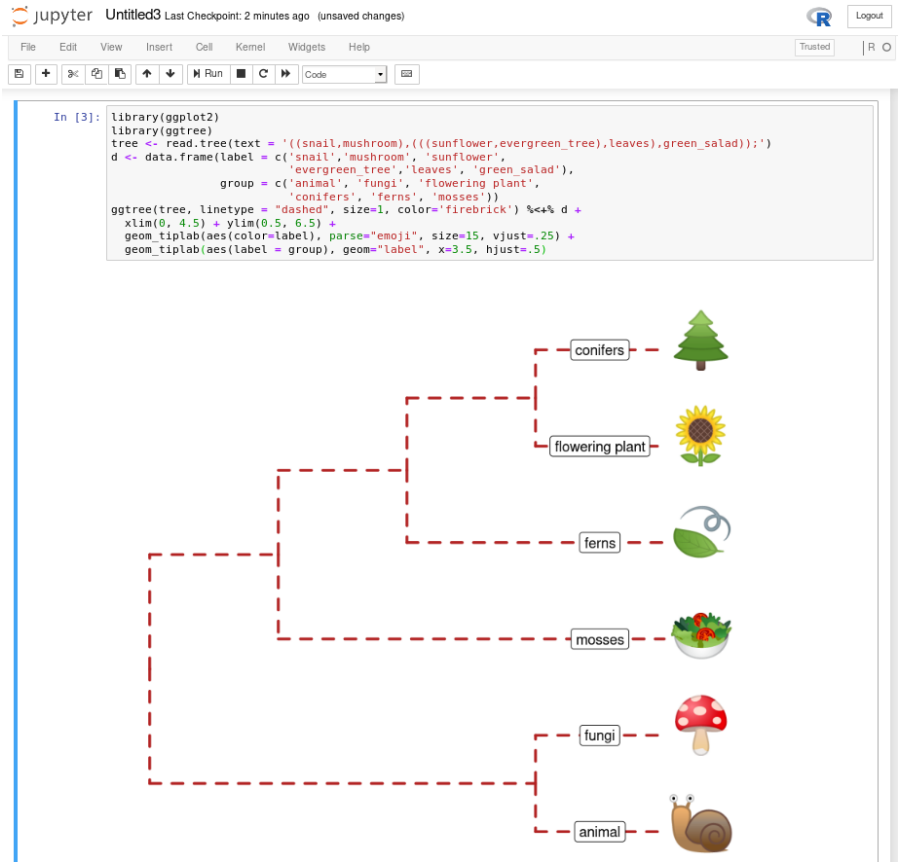


Figure F.1: **ggtree in Jupyter notebook.** Running ggtree in Jupyter notebook via R kernel.

G

Figures and Tables

Table G.1: Geometric layers that supported by 'geom_facet()' and 'facet_plot()'

Package	Geom Layer	Description
ggalt	geom_dumbbell	creates dumbbell charts
ggbio	geom_alignment	shows interval data as alignment
ggfittext	geom_fit_text	shrinks, grows or wraps text to fit inside a defined rectangular area
gggenes	geom_gene_arrow	draws genes as arrows
ggimage	geom_image	visualizes image files
	geom_phylopic	queries image files from phylopic database and visualizes them
ggplot2	geom_hline	adds horizontal lines
	geom_jitter	adds a small amount of random variation to the location of each point
	geom_label	draws a rectangle behind the text
	geom_point	creats scatterplots
	geom_raster	a high performance special case for all the tiles are the same size
	geom_rect	draws rectangle by using the locations of the four coners
	geom_segment	draws a straight line between points
	geom_spoke	a polar parameterisation of 'geom_segment()'

geom_text	adds text to the plot	
geom_tile	draws rectangle by using the center of the tile and its size	
geom_vline	adds vertical lines	
ggrepel	geom_text_repel	adds text to the plot. The text labels repel away from each other and away from the data points
	geom_label_repel	draws a rectangle underneath the text. The text labels repel away from each other and away from the data ponts
ggridges	geom_density_ridges	arranges multiple density plots in a staggered fashion
	geom_density_ridges_gradient	works just like 'geom_density_ridges' except that the 'fill' aesthetic can vary along the x axis
	geom_ridgeline	plots the sum of the 'y' and 'height' aesthetics versus 'x', filling the area between 'y' and 'y + height' with a color
	geom_ridgeline_gradient	works just like 'geom_ridgeline' except that the 'fill' aesthetic can vary along the x axis
ggstance	geom_barh	horizontal version of 'geom_bar()'
	geom_boxploth	horizontal version of 'geom_boxplot()'
	geom_crossbarh	horizontal version of

‘geom_crossbar()’		
geom_errorbarh	horizontal version of ‘geom_errorbarh()’	
geom_histogramh	horizontal version of ‘geom_histogram()’	
geom_linerangeh	horizontal version of ‘geom_linerange()’	
geom_pointrangeh	horizontal version of ‘geom_pointrange()’	
geom_violinh	horizontal version of ‘geom_violin()’	
ggtree	geom_motif	draws aligned motifs

H

Session Info

The book was written using R Markdown and was compiled by bookdown (https://CRAN.R-project.org/package=bookdown) package. Here is the session information on the system on which this thesis was compiled:

```
## R version 3.6.1 (2019-07-05)
##  Platform: x86_64-pc-linux-gnu (64-bit)
##  Running under: Anarchy Linux
##
##  locale:
##   [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##   [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##   [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##   [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##   [9] LC_ADDRESS=C               LC_TELEPHONE=C
##  [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
##  attached base packages:
##   [1] grid      stats4    parallel  stats     graphics
##   [6] grDevices utils     datasets  methods   base
##
##  other attached packages:
##   [1] gggenes_0.4.0        tibble_2.1.3
##   [3] reshape2_1.4.3       ggstance_0.3.3
##   [5] data.tree_0.7.8      comicR_0.2
##   [7] gridSVG_1.7-0        XML_3.98-1.20
##   [9] htmltools_0.3.6      ggrepel_0.8.1
##  [11] ggridges_0.5.1       scales_1.0.0
##  [13] phyloseq_1.28.0      ade4_1.7-13
##  [15] emojifont_0.5.2      ggnewscale_0.3.0
##  [17] ggimage_0.2.3        jsonlite_1.6
##  [19] stringr_1.4.0        phytools_0.6-99
```

```
##  [21] maps_3.3.0           tidyr_1.0.0
##  [23] ggtree_1.99.1        treeio_1.9.3
##  [25] tidytree_0.2.8       kableExtra_1.1.0
##  [27] dplyr_0.8.3          cowplot_1.0.0
##  [29] phylobase_0.8.6      igraph_1.2.4.1
##  [31] OutbreakTools_0.1-16 knitr_1.25
##  [33] network_1.15         ggplot2_3.2.1
##  [35] Biostrings_2.52.0    XVector_0.24.0
##  [37] IRanges_2.18.2       S4Vectors_0.22.1
##  [39] BiocGenerics_0.30.0  ape_5.3
##  [41] pagedown_0.4         rmarkdown_1.15
##
##  loaded via a namespace (and not attached):
##    [1] uuid_0.1-2              backports_1.1.4
##    [3] fastmatch_1.1-0         selectr_0.4-1
##    [5] plyr_1.8.4              lazyeval_0.2.2
##    [7] splines_3.6.1           rncl_0.8.3
##    [9] digest_0.6.21           foreach_1.4.7
##   [11] viridis_0.5.1           magick_2.2
##   [13] fansi_0.4.0             magrittr_1.5
##   [15] cluster_2.1.0           ggfittext_0.8.1
##   [17] sna_2.4                 readr_1.3.1
##   [19] sysfonts_0.8            prettyunits_1.0.2
##   [21] jpeg_0.1-8              colorspace_1.4-1
##   [23] rvest_0.3.4             xfun_0.9
##   [25] hexbin_1.27.3           crayon_1.3.4
##   [27] RCurl_1.95-4.12         zeallot_0.1.0
##   [29] brew_1.0-6              survival_2.44-1.1
##   [31] phangorn_2.5.5          iterators_1.0.12
##   [33] glue_1.3.1              polyclip_1.10-0
##   [35] gtable_0.3.0            zlibbioc_1.30.0
##   [37] webshot_0.5.1           Rook_1.1-1
##   [39] Rhdf5lib_1.6.1          Rcpp_1.0.2
##   [41] showtextdb_2.0          plotrix_3.7-6
##   [43] viridisLite_0.3.0       progress_1.2.2
##   [45] gridGraphics_0.4-1      animation_2.6
##   [47] htmlwidgets_1.3         httr_1.4.1
##   [49] DiagrammeR_1.0.1        RColorBrewer_1.1-2
##   [51] ellipsis_0.3.0          farver_1.1.0
##   [53] pkgconfig_2.0.3         utf8_1.1.4
##   [55] labeling_0.3            ggplotify_0.0.4
##   [57] tidyselect_0.2.5        rlang_0.4.0
##   [59] visNetwork_2.0.8        munsell_0.5.0
```

```
##   [61] tools_3.6.1             cli_1.1.0
##   [63] downloader_0.4          statnet.common_4.3.0
##   [65] ggmap_3.0.0             evaluate_0.14
##   [67] biomformat_1.12.0       yaml_2.2.0
##   [69] purrr_0.3.2             RgoogleMaps_1.4.4
##   [71] showtext_0.7            nlme_3.1-140
##   [73] xml2_1.2.2              networkDynamic_0.10.0
##   [75] compiler_3.6.1          rstudioapi_0.10
##   [77] curl_4.2                rgexf_0.15.3
##   [79] png_0.1-7               clusterGeneration_1.3.4
##   [81] tweenr_1.0.1            RNeXML_2.3.0
##   [83] stringi_1.4.3           highr_0.8
##   [85] lattice_0.20-38         Matrix_1.2-17
##   [87] vegan_2.5-6             permute_0.9-5
##   [89] multtest_2.40.0         vctrs_0.2.0
##   [91] pillar_1.4.2            lifecycle_0.1.0
##   [93] BiocManager_1.30.4      combinat_0.0-8
##   [95] data.table_1.12.2       bitops_1.0-6
##   [97] R6_2.4.0                bookdown_0.13
##   [99] gridExtra_2.3           codetools_0.2-16
##  [101] MASS_7.3-51.4           gtools_3.8.1
##  [103] assertthat_0.2.1        rhdf5_2.28.0
##  [105] proto_1.0.0             rjson_0.2.20
##  [107] withr_2.1.2             mnormt_1.5-5
##  [109] mgcv_1.8-28             expm_0.999-4
##  [111] hms_0.5.1               influenceR_0.1.0
##  [113] quadprog_1.5-7          coda_0.19-3
##  [115] rvcheck_0.1.4           ggforce_0.3.1
##  [117] numDeriv_2016.8-1.1     scatterplot3d_0.3-41
##  [119] Biobase_2.44.0
```

References

Arenas, Miguel. 2015. "Trends in Substitution Models of Molecular Evolution." *Frontiers in Genetics* 6 (October). https://doi.org/10.3389/fgene.2015.00319.

Berger, Simon A., Denis Krompass, and Alexandros Stamatakis. 2011a. "Performance, Accuracy, and Web Server for Evolutionary Placement of Short Sequence Reads Under Maximum Likelihood." *Systematic Biology*, March, syr010. https://doi.org/10.1093/sysbio/syr010.

———. 2011b. "Performance, Accuracy, and Web Server for Evolutionary Placement of Short Sequence Reads Under Maximum Likelihood." *Systematic Biology* 60 (3): 291–302. https://doi.org/10.1093/sysbio/syr010.

Bosi, Emanuele, Jonathan M. Monk, Ramy K. Aziz, Marco Fondi, Victor Nizet, and Bernhard Ø. Palsson. 2016. "Comparative Genome-Scale Modelling of Staphylococcus Aureus Strains Identifies Strain-Specific Metabolic Capabilities Linked to Pathogenicity." *Proceedings of the National Academy of Sciences of the United States of America* 113 (26): E3801–E3809. https://doi.org/10.1073/pnas.1523199113.

Bouckaert, Remco, Joseph Heled, Denise Kühnert, Tim Vaughan, Chieh-Hsi Wu, Dong Xie, Marc A. Suchard, Andrew Rambaut, and Alexei J. Drummond. 2014. "BEAST 2: A Software Platform for Bayesian Evolutionary Analysis." *PLoS Comput Biol* 10 (4): e1003537. https://doi.org/10.1371/journal.pcbi.1003537.

Boussau, Bastien, Gergely J. Szöllősi, Laurent Duret, Manolo Gouy, Eric Tannier, and Vincent Daubin. 2013. "Genome-Scale Coestimation of Species and Gene Trees." *Genome Research* 23 (2): 323–30. https://doi.org/10.1101/gr.141978.112.

Callahan, Benjamin J., Paul J. McMurdie, Michael J. Rosen, Andrew W. Han, Amy Jo A. Johnson, and Susan P. Holmes. 2016. "DADA2: High-Resolution Sample Inference from Illumina Amplicon Data." *Nature Methods* 13 (7): 581–83. https://doi.org/10.1038/nmeth.3869.

Chen, Zigui, Wendy C. S. Ho, Siaw Shi Boon, Priscilla T. Y. Law, Martin C. W. Chan, Rob DeSalle, Robert D. Burk, and Paul K. S. Chan. 2017. "Ancient Evolution and Dispersion of Human Papillomavirus 58 Variants." *Journal of Virology* 91 (21): e01285–17. https://doi.org/10.1128/JVI.01285-17.

Chevenet, François, Christine Brun, Anne-Laure Bañuls, Bernard Jacq, and Richard Christen. 2006. "TreeDyn: Towards Dynamic Graphics and Annotations for Analyses of Trees." *BMC Bioinformatics* 7 (October): 439. https://doi.org/10.1186/1471-2105-7-439.

Felsenstein, J. 1981. "Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach." *Journal of Molecular Evolution* 17 (6): 368–76.

Felsenstein, Joseph. 1989. "PHYLIP - Phylogeny Inference Package (Version 3.2)." *Cladistics* 5: 164–66.

———. 1978. "Cases in Which Parsimony or Compatibility Methods Will Be Positively Misleading." *Systematic Biology* 27 (4): 401–10. https://doi.org/10.1093/sysbio/27.4.401.

Fitch, Walter M. 1971. "Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology." *Systematic Zoology* 20 (4): 406–16. https://doi.org/10.2307/2412116.

Gentleman, Robert C, Vincent J Carey, Douglas M Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, et al. 2004. "Bioconductor: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology* 5 (10): R80. https://doi.org/10.1186/gb-2004-5-10-r80.

Goldman, N., and Z. Yang. 1994. "A Codon-Based Model of Nucleotide Substitution for Protein-Coding DNA Sequences." *Molecular Biology and Evolution* 11 (5): 725–36.

Grubaugh, Nathan D., Jason T. Ladner, Moritz U. G. Kraemer, Gytis Dudas, Amanda L. Tan, Karthik Gangavarapu, Michael R. Wiley, et al.

2017. “Genomic Epidemiology Reveals Multiple Introductions of Zika Virus into the United States.” *Nature* 546 (7658): 401–5. https://doi.org/10.1038/nature22400.

Gupta, Ankit, and Vineet K Sharma. 2015. “Using the Taxon-Specific Genes for the Taxonomic Classification of Bacterial Genomes.” *BMC Genomics* 16 (1). https://doi.org/10.1186/s12864-015-1542-0.

He, Ya-Qing, Long Chen, Wen-Bo Xu, Hong Yang, Han-Zhong Wang, Wen-Ping Zong, Hui-Xia Xian, et al. 2013. “Emergence, Circulation, and Spatiotemporal Phylogenetic Analysis of Coxsackievirus A6- and Coxsackievirus A10-Associated Hand, Foot, and Mouth Disease Infections from 2008 to 2012 in Shenzhen, China.” *Journal of Clinical Microbiology* 51 (11): 3560–6. https://doi.org/10.1128/JCM.01231-13.

He, Zilong, Huangkai Zhang, Shenghan Gao, Martin J. Lercher, Wei-Hua Chen, and Songnian Hu. 2016. “Evolview V2: An Online Visualization and Management Tool for Customized and Annotated Phylogenetic Trees.” *Nucleic Acids Research* 44 (W1): W236–241. https://doi.org/10.1093/nar/gkw370.

Höhna, Sebastian, Tracy A. Heath, Bastien Boussau, Michael J. Landis, Fredrik Ronquist, and John P. Huelsenbeck. 2014. “Probabilistic Graphical Model Representation in Phylogenetics.” *Systematic Biology* 63 (5): 753–71. https://doi.org/10.1093/sysbio/syu039.

Höhna, Sebastian, Michael J. Landis, Tracy A. Heath, Bastien Boussau, Nicolas Lartillot, Brian R. Moore, John P. Huelsenbeck, and Fredrik Ronquist. 2016. “RevBayes: Bayesian Phylogenetic Inference Using Graphical Models and an Interactive Model-Specification Language.” *Systematic Biology* 65 (4): 726–36. https://doi.org/10.1093/sysbio/syw021.

Huelsenbeck, J. P., and F. Ronquist. 2001. “MRBAYES: Bayesian Inference of Phylogenetic Trees.” *Bioinformatics (Oxford, England)* 17 (8): 754–55.

Huson, Daniel H., and Celine Scornavacca. 2012. “Dendroscope 3: An Interactive Tool for Rooted Phylogenetic Trees and Networks.” *Systematic Biology* 61 (6): 1061–7. https://doi.org/10.1093/sysbio/sys062.

Jombart, Thibaut, David M. Aanensen, Marc Baguelin, Paul Birrell, Simon Cauchemez, Anton Camacho, Caroline Colijn, et al. 2014. “OutbreakTools:

A New Platform for Disease Outbreak Analysis Using the R Software." *Epidemics* 7 (June): 28–34. https://doi.org/10.1016/j.epidem.2014.04.003.

Kuczynski, Justin, Jesse Stombaugh, William Anton Walters, Antonio González, J. Gregory Caporaso, and Rob Knight. 2011. "Using QIIME to Analyze 16S rRNA Gene Sequences from Microbial Communities." *Current Protocols in Bioinformatics / Editoral Board, Andreas D. Baxevanis ... [et Al.]* CHAPTER (December): Unit10.7. https://doi.org/10.1002/0471250953.bi1007s36.

Kumar, Sudhir, Glen Stecher, and Koichiro Tamura. 2016. "MEGA7: Molecular Evolutionary Genetics Analysis Version 7.0 for Bigger Datasets." *Molecular Biology and Evolution* 33 (7): 1870–4. https://doi.org/10.1093/molbev/msw054.

Kunin, Victor, and Philip Hugenholtz. 2010. "PyroTagger : A Fast , Accurate Pipeline for Analysis of rRNA Amplicon Pyrosequence Data." *The Open Journal*, 1–8. http://www.theopenjournal.org/toj_articles/1.

Lam, Tommy Tsan-Yuk, Chung-Chau Hon, Philippe Lemey, Oliver G. Pybus, Mang Shi, Hein Min Tun, Jun Li, Jingwei Jiang, Edward C. Holmes, and Frederick Chi-Ching Leung. 2012. "Phylodynamics of H5N1 Avian Influenza Virus in Indonesia." *Molecular Ecology* 21 (12): 3062–77. https://doi.org/10.1111/j.1365-294X.2012.05577.x.

Lam, Tommy Tsan-Yuk, Boping Zhou, Jia Wang, Yujuan Chai, Yongyi Shen, Xinchun Chen, Chi Ma, et al. 2015. "Dissemination, Divergence and Establishment of H7N9 Influenza Viruses in China." *Nature* 522 (7554): 102–5. https://doi.org/10.1038/nature14348.

Lemmon, Alan R., and Emily C. Moriarty. 2004. "The Importance of Proper Model Assumption in Bayesian Phylogenetics." *Systematic Biology* 53 (2): 265–77. https://doi.org/10.1080/10635150490423520.

Letunic, Ivica, and Peer Bork. 2007. "Interactive Tree of Life (iTOL): An Online Tool for Phylogenetic Tree Display and Annotation." *Bioinformatics* 23 (1): 127–28. https://doi.org/10.1093/bioinformatics/btl529.

Liang, Huyi, Tommy Tsan-Yuk Lam, Xiaohui Fan, Xinchun Chen, Yu Zeng, Ji Zhou, Lian Duan, et al. 2014. "Expansion of Genotypic Diversity and Establishment of 2009 H1N1 Pandemic-Origin Internal Genes in Pigs in

China." *Journal of Virology*, July, JVI.01327–14. https://doi.org/10.1128/JVI.01327-14.

Lott, Steffen C., Björn Voß, Wolfgang R. Hess, and Claudia Steglich. 2015. "CoVennTree: A New Method for the Comparative Analysis of Large Datasets." *Frontiers in Genetics* 6: 43. https://doi.org/10.3389/fgene.2015.00043.

Maddison, David R., David L. Swofford, Wayne P. Maddison, and David Cannatella. 1997. "Nexus: An Extensible File Format for Systematic Information." *Systematic Biology* 46 (4): 590–621. https://doi.org/10.1093/sysbio/46.4.590.

Matsen, Frederick A., Noah G. Hoffman, Aaron Gallagher, and Alexandros Stamatakis. 2012. "A Format for Phylogenetic Placements." *PLoS ONE* 7 (2): e31009. https://doi.org/10.1371/journal.pone.0031009.

Matsen, Frederick A., Robin B. Kodner, and E Virginia Armbrust. 2010. "Pplacer: Linear Time Maximum-Likelihood and Bayesian Phylogenetic Placement of Sequences onto a Fixed Reference Tree." *BMC Bioinformatics* 11: 538. https://doi.org/10.1186/1471-2105-11-538.

Matsen, Frederick A, Robin B Kodner, and E Virginia Armbrust. 2010. "Pplacer: Linear Time Maximum-Likelihood and Bayesian Phylogenetic Placement of Sequences onto a Fixed Reference Tree." *BMC Bioinformatics* 11 (1): 538. https://doi.org/10.1186/1471-2105-11-538.

McMurdie, Paul J., and Susan Holmes. 2013. "Phyloseq: An R Package for Reproducible Interactive Analysis and Graphics of Microbiome Census Data." *PLoS ONE* 8 (4): e61217. https://doi.org/10.1371/journal.pone.0061217.

Neher, Richard A., Trevor Bedford, Rodney S. Daniels, Colin A. Russell, and Boris I. Shraiman. 2016. "Prediction, Dynamics, and Visualization of Antigenic Phenotypes of Seasonal Influenza Viruses." *Proceedings of the National Academy of Sciences* 113 (12): E1701–E1709. https://doi.org/10.1073/pnas.1525578113.

Page, Roderic D. M. 2002. "Visualizing Phylogenetic Trees Using TreeView." *Current Protocols in Bioinformatics* Chapter 6 (August): Unit 6.2. https://doi.org/10.1002/0471250953.bi0602s01.

Paradis, Emmanuel, Julien Claude, and Korbinian Strimmer. 2004. "APE: Analyses of Phylogenetics and Evolution in R Language." *Bioinformatics* 20 (2): 289–90. https://doi.org/10.1093/bioinformatics/btg412.

Pond, Sergei L. Kosakovsky, Simon D. W. Frost, and Spencer V. Muse. 2005. "HyPhy: Hypothesis Testing Using Phylogenies." *Bioinformatics (Oxford, England)* 21 (5): 676–79. https://doi.org/10.1093/bioinformatics/bti079.

Rannala, B., and Z. Yang. 1996. "Probability Distribution of Molecular Evolutionary Trees: A New Method of Phylogenetic Inference." *Journal of Molecular Evolution* 43 (3): 304–11.

R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Retief, J. D. 2000. "Phylogenetic Analysis Using PHYLIP." *Methods in Molecular Biology (Clifton, N.J.)* 132: 243–58.

Revell, Liam J. 2012. "Phytools: An R Package for Phylogenetic Comparative Biology (and Other Things)." *Methods in Ecology and Evolution* 3 (2): 217–23. https://doi.org/10.1111/j.2041-210X.2011.00169.x.

Sanderson, Michael J. 2003. "R8s: Inferring Absolute Rates of Molecular Evolution and Divergence Times in the Absence of a Molecular Clock." *Bioinformatics* 19 (2): 301–2. https://doi.org/10.1093/bioinformatics/19.2.301.

Schliep, Klaus Peter. 2011. "Phangorn: Phylogenetic Analysis in R." *Bioinformatics* 27 (4): 592–93. https://doi.org/10.1093/bioinformatics/btq706.

Schloss, Patrick D., Sarah L. Westcott, Thomas Ryabin, Justine R. Hall, Martin Hartmann, Emily B. Hollister, Ryan A. Lesniewski, et al. 2009. "Introducing Mothur: Open-Source, Platform-Independent, Community-Supported Software for Describing and Comparing Microbial Communities." *Applied and Environmental Microbiology* 75 (23): 7537–41. https://doi.org/10.1128/AEM.01541-09.

Schmidt, Heiko A., Korbinian Strimmer, Martin Vingron, and Arndt von Haeseler. 2002. "TREE-PUZZLE: Maximum Likelihood Phylogenetic

Analysis Using Quartets and Parallel Computing." *Bioinformatics (Oxford, England)* 18 (3): 502–4.

Schön, Isa, Rylan Shearn, Koen Martens, Annette Koenders, and Stuart Halse. 2015. "Age and Origin of Australian Bennelongia (Crustacea, Ostracoda)." *Hydrobiologia* 750 (1): 125–46. https://doi.org/10.1007/s10750-014-2159-z.

Shoemaker, J. S., and W. M. Fitch. 1989. "Evidence from Nuclear Sequences That Invariable Sites Should Be Considered When Sequence Divergence Is Calculated." *Molecular Biology and Evolution* 6 (3): 270–89.

Stamatakis, Alexandros. 2014. "RAxML Version 8: A Tool for Phylogenetic Analysis and Post-Analysis of Large Phylogenies." *Bioinformatics*, January, btu033. https://doi.org/10.1093/bioinformatics/btu033.

Venkatesh, Divya, Marjolein J. Poen, Theo M. Bestebroer, Rachel D. Scheuer, Oanh Vuong, Mzia Chkhaidze, Anna Machablishvili, et al. 2018. "Avian Influenza Viruses in Wild Birds: Virus Evolution in a Multihost Ecosystem." *Journal of Virology* 92 (15). https://doi.org/10.1128/JVI.00433-18.

Vos, Rutger A., James P. Balhoff, Jason A. Caravas, Mark T. Holder, Hilmar Lapp, Wayne P. Maddison, Peter E. Midford, et al. 2012. "NeXML: Rich, Extensible, and Verifiable Representation of Comparative Data and Metadata." *Systematic Biology* 61 (4): 675–89. https://doi.org/10.1093/sysbio/sys025.

Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. 1st ed. Springer.

———. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer. http://ggplot2.org.

Wilgenbusch, James C., and David Swofford. 2003. "Inferring Evolutionary Trees with PAUP*." *Current Protocols in Bioinformatics* Chapter 6 (February): Unit 6.4. https://doi.org/10.1002/0471250953.bi0604s00.

Wilkinson, Leland, D. Wills, D. Rope, A. Norton, and R. Dubbs. 2005. *The Grammar of Graphics*. 2nd edition. New York: Springer.

Yang, Z. 1994. "Maximum Likelihood Phylogenetic Estimation from DNA Sequences with Variable Rates over Sites: Approximate Methods." *Journal*

of Molecular Evolution 39 (3): 306–14.

Yang, Ziheng. 2007. "PAML 4: Phylogenetic Analysis by Maximum Likelihood." *Molecular Biology and Evolution* 24 (8): 1586–91. https://doi.org/10.1093/molbev/msm088.

Yu, Guangchuang, Tommy Tsan-Yuk Lam, Huachen Zhu, and Yi Guan. 2018. "Two Methods for Mapping and Visualizing Associated Data on Phylogeny Using Ggtree." *Molecular Biology and Evolution* 35 (12): 3041–3. https://doi.org/10.1093/molbev/msy194.

Yu, Guangchuang, David K. Smith, Huachen Zhu, Yi Guan, and Tommy Tsan-Yuk Lam. 2017. "Ggtree: An R Package for Visualization and Annotation of Phylogenetic Trees with Their Covariates and Other Associated Data." *Methods in Ecology and Evolution* 8 (1): 28–36. https://doi.org/10.1111/2041-210X.12628.

Zmasek, Christian M., and Sean R. Eddy. 2001. "ATV: Display and Manipulation of Annotated Phylogenetic Trees." *Bioinformatics* 17 (4): 383–84. https://doi.org/10.1093/bioinformatics/17.4.383.

Footnotes:

1. http://evolution.genetics.washington.edu/phylip/newick_doc.html

↩

2. http://www.genetics.wustl.edu/eddy/forester/NHX.html↩
3. https://sites.google.com/site/cmzmasek/home/software/archaeopteryx

↩

4. http://biopython.org/wiki/PAML↩

5. http://bioperl.org/howtos/PAML_HOWTO.html↩

6. https://github.com/ropensci/software-review/issues/179#issuecomment-369164110↩

7. users can also use `read.tree` and `read.nexus` from ape (https://CRAN.R-project.org/package=ape) to import phylogenetic tree from newick string and NEXUS file↩

8. https://groups.google.com/forum/#!msg/bioc-ggtree/Q4LnwoTf1DM/uqYdYB_VBAAJ↩

9. http://beast.community/nexus_metacomments↩

10. http://beast.community/figtree↩

11. http://tree.bio.ed.ac.uk/software/figtree/↩

12. http://www.cbs.dtu.dk/services/NetNGlyc/↩

13. If you want to plot the tree above the highlighting area, visit FAQ (page 250) for details.↩

14. https://guangchuangyu.github.io/2016/06/identify-method-for-ggtree↩

15. see also discussion in https://github.com/GuangchuangYu/ggtree/issues/78 and https://groups.google.com/d/msg/bioc-ggtree/VQqbF79NAWU/IjIvpQOBGwAJ↩

16. https://twitter.com/guangchuangyu/status/593443854541434882

↩

17. https://twitter.com/guangchuangyu/status/662095056610811904 and https://twitter.com/guangchuangyu/status/667337429704011777↩

18. Google Noto emoji fonts was installed in my system↩

19. https://guangchuangyu.github.io/2016/07/how-to-bug-author/ ↩

20. https://groups.google.com/d/msg/bioc-ggtree/hViM6vRZF94/MsZT8qRgBwAJ and https://github.com/GuangchuangYu/ggtree/issues/106↩

21. https://twitter.com/hadleywickham/status/600280284869697538↩

22. https://cran.r-project.org/web/packages/ggrepel/vignettes/ggrepel.html↩

23. https://groups.google.com/d/msg/bioc-ggtree/gruC4FztU8I/mwavqWCXAQAJ, https://groups.google.com/d/msg/bioc-ggtree/UoGQekWHIvw/ZswUUZKSGwAJ and https://github.com/GuangchuangYu/ggtree/issues/95↩

24. example from: https://groups.google.com/d/msg/bioc-ggtree/T2ySvqv351g/mHsyljvBCwAJ↩