## 树形图

### 进化树构建

#### 利用ETE构建、绘制和分析系统发育树

本节作者：邓子祺

单位：西班牙植物生物技术和基因组中心（Centre for Plant Biotechnology and Genomics）

版本1.0.0，更新日期：2020年8月31日

#### 简介

目前有很多生物信息学工具用于构建和可视化分析系统发育树，但此类应用很少适合需要更高自动化水平的大规模数据分析。而目前许多全基因组分析都依赖于树状数据的可视化，因此，越来越需要自动化工具来大规模处理树状结构。本教程我们为大家介绍ETE(Environment for Tree Exploration) toolkit，是一款基于python的工具包，其优势在于: 1) 一键完成“序列比对-修剪-模型测试-建树”流程, 并可根据需求进行后续遗传模型分析; 2) ETE设有多个python模块，可通过脚本对树进行自动化注释以及可视化。本教程测试数据储存在data/目录中，如修改请根据具体目录修改对应的代码 #### ETE下载与安装 推荐在Linux或MacOS系统运行ETE，对其模块兼容更友好 - 使用conda环境（推荐）

# 安装 conda 环境  
# Install Minconda (you can ignore this step if you already have Anaconda/Miniconda)  
wget http://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86\_64.sh -O Miniconda3-latest-Linux-x86\_64.sh  
bash Miniconda3-latest-Linux-x86\_64.sh -b -p ~/anaconda\_ete/  
export PATH=~/anaconda\_ete/bin:$PATH;  
  
# 安装 ETE 包  
# Install ETE   
conda install -c etetoolkit ete3  
  
# 安装 ETE 外部工具包  
conda install -c etetoolkit ete\_toolchain   
  
# Check installation  
ete3 build check

* 其他安装

1.安装依赖文件

# 在如Ubuntu, Mint，等基于Debian的系统(APT)  
sudo apt-get install python-numpy python-qt4 python-lxml python-six  
  
# 在CentOS, Fedora, 等(YUM)  
sudo yum install PyQt4.x86\_64 numpy.x86\_64 python-lxml.x86\_64 python-six.noarch  
  
# 在conda环境  
conda install -c anaconda six numpy pyqt  
  
# 基于pip安装  
pip install six numpy PyQt5

2.使用pip安装ETE3

pip install --upgrade ete3

或使用EasyInstall

easy\_install -U ete3

或通过源代码安装

wget https://files.pythonhosted.org/packages/21/17/3c49b7fafe10ed63bb7904ebf9764b98db726aa5fd482fb006818854bc04/ete3-3.1.1.tar.gz  
tar -xvf ete3-3.1.1.tar.gz  
python setup.py install

3.安装其他工具

# 此为可选择步骤，仅为ete-build和ete-evol服务。若仅使用ete3的python API模块，可跳过此步  
ete3 upgrade-external-tools

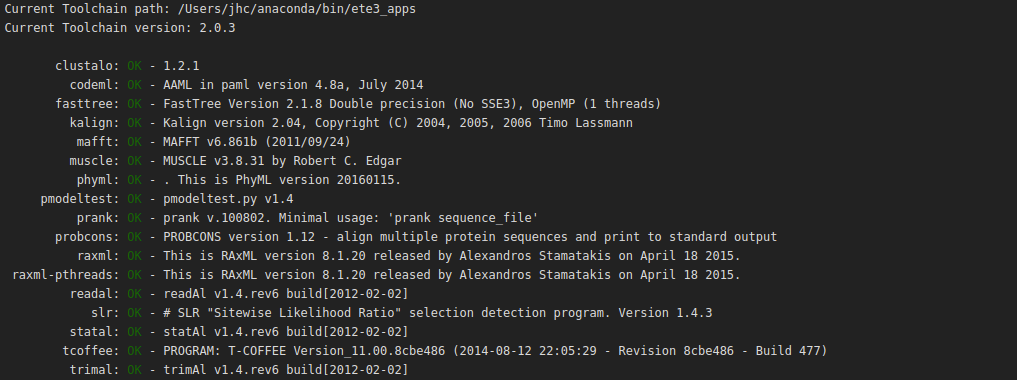
#### ETE命令行工具教程(ETE tools cookbook)

一键完成 “多序列比对 -> 修剪 -> 模型测试 -> 建树” 流程

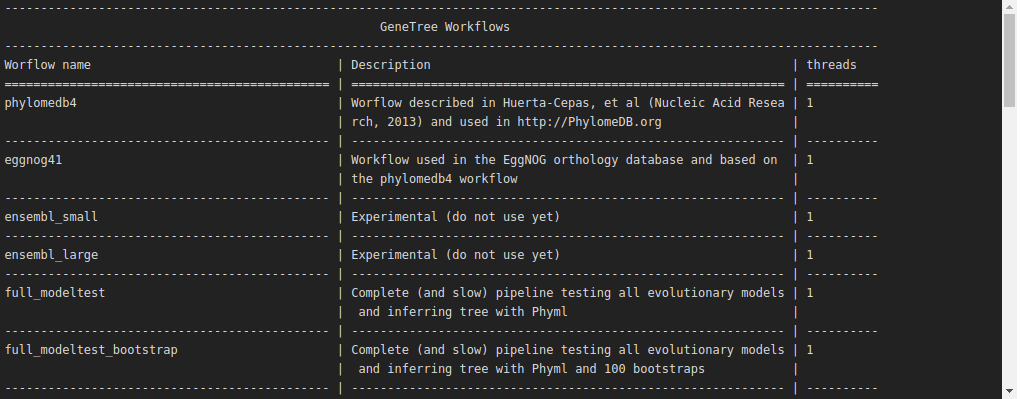
##### ete-build命令既定流程基本运行

*ete-build* 命令需要几个外部程序来进行系统发育树， 序列比对以及其他任务。建议通过conda对外部工具进行预安装(https://anaconda.org/etetoolkit/ete\_toolchain) 或（https://anaconda.org/etetoolkit/ete3\_external\_apps）

1. 检查所有外部应用程序是否可用

* ete3 build check
* 
* image

1. 查看预先系统自带的建树流程及其详细解释

* ete预先了设计了多个流程用以完成从原始序列到后续进化树构建的各个步骤。运行以下命令可以列出系统自带的流程及其详细步骤
* ete3 build workflows genetree
* 
* image

1. 准备FASTA原始序列

* 确保输入序列为正确的FASTA格式
* 输入序列可以为氨基酸或DNA序列
* 建议序列名不包含异常符号，并且序列名不能重复，否则程序报错。

这里我们以NUP62同源氨基酸序列为例子

cat data/NUP62.aa.fa | head -n15

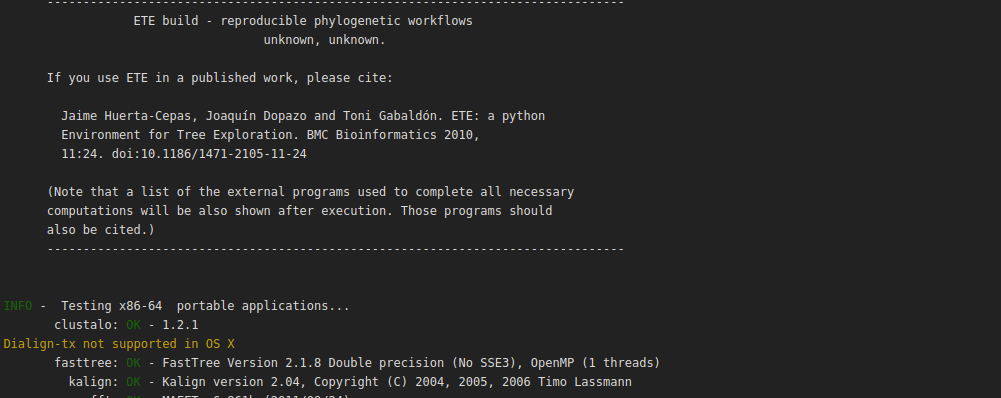


image

1. 从2)的列表中选择建树流程

基本运行只需添加三个参数 - 如果输入序列为氨基酸序列，添加 -a (若为DNA序列，则为 -n) - -o 用于定义输出文件夹（若文件夹不存在，则自动创建；否则请根据具体需求使用--resume或--clearall） - -w 用于选择执行工作流程，在这里以standard\_fasttree为例子

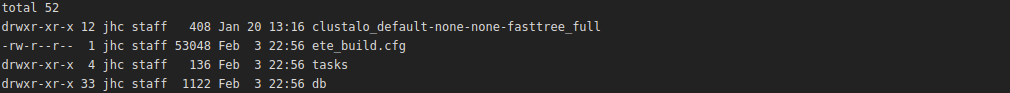
ete3 build -w standard\_fasttree -a data/NUP62.aa.fa -o NUP62\_tree/ --clearall



image

若操作正确无报错，流程结束后，可以在输出文件夹获取生成文件的列表以及对所用软件的引用。 所有结果将存储在提供的输出目录中，其结构如下：

ls -ltr NUP62\_tree/

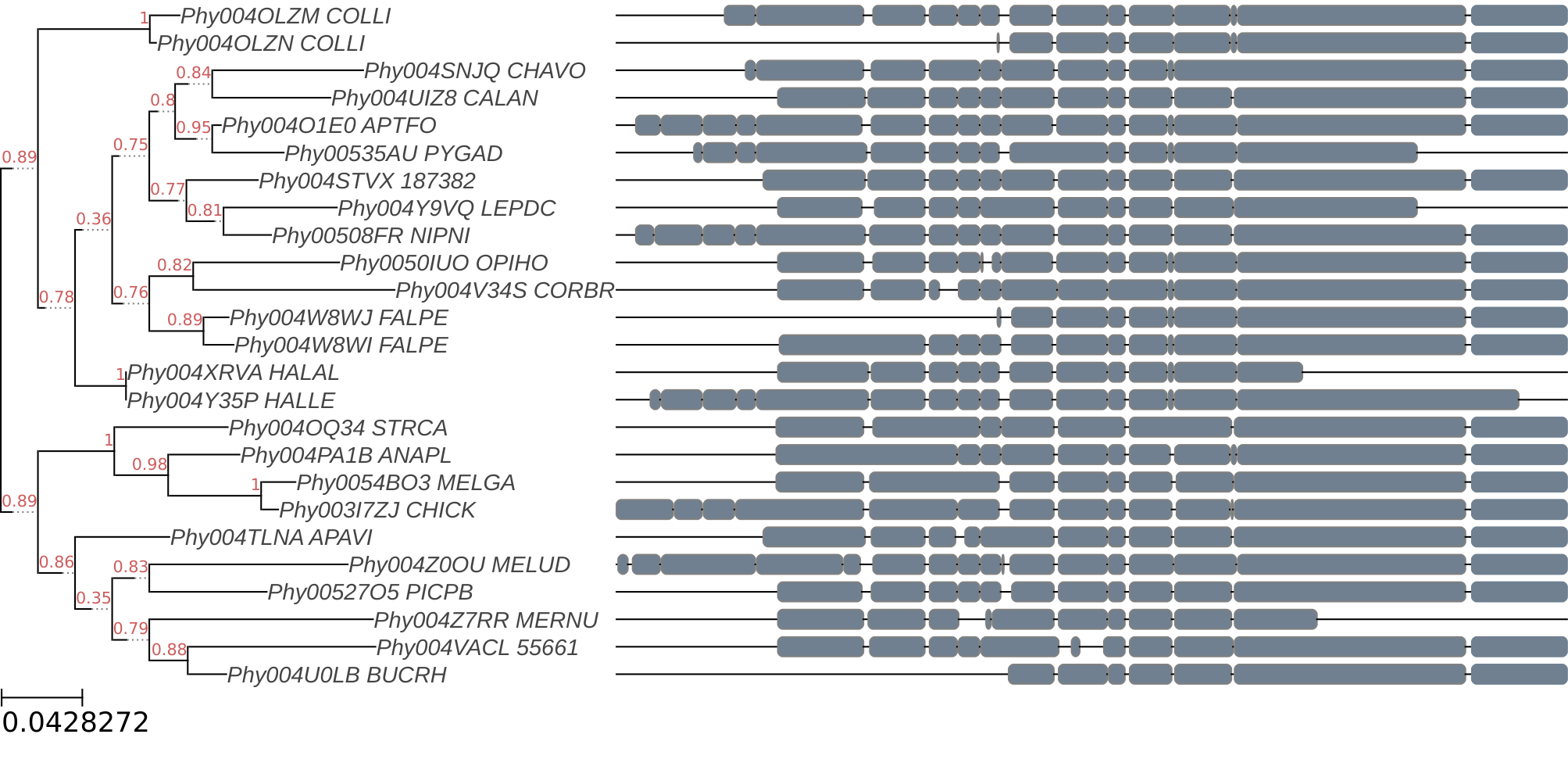


image

输出文件包括以下内容： - ete\_build.cfg 是工作流程的配置文件，你可以找到所有工作流程的具体步骤及其配置 - db/ 和 tasks/ 作为运行进程时的临时文件夹。你可以找到不同进程产生的临时文件，可用于恢复工作进程或者debug其他错误。每次运行（例如，FastTree, ClustalO, 等等）会在tasks/目录产生一个或多个临时文件夹。 - 你的目标文件存放在以工作流程具体步骤命名的目录中。在这里standard\_fasttree流程使用了Clustal Omega和Fasttree, 并且没用引用模型测试（model testing）和序列比对修剪（alignment trimming），因此目标文件夹命名为clustalo\_default-none-none-fasttree\_full

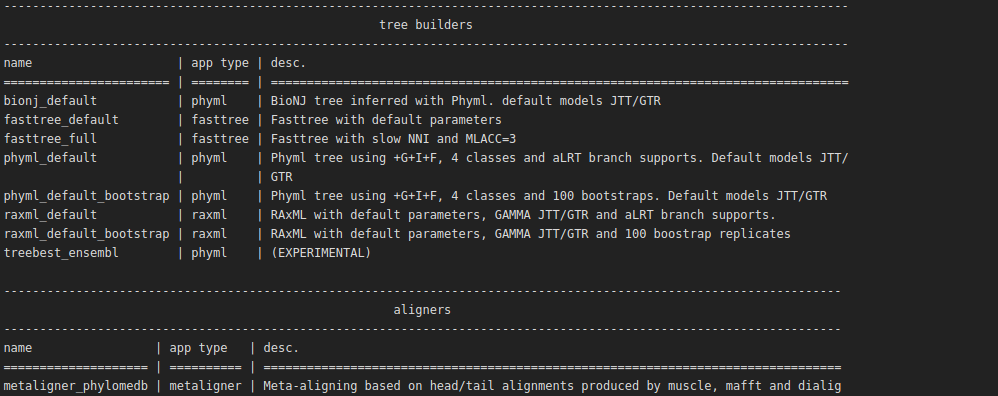
1. 查看目标树 >建树完成后你可以通过命令行在终端快速浏览树

* ete3 view --text -t NUP62\_tree/clustalo\_default-none-none-fasttree\_full/NUP62.aa.fa.final\_tree.nw
* 
* image

或直接打开浏览.png,.pdf或.svg格式图片 

##### ete-build自定义建树流程

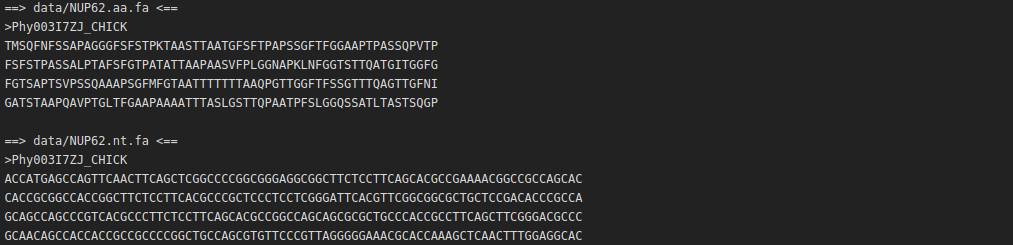
除了ete预先构建好的工作流程以外，用户也可以通过不同的工具组合自定义工作流程

* 获取可以自定义的各模块工具
* ete3 build apps
* 
* image
* 流程模块介绍
  + 建树工作流程包括四大模块，分别为aligners, alignment trimmers, model testers 以及 tree builders
  + 自定义流程遵循以下模板：aligner-trimmer-model\_tester-builder
  + 使用 - 作为模块之间的分割符
  + 使用none 表示跳过某个模块
  + 例如，自定义流程:mafft\_linsi-trimal01-pmodeltest\_soft\_ultrafast-raxml\_default
  + 实际上，ete3构建好的工作流程是实际特定模块组合的别名。 例如，standard\_fasttree 全名为 clustalo\_default-none-none-fasttree\_default
* 运行自定义流程
  + 以上述自定义流程为例子，基本运行为:
  + ete3 build -a data/NUP62.aa.fa -o custom/ --clearall -w mafft\_linsi-trimal01-pmodeltest\_soft\_ultrafast-raxml\_default
  + 自定义不同的模块细节
    - 查看已有模块的定义
    - ete3 build show phyml\_default\_bootstrap  
      [phyml\_default\_bootstrap]  
       --alpha = e  
       --bootstrap = 100  
       --nclasses = 4  
       --pinv = e  
       -f = m  
       -o = tlr  
       \_aa\_model = JTT  
       \_app = phyml  
       \_desc = Phyml tree using +G+I+F, 4 classes and 100 bootstraps. Default models JTT/GTR  
       \_nt\_model = GTR
    - 修改模块的内容 ``` ete3 build show phyml\_default\_bootstrap > customized.config #修改后 [phyml\_bootstrap\_200] --alpha = e --bootstrap = 200 --nclasses = 4 --pinv = e -f = m -o = tlr *aa*model = JTT *app = phyml* desc = Phyml tree using +G+I+F, 4 classes and 100 bootstraps. Default models JTT/GTR *nt*model = GTR

## 使用新定义的模块, 插入-c以使用自定义模块文件

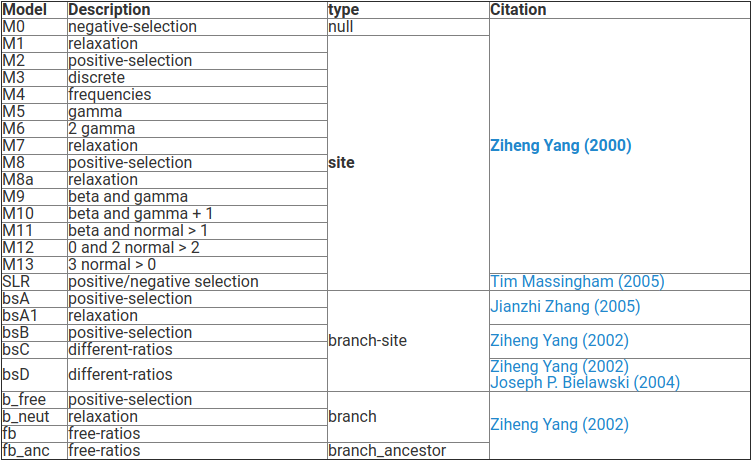
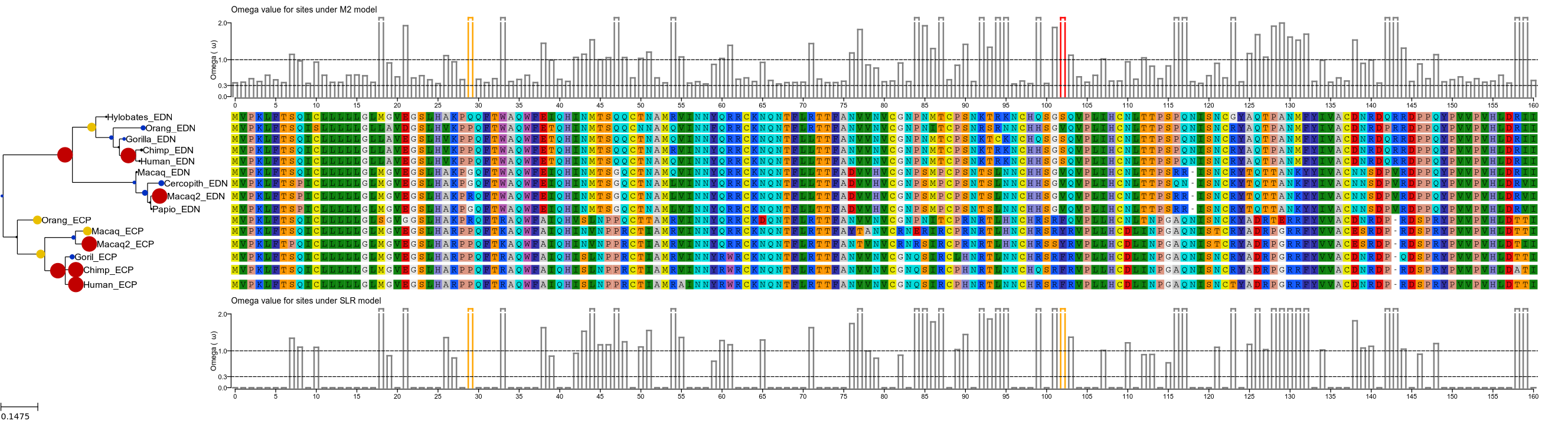
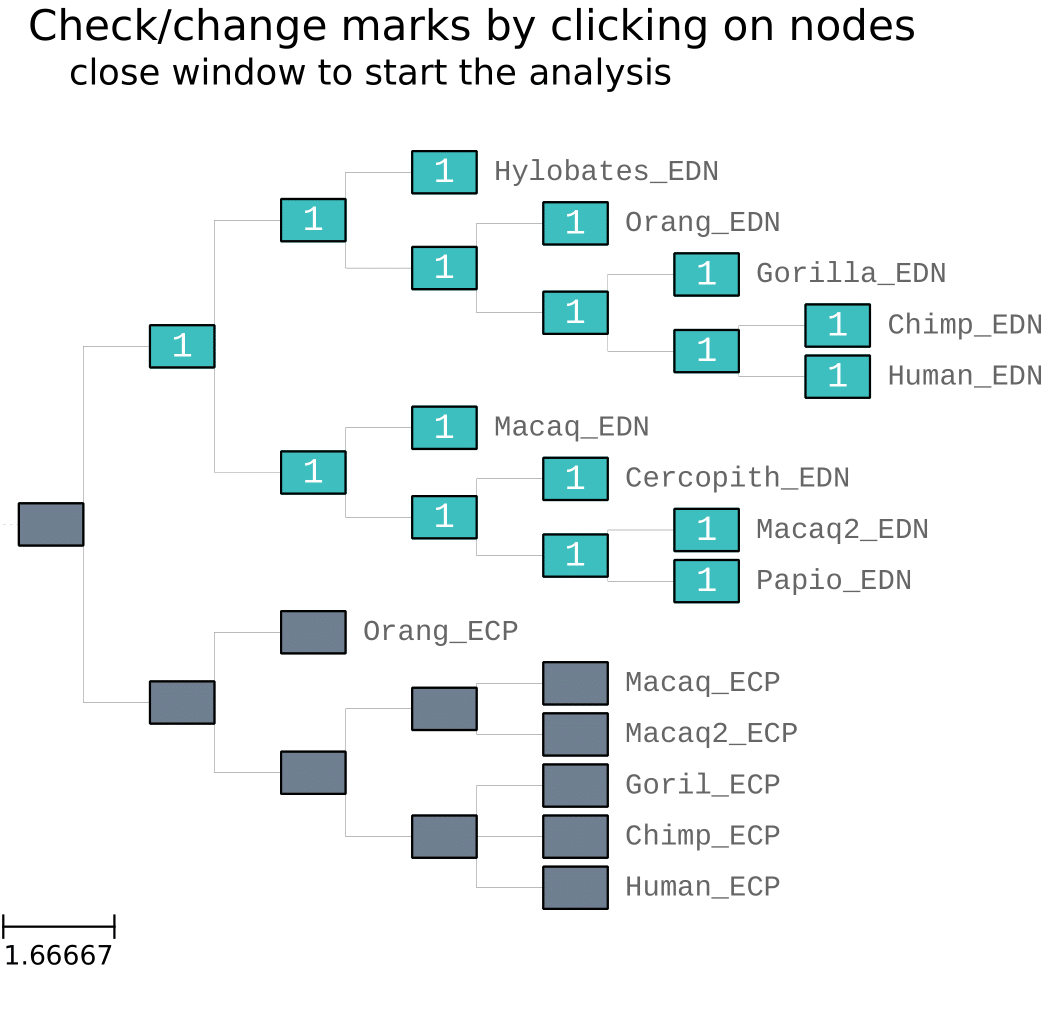
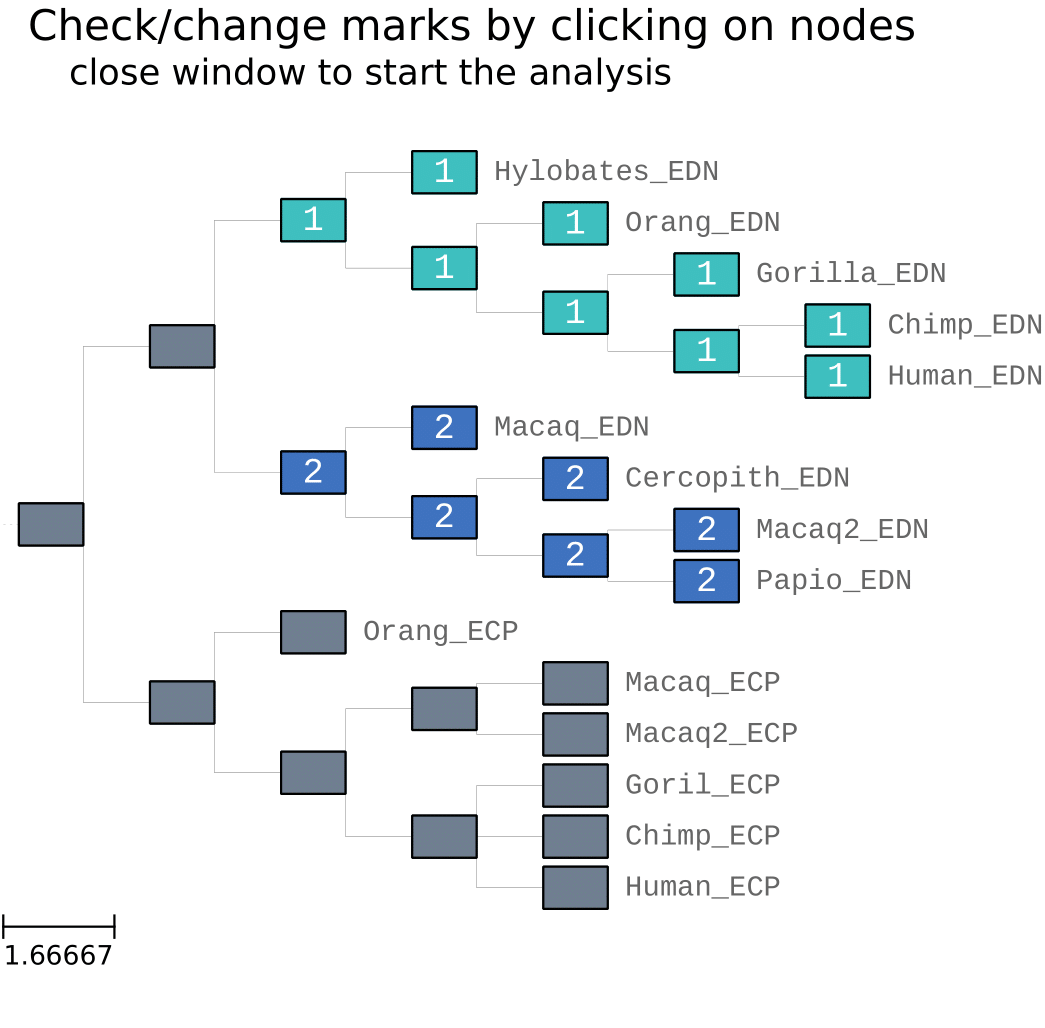
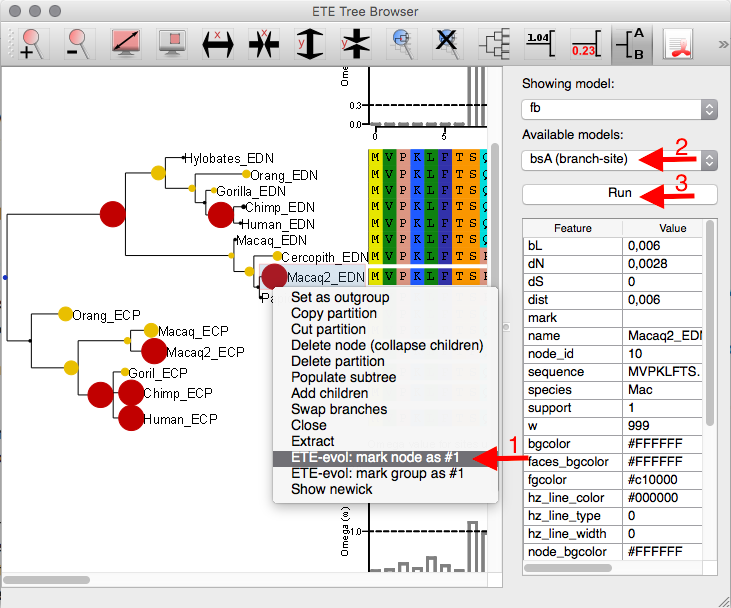
* + ete3 build -a data/NUP62.aa.fa --clearall -o phyml\_bootstrap\_200 -w mafft\_einsi-trimal\_auto-none-phyml\_bootstrap\_200 -c customized.cfg --cpu 5 ```
  + 若使用预先比对好的序列, 可利用自定义流程的格式，用none代替aligner
  + ete3 build -a data/P53.alg.fa -w none-none-pmodeltest\_soft\_ultrafast-raxml\_default -o manual\_alg --clearall

##### ete-build构建基于氨基酸序列对应核苷酸比对的进化树

* + 当氨基酸比对中的平均序列一致性高于给定的阈值，ete-build会自动将氨基酸转化为基于密码子的比对，并根据核苷酸模型推断出进化树
* 准备氨基酸和核苷酸序列FASTA文件 >注意氨基酸和核苷酸序列的名称需一一对应
* head data/NUP62.aa.fa data/NUP62.nt.fa -n5
* 
* image
* 在ete-build 命令中激活mixed模式
  + 在ete-build命令中加入序列FASTA文件（-a为氨基酸，-n为核苷酸）
  + 指定aa转换nt的阈值，作为建立基于蛋白质的系统发育树所允许的最大蛋白质序列一致性
  + ete3 build -a data/NUP62.aa.fa -n data/NUP62.nt.fa -o mixed\_types/ -w standard\_fasttree --clearall --nt-switch-threshold 0.9
  + 核苷酸序列可以含有终止密码子，最终获得的核苷酸比对序列存储在输出文件夹的\*.used\_alg.fa文件中

##### 利用ete-evol进行遗传进化分析

ete-evol 通过预先设置的遗传模型运行CodeML和Slr分析并在系统发育树相应位置生成结果图

* 简单介绍evol工作流程
  + 运行ete3 evol流程仅需要fasta文件，对应的系统发育树文件和一个模型名称。同样，如果调用branch或branch-site模型，则应传递额外的信息，以便将特定的演化约束应用于树的各个部分。
  + 可用模型名称对应于已发表的几个预定义的进化模型（可查询下一步的参照表，以及PAML和Slr的官方文档）。
  + 预定义模型中都可以通过命令行轻松修改，例如--codeml\_param getSE,1
  + 运行程序后，可以将模型与似然比检验（LRT）进行比较。亦可以默认使用预定义的比较（--tests选项）完成此操作。
* 遗传进化模型
  + 程序已预先设置好遗传进化模型，并可以同时运行多个模型
  + ete3 evol -t data/ECP\_EDN\_15.nw --alg data/ECP\_EDN\_15.fasta -o results1/ --models fb M2 SLR --cpu 3 -i ete-evol-results1-tree.png --clear\_all
  + 预设置的遗传模型一览:  在2到5分钟后，CodeML/Slr的结果文件将存储在结果文件夹resutls1/中, 结果图如下: 
* 注释进化树
  + 自动化实现多棵树的CodeML分析主要困难之一是要为每个运行的模型进行专门分支标记。ete3 evol能够实现每个所需分支模型的多个标记。例如:
  + ete3 evol -t data/ECP\_EDN\_15.nw --alg data/ECP\_EDN\_15.fasta --models b\_neut b\_free --mark Papio\_EDN,,,Orang\_EDN -o results2/ -i ete-evol-results2-tree.png --clear\_all
  + 结果如图(使用标签--interactive实现): 
  + 三个逗号表示从标记共同祖先一直到具有共同祖先物种的分支，如果只需要标记共同祖先，则只需要两个逗号
  + 使用单个逗号可轻松标记树的任何部分。 在标签--marks下，使用空格来分割不同的标签集，不同分支模型将在各组标记上运行。 例如：
  + ete3 evol -t data/ECP\_EDN\_15.nw --alg data/ECP\_EDN\_15.fasta --models b\_neut b\_free --mark Human\_EDN,,,Hylobates\_EDN,Macaq\_EDN,,,Papio\_EDN Macaq\_ECP,,Macaq2\_ECP,Human\_ECP,,Goril\_ECP -i ete-evol-results3-tree.png -o results3/ --clear\_all
  + 结果如图: 
  + 使用--view标签可以开启GUI模式对树进行交互，GUI模式能浏览所有进化模型并比较不同的进化速率。
  + ete3 evol -t data/ECP\_EDN\_15.nw --alg data/ECP\_EDN\_15.fasta -o results1/ --models fb M2 SLR --cpu 3 --clear\_all --view
  +  如图为默认加载M2, SLR和fb的GUI界面。 标记数字的箭头表示为进行branch-site模型的步骤: 1- 确认进行的分析的branch, 2- 选择运行的模型, 3- 点击Run.

#### 利用ETE进行树型分析/可视化实战

ETE提供一个全面的Python工具包(API)，可用于对系统发育树进行操作、比较、注释、可视化以及基因组规模上的常用系统发育分析

##### ETE数据结构的操作

* 读取Newick格式的树 ``` from ete3 import Tree

# 以newick格式的字符串读取树

* t = Tree("(A:1,(B:1,(E:1,D:1):0.5):0.5);")

# 以newick文件读取树

* t = Tree("data/example\_tree\_1.nw")

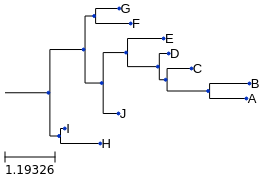
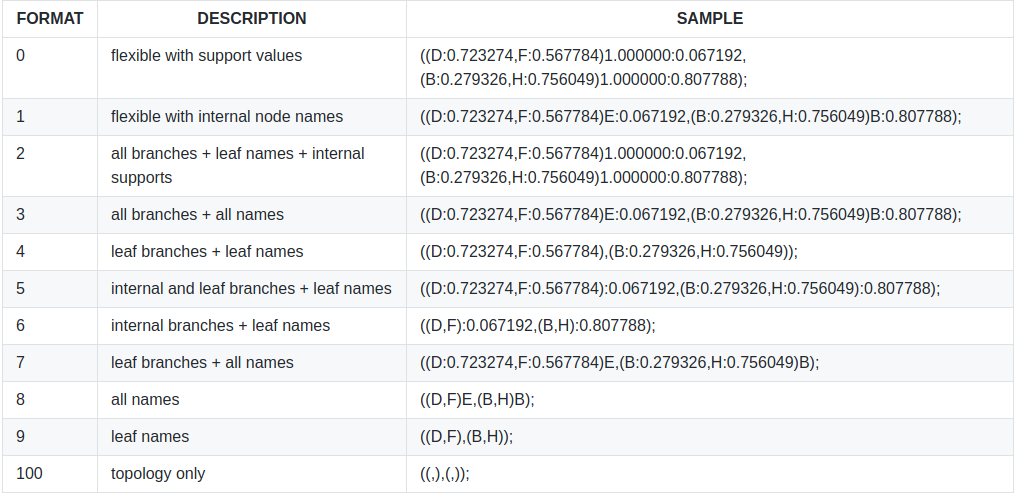
# 生成随机树, 可任意定义node的数量和名称

* t = Tree() t.populate(10,names\_library=['A','B','C','D','E','F','G','H','I','J'], random\_branches=True, reuse\_names=False)

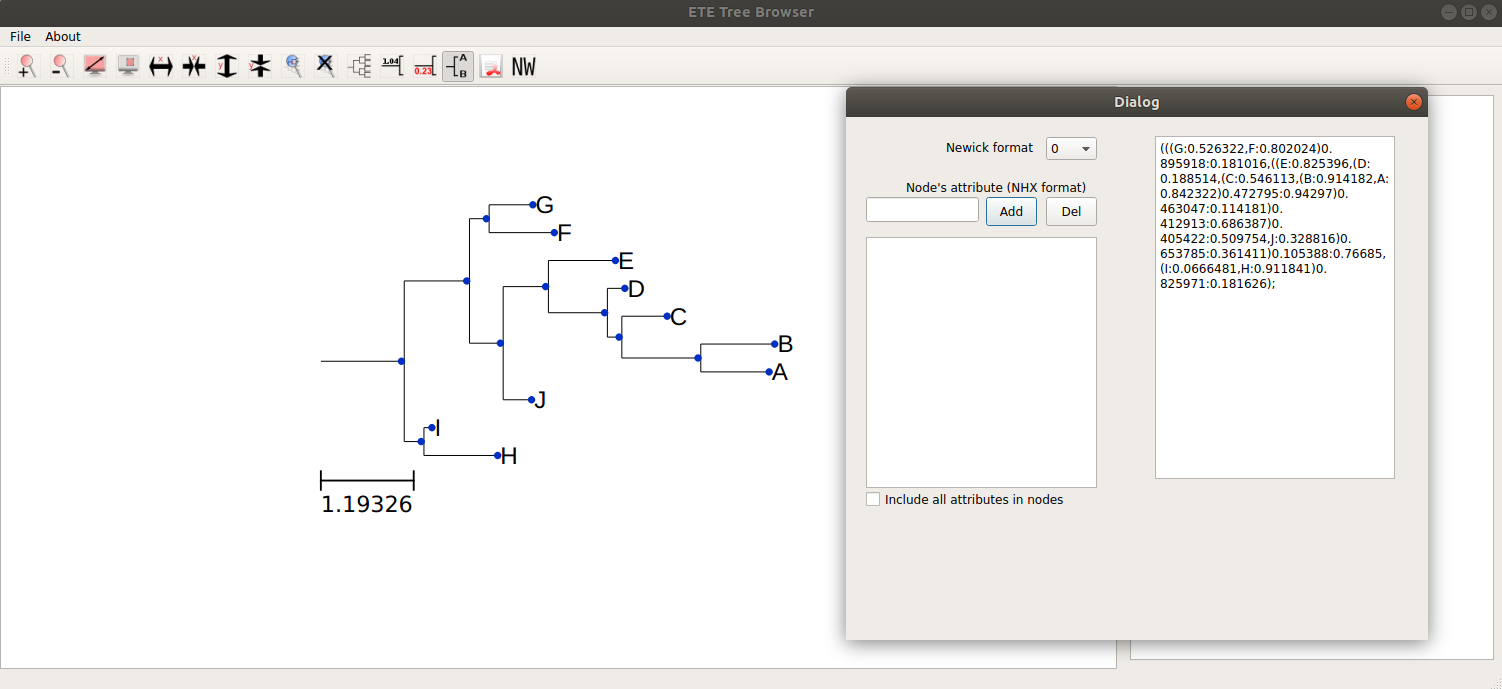
# 打印树

* print(t) #若直接打印树则只显示拓扑结构
* ![image](https://note.youdao.com/yws/api/personal/file/WEBbae4fcb3555aab0e390681c8edb8fc75?method=download&shareKey=d40488e2358095b8e90bb26ac1b2861e)

# 渲染树

* t.render('%%inline') ``` 
* 写出newick树文件
  + Tree文件能支持以下11种格式:  ``` # 默认格式fromat=0 t.write(outfile='data/example\_tree\_1.printed.nw')

# 选择输出格式=9，

* t.write(outfile='data/example\_tree\_1.printed9.nw', format=9)
* - 也可以选择打开GUI交互式进行交互式操作
* # 打开GUI t.show() ``` 
* 树结点(Node)的属性(attributes)介绍 ``` from ete3 import Tree t = Tree("data/example\_tree\_1.nw") print(t) # /-G # /-| # | -F # | # | /-E # /-| /-| # | | | | /-D # | | | -| # | | | | /-C # | -| -| # | | | /-B #--| | -| # | | -A # | | # | -J # | # | /-I # -| # -H

# 树中的所有nodes都拥有以下三个属性

* print(t.name) # '' print(t.dist) # 0.0 print(t.support) # 1.0

# 获取不同结点(node)信息

# 在载入树后，变量默认为树的根(root)

* print(t.is\_root()) # True # 获取下一级子结点 children = t.get\_children()) print(childer[0]) # /-G # /-| # | -F # | # | /-E # --| /-| # | | | /-D # | | -| # | | | /-C # -| -| # | | /-B # | -| # | -A # | # -J
* print(childer[1]) # /-I # --| # -H

# 获取再下一级结点

* ch1, ch2 = children grandchildren1 = ch1.get\_children() grandchildren2 = ch2.get\_children()

# grandchildren2[0]和grandchildren2[1]已经是最后一级的结点

* print(grandchildren2[0].is\_leaf()) # True print(grandchildren2[0]) # --I print(grandchildren2[1]) # --H
* print t.is\_leaf() print t.get\_tree\_root() print t.children[0].get\_tree\_root() print t.children[0].children[0].get\_tree\_root()

# 也可以用其他方法获取叶节点

* leaf\_I = t.get\_leaves\_by\_name("I")[0] leaf\_I = t.search\_nodes(name="I")[0]

# 获取结点的共同祖先(common ancestor), 注意:若获取单个结点的共同祖先，将返回根

* anc = t.get\_common\_ancestor("J", "B", "D") print(anc) # /-E # /-| # | | /-D # | -| # | | /-C # --| -| # | | /-B # | -| # | -A # | # -J

# 使用traverse()历遍整棵树所有结点

* num = 1 for node in t.traverse(): #只返回叶节点 if node.is\_leaf(): print(num, node.name) num +=1

# 也可以选择只历遍所有叶节点

* for leaf in t.iter\_leaves(): print(leaf) # 1 I # 2 H # 3 G # 4 F # 5 J # 6 E # 7 D # 8 C # 9 B # 10 A ```
* 对结点进行注释 ``` import random from ete3 import Tree

# 载入树

* t = Tree("data/example\_tree\_1.nw")

# 对特定的叶节点进行注释

* for leaf in t.traverse(): if leaf.name in "AEIOU": leaf.add\_features(vowel=True, confidence=random.random()) else: leaf.add\_features(vowel=False, confidence=random.random())
* print("This tree has", [leaf.name for leaf in t.iter\_leaves() if leaf.vowel==True], "vowel leaves") # This tree has ['E', 'A', 'I'] vowel leaves

# 现在我们可以利用结点的属性对其进行简单的计算，

# 例如, 在结点J，B和D的共同祖先下的结点中，找出所有支长大于0.5的集合

* ancestor = t.get\_common\_ancestor("J", "B", "D") matches = [leaf for leaf in ancestor.traverse() if leaf.dist>0.5]

# 把符合条件的信息添加到ancestor结点的属性中

* ancestor.add\_feature("long\_branch\_nodes", matches)
* print("These are nodes under ancestor with long branches", [n.name for n in ancestor.long\_branch\_nodes]) # These are nodes under ancestor with long branches ['', 'E', '', 'C', '', 'B', 'A'] ```
* 树与树进行比较 ``` # compare()函数能用于比较两棵树的拓扑结构，其中使用了不同计算距离的方法，如Robinson Foulds distance
* from ete3 import Tree t1 = Tree("data/example\_tree\_1.nw")

# 导入比较树

* t2 = Tree("data/example\_tree\_2.nw") print(t2) # /-D # /-| # | -C # /-| # | | /-B # | -| # | -A # | # --| /-J # | /-| # | | | /-I # | | -| # | | | /-H # -| -| # | -G # | # | /-F # -| # -E

# 对两树进行比较计算

* rf, max\_rf, common\_leaves, parts\_t1, parts\_t2, discard\_t1, discard\_t2 = t1.robinson\_foulds(t2)

# 输出结果为:

# RF距离

# 最大RF距离

# 两树共同拥有的叶节点

# t1所拥有t2没有的partition

# t2所拥有t1没有的partition

# t1舍弃的partition

# t2舍弃的partition

* print("RF distance is %s over a total of %s" %(rf, max\_rf)) # RF distance is 12 over a total of 16

# 在这里RF=12，即为两树之间有12个不相同的partition

* ```
* 计算进化分支(branch)的长度 ``` # 使用get\_distance()方法可计算树中两个结点之间的分支长度，该方法可以用于计算: a)两个叶节点的分支长度(通过传递两叶节点名称作为参数); b)当前结点到目标结点的分支长度（仅传递目标结点为参数）
* from ete3 import Tree t1 = Tree("data/example\_tree\_1.nw")

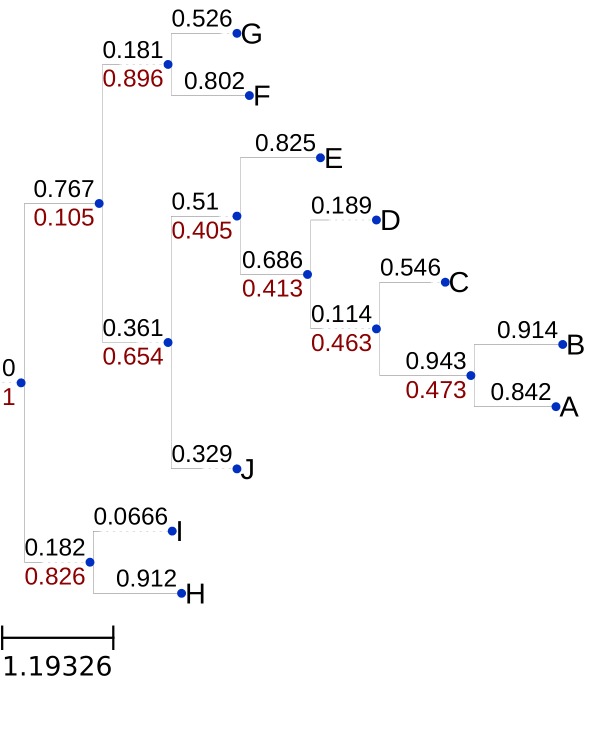
# 如下图，计算I到A的距离

* print(t.get\_distance("I","A")) # 4.4721491

# 将参数topology\_only设为True可计算两者之间的结点树（设分支长度为1）

* print(t.get\_distance("I","A", topology\_only=True)) # 8.0

# 当只输入一个叶节点，即计算叶节点到根的距离

* print(t.get\_distance("I")) # 0.2482741 ```
* 
* image
* 对树结构进行添加、删除与修剪
  + 添加 ``` from ete3 import Tree

# 通过add\_child()和add\_sister()为树添加分支或结点

* t = Tree() ch1 = t.add\_child(name="A", dist=0.9, support=70) ch2 = t.add\_child(name="B", dist=0.5, support=80) ch3 = ch2.add\_child(name="C", dist=0.3, support=10) ch4 = ch2.add\_child(name="D", dist=0.8, support=90) t.show()
* ![image](https://note.youdao.com/yws/api/personal/file/WEB65e8f799a0bf3718b5a4695c88b4c171?method=download&shareKey=340ccf5899835288f13a2bfa3193fab7)  
  - 删除

# 通过detach()方法对目标结点进行删除

* leaf\_C = t.get\_leaves\_by\_name("C")[0] removed\_node = leaf\_C.detach() print(t) # /-C # -- /-| # -D
* - 修剪

# 通过prune()方法可对指定树叶节点进行修剪

* from ete3 import Tree t = Tree("data/example\_tree\_1.nw") print(t) # /-G # /-| # | -F # | # | /-E # /-| /-| # | | | | /-D # | | | -| # | | | | /-C # | -| -| # | | | /-B #--| | -| # | | -A # | | # | -J # | # | /-I # -| # -H
* t.prune(['I','A','F'], preserve\_branch\_length=True) # 使用preserve\_branch\_length参数保留原来叶节点分支支长
* print(t) # /-F # /-| #--| -A # | # -I
* t.show() ``` 

##### ETE树的美化与注释

###### 树的基本可视化

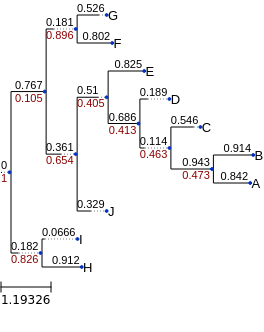
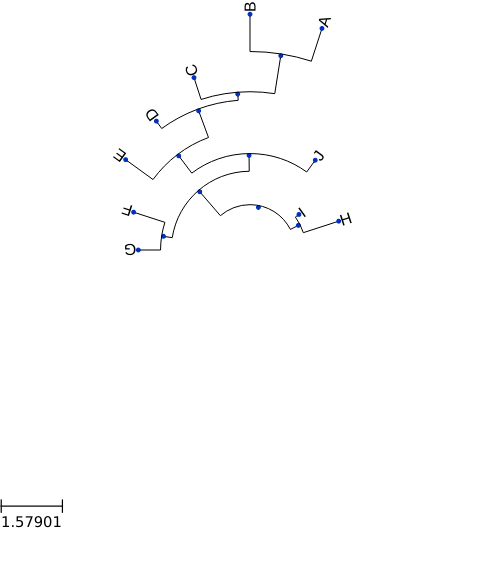
```  
#使用以下几种方法可以将树newcik文件生成多种图片文件  
from ete3 import Tree  
t = Tree("data/example\_tree\_1.nw")  
  
# 使用show()可直接对树图片文件进行交互操作  
t.show()  
  
# 使用render()将树生成png/svg.pdf等图片文件  
t.render("imgage.png")  
t.render("imgage.svg")  
t.render("imgage.pdf")  
  
# 若需要在Jupyter Notebook上渲染, 则为  
t.render("%%inline")  
```

###### 通过TreeStyle类对树图像进行美化

TreeStyle类可用于创建一组自定义选项，以控制树图像的一般外观。 可以将树样式传递给TreeNode.show()和TreeNode.render()。

* 修改用于渲染树枝的参数 ``` from ete3 import Tree, TreeStyle t = Tree("data/example\_tree\_1.nw")

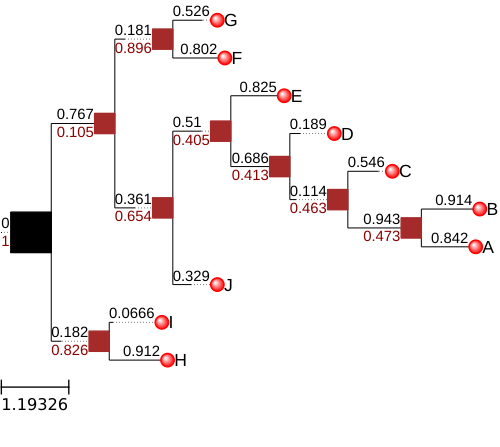
# 定义TreeStyle

* ts = TreeStyle() ts.show\_leaf\_name = True ts.show\_branch\_length = True ts.show\_branch\_support = True t.render(file\_name="%%inline", tree\_style=ts) ``` 
* 修改树型 ``` from ete3 import Tree, TreeStyle t = Tree("data/example\_tree\_1.nw")
* ts = TreeStyle() ts.mode = "c" # 将树图改为圈图 ts.arc\_start = -180 ts.arc\_span = 180 t.render(file\_name="%%inline", w=500, tree\_style=ts) ``` 

###### 通过NodeStyle()对树结点进行注释

* 类似TreeStyle类，NodeStyle是针对结点外观修饰的类
* 需要注意的是，上一章节中对树的注释是基于其树结点（TreeNode类）的数据结构，而通过TreeStyle对树注释则是对通过对树图像进行渲染。

# 用之前例的树示范   
from ete3 import Tree, NodeStyle, TreeStyle  
t = Tree("data/example\_tree\_1.nw")  
  
# 定义基本TreeStyle  
ts = TreeStyle()  
ts.show\_leaf\_name = True  
ts.show\_branch\_length = True  
ts.show\_branch\_support = True  
  
# 把叶节点渲染成直径为10p的红色小球，非叶结点为棕色的矩形  
for n in t.traverse():   
 # 历遍所有结点并输入对应NodeStyle的属性  
 if n.is\_leaf(): # Decide if leaf node  
 nstyle = NodeStyle()  
 nstyle["shape"] = "sphere"  
 nstyle["size"] = 10  
 nstyle["fgcolor"] = "red"  
 n.set\_style(nstyle) # 最后使用set\_style()将NodeStyle类属性传递到结点中  
 else:  
 nstyle = NodeStyle()  
 nstyle["shape"] = "square"  
 nstyle["size"] = 15  
 nstyle["fgcolor"] = "brown"  
 n.set\_style(nstyle)   
  
# 渲染根结点的作为图片的属性  
t.img\_style["size"] = 30  
t.img\_style["fgcolor"] = "black"  
   
t.render(file\_name="%%inline", w=500, tree\_style=ts)

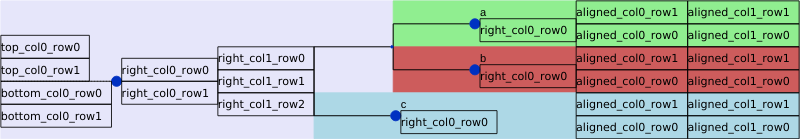


image

###### 通过Node faces属性进一步注释结点信息

Nodes faces可看作树结点需要添加额外附加的信息，可以为文本(TextFace)、几何形状(CircleFace)、直至分子序列注释(SequenceFace)、热图(ProfileFace)。更详细的信息可参考ete3的官方文档

# 参考以下例子  
from ete3 import Tree, TextFace, NodeStyle, TreeStyle  
  
t = Tree("((a,b),c);")  
  
# 定义TextFace的文本信息  
right\_c0\_r0 = TextFace("right\_col0\_row0")  
right\_c0\_r1 = TextFace("right\_col0\_row1")  
right\_c1\_r0 = TextFace("right\_col1\_row0")  
right\_c1\_r1 = TextFace("right\_col1\_row1")  
right\_c1\_r2 = TextFace("right\_col1\_row2")  
  
top\_c0\_r0 = TextFace("top\_col0\_row0")  
top\_c0\_r1 = TextFace("top\_col0\_row1")  
  
bottom\_c0\_r0 = TextFace("bottom\_col0\_row0")  
bottom\_c0\_r1 = TextFace("bottom\_col0\_row1")  
  
aligned\_c0\_r0 = TextFace("aligned\_col0\_row0")  
aligned\_c0\_r1 = TextFace("aligned\_col0\_row1")  
  
aligned\_c1\_r0 = TextFace("aligned\_col1\_row0")  
aligned\_c1\_r1 = TextFace("aligned\_col1\_row1")  
  
all\_faces = [right\_c0\_r0, right\_c0\_r1, right\_c1\_r0, right\_c1\_r1, right\_c1\_r2,   
 top\_c0\_r0, top\_c0\_r1,   
 bottom\_c0\_r0, bottom\_c0\_r1,   
 aligned\_c0\_r0, aligned\_c0\_r1,  
 aligned\_c1\_r0, aligned\_c1\_r1]  
  
# 修改Face的属性，添加边界  
for f in all\_faces:  
 f.border.width = 1  
 f.margin\_bottom = 5  
 f.margin\_top = 5  
 f.margin\_right = 10  
  
# 为树根结点添加定义好的Textface，并定义好方位  
t.add\_face(right\_c0\_r0, column=0, position="branch-right")  
t.add\_face(right\_c0\_r1, column=0, position="branch-right")  
  
t.add\_face(right\_c1\_r0, column=1, position="branch-right")  
t.add\_face(right\_c1\_r1, column=1, position="branch-right")  
t.add\_face(right\_c1\_r2, column=1, position="branch-right")  
  
t.add\_face(top\_c0\_r0, column=0, position="branch-top")  
t.add\_face(top\_c0\_r1, column=0, position="branch-top")  
  
t.add\_face(bottom\_c0\_r0, column=0, position="branch-bottom")  
t.add\_face(bottom\_c0\_r1, column=0, position="branch-bottom")  
  
# 修改叶节点的结点的背景颜色  
a = t.get\_leaves\_by\_name("a")[0]  
a.set\_style(NodeStyle())  
a.img\_style["bgcolor"] = "lightgreen"  
  
b = t.get\_leaves\_by\_name("b")[0]  
b.set\_style(NodeStyle())  
b.img\_style["bgcolor"] = "indianred"  
  
c = t.get\_leaves\_by\_name("c")[0]  
c.set\_style(NodeStyle())  
c.img\_style["bgcolor"] = "lightblue"  
  
t.set\_style(NodeStyle())  
t.img\_style["bgcolor"] = "lavender"  
t.img\_style["size"] = 12  
  
# 为各叶节点添加其对应的TextFace注释，并要求对齐  
for leaf in t.iter\_leaves():  
 leaf.img\_style["size"] = 12  
 leaf.add\_face(right\_c0\_r0, 0, "branch-right")  
 leaf.add\_face(aligned\_c0\_r1, 0, "aligned")  
 leaf.add\_face(aligned\_c0\_r0, 0, "aligned")  
 leaf.add\_face(aligned\_c1\_r1, 1, "aligned")  
 leaf.add\_face(aligned\_c1\_r0, 1, "aligned")  
  
# 输出树图像  
ts = TreeStyle()  
ts.show\_scale = False  
t.render("face\_positions.png", w=800, tree\_style=ts)

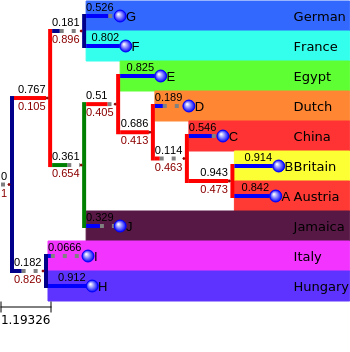


image

###### 设计和运用layout布局函数

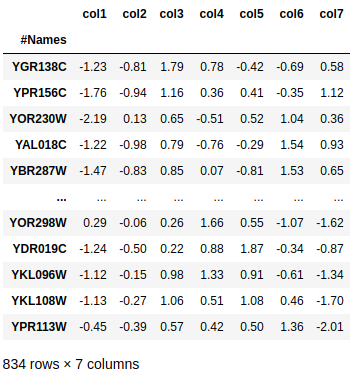
我们可以通过设计布局函数来对树进行特定条件的绘制

# 以example\_tree\_1为例，我们将定制布局函数，实现自动化注释:  
# 1）修改叶节点名称  
# 2）用不同颜色标记不同范围的bootstrap值分支  
  
from ete3 import Tree, NodeStyle, TreeStyle  
t = Tree("data/example\_tree\_1.nw")  
  
# 准备叶节点对应名称  
namemap = {'A': 'Austria',   
 'B': 'Britain',   
 'C':'China',   
 'D':'Dutch',   
 'E':'Egypt',  
 'F':'France',  
 'G':'German',  
 'H':'Hungary',  
 'I':'Italy',  
 'J':'Jamaica'}  
  
# 准备叶节点对应颜色  
colormap = { 'Austria': '#FF3933',  
 'Britain': '#FCFF33',  
 'China': '#FF3333',  
 'Dutch': '#FF8633',  
 'Egypt': '#5EFF33',  
 'France': '#33FFEC',  
 'German': '#3368FF',  
 'Hungary':'#5E33FF',  
 'Italy': '#F333FF',  
 'Jamaica':'#581845'}  
  
# 设计布局函数，自动化注释树结点的外观  
def mylayout(node):  
 # 更改结点分支外貌特征  
 node.img\_style['vt\_line\_width'] = 4  
 node.img\_style['hz\_line\_width'] = 4  
 node.img\_style['vt\_line\_type'] = 0  
 node.img\_style['hz\_line\_type'] = 0  
   
 # 设置结点自展值阈值以及相应的颜色  
 if node.support > 0.8:  
 node.img\_style["hz\_line\_color"] = 'darkblue' # change horizontal branch color   
 node.img\_style["vt\_line\_color"] = 'darkblue' # Change vertical branch color  
 elif 0.5 < node.support < 0.8:  
 node.img\_style["hz\_line\_color"] = 'green' # change horizontal branch color   
 node.img\_style["vt\_line\_color"] = 'green' # Change vertical branch color  
 else:  
 node.img\_style["hz\_line\_color"] = 'red' # change horizontal branch color   
 node.img\_style["vt\_line\_color"] = 'red' # Change vertical branch color  
   
   
 if node.is\_leaf():  
 # 添加叶节点信息  
 new\_name = namemap[node.name]  
 new\_nameFace = faces.TextFace(new\_name)  
 faces.add\_face\_to\_node(new\_nameFace, node, column=1, aligned=True)  
  
 # 设置叶节点外貌  
 node.img\_style["size"] = 12  
 node.img\_style["shape"] = "sphere"  
 node.img\_style["fgcolor"] = "blue"  
 node.img\_style["bgcolor"] = colormap[new\_name]  
   
 node.img\_style["hz\_line\_color"] = 'blue' # change branch color   
   
 # 设置其他非叶节点的节点外貌  
 else:  
 node.img\_style["size"] = 2  
 node.img\_style["shape"] = "circle"  
 node.img\_style["fgcolor"] = "darkred"  
  
# 基本树style设置  
ts = TreeStyle()  
ts.show\_leaf\_name = True  
ts.show\_branch\_length = True  
ts.show\_branch\_support = True  
# 指定布局函数到TreeStyle类中  
ts.layout\_fn = mylayout   
  
t.img\_style["size"] = 30  
t.img\_style["fgcolor"] = "black"  
t.render(file\_name="ete-api-layout.png",tree\_style=ts)



image

###### 综合实战: 系统发育树+热图+气泡效果（自定义叶节点名称、外观）

* 读入矩阵文件，矩阵文件格式需满足:
  + 矩阵为TAB键分割，第一行是每列的名字
  + 矩阵每一行第一列为行名字，与树的节点对应
  + 矩阵可以存储在一个文件中，也可以是字符串
  + import pandas as pd  
    import numpy as np  
    PATH = "./data/"  
    data = pd.read\_table(PATH+"diauxic.array", header=0, index\_col=0)  
    data.index.name = "#Names" #修改第一行的名字使其符合ETE的要求  
    data\_mat = data.to\_csv(None, sep="\t", float\_format="%.2f") #将数据表转化成字符串  
    header = list(data.columns.values) #获取列的名字用于标记  
    data
  + 
  + image
* print(data\_mat[:150]) # 取一部分查看格式转化为正确
* '#Names\tcol1\tcol2\tcol3\tcol4\tcol5\tcol6\tcol7\nYGR138C\t-1.23\t-0.81\t1.79\t0.78\t-0.42\t-0.69\t0.58\nYPR156C\t-1.76\t-0.94\t1.16\t0.36\t0.41\t-0.35\t1.12\nYOR230W\t-2.19\t0'
* header
* ['col1', 'col2', 'col3', 'col4', 'col5', 'col6', 'col7']
* 导入树与矩阵 ``` # 导入必需modules，其中ClusterTree是加载热图的module from ete3 import ClusterTree, TreeStyle, AttrFace, ProfileFace, TextFace from ete3.treeview.faces import add\_face\_to\_node

# 文件路径

* PATH = "./data/"

# 导入树和上一步转化好格式的矩阵

* t = ClusterTree(PATH+"diauxic.nw", data\_mat)

# 矩阵文件已经负载到树的根结点属性中

* array = t.arraytable

# 一般化矩阵数据以创建热图颜色梯度

* matrix\_dist = [i for r in range(len(array.matrix))  
   for i in array.matrix[r] if np.isfinite(i)] matrix\_max = np.max(matrix\_dist) matrix\_min = np.min(matrix\_dist) matrix\_avg = matrix\_min+((matrix\_max-matrix\_min)/2) ```
* 设计布局 ``` # 为热图设计一个有8列的AttrFace, 作为叶节点的一个属性 nameFace = AttrFace("name", fsize=8)

# 设计布局函数

* def mylayout(node): profileFace = ProfileFace(matrix\_max, matrix\_min, matrix\_avg, 200, 14, "heatmap") cbarsFace = ProfileFace(matrix\_max,matrix\_min,matrix\_avg, 200, 70,"cbars")
* # 如果结点为叶节点，加入热图  
  if node.is\_leaf():  
   # And a line profile  
   add\_face\_to\_node(profileFace, node, 0, aligned=True)  
   node.img\_style["size"]=0  
   add\_face\_to\_node(nameFace, node, 1, aligned=True)  
    
  # 如果结点为中间结点，根据其silhouette值创建气泡效果  
  else:  
   # silhouette值大于0，为绿色气泡  
   if node.silhouette>0:  
   validationFace = TextFace("Silh=%0.2f" %node.silhouette,  
   "Verdana", 10, "#056600")  
   node.img\_style["fgcolor"]="#056600"  
   # 否则为红色  
   else:  
   validationFace = TextFace("Silh=%0.2f" %node.silhouette,  
   "Verdana", 10, "#940000")  
   node.img\_style["fgcolor"]="#940000"  
    
   # 结点大小与silhouette成正比.  
   node.img\_style["shape"]="sphere"  
   if node.silhouette<=1 and node.silhouette>=-1:  
   node.img\_style["size"]= 15+int((abs(node.silhouette)\*10)\*\*2)  
    
   # 如果结点是内部的，绘制一个带有分区平均表达式的条形图  
   add\_face\_to\_node(validationFace, node, 0)  
   if len(node)>100:  
   add\_face\_to\_node(cbarsFace, node, 1)

# 出图

* ts = TreeStyle() ts.layout\_fn = mylayout t.render("ete-cluster-final-result.pdf", tree\_style=ts) ```

#### 扩展阅读

* [ETE Toolkit 官方文档](http://etetoolkit.org/)
* [19th European Conference on Computational Biology (ECCB2020) ETE Toolkit 教程](https://github.com/etetoolkit/course)

#### 参考文献

* ETE 3: Reconstruction, analysis and visualization of phylogenomic data. Jaime Huerta-Cepas, Francois Serra and Peer Bork. Mol Biol Evol 2016; doi: 10.1093/molbev/msw046