5-Phylogeny

April 21, 2023

1 How to generate and plot a phylogenetic tree

1.1 Alignments

1.1.1 Pairwise alignments

First, we will learn how to make **pairwise alignments**. We will align the ftsZ protein sequence that was generated in the previous lecture to the ftsA gene. As in the previous case, the first step is to load the necessary modules.

[1]:
```
from Bio import SeqIO
from Bio.Align import PairwiseAligner
```

[2]:
```
#We extract the sequence from ftsZ...
with open("inputs/tutorial5/ftsZ.faa") as prot:
    record = SeqIO.read(prot, "fasta")
    ftsZ = record.seq

#..and do the same for ftsA
with open("inputs/tutorial5/ftsA.faa") as prot2:
    record = SeqIO.read(prot2, "fasta")
    ftsA = record.seq
```

Now, we are ready to perform the pairwise alignment:

[3]:
```
aligner = PairwiseAligner()
alignment = aligner.align(ftsZ[:70], ftsA[:70]) #Taking only the first 70␣
 ↪positions
print(f"{len(alignment)} alignments were generated.")
```

```
7975307861509700880 alignments were generated.
```

We can print the first alignment out of the many that we generated:

[4]:
```
print(alignment[0])
```

```
target            0 MFEPMEL---TNDAVI-K-VIGVGGGGGNAV-EHMVRERIEGVEFF-A-VNTDAQALRKT
                  0 |---------|-|----|-|--||--------|------|-|-----|-|---|-||---
query             0 M------IKAT-D---RKLV--VG-------LE------I-G----TAKV---A-AL---

target           52 AVGQT---------IQIG--S----GIT--KGLG-------------A------  70
```

```
                60 -||----------|-||--|----|----||-|-------------|------ 114
query           23 -VG--EVLPDGMVNI-IGVGSCPSRG--MDKG-GVNDLESVVKCVQRAIDQAEL  70
```

1.1.2 Multiple sequence alignments

If we want to make a phylogenetic tree, we need a **multiple sequence alignment (MSA)** as an input. Unfortunately, Biopython doesn't have a module to generate MSAs, but it does have a module to read MSAs. We will use a pre-generated alignment of ftsZ gene across different bacteria, stored in the file `MSA.faa`. The MSA was generated by running Mafft.

[5]:
```
from Bio import AlignIO
from Bio.Align import MultipleSeqAlignment
```

We can easily read the alignment with the `read` function in the `AlignI` module:

[6]:
```
align = AlignIO.read("inputs/tutorial5/MSA.faa", "fasta") #Read the alignment
print(align[:5, :50]) #Print the first 5 rows and 50 first positions
```

```
Alignment with 5 rows and 50 columns
MT--------------------------------IQLQKPDITEL----- AAC45821.1
MA--------------------------------INLQKPDITEL----- AAC45824.1
MT--------------------------------INLQKPDITEL----- CDL76811.1
MS------------------------------------NNQNY----- CAN01912.1
MLASSSLTSKRSASASSSASRVPHVRAAPVPQRAVTPQPATSTSYPSQPV BAB91150.1
```

Format conversion is rather easy, for example, here I convert the first two records to Phylip format:

[7]:
```
print(format(align[:2], "phylip"))
```

```
 2 739
AAC45821.1 MT-------- ---------- ---------- ----IQLQKP DITEL-----
AAC45824.1 MA-------- ---------- ---------- ----INLQKP DITEL-----

           ---------- ---------- ---------- KPRITVFGVG GGGGNAVNNM
           ---------- ---------- ---------- KPRITVFGVG GGGGNAVNNM

           ITVGLQGVDF VVANTDAQAL TMTKADRVIQ LGVNVTEGLG AGSQPEVGRA
           ITAGLQGVDF VVANTDAQAL TMTKAERIIQ MGVAVTEGLG AGSQPEVGRA

           AAEECIDEII DHLNGTH--M CFVTAGMGGG TGTGAAPVVA QAARNKGILT
           AAEECIDEII DHLQGTH--M CFVTAGMGGG TGTGAAPIVA QAARNKGILT

           VGVVTKPFHF EGGRRMRLAE QGIEELQKSV DTLIVIPNQN LFRIANDKTT
           VGVVTKPFHF EGGRRMRIAD QGISDLQKSV DTLIVIPNQN LFRIANDKTT

           FADAFAMADQ VLYSGVACIT DLMVKEG-LI NLDFADVRSV MREMARPMMG
           FADAFAMADQ VLYSGVACIT DLMVKEG-LI NLDFADVRSV MREMGRAMMG

           TGEAS----- --GPARAMQA AEAAIANPLL DETSMKGAQG LLISITGGRD
```

```
TGEAS----- --GEGRAMAA AEAAIANPLL DETSMKGAQG LLISITGGRD

--LTLFEVDE AATRIREEVD PDANIILGAT FDEAL-EGLI RVSVVATGID
--LTLFEVDE AATRIREEVD PDANIILGAT FDEEL-EGLI RVSVVATGID

RVAGIGEQNI AEMRA----- ---------- ------AAAK PLIRPSAAVA
RTAAEVAGRS ADFRP----- ---------- ------VAPK PIVRPSAAV-

PAPAAVQPAH AVSQAP---- -------KTV DQIAQTIRSA EAEMERELGF
--PAQPQPTV SLQPVPQPQP VQQPLQQQNV DHIALAIR-- EAEMERELDI

AA---HQQPS ---------Q DFRP------ ---QSKLFAS -SPA--EAPA
AARAQVAAPA PQPQPHLQEE AFRP------ ---QSKLFAG VAPT--EAAP

ALR-PAQPVQ QAAPAPVAQA PVYHAPEQVA VPAP-RMQQA QAPVYQEPAP
VMR-PAQPAP R--------- -------PVE MQAPVQPQMQ AQPVQQEPTQ

VGR-QPEPV- RMPKVEDFPP VVKAEMDHRD RA-TPVAQEE RGPMGLLKRI
VVRQQAEPV- RMPKVEDFPP VVKAEMDYRT QP-APAHQEE RGPMGLLNRI

TNSLGRREEE --EVPSDMMD A----PSMAP -QRRAPLSPE ASLYAPRRGQ
TSSLGLRERE ATNVSSDMTA AA---PSAAS -QQRRPLSPE ASLYAPRRGQ

LDDHGRATPS SSSHHDDDQL EIPAFLRRQS N--------
LDDHGRAAPQ MRS-HEDDQL EIPAFLRRQS S--------
```

[8]:
```
#We can take a look at the sequences in the alignment
for record in align: #Loop through records
    print(record.description) #Print the descriptions
```

```
AAC45821.1 FtsZ [Agrobacterium tumefaciens]
AAC45824.1 FtsZ [Sinorhizobium meliloti]
CDL76811.1 ftsZ [Brucella canis str. Oliveri]
CAN01912.1 ftsZ [Clavibacter michiganensis subsp. michiganensis NCPPB 382]
BAB91150.1 FtsZ [Chlamydomonas reinhardtii]
AAB18965.1 FtsZ [Neisseria gonorrhoeae]
AAA16512.1 FtsZ [Staphylococcus aureus]
AAC24604.1 FtsZ [Thermotoga maritima]
AAA56889.1 FtsZ [Streptomyces griseus]
AAA26281.1 ftsZ [Sinorhizobium meliloti]
ABQ96888.1 FtsZ [Bacillus subtilis subsp. spizizenii ATCC 6633 = JCM 2499]
AAD10533.1 FtsZ [Streptomyces coelicolor A3(2)]
CSB13334.1 FtsZ [Vibrio cholerae]
CAD00110.1 ftsZ [Listeria monocytogenes EGD-e]
CAC97368.1 ftsZ [Listeria innocua Clip11262]
AAA85622.1 FtsZ [Borreliella burgdorferi]
AAA95993.2 FtsZ [Pseudomonas aeruginosa PAO1]
```

```
BAA28179.1 FtsZ [Porphyromonas gingivalis]
CAD6022189.1 ftsZ [Escherichia coli]
```

1.2 Phylogenetic trees

The Phylo module in Biopython has tools for tree-building.

[9]:
```
import matplotlib.pyplot as plt
from Bio import Phylo
from Bio.Phylo.TreeConstruction import DistanceCalculator,␣
↪DistanceTreeConstructor
```

Here, we will first generate a tree file and then try to parse it. The first step to generating a tree is calculating the distances between sequences. We will use the Blosum62 distance matrix.

[10]:
```
calculator = DistanceCalculator("blosum62") #Create a calculator that uses the␣
↪Blosum62 distance matrix
distances = calculator.get_distance(align) #Apply the calculator on the MSA
print(distances)
```

```
AAC45821.1  0.000000
AAC45824.1  0.196037    0.000000
CDL76811.1  0.275825    0.269802    0.000000
CAN01912.1  0.548234    0.550079    0.539683    0.000000
BAB91150.1  0.581837    0.563703    0.564278    0.507065    0.000000
AAB18965.1  0.600631    0.626299    0.622523    0.560482    0.579490    0.000000
AAA16512.1  0.578669    0.569320    0.581273    0.447311    0.498353    0.551427
0.000000
AAC24604.1  0.604460    0.596831    0.596831    0.530377    0.568696    0.570410
0.514863    0.000000
AAA56889.1  0.549563    0.554816    0.531882    0.256013    0.503366    0.563281
0.445953    0.516364    0.000000
AAA26281.1  0.291183    0.293333    0.271363    0.519473    0.572180    0.548715
0.540459    0.574906    0.513142    0.000000
ABQ96888.1  0.580457    0.556908    0.561673    0.433574    0.491247    0.531351
0.285409    0.506595    0.408691    0.533535    0.000000
AAD10533.1  0.565989    0.572248    0.535658    0.242655    0.506596    0.543094
0.439536    0.515758    0.092564    0.506571    0.395414    0.000000
CSB13334.1  0.583546    0.589848    0.596144    0.541960    0.541644    0.530494
0.548884    0.535971    0.532620    0.556609    0.535752    0.528548    0.000000
CAD00110.1  0.567413    0.554696    0.563890    0.449566    0.489213    0.537771
0.282620    0.482614    0.437232    0.512836    0.231792    0.423581    0.533960
0.000000
CAC97368.1  0.568831    0.554068    0.563758    0.456463    0.489213    0.537242
0.286089    0.479017    0.434457    0.511642    0.226096    0.424837    0.535306
0.015979    0.000000
AAA85622.1  0.556663    0.556936    0.554560    0.532633    0.583159    0.564103
0.495491    0.548575    0.548335    0.517815    0.479526    0.535578    0.550933
0.480021    0.477357    0.000000
```

```
AAA95993.2  0.574665    0.572816    0.588656    0.566971    0.535097    0.488654
0.526709    0.560552    0.573639    0.553719    0.505365    0.563755    0.398361
0.516754    0.515183    0.536651    0.000000
BAA28179.1  0.691736    0.687793    0.700887    0.657556    0.653968    0.690614
0.655858    0.620528    0.658355    0.674528    0.659540    0.655358    0.642343
0.656529    0.654650    0.633567    0.671362    0.000000
CAD6022189.1    0.575263    0.565998    0.572254    0.544560    0.516271
0.533867    0.558776    0.551143    0.541126    0.541818    0.521363    0.530871
0.206698    0.537155    0.539725    0.529315    0.393651    0.630309    0.000000
    AAC45821.1  AAC45824.1  CDL76811.1  CAN01912.1  BAB91150.1  AAB18965.1
AAA16512.1  AAC24604.1  AAA56889.1  AAA26281.1  ABQ96888.1  AAD10533.1
CSB13334.1  CAD00110.1  CAC97368.1  AAA85622.1  AAA95993.2  BAA28179.1
CAD6022189.1
```

Now we can build a tree, using a similar procedure as when generating the distance matrix:

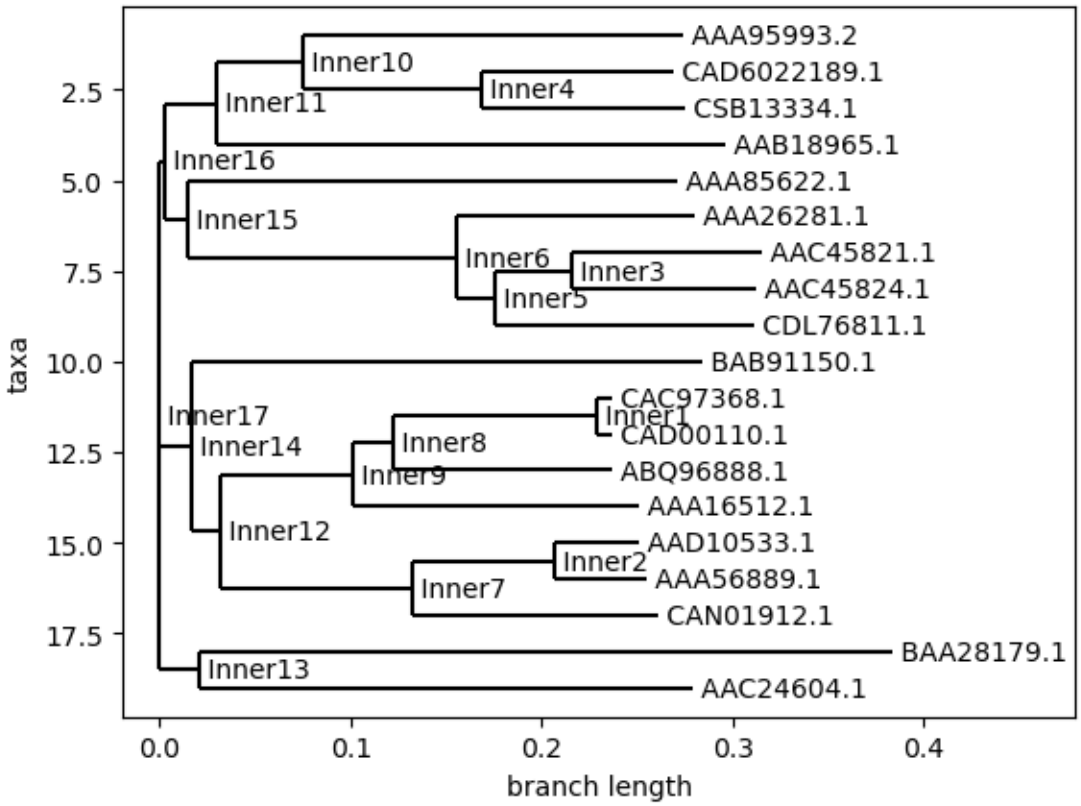
[11]:
```
constructor = DistanceTreeConstructor(calculator) #Generate tree constructor␣
 ↪object
ftsZ_tree = constructor.build_tree(align) #Apply constructor on the MSA
print(ftsZ_tree)
```

```
Tree(rooted=False)
    Clade(branch_length=0, name='Inner17')
        Clade(name='Inner16')
            Clade(name='Inner11')
                Clade(name='Inner10')
                    Clade(name='AAA95993.2')
                    Clade(name='Inner4')
                        Clade(name='CAD6022189.1')
                        Clade(name='CSB13334.1')
                Clade(name='AAB18965.1')
            Clade(name='Inner15')
                Clade(name='AAA85622.1')
                Clade(name='Inner6')
                    Clade(name='AAA26281.1')
                    Clade(name='Inner5')
                        Clade(name='Inner3')
                            Clade(name='AAC45821.1')
                            Clade(name='AAC45824.1')
                        Clade(name='CDL76811.1')
        Clade(name='Inner14')
            Clade(name='BAB91150.1')
            Clade(name='Inner12')
                Clade(name='Inner9')
                    Clade(name='Inner8')
                        Clade(name='Inner1')
                            Clade(name='CAC97368.1')
                            Clade(name='CAD00110.1')
```

```
                Clade(name='ABQ96888.1')
            Clade(name='AAA16512.1')
        Clade(name='Inner7')
            Clade(name='Inner2')
                Clade(name='AAD10533.1')
                Clade(name='AAA56889.1')
            Clade(name='CAN01912.1')
    Clade(name='Inner13')
        Clade(name='BAA28179.1')
        Clade(name='AAC24604.1')
```

But this is not a very nice way to print the tree, is it?

[12]:
```
fig = Phylo.draw(ftsZ_tree)
```



This is a bit nicer, but the record ids are not very informative. For the sake of tree visualization, we will change record ids to species names:

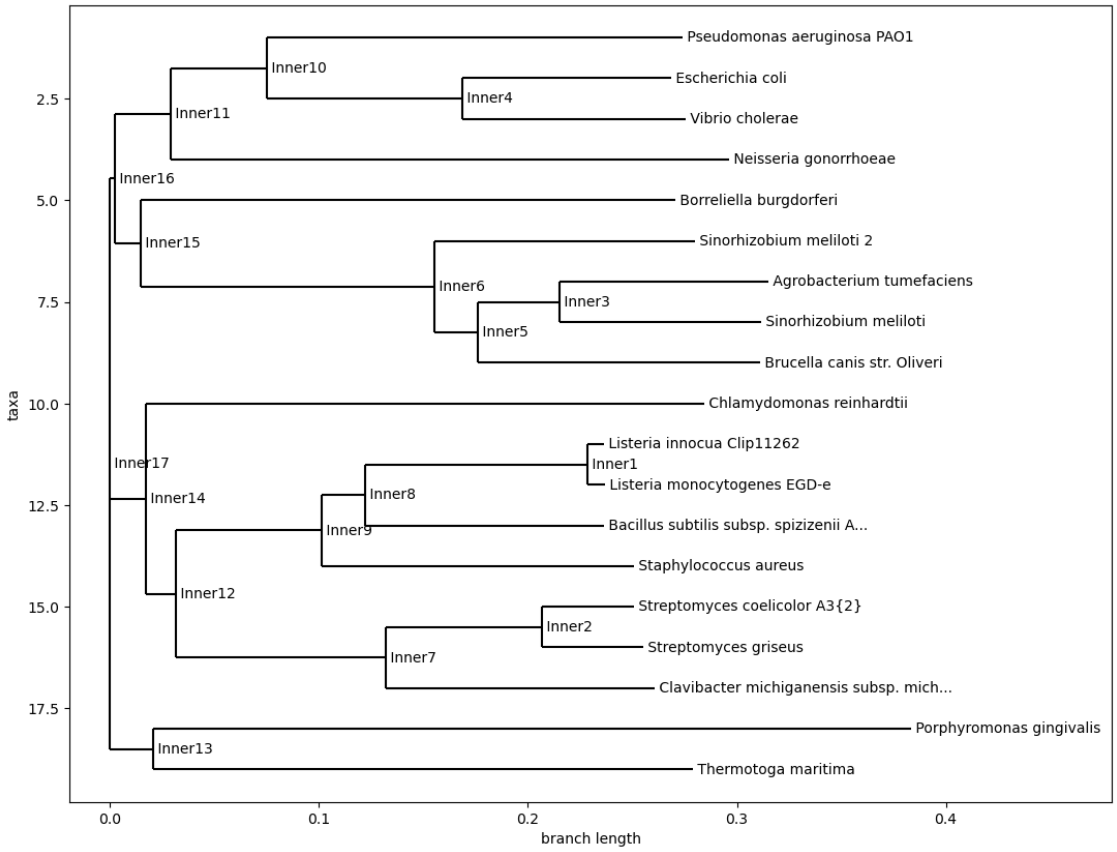
[13]:
```
names_list = [] #Create and empty list to store node names (to avoid duplicates)
for record in align: #Loop through records
    strain_name = record.description.split("[") #Split the description with the␣
 ↪opening square bracket
```

```
    strain_name = strain_name[1] #Take the second element of the resulting list␣
↪(strain name)
    strain_name = strain_name.strip("]") #Remove the closing brackets
    strain_name = strain_name.replace("(", "{") #Change parentheses to other␣
↪type of brackets to avoid parsing errors
    strain_name = strain_name.replace(")", "}")
    if strain_name in names_list: #Conditional statement to modify duplicate␣
↪names
        names_list.append(strain_name) #Add strain name to list again
        strain_name += f" {names_list.count(strain_name)}" #Add the number of␣
↪times the sequence is present
    else:
        names_list.append(strain_name) #Add strain name to list
    record.id = strain_name
    print(record.id)
```

```
Agrobacterium tumefaciens
Sinorhizobium meliloti
Brucella canis str. Oliveri
Clavibacter michiganensis subsp. michiganensis NCPPB 382
Chlamydomonas reinhardtii
Neisseria gonorrhoeae
Staphylococcus aureus
Thermotoga maritima
Streptomyces griseus
Sinorhizobium meliloti 2
Bacillus subtilis subsp. spizizenii ATCC 6633 = JCM 2499
Streptomyces coelicolor A3{2}
Vibrio cholerae
Listeria monocytogenes EGD-e
Listeria innocua Clip11262
Borreliella burgdorferi
Pseudomonas aeruginosa PAO1
Porphyromonas gingivalis
Escherichia coli
```

Now we re-run and draw the tree again:

[14]:
```
ftsZ_tree = constructor.build_tree(align)
fig = plt.figure(figsize=(13, 10), dpi=100) #Set a bigger figure size
axes = fig.add_subplot(1, 1, 1) #Add axes to pass to the phylo plot
Phylo.draw(ftsZ_tree, axes=axes)
```



1.2.1 This is great, but... what am I looking at?

This is not a great tree, as you can see the two sequences from *S. meliloti* don't cluster together, what suggests that these two genes could be paralogs. Even so, most species that belong to the same genus (*Streptomyces*, *Listeria*) have clustered together, and *S. meliloti* has clustered together with *A. tumefaciens*, that belongs to the same family. The clustering of *A. tumefaciens* and *S. meliloti* with *Brucella* doesn't make any sense phylogenetically speaking, but as I mention the gene could be the same paralog (these three sequences were 550-600 amino acids long, as opposed to the rest that were ~400 amino acids long).

Phylogenetic relationships are always complicated. We don't only have the problem of **paralogs** (copies of the same gene in a genome that can have evolved idependently), but also events of **horizontal gene transfer (HGT)**, where different species of bacteria can exchange genetic material. Not to mention that some genes can change a lot after **speciation events**, for example, when a parasite adapts to a new host. In general, single-gene trees like this one are not reliable predicting the true phylogenetic relationship between two organisms.

You should also keep in mind that this tree is **unrooted**, which means that the sequences at the root (*P. gingivalis* and *T. maritima*) are not necessarily the most early divergent. Not to mention that there is a trifurcation at the root of the phylogeny, instead of a bifurcation. That means that the node is unresolved.

Now let's look more in depth into the data itself. Like with SeqIO, we can use the Phylo module to read (Phylo.read for single trees and Phylo.parse for multiple trees) or to write trees, as we can see in the cell below:

[17]:
```
Phylo.write(ftsZ_tree, "ftsZ.nex", "nexus") #Write tree file in nexus format

with open("ftsZ.nex") as nexus: #Open the file
    for line in nexus: #Loop through file
        print(line) #Print content of file
```

```
#NEXUS

Begin Taxa;

 Dimensions NTax=19;

 TaxLabels Pseudomonas aeruginosa PAO1 Escherichia coli Vibrio cholerae
Neisseria gonorrhoeae Borreliella burgdorferi Sinorhizobium meliloti 2
Agrobacterium tumefaciens Sinorhizobium meliloti Brucella canis str. Oliveri
Chlamydomonas reinhardtii Listeria innocua Clip11262 Listeria monocytogenes
EGD-e Bacillus subtilis subsp. spizizenii ATCC 6633 = JCM 2499 Staphylococcus
aureus Streptomyces coelicolor A3{2} Streptomyces griseus Clavibacter
michiganensis subsp. michiganensis NCPPB 382 Porphyromonas gingivalis Thermotoga
maritima;

End;

Begin Trees;

 Tree tree1=((((('Pseudomonas aeruginosa PAO1':0.19915,('Escherichia
coli':0.10004,'Vibrio
cholerae':0.10666)Inner4:0.09351)Inner10:0.04570,'Neisseria
gonorrhoeae':0.26671)Inner11:0.02694,('Borreliella
burgdorferi':0.25567,('Sinorhizobium meliloti 2':0.12469,(('Agrobacterium
tumefaciens':0.09973,'Sinorhizobium meliloti':0.09630)Inner3:0.03948,'Brucella
canis str. Oliveri':0.13532)Inner5:0.02072)Inner6:0.14018)Inner15:0.01254)Inner1
6:0.00247,('Chlamydomonas reinhardtii':0.26711,((((('Listeria innocua
Clip11262':0.00788,'Listeria monocytogenes
EGD-e':0.00810)Inner1:0.10662,'Bacillus subtilis subsp. spizizenii ATCC 6633 =
JCM 2499':0.11433)Inner8:0.02078,'Staphylococcus
aureus':0.14963)Inner9:0.06972,(('Streptomyces coelicolor
A3{2}':0.04406,'Streptomyces griseus':0.04851)Inner2:0.07448,'Clavibacter
michiganensis subsp. michiganensis NCPPB
382':0.12857)Inner7:0.10052)Inner12:0.01437)Inner14:0.01716,('Porphyromonas
gingivalis':0.36252,'Thermotoga
maritima':0.25801)Inner13:0.02084)Inner17:0.00000;

End;
```

This is how the nexus format looks like: it contains node names and, next to the names, branch lengths. This is not "good practice". If we want to show a phylogenetic tree, we should also show **bootstrap support values** for the branches, but this toy example illustrates how the information is stored.

If we want to generate a tree object from a file, we just have to use the Phylo.read function:

[18]:
```
new_tree = Phylo.read("ftsZ.nex", "nexus")
print(new_tree)
```

```
Tree(name='tree1', rooted=False, weight=1.0)
    Clade(branch_length=0.0, name='Inner17')
        Clade(branch_length=0.00247, name='Inner16')
            Clade(branch_length=0.02694, name='Inner11')
                Clade(branch_length=0.0457, name='Inner10')
                    Clade(branch_length=0.19915, name=''Pseudomonas aeruginosa
PAO1'')
                    Clade(branch_length=0.09351, name='Inner4')
                        Clade(branch_length=0.10004, name=''Escherichia coli'')
                        Clade(branch_length=0.10666, name=''Vibrio cholerae'')
                Clade(branch_length=0.26671, name=''Neisseria gonorrhoeae'')
            Clade(branch_length=0.01254, name='Inner15')
                Clade(branch_length=0.25567, name=''Borreliella burgdorferi'')
                Clade(branch_length=0.14018, name='Inner6')
                    Clade(branch_length=0.12469, name=''Sinorhizobium meliloti
2'')
                    Clade(branch_length=0.02072, name='Inner5')
                        Clade(branch_length=0.03948, name='Inner3')
                            Clade(branch_length=0.09973, name=''Agrobacterium
tumefaciens'')
                            Clade(branch_length=0.0963, name=''Sinorhizobium
meliloti'')
                        Clade(branch_length=0.13532, name=''Brucella canis str.
Oliveri'')
        Clade(branch_length=0.01716, name='Inner14')
            Clade(branch_length=0.26711, name=''Chlamydomonas reinhardtii'')
            Clade(branch_length=0.01437, name='Inner12')
                Clade(branch_length=0.06972, name='Inner9')
                    Clade(branch_length=0.02078, name='Inner8')
                        Clade(branch_length=0.10662, name='Inner1')
                            Clade(branch_length=0.00788, name=''Listeria innocua
Clip11262'')
                            Clade(branch_length=0.0081, name=''Listeria
monocytogenes EGD-e'')
                        Clade(branch_length=0.11433, name=''Bacillus subtilis
subsp. spizizenii ATCC 6633 = JCM 2499'')
                    Clade(branch_length=0.14963, name=''Staphylococcus aureus'')
```

```
                Clade(branch_length=0.10052, name='Inner7')
                    Clade(branch_length=0.07448, name='Inner2')
                        Clade(branch_length=0.04406, name=''Streptomyces
coelicolor A3{2}'')
                        Clade(branch_length=0.04851, name=''Streptomyces
griseus'')
                    Clade(branch_length=0.12857, name=''Clavibacter
michiganensis subsp. michiganensis NCPPB 382'')
            Clade(branch_length=0.02084, name='Inner13')
                Clade(branch_length=0.36252, name=''Porphyromonas gingivalis'')
                Clade(branch_length=0.25801, name=''Thermotoga maritima'')
```

1.3 Plotting trees with ete3

We have seen how to manage trees in Biopython, but perhaps we want to plot them in a nicer way. The **ete3** library allows you to plot the trees that you generate nicely! It has some other functions; for example, you can also test **models of evolution** on your data, but we won't cover this in the course.

For this section, instead of the `biopython.yml` environment, called `bioinfo`, we will use the environment in the `trees.yml` file, called `ete3`. Without this new environment, you might run into errors when trying to save the tree to an image. As always, the first step is to load the required modules:

[41]:
```
from ete3 import Tree
```

Now we are ready to load our data! Out input tree file will be `GH32_protein.treefile`, in the `inputs/tutorial5` folder:

[42]:
```
treefile = "inputs/tutorial5/GH32_protein.treefile"

t = Tree(treefile, format = 0)
t.render("GH32_domains_1.png");
```

Loading a tree is easy, we just had to use the `Tree` function and specify the tree format. In this case, the format is `0`, which indicates that we have support values. We could also have used format `2`, that indicates that we have all branches, leaf names and support values. You can get started with `ete3` and look into the available formats here. If you write `t.show()`, you should be able to visualize your tree in the **ETE browser**; however, this will kill the Python kernel (you will have to run all this section from scratch), so instead we will save the outputs to images with the `t.render()` command.

Now look at the file that you generated, `GH32_domains_1.png`. This is not an ideal representation of the data, is it? If we have information such as where the root of the tree is placed, we can set the root with the `set_outgroup()` method. If we don't want to place the root on a leaf, but on an internal node, we can use the method `get_common_ancestor()`. With this method, we specify two leaves and get the internal node where both of their branches are joined in the tree:

[44]:
```
t = Tree(treefile, format = 0)
outnode = t.get_common_ancestor("C2_S3", "E1_S3")
```

```
t.set_outgroup(outnode)
t.render("GH32_domains_2.png");
```

Perhaps now we want to **style** the tree a bit. It looks quite small as it is, perhaps we want to make it a bit bigger. We also don't have any information on support values, even though they are in the original tree file. We should include them! Otherwise our phylogeny won't be very interpretable. We can change these display options by using the `TreeStyle` module and creating a `TreeStyle` object for our tree.

[45]:
```
from ete3 import TreeStyle
```

[46]:
```
t = Tree(treefile, format = 0)
outnode = t.get_common_ancestor("C2_S3", "E1_S3")
t.set_outgroup(outnode)

ts = TreeStyle()
ts.show_branch_support = True # Show support values
ts.scale =  200 # Scale tree up
t.render("GH32_domains_3.png", tree_style = ts);
```

Much nicer! We can look at the support value now, and the image is bigger. Now we might want to **style the nodes** individually, for example, by removing the blue circles around them and by adding colors to leave names. In our case, leaves represent different **strains** of a bacterial species, *Apilactobacillus kunkeei*, so perhaps we want to color them according to the phylogroup to which each strain belongs.

[59]:
```
from ete3 import NodeStyle, TextFace
```

[64]:
```
t = Tree(treefile, format = 0)
outnode = t.get_common_ancestor("C2_S3", "E1_S3")
t.set_outgroup(outnode)

ts = TreeStyle()
ts.show_leaf_name = False # Remove default leaf name text
ts.show_branch_support = False # Remove default support values
ts.scale = 400 # Make the tree even bigger
```

We need to provide the colors for each phylogroup. The phylogroup is defined by the first letter of the leaf name, so we will use these letters as keys and the colors as values in a dictionary:

[50]:
```
leaf_color = {"A": "#2C85D8", "B": "#21C1D1", "C": "#2AA380", "E": "#7ACA2E",␣
 ↪"F": "#A3C615"}
```

Now, we can apply first the changes to the nodes and lines, then add the formatted support values and finally add the formatted leaf names:

[65]:
```
ns = NodeStyle() # Create node style

#----------------------------#
```

```
# 1. Format nodes and branches #
#------------------------------#
ns["size"] = 0 # Removes blue circles at the tips of leaves
ns["vt_line_width"] = 5 # Increases width of vertical branch lines
ns["hz_line_width"] = 5 # Increases witdth of horizontal branch lines

#--------------------------------------------------#
# 2. Add style and formatted support values to nodes#
#--------------------------------------------------#
for n in t.traverse(): # Loop through nodes in the tree
   n.set_style(ns) # Apply style to all nodes

   if n not in t.get_leaves() and n.support > 10: # If the node is not a leave␣
 ↪and its suport value is > 10
      support_face = TextFace(int(n.support), fgcolor = "grey", fsize = 24) #␣
 ↪Create text to plot for support values
      n.add_face(support_face, position = "branch-top") # Add support value␣
 ↪text to node

#------------------------------#
# 3. Add formatted leaf names #
#------------------------------#
for leaf in t.get_leaves(): # Loop through leaves in the tree
    color = leaf_color[leaf.name[0]] # Get color based on first letter of leaf␣
 ↪name (phylogroup)
    name_face = TextFace(leaf.name, fgcolor = color, fsize = 36) # Create text␣
 ↪to plot with leaf name
    leaf.add_face(name_face, position = "branch-right") # Add leaf name text to␣
 ↪node

t.render("GH32_domains_4.png", tree_style = ts);
```

We added the node style to the nodes using the `set_style()` method for each node. Then, we added our texts (support values and names) with the `add_face()` method. The `position` argument that we specifies tells the computer where we want to plot the texts, on top of the branches (bootstrap support values) or to the right of the branches (leaf names). The first argument is the text that we want to plot. It has to be a **`TextFace`** object. That's why created `TextFace` objects for support values and leaf names. The first argument that we gave to the object was the text itself. Then, we provided other two arguments, `fgcolor`, that is the color that we want for our text, and `fsize`, that refers to the font size.

1.4 Okay, what do you want me to do this time?

Now you can take the tree that you created, `ftsZ.nex`, and try to **plot it in ete3**. Not just the standard plot, no. You need to try to change the style of nodes and leaf names.

OBS! In this case, you don't have support values for your branches, so make sure that you **choose the right format** when reading the tree into an object.