# Visualization of Metagenomic and Metatranscriptomic Data in R

## David Levy-Booth

## March 17 - 19, 2020

## Overview

Metagenomics and metatranscriptomics offer unparalleled insight into the taxonomic and functional diversity present across a multitude of environments. Yet the vast amounts of data these tools produce is challenging to effectively analyze and visualize.

We will analyze a unique thermal swamp metagenome and metatranscriptome dataset to develop strategies for effective visual communication.
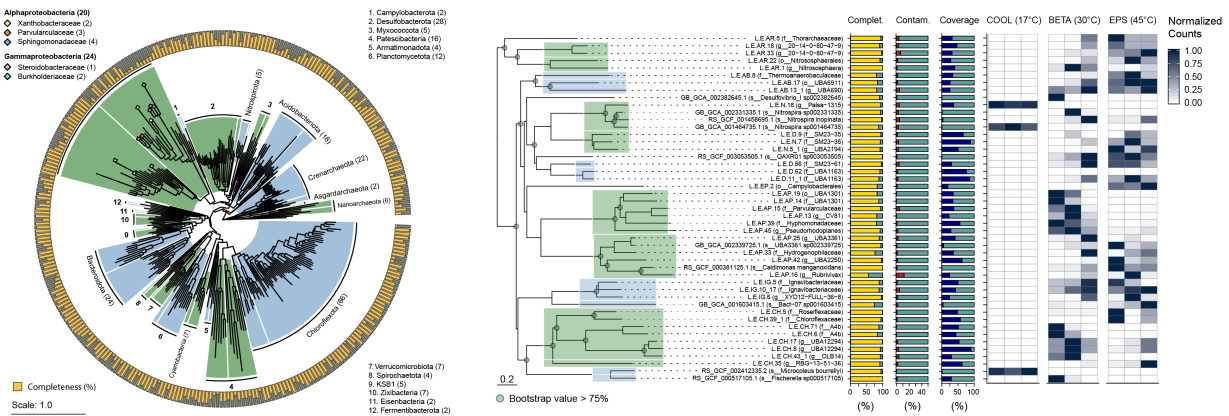


Figure 1: Thermal Swamp Metagenomics

This workshop is designed around the application of whole-genome phylogenetic analysis to organize the diverse data types that we can generate from metagenomics and metatranscriptomics analysis.

You will learn:

1. Manipulating and visualizing whole-genome phylogenies with ggtree
2. Mapping genome assembly statistics to phylogenetic trees with ggplot2
3. Processing count data from Meiji and mapping to phylogenetic trees
4. Genomic functional annotation and pathway analysis with pathview
5. Mapping metatranscriptomic count data and coverage to circular genome maps with circlize
6. Bonus: Mapping metatranscriptomic count data and coverage to linear contig maps

This workshop assumes some prior experience with R (such as that in our Introduction to R workshop). All code presented in this workshop is contained in the "metagenomics.R" R script file.

You can follow along with the workshop by selecting the relevant line(s) of code and pressing ctrl+enter on Windows, cmd+enter on MacOS, or using the "Run" button in the upper right of your script window to execute it.

Note that the majority of this workshop uses base R functions. No experience with Tidyverse is required.

Please call over the instructor or a TA if you have any problems with executing the code or if you need any help with the exercises.

## Prior to the workshop

### Setup Instructions

Please come to the workshop with your laptop setup with the required software and data files as described in our setup instructions.

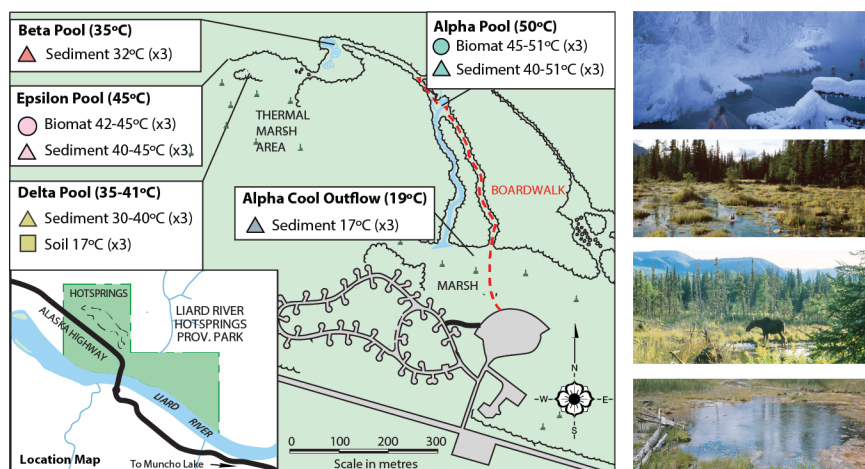### Background: Field site and shotgun sequencing methods



Figure 2: Liard River Hot Springs

Thermal swamp complexes are rare ecosystems containing unknown diversity of biogeochemical cycling and organic biomass-recycling microbial consortia. By applying metagenomic and metatranscriptomic techniques across thermally-distinct environments we can better understand the organisms responsible for nutrient transformations in these ecosystems, and how their distributions are shaped by thermal properties. Characterizing the thermal microbiome can also open up new biotechnological possibilities including lignin bio-refinement.

Liard River Hot Springs (59.431, 126.1) is a complex of carbonate-hosted hot springs that feed thermally-warmed meteoric waters to extensive swampland underlayed by sandstone and shale. The pools are surrounded by unique thermal meadow vegetation and mixed forest (*Populus tremuloides, Betula papyrifera, Picea glauca*). The two main hot spring samples we will analyze in this workshop are Beta (30-35°C) and Epsilon (40-45°C) pools. Water cools as it traverses the marshland via the Alpha river outflow to about 17°C at our final sampling location (Cool).

One ng DNA from three replicates of selected sediment samples (Cool, Beta, Epsilon) was prepared for shotgun sequencing on one NextSeq550 (Illumina) run in High Output mode using Nextera XT library preparation (Illumina).

As the warmest undisturbed pool, 2 g sediment from Epsilon was incubated with 10 mg of vanillate, Eucalyptus milled-wood lignin (EMWL), Eucalyptus Kraft lignin (EKL), Coniferyl-alcohol dehydrogenase-polymer (DHP). Control incubations were conducted with no exogenous carbon. RNA was extracted from approx. 0.5 g sediment. RNA integrity was measured using an Agilent 2100 Bioanalyzer (Agilent Technologies) before

and after rRNA removal with RiboMinus Transcriptome Isolation Kit, bacteria (ThermoFisher). Sequencing libraries were prepared using SuperScript Double-Stranded cDNA synthesis (ThermoFisher) and NexteraXT (Illumina) for sequencing using NextSeq550 (Illumina) High Output.

Genomes were extracted from metagenome assemblies using MetaBat2 and manually refined.

## Background: Genome Taxonomy Database

The Genome Taxonomy Database (GTDB) https://gtdb.ecogenomic.org/ is a standardized bacterial and archaeal taxonomy based on genome phylogeny. It contains about 24,000 species-level genomes with revised taxonomy for the genomic era. We will leverage the GTDB and its associated classification tool, GTDB-Tk https://github.com/Ecogenomics/GTDBTk to classify and place the whole-genome phylogeny of our assembled genomes.

For each genome our job will be to answer the following biological questions:

1. What are they? (Taxonomic classification)
2. Can we trust them? (Assembly statistics)
3. Where are they? (Abundance calculations)
4. What can they do? (Function/Pathway analysis)
5. What are they doing? (Metatranscriptomics)

## Getting started

## Making an RStudio Project

Projects allow you to divide your work into self-contained contexts.

Let's create a project to work in.

In the top-right corner of your RStudio window, click the "Project: (None)" button to show the projects drop-down menu. Select "New Project." > "New Directory" > "New Project." Under directory name, input "metagenomics" and choose a parent directory to contain this project on your computer.

## Installing and loading packages

At the beginning of every R script, you should have a dedicated space for loading R packages. R packages allow any R user to code reproducible functions and share them with the R community. Here, we load the necessary packages which must already be installed (see setup instructions for details).

```
# Suite of packages for data manipulation and visualization
library(tidyverse)
# Working with trees
library(ape)
library(phytools)
# Plotting Grammar of Graphics
library(ggplot2)
# Tree visualization
library(ggtree)
# String maniputation
library(stringr)
# Data structure manipulation
library(reshape2)
# Multi-panel figure layouts
```

```r
library(cowplot)
library(grid)
# Color maniputation
library(RColorBrewer)
# Pathway analysis with KEGG Orthologs
library(pathview)
#plot circular graphs and contig maps
library(circlize)
```

## Downloading data

The following command downloads the data from our GitHub and since you're working in a Project, saves it in the Project directory under the 'data' directory on your computer.

```r
#All data can be found in the directory
download.file(
  "https://raw.githubusercontent.com/davidlevybooth/Metagenomics_Workshop/master/data/checkm.qa.tsv",
  "data/checkm.qa.tsv")
download.file(
  "https://raw.githubusercontent.com/davidlevybooth/Metagenomics_Workshop/master/data/gtdb.classificatio
  "data/gtdb.classification.tree")
download.file(
  "https://raw.githubusercontent.com/davidlevybooth/Metagenomics_Workshop/master/data/gtdb.classificatio
  "data/gtdb.classification.tsv")
download.file(
  "https://raw.githubusercontent.com/davidlevybooth/Metagenomics_Workshop/master/data/gtdb.taxonomy.tsv
  "data/gtdb.taxonomy.tsv")
download.file(
  "https://raw.githubusercontent.com/davidlevybooth/Metagenomics_Workshop/master/data/metagenome.counts
  "data/metagenome.counts.tsv")
download.file(
  "https://raw.githubusercontent.com/davidlevybooth/Metagenomics_Workshop/master/data/KO.genomes.tsv",
  "data/KO.genomes.tsv")
download.file(
  "https://raw.githubusercontent.com/davidlevybooth/Metagenomics_Workshop/master/data/KO.pathways.tsv",
  "data/KO.pathways.tsv")
download.file(
  "https://raw.githubusercontent.com/davidlevybooth/Metagenomics_Workshop/master/data/LE.CH5.EKL.cov",
  "data/LE.CH5.EKL.cov")
download.file(
  "https://raw.githubusercontent.com/davidlevybooth/Metagenomics_Workshop/master/data/LE.CH5.gff",
  "data/LE.CH5.gff")
```

## Downloading custom functions

Throughout this workshop we will develop and apply functions that we wish to recall later. The R script "metagenome.utilities.R" contains a series of helper scripts to carry out advanced metagenomic and meta-transcriptomic analysis. Advanced users are encouraged to read through this file and alter the functions therein.

```
download.file(
  "https://raw.githubusercontent.com/davidlevybooth/Metagenomics_Workshop/master/scripts/metagenome.util
  "scripts/metagenome.utilities.R")
```

**Loading custom functions**

```
#Custom functions for metagenomics and metatranscriptomics
source("scripts/metagenome.utilities.R")
```

**Introduction to phylogenomics**

Please see the slide deck for an introduction to, and motivations for, phylogenetic analysis of microbial genomes.

Note that due to time and memory constraints, the genomes that we will use in this workshop have already undergone classification with GTDB-Tk.

# Section 1. Basic Tree Plotting

Load the GTDB-Tk tree for Thermal Swamp Metagenome-Assembled Genomes (MAGs).

We use the analyses of phylogenetics and evolution (ape) package, which provides basic functionality for loading and manipulating phylogenetic data. Here, we will use the read.tree() function to open the GTDB-Tk classification tree for the genomes that we will use in throughout these exercises.

Note that GTDB-Tk will separately analyze and place archaeal and bacterial genomes using 122 or 120 single copy genes (SCGs). We then used the bind.tree() function to combine both trees at their default roots. This combined tree is what is provided for you.

```
tree.path <- "data/gtdb.classification.tree"

#Use read.tree(<tree.path>) to load tree file from path.
#Important: Make sure you use the "ape" version of the read.tree() function
tree <- ape::read.tree(tree.path)


#Inspect tree object
tree
```

```
##
## Phylogenetic tree with 24743 tips and 24738 internal nodes.
##
## Tip labels:
##   GB_GCA_002780705.1, GB_GCA_002841685.1, GB_GCA_002779055.1, GB_GCA_900317315.1, GB_GCA_900316815.1,
## Node labels:
##   d__Bacteria, 1.0-p__CG03-c__CG03-o__CG03-f__CG03, 1.0-g__CG03, 1.0, 0.842, 0.718, ...
##
## Unrooted; includes branch lengths.
```

Let's also load the taxonomy paths for the GTDB Genomes. The taxonomy of the GTDB is based on a revised, standardized taxonomic-rank normalization according to relative evolutionary divergence (RED). (See Park et al., 2018. Nat. Biotech 4229)

```
GTDB.taxonomy.path <- "data/gtdb.taxonomy.tsv"

#Load data
GTDB.taxonomy <- read.csv(GTDB.taxonomy.path, sep="\t")
```

What does the taxonomy data look like?

```
head(GTDB.taxonomy)
```

```
##            GenomeID
## 1 GB_GCA_002506705.1
## 2 GB_GCA_002687735.1
## 3 GB_GCA_002256595.1
## 4 GB_GCA_002725275.1
## 5 GB_GCA_002457605.1
## 6 RS_GCF_002114285.1
##
## 1                    d__Archaea;p__Thermoplasmatota;c__Poseidoniia;o__Poseidoniales;f__Poseidoniacea
## 2                      d__Archaea;p__Iainarchaeota;c__Iainarchaeia;o__Iainarchaeales;f__Iainarchaea
## 3     d__Archaea;p__Halobacterota;c__Methanosarcinia;o__Methanotrichales;f__Methanotrichaceae;g__Met
## 4                d__Archaea;p__Thermoplasmatota;c__Poseidoniia;o__Poseidoniales;f__Poseidoniaceae;g_
## 5 d__Archaea;p__Thermoplasmatota;c__Poseidoniia;o__Poseidoniales;f__Thalassoarchaeaceae;g__Thalassar
## 6                d__Archaea;p__Halobacterota;c__Halobacteria;o__Halobacteriales;f__Haloferacaceae;g__
##      Kingdom              Phylum              Class              Order
## 1 d__Archaea p__Thermoplasmatota     c__Poseidoniia     o__Poseidoniales
## 2 d__Archaea     p__Iainarchaeota    c__Iainarchaeia    o__Iainarchaeales
## 3 d__Archaea     p__Halobacterota c__Methanosarcinia o__Methanotrichales
## 4 d__Archaea p__Thermoplasmatota     c__Poseidoniia     o__Poseidoniales
## 5 d__Archaea p__Thermoplasmatota     c__Poseidoniia     o__Poseidoniales
## 6 d__Archaea     p__Halobacterota     c__Halobacteria  o__Halobacteriales
##               Family              Genus                           Species
## 1      f__Poseidoniaceae         g__MGIIa-K1         s__MGIIa-K1 sp002506705
## 2      f__Iainarchaeaceae           g__UBA493           s__UBA493 sp002687735
## 3   f__Methanotrichaceae     g__Methanothrix    s__Methanothrix sp002256595
## 4      f__Poseidoniaceae         g__Poseidonia       s__Poseidonia sp002725275
## 5 f__Thalassoarchaeaceae g__Thalassarchaeum s__Thalassarchaeum sp002457605
## 6      f__Haloferacaceae       g__Halorubrum     s__Halorubrum ezzemoulense
```

Isolate tip labels. Each tip is a genomic placement.

```
tips <- tree$tip.label
head(tips)
```

```
## [1] "GB_GCA_002780705.1" "GB_GCA_002841685.1" "GB_GCA_002779055.1"
## [4] "GB_GCA_900317315.1" "GB_GCA_900316815.1" "GB_GCA_900317595.1"
```

Question: What other data are associated with tree objects?

If we try to plot all 23K GTDB bacterial genomes with MAG placements the figure would be a mess, and my computer will probably explode. Let's subset the number of tips to a reasonable number first using a custom function.

```
tips.keep <- sample_gtdb_tips(tips, 1000); length(tips.keep)
```

```
## [1] 1000
```

Use keep.tips() to subset tree.

```
tree.subset <- keep.tip(tree, tips.keep)
```

```
#if you are having difficulty with you subset tree, please uncomment to load from the following file:
#tree.subset <- read.tree("results/tree.subset")
```

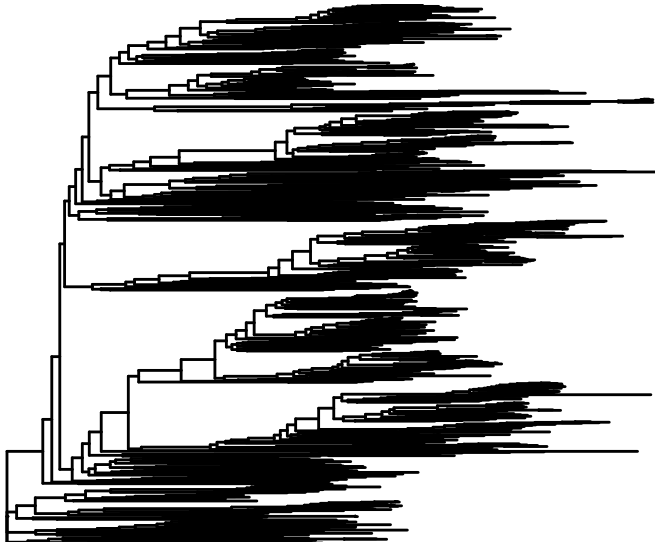> Question: How else could we have accomplished this?

Plot the reduced tree with ggtree.

Note that we set xlim() to make room for longer labels (ggtree generally does a poor job of managing plot space)

```
tree.plot <- ggtree(tree.subset) + xlim(0,4)
```

```
## Warning: `expand_scale()` is deprecated; use `expansion()` instead.
```
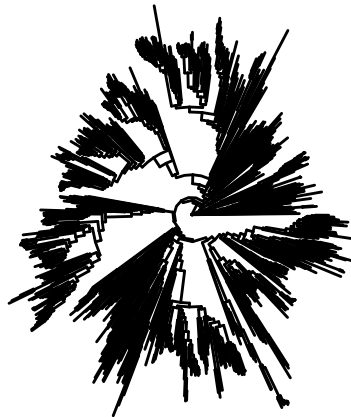
```
tree.plot
```



We can take advantage of several layouts.

For example, while we will focus on plotting data to linear trees, let's take a look at plotting a circular tree, just for kicks.
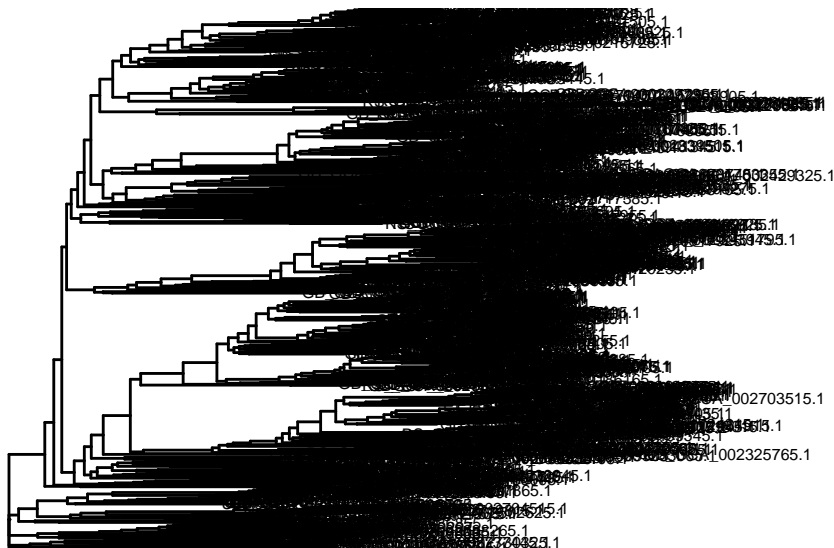
```
ggtree(tree.subset, layout="circular")
```

## Warning: `expand_scale()` is deprecated; use `expansion()` instead.



Plot the reduced tree with ggtree – add tip labels with geom_tiplab()

```
tree.plot + geom_tiplab(size = 2)
```



Hmm, not very readable. Let's add phyla labels with geom_cladelabel() instead.

First, let's use the taxonomy table to select a random phyla from the GTDB. I'm using Firmicutes_A, but we could select any number of other targets. Note that we are trying to isolate one specific group of genomes by GenomeID that all share the same Phylum. To do this we subset the GTDB taxonomy table using the

which() function, *which* creates a Boolean vector of rows that match our condition: in this case if the Phylum column contains the string "p___Firmicutes_A"

Let's review some base R fundamentals before we proceed:

A data.frame in R is made of rows of observations and columns of variables. For metagenome or microbiome count data, rows are typically genome, OTUs, etc., which columns containing counts in each sample. Note that this organization is transposed for some ecological statistics packages (e.g., in *vegan*) that expect rows of samples with columns of species/OTUs/genomes.

Regardless, the rows and columns within a data.frame is represented by positions between square brackets, e.g., [rows, columns]. So if we want to take the first 4 rows we can subset the data.frame like this:

```r
data.frame[1:4, ]
```

This means that we can use index (index) vectors OR Boolean (True, False) vectors to subset data with a host of functions (which, match, etc.). Below we use which to subset the data.frame, which has analogous functionality to filter() in tidyverse.

To demonstrate this we will subset the GTDB taxonomy table to only our clade of interest. We then only take the GenomeIDs for these subset genomes. We must also ensure that the resulting list is a character vector because a factor will do screwy things to the findMRCA() function.

```r
#Assign our phylogenetic clade of interest to a variable called "Clade"
Clade <- "p__Firmicutes_A"


#Here we combine a few base R functions to subset the GTDB taxonomy table to only:
Firmicutes_A_tips <- as.character(GTDB.taxonomy[which(GTDB.taxonomy[,"Phylum"] == Clade),]$GenomeID)


head(Firmicutes_A_tips)
```

```
## [1] "GB_GCA_900317315.1" "GB_GCA_900316815.1" "GB_GCA_900317595.1"
## [4] "GB_GCA_900316435.1" "GB_GCA_900318905.1" "GB_GCA_900315195.1"
```

Second, we have to learn to use the findMRCA() function in the phytools package. This allows us to find the common ancestor nodes for a list of tree tips.

```r
Firmicutes_A_node <- findMRCA(tree, Firmicutes_A_tips, type="node")
Firmicutes_A_node
```

```
## [1] 24762
```

Next, I created the collapse_nodes() function to apply findMRCA() over the whole tree to collect nodes for all phyla. It by iterating through each clade at a specified taxonomic level (in this case "Phylum"), and calculates the ancestor node, how many tips belong to each clade, and the maximum branch length of the clade (useful for plotting collapsed clades).

```r
Phyla.nodes <- collapse_nodes("Phylum", tree.subset, GTDB.taxonomy)
```

Almost done, next we can highlight clades of interest.

We use the which() function again as above.

Question: How would we filter the table using tidyverse notation?

```
#In tidyverse
Phyla.nodes %>%
  filter(Group == "p__Firmicutes_A") %>%
  .$Node
```

```
## [1] 1014
```

```
#In base R
Firmicutes_A_node <- Phyla.nodes[which(Phyla.nodes$Group == "p__Firmicutes_A"), ]$Node

Firmicutes_A_node
```

```
## [1] 1014
```

```
tree.plot <- ggtree(tree.subset) + xlim(0,4)
```

```
## Warning: `expand_scale()` is deprecated; use `expansion()` instead.
```

```
tree.plot + geom_balance(node=Firmicutes_A_node,
                         fill='darkgreen',
                         color='white',
                         alpha=0.6,
                         extend=1)  +
          geom_cladelabel(node = Firmicutes_A_node,
                          "Firmicutes A",
                          offset = 1,
                          fontsize = 2)
```



ADVANCED: Plot highlights for other clades. How do you highlight more than one clade?

What about for ALL major clades?

We can run geom_balance() to produce colored ranges for all phyla in our dataframe in a loop.

The "R" way to run this would be using an apply function, as R is not engineered to effectively use "FOR" loops. But they work in a pinch:

```r
colors <- some_color(Phyla.nodes)

#For loops are a basic tool used in scripting languages.
#Is there a different/better way to achieve this functionality in R?

for(i in 1:nrow(Phyla.nodes)) {
  if(Phyla.nodes$TipCount[i] > 10) {
  tree.plot <- tree.plot +
    geom_balance(node=Phyla.nodes$Node[i], fill=colors[i], color='white', alpha=0.6, extend=0.6)  +
    geom_cladelabel(node = Phyla.nodes$Node[i], Phyla.nodes$Group[i], offset = 0.6, fontsize = 2)
  }
}
tree.plot
```
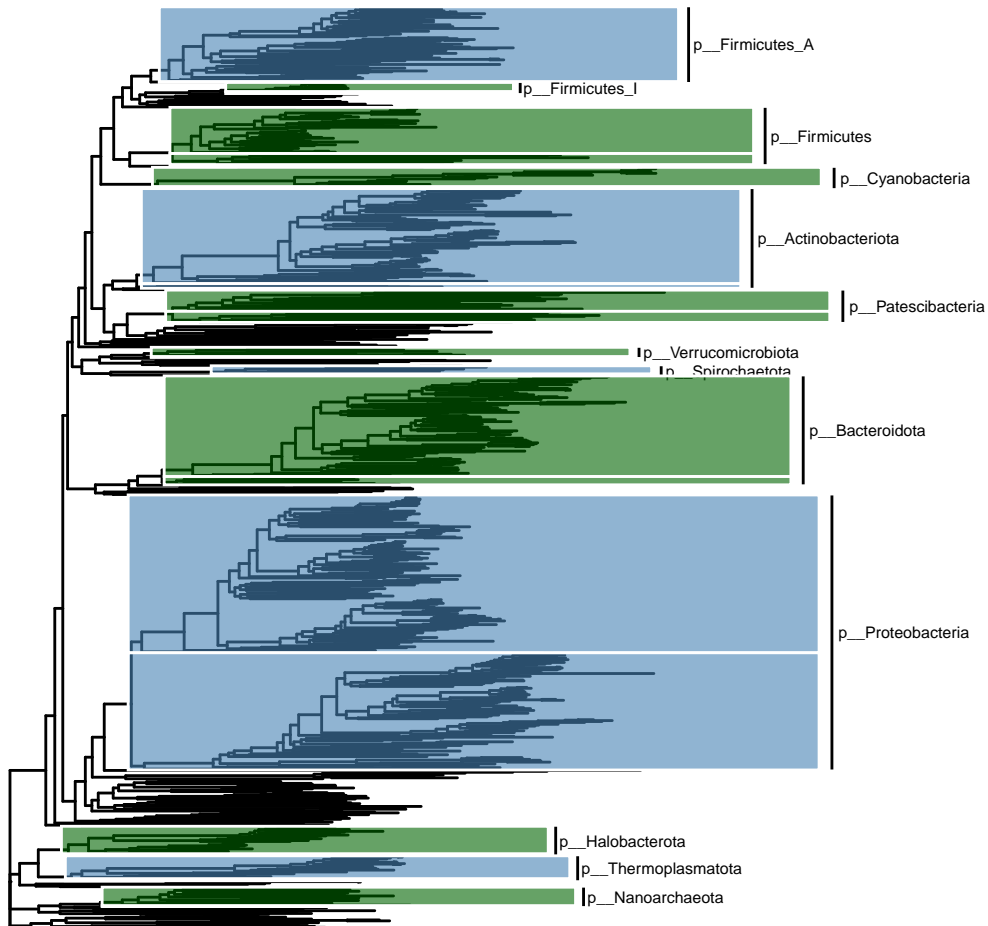


Can we do this for a circular plot?

To assist with adding colored ranges to the tree you are provided with a new custom function called clade_colors() that simply runs the loop above. It can be used with rectangular or circular trees.

```r
tree.plot <- ggtree(tree.subset, layout="circular")
```

```
## Warning: `expand_scale()` is deprecated; use `expansion()` instead.
```

```r
tree.plot <- clade_colors(tree.plot, Phyla.nodes, colors)
tree.plot
```
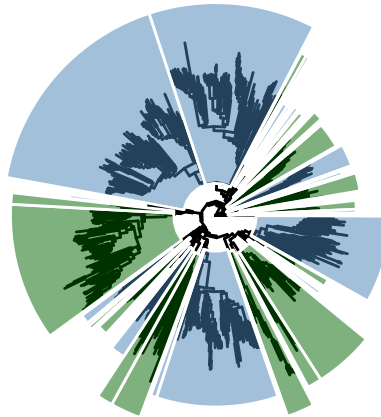


(Looks pretty badass to me!)

One more thing: adding bootstrap values to trees

```r
tree.subset$node.label <- parse_bootstraps(tree.subset, method = "parse")
```

```
## Warning in parse_bootstraps(tree.subset, method = "parse"): NAs introduced by
## coercion
```

ggtree is sometimes counterintuitive. Case in point geom_nodepoint() counts tips as nodes. To account for this we have to build a vector of bootstrap values that is offset by the number of tips for correct bootstrap placement in tree.

```r
bs_count <- parse_bootstraps(tree.subset, method = "count")
tail(bs_count)
```

```
## [1]  94 100  NA  98  NA  NA
```

Plot numeric bootstrap values with the ggtree::geom_text() implementation:

```r
ggtree(tree.subset) + xlim(0,3) + geom_text(aes(label = bs_count), size = 2)
```

```
## Warning: `expand_scale()` is deprecated; use `expansion()` instead.
```

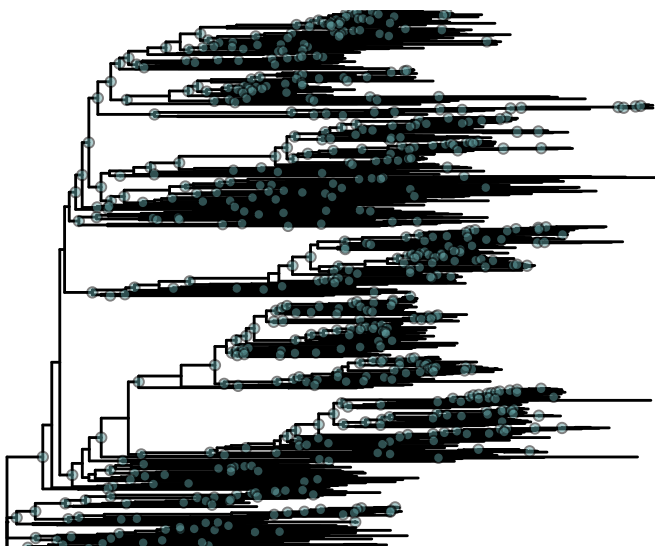## Warning: Removed 1191 rows containing missing values (geom_text).



Simple graphical representation of Bootstrap values

```
tree.plot <- ggtree(tree.subset) +
            xlim(0,4) +
            geom_nodepoint(aes(subset= bs_count >= 90),
                            fill = "cadetblue",
                            size=1.5,
                            alpha = 0.5,
                            shape = 21)
```
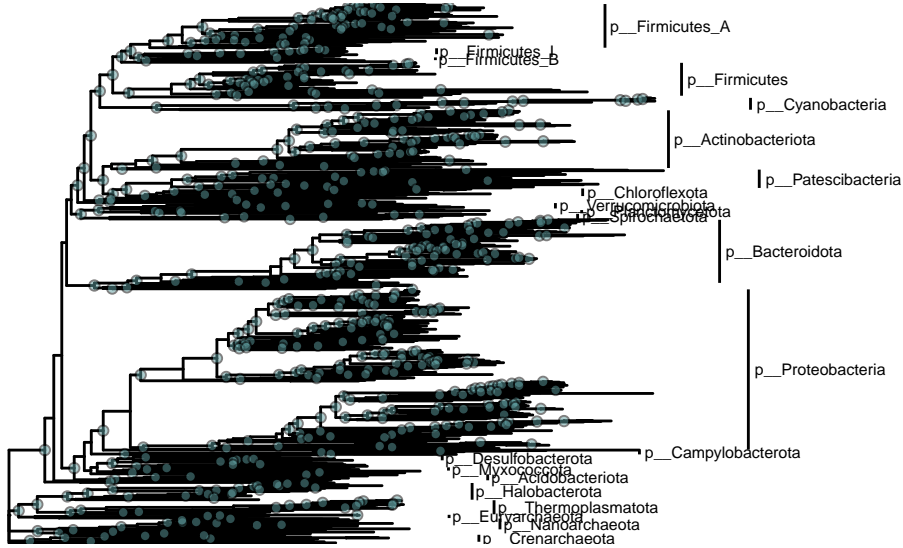
## Warning: `expand_scale()` is deprecated; use `expansion()` instead.
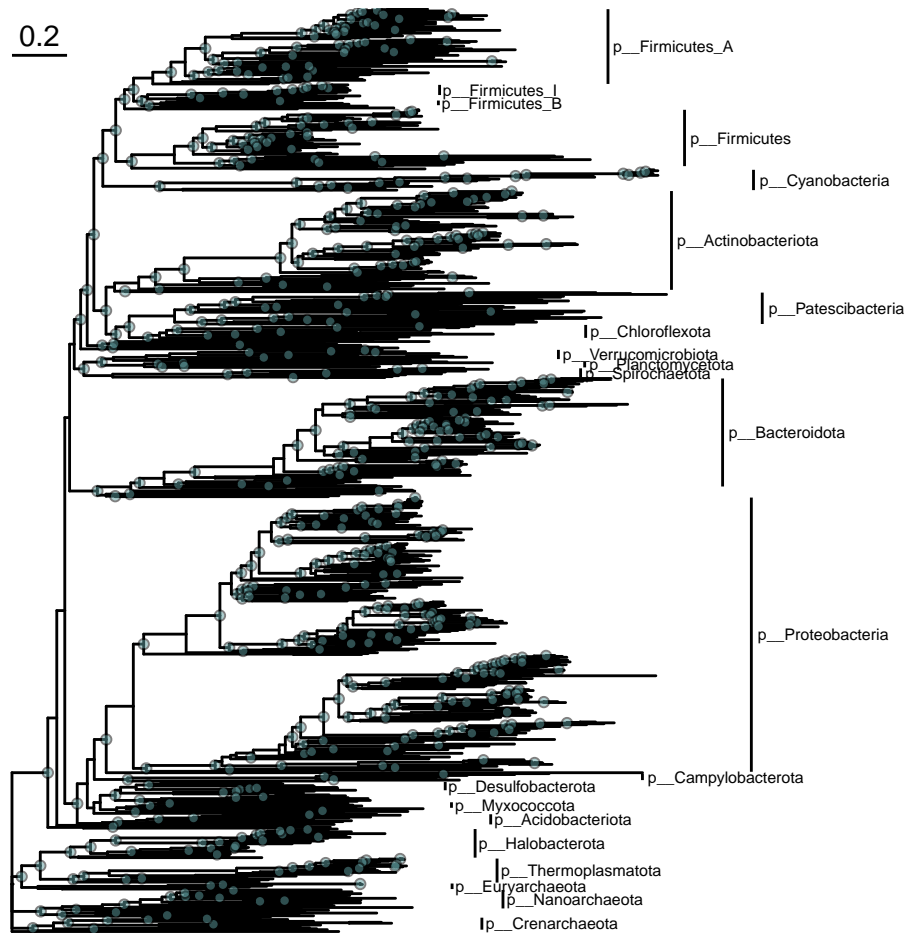
```
tree.plot
```

And add phyla labels:

```
tree.plot <- clade_labels(tree.plot, Phyla.nodes, tiplimit = 5, fontsize = 2)
tree.plot
```



Finally, we must add a scale representing the substitution distance

```
tree.plot + geom_treescale(x=0, y=length(tree.subset$tip.label)-50, width=0.2, offset = 10)
```

Exercise: Using the above functions, find and highlight the phylum of your genome in a tree with bootstrap values and a scale

# Section 2. Plotting MAG phylogenies

Set path for MAG taxonomy table and import.

Note that we use the read.csv() function here, but set the delimiter to tabs.

Question: What are the differences between read.csv(), read.table() and read.delim()?

```
mag.taxonomy.path <- "data/gtdb.classification.tsv"
mag.taxonomy <- read.csv(mag.taxonomy.path, sep="\t")
```

Subset tree to show only MAG placements using keep.tips().

```
tree.mags <- keep.tip(tree, as.character(mag.taxonomy$GenomeID))
tree.mags
```

##

```
## Phylogenetic tree with 47 tips and 46 internal nodes.
##
## Tip labels:
##   L.E.CL.63, RS_GCF_002412335.2, L.E.CH.35, L.E.CH.43_1, L.E.CH.8, L.E.CH.17, ...
## Node labels:
##   d__Bacteria, 0.869, 0.379, 0.948, 1.0-o__Cyanobacteriales, 0.846, ...
##
## Rooted; includes branch lengths.
```

Plot MAG tree with ggtree()

```
tree.plot <- ggtree(tree.mags, ladderize=F) + geom_tiplab(size = 2) + xlim(0,2.6)
```

```
## Warning: `expand_scale()` is deprecated; use `expansion()` instead.
```

```
tree.plot
```
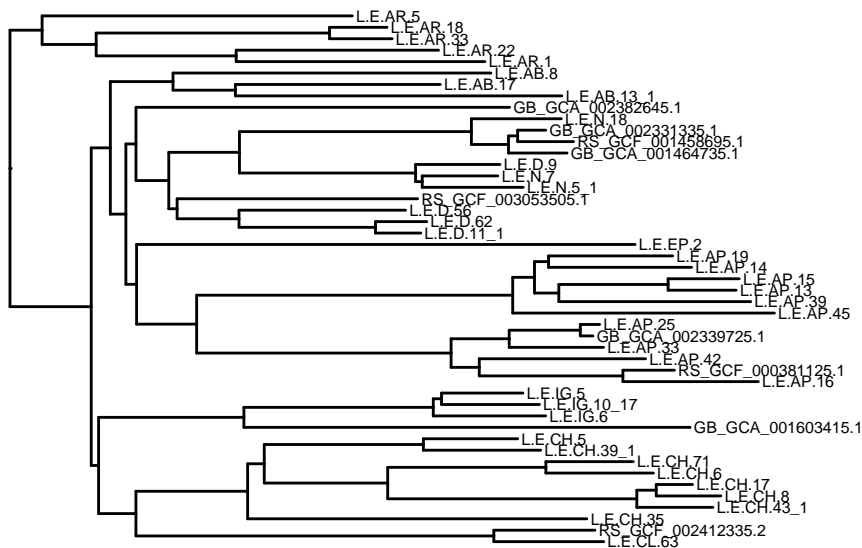


Question: Why is 'ladderize=F' so important here? Hint: it has to do with the order of the genomes vertically.

Add arbitrary color ranges with the predefined some_color() function.

Advanced: Check out the some_color() function in the metagenome.utilities.R file. how would you change this function to change how colored ranges are applied to taxonomic clades?

```
Phyla.nodes <- collapse_nodes("Phylum", tree.mags, mag.taxonomy)
colors <- some_color(Phyla.nodes)
```

Create tree plot with custom clade colors:

```
tree.plot <- clade_colors(tree.plot, Phyla.nodes, colors, extend = 0.6)
```

Add bootstraps with the parse_bootstraps() function:

```
tree.mags$node.label <- parse_bootstraps(tree.mags, method = "parse")
```

```
## Warning in parse_bootstraps(tree.mags, method = "parse"): NAs introduced by
## coercion
```

```
bs_count <- parse_bootstraps(tree.mags,method = "count")
```

> Question: What's the deal with the raw node values in the GTDB tree? How do we even find this information?

Finally: Add it all together for a great looking phylogeny of our MAGs:

```
tree.plot <- tree.plot +
  geom_nodepoint(aes(subset= bs_count >= 75), fill = "cadetblue", size=2, alpha = 0.5, shape = 21) +
  geom_treescale(x=0, y=1, width=0.1, offset = 0.5)
```

```
tree.plot
```
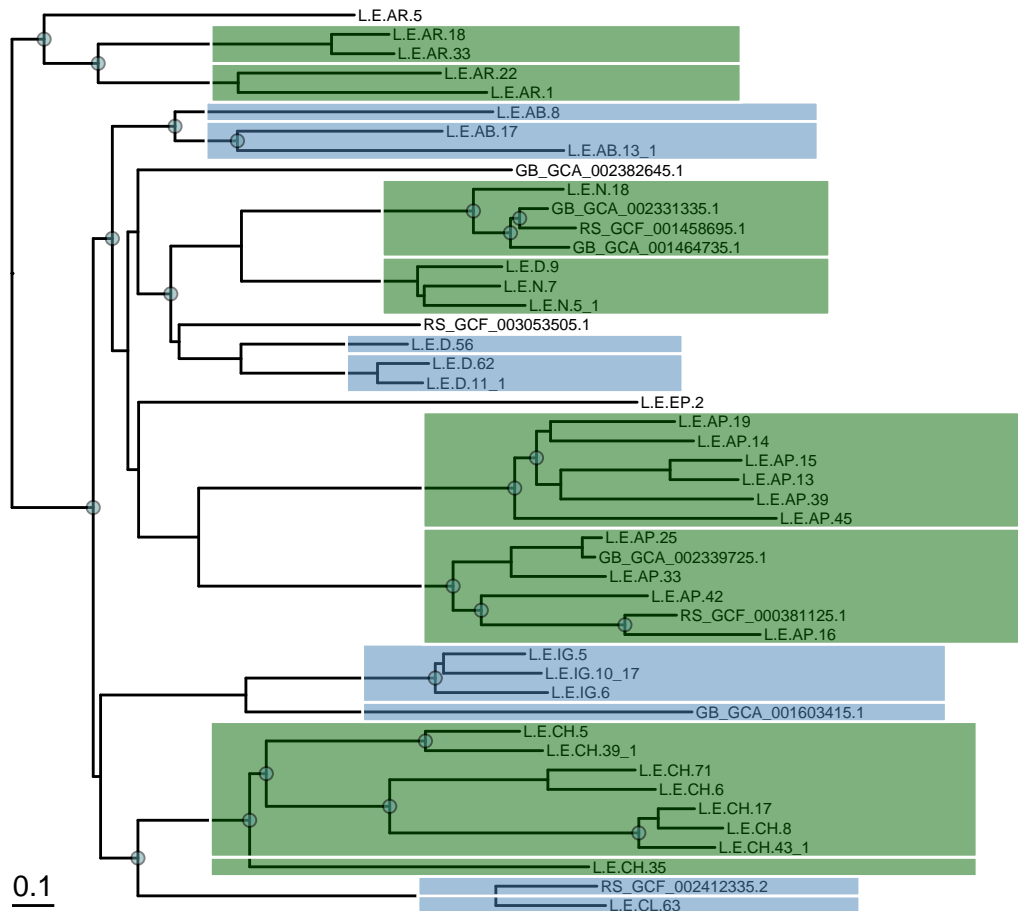


> ADVANCED: How would you go about highlighting only your genome of interest?

17

# Section 3. MAG statstic plots

We can use checkM data to map genome binning metrics to MAG tree. CheckM uses single copy genes (SCGs) to determine the levels of completeness and contamination ("redundancy") in a genome bin. We also use BBMap to calculate coverage.

```
MAG.checkM.path <- "data/checkm.qa.tsv"
MAG.checkM <- read.csv(MAG.checkM.path, sep="\t")

head(MAG.checkM)
```

```
##       GenomeID Completeness Contamination AR_Completeness AR_Contamination
## 1    L.E.CH.35        91.25          1.98            8.75            48.02
## 2 L.E.CH.43_1        92.58          3.82            7.42            46.18
## 3     L.E.CH.8        92.73          2.91            7.27            47.09
## 4    L.E.CH.17        99.09          2.00            0.91            48.00
## 5     L.E.CH.6        83.64          0.00           16.36            50.00
## 6    L.E.CH.71        88.18          0.00           11.82            50.00
##   Coverage AR_Coverage
## 1    62.95       37.05
## 2    29.67       70.33
## 3    90.25        9.75
## 4    53.55       46.45
## 5    43.30       56.70
## 6    51.30       48.70
```

To plot data we must organize data in the 'long' format using reshape2::melt()

> Question: What is the difference between 'wide' and 'long' data formats? When would each be used?

```
MAG.checkM.long <- melt(MAG.checkM, id.vars = "GenomeID")
```

"Wide" data:

```
head(MAG.checkM)
```

```
##       GenomeID Completeness Contamination AR_Completeness AR_Contamination
## 1    L.E.CH.35        91.25          1.98            8.75            48.02
## 2 L.E.CH.43_1        92.58          3.82            7.42            46.18
## 3     L.E.CH.8        92.73          2.91            7.27            47.09
## 4    L.E.CH.17        99.09          2.00            0.91            48.00
## 5     L.E.CH.6        83.64          0.00           16.36            50.00
## 6    L.E.CH.71        88.18          0.00           11.82            50.00
##   Coverage AR_Coverage
## 1    62.95       37.05
## 2    29.67       70.33
## 3    90.25        9.75
## 4    53.55       46.45
## 5    43.30       56.70
## 6    51.30       48.70
```

"Long" data:

```
head(MAG.checkM.long)
```

```
##       GenomeID     variable value
## 1    L.E.CH.35 Completeness 91.25
## 2 L.E.CH.43_1 Completeness 92.58
## 3     L.E.CH.8 Completeness 92.73
## 4    L.E.CH.17 Completeness 99.09
## 5     L.E.CH.6 Completeness 83.64
## 6    L.E.CH.71 Completeness 88.18
```

Remember to set GenomeID as an ordered factor to match tree tip order. Factors are important in R for organizing data without changing the order of the underlaying data.

```
MAG.checkM.long$GenomeID <- factor(MAG.checkM.long$GenomeID,
                                   levels = tree.mags$tip.label)

MAG.checkM.long$variable <- factor(MAG.checkM.long$variable,
                                   levels = rev(unique(MAG.checkM.long$variable)))
```

Create separate 'completeness' data.frame (still in "long" format).

We will use the imported 'factorize()' function to set a variable to be a factor where the factor levels are simply the current order of the variable in the data.frame.

But first we get to learn a new function!

> Question: What is grepl()? How does it differ from grep()?
>
> Answer: GREP stands for Global Regular Expression Print. We can match strings (characters), including with a wide variety of "wildcards" to subset a data.frame() by creating a numeric index vector for matching entries.
>
> grepl(), (grep list) in contrast, produces a Boolean vector. This is really useful for certain functions like ifelse() that apply different outcomes based on Booleans.

Here, we could use grep() or grepl() to catch rows containing both the "Completeness" variable as well as the completeness residual "Completeness_AR" variable.

```
MAG.checkM.long.completeness <- MAG.checkM.long[grepl("Completeness", MAG.checkM.long$variable), ]

MAG.checkM.long.completeness$variable <- factorize(MAG.checkM.long.completeness$variable, rev = TRUE)
```

> Question: Why do you think we used rev = TRUE here?
>
> Answer: When we plot data in ggplot2 with flipped x and y axes using coord_flip(), the order of the "y-axis" can be reversed from what we expect. The factorize() function is designed to account for this.

This is how we would create the completeness plot with ggplot2.

Within ggplot function we first specify the data.frame that contains the data.

We then set the plotting aesthetics with aes() – including the "x" and "y" variables. Here we also indicate that factors will be differentiated using "fill".

Question: How does "fill" differ from "color"?

We could just plot the bin completeness with a single bar, but I would like to have more control over the aesthetics of the unplotted region. To do this we have calculated the "residual" (100-completeness) and can assign a different fill color for this region manually. The residuals are plotted with completeness values as a stacked bar plot.

```
ggplot(MAG.checkM.long.completeness,
       aes(x = GenomeID, y = value, fill = variable)) +
  geom_bar(stat = "identity", color = "black") + #The "black" color provides the border.
  scale_fill_manual(values = c("cadetblue", "gold")) + #Manually assign fill for residual and completen
  coord_flip() + #Switches the x and y axis to produce vertical stacked bar plot.
  theme(legend.position = "none") + #We don't need a legend for these data
  ylab("Completeness") #Add label
```

That will work, but it would be cool to create a function that lets us apply extra formatting to all similar plot types. tree_bar_plot() replicates and extends the above ggplot plotting instructions. All we need to supply is the data and the x-axis label.

```
MAG.completeness <- tree_bar_plot(MAG.checkM.long.completeness, "Completeness")
MAG.completeness
```
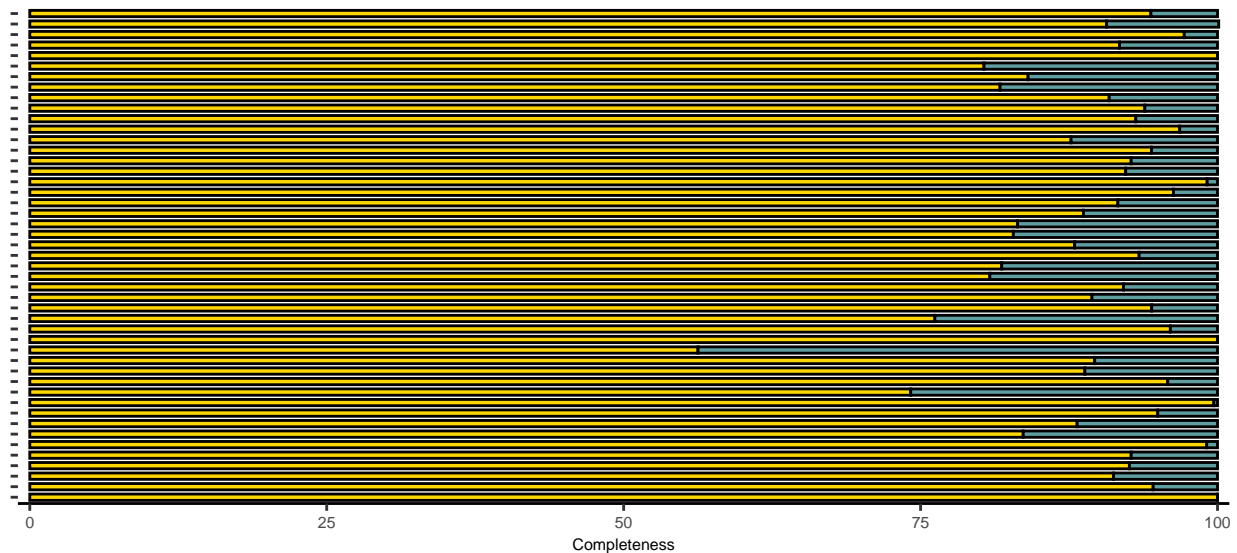


Question: Why did we not include y-axis text? How can we change the above plot to double check that we got the y-axis order correct?

ADVANCED: Create similar plots for genome Contamination and Coverage

20

```r
MAG.checkM.long.contamination <- MAG.checkM.long[grepl("Contamination", MAG.checkM.long$variable), ]
MAG.checkM.long.contamination$variable <- factorize(MAG.checkM.long.contamination$variable, rev = TRUE)

MAG.checkM.long.coverage <- MAG.checkM.long[grepl("Coverage", MAG.checkM.long$variable), ]
MAG.checkM.long.coverage$variable <- factorize(MAG.checkM.long.coverage$variable, rev = TRUE)


MAG.contamination <- tree_bar_plot(MAG.checkM.long.contamination, "Contamination",
                                   colors = c("cadetblue", "firebrick"))

MAG.coverage <- tree_bar_plot(MAG.checkM.long.coverage, "Coverage",
                              colors = c("cadetblue", "darkblue"))
```

Question: Is there an even more efficient way to create multiple plots using ggplot2?

Put it all together to plot genome statistics with the tree we created earlier:

```r
tree.plot <- tree.plot + scale_y_tree()
```

```
## Warning: `expand_scale()` is deprecated; use `expansion()` instead.
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which will
## replace the existing scale.
```

Question: Wait, why do we need this scale_y_tree() function?

Answer: scale_y_tree() is critical for aligning ggtree plots with ggplots.

Change values in scale_y_tree() function in data/metagenome.utilities.R if you need to change the vertical alignment with a ggplot. Warning: setting the y-scale is only for advanced users.

Align all plots together with cowplot::plot_grid()

Notice that we should be able to horizontally align the plots at the bottom axis with align = "h" and axis = "b"

```r
MAG.title <- plot_title("Liard River Phylogeny and Statistics")

MAG.plots <- plot_grid(tree.plot,
                       MAG.completeness,
                       MAG.contamination,
                       MAG.coverage,
                       align = "h",
                       axis = "b",
                       ncol = 4,
                       rel_widths = c(1.25, 0.25, 0.25, 0.25))

#Nesting the plot_grid() function allows us to easily add titles to our groups of plots!
plot_grid(MAG.title,
          MAG.plots,
          ncol = 1,
          rel_heights = c(0.1, 1))
```
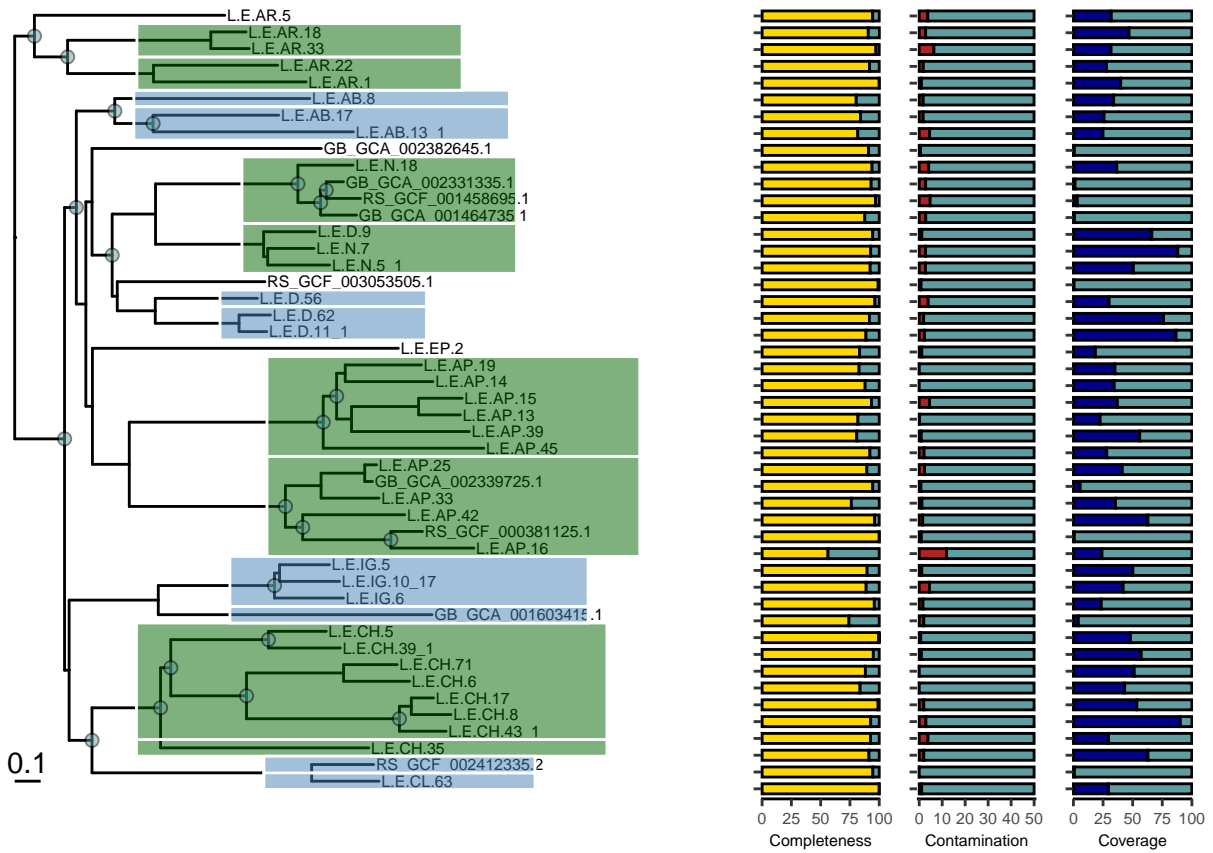
## Liard River Phylogeny and Statistics



Looks great!

# Section 4. Aligning abundance plots with phylogenetic trees.

Now we will use a heatmap to explore MAG environmental abundance and temperature range.

Load count data. Meiji was used to align shotgun metagenome reads to MAGs and select GTDB genomes.

```
counts.path <- "data/metagenome.counts.tsv"
counts <- read.table(counts.path, sep="\t", header = TRUE)
```

We should also parse the lowest level of taxonomic classification so we can tell what the heck each genome is without having to plot the whole taxonomy string.

```
counts$Classification <- tax_class(counts$Taxonomy)
```

Optionally, we can calculate relative abundance for each sample:

```
counts.ra <- relative_abundance(counts)
```

```
## Warning in if (nums == ncol(count_table)) {: the condition has length > 1 and
## only the first element will be used
```

Or normalize across taxa:

```
counts.norm <- normalize(counts)
```

Now, match the count data to the order of the tree (remember why we set 'ladderize=F'?), then format for plotting.

We use the match() function a few more times to ensure these data are organized correctly.

- *%in%* subsets a data.frame by creating Boolean vector of presence or absence in a list.
- *match* creates an index vector to re-order data.frame rows using a character vector.
- *order()* re-orders data.frame rows based on alphanumeric variables.
- *factor()* re-orders hierarchy of data.frame row entries without changing the data itself

    *match* is a really useful function for keeping data organized across data.frames!

    Question: Can you tell what the tax_class() function is doing by looking at the function in the "metagenome.utilities.R" script?

```
#We once again match the counts.norm and mags.taxonomy.norm data.frames with the order of tips in the t
#It can be confusing that the first vector in the match function is actually
#what you're trying to match to, so be careful with the order.

counts.norm <- counts.norm[match(tree.mags$tip.label, counts.norm$GenomeID), ]
mag.taxonomy.norm <- mag.taxonomy[match(tree.mags$tip.label, mag.taxonomy$GenomeID), ]

#Now that the tree tip, counts.norm and mag.taxonomy.norm all match we can consolidat the variables we
counts.norm$GenomeID <- tree.mags$tip.label
counts.norm$Taxonomy <- mag.taxonomy.norm$Taxonomy

#Finally we use the tax_class() function to pull out the lowest level of GTDB classification for each g
counts.norm$Classification <- tax_class(counts.norm$Taxonomy)
```

Melt data.frame for plotting and do some more formatting:

```
#Transform the count data to the "long" format for plotting.
#reshape2::melt() creates three types of columns: IDs, variables and values.

#We want to retain "GenomeID", "Taxonomy", and "Classification" as IDs.
#And we keep the actual counts as the "value". Making each sample a "variable"
counts.long <- melt(counts.norm,
                    id.vars = c("GenomeID", "Taxonomy", "Classification"),
                    value.name = "norm.count")

#Use factorize() again to lock in the current variable order when plotting
counts.long$variable <- factorize(counts.long$variable)

#We use the paste0 function to create a new label by combining the genome name and
#the classification.
```

```
counts.long$Label <- paste0(counts.long$Genome, " (", counts.long$Classification, ")")
counts.long$Label <- factorize(counts.long$Label)

#Finally, we use cut_last(), which is a wrapper for the substr() function.
#This allows us to trim characters from the end of our sample variables
#removing the replicates and keeping the pool IDs.
counts.long$Pool <- cut_last(counts.long$variable, 3)
counts.long$Pool <- factorize(counts.long$Pool)

#Let's take a look:
head(counts.long)
```

```
##              GenomeID
## 1          L.E.CL.63
## 2 RS_GCF_002412335.2
## 3          L.E.CH.35
## 4        L.E.CH.43_1
## 5           L.E.CH.8
## 6          L.E.CH.17
##
## 1  d__Bacteria;p__Cyanobacteria;c__Cyanobacteriia;o__Cyanobacteriales;f__Nostocaceae;g__Fischerella;
## 2 d__Bacteria;p__Cyanobacteria;c__Cyanobacteriia;o__Cyanobacteriales;f__Phormidiaceae;g__Microcoleus
## 3                                  d__Bacteria;p__Chloroflexota;c__Dehalococcoidia;o__GIF9;f__AB-5
## 4                                   d__Bacteria;p__Chloroflexota;c__Anaerolineae;o__Anaerolineale
## 5                                  d__Bacteria;p__Chloroflexota;c__Anaerolineae;o__Anaerolineales;f_
## 6                                  d__Bacteria;p__Chloroflexota;c__Anaerolineae;o__Anaerolineales;f_
##             Classification variable    norm.count
## 1 s__Fischerella sp000517105  COOL.01 0.0000000000
## 2  s__Microcoleus bourrellyi  COOL.01 0.9159663866
## 3            g__RBG-13-51-36  COOL.01 0.0000000000
## 4                 g__OLB14  COOL.01 0.0000000000
## 5              g__UBA12294  COOL.01 0.0001697169
## 6              g__UBA12294  COOL.01           NA
##                                                Label Pool
## 1        L.E.CL.63 (s__Fischerella sp000517105) COOL
## 2 RS_GCF_002412335.2 (s__Microcoleus bourrellyi) COOL
## 3                 L.E.CH.35 (g__RBG-13-51-36) COOL
## 4                      L.E.CH.43_1 (g__OLB14) COOL
## 5                     L.E.CH.8 (g__UBA12294) COOL
## 6                    L.E.CH.17 (g__UBA12294) COOL
```

Plot Gene Count Heatmap with ggplot2 using geom_tile()

Question: Why are we using facets to divide the data from different pools into different subplots?

```
counts.heatmap <- ggplot(data = counts.long, mapping = aes(x = variable, y = Label, fill = norm.count))
  geom_tile(color = "grey50") +
  scale_fill_gradient(low = "#FFFFFF", high = "#012345", na.value = "#FFFFFF") +
  scale_y_discrete(drop=FALSE) + #keep even "empty" genomes to preserve all elements in the tree
  theme(axis.text = element_text(size = 8, vjust = 0.4),
        axis.text.x = element_blank(),
        axis.title = element_blank(),
```
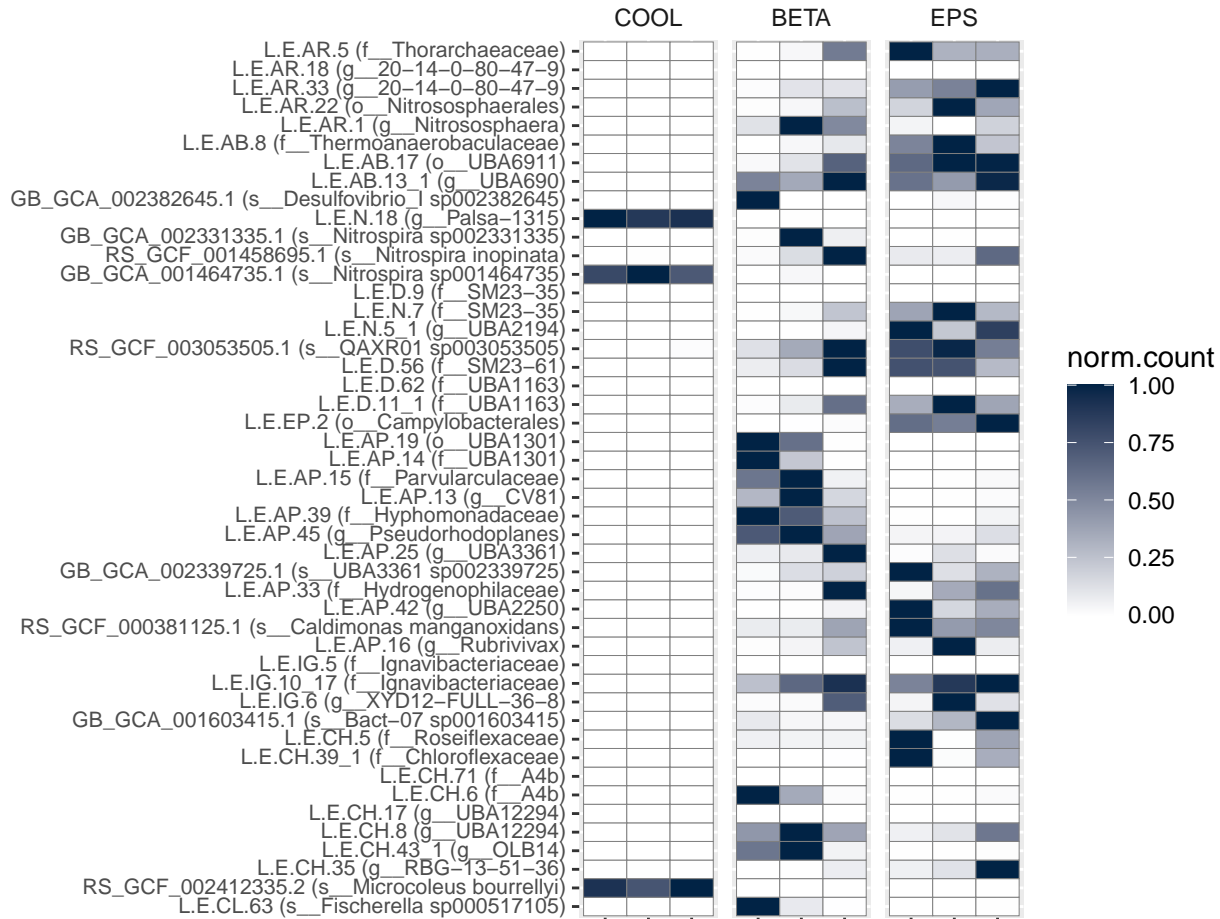
```
        strip.background = element_blank()) +
  facet_grid(.~Pool, scales = "free_x")

#Let's take a look at the plot!
counts.heatmap
```



Finally, combine tree and heatmap plot with cowplot::plot_grid()

```
counts.long$Genome <- factor(counts.long$GenomeID, levels = tree.mags$tip.label)

new.count.heatmaps <- heatmap_plot(counts.long) + theme(axis.text.y = element_blank())

#Put them together with cowplot::plot_grid()
MAG.title <- plot_title("Liard River MAG Phylogeny and Abundance")

MAG.plots <- plot_grid(tree.plot,
                       new.count.heatmaps,
                       align = "h",
                       axis = "bt",
                       ncol = 2,
                       rel_widths = c(1, 1))
```
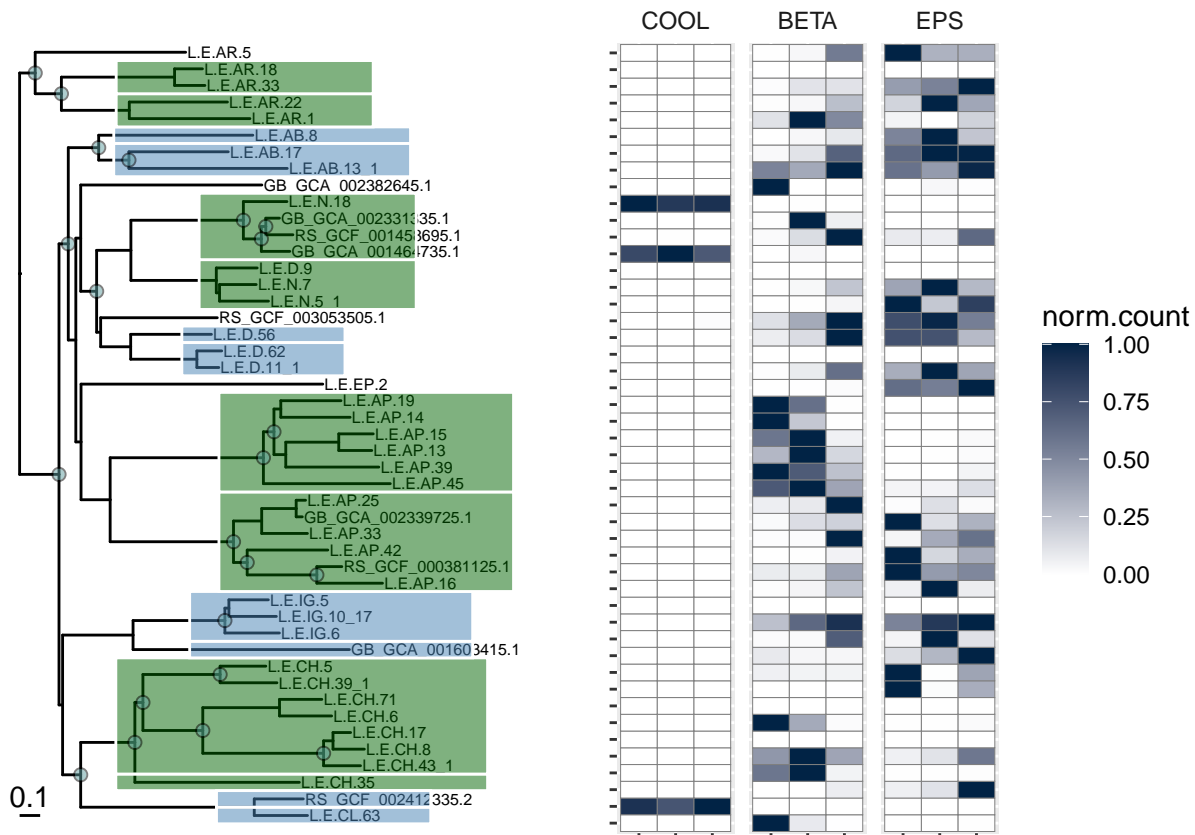
```
plot_grid(MAG.title,
          MAG.plots,
          ncol = 1,
          rel_heights = c(0.1, 1))
```

## Liard River MAG Phylogeny and Abundance



Question: What is the temperature range of your genome? In what pool is your genome most abundant?

# Section 5. Functional Annotation

A note on functional annotation:

To understand the ecological role(s) and functional potential of any discrete shotgun assembly (genome, metagenome, transcriptome), we must *annotate* their genes with confidence. To do this we can first predict open reading frames (ORFs) using Prodigal, then use KOFAMSCAN to apply profile hidden Markov Models to annotate KEGG orthologs and pathways.

These steps have already been run for you and the results tables for all genomes are concatenated. This organization works well for ggplot2, in which data is primarily supplied to in melted (long) format.

We also have a table of KEGG Orthologs (KOs) to help us organize data into different pathways (KO.pathways). These data were organized manually, which was a real pain in the butt.

26

Anyways, let's load our data:

```
# Load KEGG pathways of interest
KO.pathways <- read.table("data/KO.pathways.tsv", sep = "\t", header = TRUE)
KO.pathways$Function <- factorize(KO.pathways$Function)
#This is important to keep the pathway data organized for later!

#Load MAG kofamscan annotations
KO.mag <- read.delim("data/KO.genomes.tsv", sep = "\t", header = TRUE)

head(KO.mag)
```

```
##       GenomeID        ORF     KO Threshold BitScore   Evalue
## 1 L.E.AB.13_1  k119_25969_3 K05349    386.33    693.2 1.0e-208
## 2 L.E.AB.13_1  k119_64607_3 K09144     81.30    152.8   3.3e-45
## 3 L.E.AB.13_1  k119_64963_2 K02429    125.13    520.1 9.4e-157
## 4 L.E.AB.13_1 k119_141426_1 K12510     88.07    153.7   1.8e-45
## 5 L.E.AB.13_1 k119_141426_2 K12511    133.93    220.7   6.1e-66
## 6 L.E.AB.13_1 k119_141426_4 K07322     52.57    227.1   7.4e-68
##                                          Description
## 1                    beta-glucosidase [EC:3.2.1.21]
## 2                             uncharacterized protein
## 3    MFS transporter, FHS family, L-fucose permease
## 4                             tight adherence protein B
## 5                             tight adherence protein C
## 6 regulator of cell morphogenesis and NO signaling
```

Question: How were the pathways and genes annotated?

Select only genomes KOs in pathways of interest:

- N-cycling
- S-cycling
- CH4-cycling
- Aromatic degradation
- Photosynthesis
- Etc.

Look, we're using match again to ensure that the MAG and Pathway data are in the same order.

```
KO.mag <- KO.mag[which(KO.mag$KO %in% KO.pathways$KO), ]
KO.mag$Function <- KO.pathways[match(KO.mag$KO, KO.pathways$KO), ]$Function
KO.mag$Pathway <- KO.pathways[match(KO.mag$KO, KO.pathways$KO), ]$Pathway
```

Select pathways to plot:

```
keep.pathways <- c("Nitrogen Metabolism", "Sulphur Metabolism", "Photosynthesis", "Hydrogen Metabolism"

#Isolate selected pathways in data.frame
KO.mag <- KO.mag[which(KO.mag$Pathway %in% keep.pathways), ]
```

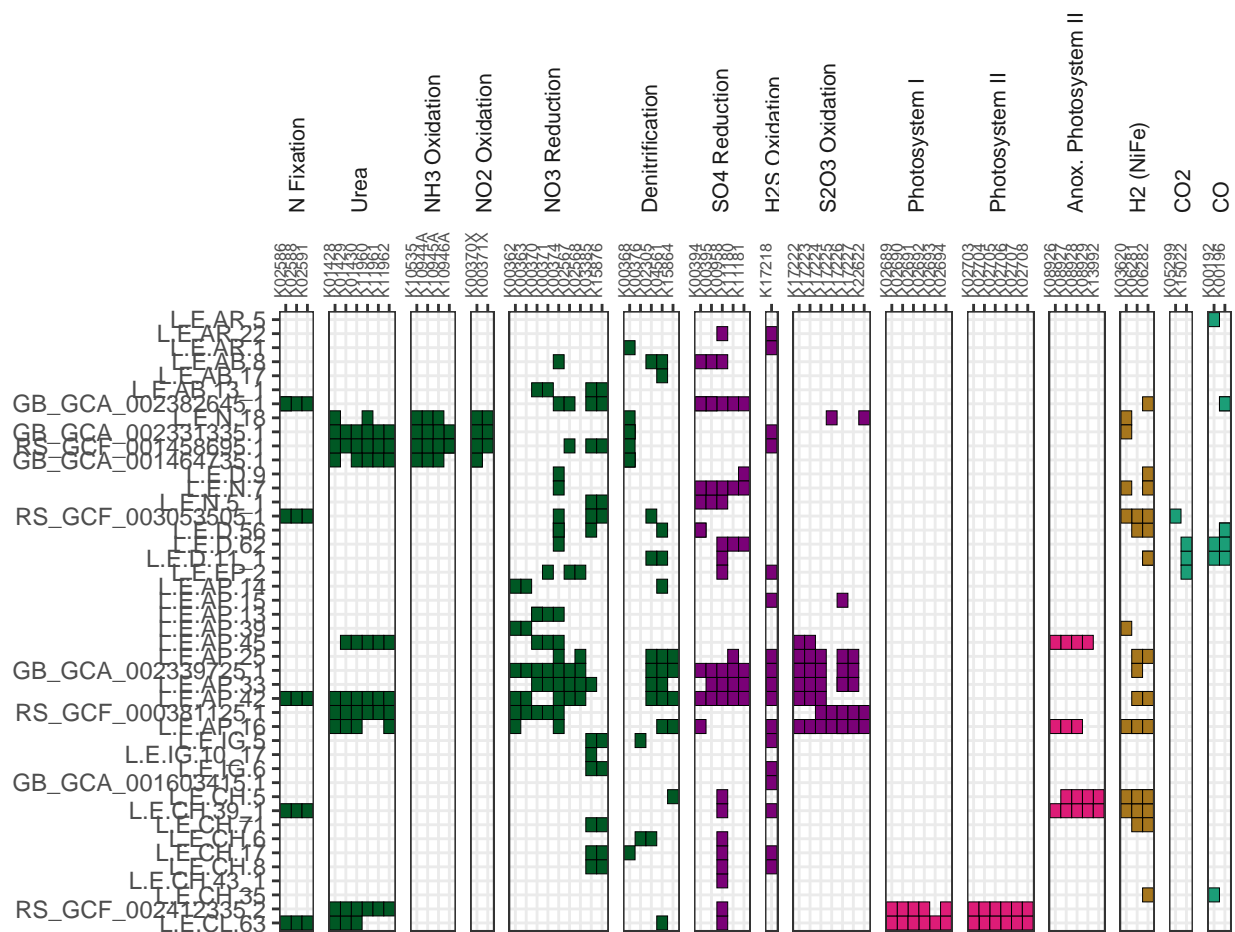Order the genomes for plotting based on order of tips in tree using a factor:

```
KO.mag$Genome <- factor(KO.mag$GenomeID, levels = as.character(tree.mags$tip.label))
```

Generate pathway colors and plot functional annotations. Plots use geom_tile() again but this time add color from the pallet by pathway function.

We also apply facetting to divide the individual functions.

```
pathway.colors <- set_path_colors()


#Plot functional annotations with ggplot2
function.plot <- ggplot(KO.mag, aes(x = KO, y = Genome, fill = Function)) +
  geom_tile(color = "black") + #geom_tile creates each square for an annotated gene
  scale_fill_manual(values = pathway.colors) + #I use a named vector to apply fills to specific conditi
  theme_bw() + # this is just making it pretty
  theme(axis.title = element_blank(),
        #axis.text.x = element_blank(),
        axis.text.x = element_text(angle = 90, hjust = 0, size = 6),
        legend.position = "none",
        strip.text.x = element_text(size = 8, angle = 90, hjust = 0, vjust = 0.6),
        strip.background = element_blank(),
        strip.placement = "outside") +
  scale_x_discrete(position = "top") +
  facet_grid(.~Function, scales="free_x", space="free_x") #facetting by Function
function.plot
```

N Fixation　Urea　NH3 Oxidation　NO2 Oxidation　NO3 Reduction　Denitrification　SO4 Reduction　H2S Oxidation　S2O3 Oxidation　Photosystem I　Photosystem II　Anox. Photosystem II　H2 (NiFe)　CO2　CO

Advanced: plot the aromatic degradation pathway

Hint: Find the label for the aromatic degradation genes in KO.mag$Pathway, then add label to the "keep.pathways" list.

# Section 6. Pathway Analysis

Optionally, we can use the *pathview* package to create Pathway Analysis maps. We won't spend a ton of time on this, but it can be useful to manually inspect your pathway data.

We will look at the benzoate degradation map for the Chloroflexi MAG L.E.CH.5.

- Benzoate degradation is KEGG pathway 00362 (Find more here: https://www.genome.jp/kegg/pathway.html)
- Pathview creates XML for KEGG pathway annotation.
- It writes the final file to your working directory using the pathway # and suffix string.
- Warning: if XML data is missing, pathview will attempt to download it (ensure internet connection)

```
#Create labelled benzoate degradation pathway map for genome: CH.5
CH5 <- KO.mag[which(KO.mag$GenomeID == "L.E.CH.5"), ]
```

```
pathview(gene.data = as.character(CH5$KO),
         pathway.id = "00362",
         species = "ko",
         gene.idtype = "KEGG",
         kegg.native = TRUE,
         out.suffix = "CH5")
```
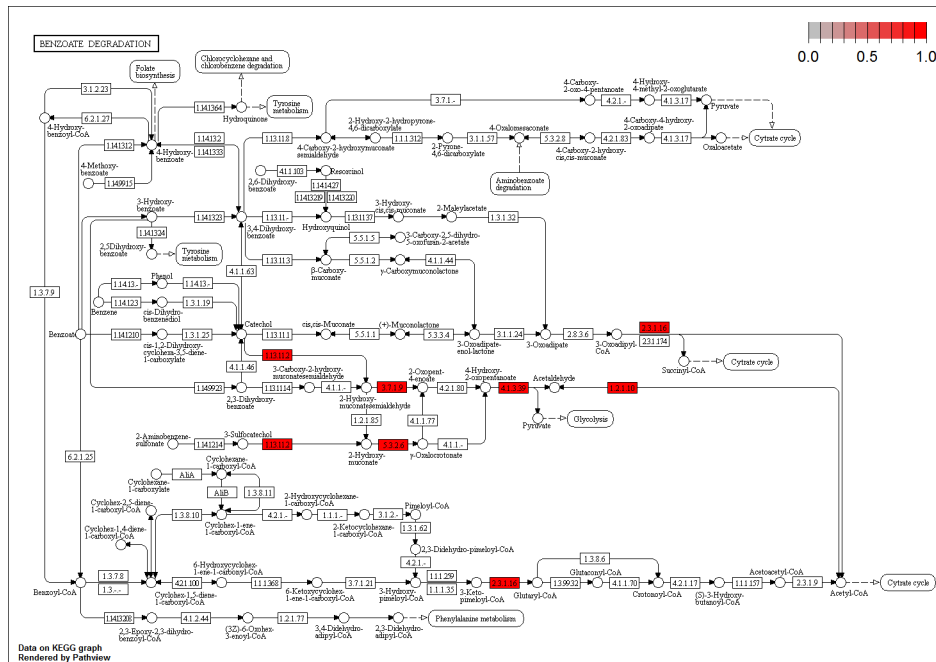


Figure 3: Roseiflexus sp. CH5 benzoate degradation

We can also supply quantitative data. Advanced uses are welcome to implement this on their own time.

# Section 7. Mapping Metatranscriptomic Data

Metatranscriptome analysis can be difficult to analyze effectively. In a traditional transcriptome study, we target cells generally in the same growth phase to obtain our desired signal. In an environmental sample, or even in an incubation, memebers of the microbiome may be in wildly different growth phases.

The following data comes from incubating Epsilon sediment at 45°C with the following substrates: * 0.1% Kraft Lignin (EKL) * 0.1% Milled Wood Lignin (EMWL) * 0.1% DHP Lignin (DHP) * 0.1% Vanillate (VAN) * No exogenous carbon controls (NC)

To understand the broad changes in the Liard hot spring transcriptome during incubations with different lignin substrates, we will first analyze reads that have been mapped with Meiji to our refined genome bins from above, but also less-complete genomic and transcriptomic bins.

To avoid extraneous data parsing, let's simply upload our data file already in the "long" format, factorize the labels to maintain order on the y-axis, then plot as a heatmap to view whole-transcriptome abundance across samples:

```
metatrans.long <- read.table("data/metatranscriptome.counts.long.tsv", sep = "\t")
```

```
metatrans.long$Label <- factorize(metatrans.long$Label)

head(metatrans.long)
```

```
##            GenomeID     variable      value Substrate
## 1 GCF_000381125.1 EPS_DHP_96_01 0.03116213    DHP 96
## 2            LS.5 EPS_DHP_96_01 0.00000000    DHP 96
## 3            NO.1 EPS_DHP_96_01 0.14319252    DHP 96
## 4          MB2.62 EPS_DHP_96_01 0.19562015    DHP 96
## 5            CO.1 EPS_DHP_96_01 0.49088305    DHP 96
## 6        L.E.AP.42 EPS_DHP_96_01 1.00000000    DHP 96
##                              Label
## 1 GCF_000381125.1 (s__Caldimonas)
## 2              LS.5 (g__CV81)
## 3    NO.1 (g__Novosphingobium_A)
## 4    MB2.62 (f__Magnetospiraceae)
## 5            CO.1 (g__Rubrivivax)
## 6          L.E.AP.42 (g__UBA2250)
```

Next we use the sample heatmap plot as above to see if we can find bins that are particularly active in our various lignin compound incubations:

```
trans.heatmap <- ggplot(data = metatrans.long, mapping = aes(x = variable, y = Label, fill = value)) +
  geom_tile(color = "grey50") +
  scale_fill_gradient(low = "#FFFFFF", high = "#012345", na.value = "#FFFFFF") +
  scale_y_discrete(drop=FALSE) +
  theme(axis.text = element_text(size = 8, vjust = 0.4),
        axis.text.x = element_blank(),
        axis.title = element_blank(),
        strip.background = element_blank()) +
  facet_grid(.~Substrate, scales = "free_x")
trans.heatmap
```
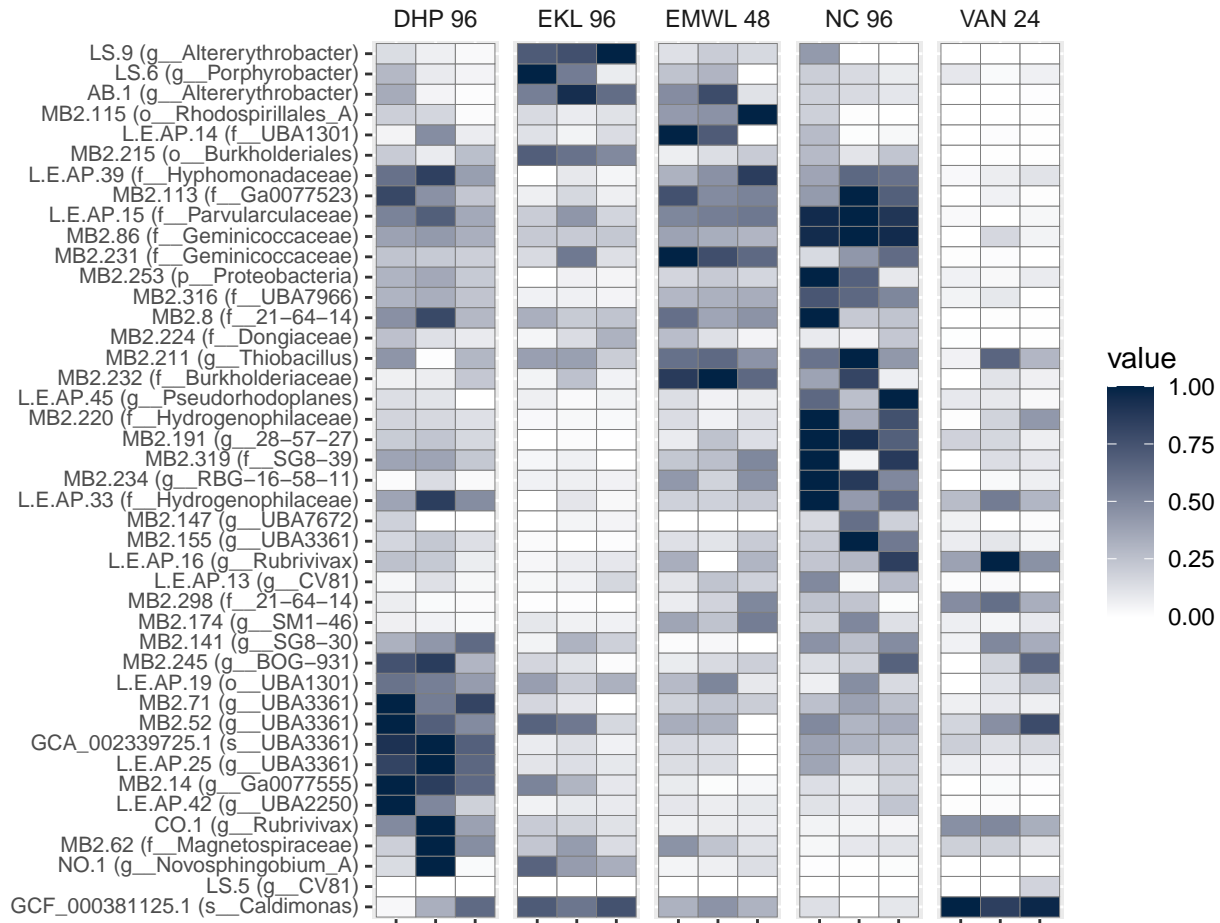
# Section 8. Circular genome plots with gene expression data

One more interesting way we can plot the metatranscriptomics data is by looking at gene expression in a single genome bin.

We will start by looking at expression patterns of the Chloroflexi MAG L.E.CH.5 grown on Kraft Lignin.

First, prepare the data:

```
#Load the coverage table generated with BBMap
CH5.EKL.COV <- read.table("data/LE.CH5.EKL.cov", sep = "\t", header = TRUE)
cmax <- max(CH5.EKL.COV$cov)

#Load the genome features file (gff) for L.E.CH.5.
# We have already converted the gff to a tab-delimited table for easy loading.
CH5.gff <- read.table("data/LE.CH5.gff", sep = "\t", header = TRUE)

#Turn the genome name into a factor for circlize plotting to work
CH5.gff$seqnames <- factor(CH5.gff$seqnames)
```

Next, we use the genomic plotting functions in *circlize* to plot the expression coverage and genome open reading frames (ORFs). To learn much more about plotting with circlize, please refer to the circlize ebook.

For now, understand that the genomicTrackPlotRegion() function requires a data.frame that it will split into two components. The "region", which includes the first three columns: "seqnames", "start" and "end", and the "values", which includes everything else.

Note that it takes a while for the genomic region plot to complete.

```
#We begin bit initializing the plotting parameters
circos.par(start.degree = 0, track.margin = c(0,0), cell.padding = c(0,0), gap.degree = 5)
circos.initializeWithIdeogram(CH5.gff, plotType = c("axis", "labels"))


#Plotting Gene Expression
circos.genomicTrackPlotRegion(CH5.EKL.COV, stack = FALSE, panel.fun = function(region, value, ...) {
  circos.genomicRect(region, value, ytop.column = 2, ybottom = 0, col = "#F24405", border = "#F24405")
}, track.height = 0.2, bg.border = "grey80", ylim = c(0,cmax))


#Coding Regions
circos.genomicTrackPlotRegion(CH5.gff, stack = FALSE, panel.fun = function(region, value, ...) {
  circos.genomicRect(region, value,
                  ytop = value[,2], ybottom = value[,3], #Plots ORF positions by strand
                  col = "grey40", border = NA)
}, track.height = 0.1, bg.border = "grey50", ylim = c(-1,1))
```
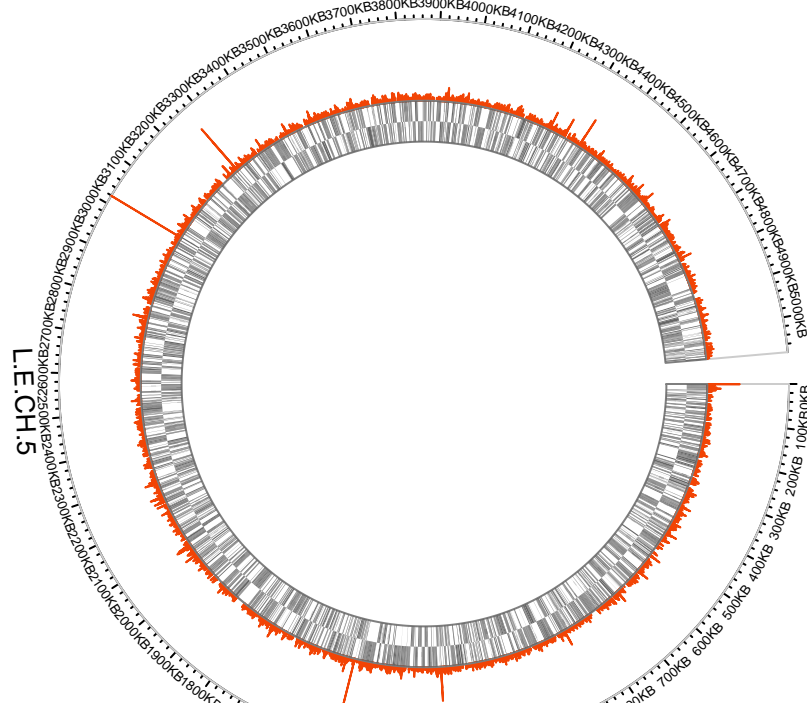
```
circos.clear()
```

That's it! From here we can highlight genes of interest, add labels and draw connections between genome regions. If you would like to peruse these options further, please let the instructor know.

## Summary

The mission of this workshop was to learn how to use genomic phylogeny to organize and make sense of data generated during metagenomic and metatranscriptomic analysis.

These analysis generate a number of distinct data types we would do well to review:

- Genome classification and statistics: To understand the microbial populations in our microbiome, we need to generate information regarding what they are and how trustworthy this information is.

- Count data: the primary data type we worked with was count data, whether it is for a whole-genome or a single ORF. We can apply statistical models to understand genome abundance, gene potential or expression across environments or treatments.

- Sequence data: We did not touch on this in much depth, but by generating distinct shotgun assemblies, we can investigate unique sequences that can result in improved phylogenetic understanding, functional profiling and enzyme discovery.

In this workshop we investigated a rare thermal microbiome, and learned about the variety of ways microbes generate energy in these systems.

I sincerely appreciate your attendance and effort in this workshop and please let me know if you have any questions, comments or ideas!

David Levy-Booth, University of British Columbia (dlevyboo@mail.ubc.ca) March 17-19, 2020