R Intro

Jesús N. Pinto-Ledezma and Jeannine Cavender-Bares

The main goal of this practice is to present basic aspects for anyone to be free about initial fear and start using R to perform data analysis. Every learning process become more effective when theory is combined with practice, so we stringly recommend that you follow the exercises in this short tutorial at the same time that you run the commands on your computer, and not just read it passively.

Why R?

R is a language and a statistical programming environment and graphics or also called an **"object oriented programming"**, which means, that using R involves the creation and manipulation of objects on a screen, where the user has to say exactly what they want to do rather than simply press a button **(black box paradox)**. So, the main advantage of R is that the user has control over what is happening and also the fully understanding of what he/she want before performing any analysis.

With R is possible to manipulate and to analyze data, make graphics and write since small commands to entire programs. Basically, R is the open version of the S language, created by Bell's Lab in 1980. Interestingly, S language is super popular among different areas of sciences and is the base for comercial products such us, SPSS, STATA, SAS among others. Thus, if we have to add another advantage of R, is that R is **open language and free**!

There are different sources and web-pages with a lot of information about R, most of them are super useful and can be found at DataCamp (https://www.datacamp.com/), CRAN (https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf), R Tutorial (http://www.r-tutor.com/r-introduction).

Also, when we are reporting our results in the form of a report, scientific paper or any kind of document, we need to cite the software used, the easiest to cite R is using the internal function **citation()**.

First steps

First that all, we need to know about WHERE are we working at? That is our Working directory.

```r
getwd()
```

```
## [1] "/Users/jesusn.pinto-ledezma/Documents/GitHub/BiodiversitySciences"
```

If the working directory is not the correct, we just need to order R to **SET** the correct address.

Ok, we are now in the correct place, so we can continue with the practice.

Directory structure

For training purposes, we will create a **directory structure** where the main folder is our current working place, so we will create a series of **subfolders** where we store, the data, the scripts and whatever we want… To do that we will use the function **dir.create()**. Lets practice!

To check if the subfolders were created, just use the function **dir()**, this simple function will print in the console the name of the files that are currently in your working directory.

We can SET our working directory into one of the subfolders that we created using the function setwd()

```r
setwd("Results")
```

However, for practicity is super-ultra-mega recommendable to work in a **MAIN FOLDER**, so go back to the previous folder or main folder we just need to use again the function **setwd()**, instead of using a folder name, we will use simple two dots, yes two dots **..**. This simple operation will return to the main folder.

```r
setwd("..")
```

The importance of the question mark "?" or the help function

Maybe, the most important (at least for Jesús) function of R is the **help** or **?**. Using help or the question mark, we can ask to R about almost anything (saddly we can't order pizza, yet)…so, lets practice!

```r
help("logarithm")
?log
??log
```

Other important and useful functions in R, are: **head()**, **tail()**, **dim()**, **str**, **summary()**, **names()**, **class()**, **rm()**, **save.image**, **saveRDS and readRDS**, **load**, **source**, all these simple functions help us to understand the data with we are working.

Objects: creation and manipulation

In R you can create and manipulate different data, from a simple numeric vector to complex spatial and/or phylogenetic data frames. The main six kinds of objects that you can create and manipulate in R, are: vector, factor, matrix, data frame, list and functions.

So, lets start with the first object, the **Vector**.

Vector

Within vector exist three types: numeric, character and logic.

Numeric vector

IMPORTANT R is case sensitive, so we need to put atention when you name the objects.

```r
a <- 10

b <- c(1, 2, 3, 4, 5)

seq_test <- seq(from = 1, to = 20, by = 2) # Here is a sequence of numbers from 1 to 20, every two numb

x = seq(10, 30) # This is a sequence from 10 to 30. What is the difference with the previous numeric ve

sample(seq_test, 2, replace = T) # Sort two numbers within the object seq_test

rep_test <- rep(1:2, c(10, 3)) # Repeat the number one, ten times and the number 2, three times

ex <- c(1:10) # Create a sequence of 1 to 10

length(ex) # Lenght of the object example
```

```
aa <- length(ex) # What we are doing in here?

str(seq_test) # Look at the structure of the data
```

Character vector

```
research_groups <- c(Jeannine = "Oaks", Jesus = "Furnariides", Kirsten = "Fossils")

research_groups

str(research_groups)
```

You can try to create a different character vector, for example, using the names of your peers.

Logic vector

This kind of vector is super useful when the purpose is to create or build functions. The elements of a logic vector are **TRUE, FALSE, NA** (not available).

```
is.factor(ex) # It is a factor? (FALSE)

is.matrix(ex) # It is a matrix? (FALSE)

is.vector(ex) # It is a vector? (TRUE)

a < 1   # 'a' is lower than 1? (FALSE)
a == 1   # 'a' is equal to 1? (TRUE)
a >= 1   # 'a' is higher or equal to 1? (TRUE)
a != 2   # the object 'a' is different of two? (TRUE) (!= negation)
```

Factor

A factor is useful to create categorical variables, that is very common in statistical analysese, such us the Anova.

```
data <- factor(c("small", "medium", "large"))

is.factor(data) # Check if the object is correct.
```

Matrix

A matrix is bidimensional arrangement of **vectors**, where the vectors need to be of the same type, that is, two or more numeric vectors, or two or more character vectors.

```
matx <- matrix(1:45, nrow = 15)
rownames(matx) <-  LETTERS[1:15]
colnames(matx) <- c("Sample01", "Sample02", "Sample03")

matx # Inspect the matrix
class(matx) # Ask which kind of data is?
matx[, 1] # We can use brackets to select a specific column
matx[1, ] # We can use brackets to select a specific row
```

```r
head(matx)
tail(matx)
#fix(matx)
str(matx)
summary(matx)
```

In general when we are exploring our data for example using **head()** the function will return only the 6 first rows of our matrix, however, we can add another argument into the function. For example, **head(matx, 10)**, just add the number 10 after the comma adn is possible to see the first 10 lines. This simple operation is useful specially when our matrix is large **>500 rows**.

Data frame

The difference between a matrix and a data frame is that a data frame can handle different types of vectors. You can explore more about the data frames asking R **?data.frame**.

```r
df <- data.frame(species = c("rufus", "cristatus", "albogularis", "paraguayae"), habitat = factor(c("fo

class(df)
matx2 <- as.data.frame(matx) # We can also transform our matrix to a data frame
class(matx2)
str(df)
#fix(df)
#edit(df)
```

List

The list is an object that consist of an assembly of objects sorted in a hierarquical way. Here we will use the data previously created.

```r
lst <- list(data, df, matx)

str(lst)
class(lst)
```

Now, inspect the objects thar are stored into our object **lst**. To do this, we just need to use two brackets [[]].

```r
lst[[1]]
lst[[2]]
lst[[3]]
```

Install and load packages

Although R is a programing language, it is also possible to use different auxiliary packages that are available for free to download and to install in our computers. Install new packages into R is easy and just need a simple function **install.packages()**. For more information of how to install new packages, you just need to ask R, using **?install.packages**

```r
install.packages("PACKAGE NAME")
```

The reverse function is **remove.packages()**.

Most of the time, we do not remember if we already have a package installed in our computer, so if we are tired and do not want to go to our R folder and check is the package is installed, we can use the next command.

```
if ( ! ("PACKAGE NAME" %in% installed.packages())) {install.packages("PACKAGE NAME", dependencies = T)}
```

To load an installed package you can just type, **library() or require()**

```
library("PACKAGE NAME")
require("PACKAGE NAME")
```

R as a calculator

R can be used as a calculator, for example, we can use the information created before to make some arithmetic operations.

```
b[4]+seq_test[10]
b[4]*seq_test[10]

seq_test[5]/df[3, 3]
matx[, 3][4]-df[4, 4]

seq_test^7
seq_test*7
seq_test+7
seq_test-7

mean(seq_test)
max(seq_test)
min(seq_test)
sum(seq_test)
log(seq_test)
sqrt(seq_test)

cor(matx[, 1], matx[, 2])
```

Data import/export

As indicated before, in R you can handle different information (from vector to data frames) and basically most of our data is stored in a Excel spreadsheet or in files that have the extension of **.csv** (comma-separated values file) or **.txt** (Text X Text or text file that contains unformatted text).

Most of these files are imported in R are **data frames**, but, as we were practicing, we now have the tools to handle or transfor the information into different objects.

The function to import data to R is simple **read.table()** or **read.csv()**, and using these simple function, you can import the data and transform in other kind or objects So, lets practice!

```
dat <- read.table("Data/Sample.txt")

dat2 <- read.table("Data/Sample.txt", row.names = 1, header = TRUE)

dat3 <- read.csv("Data/Sample.csv")
```

```
class(dat)
class(dat2)
class(dat3)

dat3Sample <- dat3[1:50, 1:4]
dim(dat3Sample)

dat4 <- na.omit(as.matrix(read.csv("Data/Sample.csv", row.names = 1, header = TRUE)))
class(dat4)
head(dat4, 10)
dat4[1:20, 1:4]
```

You can also import your data using the same functions, but without specifying the address. Notice that we do not recomend this procedure as you can't control the **directory structure**, but is useful when you just are exploring data.

You can also save your data from R using the function **write.table** or **write.csv**. Lets save the dat3Sample. Notice that always we need to specify the correct address, in our case we will save the data in the subolder **Data**.

```
is.na(dat3Sample)
write.csv(dat3Sample, file = "Data/dat3Sample.csv")
```

Phylogenetic data

To study biodiversity is important to first understand the data and one common data used now is the phylogentic data or phylogenetic trees that describe the evolutionary relationships between and among lineages. From here until the end of this short tutorial we will try to explain the basics of how to import/export and handle phylogenetic information. Extra information you can find at https://www.r-phylo.org/wiki/HowTo/Table_of_Content e http://www.mpcm-evolution.org/practice/online-practical-material-chapter-2.

Formats

The two most common formats in which the phylogenies are stored are the Newick and Nexis (Maddison et al. 1997).

The Newick format represent the phylogenetic relationships as **"(", "," and ":"**, so the species relationship can be represented as follow:

```
((A:10,B:9)D:5,C:15)F;
```

Using this notation, the parenthesis link the lineages to a specific node of the tree and the comma **","** separate the lineages that descend from that node. The colon punctuation **":"** can be used after the name of the node and the subsequent numeric values represent the branch lenght. Finally, the semicolon punctution **";"** indicate the end of the phylogenetic tree.

Now we can see how this format works, but first, check if we have the R packages for this purpose.

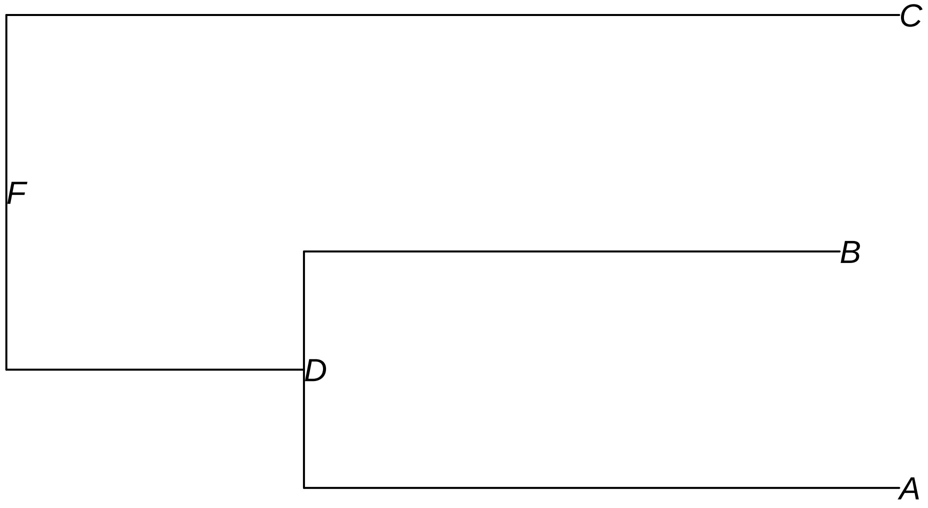
```
if ( ! ("ape" %in% installed.packages())) {install.packages("ape", dependencies = T)}
```

```
require(ape)
```

```
## Here we will create a phylogenetic tree in Newick format
newick_tree <- "((A:10,B:9)D:5,C:15)F;"
```

```r
## Read the tre
newick_tree <- read.tree(text = newick_tree)

## And now we can plot the phylogentic tree
plot(newick_tree, show.node.label = TRUE)
```



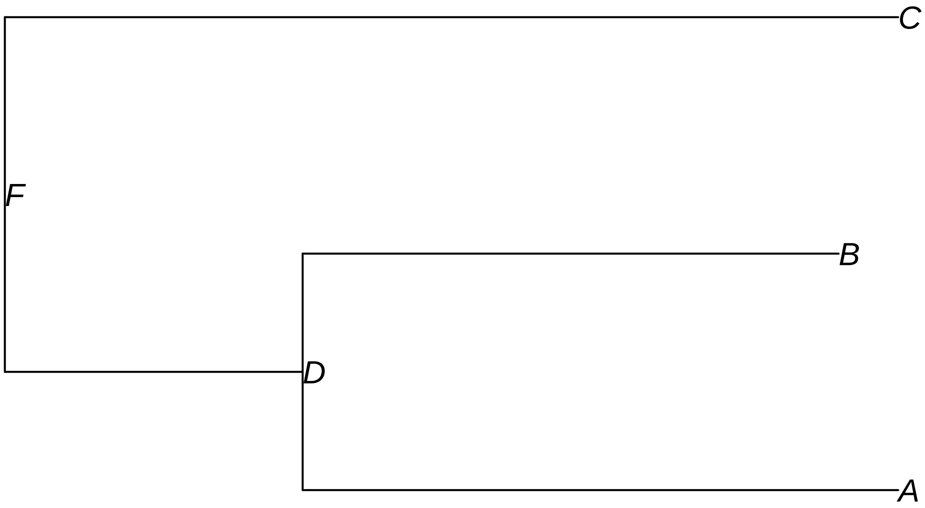
The other format is the **Nexus**, and after some time we can say the Nexus format have more flexibility for working. An example of a Nexus format is as follow:

```
#NEXUS
BEGIN TAXA;
DIMENSIONS NTAXA=3;
TaxLabels A B C;
END;
BEGIN TREES;
TREE=((A:10,B:9)D:5,C:15)F;
END;
```

```r
## First create a Nexus file in the working directory
cat(
 "#NEXUS
 BEGIN TAXA;
 DIMENSIONS NTAXA=3;
 TaxLabels A B C;
 END;
 BEGIN TREES;
 TREE=((A:10,B:9)D:5,C:15)F;
 END;",
file = "Data/Nexus_tree.nex"
)
```

```r
## Now read the phylogenetic tree, but look that instead of using read.tree we are using read.nexus
nexus_tree <- read.nexus("Data/Nexus_tree.nex")
```

```r
## lets plot the example
plot(nexus_tree, show.node.label = TRUE)
```



Now lets inspect our phylogenetic trees.

```
str(nexus_tree)
```

```
## List of 5
##  $ edge       : int [1:4, 1:2] 4 5 5 4 5 1 2 3
##  $ edge.length: num [1:4] 5 10 9 15
##  $ Nnode      : int 2
##  $ node.label : chr [1:2] "F" "D"
##  $ tip.label  : chr [1:3] "A" "B" "C"
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

```
nexus_tree$tip.label
```

```
## [1] "A" "B" "C"
```

If we want to know about the brach lenght of the tree we justneed to select **edge.lenght**

```
nexus_tree$edge.length
```

```
## [1]  5 10  9 15
```

An important component of a phylo object is the matrix called **edge**. In this matrix, each **row** represent a **branch** in the tree and the **first column** shows the index of the ancestral node of the branch and the **second column** shows the descendant node of that branch. Lets inspect!

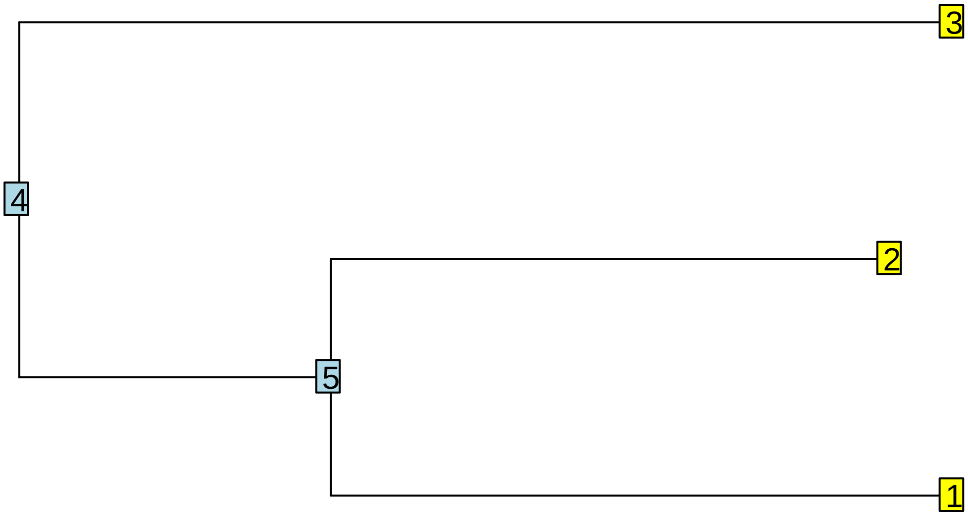
```
nexus_tree$edge
```

```
##      [,1] [,2]
## [1,]    4    5
## [2,]    5    1
## [3,]    5    2
## [4,]    4    3
```

We know, is a little hard to follow even with small trees as the example, but, if we plot the information is easy to understand.

```
# Lets plot the tree
plot(nexus_tree, show.tip.label = FALSE)
# Add the internal nodes
nodelabels()
```

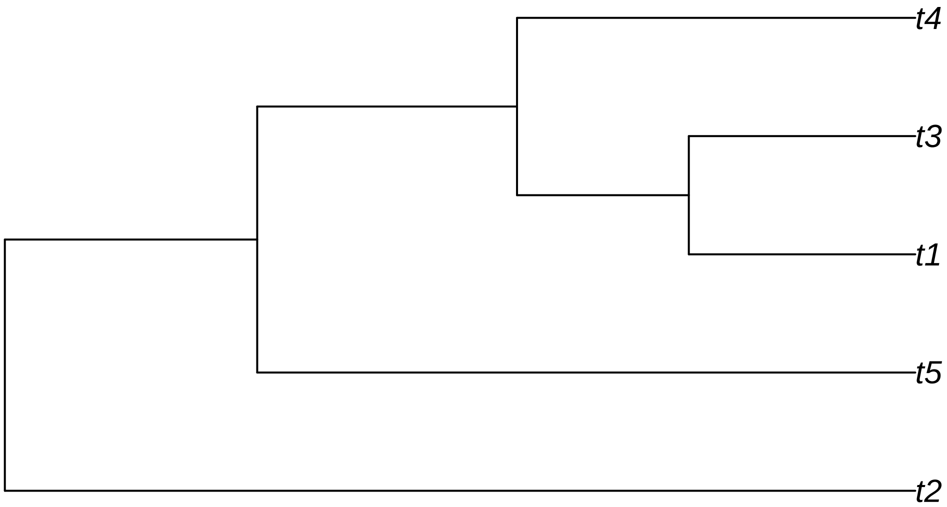
```
# Add the tips or lineages
tiplabels()
```



Finally, the phylogenies also can be imported in form of a list and in phylogenetic omparative methods this list of phylogenies is called **multiPhylo**, and we can import/export these multiPhylos in the two formats.

```
# Simulate 10 phylogenies, each one with 5 species
multitree <- replicate(10, rcoal(5), simplify = FALSE)
# Store the list of trees as a multiPhylo object
class(multitree) <- "multiPhylo"

# Plot a single tree from the 10
plot(multitree[[1]])
```



```
# exportando as filogenias num único arquivo Newick.
write.tree(phy = multitree, file = "Data/multitree_example.txt")
multitree_example <- read.tree("Data/multitree_example.txt")
multitree
```

```
## 10 phylogenetic trees
```

We have covered basic aspects of R, from exploring and managing object to import/export data. We hope that this short tutorial can be useful not only for the **Biodiversity Sciences** course, but for your especific projects. Remember, practice, practice, practice!

References

Maddison, D. R., Swofford, D. L. and Maddison, W. P. (1997). NEXUS: An Extensible File Format for Systematic Information. Systematic Biology 46, 590.