Recap From Last Time:

- **UNIX pipes and redirects**: How UNIX commands can be combined to generate flexible solutions to data manipulation tasks.
- **UNIX commands**: Further exploration of the 22 key UNIX commands that you will use during ~95% of your future UNIX work.
- **Jetstream**: Starting up instances; ssh access from your Terminal application; Demoed installing and running bioinformatics software for a genome scale annotation.
- **Cloud computing**: Many bioinformatic tasks require large amounts of computing power and can’t realistically be run on your own machine. These tasks are best performed using remote computers or cloud computing, which can only be accessed through a shell.

Today’s Learning Goals

- Familiarity with R’s basic syntax.
- Familiarity with major R data structures.
- Understand the basics of using functions.
- Be able to use R to read and parse comma-separated (.csv) formatted files ready for subsequent analysis.
- Appreciate how you can use R scripts to aid with reproducibility.

Side-Note: StackOverflow is your friend!
What is R?

R is a freely distributed and widely used programming language and environment for statistical computing, data analysis and graphics.

R provides an unparalleled interactive environment for data analysis.

It is script-based (i.e. driven by computer code) and not GUI-based (point and click with menus).
What is **R**?

R is a freely distributed and widely used programming language and environment for statistical computing, data analysis, and graphics.

R provides an unparalleled interactive environment for data analysis. It is script-based (i.e. driven by computer code) and not GUI-based (point and click with menus).

Type "R" in your terminal

This is the R prompt

What R is **NOT**

- A performance optimized software library for incorporation into your own C/C++ etc. programs.
- A molecular graphics program with a slick GUI.
- Backed by a commercial guarantee or license.
- Microsoft Excel!
What about Excel?

- Data manipulation is easy
- Can see what is happening
- **But:** graphics are poor
- Looping is hard
- Limited statistical capabilities
- Inflexible and irreproducible
- There are many many things Excel just cannot do!

**Rule of thumb:** Every analysis you do on a dataset will have to be redone 10–15 times before publication. Plan accordingly!

Why use R?

- Productivity
- Flexibility
- Designed for data analysis

---

**IEEE 2016 Top Programming Languages**

<table>
<thead>
<tr>
<th>Language Rank</th>
<th>Types</th>
<th>Spectrum Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. C</td>
<td>📊</td>
<td>100.0</td>
</tr>
<tr>
<td>2. Java</td>
<td>📊</td>
<td>98.1</td>
</tr>
<tr>
<td>3. Python</td>
<td>📊</td>
<td>98.0</td>
</tr>
<tr>
<td>4. C++</td>
<td>📊</td>
<td>95.9</td>
</tr>
<tr>
<td>5. R</td>
<td>📊</td>
<td>87.9</td>
</tr>
<tr>
<td>6. C#</td>
<td>📊</td>
<td>86.7</td>
</tr>
<tr>
<td>7. PHP</td>
<td>📊</td>
<td>82.8</td>
</tr>
<tr>
<td>8. JavaScript</td>
<td>📊</td>
<td>82.2</td>
</tr>
<tr>
<td>9. Ruby</td>
<td>📊</td>
<td>74.5</td>
</tr>
<tr>
<td>10. Go</td>
<td>📊</td>
<td>71.9</td>
</tr>
</tbody>
</table>

R and Python: The Numbers

- R is the “lingua franca” of data science in industry and academia.
- Large user and developer community.
- As of Aug 1st 2016 there are 8811 add on R packages on CRAN and 1211 on Bioconductor - more on these later!
- Virtually every statistical technique is either already built into R, or available as a free package.
- Unparalleled exploratory data analysis environment.

<table>
<thead>
<tr>
<th>Modularity</th>
<th>Core R functions are modular and work well with others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactivity</td>
<td>R offers an unparalleled exploratory data analysis environment</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Access to existing tools and cutting-edge statistical and graphical methods</td>
</tr>
<tr>
<td>Support</td>
<td>Extensive documentation and tutorials available online for R</td>
</tr>
<tr>
<td>R Philosophy</td>
<td>Encourages open standards and reproducibility</td>
</tr>
</tbody>
</table>

Modularity | Core R functions are modular and work well with others
---|---------------------------------------------
Interactivity | R offers an unparalleled exploratory data analysis environment
Infrastructure | Access to existing tools and cutting-edge statistical and graphical methods
Support | Extensive documentation and tutorials available online for R
R Philosophy | Encourages open standards and reproducibility
Modularity

R was designed to allow users to interactively build complex workflows by interfacing smaller ‘modular’ functions together.

get.seq() → hminer() → pdbain() → pdbfit() → pca() → plot()

An alternative approach is to write a single complex program that takes raw data as input, and after hours of data processing, outputs publication figures and a final table of results.

All-in-one custom ‘Monster’ program

‘Scripting’ approach

Another common approach to bioinformatics data analysis is to write individual scripts in Perl/ Python/Awk/C etc. to carry out each subsequent step of an analysis.

1. ...
2. ...
3. ...

This can offer many advantages but can be challenging to make robustly modular and interactive.

Interactivity & exploratory data analysis

Learning R will give you the freedom to explore and experiment with your data.

“Data analysis, like experimentation, must be considered as a highly interactive, iterative process, whose actual steps are selected segments of a stubbily branching, tree-like pattern of possible actions”. [J. W. Tukey]

Interactivity & exploratory data analysis

Learning R will give you the freedom to explore and experiment with your data.

“Data analysis, like experimentation, must be considered as a highly interactive, iterative process, whose actual steps are selected segments of a stubbily branching, tree-like pattern of possible actions”. [J. W. Tukey]

Bioinformatics data is intrinsically high dimensional and frequently ‘messy’ requiring exploratory data analysis to find patterns - both those that indicate interesting biological signals or suggest potential problems.
How do we use R?

Two main ways to use R

1. Terminal
2. RStudio
We will use **RStudio** today

Let's get started…

Some simple R commands

1. \( 2 + 2 \)
   - Result of the command: \( 4 \)

2. \( 3^2 \)
   - Result: \( 9 \)

3. \( \sqrt{25} \)
   - Result: \( 5 \)

4. \( 2 \times (1 + 1) \)
   - Result: \( 4 \)

5. \( 2 \times 1 + 1 \)
   - Result: \( 3 \)

6. \( \exp(1) \)
   - Result: \( 2.718282 \)

7. \( \log(2.718282) \)
   - Result: \( 1 \)

8. \( \log(10, \text{base}=10) \)
   - Result: \( 1 \)

9. \( \log(10 \text{, base} = 10) \)
   - Result: \( 1 \)

10. \( x = 1:50 \)
    - Command: \( \text{plot(x, sin(x))} \)

Does your plot look like this?
Learning a new language is hard!

Key point: You need to visualize your data!

Options: ?plot, ?plot.default
Error Messages

Sometimes the commands you enter will generate errors. Common beginner examples include:

- Incomplete brackets or quotes e.g. 
  
  ```
  ((4+8)*20 <enter>
  +
  ```
  This returns a + here, which means you need to enter the remaining bracket - R is waiting for you to finish your input.
  Press <ESC> to abandon this line if you don’t want to fix it.

- Not separating arguments by commas e.g. 
  ```
  plot(1:10 col="red")
  ```

- Typos including miss-spelling functions and using wrong type of brackets e.g. 
  ```
  exp[4]
  ```

Your turn!

http://tinyurl.com/bggn213-rintro

If you have done the introductory DataCamp course then feel free to jump to section #3 Object Assignment

Topics Covered:

- Calling Functions
- Getting help in R
- Vectors and vectorization
- Workspace and working directory
- RStudio projects
Vectors

• Vectors are the most basic data structure in R
• All elements of a vector must be the same type

| dbl_var <- c(1, 2.5, 4.5) |
| log_var <- c(TRUE, FALSE, T, F) |
| chr_var <- c("these are", "some", "strings") |

• When you attempt to combine different types they will be coerced to the most flexible type.

| var <- c(1, "G", "4", 0.05, TRUE) |

Names

• You can name a vector in several ways:
  • When creating it:  
    ```
    x <- c(a = 1, b = 2, c = 3)
    ```
  • By modifying an existing vector in place:
    ```
    x <- 1:3; names(x) <- c("a", "b", "c")
    ```
  • You can then use the names to access (subset) vector elements:
    ```
    x["a"]
    ```

Why is this useful?

• Because if you know the name (i.e. your label) then you don’t have to remember which element of a vector the data you are after was stored in.

Consider this fictional example:

```
> grades <- c(alice=80, barry=99, chandra=60, chris=100)
> grades["barry"]
  Barry
    99
> which.max(grades)
  chris
    4
> sort(grades)
  chandra alice barry chris
    60  80  99  100
```

What would happen?

```
1. > x <- 1:3; names(x) <- c("a", "b", "c", "d")
2. > x <- 1:3; names(x) <- 1:3; x[3]
3. > x["3"]
```
R has many data structures

These include:
- vector
- data frame
- list
- matrix
- factors

**data.frame**

- *data.frame* is the *de facto* data structure for most tabular data and what we use for statistics and plotting with *ggplot2* - more on this later!
- Arguably the most important R data structure
- Data frames can have additional attributes such as *rownames()* and *colnames()*, which can be useful for annotating data, with things like *subject_id* or *sample_id*

**data.frame** continued...

- Created with the function *data.frame()*
  
    ```r
dat <- data.frame(id = letters[1:10], x = 1:10, y = 11:20)
    ```

- Or more commonly when reading delimited files (i.e. importing data) with the functions
  *read.csv*, *read.table*, *read_xlsx* etc...
  
    ```r
dep <- read.csv2("http://bio3d.uib.no/data/pdb_deposition2.csv")
    ```

- R Studio can do this for you via:
  **File > Import Dataset > From CSV**

**Useful data.frame Functions**

- *head()* - and *tail()* shows first 6 rows and last 6 rows respectively
- *dim()* - returns the dimensions (i.e. number of rows and columns)
- *nrow()* and *ncol()* returns the number of rows and columns separately.
- *rownames()* and *colnames()* - shows the names attribute for rows and columns
- *str()* - returns the structure including name, type and preview of data in each column
Key Points

• R’s basic data types are **logical**, **character**, **numeric**, integer and complex.

• R’s basic data structures include **vectors**, lists, **data frames**, matrices and factors.

• Objects may have attributes, such as **name**, **dimension**, and **class**.

---

**R scripts**

• A simple text file with your R commands (*e.g.* lecture7.r) that contains your R code for one complete analysis

• **Scientific method**: complete record of your analysis

• **Reproducible**: rerunning your code is easy for you or someone else

• In RStudio, select code and type `<ctrl+enter>` to run the code in the R console

• **Key point**: Save your R script!

---

**Side-note: Use the code editor for R scripts**

1. Code Editor
2. R Console
3. Workspace and History
4. Plots and files

**Side-note: RStudio shortcuts**

- Sends current line or selection to console (faster to type: `command/ctrl+enter`)
- Sends entire file to console
- Re-send the lines of code you last ran to the console (useful after edits)

Other RStudio shortcuts:
- Up/Down arrows (recall cmds)
- `Ctrl + 2` (move cursor to console)
- `Ctrl +1` (move cursor to editor)
**Rscript: Third way to use R**

1. Terminal
2. RStudio
3. **Rscript**

*From the command line!*  
>`Rscript --vanilla my_analysis.R`  
  # or within R: `source(my_analysis.R)`

**Side-Note: R workspaces**

- When you close RStudio, **SAVE YOUR .R SCRIPT**
- You can also save data and variables in an R workspace, but this is generally not recommended
- Exception: working with an enormous dataset
- Better to start with a clean, empty workspace so that past analyses don’t interfere with current analyses
- `rm(list = ls())` clears out your workspace
- You should be able to reproduce everything from your R script, so **save your R script, don’t save your workspace!**

**Learning Resources**

- **TryR.** An excellent interactive online R tutorial for beginners.  
- **RStudio.** A well designed reference card for RStudio.  
  [https://help.github.com/categories/bootcamp/](https://help.github.com/categories/bootcamp/)
- **DataCamp.** Online tutorials using R in your browser.  
  [https://www.datacamp.com/](https://www.datacamp.com/)
  [http://r4ds.had.co.nz/](http://r4ds.had.co.nz/)
Key Points

• R’s basic data types are logical, character, numeric, integer and complex.

• R’s basic data structures include vectors, lists, data frames, matrices and factors.

• Objects may have attributes, such as name, dimension, and class.

• DataCamp, StackOverflow and help() are your friends.

Final Knowledge Check!

• What is R and why should we use it?

• Familiarity with R’s basic syntax.

• Familiarity with major R data structures namely vectors and data.frames (with more on lists and matrices next day).

• Understand the basics of using functions (arguments, vectorization and re-cycling).

• Be able to use R to read and parse comma-separated (.csv) formatted files ready for subsequent analysis.

• Appreciate how you can use R scripts to aid with reproducibility.

http://swcarpentry.github.io/r-novice-inflammation/

Sections: 1, 11 & 12 only!
Help from within R

• Getting help for a function
  > help("log")
  > ?log

• Searching across packages
  > help.search("logarithm")

• Finding all functions of a particular type
  > apropos("log")

[7] "SSlogis" "as.data.frame.logical" "as.logical"
   "as.logical.factor" "dlogis" "is.logical"
[13] "log" "log10" "log1p" "log2" "logLik" "logb"
[19] "logical" "loglin" "plogis" "print.logLik" "qlogis"
   "rlogis"

Optional Exercise

Use R to do the following. Create a new script to save your work and code up the following four equations:

\[
1 + 2(3 + 4)
\]
\[
\ln(4^3 + 3^{2+1})
\]
\[
\sqrt{(4 + 3)(2+1)}
\]
\[
\left(\frac{1+2}{3+4}\right)^2
\]