ETH zürich

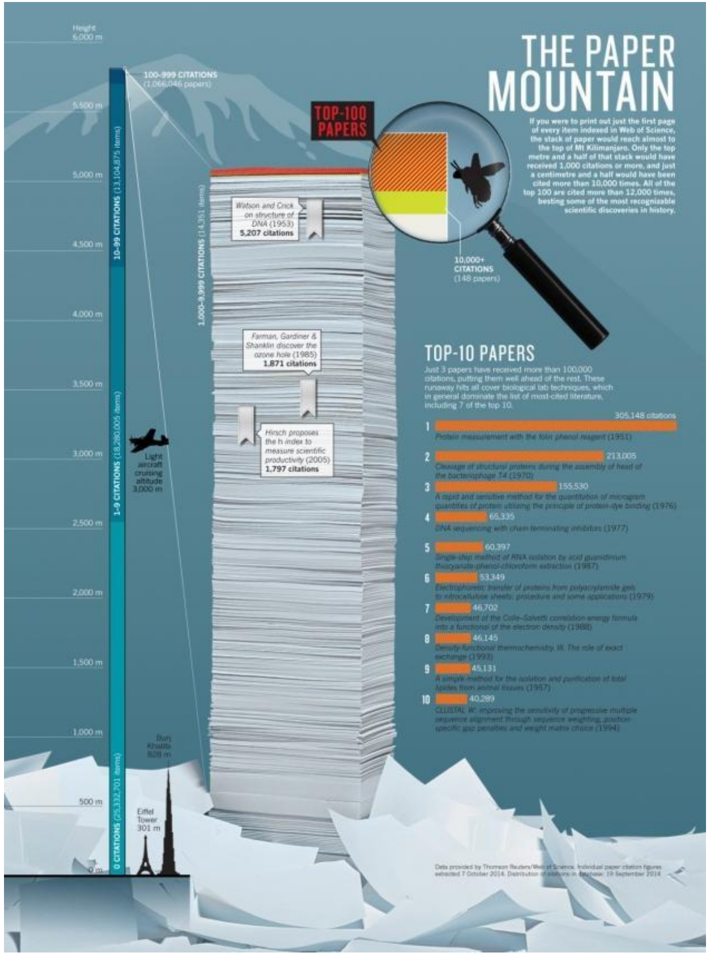
551-1119-00L Microbial Community Genomics

Lecture:
Best practices for data/project/software management

Samuel Miravet-Verde 15-Nov-23

D BIOL

0. Computational biology || Why 'good practices' are required?



If you were to print out just the first page of every item indexed in Web of Science, the stack of paper would reach almost to the top of Mt Kilimanjaro. Only the top metre and a half of that stack would have received 1,000 citations or more, and just a centimetre and a half would have been cited more than 10,000 times. All of the top 100 are cited more than 12,000 times, besting some of the most recognizable scientific discoveries in history.

Just 3 papers have received more than 100,000 citations, putting them well ahead of the rest. These runaway hits all cover biological lab techniques, which in general dominate the list of most-cited literature, including 7 of the top 10.

Data provided by Thomson Reuters/Web of Science. Individual paper citation figures extracted 7 October 2014. Distribution of citations in database: 19 September 2014

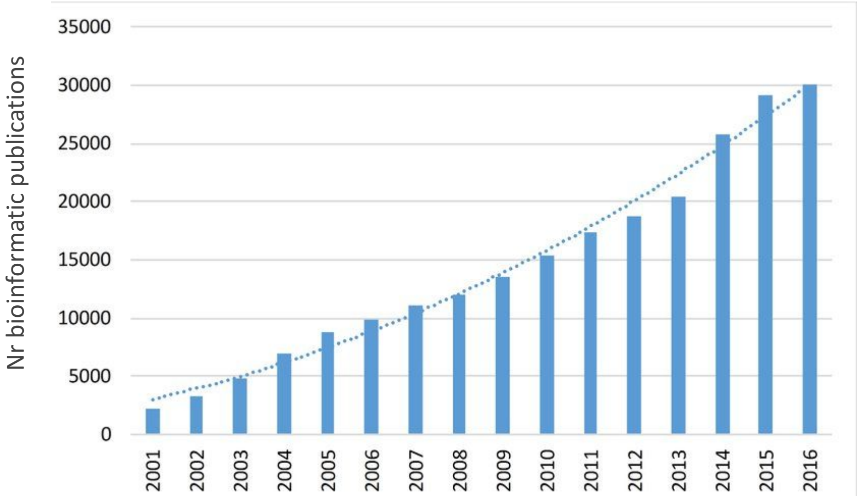
https://www.nature.com/articles/d41586-019-00381-w

ALGORITHM: a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

3 bioinformatics papers in the top 15 most cited papers:

- 10th : ClustalW (1994) (40,000 cit. in 2014)
- 12th : BLAST (1990)
- 14th : PSI-BLAST (1997)

High-reaching tools, we have to ensure bioinformatic tools and analyses done with these tools are properly done



0. Computational biology || Why ‘good practices’ are required?

What's the point of writing good scientific software?

“**Software written by academics** has a reputation of being **poorer quality** than that software written by professional **software developers**”

“Well documented software takes **extra time** and in a competitive academic job market I feel like this is a **luxury**.”

How Not to Be a Bioinformatician

“Stay low level at every level. Develop your code by anecdote: avoid planning phases, requirement analysis exercises or any structure to your code.”

“If you create the application, **make it difficult to build and interpret**. Have plenty of hidden dependencies and bizarre variables.“

1. The data analysis process || Basic workflow to analyse data

1. Define the question

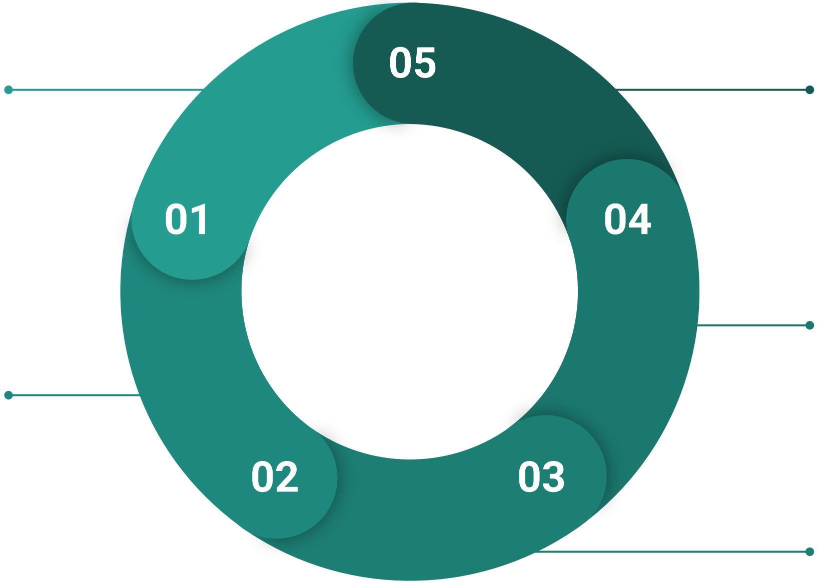
Together with supervisor and/or collaborators

Plan experiments and potential roadblocks to deal during development

Consider the assumptions you will be taking during your analysis

2. Data collection

Either from samples or public resources (there are MANY!)



5. Sharing results

Summarize remarkable results, helped by text, tables and visualizations

Keep track of intermediary and non-reported results

4. Analyze the data

Apply the planned analyses.

Fail Early, Fail Fast → be flexible and adjust to data requirements

3. Clean Data

Identify outliers, non-annotated data, etc. and remove it

Discarded data can still be used for other purposes (e.g., controls)

Good enough practices in scientific computing | PLOS Computational Biology

1. The data analysis process || Structuring a working directory

Structuring and keeping your working directory organized is essential to ensure **reproducibility**…

… Also for your mental health (and possible collaborators or future contributors!)

A good (and flexible) example:

```
.
├── config.yml
├── data
├── envs
├── LICENSE
├── README.md
├── reports
├── results
├── src
├── start_project.sh
└── workflows
```

[MiqG/project_template: Template to give structure to new projects from the start (github.com)](https://github.com/MiqG/project_template)

For your projects:

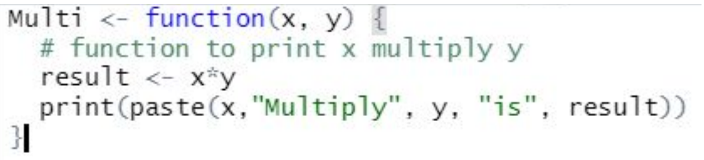
```
.
├── data
├── reports
├── results
├── scripts
└── analysis.R
```

General:
meaningful naming for files

- **Data**: original source of information, also processed tables
- **Reports**: documents summarizing results
- **Results**: figures, final tables
- **Scripts**: source code of functions and procedures you run on the data
- **Analysis**: your main document including all the analysis workflow

2. Documenting and reporting || Good practices when coding

- **Commenting** code and **describing** functions (#)



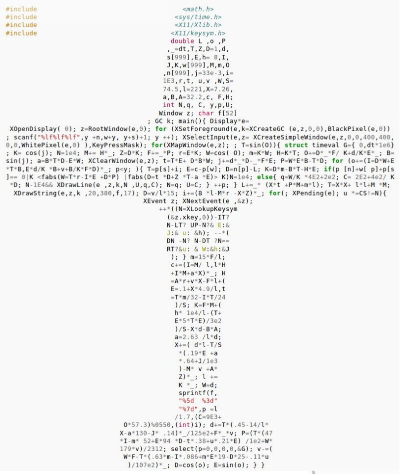
```
Multi <- function(x, y) {
  # function to print x multiply y
  result <- x*y
  print(paste(x,"Multiply", y, "is", result))
}
```

- Conventionally is advisable to write a **main function** working as workflow schema and calling more complex functions.

- **Be expressive,** maybe what you see very clear is not trivial for a different developer.

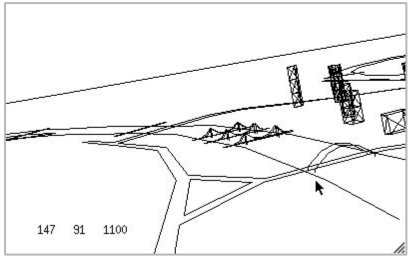
[Best Practices for Scientific Computing]

- **Keep it simple** and **contained**, we work for usability and accessibility. Very elaborated pieces of code can be unreadable.

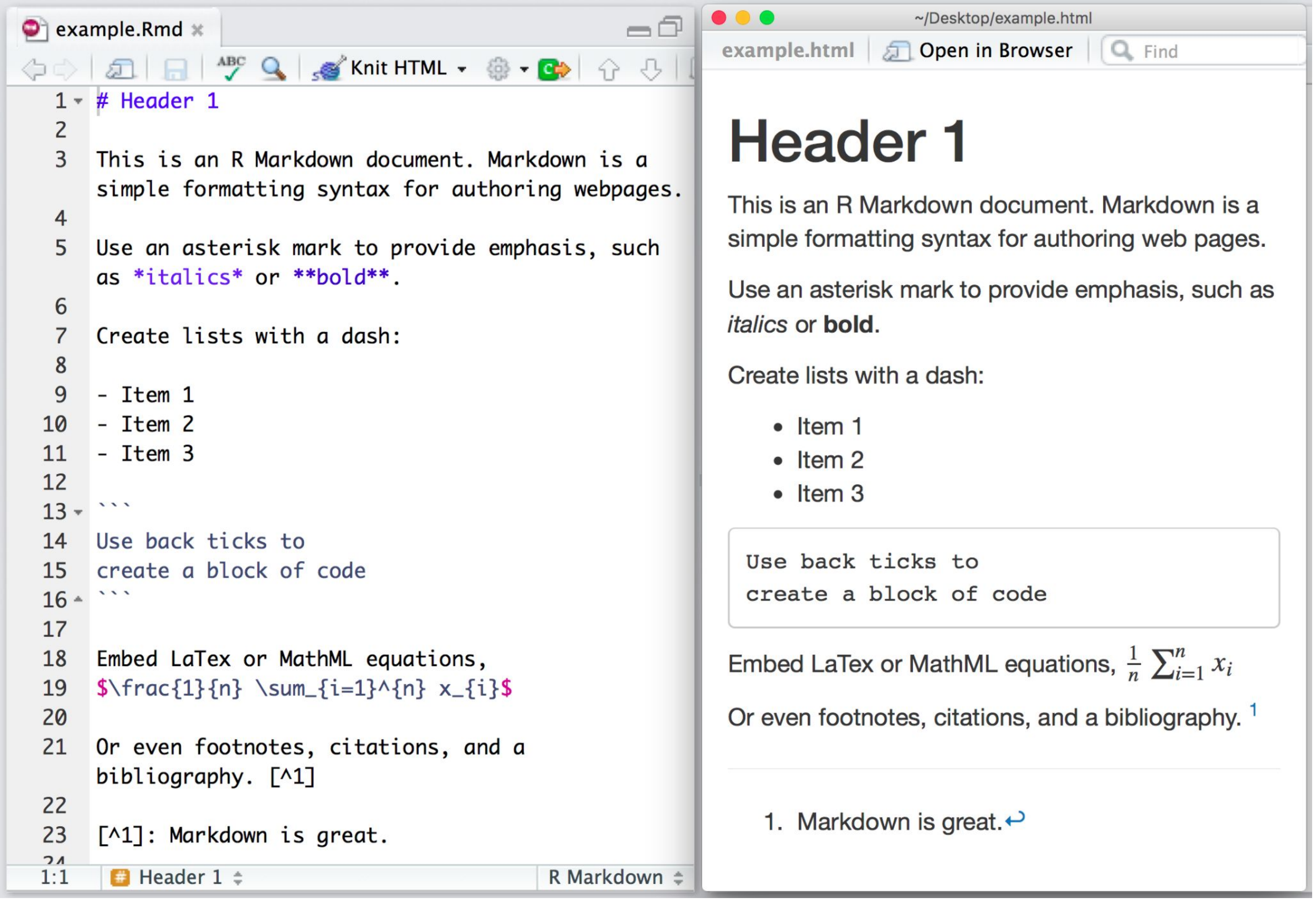


Yes, this is a program… winner IOCCC in 1998

… in fact, it codes a flight simulator



2. Documenting and reporting || Embedding code in markdown



~~~
# Header 1

This is an R Markdown document. Markdown is a simple formatting syntax for authoring webpages.

Use an asterisk mark to provide emphasis, such as *italics* or **bold**.

Create lists with a dash:

- Item 1
- Item 2
- Item 3

```
Use back ticks to
create a block of code
```

Embed LaTex or MathML equations,
$\frac{1}{n} \sum_{i=1}^{n} x_{i}$

Or even footnotes, citations, and a bibliography. [^1]

[^1]: Markdown is great.
~~~

Header 1

This is an R Markdown document. Markdown is a simple formatting syntax for authoring web pages.

Use an asterisk mark to provide emphasis, such as *italics* or **bold**.

Create lists with a dash:

- Item 1
- Item 2
- Item 3

```
Use back ticks to
create a block of code
```

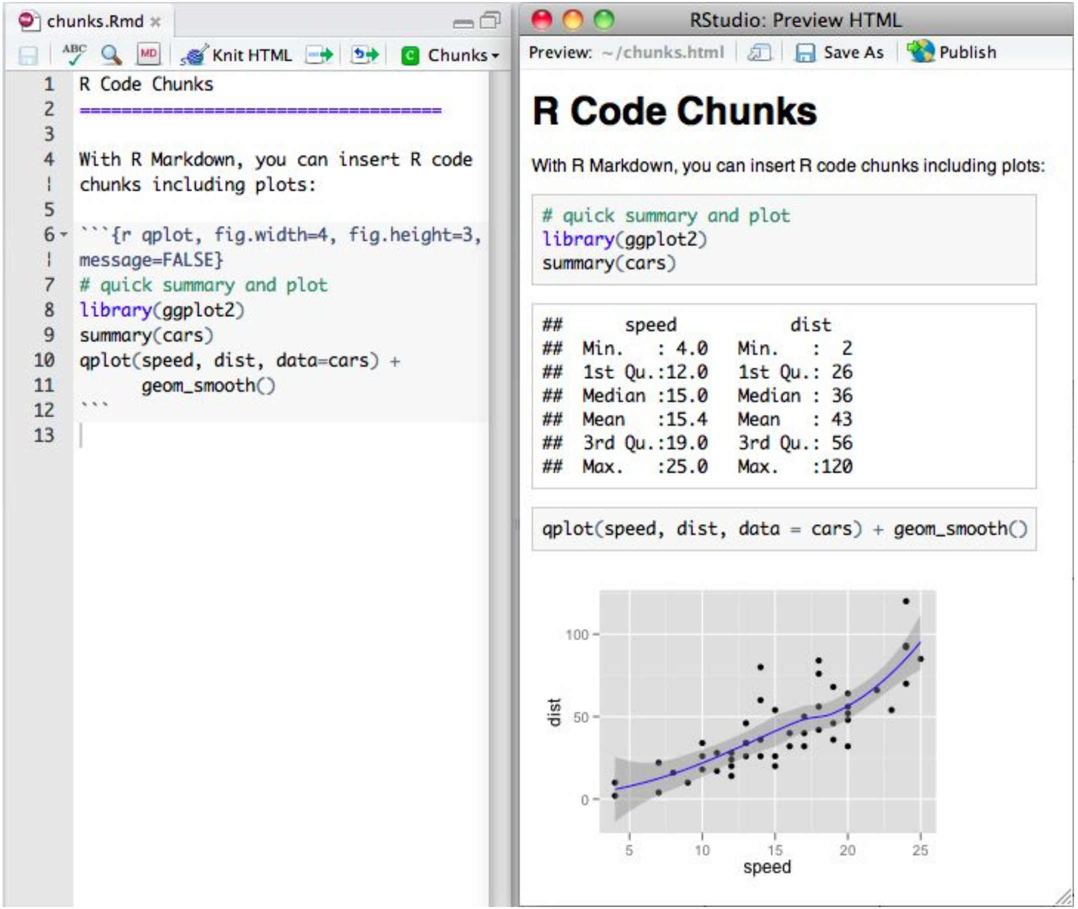
Embed LaTex or MathML equations, $\frac{1}{n} \sum_{i=1}^{n} x_{i}$

Or even footnotes, citations, and a bibliography. [1]

---

1. Markdown is great.↩

2. Documenting and reporting || Embedding code in markdown



~~~
R Code Chunks
=================================

With R Markdown, you can insert R code
chunks including plots:

```{r qplot, fig.width=4, fig.height=3,
message=FALSE}
# quick summary and plot
library(ggplot2)
summary(cars)
qplot(speed, dist, data=cars) +
      geom_smooth()
```
~~~

R Code Chunks

With R Markdown, you can insert R code chunks including plots:

```
# quick summary and plot
library(ggplot2)
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2
##  1st Qu.:12.0   1st Qu.: 26
##  Median :15.0   Median : 36
##  Mean   :15.4   Mean   : 43
##  3rd Qu.:19.0   3rd Qu.: 56
##  Max.   :25.0   Max.   :120
```

```
qplot(speed, dist, data = cars) + geom_smooth()
```

https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf

Extra:

You can call Bash (Unix Shell) programs by calling `system`:

```
# In R
system('ls')

# In python
import os
os.system('ls')
```

3. Data integrity || Keeping data & code safe

Data integrity refers to **the reliability and trustworthiness of data throughout its lifecycle**. It can describe the state of your data or even your code → **ALWAYS KEEP YOUR RAW DATA SAFE**

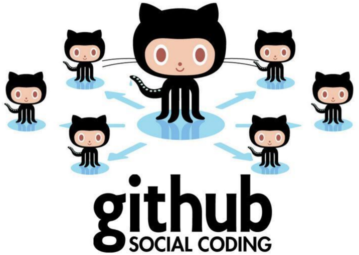
Digital Object Identifiers (DOI) are useful to keep track of data versions (and publications, code, etc…)

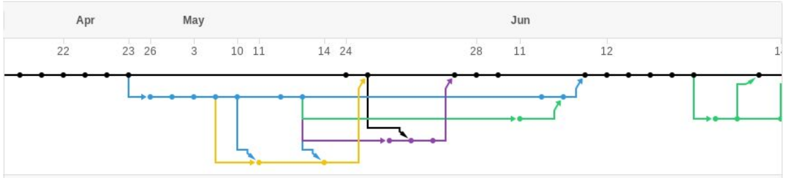
- Zenodo can generate a DOI associated to your dataset, processed data, programs…

zenodo

Version control allows to keep track of changes in your code by storing it together the modifications one or more users perform on a file (even if they happen at the same time) → **REPOSITORIES**

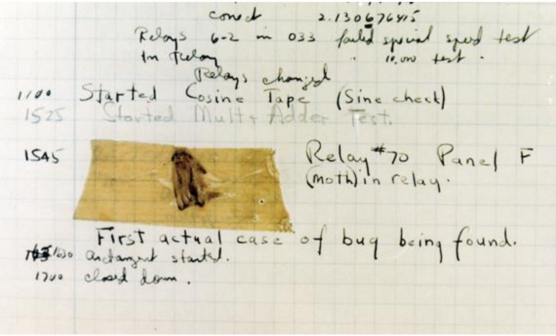
- Github is the most widely used platform. It allows
  - **branching** → to define new functions for your code without breaking the main program/analysis
  - **forking** → copying repositories made by other people
  - **wiki** → markdown edition to document your code
  - **issues** → direct communication with the developers of a tools to ask for new functionalities, report bugs…





https://docs.github.com/en/get-started

4. Bugs || Types of bugs to take into account when coding

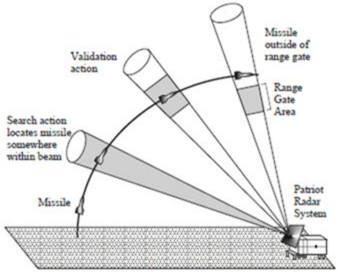


correct 2.130676415
Relays 6-2 in 033 failed special speed test
in relay . 11.000 test .
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 Antangent started.
1700 closed down.

- Grace Murray Hopper, Sept. 9th, 1947 →**"First actual case of bug being found"** in *the Harvard Mark II* computer.

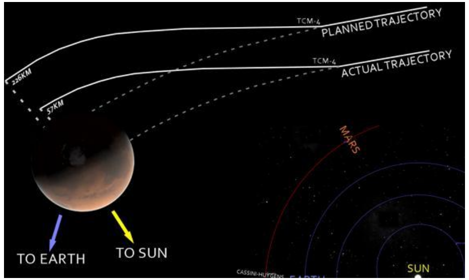
Computer **crashed**

- Now:
  - **Syntax errors**: cannot start
  - **Runtime error**: cannot finish
  - **Semantic errors**: the program runs and produces an output… BUT IT IS NOT THE EXPECTED.



Dhahran Patriot Missile assumption, 1991: minimal round → ⅓ of delay / 100h

Death of **28 soldiers** from the U.S. Army



Mars Climate Orbiter incompatibility, 1998:

- NASA : metric system
- External Software : US customary units

Collision in Mars for **$327.6M**

Y2K bug (lack of foresight), 2000: "19" before year variables → waste of memory

Incalculable, not only money

5. Exercise || Planning the analysis procedure

GOAL define the number of functions, required steps or algorithm, and expected output for the following program specifications

1. Given a fasta file with nucleotide sequences, calculate the length and GC%
2. Given a table relating genome entries with their taxonomy, identify the ten most frequent OTUs
3. Given a list of genomes and a table of annotations, explore how the GC% compares between coding and non-coding regions
4. Given a metagenomic sample from a healthy donor and a patient, compare them at the taxonomic level

551-1119-00L Microbial Community Genomics

Lecture:
Best practices for data/project/software management

Samuel Miravet-Verde 15-Nov-23