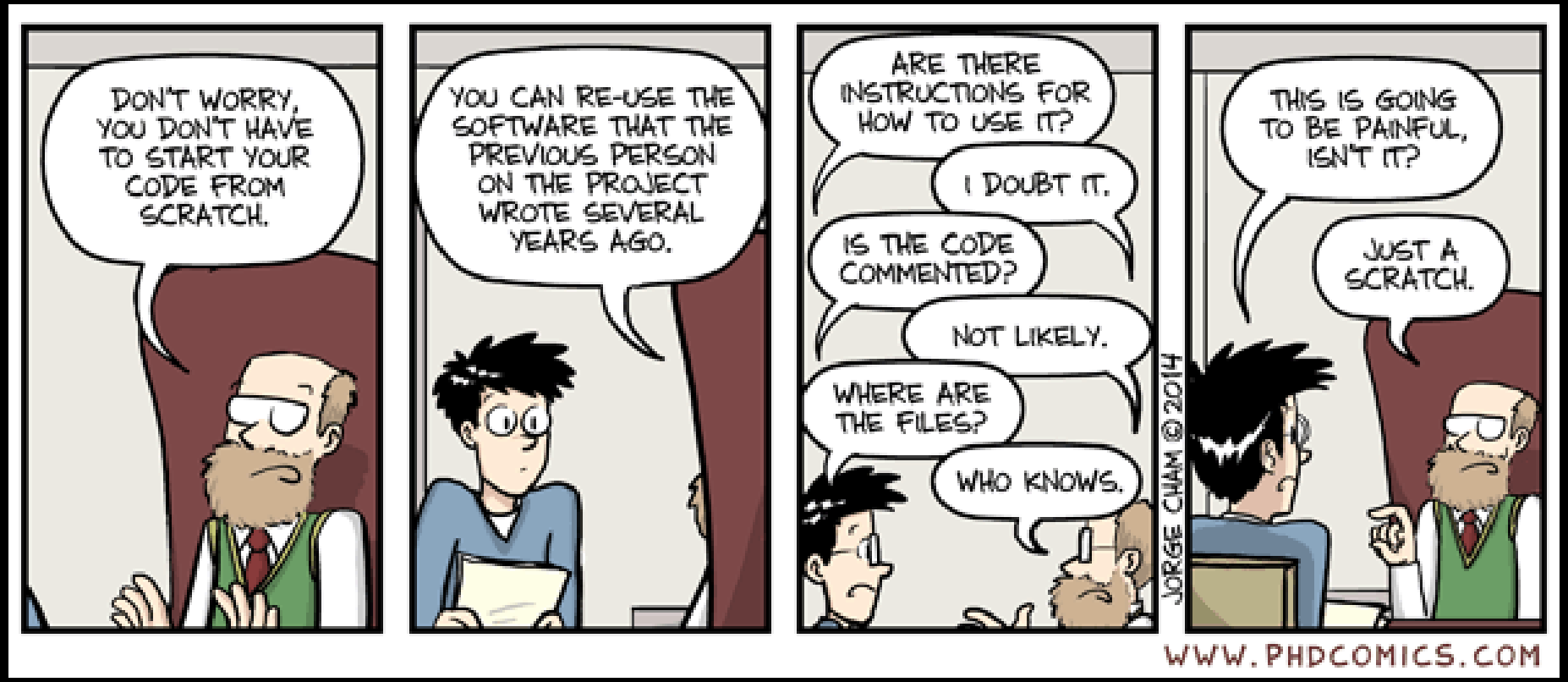
How to structure a Bioinformatics project

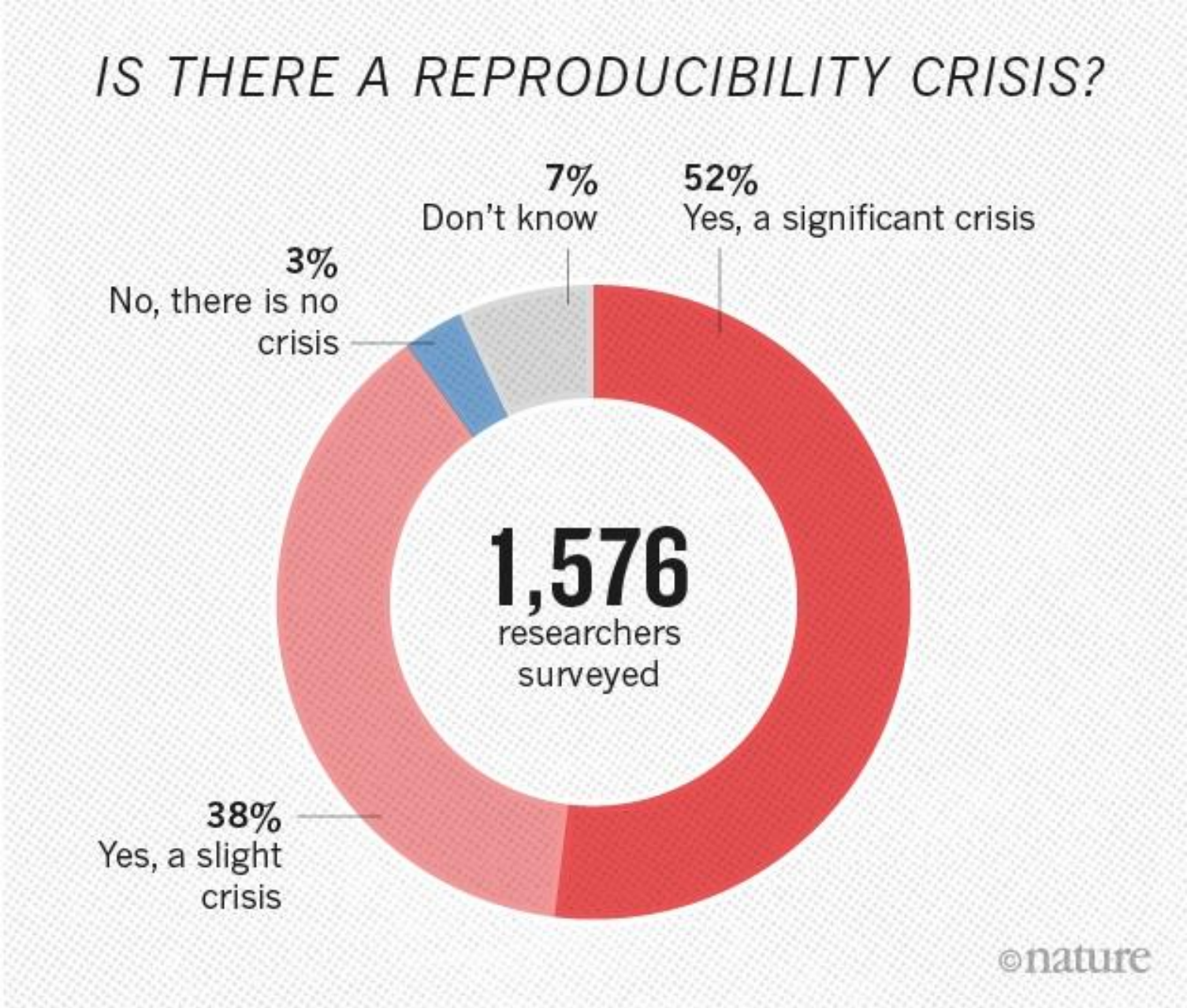
Jorge Eduardo Amaya Romero

https://github.com/jorgeamaya

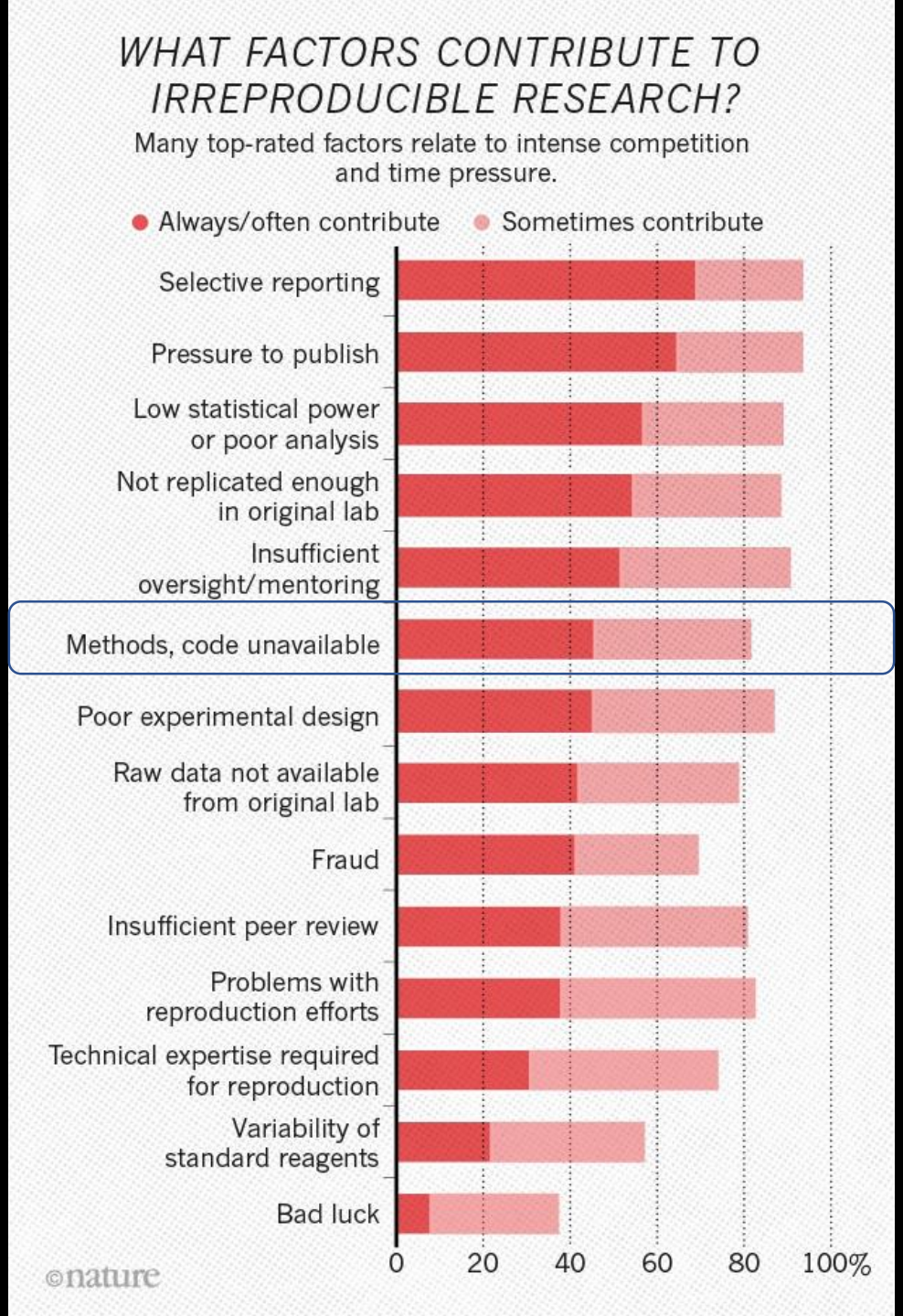


Content

- Why all this is important.
- How to structure your project.
- How to write scripts that are easy to understand.
- How to write a good README with Markdown.
- How to use dependencies in Slurm.
- How to get started with GitHub.



Nature (533, 2016) p452 -454


WHAT FACTORS CONTRIBUTE TO IRREPRODUCIBLE RESEARCH?
Many top-rated factors relate to intense competition and time pressure.
Always/often contribute
Sometimes contribute
Selective reporting
Pressure to publish
Low statistical power or poor analysis
Not replicated enough in original lab
Insufficient oversight/mentoring
Methods, code unavailable
Poor experimental design
Raw data not available from original lab
Fraud
Insufficient peer review
Problems with reproduction efforts
Technical expertise required for reproduction
Variability of standard reagents
Bad luck
0
20
40
60
80
100%
©nature

Availability of computer code

Authors must make available upon request, to editors and reviewers, any previously unreported custom computer code used to generate results that are reported in the paper and central to its main claims. Any practical issues preventing code sharing will be evaluated by the editors who reserve the right to decline the paper if important code is unavailable.

Upon publication, Nature Research Journals consider it best practice to release custom computer code in a way that allows readers to repeat the published results.

For all studies using custom code that is deemed central to the conclusions, a statement must be included in the Methods section, under the heading "Code availability", indicating whether and how the code can be accessed, including any restrictions to access.

Further reading

Top of page

http://www.nature.com/authors/policies/availability.html#code

Why is it important to produce neat code?

- Don’t waste your collaborators’ time.
- **Don’t waste your own time.**
- Don’t reinvent the wheel.
- Science moves forward through collaboration.
- Most top journals have code and data accessibility policies in place.

Evaluating the Use of ABBA–BABA Statistics to Locate Introgressed Loci

Simon H. Martin,*[1] John W. Davey,[1] and Chris D. Jiggins[1]
[1]Department of Zoology, University of Cambridge, Cambridge, United Kingdom
***Corresponding author:** E-mail: shm45@cam.ac.uk.
Associate editor: Doris Bachtrog

Abstract

Several methods have been proposed to test for introgression across genomes. One method tests for a genome-wide excess of shared derived alleles between taxa using Patterson's *D* statistic, but does not establish which loci show such an excess or whether the excess is due to introgression or ancestral population structure. Several recent studies have extended the use of *D* by applying the statistic to small genomic regions, rather than genome-wide. Here, we use simulations and whole-genome data from *Heliconius* butterflies to investigate the behavior of *D* in small genomic regions. We find that *D* is unreliable in this situation as it gives inflated values when effective population size is low, causing *D* outliers to cluster in genomic regions of reduced diversity. As an alternative, we propose a related statistic $\hat{f}_d$, a modified version of a statistic originally developed to estimate the genome-wide fraction of admixture. $\hat{f}_d$ is not subject to the same biases as *D*, and is better at identifying introgressed loci. Finally, we show that both *D* and $\hat{f}_d$ outliers tend to cluster in regions of low absolute divergence (d_{XY}), which can confound a recently proposed test for differentiating introgression from shared ancestral variation at individual loci.

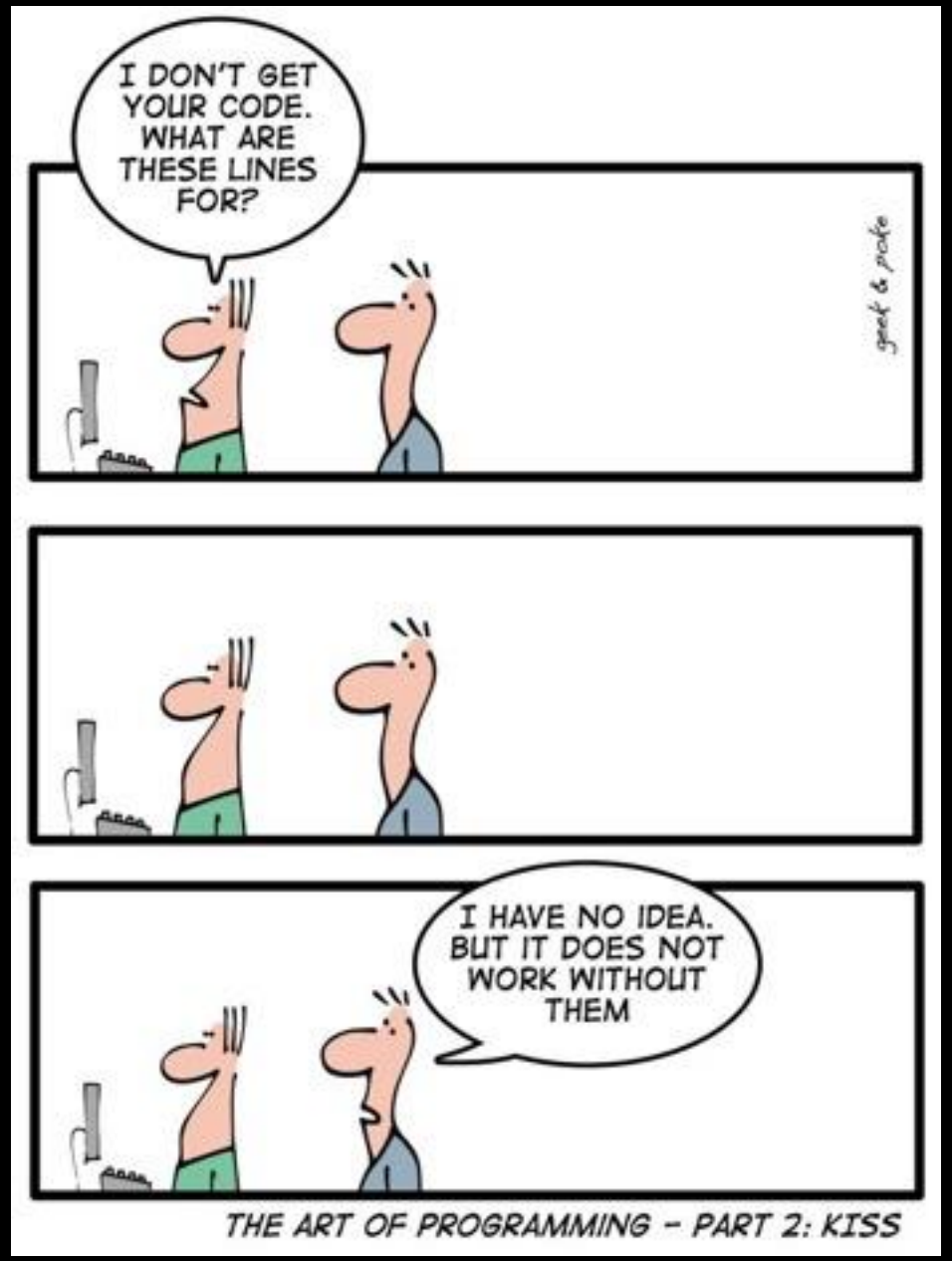
Key words: **ABBA–BABA, gene flow, introgression, population structure, *Heliconius*, simulation.**

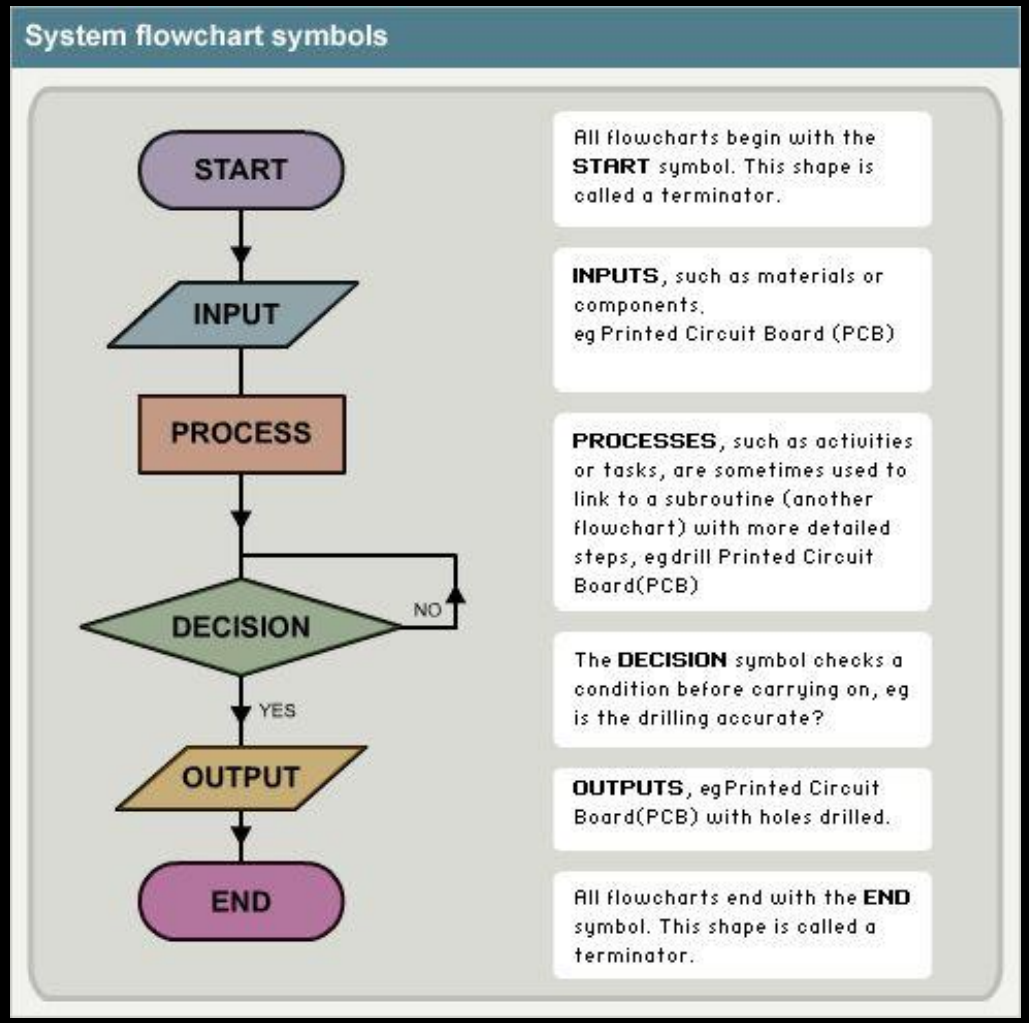
https://github.com/johnomics/Martin_Davey_Jiggins_evaluating_introgression_statistics

http://dx.doi.org/10.5061/dryad.j1rm6

Plan and stay organized

- **Don't code on the fly:** Sit at the computer ONLY when you have a clear plan in mind.
- Take notes on paper (flowchart, pseudocode, **a plain list**).
- Capitalize directories. Scripts and files go in lower case.
- Make a README file per directory.



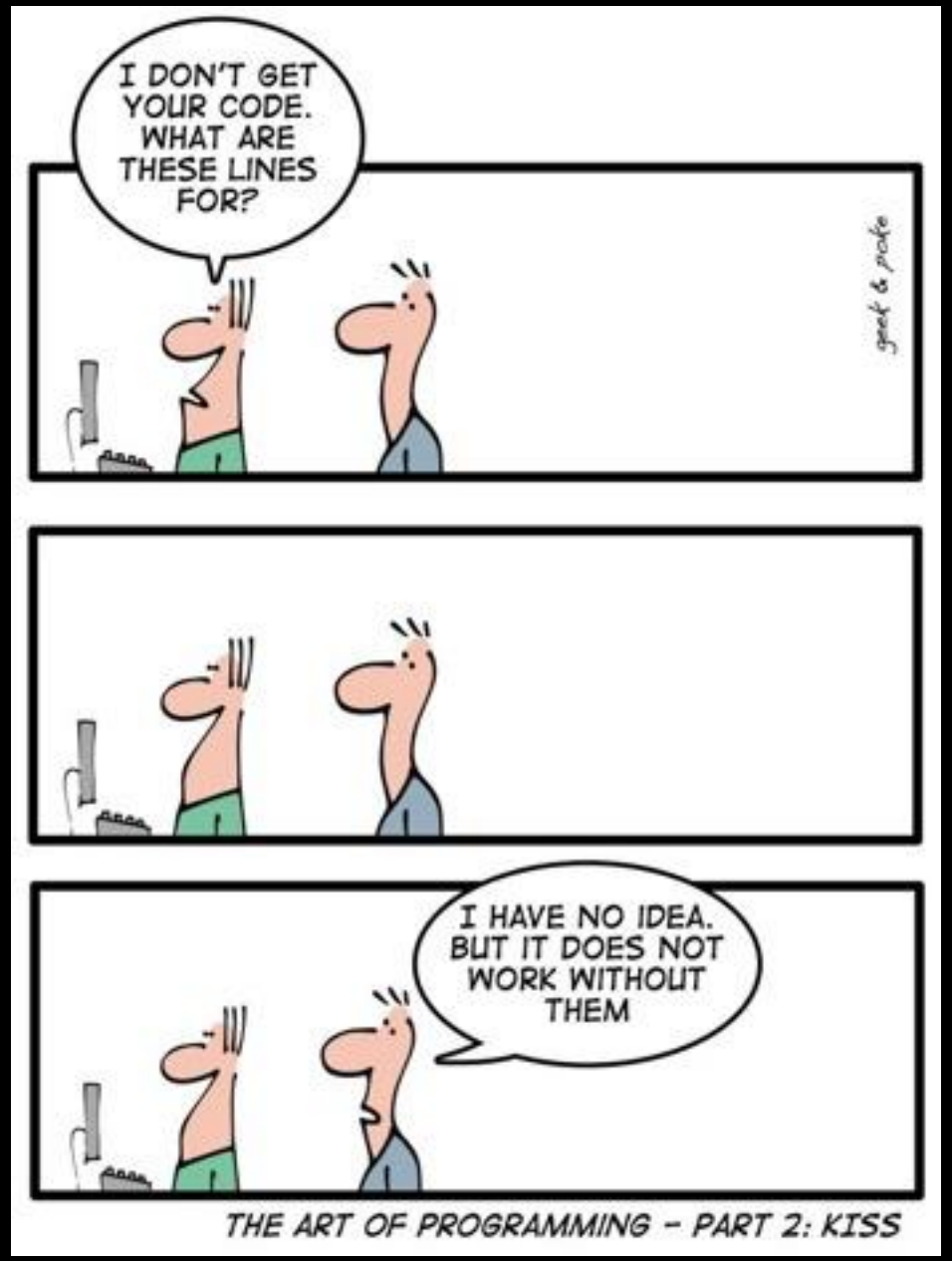


Pseudocode to Calculate the Sum & Average fo 10 Numbers

```
begin
        initialize counter to 0
        initialize accumulator to 0
        loop
                read input from keyboard
                accumulate input
                increment counter
        while counter < 10
        calculate average
        print sum
        print average
end
```

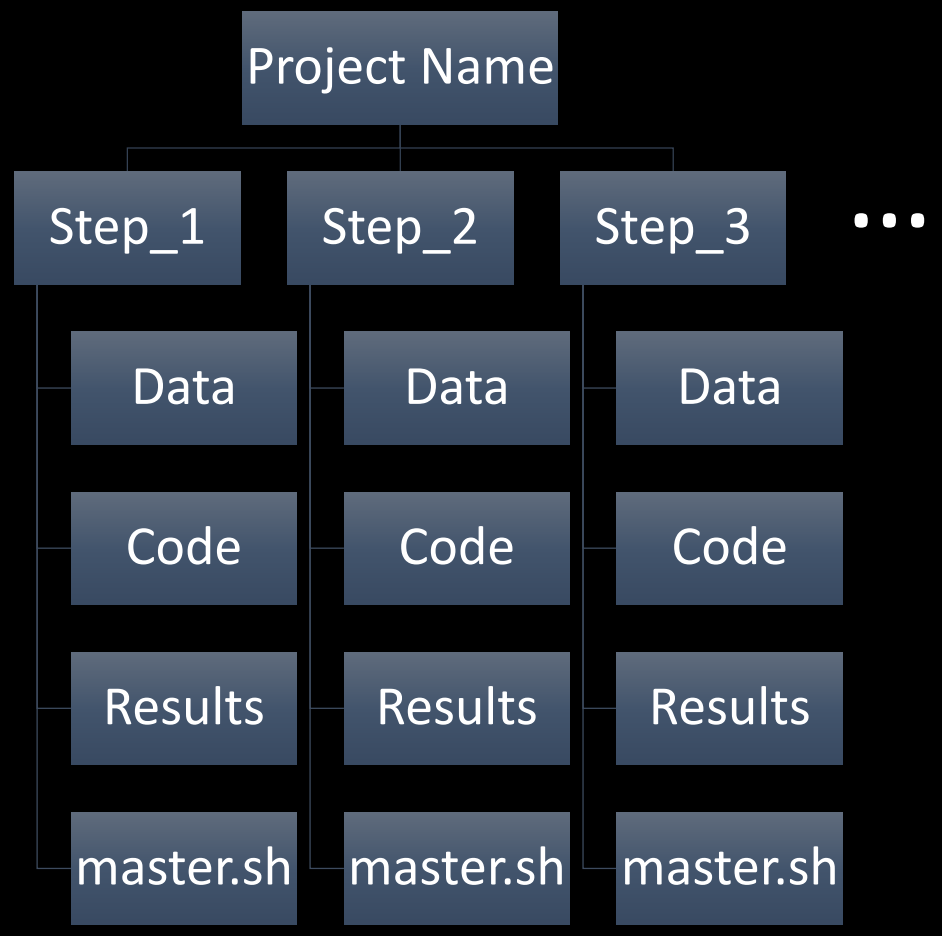
Plan and stay organized

- **Don’t code on the fly:** Sit at the computer ONLY when you have a clear plan in mind.
- Take notes on paper (flowchart, pseudocode, **a plain list**).
- Capitalize directories. Scripts and files in lower case.
- Make a README file per directory.



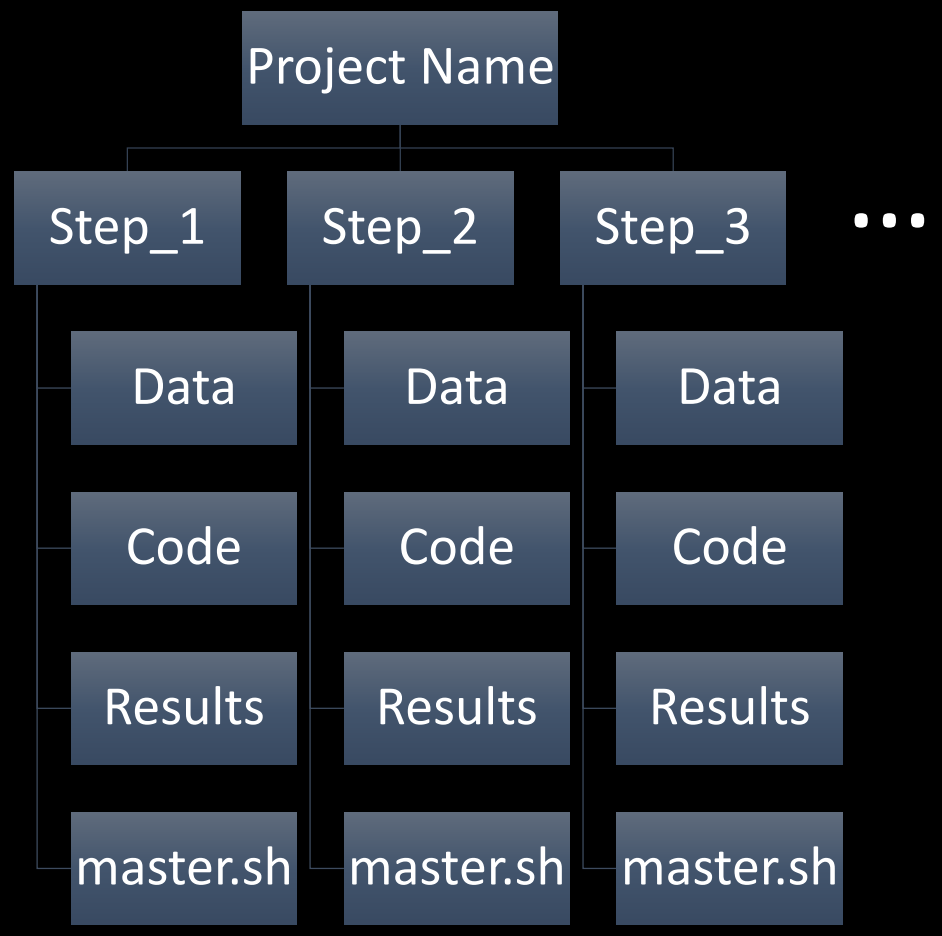
Use a consistent structure for your project

- A project directory that will include all the steps of your project (and a README file).
- Create subdirectories for each step of the project.
- Each subdirectory must include three subsubdirectories: Code, Data, Results, a master.sh script, and a README file.
- A TMP directory is optional but must be gone by the end.



Use a consistent structure for your project

- All processes in a step must be executed from master.sh by calling scripts in Code.
- **NEVER** write to Data or Code (you might corrupt your Data or your scripts).
- Emulate in Results the order of subdirectories in Data (to the extent possible).
- (Data and Results can also be created from master.sh. That way, if the code produces undesirable results, you can quickly delete these two directories, correct your code and try again.)



master.sh

```
#!/bin/bash
#Step_1
#Purpose: Add a second line to example_data.txt file
#Written by: Jorge Eduardo Amaya Romero
#Date: May 18 2017

ln -s /n/mallet_lab/jorge/My_Data/* Data/. #Link your precious data from an external repository
Step2 use results from Step1
ln -s ../Step1/Results/* Data/.

cat Data/example_data.txt > Results/example_data.txt
echo "The second line of an example file" >> Results/example_data.txt
```

- Linking ensures that:
  - You won’t delete your data.
  - If the paths change (for instance, you move to a different lab or you start a new project), you only need to change one line of code. The rest of the pipeline will remain functional.

Make your code easy to understand

- Include headers in every script.
  - Title of the script
  - Description of what the script does.
  - Last date the script was modified.
  - Authors.
- Make plenty of comments (explain the **WHY or Purpose**, not how.)
- Avoid lines that are too long to fit in the width of your screen.
- In languages like Python or R, avoid excessive nesting

```
#!/bin/python

line ='1,2,3,4,5,6,6.5’
printfloat(int(float(str(sum([float(line.split(',')[i])foriinrange(0,len(line.split(',')))]))))
```

Make your code easy to understand

- Use indentation even in programs in which it isn’t obligatory.
- Be consistent with your indentation patters (in style and means – don’t mix tabs with spaces.)

```
function foo
disp('classic smart indenting');

if youCanFollowThisCode
    myNestedFunction;
end

    function myNestedFunction
    if weShouldDoSomething
        disp('we did something');
    end
    end

end
```

Make your code easy to understand

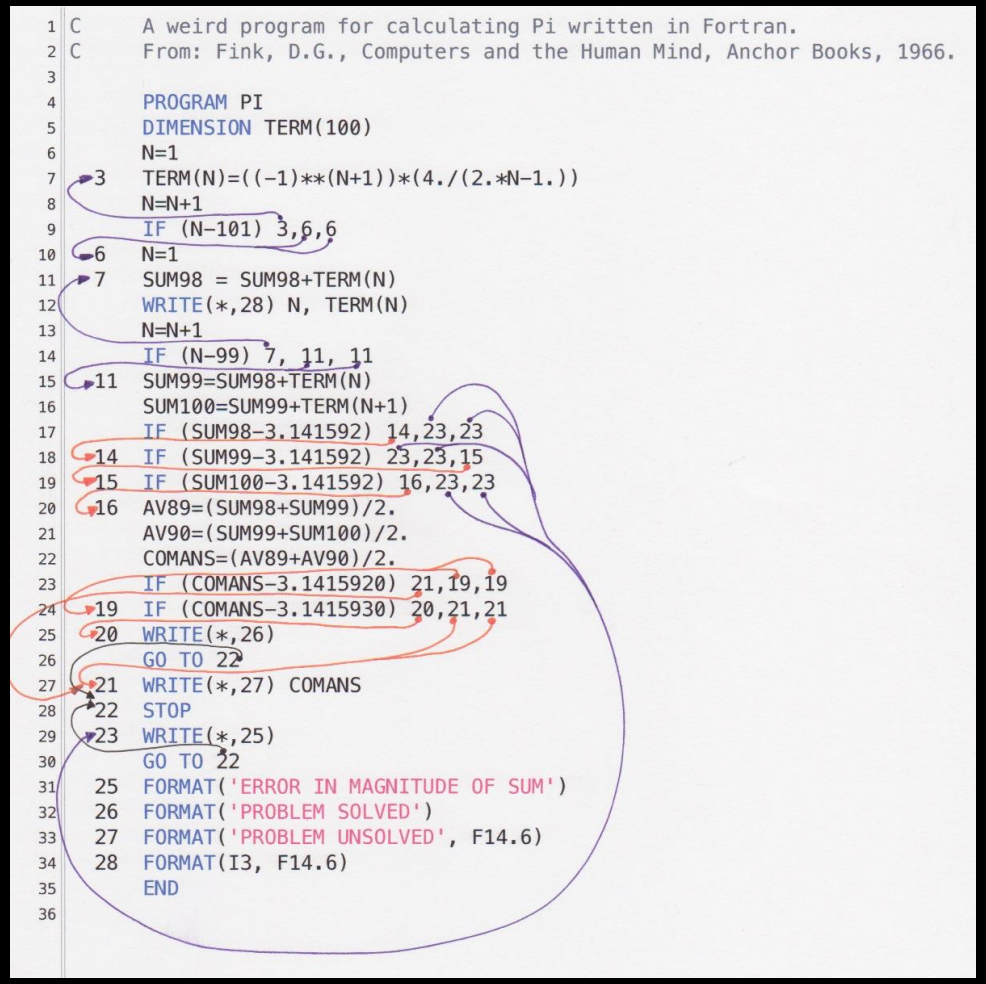
- Use indentation even in programs in which it isn’t obligatory.
- Be consistent with your indentation patters (in style and means – don’t mix tabs with spaces.)

```
function foo
disp('classic smart indenting');

if youCanFollowThisCode
    myNestedFunction;
end

    function myNestedFunction
    if weShouldDoSomething
        disp('we did something');
    end
    end

end
```

```
function foo
    disp('classic smart indenting');

    if youCanFollowThisCode
        myNestedFunction;
    end

    function myNestedFunction
        if weShouldDoSomething
            disp('we did something');
        end
    end

end
```

Create scripts based on functions

- To avoid “Spaghetti code”.
- To avoid repetitive code.
- To eases error correction.
- Rules to make a good function:
  - A function should perform ONLY one task.
  - A function should fit on the screen.
  - (If your function is getting long, consider splitting it in smaller functions.)

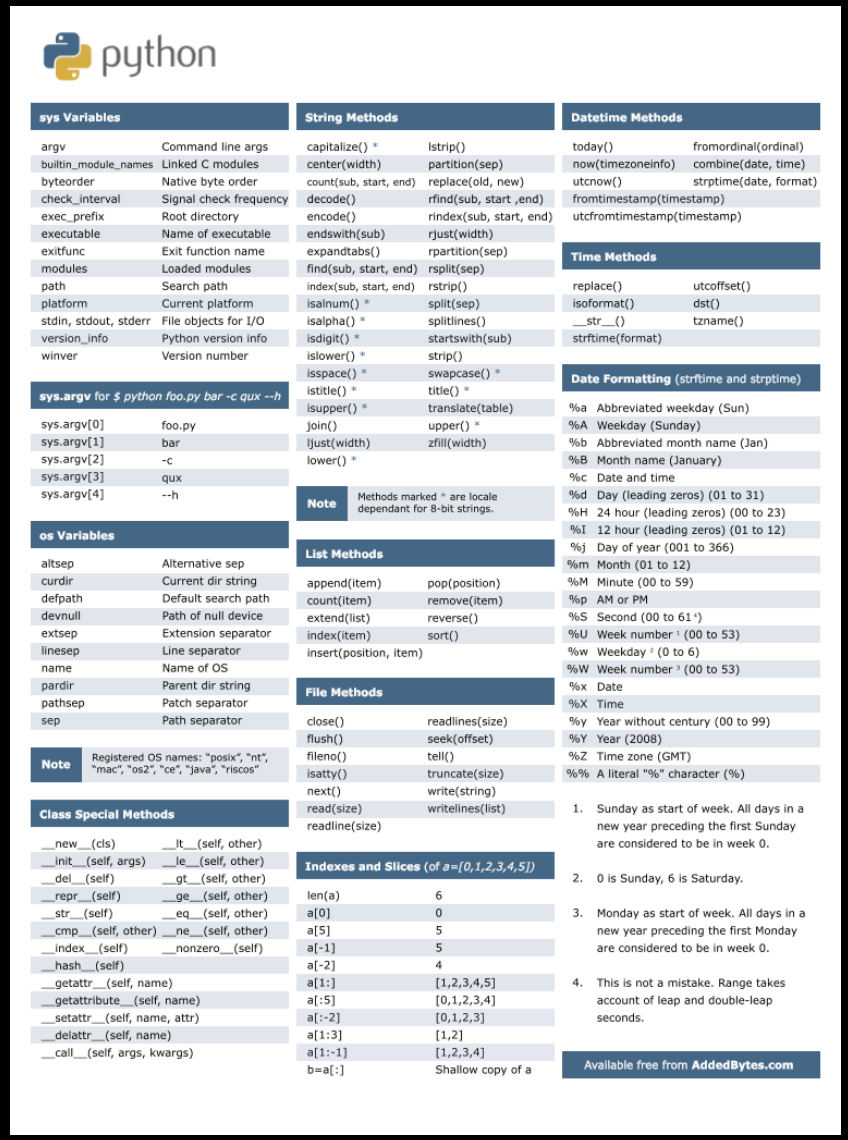


```
C     A weird program for calculating Pi written in Fortran.
C     From: Fink, D.G., Computers and the Human Mind, Anchor Books, 1966.

      PROGRAM PI
      DIMENSION TERM(100)
      N=1
    3 TERM(N)=((-1)**(N+1))*(4./(2.*N-1.))
      N=N+1
      IF (N-101) 3,6,6
    6 N=1
    7 SUM98 = SUM98+TERM(N)
      WRITE(*,28) N, TERM(N)
      N=N+1
      IF (N-99) 7, 11, 11
   11 SUM99=SUM98+TERM(N)
      SUM100=SUM99+TERM(N+1)
      IF (SUM98-3.141592) 14,23,23
   14 IF (SUM99-3.141592) 23,23,15
   15 IF (SUM100-3.141592) 16,23,23
   16 AV89=(SUM98+SUM99)/2.
      AV90=(SUM99+SUM100)/2.
      COMANS=(AV89+AV90)/2.
      IF (COMANS-3.1415920) 21,19,19
   19 IF (COMANS-3.1415930) 20,21,21
   20 WRITE(*,26)
      GO TO 22
   21 WRITE(*,27) COMANS
   22 STOP
   23 WRITE(*,25)
      GO TO 22
   25 FORMAT('ERROR IN MAGNITUDE OF SUM')
   26 FORMAT('PROBLEM SOLVED')
   27 FORMAT('PROBLEM UNSOLVED', F14.6)
   28 FORMAT(I3, F14.6)
      END
```

https://craftofcoding.files.wordpress.com/2013/10/lore_spaghetti.pdf

Some finals remarks

- Don’t obsess with making your code efficient (unless it’s necessary).
- If it works, it’s fine!
- Don’t make your code a black box. While you’re programming, print regularly to the screen. You can remove the prints later.
- Look for cheat sheets.



python

sys Variables

argv	Command line args
builtin_module_names	Linked C modules
byteorder	Native byte order
check_interval	Signal check frequency
exec_prefix	Root directory
executable	Name of executable
exitfunc	Exit function name
modules	Loaded modules
path	Search path
platform	Current platform
stdin, stdout, stderr	File objects for I/O
version_info	Python version info
winver	Version number

sys.argv for *$ python foo.py bar -c qux --h*

sys.argv[0]	foo.py
sys.argv[1]	bar
sys.argv[2]	-c
sys.argv[3]	qux
sys.argv[4]	--h

os Variables

altsep	Alternative sep
curdir	Current dir string
defpath	Default search path
devnull	Path of null device
extsep	Extension separator
linesep	Line separator
name	Name of OS
pardir	Parent dir string
pathsep	Patch separator
sep	Path separator

Note Registered OS names: “posix”, “nt”, “mac”, “os2”, “ce”, “java”, “riscos”

Class Special Methods

__new__(cls)	__lt__(self, other)
__init__(self, args)	__le__(self, other)
__del__(self)	__gt__(self, other)
__repr__(self)	__ge__(self, other)
__str__(self)	__eq__(self, other)
__cmp__(self, other)	__ne__(self, other)
__index__(self)	__nonzero__(self)
__hash__(self)	
__getattr__(self, name)	
__getattribute__(self, name)	
__setattr__(self, name, attr)	
__delattr__(self, name)	
__call__(self, args, kwargs)	

String Methods

capitalize() *	lstrip()
center(width)	partition(sep)
count(sub, start, end)	replace(old, new)
decode()	rfind(sub, start ,end)
encode()	rindex(sub, start, end)
endswith(sub)	rjust(width)
expandtabs()	rpartition(sep)
find(sub, start, end)	rsplit(sep)
index(sub, start, end)	rstrip()
isalnum() *	split(sep)
isalpha() *	splitlines()
isdigit() *	startswith(sub)
islower() *	strip()
isspace() *	swapcase() *
istitle() *	title() *
isupper() *	translate(table)
join()	upper() *
ljust(width)	zfill(width)
lower() *	

Note Methods marked * are locale dependant for 8-bit strings.

List Methods

append(item)	pop(position)
count(item)	remove(item)
extend(list)	reverse()
index(item)	sort()
insert(position, item)	

File Methods

close()	readlines(size)
flush()	seek(offset)
fileno()	tell()
isatty()	truncate(size)
next()	write(string)
read(size)	writelines(list)
readline(size)	

Indexes and Slices (of *a=[0,1,2,3,4,5]*)

len(a)	6
a[0]	0
a[5]	5
a[-1]	5
a[-2]	4
a[1:]	[1,2,3,4,5]
a[:5]	[0,1,2,3,4]
a[:-2]	[0,1,2,3]
a[1:3]	[1,2]
a[1:-1]	[1,2,3,4]
b=a[:]	Shallow copy of a

Datetime Methods

today()	fromordinal(ordinal)
now(timezoneinfo)	combine(date, time)
utcnow()	strptime(date, format)
fromtimestamp(timestamp)	
utcfromtimestamp(timestamp)	

Time Methods

replace()	utcoffset()
isoformat()	dst()
__str__()	tzname()
strftime(format)	

Date Formatting (strftime and strptime)

%a	Abbreviated weekday (Sun)
%A	Weekday (Sunday)
%b	Abbreviated month name (Jan)
%B	Month name (January)
%c	Date and time
%d	Day (leading zeros) (01 to 31)
%H	24 hour (leading zeros) (00 to 23)
%I	12 hour (leading zeros) (01 to 12)
%j	Day of year (001 to 366)
%m	Month (01 to 12)
%M	Minute (00 to 59)
%p	AM or PM
%S	Second (00 to 61 [4])
%U	Week number [1] (00 to 53)
%w	Weekday [2] (0 to 6)
%W	Week number [3] (00 to 53)
%x	Date
%X	Time
%y	Year without century (00 to 99)
%Y	Year (2008)
%Z	Time zone (GMT)
%%	A literal "%" character (%)

1. Sunday as start of week. All days in a new year preceding the first Sunday are considered to be in week 0.
2. 0 is Sunday, 6 is Saturday.
3. Monday as start of week. All days in a new year preceding the first Monday are considered to be in week 0.
4. This is not a mistake. Range takes account of leap and double-leap seconds.

Available free from **AddedBytes.com**

Basic Markdown Syntax

- Title
  - # Title
- Subtitle
  - ## Subtitle
- Subsubtitle
  - ### Subsubtitle
- Normal text
  - BlaBlaBla

- Code
  - ’’’
  - ./master.sh
  - ’’’
- Italics
  - *Italics*
- Bold
  - **Bold**

- Items
  - * ABC
  - * DEF
  - * GHI
- Items
  - 1. ABC
  - 2. CDE
  - 3. FGH

Basic Markdown Syntax

- Images
  - ![rug](https://www.rug.nl/_definition/shared/images/logo--en.png)
- Emojis
  - 👍 :thumbsup , ⚠️ :warning: See complete list -> https://gist.github.com/rxaviers/7360908
- Hyperlinks
  - [picard](https://github.com/broadinstitute/picard)

- This is enough to make decent README files.
- Don’t forget to add md extension to the file (README.md)
- More information on how to use markdown: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#images

Your plain and boring README file

```
# Find appropriate sampling rates for MITObim
*Written by: Jorge Eduardo Amaya Romero*

Read this whole file before using the script

##Description:
Since MITObim requires appropriate sampling rates to produce nice results, these scripts subsample, assemble, align, evaluate and
report the quality of the assemblies for 765 malaria mitochondrial genomes in an authomatic fashion. If you plan to refurbish
these script to fit your needs, run
```
grep "Refurbish" *.sh
grep "Refurbish" Code/control.py
```
in the main directory and you'll get a report of all the lines that must be edited. This scripts were desing to run on Slurm
Version 16.05

##Clone this repository
You should be able to clone this repository and start working immediately, just do:

```
git clone git://github.com/jorgeamaya/Malaria/SamplingTest.git
```

##Scripts

1. sam.sh and subordinate_sam.sh: Subsample the data sets.
2. ass.sh and subordinate_ass.sh: Perform the assemblies.
3. ali.sh and subordinate_ali.sh: Aligns the sequences.
4. remove.sh: Reports wich sequences produces assemblies of good quality and which sequences must be realigned at a higher rate.
5. clean.sh: Cleans the Results directory.
6. quality.sh: Check if there are assemblies with more than an arbitrary threshold of ambiguities.
```

- …will look like this in a web-browser…

Find appropriate sampling rates for MITObim

Written by: Jorge Eduardo Amaya Romero

Read this whole file before using the script

##Description: Since MITObim requires appropriate sampling rates to produce nice results, these scripts subsample, assemble, align, evaluate and report the quality of the assemblies for 765 malaria mitochondrial genomes in an authomatic fashion. If you plan to refurbish these script to fit your needs, run

```
grep "Refurbish" \*.sh
grep "Refurbish" Code/control.py
```

in the main directory and you'll get a report of all the lines that must be edited. This scripts were desing to run on Slurm Version 16.05

##Clone this repository You should be able to clone this repository and start working immediately, just do:

```
git clone git://github.com/jorgeamaya/Malaria/SamplingTest.git
```

##Scripts

1. sam.sh and subordinate_sam.sh: Subsample the data sets.
2. ass.sh and subordinate_ass.sh: Perform the assemblies.
3. ali.sh and subordinate_ali.sh: Aligns the sequences.
4. remove.sh: Reports wich sequences produces assemblies of good quality and which sequences must be realigned at a higher rate.
5. clean.sh: Cleans the Results directory.
6. quality.sh: Check if there are assemblies with more than an arbitrary threshold of ambiguities.

Extra: How to create an alias in Linux

- Purpose: Save time by not typing frequently used commands.

```
echo "alias q='squeue -u $USER'" >> ~/.bashrc
```

- Log out and reconnect or run source.
- Now just type q.
- Other useful aliases

```
alias s='srun -p interact --pty --mem 40000 -t 0-04:00 /bin/bash'
alias u='git push -u origin master'
```

Version control (with GitHub)

- What is version control?
  - To do version control is to record changes to files or set of files, which is done with a Version control system (VSC).
- Why is it important?
  - Because it allows you to keep track of the changes you've done to your program.

Version control (with GitHub)

- What is version control?
  - To do version control is to record changes to files or set of files, which is done with a Version control system (VSC).
- Why is it important?
  - Because it allows you to keep track of the changes you've done to your program.

Start program

Version control (with GitHub)

- What is version control?
  - To do version control is to record changes to files or set of files, which is done with a Version control system (VSC).
- Why is it important?
  - Because it allows you to keep track of the changes you've done to your program.

Start program → Add feature 1

Version control (with GitHub)

- What is version control?
  - To do version control is to record changes to files or set of files, which is done with a Version control system (VSC).
- Why is it important?
  - Because it allows you to keep track of the changes you've done to your program.

Version control (with GitHub)

- What is version control?
  - To do version control is to record changes to files or set of files, which is done with a Version control system (VSC).
- Why is it important?
  - Because it allows you to keep track of the changes you've done to your program.

Version control (with GitHub)

- What is version control?
  - To do version control is to record changes to files or set of files, which is done with a Version control system (VSC).
- Why is it important?
  - Because it allows you to keep track of the changes you've done to your program.

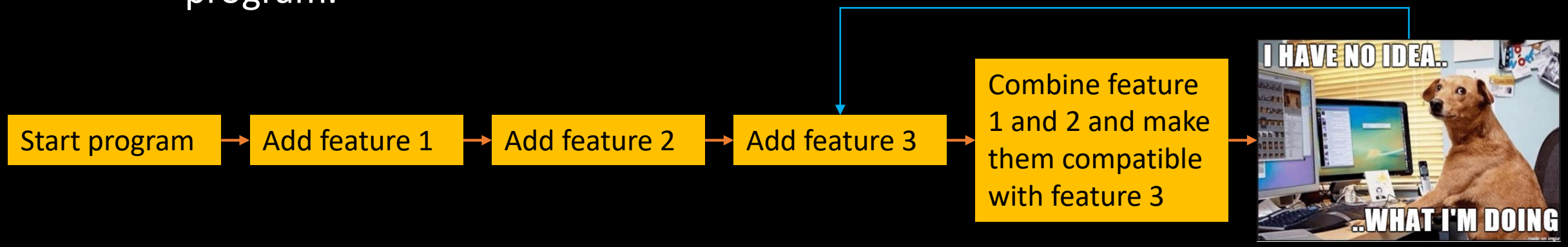
Version control (with GitHub)

- What is version control?
  - To do version control is to record changes to files or set of files, which is done with a Version control system (VSC).
- Why is it important?
  - Because it allows you to keep track of the changes you've done to your program.

Start program → Add feature 1 → Add feature 2 → Add feature 3 → Combine feature 1 and 2 and make them compatible with feature 3 →

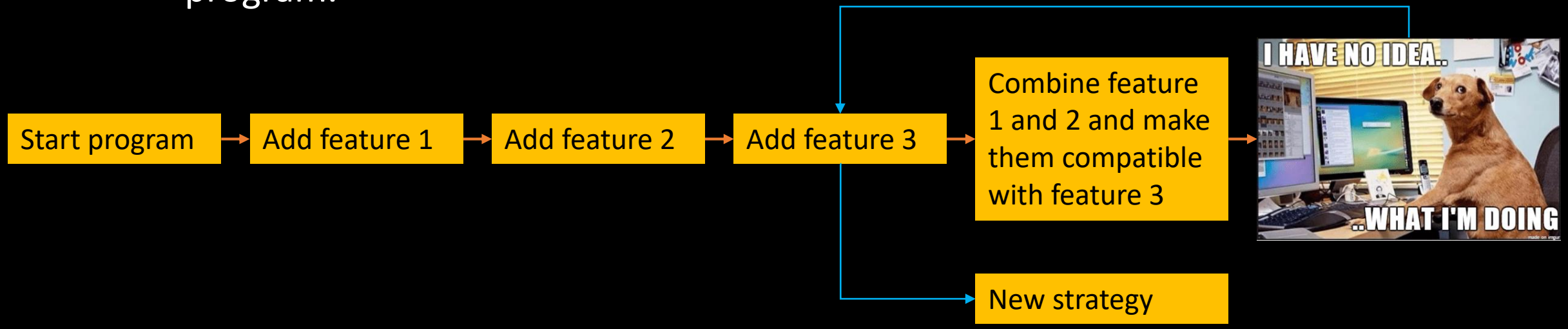
Version control (with GitHub)

- What is version control?
  - To do version control is to record changes to files or set of files, which is done with a Version control system (VSC).
- Why is it important?
  - Because it allows you to keep track of the changes you've done to your program.



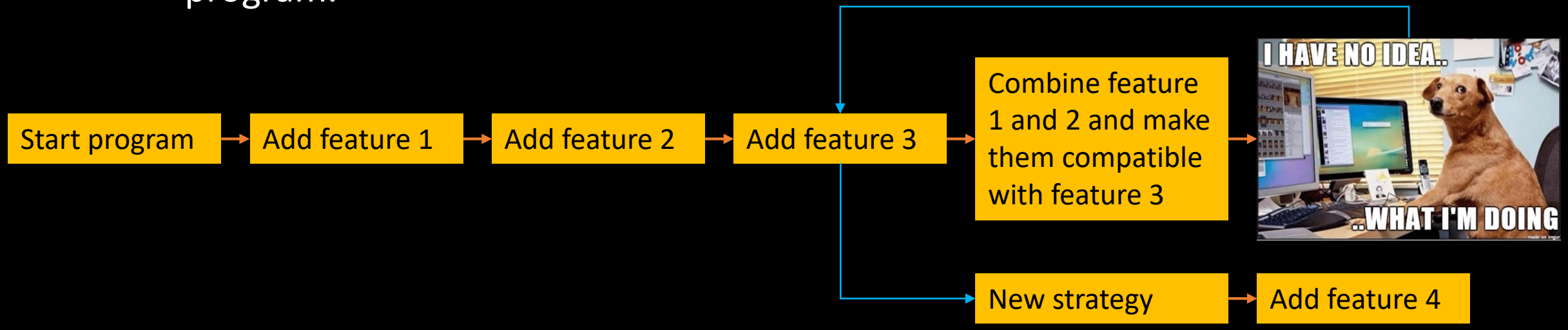
Version control (with GitHub)

- What is version control?
  - To do version control is to record changes to files or set of files, which is done with a Version control system (VSC).
- Why is it important?
  - Because it allows you to keep track of the changes you've done to your program.



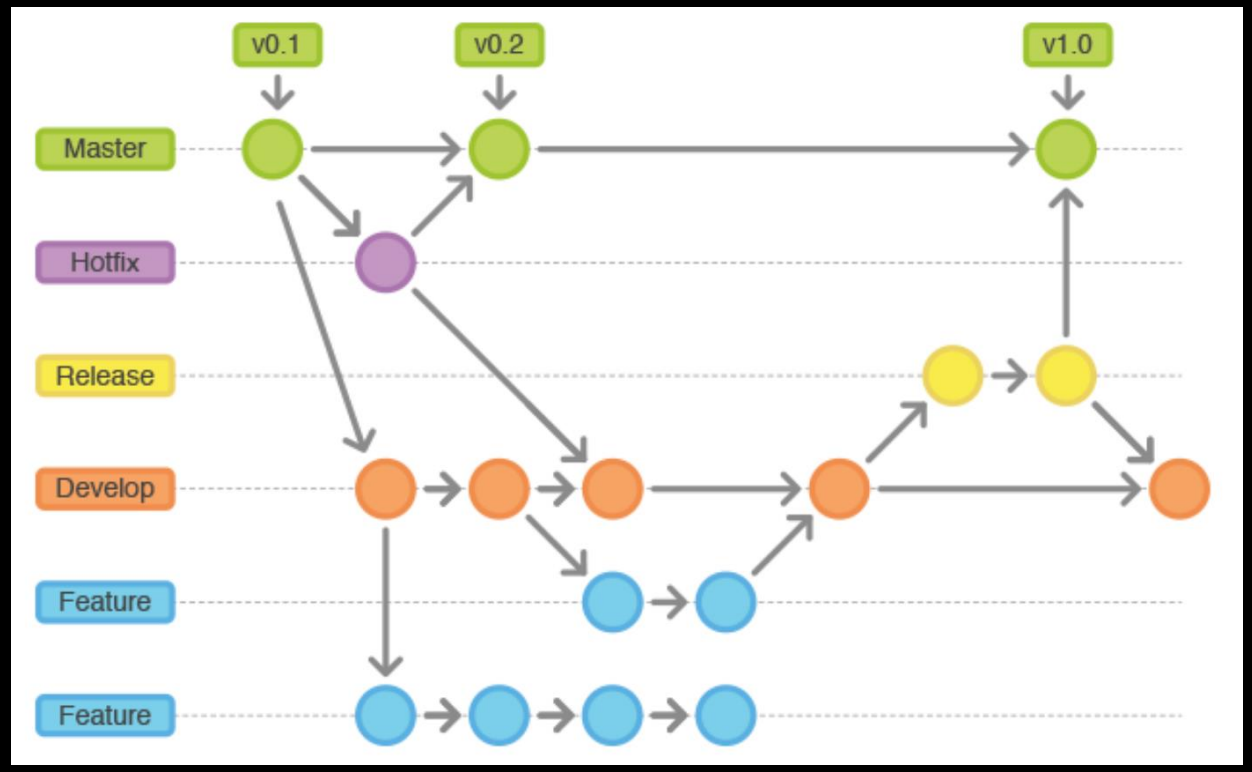
Version control (with GitHub)

- What is version control?
  - To do version control is to record changes to files or set of files, which is done with a Version control system (VSC).
- Why is it important?
  - Because it allows you to keep track of the changes you've done to your program.

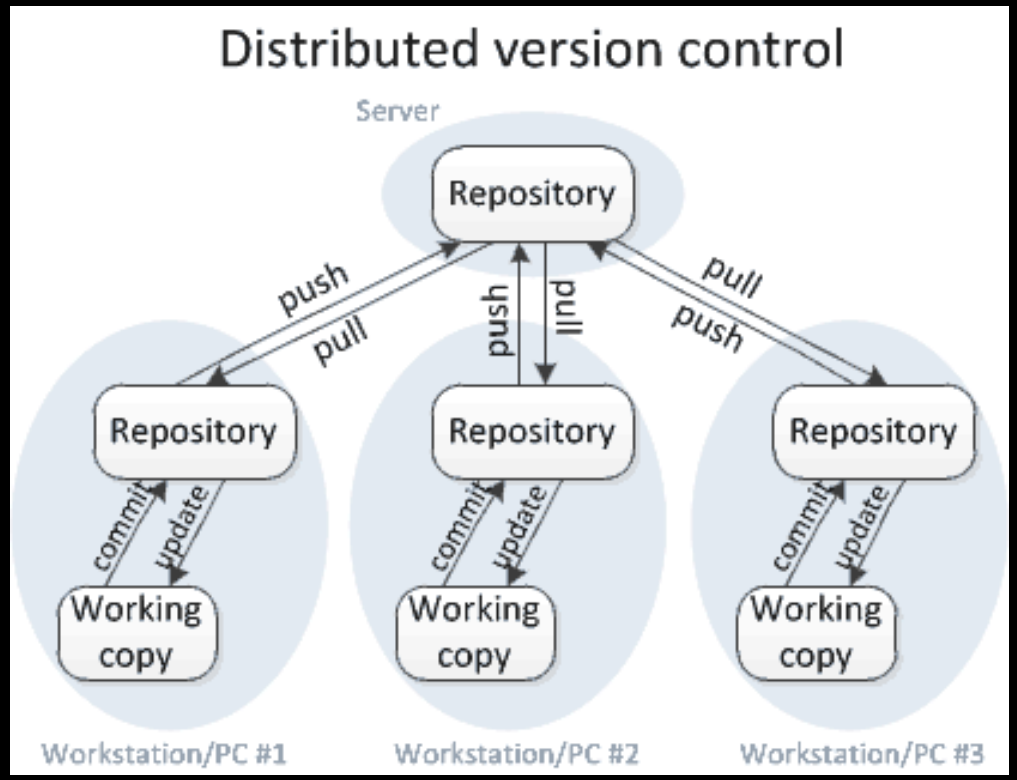


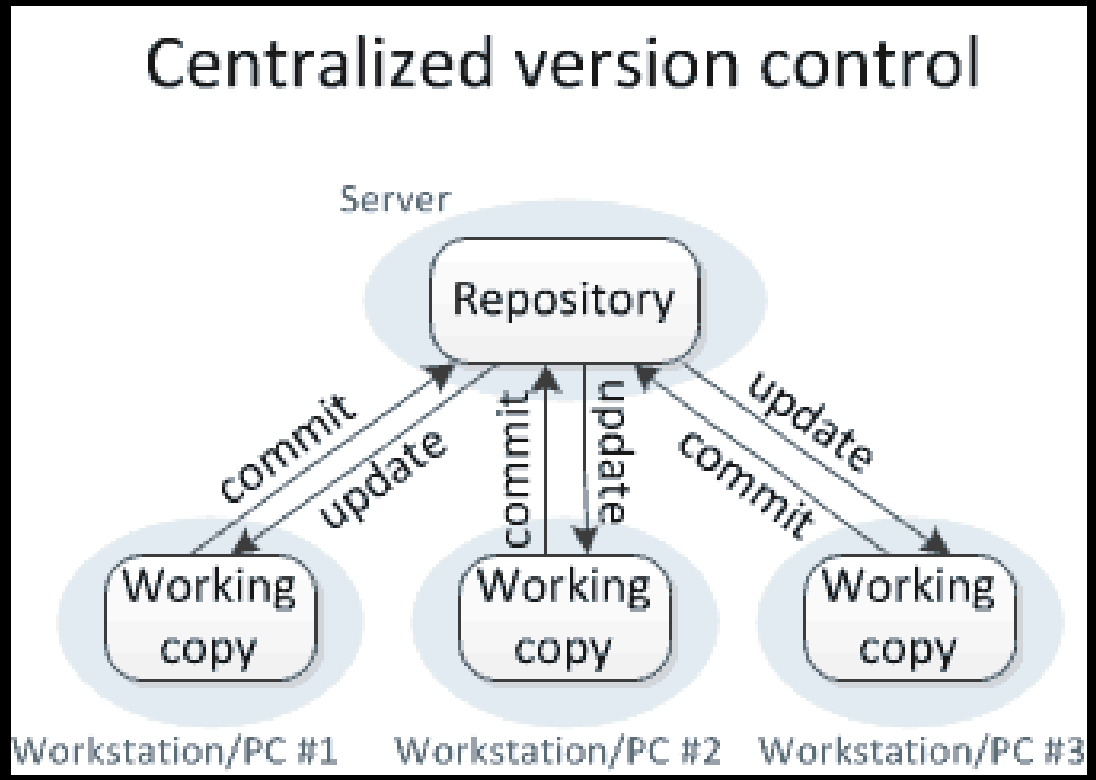
Version control (with GitHub)

- The VSC is a “database” of all the changes you made to your files.
- Revert individual files to a previous state.
- Revert the whole project to a previous state.
- Review your productivity.
- Recover files if they are lost.
- It’s like the undo – redo buttons in Office.



Version control (with GitHub)





How to create a GitHub

- Create an account in GitHub (obviously).
- Create a new repository (don't close your browser: you'll new the URL) https://help.github.com/articles/creating-a-new-repository/.
- In Bash, move to the directory that contains your project.

```
module load git/2.1.0-fasrc01
#Initialize the directory
git init #Initialize the directory.
git remote add origin "remote repository URL" #Link your virtual repository
git remote #Am I connected?
#Every time you want to upload…
git add . #Stage files to be uploaded
git status #Check if you are staging the correct files
git commit –m "Initial commit" #Commit the files with a message
git push -u origin master #Push to your virtual repository
```

- "remote repository URL" must be changed to your repository URL.

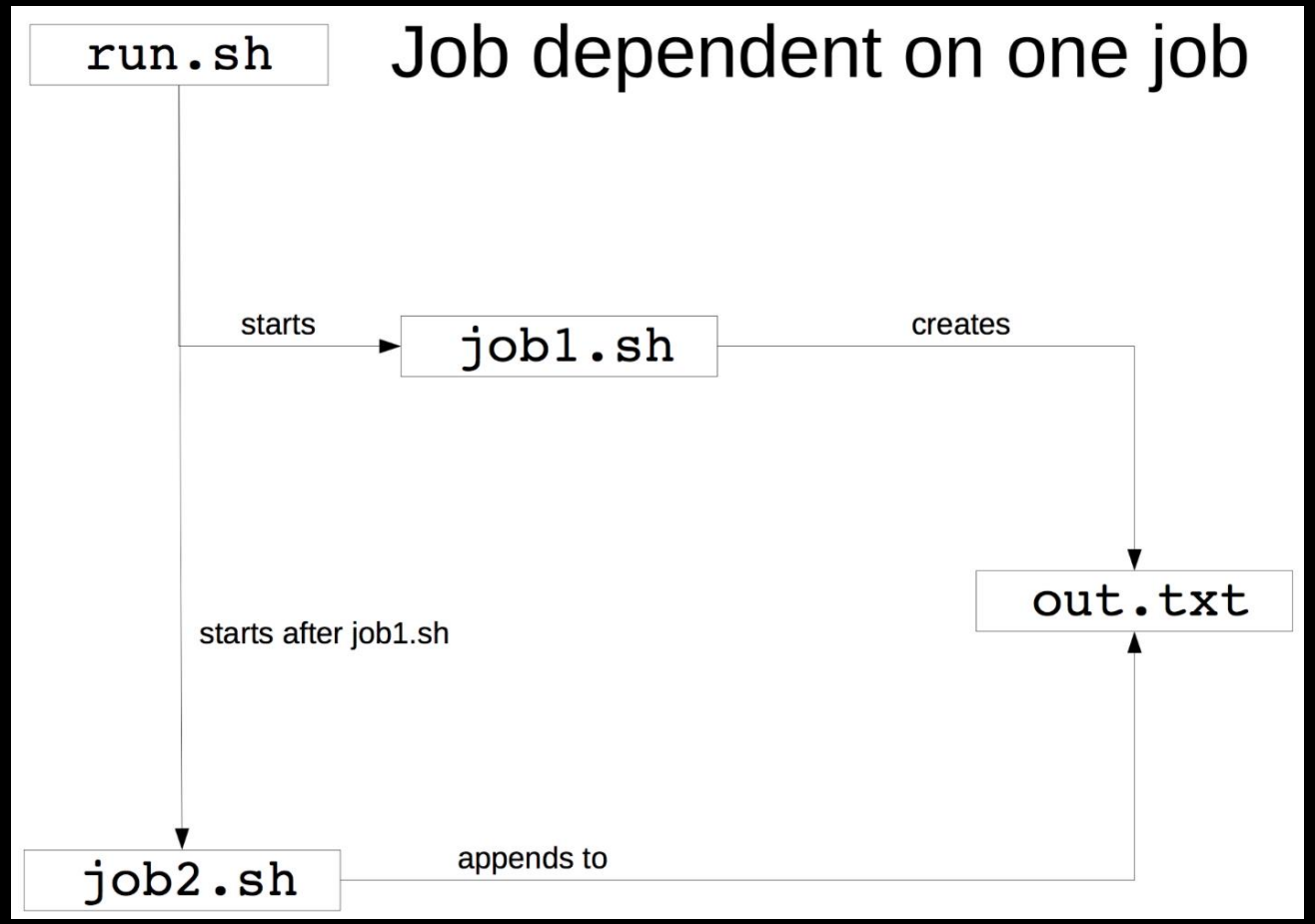
Ignoring data

- Don’t upload sensible data. Just create a “.gitignore” file in your project directory and list the files and directories to be skipped, one per line.
- What if you run git status and realize you staged something you shouldn’t? Remove it with

```
git reset <file>
```

- Then add the directory to .gitignore and stage again.
- More info: https://help.github.com/articles/adding-an-existing-project-to-github-using-the-command-line/

How to create subordinate processes



How to create subordinate processes

- Run several scripts in sequence rather than running just one script at a time. This should be done from master.

```
step1=`sbatch Code/subordinate_step_1.sh | cut -d ' ' -f 4`
step2=`sbatch --dependency=afterok:$step1 Code/subordinate_step_2.sh | cut -d ' ' -f 4`
step3=`sbatch --dependency=afterok:$step2 Code/subordinate_step_3.sh | cut -d ' ' -f 4`
step4=`sbatch --dependency=afterok:$step3 Code/subordinate_step_4.sh | cut -d ' ' -f 4`
```

- Step1 will be scheduled normally.
- Step2 will start only after Step1 has run successfully.
- Step3 will start only after Step2 has run successfully.
- Step4 will start only after Step3 has run successfully.

For more info on how to do this in Slurm
https://github.com/jorgeamaya/Peregrine/blob/master/job_dependencies.pdf

How to create subordinate processes

- subordinate_final.sh will run only after the previous scripts have finished.

```
for i in{1..3}
do
	cmd="sbatch Code/subordinate_step_${i}.sh"
	jobids+=(`$cmd | cut-d"-f4`)
done

jobsnames=$(printf":%s""${jobids[@]}")
jobsnames=${jobsnames:1}

echo$jobsnames
sbatch--dependency=afterok:$jobsnamesCode/subordinate_final.sh
```

For more info on how to do this in Slurm
https://github.com/jorgeamaya/Peregrine/blob/master/job_dependencies.pdf