



# Intro to best practices in **Jupyter, Python, Git**

# Outline



## 1. Jupyter notebooks

- a. Interactive reports
- b. The notebook
- c. Running a cell

## 2. Beautiful Python

- a. Readable code
- b. Style conventions
- c. Automate

## 3. Version control with Git

- a. Why do we need it
- b. Objects: commits, branches
- c. Actions: commit, add, push, pull, merge

## 4. Collaborate on GitHub

- a. Main features
- b. Pull requests

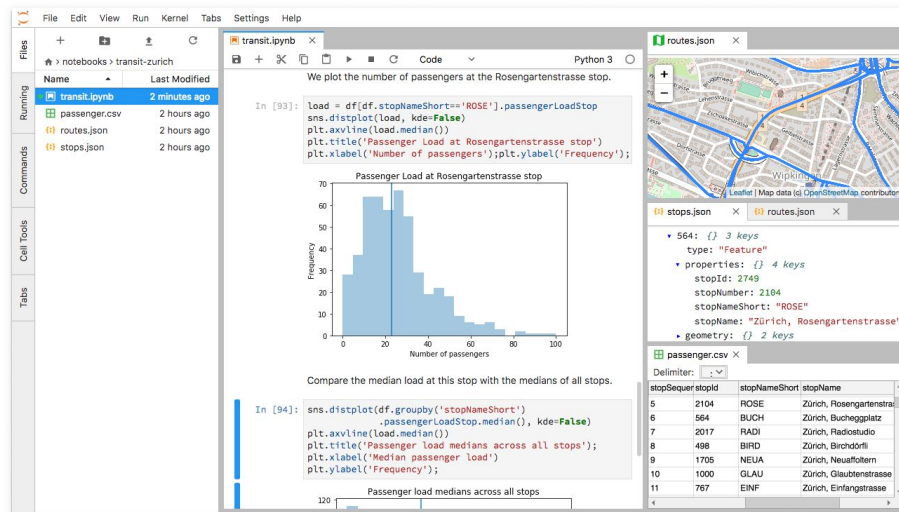
1

# Jupyter notebooks

Interactive documents

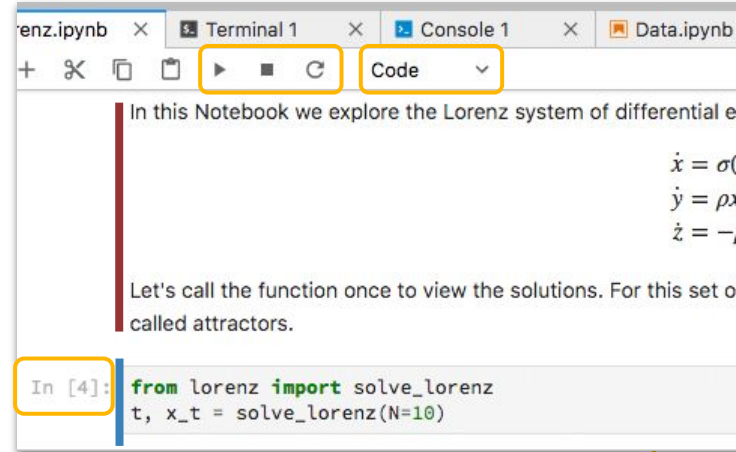
# Interactive reports

- Rich text, executable code, interactive outputs.
- Plain text JSON file (\*.ipynb)
- Perfect to lay out theory and exercises together



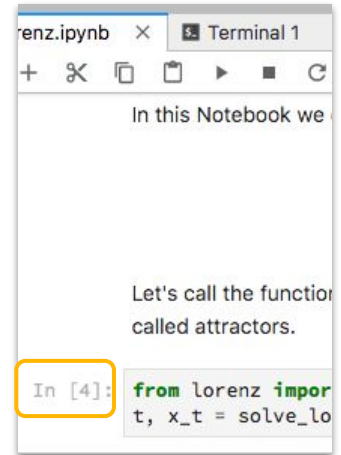
# The notebook

- Composed of blocks called **cells**.  
They can be:  
rich text (Markdown) or code (Python)



# Running a cell

- In edit mode, press **Shift+Enter** or **Ctrl+Enter**
  - Markdown cells: Double click on them
  - Code cells: Just type
- Each cell can be **run** many times
  - Careful! Position in the file does not guarantee execution order.
  - Always look at the cell index!

A screenshot of a Jupyter Notebook interface. The window title is "renz.ipynb" and "Terminal 1". The notebook content includes the text "In this Notebook we" and "Let's call the function called attractors." Below this, a code cell is shown with the prompt "In [4]:" and the code "from lorenz impor" and "t, x\_t = solve\_lo". The code cell is highlighted with a yellow border.

```
renz.ipynb x Terminal 1
+ ✂ 📄 ▶ ■ ↻
In this Notebook we

Let's call the function
called attractors.

In [4]: from lorenz impor
t, x_t = solve_lo
```

2

# Beautiful Python

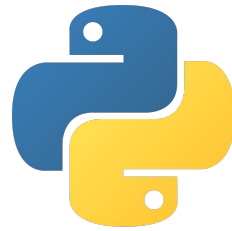
A matter of style!



# Read >>> write

- Code is more often read than written
- Spend time in writing **readable** code
- Simple is better than smart ;)
- Learn and use the language style conventions and idioms
- Good style =
  - = good calligraphy and punctuation =
  - = easier to understand!





# PEP8

- The Python style guide is detailed in [this document](#)
- Specifies things like:
  - How to name your objects (snake\_case, CamelCase, UPPER)
  - How to use horizontal whitespace (spaces, tabs)
  - How to use vertical whitespace (blank lines)
  - How to document your functions and classes
  - How long lines should be (rather short, <100 characters)



# Naming stuff

- Data, functions and instances are named with **snake\_case\_names**
- Classes are named with **CamelCaseNames**
- Constants are named with **ALL\_UPPERCASE\_NAMES**
  
- Use full words! Smart abbreviations are not that smart!

```
m = open_molecule("protein.pdb")
```

```
protein = open_molecule("protein.pdb")
```



# Whitespace

- Indent with four spaces
- Leave one space:
  - at both sides of operators (name = "protein", a > b),
  - after a comma
- No spaces:
  - In keyword arguments: some\_function(option="value")
  - After parenthesis, square brackets or curly braces
- Read the PEP8 for more details



# Automate!

- Naming your objects cannot be automated, but the other stuff can!
- Use autoformatters, like **black** and **black-nb**
- Do it before committing your code to version control

3

# Version control

With Git



# Why

- Ever done this?
  - `script.py`
  - `script_v2.py`
  - `script_v3_final.py`
  - `script_v3_final.fixed.py`
  - `script_v3+v2fallback_finalforsure.py`
  - `script_start_again_v1.py`
- Version Control Software (VCS) provides:
  - history, provenance, collaboration
- It enables workflows!



# GitHub $\neq$ Git

- **Git** is a Distributed Version Control System (DVCS) built by Linus Torvalds (creator of Linux) to streamline collaborative development on the Linux kernel
- **GitHub** is a (incredibly useful) social network that works as a centralized Git server



# Concepts

- Repository
- Diff
- Commit
- Branch





# Repositories

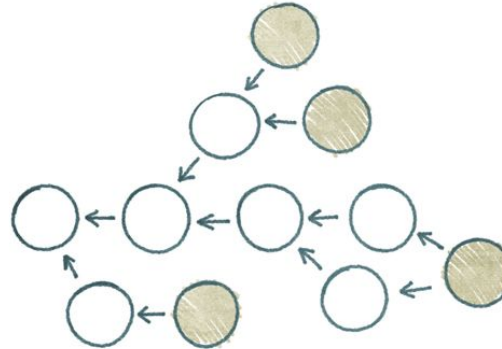
- A Git-enabled project. It contains a **.git** folder
- In other words, a directory where files changes are tracked
- A repository contains three stages or states:
  - The working directory itself: where you code
  - The staging area: prepared to save
  - The actual repository: changes are now in history



# Diffs

- If you have two versions of a script, you can use **diff**:
- These changes are tracked by Git in a directed acyclic graph

```
--- a/libraries/vecmath/src/vecmath.cpp
+++ b/libraries/vecmath/src/vecmath.cpp
@@ -1,4 +1,4 @@
-#if defined(__ANDROID__)
+#if defined(__ARM__)
   #include "neon_mathfun.h"
#else
   #if !defined(__PNACL__)
```



From [https://ericsink.com/vcbe/html/directed\\_acyclic\\_graphs.html](https://ericsink.com/vcbe/html/directed_acyclic_graphs.html)



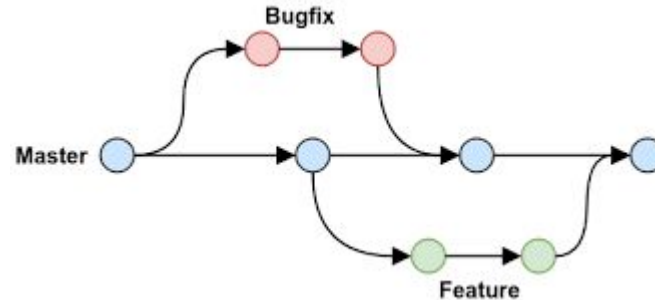
# Commits

- A commit is a set of changes (diffs) that belong together
- They can come from one or more files
- Changes in the same file can belong to different commits!
  
- Think of them as a **labeled** *box* that contain related modifications to your code
- Yes, labeled: they must contain a meaningful description. Be informative!



# Branches

- Changes do not need to be necessarily sequential
- You (and your team) can work in parallel!
- Several branches can coexist
- In the end, we expect most of them to be **merged** into the main one: **master**.



From <https://blog.programster.org/git-workflows>



# Commands

- Add
- Commit
- Push

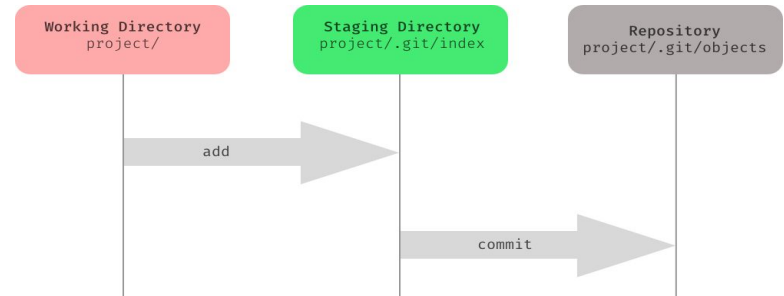
- Pull

- Branch
- Merge



# Add & commit

- `git add`  
Move changes (diffs) to the staging area
- `git commit`  
Consolidate the staging area into the repository, with a description



From <https://medium.com/hackernoon/understanding-git-index-4821a0765cf>



# Push

- Sync your changes with a remote copy
- Most of the time, it means “upload changes to GitHub”



# Pull

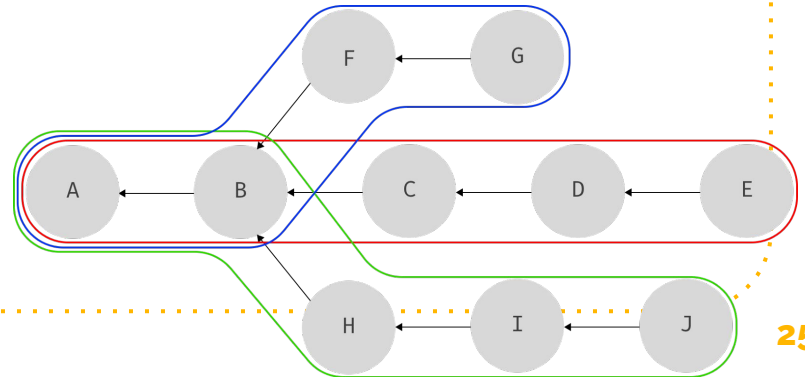
- The opposite: "Download latest changes from GitHub"
- Careful! States must be compatible. If you have been working in parallel, you might need to resolve some conflicts.





# Branch

- `git branch` will list your current branches
- To create a new branch, use `git checkout -b <name>`
- It means “create a new version of the code starting with the current state”
- To move to an existing branch: `git checkout <branch_name>`





# Merge

- When you are ready with your branch (all features have been implemented / all bugs have been fixed), you can merge into the main branch
- **git merge that\_other\_branch**
- This means “bring those changes to the current branch”
- We will do this for you on Github

# 4

## Main features in GitHub

The social network for code collaboration



*In a nutshell:*

***GitHub is Google Drive  
for source code***



# In GitHub, you...

- **Store** an up-to-date copy of your code
- **Browse** and **download** the source
- See the **history** of changes (commit log)
- Ask for **help** or suggest improvements (issues)
- **Contribute** to the project (pull request)
- **Test** new changes
- Read the **documentation** (wiki)
- Publish installers and other artifacts (**releases**)



# Social code

- Each software package has its own **repository**
- Repositories can be owned by **users** or **organizations**
- Large organizations can be optionally divided in **teams**
- **Cross-repo collaboration is encouraged by design**
  
- **Example:** I am *@jaimergp*, and belong to several organizations: *@volkamerlab*, *@choderalab*, *@openforcefield*, *@conda-forge*... Within conda-forge, I am part of the *@conda-forge/openmm* team, among others.

# History

- See all your changes
- Chronologically!
- Local equivalent:  
`git log`



volkamerlab / TeachOpenCADD

<> Code 1 Issues 1 Pull requests 5 Actions

Branch: master

Commits on Oct 23, 2019

Merge pull request #6 from jaimergp/bye-pymol ...  
dominiquesydow committed on Oct 23

Commits on Oct 22, 2019

Merge pull request #3 from jaimergp/online-api ...  
jaimergp committed on Oct 22

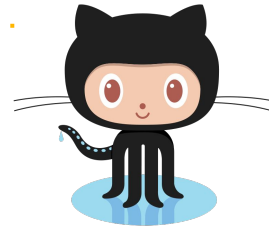
Commits on Oct 21, 2019

Consistent citations in references  
jaimergp committed on Oct 21

Apply more suggestions by @dominiquesydow  
jaimergp committed on Oct 21

Commits on Oct 16, 2019

Apply suggestions by @dominiquesydow ...  
jaimergp and dominiquesydow committed on Oct 16



# Issues

- Report problems
- Suggest features
- Get help
- No local equivalent!

🔔 324 Open ✓ 871 Closed	Author ▾	Labels ▾
🔔 <b>PowerPC &amp; ARM fixes</b> <span>enhancement</span>		
#2499 opened 15 hours ago by jaimergp		
🔔 <b>Another interesting PME alternative</b>		
#2498 opened 2 days ago by jchodera		
🔔 <b>Discrepancy in results between OpenMM versions</b>		
#2496 opened 4 days ago by jrossyra		
🔔 <b>OpenMM subversion compatibility?</b>		
#2495 opened 5 days ago by des2037		
🔔 <b>no switchDistance option for GromacsTopFile.createSystem?</b> <span>enhancement</span>		
#2492 opened 7 days ago by ruixingw		



# PRs

- Pull Requests
- Discuss changes before **git merge**
- Every upload (push) can trigger events in remote services
- **Essential in best practices**



A screenshot of a GitHub pull request interface. At the top, a commit history shows several commits with status icons (green for successful, red for failed). A modal window is open over the commit history, displaying a summary: "Some checks were not successful" with "16 failing and 1 successful checks". Below this, a list of failed checks is shown, including "Travis CI - Pull Request — Build Failed", "continuous-integration/drone/pr — Build is failing", "continuous-integration/travis-ci/pr — The Travis CI...", "openmm-feedstock — #20191203.4 failed", and "openmm-feedstock (linux linux\_cuda\_compiler\_ver...". Below the commit history, a comment from user "jaimergp" is visible, dated "3 days ago". The comment text reads: "All needed dependencies are now available! However, builds for PowerPC and ARM are failing for different reasons (although apparently related to the same thing?). ppc64le fails here (this error appears several times). We might be able to fix it by adding the suggested flag ( -DNO\_WARN\_X86\_INTRINSICS )." Below the text, there are two code blocks. The first code block shows a terminal output snippet with an error message: "Error: 'Please read comment above. Use -DNO\_WARN\_X86\_INTRINSICS to disable this error.'" The second code block shows another terminal output snippet with a compilation error: "2599 | /drone/src/build\_artifacts/openmm\_1575605706819/work/platforms/cpu/src/CpuPlatform.cpp:34: 2600 | #include &lt;smmintrin.h&gt; 2601 | 2602 | compilation terminated." The GitHub interface includes standard elements like "Author", "Member", and "Add comment" buttons.



# PRs

- You can create a PR from:
  - A **branch** in the repo
  - A **fork** (copy of your repo in a different account)

♥ Sponsor

👁 Watch 3

★ Star 0

🍴 Fork 4

## Attempt osx build and other changes #11

🔗 Open simonbray wants to merge 16 commits into conda-forge:master from simonbray:fix-1

🗨 Conversation 9   🔗 Commits 16   📄 Checks 5   📄 Files changed 10



simonbray commented on Oct 17 • edited

Member + 😊 ...

Checklist

- Used a [fork of the feedstock to propose changes](#)
- Bumped the build number (if the version is unchanged)
- Reset the build number to 0 (if the version changed)
- Re-rendered with the latest conda-smithy (Use the phrase `@conda-forge-admin, please rerender` in a comment in this PR for automated rerendering)
- Ensured the license file is being packaged.

Build for OSX and make other changes (version number, license, maintainers).

# Testing (CI)

- CI: **C**ontinuous **I**ntegration
- Every push action in a PR can trigger remote services
- Examples:
  - Azure Pipelines
  - GitHub Actions



osx osx\_python3.8  
Agent: Azure Pipelines 4

- ✓ Prepare job · succeeded
- ✓ Initialize job · succeeded
- ✓ Checkout · succeeded
- ✓ CmdLine · succeeded
- ✓ Remove homebrew · succeeded
- ✓ Add conda to PATH · succeeded
- ✓ Add conda-forge-ci-setup=2 · succeeded
- ✓ Configure conda and conda-build · succeeded
- ✓ Mangle compiler · succeeded
- ✓ Generate build number clobber file · succeeded
- ✗ Build recipe · 1 error

```
Bash exited with code '1'.
```

- ✓ Upload package · succeeded
- ✓ Post-job: Checkout · succeeded
- ✓ Finalize Job · succeeded



# How to create a Pull Request

1. Create your branch/fork
2. Make changes
3. Add, Commit and Push
4. Go to the repo on GitHub and create the PR

New pull request



# Thanks!

Any questions?