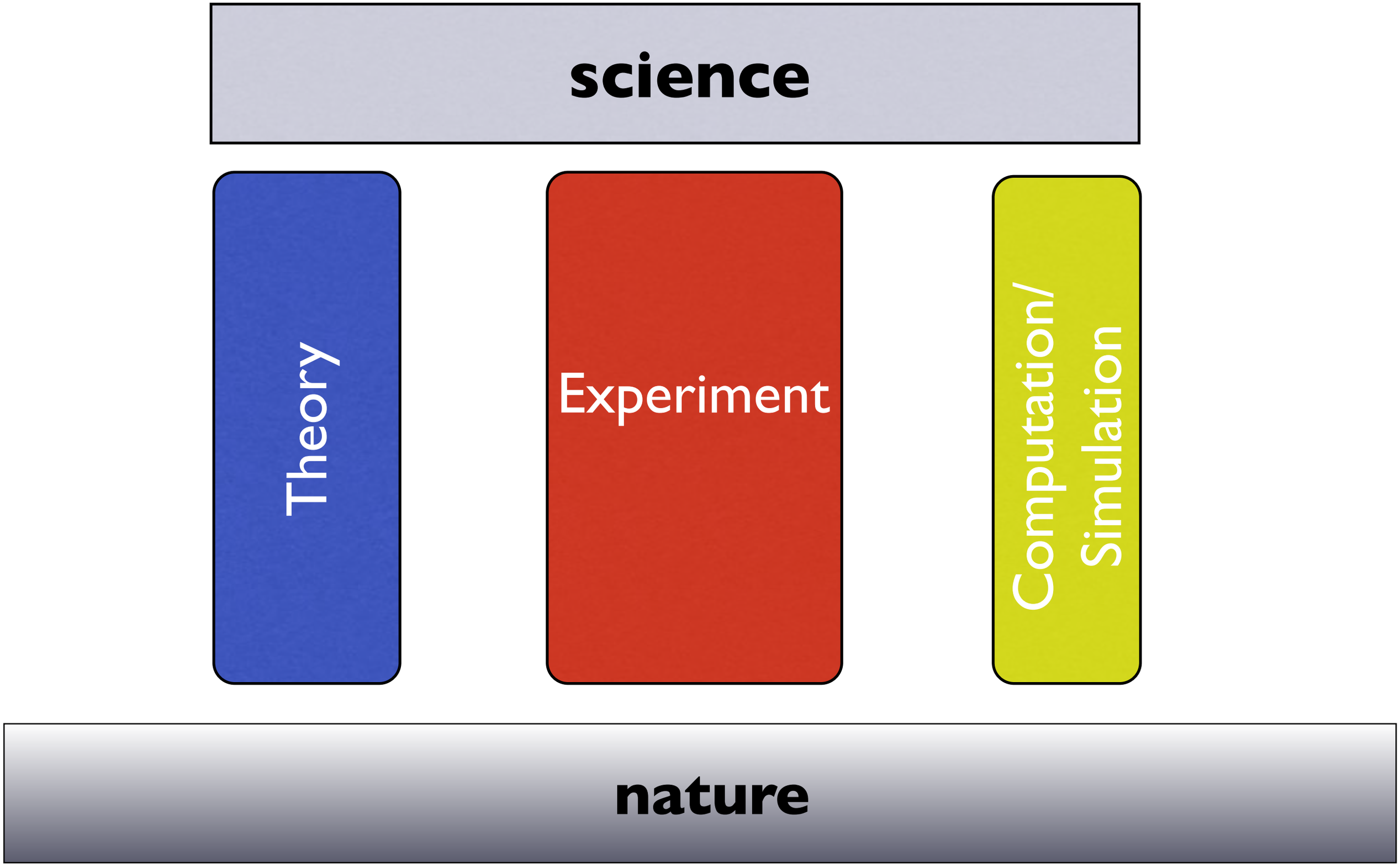
Good software engineering practices

Oliver Beckstein

Summer 2018 Beckstein Lab Mini-Workshops
August 6, 2018

https://becksteinlab.physics.asu.edu/learning/117/summer-2018-mini-workshops

Role of Software in Academic Research


science
Theory
Experiment
Computation/
Simulation
nature

What fraction of your projects requires scientific software?

1. < 10%
2. 10% – 30%
3. 30% – 50%
4. > 50%

How much of your time do you spend on writing software?

1. < 10%
2. 10% – 30%
3. 30% – 50%
4. > 50%

What do you consider **most important** in scientific software that you use?

1. fast
2. graphical user interface
3. correct
4. has documentation
5. source available
6. user-friendly
7. written in specific programming language (FORTRAN, C, C++, Python, rust, Ruby, Julia, ook, assembly, …)
8. open source (licence e.g., BSD-3, MIT, GPL, CC0, …)
9. support available
10. free ($$$)
11. runs on specific OS (Linux, macOS, Windows) or architecture (x86, GPU, Phi, …)

...ERROR

Z. Merali, *Nature* 467 (2010), 775

...why scientific programming does not compute

>

...SCIENTISTS AND THEIR SOFTWARE

A survey of nearly 2,000 researchers showed how coding has become an important part of the research toolkit, but it also revealed some potential problems.

> **45%** said scientists spend more time today developing software than five years ago."

> **38%** of scientists spend at least one fifth of their time developing software.

> Only **47%** of scientists have a good understanding of software testing.

> Only **34%** of scientists think that formal training in developing software is important.

...PRACTICING SAFE SOFTWARE

> Five tips to make scientific code more robust.

...ERROR

Z. Merali, *Nature* 467 (2010), 775

...why scientific programming does not compute

>

...SCIENTISTS AND THEIR SOFTWARE

A survey of nearly 2,000 researchers showed how coding has become an important part of the research toolkit, but it also revealed some potential problems.

> **45%** said scientists spend more time today developing software than five years ago."

> **38%** of scientists spend at least one fifth of their time developing software.

> Only **47%** of scientists have a good understanding of software testing.

> Only **34%** of scientists think that formal training in developing software is important.

...PRACTICING SAFE SOFTWARE

> Five tips to make scientific code more robust.

→ Use a version-control system:

Put source code, raw data files, parameters and other primary material into it to record what you did, and when.

...ERROR

Z. Merali, *Nature* 467 (2010), 775

...why scientific programming does not compute

>

...SCIENTISTS AND THEIR SOFTWARE

A survey of nearly 2,000 researchers showed how coding has become an important part of the research toolkit, but it also revealed some potential problems.

> **45%** said scientists spend more time today developing software than five years ago."

> **38%** of scientists spend at least one fifth of their time developing software.

> Only **47%** of scientists have a good understanding of software testing.

> Only **34%** of scientists think that formal training in developing software is important.

...PRACTICING SAFE SOFTWARE

> Five tips to make scientific code more robust.

Use a version-control system:

Put source code, raw data files, parameters and other primary material into it to record what you did, and when.

Track your materials:

Know the source of your software. Keep a record of what raw data were processed to produce a particular result, what tools were used to do the processing, and how the tools were set up.

...ERROR

Z. Merali, *Nature* 467 (2010), 775

...why scientific programming does not compute

>

...SCIENTISTS AND THEIR SOFTWARE

A survey of nearly 2,000 researchers showed how coding has become an important part of the research toolkit, but it also revealed some potential problems.

> **45%** said scientists spend more time today developing software than five years ago."

> **38%** of scientists spend at least one fifth of their time developing software.

> Only **47%** of scientists have a good understanding of software testing.

> Only **34%** of scientists think that formal training in developing software is important.

...PRACTICING SAFE SOFTWARE

> Five tips to make scientific code more robust.

Use a version-control system:

Put source code, raw data files, parameters and other primary material into it to record what you did, and when.

Track your materials:

Know the source of your software. Keep a record of what raw data were processed to produce a particular result, what tools were used to do the processing, and how the tools were set up.

Write testable software:

Build large codes from smaller, easily testable chunks.

Test the software:

And get somebody else to read it and look for bugs.

Z. Merali, *Nature* 467 (2010), 775

...why scientific programming does not compute

>

...SCIENTISTS AND THEIR SOFTWARE

A survey of nearly 2,000 researchers showed how coding has become an important part of the research toolkit, but it also revealed some potential problems.

> **45%** said scientists spend more time today developing software than five years ago."

> **38%** of scientists spend at least one fifth of their time developing software.

> Only **47%** of scientists have a good understanding of software testing.

> Only **34%** of scientists think that formal training in developing software is important.

...PRACTICING SAFE SOFTWARE

> Five tips to make scientific code more robust.

Use a version-control system:

Put source code, raw data files, parameters and other primary material into it to record what you did, and when.

Track your materials:

Know the source of your software. Keep a record of what raw data were processed to produce a particular result, what tools were used to do the processing, and how the tools were set up.

Write testable software:

Build large codes from smaller, easily testable chunks.

Test the software:

And get somebody else to read it and look for bugs.

Encourage sharing of software:

Make the code that you use in research freely available, when possible.

...ERROR

Z. Merali, *Nature* 467 (2010), 775

...why scientific programming does not compute

>

...SCIENTISTS AND THEIR SOFTWARE

A survey of nearly 2,000 researchers showed how coding has become an important part of the research toolkit, but it also revealed some potential problems.

> **45%** said scientists spend more time today developing software than five years ago."

> **38%** of scientists spend at least one fifth of their time developing software.

> Only **47%** of scientists have a good understanding of software testing.

> Only **34%** of scientists think that formal training in developing software is important.

...PRACTICING SAFE SOFTWARE

> Five tips to make scientific code more robust.

Use a version-control system:

Put source code, raw data files, parameters and other primary material into it to record what you did, and when.

Track your materials:

Know the source of your software. Keep a record of what raw data were processed to produce a particular result, what tools were used to do the processing, and how the tools were set up.

Write testable software:

Build large codes from smaller, easily testable chunks.

Test the software:

And get somebody else to read it and look for bugs.

Encourage sharing of software:

Make the code that you use in research freely available, when possible.

OPEN ACCESS Freely available online

PLOS | COMPUTATIONAL BIOLOGY

Editorial

Ten Simple Rules for Reproducible Computational Research

Geir Kjetil Sandve[1,2]*, Anton Nekrutenko[3], James Taylor[4], Eivind Hovig[1,5,6]

Citation: Sandve GK, Nekrutenko A, Taylor J, Hovig E (2013) Ten Simple Rules for Reproducible Computational Research. PLoS Comput Biol 9(10): e1003285. doi:10.1371/journal.pcbi.1003285

1. For every result, keep track how it was produced
2. Avoid manual data manipulation steps
3. Archive the exact versions of all external programs used
4. Version control all scripts
5. Record all intermediate results, when possible in standardized formats
6. For analyses that include randomness, note underlying random seeds
7. Always store raw data behind plots
8. Generate hierarchical analysis output, allowing layers of increasing details to be inspected
9. Connect textual statements to the underlying result
10. Provide public access to scripts, runs, and results

OPEN ACCESS Freely available online

PLOS | COMPUTATIONAL BIOLOGY

Editorial

Ten Simple Rules for Effective Computational Research

James M. Osborne[1,2]*, Miguel O. Bernabeu[3,4], Maria Bruna[1,2], Ben Calderhead[3], Jonathan Cooper[1], Neil Dalchau[2], Sara-Jane Dunn[2], Alexander G. Fletcher[5], Robin Freeman[2,3], Derek Groen[4], Bernhard Knapp[6], Greg J. McInerny[1,2], Gary R. Mirams[1], Joe Pitt-Francis[1], Biswa Sengupta[7], David W. Wright[3,4], Christian A. Yates[5], David J. Gavaghan[1], Stephen Emmott[2], Charlotte Deane[6]

Citation: Osborne JM, Bernabeu MO, Bruna M, Calderhead B, Cooper J, et al. (2014) Ten Simple Rules for Effective Computational Research. PLoS Comput Biol 10(3): e1003506. doi:10.1371/journal.pcbi.1003506

1. Look before you leap
2. Develop a prototype first
3. Make your code understandable to others (and yourself)
4. Don't underestimate the complexity of the task
5. Understand the mathematical, numerical, and computational methods underpinning your work
6. Use pictures: They really are worth a thousand words
7. Version control *everything*
8. Test *everything*
9. Share *everything*
10. Keep going!

OPEN ACCESS Freely available online

PLOS | BIOLOGY

Community Page

Best Practices for Scientific Computing

Greg Wilson[1]*, D. A. Aruliah[2], C. Titus Brown[3], Neil P. Chue Hong[4], Matt Davis[5], Richard T. Guy[6¤], Steven H. D. Haddock[7], Kathryn D. Huff[8], Ian M. Mitchell[9], Mark D. Plumbley[10], Ben Waugh[11], Ethan P. White[12], Paul Wilson[13]

Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, et al. (2014) Best Practices for Scientific Computing. PLoS Biol 12(1): e1001745. doi:10.1371/journal.pbio.1001745

1. Write programs for people, not computers

(a) keep program units small

(b) use meaningful, distinctive, consistent names

(c) make style and formatting consistent

2. Let the computer do the work

(a) Make the computer repeat tasks

(b) Save recent commands in file for re-use

(c) Use a build tool to automate workflows

3. Make incremental changes

(a) Work in small steps with frequent feedback and course correction ("agile")

(b) Use a version control system (VCS)

(c) Version-control all manually created content

4. Don't repeat yourself (or others)

(a) Every piece of data must have a single authoritative representation in the system

(b) Modularize code (instead of copying and pasting)

(c) Re-use code instead of rewriting it

OPEN ACCESS Freely available online

PLOS | BIOLOGY

Community Page

Best Practices for Scientific Computing

Greg Wilson[1]*, D. A. Aruliah[2], C. Titus Brown[3], Neil P. Chue Hong[4], Matt Davis[5], Richard T. Guy[6¤], Steven H. D. Haddock[7], Kathryn D. Huff[8], Ian M. Mitchell[9], Mark D. Plumbley[10], Ben Waugh[11], Ethan P. White[12], Paul Wilson[13]

Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, et al. (2014) Best Practices for Scientific Computing. PLoS Biol 12(1): e1001745. doi:10.1371/journal.pbio.1001745

5. **Plan for mistakes**
   - (a) add assertions to programs to check their operation
   - (b) Use an off-the-shelf unit testing library
   - (c) Turn bugs into test cases
   - (d) Use a symbolic debugger
6. **Optimize software only after it works correctly**
   - (a) Use a profiler to identify bottlenecks
   - (b) Write code in the highest-level language possible
7. **Document design and purpose, not mechanics**
   - (a) Document interfaces and reasons, not implementation
   - (b) Refactor code in preference to explaining how it works
   - (c) Embed the documentation in the software (and use documentation generators)
8. **Collaborate**
   - (a) Use pre-merge code reviews
   - (b) Use pair programming (bringing someone new up to speed, tricky problems)
   - (c) Use an issue tracking tool

Basic software engineering

1. Version control
2. Automated testing
3. Documentation

Basic software engineering

1. Version control
2. Automated testing
3. Documentation

https://software-carpentry.org/

SOFTWARE CARPENTRY

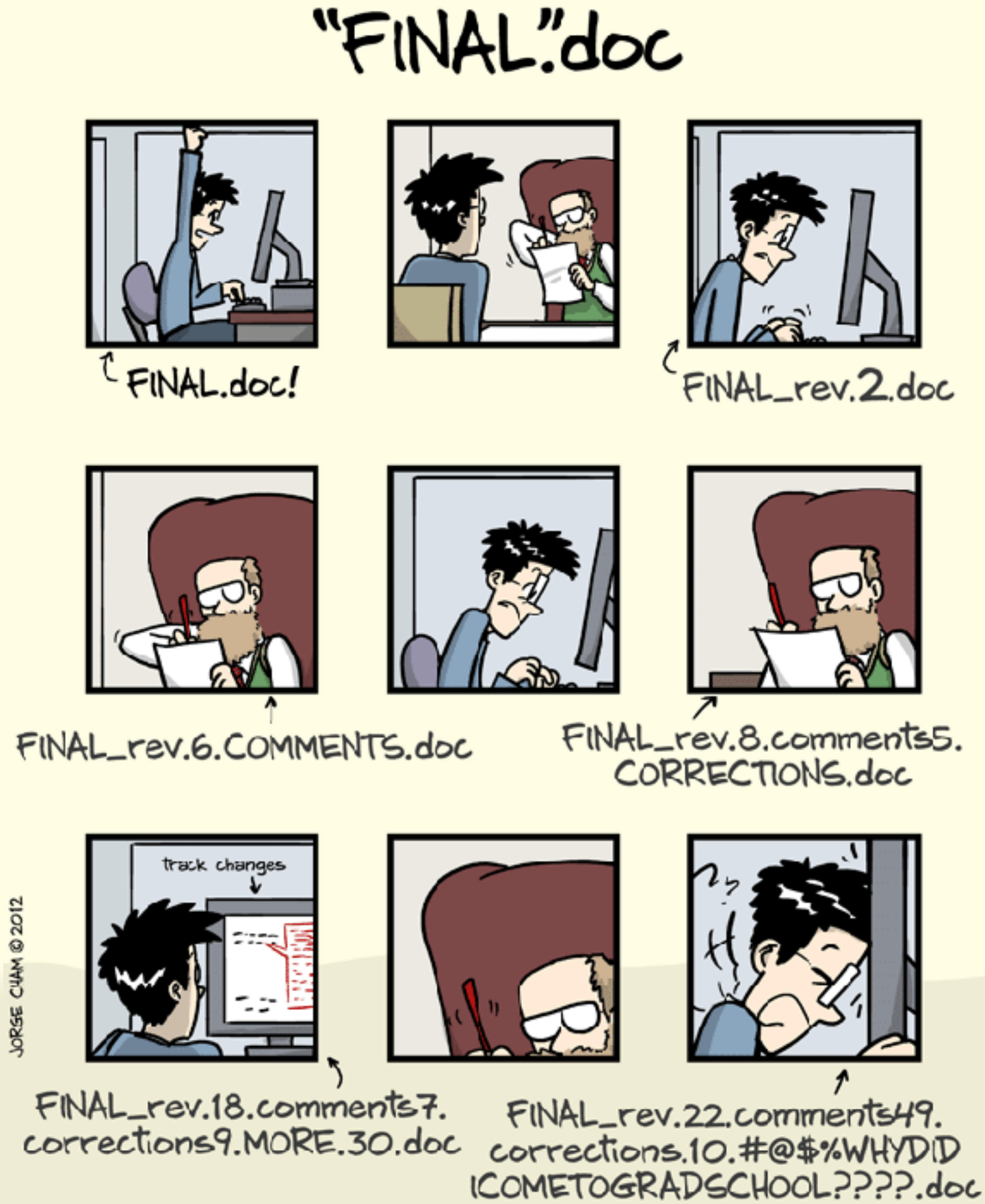
Getting Scientists to Write Better Code by Making Them More Productive

By Greg Wilson Computing In Science & Engineering, 8(6):66–69, 2006.

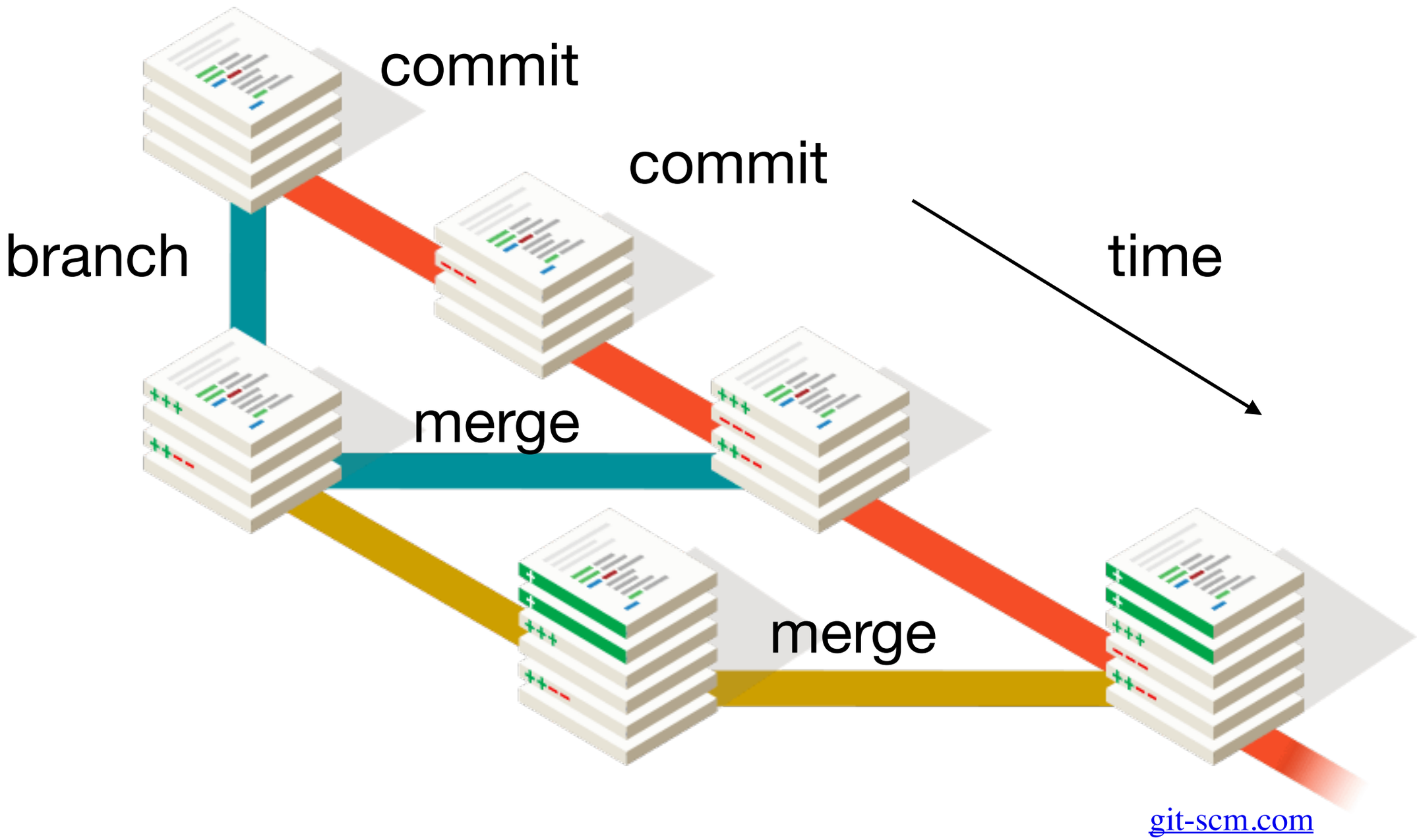
Version control with `git`

http://asu-compmethodsphysics-phy494.github.io/ASU-PHY494/2018/01/30/04_Git_basics/

Why version control ?



Version control with `git`

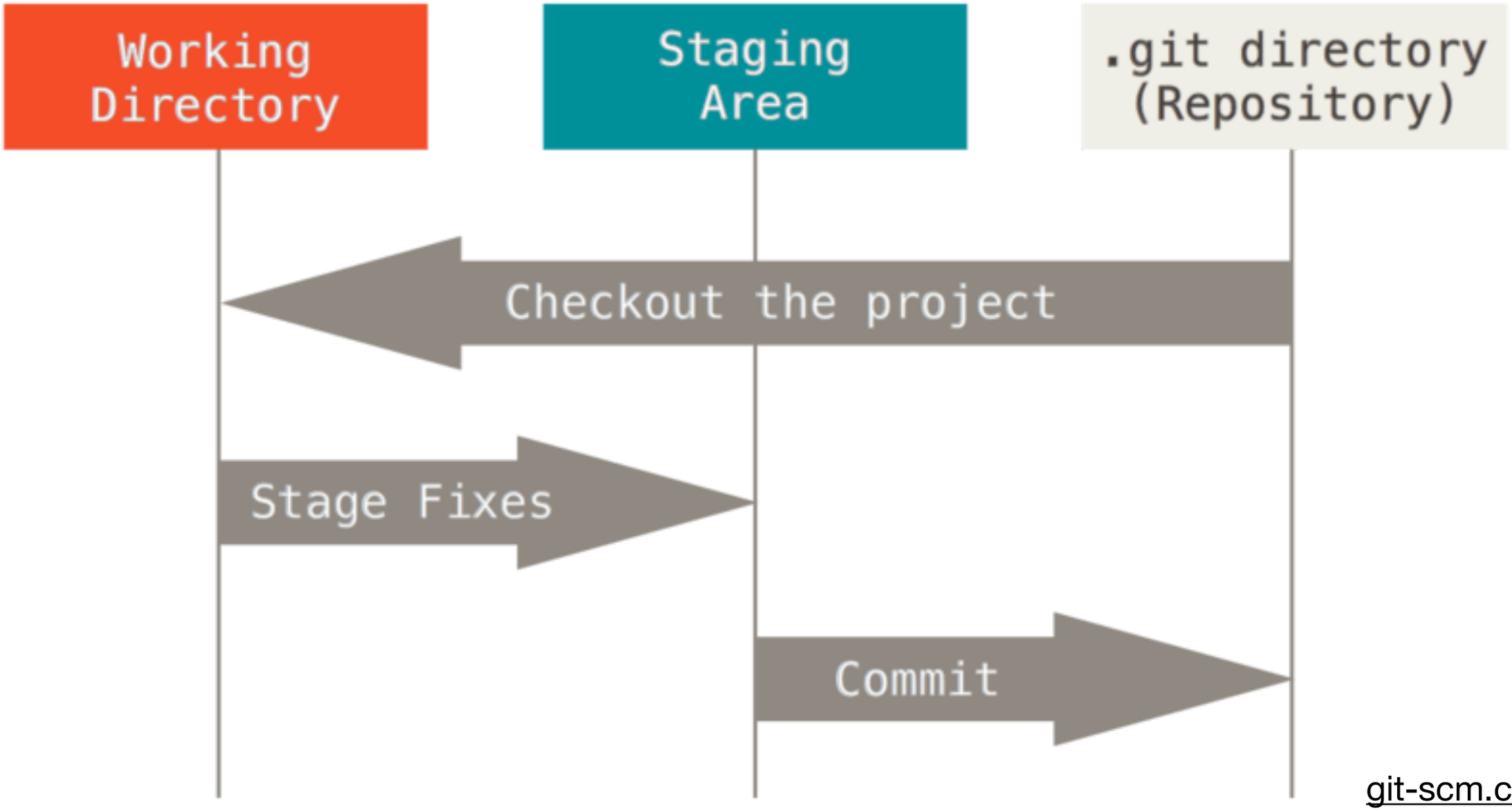


http://asu-compmethodsphysics-phy494.github.io/ASU-PHY494/2018/01/30/04_Git_basics/

Create repository

```
cd mean_calculator
git init
```

Stages of git



git-scm.com

modified **staged (“added”)** **committed**

`git` workflow

1. **modify** files in working directory
2. **selectively stage changes** that you want to include in your next commit (adds only those files to the staging area)
3. **commit** your changes (takes files from the staging area and stores them permanently in your Git repository)

Check status

```
git status
```

Use this command gratuitously!

Add files

```
git add README.md *.py
```

Commit files

```
git commit
```

When your editor pops up, enter a **commit message**: Convention:

- first line (<60 char): one line summary
- second line: blank
- third and following lines: more details The first line is mandatory (you cannot have a commit without a message), the rest is optional. *The commit message should succinctly summarize the changes in the commit.*

Check status

```
git status
```

Use this command gratuitously!

`git` workflow

1. **modify** files in working directory
2. **selectively stage changes** that you want to include in your next commit (adds only those files to the staging area)

   Adding files/changes: `git add`

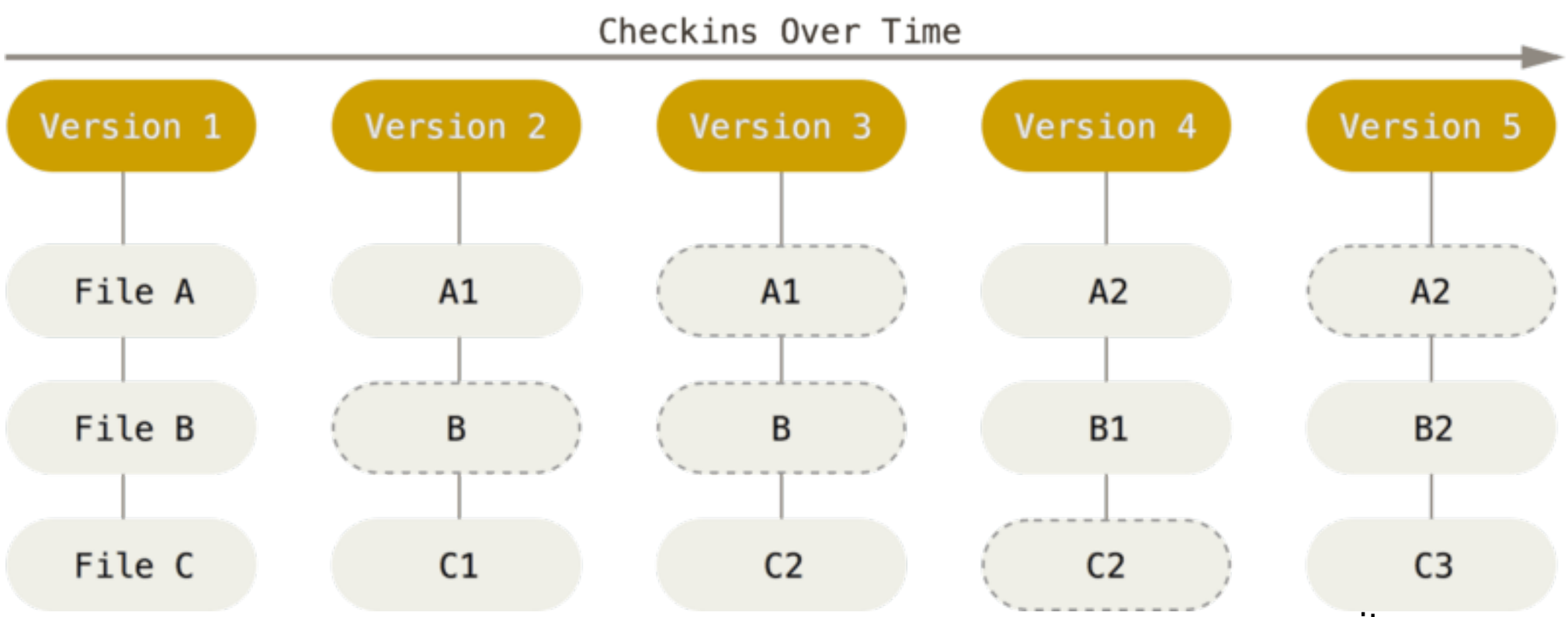
   Removing files: `git rm`

   Renaming files: `git mv`
3. **commit** your changes (takes files from the staging area and stores them permanently in your Git repository)

   ```
   git commit -m “message”
   ```

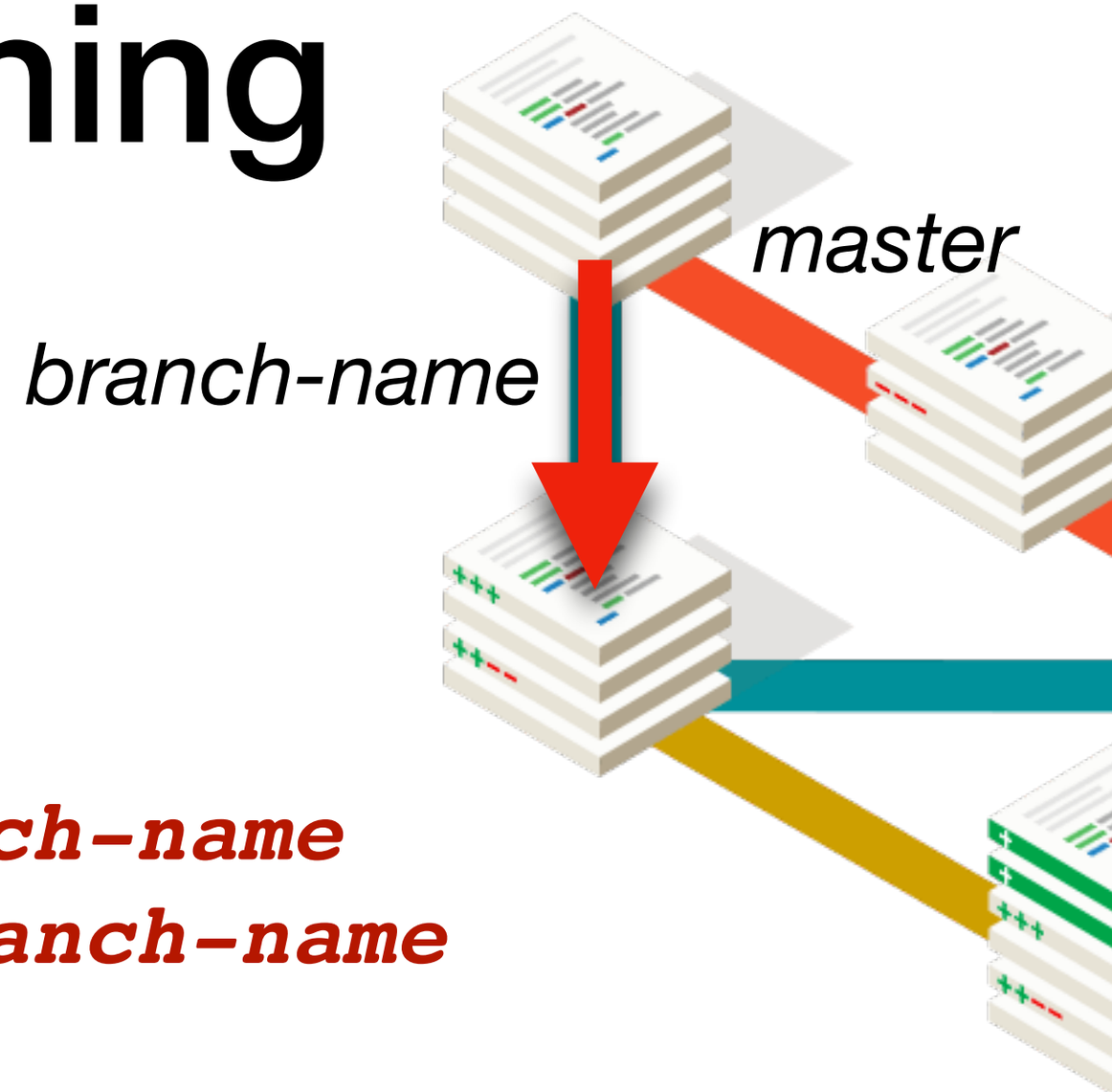
History

`git log`



git-scm.com

Branching

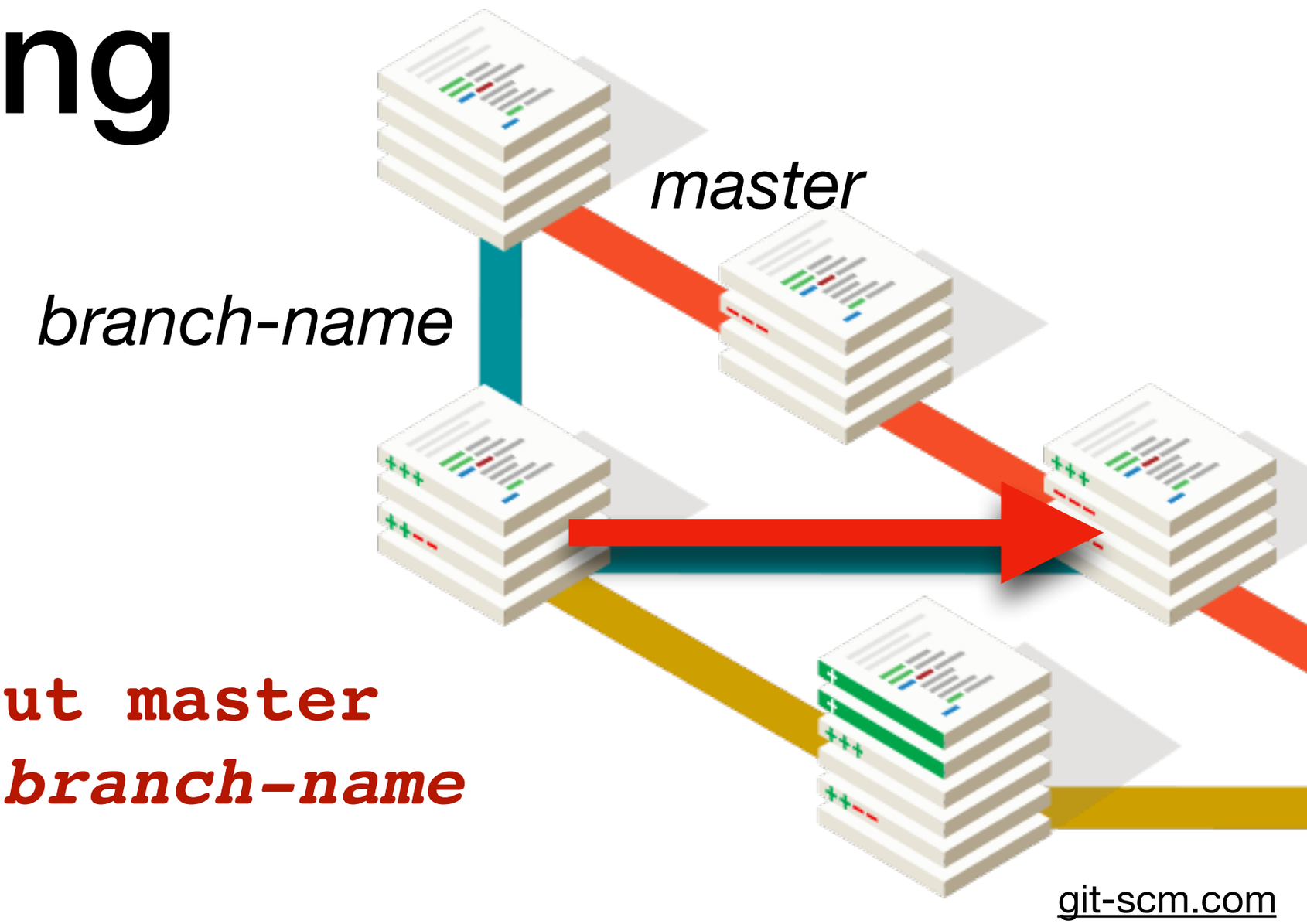


git-scm.com

```
git branch branch-name
git checkout branch-name

git branch
```

- Main branch: *master*
- Other branches: use meaningful names
- Short-cut: `git checkout -b` *`branch-name`*

Merging




```
git checkout master
git merge branch-name
```

git-scm.com

- git tries automatic merging (and often succeeds)
- But… sometimes cannot reliably merge: **manually fix conflicts** (edit files: look for conflict markers `<<<<`, `====`, `>>>>`)
- Read the output from git merge *very carefully*!

Remote repositories

```
git clone remote-repo-url
cd repo
```

Create a local repository that is linked to the remote “origin”

```
git pull
```

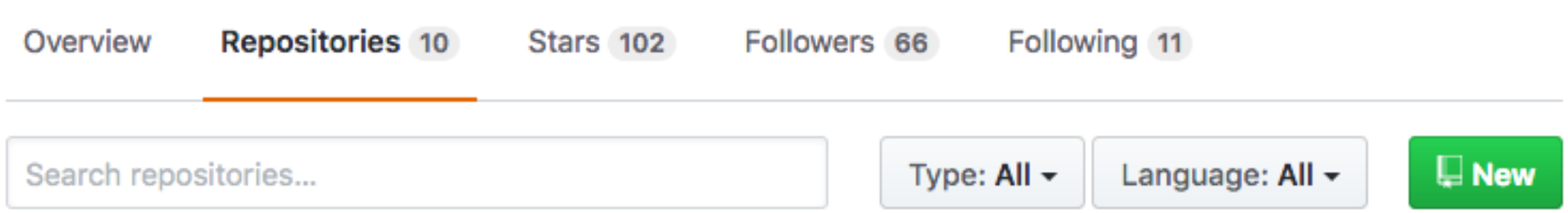
Update local repository with remote content (“read”); merge if necessary.

```
git push
```

Update remote repository with local content (“write”); *must* pull first if some remote content is not present in local.

GitHub https://github.com/

- Cloud-based repositories
- Free for open source (and education)
- Create account and create new repo *mean_calculator*



- Add LICENSE (later) and README
- Clone and add your files, then push

```
git clone https://github.com/YOUR_USER_NAME/mean_calculator
```

Testing

9/9

0800 Arctan started { 1.2700 9.037 847 025
1000 " stopped - arctan ✓ 9.037 846 ?95 correct
1300 (032) MP - MC ~~1.9826447000~~ ~~2.130476415 (-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
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.

Relay 2145
Relay 3370

1545 

Relay #70 Panel F
(moth) in relay.

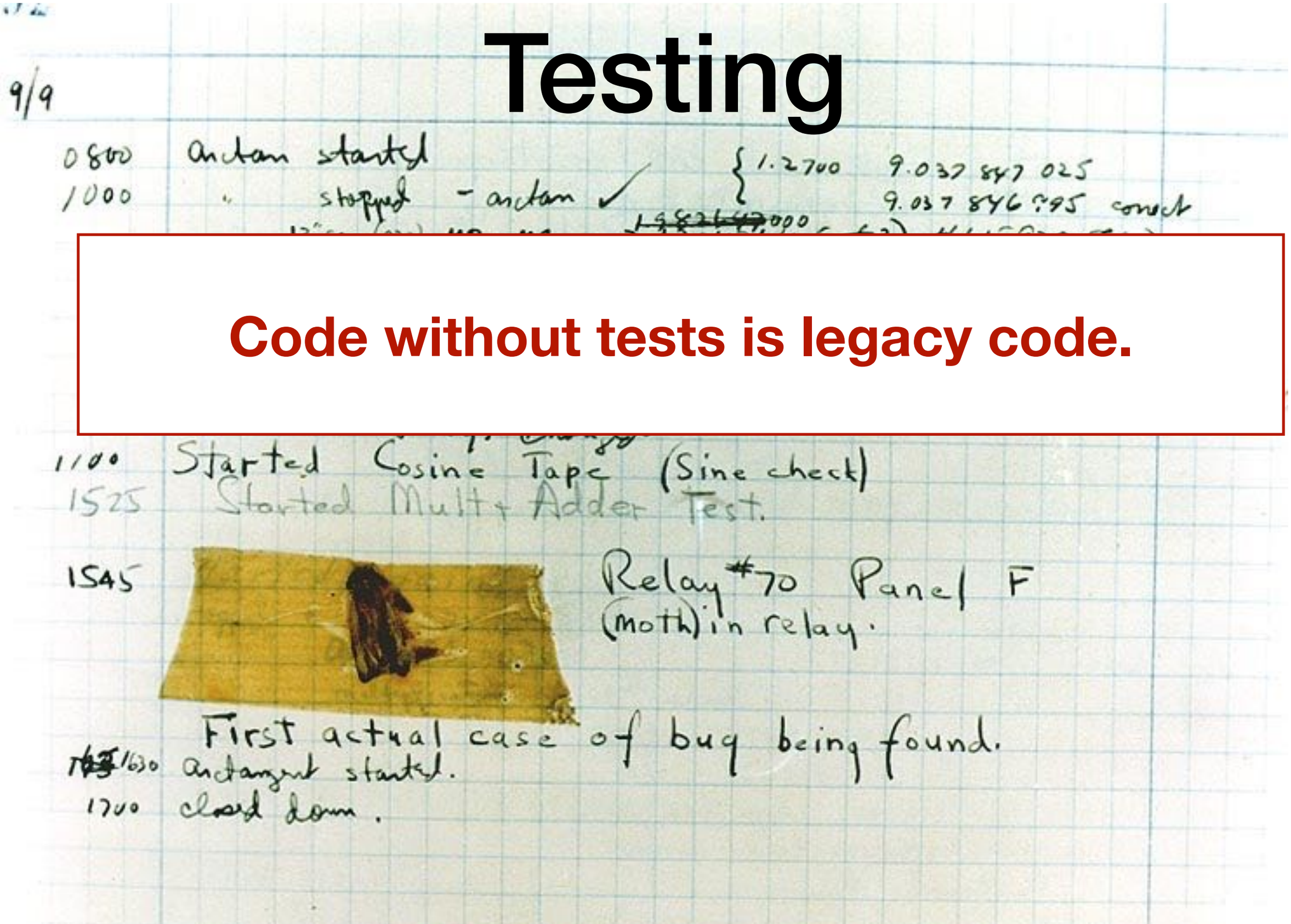
First actual case of bug being found.

1630 Arctangent started.
1700 closed down.

Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988. - U.S. Naval Historical Center Online Library Photograph NH 96566-KN via https://en.wikipedia.org/wiki/Software_bug#/media/File:H96566k.jpg

Testing

Code without tests is legacy code.



Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988. - U.S. Naval Historical Center Online Library Photograph NH 96566-KN via https://en.wikipedia.org/wiki/Software_bug#/media/File:H96566k.jpg

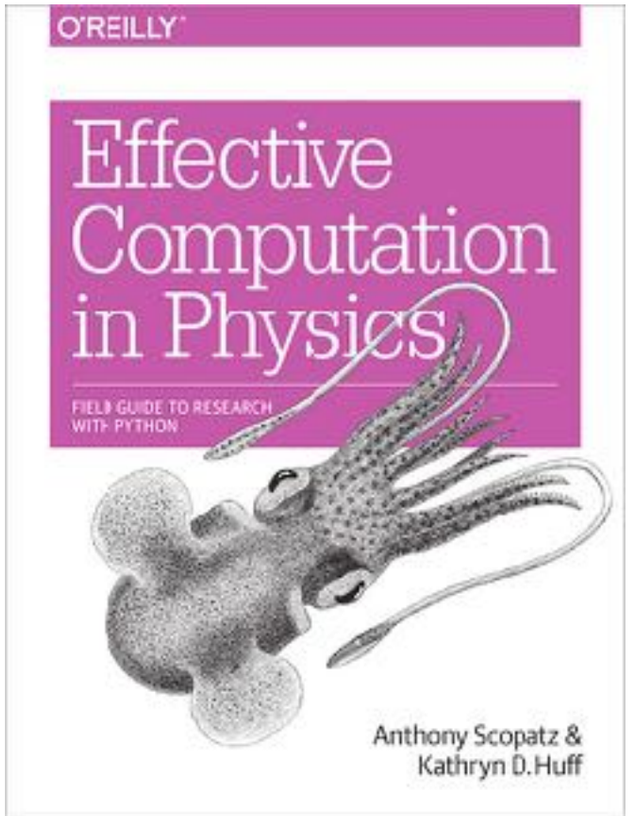
Tests

- Assert that your code produces known results.

- Tests are functions that
  - run your code
  - compare computed to known correct values
  - raise exception or return error if they disagree

- Write tests for
  - functions/methods/classes (unit tests)
  - modules/libraries (integration tests)

- Regression tests (compare to past values)



https://katyhuff.github.io/python-testing/

pytest

mean.py

```
def mean(num_list):
    return sum(num_list)/len(num_list)
```

test_mean.py

```
import pytest

from mean import mean

def test_ints():
    num_list = [1,2,3,4,5]
    obs = mean(num_list)
    exp = 3
    assert obs == exp

def test_zero():
    num_list=[0,2,4,6]
    obs = mean(num_list)
    exp = 3
    assert obs == exp
```

Run the tests

```
pytest
```

Continuous integration (CI)

- Does my software work on someone else's computer?
- With different versions of libraries/Python/ …?

CI server

1. Checks out code from repository (triggered by push or pull request)
2. Spins up instances of operating systems (Linux, macOS, windows)
3. with required software versions (e.g., Python 2.7, 3.6)
4. Installs environment (libraries, …)
5. Builds and installs software
6. Runs test scripts.
7. Checks for errors
8. Reports results (include coverage)

CI Providers

Providers with free plans (for open source)

Travis CI https://travis-ci.com/ (Linux, macOS)

Appveyor https://ci.appveyor.com/ (Windows)

Circle CI https://circleci.com/ (Linux, macOS*)

…

Coverage reporting

codecov https://codecov.io/

coveralls https://coveralls.io/

Example: `pytest` + Travis CI

Based on Katy Huff's https://katyhuff.github.io/python-testing/08-ci/

1. Create new repo on GitHub: *mean_calculator*

2. Clone locally

3. Add and commit example files from https://github.com/Becksteinlab/workshop_testing

4. Create account on https://travis-ci.com and allow *GitHub Apps Integration* to access all your repositories

5. Push changes (including the `.travis.yml` file): should trigger build on Travis-CI

6. check https://travis-ci.com/ (when logged in, shows all your builds)

Documentation

- Code without documentation is close to useless (to others and to your future self).
- No-one likes writing documentation.

Getting docs done

- Some/any documentation is better than none.
  - Require docs in your projects.
  - Keep code and docs together (easier to write and maintain)
- Use tools that make it *easy to generate docs* (HTML, PDF, …)
  - Document generators: *sphinx*, *doxygen*, …
  - Human readable formats: restructured text (reST), markdown, …
  - Automate doc creation: ReadTheDocs https://readthedocs.org/, GitHub pages https://pages.github.com/ + CI

MDAnalysis documentation

https://www.mdanalysis.org/mdanalysis/

Release: 0.18.1-dev

Date: Aug 05, 2018

Star 219

Navigation

Related Topics

Documentation overview

- Next: 1. Overview over MDAnalysis

Quick search

Go

MDAnalysis (www.mdanalysis.org) is an object oriented python toolkit to analyze molecular dynamics trajectories generated by CHARMM, Gromacs, Amber, NAMD, LAMMPS, DL_POLY and other packages; it also reads other formats (e.g., PDB files and XYZ format trajectories; see Table of supported coordinate formats and Table of Supported Topology Formats for the full lists). It can write most of these formats, too, together with atom selections for use in Gromacs, CHARMM, VMD and PyMol (see Selection exporters).

It allows one to read molecular dynamics trajectories and access the atomic coordinates through NumPy arrays. This provides a flexible and relatively fast framework for complex analysis tasks. Fairly complete atom Selection commands are implemented. Trajectories can also be manipulated (for instance, fit to a reference structure) and written out in a range of formats.

Getting involved

Please report **bugs** or **enhancement requests** through the Issue Tracker. Questions can also be asked on the mdnalysis-discussion mailing list.

The MDAnalysis community subscribes to a Code of Conduct that all community members agree and adhere to — please read it.

Installing MDAnalysis

The easiest approach to install the latest release is to use a package that can be installed either with pip or conda.

pip

Installation with pip and a *minimal set of dependencies*:

https://gromacswrapper.readthedocs.io/en/develop/

GromacsWrapper — a Python framework for Gromacs

Release: 0.6.1+70.g6b87aa8

Date: August 03, 2018

GromacsWrapper is a Python package that wraps system calls to Gromacs tools into thin classes. This allows for fairly seamless integration of the gromacs tools into Python scripts. This is generally superior to shell scripts because of Python's better error handling and superior data structures. It also allows for modularization and code re-use. In addition, commands, warnings and errors are logged to a file so that there exists a complete history of what has been done.

See INSTALL for download and installation instructions. Documentation is primarily provided through the Python doc strings (from which most of the online documentation is generated).

The source code itself is available in the GromacsWrapper git repository.

Warning

Please be aware that this is **alpha** software that most definitely contains bugs. The API is not stable yet and can change between releases.

It is *your* responsibility to ensure that you are running simulations with sensible parameters.

The package and the documentation are still in flux and any feedback, bug reports, suggestions and contributions are very welcome. See the package README: GromacsWrapper for contact details.

See also

GromacsWrapper

Table Of Contents

Next topic

Installation

This Page

Show Source

Quick search

Go

Read the Docs

orbeckst

Projects >

GromacsWrapper

View Docs

Overview | Downloads | Search | Builds | Versions | Admin

Versions

latest	Public	Edit
master	Public	Edit
develop	Public	Edit

Build a version

latest

Build version

Repository

https://github.com/Becksteinlab/GromacsWrapper

Last Built

3 days, 9 hours ago passed

Maintainers

Badge

docs passing

Tags

Sharing code

Publish your computer code: it is good enough

Freely provided working code — whatever its quality — improves programming and enables others to engage with your research, says **Nick Barnes**.

14 October 2010 | Nature 467, 753 (2010) | doi:10.1038/467753a

I am a professional software engineer and I want to share a trade secret with scientists: most professional computer software isn't very good. The code inside your laptop, television, phone or car is often badly documented, inconsistent and poorly tested. […]

That the code is a little raw is one of the main reasons scientists give for not sharing it with others. Yet, software in all trades is written to be good enough for the job intended. So if your code is good enough to do the job, then it is good enough to release — and releasing it will help your research and your field.

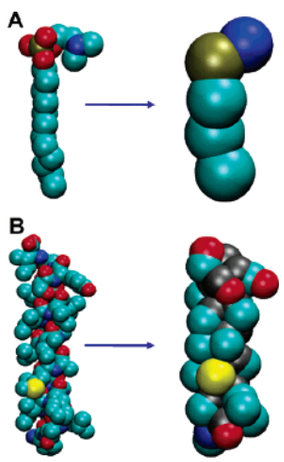
It is not common practice	It should be…
People will pick holes and demand support and bug fixes.	Open-ness is the proper scholarly approach. Nobody is entitled to demand technical support for freely provided code: if the feedback is unhelpful, ignore it.
The code is valuable intellectual property!	Rarely… almost all value is *your expertise*. Code not backed by experts = *abandonware*
Too much work to polish code!	Does not have to be perfect – good enough is good!

Available = Citations

J|A|C|S
ARTICLES
Published on Web 02/04/2006

Insertion and Assembly of Membrane Proteins via Simulation

Peter J. Bond and Mark S. P. Sansom*



P. J. Bond and M. S. P. Sansom. Bilayer deformation by the Kv channel voltage sensor domain revealed by self-assembly simulations. *Proc Natl Acad Sci* 104(8): 2631–2636, 2007. **110 citations**
P. J. Bond and M. S. P. Sansom. Insertion and assembly of membrane proteins via simulation. *JACS* 128(8):2697–2704, Mar 2006. **331 citations**

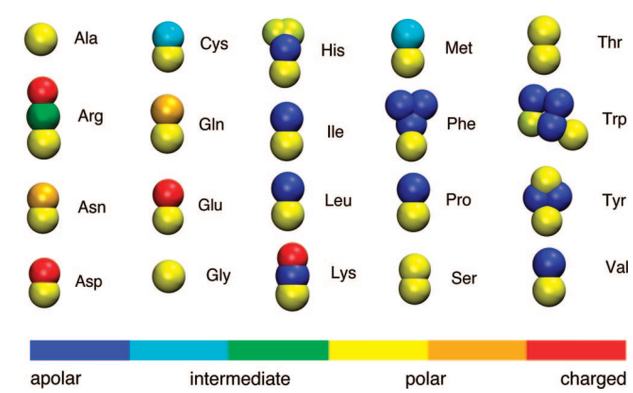
441 citations

J. Chem. Theory and Comput. **2008,** *4,* 819–834

JCTC Journal of Chemical Theory and Computation

The MARTINI Coarse-Grained Force Field: Extension to Proteins

Luca Monticelli,[†] Senthil K. Kandasamy,[‡] Xavier Periole,[§] Ronald G. Larson,[‡] D. Peter Tieleman,[†] and Siewert-Jan Marrink*,[§]



L. Monticelli, S. K. Kandasamy, X. Periole, R. G. Larson, D. P. Tieleman, and S.-J. Marrink. The MARTINI coarse-grained force field: Extension to proteins. J Chem Theory Comput, 4(5):819–834, 2008. **1476 citations**

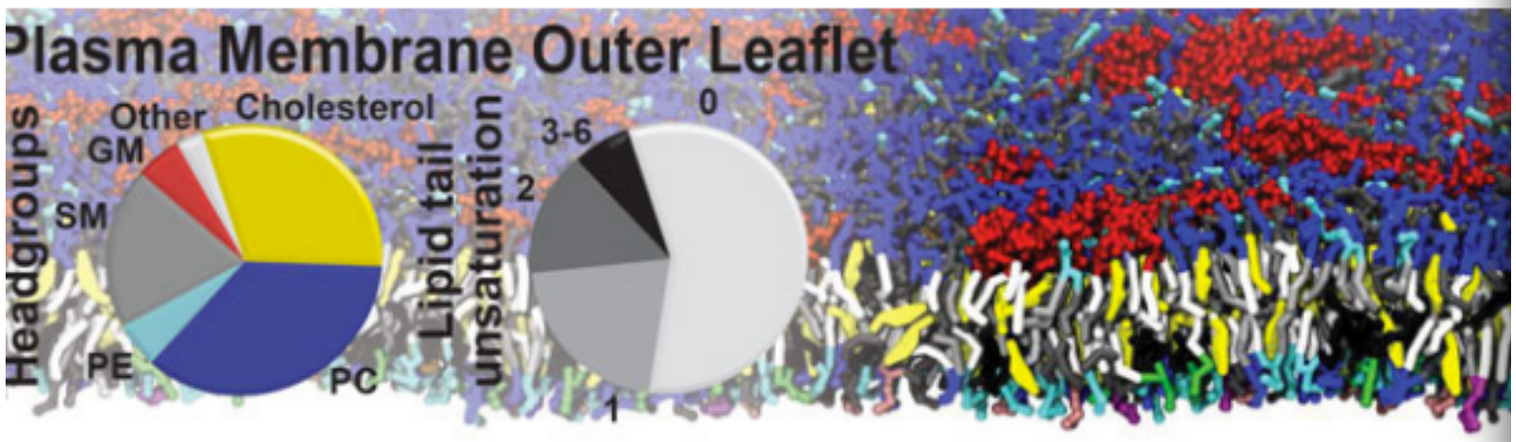
1476 citations

Google Scholar, Aug 2018

Home | About | Downloads | Tutorials | Publications | Contact | Forum

Martini

Coarse Grain Forcefield for Biomolecules



J. Chem. Theory and Comput. **2008,** *4,* 819–834

JCTC Journal of Chemical Theory and Computation

The MARTINI Coarse-Grained Force Field: Extension to Proteins

Luca Monticelli,[†] Senthil K. Kandasamy,[‡] Xavier Periole,[§] Ronald G. Larson,[‡] D. Peter Tieleman,[†] and Siewert-Jan Marrink*,[§]

Download categories

- Force field parameters
- Example applications
- **Tools**
  - **Proteins and bilayers**
  - Resolution transformation
  - Visualization

Proteins and bilayers

martinize

Last Updated: Thursday, 17 August 2017 11:59

Martinize is a python script to generate Martini protein topology and structure files based on an atomistic structure file. It replaces the old seq2itp, atom2cg and ElNeDyn scripts. The produced topology and structure files are in a format suitable for Gromacs.

http://cgmartini.nl/index.php/tools2/proteins-and-bilayers

Public repositories

Source code / VCS

- GitHub https://github.com/
- BitBucket https://bitbucket.org/
- SourceForge https://sourceforge.net/

Data / Source code (snapshot)

- Zenodo https://zenodo.org/
- figshare https://figshare.com/
- DataDryad https://www.datadryad.org/

Licensing

With material from and based on: *A short lecture on Open Licensing,* **Lorena A. Barba** (The George Washington University), https://doi.org/10.6084/m9.figshare.4516892.v1

See also: https://barbagroup.github.io/essential_skills_RRC/

Free and Open Source Software (FOSS)

- **Great impact** (e.g., Linux, Apache, git, Python, …)
- Alternative to intellectual-property instruments
- **OS licenses:** allow people to coordinate their work freely, within the confines of copyright law, while making access and wide distribution a priority

open source initiative

https://opensource.org/

Open source

Making the source public to read *is not enough.*

We must attach a **license** that allows others to modify and distribute the code.

OPEN ACCESS Freely available online

PLoS COMPUTATIONAL BIOLOGY

Education

A Quick Guide to Software Licensing for the Scientist-Programmer

Andrew Morin[1], Jennifer Urban[2], Piotr Sliz[1]*

PLoS Computational Biology | www.ploscompbiol.org 1 July 2012 | Volume 8 | Issue 7 | e1002598

- Software is creative work: **copyright** is *automatically* attached to it
- The creator (or typically their institution) owns the copyright.
- Public code is **unusable without a license:** attach a license to any code you want to make public.

Permissive vs *Copy-Left*

All are “open”: allow free **use**, **distribution**, **modification**

https://opensource.org/licenses

- Fewest restrictions
- Only require that authors be given credit
- examples: BSD, MIT License, Apache License

- Guarantees perpetual access to source code
- *Requires* derivative works to be under same license
- Considered restrictive (and industry dislikes it)
- examples: GNU GPL, LGPL

MIT License

https://choosealicense.com/licenses/mit/

A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

- Commercial use
- Distribution
- Modification
- Private use

Conditions

- License and copyright notice

Limitations

- Liability
- Warranty

```
MIT License

Copyright (c) [year] [fullname]

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
```

GNU General Public License v3.0

https://choosealicense.com/licenses/gpl-3.0/

GNU GPLv3

Permissions of this strong copyleft license are conditioned on making available complete source code of licensed works and modifications, which include larger works using a licensed work, under the same license. Copyright and license notices must be preserved. Contributors provide an express grant of patent rights.

Permissions

- Commercial use
- Distribution
- Modification
- Patent use
- Private use

Conditions

- Disclose source
- License and copyright notice
- Same license
- State changes

Limitations

- Liability
- Warranty

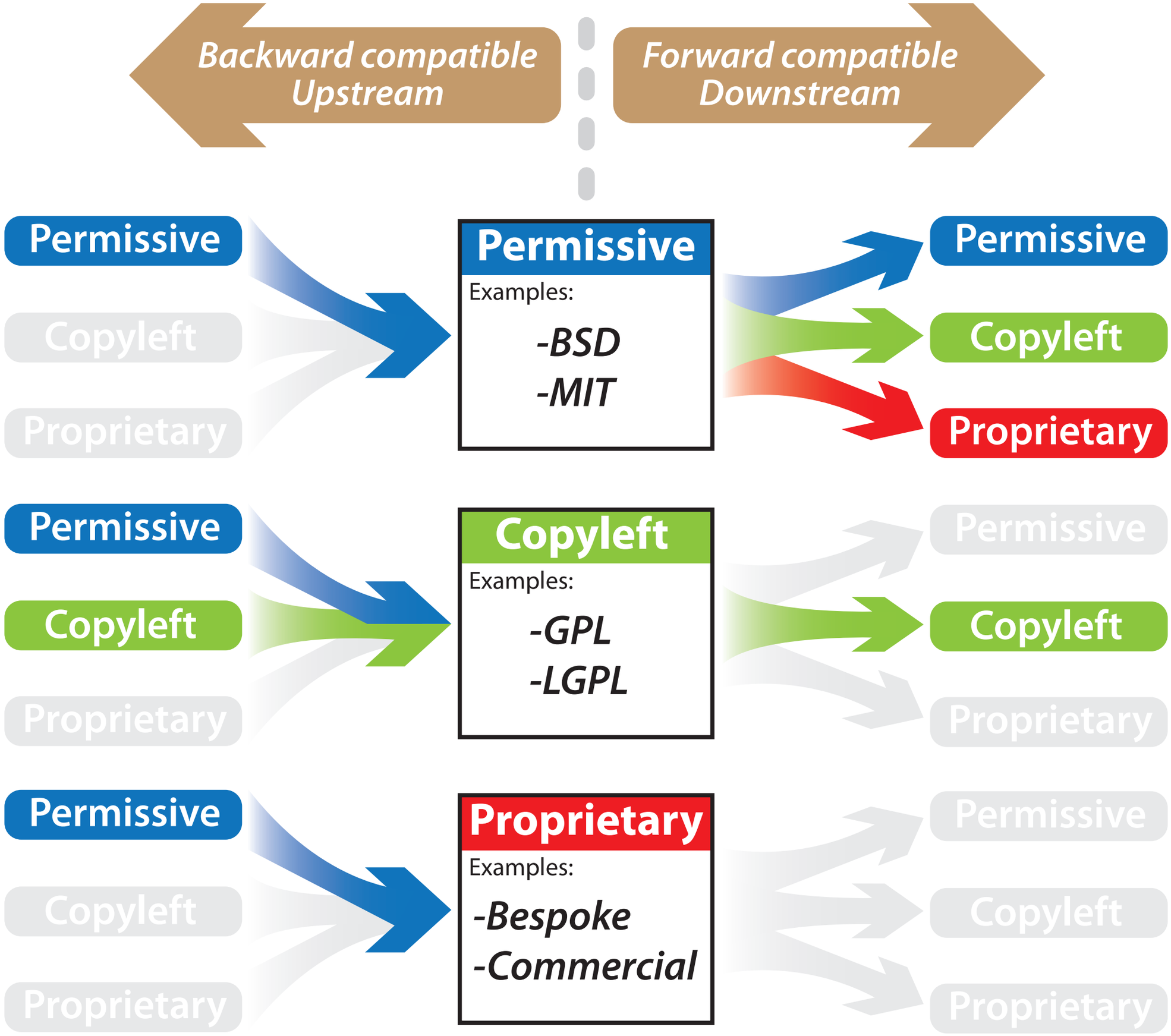
```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.
```

License compatibility

Combining code (libraries, functions) is important to create new software in a modular fashion.

Licenses must be compatible!

Not all licenses are compatible!



Morin A, Urban J, Sliz P (2012) PLoS Comput Biol 8(7): e1002598. doi:10.1371/journal.pcbi.1002598

Choosing a license

Default: simple and permissive (MIT)

Ensure openness: copy-left (GPL)

https://choosealicense.com/

Grant proposals

- NIH and NSF have data sharing guide lines and documents.
- Write: *“The research software that will be produced in this project will be released as open source under an OSI-approved license (xxx license).”*

Thank you!

- Links to material: https://becksteinlab.physics.asu.edu/learning/117/summer-2018-mini-workshops

- Videos: YouTube: Becksteinlab Mini-Workshop 2018