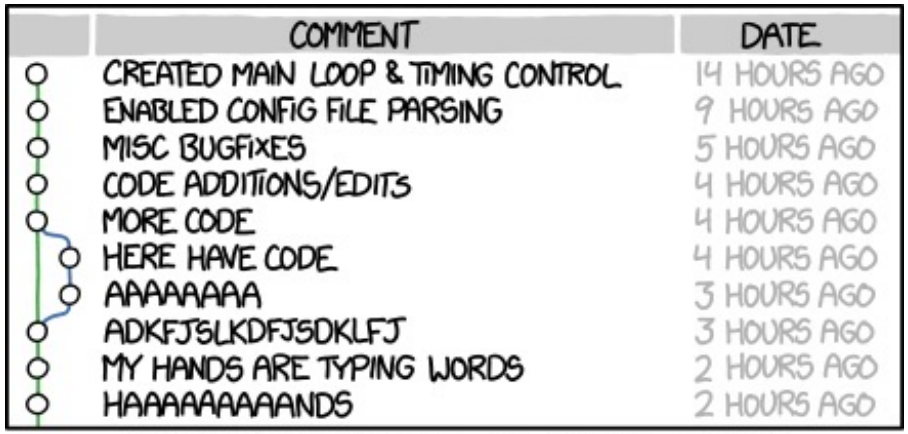
ENGSCI 233 Lecture 8

Specifications and Version Control



AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Bryan Ruddy and Andreas Kempa-Liehr

UNIVERSITY OF AUCKLAND
Waipapa Taumata Rau
NEW ZEALAND

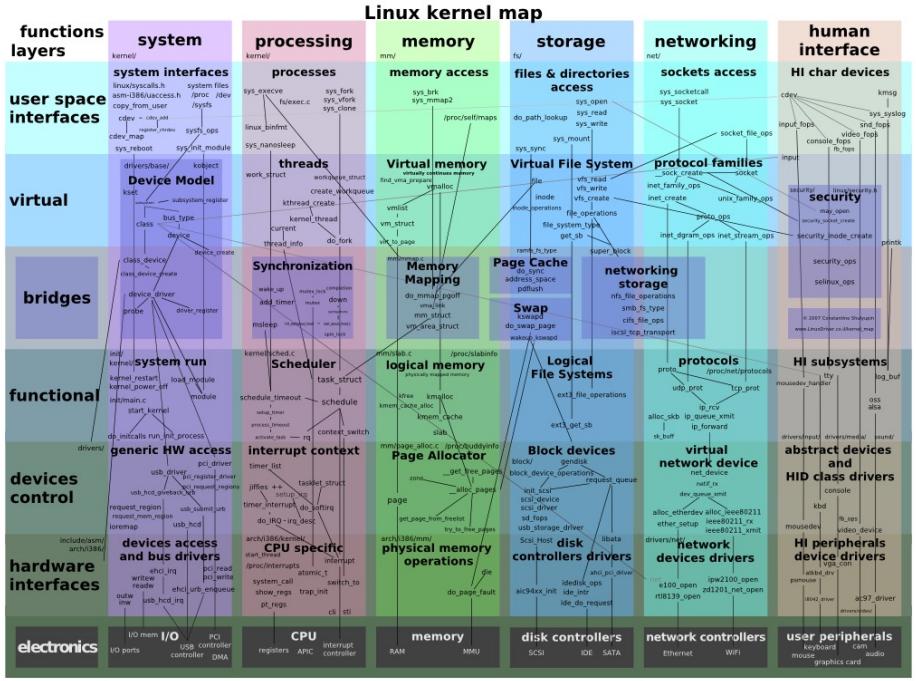
ENGINEERING
DEPARTMENT OF ENGINEERING SCIENCE

Today’s objectives:

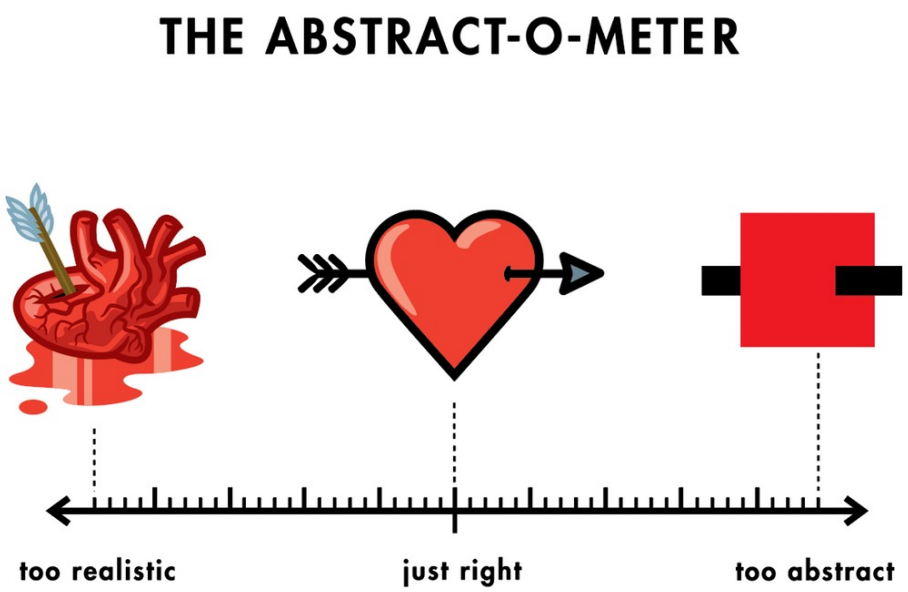
- Write clear specifications for functions
- Understand the key parts of a function specification
- Understand the operation of distributed version control (git)
- Use git version control to maintain a software project

How can we achieve great things? (with software)

- Programs can get big.
- Divide and conquer is the only way to go.
- How do you write the little bits without having to do it all at once?



Abstraction is a powerful tool.



- You don’t need to know what’s inside a function to use it
- You don’t need to know where a function will be used to write it
- You just have to know what it is meant to do!

How do we make abstraction work?

- The implementor and client of a function must agree on its purpose and interface.
- The interface is the set of inputs and outputs.
- The purpose is how the outputs relate to the inputs.

```
[In [1]: class SimpleCalculator:
    ...:     """Calculator class for demonstrating doc strings."""
    ...:     def add(self, a, b, c=0, d=0):
    ...:         """Add up to four numbers."""
    ...:         return a + b + c + d
    ...:

[In [2]: SimpleCalculator.__doc__
Out[2]: 'Calculator class for demonstrating doc strings.'

[In [3]: calculator = SimpleCalculator()

[In [4]: calculator.__doc__
Out[4]: 'Calculator class for demonstrating doc strings.'

[In [5]: calculator?
Type:        SimpleCalculator
String form: <__main__.SimpleCalculator object at 0x7fae307a16d0>
Docstring:   Calculator class for demonstrating doc strings.
```

What is your purpose?

```
In [8]: str(4)
Out[8]: '4'

In [9]: print(str.__doc__)
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or
errors is specified, then the object must expose a data buffer
that will be decoded using the given encoding and error handler.
Otherwise, returns the result of object.__str__() (if defined)
or repr(object).
encoding defaults to sys.getdefaultencoding().
errors defaults to 'strict'.
```

- Implementation-neutral description
- “sort a list of numbers”
- “solve a system of linear equations”
- “display a dialog box on the screen”
- “find the shortest path through a network”

Document your ins and outs.

NumPy/SciPy Docstrings Example

- List the inputs to the function
- List the things output via the return statement
- Note the inputs that are modified and used as outputs
- Note their data types
- State their meanings, not just names

```
In [15]: import numpy as np

In [16]: np.argwhere?
Signature: np.argwhere(a)
Docstring:
Find the indices of array elements that are non-zero, grouped by element.

Parameters
----------
a : array_like
    Input data.

Returns
-------
index_array : (N, a.ndim) ndarray
    Indices of elements that are non-zero. Indices are grouped by element.
    This array will have shape ``(N, a.ndim)`` where ``N`` is the number of
    non-zero items.

See Also
--------
where, nonzero

Notes
-----
``np.argwhere(a)`` is almost the same as ``np.transpose(np.nonzero(a))``,
but produces a result of the correct shape for a 0D array.

The output of ``argwhere`` is not suitable for indexing arrays.
For this purpose use ``nonzero(a)`` instead.

Examples
--------
>>> x = np.arange(6).reshape(2,3)
>>> x
array([[0, 1, 2],
       [3, 4, 5]])
>>> np.argwhere(x>1)
array([[0, 2],
       [1, 0],
       [1, 1],
       [1, 2]])
File:      /opt/anaconda3/lib/python3.8/site-packages/numpy/core/numeric.py
Type:      function
```

NumPy/SciPy Docstrings Example

Your docstrings will generate the documentation

```
[In [15]: import numpy as np

[In [16]: np.argwhere?
Signature: np.argwhere(a)
Docstring:
Find the indices of array elements that are non-zero, grouped by element.

Parameters
----------
a : array_like
    Input data.

Returns
-------
index_array : (N, a.ndim) ndarray
    Indices of elements that are non-zero. Indices are grouped by element.
    This array will have shape ``(N, a.ndim)`` where ``N`` is the number of
    non-zero items.

See Also
--------
where, nonzero

Notes
-----
``np.argwhere(a)`` is almost the same as ``np.transpose(np.nonzero(a))``,
but produces a result of the correct shape for a 0D array.

The output of ``argwhere`` is not suitable for indexing arrays.
For this purpose use ``nonzero(a)`` instead.

Examples
--------
>>> x = np.arange(6).reshape(2,3)
>>> x
array([[0, 1, 2],
       [3, 4, 5]])
>>> np.argwhere(x>1)
array([[0, 2],
       [1, 0],
       [1, 1],
       [1, 2]])
File:      /opt/anaconda3/lib/python3.8/site-packages/numpy/core/numeric.py
Type:      function
```

NumPy User Guide API reference Development Learn

Search the docs ...

- numpy.sort
- numpy.lexsort
- numpy.argsort
- numpy.ndarray.sort
- numpy.msort
- numpy.sort_complex
- numpy.partition
- numpy.argpartition
- numpy.argmax
- numpy.nanargmax
- numpy.argmin
- numpy.nanargmin
- numpy.argwhere
- numpy.nonzero
- numpy.flatnonzero
- numpy.where
- numpy.searchsorted
- numpy.extract
- numpy.count_nonzero
- Statistics
- Test Support (`numpy.testing`)
- Window functions
- Typing (`numpy.typing`)
- Global State
- Packaging (`numpy.distutils`)
- NumPy Distutils - Users Guide
- NumPy C-API

numpy.argwhere

`numpy.argwhere(a)` [source]

Find the indices of array elements that are non-zero, grouped by element.

Parameters: a : *array_like*
Input data.

Returns: index_array : *(N, a.ndim) ndarray*
Indices of elements that are non-zero. Indices are grouped by element. This array will have shape `(N, a.ndim)` where `N` is the number of non-zero items.

See also

`where`, `nonzero`

Notes

`np.argwhere(a)` is almost the same as `np.transpose(np.nonzero(a))`, but produces a result of the correct shape for a 0D array.

The output of `argwhere` is not suitable for indexing arrays. For this purpose use `nonzero(a)` instead.

Examples

```
>>> x = np.arange(6).reshape(2,3)
>>> x
array([[0, 1, 2],
       [3, 4, 5]])
>>> np.argwhere(x>1)
array([[0, 2],
       [1, 0],
       [1, 1],
       [1, 2]])
```

```python
class Animal:
    """
    A class used to represent an Animal
    """

    says_str = "A {name} says {sound}"

    def __init__(self, name, sound, num_legs=4):
        self.name = name
        self.sound = sound
        self.num_legs = num_legs

    def says(self, sound=None):
        """Prints what the animals name is and what sound it makes.

        If the argument `sound` isn't passed in, the default Animal
        sound is used.

        Parameters
        ----------
        sound : str, optional
            The sound the animal makes (default is None)

        Raises
        ------
        NotImplementedError
            If no sound is set for the animal or passed in as a
            parameter.
        """

        if self.sound is None and sound is None:
            raise NotImplementedError("Silent Animals are not supported!")

        out_sound = self.sound if sound is None else sound
        print(self.says_str.format(name=self.name, sound=out_sound))
```

Your code might raise errors.

- List any error conditions you intend to generate
- Say what they mean

Different docstring formats

Formatting Type	Description	Supported by Sphynx	Formal Specification
Google docstrings	Google's recommended form of documentation	Yes	No
reStructuredText	Official Python documentation standard; Not beginner friendly but feature rich	Yes	Yes
NumPy/SciPy docstrings	NumPy's combination of reStructuredText and Google Docstrings	Yes	Yes
Epytext	A Python adaptation of Epydoc; Great for Java developers	Not officially	Yes

https://realpython.com/documenting-python-code/#docstring-formats

Your code may need resources.

- Does it save files?
- Does it take input from hardware?
- Does it use a lot of memory?
- Does it cause physical output?
- Does it require a specific processor?
- Does it communicate with other computers?

Example

README.md

build passing | codecov 87% | DOI 10.5281/zenodo.6047460

GPU-enhanced Neuronal Networks (GeNN)

GeNN is a GPU-enhanced Neuronal Network simulation environment based on code generation for Nvidia CUDA.

INSTALLING GeNN

These instructions are for installing the release obtained from https://github.com/genn-team/genn/releases. For full instructions and cloning git branches of the project, see the documentation available at http://genn-team.github.io/genn/

WINDOWS INSTALL

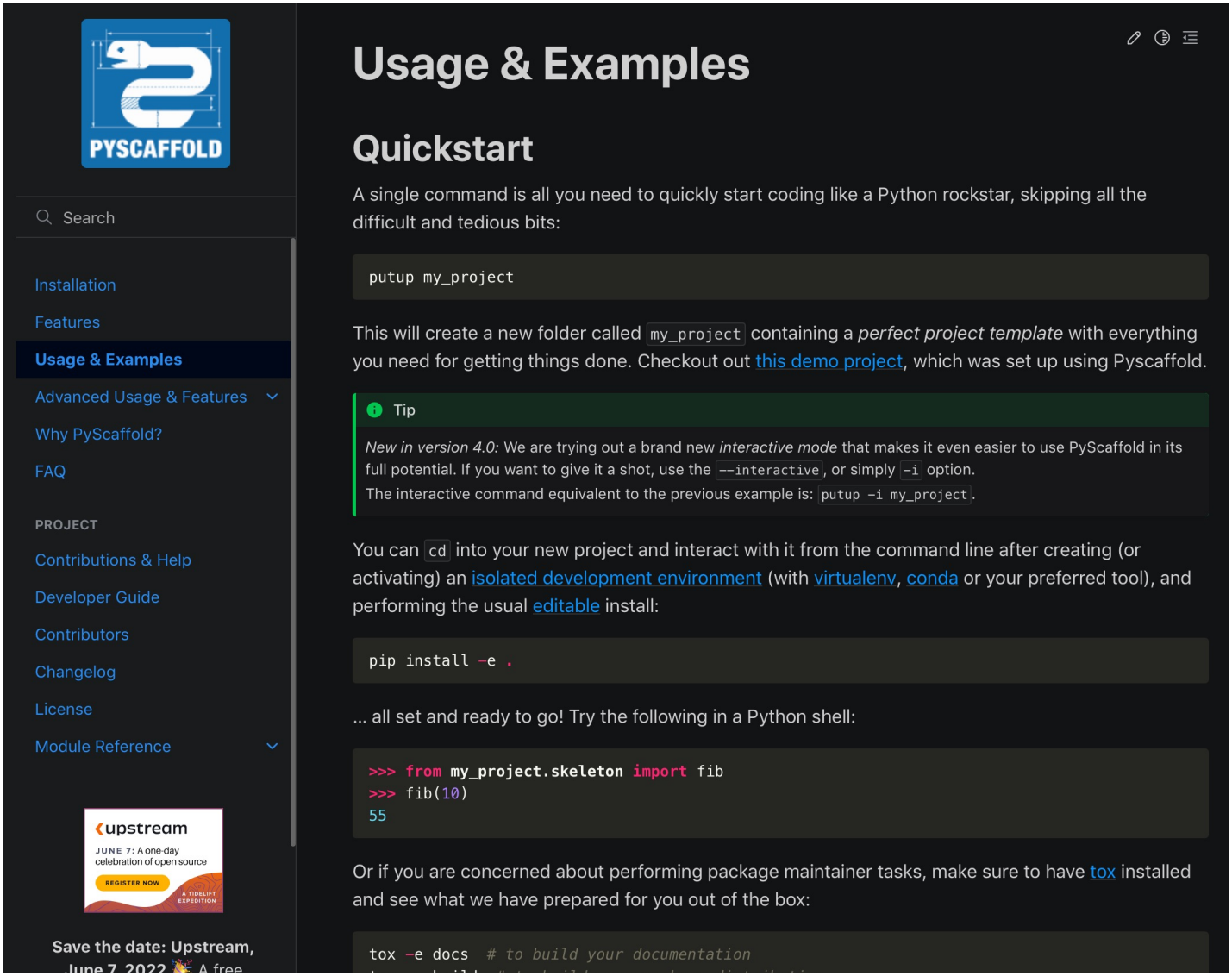
1. Download and unpack GeNN.zip to a convenient location, then download and install the Microsoft Visual C++ compiler and IDE from: http://www.visualstudio.com/en-us/downloads. Be sure to select the 'Desktop development with C++' configuration' and the 'Windows 8.1 SDK' and 'Windows Universal CRT' individual components. If your machine has an NVIDIA GPU, then download and install a compatible version of the Nvidia CUDA toolkit from: https://developer.nvidia.com/cuda-downloads. Note that the latest version of Visual Studio is not necessarily compatible with the latest version of the CUDA toolkit.

Create your own Python package

pip install --upgrade pyscaffold

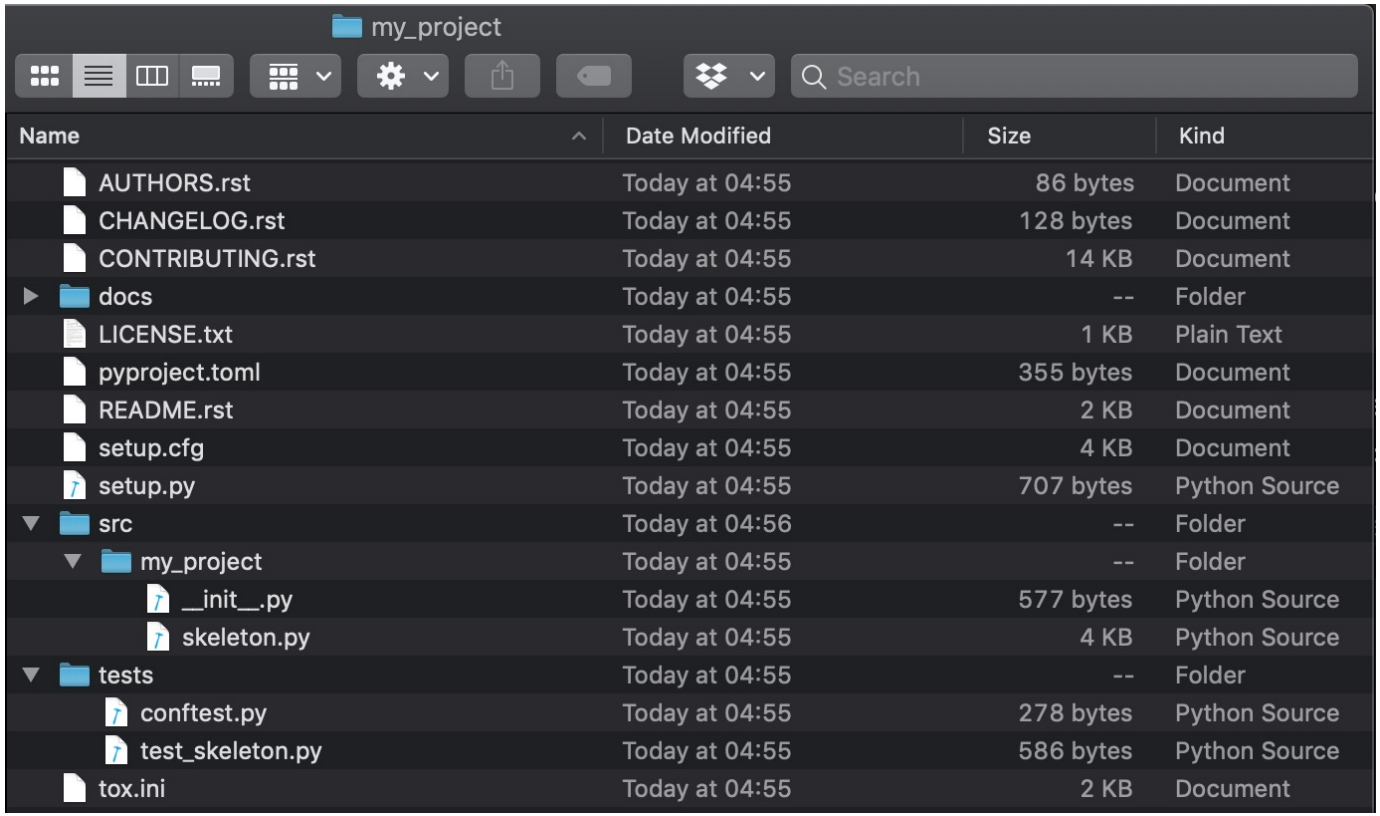
or

conda install -c conda-forge pyscaffold



pyScaffold

putup my_project



Name	Date Modified	Size	Kind
AUTHORS.rst	Today at 04:55	86 bytes	Document
CHANGELOG.rst	Today at 04:55	128 bytes	Document
CONTRIBUTING.rst	Today at 04:55	14 KB	Document
docs	Today at 04:55	--	Folder
LICENSE.txt	Today at 04:55	1 KB	Plain Text
pyproject.toml	Today at 04:55	355 bytes	Document
README.rst	Today at 04:55	2 KB	Document
setup.cfg	Today at 04:55	4 KB	Document
setup.py	Today at 04:55	707 bytes	Python Source
src	Today at 04:56	--	Folder
my_project	Today at 04:55	--	Folder
__init__.py	Today at 04:55	577 bytes	Python Source
skeleton.py	Today at 04:55	4 KB	Python Source
tests	Today at 04:55	--	Folder
conftest.py	Today at 04:55	278 bytes	Python Source
test_skeleton.py	Today at 04:55	586 bytes	Python Source
tox.ini	Today at 04:55	2 KB	Document

Further reading

- **PEP 257 – Docstring Conventions**
  https://peps.python.org/pep-0257/

- Cookiecutter Data Science
  A logical, reasonably standardized, but flexible project structure for doing and sharing data science work.
  https://drivendata.github.io/cookiecutter-data-science/

https://realpython.com/documenting-python-code/

How do we handle change?

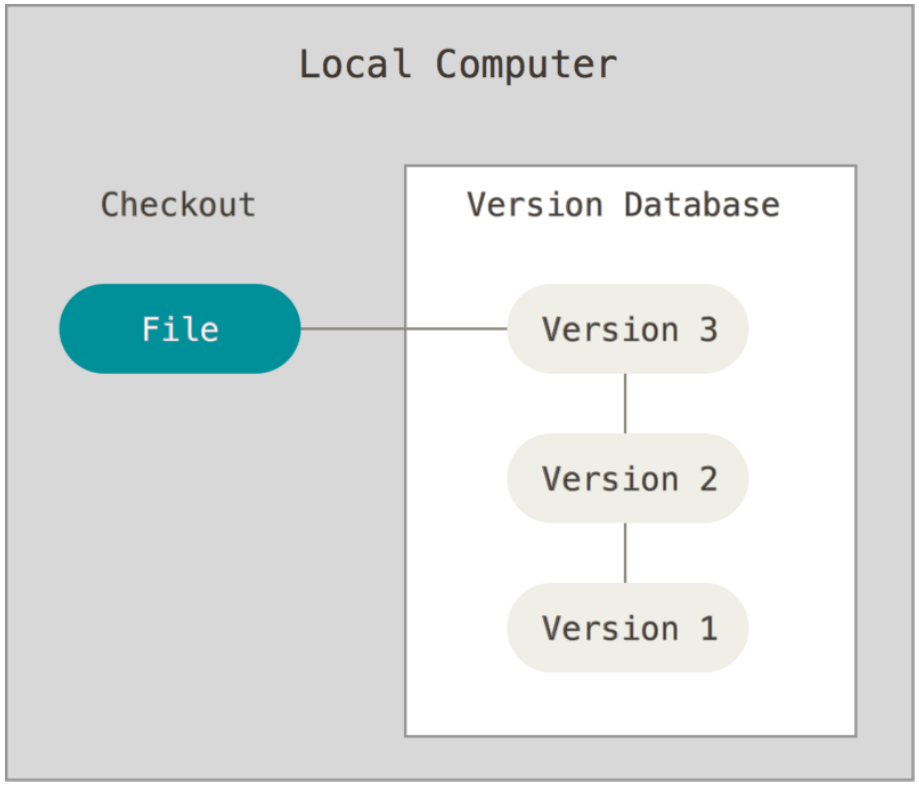
(while preserving our sanity!)

Everyone does version control…

- Most people just do it poorly.
- Keeping all the versions as different filenames is a mess.
- How do you know what's current?
- How do you know what changed?
- How do you collaborate?

Name	Date modified	Type
1.0_article_layerEstimation_BOE.pdf	14/06/2019 5:26 PM	Adobe Acrobat D...
1.0_article_layerEstimation_BOE_AT.pdf	14/06/2019 5:25 PM	Adobe Acrobat D...
1.0_layerEstimation_JBP pn.pdf	10/05/2019 9:54 PM	Adobe Acrobat D...
1.0_layerEstimation_JBP.pdf	9/05/2019 3:21 PM	Adobe Acrobat D...
1.0_LayerEstimation_JBP_CoverLetter.docx	10/05/2019 9:54 PM	Microsoft Word D...
1.0_layerPenetration.pdf	3/05/2019 11:47 AM	Adobe Acrobat D...
1.0_layerPenetration_AT.pdf	6/05/2019 1:38 PM	Adobe Acrobat D...
1.0_layerPenetration_AT_BPR.pdf	6/05/2019 2:00 PM	Adobe Acrobat D...
2.0_layerEstimation_JBP.pdf	14/06/2019 5:26 PM	Adobe Acrobat D...
2.0_layerPenetration.pdf	9/05/2019 1:20 PM	Adobe Acrobat D...
2.0_layerPenetration_AT.pdf	9/05/2019 1:19 PM	Adobe Acrobat D...

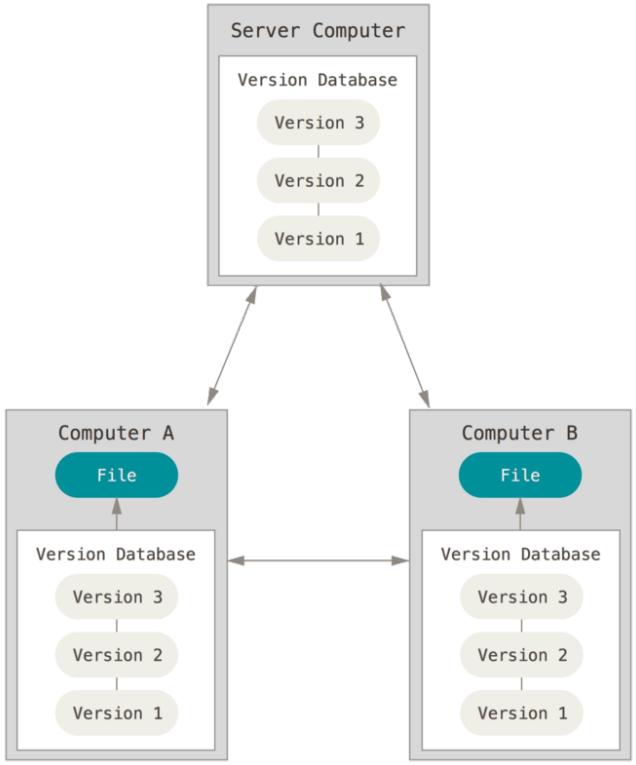
Version control tools work better.



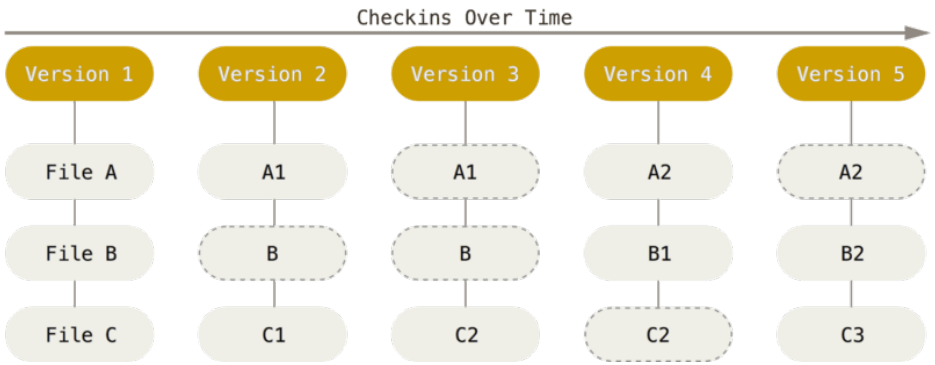
- Automatically track differences between versions
- Only view the current version
- Easily go back to old versions if needed
- We will be using git

Distributed version control works everywhere.

- Someone else's computer ("the cloud") holds the main database
- Other computers also have databases
- There are mechanisms to keep all the databases in sync
- Git is the most popular distributed version control system



Your repository tracks changes.




```
git add .

git commit -m "This is a commit message"
```

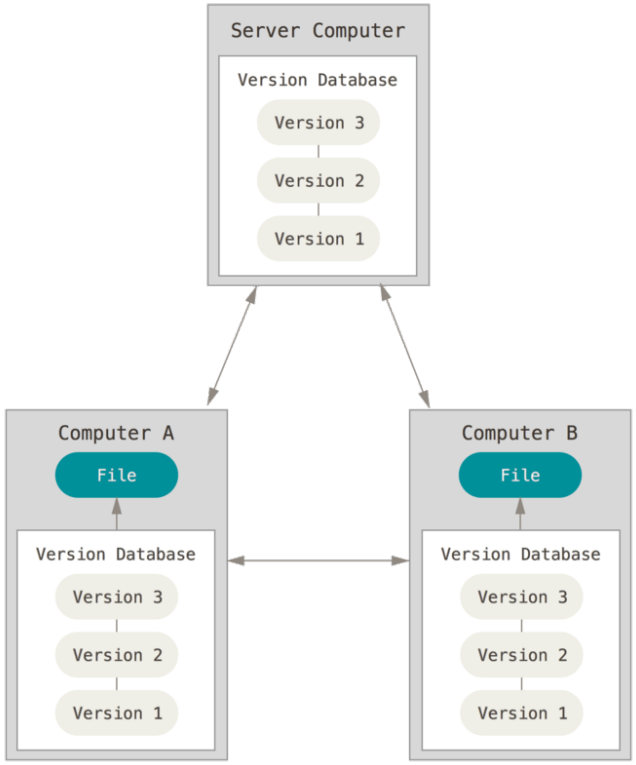
- New versions of files saved and tracked; old versions reused until changed
- Git add tells the system which files should be tracked
  - Not yet saved!
- Git commit saves a new version of the project
- Commit messages help you know what changed

What if you need to go back?

- Each commit has a unique identifier
  - You see it when you commit
  - You can look it up
- You can just look at an old version
  - `git checkout <identifier>`
- You can switch back to an old version
  - `git revert <identifier>`

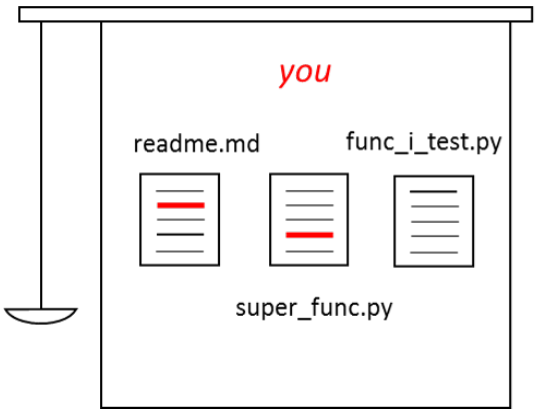
Your repository can be everywhere.

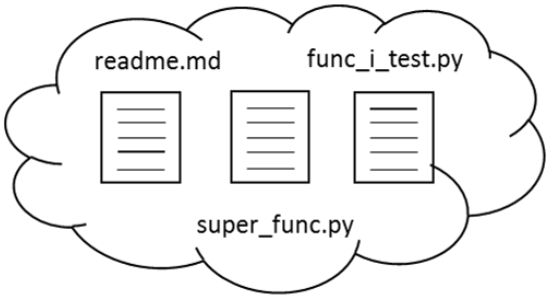
- You can download the whole repository to a new computer
  - `git clone <repo name>`
- You can save your commits to the cloud
  - `git push`
- You can load commits from elsewhere to your computer
  - `git pull`

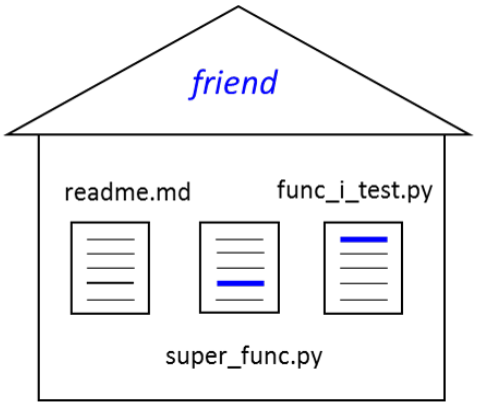


You can collaborate on your repository.

- You can work in multiple places at the same time
- You can give other people access to your repository
- You can let the whole world contribute!

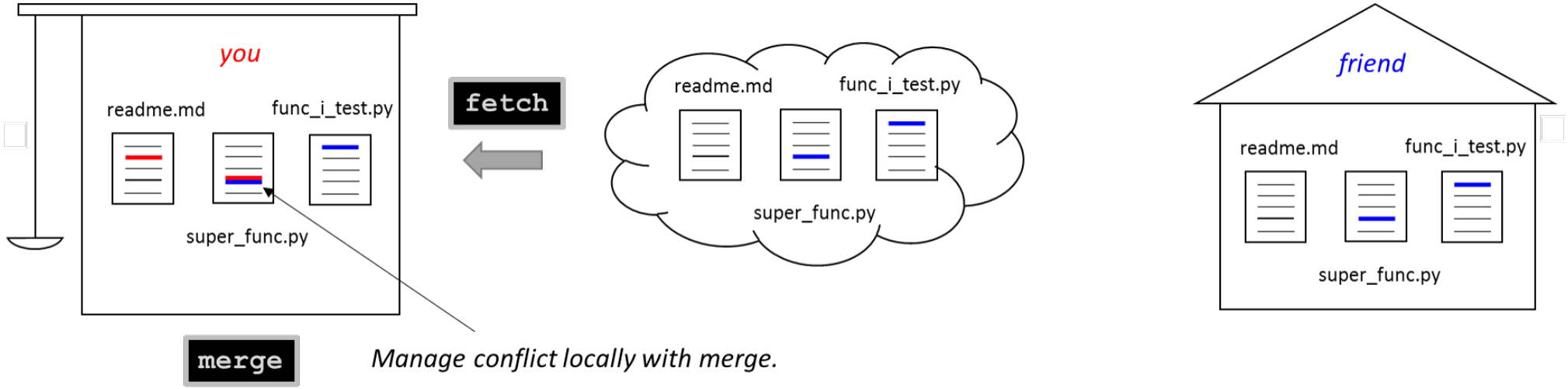






What if there's a conflict?

- Your push will fail
- You can download the conflicting changes using `git fetch`
- Change your file to handle the conflict, then use `git merge`



What if you want to try something new?

- You can make your own copy ("fork") of someone else's repository
- You can create branches to organize a set of commits you're not sure about yet
- You can merge these branches into the main "trunk" when you're happy with them
- Branches can get complicated, so they're not required for this course.

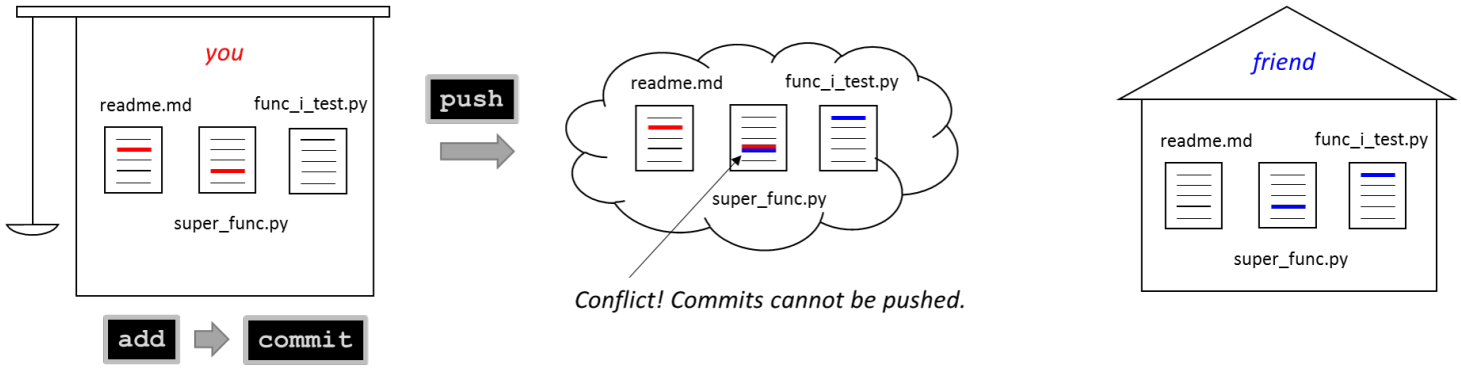


Image References

Slide 1: *Git Commit*, by Randall Munroe, from https://xkcd.com/1296/ (CC BY-NC 2.5)
Slide 3: by Conan at English Wikipedia, from https://commons.wikimedia.org/wiki/File:Linux_kernel_map.png (CC BY 3.0)
Slide 4: by Mr. MacKenty, from https://computersciencewiki.org/index.php/File:Abstract_heart.png (CC BY-NC-SA 4.0)
Slide 17: by Scott Chacon and Ben Straub, from https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (CC BY-NC-SA 3.0)
Slides 18, 21: by Scott Chacon and Ben Straub, from https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (CC BY-NC-SA 3.0)
Slide 19: by Scott Chacon and Ben Straub, from https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F (CC BY-NC-SA 3.0)