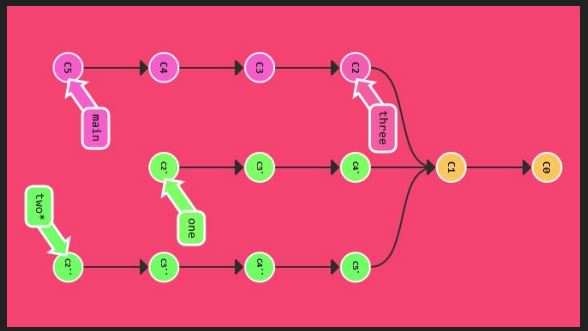
Git and Code Design

CMSE 890-402

What is Git?

- Git is a **version control system**
- Tracks *changes* to files
- *Only* the changes are stored, as a “diff”(erence)

Version control is also a DAG!



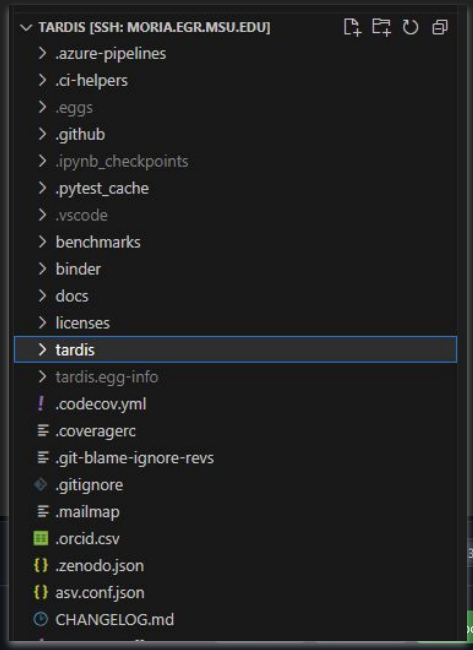
learngitbranching.js.org

20 tardis/energy_input/gamma_ray_transport.py

```
262 262 +
263 263     def main_gamma_ray_loop(
264 264         num_decays,
265     -       model,
    265 +       model_state,
266 266         plasma,
267 267         time_steps=10,
268 268         time_end=80.0,
        @@ -281,7 +281,7 @@ def main_gamma_ray_loop(
281 281         ----------
282 282         num_decays : int
283 283             Number of decays requested
284     -       model : tardis.Radial1DModel
    284 +       model_state : tardis.model.ModelState
285 285             The tardis model to calculate gamma ray propagation through
286 286         plasma : tardis.plasma.BasePlasma
287 287             The tardis plasma with calculated atomic number density
```

Repository

- Location for version controlled files
- Can be **local** or **remote**
  - Local: on your computer
  - Remote: on a server such as GitHub
- Contains:
  - .git (file history, repository configuration)
  - Additional configuration files
  - Tracked files
  - Untracked files (local only)



Commit

- A set of changes to files in the repository
- Contains information about
  - The author
  - What has changed
  - Message from the author

```
commit a6e07376f5a5b6d3d9b983f7003b0a78de7c52f2
Author: Andrew Fullard <andrewgfullard@gmail.com>
Date:   Mon Jul 17 14:02:14 2023 -0400

    Added init to radiation_field

    Also small docstring fix for Composition

diff --git a/tardis/model/base.py b/tardis/model/base.py
index c6b9074ac..b91ab1337 100644
--- a/tardis/model/base.py
+++ b/tardis/model/base.py
@@ -33,7 +33,7 @@ class Composition:
     ----------
     density : astropy.units.quantity.Quantity
         An array of densities for each shell.
-    isotopic_mass_fraction : pd.DataFrame
+    elemental_mass_fraction : pd.DataFrame
```

Push

- Moves changes from your **local** repository to the **remote**
- If the **remote** has changed you can:
  - **Merge** (easy but messy)
  - **Rebase** (difficult but cleaner)

Set up SSH keys

https://docs.github.com/en/authentication/connecting-to-github-with-ssh/about-ssh

Push your DFD image(s) to GitHub!

https://classroom.github.com/a/92rapZrr

GitHub classroom

git clone <path to repository>

git status

git add .

git commit -m “my changes”

git push

VS Code interface for version control

Andrew open up VSCode and demo it

Code Design Principles

- Why should we follow design principles when writing software?

> *“Code is read more often than it is written”* - apocryphal (but also, Guido van Rossum, co-creator of Python)

Common design principles

- **Simplicity**: keep solutions as simple as possible
- **Modularity**: break systems down into components
- **Separation of concerns**: focus modules on each system component
- **Don’t Repeat Yourself**: re-use existing code
- **Single Responsibility Principle**: each component (module, function) should have one purpose

Pseudocode

- Simple way to describe algorithms
- Easily translated to many scripting languages
- **Informal!**
- Can use explicit start-end markers or indentation for blocks
  - Your call depending on what you prefer
- Should be **human readable**
  - Minimise use of named functions

Functions

- Perform a single task (SRP)
- Take zero - n inputs
- Produce zero - n outputs (but ideally at least 1 output)
- Named to reflect their purpose
    - Following some style guide

```
my_function(argument)

    output = do something to argument

    return output
```

Modules

- Consist of multiple functions
- Can be formalized as a **class** in some languages
  - In which case, functions are called **methods**

```
my_function()


my_other_function()


a_third_function()
```

```
my_class()

    a_method()


    another_method()
```

Packages

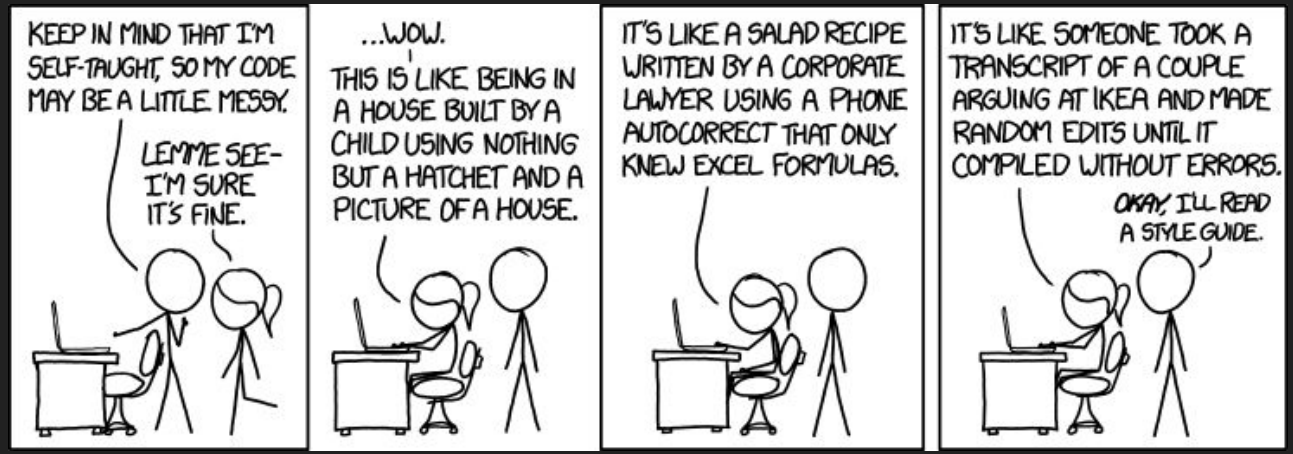
- Consist of one or multiple modules
- Often distributed as a single item
- Often controlled by a *package manager*

Package repository

- Hosts packages on the web
- Used by package managers to access new packages
- Maintained by businesses and nonprofits
  - Trust is important!
- Python has PyPI, conda-forge, anaconda cloud and more
- R has CRAN
- Java has Maven

Style

- Style is a *choice*
  - But it should be *consistent* across code to make it readable!
- A style should enhance readability
- Styles are language-dependent
- “linters” can be used to easily enforce a style



xkcd 1513

Activity

- Install the Python and “autopep8” extensions in VSCode
- Set up the linter to enforce PEP8 style (“format on save”)
- Open the script “bad_example.py” from the classroom repository
  - https://classroom.github.com/a/l6HGFtt6
- Fix the bad styling determined by the linter
- Commit the result
  - Don’t forget to push

- Remember: the hardest thing about code style is agreeing to a guide in a team

Homework: Convert your DFDs to pseudocode

- What should be a package, module or function?
- What should each function take as inputs?
- What outputs should the functions produce?

- Write the pseudocode in a standard text file
- If functions can be reused, reuse them!
- Commit and push your changes to GitHub at the DFD classroom repository