Git and Code Design

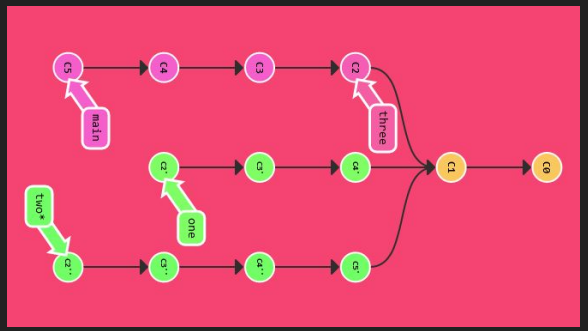
CMSE 890-402

Git basics

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!



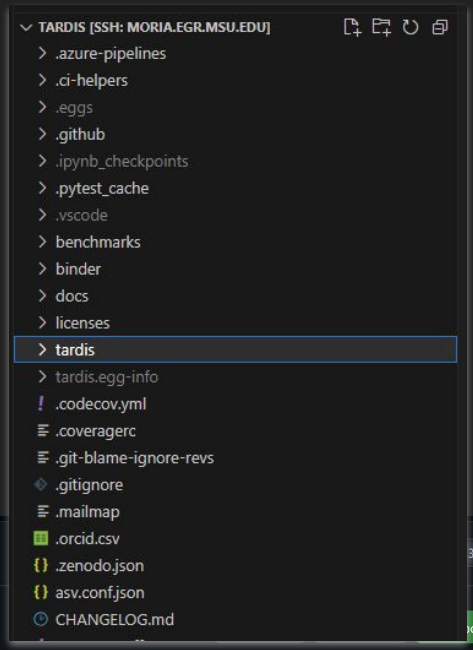
learngitbranching.js.org

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
- Defined by a 40-character “hash”
  - SHA-1 algorithm for reference

```
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)

Recommended collaborative workflow (one branch)

- Pull from the remote
- Work on your changes
- Commit your changes
- Pull from the remote again
  - Resolve any merge issues
- Push to the remote

Set up SSH keys

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

Andrew will walk through this.

VS Code interface for version control

Andrew open up VSCode and demo it

GitHub classroom

https://classroom.github.com/a/HGLRlRF5

git clone <path to repository>

git status

git add .

git commit -m “my changes”

git push

Push your DFD image(s) to GitHub!

https://classroom.github.com/a/HGLRIRF5

Code design and organization

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
```

Simple pseudocode example

```
function average_data(data_array) is
    sum = 0
    number = 0
    for each data in data_array do
        sum = sum + data
        number = number + 1
    average = sum / number
    return average
```

Define the function name and inputs

Define variables

Do a loop

Return the variable for use

Pseudocode function examples

```
def fizzbuzz():
  for i in range(1,101):
    print_number = true
    if i is divisible by 3:
      print "Fizz"
      print_number = false
    if i is divisible by 5:
      print "Buzz"
      print_number = false
    if print_number: print i
    print a newline
```

https://en.wikipedia.org/wiki/Pseudocode

```
function binary_search(A, n, T) is
    L := 0
    R := n − 1
    while L ≤ R do
        m := floor((L + R) / 2)
        if A[m] < T then
            L := m + 1
        else if A[m] > T then
            R := m − 1
        else:
            return m
    return unsuccessful
```

https://en.wikipedia.org/wiki/Binary_search

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

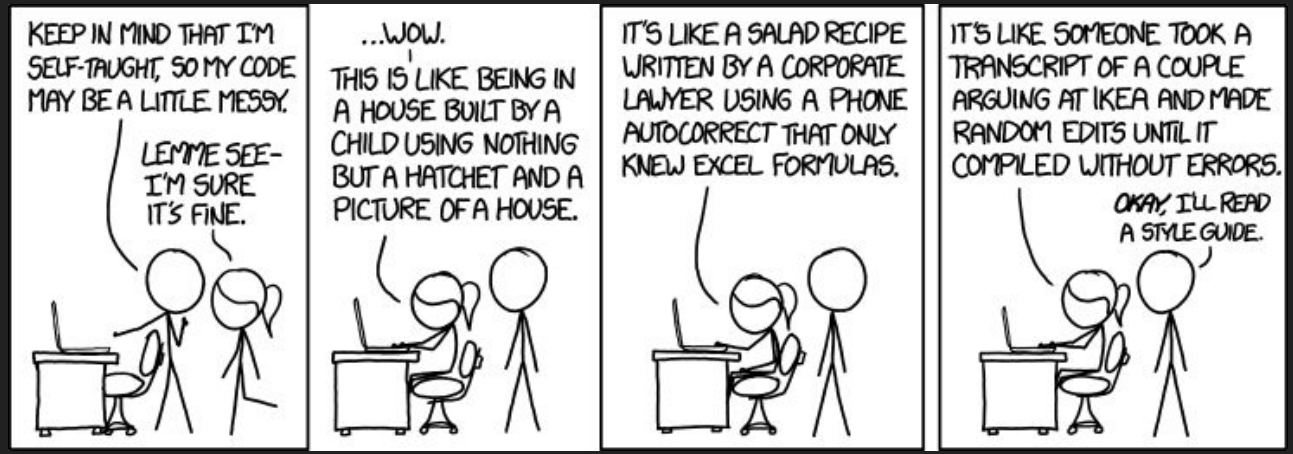
- Consist of one or multiple modules
- Often distributed as a single item
- Often controlled by a *package manager*

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 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

Python style example: PEP8

- https://peps.python.org/pep-0008/
- Use 4 spaces per indentation level.
- Limit all lines to a maximum of 79 characters.
- Surround top-level function and class definitions with two blank lines.
- Imports should usually be on separate lines.
- Comments that contradict the code are worse than no comments. Always make a priority of keeping the comments up-to-date when the code changes!
- Use inline comments sparingly.

R style example: tidyverse

- https://style.tidyverse.org/
- File names should be meaningful and end in .R
- Variable and function names should use only lowercase letters, numbers, and _.
- Strive to limit your code to 80 characters per line.
- Curly braces, {}, define the most important hierarchy of R code. To make this hierarchy easy to see:
  - { should be the last character on the line. Related code (e.g., an if clause, a function declaration, a trailing comma, …) must be on the same line as the opening brace.
  - The contents should be indented by two spaces.
  - } should be the first character on the line.
- Use <-, not =, for assignment.
- Each line of a comment should begin with the comment symbol and a single space: #

Activity

- Install the Python and “Ruff” extensions in VSCode
- Open the script “bad_example.py” from the classroom repository
  - https://classroom.github.com/a/Wdmj7EF7
- Fix the bad styling determined by the linter
- Rename or comment on bad practices in the code
- Commit the result
  - **Don’t forget to push**

- Remember: the hardest thing about code style is agreeing to a guide in a team
- You can set up the linter to enforce style automatically where possible (“format on save”)!

Homework: Convert your DFDs to pseudocode

- What processes should be a package, module or function?
  - Group similar processes!
  - If functions can be reused, reuse them!
- What should each process take as inputs?
- What outputs should the processes produce?
  - Where will they be stored?
- Remember: this is a workflow, not a single task
- Your functions can simply be calling external software, but make that clear!

- Write the pseudocode in a standard text file
- **Commit and push** your changes to GitHub **at the DFD classroom repository**

Pre-class 3

Install pytest into a Python environment. https://docs.pytest.org/en/7.4.x/index.html. If you are new to Python, I strongly recommend using miniforge to set up a Python environment. Documentation is here https://conda-forge.org/download/ for all operating systems. To install pytest in a new environment, run

```
conda env create --name preclass3 pytest
```

in a terminal after you have installed miniforge. Run

```
conda activate preclass3
```

to use the environment in your terminal. In VS Code, you should be able to select the environment using the Python extension, see https://code.visualstudio.com/docs/python/environments#_working-with-python-interpreters

Once you have pytest installed, complete as much as you can of the "Get Started" section of the pytest documentation at https://docs.pytest.org/en/7.4.x/getting-started.html

Upload screenshots of your results on D2L.