

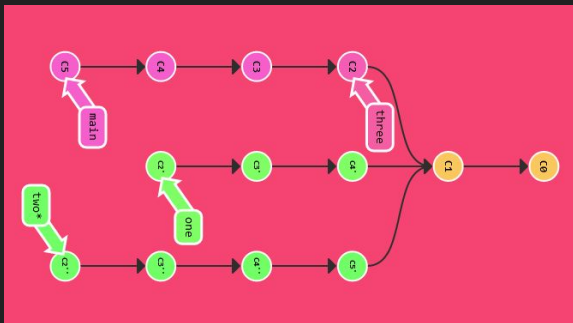
Git and Code Design

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!



learnitbranching.js.org

20 tardis/energy_input/gamma_ray_transport.py

```

262 262 +
263 263     def main_gamma_ray_loop(
264 264         num_decays,
265 265         - model,
266 265 + model_state,
267 266         plasma,
268 267         time_steps=10,
269 268         time_end=80.0,
270 269     ):
271 270         @@ -281,7 +281,7 @@ def main_gamma_ray_loop(
272 271         -----
273 272         num_decays : int
274 273             Number of decays requested
275 274         - model : tardis.Radial1DModel
276 274 + model_state : tardis.model.ModelState
277 275             The tardis model to calculate gamma ray propagation through
278 276         plasma : tardis.plasma.BasePlasma
279 277             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)

The image shows a terminal window with a file explorer view of a repository. The root directory is 'TARDIS [SSH: MORIA.EGR.MSU.EDU]'. The files and directories listed are: .azure-pipelines, .ci-helpers, .eggs, .github, .ipynb_checkpoints, .pytest_cache, .vscode, benchmarks, binder, docs, licenses, and **tardis** (highlighted). Below 'tardis' are sub-items: tardis.egg-info, .codecov.yml, .coveragerc, .git-blame-ignore-revs, .gitignore, .mailmap, .orcid.csv, .zenodo.json, asv.conf.json, and CHANGELOG.md.

Below the terminal window is a screenshot of the GitHub repository page for 'tardis'. The repository is public and has 74 branches and 132 tags. The commit history shows a post-release on 2023.07.23 (#2369) with 644,048 commits and 4,417 commits. The commit history table is as follows:

Commit	Message	Time
tardis	integrate new packetsource (#2367)	5 days ago
licenses	APE 17 migration (#1674)	2 years ago
docs	Post-release 2023.07.23 (#2369)	3 days ago
binder	Enabling Repository on Binder (#1678)	2 years ago
benchmarks	[WIP] Basic Benchmarks Using ASV (#2260)	last month
.github/workflows	Benchmarks Push To Repo Bug Fix (#2330)	last month
.ci-helpers	Fix for release dates (#2243)	4 months ago
.azure-pipelines	Port update reference data pipeline to GitHub Actions (#2249)	3 months ago

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/858KPncS>

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
```

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*

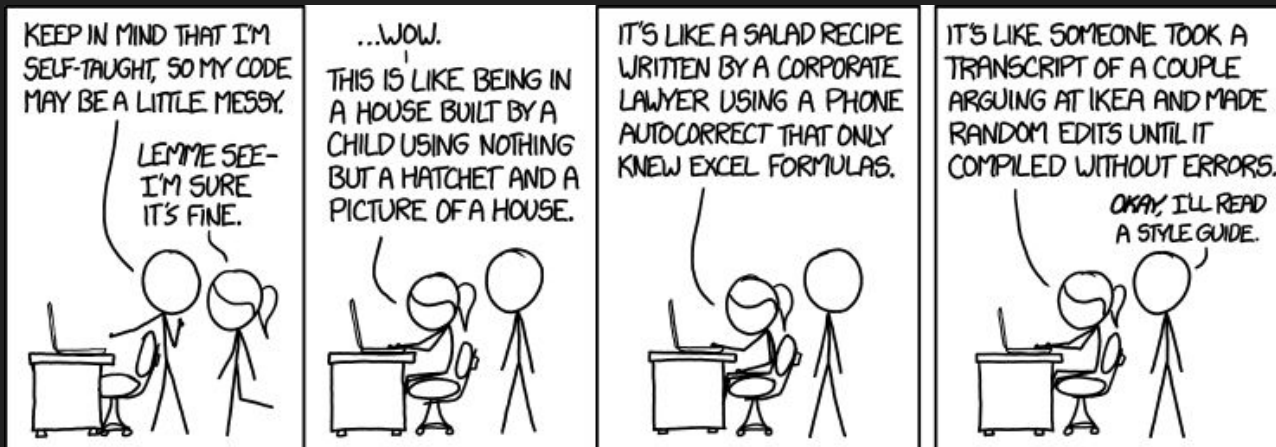


Package repository

- 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



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/4U6bFqza>
 - 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
 - You can set up the linter to enforce style automatically where possible (“format on save”)!

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