Welcome to
CHE 384T: Computational Methods in Materials Science

Introduction to Computational Materials Science

Programming Day 3

The University of Texas at Austin
McKetta Department
of Chemical Engineering
Cockrell School of Engineering

Programming Day Agenda

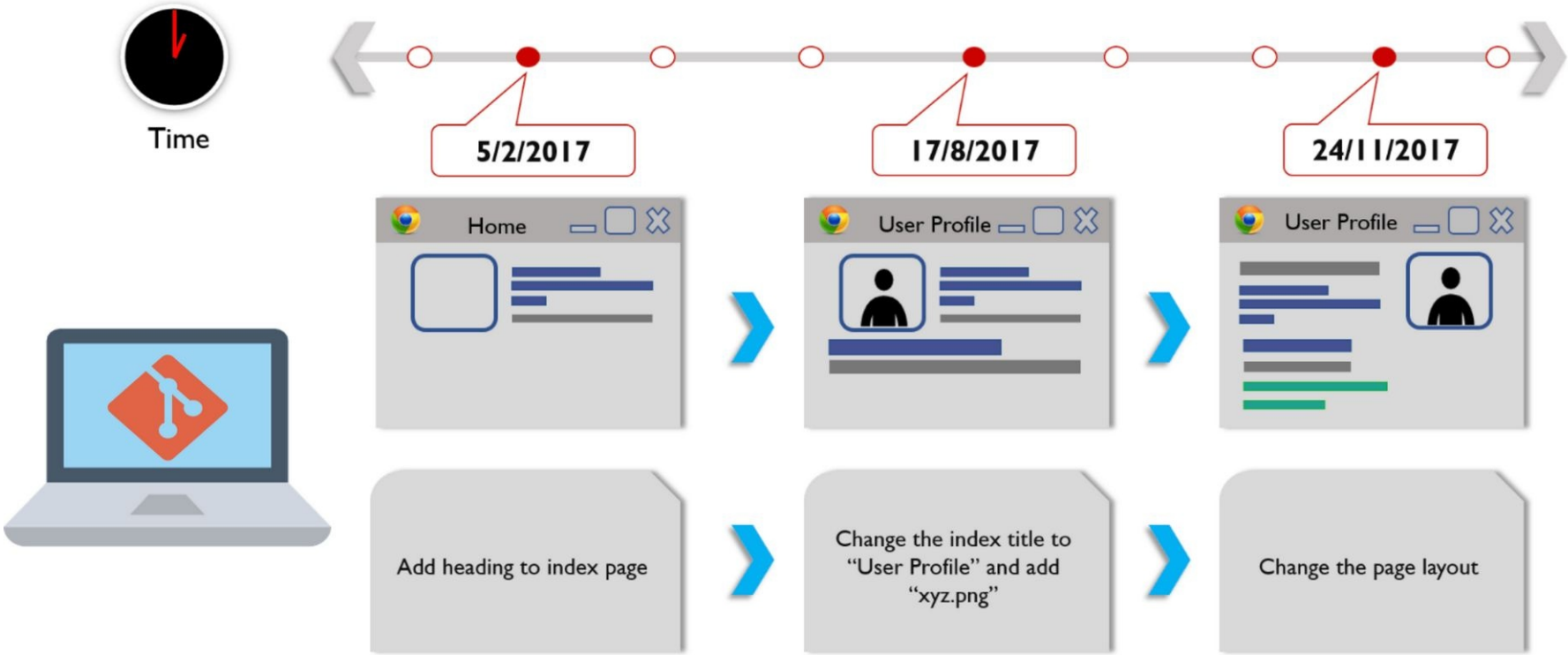
Setting up your environment:

- Git and Github: version control (and backups)

Python helpers:

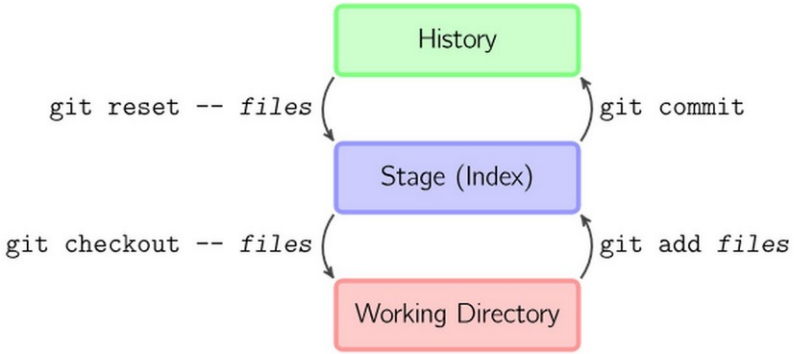
- Formatting Python conventions: PEP
- Documentation generation: sphinx, doxygen
- Some ways to optimize Python code

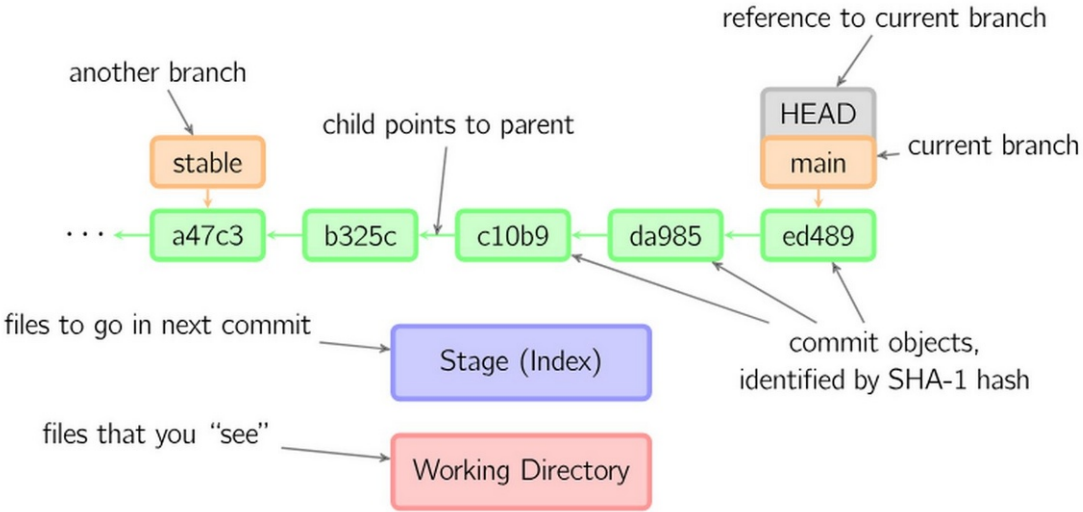
Git: version control



https://www.edureka.co/blog/git-vs-github/

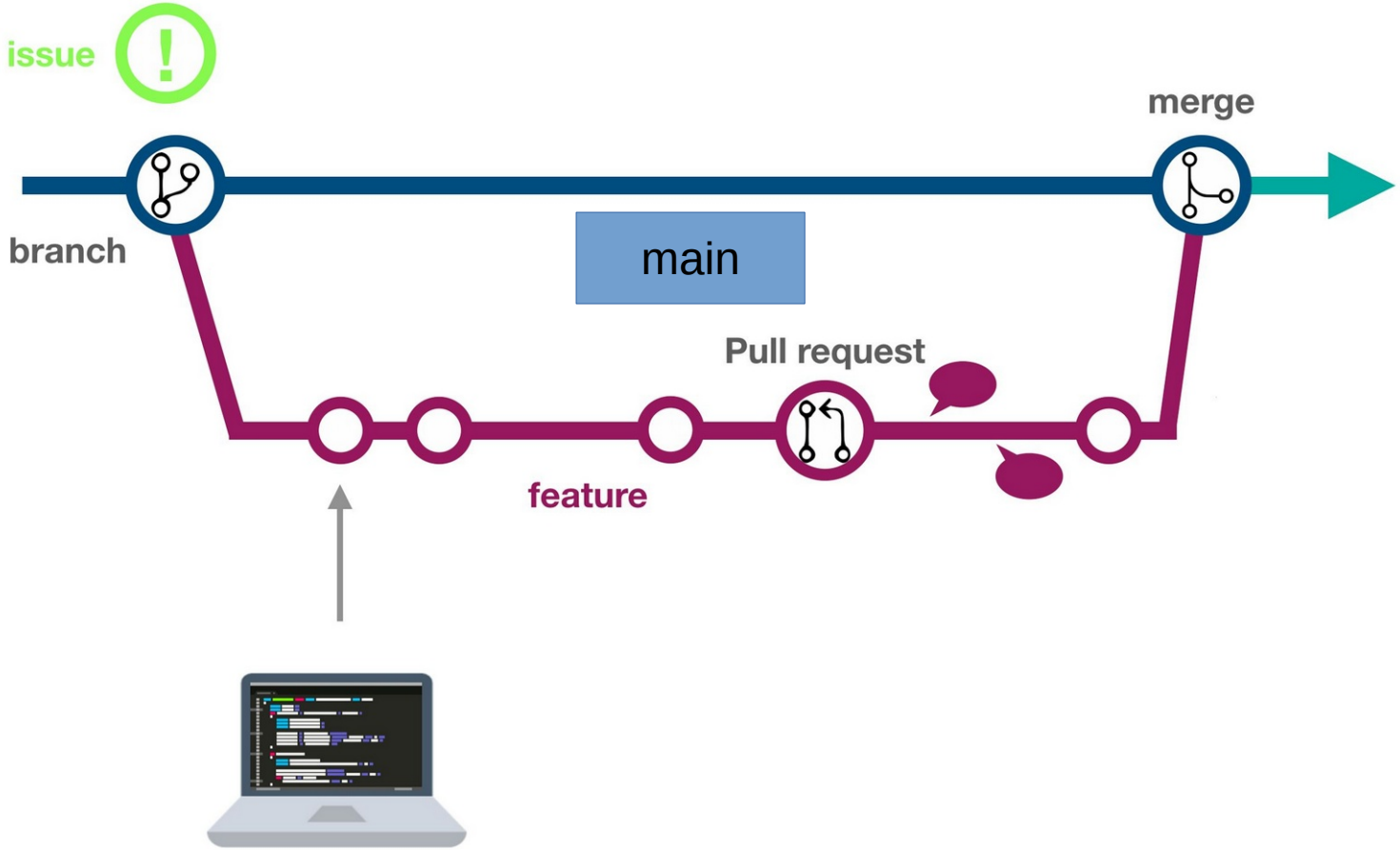
Git: version control



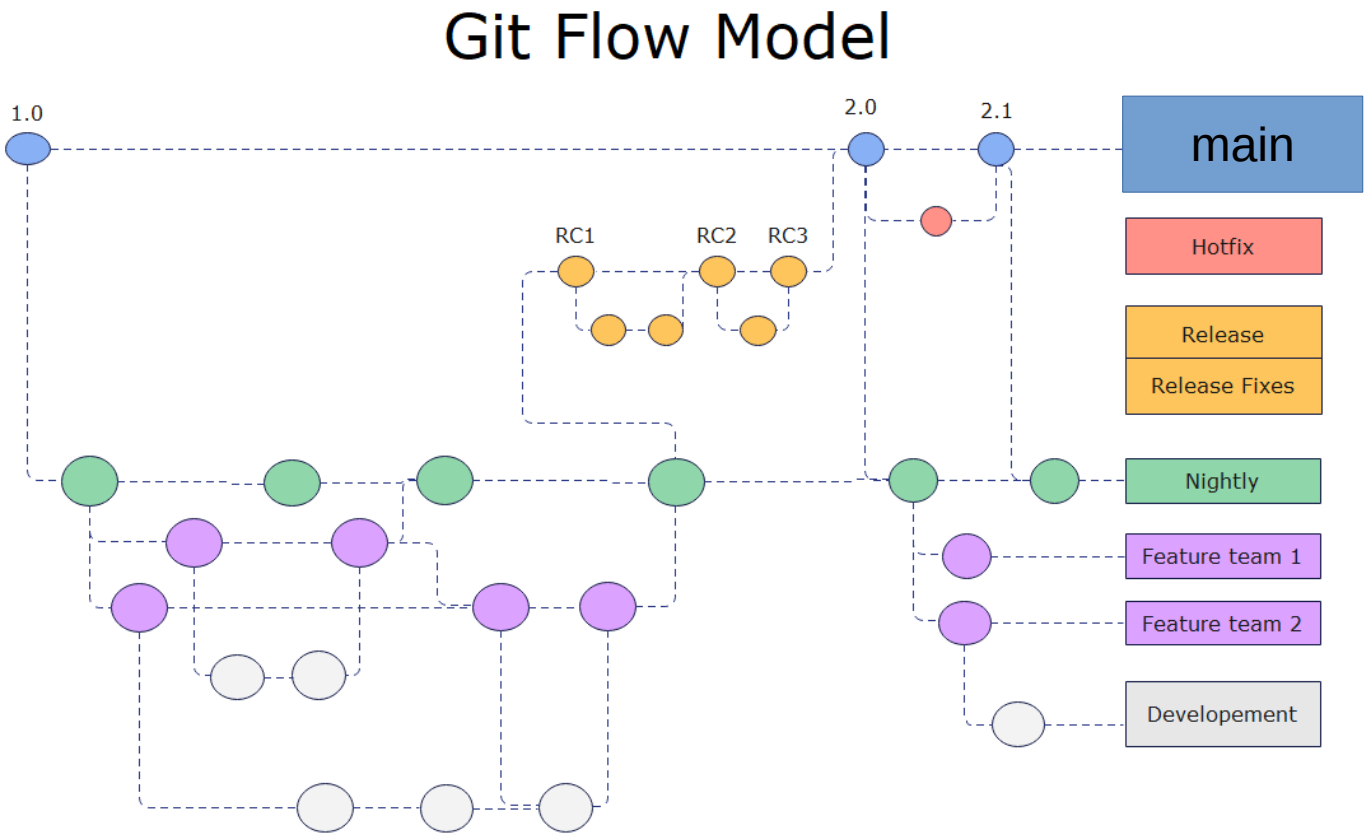


https://marklodato.github.io/visual-git-guide/index-en.html

Git: version control



Git: version control

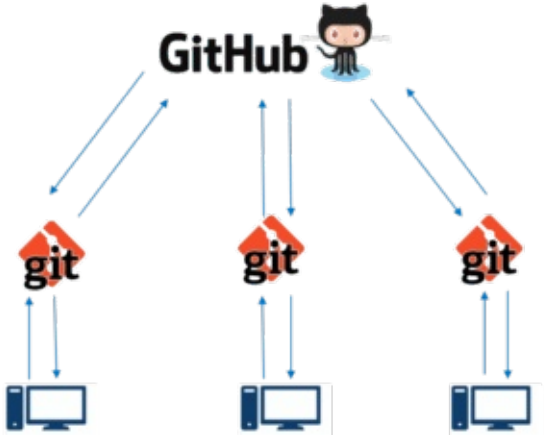


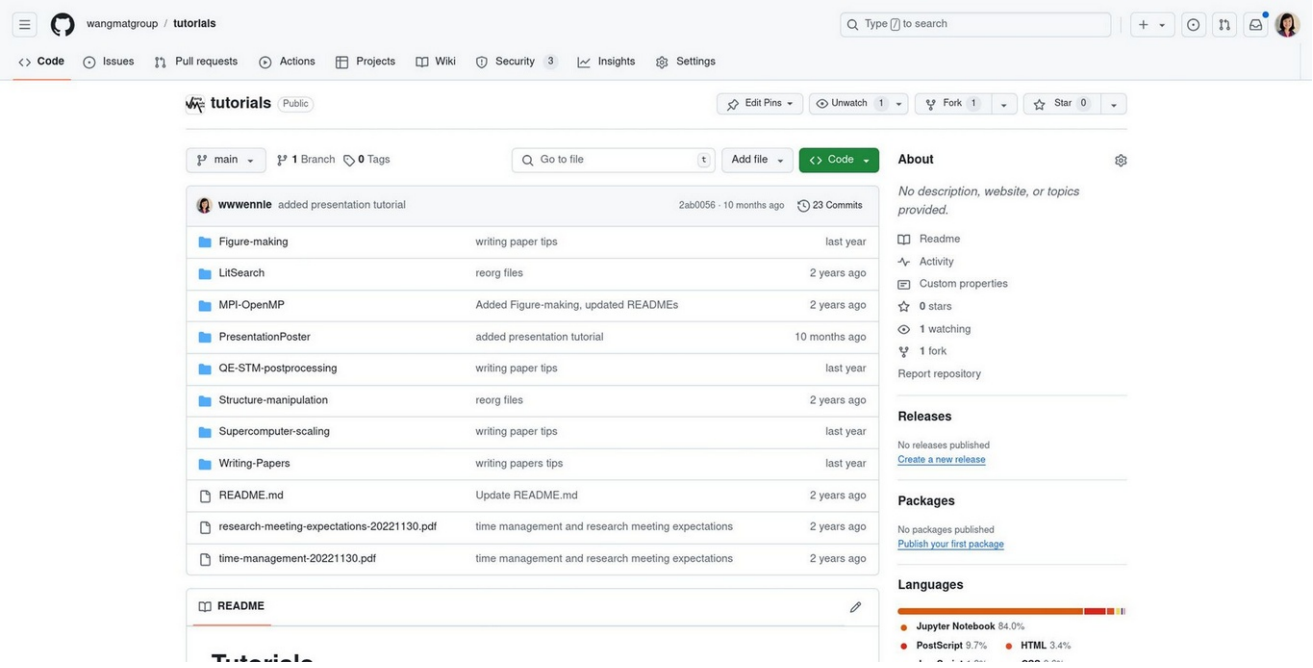
Git: version control

Visualize branches in the terminal: git lg / git lg1

```
* 525a852 - (4 years, 4 months ago) notification test - German Laullon (origin/test_on_10.8)
* 58b5a57 - (4 years, 4 months ago) morden Objective-c - German Laullon
* aef1d39 - (4 years, 4 months ago) first build. - German Laullon
*   bff6661 - (4 years, 4 months ago) Merge pull request #169 from taybin/patch-1 - German Laullon (HEAD -> master, ori
|\
| * ef61b97 - (4 years, 9 months ago) Update Rakefile to work with ruby-1.9.2. - Taybin Rutkin
* |   3b06317 - (4 years, 4 months ago) Merge pull request #175 from barrywardell/master - German Laullon
|\ \
| * | 907e967 - (4 years, 10 months ago) Fix bug where submodules were incorrectly grouped when the first part of their
* | |   dd1b324 - (4 years, 4 months ago) Merge pull request #195 from jphalip/master - German Laullon
|\ \ \
| * | | 8ebb58c - (4 years, 5 months ago) Added "Copy Reference to Clipboard" context menu item in sidebar. - Julien Ph
| * | | 238a97a - (4 years, 6 months ago) Fixed a tiny typo. - Julien Phalip
| * | | 3386fc7 - (4 years, 6 months ago) Make sure the commit view gets refreshed when 'Stage' gets selected and the a
| * | | 7fafdb8 - (4 years, 6 months ago) Tweaked capitalization of words in contextual menu items. - Julien Phalip
| * | | 5958f5b - (4 years, 6 months ago) Don't display all the files selected for deletion to avoid the confirmation s
| * | | 3575e87 - (4 years, 6 months ago) Prevent large files from getting loaded in the commit view to prevent the app
| * | | 748621c - (4 years, 6 months ago) Made the "(Un)stage lines" functionality in the commit view work only if the
| * | | d249fd6 - (4 years, 6 months ago) When a branch gets selected in the sidebar, make sure its corresponding commi
| * | | 839c9b6 - (4 years, 6 months ago) Made the diff tables adapt nicely to the window's size. Fixes #50. - Julien P
* | | |   8badd8a - (4 years, 4 months ago) Merge pull request #196 from Kyriakis/master - German Laullon
|\ \ \ \
| * | | | af773ba - (4 years, 6 months ago) No check for refs on remotes while deleting them - Robert Kyriakis
| |/ / /
* | | |   47a8a1a - (4 years, 4 months ago) Merge pull request #197 from Uncommon/arcfix - German Laullon
|\ \ \ \
| * | | | d1a8ba9 - (4 years, 6 months ago) fix for ARC errors and other warnings - David Catmull
| |/ / /
* | | | b94240c - (4 years, 4 months ago) [y so slow] :D - German Laullon
|/ / /
* | |   8ce13ad - (4 years, 6 months ago) Merge pull request #194 from jphalip/master - German Laullon
|\ \ \
| * | | 3e045a4 - (4 years, 6 months ago) Ensure that the previously selected commit remains selected after refreshing.
|/ / /
* | |   aae5e7c - (4 years, 6 months ago) Merge pull request #192 from jphalip/master - German Laullon
|\ \ \
| * | | cd2e8de - (4 years, 6 months ago) Made the sign-off button optional and hidden by default, as this is a feature
| * | | 42304da - (4 years, 6 months ago) Display file list at the top of the diff window. - Julien Phalip
| * | | 24d05af - (4 years, 6 months ago) Minor UI improvements to commit count status label. - Julien Phalip
|/ / /
* | |   f38201d - (4 years, 6 months ago) Merge pull request #191 from jphalip/8a9b9629246dc7d97d748f8de414e14a3e019c94
|\ \ \
| * | | 8a9b962 - (4 years, 6 months ago) Normalized the menu items to use capitalized initials and ellipses, and made
| * | | 18d7767 - (4 years, 6 months ago) In the 'Create branch' sheet: use a placeholder in the branch name textfield
* | | |   6f45075 - (4 years, 6 months ago) Merge pull request #190 from mullr/master - German Laullon
```

Github (social coding)





Git and Github: version control

git	GitHub
1. It is a software	1. It is a service
2. It is installed locally on the system	2. It is hosted on Web
3. It is a command line tool	3. It provides a graphical interface
4. It is a tool to manage different versions of edits, made to files in a git repository	4. It is a space to upload a copy of the **Git** repository
5. It provides functionalities like Version Control System Source Code Management	5. It provides functionalities of Git like VCS, Source Code Management as well as adding few of its own features

https://www.edureka.co/blog/git-vs-github/

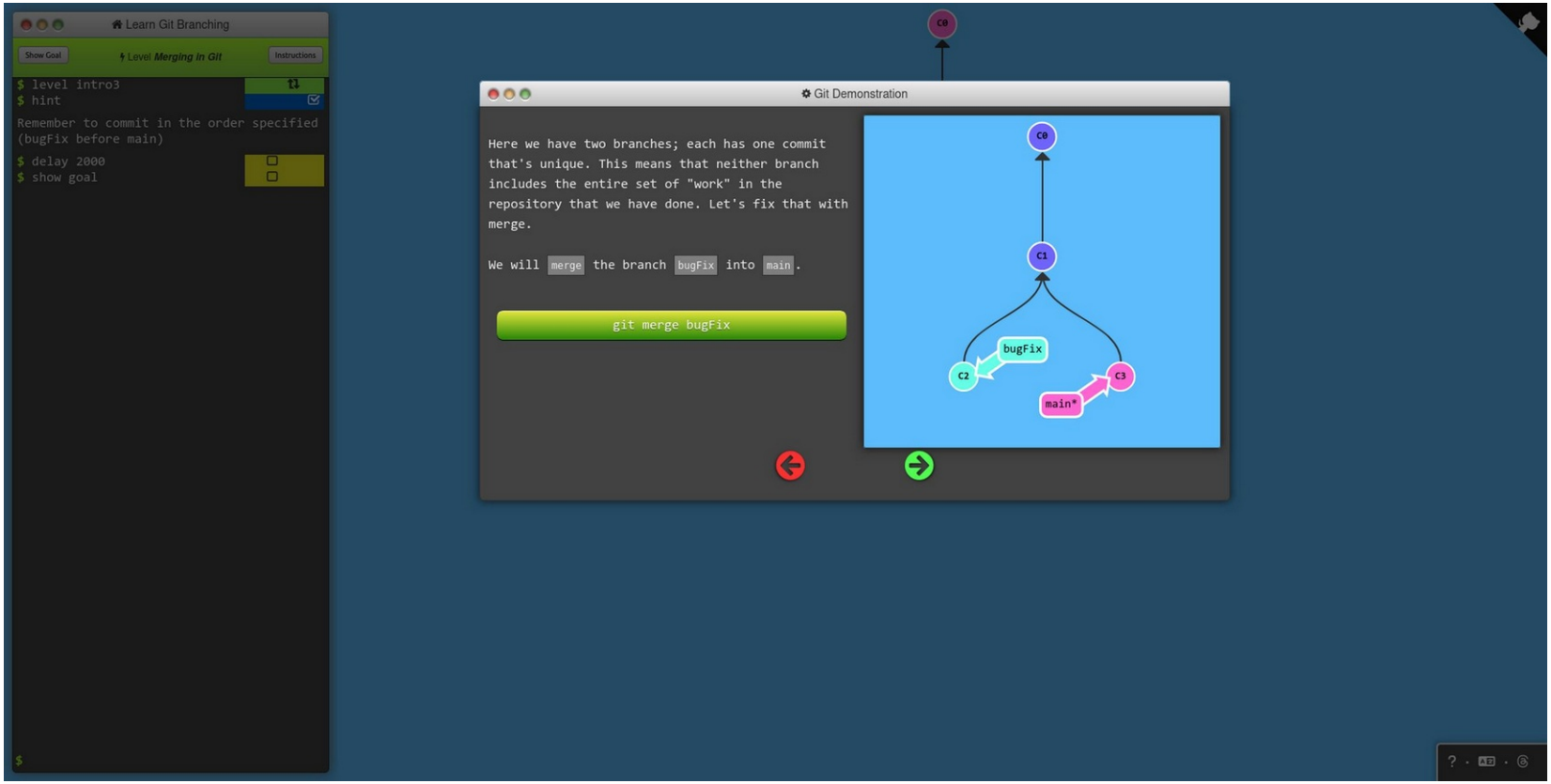
Git and Github: version control

Git

- `git clone` - clone remote repository
- `git pull` - pull most recent version from remote
- `git add` - add local files to be staged for remote
- `git commit` - stage/commit local changes
- `git push` - push local commits to remote
- `git branch` - list all available branches
- `git checkout` - move to new branch
- `git status` - checks which branch you are on and if you have any unsaved changes
- `git log` - shows log of previous commits on current branch
- `git diff` - shows details of changes made

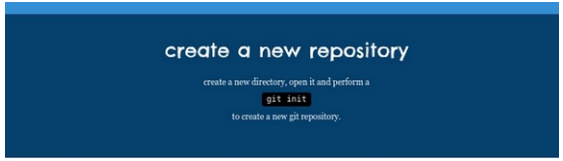
Git and Github: version control

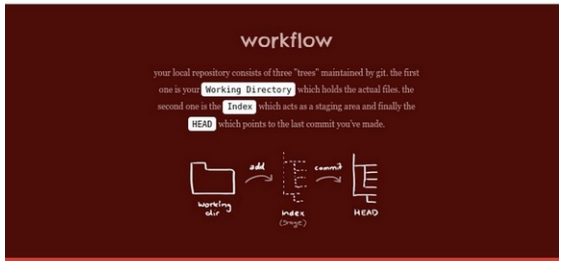
Tutorials on Git: https://learngitbranching.js.org/

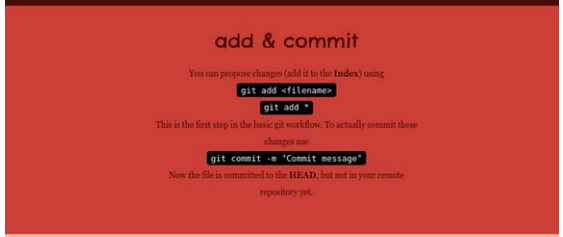


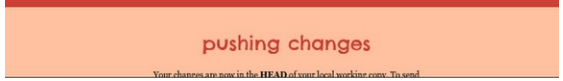
Git and Github: version control

Tutorials on Git: http://rogerdudler.github.io/git-guide/



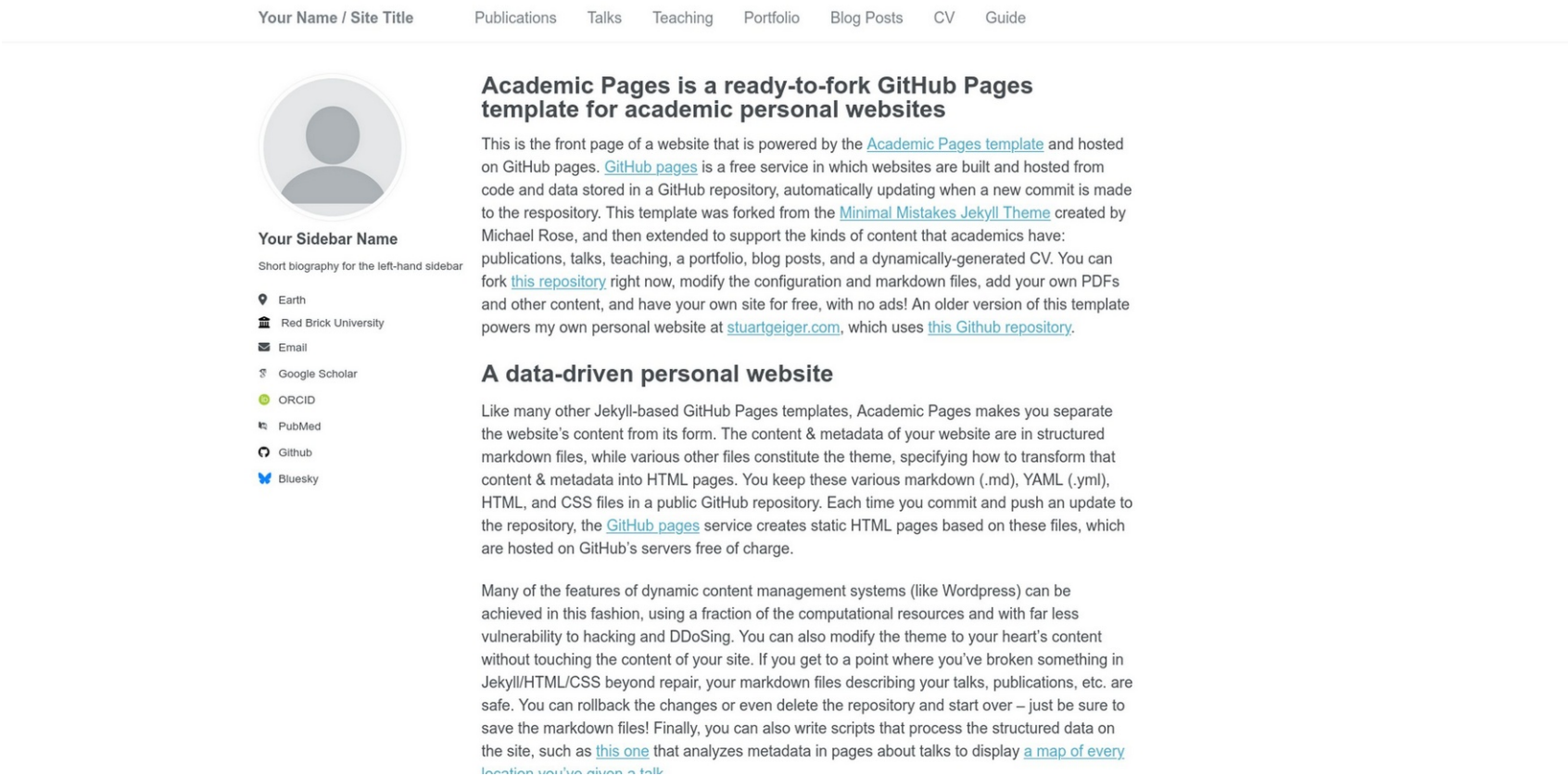






Git and Github: version control

Other things you can do: personal website,
e.g., https://github.com/academicpages/academicpages.github.io



Your Name / Site Title

Publications Talks Teaching Portfolio Blog Posts CV Guide

Your Sidebar Name

Short biography for the left-hand sidebar

Earth
Red Brick University
Email
Google Scholar
ORCID
PubMed
Github
Bluesky

Academic Pages is a ready-to-fork GitHub Pages template for academic personal websites

This is the front page of a website that is powered by the Academic Pages template and hosted on GitHub pages. GitHub pages is a free service in which websites are built and hosted from code and data stored in a GitHub repository, automatically updating when a new commit is made to the respository. This template was forked from the Minimal Mistakes Jekyll Theme created by Michael Rose, and then extended to support the kinds of content that academics have: publications, talks, teaching, a portfolio, blog posts, and a dynamically-generated CV. You can fork this repository right now, modify the configuration and markdown files, add your own PDFs and other content, and have your own site for free, with no ads! An older version of this template powers my own personal website at stuartgeiger.com, which uses this Github repository.

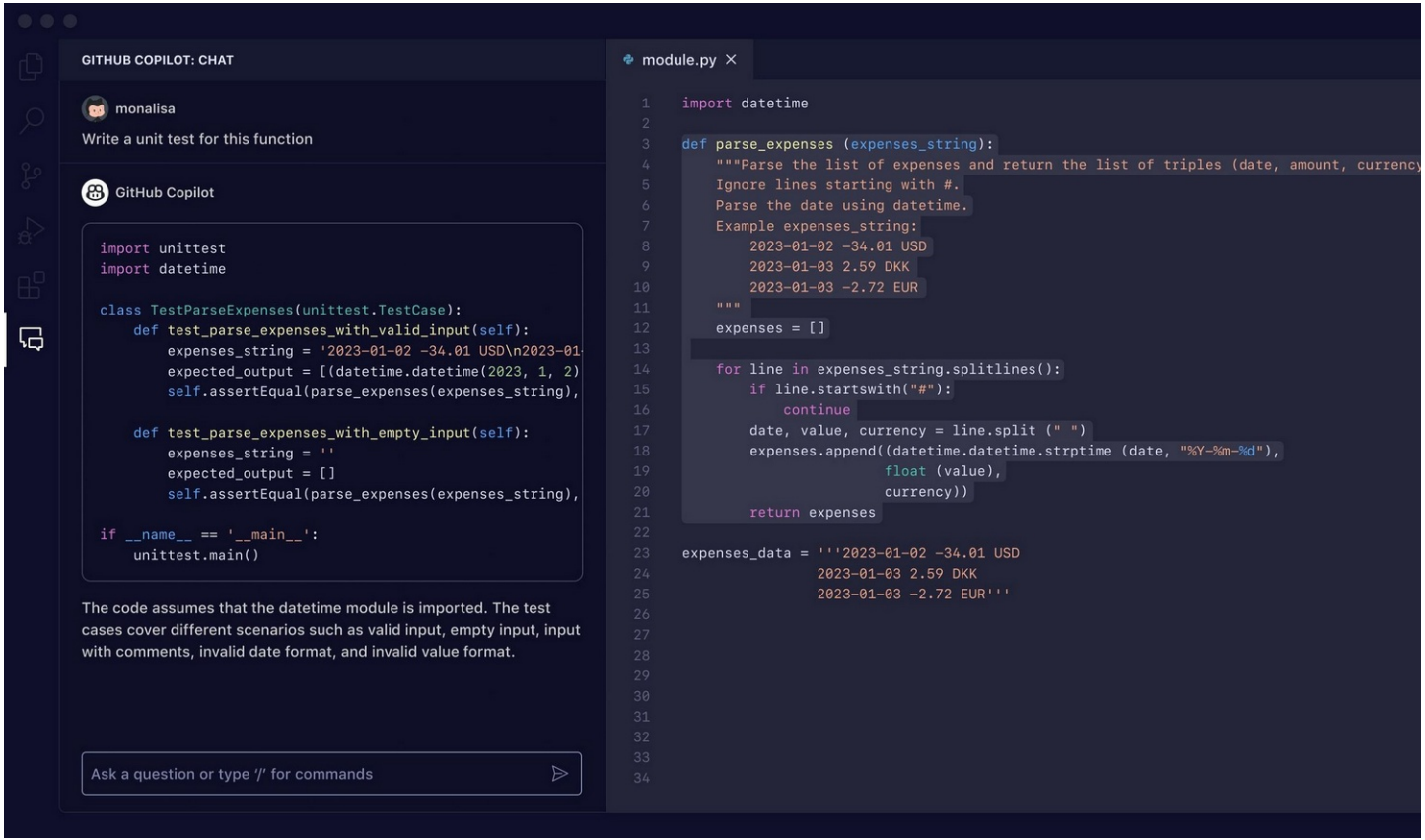
A data-driven personal website

Like many other Jekyll-based GitHub Pages templates, Academic Pages makes you separate the website's content from its form. The content & metadata of your website are in structured markdown files, while various other files constitute the theme, specifying how to transform that content & metadata into HTML pages. You keep these various markdown (.md), YAML (.yml), HTML, and CSS files in a public GitHub repository. Each time you commit and push an update to the repository, the GitHub pages service creates static HTML pages based on these files, which are hosted on GitHub's servers free of charge.

Many of the features of dynamic content management systems (like Wordpress) can be achieved in this fashion, using a fraction of the computational resources and with far less vulnerability to hacking and DDoSing. You can also modify the theme to your heart's content without touching the content of your site. If you get to a point where you've broken something in Jekyll/HTML/CSS beyond repair, your markdown files describing your talks, publications, etc. are safe. You can rollback the changes or even delete the repository and start over – just be sure to save the markdown files! Finally, you can also write scripts that process the structured data on the site, such as this one that analyzes metadata in pages about talks to display a map of every location you've given a talk

Git and Github: version control

Other things you can do: Github Copilot (coding with AI; currently free for students/educators)



Python Enhancement Proposals (PEP)

The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Easter Egg

```
>>> import this
```

Python Enhancement Proposals (PEP)

Guido Van Rossum: Code is read more than it is written

https://peps.python.org/pep-0008/: formatting code

PEP 257.

Whitespace in Expressions and Statements

Pet Peeves

Avoid extraneous whitespace in the following situations:

- Immediately inside parentheses, brackets or braces:

```
# Correct:
spam(ham[1], {eggs: 2})
```

```
# Wrong:
spam( ham[ 1 ], { eggs: 2 } )
```

- Between a trailing comma and a following close parenthesis:

```
# Correct:
foo = (0,)
```

```
# Wrong:
bar = (0, )
```

- Immediately before a comma, semicolon, or colon:

```
# Correct:
```

Python Enhancement Proposals (PEP)

Guido Van Rossum: Code is read more than it is written

https://peps.python.org/pep-0257/ : Doc string formatting

Contents

Page Source (GitHub)

replaces a superclass method and does not call the superclass method, use the verb "extend" to indicate that a subclass method calls the superclass method (in addition to its own behavior).

Do not use the Emacs convention of mentioning the arguments of functions or methods in upper case in running text. Python is case sensitive and the argument names can be used for keyword arguments, so the docstring should document the correct argument names. It is best to list each argument on a separate line. For example:

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```

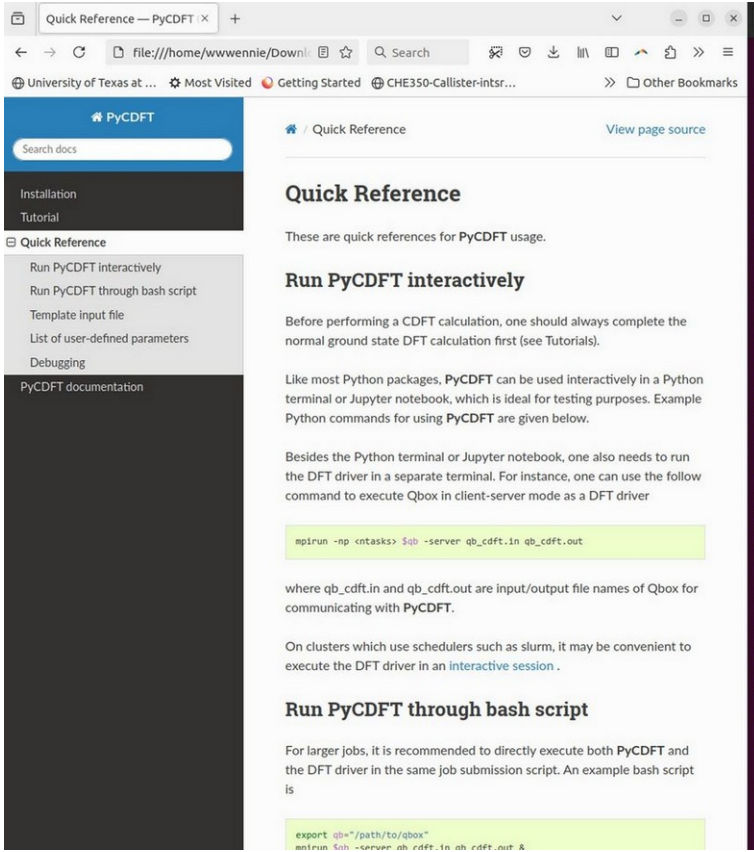
Unless the entire docstring fits on a line, place the closing quotes on a line by themselves. This way, Emacs' `fill-paragraph` command can be used on it.

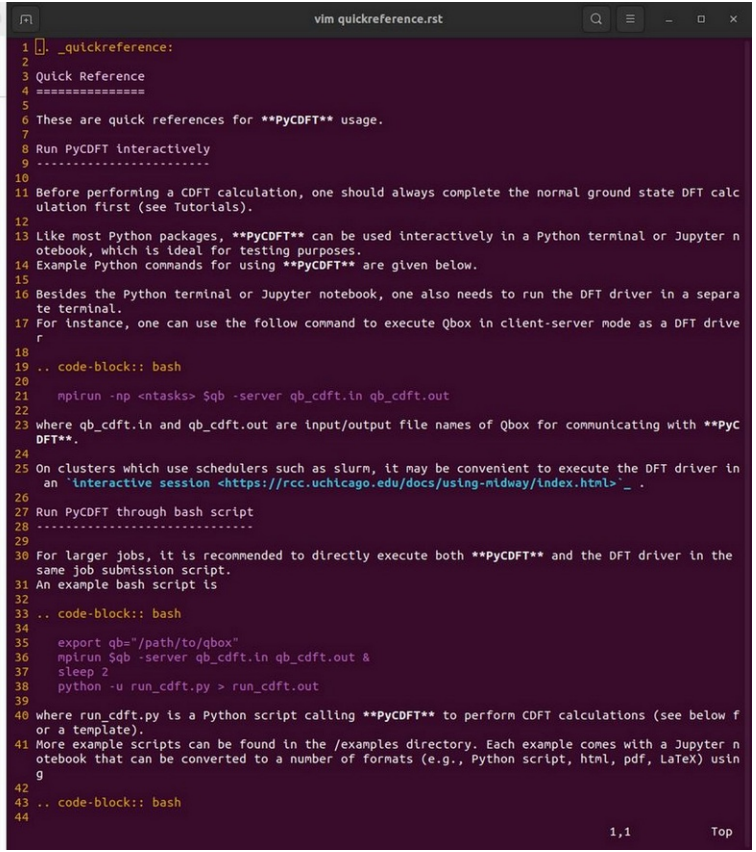
Handling Docstring Indentation

Docstring processing tools will strip a uniform amount of indentation from the second and further lines of the docstring, equal to the minimum indentation of all non-blank lines after the first line. Any indentation in the first line of the docstring (i.e., up to the first newline) is insignificant and removed. Relative indentation of later lines in the docstring is retained. Blank lines should be removed from the beginning and end of the docstring.

Creating documentation: Sphinx

Generate beautiful documentation, e.g., html

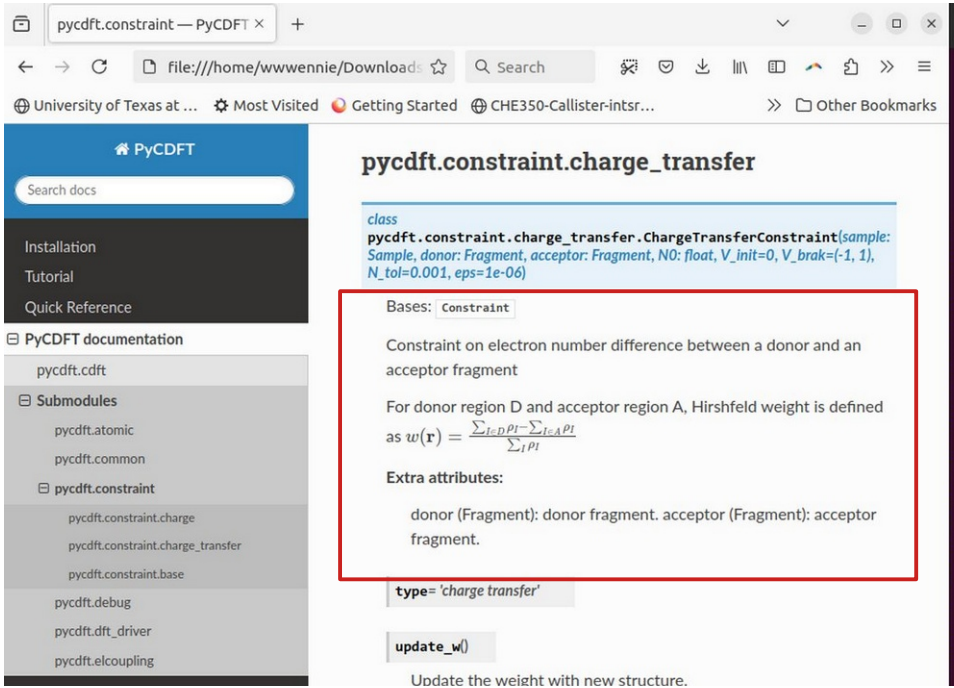


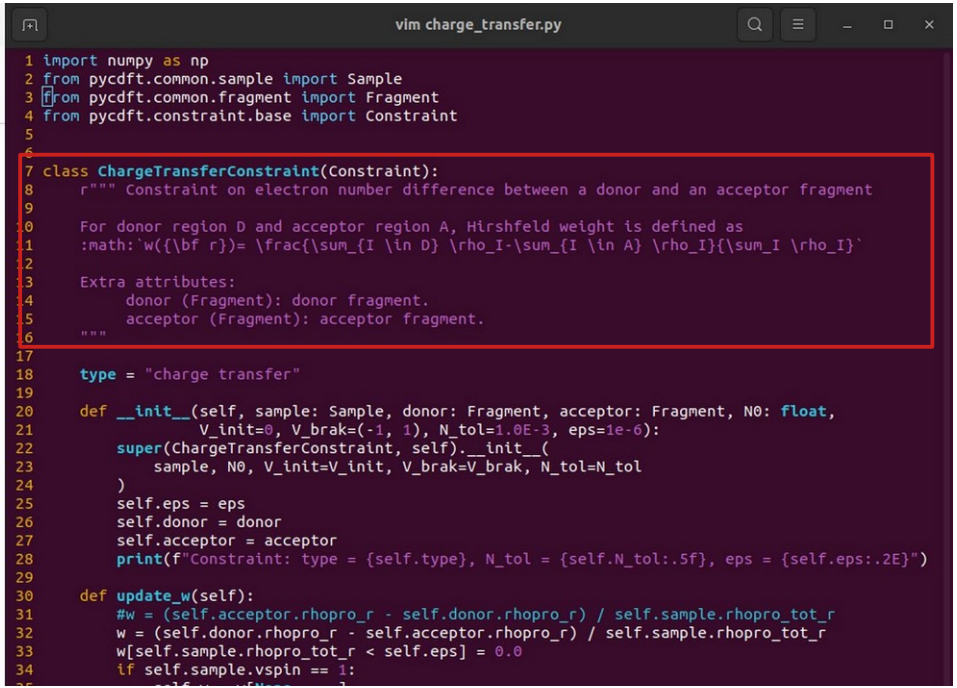


https://www.sphinx-doc.org/en/master/index.html

Creating Documentation: Sphinx

Generate beautiful documentation, e.g., html





With equations!

Sphinx

Creating documentation: Doxygen

Markdown rendering

```
# Markdown rendering

Type on the __left__, see it
_rendered_ on the __right__.

This is [link](https://www.doxygen.nl).

- And this is
- A list.

\emoji :tada:
```

Markdown rendering

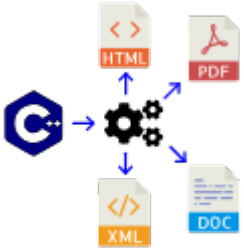
Type on the **left**, see it *rendered* on the **right**.

This is link.

- And this is
- A list.

🎉

Several output formats



Cross-referencing

◆ stripFromIncludePath()

QCString stripFromIncludePath (const **QCString** & path)

strip part of *path* if it matches one of the paths in the **Config_getList(INCLUDE_PATH)** list

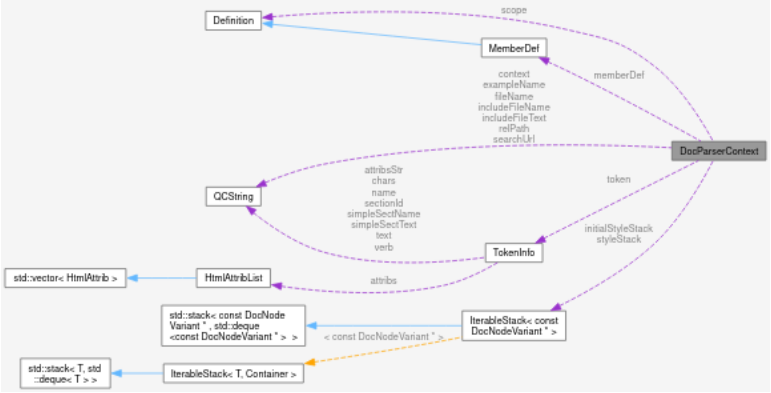
Definition at line **337** of file **util.cpp**.

```
{
  return stripFromPath(path,Config_getList(STRIP_FROM_INC_PATH));
}
```

References **Config_getList**, and **stripFromPath()**.

Referenced by **addIncludeFile()**, and **findFileDef()**.

Graph representation of relationships between classes and functions



Basic strategies for speeding up Python

Python can be slower compared to other scientific computing languages, such as C, C++ or Fortran

For small to medium coding tasks, the difference in speed is imperceptible or tolerable

For larger coding tasks (e.g., more complex computations), it may be worth the time to invest on speeding up portions of the code

- Implement and test the code (on a small test case)
- Profile if the code is slow
- Optimize parts of the code with bottleneck performance
- Rinse and repeat

For even larger computing tasks, it may worth using a different programming language

Basic strategies for speeding up Python

Python

- **Interpreted language:** sequential processing and translation to machine code on the fly
- **Dynamic typing:** variable types are determined at runtime and checked throughout
- **Automatic memory management:** “garbage collection”- check and reclaim unused memory
- **Abstracts hardware details**

C/C++/Fortran

- **Compiled language: t**ranslation to machine code before execution
- **Static typing:** variable types are explicitly typed at time of compilation
- **Manual memory management:** programmer dictates when memory is allocated and freed
- **Fine-grain control of hardware details:** e.g., memory layout, CPU instructions, hardware specific optimizations

Basic strategies for speeding up Python

Profile your code: https://realpython.com/python-profiling/

Basic `time` module: Measure the Execution Time

Intermediate `timeit` module: Benchmark Short Code Snippets with basic statistics
e.g., average of *n* runs, best time of *n* runs
Use directly in the code or use command-line interface

Advanced `cProfile` module: Collect Detailed Runtime Statistics,
deterministic profiler
e.g., number of function calls
very detailed but lots of overhead

Pyinstrument (third party tool): Take Snapshots of the Call Stack
statistical profiler
filters out insignificant calls that do not affect performance

Basic strategies for speeding up Python

Common places to look for performance (memory/speed) bottlenecks and some solutions

- **Loops:** nested loops, loops with large datasets; vectorize with numpy or pandas
- **Too much object creation**; reuse objects, minimize data structure conversions
- **Memory-heavy data structures**: dictionaries, lists; numpy and pandas; generators or iterators to avoid loading memory of large data structures
- **Global Interpreter Lock:** limits multi-threading; `multiprocessor` module
- **Inefficient use of lists or string operations:** use built-in functions
- **Too much copying of data:** shallow v deep copy
- **Too many function calls:** move outside loop, one-line functions
- **Lack of pre-allocated memory:** dynamic resizing of data structures; pre-allocated with `numpy.empty()` or `numpy.zeros()`
- **Non-optimized i/o:** line-by-line reading; read the file in chunks with buffered i/o

https://wiki.python.org/moin/PythonSpeed/PerformanceTips

An example: vectorize loops with numpy arrays

```python
import numpy as np
import time

# Define the function to integrate
def f(x):
    return np.sin(x)

# Integration bounds and number of intervals
a = 0  # lower bound
b = np.pi  # upper bound
n = 10**6  # number of intervals
dx = (b - a) / n  # step size

# Create the x values (intervals)
x = np.linspace(a, b, n)

### 1. Using a for loop ###
start_time = time.time()

integral_loop = 0.0
for i in range(1, n):
    integral_loop += f(x[i]) * dx

end_time = time.time()
loop_time = end_time - start_time
print(f"Time taken by for loop: {loop_time:.6f} seconds")

### 2. Using vectorized NumPy operations (Trapezoidal rule) ###
start_time = time.time()

integral_vectorized = np.sum(f(x)) * dx

end_time = time.time()
vectorized_time = end_time - start_time
print(f"Time taken by vectorized operation: {vectorized_time:.6f} seconds")

# Compare the results of the two approaches
print(f"Integral (for loop): {integral_loop:.6f}")
print(f"Integral (vectorized): {integral_vectorized:.6f}")
```

Numerically evaluate definite integral

$$\int_0^{\pi} \sin(x)dx = 2$$

with basic code profiling

Output (on 2017 Lenovo Thinkpad)

```
Time taken by for loop: 2.366934 seconds
Time taken by vectorized operation: 0.015893 seconds
Integral (for loop): 1.999998
Integral (vectorized): 1.999998
```