2023

1

Data Science and AI

Module 1
Part 2:

Python for Data Science

Agenda: Module 1 Part 2

- Python Fundamentals

- Software Engineering Best Practices

- Using Git & GitHub for Version Control

Python versions: 2.7 vs 3.x

- version 2.x
  - large code base
  - last version = 2.7 (no more releases!)
- version 3.x
  - *print* is a function
  - raising & catching exceptions
  - integer division (2.x truncates; 3.x converts to float)
  - short → long integers
  - octal constants: 0*nnn* → 0o*nnn*
  - unicode strings
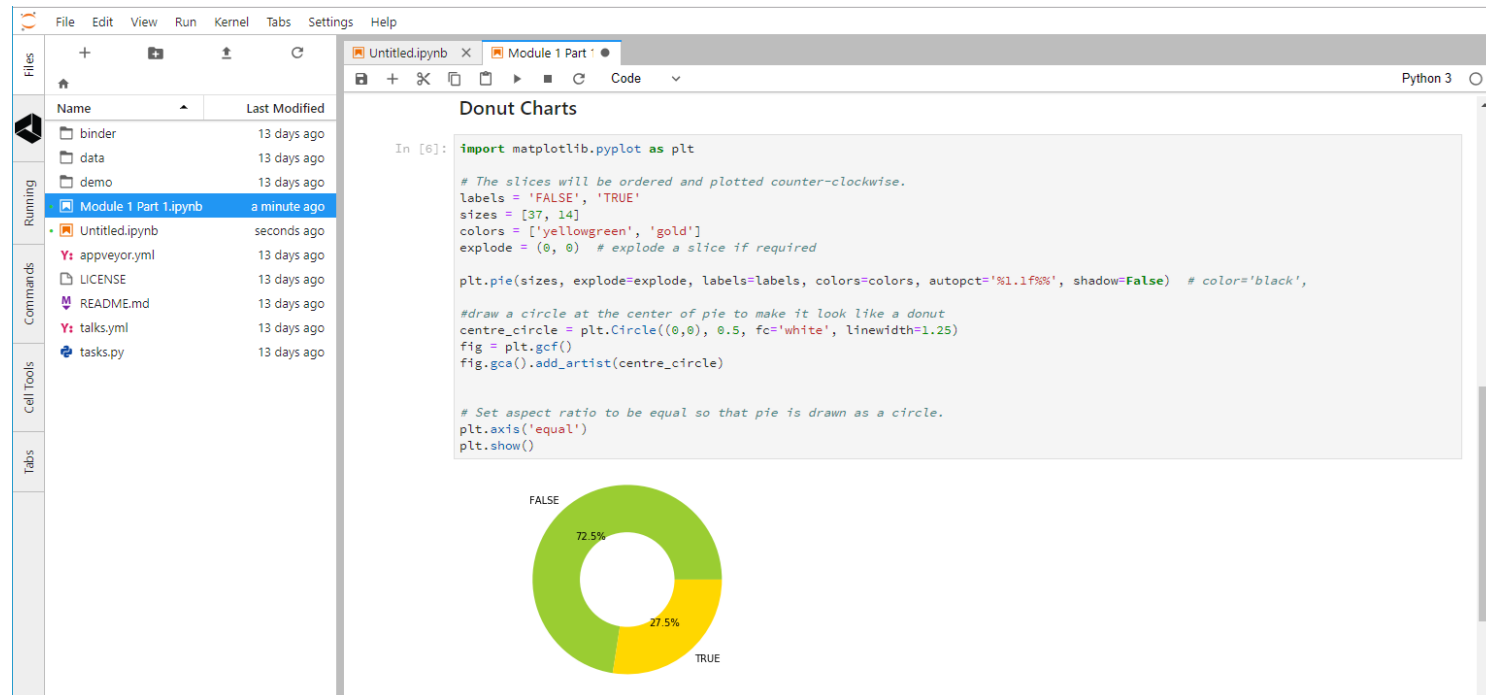 - …

Developing and running Python

- Jupyter notebook

- Visual Studio Code (VSC)

  - VSC now has built-in Jupyter notebook support

- Jupyter Lab

- Command prompt

- Anaconda

  - Anaconda distribution is **the recommended way** to configure and manage your Python development and running environment(s).
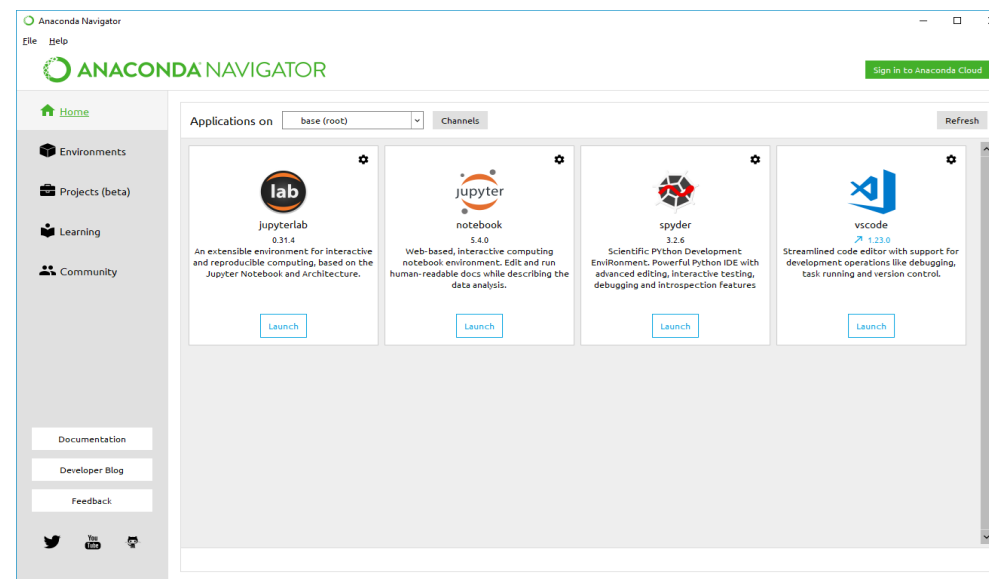
Jupyter Notebooks

- shareable

- environment-based

- interactive or batch execution

- > 40 languages
  - Python, R, Scala, …

- Big Data support
  - Spark

Generic Data Types

Numeric	Text	Other
integer • signed, unsigned	character • unicode	Boolean • true, false Binary • 2^n
floating-point ('float') • double = 2 x float	string • character array • 0-based *or* 1-based • null-terminated *or* length-encoded • usually immutable in OOP	unassigned • null • NA undefined • NA • +, − infinity
complex • 2 x double (real, imaginary)	document • key-value pairs (JSON strings)	BLOB • images, video • signals

Classes

```python
class phasor:
    def __init__(self, r=0, p=0):
        self.r = r
        self.p = p
    def real(self):
        return (self.r * math.cos(self.p))
    def imag(self):
        return (self.r * math.sin(self.p))

z = phasor(2.7, 0.4 * math.pi)
```

- 2 underscores before/after init

- the **self** parameter is not explicitly mapped to the function call

Pandas

- high-performance, easy-to-use data structures and data analysis tools
  - DataFrame class
  - IO tools
  - data alignment
  - handling of missing data
  - manipulating data sets
    - reshaping, pivoting
    - slicing, dicing, subsetting
    - merging, joining

import pandas as pd

https://pandas.pydata.org/

Scikit-learn

- biggest library of ML functions for Python
  - classification
  - regression
  - clustering
  - dimensional reduction
  - model selection & tuning
  - preprocessing

$ pip install -U scikit-learn
or
$ conda install scikit-learn

http://scikit-learn.org/stable/

Other Python Packages for Data Science

- statsmodels
  - statistical modelling & testing
  - R-style formulae

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

- BeautifulSoup
  - reading & parsing XML & HTML data

```
from bs4 import BeautifulSoup
```

- Natural Language Toolkit
  - tokenising, tagging, analysing text

```
import nltk
```

Lab 1.2.1: Numpy

1. Explain the following NumPy methods and create working examples in Jupyter notebook using the data created for you in the beginning of the Lab notebook:

2. Structure your code using functions (prepare to discuss the value of using functions).

- ndim
- shape
- Size
- itemsize
- data
- linspace
- mean
- min

- max
- cumsum
- std
- sum

...

3. Stretch exercise. Use matplotlib to explore the data

Lab 1.2.2: Pandas

1. Explore and download Employee Attrition file from Kaggle
   (https://www.kaggle.com/HRAnalyticRepository/employee-attrition-data)
2. Explain the following Pandas methods and create working examples in the lab Jupyter notebook.
3. Structure your code using functions (prepare to discuss the value of using functions.

- read_csv
- describe

- loc

- iloc

- index

- sort_index

- set_index

- sample

- …

4. Stretch exercise. Use matplot to explore some of the data in the data frame

Software Engineering Best Practices

- Object-Oriented Programming

- Refactoring

- Coding for readability

- Coding for testability

- Documenting

Object-Oriented Programming

- an *object* encapsulates
    - data (*attributes*)
    - procedures (*methods*)
- a *class* is a prototype for an object
    - *instantiation*: creating an object (in memory) from a class definition

def: **encapsulation**
- attributes of the class should only be accessible by methods of the class
    - get()
    - set()

Creating and Using a Class in Python

```
class myclass:
    def __init__(self, param1, …):
        # initialise class attributes

    def method1(self, ):
        # do something
        return (method1result)


obj1 = myclass(arg1, …)
```

- define class by name
  - initialisation code
    - only **self** is mandatory
    - may use arguments passed from caller
  - define methods
    - only **self** is mandatory
    - may use arguments passed from caller
    - may use attributes
    - may return a value

- invoke class name in assignment to instantiate an object
  - omit **self**

Other OOP Concepts

def: **abstraction**

- data and procedures that do not need to be accessible to the caller should be hidden within the class

def: **inheritance**

- new classes can be based on and extend an existing class

def: **polymorphism**

- a class can implement multiple methods with the same name and function, but which operate on different parameters (type and/or number)

Refactoring

def: Restructuring existing code without changing its behaviour

Examples

- abstract reused code to functions
  - generalise functions (polymorphism?)
- use get, set methods
- simplify structure of nested loops, logic
- minimise use of global variables
  - in Python, this includes all variables defined in main program

Coding for Readability (Maintainability)

Examples

- indent blocks
  - mandatory in Python

- white space
  - between groups of lines
  - between symbols
- comments: inline (to explain logic, return values, etc.)
  - sectional (to explain functional blocks)
  - header (to explain program or module)
    - purpose, authors, date
    - dependences, assumptions

- comments are for coders
  - maintaining or extending your code
- documentation is for users
  - explaining what the application is for and how to use it

© 2023 Institute of Data

Coding for Testability

Examples

- avoid side-effects in functions
- enable testing via compiler flags

```
##define TEST_MODE
#if TEST_MODE
print("test mode activated")
#endif
```

- write tests *before* functions
  - specify return type(s) supported
  - test return type(s), validity
  - pass sample data as arguments
  - print result

- test *frequently*
  - avoid marathon coding sessions
- code top-down
  - create wireframe code to test logic, structures
  - fill in the details later

pytest

https://docs.pytest.org/en/latest/

© 2023 Institute of Data

Questions?

Appendices

Version Control with Git & GitHub

- Forking
- Cloning
- Communicating issues
- Managing notifications
- Creating branches
- Making commits
- Introducing changes with Pull Requests

Git & GitHub

- web-based, API
- host code, data, resources
- version control
  - integrates with open-source and commercial IDE tools
- share, collaborate
  - branching
- showcase achievements
- command line & desktop versions
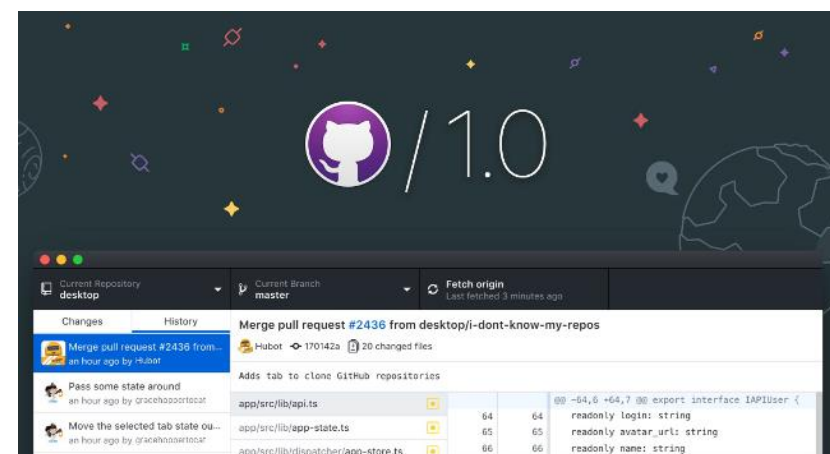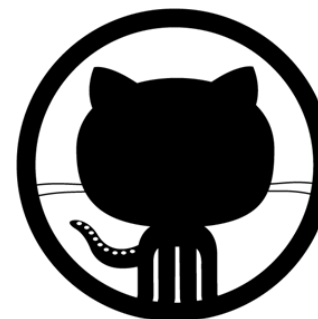
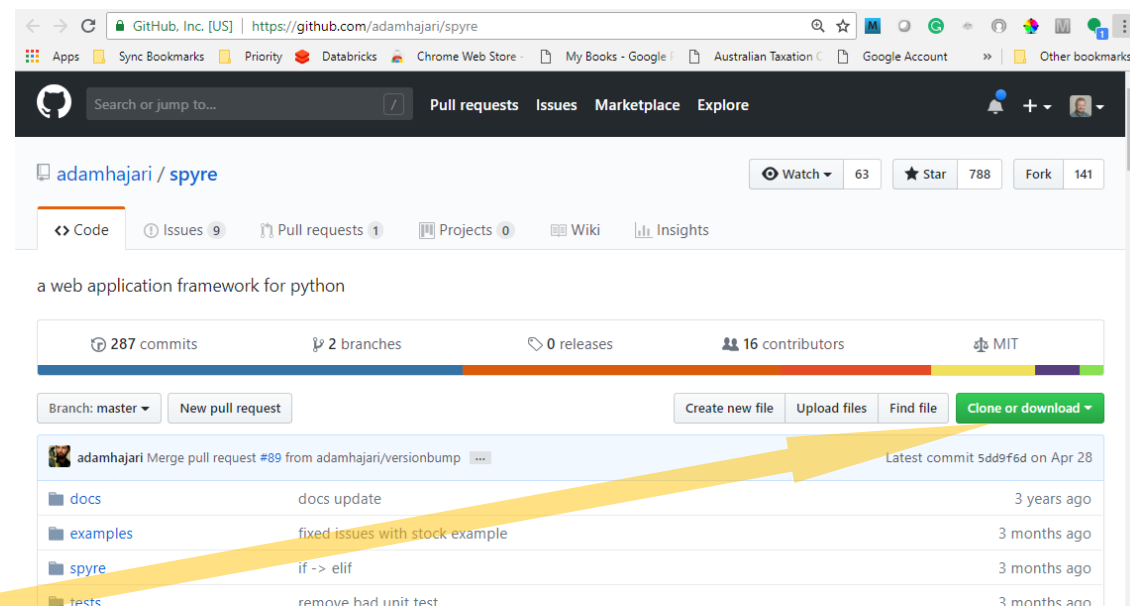# GitHub: Forking & Cloning a Repo

- *fork:* make your own copy of someone else's repo, on GitHub
    1. click <Fork>

- *clone:* create a (working) copy of the repo on your computer

- GitHub Desktop procedure:
    1. click <Clone or download>
    2. click <Open in Desktop>
   3. navigate to target (local) folder
    4. click <Clone>

- command-line procedure:
    1. $ cd yourpath
    2. $ git clone https://github.com/ yourgithubname/yourgithubrepo

25

© 2023 Institute of Data

GitHub: Creating a New Repo

- from your GitHub home page

  1. <New repository>

  2. clone the repo to your local drive

  3. copy files, folders into it

  4. commit changes

  5. generate a *pull* request

- Creating a branch

  - to allow development in isolation from source repo

    - protects your changes from changes to source
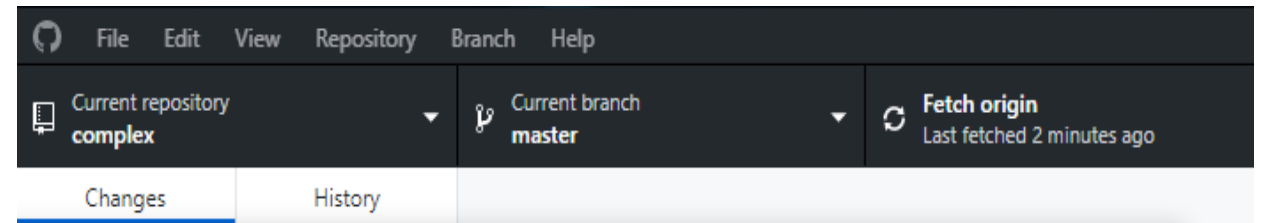
    - rejoin main branch when ready
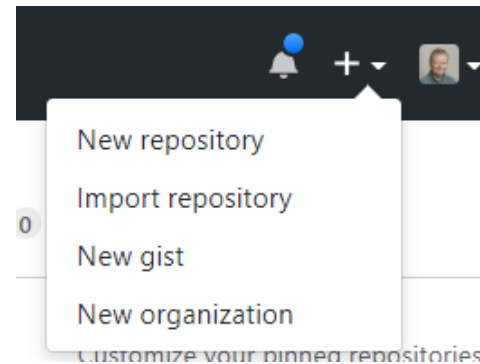
GitHub: Refreshing Local Repo from Source

Desktop

• <Fetch origin>

Command-line

$ git checkout master
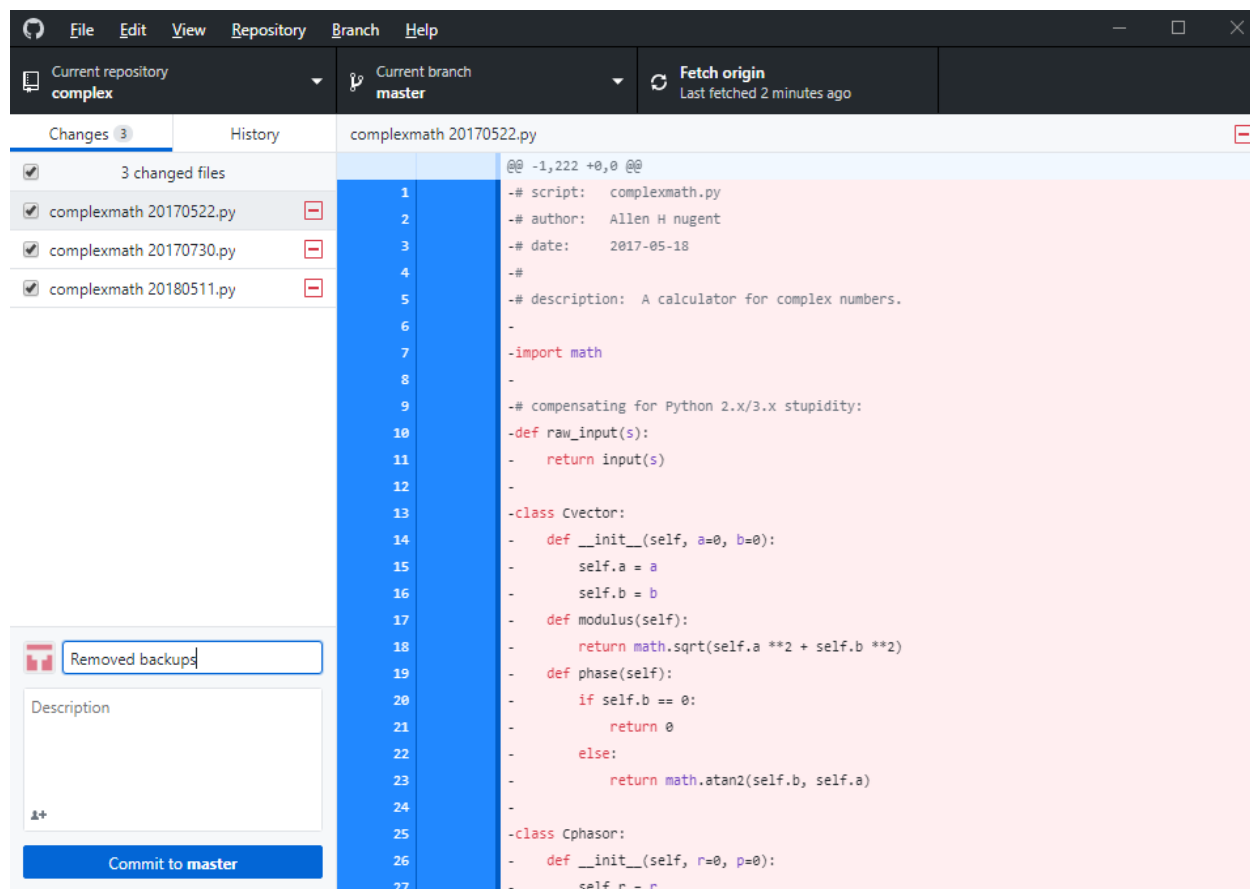
$ git fetch upstream

$ git merge upstream/master

• Ensure you're in the master branch

• Grab the latest changes from the master

• Merge the master changes with your repo

GitHub: Commit & Pull Request

Desktop

- enter comments in text box

- <Commit to master>

- Repository > Push
  or
  <Push origin>

© 2023 Institute of Data

GitHub: Commit & Pull Request

Command-line

- commit

  $ git status

  $ git add filename

  $ git add .

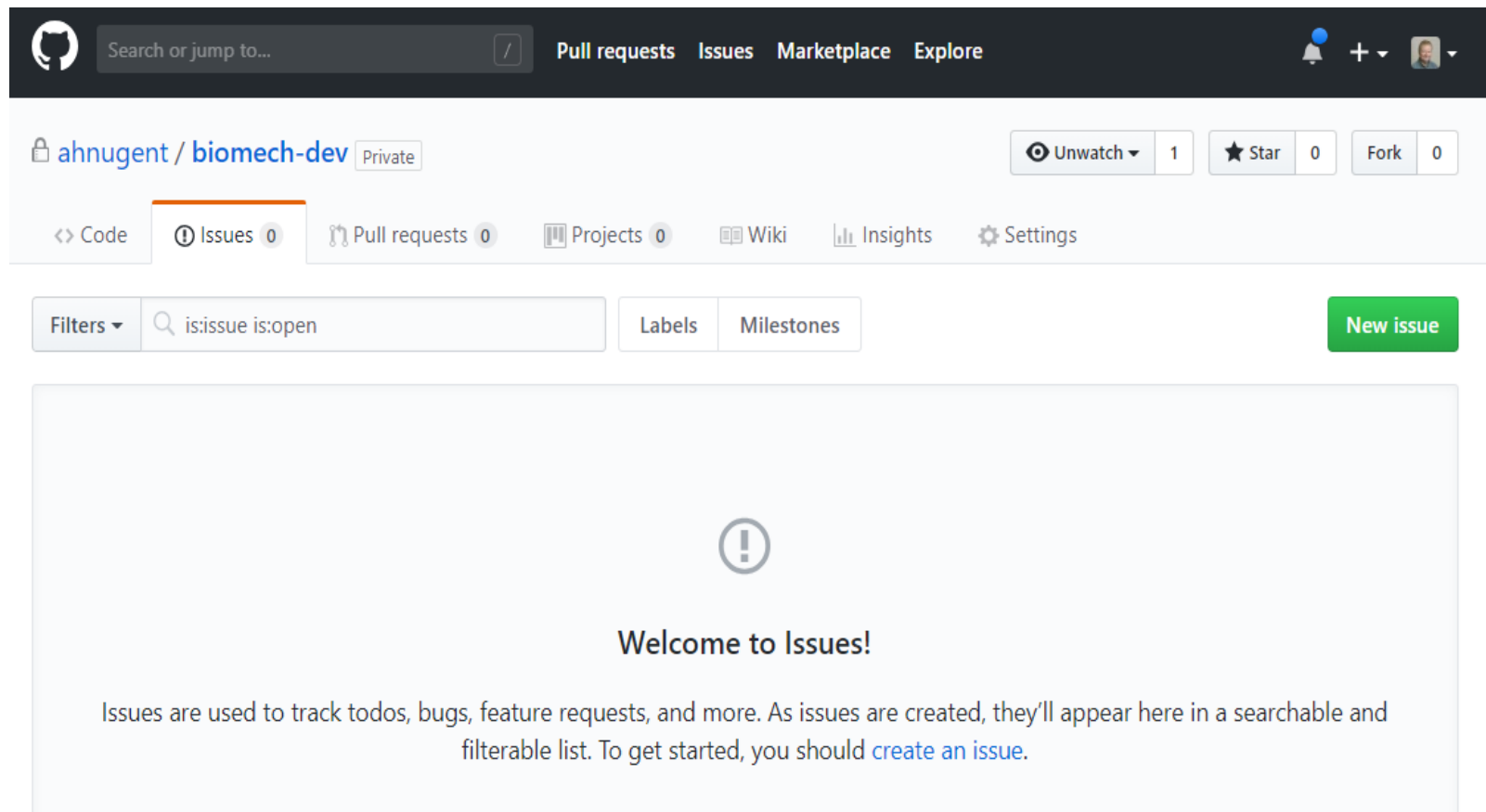 $ git commit -m your_comments

  $ git status

- pull request

  $ git push origin master

- show changes

- stage one file

- stage all change

- commit file(s), with comments

- origin = your GitHub repo (forked from source repo)

- master = source repo

GitHub: Issues

- track
  - issues / bugs
  - to-do items
  - feature requests
- search
- filter

© 2023 Institute of Data

GitHub: Notifications

Triggers

- you, a team member, or a parent team are mentioned
- you're assigned to an issue or pull request
- a comment is added in a conversation you're subscribed to
- a commit is made to a pull request you're subscribed to
- you open, comment on, or close an issue or pull request
- a review is submitted that approves or requests changes to a pull request you're subscribed to
- you or a team member are requested to review a pull request
- you or a team member are the designated owner of a file affected by a pull request
- you create or reply to a team discussion

End of Presentation!

32