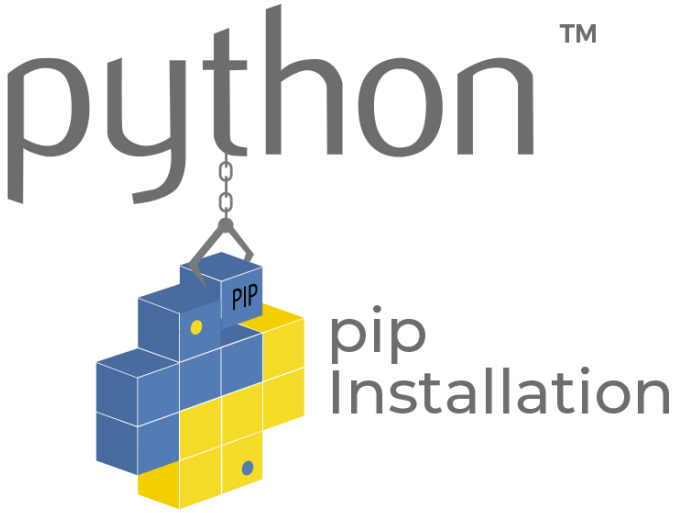
Packages, Virtual Environments, and GitHub



Modules

A module is a file containing Python definitions and statements. The file name is the module name with the suffix `.py` appended. Within a module, the module's name (as a string) is available as the value of the global variable `__name__`. For instance, use your favorite text editor to create a file called `fibo.py` in the current directory with the following contents:

The methods fib() and fib2() are listed in a file named fibo.py.

They can be imported simply:

```
import fibo
```

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

__main__

- A module may contain some initialization code in a “main” function.

- Unlike other languages “main” is not explicitly a function but part of the namespace when a file/module is executed. Thus all files have __main__ if they are directly executed via “python <filename>.py”

```
python fibo.py 50
```

Execute the fibo module directly

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

Initialization of the fido module

```python
def fib(n):  # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a + b
    print()


def fib2(n):  # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a + b
    return result


# if this module is called directly, take the command line
# argument and do something with it

if __name__ == "__main__":
    fib(int(sys.argv[1]))
```

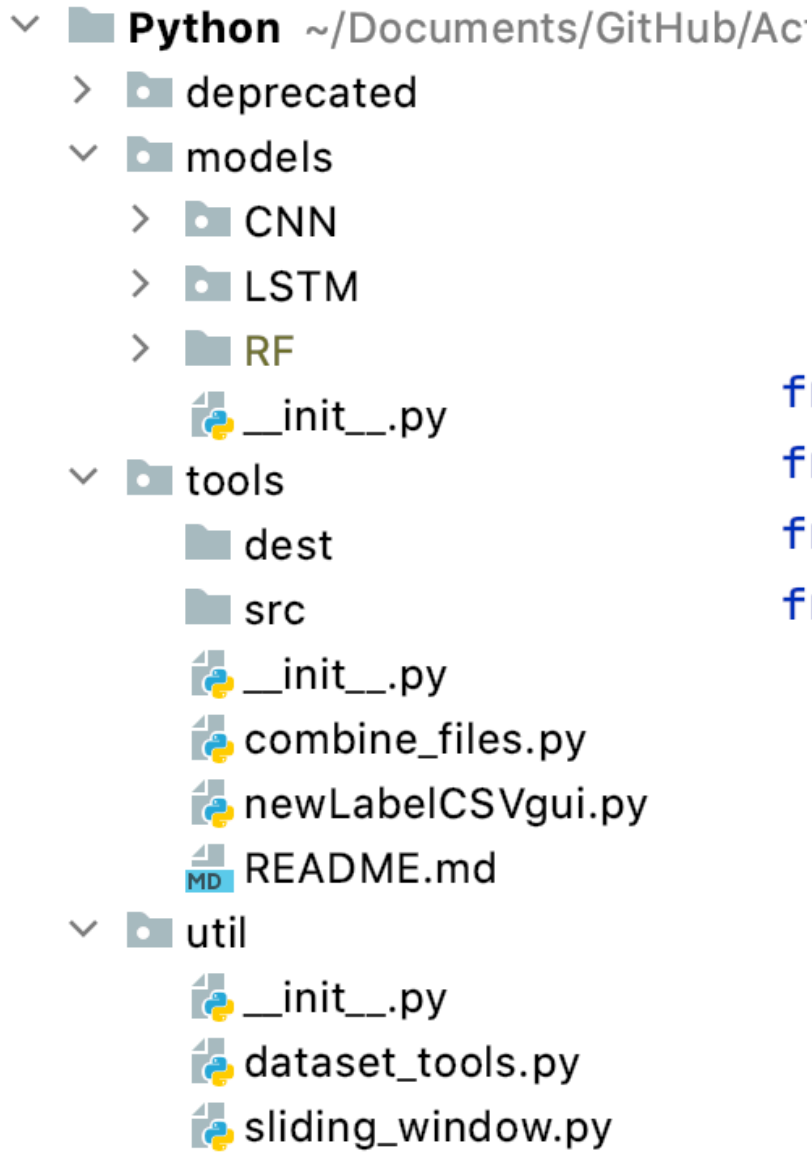
Packages

- When there are many modules, even with submodules, a more complex organization is required.

- A package is like a module but organized as a *directory*, rather than a single *file*. Each folder has its own __init__.py to perform initialization (like the __main__ in modules)

- Due to their complexity, packages are often installed via a package manager. We will discuss the python package installer (pip) later.

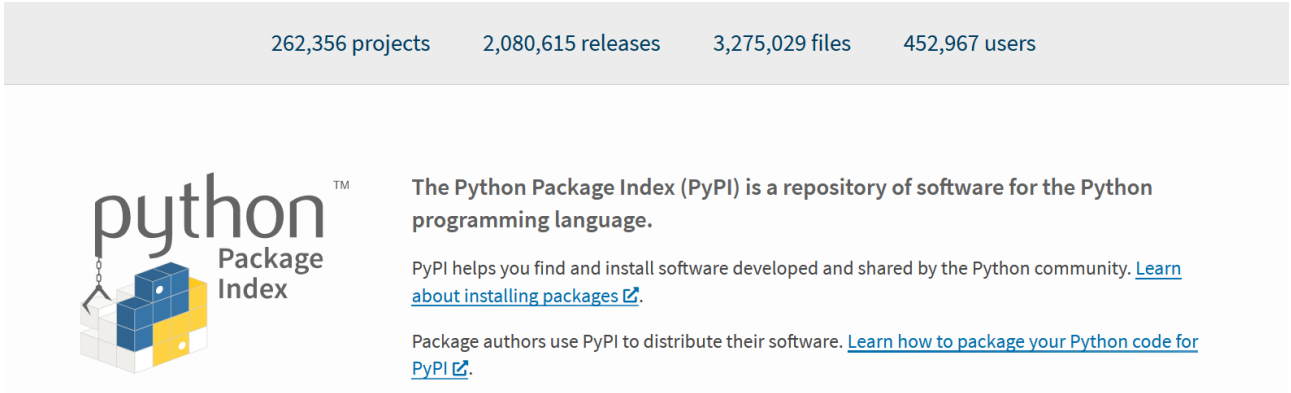
```
sound/                            Top-level package
      __init__.py                 Initialize the sound package
      formats/                    Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                    Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                    Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```




```
from util.dataset_tools import read_data,normalize_dataset
from project_config_loader import ProjectConfig
from models.CNN.conv_nn import conv_nn
from models.CNN.CNN_Params import CNN_Params
```

Pip and PyPI

- Pip is the installer for the Python Package Index (PyPI)
- *pip install <package>*
- Manages all the “fun” of installing packages and their dependencies.
- Within PyCharm, pip installation are handled in the background or can be run manually in the Terminal.




```
jason@DESKTOP-7JQ1H37:~$ pip install scipy
Collecting scipy
  Downloading https://files.pythonhosted.org/packages/24/40/11b
/scipy-1.2.3-cp27-cp27mu-manylinux1_x86_64.whl (24.8MB)
    100% |                                | 24.8MB 68kB/s
Collecting numpy>=1.8.2 (from scipy)
  Downloading https://files.pythonhosted.org/packages/3a/5f/47e
/numpy-1.16.6-cp27-cp27mu-manylinux1_x86_64.whl (17.0MB)
    100% |                                | 17.0MB 102kB/s
Installing collected packages: numpy, scipy
Successfully installed numpy-1.16.6 scipy-1.2.3
```

Manually Installing SciPy in the Terminal

```
(venv) jforsyth@Jasons-MacBook-Pro lecture-examples % pip install scipy
Collecting scipy
  Downloading scipy-1.8.0-cp38-cp38-macosx_12_0_universal2.macosx_10_9_x86_64.whl (55.3 MB)
     |████████████████████████████████| 55.3 MB 5.3 MB/s
Requirement already satisfied: numpy<1.25.0,>=1.17.3 in /Users/jforsyth/Documents/GitHub/ENG
 (1.22.2)
Installing collected packages: scipy
Successfully installed scipy-1.8.0
```

Ok, so I install packages, where do they go?

Windows ¶

The Python installers for Windows include pip. You should be able to access pip using:

```
py -m pip --version
pip 9.0.1 from c:\python36\lib\site-packages (Python 3.6.1)
```

Linux and macOS

Afterwards, you should have the newest pip installed in your user site:

```
python3 -m pip --version
pip 9.0.1 from $HOME/.local/lib/python3.6/site-packages (python 3.6)
```

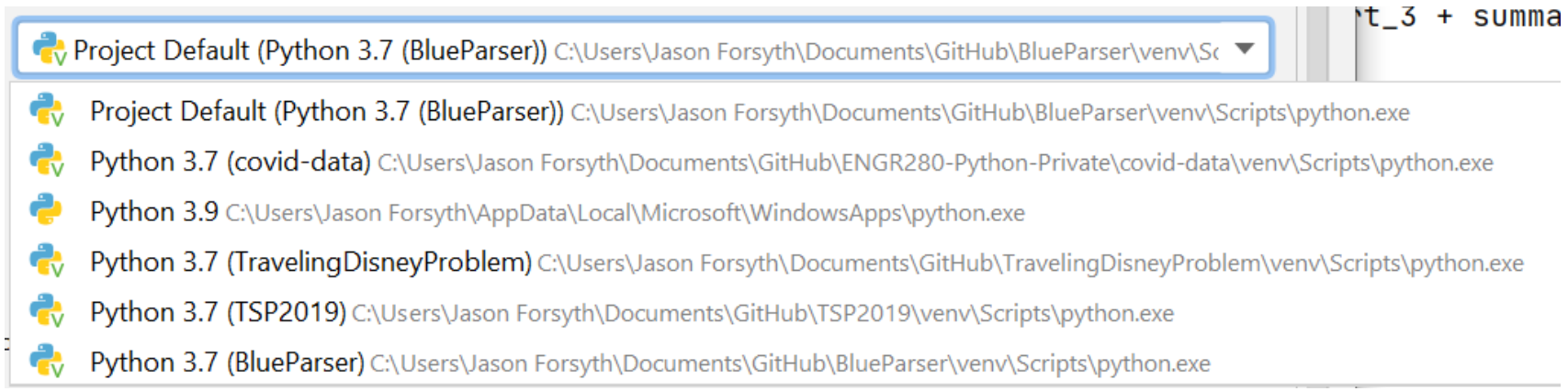
Great! All my packages in one nice place…

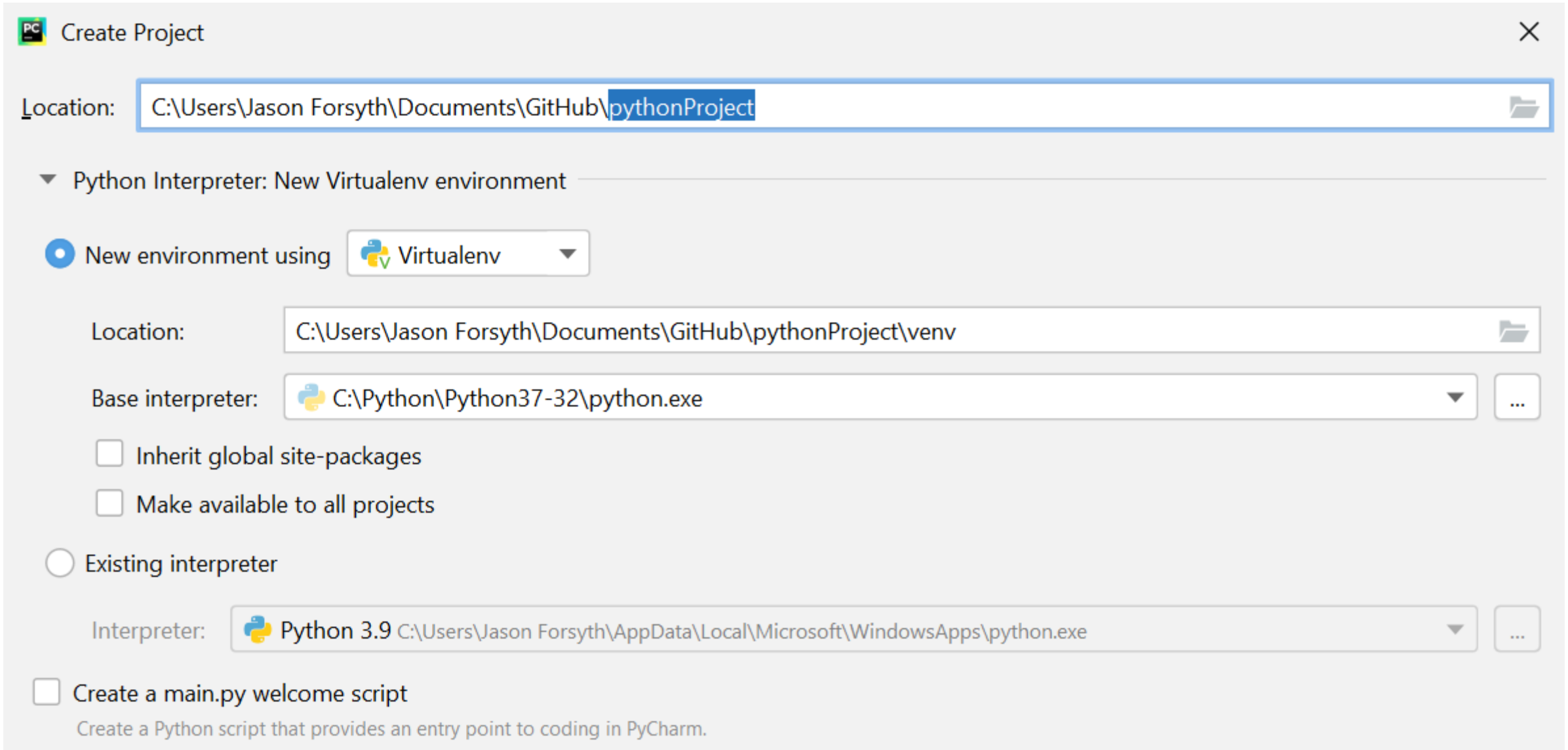
- It’s wonderful until you need a new package: do you admin rights to add it?
- What happens when you’ve written 10,000 lines with tensorflow v1.0 but the newest version is v2.0? How do you get students running a clean/old install for that version?
- What happens when your package conflicts with another? You need numPy v0.1 that needs the latest version of QT, but then your old code needs QT-beta?
- Want to try out a new package version without ruining all your code?

The solution is virtual environments

- Give each application its own environment to install its own package version. Once it runs in the environment, leave it alone.

- Local packages from the global installation can be imported (for large things like numpy, scipy…etc) while each application can manage it's own packages




Create Project
Location:
C:\Users\Jason Forsyth\Documents\GitHub\pythonProject
Python Interpreter: New Virtualenv environment
New environment using
Virtualenv
Location:
C:\Users\Jason Forsyth\Documents\GitHub\pythonProject\venv
Base interpreter:
C:\Python\Python37-32\python.exe
...
Inherit global site-packages
Make available to all projects
Existing interpreter
Interpreter:
Python 3.9 C:\Users\Jason Forsyth\AppData\Local\Microsoft\WindowsApps\python.exe
...
Create a main.py welcome script
Create a Python script that provides an entry point to coding in PyCharm.

Project: BlueParser › **Python Interpreter** For current project

Python Interpreter: Python 3.7 (BlueParser) C:\Users\Jason Forsyth\Documents\GitHub\BlueParser\venv\Scripts\python.exe

Package	Version	Latest version
certifi	2019.11.28	▲ 2020.6.20
chardet	3.0.4	3.0.4
cycler	0.10.0	0.10.0
idna	2.8	▲ 2.10
kiwisolver	1.1.0	▲ 1.2.0
matplotlib	3.1.1	▲ 3.3.2
numpy	1.15.3	▲ 1.19.2
pandas	0.24.2	▲ 1.1.2
pip	20.2.3	20.2.3
pynput	1.4.2	▲ 1.7.1
pyparsing	2.4.0	▲ 2.4.7
pyserial	3.4	3.4
python-dateutil	2.8.0	▲ 2.8.1
pytz	2019.1	▲ 2020.1
requests	2.22.0	▲ 2.24.0
scipy	1.3.0	▲ 1.5.2
setuptools	50.3.0	50.3.0
six	1.12.0	▲ 1.15.0
urllib3	1.25.7	▲ 1.25.10
virtualenv	16.1.0	▲ 20.0.31

Currently, there are two common tools for creating Python virtual environments:

- venv is available by default in Python 3.3 and later, and installs pip and setuptools into created virtual environments in Python 3.4 and later.
- virtualenv needs to be installed separately, but supports Python 2.7+ and Python 3.3+, and pip, setuptools and wheel are always installed into created virtual environments by default (regardless of Python version).

The basic usage is like so:

Using venv:

```
python3 -m venv <DIR>
source <DIR>/bin/activate
```

Using virtualenv:

```
virtualenv <DIR>
source <DIR>/bin/activate
```

Packages and Virtual Environment Summary

- Package management is a huge headache; avoid at all costs.
- pip is the package manager (not apt-get, windows update…etc.)
- By default packages are installed globally but this should only be a few “major” or common packages.
- Use virtual environments for each project; I really can’t see a downside. This is the default in PyCharm

Version Control: Why Everything But Git is Terrible

How do to manage a distributed and collaborative software project?

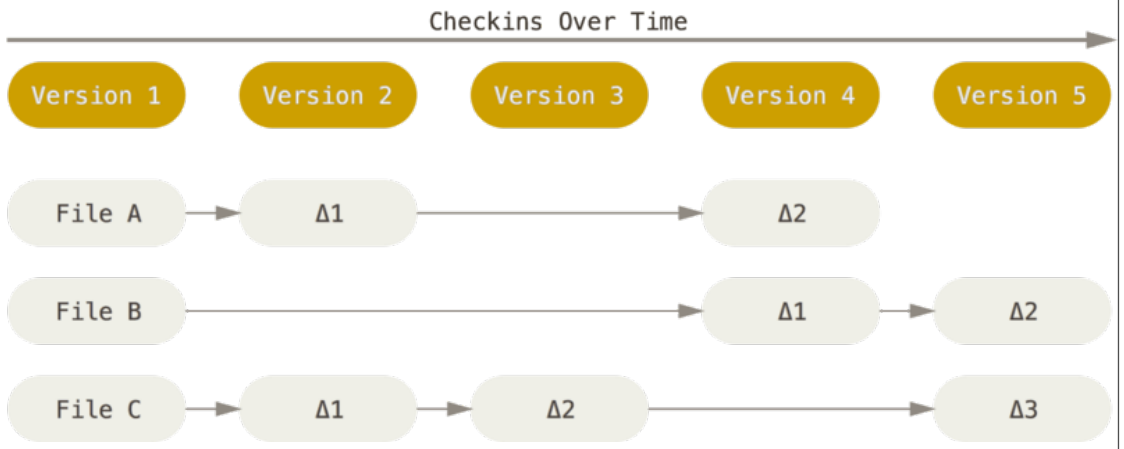
- Code bases are millions of lines with potentially thousands of developers in different times zones, work flows…

- **Challenge #1: how to notice and manages in changes?**
  - What happens if you depend on Helper.py file and then someone else modifies it? What if someone depends on your code?

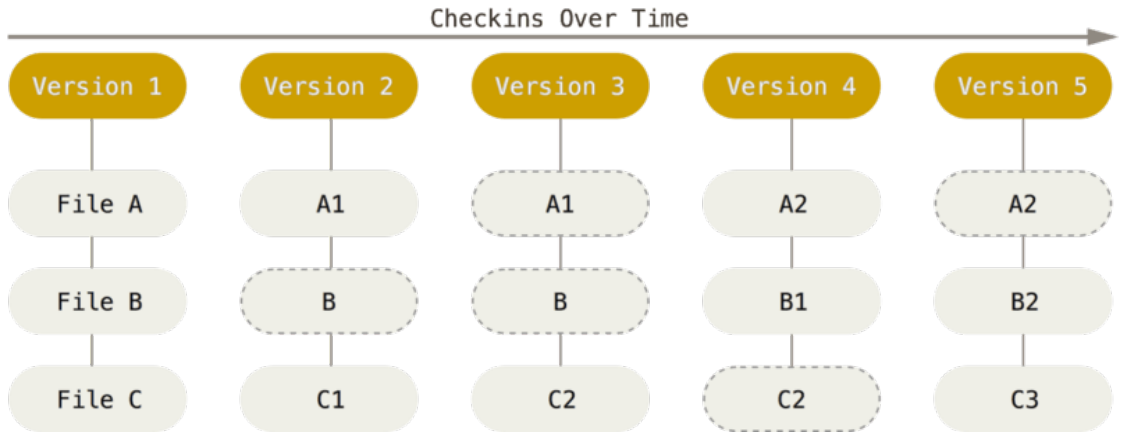
- **Challenge #2: what if two people are working on the same document?**
  - It’s a giant .py file but you’re editing different/same methods. Local the file? Lock the module?

It’s an obvious problem with many previous solutions

- Many previous version control software tools have existed. Mercurial, Subversion, CVS, BitKeeper….
- I’ve used several of these; they’re not fun…
- Honestly, they’re all terrible in the face of Git Kitty.

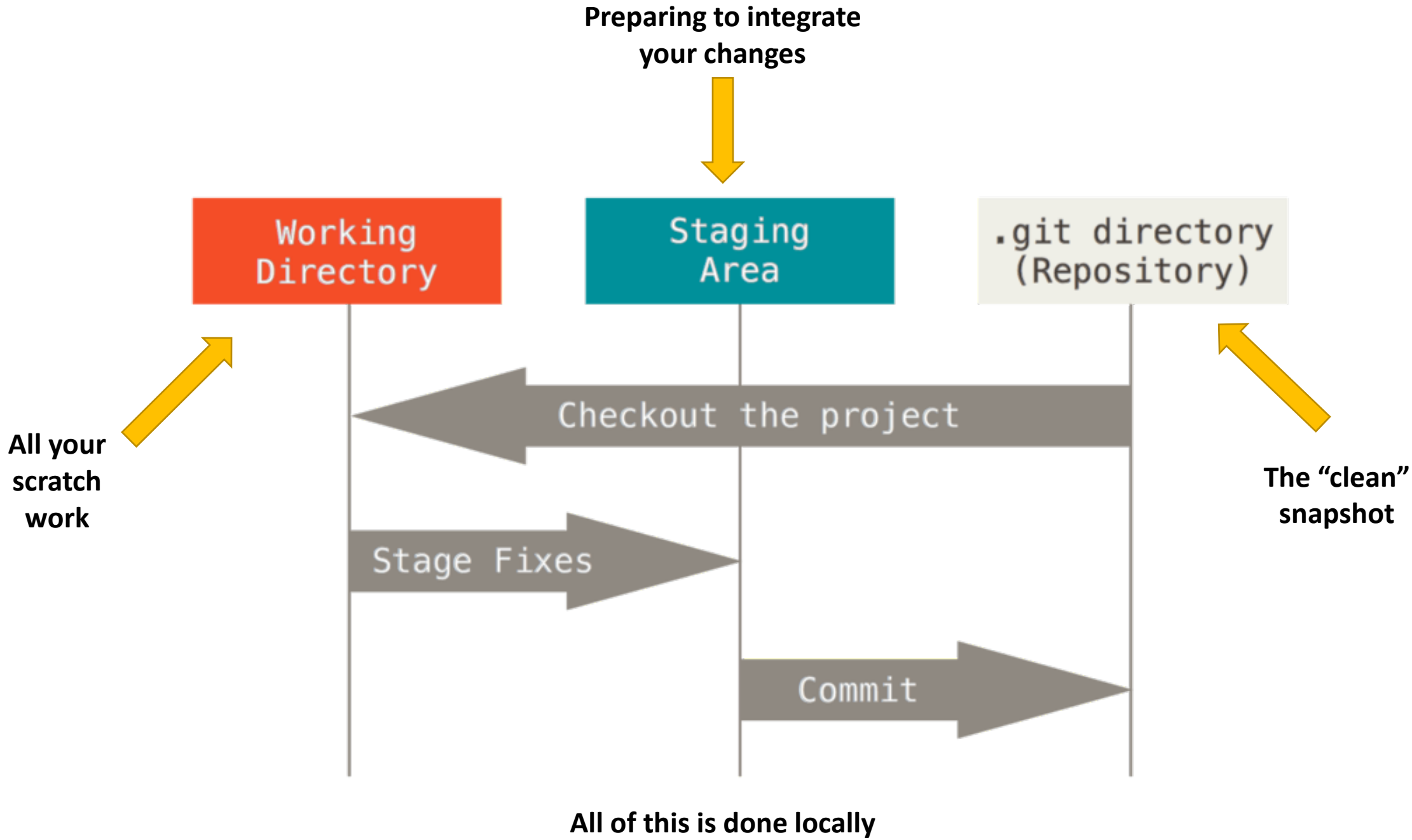


A “Delta” Approach to Managing Files



A “Snap Shot” Approach to Managing Files

Git takes a “snapshot” of the file system each time but only stores “delta“ information.
Each snapshot is a “whole” image.

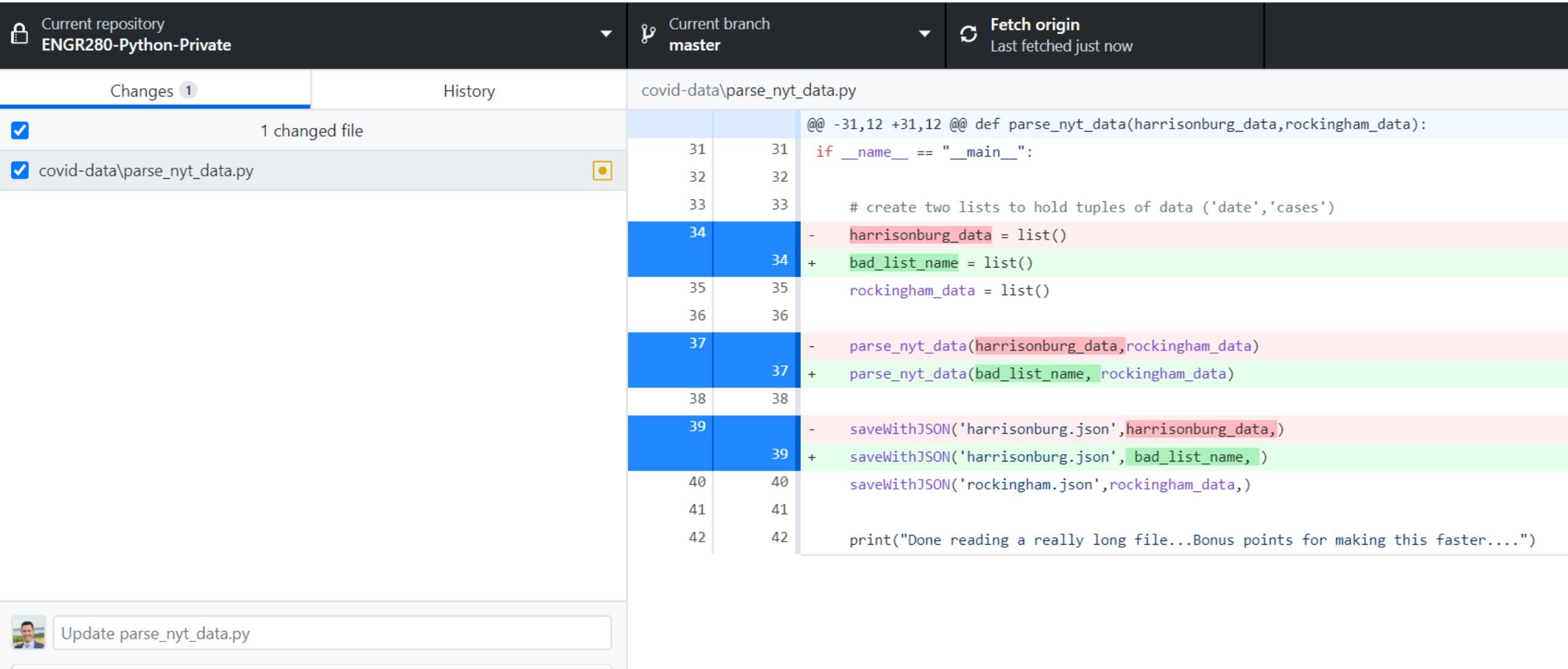

Preparing to integrate your changes
Working Directory
Staging Area
.git directory (Repository)
All your scratch work
Checkout the project
The “clean” snapshot
Stage Fixes
Commit
All of this is done locally

Some Git Terminology

- *repo*: a repository where code is kept and maintained. The repo is under *version control*. Your repo is local.

- *stage*: to prepare files for a commit. Not all changes need to be staged. Only staged changes are committed.

- *commit*: to send a change to a repo. Should not cause conflicts with elements already in the repo.



Staging and Committing a potential change to the repo

Current repository
ENGR498-Private

Current branch
master

Fetch origin
Last fetched 2 minutes ago

Changes | History

No branches to compare

Re-arrange variable updates
Jason Forsyth • Oct 8, 2019

Initial version of ADXL345 class
Jason Forsyth • Oct 8, 2019

Add in memory reading/writing examples.
Jason Forsyth • Sep 30, 2019

Add in security warning
Jason Forsyth-VM • Sep 19, 2019

Move #define for security warnings
Jason Forsyth • Sep 17, 2019

Add in datasets
Jason Forsyth • Sep 6, 2019

Add compile instructions
Jason Forsyth • Sep 6, 2019

Remove filepath info
Jason Forsyth • Sep 5, 2019

Expose boolean for readability
Jason Forsyth • Sep 5, 2019

Add small CSV parse program
Jason Forsyth • Sep 5, 2019

Add Diagnostic 2
Jason Forsyth • Aug 30, 2019

Re-arrange variable updates

Jason Forsyth a70f356 1 changed file Hide Whitespace

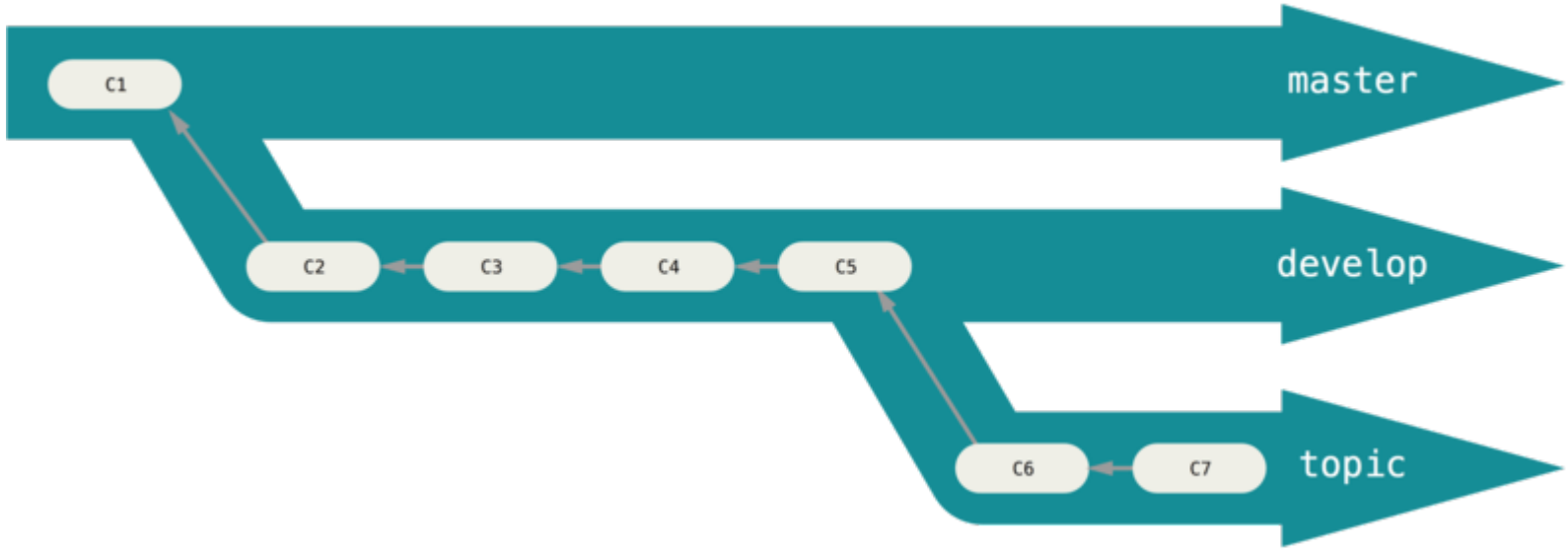
milestones\spi-class\ADXL345.cpp

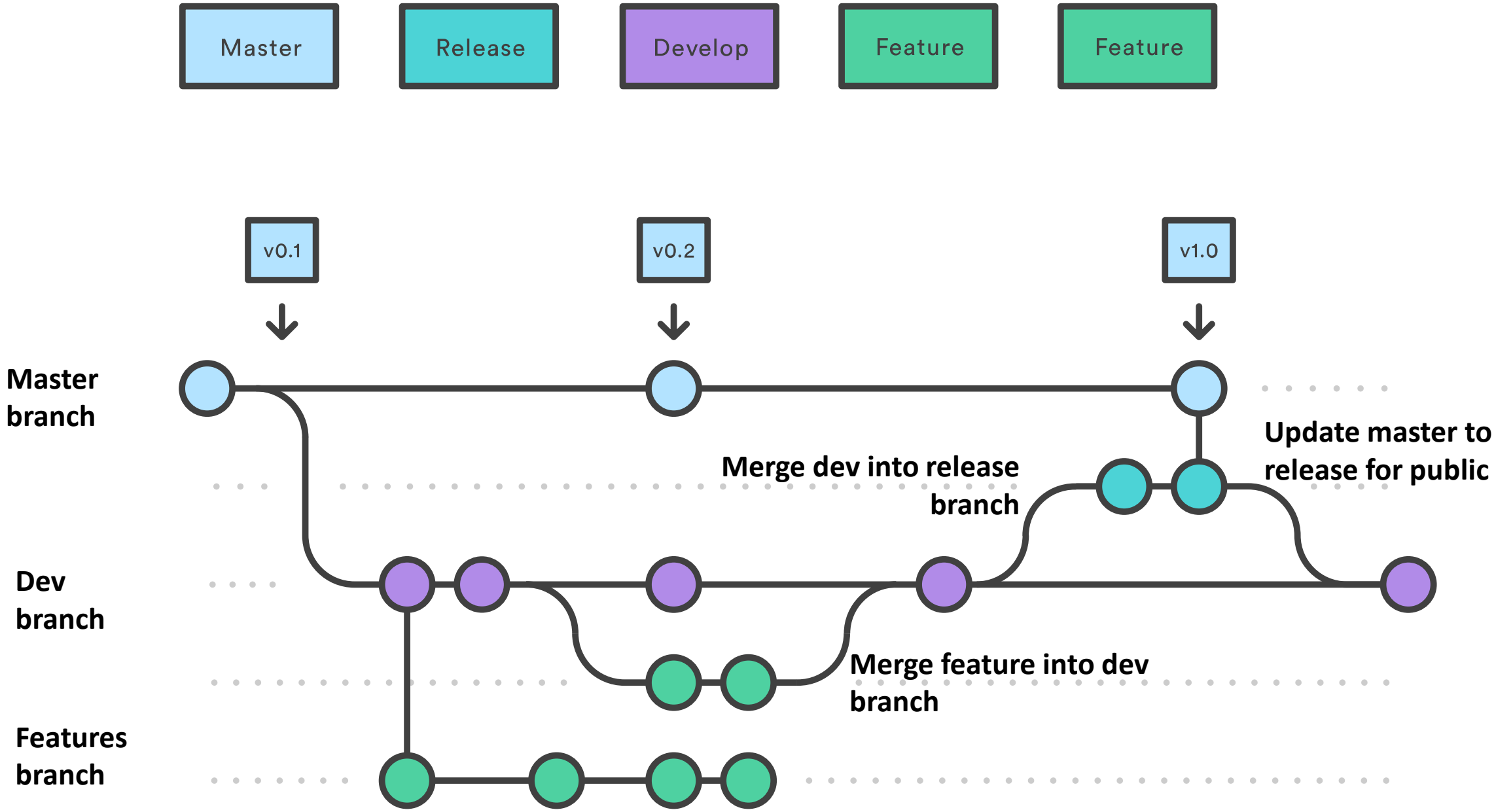
```
@@ -24,9 +24,11 @@ void ADXL345::setup(int _CS_PIN, ADXL_RANGE _RANGE/*=ADXL_RANGE::TWO_G*/, ADXL_R
 {

    //***Step #0***
-   //record where the CS is attached and set that pin as output
-   //and initial value is HIGH
+   //record where the CS is attached, the current range, and resolution
+   //set the CS pin to be an OUTPUT HIGH
    csPin=_CS_PIN;
+   currentRange=_RANGE;
+   currentResolution=_RESOLUTION;
    pinMode(csPin,OUTPUT);
    digitalWrite(csPin,HIGH);

@@ -40,11 +42,9 @@ void ADXL345::setup(int _CS_PIN, ADXL_RANGE _RANGE/*=ADXL_RANGE::TWO_G*/, ADXL_R
    SPI.setDataMode(SPI_MODE3);

    //***Step #3 (Optional) ***
-   currentRange=_RANGE;
    //write to the DATA_FORMAT register to adjust the sampling range

    //***Step #4 (Optional) ***
-   currentResolution=_RESOLUTION;
    //write to the DATA_FORMAT register to adjust the sampling range

    //***Step #5***
```

A major benefit of all this is Branches

- *branch*: an independent sequence of commits.

- *merge*: to combine two branches



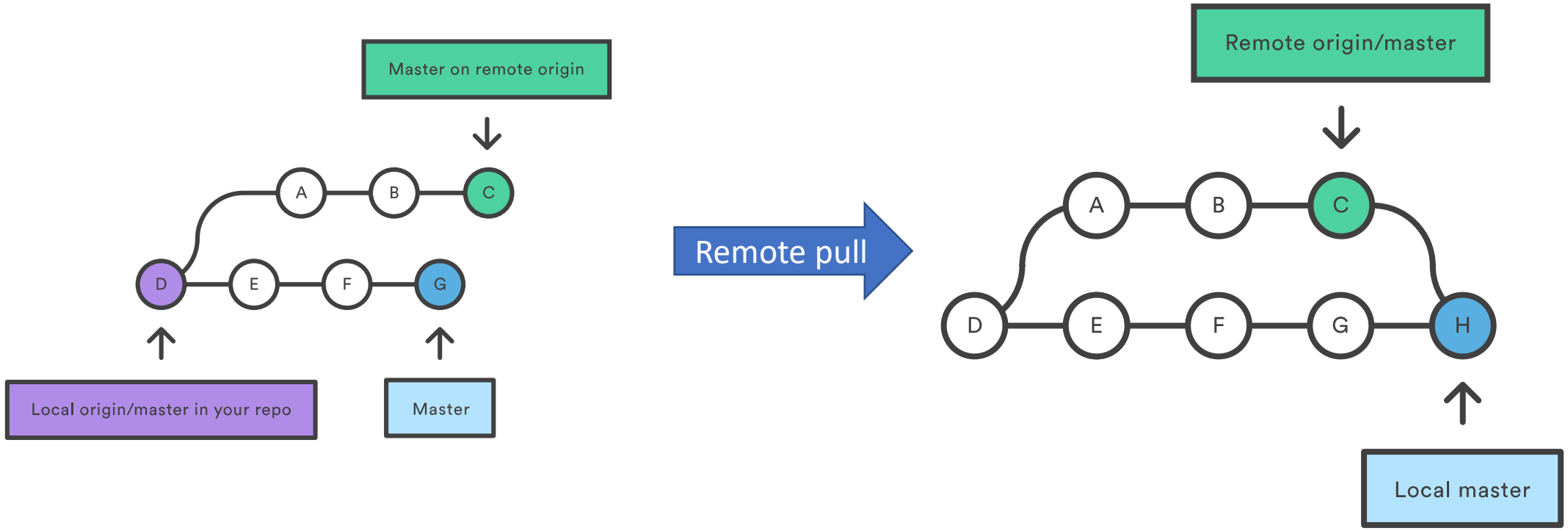


https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Working Remotely

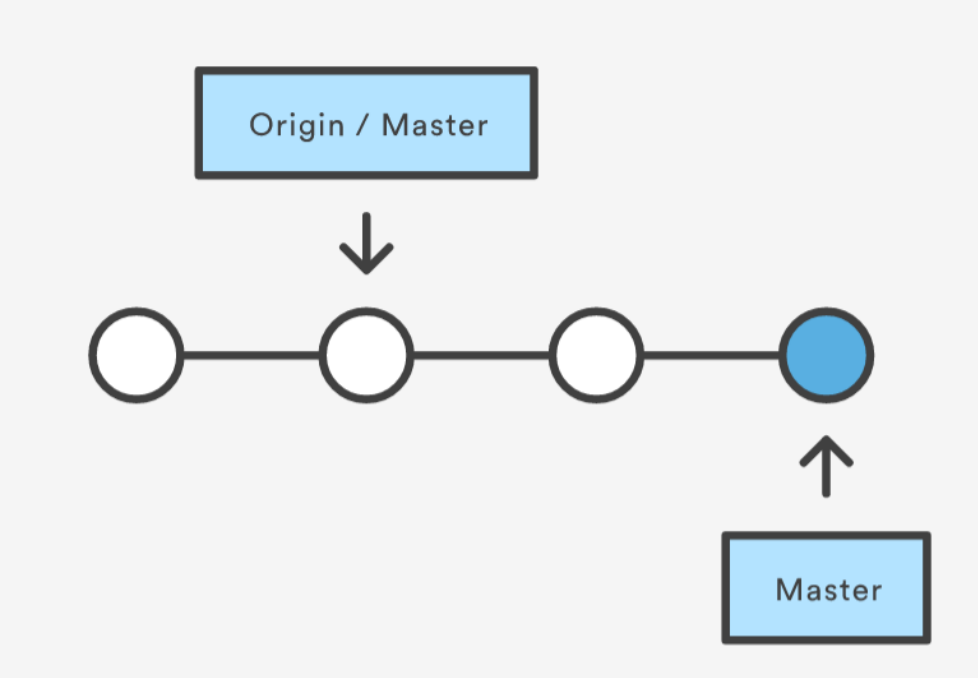
- *local:* your repo on your desktop, laptop…etc.
- *remote:* an associated external repo that will be kept sync. Remote repo will be called origin/
- *push*: to send committed changes in a repo to another remote location.
- *pull*: to receive changes from a remote repo.
- Synchronization of push/pulls may require additional merges locally or remotely

Pulling In Remote Changes

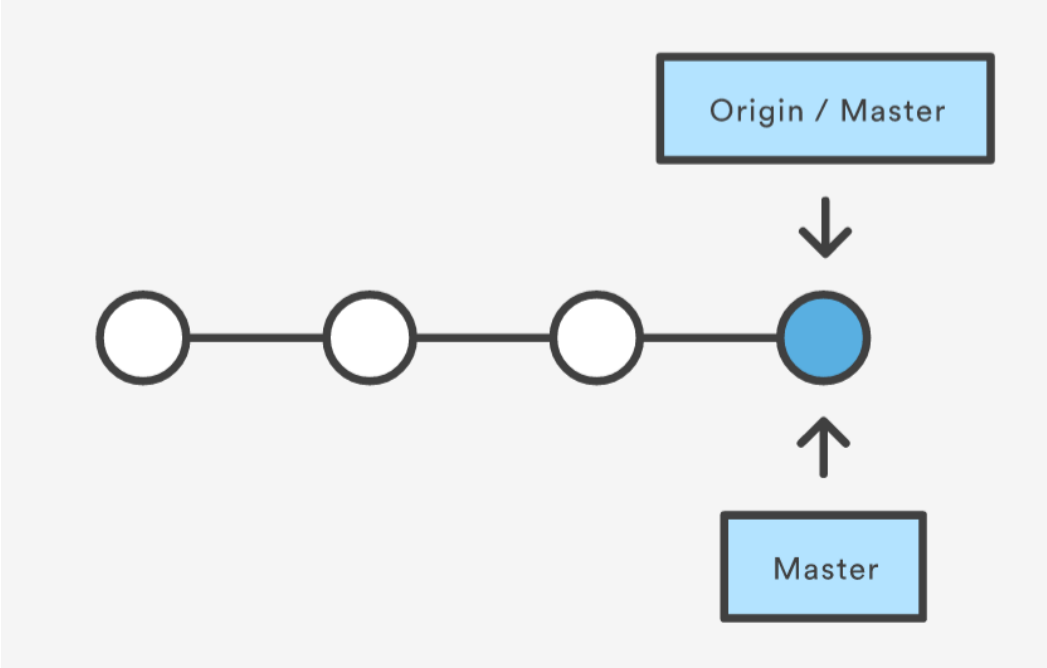


https://www.atlassian.com/git/tutorials/syncing/git-pull

Pushing Changes to Remote



Remote master is “behind” local master



After push, remote master and local are in sync

https://www.atlassian.com/git/tutorials/syncing/git-push

Git Benefits

- All changes are tracked. You can always go backwards. ☺
- Because you have all the changes (and they're hashed), you can check for any modification. Did someone change the code and you not know?
- Git repos can live on their own. Once you have a copy you can do what you like but then still compare against others.

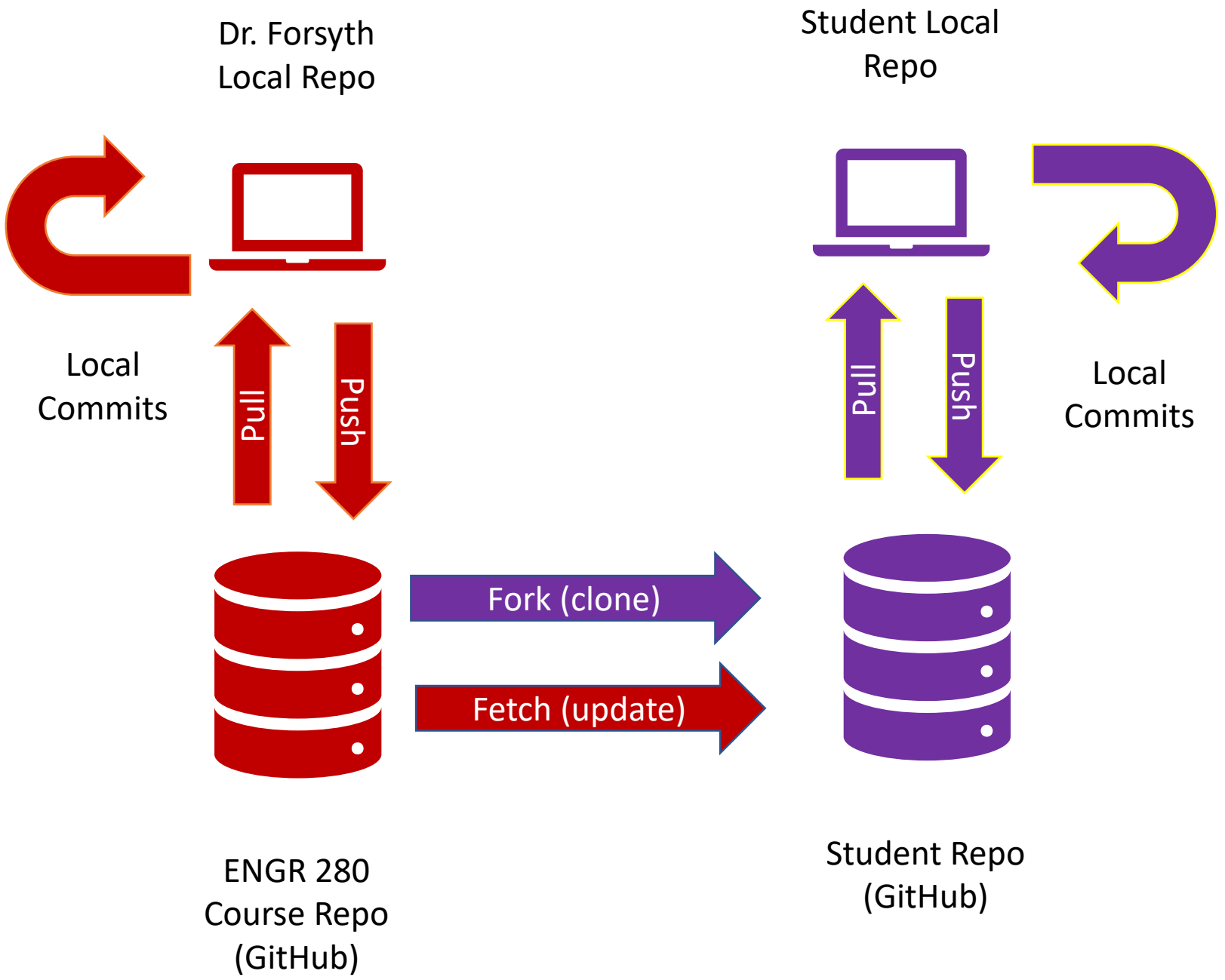
Using Git in this Course

1. Fork Dr. Forsyth repo to create a copy. Upstream should be pointed to your own copy.

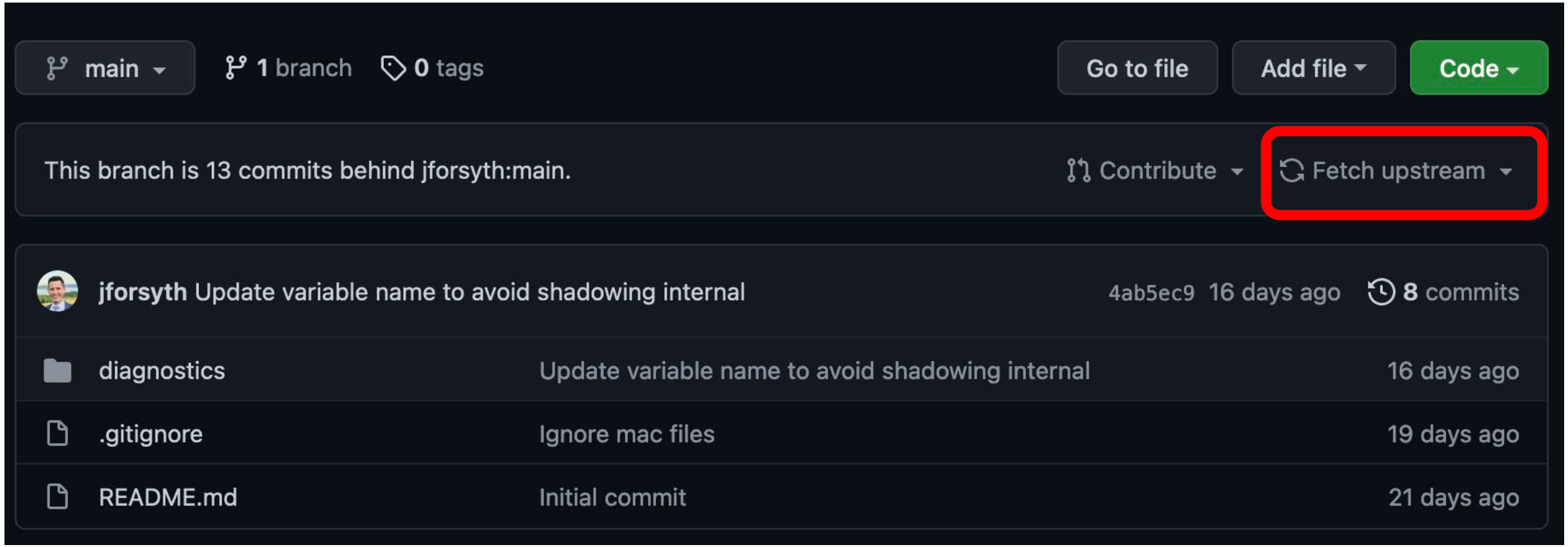
2. Student commits made to local repo. May “push” upstream to GitHub.

3. If changes are made to course repo, can Fetch to bring changes to GitHub repo.

4. Those changes must be pulled into local repo to see on laptop.



3. Fetch Upstream Changes

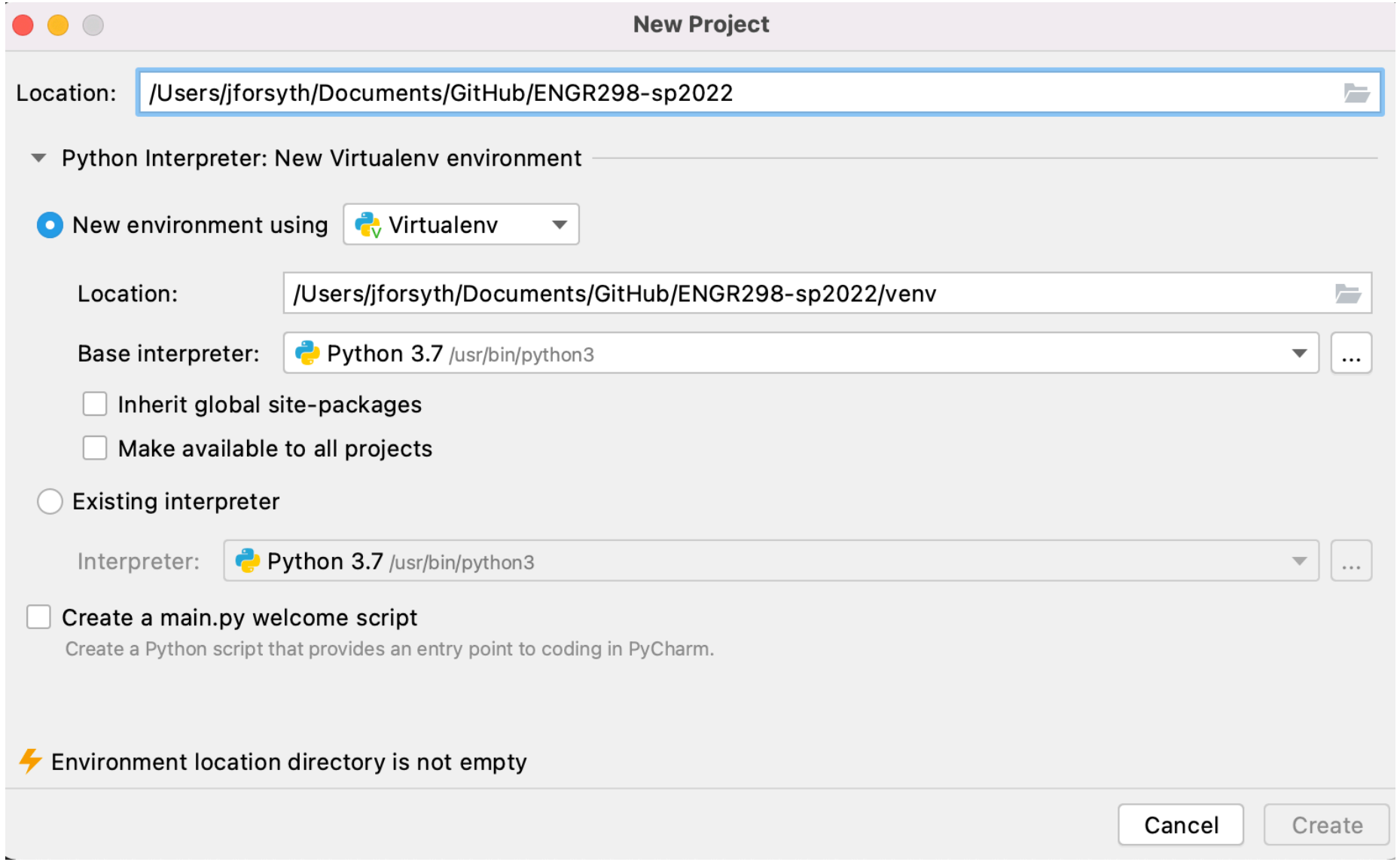


Dealing with Merge Conflicts and Push

- Conflicts should not occur when fetching updates of the course repo. This implies that ”my” files have been modified.

- Your GitHub client should “target” your own repo, not mine, so changes are pushed to your repo. If you wish to push a change to me, issue a Pull Request. (In the rare event there are errors).

In PyCharm, Target GitHub



Final Thoughts

- Keep a folder called “workspace” in your repo. Place all work/submissions there. May wish to organize by assignment, project…etc.

- May wish to show off your work on GitHub. Direct evidence to employers “what you have done” in this course.

- May need to tweak GitHub processes but main objective is to provide easy access to course files.