Git with a Shared Repository

Dept of Economics, Univeristy of Zurich

Programming Practices

October 8, 2018

Learning Objectives

- At the end of the session you will:
  1. Understand the value Git adds to your collaboration with your coauthors
  2. Know the vocabulary and basic concepts
  3. Command the basic workflow with a central repository
  4. Know how to deal with conflicting merges

Collaboration

- [...] the number of authors per paper [in one of the top-5 journals] has increased from 1.3 in 1970 to 2.3 in 2012.
  - Card and DellaVigna 2013
- Need effective means for working with others
- Same issue is to keep multiple machines in sync

Just work on Dropbox?

- Sensitive Data
- Simultaneous work: almost sure to get conflicted copies
- Fully automated synchronisation tools do not scale to complex (=real-world) workflows

Why Git?

- A (clear) protocol that supports complex workflows
- (Easy) merging of plain text files

Vocabulary and Basic Concepts

Schematic Git Workflow

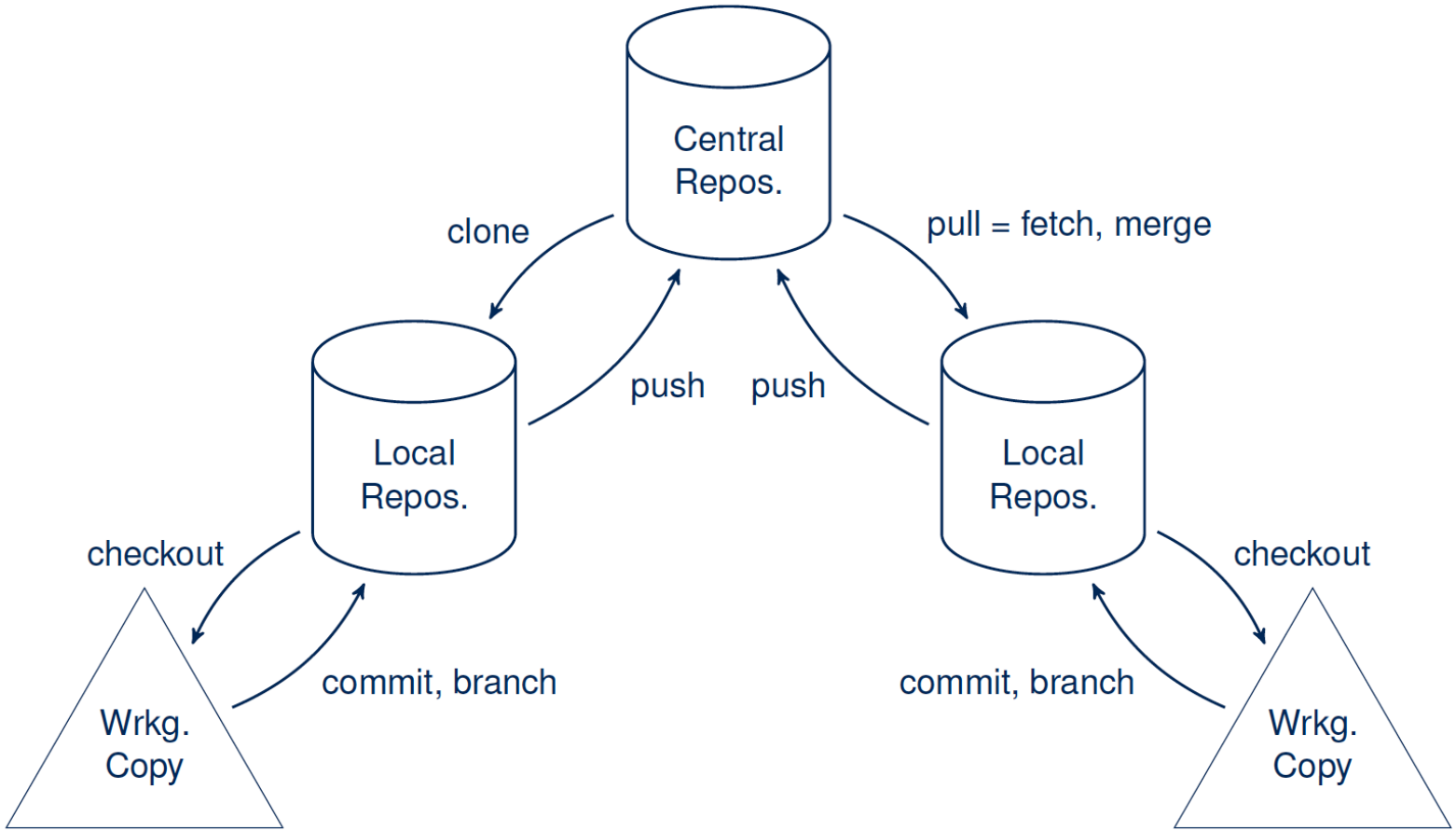


Figure 1: Git Workflow

Register and sign in

- Git hosting sercices e.g.:
  - github.com
  - gitlab.com
  - bitbucket.com
  - econgit.uzh.ch

Cloning a Repository

- move to the location you want to download the repository
- use the github.com URL given to you
- `$ git clone [URL]`

Creating a New Project on the Server

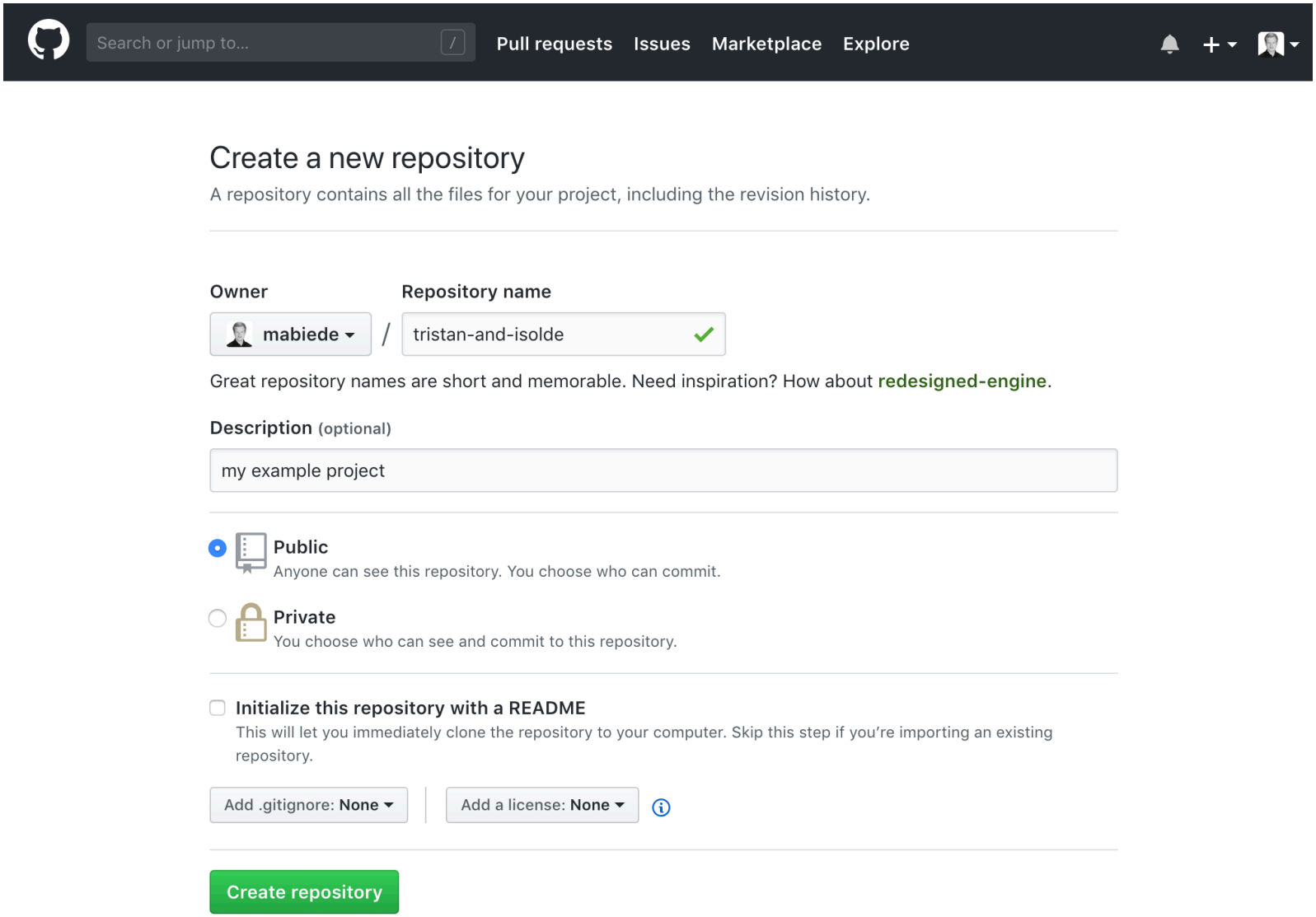


Figure 2: Creating a GitHub project

Making a Local Clone of the Project

- Go to the parent folder of where you want your project to live
- `$ git clone [URL]`

Adding a URL to an existing project

- get the HTTPS URL from GitHub
- go to your local repository using the terminal
- `git remote add origin [HTTPS]`
- check with `git remote -v`

HTTPS or SSH?

- We use HTTPS here because it does not require additional configuration. After the workshop you may want to set up SSH access, which is a bit more secure.
- Follow the instructions to generate a SSH key on GitHub.

Let's do it

- create a new project on GitHub.
- make a clone to your local machine

Basic Workflow

Form Groups of Two People

- one is Tristan, the other is Isolde
- Follow the workflow

Clone a respository

- Both Tirstan and Isolde clone the HTTPS of the `tristan-and-isolde` repository we will give you
- check out the contents: you should have downloaded two files:
  - `heart.py`
  - `.gitignore`
- have a look at them using **`less`** (press `q` to exit)
- you may want to keep the `.gitignore` as a template for future use

Tristan's heart script

- Tristan initiates a new project on GitHub. Call it `tristan-loves-isolde`.
- Add both `heart.py` and `.gitignore` to the project

heart.py

```python
import scipy
import matplotlib.pyplot as plt


def plot_heart():
    fig = plt.figure()

    x = scipy.linspace(-2, 2, 1000)
    y1 = scipy.sqrt(1 - (abs(x) - 1) ** 2)
    y2 = -3 * scipy.sqrt(1 - (abs(x) / 2) ** 0.5)

    plt.fill_between(x, y1, color='red')
    plt.fill_between(x, y2, color='red')
    plt.xlim([-2.5, 2.5])
    plt.text(
        0,
        -0.4,
        'Anybody?',
        fontsize=24,
        fontweight='bold',
        color='white',
        horizontalalignment='center'
    )

    return fig


```

Tristan's local Git preparations

- `$ git add heart.py`
- `$ git add .gitignore`
- or simply `$ git add .`
- `$ git status`

Tristan's local Git preparations II

- `$ git commit -m "First version of heart"`
- `$ git status`

First Push to the Remote Repository

- Push to the remote (central) repository
- `$ git push origin master`

Isolde Finds and Clones the Repository

- Isolde changes to the folder of her workspace where she wants to work.
- She clones the project from the **remote** repository
- `$ git clone [URL]`
- AGAIN: make sure you have the right credentials, i.e., you enter the username and password correctly

Isolde Changes the Code

- make some funny changes in the python script

She is happy, commits, and pushes

- `$ git status`
- if not added to the index: `$ git add heart.py`
- `$ git commit -am "Included Tristan in the heart's message."`
- `$ git push origin master`

Resolving Merge Conflicts

Tristan pulls the newest version from the central repository

- `$ git pull`
- and he does some changes...

At the same time...

- Isolde makes some conflicting changes. She commits and pushes them.
- Tristan now commits and intends to push...
- `$ git commit -am "fixed typo."`
- `$ git push origin master`
- ... but there will be an error message.

Error message

```
$ git push origin master

Username for 'https://git.yyy.de': tristan
Password for 'https://tristan@git.yyy.de':
To https://git.yyy.de/tristan-and-isolde.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to
'https://git.yyy.de/tristan-and-isolde.git'

Updates were rejected because the remote contains work that you do
not have locally. This is usually caused by another repository pushing
to the same ref. You may want to first merge the remote changes (e.g.,
'git pull') before pushing again.
See the 'Note about fast-forwards' in 'git push --help' for details.
```

Figure 4: Error message when conflicting files

What Has Happened?

- Read the message
- Git cannot make the change on the remote without losing commits, so it refuses to push.
- Usually this is caused by another user pushing to the same branch
- You can remedy this by pulling from the remote and resolve the conflict

Tristan Configures and Pulls the Remote

- The pull is a fetch followed by a merge
- Howver, there are conflicts
- `$ git pull origin master`

Tristan Solves the merge Conflicts

Git Conflict Resolver (Sublime Text Editor)

- A Sublime Text plugin to help you solve merge conflicts
- Commands
  - Find Next Conflict
  - Keep Ours
  - Keep Theirs
  - Keep Common Ancestor
  - Show Conflict Files

Git Conflict Resolver (Sublime Text Editor) II

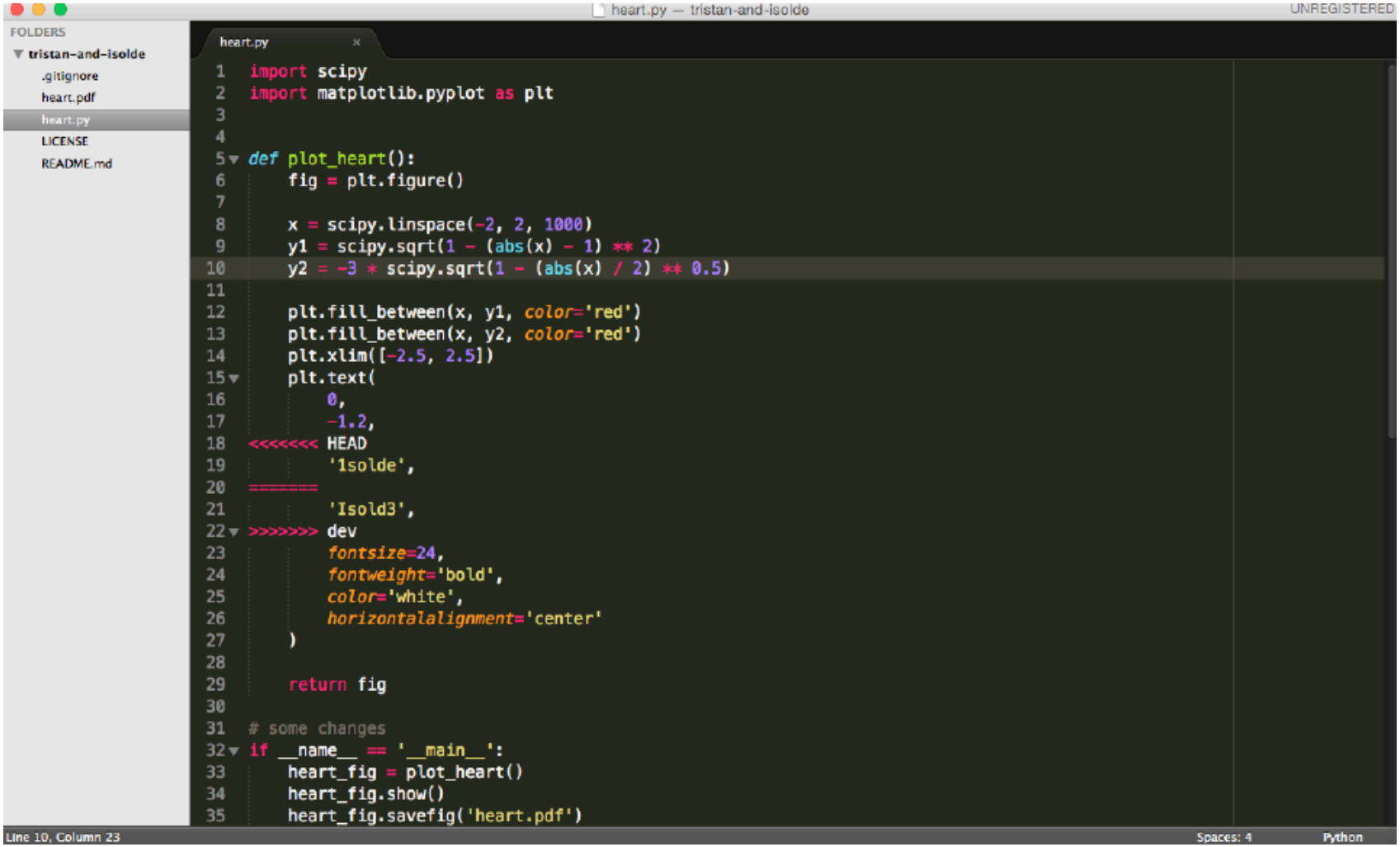
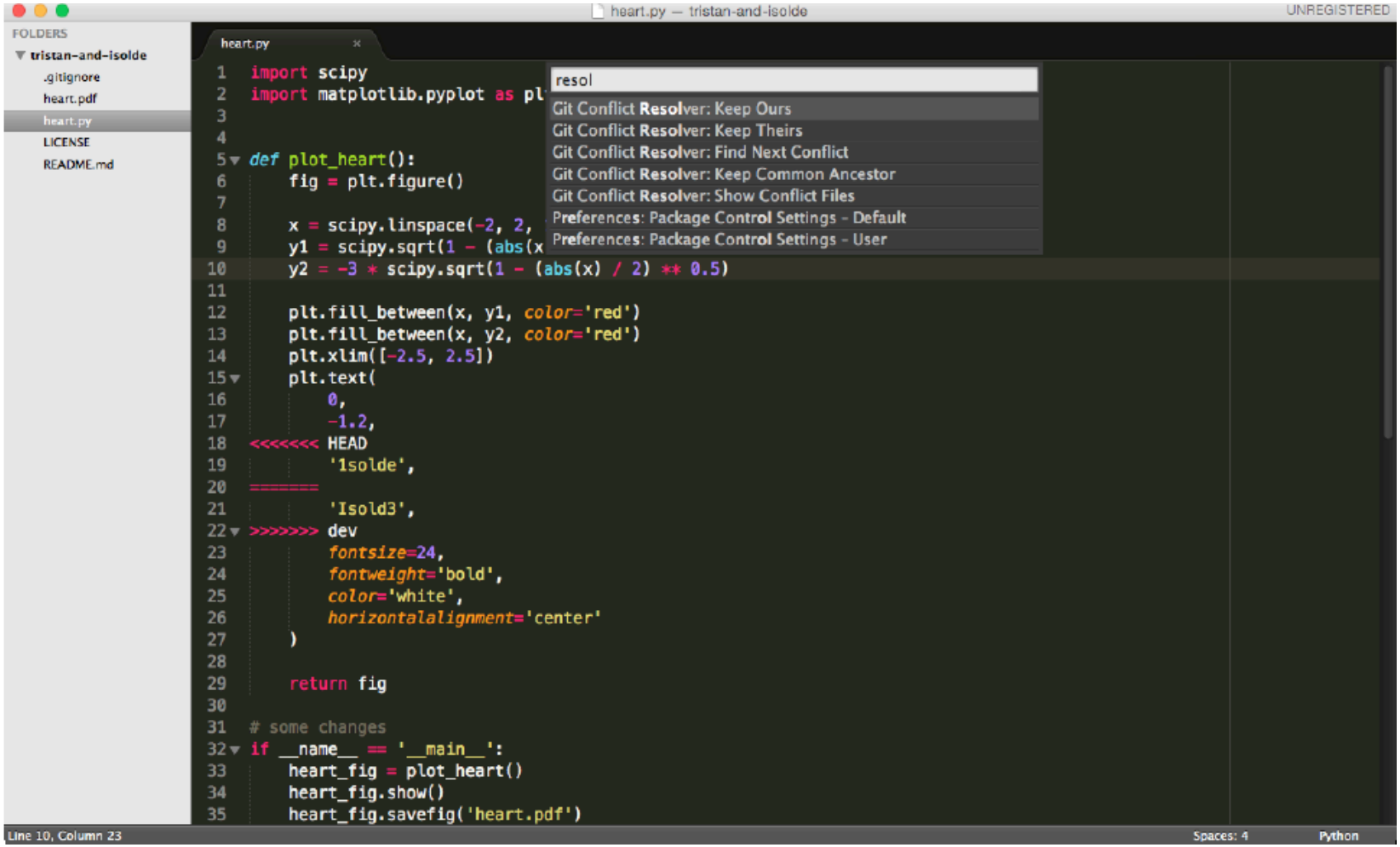


Figure 5: Resolve Conflict

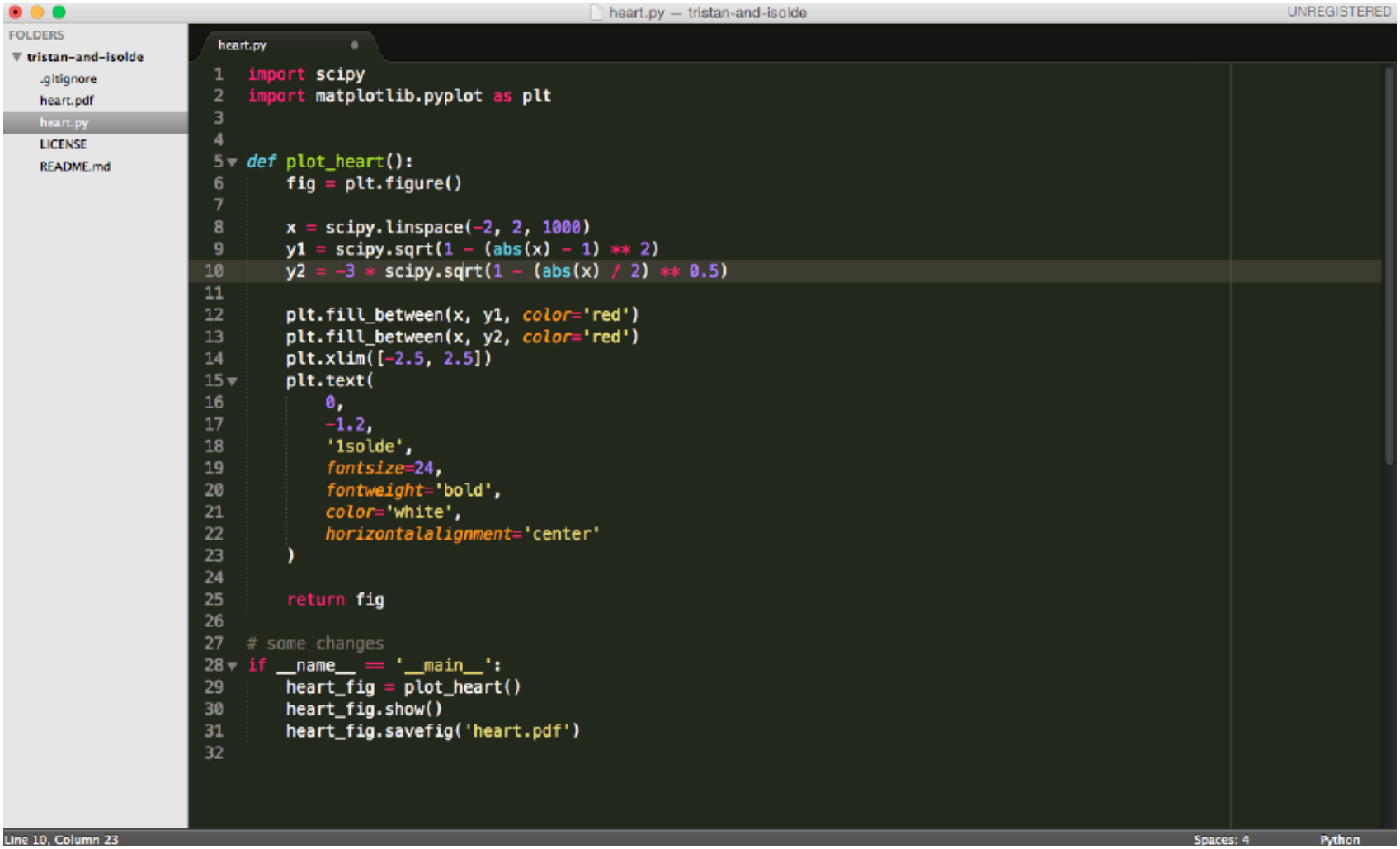
Git Conflict Resolver (Sublime Text Editor) III

- Use the plugin to choose from standardized optoins



Git Conflict Resolver (Sublime Text Editor) IV

- Sublime will adjust the code accordingly



Git Conflict Resolver (MS Visual Studio Code)

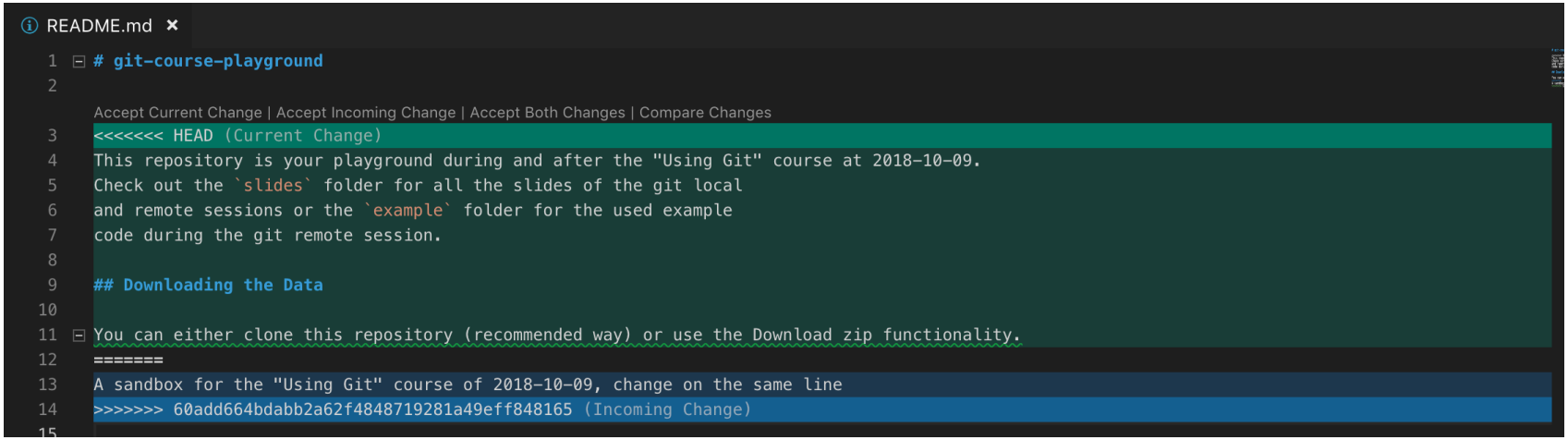


Figure 8: Resolve Conflict

Tristan Commits the Merged File

- `$ git commit status`
- `$ git add heart.py`
- `$ git commit -am "resovled merge conflict"`
- `$ git log`
- `$ git status`
- `$ git push`

Recommended Workflow in Teams

- Everybody has his or her own branch
- Frequent merges
- A master branch where only universally accepted changes enter
- Benefits:
  - Freedom to merge only when it is convenient
  - You can always push your changes upstream

Recommended Workflow in Teams II

- If you encounter merge conflicts frequently, it is a sign that something is wrong with the workflow in your project
  - not talking to co-authors as often as you should?
  - responsibilities not clearly assigned?
- Git helps you detect this

Acknowledgements

- This course is designed after and borrows a lot from:
  - Effective Programming Practices for Economists, a course by Hans-Martin von Gaudecker
  - Software Carpentry and Data Carpentry designed by Greg Wilson
  - Shotts, W.E. (2012). The Linux Command Line. San Francisco: No Starch Press.

Programming Practices Team

Programming Practices were updated by

- Marc Biedermann

and created by

- Lachlan Deer
- Adrian Etter
- Julian Langer
- Max Winkler

at the Department of Economics, University of Zurich. These slides were originally from the 2016 edition of Programming Practices for Economics Research.