

# 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

- ▶ [...] 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?

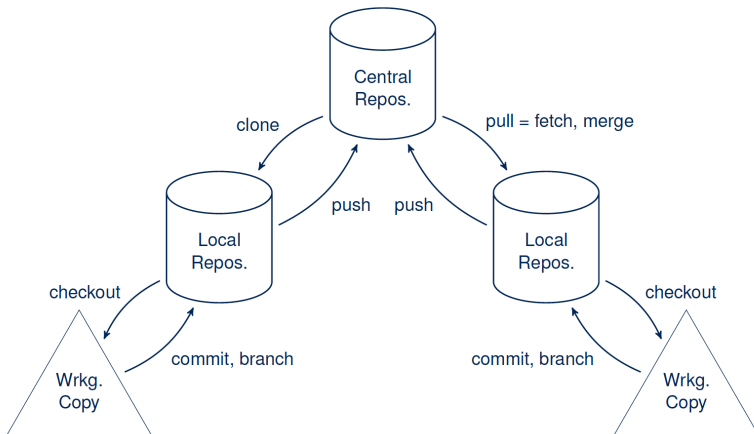
- ▶ Sensitive Data
- ▶ Simultaneous work: almost sure to get conflicted copies
- ▶ Fully automated synchronisation tools do not scale to complex (=real-world) workflows

# 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 services e.g.:
  - ▶ [github.com](https://github.com)
  - ▶ [gitlab.com](https://gitlab.com)
  - ▶ [bitbucket.com](https://bitbucket.com)
  - ▶ [eongit.uzh.ch](https://eongit.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

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



Repository name



Great repository names are short and memorable. Need inspiration? How about [redesigned-engine](#).

Description (optional)

my example project



**Public**

Anyone can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None**

Add a license: **None**



Create repository

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 repository

- ▶ 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
1 import scipy
2 import matplotlib.pyplot as plt
3
4
5 def plot_heart():
6     fig = plt.figure()
7
8     x = scipy.linspace(-2, 2, 1000)
9     y1 = scipy.sqrt(1 - (abs(x) - 1) ** 2)
10    y2 = -3 * scipy.sqrt(1 - (abs(x) / 2) ** 0.5)
11
12    plt.fill_between(x, y1, color='red')
13    plt.fill_between(x, y2, color='red')
14    plt.xlim([-2.5, 2.5])
15    plt.text(
16        0,
17        -0.4,
18        'Anybody?',
19        fontsize=24,
20        fontweight='bold',
21        color='white',
22        horizontalalignment='center'
23    )
24
25    return fig
26
27
```

# 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
- ▶ However, there are conflicts
- ▶ `$ git pull origin master`

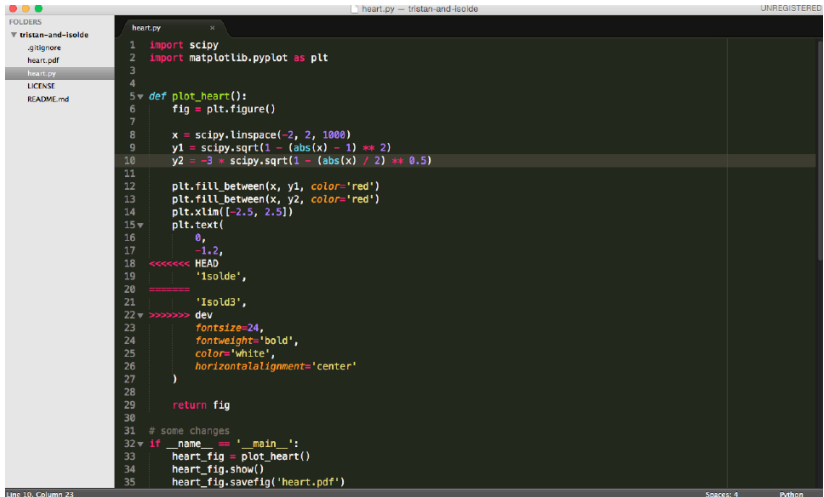
# Tristan Solves the merge Conflicts



# 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

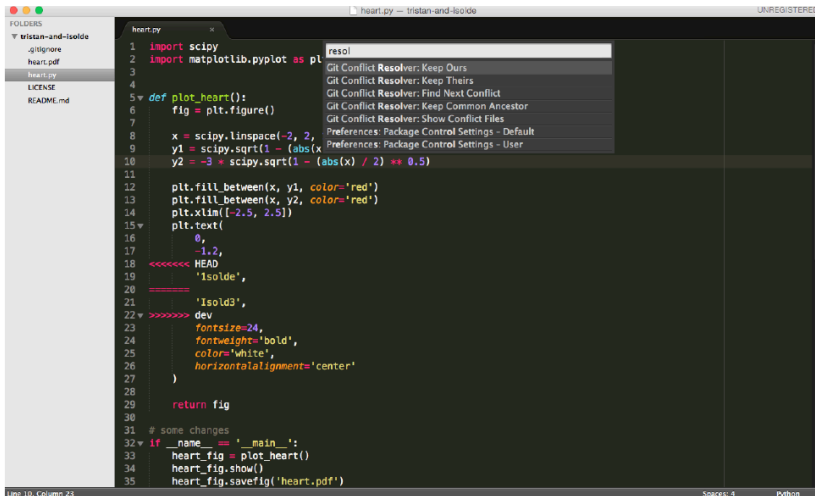


```
1 import scipy
2 import matplotlib.pyplot as plt
3
4
5 def plot_heart():
6     fig = plt.figure()
7
8     x = scipy.linspace(-2, 2, 1000)
9     y1 = scipy.sqrt(1 - (abs(x) - 1) ** 2)
10    y2 = -3 + scipy.sqrt(1 - (abs(x) / 2) ** 0.5)
11
12    plt.fill_between(x, y1, color='red')
13    plt.fill_between(x, y2, color='red')
14    plt.xlim([-2.5, 2.5])
15    plt.text(
16        0,
17        -1.2,
18        <<<<<< HEAD
19        'Isolde',
20        =====
21        'Isold3',
22    >>>>>> dev
23        fontsize=24,
24        fontweight='bold',
25        color='white',
26        horizontalalignment='center'
27    )
28
29    return fig
30
31 # some changes
32 if __name__ == '__main__':
33     heart_fig = plot_heart()
34     heart_fig.show()
35     heart_fig.savefig('heart.pdf')
```

Figure 5: Resolve Conflict

# Git Conflict Resolver (Sublime Text Editor) III

- Use the plugin to choose from standardized options



The screenshot shows the Sublime Text Editor interface with a file named 'heart.py' open. The editor is displaying Python code that uses SciPy and Matplotlib to plot a heart shape. A context menu is open over the code, showing options for resolving a Git conflict. The menu options are:

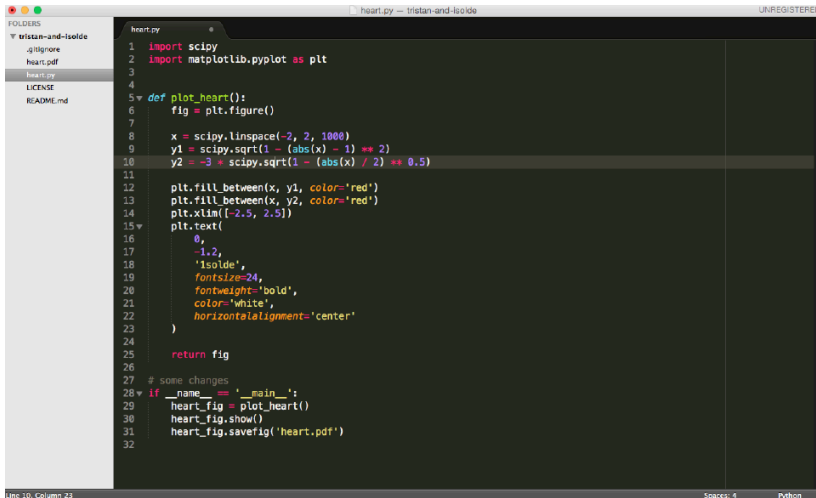
- Git Conflict Resolver: Keep Ours
- Git Conflict Resolver: Keep Theirs
- Git Conflict Resolver: Find Next Conflict
- Git Conflict Resolver: Keep Common Ancestor
- Git Conflict Resolver: Show Conflict Files
- Preferences: Package Control Settings - Default
- Preferences: Package Control Settings - User

The code in the background is as follows:

```
1 import scipy
2 import matplotlib.pyplot as plt
3
4
5 def plot_heart():
6     fig = plt.figure()
7
8     x = scipy.linspace(-2, 2,
9     y1 = scipy.sqrt(1 - (abs(x
10    y2 = -3 * scipy.sqrt(1 - (abs(x) / 2) ** 0.5)
11
12    plt.fill_between(x, y1, color='red')
13    plt.fill_between(x, y2, color='red')
14    plt.xlim([-2.5, 2.5])
15    plt.text(
16        0,
17        -1.2,
18        <<<<<< HEAD
19        'Isolde',
20
21        'Isold3',
22    >>>>> dev
23        fontsize=24,
24        fontweight='bold',
25        color='white',
26        horizontalalignment='center'
27    )
28
29    return fig
30
31 # some changes
32 if __name__ == '__main__':
33     heart_fig = plot_heart()
34     heart_fig.show()
35     heart_fig.savefig('heart.pdf')
```

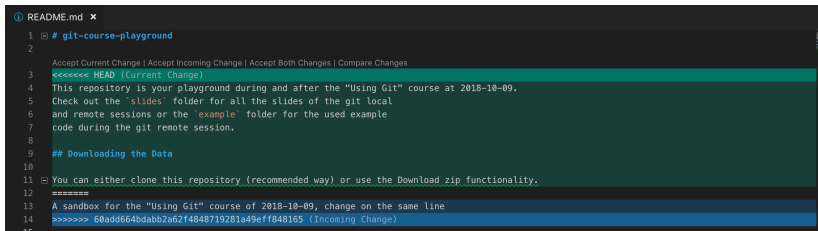
# Git Conflict Resolver (Sublime Text Editor) IV

- ▶ Sublime will adjust the code accordingly



```
1 import scipy
2 import matplotlib.pyplot as plt
3
4
5 def plot_heart():
6     fig = plt.figure()
7
8     x = scipy.linspace(-2, 2, 1000)
9     y1 = scipy.sqrt(1 - (abs(x) - 1) ** 2)
10    y2 = -3 + scipy.sqrt(1 - (abs(x) / 2) ** 0.5)
11
12    plt.fill_between(x, y1, color='red')
13    plt.fill_between(x, y2, color='red')
14    plt.xlim([-2.5, 2.5])
15    plt.text(
16        0,
17        -1.2,
18        'Isolde',
19        fontsize=24,
20        fontweight='bold',
21        color='white',
22        horizontalalignment='center'
23    )
24
25    return fig
26
27 # some changes
28 if __name__ == '__main__':
29     heart_fig = plot_heart()
30     heart_fig.show()
31     heart_fig.savefig('heart.pdf')
32
```

# Git Conflict Resolver (MS Visual Studio Code)



The screenshot shows the Git Conflict Resolver interface in VS Code for a file named README.md. The interface is divided into three horizontal sections. The top section, labeled 'HEAD (Current Change)', contains the text: 'This repository is your playground during and after the "Using Git" course at 2018-10-09. Check out the 'slides' folder for all the slides of the git local and remote sessions or the 'example' folder for the used example code during the git remote session.' The bottom section, labeled '60add664bdabb2a62f4848719281a49eff848165 (Incoming Change)', contains the text: 'A sandbox for the "Using Git" course of 2018-10-09, change on the same line'. The middle section, labeled 'You can either clone this repository (recommended way) or use the Download zip functionality.', is currently empty. The interface also includes a toolbar at the top with options like 'Accept Current Change', 'Accept Incoming Change', 'Accept Both Changes', and 'Compare Changes'.

```
1  # git-course-playground
2
3  Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
4  ===== HEAD (Current Change)
5  This repository is your playground during and after the "Using Git" course at 2018-10-09.
6  Check out the 'slides' folder for all the slides of the git local
7  and remote sessions or the 'example' folder for the used example
8  code during the git remote session.
9
10 ## Downloading the Data
11 You can either clone this repository (recommended way) or use the Download zip functionality.
12 =====
13 A sandbox for the "Using Git" course of 2018-10-09, change on the same line
14 >>>>>> 60add664bdabb2a62f4848719281a49eff848165 (Incoming Change)
15
```

**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.
- ▶ The course material from above sources is made available under a Creative Commons Attribution License, as is this courses material.

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.