Version Control II

Introduction to git – git-remote

Programming Practices for Economics Research

Department of Economics, Univeristy of Zurich

Fall 2017

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

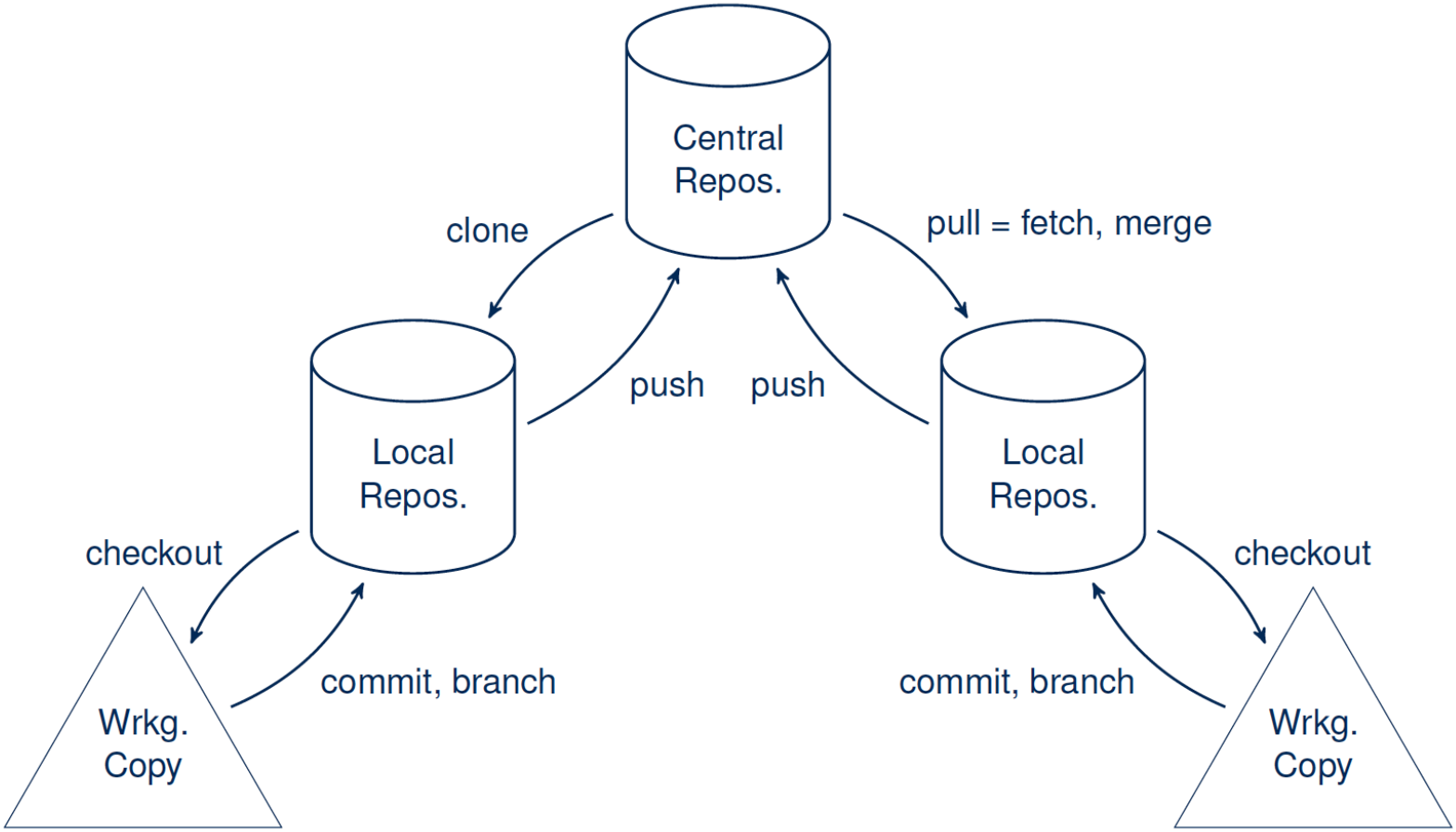

Figure 1: Git Workflow

econgit.uzh.ch and github.com

- register and sign in

Cloning a Repository

- move to he location you want to download the repository
- use the econgit.uzh.ch URL given to you
- make sure you configure Git with your @uzh.ch email addresses when you use econgit.uzh.ch
- `$ git clone [URL]`

Creating a New Project on the Server

econgit.uzh.ch or github

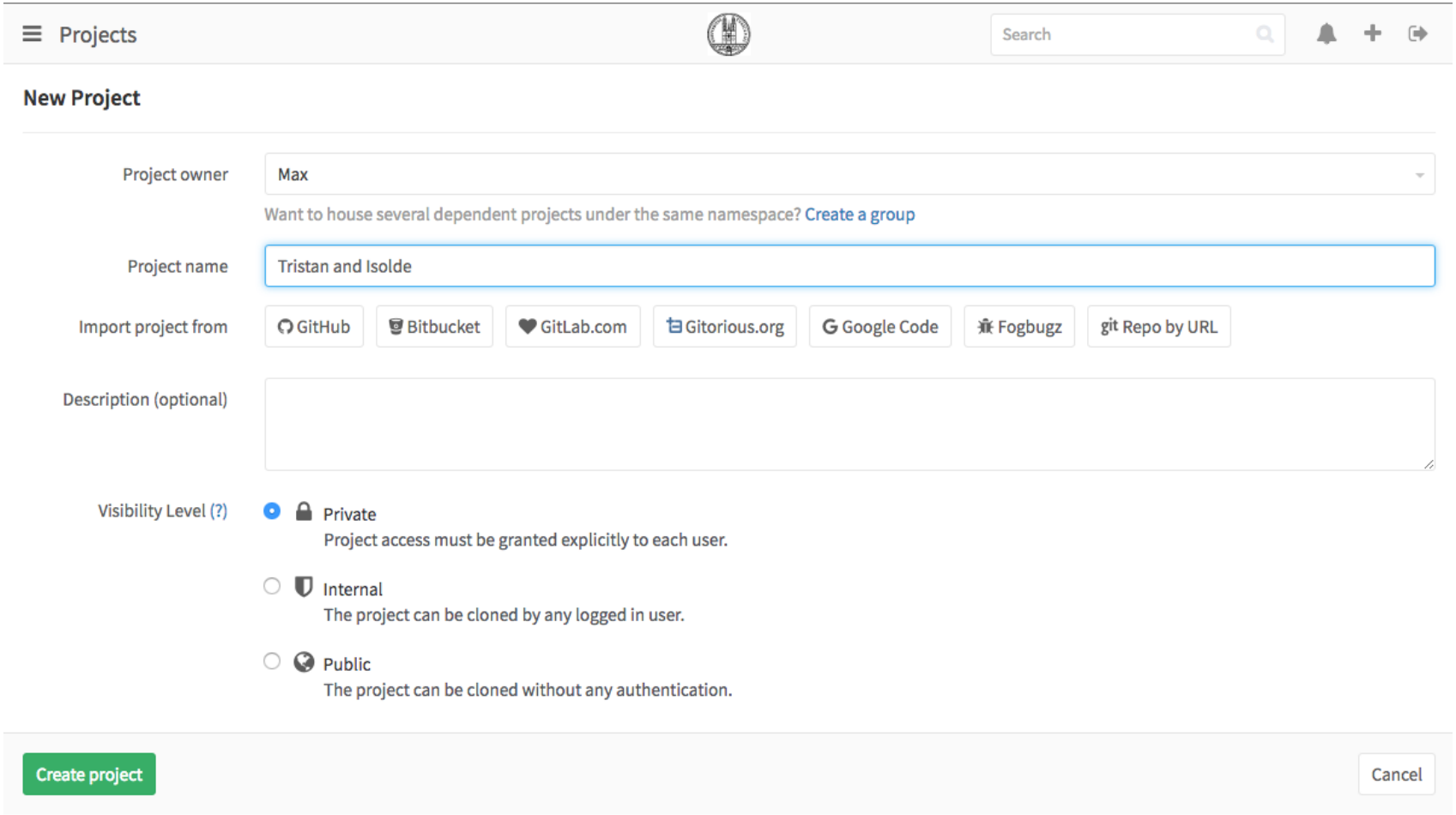

Figure 2: Creating a econgit.uzh.ch project

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 or econgit
- 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 or econgit.

Let's do it

- create a new project on GitHub or econgit.
- 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 **`subl`**
- you may want to keep the `.gitignore` as a template for future use

Tristan's heart script

- Tristan initiates a new project on GitHub or econgit. 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

- `$ 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 (Atom Editor)

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

Git Conflict Resolver (Atom Editor)

heart.py — tristan-and-isolde

UNREGISTERED

FOLDERS

- tristan-and-isolde
  - .gitignore
  - heart.pdf
  - heart.py
  - LICENSE
  - README.md

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         -1.2,
18 <<<<<<< HEAD
19         '1solde',
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')
```

Line 10, Column 23 Spaces: 4 Python

Figure 5: Error message when conflicting files

Git Conflict Resolver (Atom Editor)

- Use the plugin to choose from standardized optoins

heart.py — tristan-and-isolde

UNREGISTERED

FOLDERS

- tristan-and-isolde
  - .gitignore
  - heart.pdf
  - heart.py
  - LICENSE
  - README.md

heart.py

```
1  import scipy
2  import matplotlib.pyplot as pl
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         '1solde',
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')
```

resol

- 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

Line 10, Column 23 | Spaces: 4 | Python

Git Conflict Resolver (Atom Editor)

- Sublime will adjust the code accordingly

heart.py — tristan-and-isolde

UNREGISTERED

FOLDERS

- tristan-and-isolde
  - .gitignore
  - heart.pdf
  - heart.py
  - LICENSE
  - README.md

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,
        -1.2,
        '1solde',
        fontsize=24,
        fontweight='bold',
        color='white',
        horizontalalignment='center'
    )

    return fig

# some changes
if __name__ == '__main__':
    heart_fig = plot_heart()
    heart_fig.show()
    heart_fig.savefig('heart.pdf')
```

Line 10, Column 23 Spaces: 4 Python

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

- 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 Team

Programming Practices for Economics Research was created by

```
* Lachlan Deer
* Adrian Etter
* Julian Langer
* Max Winkler
```

at the Department of Economics, University of Zurich. These slides are from the 2017 edition.