Git Local

Dept of Economics, Univeristy of Zurich

Programming Practices

October 8, 2018

Learning Objectives

- At the end of the session you will be able to:
  1. Convey the advantages of Version Control Systems
  2. Understand the vocabulary of Git
  3. Work with Git on you computer
  4. Use branches and merge work streams
  5. Know where to read up advanced stuff

Why Git?

Here Is the Problem

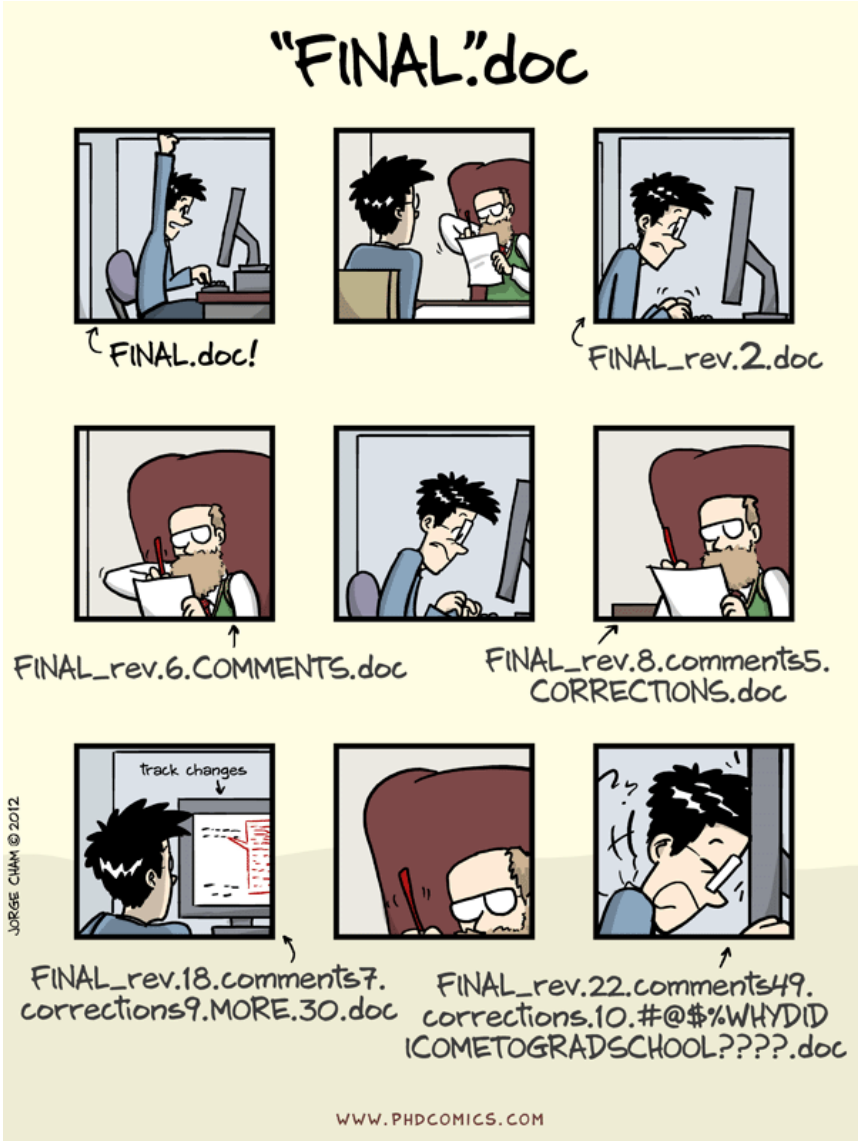


Figure 1: Final Doc

Ad-hoc solutions

- Save stuff regularly:
  - my-project
  - my-project-v1
  - my-project-v2
  - my-project-v3
  - my-project-v4
  - ...
- it is very likely that you get lost and miss important stuff

Use Git

- Git is a Version Control System.
- A Git project is represented by a *repository*, which contains the complete history of the project from its inception.
- A repository in turn consists of a set of individual snapshots of project content —- collections of files and directories —- called *commits.*
- The set of all commits in a repository, connected by lines indicating their parent commits, forms a picture called the repository *commit graph*.

Use Git II

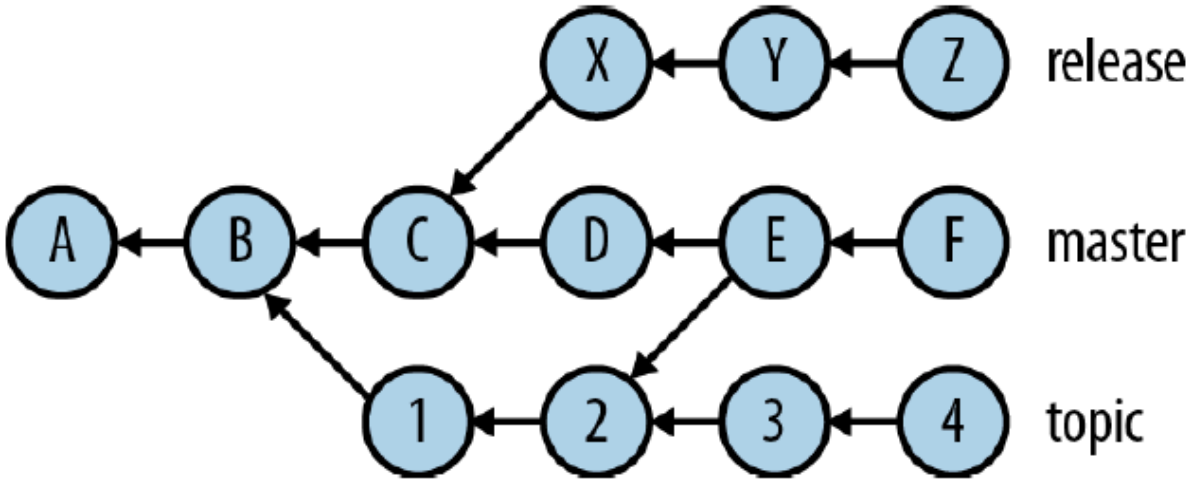


Figure 2: Commit Graph

Use Git III

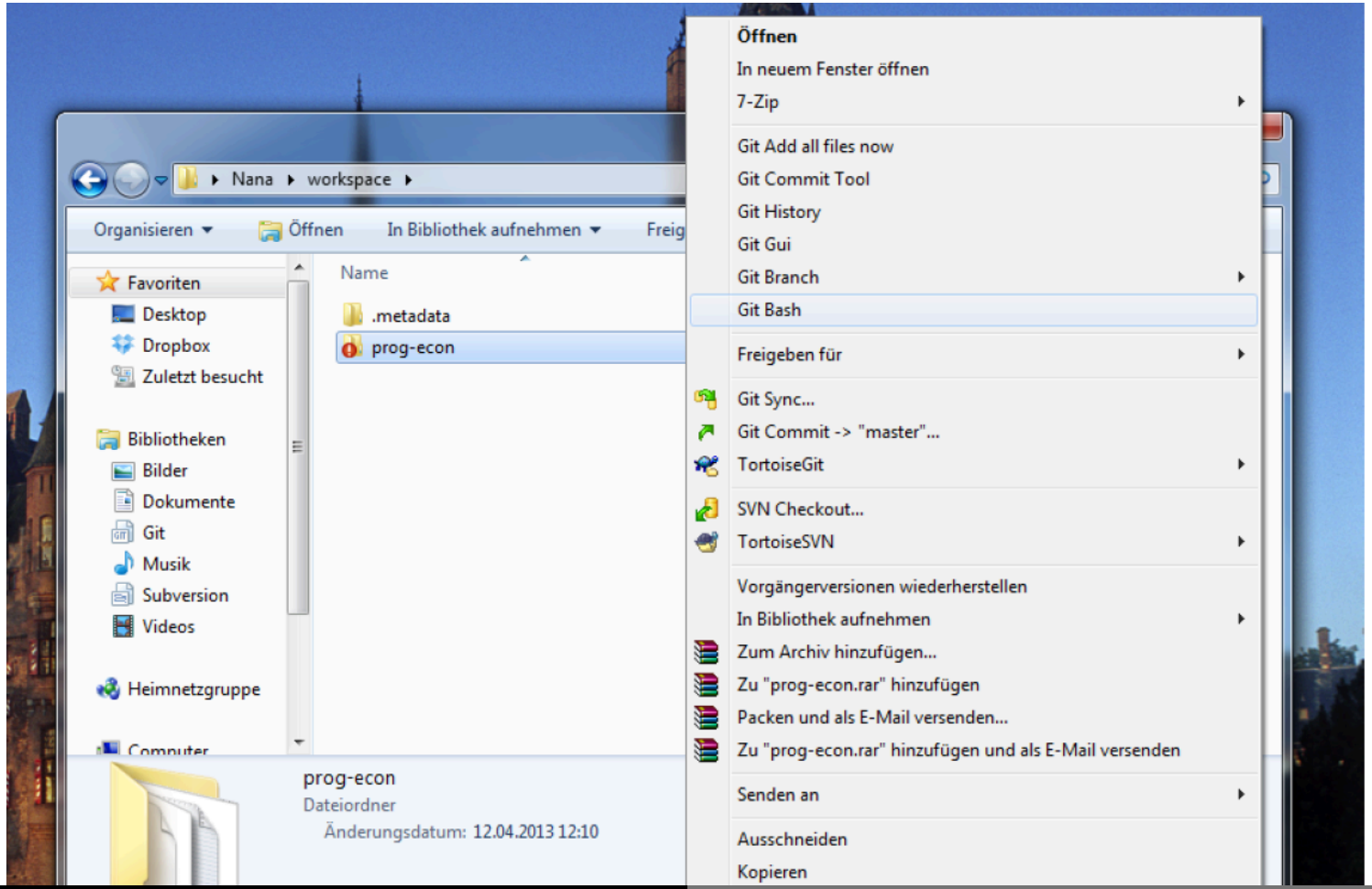
- The labels on the right side of the previous picture —- master, topic, and release —- denote *branches.*
- The branch is defined as the collection of all commits that are reachable from the tip – the latest commit of a branch – by following the parent arrows backward along the history.

Use Git IV

- There are two contexts in which version control is useful: private and public.
  - When working on your own, it's useful to commit *early and often*, so that you can explore different ideas and make changes freely without worrying about recovering earlier work.
  - Once you go public, it's important that others will understand your progress and your commits.

Use Git Bash on Windows

- Use cygwin
- Navigate to your project's folder, right-click, and select "Git Bash"



Git on Mac/Linux Terminal

- Open a terminal in /Applications/Utilities and get started

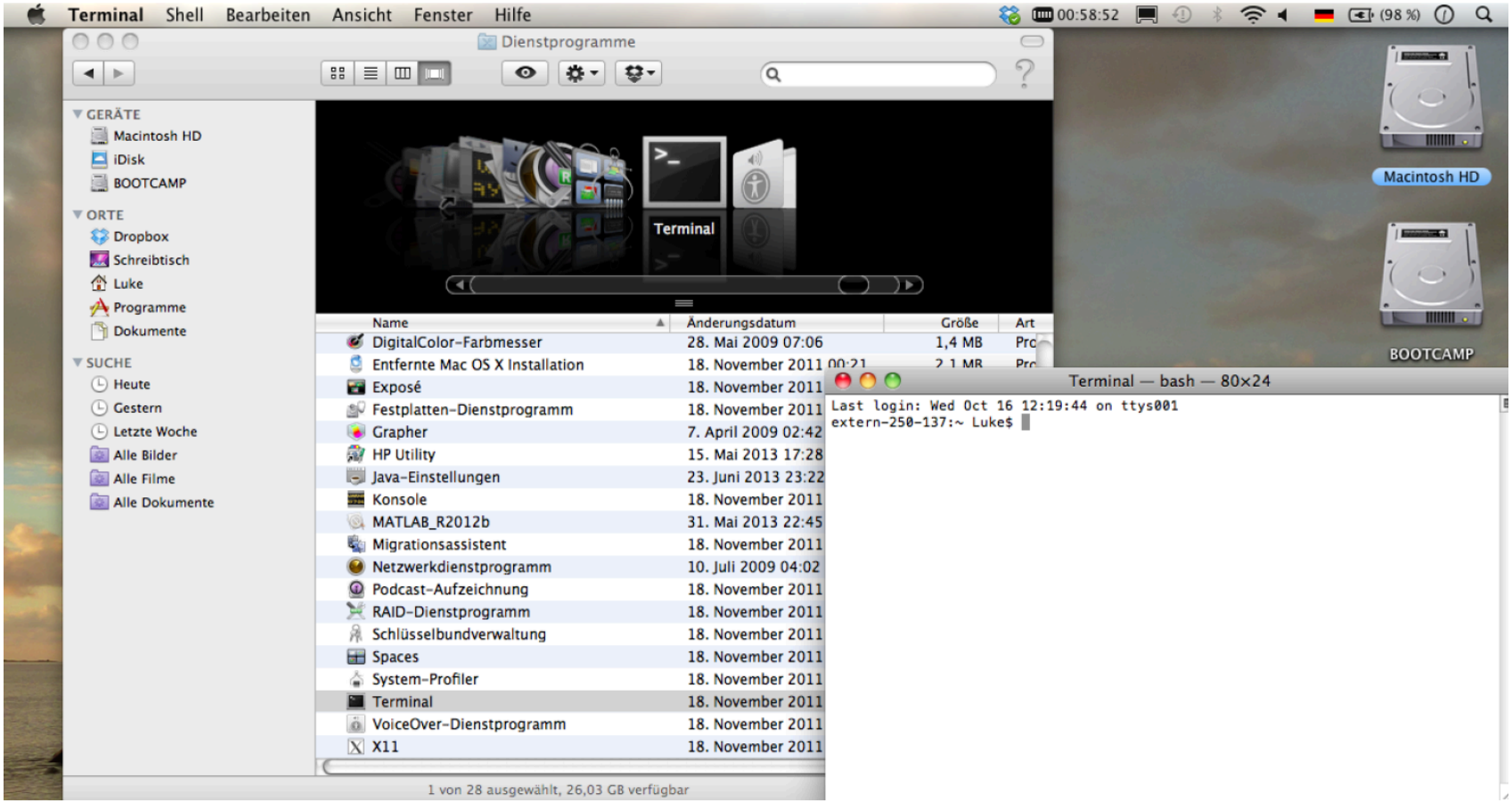


Figure 4: Git on Mac

Git on *cluster.pelkmanslab.org*

- *(For people in the Pelkmans Lab)*
- You can run all commands in this course from the Lab cluster:
  - SSH into the cluster:
    - **`ssh username@cluster.pelkmanslab.org`**
  - Run all `git` commands at the cluster prompt:
    - **`git help`**

The Basic Workflow

Set up your Git Credentials

- Eventually, a big advantage of Git will be that it tracks who did what and when
- So first tell Git who you are and how you may be reached
- `$ git config --global user.name "First Last"`
- `$ git config --global user.email "first.last@econ.uzh.ch"`
- use the `--local` option if you want to define a different name or an email address for the current repository.

Creating a New Local Repository

- `$ git init`
  - create a new **.git** directory in the current working directory
- `$ git status`

The Index or Staging Area

- move or create a file in your working directory
- `$ git add [somefile]`
  - add some file to the Git index
- `$ git status`

Importing an Existing Project

- `$ git init`
- `$ git add .`
- `$ git status`

In action...

- Create your first Git repository
- create a new file using `nano newfile.txt`, write something and save it
- type `git status`
- add it to the index
- confirm with `git status`

Keep Only Source Files under Version Control

- An output file, i.e., a pdf file, changes every time you recreate the file, even if there are no material changes to the file
  - there will be many fake changes of the repository
  - the repository size will explode
- Hence, keep only sources under VC!
  - Original data and source code from statistics programs, LATEX sources, etc.

Ignoring Output and Nuisance Files

- In large projects, it become impossible to manually select files to be added to the index.
- Hence, specify patterns to be ignored in a file called `.gitignore`, which lives in the project root.
- Use template `.gitignore` files
  - `$ git add .gitignore -f`
- It is still possible to manually add files that are ignored.

The First Commit

- Commit to the local repository with a meaningful message
- `$ git commit -m "Initial commit."`
- `$ git status`

Let's Do It

- Make Your First Commit

The Second Commit

- Work on the files you added to the index.
- `$ git commit -am "Changes XZY."`
  - commit all changes to the local repository. the -a option adds all tracked, modified files to the index before committing and commits changed and deleted files, but not new ones.
- use short and meaningful messages.
- `$ git status`

The Third Commit

- Make some changes in your file
- type `$ git diff` to see the changes you made since your last commit
- if you are happy, commit your progress

Changing The Index

- `$ git add [filename]`
- check with `$ git diff --staged`
- `$ git commit -am "Changes XZY."`
- `$ git status`

Changing The Index II

- `$ git add -u`
  - include all files in the current index, except new ones
- `$ git add -A`
  - include all files in the working tree, including new files.
- `$ git rm [filename]`
  - delete the file from the index **and delete the working file**
- `$ git mv [oldname] [newname]`
  - rename the file
- `$ git reset`
  - reset the index to match the current commit
- `$ git commit -am "Changes XZY." --amend`
  - discard the previous commit and put a new one in its place to include new files (-a does not include new files).
- `$ git status`

View the Log of Commits

- `$ git log`
  - show the history of commits
- `$ git log -g`
  - shows the history of operations, including ammended commits

Discarding the Last Commit

- `$ git reset HEAD~`
  - move the branch back to one commit, discarding the latest one
  - you can still recover the latest one using `$ git log -g`
- `$ git reset HEAD~3`
  - discard any number of consecutive commits; here, go back to the fourth commit (0 is the current commit)

Undoing Commit

- `$ git revert [HASH]`
  - use `git log` to get the HASH
  - make a new commit undoing the earlier commit's change
  - you can still recover the latest one using `$ git log -g`

Restore an Old Commit

- `$ git checkout HEAD [yourfile.txt]`
  - recovers the last saved commit
- `$ git checkout [HASH] [yourfile.txt]`
  - recovers any previous commmit according to its hash. Recover the commit number that captures the state of your repository *before* the change you are trying to undo.

NOTE: if you forget `[yourfile.txt]` in that command, git will tell you that "You are in 'detached HEAD' state." In this state, you shouldn't make any changes. You can fix this by reattaching your head using `git checkout master`.

Let's Do It

- Make some changes to your file. Use `$ git diff`
- Add more files to the index
- Make some commits
- Check out the history of your commits
- move back and forth on your branch
- undo some changes

Some Final Remarks

- When everything stops working...
- ... don't panic!!!
  - Situation from the last commit is always in the repository
  - So be sure to commit frequently
  - Always solve problems immediately so that you won't loose much information should you have to go back
- Won't happen much now – but things become a bit tricky once we use Git for collaboration

Where to Find Help

Books

- Here are two good books to look up stuff:
  - Loeliger and McCullough (2012)
  - Silverman (2013)

What you should have taken away...

1. why is it cool to use Git?
2. do you understand the vocabulary: repository, branches, commits?
3. can you track your own work?
4. can you set up new branches to experiment and merge it to your master when you are happy?
5. Do you know where to look up stuff if you want to know more?

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.

References

Loeliger, Jon, and Matthew McCullough. 2012. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. " O'Reilly Media, Inc."

Silverman, Richard E. 2013. *Git Pocket Guide*. " O'Reilly Media, Inc."