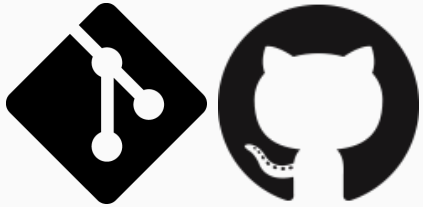
Git for economists



Nicolas Reigl (TalTech / Eesti Pank) & Heili Hein (TalTech)

19 January 2021

Introduction

- **WHY?**
  - Good to stay in contact with other researchers, share skills
  - Git has become a core skill in many professions
- **WHAT?**
  - 1.5 hours introduction to Git & GitHub
  - Split between a **presentation** and a **live-tutorial**
  - Ask questions whenever you feel like it!

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No 952574.

Prologue

What is Git?

Git is officially defined as a *distributed version control system* (VCS)

- A system that tracks changes to our project files over time
  - record project changes and go back to specific versions of tracked files, at any given point in time
- System can be used by many people to collaborate efficiently
  - each developer can have their own version of the project, distributed on their computer
  - later the individual versions of the project can be merged and adapted to the main version of the project

What is GitHub?

- GitHub is an online platform built for collaborative software development that uses the git protocol
- Git and GitHub are distinct things
- GitHub provides free unlimited public repos
- Paid subscriptions for a wide range of services
    - University affiliation can get you more free stuff

Alternatives

- Gitlab
- Bitbucket
- Host your private git server

Why learn Git as an economist?

Common tasks for modern day economists:

- Statistical programming
- Collaborate (interdisciplinary) (on open source projects)
- *New*: Teach online

Economists work with code

But it seems we are not adopting many important tools from the experts on code

Why?

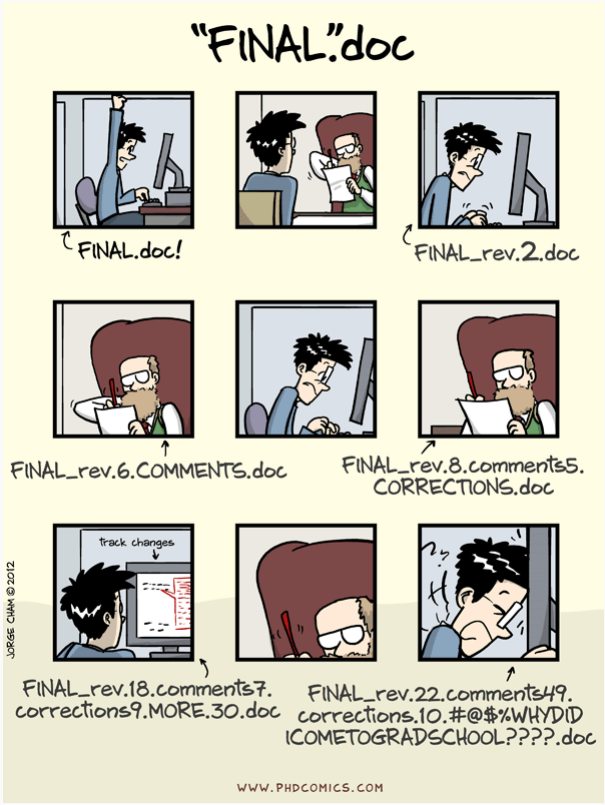
- Learning new things is costly
- Rethink our workflows
- It's simply not thought!

Who uses Git?

- Institutions (NY FED)
- Econ Projects (QuantEcon Project)
- University Departments (ScPo Department of Economics)
- Econ software platforms (Dynare)
- Econometric packages (FixEst Package)
- Individual economists (Victor Chernozhukov)

Why use version control (VC)?

Do you have files like final.txt, final_revised.txt, final_revised2.txt, final_revised2_revised.txt?



'Piled Higher and Deeper' by Jorge Cham www.phdcomics.com

VC with Git

- Keep track of changes to text files line-by-line over time
- Keep track of what you changed between different versions
- Branching and merging: Experiment safely; integrate when done (robustness checks, revisions)
- Revert any change if needed
- Back up and distribute copies of files
- Co-authored research is common: version control allows easier tracking of collaboration and changes
    - Work offline and asynchronously
    - Collaborate remotely, centralized, in real time

VC visualized: changes

What is version control? https://uidaholib.GitHub.io/get-git/1why.html

VC visualized: branches

What is version control? https://uidaholib.GitHub.io/get-git/1why.html

Reproducibility

- Many journals require data & code with submissions
- Proper version control makes reproduction easier
  - GiHub: DOI integration, off-the-shelf-licensing

> VC and open sourcing of your code operationalises the idea of open science and reproducability

Grant McDermont, Big data in Economics

Teaching

- Update and test new teaching material before you make it public
- Easier to upload and distribute new teaching material (when using GitHub)
- Facilitate collaborative student projects
- Important industry skill

Learning Git(Hub)

Resources

- Git docs
- Atlasian git tutorial
- Software Carpentry git tutorial
- QuantEcon git tutorial
- Flight rules for Git

Important Git concepts

- **repository (repo)**: a collection of files and their history
- **commit**: a snapshot of your files at a given time
- **staging area**: where you list files that will go into your next commit
- **(local) repository**: where the commits are stored; i.e., it contains the full history of previous versions of the files in the repository, and relevant metadata
- **(remote) repository**: a version of the repository hosted elsewhere
- **gitignore**: a file that declares what should **NOT** be tracked
- **main (or master)**: main version of the code
- **branches**: sequence of commits which can trace *alternative* history and versions of your code in the repo
- **HEAD**: pointer to branch ("Where your next commit would go")

Basic Git shell commands

- `git init`: start a new repository in the working directory
- `git status`: query the state of your directory
- `git add`: add new or modified files to the staging area
  - `git add .`: adds all file **excluding** those defined in the .gitignore
  - `git add -u`: updates the index of an already staged file
  - `git add -A`: adds all file **including** those defined in the .gitignore
- `git commit`: commit changes in the index
- `git branch`: create a branch or list branches
- `git checkout`: switch within a branch or between branches
- `git diff`: show what changed wrt. the last commit or index
- `git log`: show the commit history
- `git push`: send your local changes to a remote server
- `git pull` (`git fetch`; `git merge`): get last changes from a remote
- `git clone`: create local copy of public source project

Setup

- Download Git
- Install and if necessary set path
- Terminal editor
- Configure your name and & email

```
git config --global user.name "Your name"
git config --global user.email "your@email.com"
```

Tips:

```
git config --global core.editor "code --wait" # sets VS Code as core editor
git config --global -e
```

Line ending configuration:

```
git config --global core.autocrlf true # Windows
git config --global core.autocrlf input # OSX
```

Collaborate with others

- Create GitHub account
- Star, clone, and fork other projects
- GitHub Issues[1] are a great way to collaborate (and learn new things)

[1] If you spot any mistakes in this presentation, please file an issue here!

Editors

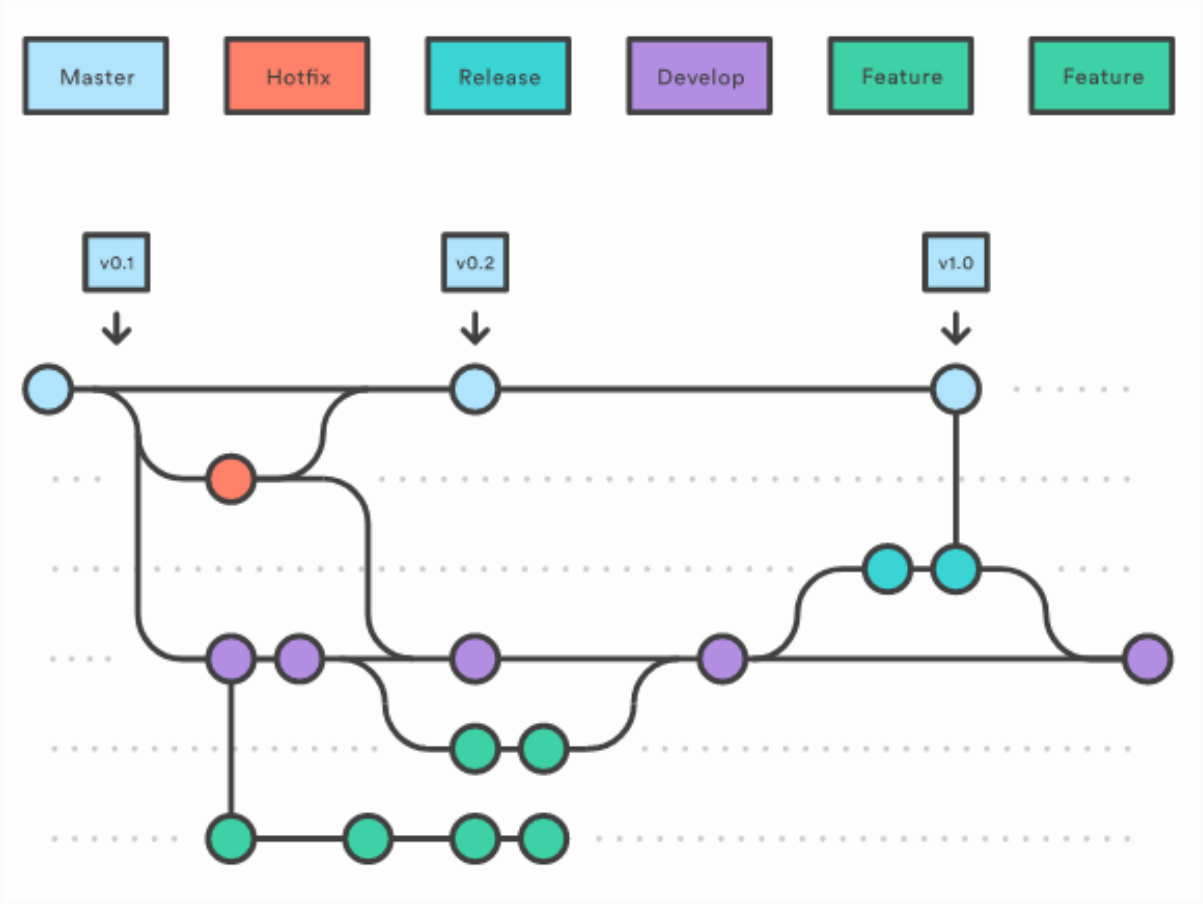
Command line based editors:

- Terminal with git capable shell
- Git shell

Graphical User Interfaces (GUIs)

- RStudio
- VSCode
- Gitkraken

Git workflows



A Review of Git Workflow Management https://medium.com/@hengfeng/a-review-of-git-workflow-management-7f9fbebd9370

Basic workflow

1. Create an empty folder
2. Initiate repository (`git init`)
3. Add files & annotate them
4. Add the files to the staging area (`git add`)
5. Commit the files (`git commit`)

Personal remote workflow (option 1)

- Steps 1-5
- Create an *empty* repo on GitHub
- Link the GitHub remote to your local repo
- Push from the local repo to the remote

Pros:

- Decide when to push to GitHub
- Language specific project Environments (Julia Environments)

Cons:

- Not a good workflow for non-empty remotes
- In case of non-language environment -> no README.md, no license, no .gitignore

Always pull from the upstream repo *before* you push any changes!

Collaborating workflow

- Fork a public repo on GitHub
- Clone the *forked* repo to local empty folder
- (*Optional: create a dev branch*)
- Work on the forked repo (*main or dev and then merge back to main*)
- Push back to the personal fork
- Create a pull request
- Review process of pull request

Merge conflicts

- Collaborators work on separated isolated branches to avoid *live* merge conflicts
- `git merge` command's primary function is to combine separate branches and resolve any conflicting edits
- Most of the time Git will figure out how to resolve conflicting changes

Merge conflicts (cont.)

What if Git can not figure out how to resolve a conflicting change?

```
# README
Some text here.
<<<<<<< HEAD
Text added by Partner 2.
=======
Text added by Partner 1.
>>>>>>> 814e09178910383c128045ce67a58c9c1df3f558.
More text here.
```

- `<<<<<<< HEAD` Indicates the start of the merge conflict.
- `=======` Indicates the break point used for comparison.
- `>>>>>>> <long string>` Indicates the end of the lines that had a merge conflict.

Merge conflicts (cont.)

Fixing these conflicts is a simple matter of (manually) editing the conflicting file[2].

- Delete the lines of the text that you don't want
- Then, delete the special Git merge conflict symbols

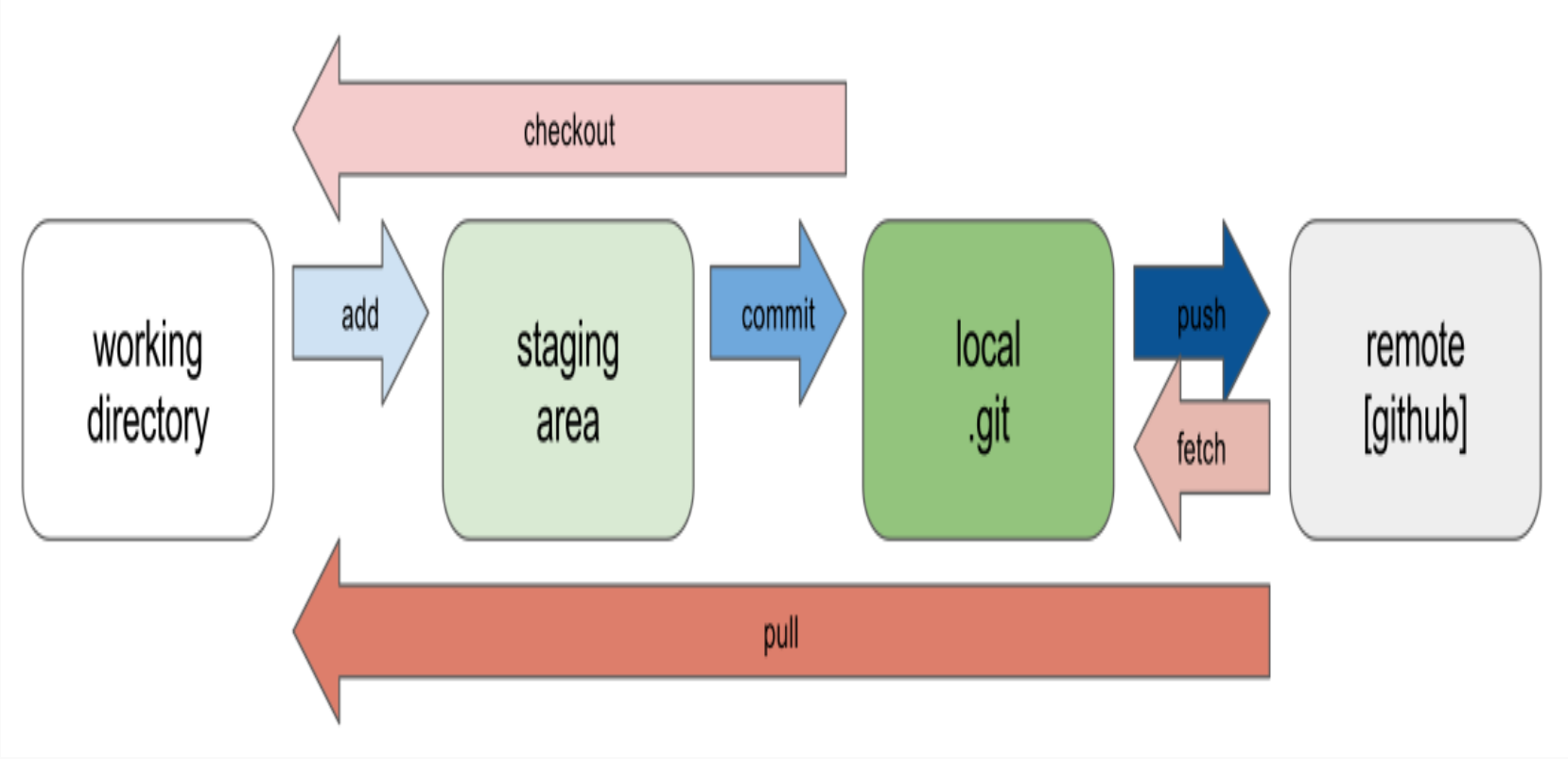
Once that's done, you should be able to stage, commit, pull and finally push your changes to the GitHub repo without any errors.

Caveats

- P1 gets to decide what to keep because they fixed the merge conflict
- However, the full commit history is preserved, so P2 can always recover their changes if desired

[2] If you know *VIM* you can use the terminal to edited the conflicting file but for more complicated merge conflicts, switching to a GUI can proove helpful. For example, VS Code offers a powerful merge conflict resolver interface. See Tips: Diff and merge tools in the Appendix.

Workflow visualized



Workflow https://uidaholib.GitHub.io/get-git/3workflow.html

Live tutorial

Questions

Appendix

Tips: terminal and GUIs

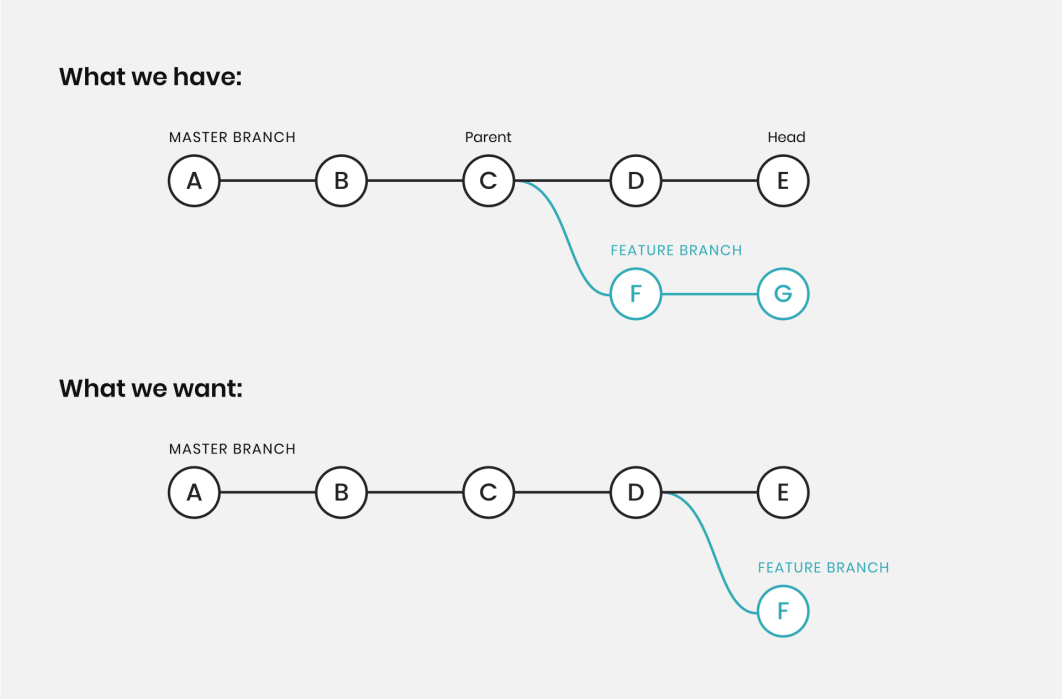
- Master the shell
- git with r: Jenny Bryan's amazing guide

Tips: working with branches

- `git fecht`, `git merge` is better than `git pull` if you face *difficult* merges
  - `git fecht`: tells the remote to retrieve the latest meta-data from the remote but does not transfer any files
    - Get fresh view on all the things that happened on the remote
  - `git pull`: retrieves the metadata and a copy of your files from the remote
  - `git pull`: tries to merge remote changes with your local ones, so a *merge conflict* can happen
  - Don't use `git pull` if you have a *dirty* local repo

Tips: working with branches (cont.)

`git rebase`: Another powerful Git utilities that specializes in integrating changes from one branch onto another (similar to `git merge`). However, do **NOT** `git rebase` public open source repos.



Regain Control of Branches with Git Rebase --onto https://www.headway.io/blog/regain-control-of-branches-with-git-rebase-onto

Tips: Diff and merge tools

Add the following configuration to your *gitconfig* to use the VS Code editor for diff and merge operations.

```
tool
[merge]
    tool = vscode
[mergetool "vscode"]
    cmd = code --wait $MERGED
[diff]
    tool = vscode
[difftool "vscode"]
    cmd = code --wait --diff $LOCAL $REMOTE
```

Type `git difftool` in your terminal to open VS Code for diff operations and `git mergetool` to open VS Code for merge operations.

More tips: branch, commit, and push

My personal approach

- Even if you work only on a solo project creating branches can be very useful
  - You can experiment without taking risks to the main part of the project
- Commit early and often
- Keep a logical sequence of commits
  - For example, create a *separate* commit if you work on the data section of a paper and another *separate* commit if you work on the methodology section of your paper.
- Write meaningful commit messages
- Don't keep your collaborators waiting. Push everything that you want to collaborators to see asap

Personal remote workflow (Option 2)

- Create a new repo on GitHub
  - Add README.md, .gitignore and license. Can also add collaborators
- Clone remote to an empty, non-VC local directory
- Work on your local repo
- Push from the local to the remote repo

Pros:

- Creating the repo on GitHub first means that it will always be "upstream" of your (and any other) local copies
  - In effect, this allows GitHub to act as the central node in the distributed VC network
  - Especially valuable when you are collaborating on a project with others

Cons:

- Not every project goes on GitHub
- Certain local projects *might* collide with GitHub created repos

Always pull from the upstream repo *before* you push any changes