Collaborative Programming

Kilian Lieret

Princeton University

Markdown-based slides available as open source; contributions welcome!

Please open these slides on your laptop as we'll jump back and forth between the slides and the browser!

Structure of this lecture

1. **Collaboration in the browser**
    1. GitHub Gists
    2. Creating issues
    3. Forking a repository
    4. Creating commits
    5. Opening pull requests
    6. Handling merge conflicts
2. **Working with a local repository**
3. **Learning to love git** (showcase of advanced topics)

GitHub Gists

the easiest way to share code on GitHub

The "gist" interface works more like a pastebin (and is very easy to use!)

Use case:

- Embed code snippets in websites that don't natively support it properly (like medium)
- Quick sharing of simple code snippets

However: **Gists are not meant for collaboration or larger pieces of work!**

Tasks

1. Go to github.com and click the "+" symbol and select "Gist" **or** directly go to gist.github.com
2. Create a simple file and publish it

Let’s explore the history of an open source repository

Tasks

1. Navigate to https://github.com/klieret/collab-git-playground-codas-hep-23 ("playground repository")
2. Explore the history and find a hidden tiger 🐅!

Let’s open an issue!

Tasks

1. Please navigate to the playground repository
2. Open an issue with a random feature request

Bonus tasks

- Edit the **title & description** of your issue
- Add a **comment** mentioning another participant
- Use an emoji **reaction**
- **Close & reopen** your issue
- Check for other issues and comment there

Advanced

- Install the gh command line tool
- Clone the repository
- Use the CL to open an issue

Forking & committing changes

While you can open issues, you do not have permissions to directly modify content.

Tasks

1. Click the **fork** button. This will create a "copy" of the whole repository
2. Open the `content` folder and click `Add file` > `Create new file`
3. Call your file `<your gh username>_first` and add a few lines to it
4. Add a commit message and commit
5. Confirm that you see your file & a new commit

⚠️ Usually you always want to commit to a separate branch in this scenario (later!)

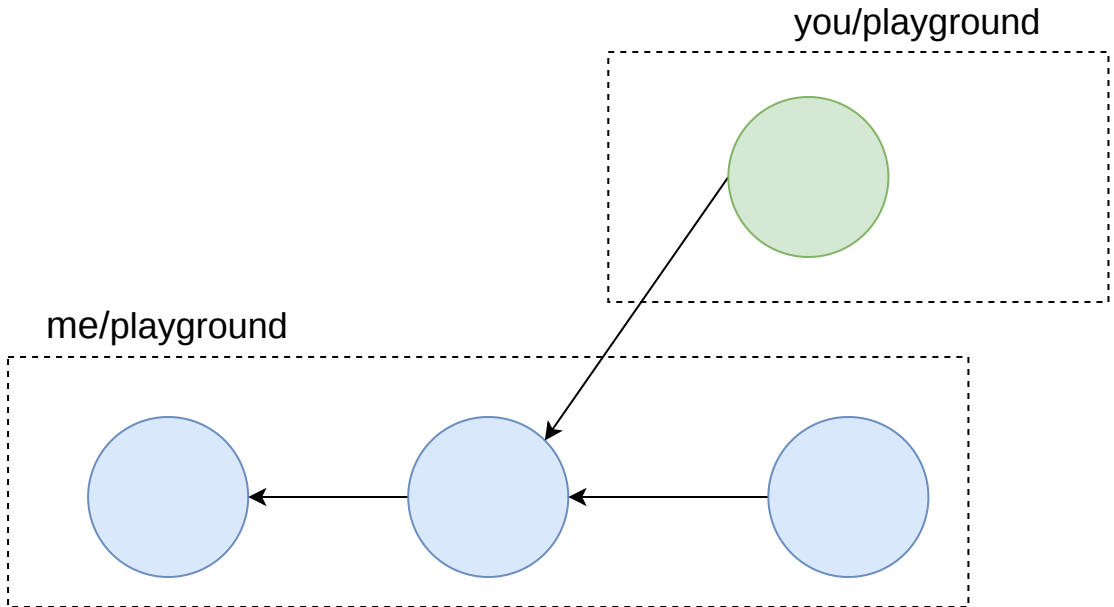
Bonus tasks

- Add a second file
- Open your previous file and make changes to the text

Advanced

Do the same with the gh CLI.

What did we do?



- Every node is a commit. Every commit points to a parent.
- Your fork "branched off" of the original repository: You’re adding additional commits to a parallel reality
- Next step: Bringing your commits back to the original repository

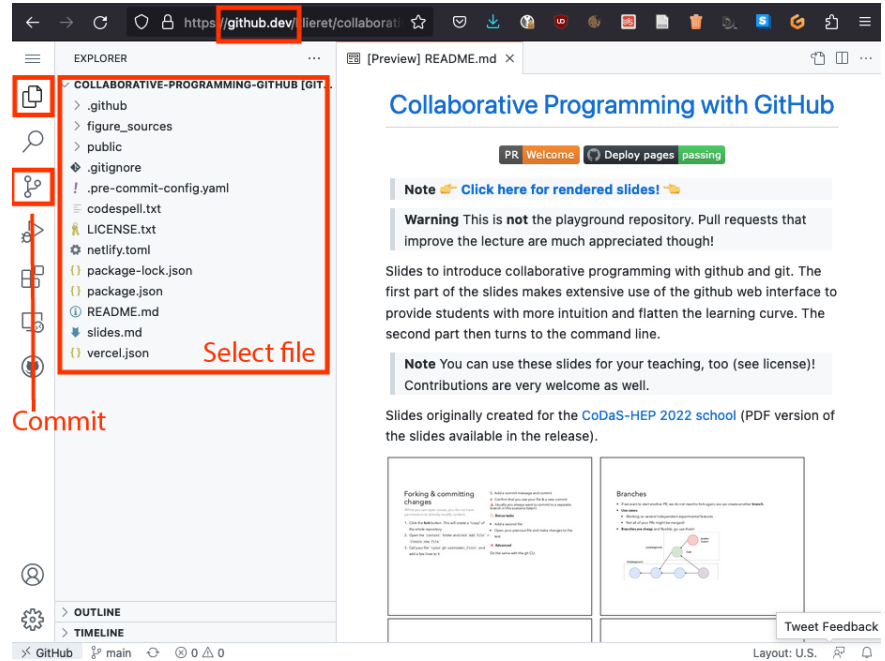
Advanced editing

Starting `VSCode` in the browser

Tasks

1. Navigate to the playground repository
2. Do one of these:
   - Press `.` (opens in same tab),
   - Press `>` (opens in new tab)
   - change the URL from `github.com` to `github.dev`.
3. Make some additional changes
4. Commit by clicking on the git tab in the left menu, adding a message and pressing 'commit & push'

Change back to the previous view by changing from `github.dev` back to `github.com`.

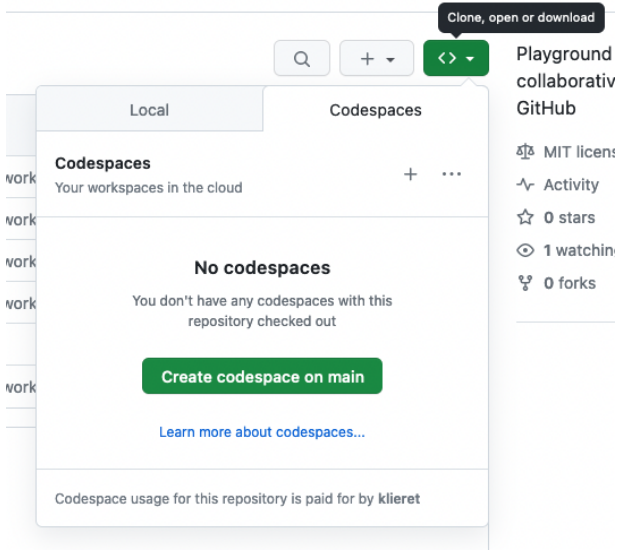


Advanced-er editing with GitHub Codespaces

A full development platform

Tasks

1. Go back to the repository and open GitHub codespaces:



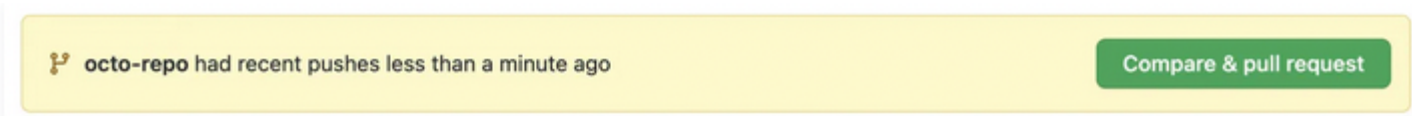
2. Type `` `echo 'hello world'` `` in the terminal
3. Install stuff: `` `sudo apt-get update && sudo apt-get install fortune &&` ``

Creating a PR

How to bring our changes back to the original repository

Tasks

- If you create new commit on a fork, github will already offer you a button to open the PR. Click it!

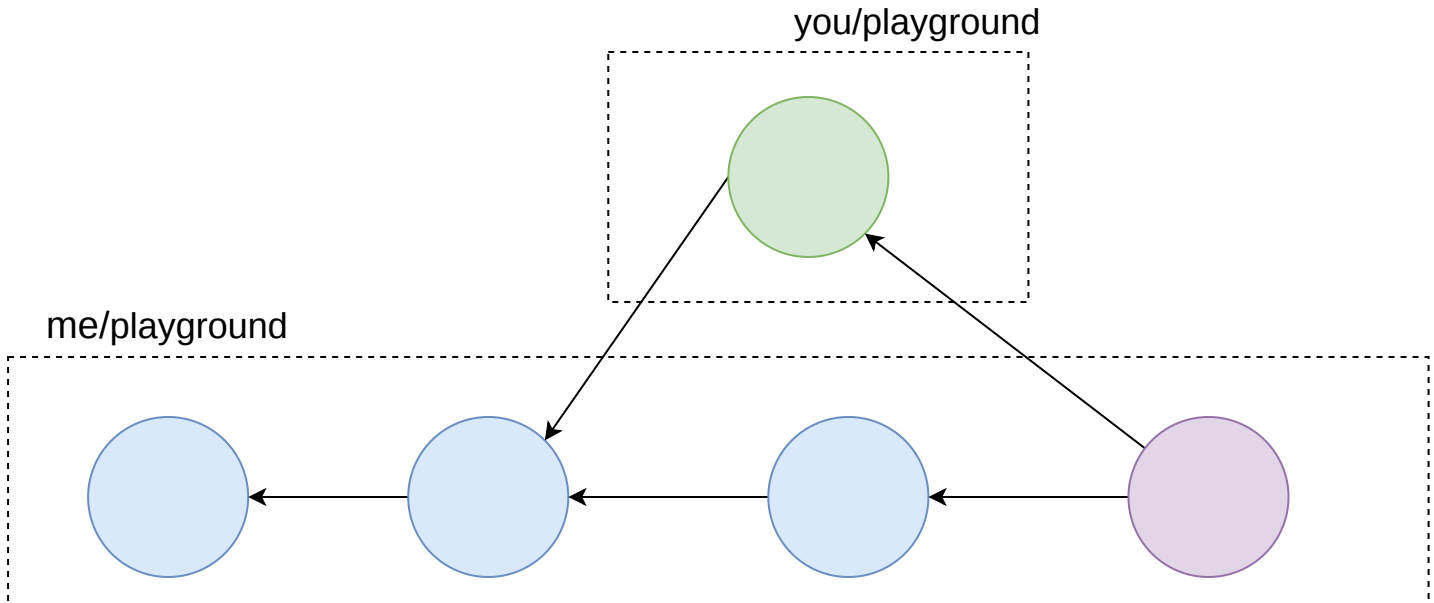


Bonus tasks

- Mention one of your issues. If you write `` `Closes #<number of your issue>` `` and the PR is merged, the issue will automatically close.
- Check the differences that the PR will create
- Comment under one of the differences
- Mention another participant `` `@<name>` ``

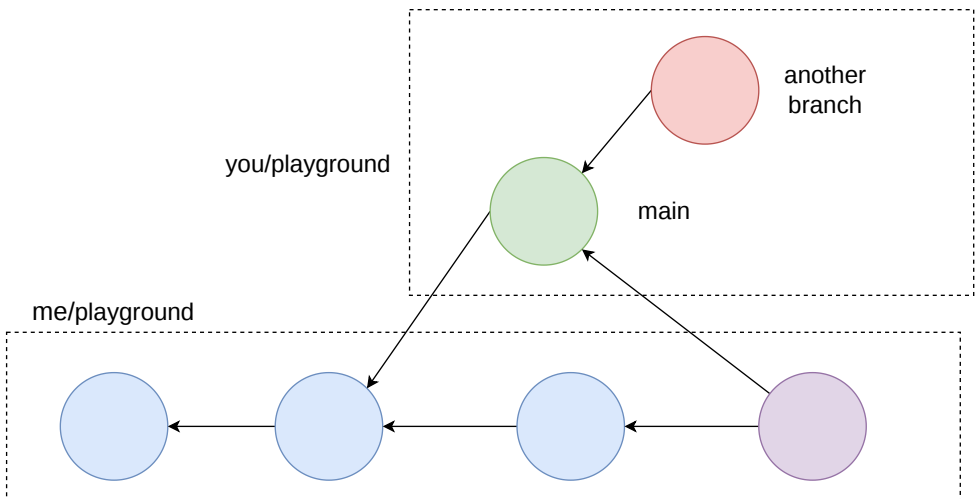
What did we do?

Someone just merged your pull request!



Branches

- If we want to start another PR, we do not need to fork again.
- This time however, we first create another **branch** in our fork.
- Use cases:
  - Working on several independent experimental features
  - Not all of your PRs might be merged!
- **Branches are cheap** and flexible, always use them!



Forks vs Branches

- A *fork* copies the entire repository:
  - Similar to copying the entire local project folder (including your `.git` repository)
  - If the original repository is deleted, your fork persists
  - You *own* your fork and have every permission there
- A *branch* belongs to its repository and only tracks certain changes
- Branches are cheap and easy, forks are expensive
  - **If you have write permissions for a repository you do not usually need/want to fork it**
    - If you need to fork, fork once and then use branches

Branches

1. Add another file `content/<your gh username>_second`
2. Select `Create a new branch for this commit and start a pull request`
3. Give your branch a reasonable name (whitespace discouraged)
4. Commit!
5. Create another PR to either:
   - Your neighbor's `main` branch
   - The original repository (`klieret/ ...`)
   - Your own `main` branch
6. If you want to do the bonus exercises, mark your PR as `draft`
7. If you receive a PR, merge it (unless it's a draft)

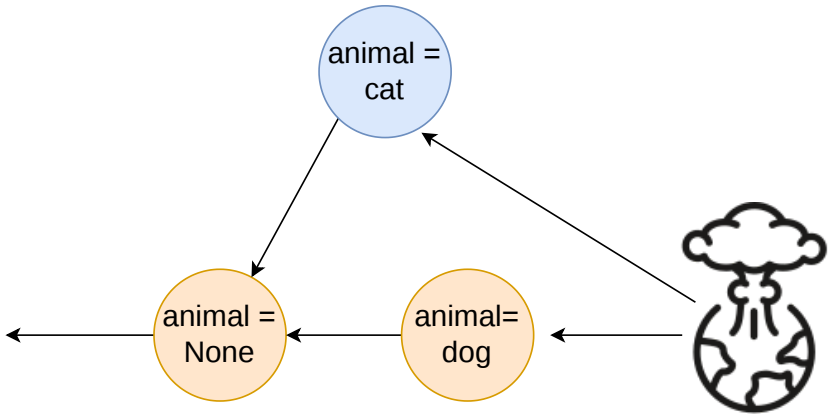
Bonus task: Adding additional commits to a PR

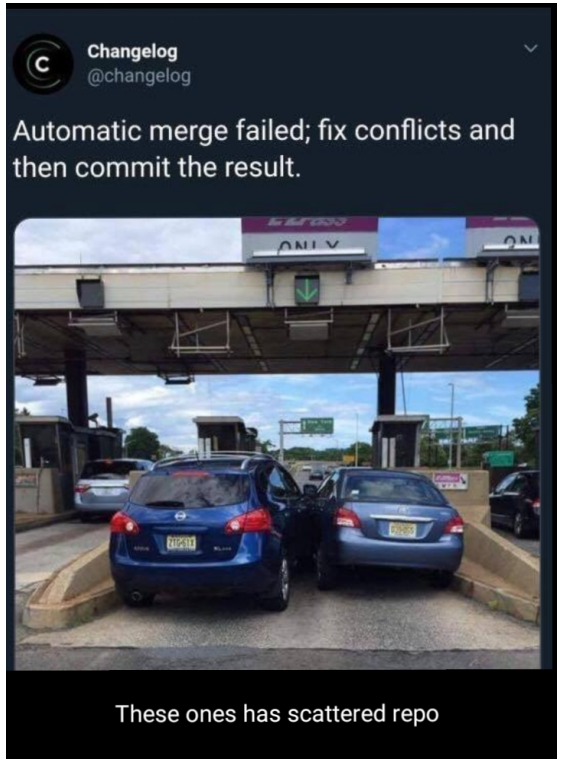
5. Go back to the default view of your repository and verify that you now have multiple branches
6. Select your new branch
7. Modify your just created file and create a new commit on the same branch
8. Check that your PR has been updated by this new commit
9. Remove `draft` status and ask repository owner to review + merge

Bonus task: Go crazy!

Commit to various branches, create PRs between your branches or to your neighbors branches.

Merge conflicts



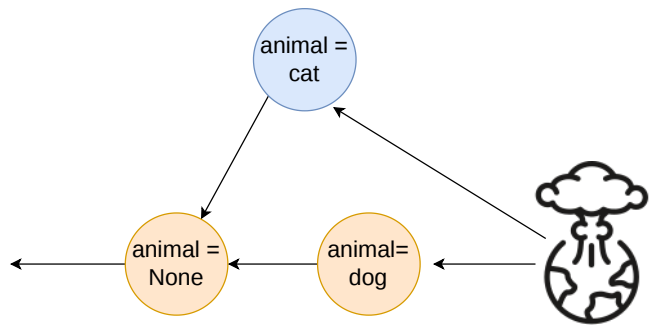


- source

Merge conflicts

⚠️ **Please follow these instructions *precisely*!**

1. Go to your fork
2. Verify that you are on the `main` branch (yellow)
3. Change something in `content/<your gh username>_first` and commit to the branch (!) `merge-conflict` (blue)
4. Open a pull request to your own `main` branch. Do not merge the PR yet!
5. Change to your `main` branch again
6. Change the same (!) line to something different and commit (to `main`)
7. Check back on your PR, it should warn you about a conflict
8. Resolve the conflict by determining how both changes should be reconciled
9. Commit the merge



Bonus tasks: Verify that if you change different lines with unchanged lines between them, git will do the merge automatically.

Part 2: The command line

Let’s get you set up

Configure name, email and editor

If you run git for the first time,

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
# Choose your favorite editor, e.g., nano or vim
git config --global core.editor nano
# Requires git 2.28
git config --global init.defaultBranch main
```

If you haven’t done already, generate ssh keys for authentication to github

```
ssh-keygen
# follow the instructions
cat ~/.ssh/id_rsa.pub
```

and add the key to github. Then clone your repository:

```
git clone git@github.com:<your username>/collaborative-programming-github.git
```

Please raise your hand if you have any issues!

Your first commit

```
cd collaborative-programming-github
cd content
ls
# Get changes that were done on the remote, just in case
git pull
# show status of git repository
git status
# Create new file
touch <your gh handle>-third.txt
# Status is dirty now
git status
# Commit file
git commit <your gh handle>-third.txt -a -m "My third file"
# Clean again
git status
# View past commits (quit with q)
git log
# Push to the remote
git push
```

Bonus tasks:

- Create a few more commits (changing the file)
- Commit without the `-m` option and enter your commit message manually

Changing multiple files in one commit

```
# change all three of your files
git status
# multiple files should now show "unstaged changes"
git add <your gh handle>-first.txt <your gh handle>-second.txt
git status
# two files "staged"
# Commit. Careful: Do not use the -a option
git commit -m "Committing changes to two files"
git status
# one file still showing unstaged changes
git add <your gh handle>-third.txt
git commit -m "Commit to one file"
# Bring changes to github again
git push
```

Hints:

- If you want to add everything to the stage: `` `git add .` `` or use the `` `-a` `` option for git commit
- If you want to remove a file from the staging area: `` `git reset <file>` ``
- If you want to unstage all files: `` `git reset` ``

Branches

```
git branch my-new-branch
git status
# still on branch 'main'
git switch my-new-branch  # or: git checkout my-new-branch
git status

# Now use your previous knowledge to create some more commits

git status
# Make sure that everything is committed
git log
# Verify that you have added a view commits


git switch main
# Verify that the changes from the other branch are not present
git log
# Also our commits aren't present
```

Merging

Bring the commits from `my-new-branch` back to `main`

Advanced: Add more commits to main before merging to set yourself up for a merge conflict

```
# On branch main
git merge my-new-branch
# Should work directly unless you're doing the advanced exercise
```

Advanced: Manually modify the files to resolve the conflict, then `git commit -a`.

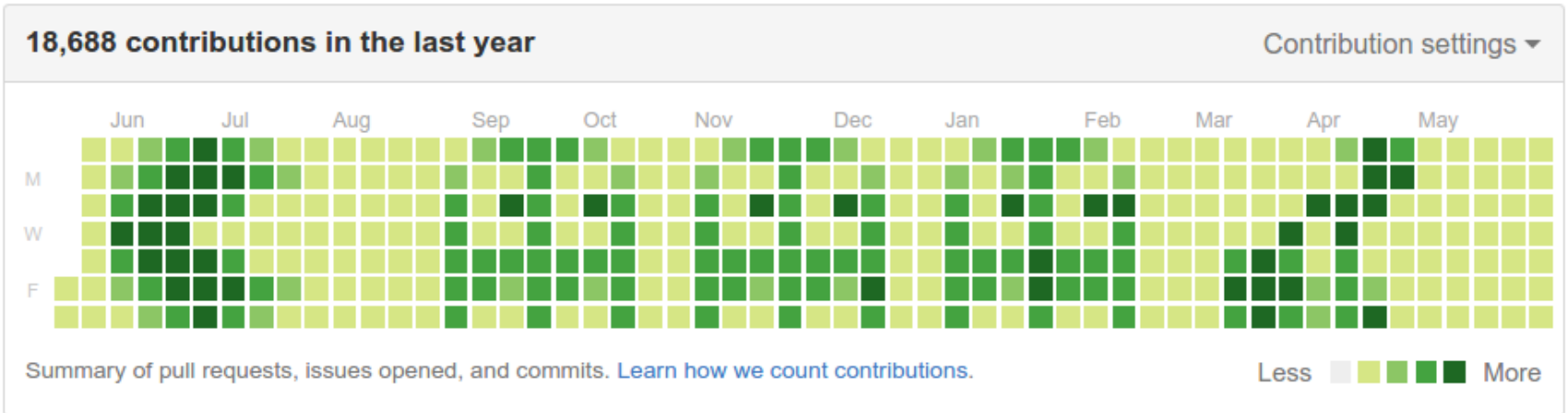
What we didn’t tell you about today

… but what you should really know about

- `git show`: Show details about a commit
- `git diff`: Show differences
- `git stash`: Temporarily put changes aside
- `.gitignore` files: Avoid tracking irrelevant files
- `git revert`: Revert changes
- `git checkout`: Jump through history (or between branches)
- …

Take a look at a cheat sheet like this one and make sure you understand all commands listed.

Part III: Learning to love git



source

- If you are developing software, you almost certainly will use git, no matter where.
- Learning to master git is perhaps THE most transferable skill you can hone.
- Git would not be so dominant if you could not learn to love it.

Your git config

- All repository specific settings live in `` `<your repo>/.git/config` ``. Take a look!
- You can set global settings in `` `~/.git/config` ``. Take a look!

> **Rule of thumb:** If you are unsure about the metadata of your repository or about commands that modify it, take a look at your `` `.git/config` ``.

Defining aliases

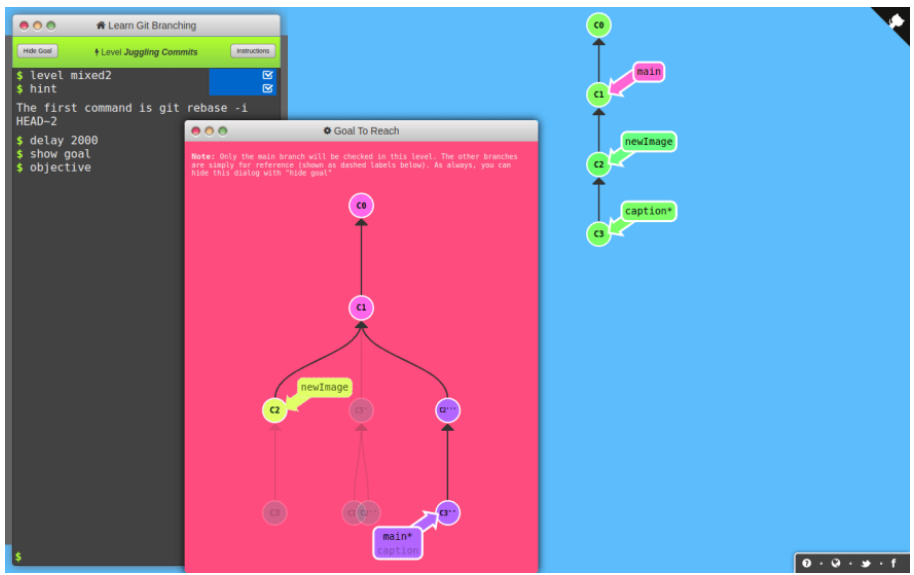
```
# Type `git c` instead of `git commit`
git config --global alias.c commit
git config --global alias.ca commit -a
# ...
# Use `g` instead of git
alias g="git"
# You need to put this definition in your bashrc (or other zshrc etc.) to make it last
```

Alternatively you can also directly write into your config file.

Practice, practice, practice

and then some more

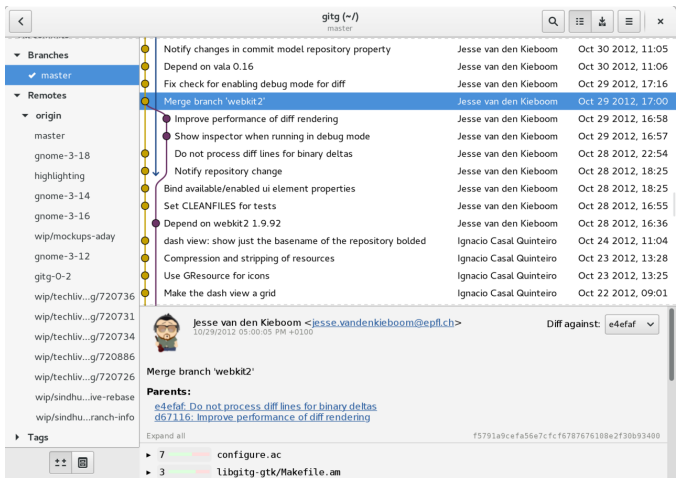
This page has very nice suggestions for different levels, all of them using **gamification** but increasing in realism.



Graphical tools

can give you more intuition

Go here for a curated list of them.



source

Best practices

Licensing Don’t forget to add a license to your repository or nobody can use it! See for example the GitHub docs for more information.

GitHub actions You can automatically run unit tests or other automated tasks every time you commit (or perform other actions on GitHub). Take a look at GitHub actions. For simple checks with almost no setup, also take a look at pre-commit and its GitHub integration.

Thanks!

You can also practice by improving these very slides! Go to https://github.com/klieret/collaborative-programming-github. Issues, forks and PRs are very welcome! You only need to speak markdown to help.