

21 – Collaborative development with hosted repositories

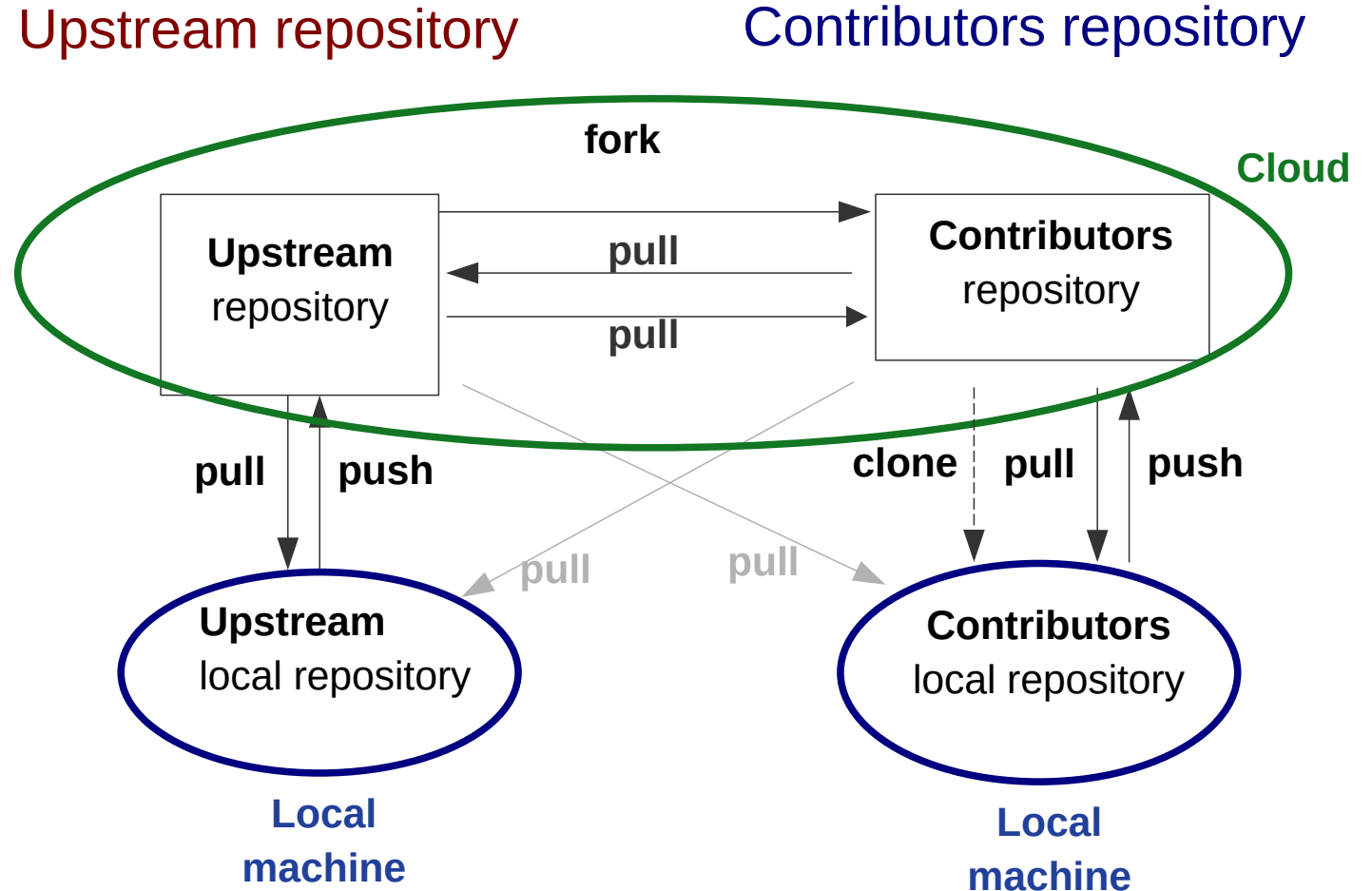
Bálint Aradi

Scientific Programming in Python (2025)

<https://atticlectures.net/scipro/python-2025/>

Remote git-hosting

- Public git hosting sites use the “**fork-pull-push**” workflow
- Similar to “branch & merge in two repositories”
- Local repository is “published” via **push** to public hosting site
- Changes from other repositories are imported via **pulls** from the public repositories at the hosting site



Create new repository (upstream)

- **Create** new repository locally (**local repository**) and make first commit

```
mkdir hello  
cd hello  
git init
```

```
print("Hello!")    hello.py
```

```
git add hello.py  
git commit -m "Initial checkin"
```

- Create repository with same name on Git the hosting server (**remote repository**)
- **Connect** local repository with remote repository

```
git remote add origin git@GIT_HOST:USERNAME/hello.git  
git remote -v
```

Registers remote repository as "origin"

Lists registered remote repositories

- **Push** local copy to remote repository "origin"

```
git push -u origin main
```

Replace this with your git hosting server
(e.g. github.com, gitlab.com, ...)

-u: connect main permanently with origin/main

Fork & clone existing repository (contributor)

- **Fork** repository of upstream user on the Git hosting server
(creates a copy on the hosting server, copy remains [associated with upstream repository](#))
- **Clone** repository from your namespace

```
git clone git@GIT_HOST:YOUR_USERNAME/hello.git
```

- **Register upstream** repository (e.g. for keeping track of updates on upstream/main)

```
cd hello  
git remote add upstream git@GIT_HOST:UPSTREAM_USER/hello.git  
git remote -v
```

Developing a feature (upstream)

Upstream: develop feature in feature branch, merge into main, push

```
git switch -c cleanup
```



```
def main():  
    print("Hello!")  
  
if __name__ == "__main__":  
    main()
```

hello.py

```
git add -u  
git commit -m "Wrap script in main()"
```

Developing a feature (upstream)

```
git add README.rst
git commit -m "Add readme file"
```

```
git switch main
git merge --no-ff cleanup
git branch -d cleanup
```

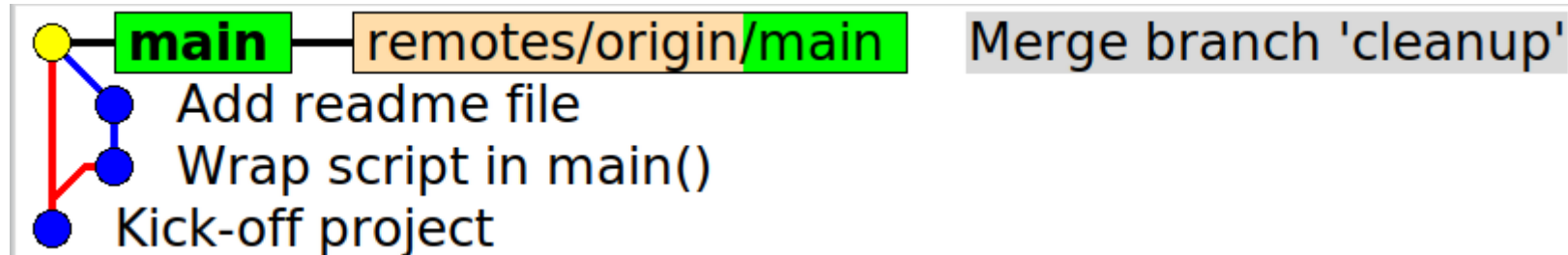
***** README.rst

Hello

Trivial demo project.

Publish your project changes on the remote git-host

```
git push origin main
```



Developing a feature (contributor)

Contributor: develop feature in feature branch, synchronize changes, push, make PR

```
git switch main  
git switch -c extend
```



```
print("Hello, World!")  hello.py
```

```
git add -u  
git commit -m "Extend greeting"
```



```
print("Hello, World!")  hello.py  
print("How are you doing?")
```

```
git add -u  
git commit -m "Make greeting more polite"
```

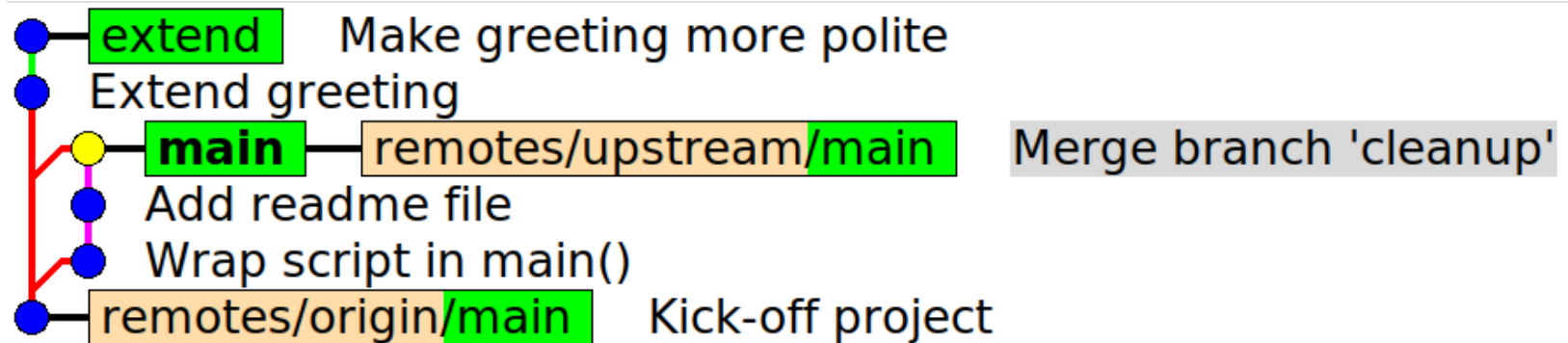
Developing a feature (contributor)

Developer 2: synchronize main branch with devel1/main

```
git switch main  
git pull --ff-only upstream main
```

Fast-forward only: Would fail if main had been modified apart of being pulled from devel1/main

→ Main branch of contributor identical to upstream/main



Developing a feature (contributor)

Developer 2: merge updated main into feature branch, fix eventual conflicts

```
git switch extend
git merge main
```

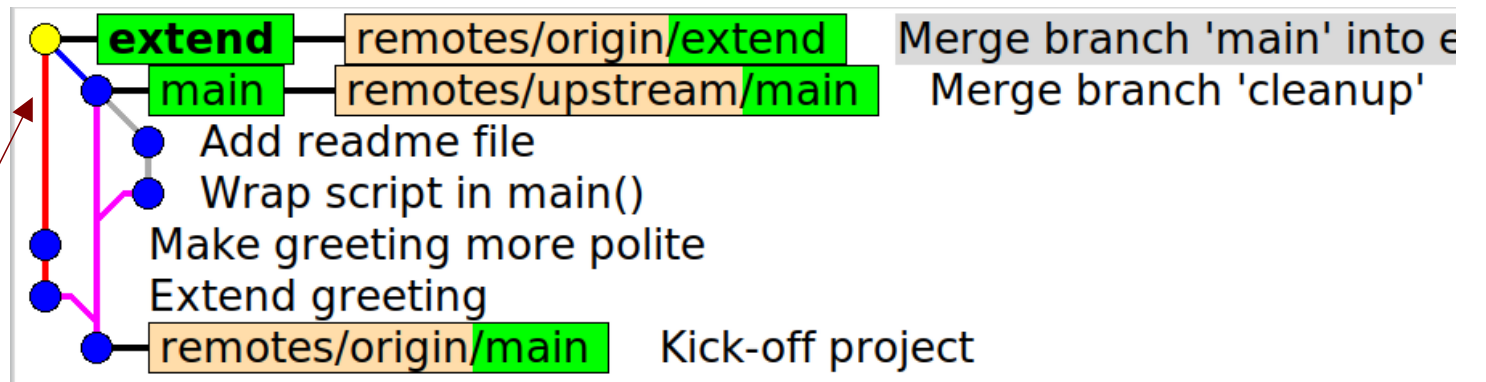
```
def main():  
    print("Hello, World!")  
    print("How are you doing?")  
  
if __name__ == "__main__":  
    main()
```

hello.py
(resolved)

```
git add hello.py  
git commit
```

Publish your changes

```
git push origin extend
```



Make pull request / merge request (contributor)

Active branches

Branch	Updated	Check status	Behind	Ahead	Pull request
extend 	 6 minutes ago		0	6	 

 New pull request

Extend greeting #1

 Open baradi09 wants to merge 3 commits into [aradi:main](#) from [baradi09:extend](#) 

 Conversation 0

 Commits 3

 Checks 0

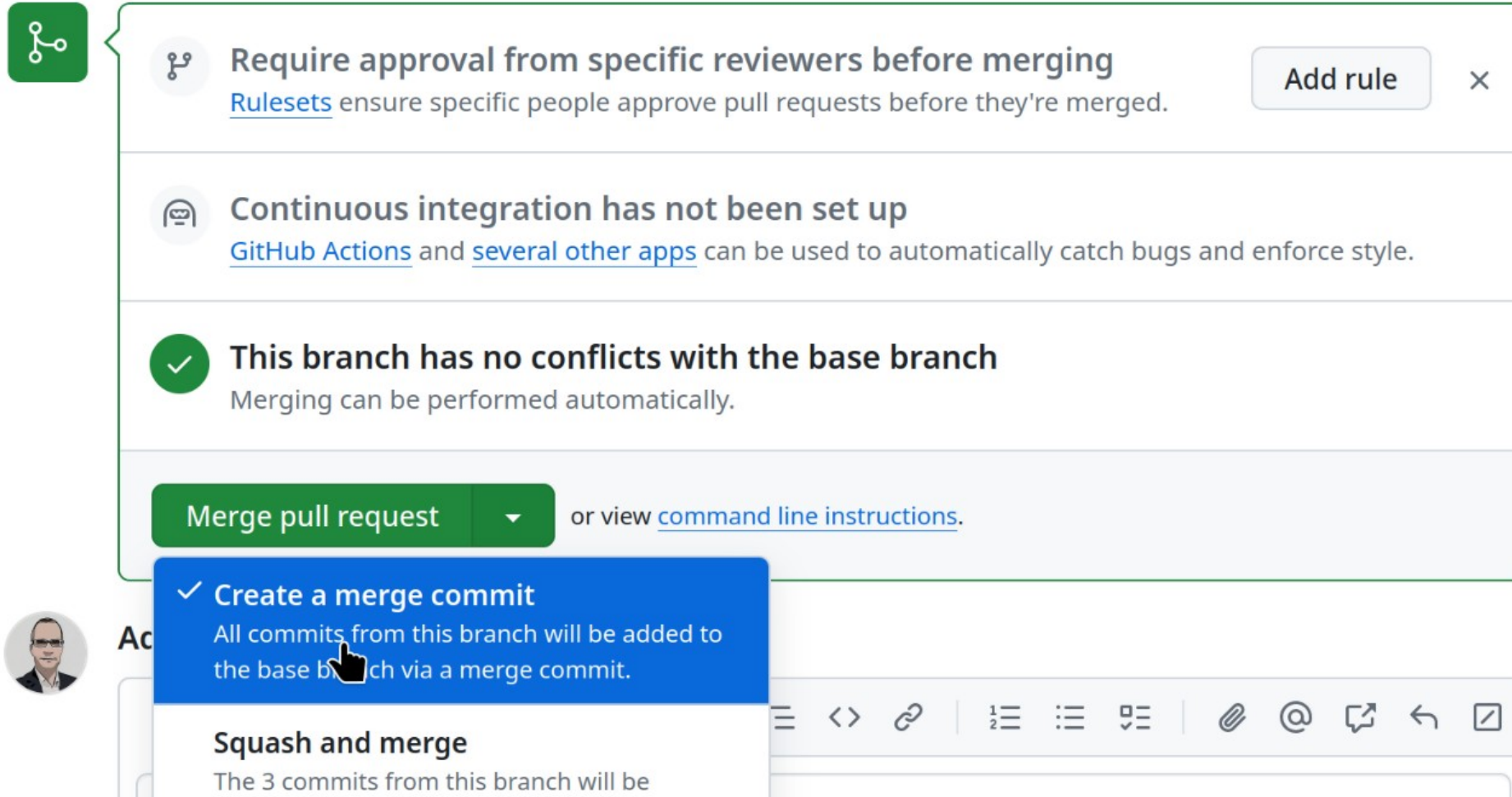
 Files changed 1

- You can add further commits to this branch during review and push it as before

Accept pull request (upstream)

- When PR gets accepted, changes are merged to upstream/main

Add more commits by pushing to the **extend** branch on **baradi09/hello**.



The screenshot shows a GitHub pull request interface. At the top, a green icon with a branching diagram is visible. Below it, a box contains three items:

- Require approval from specific reviewers before merging**
[Rulesets](#) ensure specific people approve pull requests before they're merged. Add rule ×
- Continuous integration has not been set up**
[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.
- ✓ This branch has no conflicts with the base branch**
Merging can be performed automatically.

Below these items is a green button labeled "Merge pull request" with a dropdown arrow, followed by the text "or view [command line instructions](#)."

A blue modal dialog is open over the "Merge pull request" button. It contains:

- ✓ Create a merge commit**
All commits from this branch will be added to the base branch via a merge commit.

Below the modal, the "Squash and merge" section is partially visible, showing the text "The 3 commits from this branch will be".

At the bottom of the interface, a toolbar with various icons (code, link, list, etc.) is visible.

Sync main with upstream/main (contributor)

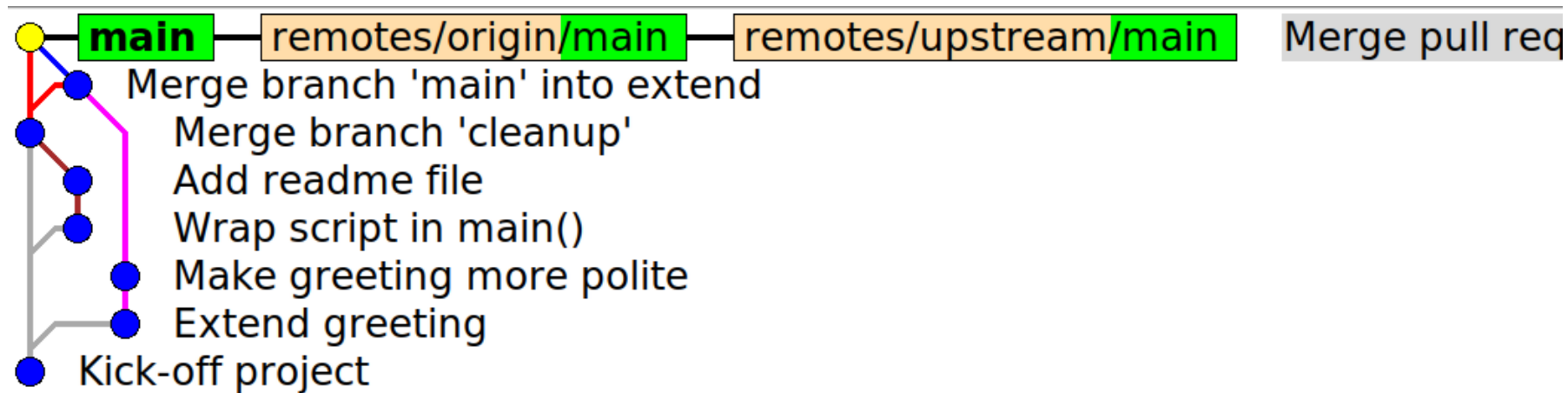
```
git pull --ff-only upstream main  
git push origin main
```

```
git branch -d extend
```

Deletes local branch

```
git push --delete origin extend
```

Deletes remote branch



Contributors workflow (summary)

Fork / Pull / Push model

- Keep up to date with current changes on main
- **Create** feature **branch**
- **Develop** feature
- (**Synchronize** with upstream changes, if needed)
- **Push** branch
- Make **pull / merge request**
- When feature had been merged:
 synchronize,
 delete local and remote branch

```
git switch main  
git pull --ff-only upstream main
```

```
git switch -c BRANCH_NAME
```

Develop locally

Synchronize with upstream if needed

```
git push origin BRANCH_NAME
```

Make pull request on host

Wait for upstream to merge changes

```
git switch main  
git pull --ff-only upstream main  
git branch -d BRANCH_NAME  
git push --delete origin BRANCH_NAME
```

Few random notes

- **Repository** can be made **private**, contributors must be then invited to the repository (they usually gain write access to the repository then)
- If all developers have write access to the repository (small projects), the same repository might contain all temporary feature branches (no forking necessary)
- Git hosting services also offer automatic testing / code checking, etc. (CI – **continuous integration**)
- It is a good idea to configure git globally to **only allow fast-forward pulls**

```
[pull]                                ~/.gitconfig  
ff = only
```

↑
Makes --ff-only the default for pulls

- Two strategies for **incorporating project changes from upstream**:
 - **Merging** (as explained here)
 - **Rebasing** (might be more complicated, but allows linear history)

Further reading:

- [GitHub quickstart guide](#)
- Any other GitHub / GitLab / Bitbucket tutorial
- Several projects have their own detailed Git-workflow guides (e.g. [DFTB+ Git-workflow](#))