

20 – Collaborative development with Git

Bálint Aradi

Scientific Programming in Python (2024)

<https://atticlectures.net/scipro/python-2024/>

Branch & merge in two repositories

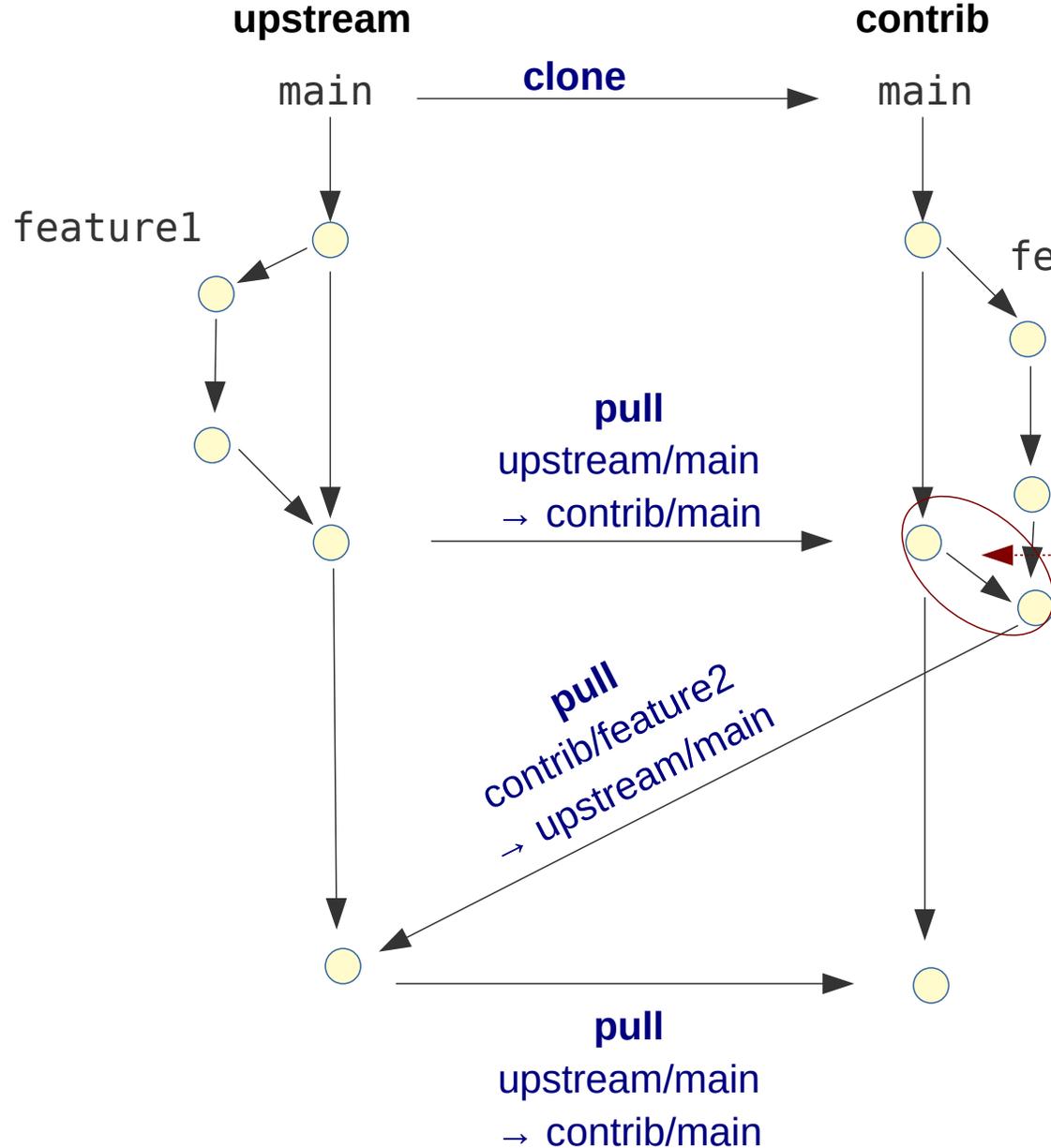
Typical scenario (e.g. open source projects)

- Program is developed by **multiple developers** simultaneously
- There is one “official” (**upstream**) version of the project with main developer(s) (developer(s) in charge) and several **contributors**.
- Parties have only **read-only access** to each others repositories

Typical workflow

- Every developer regularly synchronizes **main** to keep it identical to **upstream/main**
- Each developer implements features in **feature branches** derived from his/her main branch
- The main branch of the contributors is never modified directly, only when synchronized with **upstream/main**
- If feature development is finished, **main developer pulls contributors feature branch** and merges it into upstream main

Branch & merge in two repositories



- Work-flow works very well also with **large nr. of contributors**
- Developers need write access only to their own repositories

Merge

contrib/main → contrib/feature2

Brings feature branch up-to-date with upstream/main (ensures it contains all changes on upstream/main)
→ no conflict should arise if feature branch is merged into upstream/main

Branch & merge in two repositories (#1)

Developer 1: create "official" repository

```
mkdir -p gitdemo/devel1/hello
cd !$
git init
```

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

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

● **main** Initial checkin

Developer 2: clone repository of Developer 1

```
mkdir -p gitdemo/devel2
cd !$
git clone -o devel1 ../devel1/hello
```

● **main** — **remotes/devel1/main** Initial checkin

main in
current repo

main in
devel1's repo

How we refer to cloned
repository (default: origin)

Branch & merge in two repositories (#2)

Developer 1: develop feature in feature branch and merge into main

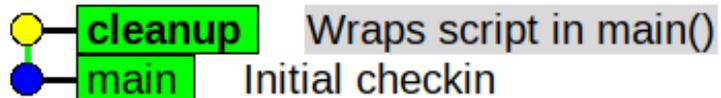
```
git branch cleanup  
git switch cleanup
```



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

hello.py

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



Branch & merge in two repositories (#4)

Developer 2: develop feature in a branch

```
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?")
```



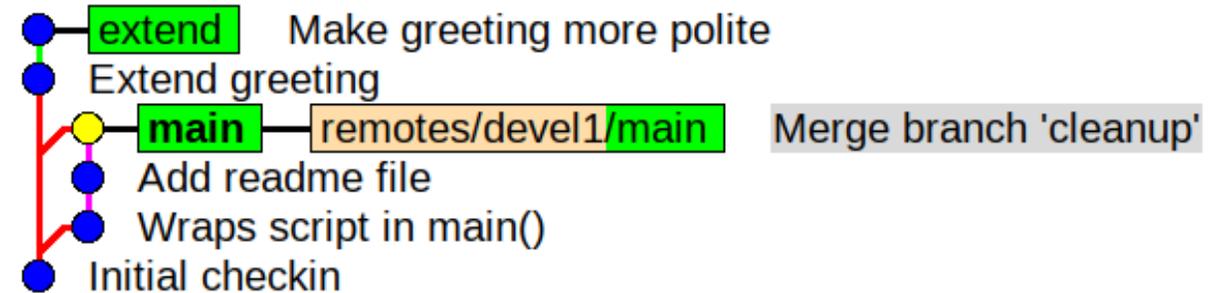
```
git add -u  
git commit -m "Make greeting more polite"
```

Branch & merge in two repositories (#5)

Developer 2: synchronize main branch with devel1/main

```
git switch main  
git pull --ff-only devel1 main
```

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



→ Main branch of developer 2 identical to devel1/main

Branch & merge in two repositories (#6)

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)

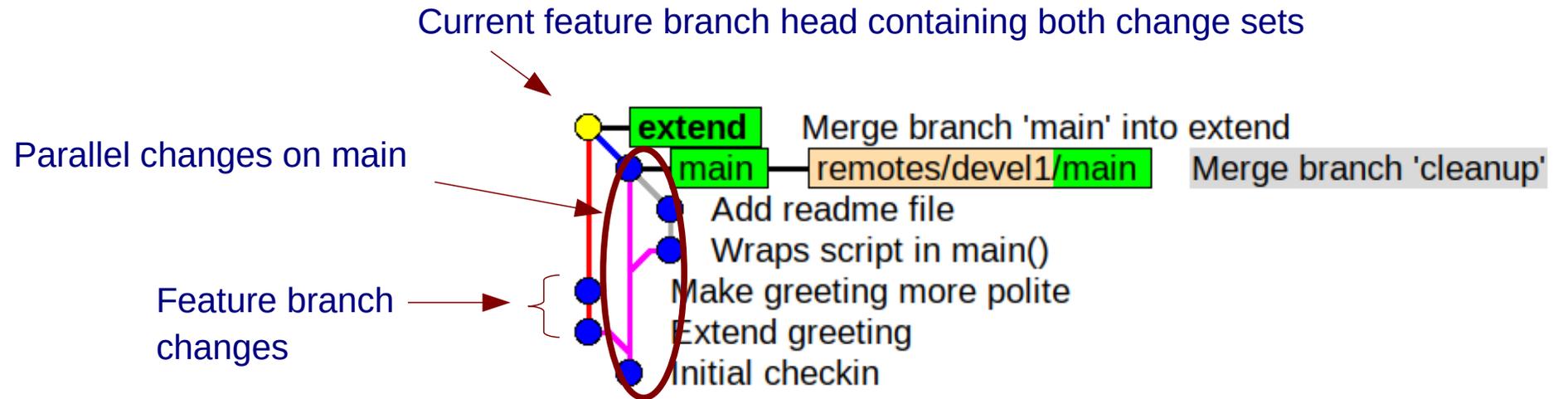


```
git add hello.py  
git commit
```

```
<<<<<<< HEAD  
def main():  
    print("Hello!")  
  
if __name__ == "__main__":  
    main()  
  
=====  
print("Hello, World!")  
print("How are you doing?")  
>>>>>> extend  
hello.py
```

Branch & merge in two repositories (#7)

- Updated feature branch now contains all changes from the original feature branch as well as all changes happened on devel1/main since the feature branch was created



- Feature branch is ready to be merged into devel1/main
No conflicts expected (as they had been resolved by Developer 2)
- Publish feature branch (send repository to Developer 1, push it to GitHub/GitLab)
- Issue **pull request / merge request:**
Ask Developer 1 to pull and merge the updated feature branch into devel1/main

Branch & merge in two repositories (#8)

Developer 1: Fetch and investigate changes from Developer 2

```
git remote add devel2 ../../devel2/hello
git remote -v
```

Register contributors repository
(needed only once)

```
devel2 ../../devel2/hello (fetch)
devel2 ../../devel2/hello (push)
```

```
git fetch devel2 extend
```

Fetch content of "extend" branch
from devel2's repository

```
From ../../devel2/hello
```

```
* branch          extend      -> FETCH_HEAD
* [new branch]    extend      -> devel2/extend
```

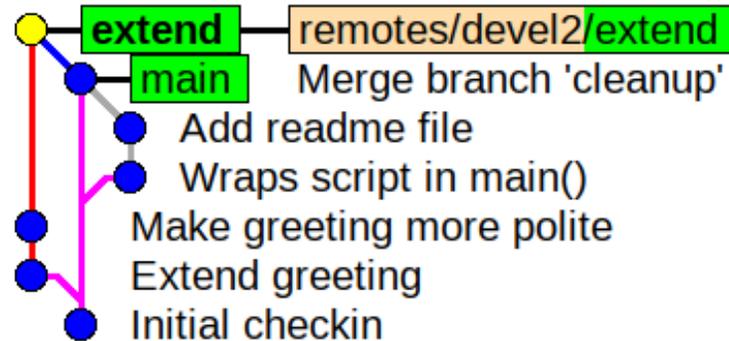
```
git switch extend
```

Check out contrib/extend as extend for further inspection

Branch 'extend' set up to track remote branch 'extend' from 'devel2'.
Switched to a new branch 'extend'

Branch & merge in two repositories (#9)

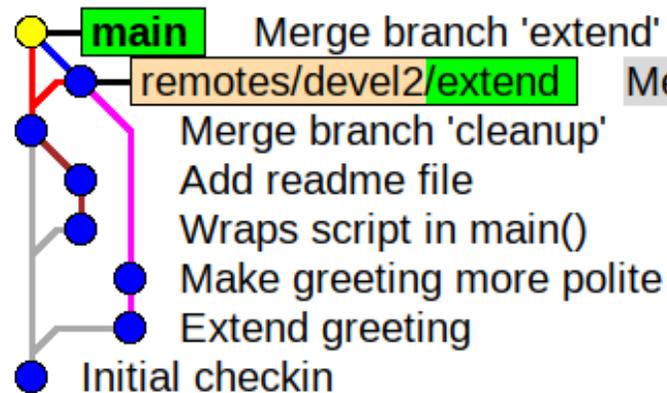
- Developer 1 has an exact local copy of Developer 2's feature branch



Merge branch 'main' into extend

Developer 1: merge feature branch into main

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



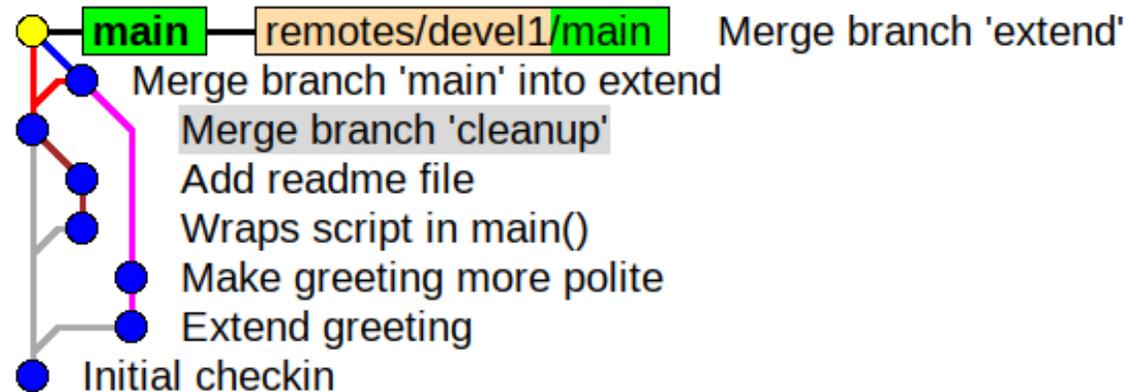
Merge branch 'main' into extend

Developer 1's main contains all previous commits + changes from contributor

Branch & merge in two repositories (#10)

Developer 2: sync main branch with Developer 1's main

```
git switch main  
git pull --ff-only devel1 main  
git branch -d extend
```



Developer 2's main branch indential to devel1/main!

Publishing a repository

Publish repository, so that others can clone it and pull from it

- Allow read-access to repository in local file system (multi-user environment)
- Upload repository to public file- / webserver
- Send repository (including .git/) as an archive
- Publish repository on a git hosting site (e.g. [GitHub](#), [GitLab](#), [Bitbucket](#))
 - Very convenient and de-facto standard for open-source projects
 - **Note:** Those site are **commercial** ones (with commercial interests), but usually offer free of charge services for private persons, students, etc.
- Run your own git hosting infrastructure (e.g. self-hosted GitLab)

Some random git notes

- Git is very flexible and powerful, allowing for almost **arbitrary workflows**
 - Most open source projects document their git-workflow (e.g. [DFTB+ git-workflow](#))
 - If you start your own project, pick a common one (e.g. [GitHub flow](#))!
- Public git-hosting sites are usually offering very good tutorials on git and git-workflows (see for example [GitHub guides](#))
- The free “git book” [Pro Git](#) contains an excellent introduction to git.
- Instead of merging a source branch into a target one, one can also rebase the target branch upon the source branch. (**Rebasing** is not trivial, so make sure you **understand its consequences**, before you do it.)