Type hints & Git workflow via git hosting

Bálint Aradi

Course: Scientific Programming / Wissenchaftliches Programmieren (Python)

https://www.bccms.uni-bremen.de/people/b-aradi/wissen-progr/python/2023

Outline

- Type hints

- Git workflow with remote hosting

Type hinting

Type hinting

- Python allows for annotation of variables
- Annotation information can be used by IDE and tools to check type-consistency
- Annotations are not evaluated at run-time (no type safety)

```
def fibonacci(nterm: int, order : int = 2) -> int:
    ...
    return num
```

```
def fibonacci(nterm: int, order : int = 2) -> List[int]:
    ...
    return fibolist
```

```
def print_hello(name: str) -> None:
    ...
```

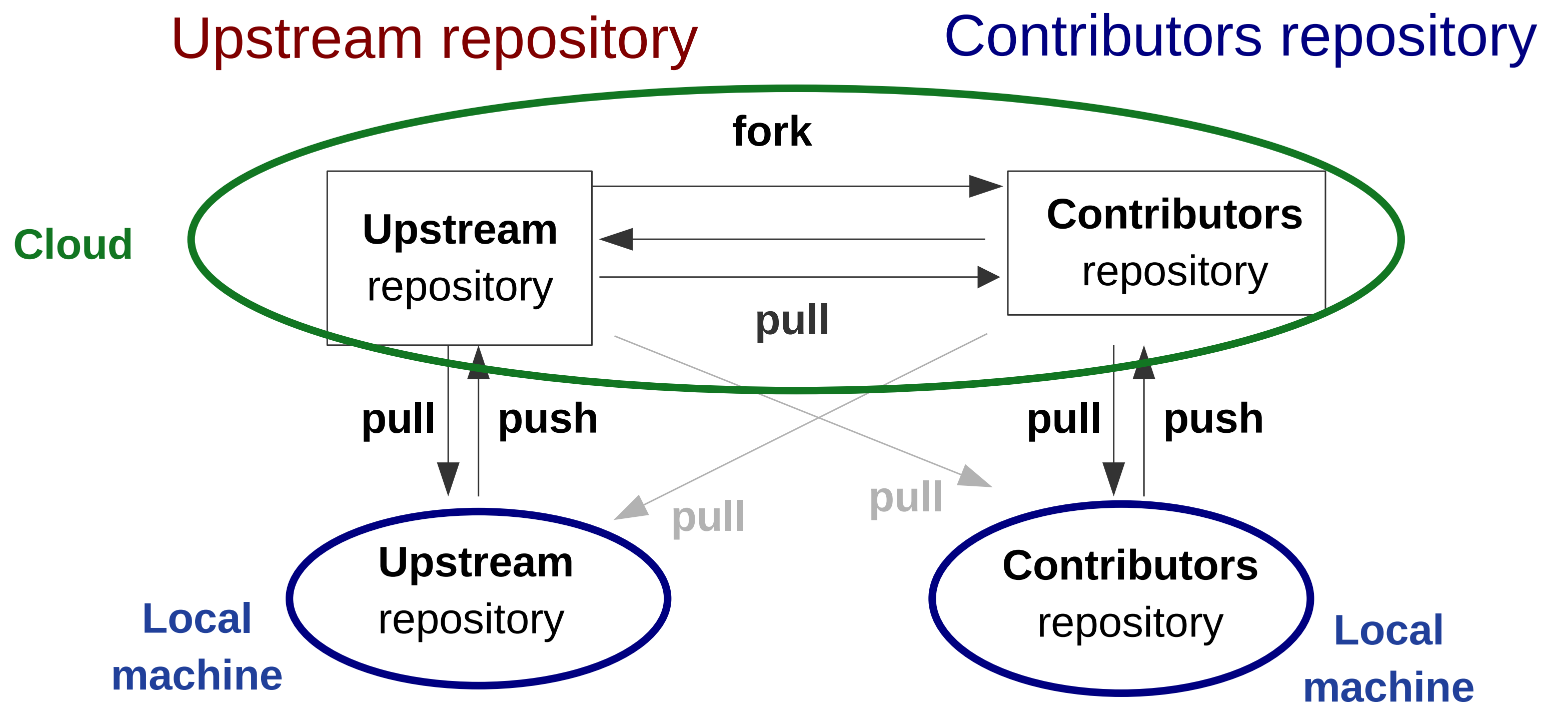
See also

- Type hinting cheat sheet of MyPy

Multiple repositories, multiple branches

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



Authentication via ssh

- Generate ssh key-pair (choose a proper passphrase!)

```
ssh-keygen
```

$\leftarrow$ Stores ssh key-pair in .ssh/

`id_rsa`	**Private** key (never give it away!!!)
`id_rsa.pub`	**Public** key (put on the remote host)

- Register the **public** part of your SSH-key on the remote Git hosting service

GitHub: `Settings / SSH and GPG keys / New SSH key`

- Make sure, **ssh-agent** starts automatically
  - Windows (see next slide)
  - Linux / macOS: very likely, this is already set up out of the box

- Unlock your key for **ssh-agent**

```
ssh-add
```

You only have to enter your passphrase once, ssh-agent authenticates you whenever needed

Autostarting ssh-agent in Git-Bash (Windows)

.bashrc

```
env=~/.ssh/agent.env
agent_load_env () {
    test -f "$env" && . "$env" >| /dev/null
}
agent_start () {
    umask 077
    ssh-agent >| "$env"
    . "$env" >| /dev/null
}
agent_load_env
agent_run_state=$(ssh-add -l >| /dev/null 2>&1; echo $?)
if [ ! "$SSH_AUTH_SOCK" ] || [ $agent_run_state = 2 ]; then
    agent_start
fi
unset env
```

Copy & paste this into the **~/.bashrc** file

ssh-agent should start automatically when you open GitBash terminal

Create new repository (Upstream)

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

```
mkdir greetingdemo
cd greetingdemo
git init
…
git add …
git commit ...
```

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

Registers remote repository as “origin”

```
git remote add origin git@github.com:USERNAME/greetingdemo.git
git remote -v
```

Lists registered remote repositories

- **Push** local copy to remote repository “origin”

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

-u: connect main permanently with origin/main

Fork & clone existing repository (Contributor)

- **Fork** repository of other 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@github.com:YOUR_USERNAME/greetingdemo.git
```

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

```
git remote add upstream git@github.com:UPSTREAM_USER/greetingdemo.git
git remote -v
```

Developing a feature (Contributor)

- **Create feature branch** in your local repository (e.g. “docs”)
- **Implement** your feature (e.g. add readme)
- When finished, **push** your branch to your remote repository

```
git push origin BRANCH_NAME
```

- Make a **pull request** (merge request) **to upstream/main** to incoprorate your changes
- You can add further commits to this branch during review and push it as above

Delete merged branch (Contributor)

- Once feature had been merged to upstream/main, feature branch should be deleted

```
git switch main
git branch -d docs
git push --delete origin doc
```

Deletes local branch

Deletes remote branch

Synching local main with upstream/main (Contributor)

- Contributors main branch should always be a 1-1 copy of upstream/main

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

Ensures, that pull is only successful, if local main branch has not been manipulated...

- It is a good idea to configure git globally to only allow fast-forward pulls

~/.gitconfig

```
[pull]
    ff = only
```

Makes --ff-only the default behaviour at pulls

- After bringing your local main branch up to date, you might want to push that to your remote repo:

```
git push origin main
```

Contributors workflow (summary)

Fork / Pull / Push model

- Keep up to date with current changes on main

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

- Create feature branch

```
git switch -c BRANCH_NAME
```

- Develop feature, push branch

```
git push origin BRANCH_NAME
```

- Make pull / merge request

PULL/MERGE REQUEST

- When feature had been merged, delete branch

```
git switch main
git branch -d BRANCH_NAME
git push --delete origin BRANCH_NAME
```

Upstream developers workflow

Workflow for upstream developer is basically identical to contributors workflow

- Keep up to date with current changes on main

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

- Create feature branch

```
git switch -c BRANCH_NAME
```

- Develop feature, push branch

```
git push origin BRANCH_NAME
```

- Make pull / merge request

PULL/MERGE REQUEST

- When feature had been merged, delete branch

```
git switch 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**)

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)

Have fun!