Git Workflows

1 Git setup

- **Github-first approach**: start by creating a repository on GitHub, and then clone it to your computer: `git clone <github-repo.git>`
- **Local-first approach**: initialize version control in local directory via `git init`. Then either publish your local directory to Github via VS Code, or create a Github repo using the Github web site and link your local repo to the Github repo via `git remote add origin <github-repo.git>`.

2 Solo project

Scenario: You are doing a solo project without collaborators. You are using Git and Github for the purpose of version control and backup.

2.1 Commits in main branch

The simplest workflow would be to just do all the work in the main branch:

```
git add .                               # Stage all changed files
git commit -m "Message"                 # Commit changes
git push                                # Push to your Github repo
```

2.2 Feature branch workflow

Alternatively, you can use the **feature branching** workflow: all edits and commits are done in feature branches instead of the main branch. In this way, you separate your stable version from the unstable, experimental branches.

```
git checkout main                       # Main branch is starting point
git checkout -b <feature branch>        # Create and checkout feature branch
git add .                               # Stage all changed files
git commit -m "Message"                 # Commit changes
git push [--set-upstream origin <feature branch>] # Push to your corresponding branch on Github
```

Even though you have no collaborators, it is good practice to push your feature branch to Github and open a pull request on GitHub. Since it is your own work, you can immediately merge the changes into the main branch on GitHub. Finally, you update your local main branch.

```
git pull
```

Since the feature branch is merged, you don't need it anymore. Hence, you can clean up by deleting the branch from the local repo and from Github.

```
git branch -d <branch name>
```

3 Collaboration projects

3.1 Via shared GitHub repo

Scenario: Collaboration is managed via one central GitHub repository, from which everyone pulls and to which everyone pushes. This scenario is often relevant in a company setting or in case of private projects with few collaborators.

There are several collaborative Git workflows, but often they are variants of the **feature branch workflow**, which was already described above: Initially, you clone the shared GitHub repo to your computer.

```
git clone <central-repo.git>       # Clone central repo
```

You edit, add, and commit in a dedicated feature branch, and then push to shared GitHub repo.

```
git checkout main                       # Main branch is starting point
git checkout -b <feature branch>        # Create and checkout branch
git add .                               # Stage all changed files
git commit -m "Message"                 # Commit changes
git push [--set-upstream origin <feature branch>] # Push to your own fork
```

Then you open a pull request on GitHub. Your team reviews the changes. Until the pull request is finally merged, you can incorporate further changes to your pull request via the add-commit-push sequence. Note that the review is often an iterative process, meaning that it involves several rounds of questions/suggestions and answers/edits/commits. Afterwards, the feature branch can be merged into the main branch directly on GitHub.

3.2 Via forked GitHub repo

Scenario: There is no shared central repository from which everyone pulls and to which everyone pushes. Instead, each contributor gets her own GitHub repository (fork), which is a copy of the official repository (or upstream repository). Local changes are pushed to the fork, and from there a pull request to the main branch of the official repository is opened. This process is called forking workflow. The scenario is typical for open source projects.

1. In order to contribute to the project, you **fork** the upstream repo, which means that you create a copy on your own GitHub page. We will also this one **origin**. There is a fork button in the upper right corner of a GitHub repo website.

2. You clone to your computer, but note that your cloning your fork rather than the official repo.

```
git clone <your fork.git>
```

3. You edit, add, and commit in a dedicated feature branch, and then push to GitHub. However, you don't push to the upstream repo but to your fork (origin).

```
git checkout main                       # Main branch is starting point
git checkout -b <feature branch>        # Create and checkout branch
git add .                               # Stage all changed files
git commit -m "Message"                 # Commit changes
git push [--set-upstream origin <feature branch>] # Push to your own fork
```

4. Then you open a pull request on GitHub to the main branch of the upstream repo, where the pull request is reviewed. Until the pull request is finally merged, you can add further changes to your pull request via the add-commit-push cycle.

If you want to regularly **synchronize with or contribte to the upstream repo**, then you should (just once) register the official repository as a further remote repository.

```
git remote add upstream <upstream-repo.git>  # Only once: Add upstream repo as remote repo
```

This allows you to pull all changes directly from the upstream repository

```
git checkout main                            # Checkout local main branch
git pull upstream main                       # Merge changes from upstream main
```

4 Fixing problems

4.1 Accidental edits in main branch

If your team follows a **feature branch workflow** (or other branch-based workflows), all edits and commits are made in dedicated branches instead of the main branch. Now suppose you accidentally make changes or even commits to the main branch. How can these changes be moved to a different branch, and deleted from the main branch?

4.1.1 Move into NEW branch

```
git checkout main          # Switch into main branch (that has the unintended commits)
git branch <newbranch>     # Create a new branch from it (they now contain these commits)
git checkout main          # Switch into main branch
git reset --hard HEAD~3    # Move main back by 3 commits
```

4.1.2 Move into EXISTING branch

```
git checkout <exisingbranch>
git merge main
git checkout main
git reset --hard HEAD~3
```