Problemset: Git and Github (I)

Note: Many of the following tasks can be carried out on the command line, or using VS Code's Source Control GUI. In order to get practice, repeat the workflows in both ways. I recommend starting on the command line.

1 Basic Git Workflow

First, create a new directory. Then initialize git for this directory (`git init`). How does the directory change? Run `git status` to see the current state of your git repository. From now on run `git status` after each git-related action that you do to understand the possible states that your git repository can be in.

Run the first basic git workflow:

1. Create a new file `readme.md` and save it.
2. `add` the file to the staging area.
3. `commit` these changes to your local git repository using a clear commmit message.

Carry out some standard file operations (create/edit/delete) and repeat the above git workflow. Use `git status`, `git log` and `git diff <12345>` along the way.

```
# Check status
git status                # Verbose information on working directory
git status -s             # Short info
git status -s -b          # Short info + info about current branch

# Add to staging area
git add .                 # Add all changed files
git add <file1>           # ... or specific files

# Commit to local repo
git commit -m "Message"
git commit -am "Short message"   # Add and commit in one step

# Log
git log                   # Complete log
git log --oneline -n 2    # Log of last 2 commits in compact format

# Diff
git diff HEAD~1 HEAD      # Compare previous with current (HEAD) commit
git diff <#12345> <#>     # Compare arbitrary commits
git diff HEAD             # Compare uncommited changes to HEAD
```

2 Workflow including Github

Publish your repo to Gitub from VS Code. Then open the repo on Github and look at how the information is presented there.

Again carry out some standard file operations (create/edit/delete) and repeat the basic git workflow - but this time finalizing the worflow by pushing to Github (`git push`).

Now edit one of the files directy on Github and commit this change there. Then use `git pull` to incorporate your remote changes into your working directory.

Repeat the previous operation in a slightly different way. Again edit one of the files on Github and commit. Then `git fetch` your remote changes. Can you already see the changes in your working directory? What does `git fetch` do? Also use `git status` to check the current state of your repository. Then use `git merge origin/main` to merge into your working directory.

```
# Push to Github
git push                    # Push to associate GitHub channel (here: main channel)
git pull                    # Pull GitHub changes to working directory
git fetch                   # Fetch Gihub changes to origin/main
git merge origin/main       # Merge remote tracking branch into working directory
```

3 Reset a commit

Undo previous commits using `git reset`: (1) Create one or more "bad" commits (e.g. delete an important file, or delete some important code). (2) Use git diff and git log to verify which commit is the last good one, and which commit introduced problems. (3) Reset to the last "good" commit.

Repeat this reset process in two other variants: using the `--hard` flag, and the `--soft` flag? How do these variants differ? Use `git log` and `git status` to verify.

```
git reset        <#last-good-commmit>
git reset --soft <#last-good-commmit>
git reset --hard <#last-good-commit>
```

4 Revert a commit

Another way of undoing a bad commit is using `git revert`: (1) Create a "bad" commit. (2) Use git diff and git log to verify which commit is the last good one, and that the last commit introduced problems. (3) Revert the bad commit using `git revert <badcommit>`. How is reverting different from resetting? What are the advantages and disadvantages?

```
# Log
git log                     # Complete log
git log --oneline -n 2      # Log of last 2 commits in compact format

# Diff
git diff HEAD~1 HEAD        # Compare previous with current (HEAD) commit
git diff <#12345> <#>       # Compare arbitrary commits
git diff HEAD               # Compare uncommited changes to HEAD

# Revert
```

```
git revert <#bad-commit>
git revert --no-edit <#bad commit>
```

5 Clone Github repository

In the beginning, you followed a local-first approach: you first created a local git repository, and then published it to Github.

Now, follow a Github-first approach: reate a new Github repository using the Github website. Then copy the HTTPS url and clone it to your local system (`git clone <url>`). Make sure that you first switch to the location on your system where you want to get the new repository cloned to.

You can also use `git clone` to clone some public Github repo of other authors: Search for Python packages that are published on Github. Then clone one such package to your local system.