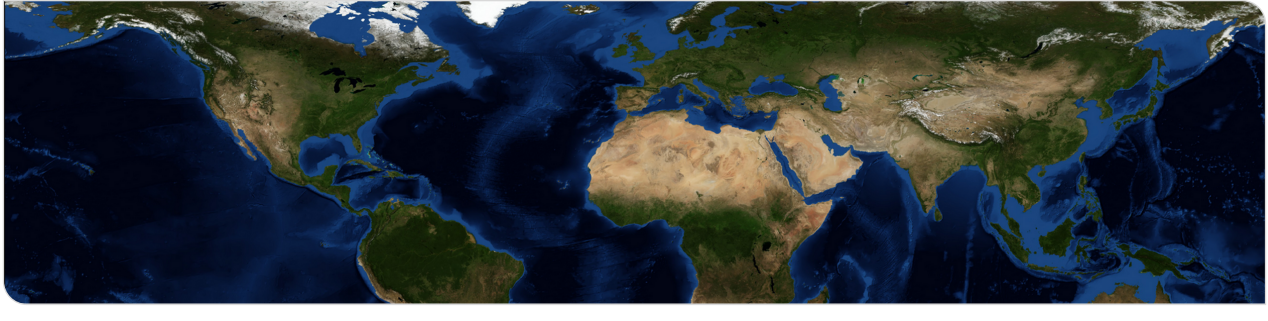
Karlsruher Institut für Technologie

Introduction to Python

8. Git

Dr. Jutta Vüllers, Dr. Julia Fuchs, Dr. Annika Bork-Unkelbach | 06 December 2022



Overview

Motivation

Coding workflow:

- write a piece of code
- test it
- modify it (remove bugs, add functionality)
- test again
- modify again

How to keep track of the changes made to the code?

1. The “easy” way: save every change to a different file

Motivation

Coding workflow:

- write a piece of code
- test it
- modify it (remove bugs, add functionality)
- test again
- modify again

How to keep track of the changes made to the code?

1. ~~The "easy" way: save every change to a different file~~
2. Use version control

version control

Version control: Software tool to document, manage and archive changes to your code
Version control systems use a digital archive (repository) consiting of

- a folder containing the code
- a database to manage and archive changes

central version control
- one repository on a central server
- local working copy
- working offline not possible
- only one person can apply changes at a time

distributed version control
- several repositories on different machines
- local repository and working copy
- working offline using the local repository
- everyone on the team can apply changes everytime

What is git?

Git is the most common (distributed) version control system. It's free and open source:
`https://git-scm.com/`

With git you can

- keep track of changes in your code
- can work together with other on the same code
- keep track of who changed something in the code

Please have a look at this general overview:
`https://www.youtube.com/watch?v=2ReR1YJrNOM`

Git installation on Windows

Please use the following installer for windows machines:
`https://gitforwindows.org/`

Please also read the FAQs carefully!
`https://github.com/git-for-windows/git/wiki/FAQ`

Git installation on Ubuntu

Open a terminal and type:

```
sudo apt-get install git
```

Installation of the git plugin for Jupyter lab - optional

You can install a plugin for better intergration with Jupyter lab:
Search for git in the extensions or follow the instructions on the project website:
`https://github.com/jupyterlab/jupyterlab-git`

Test your git installation

Now that you have installed git on your system, you are ready to test your installation! From now on, we will work with the **git bash** :

- Windows: Search for the app called git bash and open it
- Ubuntu: Open a terminal

```
git --version
```

Get an overview of the help functionality:

```
git --help
```

Configure your git installation

One important feature of git is, that it keeps track of who made a change to a certain file in a project. Therefore, you need to configure your username and email. First, check your global setup:

```
git config --list
```

Next, change the settings, so that you use you KIT ID as username and your student's email as email:

```
git config --global user.name ab1234
git config --global user.email ab1234@student.kit.edu
```

Finally, check if it worked out:

```
git config --list
```

Setup your first git project - preparation

- On your system, navigate to an appropriate location and create a new (empty) folder.
- Give it meaningful name
- open git bash here (or navigate to this location within git bash / terminal)

I created a folder called git_examples on d drive in a general folder called Python Course:

```
gs2614@imk-asf-ms-nab1 MINGW64 /d/Daten/Lehre/Python Course
cd git_examples
gs2614@imk-asf-ms-nab1 MINGW64 /d/Daten/Lehre/Python Course/git_examples
```

Exercise 8.1: Setup your first git project

Initialize an empty git repository:

```
git init
```

- Do you recieve any output?
- Please also open your file browser and navigate to this folder. Do you find any changes?
- You can save your answers in a txt document and place this document within your git folder. Please give it a meaningfull name, for example: ab1234_answers.txt

Exercise 8.1: Setup your first git project

Check the status of your repository:

```
git status
```

Exercise 8.2: Adding documents

Please take one Jupyter Notebook from the last exercises and copy it to your git folder. Next, check the status of your repository.

```
git status
```

Familiarize yourself with the add command:

```
git help add
```

Now, we want git to take a snapshot of all files (changes in files, new files) in the current directory:

```
git add .
```

The files you added are now stored in a temporary staging area (git index).

Exercise 8.3: Committing documents

In the last step, we added new documents to our project and added them to the staging area (git index). In the next step, we want to store a snapshot of these files permanently:
Familiarize yourself with the commit command:

```
git help commit
```

Then commit the changes (new files) to your repository:

```
git commit
```

You will be prompted to give a commit message. This is basically some plain text to describe the changes you made to your repository. Another way of committing the changes is:

```
git commit -m "added answers and jupyter notebook from exercise 2"
```

Exercise 8.4: Making changes

- Write the output of the commit command to the answers file.
- Make some changes to your jupyter notebook and save everything.
- Create a new jupyter notebook with some markdown descriptions. Next, you can add a small piece of code (for example some simple calculations or user input) to it.

Now, lets check if git can show us the changes:

```
    git diff
```

Next, we want to add our changes to the git index:

```
    git add file1 file2 file3
```

or:

```
    git add .
```

Exercise 8.5: Committing changes

Now let's check for the changes we just added to the index. The following command shows us the changes of or in files that have been recently added and can now be committed:

```
git diff --cached
```

Please verify that all your changes are displayed. Now, we are ready to commit again:

```
git commit -m "some meaningful message"
```

Exercise 8.6: Changing and committing a file (shortcut)

Now, add some functionality to your code. Since we only modified a file and did not add a new one, we can now use the following shortcut to add and commit our changes:

```
    git commit -a -m "some meaningful message"
```

Exercise 8.7: Overview of your changes with git log

Up to now, you should have set up your repository, added files to it, made some changes and added them again. The following commands will show you, why using git for managing your code is a good idea. Please save the output in your answers file.

```
    git log
```

For more details, have a look at the help:

```
git help log
```

This command should give you the history of your project. If you made a lot of changes to your project, the output might be a bit confusing and inconvenient to search. The following command will give you a rather brief overview of your project:

```
    git log --oneline
```

Exercise 8.8: Overview of your changes with git log - more details

Now that we have a rough overview of the changes, we would like to see the changes in more detail. First of all, we want to see the number of lines that were inserted or deleted:

```
git log --stat
```

Or:

```
git log --stat --summary
```

Next, lets display the actual changes in the code (files):

```
git log -p
```

Leave with q.

Exercise 8.9: Overview of your changes with git log - filtering

Sometimes we are not interested in the complete history of our project, but we are looking for changes in the last 2 commits, or changes made after a certain date or changes made by a specific author. Try the following commands (with your settings, e.g. dates and user name) and write the output to your answers file. Try also to modify the commands.

```
git log -2
```

```
git log --after="2022-12-4" --before="2022-12-6"
```

```
git log --author="Annika"
```

Exercise 8.10: Creating branches

Sometimes it is useful to keep several copies of your project. For example if you want to add a new feature to your code or if you have to fix a bug (and don't know if this bug fix will impact other parts of your code). The copies of your project(s) are called branches. In this exercise, we will create a new branch of our project to add a new feature to your code:

```
git branch new_feature
```

Get an overview of the existing branches:

```
git branch
```

The branch you are working in, should be marked. Before changing the code, switch to the new branch:

```
git switch new_feature
```

Exercise 8.11: Working with branches

- Once you have switched to your new branch, add a new feature to your code. This can be another calculation or some user input / output.
- Commit the your changes.
- Change to your old branch.
- Add some comments to your code
- Commit the changes.

Now, you should have two copies of your code (branches) which contain different changes. In the next step, we want to combine (merge) our changes:

```
git merge new_feature
```

If your changes can not be merged, you can view the conflicts with:

```
git diff
```

How to resolve conflicts

If you receive any conflicts, please carefully read the help section of the merge command and follow the instructions:

```
git help merge
```

Exercise 8.12: Working with a remote repository: Gitlab

In order to collaborate with your teammates, you can install git for example on a central server. At KIT, a central git repository, includig a web application (gitlab), is hosted by SCC and available for all KIT members:
`https://gitlab.kit.edu`

Some useful links:
`https://gitlab.kit.edu/help`
`https://docs.gitlab.com/ee/tutorials/`
`https://www.youtube.com/watch?v=Jt4Z1vwtXT0&list=PLhW3qG5bs-L8YSnCiyQ-jD8XfHC2W1NL_`

Gitlab is a web application to manage and organize git repositories with additional tools dedicated to professional software development.
Sign in using your KIT account and create a new project (don't forget an appropriate name).

Exercise 8.13: Clone the remote repository

Now you initialized a new project in a remote repository. Next, you should create an empty directory on your machine, where you want to setup your local repository (local copy of this project). In order to work with a local copy of this repository, you have to clone it. Therefore you need to tell your local git the location of the remote repository. You can retrieve this address by clicking on the blue clone button. You can also select how you want to connect to this remote repository:

```
git clone https://gitlab.kit.edu/skl/teaching_python.git
```

Using https you will be asked to setup an access token. Please follow the instructions.

Exercise 8.13: Alternativly push existing repository to remote

In order to push an existing repository to a new gitlab project, you have to:

- create a new project
- click on the branch and edit its security permissions, because it should be protected by default
- follow the commands when you scroll down on your projects website

Exercise 8.14: Apply changes

- Check with your file browser if the cloning of the project worked
- Make some changes to your project: Add a Jupyter Notebook and a text document (for example your answer document)
- Add and commit everything
- Check the remote copy in gitlab: Does it show any updates?

Now, we want to apply our changes to the remote repository:

```
git push origin main
```

Exercise 8.15: Check for changes

When working together on a project, several developers can apply changes to their code locally. Before applying your changes to the remote repository, it can be helpful to check if anyone else already applied changes.
On gitlab, you can view a list of all commits and check the history of single files. On your local machine, you can give the following commands to check the history of the remote repository:

```
git fetch origin
git status
```

And for more details:

```
git log origin/main
git diff origin/main
```

Exercise 8.15: Try yourself!

Now, we want to simulate changes and fetch them:

- Go to your gitlab project and edit something
- Add a new file
- On your local machine, use the commands from the last slide to check for the updates and view the details

Exercise 8.16: Merge your changes

Before applying further changes, it is convenient to get the most recent updates from the remote repository. Try the following commands:

```
git merge origin/main
```

Afterwards, check if everything worked out:

```
git status
```

Exercise 8.17: Shortcut for updates

Sometimes, you simply want to apply any updates from the remote to your local repository. In this case, you can use a combination of the fetch and merge commands:

```
git pull origin
```

Attention: The pull command will always merge into the current branch! Check and switch your branch and pull again. What happens?