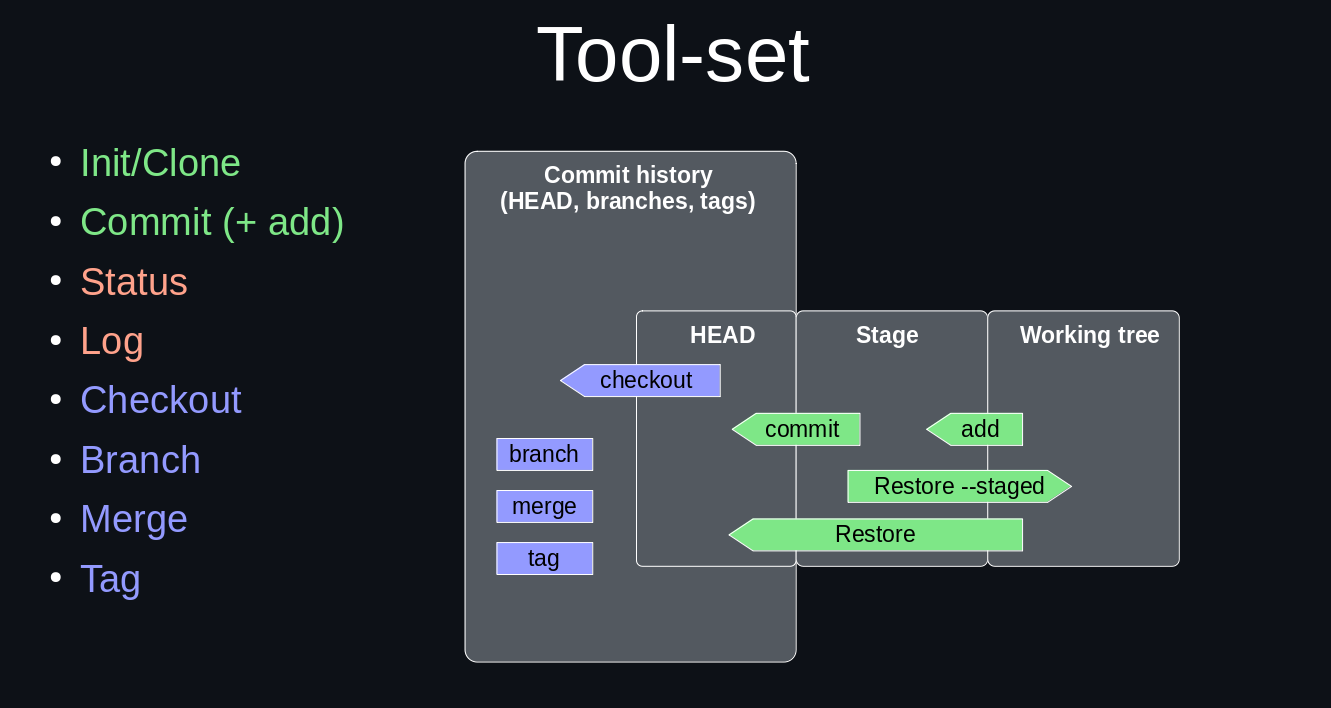
Git/Github workshop – Day 1 Practical session

Mon 09th Oct. 2023 – Ian Dirk Fichtner

PART 1: Try it out

Until now, we’ve covered the concept of Git and its main commands.



Let’s put them now to practice with an example project. The referenced files are accessible via [the following link](https://github.com/idf-io/Git-Github_Workshop/blob/main/code/fastaReader.py).

While following the instructions, feel free to navigate the repository to different parts, use the `git status` and `git log` commands to understand what is happening and even make some of your own experiments and changes if you feel so.

**PART 1: basic functionality (init, add, commit, status and log)**

1. Create your first git repository named “fastaReader”
2. Create, stage and commit a README.md [markdown](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax) file where you briefly describe the project.
3. Create a data/ folder and paste the `example.fasta` file such as in [this link](https://github.com/idf-io/Git-Github_Workshop).
   * What is a [fasta file](https://en.wikipedia.org/wiki/FASTA_format)?
4. Commit the file.
5. Tag the commit as “first-code”
6. Create a `fastaReader.py` python file as in [here](https://github.com/idf-io/Git-Github_Workshop/blob/main/code/fastaReader.py). Read it, understand it and execute it.
7. Commit the file and the output.
8. Modify the script to output lower case letters.
9. Execute it again and commit the script and output.
10. Make another change to the file (any type).
11. Execute it again and commit the script and output.
12. Create a file named “test.py”.
13. Add it to the stage.
14. Then decide that you don’t want it anymore. Remove it from the stage and/or delete it.
15. Move around the commits you made and try some of the commands from the presentation out. We are going to need the part until now intact so don’t modify the last commit.

**PART 2: Branches and merging branches**

Now we have the idea to be able to read in DNA seq instead of AA and to translate the DNA seq to AA seq.

1. Go back to the commit in number 4 (e.g. `git checkout first-code”
2. Create a new branch named “DNAtoAA”
3. Find a DNA fasta file on the internet or invent it (analogous to `example.fasta`).
4. Save it in `data/` as `example\_dna.fasta`.
5. Modify the script to read this file and output a file with name “modified\_dna2aa.fasta”.
6. Commit the change.
7. Modify the script to translate DNA to AA.
8. Execute the script.
9. Commit the script and output.
10. Inspect the differences between the branches “Main” and “DNAtoAA”
11. Now merge them in “Main”
12. Delete the old branch

**Handle some merge conflicts.**

To experience the different kinds of merge conflicts. Use the `git log (--all --oneline --decorate –graph)` command to keep track of your branches (refs)

Fast-forward merge

1. Take you current repository and an arbitrary branch e.g. “Main”
2. Make a new branch.
3. Make some changes
4. Using git log, you should see that the “Main” branch reference is in the same commit history as the new branch but further behind.
5. Merge the new branch onto “Main”
6. The only thing that will have happened is that the “Main” branch reference will have been moved forward to where the new branch is.

3-way merge

1. Take you current repository and an arbitrary branch e.g. “Main”
2. Make a new branch
3. Make some changes
4. Checkout “Main” again.
5. Make some changes e.g. add new files and modify existing ones.
6. Merge to the the new branch to “Main”
   * If git is not clever enough, you will have to resolve merge conflict. This is usually only the case if both branches modified the same code line.

Final merge task

1. Clone [the following repository](https://github.com/idf-io/Git-Github_Workshop/tree/main)
2. Navigate to the branch “mods1”
3. Have a brief look at it.
4. Navigate back to the branch “main”
5. Merge “mods1” into “Main”
6. Resolve the merge conflict
7. Delete “mods1”