git in a nutshell – important commands

The time machine – git locally

0. start the time machine (only once!)
   → go to your folder
   → run: `git init`

1. time machine engine check
   → shows you files that are new or changed
   → run: `git status`

2. glimpse into the past
   → find out what snapshots your time machine has stored – they are called commits
   → run: `git log`
   → `git log -2` gives you the last 2 commits

3. something changed – but what?
   → find out what changed since the last commit
   → run: `git diff`
   → this will give you changes in all files that are indexed by git
   → for a specific file run: `git diff myfile.py`

4. make a new snapshot you can return to
   → this is a 2-step process!
   → first, you have to tell git which of the changed files should enter the snapshot: `git add myfile.py`
   → if it is <u>all</u> changed files, you can also say: `git add .`
   → then, you tell git how you want to log this snapshot – this will create the snapshot, or also: the commit
   → run: `git commit -m "fixed sampling bug"`
   this will commit everything you added in step 1 ↑ (git commit)
   this adds a commit message ↑ (-m "fixed sampling bug")

5. time travel time
   → you want to go back in time to an old version
   → find out which commit is the one you want to revert: `git log`
   → revert the work from the commit:
   `git revert a1b2cd`
   ↑ this is the commit hash you identified using log

Parallel universes – branching in git

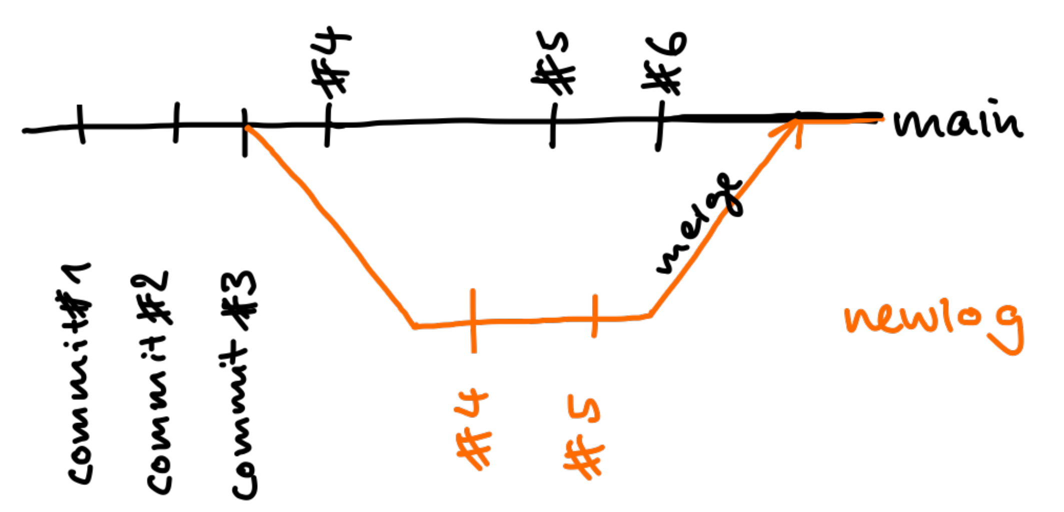
0. imagine you want to try a new analysis but you are not sure you want to keep it
   → with git, you can have multiple copies of your work in a CLEAN way
   → they are called branches
   → you have always at least one branch: main

1. start a new branch
   → run: `git checkout -b newlog`
   ↑ "create new branch" ↑ name of new branch

2. you can swap to this new branch
   → check where you are first: `git branch`
   → jump to the new branch:
   `git checkout newlog`

3. now you have two parallel universes with a common history.
   → you can merge them again, for this, first go to the branch you want to continue with:
   `git checkout main`
   → merge the other branch into main:
   `git merge newlog`
   ⚠ to hop between branches you always have to commit your work first!

Branching in git



NOTE: this might create a merge conflict. no need to panic – but manual sorting will be required.

The backup machine – git and GitHub

0. you can link an online service like GitHub for backups, sharing, collaborating

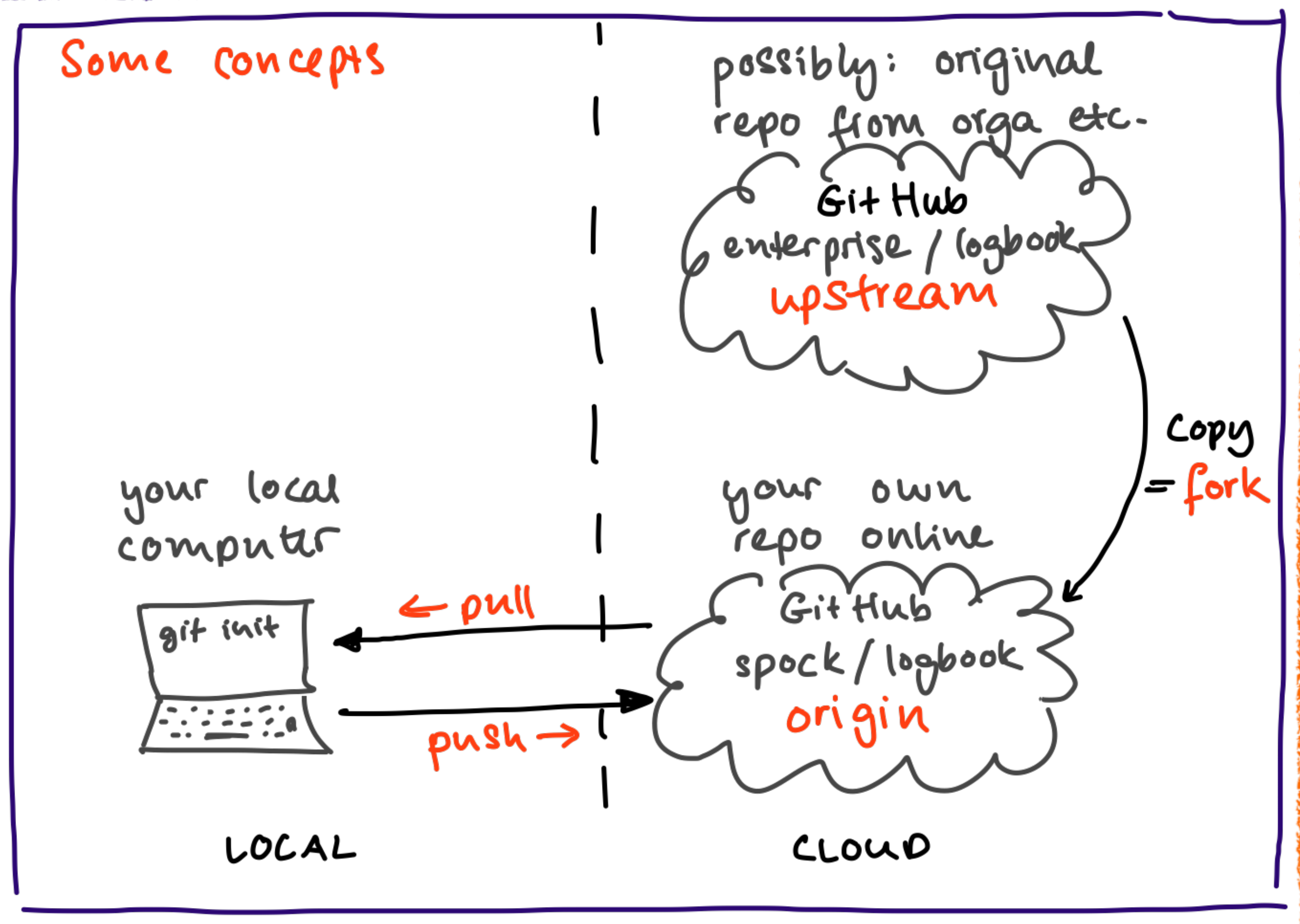
1. First scenario: you start from scratch
   → make an empty repository ("repo") online and copy its URL
   → go to the parent folder on your computer
   → run: `git clone git@github.com:spock/logbook.git`
   this is the URL of your repo
   ⚠ format is different for SSH and HTTPS!

2. Second scenario: you have a local git folder
   → create an empty repo and copy the URL
   → add the URL to your local folder
   → run: `git remote add origin git@github........`
   URL, see 1.

3. check on what you have linked
   → run: `git remote -v`
   → that will show you if you have any repos linked, both origin and upstream

4. get your changes online
   → we push the changes to the online repo
   → run: `git push origin main`
   ↑ where? ↑ what? (branch)

5. get changes from the online repo:
   → we copy the online status (e.g. collab)
   → run: `git pull origin`



Some things to keep in mind:
- GitHub repos can be public, careful what you share!
  → private info, sensitive info
- if you share code, include a license!

© Dr. Britta Westner • britta-wstnr.github.io