Let's Git together

Alessandro Sciarra

Z02 – Software Development Center

XX August 2022

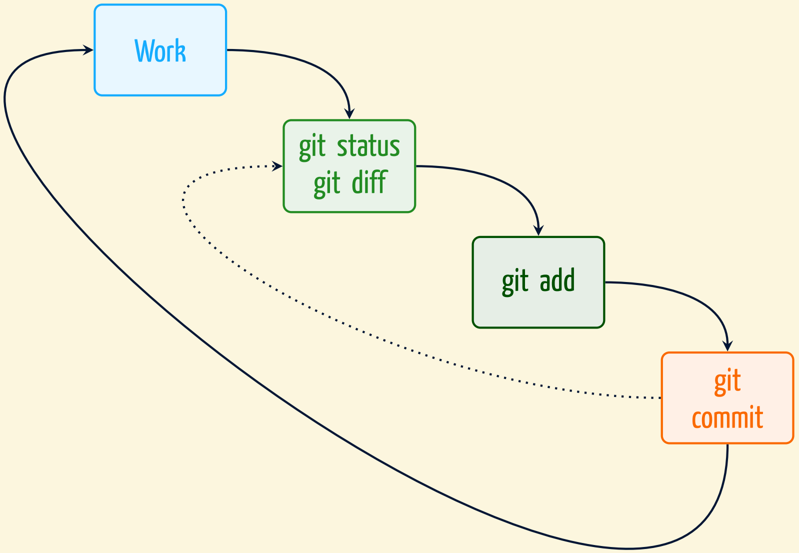
GOETHE UNIVERSITÄT FRANKFURT AM MAIN

CRC-TR 211 Strong-interaction matter under extreme conditions

Outline of the talk

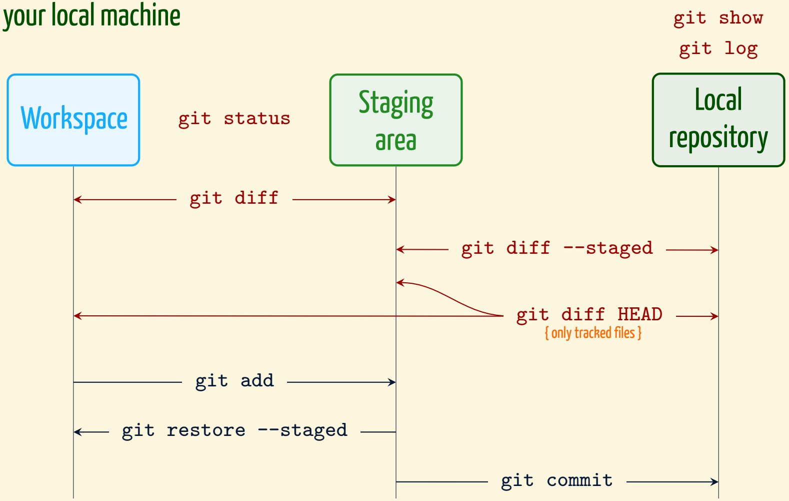
A short recap from last time

By now, this is how your workflow looks like



Our mental picture, so far

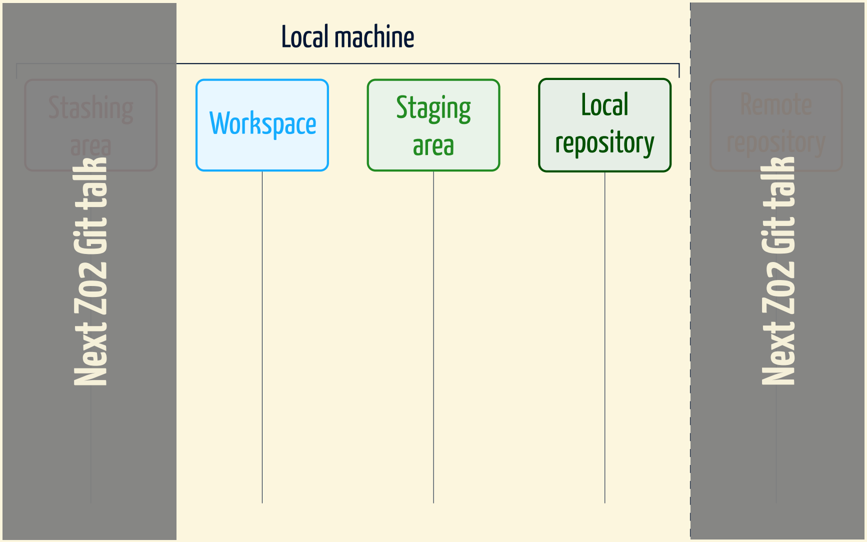
On your local machine



Commands marked in dark red do not change anything in the repository!

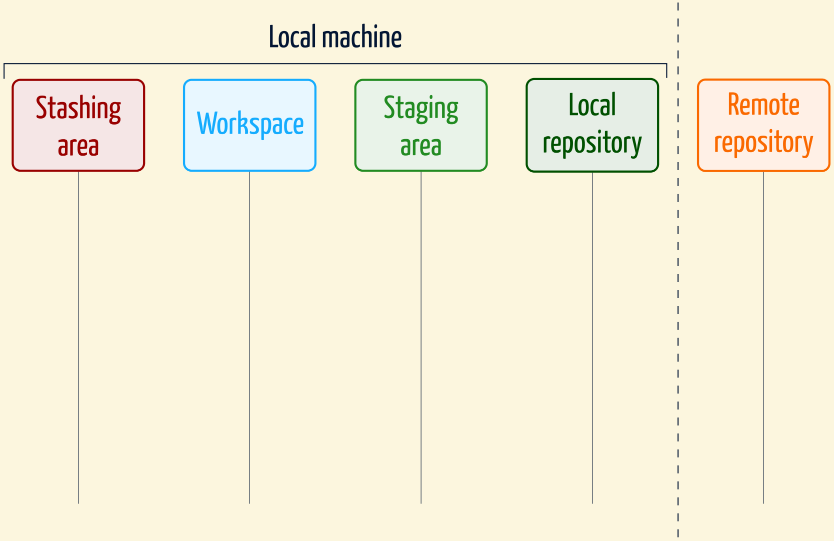
Git introduced `git-restore` in v2.23 but this stayed buggy for a while. Use it from v2.27 on, otherwise use `git-reset`.

The complete correct abstract mental setup



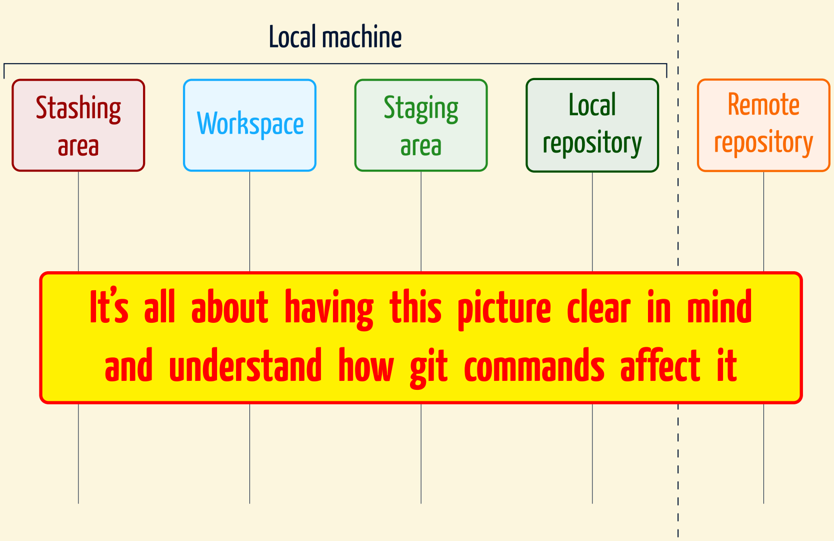
What we explored last time

The complete correct abstract mental setup



Today we'll complete the picture

The complete correct abstract mental setup



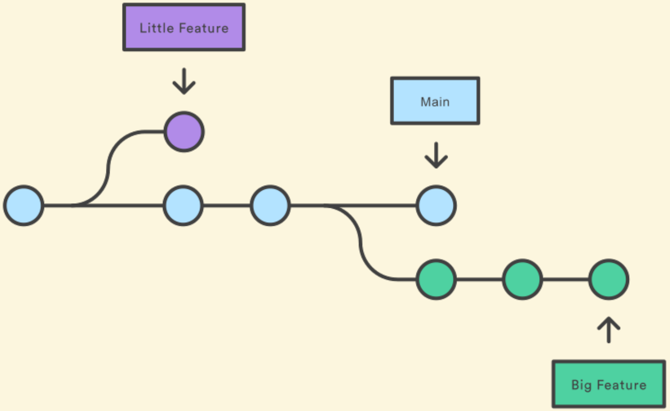
It’s all about having this picture clear in mind and understand how git commands affect it

Today we’ll complete the picture

Using branches

A key feature of Git

- Branches store **different versions of your project**
- Technically just pointers to a commit



A key feature of Git

- Branches store **different versions of your project**
- Technically just pointers to a commit
- They enable parallel development
    - Implement new features
    - Fix bugs
    - Try out something
    - [...]
- The always existing `main` branch:
    - By default created at initialization
    - Development should be done on other branches
    - Till few years ago it was called `master`

Git branch

```
# List all existing local branches
$ git branch
* main
```

```
# Create a new branch
$ git branch new-branch
$ git branch
* main
  new-branch
```

```
# Delete a branch
$ git branch -d new-branch
Deleted branch new-branch (was a45b032).
$ git branch
* main
```

Git is safe

If a modifications would be lost, Git does not allow you to delete the branch using the `-d` option. Use the `-D` option instead.

Git switch

This will in general change your workspace!

```
# Switching to another branch
$ git branch
* main
  new-branch
$ git switch new-branch
Switched to branch 'new-branch'
$ git branch
  main
* new-branch
```

Git is safe

You may switch branches with uncommitted changes in the work-tree if and only if said switching does not require clobbering those changes.

Git switch

This will in general change your workspace!

```
# Switching to another branch
$ git branch
* main
  new-branch
$ git switch new-branch
Switched to branch 'new-branch'
$ git branch
  main
* new-branch
```

```
# Creating and switching to a new branch at once
$ git switch -c another-branch
Switched to a new branch 'another-branch'
$ git branch
* another-branch
  main
```

Merging branches

- To merge means to unify the snapshots of two different branches
- This is automatically done by Git in a clever way
- When Git does not know how to merge the content of some file, it will create a conflict
- If conflicts occur, the merge will not automatically finish
- A merge can be aborted
- To fix conflicts, open and manually adjust files where Git failed

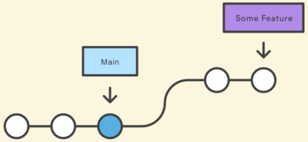
Git is safe, conflicts are not a bad thing!

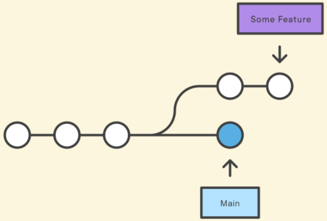
Different types of merge

Fast-forward merge

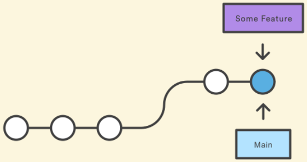
Three-way merge

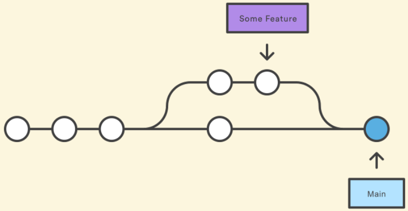
Before





After





Git merge: How does it work?

1. If possible, Git performs a fast-forward merge
2. Otherwise a three-way merge is done and a new commit created

Be sure to be on the correct branch!

```
git merge <source-branch>
```

It incorporates changes from the specified branch into the present branch!

```
# It is possible to force a three-way merge:
$ git merge --no-ff <source-branch>
```

Merge conflicts: Fixing procedure

```
# A general example
$ git merge <branch_name>
Auto-merging <file>
CONFLICT (content): Merge conflict in <file>
Automatic merge failed; fix conflicts and then commit the result.
```

1. Run `git status` to see unmerged paths
2. Find problematic hunks in files that contain conflicts
   ↳ Look for delimiters in the files: `<<<<<<<`, `=======`, `>>>>>>>`
3. Remove delimiters and adjust content
4. Check the project works (e.g. compile, run tests)
5. `git add` the files with fixed conflicts
6. Commit added files
   ↳ Git propose you an auto-generated commit message

Live example!

Working with remote repositories

A bare repository

The stashing area

The remaining git commands

That’s **NOT** all folks

1. Start using Git. **Now.** Not tomorrow or next week, today!
   $\longrightarrow$ Repeat what done on these slides

That’s **NOT** all folks

1. Start using Git. **Now.** Not tomorrow or next week, today!
   $\longrightarrow$ Repeat what done on these slides
2. Was anything unclear? Do you get stuck at some point?
   $\longrightarrow$ Drop me ✉ an email

That’s **NOT** all folks

1. Start using Git. **Now.** Not tomorrow or next week, today!
   $\longrightarrow$ Repeat what done on these slides
2. Was anything unclear? Do you get stuck at some point?
   $\longrightarrow$ Drop me ✉ an email
3. Git is much more than this!
   $\longrightarrow$ Come to next Z02 talk: «Let’s git **together**»

```
git clone
git branch
git switch
git checkout
git merge
git pull
git push
```

That's **NOT** all folks

Thank you!

1. Start using Git. **Now.** Not tomorrow or next week, today!
   → Repeat what done on these slides
2. Was anything unclear? Do you get stuck at some point?
   → Drop me an email
3. Git is much more than this!
   → Come to next Z02 talk: «Let's git **together**»

git clone
git branch
git switch
git checkout
git merge
git pull
git push

Believe me, it's worth it!