Let's Git together

Alessandro Sciarra

Z02 – Software Development Center

XX August 2022

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

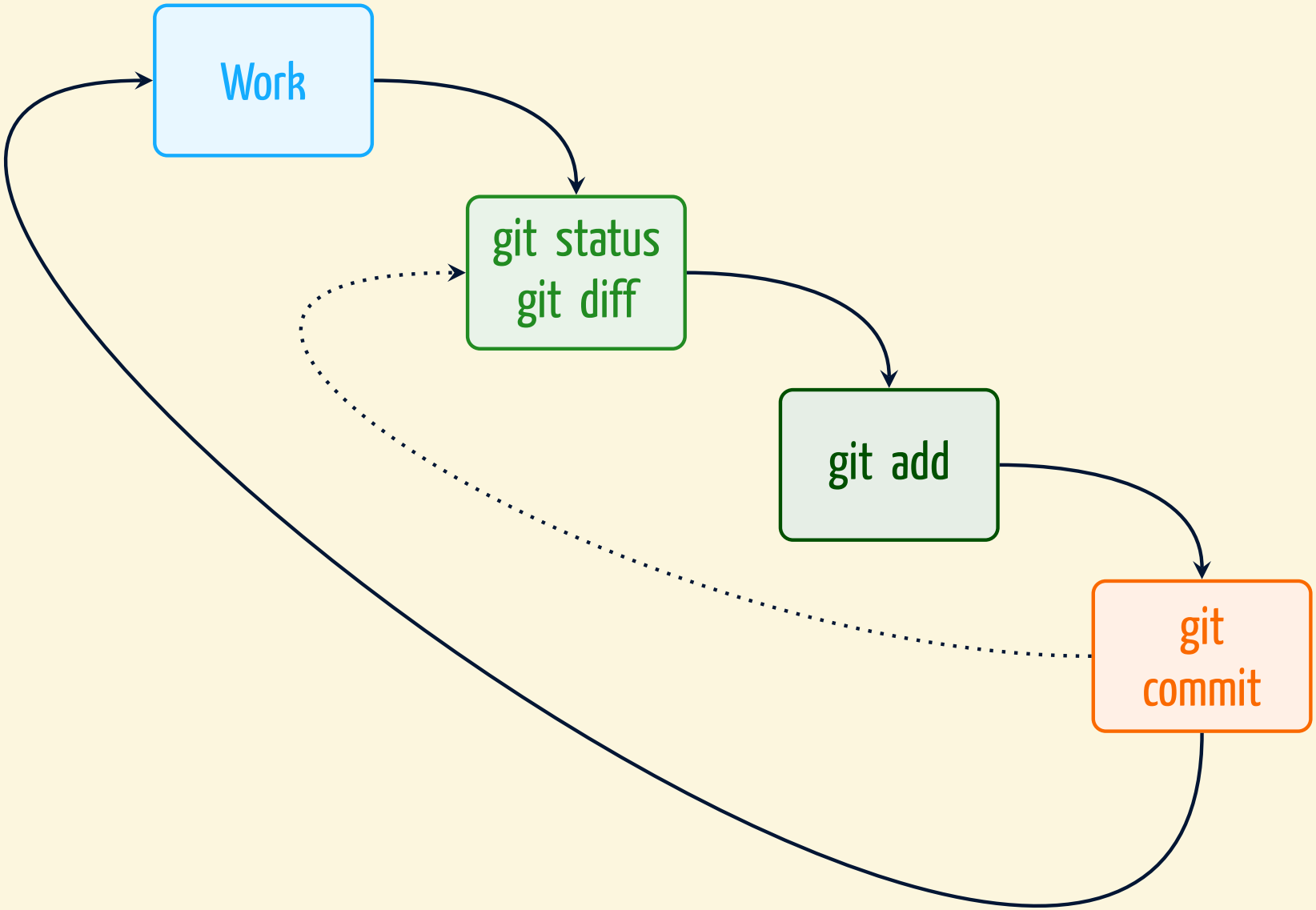
CRC-TR 211
Strong-interaction matter under extreme conditions

Outline of the talk

1. A short recap from last time
2. Using branches
3. Working with remote repositories
4. A bare repository
5. The stashing area
6. The remaining git commands

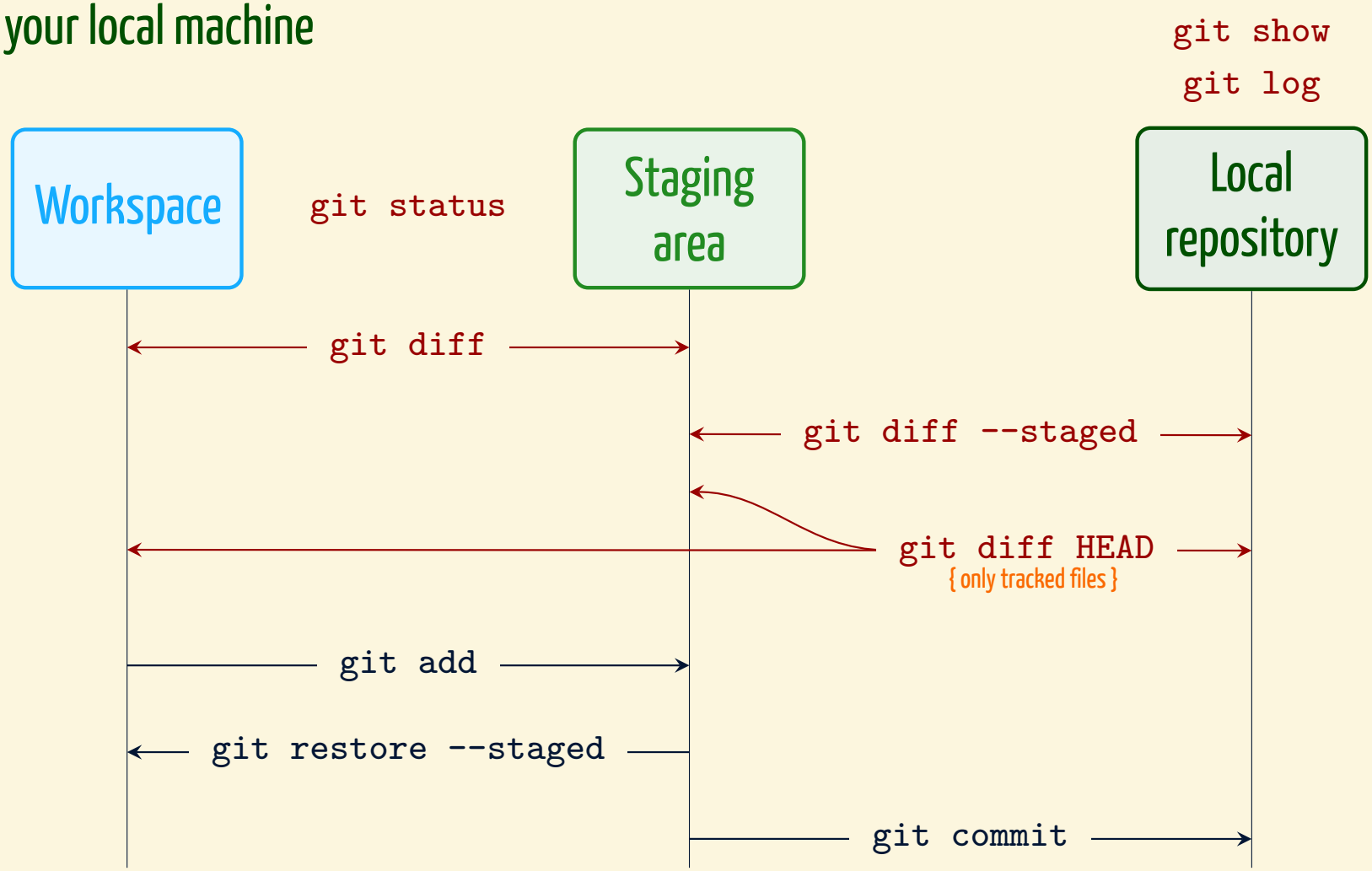
A short recap from last time

By now, this is how your workflow looks like



Our mental picture, so far

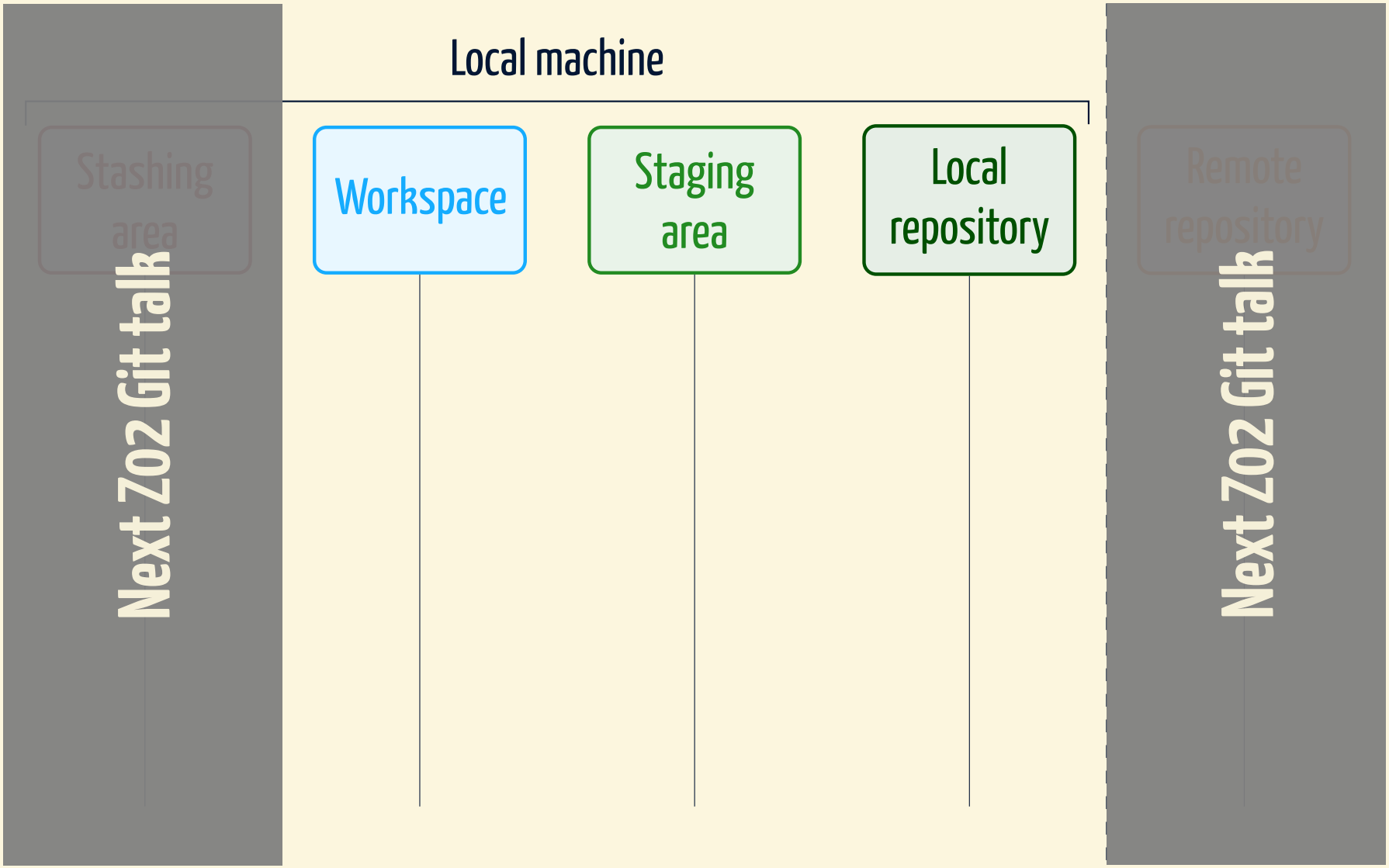
On your local machine



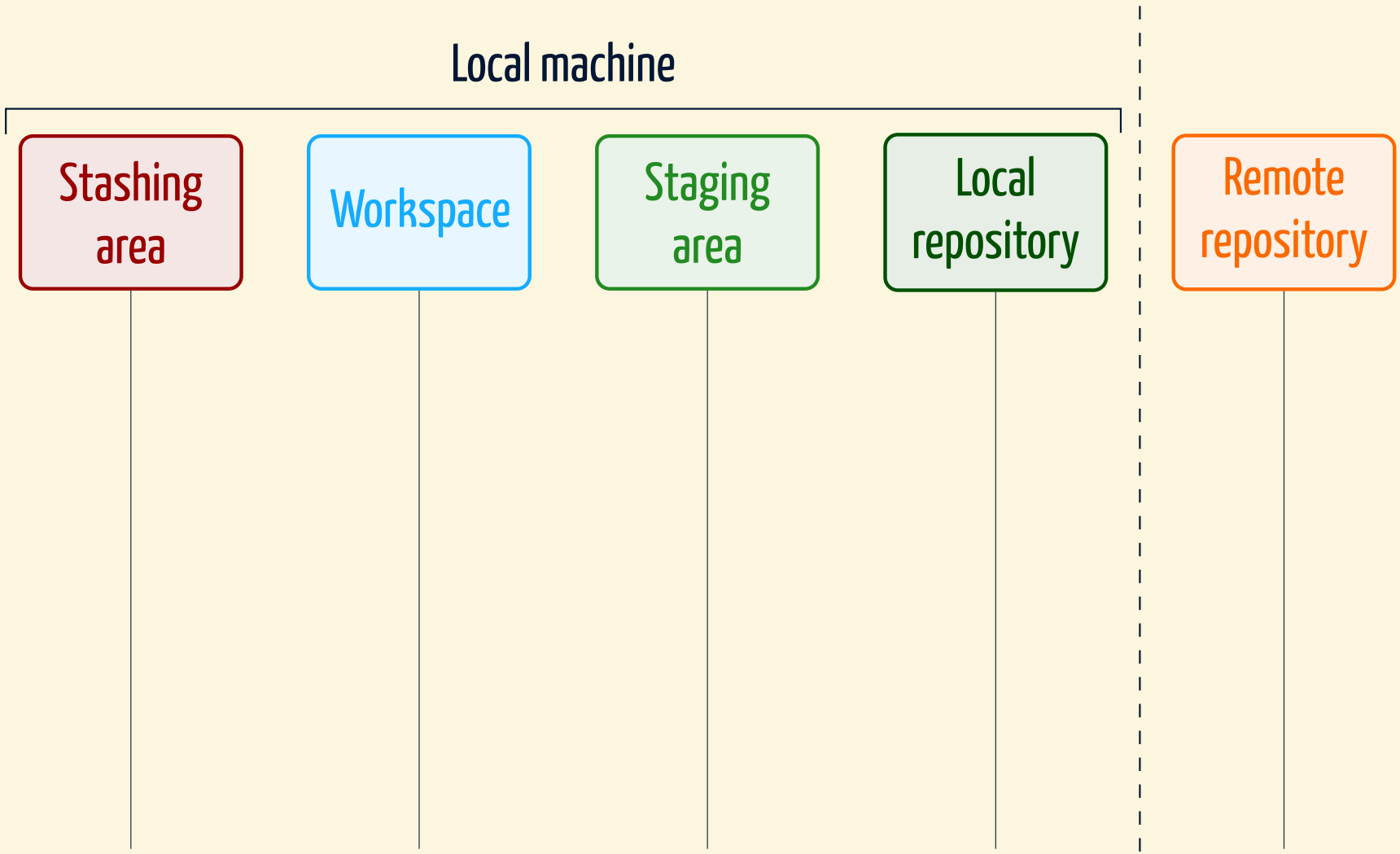
Commands marked in dark red do not change anything in the repository!

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

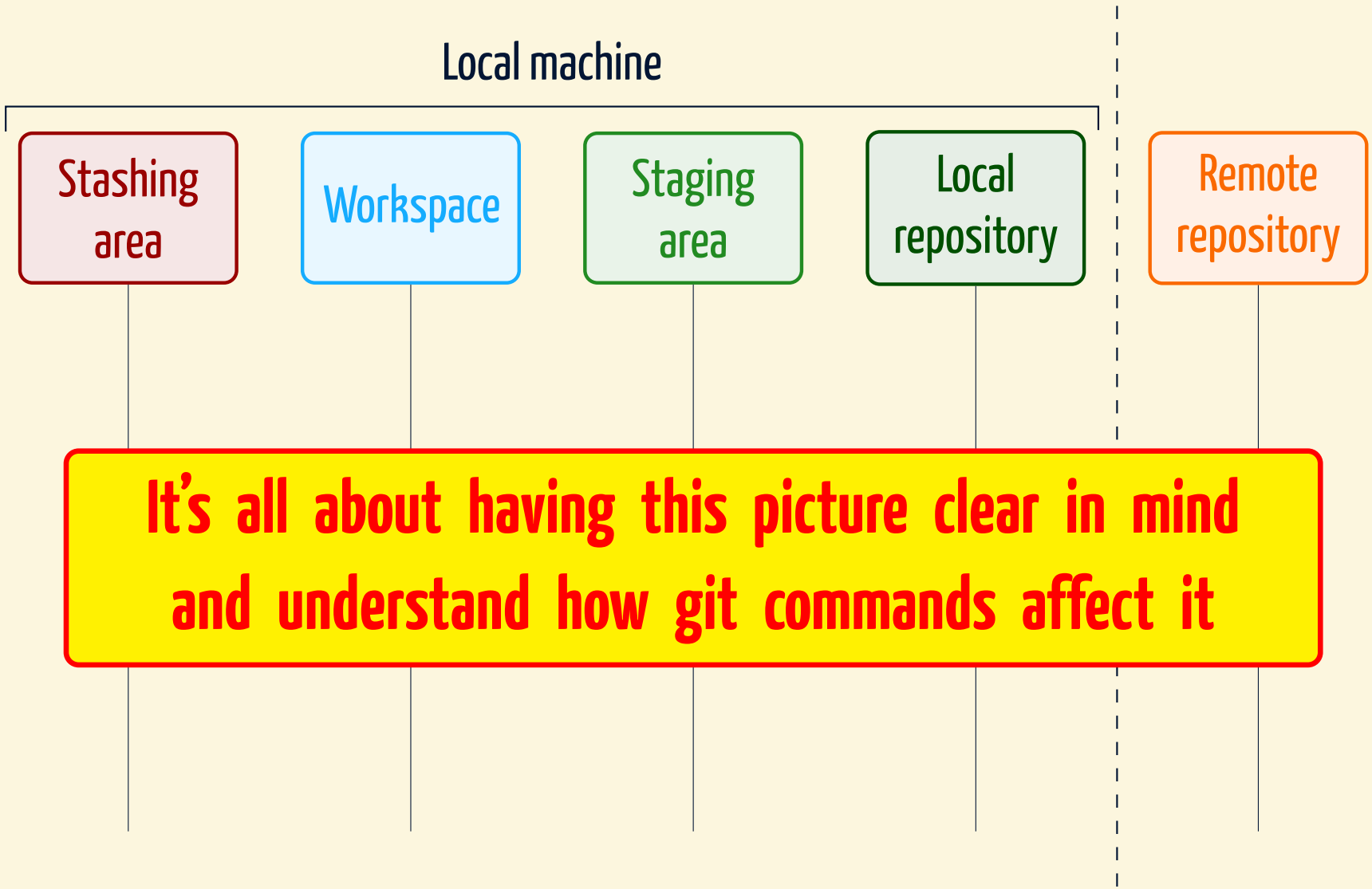


The complete correct abstract mental setup



Today we'll complete the picture

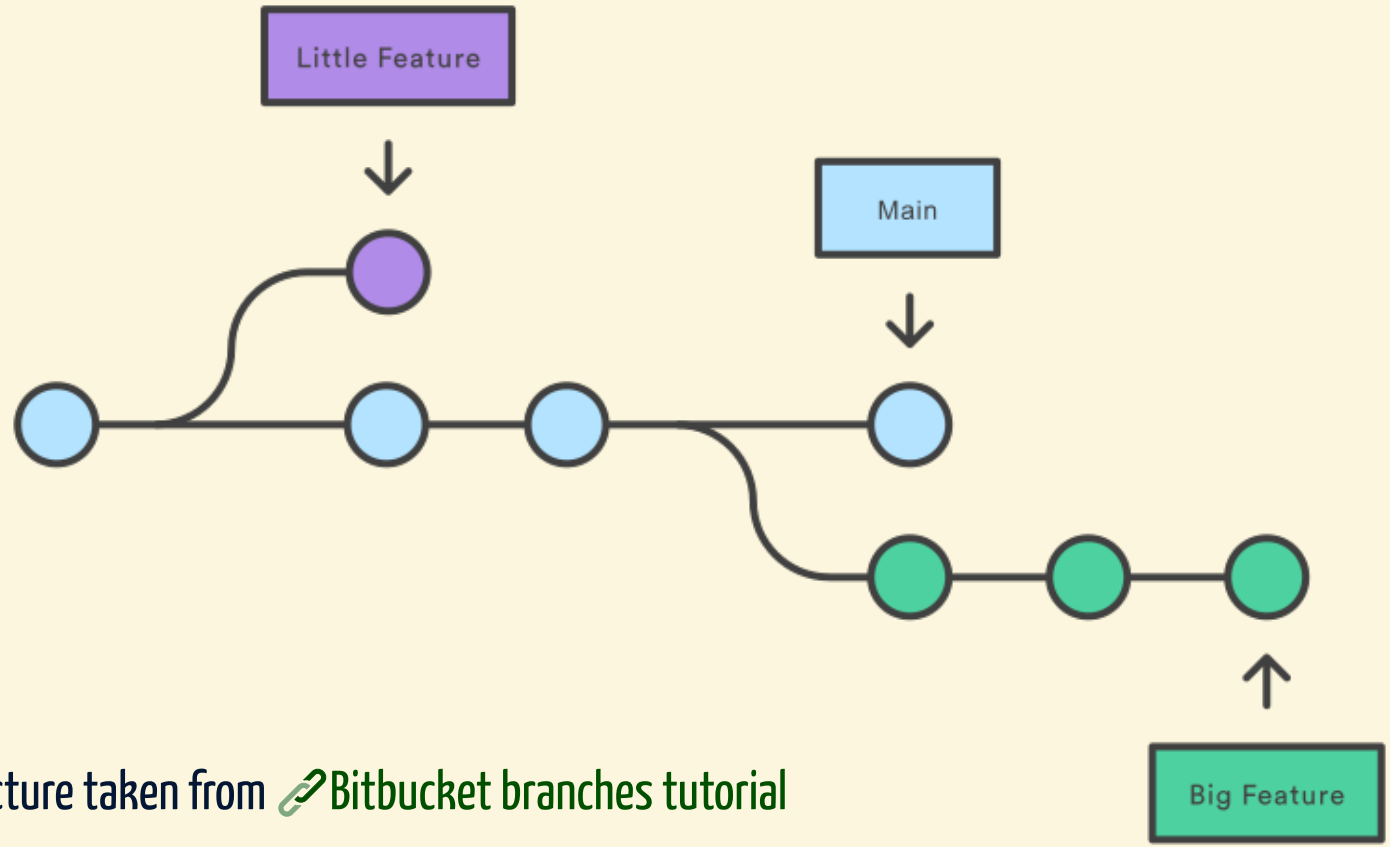
The complete correct abstract mental setup



Using branches

A key feature of Git

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



Picture taken from Bitbucket branches tutorial

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

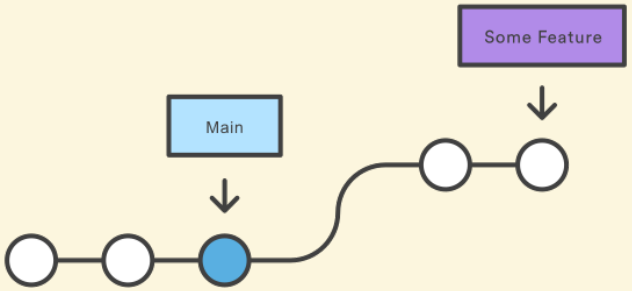
- To merge means to unify the snapshots of two different branches
- This is automatically done by Git in a clever way
- When Git does not know how to merge the content of some file, it will create a conflict
- If conflicts occur, the merge will not automatically finish
- A merge can be aborted
- To fix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

Git is safe, conflicts are not a bad thing!

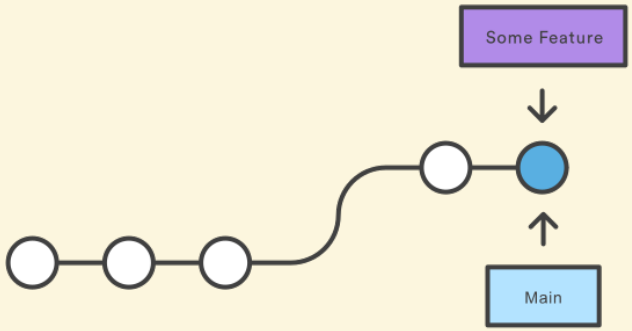
Different types of merge

Fast-forward merge

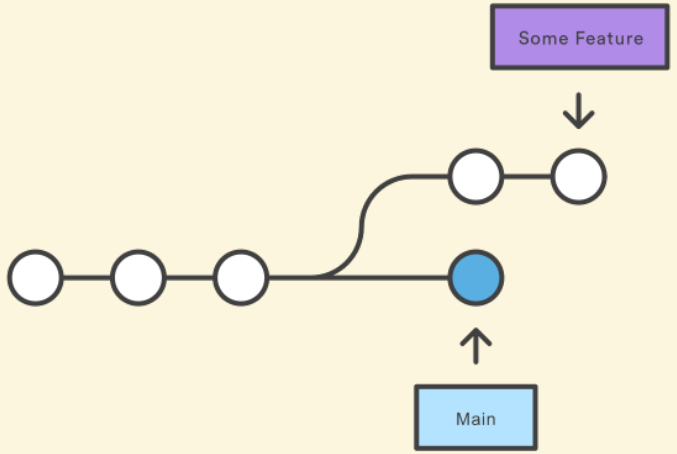
Before

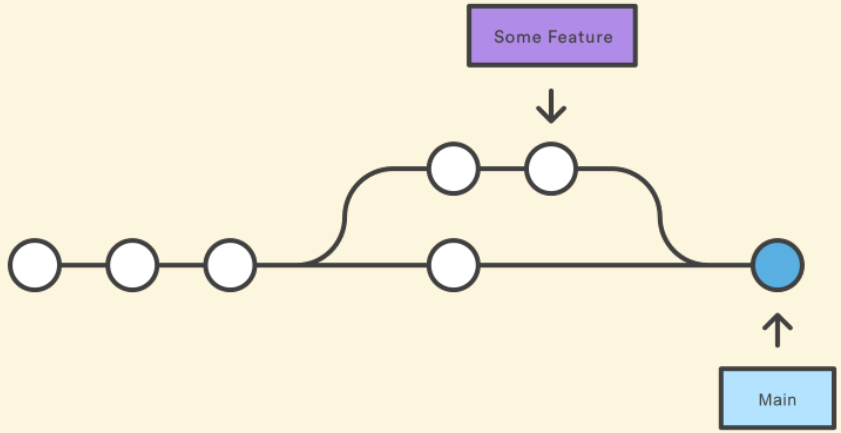


After



Three-way merge





Pictures taken from Bitbucket git-merge tutorial .

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

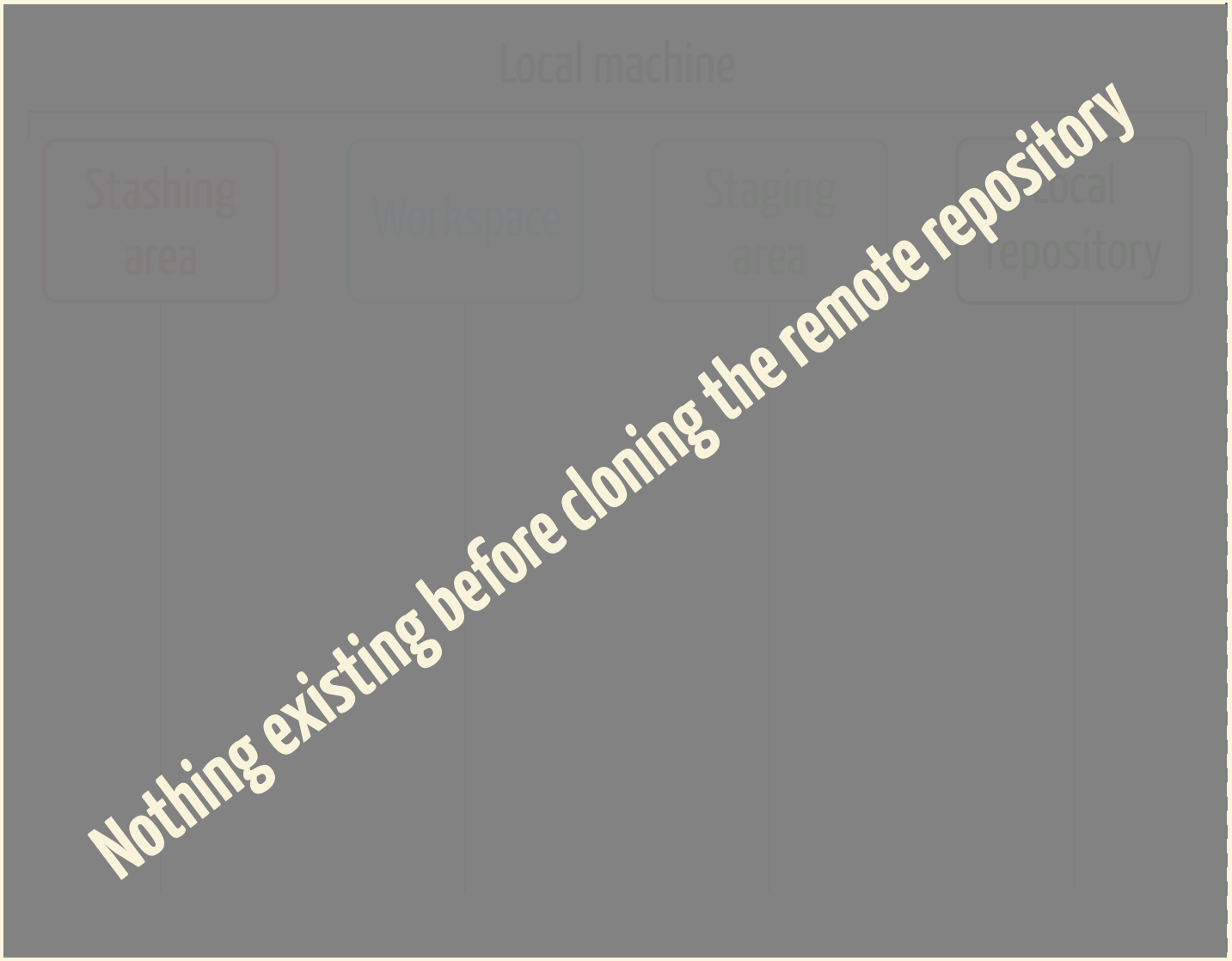
```
# A general example
$ git merge <branch_name>
Auto-merging <file>
CONFLICT (content): Merge conflict in <file>
Automatic merge failed; fix conflicts and then commit the result.
```

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

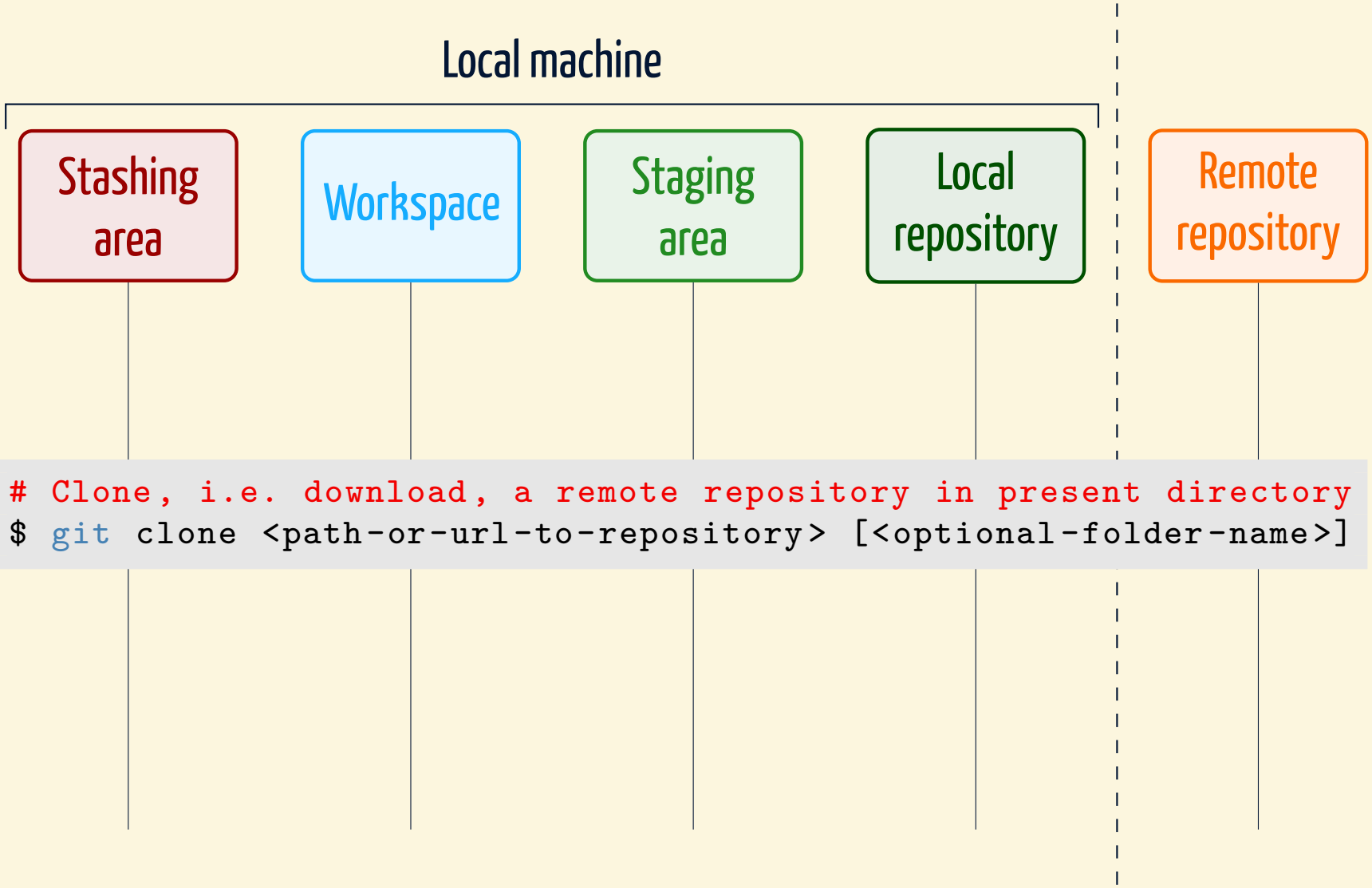
Live example!

Working with remote repositories

Cloning a remote repository



Cloning a remote repository




```
# Clone, i.e. download, a remote repository in present directory
$ git clone <path-or-url-to-repository> [<optional-folder-name>]
```

Cloning a remote repository

```
# Clone the repository of this course
$ ls
$ git clone git@github.com:AxelKrypton/Git-crash-course.git
Cloning into 'Git-crash-course'...
remote: Enumerating objects: 82, done.
remote: Counting objects: 100% (66/66), done.
remote: Compressing objects: 100% (47/47), done.
remote: Total 82 (delta 26), reused 52 (delta 18), pack-reused 16
Receiving objects: 100% (82/82), 4.47 MiB | 4.49 MiB/s, done.
Resolving deltas: 100% (29/29), done.
$ ls
Git-crash-course
```

The local repository is aware of the remote one!

Git remote

In the local repository, the remote one is (by default) referred as **origin**

```
# Check out the remote information
$ cd Git-crash-course
$ git remote
origin
$ git remote -v
origin    git@github.com:AxelKrypton/Git-crash-course.git (fetch)
origin    git@github.com:AxelKrypton/Git-crash-course.git (push)
$ git remote rename origin GitHub
$ git remote
GitHub
# A repository can have multiple remote locations
$ git remote add MyServer <url-to-new-remote>
$ git remote
GitHub
MyServer
```

First interactions with the remote repository

```
$ git branch -r
  origin/HEAD -> origin/main
  origin/main
  origin/experiment
$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
  remotes/origin/experiment
```

```
# Switch to a new branch that mirrors the state of a remote one
$ git switch experiment
Branch 'experiment' set up to track remote branch 'experiment'
    from 'origin'.
Switched to a new branch 'experiment'
$ git branch -vv
* experiment    a1d62e63 [origin/experiment] last-commit-message
  main          a1d62e63 [origin/main] last-commit-message
```

We'll come back to the idea of tracking in a moment!

Fetching and pulling

When collaborating in a project, the remote repository will in general change because of other people's work

```
$ git fetch <remote-name>
```

- Information about the remote repository (e.g. branches) can be updated by fetching from a remote
- Fetching does not change the local workspace!

Fetching and pulling

When collaborating in a project, the remote repository will in general change because of other people's work

```
$ git pull <remote-name> <remote-branch-name>
```

- Pulling instead is updating both the information about the remote repository and the local workspace
- Git pull is actual a shortcut to do a fetch followed by a merge with a remote branch

Fetching and pulling: Examples

```
# If there is only one remote, you can omit it
$ git fetch origin
a1e8fb5..45e66a4 main -> origin/main
a1e8fb5..9e8ab1c develop -> origin/develop
* [new branch] some-feature -> origin/some-feature
# To remove locally references to remote deleted branches:
$ git fetch --prune
From github.com:AxelKrypton/Git-crash-course
- [deleted]           (none)     -> origin/experiment
```

```
# Be sure to be on the correct branch before pulling
$ git branch
* main
$ git pull origin main
From github.com:AxelKrypton/Git-crash-course
* branch            main       -> FETCH_HEAD
Already up to date.
```


These 3-way merge commits because of pulling operations are not ideal and next time we'll learn how to avoid them.

Fetching and pulling: Examples

If you created commits on the present branch, pulling it from remote will perform a merge and, if this is not a fast-forward merge, the editor to make a commit with an auto-generated message will be displayed to you. Conflicts might occur as well.

```
$ git log --oneline
a45b032 (HEAD -> main) Some work done locally on main
6e5ea4b (origin/main, origin/HEAD) Last commit pulled
236d4af Previous history
# If I now pull and some new commit exists after 6e5ea4b
#  => a 3-way merge occurs and the editor will open
$ git pull origin master
[...]
```


These 3-way merge commits because of pulling operations are not ideal and next time we'll learn how to avoid them.

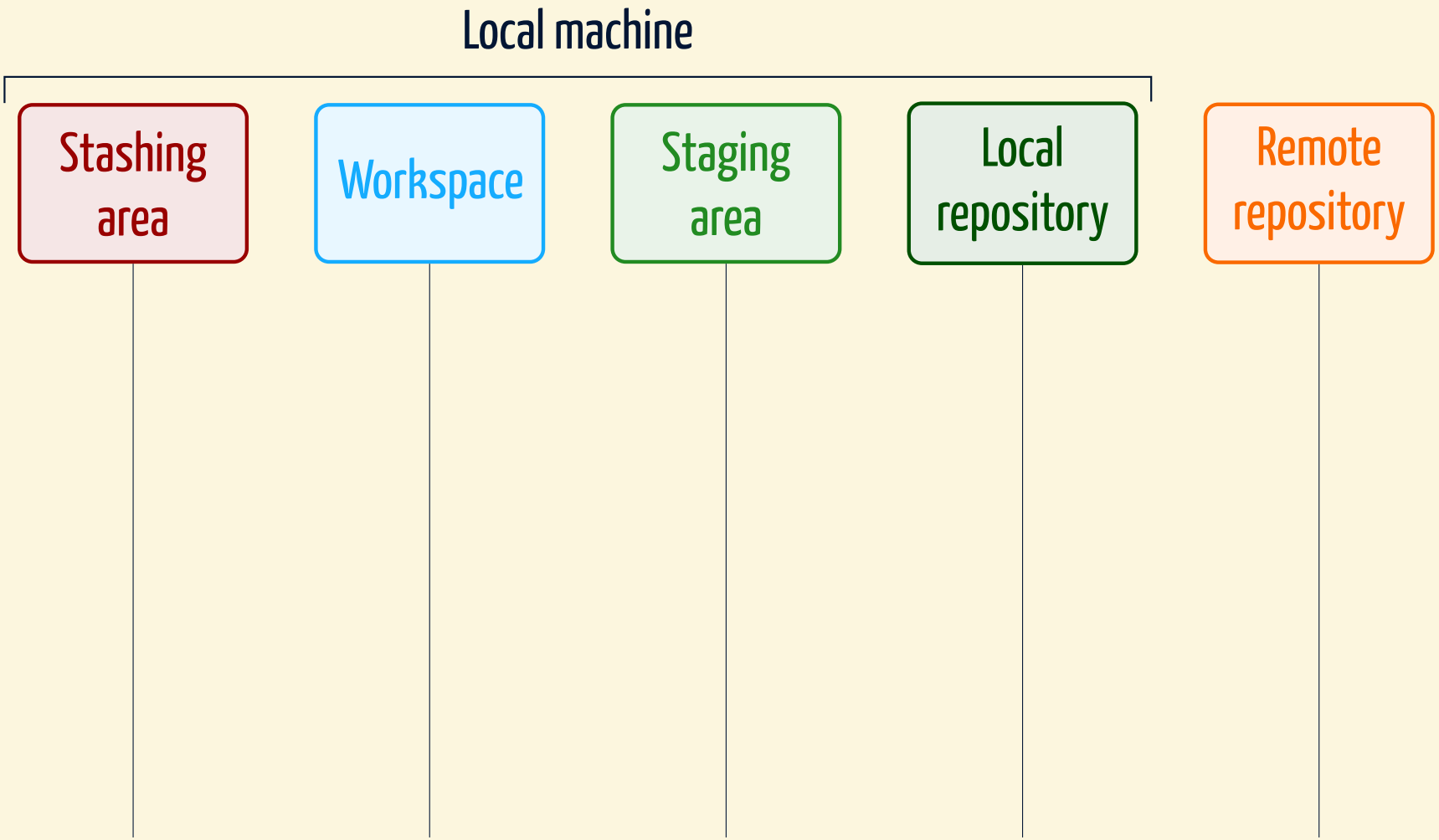
Pushing your own work

- To push means to make the changes done in the local repository available in the remote one, i.e. to update a remote branch with a local one
- Only changes that are committed are pushed
- If the remote and the local history diverge (i.e. you forgot to pull before committing), the push operation will be rejected

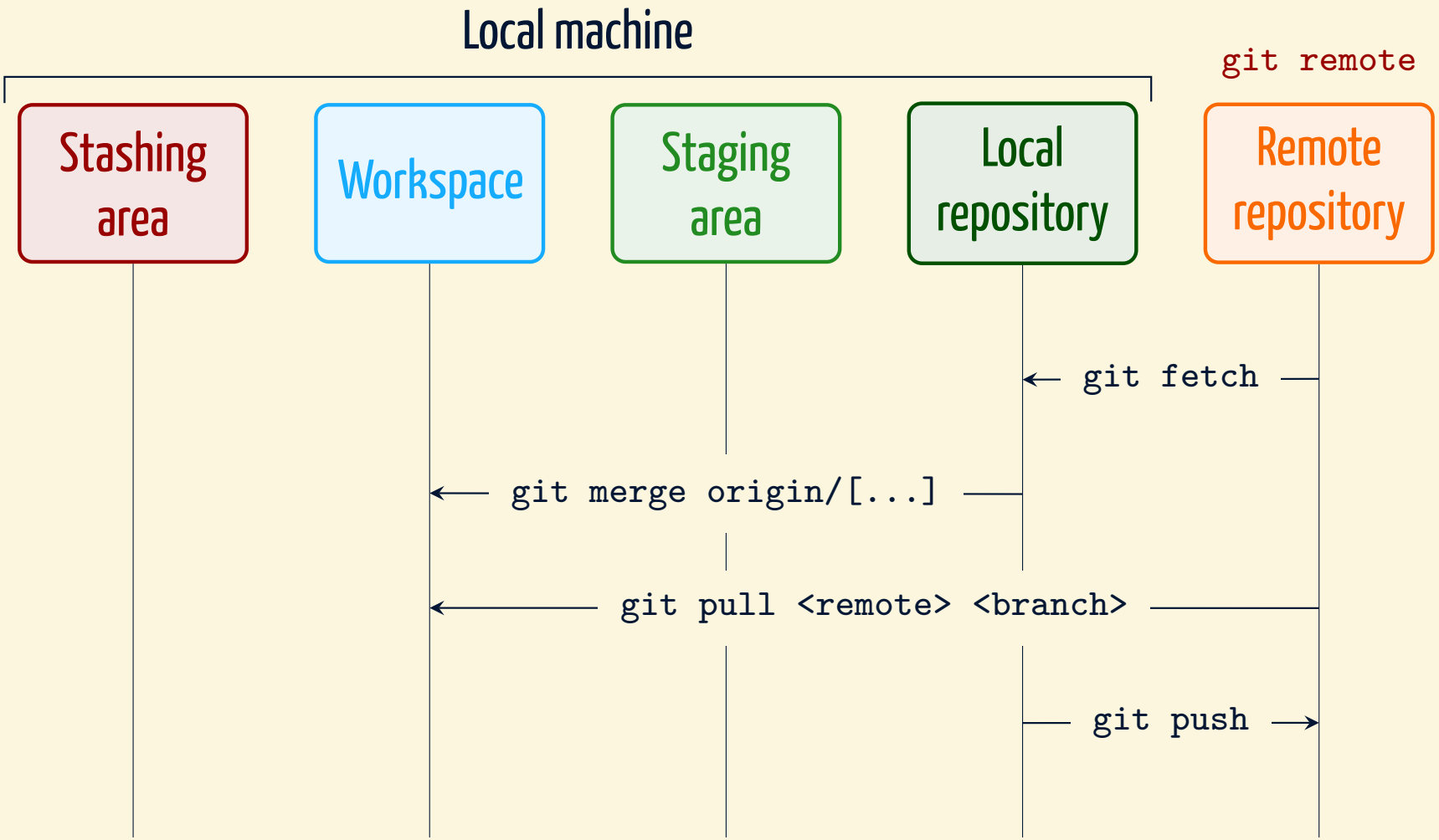
Push before saying to your collaborators you did some changes!

```
$ git push origin main
[...]
# To delete remote branches 'git push' is also used:
$ git push <remote-name> --delete <branch_name>
```

How does this fit into our mental picture?



How does this fit into our mental picture?



Keep in mind that it is very tough (if not impossible) to undo these commands.

Few words about tracking branches

A convenient feature

It is possible to make a default association between a local and a remote branch which is used for pull and push operations!

```
$ git branch -a -vv
  exp1                 6e5ea4b Commit msg
  exp2                 6e5ea4b Commit msg
* main                 6e5ea4b [origin/main] Commit msg
  remotes/origin/HEAD -> origin/main
  remotes/origin/main 6e5ea4b Commit msg
  remotes/origin/exp1 6e5ea4b Commit msg
```

```
# First possibility (when remote branch already exists)
$ git switch exp1
Switched to branch 'exp1'
$ git branch --set-upstream-to=origin/exp1
Branch 'exp1' set up to track remote branch 'exp1' from 'origin'.
```

Few words about tracking branches

A convenient feature

It is possible to make a default association between a local and a remote branch which is used for pull and push operations!

```
# Second possibility (even without remote branch, yet)
$ git switch exp2
Switched to branch 'exp2'
$ git push -u origin exp2
[...]
To github.com:AxelKrypton/Git-crash-course.git
* [new branch]      exp2 -> exp2
Branch 'exp2' set up to track remote branch 'exp2' from 'origin'.
```

```
# Check branch tracking associations
$ git branch -vv
  exp1              6e5ea4b [origin/exp1] Commit msg
* exp2              6e5ea4b [origin/exp2] Commit msg
  main              6e5ea4b [origin/main] Commit msg
```

Few words about tracking branches

A convenient feature

It is possible to make a default association between a local and a remote branch which is used for pull and push operations!

Between tracking branches

It is then possible to simply use **git pull** and **git push** commands!

```
# Check branch tracking associations
$ git branch -vv
  exp1              6e5ea4b [origin/exp1] Commit msg
* exp2              6e5ea4b [origin/exp2] Commit msg
  main              6e5ea4b [origin/main] Commit msg
```

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!
   → Repeat what done on these slides

That's **NOT** all folks

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

That's **NOT** all folks

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
```

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!