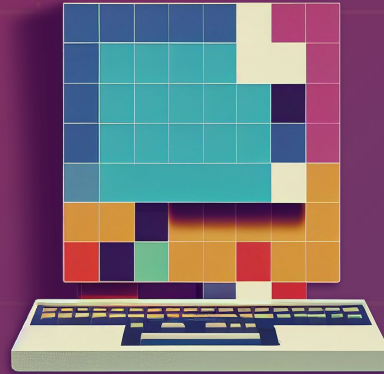


*SCAN ME*



# Software Development Technologies

13 January 2025

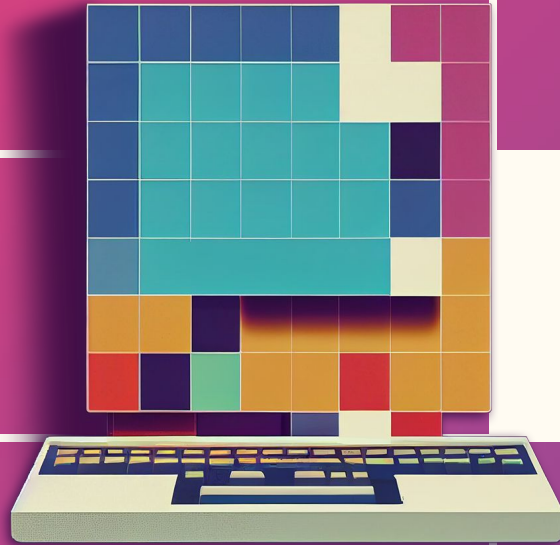


## **Errahel Nazih**

Senior Lecturer at BRICS School, Irkutsk National  
Research Technical University

**Email:** erraheln@ex.istu.edu / naziherrahel@gmail.com

**Phone:** +7 999 686 47 49



# Introduction

n

*Overview*

- **Learn to develop** and operate AI-enabled (prediction) services
- **Develop and run** feature pipelines
- **Deploy features and models**
- **Train models** and and run inference pipelines
- **Develop a UI** for your prediction service
- **Learn MLOps fundamentals:** versioning, automated testing, data validation, and operations
- **Develop and run** a real-time machine learning system



## Module 01

the basics of **Git**, a version control system for tracking changes, collaborating on projects, and managing code efficiently.

## Module 02

Pandas and ML Pipelines in Python

## Module 03

Feature Store and Model Registry. Credit-card fraud prediction service.

## Module 04

Training Pipelines and Inference Pipelines

## Module 05

Bring a Prediction Service to Life with a User Interface

## Module 06

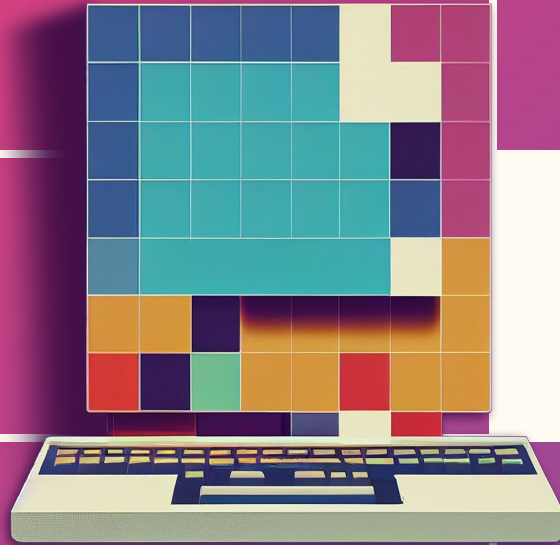
Automated Testing and Versioning of features and models

## Module 07

Real-time machine learning systems. Project presentation.

- **Exercises** - There will be homework exercises at the end of each module, and a final project at the end of the course.
- **Assessment**  
Presentations  
Scores for homework and the project make up your mark





# Introduction To Git



- 1 Why should you use it?
- 2 What is Git?
- 3 How to use Git locally?
- 4 Summary and conclusions

**Why should you use it?**

**OK, let's do it without git**

# A possible workflow

Writing a review or a thesis

## "FINAL".doc



FINAL.doc!



FINAL\_rev.2.doc



FINAL\_rev.6.COMMENTS.doc



FINAL\_rev.8.comments5.  
CORRECTIONS.doc



FINAL\_rev.18.comments7.  
corrections9.MORE.30.doc



FINAL\_rev.22.comments49.  
corrections.10. #@\$%WHYDID  
ICOMETOGRADSCHOOL?????.doc

JORGE CUAM © 2012

# A possible workflow

Writing a review or a thesis

- How do you make writing experiments?

# A possible workflow

Writing a review or a thesis

- **How do you make writing experiments?**
  - You make a backup of your file
  - You comment out a block of text in your source
  - If the old version was better, you restore it by hand
  - If the new version is better, you clean up by hand



# A possible workflow

Writing a review or a thesis

- How do you make writing experiments?
- How do you create/view checkpoints?

# A possible workflow

Writing a review or a thesis

- How do you make writing experiments?
- How do you create/view checkpoints?
  - Create a .tar or .zip file
  - Copy it somewhere and uncompress if needed

# A possible workflow

Writing a review or a thesis

- How do you make writing experiments?
- How do you create/view checkpoints?
- Which version did you send to your supervisor/colleagues?

# A possible workflow

Writing a review or a thesis

- How do you make writing experiments?
- How do you create/view checkpoints?
- Which version did you send to your supervisor/colleagues?
  - Put a copy of the PDF file or of the compressed folder somewhere
  - Keep the sent email for later use

# A possible workflow

Writing a review or a thesis

- How do you make writing experiments?
- How do you create/view checkpoints?
- Which version did you send to your supervisor/colleagues?
- How long did it take to write this section?
- When did I start writing this chapter?
- How much did I write on average per day?

# A possible workflow

Writing a review or a thesis

- How do you make writing experiments?
- How do you create/view checkpoints?
- Which version did you send to your supervisor/colleagues?
- How long did it take to write this section?
- When did I start writing this chapter?
- How much did I write on average per day?

**Everything by hand, error-prone and big overhead!**

# Another possible workflow

Collaborating on a project

- How can you collaborate on the same project with colleagues?

# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
  - You work on separate parts at the same time
  - Only one person works at the same time



# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- How do you merge work from other people in the team?

# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- **How do you merge work from other people in the team?**
  - You send the changed files per email and put them in the folder by hand
  - Copy/Rsync in some shared place the new status of the project
  - If only one person works at once, a compressed archive can be exchanged

# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- How do you merge work from other people in the team?
- How do you work on different machines?

# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- How do you merge work from other people in the team?
- **How do you work on different machines?**
  - You don't, use SSH
  - Different machines are as different people, see above

# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- How do you merge work from other people in the team?
- How do you work on different machines?
- How do you know who did what?

# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- How do you merge work from other people in the team?
- How do you work on different machines?
- **How do you know who did what?**
  - This information is not important
  - Sending work around per email allows to trace this...
  - Put comments into the source!

# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- How do you merge work from other people in the team?
- How do you work on different machines?
- How do you know who did what?
- How do you give credit to authors?

# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- How do you merge work from other people in the team?
- How do you work on different machines?
- How do you know who did what?
- **How do you give credit to authors?**
  - Detailed information is not important
  - A rough idea about who worked on what is enough
  - See comments into the source!



# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- How do you merge work from other people in the team?
- How do you work on different machines?
- How do you know who did what?
- How do you give credit to authors?
- How do you go back in history e.g. in case of a bug?

# Another possible workflow

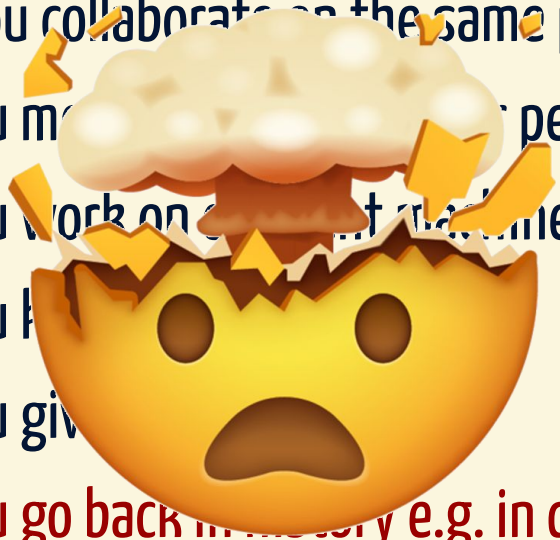
## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- How do you merge work from other people in the team?
- How do you work on different machines?
- How do you know who did what?
- How do you give credit to authors?
- **How do you go back in history e.g. in case of a bug?**
  - Again, use the archives sent around per email
  - Using a shared place, this is not possible → debug!

# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?
- How do you manage the workflow of people in the team?
- How do you work on a project at different times?
- How do you keep track of the progress?
- How do you give feedback?
- **How do you go back in history e.g. in case of a bug?**
  - Again, use the archives sent around per email
  - Using a shared place, this is not possible → debug!



**OK, and how would it be with Git?**

# A possible workflow

## Writing a review or a thesis

- How do you make writing experiments?
  - Just do them (staging/stash area)
  - git-branch
- How do you create/view checkpoints?
  - git-log    git-tag    git-checkout
- Which version did you send to your supervisor/colleagues?

- git-log    git-tag

- How long did it take to write this section?

- When did I start writing this chapter?

- How much did I write on average per day?

} Git-shortlog  
git-log  
git-log gitstats<sup>◁</sup>

# Another possible workflow

## Collaborating on a project

- How can you collaborate on the same project with colleagues?

- `git-pull`   `git-push`   `git-branch`

- How do you merge work from other people in the team?

- `git-merge`

- How do you work on different machines?

- `git-pull`   `git-push`

- How do you know who did what?

- `git-blame`

- How do you give credit to authors?

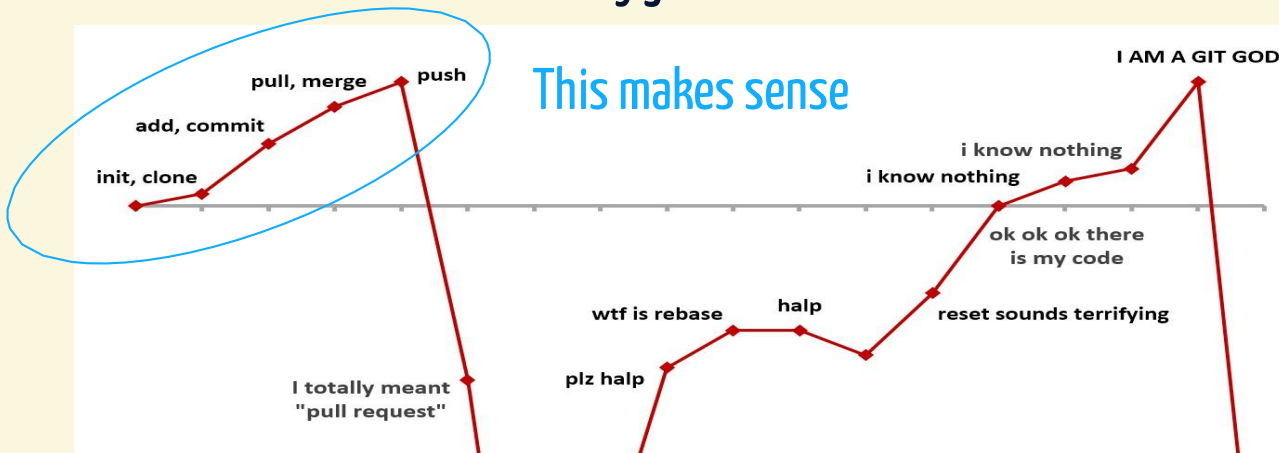
- `git-shortlog`

- How do you go back in history e.g. in case of a bug?

- `git-checkout`   `git-bisect`

# Yes, but I have to learn all those commands!

There are many jokes on the web...



## ...but after all it is about having the correct mental set up!

# Yes, but I have to learn all those commands!

- As any new tool, it needs some practice
- The short- to long-term payoff is worth the effort
- It is plenty of GUI clients
  - Sourcetree: A Free GIT Client For Windows And Mac
  - Guitar: Portable { Windows, Mac & Linux }
  - Git-Cola: Powerful GUI For GIT { Windows, Mac, Ubuntu & Linux }
  - [...]
- You can work in the terminal  
→ after this course it will be possible and straightforward!



# Last but not least



Which large famous products are developed using Git?

Linux, Homebrew, Windows, Tensorflow, Angular, Inkscape, ...

# Last but not least



Which large famous products are developed using Git?

Linux, Homebrew, Windows, Tensorflow, Angular, Inkscape, ...

And if I do not have so large projects?

It doesn't matter! There are too many advantages<sup>†</sup> having a project under a source code management tool. Even alone.

**Simply use one (Git). Now.**

For collaborative projects like maintaining code in a group, handing it over from person to person and so on, Git is simply a must. **As project leader, you should think about requiring everybody to work in a Git repository.**

# What is Git?

# How does Git define itself?

«Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Git is easy to learn and has a tiny footprint with lightning fast performance.»

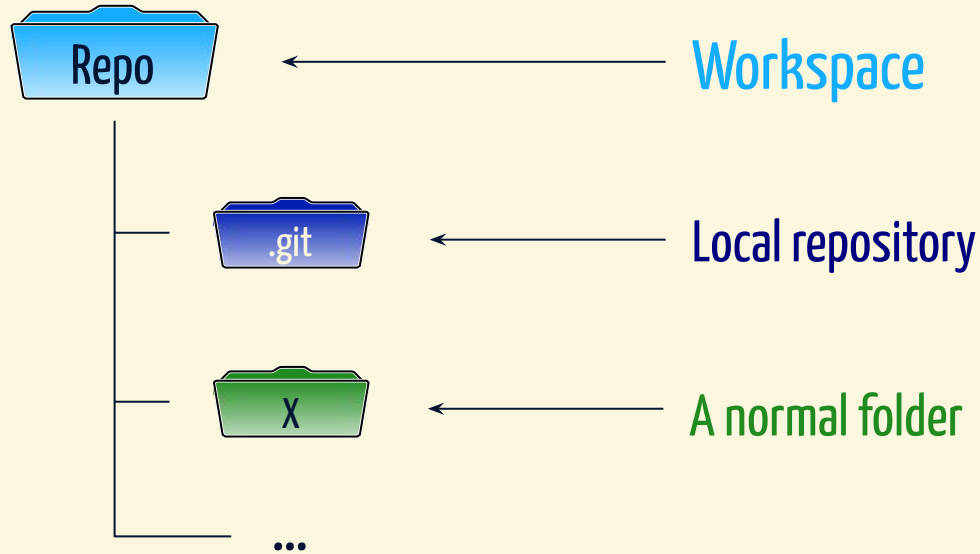


[Git homepage](https://git-scm.com/)

- Free and open
- <sub>2</sub> **Distributed version control system**
- <sub>3</sub> **From small to very large projects**
- <sub>4</sub> **With speed and efficiency**
- <sub>5</sub> **Easy to learn**

# How does it work?

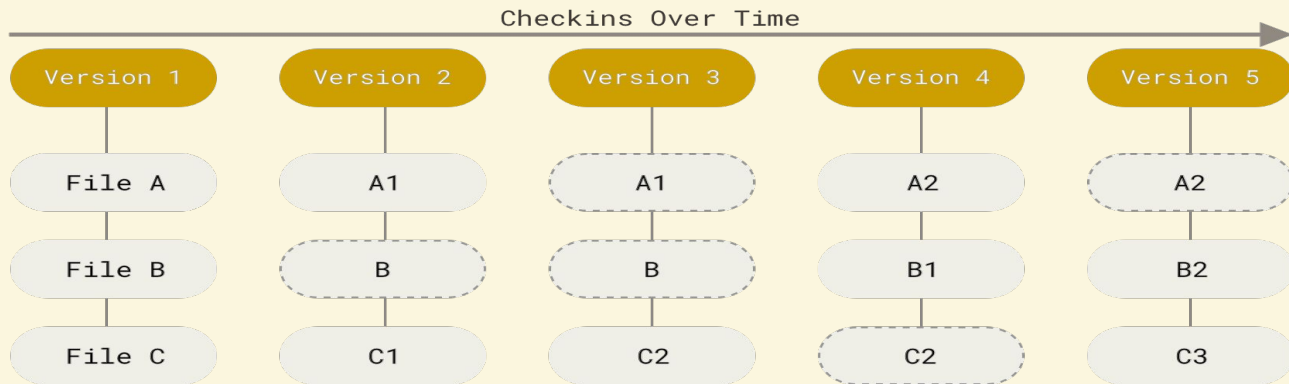
- **Repository:** a database containing all versions of the files



# How does it work?

- **Repository:** a database containing all versions of the files
- **Snapshot-based system**
  - Snapshots are called commits
  - Commits are named by checksums (also used to ensure data integrity)

{ It's impossible to change the contents of any file or directory without Git knowing about it }



[From the Git-Book](#)


# How does it work?

- **Repository:** a database containing all versions of the files
- Snapshot-based system
  - Snapshots are called commits
  - Commits are named by checksums (also used to ensure data integrity)

{ It's impossible to change the contents of any file or directory without Git knowing about it }

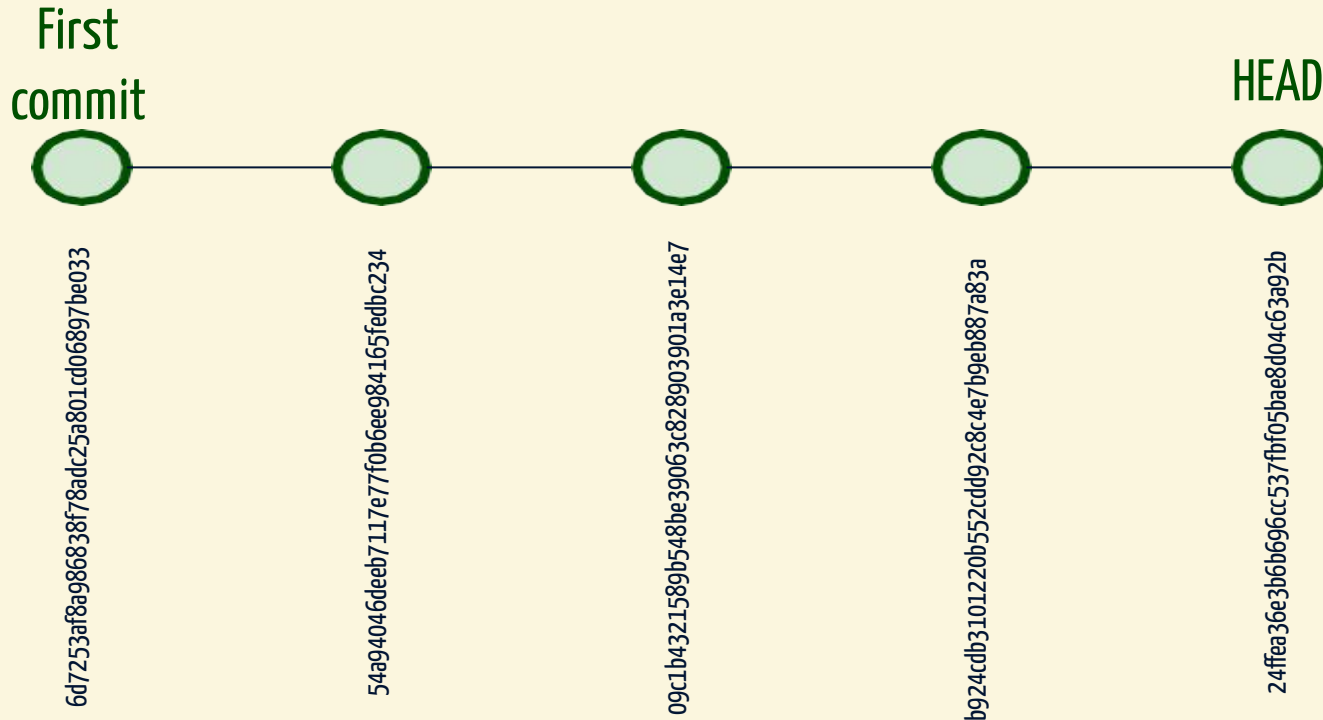
- Almost every operation is local
  - Working without network connecting
  - Distributed system → everyone carries a backup!

Are you curious to know how Git works bottom-up?

Refer to  [this 31-pages document](#) , well written, but not needed at start.

# An example of Git history

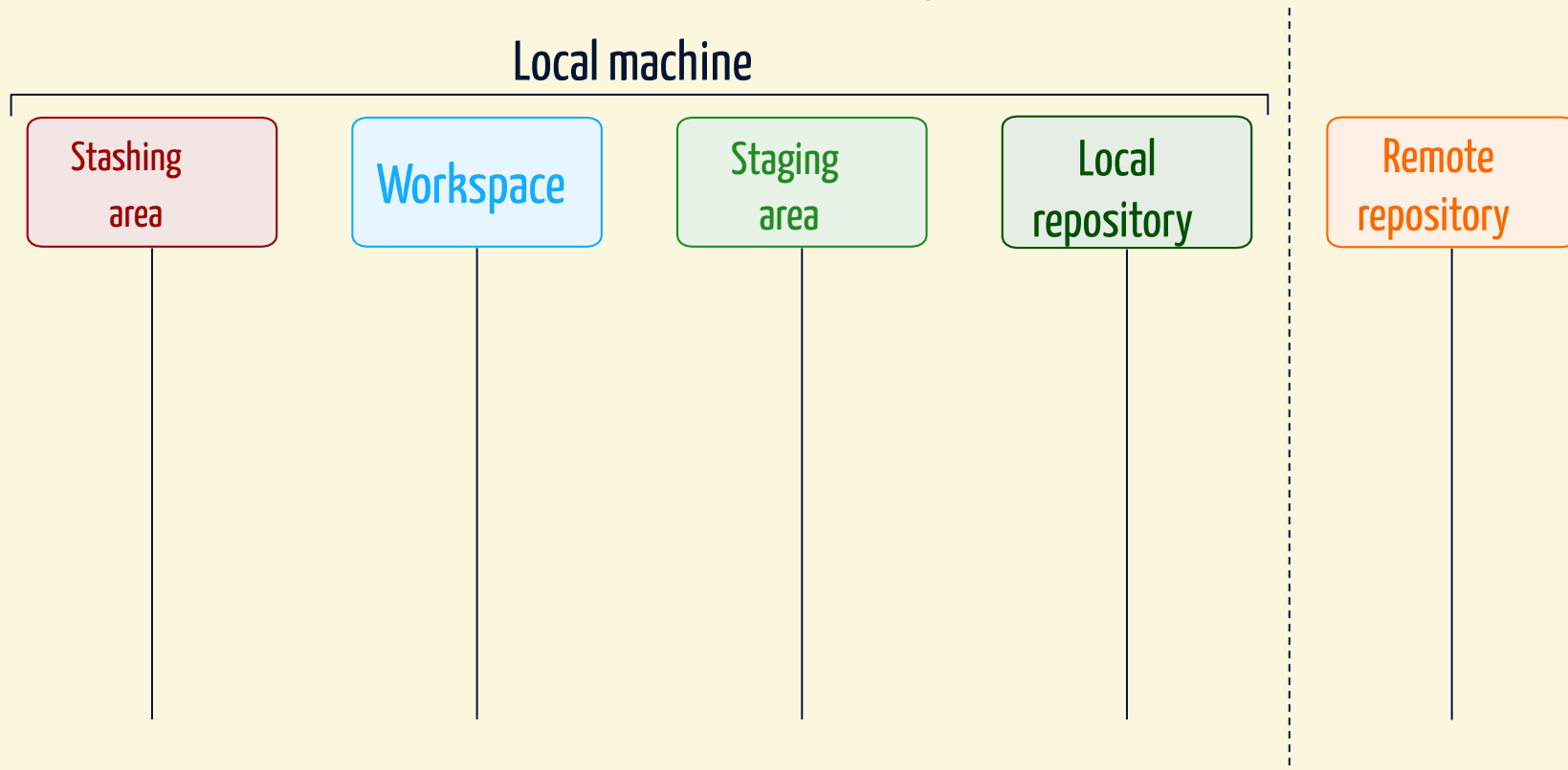
Every commit is a snapshot of the state of the repository at that point



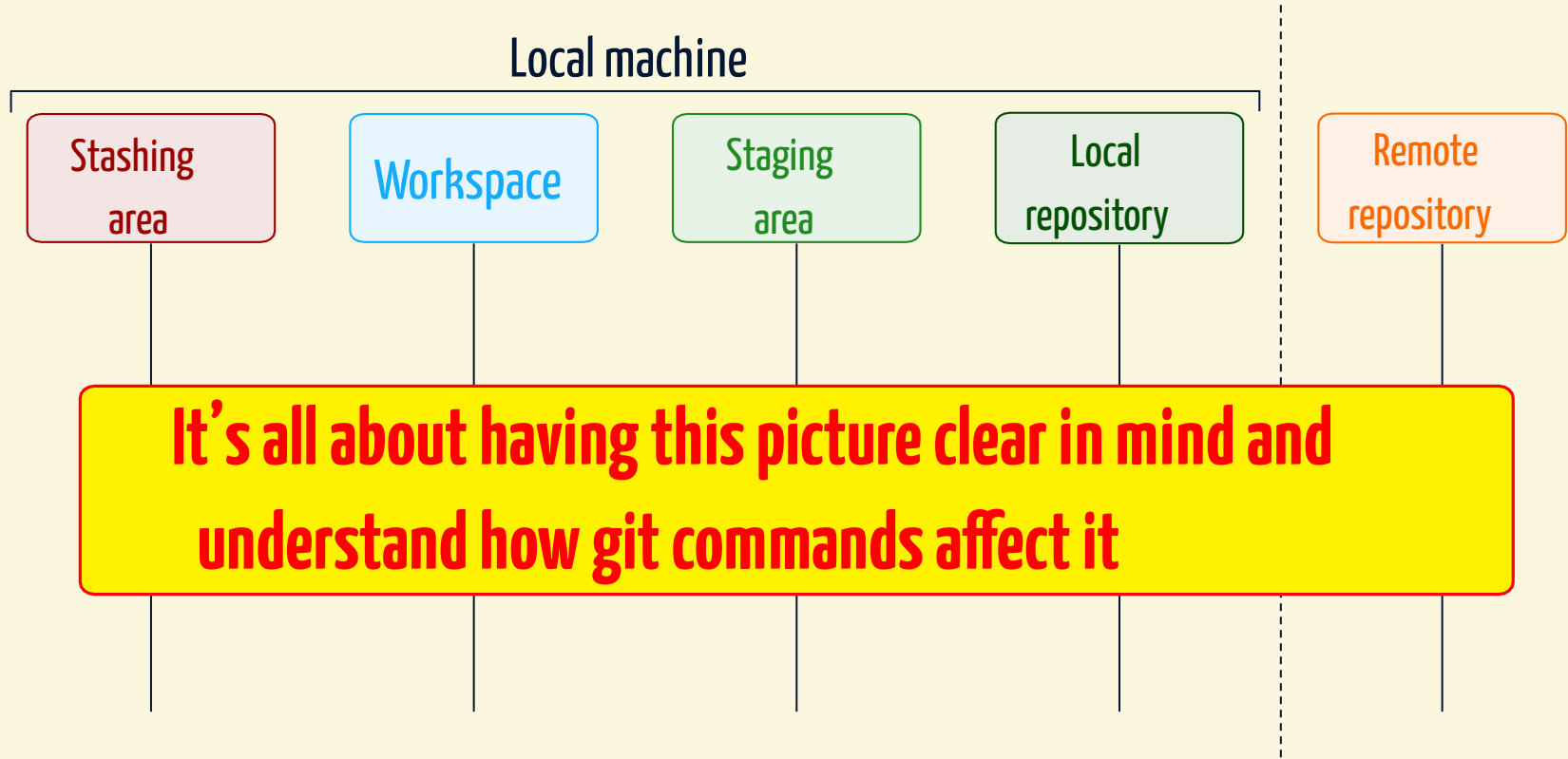


# The correct abstract mental setup

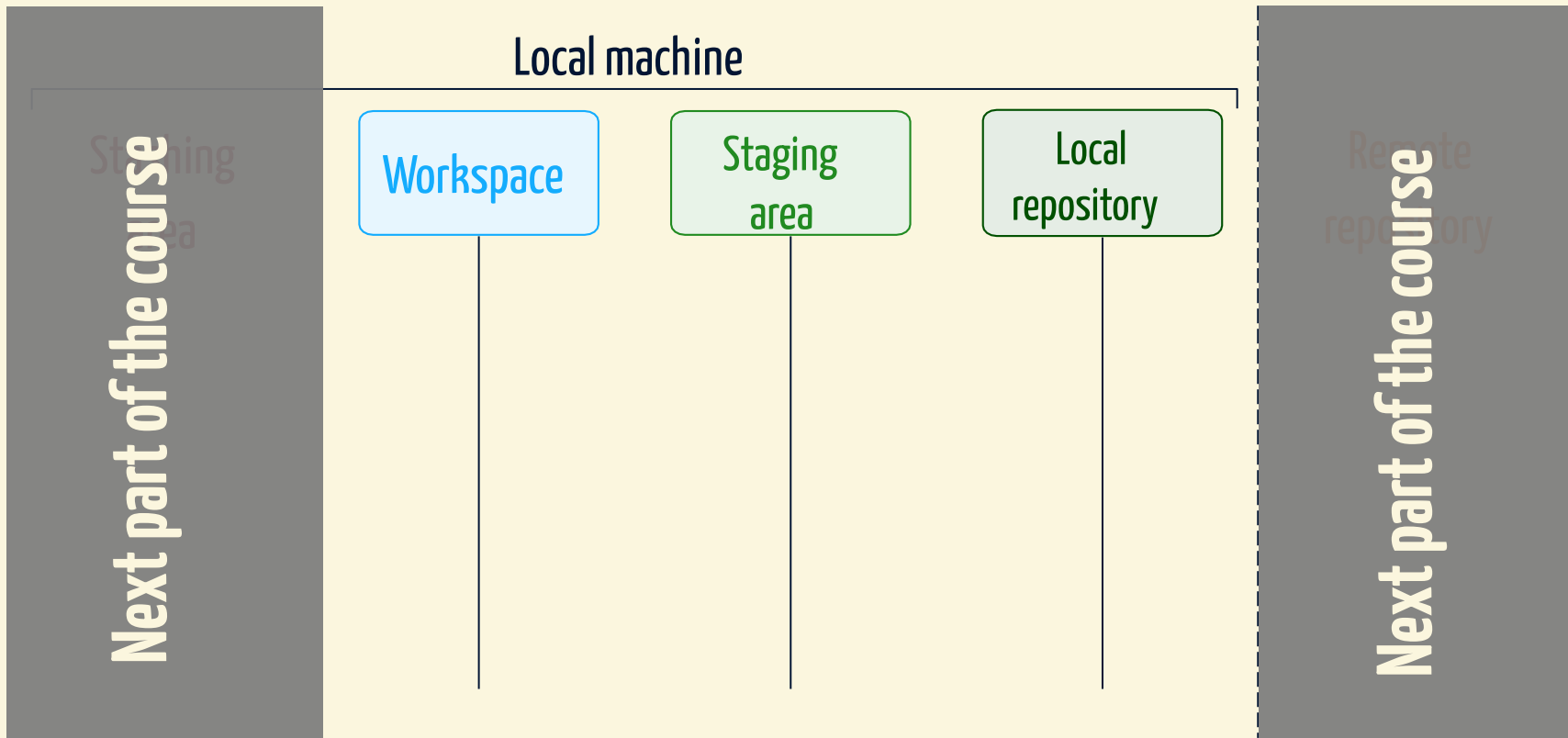
# The correct abstract mental setup



# The correct abstract mental setup



# The correct abstract mental setup



# How to use Git locally?

# Preliminary steps

Be sure to introduce yourself to Git on **each** machine from which you work

- 1 It is likely that Git is installed on your machine.
  - Check it in a terminal e.g. via `git version`
  - If needed, [install it](#)
- 2 Optionally, [get/enable autocompletion in the terminal](#)
- 3 Tell Git who you are and your email address

→ this information will be used to sign your work in history

```
$ git config -- global user.name 'NazihErrahel'
```

```
$ git config -- global user.email 'naziherrahel@gmail.com'
```

- 4 Set your favourite editor e.g. to write commit messages

```
$ git config -- global core.editor 'emacs -nw'
```

# Asking for help about Git

## 1 There are 3 ways in terminal

- `git help <command>`
- `git <command> --help`
- `man git-<command>`

e.g. `git help config`

e.g. `git config --help`

e.g. `man git-config`

## 2 List of commands on the [official reference](#)

## 3 Ask Google

There is plenty of cheat-sheets online:

 [GitHub education](#)

 [GitLab](#)

 [Bitbucket](#)

# Creating a repository

It is as simple as running one command

```
$ git config --get user.name naziherrahel
$ git config --get user.email
naziherrahel@gmail.com
# Suppose to be in a folder you want to turn into a repository
$ pwd
/home/nazih/Documents/first - repo
$ ls -a
. .. Paper.aux Paper.log Paper.out Paper.pdf Paper.tex
```



# Creating a repository

It is as simple as running one command

```
$ git config --get user.name naziherrahel
$ git config --get user.email
naziherrahel@gmail.com
# Suppose to be in a folder you want to turn into a repository
$ pwd
/home/nazih/Documents/first - repo
$ ls -a
. .. Paper.aux Paper.log Paper.out Paper.pdf Paper.tex
```

```
$ git init # <--- Here you go!
Initialised empty Git repository in ~/Documents/first - repo/.git/
$ ls -a
. .. .git Paper.aux Paper.log Paper.out Paper.pdf Paper.tex
```

# Creating a repository

It is as simple as running one command

```
$ git config --get user.name naziherrahel
$ git config --get user.email
naziherrahel@gmail.com
# Suppose to be in a folder you want to turn into a repository
$ pwd
/home/nazih/Documents/first - repo
$ ls -a
... Paper.aux Paper.log Paper.out Paper.pdf Paper.tex
```

```
$ git init    # <--- Here you go!
Initialised empty Git repository in ~/Documents/first - repo/.git/
$ ls -a
... .git Paper.aux Paper.log Paper.out Paper.pdf Paper.tex
```

Do not shoot yourself!

Never ever touch by hand the content of the .git folder.

# What comes next?

# What comes next?

On your local machine

Workspace

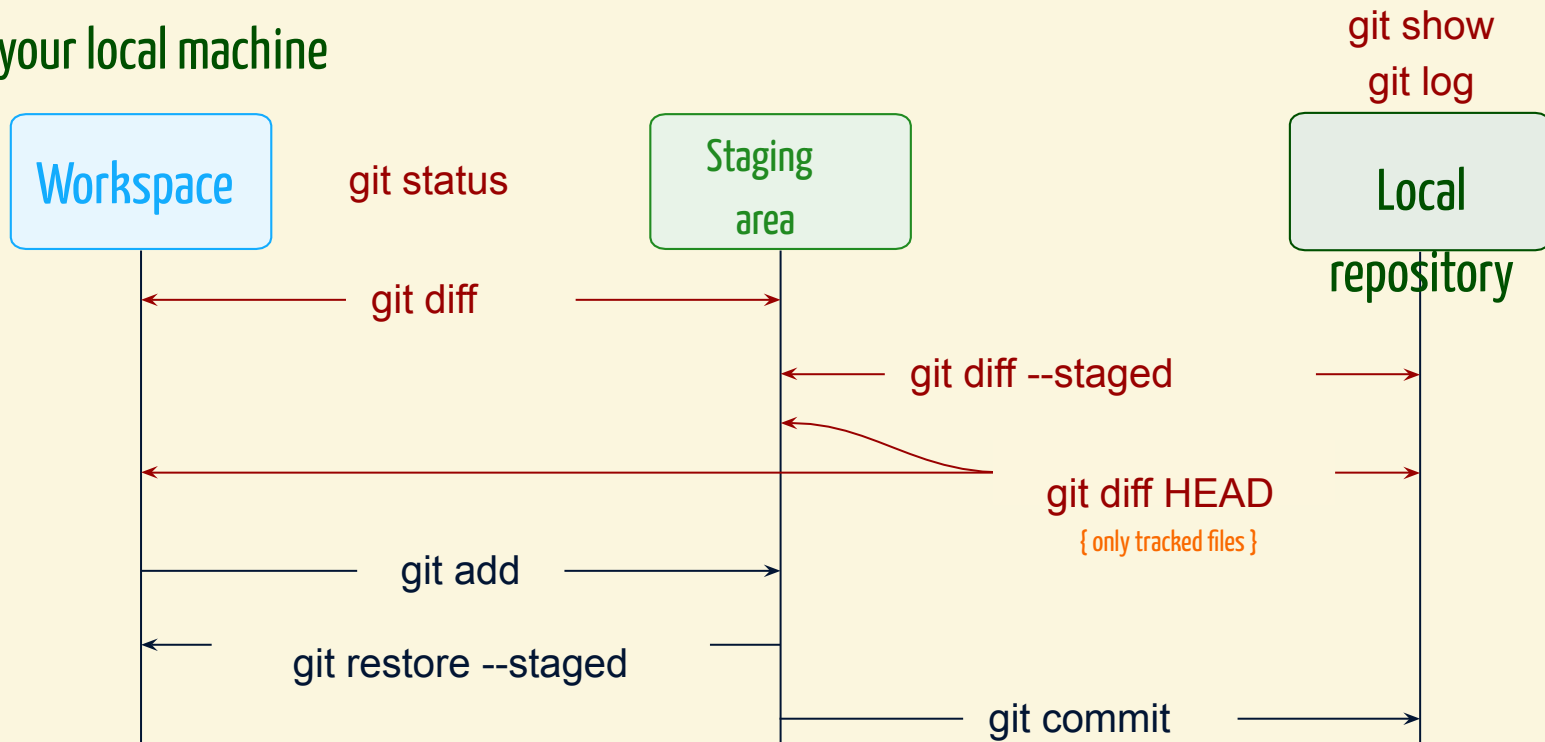
Staging  
area

Local  
repository

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

# What comes next?

On your local machine



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

# Git status

```
$ git status
```

```
On branch main
```

```
No commits yet
```

```
Untracked files:
```

```
( use " git add <file>..." to include in what will be committed)
```

```
Paper.aux
```

```
Paper.log
```

```
Paper.out
```

```
Paper.pdf
```

```
Paper.tex
```

```
nothing added to commit but untracked files present ( use " git add" to track)
```

**You do not want to put everything in a repository!**

**It is possible to tell git to ignore some files, like temporary ones**

# Letting Git ignore some files

```
$ printf '*.%.s\n' aux log out pdf > .aitignore
```

```
$ cat .gitignore
```

```
*.aux
```

```
*.log
```

```
*.out
```

```
*.pdf
```

```
$ git status
```

```
On branch main
```

```
No commits yet
```

```
Untracked files:
```

```
( use " git add <file>..." to include in what will be committed)
```

```
.gitignore Paper.tex
```

```
nothing added to commit but untracked files present ( use " git add" to track)
```



# Our first commit

## In your terminal

```
$ git log
fatal: your current branch 'main ' does not have any commits yet

$ git add .gitignore

$ git status
On branch main

No commits yet

Changes to be committed:
  ( use " git rm -- cached <file>..." to unstage)
    new file:   .gitignore

Untracked files:
  ( use " git add <file>..." to include in what will be committed)
    Paper.tex

$ git commit
```



# Our first commit

In your favourite editor

```
■  
# Please enter the commit message for your changes. Lines starting  
# with ' #' will be ignored , and an empty message aborts the commit.  
  
# On branch main  
#  
# Initial commit  
  
# Changes to be committed:  
#   new file:   .gitignore  
# Untracked files:  
#   Paper.tex
```

# Our first commit

## In your favourite editor

Add .gitignore file for TeX project



# Please enter the commit message for your changes. Lines starting  
# with ' #' will be ignored , and an empty message aborts the commit.

# On branch main

#

# Initial commit

# Changes to be committed:

#   new file:   .gitignore

# Untracked files:

#   Paper.tex

# Our first commit

## In your terminal

```
$ git log
fatal: your current branch 'main ' does not have any commits yet

$ git add .gitignore
$ git status
On branch main

No commits yet

Changes to be committed:
  ( use " git rm -- cached <file>..." to unstage)
    new file:   .gitignore

Untracked files:
  ( use " git add <file>..." to include in what will be committed)
    Paper.tex

$ git commit
# Your editor opens -> type commit message , save and exit
[main (root - commit) bb8c78b] Add .gitignore file for TeX project
 1 file changed , 4 insertions(+)
 create mode 100644 .gitignore
```

# Inspecting history

```
$ git log
```

```
commit bb8c78b68075dacf8467420bc00867c73ef5ba8c (HEAD -> main)
```

```
Author: nazierrahel <nazierrahel@gmail.com>
```

```
Date: Thu Sept 23 10:13:05 2024 +0100
```

```
Add .gitignore file for TeX project
```

```
$ git log -- oneline
```

```
bb8c78b (HEAD -> main) Add .gitignore file for LaTeX project
```

Use a pager to avoid polluting terminal

```
$ git config -- global core.pager 'less -+ $LESS -R'
```

Use **git show** or **git show <SHA1>** to inspect what has been done in last or given commit

# Our second commit

```
$ git status On branch main
```

```
Untracked files:
```

```
( use " git add <file>..." to include in what will be committed)
```

```
Paper.tex
```

```
nothing added to commit but untracked files present
```

```
( use " git add" to track)
```

```
$ git add Paper.tex # Always add to the staging  
# area before committing!
```

```
$ git status
```

```
On branch main
```

```
Changes to be committed:
```

```
( use " git restore -- staged <file>..." to unstage)
```

```
new file: Paper.tex
```

```
$ git commit -m 'Add paper main document'
```

```
[main 9c6154d] Add paper main document
```

```
1 file changed , 147 insertions(+)
```

```
create mode 100644 Paper.tex
```

# Use good commit messages

```
$ git log -- oneline
```

```
9c6154d ( HEAD -> main) Add paper main document
```

```
bb8c78b Add .gitignore file for LaTeX project
```

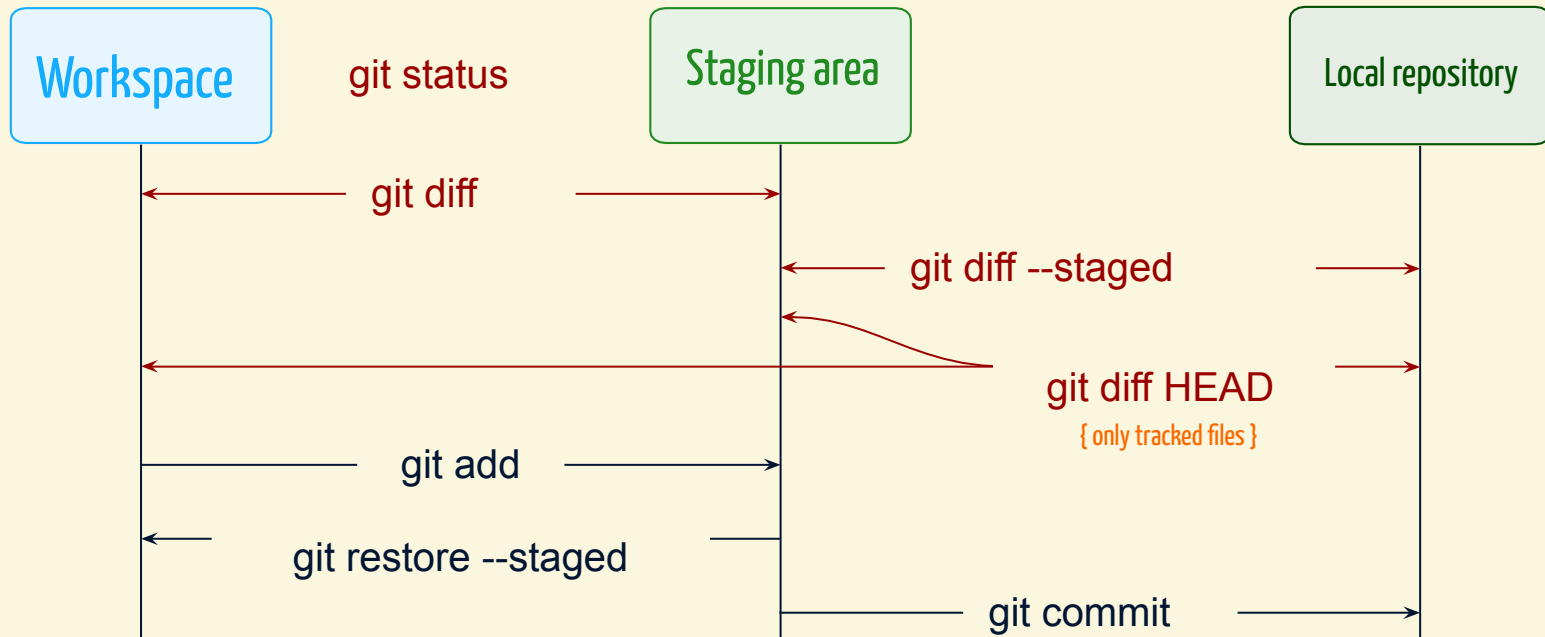
- Write them like an email to yourself (or to the other developers)
  - Subject line + body, follow [the 50/72 rule](#)
- Subject: Summarize what has been done
  - **Use present tense and no period at the end!**
- Body: **After empty line**, document why you made the changes
  - \ { add one only if needed }

## Good commits

Commit small and conceptually separated changes, commit often and do not add binary files to your repository.

# Back to our mental picture

On your local machine



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

# Working and displaying changes

## In your terminal

# Make some changes

\$ git status

On  
branch main

Changes not staged for commit:

( use " git add <file>..." to update what will be committed)

( use " git restore <file>..." to discard changes in working directory)

modified: Paper.tex

no changes added to commit ( use " git add" and/or " git commit -a")

\$ git diff



# Working and displaying changes

In your pager, e.g. less

```
diff --git a/Paper.tex b/Paper.tex
index 3c408e5..3669114 100644
--- a/Paper.tex
+++ b/Paper.tex
@@ -42,7 +42,7 @@ pdftitle={LaTeX Seminar for PhD students}
    {Sprecher \& Seminarleiter}%
}

- \date{\May 16, 2024}
+ \date{23. Dezember 2021}

\newcommand{\ etc}{etc.}
\newcommand{\ zB}{z.B.}
```

# Working and displaying changes

## In your terminal

# Make some changes

\$ git status On branch

main

Changes not staged for commit:

( use " git add <file>..." to update what will be committed)

( use " git restore <file>..." to discard changes in working directory) modified: Paper.tex

no changes added to commit ( use " git add" and/or " git commit -a")

\$ git diff

\$ git diff -- staged # Nothing in the staging area!

\$ git add Paper.tex

\$ git diff # No changes anymore in the workspace!

\$ git diff -- staged # Our changes are now staged

23\$ git commit -m 'Fix date for main document' # ...

# What else can I easily explore?

- Stage all tracked modified files at once

```
git add -u
```

- Stage partial modification in a file

```
git add -p
```

- Define your aliases

```
git config -- global alias.unstage 'restore -- staged --'
```

```
# From now on, you can use 'git unstage '
```

- Let git correct you when you mistype<sup>⋄</sup>

```
git config -- global help.autocorrect 1
```

- Change/correct your last commit message

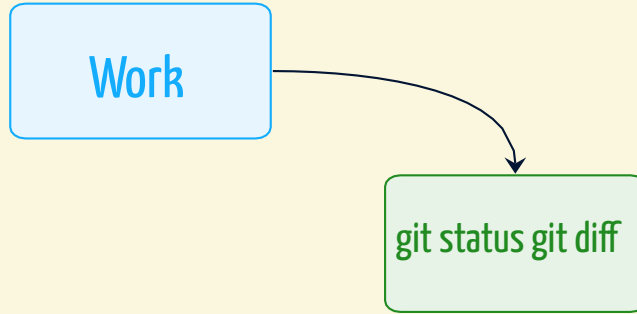
```
git commit --amend
```

# Summary and conclusions

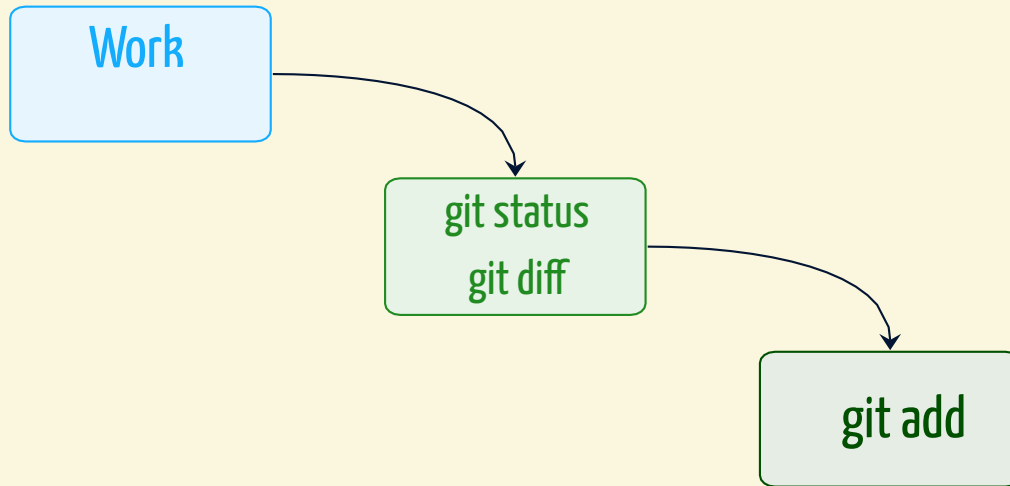
# Your workflow from now on, right?

Work

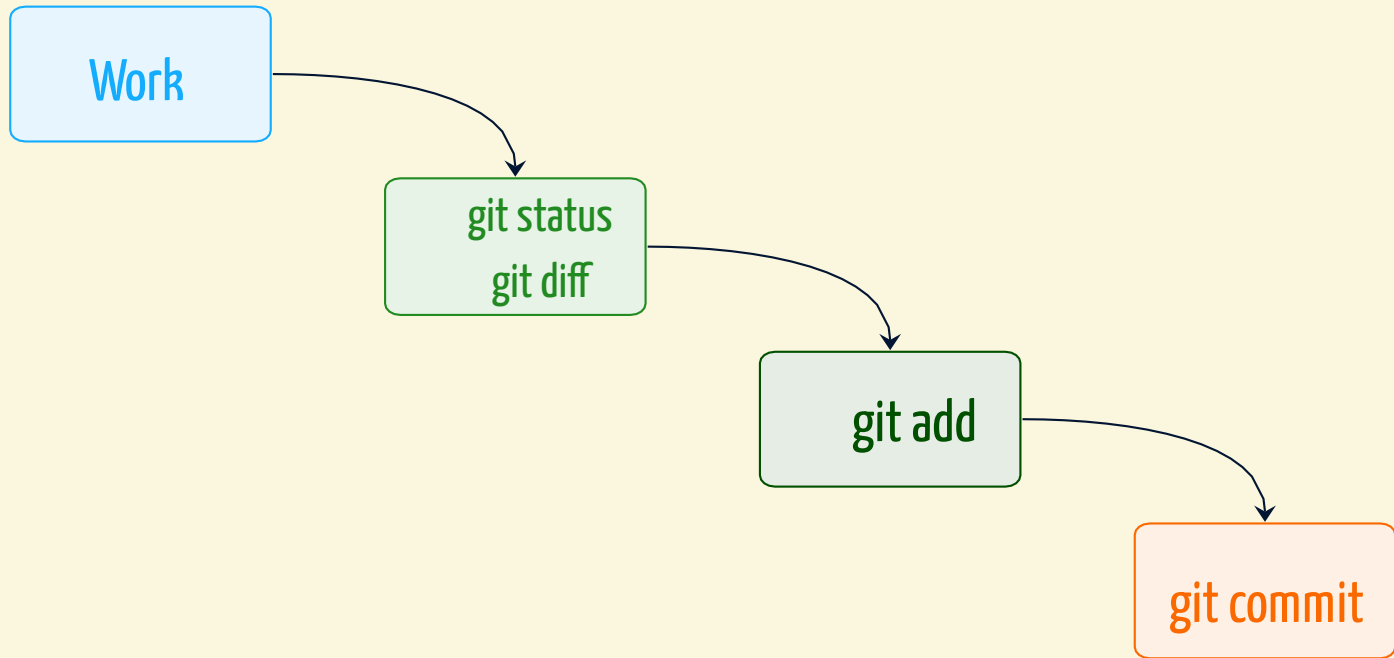
# Your workflow from now on, right?



# Your workflow from now on, right?

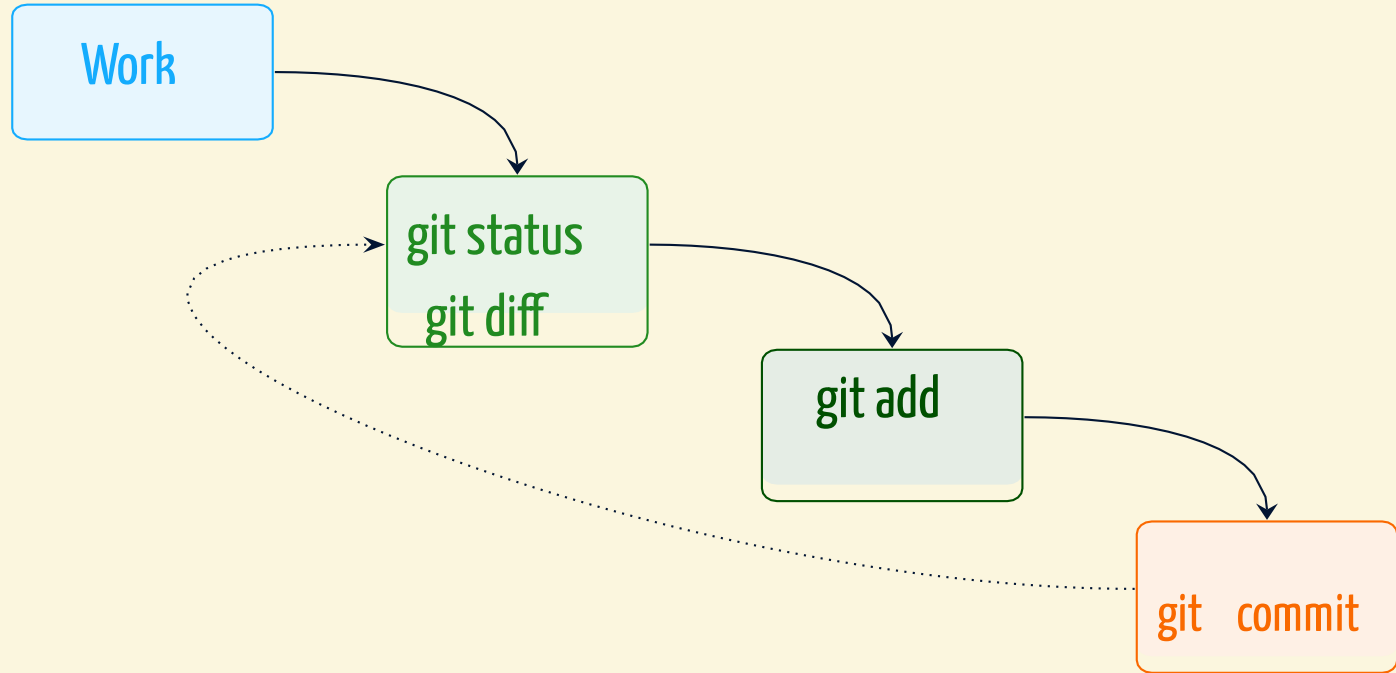


# Your workflow from now on, right?

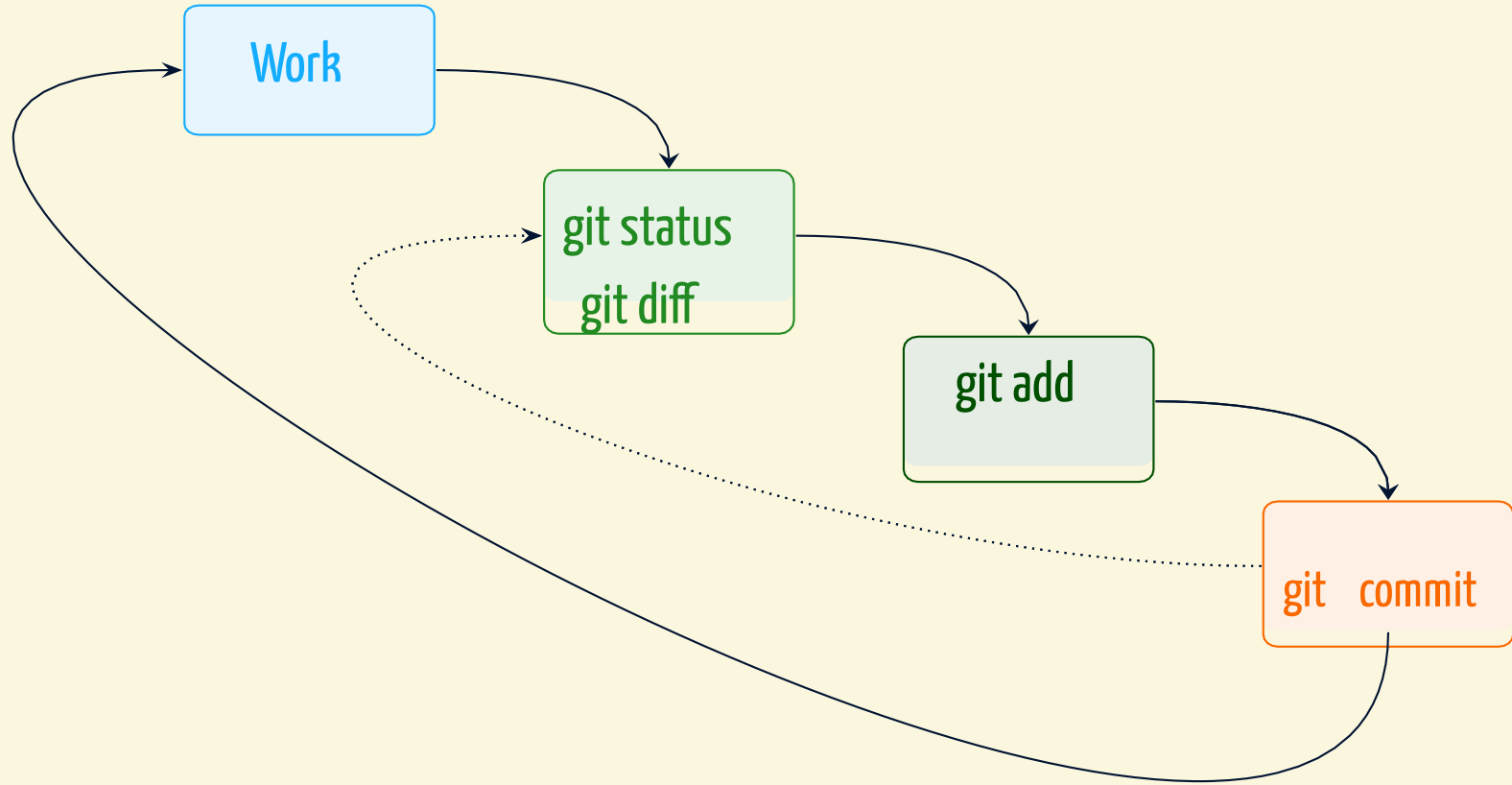


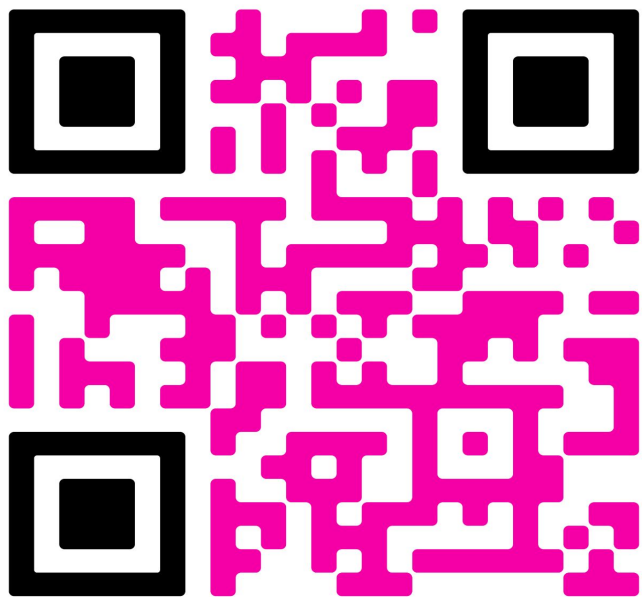


# Your workflow from now on, right?



# Your workflow from now on, right?





**Quiz Time!!**



**SCAN ME**