Git

Agenda

1. Introduction - *Version Control System*
2. Prerequisites
3. Git
   - History
   - Architecture
   - Data flow
   - The most important commands
4. Exercises

Version Control System

Software supporting the daily work of programmers through (facilitating):

- cooperation in the implementation of IT projects,
- preserving the history of changes,
- protection against loss of work effects,
- testing changes in existing software,
- version management, taking into account "tailor-made" solutions for a specific client.

Prerequisites

Git installation

You can download it from: https://git-scm.com/downloads

On Linux (RPM-based)

```
$ sudo dnf install git-all
```

On Linux (Debian-based)

```
$ sudo apt-get install git-all
```

On MacOS

```
$ brew install git
```

Git

Distributed version control system

*codeburst.io

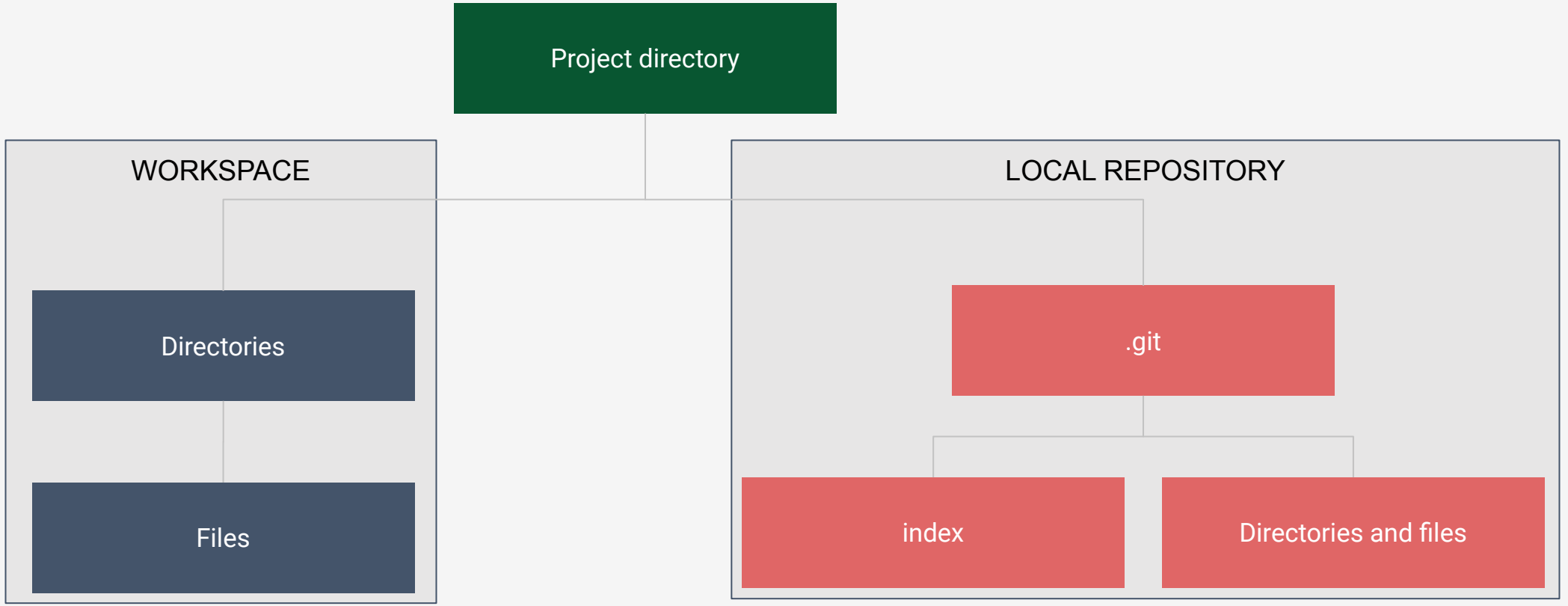
A minute of history

- Git was created by the creator of the Linux operating system - Linus Torvalds.
- Initially, it was used to manage the versions of the source code of the Linux kernel.
- The initial version was created in 3 days.
- The assumptions for his creation were the following:
  - Incorporate the Concurrent Versions System (CVS) as an example of what not to do. If in doubt, make the exact opposite decision.
  - Supports a BitKeeper-like distributed workflow.
  - Consideration of very strong safeguards against data corruption, both by accidental and deliberate action.

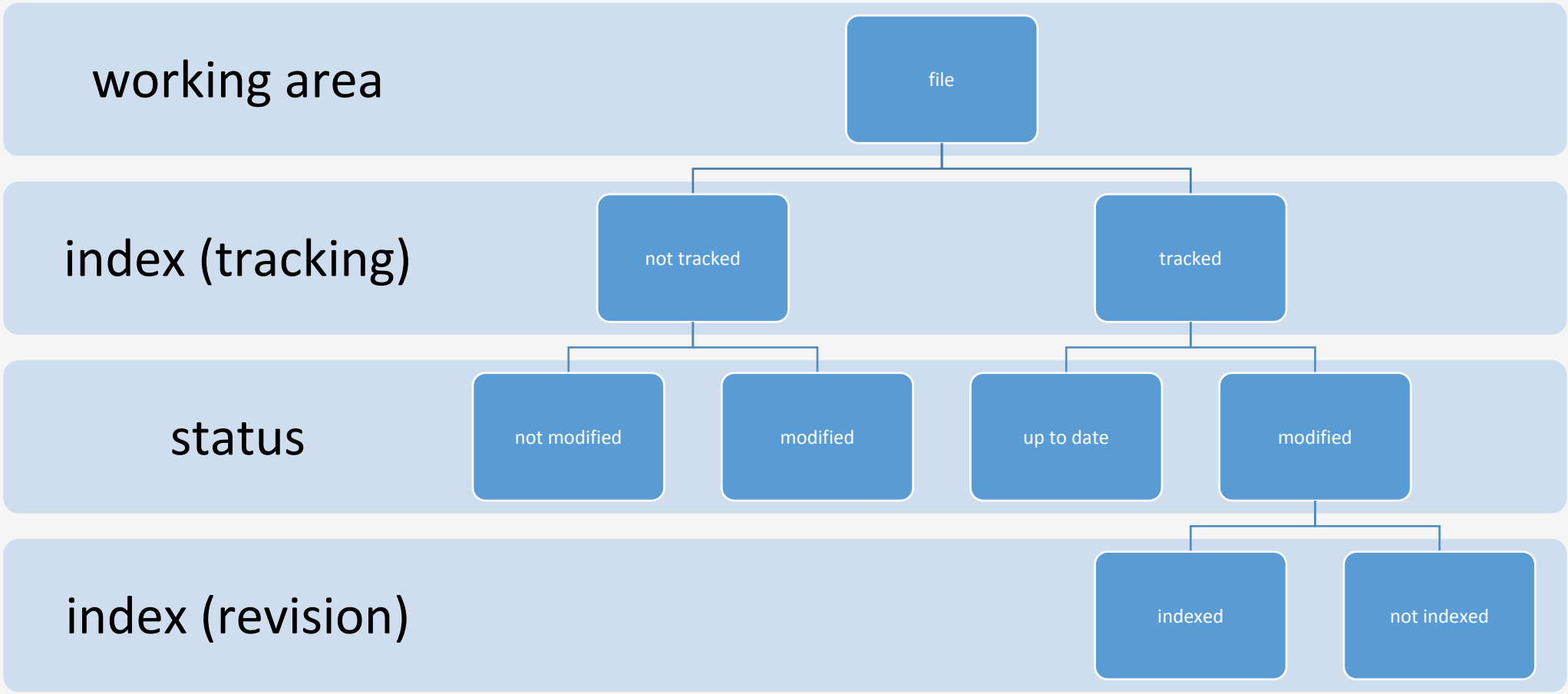
What has been achieved.

Architecture

Git is a **distributed** system that uses **graph theory**. It does not require access to the remote repository to manage **revisions of the code.**



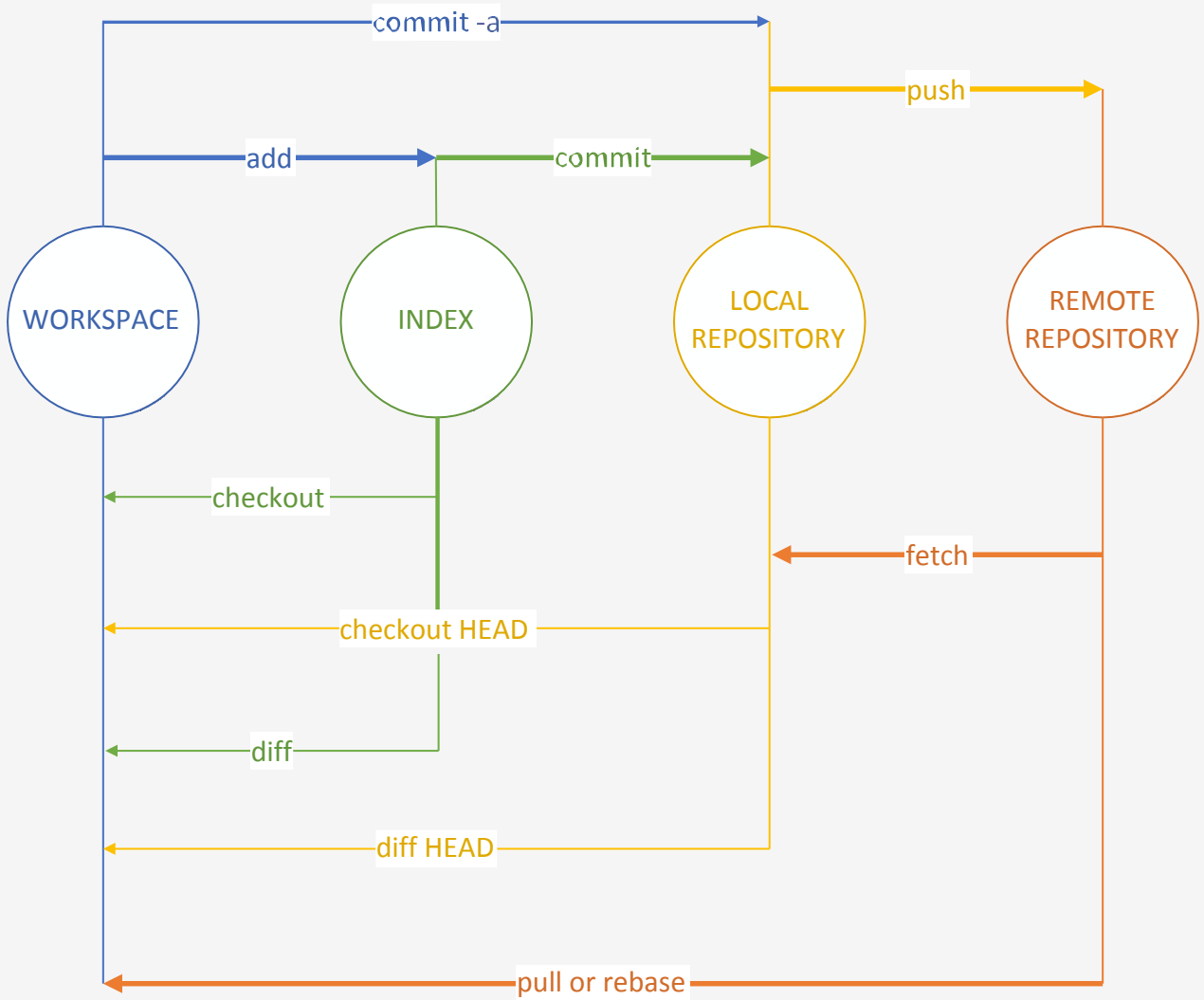
File status



* Git - distributed version control system

Git workflow

- The programmer adds (indexes) the file: **git add <file_name>**
- Places it in the local repository: **git commit -m "message regarding the file"**
- Gets the current content of remote repository: **git pull**
- Sends the code to the remote repository: **git push**



General

The terms "send", "download" for files in the case of the Git system do not reflect the operations actually performed.

Git uses **incremental procedures,** thanks to which gets very high **performance.**

Configuration

git init

Initializes local repository in current directory.

The operation is performed **only once** - when we want to create new project.

- git init

git config

Environment configuration, including username and email address setting, that enables the authorization on the remote repository.

The operation can be performed repeatedly, depending on the need.

- git config --global user.name „<full name>”
- git config --global user.email „<email address>”

Main commands

git add

Adds a file (or files) to the index - it causes their "tracking" by the git

The operation is performed after each modification of the file that we intend to manage (or we have managed already) using the repository.

- git add <file name> **or** git add <name 1> <name 2> … **or** git add . (add all)

git commit

Send files that have been modified (and are in the index file) to the local repository.

The operation is performed after each modification of indexed / tracked files.

- git commit (will open the file inside which you will be able to enter information on the introduced changes)
- git commit –m „message about the changes”

Main commands c.d

git push

Sends modified files from the local repository (updates files) to the remote repository.

The operation is performed every time when we want to update the state of our remote repository with local commits (changes).

- git push

git pull

It downloads files (updates) from the remote repository and tries to automatically update their counterparts in the local (workspace) repository.

The operation should be performed before each "push" operation.

- git pull

Main commands c.d

git checkout

Change a branch or update local files according to the version included in index file.

Used at any time when there is a need to test the data functionality or refresh the workspace content.

- git checkout –b <branch_name>
- git checkout

git merge

The combination of different solutions (branches) into one source code.

Used during the code integration

- git merge <branch_name>/master

Helpful Commands

git status

Displays information about the current state of the software (indexed / unindexed files)

Used at any time. It allows you to verify the current state of the repository.

- git status

gitk

Displays a graphical representation of the current state of the repository.

Used at any time. Runs the window application.

- gitk& **or** gitk.exe (PowerShell)

Practice

Exercises - individual work

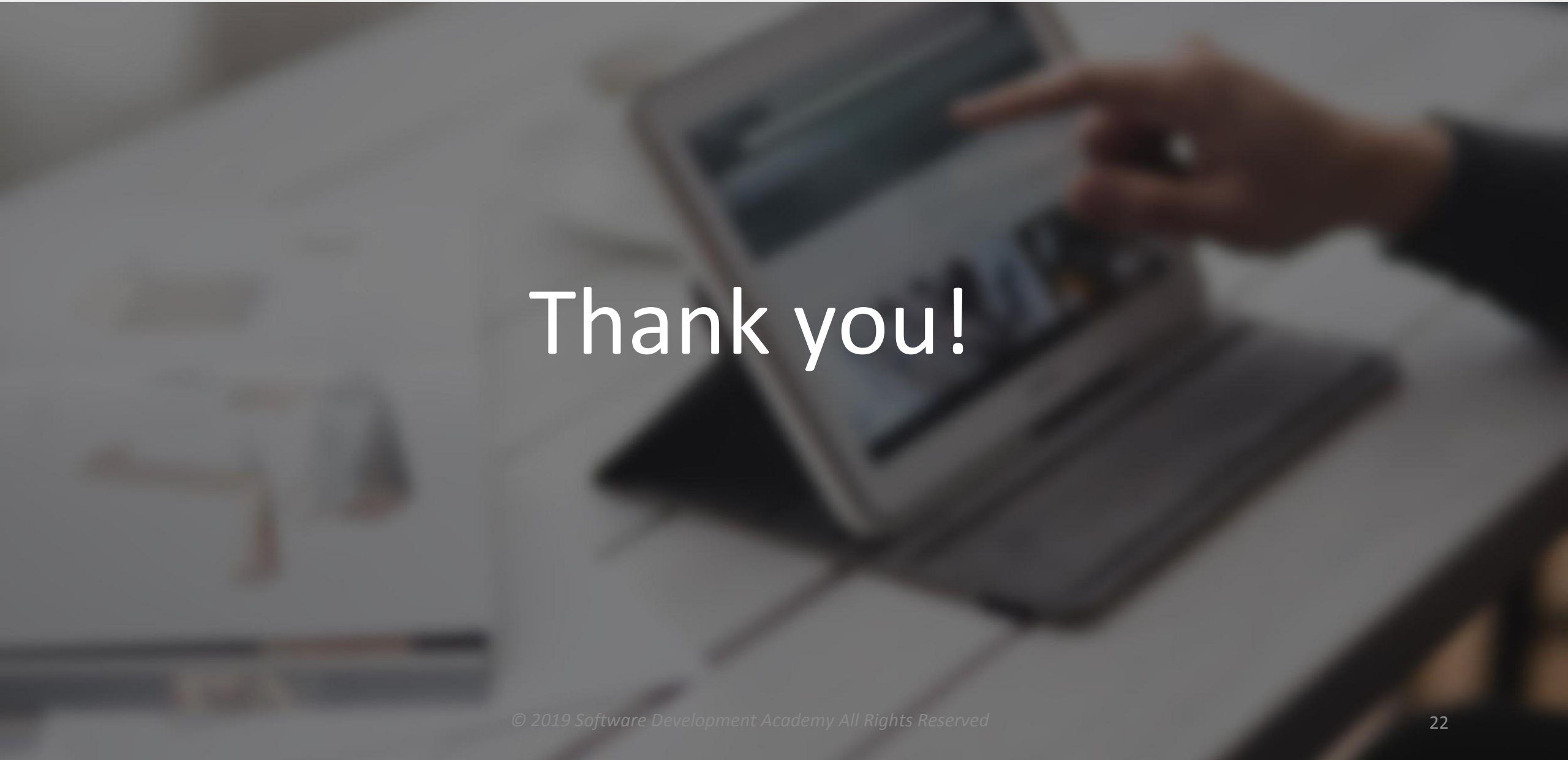
1. Create a local and remote repository.
2. Configure the local environment (user, email address)
3. Add files to the index (git add)
4. Add the file to the local repository with the "init" message (git commit -m "init")
5. Send files to the remote repository
6. Run gitk from the command line
7. Verify the result via gitk and GitLab
8. Modify the created code snippet (add a comment, modify the selected line). Repeat steps 5 through 8.
9. Download the repository to a different place on the disk (git clone ...)

Exercises - individual work

1. Create a branch with any name, e.g. "test_branch" (git checkout -b "test_branch")
2. Modify any file, send its contents to the remote repository.
3. Go to the place where the repository was cloned.
4. Modify the same file, but in a different way, send the content to the remote repository.
5. Merge the repository from the "test_branch" level.
6. Present the effect.

Exercises - work in groups

1. Work in a groups of 3-4.
2. Choose any small project (2-3 classes + class containing the main method).
3. Divide the tasks between yourself.
4. Create a team on the GitLab site.
5. Complete the project via a shared repository.



Thank you!