eurac research

Git – Fast Version Control

INT-GIT-22

Daniel Summerer

27.04.2022

Objectives

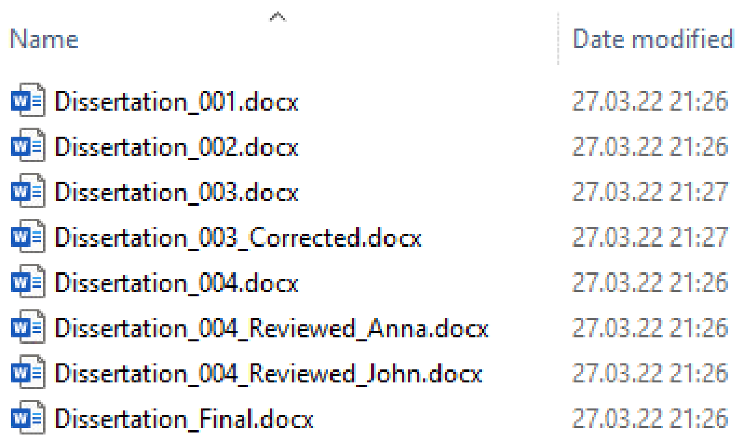
- Understand the basics of Git
- Get familiar with the Git command line
- Learn about branching and merging
- Resolve merge conflicts
- Set up projects on GitHub or GitLab
- Collaborate on and maintain projects using GitHub or GitLab

Agenda

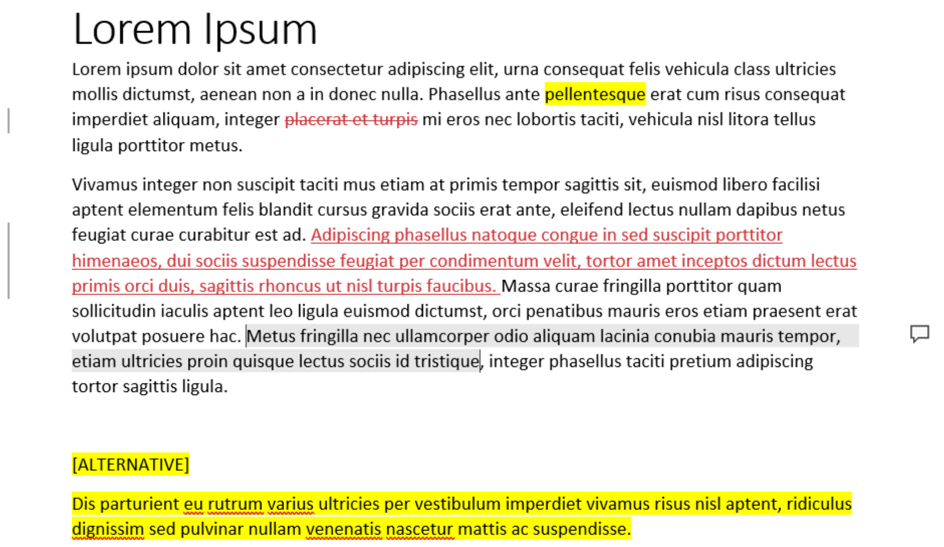
09:00 – 09:45	Introduction to Git
09:45 – 10:00	Break
10:00 – 11:00	Working with Git
11:00 – 12:30	Remote Repositories and GitLab
12:30 – 13:30	Lunch Break
13:30 – 15:00	Branching and Merging
15:00 – 15:15	Break
15:15 – 16:00	Hands-On Exercises
16:00 – 17:00	Advanced Features + Q&A

Version Control

Basic Version Control



Name	Date modified
Dissertation_001.docx	27.03.22 21:26
Dissertation_002.docx	27.03.22 21:26
Dissertation_003.docx	27.03.22 21:27
Dissertation_003_Corrected.docx	27.03.22 21:27
Dissertation_004.docx	27.03.22 21:26
Dissertation_004_Reviewed_Anna.docx	27.03.22 21:26
Dissertation_004_Reviewed_John.docx	27.03.22 21:26
Dissertation_Final.docx	27.03.22 21:26



Lorem Ipsum

Lorem ipsum dolor sit amet consectetur adipiscing elit, urna consequat felis vehicula class ultricies mollis dictumst, aenean non a in donec nulla. Phasellus ante pellentesque erat cum risus consequat imperdiet aliquam, integer ~~placerat et turpis~~ mi eros nec lobortis taciti, vehicula nisl litora tellus ligula porttitor metus.

Vivamus integer non suscipit taciti mus etiam at primis tempor sagittis sit, euismod libero facilisi aptent elementum felis blandit cursus gravida sociis erat ante, eleifend lectus nullam dapibus netus feugiat curae curabitur est ad. Adipiscing phasellus natoque congue in sed suscipit porttitor himenaeos, dui sociis suspendisse feugiat per condimentum velit, tortor amet inceptos dictum lectus primis orci duis, sagittis rhoncus ut nisl turpis faucibus. Massa curae fringilla porttitor quam sollicitudin iaculis aptent leo ligula euismod dictumst, orci penatibus mauris eros etiam praesent erat volutpat posuere hac. Metus fringilla nec ullamcorper odio aliquam lacinia conubia mauris tempor, etiam ultricies proin quisque lectus sociis id tristique, integer phasellus taciti pretium adipiscing tortor sagittis ligula.

[ALTERNATIVE]

Dis parturient eu rutrum varius ultricies per vestibulum imperdiet vivamus risus nisl aptent, ridiculus dignissim sed pulvinar nullam venenatis nascetur mattis ac suspendisse.

Basic Version Control

These approaches have some flaws:

- Changes are not visible
- Tracking changes is not possible
- Different versions cannot be compared
- Restoring previous versions is not possible
- Collaboration is difficult

Definition

Version control is a set of systems responsible for managing changes to computer programs, documents and files in general.

Version control is a part of software configuration management (SCM).

Benefits

- Backup
- View change history of a file
- Work in parallel on the same files, avoiding conflicts
- Review and merge changes made by different people
- View and restore previous versions
- Distributed work

Use Cases

- Papers, dissertations, publications
- Data analysis, statistics
- Software development
- Task automation
- Issue tracking
- Basic project management

Version Control vs. Versioning

Version Control

- Track changes to files
- Enables collaboration and avoids conflicts
- Software development

Versioning

- Track stages of a software program or project
- Identify different versions of a program
- Software release and distribution

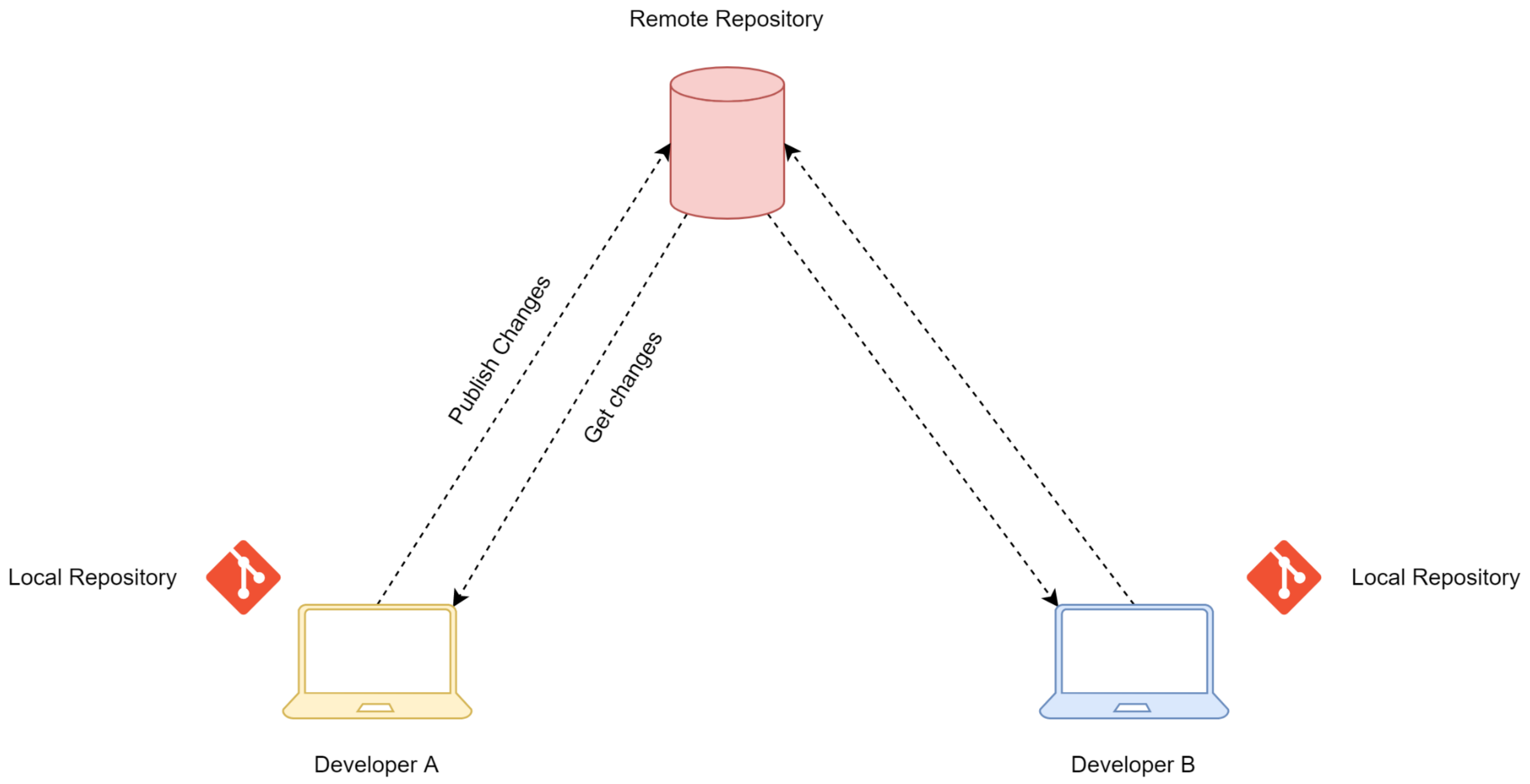
Git Overview

What is Git

Git is a version control system

- Free and open source
- Distributed
- Multi-platform

Basic Architecture



Basic Architecture

Repository

Basic element, used to manage projects with Git

Local Repository

Folder on the local PC, used to work on the files

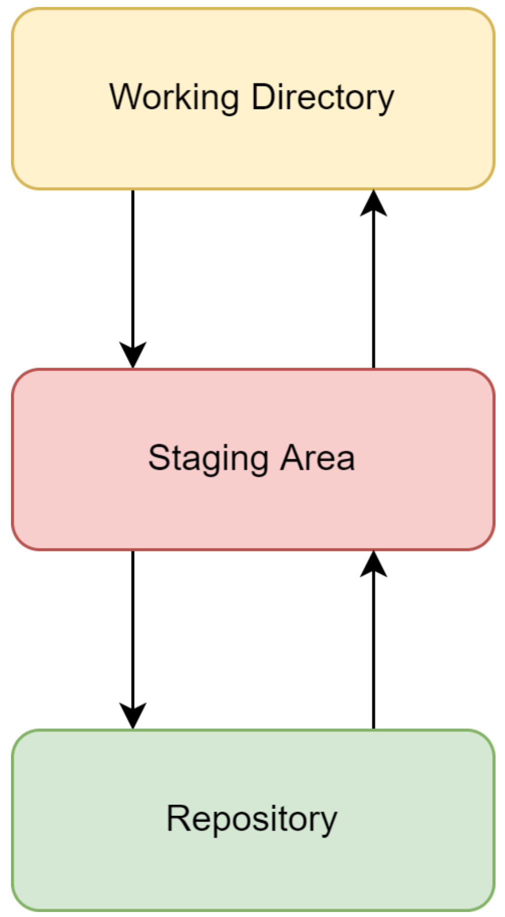
Remote Repository

Repository on a file server, shared with all contributors

Basic Architecture

A local repository consists of 3 main components

- Working Directory
- Staging Area
- Repository



Working Directory

Folder on the local computer.

- Specific structure
- Recognized by Git as part of a repository

Files managed in the working directory are read and tracked by Git

File Tracking

Files in a local Git repository folder can be in 2 states:

- Tracked
  Changes to these files are tracked by Git
- Untracked
  Files are present in the repository, but changes to these files are not tracked by Git

New files are always untracked. They need to be added to the staging area, to track them.

Staging Area

Intermediate step before a commit.

Use cases:

- Validation
- Review
- Correction
- Partial Commits

Commit

Snapshot of the changes staged in the staging area.

- Unique number (called commit hash)
- Message
- Saved to commit history
- Repository can be viewed at and restored to a commit

Git does not store a copy of the file, but only the changes applied to the file.

Supported file types

Git supports any type of file.

However, you get the best experience with text files, because they allow for change tracking.

Examples of text files: plaintext, HTML, LaTex, source code.

Competitors

Mercurial

- Distributed model
- Fewer commands

Subversion

- Centralized model
- Low adoption rate

Let’s get started!

Installation

Download Git from the following website and install it on your machine
https://git-scm.com/downloads

Git CLI

Git is a command-line tool.

You can use any CLI you like. For this workshop, we are going to use:

- Windows: Git Bash
- Mac OS: Terminal
- Linux: Terminal

Command Line Interface

Basic Commands

ls

List items in the current folder

Options

- -a: Show all elements
- -l: Show elements with details
- -t: Order elements by modified time

Example

```
$ ls
$ ls -al
```

Basic Commands

cd

Change directory to the specified path

Path shortcuts:

- . → Current folder
- .. → Upper folder
- ~ → Home folder
- / → Root folder

Example

```
$ cd Documents
$ cd ..
```

Working with Git

Git Version

First, let’s check if Git is properly installed

```
$ git --version
> git version 2.35.1.windows.2
```

Basic Configuration

After the installation, it is necessary to configure the following information:

- Name
- Email address

This information is added to every commit to identify the user who created the commit.

```
$ git config --global user.name "Summerer Daniel"
$ git config --global user.email "daniel.summerer@eurac.edu"
```

Create your first repository

Now, we are ready to create our first repository.

- Create a folder on your desktop and call it *MyRepository*.
- Open the CLI and navigate to this folder

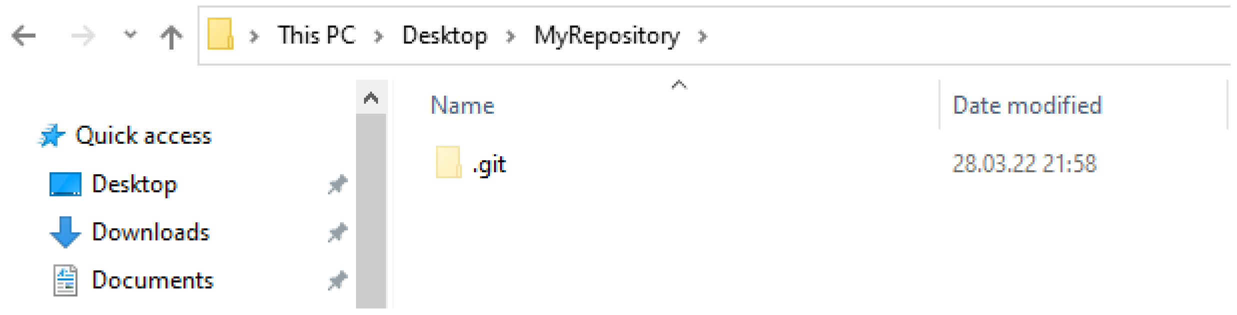
Run the following command to initialize the *MyRepository* folder as a Git repository

```
$ git init
> Initialized empty Git repository in C:/Users/dasummerer/Desktop/MyRepository/.git/
```

Repository Structure

The *MyRepository* folder contains a hidden folder called .git.

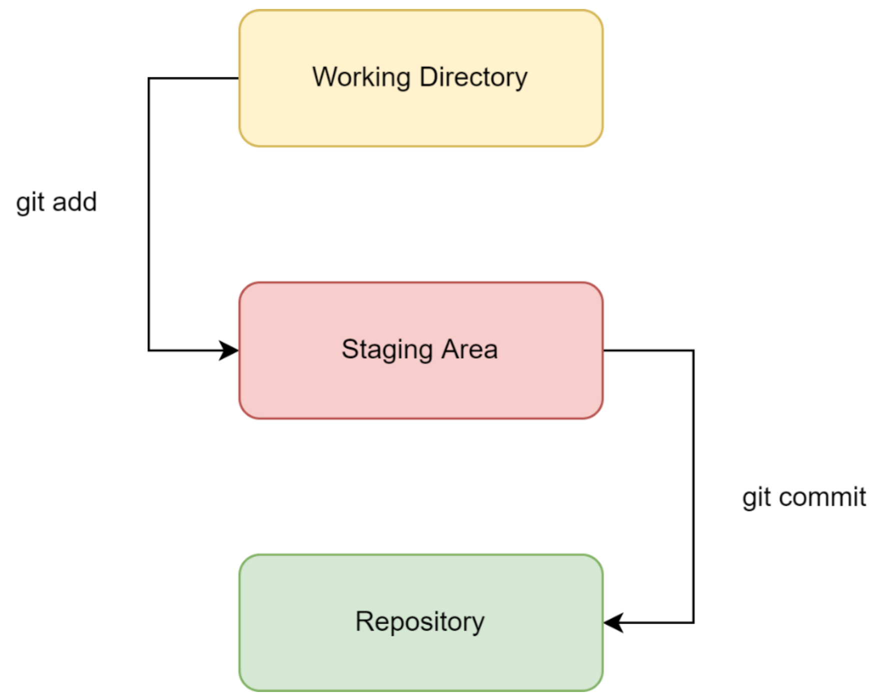
This folder is used by Git to store information about your repository, such as the commit history, file changes, etc.



.git 28.03.22 21:58

Basic Workflow

1. Create a repository, by initializing it locally or cloning it from a Git server.
2. Every time you add, edit or delete files in your repository, Git considers them modified.
3. Select the modified files to add to the staging area.
4. Commit your staged files, thereby creating a permanent snapshot of the files.



Basic Workflow

git status

Informs you about the status of the repository:

- Untracked files
- Changed files
- Change types
- Staged files
- Unpublished commits

```
$ git status
```

Basic Workflow

git add

Adds a file or change to the staging area.

```
$ git add <file>
```

This also enables tracking for untracked files.

Example

```
$ git add File01.txt
```

Basic Workflow
git add

To add all files in the current directory, use the --all option

```
$ git add --all
```

Or specify the current directory as a path

```
$ git add .
```

Basic Workflow

git commit

Creates a commit for all staged files

```
$ git commit
```

This opens a text editor, where you need to specify the commit message.

Add the -m option to specify the commit message directly in the commit statement

```
$ git commit -m "My message"
```

Basic Workflow

git log

Shows the commit history, including

- Commit hash
- Message
- Author
- Date

```
$ git log
```

Hands-On

1. Create a text file in the MyRepository folder
2. Stage it and commit it
3. Apply changes to the file
4. Use git status to see what has changed
5. Stage and commit your changes
6. Use git log to see the commit history

Analyzing Differences

Differences

Git stores the changes associated to every commit. You can use this information to see what has changed between two commits.

A change is also called a difference.

Command

git diff

Difference between working directory and last commit

```
$ git diff
```

Difference between working directory and a specific commit

```
$ git diff <hash>
```

Difference between two commits

```
$ git diff <hash-A> <hash-B>
```

Analyzing the Output

Git diff produces an output that shows:
- Lines that were added
- Lines that were deleted

Modified lines are shown as a combination of insertion and deletion.

The output is truncated only to the lines that changed.

Hands-On

1. Apply changes to the text file in your repository
2. Analyze changes in the working directory
3. Analyze changes between two commits

Revert Commits

Motivation

After analyzing the differences, you may decide to undo your changes. This is done by reverting a commit.

To revert a commit, Git applies your changes “backwards” and creates a new commit.

Revert

git revert

Revert the changes of last commit

```
$ git revert HEAD
```

Revert the changes of a specific commit

```
$ git revert <commit-hash>
```

Revert changes of N-th last commit (numbers start at 0)

```
$ git revert HEAD~<N>
```

Hands-On

1. Change the file, and commit your changes
2. Revert the changes applied in your last commit

Remote Repositories

Recap

So far, we have been working on a local repository.

To enable collaboration, we need a remote repository, accessible by all project contributors.

- The remote repository will keep the common code base.
- Every contributor works on their local repository.
- Local changes are pushed to the remote repository.

Distributed Model

Git is a distributed version control system

- Single remote repository
- Multiple local repositories
- Every local repository is a copy of the remote repository

Advantages with respect to centralized model:

- Independent development
- Flexible workflows
- No single point of failure

Remote Repository

A remote repository can be created on any server that runs git.

It is recommended to use DevOps platforms.

GitHub

GitLab

ATLASSIAN
Bitbucket

Remote

A remote is a connection to a remote repository.

Your local repository needs a remote to know where to publish your local changes.

Specifying Remotes

Cloning (Recommended)

- Remote repository created before the local repository
- Remote is set by cloning the repository

Manual Setting

- Local repository created before remote one
- Remote is set by adding a reference through a command

Work with Remote Repository

git clone

Initializes a local repository as a clone of the remote repository.

Depending on the authentication type enabled/configured on the remote server, you can clone it via HTTPS or via SSH.

Via SSH (Recommended)

```
$ git clone git@gitlab.inf.unibz.it:daniel.summerer/my-repository.git
```

Via HTTPS

```
$ git clone https://gitlab.inf.unibz.it/daniel.summerer/my-repository.git
```

Work with Remote Repository

git push

Pushes your local commits to the remote repository

```
$ git push origin <remote-branch>
```

You can set the local branch to automatically track a remote branch, by executing

```
$ git push --set-upstream origin <branch>
```

The next time, you can push by simply writing

```
$ git push
```

Work with Remote Repository

git pull

Retrieves the latest changes from the remote repository

```
$ git pull
```

GitLab

Introduction to GitLab

Local GitLab server:
https://gitlab.inf.unibz.it/

Login selecting “ScientificNet”. Your account will be automatically unlocked after 5 minutes.

GitLab Basics

Repositories are called projects.

Groups are collections of projects. Projects can be created under the personal profile or in a group.

The visibility level determines who can view the repository

- Private → Access must be granted explicitly
- Internal → Any user with access to the server
- Public → Accessible from the outside

GitLab Basics

A person working on a GitLab project is called project member or contributor.

Contributors have different roles (permissions) in the project:

- Maintainer
- Developer
- Reporter
- Guest

Set Up SSH

Before we can start using our local GitLab instance, SSH must be configured:

1. Generate SSH Key (Link)
2. Add SSH Key to the GitLab account (Link)
3. Verify your setting (Link)

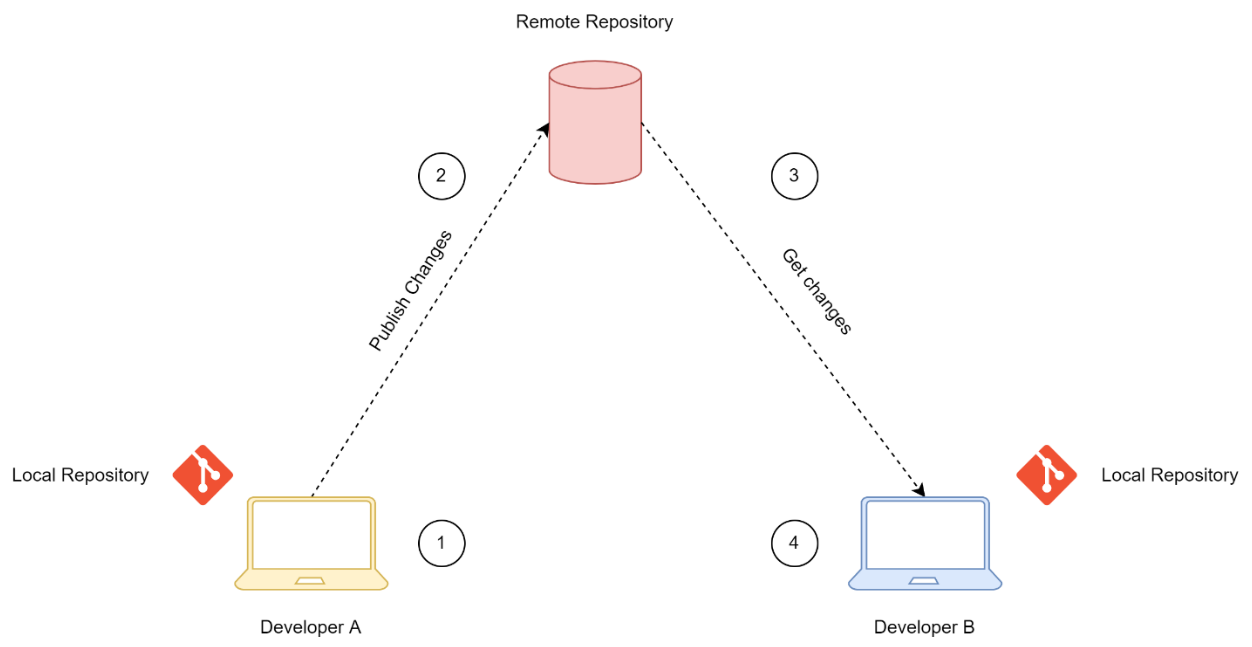
Hands-On

1. Create a project on GitLab
2. Clone it to your local computer
3. Create a text file in your local repository
4. Commit your changes and push them to the remote repository

Enabling Collaboration

Multiple users can work on the same remote repository

1. Work on local repository
2. Push changes to remote repository
3. Other contributors pull changes from the remote repository



Enabling Collaboration

To collaborate on a project, contributors must have access to the remote repository
→ Permissions

Changes to the same file are possible, but could lead to conflicts
→ Merge Conflicts

Hands-On

1. Clone the sample repository
2. Person A changes a file and pushes their changes
3. Person B pulls changes
4. Person B changes another file and pushes their changes
5. Person A pulls changes

Person A and person B could also work on different files simultaneously.

Branching and Merging

Motivation

So far, every commit was sequential.

This could lead to problems

- Work on different parts of the project simultaneously
- Fix a bug while you are making changes
- Testing
- Different deployment environments

Branch

A branch is a separate version of the main repository.

- Changes in one branch do not affect other branches
- Commits are independent from those in other commits
- Multiple branches are possible

Every repository has at least one branch: the master branch.

Master or Main?

Historically, the default branch in a repository was called master.
However, this term is considered offensive and antiquated.

As of 2022, GitHub and GitLab call the default branch main.

Merge

When the work on one branch is completed, a branch can be merged into another branch.

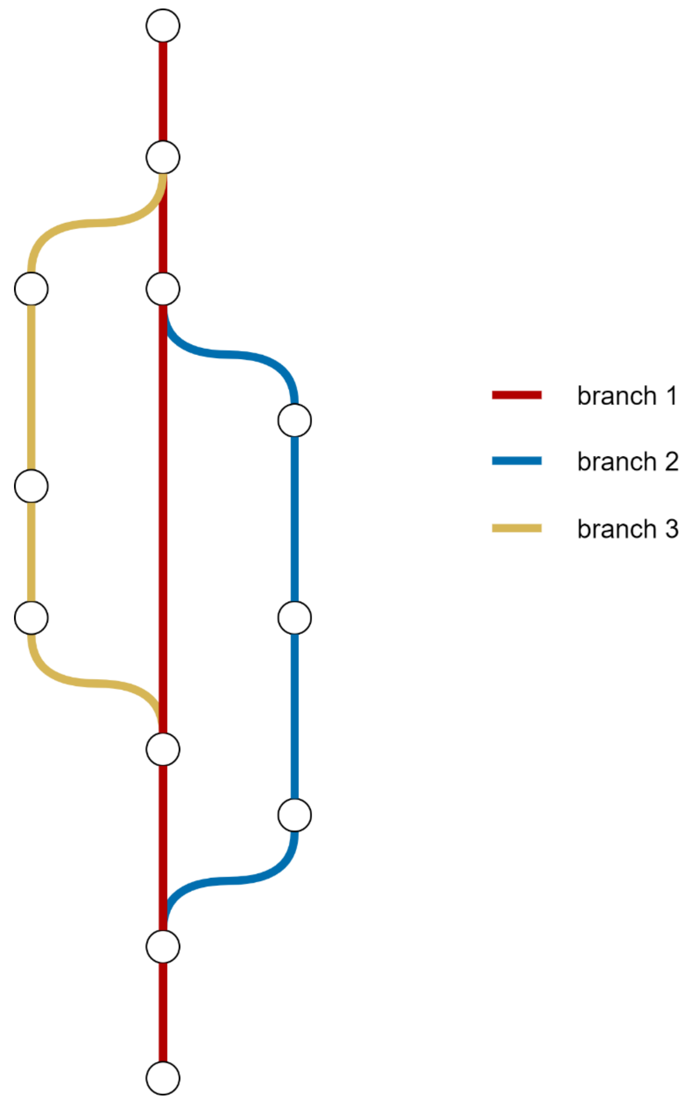
This integrates the changes of one branch with the changes on the other branch

Merge ≠ Overwrite

Representation

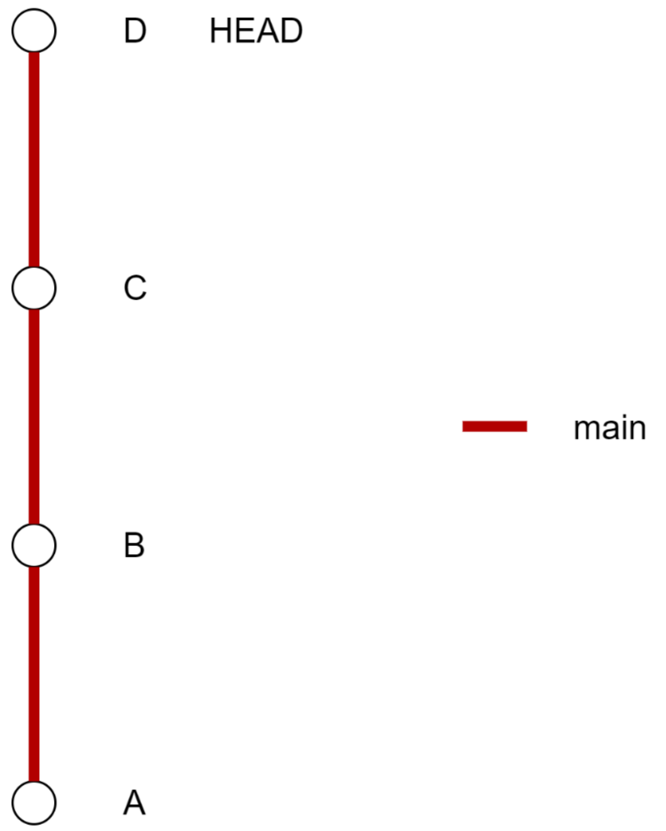
Commit history is represented using line diagrams

- Lines → Branches
- Nodes → Commits



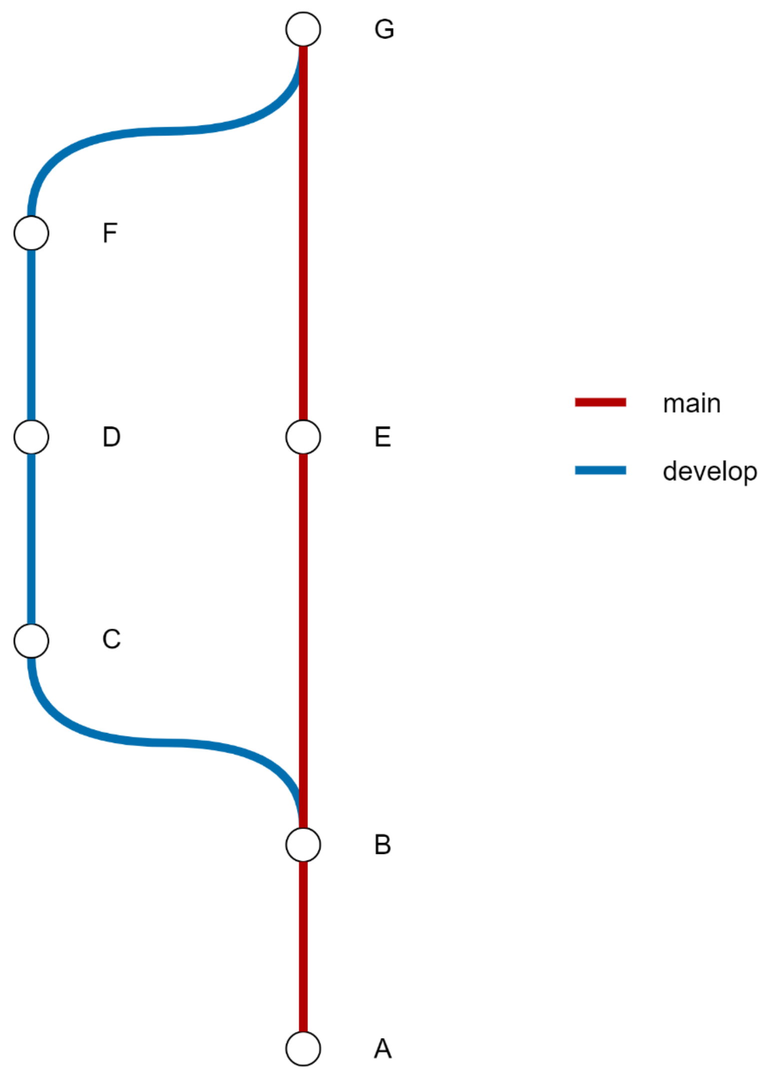
Examples

- 1 branch
- Sequential commits
- HEAD
  Reference to the last commit on the current branch



Examples

- 2 branches
- Branch develop created from commit B
- Branch develop merged into branch main with commit G

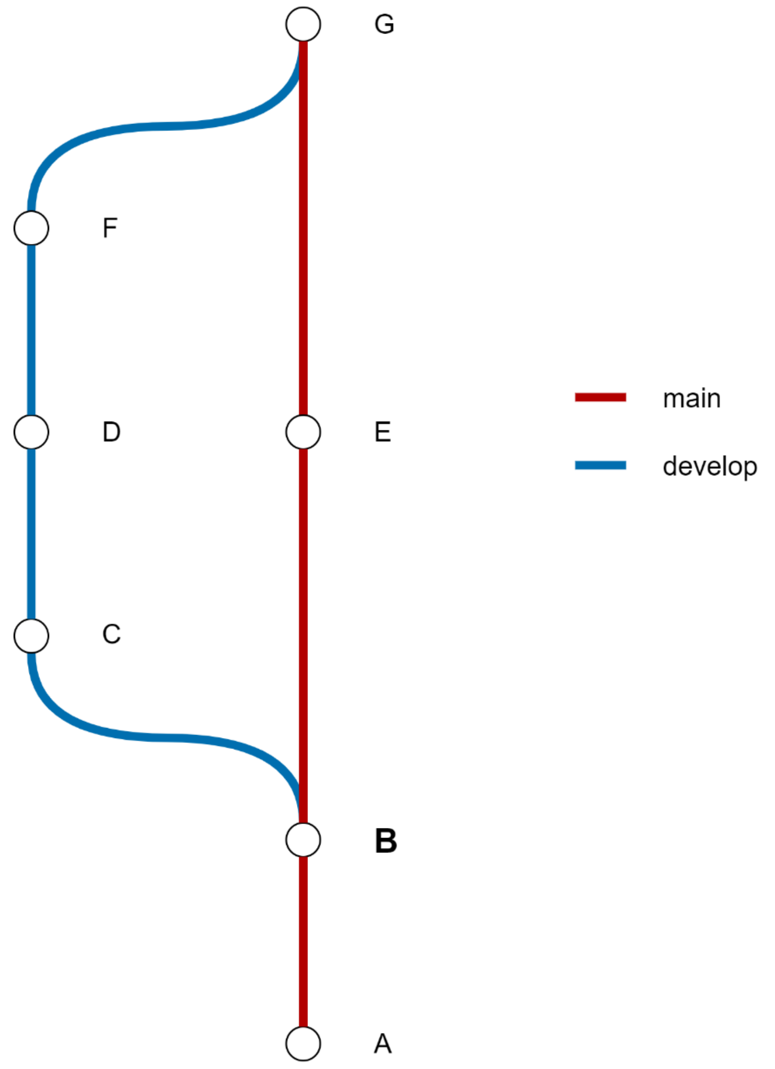


Creating Branches

A new branch is created from an existing branch.

The new branch “starts” from the last commit of this branch.

Example
Branch develop starts from commit B

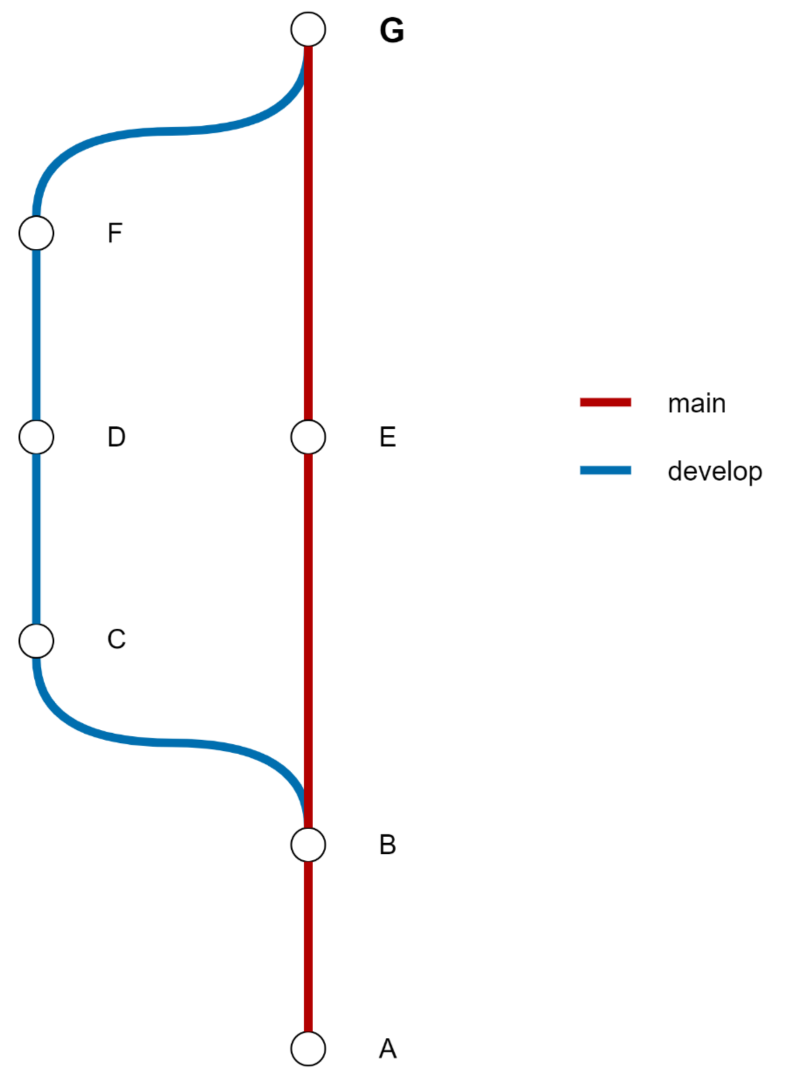


Merging Branches

Branches are merged with a merge commit.

This commit applies all changes from the source branch on the target branch.

Example
Branch develop is merged into branch main with commit G



Working with Branches

git branch

Creates a new branch, based on the last commit of the current branch

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

To start working on the new branch, you will have to move to the new branch.

Working with Branches

git checkout

Move to a specified branch

```
$ git checkout <branch-name>
```

To create a branch and immediately move to it, use the -b option

```
$ git checkout -b <branch-name>
```

Working with Branches

git merge

Merge a branch into the current branch

```
$ git merge <branch-name>
```

To use the command effectively:

1. Move to the branch you want to merge the changes into (target branch)
2. Run the merge command

```
$ git checkout <target-branch>
$ git merge <source-branch>
```

Hands-On

1. Create a second branch, called “develop”
2. Push the branch to the remote repository
3. Apply changes on the develop branch
4. Merge the develop branch back into the main branch

Merge Conflicts

Problem

Git's merge algorithm is very powerful:

- Changes from different files
- Changes in different parts of a file
- Renamed files

But what if 2 contributors changed the same part of the same file?
→ Merge Conflict

Merge Conflict

A merge conflict originates when Git is not able to automatically merge two branches.

Git augments the files that cannot be merged with text information. The contributor should resolve the conflict by their own.

This works for text files only.

Tip

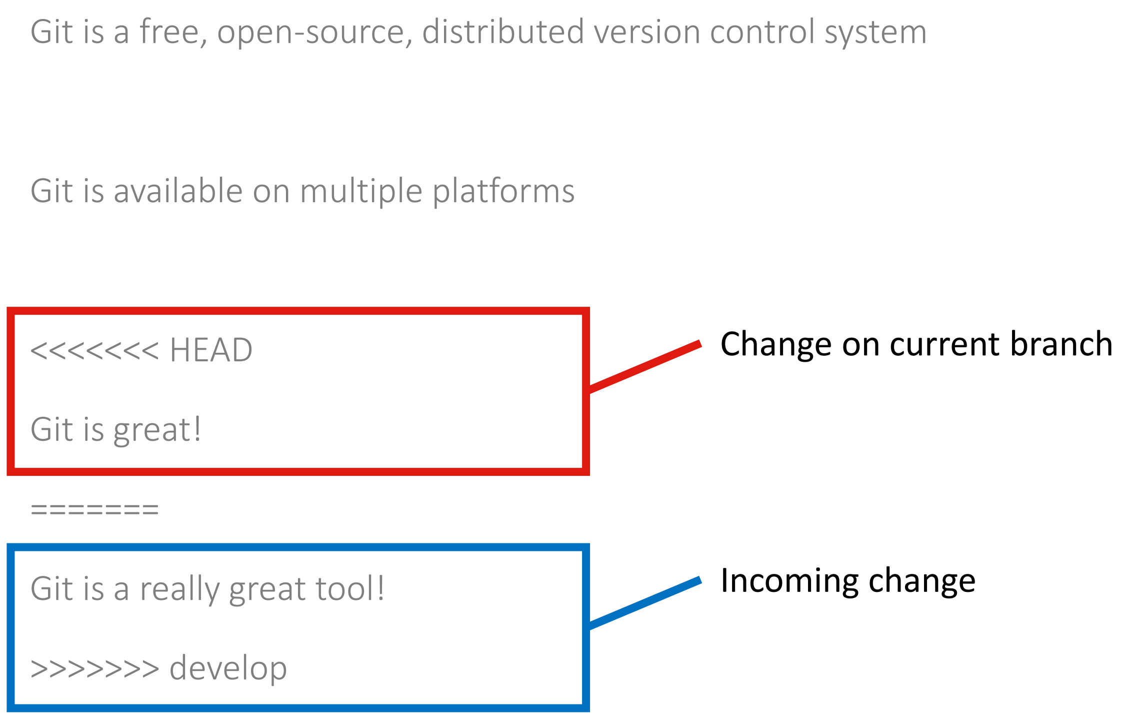
Code editors provide tools to resolve merge conflicts

Merge Conflict

Conflict information is just plain text

- Accept incoming change
- Accept local change
- Change the part completely

After all conflicts are resolved, stage and commit your changes



```
Git is a free, open-source, distributed version control system

Git is available on multiple platforms

<<<<<<< HEAD
Git is great!
=======
Git is a really great tool!
>>>>>>> develop
```

Hands-On

1. Create a file on the main branch and add some lines of text
2. Commit and push your changes
3. Create a new branch “develop”, based on the main branch
4. Modify a line in the file on the develop branch
5. Modify the same line on the main branch
6. Merge the develop branch into the main branch
7. Solve the merge conflict

Advanced Topics

Organizing Branches

Branches can be named and organized at wish.

Yet, a consistent and clean structured branching system helps organize the project better.

→ Git Flow (Link)

GitLab Features

- Merge Requests
- Change Comparison
- Repository Graph
- Issue Tracker
- CI/CD Pipelines