MarMic 2024

Git, GitHub/Gitlab

Nico Harms - 2024-01-30

Introduction

In this workshop, we'll explore the fundamentals of modern version control and collaboration using Git, along with platforms like GitHub and GitLab.

MarMic - Git, GitHub/Gitlab

Introduction

- Who is this for?
  - This workshop is crucial for software developers, data scientists, and anyone involved in collaborative projects.
- Why is this important?
  - In today's fast-paced and interconnected world, effective collaboration, code management, and version control are essential.
- By the end of this workshop, you will:
  - Create and manage Git repositories.
  - Know how to collaborate with team members and peers on shared projects.

MarMic - Git, GitHub/Gitlab

Schedule

Day 1 - Introduction to Git and Basic Concepts

- Welcome and Introduction
- Understanding Version Control
- Introduction to Git
- Git Basics
- Git Workflow

Day 2 - Branching, Collaboration, and Advanced Topics

- Recap of Day 1
- Git Branching and Merging
- Remote Repositories
- Collaborative Workflows
- Advanced Topics (time permitting)

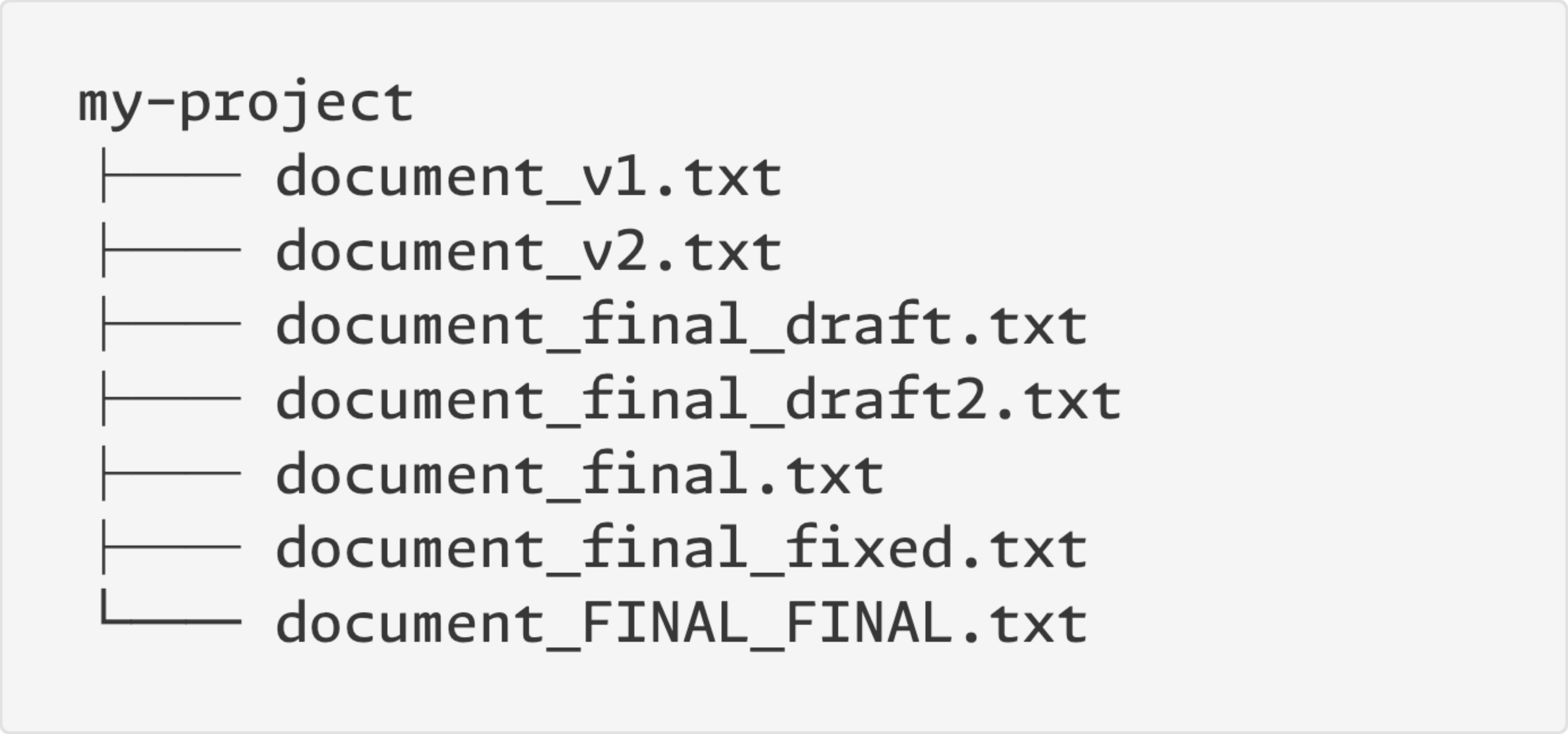
Day 1
Introduction to Git and Basic Concepts

Understanding Version Control

Understanding Version Control

The Problem

- Have you ever found yourself juggling multiple versions of a file, like "document.txt," "document_v2.txt," and „document_final.txt"?



```
my-project
 ├─── document_v1.txt
 ├─── document_v2.txt
 ├─── document_final_draft.txt
 ├─── document_final_draft2.txt
 ├─── document_final.txt
 ├─── document_final_fixed.txt
 └─── document_FINAL_FINAL.txt
```

Understanding Version Control

The Solution

- A version control system is an invaluable tool that empowers you to monitor changes to your files and collaborate seamlessly on projects.
- Types of VCS (e.g. Subversion, CVS, Mercurial, git)
- Advantages:
  - Chronological record
  - Easily to revert
  - Collaborate efficiently

Understanding Version Control

A brief history

- 1970s-1980s: Emergence of early systems like IBM's **Source Code Control System** (SCCS) and the Revision Control System (RCS).
- 1990s: Development of **Concurrent Versions System** (CVS), supporting concurrent work by multiple developers.
- Early 2000s: Introduction of **Subversion** (SVN), offering atomic commits and enhanced branching and merging capabilities.
- 2005: Birth of **Git** by Linus Torvalds, focusing on speed, efficiency, and distributed version control. Git quickly became the global standard for software development.
- „Version Control Light“ - Various Cloud providers since 2007

Introduction to Git

Introduction to Git

Git is a powerful version control system

- Tracks changes made to files over time
- Allows creation of branches for independent work
- Facilitates seamless collaboration and merging of changes

Introduction to Git

Basic Git Commands

- `git init`: Initialize a new Git repository
- `git status`: Check the status of your repository
- `git add`: Add changes to the staging area
- `git commit`: Create a new commit with the staged changes
- `git log`: View a log of your commit history
- `git diff`: Compare changes between different versions of your files

Introduction to Git

Real-World Applications of Git

- **Academia**: Tracking changes in research papers and collaboration
- **Data Science**: Managing code and data, fostering team collaboration
- **Design**: Version controlling creative work and collaboration
- **Writing**: Tracking writing versions and collaboration with editors and authors

Git Basics

Git Basics

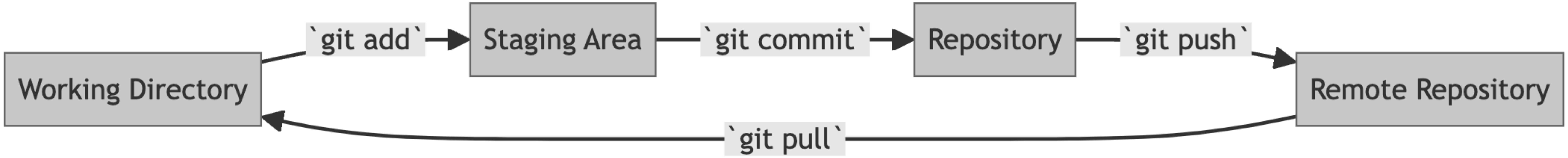
Git Terminology

- **Repository**: Container for your project, holding all files, directories, and history of changes.
- **Commit**: Snapshot of your project's files at a specific point in time, including changes made since the last commit.
- **Branch**: Parallel version of your repository, allowing work on new features or changes without affecting the main codebase.

Git Basics

Three Main Stages of Git

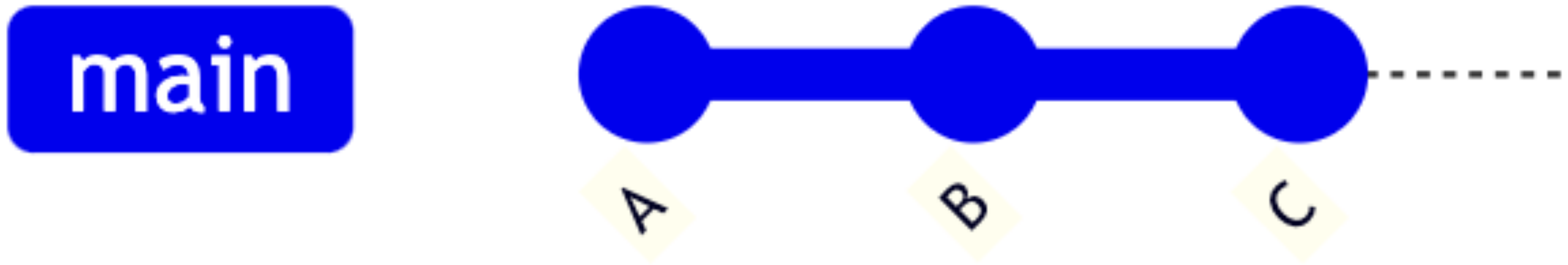
- **Working Directory**: Where you make changes to your code.
- **Staging Area**: Temporary holding space for reviewing and selecting changes for the next commit.
- **Repository**: Stores snapshots of your project's files, creating a history of commits.
- **Remote Repository**: Copy of the repository stored on a remote server, enabling collaboration, backup, sharing, and pulling changes.



Git Basics

What is a Commit?

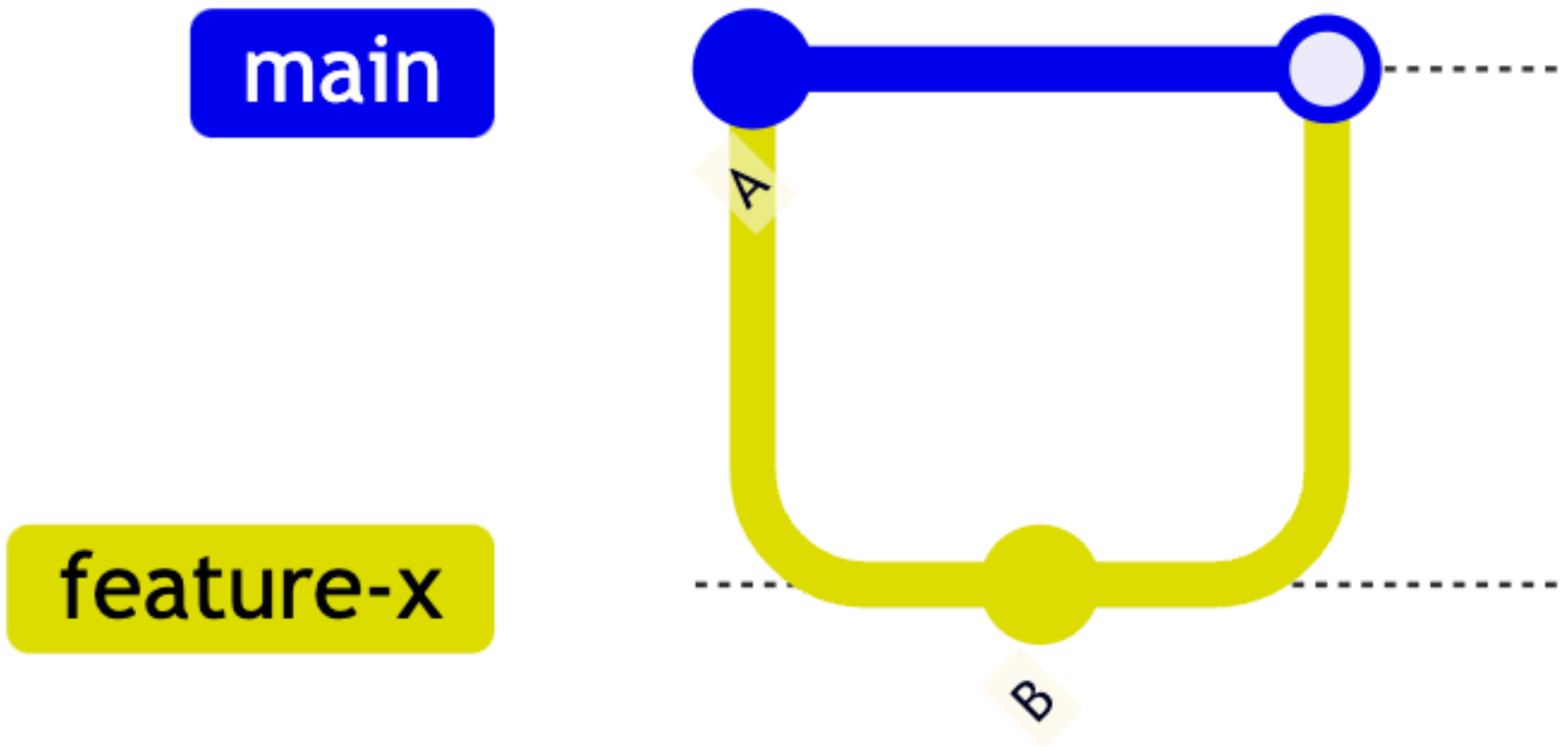
- **Definition**: A snapshot of your project's files at a particular moment in time.
- **Contents**: Changes made to the files since the last commit, metadata, and a unique identifier (SHA).
- **Linear History**: Commits are stored sequentially, helping track progress, collaborate, and revert to previous project versions.



Git Basics

What is a Branch?

- **Definition**: A parallel version of your repository for independent work.
- **Creating a Branch**: Starts based on the current state of the repository.
- **Isolation**: Allows experimentation with new features or changes without impacting the main version.



Git Workflow

Git Workflow

Key Concepts

- Git operates with a local repository on your machine.
- Changes are stored locally until you use the `git push` command.

Git Workflow

Example Workflow

1. Initialize a new Git repository using the `git init` command.
2. Make changes to the project files.
3. Use the `git status` command to see the changes made in the working directory.
4. Use the `git add` command to add changes to the staging area.
5. Commit your changes using the `git commit` command.
6. View the commit history using the `git log` command.

Git Workflow

Additional Commands

- `git diff`: See the differences between the working directory and the latest commit.
- `git show`: See the details of a specific commit.
- `git log --all --graph --oneline`: Show a graphical representation of the commit history, including all branches and commits.

Hands On

E1

Basic git configuration

- Open https://training.hub.gfbio.dev in your browser
- Go into your terminal Initially set up your user:
  - `git config --global user.name "Your Name"`
  - `git config --global user.email "your.mail@example.com"`
  - `git config --global init.defaultBranch main`
- Verify that your git user and mail are set correctly
  - `git config --list`

E2

- Create your first repository

- Create a new directory inside of your terminal for your project

- Initialize a new git repository in that directory by running the command `git init`

- In the directory you initialized as a git reposiory create a new file called `README.md`

- Fill the file with some content

E3

Create your first commit

- Modify the `README.md` file and save it
- Add this file to the staging area with `git add README.md`
- Now you can commit via `git commit --message "Your specific commit message"` to the repository
- Repeat this process 2 more times and choose good commit messages each time.

E4

Multiline Commit Message

- Modify your `README.md` file and save it.
- Add the file to the staging area
- When commiting the file ommit the `—message` part. This will open an editor.
- Within this editor you may write longer commit messages. The first line will be most prominent, therefore set is wisely.
- When you are done writing your message, save and close the file (`Ctrl+x` for nano, `:wq` for vi/vim)
- Take a look at `git log` now.

E5

Displaying differences

- When entering `git diff` in the terminal you will see the difference between of the unstaged files and the rest of your repository
- By using `git diff [<commit hash>|<branch>]` you may compare the current state of the repository with a specific commit or branch
- With `git diff [<commit hash>|<branch>] [<commit hash>|<branch>]` You can compare branches with branches, commits with commits, branches with commits and the other way around.

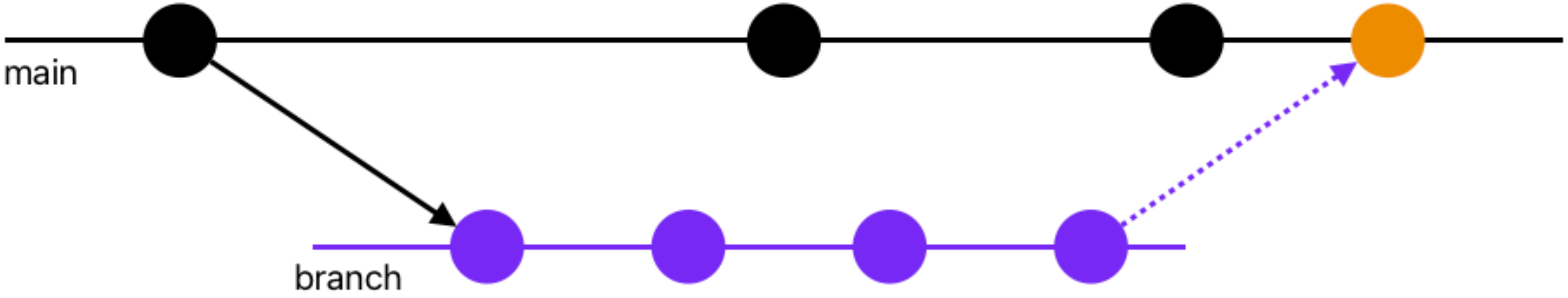
Day 2
Branching, Collaboration and Advanced Topics

Git Branching and Merging

Git Branching and Merging

What are Git Branches again?

- Git branches are separate lines of development.
- Developers can work on different features or bug fixes simultaneously.
- Branches don't impact the main codebase until merged.
- Completed work can be merged back into the main branch.



Git Branching and Merging

Creating a Branch

- **Creating a Branch**: Use `git switch --create <branch-name>` to create a new branch.

- **Switching Between Branches**: Use `git switch <branch-name>` to switch to a different branch. Changes can be made to the branch files using `git add` and `git commit`.

- **Merging Branches**: Use `git merge <branch-name>` to merge changes from a branch back into the main branch. This creates a new commit representing the merge.

- **Handling Merge Conflicts**: Conflicts occur when the same lines of code are modified in both branches. Git marks conflicts in files with special markers, and they must be resolved manually.

Git Branching and Merging

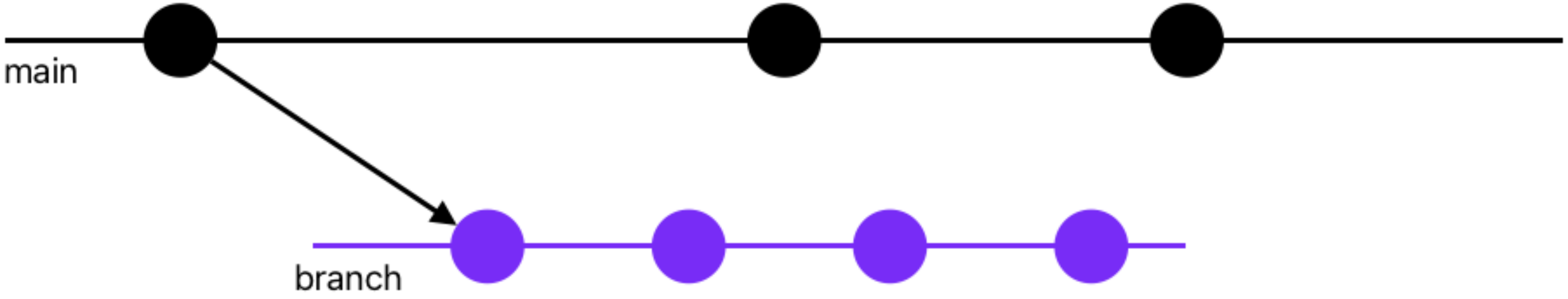
Merge Strategies

- **Fast-Forward Merge**: Moves the current branch to the latest commit of the branch being merged when branches have not diverged.
- **Merge Commit**: Creates a new commit with multiple parents when merging diverged branches.
- **Forcing a Merge Commit**: Use `git merge --no-ff` to force a merge commit even when a fast-forward merge is possible.
- **Squashing Commits**: Combine all branch commits into a single commit with `git merge --squash`.

Git Branching and Merging

Rebasing

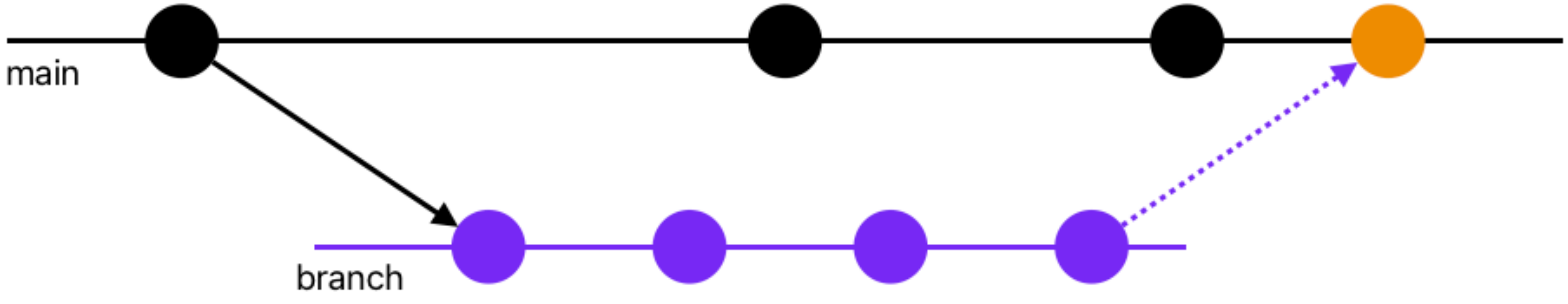
- Include commits from other branches on my work



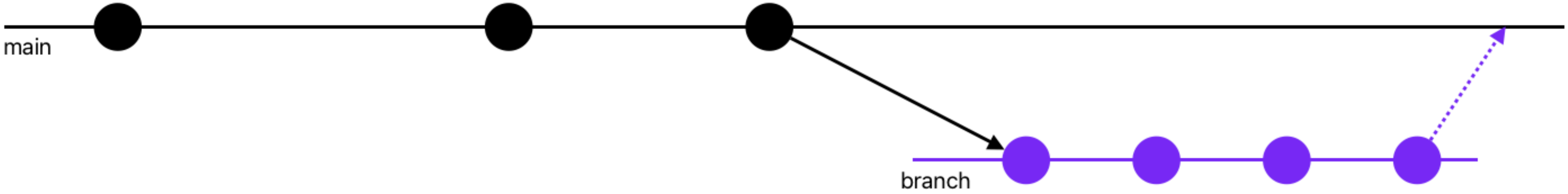
Current state of the repository

Git Branching and Merging

Rebasing



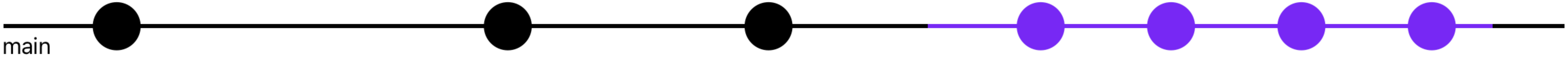
Creating a branch, doing work and merging without rebase



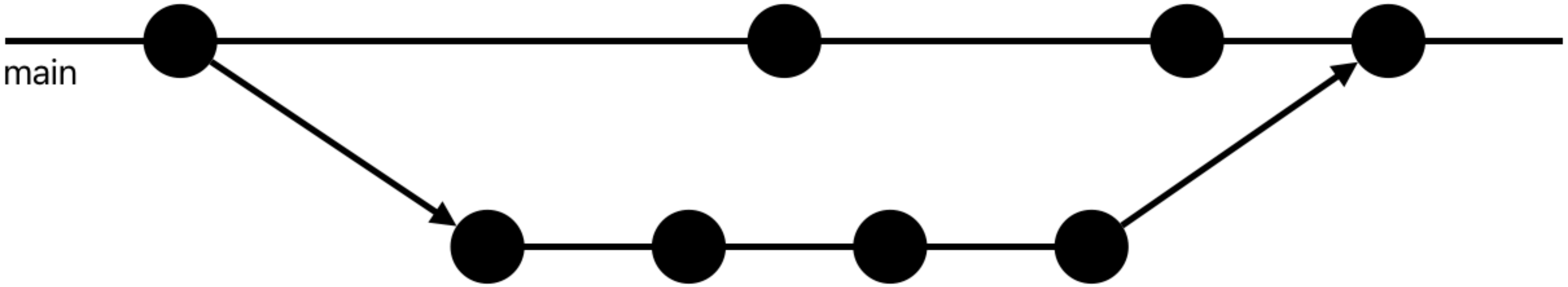
Using `git rebase` before merging

Git Branching and Merging

Rebasing



Resulting graph of main branch with rebasing



Resulting graph of main branch without rebasing

Collaborative Workflows

Collaborative Workflows

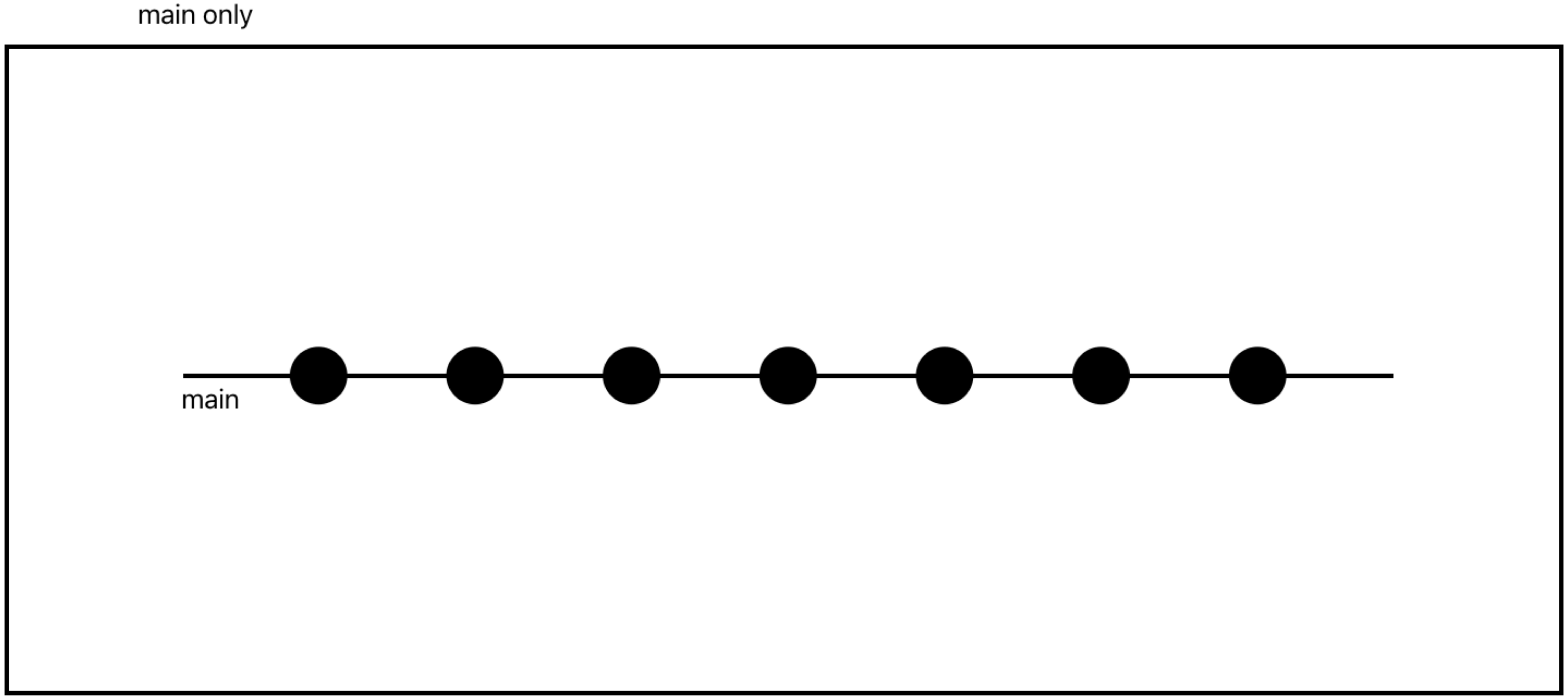
Project Factors to Consider

- **Project Type & Size**: Match the workflow to your project's complexity and scale.
- **Team Dynamics:** Ensure the workflow fits your team's size and collaboration style.
- **Developer Expertise**: Choose a workflow that suits your team's Git proficiency.
- **Agility vs. Structure**: Decide on the level of flexibility and organization your project needs.
- **Project Lifecycle**: Adjust the workflow according to your project's maturity.

Collaborative Workflows

- main only
- main/dev
- Feature Branch/Forking
- GitHub/GitLab Flow
- Trunk-Based Development
- Gitflow

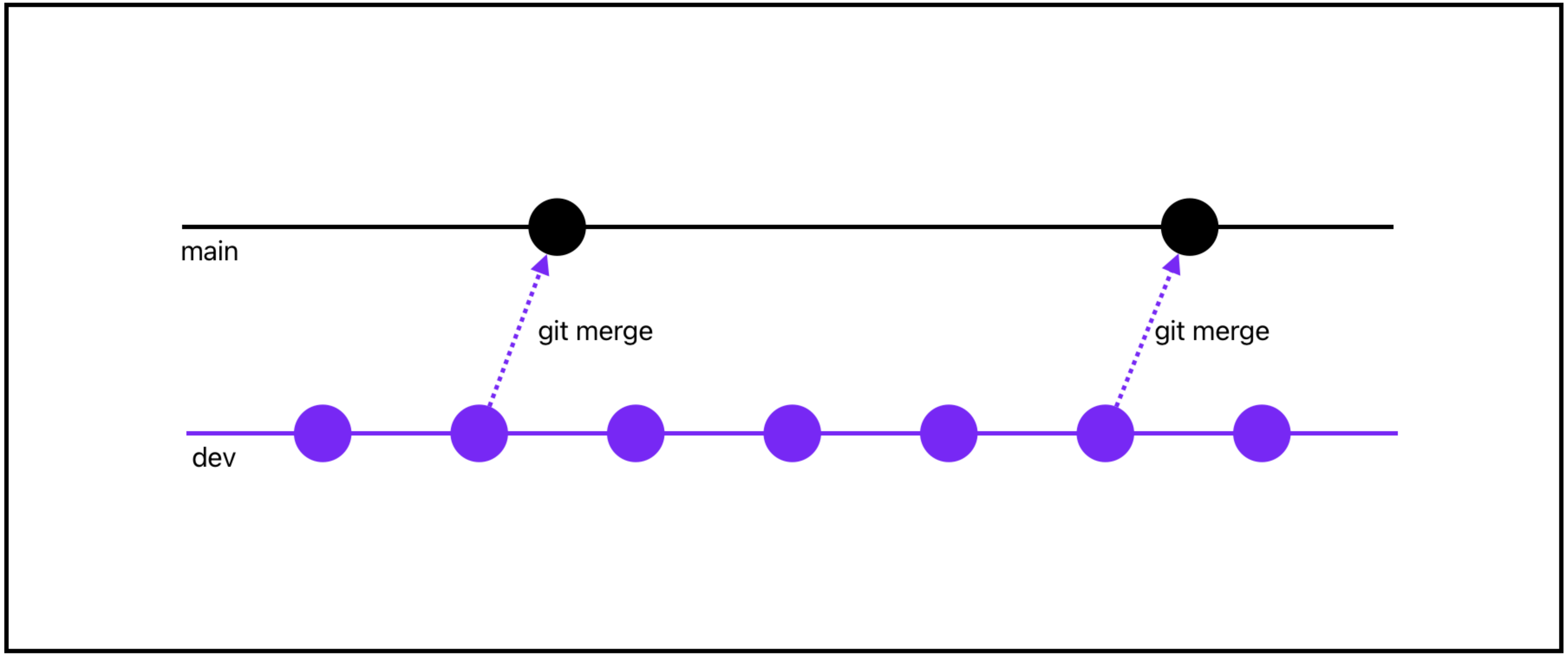
Collaborative Workflows



Collaborative Workflows

main/dev

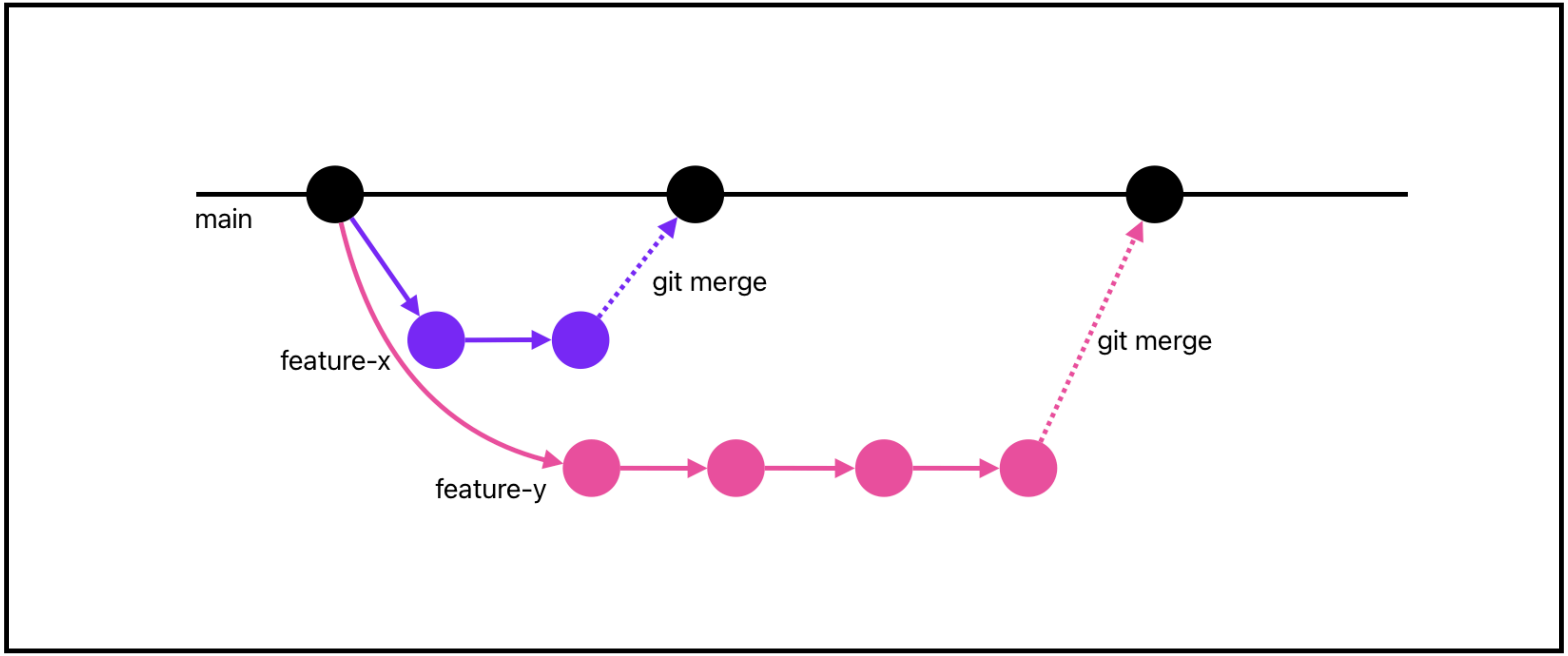
main/dev only



Collaborative Workflows

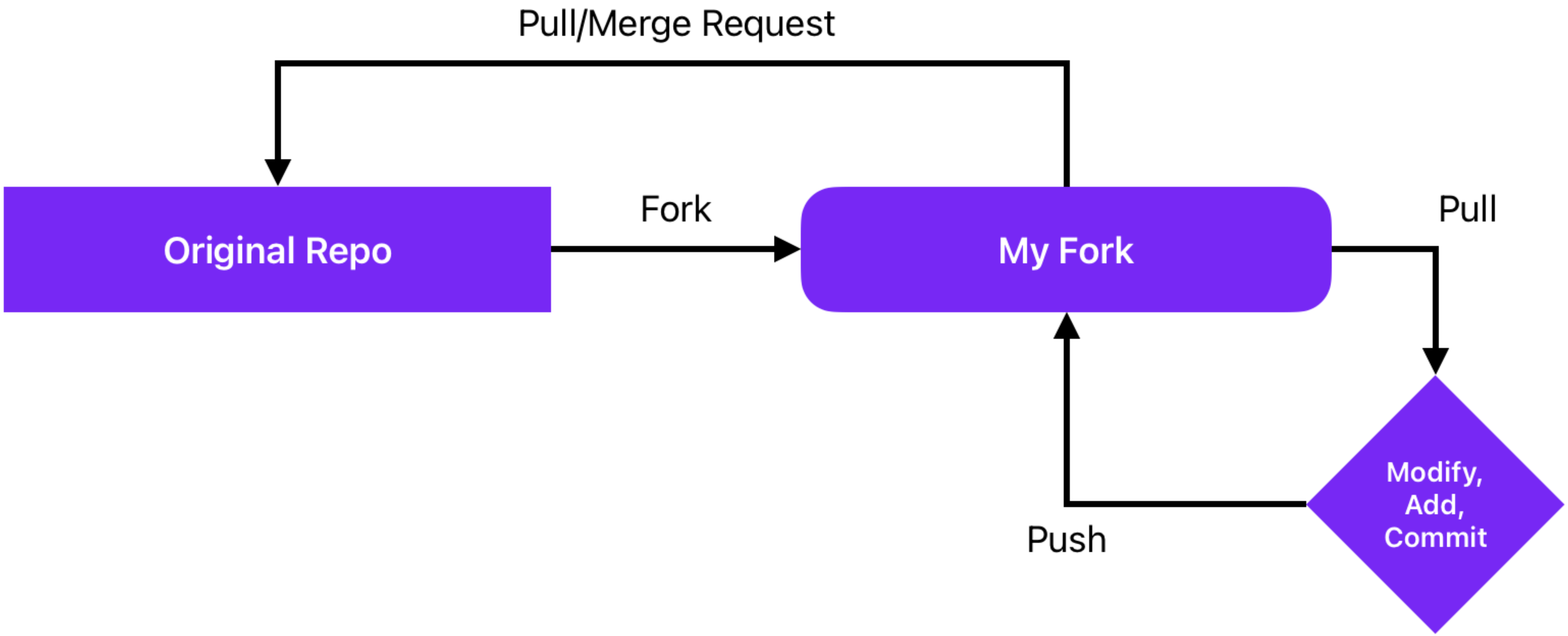
Feature Branch/Forking

Feature Branch/Forking



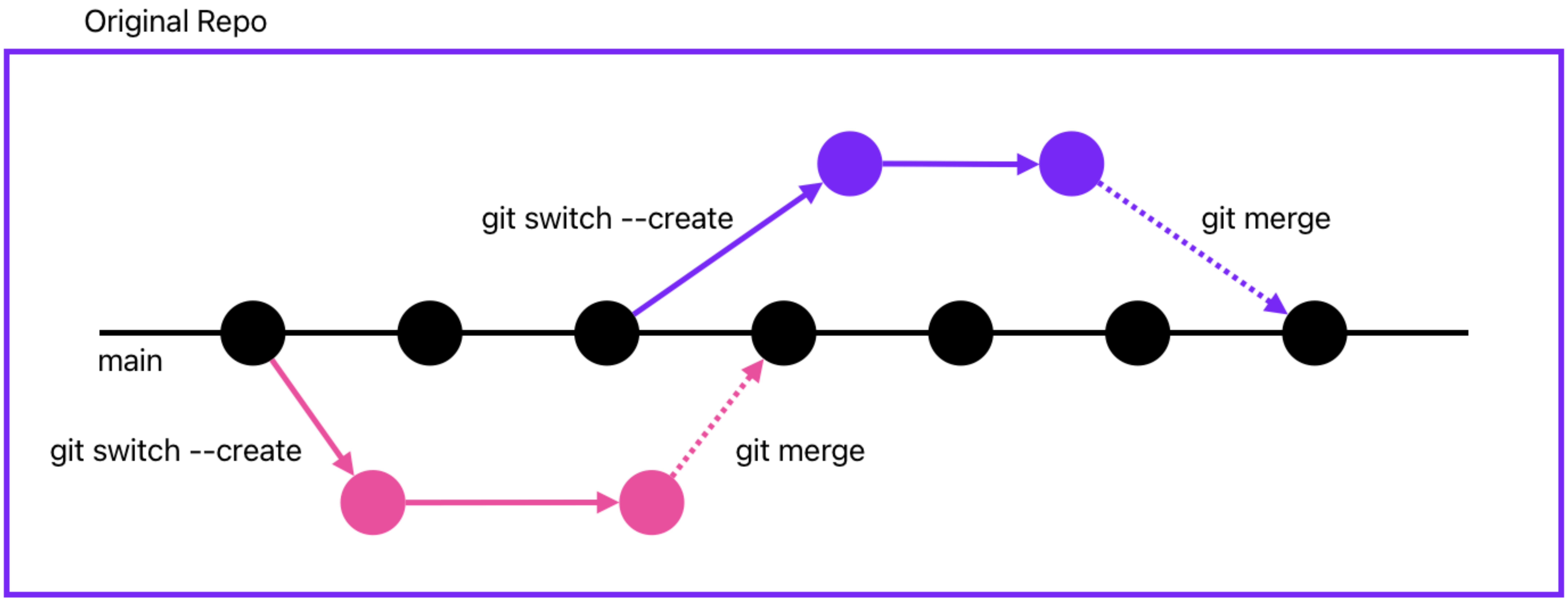
Collaborative Workflows

GitHub/GitLab Flow



Collaborative Workflows

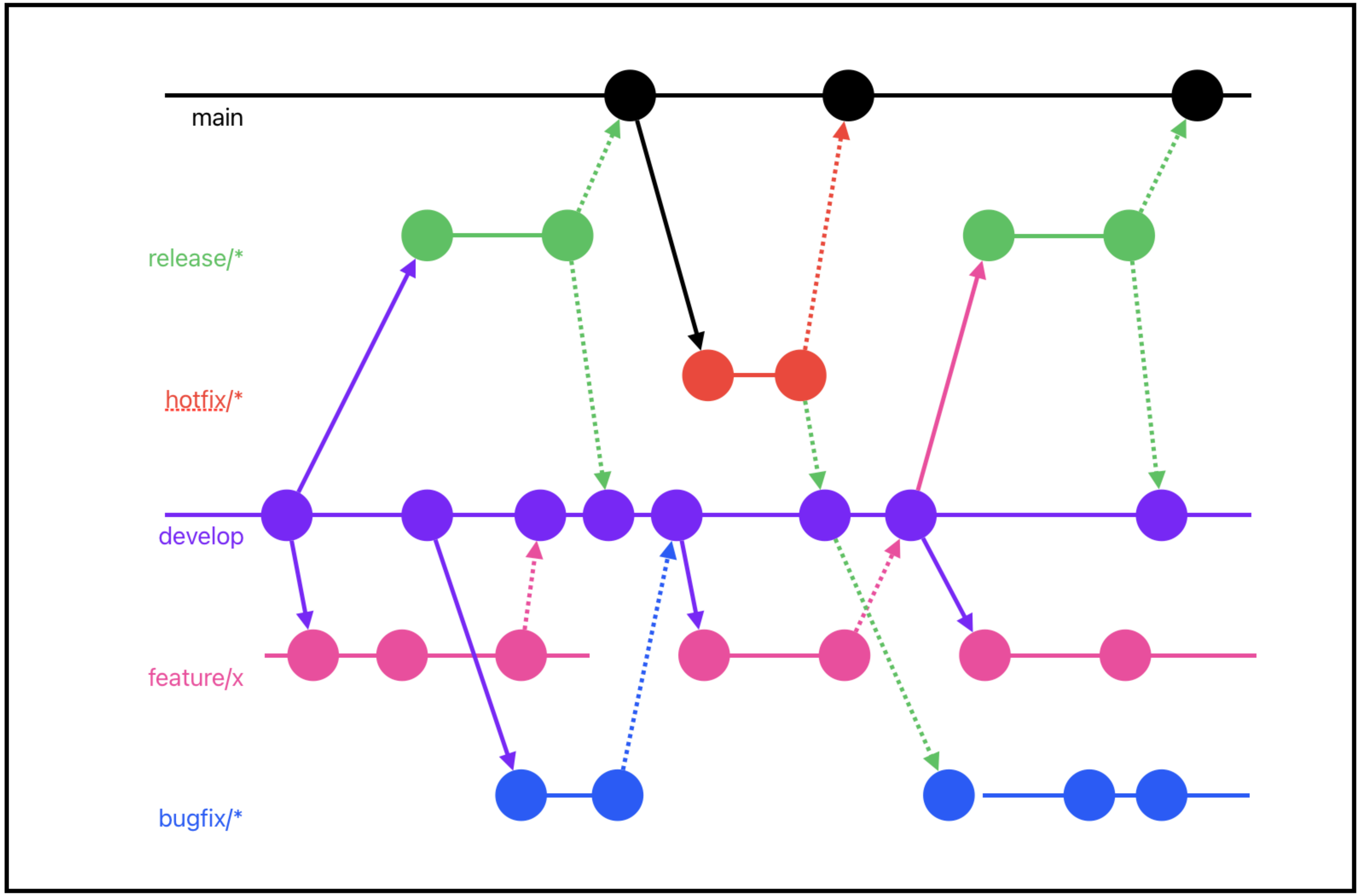
Trunk-Based Development



Collaborative Workflows

Gitflow

Trunk base development



Remote Repositories

Remote Repositories

GitHub and GitLab

- Web-based interface for viewing and editing files
- Collaborative coding with team members
- Built-in code review tools
- Issue tracking and project management
- Automatic backups and versioning of code

Remote Repositories

Features

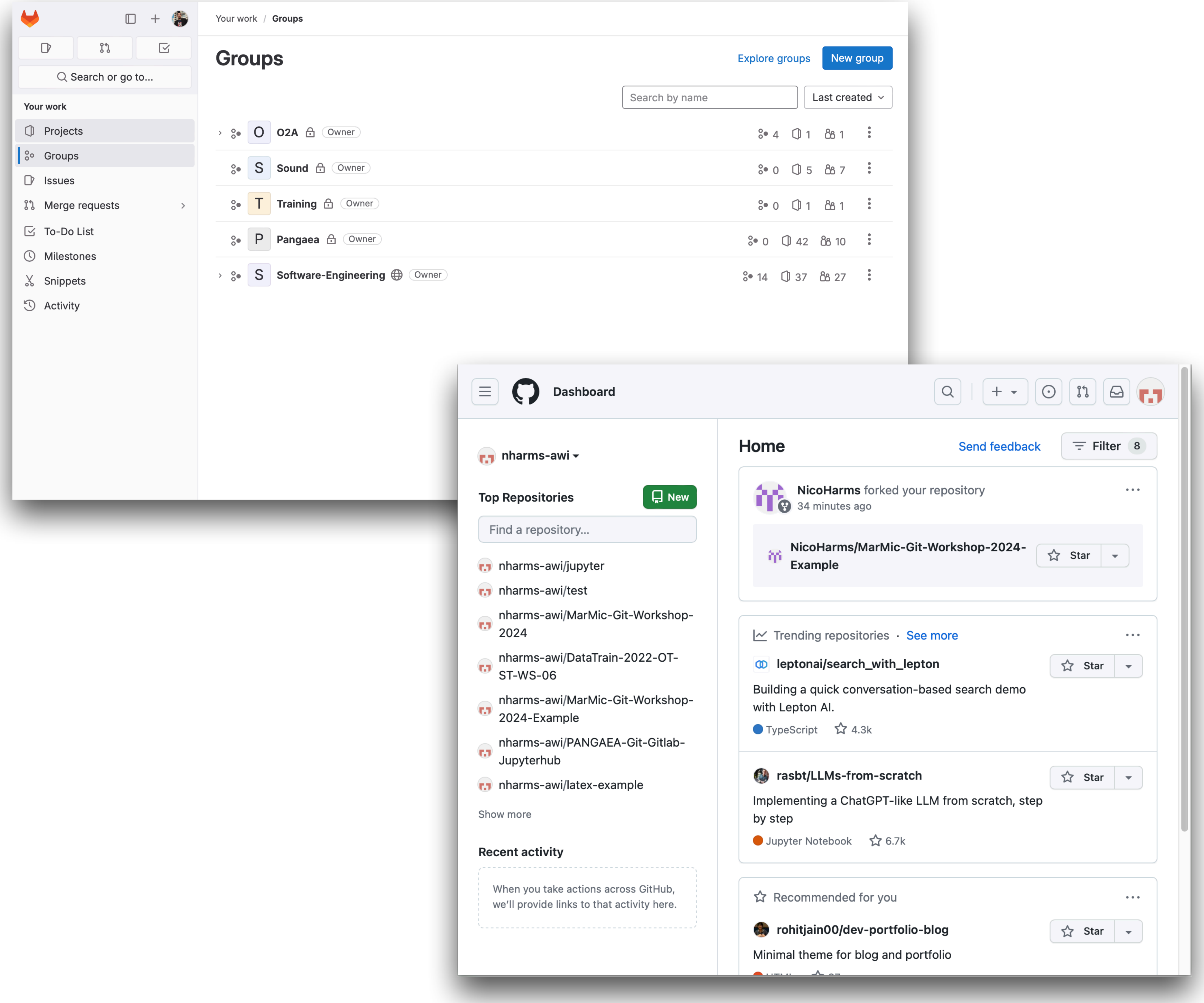
- More Features
- Pull/Merge Requests
- Collaboration
- Wikis and Pages
- Continuous Integration
- More…

Remote Repositories

Working with remotes

- git clone: Copy the default branch to your drive
- git fetch: Update the current branch of the local repository with changes from remote
- git pull: Update the current branch of the local repository and working area with changes from remote
- git push: Push the current branch to the remote, with all committed changes

GitHub/GitLab



Your work / Groups

Groups

Explore groups · New group

Search by name · Last created

O2A	Owner	4	1	1
Sound	Owner	0	5	7
Training	Owner	0	1	1
Pangaea	Owner	0	42	10
Software-Engineering	Owner	14	37	27

Your work

- Projects
- Groups
- Issues
- Merge requests
- To-Do List
- Milestones
- Snippets
- Activity

Search or go to...

Dashboard

nharms-awi

Top Repositories · New

Find a repository...

- nharms-awi/jupyter
- nharms-awi/test
- nharms-awi/MarMic-Git-Workshop-2024
- nharms-awi/DataTrain-2022-OT-ST-WS-06
- nharms-awi/MarMic-Git-Workshop-2024-Example
- nharms-awi/PANGAEA-Git-Gitlab-Jupyterhub
- nharms-awi/latex-example

Show more

Recent activity

When you take actions across GitHub, we'll provide links to that activity here.

Home · Send feedback · Filter 8

NicoHarms forked your repository
34 minutes ago

NicoHarms/MarMic-Git-Workshop-2024-Example · Star

Trending repositories · See more

leptonai/search_with_lepton · Star
Building a quick conversation-based search demo with Lepton AI.
TypeScript · 4.3k

rasbt/LLMs-from-scratch · Star
Implementing a ChatGPT-like LLM from scratch, step by step
Jupyter Notebook · 6.7k

Recommended for you

rohitjain00/dev-portfolio-blog · Star
Minimal theme for blog and portfolio

Advanced Topics

Advanced Topics

Troubleshooting and Special Files

- **Troubleshooting**: Common issues include conflict resolution, unwanted commits, and recovering lost commits. Understanding how to troubleshoot these issues is crucial for effective Git usage.

- **Special Files**: Git and GitHub treat certain files differently, using them for configuring repositories and providing documentation. These include `.gitignore` and `README.md`.

Advanced Topics

Rebasing, Tags, and Stashing

- **Rebasing**: Allows you to integrate changes from one branch into another by reapplying commits. Useful for keeping your branch up to date with the main branch without creating a new merge commit.
- **Tags**: Labels you can apply to specific commits. Useful for marking significant versions of your code, such as release versions.
- **Stashing**: Allows you to store your work in progress and switch to another branch or address unexpected changes. Useful when navigating complex workflows and addressing unforeseen challenges.

Wrap-up and Next Steps