Git/GitHub

DataTrain 2024 - OT-ST-2024-06

Nico Harms - 2024-06-03

Nico Harms

Software Engineer

AWI

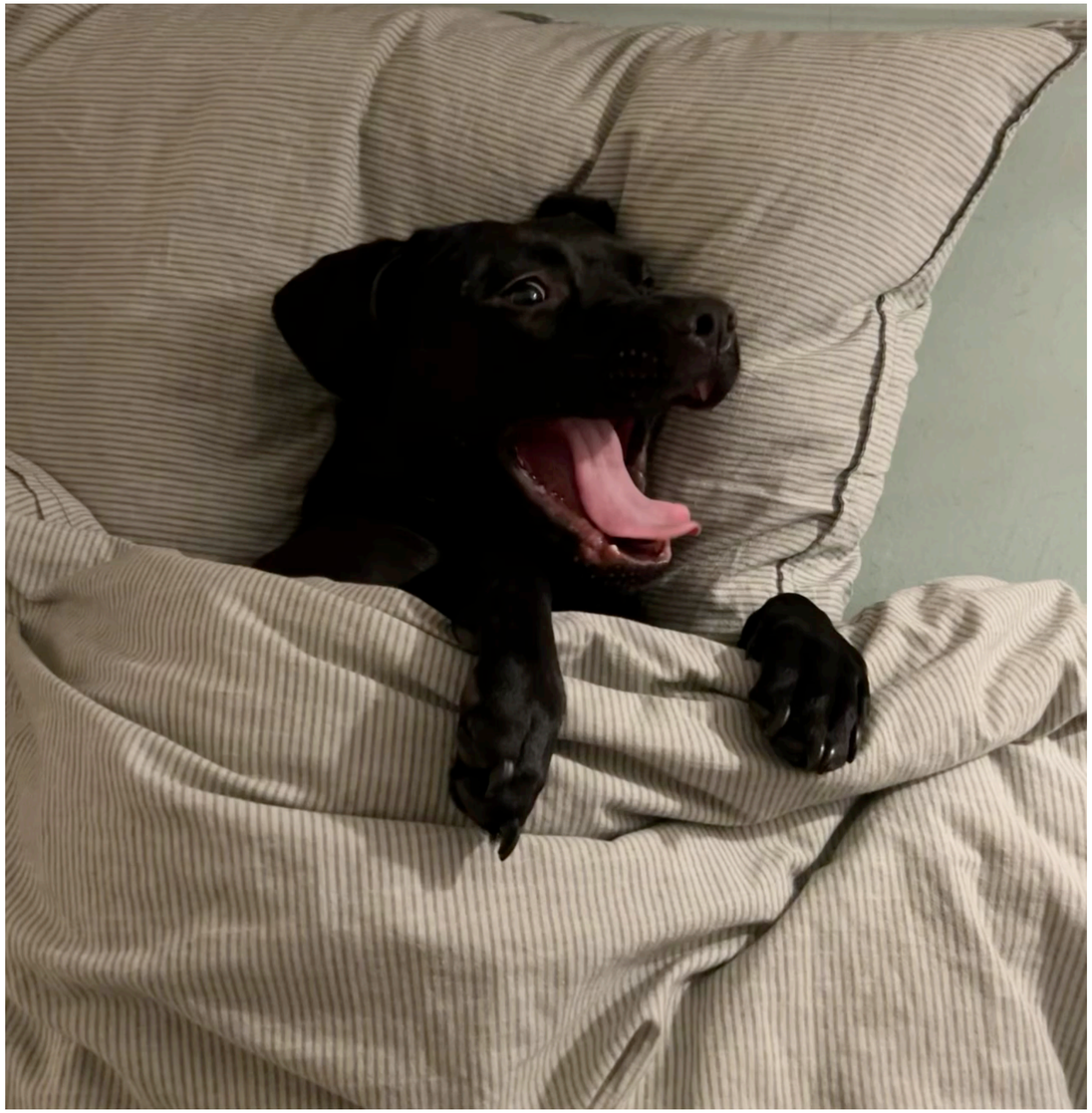
Shiva

Director of Cuddle Operations

Shiva

Responsible for:

- Cuddles
- Chaos



Links

Workshop Repository

https://github.com/nharms-awi/DataTrain-2024/

Fragen

https://github.com/nharms-awi/DataTrain-2024/issues

Introduction

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

Git/GitHub - DataTrain 2024

Introduction

- Who is this for?
  - This workshop is 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.

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup
- Introduction to the CLI
- Hands-On: CLI
- Why Version Control?
- What is Git?
- What is GitHub?
- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files
- Branching and Merging
- Hands-On: Branching and Merging
- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration
- Q&A and Wrap-Up

Checking Setup

https://shorturl.at/HzTdh

- Check the installed Version of git via `git --version`
- Check your git configuration via

```
git config --global user.name

git config --global user.email
```

- Verify your git configuration and setup via the „Verify your setup“ script

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup
- Introduction to the CLI
- Hands-On: CLI
- Why Version Control?
- What is Git?
- What is GitHub?
- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files
- Branching and Merging
- Hands-On: Branching and Merging
- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration
- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup
- Introduction to the CLI
- Hands-On: CLI
- Why Version Control?
- What is Git?
- What is GitHub?
- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files
- Branching and Merging
- Hands-On: Branching and Merging
- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration
- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Introduction to the Command Line Interface (CLI)

- Definition: A text-based interface to interact with the operating system.
- Contrast with GUI: Instead of windows, icons, and buttons, commands are typed.
- Importance:
  - Efficiency and precision.
  - Preferred by developers, system administrators, and advanced users.

Why is the CLI Important?

- Speed: Faster than using a GUI for many tasks.
- Flexibility: Greater control over system operations.
- Power: Ability to perform complex tasks with simple commands.
- Example users: Developers, System Administrators.

Basic CLI Commands - Navigating Directories

- cd (change directory)
  - Usage: `cd directory_path`
  - Example: `cd Documents`
- ls (list)
  - Usage: `ls [options] [directory]`
  - Example: `ls, ls -l`

Creating and Managing Files and Directories

- mkdir (make directory)
  - Usage: `mkdir directory_name`
  - Example: `mkdir new_folder`
- rm (remove)
  - Usage: `rm [options] file_or_directory`
  - Example: `rm file.txt`, `rm -r directory_name`
- cp (copy)
  - Usage: `cp source destination`
  - Example: `cp file.txt /home/user/Documents/`
- mv (move)
  - Usage: `mv source destination`
  - Example: `mv file.txt /home/user/Documents/`, `mv old_name.txt new_name.txt`

Viewing File Contents

- nano
  - Usage: `nano file_name`
  - Example: `nano file.txt`
  - Commands: `Ctrl + X` to exit, `Y` to save changes
- vim
  - Usage: `vim file_name`
  - Example: `vim file.txt`
  - Commands:
    - `i` to enter insert mode
    - `Esc`, `:wq` to save and exit
    - `Esc`, `:q!` to exit without saving

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Hands-On: Practice with Basic CLI Commands

https://shorturl.at/8N83C

- Step 1: Create repositories folder: `mkdir ~/repositories`
- Step 2: Create datatrain-cli folder: `mkdir ~/repositories/datatrain-cli`
- Step 3: Navigate to folder: `cd ~/repositories/datatrain-cli`
- Step 4: Create file: `touch hello.txt`
- Step 5-7: Edit hello.txt (Include brief commands for nano and vim)
- Step 8-10: Manage subfolder and files within it (Create, copy, and view file contents)
- Step 11-15: Advanced file management (Edit, delete, rename, list directory contents, free practice)

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Why Version Control?

- Importance in collaborative projects (software development, writing, research).
- Managing changes and tracking progress becomes challenging as projects grow.
- Introduction to Version Control System (VCS):
  - Tool for managing changes to documents, programs, and files.
  - Tracks and merges changes from multiple contributors.
  - Allows reversion to previous versions, comparing iterations, and maintaining history.

Benefits of a Version Control System

- Streamlines collaboration and reduces conflict risks.
- Enhances productivity by providing a reliable version management system.
- Essential for maintaining order and efficiency in both solo and team projects.

Common Challenges in Document Iteration

- Manual version tracking becomes impractical and chaotic.
- Example of document versioning:
  - Initial file: document.txt
  - Progression: document-v2.txt, document-v2-final_draft.txt
  - Feedback incorporation: document-final-fixed.txt, document-FINAL-FINAL.txt
- Result: A cluttered project folder with multiple active versions.

Iterations of Working Documents

- Example project folder:

```
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
```

What Is a Version Control System?

- Comparison with book editions.
- Essential functions of a VCS:
  - Track changes across multiple files.
  - Compare different versions.
  - Restore previous states.
  - Facilitate collaboration.
- Importance in software development to avoid bugs and manage extensive codebases.

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

What is Git?

- Definition: Git is a free and open-source distributed version control system.
- Purpose: Designed for handling projects of all sizes with speed and efficiency.
- Creation: Developed by Linus Torvalds in 2005 for managing the Linux kernel's source code.

What is Git?

How Git Works - Distributed Version Control

- Distributed Approach: Each developer has a local copy of the entire repository, including its complete history.
- Advantages:
  - Speed: Local operations are extremely fast.
  - Flexibility: Work offline and commit changes locally.
  - Resilience: Reduced risk of data loss due to replication.

What is Git?

Snapshots, Not Differences

- Data Storage: Git stores data as snapshots of the entire repository.
- Efficiency: Unchanged files are linked to previous identical files rather than being stored again.

What is 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

What is Git?

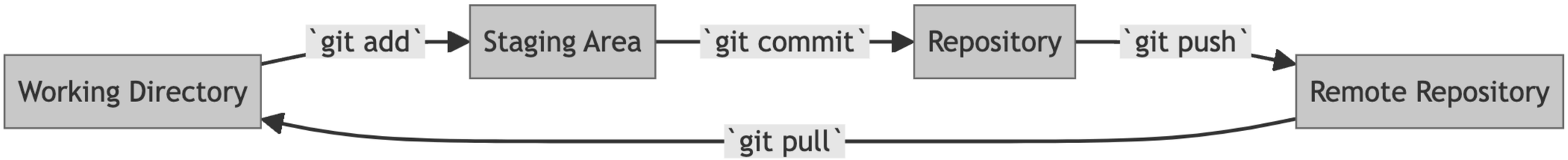
Git Terminology

- **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.

What is Git?

Three Main Stages of Git

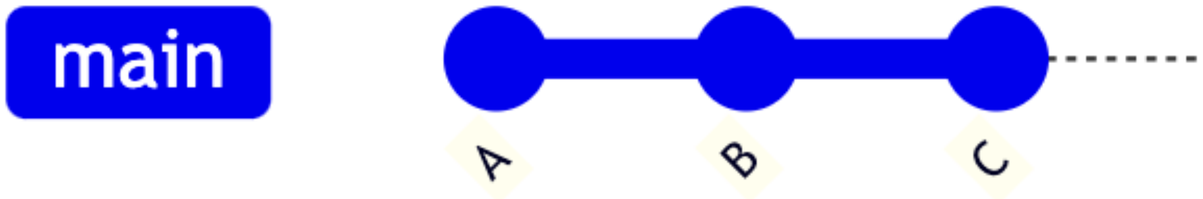
- **Working Directory:** Where you make changes to files.
- **Staging Area:** Format and review changes before committing.
- **Local Repository:** Stores the history of all commits.
- **Remote Repository:** Shared version of the project hosted on a network.



What is Git?

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.



What is Git?

Branching and Merging

- Branching Model: Lightweight movable pointers to snapshots.
- Default Branch: main
- Advantages:
  - Parallel Development: Develop different features simultaneously.
  - Isolation: Work without disturbing the main codebase.
  - Traceability: Easy history tracking of changes.

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

What is GitHub?

- Owned by microsoft
- Offers additional functionality beyond Git.
- GitHub extends Git’s capabilities for enhanced collaboration and project management.
- Features: Wiki, issues, discussions, project management, and automation.

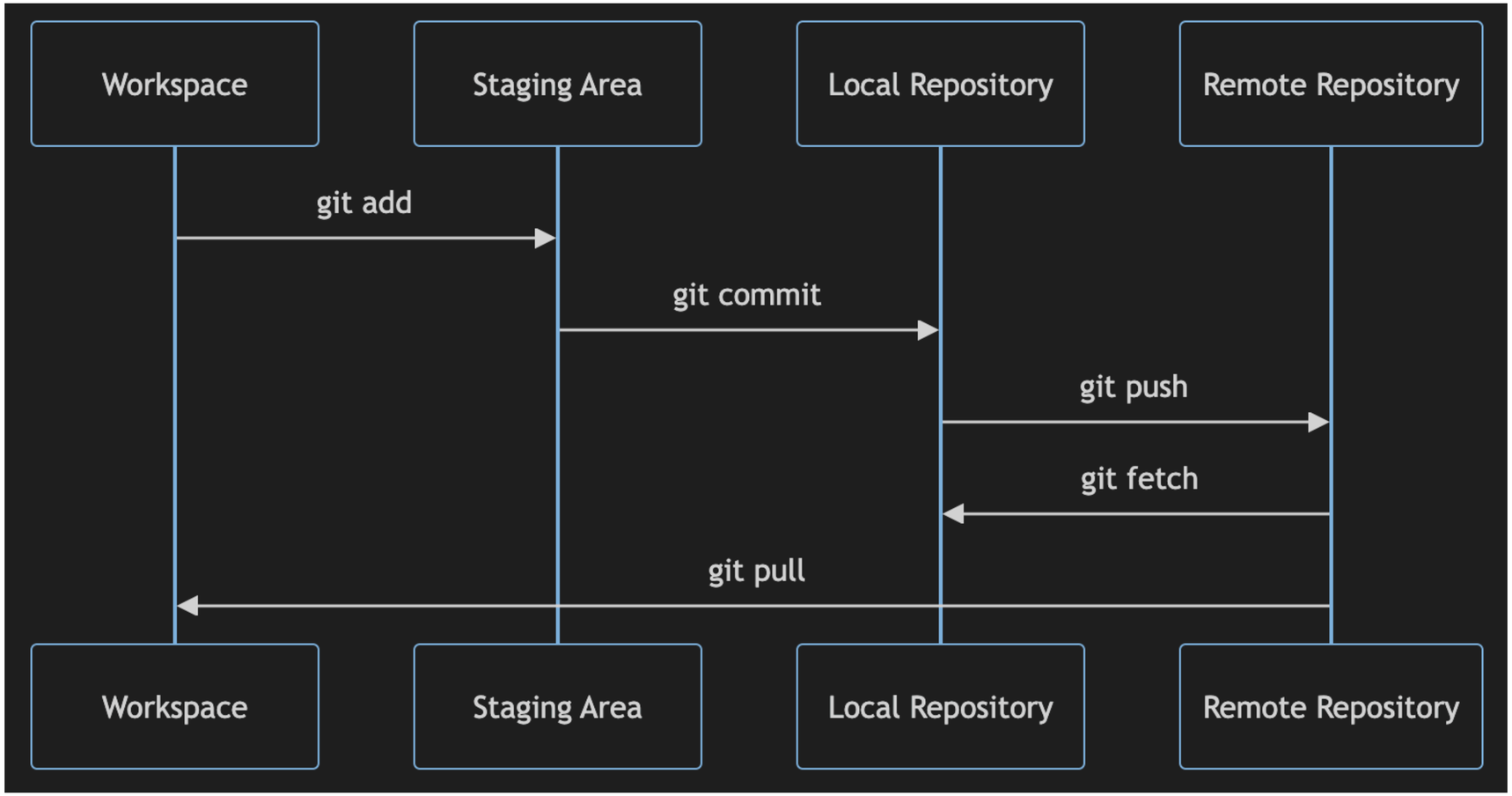
What is GitHub?

Enhanced Collaboration and Project Management

- Project Management:
  - Integrated issue tracking
  - Project boards
  - Milestones
- Collaboration:
  - Pull requests
  - Code reviews
  - Discussion threads
- Documentation:
  - Wikis
  - Markdown support

What is GitHub?

Working with Remotes



Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Basic Git Commands

- Essential Git commands:
  - initializing a git new repository: `git init`
  - checking status: `git status`
  - adding changes: `git add`
  - and committing: `git commit`
- Workflow: modify file -> add -> commit

Basic Git Commands

Project Initialization

- Initialize a Git repository: `git init -b main`
- Command Breakdown:
  - `git init`: Creates a new Git repository.
  - `-b main`: Specifies the initial branch name.
- **Note**: Do not delete the .git directory.

Basic Git Commands

Checking Repository Status

- Check repository status: `git status`
- Command Breakdown:
  - `git status`: Shows the status of changes.
- Example Output:
  - "On branch main“
  - "No commits yet"
  - "nothing to commit"

Basic Git Commands

Adding Changes to the Staging Area

- Stage a file: `git add README.md`
- Command Breakdown:
  - `git add <filename>`: Adds changes to the staging area.
- Example: Staging README.md for the next commit.

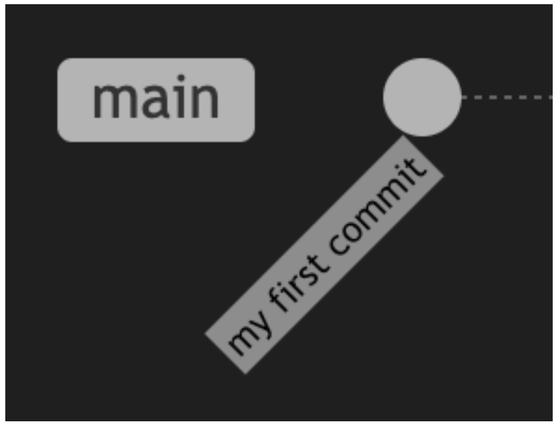
Basic Git Commands

Committing Changes

- Commit staged changes: `git commit -m "Add README file“`
- Command Breakdown:
  - `git commit`: Records changes to the repository.
  - `-m "Commit message"`: Adds a descriptive commit message.
- Example: Creating a commit with the message "Add README file".

Basic Git Commands

Visualizing Commits



Basic Git Commands

Understanding Workspaces

- Workspace Components:
  - **Working Directory**: Where changes are made.
  - **Staging Area**: Prepares changes for commit.
  - **Repository**: Stores project history and commits.
- Workflow Diagram:

Basic Git Commands

Summary of Basic Commands

- `git init -b main`: Initialize a new repository.
- `git status`: Check the status of changes.
- `git add <filename>`: Stage changes.
- `git commit -m "Commit message"`: Commit changes with a message.

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Hands-On: Practice with Basic Git Commands

- Practice creating and working with your first repository.
- Follow step-by-step instructions to apply basic Git commands.

Hands-On: Practice with Basic Git Commands

- Initialize a Repository: Open terminal, navigate to project directory, run `git init -b main`
- Create README.md, check status, add to staging:
  - `echo "# My First Git Project" > README.md`
  - `git status`
  - `git add README.md`
- Commit with a message: `git commit -m "Add README file"`
- Edit README.md, add, and commit: add content,
  - `git add README.md`,
  - `git commit -m "Update README with introduction"`
- Create, edit, add, commit files. Use git log to explore history.

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git History

- Git keeps a detailed history of your repository.
- Enables reviewing and reverting to previous states.
- Essential for tracking changes and collaboration.

Git History

Viewing the History of a Repository

- Command: `git log`
  - Shows commit hash, author, date, message.
- Simplified view:
  - Command: `git log --all --graph --oneline`
- Useful options:
  - `--author=<author>`
  - `--since=<date>`
  - `--until=<date>`
  - `--grep=<pattern>`

Git History

Viewing the Diff of a Commit

- Command: `git diff <commit_hash> <commit_hash>`
  - Example: `git diff 990a517 990a517`
- View diff with commit details:
  - Command: `git show <commit_hash>`

Git History

Viewing the History of a File

- Command: `git log -- <file_path>`
- Example: `git log -- README.md`
- See diffs for file changes:
  - Command: `git log -p -- <file_path>`

Git History

Undoing Changes

- Discard local changes:
  - Command: `git checkout -- <file_path>`
- Undo last commit, keep changes:
  - Command: `git reset --soft HEAD~`
- Undo last commit, discard changes:
  - Command: `git reset --hard HEAD~`
- Revert a specific commit:
  - Command: `git revert <commit_hash>`

Git History

Tagging Commits

- List tags:
  - Command: `git tag --list`
- Create a lightweight tag:
  - Command: `git tag <tag_name>`
- Create an annotated tag:
  - Command: `git tag -a <tag_name> -m "your message"`
- Tag a specific commit:
  - Command: `git tag <tag_name> <commit_hash>`
- View tag details:
  - Command: `git show <tag_name>`

Git History

Navigating the Project History

- Check out a specific commit:
  - Command: `git checkout <commit_hash>`
- Check out a specific tag:
  - Command: `git checkout <tag_name>`

Special Files

.gitignore

- Tells Git which files or directories to ignore.
- Important for keeping unnecessary files out of the repository.
- Examples of Entries:
  - *.log - Ignore all .log files.
  - *.tmp - Ignore all temporary files.
  - node_modules/ - Ignore the node_modules directory.
  - .env - Ignore environment variable files.

Special Files

README.md

- Markdown file providing project overview.
- First file presented to viewers on GitHub.

Special Files

LICENSE

- Specifies terms for using, modifying, and distributing your code.
- Ensures clarity about usage rights.
- Common Licenses:
  - MIT
  - Apache 2.0
  - GPLv3
  - Creative Commons

Special Files

- Special files like .gitignore, README.md, LICENSE, and CHANGELOG.md are crucial for project management.
- Enhance organization, collaboration, and clarity in your projects.
- For more information, refer to:
  - Git documentation (https://www.git-scm.com/doc)
  - GitHub documentation (https://docs.github.com/en)

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Branching and Merging

- Branches allow isolated workspaces in Git
- Enable development of new features, experiments, and collaboration
- Merges integrate changes from branches into the main project

Branching and Merging

Creating Branches

- Start with the main branch and create commits
- Use `git switch --create <new_branch>` to create a new branch
- New branch starts from the current state of the original branch
- Changes in the new branch do not affect the original branch

Branching and Merging

Switching Branches

- Use git switch <branch_name> to change branches
- Always commit or stash changes before switching
- Prevents loss of uncommitted work

Branching and Merging

Merging Branches

- Use git merge <branch_name> to merge changes
- Example: Merge formatting branch into main
- Ensure you are on the target branch before merging

Branching and Merging

Merging Branches - Example

- Command: `git switch main`
- Command: `git merge formatting`
- Explanation:
  - Switch to main branch
  - Merge changes from formatting branch into main

Branching and Merging

Resolving Merge Conflicts

- Merge conflicts occur when the same lines are edited in different branches
- Git markers in conflicted files:
  - `<<<<<< HEAD`
  - `======`
  - `>>>>>> conflicting_branch_name`

Branching and Merging

Resolving Conflicts

- Edit the conflicted file
- Remove conflict markers
- Stage the resolved file: git add <filename>
- Commit the changes: git commit

Branching and Merging

Deleting Branches

- Delete local branch: `git branch --delete <branch_name>`
- Delete remote branch: `git push origin --delete <branch_name>`
- Cleaning up branches keeps the repository organized

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Hands-On: Practice with Branching and Merging

- Initialize repository: `mkdir merge_conflict_demo && cd merge_conflict_demo && git init`
- Create file and initial commit: `echo "This is the first line of the file." > example.txt && git add example.txt && git commit -m "Initial commit with example.txt"`
- Create and switch to new branch: `git switch --create branch1`
- Modify file and commit: `echo "This is a line added in branch1." >> example.txt && git add example.txt && git commit -m "Add a line in branch1"`
- Switch back to main and make changes: `git switch main && echo "This is a line added in main." >> example.txt && git add example.txt && git commit -m "Add a line in main"`
- Merge branch1 into main: `git merge branch1`
- Edit example.txt to resolve conflict markers:
- Stage and commit: `git add example.txt && git commit -m "Resolved merge conflict“`
- Check commit history: `git log --oneline —graph`
- Experiment with more branches, changes, and merges to practice resolving conflicts.

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

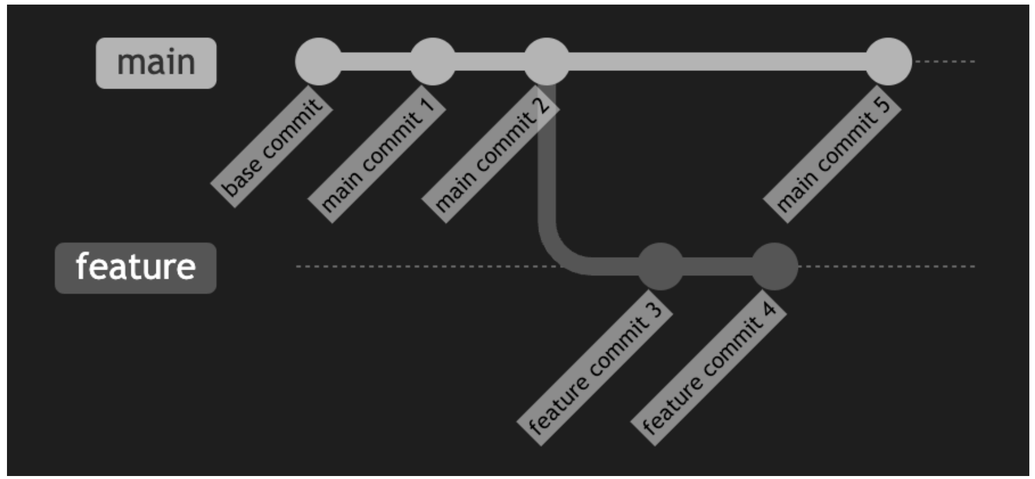
Rebasing and Squashing Commits

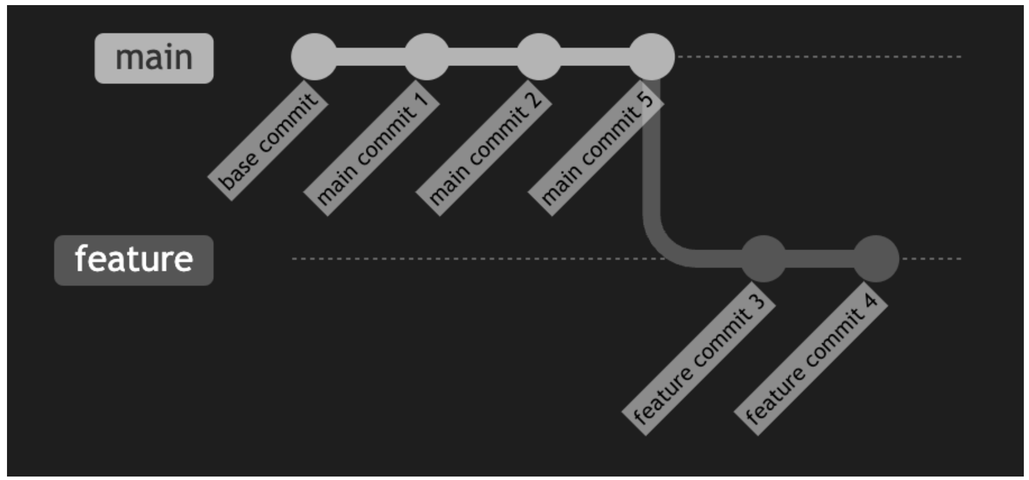
- Rebasing and squashing manage and clean up commit history
- Ensures a clean and understandable repository history

Rebasing and Squashing Commits

Rebasing Commits

- Moves commits to a new base
- Maintains a linear project history
- Commands:
  - `git switch feature_branch`
  - `git rebase main`





Rebasing and Squashing Commits

Interactive Rebasing

- Allows editing, reordering, and squashing commits
- Command:
  - `git rebase -i HEAD~<number_of_commits>`
- Actions:
  - pick, reword, edit, squash, fixup, drop

Rebasing and Squashing Commits

Resolving Conflicts During Rebasing

- Conflicts can occur, resolve manually
- Steps:
  - Resolve conflict in files
  - Stage resolved changes: `git add <resolved_file>`
  - Continue rebase: `git rebase --continue`
  - Abort rebase if needed: `git rebase --abort`

Rebasing and Squashing Commits

Cleaning Up Commit History

- Rebase to clean up history before merging
- Use interactive rebase to:
  - Reorder, squash, edit, or drop commits
- Example:
  - `git rebase -i HEAD~4`
  - Change to pick and squash

Rebasing and Squashing Commits

Squashing Commits

- Combines multiple commits into one
- Useful for grouping small commits
- Interactive rebase command:
  - `git rebase -i HEAD~<number_of_commits>`
  - Change pick to squash for desired commits

Rebasing and Squashing Commits

- Combines all commits into a single commit when merging
- Commands:
  - Switch to main: `git switch main`
  - Squash merge: `git merge --squash feature_branch`
  - Commit: `git commit -m "Merged feature_branch with squash"`

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Collaborating with Git and GitHub

- Collaboration is essential in software development and other domains
- Git and GitHub provide tools to facilitate collaboration

Collaborating with Git and GitHub

Forking Repositories on GitHub

- Fork to create a personal copy of a repository
- Useful for experimenting without affecting the original project
- Ideal for contributing to open-source projects

Collaborating with Git and GitHub

Cloning Repositories

- Clone to create a local copy of a remote repository
- Allows local development and testing
- First step after forking a repository

Collaborating with Git and GitHub

Pushing Changes to a Remote Repository

- Push updates your changes to the remote repository
- Makes your commits available to others
- Essential for sharing work and collaboration

Collaborating with Git and GitHub

Pulling Changes from a Remote Repository

- Pull to update your local repository with remote changes
- Ensures you have the latest code
- Helps prevent conflicts when pushing changes

Collaborating with Git and GitHub

Collaborating with Others on GitHub

- **Issues**: Track bugs and tasks
- **Projects**: Organize work
- **Wikis**: Document the project
- **Discussions**: Engage in conversations
- **Pull Requests**: Propose and review changes

Collaborating with Git and GitHub

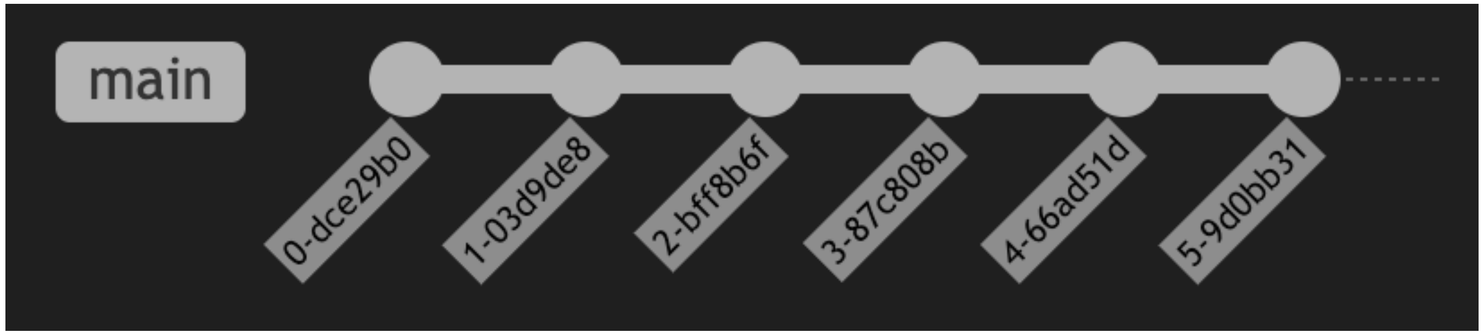
Creating and Managing Pull Requests

- Propose changes to a repository
- Allows for code review and discussions
- Ensures changes meet project standards before merging

Collaborating with Git and GitHub

Collaboration Workflows - Centralized Workflow

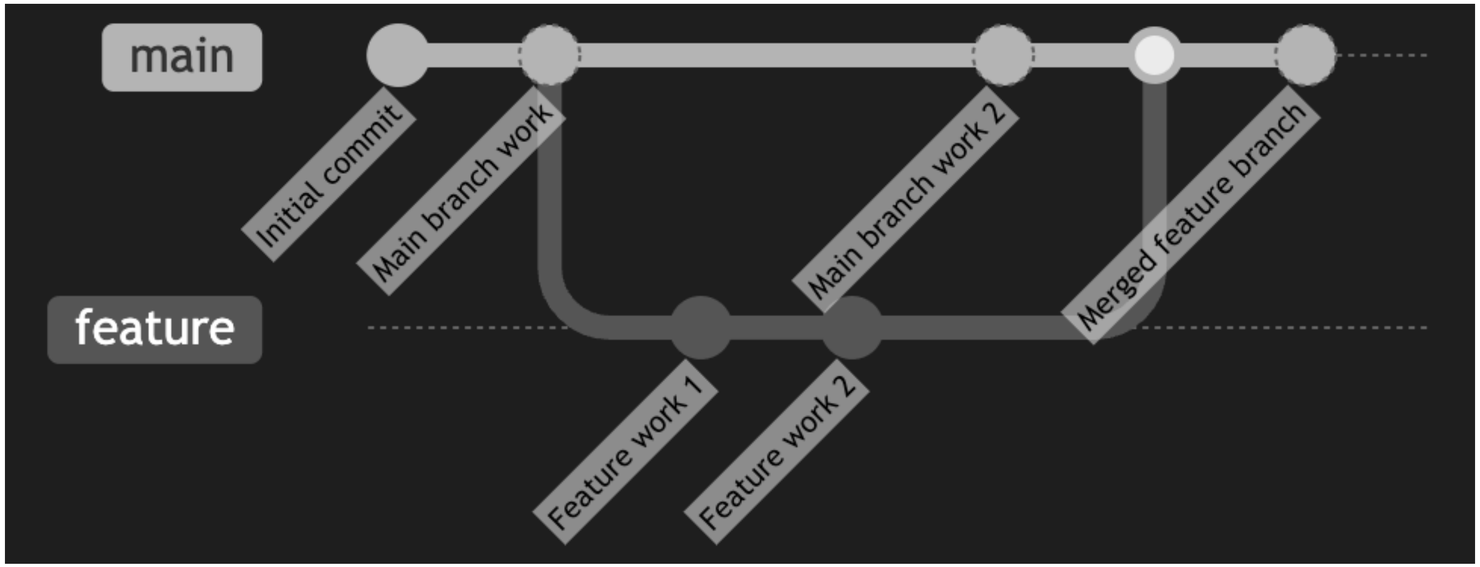
- All changes pushed directly to the main branch
- Suitable for small teams or solo projects



Collaborating with Git and GitHub

Collaboration Workflows - Feature Branch Workflow

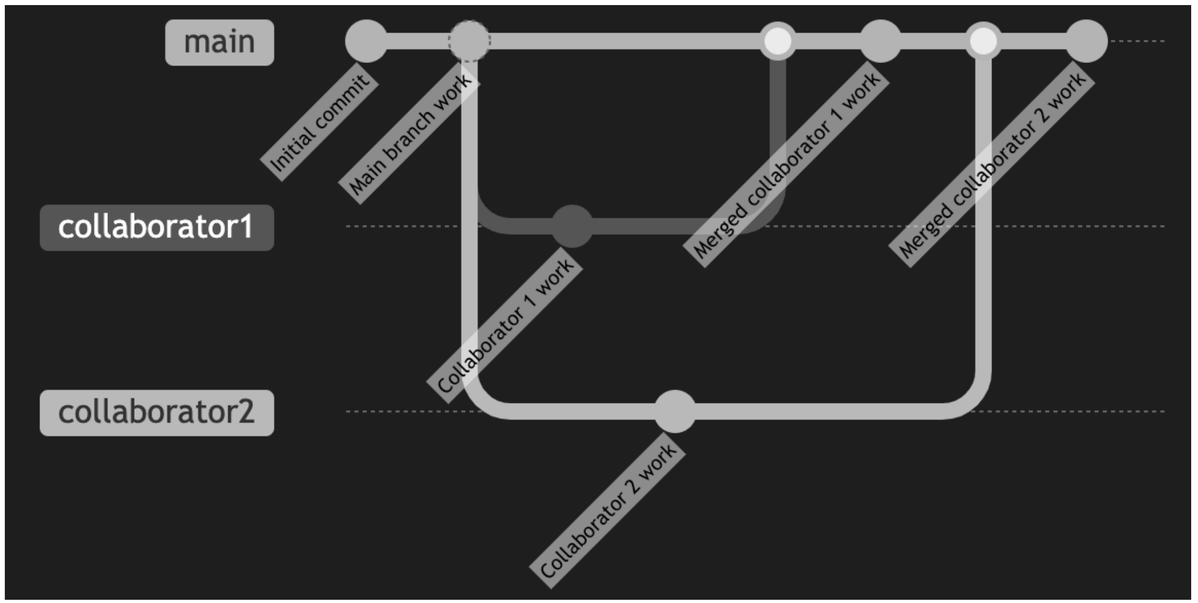
- Each feature developed in its own branch
- Merged into main branch via pull request



Collaborating with Git and GitHub

Collaboration Workflows - Forking Workflow

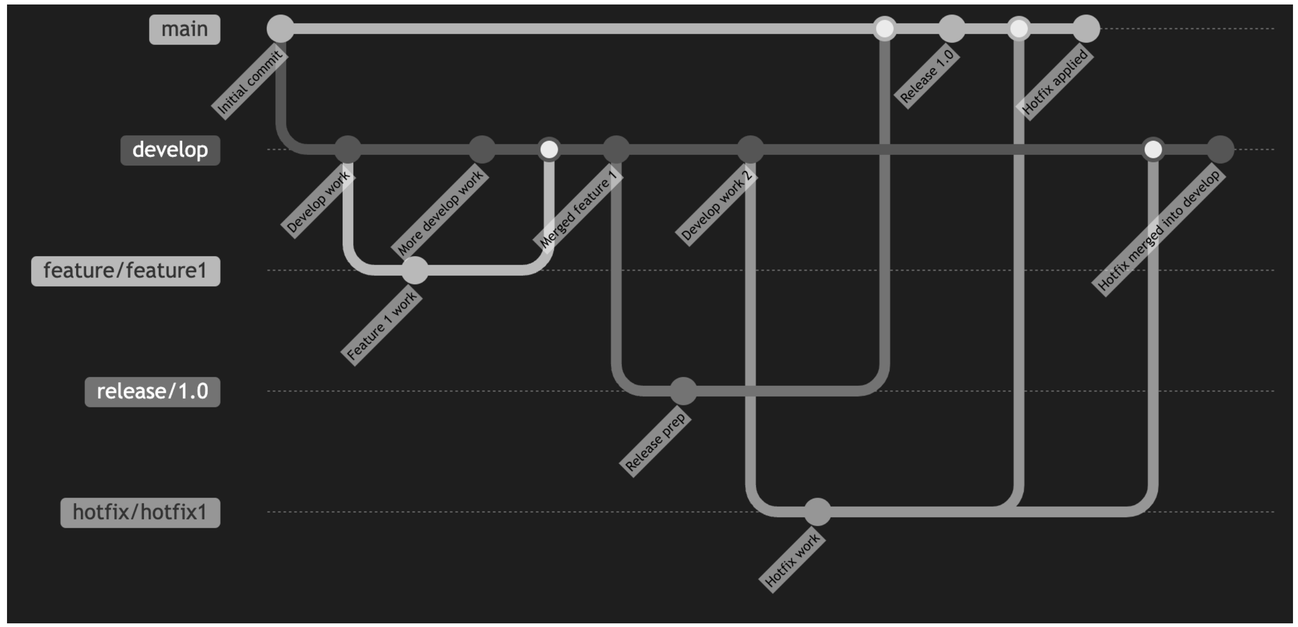
- Each collaborator forks the main repository
- Propose changes via pull requests
- Common in open-source projects



Collaborating with Git and GitHub

Collaboration Workflows - Gitflow Workflow

- Uses multiple branches (main, develop, feature/*, release/*, hotfix/*)
- Suited for larger projects with continuous releases



Collaborating with Git and GitHub

Handling Merge Conflicts

- Conflicts occur when changes conflict between branches
- Resolve conflicts manually using Git's conflict markers
- Continue merge or rebase after resolving conflicts

Collaborating with Git and GitHub

Best Practices for Collaboration

- Write clear commit messages
- Use branches effectively
- Regularly pull changes
- Review changes thoroughly

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Hands-On: Practice with Collaborating with Git and GitHub

- Forking a repository
- Cloning the repository
- Making changes
- Committing changes
- Pushing changes
- Creating a pull request
- Engaging in discussion
- Reviewing and merging

Hands-On: Practice with Collaborating with Git and GitHub

Fork a Repository

- Go to DataTrain-2024-handson (https://github.com/nharms-awi/DataTrain-2024-handson)
- Click the Fork button
- Choose your GitHub account

Hands-On: Practice with Collaborating with Git and GitHub

Clone the Repository

- Open terminal
- Navigate to your repository directory
- Run: `git clone https://github.com/<your-username>/DataTrain-2024-handson.git`
- Navigate to directory: cd DataTrain-2024-handson

Hands-On: Practice with Collaborating with Git and GitHub

Make Changes

- Open the repository in your code editor
- Example change: Add a new file, e.g. your initials and .md (nh.md)
- Example content:

```
## I am NH

This is my new file.
```

Hands-On: Practice with Collaborating with Git and GitHub

Commit Your Changes

- Stage and commit changes:
  - Stage: `git add <your_file>`
- Commit: `git commit -m "add new file <file_name>“`

Hands-On: Practice with Collaborating with Git and GitHub

Push Changes to GitHub

- Push to main branch: `git push origin main`

Hands-On: Practice with Collaborating with Git and GitHub

Create a Pull Request

- Go to your forked repository on GitHub
- Click Compare & pull request
- Provide a description
- Click Create pull request

Hands-On: Practice with Collaborating with Git and GitHub

Review and Merge

Hands-On: Practice with Collaborating with Git and GitHub

Go Wild!

- Create and switch branches
- Merge pull requests
- Resolve conflicts
- Explore workflows
- Create issues for questions or suggestions

Hands-On: Practice with Collaborating with Git and GitHub

Review and Merge

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Git/GitHub - DataTrain 2024

Schedule

Basics

- Introduction
- Checking Setup

- Introduction to the CLI
- Hands-On: CLI

- Why Version Control?
- What is Git?
- What is GitHub?

- Basic Git Commands

Lunch Break

Advanced Topics

- Hands-On: Basic Git Commands
- Git History and Special Files

- Branching and Merging
- Hands-On: Branching and Merging

- Rebasing and Squashing
- Collaboration with Git and Github
- Hands-On: Collaboration

- Q&A and Wrap-Up

Q&A

Wrap-Up

- Introduction to CLI
- Version Control Systems
  - Importance of tracking changes, collaborating, and maintaining project history
  - Git as a distributed VCS for efficient branching, merging, and collaboration
  - Introduction to GitHub for hosting repositories, code reviews, bug tracking, project management, and discussions
- Fundamental Git Commands
- Collaboration with Git and GitHub