Version Control for Researchers

Richard Polzin (4.11.2025)

✨Version Control✨

✨Version Control✨

What?

✨Version Control✨

What?

- A system for managing changes to files over time

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??
- Tracking changes and ensuring reproducibility

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??
- Tracking changes and ensuring reproducibility
- Avoiding "final_version_v3_revised_FINAL.py" 😵

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??
- Tracking changes and ensuring reproducibility
- Avoiding "final_version_v3_revised_FINAL.py" 😵
- Backup and restoring previous versions

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??
- Tracking changes and ensuring reproducibility
- Avoiding "final_version_v3_revised_FINAL.py" 😵
- Backup and restoring previous versions

How?

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??
- Tracking changes and ensuring reproducibility
- Avoiding "final_version_v3_revised_FINAL.py" 😵
- Backup and restoring previous versions

How?

Git of course!

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??
- Tracking changes and ensuring reproducibility
- Avoiding "final_version_v3_revised_FINAL.py" 😵
- Backup and restoring previous versions

How?

Git of course!

- A distributed version control system (VCS)

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??
- Tracking changes and ensuring reproducibility
- Avoiding "final_version_v3_revised_FINAL.py" 😵
- Backup and restoring previous versions

How?

Git of course!

- A distributed version control system (VCS)
- Tracks changes in code and text files

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??
- Tracking changes and ensuring reproducibility
- Avoiding "final_version_v3_revised_FINAL.py" 😵
- Backup and restoring previous versions

How?

Git of course!

- A distributed version control system (VCS)
- Tracks changes in code and text files
- Enables collaboration across different versions

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??
- Tracking changes and ensuring reproducibility
- Avoiding "final_version_v3_revised_FINAL.py" 😬
- Backup and restoring previous versions

How?

Git of course!

- A distributed version control system (VCS)
- Tracks changes in code and text files
- Enables collaboration across different versions
- Supports parallel development

✨Version Control✨

What?

- A system for managing changes to files over time
- Allows simultaneous work on the same project
- A history of changes and the ability to revert
- Logically separate features 🤯

Why?

- Makes collaboration easier 🏅
- Who deleted my files? Where is my main.py??
- Tracking changes and ensuring reproducibility
- Avoiding "final_version_v3_revised_FINAL.py" 😬
- Backup and restoring previous versions

How?

Git of course!

- A distributed version control system (VCS)
- Tracks changes in code and text files
- Enables collaboration across different versions
- Supports parallel development
- Provides a detailed history of changes for accountability

Git Workshop

Key Concepts

Git Workshop

Key Concepts

- **Repository (Repo):** A directory containing all project files and history

Git Workshop

Key Concepts

- **Repository (Repo):** A directory containing all project files and history

Git Workshop

Key Concepts

- **Repository (Repo):** A directory containing all project files and history
- **Commit:** A snapshot of changes

Git Workshop

Key Concepts

- **Repository (Repo):** A directory containing all project files and history
- **Commit:** A snapshot of changes
- **Branch:** Parallel versions of the repository

Git Workshop

Key Concepts

- **Repository (Repo):** A directory containing all project files and history
- **Commit:** A snapshot of changes
- **Branch:** Parallel versions of the repository
- **Merge:** Combining different branches

Git Workshop

Key Concepts

- **Repository (Repo):** A directory containing all project files and history
- **Commit:** A snapshot of changes
- **Branch:** Parallel versions of the repository
- **Merge:** Combining different branches
- **Remote:** A repository hosted elsewhere (e.g., GitHub, GitLab)

Initial Setup

Initial Setup

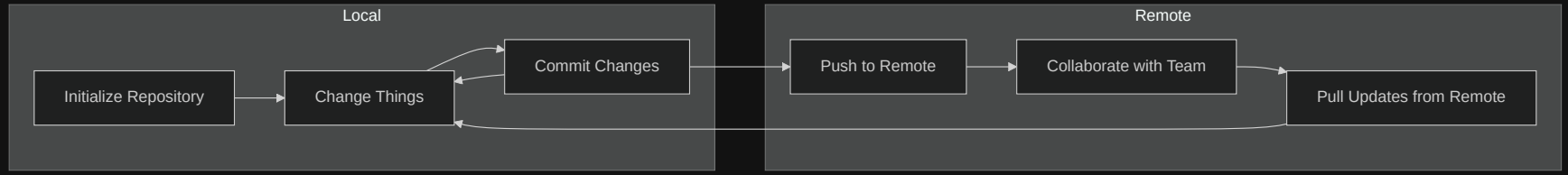
```
# Install Git
sudo apt install git  # Linux
brew install git  # macOS
choco install git.install  # Windows

# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Initial Setup

```
# Install Git
sudo apt install git  # Linux
brew install git  # macOS
choco install git.install  # Windows

# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

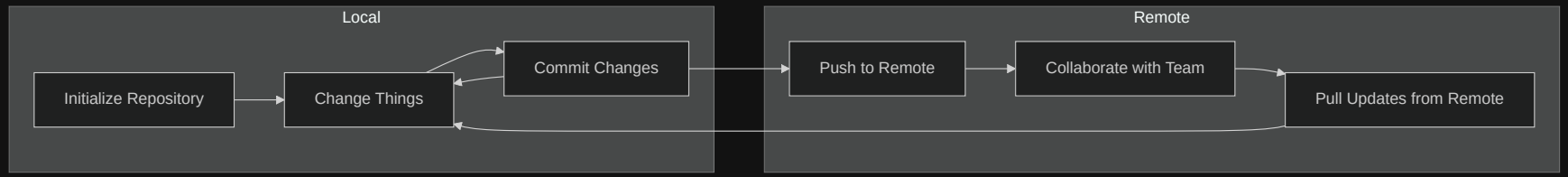
Initial Setup

```
# Install Git
sudo apt install git  # Linux
brew install git  # macOS
choco install git.install  # Windows

# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```



Initial Setup

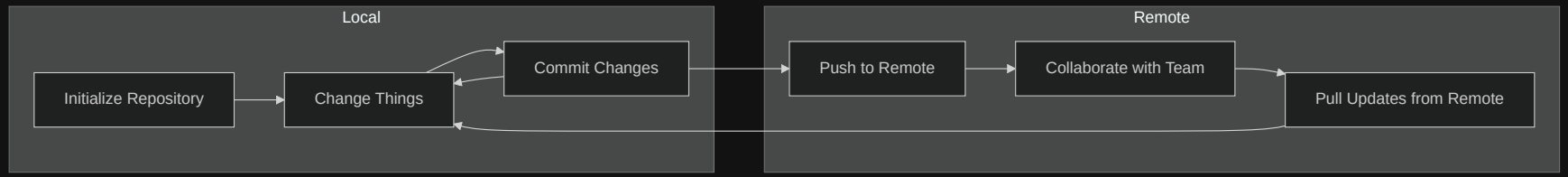
```
# Install Git
sudo apt install git  # Linux
brew install git  # macOS
choco install git.install  # Windows

# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```



- **Initialize Repository:** Start a new repository with `git init`.

Initial Setup

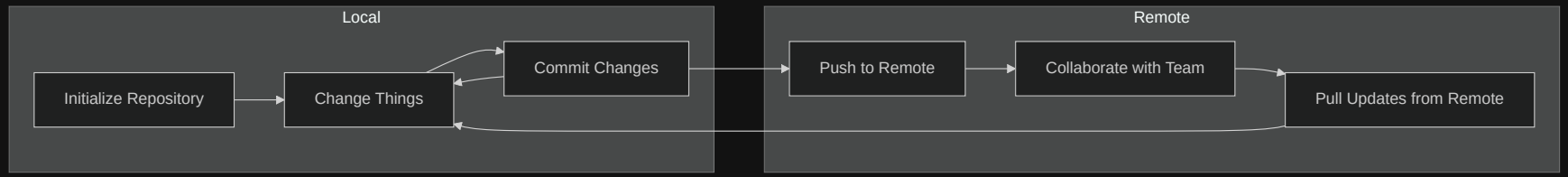
```
# Install Git
sudo apt install git  # Linux
brew install git  # macOS
choco install git.install  # Windows

# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```



- **Initialize Repository:** Start a new repository with `git init`.
- **Make Changes:** Modify files in your working directory.

Initial Setup

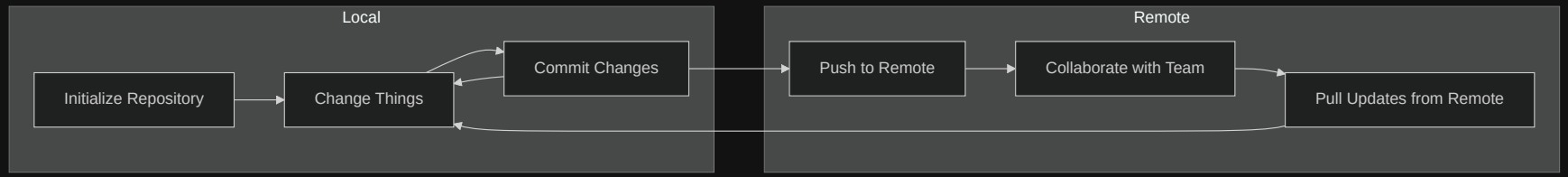
```
# Install Git
sudo apt install git  # Linux
brew install git  # macOS
choco install git.install  # Windows

# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```



- **Initialize Repository:** Start a new repository with `git init`.
- **Make Changes:** Modify files in your working directory.
- **Commit Changes:** Save snapshots of your changes with `git commit`.

Initial Setup

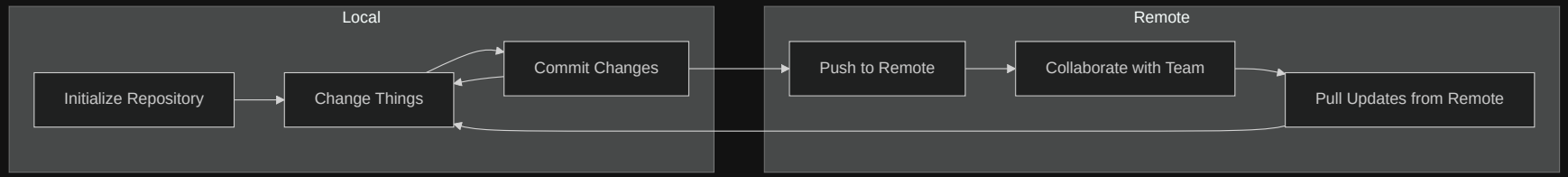
```
# Install Git
sudo apt install git  # Linux
brew install git  # macOS
choco install git.install  # Windows

# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```



- **Initialize Repository:** Start a new repository with `git init`.
- **Make Changes:** Modify files in your working directory.
- **Commit Changes:** Save snapshots of your changes with `git commit`.
- **Push to Remote:** Upload your commits to a remote repository with `git push`.

Initial Setup

```
# Install Git
sudo apt install git  # Linux
brew install git  # macOS
choco install git.install  # Windows

# Configure Git
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```



- **Initialize Repository:** Start a new repository with `git init`.
- **Make Changes:** Modify files in your working directory.
- **Commit Changes:** Save snapshots of your changes with `git commit`.
- **Push to Remote:** Upload your commits to a remote repository with `git push`.
- **Pull Updates:** Fetch and integrate changes from the remote repository with `git pull`.

Example

```
$ # Initialize a new Git repository
$ git init my_project # Create the directory and a .git folder in it
$ cd my_project
$ # Create a file and commit it
$ echo "# My Research Project" > README.md
$ git add README.md
$ git commit -m "Initial commit"
> Initial commit
> 1 file changed, 1 insertion(+)
> create mode 100644 README.md
$ # Check the status
$ git status
> On branch main
> nothing to commit, working tree clean
$ # Change a file
$ echo "\nthis is my cool description." >> README.md
```

Example

```
$ # Initialize a new Git repository
$ git init my_project # Create the directory and a .git folder in it
$ cd my_project
$ # Create a file and commit it
$ echo "# My Research Project" > README.md
$ git add README.md
$ git commit -m "Initial commit"
> Initial commit
> 1 file changed, 1 insertion(+)
> create mode 100644 README.md
$ # Check the status
$ git status
> On branch main
> nothing to commit, working tree clean
$ # Change a file
$ echo "\nthis is my cool description." >> README.md
```

Example

```
$ # Initialize a new Git repository
$ git init my_project # Create the directory and a .git folder in it
$ cd my_project
$ # Create a file and commit it
$ echo "# My Research Project" > README.md
$ git add README.md
$ git commit -m "Initial commit"
> Initial commit
> 1 file changed, 1 insertion(+)
> create mode 100644 README.md
$ # Check the status
$ git status
> On branch main
> nothing to commit, working tree clean
$ # Change a file
$ echo "\nthis is my cool description." >> README.md
```

Example

```
$ # Initialize a new Git repository
$ git init my_project # Create the directory and a .git folder in it
$ cd my_project
$ # Create a file and commit it
$ echo "# My Research Project" > README.md
$ git add README.md
$ git commit -m "Initial commit"
> Initial commit
> 1 file changed, 1 insertion(+)
> create mode 100644 README.md
$ # Check the status
$ git status
> On branch main
> nothing to commit, working tree clean
$ # Change a file
$ echo "\nthis is my cool description." >> README.md
$ # Check the status
```

Example

```
$ # Create a file and commit it
$ echo "# My Research Project" > README.md
$ git add README.md
$ git commit -m "Initial commit"
> Initial commit
> 1 file changed, 1 insertion(+)
> create mode 100644 README.md
$ # Check the status
$ git status
> On branch main
> nothing to commit, working tree clean
$ # Change a file
$ echo "\nthis is my cool description." >> README.md
$ # Check the status
$ git status
> On branch master
> Changes not staged for commit:
```

Example

```
$ echo "# My Research Project" > README.md
$ git add README.md
$ git commit -m "Initial commit"
> Initial commit
> 1 file changed, 1 insertion(+)
> create mode 100644 README.md
$ # Check the status
$ git status
> On branch main
> nothing to commit, working tree clean
$ # Change a file
$ echo "\nthis is my cool description." >> README.md
$ # Check the status
$ git status
> On branch master
> Changes not staged for commit:
>   (use "git add <file>..." to update what will be committed)
```

Example

```
> Initial commit
> 1 file changed, 1 insertion(+)
> create mode 100644 README.md
$ # Check the status
$ git status
> On branch main
> nothing to commit, working tree clean
$ # Change a file
$ echo "\nthis is my cool description." >> README.md
$ # Check the status
$ git status
> On branch master
> 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:   README.md
>
```

Example

```
> create mode 100644 README.md
$ # Check the status
$ git status
> On branch main
> nothing to commit, working tree clean
$ # Change a file
$ echo "\nthis is my cool description." >> README.md
$ # Check the status
$ git status
> On branch master
> 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:   README.md
>
> no changes added to commit (use "git add" and/or "git commit -a")
$ # Check the differences
```

Example

```
> On branch main
> nothing to commit, working tree clean
$ # Change a file
$ echo "\nthis is my cool description." >> README.md
$ # Check the status
$ git status
> On branch master
> 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:   README.md
>
> no changes added to commit (use "git add" and/or "git commit -a")
$ # Check the differences
$ git diff
> diff --git a/README.md b/README.md
> index 22c86a3..0628ec3 100644
```

Example

```
> On branch master
> 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:   README.md
>
> no changes added to commit (use "git add" and/or "git commit -a")
$ # Check the differences
$ git diff
> diff --git a/README.md b/README.md
> index 22c86a3..0628ec3 100644
> --- a/README.md
> +++ b/README.md
> @@ -1 +1,3 @@
>  # My Research Project
> +
> +this is my cool description.
```

Example

```
>   modified:   README.md
>
> no changes added to commit (use "git add" and/or "git commit -a")
$ # Check the differences
$ git diff
> diff --git a/README.md b/README.md
> index 22c86a3..0628ec3 100644
> --- a/README.md
> +++ b/README.md
> @@ -1 +1,3 @@
>  # My Research Project
> +
> +this is my cool description.
$ # Check the commit history
$ git log
> commit 0127a4e6b03cec81c38391dc643f50fdfee75f4b (HEAD -> main)
> Author: Richard Polzin <richard.polzin@posteo.de>
```

Example

```
$ git diff
> diff --git a/README.md b/README.md
> index 22c86a3..0628ec3 100644
> --- a/README.md
> +++ b/README.md
> @@ -1 +1,3 @@
>  # My Research Project
> +
> +this is my cool description.
$ # Check the commit history
$ git log
> commit 0127a4e6b03cec81c38391dc643f50fdfee75f4b (HEAD -> main)
> Author: Richard Polzin <richard.polzin@posteo.de>
> Date:   Mon Feb 3 13:37:57 2025 +0100
>
>     Initial commit
```

Example

```
$ git diff
> diff --git a/README.md b/README.md
> index 22c86a3..0628ec3 100644
> --- a/README.md
> +++ b/README.md
> @@ -1 +1,3 @@
>  # My Research Project
> +
> +this is my cool description.
$ # Check the commit history
$ git log
> commit 0127a4e6b03cec81c38391dc643f50fdfee75f4b (HEAD -> main)
> Author: Richard Polzin <richard.polzin@posteo.de>
> Date:   Mon Feb 3 13:37:57 2025 +0100
>
>     Initial commit
```

Recap

Recap

- **Version Control:** Manage changes to files over time, enable collaboration and track history.

Recap

- **Version Control:** Manage changes to files over time, enable collaboration and track history.
- **Git:** The (coolest 😉) software to do version control with.

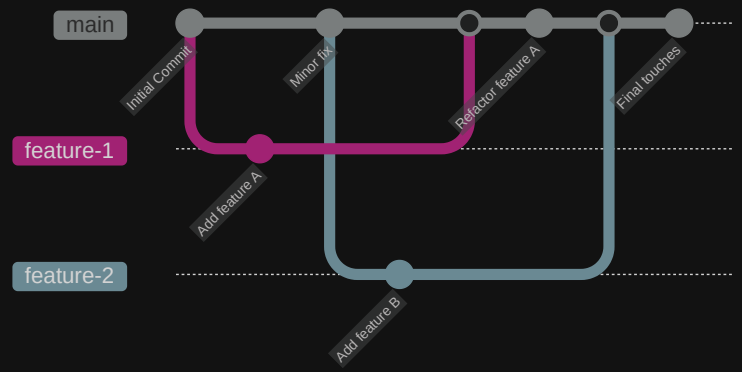
Recap

- **Version Control:** Manage changes to files over time, enable collaboration and track history.
- **Git:** The (coolest 😉) software to do version control with.
- **Key Concepts:** Repository, Commit, Branch, Merge, Remote.

Recap

- **Version Control:** Manage changes to files over time, enable collaboration and track history.
- **Git:** The (coolest 😉) software to do version control with.
- **Key Concepts:** Repository, Commit, Branch, Merge, Remote.
- **Basic Commands:** `git init`, `git add`, `git commit`, `git status`, `git log`, `git diff`.

Branching and Merging



Branching and Merging

Branching and Merging

- **Isolation:** Work on different features or fixes separately from the main codebase.

Branching and Merging

- **Isolation:** Work on different features or fixes separately from the main codebase.
- **Parallel Development:** Enable team members to work on separate features simultaneously.

Branching and Merging

- **Isolation:** Work on different features or fixes separately from the main codebase.
- **Parallel Development:** Enable team members to work on separate features simultaneously.
- **History Tracking:** Maintain individual commit histories for easy tracking and reversion.

Branching and Merging

- **Isolation:** Work on different features or fixes separately from the main codebase.
- **Parallel Development:** Enable team members to work on separate features simultaneously.
- **History Tracking:** Maintain individual commit histories for easy tracking and reversion.
- **Merging:** Combine changes from different branches back into the main codebase.

Branching and Merging

- **Isolation:** Work on different features or fixes separately from the main codebase.
- **Parallel Development:** Enable team members to work on separate features simultaneously.
- **History Tracking:** Maintain individual commit histories for easy tracking and reversion.
- **Merging:** Combine changes from different branches back into the main codebase.
- **Experimentation:** Safely test new ideas without affecting the stable codebase.

Branching and Merging

- **Isolation:** Work on different features or fixes separately from the main codebase.
- **Parallel Development:** Enable team members to work on separate features simultaneously.
- **History Tracking:** Maintain individual commit histories for easy tracking and reversion.
- **Merging:** Combine changes from different branches back into the main codebase.
- **Experimentation:** Safely test new ideas without affecting the stable codebase.

```
# Create and switch to a new branch
git checkout -b new_feature

# Merge changes back to main branch
git checkout main
git merge new_feature
```

Branching and Merging

- **Isolation:** Work on different features or fixes separately from the main codebase.
- **Parallel Development:** Enable team members to work on separate features simultaneously.
- **History Tracking:** Maintain individual commit histories for easy tracking and reversion.
- **Merging:** Combine changes from different branches back into the main codebase.
- **Experimentation:** Safely test new ideas without affecting the stable codebase.

```
# Create and switch to a new branch
git checkout -b new_feature

# Merge changes back to main branch
git checkout main
git merge new_feature
```

Branching and Merging

- **Isolation:** Work on different features or fixes separately from the main codebase.
- **Parallel Development:** Enable team members to work on separate features simultaneously.
- **History Tracking:** Maintain individual commit histories for easy tracking and reversion.
- **Merging:** Combine changes from different branches back into the main codebase.
- **Experimentation:** Safely test new ideas without affecting the stable codebase.

```
# Create and switch to a new branch
git checkout -b new_feature

# Merge changes back to main branch
git checkout main
git merge new_feature
```

Branching and Merging

- **Isolation:** Work on different features or fixes separately from the main codebase.
- **Parallel Development:** Enable team members to work on separate features simultaneously.
- **History Tracking:** Maintain individual commit histories for easy tracking and reversion.
- **Merging:** Combine changes from different branches back into the main codebase.
- **Experimentation:** Safely test new ideas without affecting the stable codebase.

```
# Create and switch to a new branch
git checkout -b new_feature

# Merge changes back to main branch
git checkout main
git merge new_feature
```

Collaborating with Remotes

Collaborating with Remotes

- **Remote Repository:** A version of your project hosted on the internet.

Collaborating with Remotes

- **Remote Repository:** A version of your project hosted on the internet.
- **Push/Pull:** Upload/Download changes to and from remote.

Collaborating with Remotes

- **Remote Repository:** A version of your project hosted on the internet.
- **Push/Pull:** Upload/Download changes to and from remote.
- **Origin:** The name of the remote repository.

Collaborating with Remotes

- **Remote Repository:** A version of your project hosted on the internet.
- **Push/Pull:** Upload/Download changes to and from remote.
- **Origin:** The name of the remote repository.
- **Clone:** Create a local copy of a remote repository.

Best Practices (for Researchers)

Best Practices (for Researchers)

- 📝 Use meaningful commit messages

Best Practices (for Researchers)

- 📝 Use meaningful commit messages
- 📁 Keep repositories organized

Best Practices (for Researchers)

- 📝 Use meaningful commit messages
- 📁 Keep repositories organized
- 🚫 Use `.gitignore` for large files and temporary files

Best Practices (for Researchers)

- 📝 Use meaningful commit messages
- 📁 Keep repositories organized
- 🚫 Use `.gitignore` for large files and temporary files
- 🔄 Regularly push to a remote repository

Best Practices (for Researchers)

- 📝 Use meaningful commit messages
- 📁 Keep repositories organized
- 🚫 Use `.gitignore` for large files and temporary files
- 🔄 Regularly push to a remote repository
- 🌿 Use branches for experimental changes

GitHub, GitLab, and Alternatives

- **GitHub:** Popular for open-source projects, big publicity/community
- **GitLab:** Itself Open-Source, available from RWTH-Aachen for Education only
- **Gitlab-CE:** Same as above, but more free license, less features

Advanced Topics (t.b.d.)

- Using Git with Jupyter Notebooks
- Git Large File Storage (LFS) for datasets
- Continuous Integration (CI) for automating workflows
- Working with submodules for modular projects

Summary

- Version Control helps manage research projects efficiently
- Enables collaboration and reproducibility
- Learning basic Git commands is a valuable skill
- Use hosted platforms for better sharing and tracking
- Start early, commit often, write good commit messages