Antony Kervazo-Canut

Git for Teenagers

SUITABLE FOR ADULTS

Introduction

Git is a free and open-source decentralized version control system designed to manage any project, from the smallest to the largest, with speed and efficiency. It allows multiple developers to work together on the same project, effectively managing the different versions of each file and ensuring that changes do not conflict with one another.

Git was created by Linus Torvalds, the same Finnish computer scientist who initiated the development of the Linux kernel. In 2005, faced with the version management needs for the development of the Linux kernel, Torvalds designed Git to improve productivity and collaboration among developers.

The name "Git" is a play on words by Linus Torvalds. According to him, it's a term of affection, contempt, or both, depending on the context, for himself. "Git" is a British slang term meaning "unpleasant person." Torvalds chose this name because he liked to think that "I am an unpleasant person and I name my projects after myself." On the other hand, "git" can also be considered an acronym for "global information tracker," reflecting its primary function as a version control system.

How it works

The strength of Git lies in its ability to save files in a folder (the repository) and to manage the files with regular backups named "commits."

The local repository then manages a history of modifications and can send the changes ("push") to a remote repository to share the local modifications with others or simply for backup purposes.

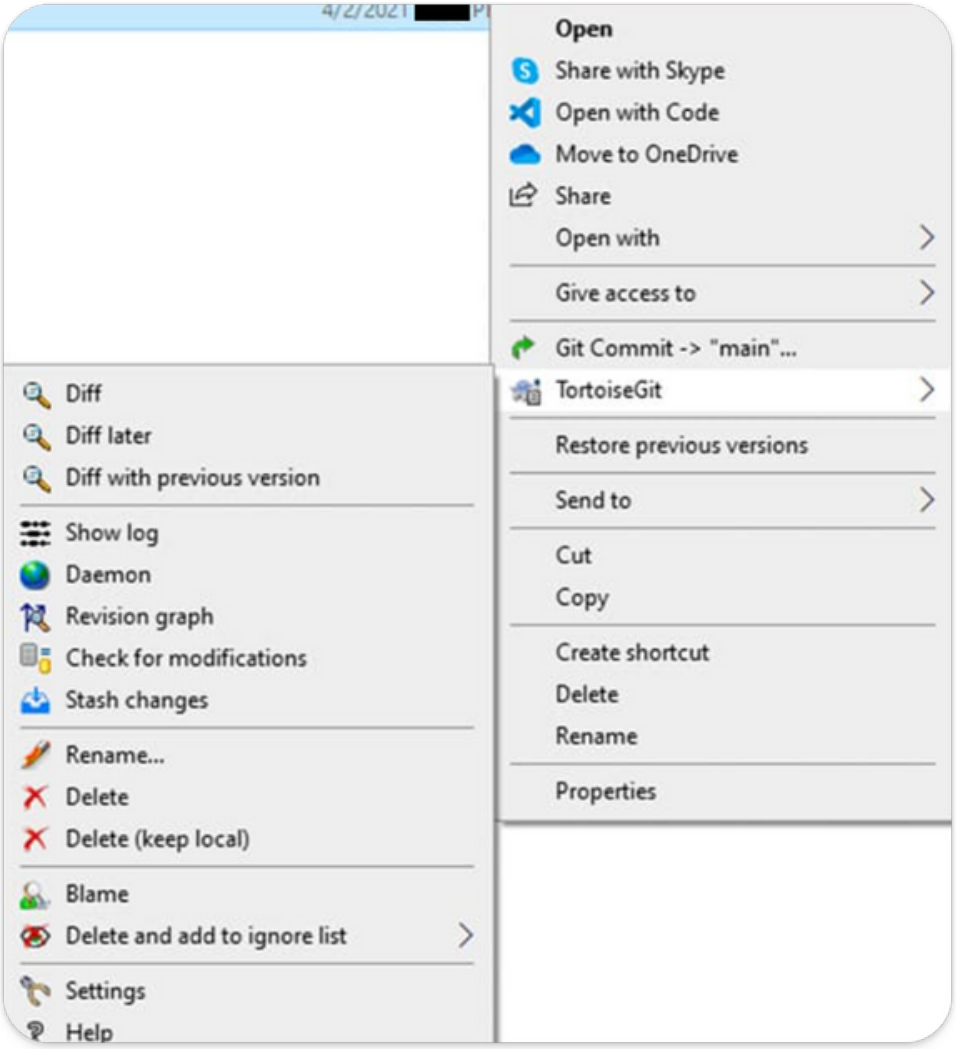
Git also incorporates a branching system. A "branch" then becomes an alternative version of the folder and files on which it is possible to work without impacting the main work.

At the end of a task and upon satisfaction, it is possible to "merge" a working branch onto the main branch to combine all the changes.

Other users will then be able to retrieve the updated files.

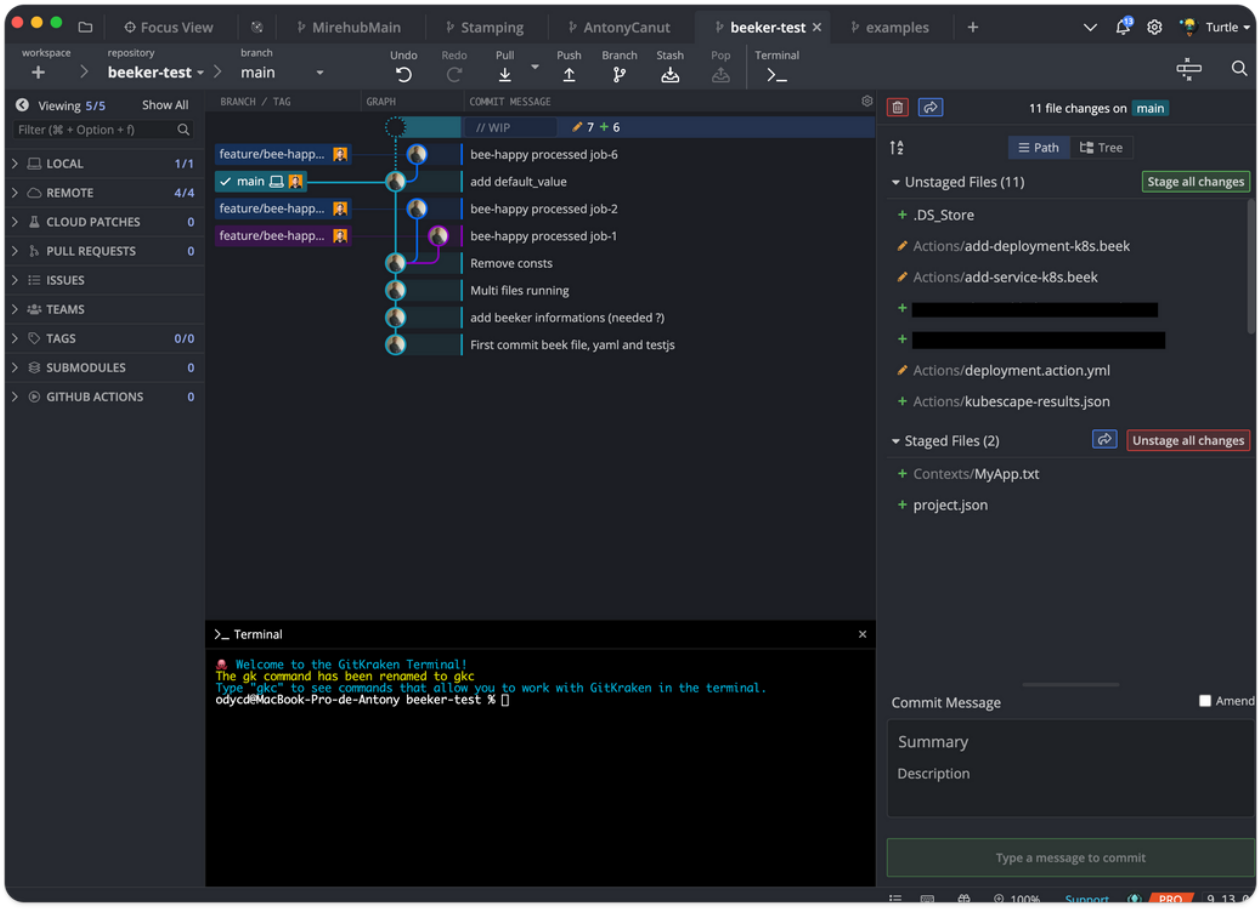
Third-Party Applications (TortoiseGit)

TortoiseGit is a Windows interface for Git that is not tied to any specific IDE, allowing you to use it with the development tools of your choice. It offers easy explorer integration with Git commands available directly from the context menu. TortoiseGit stands out for its ease of use, with descriptive dialogs and a visual indication of the status of files directly in Windows Explorer. It also offers integration with issue tracking systems and various useful tools like TortoiseGitMerge for merging and conflict resolution.



Third-Party Applications (GitKraken)

For those who prefer a graphical user interface (GUI) over the command line, GitKraken is a popular third-party application that simplifies the management of Git repositories. GitKraken provides a clear visualization of branch history, simplifies the merge and rebase processes, and incorporates numerous features to enhance productivity and collaboration.



It easily connects to online repository platforms such as GitHub, GitLab, and Bitbucket, facilitating the management of projects hosted on these services. With features like pull requests and diff previews, GitKraken helps teams collaborate more effectively on their projects.

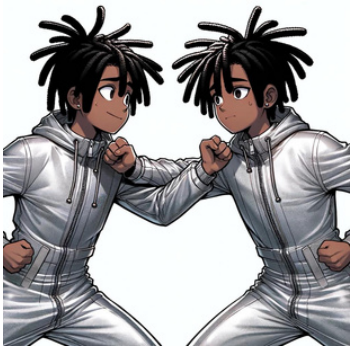
GitKraken is free for individual developers on open source repositories.

Git Configuration

The first thing to do is to configure your username and email address. This information will be attached to your commits and tags. To do this, open a terminal and enter the following commands (different for using a GUI):

```
# Set your username
git config --global user.name "Your Name"
# This command sets the username that will be used for all
your commits.

# Set your email address
git config --global user.email "your.email@example.com"
# This command sets the email address that will be associated
with your commits.

# Configure the default text editor
git config --global core.editor "editor_name"
# Replace "editor_name" with the command name of your editor.
# Examples: nano, vim, code (for VS Code).
```

Init & Clone



To start using Git on a new project, you first need to initialize a repository. This creates a new subfolder named .git that contains all the files necessary for the version tracking system. To initialize a repository, navigate to your project folder from your terminal and type the init command.

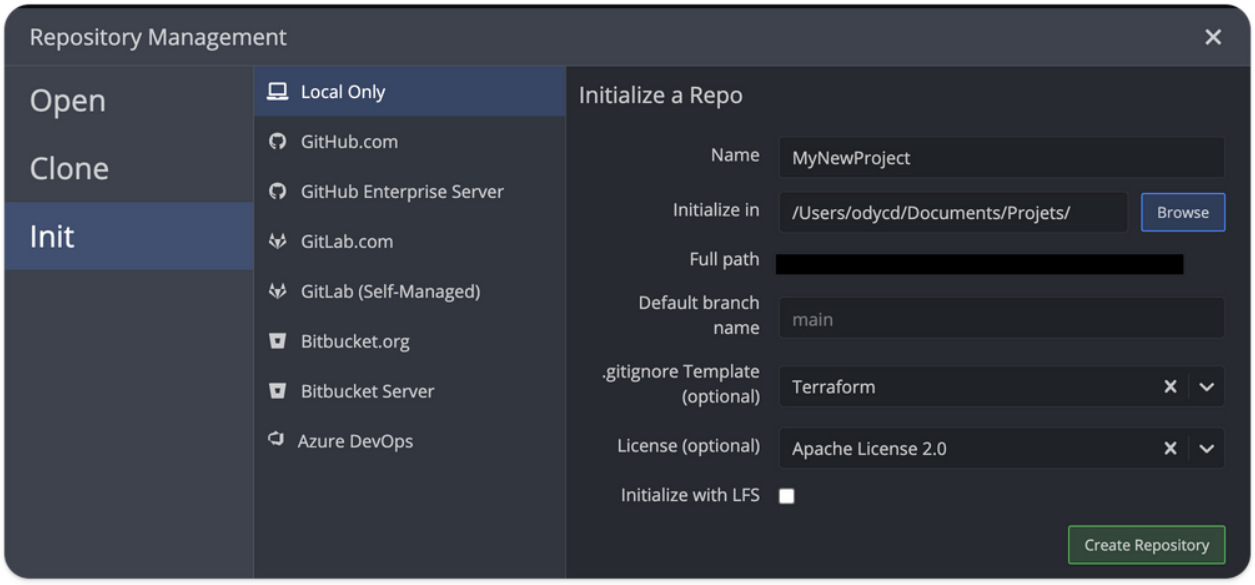
If you want to work on an existing project that is already under version tracking with Git, you can clone this repository to your local machine. Cloning a repository not only copies the project files but also the entire version history.

```
# Initialize a new Git repository
git init

# Clone an existing repository
git clone repository_URL

# Clone a repository into a specific folder
git clone repository_URL folder_name
```

Init
(GitKraken)

Using GitKraken, the operations of initializing and cloning a Git repository can be carried out via an intuitive graphical interface, making these processes more accessible to beginners or more comfortable for those who prefer a visual interface.

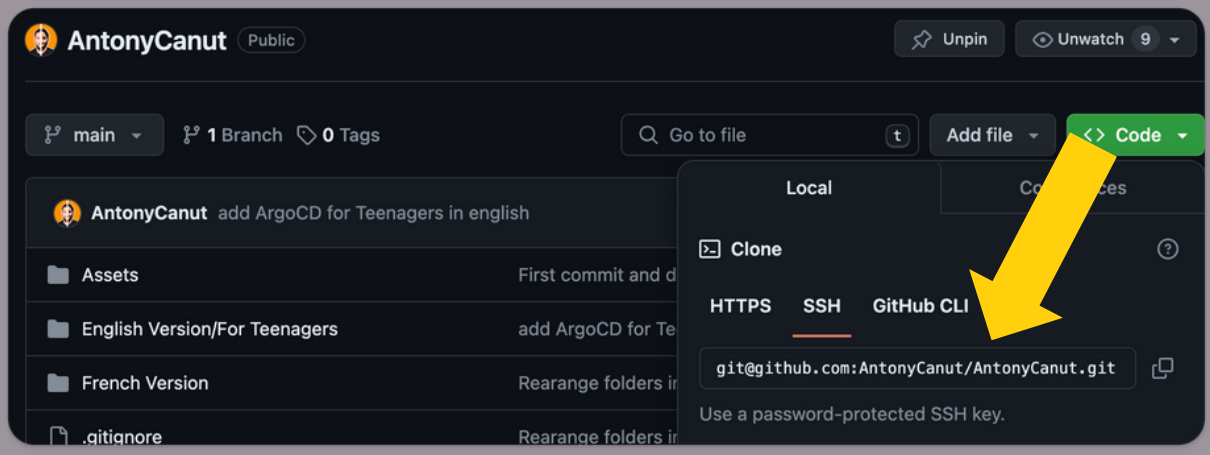
GitKraken facilitates the initialization of a new Git repository without having to open a terminal.



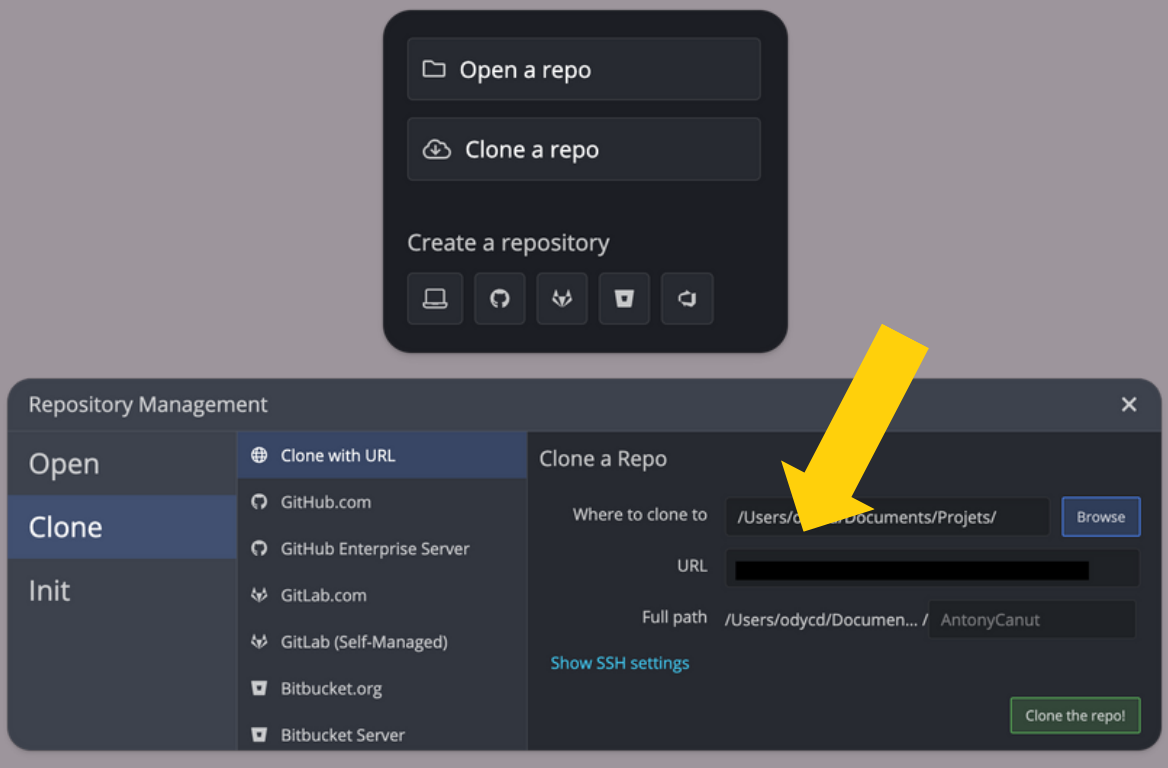
Cloning an existing Git repository is just as simple with GitKraken, allowing you to download projects and their version history to your local machine.

Clone
(GitKraken)

GITHUB



GITKRAKEN



Fetch & Pull

In the daily management of your projects with Git, it's essential to know how to retrieve the latest changes from a remote repository and integrate them into your local repository. The fetch and pull commands are at the heart of this process, allowing you to synchronize your work with that of other contributors.

The fetch command downloads changes from the remote repository but does not merge them into your current working branch. It's a way to stay informed about modifications without applying them immediately.

The pull command, on the other hand, is essentially a combination of fetch followed by a merge. It downloads the changes from the remote repository and directly merges them into your current working branch.

Tools like GitKraken automate the fetch process and facilitate the pulling process.

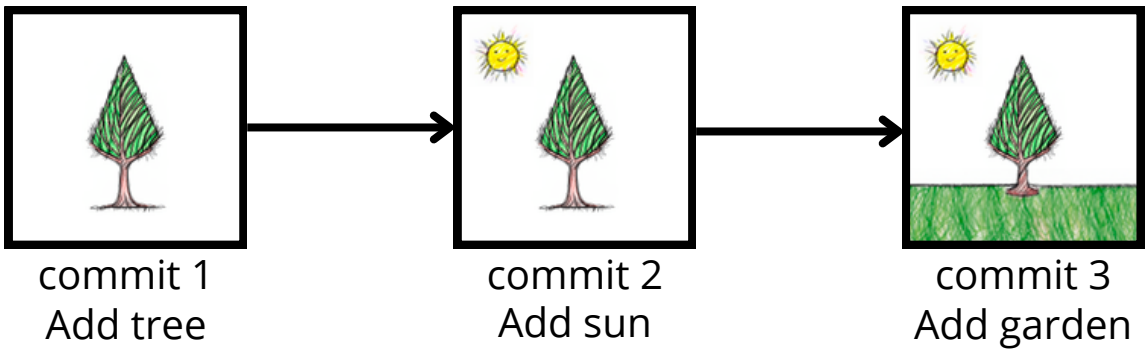
```
# Download changes from the remote repository
git fetch origin

# Download and merge changes from the remote repository
git pull origin main
```

Commit

Commits are at the heart of Git's versioning system, allowing you to save snapshots of your project at a given time. These snapshots include not only your files and their modifications but also a commit message describing the changes.

To make a commit with Git, you first need to add the modified files to the staging area (index) with git add, and then use git commit to create the snapshot.

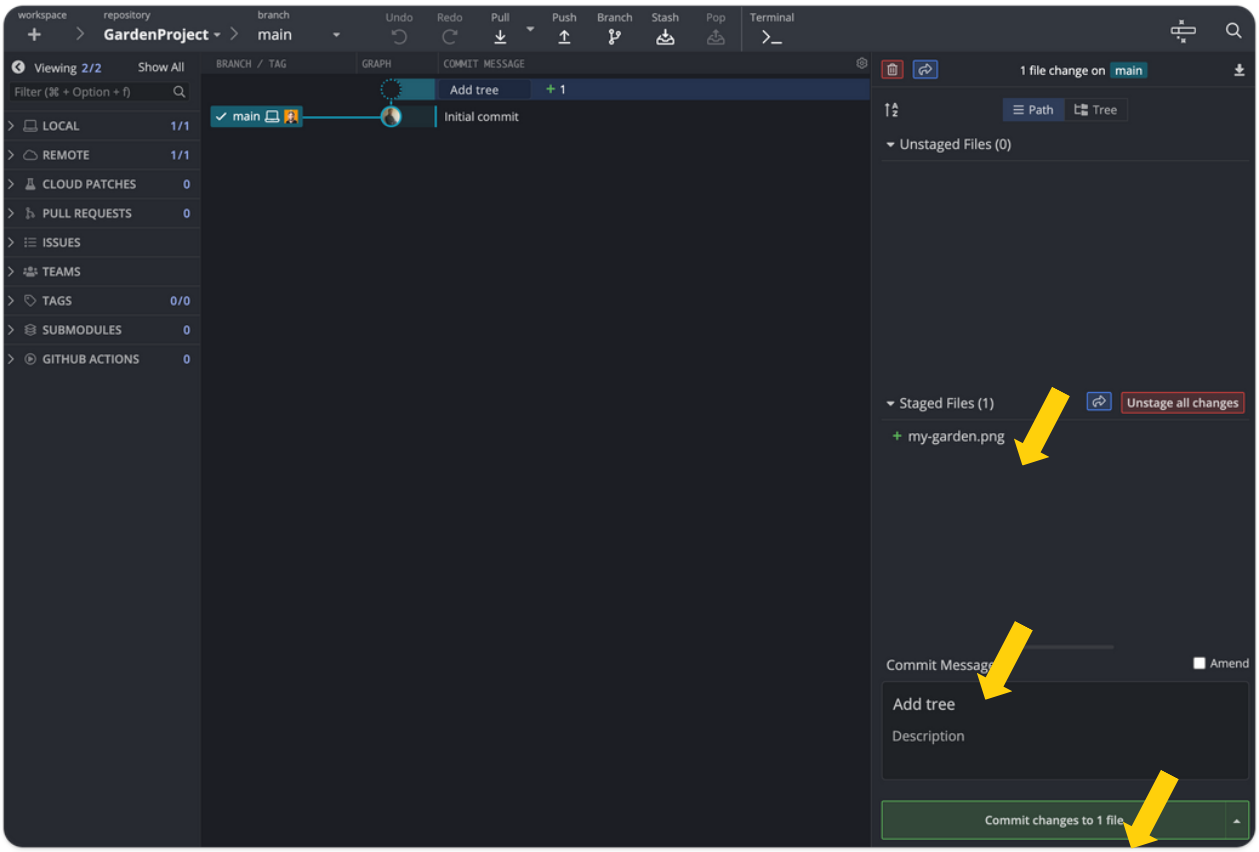
```
# Add files to the staging area
git add path/to/your/file

# Create a commit
git commit -m "Your commit message"
```

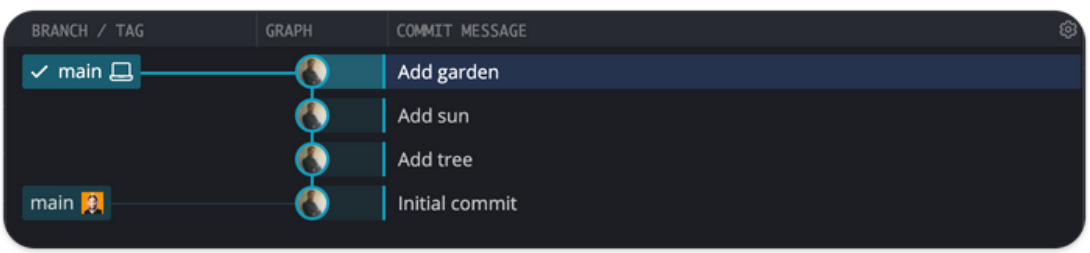


Commit
(GitKraken)

GitKraken clearly displays modified, untracked, and pending commit files, allowing you to easily prepare your next commit without having to remember file paths or specific commands.

You can select the files to include in the commit directly from the GitKraken interface, by checking the boxes next to each file. This avoids having to type each file name individually in the terminal.





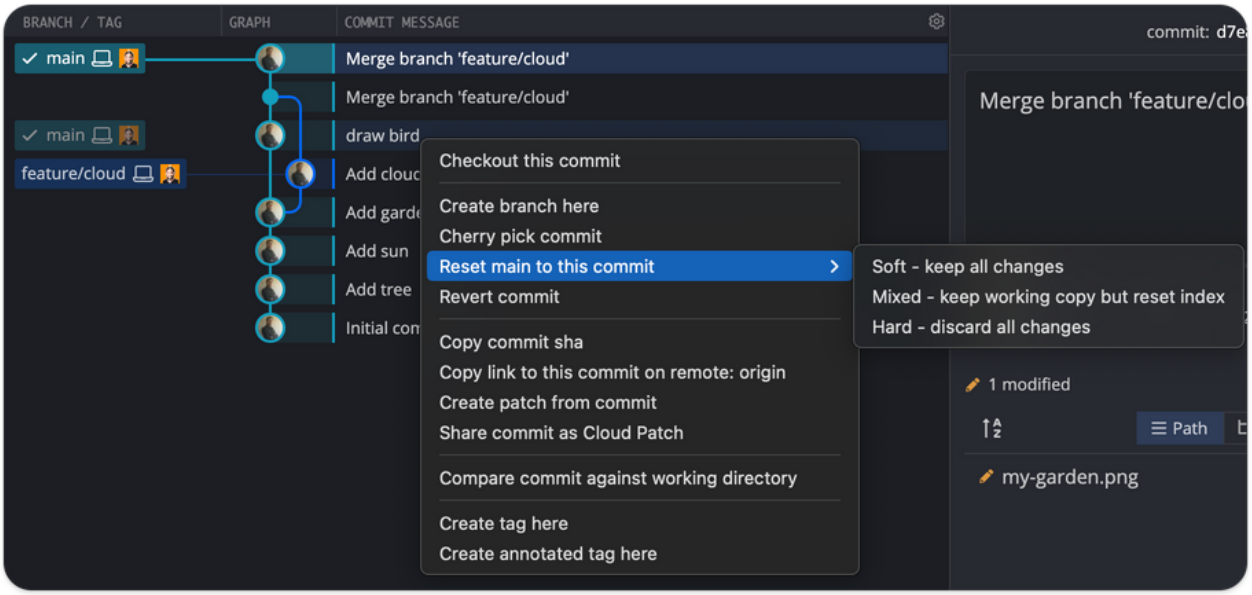
History

Navigating Git history and exploring previous commits are essential skills for any Git user. Whether it's for auditing changes, understanding the evolution of a project, or simply reverting to a stable earlier version.

```
# View the commit history
git log

# Display the history in a one-line format
git log --pretty=oneline

# View changes introduced by each commit
git log -p

# Compare the current content with a specific commit
git diff <commit_hash>

# Examine the content of a specific commit
git show <commit_hash>

# Move to the state of a specific commit
git checkout <commit_hash>

# Display the history of changes to a specific file
git log -- <file_path>

# Revert to a previous commit while keeping recent changes
git reset --soft <commit_hash>
```

History
(GitKraken)

If you want to examine the state of the project at a given commit or test an old version, GitKraken allows you to checkout a commit simply by right-clicking on the desired commit and selecting "Checkout this commit". This will put your workspace in the state it was at the time of that commit.

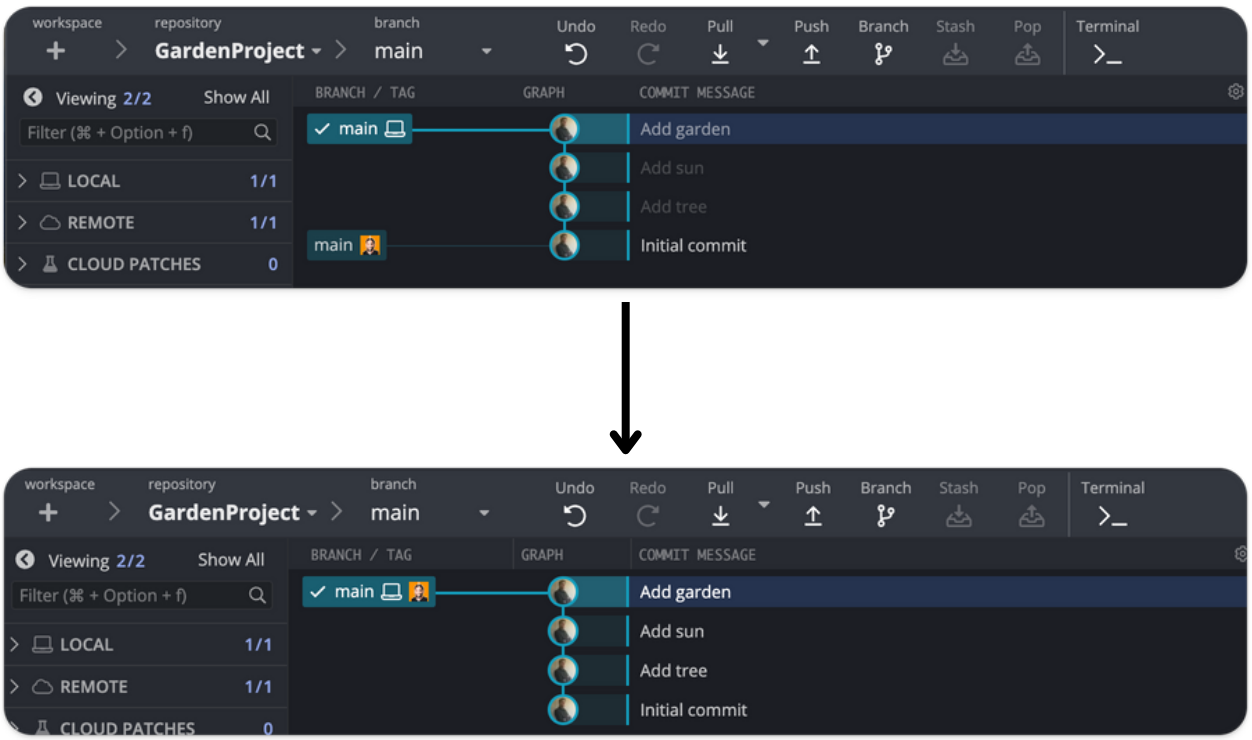


Push

After creating commits locally, the next step in the Git workflow is usually to "push" these commits to a remote repository to share your changes with the team or to back up your work on an external server.

This remote repository can be hosted on Github, Gitlab, Azure DevOps, and many others.

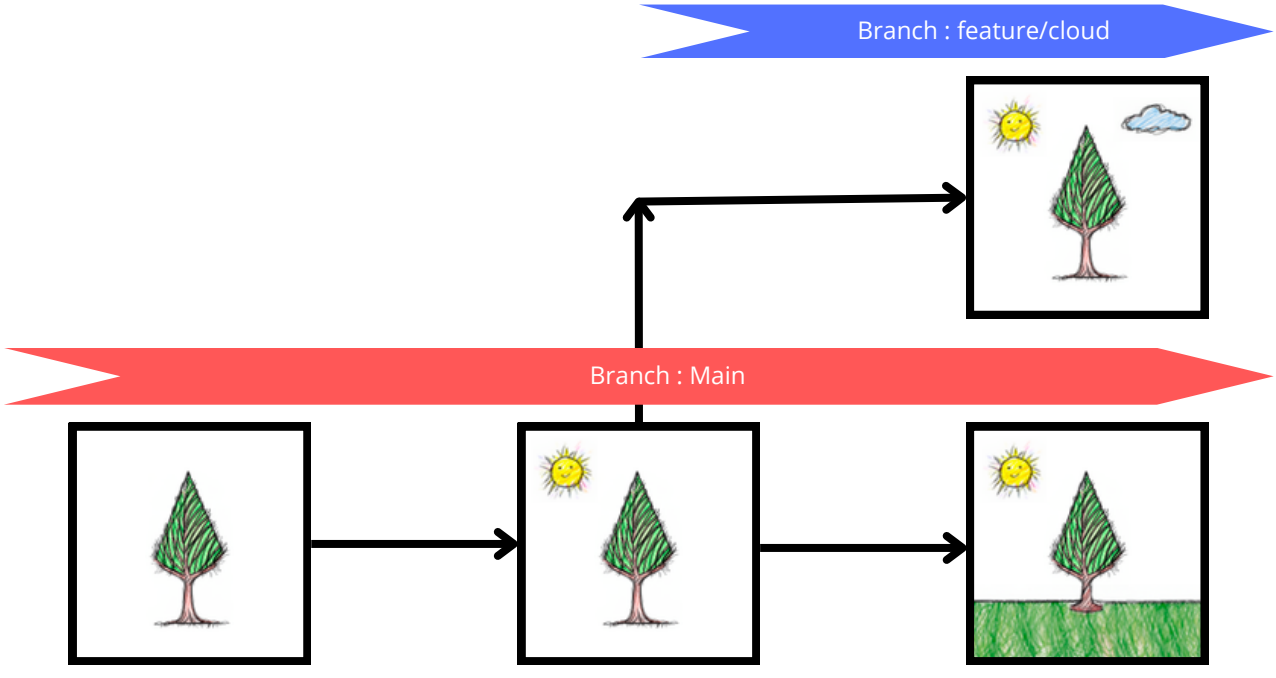
```
# Push local commits to the remote repository
git push
```



Branch

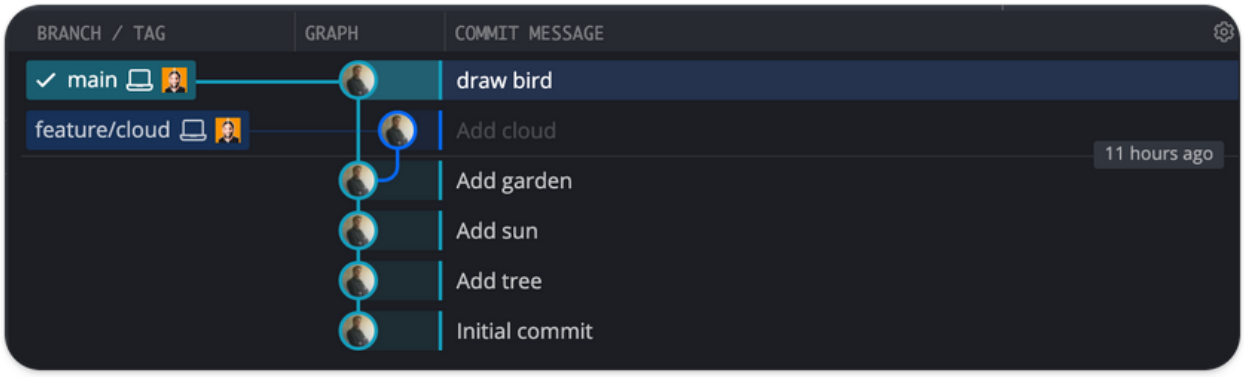
Branches are a key concept in Git, allowing developers to work on different features, fixes, or experiments in parallel without interfering with the work of others or with the main development line.

```
# Create a new branch
git branch new_branch_name

# Switch to an existing branch
git checkout branch_name

# Delete a local branch
git branch -d branch_name
# Use '-d' to delete a merged branch or '-D' to force
deletion.
```

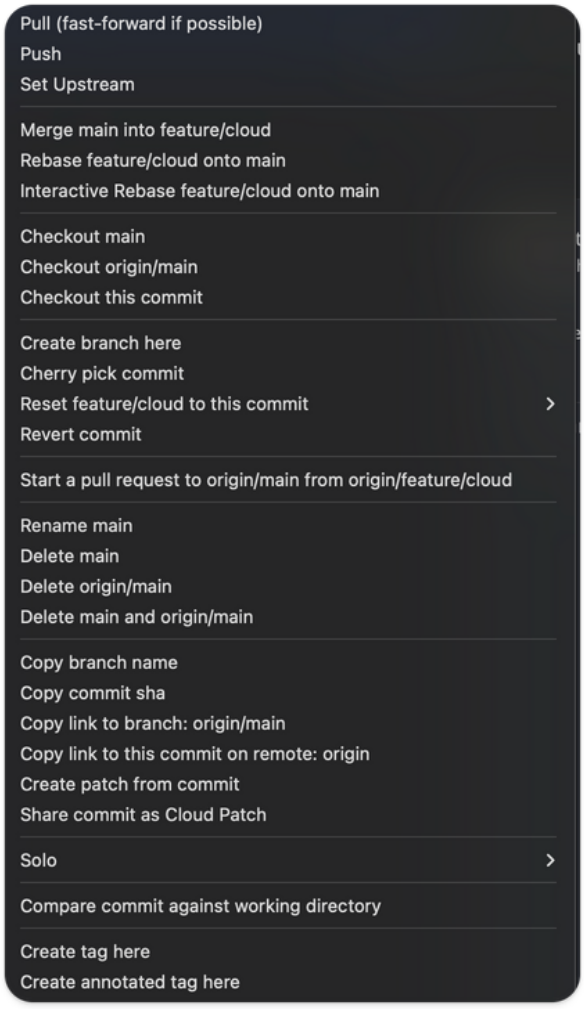


Branch
(GitKraken)

GitKraken simplifies and enriches the branch management experience through its intuitive graphical interface. Creating branches is done with a single click, and their management is very visual.



Creating or deleting branches in GitKraken is done through a simple interface, often with just a right-click and the selection of an option from a menu. This avoids having to type specific commands and reduces the risk of error.

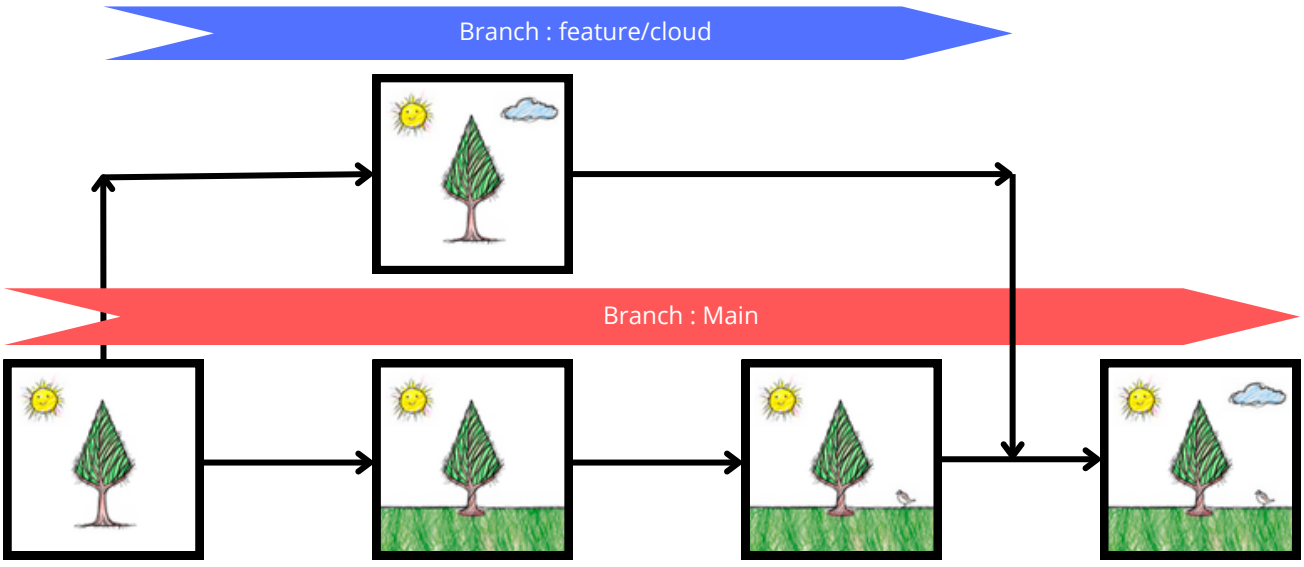


Switching between branches is also simplified, with an interface that allows you to select the desired branch from a dropdown list or the branch graph.

Merge

The merge is a key process in Git that allows integrating changes from one branch into another, often used to combine work on features developed in parallel or to incorporate fixes into the main branch.

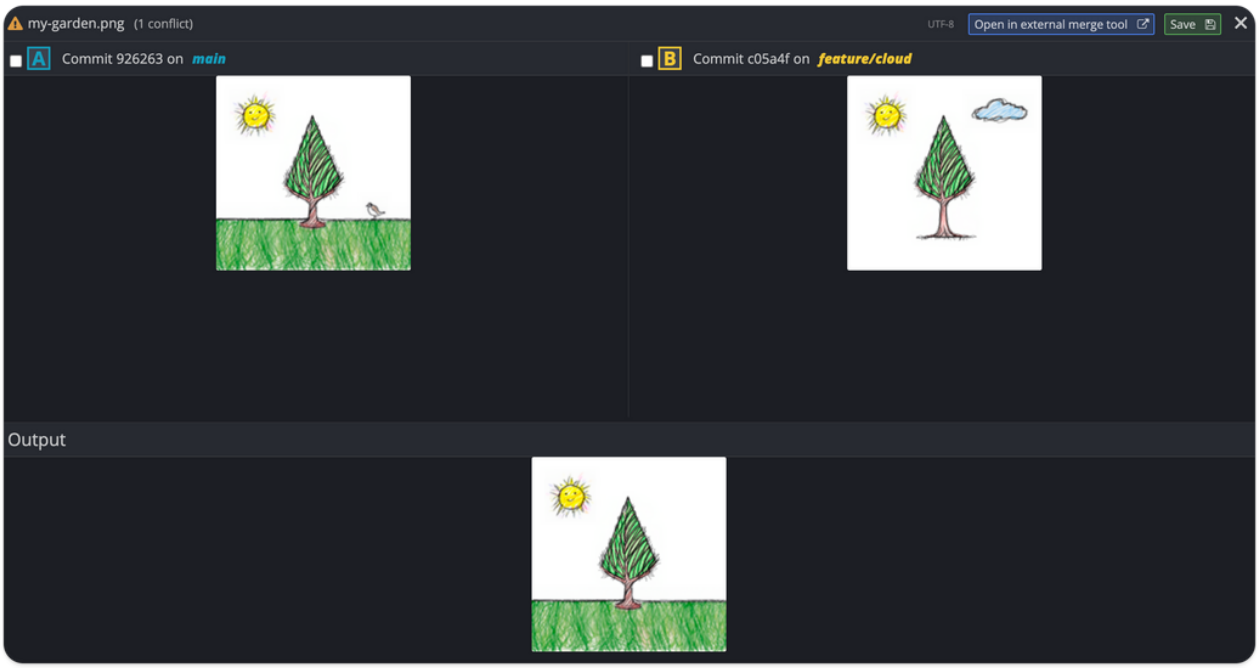
```
# Switch to the branch that will receive the changes
git checkout receiving_branch
# Make sure you are on the branch that should receive the
changes.

# Merge the branch
git merge source_branch
# This command integrates the changes from "source_branch"
into "receiving_branch".
```

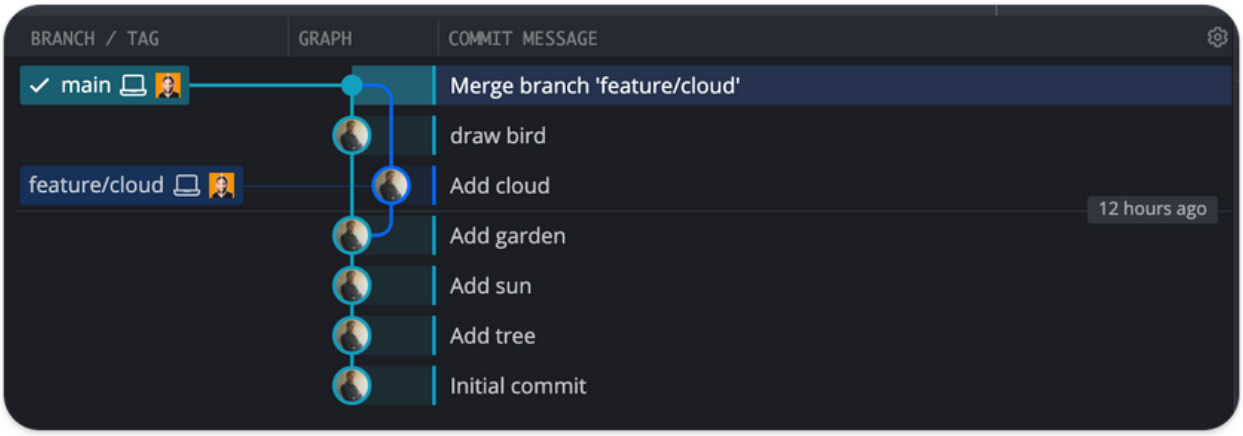


Merge (GitKraken)

When a merge is initiated, GitKraken provides a simplified interface that summarizes the changes to be integrated and allows starting the process in a few clicks.

If conflicts arise, GitKraken presents an integrated conflict resolution tool, making the resolution process clearer and less intimidating than in the command line. This tool allows for manually choosing between versions of conflicting snippets or merging them line by line.





Rebase

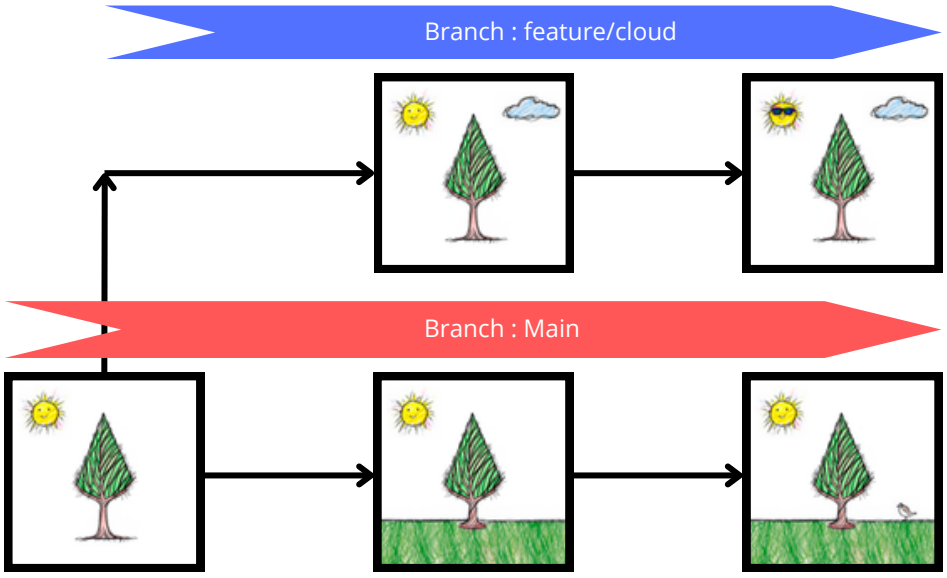
Rebase is a powerful feature of Git that allows modifying the commit history of a branch. It is often used to update a feature branch with the latest changes from another branch, typically the main branch, or to clean up the history before merging a branch.

Interactive rebase (git rebase -i) is a powerful tool for modifying commit history. It allows for squashing, editing, deleting, or rearranging commits.

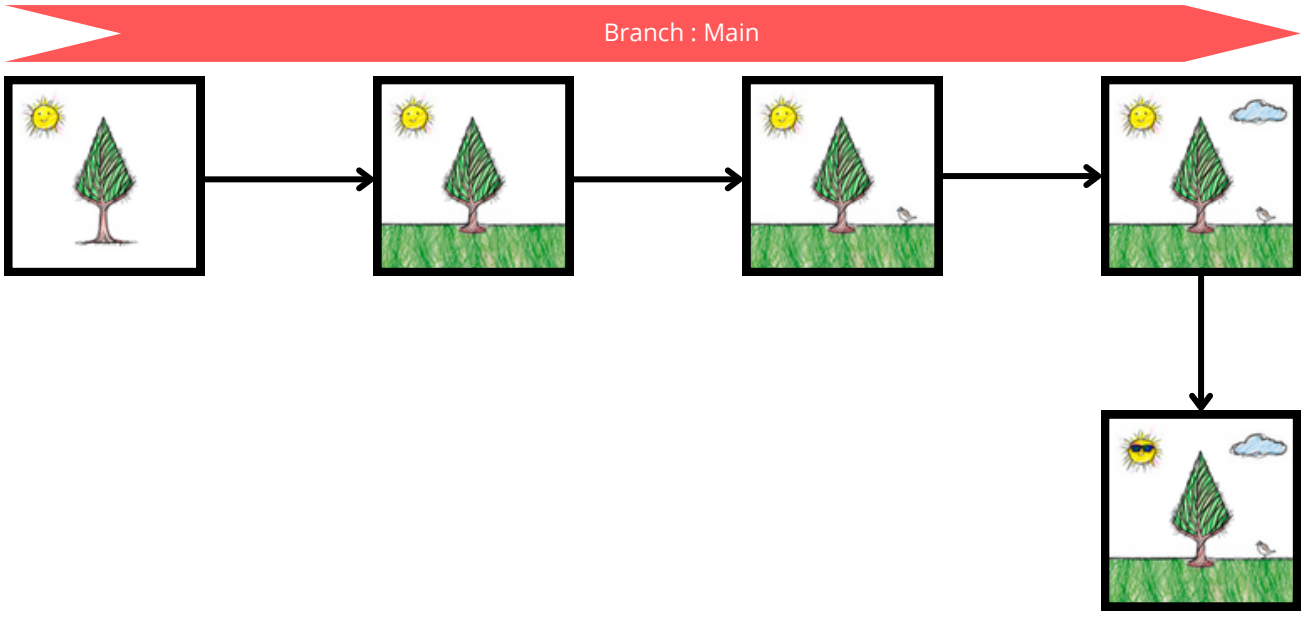
```
# Update the feature branch with the latest changes from the
main branch
git checkout feature-branch
git rebase master

# Start an interactive rebase for the last X commits
git rebase -i HEAD~X

# If the rebase doesn't go as planned, cancel it
git rebase --abort
```

Rebase

Before Rebase



After Rebase



Merge vs Rebase

Although Merge and Rebase achieve the same goal: the integration of changes, they do so in different ways and have their own advantages and use cases.

Merging involves taking the commits from one branch (for example, a feature branch) and integrating them into another (often the main or master branch). A merge commit is created for this integration, preserving the history of all the commits from both the feature branch and the target branch.

Advantages:

- Preserves the complete history of commits.
- Merge commits provide clear context about the integration of branches.

Rebasing rewrites the history by moving the feature branch to start from the latest commit of the target branch. This creates a linear sequence of commits, as if all the work had been done in chronological order.

Advantages:

- Creates a clean, linear project history, making it easier to navigate and review.
- Avoids unnecessary merge commits and makes the history more understandable.

Stash

The "stash" is a Git feature that allows temporarily setting aside uncommitted changes so you can switch branches with a clean working directory. This capability is particularly useful in multitasking workflows where a clean development context is needed to switch between different tasks.

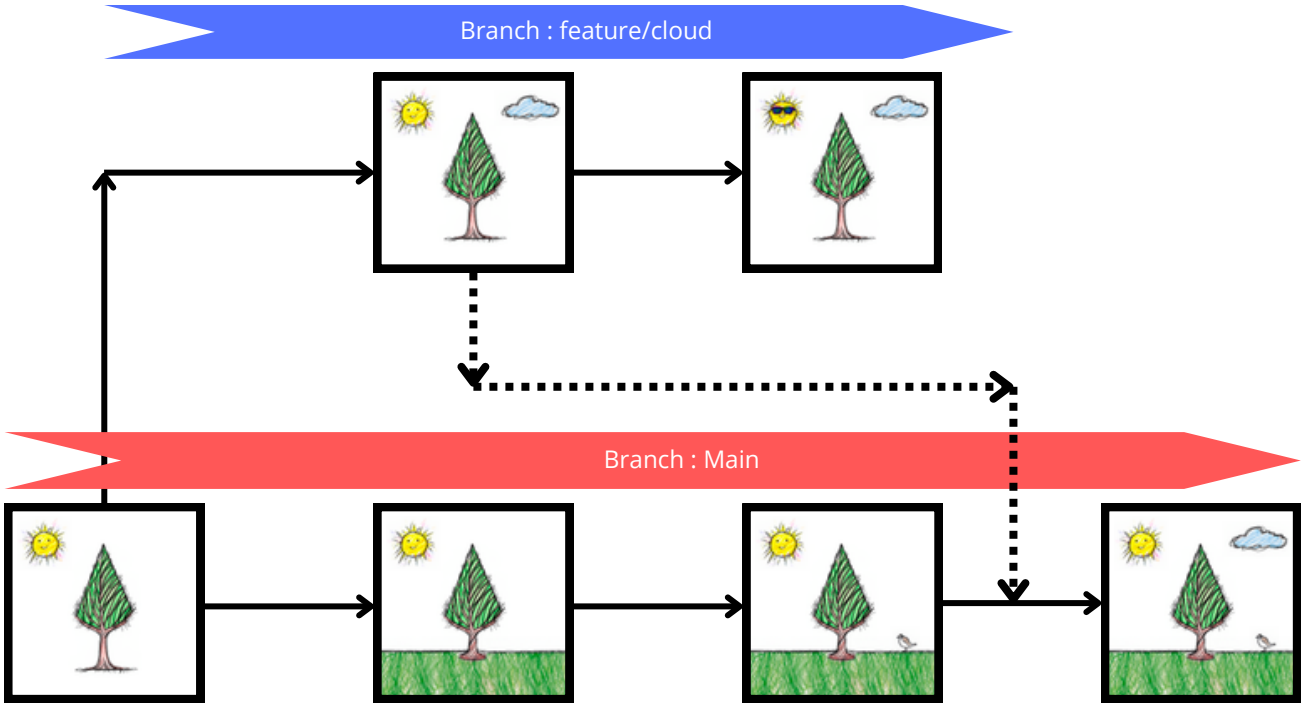
```
# Stash the current modifications
git stash push -m "Descriptive message for the stash"
# This command sets aside the current modifications and
associates a descriptive message with them.

# List all stashes
git stash list
# Displays a list of all available stashes with their
identifiers and associated messages.

# Apply the last stash
git stash pop
# This command applies the modifications from the last stash
and removes it from the stash stack.
```

With GitKraken, applying a stash to your working directory is as simple as clicking on the desired stash and selecting the appropriate action, without needing to remember specific commands or stash identifiers.

Cherry Pick

The "cherry pick" is an advanced Git feature that allows selecting and applying specific commits from one branch to another. This can be particularly useful for integrating specific fixes or features without merging the entire branches.

```
# Perform a cherry pick of a commit
git cherry-pick <commit_hash>
# Replace "<commit_hash>" with the hash of the commit you
wish to apply to the current branch
```



Bisect

Git bisect is a powerful tool for diagnosing and identifying the specific commit that introduced a bug into the code. Using a binary search approach, Git bisect allows quickly narrowing down the search space to find the faulty commit.

Unfortunately, GitKraken will still require the user to use command lines, but its interface will make the exercise simpler.

```
# Start a bisect session
git bisect start
# This tells Git to start the bisect process.

# Mark the current commit or a specific commit as bad
git bisect bad <commit_hash>
# If no hash is provided, Git uses the current commit. This
marks the starting point for the search.

# Mark a known commit without the bug as good
git bisect good <commit_hash>
# This sets the comparison point for the binary search.

# Git then bisects the commit history to find the first bad
commit.
# Once identified, Git will indicate the first bad commit,
i.e., the commit that introduced the bug.

# End the bisect session
git bisect reset
# This ends the bisect process and returns to the previously
checked-out branch.
```

.gitignore

The .gitignore file plays a crucial role in Git projects, allowing developers to specify files and folders to be ignored in version tracking. This mechanism ensures that non-essential, temporary, or sensitive files are not inadvertently added to the repository. It is located at the root of your project.

```
# Ignore all .log files
*.log

# Ignore a specific folder
node_modules/

# Do not ignore a specific file despite previous rules
!important.log

# Ignore all files in a specific folder, except for a
subfolder
build/
!build/important/
```

gitignore.io is an online service that simplifies the creation of well-structured .gitignore files suited for many development environments, programming languages, and tools. This tool is particularly useful for developers working on complex projects or who use multiple tools and languages, as it helps automatically generate .gitignore rules that match their specific needs.

Tag

Tags in Git are used to mark specific points in a project's history, often used to identify release versions. Unlike branches, which evolve over time, tags are fixed and represent a permanent snapshot of a project's state at a given moment.

```
# List all existing tags
git tag

# Create a lightweight tag
git tag tag_name

# Create an annotated tag with a message
git tag -a tag_name -m "tag message"

# Display information about an annotated tag
git show tag_name

# Push a specific tag to a remote repository
git push origin tag_name

# Push all tags to the remote repository
git push origin --tags

# Delete a tag locally
git tag -d tag_name

# Delete a tag from a remote repository
git push --delete origin tag_name
```

Submodules

Git submodules allow integrating and managing Git projects within another Git project, acting as references to specific repositories at a particular commit. This feature is useful for including external libraries, frameworks, or other projects on which your main project depends.

```
# Add a submodule
git submodule add <repository-url> <path-to-submodule>

# Initialize submodules
git submodule init

# Update submodules
git submodule update
```

To make changes to a submodule, navigate to its directory and work as you would in a normal Git repository. For the changes to be reflected in your main project, you will need to commit the changes in the submodule, then commit the new referenced commit in the main project.

If you have write access to the submodule's repository, you can push the changes directly from the submodule's directory. Also, make sure to push the update commit in the main project so that other users can get the same version of the submodule.

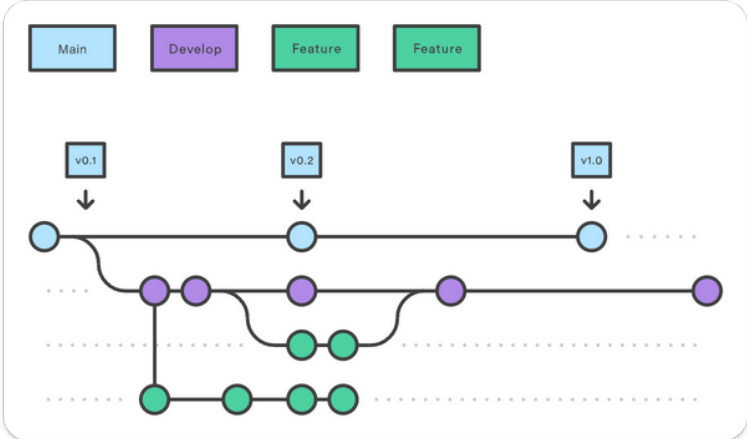
Methodologies
GitFlow

Effective branch management is crucial in any software development project using Git. Several methodologies have been developed to structure workflow, with GitFlow being one of the most popular.

GitFlow is a branch management methodology designed for release-based software development. It defines a fixed structure of branches for different tasks, such as development, releases, and emergency fixes.

- Main branches:
  - main: contains the production code.
  - develop: serves as the main development branch, where features are merged for upcoming releases.

- Support branches:
  - feature: branches used to develop new features.
  - release: prepare a new production version.
  - hotfix: allow for quick fixes on the production branch.



source : Atlassian

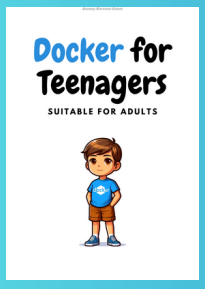
Git... Hub ?

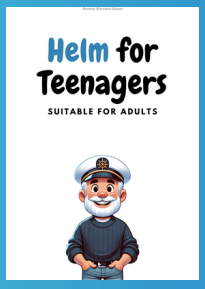
GitHub is a collaborative development platform used for hosting projects utilizing the Git version control system. It facilitates collaboration among developers by providing tools for code sharing, project management, and the integration of various services.

- Git Repositories: GitHub allows users to create and manage Git repositories where project code is stored, versioned, and shared.
- Fork and Pull Request: Users can "fork" projects (create a personal copy) and submit "pull requests" to propose modifications to third-party projects, thus facilitating open collaboration.
- Project Management: With features like issues, milestones, and projects, GitHub helps teams organize development, track bugs, and plan tasks.
- GitHub Actions: An automation tool to create custom workflows, including continuous integration and delivery (CI/CD), testing, and more.
- GitHub Pages: A free hosting service to publish websites directly from GitHub repositories.
- Code Review: Pull requests include a code review system, allowing teams to comment on and approve changes before they are integrated.
- Security: GitHub offers tools to identify and fix security vulnerabilities in project dependencies.

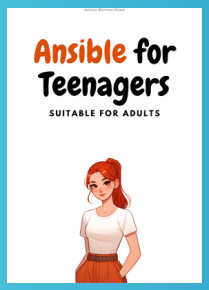
In the same collection

Container Orchestration and Management



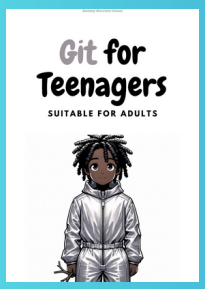


Infrastructure as Code



Security & Secrets Management

Development & CI/CD



↓ FOLLOW ME ↓

ANTONYCANUT

ANTONY KERVAZO-CANUT