Git

/gɪt/

http://git-scm.com

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency

Initially designed and developed by Linus Torvalds for Linux Kernel development.

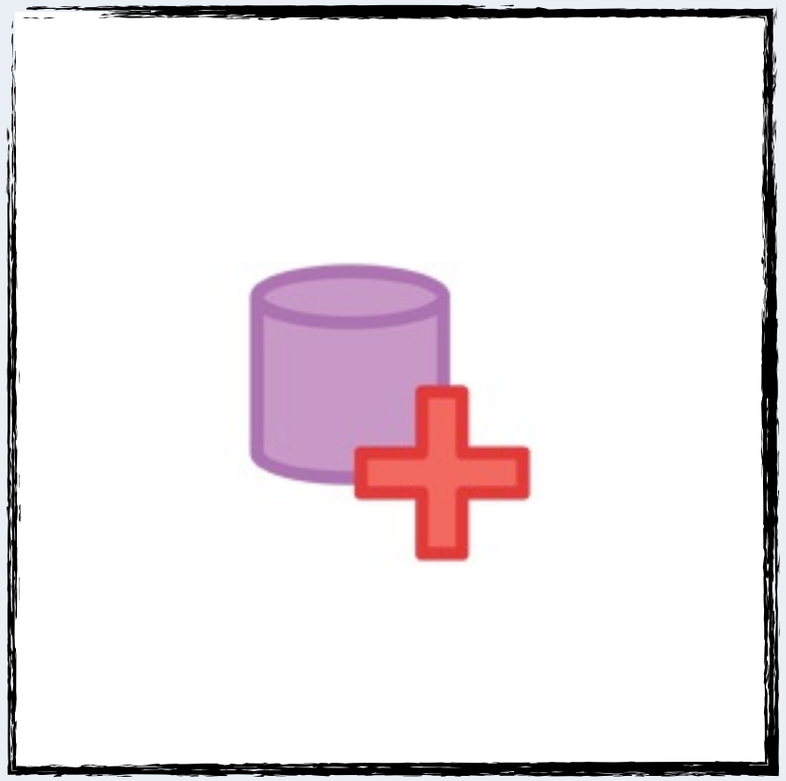
Initial released: 7 April 2005
License: GNU General Public License v2

Git Basics

Setting up a Git repository

git init

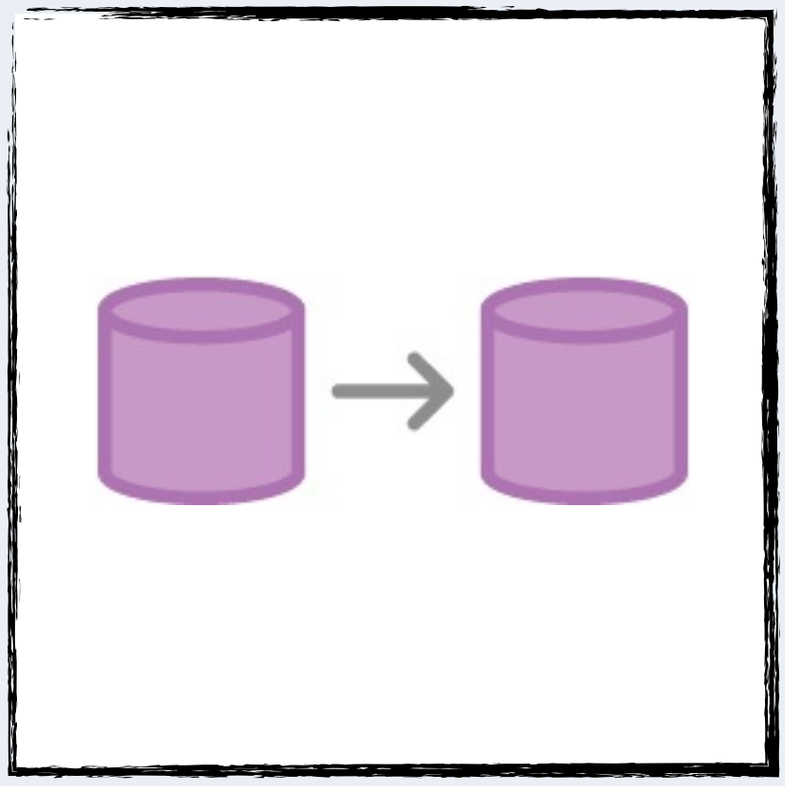
The git init command initialises a new Git repository. If you want to place a project under revision control, this is the first command you need to learn.



```
git init <directory>
```

git clone

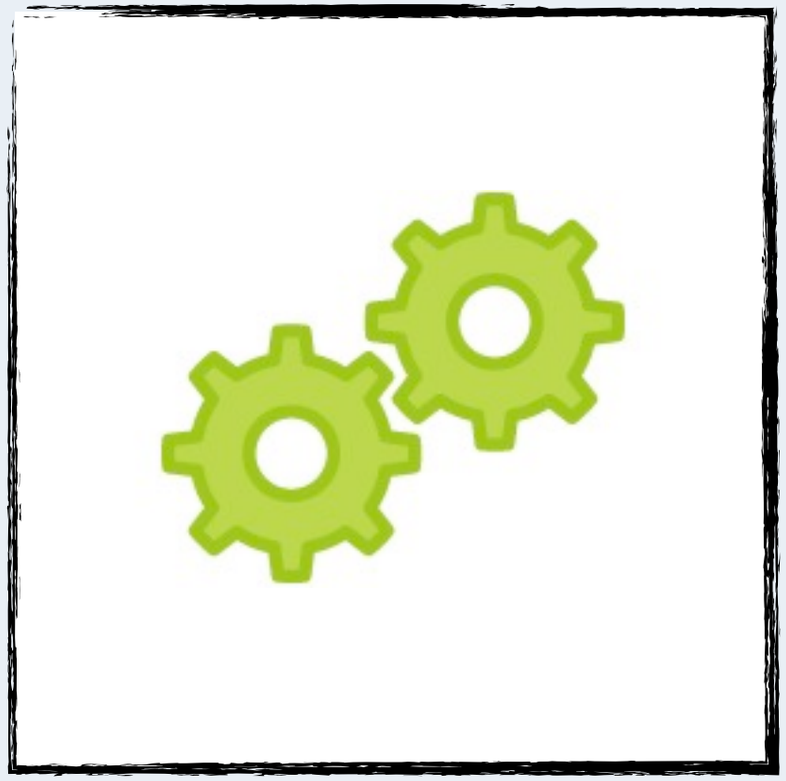
The git clone command creates a copy of an existing Git repository. Cloning is the most common way for developers to obtain a working copy of a central repository.



```
git clone <repo> <directory>
```

git config

The git config is a convenient way to set configuration options for your Git installation. You'll typically only need to use this immediately after installing git on a new development machine.



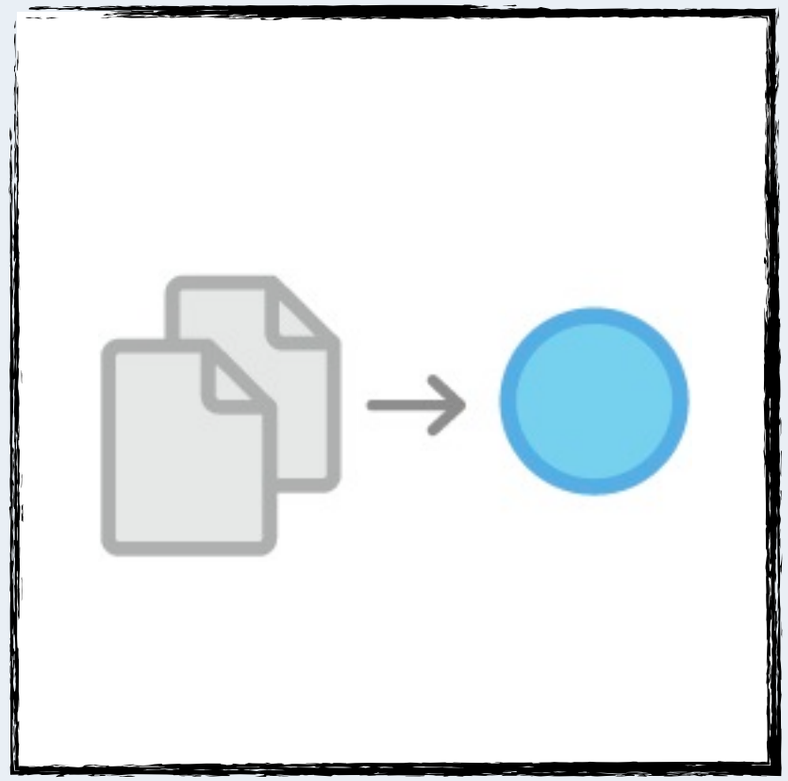
```
git config <options>
```

Git Basics

Recording snapshots

git add

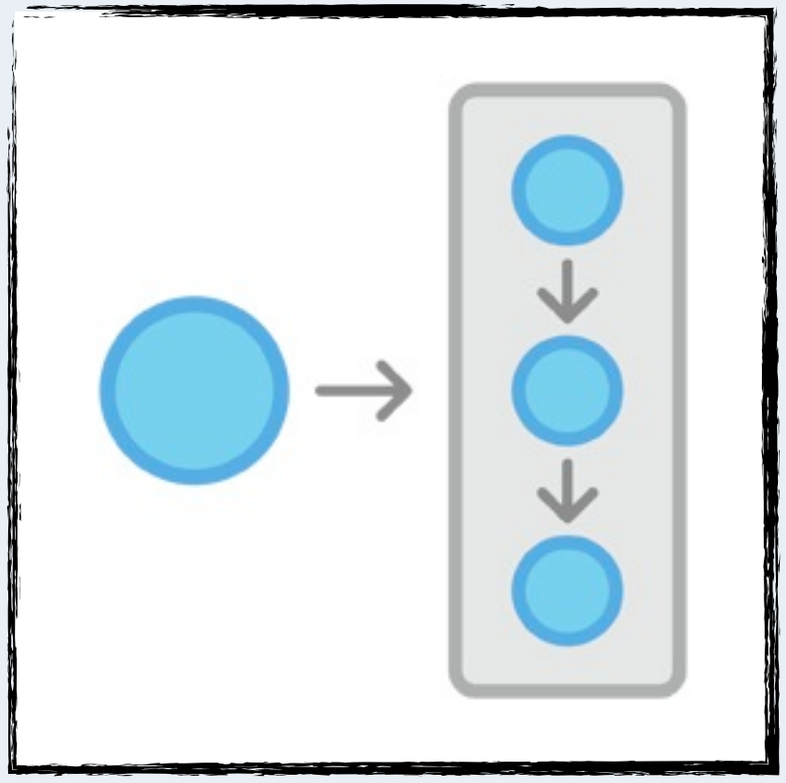
The git add command moves changes from the working directory to the staging area. This gives you the opportunity to prepare a snapshot before committing it to the official history.



```
git add <file/directory>
```

git commit

The git commit takes the staged snapshot and commits it to the project history. Combined with git add, this defines the basic workflow for all Git users.



```
git commit -m <message>
```

Git Basics

Inspecting a Git repository

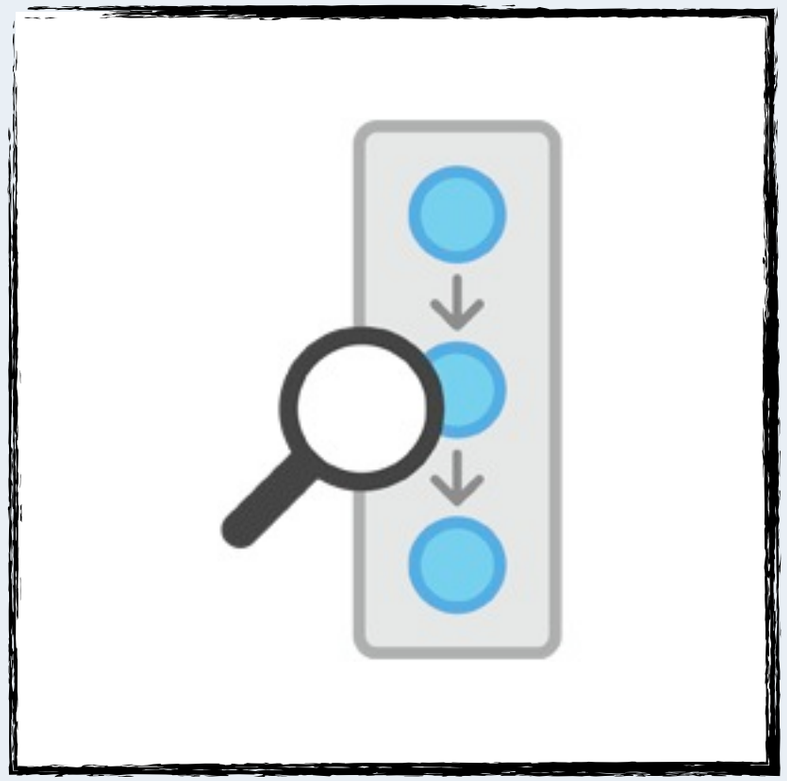
git status

The git status command displays the state of the working directory and the staged snapshot. You'll want to run this in conjunction with git add and git commit to see exactly what's going being included in the next snapshot.

```
git status
```

git log

The git log command lets you explore the previous revisions of a project. It provides several formatting options for displaying committed snapshots.



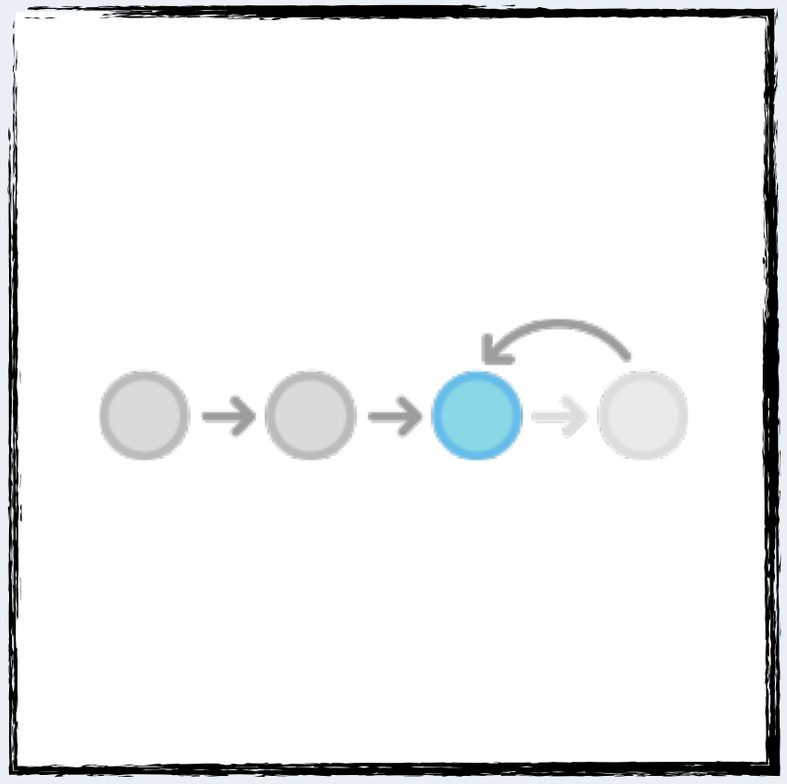
```
git log <options>
```

Undoing Changes

Viewing old commits

git checkout

The git checkout command serves 3 different functions: checking out files, checking out commits and checking out branches.



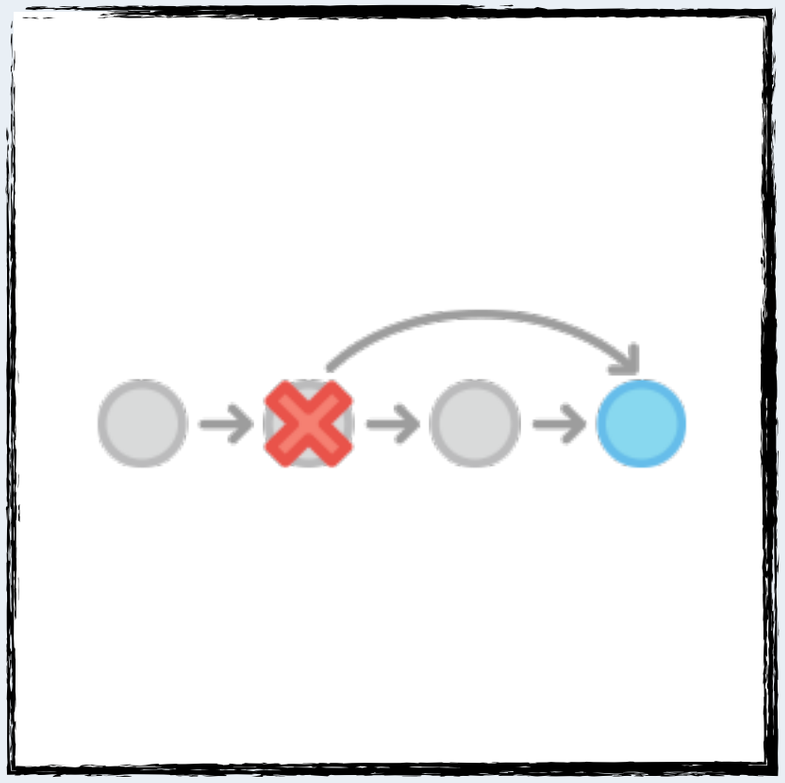
```
git checkout <commit/branch>
```

Undoing Changes

Undoing public changes

git revert

The git revert commands undoes a committed snapshot. When you discover a faulty commit, reverting is a safe and easy way to completely remove it from the code base.



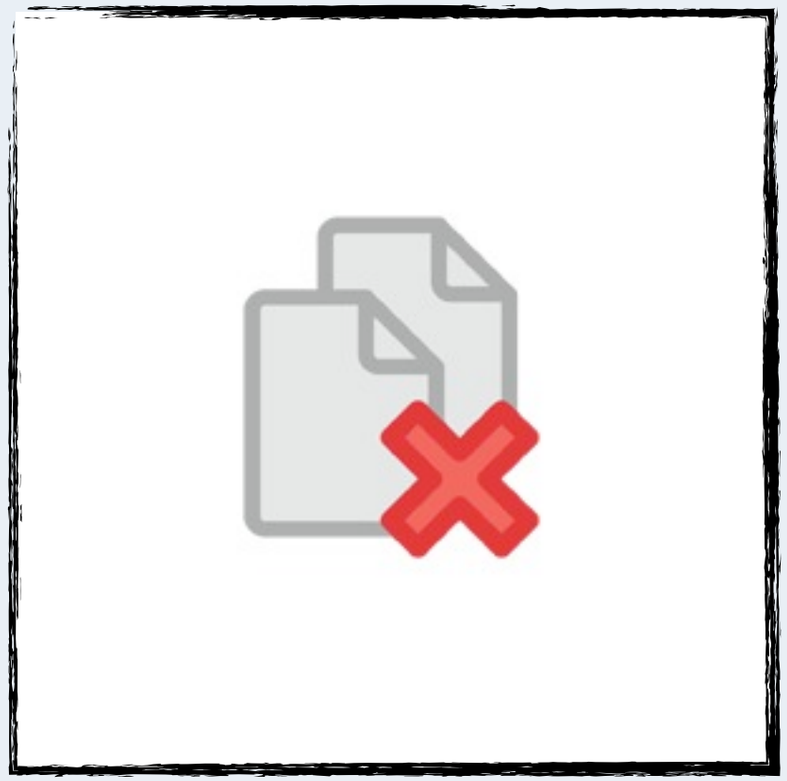
```
git revert <commit>
```

Undoing Changes

Undoing local changes

git reset

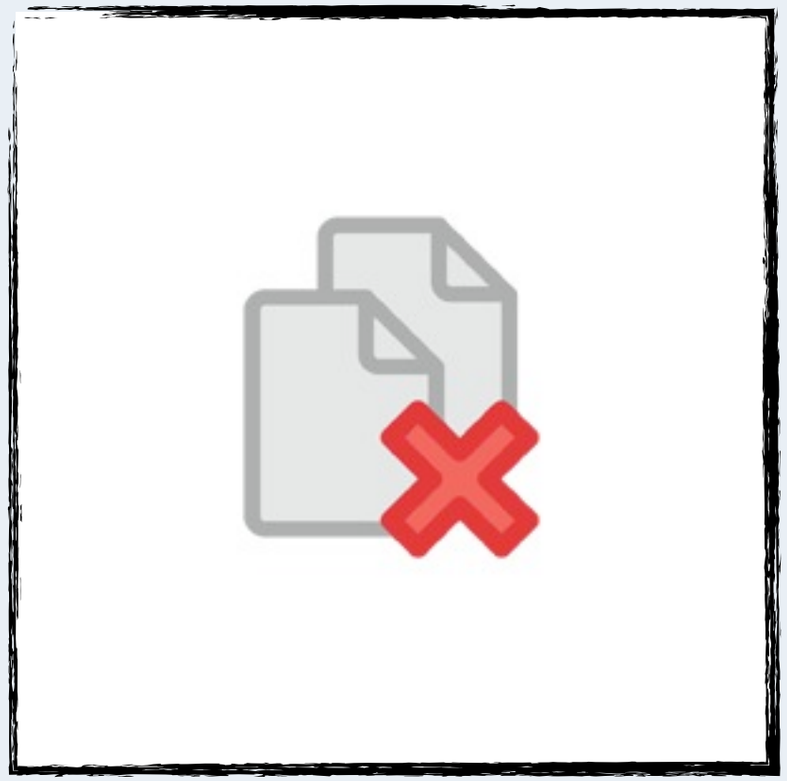
The git reset commands undoes changes to files in the working directory. Resetting lets you clean up or completely remove changes that have not been pushed to a public repository.



```
git reset <options> <file>
```

git clean

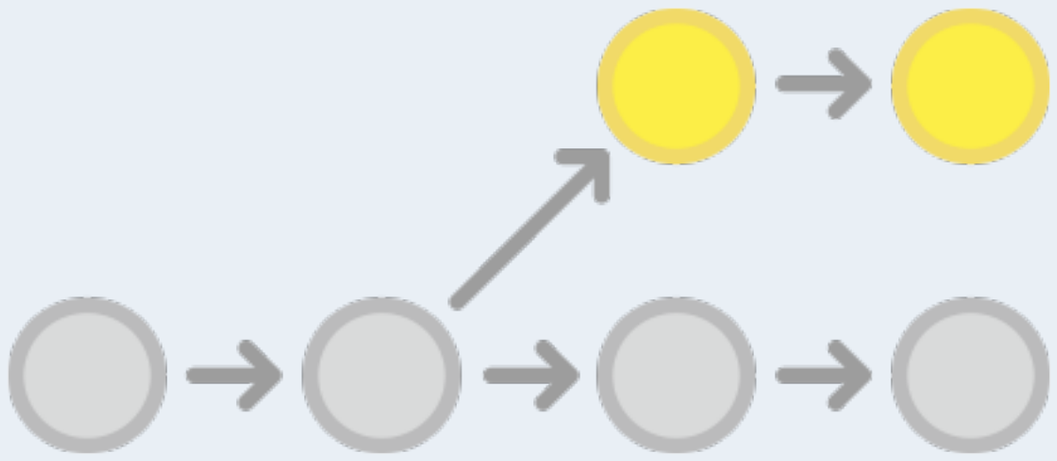
The git clean commands removes untracked files from the working directory. This is the logical counterpart to git reset, which (typically) only operates on tracked files.



```
git clean <options>
```

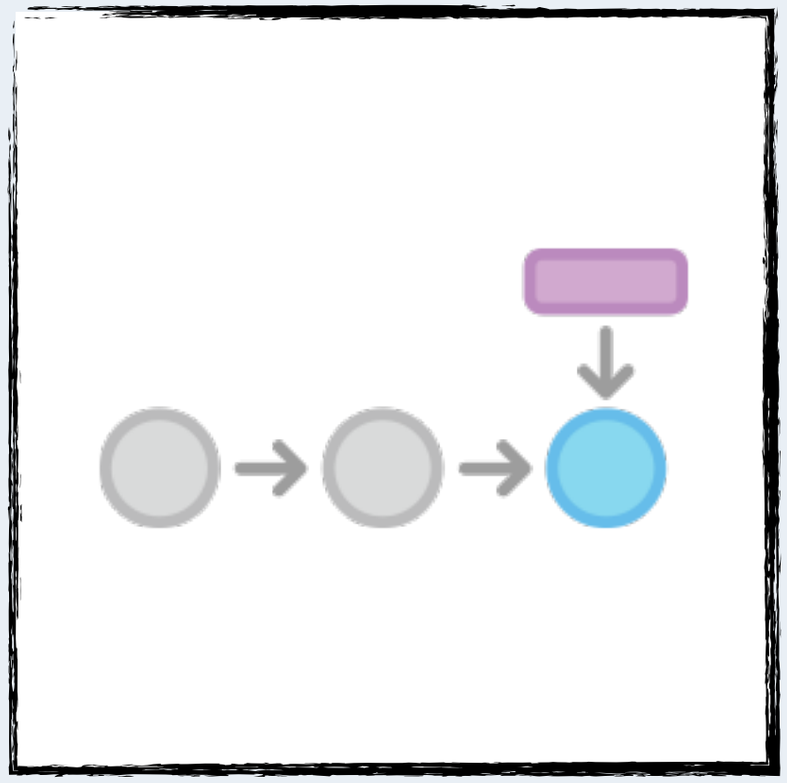
Git Branches

Git branches



git branch

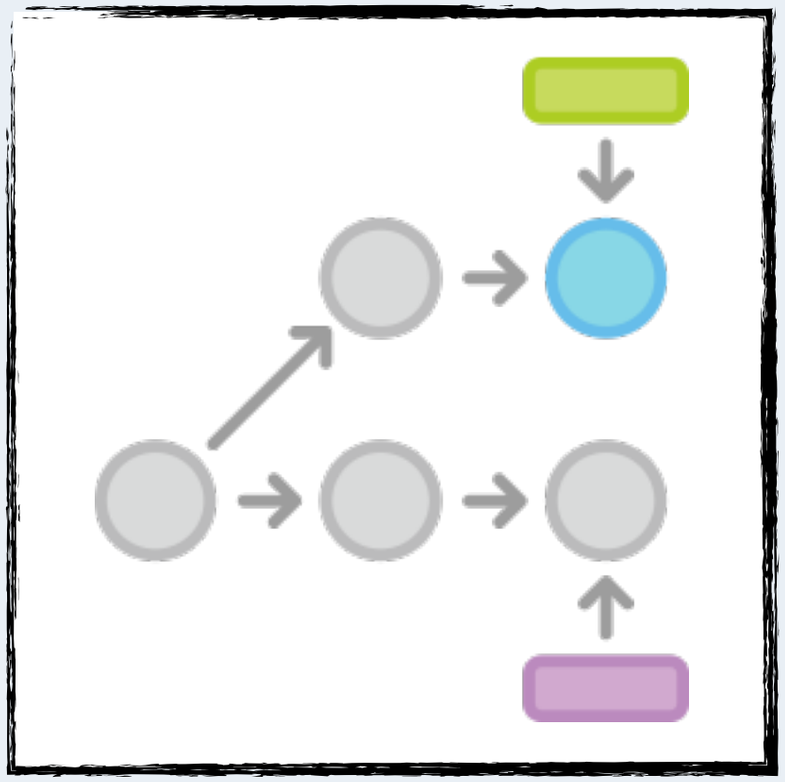
The git branch command is your general-purpose branch administration tool. It lets you create isolated development environments within a single repository.



```
git branch <branch>
```

git checkout

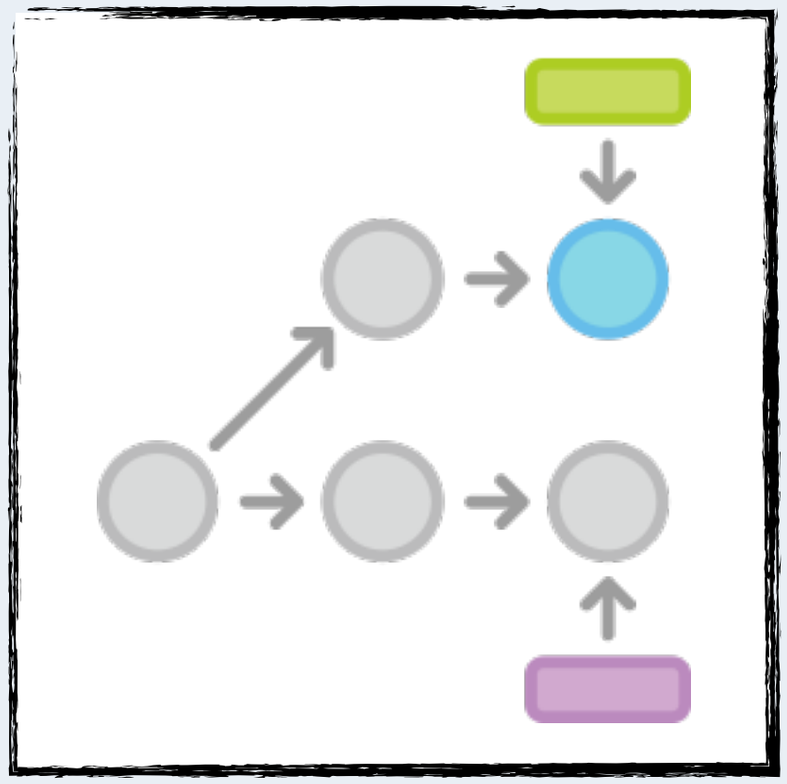
In addition to checking out old commits and old file revisions, git checkout is also the means to navigate existing branches. Combined with the basics Git commands, it's a way to work on a particular line of development.



```
git checkout <options> <branch>
```

git merge

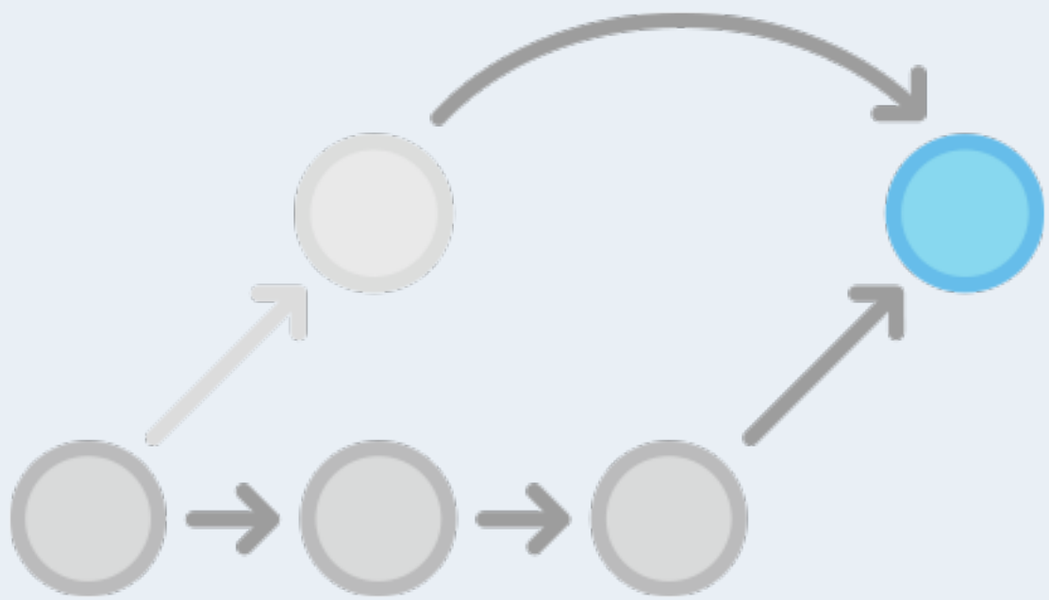
The git merge command is a powerful way to integrate changes from divergent branches. After forking the project history with git branch, git merge lets you put it back together again.



```
git merge <options> <branch>
```

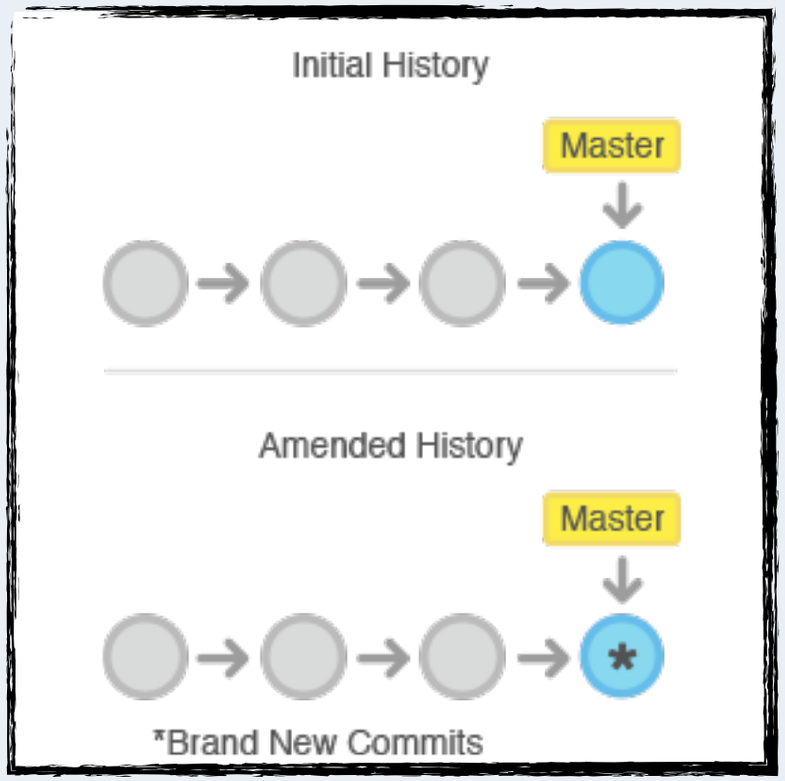
Rewriting Git History

Rewriting Git history



git commit --amend

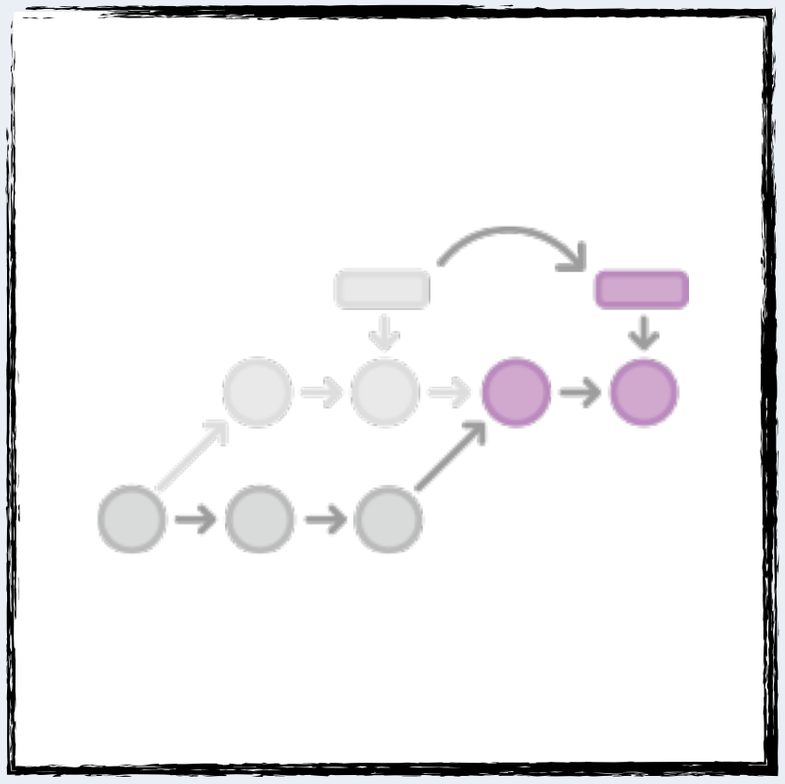
Passing the --amend flag to git commit lets you amend the most recent commit. This is very useful when you forget to stage a file or omit important information from the commit message.




```
git commit --amend
```

git rebase

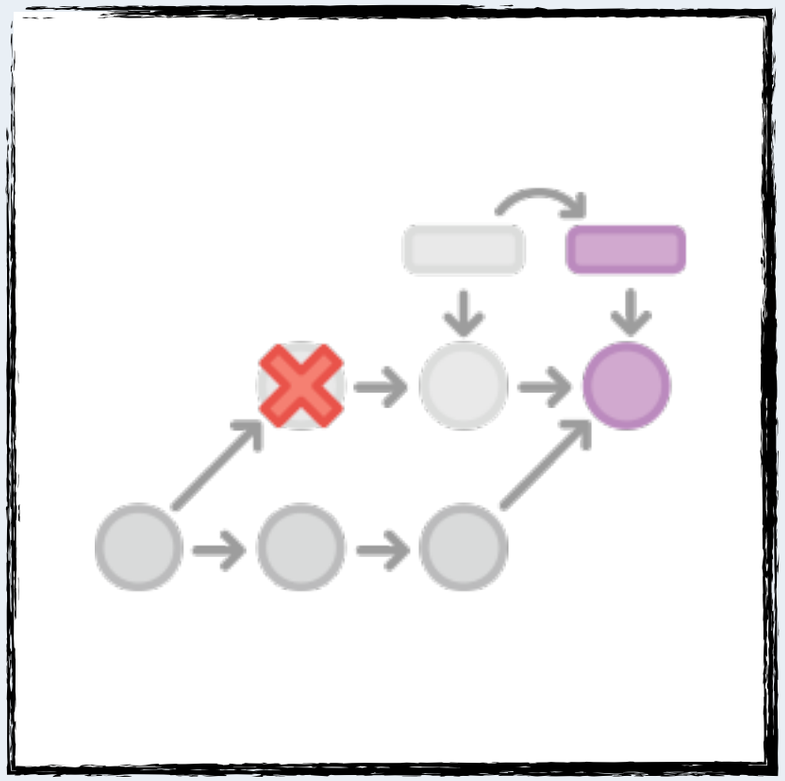
Rebasing lets you move branches around, which helps you avoid unnecessary merge commits. The resulting linear history is often much easier to understand and explore.



```
git rebase <base>
```

git rebase -i

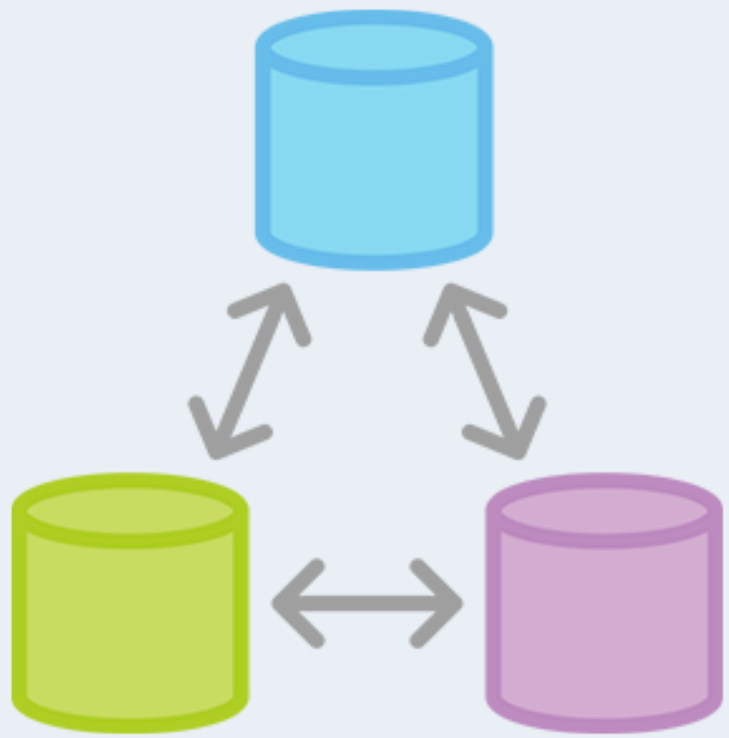
The -i flag is used to begin an interactive rebasing session. This provides all benefits of a normal rebase, but gives you the opportunity to add, edit, or delete commits along the way.



```
git rebase -i <base>
```

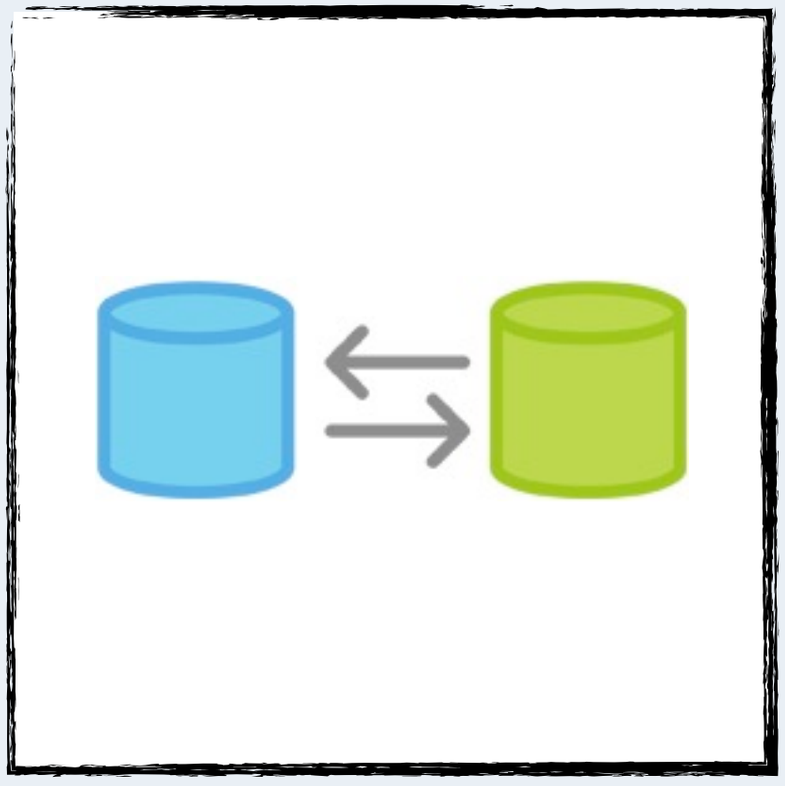
Remote Repositories

Remote Git repositories



git remote

The git remote command is a convenient tool for administering remote connections. Instead of passing the full URL to the fetch, pull, and push commands, it lets you use a more meaningful shortcut.



```
git remote <options>
```

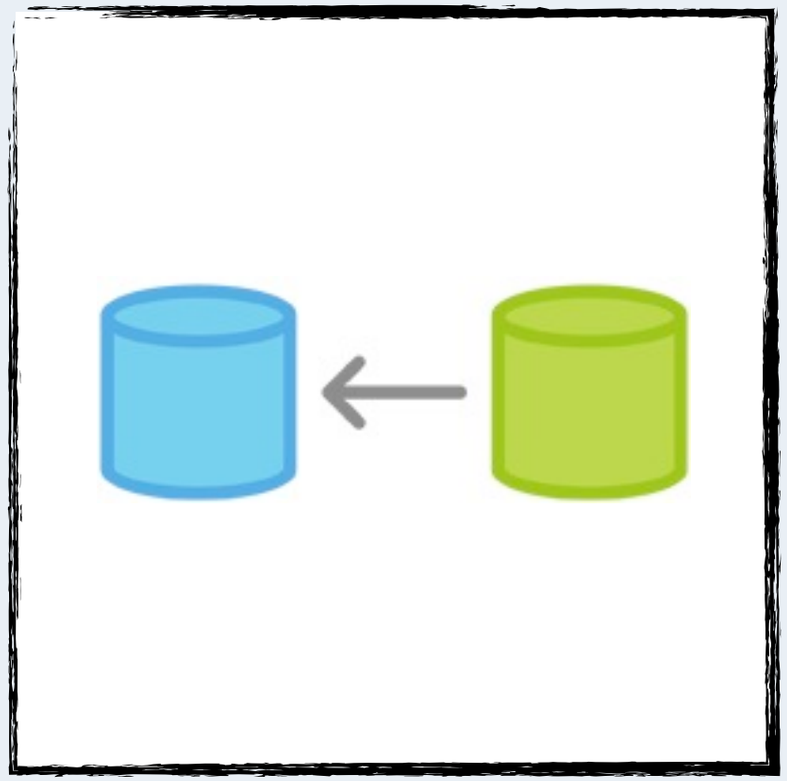
git fetch

Fetching downloads a branch from another repository, along with all its associated commits and files. But, it doesn't try to integrate anything into your local repository. This gives you the chance to inspect changes before merging them with your project.

```
git fetch <remote> <branch>
```

git pull

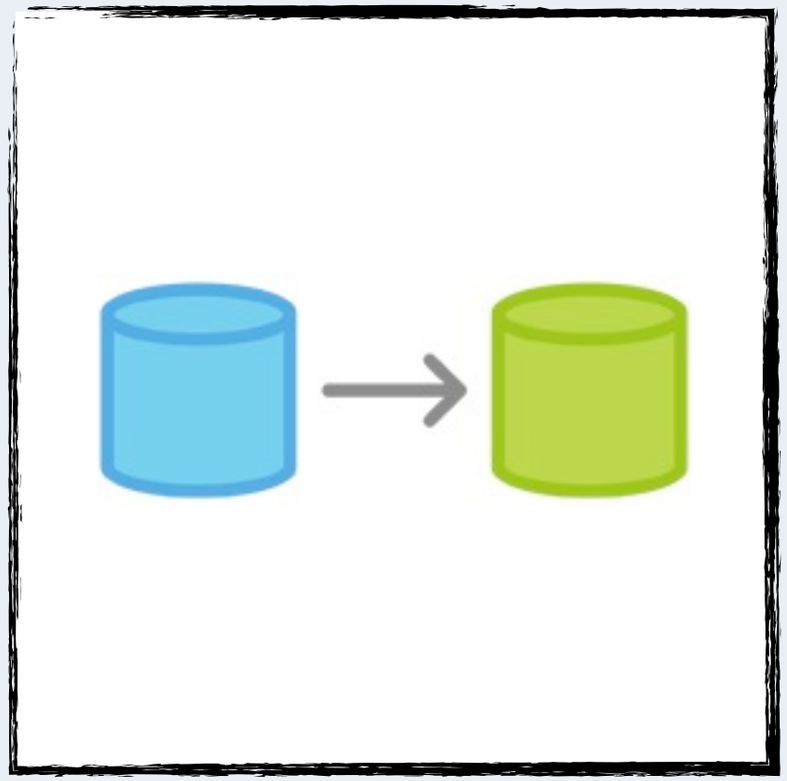
Pulling is the automated version of git fetch. It downloads a branch from a remote repository, then immediately merges it into the current branch.



```
git pull <options> <remote>
```

git push

Pushing is the opposite of fetching (with a few caveats). It lets you move a local branch to another repository, which serves as a convenient way to publish contributions.

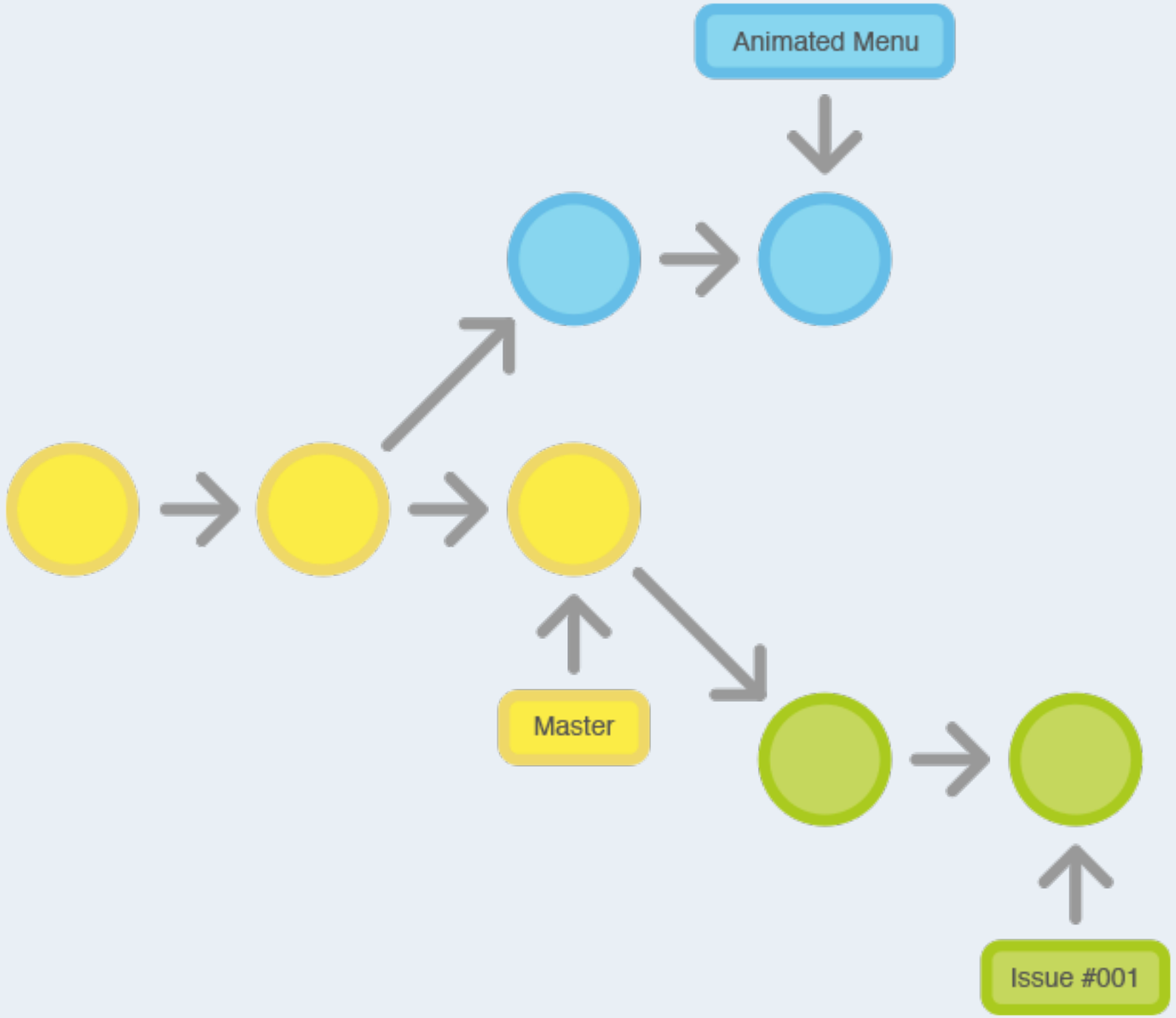


```
git push <remote> <branch>
```

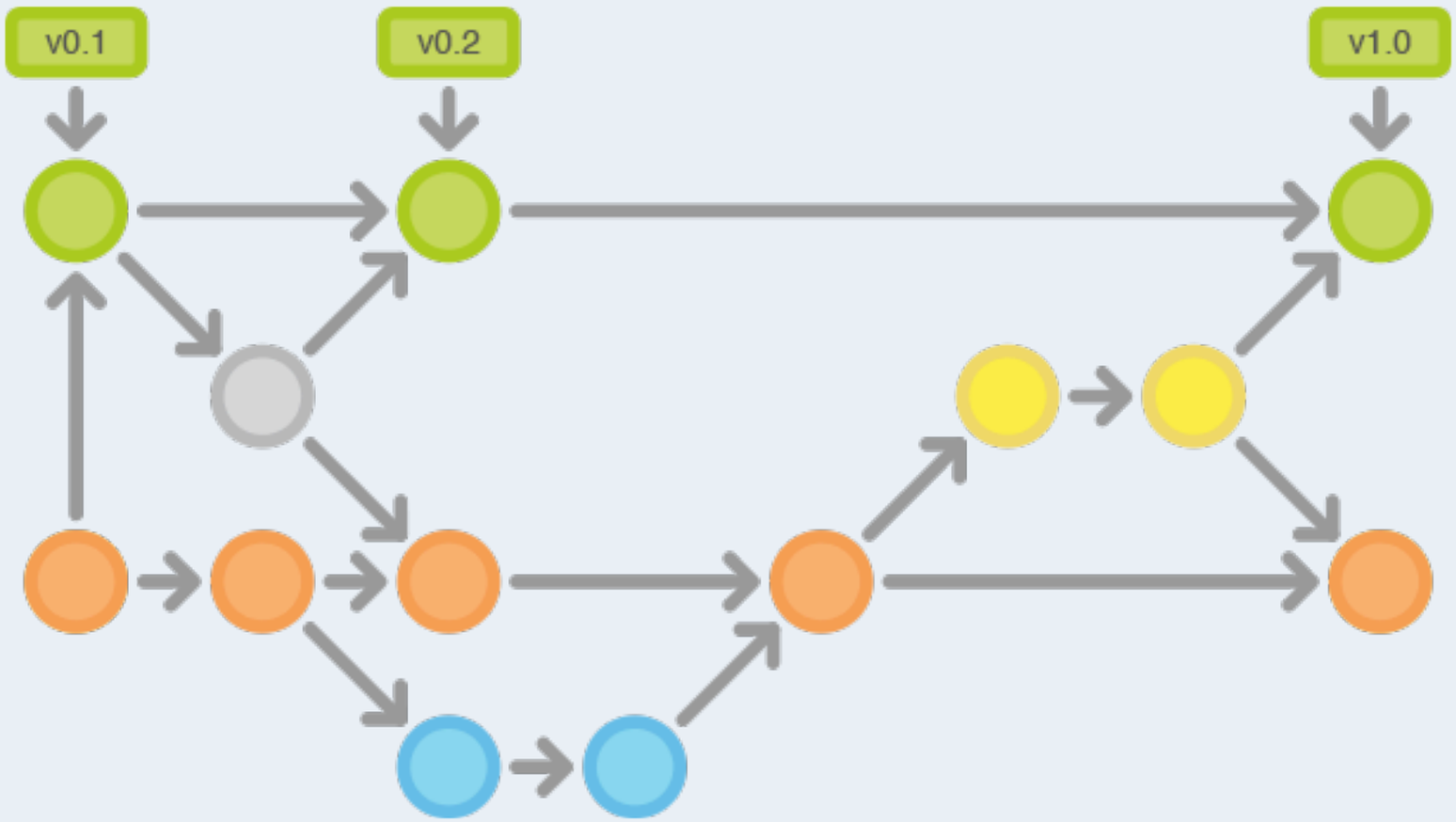
Git Workflow

Centralised Workflow

Feature Branch



GitFlow



Forking