



Rachel M. Carmena

## How to teach Git




Published: 12 December 2018

Last updated: 10 August 2021

### The problem I found

Some of my professional experiences have involved participating in cross-functional areas, so I knew all my colleagues' way of working. I remember a company which just started using Git a few weeks before I joined.

I found post-its on screens with 3 steps: first add, second commit, third push.



1°. add  
2°. commit  
3°. push

They didn't know the reason for those steps. They only knew that they should follow them in order not to get into trouble. However, problems happened frequently, so I decided to prepare a workshop about Git.

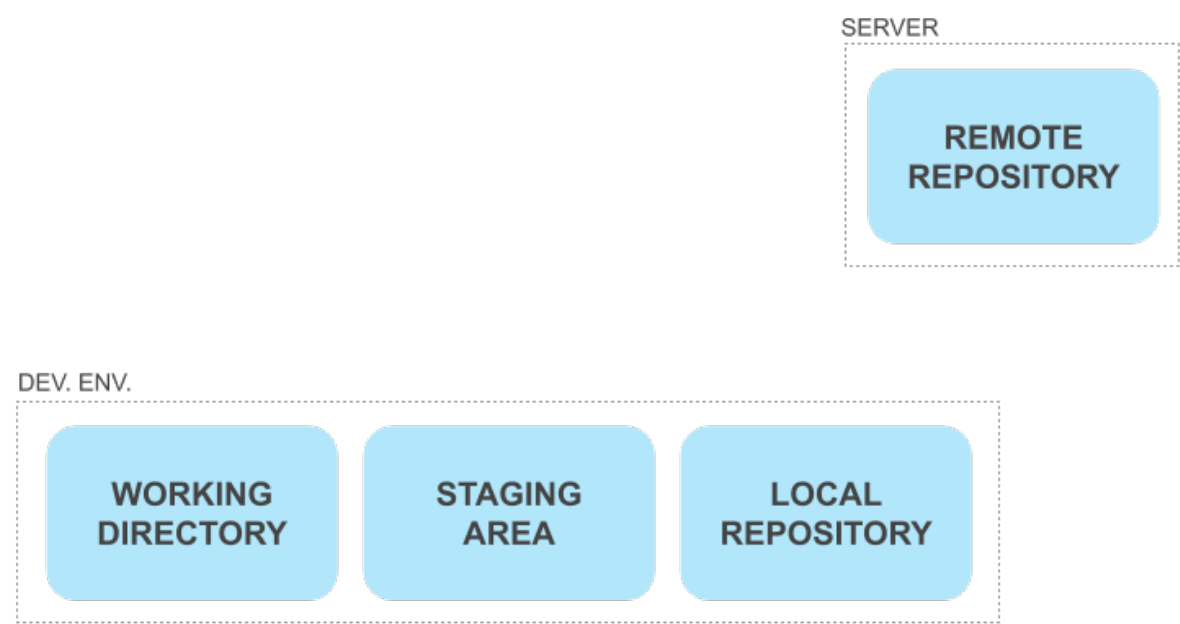
# The idea

I love to have maps in my mind. I don't write "mind maps" because they are a well-known type of diagrams. Now I'm talking about having frames, structures or any kind of graphical representation in the mind. For example, I started learning addition by imagining dice in my mind.

So I prepared some drawings. It's not necessary to be able to see the drawings to understand this post. I include an explanation for each of them because of my awareness of accessibility.

Furthermore, in this case, it's very important to teach the vocabulary. Otherwise, they won't understand the Git messages. The drawings are a good way to introduce them to that vocabulary.

## A distributed version control system



The general drawing contains 4 areas distributed as follows.

A **development environment** with:

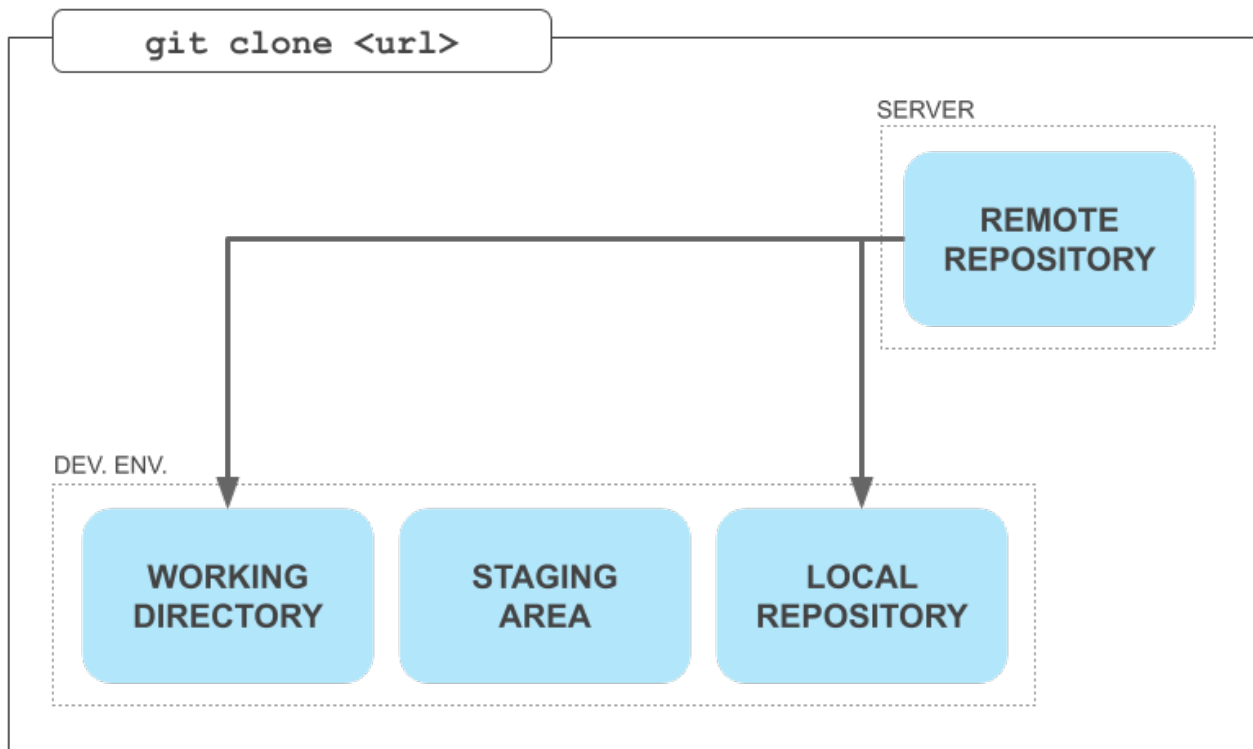
- Working directory
- Staging area or index
- Local repository

A **server** with:

- Remote repository

At that time, you can explain the benefits of a distributed version control system.

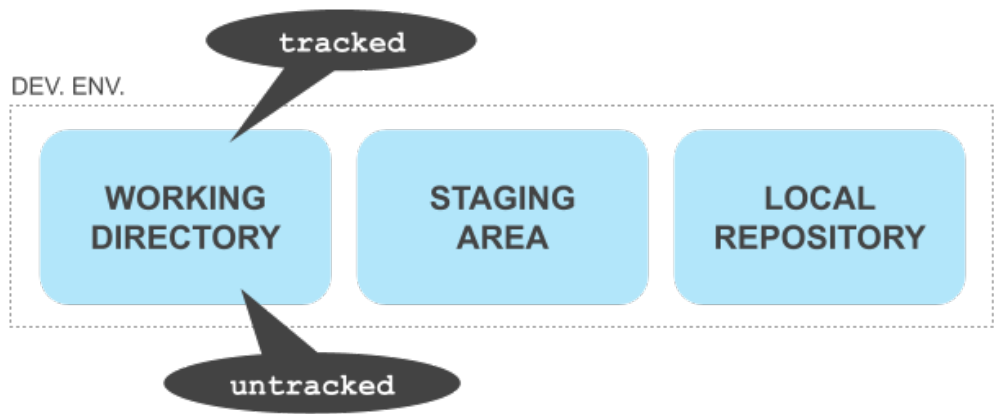
## Cloning a repository



When cloning a repository, the data from the remote repository travel to 2 areas:

- Working directory
- Local repository

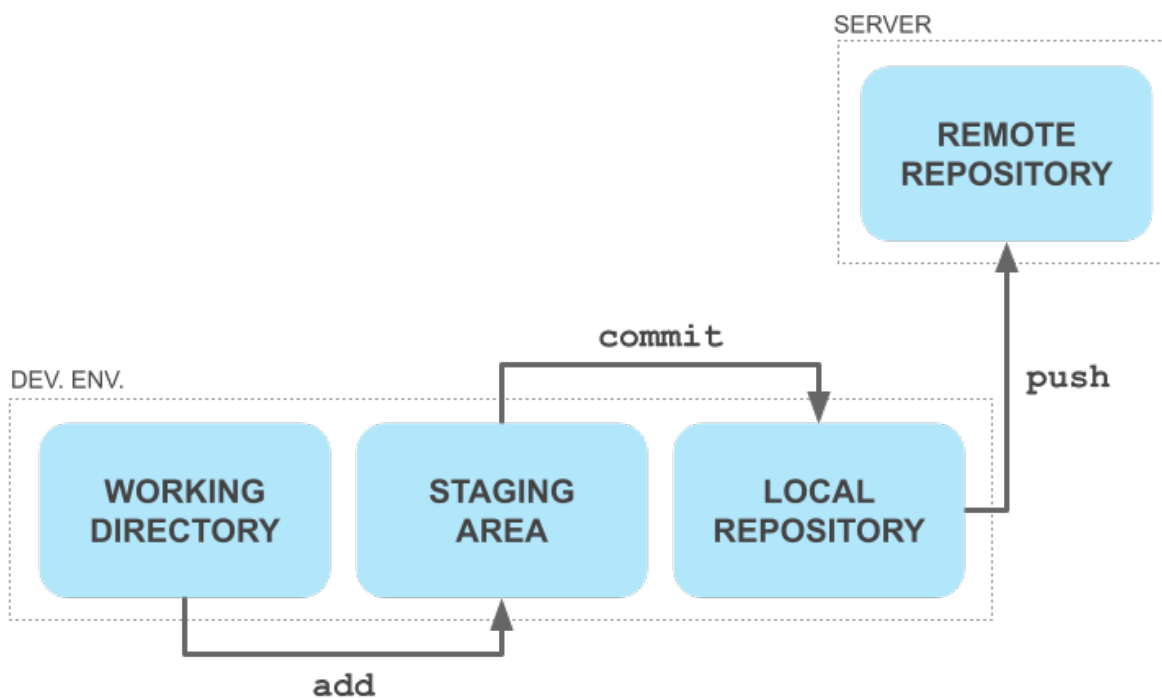
## Making changes in the working directory



There are 2 types of files in the working directory:

- **Tracked:** Files that are known by Git.
- **Untracked:** Files that have not yet been added, so they aren't known by Git.

## Updating the remote repository

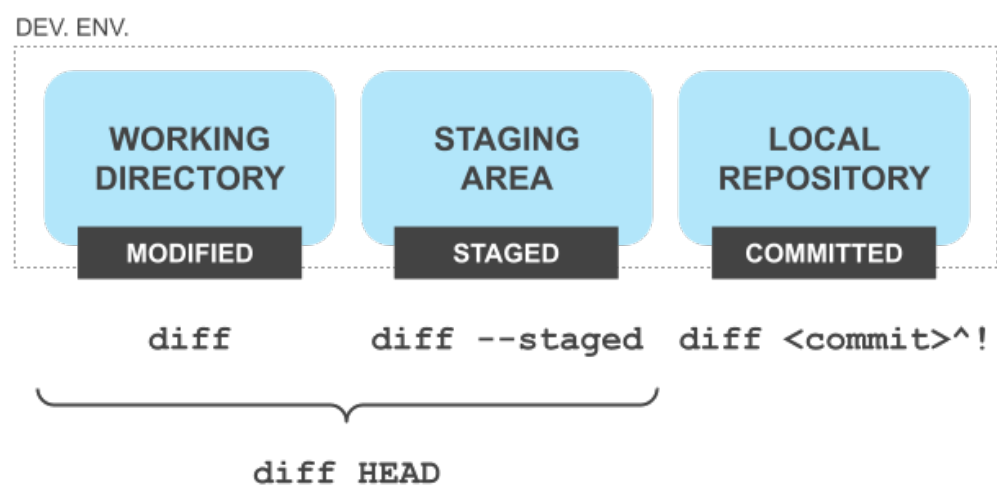


As soon as changes are ready in the **working directory**, they must be added in the **staging area**.

When there is a set of changes with a single purpose in the **staging area**, it's the time to create a commit with a message about that purpose in the **local repository**.

When there is one or several commits in the **local repository** that are ready to be shared with the rest of the world, they must be pushed to the **remote repository**.

At that time, you can talk about the different states of a file in the development environment: **modified**, **staged** and **committed**.

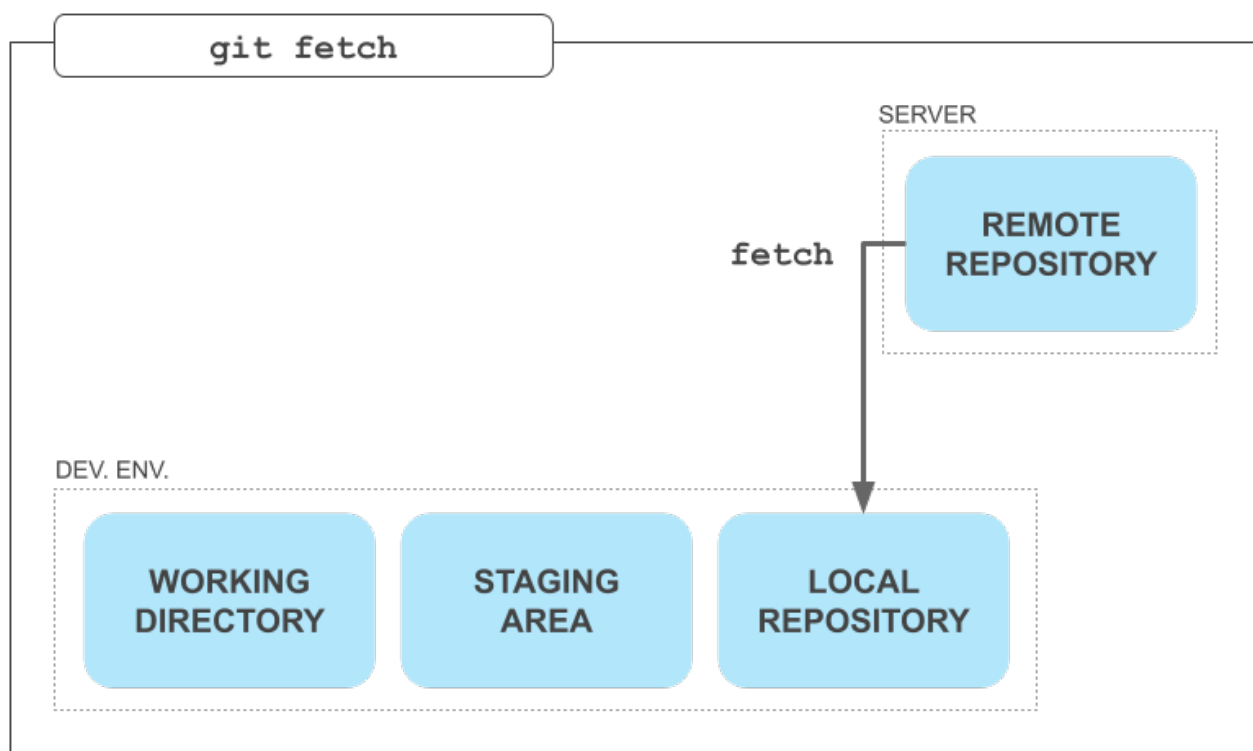


Furthermore, you can explain:

- How to show the changes in the **working directory**: `git diff`.
- How to show the changes in the **staging area**: `git diff --staged`.
- How to show the changes in the **working directory** and the **staging area**: `git diff HEAD`.
- How a file can be changed in the **working directory** after being added to the **staging area**.
- etc.

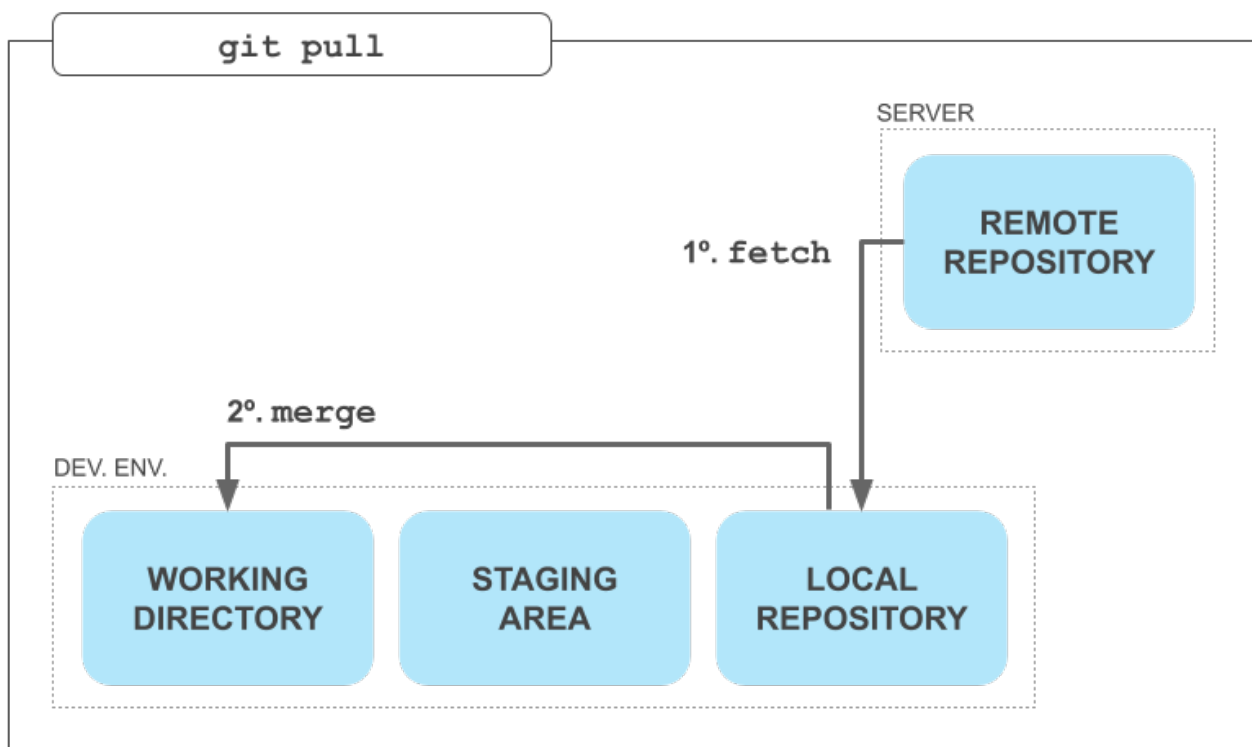
## Updating the development environment

### Fetching



When executing `git fetch`, the data from **remote repository** only travel to the **local repository**.

## Pulling



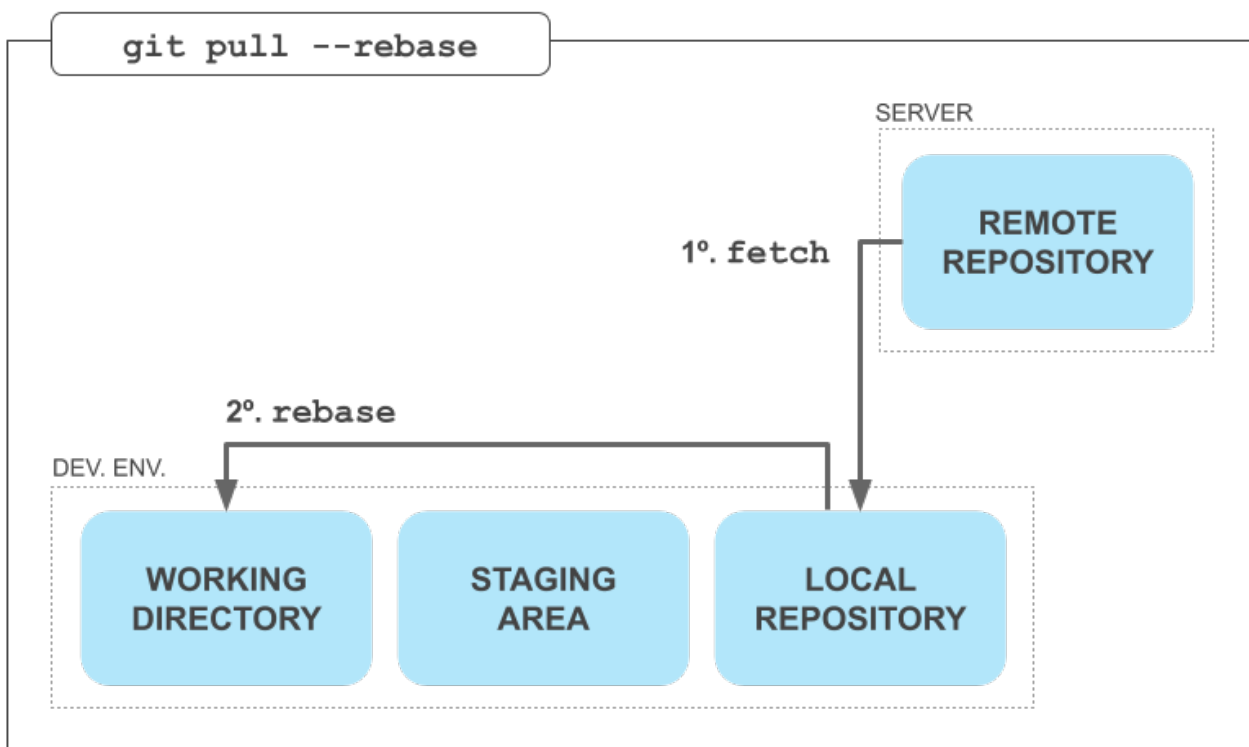
When executing `git pull`, the data from **remote repository** travel to two areas:

- To **local repository**: `fetch`.
- To **working directory**: `merge`.

If you take care of the commit history, consider the use of `git pull --rebase`.

Instead of `fetch + merge`, it consists of `fetch + rebase`.

Your local commits will be replayed and you won't see the known *diamond shape* in commit history.



## Next steps

You can add another area in the development environment to explain stashing: **dirty working directory**.

If people internalize these concepts, it will be easier for you to go a step further with branches, commit history, rebasing, etc. because you will have built a solid basis.

## Friendly reminder

I've worked with other version control systems (Visual SourceSafe, TFS and Subversion) and, in my humble experience, a lack of knowledge can be harmful with both an old tool and a new one. Don't only focus on choosing a tool, but also on mastering it.

## Further reading

- [Pro Git book](#)

## Received feedback

My friend [Marc Villagrasa](#) reminds me that he found it very useful to solve [Git challenges](#) and to share the solutions among colleagues.

Resources from [comments at Hacker News](#):

- [Linus Torvalds' greatest invention](#)
- [Tech Talk: Linus Torvalds on git](#)
- [Linux.conf.au 2013 - Git For Ages 4 And Up](#)
- [Git From the Bottom Up](#)
- [Git From the Bottom Up \(PDF\)](#)
- [Learn Git Branching](#)
- [Flight rules for git](#)
- [Emacs package: Magit](#)
- [How to write a Good Commit Message](#)
- [Become a git guru](#)
- [Git Immersion](#)
- [Udacity: How to Use Git and GitHub](#)

After reading more [comments at Reddit](#), I think that a more accurate title for this post would be **An idea to teach Git**, because it's only an idea that appeared in my mind when learning Git by myself a few years ago with [Pro Git book](#). This post is not a complete guide, only a starting point for trainers. I'm sure all of these resources will be very useful as well. Thanks!

And thanks [Stuart Maxwell](#) who shared this post at Hacker News and [u/cryptoz](#) who shared it at Reddit!

Resources from comments at Twitter:

- [This post explained in Japanese](#)
- [How to explain git in simple words?](#)
- [Learn Git](#)

Other related things:

- [Better documentation through commit messages](#) by Josh Kelley

## Bonus

This article inspired [Nico Riedmann](#) to create this awesome post: [Learn git concepts, not commands](#).

[« How to make images accessible](#) [Test-driven programming workflows](#) »



This blog by [Rachel M. Carmena](#) is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

Based on a work at <https://github.com/rachelcarmena/rachelcarmena.github.io>.

PERSONAL ADDENDUM: she also appreciates receiving the gathered feedback when using the included content in order to improve it and to grow together.

✉ [Send a message](#)