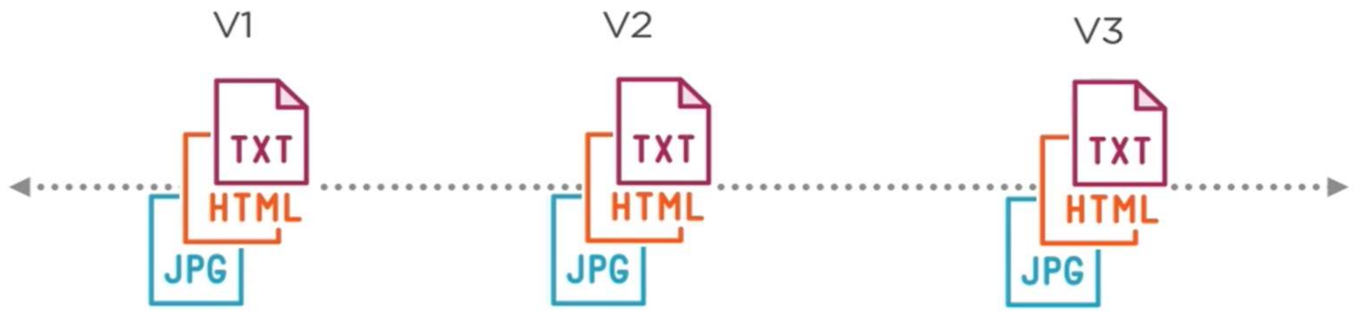
Getting Started with Git

git `--everything-is-local`

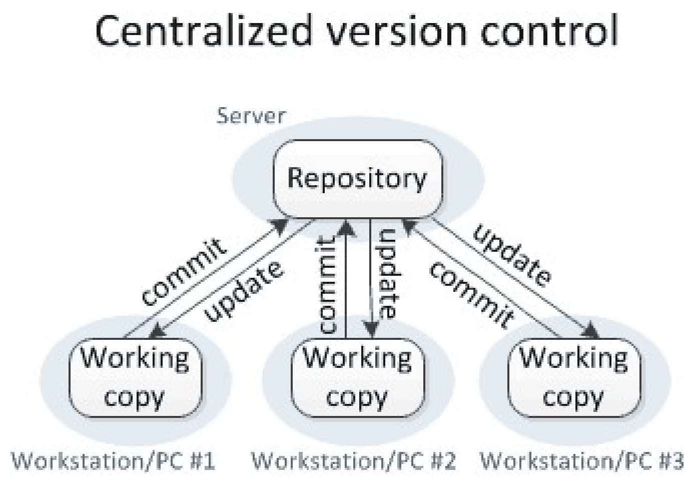
What is Git?

Version Control System

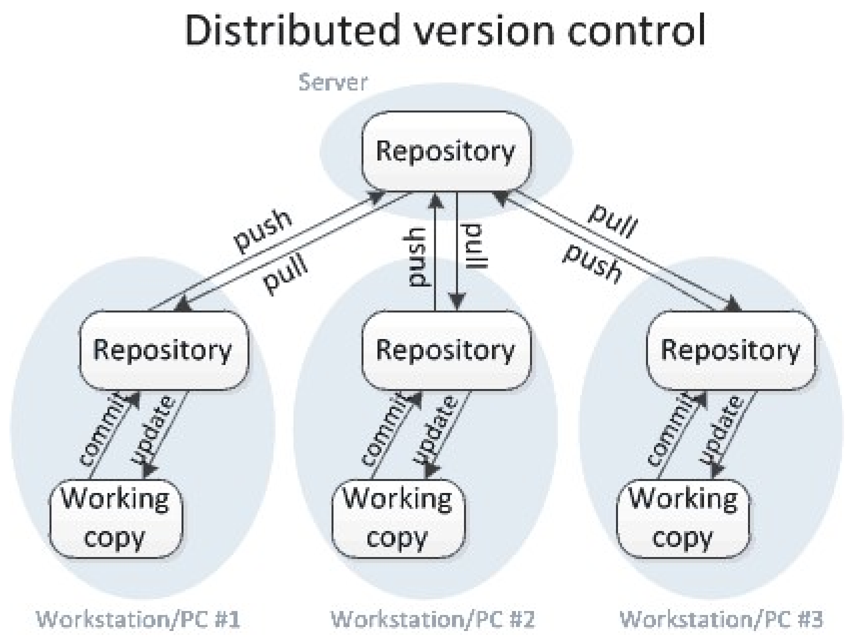
- Software designed to record changes made to files over time
- Ability to revert back to a previous file version or project version
- Compare changes made to files from one version to another
- Allows different team members manipulate the same files
- Makes easy finding bugs



Types of VCS

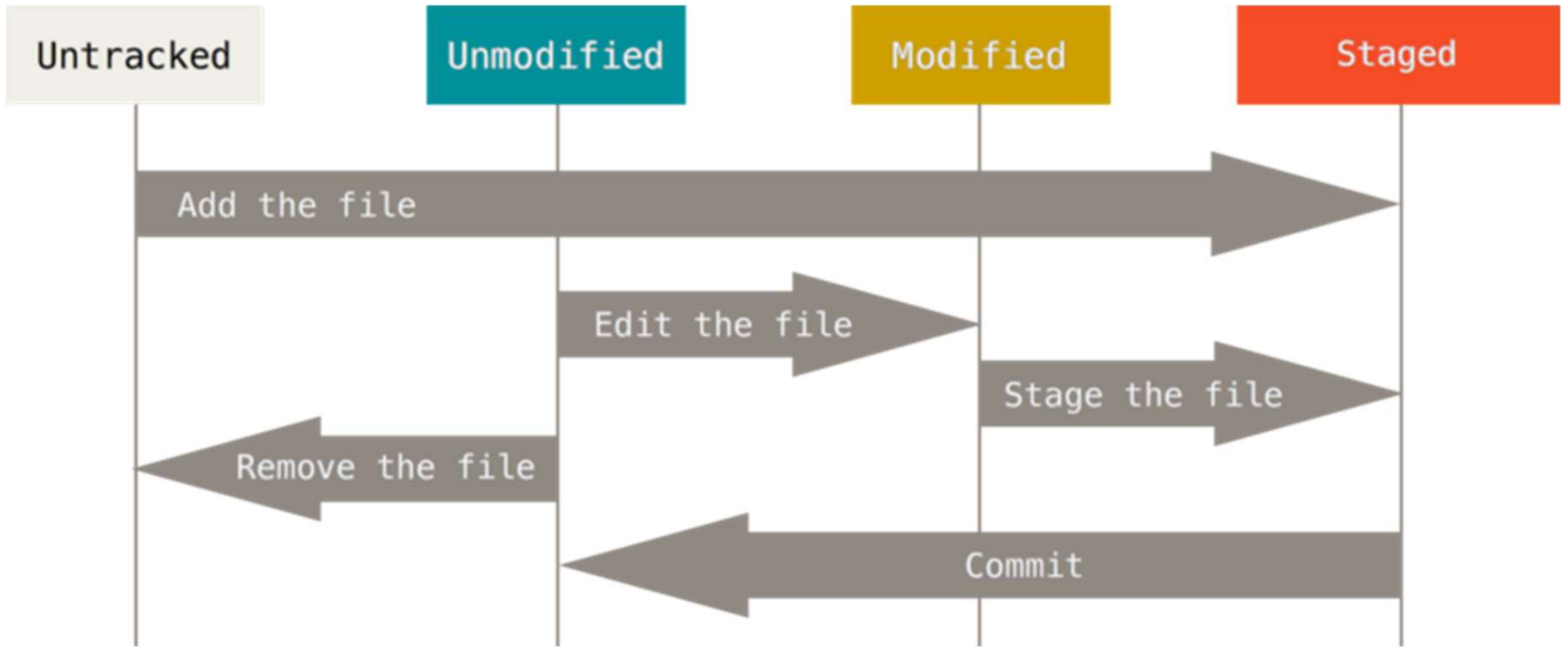


CVS , Subversion (SVN)

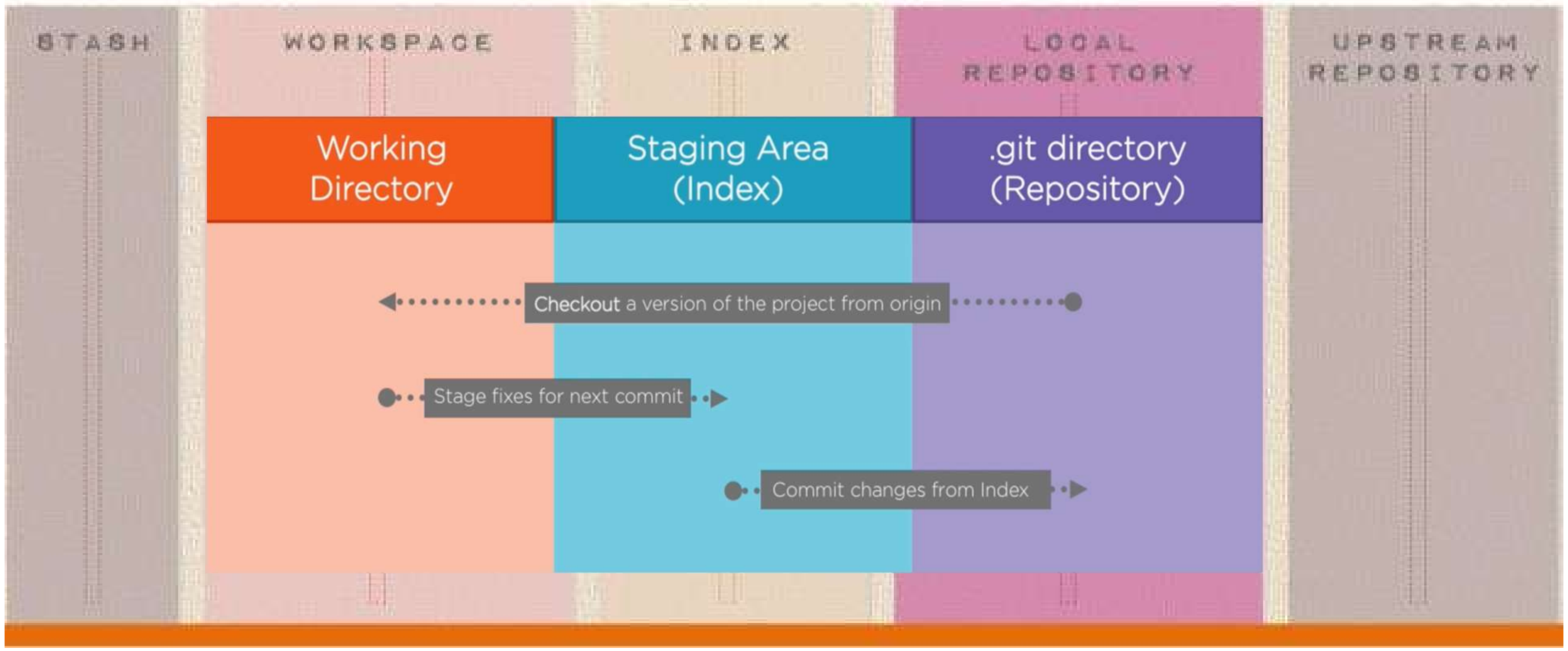


Mercurial,Git

The Stages of a File



The areas of a Git Project


STASH
WORKSPACE
INDEX
LOCAL REPOSITORY
UPSTREAM REPOSITORY
Working Directory
Staging Area (Index)
.git directory (Repository)
Checkout a version of the project from origin
Stage fixes for next commit
Commit changes from Index

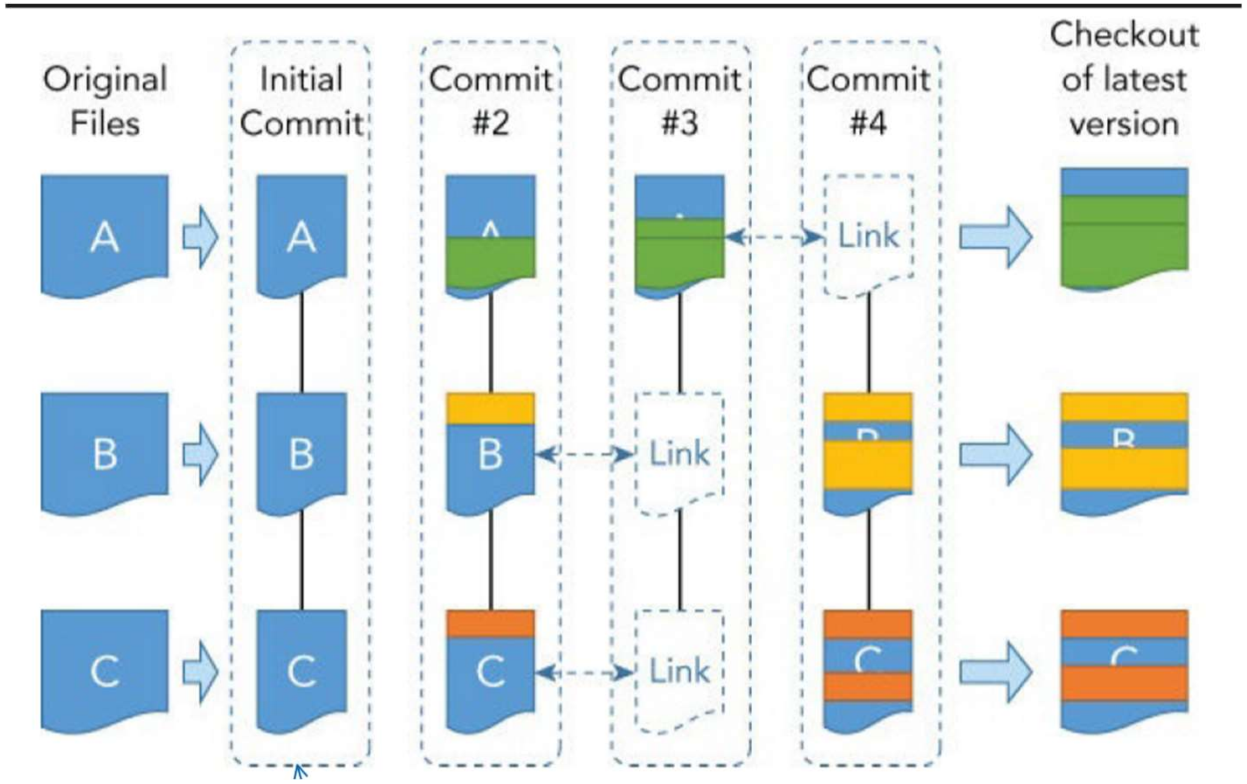
The Two Questions...

for understand any git command

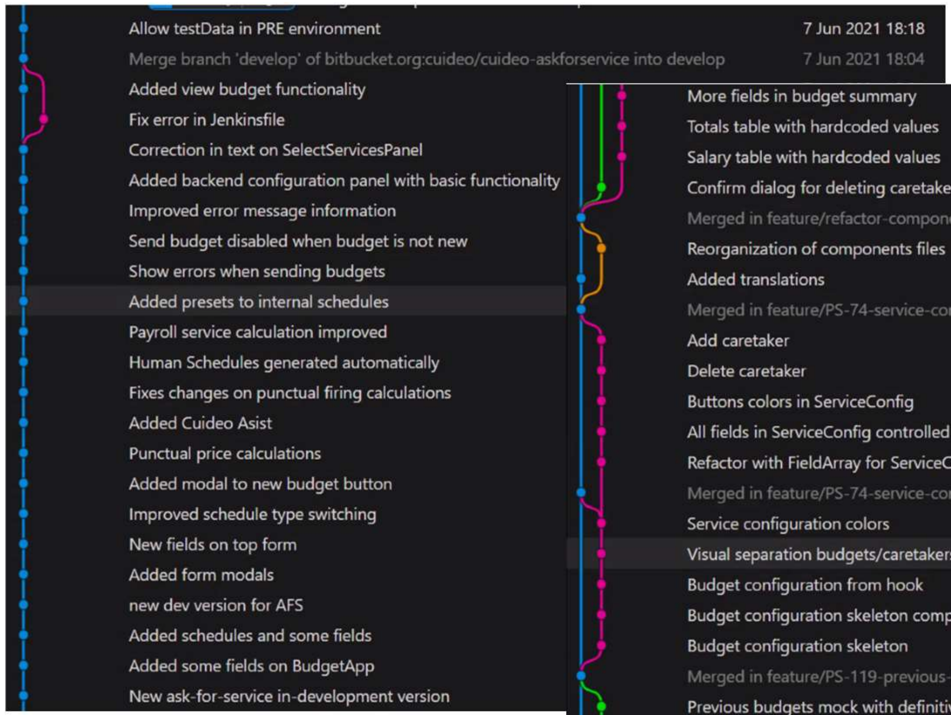
How does this command move information across the Four Areas?

How does this command change the Repository?

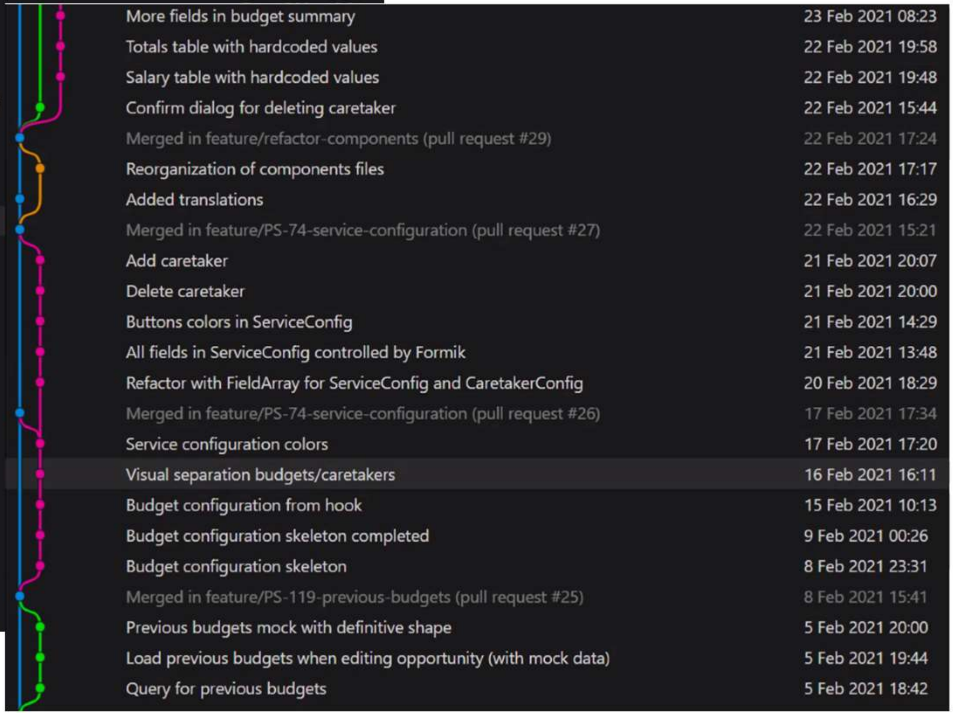
Commits


Original Files
Initial Commit
Commit #2
Commit #3
Commit #4
Checkout of latest version
A
B
C
Link
Link
Link

Logs: history of commits

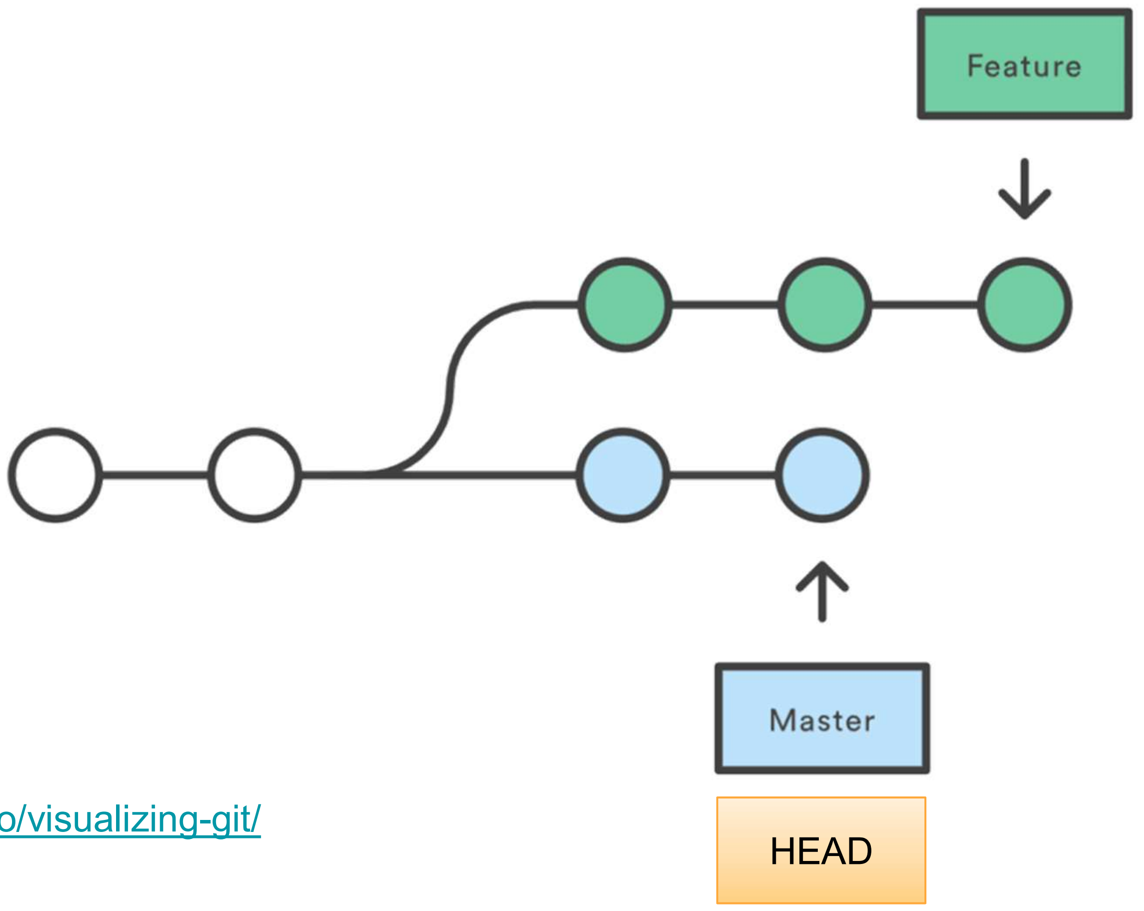


```
Allow testData in PRE environment                                              7 Jun 2021 18:18
Merge branch 'develop' of bitbucket.org:cuideo/cuideo-askforservice into develop  7 Jun 2021 18:04
Added view budget functionality
Fix error in Jenkinsfile
Correction in text on SelectServicesPanel
Added backend configuration panel with basic functionality
Improved error message information
Send budget disabled when budget is not new
Show errors when sending budgets
Added presets to internal schedules
Payroll service calculation improved
Human Schedules generated automatically
Fixes changes on punctual firing calculations
Added Cuideo Asist
Punctual price calculations
Added modal to new budget button
Improved schedule type switching
New fields on top form
Added form modals
new dev version for AFS
Added schedules and some fields
Added some fields on BudgetApp
New ask-for-service in-development version
```



```
More fields in budget summary                                          23 Feb 2021 08:23
Totals table with hardcoded values                                     22 Feb 2021 19:58
Salary table with hardcoded values                                     22 Feb 2021 19:48
Confirm dialog for deleting caretaker                                  22 Feb 2021 15:44
Merged in feature/refactor-components (pull request #29)               22 Feb 2021 17:24
Reorganization of components files                                     22 Feb 2021 17:17
Added translations                                                     22 Feb 2021 16:29
Merged in feature/PS-74-service-configuration (pull request #27)       22 Feb 2021 15:21
Add caretaker                                                          21 Feb 2021 20:07
Delete caretaker                                                       21 Feb 2021 20:00
Buttons colors in ServiceConfig                                        21 Feb 2021 14:29
All fields in ServiceConfig controlled by Formik                       21 Feb 2021 13:48
Refactor with FieldArray for ServiceConfig and CaretakerConfig         20 Feb 2021 18:29
Merged in feature/PS-74-service-configuration (pull request #26)       17 Feb 2021 17:34
Service configuration colors                                           17 Feb 2021 17:20
Visual separation budgets/caretakers                                   16 Feb 2021 16:11
Budget configuration from hook                                         15 Feb 2021 10:13
Budget configuration skeleton completed                                9 Feb 2021 00:26
Budget configuration skeleton                                          8 Feb 2021 23:31
Merged in feature/PS-119-previous-budgets (pull request #25)           8 Feb 2021 15:41
Previous budgets mock with definitive shape                            5 Feb 2021 20:00
Load previous budgets when editing opportunity (with mock data)        5 Feb 2021 19:44
Query for previous budgets                                             5 Feb 2021 18:42
```

Branches



http://git-school.github.io/visualizing-git/

HEAD

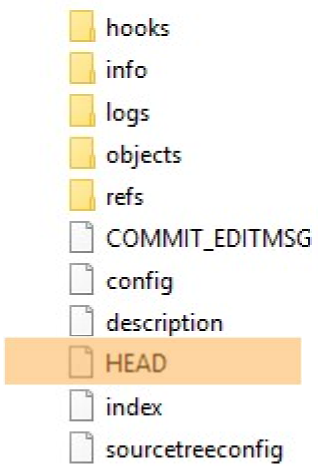
bash-shell

```
$ cd .git
$ cat HEAD -> ref: refs/heads/master
$ git checkout -b newFeature
$ cat HEAD -> ref: refs/heads/newFeature
```

Windows

```
cd .git
type HEAD -> ref: refs/heads/master
git checkout -b newFeature
type HEAD -> ref: refs/heads/newFeature
```

.git



```
hooks
info
logs
objects
refs
COMMIT_EDITMSG
config
description
HEAD
index
sourcetreeconfig
```

Install Git on Linux

Debian

```
sudo apt-get install git
```

Fedora

```
sudo yum install git
```

Install Git on Windows

https://git-scm.com/download/win

Install Git on Mac

https://git-scm.com/download/mac

Homebrew

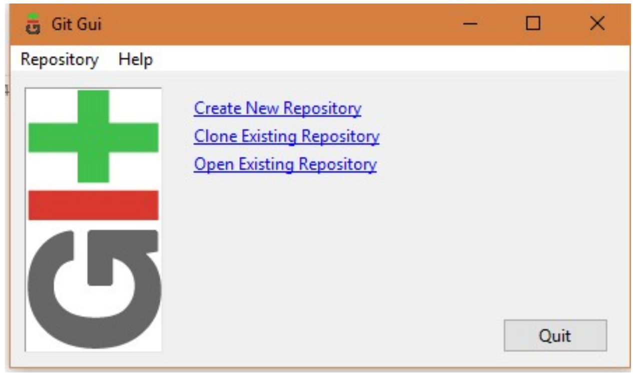
```
brew install git
```

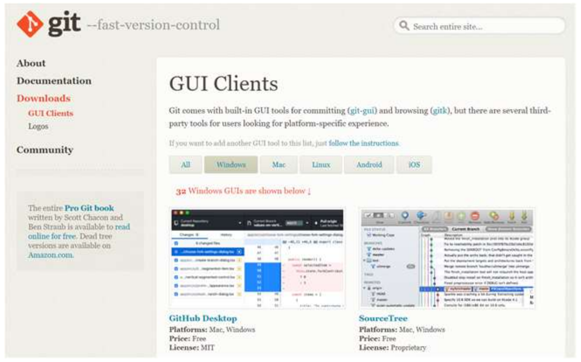
Command Line (CLI) v Graphic Interfaces

Run Commands
Type a specific command and then hit ENTER to execute

Windows
Command prompt or Powershell

Mac and Linux
Terminal





GUI Clients

Git comes with built-in GUI tools for committing (git-gui) and browsing (gitk), but there are several third-party tools for users looking for platform-specific experience.

If you want to add another GUI tool to this list, just follow the instructions.

32 Windows GUIs are shown below ↓

The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

GitHub Desktop
Platforms: Mac, Windows
Price: Free
License: MIT

SourceTree
Platforms: Mac, Windows
Price: Free
License: Proprietary

https://git-scm.com/download/gui/windows

Using the Command Line

pwd	Print working directory
cd	Change working directory (cd .. or cd ~)
ls / dir	List files in a directory (dir for windows users)
mkdir	Create a new empty folder

Command Line (CLI)

CLI gives us more control than GUI

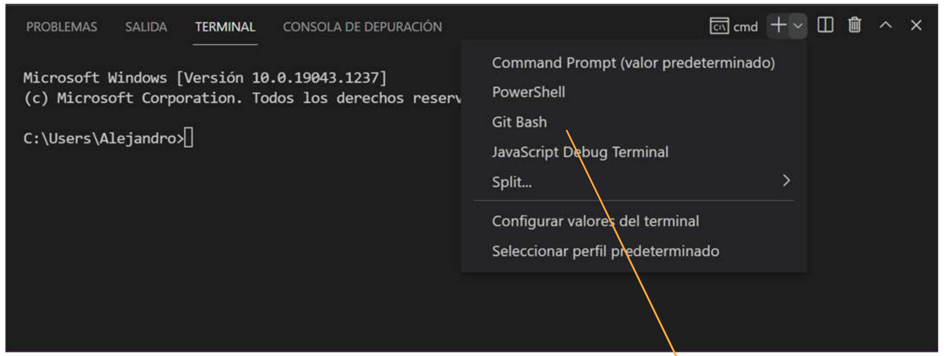
CLI commands

- All commands start with *git*
- Getting help:

```
git <command> --help
git config --help
git status --help
```

```
git --version
```

CLI en VSC

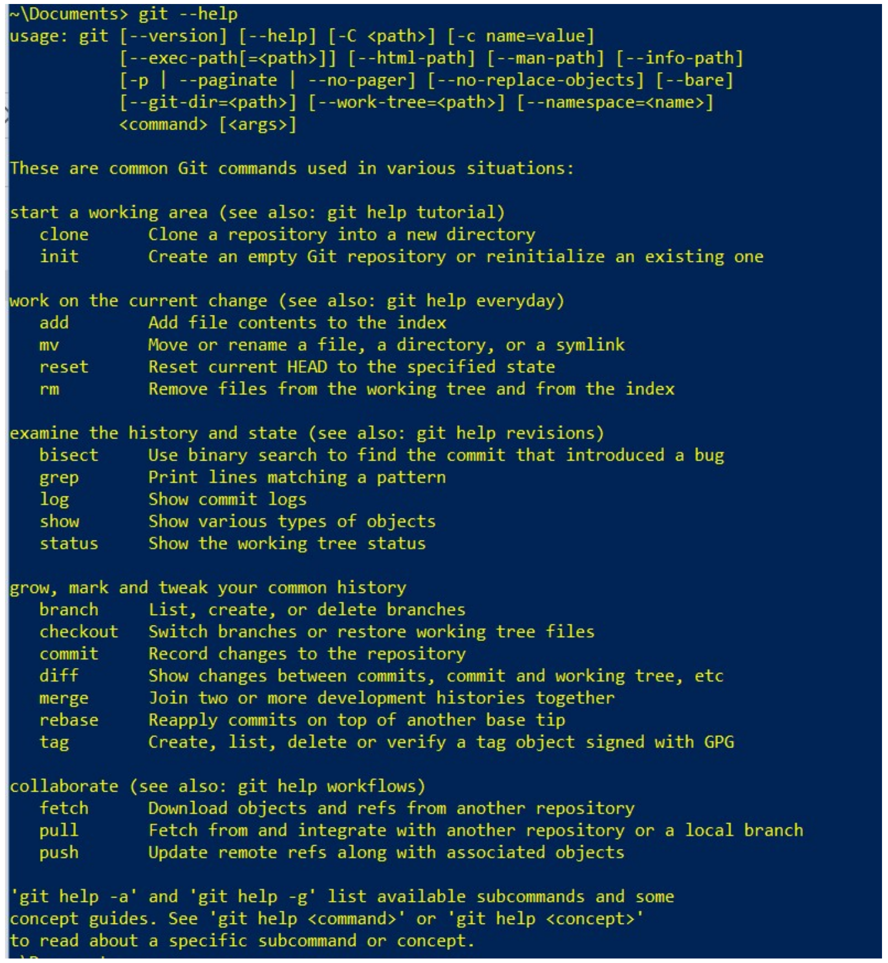

PROBLEMAS
SALIDA
TERMINAL
CONSOLA DE DEPURACIÓN
cmd
Command Prompt (valor predeterminado)
PowerShell
Git Bash
JavaScript Debug Terminal
Split...
Configurar valores del terminal
Seleccionar perfil predeterminado

```
Microsoft Windows [Versión 10.0.19043.1237]
(c) Microsoft Corporation. Todos los derechos reserv

C:\Users\Alejandro>
```

bash / gig bash
Autocomplete with tab

Git Help

```
man git

git help config

git help
```



```
~\Documents> git --help
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

Git Config

Git configuration

Configuration variables

- System (O.S.): **`git config --system`**
- Global (O.S. user): **`git config –global`**
- Local (project): **`git config [--local]`**

```
git config
git config --global user.name “Alejandro Cerezo”
git config --global user.email “alce65@hotmail.es”
git config –list [--show-origin]
```

Extra Configuration

- - Si no habéis instalado Git diciéndole que VSCode sea su editor por defecto, para configurarlo hay que lanzar este comando:
  - Mac:
    `git config --global core.editor "code” --wait`
  - Windows:
    `git config --global core.editor "<ruta del VSCode>\Code.exe" --wait`

- Otras configuraciones de Git:
  - `git config --global core.autocrlf false` <- controlado desde el .editorconfig
  - `git config --global core.ignorecase true` <- si estáis en WindowsA

- Install Git
- Configure Git
- Initialize a new Git project
- Use git

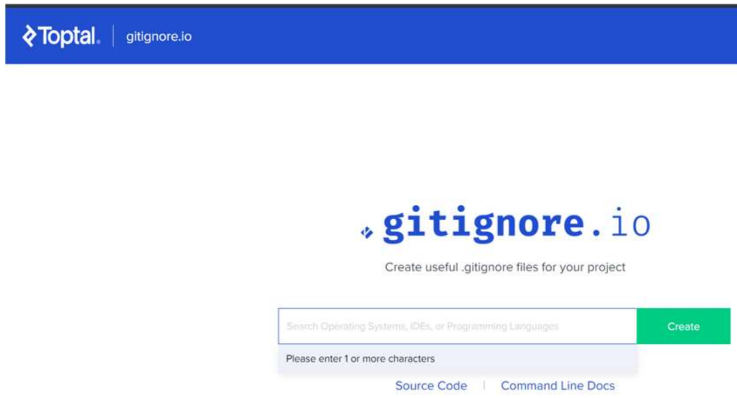
Initialize a New Git Repository

```
git init        // Create an empty Git repository or reinitialize an existing one
cd .git         // Enter Git configuration folder
ls              // List all the files in the current folder
cd ..           // Move to the parent folder
```

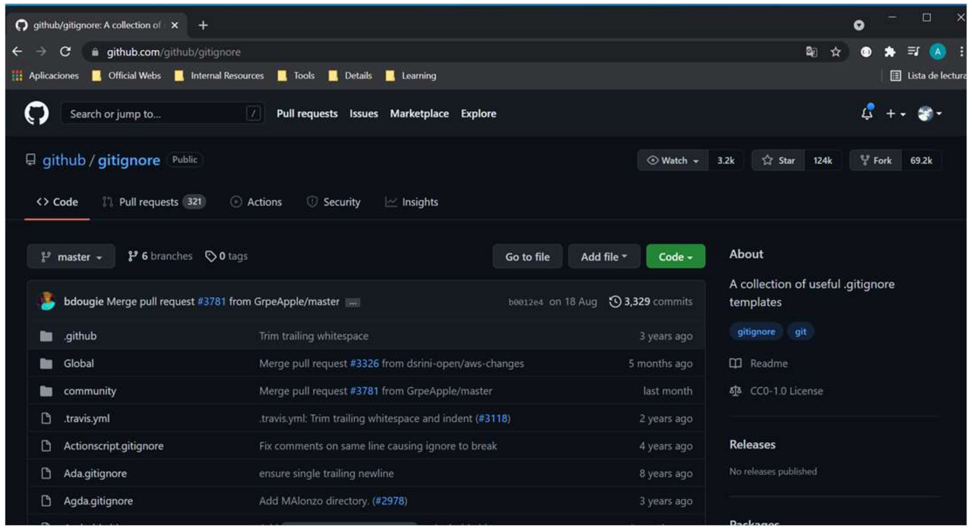
- Creates a *.git* directory
- The repository (commits and all the info that Git manages) lives inside *.git*
- We can opt out just deleting the *.git* directory

Files: .gitignore - README.md

.gitignore



Toptal
gitignore.io
gitignore.io
Create useful .gitignore files for your project
Search Operating Systems, IDEs, or Programming Languages
Create
Please enter 1 or more characters
Source Code
Command Line Docs



bdougie Merge pull request #3781 from GrpeApple/master … b0012e4 on 18 Aug 3,329 commits

.github	Trim trailing whitespace	3 years ago
Global	Merge pull request #3326 from dsrini-open/aws-changes	5 months ago
community	Merge pull request #3781 from GrpeApple/master	last month
.travis.yml	.travis.yml: Trim trailing whitespace and indent (#3118)	2 years ago
Actionscript.gitignore	Fix comments on same line causing ignore to break	4 years ago
Ada.gitignore	ensure single trailing newline	8 years ago
Agda.gitignore	Add MAlonzo directory. (#2978)	3 years ago

About

A collection of useful .gitignore templates

gitignore git

Readme

CC0-1.0 License

Releases

No releases published

https://git-scm.com/docs/gitignore
https://www.atlassian.com/git/tutorials/saving-changes/gitignore

Files Tracked By Git

Committed	Unmodified changes from the last commit snapshot
Modified	Changes made to files since last commit snapshot
Staged	Changes marked to be added into the next commit snapshot

git add / commit: Track a New File

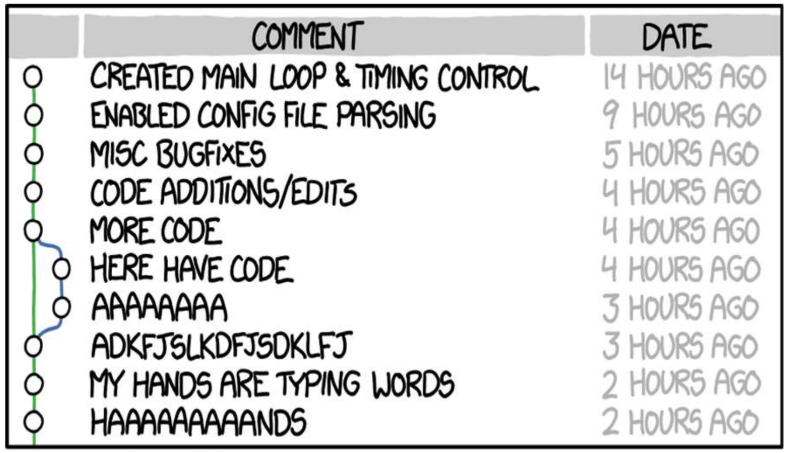
git add
git commit
git status

```
touch lessons.txt

git status

git add lessons.txt

git status

git commit´- m “Add Lessons File”
```

Commit Messages

Important things to remember:

- The message should describe the changes in a commit
- Avoid generic messages like "Changes", "New file", "Fixes", "CSS", "Merge"...
- Don't write too long messages (50 chars)
- Isolate single features or fixes in each commit (what not to do)



AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

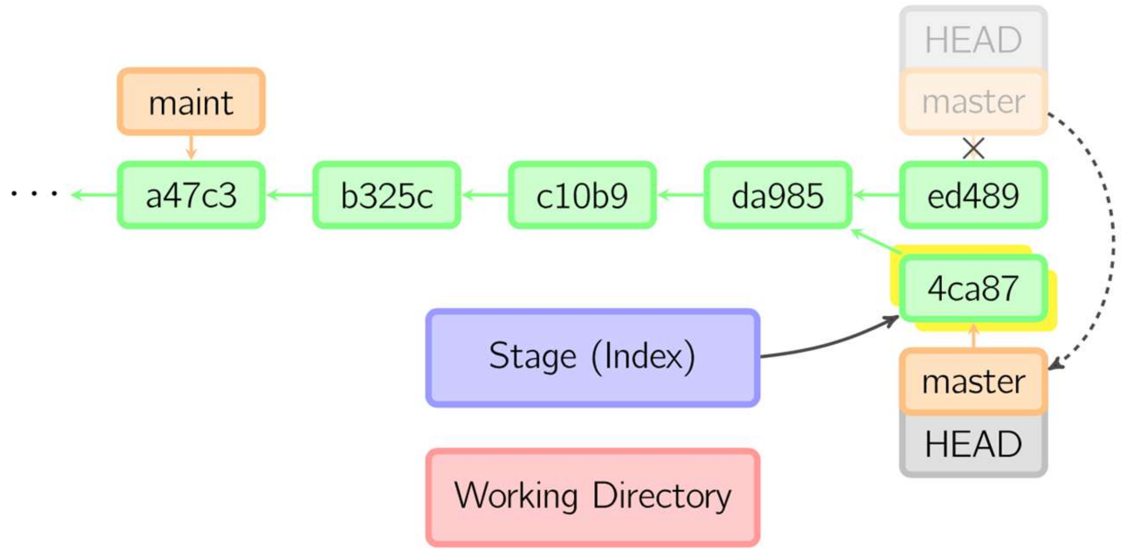
https://chris.beams.io/posts/git-commit/

Rules of a great Git commit message

- Separate subject from body with a blank line
- Limit the subject line to 50 characters
- Capitalize the subject line
- Do not end the subject line with a period
- Use the imperative mood in the subject line
- Wrap the body at 72 characters
- Use the body to explain what and why vs. how

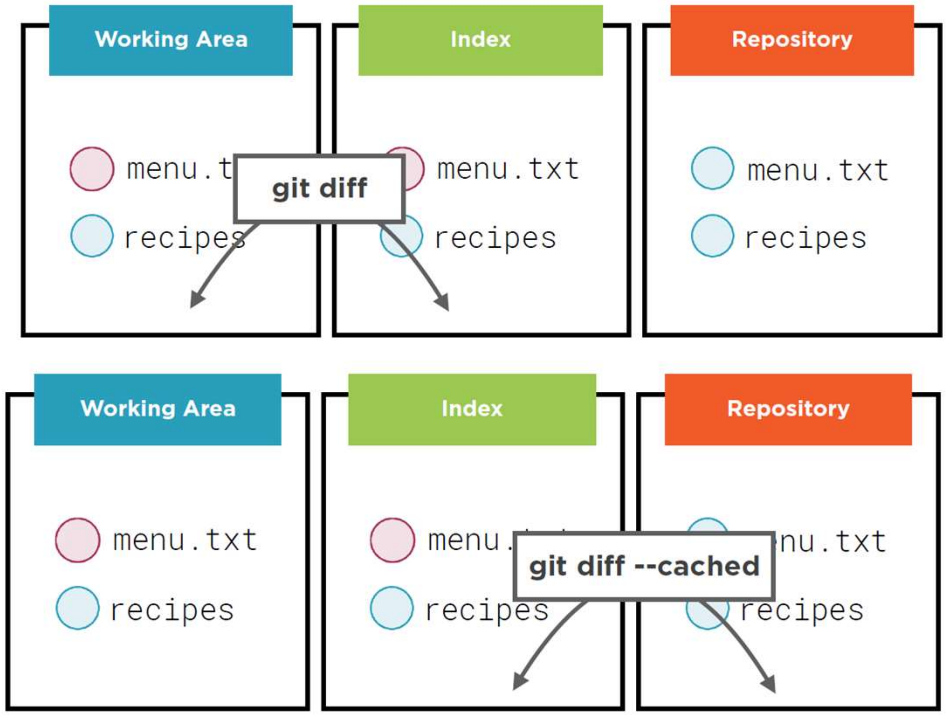
Modify last commit

```
git commit --amend
```



Last commit desapear / reapear with the added content

git diff


Working Area
Index
Repository
menu.txt
recipes
git diff
Working Area
Index
Repository
menu.txt
recipes
git diff --cached

Git Diff Explained

```
git diff --staged

diff --git a/file1.txt  b/file1.txt

Index 9863745..f30c839  100644

--- a/file 1.txt

+++ b/file1.txt

@@ -12, 2 +12, 3 @@

Example lines…

- Old content

+ New content
```

◀ Compared Files

◀ File Metadata

◀ Change Markers for File A/B

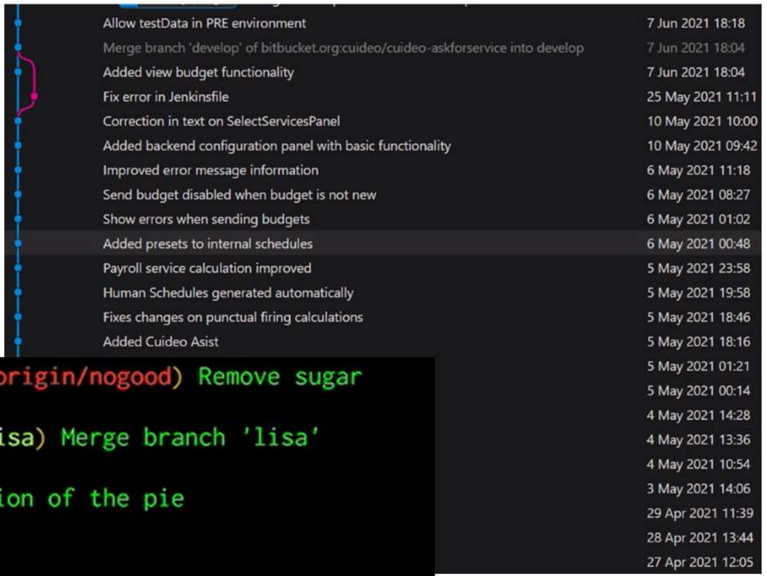
◀ Chunk Header

◀ Chunk Changes

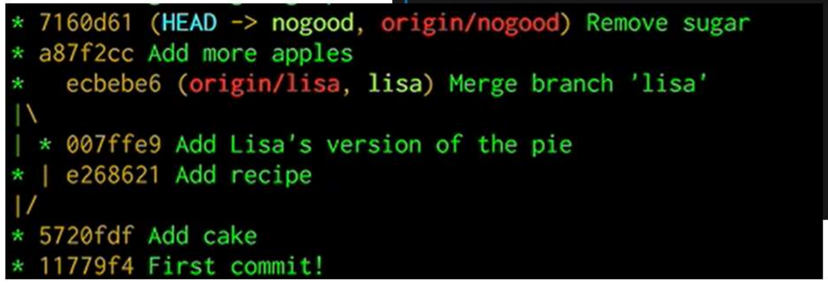
git log: Check Commit History

```
git log

git log -1

git log --oneline

git log --stat

git log --patch
```

VSC Git Graph



```
Allow testData in PRE environment                                          7 Jun 2021 18:18
Merge branch 'develop' of bitbucket.org:cuideo/cuideo-askforservice into develop   7 Jun 2021 18:04
Added view budget functionality                                            7 Jun 2021 18:04
Fix error in Jenkinsfile                                                   25 May 2021 11:11
Correction in text on SelectServicesPanel                                  10 May 2021 10:00
Added backend configuration panel with basic functionality                 10 May 2021 09:42
Improved error message information                                         6 May 2021 11:18
Send budget disabled when budget is not new                                6 May 2021 08:27
Show errors when sending budgets                                           6 May 2021 01:02
Added presets to internal schedules                                        6 May 2021 00:48
Payroll service calculation improved                                       5 May 2021 23:58
Human Schedules generated automatically                                    5 May 2021 19:58
Fixes changes on punctual firing calculations                              5 May 2021 18:46
Added Cuideo Asist                                                         5 May 2021 18:16
                                                                           5 May 2021 01:21
                                                                           5 May 2021 00:14
                                                                           4 May 2021 14:28
                                                                           4 May 2021 13:36
                                                                           4 May 2021 10:54
                                                                           3 May 2021 14:06
                                                                           29 Apr 2021 11:39
                                                                           28 Apr 2021 13:44
                                                                           27 Apr 2021 12:05
```



```
* 7160d61 (HEAD -> nogood, origin/nogood) Remove sugar
* a87f2cc Add more apples
*   ecbebe6 (origin/lisa, lisa) Merge branch 'lisa'
|\
| * 007ffe9 Add Lisa's version of the pie
* | e268621 Add recipe
|/
* 5720fdf Add cake
* 11779f4 First commit!
```

git log –graph –decorate --oneline

Aliases

alias.<name> <command>

Aliases by CLI

- System (O.S.): **`git config --system alias.co git commit`**
- Global (O.S. user): **`git config –global alias.co git commit`**
- Local (project): **`git config [--local] alias.co git commit`**

Aliases by editing files

- /programs/git/gitconfig
- $HOME/<urer>]/.gitconfig (specific for the user)
- .git//config (specific for the repository)

Use example

```
git config --global alias.hist
           git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short
```

Remove and Move Files

From the O.S.

From git

```
// Remove file
git rm <...file...>

// Remove from tracked files (unstaged the file.)
git rm --cached <...file...>

git reset HEAD <...file...>


// Rename files
git mv <...file...> <...newFile...>
```

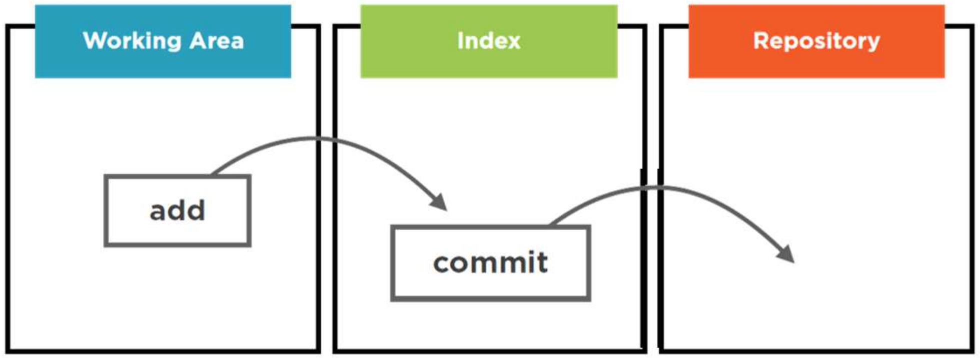
Time travel: git checkout

- In the repository, it **moves the HEAD** reference => change de repository: changes the current commit
  - generally, to another branch
  - to another commit in the branch

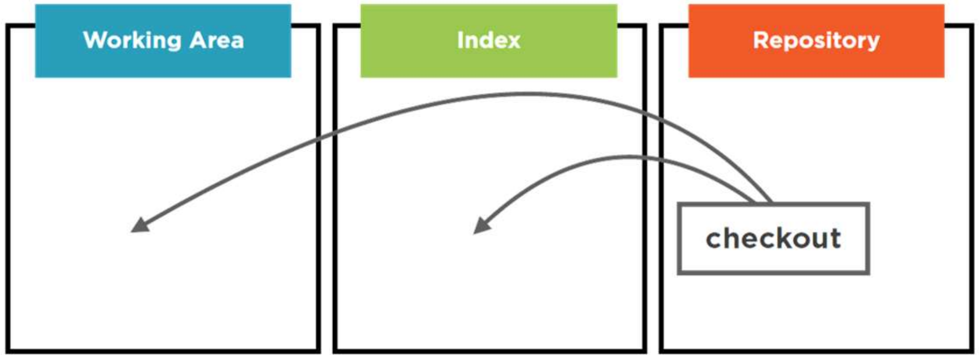
- It takes data from the new current commit, and it copies that data from the repository
  - to the working area and
  - To the index.

Time travel to the past

In sumary



New active commit (with HEAD)



Previous commit changes to active (with HEAD)

git restore & git switch

In the repository, git checkout moves the HEAD reference => change de repository: changes the current commit

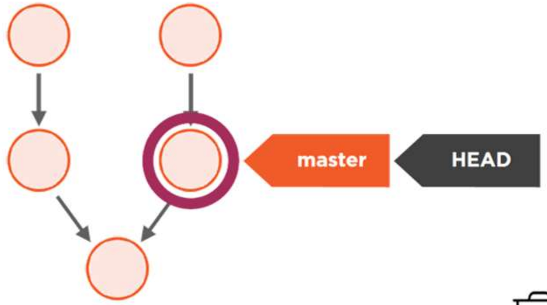
- generally, to another branch → git switch
- to another commit in the branch (recovering and earli commit) → git restore

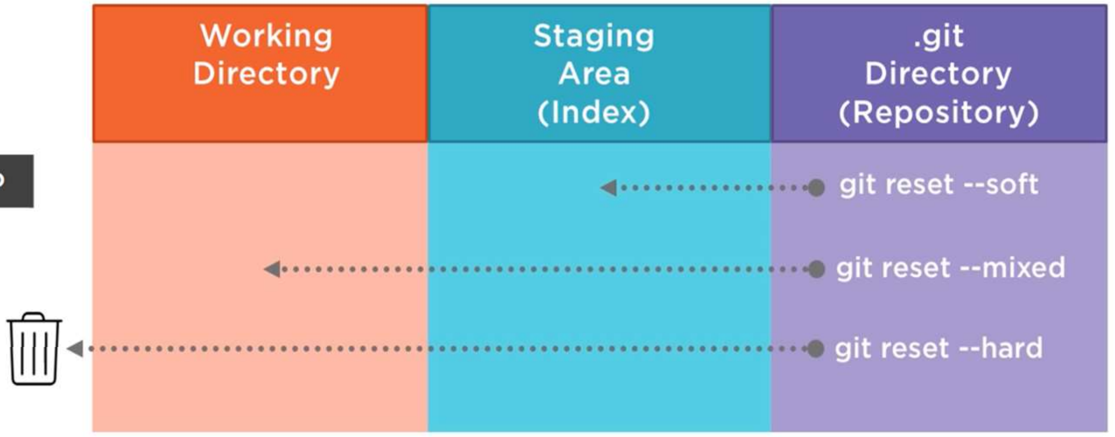
If we need to unstage some file:
git restore --staged index.html
If we need to undo changes in working directory:
git restore scripts.js
Warning: this operation can't be reverted!

Better functionality for unstage a file, in stead of reset

The Three Options of Git Reset

1. It **moves the current branch**, so it also changes the current commit.
2. Optionally, it copies the files and directories from the new current commit to the working area and the index



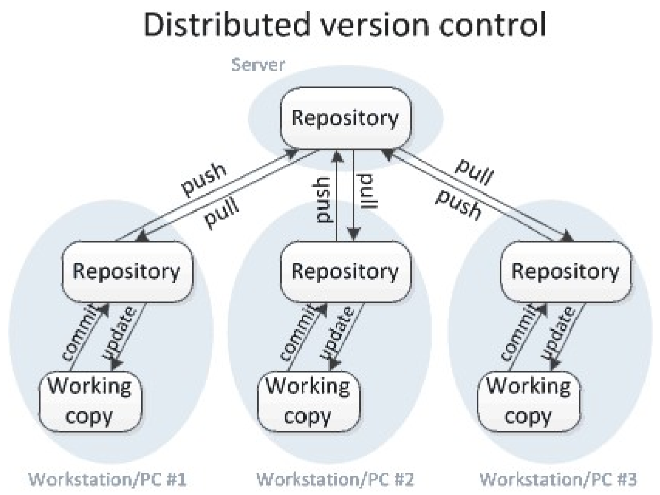


Squashing

- git commit a
- git commit b
- git commit c
- git reset –soft HEAD~3
- // a – b – c came back to index
- git commit a-b-c

Commits in summary (again)

- It's a snapshot of our project
- It has an identificator
- It has a message describing the changes
- It doesn't dissapear (not true)
- It knows which its previous commit/s are
- It knows who its author is
- It knows the date and time when it was created



Servers

- *gitolite* http://gitolite.com/gitolite/index.html
- *gitosis* https://git-scm.com/book/es/v1/Git-en-un-servidor-Gitosis
- *GitLab*, para Linux (interfaz Web,)https://about.gitlab.com/equipos

Hosting (repositories)

- GitHub - https://github.com/
- *Bitbucket* https://bitbucket.org/
- *GitLab* https://about.gitlab.com/

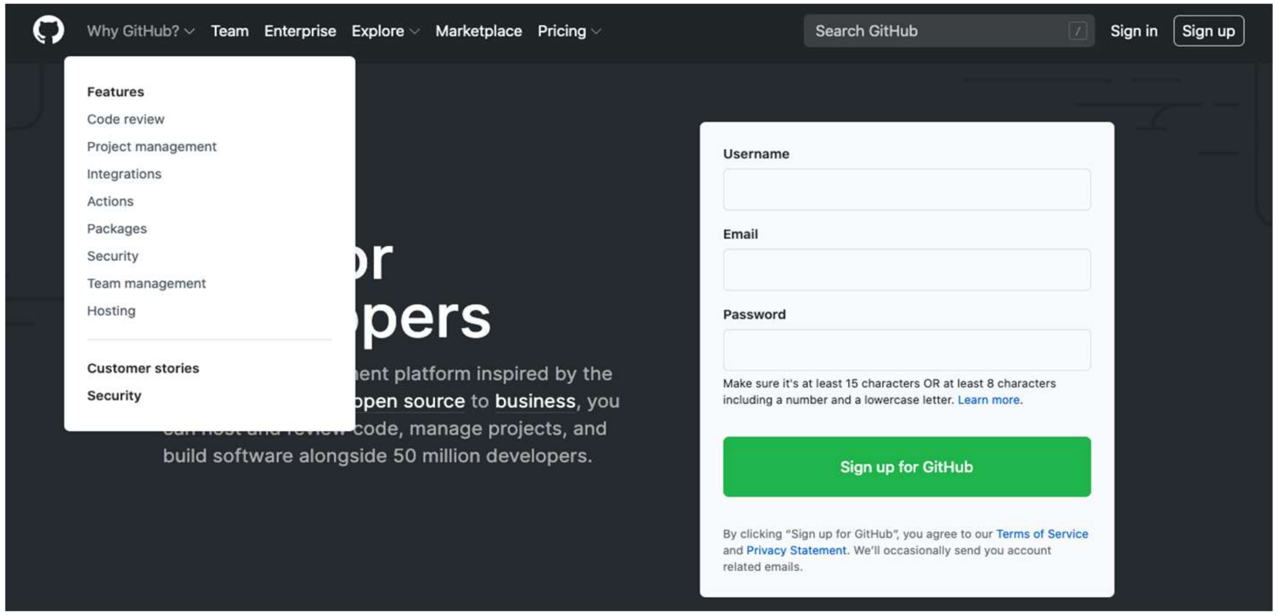
Repesitories Hosting

GitHub

GitLab

ATLASSIAN
Bitbucket

Colaboration



r
pers

ent platform inspired by the pen source to business, you code, manage projects, and build software alongside 50 million developers.

Username

Email

Password

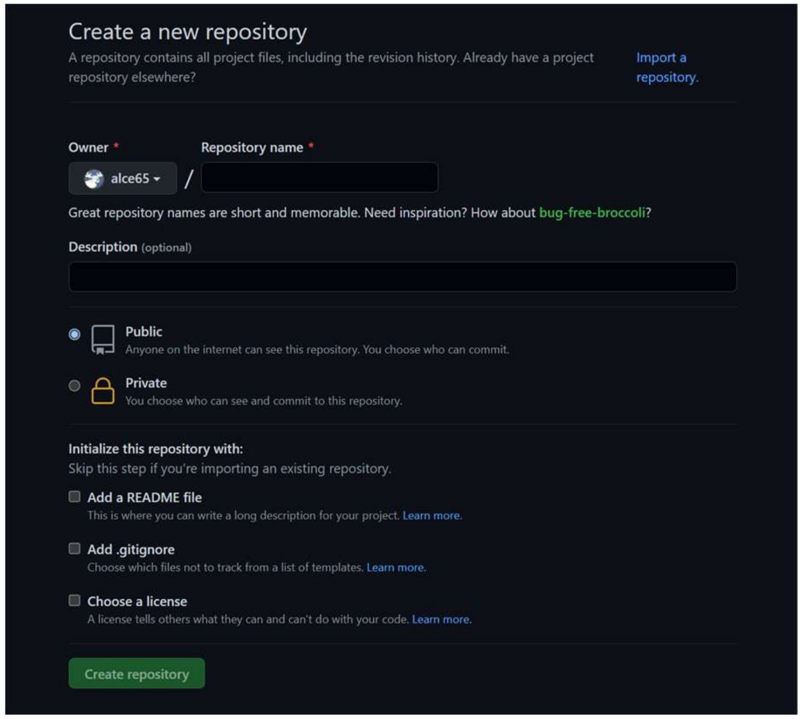
Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our Terms of Service and Privacy Statement. We'll occasionally send you account related emails.

More than 2.9 million businesses and organizations use GitHub

Create a new GitHub repository



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * / Repository name *

alce65 /

Great repository names are short and memorable. Need inspiration? How about bug-free-broccoli?

Description (optional)

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ Add a README file
This is where you can write a long description for your project. Learn more.

☐ Add .gitignore
Choose which files not to track from a list of templates. Learn more.

☐ Choose a license
A license tells others what they can and can't do with your code. Learn more.

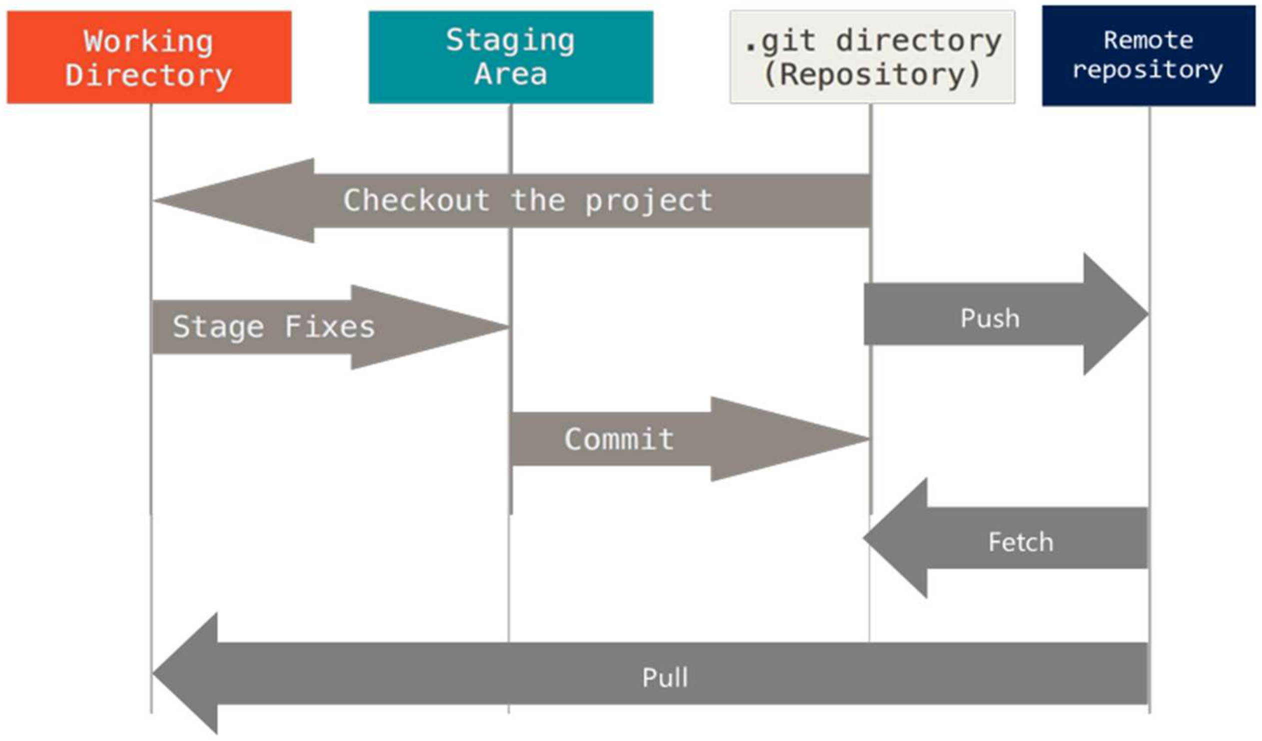
Push a Git Repository to a Code Hosting Provider

git remote : connect to remote
git push : update remote

```
echo “# skylab-bootcamp-202007” >> README.md
git add .
git commit -m “Create readme file”
git remote add origin https://github.com/<...>/<...>.git
git push -u origin master
```

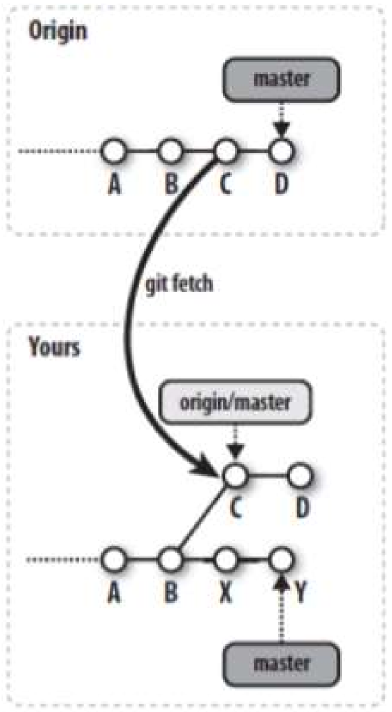
- Code hosting providers: Github
- Create an account
- Push out Git project to a code hosting provider

Extended Git Basic Operations

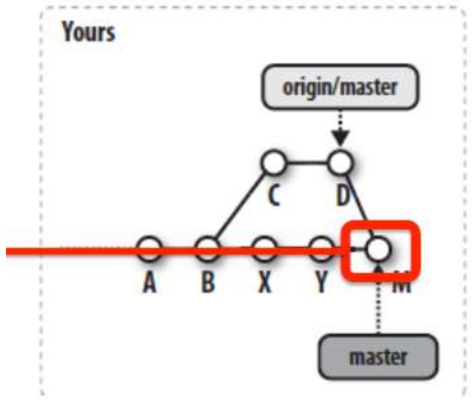


Import data from a GitHub repository

git pull
git fetch



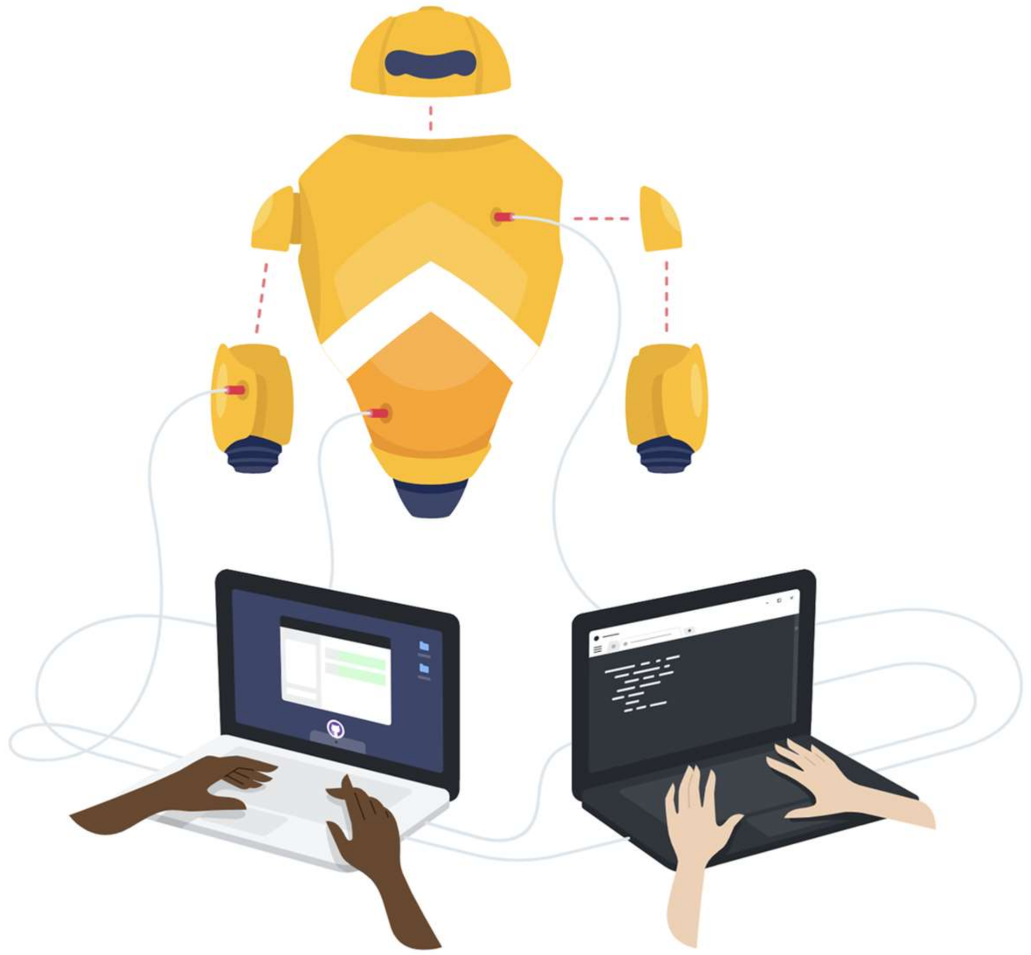
git fetch <repo> <branch>



git pull <repo> <branch>

Git Pull = Git Fetch + Git Merge

git pull -rebase <repo> <branch>



Write better code

Collaboration makes perfect. The conversations and code reviews that happen in pull requests help your team share the weight of your work and improve the software you build.

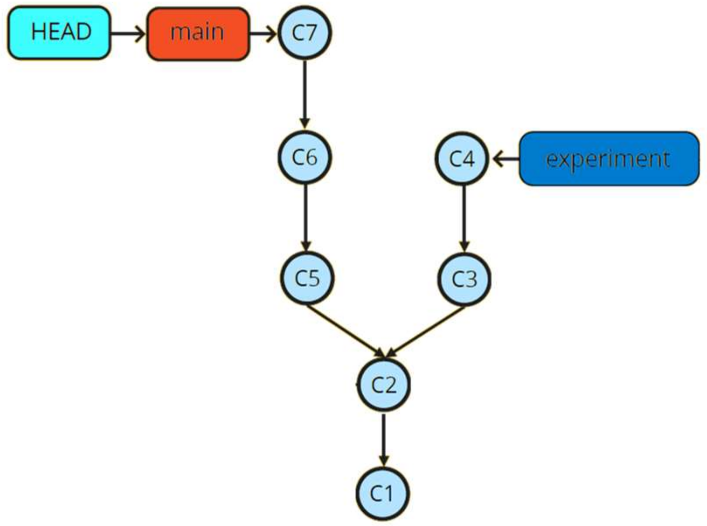
Manage your chaos

Take a deep breath. On GitHub, project management happens in issues and project boards, right alongside your code. All you have to do is mention a teammate to get them involved.

Find the right tools

Browse and buy apps from GitHub Marketplace with your GitHub account. Find the tools you like or discover new favorites—then start using them in minutes.

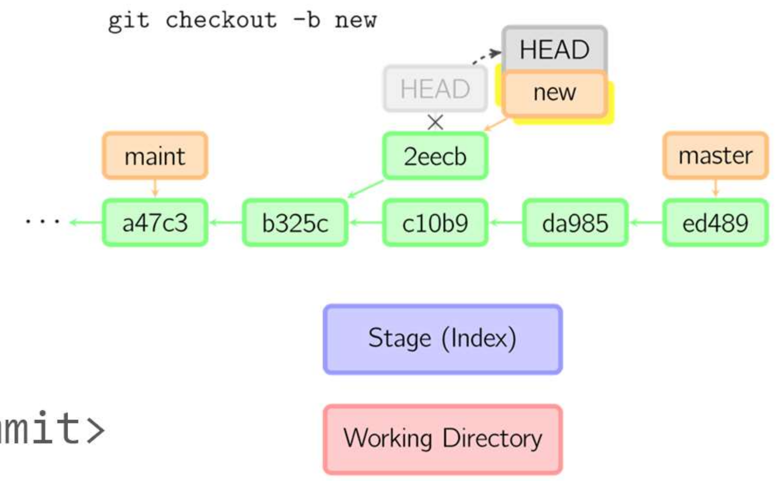
Branches



- Branches allow developing in isolated contexts
- Branches are just references
- A branch points to a commit
- A repo has at least one branch
- We shouldn't commit to the main branch

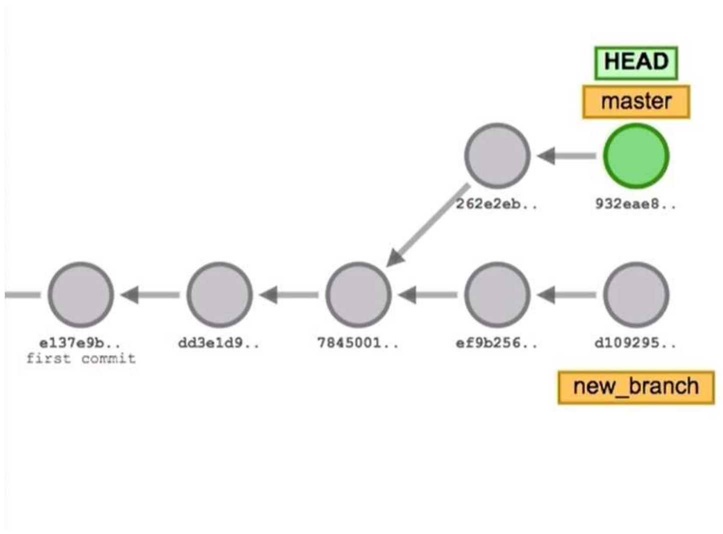
Branches: commands

- Create:
  `git branch <branch-name>`
- Create and move to
  `git checkout -b <branch-name>`
- Create fron one commit
  `git branch <branch-name> <id-commit>`



Introduction to Branches

http://git-school.github.io/visualizing-git/



```
git commit
git commit
git checkout -b new_branch
git commit
git commit
git checkout master
git commit
git commit
git checkout new_branch
git commit
```

More branch commands

- `git branch` |-> List
- `git branch -a` |-> List with remotes
- `git show-branch` |-> View
- `git branch -m <branch-old> <branch-new>` |-> rename
- `git branch -d <branch>` |-> delete
- `git branch -D <branch>` |-> forced delete
- `git push <remote> :<branch>` |-> remote delete

Working with Branches: stash

```
git stash / git stash --include-untracked

git stash list

git stash show

git stash apply

git stash clear

git stash pop
```

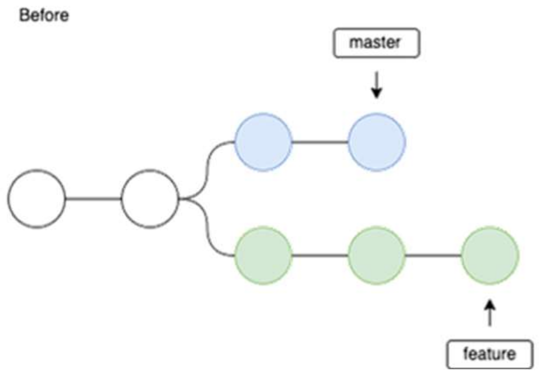
Stash (like a clipboard)

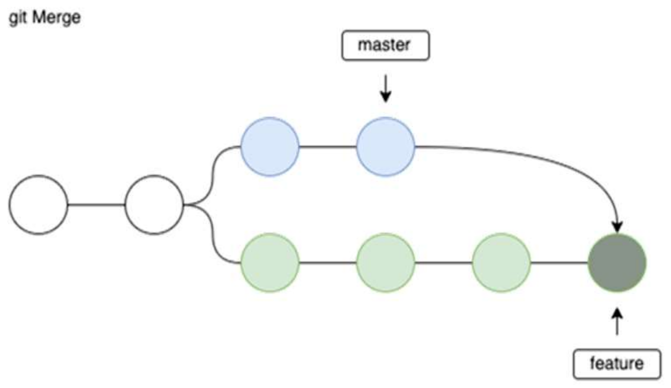
- **takes all the data from the working area and the index** that is not in the current commit in the repository,
- copies all of that data to the stash.

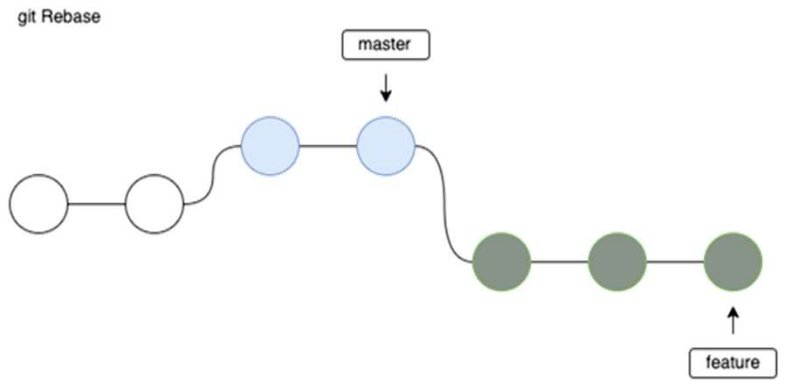
And then it also **checks out the current commit** (‘clean’ working area and the index)

Combining branches

- Merge
  - Fast-forward
  - Recursive or tree-way (no-ff)
  - squash
- Rebase





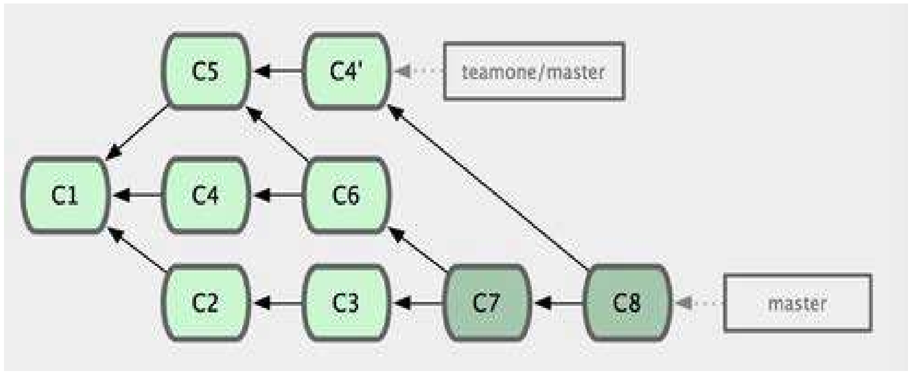


Merge branches

git merge <source_branch>
(run it while on the target branch)

Includes in one branch the work from another branch

Goal: we want one branch to have/include commits from another branch

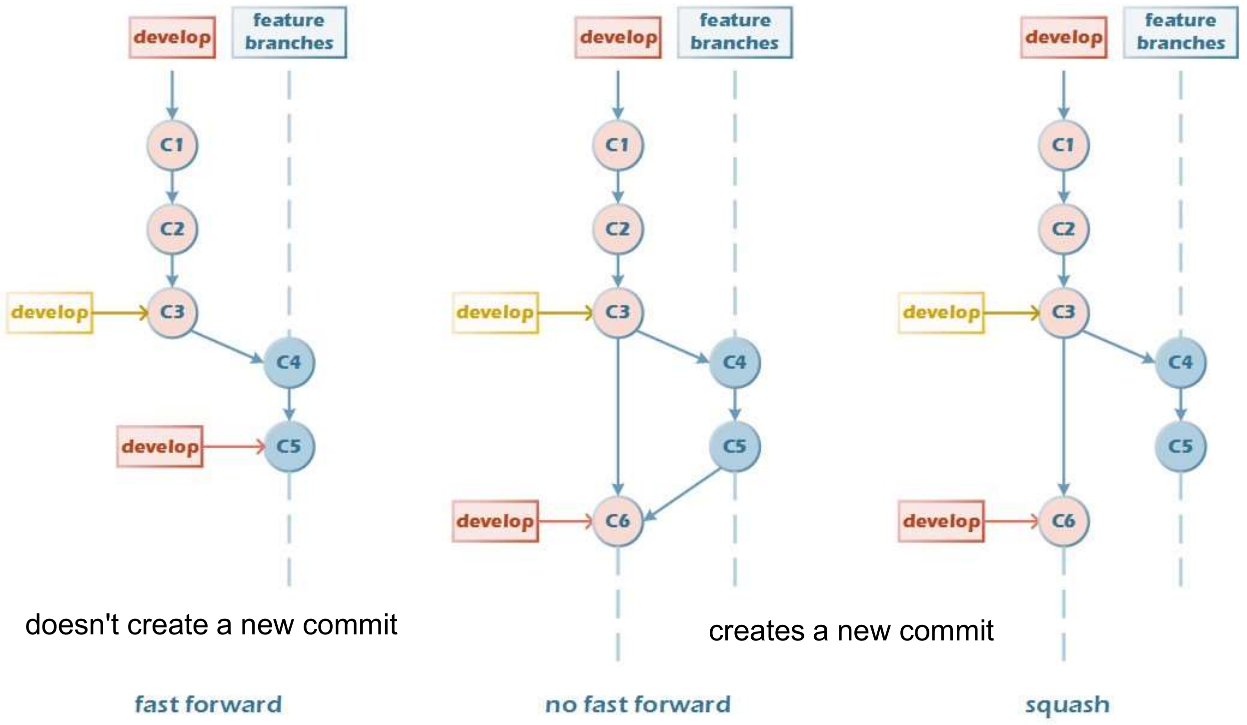


https://medium.com/@mena.meseha/git-merge-vs-rebase-556563b26431

Merge branches - Scenario 1

After the feature branch is cut out, there is no new commit on the develop branch.

no conflicts



doesn't create a new commit

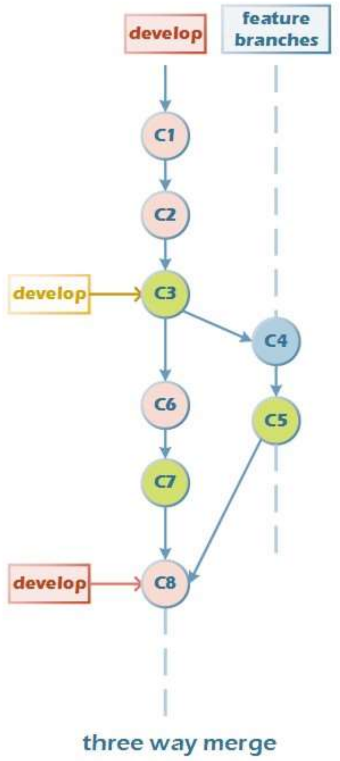
creates a new commit

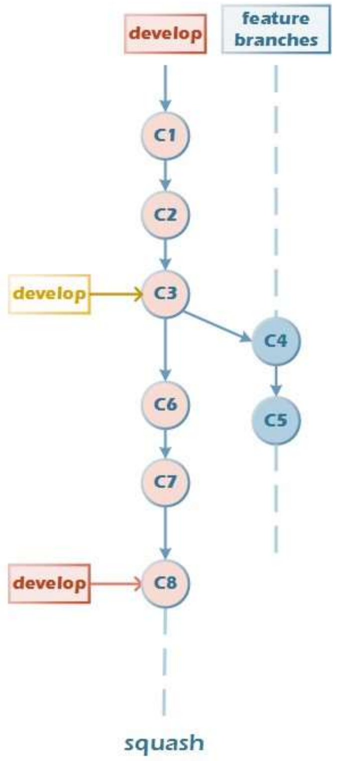
Merge branches - Scenario 2

After cutting out the feature branch, C6 and C7 are submitted on the develop branch.

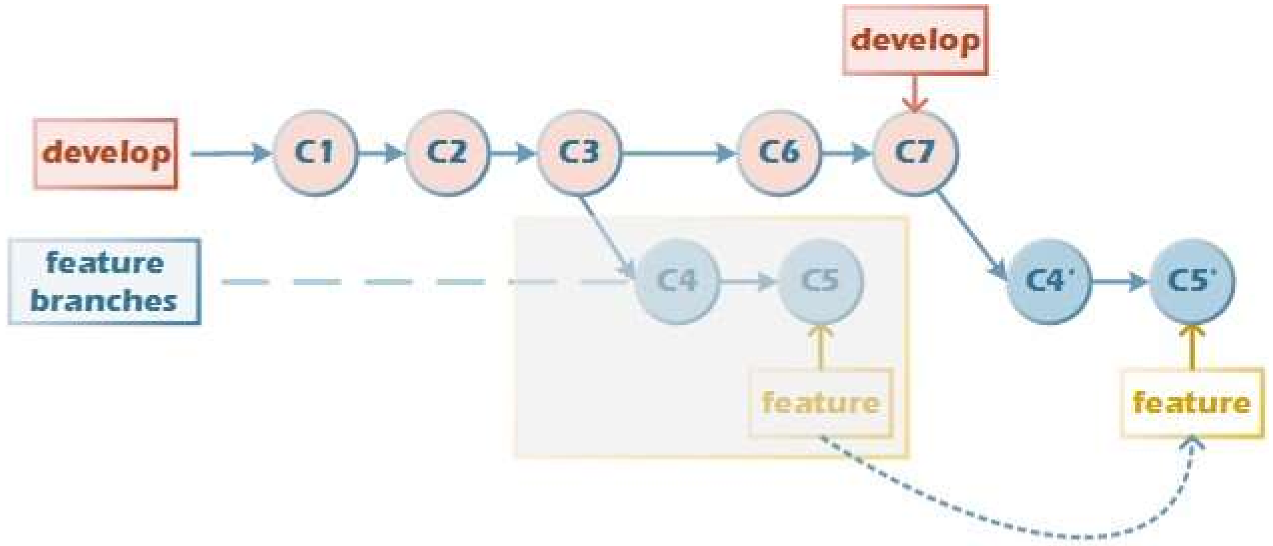
cannot be fast forward.

creates a new commit, possibility of conflicts





Rebase



Rebase, has no effect on the specified base itself; just rewrite the commit history after the base.

Git merge conflicts

- Managing contributions between multiple distributed authors (usually developers).
- The git merge command's primary responsibility is to combine separate branches and resolve any conflicting edits.
    - Most of the time, Git will figure out how to automatically integrate new changes.
    - A merge will fail to start when Git sees there are changes in either the working directory or staging area of the current project.
    - Sometimes multiple developers may try to edit the same content: Developer A tries to edit code that Developer B is editing .
      In the merge, Git cannot automatically determine what is correct and a conflict may occur..

Resolving conflicts

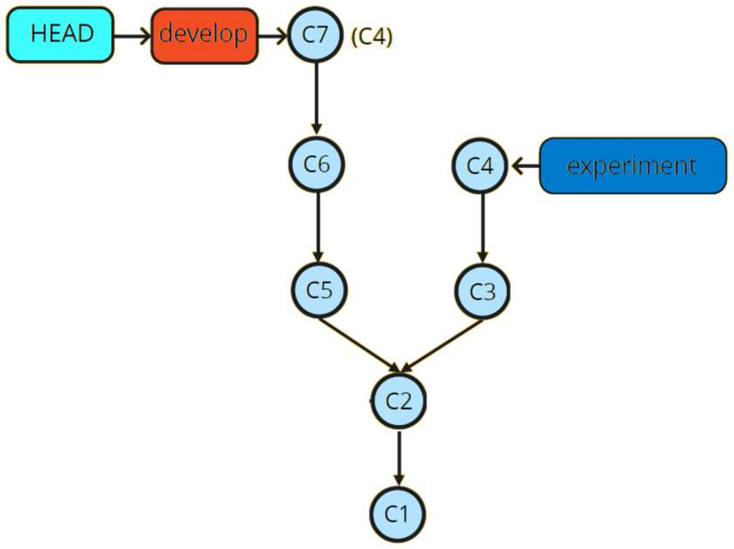
```
$ git merge new_branch_to_merge_later
Auto-merging merge.txt
CONFLICT (content): Merge conflict in merge.txt
Automatic merge failed; fix conflicts and then commit the result.
```

git status
git diff

```
$ cat merge.txt
<<<<<<< HEAD
this is some content to mess with
content to append
=======
totally different content to merge later
>>>>>>> new_branch_to_merge_later
```

Cherry-picking

- Command to cherry-pick a commit:
  git cherry-pick <commit> -x
- Apply a single commit to a branch
- It doesn't apply previous commits
- Modifier -x adds "Cherry-picked from XXXX" to the commit message
- Conflicts: git status



Changes in the history

- git commit --amend
- git rebase -i
- git reflog
- git filter-repo
- git revert

The Golden Rule:
never change shared history.

Distributed Workflow

Distribution Model

How many repositories do you have? Who can access them? …

Branching Model

Which branches do you have? How do you use them? …

Constraints

Do you merge or do you rebase? Can you push unstable code? …

Distribution Model

- Peer to Peer Model
- Centralized Model
- Pull Request Model
- Dictator and Lieutenants Model

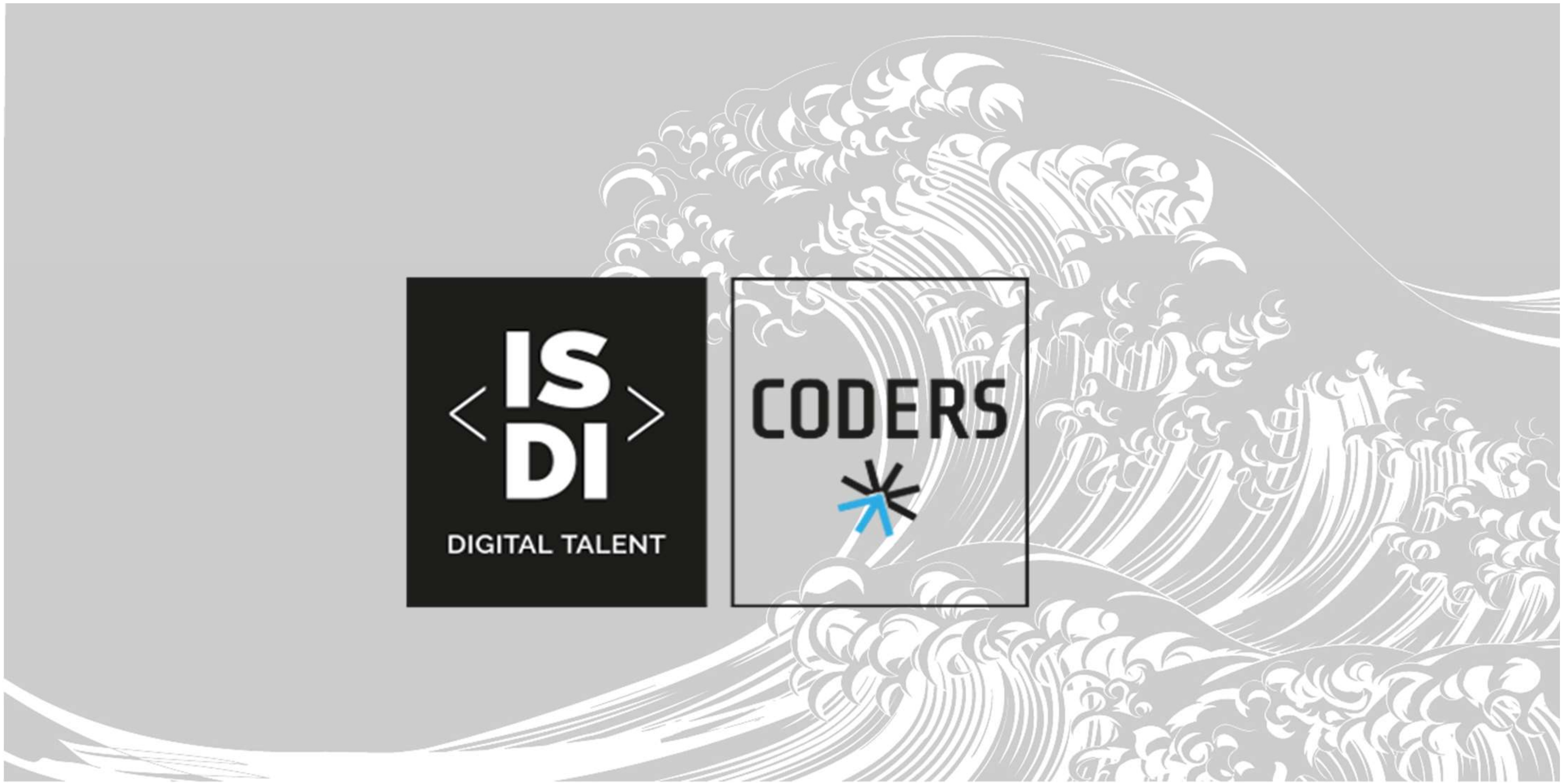
Many projects use a mixed Distribution Model.

Branching Models

- Stable and unstable branches
- Common Branches
    - Integration Branch
    - Release Branch
    - Feature Branch
    - Hotfix Branch

A Few Examples of Constraints

- rebase, don’t merge / merge, don’t rebase
- Only developer X can do Y on branch Z
- Don’t push to a red build
- Squash a feature to a single commit before you merge it to master


IS DI
DIGITAL TALENT
CODERS