Curso: <https://www.youtube.com/watch?v=kEPF-MWGq1w>

Github: [https://github.com](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqa2NXbFZqVmlTYzFIM1cwY1hKTGo1bnJQcHEwQXxBQ3Jtc0tsenhud1R0My03VTBtM0liZ3FIcWVkb2gzSUdMUGVBQ0RSRnVITE9qMHVRNXk0dDkwdUdEdVJNbGhsTXRPc2FEZG9GN0x1TmZyQml4aXNCYV96VHVQQy15bWlGQTJWd3RYdG9LaFp5VUUwMXd6ZllrNA&q=https%3A%2F%2Fgithub.com%2F&v=kEPF-MWGq1w)

Bitbucket (alternativa a Github): [https://bitbucket.com](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbW81aWtCR05uekNxb2ZGRWh3M1FkWDJON094Z3xBQ3Jtc0ttanNNbFJJT0xfRnNhcDBQMjBzb2VDX2lVdU1wbXpobkZseS1JZVR6NjZOXzdFSF9YMktzSWZpVkZMS2VqNkU0c2JiUVBPWkpQRWx6bnpKUnEwNmNMZ0pnSGVBV19JeGJJUTEwSmN5TzVwODBoRnNuVQ&q=https%3A%2F%2Fbitbucket.com%2F&v=kEPF-MWGq1w)

Check if Git is already installed: [cpc@CPCtelera ShooterOlmo]$ git version

Install Git if it isn’t already installed ([https://git-scm.com](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqblR1d0J2NnVHcGU3VWpTaHhFOWFhWDREbUFvUXxBQ3Jtc0tsUVhBLUFFcmZzRVFkQVFaRDdOZC1hTHVhUDBpdGlCRTUtSkEyR3hZeGtiaTRlQzFiY0xpbUFfbUFFbGx3dS10LXpmWW5kUE1ULU0tS0pWVV8zcldFRXFOWHJXUEVOSGdxTUJmaENyQm1tOHVmaVFTNA&q=https%3A%2F%2Fgit-scm.com%2F&v=kEPF-MWGq1w" \t "_blank))

To configure your credentials so that anyone can know if and when you’ve mae changes to the repository:

> git config –-global user.name “Olmo Ramiro”

> git config –-global user.email [olmo.ramiro@gmai.com](mailto:olmo.ramiro@gmai.com)

> git config –-list <- Shows current config

Example:

core.repositoryformatversion=0

core.filemode=true

core.bare=false

core.logallrefupdates=true

remote.origin.url=https://github.com/lronaldo/cpctelera

remote.origin.fetch=+refs/heads/\*:refs/remotes/origin/\*

branch.master.remote=origin

branch.master.merge=refs/heads/master

branch.development.remote=origin

branch.development.merge=refs/heads/development

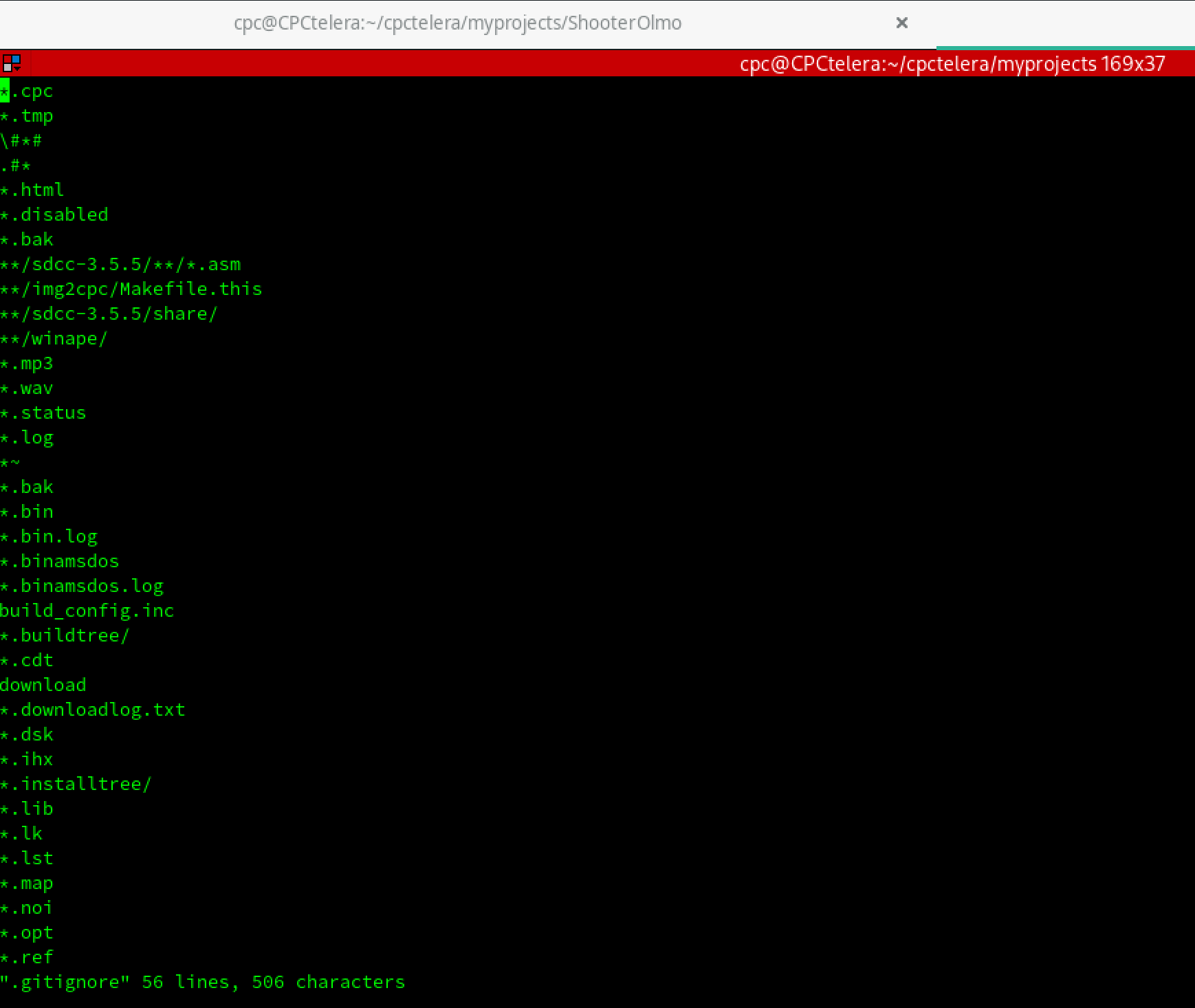
# Create a local Repository Managed by GIT:

Go to the folder where your code is and run:

[cpc@CPCtelera myprojects]$ git init <-This command creates a directory called .git where all the changes will be stored locally.

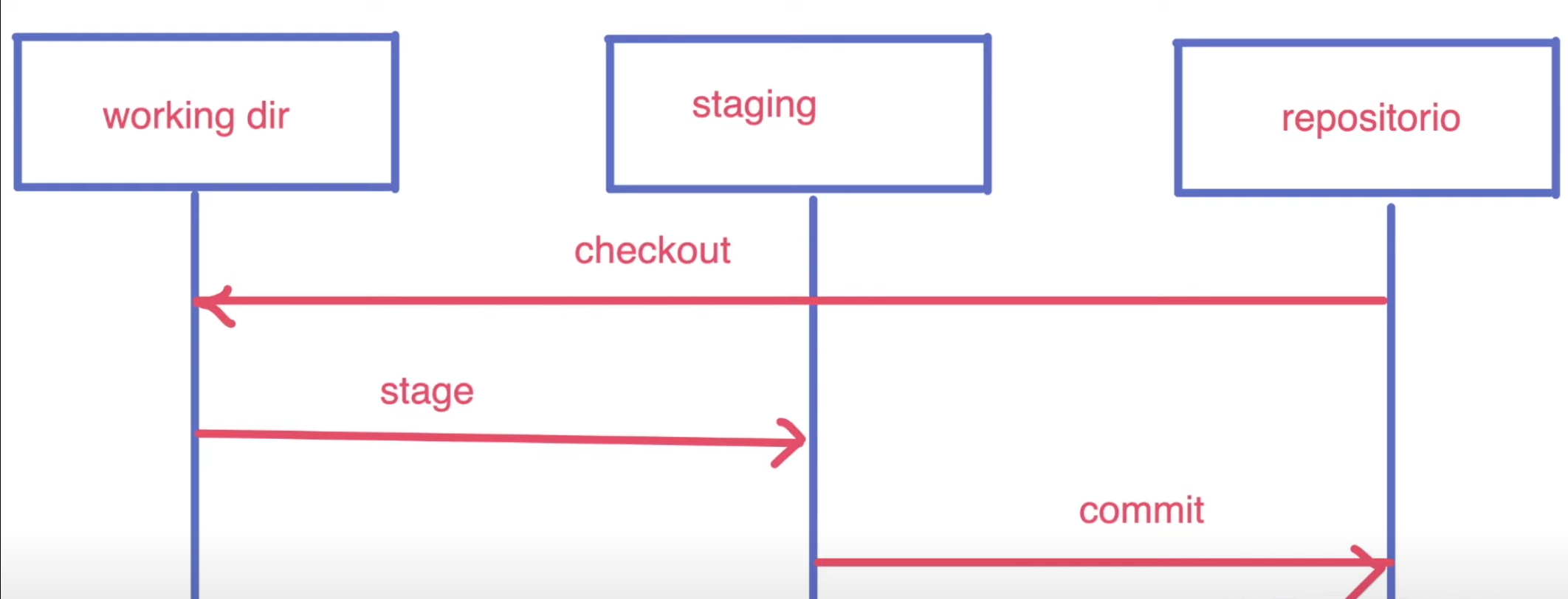
If we don’t want a file or certain files to be part of the repository then we can create a .gitingore file:

[cpc@CPCtelera myprojects]$ vi .gitignore

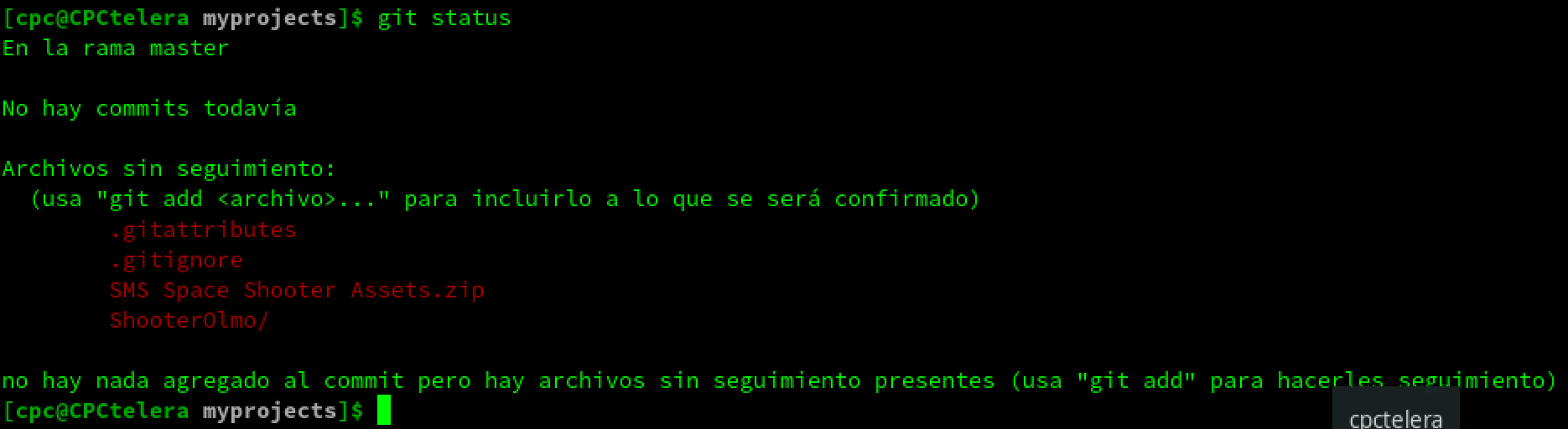


Code can be in the following areas or states:

* Working Dir is where you’re making your code modifications
* Staging is sort of an intermediate stage
* Repository is where the changes get stored when they’ve been committed.

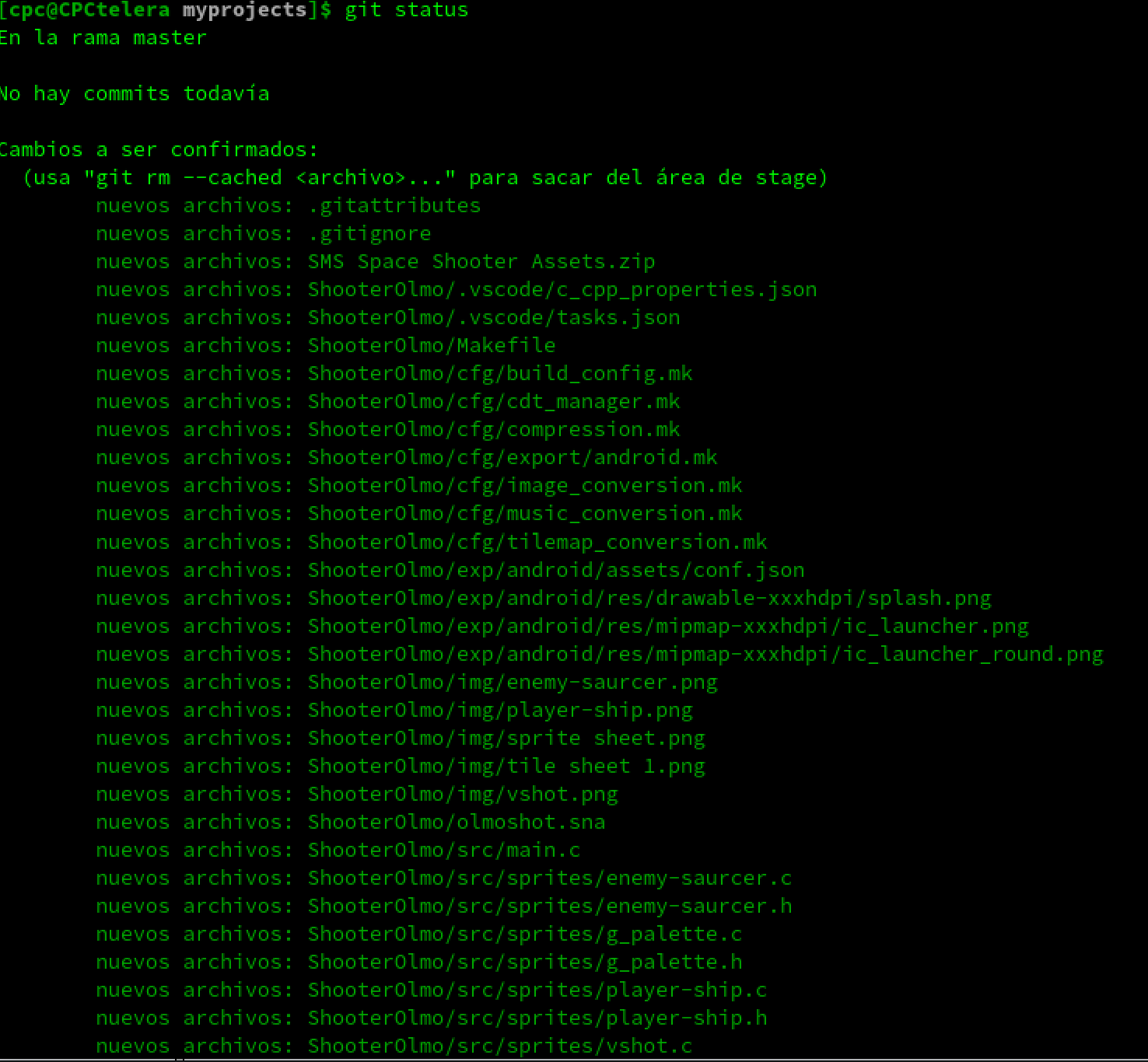


[cpc@CPCtelera myprojects]$ git status



We can add all the files in the current directory by using –A:

[cpc@CPCtelera myprojects]$ git add –A



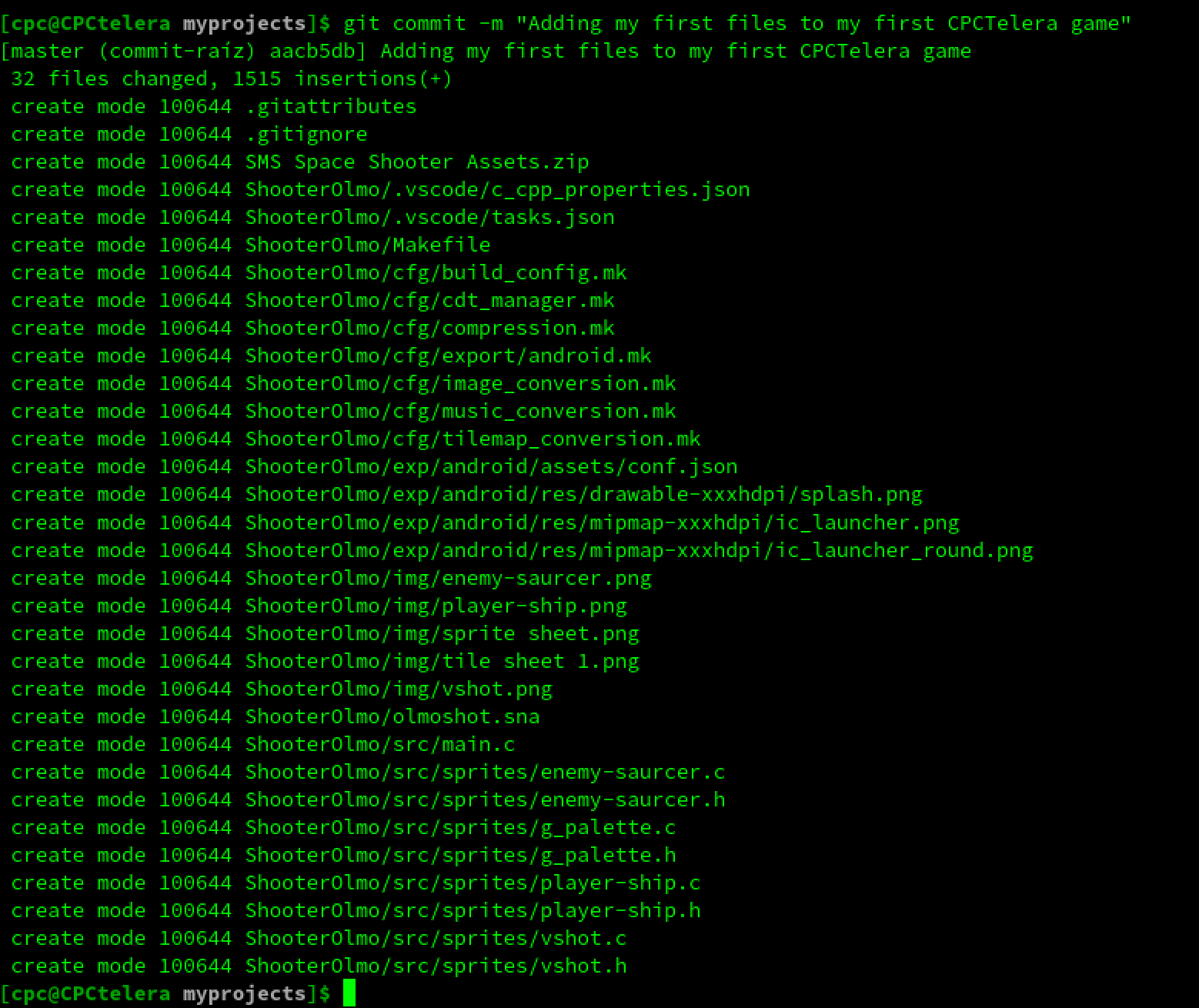
All these files have been added to the staging area and the master branch, but they are still not been committed.

If we want to remove files from the staging area we can use the command:

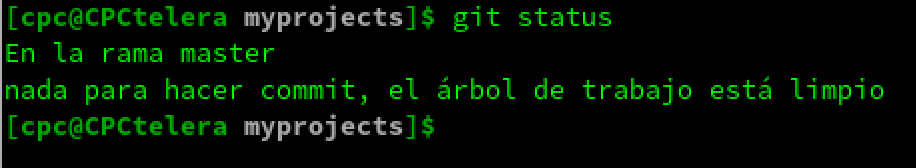
[cpc@CPCtelera myprojects]$ git rm -–cached <file name> <-This command removes <file name> from the staging area

Now we can add these changes to the repository by running commit and adding a message.

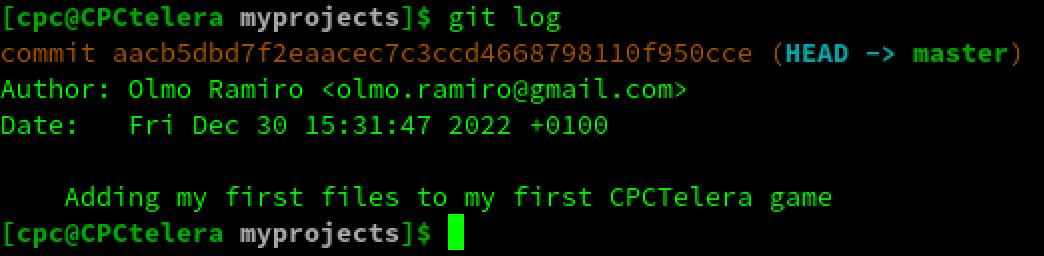
[cpc@CPCtelera myprojects]$git commit –m “Adding the my first version”



And if I try to run git status again then I will get nothing because the files are no longer in the working dir or in the staging area.



[cpc@CPCtelera myprojects]$ git log <-This command shows all the commits s



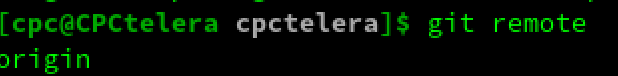
# Git and GitHUB – Remote Repositories

## https://git-scm.com/book/es/v2/Fundamentos-de-Git-Trabajar-con-Remotos

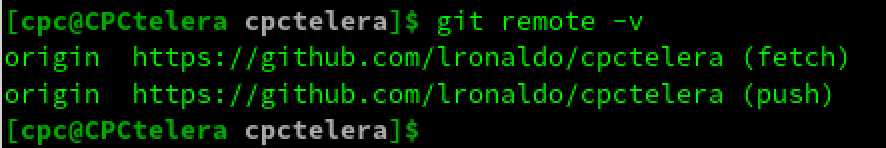
Origin makes reference to the remote “URL” where push, pull, and fetch will be working with

[cpc@CPCtelera myprojects]$ git clone “URL” <-This command download a repository from the URL and gets all the commits and logs

Git clone sets the current “origin” to be the URL, however there may be more URLs to work with and origin can also be changed so that it points to a different URL.

[cpc@CPCtelera myprojects]$ Git remote <-This command shows the origin an any other existing pointers 

[cpc@CPCtelera myprojects]$ it remote –v <-This command shows the URLs assigned to the each pointer



### Añadir Repositorios Remotos

En secciones anteriores hemos mencionado y dado alguna demostración de cómo añadir repositorios remotos. Ahora veremos explícitamente cómo hacerlo. Para añadir un remoto nuevo y asociarlo a un nombre que puedas referenciar fácilmente, ejecuta git remote add [nombre] [url]:

$ git remote

origin

$ git remote add pb https://github.com/paulboone/ticgit

$ git remote -v

origin https://github.com/schacon/ticgit (fetch)

origin https://github.com/schacon/ticgit (push)

pb https://github.com/paulboone/ticgit (fetch)

pb https://github.com/paulboone/ticgit (push)

A partir de ahora puedes usar el nombre pb en la línea de comandos en lugar de la URL entera. Por ejemplo, si quieres traer toda la información que tiene Paul pero tú aún no tienes en tu repositorio, puedes ejecutar git fetch pb:

$ git fetch pb

remote: Counting objects: 43, done.

remote: Compressing objects: 100% (36/36), done.

remote: Total 43 (delta 10), reused 31 (delta 5)

Unpacking objects: 100% (43/43), done.

From https://github.com/paulboone/ticgit

\* [new branch] master -> pb/master

\* [new branch] ticgit -> pb/ticgit

La rama maestra de Paul ahora es accesible localmente con el nombre pb/master - puedes combinarla con alguna de tus ramas, o puedes crear una rama local en ese punto si quieres inspeccionarla. (Hablaremos con más detalle acerca de qué son las ramas y cómo utilizarlas en [[ch03-git-branching]](https://git-scm.com/book/es/v2/ch00/ch03-git-branching).)

[cpc@CPCtelera myprojects]$ git diff <-This command shows the differences between the files in your working dir and the repository

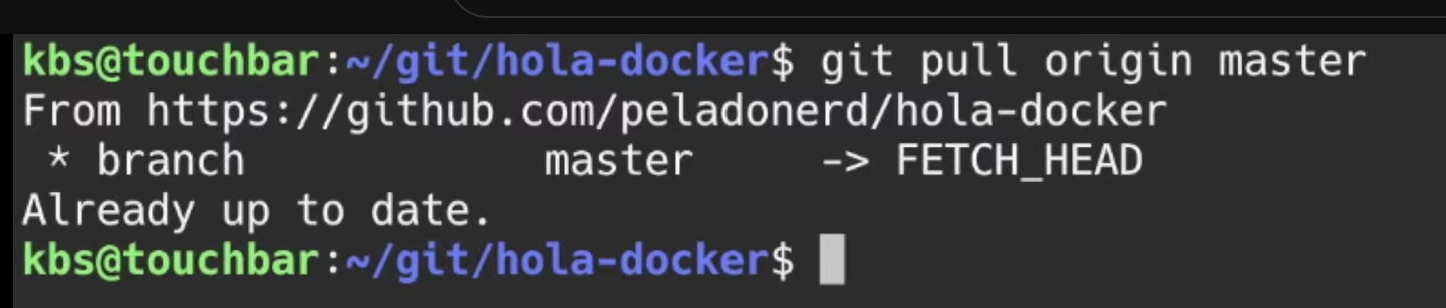


To upload our changes to the repository in github or on the internet we need to run:

[cpc@CPCtelera myprojects]$ git push origin master

To make sure we’re using the code’s latest version from github or the internet then we need to run:

[cpc@CPCtelera myprojects]$ git pull origin master



# Git Secrets:

git secrets --scan [-r|--recursive] [--cached] [--no-index] [--untracked] [<files>...]

git secrets --scan-history

git secrets --install [-f|--force] [<target-directory>]

git secrets --list [--global]

git secrets --add [-a|--allowed] [-l|--literal] [--global] <pattern>

git secrets --add-provider [--global] <command> [arguments...]

git secrets --register-aws [--global]

git secrets --aws-provider [<credentials-file>]

## [Description](https://github.com/awslabs/git-secrets/blob/master/README.rst#id5)

git-secrets scans commits, commit messages, and --no-ff merges to prevent adding secrets into your git repositories. If a commit, commit message, or any commit in a --no-ff merge history matches one of your configured prohibited regular expression patterns, then the commit is rejected.

## [Installing git-secrets](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id6)

git-secrets must be placed somewhere in your PATH so that it is picked up by git when running git secrets.

### [\*nix (Linux/macOS)](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id7)

You can use the install target of the provided Makefile to install git secrets and the man page. You can customize the install path using the PREFIX and MANPREFIX variables.

make install

### [Windows](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id8)

Run the provided install.ps1 powershell script. This will copy the needed files to an installation directory (%USERPROFILE%/.git-secrets by default) and add the directory to the current user PATH.

PS > ./install.ps1

### [Homebrew (for macOS users)](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id9)

brew install git-secrets

Warning

**You're not done yet! You MUST install the git hooks for every repo that you wish to use with** git secrets --install.

Here's a quick example of how to ensure a git repository is scanned for secrets on each commit:

cd /path/to/my/repo

git secrets --install

git secrets --register-aws

## [Advanced configuration](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id10)

Add a configuration template if you want to add hooks to all repositories you initialize or clone in the future.

git secrets --register-aws --global

Add hooks to all your local repositories.

git secrets --install ~/.git-templates/git-secrets

git config --global init.templateDir ~/.git-templates/git-secrets

Add custom providers to scan for security credentials.

git secrets --add-provider -- cat /path/to/secret/file/patterns

## [Before making public a repository](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id11)

With git-secrets is also possible to scan a repository including all revisions:

git secrets --scan-history

## [Options](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id12)

### [Operation Modes](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id13)

Each of these options must appear first on the command line.

*--install*

Installs git hooks for a repository. Once the hooks are installed for a git repository, commits and non-fast-forward merges for that repository will be prevented from committing secrets.

*--scan*

Scans one or more files for secrets. When a file contains a secret, the matched text from the file being scanned will be written to stdout and the script will exit with a non-zero status. Each matched line will be written with the name of the file that matched, a colon, the line number that matched, a colon, and then the line of text that matched. If no files are provided, all files returned by git ls-files are scanned.

*--scan-history*

Scans repository including all revisions. When a file contains a secret, the matched text from the file being scanned will be written to stdout and the script will exit with a non-zero status. Each matched line will be written with the name of the file that matched, a colon, the line number that matched, a colon, and then the line of text that matched.

*--list*

Lists the git-secrets configuration for the current repo or in the global git config.

*--add*

Adds a prohibited or allowed pattern.

*--add-provider*

Registers a secret provider. Secret providers are executables that when invoked output prohibited patterns that git-secrets should treat as prohibited.

*--register-aws*

Adds common AWS patterns to the git config and ensures that keys present in ~/.aws/credentials are not found in any commit. The following checks are added:

* AWS Access Key IDs via (A3T[A-Z0-9]|AKIA|AGPA|AIDA|AROA|AIPA|ANPA|ANVA|ASIA)[A-Z0-9]{16}
* AWS Secret Access Key assignments via ":" or "=" surrounded by optional quotes
* AWS account ID assignments via ":" or "=" surrounded by optional quotes
* Allowed patterns for example AWS keys (AKIAIOSFODNN7EXAMPLE and wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY)
* Known credentials from ~/.aws/credentials

Note

While the patterns registered by this command should catch most instances of AWS credentials, these patterns are **not** guaranteed to catch them **all**. git-secrets should be used as an extra means of insurance -- you still need to do your due diligence to ensure that you do not commit credentials to a repository.

*--aws-provider*

Secret provider that outputs credentials found in an INI file. You can optionally provide the path to an INI file.

### [Options for --install](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id14)

*-f, --force*

Overwrites existing hooks if present.

*<target-directory>*

When provided, installs git hooks to the given directory. The current directory is assumed if <target-directory> is not provided.

If the provided <target-directory> is not in a git repository, the directory will be created and hooks will be placed in <target-directory>/hooks. This can be useful for creating git template directories using with git init --template <target-directory>.

You can run git init on a repository that has already been initialized. From the [git init documentation](https://git-scm.com/docs/git-init):

From the git documentation: Running git init in an existing repository is safe. It will not overwrite things that are already there. The primary reason for rerunning git init is to pick up newly added templates (or to move the repository to another place if --separate-git-dir is given).

The following git hooks are installed:

1. pre-commit: Used to check if any of the files changed in the commit use prohibited patterns.
2. commit-msg: Used to determine if a commit message contains a prohibited patterns.
3. prepare-commit-msg: Used to determine if a merge commit will introduce a history that contains a prohibited pattern at any point. Please note that this hook is only invoked for non fast-forward merges.

Note

Git only allows a single script to be executed per hook. If the repository contains Debian-style subdirectories like pre-commit.d and commit-msg.d, then the git hooks will be installed into these directories, which assumes that you've configured the corresponding hooks to execute all of the scripts found in these directories. If these git subdirectories are not present, then the git hooks will be installed to the git repo's .git/hooks directory.

#### Examples

Install git hooks to the current directory:

cd /path/to/my/repository

git secrets --install

Install git hooks to a repository other than the current directory:

git secrets --install /path/to/my/repository

Create a git template that has git-secrets installed, and then copy that template into a git repository:

git secrets --install ~/.git-templates/git-secrets

git init --template ~/.git-templates/git-secrets

Overwrite existing hooks if present:

git secrets --install -f

### [Options for --scan](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id15)

*-r, --recursive*

Scans the given files recursively. If a directory is encountered, the directory will be scanned. If -r is not provided, directories will be ignored.

-r cannot be used alongside --cached, --no-index, or --untracked.

*--cached*

Searches blobs registered in the index file.

*--no-index*

Searches files in the current directory that is not managed by git.

*--untracked*

In addition to searching in the tracked files in the working tree, --scan also in untracked files.

*<files>...*

The path to one or more files on disk to scan for secrets.

If no files are provided, all files returned by git ls-files are scanned.

#### Examples

Scan all files in the repo:

git secrets --scan

Scans a single file for secrets:

git secrets --scan /path/to/file

Scans a directory recursively for secrets:

git secrets --scan -r /path/to/directory

Scans multiple files for secrets:

git secrets --scan /path/to/file /path/to/other/file

You can scan by globbing:

git secrets --scan /path/to/directory/\*

Scan from stdin:

echo 'hello!' | git secrets --scan -

### [Options for --list](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id16)

*--global*

Lists only git-secrets configuration in the global git config.

### [Options for --add](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id17)

*--global*

Adds patterns to the global git config

*-l, --literal*

Escapes special regular expression characters in the provided pattern so that the pattern is searched for literally.

*-a, --allowed*

Mark the pattern as allowed instead of prohibited. Allowed patterns are used to filter out false positives.

*<pattern>*

The regex pattern to search.

#### Examples

Adds a prohibited pattern to the current repo:

git secrets --add '[A-Z0-9]{20}'

Adds a prohibited pattern to the global git config:

git secrets --add --global '[A-Z0-9]{20}'

Adds a string that is scanned for literally (+ is escaped):

git secrets --add --literal 'foo+bar'

Add an allowed pattern:

git secrets --add -a 'allowed pattern'

### [Options for --register-aws](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id18)

*--global*

Adds AWS specific configuration variables to the global git config.

### [Options for --aws-provider](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id19)

*[<credentials-file>]*

If provided, specifies the custom path to an INI file to scan. If not provided, ~/.aws/credentials is assumed.

### [Options for --add-provider](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id20)

*--global*

Adds the provider to the global git config.

*<command>*

Provider command to invoke. When invoked the command is expected to write prohibited patterns separated by new lines to stdout. Any extra arguments provided are passed on to the command.

#### Examples

Registers a secret provider with arguments:

git secrets --add-provider -- git secrets --aws-provider

Cats secrets out of a file:

git secrets --add-provider -- cat /path/to/secret/file/patterns

## [Defining prohibited patterns](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id21)

egrep-compatible regular expressions are used to determine if a commit or commit message contains any prohibited patterns. These regular expressions are defined using the git config command. It is important to note that different systems use different versions of egrep. For example, when running on macOS, you will use a different version of egrep than when running on something like Ubuntu (BSD vs GNU).

You can add prohibited regular expression patterns to your git config using git secrets --add <pattern>.

## [Ignoring false positives](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id22)

Sometimes a regular expression might match false positives. For example, git commit SHAs look a lot like AWS access keys. You can specify many different regular expression patterns as false positives using the following command:

git secrets --add --allowed 'my regex pattern'

You can also add regular expressions patterns to filter false positives to a .gitallowed file located in the repository's root directory. Lines starting with # are skipped (comment line) and empty lines are also skipped.

First, git-secrets will extract all lines from a file that contain a prohibited match. Included in the matched results will be the full path to the name of the file that was matched, followed by ':', followed by the line number that was matched, followed by the entire line from the file that was matched by a secret pattern. Then, if you've defined allowed regular expressions, git-secrets will check to see if all of the matched lines match at least one of your registered allowed regular expressions. If all of the lines that were flagged as secret are canceled out by an allowed match, then the subject text does not contain any secrets. If any of the matched lines are not matched by an allowed regular expression, then git-secrets will fail the commit/merge/message.

Important

Just as it is a bad practice to add prohibited patterns that are too greedy, it is also a bad practice to add allowed patterns that are too forgiving. Be sure to test out your patterns using ad-hoc calls to git secrets --scan $filename to ensure they are working as intended.

## [Secret providers](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id23)

Sometimes you want to check for an exact pattern match against a set of known secrets. For example, you might want to ensure that no credentials present in ~/.aws/credentials ever show up in a commit. In these cases, it's better to leave these secrets in one location rather than spread them out across git repositories in git configs. You can use "secret providers" to fetch these types of credentials. A secret provider is an executable that when invoked outputs prohibited patterns separated by new lines.

You can add secret providers using the --add-provider command:

git secrets --add-provider -- git secrets --aws-provider

Notice the use of --. This ensures that any arguments associated with the provider are passed to the provider each time it is invoked when scanning for secrets.

## [Example walkthrough](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id24)

Let's take a look at an example. Given the following subject text (stored in /tmp/example):

This is a test!

password=ex@mplepassword

password=\*\*\*\*\*\*

More test...

And the following registered patterns:

git secrets --add 'password\s\*=\s\*.+'

git secrets --add --allowed --literal 'ex@mplepassword'

Running git secrets --scan /tmp/example, the result will result in the following error output:

/tmp/example:3:password=\*\*\*\*\*\*

[ERROR] Matched prohibited pattern

Possible mitigations:

- Mark false positives as allowed using: git config --add secrets.allowed ...

- List your configured patterns: git config --get-all secrets.patterns

- List your configured allowed patterns: git config --get-all secrets.allowed

- Use --no-verify if this is a one-time false positive

Breaking this down, the prohibited pattern value of password\s\*=\s\*.+ will match the following lines:

/tmp/example:2:password=ex@mplepassword

/tmp/example:3:password=\*\*\*\*\*\*

...But the first match will be filtered out due to the fact that it matches the allowed regular expression of ex@mplepassword. Because there is still a remaining line that did not match, it is considered a secret.

Because that matching lines are placed on lines that start with the filename and line number (e.g., /tmp/example:3:...), you can create allowed patterns that take filenames and line numbers into account in the regular expression. For example, you could whitelist an entire file using something like:

git secrets --add --allowed '/tmp/example:.\*'

git secrets --scan /tmp/example && echo $?

# Outputs: 0

Alternatively, you could allow a specific line number of a file if that line is unlikely to change using something like the following:

git secrets --add --allowed '/tmp/example:3:.\*'

git secrets --scan /tmp/example && echo $?

# Outputs: 0

Keep this in mind when creating allowed patterns to ensure that your allowed patterns are not inadvertently matched due to the fact that the filename is included in the subject text that allowed patterns are matched against.

## [Skipping validation](https://github.com/awslabs/git-secrets/blob/master/README.rst" \l "id25)

Use the --no-verify option in the event of a false positive match in a commit, merge, or commit message. This will skip the execution of the git hook and allow you to make the commit or merge.

http://cdimage.debian.org/cdimage/unofficial/non-free/firmware/bullseye/11.6.0/