

Git Submodules

On this page

- [1. Why submodules?](#)
- [2. Key concepts](#)
- [3. Cloning a repo with submodules](#)
- [4. Adding ERC-Protobufs as a submodule](#)
- [5. Updating ERC-Protobufs \(pinning a new commit\)](#)
- [6. Branches, detached HEAD, and what “pinned” means](#)
- [7. Common mistakes](#)
- [8. Command cheat sheet](#)

1) Why submodules?

A [git submodule](#) lets one repository “mount” another repository at a specific commit. That sounds fancy, but it’s really just Git saying: “this folder is a separate repo, and we are pinning it to a specific SHA because reproducibility is not optional.”

We use this for shared code/assets that:

- should be versioned and reviewed independently,
- must be *pinned* (reproducible builds, fewer “works on my machine” sightings),
- is shared across multiple firmware/software repositories.

RoboTeam example: *ERC-Protobufs* can be shared between embedded firmware, tooling, and PC-side code. Pinning ensures everyone generates/uses the same message definitions — which is what prevents “my robot speaks protobuf dialect #3” incidents.

2) Key concepts

Term	What it means	Why care about it?
------	---------------	--------------------

<code>.gitmodules</code>	A file in the parent repo that stores submodule name/path/URL.	This is what gets committed so everyone else can actually fetch the submodule without guessing.
"Pinned commit"	The parent repo records a specific commit SHA for the submodule.	Builds are reproducible; updating is an explicit change (and therefore reviewable).
Detached HEAD	By default, a submodule checks out the exact pinned commit, not a branch.	Normal. It looks scary the first time, but it just means "you're on a commit, not a branch."
<code>git submodule update</code>	Checks out the submodule commit referenced by the parent repo.	Use after switching branches or pulling changes, because submodules do not magically follow along.

3) Cloning a repo with submodules

If a repository already uses [ERC-Protobufs](#) as a submodule, you must fetch it after cloning. Otherwise Git will politely give you an empty folder and let you discover the problem at build time.

Recommended (one command)

```
git clone --recurse-submodules <PARENT_REPO_URL>
```

If you already cloned (two commands)

```
git submodule init  
git submodule update
```

Tip: Add `--recursive` if the submodule itself contains submodules:

```
git submodule update --init --recursive
```

Symptom you forgot submodules: build errors like "file not found", missing generated headers, missing `.proto` files, or empty directories where ERC-Protobufs should be.

4) Adding ERC-Protobufs as a submodule

Use this when a parent repository needs to include [ERC-Protobufs](#) for builds/code generation. This is a dependency decision, not a casual Friday activity.

Step-by-step

1. Choose where it should live in your repo, for example: `third_party/ERC-Protobufs` (or `libs/ERC-Protobufs`).
2. Add the submodule:

```
git submodule add <ERC_PROTOBUFS_REPO_URL> third_party/ERC-Protobufs
```

3. Commit the changes:

```
git add .gitmodules third_party/ERC-Protobufs
git commit -m "Add ERC-Protobufs as a submodule"
```

What gets committed? (a.k.a. “what did I just do to the repo”)

- `.gitmodules` file (submodule metadata)
- a “gitlink” entry at `third_party/ERC-Protobufs` that pins a specific commit SHA

The submodule’s full contents are not copied into the parent repo history. You’re committing a pointer, not a copy.

Protocol for RoboTeam: confirm with your lead whether the submodule path is standardized across repositories (helps tooling and scripts, and prevents everyone inventing `thirdparty/` in six different spellings).

5) Updating ERC-Protobufs (pinning a new commit)

Updating a submodule means: “the parent repo now points to a newer commit of ERC-Protobufs”. This should be done intentionally and reviewed because it can change message definitions and compatibility. Treat it like an API bump, not like updating a meme folder.

Update flow (safe + explicit)

1. Enter the submodule directory:

```
cd third_party/ERC-Protobufs
```

2. Fetch latest commits:

```
git fetch --all --tags
```

3. Check out the desired commit (or a tag):

```
git checkout <commit-sha-or-tag>
```

4. Go back to the parent repo and commit the updated pin:

```
cd ../../
git status
git add third_party/ERC-Protobufs
git commit -m "Bump ERC-Protobufs submodule to <sha-or-tag>"
```

Optional (if you want the newest remote-tracking commit): inside the parent repo:

```
git submodule update --remote --merge
```

This requires the submodule to have a branch configured; it is less explicit, so use with care (automation is great right up until it updates something you didn't mean to update).

Do not “fix” submodule issues by deleting the folder. That often creates messy diffs and makes Git sad. Use actual submodule commands instead.

6) Branches, detached HEAD and what “pinned” means

When you run `git submodule update`, Git checks out the exact commit recorded by the parent repo. This usually results in a **detached HEAD** state inside the submodule.

This is normal. A consumer repo typically should not make local changes inside the submodule. If you need to change ERC-Protobufs itself, do that in the ERC-Protobufs repository and then bump the pin in the consumer repo. Submodules are for consuming, not freestyle surgery.

How to tell what commit you are pinned to

```
# From the parent repo root:
git submodule status
```

How to see what changed after a submodule bump

```
# From the parent repo root:  
git diff --submodule
```

7) Common mistakes

“Directory is empty / looks uninitialized”

```
git submodule update --init --recursive
```

“Submodule shows changes but I didn't touch it”

Often caused by being on the wrong commit, or having local edits in the submodule. Either way, Git is not gaslighting you — something really is different.

```
cd third_party/ERC-Protobufs  
git status  
git reset --hard  
git clean -fd  
cd ../../  
git submodule update --init --recursive
```

Warning: `git reset --hard` and `git clean -fd` will delete local submodule changes. Only do this if you are sure you don't need them (i.e., you didn't secretly do work inside the submodule and forget).

“I switched branches and submodules are wrong”

```
git submodule update --init --recursive
```

“I updated the submodule but forgot to commit in the parent repo”

After updating inside the submodule, you must commit the new pin from the parent repo. Otherwise you updated your local checkout and told nobody, which is the Git equivalent of whispering into the void.

```
git add third_party/ERC-Protobufs
git commit -m "Bump ERC-Protobufs submodule"
```

8) Command cheat sheet

Clone with submodules

```
git clone --recurse-submodules <repo-url>
```

Initialize/update after cloning

```
git submodule update --init --recursive
```

Show pinned commits

```
git submodule status
```

Add ERC-Protobufs

```
git submodule add <erc-protobufs-url> third_party/ERC-Protobufs
git commit -m "Add ERC-Protobufs submodule"
```

Bump ERC-Protobufs to a specific commit/tag

```
cd third_party/ERC-Protobufs
git fetch --all --tags
git checkout <sha-or-tag>
cd ../../
git add third_party/ERC-Protobufs
git commit -m "Bump ERC-Protobufs submodule to <sha-or-tag>"
```

