我可不可以这么理解，其实git commit和git push都只是记录了源代码的修改的部分而不是全部源代码，本质上这个blob其实就是那些被修改过的源代码文件，而tree就是那些被修改的源代码文件的名单和目录，这些组成了一次commit？还有一个问题是，那这些所有的历史版本，是只保留在commit的本地里，还是一块都push到远端了？

**ChatGPT 说：**

你的理解非常接近，但有一点需要调整：**Git 并不是仅仅存储被修改的部分，而是存储整个文件的快照（完整内容）**，但它通过哈希和数据结构的优化来避免重复存储相同内容。让我们详细解释你的问题。

**1. Git Commit 和 Git Push 记录了哪些内容？**

Git **commit** 记录的是整个项目的 **快照（snapshot）**，但它不会存储所有的文件，而是：

* **对于新文件**，存储完整的文件内容（blob）。
* **对于未修改的文件**，只存储引用（不会重复存储相同内容）。
* **对于修改过的文件**，存储该文件的新快照（而不是仅仅存增量变更）。

**Commit 内部结构**

一个 commit 由以下几个部分组成：

1. **Tree（目录）**
   * 记录了当前 commit 的所有文件和目录的结构。
   * 目录内的每个文件都指向一个 blob（内容快照）。
   * **如果某个文件未修改，Git 直接复用旧的 blob，不会重新存储**。
2. **Blob（文件快照）**
   * 每个文件的内容都会被存储为 blob，由 SHA-1 哈希值标识。
   * 如果文件内容未变，则 Git 直接复用已有的 blob，不会创建新副本。
3. **Parent Commit（父提交）**
   * commit 记录了它的上一个 commit（也就是它的历史版本）。
   * 这样就形成了一个**链式版本历史**，可以回溯到任何版本。

📌 **示例：Git Commit 结构**

sql

复制编辑

commit abc123 <-- 记录一个完整的快照

├── tree xyz456 <-- 记录项目的目录结构

│ ├── blob 123abc (README.md) <-- 未修改，复用旧快照

│ ├── blob 456def (index.html) <-- 新版本快照

│ ├── tree 789ghi (src/) <-- 目录

│ ├── blob 111aaa (main.js) <-- 新版本快照

│ ├── blob 222bbb (app.js) <-- 未修改，复用旧快照

│

├── parent commit (前一个 commit)

├── 提交者、时间等元数据

🔹 **关键点：**

* README.md 和 app.js **没有变动**，所以 Git 复用了以前的 blob（不会占用额外存储）。
* index.html 和 main.js **被修改了**，所以 Git 生成新的 blob 并存储。
* commit 记录整个目录结构（tree），用于跟踪所有文件。

**2. Git Commit 是在本地保存，还是会 Push 到远端？**

Git 的版本历史（所有 commit）**默认是存储在本地仓库**，只有执行 git push 时，才会把历史提交推送到远端仓库。

**本地 vs 远端**

| **操作** | **存储位置** | **作用** |
| --- | --- | --- |
| git commit | **本地仓库** (.git 目录) | 记录变更，创建一个本地的 commit |
| git push | **远程仓库** (origin 或 GitHub 等) | 把本地 commit 发送到远程仓库 |
| git pull | **本地 & 远程** | 从远程仓库拉取最新的 commit |

**是否所有历史版本都会被 push 到远端？**

✅ **是的，默认情况下，所有 commit（快照历史）都会被 push**，但有一些特殊情况：

1. **如果 push 之前做了 rebase、squash 等操作**
   * 可能会导致 commit ID 变化，某些 commit 被合并。
2. **如果 push 发生错误，比如远程分支比本地新**
   * 可能需要 git pull --rebase 解决冲突。
3. **如果 push 只推送了部分分支**
   * 你可以 git push origin 分支名 来只推送特定的分支，而不推送所有历史。

**3. 你可以这样理解 Git 的存储方式**

* **Git 并不是直接存储“修改的部分”，而是存储整个文件的快照（blob）。**
* **Tree 记录文件结构，commit 记录 tree，形成版本链条。**
* **相同内容的文件不会重复存储，而是直接复用已有 blob，节省空间。**
* **所有 commit 默认都存在本地仓库，push 之后才会同步到远端。**

**4. 总结**

| **问题** | **答案** |
| --- | --- |
| git commit 和 git push 记录的是修改部分吗？ | ❌ 不是增量存储，而是完整的快照（但相同内容不会重复存储）。 |
| blob 是被修改的文件吗？ | ✅ blob 是文件的完整内容（快照），如果文件没变，Git 直接复用旧的 blob。 |
| tree 是文件目录的名单吗？ | ✅ tree 记录项目的目录结构，并指向 blob（文件快照）。 |
| commit 里到底存了什么？ | ✅ commit 记录 tree（目录）、parent commit（历史）、作者信息等。 |
| 历史版本是在本地存的，还是 push 到远端了？ | ✅ commit 默认在本地，执行 git push 后才同步到远程仓库。 |

**5. 推荐实践（实验）**

你可以用以下命令看看 Git 内部的实际存储：

bash

复制编辑

# 查看 commit 历史

git log --oneline --graph --all

# 查看最近一次 commit 的 tree

git ls-tree HEAD

# 查看某个文件的 blob ID

git ls-tree HEAD src/main.js

# 查看某个 commit 具体存储了哪些文件

git cat-file -p HEAD

这些命令能帮助你深入理解 Git 如何存储快照！

希望这些解释能帮到你 😃 🚀