инициализация репозитория это как бы регистрация данной папки в git, и после этого git будет понимать что такой-то проект зарегестрован в такой-то папке

| **Действие** | **Пояснение** |
| --- | --- |

|  |  |
| --- | --- |
| git init | "Эй, Git! Начни отслеживать эту папку как проект." |
|  |  |

|  |  |
| --- | --- |
| Git создаёт .git/ | В этой скрытой папке он будет хранить всю историю и внутренние данные |

|  |  |
| --- | --- |
| Теперь можно использовать git add, git commit, git status, и т.д. | Потому что Git теперь знает: эта папка — репозиторий |

После создания репоз на компе создаем его на гитхабе

После этого команда git add . (добавл. Все файлы из папки, или же можем указать конкретное имя файла вместо точки)

| **Git-команда** | **Русский аналог** | **Пояснение** |
| --- | --- | --- |
| git init | Инициализация | Регистрируем папку как Git-проект |
| git add | Отмечаю для фиксации | Говорим Git: «Вот эти файлы я хочу сохранить» |
| git commit -m "..." | Фиксация + комментарий | Создаём «снимок» изменений с подписью |
| git status | Проверка состояния | Что изменилось, что отслеживается, что ещё не сохранено |
| git log | История изменений | Список всех фиксаций (коммитов) с комментариями |
| git diff | Сравнение изменений | Что именно изменилось в коде |
| git clone URL | Скопировать проект | Копируем чужой (или свой) проект с GitHub на компьютер |
| git remote add origin URL | Добавить удалённый репозиторий | Связать папку на компьютере с GitHub-версией |
| git push | Отправить на GitHub | Загружаем изменения в удалённый репозиторий |
| git pull | Загрузить с GitHub | Получаем свежие изменения с GitHub |
| git branch | Список веток | Посмотреть или создать ветку |
| git checkout имя\_ветки | Перейти в другую ветку | Сменить рабочее направление разработки |
| git merge имя\_ветки | Объединить ветки | Совмещаем изменения из одной ветки в другую |

git reset --hard HEAD^1 - жесткий зброс на один коммит назад

git reset --soft HEAD^1 - мягкий зброс (созданный коммит становиться просто незакоммиченным в в стейдж зоне)

git reset . – извлекает изменения из стейдж зоны

git checkout -- . удаляет все измениния после извлечения з стейдж зоны

branch ветки – нужны чтоб экспериментировать с кодом не нанося вред главному коду, создаешь отдельную ветку – экспериментируешь и если все работает сливаешь экспериментальную ветку с главной, таким образом эксперименты в коде не наносят вряд главному коду.

**# 🧾 Git Шпаргалка: Как удалить последний коммит и переписать историю**

**## 🎯 Цель:**

Удалить последний коммит из ветки (локально и на удалённом репозитории).

---

**## ✅ Шаги по порядку:**

**### 1. Проверить текущие ветки и коммиты:**

```bash

git branch -v

```

---

**### 2. Найти нужный коммит (например, в develop):**

```bash

git branch develop

```

Или просто откатиться на один коммит назад:

```bash

git reset --hard HEAD^

```

---

**### 3. Выполнить жёсткий откат:**

```bash

git reset --hard develop

```

ИЛИ

```bash

git reset --hard <commit-hash>

```

> ⚠️ `--hard` удаляет изменения в файлах. Будь внимателен!

---

**### 4. Принудительно запушить изменения:**

```bash

git push origin main --force

```

> ⚠️ `--force` полностью перезаписывает историю удалённой ветки!

---

**## ✅ Результат:**

- Локальная история откатилась к нужному коммиту.

- Удалённая история (`origin/main`) синхронизирована.

- Лишний коммит удалён.

git checkout -b develop - создание и переход в новую ветку под названием develop