GIT

- Как удалить историю GIT
- Как откатить последний коммит
- Полное слияние ветки test в main с удалением test
- Процесс разработки и интеграции изменений в основную ветку

Как удалить историю GIT

1. **Создайте новый начальный коммит:**

- Переключитесь на новую ветку без истории:
  `bash git checkout --orphan new-branch`
- Добавьте все файлы в новый коммит:
  `bash git add -A`
- Создайте начальный коммит:
  `bash git commit -m "Начальный коммит"`

---

2. **Удалите старую ветку `main`:**

- Удалите текущую ветку `main`:
  `bash git branch -D main`

---

3. **Переименуйте новую ветку в `main`:**

- Переименуйте новую ветку (`new-branch`) в `main`:
  `bash git branch -m main`

---

4. **Принудительно отправьте изменения в удаленный репозиторий:**

- Принудительно замените историю в удаленном репозитории:
  `bash git push -f origin main`

---

5. **Очистите локальный репозиторий (опционально):**

Если вы хотите очистить локальный репозиторий от старой истории, выполните:

```
git reflog expire --expire=now --all
git gc --prune=now --aggressive
```

---

6. **Обновите локальный репозиторий (если необходимо):**

Если у вас есть другие клоны репозитория, обновите их:

```
git fetch --all
git reset --hard origin/main
```

Важно:

- **Это действие необратимо**: после удаления истории коммитов восстановить их будет невозможно.
- **Уведомите коллег**: если вы работаете в команде, убедитесь, что все участники знают о предстоящих изменениях, так как это может повлиять на их локальные репозитории.

Теперь ваш удаленный репозиторий будет содержать только один коммит с текущим состоянием проекта.

Как откатить последний коммит

Инструкция для отката последнего коммита, в зависимости от ситуации.

---

1. Откат последнего коммита без публикации

Если коммит ещё **не был отправлен** в удалённый репозиторий:

- Удалить коммит, сохранив изменения в индексе (стадии):

```
git reset --soft HEAD~1
```

- Удалить коммит и убрать изменения из индекса, оставив их в рабочей папке:

```
git reset --mixed HEAD~1
```

- Удалить коммит и изменения полностью (⚠ безвозвратно):

```
git reset --hard HEAD~1
```

---

2. Откат опубликованного коммита (без изменения истории)

Если коммит уже **отправлен в удалённый репозиторий**:

- Создать новый коммит, который отменяет изменения предыдущего:

```
git revert HEAD
```

После этого отправить изменения в репозиторий:

```
git push
```

---

3. Жёсткий откат опубликованного коммита (с изменением истории)

Если нужно полностью удалить коммит (и изменения), и ты готов "переписать" историю репозитория:

- Удалить последний коммит и изменения:

```
git reset --hard HEAD~1
```

- Принудительно отправить изменения:

```
git push --force
```

⚠ **Предупреждение:** Этот способ может привести к проблемам для других разработчиков, так как история репозитория будет изменена.

Вывод

Выбирай способ в зависимости от ситуации: для локальных коммитов подойдут `reset`, для опубликованных — безопасный `revert` или жёсткий `reset --hard`. Если есть сомнения, уточни детали своей задачи!

Полное слияние ветки test в main с удалением test

□□ Задача

У вас есть две ветки: `main` и `test`. Вам нужно:

- **Полностью заменить содержимое `main` на текущее состояние `test`.**
- **Удалить ветку `test` после слияния.**

□ Пошаговое решение

1□ **Переключаемся в `main`**

```
git checkout main
```

2□ **Обновляем `main` перед слиянием** (если работаем с удалённым репозиторием)

```
git pull origin main
```

3□ **Полностью заменяем `main` на `test`**

```
git reset --hard test
```

4⃣ Принудительно отправляем изменения в удалённый репозиторий

```
git push --force origin main
```

⚠️ **Внимание!** Использование `--force` полностью заменит удалённую ветку `main` на состояние `test`. Будьте уверены, что ничего важного не потеряете!

5⃣ Удаляем ветку `test` локально

```
git branch -D test
```

6⃣ Удаляем ветку `test` на удалённом сервере

```
git push origin --delete test
```

🔍 Объяснение команд

Команда	Описание
`git checkout main`	Переключаемся в ветку `main`.
`git pull origin main`	Загружаем актуальные изменения `main`.
`git reset --hard test`	Полностью заменяем содержимое `main` на `test`.
`git push --force origin main`	Принудительно обновляем `main` на удалённом сервере.
`git branch -D test`	Удаляем локальную ветку `test`.
`git push origin --delete test`	Удаляем ветку `test` в удалённом репозитории.

□□ Итог

- Ветка **`main` станет идентичной `test`**.
- Ветка **`test` будет удалена локально и на сервере**.

⚠ Важно!

- **Если `main` используется другими разработчиками**, предупредите их перед выполнением `git push --force origin main`, так как это **перезапишет всю историю**.
- **Более безопасный способ**, если `main` нельзя перезаписывать принудительно:

```
git checkout main
git merge --strategy=ours test
git push origin main
```

Процесс разработки и интеграции изменений в основную ветку

Процесс разработки и интеграции изменений в основную ветку (`main` или `master`) в GIT обычно включает несколько этапов. Описание ниже основано на стандартном Git Flow, но может адаптироваться под конкретные требования.

---

1. Планирование и создание задачи

Прежде чем приступить к разработке, необходимо:

- Определить задачу (например, новую фичу, исправление бага, рефакторинг).
- Завести тикет в системе управления задачами (Jira, YouTrack, GitLab Issues и т.д.).
- Определить, в какой ветке будет вестись работа.

---

2. Создание новой ветки

Разработчик создает новую ветку на основе последней актуальной версии `main`:

```
git checkout main
git pull origin main
git checkout -b feature/new-feature
```

Где `feature/new-feature` — имя ветки, соответствующее задаче.

Если работа ведется с багфиксом, может использоваться ветка `bugfix/bug-id`.

3. Разработка и коммиты

Разработчик пишет код, делает изменения и фиксирует их:

```
git add .
git commit -m "Добавлена новая функциональность"
```

Рекомендуется делать осмысленные коммиты и использовать семантические сообщения (`fix:`, `feat:`, `refactor:`, `docs:`, `test:`).

---

4. Локальное тестирование

Перед отправкой изменений важно:

- Запустить юнит-тесты.
- Протестировать код в рабочем окружении (локальном, dev).
- Убедиться, что не нарушена работа других модулей.

---

5. Отправка изменений в удаленный репозиторий

После завершения работы над задачей:

```
git push origin feature/new-feature
```

Создается **Merge Request (Pull Request)** в GitLab/GitHub.

---

6. Код-ревью

- Коллеги проверяют код, дают замечания.
- Разработчик вносит правки, дополняет тестами (если нужно).

- Повторяет `git push` до окончательного одобрения.

---

7. Интеграция в основную ветку

После успешного ревью:

- Ветка `feature/new-feature` вливается в `develop` (или сразу в `main`, если процесс без `develop`).
- Используется `merge` или `rebase` (в зависимости от стратегии проекта):

```
git checkout main
git pull origin main
git merge feature/new-feature
git push origin main
```

- В некоторых случаях может использоваться **Squash & Merge** для объединения коммитов.

---

8. Автоматизация через CI/CD

После слияния в `main` могут автоматически запускаться:

- Автоматические тесты.
- Деплой на тестовый или production сервер (в зависимости от настроек CI/CD).

---

9. Удаление ненужных веток

После успешного слияния:

```
git branch -d feature/new-feature
git push origin --delete feature/new-feature
```

Это помогает поддерживать чистоту репозитория.

10. Деплой и релиз

Если проект использует версионирование:

- Генерируется новая версия (`git tag v1.2.3`).
- Запускается CI/CD пайплайн, который деплоит код.

Итог

Этот процесс помогает обеспечить стабильность разработки, контроль качества и чистоту репозитория. В реальных проектах могут быть добавлены дополнительные этапы, например, работа с `hotfix`-ветками, релизными ветками (`release/`), более строгий контроль тестирования и деплоя.