Основные команды Git с примерами и описанием:

---

1. Настройка Git

```
git config --global user.name "Ваше Имя"
git config --global user.email "ваш@email.com"
```

- Устанавливает имя и email для всех коммитов.

---

2. Инициализация репозитория

```
git init
```

- Создает новый локальный репозиторий в текущей папке.

---

3. Клонирование репозитория

```
git clone https://github.com/user/repo.git
```

- Копирует удаленный репозиторий на ваш компьютер.

---

4. Проверка состояния

```
git status
```

- Показывает измененные, добавленные и удаленные файлы.

---

5. Добавление файлов в индекс (staging)

```
git add file.txt       # Добавляет конкретный файл
git add .              # Добавляет все измененные файлы
git add *.js           # Добавляет все JS-файлы
```

- Подготавливает файлы к коммиту.

---

6. Фиксация изменений (коммит)

```
git commit -m "Описание изменений"
```

- Сохраняет изменения в локальном репозитории.

---

7. Просмотр истории коммитов

```
git log                # Полная история
git log --oneline      # Краткая история (хэш + сообщение)
git log -p             # История с изменениями в коде
```

- Показывает историю коммитов.

---

8. Ветвление

```
git branch                  # Список веток
git branch new-feature      # Создает новую ветку
git checkout new-feature    # Переключается на ветку
git checkout -b new-branch  # Создает и переключается
```

- Работа с ветками.

---

9. Слияние веток

```
git checkout main           # Переход в основную ветку
git merge new-feature       # Вливает `new-feature` в `main`
```

- Объединяет изменения из одной ветки в другую.

---

10. Удаленные репозитории

```
git remote -v                   # Список подключенных репозиториев
git remote add origin URL       # Добавляет удаленный репозиторий
git push -u origin main         # Отправляет изменения на сервер
git pull origin main            # Забирает изменения с сервера
```

- Работа с GitHub/GitLab и другими хостами.

---

11. Отмена изменений

```
git restore file.txt          # Отменяет изменения в файле (до staging)
git reset --hard HEAD        # Отменяет все локальные изменения (осторожно!)
git revert commit-hash       # Создает коммит, отменяющий изменения
```

- Возвращает проект к предыдущему состоянию.

---

12. Временное сохранение изменений (stash)

```
git stash            # Временно сохраняет изменения
git stash pop       # Возвращает изменения
```

- Полезно при срочном переключении между ветками.

---

13. Теги (версии)

```
git tag v1.0.0              # Создает тег
git push origin v1.0.0      # Отправляет тег на сервер
```

- Используется для отметки версий (например, релизов).

---

14. Просмотр изменений

```
git diff                    # Показывает незакоммиченные изменения
git diff --staged           # Показывает изменения в staged-файлах
```

- Сравнивает изменения в коде.

---

15. Переименование ветки

```
git branch -m old-name new-name  # Локально
git push origin :old-name        # Удаляет старую ветку на сервере
git push origin new-name         # Пушит новую ветку
```

Пример рабочего процесса

```
git clone https://github.com/user/project.git
cd project
git checkout -b feature
# Редактируем файлы...
git add .
git commit -m "Добавил новую функцию"
git push origin feature
```

Сложные, но полезные команды Git: `rebase`, `cherry-pick`, `reset`, `reflog` и другие. Они помогают в нестандартных ситуациях, но требуют аккуратности.

---

1. Git Rebase — «перезапись истории»

Зачем:

- Переместить ваши коммиты на верх чужой ветки (например, `main`), чтобы история была линейной.
- Объединить несколько коммитов в один (squash).
- Изменить старые коммиты (сообщения, содержимое).

Примеры:

▶ Перебазирование ветки на `main`:

```
git checkout feature
git rebase main
```

- Берет ваши коммиты из `feature` и применяет их поверх актуального `main`.
- Если были конфликты, Git предложит их разрешить (аналогично слиянию).

▶ Интерактивный rebase (изменить историю):

```
git rebase -i HEAD~3  # Редактирует последние 3 коммита
```

Откроется редактор с вариантами действий:

- `pick` — оставить коммит как есть.
- `squash` — объединить с предыдущим.
- `edit` — изменить коммит.
- `drop` — удалить коммит.

▶ Опасность:

- Нельзя делать `rebase` для коммитов, уже отправленных в общий репозиторий (это перепишет историю для всех!).

---

2. Git Cherry-Pick — «перенос коммитов»

Зачем:

- Взять один конкретный коммит из другой ветки и применить его к текущей.

Пример:

```
git checkout main
git cherry-pick abc123  # Применяет коммит с хэшем `abc123` в `main`
```

- Полезно, если нужно перенести исправление бага из `feature` в `main`, не сливая всю ветку.

Конфликты:

- Если изменения конфликтуют, Git предложит разрешить их вручную (как при `merge`).

---

3. Git Reset — «отмена коммитов»

Зачем:

- Отменить локальные изменения или коммиты.

Типы reset:

▶ `--soft` — отменяет коммит, но оставляет изменения в staging:

```
git reset --soft HEAD~1  # Отменяет последний коммит, но изменения остаются
```

▶ `--mixed` (по умолчанию) — отменяет коммит и снимает изменения с staging:

```
git reset HEAD~1  # Коммит отменён, изменения остались в рабочей директории
```

▶ `--hard` — полностью удаляет коммит и изменения (осторожно!):

```
git reset --hard HEAD~1  # Всё пропало (если не было backup)
```

---

4. Git Reflog — «спасение потерянных коммитов»

Зачем:

- Показывает всю историю действий в Git (даже удалённые коммиты).
- Помогает восстановить данные после `reset --hard` или удаления ветки.

Пример:

```
git reflog  # Показывает историю перемещений HEAD
```

Вывод:

```
abc123 HEAD@{0}: commit: Добавил новую функцию
def456 HEAD@{1}: reset: moving to HEAD~1
```

Чтобы вернуть удалённый коммит:

```
git checkout abc123  # Переходим к коммиту
git checkout -b saved-branch  # Создаём ветку для сохранения
```

---

5. Git Stash — временное сохранение изменений

Зачем:

- Если нужно переключиться на другую ветку, но текущие изменения не готовы для коммита.

Примеры:

```
git stash           # Сохраняет изменения в «стэш»
git stash list      # Показывает список сохранений
git stash apply     # Возвращает последние изменения
git stash pop       # Возвращает и удаляет из стэша
git stash drop      # Удаляет последний стэш
```

---

6. Git Bisect — поиск «когда сломалось»

Зачем:

- Найти коммит, в котором появился баг, методом бинарного поиска.

Пример:

```
git bisect start
git bisect bad       # Текущий коммит с багом
git bisect good abc123  # Коммит, где всё работало
```

Git будет перемещаться между коммитами, а вы проверяете:

```
git bisect good  # Если бага нет
git bisect bad   # Если баг есть
```

В конце Git укажет проблемный коммит.

---

7. Подводные камни

- `rebase` **vs** `merge`**:**
  - `rebase` делает историю чище, но опасен для общих веток.
  - `merge` безопаснее, но создаёт «лишние» коммиты слияния.
- `force push` **(**`git push -f`**):**
  - Перезаписывает историю на сервере. Может сломать репозиторий для других!
- **Удалённые ветки:**

```
git push origin --delete old-branch  # Удаляет ветку на сервере
```

Итог

Команда	Когда использовать
`rebase`	Чтобы «переставить» свои коммиты поверх чужой ветки (перед пул-реквестом).
`cherry-pick`	Чтобы перенести один коммит из другой ветки.
`reset`	Чтобы отменить локальные коммиты (осторожно с `--hard`!).
`reflog`	Чтобы найти потерянные коммиты после `reset` или удаления ветки.
`stash`	Чтобы временно сохранить изменения без коммита.
`bisect`	Чтобы найти коммит, в котором появился баг.

Эти команды — мощные инструменты, но требуют понимания. Перед сложными операциями делайте **backup** (`git clone --mirror` или копируйте папку `.git`).