Расширенный мини-курс по командам Git

Полное руководство по синтаксису, откату изменений и рабочим процессам Git

Создано для глубокого изучения системы контроля версий Git

Содержание

1 Что такое Git?

Git — распределённая система контроля версий для отслеживания изменений в коде, совместной работы и управления версиями. Представьте Git как библиотеку: репозиторий — книга, коммиты — главы, ветки — черновики, а удалённый сервер — облако.

1.1 Зачем нужен Git?

- **История**: Сохраняет все правки.
- **Командная работа**: Упрощает слияние кода.
- **Ветвление**: Параллельные задачи.
- **Откат**: Восстановление версий.

2 Установка Git

Установка для Ubuntu/Debian.

2.1 Шаги установки

1. Обновите пакеты:

```
    sudo apt update
```

2. Установите Git:

```
    sudo apt install git
```

3. Проверьте версию:

```
    git --version
```

4. Настройте пользователя:

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

Что происходит? Git готов, пользователь настроен.

3 Базовая терминология

Репозиторий Папка с `.git`.

Коммит Снимок изменений.

Ветка Параллельная разработка.

HEAD Текущий коммит/ветка.

Индекс Промежуточная область.

Удалённый репозиторий Копия на сервере.

4 Синтаксис команд Git

4.1 Структура команды

- **Формат**: `git [команда] [опции] [аргументы]`.

```
    git commit -m "Исправлен баг"

```

4.2 Правила написания

- **Регистр**: `git commit`, не `git Commit`.
- **Кавычки**: `git branch "feature/login"`.
- **Проверка**: `git help commit`.

Аналогия: Команды — как рецепты: точность важна.

5 Базовые команды

5.1 Пример сценария

```
git init
echo "Проект" > README.md
git add README.md
git commit -m "Начальный коммит"
git status
git log --oneline --graph
```

5.2 Подробный анализ

- `git init`: Создаёт `.git`.
- `git add README.md`: Добавляет файл в индекс.
- `git commit -m "Начальный коммит"`: Фиксирует снимок.
- `git status`: Показывает состояние.
- `git log -oneline -graph`: История с ветками.

Аналогия: `git init` — новая книга, `git commit` — глава.

6 Работа с ветками

6.1 Пример сценария

```
git branch feature
git checkout feature
echo "Функция" >> app.js
git add app.js
git commit -m "Новая функция"
git checkout main
git merge feature
```

6.2 Подробный анализ

- `git branch feature`: Создаёт ветку.
- `git checkout feature`: Переключает на ветку.
- `git merge feature`: Сливает изменения.

7 Работа с удалёнными репозиториями

7.1 Пример сценария

```
git clone https://github.com/user/repo.git
cd repo
git remote add upstream https://github.com/original/repo.git
git fetch upstream
git checkout main
git merge upstream/main
git push origin main
git pull --rebase origin main
```

7.2 Подробный анализ

- `git clone`: Копирует репозиторий.
- `git remote add upstream`: Добавляет удалённый репозиторий.
- `git fetch upstream`: Загружает данные.
- `git pull –rebase`: Ребазает изменения.

8 Продвинутые команды

8.1 Пример сценария

```
git checkout feature
git cherry-pick 123abc
git stash
git stash pop
git tag v1.0
git diff main feature
git blame README.md
```

8.2 Подробный анализ

- `git cherry-pick 123abc`: Копирует коммит.
- `git stash`: Сохраняет изменения.
- `git stash pop`: Восстанавливает изменения.
- `git tag v1.0`: Создаёт тег.
- `git diff main feature`: Сравнивает ветки.
- `git blame README.md`: Показывает авторов.

9 Откат изменений

Откат изменений позволяет исправить ошибки или вернуться к прошлому состоянию.

9.1 Пример сценария

Откат к коммиту, отмена коммита, восстановление файла.

```
# Откат последнего коммита, сохраняя изменения
git reset --soft HEAD^
# Откат к конкретному коммиту, удаляя изменения
git reset --hard 123abc
# Создание коммита, отменяющего другой
git revert 456def
# Восстановление файла из коммита
git checkout 123abc -- file.txt
# Проверка истории операций
git reflog
git checkout 789ghi
```

9.2 Подробный анализ

- `git reset –soft HEAD^`:
  - **Что делает?** Откатывает последний коммит, сохраняя изменения в индексе.
  - **Зачем?** Исправляет коммит, не теряя код.
  - **Как работает?** Перемещает HEAD и ветку на предыдущий коммит, оставляя изменения в индексе и рабочей директории.
  - **Ошибки**: Неправильный хеш или указатель (например, `HEAD^^`) могут откатить слишком далеко.
- `git reset –hard 123abc`:
  - **Что делает?** Откатывает ветку к коммиту `123abc`, удаляя все последующие изменения.
  - **Зачем?** Полностью возвращает проект к прошлому состоянию.
  - **Как работает?** Перемещает HEAD, сбрасывает индекс и рабочую директорию до состояния коммита `123abc`.
  - **Ошибки**: Потеря несохранённых изменений; используйте с осторожностью. Восстановить можно через `git reflog`.
- `git revert 456def`:
  - **Что делает?** Создаёт новый коммит, отменяющий изменения коммита `456def`.
  - **Зачем?** Безопасно отменяет изменения, сохраняя историю.
  - **Как работает?** Анализирует коммит `456def`, создаёт обратные изменения и фиксирует их в новом коммите.
  - **Ошибки**: Конфликты при слиянии требуют ручного разрешения:

```
        git add .
        git revert --continue

```

- `git checkout 123abc - file.txt`:
  - **Что делает?** Восстанавливает файл `file.txt` из коммита `123abc`.
  - **Зачем?** Возвращает конкретный файл к прошлому состоянию.
  - **Как работает?** Копирует версию файла из коммита в рабочую директорию и индекс.
  - **Ошибки**: Неправильный хеш или имя файла вызовут ошибку.
- `git reflog`:
  - **Что делает?** Показывает историю операций с HEAD.
  - **Зачем?** Восстанавливает потерянные коммиты после `reset`.
  - **Как работает?** Выводит хеши и действия (commit, reset, checkout).
  - **Ошибки**: Нет, команда безопасна.
- `git checkout 789ghi`:
  - **Что делает?** Переключает HEAD на коммит `789ghi`.
  - **Зачем?** Восстанавливает состояние после ошибочного `reset`.
  - **Как работает?** Переводит репозиторий в detached HEAD, позволяя создать новую ветку.
  - **Ошибки**: Неправильный хеш вызывает ошибку.

Аналогия: `git reset -soft` — стирание последней страницы, но черновик остаётся; `git reset -hard` — сжигание всех страниц после нужной; `git revert` — добавление страницы с исправлениями; `git checkout - file` — копирование старой страницы; `git reflog` — журнал правок.

Шаги:

1. Проверьте историю:

```
git log --oneline
```

2. Выполните откат, например:

```
git reset --hard 123abc
```

3. Если нужен восстановленный коммит:

```
git reflog
git checkout 789ghi
git branch recovered
```

10 Полезные советы

- **.gitignore**:

```
node_modules/
*.log
```

- **Алиасы**:

```
git config --global alias.lg "log --oneline --graph"
```

11 Примеры сценариев

11.1 Откат последнего коммита

```
git reset --soft HEAD^
git commit -m "Исправленный коммит"
```

11.2 Восстановление удалённой ветки

```
git reflog
git checkout 123abc
git branch recovered
git push origin recovered
```

12 Справочник команд Git

Ниже приведены все рассмотренные команды с флагами, примерами и описаниями.

Команда	Пример	Описание
`git init`	`git init`	Инициализирует новый репозиторий, создавая `.git`.
`git add [file]`	`git add README.md`	Добавляет файл в индекс для коммита.
`git commit -m [msg]`	`git commit -m "Начальный коммит"`	Фиксирует изменения в репозитории.
`git commit –amend`	`git commit –amend -m "Исправлено"`	Исправляет последний коммит.
`git status`	`git status`	Показывает состояние файлов (изменены, в индексе).
`git log –oneline –graph`	`git log –oneline –graph`	Показывает историю коммитов с ветками.
`git branch [name]`	`git branch feature`	Создаёт новую ветку.
`git checkout [branch]`	`git checkout feature`	Переключает на ветку или коммит.
`git checkout [commit] – [file]`	`git checkout 123abc – file.txt`	Восстанавливает файл из коммита.
`git merge [branch]`	`git merge feature`	Сливает ветку в текущую.
`git clone [url]`	`git clone https://github.com/user/repo.git`	Клонирует удалённый репозиторий.
`git remote add [name] [url]`	`git remote add upstream https://github.com/original/repo.git`	Добавляет удалённый репозиторий.
`git fetch [remote]`	`git fetch upstream`	Загружает данные из удалённого репозитория.

`git push [remote] [branch]`	`git push origin main`	Отправляет коммиты в удалённый репозиторий.
`git push –force`	`git push origin –force`	Принудительно обновляет удалённую ветку.
`git push [remote] [tag]`	`git push origin v1.0`	Отправляет тег в удалённый репозиторий.
`git pull [remote] [branch]`	`git pull origin main`	Загружает и сливает изменения.
`git pull –rebase`	`git pull –rebase origin main`	Загружает и ребазает изменения.
`git branch –set-upstream-to=[remote]/[branch]`	`git branch –set-upstream-to=origin/main`	Связывает локальную ветку с удалённой.
`git cherry-pick [commit]`	`git cherry-pick 123abc`	Копирует коммит в текущую ветку.
`git stash`	`git stash`	Сохраняет изменения в стек.
`git stash list`	`git stash list`	Показывает стек stash.
`git stash pop`	`git stash pop`	Применяет и удаляет последний stash.
`git tag [name]`	`git tag v1.0`	Создаёт тег на текущем коммите.
`git diff [branch1] [branch2]`	`git diff main feature`	Сравнивает ветки.
`git blame [file]`	`git blame README.md`	Показывает авторов строк файла.
`git clean -fd`	`git clean -fd`	Удаляет неотслеживаемые файлы и папки.
`git bisect start`	`git bisect start`	Начинает поиск бага.
`git bisect bad/good`	`git bisect bad`	Отмечает коммит как проблемный/рабочий.
`git bisect run [script]`	`git bisect run ./test.sh`	Автоматизирует поиск бага.
`git bisect reset`	`git bisect reset`	Завершает bisect.
`git worktree add [path] [branch]`	`git worktree add ../hotfix v2.0`	Создаёт новую рабочую директорию.
`git worktree remove [path]`	`git worktree remove ../hotfix`	Удаляет рабочую директорию.
`git reset –soft [commit]`	`git reset –soft HEAD^`	Откатывает коммит, сохраняя изменения в индексе.

`git reset –hard [commit]`	`git reset –hard 123abc`	Откатывает к коммиту, удаляя изменения.
`git revert [commit]`	`git revert 456def`	Создаёт коммит, отменяющий изменения.
`git reflog`	`git reflog`	Показывает историю операций с HEAD.