Git и GitHub | Я.Шпора

Установка и первоначальная настройка GIT

Проверьте, не установлен ли GIT на ваш компьютер:

```
git --version
```

Установка на Windows

Загрузите Git для Windows с официального сайта и установите его.

Установка на macOS

Установите git через пакетный менеджер **Homebrew**:

```
brew install git
```

Установка на Linux

Установите git через пакетный менеджер APT.

```
sudo apt-get install git
```

Стартовая настройка

Первоначальные настройки программы GIT — внесение персональной информации, которая будет отображаться для других пользователей:

```
# Указать имя, которым будут подписаны коммиты.
git config --global user.name "Your Name"
# Указать электропочту, которая будет в описании коммита.
git config --global user.email "e@w.com"
```

Эти настройки выполняются лишь один раз.

Консольные команды

Создать новый репозиторий

Репозиторий — директория на компьютере или на удалённом сервере, изменения в которой отслеживаются программой GIT.

Команда	Описание
`git init`	Объявить репозиторием текущую (существующую) директорию.
`git init folder_name`	Создать новую директорию *folder_name* и объявить её репозиторием.

Клонирование репозитория

Клонирование репозитория — это создание на локальном компьютере копии репозитория, размещённого на удалённом сервере. GIT будет синхронизировать состояния локального и удалённого репозиториев.

```
# Клонировать удалённый репозиторий в одноимённую директорию.
git clone <https://github.com/user_login/repo_name.git>
# В директории, из которой выполнена команда,
# будет создана папка repo_name с репозиторием.

# Клонировать удалённый репозиторий в директорию folder_name.
git clone <https://github.com/user_login/repo_name.git> folder_
name
# В директории, из которой выполнена команда,
# будет создана папка folder_name с репозиторием repo_name.
```

Вместо имени директории *folder_name* можно написать точку `.`, она означает директорию, из которой выполнена команда.

Команда `git clone <url> .` склонирует репозиторий в текущую директорию.

Просмотр изменений репозитория

Команда	Описание
`git status`	Показать текущее состояние репозитория (например — отслеживаемые, изменённые и новые файлы).
`git diff`	Сравнить рабочую директорию и индекс (неотслеживаемые файлы игнорируются).

Добавление изменений в индекс

Индекс хранит список и текущее состояние всех файлов и директорий, которые отслеживает GIT в репозитории.

Команда	Описание
`git add .`	Добавить в индекс все новые, изменённые, удалённые файлы из текущей директории и её поддиректорий.
`git add text.txt`	Добавить в индекс файл *text.txt.*

Удаление изменений из индекса

Если необходимо убрать файлы из индекса, то можно использовать следующие команды.

Команда	Описание
`git reset`	Удалить из индекса все добавленные в него изменения (в рабочей директории все изменения сохранятся). Эта команда — антипод `git add`: она применяется, чтобы снять индексацию (*unstaged*) всех изменений, которые были добавлены в индекс (*staged*) с помощью `git add`.
`git reset readme.txt`	Удалить из индекса все изменения файла *readme.txt* , внесённые после последнего коммита (в самом файле изменения сохранятся).

Отмена изменений

Чтобы вернуться к версии в последнем коммите:

Команда	Описание
`git checkout text.txt`	**ОПАСНО**: отменить в файле все изменения, сделанные после последнего коммита. Состояние файла в рабочей директории «откатится» к последнему коммиту.
`git reset --hard`	**ОПАСНО:** отменить все изменения, сделанные после последнего коммита, во всех файлах. Незакомиченные изменения будут удалены из индекса и из рабочей директории. Состояние всех файлов в рабочей директории «откатится» к последнему коммиту.

Коммиты

Коммит — сохранение текущего состояния репозитория.

Команда	Описание
`git commit -m "Комментарий к коммиту"`	Зафиксировать в коммите проиндексированные изменения (закоммитить), добавить сообщение к коммиту.
`git commit -a -m "Комментарий к коммиту"`	Зафиксировать отслеживаемые файлы (именно отслеживаемые, но не новые файлы) и закоммитить, добавить сообщение к коммиту.

Отмена коммитов и перемещение по истории

Отмена коммитов, отправленных в удалённый репозиторий, выполняется созданием нового коммита, в котором «стираются» изменения, которые внесены в отменяемых коммитах. Это позволяет избежать проблем с историей разработки у других участников проекта.

Git опирается на работу с **указателями**. Самый часто используемый — `HEAD`. Это указатель на текущий коммит.

Создание коммита, отменяющего изменения, внесённые последним коммитом:

```
git revert HEAD # Откатит состояние репозитория к коммиту, на к
оторый указывает HEAD.
```

Вместо `HEAD` можно указать id коммита, к которому нужно откатить состояние репозитория.

Отмена изменений в файле

Откат файла в рабочей директории (в примере — файл `index.html`) к состоянию определённого коммита (в примере — коммит с хешем `5589877`):

```
git checkout 5589877 index.html
```

Работа с коммитами и ветками

Команда	Описание
`git checkout b9533bb`	Переключиться на коммит с хешем `b9533bb` (указатель HEAD переместится на указанный коммит, рабочая директория вернётся к состоянию этого коммита).
`git checkout dev`	Переключиться на ветку `dev`, на самый свежий её коммит.

Команда	Описание
`git branch`	Показать список веток.
`git branch new_branch`	Создать ветку с именем `new_branch`. Новая ветка начнётся с текущего коммита.
`git checkout -b new_branch`	Создать ветку с именем `new_branch` и переключиться на неё.
`git checkout -b new_branch 5589877`	Создать ветку с именем `new_branch`, начинающуюся с коммита с хешем `5589877`, и переключиться на эту ветку.
`git merge dev`	Влить в текущую ветку данные из ветки `dev`.

Удалённые репозитории

Команда	Описание
`git remote -v`	Показать список удалённых репозиториев, связанных с локальным.
`git remote add origin <URL>`	Подключить к локальному репозиторию удалённый репозиторий с указанным URL; дать этому репозиторию служебное имя `origin`.
`git fetch origin`	Скачать все ветки из удаленного репозитория с именем origin, но не сливать со своими (локальными) ветками.
`git push origin main`	Отправить в удалённый репозиторий с именем origin данные своей ветки main
`git pull origin`	Влить изменения из всех веток удалённого репозитория в соответствующие локальные ветки.
`git pull origin main`	Влить изменения из указанной ветки удалённого репозитория в соответствующую локальную ветку.

Работа нескольких пользователей в одном репозитории

Рассмотрим сценарий работы тимлида и нескольких разработчиков с общим репозиторием.

Тимлид создаёт репозиторий и добавляет остальных разработчиков в коллабораторы этого репозитория. Для этого нужно в интерфейсе GitHub в настройках репозитория в левом меню выбрать пункт **Collaborators,** указать GitHub-логин разработчика и пригласить его.

Пусть на момент создания в репозитории есть единственный файл *README.md* с таким содержимым:

```
# Название раздела
Текст раздела
```

В репозитории есть единственная ветка **main.**

Обычно разработка ведется не в ветке **main**, а в ветке **develop**. Тимлид создаёт новую ветку для разработчиков:

```
git checkout -b develop  # Создали ветку develop и сразу переключи
```

Чтобы сделать ветку **develop** доступной для других разработчиков на GitHub, тимлид должен локально выполнить команду:

```
git push --set-upstream origin develop
```

Соразработчики должны самостоятельно создавать новые ветки под каждую фичу. Если ветка создана для разработки нового функционала проекта, её имя может начинаться со слова `feature`.

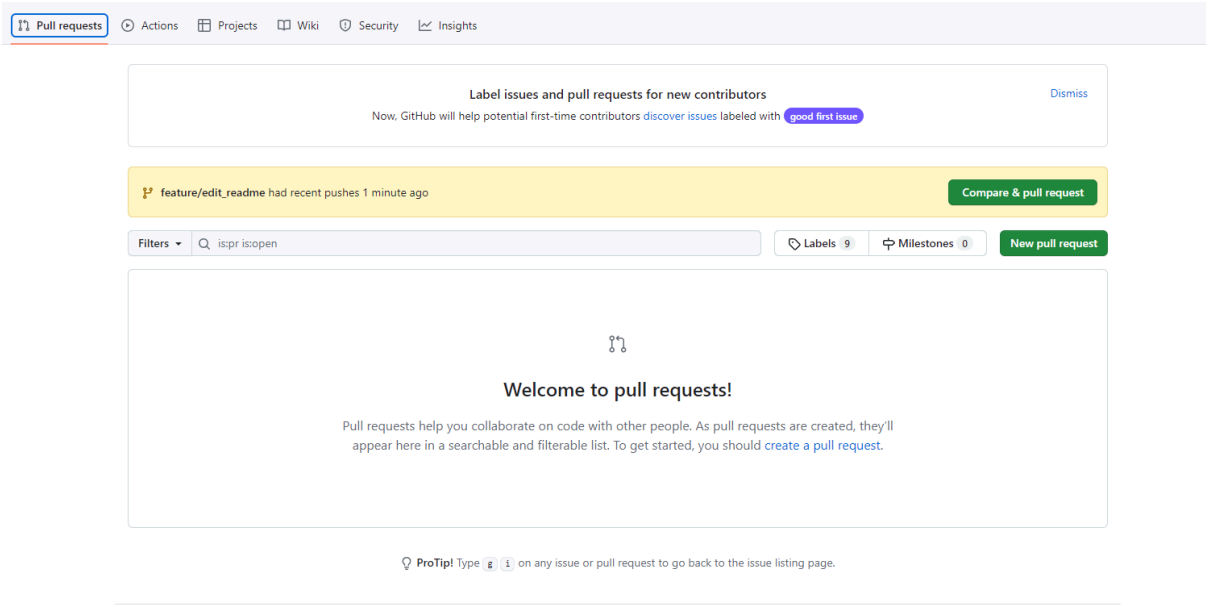
Соразработчики клонируют на свои компьютеры общий репозиторий. Каждый создаёт для своей задачи новую ветку. Один из разработчиков создал ветку с именем, например, **feature/edit_readme**, и занялся редактированием файла README.md. Файл стал таким:

```
# Командный проект
## Введение
Здесь будет общий обзор проекта
```

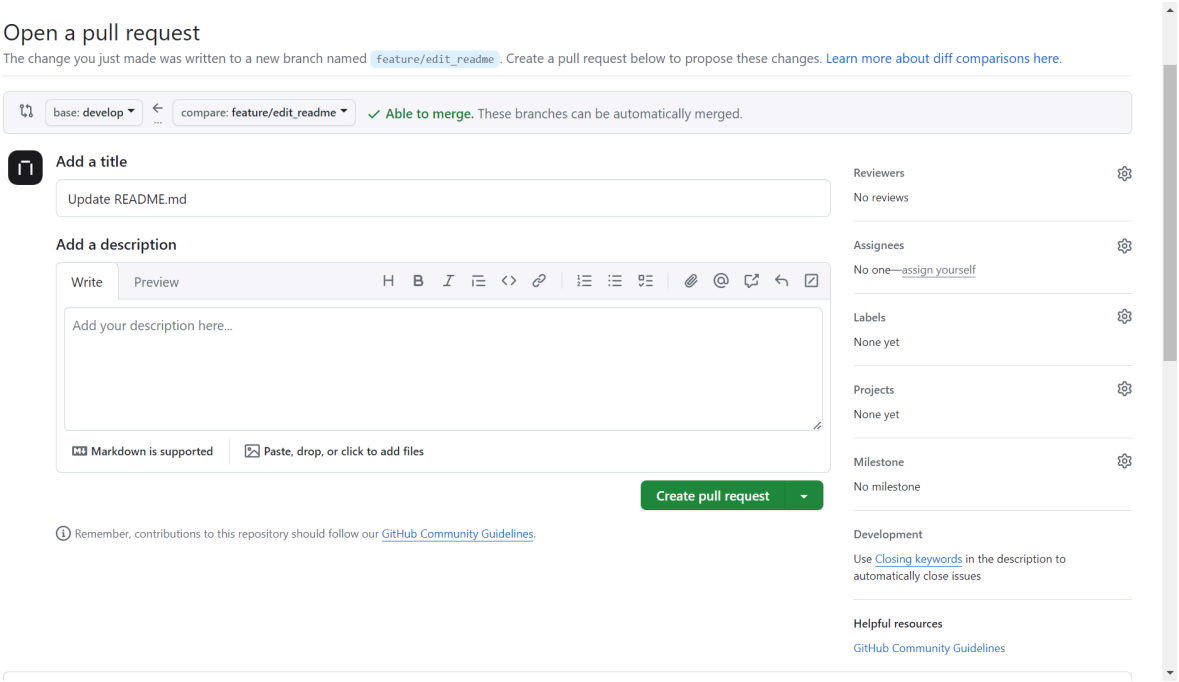
Разработчик сделал коммит и запушил изменения. Теперь в удалённом репозитории на GitHub тоже появилась ветка **feature/edit_readme.**

Чтобы изменения из ветки **eature/edit_readme** попали в общую ветку **develop**, разработчик делает Pull Request (запрос на изменение) в общем репозитории на GitHub.

Первый шаг: на странице репозитория в верхнем меню выбрать пункт *Pull Request*. Откроется такая страница:



Второй шаг: нажать на кнопку *Compare & pull request***.** Откроется новая страница:



На этой странице нужно выбирать ветку, куда должны быть влиты изменения, добавить описание, назначить наблюдателей, поставить метки — и после этого нажать на кнопку *Create pull request*.

Тимлид получит уведомление, откроет Pull Request, проверит его и, если в коде всё в порядке, примет его. Но может и отклонить.

При слиянии веток могут возникнуть конфликты: например, два разработчика в разных ветках изменили код в одной и той же строке, и в процессе слияния веток Git не может решить, какой код оставить в финальной версии.

В коде файлов с конфликтами Git даёт подсказки на тех строках, где в разных ветках текст отличается. Например:

```
<<<<<<< HEAD
# Командный проект (Current Change)
=======
# Групповой проект (Incoming change)
>>>>>>> 4006e63cdf36fc34b68287664733414d2203aa59
```

Такие конфликты решаются вручную: код файла, в котором возник конфликт, исправляют и вновь коммитят.

Практикум