0.2. Введение в Git

Начало работы, ключевые команды, практическое применение в разработке

Никита Ковалев

26 февраля 2022

Примерный план

1. Начало работы
2. Основные рабочие инструменты
3. Разветвление: мерджи и пулл-реквесты, конфликты

Основные понятия

- **Система контроля версий (VCS)** — набор инструментов, позволяющий создавать репозитории и осуществлять управление ими.
- **Репозиторий** — хранилище кодовой базы проекта с момента инициализации. Применяется в разработке для многих целей: начиная с сохранения истории изменений и заканчивая организацией процесса синхронной разработки.
- **Ветка** — элемент репозитория, отвечающий за одно направление последовательных изменений в кодовой базе. Совокупность всех веток составляет собой репозиторий.
- **Коммит** — элемент ветки, отвечающий за одно изменение кодовой базы на соответствующей ветке. У коммита есть такие свойства, как название и автор и уникальный тэг, а основной информацией является множество изменений, внесенных в код.
- **Тэг** — метка, ставящаяся обычно на состояния важных моментов в жизни проекта. Например, конечно, это могут быть очередные релизы.

Инициализация и клонирование

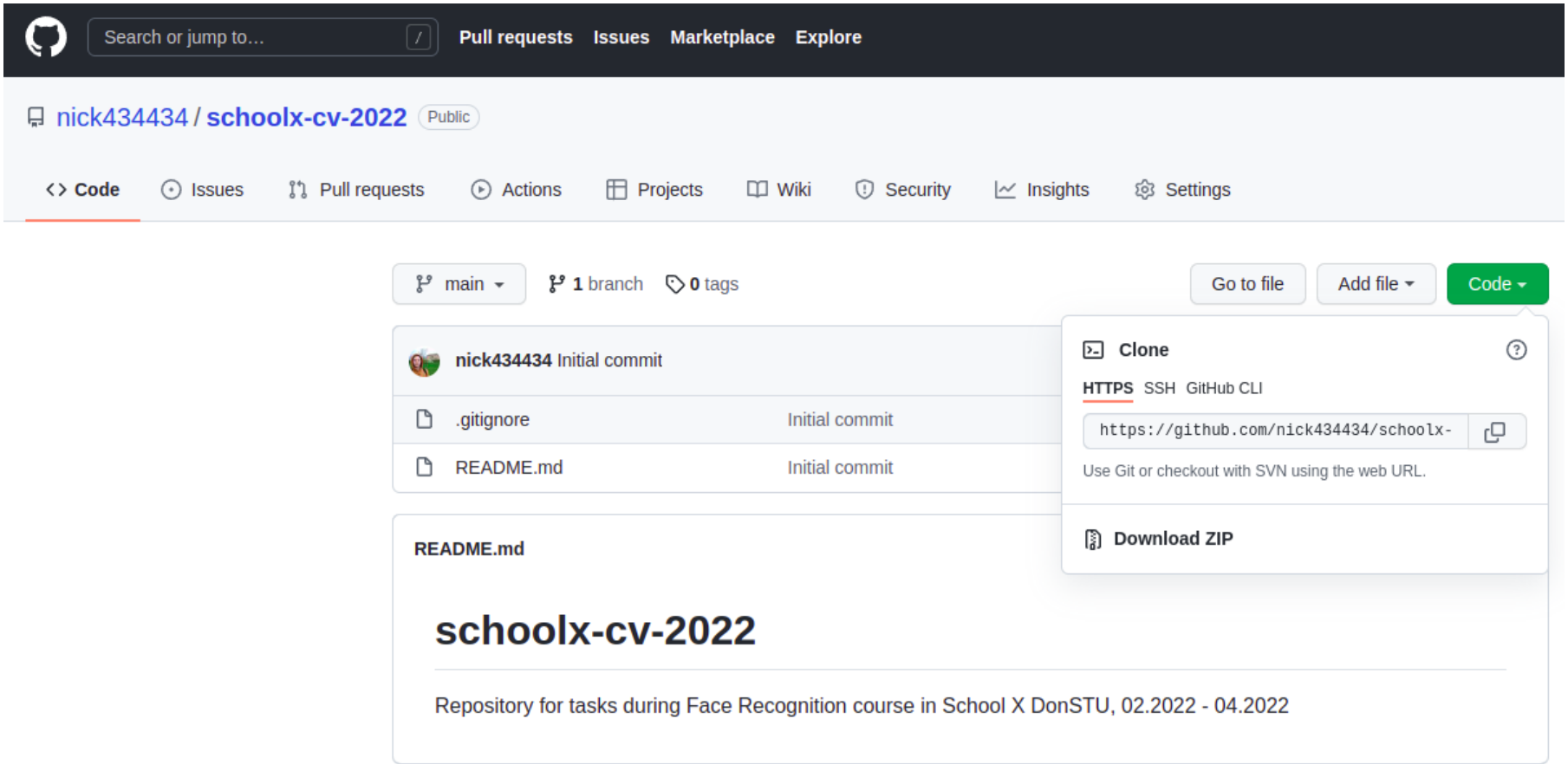
Для создания репозитория в github (gitlab, bitbucket, etc.) достаточно сделать это через интерфейс соответствующего веб-сайта, отметив пункт о создании репозитория с README. В таком случае репозиторий будет не пустой и его уже можно будет клонировать. Операция клонирования нужна для того, чтобы разработчик, начинающий работу над проектом, мог сделать это на основе имеющейся кодовой базы, не задействуя коллег для передачи ему кода приложения. Клонирование выполняется из командной строки следующим образом:

```
$ git clone https://github.com/nick434434/schoolx-cv-2022.git
```

С этого момента разработчик имеет доступ к коду со своего компьютера, и это называется локальным репозиторием. Для того, чтобы работа нескольких разработчиков, развивающих проект синхронно, была эффективна, каждому стоит регулярно публиковать внесенные изменения в публичный репозиторий и подгружать коммиты коллег в свой локальный.

Инициализация и клонирование: наглядно

Вот так выглядит репозиторий, инициализированный с созданием .gitignore и README. Скопировать ссылку для клонирования можно в контекстном меню 'скачивания кода', отмеченного зеленой кнопкой.



Работа с remote: https vs. ssh

Remote — удаленный репозиторий, характеризующийся веб-адресом, над которым ведется работа. Он необходим для обеспечения отправления своих изменений и подгрузки изменений других членов команды. Ремоутов в локальном репозитории допускается несколько.
Изменить веб-адрес remote можно с помощью команды `$ git remote set-url <remote-name> <remote-url>`. Добавить новый — `$ git remote add <remote-name> <remote-url>`. Сам веб-адрес бывает (как минимум) двух типов — обычный https и ssh-ссылка. Основное отличие с точки зрения пользования ими — ssh не требует ввода ваших логина и пароля при выполнении операций между локальным и удаленным репозиториями (о них речь пойдет далее).
При использовании в качестве remote ssh-ссылку разработчик должен загрузить ssh-ключ в свой аккаунт на сервисе, предоставляющем хостинг репозитория. Чтобы создать такой ключ у себя на устройстве, необходимо исполнить команду `$ ssh-keygen`. Для получения содержимого ключа затем достаточно вывести его в консоль командой `$ cat ~/.ssh/id_rsa.pub`.

Мантра


In case of fire
1. git commit
2. git push
3. exit building
Git > Your life

checkout, fetch, pull

Команда `git checkout <branch>` используется для загрузки в папку проекта локального состояния той или иной ветки. Помимо этого, с помощью неё можно создавать новые ветки — достаточно добавить аргумент `-b` перед <branch>. Также эту команду нельзя выполнять, если есть незакоммиченные изменения — чекаут перезаписывает текущее состояние локального репозитория, поэтому всегда нужно или сделать коммит, или удалить изменения (вернуться к последнему локально доступному коммиту).-

В свою очередь `git fetch` позволяет загружать в локальный репозиторий изменения из удаленного, не меняя при этом содержимое текущей активной ветки. При выполнении этой команда обновляются ветки, которые отвечают за удаленный репозиторий — а-ля `origin/<branch>`.

Комбинацией двух предыдущих команд является `git pull`. Он осуществляет загрузку изменений с сервера с помощью fetch, выполняя затем checkout в текущую ветку для локальной подгрузки изменений. Если текущее локальное состояние ветки отличается от состояния на сервере (ответвляющиеся коммиты, не добавленные ранее на сервер), будет произведен мердж текущего состояния с полученным удаленным.

push

Еще одно ключевое средство — команда `git push`. Она осуществляет публикацию новых локальных изменений в удаленный репозиторий. Использовать нужно, понимая все возможные последствия, особенно это касается опций типа `--force`. Она игнорирует некоторые проверки и 'насильно' публикует версию из локального репозитория поверх того, что уже было добавлено в удаленный.

stash, diff, log

Есть также много вспомогательных команд. Одна из таких — `git stash`. Она позволяет сохранять какие-то локальные изменения в формате стэка версий и затем доставать их оттуда при необходимости с помощью `git stash pop`. Помимо простого использования можно создавать именованые снимки изменений, по которым будет удобно обращаться в случае даже потери порядка изменений. Также ненужное изменение можно удалить со стэка — `git stash drop`.

Другая полезная штука — `git diff`. С её помощью без использования средств разработки (IDE и т.д.) можно детально и в удобных форматах просматривать детальные изменения в коде по сравнению с последним локальным коммитом или изучать статистику количества отредактированных строк и файлов (опция `--stat`).

Команда `git log` позволяет отследить историю локальных коммитов и информацию о них, в специальных ситуациях — найти метку нужного вам состояния проекта, и так далее.

merge

В процессе разработки практически всегда есть нужда выделять фичи в разные ветки, а также невозможно вестиписать код двум людям параллельно в одном месте. Для этих целей и осуществляется ветвление кодовой базы - новые ветки должны служить программистам механизмом облегчения работы по обособлению своих изменений. Но затем есть необходимость изменения с разных веток объединять обратно в одно общее состояние проекта — для этого есть, например, команда `git merge`.
С помощью этой команды пользователь может слить в текущую ветку изменения, которые были сделаны кем-то на другой ветке и закоммичены/подгружены в локальный репозиторий. В результате выполнения команды создается новый коммит, который помечается специальный merge-сообщением. Этот коммит содержит в себе изменения, необходимые для перехода от текущего состояния к новому, включающему в себя и текущие новые коммиты, и обновления со сливаемой ветки.
В случае, если автоматическая система сделать это не может, мердж объявляется конфликтным и в консоль выводится список файлов, которые необходимо слить вручную, а после этого создать мердж-коммит, завершив таким образом конфликт и процесс слияния.