



УРОК 8. ВВЕДЕНИЕ В GIT

ВИДЫ СИСТЕМ КОНТРОЛЯ ВЕРСИЙ	2
СОЗДАНИЕ GIT	5
ОСНОВНЫЕ ПРИНЦИПЫ И ФУНКЦИИ GIT	6
ОТЛИЧИЕ GIT ОТ GITHUB	7
ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ	8
Создание учетной записи Github и добавление ssh ключа	8
Создание нового репозитория в github	13
РАБОТА С GIT	16
GIT INIT	17
ДИРЕКТОРИЯ .GIT	18
ОСНОВНЫЕ СТАДИИ РАБОТЫ С GIT	19
ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ	21



ВИДЫ СИСТЕМ КОНТРОЛЯ ВЕРСИЙ

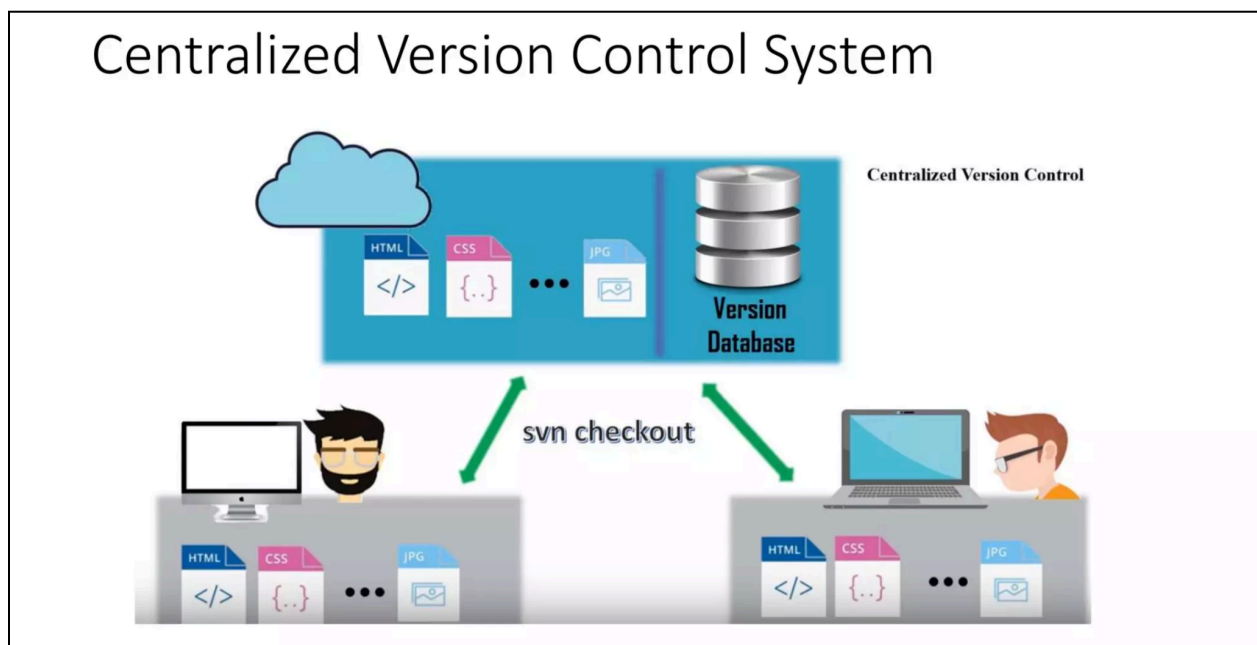


Системы контроля версий — это программные инструменты, помогающие командам разработчиков управлять изменениями в исходном коде с течением времени. В свете усложнения сред разработки они помогают командам разработчиков работать быстрее и эффективнее.

Как возникла необходимость в Системе контроля версий кода:

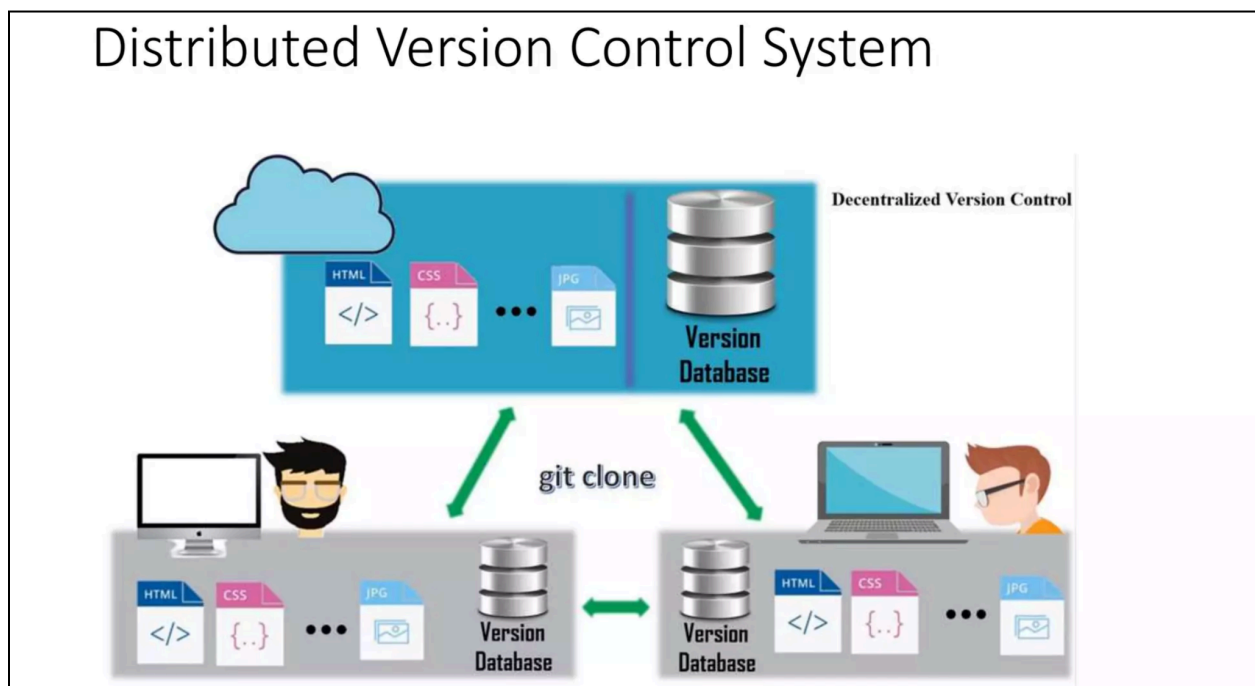
- Проблемы совместной работы над кодом: Исходный код программ часто разрабатывается несколькими людьми, и требуется эффективное управление версиями и совместной разработкой.
- Старые методы управления версиями: Перед появлением Git использовались централизованные системы управления версиями, такие как Subversion (SVN) и CVS. Они имели свои ограничения в масштабируемости и гибкости.

1. Централизованные системы контроля версий (Centralized Version Control Systems, CVCS):



- Примеры: CVS (Concurrent Versions System), SVN (Subversion), Perforce.
- В CVCS существует единый центральный сервер, который содержит все версии файлов и историю изменений.
- Разработчики работают с локальными копиями файлов, которые синхронизируются с сервером при необходимости.
- Ограничения CVCS включают недостаток распределенности, уязвимость к отказам сервера и медленную скорость работы при выполнении операций, таких как слияние и обновление.

2. Распределенные системы контроля версий (Distributed Version Control Systems, DVCS):



- Примеры: Git, Mercurial, Bazaar.
- В DVCS каждый разработчик имеет полную копию репозитория, включая всю историю изменений.
- Репозиторий может быть скопирован (клонирован) на локальный компьютер, где разработчик может работать автономно, без доступа к центральному серверу.



- Это обеспечивает большую гибкость и устойчивость к отказам сервера, а также улучшенную производительность при выполнении операций.

3. Локальные системы контроля версий (Local Version Control Systems):

- Пример: RCS (Revision Control System).
- Локальные СКВ работают с отдельными файлами на одном компьютере.
- Они обеспечивают базовый уровень контроля версий, но не подходят для совместной работы над проектами или для распределенных команд разработчиков.



СОЗДАНИЕ GIT

Начало:

- История Git началась в 2005 году, когда Линус Торвальдс, создатель ядра Linux, столкнулся с необходимостью эффективного управления версиями кода для своего проекта.
- В это время он использовал систему контроля версий BitKeeper, но возникли разногласия с разработчиками BitKeeper относительно доступа к коду и других вопросов.

Начало разработки Git:

- Линус решил создать свою собственную систему контроля версий, которая была бы более гибкой и подходящей для нужд разработки Linux.
- Он начал работу над Git, придерживаясь нескольких ключевых принципов, таких как скорость, простота и распределенная природа системы.

Рост популярности:

- С течением времени Git стал широко принятым стандартом в индустрии разработки программного обеспечения.
- Git быстро получил популярность благодаря своей эффективности, гибкости и широкому набору функций, которые делали его подходящим для широкого спектра проектов.



ОСНОВНЫЕ ПРИНЦИПЫ И ФУНКЦИИ GIT

Основные принципы Git:

- Распределенная система контроля версий: Каждый разработчик имеет полную копию репозитория, что позволяет работать автономно и уменьшает зависимость от центрального сервера.
- Система снимков (Snapshots): Git сохраняет снимки проекта в разные моменты времени, а не отслеживает изменения файлов, как это делают другие системы контроля версий.
- Эффективность и скорость: Git оптимизирован для работы с большими проектами и обладает быстрыми операциями за счет использования локального хранилища.

Основные функции Git:

- Управление версиями: Git позволяет отслеживать изменения в коде и возвращаться к предыдущим версиям проекта.
- Ветвление и слияние (Branching and Merging): Разработчики могут создавать отдельные ветки для разработки новых функций, а затем объединять их с основной веткой (обычно "master" или "main").
- Работа с удаленными репозиториями: Git поддерживает синхронизацию изменений между локальным и удаленным репозиториями через протоколы, такие как HTTP, SSH, и другие.
- Контроль доступа и авторства: Git предоставляет средства для управления доступом к репозиториям и отслеживания авторства изменений.



ОТЛИЧИЕ GIT ОТ GITHUB



Git - это сама система контроля версий, которая работает локально на компьютере разработчика.



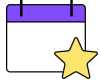
GitHub - это веб-платформа, предоставляющая хостинг для Git-репозиторий и дополнительные функции для совместной разработки и управления проектами.

Git:

- **Система контроля версий (СКВ):** Git является распределенной системой контроля версий, разработанной Линусом Торвальдсом. Он предоставляет инструменты для отслеживания изменений в исходном коде проекта и управления версиями файлов.
- **Локальное хранилище:** Git работает локально на компьютере разработчика. Он позволяет создавать коммиты, ветви, слияния и другие операции непосредственно на компьютере, без необходимости подключения к сети или центральному серверу.
- **Командная строка и интерфейсы:** Git предоставляет командную строку и графические интерфейсы для выполнения операций управления версиями.

GitHub:

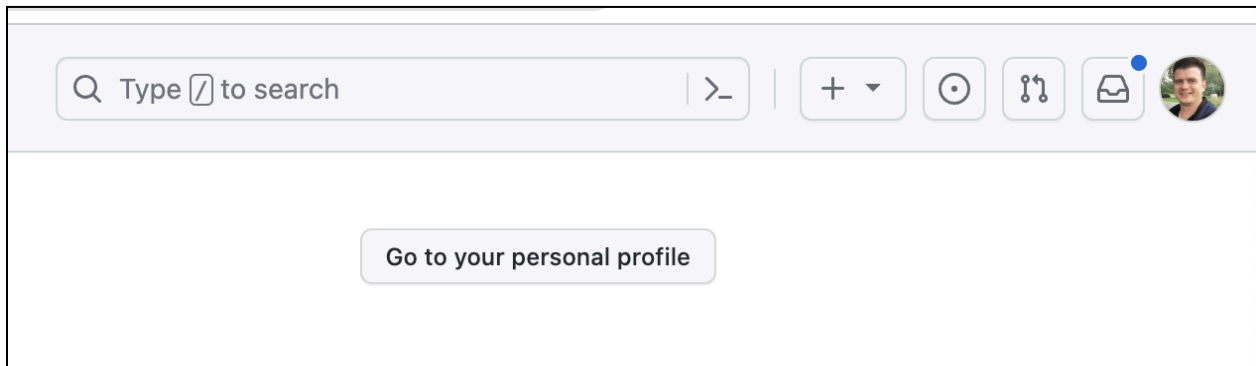
- **Веб-платформа для хостинга Git-репозиторий:** GitHub - это веб-платформа, от компании Microsoft, которая предоставляет хостинг для Git-репозиторий. Она позволяет разработчикам хранить, управлять и совместно работать над проектами с использованием Git.
- **Социальные и совместные возможности:** GitHub предлагает различные социальные функции, такие как возможность следить за проектами, следить за активностью других пользователей, отмечать интересные репозитории, существует система комментариев и рейтингов.
- **Интеграция с инструментами разработки:** GitHub интегрируется с различными инструментами разработки, такими как среды разработки (IDE).



ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Добавим свой ssh ключ.

Для этого нажимаем на верхний правый угол, где есть иконка пользователя:



Ключ скопировать из `~/.ssh/id_rsa.pub` по умолчанию или из той локации, где находится ваш публичный SSH ключ.

Просмотр публичного ключа (если ошибка, то надо сгенерировать)

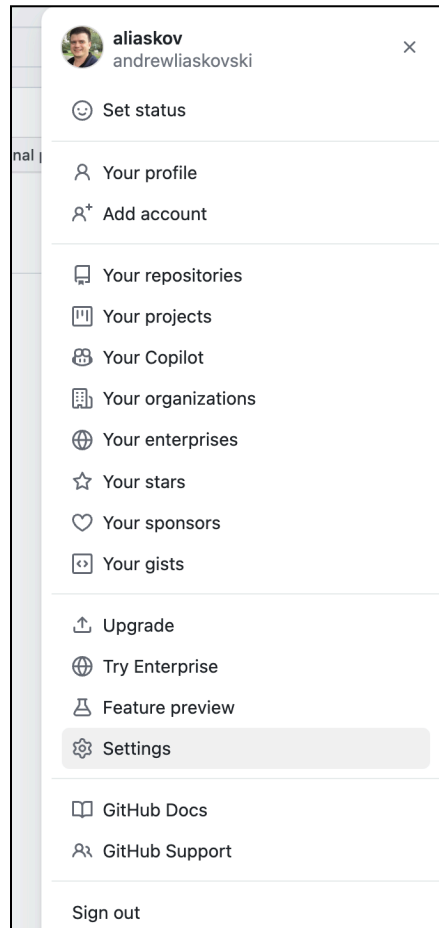
- `cat ~/.ssh/id_ed25519.pub`

- `cat ~/.ssh/id_rsa.pub`

- `ssh -T git@github.com`


Проверить, что публичный ключ выгружен на GitHub

И выбираем settings:



Затем необходимо нажать на SSH and GPG keys в меню слева.



**andrewliaskovski (aliaskov)**
Your personal account [Switch settings context](#) ▼

Public profile

Account

Appearance

Accessibility

Notifications

Access

Billing and plans

Emails

Password and authentication

Sessions

SSH and GPG keys

Organizations

SSH keys

This is a list of SSH

Authentication

id_rs
SHA25
Adde
Last u

Check out our guide

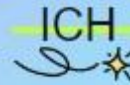
GPG keys

Теперь нажимаем на New SSH key:

SSH keys

New SSH key

В открывшемся окне добавим Title - это название для ключа, который вы добавляете. Это может быть удобным в случае, если вы будете работать на разных компьютерах и это позволит управлять этими ключами, сгенерированными на разных устройствах.



Add new SSH Key

Title

ssh_key_laptop

Key type

Authentication Key ▾

Key

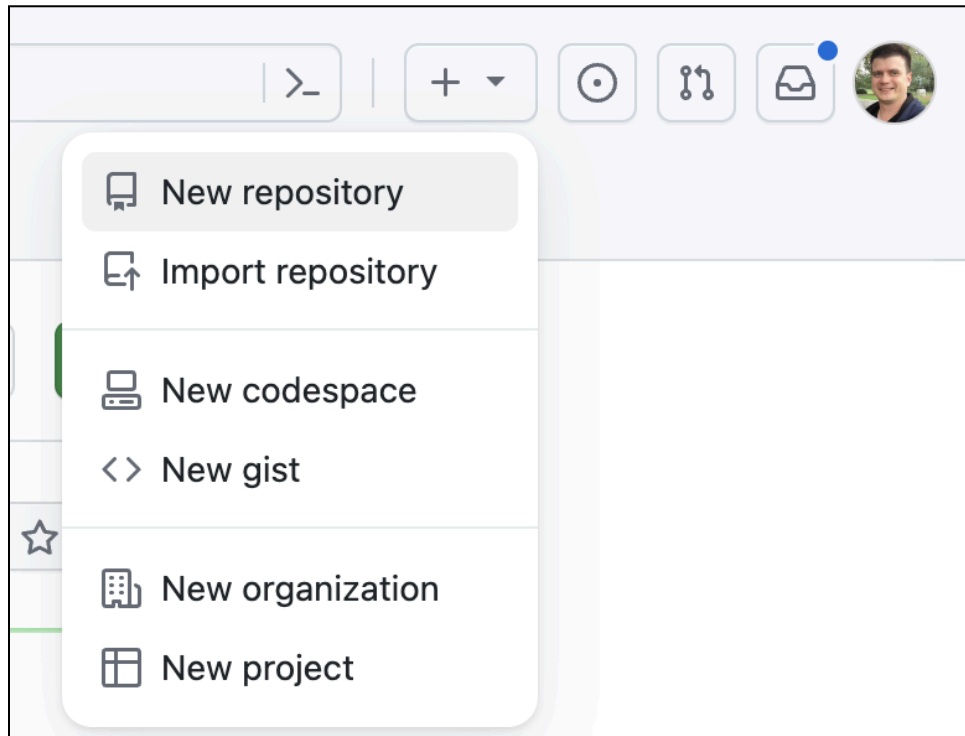
```
ssh-rsa
AAAAB3NzaC1yc253535355355VIFhMikro7bXz15yAUSWsemcD6B6oPUhlZzYXPjteYeSBCB8LzJhJXEjtG4nxD4w8xIPbudCdc2T/2L43435
33Rq4gh2BEM2SRuCjYjBC7kpBJLrGjJIMJhlbcFJmUR2TJ15kle3FQ353535SPCBkFC+OpOzUp0p9wZVc8ZPZD1H41KVRBcDkS9eRPevmO
sAriHVVWcEb1m+XDTJ0UbnswHijEahoC1yZpgrergerglUJujydCEgxJ8Qdtbq2uQegeg345352p0EPDlpEtqb5w2QLUlhg5Y/l+b8mKfYdF/AA2p
U+N+5fhC/29df7xxQuGmKw3bwqD1S+b7mu1VY2PqEm1l/MvqvdbOCsisJXs2n/LjoINALKRHjrl00eWOLK626uNxqUZY4b3smmJgZ+Ckcx3bz
moYSc5f/6KgfaLbSWf3Nns= andrei.liaskovski@w0eqc
```

Add SSH key

Нажимаем на Add SSH key - Готово!

Создание нового репозитория в github

Для этого нажмем на + в верхнем правом углу и выберем New repository.



Введем название для нового репозитория и оставим все остальные поля по умолчанию:



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Required fields are marked with an asterisk (*).

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner *

 aliaskov ▾

Repository name *

git_intro

✔ git_intro is available.

Great repository names are short and memorable. Need inspiration? How about [bug-free-journey](#) ?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)



You are creating a public repository in your personal account.

Create repository

И нажимаем на Create repository.



РАБОТА С GIT

Для пользователей windows нам будет удобен git bash , для пользователей macOS - работаем в терминале, убедиться в наличии git можно просто набрав команду git. Если на компьютере не установлен набор программ разработчиков - будет установлен xcode , в комплекте которого есть git

Команда git config используется для управления настройками Git. Она позволяет задавать и получать различные параметры конфигурации Git, включая настройки пользователя, настройки репозитория, а также глобальные настройки.

```
git config --global user.name "Ваше имя"
git config --global user.email "ваш@адрес.почты"
```



GIT INIT

git init используется для инициализации нового репозитория Git в текущем каталоге или в указанной директории. После выполнения этой команды происходит несколько важных действий:

- Создание скрытой директории .git: Git создает скрытую директорию .git в текущем каталоге (или указанной директории), которая содержит все данные, необходимые для функционирования репозитория.
- Инициализация пустого репозитория: Git создает пустой репозиторий без каких-либо файлов или истории коммитов. В этом начальном состоянии репозиторий готов принимать файлы и отслеживать их изменения.
- Настройка конфигурации репозитория: Git создает файл конфигурации .git/config, который содержит настройки репозитория, такие как информация о пользователе и настройки поведения Git.
- Создание пустого индекса (staging area): Git создает пустой индекс, который является промежуточным хранилищем для изменений, подготовленных к коммиту. В начальном состоянии индекс пустой.



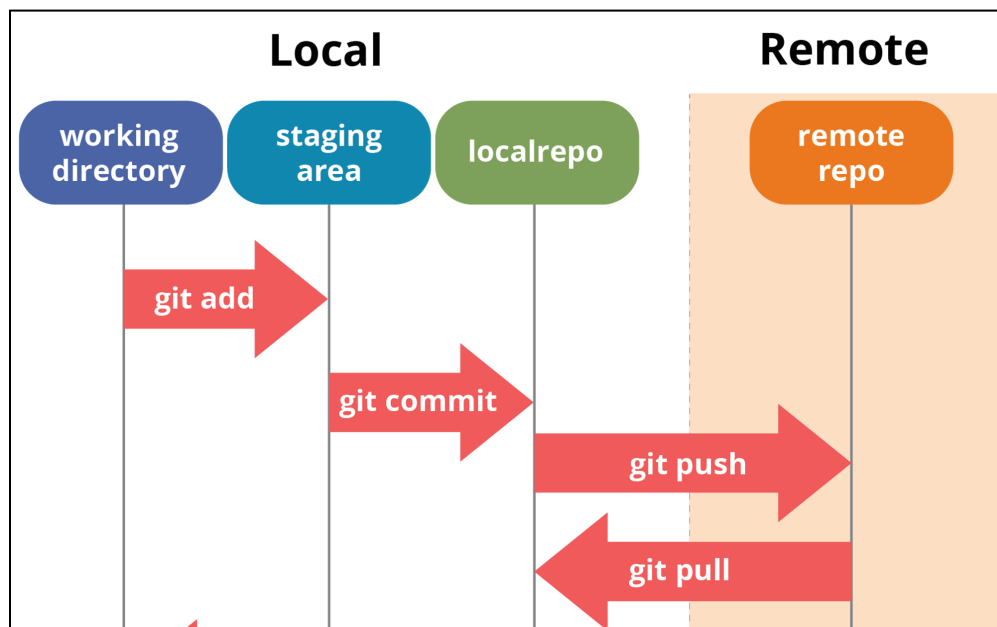
ДИРЕКТОРИЯ .GIT

Директория .git содержит все данные, необходимые Git для управления репозиторием.

- Конфигурационные файлы: Эти файлы содержат настройки репозитория, такие как информация о пользователе и параметры поведения Git.
- Хранилище объектов (objects/): Это директория, где хранятся все файлы и история изменений. Каждый файл и коммит представлен объектом, который сохраняется здесь.
- Индекс (index): Этот файл представляет собой список изменений, которые готовы к коммиту. Он является промежуточным хранилищем перед сохранением изменений в истории.
- Ссылки (refs/): Здесь хранятся ссылки на различные метки, ветки и другие объекты в репозитории. Это позволяет Git отслеживать и перемещаться по истории изменений.
- Хуки (hooks/): Эта директория содержит скрипты, которые можно запускать на определенных этапах работы с репозиторием, таких как перед коммитом или при получении данных из удаленного репозитория.
- Конфигурация удаленных репозиториях (config): Этот файл содержит информацию о удаленных репозиториях, таких как URL-адреса и настройки подключения.
- Другие вспомогательные файлы и директории: В директории .git могут быть и другие файлы и поддиректории, включая файлы журналов, информацию о подмодулях и т. д.



ОСНОВНЫЕ СТАДИИ РАБОТЫ С GIT



1. git clone:

- Скачивает репозиторий и создает копию на локальном компьютере, предварительно создав директорию с названием репозитория.
- Использование: `git clone <repo url>`
- Это копирование репозитория из github.

2. git add:

- Добавляет измененные файлы в индекс (staging area), подготавливая их к коммиту.
- Использование: `git add <file>` для добавления конкретного файла, или `git add .` для добавления всех измененных файлов.
- Это как подготовка к тому, чтобы сказать Git: "Эти файлы нужно сохранить".

3. git commit:

- Создает коммит с текущими изменениями в индексе.



- Использование: `git commit -m "Комментарий к коммиту"` для создания коммита с сообщением.
- Это как сохранение текущего состояния вашего проекта.

4. git push:

- Отправляет коммиты из локального репозитория в удаленный репозиторий.
- Использование: `git push <remote> <branch>` для отправки коммитов на указанную ветку удаленного репозитория.
- Это как публикация ваших изменений онлайн.

5. git pull:

- Получает изменения из удаленного репозитория и объединяет их с локальными изменениями.
- Использование: `git pull <remote> <branch>` для получения изменений из указанной ветки удаленного репозитория и их объединения.
- Это как обновление вашей работы с изменениями, сделанными другими.



ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Создать репозиторий в github, добавить, закоммитить и пушить в него из нового локального репозитория.

Решение:

```
git init
echo "# test" >> README.md
```

```
git status
git add <files> или git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:USERNAME/REPO_NAME.git
git push -u origin main
git log
```

git init:

- Создает новый локальный репозиторий Git в текущей директории, где вы работаете.

echo "# test" >> README.md:

- Создает новый файл README.md и записывает в него строку "# test".

git status:

- Показывает текущее состояние репозитория, включая измененные файлы, файлы в индексе и т. д.

git add <files> или git add README.md:

- Добавляет файлы (в данном случае README.md) в индекс, подготавливая их к коммиту.

git commit -m "first commit":

- Создает коммит с добавленными изменениями и сохраняет их в локальном репозитории, сопровождаемый сообщением "first commit".

git branch -M main:

- Переименовывает основную ветку (по умолчанию master) в main.

git remote add origin git@github.com:USERNAME/REPO_NAME.git:



- Добавляет удаленный репозиторий с именем origin, указывающий на ваш репозиторий на GitHub.

git push -u origin main:

- Отправляет ваши коммиты из локального репозитория на удаленный репозиторий (GitHub) в ветку main. Опция -u устанавливает отслеживание ветки origin/main, чтобы в будущем git push без указания ветки отправлял изменения в эту ветку.

git log:

- Показывает список всех коммитов в вашем репозитории, начиная с самого последнего, с информацией о каждом коммите (автор, дата, сообщение и т. д.).