**giGit** — распределённая система хранения версий для фиксации изменений в файлах, которая представляет собой набор консольных утилит. Она разработана для координации работы между программистами, но в целом может использоваться и для других целей, которые мы рассмотрим чуть позже.

**ДРУГИЕ СООБЩЕНИЯ, КОТОРЫЕ ВЫ МОЖЕТЕ ВСТРЕТИТЬ ПРИ ПРОВЕРКЕ СТАТУСА:**

* + **On branch master**. Сообщение означает, что в настоящий момент активна ветка *master*. Как правило, это основная ветка в репозитории.
  + **Your branch is up to date with origin/master**. Сообщение означает, что файлы в ветке *master* синхронизированы с веткой *master* на сервере. *Origin* — это и есть удалённый сервер.
  + **Working tree clean** дословно означает, что рабочий каталог не содержит изменённых файлов. Система контроля версий *Git* использует так называемую архитектуру трёх деревьев. *Working tree* связана с рабочей директорией *Git*, в которой содержатся рабочие версии файлов.
  + **Nothing to commit** значит, что все изменения были зафиксированы и нет новых изменений в файлах для фиксации изменений.
* **ВНИМАНИЕ:** В новых версиях *Git* ветка *master* может быть по умолчанию переименована в *main.* Посмотреть какая ветка у вас можно с помощью команды git branch. Если у вас ветка *main,* то работать с ней следует аналогично ветке *master.*

**коммит**

Другими словами, **коммит**— это состояние репозитория в определённый момент времени. Именно к коммиту мы можем возвращаться, чтобы посмотреть состояние объектов в какое-то время.

## ****Создание коммита****

Давайте создадим первый коммит.

Для этого нам надо сначала добавить все те файлы, изменения в которых или создание/удаление которых мы хотим зафиксировать.

Чтобы добавить все файлы, которые находятся в директории, выполните следующую команду:

**git add -A**

В будущем, чтобы добавлять отдельные файлы, используйте ту же команду add, но с указанием имени файла:

**git add this.js**

Если сейчас проверить состояние репозитория при помощи команды git status, то уже можно заметить изменения.

Но пока мы только собрали все те файлы, изменения в которых нам важны. Теперь их надо обвязать общей верёвочкой и единым ярлыком, то есть собрать их в коммит. Для этого необходимо выполнить следующую команду:

git commit -m "Init Skillfactory"

## ****Содержимое директории .git****

Рассмотрим, что содержится в папках и файлах:

* + В **файле config** находятся настройки данного репозитория. Его содержимое представлено в текстовом формате.
  + **Файл HEAD** указывает на текущую ветку.
  + В **файле index** хранится содержимое индекса.
  + В **директории objects** находится, собственно, база данных объектов Git. Если открыть каталог objects, то в нём будут находиться каталоги, имена которых представлены двумя шестнадцатеричными числами, внутри которых будут файлы, имена которых представлены 38 шестнадцатеричными числами. Вместе имя каталога и файл образуют 40-разрядный хэш, взятый от имени файла и его содержимого.
  + В **директории refs** находятся ссылки на объекты коммитов в этой базе (ветки).
  + **Директория logs** хранит логи коммитов.
  + В **директории info** расположен файл с глобальными настройкам игнорирования файлов. Он позволяет исключить файлы, которые вы не хотите помещать в .gitignore. Позднее мы остановимся на назначении файла .gitignore.
  + В **директории hooks** располагаются клиентские и серверные триггеры. Желающие прочитать про них могут обратиться к руководству по Git.

## ****Сохранение изменений****

По сути, коммит в Git можно понимать как аналог сохранения в приложениях, которыми вы пользуетесь каждый день. При этом традиционное «сохранить» в терминах файловой системы — перезапись существующего файла или создание нового, в то время как **коммит в Git — операция над коллекцией файлов и директорий.**

git add и git commit — две самые часто используемые команды Git.

## ****git add****

Команда git add добавляет изменения из рабочей директории в область подготовленных файлов. Она сообщает Git, что вы хотите включить изменения в следующий коммит.

Однако git add не влияет на состояние репозитория: изменения, на самом деле, не внесены в историю.

Чтобы узнать состояние рабочей директории и области подготовленных файлов, понадобится команда git status.

## ****git commit****

Команда git commit делает «снимок» изменений проекта. Добавленные коммиты можно рассматривать как «безопасные» версии проекта: Git никогда не изменит их, если только вы специальным образом не потребуете этого.

Если выполнить команду git commit без каких-либо флагов, то запустится текстовый редактор, запрашивающий у вас сообщение для коммита. Чтобы

сразу указать сообщение для коммита используется флаг -m (message)

|  |  |
| --- | --- |
| **git commit -a** | При выполнении команды с флагом -a будет сделан снимок всех изменений в рабочем каталоге. Это включает в себя только изменения отслеживаемых файлов (то есть тех, которые были добавлены с помощью git add ранее). |
| **git commit -am "commit message"** | Сочетание флагов -a и -m создаёт коммит всех изменений с сообщением для коммита. |
| **git commit --amend** | Выбранные изменения будут добавлены в последний коммит. Команда также вызовет открытие системного текстового редактора для внесения изменений в сообщение коммита. Эту команду часто используют, если была допущена ошибка. |

## ****Целостность данных****

Невозможно изменить содержимое файла так, чтобы Git не узнал об этом. Для того чтобы определять целостность данных при каждом их изменении, используется **Secure Hash Algorithm 1** (SHA-1) — алгоритм криптографического хеширования.

SHA-1 хеш выглядит примерно так:

**c46b84886abe275675c78abfe276e699451f26f0**

«Под капотом» Git работает именно с такого рода хешами, а не с именами файлов.

## 5. Ветвление и коммиты слияния

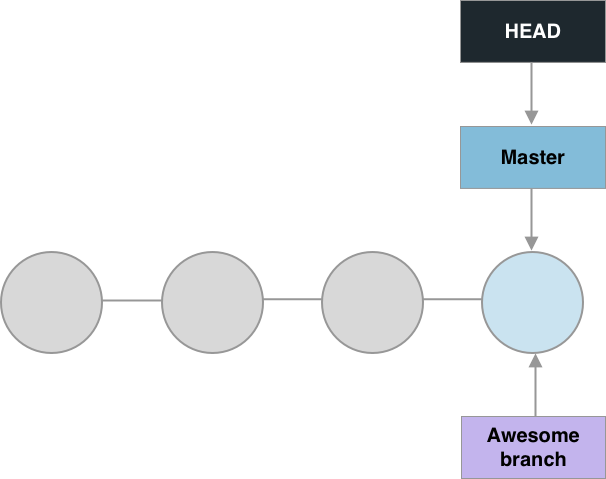
В этом модуле мы уже затронули такое понятие, как **ветка.** В каждом репозитории есть как минимум одна ветка. По умолчанию Git создаёт ветку master, где идёт основная разработка.

Давайте создадим новую ветку. Для этого выполним в Git-репозитории следующую команду:

git branch awesome-branch

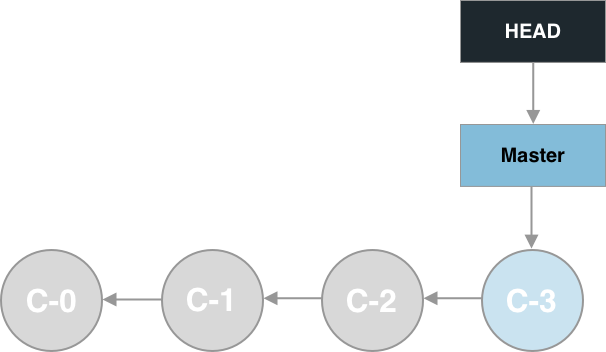
Чтобы перейти на существующую ветку, необходимо выполнить следующую команду:

git checkout awesome-branch



Команда git checkout позволяет вам перемещаться между ветками, которые уже были созданы. Когда вы переключаетесь на ветку, то файлы в вашей рабочей директории меняются в соответствии с той версией, на которую указывает выбранная ветка. Все новые коммиты будут теперь добавляться в текущую ветку, пока вы снова не переключитесь на другую.

## ****Ветвление и коммиты слияния****



тобы приступить к работе над изменениями для Чёрной пятницы, создадим специальную ветку:

git branch black-friday

\*Если вы ошиблись в названии, например, допустили опечатку, вы можете изменить название ветки с помощью команды: git branch -m старое-имя-ветки новое-имя-ветки.

И сразу переключимся на эту ветку:

git checkout black-friday

После выполнения этих команд состояние репозитория изменится.

После этого мы можем менять файлы проекта и не бояться, что наш сайт как-то пострадает, ведь мы делаем это в изолированной ветке.

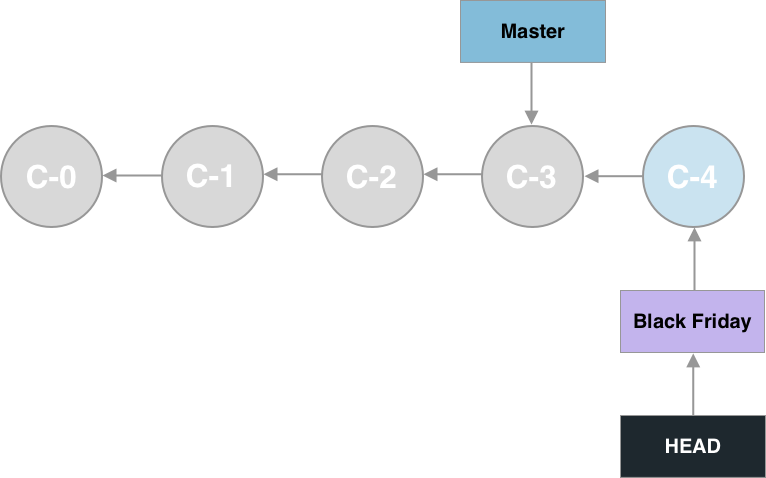
Итак, допустим, мы изменили файл index.html. Теперь мы хотим зафиксировать эти изменения.

Файл index.html мы привели как пример. Вы же можете загрузить любой файл.

git add index.html

git commit -m ''basic changes for Black Friday''

Что изменилось на этот раз?



Чтобы внести изменения из одной ветки в другую, нужно выполнить команду **git merge**. По-русски эти команды часто называют «вмержить», «смержить» или «сделать мерж», хотя, конечно, это всё производные от английских терминов. Правильно на русский язык merge можно перевести как «слияние», поэтому говорят также «слияние веток».

**Слияние**— это способ соединить обратно раздвоенную историю. Команда git merge позволяет взять независимые части разработки и интегрировать их в одну ветку.

Если бы мы хотели влить новые изменения в текущую ветку (Black Friday), то надо было бы выполнить команду: git merge master, что означало бы, что в текущую ветку мы хотим влить ветку с именем master. Это очень полезно, когда программист долго разрабатывает какой-то новый функционал, а за это время в основной ветке происходят какие-то изменения, с которыми ему необходимо считаться.

В нашем же случае нам надо наоборот в ветку master влить black-friday. Как это можно сделать?

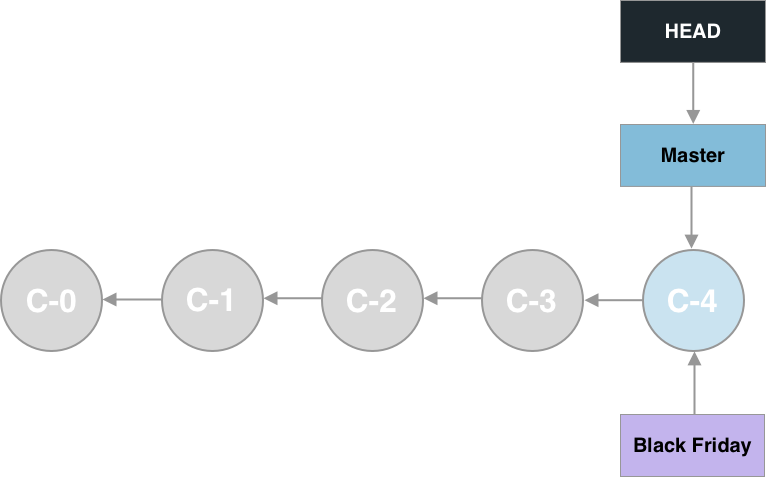
Для этого нам надо сначала переключиться на ту ветку, в которую мы хотим вливать, а затем уже «мержить».

git checkout master

git merge black-friday

Ура! Изменения для Чёрной пятницы теперь доступны всем!

Посмотрим, как выглядит репозиторий теперь:



Значит, её нужно удалять, потому что незачем хранить лишнее. Чтобы удалить ветку, достаточно выполнить команду:

git branch -d black-friday

Создадим новую ветку в Git с помощью известной нам команды git branch, после которой следует имя создаваемой ветки. Назовем её следующим образом:

git branch testbranch00

\*Вы можете использовать своё имя ветки. Далее по тексту будет использоваться имя ветки testbranch00.

Выполните следующие команды:

* 1. Для проверки состояния выполните команды git status и git log. Последняя показывает историю изменений: перечисляет коммиты, сделанные в репозитории, в обратном хронологическом порядке. Последние коммиты находятся вверху.
  2. Команду git branch --list. Она выведет список веток, которые существуют в вашем локальном репозитории. Текущая ветка master будет выделена зелёным цветом.
  3. Команду git branch --list -v . Вы увидите информацию о последних коммитах каждой из веток. Так как коммитов в ветку testbranch00 ещё не было, то она указывает на коммит в master, от которого была создана.
  4. Команда git checkout переключит нас на созданную ветку. Выполняя команду git checkout, укажите после неё имя ветки, на которую вы хотите переключиться. Например, git checkout testbranch00. Вы получите сообщение о переключении на ветку testbranch00.
  5. Выполните команду git status. Вы увидите сообщение о том, что вы находитесь на ветке testbranch00.

## 6. Конфликты

Рассмотрим гипотетический проект с двумя ветками: master и feature. В обеих ветках был изменён файл file.txt. Допустим, мы находимся на ветке feature и хотим влить изменения из master. Выполняем уже известную команду:

git merge master

**Merge** **conflict** **in** **file**.txt

**Automatic** **merge** **failed**

**Fix** **conflicts** **and** **then** **commit** **the** **result**

Конфликт возник между изменениями, которые находятся между “<<<<<<< HEAD”  и “=======”, и изменениями между “=======” и “>>>>>>> master”. В данный момент файл содержит обе версии.

Если проверить сейчас git status, то вы увидите примерно следующее сообщение:

На ветке feature

У вас есть не слитые пути.

  (разрешите конфликты, затем запустите «git commit»)

  (используйте «git merge --abort», чтобы остановить операцию слияния)

Не слитые пути:

  (используйте «git add <файл>…», чтобы пометить разрешение конфликта)

оба изменены:   file.txt

нет изменений добавленных для коммита

(используйте «git add» и/или «git commit -a»)

Сейчас слияние находится в подвешенном состоянии. Его надо либо продолжить, либо прервать.

Если выполнить **git merge --abort**, как указано в инструкции, то всё вернётся в такое состояние, каким было на ветке feature до попытки слияния.

Чтобы продолжить слияние, исправьте в текстовом редакторе файл так, чтобы получить нужное состояние. В зависимости от ситуации вам понадобится использовать версию из ветки feature или из ветки master, или объединить изменения. В любом случае в финальной версии файла не должны содержаться специальные вставки **git merge** “<<<<<<< HEAD”, “=======” и “>>>>>>> master”.

После того как вы исправили **file.txt**, его стоит добавить как перед обычным коммитом с помощью **git add**.

После этого запустите **git commit**, чтобы завершить слияние.

## Установка Git

GitHub предоставляет оконное приложение с графическим интерфейсом для выполнения основных операций с репозиторием, и консольную версию Git с автоматическими обновлениями для расширенных сценариев работы.

### GitHub Desktop

[desktop.github.com](https://desktop.github.com/)

Дистрибутивы Git для систем Linux и POSIX доступны на официальном сайте Git SCM.

### Git для всех платформ

[git-scm.com](https://git-scm.com/)

## Первоначальная настройка

Настройка информации о пользователе для всех локальных репозиториев

$ git config --global user.name "[имя]"

Устанавливает имя, которое будет отображаться в поле автора у выполняемых вами коммитов

$ git config --global user.email "[адрес электронной почты]"

Устанавливает адрес электронной почты, который будет отображаться в информации о выполняемых вами коммитах

## Создание репозитория

Создание нового репозитория или получение его по существующему URL-адресу

$ git init [название проекта]

Создаёт новый локальный репозиторий с заданным именем

$ git clone [url-адрес]

Скачивает репозиторий вместе со всей его историей изменений

## Внесение изменений

Просмотр изменений и создание коммитов (фиксация изменений)

$ git status

Перечисляет все новые или изменённые файлы, которые нуждаются в фиксации

$ git diff

Показывает различия по внесённым изменениям в ещё не проиндексированных файлах

$ git add [файл]

Индексирует указанный файл для последующего коммита

$ git diff --staged

Показывает различия между проиндексированной и последней зафиксированной версиями файлов

$ git reset [файл]

Отменяет индексацию указанного файла, при этом сохраняет его содержимое

$ git commit -m "[сообщение с описанием]"

Фиксирует проиндексированные изменения и сохраняет их в историю версий

## Коллективная работа

Именованные серии коммитов и соединение результатов работы

$ git branch

Список именованных веток коммитов с указанием выбранной ветки

$ git branch [имя ветки]

Создаёт новую ветку

$ git switch -c [имя ветки]

Переключается на выбранную ветку и обновляет рабочую директорию до её состояния

$ git merge [имя ветки]

Вносит изменения указанной ветки в текущую ветку

$ git branch -d [имя ветки]

Удаляет выбранную ветку

## Операции с файлами

Перемещение и удаление версий файлов репозитория

$ git rm [файл]

Удаляет конкретный файл из рабочей директории и индексирует его удаление

$ git rm --cached [файл]

Убирает конкретный файл из контроля версий, но физически оставляет его на своём месте

$ git mv [оригинальный файл] [новое имя]

Перемещает и переименовывает указанный файл, сразу индексируя его для последующего коммита

## Игнорирование некоторых файлов

Исключение временных и вторичных файлов и директорий

\*.log

build/

temp-\*

Git будет игнорировать файлы и директории, перечисленные в файле .gitignore с помощью wildcard синтаксиса

$ git ls-files --others --ignored --exclude-standard

Список всех игнорируемых файлов в текущем проекте

## Сохранение фрагментов

Сохранение и восстановление незавершённых изменений

$ git stash

Временно сохраняет все незафиксированные изменения отслеживаемых файлов

$ git stash pop

Восстанавливает состояние ранее сохранённых версий файлов

$ git stash list

Выводит список всех временных сохранений

$ git stash drop

Сбрасывает последние временно сохранённыe изменения

## Просмотр истории

Просмотр и изучение истории изменений файлов проекта

$ git log

История коммитов для текущей ветки

$ git log --follow [файл]

История изменений конкретного файла, включая его переименование

$ git diff [первая ветка]...[вторая ветка]

Показывает разницу между содержанием коммитов двух веток

$ git show [коммит]

Выводит информацию и показывает изменения в выбранном коммите

## Откат коммитов

Удаление ошибок и корректировка созданной истории

$ git reset [коммит]

Отменяет все коммиты после заданного, оставляя все изменения в рабочей директории

$ git reset --hard [коммит]

Сбрасывает всю историю вместе с состоянием рабочей директории до указанного коммита.

## Синхронизация с удалённым репозиторием

Регистрация удалённого репозитория и обмен изменениями

$ git fetch [удалённый репозиторий]

Скачивает всю историю из удалённого репозитория

$ git merge [удалённый репозиторий]/[ветка]

Вносит изменения из ветки удалённого репозитория в текущую ветку локального репозитория

$ git push [удалённый репозиторий] [ветка]

Загружает все изменения локальной ветки в удалённый репозиторий

$ git pull

Загружает историю из удалённого репозитория и объединяет её с локальной. pull = fetch + merge

|  |  |
| --- | --- |
|  | github\_pat\_11A5XGGAI0Gjdd0Vkp6aKg\_UBosgnJixrQWIeb6o5lOoT6r1TnobHMjlnkyRfkU7yx7VHFFE2AsqCPUUVT |
|  | TulinVM |

https://github.com/TulinVM/Skillfactory/blob/main/krest\_null.py

## 7. GitHub

**GitHub** — веб-сервис, который основан на системе Git. Его часто называют «социальной сетью для разработчиков», которая помогает удобно вести коллективную разработку IT-проектов. Здесь можно публиковать и редактировать свой код, комментировать чужие наработки, следить за новостями других пользователей. У программистов есть возможность объединять репозитории и выводить вклад каждого участника в определённый репозиторий в виде дерева.

**Ключевые особенности GitHub, помимо функций Git репозитория:**

* + Настройка вариантов совместной работы команды с файлами репозитория.
  + Публичные и личные репозитории для бесплатного аккаунта. Публичные репозитории доступны как для чтения, так и для записи другим авторизованным участникам данного сервиса, а также для чтения любому, имеющему ссылку. Приватные репозитории доступны только их владельцу.
  + Добавление в репозиторий файлов пользователями, имеющими доступ «по записи». Для этого используется интерфейс Git, установленный локально на компьютерах, или веб-интерфейс GitHub.
  + Клонирование репозитория целиком с помощью интерфейсов локально установленного Git и сервиса GitHub.
  + Создание организации в рамках своего аккаунта, добавление репозиториев и приглашение пользователей для участия. Гибкое назначение им прав на работу с файлами репозитория.
  + Функционирование в качестве полноценной системы управления проектами. Это значит, что пользователи могут создавать задачи, связывать задачи с конкретным репозиторием, определять процесс работы над задачами, привлекать других пользователей к проекту.
  + Наличие API, благодаря которому можно интегрировать GitHub в другие приложения, например, в системы управления проектами типа JIRA. Это позволяет построить интегрированную систему, в которой задачи в JIRA интегрируются с репозиториями в GitHub. Благодаря чему пользователи видят, какие изменения в коде связаны с той или иной задачей.
  + Подсветка синтаксиса для большинства языков.
  + Наличие справочной системы, wiki, для любого репозитория.
  + Платные аккаунты для работы с GitHub добавляют множество других весьма полезных функций.

В этом модуле мы затронем работу с небольшой частью функционала GitHub. Более глубоко узнать GitHub вы сможете самостоятельно, в том числе читая документацию после авторизации в этом сервисе.

Теперь вы можете создавать свои репозитории или присоединиться к работе над проектами других разработчиков.

Чтобы создать репозиторий на GitHub, воспользуйтесь следующей [инструкцией](https://help.github.com/en/articles/create-a-repo). В инструкции создаётся репозиторий hello-world, но вы можете назвать его иначе.

Теперь у вас есть локальный репозиторий и репозиторий на сервере (на github.com). Пришло время связать локальный репозиторий на вашем компьютере с удалённым репозиторием, который вы только что создали на GitHub. Чтобы это осуществить, снова перейдём в терминал.

// в папке Skillfactory

git remote add origin https://github.com/<your\_name>/<remote\_repository\_name>.git

Обратите внимание, что вместо <your\_name> и <remote\_repository\_name> следует указать ваш аккаунт на Github и название созданного на Github репозитория.

Остался последний шаг: пришло время добавить локальные изменения (коммит) на удалённый сервер. Для этого в терминале выполните команду:

// в папке Skillfactory

git push origin master

Терминал запросит ваш Username и пароль.

Теперь вы можете видеть изменения в репозитории по адресу:

https://github.com/<your\_name>/<remote\_repository\_name>

## ****Альтернативные способы синхронизации****

Локальный и удалённый репозиторий можно синхронизировать **разными способами**. В отличие от способа выше, можно сначала создать удалённый репозиторий, а потом подключить к нему локальный. То есть процесс будет в другом порядке и с новой командой:

git clone

Пример:

git clone https://github.com/TulinVM/Skillfactory

## ****GitHub Pages****

Также можно перенести файлы с компьютера на удалённый репозиторий вручную, как это сделал наш коллега из другого курса. В видео ниже он демонстрирует, как выложить ваш проект в сеть для демонстрации заказчику или проверки ментором.Эта функция реализуется с помощью настройки GitHubPages.

GitHub позволяет не только хранить ваши проекты в виде репозиториев, но и использовать его как **хостинг** для фронтэнд проекта. То есть для такого проекта, где есть только HTML, CSS и JS. Использовать его с бэкендом или CMS и базой данных не получится.

Чтобы превратить репозиторий в полноценный сайт, достаточно открыть страницу вашего репозитория на GitHub, открыть вкладку Настроек (Settings):

В ней найти раздел **Github Pages**:

В корневом каталоге репозитория при этом обязательно должен лежать файл index.html, иначе браузер не сможет открыть страницу по адресу. Для серверов у хостинг-провайдеров это тоже верно.

Готово! Вы должны увидеть зелёное сообщение о том, что страница опубликована, и её адрес в интернете. Реальный доступ может появиться не мгновенно. Если вы видите по адресу ошибку **404**, просто подождите немного или перезагрузите страницу с помощью Ctrl+F5**.**

Помимо файлов с HTML, CSS и JS кодом, вы можете размещать на Github Pages все стандартные форматы **изображений**, которые поддерживают браузеры (jpg, png, svg).

## ****Освежим основные понятия и команды****

Система контроля версий (СКВ) Система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

GitHub Веб-сервис, предоставляющий удалённый репозиторий, который можно синхронизировать с локальным, и иметь к нему доступ с любого устройства. Обеспечивает расширенные возможности для командной разработки.

Коммит (единица измерений) Совокупность изменений, которые были внесены единоразово одним автором в конкретный момент времени.

3 особенности архитектуры Git:

Хранение изменений (в виде слепка в определённый момент).

Возможность локальных операций (история проекта хранится на локальном диске).

Целостность данных (достигается с помощью SHA-1 — алгоритма криптографического хеширования).

3 основных состояния файлов:

Изменённое — состояние, когда в файлы вносились исправления, но после этого не совершалось никаких действий.

Подготовленное для фиксации — файлы изменены и отмечены для включения в коммит.

Зафиксированное — изменения файлов сохранены в локальной базе.

3 области данных в Git:

Рабочая директория.

Индекс, в котором хранятся изменения между данными, которые зафиксированы в репозитории, и данными, которые находятся в рабочей директории.

Сам git-репозиторий.

Ветвление Практика, упрощающая работу над проектом. Позволяет вносить изменения параллельно, в другой ветке, не влияя на основную версию.

Ветка Подвижный указатель на один из созданных коммитов.

Master Текущая ветка. При создании новых коммитов, когда не создаются и не меняются ветки, вся работа производится в ней. Она указывает на последний сделанный коммит и при каждом новом коммите сдвигается вперёд автоматически.

Слияние Способ соединить обратно раздвоенную историю изменений в нескольких ветках.

git add Команда добавления файлов в индекс следующего коммита git. Используется для обозначения всех вопросов, связанных с индексированием файлов.

git commit Команда для записи индексированных изменений в репозиторий.

git status Команда для проверки статуса репозитория.

git log Команда, которая выводит историю коммитов.

git merge Команда, с помощью которой происходит слияние веток с сопутствующими изменениями в них.