**GITHUB**

**https://github.com/Egnat/New-year\_2023**

**Регистрация, пароль** [Sergey\_p\_i@mail.ru](mailto:Sergey_p_i@mail.ru) **rhpf100879 или rjpf100879**

<https://github.com/Egnat?tab=repositories> **ссылка на все репы**

**Установка на комп Виндоус**

<https://git-scm.com/>

**перейти на эту ссылку**

**https://git-scm.com/download/win**

**Комментарии для других операционных систем:**

* 1. **LINUX**
     + - Если вы никогда не пользовались терминалом, воспользуйтесь следующей [инструкцией](https://help.ubuntu.ru/wiki/%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B0), чтобы его открыть.
       - https://help.ubuntu.ru/wiki/%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F\_%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B0
       - Откройте терминал и установите приложение при помощи пакетного менеджера вашего дистрибутива. Для Ubuntu команда будет выглядеть так:

**sudo apt-get install git**

* 1. **OS X**
     + - Откройте терминал. Если вы никогда им пользовались, воспользуйтесь следующей [инструкцией](https://support.apple.com/ru-ru/guide/terminal/apd5265185d-f365-44cb-8b09-71a064a42125/mac), чтобы его открыть.
       - https://support.apple.com/ru-ru/guide/terminal/apd5265185d-f365-44cb-8b09-71a064a42125/mac
       - Для установки Git проще всего воспользоваться homebrew. Этот инструмент устанавливает нужные вам пакеты, не предоставляемые Apple. Если он не установлен, воспользуйтесь [инструкцией](https://brew.sh/index_ru) или информацией [здесь](https://it-here.ru/instruktsii/kak-ustanovit-homebrew-na-mac/).
       - <https://brew.sh/index_ru>
       - https://it-here.ru/instruktsii/kak-ustanovit-homebrew-na-mac/
       - После установки homebrew откройте терминал и запустите команду:

**brew install git**

Также может быть полезно: [Как настроить терминал на Mac](https://yablyk.com/642080-kak-nastroit-terminal-v-mac-macos-i-sdelat-ego-bolee-poleznym/).

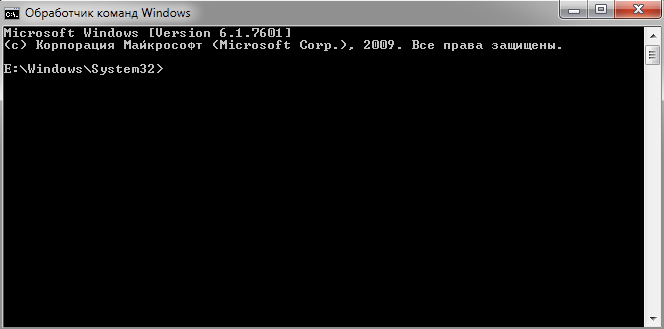
**https://yablyk.com/642080-kak-nastroit-terminal-v-mac-macos-i-sdelat-ego-bolee-poleznym/**

## ****Базовые настройки пользователя****

**У Git нет графической версии.** Поэтому работа с Git ведётся через терминал. Также его называют консолью или командной строкой. Обсуждение на тему того, в чём же между ними разница, можете прочитать [здесь](https://toster.ru/q/273105).

**Терминал** — текстовый интерфейс. Чтобы им пользоваться, нужно знать специальные команды, которые надо будет вводить с клавиатуры.

Терминал есть во всех операционных системах. Так как на данный момент самой популярной является Windows, все примеры будут проиллюстрированы на ней.



После того, как все действия по установке завершены, давайте проверим, что Git появился в системе компьютера. Откройте терминал и введите git --version. На экране терминала должна появиться текущая версия программы на вашей машине. Эта проверка подходит для всех операционных систем.

blahblahs-MacBook-Air:~ blahblah$ git --version

git version 2.26.2

**Git** — довольно гибкая система, и в ней есть множество настроек. Но мы начнём с необходимых. Для начала установим имя пользователя и адрес электронной почты. Эти данные будут отображены в каждом изменении, которое вы вносите, и другие пользователи смогут понять, кто автор тех или иных изменений.

В терминале запустите по очереди команды:

git config --global user.name "Name"

git config --global user.email email@example.ru

\*Обратите внимание, что в этих командах есть опция --global. Это значит, что вам достаточно будет один раз ввести эти данные и они будут сохранены для всех проектов. Если вы захотите поменять информацию, то нужно будет ввести ту же команду, только без  --global.

## ****Некоторые полезные команды терминала****

Для того чтобы пользоваться Git, очень полезно освоить базовые команды терминала [UNIX-систем](https://www.dmosk.ru/terminus.php?object=unix):

|  |  |
| --- | --- |
| **Название команды** | **Действие** |
| pwd | показать текущий каталог |
| ls | показать файлы в данной папке, кроме скрытых |
| ls -f | показать файлы в данной папке, включая скрытые |
| cd c: | перейти к диску (в примере диск C) |
| cd app | перейти в конкретный каталог (в примере папка app) |
| cd - | вернуться назад |
| cd .. | подняться на один уровень вверх |
| cd ../.. | подняться на два уровня вверх |
| mkdir | создать папку |

Если вы разобрались с терминалом, то можете попробовать поочередно вписать команды выше и посмотреть на результат, который будет выведен на экране.

Для ОС Windows ситуация несколько сложнее. Чтобы разобраться с терминалом, вы можете установить в качестве второй ОС Linux, например, [Ubuntu](https://ubuntu.com/" \t "_blank). Также вы можете установить терминал для Windows.

Несколько полезных ссылок для пользователей Windows:

<https://devblogs.microsoft.com/commandline/introducing-windows-terminal/>

<https://devblogs.microsoft.com/commandline/>

https://techrocks.ru/2019/08/14/best-free-terminals-for-windows/

Со списком соответствия команд в UNIX-системах и OC Windows вы можете ознакомиться [здесь](https://white55.ru/cmd-sh.html). ttps://white55.ru/cmd-sh.html

**Создание локального репозитория**

Удобнее всего иметь на компьютере папку в корневой директории и в ней вести все свои проекты. Такую папку можно назвать, например, *projects* или *dev* (от слова *development* — «разработка»).

**Как создать репозиторий:**

|  |  |
| --- | --- |
| 1 | **mkdir projects**  Создать новую папку в корневом каталоге (терминал открывается в нём по умолчанию). |
| 2 | **cd projects**  Перейти внутрь *projects*. |
| 3 | **mkdir Skillfactory**  Создать новую папку *Skillfactory*. |
| 4 | **cd Skillfactory**  Перейти внутрь *Skillfactory*. |
| 5 | **git init**  Инициализировать папку *Skillfactory* как *Git*-репозиторий. |

В терминале у вас должна появиться примерно такая запись (только с вашими данными вместо *blahblah)*:

**Initialized empty Git repository in /Users/blahblah/projects/Skillfactory/.git/**

**Blahblahs-MacBook-Air:Skillfactory blahblah$ .**

И это значит, что локальный *Git*-репозиторий создан!

Чтобы проверить статус, в котором находится ваш репозиторий, наберите в терминале:

git status

Статус репозитория показывает информацию о текущем состоянии репозитория: есть ли что-то новое в репозитории, изменились ли какие-то файлы, и если да, то какие, и так далее.

Так как пока ваш репозиторий пуст, то при проверке статуса он выдаст сообщение: On branch master

No commits yet

**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.

Важно иметь в виду, что в изменениях хранится не изменённая папка целиком, а лишь те части файлов, которые менялись. Совокупность изменений, которые были внесены единоразово одним автором в конкретный момент времени, называется **коммитом**.

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"

git commit — команда, которая создаёт коммит с рядом параметров.

Флаг -m означает, что следом идёт сообщение (message): ярлык для наших изменений.

Давайте ещё раз проверим статус репозитория при помощи команды git status.

Чтобы было легче ориентироваться, рекомендуется писать коммиты в похожем стиле, пользуясь некоторыми правилами. В каждой компании такие правила устанавливаются по-своему, но есть некоторые общие принципы. Более подробно об этом можете почитать в этой [статье](https://chris.beams.io/posts/git-commit/).

<https://cbea.ms/git-commit/>

**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. |

Если вы используете приложение с графическим интерфейсом, то add и commit вы можете также делать там.

**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.

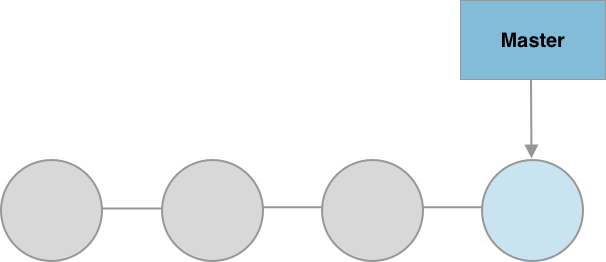
В этом модуле мы уже затронули такое понятие, как **ветка.** В каждом репозитории есть как минимум одна ветка. По умолчанию Git создаёт ветку master, где идёт основная разработка.

Допустим, у вас вдруг появилась гениальная идея по улучшению кода или реализации инновационной фичи. Но пока вы хотите посмотреть, что из этого получится, не внося изменения в уже существующий процесс. В этом вам поможет **ветвление**, которое позволяет отклониться от основного вектора разработки и внести какие-то изменения параллельно, в другой ветке, не влияя на основную версию.

Ветвление появилось давно, но в ранних системах контроля версий было дорогостоящим. С появлением Git стало возможным изменить процессы разработки в командах так, чтобы ветвление и слияние веток осуществлялись довольно часто.

Устройство системы ветвления в Git можно назвать одним из главных преимуществ этой системы контроля версий, так как система позволяет очень быстро переключаться между ветками.

Важно понимать, что **ветка** — это всего лишь указатель на коммит, не меняющий репозиторий. Когда вы делаете коммиты, не создавая и не меняя ветки, то работаете в ветке master. Она указывает на последний сделанный коммит и при каждом новом коммите сдвигается вперёд автоматически. Выглядит это примерно так:

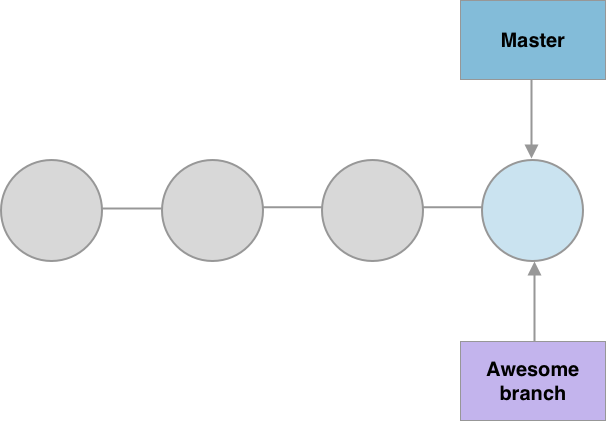


Давайте создадим новую ветку. Для этого выполним в Git-репозитории следующую команду:

git branch awesome-branch

В приложениях для Git с графическим интерфейсом это тоже можно сделать. Например, если вы используете GitHub Desktop, то можете воспользоваться этой [инструкцией](https://help.github.com/en/desktop/contributing-to-projects/creating-a-branch-for-your-work).

После того как вы создали новую ветку, состояние репозитория выглядит следующим образом:

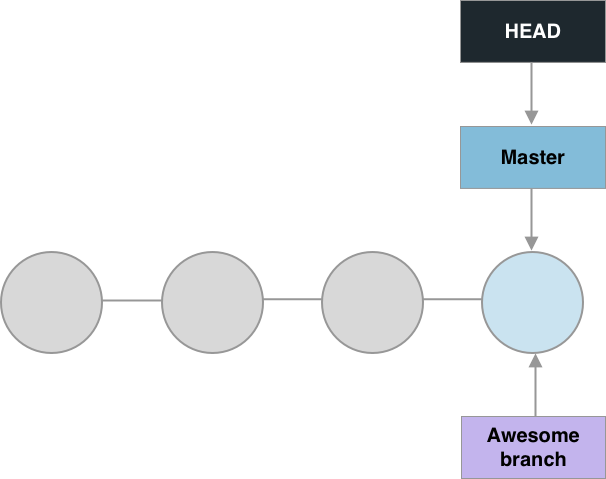


При этом история репозитория никак не изменилась. Произошло только создание нового указателя на уже существующий коммит.

Но если веток несколько и можно с ними по-разному взаимодействовать, то как Git узнаёт, в какую ветку необходимо вносить изменения? Есть какая-то «текущая»?

Да, действительно есть. Git хранит специальный указатель, который называется **HEAD** (верхушка). HEAD указывает на локальную ветку, на которой вы находитесь.

В данный момент вы всё ещё находитесь на ветке master. А как же awesome-branch? Команда git branch только создала новую ветку, она не переключила вас на неё. Так что, учитывая новое знание о HEAD, состояние системы на данный момент, на самом деле, выглядит так:



Чтобы перейти на существующую ветку, необходимо выполнить следующую команду:

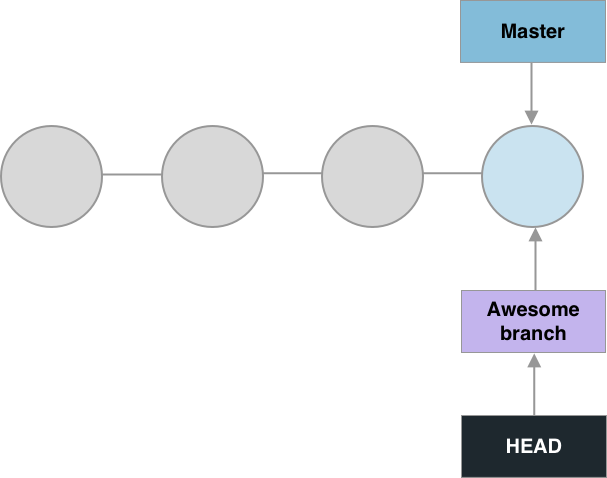
git checkout awesome-branch

Терминал зафиксирует результат сообщением:

Switched to branch 'awesome-branch'

В приложениях для Git с графическим интерфейсом тоже можно переключаться между ветками. Например, если вы используете GitHub Desktop, то можете воспользоваться этой [инструкцией](https://help.github.com/en/desktop/contributing-to-projects/switching-between-branches).

Как вы могли догадаться, после переключения HEAD указывает на другую ветку:



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.

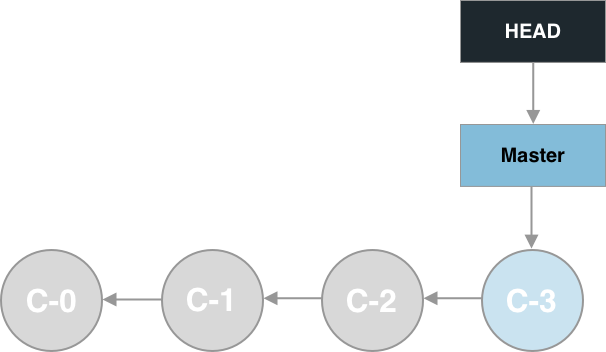
Если вы когда-либо играли в компьютерные игры, то вам легко представить, что это очень удобно и довольно интуитивно. В игре вы можете столкнуться с нелинейным сюжетом, где нужно выбрать, какой из навыков прокачивать. Например, или силу, или ум. Вы можете сохраниться в этом месте и развивать в двух разных «ветках» два разных навыка.

**Ветвление и коммиты слияния**

Ниже мы рассмотрим пример ветвления на гипотетическом проекте. Пока делать в терминале ничего не нужно — на этом этапе вам важно усвоить механику в теории. Практиковаться будем чуть позже.

Допустим, мы работаем над веб-сайтом. Предположим, что скоро Чёрная пятница и нам надо создать для работы над изменениями специальную ветку. Непосредственно перед днём распродаж надо будет сделать так, чтобы наши изменения попали к пользователям.

Представим, что работа над проектом ведётся уже какое-то время и вы уже имеете несколько коммитов:



Чтобы 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.

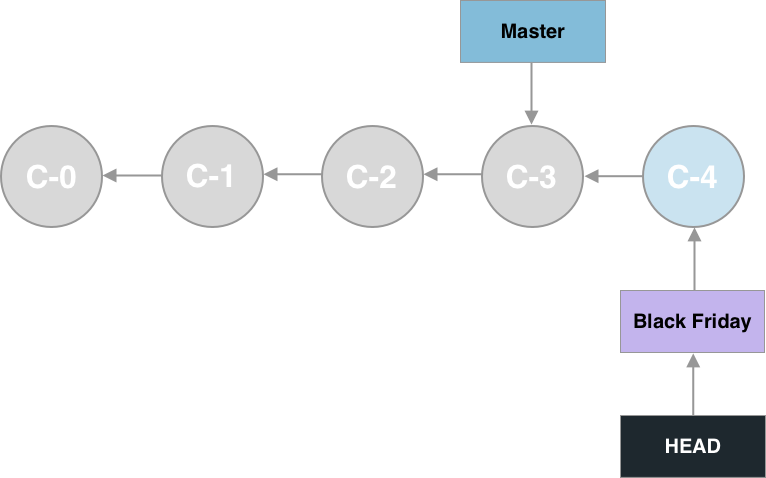
После этого мы можем менять файлы проекта и не бояться, что наш сайт как-то пострадает, ведь мы делаем это в изолированной ветке.

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

git add index.html

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

Что изменилось на этот раз?



После того как внесены все необходимые изменения, протестированы и согласованы, приходит время обновлять главную ветку *master*. Для простоты будем считать, что именно ветка мастер отвечает за версию проекта, которую все пользователи видят на сайте, хотя это не всегда так.

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?

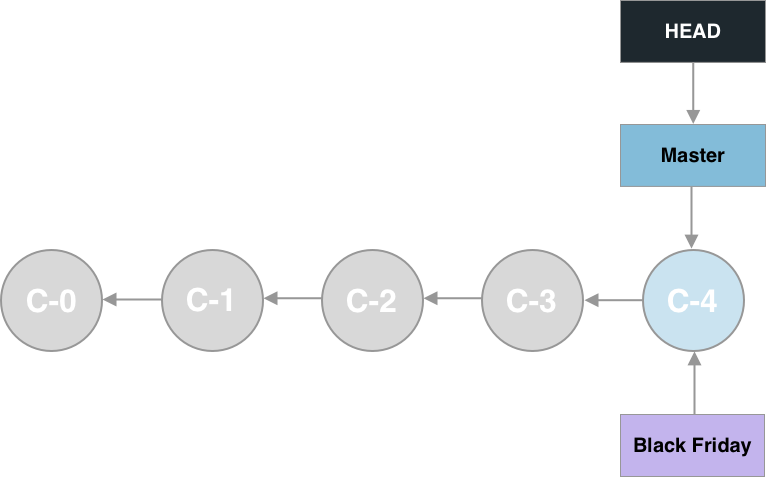
Для этого нам надо сначала переключиться на ту ветку, в которую мы хотим вливать, а затем уже «мержить».

git checkout master

git merge black-friday

Ура! Изменения для Чёрной пятницы теперь доступны всем!

Посмотрим, как выглядит репозиторий теперь:



Конечно, это очень простой пример. В реальной жизни веток бывает очень много, они разрабатываются параллельно и при необходимости сливаются (не всегда именно с главной).

Если все изменения уже находятся в главной ветке и Чёрная пятница прошла, нужна ли отдельная ветка, которая указывает на последний коммит этих изменений?

Чаще всего нет: если в ветке ничего не разрабатывается, то она продолжает указывать на один и тот же коммит, который с течением разработки остаётся где-то далеко в истории коммитов. Если вам когда-нибудь понадобится вернуться к этой версии или посмотреть, какие изменения были тогда внесены, для этого будет достаточно воспользоваться механизмом коммитов.

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

git branch -d black-friday

Как только вы перестали работать с веткой и слили все изменения с основной, можете удалять ветку без потери истории изменений. Обычно ветку оставляют до тех пор, пока не удостоверятся, что новый функционал работает как ожидалось, а затем сразу удаляют.

Даже если вы пользуетесь приложением с графическим интерфейсом и планируете делать так всегда, необходимо понимать, как работают основные команды в *Git*, ведь когда вы нажимаете ту или иную кнопку графического интерфейса, работает тот же механизм, что и при работе с командами *Git* в консоли.

**Проделаем аналогичные действия**

Пример с Чёрной пятницей пока может выглядеть для вас достаточно пространно, так как это упрощённый пример работы над реальным проектом (который у вас ещё впереди).

Создадим ветку в локальном репозитории. Откройте *Git CMD*. Перейдите в каталог локального репозитория (ранее мы назвали его *Skillfactory*). В настоящий момент вы находитесь в ветке *master*. Запомните, какие есть файлы в рабочем каталоге. Файлы, которые там есть, соответствуют ветке *master*.

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

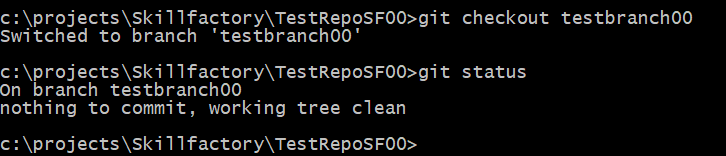
\*Вы можете использовать своё имя ветки. Далее по тексту будет использоваться имя ветки *testbranch00*.

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.

Выполнение команд *git branch --list*

* 1. 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*.
  2. Выполните команду git status. Вы увидите сообщение о том, что вы находитесь на ветке *testbranch00*.

Выполнение команды *git checkout*

## img****Дополнительно****

* + [Документация Git](https://git-scm.com/docs),
* **https://git-scm.com/docs**
  + [Книга «Pro Git» (на русском)](https://git-scm.com/book/ru/v2),
  + <https://git-scm.com/book/ru/v2>
  + [Гид по изучение Git от Atlassian](https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud),
  + Не открывается
  + [Шпаргалка](https://github.github.com/training-kit/downloads/ru/github-git-cheat-sheet/),
  + <https://training.github.com/downloads/ru/github-git-cheat-sheet/>
  + [Git Handbook](https://guides.github.com/introduction/git-handbook/),
  + <https://docs.github.com/en/get-started/using-git/about-git>
  + Поиграть с визуализацией можно [здесь](http://git-school.github.io/visualizing-git/),
  + <https://git-school.github.io/visualizing-git/>
  + Обязательно поупражняйтесь в [этом тренажёре](https://learngitbranching.js.org/).
  + https://learngitbranching.js.org/?locale=ru\_RU

**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.

**ВАЖНО!**

**«Начиная с 1 октября 2020 года, все создаваемые вами новые репозитории вместо "master" будут использовать "main" в качестве обозначения ветки по умолчанию»,— говорится в уведомлении GitHub.**

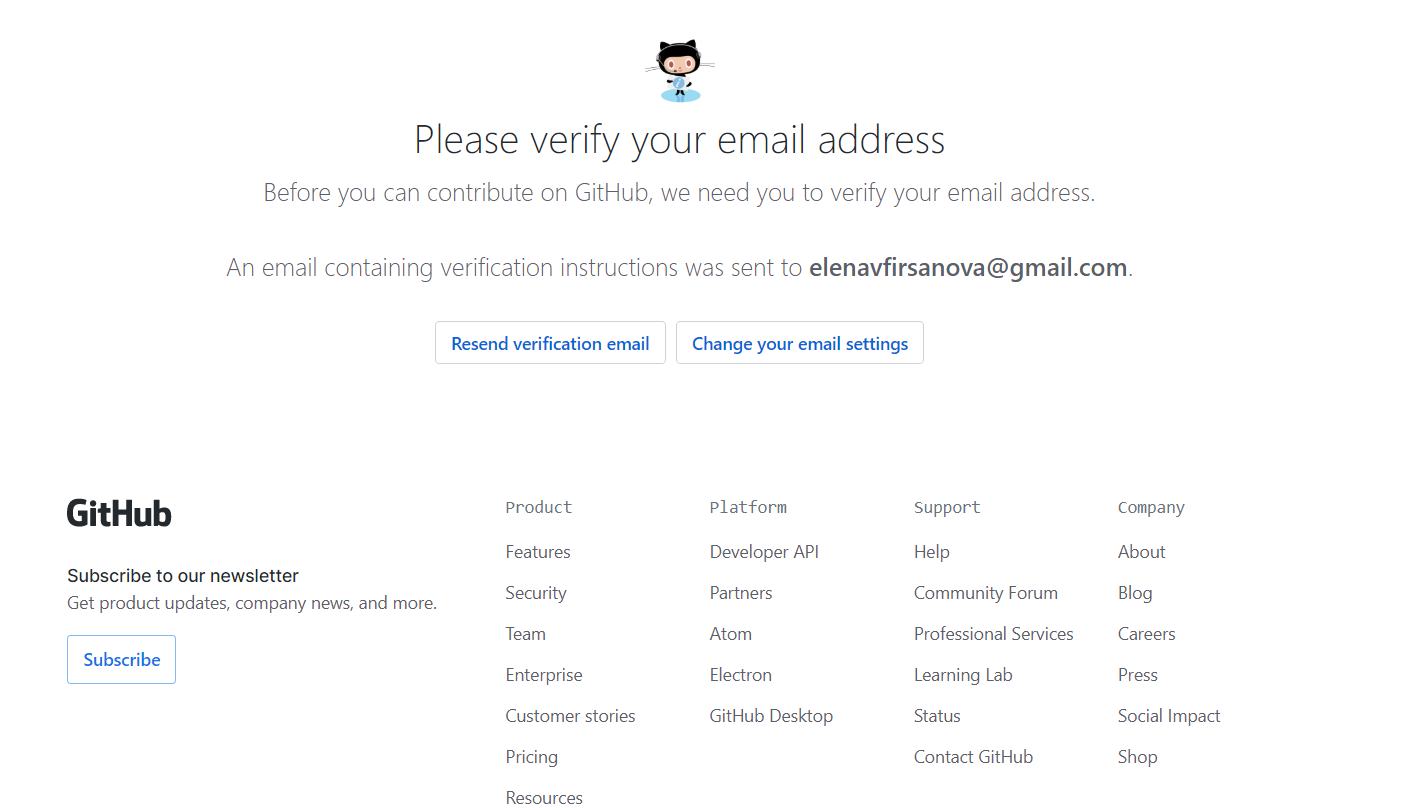
Подробнее:[**здесь**](https://www.securitylab.ru/news/512291.php)**.** **https://www.securitylab.ru/news/512291.php**

## Регистрация и создание репозитория

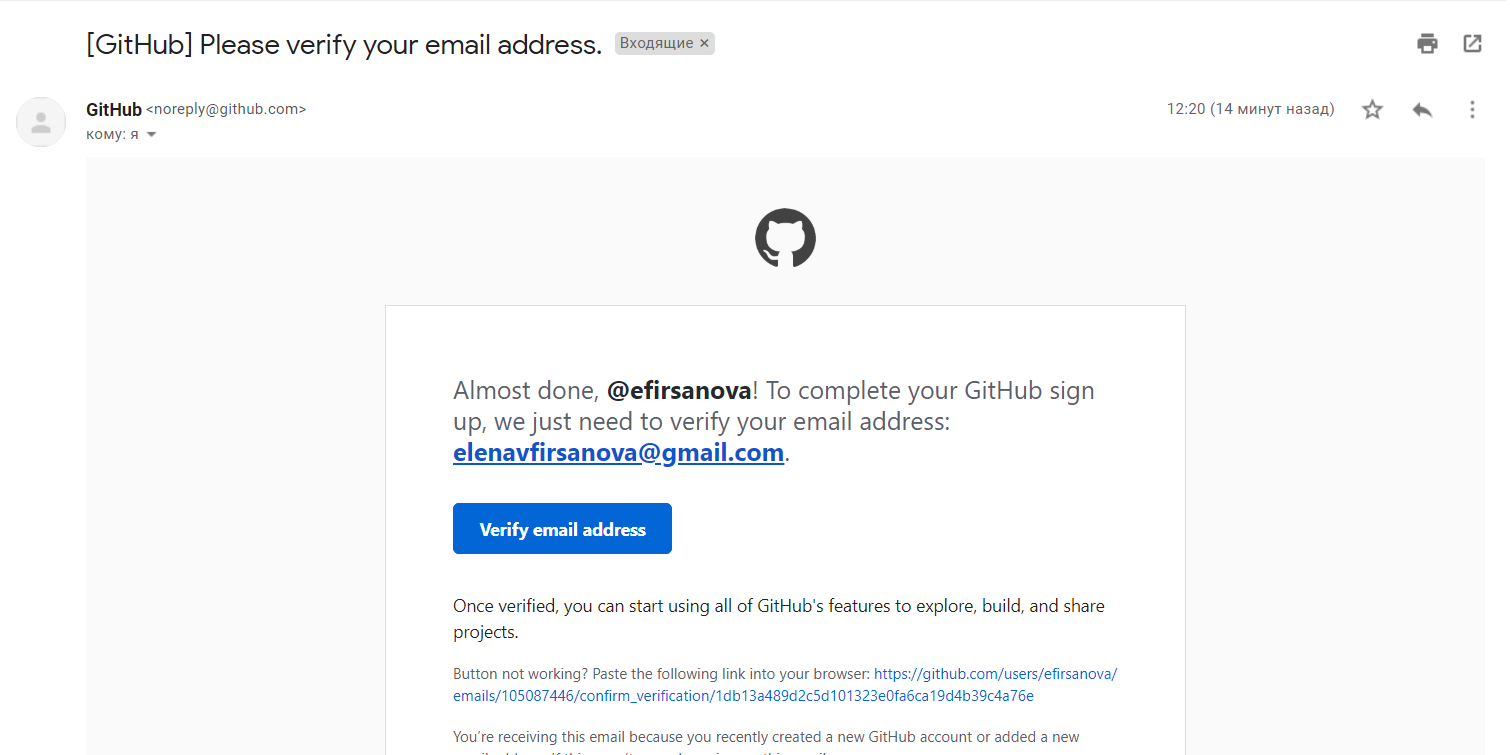
Прежде чем перейти к знакомству с GitHub, необходимо создать аккаунт на этом сервисе. Ниже есть пошаговая инструкция, но если устали от текста, сначала посмотрите это видео. https://lms.skillfactory.ru/courses/course-v1:Skillfactory+FR+2020/courseware/f095c3055e9c4b89b99b5b74ceb2dd8d/7f7b221e97814d7d9a1059be0922740e/3?activate\_block\_id=block-v1%3ASkillfactory%2BFR%2B2020%2Btype%40vertical%2Bblock%402d9598f2c9724a2ba0bf95a8f1a2706e

Выполните следующие шаги:

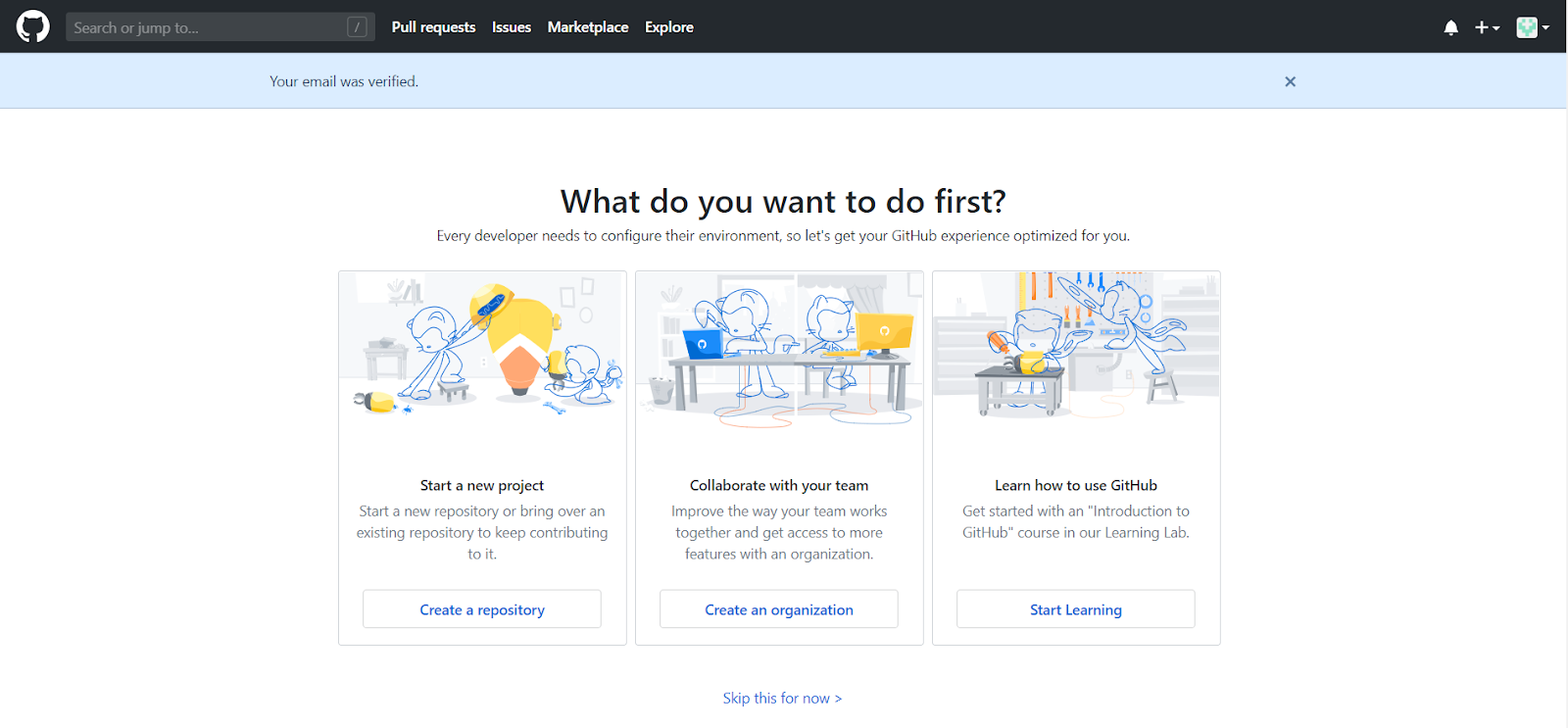
* 1. Есть ли у вас учётная запись на этом сервисе?
     + - Есть — пропустите этот шаг.
       - Нет — перейдите [по ссылке](https://github.com/) к форме регистрации.
  2. Выберите имя пользователя, почтовый ящик, пароль и кликните на Sign Up. После проверки ваших учётных данных Git предложит решить простую задачу, чтобы удостовериться, что вы человек, а не робот.
  3. Кликните на кнопку Join free plan, на открывшейся странице выберите ваши активности и то, как вы планируете использовать GIT.
  4. Кликните Complete setup, и на следующей странице GitHub предложит вам верифицировать ваш email адрес.

Верификация email

* 1. Откройте папку входящие своего email и найдите сообщение от GitHub. Откройте его и кликните в этом сообщении на кнопку Verify email address.

Сообщение от GitHub в электронной почте

* 1. После этого вы будете перенаправлены на свою страничку GitHub, где увидите сообщение об успешном подтверждении вашего email и предложение о первых шагах на GitHub.

Первые шаги на GitHub. Что сделать в первую очередь?

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>

## ****Видеоинструкция ниже:****

<https://lms.skillfactory.ru/courses/course-v1:Skillfactory+FR+2020/courseware/f095c3055e9c4b89b99b5b74ceb2dd8d/7f7b221e97814d7d9a1059be0922740e/3?activate_block_id=block-v1%3ASkillfactory%2BFR%2B2020%2Btype%40vertical%2Bblock%402d9598f2c9724a2ba0bf95a8f1a2706e>

**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

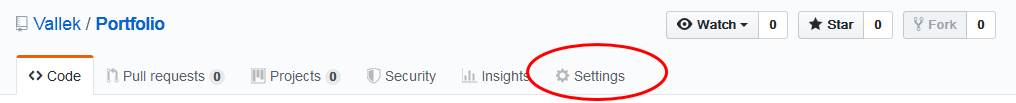
**GitHub Pages**

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.*

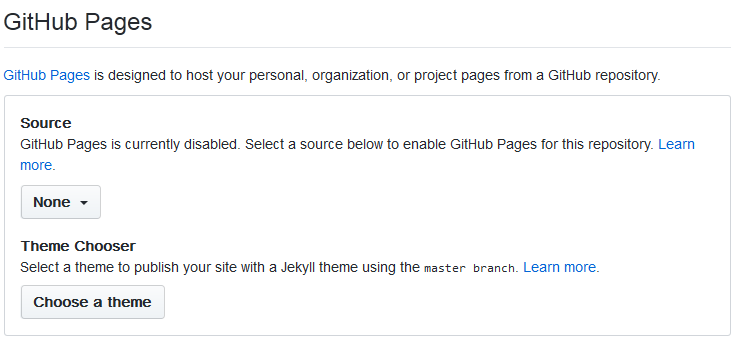
**Потренируйтесь добавлять файлы таким образом, используя результат вёрстки, который получился у вас в модуле 5.**

*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.

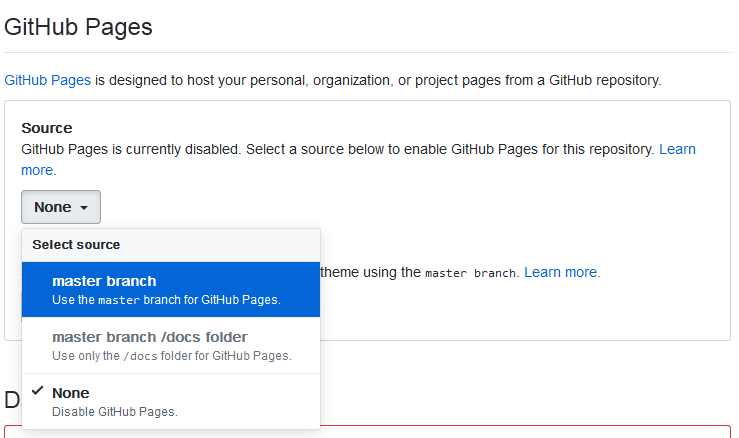
Чтобы превратить репозиторий в полноценный сайт, достаточно открыть страницу вашего репозитория на *GitHub*, открыть вкладку Настроек (*Settings*):



В ней найти раздел **Github Pages**:



В нём выбрать источник для страницы. Как правило, это *master* ветка:



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**.**

|  |  |
| --- | --- |
| **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 указываетна 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. |

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

## ****Дополнительно****

* + Всё, что мы делали с помощью терминала, можно делать и в приложениях с графическим интерфейсом. Для работы в них ознакомьтесь с возможностями интерфейса. Например, [GitHub Desktop](https://help.github.com/en/desktop/getting-started-with-github-desktop" \t "_blank).
  + <https://docs.github.com/en/desktop/installing-and-configuring-github-desktop>
  + Всё, что вы хотели знать о GitHub, можно найти [здесь](https://help.github.com/en#dotcom).
  + https://support.github.com/#dotcom

Итак, если вы проходили модуль планомерно, то сейчас у вас в арсенале должен быть следующий набор:

* + Локальный репозиторий, с которым вы работаете через терминал или графический интерфейс.
  + Личная страница на *GitHub* с удалённым репозиторием, который вы можете синхронизировать с локальным разными способами.
  + Несколько основных команд для манипуляций с репозиторием.

Даже если всё вышеперечисленное удалось освоить с блеском, у вас может остаться неоднозначное впечатление от инструмента*Git*. Опытные разработчики подтверждают — не просто осознать роль этого инструмента пока тренируешься на тестовых файлах.

Этот модуль был призван обучить вас работать с командной строкой, «набить руку» на основных командах, переключаться между репозиториями. А когда дело дойдёт до разработки большого проекта, полученные навыки позволят вам быстрее сориентироваться в процессе разработки и в целом чувствовать себя увереннее наравне с коллегами.

**Освежим основные понятия и команды**

https://lms.skillfactory.ru/courses/course-v1:Skillfactory+FR+2020/courseware/f095c3055e9c4b89b99b5b74ceb2dd8d/7f7b221e97814d7d9a1059be0922740e/3?activate\_block\_id=block-v1%3ASkillfactory%2BFR%2B2020%2Btype%40vertical%2Bblock%402d9598f2c9724a2ba0bf95a8f1a2706e

**git init  
Эта команда создаст .git репозиторий в Вашем проекте. Репозиторий или “repo” это коллекция всех изменений, которые были совершены на протяжении всего времени после инициализации репозитория. Это первое что нужно сделать для нового проекта.**

**После того как я создал файлы в папке я должен их закомитить git commit, потом появится окно, в нём нужно вводить команду initinal commit => shift z => shift z . Файлы закомичены , git status – проверяет файлы на наличие закомиченной инфы. Следующие команды в работе с файлами в папке(моём репозитории). Git add . добавляет инфу в файлах, снова git status, её лучше вводить чаще для проверки состояния моего репозитория, git commit –m (“ здесь лучше писать название проекта, папки, в которых были изменения и название ветки, в которой работаю, на пример, master”) – коммитит все изменнеия сразу во всех папках.**

**Создать нужно репу в гите. Заходим в гит, справа есть меню Ваши репы. Создать репу и дальше по выбору, что мне нужно частный или общественный.**

**Подключить комп к гитхабу -**

**git remote add origin** [**https://github.com/Egnat/hello-world.git**](https://github.com/Egnat/hello-world.git)

**После того, когда моя репа соединена с репой гита и файлы готовы, отправляем их в репу гита git push origin master или название ветки, на которой делался проект**

### Копирует инфу с гитхаба (проект) - git pull origin master или название др. ветки

Скачать чужой репу. Создаю у себя в компе папку, через терминал вхожу в нее cd название папки enter git clone скопированная ссылка чужой репы enter

**Команда « touch » является стандартной командой, которая используется в операционной системе Linux для создания и изменения временных меток файла. echo > — непосредственная команда, создающая файл.**

## Создать папку — mkdir или md

Создаём пустую папку code внутри папки html :

Создаём несколько пустых вложенных папок — для этого записываем их через косую черту.

## Удалить папку — rmdir или rd

При этом удалить можно только пустую папку. Если попытаться удалить папку, в которой что-то есть, увидим ошибку.

## Показать дерево папок — tree

Если вы хотите посмотреть содержимое всего диска, введите tree в корне нужного диска. Получится красивая анимация, а если файлов много, то ещё и немного медитативная.

## Удаление файла — del или erase

Команда для удаления одного или нескольких файлов.

## Переименование файла — ren или rename

Последовательно вводим ren, старое и новое имя файла.

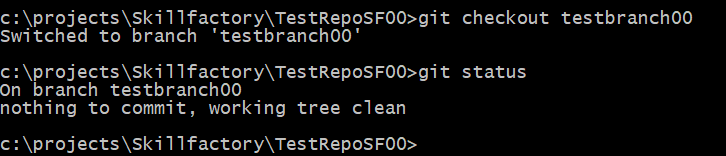
**Чтобы приступить к работе над изменениями для Чёрной пятницы, создадим специальную ветку:**

**git branch 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**

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.
5. Например, git checkout testbranch00. Вы получите сообщение о переключении на ветку *testbranch00*.
6. Выполните команду git status. Вы увидите сообщение о том, что вы находитесь на ветке *testbranch00*.



**Чтобы удалить ветку, достаточно выполнить команду:**

git branch -D black-Friday

**объединение веток** - git merge

**создать и сразу перейти на ветку –**

git checkout –b название ветки

запись изменений в файле

git commit –m “навание папки название ветки запись изменений”

копирует в комп с гитхаба - git clone

<C:\\Users\\SkillF> git clone ссылка моего реп-я https://github.com/Egnat/hello-world

1. **Если введем git log --help - он предоставит нам документацию по какой-то определенной команде (в данном случае это - log)**

## Команды одной строкой

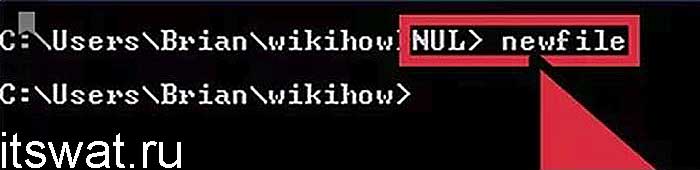
Очистить консоль — **cls**

Информация о системе — **systeminfo**

Информация о сетевых настройках — **ipconfig**

Список запущенных процессов — **tasklist**

Как создать новый текстовый файл через cmd? Для этого вводится команда NUL> с именем файла. Пример на изображении:



|  |  |
| --- | --- |
| **Название команды** | **Действие** |
| pwd | показать текущий каталог |
| ls | показать файлы в данной папке, кроме скрытых |
| ls -f | показать файлы в данной папке, включая скрытые |
| cd c: | перейти к диску (в примере диск *C*) |
| cd app | перейти в конкретный каталог (в примере папка *app*) |
| cd - | вернуться назад |
| cd .. | подняться на один уровень вверх |
| cd ../.. | подняться на два уровня вверх |
| mkdir | создать папку |

**СОЗДАНИЕ В ПРОЕКТЕ ФАЙЛА ГИТИГНОР, ЧТОБЫ НОД\_МОДУЛС НЕ КОПИРОВАЛАСЬ В ГИТХАБ**

Перед установкой библиотек мы должны создать файл с названием .gitignore  и записать в него одну строчку: node\_modules, чтобы git не учитывал сотни тысяч файлов сторонних библиотек, которые будут лежать в /node\_modules/.

