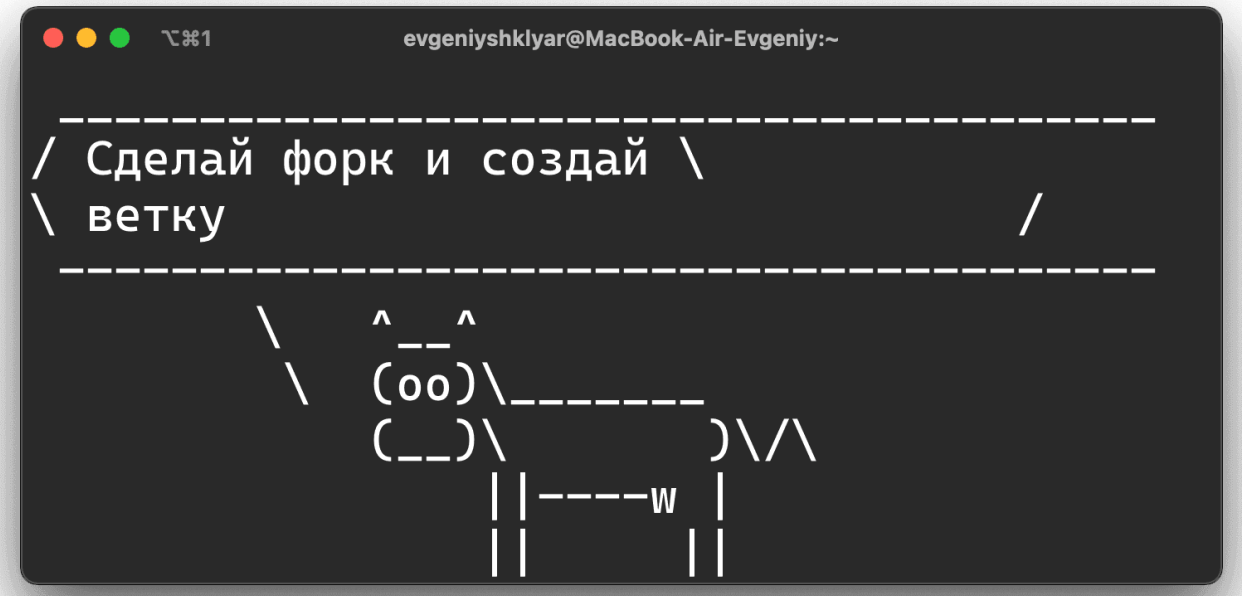
Работа с Git через консоль

7 августа 2023

Git

Валерия Зеленая

Задача: форкнуть репозиторий в GitHub, создать ветку и работать с кодом.



Сразу появляется много вопросов — что такое GitHub, какие для этого нужны команды, зачем, а главное, как всем этим пользоваться? Давайте разберёмся.

Больше из рубрики Git: введение, основные команды, решение проблем.

Когда мы пишем код, мы постоянно туда что-то добавляем, удаляем, и иногда всё может ломаться. Поэтому перед любыми изменениями стоит сделать копию проекта. Если собирать проекты в папки с именами `проект1`, `проект1_финали` `проект2_доделка`, вы быстро запутаетесь и точно что-нибудь потеряете. Поэтому для работы с кодом используют системы контроля версий.

Система контроля версий — программа, которая хранит разные версии одного документа, позволяет переключаться между ними, вносить и отслеживать изменения. Таких систем много и все они работают по принципу компьютерной игры, где вы можете вернуться к месту сохранения, если что-то пошло не так.

Git — самая популярная система контроля версий. С Git можно работать через командную строку (или терминал). В каждой системе своя встроенная программа для работы с командной строкой. В Windows это PowerShell или cmd, а в Linux или macOS — Terminal. Вместо встроенных программ можно использовать любую другую — например, Git Bash в Windows или iTerm2 для macOS.

> Как работает терминал: мы вводим команду и получаем ответ компьютера — или всё получилось, или где-то ошибка, или нужно ввести что-то ещё — например, пароль. Поэтому большая часть этой инструкции состоит из команд для терминала. Сначала будет непривычно, но вам понравится.

Но давайте по порядку — установим Git на компьютер.

Устанавливаем и настраиваем Git

Windows. Скачайте Git для Windows, запустите exe-файл, следуйте инструкциям.

macOS. Скачайте Git для macOS и запустите dmg-файл. Если он не запускается, зайдите в Системные настройки — Безопасность и нажмите кнопку *Open anyway* (Всё равно открыть).

Linux. Установите Git через встроенный менеджер пакетов. Если у вас Ubuntu, используйте команду `sudo apt-get install git`. Команды для других дистрибутивов можно посмотреть здесь.

Как проверить, что Git установился

Откройте терминал и введите команду

```
git --version
```

Если Git установлен, то вы увидите номер версии, например, `2.35.1`.

Настраиваем Git

Теперь нужно ввести имя и адрес электронной почты, чтобы ваши действия в Git были подписаны, а ещё для привязки к GitHub.

Добавить имя (введите его внутри кавычек):

```
git config --global user.name "ваше имя"
```

Добавить электронную почту (замените email@example.com на вашу почту):

```
git config --global user.email email@example.com
```

Опция `--global` значит, что имя и почта будут использоваться для всех ваших действий в Git. Если вы хотите менять эту информацию для разных проектов, то вводите эти же команды, только без опции `--global`.

Регистрируемся на GitHub

GitHub (или Гитхаб) — веб-сервис на основе Git, который помогает совместно разрабатывать IT-проекты.
На Гитхабе разработчики публикуют свой и редактируют чужой код, комментируют проекты и следят за новостями других пользователей.

Профиль на Гитхабе и все проекты в нём — ваше публичное портфолио разработчика, поэтому нужно завести профиль, если у вас его ещё нет.

1. Зайдите на сайт https://github.com и нажмите кнопку *Sign up*.
2. Введите имя пользователя (понадобится в дальнейшей работе), адрес электронной почты (такой же, как при настройке Git) и пароль.
3. На почту придёт код активации — введите на сайте.
4. Появится окно с выбором тарифного плана. Если вы пользуетесь Гитхабом для учёбы, то укажите, что профиль нужен только для вас и вы студент.
5. Опросы и выбор интересов можно пропустить.

На этом всё — вы зарегистрировались и у вас есть собственный профиль.

Устанавливаем SSH-ключи

Чтобы получить доступ к проектам на GitHub со своего компьютера и выполнять команды без постоянного ввода пароля, нужно, чтобы сервер вас узнавал. Для этого используются SSH-ключи.

> SSH — протокол для безопасного соединения между компьютерами.

SSH-ключ состоит из двух частей — открытого и закрытого ключа. Открытый ключ мы отправляем на сервер. Его можно не прятать от всех и не переживать, что кто-то его украдёт, потому что без закрытого ключа он бесполезен. А вот закрытый ключ — секретная часть, доступ к нему должен быть только у вас. Это важно.

Мы будем подключаться к GitHub по SSH. Это работает так:

1. Вы отправляете какую-то информацию на GitHub, который знает ваш открытый ключ.
2. GitHub по открытому ключу понимает, что вы это вы, и отправляет что-то в ответ.
3. Только вы можете расшифровать этот ответ, потому что только у вас есть подходящий закрытый ключ.

А чтобы подключиться к GitHub с помощью SSH-ключа, сначала нужно его создать.

Проверяем SSH-ключи

Перед созданием нового SSH-ключа проверим, есть ли на компьютере другие ключи. Обычно они лежат в папке с названием `.ssh` — поэтому посмотрим, есть ли в ней что-то, с помощью команды в терминале:

```
ls -al ~/.ssh
```

Если у вас уже есть SSH-ключ, то в списке будут файлы с именами вроде `id_rsa.pub`, `id_ecdsa.pub` или `id_ed25519.pub`. А если терминал ругается, что

директории `~/.ssh` не существует, значит, у вас нет SSH-ключей. Давайте это исправим.

Создаём новый SSH-ключ

Откройте терминал и скопируйте туда эту команду. Не забудьте подставить в кавычки почту, на которую вы регистрировались на Гитхабе.

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

`ed25519` — это алгоритм для генерации ключей. Если ваша система не поддерживает алгоритм `ed25519` (и вы увидели ошибку), используйте немного другую команду с алгоритмом `rsa`:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Терминал спросит, куда сохранить ключ. Если не хотите менять имя файла, которое предлагает терминал, просто нажмите Enter.

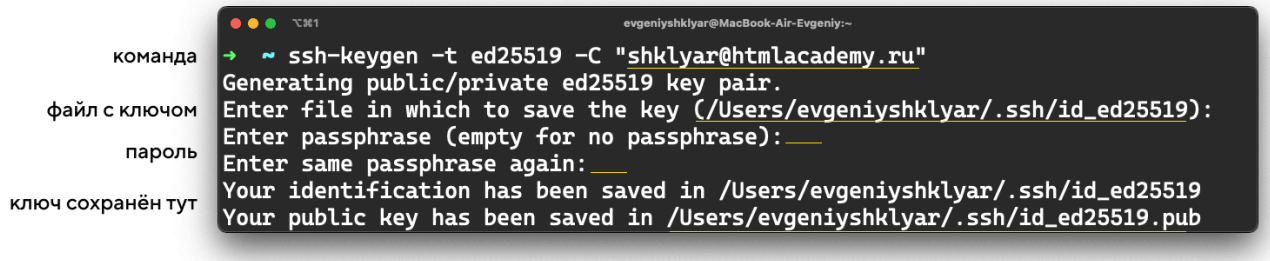
```
> Generating public/private имя-ключа key pair.
> Enter a file in which to save the key (/c/Users/ваш-профиль/.ssh/id_имя-ключа):*[Press enter]*
```

Теперь нужно добавить пароль, которым будет зашифрован ваш ключ. Это стоит сделать, иначе в дальнейшем могут быть проблемы с настройкой, да и так просто безопаснее.

В результате создаётся новый SSH-ключ, привязанный к вашей электронной почте.

Создание ключа по шагам:

Создание нового SSH-ключа `ssh-keygen -t ed25519 -C "your_email@example.com"`



Добавляем SSH-ключ в ssh-agent

`ssh-agent` — программа для хранения и управления SSH-ключами. Давайте запустим её и добавим туда наш SSH-ключ. Запускаем командой `eval "$(ssh-agent -s)"`:

```
eval "$(ssh-agent -s)"
```

Если в ответ терминал покажет надпись «Agent pid» и число — значит, всё ок, агент запущен.

Теперь добавим наш ключ командой.

```
ssh-add ~/.ssh/id_ed25519
```

Если у вашего ключа другое имя, замените название `id_ed25519` именем файла с ключом *(это правило применяется и дальше в инструкции)*. Если вы устанавливали пароль на ключ, введите его два раза после ввода команды `ssh-add` (терминал подскажет, когда это сделать).

Теперь, если всё хорошо, появится надпись *Identity added* — значит, можно переходить к добавлению ключа на GitHub.

Копируем SSH-ключ

Чтобы добавить ключ на GitHub, нужно сначала его скопировать из вашего файла командой `clip`. Вы не увидите ключ на экране, но он появится в буфере обмена, и его можно будет вставить на Гитхаб.

```
clip < ~/.ssh/id_ed25519.pub
```

Команда `clip` может не сработать на вашем компьютере, тогда есть два способа узнать ключ — простой и сложный.

Сложный способ. Найдите скрытую папку `.ssh`, откройте файл `id_ed25519.pub` в текстовом редакторе и скопируйте его содержимое.

Простой способ. Введите команду ниже и ключ появится прямо в терминале — его нужно вручную скопировать в буфер обмена. Ключ начинается с `ssh-ed22519` или `ssh-rsa` (или похожей строки) — поэтому копируйте строку прямо с самого начала.

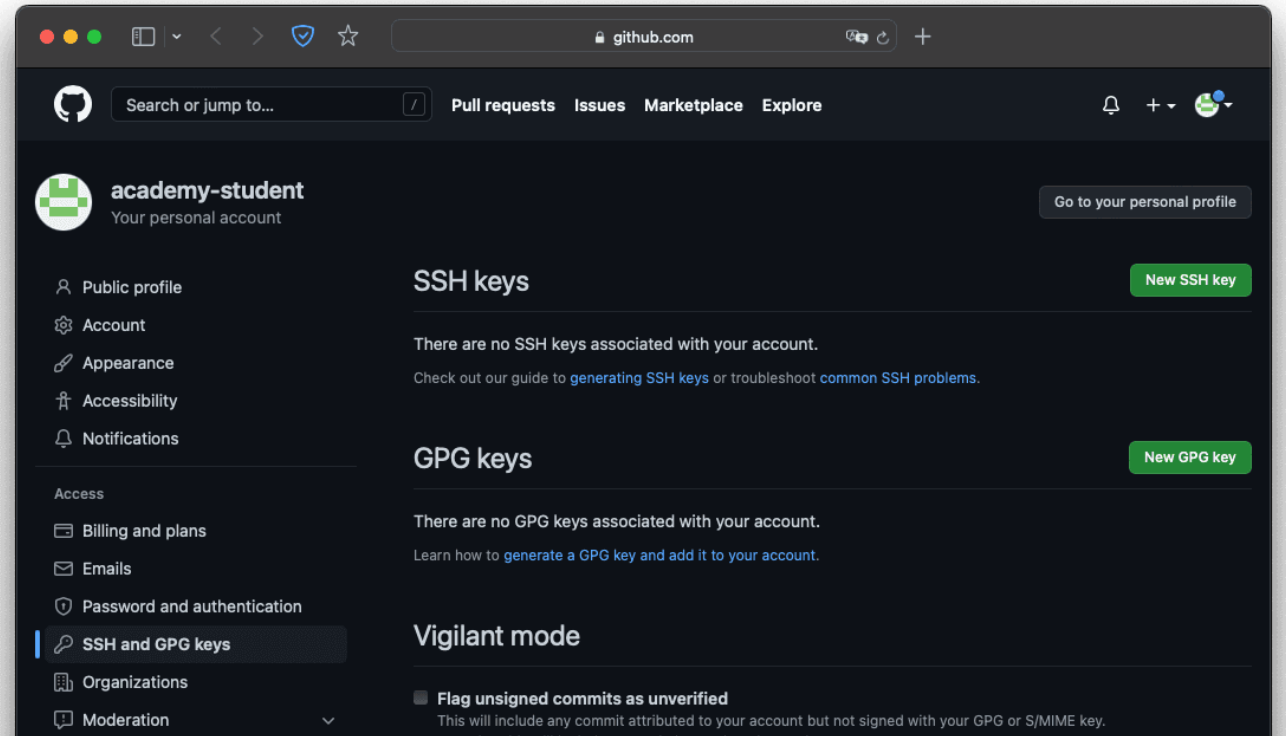
```
~ cat ~/.ssh/id_ed25519.pub

ssh-ed25519 AAAAC3NzaCZvnr4ax+Fr shklyar@htmlacademy.ru
```

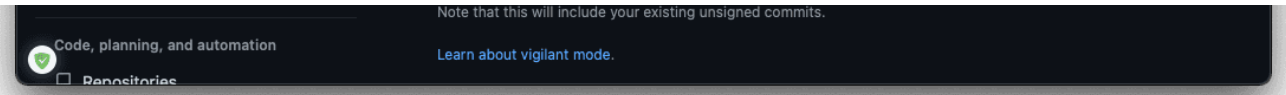
Не копируйте этот ключ из статьи — он уже не работает.

Добавляем SSH-ключ на GitHub

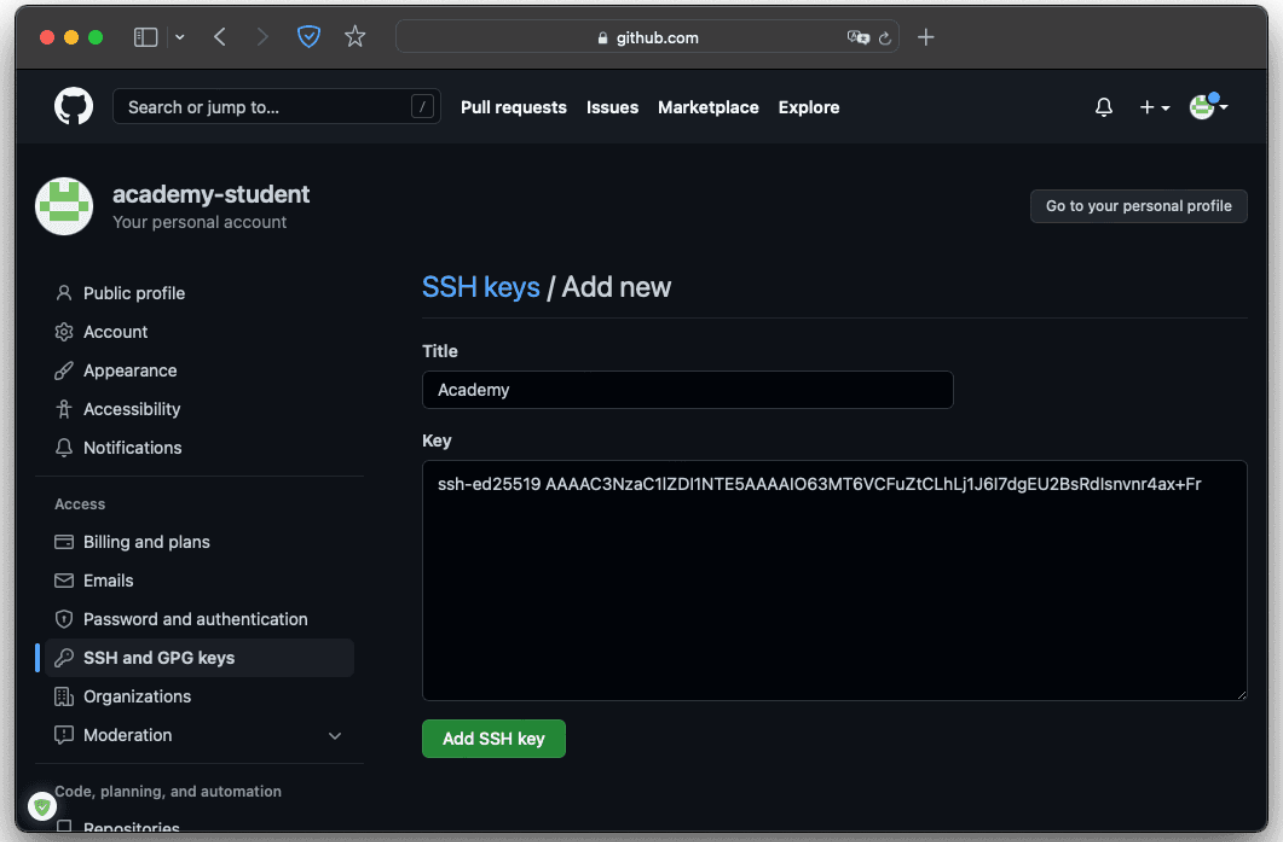
Это нужно сделать, чтобы GitHub вас узнавал.

Перейдите на страницу для работы с ключами в вашем профиле на GitHub и нажмите кнопку *New SSH key*.



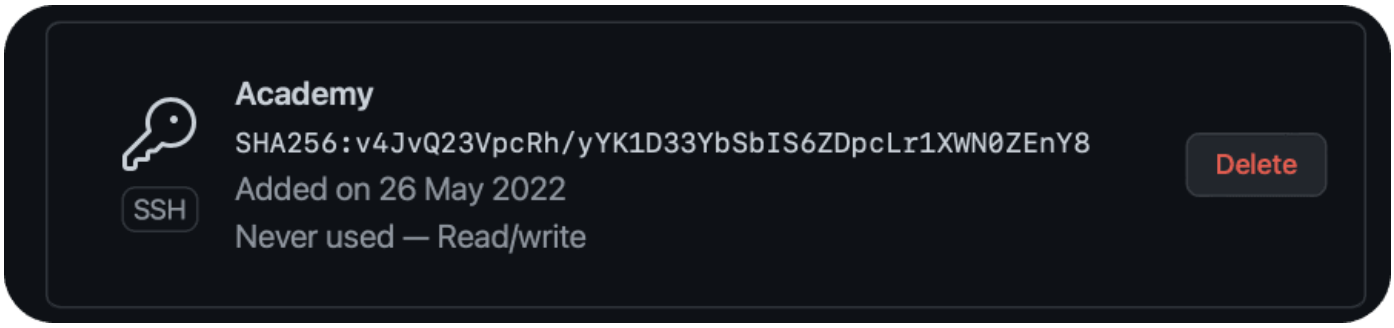


В поле *Title* нужно добавить название нового ключа. Например, если вы используете Mac, вы можете назвать ключ *MacBook Air*, или, если ключ для курсов Академии, то *Academy*. А ключ, который вы скопировали на прошлом шаге, вставьте в поле *Key*.



Не копируйте ключ со скриншота — он уже не работает.

Теперь нажмите кнопку *Add SSH key* и, если потребуется, введите свой пароль от GitHub, чтобы подтвердить сохранение. Если всё сделано верно, новый ключ появится в списке на странице https://github.com/settings/keys.



Теперь мы можем поработать с проектом в репозитории.

Что такое репозиторий

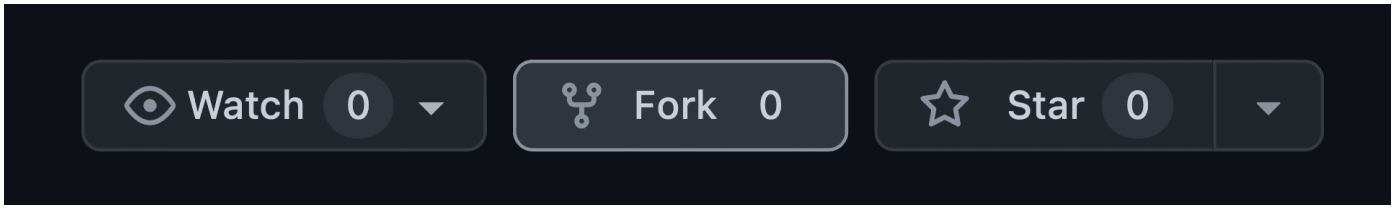
Репозиторий — папка с файлами вашего проекта на сервере GitHub. Так вы можете работать с проектом откуда

угодно, не переживая, что какие-то фаилы потеряются — все данные останутся в репозитории.

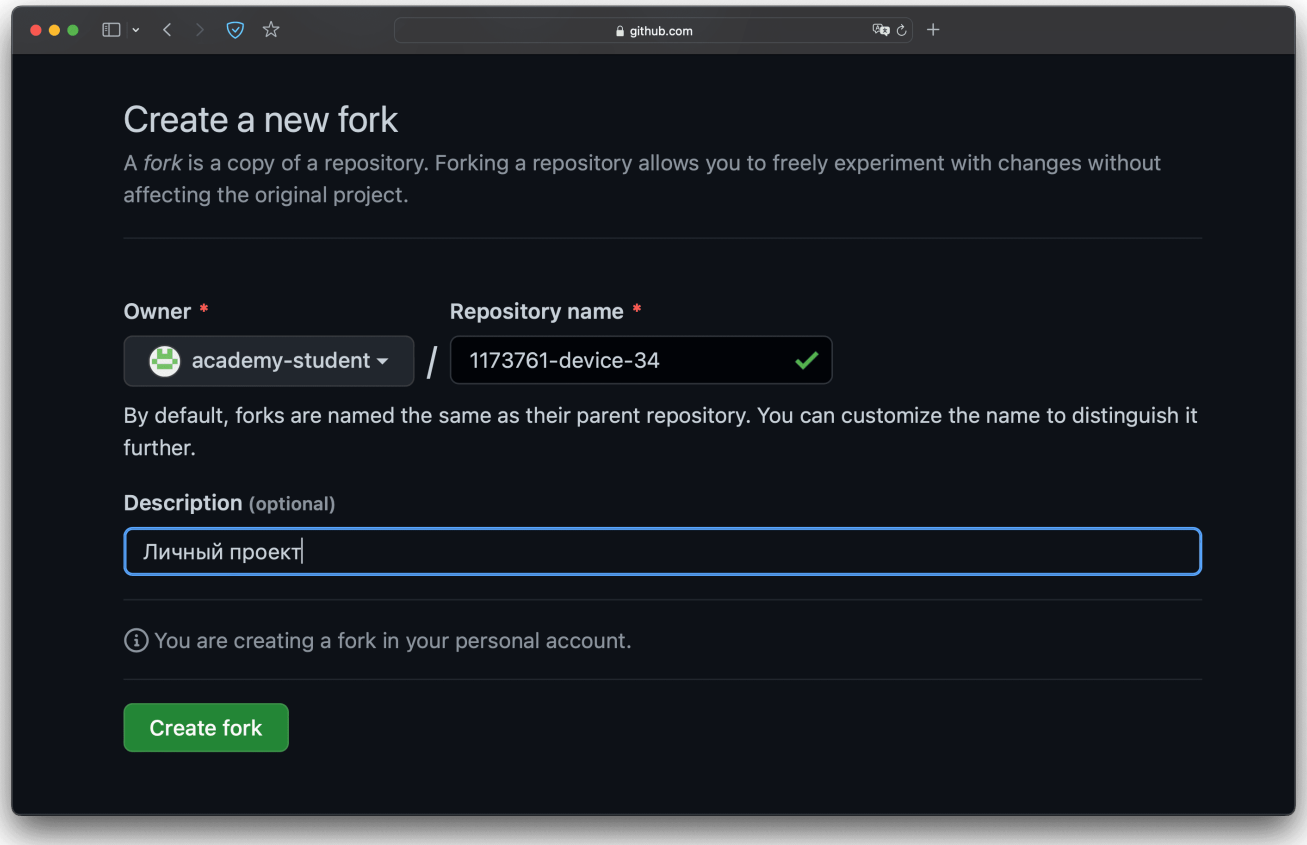
Если над проектом работает несколько программистов, сначала создаётся *мастер-репозиторий* — это общий репозиторий с рабочей версией проекта. А каждый программист работает с *форком* — то есть полной копией мастер-репозитория. В форке вы можете безнаказанно менять код и не бояться что-то сломать в основной версии проекта.

Делаем форк мастер-репозитория

Заходим в нужный репозиторий и нажимаем на «вилку» с надписью *fork*.



Появится окно *Create a new fork* — проверьте, что он называется так, как вам нужно, и жмите кнопку *Create fork*. Через пару секунд всё готово.



Клонируем форк на компьютер — git clone

Клонировать форк — значит скачать его, чтобы работать с кодом на своём компьютере. Тут нам и пригодится SSH.

Открываем терминал и переходим в папку с будущим проектом — для этого используем команду `cd your-project`. Если вы хотите, чтобы проект лежал в папке `device`, введите

```
cd device
```

Если такой папки на компьютере нет, то сначала введите `md your-project`, чтобы создать эту папку, а затем `cd your-`

`project`. Когда перейдёте в папку, введите команду `git clone` для клонирования репозитория:

```
git clone git@github.com:your-nickname/your-project.git
```

Замените `your-nickname` на ваше имя пользователя на GitHub, а `your-project` на название проекта. Проще всего их найти прямо наверху страницы репозитория.

Если вы правильно настроили SSH-ключи, Git скопирует репозиторий на ваш компьютер.

```
➜  device git clone git@github.com:academy-student/1173761-device-34.git
Клонирование в «1173761-device-34»…
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 15 (delta 0), reused 15 (delta 0), pack-reused 0
Получение объектов: 100% (15/15), 145.07 КиБ | 900.00 КиБ/с, готово.
```

Если вы видите ошибку `Error: Permission denied (publickey)`, скорее всего, вы ошиблись в настройке SSH-ключа. Вернитесь в этот раздел инструкции и повторите процесс настройки.

Кстати, если вы хотите, чтобы название папки с проектом у вас на компьютере отличалось от имени репозитория, можете дополнить команду клонирования, добавив в конце другое название:

```
git clone git@github.com:_your-nickname_/_your-project_.git folder_name
```

Теперь на вашем компьютере в папке `your_project` или в той, название которой вы указали, находится полная копия репозитория с GitHub.

В каждом репозитории есть как минимум одна основная ветка, которую создаёт сам Git — она называется `master`. Обычно в ней хранят проверенную версию программы без ошибок.

А если вы хотите исправить ошибку в коде или добавить что-то в проект, но не хотите сломать код в основной ветке, нужно создать новую ветку из `master` и работать из неё. Каждая ветка — что-то вроде второстепенной дороги, которая затем снова соединится с основной.

Создаём новую ветку — git branch

Откройте терминал и введите команду

```
git branch
```

Она показывает список веток, с которыми мы работаем в проекте, и выделяет текущую. Если мы находимся в `master` , то создаём новую ветку командой

```
git checkout -b имя-новой-ветки.
```

```
➜  1173761-device-34 git:(master) git checkout -b task1
Переключено на новую ветку «task1»
➜  1173761-device-34 git:(task1)
```

Если текущая ветка не `master`, переключитесь на неё с помощью команды `checkout`. После `git checkout` надо указать название нужной ветки.

```
git checkout master
```

```
git checkout master
```

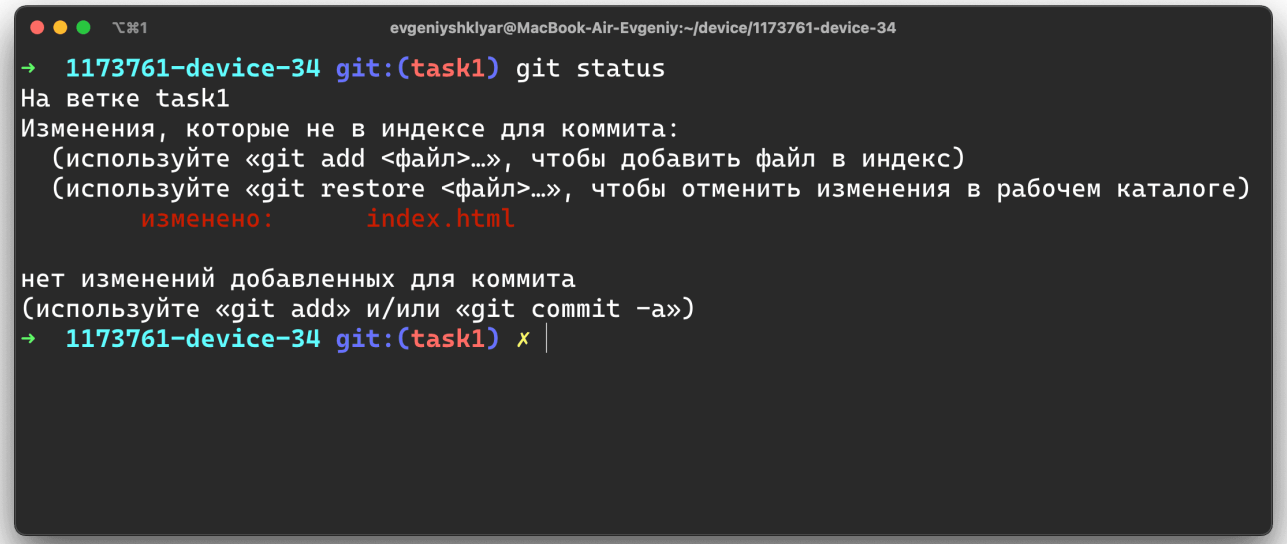
Мы делаем это, чтобы новая ветка содержала свежую рабочую версию проекта. Если вы ошиблись в названии, например, допустили опечатку, вы можете изменить название ветки с помощью команды:

```
git branch -m старое-имя-ветки новое-имя-ветки.
```

Сохраняем изменения — git add

После того, как вы создали ветку и поработали в ней у себя на компьютере, нужно сохранить результат, чтобы появился в репозитории и не пропал.

Если вы хотите сохранить изменения не во всех файлах, для начала введите команду `git status`. Она покажет текущее состояние в вашей ветке, а именно список с названиями изменённых файлов, если они есть, и укажет на те, которые ожидают записи и сохранения (обычно они выделены красным цветом).



Чтобы сохранить все изменения разом, используйте команду

```
git add -A
```

Чтобы сохранить изменения только отдельных файлов, укажите их имена вручную. Например, если вы изменили файл `index.html`, введите

```
git add index.html
```

Если название очень длинное, вы начните его писать, нажмите Tab и терминал сам предложит продолжение пути к файлу.

Делаем коммит — git commit

Сделать коммит — значит зафиксировать все сохранённые изменения и дать им название. Это делается с помощью команды `commit`

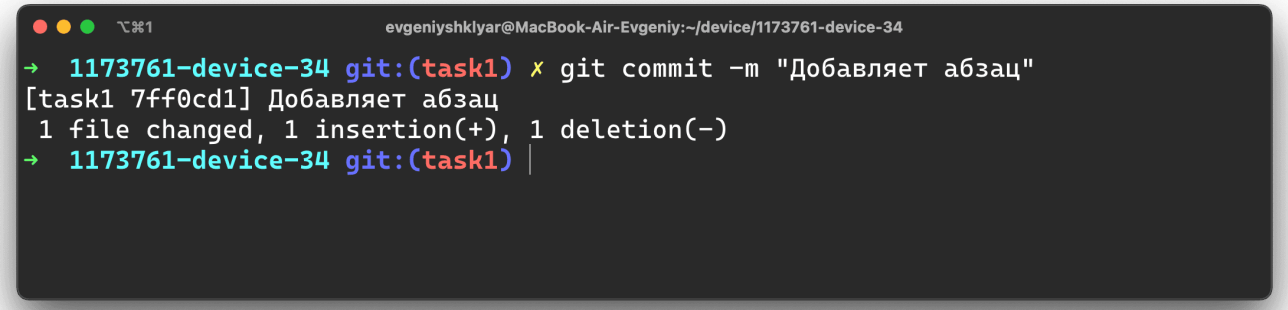
```
git commit -m "ваше сообщение"
```

Текст сообщения должен быть лаконичным и вместе с этим сообщать о том, что делает коммит (внесённые изменения). Например,

— Добавляет имя наставника в Readme

— Вводит функцию сортировки изображений

— Правит ошибку в поиске городов на карте

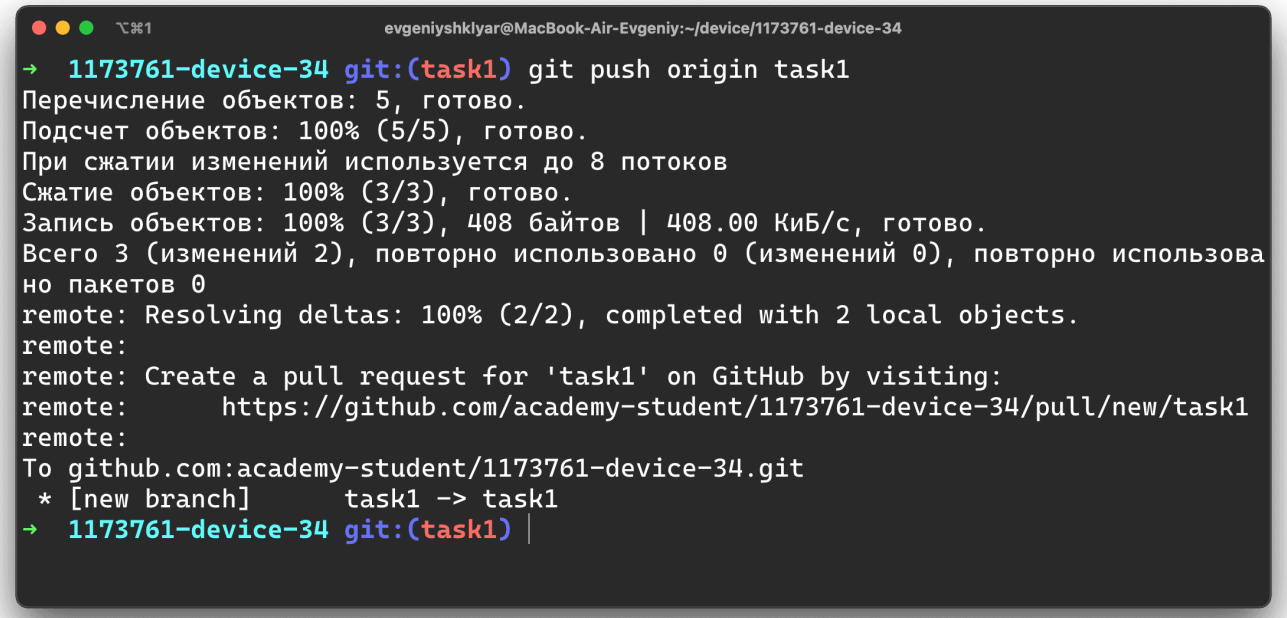


Отправляем изменения на GitHub — git push

Сохранённые изменения пока не видны коллегам, потому что находятся в нашем локальном репозитории. Нужно отправить коммиты на GitHub. Для этого введите команду

```
git push origin название-текущей-ветки
```

Где `origin` означает репозиторий на компьютере, то есть ваш форк. Слово `origin` — часть команды, не меняйте это название на своё.

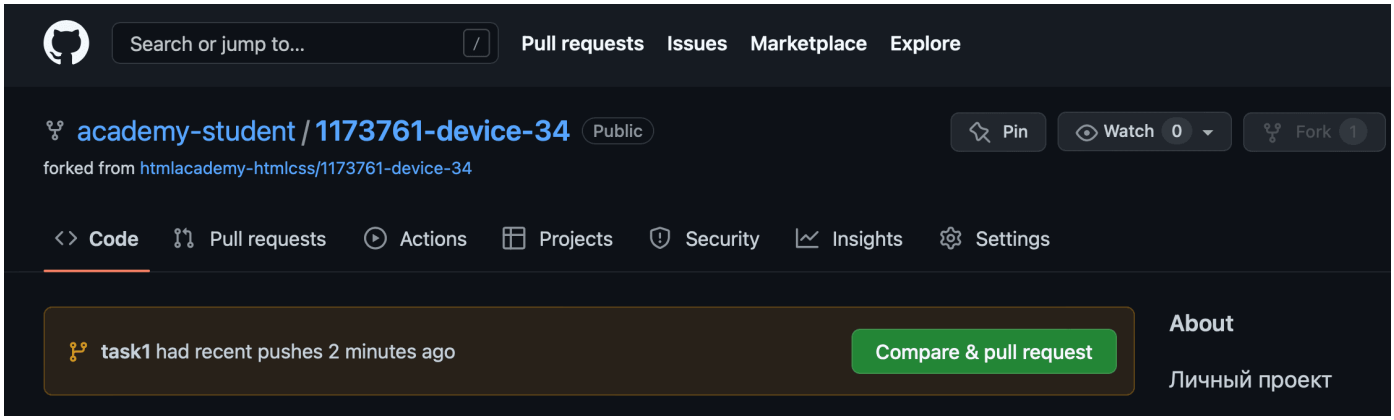


Создаём пулреквест

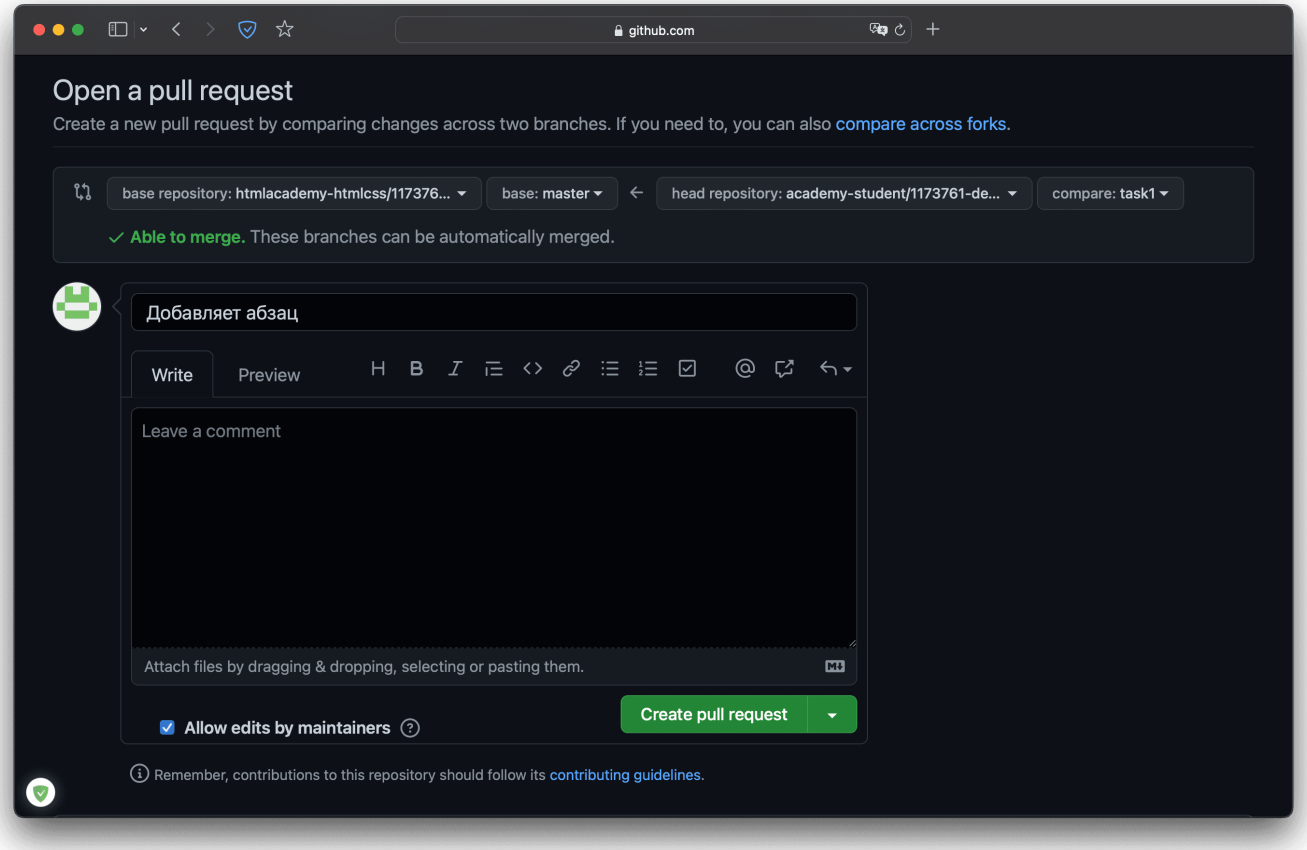
Пулреквест (или PR) — это предложение изменить код в репозитории. PR должен проверить администратор мастер-репозитория — это может быть коллега-разработчик, техлид или наставник на курсе.

Если к коду нет вопросов, пулреквест принимается. Если нужно что-то исправить — отклоняется, и придётся исправить код и снова пройти цепочку `git add` — `git commit` — `git push`. Если вы и дальше работаете в той же ветке, а пулреквест ещё не принят, все ваши изменения автоматически добавятся в пулреквест, созданный из этой ветки после команды `git push origin название-текущей-ветки`.

Чтобы создать пулреквест, зайдите на страницу вашего форка на GitHub. Вверху появилась плашка *Compare & pull request*, а ещё можно зайти на вкладку *Pull Requests*.



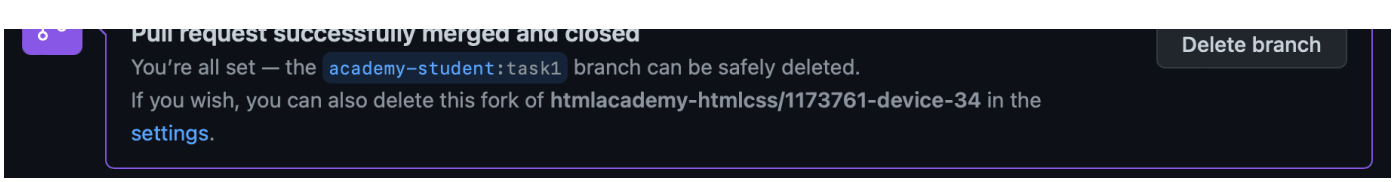
Нажмите на неё и окажетесь на странице открытия пулреквеста. Проверьте описание и нажмите *Create pull request*.



Готово, теперь ждём остаётся ждать одобрения пулреквеста или комментариев к нему.

Синхронизируем репозитории

Предположим, вы исправили код, руководитель или наставник одобрил ваши правки и принял пулреквест.



Теперь код в мастер-репозитории обновился, а в вашем форке нет, вы ведь не обновляли свою версию репозитория с тех пор, как клонировали её себе на компьютер. Приведём форк в актуальное состояние.

В локальном репозитории переключаемся на ветку `master`.

```
git checkout master
```

Забираем изменения из ветки `master` мастер-репозитория

```
git pull git@github.com:academy-student/1173761-device-34.git master
```

Отправляем изменения уже из своей ветки `master` в ваш форк на GitHub с помощью команды

```
git push origin master
```

Готово, теперь форк и оригинальный репозиторий находятся в актуальном состоянии.

Словарик

Система контроля версий — программа, которая хранит разные версии одного документа, позволяет переключаться между ними, вносить и отслеживать изменения.

Git — самая популярная система контроля версий. С Git можно работать через терминал.

Как работает терминал: мы вводим команду и получаем ответ компьютера — или всё получилось, или где-то ошибка, или нужно ввести что-то ещё.

GitHub (или Гитхаб) — веб-сервис, основанный на Git, который помогает совместно разрабатывать IT-проекты. На Гитхабе разработчики публикуют свой и редактируют чужой код, комментируют проекты и следят за новостями других пользователей.

SSH-ключ нужен, чтобы получить доступ к проектам на GitHub со своего компьютера и выполнять команды без постоянного ввода пароля, нужно, чтобы сервер нас узнавал.

ssh-agent — программа для хранения и управления SSH-ключами.

Репозиторий — папка с файлами вашего проекта на сервере GitHub или у вас на компьютере.

Мастер-репозиторий — это общий для всей команды репозиторий с рабочей версией проекта.

Форк — полная копия мастер-репозитория, в которой вы можете безопасно работать.

Клонировать форк — скачать его командой `git clone`, чтобы работать с кодом на своём компьютере.

Пулреквест (или PR) — предложение изменить код в репозитории. PR должен проверить администратор мастер-репозитория — это может быть коллега-разработчик, техлид или наставник на курсе.

Подкаст

Бесплатные учебники

Читать дальше

5 частых ошибок при работе с Git

Git — это важный и довольной понятный инструмент для контроля версий в разработке программного обеспечения, но иногда он может выдавать ошибки, которые сбивают с толку. Если вы столкнулись с одной из этих ошибок, попробуйте наше решение.

Читать дальше

Git

27 августа 2023

GitHub Desktop: обзор и первая настройка

Самая короткая инструкция о том, как сохранить файлы в GitHub и ничего не сломать. И самое главное — никакой консоли, всё через окошки и с помощью мышки. Для этого используем GitHub Desktop.

Внимание! GitHub Desktop не работает на Windows 7×32, поэтому если у вас эта версия системы, обновитесь до Windows 10 или воспользуйтесь программой GitKraken.

В этой статье идёт рассказ о системах контроля версий. Если вы совсем ничего о них не знаете, прочитайте статьи «Словарь терминов для Git и GitHub» и «Введение в системы контроля версий», чтобы понять терминологию и разобраться, зачем мы вообще это делаем.

Читать дальше

Git

7 августа 2023

Как склеить коммиты и зачем это нужно

Когда вы открываете пулреквест и ваш код смотрят и комментируют другие, бывает нужно что-то исправить. Обычно такие изменения мы комментируем сообщением вроде «Увеличил шрифт на 2px» или «Поменял оттенок фона в шапке». Такие маленькие изменения интересны, только пока они в пулреквесте. Ревьювер (человек, который смотрит ваш код) может легко узнать, что и когда вы изменили, а не читать

(человек, который смотрит ваш код), может легко узнать, что и когда вы изменили, а не читать весь `diff` заново, а вы можете легко откатить коммит, если он не нужен. Но когда приходит время вливать пулреквест, эти маленькие коммиты теряют свою ценность. Поэтому лучше их склеить в один.

Читать дальше

Git

14 июня 2023

Основные команды для работы с Git

Работа с Git через терминал — это обязательная часть практики фронтендера. Однако для начинающих разработчиков этот инструмент может показаться сложным. Чтобы вам было проще учиться, мы собрали основные команды для работы с Git.

> ☝ В некоторых командах мы будем писать URL-адрес удалённого репозитория и название проекта в квадратных скобках, вот так — `[ссылка на удалённый репозиторий]`. Мы делаем это только для наглядности. Вам квадратные скобки ставить не нужно.

Читать дальше

Git

22 февраля 2023

Как бесплатно залить сайт на GitHub Pages

Допустим, вы сделали какой-то проект, например, собрали себе портфолио по шаблону, и теперь хотите выложить его в интернет. Если вы использовали только HTML и CSS, то необязательно платить деньги, чтобы загрузить сайт куда-то. Вы можете бесплатно выложить сайт на сервис GitHub Pages. Всё, что нужно — аккаунт на Гитхабе.

Читать дальше

Git

29 ноября 2022

Регистрация на GitHub

Создание нового аккаунта на GitHub состоит всего из 10 шагов — и вся регистрация занимает меньше пяти минут.

🚧 Обратите внимания, что интерфейс Гитхаба регулярно меняется, так что внешне он может отличаться, когда вы читаете эту статью.

Начало регистрации. Так выглядит главный экран Гитхаба, когда вы не зарегистрированы. Главное, что вам нужно заметить — большое поле для ввода почты и зелёная кнопка. Вводите свой адрес и переходите на следующий шаг.

Ввод почты. На следующем шаге начинается регистрация. Подтвердите свою почту с прошлого шага и нажмите *Continue* (Продолжить).

Пароль. Придумайте сложный пароль, чтобы его никто не взломал. Например, Гитхаб просит, чтобы в пароле было не меньше 15 символов или 8 символов, но тогда должны быть и латинские буквы, и цифры.

Имя профиля. Теперь выберите имя вашего профиля — оно будет использоваться в интерфейсе, в коммитах и комментариях. То есть именно так вас будет видеть любой пользователь Гитхаба. Для разработчика Гитхаб вместо визитки, так что выбирайте что-нибудь приличное, лучше, если ник будет совпадать с вашими никнеймами на других сайтах.

Если имя недоступно, Гитхаб вам об этом скажет. А если доступно — жмите *Continue*.

Рассылки. Дальше Гитхаб спросит, хотите ли вы подписаться на рассылку об обновлениях. Впечатайте латинскую У, если хотите, или n, если письма вам не нужны. *Готовы спорить, мы знаем, что вы выберете.*

Капча, чтобы проверить, что вы не робот. Нам при регистрации пришлось два раза выбрать спиральную галактику — не сильно сложно. А если вы робот — не причиняйте вред человеку своим действием или бездействием.

Подтверждение почты. После капчи вам придёт письмо с кодом на почту. Введите его на следующей странице.

Вот здесь. Главное — не ошибайтесь.

Общая информация о вас и вашей команде. Если вы регистрируете аккаунт для себя, выбирайте *Just me*. Второй пункт — студент вы или учитель. Выбирайте «Студент», если вы не учитель.

Интересы. Дальше Гитхаб спросит вас об интересах — то есть о том, зачем вы регистрируете аккаунт. Из вариантов:

— Совместная разработка и код ревью.

— Автоматизация. CI/CD, API и другие админские вещи.

— Безопасность. Двухфакторная аутентификация, ревью, сканирование кода и списки зависимостей.

— Приложения. Выбирайте, если будете использовать GitHub Mobile, CLI, Desktop.

— Управление проектами. Проекты, метки, ишьи, вики и другие управленческие дела.

— Управление командами. Организации, приглашения, роли, домены.

— Сообщество. Выбирайте, если Гитхаб интересен вам как соцсеть.

Вы можете выбрать несколько пунктов или пропустить и не указывать ничего, для этого пролистайте страницу вниз для кнопки *Skip customization*.

Выбор тарифа. На выбор бесплатный тариф или платный GitHub Pro. Практика показывает, что для большинства личных проектов хватит бесплатного тарифа. В сентябре 2022 в него входили:

— Безлимитное количество репозиториев.

— 2000 минут CI/CD в месяц.

— 2000 минут CI/CD в месяц.

— 500 мегабайт места в хранилище пакетов.

— Поддержка сообщества.

Выбор тоже можно пропустить, тогда у вас будет бесплатный тариф.

Всё готово. Теперь у вас есть аккаунт. Можете создать репозиторий и работать с ним, или склонировать чужой. А для работы у вас есть несколько удобных вариантов:

Читать дальше

Git

28 сентября 2022

Работа с Git в Visual Studio Code

Если вы вёрстаете сайты или пишете код в редакторе Visual Studio Code, то Git за пять минут настраивается прямо внутри редактора. Не нужно запоминать команды для консоли, не нужно тыкать в лишние приложения.

Следуйте инструкции и всё получится.

Другие способы работать с Git: в терминале, в GitHub Desktop.

Читать дальше

Git

16 сентября 2022

Markdown за 5 минут

Маркдаун, он же markdown — удобный и быстрый способ разметки текста. Маркдаун используют, если недоступен HTML, а текст нужно сделать читаемым и хотя бы немного размеченным (заголовки, списки, картинки, ссылки).

Главный пример использования маркдауна, с которым мы часто сталкиваемся — файлы `readme.md`, которые есть в каждом репозитории на Гитхабе. md в имени файла это как раз сокращение от markdown.

Другой частый пример — сообщения в мессенджерах. Можно поставить звёздочки вокруг текста в Телеграме, и текст станет полужирным.

Версии маркдауна отличаются, поэтому перепроверьте, какую вы используете.

Читать дальше

Git

5 октября 2021

Шпаргалка по Git. Решение основных проблем

Поговорим о решении проблем с Git.

Читать дальше

Git

11 декабря 2020

Полезные команды для работы с Git

Работа с Git через терминал — это обязательная часть практики каждого современного фронтенд-специалиста. Однако, для начинающих это может показаться сложным. Чтобы упростить процесс обучения, мы собрали для вас все самые необходимые команды, которые пригодятся в работе с Git на первое время.

Читать дальше

Git

1 января 2020

Профессии

Фронтенд-разработчик

JavaScript-разработчик

Фулстек-разработчик

Курсы

HTML и CSS.
Профессиональная вёрстка сайтов

HTML и CSS.
Адаптивная вёрстка и автоматизация

JavaScript.
Профессиональная разработка веб-интерфейсов

JavaScript.
Архитектура клиентских приложений

React.
Разработка сложных клиентских приложений

Node.js.
Профессиональная разработка REST API

Node.js и Nest.js.
Микросервисная архитектура

TypeScript. Теория типов

Алгоритмы и структуры данных

Паттерны проектирования

Webpack

Vue.js 3. Разработка клиентских приложений

Git и GitHub

Анимация для фронтендеров

Журнал

Учебник по Git

Справочник по HTML

Истории успеха

Информация

Об Академии

О центре карьеры

Услуги

Работа наставником

Для учителей

Остальное

Написать нам

Мероприятия

Форум

Акции

Участник