

Приднестровский государственный университет им. Т.Г. Шевченко  
Инженерно-технический институт

**РАЗРАБОТКА АДАПТИВНОЙ ВЁРСТКИ ВЕБ-САЙТА  
ПРИ ПОМОЩИ ТЕХНОЛОГИЙ HTML5 И CSS3**

**Лабораторный практикум**

Разработал:  
ст. преподаватель  
кафедры ИТиАУПП  
Бричаг Д.В.

г. Тирасполь  
2021 г.

## Лабораторная работа №9

Применение контроля версий в разработке. Работа с git – репозиторием.

Хранение репозитория в облаке.

**Цель работы:** ознакомиться с принципами контроля версий. Ознакомиться с хранилищами репозиториев. Научиться пользоваться командами git и хранить кодовую базу в удалённых репозиториях. Публиковать проекты в глобальной сети при помощи автоматических сервисов. Научиться разрабатывать вёрстку при помощи новых версий CSS-фреймворков.

### Теоретическая справка

Ваш проект обладает фактически всеми атрибутами современного веб-сайта. На его страницах есть разделы, которые хорошо знакомы любому пользователю сети. От меню и шапки, до карты и контактной формы.

Отличное время продемонстрировать его миру. Можно арендовать хостинг, организовать на нём веб-сервер, загрузить туда файлы проекта, приобрести домен и сайт готов. Но тут есть несколько нюансов.

Во-первых, необходимы начальные средства для приобретения хостинга или, хотя бы домена. Во-вторых, нужны платёжные средства – международная карта или цифровые деньги, чтобы расплатиться с провайдером. В-третьих, нужны навыки настройки веб-сервера.

Допустим первые два шага не проблема, а третий уже реализован провайдером (большинство хостинг провайдеров продают настроенный для работы хост). Вы успешно развернули свой проект в сети. Показали друзьям – а они нашли баг. Совсем пустячный, но неприятный. Вы отлично помните, как это поправить: открываете код, правите, сохраняете, копируете файл, заходите в панель управления, находите файл, который хотите обновить – удаляете старую версию, загружаете новый. И всё это ради того, чтобы поправить `text-align: center`, на `text-align: left`.

А если необходимо поправить несколько файлов? Можно, конечно, перезаписать весь проект. И это будет достаточно быстро. Пока проект не очень тяжёлый. И пока файлы обновляются пользователь в сети может попасть на нерабочий сайт. Честно говоря, раньше так и было. Разработчики ставили страницу заглушку и вводили сайт на техническое обслуживание. На рисунке 178 страница технических работ на сайте twitter.com. Да, 2008 год, сейчас так большие сайты не делают.

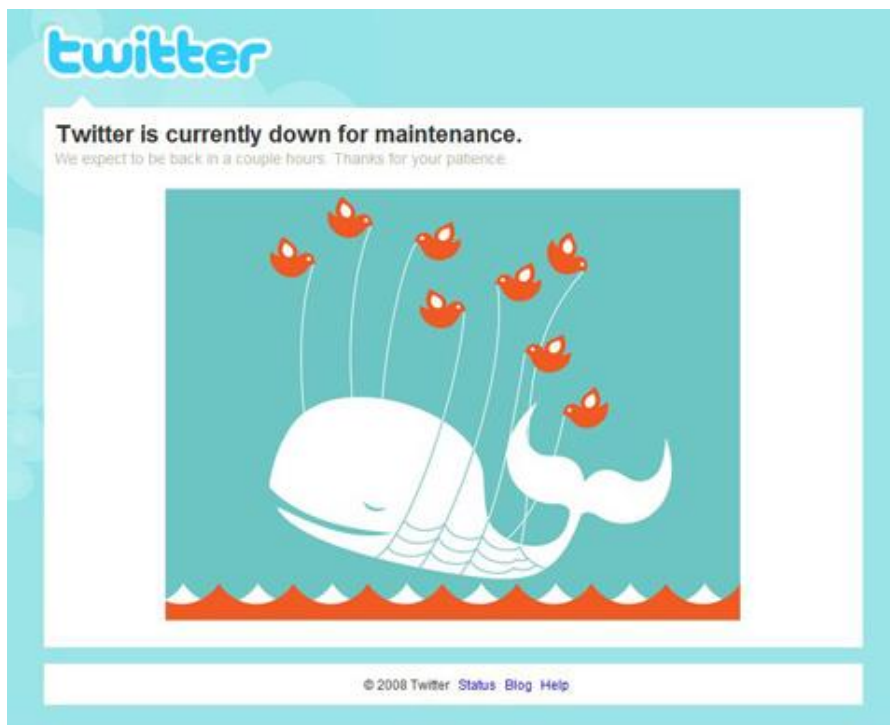


Рис. 178 – Исправления в меню

Не беда, если у вас обычный форум любителей следить за ростом елей (ничего против клуба не имею, просто ель растёт катастрофически медленно). Но если у вас онлайн магазин, то во время технических работ ценный клиент может уйти к конкурентам. Да и просто вспомните реакцию пользователей, когда «вконтакте» перестаёт отвечать в течении 10 минут.

Значит процесс нужно автоматизировать. То есть сделать так, чтобы при обновлении кода веб-приложения изменения автоматически публиковались в сети, прозрачно для пользователя.

Ещё один важный момент. Сами изменения необходимо контролировать. Представьте вы изменили 10 файлов из 30 вашего сайта. Через какое-то время заметили, что правка вызывает ошибки и нужно быстро откатить обратно код «как был». Держать после каждого изменения копию проекта? Ведь не упомянешь, что менялось в прошлый раз, а что в позапрошлый. Да и был ли раньше файл или его добавили вместе с правками. Нет, для этого есть системы контроля версий. Кроме того, они значительно упрощают совместную разработку. Ведь если 2 разработчика будут править один сайт, то нужно убедиться, что каждый из них не менял код в тех файлах, в которых работал его коллега.

Исходя из вышесказанного, можно выделить 3 основных тезиса:

1. Необходим контроль версий.
2. Необходимы инструменты совместной разработки.
3. Нужны средства автоматического обновления ПО без остановки сервиса.

## **1. Контроль версий - Git**

В качестве системы контроля версий рассмотрим git. Git (произносится «гит») — распределённая система контроля версий (далее будем называть СКВ). Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux. Программа является свободной и выпущена под лицензией GNU GPL версии 2.

Что же такое Git, если говорить коротко? Очень важно понять эту часть материала, потому что если вы поймёте, что такое Git и основы того, как он работает, тогда, возможно, вам будет гораздо проще его использовать.

## Снимки, а не различия

Основное отличие Git от любой другой СКВ (включая Subversion и её собратьев) — это подход к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы (CVS, Subversion, Perforce, Bazaar и т.д.) представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени (обычно это называют контролем версий, основанным на различиях).

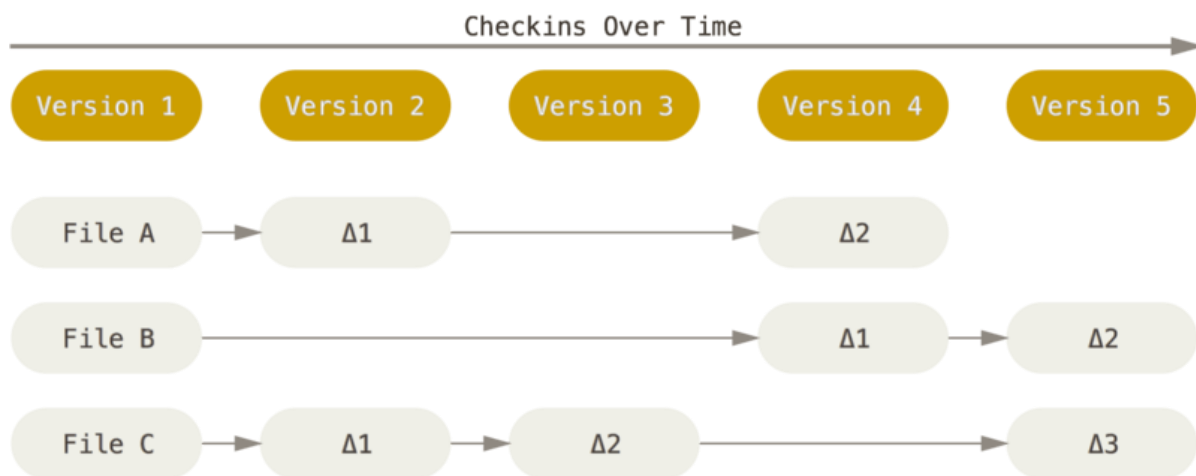


Рис. 179 – Хранение данных как набора изменений относительно первоначальной версии каждого из файлов

Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как, скажем, **поток снимков**.

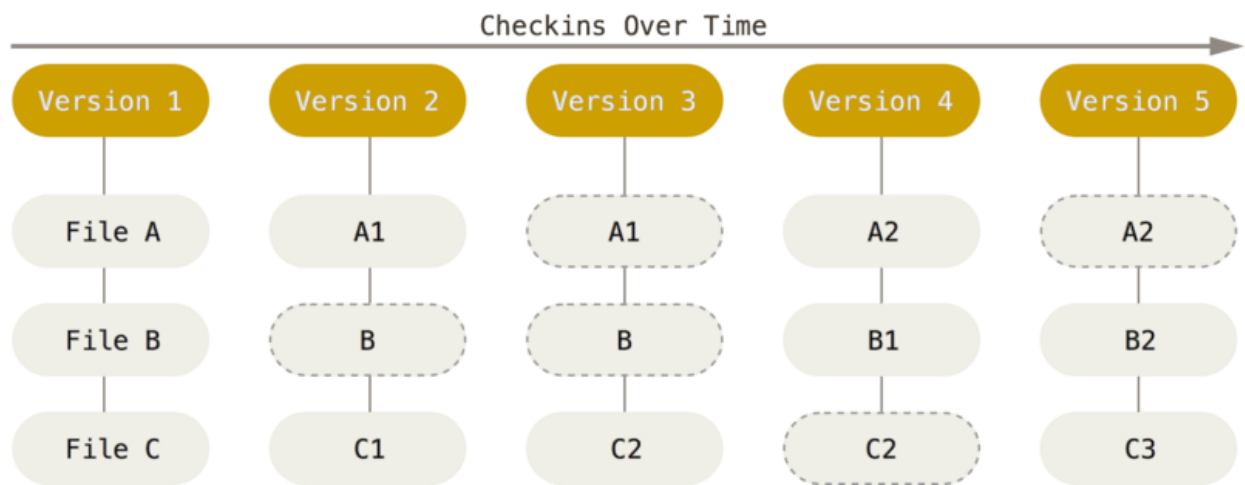


Рис. 180 – Хранение данных как снимков проекта во времени

Это очень важное отличие между Git и почти любой другой СКВ, оно делает Git больше похожим на миниатюрную файловую систему с удивительно мощными утилитами, настроенными над ней, нежели просто на СКВ.

### **Почти все операции выполняются локально**

Для работы большинства операций в Git достаточно локальных файлов и ресурсов — в основном, системе не нужна никакая информация с других компьютеров в вашей сети.

Для примера, чтобы посмотреть историю проекта, Git не нужно соединяться с сервером для её получения и отображения — система просто считывает данные напрямую из локальной базы данных. Если вам необходимо посмотреть изменения, сделанные между текущей версией файла и версией, созданной месяц назад, Git может найти файл месячной давности и локально вычислить изменения, вместо того, чтобы запрашивать удалённый сервер выполнить эту операцию, либо вместо получения старой версии файла с сервера и выполнения операции локально.

Это также означает, что есть лишь небольшое количество действий, которые вы не сможете выполнить, если вы находитесь офлайн или не имеете доступа к VPN в данный момент. Если вы в самолёте или в поезде и хотите немного поработать, вы сможете создавать коммиты без каких-либо проблем (в вашу локальную копию,

помните?): когда будет возможность подключиться к сети, все изменения можно будет синхронизировать. Если вы ушли домой и не можете подключиться через VPN, вы всё равно сможете работать. Добиться такого же поведения во многих других системах либо очень сложно, либо вовсе невозможно.

### **Git обычно только добавляет данные**

Когда вы производите какие-либо действия в Git, практически все из них только добавляют новые данные в базу Git. Очень сложно заставить систему удалить данные либо сделать что-то, что нельзя впоследствии отменить. Как и в любой другой СКВ, вы можете потерять или испортить свои изменения, пока они не зафиксированы, но после того, как вы зафиксируете снимок в Git, будет очень сложно что-либо потерять, особенно, если вы регулярно синхронизируете свою базу с другим репозиторием.

### **Три состояния**

Теперь внимательно! Это самая важная вещь, которую нужно запомнить о Git, если вы хотите, чтобы остаток процесса обучения прошёл гладко. У Git есть три основных состояния, в которых могут находиться ваши файлы: зафиксированное (committed), изменённое (modified) и подготовленное (staged).

- Зафиксированный значит, что файл уже сохранён в вашей локальной базе.
- К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы.
- Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит.

Мы подошли к трём основным секциям проекта Git: Git-директория (Git directory), рабочая директория (working directory) и область подготовленных файлов (staging area).

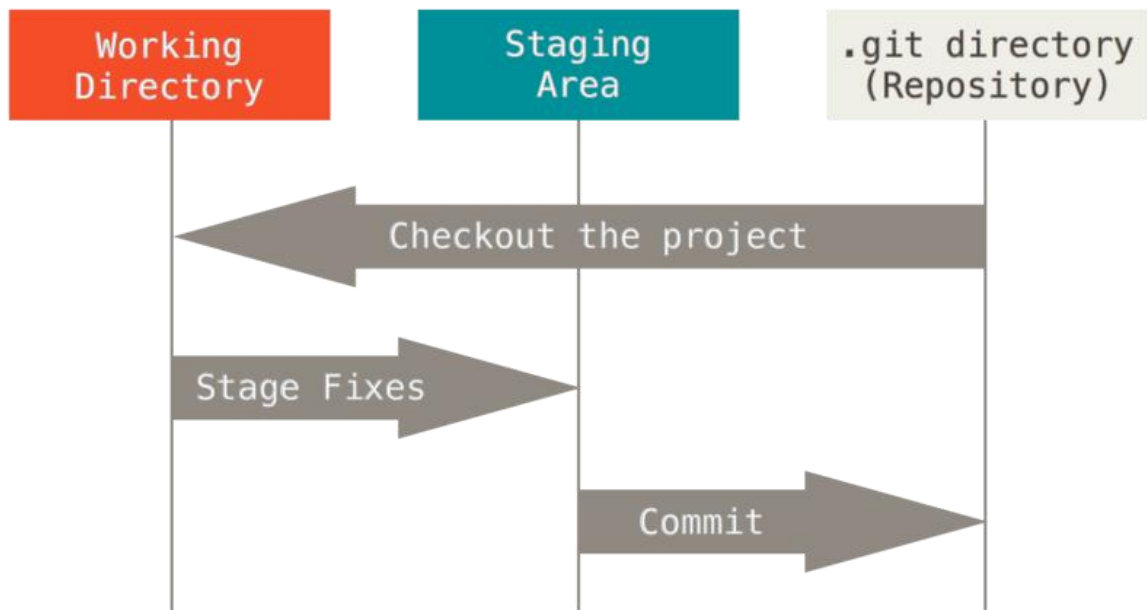


Рис. 181 – Рабочая директория, область подготовленных файлов и директория Git

Git-директория — это то место, где Git хранит метаданные и базу объектов вашего проекта. Это самая важная часть Git, и это та часть, которая копируется при клонировании репозитория с другого компьютера.

Рабочая директория является снимком версии проекта. Файлы распаковываются из сжатой базы данных в Git-директории и располагаются на диске, для того чтобы их можно было изменять и использовать.

Область подготовленных файлов — это файл, обычно располагающийся в вашей Git-директории, в нём содержится информация о том, какие изменения попадут в следующий коммит. Эту область ещё называют “индекс”, однако называть её stage-область также общепринято.

Базовый подход в работе с Git выглядит так:

1. Вы изменяете файлы в вашей рабочей директории.
2. Вы выборочно добавляете в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки **только** этих изменений в область подготовленных файлов.



3. Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в вашу Git-директорию.

Если определённая версия файла есть в Git-директории, эта версия считается **зафиксированной**. Если версия файла изменена и добавлена в индекс, значит, она **подготовлена**. И если файл был изменён с момента последнего распаковывания из репозитория, но не был добавлен в индекс, он считается **изменённым**.

Не волнуйтесь, если сейчас ничего непонятно. После практики вернитесь к этому разделу, чтобы заново понять принцип работы. Git используется в подавляющем большинстве софтверных компаний и навыки работы с ним будут полезными разработчика любого направления.

## 2. Совместная разработка – GitHub

GitHub это система управления проектами и версиями кода, а также платформа социальных сетей, созданная для разработчиков. Среди прочего, это позволяет вам работать совместно с другими людьми по всему миру, планировать свои проекты и отслеживать свою работу.

GitHub также является одним из крупнейших онлайн-хранилищ совместной работы по всему миру.

Если Git — это сердце GitHub, то Hub — это его душа. Концентратор в GitHub — это то, что превращает командную строку, такую как Git, в крупнейшую социальную сеть для разработчиков. Помимо участия в определённом проекте, GitHub позволяет пользователям общаться с единомышленниками. Вы можете следить за людьми и смотреть, что они делают или с кем они общаются.

### Репозиторий

Репозиторий или хранилище — это каталог, в котором хранятся файлы вашего проекта. Он может быть расположен в хранилище GitHub или в локальном хранилище на вашем компьютере. Вы можете хранить файлы кодов, изображения, аудио или всё, что связано с проектом, в хранилище.

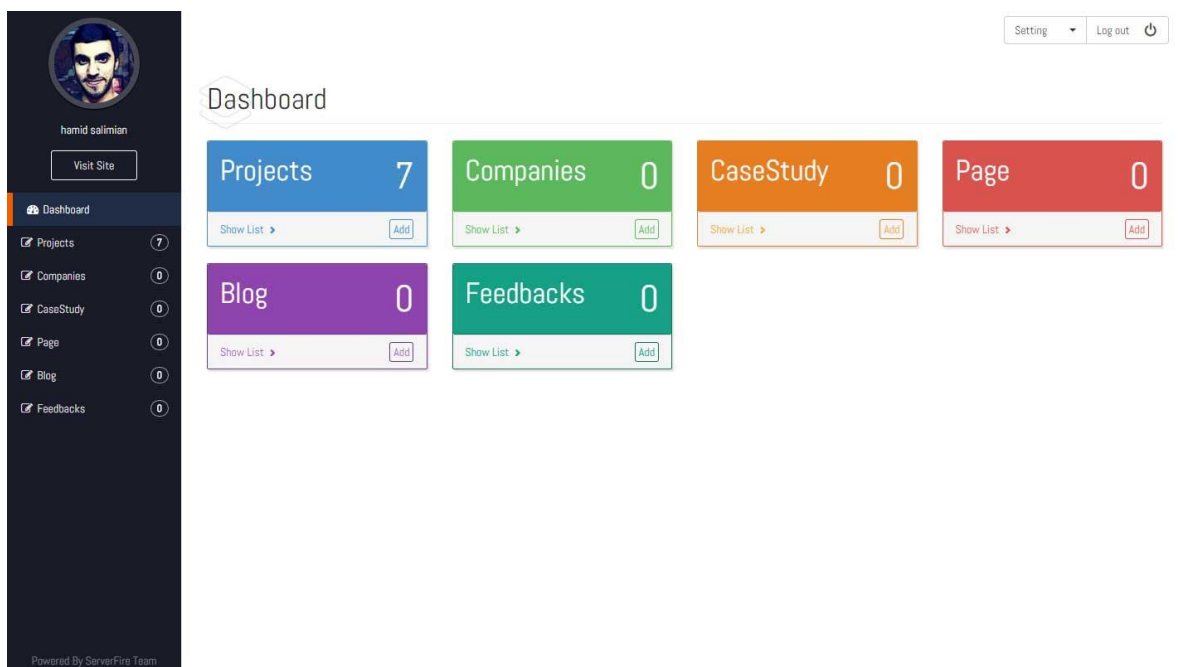


Рис. 182 – Панель управления проектами в GitHub

## Ветка

Ветка – это копия вашего репозитория. Вы можете использовать ветку, когда хотите сделать разработку изолированно. Работа с веткой не повлияет на центральное хранилище или другие ветки. Если вы сделали эту работу, вы можете объединить свою ветку с другими ветками и центральным репозиторием, используя запрос на извлечение.

## Запрос на извлечение

Запрос на извлечение означает, что вы сообщаете другим, что вы передали изменения, внесённые вами в ветке, в главный репозиторий. Соавторы хранилища могут принять или отклонить запрос на извлечение. После его открытия вы можете обсудить и проанализировать свою работу с соавторами.

## Форкинг репозитория

Форкинг репозитория означает, что вы создаёте новый проект на основе существующего репозитория. Говоря простым языком, разветвление репозитория означает, что вы копируете существующий репозиторий, вносите некоторые

необходимые изменения, сохраняете новую версию в качестве нового репозитория и называете это своим собственным проектом.

Это отличная функция, которая ускоряет разработку проекта. Поскольку это совершенно новый проект, центральное хранилище не будет затронуто. Если «главный» репозиторий обновлён, вы также можете применить это обновление к вашему текущему форку.

GitHub не ограничен только для разработчиков

GitHub это отличная платформа, которая меняет работу разработчиков. Тем не менее, каждый, кто хочет эффективно управлять своим проектом и работать совместно, также может узнать, что такое GitHub и как им пользоваться.

Если ваша команда работает над проектом, который нуждается в постоянных обновлениях и хочет отслеживать внесённые изменения, GitHub подходит для вас. Есть и другие альтернативы, такие как GitLab, BitBucket, но начнём знакомство с наиболее известной из них.

### **3. Хостинг с автоматическим обновлением кода**

Если вы заинтересуетесь веб-разработкой и будете читать материалы в тематических телеграм-каналах или листать статьи на хабре, то будете встречать понятие «деплой». Деплой - (англ. deploy) это развертывание – помещение исполняемого кода на сервер, где он будет работать. В более широком смысле, деплой подразумевает весь путь кода из среды разработки до потребителя. Часто он бывает сложный и тернистый. Веб-приложения могут быть очень большими (Вконтакте, Twitter, Tik-Tok, банкинг Сбера, Яндекс.Карты, Раурал и т.д.), а процесс деплоя очень сложным. Серверов может быть огромное множество, у пользователей должна отображаться одна версия приложения вне зависимости от страны входа и сервера, который эту страницу показывает. А также, серверы

должны уметь обрабатывать одинаково запросы, и отдавать верные данные приложению при изменении версии.

У GitHub есть сервис GitHub Pages, позволяющий публиковать ваши веб-проекты на основе репозитория. В этом случае деплой будет происходить автоматически при обновлении кода в репозитории. Но у этого сервиса есть несколько ограничений: первое и самое важное – вам необходимо использовать в качестве git-репозитория именно GitHub. Второе – нельзя масштабировать проект как на обычном хостинге.

Поэтому в лабораторной работе мы рассмотрим сервис Netlify. Это не совсем хостинг, поскольку он предоставляет больше автоматических решений, чем традиционный хостинг. Его задача – автоматизировать рутину настроек сервера для front-end разработчика.

Конечно, всё хорошее и удобное стоит денег, но у Netlify есть отличная опция бесплатного пользования. Для маленьких студенческих проектов – это то, что нужно. Ниже я перечислю в чём именно особенности этого сервиса:

- Рабочий процесс «все в одном», который сочетает в себе глобальное развертывание, непрерывную интеграцию и автоматический HTTPS.
- Единый упрощенный рабочий процесс.
- Вы получите все инструменты, необходимые для развертывания и управления сайтом.
- Вы сможете заменить свою хостинговую инфраструктуру, непрерывную интеграцию и конвейер развертывания одним рабочим процессом.
- Специализированная, многооблачная инфраструктура, которая рассчитана на скорость, автоматизированное масштабирование.

### **Возможности сервиса**

Автомное развертывание с мгновенной публикацией и откатами. Развертывания являются неизменными. То есть, каждое развертывание приводит к версии сайта, которая никогда не изменяется. Последующие обновления создают новые

экземпляры сайта для замены предыдущих версий (которые тихо благодарны за их обслуживание и выходят на пенсию, но не уничтожаются). Это означает, что вы можете вернуться к любой предыдущей сборке в любое время одним нажатием кнопки в консоли администратора или через API.

Уведомления и постоянные ссылки. Netlify позволяет настраивать уведомления на основе различных событий развертывания. Вы можете определить, кто получает уведомления о таких событиях, как начало развертывания или когда успешное развертывание завершается неудачей, блокируется или разблокируется.

Развертывания филиалов и субдомены. Netlify дает вам некоторый контроль над развертыванием. Вы можете выбрать, нужно ли развертывать только свою производственную ветвь, все ветви или несколько названных ветвей.

Тестирование A/B, многовариантное тестирование или отдельное тестирование. Разработчики называют A/B тестирование несколькими терминами, но Netlify называет его «разделенным тестированием», потому что он делает именно это: разбивает трафик на ваш сайт между любыми филиалами, которые вы указали. Вы можете разделить свой трафик на столько филиалов, сколько хотите, указав, какой процент трафика должен идти туда.

Команды контекстной сборки. Вы можете не только развернуть разные филиалы, но и настроить контент и среды в своих развертываниях в соответствии с различными контекстами, такими как постановка, тестирование и производство.

Управление SSL и бесплатный SSL от Let's Encrypt. Netlify упрощает настройку HTTPS для ваших доменов. У вас доступна опция управляемого SSL, пользовательского SSL и даже выделенного IP-протокола SSL для тех случаев, когда защищенный канал просто необходим.

Выполнение тестов с непрерывной интеграцией Netlify. Кроме того, в дополнение к специально созданному CDN для размещения ваших сайтов, Netlify

также предоставляет контейнерную среду для запуска ваших сборок. Это означает, что любая сборка, которую вы запускаете в локальной среде или на сервере непрерывной интеграции, может работать непосредственно в Netlify.

**Обработка формы.** Если вам нужна форма на вашем сайте, которая принимает контент, представленный вашими посетителями, Netlify может разместить это за вас. Просто добавьте атрибут в HTML-код вашей формы, Netlify создаст подходящую конечную точку для неё и сделает все данные, доступные вам через интерфейс администратора и API.

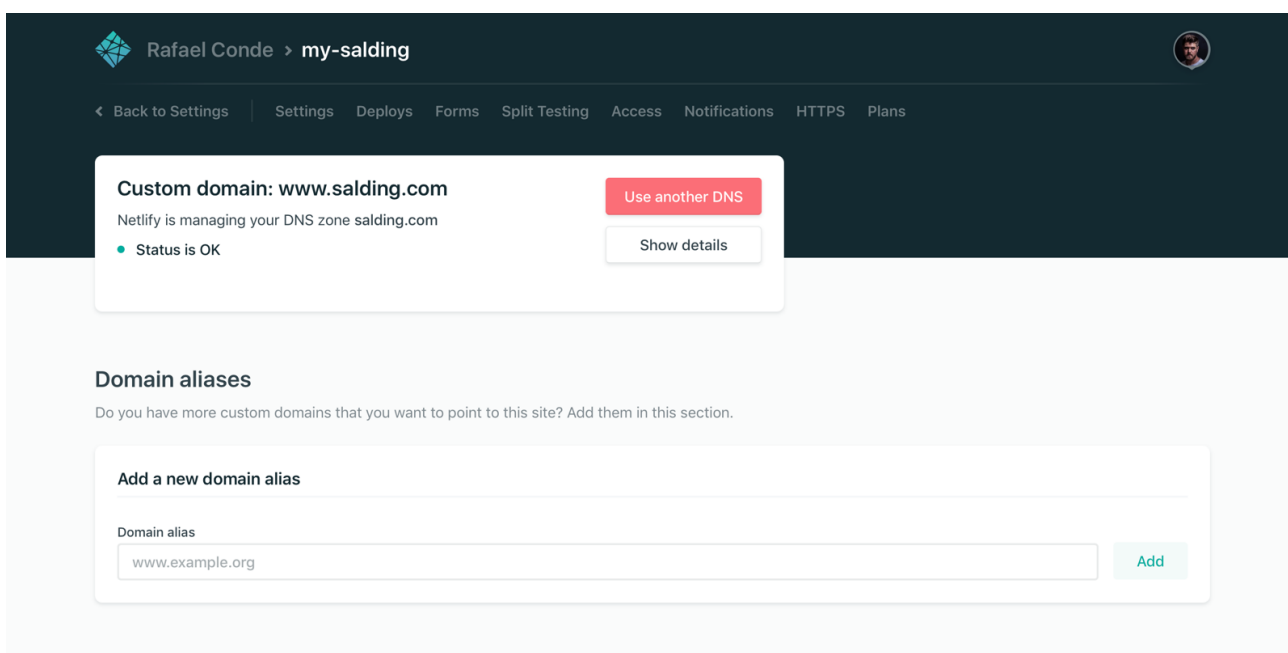
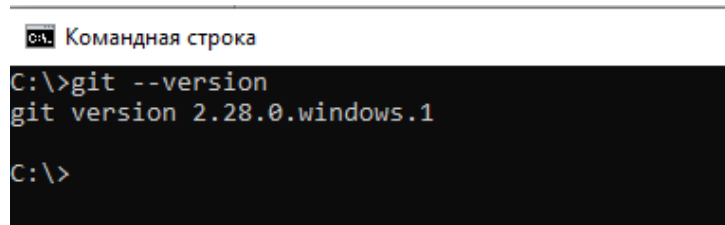


Рис. 183 – Панель управления проектов в Netlify

## Практическая часть

Начнём лабораторную работу с инициализации git в проекте. Для этого откроем командную строку. И введём команду `git --version`



```
C:\>git --version
git version 2.28.0.windows.1
C:\>
```

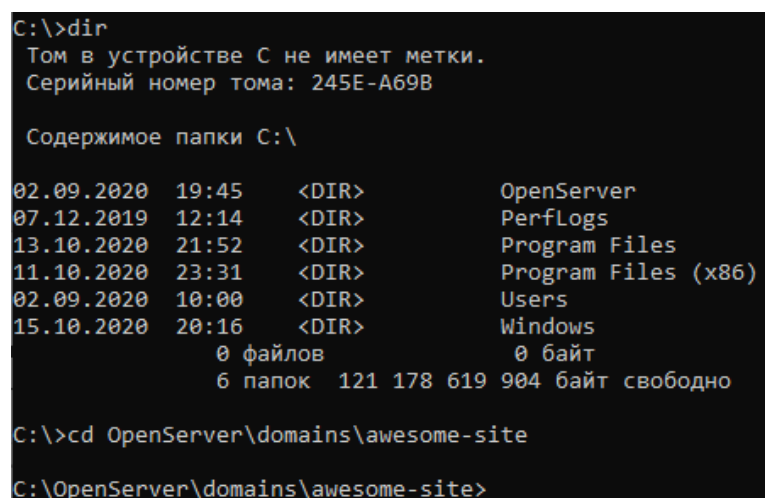
Рис. 184 – Консоль с проверкой установленного git

Если команда выполнена и отобразилась версия git – значит он уже установлен на компьютере. Если возникла ошибка "git" не является внутренней или внешней командой», значит нужно установить git, предварительно скачав его с официального сайта по адресу:

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

После установки, снова открываем консоль и проверяем, что команда выше выполняется.

Теперь необходимо инициализировать проект в git: перейдите в консоли к папке с проектом. Например, для пути `C:\OpenServer\domains\awesome-site\` команда будет выглядеть «`cd OpenServer\domains\awesome-site`»



```
C:\>dir
Том в устройстве C не имеет метки.
Серийный номер тома: 245E-A69B

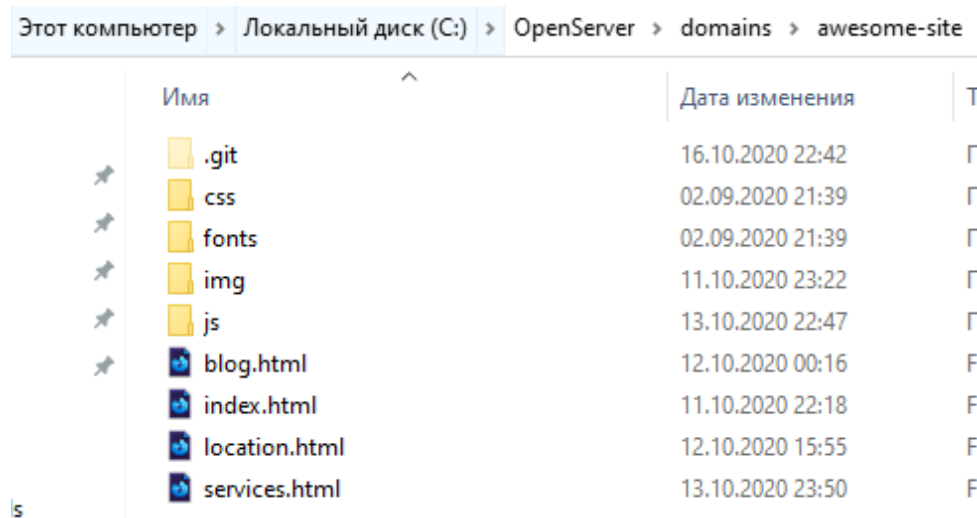
Содержимое папки C:\

02.09.2020  19:45    <DIR>        OpenServer
07.12.2019  12:14    <DIR>        PerfLogs
13.10.2020  21:52    <DIR>        Program Files
11.10.2020  23:31    <DIR>        Program Files (x86)
02.09.2020  10:00    <DIR>        Users
15.10.2020  20:16    <DIR>        Windows
              0 файлов              0 байт
              6 папок 121 178 619 904 байт свободно

C:\>cd OpenServer\domains\awesome-site
C:\OpenServer\domains\awesome-site>
```

Рис. 185 – Навигация в консоли

После чего необходимо ввести команду `git init`. Git создаст скрытую папку `.git` с информацией о версиях вашего проекта.



Имя	Дата изменения	Тип
.git	16.10.2020 22:42	Папка
css	02.09.2020 21:39	Папка
fonts	02.09.2020 21:39	Папка
img	11.10.2020 23:22	Папка
js	13.10.2020 22:47	Папка
blog.html	12.10.2020 00:16	Файл
index.html	11.10.2020 22:18	Файл
location.html	12.10.2020 15:55	Файл
services.html	13.10.2020 23:50	Файл

Рис. 186 – Папка `.git` в корне проекта

Если мы сейчас введём в консоли команду `git status`, то получим информацию о текущем состоянии нашего проекта. Гит сообщил нам, что мы находимся на ветке `master`, никаких файлов не добавлено, ничего коммитить не нужно.

```
C:\OpenServer\domains\awesome-site>git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    blog.html
    css/
    fonts/
    img/
    index.html
    js/
    location.html
    services.html

nothing added to commit but untracked files present (use "git add" to track)
C:\OpenServer\domains\awesome-site>
```

Рис. 187 – Отчёт команды `git status`



А если мы хотим добавить к гиту файлы, нужно ввести команду `git add`. Так и поступим. Нам необходимо добавить все файлы проекта. поэтому введём в качестве аргумента «точку». Это значит добавить все файлы каталога.

При повторном вводе команды `git status` видно, что все файлы проекты были добавлены. В качестве подсказки они будут подкрашены зелёным цветом. Теперь необходимо зафиксировать изменения. Вводим команду `git commit -m` “Первый коммит инициализации” (в кавычках может быть любой комментарий, который объясняет, что было сделано в этой итерации изменений).

```
new file:   location.html
new file:   services.html

C:\OpenServer\domains\awesome-site>git commit -m "Первый коммит в проекте"
[master (root-commit) cef3cae] Первый коммит в проекте
56 files changed, 1337 insertions(+)
create mode 100644 blog.html
create mode 100644 css/fonts.css
create mode 100644 css/style.css
create mode 100644 fonts/OpenSans-Light.ttf
create mode 100644 fonts/OpenSans-Regular.ttf
create mode 100644 fonts/OpenSans-Semibold.ttf
create mode 100644 img/01.jpg
create mode 100644 img/blog/aaron-burden-ufFIweqSPd4-unsplash.jpg
create mode 100644 img/blog/blog00.jpg
create mode 100644 img/blog/blog01.jpg
create mode 100644 img/blog/blog02.jpg
create mode 100644 img/blog/blog03.jpg
create mode 100644 img/blog/blog04.jpg
create mode 100644 img/blog/blog05.jpg
create mode 100644 img/blog/blog06.jpg
create mode 100644 img/blog/blog07.jpg
```

Рис. 188 – Первый коммит

Отлично, теперь наши изменения нужно хранить в облаке с доступом других участников. Как уже писалось выше, будем использовать GitHub.

Для начала необходимо пройти процесс регистрации по адресу <https://github.com/> и заполняем поля формы Sign Up.

Why GitHub? Team Enterprise Explore Marketplace Pricing Search GitHub Sign

# Built for developers

GitHub is a development platform inspired by the way you work. From **open source** to **business**, you can host and review code, manage projects, and build software alongside 50 million developers.

Username: testemail7 ✓

Email: test-email-7@yandex.ru ✓

Password: ..... ✓

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

**Sign up for GitHub**

By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.

Рис. 189 – Форма регистрации GitHub

Жмём Sign up for GitHub, проходим проверку против ботов и жмём Join a Free Plan. Дальше гитхаб просит заполнить небольшую анкету. Тут прожимаем как есть: Student, I'm new to programming и первый ряд ответов на третий вопрос. Но в целом, совершенно не важно какие варианты вы выберете, это лишь поможет алгоритмам GitHub лучше подстроится под вас.

What kind of work do you do, mainly?

Software Engineer I write code	Student I go to school
Product Manager I write specs	UX & Design I draw interfaces
Data & Analytics I write queries	Marketing & Sales I look at charts
Teacher I educate people	Other I do my own thing

How much programming experience do you have?

None I don't program at all	A little I'm new to programming
A moderate amount I'm somewhat experienced	A lot I'm very experienced

What do you plan to use GitHub for?  
(Select up to 3)

--	--	--

Рис. 190 – Анкета нового пользователя

После чего будет выслано письмо для завершения регистрации. Откройте свой ящик и перейдите по ссылке в письме. Откроется новая страница GitHub, её можно пропустить, а можно и воспользоваться быстрыми командами из неё. Выберем первый раздел «Start a new project» с кнопкой «Create a repository», ведь именно для этого мы здесь, нам нужен репозиторий для нашего нового проекта.

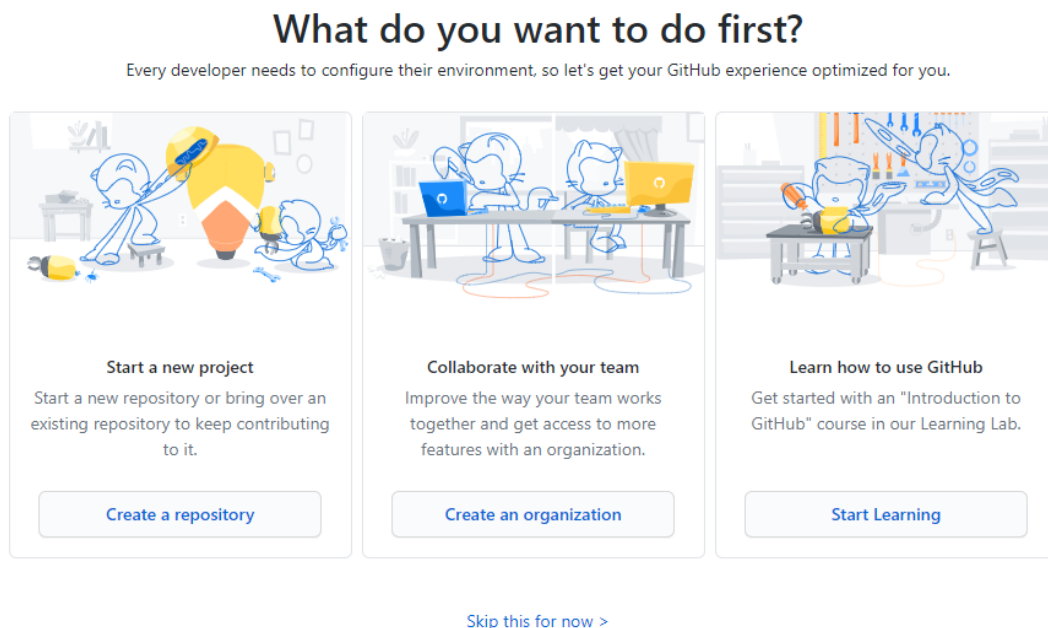



Рис. 191 – Окно приветствия и быстрого старта

На новой странице заполняем поля нашего проекта. В поле “Repository name” вносим название нашего проекта. Можем заполнить ниже поле описания, где в нескольких словах можно описать о чём этот проект (это поле, как и многие другие поддерживают эмодзи, так что можно их писать прямо здесь, к слову, для быстрого доступа к панели выбора эмодзи в Windows 10/11 используйте комбинацию «Win + .»)), а доступ оставляем Public. Ниже галочки нигде не ставим и жмём кнопку «Create repository».

## Create a new repository

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

Owner \*

 testemail7 ▾

Repository name \*

/ awesome-site ✓

Great repository names are short and memorable. Need inspiration? How about [psychic-pancake?](#)

Description (optional)

This is my very first project and i am so excited! 😊



**Public**

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



**Private**

You choose who can see and commit to this repository.

**Initialize this repository with:**

Skip this step if you're importing an existing repository.

☐ **Add a README file**

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

☐ **Add .gitignore**

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

☐ **Choose a license**

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

**Create repository**

Рис. 192 – Создание репозитория

На следующей странице, репозиторий создан, а GitHub вежливо предлагает подсказки, как дальше действовать. Так и поступим. Скопируйте ссылку как на рисунке под указателем 1. Мы будем действовать примерно, как указано на рисунке под указателем 2.

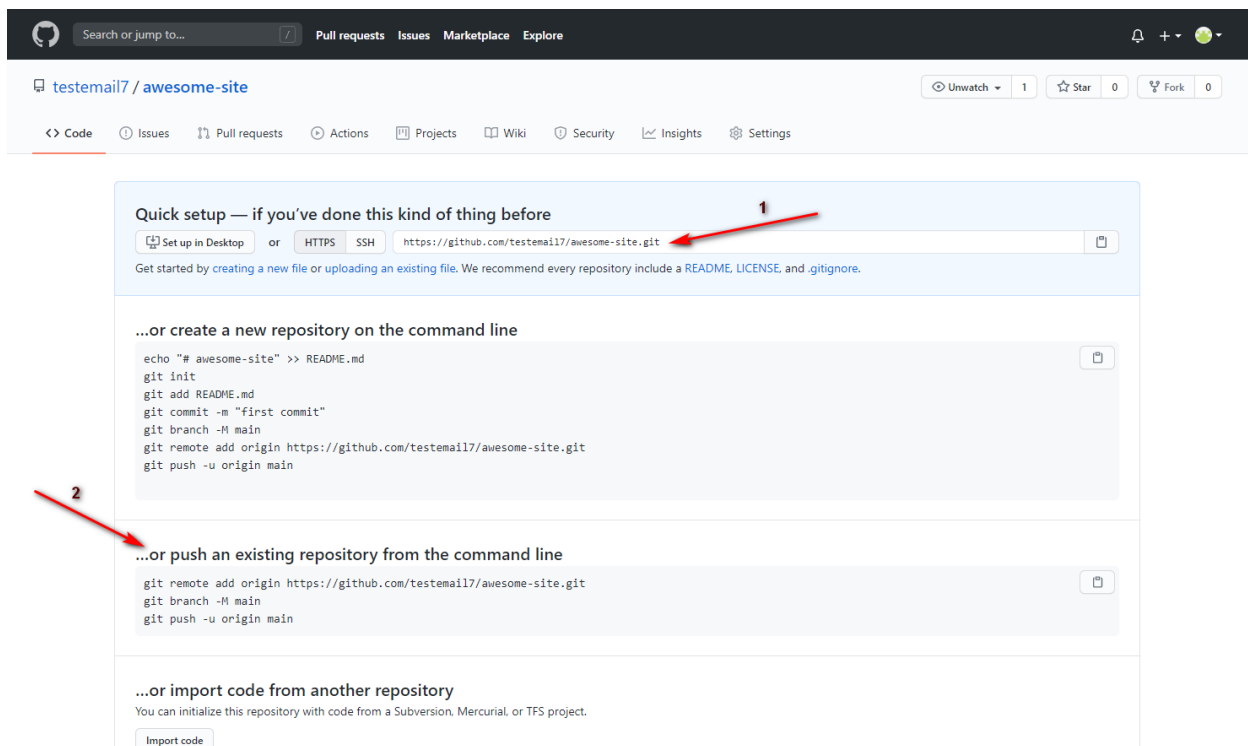


Рис. 193 – Окно помощи после создания репозитория

Вернёмся к консоли и внесём сначала команду `git remote add origin` и ссылку на ваш репозиторий, которую вы скопировали в GitHub. А затем команду `git push -u origin master`. После второй команды скорее всего появится модальное окно, как на скрине. Всё хорошо, просто GitHub хочет знать, кто ему пытается переслать данные. Здесь вводите свои данные от вашей учётной записи GitHub.

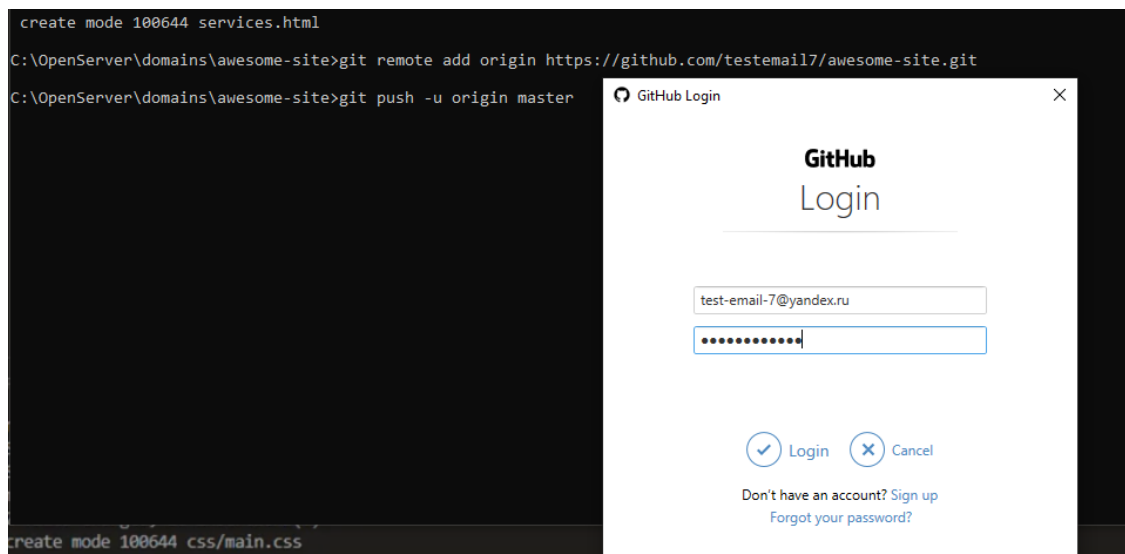


Рис. 194 – Авторизация push запроса к GitHub

После ввода своих данных проект будет отправлен на удалённый репозиторий. Если всё прошло успешно, консоль должна выглядеть так:

```
C:\OpenServer\domains\awesome-site>git push -u origin master
Enumerating objects: 65, done.
Counting objects: 100% (65/65), done.
Delta compression using up to 4 threads
Compressing objects: 100% (64/64), done.
Writing objects: 100% (65/65), 23.92 MiB | 3.16 MiB/s, done.
Total 65 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), done.
To https://github.com/testemail7/awesome-site.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
C:\OpenServer\domains\awesome-site>
```

Рис. 195 – Завершение команды git push

Следующим шагом вернёмся на страницу GitHub. Если вы не закрывали вкладку, обновите страницу. Теперь вы увидите, что все файлы вашего проекта уже загружены, а напротив файла стоит пометка коммита – тот комментарий, который оставляли, когда выполняли команду git commit. Другие участники теперь по ним смогут быстро ориентироваться, что было изменено последний раз в файлах. Поэтому не стоит пренебрегать этой удобным свойством.

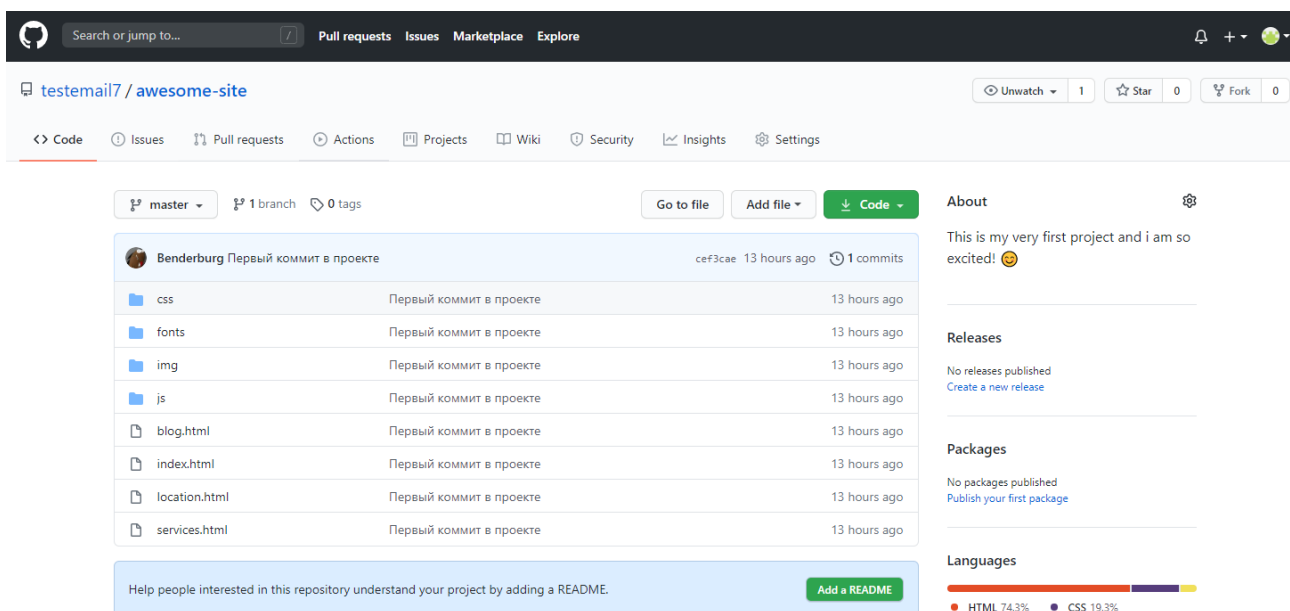


Рис. 196 – Просмотр проекта в удалённом репозитории

Найдите и кликните ссылку «1 branch» – здесь отображаются ветки проекта. Вы уже писали `git push origin master`, где `master` название главной ветки проекта. Также здесь можно увидеть коммиты в репозиторий

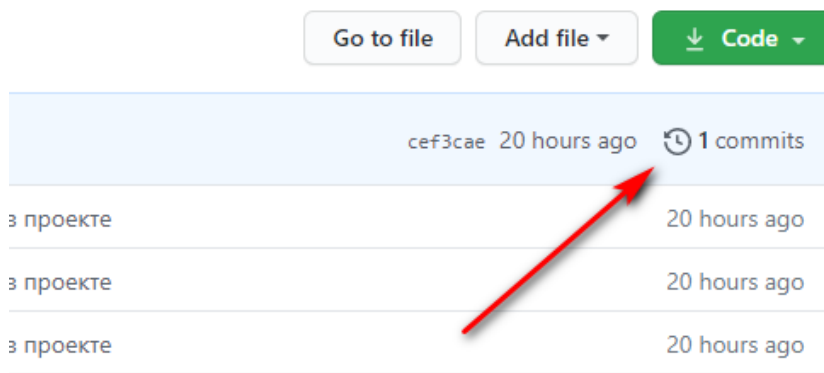


Рис. 197 – Ссылка на отчёт по коммитам

Нажмите, чтобы перейти на страницу со списком коммитов. Нажмите на последний коммит и на этой странице отобразятся все изменения, которые были внесены последний раз – то есть был добавлен весь проект.

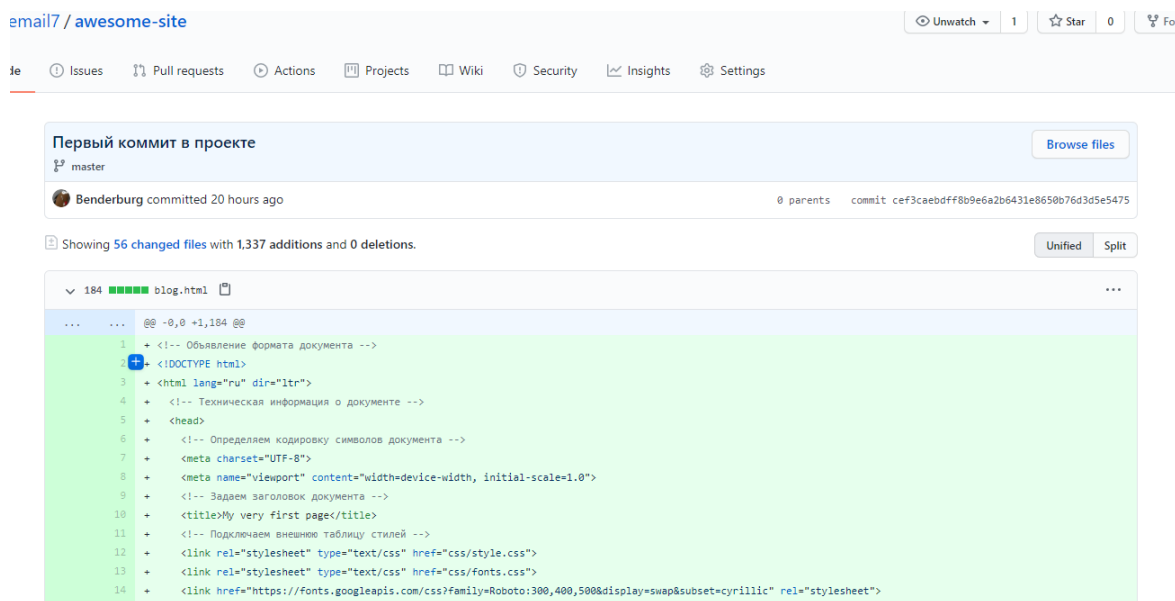


Рис. 198 – Просмотр отчёта изменений

Отлично, мы готовы опубликовать проект в сети. Для этого необходимо зарегистрировать аккаунт в [Netlify.com](https://www.netlify.com/). К счастью, здесь можно зарегистрироваться нашим аккаунтом GitHub, это сэкономит несколько кликов, в сравнении

с традиционной регистрацией через почтовый ящик. Откройте ссылку <https://app.netlify.com/signup> и нажмите кнопку с иконкой GitHub

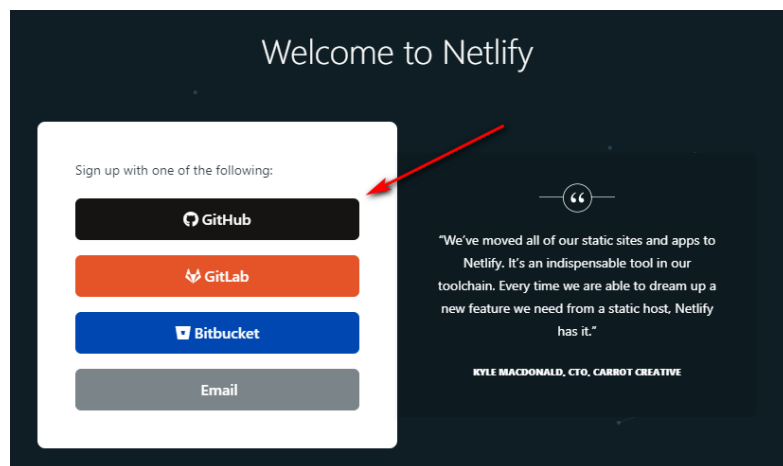


Рис. 199 – Форма регистрации в сервисе Netlify

После чего GitHub попросит подтвердить, что вы доверяете доступ к своей учётной записи – подтверждаем

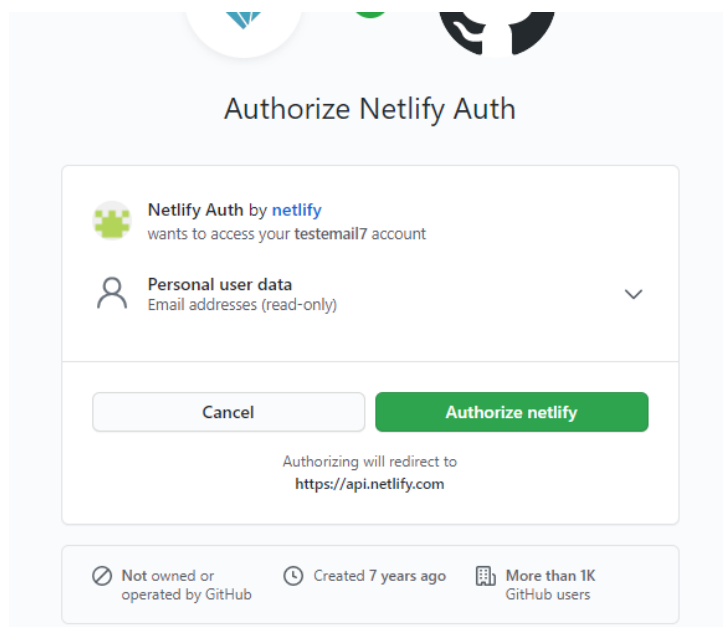


Рис. 200 – Подтверждение регистрации через учетную запись GitHub

Приветствие от Netlify подсказывает, что мы можем легко делать деплой после пуша. Что такое деплой, было описано в теории к лабораторной работе.



«Пуш» мы уже сделали, когда добавили проект в репозиторий. Создаём сайт из репозитория: жмём кнопку “New site from Git”

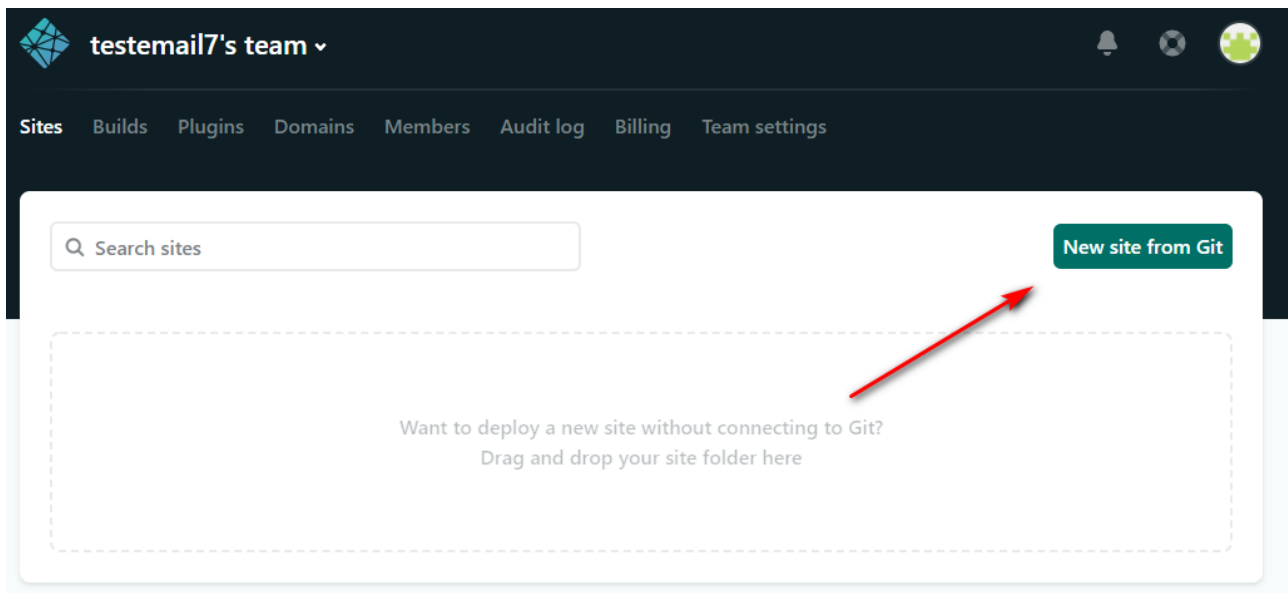


Рис. 201 – Создание сайта в Netlify

На следующей странице выбираем репозиторий, который хотим подключить. Выбираем GitHub, после чего в модальном окне GitHub снова попросит подтвердить доступ к нашим репозиториям: подтверждаем.

## Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider

2. Pick a repository

3. Build options, and deploy!

### Continuous Deployment

Choose the Git provider where your site's source code is hosted. When you push to Git, we run your build tool of choice on our servers and deploy the result.

You can [unlock options for self-hosted GitHub/GitLab](#) by upgrading to the Business plan.



Рис. 202 – Подключение проекта через репозиторий

Затем нужно выбрать к каким именно репозитория GitHub мы дадим доступ: оставляйте опцию All repositories (потом эту опцию можно сменить).

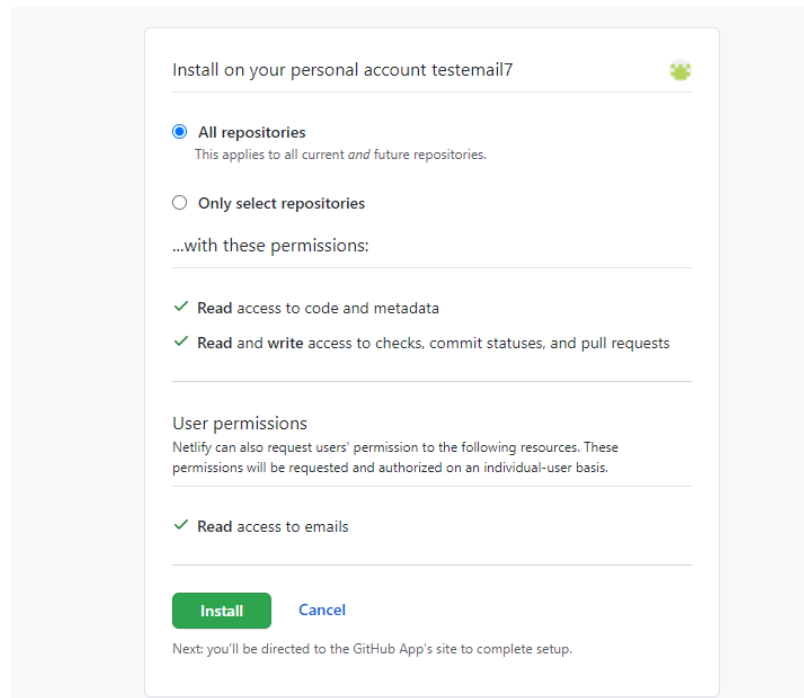


Рис. 203 – Настройки доступа к репозиториям GitHub

Жмём Install и вводим пароль. Netlify проведёт поиск существующих проектов в репозитории и предложит список для выбора, на основании какого проекта развернуть сайт. Выберите репозиторий с вашим проектом.

## Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider

2. Pick a repository

3. Build options, and deploy!

## Continuous Deployment: GitHub App

Choose the repository you want to link to your site on Netlify. When you push to Git, we run your build tool of choice on our servers and deploy the result.

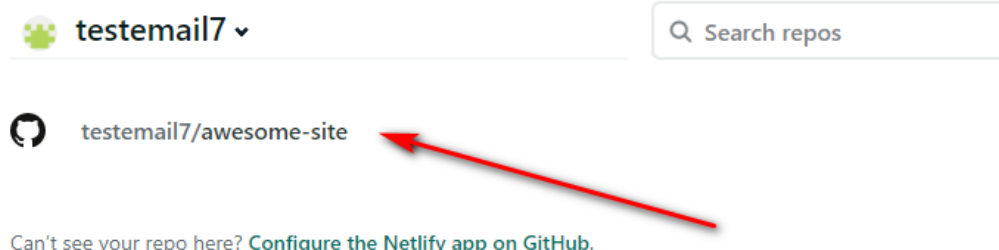


Рис. 204 – Навигация по списку имеющихся репозиториям

На последнем шаге никаких настроек менять не нужно и просто жмём кнопку «Deploy site».

После появления страницы как на рисунке ниже ваш сайт будет опубликован. Здесь Netlify подсказывает, что пункты 1 и 3 уже выполнены, сайт опубликован и защищён через HTTPS соединение. Сервис уже сгенерировал случайный домен, и мы можем перейти по нему, чтобы проверить работу нашего сайта (пункт 1 рисунка). Кроме того, мы можем поменять сгенерированную ссылку на свою, более читаемую (пункт 2 рисунка). Это бесплатно в поддомене netlify.app.

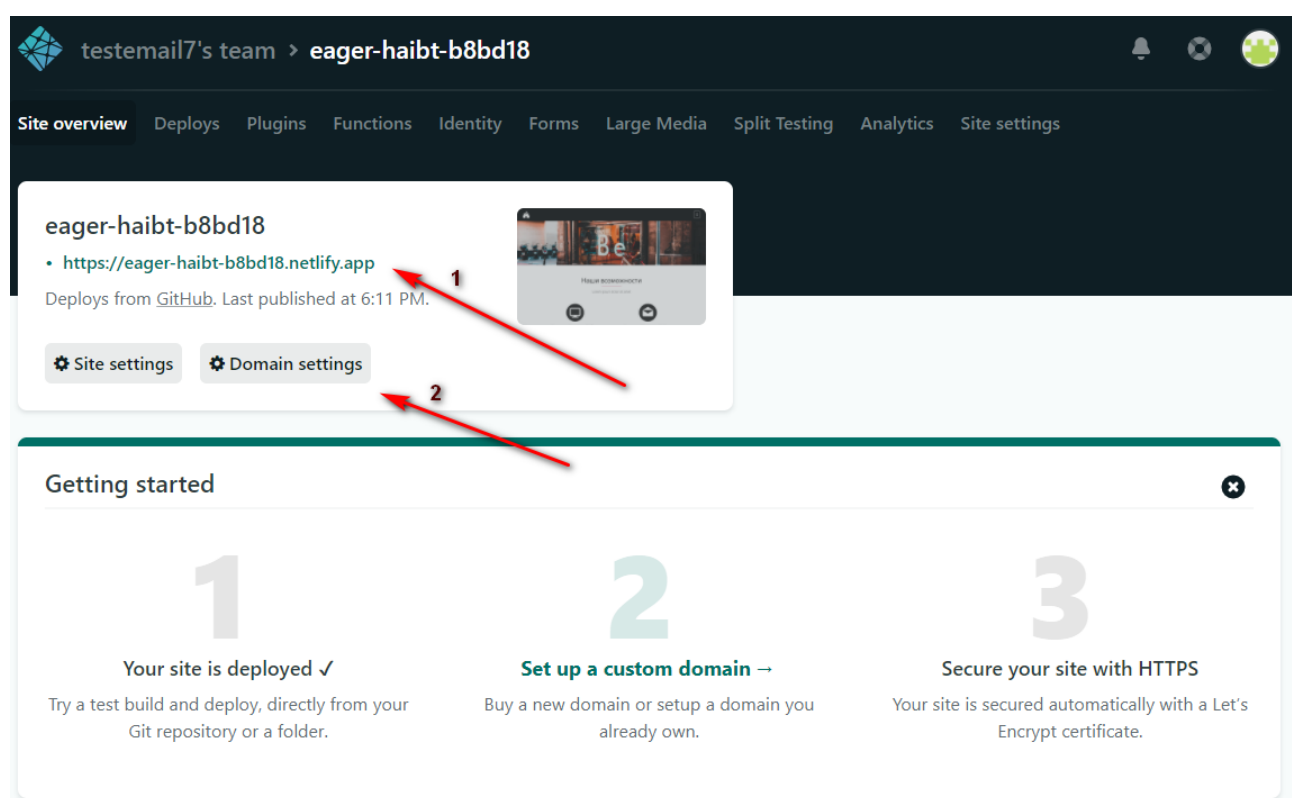


Рис. 205 – Настройка домена для сайта

В настройках сайта выберите Options -> Edit site name и укажите какое-то уникальное название.

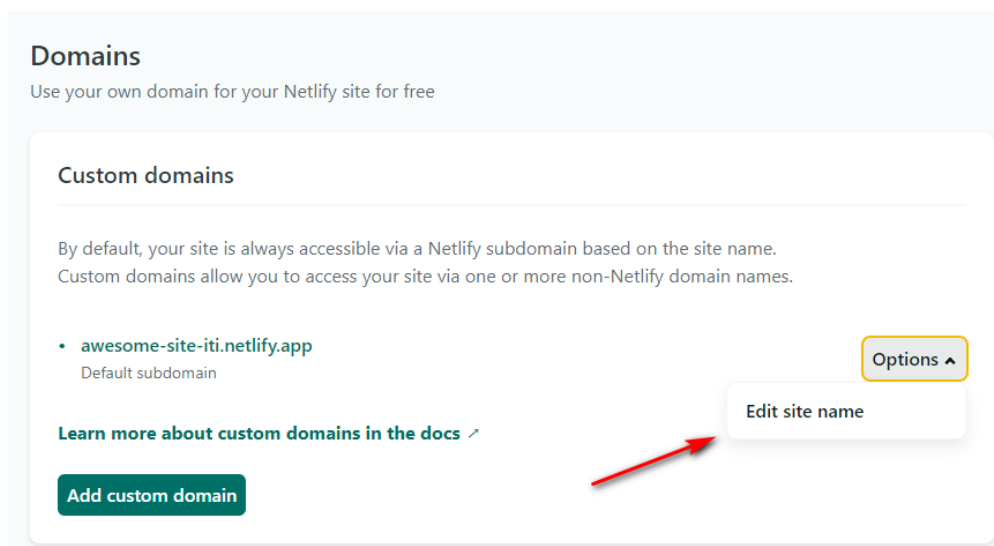


Рис. 206 – Установка личного доменного имени

Теперь можно скинуть ссылку друзьям и показать свою страничку. А ещё можно посмотреть его в живую на смартфоне и проверить, хорошо ли работает мобильная вёрстка.

Давайте внесём в проект изменения, чтобы посмотреть, как процесс модификации сайта происходит в связке Git – GitHub – Netlify.

Для начала, снова откроем консоль в нашем проекте и создадим рабочую ветку в git. Вводим команду `git checkout -b first-branch`

```
C:\OpenServer\domains\awesome-site>git checkout -b first-branch
Switched to a new branch 'first-branch'

C:\OpenServer\domains\awesome-site>git status
On branch first-branch
nothing to commit, working tree clean

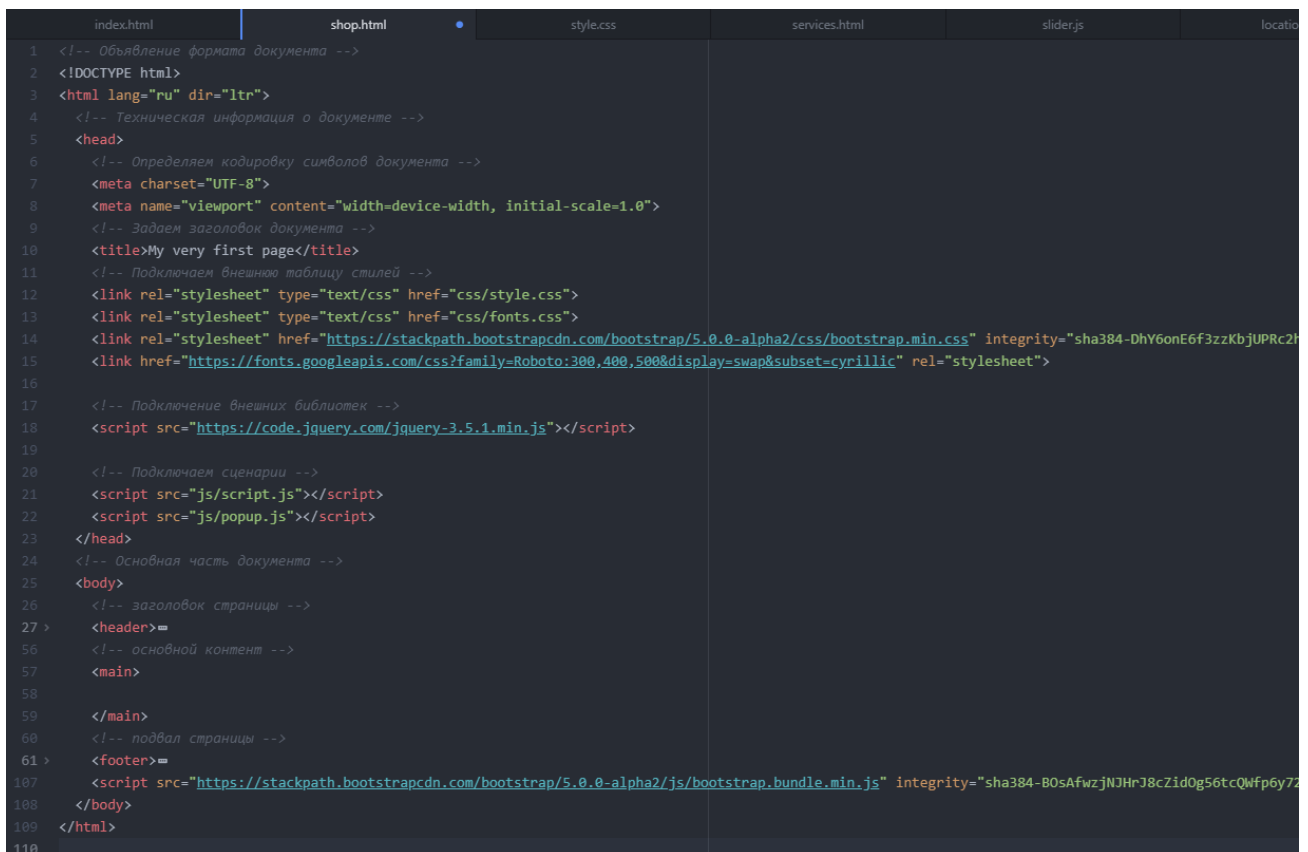
C:\OpenServer\domains\awesome-site>
```

Рис. 207 – Создание и переключение на новую ветку

Здесь `checkout` – переносит рабочее пространство в другую ветку, ключ `-b` означает, что перед переключением ветку необходимо создать, после ключа – название ветки, которую необходимо создать.

По команде `git-status` мы видим, что теперь мы находимся не на ветке `master`, где зафиксировано состояние нашего сайта, а на другой ветке, которую будем менять. Пока же это точная копия `master`.

Теперь создадим новую страницу `shop.html`. Для неё нам понадобится библиотека `Bootstrap`. Подключаем её, как и раньше, либо ссылками на официальный репозиторий, либо добавлением в проект. Внутри новой страницы копируем основные элементы: меню, обложка, футер и тег `<main>`.



```
1 <!-- Объявление формата документа -->
2 <!DOCTYPE html>
3 <html lang="ru" dir="ltr">
4   <!-- Техническая информация о документе -->
5   <head>
6     <!-- Определяем кодировку символов документа -->
7     <meta charset="UTF-8">
8     <meta name="viewport" content="width=device-width, initial-scale=1.0">
9     <!-- Задаем заголовок документа -->
10    <title>My very first page</title>
11    <!-- Подключаем внешнюю таблицу стилей -->
12    <link rel="stylesheet" type="text/css" href="css/style.css">
13    <link rel="stylesheet" type="text/css" href="css/fonts.css">
14    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/5.0.0-alpha2/css/bootstrap.min.css" integrity="sha384-DhY6onE6f3zzKbJUPRC2H"
15    <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500&display=swap&subset=cyrillic" rel="stylesheet">
16
17    <!-- Подключение внешних библиотек -->
18    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
19
20    <!-- Подключаем сценарии -->
21    <script src="js/script.js"></script>
22    <script src="js/popup.js"></script>
23  </head>
24  <!-- Основная часть документа -->
25  <body>
26    <!-- заголовок страницы -->
27    <header>=
56    <!-- основной контент -->
57    <main>
58
59    </main>
60    <!-- подвал страницы -->
61    <footer>=
107    <script src="https://stackpath.bootstrapcdn.com/bootstrap/5.0.0-alpha2/js/bootstrap.bundle.min.js" integrity="sha384-B0sAfWzjNjHrJ8cZid0g56tcQWfp6y72"
108  </body>
109 </html>
110
```

Рис. 208 – Разметка новой страницы `shop.html`

Не забудьте добавить ссылку на страницу в меню на всех страницах сайта.

Раз наша страница называется магазин, будет логичным добавить здесь раздел с товарами в виде карточек. Но прежде начнём с заголовка страницы под слайдером. Для этого создадим секцию с классом «`container`». Это стандартный

класс Bootstrap с отступами слева и справа. Это позволяет сохранять единообразие внешнего вида на всех страницах, сохраняя одинаковую ширину контейнеров. Подробнее о контейнерах вы можете почитать в документации Bootstrap по адресу <https://getbootstrap.com/docs/5.0/layout/containers/>

```
88 <section class="py-5 text-center container">
89
90 </section>
```

Рис. 209 – Секция в разметке Bootstrap

Также зададим ему классы «py-5» и «text-center». По «text-center» сразу можно догадаться, что у него стиль `text-align: center`. А вот «py» — это сокращение. Здесь `p` — это `padding`, а `y` — это ось вертикальная ось `Y`. То есть этот класс создаёт внутренние поля сверху и снизу: `padding-top` и `padding-bottom`. Соответственно класс вида `px-` будет создавать поля по горизонтальной оси `X` — `padding-left` и `padding-right`. Такой же принцип у внешних отступов `margin`: `my-` и `mx-`.

Если ось не задана — это значит, что стиль общий для всех сторон. Что касается числа, то числа задаются в интервале от 0 до 5, где 0 — это 0px, а 5 — это 3rem. Единица `rem` — условная единица величины, зависящая от размера шрифта для заданного элемента.

Подробнее на странице <https://getbootstrap.com/docs/5.0/utilities/spacing/>

Добавим следующую разметку:

```
89 <div class="row py-lg-5">
90 <div class="col-lg-6 col-md-8 mx-auto">
91   <h1 class="font-weight-light">Магазин</h1>
92   <p class="lead text-muted">Здесь будут карточки товаров нашего
93     магазина. У них будут фото, название, описание и цена.</p>
94   <p>
95     <a href="#" class="btn btn-primary my-2">Main call to action</a>
96     <a href="#" class="btn btn-secondary my-2">Secondary action</a>
97   </p>
98 </div>
99 </div>
```

Рис. 210 – Разметка заголовка витрины товаров

Здесь «row» также стандартный класс строки разметки. В строках контент разбивается по столбцам различной ширины. Строка имеет условную ширину в 12 колонок. И каждый блок может иметь ширину от 1, до 12 единиц, где 12 будет означать ширину всей строки. Это необходимо, чтобы автоматизировать процесс сохранения адаптивности на всех диагоналях экранов. Рассмотрим это на примере со столбцами карточек товаров. Про принципы разметки подробнее по адресу: <https://getbootstrap.com/docs/5.0/layout/breakpoints/>

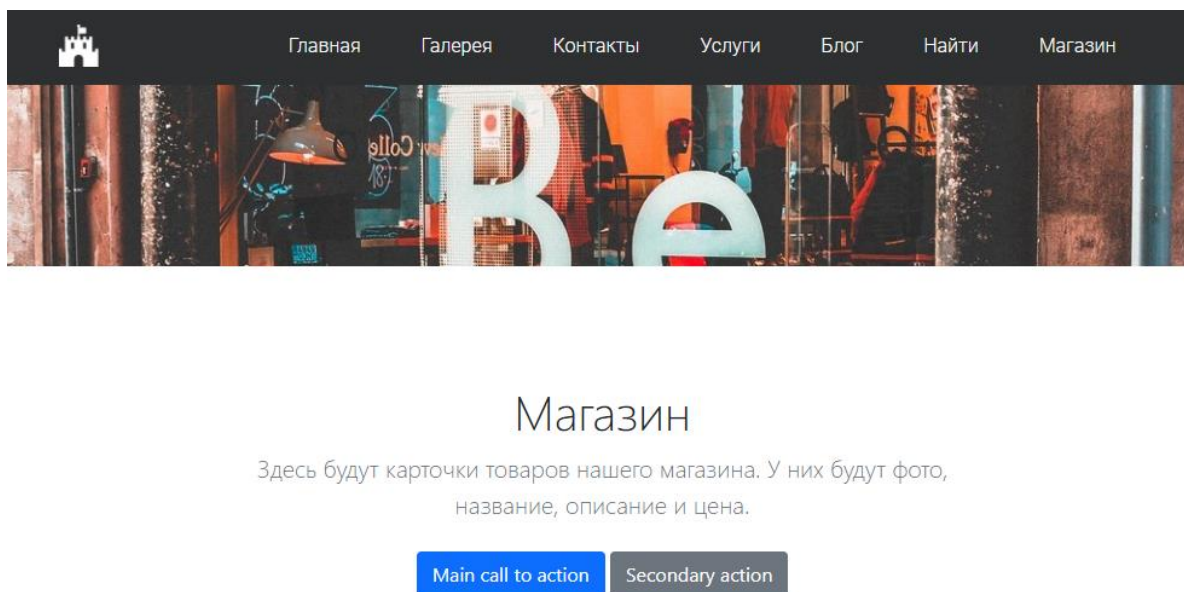


Рис. 211 – Заголовок витрины товаров страницы «Магазин»

Создаём ниже ещё одну секцию. Обратите внимание, что контейнер уже задан для блока внутри секции. Это необходимо, чтобы задать фон для секции по всей ширине страницы. А контейнер, как мы уже разобрали выше, имеет фиксированную ширину.

```
101     <section class="album py-5 bg-light">
102         <div class="container">
103
104         |
105         </div>
106     </section>
```

Рис. 212 – Секция витрины товаров

Внутри контейнера создаём «строку» - блок с классом «row». Кроме того добавим ещё классы как на рисунке.

```
<div class="row row-cols-1 row-cols-sm-2 row-cols-md-3 g-3">

</div>
```

Рис. 213 – Строка управления разметкой столбцов

Остальные классы размечают, на каких диагоналях какое количество столбцов будет отображаться. По умолчанию, будет 1 столбик – row-cols-1. Если ширина sm (768px или больше) — 2 столбика, при md (992px и больше) — 3 столбика.

Теперь добавим карточку товара с картинкой. Разметка показана на рисунке. Картинки для магазина в приложении.

```
104 <div class="row row-cols-1 row-cols-sm-2 row-cols-md-3 g-3">
105   <div class="col">
106     <div class="card shadow-sm">
107       
108       <div class="card-body">
109         <p class="card-text">
110           Это описание товара в несколько строк, чтобы показать как
111           отображается текст в карточке.
112         </p>
113         <div class="d-flex justify-content-between align-items-center">
114           <div class="btn-group">
115             <button type="button" class="btn btn-sm btn-outline-primary">Buy</button>
116             <button type="button" class="btn btn-sm btn-outline-secondary">View</button>
117           </div>
118           <small class="text-muted">15 $</small>
119         </div>
120       </div>
121     </div>
122   </div>
```

Рис. 214 – Разметка карточки товара

И снова не было добавлено ни одной строки CSS, но в результате получилась аккуратная карточка товара с кнопками, обладающими эффектами. К тому же страница сохраняет адаптивность.



Main call to action

Secondary action

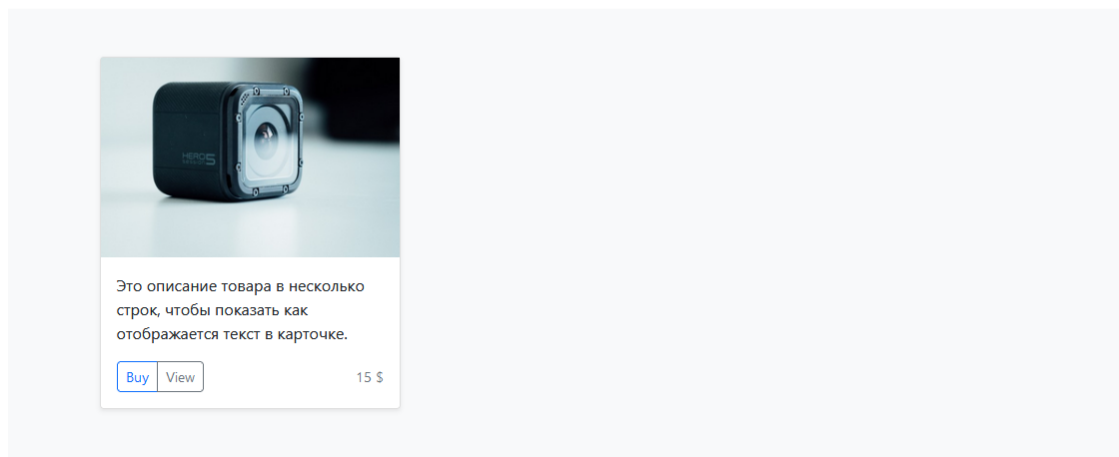


Рис. 215 – Внешний вид карточки товара

Добавьте ещё 8 таких блоков, поменяв путь к картинке, цену и текст. В результате получилась вот такая витрина товаров.

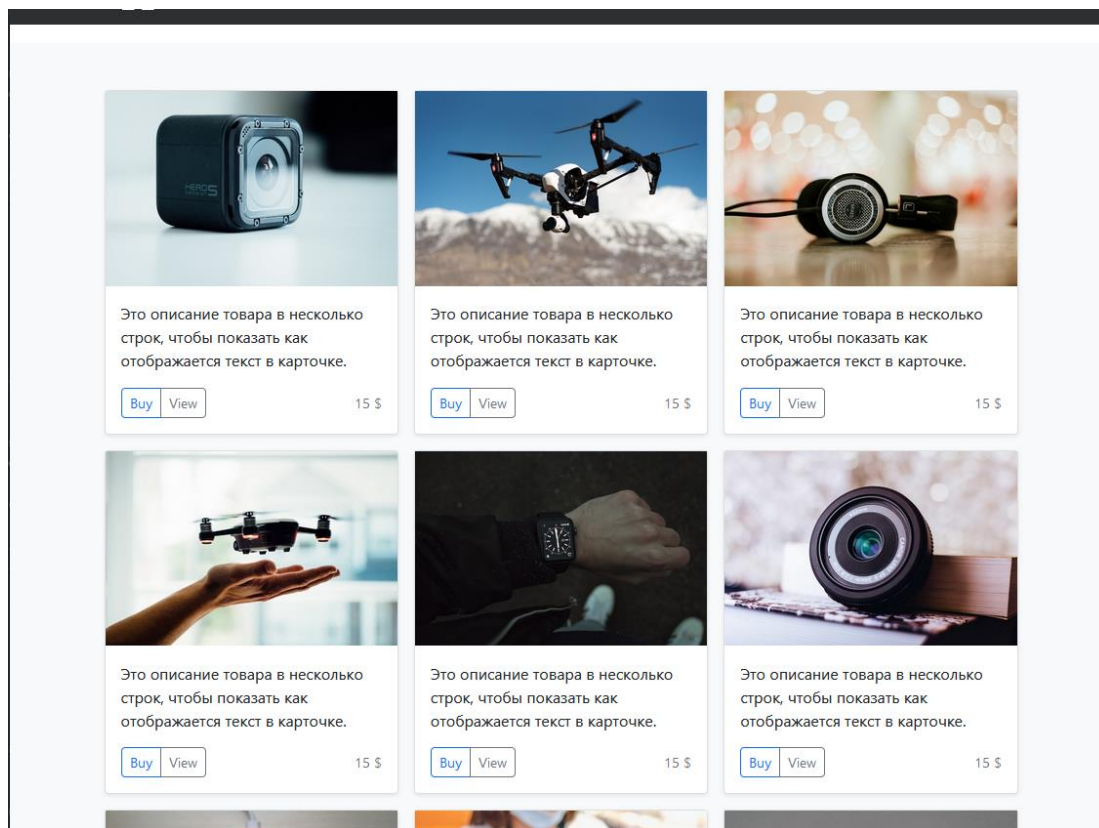


Рис. 216 – Заполненная витрина

Осталось только добавить эти изменения на наш сайт в сети. Приступим. Как вы уже могли заметить, в среде разработки файлы стали подкрашиваться другими цветами. А на полях строк кода подсветка.

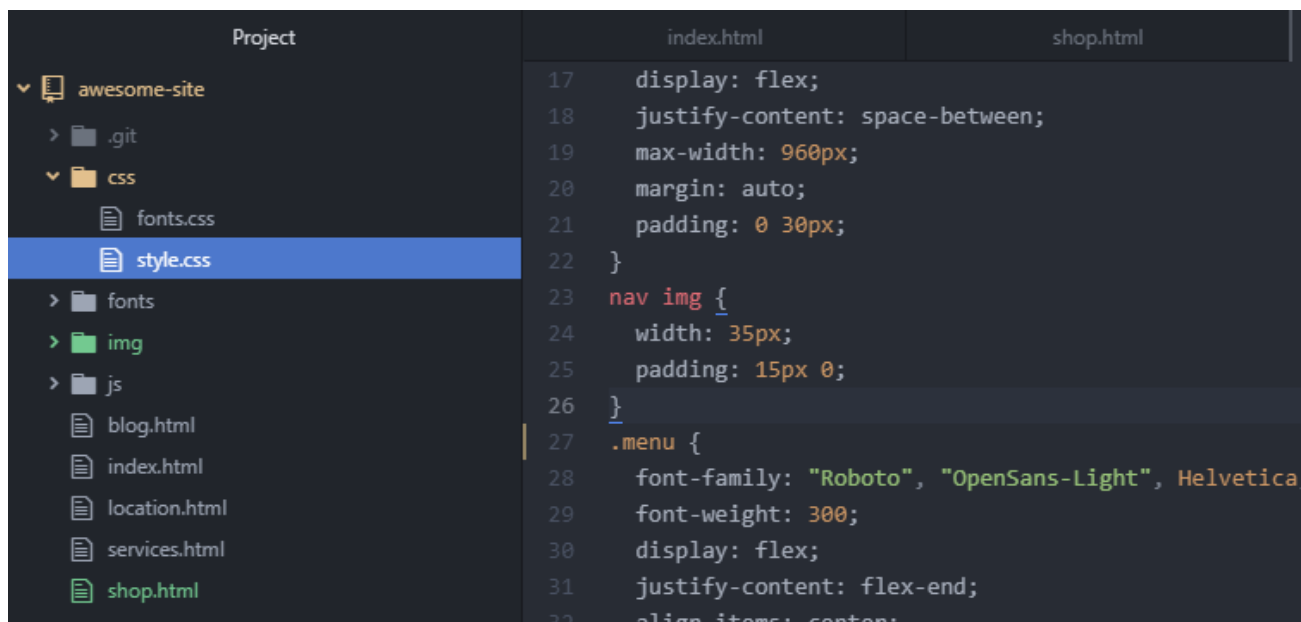


Рис. 217 – Подсказка редактора по контролю версиями

Так современные редакторы реагируют на контроль версий, помогая разработчику и подсвечивая новые и изменённые файлы, а также конкретные строки кода, где были внесены изменения. Удобно, не правда ли?

Откроем консоль и зафиксируем эти изменения. Сначала добавим новые файлы в индекс командой `git add`.

Затем командой: `git commit -m "Добавлена страница магазин, подключена библиотека Bootstrap"` зафиксируем изменения в `git`.

После этого файлы больше не подсвечиваются изменёнными, а `git status` говорит, что добавить в `git` ничего не нужно. Отлично, давайте тогда отправим ветку в репозиторий: `git push origin first-branch`. И да, мы указываем ветку, в которой делали изменения, а не `master`.

К слову, вы можете внести новые изменения после коммита, снова сделать коммит и эти изменения соберутся в один индекс изменений. Однако в GitHub

после пуша, будет видно, что изменения были внесены не одним коммитом и по каждому из них можно будет просмотреть что было изменено. Это удобно для того, чтобы откатывать изменения не по всему пушу, а по отдельным коммитам.

```
C:\OpenServer\domains\awesome-site>git push origin first-branch
Enumerating objects: 28, done.
Counting objects: 100% (28/28), done.
Delta compression using up to 4 threads
Compressing objects: 100% (24/24), done.
Writing objects: 100% (24/24), 1.52 MiB | 1.83 MiB/s, done.
Total 24 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), completed with 3 local objects.
remote:
remote: Create a pull request for 'first-branch' on GitHub by visiting:
remote:   https://github.com/testemail7/awesome-site/pull/new/first-branch
remote:
To https://github.com/testemail7/awesome-site.git
 * [new branch]      first-branch -> first-branch
C:\OpenServer\domains\awesome-site>
```

Рис. 218 – Успешное выполнение команды git push

«Пуш» выполнен успешно. Но на сайте ничего не изменилось. Дело в том, что на сайте отображается ветка master. Мы же работали, с веткой first-branch. Чтобы изменения из этой ветки применились к ветке master их надо «слить» в одну.

Зачем нужны ветки? Ветка в таком случае – рабочая модификация. Её можно проверить перед слиянием, скажем на тестовой версии сайта. Код из ветки может просмотреть ваш коллега, и заметить ошибку до внесения в основную ветку. Эту ветку ваш напарник может взять себе, дополнить, и снова отправить в репозиторий. И просто каждый из команды, может работать со своей веткой, без конфликта с изменениями другого участника.

Конечно, конфликты могут возникнуть потом, при слиянии нескольких веток с мастером. Но нам пока это не грозит, сейчас на этом останавливаться не будем.

Для слияния изменений заходим в GitHub. Здесь уже есть уведомление о том, что в репозиторий были отправлены изменения и предлагается сделать pull request – то есть запрос на слияние.

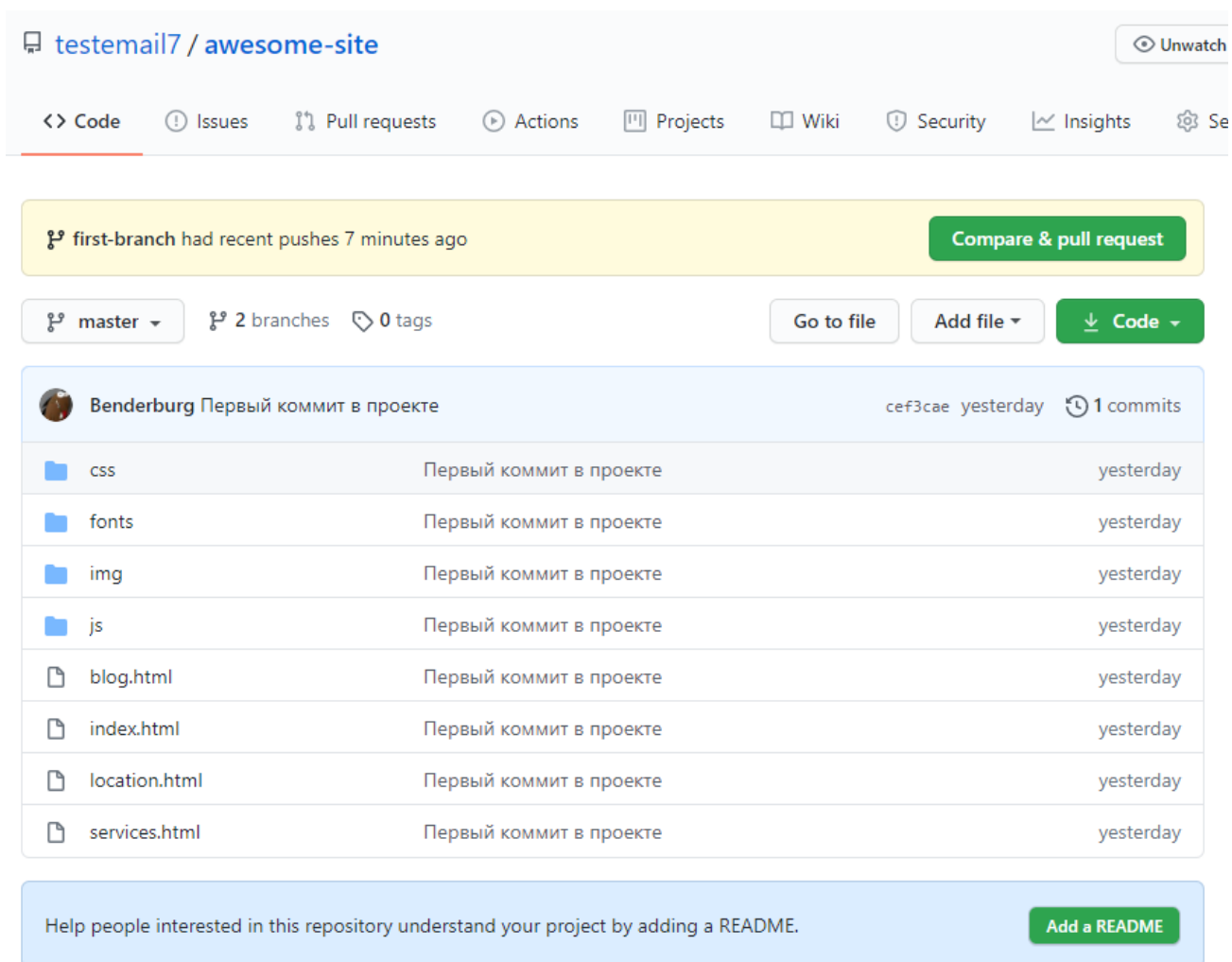


Рис. 219 – Подсказка о свежих данных, отправленных в репозиторий

Нажимаем «Compare & pull request» и переходим на страницу создания запроса.

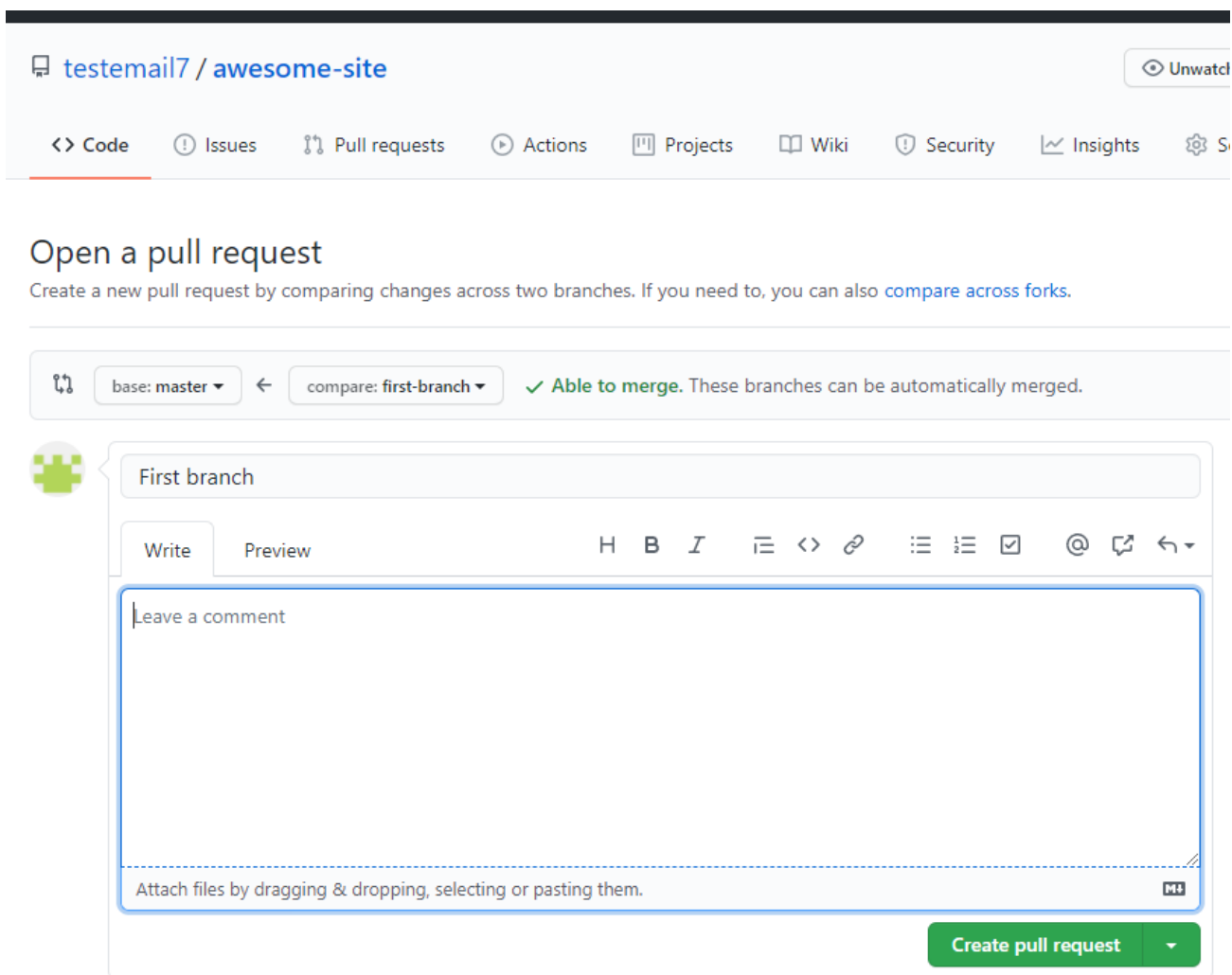


Рис. 220 – Создаём запрос на «слияние»

Ниже информация о merge и изменения, которые были в них сделаны. Что добавлено, удалено, модифицировано. В текстовом поле можем оставить комментарий для того, кто будет принимать запрос на слияние. В данном случае – это мы сами, так что можно его не заполнять. Жмём «Create pull request».

После этого нас перенаправит на страницу запроса. Так у нас есть права на слияние веток, то активна кнопка «Merge pull request» - объединить с запросом изменений.

## First branch #1

Edit Open with ▾

testemail7 wants to merge 2 commits into `master` from `first-branch`

Conversation 0 Commits 2 Checks 0 Files changed 17 +335 -4

testemail7 commented 1 minute ago

No description provided.

Benderburg added 2 commits 22 minutes ago

Добавлена страница магазин, подключена библиотека Bootstrap 076ef37

Добавлена страница магазин, подключена библиотека Bootstrap ✓7a66bb1

Add more commits by pushing to the `first-branch` branch on `testemail7/awesome-site`.

All checks have passed

3 neutral and 2 successful checks

Header rules - awesome-site-iti Completed in 4s — No header rules processed Details

Pages changed - awesome-site-iti Completed in 4s — 17 new files uploaded Details

Redirect rules - awesome-site-iti Completed in 4s — No redirect rules processed Details

Mixed content - awesome-site-iti Successful in 4s — No mixed content detected Details

netlify/awesome-site-iti/deploy-preview — Deploy preview ready! Details

This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request ▾ You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Reviewers

No reviews

Still in progress? Convert to draft

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Linked issues

Successfully merging this pull request may close these issues.

None yet

Notifications

Customize

Unsubscribe

You're receiving notifications because you're watching this repository.

2 participants

Рис. 221 – Информация о запросе

И подтвердить – тут также можно добавить комментарий для лучшего документирования изменений в проекте.

Add more commits by pushing to the `first-branch` branch on `testemail7/awesome-site`.

Merge pull request #1 from testemail7/first-branch

First branch

Confirm merge Cancel

Write Preview

H B I ≡ <> 🔗 ≡ ≡ ☑ @ ↩

Рис. 222 – Подтверждение команды объединения изменений с главным кодом приложения

Наш сайт небольшой, потому Netlify скорее всего быстро обновит информацию. Обновляйте ссылку на ваш сайт и проверяйте, что изменения опубликованы в сети.

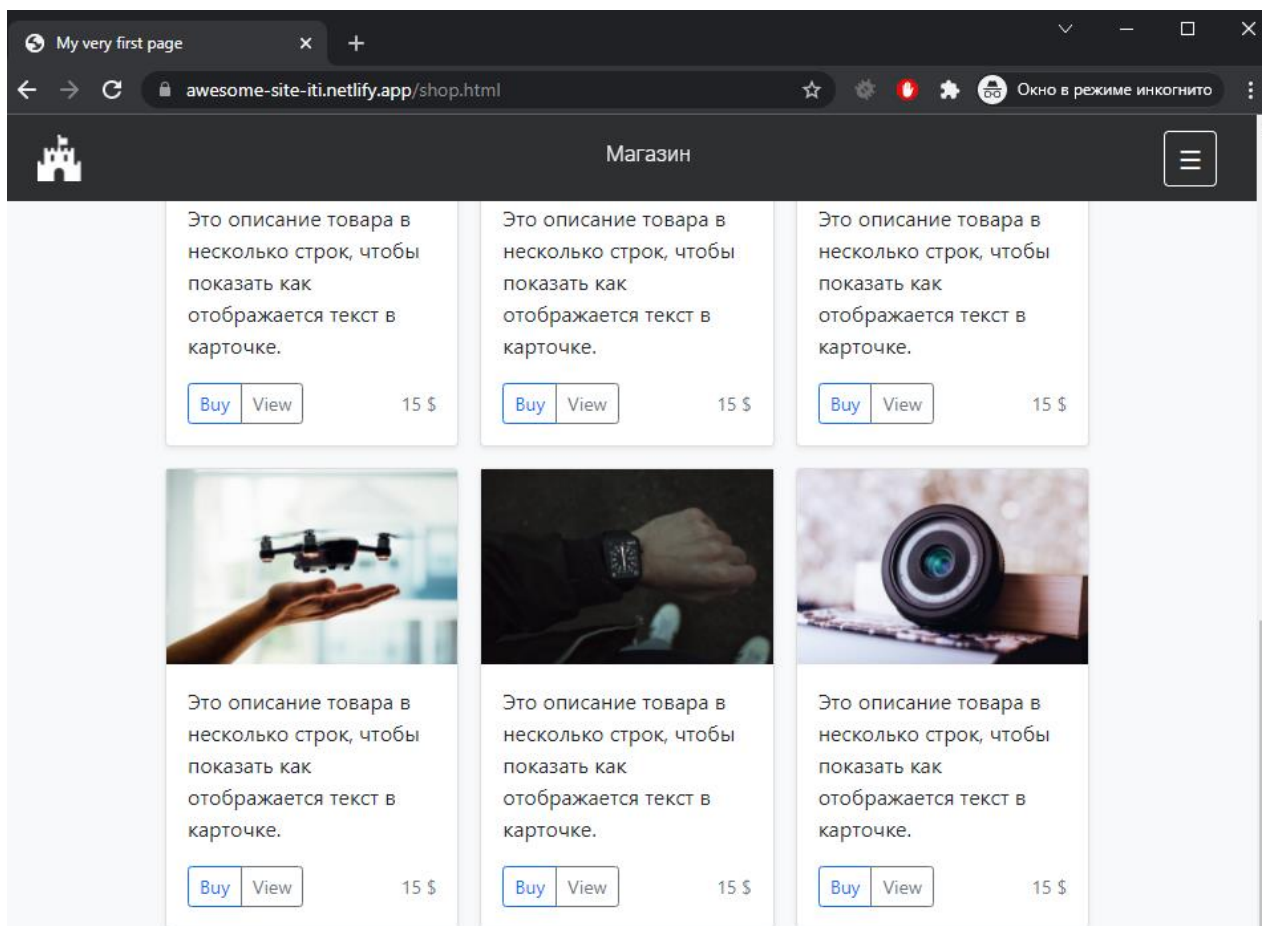


Рис. 223 – Проверка отображения актуальной версии веб-приложения в сети

Поздравляю, вы умеете создавать и публиковать сайты. Теперь можно создать свой уникальный проект, опубликовать его, прикрепить ссылки на него и репозиторий в своё портфолио и создать резюме для hh.ru или веб-студии.

### **Задание на контроль**

1. Повторить пример из практической части.
2. Установить контроль версий и инициализировать проект.
3. Зарегистрировать учётную запись в хранилище репозиторий и добавить проект в удалённый репозиторий.
4. Зарегистрировать учётную запись в сервисе Netlify и опубликовать проект в сети.
5. Подключить библиотеку Bootstrap 5 к проекту, создать новую страницу используя возможности библиотеки.
6. Добавить изменения в репозиторий и опубликовать обновления на сайт через автоматический деплой в Netlify.
7. Лабораторная работа считается защищенной, если студент показал в окне браузера страницу из практической части со стилизованным меню и ответил правильно на все заданные контрольные вопросы (следующая лабораторная работа не подлежит защите до тех пор, пока не будет защищена предыдущая).