

ДИСЦИПЛИНА	Фронтенд и бэкенд разработка
ИНСТИТУТ	ИПТИП
КАФЕДРА	Индустриального программирования
ВИД УЧЕБНОГО МАТЕРИАЛА	Методические указания к практическим занятиям
ПРЕПОДАВАТЕЛЬ	Астафьев Рустам Уралович
СЕМЕСТР	1 семестр, 2025/2026 уч. год

Ссылка на материал: <https://github.com/astafiev-rustam/frontend-and-backend-development/tree/Practice-1-1>

Практическое занятие 1: Инициализация Git-репозитория, работа с GitHub и публикация проекта на GitHub Pages

Цель: Освоить базовые принципы работы с системой контроля версий Git и платформой GitHub. Научиться создавать репозиторий, управлять ветками, оформлять Pull Request и публиковать статический сайт с помощью GitHub Pages. Заложить основу для дальнейшей работы над проектом.

Установка Git и среды разработки на компьютеры

Установка на Windows

Скачайте установщик Git с официального сайта [1](#). Запустите установщик и следуйте инструкциям. Выберите параметры по умолчанию, если не уверены.

Установка на macOS

Если у вас установлен Homebrew, просто выполните команду в терминале:

```
brew install git
```

Либо скачайте установщик с официального сайта [2](#).

Установка на Linux

Для Ubuntu или Debian выполните:

```
sudo apt update
sudo apt install git
```

Для Fedora:

```
sudo dnf install git
```

Для Arch Linux:

```
sudo pacman -S git
```

Проверка установки

В конечном счете, после установки, в терминале необходимо выполнить команду: `git --version` В результате должна быть выведена версия Git, что и будет признаком верной установки.

Установка среды разработки

В качестве среды разработки на практических занятиях будет использоваться Visual Studio Code. О ее установке можно почитать [здесь](#) 3.

Введение и теоретическая база

Работа современного веб-разработчика немыслима без систем контроля версий. Сегодня вы освоите фундаментальный инструмент — Git — и научитесь работать с платформой GitHub, которая является социальной сетью для разработчиков и мощным инструментом для командной работы.

Ключевые концепции, которые необходимо понять перед началом работы:

Репозиторий (Repository) — это хранилище вашего проекта. Это не просто папка с файлами, а специальным образом организованное пространство, внутри которого Git отслеживает все изменения, происходящие с каждым файлом. Репозиторий содержит всю историю проекта, всех авторов и все когда-либо сделанные изменения.

Коммит (Commit) — это основной объект в системе контроля версий. Коммит представляет собой снимок состояния вашего проекта в определенный момент времени. Его можно сравнить с сохранением в игре. Каждое сохранение (коммит) имеет уникальный идентификатор, автора, дату и, что самое важное, понятное сообщение, которое объясняет, что было изменено и для чего. Правилom хорошего тона является создание частых и атомарных коммитов, то есть таких, которые содержат изменения, направленные на решение одной конкретной небольшой задачи.

Ветка (Branch) — это указатель на определенный коммит. Ветки позволяют вести параллельную разработку, изолируя различные функциональности или эксперименты друг от друга. По умолчанию создается ветка с именем `main` (ранее использовалось `master`). В этой ветке должен всегда находиться стабильный, работоспособный код, готовый к развертыванию. Для повседневной работы принято создавать отдельную ветку, например, `dev` (от `development` — разработка), куда будут сливаться все новые функции перед их окончательным переносом в `main`.

Слияние (Merge) — это операция интеграции изменений из одной ветки в другую. Например, когда работа над функцией в ветке `dev` завершена и протестирована, эти изменения сливаются в ветку `main`.

Удаленный репозиторий (Remote) — это версия вашего проекта, размещенная на сервере в интернете (например, на GitHub, GitLab или Bitbucket). Он служит для резервного копирования вашего кода, публикации и, что важно, для организации командной работы. Локальный репозиторий на вашем компьютере может быть связан с несколькими удаленными репозиториями.

Pull Request (PR) или Merge Request (MR) — это центральный механизм процесса код-ревью в современных IT-командах. Это не техническая особенность Git, а функционал платформ типа GitHub. Разработчик, завершив работу в своей ветке, создает Pull Request — запрос на слияние своих изменений в целевую ветку (например, `dev -> main`). В рамках этого запроса команда обсуждает код, проверяет его и оставляет правки. После прохождения всех проверок изменения мержутся.

GitHub Pages — это бесплатный сервис хостинга статических сайтов от GitHub. Он автоматически публикует сайт из указанной вами ветки и папки репозитория. Ваш сайт будет доступен по адресу: `https://<ваш-username>.github.io/<имя-репозитория>/`.

Вспомогательные файлы проекта:

README.md — это визитная карточка вашего проекта. Файл создается в формате Markdown (.md) и должен содержать исчерпывающую информацию для тех, кто впервые видит ваш проект: название, описание, технологии, инструкции по установке и запуску, примеры использования. Наличие качественного README — признак профессионализма.

.gitignore — это конфигурационный файл, в котором указываются шаблоны файлов и папок, которые Git должен игнорировать и не добавлять в репозиторий. Это необходимо для того, чтобы не засорять историю проекта служебными файлами операционной системы, редактора кода, зависимостями (например, папка `node_modules`) или файлами с чувствительными данными (пароли, ключи).

Последовательность выполнения работы (Демонстрация и Задание)

Данный раздел является пошаговым руководством к действию. Следуйте ему для выполнения базового задания.

Шаг 1: Подготовка локального репозитория

Начните с создания на вашем компьютере новой папки для проекта. Назовите ее, например, `my-awesome-project`. Откройте эту папку в вашем редакторе кода (например, VS Code), который имеет встроенный терминал, или перейдите в нее через системный терминал (командную строку).

Инициализируйте пустой Git-репозиторий внутри этой папки. Для этого в терминале необходимо выполнить команду:

```
git init
```

Эта команда создаст скрытую папку `.git`, в которой Git будет хранить всю служебную информацию и историю изменений.

Шаг 2: Создание базовой структуры проекта и первого коммита

Теперь необходимо создать первоначальную структуру файлов и папок. Рекомендуется придерживаться общепринятых стандартов. Создайте следующие элементы:

- Файл `README.md`.
- Файл `.gitignore`.
- Папку `src` для исходных кодов.
- Папку `assets` для статических ресурсов (изображения, шрифты, иконки).

Откройте файл `.gitignore` и добавьте в него базовые шаблоны для игнорирования. Вы можете воспользоваться готовым шаблоном для вашего языка программирования на сайте gitignore.io (например, для `macOS`, `Windows`, `Linux`, `Node.js`). Минимально можно добавить служебные папки ОС:

```
# Игнорирование служебных файлов ОС
.DS_Store
Thumbs.db

# Игнорирование логов и зависимостей
npm-debug.log*
node_modules/
```

В файл `README.md` добавьте первоначальное описание проекта, используя Markdown-разметку:

```
# Мой первый проект
```

```
## Проект
Этот репозиторий является каркасом для будущего веб-приложения.

## Ссылки
- [Публичная страница на GitHub Pages](https://ВАШ_ЛОГИН.github.io/ИМЯ_РЕПОЗИТОРИЯ/)
```

Теперь нужно зафиксировать созданную структуру в истории Git. Этот процесс состоит из двух stages. Сначала с помощью команды `git add` мы добавляем изменения в так называемую «индекс» (staging area), подготавливая их к коммиту. Затем командой `git commit` мы создаем сам коммит.

Выполните в терминале следующие команды:

```
# Добавить все новые и измененные файлы в индекс (кроме тех, что в .gitignore)
git add .

# Создать коммит с описанием. Описание должно быть понятным и на английском.
git commit -m "feat: initialize project structure with README and gitignore"
```

Флаг `-m` позволяет написать сообщение коммита прямо в командной строке. Приставка `feat:` является частью соглашения по именованию коммитов (Conventional Commits) и означает добавление новой функциональности.

Шаг 3: Создание репозитория на GitHub и привязка к нему (Remote)

Перед выполнением данного шага необходимо настроить доступ по SSH ключу к удаленным репозиториям с GitHub, если у вас это ещё не сделано. Для этого необходимо:

Шаг 3.1: Проверка существующих SSH-ключей

Сначала откройте терминал (Git Bash на Windows, Terminal на Mac/Linux) и проверьте, есть ли у вас уже созданные ключи:

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

Ищите файлы с именами `id_rsa` и `id_rsa.pub` (или `id_ed25519` и `id_ed25519.pub`). Если таких файлов нет, переходите к следующему шагу.

Шаг 3.2: Генерация нового SSH-ключа

Сгенерируйте новый ключ, подставив ваш email, который привязан к GitHub.

Современный алгоритм (рекомендуется):

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

Или классический алгоритм RSA:

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

Нажмите `Enter` для всех запросов (чтобы использовать пути по умолчанию и не устанавливать парольную фразу, если не хотите).

В результате в папке `~/.ssh/` будут созданы два файла:

- `id_ed25519` (или `id_rsa`) — ваш **ЗАКРЫТЫЙ** (приватный) ключ. Никому и никогда его не передавайте!
- `id_ed25519.pub` (или `id_rsa.pub`) — ваш **ОТКРЫТЫЙ** (публичный) ключ. Его мы сейчас скопируем.

Шаг 3.3: Добавление SSH-ключа в ssh-agent (Демон управления ключами)

Запустите ssh-agent в фоновом режиме и добавьте в него ваш закрытый ключ.

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

# Добавить ваш закрытый ключ в агент.
# Для ключа ed25519:
ssh-add ~/.ssh/id_ed25519
# Или для ключа rsa:
ssh-add ~/.ssh/id_rsa
```

Шаг 3.4: Добавление SSH-ключа в ваш аккаунт GitHub

1. Скопируйте содержимое открытого ключа в буфер обмена.

- **Mac:** Выполните в терминале: `pbcopy < ~/.ssh/id_ed25519.pub`
- **Linux:** Выполните: `cat ~/.ssh/id_ed25519.pub` и скопируйте вывод мышью или с помощью `xclip`.
- **Windows (Git Bash):** Выполните: `cat ~/.ssh/id_ed25519.pub` и скопируйте вывод из терминала. Или найдите файл `.pub` в папке `C:\Users\ВАШ_ЛОГИН\.ssh\`, откройте его блокнотом и скопируйте всё содержимое.

2. Перейдите на GitHub:

- Кликните на вашу аватарку в правом верхнем углу и выберите **Settings**.
- В левом боковом меню выберите **SSH and GPG keys**.
- Нажмите кнопку **New SSH key**.
- В поле "Title" дайте ключу понятное название (например, "My Laptop").
- В поле "Key" вставьте скопированное содержимое вашего файла `*.pub`.
- Нажмите **Add SSH key**.

Шаг 3.5: Проверка подключения

Вернитесь в терминал и проверьте, что всё работает:

```
ssh -T git@github.com
```

Вы должны увидеть сообщение примерно такого содержания:

```
Hi username! You've successfully authenticated, but GitHub does not provide
shell access.
```

Это значит, что аутентификация прошла успешно!

Теперь необходимо разместить ваш код на GitHub. Войдите в свой аккаунт на github.com. На главной странице найдите и нажмите кнопку «New» для создания нового репозитория.

- **Repository name:** Укажите имя вашего репозитория (например, `my-awesome-project`).
- **Description:** Краткое описание (опционально).
- **Public/Private:** Оставьте репозиторий публичным (Public).
- **Initialize this repository with...:** НЕ ставьте галочки ни в одном из пунктов. Мы уже инициализировали репозиторий локально, и нам не нужно, чтобы GitHub создавал лишние файлы (`README`, `.gitignore`).

Нажмите «Create repository».

После создания репозитория GitHub покажет вам инструкцию по подключению существующего репозитория. Нас интересует блок «...or push an existing repository from the command line». Скопируйте оттуда две команды и выполните их в вашем терминале:

```
# Эта команда связывает ваш локальный репозиторий с удаленным на GitHub.
# origin — это стандартное имя для удаленного репозитория по умолчанию.
git remote add origin https://github.com/ВАШ_ЛОГИН/my-awesome-project.git

# Эта команда отправляет (push) вашу ветку main на удаленный репозиторий (origin)
# и устанавливает ее как upstream-ветку (флаг -u) для будущих синхронизаций.
git push -u origin main
```

После выполнения этих команд обновите страницу вашего репозитория на GitHub. Вы увидите, что все ваши файлы появились в нем.

Шаг 4: Создание ветки разработки `dev` и настройка GitHub Pages

Ветвление — ключевая практика. Мы не будем вести разработку прямо в `main`. Создадим ветку `dev`.

Эту ветку можно создать как через интерфейс GitHub, так и через терминал. Создадим ее через терминал и сразу отправим на сервер:

```
# Создать новую ветку с именем dev и переключиться на нее
git checkout -b dev

# Отправить новую ветку dev на GitHub
git push -u origin dev
```

Теперь необходимо настроить публикацию сайта через GitHub Pages. Перейдите в вашем репозитории на GitHub во вкладку «Settings». В левом меню выберите пункт «Pages».

- В разделе «Source» выберите ветку `main` (или `dev`, если хотите публиковать тестовую версию) и папку `/ (root)`.
- Нажмите «Save».

Через 1-2 минуты ваш сайт будет опубликован. Сверху страницы настроек появится зеленая плашка с ссылкой вида `https://ВАШ_ЛОГИН.github.io/my-awesome-project/`. Добавьте эту ссылку в файл `README.md`.

Шаг 5: Создание базового `index.html` и публикация через Pull Request

Любая функциональность добавляется в проект через процесс код-ревью. Сейчас мы добавим главную страницу сайта.

Убедитесь, что вы находитесь в ветке `dev` (команда `git status` должна показать `On branch dev`). Создайте в папке `src` файл `index.html` со следующим минимальным каркасом:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <!-- Важная мета-информация для корректного отображения на мобильных устройствах -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Мой первый проект</title>
  <!-- Подключение фавиконки -->
  <link rel="icon" href="../assets/favicon.ico" type="image/x-icon">
</head>
<body>
  <h1>Заглушка</h1>
  <p>Скоро здесь будет крутой сайт!</p>
</body>
</html>
```

Найдите в интернете любое изображение в формате `.ico` (фавиконка), скачайте его и поместите в папку `assets`, назвав `favicon.ico`.

Теперь необходимо зафиксировать изменения и отправить их в ветку `dev` на GitHub:

```
# Добавить измененные файлы
git add src/index.html assets/favicon.ico

# Создать коммит
git commit -m "feat: add basic index.html with favicon and viewport meta tag"

# Отправить изменения в ветку dev на GitHub
git push origin dev
```

Теперь изменения находятся в ветке `dev` на GitHub, но не в `main`. Чтобы их туда перенести, нужно создать Pull Request.

На GitHub перейдите во вкладку «Pull requests» вашего репозитория и нажмите «New pull request».

- В качестве `base` (куда мержим) выберите ветку `main`.
- В качестве `compare` (что мержим) выберите ветку `dev`.
- Нажмите «Create pull request».
- Дайте PR понятное название и описание (например, «Add initial website stub»).
- Нажмите «Create pull request» еще раз.

В идеальных условиях другой член команды провел бы ревью вашего кода. В рамках этой практики вы можете самостоятельно проверить изменения во вкладке «Files changed» и затем нажать кнопку «Merge pull request», а затем «Confirm merge». Таким образом, изменения из `dev` будут слиты в `main`.

После слияния подождите около минуты, и GitHub Pages автоматически перепубликует ваш сайт из обновленной ветки `main`. Перейдите по ссылке, которую вы указали в README, и убедитесь, что ваша страница-заглушка отображается.

Если страница не обновляется, то это может быть связано с принципами генерации. В таком случае вынесите файл `index.html` из текущей папки в корневую и проверьте результат.

Дополнительные задания (для углубленного изучения)

1. **Защита ветки `main`:** В настройках репозитория на GitHub перейдите в раздел «Branches» в меню «Settings». Добавьте правило для ветки `main`. Включите опции:

- `Require a pull request before merging` — обязательное проведение PR.
- `Require approvals` — требовать хотя бы одно одобрение от другого участника (пока можно поставить 0 для теста).
- `Require status checks to pass before merging` — требовать прохождения всех проверок. Это предотвратит прямую запись в `main` и заставит всегда использовать процесс код-ревью.

2. **Создание шаблонов:**

- **Шаблон Issue:** В корне репозитория создайте папку `.github` и внутри нее папку `ISSUE_TEMPLATE`. Создайте файл `bug_report.md` с формой для описания багов.
- **Шаблон Pull Request:** В папке `.github` создайте файл `PULL_REQUEST_TEMPLATE.md`. Внутри опишите чек-лист для проверки PR, например:

```
## Чек-лист
- [ ] HTML прошел валидацию (W3C Validator)
- [ ] Добавлена фавиконка
- [ ] Добавлен мета-тег viewport
- [ ] Изменения не сломали существующую функциональность
```

Эти шаблоны стандартизируют процесс отчетности и ревью.

Критерии оценивания (Максимум 10 баллов)

- **3 балла:** Репозиторий создан и размещен на GitHub. GitHub Pages работает корректно, страница доступна по ссылке.
 - **3 балла:** Соблюдена структура проекта: присутствуют папки `/src`, `/assets`, файлы `README.md`, `.gitignore` и `src/index.html`.
 - **2 балла:** Файл `README.md` содержит разделы «Проект» и «Ссылки» с корректной информацией и рабочей ссылкой на Pages.
 - **2 балла:** История коммитов отображается на GitHub, сообщения коммитов осмысленные и описывают проделанную работу.
-

Итог занятия

По результатам выполнения базовых заданий вы создали и опубликовали свой первый проект, освоили основной workflow современного разработчика: работа в отдельной ветке, коммиты, пуши, создание Pull Request и слияние изменений. Это основа, на которую будет нанизываться весь последующий материал. Убедитесь, что ваша «публичная заглушка» доступна онлайн, и вы понимаете стандартный процесс работы через PR.