Лекция 2. Системы контроля версий

**Оглавление**

[**1.** **Понятие системы контроля версий. Развитие систем контроля версий.** 3](#_Toc165887010)

[**Что такое система контроля версий?** 3](#_Toc165887011)

[**Централизованные и распределенные системы контроля версий** 3](#_Toc165887012)

[**Централизованные системы контроля версий** 4](#_Toc165887013)

[**Распределенные системы контроля версий** 5](#_Toc165887014)

[**2.** **Архитектура Git.** 7](#_Toc165887015)

[**Архитектура трех деревьев** 8](#_Toc165887016)

[**3.** **Начало работы в Git.** 10](#_Toc165887017)

[**Установка Git под Windows** 10](#_Toc165887018)

**Системы контроля версий**

**Цель:** формирование знаний в отношении систем контроля версий, их принципов работы и области применения, интеграции в среды разработки и использования в работе.

**План лекции:**Введение.1. Понятие системы контроля версий. Развитие систем контроля версий.  
2. Архитектура Git.  
3. Начало работы в Git.  
4. Интеграция GitHub с IDE.Заключение.Список литературы.

# **Введение.**

Системы контроля версий (Version Control Systems, VCS) стали неотъемлемой частью современного процесса разработки программного обеспечения. Эти системы позволяют разработчикам отслеживать изменения в коде, сотрудничать с другими разработчиками и управлять версиями программных продуктов. На протяжении десятилетий системы контроля версий эволюционировали от простых локальных решений до сложных распределенных систем, предоставляющих мощные инструменты для управления проектами любого масштаба.

Одной из самых популярных и широко используемых систем контроля версий является Git. Созданный в 2005 году Линусом Торвальдсом для управления разработкой ядра Linux, Git быстро завоевал популярность благодаря своей распределенной архитектуре, высокой производительности и гибкости. В этой статье мы рассмотрим архитектуру Git, которая обеспечивает его уникальные возможности, а также основные принципы работы с этой системой.

Начало работы с Git может показаться сложным для новичков, однако освоение основных команд и понятий позволяет быстро войти в курс дела и начать эффективно использовать этот инструмент. Мы обсудим основные шаги по установке и настройке Git, а также основы работы с репозиториями и ветвями.

Интеграция Git с платформой GitHub открывает еще больше возможностей для разработчиков. GitHub предоставляет мощные инструменты для совместной работы, управления проектами и автоматизации процессов разработки. Мы рассмотрим, как интегрировать GitHub с различными интегрированными средами разработки (IDE), что позволяет значительно упростить процесс работы с кодом и повысить продуктивность разработчиков.

Эта статья призвана дать общее представление о системе контроля версий Git, ее архитектуре и принципах работы, а также показать, как использовать GitHub для эффективного управления проектами и интеграции с IDE.

# **Понятие системы контроля версий. Развитие систем контроля версий.**

Система контроля версий (*Version Control System, VCS*) представляет собой программное обеспечение, которое позволяет отслеживать изменения в документах, при необходимости производить их откат, определять, кто и когда внес исправления и т.п. В статье рассмотрены виды *VCS*, принципы их работы, а также приведены примеры программных продуктов.

**Что такое система контроля версий?**

Наверное, всем знакома ситуация, когда при работе над проектом, возникает необходимость внести изменения, но при этом нужно сохранить работоспособный вариант, в таком случае, как правило, создается новая папка, название которой скорее всего будет  “Новая папка” с дополнением в виде даты или небольшой пометки, в нее копируется рабочая версия проекта и уже с ним производится работа. Со временем количество таких папок может значительно возрасти, что создает трудности в вопросе отката на предыдущие версии, отслеживании изменений и т.п. Эта ситуация значительно ухудшается, когда над проектом работает несколько человек.

Для решения таких проблем как раз и используется система контроля версий, она позволяет комфортно работать над проектом как индивидуально, так в коллективе. *VCS* отслеживает изменения в файлах, предоставляет возможности для создания новых и слияние существующих ветвей проекта, производит контроль доступа пользователей к проекту, позволяет откатывать исправления и определять кто, когда и какие изменения вносил в проект. Основным понятием *VCS* является репозиторий (*repository*) – специальное хранилище файлов и папок проекта, изменения в которых отслеживаются. В распоряжении разработчика имеется так называемая “рабочая копия” (*working copy*) проекта, с которой он непосредственно работает. Рабочую копию необходимо периодически синхронизировать с репозиторием, эта операция предполагает отправку в него изменений, которые пользователь внес в свою рабочую копию (такая операция называется *commit*) и актуализацию рабочей копии, в процессе которой к пользователю загружается последняя версия из репозитория (этот процесс носит название *update*).

**Централизованные и распределенные системы контроля версий**

Системы контроля версий можно разделить на две группы: распределенные и централизованные.

**Централизованные системы контроля версий**

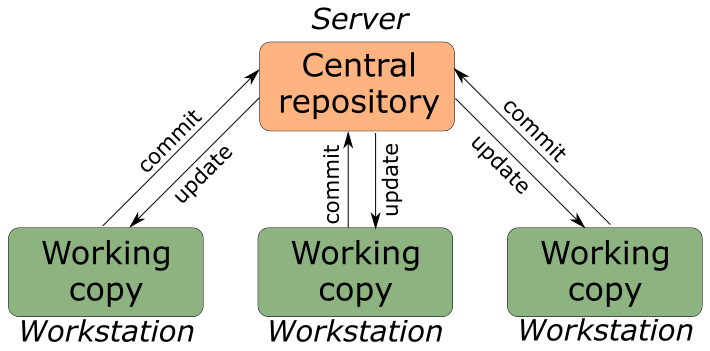
Централизованные системы контроля версий представляют собой приложения типа клиент-сервер, когда репозиторий проекта существует в единственном экземпляре и хранится на сервере (рис. 1). Доступ к нему осуществлялся через специальное клиентское приложение. В качестве примеров таких программных продуктов можно привести *CVS* (рис. 2), *Subversion* (рис. 3).

Рис. 1. Схема работы ЦСКВ

Рис. 2. Логотип CVS

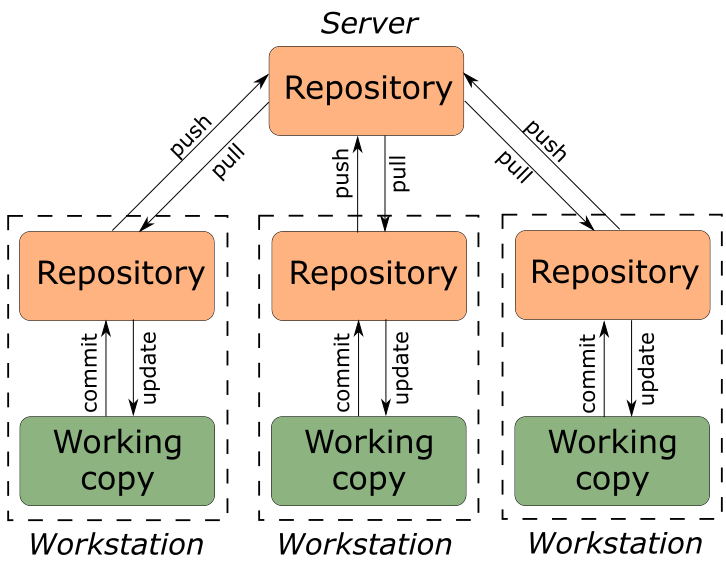
*CVS* (*Concurrent Versions System*, Система одновременных версий) одна из первых систем, получивших широкое распространение среди разработчиков, она возникла в конце 80-х годов прошлого века. В настоящее время этот продукт не развивается, это в первую очередь связано с рядом ключевых недостатков, таких как невозможность переименования файлов, неэффективное их хранение, практически полное отсутствие контроля целостности.

Рис. 3. Логотип Subversion

*Subversion* (*SVN*) – система контроля версий, созданная на замену *CVS*. *SVN* была разработана в 2004 году и до сих пор используется. Несмотря на многие преимущества по сравнению с *CVS* у *SVN* все-таки есть недостатки, такие как проблемы с переименованием, невозможность удаления данных из хранилища, проблемы в операции слияния ветвей и т.д. В целом *SVN* был (и остается) значительным шагом вперед по сравнению с *CVS.*

**Распределенные системы контроля версий**

Распределенные системы контроля версий (*Distributed Version Control System, DVCS*) позволяют хранить репозиторий (его копию) у каждого разработчика, работающего с данной системой (рис. 4). При этом можно выделить центральный репозиторий (условно), в который будут отправляться изменения из локальных и, с ним же эти локальные репозитории будут синхронизироваться. При работе с такой системой, пользователи периодически синхронизируют свои локальные репозитории с центральным и работают непосредственно со своей локальной копией. После внесения достаточного количества изменений в локальную копию они (изменения) отправляются на сервер. При этом сервер, чаще всего, выбирается условно, т.к. в большинстве *DVCS*нет такого понятия как “выделенный сервер с центральным репозиторием”.

Рис. 4. Схема работы РСКВ 

Большое преимущество такого подхода заключается в автономии разработчика при работе над проектом, гибкости общей системы и повышение надежности благодаря тому, что каждый разработчик имеет локальную копию центрального репозитория. Две наиболее известные *DVCS* – это *Git* и *Mercurial* (рис. 5).

Рис. 5. Логотип Mercurial 

Начнем с *Mercurial*, эта система представляет собой свободную *DVCS*, которая построена таким образом, что в ней отсутствует понятие центрального репозитория, для работы с этой *VCS* используется (как правило) консольная утилита *hg*. *Mercurial* обладает всеми возможностями системы контроля версий, такими как ветвление, слияние, синхронизация с другими репозиториями. Данный проект используют и поддерживают большое количество крупных разработчиков, среди них *Mozilla*, *OpenOffice*, *OpenJDK* и многие другие. Сам продукт написан на языке *Python* и доступен на большинстве современных операционных систем (*Windows*, *Mac OS*, *Linux*), также существует значительное количество утилит с графическим интерфейсом для работы с *Mercurial*. Основным конкурентом *Mercurial* на рынке распределенных систем контроля версий является *Git* (рис. 6), который, на сегодняшний день, выиграл гонку за лидерство.

Рис. 6. Логотип Git

*Git* – распределенная система контроля версий, разработанная Линусом Торвальдсем для работы над ядром операционной системы *Linux*. Среди крупных проектов, в рамках которых используется *git*, можно выделить ядро *Linux*, *Qt*, *Android*. *Git* свободен и распространяется под лицензией *GNU* *GPL* 2 и, также как *Mercurial*, доступен практически на всех операционных системах. По своим базовым возможностям *git* схож с *Mercurial* (и другими *DVCS*), но благодаря ряду достоинств (высокая скорость работы, возможность интеграции с другими *VCS*, удобный интерфейс) и очень активному сообществу, сформировавшемуся вокруг этой системы, *git* вышел в лидеры рынка распределенных систем контроля версий. Необходимо отметить, что несмотря на большую популярность таких систем как *git*, крупные корпорации, подобные *Google*, используют свои *VCS*.

# **Архитектура Git.**

Для того чтобы двигаться дальше – создавать репозитории, отправлять коммиты, делать новые бранчи (ветви) и т.п. необходимо предварительно ознакомиться с архитектурой git. Архитектуру мы рассмотрим на уровне, которого будет достаточно, чтобы понимать суть действий, которые мы будем совершать в дальнейшем.

**Архитектура трех деревьев**

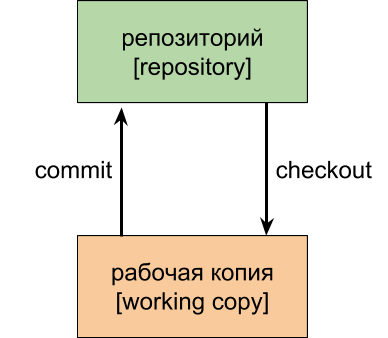
Система контроля версий git имеет архитектуру трех деревьев (рис. 7). Перед тем как перейти к ее описанию, для начала, рассмотрим архитектуру двух деревьев. Схематично она выглядит так, как представлено на рисунке ниже.

Рис. 7. Схема работы архитектуры двух деревьев

Для начала введем используемую в системах контроля версий терминологию. Набор файлов, с которым мы работаем в данный момент, называется рабочая копия (working copy). После того, как решено, что все нужные изменения на данный момент внесены, и об этом можно сообщить системе контроля версий, разработчик производит отправку изменений в репозиторий (repository). Репозиторий – это хранилище для нашего проекта, которое обслуживает система контроля версий. Сама операция отправки изменений называется commit, на русском языке ее так и называют – коммит. Если нам необходимо взять данные из репозитория, то мы осуществляем операцию checkout, которая на русском произносится как чекаут. Для названий операций commit и checkout не используют прямой перевод, предпочитают транскрипцию. В дальнейшем мы будем пользоваться как английским, так и русским эквивалентом терминов.

Для архитектуры двух деревьев регламент работы с репозиторием может выглядеть следующим образом:

1. Перед началом работы разработчик делает *checkout*, для того чтобы быть уверенным, что он будет работать с актуальной рабочей копией.
2. Разработчик вносит необходимые изменения в исходный код.
3. Разработчик отправляет изменения в репозиторий (коммитит их).
4. Повторить необходимое количество раз пункты 2 и 3.

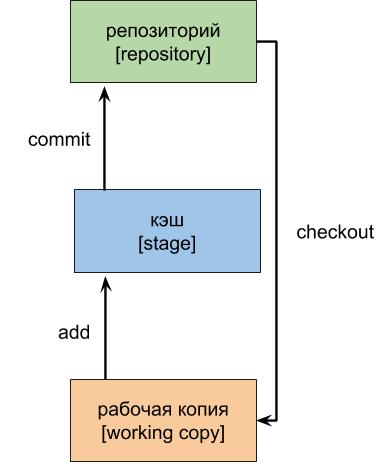
Система контроля версий git использует архитектуру трех деревьев. Схематично она выглядит так как показано на рисунке ниже.

Рис. 8. Схема работы архитектуры трех деревьев

Суть ее заключается в том, что дополнительно добавляется ещё одно место, которое можно назвать кэшем или stage в английской терминологии. Рабочая копия и репозиторий идейно не отличается от их аналогов в архитектуре двух деревьев. Наличие дополнительного элемента меняет регламент работы, которой в этом случае выглядит так:

1. Перед началом работы разработчик делает *checkout*, для того чтобы быть уверенным, что он будет работать с актуальной рабочей копией.
2. Разработчик вносит необходимые изменения в исходный код.
3. Разработчик отправляет необходимый набор файлов, изменения в которые внесены, в *stage* для того, чтобы потом построить из них коммит. До того, как изменения будут отправлены в репозиторий, разработчик может добавлять и удалять файлы из *stage*. Набор файлов в *stage*, как правило, идеологически связан между собой.
4. Разработчик отправляет изменения в репозиторий (коммитит их).
5. Повторить необходимое количество раз пункты 2 – 4.

Наличие stage добавляет гибкости в процесс разработки, вы можете внести изменения в довольно большое количество файлов, но отправить их в репозиторий в разных коммитах со своими специфическими комментариями.

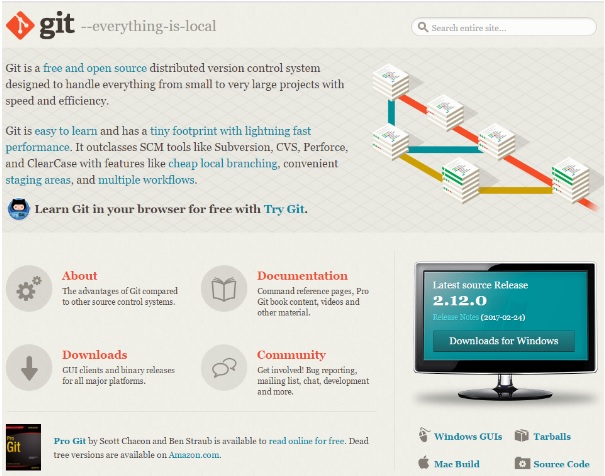
# **Начало работы в Git.**

Для того, чтобы начать работать с системой контроля версий *Git* ее необходимо предварительно установить. Рассмотрим варианты установки этой *VCS* под *MS Windows* и *Linux*.

**Установка Git под Windows**

Для установки *Git* под *Windows* необходимо предварительно скачать дистрибутив. Для этого перейдите на страницу https://git-scm.com/

Если вы зашли из‑под операционной системы (ОС) *Windows*, главная страница сайта будет выглядеть примерно так, как показано на рисунке 9 ниже. Для других ОС отличие будет заключаться в том, что изменится область для скачивания дистрибутива (см. правый нижний угол).

Рис. 9. Окно скачивания Git 

Для того чтобы скачать *Git* нужно нажать на кнопку *Downloads for Windows*, расположенную в правой части окна.

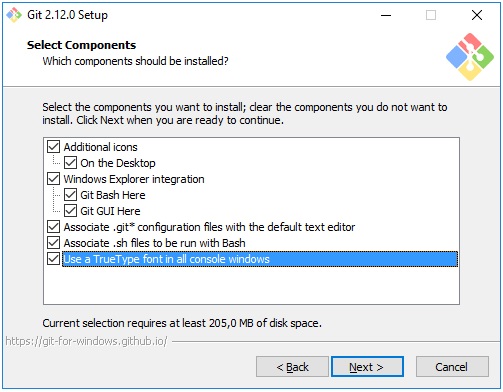
Процесс дальнейшей установки Git выглядит так.

1. Запустить установочный файл

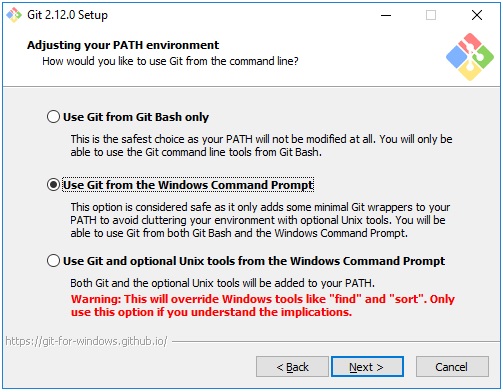
2. Ознакомиться, если есть желание, с лицензионным соглашением (рис. 10) и нажать на кнопку Next

Рис. 10. Окно лицензионной информации

3. Выбрать компоненты, которые следует установить (рис. 11):

  
Рис. 11. Окно установки Git – выбор компонентов

4. Указать способ использования Git (рис. 12):

  
Рис. 12. Окно установки Git – выбор способа использования

В этом окне доступны три возможных варианта:

* **Use Git from Git Bash only**

Переменная PATH не модифицируется и работа с Git возможна только через специализированную оболочку, которая называется Git Bash.

* ***Use Git from the Windows Command Prompt***

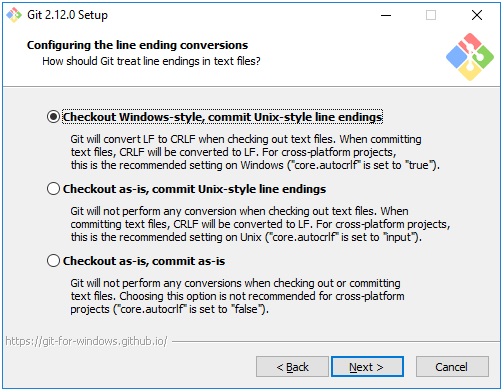
В этом случае происходит минимальная модификация переменной окружения PATH, которая позволит работать с Git через командную стоку Windows. Работа через Git Bash также возможна.

* **Use Git and optional Unix tools from the Windows Command Prompt**

В переменную PATH вносится значительное количество модификаций, которые позволят, в рамках командной строки Windows, использовать как Git так и утилиты Unix, которые поставляются вместе с дистрибутивом Git.

Наша рекомендация: опция Use Git from the Windows Command Prompt.

5. Настройка правил окончания строки (рис. 13):

  
Рис. 13. Окно установки Git – выбор правила окончания строки

Существует два варианта формирования конца строки в текстовых файлах – это Windows стиль и Unix стиль. Данное окно позволяет выбрать одну из опций, определяющих правило формирования окончания строки:

* ***Checkout Windows-style, commit Unix-style line endings***

Checkout (операция извлечения документа из хранилища и создания рабочей копии) производится в Windows стиле, а commit (операция отправки изменений в репозиторий) в Unix стиле.

* ***Checkout as-is, commit Unix-style line endigns***

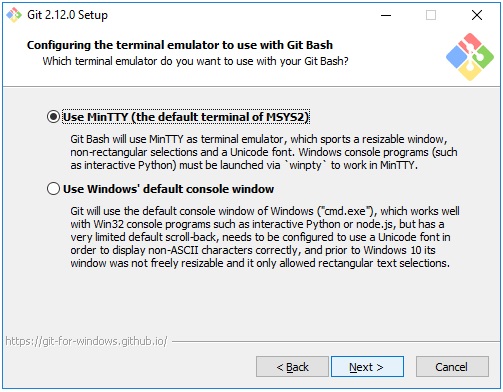
Checkout производится в том формате, в котором данные хранятся в репозитории, а commit осуществляется в Unix стиле.

* **Checkout as-is, commit as-is**

Checkout и commit производятся без дополительных преобразований.

Наша рекомендация: опция Checkout Windows-style, commit Unix-style line endings.

6. Выбор эмулятора терминала, который будет использован с Git Bash (рис. 14):

  
Рис. 14. Окно установки Git – выбор эмулятора терминала

Возможен выбор из двух вариантов:

* ***Use MinTTY (the defaul terminal of MSYS2)***

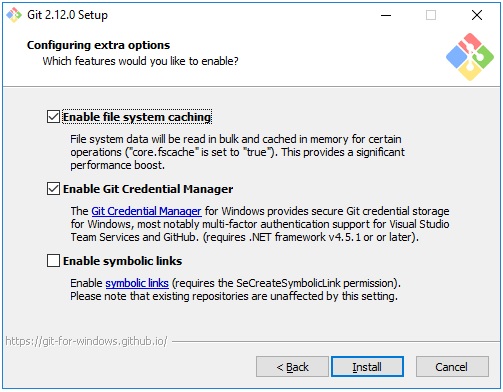
Git Bash будет использовать в качестве эмулятора терминала MinTTY.

* **Use Windows’ default console window**

Git будет использовать Windows консоль (“cmd.exe”).

Наша рекомендация: опция Use MinTTY (the defaul terminal of MSYS2).

7. Настройка дополнительных параметров (рис. 15)

  
Рис. 15. Окно установки Git – дополнительные параметры

Доступны следующие параметры:

* ***Enable file system caching***

Включение операции кэширования при работе с файлами. Эта опция позволит значительно повысить производительность.

* ***Enable Git Credential Manager***

Предоставляет возможность работы с защищенным хранилищем.

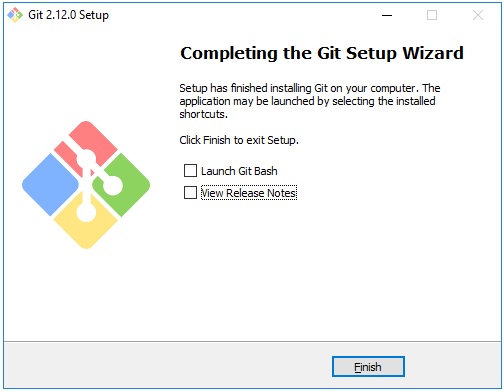
* ***Enable symbolic links***

Активирует работу с символьными ссылками.

Наша рекомендация: опции Enable file system caching и Enable Git Credential Manager.

8. Завершение установки

После нажатия на кнопку Install будет произведена установка Git на Windows, по окончании установки пользователь получит соответствующее сообщение (рис. 16).

  
Рис. 16. Окно завершения установки Git

**Установка Git под Linux**

Для установки *Git* под *Linux*, также необходимо зайти на сайт  <https://git-scm.com/> и перейти в раздел *Downloads*. В зависимости от используемой вами версии операционной системы *Linux* необходимо выбрать тот или иной способ установки *Git*.

Debian/Ubuntu

> apt-get install git

Fedora  
(Fedora 21)

> yum install git

(Fedora 22)

> dnf install git

Gentoo

> emerge --ask --verbose dev-vcs/git

Arch Linux

> pacman -S git

openSUSE

> zypper install git

Mageia

> urpmi git

FreeBSD

> pkg install git

Solaris 9/10/11 (*[OpenCSW](https://www.opencsw.org/)*)

> pkgutil -i git

Solaris 11 Express

> pkg install developer/versioning/git

OpenBSD

> pkg\_add git

Alpine

> apk add git

# **4. Интеграция GitHub с IDE**

[**Введение**](https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-ru)

Редактор Visual Studio Code (VS Code) стал одним из самых популярных для веб-разработки. Его популярность обусловлена множеством встроенных возможностей, в том числе интеграции с системой контроля исходного кода, а именно с Git. Использование возможностей Git из VS Code позволяет сделать рабочие процессы более эффективными и надежными.

В этом учебном модуле мы изучим интеграцию контроля исходного кода в VS с помощью Git.

[**Предварительные требования**](https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-ru)

Для этого обучающего модуля вам потребуется следующее:

* Git, установленный на вашем компьютере. Более подробную информацию о том, как добиться этого, можно найти в учебном модуле Введение в Git.
* Последняя версия Visual Studio Code, установленная на вашем компьютере.

[**Шаг 1 — Знакомство с вкладкой Source Control**](https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-ru#1-source-control)

Прежде всего, чтобы воспользоваться преимуществами интеграции контроля исходного кода, следует инициализировать проект как репозиторий Git.

Откройте Visual Studio Code и запустите встроенный терминал. Вы можете открыть его, используя сочетание клавиш CTRL + ` в Linux, macOS или Windows.

Используя терминал, создайте каталог для нового проекта и перейдите в этот каталог:

1. mkdir git\_test
2. cd git\_test

Затем создайте репозиторий Git:

1. git init

Также вы можете сделать это в Visual Studio Code, открыв вкладку Source Control (иконка выглядит как **развилка дороги**) в левой панели (рис. 17):

 Рис. 17. Значок вкладки Source Control

Затем нажмите кнопку **Open Folder (рис. 18)**:

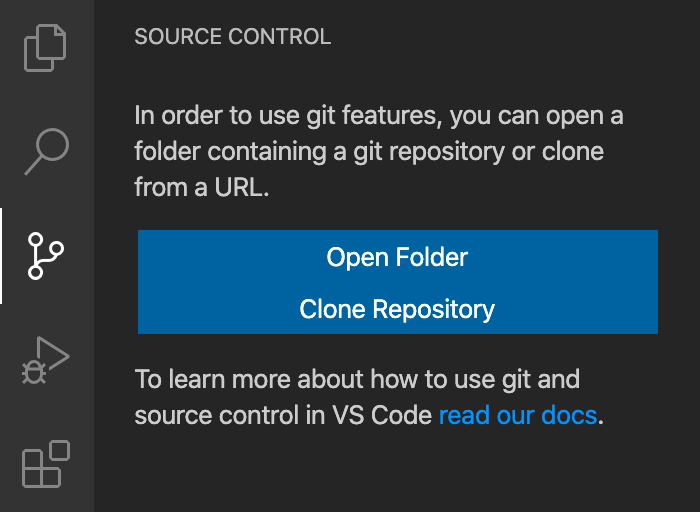


Рис. 18. Кнопка Open Folder

При нажатии кнопки откроется проводник файлов, где будет открыт текущий каталог. Выберите предпочитаемый каталог проекта и нажмите **Open**.

Затем нажмите **Initialize Repository (рис. 19)**:

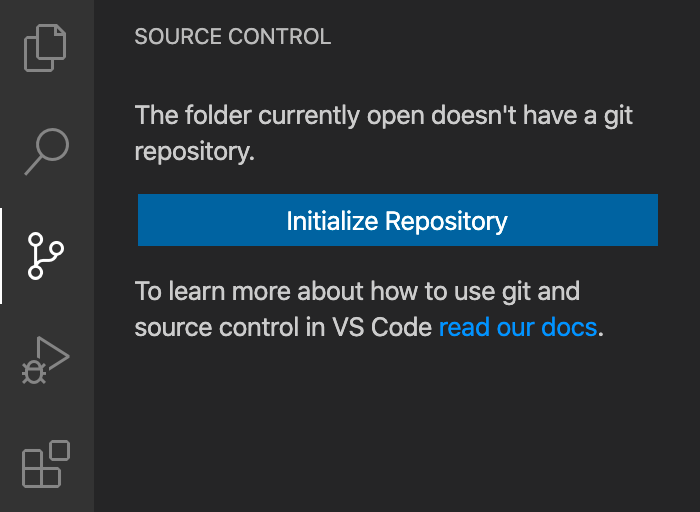


Рис. 19. Кнопка Initialize Repository

Если теперь вы посмотрите на свою файловую систему, вы увидите, что она содержит каталог .git. Чтобы сделать это, используйте терминал для перехода в каталог проекта и вывода его содержимого:

1. ls -la

Вы увидите созданный каталог .git:

1. Output
2. .
3. ..
4. .git

Это означает, что репозиторий инициализирован, и теперь вам следует добавить в него файл index.html.

После этого на **панели Source Control** вы увидите, что рядом с именем вашего нового файла отображается буква **U**. Обозначение **U** означает, что *файл не отслеживается*, то есть, что это новый или измененный файл, который еще не был добавлен в репозиторий (рис. 20):

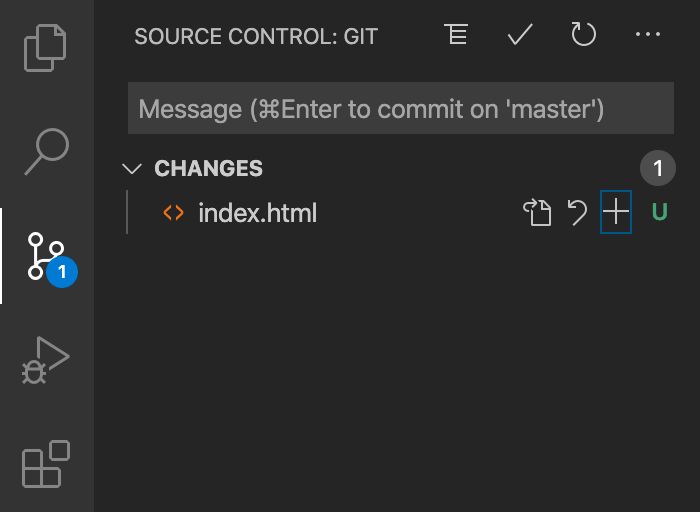


Рис. 20. Добавление файла в репозиторий

Вы можете нажать значок **плюс** (**+**) рядом с файлом index.html, чтобы включить отслеживание файла в репозитории.

После этого рядом с файлом появится буква **A**. **A** обозначает новый файл, который был добавлен в репозиторий (рис. 21).

Чтобы записать изменения, введите команду отправки в поле ввода в верхней части **панели Source Control**. Затем нажмите иконку отметки **check** для отправки файла в репозиторий.

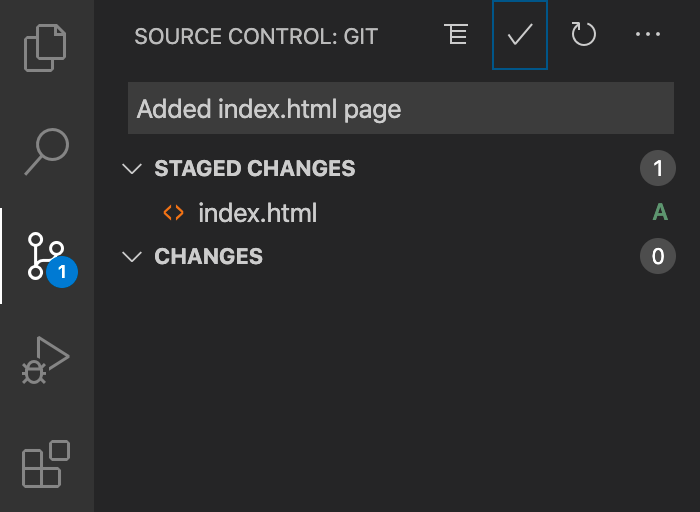


Рис. 21. Добавленный файл в репозиторий

После этого вы увидите, что несохраненных изменений нет.

Теперь добавьте немного содержания в файл index.html.

Вы можете использовать ярлык Emmet для генерирования базовой структуры кода HTML5 в VS Code, нажав !, а затем клавишу Tab. Теперь добавьте что-нибудь в раздел <body>, например, заголовок <h1>, и сохраните файл.

На панели исходного кода вы увидите, что ваш файл изменился. Рядом с именем файла появится буква **M**, означающая, что файл *изменен* (рис. 22):

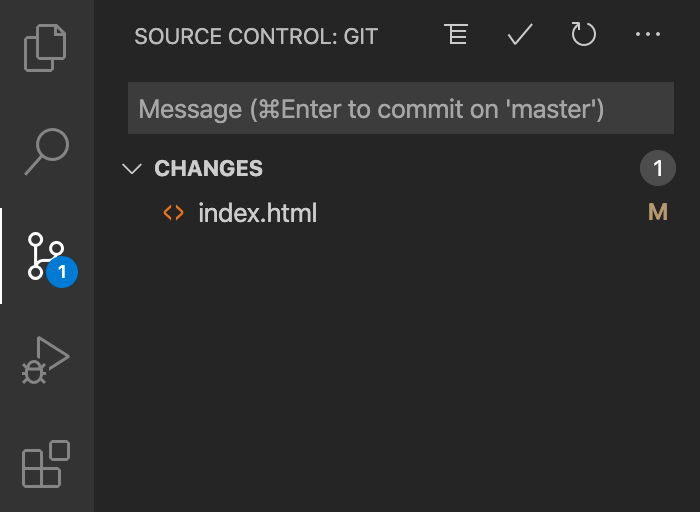


Рис. 22. Файл добавлен в репозиторий

Для практики давайте запишем это изменение в репозиторий.

Теперь вы познакомились с работой через панель контроля исходного кода, и мы переходим к интерпретации показателей gutter.

[**Шаг 2 — Интерпретация показателей Gutter**](https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-ru#2-gutter)

На этом шаге мы рассмотрим концепцию *Gutter* («Желоб») в VS Code. Gutter — это небольшая область справа от номера строки.

Если ранее вы использовали сворачивание кода, то в области Gutter находятся иконки **«Свернуть»** и **«Развернуть»**.

Для начала внесем небольшое изменение в файл index.html, например, изменим содержание внутри тега <h1>. После этого вы увидите, что измененная строка помечена в области Gutter синей вертикальной чертой. Синяя вертикальная черта означает, что соответствующая строка кода была изменена.

Теперь попробуйте удалить строку кода. Вы можете удалить одну из строк в разделе <body> вашего файла index.html. Обратите внимание, что в области Gutter появился красный треугольник. Красный треугольник означает строку или группу строк, которые были удалены.

Теперь добавьте новую строку в конец раздела <body> и обратите внимание на зеленую полосу. Вертикальная зеленая полоса обозначает добавленную строку кода.

В этом примере описаны индикаторы области Gutter для случаев изменения, удаления и добавления строки (рис. 23):

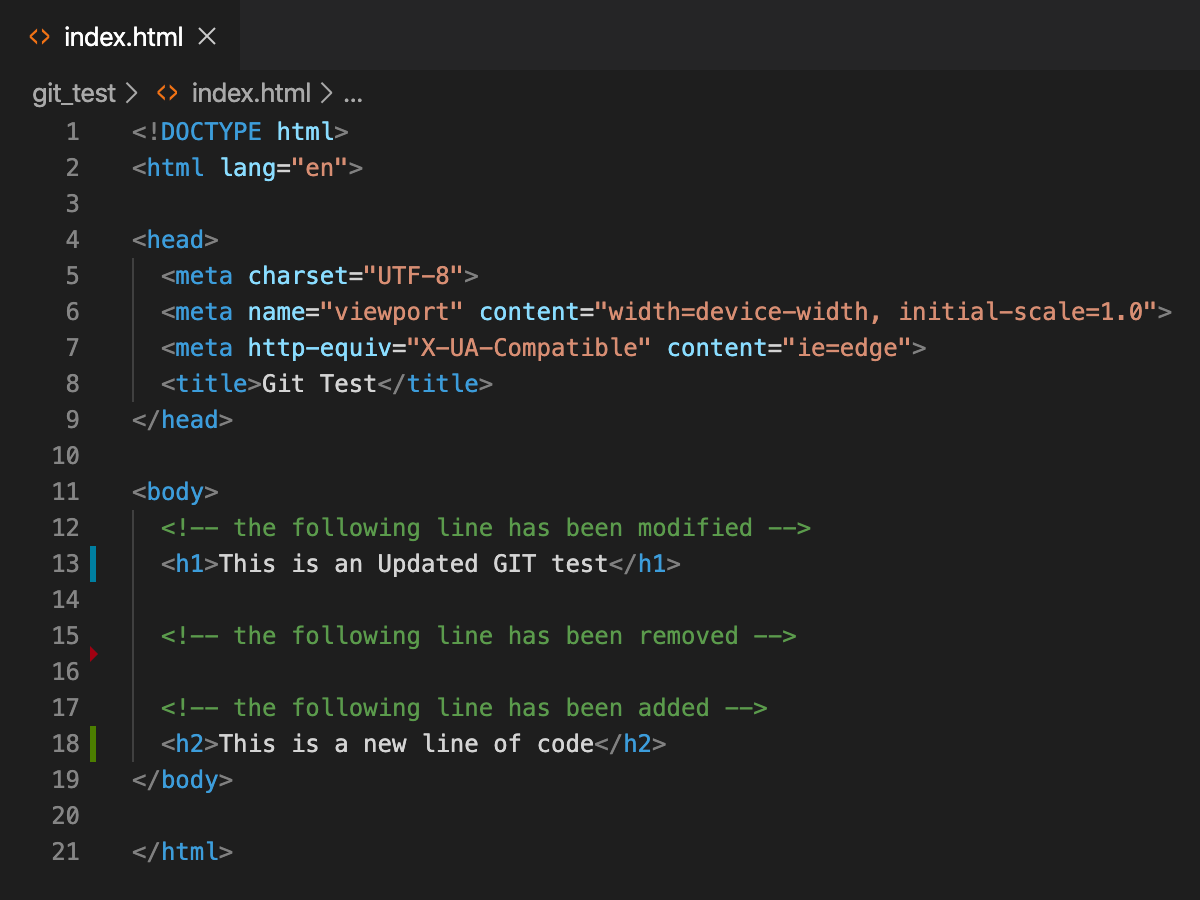


Рис. 23. Графическое отображение операций с кодом

[**Шаг 3 — Просмотр отличий файлов**](https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-ru#3)

VS Code также позволяет посмотреть отличия между разными версиями файла. Обычно для этого нужно загружать отдельный инструмент diff, так что встроенная функция повысит эффективность работы.

Чтобы посмотреть отличия, откройте панель контроля исходного кода и дважды нажмите на измененный файл. В этом случае следует дважды нажать на файл index.html. Откроется типовое окно сравнения, где текущая версия файла отображается слева, а ранее сохраненная в репозитории версия — справа (рис. 24).

В этом примере мы видим, что в текущей версии добавлена строка:

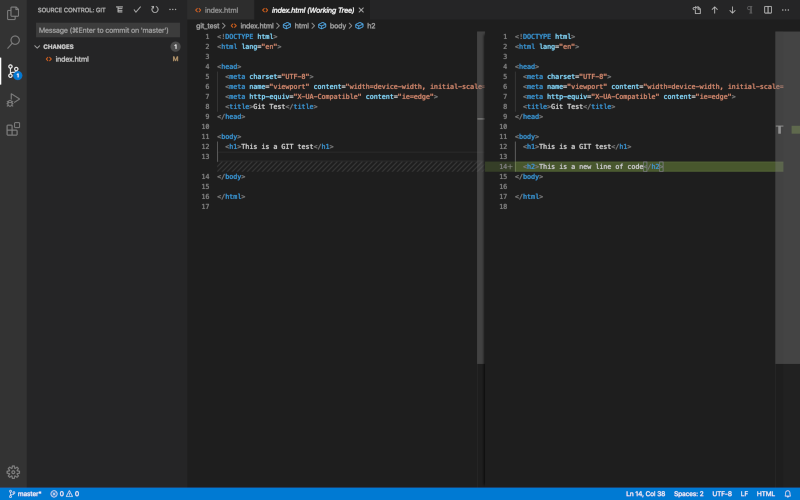


Рис. 24. Окно сравнения изменений в коде

[**Шаг 4 — Работа с ветвлением**](https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-ru#4)

Вы можете использовать нижнюю панель для создания и переключения ветвей кода. В нижней левой части редактора отображается иконка **контроля исходного кода** (которая выглядит как дорожная развилка), после которой обычно идет имя главной ветви или ветви, над которой вы сейчас работаете (рис. 25).

Индикатор ветви в нижней панели VS Code с именем: master

Рис. 25. Панель открытой ветки

Чтобы создать ветвление, нажмите на имя ветви. Откроется меню, где вы сможете создать новую ветвь:

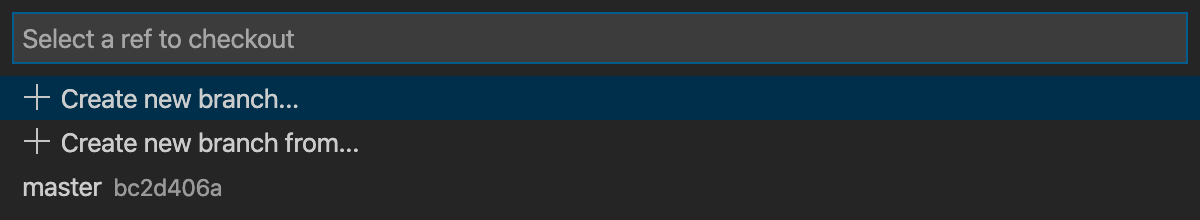


Рис. 26. Окно создания ветки

Создайте новую ветвь с именем test.

Теперь внесите изменение в файл index.html, чтобы перейти в новую ветвь test, например, добавьте текст this is the new test branch.

Сохраните эти изменения ветви test в репозитории. Затем нажмите на имя ветви в левом нижнем углу еще раз, чтобы переключиться обратно на главную ветвь master.

После переключения обратно на ветвь master вы увидите, что текст this is the new test branch, сохраненный для ветви test, отсутствует в главной ветви.

[**Шаг 5 — Работа с удаленными репозиториями**](https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-ru#5)

В этом учебном модуле мы не будем вдаваться в детали, но панель Source Control также предоставляет доступ для работы с удаленными репозиториями. Если вы уже работали с удаленными репозиториями, то вы увидите знакомые вам команды, такие как pull, sync, publish, stash и т.д.

[**Шаг 6 — Установка полезных расширений**](https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-ru#6)

В VS Code имеется не только множество встроенных функций для Git, но и несколько очень популярных расширений, добавляющих дополнительные функции.

[Git Blame](https://marketplace.visualstudio.com/items?itemName=waderyan.gitblame)

Это расширение дает возможность просматривать информацию Git Blame в панели состояния для текущей выделенной строки.

Английское слово Blame имеет значение «винить», но не стоит беспокоиться — расширение Git Blame призвано сделать процесс разработки более практичным, а не обвинять кого-то в чем-то плохом. Идея «винить» кого-то за изменения кода относится не к буквальному возложению вины, а к идентификации человека, к которому следует обращаться с вопросами в отношении определенных частей кода.

Как вы видите на снимке экрана, это расширение выводит на нижней панели инструментов небольшое сообщение, указывающее, кто изменял текущую строку кода, и когда было сделано это изменение.

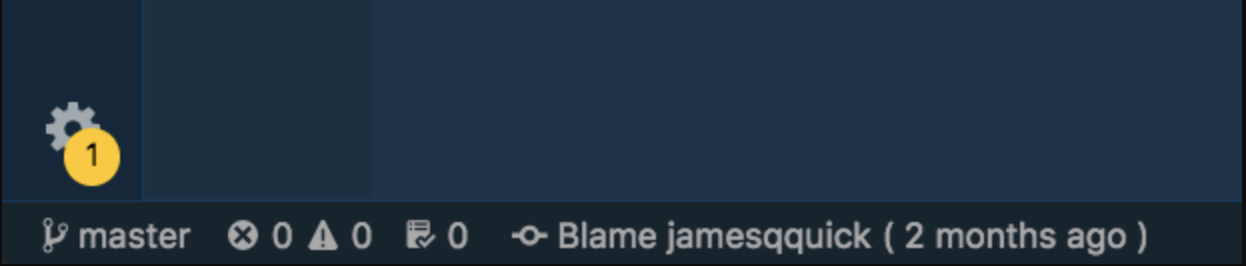


Рис. 26. Строка состояния, измененная с помощью GitBlame.

[Git History](https://marketplace.visualstudio.com/items?itemName=donjayamanne.githistory)

Хотя вы можете просматривать текущие изменения, сравнивать версии и управлять ветвлением с помощью встроенных функций VS Code, они не дают возможности просматривать историю Git. [Расширение Git History](https://marketplace.visualstudio.com/items?itemName=donjayamanne.githistory) решает эту проблему.

Как можно увидеть на снимке ниже, это расширение позволяет тщательно изучать историю файла, автора, ветви и т. д. Чтобы активировать показанное ниже окно Git History, нажмите на файл правой кнопкой мыши и выберите пункт Git: View File History:

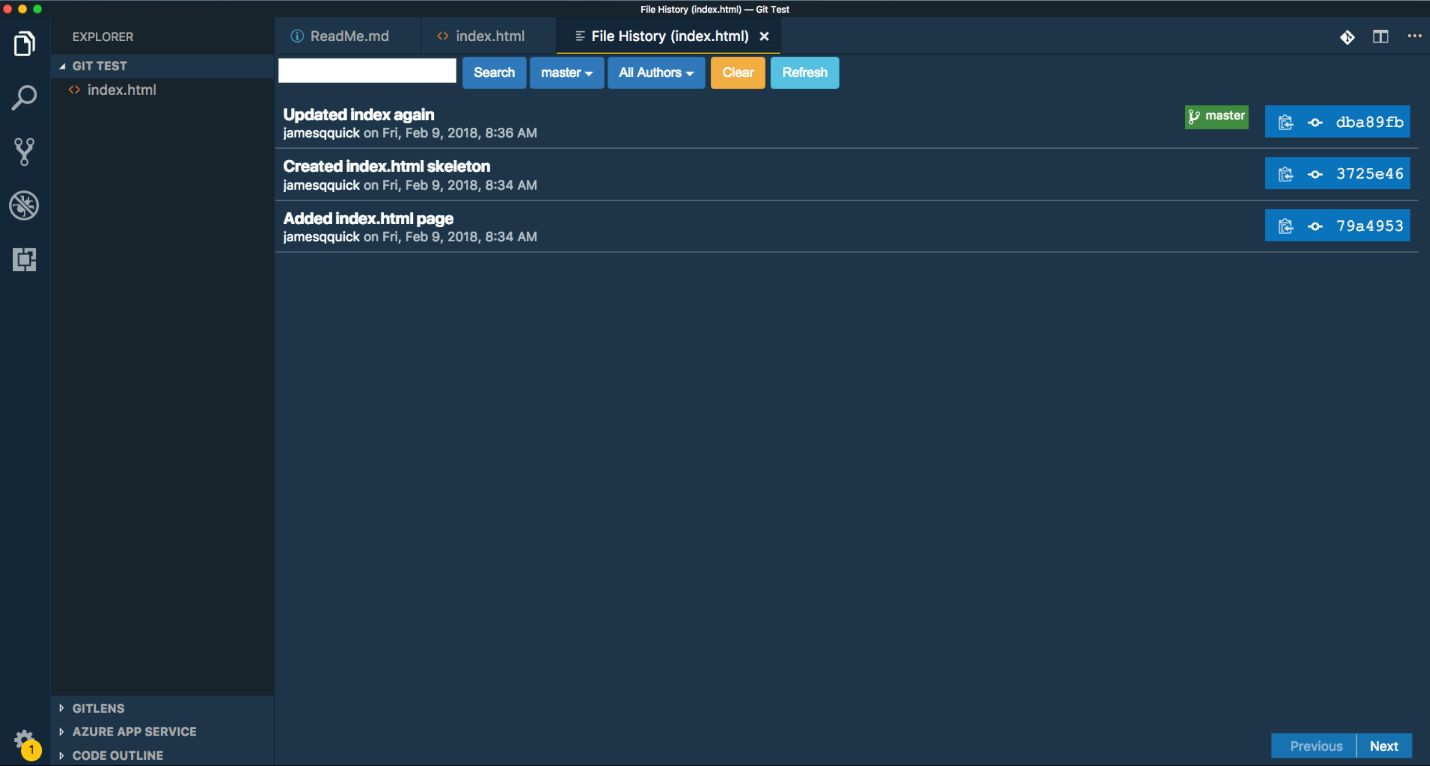


Рис. 27. Окно истории изменения файла.

Также вы сможете сравнивать ветви и записанные в репозиторий версии, создавать ветви из записанных версий и т. д.

[Git Lens](https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens)

GitLens дополняет возможности Git, встроенные в Visual Studio Code. Это расширение помогает визуализировать принадлежность кода через аннотации Git Blame и линзу кода, просматривать и изучать репозитории Git из среды VS Code, получать полезные аналитические данные с помощью мощных команд сравнения, а также выполнять многие другие задачи.

Расширение Git Lens — одно из самых мощных и популярных среди сообщества разработчиков расширений. В большинстве случаев его функции могут заменить каждое из вышеперечисленных двух расширений.

В правой части текущей строки, над которой вы работаете, отображается небольшое сообщение о том, кто внес изменение, когда это было сделано, а также сообщение о записи изменения в репозиторий. При наведении курсора на это сообщение выводится всплывающий блок с дополнительной информацией, включая само изменение кода, временную метку и т. д.

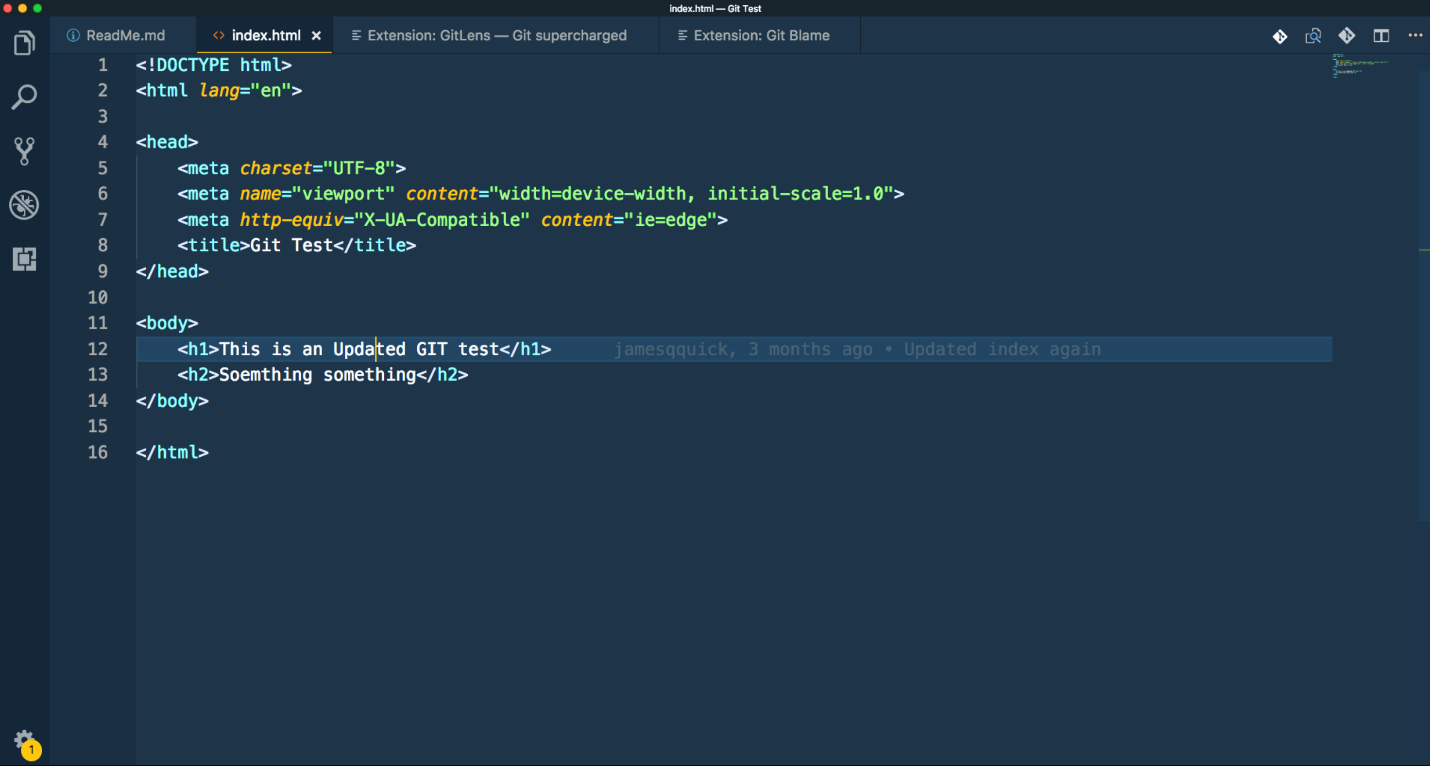
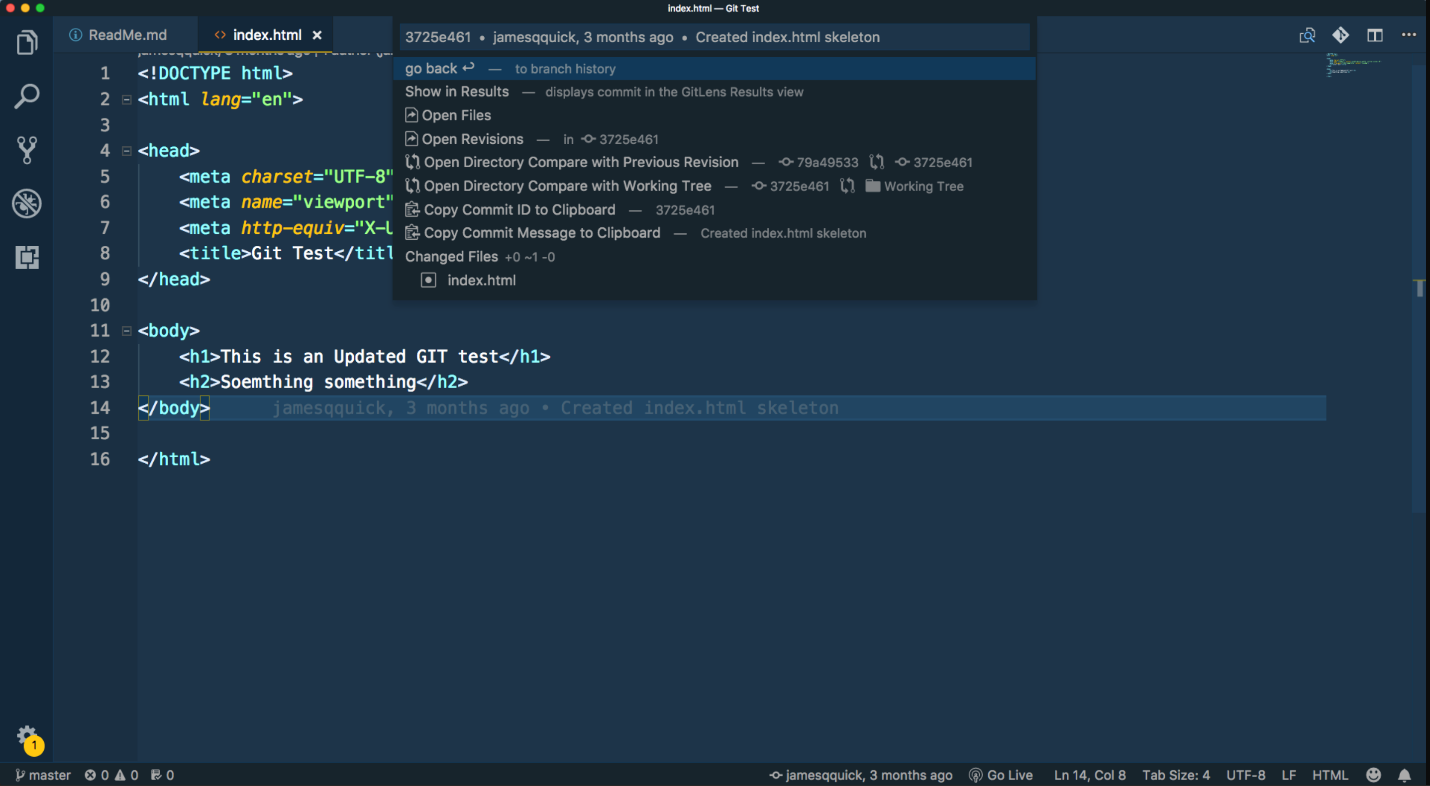


Рис. 27. Окно редактирования кода с дополнением Git Lens.

Также данное расширение предоставляет много функций, связанных с историей Git. Вы можете легко получить доступ к разнообразной информации, включая историю файлов, сравнение с предыдущими версиями, открытие определенных редакций и т. д. Чтобы открыть эти опции, вы можете нажать на текст на нижней панели состояния, где указан автор, изменивший строку кода, а также время ее изменения.

При этом откроется следующее окно (рис. 28):

 Рис. 27. Окно опций дополнения Git Lens.

Это расширение имеет очень много функций, и потребуется время, чтобы разобраться со всеми открываемыми им возможностями.

# **Заключение**

В этой статье мы рассмотрели ключевые аспекты систем контроля версий, с особым акцентом на Git и его развитие. Понимание истории и архитектуры Git позволяет глубже оценить его преимущества и понять, почему этот инструмент стал стандартом в индустрии разработки программного обеспечения. Освоение базовых команд и принципов работы с Git предоставляет разработчикам мощные средства для управления проектами, облегчая процесс отслеживания изменений, ведения ветвления и слияния кода.

Кроме того, интеграция Git с платформой GitHub и различными IDE значительно упрощает совместную работу и управление проектами. GitHub предоставляет дополнительные инструменты и сервисы, которые способствуют эффективной командной работе и автоматизации рабочих процессов. Внедрение Git и GitHub в повседневную практику разработки не только повышает продуктивность, но и способствует созданию более надежного и качественного программного обеспечения.

# **Список литературы**

1. https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-ru#5

2. https://git-scm.com/book/ru/v2/%D0%9F%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5-A%3A-Git-%D0%B2-%D0%B4%D1%80%D1%83%D0%B3%D0%B8%D1%85-%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F%D1%85-Git-%D0%B2-Visual-Studio-Code

3. https://devpractice.ru/git-for-beginners-part-1-what-is-vcs/

4. https://htmlacademy.ru/blog/git/git-console

5. https://www.atlassian.com/ru/git/tutorials/git-bash

6. https://fructcode.com/ru/blog/git-and-bitbucket-quick-start/