Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра инженерной психологии и эргономики

**Пользовательские интерфейсы информационных систем**

Отчет по практическим занятиям на тему

«Образовательный курс GitHowTo»

|  |  |
| --- | --- |
| Выполнила: | Проверил: |
| cтудентка гр. 210901  Асташко В. И | Давыдович К. И. |
|  |  |

Минск 2024

**Цель:** сформировать понимание, изучить специфику работы с инструментом контроля версий «Git» и освоить его основный функционал.

**Прохождение курса**

**0. Подготовка**

**Цели**

* Установить материалы учебника и подготовить их к работе.

1. **Скачайте учебные материалы**

Скачайте учебные материалы здесь:

* <https://githowto.com/githowto.zip>

1. **Распакуйте учебные материалы**

Пакет учебных материалов должен иметь главную директорию githowto с двумя поддиректориями:

* repositories: пустая директория, в которой будут располагаться ваши репозитории.
* files: заранее упакованные файлы для того, чтобы вы могли продолжить работать с учебными материалами на любом этапе. Если вы застрянете, просто скопируйте нужный урок в свой репозиторий.

Пакет распакован по следующему пути:   
D:\univer\третий курс\семестр 1\пиис\pz\githowto

**Часть I: Основы Git**

**Цели**

* Понять, что такое Git и чем он полезен.

В течение следующих 30 уроков мы будем изучать основы работы с Git. Git — это **система контроля версий**, которая позволяет отслеживать изменения в вашем коде с течением времени. Это очень мощный инструмент, который сейчас используется большинством разработчиков программного обеспечения. Это также отличный способ сотрудничать с другими разработчиками над одним проектом.

**Что такое Git?**

Вы можете представить себе Git как «машину времени» для вашего кода. Он позволяет вам вернуться в прошлое и увидеть, как выглядел ваш код в определенный момент времени. Он также позволяет увидеть, кто и когда вносил те или иные изменения в код. Он даже позволяет отменять изменения, которые были внесены в ваш код ранее.

Git — это программа с текстовым интерфейсом, с которой надо работать в командной строке. Если вы никогда не работали с текстовыми программами или командной строкой, то сначала это все может выглядеть пугающе. Но не волнуйтесь, в этом курсе мы также рассмотрим, как пользоваться командной строкой. К тому же, большинство современных редакторов кода имеют встроенный Git-клиент, который позволяет легко взаимодействовать с Git с помощью графического интерфейса пользователя, минуя командную строку. Однако, все же полезно научиться работать с Git с помощью командной строки:

* Это даст вам лучшее понимание того, как работает Git.
* Это позволит вам использовать Git на любом компьютере, даже если на нем не установлены ваши любимые инструменты разработчика.
* Это позволяет вам использовать Git на удаленном сервере, который не имеет графического пользовательского интерфейса.

Прежде чем мы начнем, давайте рассмотрим некоторые термины, которые мы будем использовать в этом курсе.

**Терминология**

**Репозиторий**

Репозиторий Git — это хранилище, в котором расположен ваш проект и его история. Это может быть локальное хранилище где-то на вашем компьютере или удаленное хранилище на сервисе типа GitHub или другом хостинге в Интернете. Репозиторий служит для отслеживания изменений в проекте, координации работы между несколькими людьми и отслеживания истории проекта.

Скажем, у вас на компьютере есть директория со всеми файлами вашего проекта. Когда вы инициализируете репозиторий Git в этой директории, Git создает скрытую поддиректорию под названием .git, в которой хранится вся информация о репозитории. Эта информация включает историю всех изменений, внесенных в репозиторий, а также его текущее состояние.

Скучный факт №1:\*\* По умолчанию директория .git скрыта. Если вы хотите ее увидеть, убедитесь, что в вашем файловом менеджере включена опция показа скрытых файлов.

**Коммит**

Вы можете думать о коммите как о снимке вашего проекта в определенный момент времени. Правда, коммит содержит только информацию об изменениях, которые были внесены в репозиторий с момента последнего коммита. Он не содержит все файлы репозитория (если только это не первый коммит). Таким образом, каждый коммит — это небольшой кусочек истории репозитория, основанный на предыдущем коммите. Все они связаны между собой в цепочку, формируя историю изменений вашего проекта.

**Ветка**

Ветка — это параллельная версия репозитория. Ветки позволяют вам работать над отдельными функциями вашего проекта, не влияя на основную версию. Закончив работу над новой фичей, вы можете объединить эту ветку с основной версией проекта.

**Скучный факт №2:** В репозитории всегда есть по крайней мере одна ветка, даже если вы сами ее не создавали. Обычно ее называют веткой main (или master).

Теперь у вас есть базовое понимание того, что такое Git и как он работает. В следующем уроке мы наконец-то пощупаем команды Git своими руками.

**1. Финальные приготовления**

**Цели**

* Полная готовность к работе с Git.

1. **Установка имени и электронной почты**

Если вы никогда ранее не использовали Git, для начала вам необходимо осуществить установку. Выполните следующие команды, чтобы Git узнал ваше имя и электронную почту. Эти данные используются для подписи изменений, сделанных вами, что позволит отслеживать, кто и когда сделал изменения в файле.

**Выполните**

git config --global user.name "Your Name"

git config --global user.email "your\_email@whatever.com"

1. **Имя ветки по умолчанию**

Мы будем использовать main в качестве имени ветки по умолчанию. Чтобы установить его, выполните следующую команду:

**Выполните**

git config --global init.defaultBranch main

1. **Корректная обработка окончаний строк**

Для пользователей Windows:

**Выполните**

git config --global core.autocrlf true

git config --global core.safecrlf warn

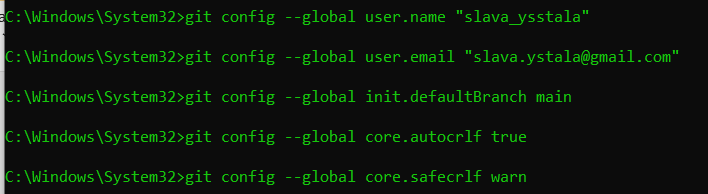


Рисунок 1 – выполнение заданий финальных приготовлений

**2. Создание проекта**

**Цели**

* Научиться создавать Git-репозиторий с нуля.

1. **Создайте страницу «Hello, World»**

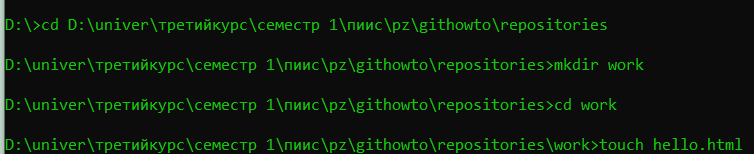
Начните работу в пустой директории (например, respositories, если вы скачали архив с предыдущего шага) с создания пустой поддиректории work, затем войдите в неё и создайте там файл hello.html с таким содержанием:

**Выполните**

mkdir work

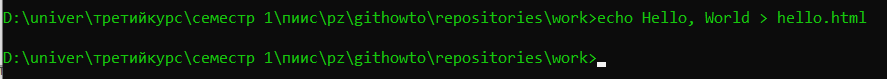
cd work

touch hello.html



**Файл: hello.html**

Hello, World



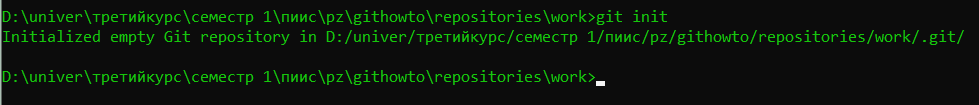
1. **Создайте репозиторий**

Теперь у вас есть директория с одним файлом. Чтобы создать Git-репозиторий из этой директории, выполните команду git init.

**Выполните**

git init

**Результат**

****

Initialized empty Git repository in /home/alex/githowto/repositories/work/.git/

1. **Добавьте страницу в репозиторий**

Теперь давайте добавим в репозиторий страницу «Hello, World».

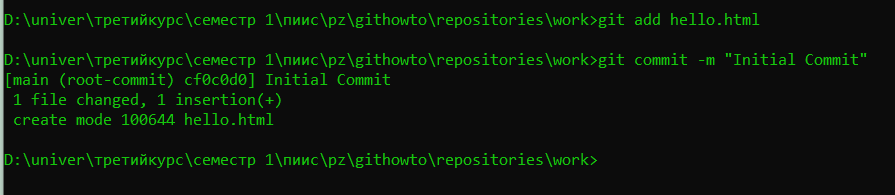
**Выполните**

git add hello.html

git commit -m "Initial Commit"

Вы увидите...

**Результат**

****

$ git add hello.html

$ git commit -m "Initial commit"

[main (root-commit) 5836970] Initial commit

1 file changed, 1 insertion(+)

create mode 100644 hello.html

**3. Проверка состояния**

**Цели**

* Научиться проверять состояние репозитория.

1. **Проверьте состояние репозитория**

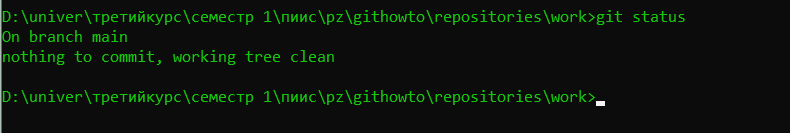
Используйте команду git status, чтобы проверить текущее состояние репозитория.

**Выполните**

git status

Вы увидите:

**Результат**

****

$ git status

On branch main

nothing to commit, working tree clean

Если после выполнения предыдущей команды вы видите On branch master вместо On branch main, это означает, что у вас немного устаревшая версия Git'а, которая не поняла нас, когда мы попросили установить имя ветки по умолчанию на main. В этом случае вы можете переименовать ветку в main с помощью следующей команды:

git branch -m master main

Команда проверки состояния сообщила, что коммитить нечего. Это означает, что в репозитории уже хранится текущее состояние рабочих файлов, и нет никаких изменений, которые могли бы ожидать записи.

Мы будем использовать команду git status, чтобы продолжать отслеживать состояние репозитория и рабочей директории.

**4. Внесение изменений**

**Цели**

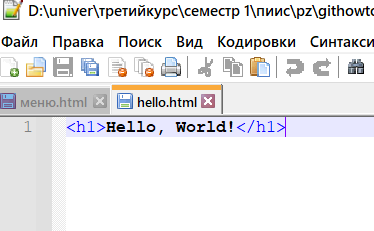
* Научиться отслеживать состояние рабочей директории.

1. **Измените страницу «Hello, World»**

Добавим кое-какие HTML-теги к нашему приветствию. Измените содержимое файла на:

**Файл: hello.html**

<h1>Hello, World!</h1>



1. **Проверьте состояние**

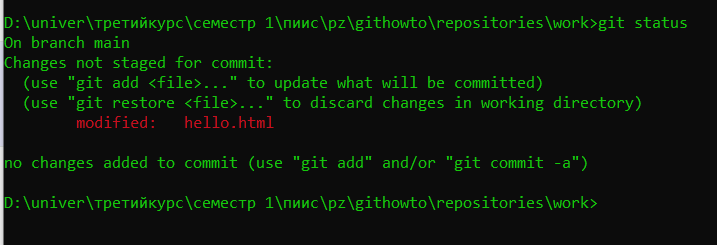
Теперь проверьте состояние рабочей директории.

**Выполните**

git status

Вы увидите...

**Результат**

****

$ git status

On branch main

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: hello.html

no changes added to commit (use "git add" and/or "git commit -a")

Первое, что нужно заметить, это то, что Git знает, что файл hello.html был изменен, но при этом эти изменения еще не зафиксированы в репозитории.

Также обратите внимание на то, что сообщение о состоянии дает вам подсказку о том, что нужно делать дальше. Если вы хотите добавить эти изменения в репозиторий, используйте команду git add. В противном случае используйте команду git сheckout для отмены изменений.

**5. Индексация изменений**

**Цели**

* Научиться индексировать изменения для последующих коммитов.

1. **Добавьте изменения**

Теперь дайте команду Git проиндексировать изменения. Проверьте состояние:

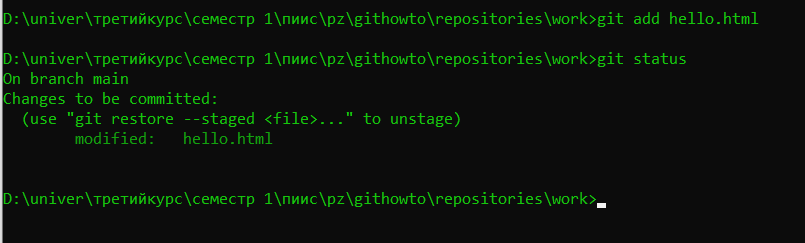
**Выполните**

git add hello.html

git status

Вы увидите:

**Результат**

****

$ git add hello.html

$ git status

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: hello.html

Изменения файла hello.html были проиндексированы. Это означает, что Git теперь знает об изменении, но изменение пока не *перманентно* (читай, *навсегда*) записано в репозиторий. Следующий коммит будет включать в себя проиндексированные изменения.

Если вы решили, что *не* хотите коммитить изменения, команда состояния напомнит вам о том, что с помощью команды git reset можно снять индексацию этих изменений.

**6. Индексация и коммит**

Отдельный шаг индексации в Git позволяет вам разделять большие изменения на маленькие коммиты. Аналогия: вы помыли машину и заодно залили жидкость для очистки стекла — эти два изменения по своей сути независимы, а потому лучше пометить их отдельно. В противном случае, в истории изменений бачка для жидкости очистки стекла будет запись "Помыл машину", что не соответствует сути изменения и может запутать того, кто потом будет разбираться в этой истории.

Предположим, что вы отредактировали три файла (a.html, b.html, и c.html). Теперь вы хотите закоммитить все изменения, при этом чтобы изменения в a.html и b.html были одним коммитом, в то время как изменения в c.html логически не связаны с первыми двумя файлами и должны идти отдельным коммитом.

В теории, вы можете сделать следующее:

git add a.html

git add b.html

git commit -m "Changes for a and b"

git add c.html

git commit -m "Unrelated change to c"

Разделяя индексацию и коммит, вы имеете возможность с легкостью настроить, что идет в какой коммит.

**7. Коммит изменений**

**Цели**

* Научиться коммитить изменения в репозиторий.

1. **Закоммитьте изменения**

Достаточно об индексации. Давайте сделаем коммит того, что мы проиндексировали, в репозиторий.

Когда вы ранее использовали git commit для коммита первоначальной версии файла hello.html в репозиторий, вы включили метку -m, которая делает комментарий в командной строке. Команда commit позволит вам интерактивно редактировать комментарии для коммита. Теперь давайте это проверим.

Если вы опустите метку -m из командной строки, Git перенесет вас в редактор по вашему выбору. Редактор выбирается из следующего списка (в порядке приоритета):

* переменная среды GIT\_EDITOR
* параметр конфигурации core.editor
* переменная среды VISUAL
* переменная среды EDITOR

У меня переменная EDITOR установлена в vim. Если вы предпочитаете GUI-редактор, то теперь можно использовать VS Code в качестве [Git-редактора](https://code.visualstudio.com/docs/sourcecontrol/overview" \l "_vs-code-as-git-editor).

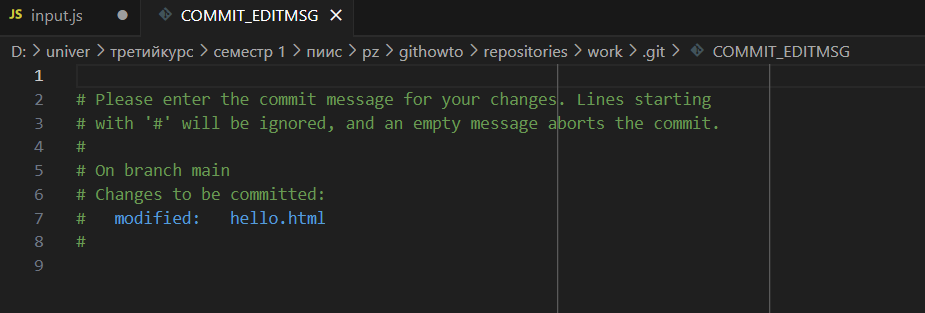
Сделайте коммит сейчас и проверьте состояние.

**Выполните**

git commit

Вы увидите в вашем редакторе:

**Результат**

****

|

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

#

# On branch main

# Changes to be committed:

# modified: hello.html

#

В первой строке введите комментарий: Added h1 tag. Сохраните файл и выйдите из редактора (для этого в редакторе по умолчанию (Vim) вам нужно нажать клавишу ESC, ввести :wq и нажать Enter). Вы увидите:

**Результат**

$ git commit

[main 78433de] Added h1 tag

1 file changed, 1 insertion(+), 1 deletion(-)

Строка «Waiting for Emacs...» получена из программы emacsclient, которая посылает файл в запущенную программу emacs и ждет его закрытия. Остальные выходные данные – стандартные коммит-сообщения.

1. **Проверьте состояние**

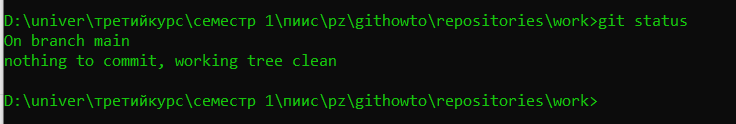
В конце давайте еще раз проверим состояние.

**Выполните**

git status

Вы увидите:

**Результат**

****

$ git status

On branch main

nothing to commit, working tree clean

Рабочая директория чиста, можем продолжить работу.

**8. Изменения, а не файлы**

**Цели**

* Понять, что Git работает с изменениями, а не файлами.

Большинство систем контроля версий работает с файлами: вы добавляете файл в систему, и она отслеживает изменения файла с этого момента.

Git фокусируется на изменениях в файле, а не самом файле. Когда вы осуществляете команду git add file, вы не говорите Git добавить файл в репозиторий. Скорее вы говорите, что Git надо отметить текущее состояние файла, коммит которого будет произведен позже.

Мы попытаемся исследовать эту разницу в данном уроке.

1. **Первое изменение: Добавьте стандартные теги страницы**

Измените страницу «Hello, World», чтобы она содержала стандартные теги <html> и <body>.

**Файл: hello.html**

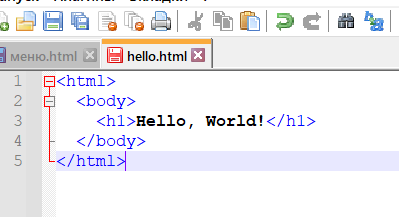
<html>

<body>

<h1>Hello, World!</h1>

</body>

</html>



1. **Добавьте это изменение**

Теперь добавьте это изменение в индекс Git.

**Выполните**

git add hello.html

1. **Второе изменение: Добавьте заголовки HTML**

Теперь добавьте заголовки HTML (секцию <head>) к странице «Hello, World».

**Файл: hello.html**

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

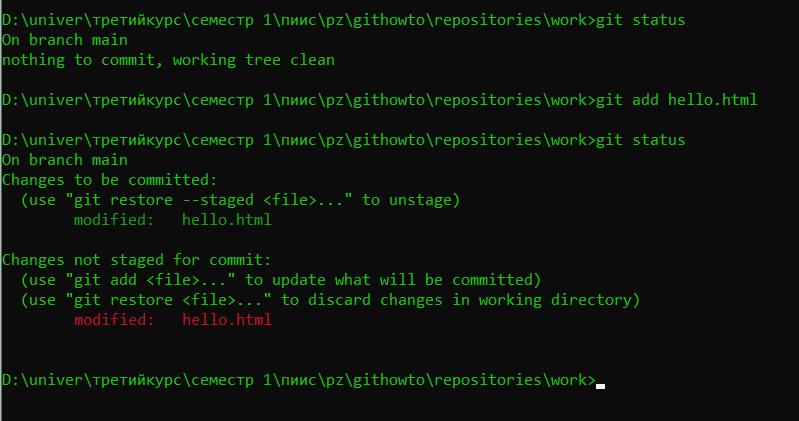
1. **Проверьте текущий статус**

**Выполните**

git status

Вы увидите:

**Результат**

****

$ git status

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: hello.html

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: hello.html

Обратите внимание на то, что hello.html указан дважды в состоянии. Первое изменение (добавление стандартных тегов) проиндексировано и готово к коммиту. Второе изменение (добавление заголовков HTML) является непроиндексированным. Если бы вы делали коммит сейчас, заголовки не были бы сохранены в репозиторий.

Давайте проверим.

1. **Коммит**

Произведите коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверьте состояние.

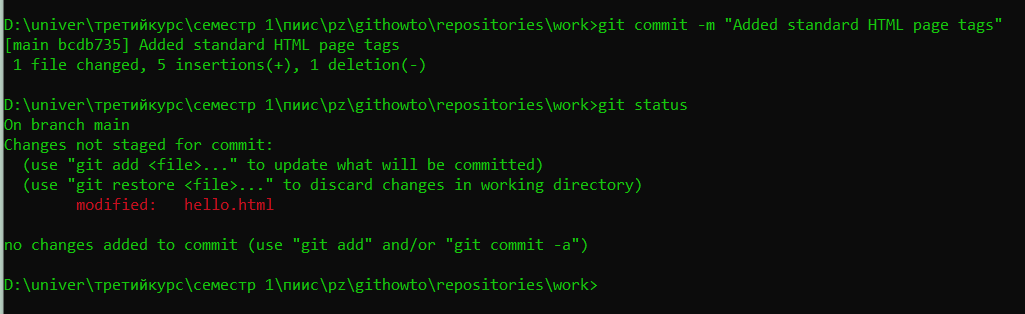
**Выполните**

git commit -m "Added standard HTML page tags"

git status

Вы увидите:

**Результат**

****

$ git commit -m "Added standard HTML page tags"

[main 46afaff] Added standard HTML page tags

1 file changed, 5 insertions(+), 1 deletion(-)

$ git status

On branch main

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: hello.html

no changes added to commit (use "git add" and/or "git commit -a")

Команда status показывает, что в файле hello.html ещё есть незаписанные изменения, но область подготовки уже пуста.

1. **Добавьте второе изменение**

Теперь добавьте второе изменение в индекс, а затем проверьте состояние с помощью команды git status.

**Выполните**

git add .

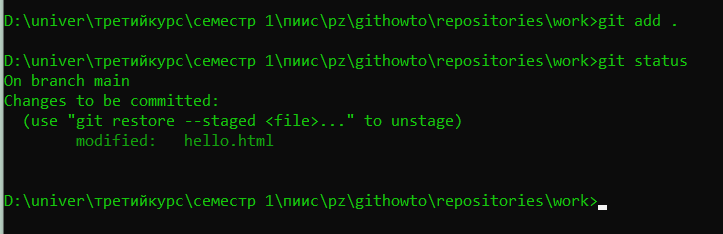
git status

Мы использовали текущую директорию (.) в качестве аргумента для добавления. Это самый короткий и удобный способ добавления всех изменений в текущей директории. Но поскольку Git добавляет в индекс *всё*, то *не лишним* будет проверить состояние репозитория перед запуском add, просто чтобы убедиться, что вы не добавили какой-то файл, который не следовало бы добавлять.

Я просто хотел показать вам трюк с add ., в дальнейшем мы будем добавлять все файлы явно.

Вы увидите:

**Результат**

****

$ git add .

$ git status

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: hello.html

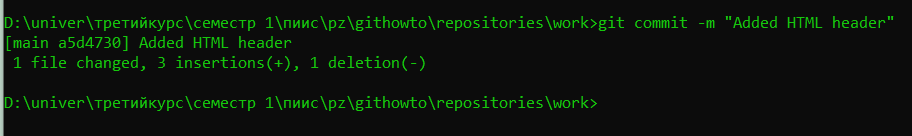
Второе изменение было проиндексировано и готово к коммиту.

1. **Сделайте коммит второго изменения**

**Выполните**

git commit -m "Added HTML header"

**Результат**

****

$ git commit -m "Added HTML header"

[main b7614c1] Added HTML header

1 file changed, 2 insertions(+)

**9. История**

**Цели**

* Научиться просматривать историю проекта.

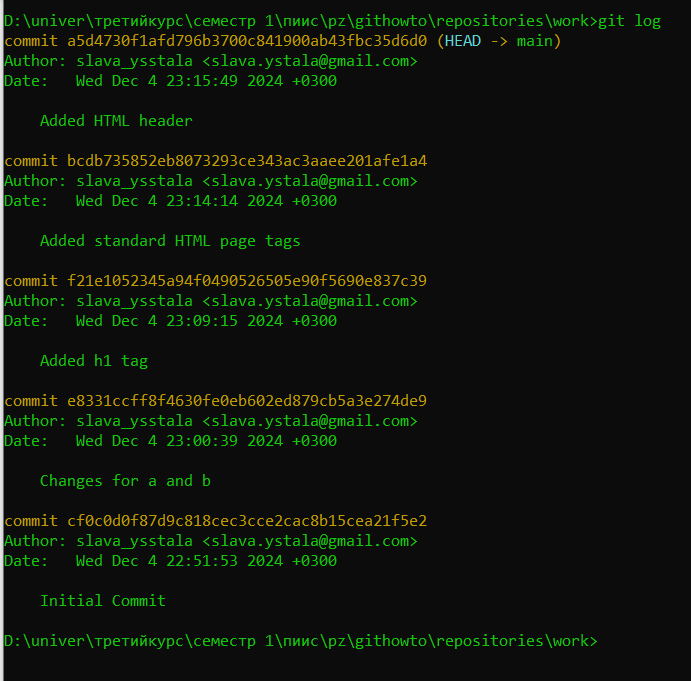
Получение списка произведенных изменений — функция команды git log.

**Выполните**

git log

Вы увидите:

**Результат**

****

$ git log

commit b7614c1aea1ffbc46400fe1a163842d6ec620a43

Author: Alexander Shvets <alex@githowto.com>

Date: Tue Nov 28 05:51:38 2023 -0600

Added HTML header

commit 46afaff2232fc3d564c40f65cb82e7e94839a1bb

Author: Alexander Shvets <alex@githowto.com>

Date: Tue Nov 28 05:51:38 2023 -0600

Added standard HTML page tags

commit 78433de967102f2b59d0a8a60eb397b2663ed282

Author: Alexander Shvets <alex@githowto.com>

Date: Tue Nov 28 05:51:38 2023 -0600

Added h1 tag

commit 58369706affbc1c27fa03a65fc7a05847278045f

Author: Alexander Shvets <alex@githowto.com>

Date: Tue Nov 28 05:51:38 2023 -0600

Initial commit

Вот список всех четырех коммитов в репозиторий, которые мы успели совершить.

* 1. **Однострочная история**

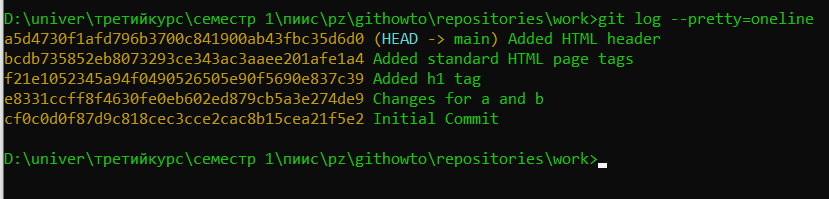
Вы полностью контролируете то, что отображает log. Мне, например, нравится однострочный формат:

**Выполните**

git log --pretty=oneline

Вы увидите:

**Результат**

****

$ git log --oneline

b7614c1 Added HTML header

46afaff Added standard HTML page tags

78433de Added h1 tag

5836970 Initial commit

* 1. **Контроль отображения записей**

Вот еще интересные варианты просмотра истории:

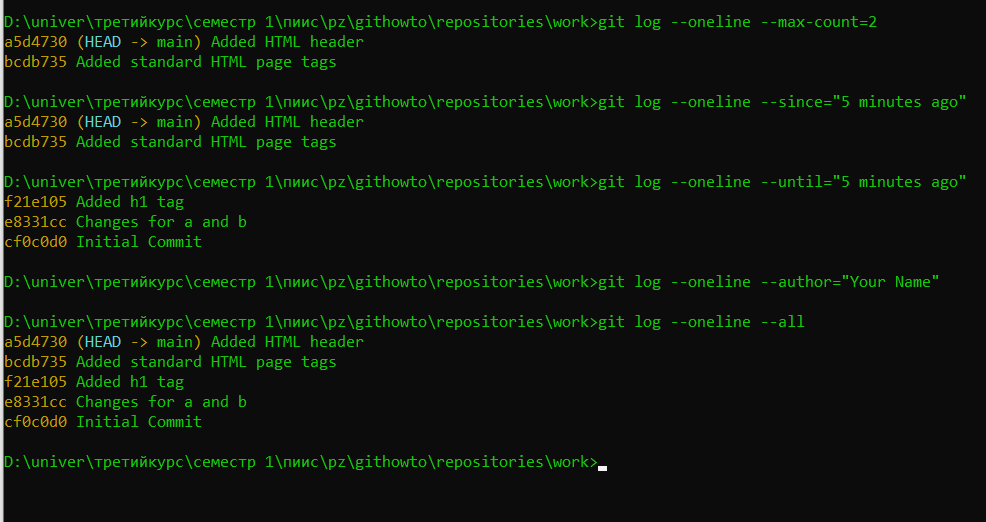
git log --oneline --max-count=2

git log --oneline --since="5 minutes ago"

git log --oneline --until="5 minutes ago"

git log --oneline --author="Your Name"

git log --oneline --all

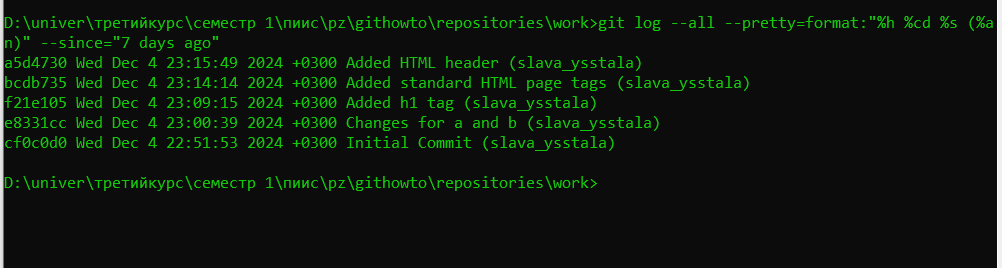


Существует огромное количество вариантов просмотра истории, вы можете порыться на странице руководства [git-log](https://git-scm.com/docs/git-log), чтобы увидеть их все.

* 1. **Изощряемся**

Вот что я использую для просмотра изменений, сделанных за последнюю неделю. Я добавлю --author=Alexander, если я хочу увидеть только изменения, которые сделал я.

git log --all --pretty=format:"%h %cd %s (%an)" --since="7 days ago"



* 1. **Конечный формат лога**

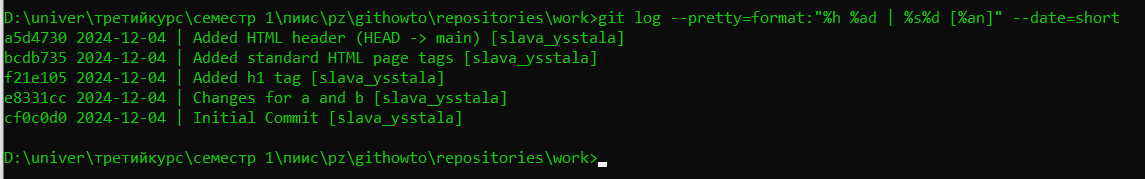
Со временем, я решил, что для большей части моей работы мне подходит следующий формат лога.

**Выполните**

git log --pretty=format:"%h %ad | %s%d [%an]" --date=short

Выглядит это примерно так:

**Результат**

****

$ git log --pretty=format:"%h %ad | %s%d [%an]" --date=short

b7614c1 2023-11-28 | Added HTML header (HEAD -> main) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

Давайте рассмотрим его в деталях:

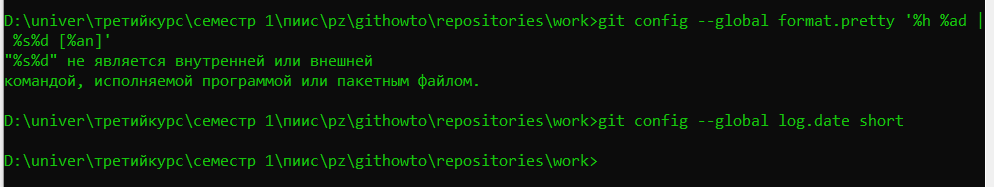
* --pretty="..." — определяет формат вывода.
* %h — укороченный хеш коммита.
* %ad — дата коммита.
* | — просто визуальный разделитель.
* %s — комментарий.
* %d — дополнения коммита («головы» веток или теги).
* %an — имя автора.
* --date=short — сохраняет формат даты коротким и симпатичным.

Таким образом, каждый раз, когда вы захотите посмотреть лог, вам придется много печатать. К счастью, существует несколько опций конфигурации Git, позволяющих настроить формат вывода истории по умолчанию:

**Выполните**

git config --global format.pretty '%h %ad | %s%d [%an]'

git config --global log.date short



* 1. **Другие инструменты**

Оба gitx (для Mac) и gitk (для любой платформы) полезны в изучении истории изменений.

**10. Получение старых версий**

**Цели**

* Научиться возвращать рабочую директорию к любому предыдущему состоянию.

Git позволяет очень просто путешествовать во времени, по крайней мере, для вашего проекта. Команда checkout обновит вашу рабочую директорию до любого предыдущего коммита.

1. **Получите хеши предыдущих коммитов**

**Выполните**

git log

**Результат**

$ git log

b7614c1 2023-11-28 | Added HTML header (HEAD -> main) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

Просмотрите историю изменений и найдите хеш первого коммита. Он должен быть в последней строке результата git log. Используйте этот хеш (достаточно первых 7 символов) в команде ниже. Затем проверьте содержимое файла hello.html.

**Выполните**

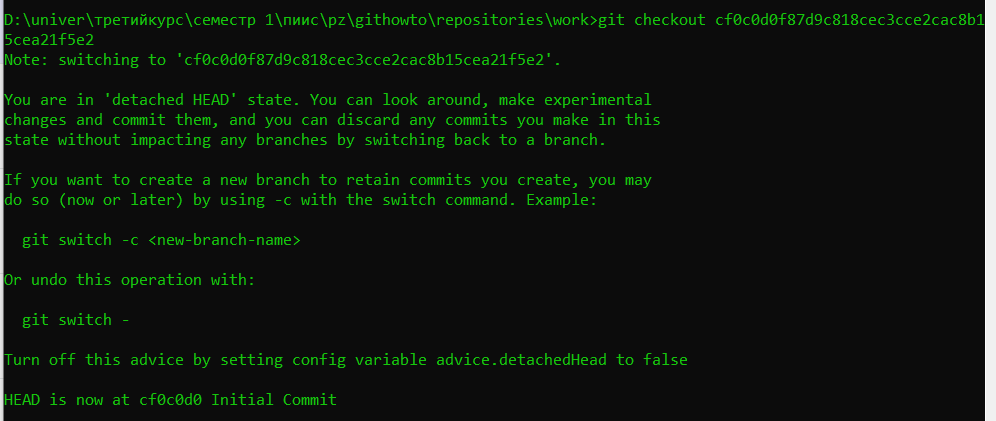
git checkout cf0c0d0f87d9c818cec3cce2cac8b15cea21f5e2

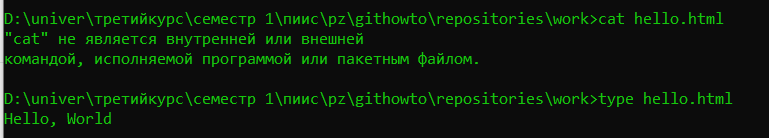
cat hello.html

Многие команды Git принимают хеши коммитов в качестве аргументов. Хеши коммитов будут отличаться в разных репозиториях, поэтому когда вы видите, что в команде есть пометка <hash>, то это значит, что вам надо подставить вместо нее реальный хеш из вашего репозитория.

Вы увидите:

**Результат**

****

****

$ git checkout 5836970

Note: switching to '5836970'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 5836970 Initial commit

$ cat hello.html

Hello, World!

Обратите внимание, что сейчас содержимое файла hello.html — это тот самый текст, с которого мы начинали.

1. **Вернитесь к последней версии в ветке**main

Чтобы вернуться к последней версии нашего кода, нам нужно переключиться на ветку по умолчанию, main. Для переключения между ветками можно воспользоваться командойswitch.

Команда checkout в течение длительного времени была своеобразным швейцарским ножом в мире Git. Она имеет множество различных опций, которые позволяют выполнять совершенно разные вещи: переключать ветки, сбрасывать код и так далее. В какой-то момент команда Git решила разделить ее на несколько команд. Командаswitch является одной из них — ее единственным назначением является переключение между ветками. Команда checkout все еще доступна, но использовать ее для переключения веток больше не рекомендуется.

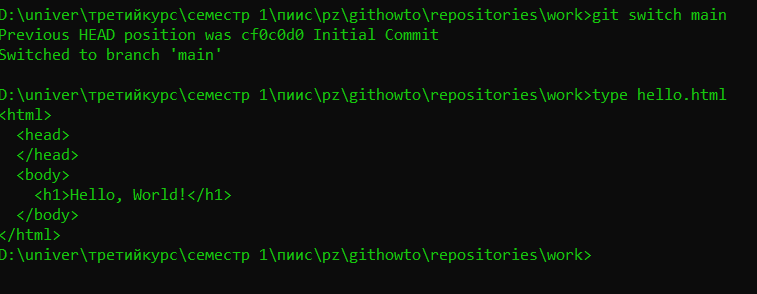
**Выполните**

git switch main

cat hello.html

Вы увидите:

**Результат**

****

$ git switch main

Previous HEAD position was 5836970 Initial commit

Switched to branch 'main'

$ cat hello.html

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

main — имя ветки по умолчанию. Переключаясь на ветку, вы попадаете на её последнюю версию.

**11. Создание тегов версий**

**Цели**

* Узнать, как создавать теги для коммитов для использования в будущем.

Думаю, вы согласитесь, что работать с хешами коммитов напрямую просто неудобно. Разве не было бы здорово, если бы вы могли обозначать конкретные коммиты понятными для человека названиями? Таким образом, вы могли бы четко видеть важные вехи в истории проекта. Кроме того, вы могли бы легко перейти к определенной версии проекта по ее названию. Именно для этого в Git придумали теги.

Давайте назовем текущую версию страницы hello.html первой, то есть v1.

1. **Создайте тег первой версии**

**Выполните**

git tag v1

git log

**Результат**

****

$ git tag v1

$ git log

b7614c1 2023-11-28 | Added HTML header (HEAD -> main, tag: v1) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

Теперь текущая версия страницы называется v1.

1. **Теги для предыдущих версий**

Обозначим версию, предшествующую текущей, названием v1-beta. Прежде всего, мы переключимся на предыдущую версию. Вместо того чтобы искать хеш коммита, мы будем использовать обозначение ^, а именно v1^, указывающее на коммит, предшествующий v1.

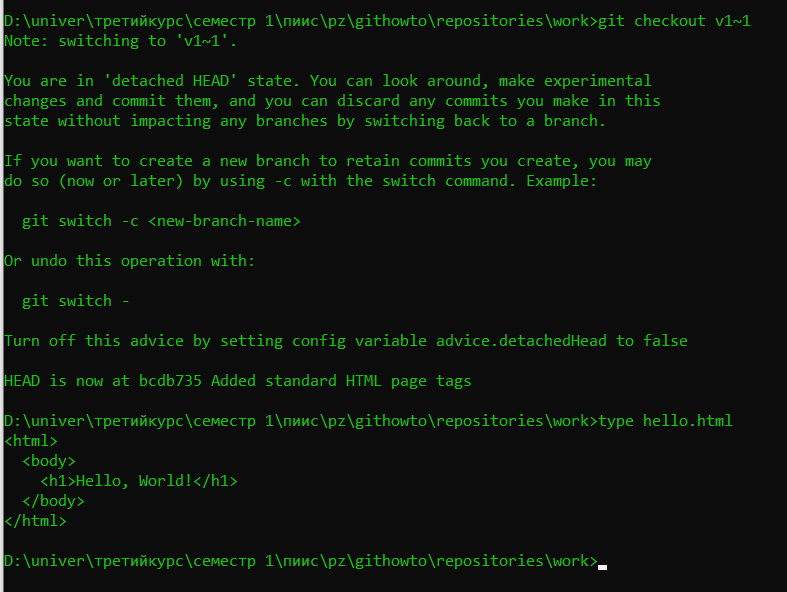
Если обозначение v1^ вызывает у вас какие-то проблемы, попробуйте также v1~1, указывающее на ту же версию. Это обозначение можно определить как «первую версию, предшествующую v1».

**Выполните**

git checkout v1~1

type hello.html

**Результат**

****

$ git checkout v1^

Note: switching to 'v1^'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 46afaff Added standard HTML page tags

$ cat hello.html

<html>

<body>

<h1>Hello, World!</h1>

</body>

</html>

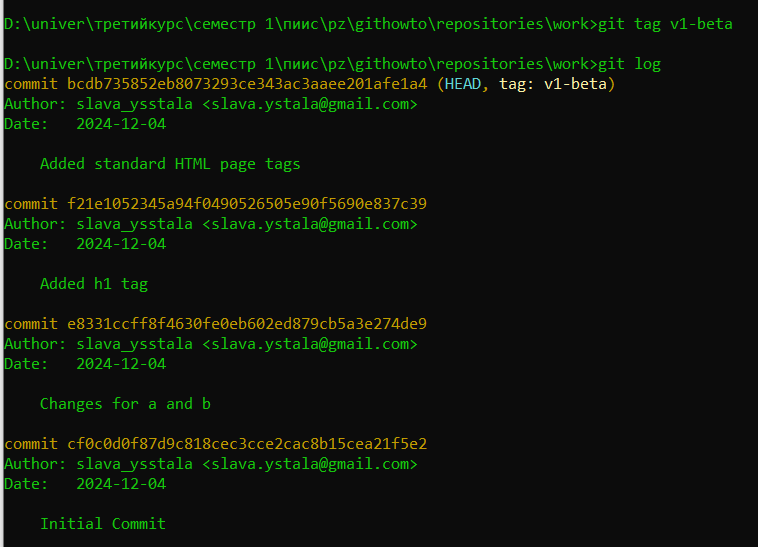
Это версия с тегами <html> и <body>, но еще *пока* без <head>. Давайте сделаем ее версией v1-beta.

**Выполните**

git tag v1-beta

git log

**Результат**

****

$ git tag v1-beta

$ git log

46afaff 2023-11-28 | Added standard HTML page tags (HEAD, tag: v1-beta) [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

1. **Переключение по имени тега**

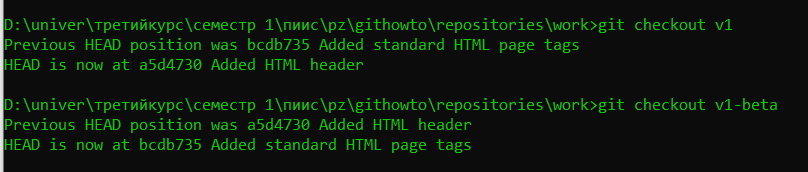
Теперь попробуйте попереключаться между двумя отмеченными версиями.

**Выполните**

git checkout v1

git checkout v1-beta

**Результат**

****

$ git checkout v1

Previous HEAD position was 46afaff Added standard HTML page tags

HEAD is now at b7614c1 Added HTML header

$ git checkout v1-beta

Previous HEAD position was b7614c1 Added HTML header

HEAD is now at 46afaff Added standard HTML page tags

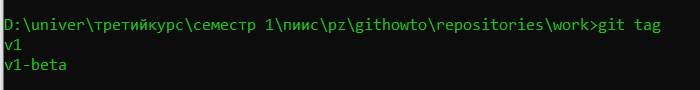
1. **Просмотр тегов с помощью команды**tag

Вы можете увидеть, какие теги доступны, используя команду git tag.

**Выполните**

git tag

**Результат**

****

$ git tag

v1

v1-beta

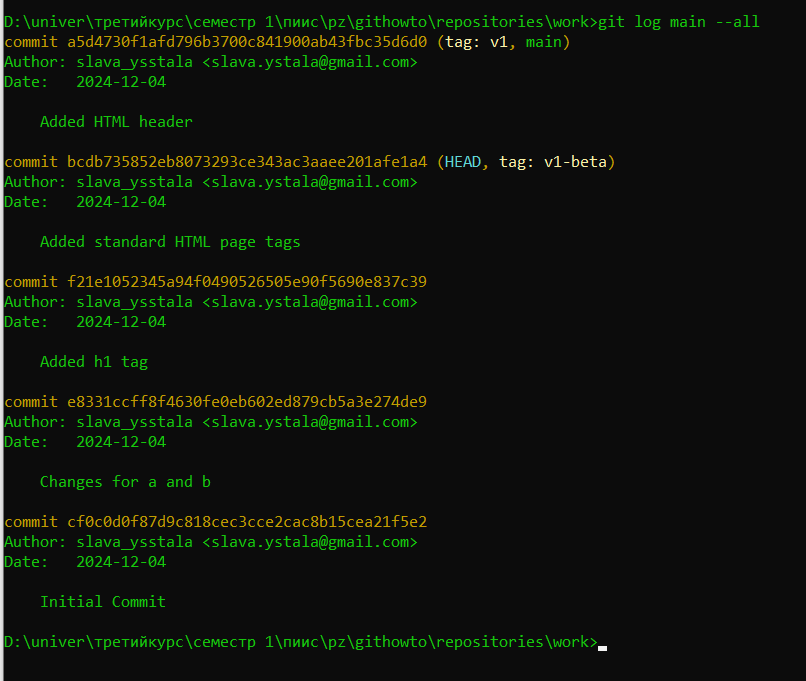
1. **Просмотр Тегов в логах**

Вы также можете посмотреть теги в логе.

**Выполните**

git log main --all

**Результат**

****

$ git log main --all

b7614c1 2023-11-28 | Added HTML header (tag: v1, main) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (HEAD, tag: v1-beta) [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

Вы можете видеть теги (v1 и v1-beta) в логе вместе с именем ветки (main). Кроме того, метка HEAD показывает коммит, на который вы переключились (на данный момент это v1-beta).

**12. Отмена локальных изменений (до индексации)**

**Цели**

* Научиться отменять изменения в рабочей директории.

1. **Переключитесь на ветку**main

Убедитесь, что вы находитесь на последнем коммите ветки main, прежде чем продолжить работу.

**Выполните**

git switch main

1. **Измените**hello.html

Иногда после того как вы изменили файл в рабочей директории, вы передумали и хотите просто вернуться к тому, что уже было закоммичено. Команда checkout справится с этой задачей.

Внесите изменение в файл hello.html в виде нежелательного комментария.

**Файл: hello.html**

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

<!-- This is a bad comment. We want to revert it. -->

</body>

</html>

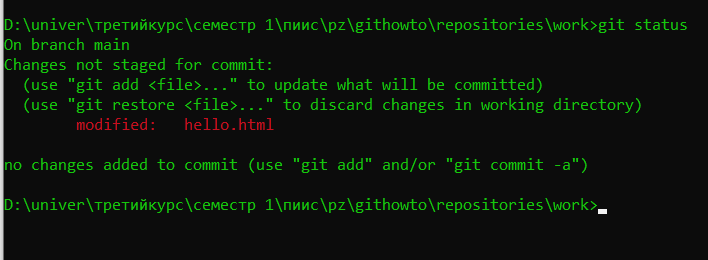
1. **Проверьте состояние**

Сначала проверьте состояние рабочей директории.

**Выполните**

git status

**Результат**

****

$ git status

On branch main

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: hello.html

no changes added to commit (use "git add" and/or "git commit -a")

Мы видим, что файл hello.html был изменен, но еще не проиндексирован.

1. **Отмена изменений в рабочем каталоге**

Используйте команду checkout для переключения в версию файла hello.html в репозитории.

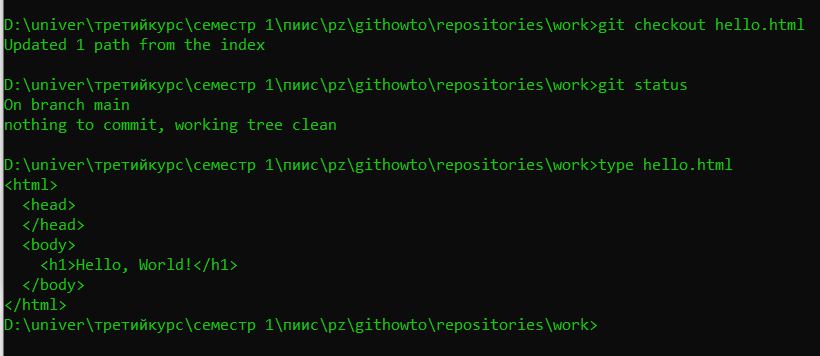
**Выполните**

git checkout hello.html

git status

type hello.html

**Результат**

****

$ git checkout hello.html

Updated 1 path from the index

$ git status

On branch main

nothing to commit, working tree clean

$ cat hello.html

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

Команда status показывает нам, что в рабочей директории не было сделано никаких незафиксированных изменений. И «нежелательный комментарий» больше не является частью содержимого файла.

**13. Отмена проиндексированных изменений (перед коммитом)**

**Цели**

* Научиться отменять изменения, которые были проиндексированы.

1. **Измените файл и проиндексируйте изменения**

Внесите изменение в файл hello.html в виде нежелательного комментария

**Файл: hello.html**

<html>

<head>

<!-- This is an unwanted but staged comment -->

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

Проиндексируйте это изменение.

**Выполните**

git add hello.html

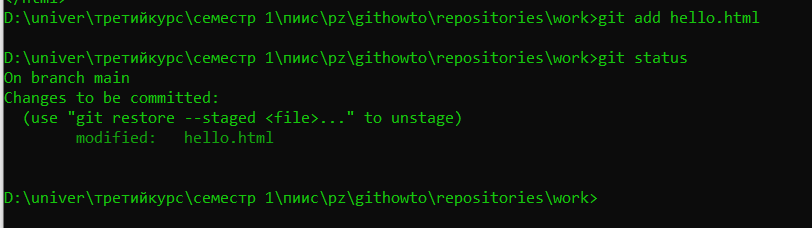
1. **Проверьте состояние**

Проверьте состояние нежелательного изменения.

**Выполните**

git status

**Результат**

****

$ git status

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: hello.html

Состояние показывает, что изменение было проиндексировано и готово к коммиту.

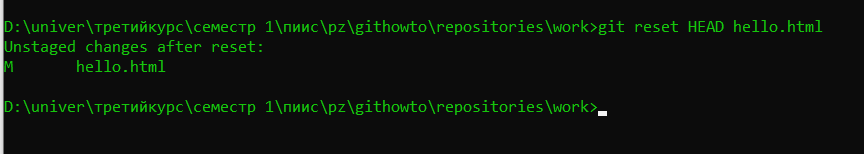
1. **Сбросьте области подготовки**

Команда reset сбрасывает область подготовки к HEAD. Это очищает область подготовки от изменений, которые мы только что проиндексировали.

**Выполните**

git reset HEAD hello.html

**Результат**

****

$ git reset HEAD hello.html

Unstaged changes after reset:

M hello.html

Команда reset (по умолчанию) не изменяет рабочую директорию. Поэтому рабочая директория всё еще содержит нежелательный комментарий. Мы можем использовать команду checkout из предыдущего урока, чтобы убрать нежелательные изменения в рабочей директории.

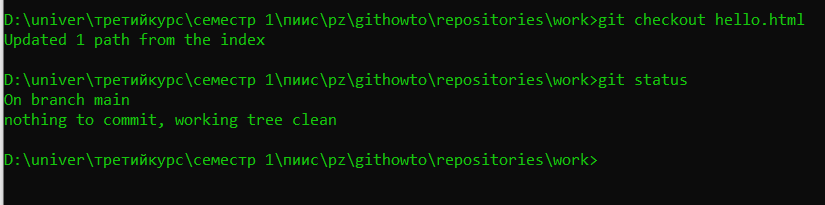
1. **Переключитесь на версию коммита**

**Выполните**

git checkout hello.html

git status

**Результат**

****

$ git checkout hello.html

Updated 1 path from the index

$ git status

On branch main

nothing to commit, working tree clean

Наша рабочая директория опять чиста.

**14. Отмена коммитов**

**Цели**

* Научиться отменять коммиты в локальном репозитории.

1. **Отмена коммитов**

Иногда вы понимаете, что новые коммиты являются неверными, и хотите их отменить. Есть несколько способов решения этого вопроса, здесь мы будем использовать самый безопасный.

Мы отменим коммит путем создания нового коммита, отменяющего нежелательные изменения.

1. **Измените файл и сделайте коммит**

Измените файл hello.html на следующий.

**Файл: hello.html**

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

<!-- This is an unwanted but committed change -->

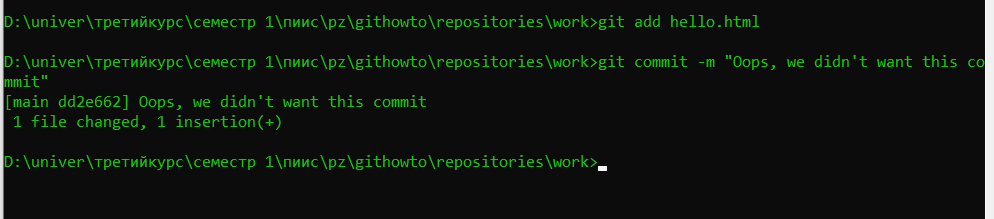
</body>

</html>

**Выполните**

git add hello.html

git commit -m "Oops, we didn't want this commit"



1. **Сделайте коммит с новыми изменениями, отменяющими предыдущие**

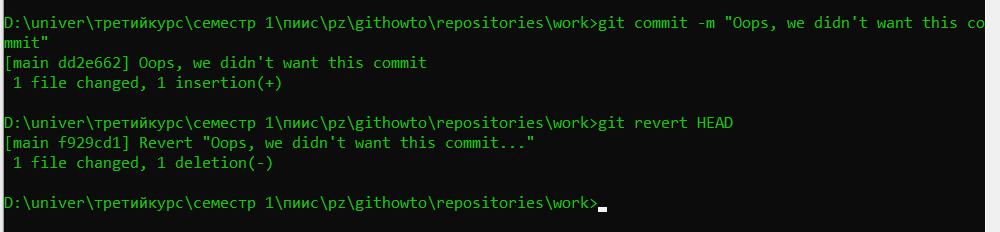
Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом.

**Выполните**

git revert HEAD

Перейдите в редактор, где вы можете отредактировать коммит-сообщение по умолчанию или оставить все как есть. Сохраните и закройте файл. Вы увидите:

**Результат**

****

$ git revert HEAD

[main 86364a1] Revert "Oops, we didn't want this commit"

Date: Tue Nov 28 05:51:38 2023 -0600

1 file changed, 1 deletion(-)

Так как мы отменили самый последний произведенный коммит, мы смогли использовать метку HEAD в качестве аргумента для отмены коммита. Мы можем отменить любой произвольной коммит в истории, указав его хеш.

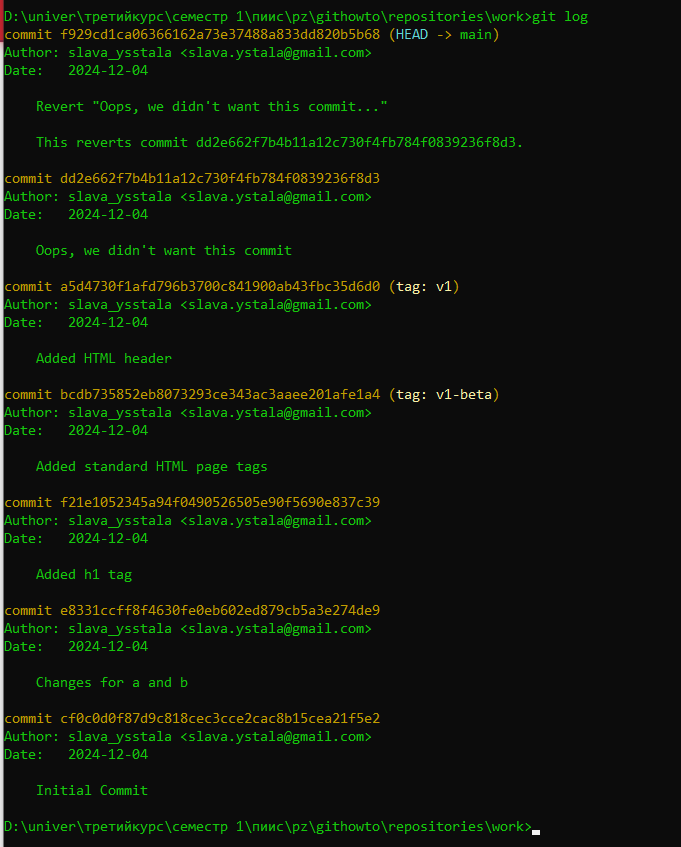
1. **Проверьте лог**

Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий.

**Выполните**

git log

**Результат**

****

$ git log

86364a1 2023-11-28 | Revert "Oops, we didn't want this commit" (HEAD -> main) [Alexander Shvets]

6a44bec 2023-11-28 | Oops, we didn't want this commit [Alexander Shvets]

b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

Эта техника будет работать с любым коммитом (хотя, возможно, возникнут конфликты). Она безопасна в использовании даже в публичных ветках удаленных репозиториев.

1. **Далее**

Далее давайте посмотрим на технику, которая может быть использована для удаления последних коммитов из истории репозитория.

**15. Удаление коммитов из ветки**

**Цели**

* Научиться удалять самые последние коммиты из ветки.

revert из предыдущего раздела является мощной командой, которая позволяет отменить любые коммиты в репозиторий. Однако, и оригинальный и «отмененный» коммиты видны в истории ветки (при использовании команды git log).

Часто мы делаем коммит, и сразу понимаем, что это была ошибка. Было бы неплохо иметь команду «возврата», которая позволила бы нам сделать вид, что неправильного коммита никогда и не было. Команда «возврата» даже предотвратила бы появление нежелательного коммита в истории git log.

1. **Команда**reset

Мы уже видели команду reset и использовали ее для согласования области подготовки с выбранным коммитом (в предыдущем уроке мы использовали коммит HEAD).

Если выполнить команду reset с указанием ссылки на коммит (т.е. метки HEAD, имени ветки или тега, хеша коммита), то команда...

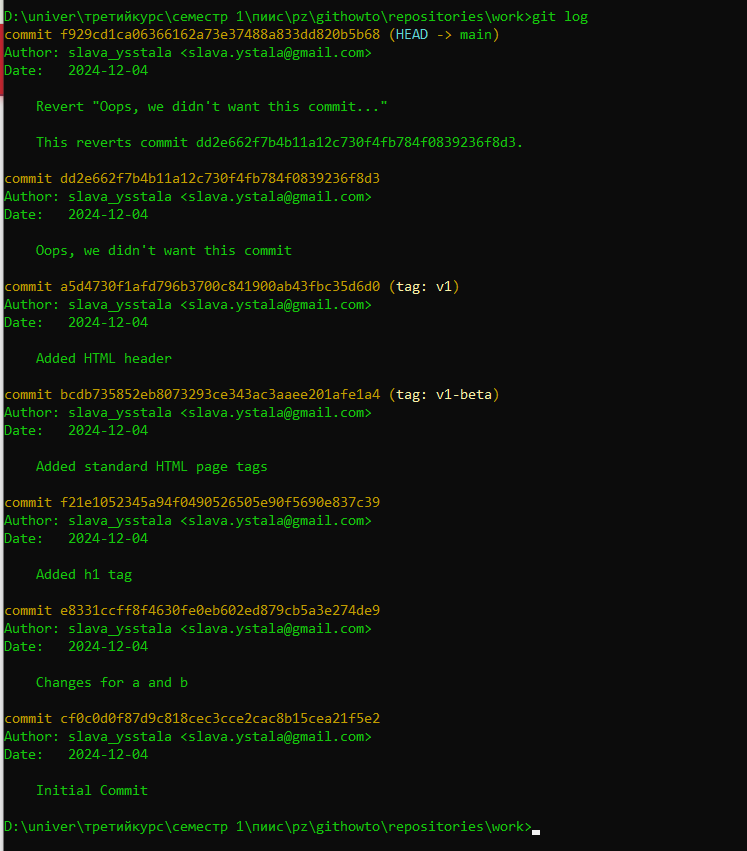
1. Изменит текущую ветку, чтобы она указывала на указанный коммит.
2. Опционально сбросит область подготовки до соответствия с указанным коммитом.
3. Опционально сбросит рабочую директорию до соответствия с указанным коммитом.
4. **Проверьте нашу историю**

Давайте сделаем быструю проверку нашей истории коммитов.

**Выполните**

git log

**Результат**

****

$ git log

86364a1 2023-11-28 | Revert "Oops, we didn't want this commit" (HEAD -> main) [Alexander Shvets]

6a44bec 2023-11-28 | Oops, we didn't want this commit [Alexander Shvets]

b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

Мы видим, что два последних коммита в этой ветке - «Oops» и «Revert Oops». Давайте удалим их с помощью сброса.

1. **Для начала отметьте эту ветку**

Но прежде чем удалить коммиты, давайте отметим последний коммит тегом, чтобы потом можно было его найти.

**Выполните**

git tag oops

**Результат**

$ git log

86364a1 2023-11-28 | Revert "Oops, we didn't want this commit" (HEAD -> main, tag: oops) [Alexander Shvets]

6a44bec 2023-11-28 | Oops, we didn't want this commit [Alexander Shvets]

b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

1. **Сброс к коммиту, предшествующему**oops

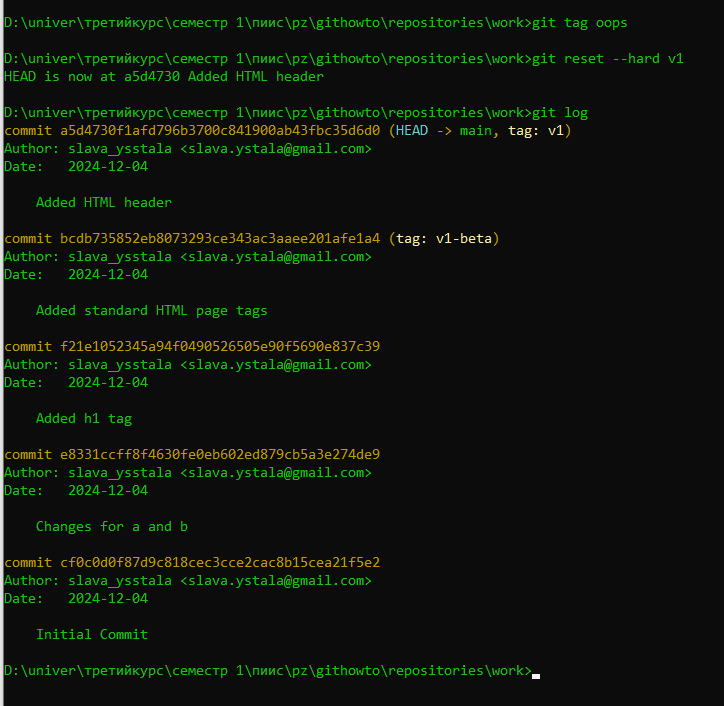
Глядя на историю лога (см. выше), мы видим, что коммит с тегом v1 является коммитом, предшествующим ошибочному коммиту. Давайте сбросим ветку до этой точки. Поскольку ветка имеет тег, мы можем использовать имя тега в команде сброса reset (если она не имеет тега, мы можем использовать хеш коммита).

**Выполните**

git reset --hard v1

git log

**Результат**

****

$ git reset --hard v1

HEAD is now at b7614c1 Added HTML header

$ git log

b7614c1 2023-11-28 | Added HTML header (HEAD -> main, tag: v1) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

Наша ветка main теперь указывает на коммит v1, а коммитов "Revert Oops" и "Oops" в ветке уже нет. Параметр --hard указывает, что рабочая директория должна быть приведена к тому состоянию, которое соответствует HEAD-коммиту ветки.

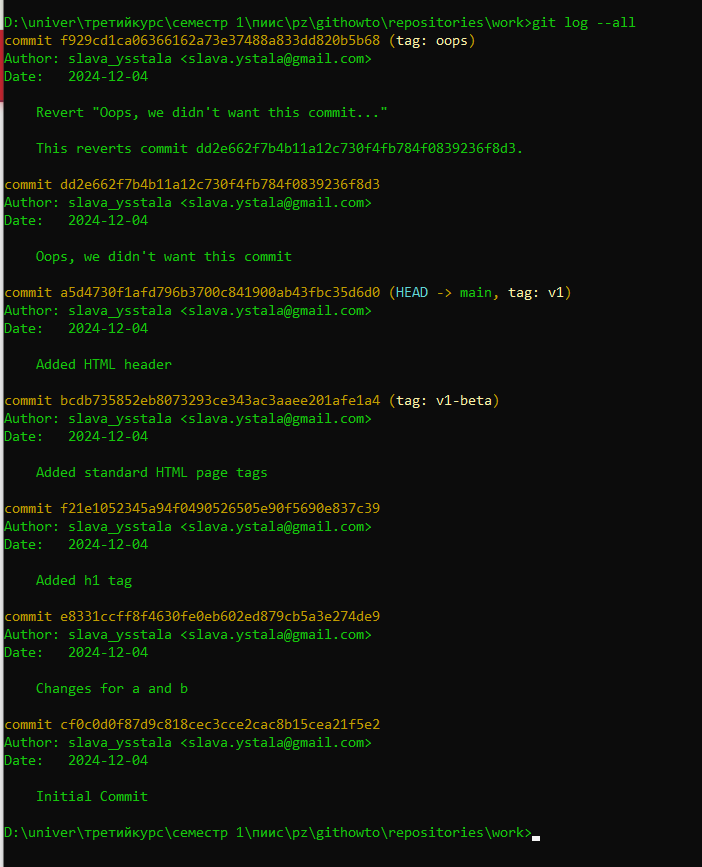
1. **Ничего никогда не теряется**

Что же случается с ошибочными коммитами? Оказывается, что коммиты все еще находятся в репозитории. На самом деле, мы все еще можем на них ссылаться. Помните, в начале этого урока мы создали для отмененного коммита тег oops? Давайте посмотрим на *все* коммиты.

**Выполните**

git log --all

**Результат**

****

$ git log --all

b7614c1 2023-11-28 | Added HTML header (HEAD -> main, tag: v1) [Alexander Shvets]

86364a1 2023-11-28 | Revert "Oops, we didn't want this commit" (tag: oops) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

6a44bec 2023-11-28 | Oops, we didn't want this commit [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

Мы видим, что ошибочные коммиты не исчезли. Они все еще находятся в репозитории. Просто они отсутствуют в ветке main. Если бы мы не отметили их тегами, они по-прежнему находились бы в репозитории, но не было бы никакой возможности ссылаться на них, кроме как при помощи хешей этих коммитов. Коммиты, на которые нет ссылок, остаются в репозитории до тех пор, пока не будет запущен сборщик мусора.

1. **Опасность сброса**

Сброс в локальных ветках, как правило, безопасен. Последствия любой «аварии» как правило, можно восстановить простым сбросом с помощью нужного коммита.

Однако, если ветка уже стала общедоступной на удаленных репозиториях, сброс может сбить с толку других пользователей ветки.

**16. Удаление тега**oops

**Цели**

* Удаление тега oops.

1. **Удаление тега**oops

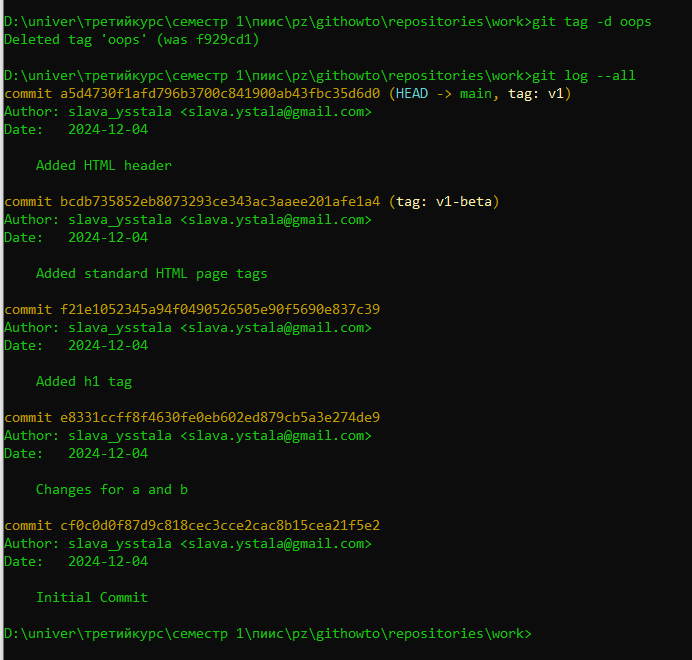
Тег oops свою функцию выполнил, давайте удалим его. Это позволит внутреннему механизму Git убрать остаточные коммиты, на которые теперь не ссылаются никакие ветки или теги.

**Выполните**

git tag -d oops

git log --all

**Результат**

****

$ git tag -d oops

Deleted tag 'oops' (was 86364a1)

$ git log --all

b7614c1 2023-11-28 | Added HTML header (HEAD -> main, tag: v1) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

Тег oops больше не будет отображаться в репозитории.

**17. Внесение изменений в коммиты**

**Цели**

* Научиться изменять существующие коммиты.

1. **Измените страницу, а затем сделайте коммит**

Добавьте в страницу комментарий автора.

**Файл: hello.html**

<!-- Author: Alexander Shvets -->

<html>

<head>

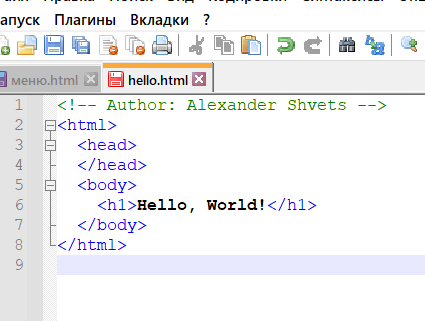
</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>



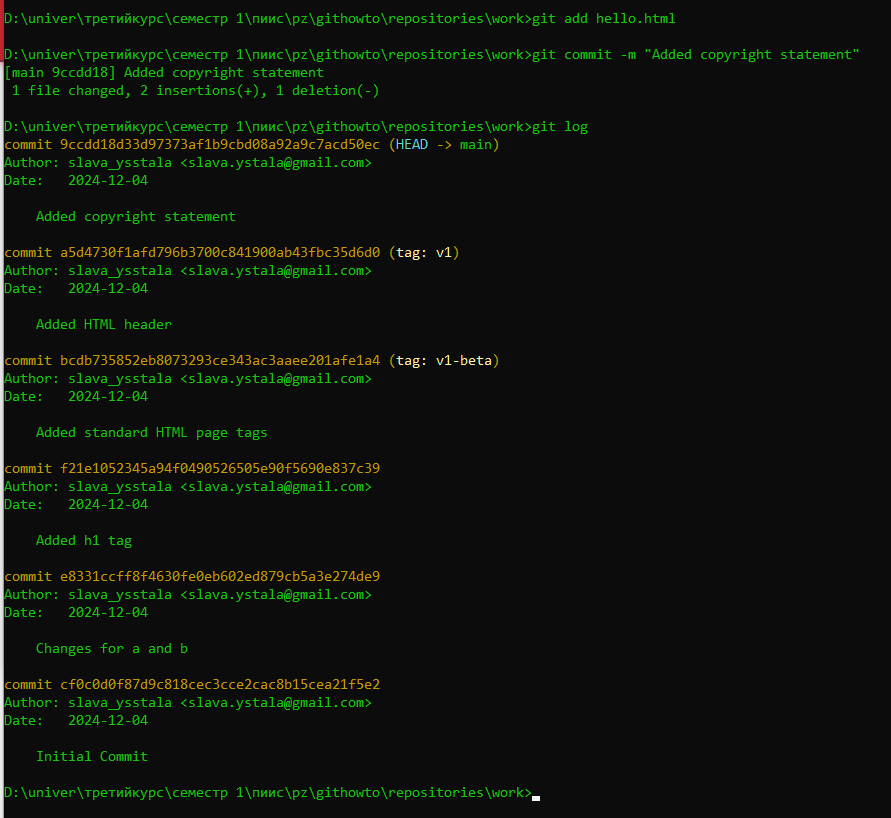
**Выполните**

git add hello.html

git commit -m "Added copyright statement"

git log

**Результат**

****

$ git add hello.html

$ git commit -m "Added copyright statement"

[main e641c0e] Added copyright statement

1 file changed, 1 insertion(+)

$ git log

e641c0e 2023-11-28 | Added copyright statement (HEAD -> main) [Alexander Shvets]

b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

1. **Ой... необходим email**

Однако после создания коммита вы понимаете, что любой хороший комментарий должен включать электронную почту автора. Обновите страницу hello.html, включив в нее email.

**Файл: hello.html**

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

1. **Измените предыдущий коммит**

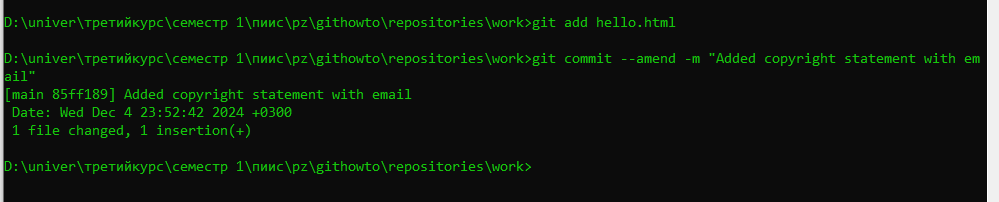
Мы действительно не хотим создавать отдельный коммит только ради электронной почты. Давайте изменим предыдущий коммит, включив в него адрес электронной почты.

**Выполните**

git add hello.html

git commit --amend -m "Added copyright statement with email"

**Результат**

****

$ git add hello.html

$ git commit --amend -m "Added copyright statement with email"

[main 9288a33] Added copyright statement with email

Date: Tue Nov 28 05:51:38 2023 -0600

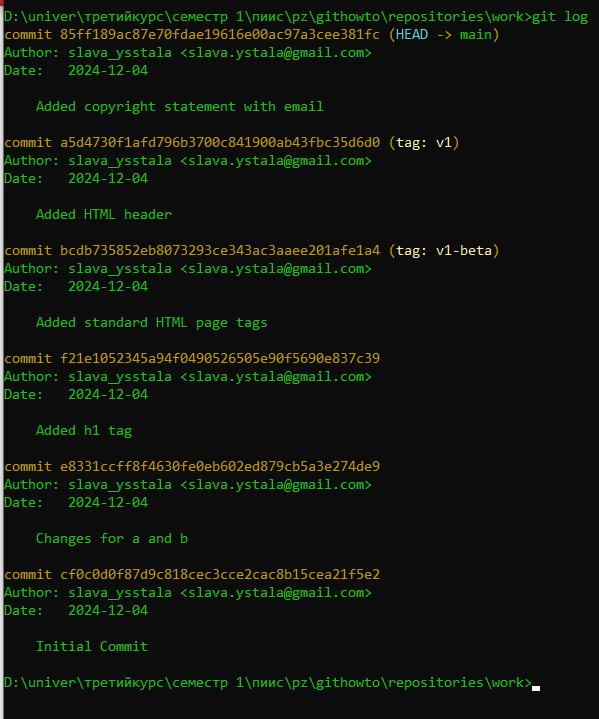
1 file changed, 1 insertion(+)

1. **Просмотр истории**

**Выполните**

git log

**Результат**

****

$ git log

9288a33 2023-11-28 | Added copyright statement with email (HEAD -> main) [Alexander Shvets]

b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

Мы можем увидеть, что оригинальный коммит «автор» заменен коммитом «автор/email». Этого же эффекта можно достичь путем сброса последнего коммита в ветке, и повторного коммита новых изменений.

**18. Создание ветки**

**Цели**

* Научиться создавать локальную ветку в репозитории.

Разработка новой функциональности всегда связана с риском: разработка может занять много времени, вы можете в конечном итоге отказаться от неё и т. д. По этой причине лучше всего изолировать разработку фичи в отдельной ветке. Когда фича будет готова, вы сможете слить эту ветку с веткой main. До того времени ветка main будет защищена от рискованного и непроверенного кода. Кроме того, вы можете работать над несколькими фичами параллельно, над каждой в собственной ветке. Вы также можете в любой момент вносить изменения в ветке main, например, чтобы исправить ошибку в стабильном коде.

1. **Создайте ветку**

Пришло время сделать нашу страницу более стильной с помощью CSS. Мы будем развивать эту возможность в новой ветке под названием style.

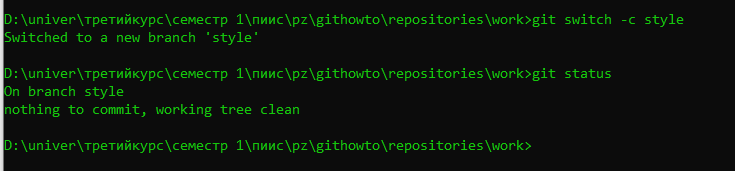
**Выполните**

git switch -c style

git status

Старожилы могут возразить, что их учили создавать ветки командой git checkout -b style. Помните, я упоминал, что команда checkout перегружена функциями и флагами? Старый способ все еще работает, но он не рекомендуется. Новая команда git switch более выразительна и менее восприимчива к ошибкам. Кроме того, в ней меньше флагов и опций, поэтому ее легче запомнить.

**Результат**

****

$ git switch -c style

Switched to a new branch 'style'

$ git status

On branch style

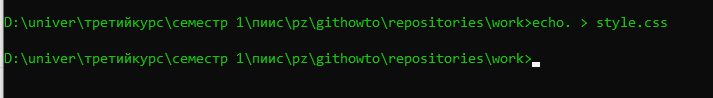
nothing to commit, working tree clean

Обратите внимание, что команда git status сообщает о том, что вы находитесь в ветке style.

1. **Добавьте файл стилей**style.css

**Выполните**

touch style.css

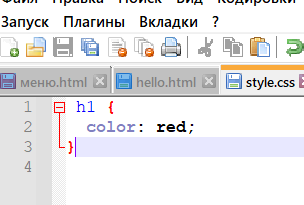


**Файл: style.css**

h1 {

color: red;

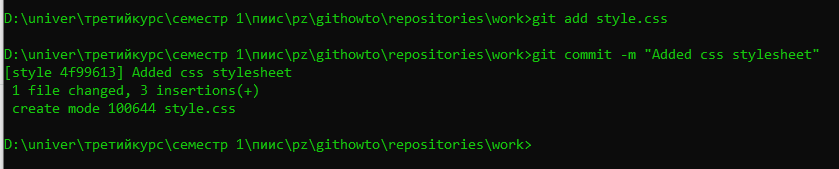
}



**Выполните**

git add style.css

git commit -m "Added css stylesheet"



1. **Измените**hello.html**, чтобы он использовал**style.css**.**

**Файл: hello.html**

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

</head>

<body>

<h1>Hello, World!</h1>

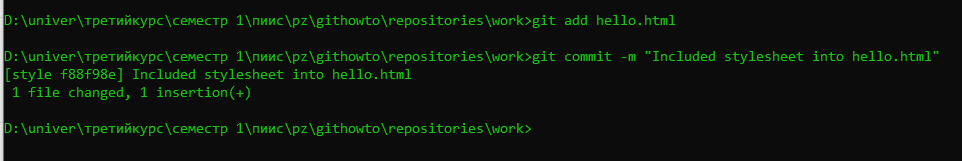
</body>

</html>

**Выполните**

git add hello.html

git commit -m "Included stylesheet into hello.html"



1. **Далее**

Теперь у нас есть новая ветка под названием style с двумя новыми коммитами. Далее мы узнаем, как переключаться между ветками.

**19. Переключение веток**

**Цели**

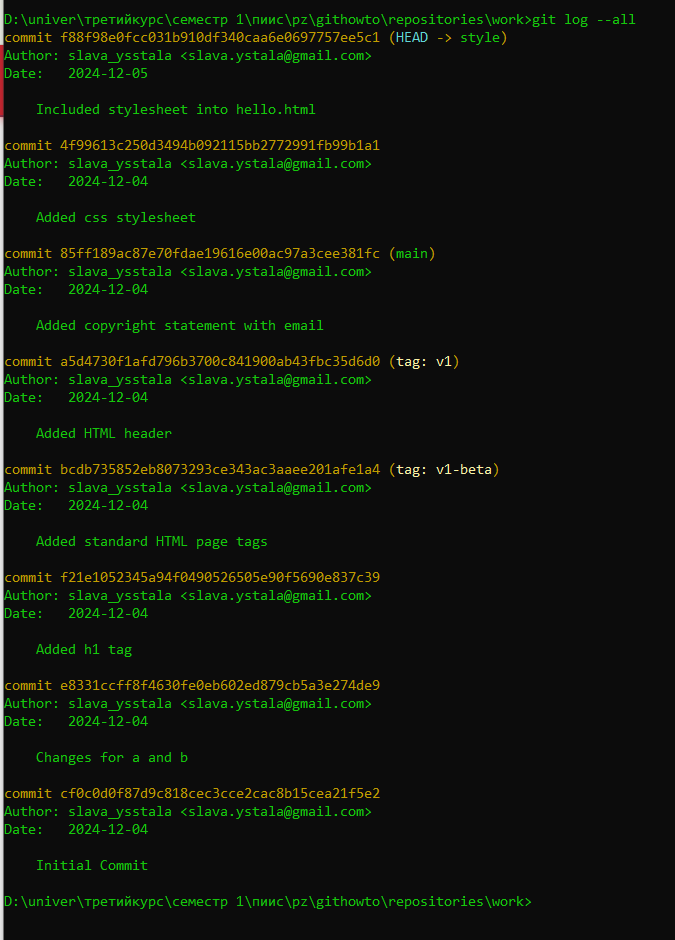
* Научиться перемещаться между ветками репозитория.

Теперь в вашем проекте есть две ветки:

**Выполните**

git log --all

**Результат**

****

$ git log --all

9288a33 2023-11-28 | Added copyright statement with email (main) [Alexander Shvets]

903eb1d 2023-11-28 | Included stylesheet into hello.html (HEAD -> style) [Alexander Shvets]

b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

555372e 2023-11-28 | Added css stylesheet [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

1. **Переключение на ветку**main

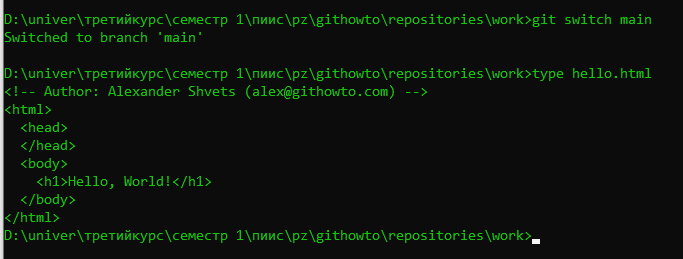
Просто используйте команду git switch для переключения между ветками.

**Выполните**

git switch main

type hello.html

**Результат**

****

$ git switch main

Switched to branch 'main'

$ cat hello.html

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

Теперь мы находимся в ветке main. Как видите, в hello.html нет никаких следов style.css. Не волнуйтесь, эти изменения все еще есть в репозитории, но мы не можем увидеть их из ветки main.

1. **Вернемся к ветке**style

**Выполните**

git switch style

type hello.html

**Результат**

****

$ git switch style

Switched to branch 'style'

$ cat hello.html

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

Мы вернулись к ветке style. Как видите, наши изменения, связанные с CSS, присутствуют.

**20. Перемещение файлов**

**Цели**

* Научиться перемещать файлы в пределах репозитория.

Я доволен нашими CSS-изменениями, но есть только один момент, который я хотел бы решить до того, как мы объединим наши изменения с main. Давайте переименуем файл hello.html в index.html. Также давайте перенесем наш файл стилей в специально отведенную директорию css.

1. **Просмотр истории изменений в конкретном файле**

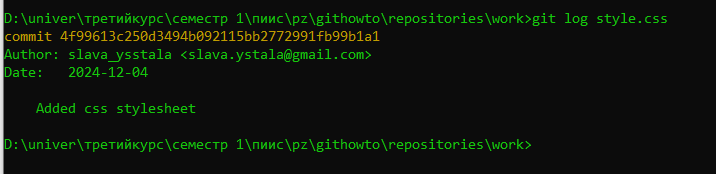
Git позволяет просматривать историю изменений конкретного файла. Давайте посмотрим историю изменений файла hello.html перед его переименованием.

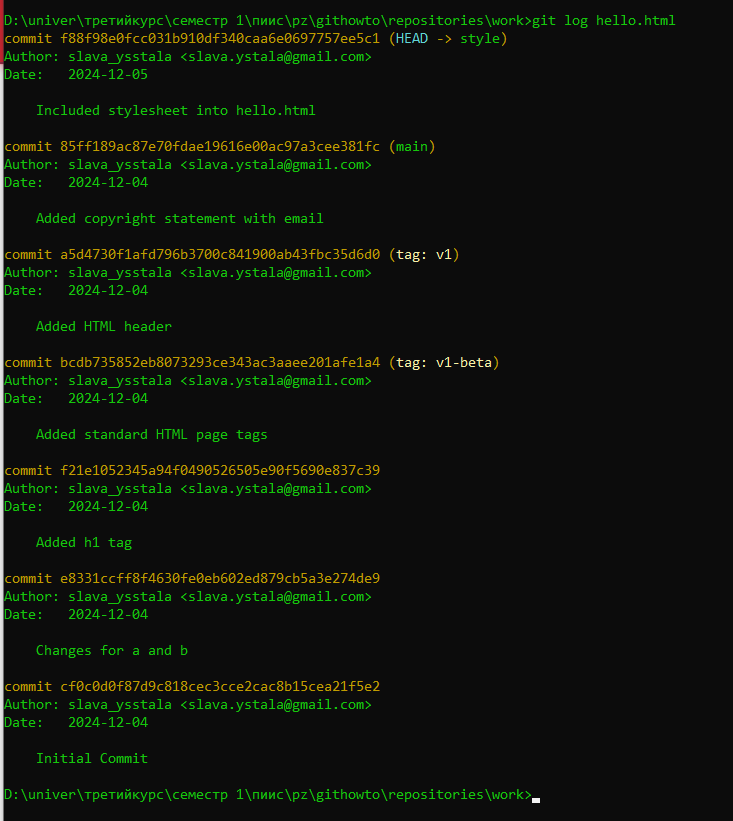
**Выполните**

git log hello.html

git log style.css

**Результат**

****

****

$ git log hello.html

903eb1d 2023-11-28 | Included stylesheet into hello.html (HEAD -> style) [Alexander Shvets]

9288a33 2023-11-28 | Added copyright statement with email (main) [Alexander Shvets]

b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

$ git log style.css

555372e 2023-11-28 | Added css stylesheet [Alexander Shvets]

1. **Просмотр различий для конкретного файла**

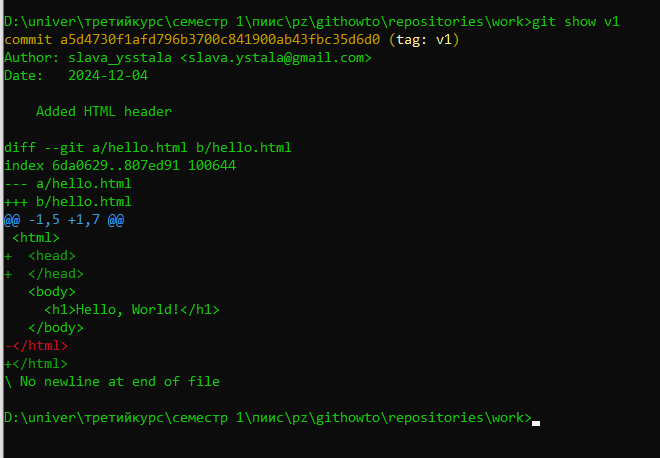
Возможность просмотра истории изменений для конкретного файла очень полезна. Она позволяет увидеть, что именно изменилось, а также кто и когда внес эти изменения. Кроме того, существует возможность увидеть изменения, связанные с конкретным коммитом. Я постоянно пользуюсь этим, чтобы разобраться, почему та или иная штука была реализована именно так в настоящей версии кода.

Команда show используется для просмотра изменений в конкретном коммите. Посмотрим изменения в файле hello.html в коммите, с тегом v1 (можно использовать любую ссылку на коммит, например, метку HEAD, хеш коммита, имя ветки или тега и т.д.).

**Выполните**

git show v1

**Результат**

****

$ git show v1

b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

diff --git a/hello.html b/hello.html

index 6da0629..0d576c4 100644

--- a/hello.html

+++ b/hello.html

@@ -1,4 +1,6 @@

<html>

+ <head>

+ </head>

<body>

<h1>Hello, World!</h1>

</body>

1. **Переименуйте файл**hello.html

Как видите, очень удобно иметь возможность видеть историю изменений конкретного файла. Но когда вы переименовываете или перемещаете какой-либо файл, есть риск потерять историю этого файла, если вы выполните эту процедуру неправильно.

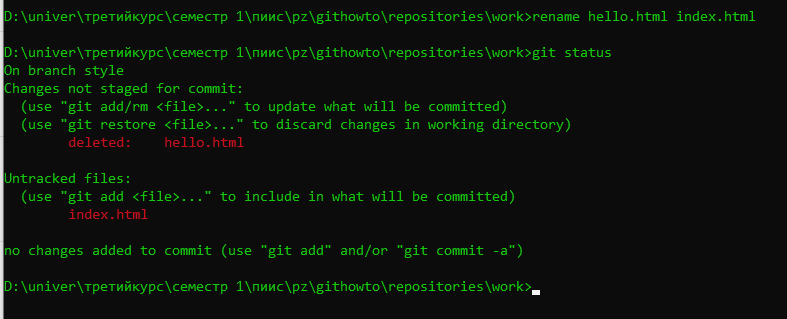
Давайте переименуем наш файл hello.html в index.html с помощью стандартной команды mv и посмотрим, что из этого получится.

**Выполните**

mv hello.html index.html

git status

**Результат**

****

$ mv hello.html index.html

$ git status

On branch style

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

deleted: hello.html

Untracked files:

(use "git add <file>..." to include in what will be committed)

index.html

no changes added to commit (use "git add" and/or "git commit -a")

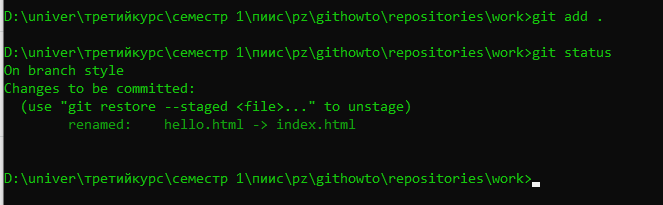
Git воспринимает наше изменение так, будто файл был удален и создан заново. Это тревожный звоночек. Нам нужно сообщить Git, что мы именно переименовали файл, а не удалили его и сразу создали новый. В простейшем случае Git сам поймёт, что файл был переименован, как только мы добавим его в индекс:

**Выполните**

git add .

git status

**Результат**

****

$ git add .

$ git status

On branch style

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

renamed: hello.html -> index.html

Видите, файл указан как переименованный. Но это всего лишь Git пытается проявить смекалку. Это не всегда работает. Например, если вы переименовали, **а также** изменили кучу файлов, Git может оказаться не в состоянии понять, что именно было переименовано. В этом случае вы можете потерять информацию об истории файлов до их переименования, поскольку файлы будут восприняты как новые.

1. **Переместите**style.css**безопасным способом**

В большинстве операционных систем переименование и перемещение файлов — это одно и то же. Итак, давайте переместим наш файл style.css в директорию css, но на этот раз сделаем это безопасно с помощью команды git mv. Эта команда гарантирует, что перемещение будет записано в истории Git как перемещение.

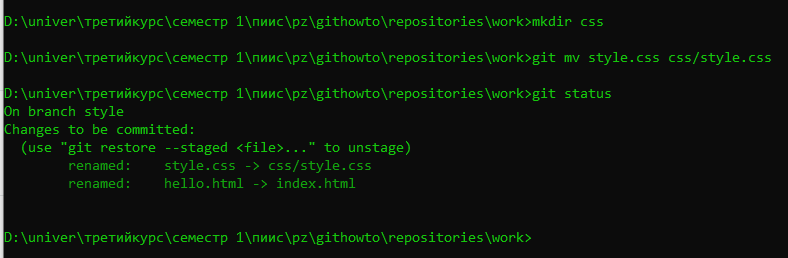
**Выполните**

mkdir css

git mv style.css css/style.css

git status

**Результат**

****

$ mkdir css

$ git mv style.css css/style.css

$ git status

On branch style

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

renamed: style.css -> css/style.css

renamed: hello.html -> index.html

Давайте закоммитим наши изменения и проверим историю изменений в файле css/styles.css. Для просмотра истории файла до его перемещения нам потребуется добавить опцию --follow. Выполним оба варианта команды, чтобы понять разницу.

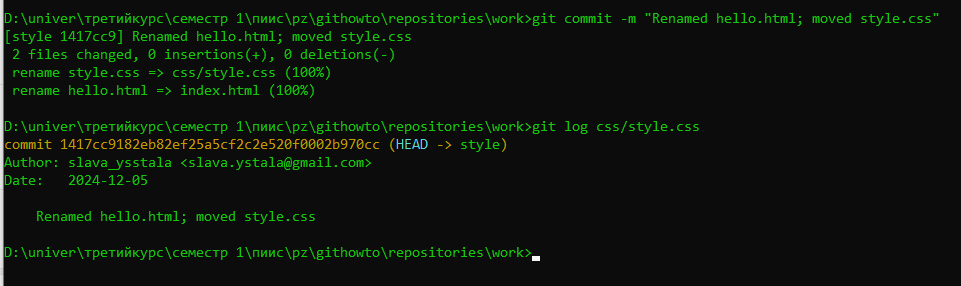
**Выполните**

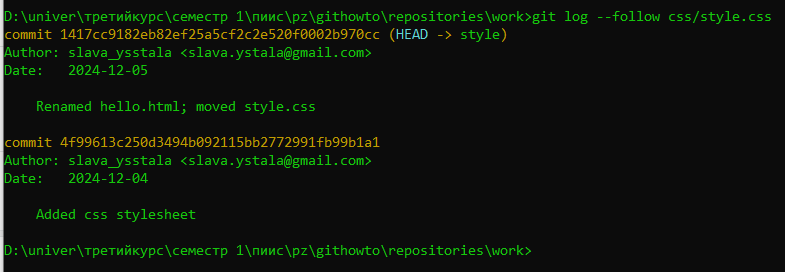
git commit -m "Renamed hello.html; moved style.css"

git log css/style.css

git log --follow css/style.css

**Результат**

****

****

$ git commit -m "Renamed hello.html; moved style.css"

[style 0ee0113] Renamed hello.html; moved style.css

2 files changed, 0 insertions(+), 0 deletions(-)

rename style.css => css/style.css (100%)

rename hello.html => index.html (100%)

$ git log css/style.css

0ee0113 2023-11-28 | Renamed hello.html; moved style.css (HEAD -> style) [Alexander Shvets]

$ git log --follow css/style.css

0ee0113 2023-11-28 | Renamed hello.html; moved style.css (HEAD -> style) [Alexander Shvets]

555372e 2023-11-28 | Added css stylesheet [Alexander Shvets]

**21. Изменения в ветке**main

**Цели**

* Научиться работать с несколькими ветками с различными (и, возможно, конфликтующими) изменениями.

Как я уже говорил, Git позволяет работать с несколькими ветками одновременно. Это очень удобно при работе в команде, поскольку люди могут параллельно работать над разными функциями. Это также полезно при работе в одиночку: разрабатывая функции в отдельных ветках, вы можете исправлять ошибки и выпускать небольшие обновления, используя стабильный код в ветке main.

1. **Создайте файл**README

Создадим файл README. В нем будет рассказано о сути нашего проекта.

**Файл: README**

This is the Hello World example from the GitHowTo tutorial.

1. **Сделайте коммит файла**README**в ветку**main

В настоящее время мы находимся в ветке style. Файл README не является частью этой ветки, поэтому перед коммитом мы должны переключиться на ветку main.

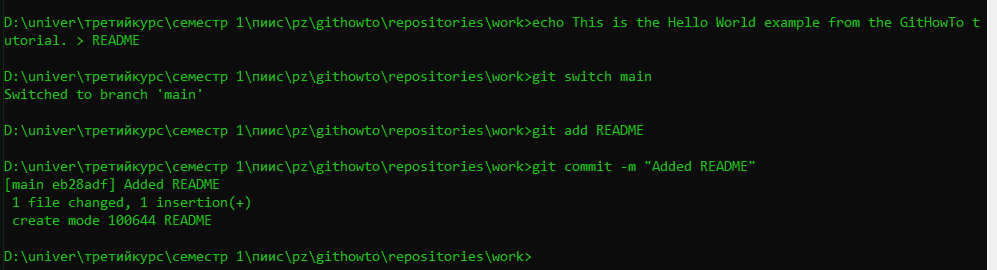
**Выполните**

git switch main

git add README

git commit -m "Added README"

**Результат**

****

$ git switch main

Switched to branch 'main'

$ git add README

$ git commit -m "Added README"

[main ee16740] Added README

1 file changed, 1 insertion(+)

create mode 100644 README

**22. Просмотр отличающихся веток**

**Цели**

* Научиться просматривать отличающиеся ветки в репозитории.

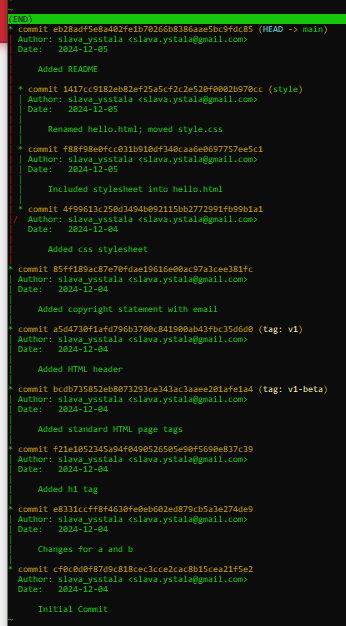
1. **Просмотрите текущие ветки**

Теперь у нас есть две расходящиеся ветки в репозитории. Используйте следующую команду log для просмотра веток и их расхождения.

**Выполните**

git log --all --graph

**Результат**

****

$ git log --all --graph

\* ee16740 2023-11-28 | Added README (HEAD -> main) [Alexander Shvets]

| \* 0ee0113 2023-11-28 | Renamed hello.html; moved style.css (style) [Alexander Shvets]

| \* 903eb1d 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]

| \* 555372e 2023-11-28 | Added css stylesheet [Alexander Shvets]

|/

\* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]

\* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

\* 46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

\* 78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

\* 5836970 2023-11-28 | Initial commit [Alexander Shvets]

Опция --all гарантирует, что мы видим все ветки, так как по умолчанию в логе показывается только текущая ветка.

Опция --graph добавляет простое дерево коммитов, представленное в виде простых текстовых линий. Мы видим обе ветки (style и main) причём ветка main помечена как HEAD, что означает, что она является текущей. Общим предком для обеих веток является ветка, в которую был внесен коммит «Added copyright statement with email».

**23. Слияние**

**Цели**

* Научиться сливать две отличающиеся ветки для переноса изменений обратно в одну ветку.

1. **Слияние веток**

Слияние переносит изменения из двух веток в одну. Давайте вернемся к ветке style и сольем main со style.

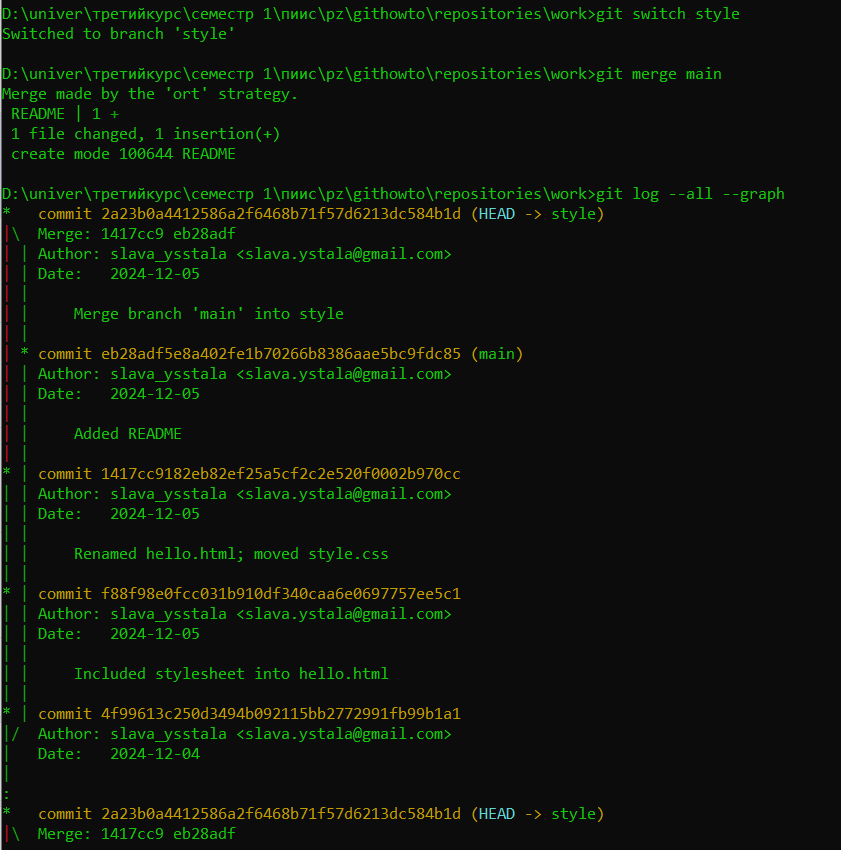
**Выполните**

git switch style

git merge main

git log --all --graph

**Результат**

****

$ git switch style

Switched to branch 'style'

$ git merge main

Merge made by the 'ort' strategy.

README | 1 +

1 file changed, 1 insertion(+)

create mode 100644 README

$ git log --all --graph

\* a33deed 2023-11-28 | Merge branch 'main' into style (HEAD -> style) [Alexander Shvets]

|\

| \* ee16740 2023-11-28 | Added README (main) [Alexander Shvets]

\* | 0ee0113 2023-11-28 | Renamed hello.html; moved style.css [Alexander Shvets]

\* | 903eb1d 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]

\* | 555372e 2023-11-28 | Added css stylesheet [Alexander Shvets]

|/

\* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]

\* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

\* 46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

\* 78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

\* 5836970 2023-11-28 | Initial commit [Alexander Shvets]

Путем периодического слияния ветки main с веткой style вы можете переносить из main любые изменения и поддерживать совместимость изменений style с изменениями в основной ветке.

Однако, это делает графики коммитов действительно уродливыми. Позже мы рассмотрим возможность перебазирования, как альтернативы слиянию.

1. **Далее**

Но что если изменения в ветке main конфликтуют с изменениями в style?

**24. Создание конфликта**

**Цели**

* Создание конфликтующих изменений в ветке main.

При слиянии двух веток Git пытается перенести изменения из одной ветки в другую. Если в обеих ветках была изменена одна и та же часть файла, Git может не справиться с автоматическим слиянием изменений. В этом случае Git сообщит о конфликте и попросит разрешить его вручную. В этом уроке мы смоделируем конфликт, а затем научимся его разрешать.

В реальной жизни конфликты слияния регулярно возникают при работе в команде. Например, вы и ваш коллега начали работать над двумя разными фичами, затрагивающими одни и те же файлы. Ваш коллега закончил работу первым и слил свои изменения в ветку main. Теперь вы хотите слить свои изменения в ветку main. Но ветка main теперь отличается от той, с которой вы начинали работать в начале — в ней появился новый код, присланный вашим коллегой. Вероятно, Git не сможет автоматически объединить ваши изменения и попросит помощи человека.

* 1. **Вернитесь в**main**и создайте конфликт**

Помните, что в нашей ветке main страница по-прежнему называется hello.html? Переключитесь обратно на ветку main и внесите следующие изменения:

git switch main

**Файл: hello.html**

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

<title>Hello World Page</title>

</head>

<body>

<h1>Hello, World!</h1>

<p>Let's learn Git together.</p>

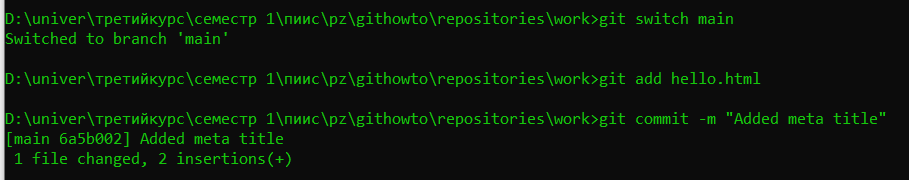
</body>

</html>

**Выполните**

git add hello.html

git commit -m "Added meta title"

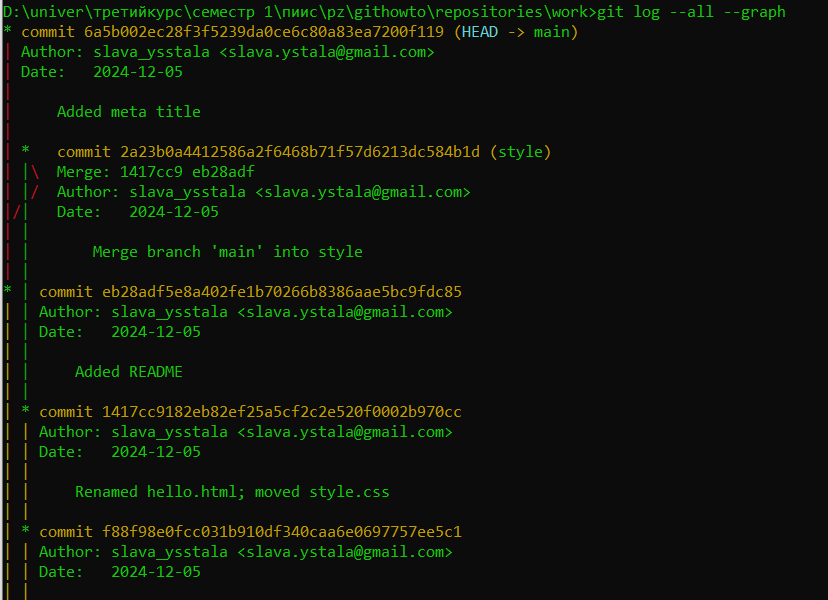


* 1. **Просмотр веток**

**Выполните**

git log --all --graph

**Результат**

****

$ git log --all --graph

\* 85c14e9 2023-11-28 | Added meta title (HEAD -> main) [Alexander Shvets]

| \* a33deed 2023-11-28 | Merge branch 'main' into style (style) [Alexander Shvets]

| |\

| |/

|/|

\* | ee16740 2023-11-28 | Added README [Alexander Shvets]

| \* 0ee0113 2023-11-28 | Renamed hello.html; moved style.css [Alexander Shvets]

| \* 903eb1d 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]

| \* 555372e 2023-11-28 | Added css stylesheet [Alexander Shvets]

|/

\* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]

\* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

\* 46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

\* 78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

\* 5836970 2023-11-28 | Initial commit [Alexander Shvets]

После коммита «Added README» ветка main была объединена с веткой style, но в настоящее время в main есть дополнительный коммит, который не был слит с style.

* 1. **Далее**

Последнее изменение в main конфликтует с некоторыми изменениями в style. На следующем шаге мы решим этот конфликт.

**25. Разрешение конфликтов**

**Цели**

* Научиться разрешать конфликты во время слияния.

1. **Слияние**main**в ветку**style

Давайте вернемся в ветку style и сольем в нее все последние изменения из ветки main.

**Выполните**

git switch style

git merge main

**Результат**

$ git switch style

Switched to branch 'style'

$ git merge main

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

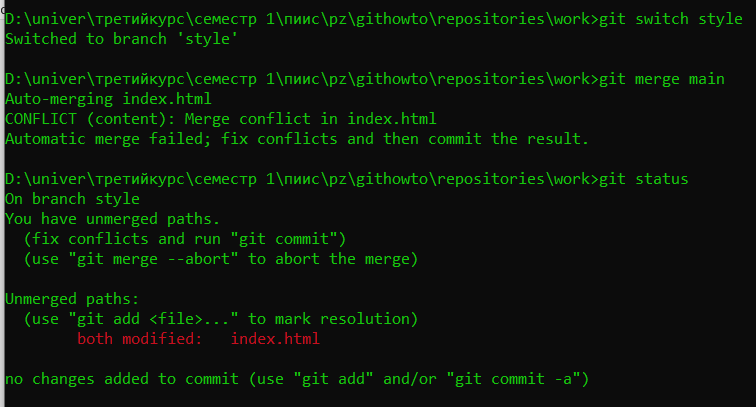
Automatic merge failed; fix conflicts and then commit the result.

Похоже, что у нас возник конфликт. Ничего удивительного! Посмотрим, что скажет по этому поводу Git:

**Выполните**

git status

**Результат**

****

$ git status

On branch style

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: index.html

no changes added to commit (use "git add" and/or "git commit -a")

Если открыть файл index.html, то можно увидеть:

**Файл: index.html**

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

<<<<<<< HEAD:index.html

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

=======

<title>Hello World Page</title>

>>>>>>> main:hello.html

</head>

<body>

<h1>Hello, World!</h1>

<p>Let's learn Git together.</p>

</body>

</html>

Часть между <<<<<<< >>>>>>> является конфликтом. Верхняя часть соответствует ветке style, которая является текущей веткой (или HEAD) репозитория. Нижняя часть соответствует изменениям из ветки main. Git не может решить, какие изменения применить, поэтому он просит вас разрешить конфликт вручную. Вы можете оставить изменения из ветки style или из main, либо объединить их любым удобным способом. Вы также можете внести в файл любые другие изменения.

Кстати, вы заметили, что наше второе изменение, тег <p>, не является частью конфликта? Это потому, что Git сумел автоматически объединить ее.

1. **Отмена слияния**

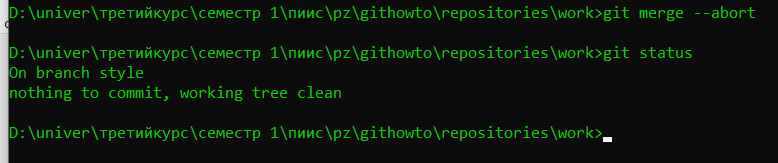
Прежде чем мы приступим к разрешению нашего конфликта, хочу заметить, что сразу бросаться к разрешению конфликта не всегда оптимально. Конфликт может быть вызван изменениями, о которых вы не знаете. Или же изменения слишком велики, чтобы разрешить конфликт сразу. По этой причине Git позволяет прервать слияние и вернуться к предыдущему состоянию. Для этого можно воспользоваться командой git merge --abort, как это было предложено командой status, которую мы выполнили ранее.

**Выполните**

git merge --abort

git status

**Результат**

****

$ git merge --abort

$ git status

On branch style

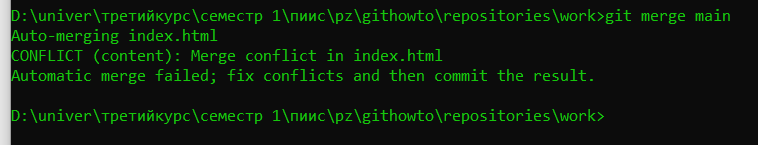
nothing to commit, working tree clean

1. **Решение конфликта**

После небольшой медитации мы готовы к разрешению конфликта. Давайте снова запустим объединение.

**Выполните**

git merge main



Чтобы разрешить конфликт, нужно отредактировать файл до состояния, которое нас устраивает, и затем закоммитить его как обычно. В нашем случае мы объединим изменения из обеих веток. Поэтому мы отредактируем файл до следующего состояния:

**Файл: index.html**

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

<title>Hello World Page</title>

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

</head>

<body>

<h1>Hello, World!</h1>

<p>Let's learn Git together.</p>

</body>

</html>

1. **Закоммитьте разрешенный конфликт**

**Выполните**

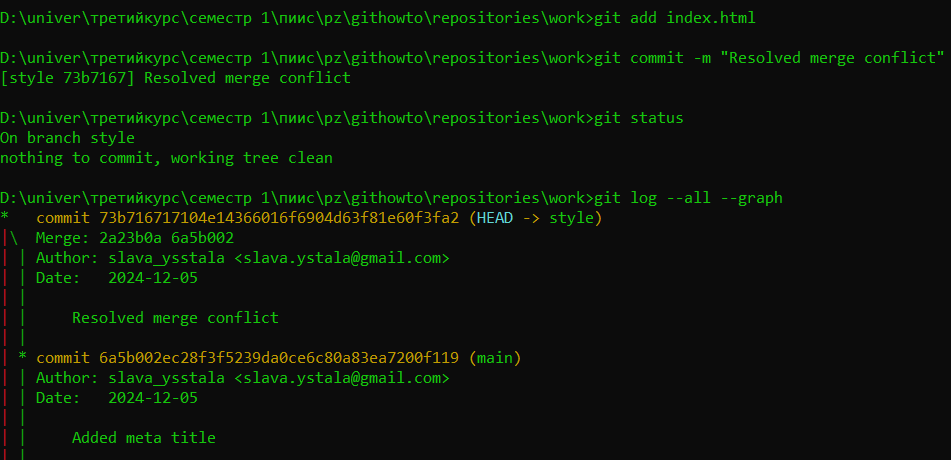
git add index.html

git commit -m "Resolved merge conflict"

git status

git log --all --graph

**Результат**

****

$ git add index.html

$ git commit -m "Resolved merge conflict"

[style 79ac6fa] Resolved merge conflict

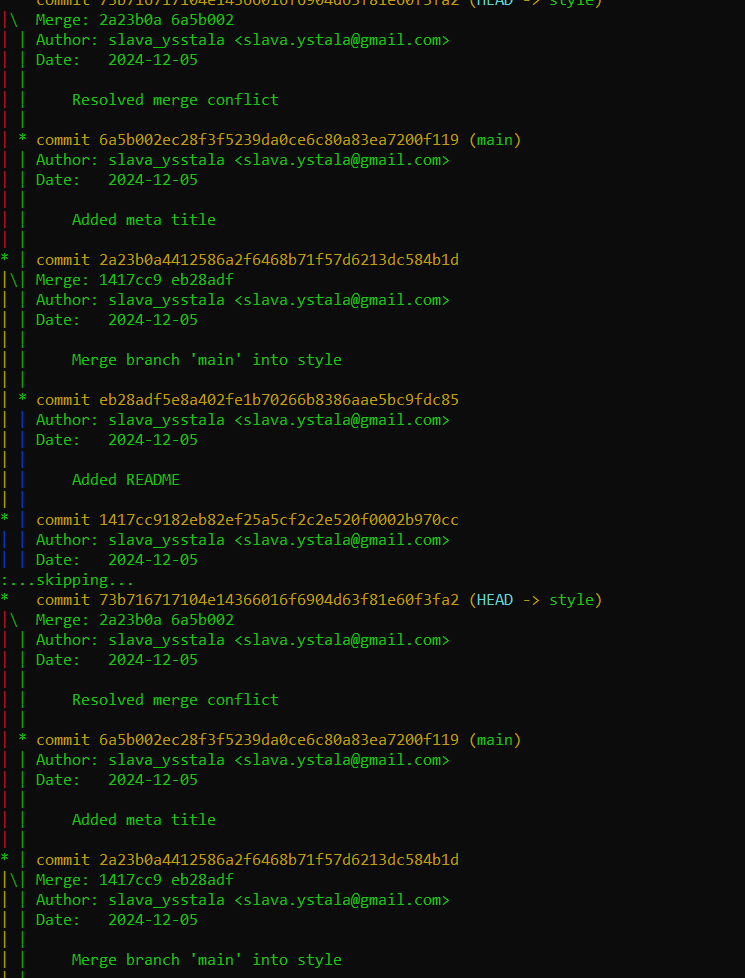
Давайте посмотрим на текущее состояние нашего хранилища и убедимся, что все в порядке:

**Выполните**

git status

git log --all --graph

**Результат**

****

$ git status

On branch style

nothing to commit, working tree clean

$ git log --all --graph

\* 79ac6fa 2023-11-28 | Resolved merge conflict (HEAD -> style) [Alexander Shvets]

|\

| \* 85c14e9 2023-11-28 | Added meta title (main) [Alexander Shvets]

\* | a33deed 2023-11-28 | Merge branch 'main' into style [Alexander Shvets]

|\|

| \* ee16740 2023-11-28 | Added README [Alexander Shvets]

\* | 0ee0113 2023-11-28 | Renamed hello.html; moved style.css [Alexander Shvets]

\* | 903eb1d 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]

\* | 555372e 2023-11-28 | Added css stylesheet [Alexander Shvets]

|/

\* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]

\* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

\* 46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

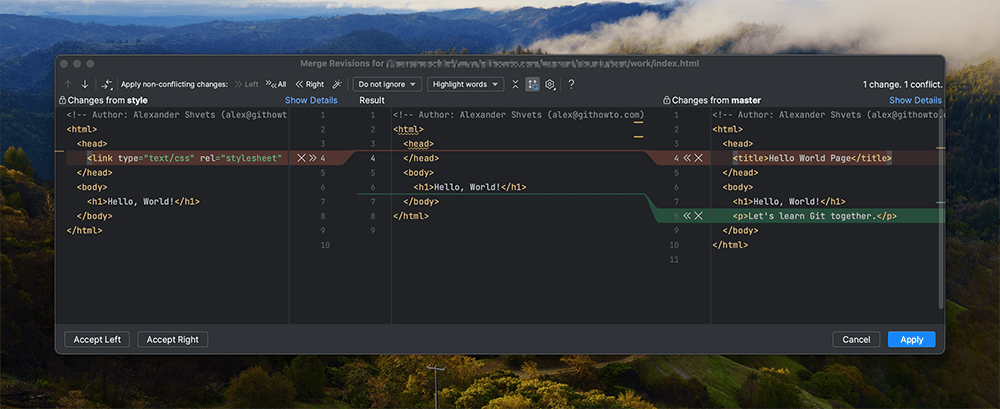
\* 78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

\* 5836970 2023-11-28 | Initial commit [Alexander Shvets]

***05.* Расширенные возможности слияния**

Git не предоставляет никаких графических инструментов слияния, но будет с удовольствием работать с любыми сторонними инструментами слияния, которые вы хотите использовать ([обсуждение таких инструментов на StackOverflow](http://stackoverflow.com/questions/137102/whats-the-best-visual-merge-tool-for-git)).

При работе с кодом в современной интегрированной среде разработки (IDE), такой как Visual Studio или IntelliJ IDEA, вы, скорее всего, будете использовать встроенный инструмент слияния. Например, вот как выглядит разрешение конфликтов в IntelliJ IDEA:



**26.**rebase**против**merge

**Цели**

* Узнать различия между перебазированием и слиянием.

**Обсуждение**

Давайте рассмотрим различия между слиянием и перебазированием. Для того чтобы это сделать, нам нужно вернуться в репозиторий в момент до первого слияния, а затем повторить те же действия, но с использованием перебазирования вместо слияния.

Мы будем использовать команду reset для возврата веток к предыдущему состоянию.

**27. Сброс ветки**style

**Цели**

* Сбросить ветку style до точки перед первым слиянием.

1. **Сбросьте ветку**style

Давайте вернемся во времени на ветке style к точке *перед* тем, как мы слили ее с веткой main. Мы можем сбросить ветку к любому коммиту при помощи команды reset. По сути, это изменение указателя ветки на любую точку дерева коммитов.

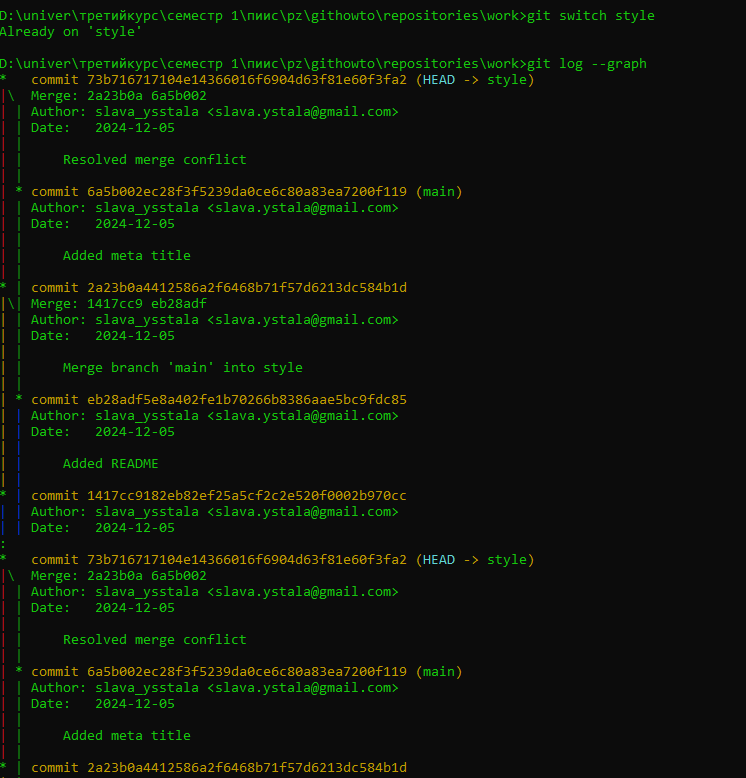
В этом случае мы хотим вернуться в ветке style в точку перед слиянием с main. Нам необходимо найти последний коммит перед слиянием.

**Выполните**

git switch style

git log --graph

**Результат**

****

$ git switch style

Already on 'style'

$ git log --graph

\* 79ac6fa 2023-11-28 | Resolved merge conflict (HEAD -> style) [Alexander Shvets]

|\

| \* 85c14e9 2023-11-28 | Added meta title (main) [Alexander Shvets]

\* | a33deed 2023-11-28 | Merge branch 'main' into style [Alexander Shvets]

|\|

| \* ee16740 2023-11-28 | Added README [Alexander Shvets]

\* | 0ee0113 2023-11-28 | Renamed hello.html; moved style.css [Alexander Shvets]

\* | 903eb1d 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]

\* | 555372e 2023-11-28 | Added css stylesheet [Alexander Shvets]

|/

\* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]

\* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

\* 46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

\* 78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

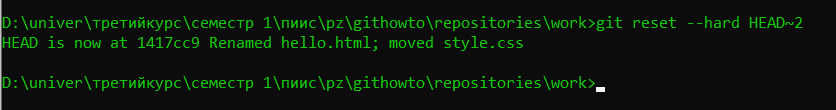
\* 5836970 2023-11-28 | Initial commit [Alexander Shvets]

Это немного трудно читать, но, глядя на данные, мы видим, что коммит «Renamed hello.html; moved style.css» был последним на ветке style перед слиянием. Давайте сбросим ветку style к этому коммиту. Чтобы сослаться на этот коммит, мы либо используем его хеш, либо посчитаем, что этот коммит находится за 2 коммита до HEAD, то есть HEAD~2 в нотации Git.

**Выполните**

git reset --hard HEAD~2

**Результат**

****

$ git reset --hard HEAD~2

HEAD is now at 0ee0113 Renamed hello.html; moved style.css

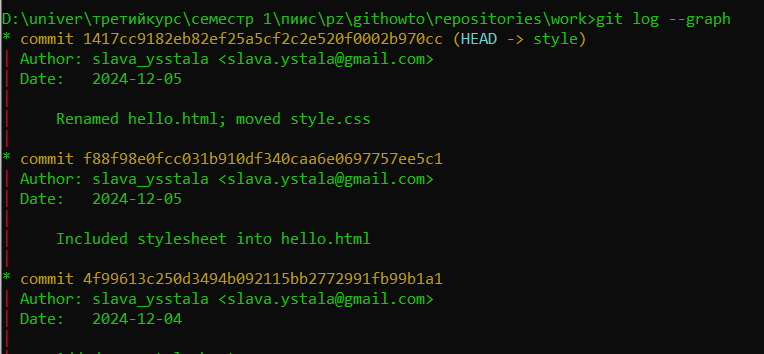
1. **Проверьте ветку**

Теперь проверим историю ветки style. В истории не должно быть коммитов слияния.

**Выполните**

git log --graph

**Результат**

****

$ git log --graph

\* 0ee0113 2023-11-28 | Renamed hello.html; moved style.css (HEAD -> style) [Alexander Shvets]

\* 903eb1d 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]

\* 555372e 2023-11-28 | Added css stylesheet [Alexander Shvets]

\* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]

\* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

\* 46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

\* 78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

\* 5836970 2023-11-28 | Initial commit [Alexander Shvets]

**28. Перебазирование**

**Цели**

* Использовать команду rebase вместо команды merge.

Мы вернули ветку style к точке перед первым слиянием. При этом в ветке main есть два коммита, которых нет в ветке style: новый файл README и конфликтующее изменение в файле index.html. На этот раз мы перенесем эти изменения в ветку style с помощью команды rebase, а не merge.

1. **Перебазируем ветку**style**на ветку**main

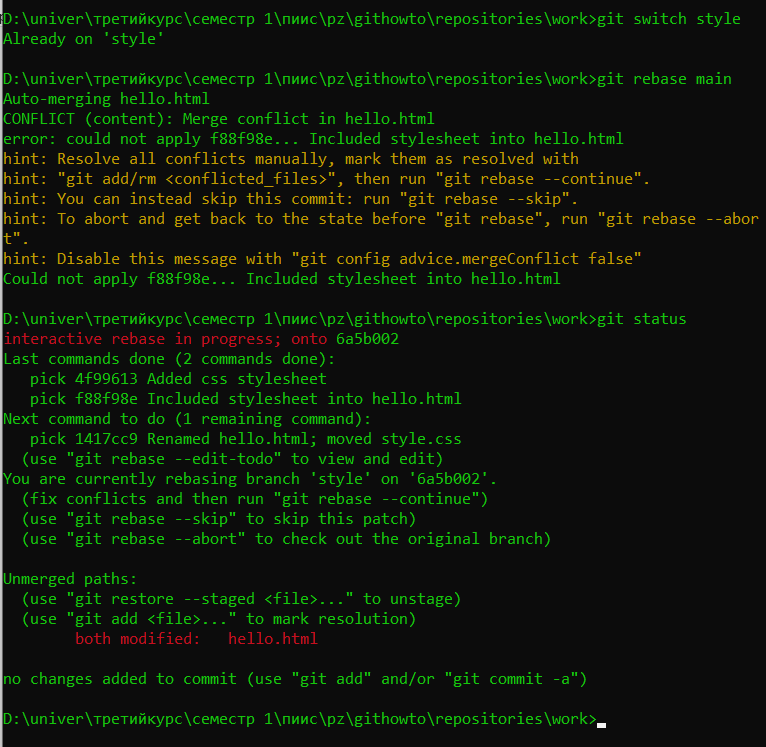
**Выполните**

git switch style

git rebase main

git status

**Результат**

****

$ git switch style

Already on 'style'

$ git rebase main

Rebasing (1/3)

Rebasing (2/3)

Auto-merging hello.html

CONFLICT (content): Merge conflict in hello.html

error: could not apply 903eb1d... Included stylesheet into hello.html

hint: Resolve all conflicts manually, mark them as resolved with

hint: "git add/rm <conflicted\_files>", then run "git rebase --continue".

hint: You can instead skip this commit: run "git rebase --skip".

hint: To abort and get back to the state before "git rebase", run "git rebase --abort".

Could not apply 903eb1d... Included stylesheet into hello.html

$ git status

interactive rebase in progress; onto 85c14e9

Last commands done (2 commands done):

pick 555372e Added css stylesheet

pick 903eb1d Included stylesheet into hello.html

Next command to do (1 remaining command):

pick 0ee0113 Renamed hello.html; moved style.css

(use "git rebase --edit-todo" to view and edit)

You are currently rebasing branch 'style' on '85c14e9'.

(fix conflicts and then run "git rebase --continue")

(use "git rebase --skip" to skip this patch)

(use "git rebase --abort" to check out the original branch)

Unmerged paths:

(use "git restore --staged <file>..." to unstage)

(use "git add <file>..." to mark resolution)

both modified: hello.html

no changes added to commit (use "git add" and/or "git commit -a")

Опять возник конфликт! Обратите внимание, что конфликт возник в файле hello.html, а не в файле index.html, как в прошлый раз. Это связано с тем, что rebase находился в процессе применения изменений style поверх ветки main. Файл hello.html в main еще не был переименован, поэтому он все еще имеет старое имя.

При слиянии возник бы «обратный» конфликт. При слиянии изменения ветки main были бы применены поверх ветки style. В ветке style файл переименован, поэтому конфликт возник бы в файле index.html.

**Файл: hello.html**

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

<<<<<<< HEAD

<title>Hello World Page</title>

=======

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

>>>>>>> 903eb1d (Included stylesheet into hello.html)

</head>

<body>

<h1>Hello, World!</h1>

<p>Let's learn Git together.</p>

</body>

</html>

1. **Разрешение конфликта**

Сам конфликт может быть разрешен тем же способом, что и предыдущий. Сначала мы изменим файл hello.html, чтобы он соответствовал нашим ожиданиям.

**Файл: hello.html**

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

<title>Hello World Page</title>

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

</head>

<body>

<h1>Hello, World!</h1>

<p>Let's learn Git together.</p>

</body>

</html>

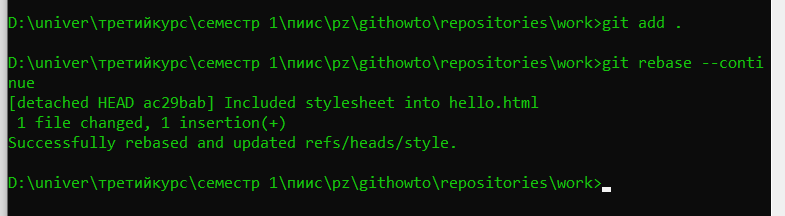
Но после этого нам не нужно коммитить изменения. Мы можем просто добавить файл в индекс и продолжить процесс rebase. Вот почему я люблю rebase! Он позволяет мне устранять конфликты, не создавая кучу уродливых конфликтов слияния.

Для простоты мы можем добавить все изменения, используя ., что означает путь к текущей директории. Git интерпретирует это как «добавить все изменения из текущей директории и её поддиректорий».

**Выполните**

git add .

git rebase –continue



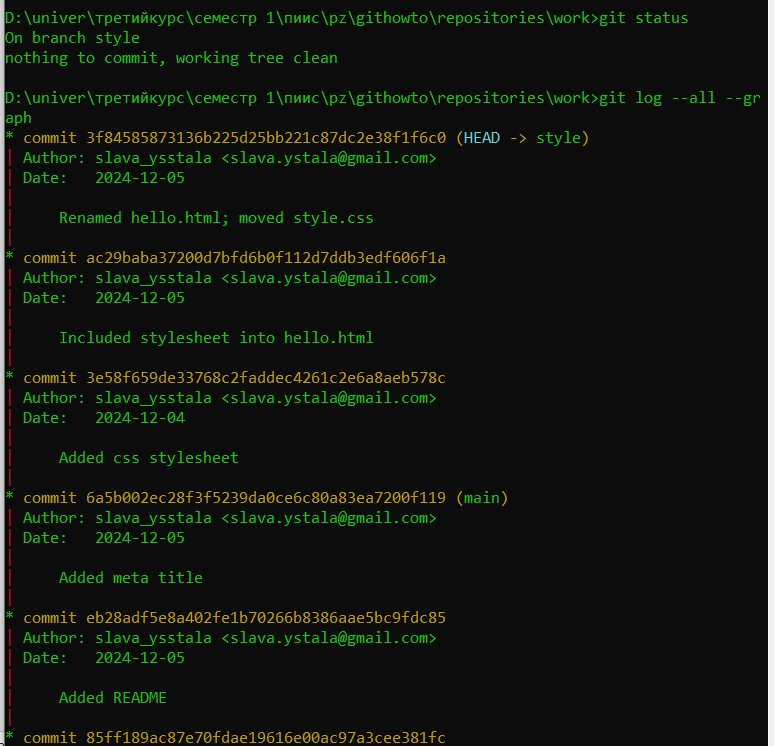
Здесь, скорее всего, Git снова откроет редактор, чтобы позволить нам изменить текст коммита. Мы можем оставить текст без изменений. После сохранения изменений Git завершит процесс rebase, и мы cможем выполнить следующие команды:

**Выполните**

git status

git log --all --graph

**Результат**

****

$ git add .

$ git rebase --continue

[detached HEAD 23149b5] Included stylesheet into hello.html

1 file changed, 1 insertion(+)

Rebasing (3/3)

[KSuccessfully rebased and updated refs/heads/style.

$ git status

On branch style

nothing to commit, working tree clean

$ git log --all --graph

\* 39a1e0f 2023-11-28 | Renamed hello.html; moved style.css (HEAD -> style) [Alexander Shvets]

\* 23149b5 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]

\* b9e6de1 2023-11-28 | Added css stylesheet [Alexander Shvets]

\* 85c14e9 2023-11-28 | Added meta title (main) [Alexander Shvets]

\* ee16740 2023-11-28 | Added README [Alexander Shvets]

\* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]

\* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

\* 46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

\* 78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

\* 5836970 2023-11-28 | Initial commit [Alexander Shvets]

***03.* Слияние VS перебазирование**

Конечный результат перебазирования очень похож на результат слияния. Ветка style в настоящее время содержит все свои изменения, а также все изменения ветки main. Однако, дерево коммитов значительно отличается. Дерево коммитов ветки style было переписано таким образом, что ветка main является частью истории коммитов. Это делает цепь коммитов линейной и гораздо более читабельной.

***04.* Когда использовать команду**rebase**, а когда команду**merge**?**

Используйте команду rebase:

* Когда вы получаете изменения из удаленного репозитория и хотите применить их к своей локальной ветке.
* Если вы хотите, чтобы история коммитов была линейной и легко читаемой.

Не используйте команду rebase:

* Если текущая ветка является публичной и общей. Переписывание таких веток будет мешать работе других членов команды.
* Если важна *точная* история ветки коммитов (поскольку команда rebase переписывает историю коммитов).

Учитывая приведенные выше рекомендации, я предпочитаю использовать команду rebase для краткосрочных, локальных веток и команду merge для веток в публичном репозитории.

**29. Слияние в ветку**main

**Цели**

* Мы поддерживали соответствие ветки style с веткой main (с помощью rebase), теперь давайте сольем изменения style в ветку main.

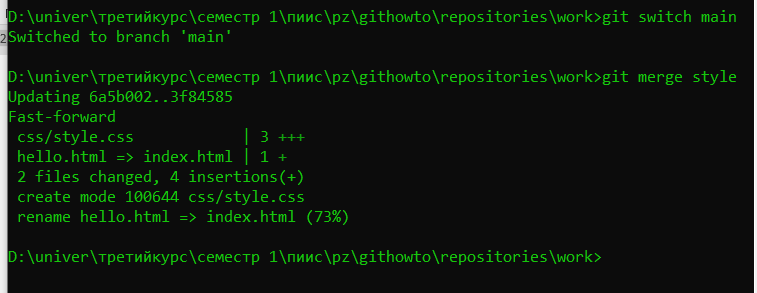
1. **Слейте**style**в**main

**Выполните**

git switch main

git merge style

**Результат**

****

$ git switch main

Switched to branch 'main'

$ git merge style

Updating 85c14e9..39a1e0f

Fast-forward

css/style.css | 3 +++

hello.html => index.html | 1 +

2 files changed, 4 insertions(+)

create mode 100644 css/style.css

rename hello.html => index.html (73%)

Поскольку последний коммит в main предшествует последнему коммиту ветки style, Git может выполнить ускоренное слияние, просто переместив указатель ветки вперед, на тот же коммит, что и ветка style.

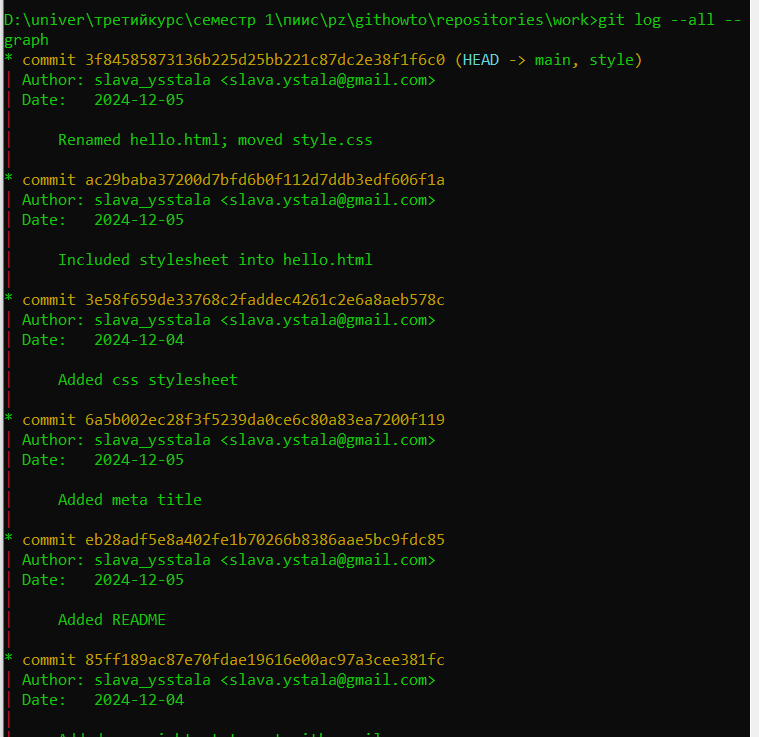
При ускоренном слиянии конфликты не возникают. Кроме того, при ускоренном слиянии не создается фиксация слияния.

1. **Просмотрите логи**

**Выполните**

git log --all --graph

**Результат**

****

$ git log --all --graph

\* 39a1e0f 2023-11-28 | Renamed hello.html; moved style.css (HEAD -> main, style) [Alexander Shvets]

\* 23149b5 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]

\* b9e6de1 2023-11-28 | Added css stylesheet [Alexander Shvets]

\* 85c14e9 2023-11-28 | Added meta title [Alexander Shvets]

\* ee16740 2023-11-28 | Added README [Alexander Shvets]

\* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]

\* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

\* 46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

\* 78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

\* 5836970 2023-11-28 | Initial commit [Alexander Shvets]

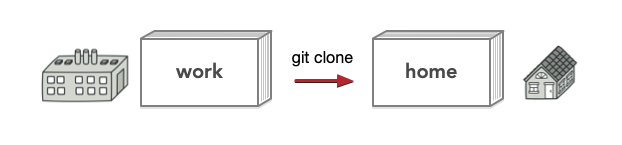
Теперь ветки style и main идентичны.

**Часть II: Несколько репозиториев**

До сих пор мы работали только с одним Git-репозиторием. Однако Git - это **распределенная** система управления версиями, а значит, она отлично подходит для работы с несколькими репозиториями. Эти дополнительные репозитории могут храниться локально, быть доступными через сеть или Интернет. Они также могут быть размещены на GitHub, GitLab, BitBucket или любом другом Git-хостинге.

В следующем разделе мы представим, что решили взять некоторую работу на дом. В былые времена можно было бы перенести этот репозиторий на флешку и взять его с собой домой. Сегодня мы, скорее всего, поделимся репозиторием через GitHub. На самом деле, не важно, как вы делитесь своей работой: Git будет работать одинаково. Большая часть информации, изложенной в этом разделе, может быть применена и для работы с несколькими репозиториями, независимо от того, хранятся ли они локально или передаются по сети.

Поэтому для простоты представим, что мы используем два независимых репозитория, располагая их локально в разных директориях: work и home.



В следующих уроках мы будем одновременно изменять обе копии наших репозиториев. Я буду подсвечивать шаги, на которых мы переключались на другой репозиторий, в блоках, аналогичных этому.

**30. Клонирование репозиториев**

**Цели**

* Научиться делать копии репозиториев.

Если вы работаете в команде, последующие 8 уроков довольно важны в понимании, т.к. вы почти всегда будете работать с клонированными репозиториями.

1. **Перейдите в директорию**repositories

**Выполните**

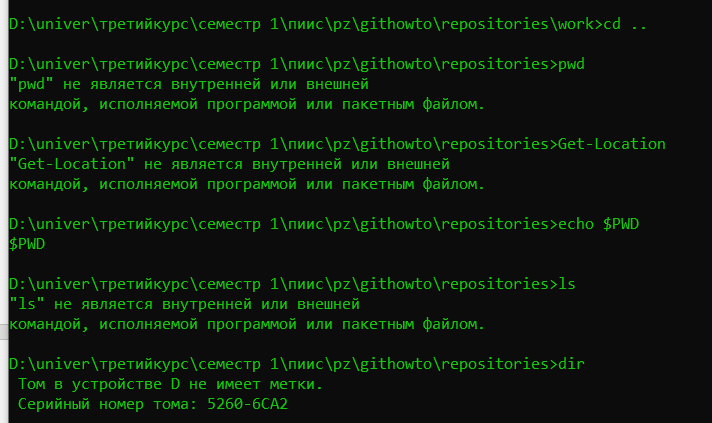
cd ..

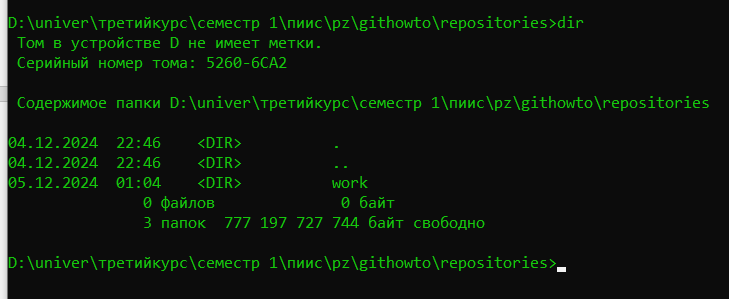
pwd

ls

Сейчас мы находимся в директории repositories.

**Результат**

****

****

$ cd ..

$ pwd

/home/alex/githowto/repositories

$ ls

work

В этот момент вы должны находиться в директории repositories. Здесь должен быть единственный репозиторий под названием work.

***02.* Создайте клон репозитория**work

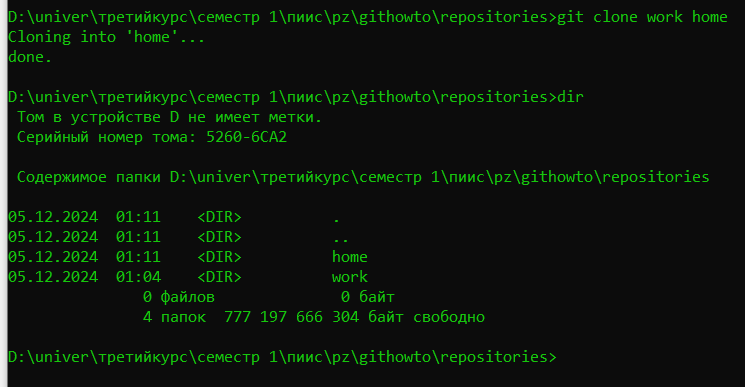
Давайте создадим клон репозитория.

**Выполните**

git clone work home

ls

**Результат**

****

$ git clone work home

Cloning into 'home'...

done.

$ ls

home

work

В вашем списке репозиториев теперь должно быть два репозитория: оригинальный репозиторий work и клонированный репозиторий home.

**31. Просмотр клонированного репозитория**

**Цели**

* Узнать о ветках в удаленных репозиториях.

1. **Посмотрите на клонированный репозиторий**

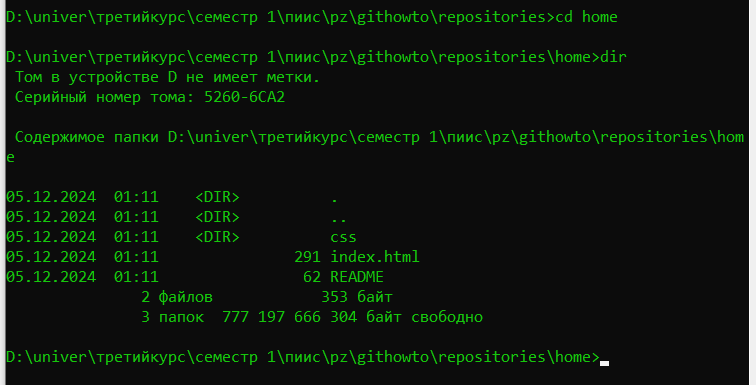
Давайте взглянем на клонированный репозиторий.

**Выполните**

cd home

ls

**Результат**

****

$ cd home

$ ls

css

index.html

README

Вы увидите список всех файлов на верхнем уровне оригинального репозитория README, index.html и css).

1. **Просмотрите историю репозитория**

**Выполните**

git log --all

**Результат**

****

$ git log --all --graph

\* 39a1e0f 2023-11-28 | Renamed hello.html; moved style.css (HEAD -> main, origin/style, origin/main, origin/HEAD) [Alexander Shvets]

\* 23149b5 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]

\* b9e6de1 2023-11-28 | Added css stylesheet [Alexander Shvets]

\* 85c14e9 2023-11-28 | Added meta title [Alexander Shvets]

\* ee16740 2023-11-28 | Added README [Alexander Shvets]

\* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]

\* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

\* 46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

\* 78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

\* 5836970 2023-11-28 | Initial commit [Alexander Shvets]

Вы увидите список всех коммитов в новый репозиторий, и он должен совпадать с историей коммитов в оригинальном репозитории. Единственная разница должна быть в названиях веток.

1. **Удаленные ветки**

Вы увидите ветку main (HEAD) в списке истории. Вы также увидите ветки со странными именами (origin/main, origin/style и origin/HEAD). Мы поговорим о них чуть позже.

**32. Что такое origin?**

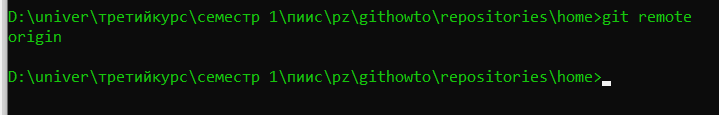
**Цели**

* Узнать об именах удаленных репозиториев.

**Выполните**

git remote

**Результат**

****

$ git remote

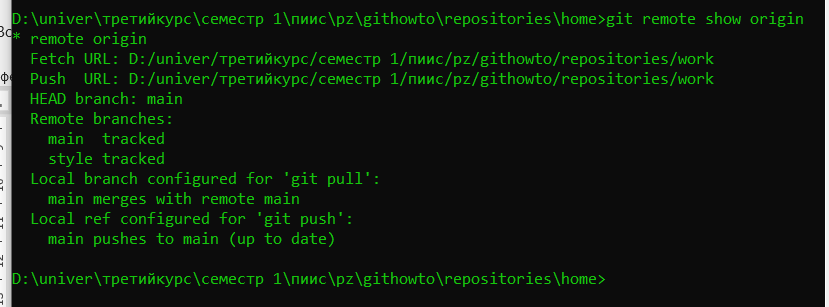
origin

Мы видим, что клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Давайте посмотрим, можем ли мы получить более подробную информацию об имени по умолчанию:

**Выполните**

git remote show origin

**Результат**

****

$ git remote show origin

\* remote origin

Fetch URL: /home/alex/githowto/repositories/work

Push URL: /home/alex/githowto/repositories/work

HEAD branch: main

Remote branches:

main tracked

style tracked

Local branch configured for 'git pull':

main merges with remote main

Local ref configured for 'git push':

main pushes to main (up to date)

Мы видим, что имя по умолчанию (origin) удаленного репозитория — изначальное work. Удаленные репозитории обычно размещаются на отдельной машине, возможно, централизованном сервере. Однако, как мы видим здесь, они могут с тем же успехом указывать на репозиторий на той же машине. Нет ничего особенного в имени origin, однако существует традиция использовать origin в качестве имени первичного централизованного репозитория (если таковой имеется).

**33. Удаленные ветки**

**Цели**

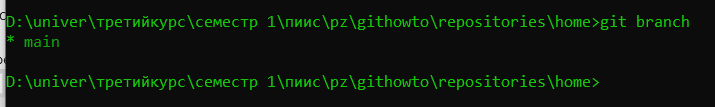
* Узнать о локальных и удаленных ветках.

Давайте посмотрим на ветки, доступные в нашем клонированном репозитории.

**Выполните**

git branch

**Результат**

****

$ git branch

\* main

Как мы видим, в списке только ветка main. Где ветка style? Команда git branch выводит только список локальных веток по умолчанию.

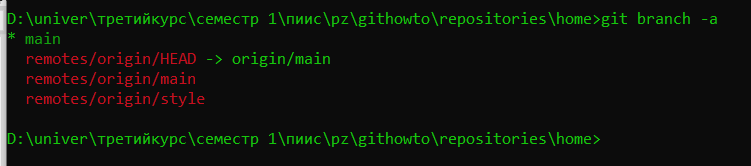
***01*Список удаленных веток**

Для того чтобы увидеть все ветки, попробуйте следующую команду:

**Выполните**

git branch -a

**Результат**

****

$ git branch -a

\* main

remotes/origin/HEAD -> origin/main

remotes/origin/style

remotes/origin/main

Git выводит все коммиты в оригинальный репозиторий, но ветки в удаленном репозитории не рассматриваются как локальные. Если мы хотим иметь собственную ветку style, мы должны сами ее создать. Через минуту вы увидите, как это делается.

**34. Изменение оригинального репозитория**

**Цели**

* Внести некоторые изменения в оригинальный репозиторий, чтобы затем попытаться подтянуть и слить изменения из удаленной ветки в текущую.

1. **Внесите изменения в оригинальный репозиторий**work

**Выполните**

cd ../work

Сейчас мы находимся в репозитории work.

Внесите следующие изменения в файл README:

**Файл: README**

This is the Hello World example from the Git tutorial.

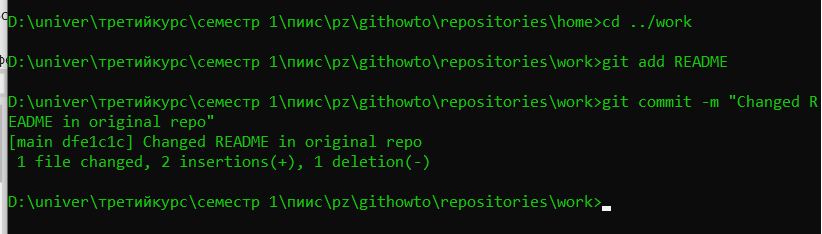
(changed in origin)

Теперь добавьте это изменение и сделайте коммит

**Выполните**

git add README

git commit -m "Changed README in original repo"



1. **Далее**

Теперь в оригинальном репозитории есть более поздние изменения, которых нет в клонированной версии. Далее мы подтянем и сольем эти изменения в клонированный репозиторий.

**35. Подтягивание изменений**

**Цели**

* Научиться подтягивать изменения из удаленного репозитория.

**Выполните**

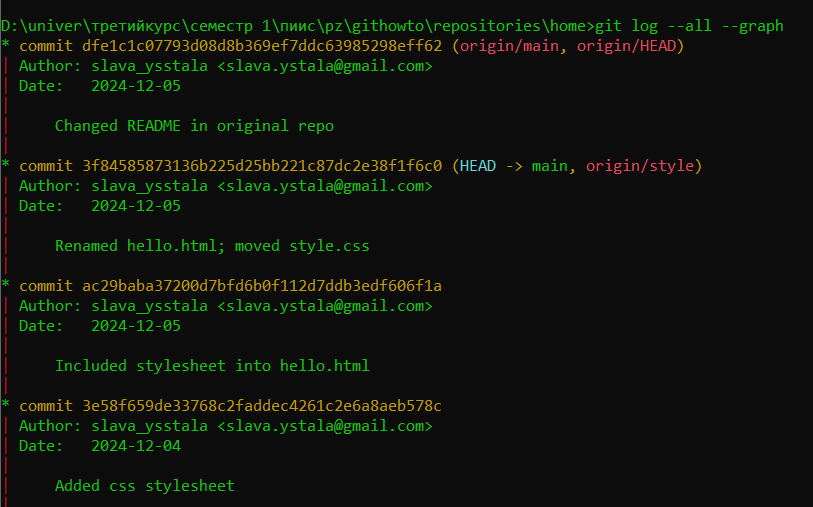
cd ../home

git fetch

git log --all

Сейчас мы находимся в репозитории home.

**Результат**

****

$ cd ../home

$ git fetch

From /home/alex/githowto/repositories/work

39a1e0f..71df43a main -> origin/main

$ git log --all --graph

\* 71df43a 2023-11-28 | Changed README in original repo (origin/main, origin/HEAD) [Alexander Shvets]

\* 39a1e0f 2023-11-28 | Renamed hello.html; moved style.css (HEAD -> main, origin/style) [Alexander Shvets]

\* 23149b5 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]

\* b9e6de1 2023-11-28 | Added css stylesheet [Alexander Shvets]

\* 85c14e9 2023-11-28 | Added meta title [Alexander Shvets]

\* ee16740 2023-11-28 | Added README [Alexander Shvets]

\* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]

\* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]

\* 46afaff 2023-11-28 | Added standard HTML page tags (tag: v1-beta) [Alexander Shvets]

\* 78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

\* 5836970 2023-11-28 | Initial commit [Alexander Shvets]

На данный момент в репозитории есть все коммиты из оригинального репозитория, но они не интегрированы в локальные ветки клонированного репозитория.

В истории выше найдите коммит «Changed README in original repo». Обратите внимание, что коммит включает в себя коммиты origin/main и origin/HEAD.

Теперь давайте посмотрим на коммит «Renamed hello.html; moved style.css». Вы увидите, что локальная ветка main указывает на этот коммит, а не на новый коммит, который мы только что подтянули.

Выводом является то, что команда git fetch будет подтягивать новые коммиты из удаленного репозитория, но не будет сливать их с вашими наработками в локальных ветках.

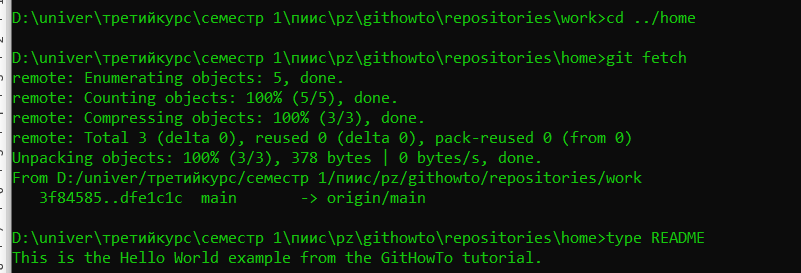
* 1. **Проверьте README**

Мы можем продемонстрировать, что клонированный файл README не изменился.

**Выполните**

type README

**Результат**

****

$ cat README

This is the Hello World example from the GitHowTo tutorial.

Как видите, никаких изменений.

**36. Слияние подтянутых изменений**

**Цели**

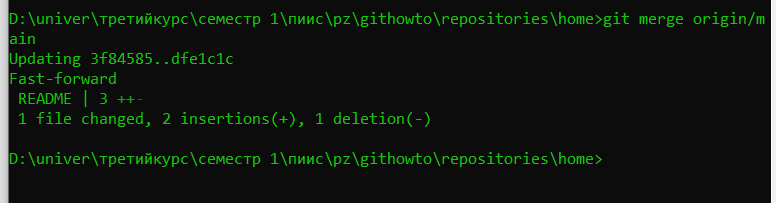
* Научиться сливать подтянутые изменения в текущую ветку.

1. **Слейте подтянутые изменения в локальную ветку**main

**Выполните**

git merge origin/main

**Результат**

****

$ git merge origin/main

Updating 39a1e0f..71df43a

Fast-forward

README | 3 ++-

1 file changed, 2 insertions(+), 1 deletion(-)

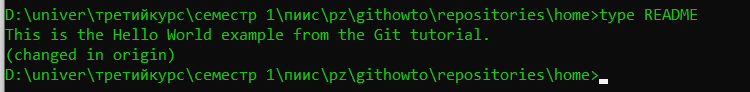
1. **Еще раз проверьте файл**README

Сейчас мы должны увидеть изменения.

**Выполните**

cat README

**Результат**

****

$ cat README

This is the Hello World example from the Git tutorial.

(changed in origin)

Вот и изменения. Хотя команда git fetch не сливает изменения, мы можем вручную слить изменения из удаленного репозитория.

1. **Команда**pull**(подтянуть)**

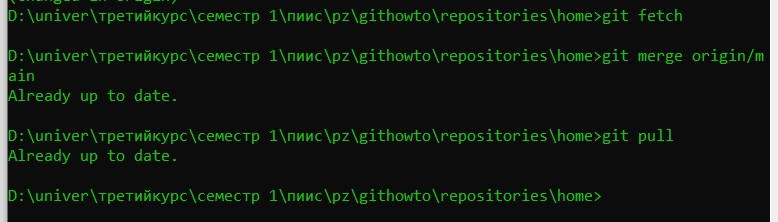
Команда fetch позволяет контролировать то, что именно подтягивается и сливается в ваши локальные ветки, но для удобства существует также команда pull, которая подтягивает и сливает изменения из удаленной ветки в текущую одним вызовом.

git pull

...эквивалентна следующим двум шагам:

git fetch

git merge origin/main



**37. Добавление ветки наблюдения**

**Цели**

* Научиться добавлять локальную ветку, которая отслеживает изменения удаленной ветки.

Ветки, которые начинаются с remotes/origin являются ветками оригинального репозитория. Обратите внимание, что у вас больше нет ветки под названием style, но система контроля версий знает, что в оригинальном репозитории ветка style была.

1. **Добавьте локальную ветку, которая отслеживает удаленную ветку**

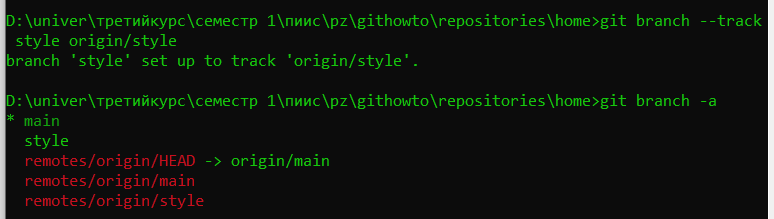
**Выполните**

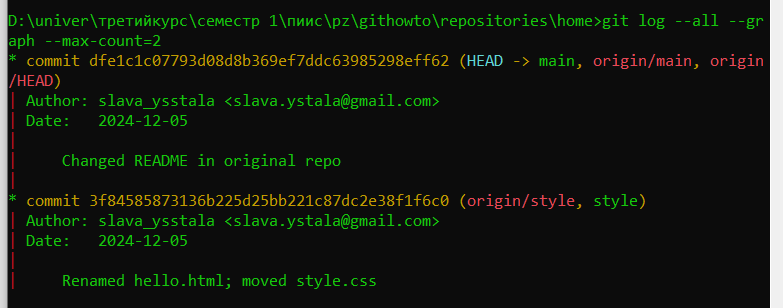
git branch --track style origin/style

git branch -a

git log --max-count=2

**Результат**

****

****

$ git branch --track style origin/style

Branch 'style' set up to track remote branch 'style' from 'origin'.

$ git branch -a

\* main

style

remotes/origin/HEAD -> origin/main

remotes/origin/main

remotes/origin/style

$ git log --all --graph --max-count=2

\* 71df43a 2023-11-28 | Changed README in original repo (HEAD -> main, origin/main, origin/HEAD) [Alexander Shvets]

\* 39a1e0f 2023-11-28 | Renamed hello.html; moved style.css (origin/style, style) [Alexander Shvets]

Теперь мы можем видеть ветку style в списке веток и логе.

**38. Чистые репозитории**

**Цели**

* Научиться создавать чистые репозитории.

Чистый репозиторий — это репозиторий, не имеющий рабочей директории. Он содержит только директорию .git, в которой Git хранит все свои внутренние данные. Основное предназначение таких репозиториев — быть центральным хранилищем, в которое разработчики могут отправлять и из которого могут получать данные. Поэтому в них нет смысла создавать рабочие файлы, они будут только впустую занимать место на диске. Чистые репозитории также используются в сервисах Git-хостинга таких, как GitHub и GitLab. В следующих уроках мы узнаем, как создать чистый репозиторий и как отправлять в него изменения.

***01*Создайте чистый репозиторий**

**Выполните**

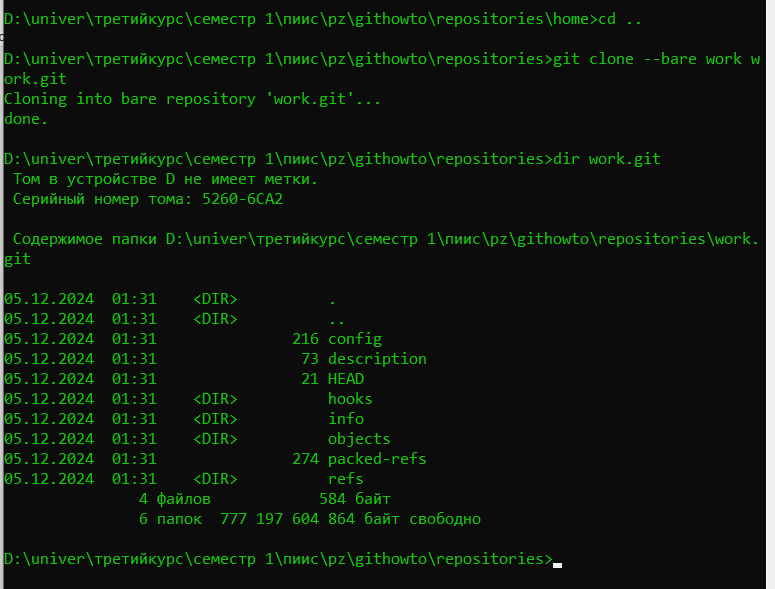
cd ..

git clone --bare work work.git

ls work.git

Сейчас мы находимся в директории repositories.

**Результат**

****

$ git clone --bare work work.git

Cloning into bare repository 'work.git'...

done.

$ ls work.git

branches

config

description

HEAD

hooks

info

objects

packed-refs

refs

Принято считать, что репозитории, заканчивающиеся на .git, являются чистыми репозиториями. Мы видим, что в репозитории work.git нет рабочей директории. По сути, это просто директория .git из обычного репозитория.

**39. Добавление удаленного репозитория**

**Цели**

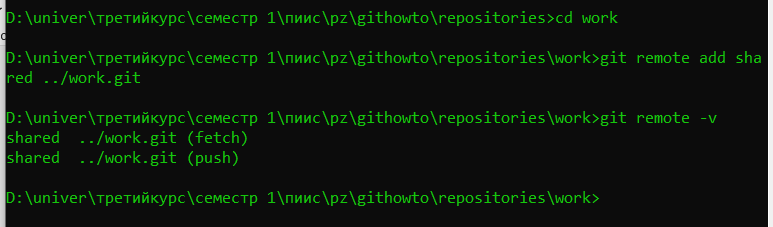
* Добавить чистый репозиторий в качестве удаленного репозитория к нашему оригинальному репозиторию.

Давайте добавим репозиторий work.git к нашему оригинальному репозиторию.

**Выполните**

cd work

git remote add shared ../work.git



Сейчас мы находимся в репозитории work.

**40. Отправка изменений**

**Цели**

* Узнать, как передавать изменения в удаленный репозиторий.

Поскольку чистые репозитории обычно располагаются на каком-либо удаленном сервере, вы не сможете туда просто зайти, дабы подтянуть изменения. Поэтому нам необходимо как-нибудь передать наши изменения в репозиторий.

Начнем с создания изменения, которое нужно передать в репозиторий. Отредактируйте README и закоммитьте его:

**Файл: README**

This is the Hello World example from the Git tutorial.

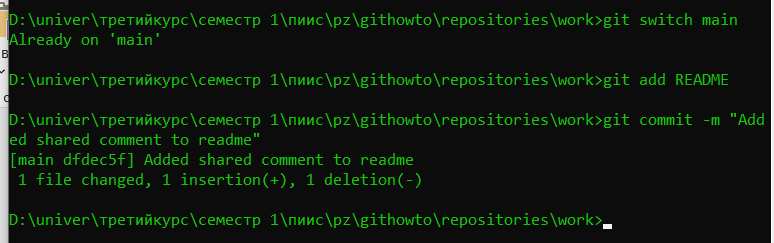
(changed in the origin and pushed to shared)

**Выполните**

git switch main

git add README

git commit -m "Added shared comment to readme"



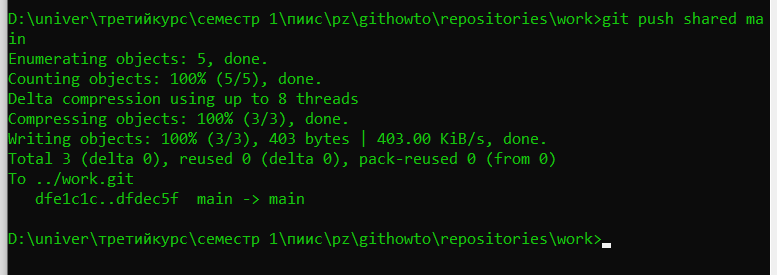
Теперь отправьте изменения в общий репозиторий.

**Выполните**

git push shared main

*Общим* называется репозиторий, получающий отправленные нами изменения. Помните, мы добавили его в качестве удаленного репозитория в предыдущем уроке?

**Результат**

****

$ git push shared main

To ../work.git

71df43a..d9d2bea main -> main

**41. Подтягивание общих изменений**

**Цели**

* Научиться подтягивать изменения из общего репозитория.

Быстро переключитесь в репозиторий home и подтяните изменения, только что отправленные в общий репозиторий.

**Выполните**

cd ../home

Сейчас мы находимся в репозитории home.

Продолжите с...

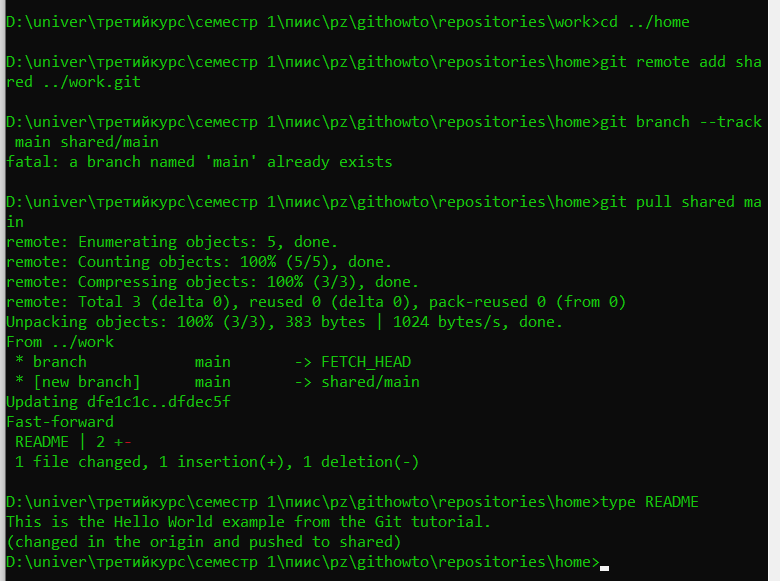
**Выполните**

git remote add shared ../work.git

git branch --track shared main

git pull shared main

cat README



**42. Размещение ваших Git-репозиториев**

Поздравляю с тем, что вы дошли до этого момента! Пожалуйста, считайте эту тему бонусной.

**Цели**

* Научиться настраивать Git-сервер для совместного использования репозиториев.

Хотите создать свой собственный GitHub? Существует множество способов совместного использования репозиториев Git по сети. Здесь приведен простой и быстрый (но ненадежный и опасный) способ.

1. **Запуск Git-сервера**

**Выполните**

# (From the "repositories" directory)

git daemon --verbose --export-all --base-path=.

Теперь в отдельном окне терминала перейдите в вашу директорию repositories:

**Выполните**

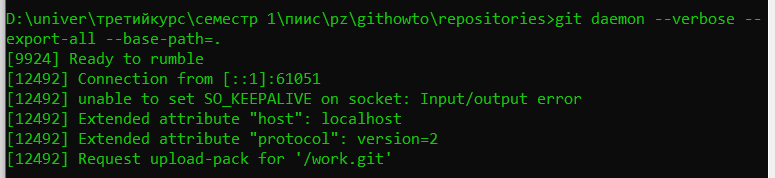
# (From the "repositories" directory)

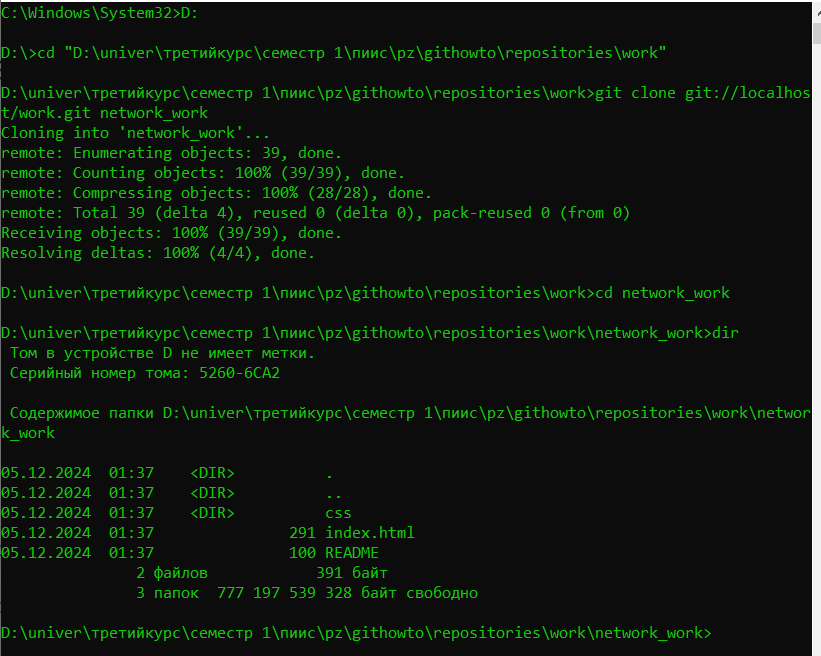
git clone git://localhost/work.git network\_work

cd network\_work

ls

Вы увидите копию проекта work.





1. **Отправка изменений в Git Daemon**

Если вы хотите разрешить отправку изменений (push) в репозиторий Git Daemon, добавьте метку --enable=receive-pack к команде git daemon. Будьте осторожны, этот сервер не производит аутентификацию, поэтому любой сможет отправлять изменения в ваш репозиторий.

1. **Совместное использование репозиториев**

На этом этапе вам открываются безграничные возможности. Смелее! Возьмите в аренду сервер, купите доменное имя, разместите на этом сервере свои репозитории и наслаждайтесь своим личным GitHub!

Если серьезно, то вы можете самостоятельно разместить свой личный сервер [GitLab](https://gitlab.com/). Этот продукт бесплатный и с открытым исходным кодом.