Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра инженерной психологии и эргономики

Дисциплина: Пользовательские интерфейсы информационных систем

Отчет

по практическому занятию

на тему:

«Git»

Проверил: Выполнил:

Студент группы 210901

Давыдович К.И. Скрипов И.О.

Минск 2024

**Терминология**

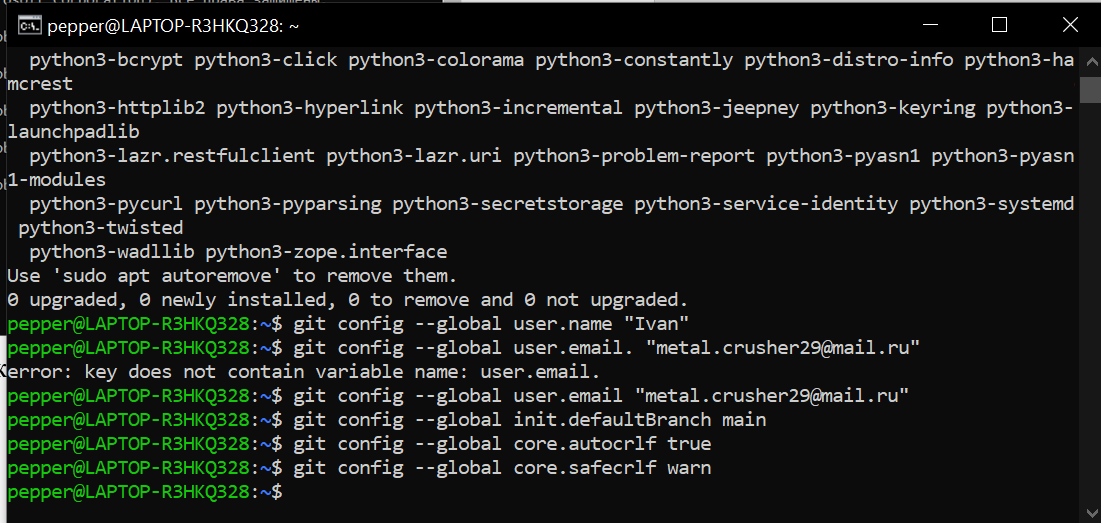
***Репозиторий Git*** — это хранилище, в котором расположен ваш проект и его история.

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.

***Ветка*** — это параллельная версия репозитория.

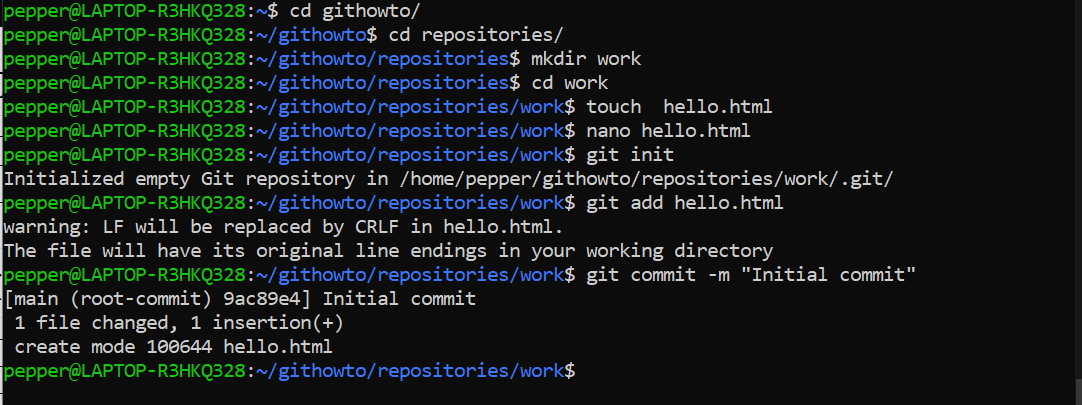
**Подготовка**

|  |  |
| --- | --- |
| git config --global user.name "Your Name"  git config --global user.email "your\_email@whatever.com" | Git узнал ваше имя и электронную почту |
| git config --global init.defaultBranch main | Имени ветки по умолчанию. Чтобы установить его |
| git config --global core.autocrlf true  git config --global core.safecrlf warn | Корректная обработка окончаний строк |



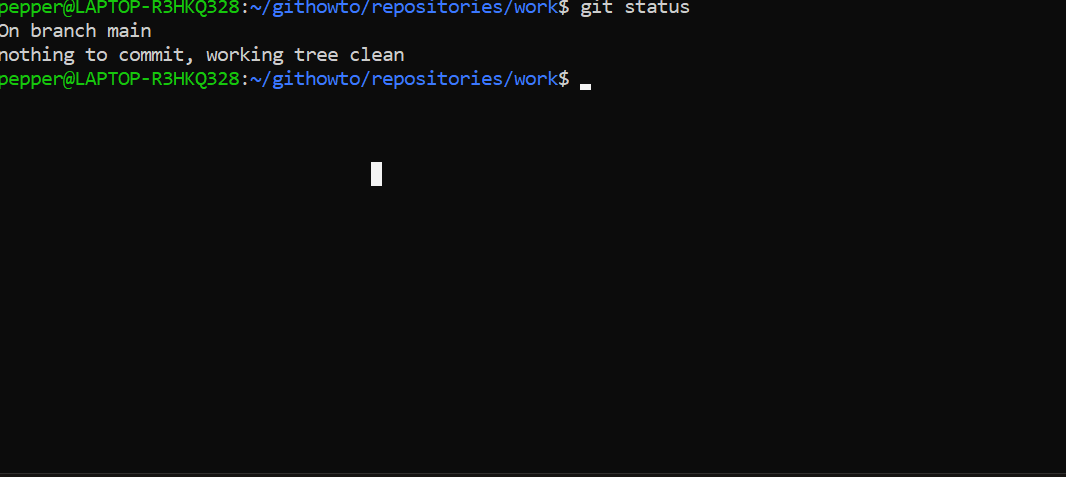
# 2. Создание проекта

|  |  |
| --- | --- |
| git init | Создайте репозиторий |
| git add hello.html  git commit -m "Initial Commit" | добавим в репозиторий |



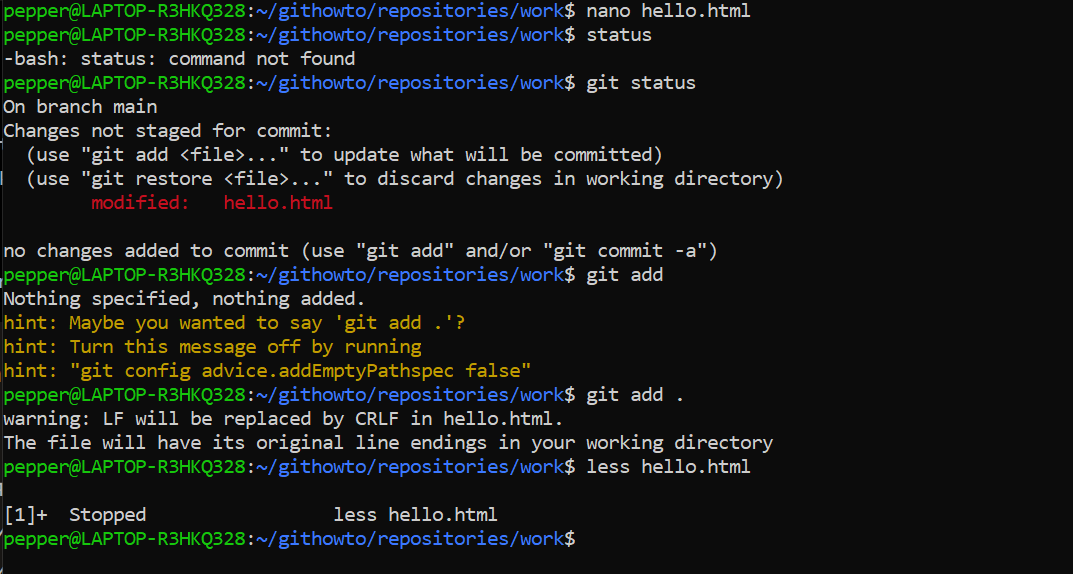
# 3. Проверка состояния

|  |  |
| --- | --- |
| git status | проверить текущее состояние репозитория. |
| git branch -m master main | Можете переименовать ветку в main |



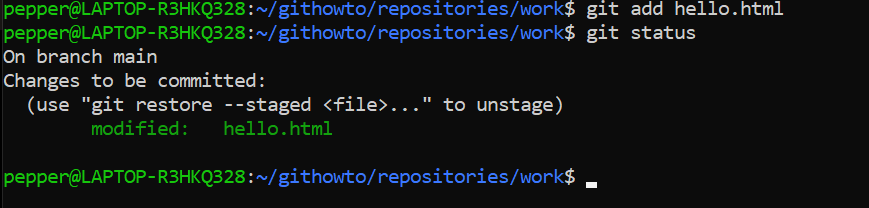
# 4. Внесение изменений

|  |  |
| --- | --- |
| git add | добавить эти изменения в репозиторий |
| git сheckout | для отмены изменений. |



# 5. Индексация изменений

|  |  |
| --- | --- |
| git add hello.html  git status | проиндексировать изменения |
| git reset | можно снять индексацию этих изменений |



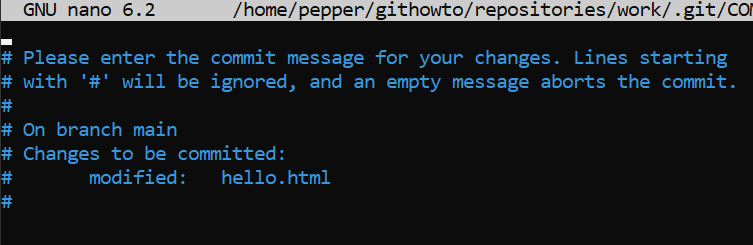
# 6. Индексация и коммит

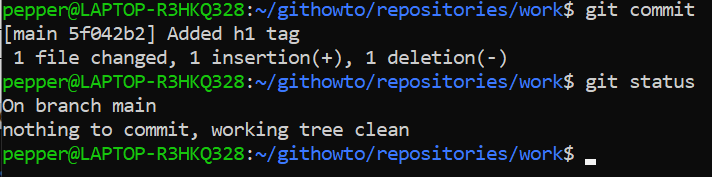
|  |  |
| --- | --- |
| git add a.html  git add b.html  git commit -m "Changes for a and b" | при этом чтобы изменения в a.html и b.html были одним коммитом, |
| git add c.html  git commit -m "Unrelated change to c" | в то время как изменения в c.html логически не связаны с первыми двумя файлами и должны идти отдельным коммитом |

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

# 7. Коммит изменений

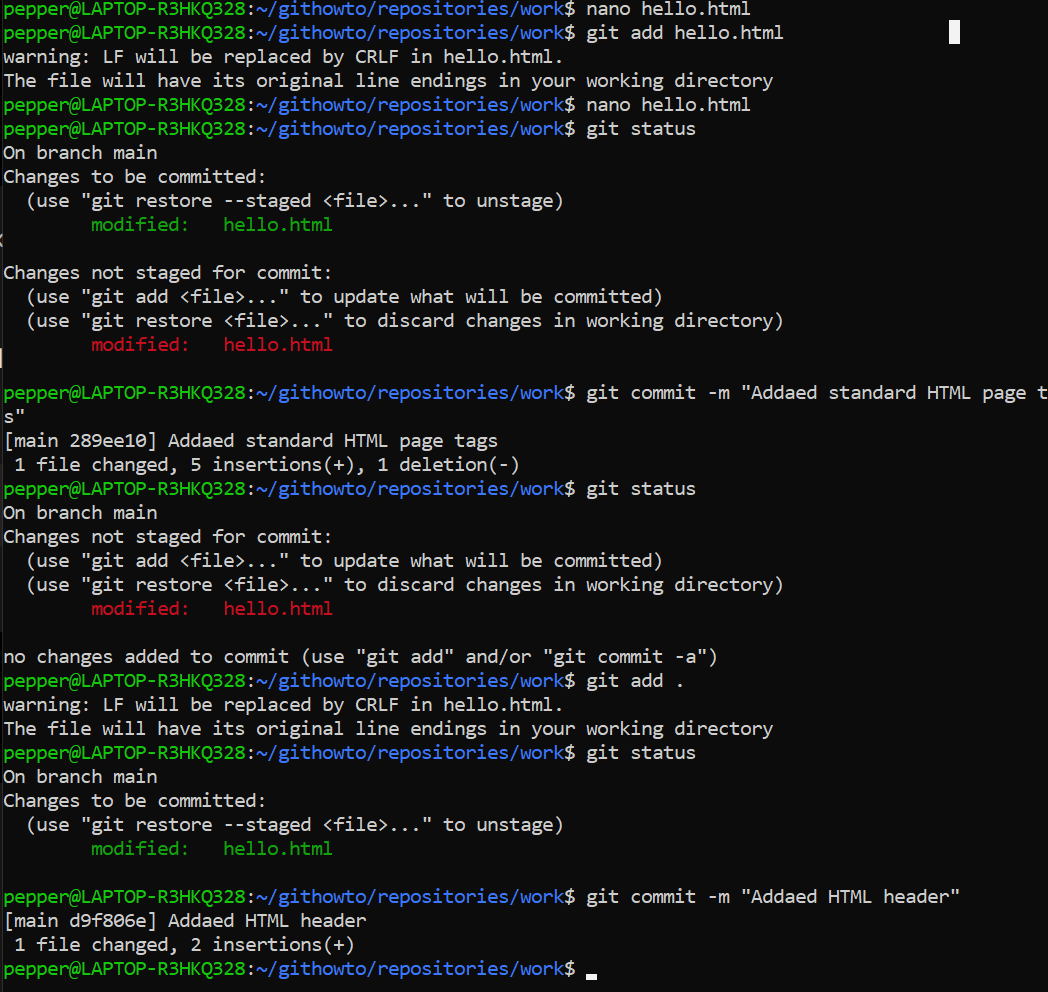
|  |  |
| --- | --- |
| git commit | позволит вам интерактивно редактировать комментарии для коммита. Git перенесет вас в редактор по вашему выбору. |
| :wq | Сохраните файл и выйдите из редактора (для этого в редакторе по умолчанию (Vim) вам нужно нажать клавишу ESC, ввести **:wq** и нажать Enter). |
| git status | еще раз проверим состояние. |





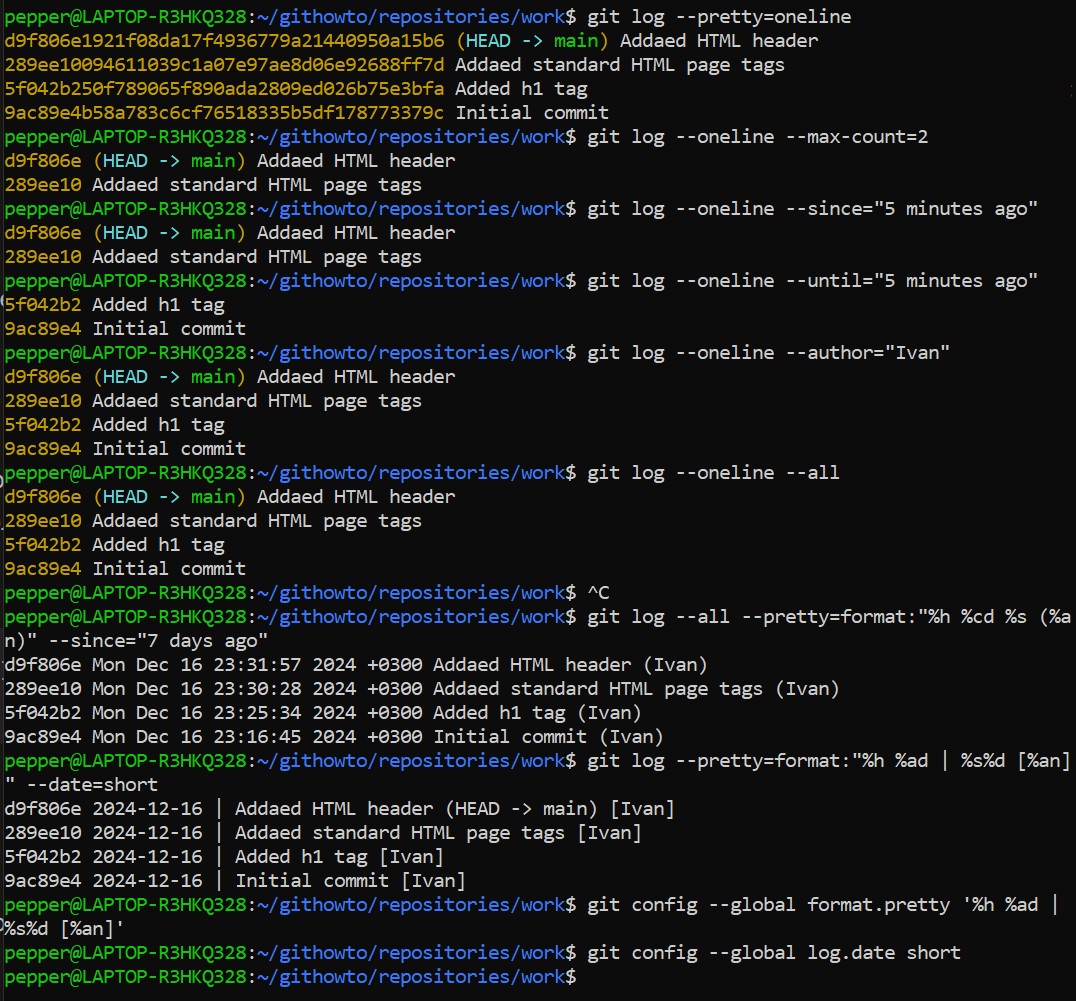
# 8. Изменения, а не файлы

|  |  |
| --- | --- |
| Первое изменение | |
| git add hello.html | Теперь добавьте это изменение в индекс Git |
| Второе изменение | |
| git status | Проверьте текущий статус ервое 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. |
| git commit -m "Added standard HTML page tags"  git status | произведите коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверьте состояние. Команда status показывает, что в файле hello.html ещё есть незаписанные изменения, но область подготовки уже пуста. |
| git add .  git status | Теперь добавьте второе изменение в индекс, а затем проверьте состояние с помощью команды git status. Мы использовали текущую директорию (.) в качестве аргумента для добавления. |
| git commit -m "Added HTML header" | Сделайте коммит второго изменения |



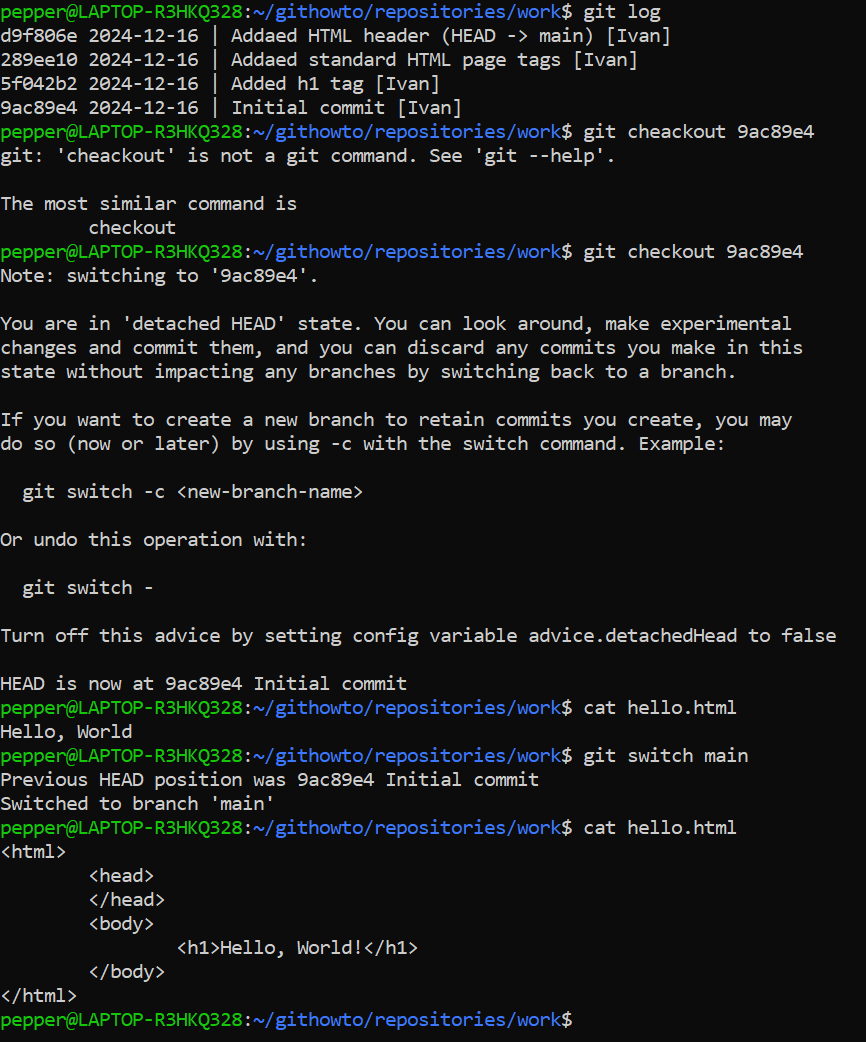
# 9. История

|  |  |
| --- | --- |
| git log | Получение списка произведенных изменений |
| git log --pretty=oneline | однострочный формат: |
| 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 | интересные варианты просмотра истории: |
| git log --pretty=format:"%h %ad | %s%d [%an]" --date=short | Со временем, я решил, что для большей части моей работы мне подходит следующий формат лога. |
| git config --global format.pretty '%h %ad | %s%d [%an]'  git config --global log.date short | настроить формат вывода истории по умолчанию: |



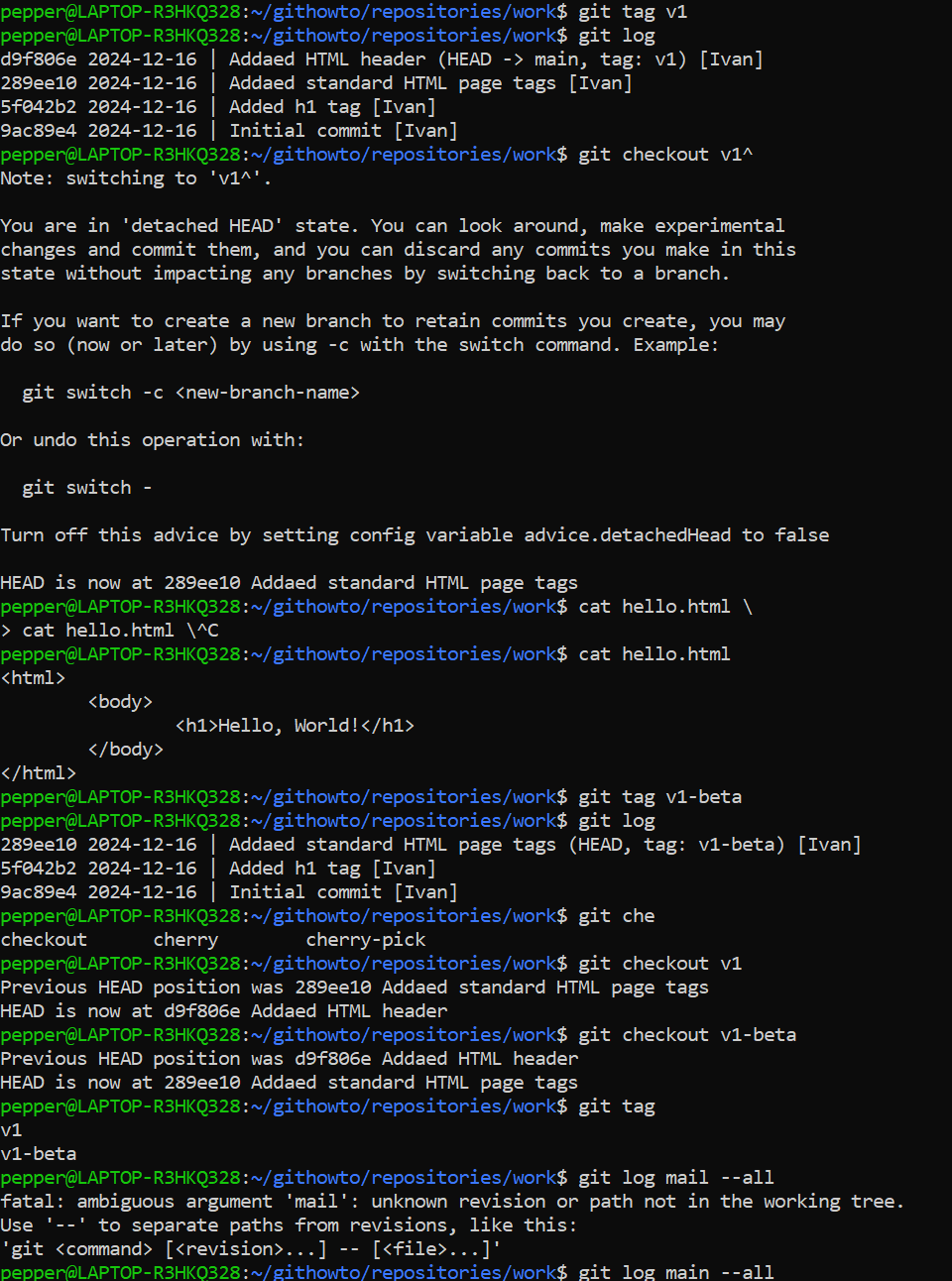
# 10. Получение старых версий

|  |  |
| --- | --- |
| git log | Получите хеши предыдущих коммитов |
| git checkout <hash>  cat hello.html | Используйте этот хеш (достаточно первых 7 символов) в команде ниже. Затем проверьте содержимое файла hello.html. Обратите внимание, что сейчас содержимое файла hello.html — это тот самый текст, с которого мы начинали. |
| git switch main  cat hello.html | Чтобы вернуться к последней версии нашего кода, нам нужно переключиться на ветку по умолчанию, main. Для переключения между ветками можно воспользоваться командой switch. |



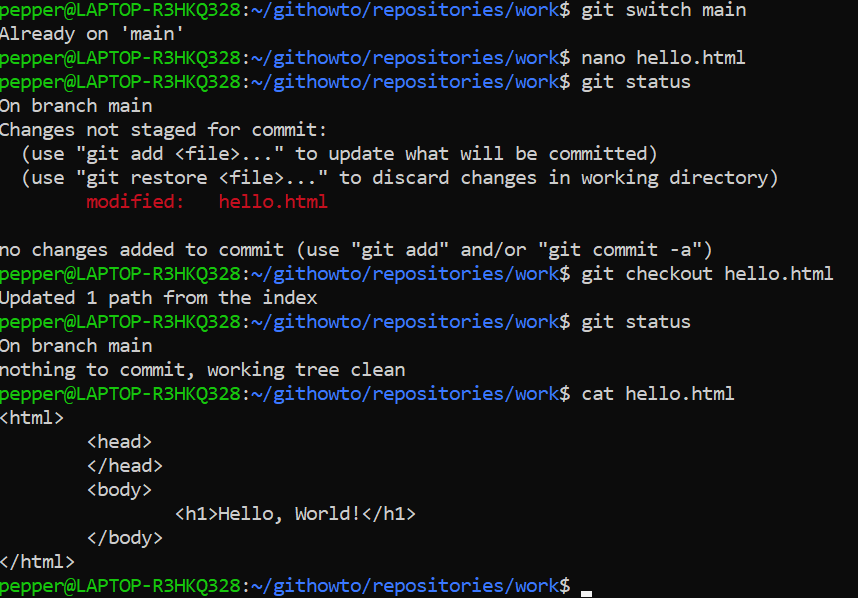
# 11. Создание тегов версий

|  |  |
| --- | --- |
| git tag v1  git log | Создайте тег первой версии |
| git checkout v1~1(…^1)  cat hello.html | 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. |
| git checkout v1  git checkout v1-beta | Переключение по имени тега |
| git tag | Просмотр тегов с помощью команды tag |



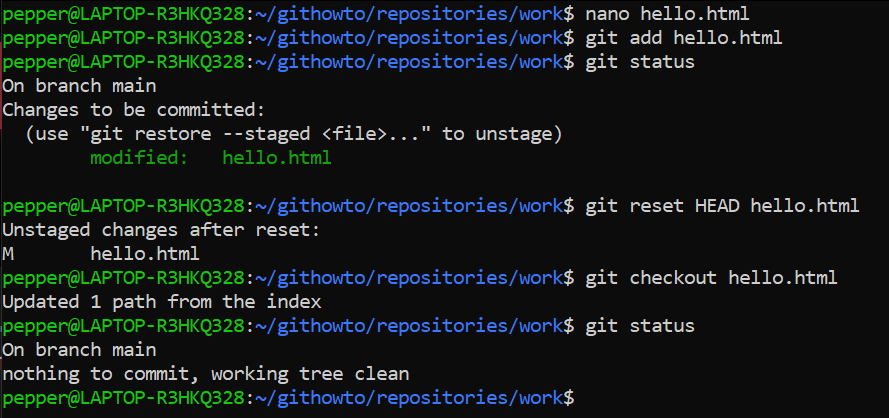
# 12. Отмена локальных изменений (до индексации)

|  |  |
| --- | --- |
| git switch main  git status | Убедитесь, что вы находитесь на последнем коммите ветки main, прежде чем продолжить работу. Сначала проверьте состояние рабочей директории. |
| git checkout hello.html  git status  cat hello.html | Используйте команду checkout для переключения в версию файла hello.html в репозитории. |



# 13. Отмена проиндексированных изменений (перед коммитом)

|  |  |
| --- | --- |
| git status | Проверьте состояние |
| git reset HEAD hello.html | Команда reset сбрасывает область подготовки к HEAD. Это очищает область подготовки от изменений, которые мы только что проиндексировали. |
| git checkout hello.html  git status | Переключитесь на версию коммита |

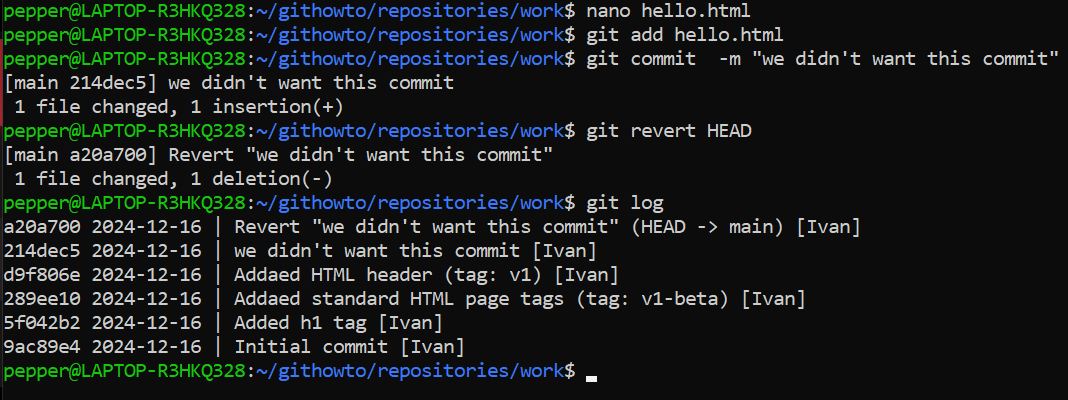


# 14. Отмена коммитов

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.

|  |  |
| --- | --- |
| git revert HEAD | Перейдите в редактор, где вы можете отредактировать коммит-сообщение по умолчанию или оставить все как есть. Сохраните и закройте файл. 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. |
| git log | Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий. |



# 15. Удаление коммитов из ветки

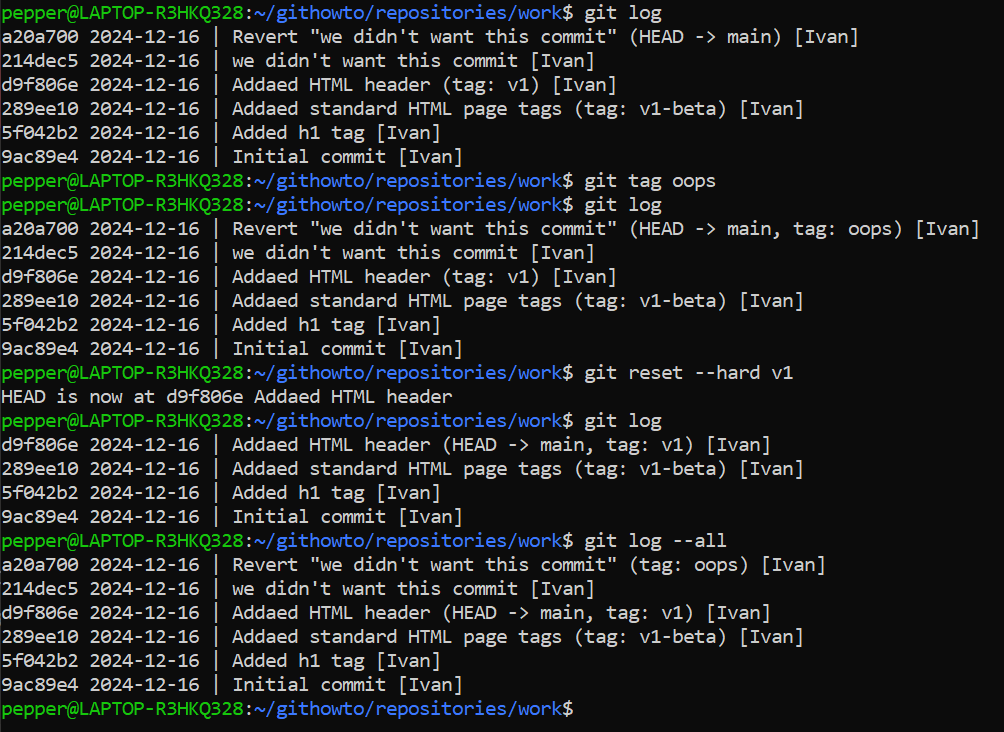
## Команда reset

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.

|  |  |
| --- | --- |
| git log | Проверьте нашу историю |
| git reset --hard v1  git log | 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 |
| git log --all | Оказывается, что коммиты все еще находятся в репозитории. 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. |

****

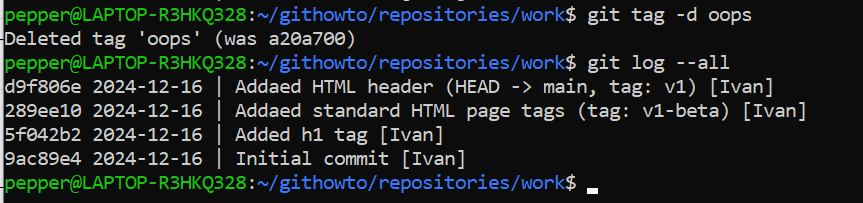
**Опасность сброса**

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.

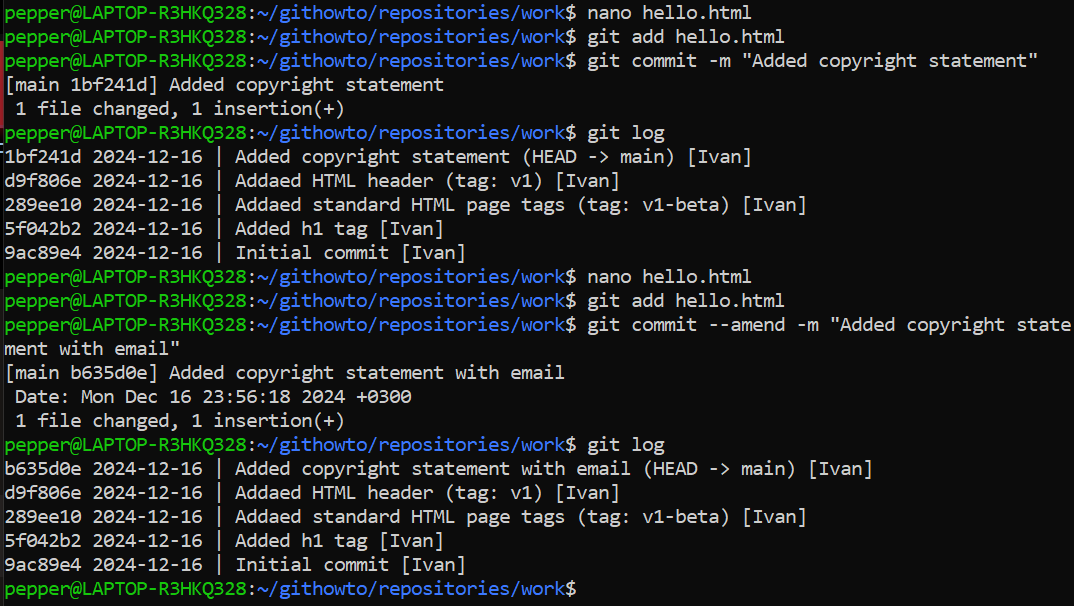
# 16. Удаление тега **oops**

|  |  |
| --- | --- |
| git tag -d oops  git log --all | 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. |



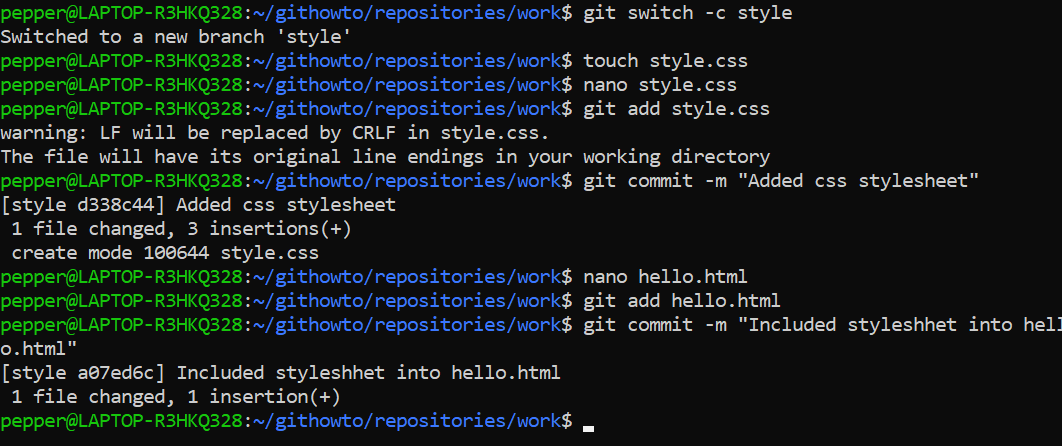
# 17. Внесение изменений в коммиты

|  |  |
| --- | --- |
| git add hello.html  git commit -m "Added copyright statement"  git log | Измените страницу, а затем сделайте коммит |
| git add hello.html  git commit --amend -m "Added copyright statement with email" | Мы действительно не хотим создавать отдельный коммит только ради электронной почты. Давайте изменим предыдущий коммит, включив в него адрес электронной почты. |
| git log | Просмотр истории |



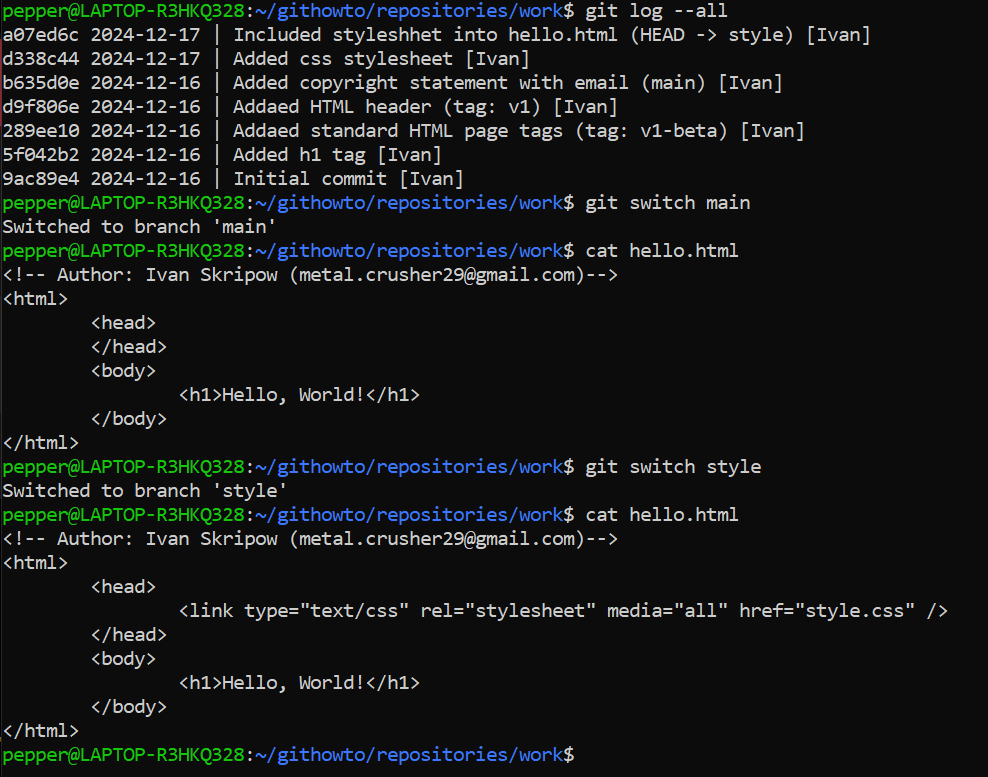
# 18. Создание ветки

|  |  |
| --- | --- |
| git switch -c style  git status | Создайте ветку Пришло время сделать нашу страницу более стильной с помощью CSS. Мы будем развивать эту возможность в новой ветке под названием style.  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. |



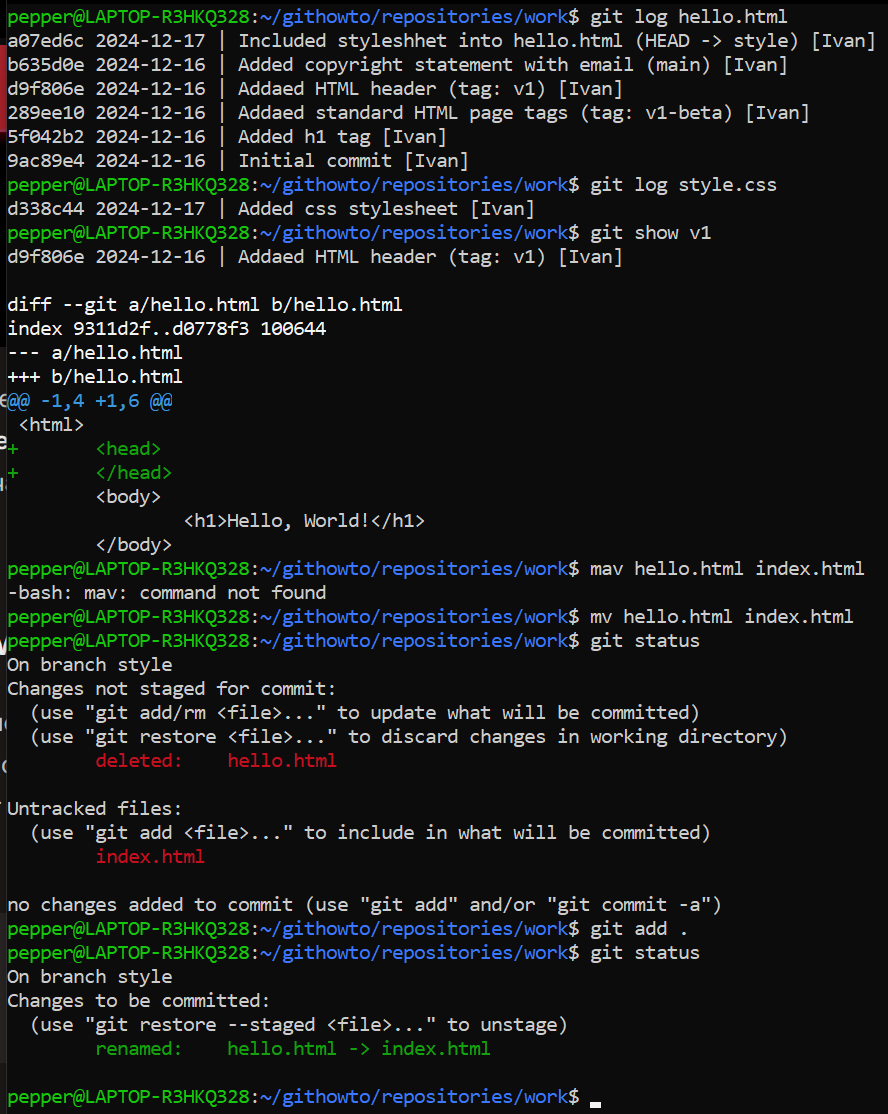
# 19. Переключение веток

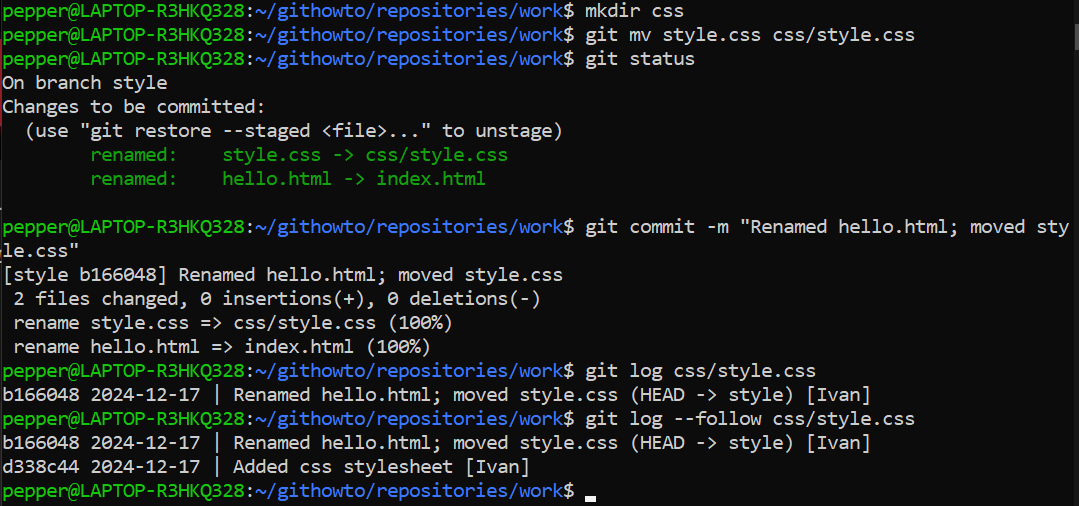
|  |  |
| --- | --- |
| git log --all | Теперь в вашем проекте есть две ветки: |
| git switch main  cat hello.html | Просто используйте команду git switch для переключения между ветками. |
| git switch style  cat hello.html | Вернемся к ветке style |



# 20. Перемещение файлов

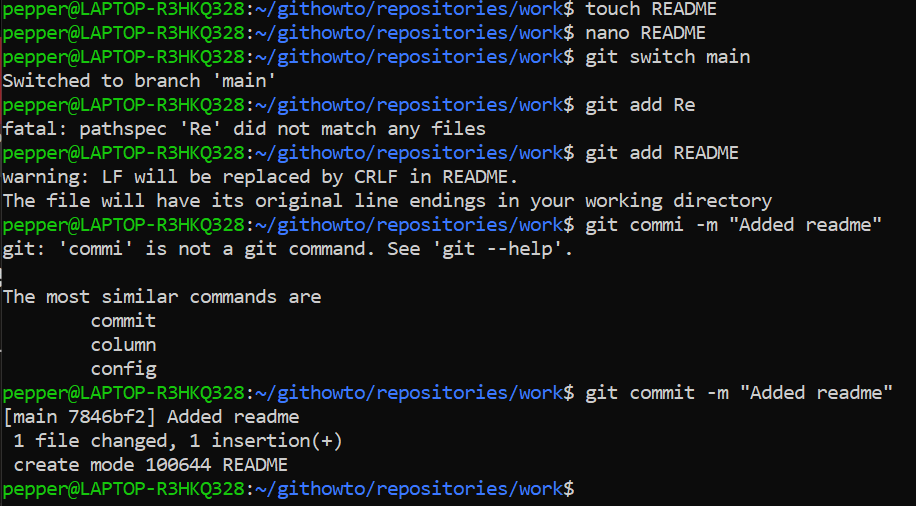
|  |  |
| --- | --- |
| git log hello.html  git log style.css | Git позволяет просматривать историю изменений конкретного файла. Давайте посмотрим историю изменений файла hello.html перед его переименованием. |
| git show v1 | Команда show используется для просмотра изменений в конкретном коммите. Посмотрим 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.). |
| mv hello.html index.html  git status | Давайте переименуем наш файл hello.html в index.html с помощью стандартной команды mv и посмотрим, что из этого получится. |
| git add .  git status | 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: идите, файл указан как переименованный. Но это всего лишь Git пытается проявить смекалку. Это не всегда работает. |
| mkdir css  git mv style.css css/style.css  git status | 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. |
| git commit -m "Renamed hello.html; moved style.css"  git log css/style.css  git log --follow css/style.css | 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. |





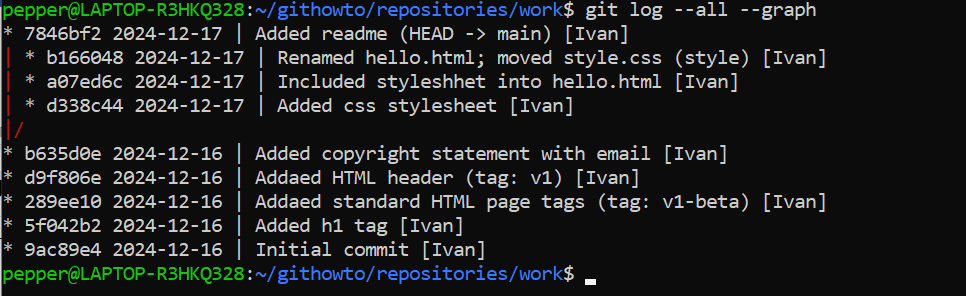
# 21. Изменения в ветке **main**

|  |  |
| --- | --- |
| git switch main  git add README  git commit -m "Added README" | В настоящее время мы находимся в ветке style. Файл README не является частью этой ветки, поэтому перед коммитом мы должны переключиться на ветку main. |



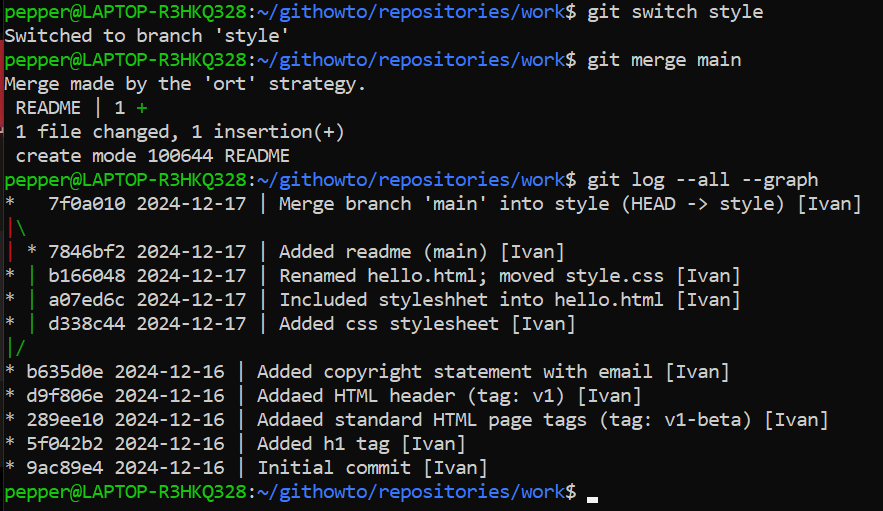
# 22. Просмотр отличающихся веток

|  |  |
| --- | --- |
| git log --all --graph | 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». |



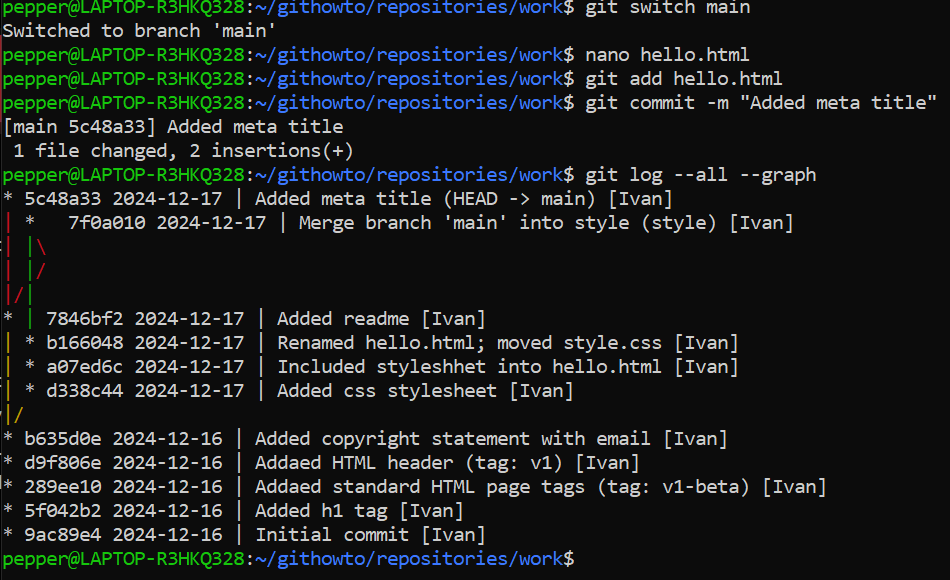
# 23. Слияние

|  |  |
| --- | --- |
| git switch style  git merge main  git log --all --graph | Влияние переносит изменения из двух веток в одну. Давайте вернемся к ветке style и сольем main со style. 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. |



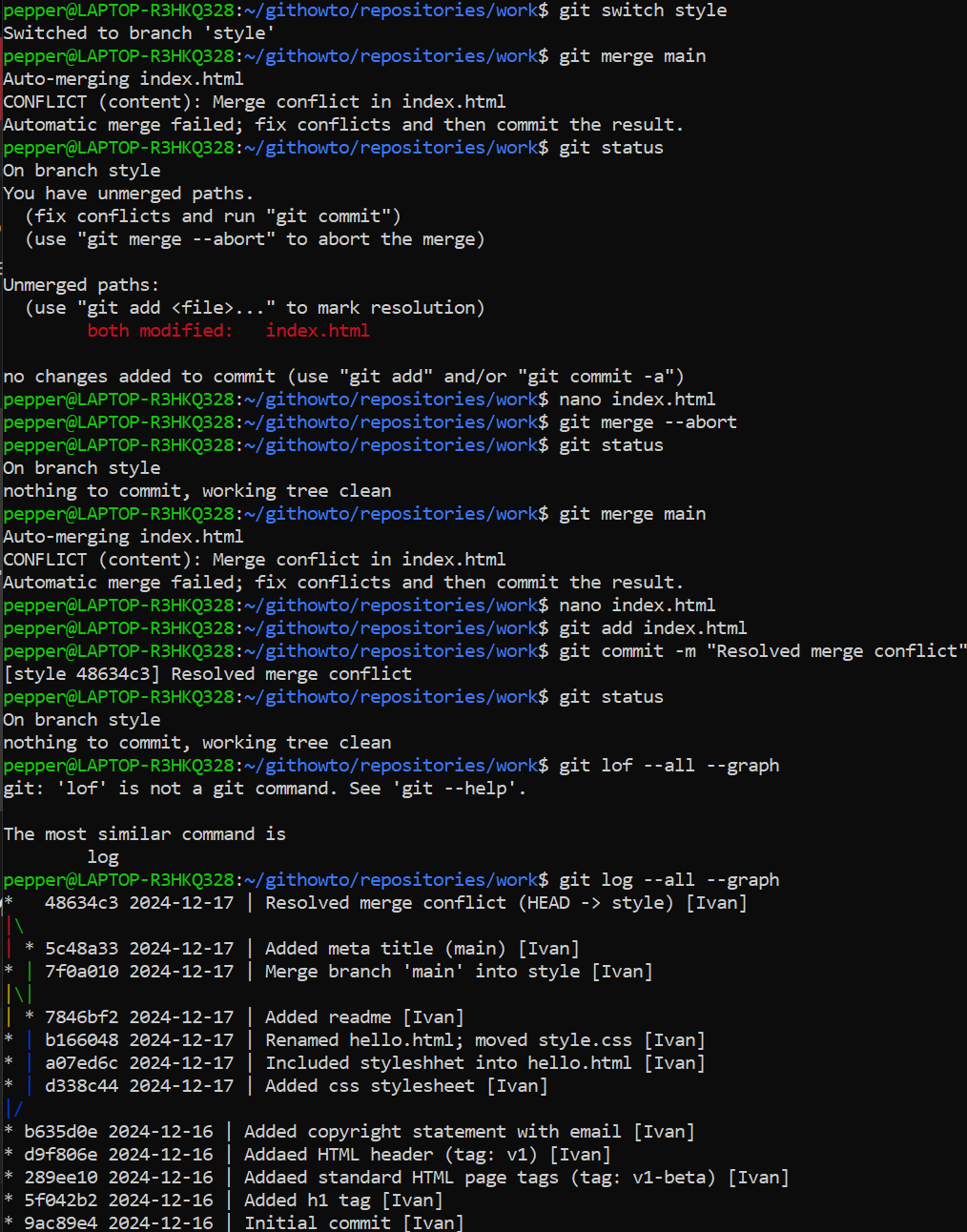
# 24. Создание конфликта

*Тут мы создали конфликт*



# 25. Разрешение конфликтов

|  |  |
| --- | --- |
| git switch style  git merge main | Давайте вернемся в ветку style и сольем в нее все последние изменения из ветки main. |
| git status | Похоже, что у нас возник конфликт. Ничего удивительного! Посмотрим, что скажет по этому поводу Git: |
| git merge --abort  git status | 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. |
| git merge main | Давайте снова запустим объединение. 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. |
| git add index.html  git commit -m "Resolved merge conflict"  git status  git log --all --graph | Закоммитьте разрешенный конфлик Давайте посмотрим на текущее состояние нашего хранилища и убедимся, что все в порядке: |



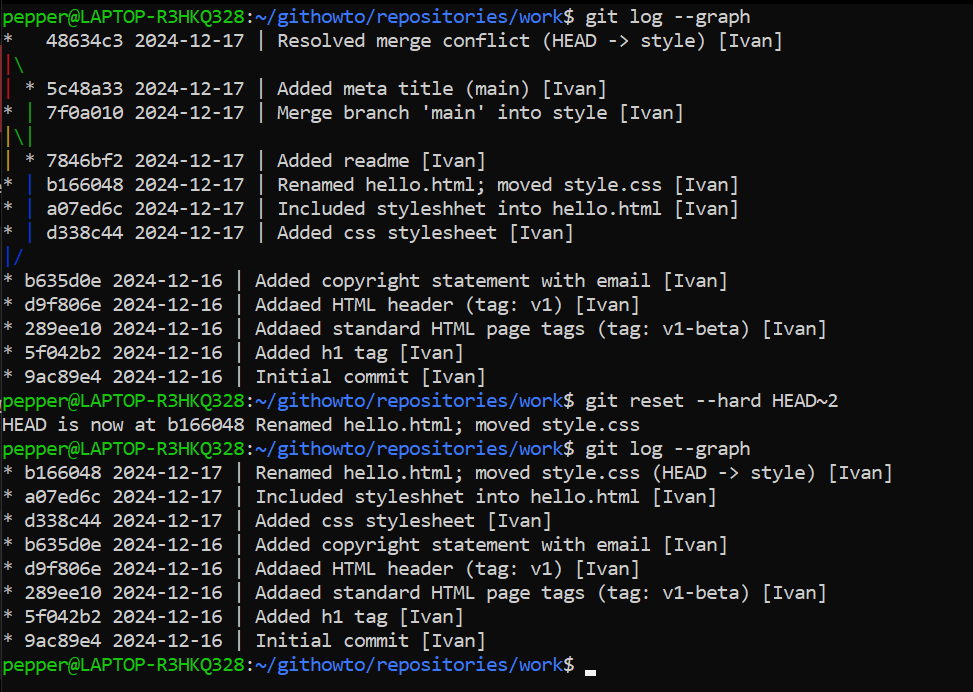
# 26. **rebase** против **merge**

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**

|  |  |
| --- | --- |
| git switch style  git log --graph | В этом случае мы хотим вернуться в ветке style в точку перед слиянием с main. Нам необходимо найти последний коммит перед слиянием. |
| git reset --hard HEAD~2 | 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. |
| git log --graph | Проверьте ветку |



# 28. Перебазирование

|  |  |
| --- | --- |
| git switch style  git rebase main  git status | Перебазируем ветку style на ветку mai |
| git add .  git rebase --continue | Сам конфликт может быть разрешен тем же способом, что и предыдущий. Сначала мы изменим файл hello.html, чтобы он соответствовал нашим ожиданиям. 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. |
| git status  git log --all --graph | 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. |

## Когда использовать команду rebase, а когда команду merge?

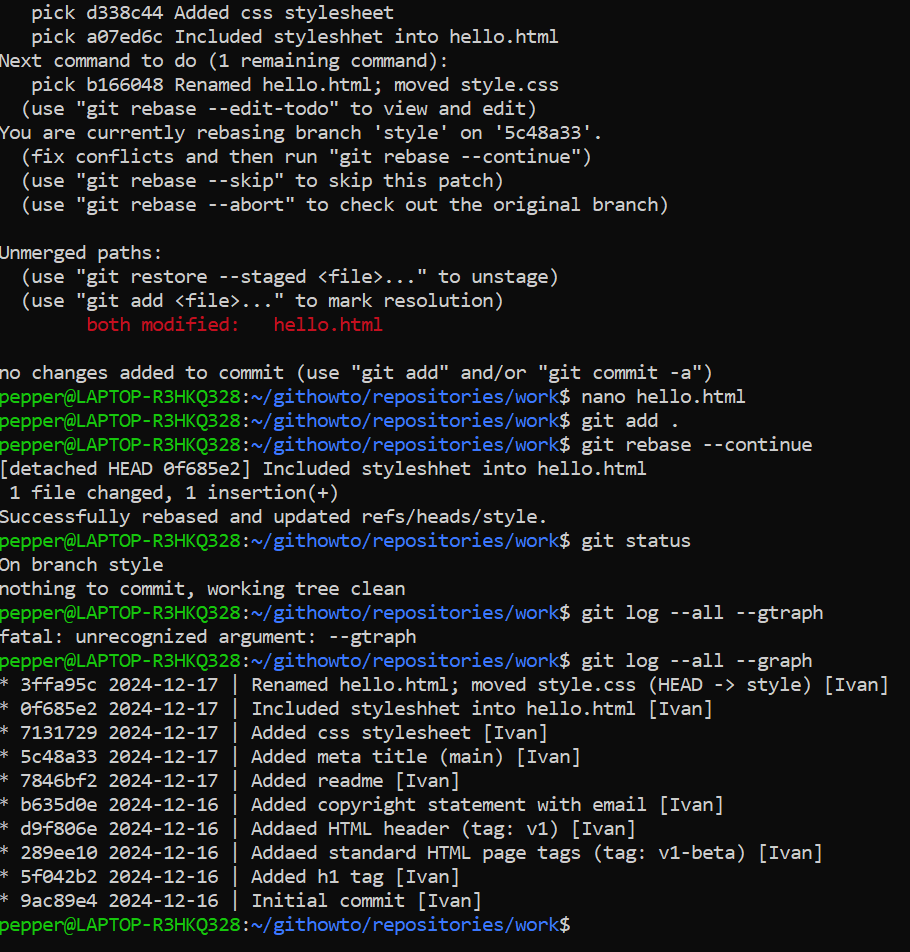
Используйте команду rebase:

* 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.

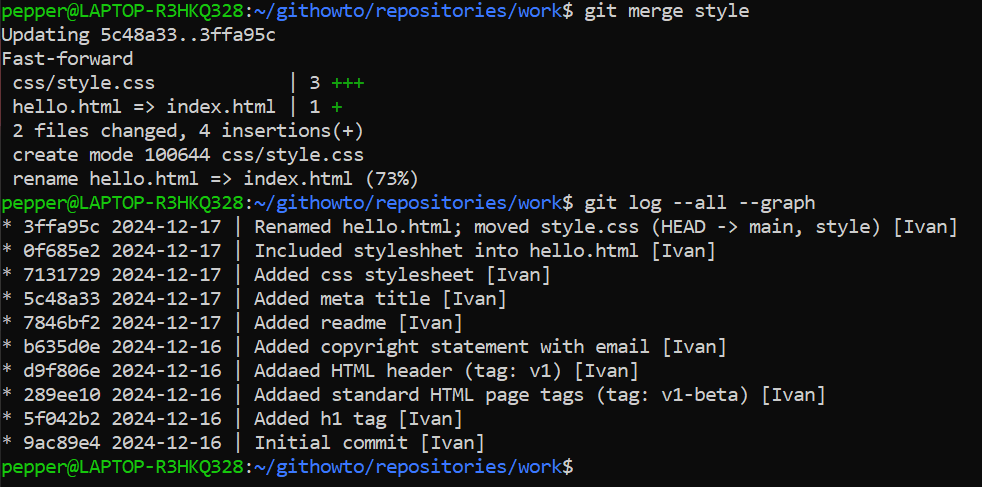


# 29. Слияние в ветку **main**

|  |  |
| --- | --- |
| git switch main  git merge style | Слейте style в main |
| git log --all --graph | Просмотрите логи |

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.

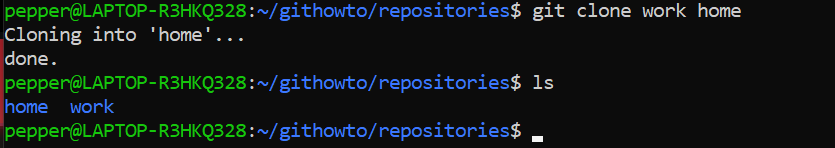
При ускоренном слиянии конфликты не возникают. Кроме того, при ускоренном слиянии не создается фиксация слияния.



# Часть II: Несколько репозиториев

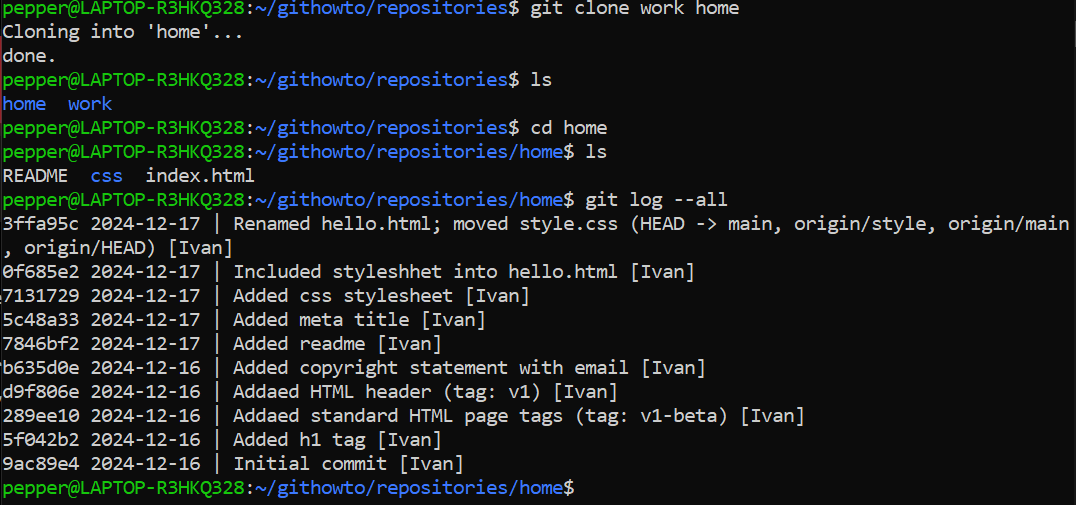
# 30. Клонирование репозиториев

|  |  |
| --- | --- |
| git clone work home  ls | Давайте создадим клон репозитория. |



# 31. Просмотр клонированного репозитория

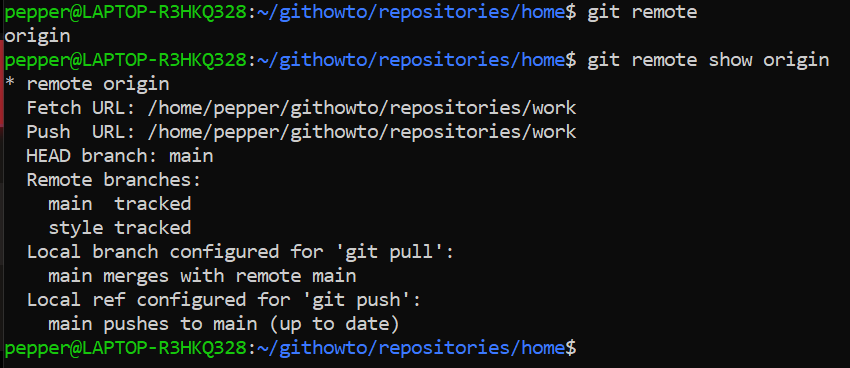
|  |  |
| --- | --- |
| git log --all | Просмотрите историю репозитория Вы увидите ветку main (HEAD) в списке истории. Вы также увидите ветки со странными именами (origin/main, origin/style и origin/HEAD). Мы поговорим о них чуть позже. |



# 32. Что такое origin?

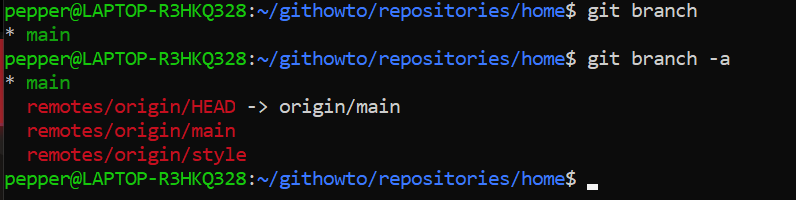
|  |  |
| --- | --- |
| git remote | репозиторий знает об имени по умолчанию удаленного репозитория |
| git remote show origin | получить более подробную информацию об имени по умолчанию: |

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).



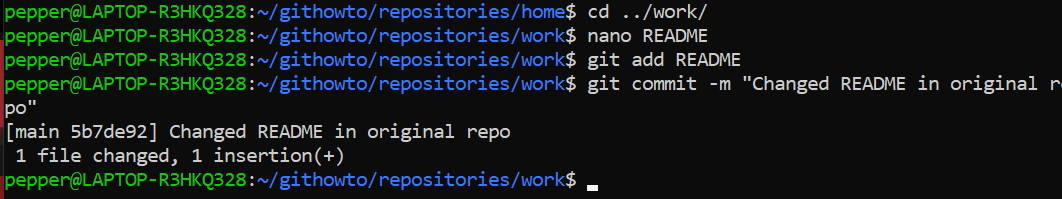
# 33. Удаленные ветки

|  |  |
| --- | --- |
| git branch | Давайте посмотрим на ветки, доступные в нашем клонированном репозитории |
| git branch -a | Список удаленных веток Для того чтобы увидеть все ветки, попробуйте следующую команду: |



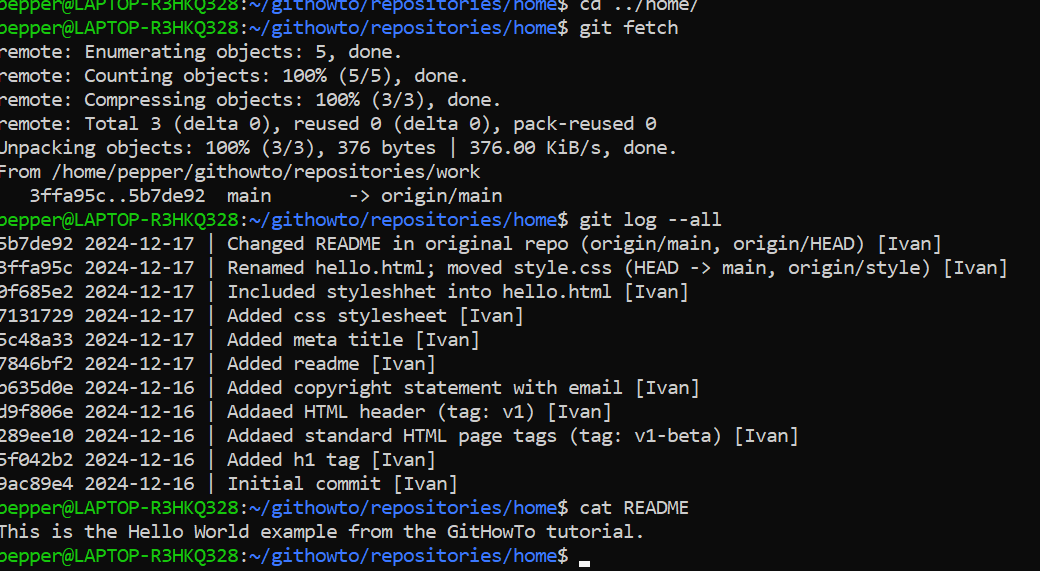
# 34. Изменение оригинального репозитория

|  |  |
| --- | --- |
| cd ../work | Сейчас мы находимся в репозитории work. |
| **Внесите следующие изменения в файл README:** | |
| git add README  git commit -m "Changed README in original repo" | Теперь добавьте это изменение и сделайте коммит |



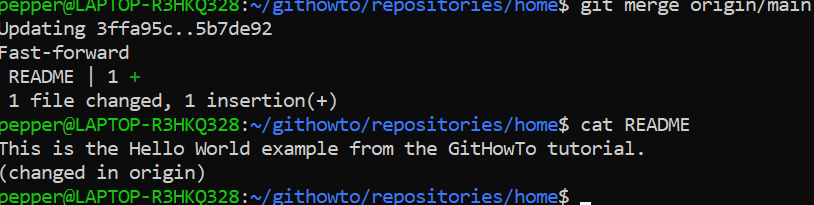
# 35. Подтягивание изменений

|  |  |
| --- | --- |
| cd ../home  git fetch  git log --all | 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. |
| cat README | Мы можем продемонстрировать, что клонированный файл README не изменил |



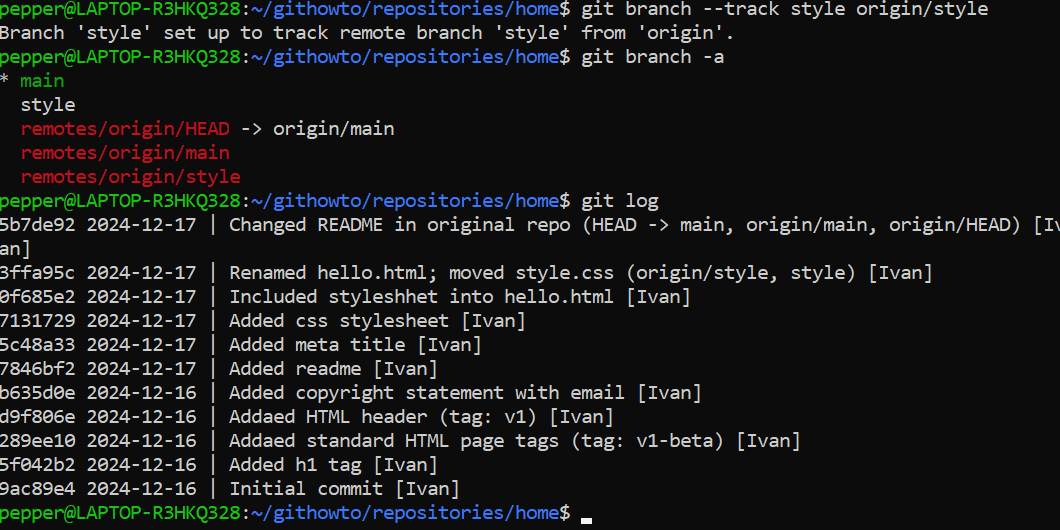
# 36. Слияние подтянутых изменений

|  |  |
| --- | --- |
| git merge origin/main | Слейте подтянутые изменения в локальную ветку main |
| git pull | 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. ...эквивалентна следующим двум шагам:  git fetch  git merge origin/main |



# 37. Добавление ветки наблюдения

|  |  |
| --- | --- |
| git branch --track style origin/style  git branch -a  git log --max-count=2 | 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. Добавьте локальную ветку, которая отслеживает удаленную ветку Теперь мы можем видеть ветку style в списке веток и логе. |

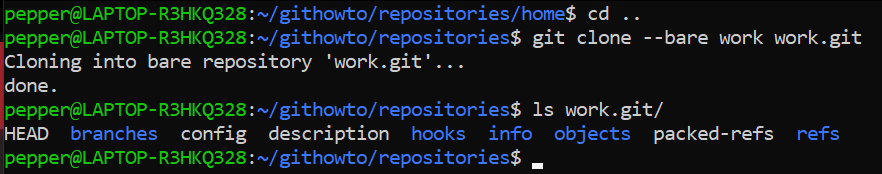


# 38. Чистые репозитории

|  |  |
| --- | --- |
| cd ..  git clone --bare work work.git  ls work.git | Создайте чистый репозиторий |

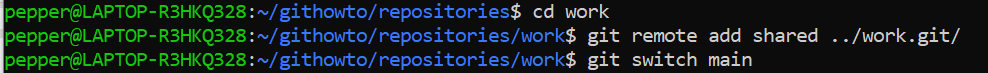
***Чистый репозиторий*** — это репозиторий, не имеющий рабочей директории. Он содержит только директорию .git, в которой Git хранит все свои внутренние данные. Основное предназначение таких репозиториев — быть центральным хранилищем, в которое разработчики могут отправлять и из которого могут получать данные.

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

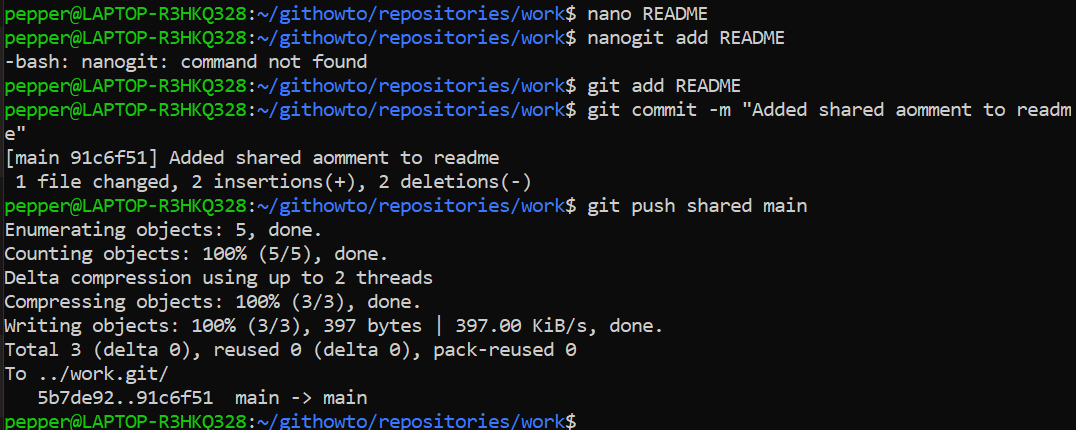
|  |  |
| --- | --- |
| cd work  git remote add shared ../work.git | Давайте добавим репозиторий work.git к нашему оригинальному репозиторию. |



# 40. Отправка изменений

|  |  |
| --- | --- |
| git switch main  git add README  git commit -m "Added shared comment to readme" | Начнем с создания изменения, которое нужно передать в репозиторий. Отредактируйте README и закоммитьте его: |
| git push shared main | Теперь отправьте изменения в общий репозиторий. |

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



# 41. Подтягивание общих изменений

|  |  |
| --- | --- |
| cd ../home | Быстро переключитесь в репозиторий home и подтяните изменения, |
| git remote add shared ../work.git  git branch --track shared main  git pull shared main  cat README | Подтяните изменения, только что отправленные в общий репозиторий. |

