
Front matter

title: "Отчёт по лабораторной работе №1"

subtitle: "Простейший вариант"

author: "Игнатенкова Варвара Николаевна"

Generic otions

lang: ru-RU toc-title: "Содержание"

Bibliography

bibliography: bib/cite.bib csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

Pdf output format

toc: true # Table of contents toc-depth: 2 lof: true # List of figures lot: true # List of tables fontsize: 12pt linestretch: 1.5
papersize: a4 documentclass: scrreprt

I18n polyglossia

polyglossia-lang: name: russian options: - spelling=modern - babelshorthands=true polyglossia-otherlangs: name: english

I18n babel

babel-lang: russian babel-otherlangs: english

Fonts

mainfont: IBM Plex Serif romanfont: IBM Plex Serif sansfont: IBM Plex Sans monofont: IBM Plex Mono mathfont: STIX
Two Math mainfontoptions: Ligatures=Common,Ligatures=TeX,Scale=0.94 romanfontoptions:
Ligatures=Common,Ligatures=TeX,Scale=0.94 sansfontoptions:
Ligatures=Common,Ligatures=TeX,Scale=MatchLowercase,Scale=0.94 monofontoptions:
Scale=MatchLowercase,Scale=0.94,FakeStretch=0.9 mathfontoptions:

Biblatex

biblatex: true biblio-style: "gost-numeric" biblatexoptions:

- parenttracker=true
- backend=biber
- hyperref=auto
- language=auto
- autolang=other*
- citestyle=gost-numeric

Pandoc-crossref LaTeX customization

figureTitle: "Рис." tableTitle: "Таблица" listingTitle: "Листинг" lofTitle: "Список иллюстраций" lotTitle: "Список таблиц"
lolTitle: "Листинги"

Misc options

indent: true header-includes:

- \usepackage
- \usepackage # keep figures where there are in the text
- \floatplacement # keep figures where there are in the text

Цель работы

Знакомство с git и изучение его работы.

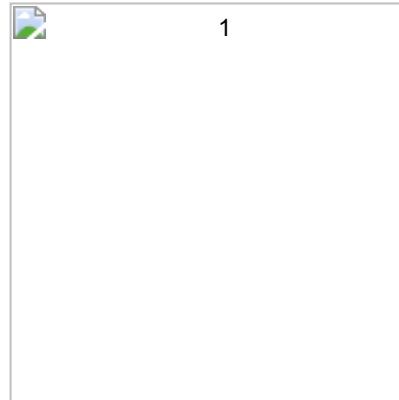
Теоретическое введение

Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года; координатор — Дзюн Хамано.

Выполнение лабораторной работы

Подготовка

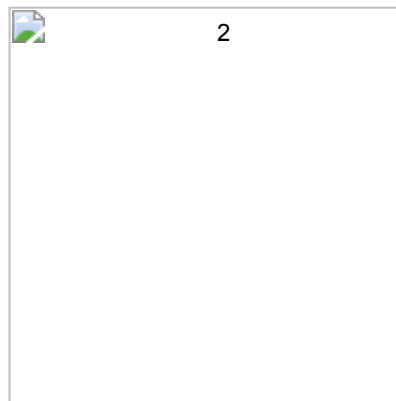
Установка имени и электронной почты



Установим имя и электронную почту для установки git.

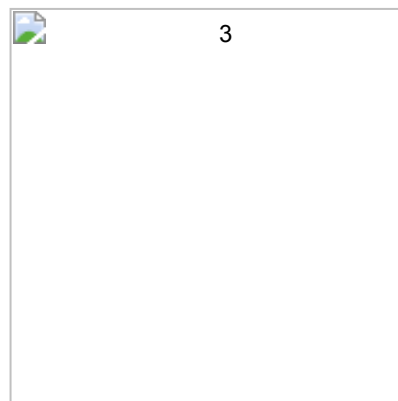
Параметры установки окончаний строк

Настроим `core.autocrlf` с параметрами `true` и `input` для перевода строк текстовых файлов в главном репозитории в



одинаковый формат.

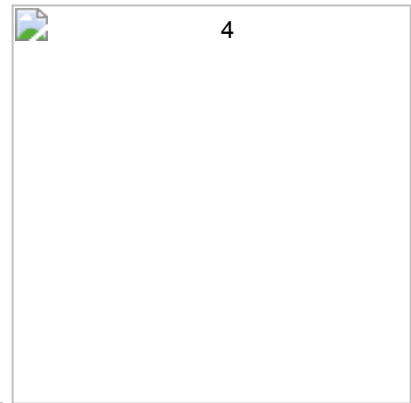
Установка отображения unicode



Установим отображение unicode.

Создание проекта

Создание страницы «Hello,World»



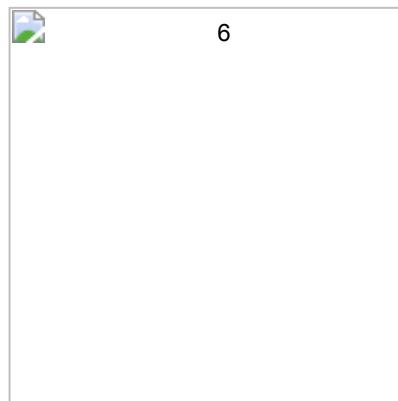
Создадим пустой каталог с именем hello и создадим файл hello.html.

Создание репозитория



Создадим git репозиторий с помощью команды `git init`.

Добавление файла в репозиторий



Добавим файл в репозиторий.

Проверка состояние репозитория



Проверим состояние репозитория.

Команда проверки состояния сообщит, что

коммитить нечего.

Внесение изменений

Изменим страницу «Hello,World»



Добавим HTML-теги в файле hello.html.

Проверим состояние рабочего



каталога.

Git знает, что файл hello.html был изменен,но при этом эти

изменения еще не зафиксированы в репозитории.

Индексация изменений

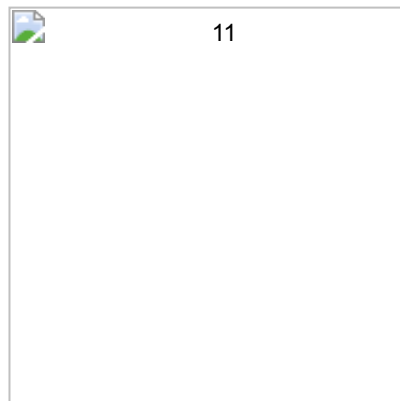
Коммит изменений



Проиндексируем изменения командой `git` и проверим состояние.

Изменения

файла `hello.html` были проиндексированы. Это означает, что `git` теперь знает об изменении, но изменение пока не записано в репозиторий. Следующий коммит будет включать в себя проиндексированные изменения.



Сделаем коммит без метки `-m`.

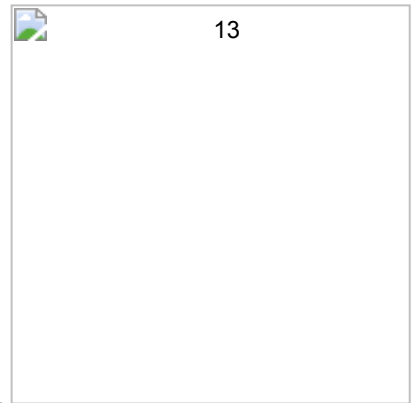
После открытия первой строки введем

комментарий, сохраним файл и выйдем из редактора. Еще раз проверим состояние.

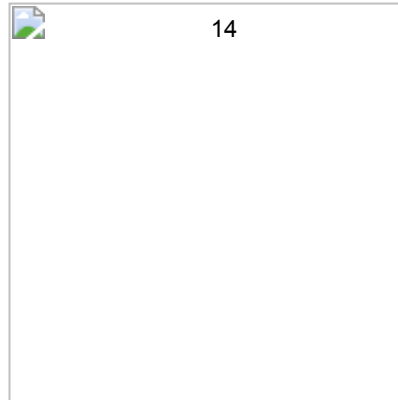


Рабочий каталог чистый, продолжим работу.

Добавим стандартные теги страницы



Измените страницу «Hello,World», чтобы она содержала стандартные теги и .



Теперь добавим это изменение в индекс git.



Теперь добавим заголовки HTML

(секцию) к странице «Hello,World».

Проверим текущий статус.



Hello.html указан дважды в состоянии. Первое изменение (добавление стандартных тегов) проиндексировано и готово к коммиту. Второе изменение(добавление заголовков HTML) является непроиндексированным.

Произведем коммит проиндексированного изменения, а затем еще раз проверим состояние.



17

Состояние команды говорит о том, что `hello.html` имеет незафиксированные изменения, но уже не в буферной зоне.

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



18

Сделаем коммит второго изменения.



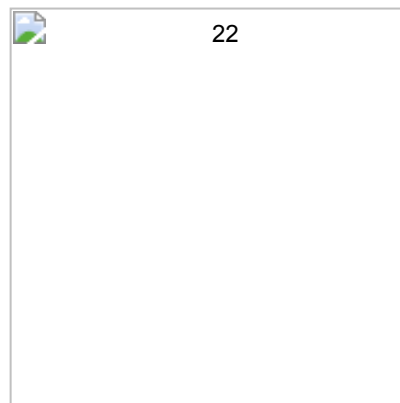
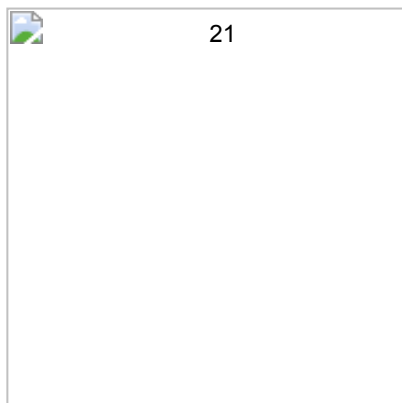
19

История

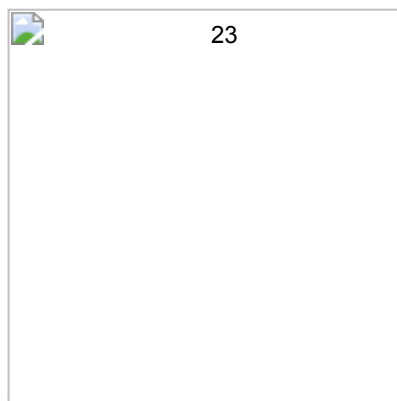


Получим список произведенных изменений.

Однострочный формат истории:



Есть много вариантов отображения лога.



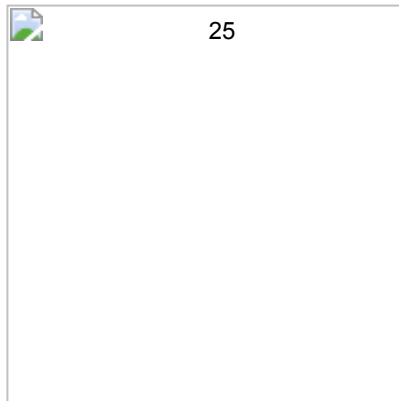
Справочная страница:

Получение старых версий



Получим хэши предыдущих версий.

Изучим данные лога и найдем хэш для первого коммита. Используем этот хэш-код в команде ниже. Затем проверим содержимое файла hello.html.



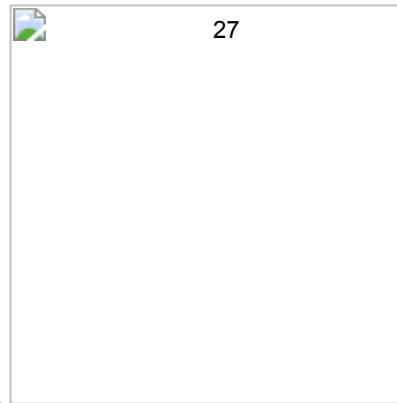
Вернемся к последней версии в ветке master.

Создание тегов версий

Давайте назовем текущую версию страницы hello первой (v1). Создадим тег первой версии.

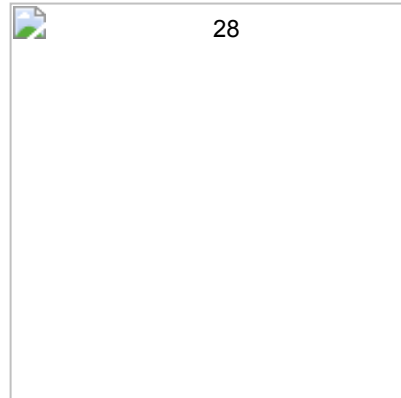


Давайте создадим тег для версии, которая идет перед текущей версией и назовем его v1-beta. В первую очередь нам надо переключиться на предыдущую версию. Вместо поиска до



хэш,мы будем использовать ^, обозначающее «родитель v1».

Это версия с



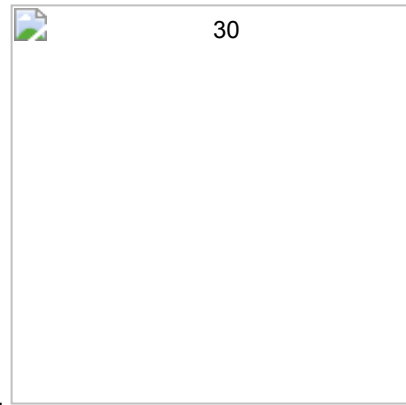
тегами и ,но еще пока без .Давайте сделаем ее версией v1-beta.

Переключение по имени тега



Теперь попробуем попереключаться между двумя отмеченными версиями.

Просмотр тегов с помощью команды tag



Мы можем увидеть, какие теги доступны, используя команду `git tag`.

Мы так



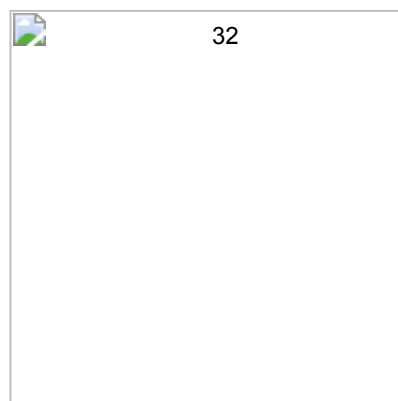
же можем посмотреть теги в логе.

Мы можем видеть теги (`v1` и `v1-beta`) в логе вместе с именем ветки (`master`). Кроме того HEAD показывает коммит, на который переключились (на данный момент это `v1-beta`).

Отмена локальных изменений (до индексации)

Переключимся на ветку `master`

Убедимся, что мы находимся на последнем коммите ветки `master`, прежде чем продолжить работу.



Изменим `hello.html`



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

Проверим состояние



Сначала проверим состояние рабочего каталога.

Мы видим, что файл

hello.html был изменен, но еще не проиндексирован.

Отмена изменений в рабочем каталоге

Используем команду `git checkout` для переключения версии файла hello.html в репозитории.



Команда `git status` показывает нам, что не было произведено никаких изменений, не зафиксированных в рабочем каталоге.

Отмена проиндексированных изменений (перед коммитом)

Изменим файл и проиндексируем изменения



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

Проиндексируем это изменение.

Проверим состояние



Проверим состояние нежелательного изменения.

Состояние показывает, что

изменение было проиндексировано и готово к коммиту.

Выполним сброс буферной зоны



Выполним отмену индексации изменения.

Команда `git reset` сбрасывает

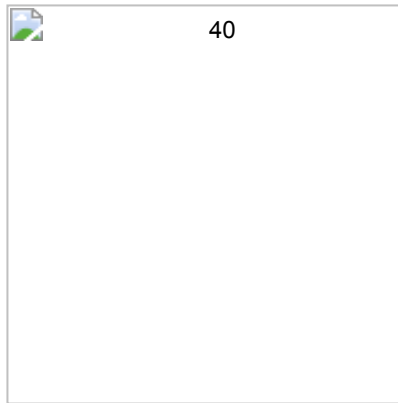
буферную зону к HEAD. Это очищает буферную зону от изменений, которые мы только что проиндексировали.

Команда `git reset` (по умолчанию) не изменяет рабочий каталог. Поэтому рабочий каталог все еще

содержит нежелательный комментарий. Мы можем использовать команду `git checkout`, чтобы

удалить нежелательные изменения в рабочем каталоге.

Переключимся на версию коммита



Наш рабочий каталог опять чист.

Отмена коммитов

Отмена коммитов

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

Изменим файл и сделаем коммит



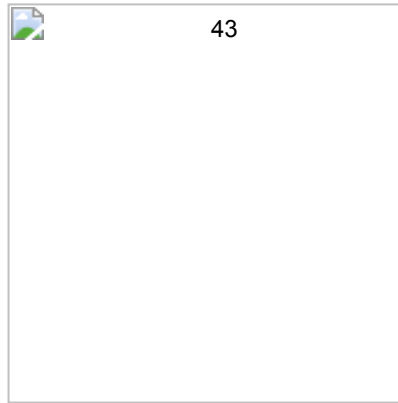
Изменим файл hello.html на следующий.

Выполним следующие команды:



Сделаем коммит с новыми изменениями,отменяющими
предыдущие

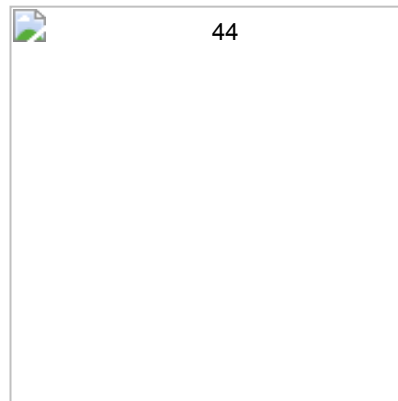
Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные



нежелательным коммитом.

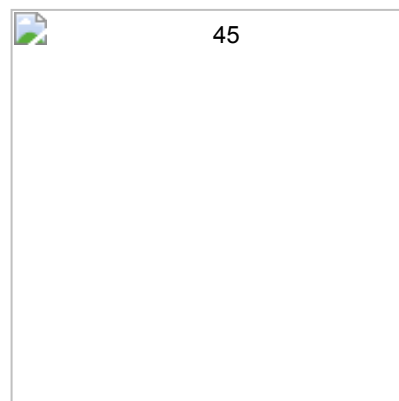
Проверим лог

Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий.



Удаление коммитов из ветки

Проверим нашу историю



Давайте сделаем быструю проверку нашей истории коммитов.

Мы видим, что

два последних коммита в этой ветке — «Oops» и «Revert Oops». Давайте удалим их с помощью сброса.

Для начала отметим эту ветку

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



Сброс коммитов к предшествующим коммитуOops

Глядя на историю лога, мы видим, что коммит с тегом «v1» является коммитом, предшествующим ошибочному



коммиту. Давайте сбросим ветку до этой точки с помощью тега.

Ничего никогда не теряется



Давайте посмотрим на все коммиты.

Мы видим, что ошибочные коммиты не исчезли. Они все еще находятся в репозитории. Просто они отсутствуют в ветке master.

Опасность сброса

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

Удаление тега oops

Удаление тега oops

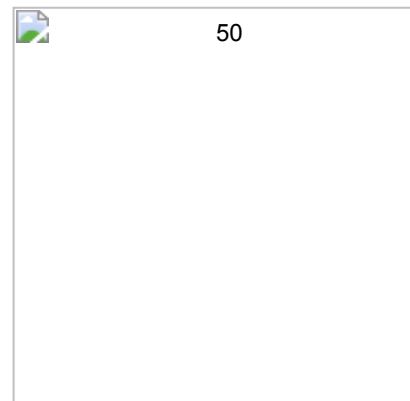
Тег oops свою функцию выполнил. Давайте удалим его и коммиты, на которые он ссылался, сборщиком мусора.



Тег «oops» больше не будет отображаться в репозитории.

Внесение изменений в коммиты

Изменим страницу, а затем сделаем коммит



Добавим в страницу комментарий автора (вставим свою фамилию).



Выполним:

Необходим email



52

Обновим страницу hello, включив в нее email.

Изменим предыдущий коммит



53

Выполним:

Просмотр истории



54

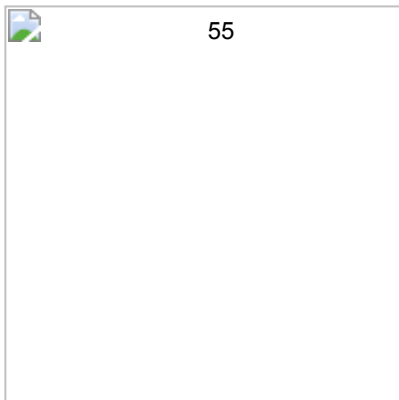
Выполним:

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

Перемещение файлов

Переместим файл hello.html в каталог lib

Сейчас мы собираемся создать структуру нашего репозитория. Давайте перенесем страницу в каталог lib.



Перемещая файлы с помощью `git mv`, мы информируем `git` о 2 вещах: • Что файл `hello.html` был удален. • Что файл `lib/hello.html` был создан. Оба этих факта сразу же проиндексированы и готовы к коммиту. Команда `git status` сообщает, что файл был перемещен.

Второй способ перемещения файлов

Следующий набор команд идентичен нашим последним действиям. Работы здесь побольше, но результат тот же.

Мы могли бы выполнить: `mkdir lib mv hello.html lib git add lib/hello.html git rm hello.html`

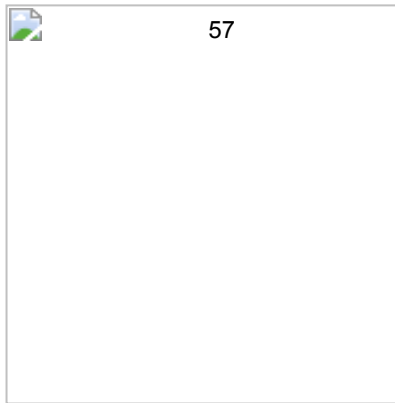
Коммит в новый каталог



Давайте сделаем коммит этого перемещения:

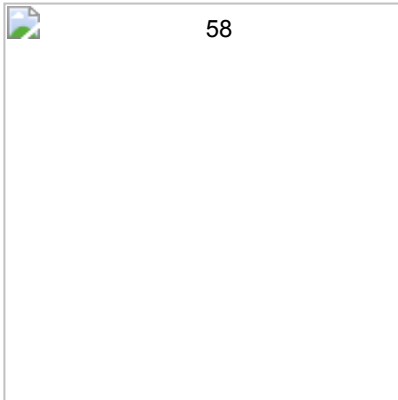
Подробнее о структуре

Добавление `index.html`



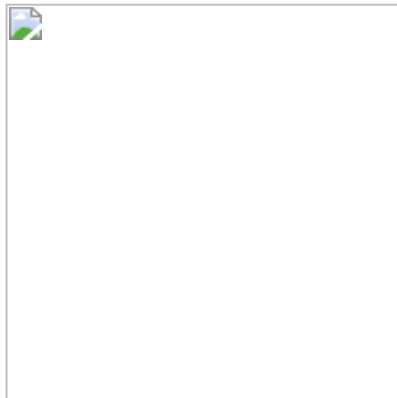
Добавим файл index.html в наш репозиторий.

Добавим файл и сделаем



коммит.

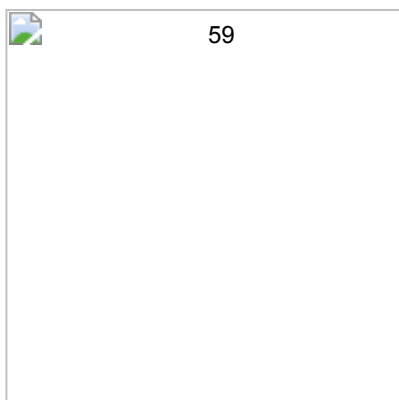
Теперь при открытии index.html, мы должны увидеть кусок страницы



hello в маленьком окошке.

Git внутри: Каталог.git

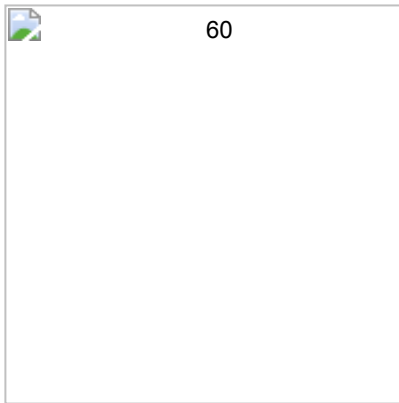
Каталог .git



Выполним:

Это каталог, в котором хранится вся информация git.

База данных объектов

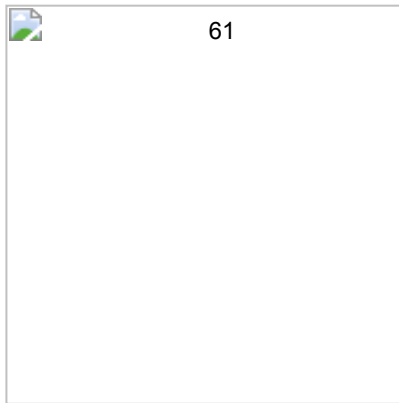


Выполним:

Мы видим набор каталогов, имена которых состоят из 2 символов.

Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося в git.

Углубляемся в базу данных объектов

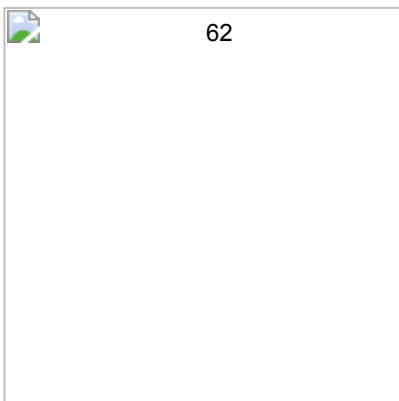


Выполним:

Смотрим в один из каталогов с именем из 2 букв. Мы видим

файлы с именами из 38 символов. Это файлы, содержащие объекты, хранящиеся в git. Они сжаты и закодированы, поэтому просмотр их содержимого нам мало чем поможет.

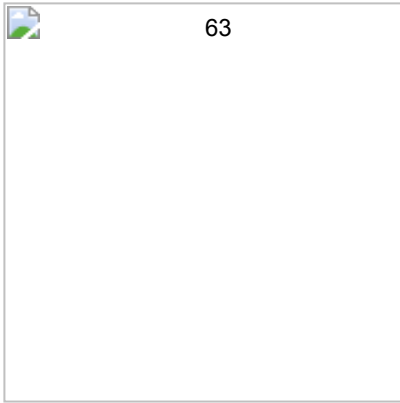
Config File



Выполним:

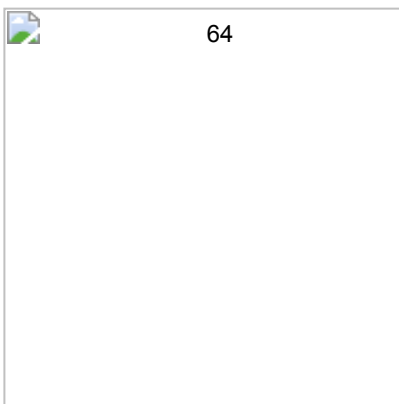
Это файл конфигурации, создающийся для каждого конкретного проекта. Записи в этом файле будут перезаписывать записи в файле `.gitconfig` нашего главного каталога, по крайней мере в рамках этого проекта.

Ветки и теги



Выполним: Каждый файл соответствует тегу, ранее созданному с помощью команды `git tag`. Его содержание — это всего лишь хэш коммита, привязанный к тегу.

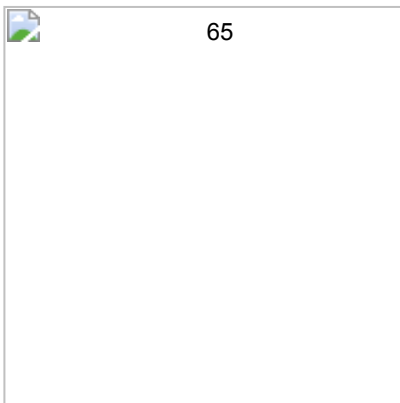
Файл HEAD



Выполним: Файл HEAD содержит ссылку на текущую ветку, в данный момент это должна быть ветка `master`.

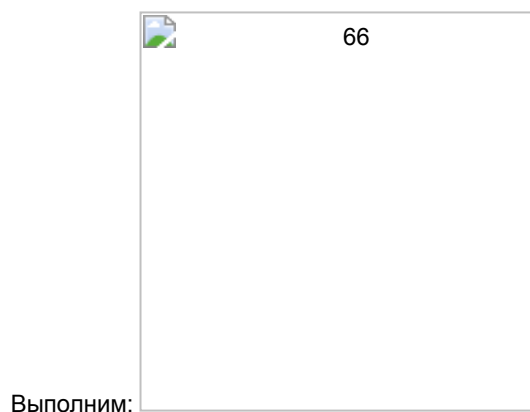
Работа непосредственно с объектами git

Поиск последнего коммита



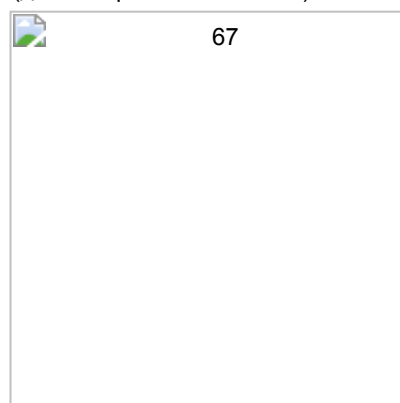
Выполним: Эта команда должна показать последний коммит в репозиторий. SHA1 хэш в нашей системе, вероятно, отличается от моего, но мы увидим что-то наподобие этого.

Вывод последнего коммита с помощью SHA1 хэша

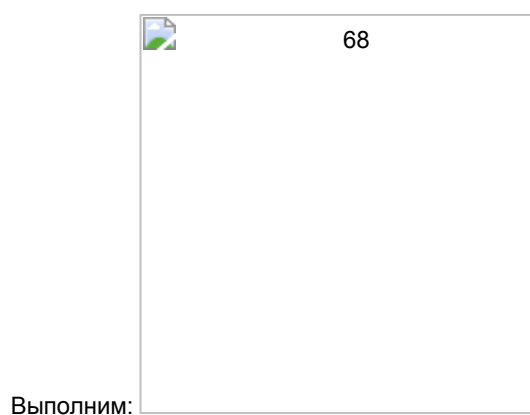


Поиск дерева

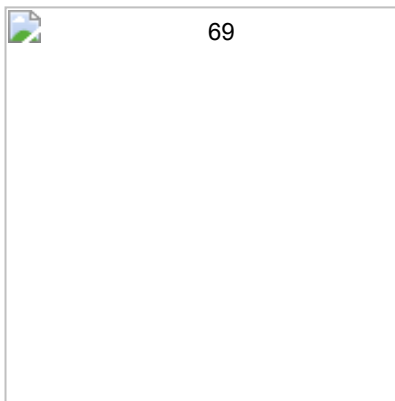
Мы можем вывести дерево каталогов, ссылка на который идет в коммите. Это должно быть описание файлов (верхнего уровня) в нашем проекте (для конкретного коммита). Используем SHA1 хэш из строки «дерева».



Вывод каталога lib



Вывод файла hello.html



Выполним:

Исследуем самостоятельно

Нахождение оригинального файла `hello.html` с самого первого коммита вручную по ссылкам SHA1 хэша в

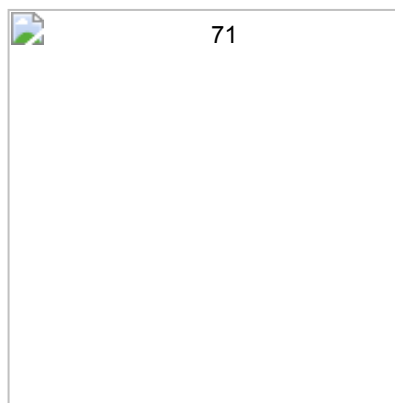


последнем коммите.

Создание ветки

Пора сделать наш `hello world` более выразительным. Так как это может занять некоторое время, лучше переместить эти изменения в отдельную ветку, чтобы изолировать их от изменений в ветке `master`.


Создадим ветку




Давайте назовем нашу новую ветку «`style`».

Добавим файл стилей `style.css`

Выполним:


72


Файл стилей:

73

Изменим основную страницу

Обновим файл hello.html, чтобы использовать стили style.css.

74

75

Выполним:

Изменим index.html



76

Обновим файл index.html, чтобы он тоже использовал style.css.

Выполним:



77

Навигация по веткам



78

Теперь в вашем проекте есть две ветки:

Переключение на ветку master



Используем команду `git checkout` для переключения между ветками:

Сейчас

мы находимся на ветке `master`. Это заметно по тому, что файл `hello.html` не использует стили `style.css`.

Вернемся к ветке `style`



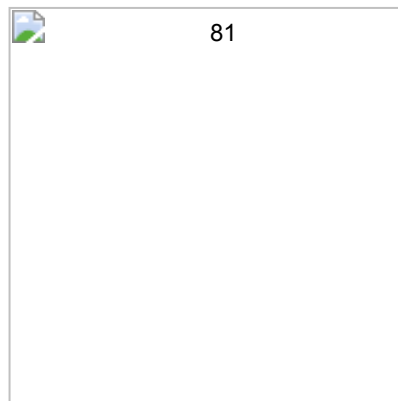
Выполним:

Содержимое `lib/hello.html` подтверждает, что мы вернулись на ветку `style`.

Изменения в ветке `master`

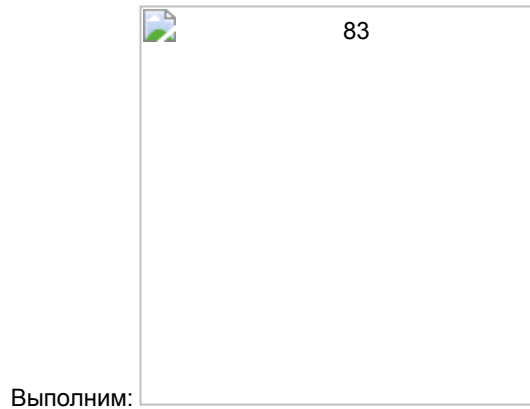
Пока вы меняли ветку `style`, кто-то решил обновить ветку `master`. Они добавили файл `README.md`.

Создайте файл `README` в ветке `master`



Создадим файл `README.md`.

Сделаем коммит изменений README.md в ветку master.



Просмотр отличающихся веток

Просмотр текущих веток

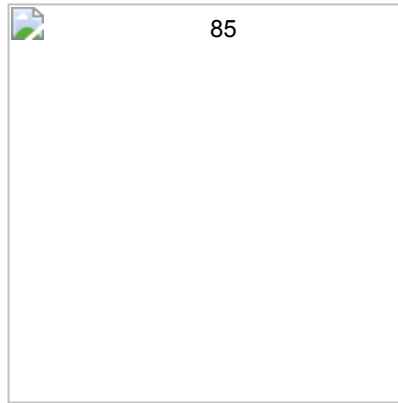
Теперь у нас в репозитории есть две отличающиеся ветки. Используем следующую лог-команду для просмотра



Слияние

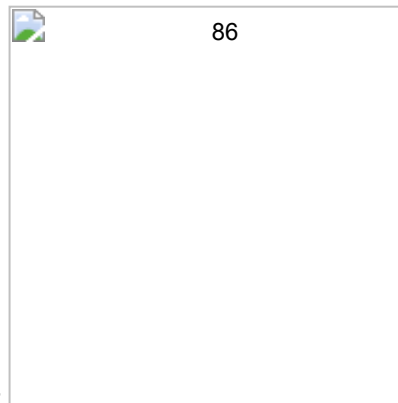
Слияние веток

Слияние переносит изменения из двух веток в одну. Давайте вернемся к ветке style и сделаем masterstyle.



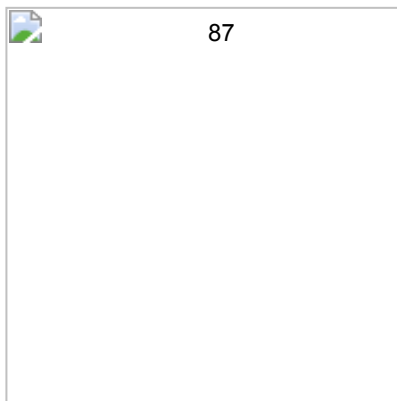
Создание конфликта

Вернемся в master и создадим конфликт

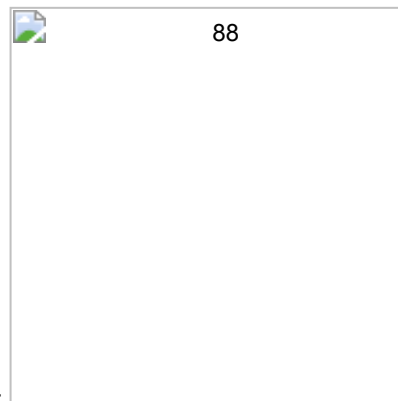


Вернемся в ветку master и внесем следующие изменения:

Файл lib/hello.html



Выполним:



Просмотр веток

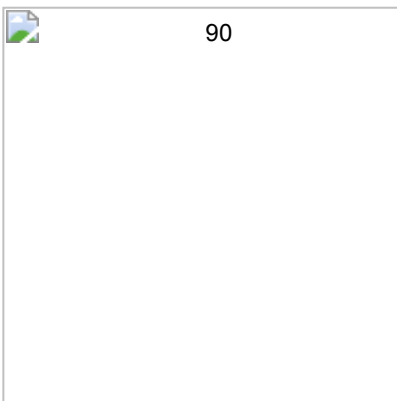


Выполним: После коммита «Added README» ветка master была объединена с веткой style, но в настоящее время в master есть дополнительный коммит, который не был слит с style. Последнее изменение в master конфликтует с некоторыми изменениями в style. На следующем шаге мы решим этот конфликт.

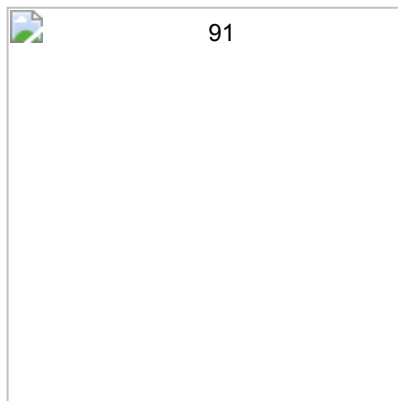
Разрешение конфликтов

Слияние master с веткой style

Теперь вернемся к ветке style и попытаемся объединить ее с новой веткой master.



Откроем lib/hello.html.



Первый раздел—

версия текущей ветки (style). Второй раздел—версия ветки master.

Решение конфликта



Внесем изменения в lib/hello.html для достижения следующего результата.

Сделаем коммит решения конфликта



93

Выполним:

Перебазирование как альтернатива слиянию

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

Сброс ветки style

Сброс ветки style

Вернемся на ветке `style` к точке перед тем, как мы слили ее с веткой `master`. Выполним:



94

Мы видим, что коммит «Updated index.html» был последним на ветке `style`



95

перед слиянием. Давайте сбросим ветку `style` к этому коммиту. Выполним:

Проверьте ветку.



96

Выполним:

Сброс ветки master

Сброс ветки master

Давайте вернемся в ветке master в точку перед внесением конфликтующих изменений.



97

Коммит «Added README» идет непосредственно перед коммитом конфликтующего интерактивного режима. Мы сбросим ветку master к коммиту «AddedREADME».



98

Лог должен выглядеть, как будто репозиторий был перемотан назад во времени к точке до какого-либо слияния.

Перебазирование



99

Используем команду rebase вместо команды merge.

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 master. Однако, дерево коммитов значительно отличается. Дерево коммитов ветки style было переписано таким образом, что ветка master является частью истории коммитов. Это делает цепь коммитов линейной и гораздо более читабельной.

Не используйте перебазирование: • если ветка является публичной и расшаренной, поскольку переписывание общих веток будет мешать работе других членов команды; • когда важна точная история коммитов ветки, так как команда rebase переписывает историю коммитов;

Учитывая приведенные выше рекомендации, рекомендуется использовать git rebase для кратковременных, локальных веток, а слияние для веток в публичном репозитории.

Слияние в ветку master

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

Слияние style в master



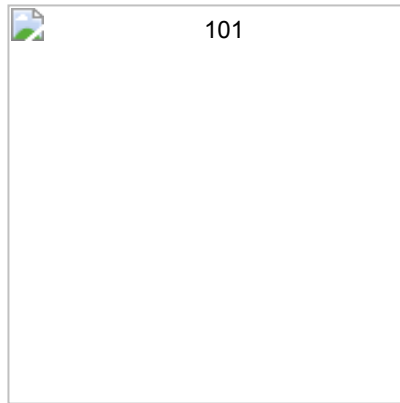
100

Выполним:

Поскольку последний коммит ветки master

прямо предшествует последнему коммиту ветки style, git может выполнить ускоренное слияние-перемотку. При быстрой перемотке вперед git просто передвигает указатель вперед, таким образом указывая на тот же коммит, что и ветка style. При быстрой перемотке конфликтов быть не может.

Просмотрим логи



Выполним:

Теперь ветки style и master идентичны.

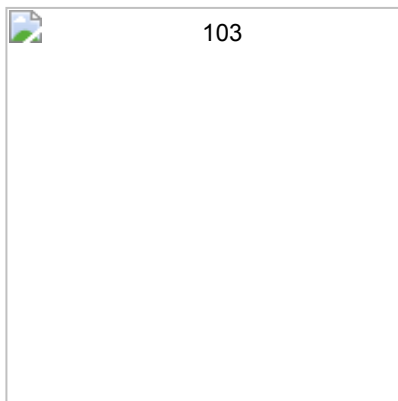
Клонирование репозитория

Перейдем в рабочий каталог



Выполним:

Создадим клон репозитория hello



Выполним:

В рабочем каталоге теперь должно быть два репозитория:

оригинальный репозиторий «hello» и клонированный репозиторий «cloned_hello».

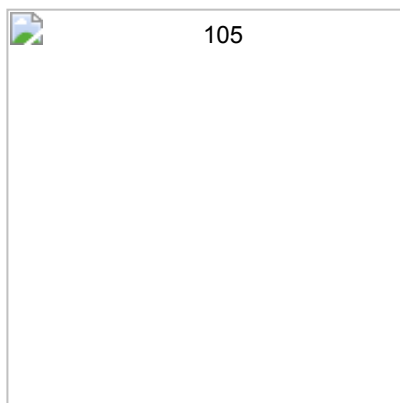
Просмотр клонированного репозитория

Давайте взглянем на клонированный репозиторий.



Выполним: Мы увидим список всех файлов на верхнем уровне оригинального репозитория README.md, index.html и lib.

Просмотрите историю репозитория



Выполним: Мы увидим список всех коммитов в новый репозиторий, и он должен (более или менее) совпадать с историей коммитов в оригинальном репозитории. Единственная разница должна быть в названиях веток.

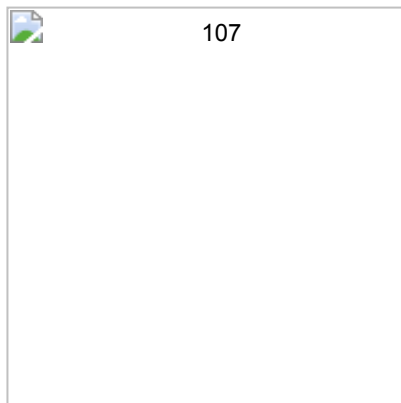
Удаленные ветки

Мы увидим ветку master (HEAD) в списке истории. Мы также увидим ветки со странными именами (origin/master, origin/style и origin/HEAD).

Что такое origin?

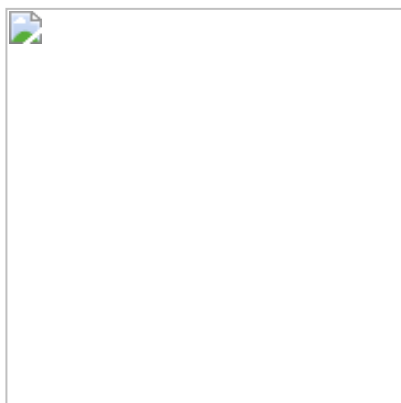


Выполним: Мы видим, что клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Давайте посмотрим, можем ли мы получить более подробную информацию



об имени по умолчанию: 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).

Удаленные ветки



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

Список удаленных веток

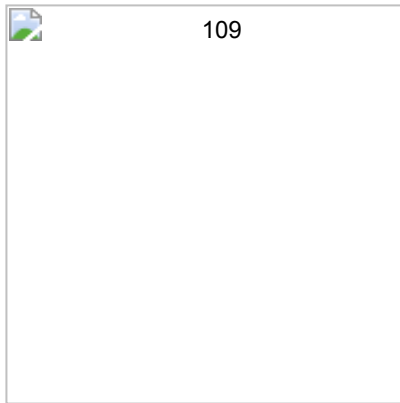


Для того, чтобы просмотреть все ветки, выполним: `git branch -a` Git выводит все коммиты в оригинальный репозиторий, но ветки в удаленном репозитории не рассматриваются как локальные. Если мы хотим собственную ветку `style`, мы должны сами ее создать.

Изменение оригинального репозитория

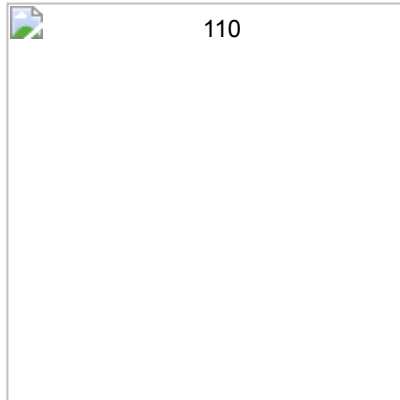
Внесем некоторые изменения в оригинальный репозиторий, чтобы затем попытаться извлечь и слить изменения из удаленной ветки в текущую.

Внесение изменения в оригинальный репозиторий hello



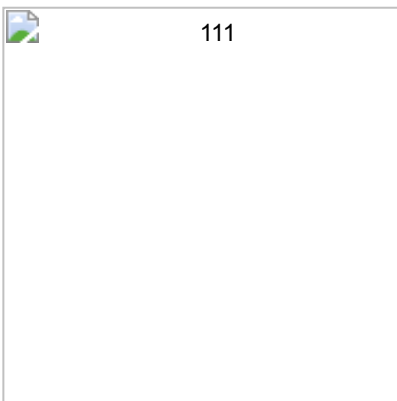
Выполним:

Внесем следующие изменения в файл README.md:



Файл README.md

Теперь добавим это изменение и сделаем коммит.



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

Извлечение изменений



Выполним:

Команда `git fetch` будет извлекать новые коммиты из удаленного репозитория, но не будет сливать их с вашими наработками в локальных ветках.

Проверим README.md



Выполним:

Слияние извлеченных изменений

Сольем извлеченные изменения в локальную ветку



Выполним:

Еще раз проверим файл README.md



Сейчас мы должны увидеть изменения. Выполним:

Хотя команда `git fetch` не

сливает изменения, мы можем вручную слить изменения из удаленного репозитория.

Добавление ветки наблюдения

Ветки, которые начинаются с `remotes/origin` являются ветками оригинального репозитория.

Добавим локальную ветку, которая отслеживает удаленную ветку



Выполним:

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

Чистые репозитории

Чистые репозитории (без рабочих каталогов) обычно используются для расширения. Обычный `git`-репозиторий подразумевает, что вы будете использовать его как рабочую директорию, поэтому вместе с файлами проекта в актуальной версии, `git` хранит все служебные, «чисто-репозиторийские» файлы в поддиректории `.git`. В удаленных репозиториях нет смысла хранить рабочие файлы на диске (как это делается в рабочих копиях), а все что им действительно нужно — это дельты изменений и другие бинарные данные репозитория. Вот это и есть «чистый репозиторий».

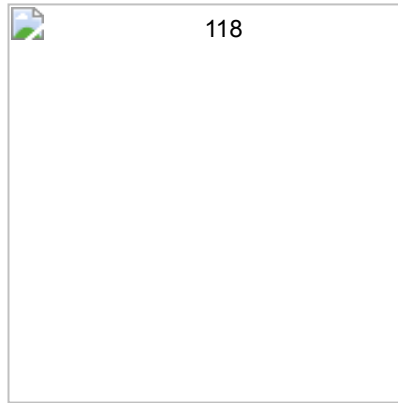
Создадим чистый репозиторий



Сейчас мы находимся в рабочем каталоге. Как правило, репозитории, оканчивающиеся на `.git` являются чистыми репозиториями. Мы видим, что в репозитории `hello.git` нет рабочего каталога. По сути, это есть не что иное, как каталог `git` нечистого репозитория.

Добавление удаленного репозитория

Давайте добавим репозиторий hello.git к нашему оригинальному репозиторию.



Отправка изменений



Отредактируем файл README.md и сделаем коммит.

Выполним:



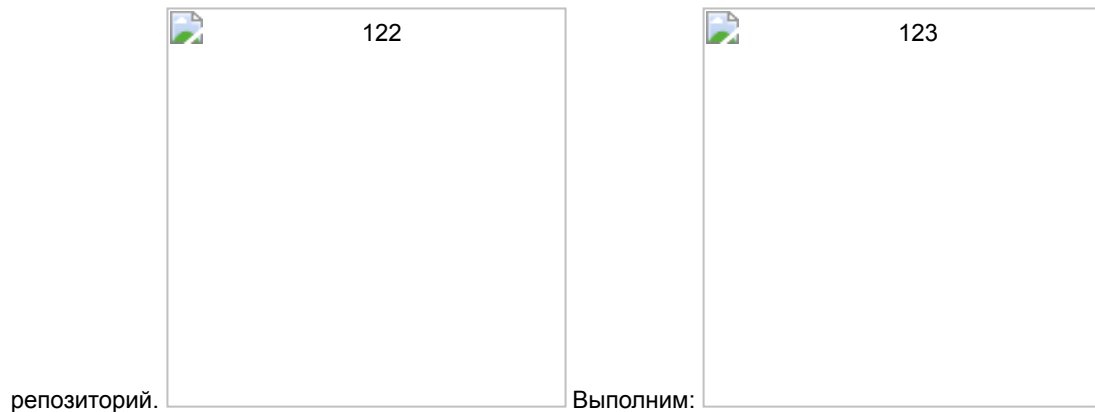
Теперь отправим изменения в общий репозиторий.



Общим называется репозиторий, получающий отправленные нами изменения.

Извлечение общих изменений

Быстро переключимся в клонированный репозиторий и извлечем изменения, только что отправленные в общий



Выводы

Мы ознакомились с работой с git и приобрели соответствующие навыки.

Список литературы