

Лабораторная работа №1

Работа с git

Кудряшов Артём Николаевич

Содержание

| | | |
|----------|--|----------|
| 1 | Цель работы | 5 |
| 2 | Теоретическое введение | 6 |
| 3 | Выполнение лабораторной работы | 7 |
| 3.1 | Подготовка | 7 |
| 3.2 | Создание проекта | 7 |
| 3.3 | Внесение изменений | 8 |
| 3.4 | История | 9 |
| 3.5 | Получение старых версий | 9 |
| 3.6 | Создание тегов версий | 10 |
| 3.7 | Отмена локальных изменений (до индексации) | 11 |
| 3.8 | Отмена проиндексированных изменений (перед коммитом) | 11 |
| 3.9 | Отмена коммитов | 12 |
| 3.10 | Удаление коммитов из ветки | 13 |
| 3.11 | Удаление тега oops | 13 |
| 3.12 | Изменение предыдущего коммита | 13 |
| 3.13 | Перемещение файлов | 14 |
| 3.14 | Подробнее о структуре | 14 |
| 3.15 | Git внутри: Каталог .git | 15 |
| 3.16 | Работа непосредственно с объектами git | 16 |
| 3.17 | Создание ветки | 17 |
| 3.18 | Навигация по веткам | 17 |
| 3.19 | Изменения в ветке master | 17 |
| 3.20 | Слияние | 18 |
| 3.21 | Создание конфликта | 18 |
| 3.22 | Разрешение конфликтов | 18 |
| 3.23 | Сброс ветки style | 18 |
| 3.24 | Сброс ветки master | 19 |
| 3.25 | Перебазирование | 19 |
| 3.26 | Слияние в ветку master | 19 |
| 3.27 | Клонирование репозитория | 20 |
| 3.28 | Что такое origin? | 20 |
| 3.29 | Удаленные ветки | 20 |
| 3.30 | Изменение оригинального репозитория | 21 |
| 3.31 | Слияние извлеченных изменений | 21 |
| 3.32 | Добавление ветки наблюдения | 21 |

| | | |
|----------|---|-----------|
| 3.33 | Создание чистого репозитория | 21 |
| 3.34 | Отправка и извлечение изменений | 22 |
| 4 | Выводы | 23 |
| | Список литературы | 24 |

Список иллюстраций

| | | |
|------|---|----|
| 3.1 | Создание репозитория | 7 |
| 3.2 | Внесение нескольких изменений в файл | 9 |
| 3.3 | Просмотр разных версий репозитория | 10 |
| 3.4 | Переключение по имени тега и просмотр доступных тегов | 10 |
| 3.5 | Добавления нежелательного комментария | 11 |
| 3.6 | Отмена коммитов | 13 |
| 3.7 | Изменение предыдущего коммита | 14 |
| 3.8 | index.html | 15 |
| 3.9 | Каталог .git | 16 |
| 3.10 | Работа непосредственно с объектами git | 16 |
| 3.11 | Редактирование файла | 17 |
| 3.12 | Просмотр имени по умолчанию удаленного репозитория | 20 |
| 3.13 | Создание чистого репозитория | 22 |
| 3.14 | Извлечение изменений | 22 |

1 Цель работы

Приобрести практические навыки работы с системой управления версиями Git.

2 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.

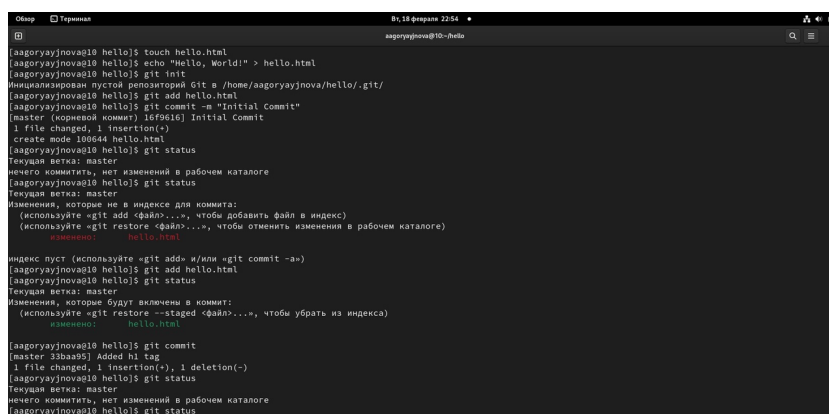
3 Выполнение лабораторной работы

3.1 Подготовка

Сначала настроим `core.autocrlf` с параметрами `true` и `input`, чтобы сделать все переводы строк текстовых файлов в главном репозитории одинаковыми, а затем настроим отображение `unicode`

3.2 Создание проекта

Создадим пустой каталог `hello`, а в нём файл с именем `hello.html`. Затем создадим `git` репозиторий из этого каталога, выполнив команду `git init`. Добавим файл в репозиторий и проверим статус, который сообщает, что коммитить нечего (рис. 3.1).



```
aaagoryayjnova@10:~$ touch hello.html
aaagoryayjnova@10:~$ echo "Hello, World!" > hello.html
aaagoryayjnova@10:~$ git init
Инициализирован пустой репозиторий git в /home/aaagoryayjnova/hello/.git/
aaagoryayjnova@10:~$ git add hello.html
[aaagoryayjnova@10:~$ git commit -m "Initial Commit"
[master (корневой коммит) 16f9e16] Initial Commit
1 file changed, 1 insertion(+)
create mode 100644 hello.html
aaagoryayjnova@10:~$ git status
Текущая ветка: master
ничего коммитить, нет изменений в рабочем каталоге
aaagoryayjnova@10:~$ git status
Текущая ветка: master
Изменения, которые не в индексе для коммита:
  (используйте «git add <файл>...», чтобы добавить файл в индекс)
  (используйте «git restore <файл>...», чтобы отменить изменения в рабочем каталоге)
    hello.html
aaagoryayjnova@10:~$ git add hello.html
aaagoryayjnova@10:~$ git status
Текущая ветка: master
Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
    hello.html
aaagoryayjnova@10:~$ git commit
[master 33baa55] Added hi tag
1 file changed, 1 insertion(+), 1 deletion(-)
aaagoryayjnova@10:~$ git status
Текущая ветка: master
ничего коммитить, нет изменений в рабочем каталоге
aaagoryayjnova@10:~$ git status
```

Рис. 3.1: Создание репозитория

3.3 Внесение изменений

Изменим содержимое файла `hello.html` на:

```
<h1>Hello, World!</h1>
```

Проверив состояние рабочего каталога увидим, что `git` знает, что файл `hello.html` был изменен, но при этом эти изменения еще не зафиксированы в репозитории. Теперь проиндексируем изменения и снова посмотрим статус, в нём указано, что изменения пока не записаны в репозиторий. И наконец закоммитим изменения, внеся их в репозиторий и снова посмотрим статус, который теперь показывает, что все изменения внесены в репозиторий.

Изменим страницу «Hello, World», чтобы она содержала стандартные теги

и

.

```
<html>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

Теперь добавим это изменение в индекс `git` и добавим заголовки HTML (секцию) к странице «Hello, World» (рис. 3.2). Проверив текущий статус увидим, что `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, затем проиндексируем оставшееся изменение, посмотрим статус и прокоммитим его.

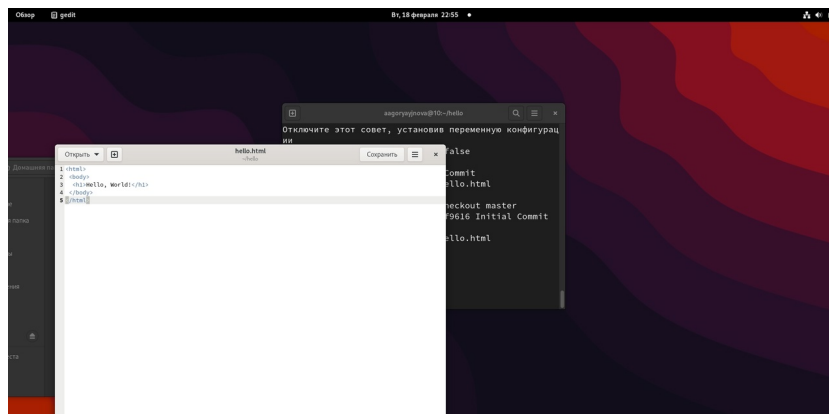


Рис. 3.2: Внесение нескольких изменений в файл

3.4 История

Получим список произведённых изменений в стандартном виде, затем в однострочном, а также с указанием времени и количества.

3.5 Получение старых версий

Изучим данные лога и найдем там хэш первого коммита, используя его вернемся к первой версии и посмотрим файл hello.html, действительно, увидим первую версию. Затем вернемся к последней версии в ветке master и вновь посмотрим на файл(рис. 3.3).

```
индекс (используйте «git add» и/или «git commit -a»)
[aaoryayjnovae10 hello]$ git add .
[aaoryayjnovae10 hello]$ git status
текущая ветка: master
Изменения, которые будут включены в коммит:
  изменено:   hello.html

[aaoryayjnovae10 hello]$ git commit -m "Added HTML header"
[master d9ca72f] Added HTML header
 1 file changed, 5 insertions(+), 1 deletion(-)
[aaoryayjnovae10 hello]$ git log
commit d9ca72f783b0f0975c5a07caf30f2ee6663803ca (HEAD -> master)
Author: inguzeva <113226441@pfur.ru>
Date:   Tue Feb 18 22:58:26 2025 +0300

    Added HTML header

commit 33baa95b8475de91218c2c710ca58ff8b32308c4
Author: inguzeva <113226441@pfur.ru>
Date:   Tue Feb 18 22:37:05 2025 +0300

    Added h1 tag

commit 16f9610aa1748763671d433c5f26844dcd9167f
Author: inguzeva <113226441@pfur.ru>
Date:   Tue Feb 18 22:33:15 2025 +0300

    Initial Commit

[aaoryayjnovae10 hello]$ git log --pretty=oneline
fatal: неопознанный аргумент: --pretty=oneline
[aaoryayjnovae10 hello]$ man git-log
[aaoryayjnovae10 hello]$ git log
commit d9ca72f783b0f0975c5a07caf30f2ee6663803ca (HEAD -> master)
```

Рис. 3.3: Просмотр разных версий репозитория

3.6 Создание тегов версий

Назовем текущую версию страницы hello первой (v1). Создадим тег первой версии и используем его для того чтобы вернуться к предыдущей, которой также присвоим тег.

Переключимся по тегам между двумя отмеченными версиями. Просмотрим все доступные теги(их два) и посмотрим теги в логе(рис. 3.4).

```
aaoryayjnovae10~$ git log master --all
HEAD сейчас на 33baa95 Added h1 tag
[aaoryayjnovae10 hello]$ git tag
v1
v1-beta
[aaoryayjnovae10 hello]$ git log master --all
error: switch 'l' ожидает числовое значение
[aaoryayjnovae10 hello]$ git log master --all
commit d9ca72f783b0f0975c5a07caf30f2ee6663803ca (tag: v1, master)
Author: inguzeva <113226441@pfur.ru>
Date:   Tue Feb 18 22:58:26 2025 +0300

    Added HTML header

commit 33baa95b8475de91218c2c710ca58ff8b32308c4 (HEAD
, tag: v1-beta)
```

Рис. 3.4: Переключение по имени тега и просмотр доступных тегов

3.7 Отмена локальных изменений (до индексации)

Убедимся, что мы находимся на последнем коммите ветки master и внесем изменение в файл hello.html в виде нежелательного комментария (рис. 3.5). Затем проверим статус, увидим, что изменения ещё не проиндексированы. Используем команду git checkout для переключения версии файла hello.html в репозитории.

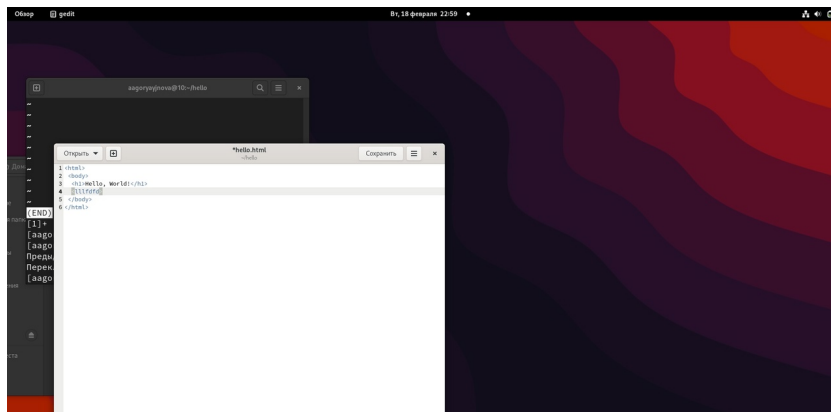


Рис. 3.5: Добавления нежелательного комментария

3.8 Отмена проиндексированных изменений (перед коммитом)

Внесем изменение в файл hello.html в виде нежелательного комментария

```
<html>
  <head>
    <!-- This is an unwanted but staged comment -->
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

Проиндексируем это изменение и проверим состояние. Состояние показывает, что изменение было проиндексировано и готово к коммиту. Используем команду `git reset`, чтобы сбросить буферную зону к HEAD. Это очищает буферную зону от изменений, которые мы только что проиндексировали. И переключимся на последнюю версию коммита, посмотрев статус увидим, что наш каталог опять чист.

3.9 Отмена коммитов

Изменим файл `hello.html` на следующий.

```
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <!-- This is an unwanted but committed change -->
  </body>
</html>
```

Проиндексируем изменения файла и прокоммитим их. Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом. Перейдем в редактор, где изменим нежелательный коммит. Проверим лог. Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий(рис. 3.6).

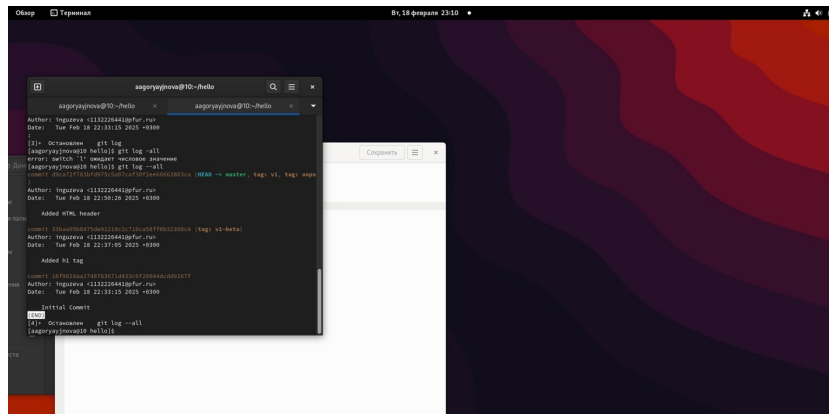


Рис. 3.6: Отмена коммитов

3.10 Удаление коммитов из ветки

Удалим последние два коммита с помощью сброса, сначала отметим последний коммит тегом, чтобы его можно было потом найти. Используем команду `git reset`, чтобы вернуться к версии до этих коммитов. Теперь в логе их нет, но если посмотреть логи с опцией `-all` можно всё ещё их увидеть, но метка HEAD находится на нужной нам версии.

3.11 Удаление тега оорс

Удалим тег оорс и коммиты, на которые он ссылался, сборщиком мусора. Теперь этот тег не отображается в репозитории.

3.12 Изменение предыдущего коммита

Добавим в страницу комментариев автора.

Затем добавим их в репозиторий. Теперь мы хотим добавить в комментарий автора почту, обновиим страницу hello, включив в неё почту. Чтобы у нас остался

один коммит, а не два, изменим последний с помощью опции `-amend`, теперь в логах отображается последняя версия коммита (рис. 3.7).

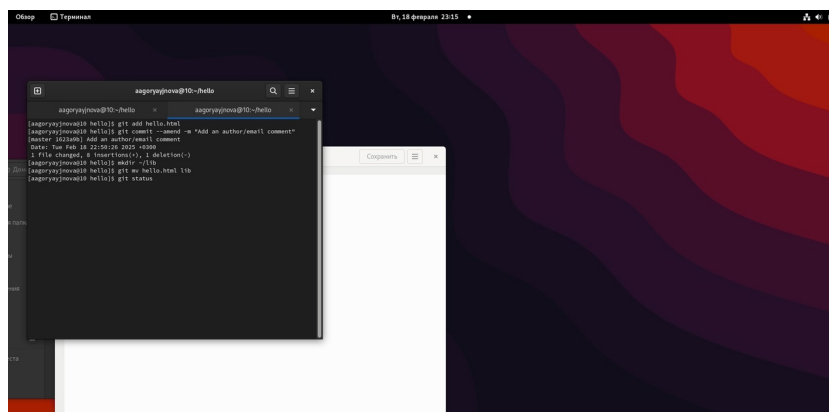


Рис. 3.7: Изменение предыдущего коммита

3.13 Перемещение файлов

Переместим наш файл в каталог `lib`. Для этого создадим его и используем команду `git mv`, сделаем коммит этого перемещения.

3.14 Подробнее о структуре

Добавим файл `index.html` в наш репозиторий

```
<html>
  <body>
    <iframe src="lib/hello.html" width="200" height="200" />
  </body>
</html>
```

Добавим файл и сделаем коммит. (рис. @fig:008).

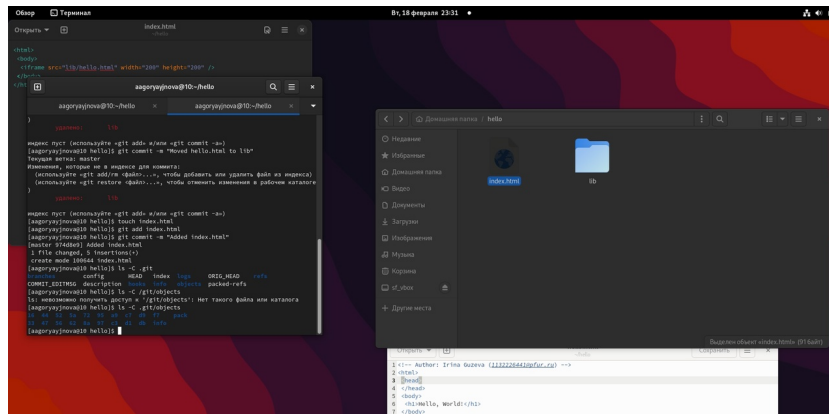


Рис. 3.8: index.html

Теперь при открытии index.html, увидим кусок страницы hello в маленьком окошке

3.15 Git внутри: Каталог .git

Просмотрим каталог, в котором хранится вся информация git. Затем посмотрим набор каталогов, имена которых состоят из 2 символов. Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося в git. Посмотрим в один из каталогов с именем из 2 букв. Увидим файлы с именами из 38 символов. Это файлы, содержащие объекты, хранящиеся в git. Посмотрим файл конфигурации, создающийся для каждого конкретного проекта. Затем посмотрим подкаталоги .git/refs/heads и .git/refs/tags, а также содержимое файла v1, в нём хранится хэш коммита, привязанный к тегу. Также посмотрим содержимое файла HEAD, который содержит ссылку на текущую ветку, в данный момент это ветка master(рис. 3.9).

3.17 Создание ветки

Создадим новую ветку «style» и перейдем в неё. Добавим туда файл стилей style.css и добавим его в репозиторий. Обновим файл hello.html, чтобы использовать стили style.css и index.html, также обновим их в репозиторий (рис. 3.11).

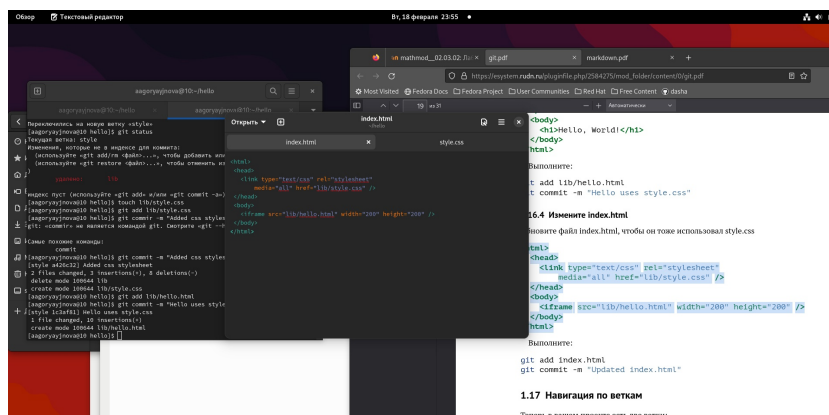


Рис. 3.11: Редактирование файла

3.18 Навигация по веткам

Посмотрим все логи. Переключимся обратно на основную ветку и посмотрим содержимое файла `lib/hello.html`, заметим, что он не использует стили, также посмотрим содержимое этого файла в новой ветке.

3.19 Изменения в ветке master

Вернемся в основную ветку и добавим файл `README.md`. Просмотрим ветки и их различия.

3.20 Слияние

Слияние переносит изменения из двух веток в одну. Вернемся к ветке `style` и сольем `master` с `style`.

3.21 Создание конфликта

Вернемся в ветку `master` и создадим конфликт, внося изменения в файл `hello.html`. Просмотрим ветки. 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`.

3.22 Разрешение конфликтов

Вернемся к ветке `style` и попытаемся объединить ее с новой веткой `master`. В файле `lib/hello.html` можно увидеть записи с обеих версий этого файла. Первый раздел — версия текущей ветки (`style`). Второй раздел — версия ветки `master`. Внесем изменения в `lib/hello.html`, оставив только необходимую нам запись и добавим этот файл в репозиторий, чтобы вручную разрешить конфликт.

3.23 Сброс ветки `style`

Вернемся на ветке `style` к точке перед тем, как мы слили ее с веткой `master`. Мы хотим вернуться в ветке `style` в точку перед слиянием с `master`. Нам необходимо найти последний коммит перед слиянием.

Мы видим, что коммит «Updated index.html» был последним на ветке `style` перед слиянием. Сбросим ветку `style` к этому коммиту.

Поищем лог ветки `style`. Увидим, что у нас в истории больше нет коммитов слияний.

3.24 Сброс ветки `master`

Добавив интерактивный режим в ветку `master`, мы внесли изменения, конфликтующие с изменениями в ветке `style`. Давайте вернемся в ветку `master` в точку перед внесением конфликтующих изменений. Это позволяет нам продемонстрировать работу команды `git rebase`, не беспокоясь о конфликтах. Просмотрим коммиты ветки `master`.

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

3.25 Перебазирование

Используем команду `rebase` вместо команды `merge`. Мы вернулись в точку до первого слияния и хотим перенести изменения из ветки `master` в нашу ветку `style`. На этот раз для переноса изменений из ветки `master` мы будем использовать команду `git rebase` вместо слияния.

3.26 Слияние в ветку `master`

Вернемся в ветку `master` и сольем ветку `style` в неё с помощью команды `git merge`.

3.27 Клонирование репозитория

Перейдем в наш рабочий каталог и сделаем клон репозитория hello, затем создадим клон репозитория. Просмотрев его увидим список всех файлов на верхнем уровне оригинального репозитория README.md, index.html и lib. Затем посмотрим историю репозитория и увидим список всех коммитов в новый репозиторий, и он совпадает с историей коммитов в оригинальном репозитории. Единствен в названиях веток.

3.28 Что такое origin?

Клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Посмотрим, подробную информацию об имени по умолчанию. Для того, чтобы увидеть все ветки используем опцию -a(рис. 3.12).

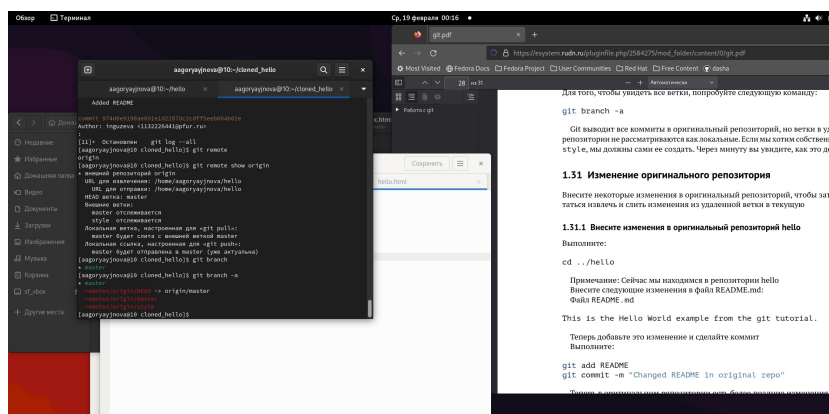


Рис. 3.12: Просмотр имени по умолчанию удаленного репозитория

3.29 Удаленные ветки

Посмотрим на ветки, доступные в нашем клонированном репозитории. Можно увидеть, что в списке только ветка master.

3.30 Изменение оригинального репозитория

Перейдем в репозиторий `hello`. Внесем изменения в файл `README.md`. Затем добавим их в репозиторий. Перейдём в клон репозитория и используем команду `git fetch`, которая будет извлекать новые коммиты из удаленного репозитория, но не будет сливать их с наработками в локальных ветках.

3.31 Слияние извлеченных изменений

Сольем внесённые изменения в главную ветку. Также можно было бы использовать команду `git pull`, которая является объединением `fetch` и `merge` в одну команду.

3.32 Добавление ветки наблюдения

Добавим локальную ветку, которая отслеживает удаленную ветку, теперь мы можем видеть ветку `style` в списке веток и логе.

3.33 Создание чистого репозитория

Как правило, репозитории, оканчивающиеся на `.git` являются чистыми репозиториями. Создадим такой в рабочем каталоге. Затем добавим репозиторий `hello.git` к нашему оригинальному репозиторию(рис. 3.13).

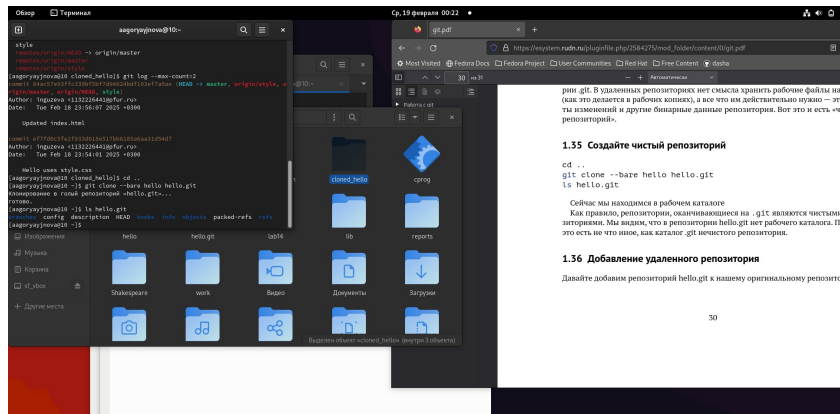


Рис. 3.13: Создание чистого репозитория

3.34 Отправка и извлечение изменений

Так как чистые репозитории, как правило, расшариваются на каком-нибудь сетевом сервере, нам необходимо отправить наши изменения в другие репозитории. Начнем с создания изменения для отправки. Отредактируем файл README.md и сделаем коммит, затем отправим изменения в общий репозиторий. Затем извлечем изменения из общего репозитория (рис. 3.14).

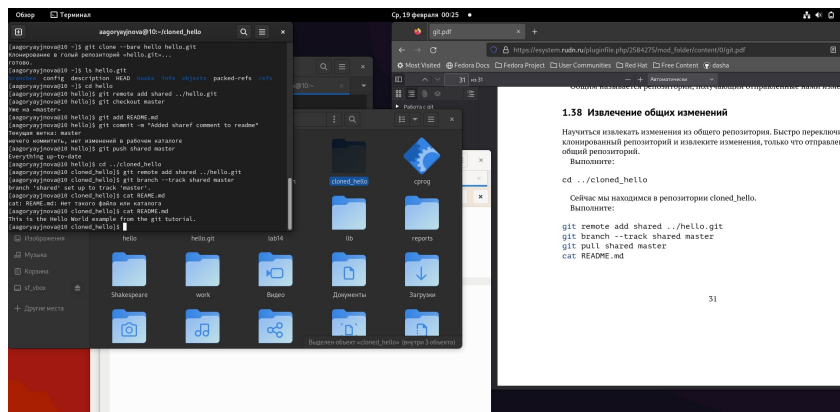


Рис. 3.14: Извлечение изменений

4 Выводы

В процессе выполнения данной лабораторной работы я приобрел практические навыки работы с Git.

Список литературы