

МОЛДАВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ
ДЕПАРТАМЕНТ ИНФОРМАТИКИ

МИХАЙ Даниел

ИЗУЧЕНИЕ LATEX И GIT

0613.4 ИНФОРМАТИКА

Практика

Директор департамента: _____
(подпись)

КАПЧЕЛЯ Титу,
доктор физико-математических наук,
преподаватель университета

Научный руководитель: _____
(подпись)

КУРМАНСКИЙ Антон,
ассистент университета

Автор: _____
(подпись)

МИХАЙ Даниел,
студент группы I2402

КИШИНЕВ – 2025 Г.

Оглавление

Аннотация	3
ВВЕДЕНИЕ	5
I. Введение в систему вёрстки L^AT_EX	7
1.1. Установка L ^A T _E X на macOS	7
1.2. Использование шаблона диплома	7
1.3. Выводы	8
II. Введение в систему контроля версий Git	9
2.1. Зачем нужен Git?	9
2.2. История создания Git	9
2.3. Ключевые особенности Git	9
2.4. Основные команды Git	9
2.5. Источники и полезные ссылки	10
2.6. Выводы для главы 2	10
III. Практика с Learn Git Branching	11
3.1. Main — Базовые концепции Git	11
3.1.1. Introduction Sequence	11
3.1.2. Ramping Up	14
3.1.3. Moving Work Around	18
3.1.4. A Mixed Bag	20
3.1.5. Advanced Topics	24
3.2. Remote — Работа с удалёнными репозиториями	27
3.2.1. Push & Pull — Git Remotes	27
3.2.2. To Origin and Beyond — Advanced Git Remotes	34
3.3. Выводы для главы 3	39
IV. Практика с Git: Git Immersion (Задания 1–33)	40
Заключение и рекомендации	63

Аннотация

Практическая работа на тему “Изучение L^AT_EX и Git”, студента Михай Даниел, группы I2402.

Структура практической работы. Практическая работа состоит из: Введения, трёх глав, Заключения с рекомендациями, библиографии и приложений. В работе представлены теоретические сведения, практические задания и иллюстрации, отражающие процесс изучения систем L^AT_EX и Git. Основной текст включает подробные инструкции и скриншоты, демонстрирующие выполнение заданий на платформе Learn Git Branching [1].

Актуальность. Системы управления версиями и профессиональной вёрстки документов играют важную роль в современной IT-индустрии. Git широко используется в командной разработке программного обеспечения [2], [3], а L^AT_EX — в научной и инженерной документации [4], [5], [6]. Их знание необходимо для любого начинающего специалиста в сфере ИТ.

Цель и задачи исследования. Целью данной работы является освоение основ использования L^AT_EX и Git, а также практическое применение полученных знаний на платформе Learn Git Branching.

Для достижения цели были поставлены следующие задачи:

- Изучить синтаксис и особенности вёрстки с использованием L^AT_EX [4], [5];
- Освоить ключевые команды Git и принципы работы с системой контроля версий [3];
- Пройти практические задания Learn Git Branching по ветвлению, слиянию и удалённым репозиториям [1];
- Подготовить отчётный документ в L^AT_EX, используя современные пакеты, включая подсветку кода с помощью `minted` [7].

Ожидаемые и полученные результаты. Ождалось овладение базовыми навыками вёрстки и контроля версий. В результате был оформлен структурированный документ в L^AT_EX, содержащий полную практическую часть по Git и Learn Git Branching, включая графические иллюстрации и разъяснения к каждому этапу.

Важные решённые проблемы. В процессе работы были решены следующие задачи:

- организация многоуровневой структуры документа в L^AT_EX;
- изучение моделей ветвления и командной работы в Git;
- визуализация выполнения практических заданий;
- интеграция технического и визуального материала в единую отчётную работу.

Практическая ценность. Результаты работы могут быть использованы как руководство для студентов, начинающих осваивать L^AT_EX и Git, а также как база для подготовки отчётной и научной документации в технических вузах.

Весь исходный код проекта доступен на GitHub по следующей ссылке: https://github.com/messsimo/uni_practice.

ВВЕДЕНИЕ

Актуальность и важность темы.

Освоение систем \LaTeX и Git [3], [4] является важным шагом в формировании технической грамотности современного ИТ-специалиста. Git применяется повсеместно в процессе разработки программного обеспечения, позволяя эффективно управлять изменениями и взаимодействовать в команде. \LaTeX , в свою очередь, обеспечивает профессиональную вёрстку документов, широко используемую в научной, инженерной и образовательной среде [5], [6].

Практическая направленность данной работы, подкреплённая выполнением заданий на интерактивной платформе Learn Git Branching [1], делает изучение не только теоретически обоснованным, но и прикладным.

Цель и задачи.

Цель — получить практические навыки работы с \LaTeX и Git, а также закрепить знания через выполнение реальных задач, включая визуализацию ветвлений и взаимодействие с удалёнными репозиториями.

Задачи:

- Изучить принципы и команды Git, в том числе ветвление, слияние и работу с удалёнными репозиториями;
- Освоить синтаксис \LaTeX , включая структуру документа, оформление кода, таблиц, рисунков;
- Подготовить отчёт с использованием современных пакетов (`minted`, `graphicx` и др.) [7];
- Пройти все ключевые блоки платформы Learn Git Branching: Introduction, Ramping Up, Moving Work Around, Mixed Bag и Advanced Topics;
- Создать структурированный отчёт, включающий скриншоты выполнения заданий и объяснение решений.

Методологическая и технологическая база.

В работе использованы:

- Платформа Learn Git Branching [1] для симуляции работы с Git;
- Компилятор XeLaTeX для создания PDF-документа;
- Пакеты `minted`, `graphicx`, `biblatex` для оформления кода, иллюстраций и библиографии;
- Git CLI и веб-интерфейс GitHub (в обучающей форме, без публикации проекта).

Научная новизна / оригинальность.

Работа сочетает сразу два технологических направления — вёрстку и контроль версий — и демонстрирует практическое применение обеих технологий в едином отчёте. Также показан прогрессивный подход к обучению Git через визуальное взаимодействие, что повышает понимание абстрактных концепций.

Практическая ценность.

Полученные материалы могут использоваться:

- как пособие для студентов технических направлений;
- как пример оформления отчётов в L^AT_EX;
- как руководство по решению задач Git через Learn Git Branching.

Краткое содержание работы.

- Первая глава знакомит с основами системы L^AT_EX, её назначением, преимуществами и базовыми конструкциями;
- Вторая глава описывает Git, его возможности и применимость в командной разработке;
- Третья глава посвящена практическому выполнению задач на платформе Learn Git Branching с подробным разбором каждой секции.

I Введение в систему вёрстки L^AT_EX

1.1 Установка L^AT_EX на macOS

Для начала работы с L^AT_EX на операционной системе macOS рекомендуется использовать дистрибутив MacTeX, который включает в себя все необходимые компоненты для компиляции документов.

Шаг 1. Установка Homebrew

Homebrew — это менеджер пакетов для macOS. Если он ещё не установлен, выполните следующую команду в терминале:

```
1 /bin/bash -c "$(curl -fsSL
→ https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Шаг 2. Установка MacTeX и дополнительных шрифтов

Установите MacTeX и необходимые шрифты с помощью Homebrew:

```
1 brew install --cask mactex font-liberation font-times-new-roman
```

После установки рекомендуется открыть утилиту **TeX Live Utility** и обновить все доступные пакеты.

Шаг 3. Установка Python и Pygments

Для использования подсветки синтаксиса (через пакет `minted`), необходимо установить Python и пакет Pygments:

```
1 brew install python
2 pip install Pygments
```

1.2 Использование шаблона диплома

Клонирование репозитория

Для начала работы с шаблоном, склонируйте репозиторий из GitHub:

```
1 cd ~
2 git clone https://github.com/antonc9018/uni_thesisTemplate
3 cd uni_thesisTemplate
```

Выбор языка

Шаблон содержит версии на разных языках. Для выбора нужного языка переименуйте соответствующий файл:

```
1 mv thesis/bare_main_ro.tex thesis/main.tex
```

Компиляция PDF-документа

Перейдите в директорию с шаблоном и выполните скрипт рендеринга:

```
1 cd thesis  
2 ./render.sh
```

После выполнения скрипта будет скомпилирован финальный PDF-документ, готовый к подаче.

1.3 Выводы

macOS предоставляет стабильную и удобную среду для работы с L^AT_EX. С помощью Homebrew можно быстро установить все необходимые компоненты, включая MacTeX, Python и средства для подсветки синтаксиса. Использование готового шаблона с GitHub позволяет быстро начать работу над дипломом или другим техническим документом.

II Введение в систему контроля версий Git

2.1 Зачем нужен Git?

Git — это распределённая система контроля версий, созданная для эффективного управления изменениями в проектах с большим количеством исходного кода. Он позволяет:

- отслеживать историю изменений;
- работать в команде без конфликтов;
- откатываться к предыдущим версиям кода;
- создавать альтернативные ветви разработки (branching);
- автоматически сливать изменения (merging).

2.2 История создания Git

Git был создан Линусом Торвальдсом в 2005 году для управления исходным кодом ядра Linux после конфликта с предыдущей системой контроля версий — BitKeeper [3]. Требования:

- высокая скорость работы;
- надёжная защита от потери данных;
- поддержка распределённой архитектуры;
- лёгкость в ветвлении и слиянии.

2.3 Ключевые особенности Git

- Каждый разработчик имеет полную копию репозитория (локально).
- Все изменения сохраняются в виде снапшотов.
- Ветвление и слияние — базовая часть рабочего процесса.
- Минимальная зависимость от центрального сервера.

Подробнее о философии Git — в книге Pro Git [3].

2.4 Основные команды Git

- `git init` — инициализация репозитория.
- `git clone` — копирование удалённого репозитория.
- `git status` — просмотр состояния файлов.
- `git add` — добавление файлов в индекс.
- `git commit` — фиксация изменений.
- `git branch` — работа с ветками.

- `git checkout` — переключение между ветками.
- `git merge` — слияние веток.
- `git pull, git push` — взаимодействие с удалёнными репозиториями.

Команды подробно описаны в официальной документации [2].

2.5 Источники и полезные ссылки

- [2] — официальная документация Git.
- [1] — платформа интерактивного обучения Learn Git Branching.
- [3] — Pro Git book.

2.6 Выводы для главы 2

Вторая глава была посвящена изучению системы контроля версий Git. Рассмотрены её архитектура, принципы работы и основные команды. Были объяснены понятия коммита, индексации, истории изменений, а также таких операций, как ветвление (branching), слияние (merging) и откат изменений (reset, revert) [2], [3].

Git продемонстрировал себя как надёжный инструмент для индивидуальной и командной разработки. Его использование позволяет систематизировать процесс разработки, минимизировать риски потери данных и упростить интеграцию изменений. Глава заложила основу для выполнения практических задач на платформе Learn Git Branching.

III Практика с Learn Git Branching

В качестве обучающей платформы используется интерактивный тренажёр Learn Git Branching [1].

Важно: последующая работа подразумевает что читатель уже ознакомился с источниками [1], [2], [3].

3.1 Main — Базовые концепции Git

3.1.1 Introduction Sequence

Introduction to Git Commits

Цель: познакомиться с понятием коммита в Git.

Коммит в Git – это снимок состояния всех отслеживаемых файлов. Git старается хранить только изменения (дeltas), чтобы минимизировать использование памяти и ускорить операции. Каждый коммит связан с предыдущим (родительским), что позволяет формировать полную историю проекта.

Команда:

```
1 git commit
```

фиг. 3.1 представлен скриншот данного задания.

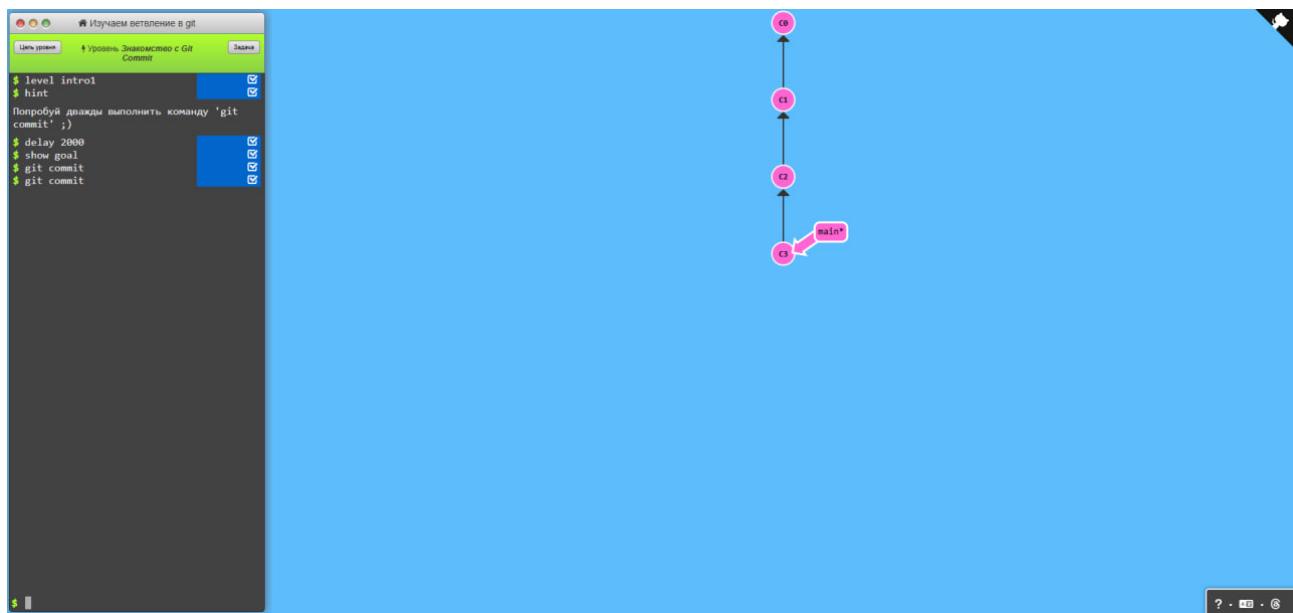


Рис. 3.1

После выполнения создаётся новый коммит, например C2, ссылающийся на C1 как на родителя.

Branching in Git

Цель: научиться создавать и переключаться между ветками.

Ветки в Git – это указатели на определённые коммиты. Они практически не занимают места, поэтому создавать много веток – это нормально и даже рекомендуется.

Команды:

```
1 git branch newImage  # создание новой ветки
2 git checkout newImage # переключение на неё
3 git commit            # коммит будет в ветке newImage
```

Альтернатива:

```
1 git checkout -b bugFix
```

фиг. 3.2 представлен скриншот данного задания.

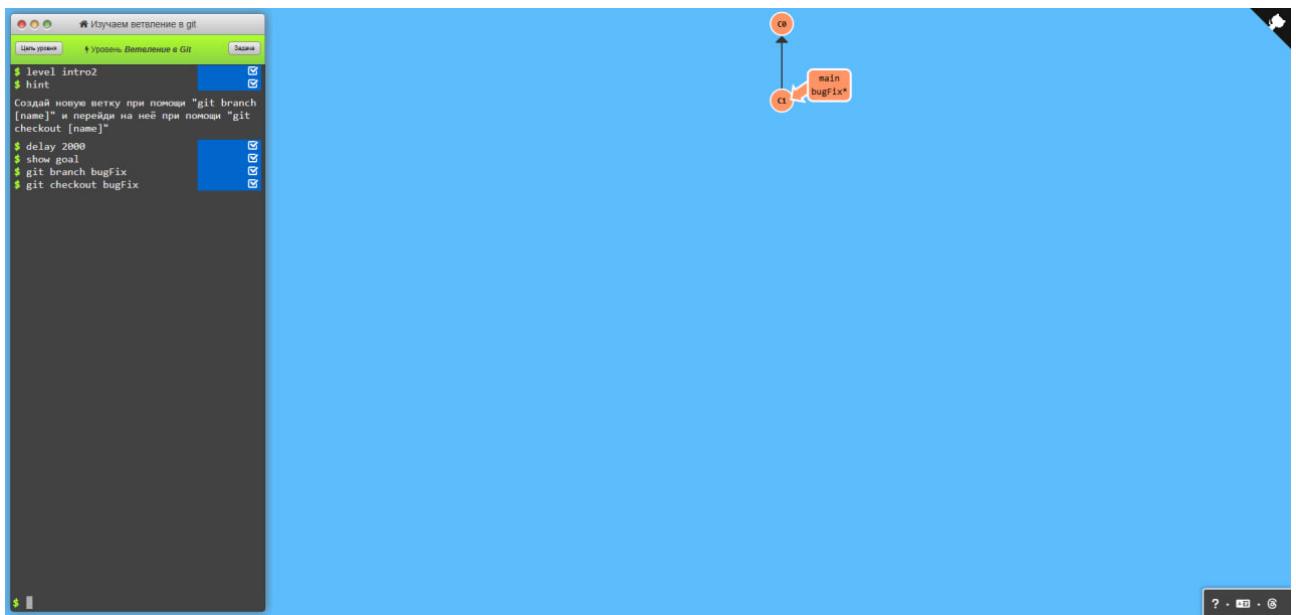


Рис. 3.2

Важно: коммиты записываются только в активную ветку (ту, где стоит символ *).

Merging in Git

Цель: объединить работу из разных веток в одну.

Merge создаёт специальный коммит с двумя родителями, включающий изменения из обеих веток. Это особенно полезно для совместной работы.

Команды:

```
1 git merge bugFix      # слияние ветки bugFix в текущую (main)
2 git checkout bugFix    # переход на другую ветку
3 git merge main        # слияние main в bugFix
```

фиг. 3.3 представлен скриншот данного задания.

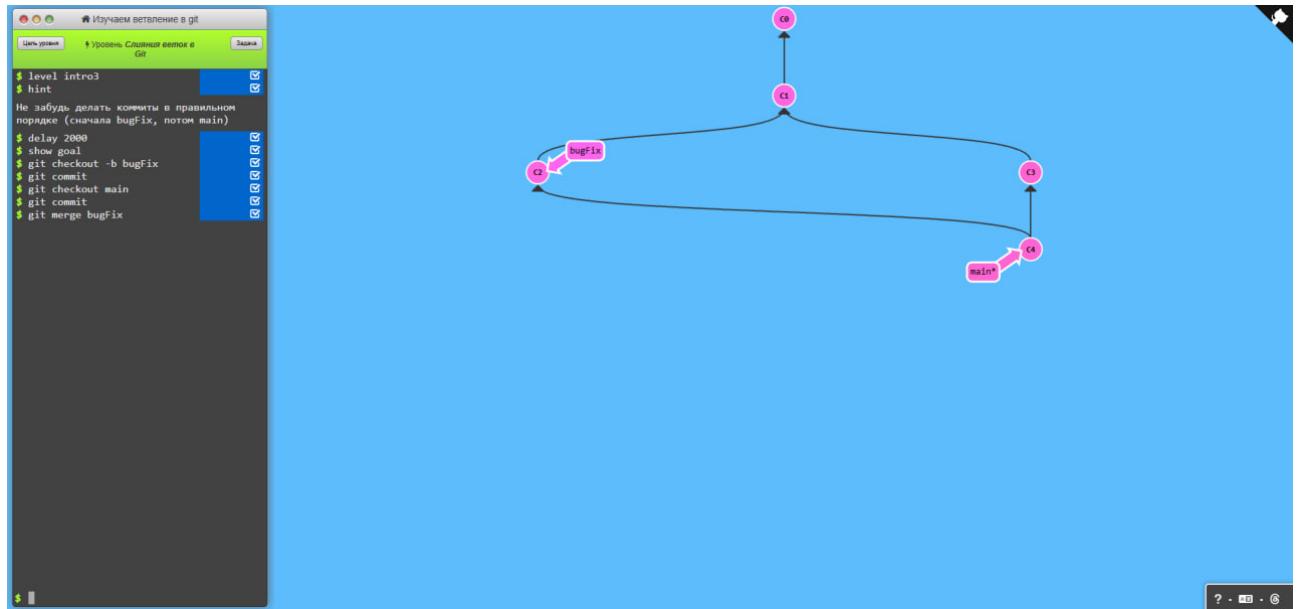


Рис. 3.3

После слияния все ветки включают весь набор коммитов проекта.

Rebase Introduction

Цель: познакомиться с альтернативным способом объединения веток.

Rebase – это перемещение коммитов из одной ветки на основание другой. Это делает историю проекта линейной и упрощает её анализ. Однако следует использовать осторожно, особенно при работе с общими ветками.

Команды:

```
1 git rebase main      # переместить коммиты bugFix поверх main
2 git checkout main
3 git rebase bugFix    # переместить main после bugFix
```

фиг. 3.4 представлен скриншот данного задания.

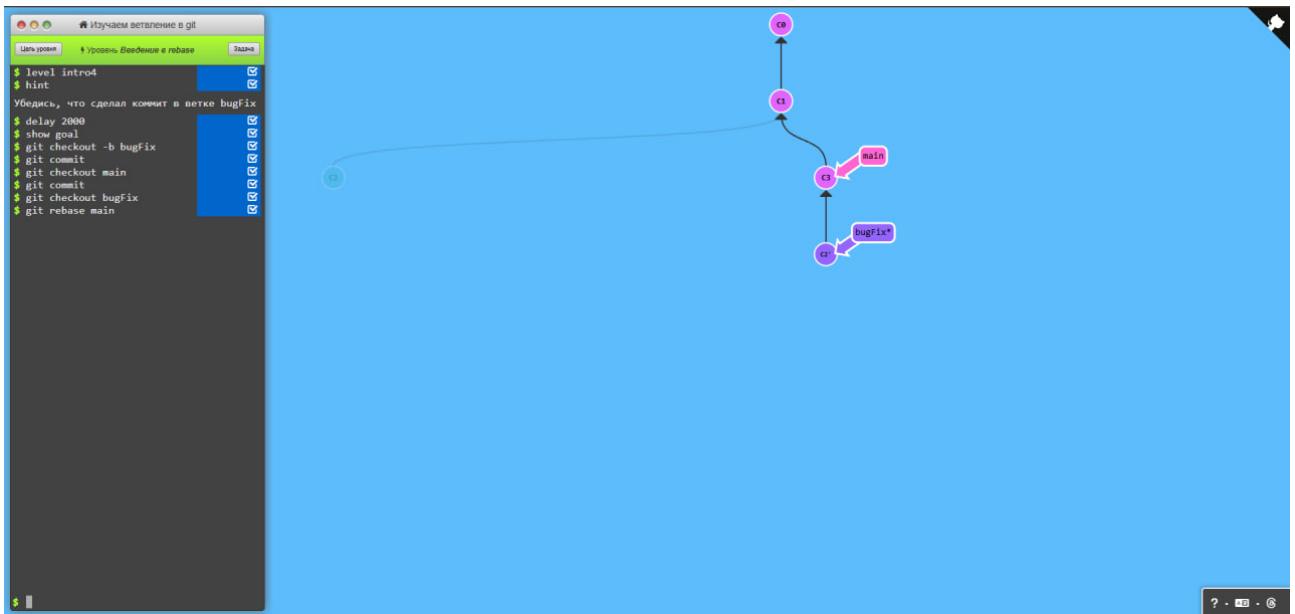


Рис. 3.4

Важно: Rebase изменяет историю коммитов и создаёт новые идентификаторы для "перемещённых" коммитов.

Дополнительную информацию о командах и концепциях Git можно найти в документации и книге Pro Git.

3.1.2 Ramping Up

Detach yo' HEAD

Цель: понять, как работает HEAD и что означает его "отсоединение".

В Git переменная HEAD указывает на текущий коммит или ветку, на которой вы находитесь. Обычно она указывает на имя ветки (например, `main`), но можно напрямую привязать HEAD к коммиту.

Это называется "отсоединённым HEAD" (detached HEAD), и в этом режиме коммиты не записываются ни в какую ветку, пока не будет выполнено явно указание.

Команды:

```
1 git checkout C1      # переход на конкретный коммит
```

Результат: HEAD указывает напрямую на коммит C1, а не на ветку.

Практика: Проверьте команды:

```
1 git checkout main
2 git commit
```

Затем отсоедините HEAD:

```
1 git checkout C2
```

фиг. 3.5 представлен скриншот данного задания.

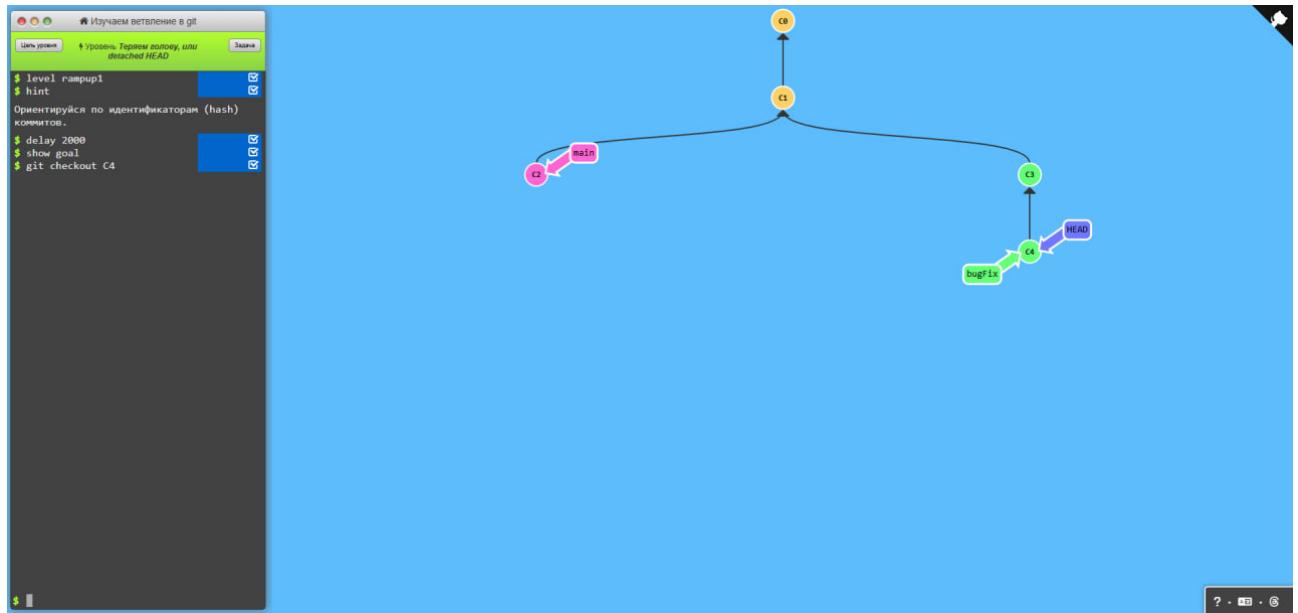


Рис. 3.5

Relative Refs (^)

Цель: освоить навигацию по коммитам с помощью относительных ссылок.

Вместо указания полного хеша коммита, Git позволяет перемещаться по дереву коммитов с помощью символов ^ и ~:

- `main^` — родитель `main`; - `main^^` — дедушка; - `HEAD^` — один шаг назад от текущего коммита.

Команды:

```
1 git checkout main^
2 git checkout HEAD^
3 git checkout HEAD^
```

фиг. 3.6 представлен скриншот данного задания.

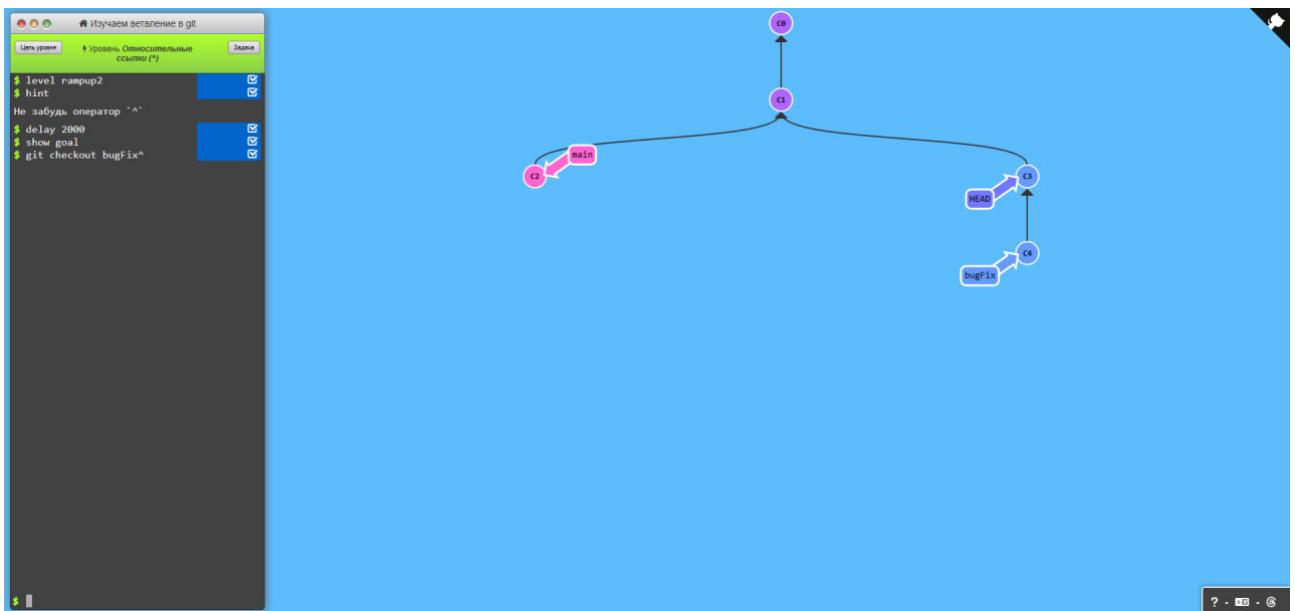


Рис. 3.6

Результат: Вы перемещаетесь назад по истории коммитов.

Relative Refs ()

Цель: быстро перемещаться назад по истории на несколько шагов.

HEAD~3 эквивалентно HEAD^^^, но значительно короче.

Команды:

```
1 git checkout HEAD~4
```

Можно использовать это в связке с -f (force) для перемещения веток:

```
1 git branch -f main HEAD~3
```

фиг. 3.7 представлен скриншот данного задания.



Рис. 3.7. Скиншот задания

Примечание: В реальных условиях запрещено перемещать текущую ветку таким способом — сначала нужно переключиться на другую.

Reversing Changes in Git

Цель: научиться отменять изменения с помощью `reset` и `revert`.

git reset — откат ветки на предыдущий коммит (используется для локальной работы).

```
1 git reset HEAD~1
```

git revert — создаёт новый коммит, отменяющий предыдущий (удобно для публичных веток).

```
1 git revert HEAD
```

фиг. 3.8 представлен скриншот данного задания.

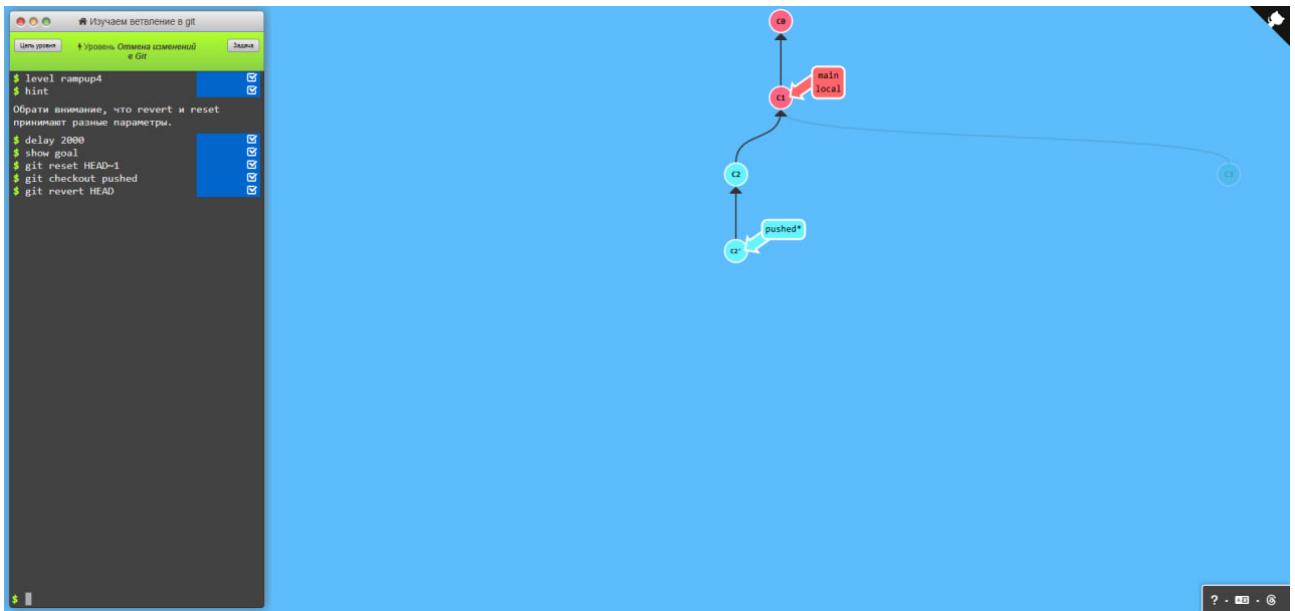


Рис. 3.8. Скиншот задания

Вывод: Reset переписывает историю. Revert создаёт откат как отдельный коммит и безопасен для совместной работы.

3.1.3 Moving Work Around

Cherry-pick Intro

Цель: научиться переносить отдельные коммиты между ветками.

Команда `git cherry-pick` позволяет выбрать один (или несколько) коммитов и "применить" их в текущую ветку. Это удобно, когда нужно скопировать конкретные изменения, не слияя всю ветку целиком.

Сценарий: допустим, нужный коммит есть в ветке `feature`, но мы хотим перенести его в `main` без `merge` или `rebase`.

Команда:

```
1 git cherry-pick <hash-коммита>
```

Пример:

```
1 git checkout main
2 git cherry-pick C4
```

фиг. 3.9 представлен скриншот данного задания.

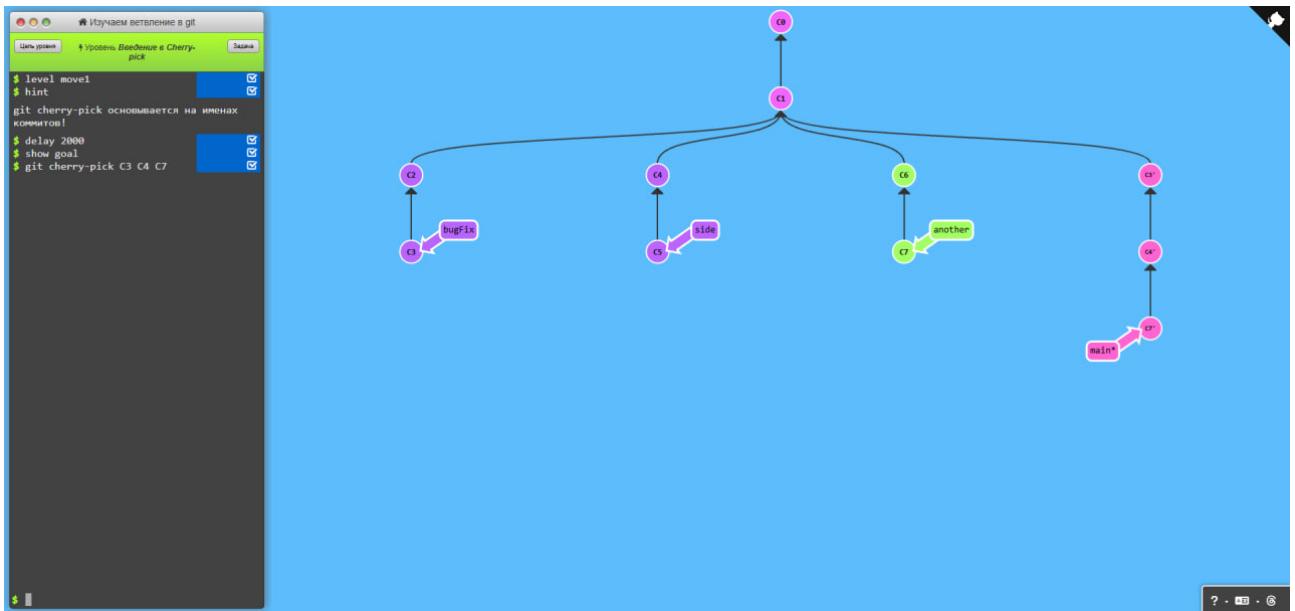


Рис. 3.9

Результат: Ветка `main` получит копию коммита `C4`.

Важно: коммит получает новый hash (так как история изменяется).

Interactive Rebase Intro

Цель: познакомиться с интерактивной перезаписью истории коммитов.

`git rebase -i` позволяет:

- изменить порядок коммитов;
- объединить коммиты (squash);
- удалить ненужные коммиты;
- изменить сообщения к коммитам.

Команда:

```
1 git rebase -i HEAD~3
```

После запуска откроется список последних 3 коммитов. В интерактивном режиме можно выбрать действия: `pick`, `reword`, `edit`, `squash`, `drop`.

Пример: чтобы объединить два последних коммита:

```
1 pick 1a2b3c Первый коммит
2 squash 4d5e6f Второй коммит
```

фиг. 3.10 представлен скриншот данного задания.

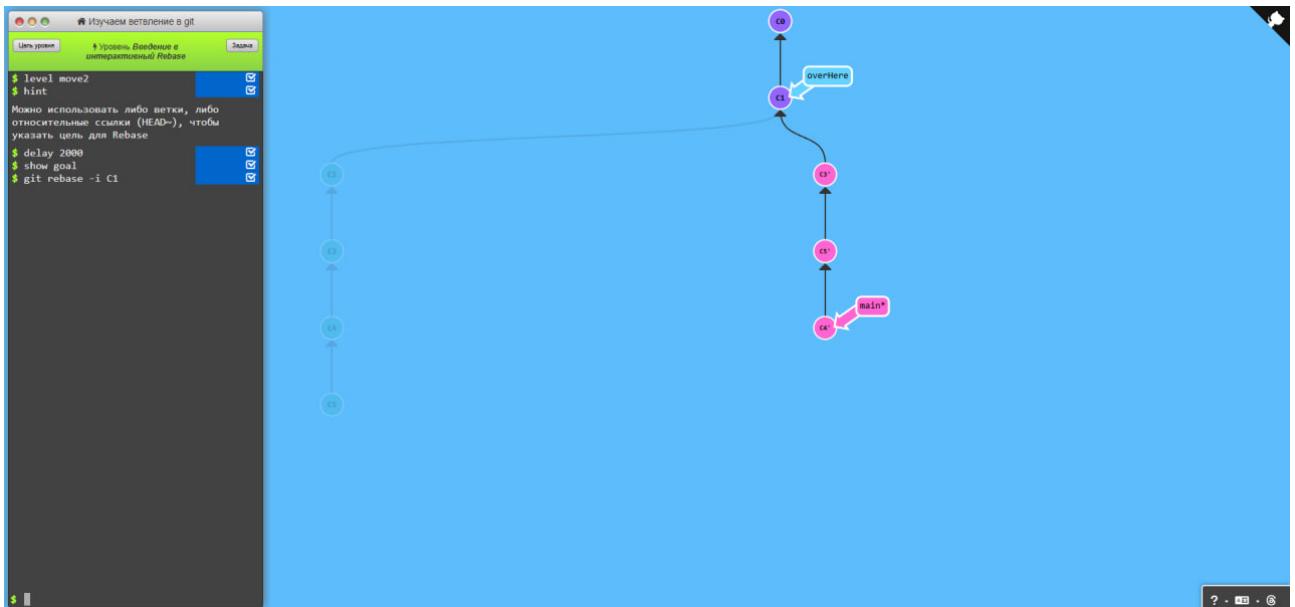


Рис. 3.10

Результат: оба коммита объединяются в один.

Важно: интерактивный rebase лучше использовать только в локальных ветках, до публикации в удалённый репозиторий.

3.1.4 A Mixed Bag

Grabbing Just 1 Commit

Цель: изолированно скопировать один конкретный коммит из другой ветки.

Иногда необходимо взять только один коммит из другой ветки, не объединяя всё её содержимое. Это решается с помощью `git cherry-pick`.

Команды:

```
1 git checkout main
2 git cherry-pick <hash-коммита>
```

фиг. 3.11 представлен скриншот данного задания.

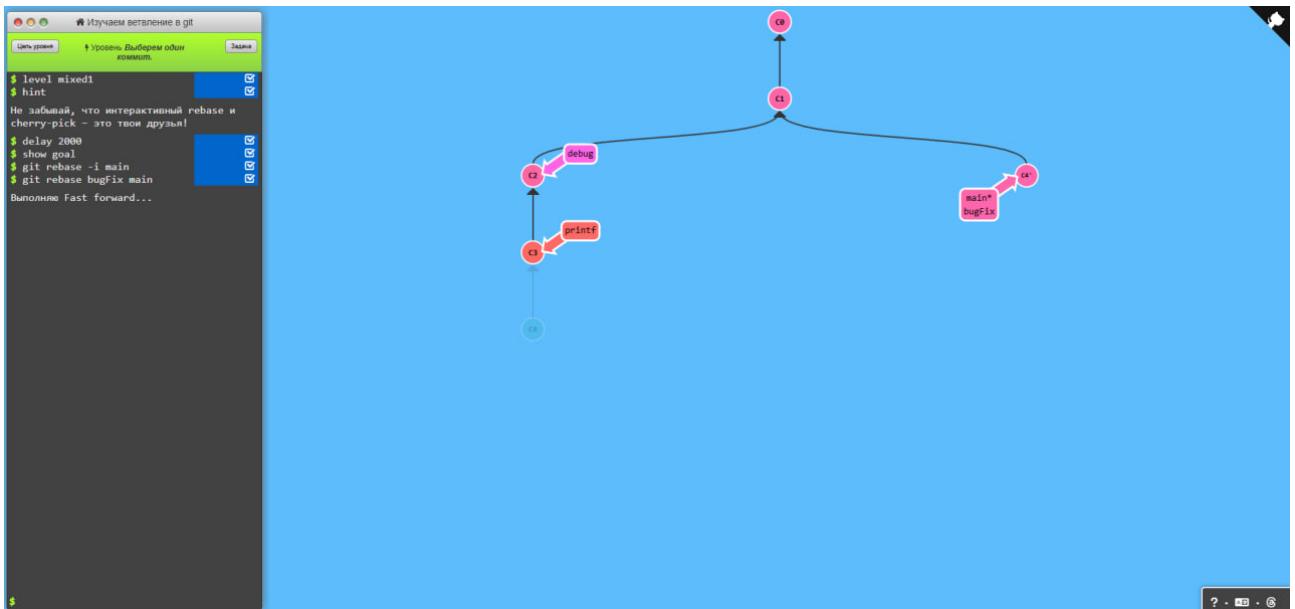


Рис. 3.11

Сценарий: один коммит на ветке содержит багфикс — берём только его.

Juggling Commits

Цель: научиться перемещать ветки и коммиты с помощью `branch -f` и `checkout`.

Можно переназначать ветки на другие коммиты, используя ключ `-f` (force). Это мощный инструмент для манипуляций в истории.

Команды:

```
1 git branch -f main HEAD~2
2 git checkout C1
```

фиг. 3.12 представлен скриншот данного задания.

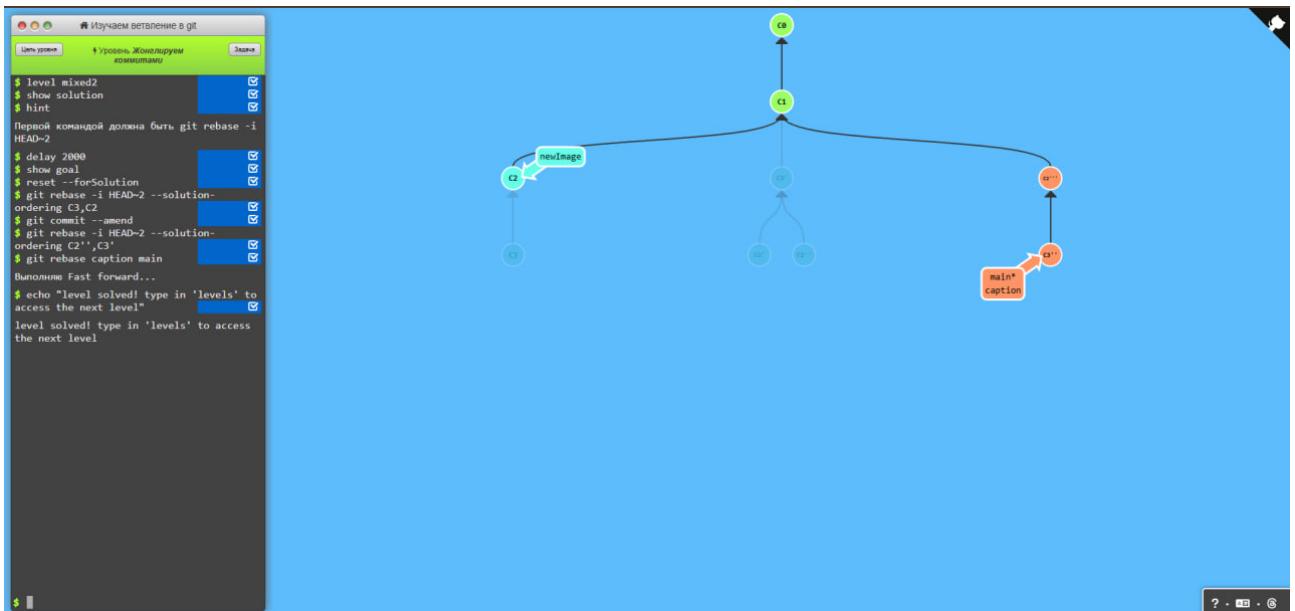


Рис. 3.12

Примечание: подобные операции применимы только к локальным веткам.

Juggling Commits #2

Цель: повторить и закрепить работу с переопределением веток.

Продолжение предыдущего задания. Здесь нужно грамотно совместить `checkout`, `commit` и `branch -f` для создания точной структуры коммитов.

Команды:

```

1 git checkout C0
2 git branch -f bugFix HEAD
3 git checkout bugFix
4 git commit

```

фиг. 3.13 представлен скриншот данного задания.

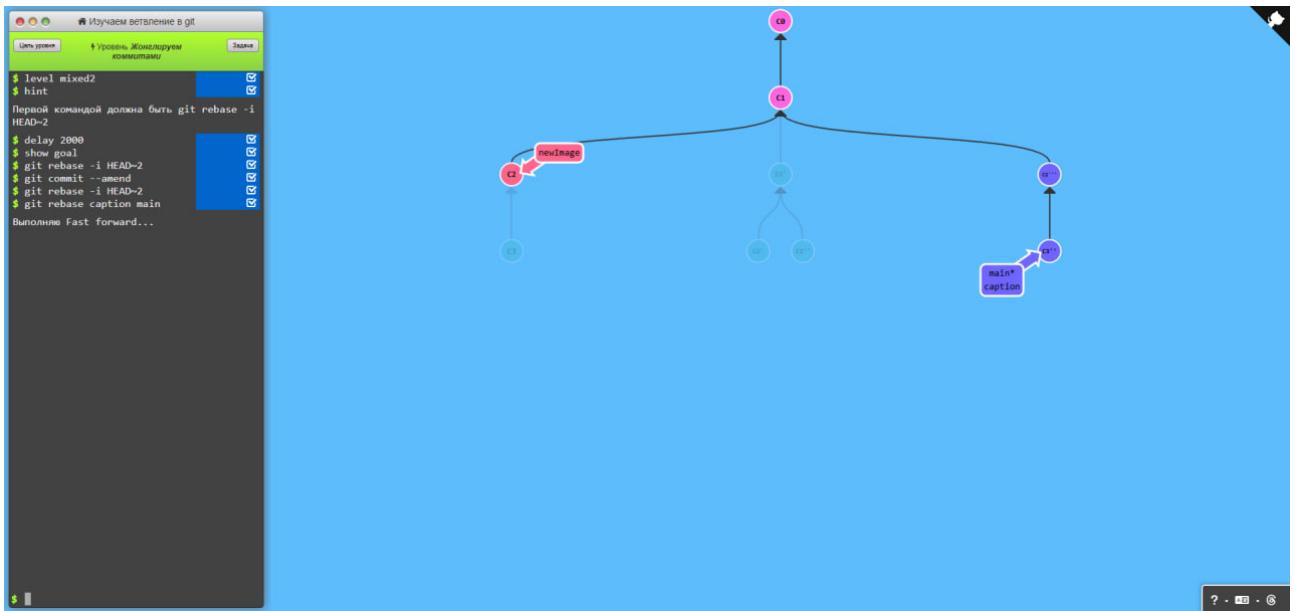


Рис. 3.13

Результат: создаётся нужная ветка на базе старого коммита с новым изменением.

Git Tags

Цель: познакомиться с тегами в Git и их созданием.

Тег (tag) — это постоянная метка на конкретном коммите. Используется для пометки релизов, важных версий и контрольных точек.

Команда:

```
1 git tag v1 C1
```

фиг. 3.14 представлен скриншот данного задания.

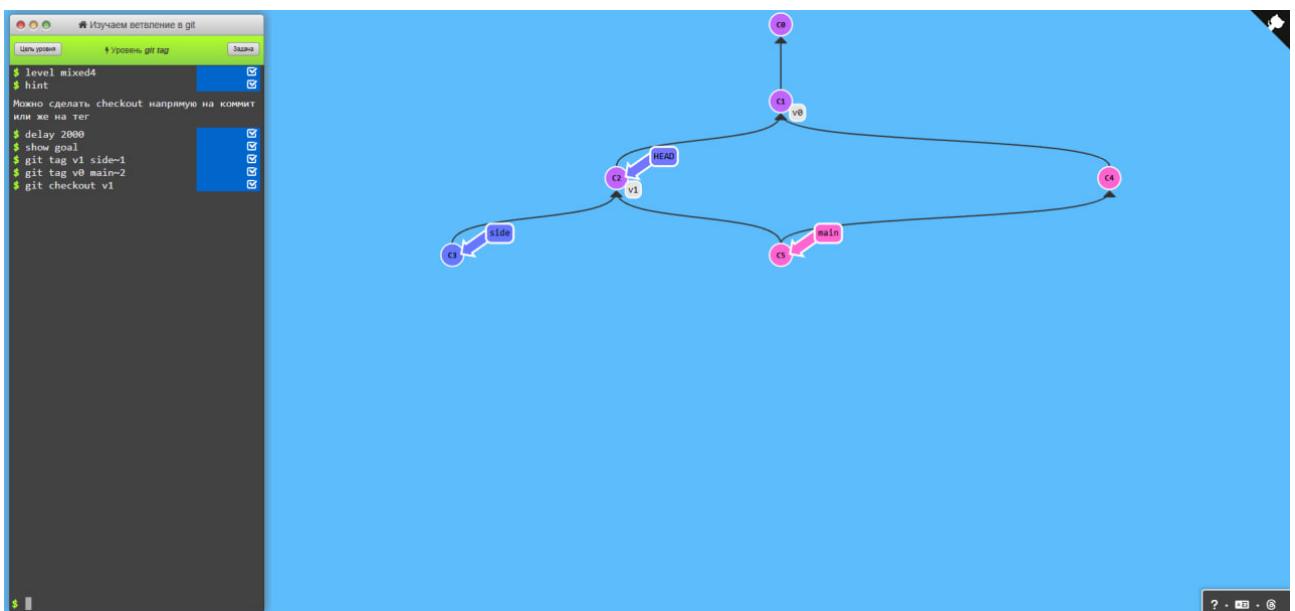


Рис. 3.14

Примечание: в отличие от ветки, тег не перемещается при новых коммитах.

Git Describe

Цель: научиться использовать `git describe` для понимания положения HEAD относительно тегов.

`git describe` показывает ближайший тег и расстояние (число коммитов) от него до текущего состояния ветки.

Команда:

```
1 git describe
```

Пример вывода:

```
1 v1-2-gabcdef
```

фиг. 3.15 представлен скриншот данного задания.

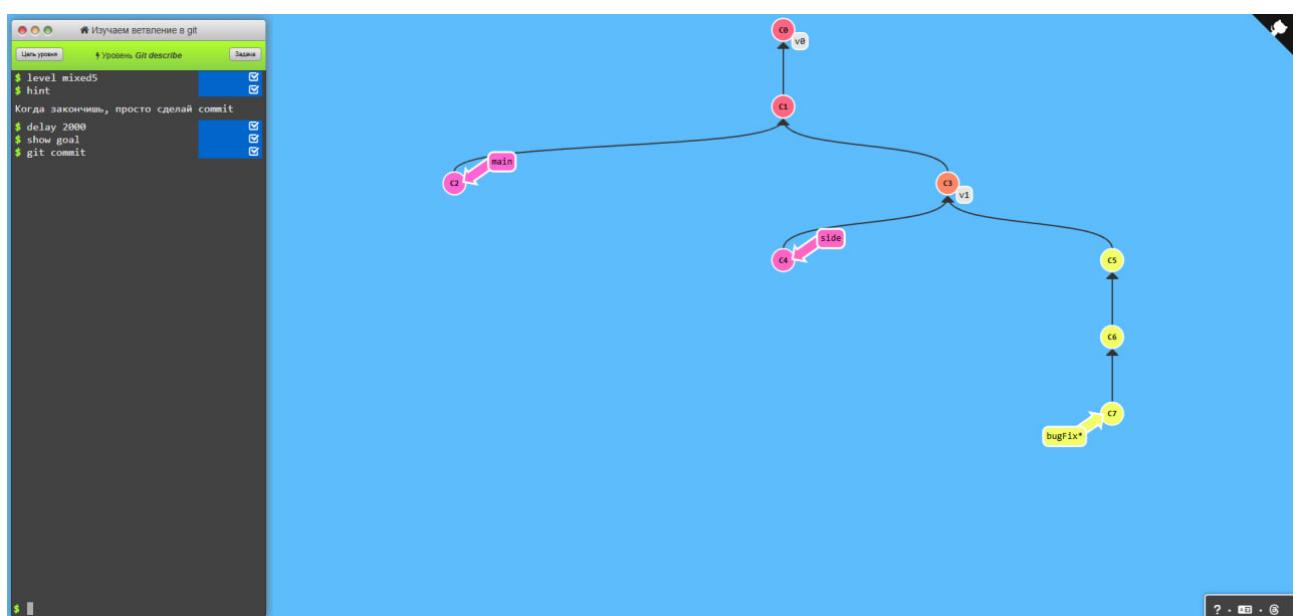


Рис. 3.15

`v1` — ближайший тег, `2` — количество коммитов после тега, `gabcdef` — hash текущего коммита.

3.1.5 Advanced Topics

Rebasing over 9000 times

Цель: повторно закрепить использование `git rebase` и научиться выстраивать линейную историю при множественных ветках.

При сложной структуре ветвлений и параллельных разработках часто необходимо "переносить" изменения с ветки на ветку, чтобы сохранить читаемость истории и избежать конфликтов при слиянии.

Команды:

```
1 git rebase main
```

фиг. 3.16 представлен скриншот данного задания.

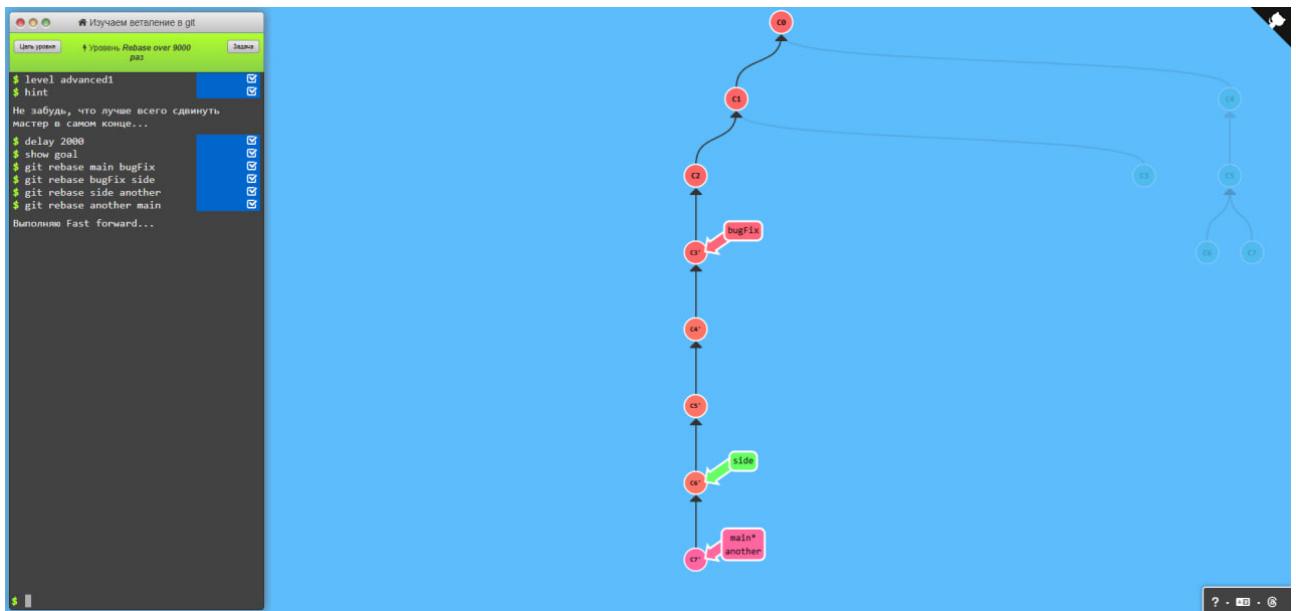


Рис. 3.16

Примечание: важно выполнять rebase с осознанием того, что вы меняете историю коммитов. Не используйте на общедоступных ветках.

Multiple Parents

Цель: изучить коммиты с несколькими родителями (merge-коммиты).

Git позволяет объединять несколько веток, создавая коммит с двумя и более родителями. Это удобно, но влечёт за собой визуальную сложность истории.

Команда:

```
1 git merge feature1
```

фиг. 3.17 представлен скриншот данного задания.

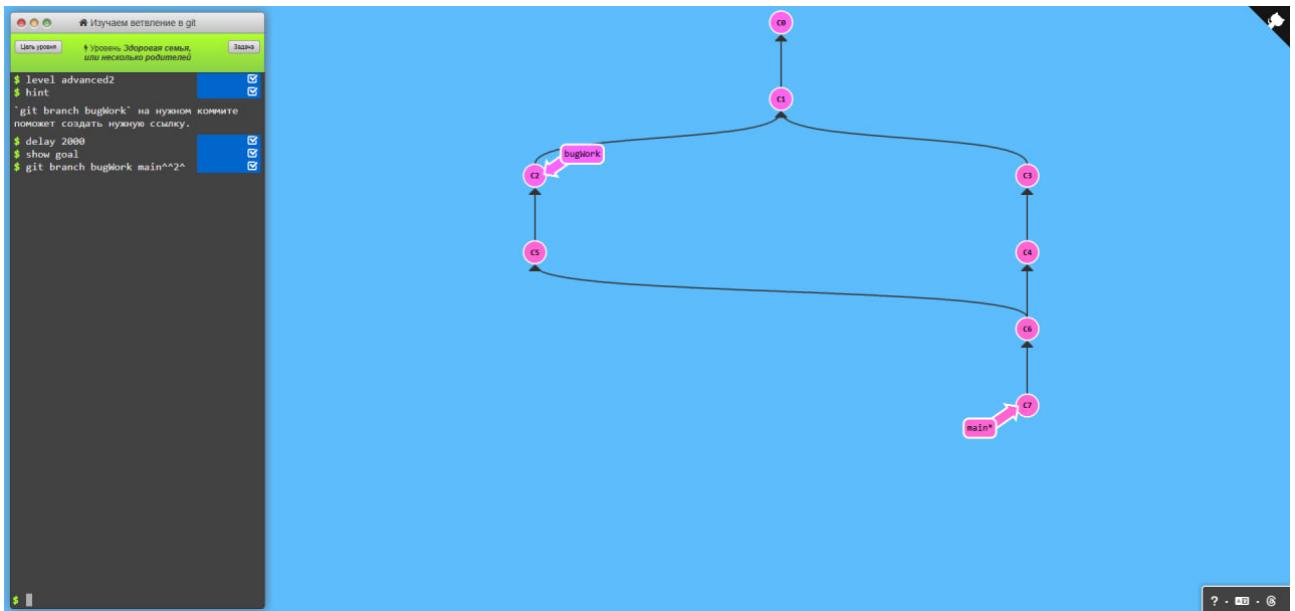


Рис. 3.17

Пример: если обе ветки имеют уникальные коммиты, результатом будет коммит с двумя родителями — $C4 = \text{merge}(C2, C3)$.

Branch Spaghetti

Цель: визуализировать и понять последствия хаотичной работы с ветками.

Когда разработчики ветвятся без правил и контроля, история превращается в "спагетти" — множество пересечений, нестабильность и путаница.

Цель уровня — выявить, как избежать этой ситуации и как можно аккуратно провести слияния или rebase для восстановления структуры.

Рекомендации:

- Используйте feature-ветки.
- Делайте squash перед merge.
- Поддерживайте линейную историю на основной ветке.

фиг. 3.18 представлен скриншот данного задания.

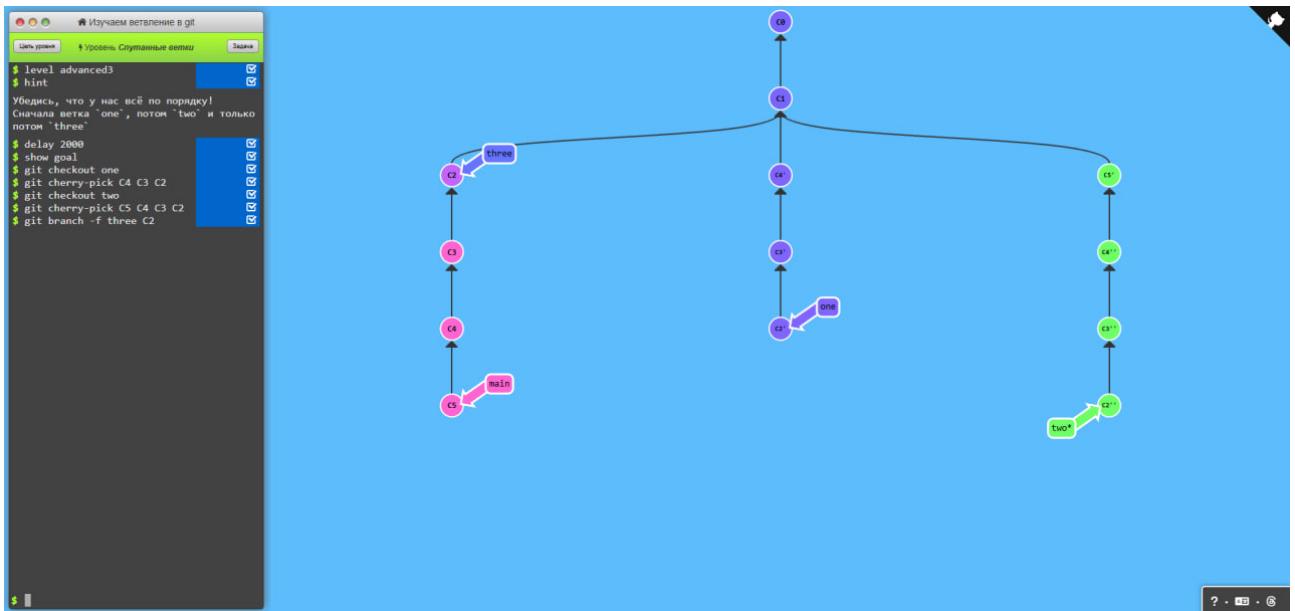


Рис. 3.18

3.2 Remote — Работа с удалёнными репозиториями

3.2.1 Push & Pull — Git Remotes

Clone Intro

Цель: понять суть удалённого репозитория и создать его копию локально.

Команда `git clone` создаёт локальную копию удалённого репозитория. В рамках LearnGitBranching она используется для создания виртуального удалённого репозитория и отображения его структуры.

Команда:

1 `git clone`

фиг. 3.19 представлен скриншот данного задания.

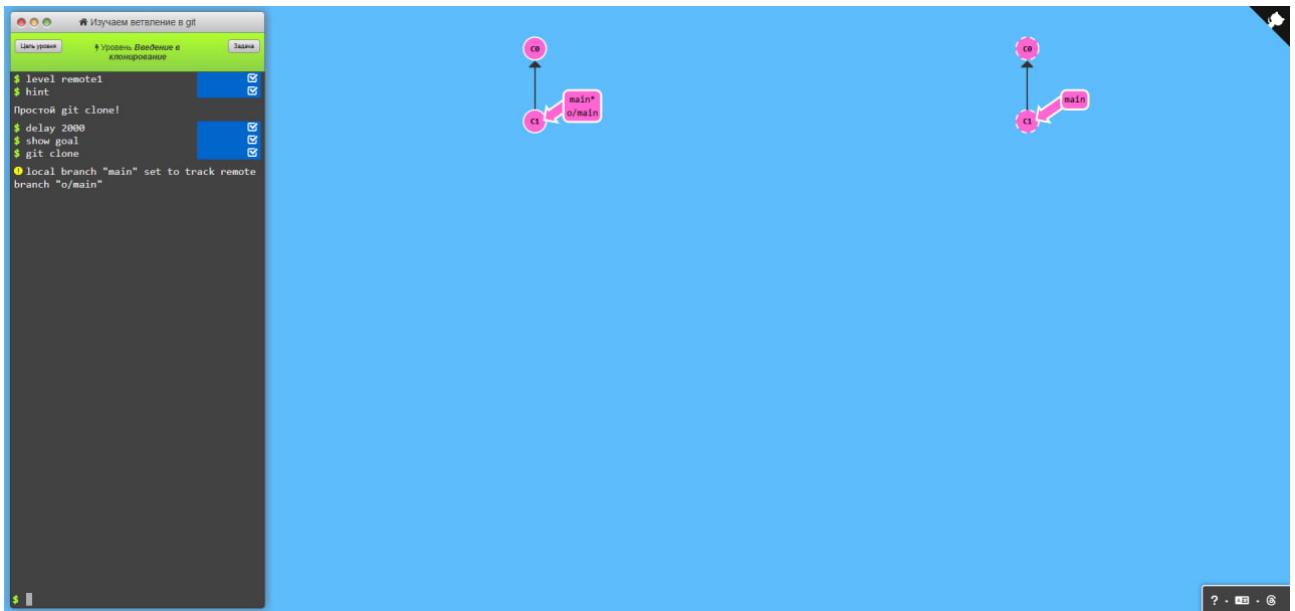


Рис. 3.19

Результат: появляется удалённый репозиторий (remote), связанный с текущим.

Remote Branches

Цель: изучить поведение удалённых веток.

Удалённые ветки (например, `o/main`) отражают состояние удалённого репозитория. Они обновляются после `fetch/pull` и не могут быть изменены напрямую.

Пример поведения:

```
1 git checkout o/main
2 git commit
```

фиг. 3.20 представлен скриншот данного задания.

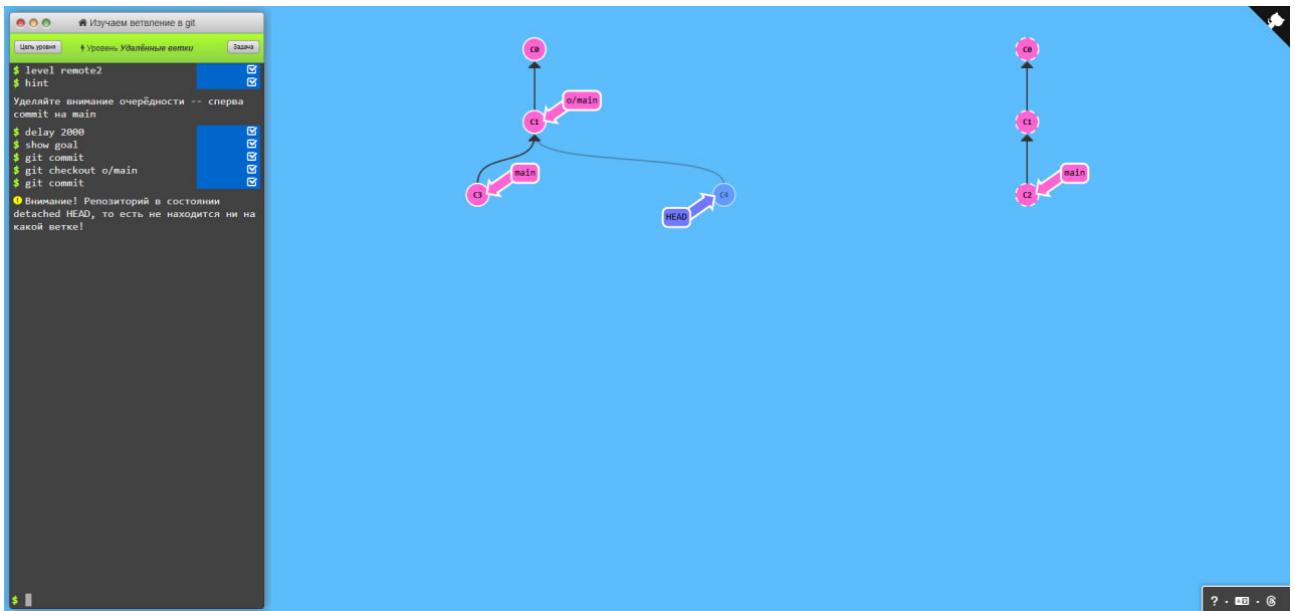


Рис. 3.20

Результат: создаётся коммит, но он не привязывается к ветке `o/main`, так как `HEAD` отсоединен.

Git Fetchin'

Цель: получить изменения с удалённого репозитория без их применения.

`git fetch` загружает изменения и обновляет удалённые ветки, но не изменяет локальные.

Команда:

```
1 git fetch
```

Поведение:

- загружает недостающие коммиты;
- обновляет `o/main` (и др.);
- не влияет на ваш рабочий каталог.

фиг. 3.21 представлен скриншот данного задания.

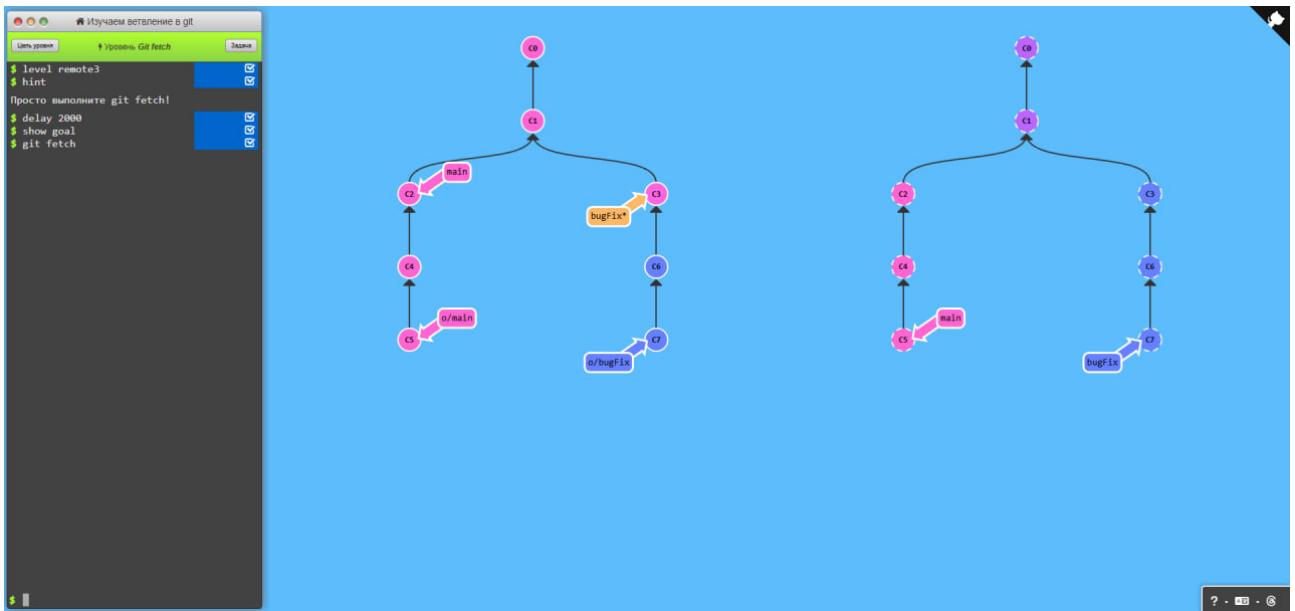


Рис. 3.21

Git Pullin'

Цель: объединить fetch и merge в одну команду.

`git pull` = `git fetch` + `git merge`. Используется для актуализации локальной ветки с удалённой.

Команды:

```
1 git pull
2 git fetch; git merge o/main
```

фиг. 3.22 представлен скриншот данного задания.

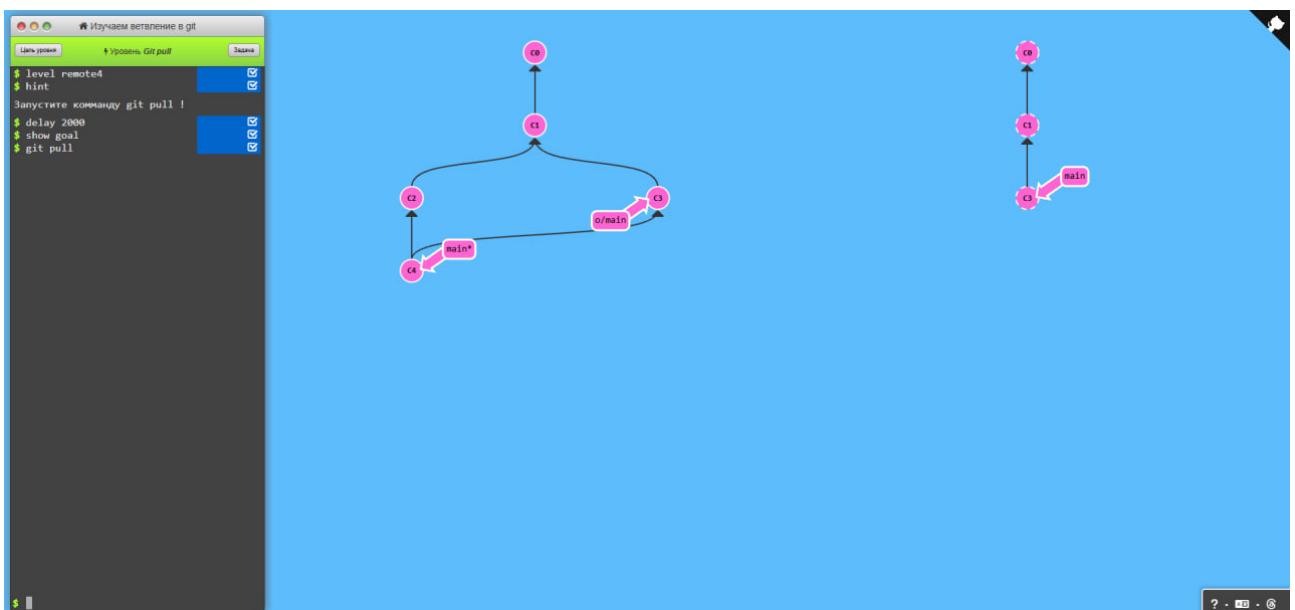


Рис. 3.22

Результат: локальная ветка будет дополнена новыми коммитами из `o/main`.

Faking Teamwork

Цель: смоделировать параллельную работу нескольких разработчиков.

В этом упражнении мы тренируемся в синхронизации изменений между локальной и удалённой ветками. Один разработчик делает коммит локально, другой — на удалённом.

Команды:

```
1 git commit          # локальный коммит
2 git fetch           # подтянуть удалённый
```

фиг. 3.23 представлен скриншот данного задания.

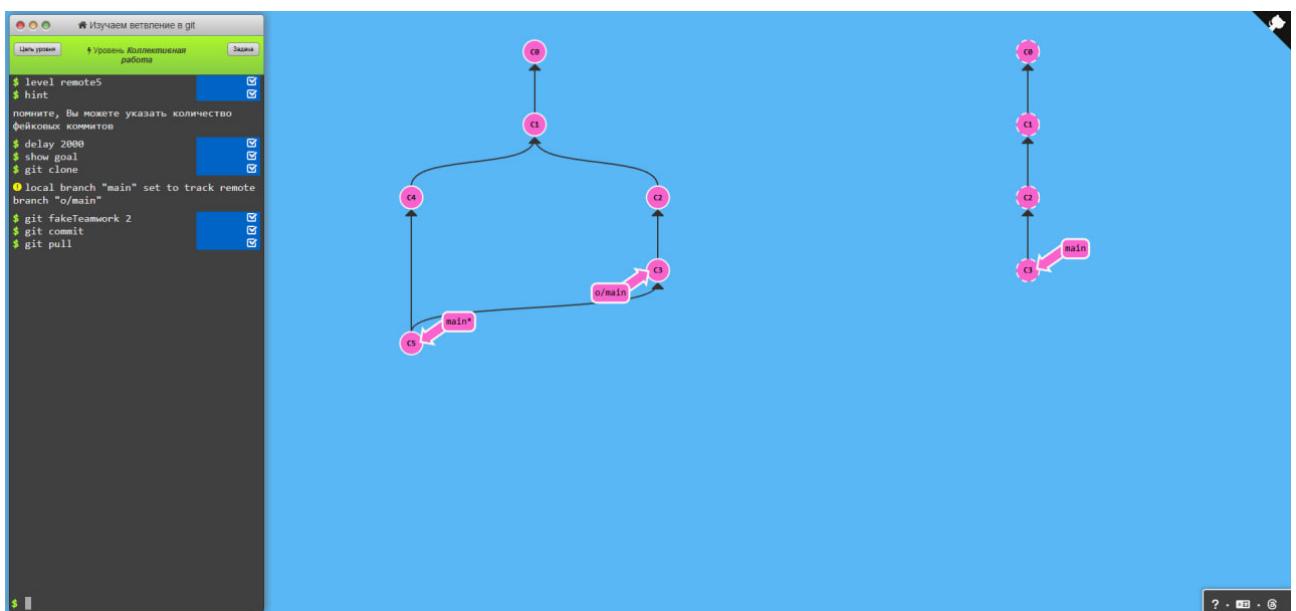


Рис. 3.23

Задача: объединить обе версии через merge или rebase.

Git Pushin'

Цель: передать изменения в удалённый репозиторий.

`git push` отправляет ваши локальные коммиты в удалённую ветку. Только если история не расходится.

Команда:

```
1 git push
```

фиг. 3.24 представлен скриншот данного задания.

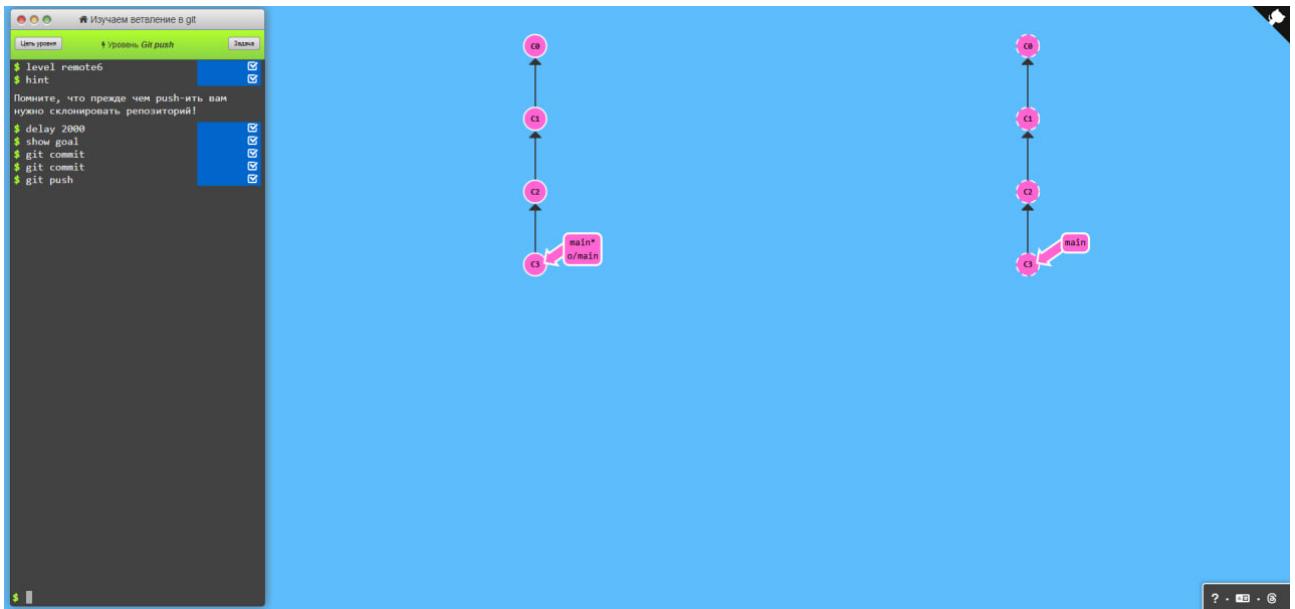


Рис. 3.24

Важно: если история различается, push будет отклонён.

Diverged History

Цель: разрешить конфликты при расхождении истории.

Когда локальная и удалённая ветки имеют разные изменения, Git требует ручного вмешательства. Нужно выполнить:

Команды:

```
1 git pull --rebase      # или git fetch + rebase
```

После: можно безопасно выполнить git push.

фиг. 3.25 представлен скриншот данного задания.

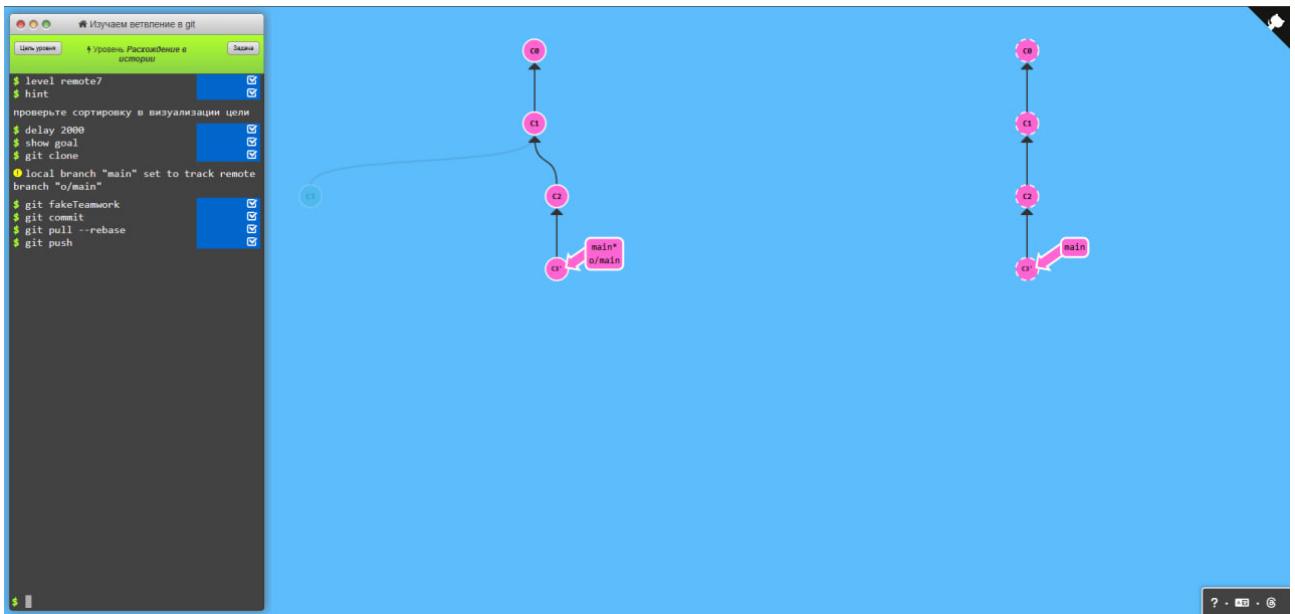


Рис. 3.25

Locked Main

Цель: изучить ситуацию, когда push запрещён без обновления локальной истории.

Некоторые удалённые репозитории (например, GitHub) запрещают push, если локальная ветка не включает последние изменения.

Решение:

1 git pull --rebase

фиг. 3.26 представлен скриншот данного задания.

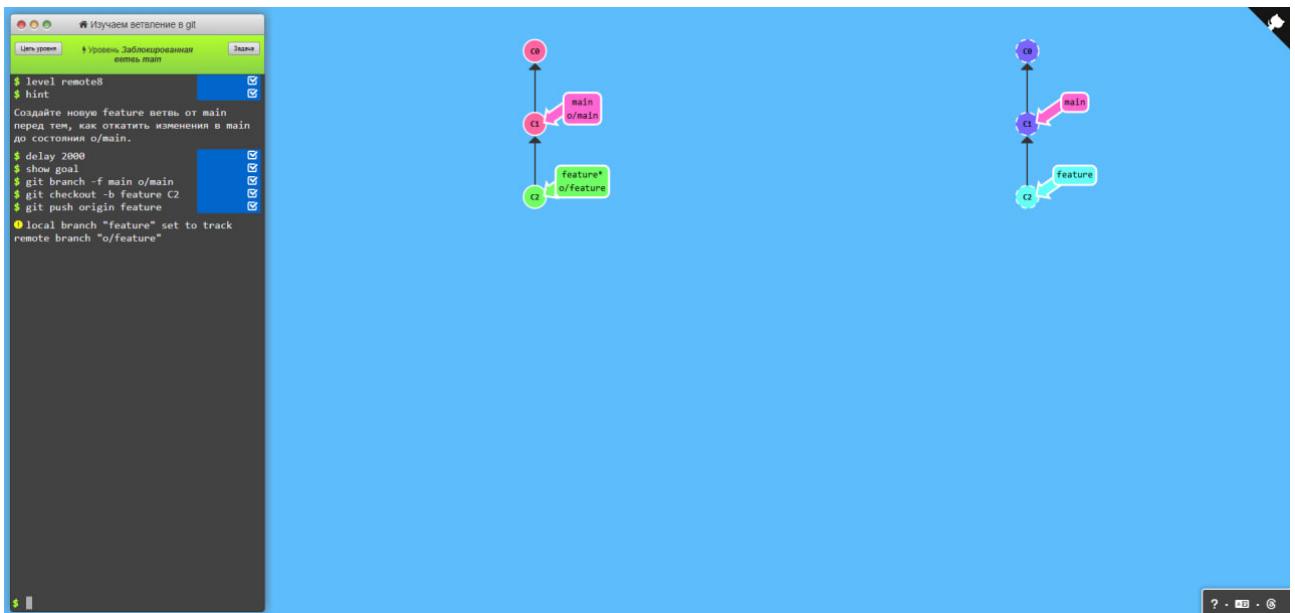


Рис. 3.26

Результат: локальная ветка обновляется и вы можете push без ошибок.

3.2.2 To Origin and Beyond — Advanced Git Remotes

Push Main!

Цель: закрепить основную команду для публикации ветки.

Задание фокусируется на использовании `git push` для основной ветки (`main`).

Команда:

```
1 git push origin main
```

фиг. 3.27 представлен скриншот данного задания.

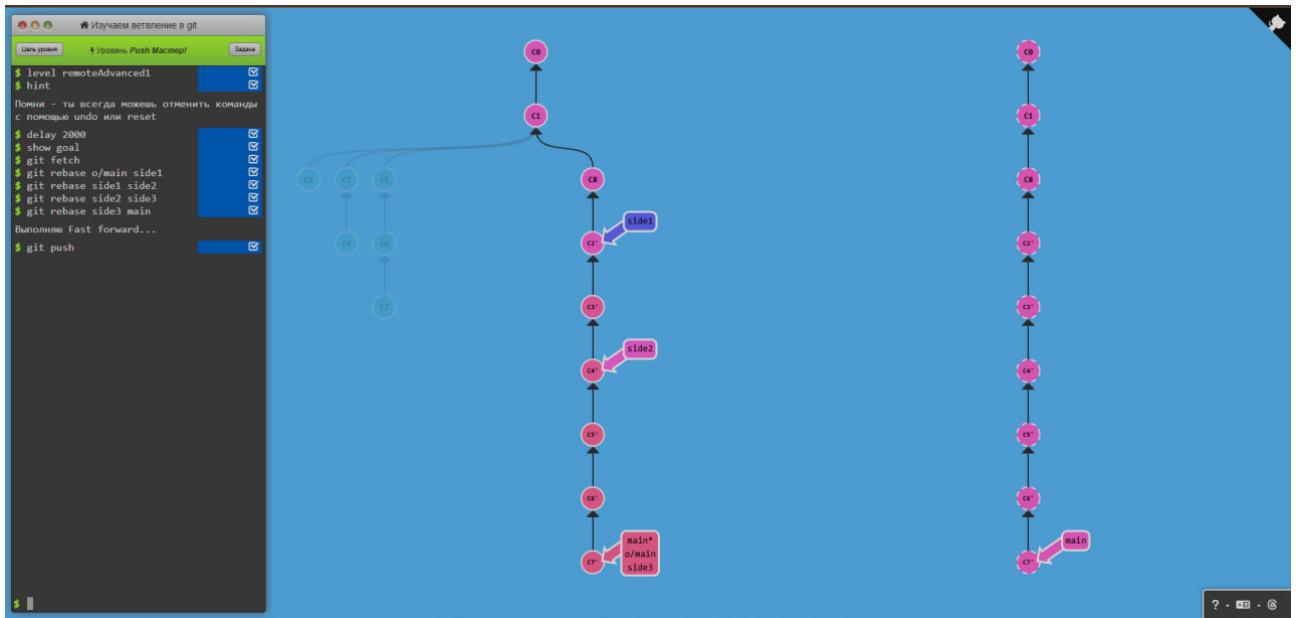


Рис. 3.27

Примечание: при явном указании ветки повышается контроль.

Merging with remotes

Цель: объединить удалённые изменения с локальными.

В случае, когда на удалённой стороне появились коммиты, которых нет у вас, нужно сначала их подтянуть:

```
1 git pull --rebase
```

Затем можно безопасно делать:

```
1 git push
```

фиг. 3.28 представлен скриншот данного задания.

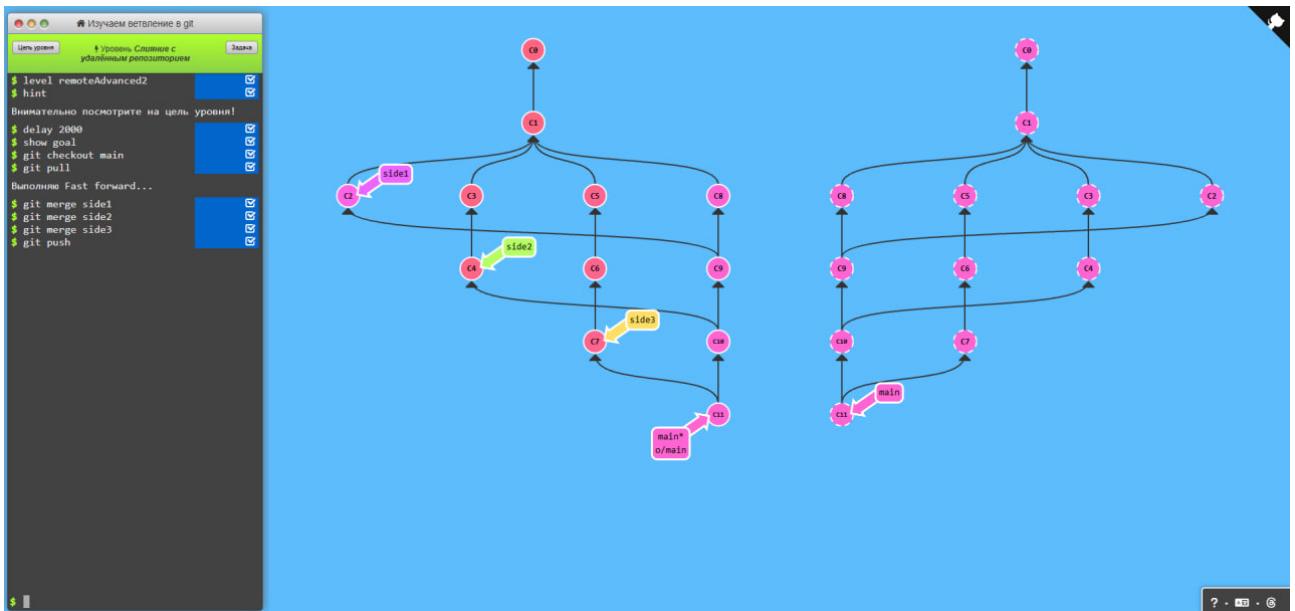


Рис. 3.28

Remote Tracking

Цель: понять, как ветки отслеживают друг друга.

При клонировании создаются локальные ветки, "отслеживающие" удалённые:

1 `git branch -vv`

фиг. 3.29 представлен скриншот данного задания.

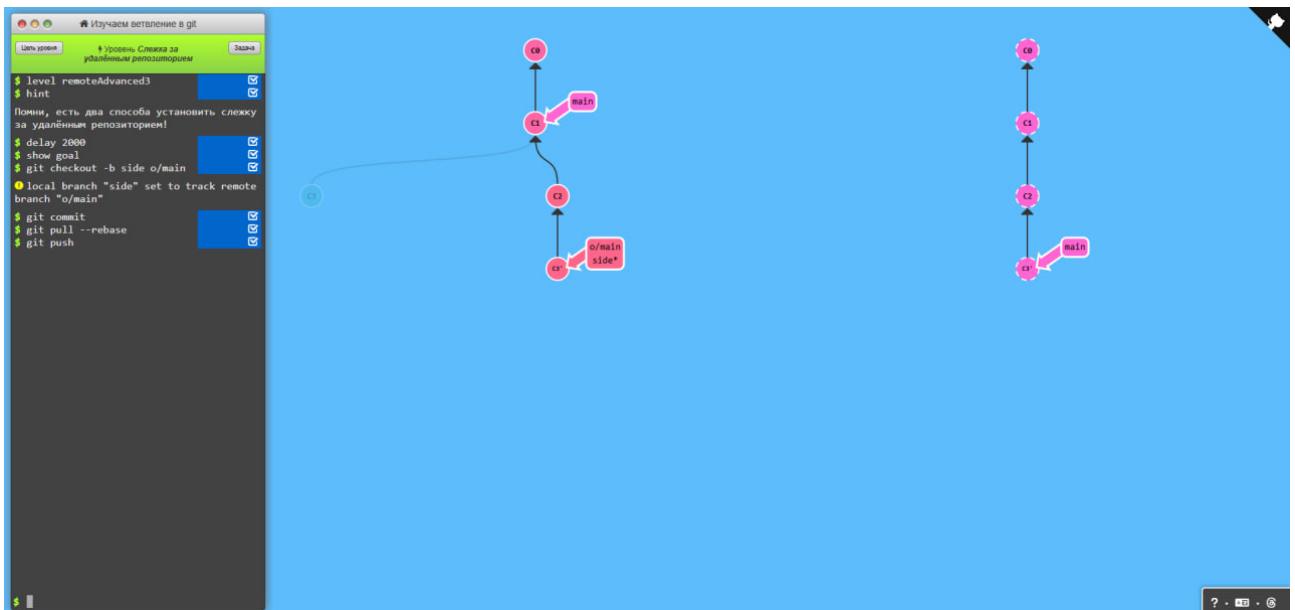


Рис. 3.29

Вывод: вы увидите, какие ветки связаны с удалёнными и на сколько коммитов они отстают или опережают.

Git push arguments

Цель: научиться задавать явно, что и куда отправляется.

Формат:

```
1 git push <remote> <source>:<destination>
```

Пример:

```
1 git push origin bugFix:main
```

Это отправит ветку `bugFix` в удалённую `main`.

фиг. 3.30 представлен скриншот данного задания.

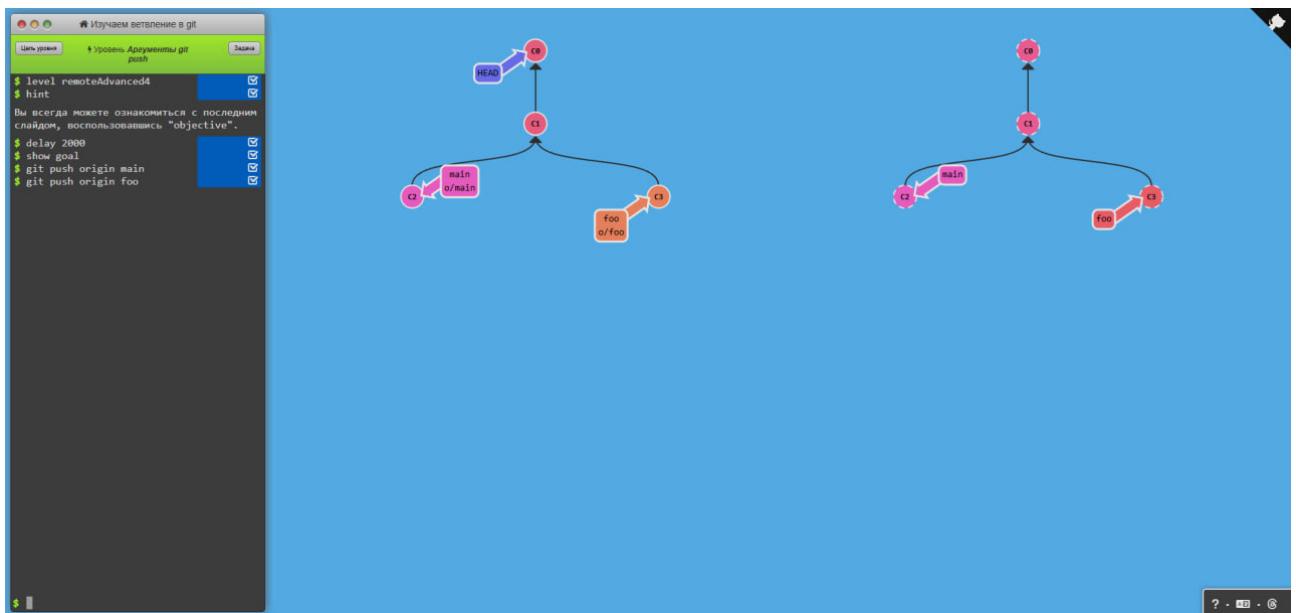


Рис. 3.30

Git push arguments – Expanded!

Цель: управлять историей удалённого репозитория.

Удаление ветки с сервера:

```
1 git push origin :feature1
```

фиг. 3.31 представлен скриншот данного задания.

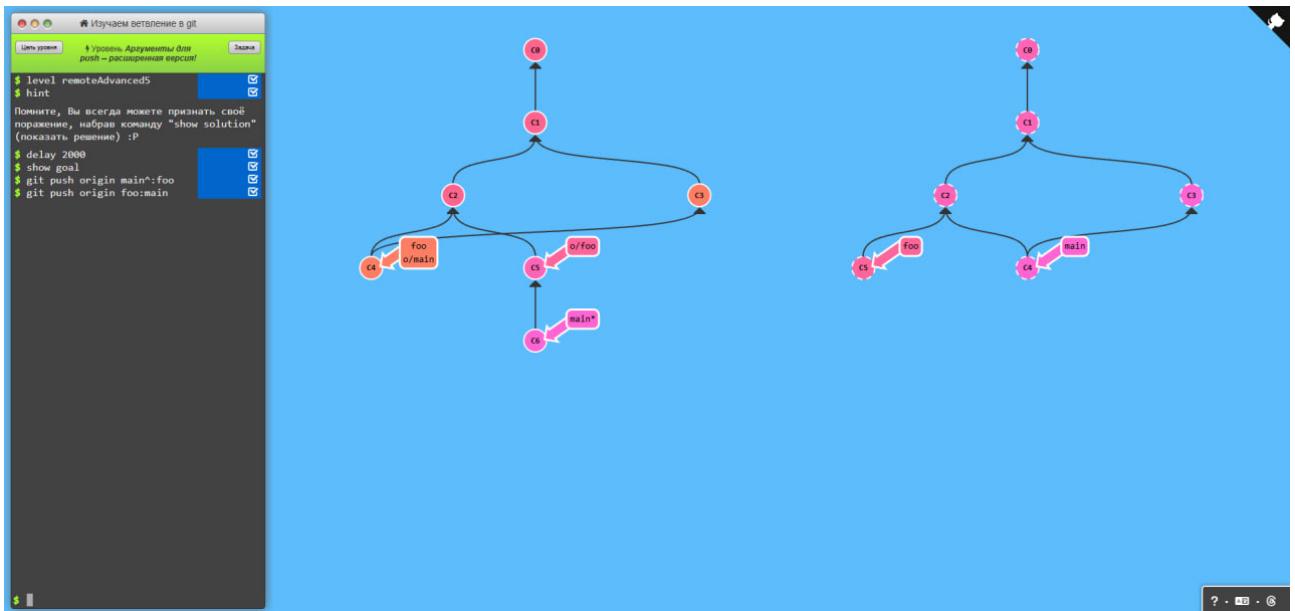


Рис. 3.31

Результат: удалённая ветка `feature1` будет удалена.

Fetch arguments

Цель: частично обновлять удалённые данные.

Можно получать не все изменения, а только конкретные ветки:

1 `git fetch origin bugFix`

фиг. 3.32 представлен скриншот данного задания.

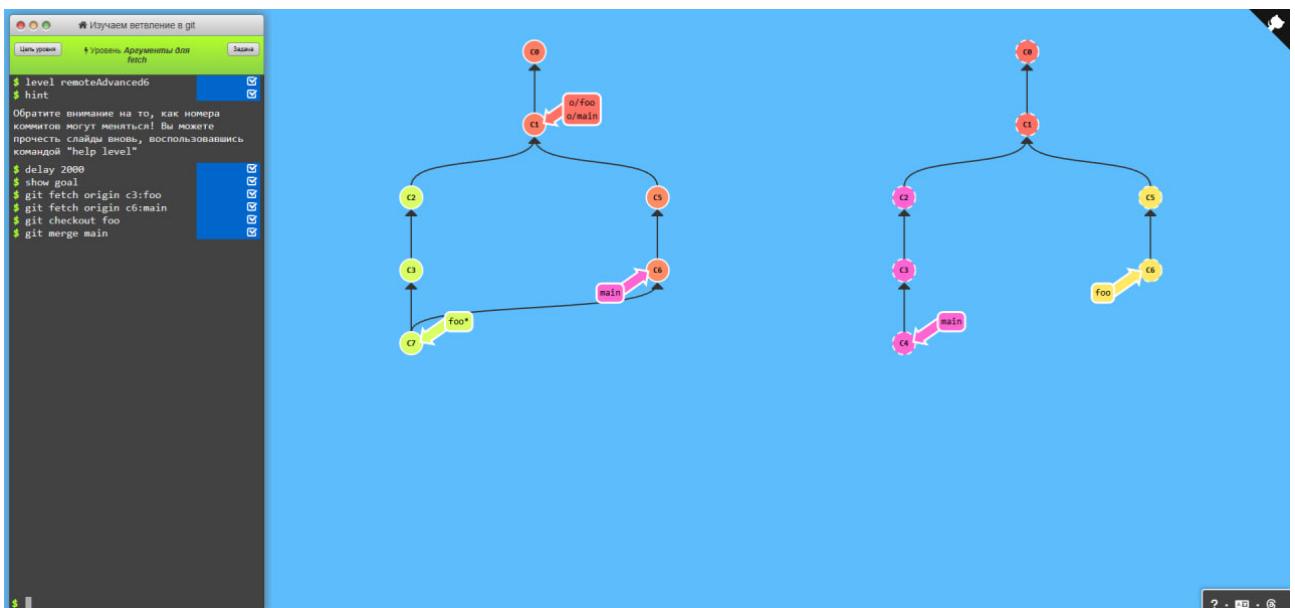


Рис. 3.32

Source of nothing

Цель: проанализировать ситуацию, когда вы клонируете пустой репозиторий.

После `git clone` не будет ни одного коммита или ветки. Вы должны создать начальный коммит вручную:

```
1 git commit
```

фиг. 3.33 представлен скриншот данного задания.



Рис. 3.33

Pull arguments

Цель: точно управлять направлением слияния.

Можно явно указать, откуда и куда тянуть изменения:

```
1 git pull origin main
```

Это тянет ветку `main` с `origin` в текущую локальную ветку.

фиг. 3.34 представлен скриншот данного задания.

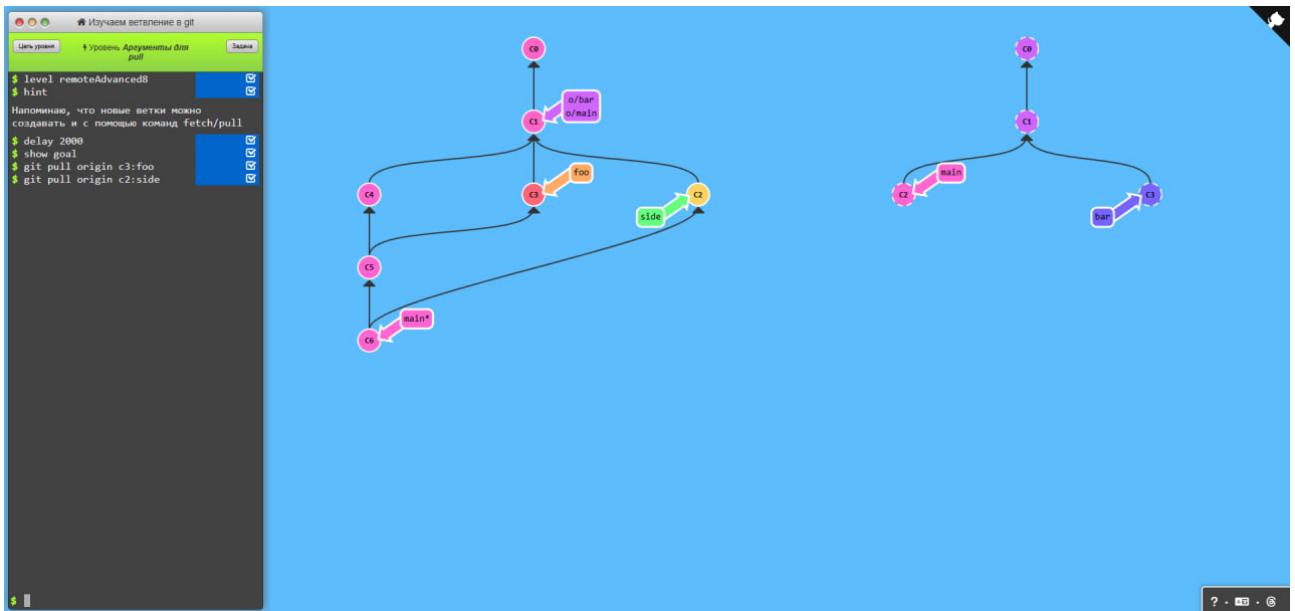


Рис. 3.34

Заключение: продвинутое взаимодействие с удалёнными репозиториями требует понимания того, что и куда передаётся. Явные аргументы повышают контроль и снижают ошибки.

3.3 Выводы для главы 3

В третьей главе была реализована практическая часть изучения Git на платформе Learn Git Branching [1]. Пользователь прошёл последовательно все основные модули: от базового ветвления и перемещения коммитов до работы с удалёнными репозиториями. Каждое задание было проанализировано и снабжено скриншотом, отражающим прогресс и логику решений.

Практика показала, что визуальный подход к обучению Git значительно ускоряет понимание концепций и развивает мышление, ориентированное на управление историями изменений. Итогом главы стало закрепление всех ключевых команд и принципов Git в интерактивной форме, что подтверждает успешное освоение материала и готовность применять знания в реальных проектах.

IV Практика с Git: Git Immersion (Задания 1–33)

Задание 1: Setup

```
danielmihai@MacBook-Air-Danik thesis % git config --global user.name "Daniel"
danielmihai@MacBook-Air-Danik thesis % git config --global user.email "danielmihai.it@mail.ru"
danielmihai@MacBook-Air-Danik thesis % git config --global core.autocrlf input
danielmihai@MacBook-Air-Danik thesis % git config --global core.safecrlf true
danielmihai@MacBook-Air-Danik thesis %
```

Рис. 4.1. Скриншот выполнения задания 1

Описание: Настроены имя и email пользователя в git. Установлены предпочтения по окончаниям строк в зависимости от ОС.

```
1 git config --global user.name "Your Name"
2 git config --global user.email "your_email@whatever.com"
3 git config --global core.autocrlf input
4 git config --global core.safecrlf true
```

Задание 2: More Setup

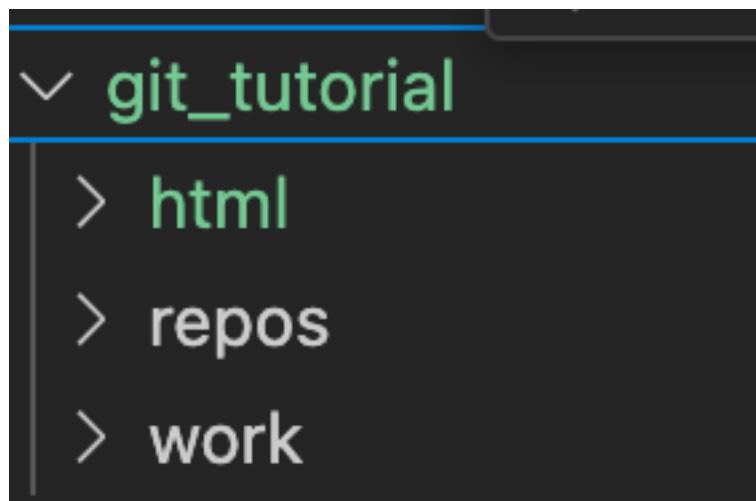


Рис. 4.2. Скриншот выполнения задания 2

Описание: Скачен и распакован учебный архив с сайта Git Immersion. Созданы директории для работы.

```
1 # manual download and unzip
```

Задание 3: Create a Project

```
● danielmihai@MacBook-Air-Danik uni_thesisTemplate % mkdir hello
● danielmihai@MacBook-Air-Danik uni_thesisTemplate % cd hello
● danielmihai@MacBook-Air-Danik hello % git init
  Инициализирован пустой репозиторий Git в /Users/danielmihai/Documents/code/uni_thesisTemplate/hello/.git/
● danielmihai@MacBook-Air-Danik hello % git remote add origin https://github.com/messsimo/uni_practice.git
✖ danielmihai@MacBook-Air-Danik hello % git add hello.rb
  fatal: спецификатор пути «hello.rb» не соответствует ни одному файлу
● danielmihai@MacBook-Air-Danik hello % git add hello.rb
● danielmihai@MacBook-Air-Danik hello % git commit -m "First Commit"
  [main (корневой коммит) 2cb1af1] First Commit
    1 file changed, 0 insertions(+), 0 deletions(-)
    create mode 100644 hello.rb
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.3. Скриншот выполнения задания 3

Описание: Создана директория hello, написан hello.rb, инициализирован git-репозиторий, сделан первый коммит.

```
1  mkdir hello
2  cd hello
3  git init
4  git add hello.rb
5  git commit -m "First Commit"
```

Задание 4: Checking Status

```
● danielmihai@MacBook-Air-Danik hello % git status
  Текущая ветка: main
  нечего коммитить, нет изменений в рабочем каталоге
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.4. Скриншот выполнения задания 4

Описание: Проверен статус репозитория, убедились в отсутствии несохранённых изменений.

```
1  git status
```

Задание 5: Making Changes

```
● danielmihai@MacBook-Air-Danik hello % git status
  Текущая ветка: main
  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)
    изменено:      hello.rb

  индекс пуст (используйте «git add» и/или «git commit -a»)
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.5. Скриншот выполнения задания 5

Описание: Изменён файл hello.rb с использованием ARGV. Просмотрено состояние перед индексацией.

```
1 git status
```

Задание 6: Staging Changes

```
● danielmihai@MacBook-Air-Danik hello % git add hello.rb
● danielmihai@MacBook-Air-Danik hello % git status
Текущая ветка: main
Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
    изменено:    hello.rb

○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.6. Скриншот выполнения задания 6

Описание: Изменения подготовлены к коммиту с помощью git add.

```
1 git add hello.rb
2 git status
```

Задание 7: Staging and Committing

```
● danielmihai@MacBook-Air-Danik hello % git add a.rb
● danielmihai@MacBook-Air-Danik hello % git add b.rb
● danielmihai@MacBook-Air-Danik hello % git commit -m "Changes for a and b"
[main 9650de5] Changes for a and b
  3 files changed, 1 insertion(+)
  create mode 100644 a.rb
  create mode 100644 b.rb
● danielmihai@MacBook-Air-Danik hello % git add c.rb
● danielmihai@MacBook-Air-Danik hello % git commit -m "Unrelated change to c"
[main 61ac871] Unrelated change to c
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 c.rb
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.7. Скриншот выполнения задания 7

Описание: Описан процесс поэтапного добавления и коммита нескольких файлов.

```
1 git add a.rb
2 git add b.rb
3 git commit -m "Changes for a and b"
4 git add c.rb
5 git commit -m "Unrelated change to c"
```

Задание 8: Committing Changes

```
④ danielmihai@MacBook-Air-Danik hello % git commit
  Текущая ветка: main
  нечего коммитить, нет изменений в рабочем каталоге
● danielmihai@MacBook-Air-Danik hello % git status
  Текущая ветка: main
  нечего коммитить, нет изменений в рабочем каталоге
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.8. Скриншот выполнения задания 8

Описание: Выполнен коммит с использованием редактора без флага -m.

```
1 git commit
2 git status
```

Задание 9: Changes, not Files

```
● danielmihai@MacBook-Air-Danik hello % git add hello.rb
● danielmihai@MacBook-Air-Danik hello % git status
  Текущая ветка: main
  Изменения, которые будут включены в коммит:
    (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
    изменено:      hello.rb

  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)
    изменено:      hello.rb

● danielmihai@MacBook-Air-Danik hello % git commit -m "Added a default value"
[main ca67c98] Added a default value
  1 file changed, 3 insertions(+), 1 deletion(-)
● danielmihai@MacBook-Air-Danik hello % git status
  Текущая ветка: main
  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)
    изменено:      hello.rb

  индекс пуст (используйте «git add» и/или «git commit -a»)
● danielmihai@MacBook-Air-Danik hello % git add .
● danielmihai@MacBook-Air-Danik hello % git status
  Текущая ветка: main
  Изменения, которые будут включены в коммит:
    (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
    изменено:      hello.rb

● danielmihai@MacBook-Air-Danik hello % git commit -m "Added a comment"
[main 3ff54eb] Added a comment
  1 file changed, 1 insertion(+)
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.9. Скриншот выполнения задания 9

Описание: Стадия и коммит изменений показаны на примере добавления значения по умолчанию и комментария.

```
1 git add hello.rb
2 git commit -m "Added a default value"
3 git add .
4 git commit -m "Added a comment"
```

Задание 10: History

```
● danielmihai@MacBook-Air-Danik hello % git log
commit 3ff54eb9ccb383ff806bc8a3c35e83b20fce4b16 (HEAD -> main)
Author: Daniel <danielmihai.it@mail.ru>
Date:   Fri Jun 20 08:26:14 2025 +0300

    Added a comment

commit ca67c9808d34109a7ff99faa25c4e54be118d1f6
Author: Daniel <danielmihai.it@mail.ru>
Date:   Fri Jun 20 08:25:53 2025 +0300

    Added a default value

commit 61ac8710753719696d6d305da342655675a0b653
Author: Daniel <danielmihai.it@mail.ru>
Date:   Fri Jun 20 08:21:39 2025 +0300

    Unrelated change to c

commit 9650de5f48fb6ce9f0c4f7037d66bf9319e53591
Author: Daniel <danielmihai.it@mail.ru>
Date:   Fri Jun 20 08:21:26 2025 +0300

    Changes for a and b

commit 2cb1af191297010333ea8c242305c75f7972bcfd
Author: Daniel <danielmihai.it@mail.ru>
Date:   Fri Jun 20 08:18:42 2025 +0300

    First Commit
● danielmihai@MacBook-Air-Danik hello % git log --pretty=oneline
3ff54eb9ccb383ff806bc8a3c35e83b20fce4b16 (HEAD -> main) Added a comment
ca67c9808d34109a7ff99faa25c4e54be118d1f6 Added a default value
61ac8710753719696d6d305da342655675a0b653 Unrelated change to c
9650de5f48fb6ce9f0c4f7037d66bf9319e53591 Changes for a and b
2cb1af191297010333ea8c242305c75f7972bcfd First Commit
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.10. Вывод git log

```

● danielmihai@MacBook-Air-Danik hello % git log --pretty=oneline --max-count=2
3ff54eb9ccb383ff806bc8a3c35e83b20fce4b16 (HEAD -> main) Added a comment
ca67c9808d34109a7ff99faa25c4e54be118d1f6 Added a default value
● danielmihai@MacBook-Air-Danik hello % git log --pretty=oneline --since='5 minutes ago'
3ff54eb9ccb383ff806bc8a3c35e83b20fce4b16 (HEAD -> main) Added a comment
ca67c9808d34109a7ff99faa25c4e54be118d1f6 Added a default value
● danielmihai@MacBook-Air-Danik hello % git log --pretty=oneline --until='5 minutes ago'
61ac8710753719696d6d305da342655675a0b653 Unrelated change to c
9650de5f48fb6ce9f0c4f7037d66bf9319e53591 Changes for a and b
2cb1af191297010333ea8c242305c75f7972bcfd First Commit
○ danielmihai@MacBook-Air-Danik hello % git log --pretty=oneline --author=<Daniel>
zsh: parse error near `\'n'
● danielmihai@MacBook-Air-Danik hello % git log --pretty=oneline --author="Daniel"
3ff54eb9ccb383ff806bc8a3c35e83b20fce4b16 (HEAD -> main) Added a comment
ca67c9808d34109a7ff99faa25c4e54be118d1f6 Added a default value
61ac8710753719696d6d305da342655675a0b653 Unrelated change to c
9650de5f48fb6ce9f0c4f7037d66bf9319e53591 Changes for a and b
2cb1af191297010333ea8c242305c75f7972bcfd First Commit
● danielmihai@MacBook-Air-Danik hello % git log --pretty=oneline --all
3ff54eb9ccb383ff806bc8a3c35e83b20fce4b16 (HEAD -> main) Added a comment
ca67c9808d34109a7ff99faa25c4e54be118d1f6 Added a default value
61ac8710753719696d6d305da342655675a0b653 Unrelated change to c
9650de5f48fb6ce9f0c4f7037d66bf9319e53591 Changes for a and b
2cb1af191297010333ea8c242305c75f7972bcfd First Commit
● danielmihai@MacBook-Air-Danik hello % git log --all --pretty=format:'%h %cd %s (%an)' --since='7 days ago'
3ff54eb Fri Jun 20 08:26:14 2025 +0300 Added a comment (Daniel)
ca67c98 Fri Jun 20 08:25:53 2025 +0300 Added a default value (Daniel)
61ac871 Fri Jun 20 08:21:39 2025 +0300 Unrelated change to c (Daniel)
9650de5 Fri Jun 20 08:21:26 2025 +0300 Changes for a and b (Daniel)
2cb1af1 Fri Jun 20 08:18:42 2025 +0300 First Commit (Daniel)
● danielmihai@MacBook-Air-Danik hello % git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
* 3ff54eb 2025-06-20 | Added a comment (HEAD -> main) [Daniel]
* ca67c98 2025-06-20 | Added a default value [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello %

```

Рис. 4.11. Форматированный вывод с графиком

Описание: Изучены различные опции git log: сокращённый, с фильтрами, с графиком.

```

1 git log
2 git log --pretty=oneline
3 git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short

```

Задание 11: Aliases

```

● danielmihai@MacBook-Air-Danik hello % alias gs='git status '
● danielmihai@MacBook-Air-Danik hello % alias ga='git add '
● danielmihai@MacBook-Air-Danik hello % alias gb='git branch '
● danielmihai@MacBook-Air-Danik hello % alias gc='git commit'
● danielmihai@MacBook-Air-Danik hello % alias gd='git diff'
● danielmihai@MacBook-Air-Danik hello % alias gco='git checkout '
● danielmihai@MacBook-Air-Danik hello % alias gk='gitk --all&'
● danielmihai@MacBook-Air-Danik hello % alias gx='gitx --all'
● danielmihai@MacBook-Air-Danik hello % alias got='git '
● danielmihai@MacBook-Air-Danik hello % alias get='git '
● danielmihai@MacBook-Air-Danik hello % gco main
  Уже на «main»
○ danielmihai@MacBook-Air-Danik hello %

```

Рис. 4.12. Создание алиасов в .gitconfig

Описание: Добавлены алиасы для часто используемых команд git, включая git hist для форматированного вывода.

```

1 [alias]
2   co = checkout
3   ci = commit
4   st = status
5   br = branch
6   hist = log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
7   type = cat-file -t
8   dump = cat-file -p

```

Задание 12: Getting Old Versions

```

● danielmihai@MacBook-Air-Danik hello % git hist
* 3ff54eb 2025-06-20 | Added a comment (HEAD -> main) [Daniel]
* ca67c98 2025-06-20 | Added a default value [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello % git checkout <hash>
zsh: parse error near `\'n'
☒ danielmihai@MacBook-Air-Danik hello % пше checkout main
zsh: command not found: пше
● danielmihai@MacBook-Air-Danik hello % git checkout main
Уже на «main»
● danielmihai@MacBook-Air-Danik hello % cat hello.rb
# Default is "World"
name = ARGV.first || "World"

puts "Hello, #{name}!"%
● danielmihai@MacBook-Air-Danik hello % git checkout main
Уже на «main»
● danielmihai@MacBook-Air-Danik hello % cat hello.rb
# Default is "World"
name = ARGV.first || "World"

puts "Hello, #{name}!"%
○ danielmihai@MacBook-Air-Danik hello %

```

Рис. 4.13. Просмотр и возврат к старой версии

Описание: Изучена возможность переключения на старые коммиты с помощью checkout по хешу.

```

1 git hist
2 git checkout <hash>
3 cat hello.rb
4 git checkout main

```

Задание 13: Tagging Versions

- danielmihai@MacBook-Air-Danik hello % git tag v1
- danielmihai@MacBook-Air-Danik hello % git checkout v1^
Примечание: переключение на «v1^».

Вы сейчас в состоянии «отсоединённого указателя HEAD». Можете осмотреться, внести экспериментальные изменения и зафиксировать их, также можете отменить любые коммиты, созданные в этом состоянии, не затрагивая другие ветки, переключившись обратно на любую ветку.

Если хотите создать новую ветку для сохранения созданных коммитов, можете сделать это (сейчас или позже), используя команду switch с параметром -c. Например:

```
git switch -c <новая-ветка>
```

Или отмените эту операцию с помощью:

```
git switch -
```

Отключите этот совет, установив переменную конфигурации advice.detachedHead в значение false

HEAD сейчас на ca67c98 Added a default value

- danielmihai@MacBook-Air-Danik hello % cat hello.rb
name = ARGV.first || "World"

puts "Hello, #{name}!"
● danielmihai@MacBook-Air-Danik hello % git tag v1-beta
● danielmihai@MacBook-Air-Danik hello % git checkout v1
Предыдущая позиция HEAD была ca67c98 Added a default value
HEAD сейчас на 3ff54eb Added a comment
● danielmihai@MacBook-Air-Danik hello % git checkout v1-beta
Предыдущая позиция HEAD была 3ff54eb Added a comment
HEAD сейчас на ca67c98 Added a default value
● danielmihai@MacBook-Air-Danik hello % git tag
v1
v1-beta
● danielmihai@MacBook-Air-Danik hello % git hist main --all
* 3ff54eb 2025-06-20 | Added a comment (tag: v1, main) [Daniel]
* ca67c98 2025-06-20 | Added a default value (HEAD, tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello % █

Рис. 4.14. Создание и использование тегов

Описание: Присвоены теги v1 и v1-beta разным коммитам. Проверена навигация по ним.

```
1 git tag v1
2 git checkout v1^
3 git tag v1-beta
4 git checkout v1
5 git tag
6 git hist main --all
```

Задание 14: Undoing Local Changes (before staging)

```
● danielmihai@MacBook-Air-Danik hello % git checkout main
Предыдущая позиция HEAD была сa67c98 Added a default value
Переключились на ветку «main»
● danielmihai@MacBook-Air-Danik hello % git status
Текущая ветка: main
Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
    .gitconfig
    .profile

индекс пуст, но есть неотслеживаемые файлы
(используйте «git add», чтобы проиндексировать их)
● danielmihai@MacBook-Air-Danik hello % git checkout hello.rb
Updated 0 paths from the index
● danielmihai@MacBook-Air-Danik hello % git status
Текущая ветка: main
Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
    .gitconfig
    .profile

индекс пуст, но есть неотслеживаемые файлы
(используйте «git add», чтобы проиндексировать их)
● danielmihai@MacBook-Air-Danik hello % cat hello.rb
# Default is "World"
name = ARGV.first || "World"

puts "Hello, #{name}!"  
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.15. Откат изменений в рабочей директории

Описание: Изменения отменены до индексации с помощью команды checkout.

```
1 git status
2 git checkout hello.rb
```

Задание 15: Undoing Staged Changes (before committing)

```
● danielmihai@MacBook-Air-Danik hello % git add hello.rb
● danielmihai@MacBook-Air-Danik hello % git status
Текущая ветка: main
Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
изменено:      hello.rb

Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
  .gitconfig
  .profile

● danielmihai@MacBook-Air-Danik hello % git reset HEAD hello.rb
Непроиндексированные изменения после сброса:
 M      hello.rb
● danielmihai@MacBook-Air-Danik hello % git checkout hello.rb
Updated 1 path from the index
● danielmihai@MacBook-Air-Danik hello % git status
Текущая ветка: main
Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
  .gitconfig
  .profile

индекс пуст, но есть неотслеживаемые файлы
  (используйте «git add», чтобы проиндексировать их)
○ danielmihai@MacBook-Air-Danik hello % █
```

Рис. 4.16. Отмена индексированных изменений

Описание: Использован git reset для удаления файла из индекса и возврата к рабочей копии.

```
1 git add hello.rb
2 git reset HEAD hello.rb
3 git checkout hello.rb
```

Задание 16: Undoing Committed Changes

```
● danielmihai@MacBook-Air-Danik hello % git add hello.rb
● danielmihai@MacBook-Air-Danik hello % git commit -m "Oops, we didn't want this commit"
 [main 78fa0ef] Oops, we didn't want this commit
 1 file changed, 1 insertion(+), 1 deletion(-)
⊗ danielmihai@MacBook-Air-Danik hello % git revert HEAD
error: Ваши локальные изменения в указанных файлах будут перезаписаны при слиянии:
      hello.rb
Сделайте коммит или спрячьте ваши изменения перед слиянием веток.
Прерываю
fatal: сбой обращения изменений коммита
● danielmihai@MacBook-Air-Danik hello % git hist
* 78fa0ef 2025-06-20 | Oops, we didn't want this commit (HEAD -> main) [Daniel]
* 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
* ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello % █
```

Рис. 4.17. Откат коммита с помощью git revert

Описание: Выполнен откат коммита путём создания нового коммита с противоположными изменениями.

```
1 git add hello.rb
2 git commit -m "Oops, we didn't want this commit"
3 git revert HEAD
4 git hist
```

Задание 17: Removing Commits from a Branch

```
● danielmihai@MacBook-Air-Danik hello % git hist
* 78fa0ef 2025-06-20 | Oops, we didn't want this commit (HEAD -> main) [Daniel]
* 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
* ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
● danielmihai@MacBook-Air-Danik hello % git tag oops
● danielmihai@MacBook-Air-Danik hello % git reset --hard v1
Указатель HEAD сейчас на коммите 3ff54eb Added a comment
● danielmihai@MacBook-Air-Danik hello % git hist
* 3ff54eb 2025-06-20 | Added a comment (HEAD -> main, tag: v1) [Daniel]
* ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
● danielmihai@MacBook-Air-Danik hello % git hist --all
* 78fa0ef 2025-06-20 | Oops, we didn't want this commit (tag: oops) [Daniel]
* 3ff54eb 2025-06-20 | Added a comment (HEAD -> main, tag: v1) [Daniel]
* ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello % █
```

Рис. 4.18. Удаление последних коммитов через reset

Описание: Удалены последние коммиты с помощью git reset --hard до состояния коммита с тегом v1.

```
1 git tag oops
2 git reset --hard v1
3 git hist --all
```

Задание 18: Remove the oops tag

```
● danielmihai@MacBook-Air-Danik hello % git tag -d oops
  Метка «oops» удалена (была 78fa0ef)
● danielmihai@MacBook-Air-Danik hello % git hist --all
  * 3ff54eb 2025-06-20 | Added a comment (HEAD -> main, tag: v1) [Daniel]
  * ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
  * 61ac871 2025-06-20 | Unrelated change to c [Daniel]
  * 9650de5 2025-06-20 | Changes for a and b [Daniel]
  * 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello % █
```

Рис. 4.19. Удаление тега oops

Описание: Удалён временный тег для «плохого» коммита.

```
1 git tag -d oops
2 git hist --all
```

Задание 19: Amending Commits

```
● danielmihai@MacBook-Air-Danik hello % git add hello.rb
● danielmihai@MacBook-Air-Danik hello % git commit -m "Add an author comment"
  [main 849d79b] Add an author comment
  1 file changed, 2 insertions(+), 1 deletion(-)
● danielmihai@MacBook-Air-Danik hello % git add hello.rb
● danielmihai@MacBook-Air-Danik hello % git commit --amend -m "Add an author/email comment"
  [main 31f8ad1] Add an author/email comment
  Date: Fri Jun 20 09:18:39 2025 +0300
  1 file changed, 2 insertions(+), 1 deletion(-)
● danielmihai@MacBook-Air-Danik hello % git hist
  * 31f8ad1 2025-06-20 | Add an author/email comment (HEAD -> main) [Daniel]
  * 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
  * ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
  * 61ac871 2025-06-20 | Unrelated change to c [Daniel]
  * 9650de5 2025-06-20 | Changes for a and b [Daniel]
  * 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello % █
```

Рис. 4.20. Изменение последнего коммита

Описание: Использован git commit –amend для обновления последнего коммита с добавлением email.

```
1 git add hello.rb
2 git commit -m "Add an author comment"
3 git add hello.rb
4 git commit --amend -m "Add an author/email comment"
5 git hist
```

Задание 20: Moving Files

```
● danielmihai@MacBook-Air-Danik hello % mkdir lib
● danielmihai@MacBook-Air-Danik hello % git mv hello.rb lib
● danielmihai@MacBook-Air-Danik hello % git status
  Текущая ветка: main
  Изменения, которые будут включены в коммит:
    (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
      переименовано: hello.rb -> lib/hello.rb

  Неотслеживаемые файлы:
    (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
      .gitconfig
      .profile

⊗ danielmihai@MacBook-Air-Danik hello % mkdir lib
  mkdir: lib: File exists
● danielmihai@MacBook-Air-Danik hello % git commit -m "Moved hello.rb to lib"
[main c2500d2] Moved hello.rb to lib
  1 file changed, 0 insertions(+), 0 deletions(-)
    rename hello.rb => lib/hello.rb (100%)
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.21. Перемещение файла hello.rb в папку lib

Описание: Файл hello.rb перемещён в папку lib с помощью git mv и зафиксирован в репозитории.

```
1  mkdir lib
2  git mv hello.rb lib
3  git status
4  git commit -m "Moved hello.rb to lib"
```

Задание 21: More Structure

```
● danielmihai@MacBook-Air-Danik hello % git add Rakefile
● danielmihai@MacBook-Air-Danik hello % git commit -m "Added a Rakefile."
[main bacd62b] Added a Rakefile.
  1 file changed, 7 insertions(+)
  create mode 100644 Rakefile
● danielmihai@MacBook-Air-Danik hello % rake
Hello, World!
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.22. Добавление Rakefile в проект

Описание: Добавлен файл Rakefile с задачами и выполнен rake.

```
1  git add Rakefile
2  git commit -m "Added a Rakefile"
3  rake
```

Задание 22: Git Internals – .git Directory

```
danielmihai@MacBook-Air-Danik hello % git add Rakefile
danielmihai@MacBook-Air-Danik hello % git commit -m "Added a Rakefile."
[main bacd62b] Added a Rakefile.
 1 file changed, 7 insertions(+)
 create mode 100644 Rakefile
danielmihai@MacBook-Air-Danik hello % rake
Hello, World!
● danielmihai@MacBook-Air-Danik hello % clear
● danielmihai@MacBook-Air-Danik hello % ls -C .git
  COMMIT_EDITMSG  ORIG_HEAD      description      index      logs      refs
  HEAD           config        hooks        info      objects
● danielmihai@MacBook-Air-Danik hello % ls -C .git/objects
   14      2c      3e      4a      52      72      78      83      96      9a      b8      c2
   18      31      3f      4b      61      74      79      84      99      aa      ba      ca
● danielmihai@MacBook-Air-Danik hello % ls -C .git/objects/main
ls: .git/objects/main: No such file or directory
● danielmihai@MacBook-Air-Danik hello % ls -C .git/objects/v1
ls: .git/objects/v1: No such file or directory
● danielmihai@MacBook-Air-Danik hello % ls -C .git/objects/09
ls: .git/objects/09: No such file or directory
● danielmihai@MacBook-Air-Danik hello % git branch
 * main
● danielmihai@MacBook-Air-Danik hello % ls -C .git/objects/main
ls: .git/objects/main: No such file or directory
○ danielmihai@MacBook-Air-Danik hello % ls -C .git/objects/<dir>
zsh: parse error near `\'n'
● danielmihai@MacBook-Air-Danik hello % ls -C .git/objects/e8
ls: .git/objects/e8: No such file or directory
● danielmihai@MacBook-Air-Danik hello % git log --oneline
 bacd62b (HEAD -> main) Added a Rakefile.
 c2500d2 Moved hello.rb to lib
 31f8ad1 Add an author/email comment
 3ff54eb (tag: v1) Added a comment
  ca67c98 (tag: v1-beta) Added a default value
  61ac871 Unrelated change to c
  9650de5 Changes for a and b
  2cb1af1 First Commit
● danielmihai@MacBook-Air-Danik hello % ls -C .git/objects/ba
cd62b307796d8a7d3d41526789c2ef12678449
● danielmihai@MacBook-Air-Danik hello % cat .git/config
[core]
  repositoryformatversion = 0
  filemode = true
  bare = false
  logallrefupdates = true
  ignorecase = true
  precomposeunicode = true
[remote "origin"]
  url = https://github.com/messsimo/uni_practice.git
```

Рис. 4.23. Содержимое директории .git

```
● danielmihai@MacBook-Air-Danik hello % ls .git/refs
  heads  tags
● danielmihai@MacBook-Air-Danik hello % ls .git/refs/heads
  main
● danielmihai@MacBook-Air-Danik hello % ls .git/refs/tags
  v1      v1-beta
● danielmihai@MacBook-Air-Danik hello % cat .git/refs/tags/v1
  3ff54eb9ccb383ff806bc8a3c35e83b20fce4b16
● danielmihai@MacBook-Air-Danik hello % cat .git/HEAD
  ref: refs/heads/main
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.24. Просмотр refs и config

Описание: Исследована структура внутренней директории .git и содержимое config, HEAD, refs.

```
1  ls -C .git
2  ls -C .git/objects
3  cat .git/config
4  ls .git/refs
5  git tag
```

Задание 23: Working with Git Objects

```
└── first commit
● danielmihai@MacBook-Air-Danik hello % git hist --max-count=1
 * bcd62b 2025-06-20 | Added a Rakefile. (HEAD -> main) [Daniel]
● danielmihai@MacBook-Air-Danik hello % git cat-file -t bcd62b
commit
● danielmihai@MacBook-Air-Danik hello % git cat-file -p bcd62b
tree 83555c99767fd09e410b11cadfc0fd55a9d77d74
parent c2500d29599652b9bfe28cd1e48183b9e2c3fcda
author Daniel <danielmihai.it@mail.ru> 1750400491 +0300
committer Daniel <danielmihai.it@mail.ru> 1750400491 +0300

Added a Rakefile.
● danielmihai@MacBook-Air-Danik hello % git cat-file -p bcd62b
tree 83555c99767fd09e410b11cadfc0fd55a9d77d74
parent c2500d29599652b9bfe28cd1e48183b9e2c3fcda
author Daniel <danielmihai.it@mail.ru> 1750400491 +0300
committer Daniel <danielmihai.it@mail.ru> 1750400491 +0300

Added a Rakefile.
● danielmihai@MacBook-Air-Danik hello % git cat-file -p bcd62b
tree 83555c99767fd09e410b11cadfc0fd55a9d77d74
parent c2500d29599652b9bfe28cd1e48183b9e2c3fcda
author Daniel <danielmihai.it@mail.ru> 1750400491 +0300
committer Daniel <danielmihai.it@mail.ru> 1750400491 +0300

Added a Rakefile.
● danielmihai@MacBook-Air-Danik hello % git cat-file -p bcd62b
tree 83555c99767fd09e410b11cadfc0fd55a9d77d74
parent c2500d29599652b9bfe28cd1e48183b9e2c3fcda
author Daniel <danielmihai.it@mail.ru> 1750400491 +0300
committer Daniel <danielmihai.it@mail.ru> 1750400491 +0300

Added a Rakefile.
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.25. Просмотр содержимого объектов с помощью cat-file

Описание: Изучены типы и содержимое git-объектов: commit, tree, blob.

```
1 git hist --max-count=1
2 git cat-file -t <hash>
3 git cat-file -p <hash>
```

Задание 24: Creating a Branch

```
● danielmihai@MacBook-Air-Danik hello % git checkout -b greet
Переключились на новую ветку «greet»
● danielmihai@MacBook-Air-Danik hello % git status
Текущая ветка: greet
Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
    .gitconfig
    .profile

индекс пуст, но есть неотслеживаемые файлы
(используйте «git add», чтобы проиндексировать их)
● danielmihai@MacBook-Air-Danik hello % git add lib/greeter.rb
● danielmihai@MacBook-Air-Danik hello % git commit -m "Added greeter class"
[greet ef1feb6] Added greeter class
 1 file changed, 8 insertions(+)
  create mode 100644 lib/greeter.rb
● danielmihai@MacBook-Air-Danik hello % git add lib/hello.rb
● danielmihai@MacBook-Air-Danik hello % git commit -m "Hello uses Greeter"
[greet 0ea2b50] Hello uses Greeter
 1 file changed, 4 insertions(+), 2 deletions(-)
● danielmihai@MacBook-Air-Danik hello % git add Rakefile
● danielmihai@MacBook-Air-Danik hello % git commit -m "Updated Rakefile"
[greet 33dc544] Updated Rakefile
 1 file changed, 1 insertion(+), 1 deletion(-)
● danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.26. Создание и переход на новую ветку greet

Описание: Создана ветка greet и добавлены новые файлы: greeter.rb, обновлены hello.rb и Rakefile.

```
1 git checkout -b greet
2 git add lib/greeter.rb
3 git commit -m "Added greeter class"
4 git add lib/hello.rb
5 git commit -m "Hello uses Greeter"
6 git add Rakefile
7 git commit -m "Updated Rakefile"
```

Задание 25: Navigating Branches

```
● danielmihai@MacBook-Air-Danik hello % git hist --all
* 33dc544 2025-06-20 | Updated Rakefile (HEAD -> greet) [Daniel]
* 0ea2b50 2025-06-20 | Hello uses Greeter [Daniel]
* ef1feb6 2025-06-20 | Added greeter class [Daniel]
* bacd62b 2025-06-20 | Added a Rakefile. (main) [Daniel]
* c2500d2 2025-06-20 | Moved hello.rb to lib [Daniel]
* 31f8ad1 2025-06-20 | Add an author/email comment [Daniel]
* 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
* ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
● danielmihai@MacBook-Air-Danik hello % git checkout main
Переключились на ветку «main»
● danielmihai@MacBook-Air-Danik hello % cat lib/hello.rb
# Default is World
# Author: Jim Weirich (jim@somewhere.com)
name = ARGV.first || "World"

puts "Hello, #{name}!"%
● danielmihai@MacBook-Air-Danik hello % git checkout greet
Переключились на ветку «greet»
● danielmihai@MacBook-Air-Danik hello % cat lib/hello.rb
require 'greeter'

# Default is World
name = ARGV.first || "World"

greeter = Greeter.new(name)
puts greeter.greet%
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.27. Переход между ветками main и greet

Описание: Выполнен переход между ветками, просмотрено содержимое файлов в каждой ветке.

```
1 git checkout main
2 cat lib/hello.rb
3 git checkout greet
4 cat lib/hello.rb
```

Задание 26: Changes in Main

```
● danielmihai@MacBook-Air-Danik hello % git checkout main
Переключились на ветку «main»
● danielmihai@MacBook-Air-Danik hello % git add README
☒ danielmihai@MacBook-Air-Danik hello % git commit -m "Added README"
Текущая ветка: main
Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
  .gitconfig
  .profile
  ReadMe

индекс пуст, но есть неотслеживаемые файлы
  (используйте «git add», чтобы проиндексировать их)
● danielmihai@MacBook-Air-Danik hello % git add README
☒ danielmihai@MacBook-Air-Danik hello % git commit -m "Added README"
Текущая ветка: main
Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
  .gitconfig
  .profile
  ReadMe

индекс пуст, но есть неотслеживаемые файлы
  (используйте «git add», чтобы проиндексировать их)
● danielmihai@MacBook-Air-Danik hello % git add ReadMe
● danielmihai@MacBook-Air-Danik hello % git commit -m "Added README"
[main b076256] Added README
  1 file changed, 1 insertion(+)
  create mode 100644 ReadMe
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.28. Изменения в основной ветке main (добавлен README)

Описание: В ветке main создан и закоммичен файл README.

```
1 git checkout main
2 git add README
3 git commit -m "Added README"
```

Задание 27: Viewing Diverging Branches

```
● danielmihai@MacBook-Air-Danik hello % git hist --all
* b076256 2025-06-20 | Added README (HEAD -> main) [Daniel]
| * 33dc544 2025-06-20 | Updated Rakefile (greet) [Daniel]
| * 0ea2b50 2025-06-20 | Hello uses Greeter [Daniel]
| * ef1feb6 2025-06-20 | Added greeter class [Daniel]
|
* bacd62b 2025-06-20 | Added a Rakefile. [Daniel]
* c2500d2 2025-06-20 | Moved hello.rb to lib [Daniel]
* 31f8ad1 2025-06-20 | Add an author/email comment [Daniel]
* 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
* ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello % █
```

Рис. 4.29. Просмотр ветвлений с помощью git hist --all

Описание: Использован git hist --all для визуализации расхождения веток main и greet.

```
1 git hist --all
```

Задание 28: Merging

```
● danielmihai@MacBook-Air-Danik hello % git hist --all
* b076256 2025-06-20 | Added README (main) [Daniel]
| * 33dc544 2025-06-20 | Updated Rakefile (HEAD -> greet) [Daniel]
| * 0ea2b50 2025-06-20 | Hello uses Greeter [Daniel]
| * ef1feb6 2025-06-20 | Added greeter class [Daniel]
|
* bacd62b 2025-06-20 | Added a Rakefile. [Daniel]
* c2500d2 2025-06-20 | Moved hello.rb to lib [Daniel]
* 31f8ad1 2025-06-20 | Add an author/email comment [Daniel]
* 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
* ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello % █
```

Рис. 4.30. Слияние ветки main в greet

Описание: Ветка main объединена с greet. Возник merge-коммит.

```
1 git checkout greet
2 git merge main
3 git hist --all
```

Задание 29: Creating a Conflict

```
danielmihai@MacBook-Air-Danik hello % git hist --all
* b076256 2025-06-20 | Added README (main) [Daniel]
| * 33dc544 2025-06-20 | Updated Rakefile (HEAD -> greet) [Daniel]
| * 0ea2b50 2025-06-20 | Hello uses Greeter [Daniel]
| * ef1feb6 2025-06-20 | Added greeter class [Daniel]
|/
* bacd62b 2025-06-20 | Added a Rakefile. [Daniel]
* c2500d2 2025-06-20 | Moved hello.rb to lib [Daniel]
* 31f8ad1 2025-06-20 | Add an author/email comment [Daniel]
* 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
* ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
danielmihai@MacBook-Air-Danik hello % clear
danielmihai@MacBook-Air-Danik hello % git checkout main
Переключились на ветку «main»
danielmihai@MacBook-Air-Danik hello % git add lib/hello.rb
danielmihai@MacBook-Air-Danik hello % git commit -m "Made interactive"
[main 4728290] Made interactive
 1 file changed, 3 insertions(+), 4 deletions(-)
danielmihai@MacBook-Air-Danik hello % git hist --all
* 4728290 2025-06-20 | Made interactive (HEAD -> main) [Daniel]
* b076256 2025-06-20 | Added README [Daniel]
| * 33dc544 2025-06-20 | Updated Rakefile (greet) [Daniel]
| * 0ea2b50 2025-06-20 | Hello uses Greeter [Daniel]
| * ef1feb6 2025-06-20 | Added greeter class [Daniel]
|/
* bacd62b 2025-06-20 | Added a Rakefile. [Daniel]
* c2500d2 2025-06-20 | Moved hello.rb to lib [Daniel]
* 31f8ad1 2025-06-20 | Add an author/email comment [Daniel]
* 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
* ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
danielmihai@MacBook-Air-Danik hello % █
```

Рис. 4.31. Создание конфликта при изменении hello.rb в main

Описание: В ветке main внесены изменения, конфликтующие с greet.

```
1 git checkout main
2 [редактирование lib/hello.rb]
3 git add lib/hello.rb
4 git commit -m "Made interactive"
```

Задание 30: Resolving Conflicts

```
● danielmihai@MacBook-Air-Danik hello % git checkout greet
  Переключились на ветку «greet»
⊗ danielmihai@MacBook-Air-Danik hello % git merge main
  Автослияние lib/hello.rb
  КОНФЛИКТ (содержимое): Конфликт слияния в lib/hello.rb
  Сбой автоматического слияния; исправьте конфликты, затем зафиксируйте результат.
● danielmihai@MacBook-Air-Danik hello % git add lib/hello.rb
● danielmihai@MacBook-Air-Danik hello % git commit -m "Merged main fixed conflict."
  [greet b335413] Merged main fixed conflict.
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.32. Разрешение конфликта и коммит

Описание: Конфликт разрешён вручную. Обновлён hello.rb, изменения закоммичены.

```
1 git checkout greet
2 git merge main
3 [разрешение конфликта]
4 git add lib/hello.rb
5 git commit -m "Merged main fixed conflict."
```

Задание 31: Rebasing vs Merging

ЧЧЧ **Описание:** Теоретическая часть по сравнению git merge и git rebase.

```
1 # без команд (обзор)
```

Задание 32: Resetting the Greet Branch

```
● danielmihai@MacBook-Air-Danik hello % git checkout greet
  Уже на «greet»
● danielmihai@MacBook-Air-Danik hello % git reset --hard 33dc544
  Указатель HEAD сейчас на коммите 33dc544 Updated Rakefile
● danielmihai@MacBook-Air-Danik hello % git hist --all
* 4728290 2025-06-20 | Made interactive (main) [Daniel]
* b076256 2025-06-20 | Added README [Daniel]
| * 33dc544 2025-06-20 | Updated Rakefile (HEAD -> greet) [Daniel]
| * 0ea2b50 2025-06-20 | Hello uses Greeter [Daniel]
| * ef1feb6 2025-06-20 | Added greeter class [Daniel]
|
* bacd62b 2025-06-20 | Added a Rakefile. [Daniel]
* c2500d2 2025-06-20 | Moved hello.rb to lib [Daniel]
* 31f8ad1 2025-06-20 | Add an author/email comment [Daniel]
* 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
* ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
* 61ac871 2025-06-20 | Unrelated change to c [Daniel]
* 9650de5 2025-06-20 | Changes for a and b [Daniel]
* 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.33. Сброс ветки greet до коммита перед merge

Описание: Ветка greet сброшена до коммита перед первым merge для демонстрации rebase.

```
1 git checkout greet
2 git hist
3 git reset --hard <hash>
```

Задание 33: Resetting the Main Branch

```
● danielmihai@MacBook-Air-Danik hello % git checkout main
  Переключились на ветку «main»
● danielmihai@MacBook-Air-Danik hello % git hist
  * 4728290 2025-06-20 | Made interactive (HEAD -> main) [Daniel]
  * b076256 2025-06-20 | Added README [Daniel]
  * bacd62b 2025-06-20 | Added a Rakefile. [Daniel]
  * c2500d2 2025-06-20 | Moved hello.rb to lib [Daniel]
  * 31f8ad1 2025-06-20 | Add an author/email comment [Daniel]
  * 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
  * ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
  * 61ac871 2025-06-20 | Unrelated change to c [Daniel]
  * 9650de5 2025-06-20 | Changes for a and b [Daniel]
  * 2cb1af1 2025-06-20 | First Commit [Daniel]
● danielmihai@MacBook-Air-Danik hello % git reset --hard 4728290
  Указатель HEAD сейчас на коммите 4728290 Made interactive
● danielmihai@MacBook-Air-Danik hello % git hist --all
  * 4728290 2025-06-20 | Made interactive (HEAD -> main) [Daniel]
  * b076256 2025-06-20 | Added README [Daniel]
  | * 33dc544 2025-06-20 | Updated Rakefile (greet) [Daniel]
  | * 0ea2b50 2025-06-20 | Hello uses Greeter [Daniel]
  | * ef1feb6 2025-06-20 | Added greeter class [Daniel]
  |
  * bacd62b 2025-06-20 | Added a Rakefile. [Daniel]
  * c2500d2 2025-06-20 | Moved hello.rb to lib [Daniel]
  * 31f8ad1 2025-06-20 | Add an author/email comment [Daniel]
  * 3ff54eb 2025-06-20 | Added a comment (tag: v1) [Daniel]
  * ca67c98 2025-06-20 | Added a default value (tag: v1-beta) [Daniel]
  * 61ac871 2025-06-20 | Unrelated change to c [Daniel]
  * 9650de5 2025-06-20 | Changes for a and b [Daniel]
  * 2cb1af1 2025-06-20 | First Commit [Daniel]
○ danielmihai@MacBook-Air-Danik hello %
```

Рис. 4.34. Откат ветки main перед конфликтным коммитом

Описание: Ветка main сброшена до коммита перед добавлением интерактивного ввода для избежания конфликта.

```
1 git checkout main
2 git hist
3 git reset --hard <hash>
4 git hist --all
```

Заключение и рекомендации

Общие выводы.

В результате выполнения практической работы были достигнуты все поставленные цели.

Студент овладел базовыми навыками работы с системой вёрстки \LaTeX и системой контроля версий Git [3], [4], а также закрепил знания с помощью платформы Learn Git Branching [1].

На практике были выполнены следующие ключевые этапы:

- подготовка отчётного документа в \LaTeX с использованием профессиональных пакетов для оформления;
- прохождение всех блоков интерактивной платформы Learn Git Branching;
- визуализация и пояснение каждой задачи с приложением скриншотов и кода;
- соблюдение требований к структуре, оформлению и библиографическому аппарату.

Полученные знания и навыки являются фундаментальными для последующей профессиональной деятельности в сфере ИТ и научных исследований. Документ может быть использован как образец оформления практических работ и отчётов.

Рекомендации.

- Рекомендуется продолжить углублённое изучение Git, включая такие темы, как rebase, stash, cherry-pick и CI/CD-интеграции.
- Освоение более продвинутых возможностей \LaTeX , таких как TikZ, Beamer и автоматическая генерация диаграмм, позволит расширить инструментарий вёрстки.
- В рамках будущих курсов или проектов имеет смысл применить полученные навыки на реальных проектах, используя GitHub как платформу совместной работы.

Дополнительные материалы.

Весь исходный код работы, включая файлы \LaTeX , изображения и библиографию, доступен по следующей ссылке:

https://github.com/messsimo/uni_practice

Скомпилированный итоговый документ в формате PDF можно скачать здесь:

https://github.com/messsimo/uni_practice

Эти материалы предоставлены для свободного изучения, повторного использования и адаптации в образовательных целях.

Список литературы

- [1] «Learn Git Branching — интерактивное обучение», <https://learngitbranching.js.org>, 2024. Дата обр. 20 июня 2025.
- [2] «Официальная документация Git», <https://git-scm.com>, 2024. Дата обр. 20 июня 2025.
- [3] «Pro Git book (на русском)», <https://git-scm.com/book/ru/v2>, 2024. Дата обр. 20 июня 2025.
- [4] «LaTeX2e: An unofficial reference manual», <https://latexref.xyz>, 2024. Дата обр. 20 июня 2025.
- [5] «The L^AT_EX Project Website», <https://www.latex-project.org>, 2024. Дата обр. 20 июня 2025.
- [6] «Overleaf: Online LaTeX Editor», <https://www.overleaf.com>, 2024. Дата обр. 20 июня 2025.
- [7] «The minted package – Highlighting source code in LaTeX», <https://ctan.org/pkg/minted>, 2024. Дата обр. 20 июня 2025.