ПРАКИТЕЧКОЕ ЗАНЯТИЕ №3

тема: изучении Git-команд и выгрузка работы

на GitHub

ОСНОВНАЯ ЧАСТЬ

1. Основы
2. Вводная часть

В задачах №1-2 расписывались команды для создания коммитов и ветвей (git commit и git branch соответственно).

В задаче №3 с помощью git checkout -b [branchName] Создали ветку и переключились на неё, создали коммит (git commit), перелечились на ветку main, создали еще один коммит, командой git branch слили ветку bugfix с веткой main тем самым получили коммит с двумя родителями. (Смотреть рисунок 1.1).

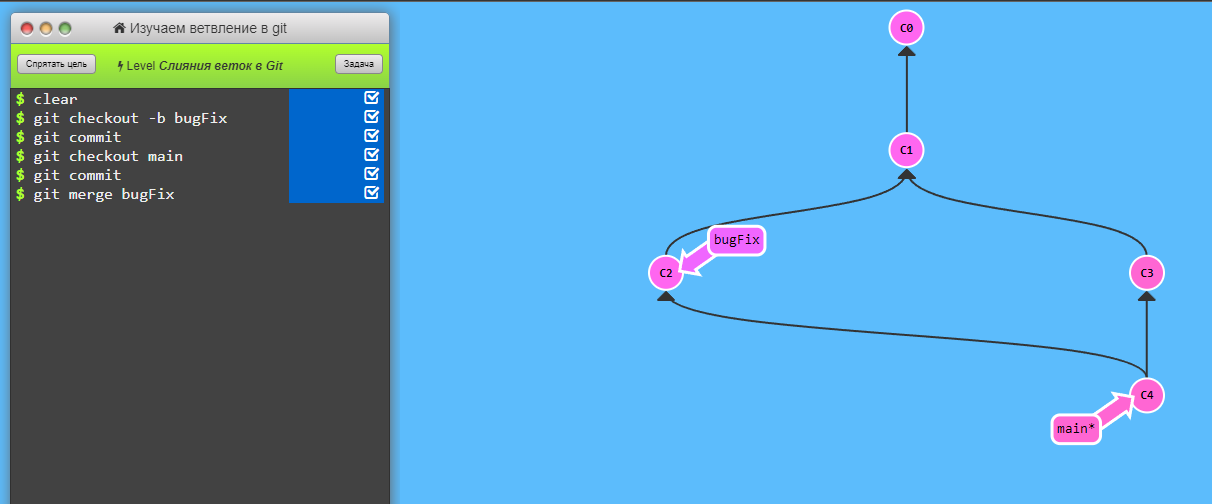


Рисунок 1.1 – Слияние веток в git

Задача №4 знакомит пользователя с командой rebase, которая, по сути, копирует коммит в другое место, цель задачи была выполнена командами из прошлой задачи, но вместо “слияния” мы использовали rebase.

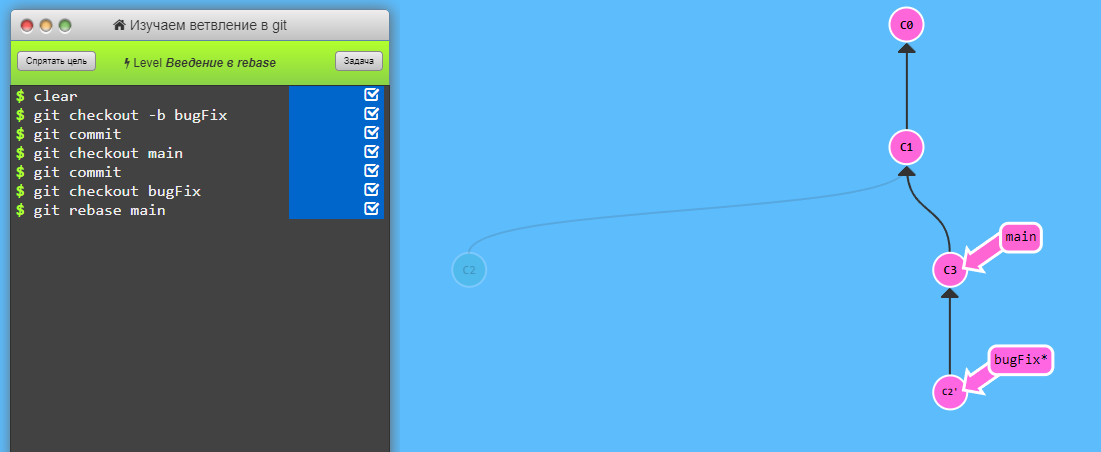


Рисунок 1.2 – Введение в rebase

1. Едем дальше

Задача №1 описывает HEAD – имя текущего выбранного коммита, если не выбран коммит, HEAD указывает на выбранную ветку. Для этого задания потребовалась команда – «git checkout c4».

Задача №2 описывает работу относительных ссылок, знакомит с оператором “^”, возвращающий родителя коммита так как не всегда удобно вводить хэш коммита.

В задаче №3 необходимо переместить указатель HEAD и некоторые ветки на другие коммиты (Смотреть рисунок 1.3).

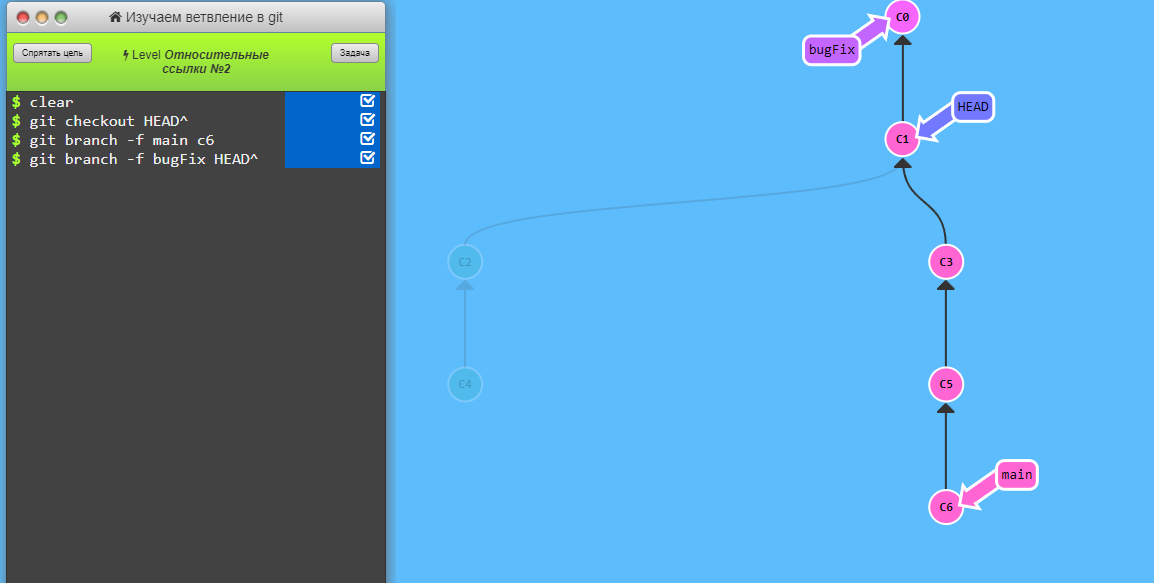


Рисунок 1.3 – Относительные ссылки №2

Задача №4 ставит цель отменить изменения на условно локальном и условной удалённом коммите, для локальных коммитов используется git reset, для которого необходимо указать до какого момента отменить изменения, для удалённого используется git revert, с указанием коммита, который необходимо изменить (Смотреть рисунок 1.4).

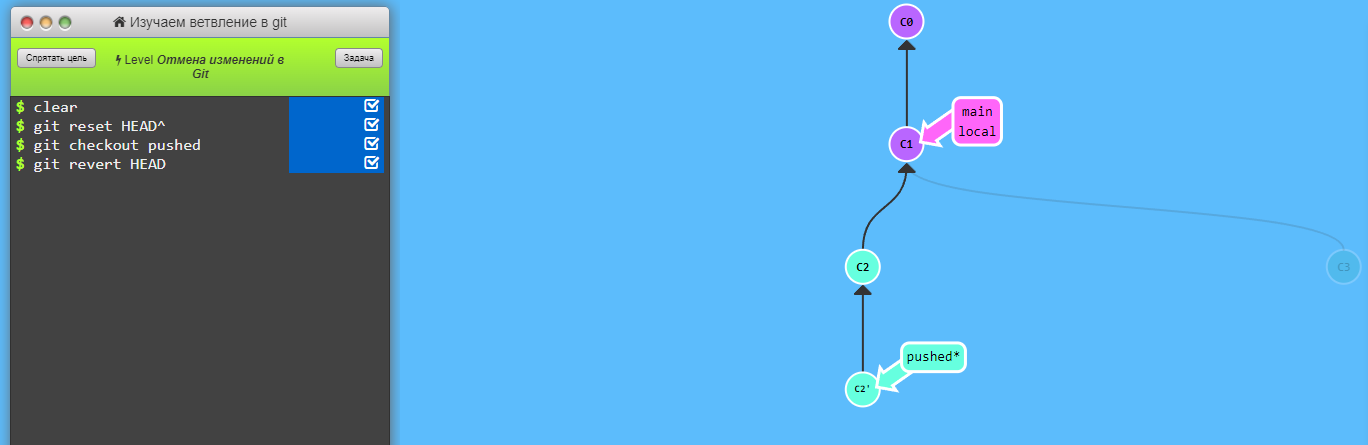


Рисунок 1.4 – отмена изменений в git

1. Перемещаем труды туда-сюда

Задача №1 описывает работу команды cherry-pick, которая переносит указанные коммиты в нужное место (HEAD). (Смотреть рисунок 1.5).



Рисунок 1.5 – Введение в cherry-pick

Задача №2 объясняет поведение команды rebase с опцией -I, а именно – позволяет менять порядок коммитов, удалять их и сливать.

1. Сборная солянка

Задача №1 требует выбора одного коммита с помощью ранее изученных инструментов (rebase / chery-pick) (Смотреть рисунок 6).

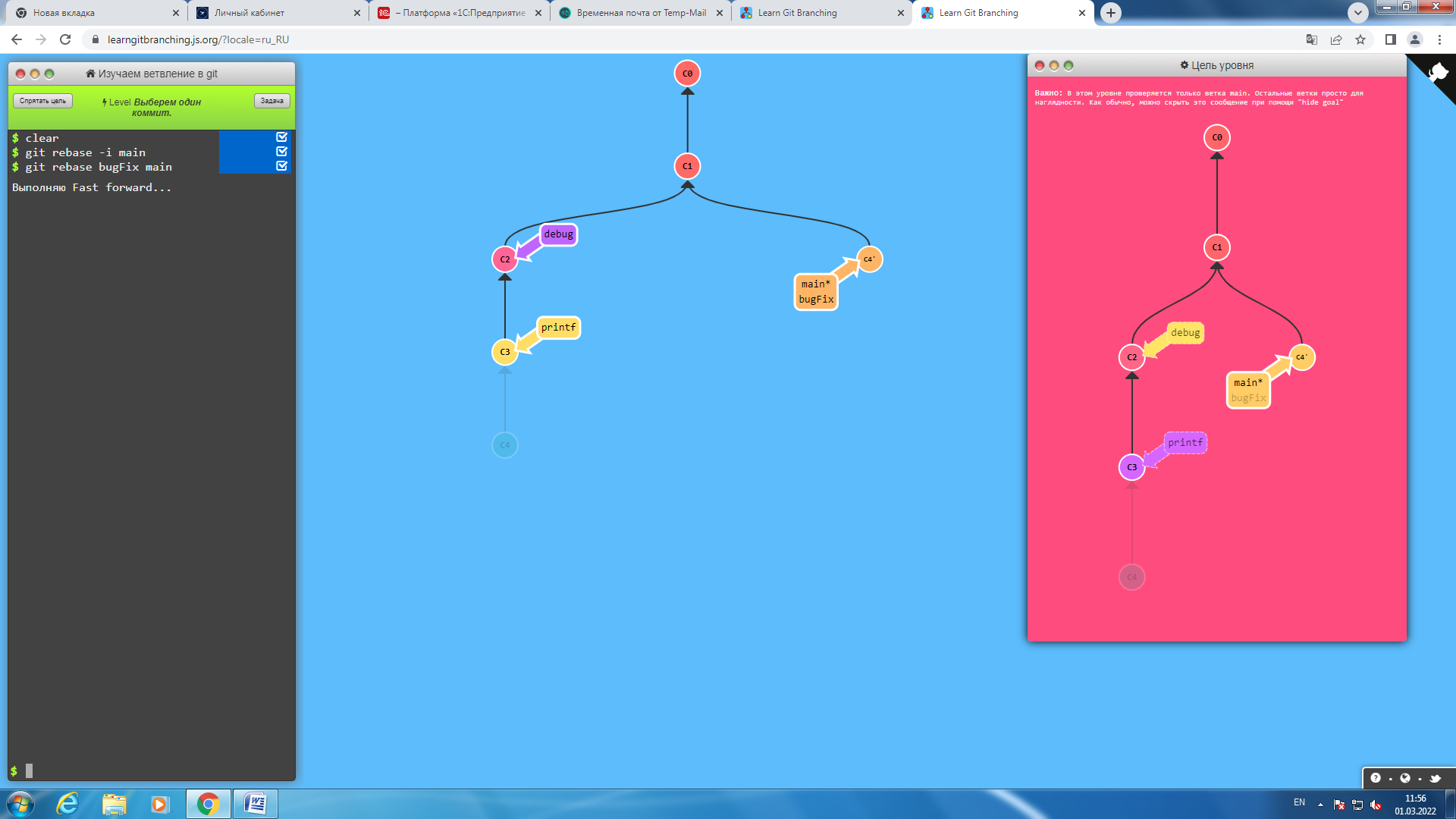


Рисунок 1.6 – Выберем один коммит

Задача №2 требует внесение изменений в коммит, выступающий родительским для последнего в ветке, для этого необходимо пересобрать ветку (git rebase -i) так, что бы необходимый коммит был вершиной ветки, внести изменения (git commit --amend), снова пересобрать в правильном порядке. (Смотреть рисунок 1.7).

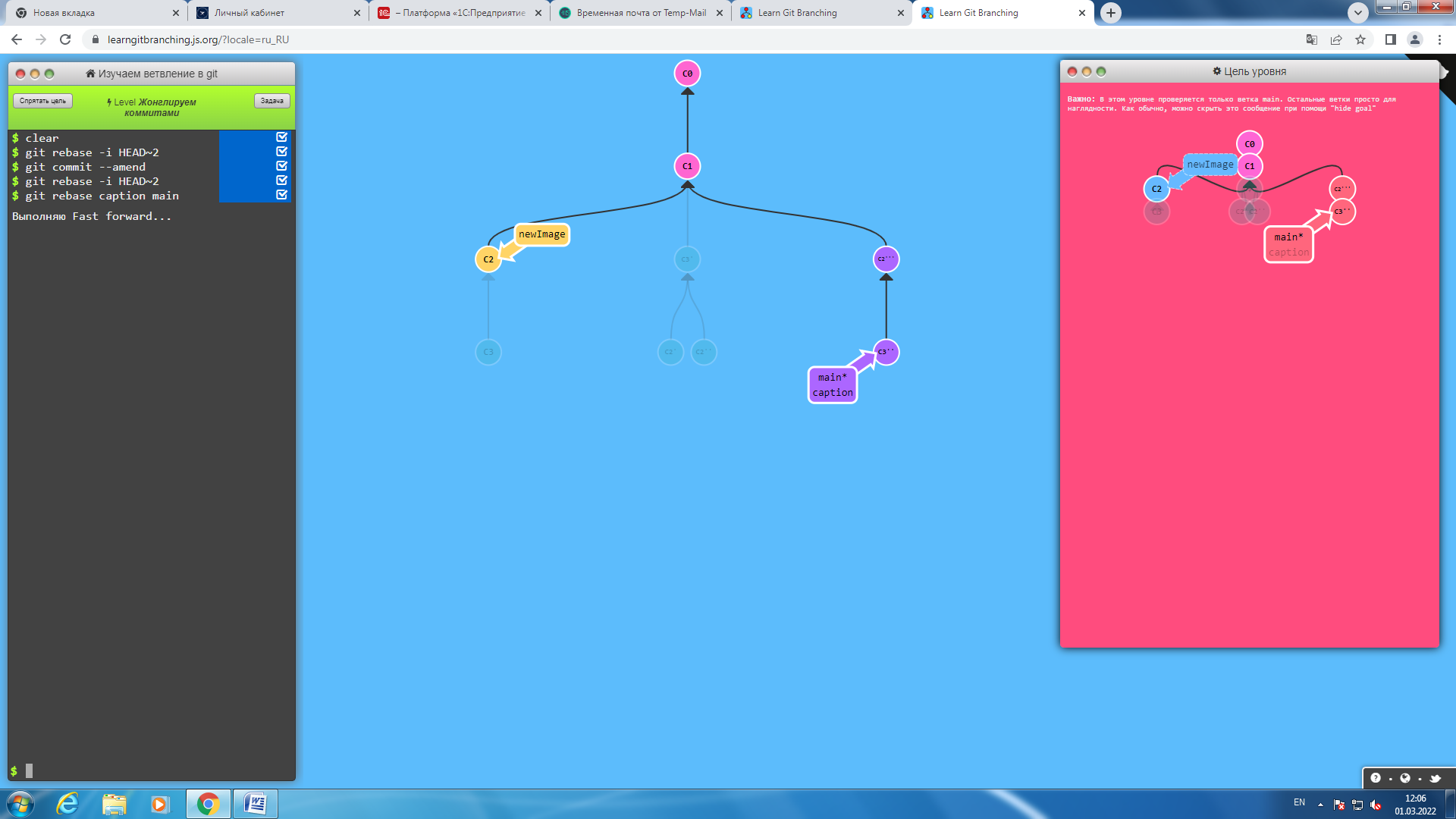


Рисунок 1.7 – Жонглируем коммитами

В задаче №3 необходимо решить проблему из второй задачи средствами git cherry-pick (смотреть рисунок 1.8).

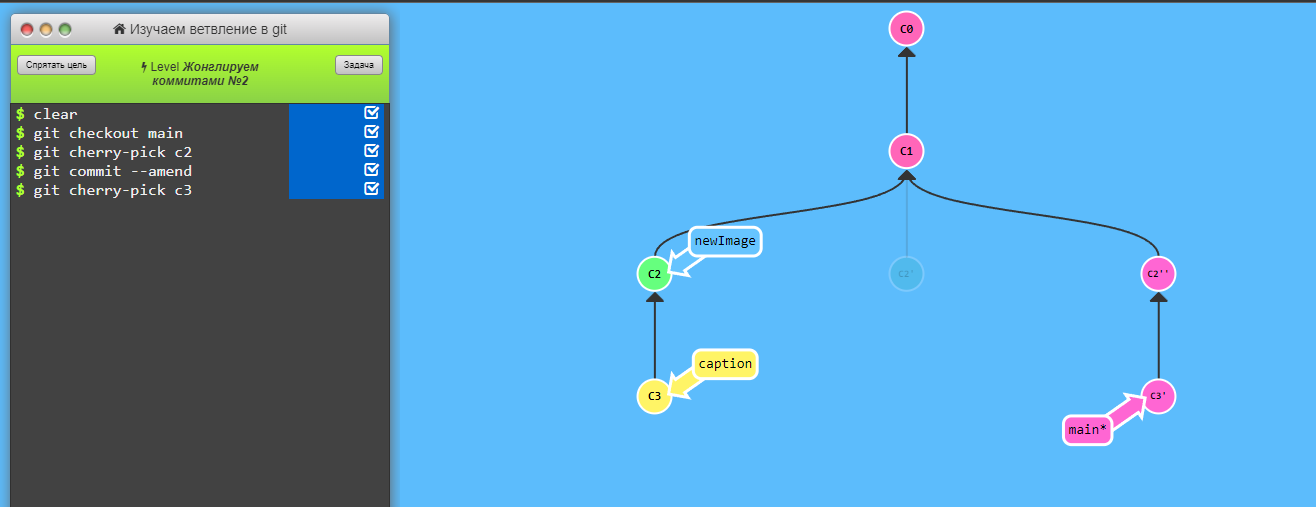


Рисунок 1.8 – Жонглируем коммитами №2

Задача №4 знакомит пользователя с Git tag, элемент, который будет всегда ссылаться на один коммит. Git tag [tag name] [commit name].

Задача №5 описывает команду git describe, которая позволяет понять как далеко от коммита находится тэг, и что это за тэг.

1. Продвинутый уровень

Задача №1 предполагает перенос различных коммитов из разных веток в main, для этого используем команду git rebase (смотреть рисунок 9).

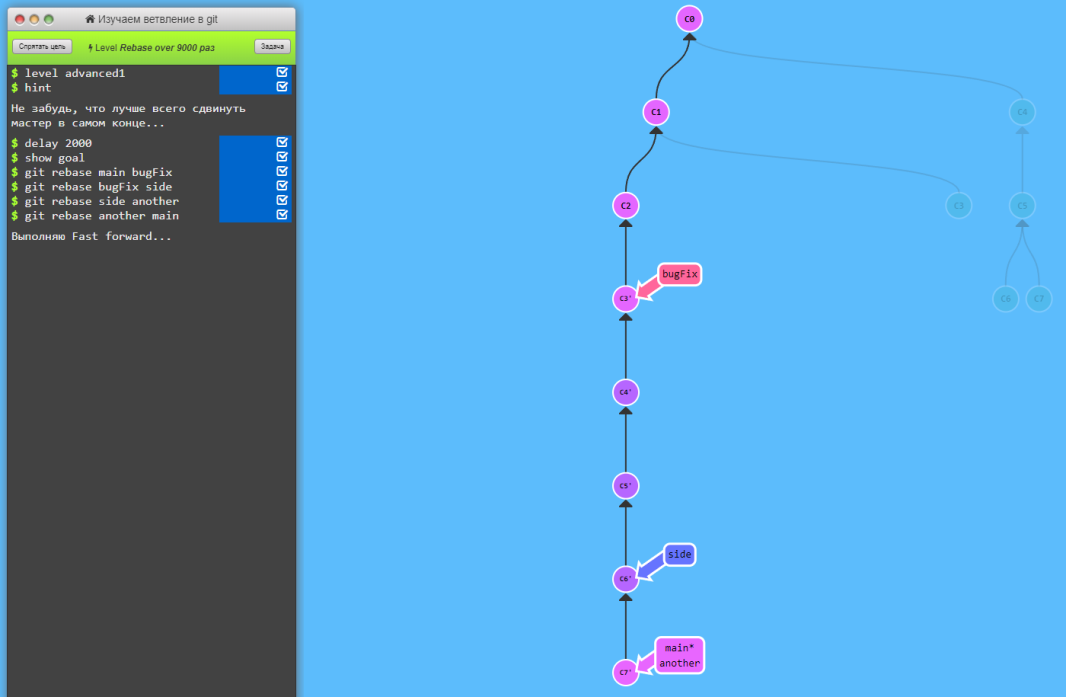


Рисунок 1.9 – Rebase over 9000 раз

Задача №2 расписывает функционал оператора «^», в случае если после оператора следует номер, он указывает на какого из родителей необходимо перейти.

Задача №3 подразумевает распределение различных коммитов из main в некоторые ветки, используем для этого git cherry-pick, git rebase (смотреть рисунок 1.10).

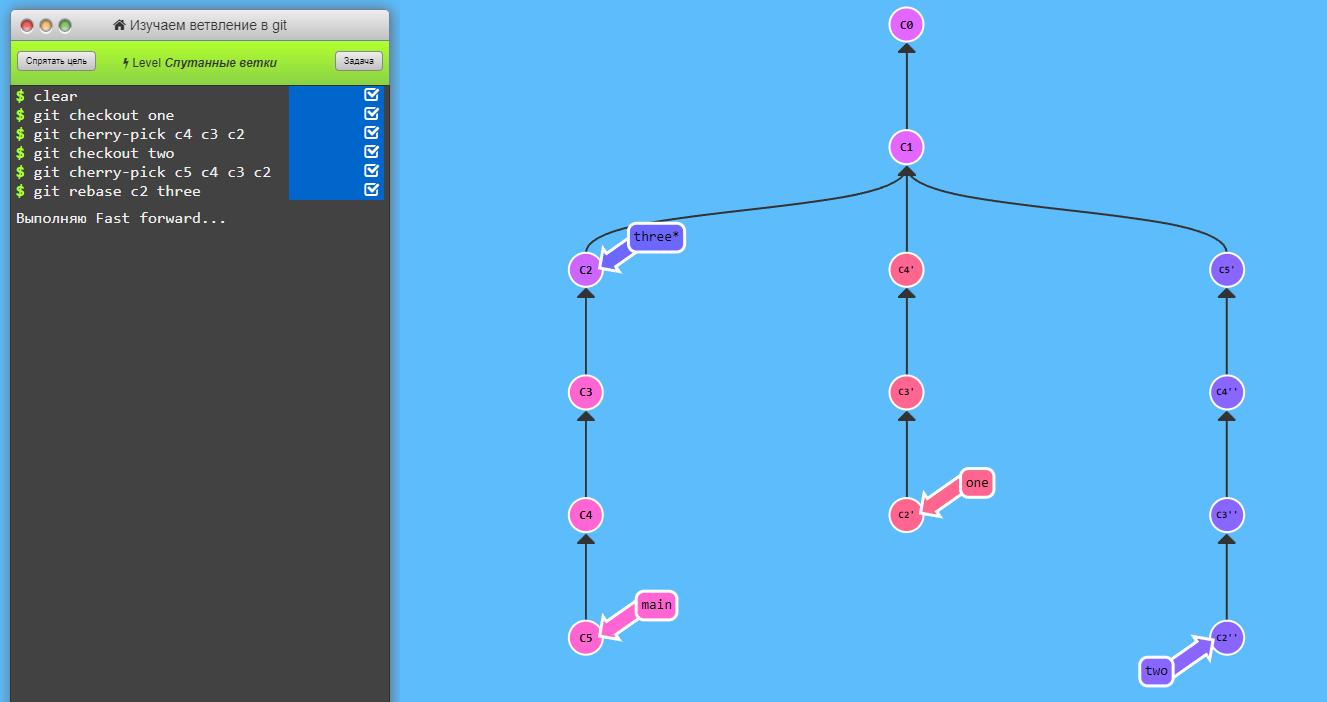


Рисунок 1.10 – Спутанные ветки

1. Удалённые репозитории в Git
2. **Push & Pull - удалённые репозитории в Git**

Задача №1 описывает работу с удаленными репозиториями, а конкретно, выгрузку локального коммита на удалённый сервер.

Задача №2 описывает поведение и особенности работы с удалёнными ветками, а именно, при создании коммита на удалённой ветке origin/main локального репозитория, ветка не переместиться на последний коммит, так как origin/main показывает состояния удалённого репозитория, и обновится тогда, когда обновится локальный репозиторий.

Задача №3 описывает команду Git fetch, которая скачивает данные из удалённого репозитория в локальный, но не сливает коммиты.

Задача №4 обесценивает команду fetch командой pull, что скачивает удалённый репозиторий и сливает (merge) его с локальным. (Смотреть рисунок 2.1)

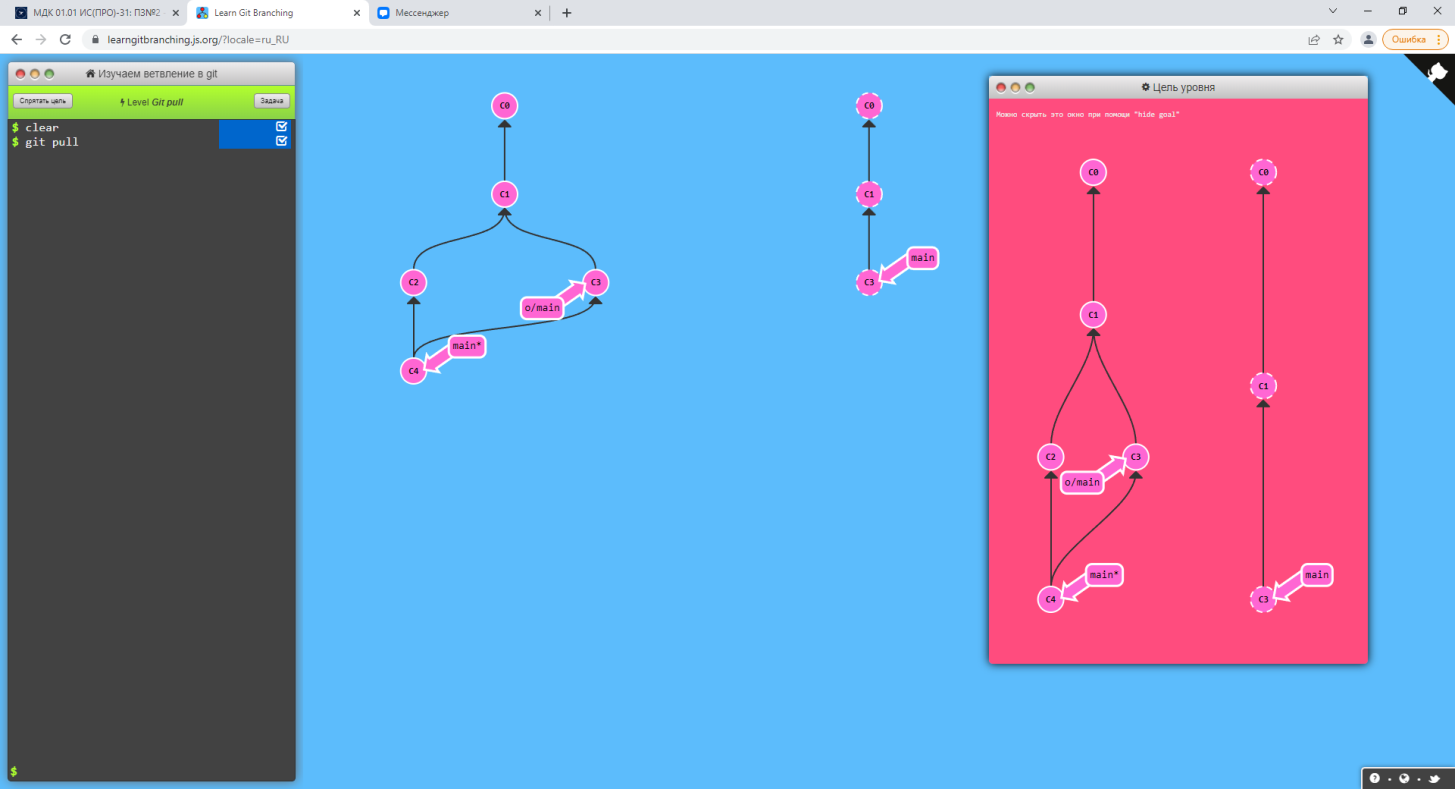


Рисунок 2.1 – Git pull

Задача №5 представляет команды из прошлых задач в “реальной” работе. С помощью команд git clone, git push симулировали работу нескольких людей.(Смотреть рисунок 2.2)

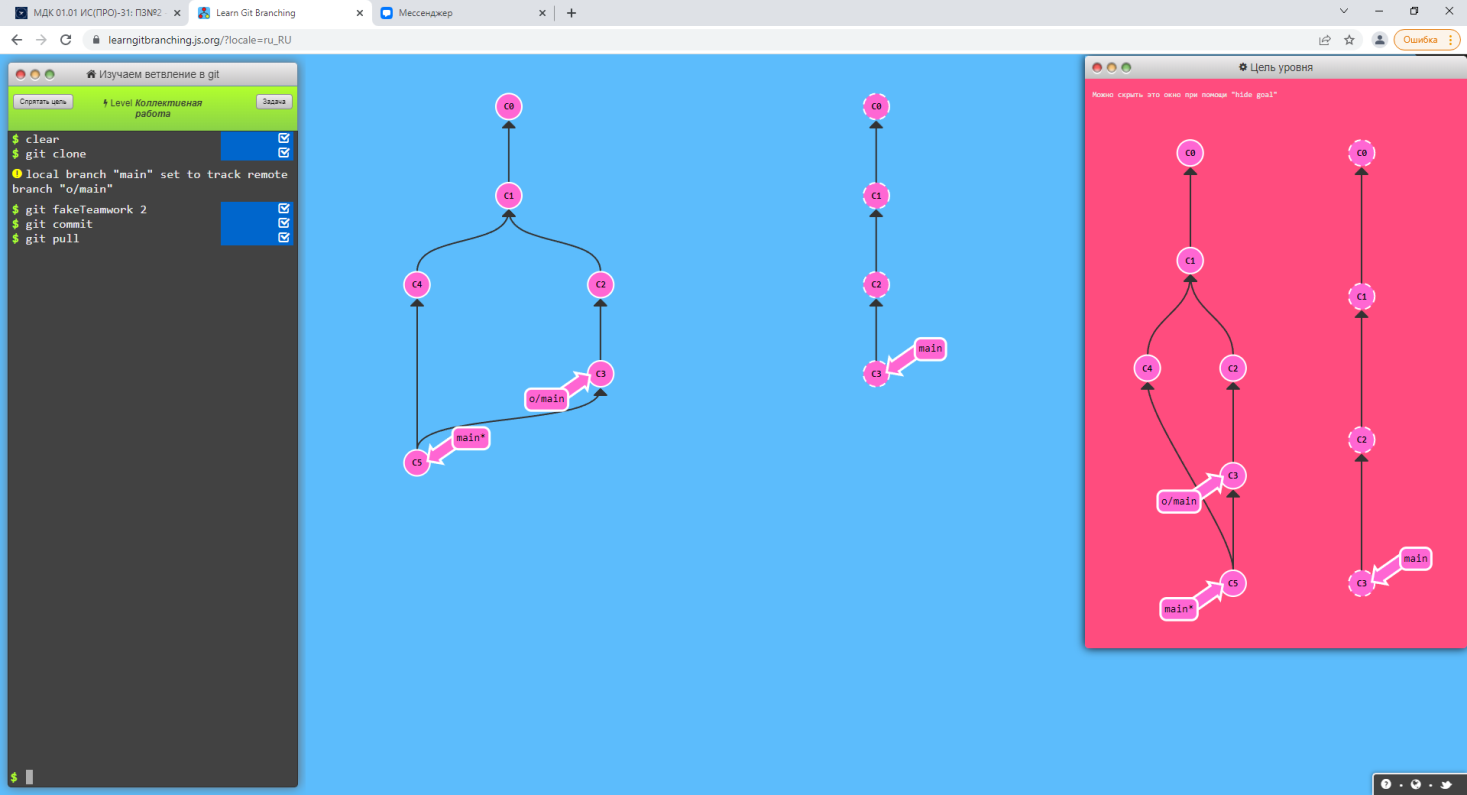


Рисунок 2.2 - Коллективная работа

Задача №6 знакомит пользователя с командой git push – противоположность команды git pull, что позволяет выгрузить локальный репозиторий в удалённый.

Задача №7 ставит проблему расхождения истории коммитов, для решения этой проблемы используется команда git pull –rebase. (Смотреть рисунок 2.3).

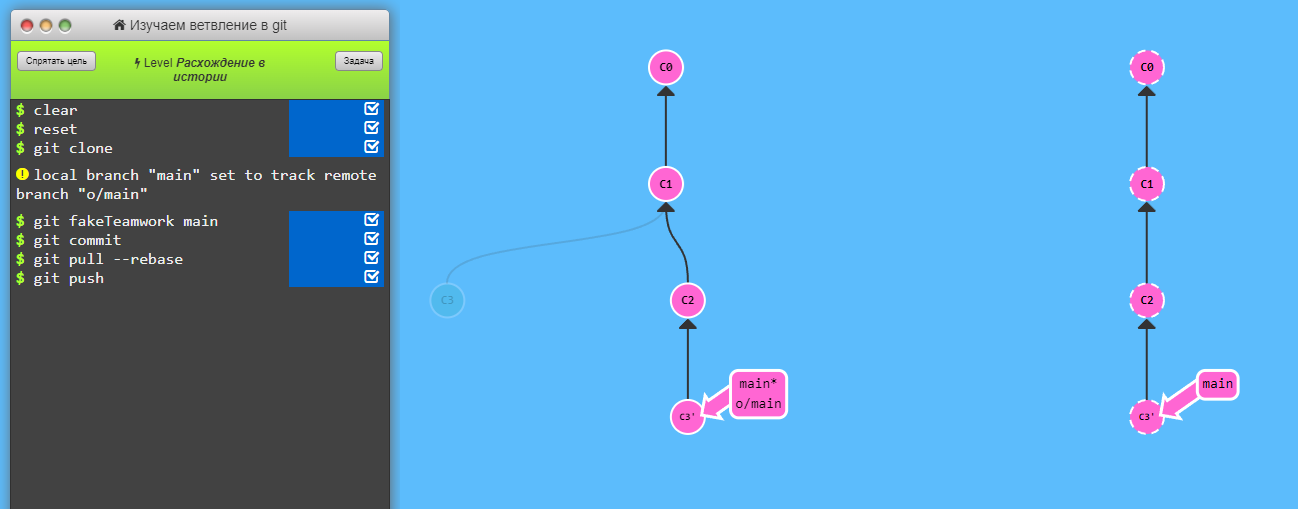


Рисунок 2.3 – Расхождение в истории

Для прохождения задачи №8 необходимо заранее перенести ветку main на коммит C1, создать на коммите С2 ветку feature и запушить изменения, таким образом имитируется ситуация закрытого доступа ветки main. (Смотреть рисунок 2.4).

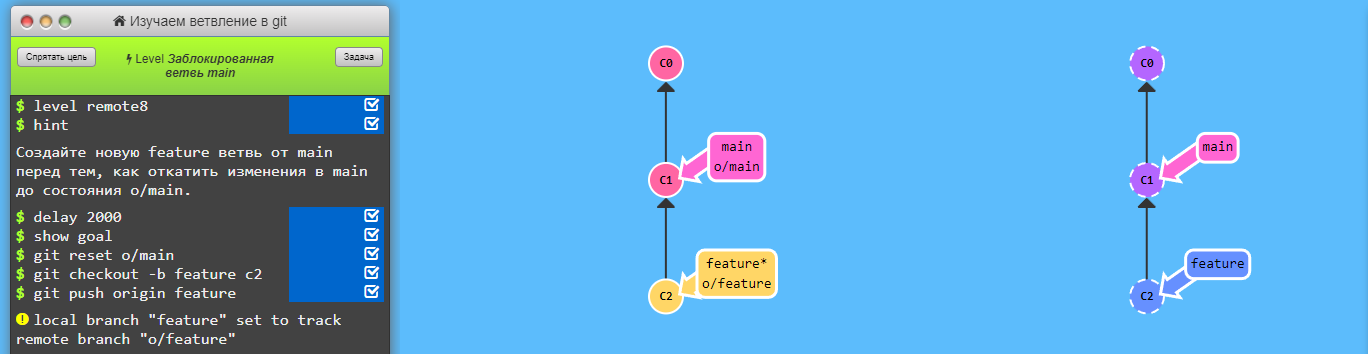


Рисунок 2.4 – Заблокированная ветвь main.

1. Продвинутое использование Git remote

В задаче №1 необходимо перенести коммиты из серии коммитов side на main ветку и запушить на сервер. (Смотреть рисунок 2.5).

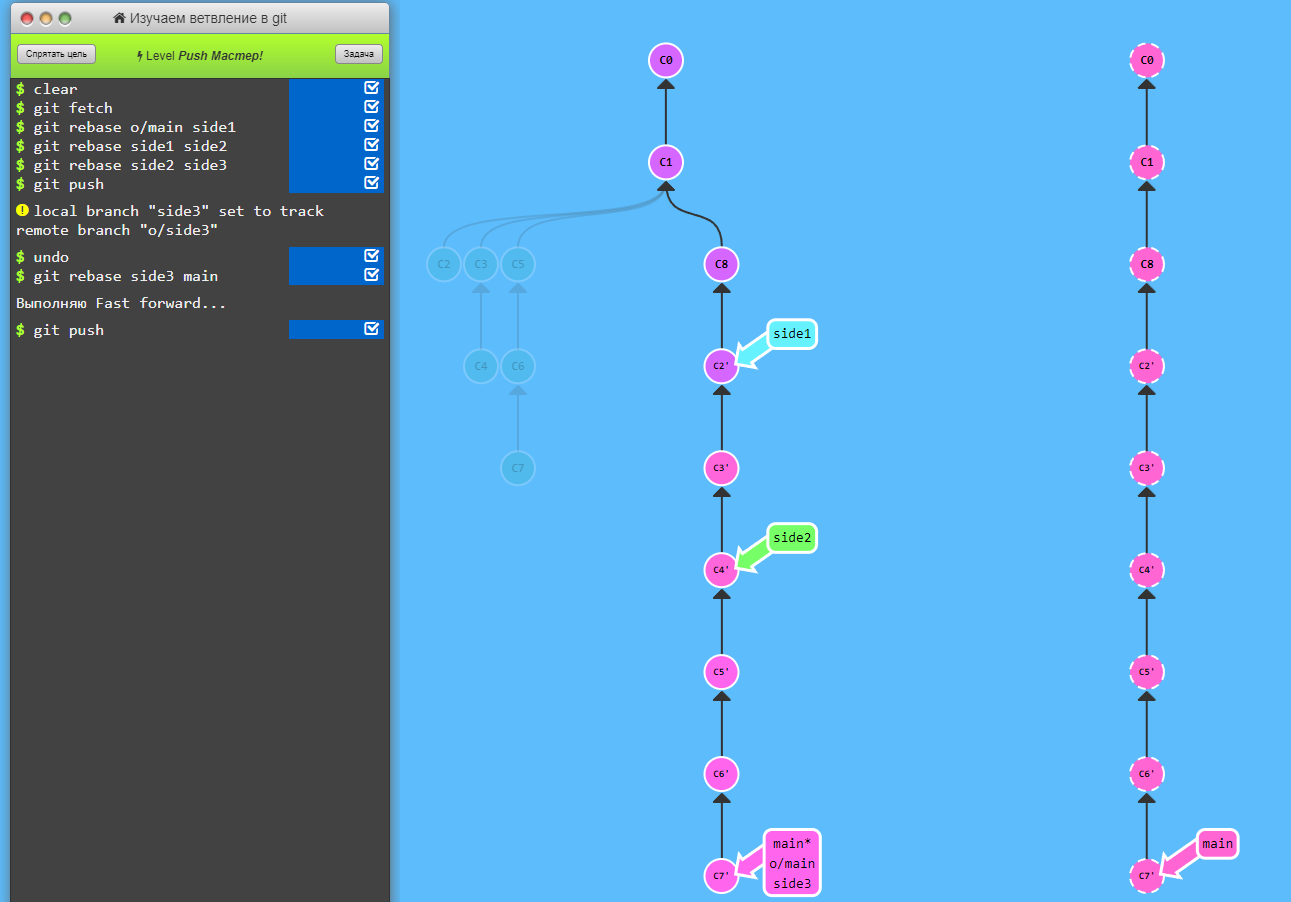


Рисунок 2.5 – Push мастер

Задача №2 показывает разницу между командами merge и rebase в условиях работы с удалённым репозиторием. Необходимо решить проблему из прошлой задачи средствами merge. (Смотреть рисунок 2.6).

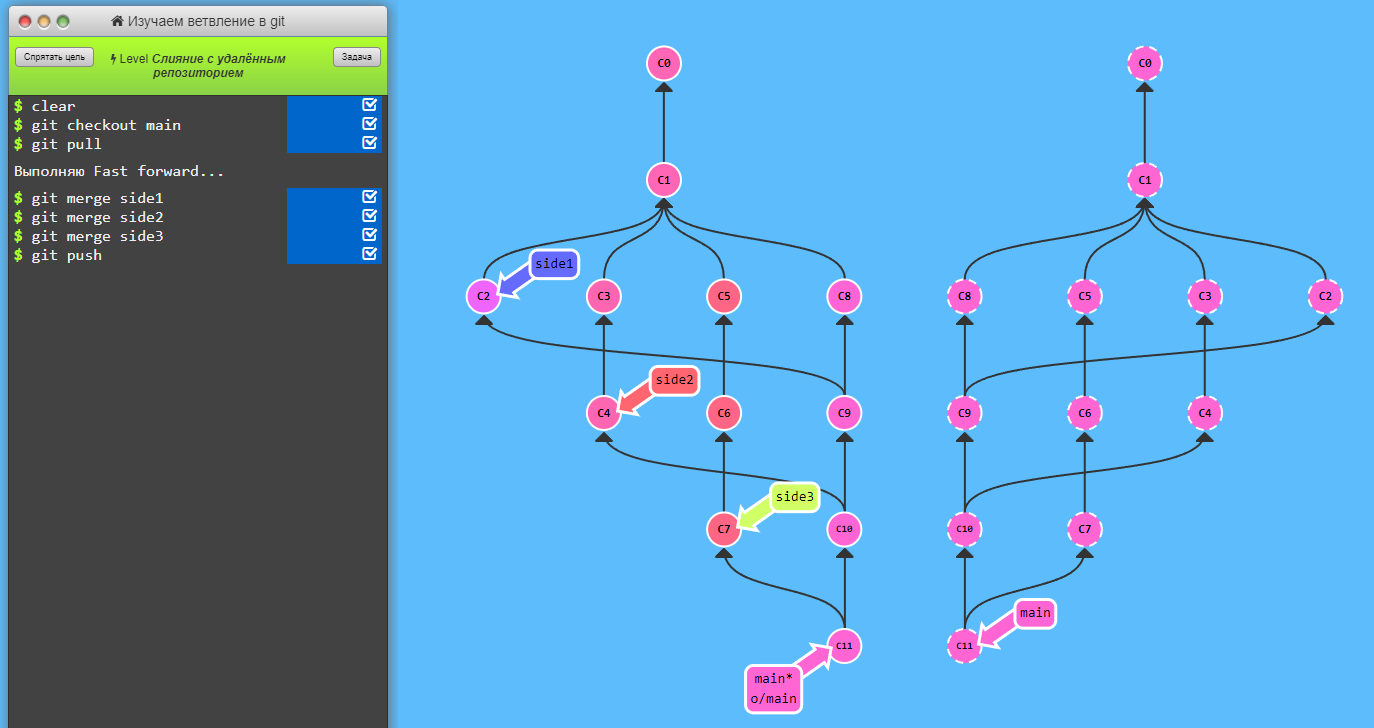


Рисунок 2.6 – Слияние с удалённым репозиторием

Задача №3 знакомит пользователя с отслеживанием удалённых веток, в ходе работы познакомились с атрибутом -U команды branch, что позволяет указать ветке отслеживать удалённую ветку, задание выполнили с помощью checkout -b [branchName] o/main. (Смотреть рисунок 2.7).

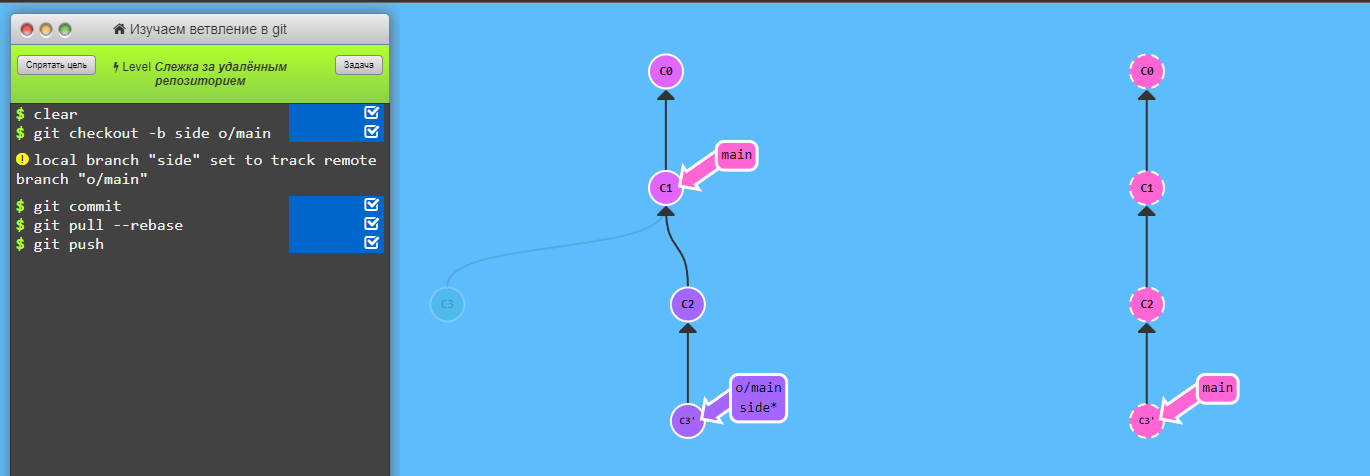


Рисунок 2.7 – Слежка за удалённым репозиторием

В задаче №4 выяснили, что команде git push можно принудительно задать место выгрузки (Смотреть рисунок 2.8)

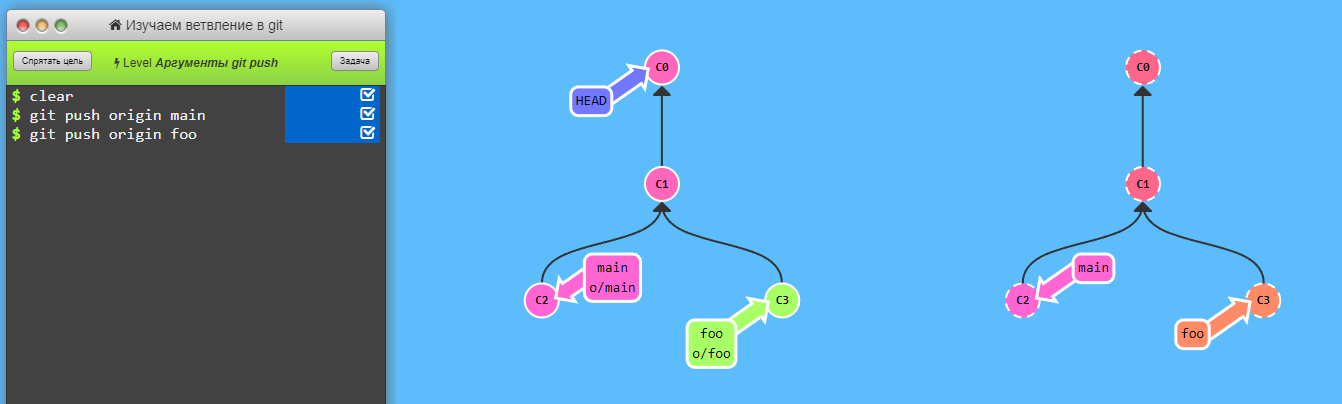


Рисунок 2.8 – Аргументы git push

Задача №5 расширяет поле возможностей команды git push, позволяя указать источник и получатель (Git push origin [Источник]: [получатель]).

Задача №6 знакомит пользователя с аргументами команды git fetch. Синтаксис и поведение команды схоже с git push (Git fetch origin [Источник]: [получатель]).

Задача №7 показывает, что произойдет, если графу источник оставить пустой. В случае push, локальная ветка удалится, так как git пытается поместить туда «ничего», для fetch, же команда с пустым аргументом источник создаст новую ветку.

В задаче №8 требуется скачать некоторые ветки, создать другие и слить одни ветки в другие. Для этого даётся команда git pull и её аргументы, позволяющие указать получателя и отправителя. (Смотреть рисунок 2.9).

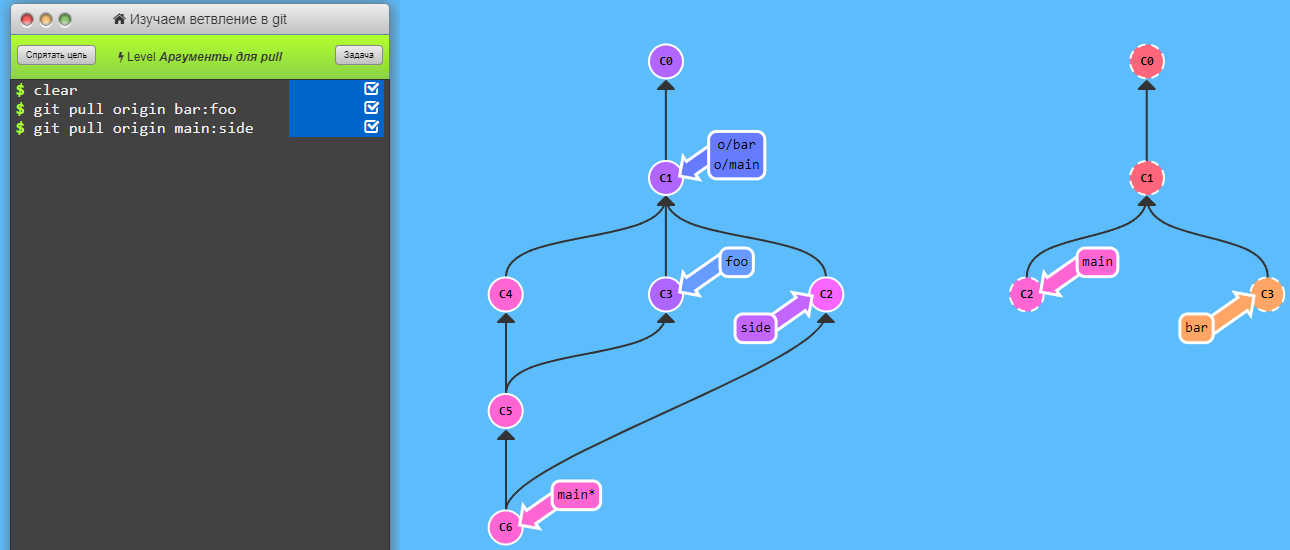


Рисунок 2.9 – Аргументы для pull