ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РЕСПУБЛИКИ ХАКАСИЯ

«ХАКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

Специальность 09.02.07

«Информационные системы и программирование»

Практическое занятие №3

тема: Learn Git Branching

Руководитель:

\_\_\_\_\_\_\_\_\_\_\_\_ Головина Е.А.

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка, дата)

Выполнил:

Студент группы ИС(ВП) – 31

\_\_\_\_\_\_\_\_\_\_\_\_ Рузикулов С.О.

(подпись)

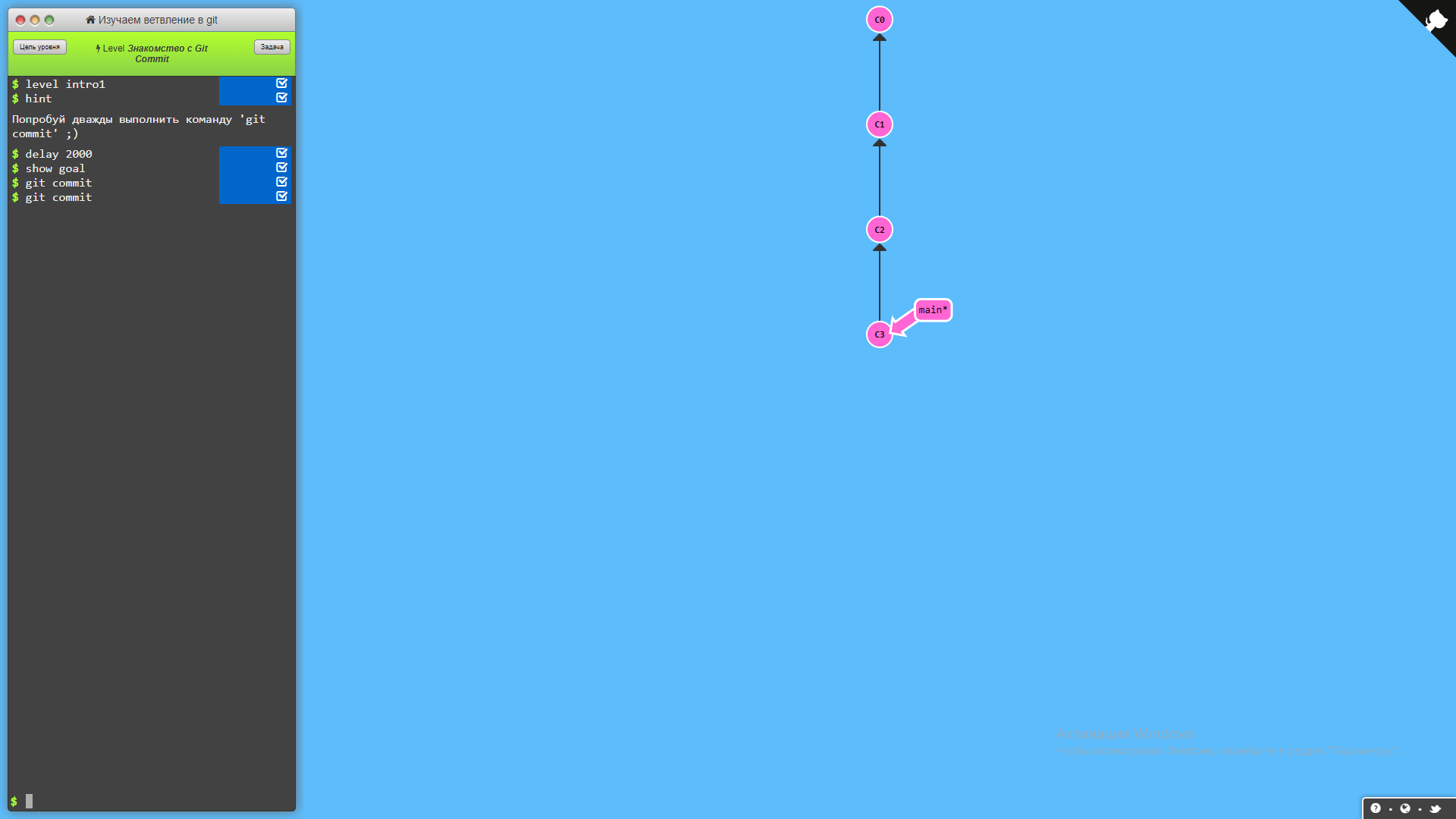
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Абакан 2022 г.

Ссылка на репозиторий GitHub - https://github.com/otcson/pz2.git.

Задание №1. Демонстрирование команды git commit.



Задание №2. Ну что ж, теперь ты готов к работе с ветками. Как только это окно закроется, создай ветку с именем bugFix и переключись на неё.

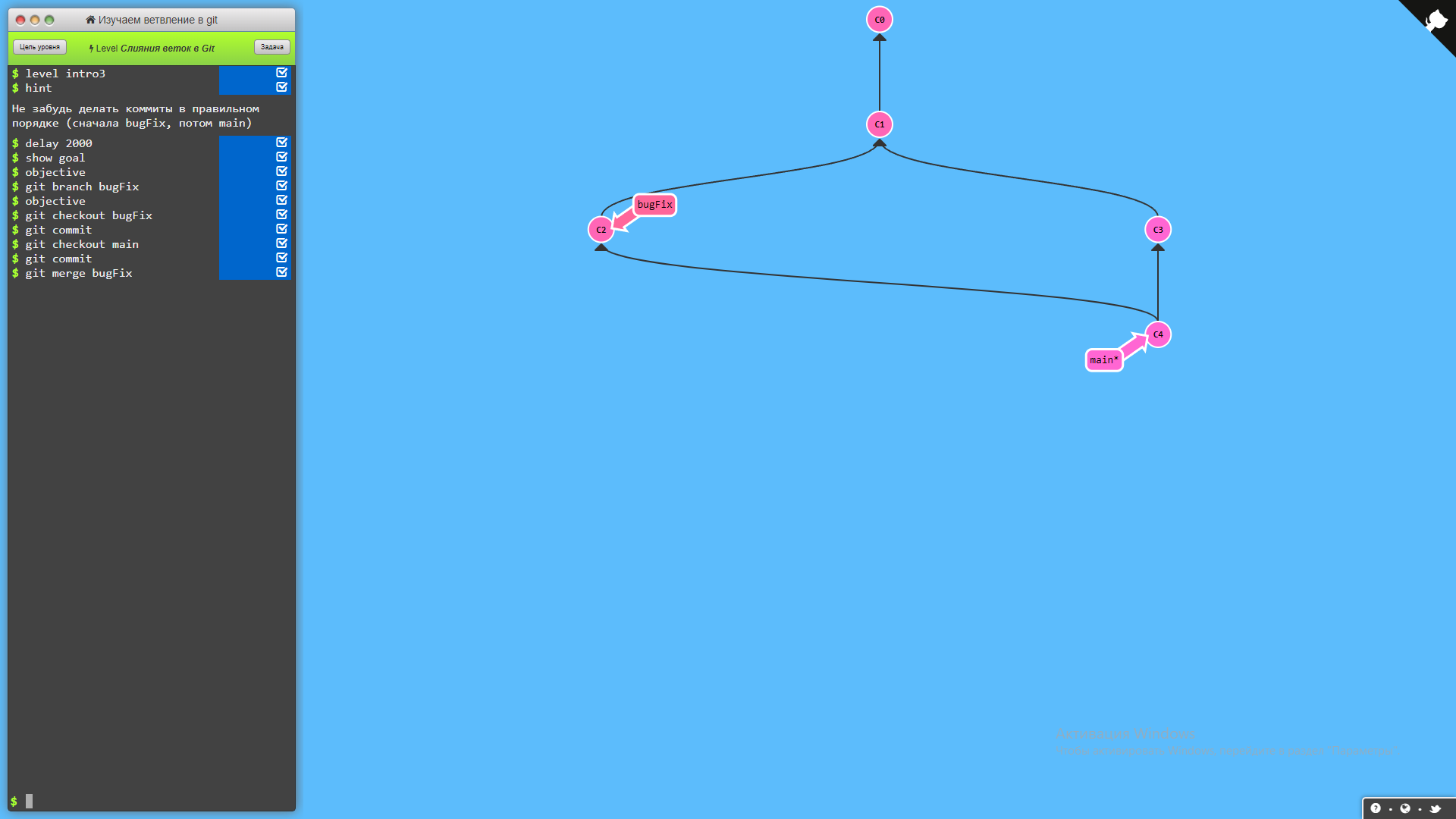
git checkout [name] эта команда перенесёт нас на новую ветку в момент, когда мы ещё не коммитили изменения.

git branch [name] создание новой ветки.



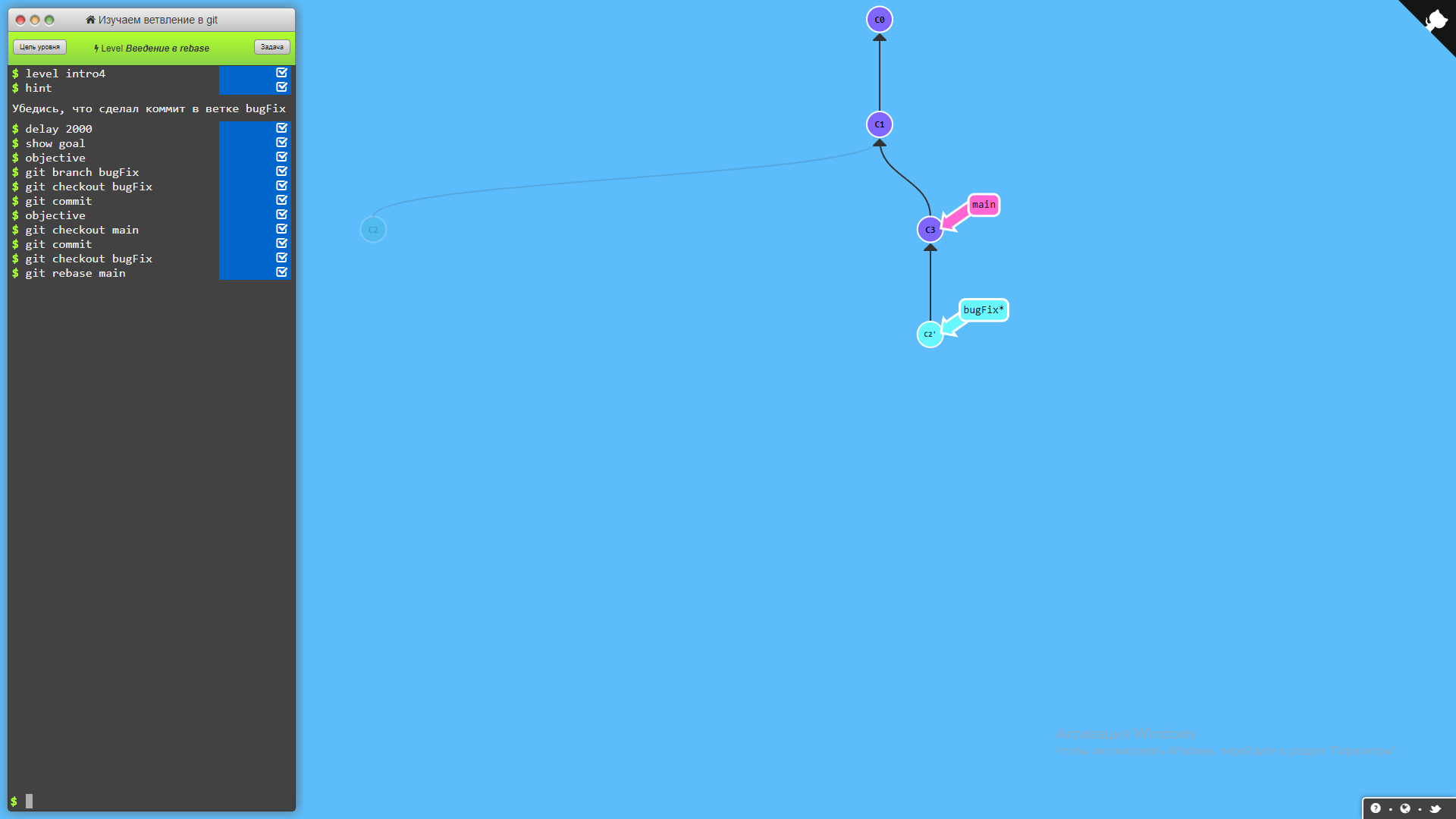
Задание №3.

Первый способ объединения изменений, который мы рассмотрим - это git merge - слияние или просто мердж. Слияния в Git создают особый вид коммита, который имеет сразу двух родителей. Коммит с двумя родителями обычно означает, что мы хотим объединить изменения из одного коммита с другим коммитом и всеми их родительскими коммитами.



Задание №4.

Несмотря на то, что это звучит достаточно непонятно, преимущество rebase в том, что c его помощью можно делать чистые и красивые линейные последовательности коммитов. История коммитов будет чище, если вы применяете rebase.



Задание №5. Рассказывает о HEAD.

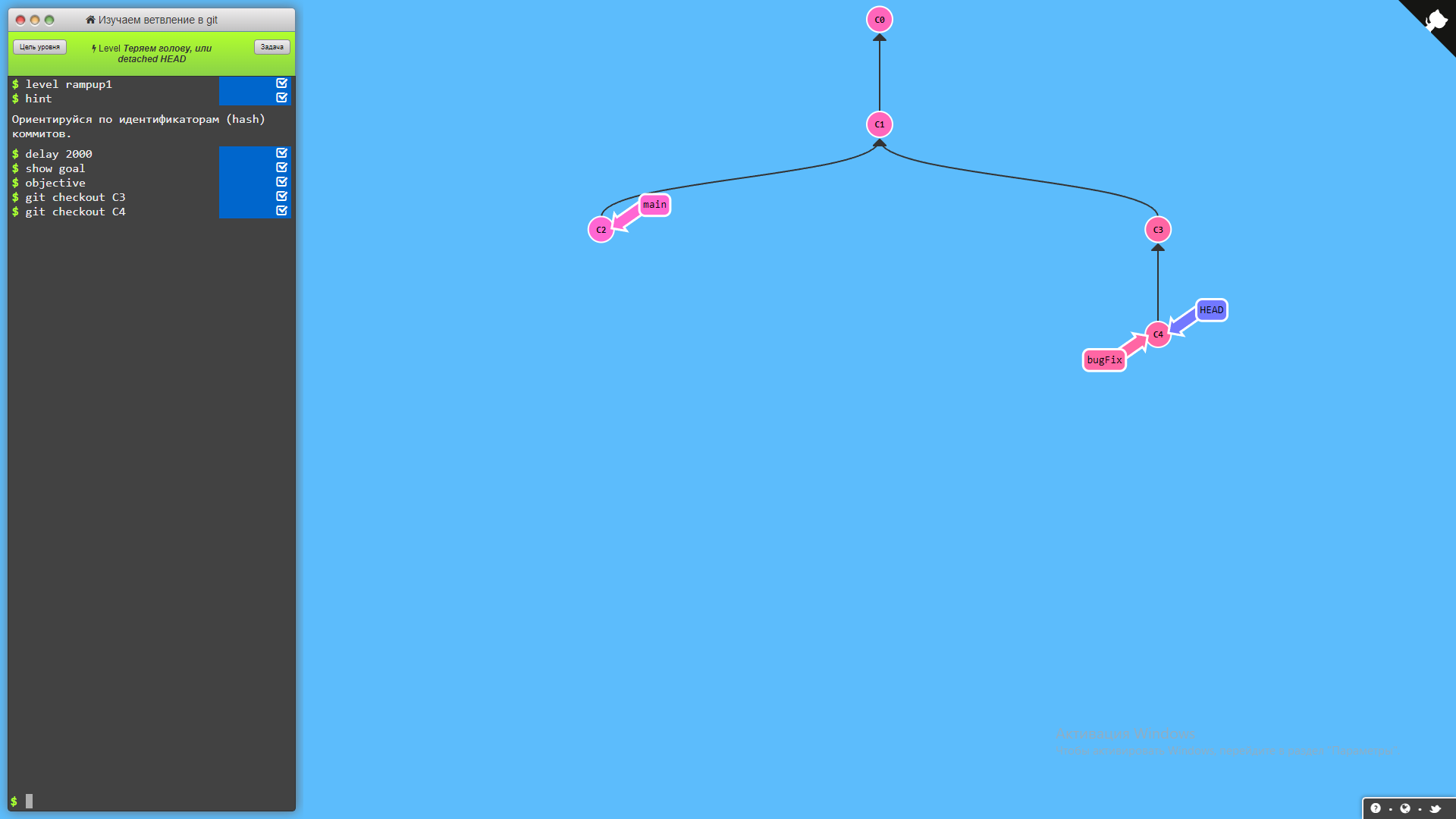
В первую очередь, поговорим о "HEAD". HEAD - это символическое имя текущего выбранного коммита — это, по сути, тот коммит, над которым мы в данным момент работаем.

HEAD всегда указывает на последний коммит из вашего локального дерева. Большинство команд Git, изменяющих рабочее дерево, начнут с изменения HEAD.

Обычно HEAD указывает на имя ветки (например, bugFix). Когда вы делаете коммит, статус ветки bugFix меняется и это изменение видно через HEAD.

Detaching HEAD

Отделение (detaching) HEAD означает лишь присвоение его не ветке, а конкретному коммиту. Посмотрим, что было до отделения:



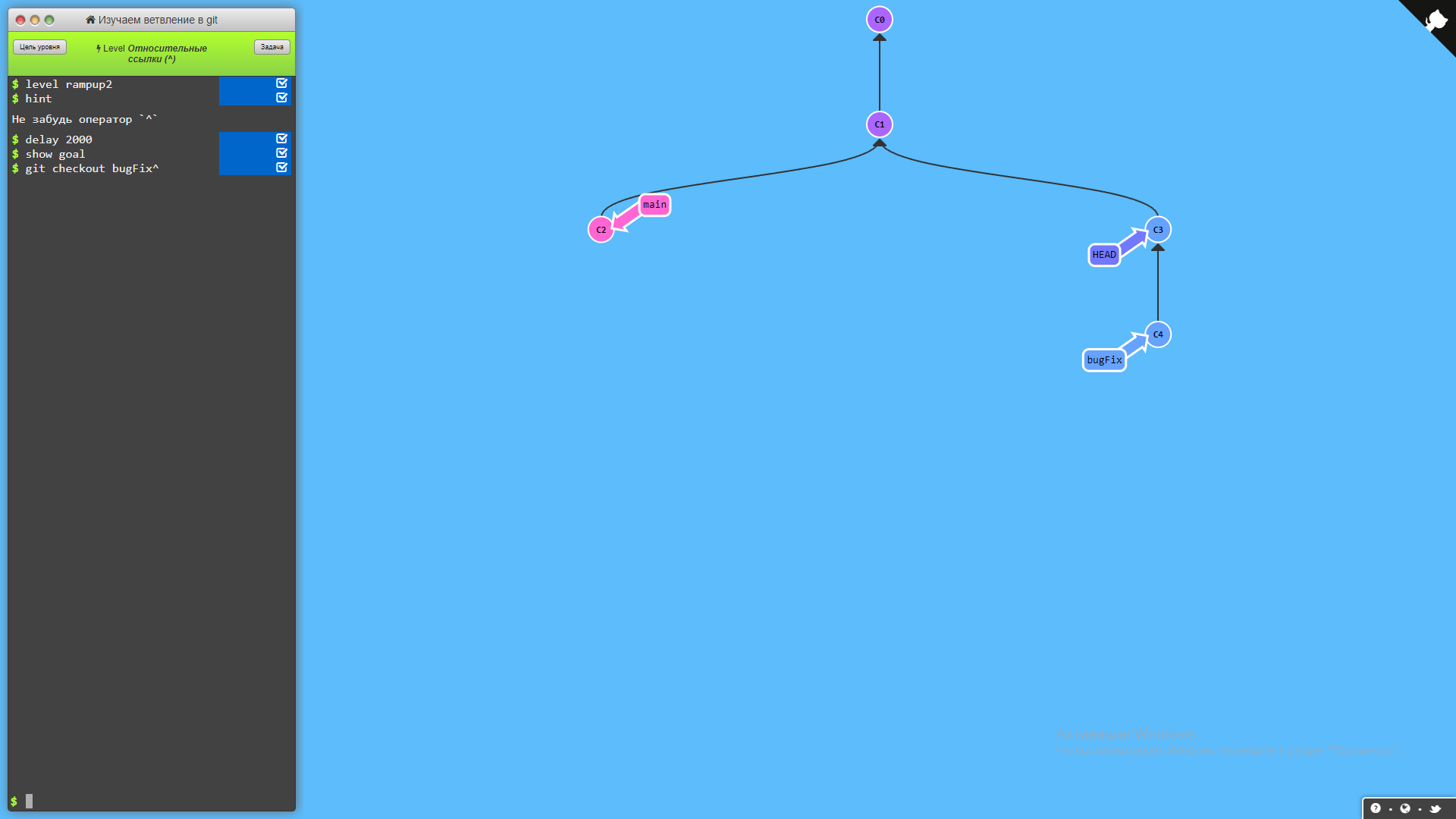
Задание №6.

git log, чтобы найти хеш нужного коммита

Относительные ссылки - мощный инструмент, но мы покажем два простых способа использования:

Перемещение на один коммит назад ^

Перемещение на несколько коммитов назад ~<num>



Задание 7.

Оператор "~"

Предположим, нужно переместиться на много шагов назад по дереву. Было бы неудобно печатать ^ несколько раз (или несколько десятков раз), так что Git поддерживает также оператор тильда (~).

К тильде (опционально) можно добавить количество родительских коммитов, через которые нужно пройти. Посмотрим, как это работает.

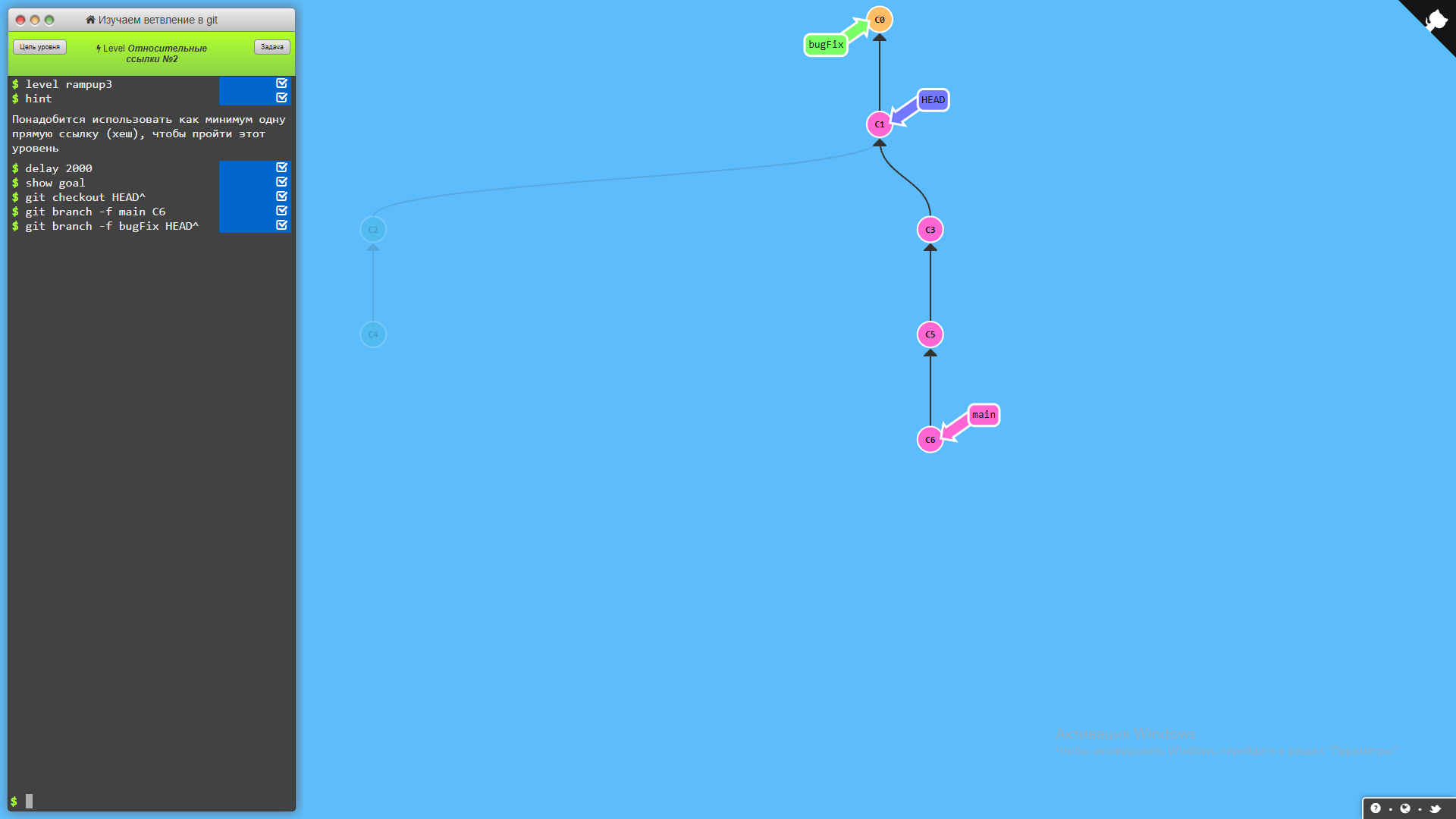
Перемещение ветки (branch forcing)

Теперь мы разбираемся в относительных ссылках, так что можно реально использовать их для дела.

Одна из наиболее распространённых целей, для которых используются относительные ссылки - это перемещение веток. Можно напрямую прикрепить ветку к коммиту при помощи опции -f.

git branch -f main HEAD~3

Переместит (принудительно) ветку main на три родителя назад от HEAD.



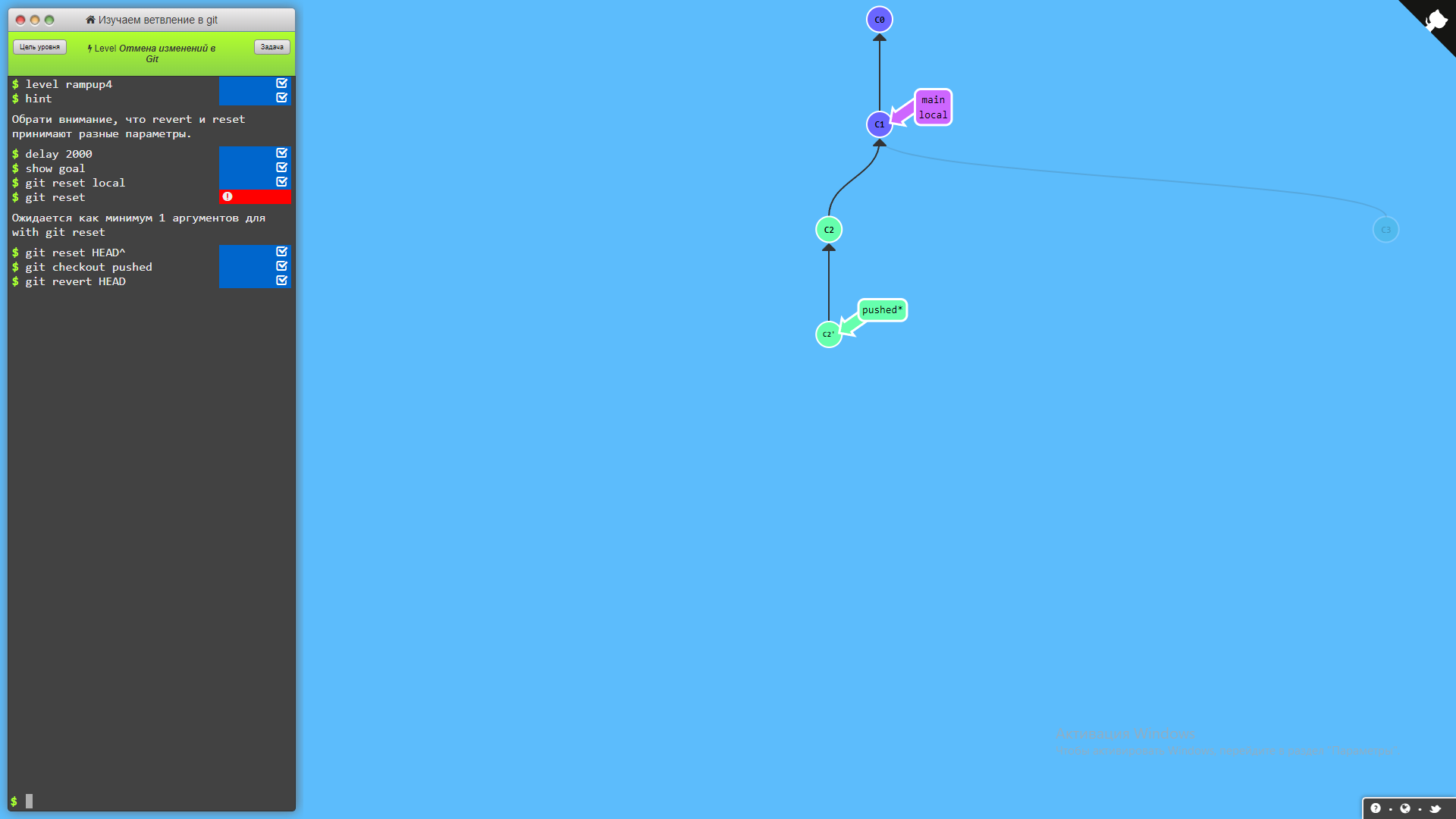
Задание №8.

Есть два основных способа отмены изменений в Git: первый - это git reset, а второй - git revert. Попробуем оба на следующем шаге.

git reset отменяет изменения, перенося ссылку на ветку назад, на более старый коммит. Это своего рода "переписывание истории"; git reset перенесёт ветку назад, как будто некоторых коммитов вовсе и не было.

Reset отлично работает на локальных ветках, в локальных репозиториях. Но этот метод переписывания истории не сработает на удалённых ветках, которые используют другие пользователи.

Чтобы отменить изменения и поделиться отменёнными изменениями с остальными, надо использовать git revert. Посмотрим, как это работает

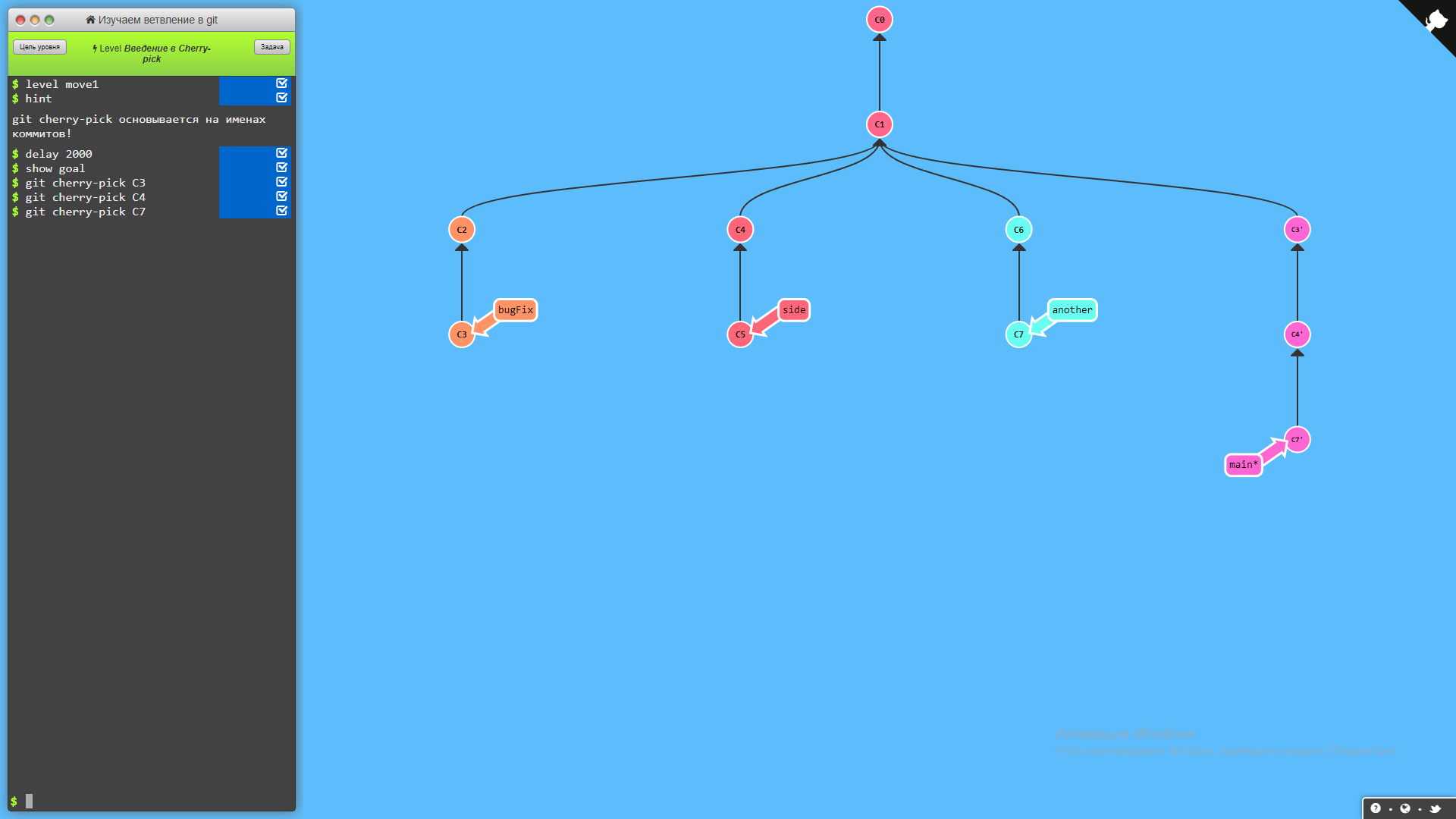


Задание №9.

Первая из таких команд - это git cherry-pick. Она выглядит вот так:

git cherry-pick <Commit1> <Commit2> <...>

Это очень простой и прямолинейный способ сказать, что ты хочешь копировать несколько коммитов на место, где сейчас находишься (HEAD). Мы обожаем cherry-pick за то, что в нём очень мало магии и его очень просто понять и применять.



Задание №10.

Всё, что нужно для интерактивного rebase - это опция -i

Если добавить эту опцию, Git откроет интерфейс просмотра того, какие коммиты готовы к копированию на цель rebase (target). Также показываются хеши коммитов и комментарии к ним, так что можно легко понять что к чему.

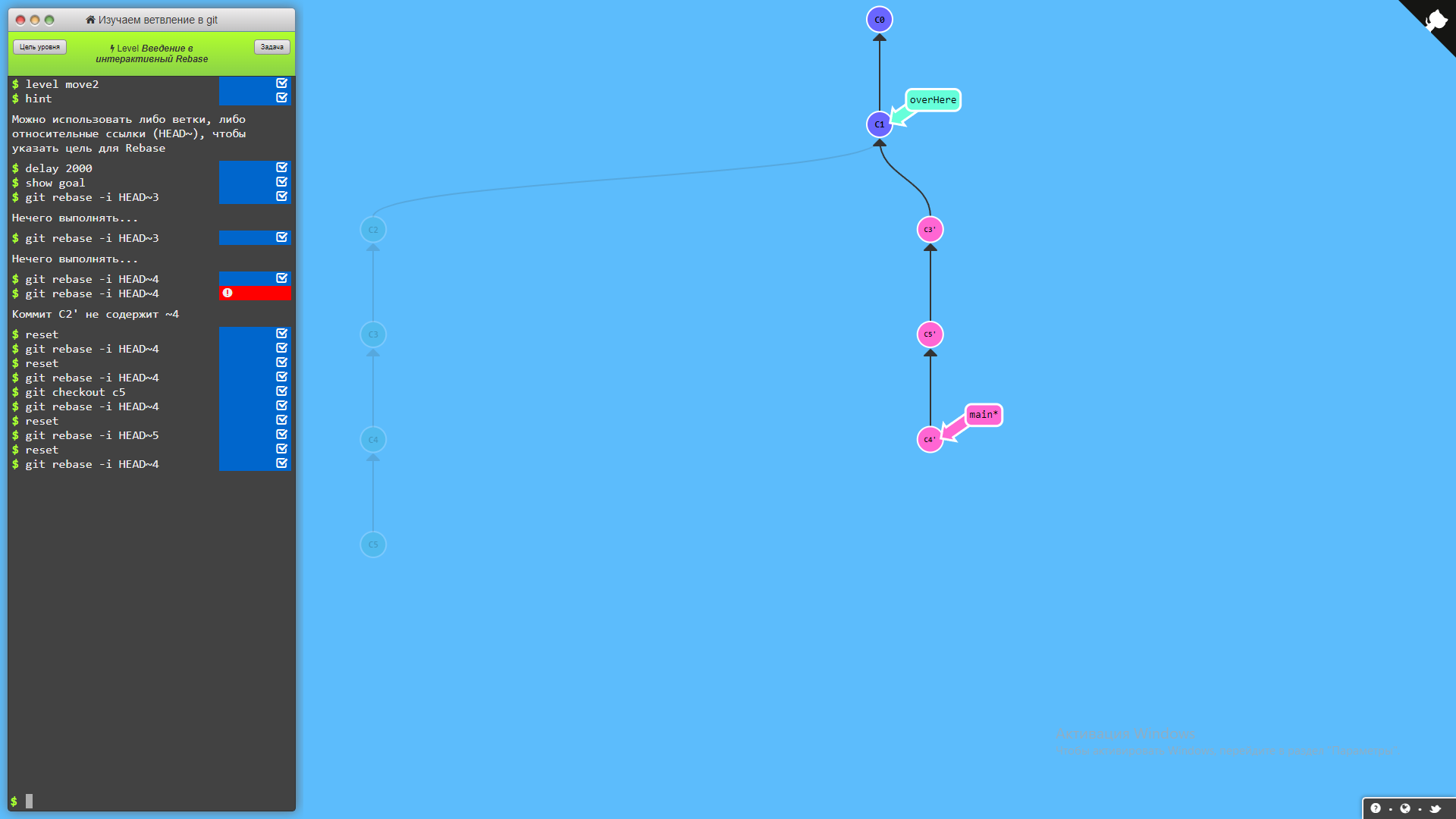
Для "реального" Git, этот интерфейс означает просто открытие файла в редакторе типа vim. Для этой обучалки, я сделал небольшое диалоговое окно, которое по сути делает то же, что и редактор.

После открытия окна интерактивного rebase есть три варианта для каждого коммита:

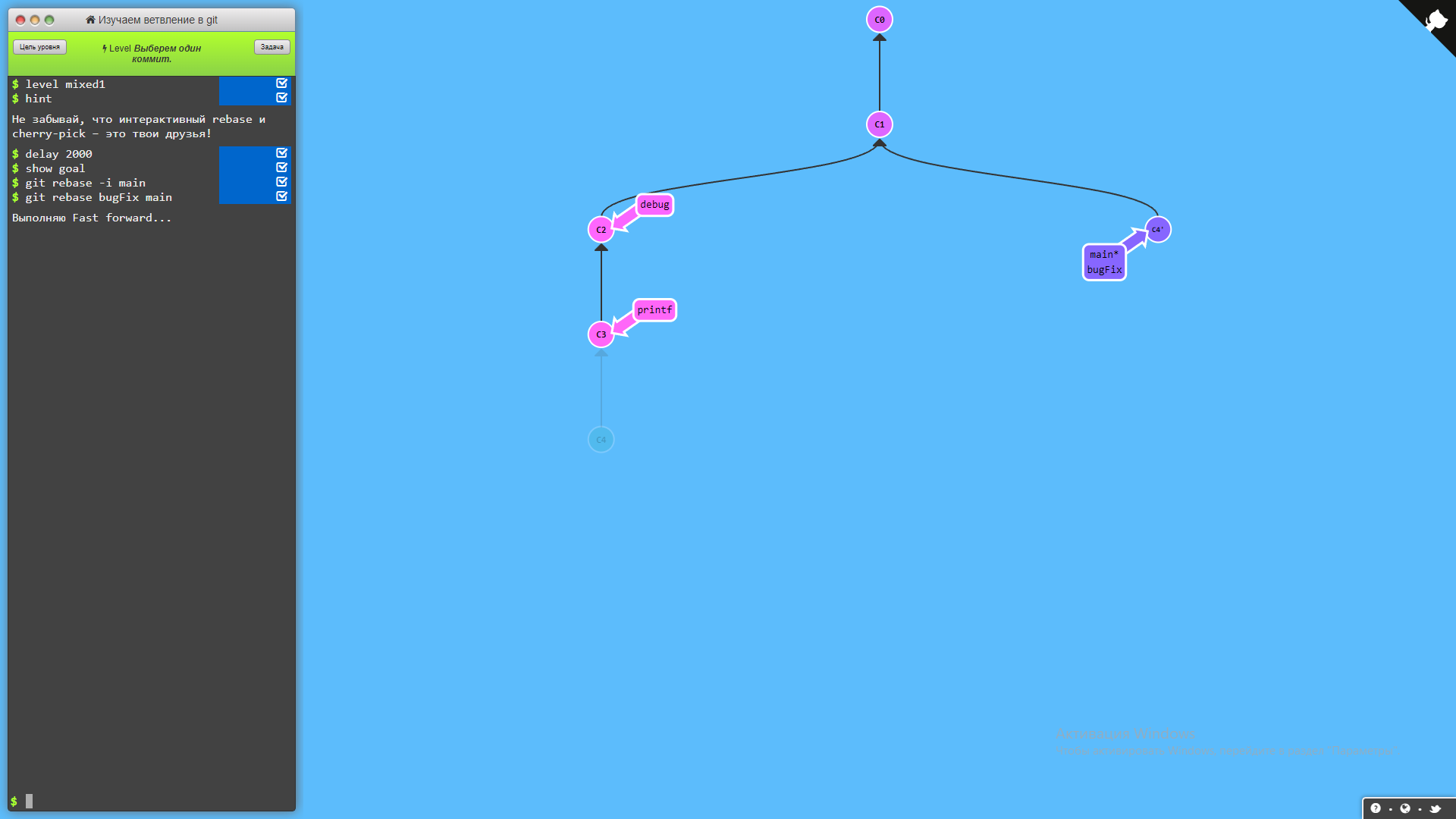
Можно сменить положение коммита по порядку, переставив строчку с ним в редакторе (у нас в окошке строку с коммитом можно перенести просто мышкой).

Можно "выкинуть" коммит из ребейза. Для этого есть pick - переключение его означает, что нужно выкинуть коммит.

Наконец, можно соединить коммиты. В этом уровне игры у нас не реализована эта возможность, но, вкратце, при помощи этой функции можно объединять изменения двух коммитов.



Задание 11. В этом уровне тебе решать, какую команду использовать, но чтобы закончить уровень, убедись, что в мастер попал коммит, на который ссылается bugFix.



Задание 12. Обрати внимание на итоговое состояние в этом уровне – так как мы дважды перемещаем коммиты, оба они получат по апострофу. Ещё один апостроф добавляется, когда мы делаем git commit --amend.

Задание 13.

