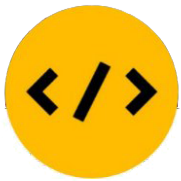
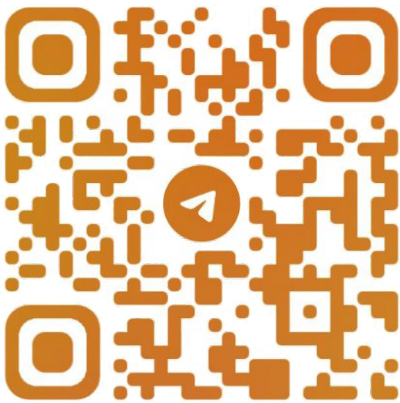
Изучаем Git

Пошаговое руководство с наглядными примерами

Learning Git

A Hands-On and Visual Guide to the Basics of Git





Изучаем Git

Пошаговое руководство
с наглядными примерами

В простой, осязаемой и наглядной форме книга знакомит с Git — самой популярной системой контроля версий. Изложены основы Git: установка, графический интерфейс и командная строка, локальные репозитории и коммиты, ветки и слияния. Рассмотрены хостинговые сервисы и аутентификация, работа с удаленным репозиторием, клонирование и локальное сохранение. Подробно рассмотрены трехсторонние слияния, конфликты слияния, запросы на слияние, а также интеграция изменений из одной ветки в другую с помощью перебазирования. Для закрепления материала служат два сквозных проекта: учебный, демонстрирующий основы, и практический, применимый для реальной работы.

Для начинающих осваивать Git

Содержание

Предисловие

Я никогда не думала, что напишу книгу, обучающую работе с Git. Но благодаря цепочке удачных событий у меня появилась творческая идея о том, как доходчиво и просто преподавать эту технологию.

Мой путь к написанию книги начался с посещения учебных курсов по программированию, где студенты изучали веб-разработку. Преподаватели кратко знакомили студентов с Git, но, учитывая, что все наши проекты выполнялись индивидуально, я не нашла существенного применения этим знаниям.

После обучения программированию я устроилась фронтенд-разработчиком в команду поддержки сайта крупной компании. Настоящее знакомство с Git началось с первого дня моей новой работы. В те первые месяцы, работая в команде крупной компании, я поняла, что меня это пугает. Каждый раз, когда мне приходилось делать с помощью Git что-то, вызывающее хоть малейшие затруднения, я думала, что уничтожу репозиторий или причиню другой серьезный ущерб.

Для того чтобы избавиться от страха навредить коллегам, я решила изучить все тонкости Git. Но после знакомства с многочисленными онлайн-учебниками мне быстро стало ясно, что бóльшая часть материалов не предназначена для таких новичков, как я. Как только я поняла основы, в моем сознании начала формироваться идея о том, как преподавать эту технологию намного проще и понятнее, используя наглядные диаграммы и цвета.

В итоге я создала онлайн-курс и опубликовала его в Интернете. Во время работы над курсом в глубине души я надеялась, что когда-нибудь смогу написать об этом книгу.

Я получила много положительных отзывов о курсе и, наконец, летом 2021 г. решила, что пора исполнить свою мечту. Книга, которую вы сейчас читаете, появилась в результате этого решения, и я надеюсь, что она поможет вам в увлекательном путешествии по миру Git!

Для кого эта книга

Эта книга предназначена для всех, кто хочет научиться работать с Git. Она специально написана для людей, которые только начинают получать технические знания или не являются техническими специалистами, но вынуждены использовать Git для совместной работы со своими коллегами. К читателям, которым будет полезна эта

книга, относятся учащиеся курсов по программированию, студенты, изучающие информатику, технические писатели, менеджеры по продуктам, дизайнеры, младшие разработчики, специалисты по обработке данных, программисты-самоучки и многие другие.

Книга адресована как дилетантам, совсем не имеющим опыта использования Git, так и тем, у кого есть небольшой опыт. Если у вас нет опыта работы с Git, это не проблема. Мы начнем с установки Git и использования командной строки, и на этом фундаменте построим здание по кирпичику.

Если у вас уже есть некоторый опыт использования Git или командной строки, можете бегло пролистать первую главу и двигаться дальше. Но я советую не пропускать ее, поскольку она закладывает основу учебного проекта Rainbow, с которым вы будете работать на протяжении всей оставшейся части книги.

Как читать эту книгу

Эта книга основана на практическом обучении, в ходе которого вы будете выполнять упражнения на своем компьютере и изучать основные концепции Git. При чтении книги вы встретите два проекта: Rainbow (Радуга) и Book (Книга).

Rainbow — это практический проект, над которым вы будете работать, выполняя упражнения, описанные в книге. Он очень простой и предназначен только для учебных целей. В свою очередь, Book — это воображаемый проект, который я буду использовать, чтобы продемонстрировать, как определенные функции Git применяются в реальных ситуациях. Давайте подробнее рассмотрим каждый проект, а также структуру книги.

> **ПРИМЕЧАНИЕ**
>
> Не беспокойтесь, если встретите в предисловии незнакомые термины, такие как *"репозиторий"* или *"коммит"*. В следующих главах вы получите все необходимые пояснения.

Проект Rainbow

Для того чтобы изучить основы Git, на протяжении всей этой книги вы будете работать над проектом Rainbow. Вы должны последовательно читать *главы 1–12* и выполнять все упражнения на своем компьютере. Например, упражнения *главы 4* предполагают, что вы уже выполнили упражнения *глав 1–3*.

Репозитории

Если говорить упрощенно, *репозиторий* — это копия проекта Git. Сначала вы создадите локальный репозиторий под названием `rainbow` для работы над проектом

Rainbow. Впоследствии (в *главе 7*) вы создадите удаленный репозиторий под названием `rainbow-remote`. И, наконец, в *главе 8* вы смоделируете совместную работу с другом над проектом Rainbow и создадите второй локальный репозиторий под названием `friend-rainbow`. Начиная с *главы 8*, всякий раз, когда в книге говорится, что ваш воображаемый "друг" что-то делает, вам придется выполнить за него действие в репозитории `friend-rainbow`.

> **ПРИМЕЧАНИЕ**
>
> Когда я говорю о проекте Rainbow с большой буквы R, я имею в виду весь проект, который начинается с одного репозитория и в конце содержит три репозитория. Когда я говорю о каталоге проекта `rainbow` или репозитории `rainbow` (с маленькой буквы `r`), я имею в виду конкретный локальный репозиторий, который является частью проекта Rainbow[1].

Коммиты

В проекте Rainbow вам предстоит создавать и редактировать файлы с названиями цветов радуги, а также некоторых цветов, не являющихся частью радуги. Это не реальный проект, отслеживаемый с помощью Git, а упрощенный пример, который позволяет вам сосредоточиться на обучении, не отвлекаясь на цель проекта.

На протяжении всей книги я буду использовать диаграммы, чтобы наглядно показать, что происходит в проекте Rainbow. Каждый раз, когда вы добавляете цвет в проект Rainbow, вы делаете *коммит* (commit) в репозитории. Коммит по сути представляет собой версию вашего проекта. На схемах коммит будет представлен кружком добавленного вами цвета, а в качестве имени коммита мы также будем использовать название цвета. Например, первый цвет, который вы добавите в свой проект Rainbow, — красный (red), поэтому кружок, представляющий этот коммит, будет окрашен в красный цвет.

Для того чтобы сделать книгу доступной читателям с нарушением цветового зрения, я включаю в схемы название коммита (или аббревиатуру названия). На рис. П1 изображен пример красного (red) коммита.

Рис. П1. Пример коммита с его полным названием

Таблица П1 содержит список всех коммитов, которые вы сделаете в проекте Rainbow в этой книге, с указанием их полных названий и сокращений.

---

[1] Для того чтобы избежать путаницы, мы пишем имя проекта Rainbow обычным шрифтом, а имя репозитория и одновременно каталога проекта `rainbow` — моноширинным шрифтом. — *Прим. перев.*

Таблица П1. *Полный список коммитов, сделанных в проекте Rainbow*

Полное название коммита	Сокращение
red (красный)	R
orange (оранжевый)	O
yellow (желтый)	Y
green (зеленый)	G
blue (голубой)	B
brown (коричневый)	Br
merge commit 1 (коммит слияния 1)	M1
indigo (синий)	I
violet (фиолетовый)	V
merge commit 2 (коммит слияния 2)	M2
gray (серый)	Gr
black (черный)	Bl
rainbow (радужный)	Ra
pink (розовый)	P
merge commit 3 (коммит слияния 3)	M3

На рис. П2 изображена полная схема связей всех коммитов.

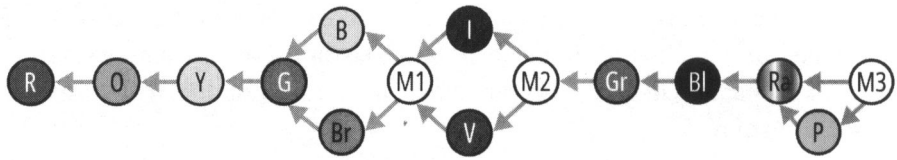


Рис. П2. 15 коммитов, которые вы сделаете в проекте Rainbow к концу этой книги

Приложения

Хотя книга предназначена для последовательного чтения от *главы 1* до *главы 12*, иногда могут возникнуть ситуации, когда вам захочется или потребуется начать с определенной главы. Например:

- вы выполнили упражнения всей книги один раз и хотите повторить их, начиная с конкретной главы;
- что-то пошло не так в проекте Rainbow в предыдущей главе, вы не смогли устранить проблему и хотите начать работу с главой заново.

В подобных случаях вы можете воспользоваться инструкциями в *приложении 1*, чтобы вручную привести проект Rainbow к необходимому состоянию в начале главы, с которой вы хотите начать.

Приложение 2 содержит краткое справочное руководство по командам, представленным в каждой главе. *Приложение 3* представляет собой руководство по визуальному языку, который применялся для построения схем в этой книге.

Проект Book

Проект Book — это воображаемый пример, который нужен для демонстрации того, как Git применяется в реальной жизни. В этом проекте мы представим, будто я пишу книгу и хочу использовать Git для контроля версий файлов. Книга будет состоять из 10 глав, представленных 10 текстовыми файлами, по одному на каждую главу: chapter_one.txt, chapter_two.txt и т. д. Время от времени я также моделирую работу над проектом Book в роли соавтора и/или редактора. Описания примеров работы с Git будут представлены в *разделах "Пример проекта Book"* соответствующих глав.

Вам не придется активно работать над проектом Book или развивать его. Он нужен только для демонстрации дополнительных примеров и описаний применения определенных функций Git.

Помимо *разделов "Пример проекта Book"* в книге вы также встретите некоторые другие разделы. Давайте кратко ознакомимся с ними.

Разделы книги

Вот краткий перечень различных разделов, которые вы встретите в этой книге.

Пример проекта Book

Как упоминалось ранее, *разделы "Пример проекта Book"* знакомят читателя с дополнительным контекстом и примерами использования функций и команд Git на основе проекта Book.

Действия

Во врезках *"Действия"* представлены пронумерованные списки действий, которые вам следует выполнить на своем компьютере. Если шаг включает команду, выделенную жирным шрифтом, вам необходимо ввести и выполнить эту команду в командной строке. Для всех команд, которые генерируют вывод, будет показан пример вывода. Примеры вывода основаны на проекте Rainbow, над которым я работала при создании этой книги; он был создан в операционной системе macOS, но в Microsoft Windows вывод команд Git должен быть таким же. При наличии существенных различий между выводом команды в Microsoft Windows и macOS об этом говорится в тексте.

Запомните команду

Во врезках *"Запомните команду"* представлены полезные команды. Некоторые из них вы будете использовать во врезках *"Действия"*.

Полный список всех важных команд, сгруппированных по главам, также доступен в виде краткого справочника в *приложении 2*.

Диаграмма

Во врезках *"Диаграмма"* схематически показаны процессы, которые происходят в проекте Rainbow. Две важные иллюстрации, которые используются в этих разделах (а также на рисунках), — это *схема Git* и *схема репозитория*. Я познакомлю вас со схемой Git в *главе 2*, а использовать схему репозитория мы начнем в *главе 4*. Каждая схема в книге строится шаг за шагом с пояснениями в тексте.

Краткое описание визуального языка, используемого в этой книге, также доступно в *приложении 3*.

Примечание

Примечания содержат полезную информацию, связанную с материалом книги.

Обучающий репозиторий Git

Хотя я стремилась предоставить в этой книге максимум информации, необходимой вам для изучения Git, существуют технологии и процессы, которые меняются слишком часто, чтобы их можно было документировать в книге. Я создала общедоступный репозиторий Learning Git (**https://github.com/gitlearningjourney/learning-git**) на GitHub, чтобы предоставлять читателям актуальную информацию об этих технологиях и процессах. Помимо прочего, репозиторий содержит:

- информацию о скачивании Git;
- ссылки на ресурсы, связанные с использованием услуг хостинга;
- информацию о настройке доступа по протоколу HTTPS или SSH к удаленным репозиториям.

На протяжении всей книги я буду указывать, когда вам следует обратиться к репозиторию Learning Git для получения дополнительных рекомендаций.

Чем эта книга не является

Это не справочник. Я не буду описывать каждую команду Git. (Поверьте, их много!) Это также не расширенное руководство по Git. Существует множество функций Git, которые не будут рассмотрены в этой книге, поскольку они не нужны для выполнения основных действий, которым я хочу вас научить. Я очень строго ограничила содержание книги. Моя цель — дать вам четкое понимание основ Git, чтобы вы могли, имея прочную основу, продолжить изучение любых дополнительных функций, которые вам понадобятся.

Эта книга также не расскажет вам, как следует организовать рабочий процесс в Git или как лучше использовать возможности Git. Насколько это возможно, я постаралась избежать каких-либо предвзятых мнений и вместо этого сосредоточилась на том, чтобы научить вас основным навыкам работы с инструментом. В зависимости от ваших индивидуальных потребностей и предпочтений этот инструмент можно использовать по-разному.

Я постаралась также предоставить вам определенную свободу выбора. Например, для выполнения упражнений из книги допустимо использовать любой текстовый редактор или хостинг, который вы предпочитаете.

Краткое содержание глав

Книга разделена на две логические части. В первой части книги (*главы 1–5*) вы узнаете о работе с локальными репозиториями на вашем компьютере. В *главах 6–12* вы дополнительно узнаете о работе со службами хостинга и удаленными репозиториями. Ниже кратко изложено содержание каждой главы.

- В *главе 1* вы подготовитесь к работе над проектом с использованием Git, установив его, изучите основы работы с командной строкой, подготовите настройки Git, создадите один из каталогов проекта, который будете использовать в оставшейся части книги, и подготовите текстовый редактор.
- В *главе 2* вы превратите каталог вашего проекта в репозиторий Git. Вы увидите схему Git, которая представляет различные области Git, включая рабочий каталог, промежуточную область, историю коммитов и локальный репозиторий. В конце главы вы создадите первый файл в каталоге вашего проекта.
- В *главе 3* вы изучите и выполните два основных шага, необходимых для первого коммита в локальном репозитории.
- *Главе 4* вы узнаете о ветках: что это такое, как их создавать, как переключать ветки и как определить, на какой ветке вы находитесь.
- В *главе 5* вы узнаете о двух типах слияний и сможете выполнить ускоренное слияние.
- В *главе 6* вы подготовитесь к работе с удаленными репозиториями, выбрав службу хостинга и настроив данные аутентификации для подключения к этим репозиториям через HTTPS или SSH.
- В *главе 7* мы обсудим различные способы работы с локальными и удаленными репозиториями. Вы научитесь создавать удаленный репозиторий и загружать в него данные.
- В *главе 8* мы начнем моделировать совместную работу над проектом Git. Для этого вы создадите второй локальный репозиторий; вы сделаете вид, что он принадлежит вашему другу, который будет помогать вам в проекте Rainbow, и находится на его компьютере. Вы узнаете о клонировании удаленных репозиториев и получении данных.

- В *главе 9* вы выполните трехстороннее слияние и узнаете о разнице между извлечением и включением данных.
- В *главе 10* мы рассмотрим пример разрешения конфликтов во время трехстороннего слияния.
- В *главе 11* вы узнаете о перебазировании. Это альтернативный способ включения изменений из одной ветки в другую вместо слияния.
- В *главе 12* вы узнаете о запросах на включение (также известных как запросы на слияние) и о том, как они облегчают совместную работу в проектах Git.
- Книга также содержит три приложения, содержание которых обсуждалось ранее в *разд. "Приложения"*.

Условные обозначения

В этой книге используются следующие типографские соглашения:

- *курсив* обозначает новые термины и места, заслуживающие особого внимания, ссылки на главы и разделы книги;
- `моноширинный шрифт` используется для команд, вывода команд, имен ветвей, имен репозиториев и другого текста на экране компьютера;
- **`моноширинный жирный шрифт`** показывает команды или другой текст, который должен быть введен пользователем;
- `<моноширинный шрифт в угловых скобках>` показывает текст, который следует заменить значениями, предоставленными пользователем, или значениями, определяемыми контекстом.

[Глава 1]

Git и командная строка

В этой главе я расскажу, что такое Git и почему мы его используем, а вы установите его на своем компьютере. Вы также настроите некоторые переменные конфигурации Git. Вы узнаете о графическом интерфейсе пользователя и командной строке — двух инструментах, которые вы будете использовать для взаимодействия с Git и проектом Rainbow. Для того чтобы вам было удобно работать в командной строке, мы подробно разберем некоторые основные действия, такие как просмотр каталога в текущем местоположении, переход в каталоги и выход из них, а также создание каталогов. В конце главы вы подготовите текстовый редактор, который будете использовать для работы над проектом Rainbow в *главе 2*.

Если у вас есть опыт работы с командной строкой, возможно, некоторая информация из этой главы покажется вам знакомой. Но лучше не пропускать главу, потому что в ней говорится о настройках, которые понадобятся в остальной части книги.

ПРИМЕЧАНИЕ

Для того чтобы понять, как пользоваться этой книгой, вам необходимо ознакомиться с *разд. "Как читать эту книгу"* в *предисловии*. Если вы еще не читали этот раздел, я настоятельно рекомендую вам вернуться и сделать это сейчас.

Что такое Git?

Git — это технология, предназначенная для отслеживания изменений в проекте и для помощи в организации совместной работы над проектом. На базовом уровне проект, контролируемый Git, состоит из каталога (папки) с файлами внутри, и Git отслеживает изменения, вносимые в файлы проекта. Это позволяет вам сохранять разные версии выполняемой вами работы, поэтому Git называют *системой контроля версий*.

Git был создан Линусом Торвальдсом для контроля версий при работе над большим проектом — разработкой ядра Linux. Однако, поскольку Git может отслеживать изменения во всех типах файлов, он подходит для самых разных проектов.

Git — мощная система, и обилие предоставляемых ею функций, а также тот факт, что она изначально была создана для работы в терминале командной строки, означает, что использовать ее немного сложнее, чем просто выбрать пункт меню **Файл → Сохранить** на своем компьютере.

Итак, Git — это система контроля версий, которую вы можете загрузить на свой компьютер и которая позволяет отслеживать историю проекта и сотрудничать с другими людьми. Теперь рассмотрим пример 1.1 проекта Book, который показывает, как я могла бы применить Git для своего проекта книги.

Пример 1.1 проекта Book

Допустим, я пишу книгу и хочу использовать Git для контроля версий всех файлов в моем проекте Book. Каждый раз, когда я вношу изменения в книгу, я могу сохранить ее версию с помощью Git. Например, предположим, что я вношу изменения в книгу в понедельник, среду и пятницу и сохраняю по одной версии в каждый из этих дней. Это означает, что у меня есть как минимум три версии моего проекта. Версия проекта в Git называется *коммитом*. В *главе 2* вы узнаете больше о коммитах. На данный момент вам достаточно знать, что в моем примере у меня есть как минимум три коммита.

Эти три коммита позволяют мне просмотреть различные версии книги, которые я сохранила в конце рабочего дня в понедельник, в среду и в пятницу. Git также позволяет мне сравнивать любые из этих коммитов (или сохраненных версий моего проекта) друг с другом, чтобы проверить, что изменилось между разными версиями. Это пример того, как Git помогает отслеживать историю моего проекта.

Теперь предположим, что я решила работать над проектом Book вместе с соавтором. Git позволяет мне и моему соавтору одновременно работать над одним проектом и объединять нашу работу, когда мы будем готовы. Например, я работаю над главой 1, а мой соавтор — над главой 2, а когда мы закончим свои главы, то сможем объединить проделанную работу.

Если мы попросим редактора просмотреть книгу, он тоже сможет внести изменения во все главы книги, которые мы написали, а затем мы интегрируем эти изменения в основную версию книги. Этот пример показывает, насколько Git полезен для совместной работы.

Далее вы узнаете о других инструментах, которые будете использовать в этом учебном процессе, и о том, как взаимодействовать с Git.

Графический интерфейс пользователя и командная строка

Два основных способа взаимодействия с компьютером — использование графического интерфейса пользователя или командной строки.

Графический интерфейс пользователя (graphical user interface, GUI) — это набор графических представлений объектов (значков, кнопок и т. д.), позволяющих взаимодействовать с компьютером. Проще говоря, это интерфейс типа "укажи и нажми". Например, каталоги, представленные значками папок на рабочем столе, являются частью графического интерфейса вашего компьютера.

Командная строка, или *интерфейс командной строки* (command line interface, CLI), который в разных операционных системах также называется терминалом или оболочкой, — это место, где вы можете вводить текстовые команды для взаимодействия с вашим компьютером.

Стандартный способ работы с Git — через командную строку. Однако существуют также способы работы с Git с помощью графического пользовательского интерфейса, например с помощью клиента Git с графическим интерфейсом или текстового редактора с интеграцией с Git. Это означает, что вы можете выполнять действия Git, нажимая кнопки и выбирая параметры вместо ввода команд в командной строке.

В этой книге вы узнаете, как использовать Git в командной строке, поскольку это позволяет вам достичь надежного понимания того, как он работает, и дает доступ ко всем его функциям. Вы будете использовать графический интерфейс вашего компьютера только для других действий, например для просмотра файлов в файловой системе или для управления файлами во время работы в текстовом редакторе. Мы подробнее рассмотрим командную строку в следующем разделе.

ПРИМЕЧАНИЕ

В этой книге я буду предоставлять конкретные инструкции для пользователей MacOS и Microsoft Windows. Если вы пользователь Linux, я предполагаю, что вы уже знаете некоторые основы командной строки.

Запуск окна командной строки

Для того чтобы использовать командную строку, необходимо открыть окно специального приложения командной строки. В любой момент времени в окне командной строки вы находитесь в одном конкретном каталоге, который мы называем *текущим каталогом*. В нашем случае каталог — это то же самое, что и папка.

Когда вы откроете окно командной строки, в левом верхнем углу появится приглашение ввода команды. Это небольшой фрагмент текста, точное содержание которого будет отличаться в зависимости от вашей операционной системы и настроек компьютера. Однако по умолчанию приглашение командной строки указывает местоположение каталога в командной строке (другими словами, ваш текущий каталог). Когда вы открываете новое окно командной строки, расположение каталога начинается с каталога текущего пользователя (домашней папки), который обозначается знаком тильды (~). Это расположение каталога будет единственной важной частью командной строки, которую вам придется указать для упражнений в этой книге. Сразу после приглашения находится курсор, указывающий место ввода команды.

На рис. 1.1 приведен пример обычной командной строки с пояснениями. В примерах этой книги мы используем знак доллара ($) в конце приглашения командной строки, но это лишь один пример того, как может закончиться приглашение ввода. В вашем случае это может быть другой символ или буква.

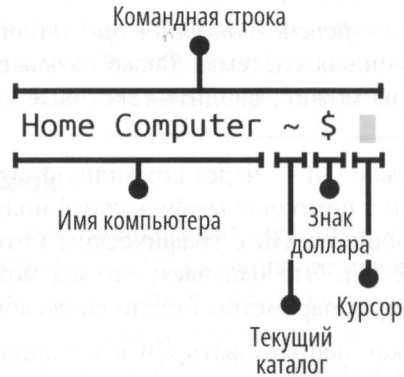


Рис. 1.1. Пример начала командной строки (приглашения ввода команды)

Приложение командной строки, которое вы будете использовать для выполнения упражнений из этой книги, зависит от используемой вами операционной системы:

- macOS — приложение командной строки называется Terminal (Терминал);
- Microsoft Windows — приложение командной строки называется Git Bash. Оно будет доступно только в том случае, если на вашем компьютере установлен Git.

> **ПРИМЕЧАНИЕ**
>
> Если вы пользователь Microsoft Windows и у вас не установлен Git, перейдите в репозиторий Learning Git (**https://github.com/gitlearningjourney/learning-git**) и установите Git для Microsoft Windows, чтобы получить доступ к Git Bash, прежде чем продолжить чтение остальной части этой главы.
>
> Если вы пользователь macOS и у вас не установлен Git, вы можете продолжить работу с оставшейся частью этого раздела. Вы установите Git в *разд. "Установка Git" далее в этой главе*.

Для того чтобы открыть окно командной строки, вы можете воспользоваться функцией поиска на своем компьютере, найти приложение командной строки, выбрать его и запустить. Выполните действия 1.1, чтобы открыть окно командной строки и рассмотреть приглашение ввода.

ДЕЙСТВИЯ 1.1

1. Используйте приложение командной строки, чтобы открыть окно командной строки.
2. Рассмотрите, из чего состоит приглашение ввода команды в окне командной строки.

На что обратить внимание:

- в командной строке указано расположение текущего каталога.

Теперь, когда вы открыли окно командной строки, давайте приступим к выполнению вашей первой команды.

Выполнение команд в командной строке

В конце приглашения ввода командной строки находится курсор, указывающий, где вы будете вводить команды. В Терминале (macOS) по умолчанию курсор находится на той же строке, что и командная строка, а в Git Bash (Microsoft Windows) — на строке ниже. После того как вы наберете команду, для ее выполнения необходимо нажать клавишу ввода (<Enter> или <Return>).

Если шаг во врезке "Действия" в этой книге содержит команду, выделенную **жирным шрифтом** после знака доллара (`$`), то ее необходимо выполнить в командной строке. Если команда возвращает выходные данные, они будут показаны под командой (не выделены жирным шрифтом). Если команда должна быть выполнена в каталоге, отличном от текущего каталога пользователя, местоположение каталога будет указано перед знаком доллара.

На рис. 1.2 показан пример того, как это будет выглядеть при выполнении команды во врезке "Действия". В данном случае я выполняю команду `pwd`, находясь в каталоге `desktop`. Вы узнаете, что делает команда `pwd`, позже в этой главе; а пока взгляните на рис. 1.2 и определите, где показаны расположение каталога, команда для выполнения и выходные данные.

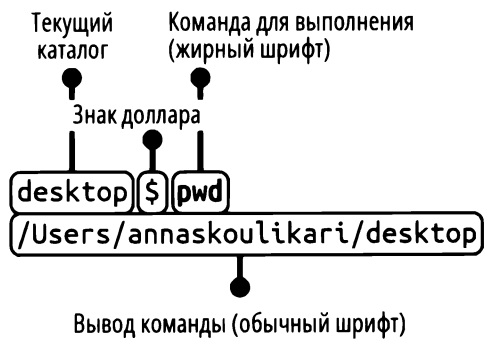


Рис. 1.2. Как выполнить команду во врезке "Действия"

В печатной версии этой книги некоторые более длинные команды переносятся на другую строку из-за ограничений ширины страницы. Если вы видите командную строку, подобную следующей:

```
rainbow $ git remote add origin https://github.com/gitlearningjourney/
rainbow-remote.git
```

вам следует ввести всю команду в одной строке без переноса (в данном случае без разрыва URL). Напоминания будут размещены в соответствующих местах. Если вы читаете электронную версию книги, которая не страдает от ограничения длины строки, вы можете игнорировать эти напоминания.

Вывод команды

Некоторые команды производят вывод, а некоторые — нет. Для команд, генерирующих выходные данные, я привожу примеры вывода, относящиеся к проекту Rainbow, над которым я работала при создании этой книги. Эти выходные данные были созданы операционной системой macOS, но выходные данные команд, связанных с Git, не различаются в разных операционных системах. В тех немногих случаях, когда вывод команд, не связанных с Git, существенно различается в разных операционных системах, я укажу это в тексте.

Если при выполнении упражнений из врезки "Действия" вы увидите выходные данные, которые радикально отличаются от примера вывода в этой книге, или получите неожиданную ошибку, возможно, вы сделали что-то не так, как указано в инструкциях, и вам следует выполнить указанные действия заново.

> **ПРИМЕЧАНИЕ**
>
> Вполне возможно, что в будущем Git претерпит обновления, которые могут незначительно повлиять на результаты. Если мне станет известно о каких-либо существенных изменениях, я постараюсь задокументировать эти ситуации на странице ошибок в репозитории Learning Git (**https://github.com/gitlearningjourney/learning-git**).

С этого момента всякий раз, когда вы встречаете команду во врезке "Действия", вам следует вводить команду, выделенную жирным шрифтом, в командной строке и выполнять ее.

Выполнение первой команды в командной строке

Первая команда, которую вы научитесь выполнять в командной строке, — это команда `git version`. Если на вашем компьютере установлен Git, он предоставит номер установленной вами версии. Если на вашем компьютере Git отсутствует, в результате вы получите сообщение о том, что необходимая программа не установлена.

Для того чтобы успешно выполнять все команды, используемые в упражнениях этой книги, я рекомендую вам иметь версию Git выше 2.28. Выполните действия 1.2, чтобы проверить, установлен ли у вас Git и какая у него версия.

> **ДЕЙСТВИЯ 1.2**
>
> 1 `$ git version`
>
> ```
> git version 2.35.1
> ```

На что обратить внимание:

- если у вас установлен Git, вы увидите его версию, установленную на вашем компьютере;
- если у вас не установлен Git, вместо этого вы увидите сообщение об ошибке.

Если выходные данные команды `git version` указывают, что у вас установлена версия Git выше 2.28, вы можете пропустить *разд. "Установка Git"* и перейти к *разд. "Параметры и аргументы команды"*.

Если выходные данные команды `git version` указывают на то, что на вашем компьютере не установлен Git или что версия Git старше 2.28, вам следует перейти к следующему разделу, чтобы установить актуальную версию Git на ваш компьютер.

Установка Git

Если у вас еще не установлен Git (версия 2.28 или новее), перейдите к действиям 1.3. В противном случае перейдите к следующему разделу.

ДЕЙСТВИЯ 1.3

1. Перейдите в репозиторий Learning Git (**https://github.com/gitlearningjourney/learning-git**) и следуйте инструкциям по загрузке Git для вашей операционной системы.

После установки Git на вашем компьютере приступим к подробному изучению команд, которые вы будете использовать в других врезках "Действия" этой книги.

Параметры и аргументы команды

Иногда вы будете использовать команды с опциями и/или аргументами. *Опции* — это настройки, которые меняют поведение команды. Опция следует за одинарным тире (-) или двойным тире (--).

Аргументы — это значения, которые предоставляют информацию команде. Они будут обозначаться угловыми скобками (<>), указывающими на то, что эти элементы следует заменить значениями, предоставленными пользователем. В упражнениях вы должны передавать значение аргумента без угловых скобок.

Вот пример команды с опцией и аргументом, которую вы будете использовать: `git commit -m "<message>"`. В этом примере `-m` — это опция, а `<message>` — аргумент, как показано на рис. 1.3. Мы рассмотрим, что делает эта команда, в *главе 3*.

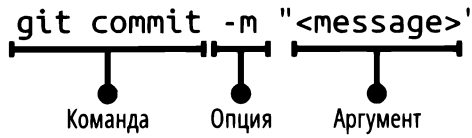


Рис. 1.3. Пример команды с опцией и аргументом

Помимо ввода команд в командной строке, важно также научиться их очищать.

Очистка командной строки

Каждый раз, когда вы вводите команду в окне командной строки, она будет указана непосредственно под предыдущей введенной вами командой (или ее выводом). После того как вы ввели много команд, окно командной строки становится сильно загроможденным. Для того чтобы очистить содержимое окна командной строки, можно воспользоваться командой `clear`.

ЗАПОМНИТЕ КОМАНДУ

clear

Очищает окно командной строки.

Выполните действия 1.4, чтобы попрактиковаться в использовании команды очистки.

ДЕЙСТВИЯ 1.4

1 `$ clear`

На что обратить внимание:

- окно командной строки очистилось.

Мы рассмотрели ввод команд в окне командной строки и его очистку. Теперь пришло время подготовить следующий инструмент, который поможет вам в обучении, — окно файловой системы.

Открытие окна файловой системы

Для того чтобы открыть окно файловой системы, которое является частью графического пользовательского интерфейса, необходимо воспользоваться стандартным приложением вашей операционной системы. На протяжении всего обучения вы будете взаимодействовать как с окном файловой системы, так и с окном командной строки. Поэтому полезно, чтобы оба окна были открыты рядом на экране вашего компьютера.

Приложение для работы с файловой системой компьютера зависит от вашей операционной системы:

- macOS — приложение файловой системы Finder;
- Microsoft Windows — приложение файловой системы Проводник (или Windows Explorer). Выполните действия 1.5, чтобы открыть окно программы для работы с файловой системой.

ДЕЙСТВИЯ 1.5

1. Найдите приложение файловой системы и откройте окно программы для работы с файловой системой рядом с окном командной строки.

Теперь у вас открыты оба окна, а мы продолжаем знакомство с основами командной строки.

Работа с каталогами

Как упоминалось ранее, в любой момент времени в окне командной строки вы находитесь в одном конкретном каталоге (текущий каталог). Если вы не изменили какие-либо настройки по умолчанию, то после запуска приложения командной строки вы окажетесь в домашнем каталоге текущего пользователя компьютера, обозначенном знаком тильды (~) в командной строке.

При переходе к другим каталогам начало командной строки изменяется, указывая каталог, в котором вы находитесь. Вы также можете использовать команду `pwd` (сокращение от *print working directory* — распечатать рабочий каталог), чтобы увидеть путь к текущему каталогу.

ЗАПОМНИТЕ КОМАНДУ

pwd

Показывает путь к текущему каталогу.

Выполните действия 1.6, чтобы попрактиковаться в использовании команды `pwd`.

ДЕЙСТВИЯ 1.6

1.
```
$ pwd
/Users/annaskoulikari
```

На что обратить внимание:

- вы находитесь в текущем каталоге пользователя.

ПРИМЕЧАНИЕ

В действиях 1.6 вывод команды `pwd` для пользователей Microsoft Windows будет аналогичен `/c/Users/annaskoulikari`, что немного отличается от вывода для пользователей macOS. Помните об этом при выполнении остальных упражнений этой главы.

В выводе команды `pwd` показан путь к текущему каталогу. В действиях 1.6 пример пути — `/Users/annaskoulikari`. Меня зовут Анна Скуликари, и `annaskoulikari` — это мое имя пользователя на моем компьютере. `Users` и `annaskoulikari` — это два каталога. Каталоги в строке пути разделяются косой чертой (`/`). Каталог `annaskoulikari` находится внутри каталога `Users`.

Знать расположение каталога в окне командной строки полезно, поскольку многие команды отображают информацию о текущем каталоге или влияют на него при их выполнении. Это знание также поможет вам перемещаться по файловой системе, о чем мы поговорим позже в этой главе.

Итак, вы научились определять текущий каталог, а теперь узнаете, как просмотреть фактическое содержимое каталога.

Просмотр содержимого каталогов

Содержимое каталога можно просмотреть как в графическом интерфейсе, так и в окне командной строки. Но прежде чем мы попробуем это сделать, я хочу уточнить, что в файловой системе существуют два типа файлов и каталогов: видимые и скрытые. *Видимые* файлы и каталоги всегда видны в файловой системе. *Скрытые* файлы и каталоги видны в файловой системе, только если вы измените настройки для их просмотра. Зачастую это файлы или каталоги, в которых хранится информация, доступ к которой нам, как пользователям, не нужен, например конфигурации приложений и различные настройки системы.

Не изменяйте и не удаляйте скрытые файлы и каталоги, если вы не знаете точно, что делаете! Как только вы измените настройки для просмотра скрытых файлов и каталогов, их значки станут частично прозрачными (серыми). Имена скрытых файлов и каталогов часто начинаются с точки (.).

В процессе изучения Git вы столкнетесь с несколькими важными скрытыми файлами и каталогами, о которых вам следует знать, поэтому обязательно научитесь просматривать их как в графическом интерфейсе, так и в командной строке.

В графическом интерфейсе, чтобы просматривать скрытые файлы и каталоги в окне файловой системы, вам необходимо явно сделать их видимыми:

- macOS — для переключения между просмотром и скрытием скрытых файлов и каталогов нажмите комбинацию клавиш <Cmd>+<Shift>+<.>;

- Microsoft Windows — измените настройки файловой системы, чтобы просмотреть скрытые файлы и каталоги. При необходимости обратитесь к онлайн-ресурсам и прочитайте пошаговые инструкции для вашего компьютера[1].

Выполните действия 1.7, чтобы просмотреть скрытые файлы и каталоги в вашей файловой системе.

ДЕЙСТВИЯ 1.7

1. Сделайте скрытые файлы и каталоги видимыми в вашей файловой системе.

В командной строке для просмотра списка видимых файлов и каталогов в текущем каталоге используется команда `ls` (сокращение от list — список).

Для того чтобы просмотреть как видимые, так и скрытые файлы и каталоги в текущем каталоге, используйте команду `ls` с опцией `-a`: `ls -a`.

ЗАПОМНИТЕ КОМАНДУ

ls

Выводит список видимых файлов и каталогов.

ls -a

Выводит список скрытых и видимых файлов и каталогов.

Выполните действия 1.8, чтобы попрактиковаться в использовании этих команд для вывода списка различных типов файлов.

ДЕЙСТВИЯ 1.8

1.
```
$ ls
Applications          Downloads       Music
Desktop               Library         Pictures
Documents             Movies          Public
```

2.
```
$ ls -a
.                     .config         Desktop
..                    .local          Documents
..CFUserTextEncoding  .npm            Downloads
.DS_Store             .ssh            Library
.Trash                .viminfo        Movies
.bash_history         .vscode         Music
.bash_sessions        .yarnrc         Pictures
.bashrc               Applications    Public
```

[1] В Windows 10, например, сделайте следующее: в строке **Поиск** кнопки **Пуск** введите с клавиатуры *проводник* (можно все строчными буквами). В результатах поиска главного меню (вверху) вы увидите строку с названием программы — **Проводник**. Щелкните по ней. Запустится программа для работы с файловой системой. На вкладке **Вид** окна программы в группе **Показать или скрыть** установите флажок **Скрытые элементы**. Теперь все скрытые файлы и папки будут вам видны. — *Прим. ред.*

На что обратить внимание:

- имена многих скрытых файлов и каталогов начинаются с точки (.);
- видимые и скрытые файлы и каталоги, показанные в выводе команд в этой книге, будут отличаться от файлов и каталогов на вашем компьютере, поскольку содержимое компьютеров у всех разное.

Теперь, когда вы научились определять текущий каталог и просматривать его содержимое в командной строке, нужно освоить перемещение между каталогами.

Перемещение из каталога и в каталог

В графическом интерфейсе, чтобы перейти в каталог, вы можете дважды щелкнуть по его значку. В командной строке для перехода в каталог применяют команду `cd` (это сокращение от change directory, что означает "изменить каталог") и передают ей либо имя каталога, либо путь к нему.

ЗАПОМНИТЕ КОМАНДУ

cd <путь_к_каталогу>

Изменяет каталог, в котором вы находитесь.

Выполните действия 1.9, чтобы попрактиковаться, входя в каталог desktop (это рабочий стол вашего компьютера).

ДЕЙСТВИЯ 1.9

1. `$ cd desktop`
2. `desktop $ pwd`
   `/Users/annaskoulikari/desktop`

На что обратить внимание:

- на шаге 1 команда `cd` не выдает никаких результатов;
- на шаге 2 приглашение командной строки и выходные данные команды `pwd` указывают, что вы находитесь в каталоге desktop.

Ранее в этой главе я упоминал, что командная строка показывает расположение каталога. Вы можете видеть, что во врезке действий 1.9 командная строка обновляется и показывает, что вашим текущим каталогом является desktop. По умолчанию способ представления местоположения каталога зависит от используемой вами операционной системы:

- macOS — в Терминале отображается имя текущего каталога;
- Microsoft Windows — в Git Bash отображается путь к текущему каталогу.

ПРИМЕЧАНИЕ

Переход в каталоги и выход из них с помощью командной строки не влияет на то, что вы просматриваете в файловой системе. Например, переход в каталог рабочего стола в командной строке не приведет к автоматическому отображению на экране содержимого каталога рабочего стола.

В графическом интерфейсе, чтобы вернуться в родительский каталог, вы можете нажать кнопку **Назад**. В командной строке, чтобы вернуться в родительский каталог, нужно передать команде `cd` две точки (`..`). Две точки обозначают родительский (верхний) каталог текущего каталога. Выполните действия 1.10, чтобы осуществить этот переход.

ДЕЙСТВИЯ 1.10

1. `desktop $ cd ..`
2. `$ pwd`
   `/Users/annaskoulikari`

На что обратить внимание:

- на шаге 2 командная строка и вывод `pwd` указывают, что вы вернулись в каталог текущего пользователя.

Итак, вы научились переходить в каталоги и выходить из них. Теперь вы узнаете, как создавать новые каталоги.

Создание каталога

В графическом интерфейсе для создания каталога нужно щелкнуть правой кнопкой мыши в окне Проводника файловой системы или выбрать соответствующий пункт меню или значок. В командной строке для создания каталога вы будете использовать команду `mkdir` (сокращение от make directory — создать каталог). При выполнении команды новый каталог будет создан внутри текущего.

ЗАПОМНИТЕ КОМАНДУ

mkdir <имя_каталога>

Создает каталог.

Для того чтобы упростить себе жизнь, избегайте использования пробелов в именах каталогов. Если имя каталога содержит пробелы, вам придется внести изменения в использование определенных команд в командной строке, что затруднит вашу работу.

ПРИМЕЧАНИЕ

В целом при работе в командной строке вам следует избегать использования пробелов в именах любых файлов, каталогов или других объектов, которые вы создаете, поскольку это может вызвать проблемы при использовании команд.

Как упоминалось в предисловии, на протяжении всей этой книги вы будете работать над одним проектом, в котором будете создавать и редактировать файлы со списком цветов радуги, а также некоторых цветов, не являющихся частью радуги. Это учебный проект, версию которого вы контролируете с помощью Git; он очень простой, но так вам будет легче сосредоточиться на изучении того, как работает Git.

Поскольку основная цель учебного проекта — составить список цветов радуги, вы присвоите каталогу проекта имя `rainbow`. Вы создадите этот каталог проекта в каталоге desktop, чтобы иметь возможность сразу увидеть его на экране компьютера. Выполните действия 1.11, чтобы создать свой каталог.

ДЕЙСТВИЯ 1.11

1. `$ cd desktop`
2. `desktop $ mkdir rainbow`
3. `desktop $ ls`
   `rainbow`

На что обратить внимание:

- на шаге 3 в выводе `ls` будет показан каталог проекта `rainbow`, который вы только что создали, а также любые другие каталоги или файлы, которые есть в каталоге вашего рабочего стола. Они не отображаются в примере вывода, поскольку содержимое вашего компьютера будет отличаться от моего.

Если вы посмотрите на рабочий стол своего компьютера, то увидите только что созданный каталог проекта `rainbow`, как показано на рис. 1.4.

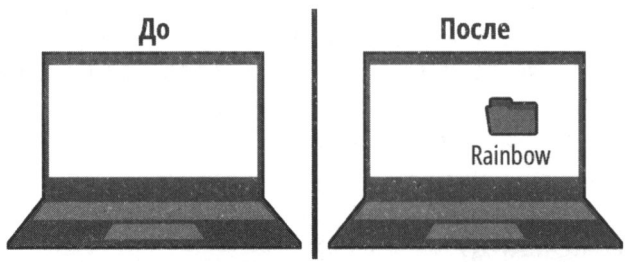


Рис. 1.4. Пример рабочего стола компьютера до и после создания каталога проекта `rainbow`

Тот факт, что вы создали каталог проекта `rainbow`, не означает, что вы перешли в него из командной строки. Выполните действия 1.12, чтобы явно перейти в каталог проекта `rainbow` в командной строке.

ДЕЙСТВИЯ 1.12

1. `desktop $ cd rainbow`
2. `rainbow $ pwd`
   `/Users/annaskoulikari/desktop/rainbow`

На что обратить внимание:

- на шаге 2 командная строка и выходные данные команды `pwd` указывают на то, что вы находитесь в каталоге `rainbow`.

Вы создали каталог `rainbow` и перешли в него. А что произойдет, если вы закроете окно командной строки и снова откроете его?

Закрытие окна командной строки

По умолчанию, если вы закроете окно командной строки, а затем откроете его снова, местоположение каталога будет сброшено до каталога текущего пользователя. Поэтому вам придется снова перейти в каталог, в котором вы хотите работать. Выполните действия 1.13, чтобы проверить это.

ДЕЙСТВИЯ 1.13

1. `rainbow $ pwd`
   `/Users/annaskoulikari/desktop/rainbow`
2. Закройте текущее окно командной строки, а затем откройте новое окно командной строки.
3. `$ pwd`
   `/Users/annaskoulikari`
4. `$ cd desktop`
5. `desktop $ cd rainbow`
6. `rainbow $ pwd`
   `/Users/annaskoulikari/desktop/rainbow`

На этом этапе мы рассмотрели основы работы с командной строкой. Далее вы настроите некоторые базовые конфигурации Git.

Настройка конфигураций Git

Конфигурации Git — это настройки, которые позволяют вам управлять работой Git. Они состоят из переменных и их значений и хранятся в нескольких разных файлах. Для работы с Git необходимо настроить значения нескольких переменных конфигурации, связанных с настройками пользователя.

Прежде чем настраивать какие-либо переменные, необходимо проверить, существует ли глобальный файл конфигурации Git в вашей файловой системе, и если да, то какие значения переменных были установлены. Для этого воспользуйтесь командой `git config`, передав ей параметры `--global` и `--list`.

Обратите внимание, что команда `git config` с опцией `--global` является примером команды, для которой не имеет значения, в каком текущем каталоге вы находитесь при ее выполнении; она будет отображать или изменять информацию только в глобальном файле конфигурации Git. Это скрытый файл с именем .gitconfig, который обычно создается в каталоге текущего пользователя.

ЗАПОМНИТЕ КОМАНДУ

git config --global --list

Выводит список переменных и их значений в глобальном файле конфигурации Git.

Выполните действия 1.14, чтобы испытать эту команду.

ДЕЙСТВИЯ 1.14

1 `$rainbow $ git config --global --list`

```
fatal: unable to read config file '/Users/annaskoulikari/.
gitconfig': No such file or directory
```

(сбой: невозможно прочитать файл конфигурации ... :

файл или каталог не существует)

На что обратить внимание:

- в данном примере вывод команды показывает ответ, который вы увидите, если никогда не настраивали какие-либо переменные в глобальном файле конфигурации Git. В этом случае вы получите сообщение об ошибке, указывающее, что файл не существует. Если вы настроили переменные в своем глобальном файле конфигурации Git, то в выводе будут отображены соответствующие переменные и их значения.

В этой книге нас интересуют две переменные: `user.name` и `user.email`. Каждый раз, когда кто-то сохраняет версию проекта (или, другими словами, делает коммит), Git запоминает имя и адрес электронной почты человека и связывает его с этой сохраненной версией. Переменные `user.name` и `user.email` нужны для настройки имени и адреса электронной почты, которые будут сохраняться для сделанных вами коммитов. Благодаря этому вы можете видеть, кто над чем работал в проекте Git. Настройка этих переменных необходима для работы с Git. Учтите, что любой, кто может видеть список ваших коммитов в любом проекте, сможет просмотреть ваш адрес электронной почты, поэтому обязательно используйте адрес, который можно сообщить другим людям.

Для того чтобы присвоить значения переменным в глобальном файле конфигурации Git, передайте их в качестве аргументов команде `git config`, вводя нужные значения в кавычки (помните, что вы не должны заключать значения в угловые скобки).

ЗАПОМНИТЕ КОМАНДУ

git config --global user.name "<имя>"

Задает ваше имя в глобальном файле конфигурации Git.

git config --global user.email "<адрес email>"

Задает ваш адрес электронной почты в глобальном файле конфигурации Git.

Если в выводе действий 1.14 этим переменным присвоены нужные вам значения, то вы можете пропустить действия 1.15. Если эти переменные не отображаются или им заданы неправильные значения, решите, какое имя и адрес электронной почты вы хотите связать с работой, выполняемой в проекте `rainbow`, и выполните действия 1.15. Обязательно подставьте значения, которые вы хотите использовать вместо моего имени пользователя и адреса электронной почты.

ДЕЙСТВИЯ 1.15

1. `rainbow $ git config --global user.name "annaskoulikari"`
2. `rainbow $ git config --global user.email "gitlearningjourney@gmail.com"`
3. ```
   rainbow $ git config --global -list
   user.name=annaskoulikari
   user.email=gitlearningjourney@gmail.com
   ```

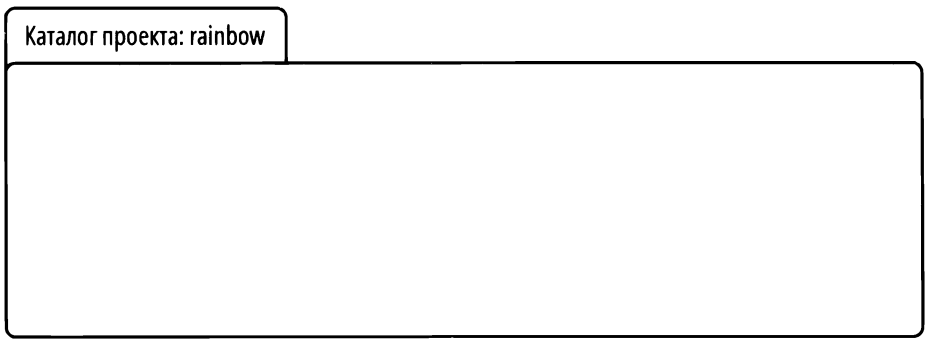
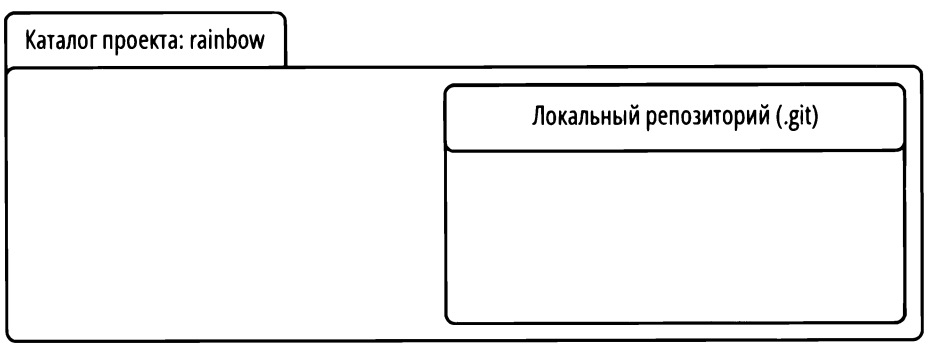
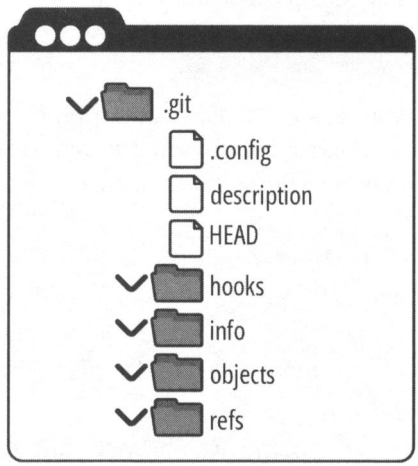
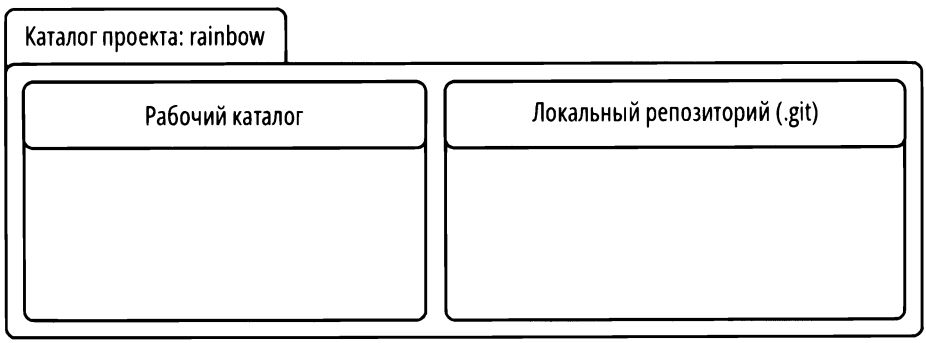
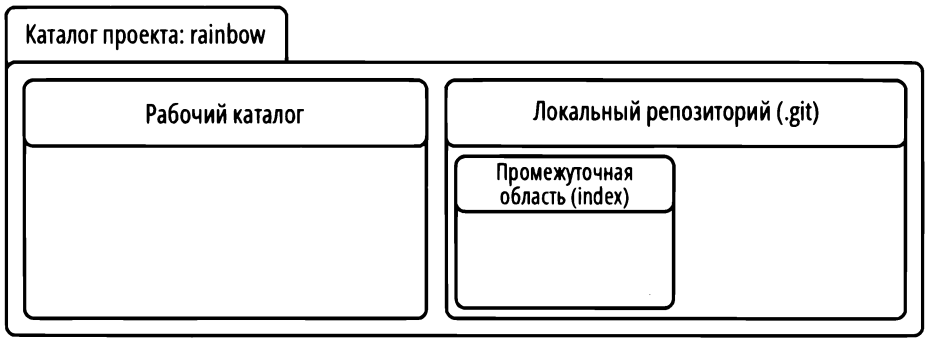
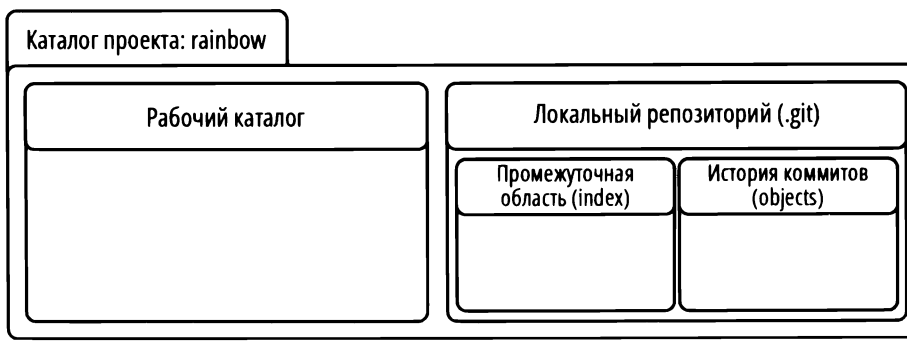
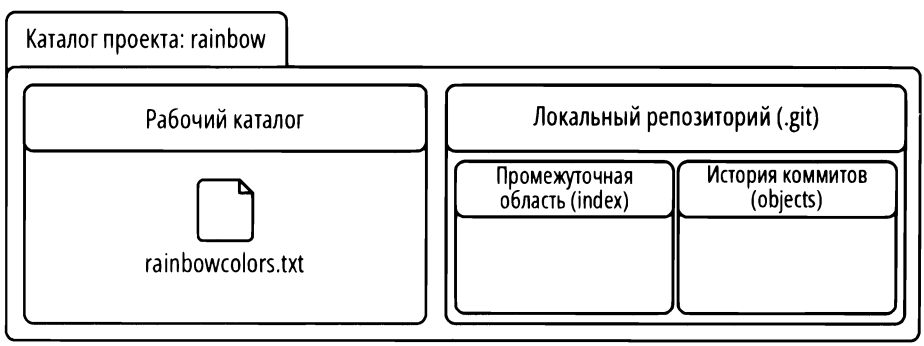
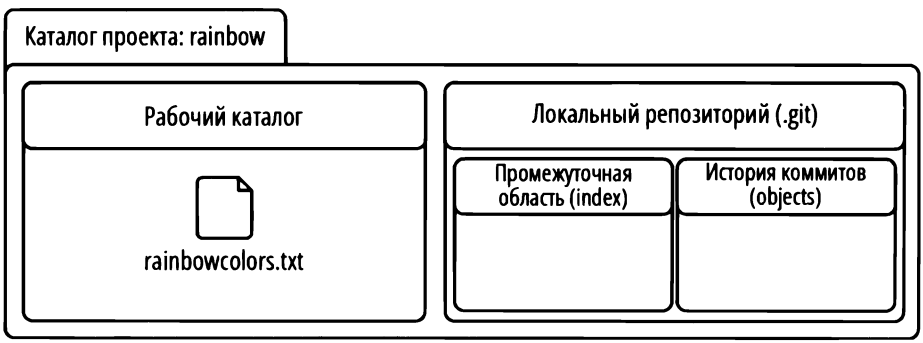
На что обратить внимание:

- на шаге 3 в выводе `git config --global --list` переменные `user.name` и `user.email` содержат введенные вами значения.

К этому моменту вы установили Git и подготовили пользовательские настройки. Последний инструмент, который вам понадобится для работы над проектом Git, — это текстовый редактор.

Подготовка текстового редактора

Проект Git состоит из файлов и каталогов, версии которых контролируются. Git может отслеживать версии файлов любого типа. В проекте `rainbow` вы будете использовать текстовый редактор для работы с простыми текстовыми файлами (имеющими расширение .txt).

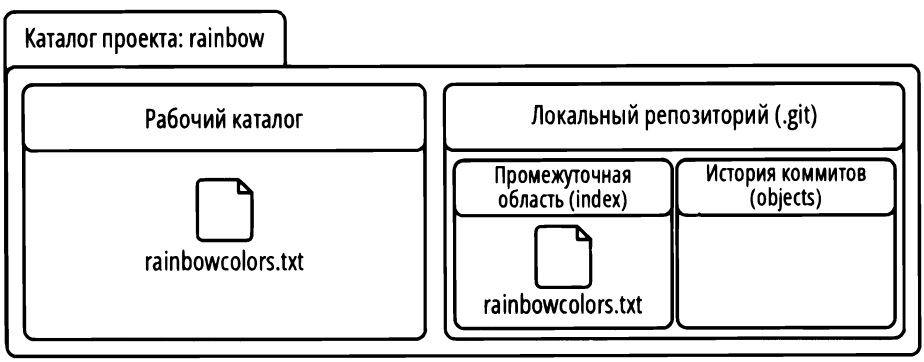
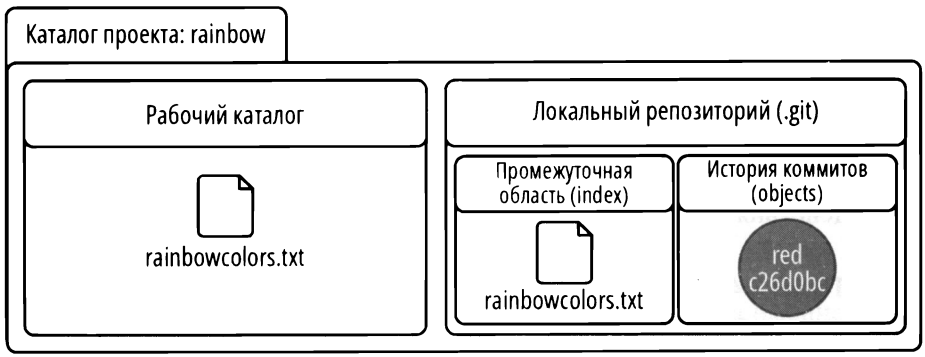
Текстовый редактор — это программа, которая позволяет пользователю редактировать обычный текст. Для выполнения многих упражнений из этой книги вам понадобится текстовый редактор. Текстовый редактор отличается от текстового процессора, который в основном используется для редактирования форматированного текста. Примерами текстовых процессоров являются Microsoft Word и Google Docs;
   ```

их нельзя использовать для управления файлами проекта Git. Форматированный текст — это текст со стилями, прикрепленными к нему или встроенными в него. Если вы видите, что текст выделен жирным шрифтом или курсивом, значит, он содержит стили форматирования.

Некоторые текстовые редакторы намного больше упрощают работу над проектами Git, чем другие. Эта книга написана таким образом, что вы можете использовать любой текстовый редактор, который вы предпочитаете. Если у вас уже установлен текстовый редактор, который вы применяли для работы над проектами Git, смело используйте его для упражнений из этой книги. Если вы не уверены, какой текстовый редактор выбрать, я настоятельно рекомендую Visual Studio Code (**https://code.visualstudio.com**): это популярный текстовый редактор, и именно его я использовала при работе над проектом Rainbow в этой книге. Выполните действия 1.16, чтобы подготовить текстовый редактор.

ДЕЙСТВИЯ 1.16

1. Выберите текстовый редактор, который вам нравится. Если он у вас еще не установлен, скачайте и установите его.
2. Откройте каталог проекта `rainbow` в окне текстового редактора.

Интегрированные терминалы

Некоторые текстовые редакторы с расширенными функциями (также называемые *интегрированными средами разработки* — integrated development environments, или IDE), такие как Visual Studio Code, оснащены собственным инструментом командной строки — так называемым *встроенным терминалом*, в котором вы можете выполнять команды, которые обычно вводите в окне командной строки.

Когда вы используете встроенный терминал, вам может быть проще управлять пространством экрана, поскольку терминал уже является частью окна текстового редактора. Однако это зависит от личных предпочтений. Если ваш текстовый редактор имеет встроенный терминал, вы можете использовать его вместо отдельного окна командной строки для выполнения команд Git в остальных упражнениях этой книги. Любой вариант будет работать нормально, поэтому вам решать, что вы предпочитаете.

В оставшейся части книги всякий раз, когда я хочу указать место, где вам нужно выполнить команду, я буду ссылаться на окно командной строки; однако имейте в виду, что это относится и к встроенному терминалу.

Итак, Git установлен и настроен, текстовый редактор ожидает ввод текста, значит, и вы готовы приступить к работе над проектом Rainbow!

Краткое содержание главы

В этой главе вы начали знакомство с Git и узнали, что это полезный инструмент для отслеживания истории проекта и совместной работы с коллегами. Вы подготовились к работе над проектом с использованием Git, установив на свой компьютер актуальную версию и задав некоторые базовые переменные конфигурации Git.

Вы также получили первые навыки работы с командной строкой, научились просматривать и создавать каталоги, а также входить в них и перемещаться между ними. Попутно вы создали каталог проекта под названием `rainbow`, который будете использовать на протяжении всей книги.

Наконец, вы подготовили текстовый редактор, позволяющий создавать и редактировать файлы для любого проекта Git, над которым вы будете работать.

Теперь вы готовы перейти к *главе 2*, где превратите каталог проекта `rainbow` в репозиторий Git и начнете изучать наиболее важные аспекты работы с Git.

[Глава 2]

Локальные репозитории

В предыдущей главе вы изучили основы командной строки и подготовились к работе с Git, загрузив его и настроив некоторые параметры.

В этой главе вы превратите созданный ранее каталог проекта `rainbow` в репозиторий Git. Вы также узнаете о четырех важных компонентах Git: это рабочий каталог, промежуточная область, история коммитов и локальный репозиторий. Для того чтобы вам было легче представить, как эти компоненты связаны между собой, мы построим диаграмму Git, включающую представление каждого компонента.

В конце этой главы вы добавите первый файл в каталог проекта Rainbow и заодно узнаете о неотслеживаемых и отслеживаемых файлах. Давайте начнем!

Текущее состояние

К началу этой главы вы должны были сделать следующее:

- установить или обновить Git на своем компьютере (вам понадобится версия не ниже 2.28);
- создать на рабочем столе пустой каталог проекта под названием `rainbow`;
- открыть окно командной строки и перейти в каталог `rainbow`;
- решить, какой текстовый редактор использовать, и открыть каталог проекта `rainbow` в окне текстового редактора;
- присвоить глобальным переменным конфигурации Git `user.name` и `user.email` свое имя и адрес электронной почты соответственно.

Знакомство с репозиторием

Репозиторий — это, по сути, способ указать на место хранения проекта, версию которого контролирует Git. На самом деле существуют два типа репозиториев:

- *локальный репозиторий* — это репозиторий, размещенный на вашем компьютере;

- *удаленный репозиторий* — это репозиторий, размещенный на сервере хостинговой службы.

Хостинговая служба — это компания, предоставляющая хостинг (размещение файлов на удаленном сервере) проектам, использующим Git. На момент написания книги основными хостингами Git являлись GitHub (**https://github.com**), GitLab (**https://about.gitlab.com**) и Bitbucket (**https://bitbucket.org/product**).

В первой части этой книги, вплоть до *главы 5*, вы познакомитесь и будете работать только с локальными репозиториями. Во второй части книги, начиная с *главы 6*, вы также научитесь работать с удаленными репозиториями.

Итак, вы научились различать локальные и удаленные репозитории, а дальше узнаете о первоначальной инициализации локального репозитория.

Инициализация локального репозитория

Локальный репозиторий представлен скрытым каталогом с именем .git, который расположен в каталоге проекта. Он содержит все данные об изменениях, внесенных в файлы проекта.

Для того чтобы превратить каталог проекта в локальный репозиторий, вам необходимо *инициализировать* репозиторий по месту расположения проекта. Когда вы инициализируете репозиторий, внутри каталога проекта автоматически создается каталог .git. Поскольку это скрытый каталог, вы не сможете его увидеть, если явно не сделаете видимыми скрытые файлы и каталоги в настройках файловой системы.

> **ПРИМЕЧАНИЕ**
>
> Никогда не трогайте какие-либо файлы или каталоги внутри вашего каталога .git. Это может иметь нежелательные последствия для вашего репозитория. Никогда не следует удалять этот каталог, если вы не хотите удалить свой репозиторий.

Во врезке действий 2.1 вы убедитесь, что ваши настройки сконфигурированы на просмотр скрытых файлов и каталогов. (Если вам нужно вспомнить, как это сделать, обратитесь к *разд. "Просмотр содержимого каталогов" главы 1*.) Затем вы проверите как в окне файловой системы, так и в окне командной строки, есть ли какие-либо видимые или скрытые файлы в каталоге проекта `rainbow`.

> **ДЕЙСТВИЯ 2.1**
>
> 1. Сделайте скрытые файлы и каталоги видимыми в вашей файловой системе.
> 2. Откройте каталог проекта `rainbow` в окне файловой системы и просмотрите его содержимое. В нем не должно быть видимых или скрытых файлов и каталогов.
> 3. `rainbow $ ls -a`

На что обратить внимание:

- на шаге 2 в окне файловой системы вы видите, что каталог проекта `rainbow` пуст;
- на шаге 3 в окне командной строки вы также видите, что каталог проекта `rainbow` пуст.

Для того чтобы подготовиться к построению диаграммы Git, мы создадим изображение пустого каталога проекта `rainbow`, показанное на диаграмме 2.1.

ДИАГРАММА 2.1



Представление пустого каталога проекта `rainbow`, который будет содержать диаграмму Git.

Для инициализации репозитория Git применяется команда `git init`. При выполнении этой команды вашим текущим каталогом должен быть каталог проекта, который вы хотите превратить в репозиторий.

Обычно пользователи Git инициализируют репозиторий, выполняя только команду `git init` без дополнительных параметров; но в проекте Rainbow вы инициализируете репозиторий с помощью команды `git init` с опцией `-b` (сокращение от `--initial-branch`) и передаете имя ветки `main`. Мы более подробно рассмотрим, что такое ветки, в *главе 4*. На данный момент вам достаточно знать, что по умолчанию Git создает ветку с именем `master`, когда вы инициализируете новый локальный репозиторий. Начиная с версии Git 2.28, имя начальной ветки можно настраивать. В этой книге я решила использовать имя `main` вместо `master`, поскольку слово "master" считается недостаточно политкорректным. Мы обсудим этот вопрос подробнее в *разд. "Немного истории Git: master и main" главы 4*.

ПРИМЕЧАНИЕ

Если вы хотите, чтобы начальная ветка во всех ваших репозиториях имела имя, отличное от `master`, присвойте нужное текстовое значение переменной с именем `init.defaultBranch` в своем глобальном файле конфигурации. Этот процесс аналогичен настройке переменных `user.name` и `user.email` в *разд. "Настройка конфигураций Git" главы 1*. Если переменной `init.defaultBranch` присвоено значение, то при инициализации репозитория Git с помощью команды `git init` начальная ветка будет получать имя, определенное в конфигурации.

ЗАПОМНИТЕ КОМАНДУ

git init

Инициализирует репозиторий Git.

git init -b <имя_ветки>

Инициализирует репозиторий Git и задает имя начальной ветки `<имя_ветки>`.

Выполните действия 2.2, чтобы превратить каталог проекта `rainbow` в репозиторий Git.

ДЕЙСТВИЯ 2.2

1. Для того чтобы увидеть создаваемый каталог .git, вы должны иметь возможность просматривать содержимое каталога вашего проекта Rainbow в окне файловой системы с включенным отображением скрытых файлов и каталогов.
2. `rainbow $ git init -b main`

   ```
   Initialized empty Git repository in/Users/annaskoulikari/
   desktop/rainbow/.git/
   ```

   (*Инициализирован пустой репозиторий Git в /Users/annaskoulikari/desktop/rainbow/.git/*)
3. Перейдите в каталог проекта Rainbow в окне файловой системы и просмотрите только что созданный каталог .git. Откройте каталог .git, чтобы просмотреть его содержимое.

На что обратить внимание:

- Git создал каталог .git внутри каталога проекта `rainbow`.

Первая область, для которой мы добавим представление в диаграмму Git, — это локальный репозиторий, представленный самим каталогом .git. Его можно увидеть на диаграмме 2.2.

ДИАГРАММА 2.2



Диаграмма Git с представлением локального репозитория.

Внутри каталога .git находятся различные файлы и каталоги. Некоторые из них представляют области Git, о которых вам предстоит узнать позже. В следующем разделе мы также добавим их представления в диаграмму Git. По мере продолжения работы с репозиторием `rainbow` в каталоге .git будут созданы новые файлы и каталоги; о некоторых из них вы узнаете по ходу дела.

На рис. 2.1 показан пример содержимого каталога .git в только что инициализированном репозитории.



Рис. 2.1. Содержимое каталога .git сразу после выполнения команды `git init`

Далее мы рассмотрим различные области, с которыми вы будете взаимодействовать при работе с Git.

Области Git

При работе с Git следует обращать внимание на четыре важные области:

- рабочий каталог;
- промежуточная область;
- история коммитов;
- локальный репозиторий.

О локальном репозитории вы узнали в предыдущем разделе. В этом разделе мы рассмотрим остальные области и то, как они связаны друг с другом, а также продолжим построение диаграммы Git. Начнем с рабочего каталога.

Знакомство с рабочим каталогом

Рабочий каталог содержит файлы и каталоги, которые представляют одну версию проекта. Это что-то вроде рабочего верстака. Здесь вы можете добавлять, редактировать и удалять файлы и каталоги.

Взгляните на пример 2.1 проекта Book, чтобы понять, как происходит управление файлами в рабочем каталоге.

Пример 2.1 проекта Book

Предположим, проект Book, над которым я работаю, состоит из 10 глав, и у меня есть 10 текстовых файлов — по одному на каждую главу: chapter_one.txt, chapter_two.txt и т. д.

Для того чтобы добавить каждый из этих файлов глав в свой проект, я бы создала эти файлы в рабочем каталоге.

Если бы я захотела внести какие-либо изменения в содержание этих глав, я бы начала с редактирования файлов в рабочем каталоге.

И, наконец, если бы я решила удалить целую главу своей книги, первым шагом было бы удаление соответствующего файла в рабочем каталоге.

Из примера 2.1 проекта Book можно сделать вывод, что рабочий каталог — это место, где вы вносите все изменения в содержимое проекта.

В случае с репозиторием `rainbow` рабочий каталог в настоящее время пуст. Мы продолжим строить диаграмму Git и добавим в нее представление рабочего каталога, как показано на диаграмме 2.3.

ДИАГРАММА 2.3



Диаграмма Git с представлением рабочего каталога и локального репозитория.

На диаграмме 2.3 каталог проекта `rainbow` содержит локальный репозиторий. Однако важно знать, что когда люди ссылаются на версию проекта, управляемую с помощью Git, они обычно ссылаются на каталог проекта как на репозиторий. Например, в нашем случае я буду постоянно упоминать "репозиторий `rainbow`".

В локальном репозитории есть две важные области, которые мы хотим изучить дальше: промежуточная область и история коммитов. Далее мы рассмотрим их, а также обсудим концепцию коммита более подробно.

Знакомство с промежуточной областью

Промежуточная область (staging area), которую часто также называют *индексом* (index), похожа на черновик. Здесь вы можете добавлять и удалять файлы, когда готовите содержимое для включения в следующую сохраненную версию вашего проекта (следующий коммит). Промежуточная область представлена файлом index в каталоге .git.

> **ПРИМЕЧАНИЕ**
>
> Файл index создается только в том случае, если вы добавили хотя бы один файл в промежуточную область вашего проекта. В каталоге проекта `rainbow` вы еще не добавили в эту область ни одного файла, поэтому файл index отсутствует в каталоге .git. В *главе 3* вы добавите файл в промежуточную область и увидите, что появился файл index.

Вы на своем опыте узнаете, почему промежуточная область так полезна, когда попрактикуетесь добавлять в нее файл в *главе 3*. А сейчас мы добавим ее представление внутри локального репозитория на схеме Git, как показано на диаграмме 2.4.

ДИАГРАММА 2.4



Диаграмма Git с представлением рабочего каталога, промежуточной области и локального репозитория.

Схема Git теперь содержит представления рабочего каталога, промежуточной области и локального репозитория. Последняя область, которую я хочу представить, — это история коммитов, но прежде чем я это сделаю, предлагаю внимательно изучить понятие коммита.

Что такое коммит?

Коммит (commit) в Git — это, по сути, одна версия проекта. Вы можете рассматривать его снимок проекта или как отдельную версию проекта, содержащую ссылки на все файлы, являющиеся частью этого коммита.

У каждого коммита есть *хеш коммита* (иногда называемый *идентификатором коммита*). Это уникальный 40-символьный хеш, состоящий из букв и цифр, который действует как имя коммита и позволяет ссылаться на него.

Пример хеша коммита: `51dc6ecb327578cca503abba4a56e8c18f3835e1`.

На самом деле для ссылки на коммит нужны только первые семь символов хеша коммита. Поэтому в приведенном выше примере хеша для ссылки на коммит вам достаточно использовать символы `51dc6ec`.

ПРИМЕЧАНИЕ

Поскольку хеши коммитов уникальны, ваши хеши коммитов в проекте Rainbow будут отличаться от тех, которые показаны в этой книге. Помните об этом, выполняя последующие упражнения из книги.

Теперь, когда вы получили представление о том, что такое коммиты, давайте обсудим последнюю область, которую мы добавим в нашу диаграмму Git, — историю коммитов.

Знакомство с историей коммитов

История коммитов (commit history) — это место, где хранится информация о существующих коммитах. Она представлена каталогом objects внутри каталога .git. Для того чтобы глубже понять историю коммитов, нам придется углубиться во внутреннее устройство Git, но это сложная тема, знание которой вам не понадобится при изучении основ использования Git. Для наших целей все, что вам нужно знать: каждый раз, когда вы делаете коммит, он сохраняется в истории коммитов.

Выполните действия 2.3, чтобы найти историю коммитов в репозитории `rainbow`.

ДЕЙСТВИЯ 2.3

1. В окне файловой системы внутри каталога проекта `rainbow` загляните в каталог .git и найдите каталог objects.

На диаграмме 2.5 мы добавляем представление истории коммитов в диаграмму Git внутри локального репозитория. На этом диаграмма Git завершена.

ДИАГРАММА 2.5



Полная диаграмма Git с представлением рабочего каталога, промежуточной области, истории коммитов и локального репозитория.

Итак, у нас есть полная диаграмма Git, показывающая наиболее важные области при работе с Git. Теперь добавим первый файл в проект Rainbow и отредактируем его с помощью текстового редактора.

Добавление файла в проект Git

Как упоминалось ранее, на протяжении всей книги вам предстоит работать над проектом, в котором вы будете перечислять цвета радуги и некоторые цвета, которые не являются частью радуги. Каждый раз, добавляя цвет, вы будете делать коммит, чтобы следить за ходом реализации проекта.

Первым шагом станет создание файла rainbowcolors.txt, в котором вы будете перечислять цвета. Как было сказано в *разд. "Создание каталога" главы 1*, важно, чтобы имя файла не содержало пробелов. Затем, используя текстовый редактор, который вы подготовили в *главе 1*, вы добавите текст "Красный — первый цвет радуги." в строке 1 этого файла. Выполните действия 2.4, чтобы создать и отредактировать файл.

ДЕЙСТВИЯ 2.4

1. С помощью текстового редактора создайте файл rainbowcolors.txt в каталоге проекта `rainbow`.
2. В строке 1 добавьте текст "Красный — первый цвет радуги." и сохраните файл.

На что обратить внимание:

- файл rainbowcolors.txt хранится в каталоге проекта `rainbow`; следовательно, он находится в рабочем каталоге.

Несмотря на то что файл rainbowcolors.txt находится в рабочем каталоге, он не является частью вашего репозитория. Он не был добавлен в промежуточную область и не был включен в коммит в истории коммитов. Это видно на диаграмме 2.6.

ДИАГРАММА 2.6



Каталог проекта `rainbow` после добавления файла rainbowcolors.txt в рабочий каталог.

Поскольку файла rainbowcolors.txt еще нет в вашем репозитории, это неотслеживаемый файл. *Неотслеживаемый файл* (untracked file) — это файл в рабочем каталоге, версии которого Git не контролирует. Он никогда не добавлялся в промежуточную область и никогда не был включен в коммит; следовательно, он не является частью репозитория.

Как только вы добавите файл в промежуточную область и включите его в коммит, он станет *отслеживаемым файлом* (tracked file). Это файл с контролем версий (другими словами, файл, который отслеживает Git).

Каждый новый файл в версии проекта, контролируемой Git, должен быть явно добавлен в промежуточную область, а затем включен в коммит, чтобы стать отслеживаемым файлом. Вы выполните эти шаги в *главе 3*.

ПРИМЕЧАНИЕ

В предисловии я упомянула, что создание и редактирование файлов со списком цветов радуги является не очень реалистичным примером проекта, версиями которого управляет Git. Напомню, что цель этого проекта — сделать процесс обучения как можно проще, чтобы вы могли сосредоточиться на том, как работает Git, а не на содержимом редактируемых файлов. Изменения, которые вы будете вносить в файлы в реальных проектах, будут выглядеть совсем иначе.

Краткое содержание главы

В этой главе мы превратили каталог проекта `rainbow` в репозиторий при помощи процесса инициализации. Мы также создали диаграмму Git, отображающую четыре важные области Git: рабочий каталог, промежуточную область, историю коммитов и локальный репозиторий.

Вы добавили свой первый файл в каталог проекта `rainbow` и узнали о важном различии между неотслеживаемыми и отслеживаемыми файлами.

В следующей главе мы рассмотрим шаги, необходимые для создания коммита. Попутно мы коснемся того, как каждая из областей диаграммы Git участвует в процессе создания коммита.

[Глава 3]

Ваш первый коммит

В *главе 2* вы узнали о различных областях, с которыми будете взаимодействовать при работе с Git: рабочий каталог, промежуточная область, история коммитов и локальный репозиторий. Мы построили диаграмму Git, состоящую из этих областей, и закончили главу добавлением первого файла в каталог вашего проекта `rainbow`.

В этой главе вы пройдете процесс создания коммита в проект Rainbow и увидите, как задействована в нем каждая из областей диаграммы Git. Я также представлю две важные команды, которые помогут вам в повседневной работе с Git. Первая команда позволит вам проверить состояние вашего рабочего каталога и промежуточной области, а вторая — просмотреть список коммитов.

Состояние рабочей среды

Теперь у вас есть каталог проекта с именем `rainbow`, внутри которого имеется каталог .git, а также один файл с именем rainbowcolors.txt в вашем рабочем каталоге. Промежуточная область и история коммитов пусты, поскольку вы еще не сделали ни одного коммита в репозитории `rainbow`. Все это можно увидеть на диаграмме 3.1.

ДИАГРАММА 3.1



К началу этой главы в каталоге проекта `rainbow` есть один неотслеживаемый файл, сохраненный в рабочем каталоге.

Зачем мы делаем коммиты?

В *главе 2* вы узнали, что коммит по сути представляет собой одну версию проекта. Каждый раз, когда вы хотите сохранить новую версию проекта, вы можете сделать коммит.

Коммиты очень важны, потому что они позволяют вам создать резервную копию вашей работы и избежать разочарования, связанного с потерей несохраненных изменений. После того как вы сделали текущий коммит, все внесенные изменения сохраняются, и вы всегда сможете вернуться и просмотреть этот коммит, чтобы увидеть, как выглядел ваш проект на тот момент.

Не существует строгого правила, предписывающего, когда именно делать коммиты. Это может зависеть от многих факторов, например, работаете ли вы над проектом самостоятельно или с другими людьми, а также над каким типом проекта вы работаете (например, пишете ли вы код, который необходимо компилировать, или документацию с описанием функций). Наконец, если вы работаете в команде, это может зависеть от рабочего процесса, которого придерживается ваша команда, и от того, какие соглашения в ней приняты (например, как члены команды используют ветки, которые мы обсудим в *главе 4*).

Среди пользователей Git в ходу присказка про коммиты: "Делай раньше, делай чаще". На мой взгляд, это хороший совет для начинающих работать с Git. На этом этапе лучше иметь слишком много коммитов, чем слишком мало. Избыток коммитов не должен вас пугать. Git предоставляет подготовленным пользователям дополнительные инструменты для очистки коммитов.

Два шага, чтобы сделать коммит

Что ж, теперь вы точно знаете, зачем нужны коммиты, и пора узнать, как они делаются. Создание коммита представляет собой двухэтапный процесс:

1. Добавьте все файлы, которые хотите включить в будущий коммит, в промежуточную область.
2. Сделайте коммит в сопровождении поясняющего сообщения.

На протяжении всего этого процесса вы будете взаимодействовать с четырьмя различными областями Git, представленными в *главе 2*: локальным репозиторием, рабочим каталогом, промежуточной областью и историей коммитов.

В процессе коммита вам пригодится команда `git status`. Помимо прочего, она сообщает вам состояние рабочего каталога и промежуточной области. Это полезно, поскольку в проекте с большим количеством файлов легко потерять представление о том, в каком состоянии находятся файлы в вашем рабочем каталоге (т. е. какие из них вы редактировали) и какие файлы вы добавили в промежуточную область.

ЗАПОМНИТЕ КОМАНДУ

git status

Показывает состояние рабочего каталога и промежуточной области.

В проекте с большим количеством файлов может быть сложно запомнить, какие файлы не отслеживаются, какие отслеживаются и какие файлы вы редактировали. На данный момент в проекте Rainbow у вас есть только один файл. Ситуацию с большим количеством файлов иллюстрирует пример 3.1 проекта Book.

Пример 3.1 проекта Book

Проект Book, над которым я работаю, состоит из 10 файлов, по одному на каждую главу книги. Поработав некоторое время над книгой между коммитами, я могу забыть, какие файлы глав я редактировала. Я знаю, что работала над главами 2 и 3, но внесла ли я какие-либо изменения в главу 4? Если да, то я не хочу их потерять.

В этом случае мне поможет команда `git status`, сообщающая, какие файлы глав я отредактировала, какие добавила в промежуточную область, а какие еще предстоит добавить.

Команда `git status` только предоставляет информацию; она никогда ничего не меняет в вашем репозитории. Смело выполняйте ее в любое время, пока вы работаете над проектом Git, чтобы узнать больше о состоянии рабочего каталога и промежуточной области.

Выполните действия 3.1, чтобы попрактиковаться в использовании команды `git status` и проверить состояние рабочего каталога и промежуточной области в репозитории `rainbow`. Для того чтобы выполнить шаги, описанные в этом руководстве, вы должны находиться в каталоге проекта `rainbow`, как показано в приглашении командной строки.

ДЕЙСТВИЯ 3.1

1
```
rainbow $ git status
On branch main

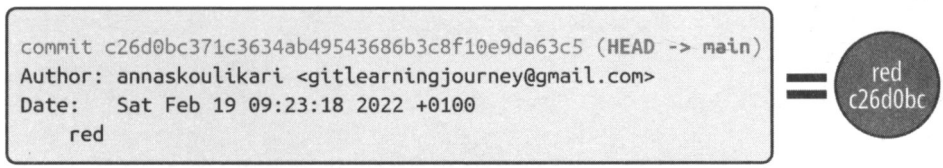
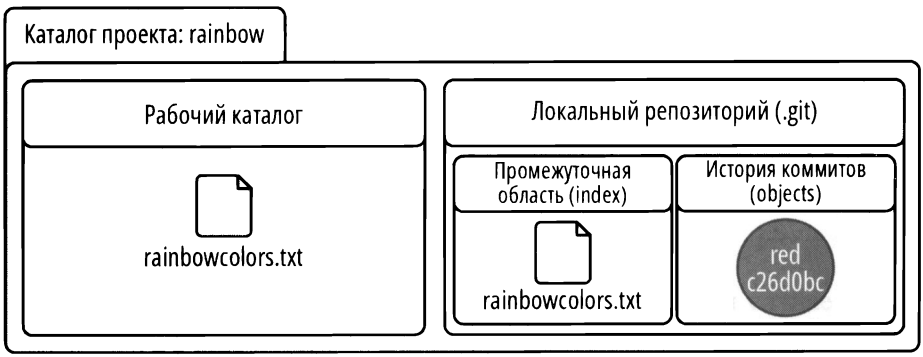
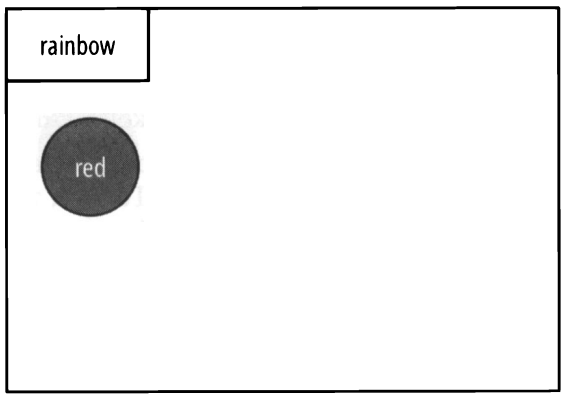
No commit yet

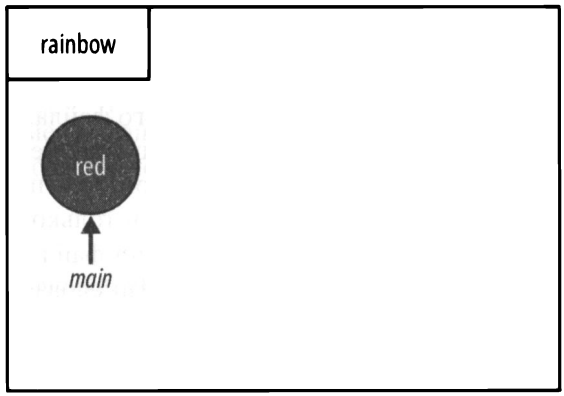
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  rainbowcolors.txt

nothing added to commit but untracked files present (use "git
add" to track)
```

(Текущая ветка main

Коммитов пока нет

Неотслеживаемые файлы:

(используйте "git add <file>..." для добавления объектов коммита)
rainbowcolors.txt

в коммит ничего не добавлено, но есть неотслеживаемые файлы
(используйте "git add" для отслеживания))

На что обратить внимание:

- вывод команды `git status` информирует вас, что коммитов еще нет. Другими словами, в настоящее время история коммитов не содержит никаких записей;
- файл rainbowcolors.txt является неотслеживаемым;
- Git дает инструкции по добавлению неотслеживаемого файла в промежуточную область:

  `use "git add <file>..." to include in what will be committed`

  (используйте "git add <file>..." для добавления объектов коммита)

В *главе 2* я упоминала, что rainbowcolors.txt — это неотслеживаемый файл, и чтобы он стал отслеживаемым, его необходимо добавить в промежуточную область и включить в коммит. Итак, давайте выполним первый шаг в процессе коммита, т. е. добавим в промежуточную область файлы, которые вы хотите включить в коммит.

Добавление файлов в промежуточную область

Для того чтобы добавить файлы в промежуточную область, воспользуйтесь командой `git add`. Если вы хотите добавить только отдельные файлы, которые вы отредактировали, можете передать имя файла или имена файлов команде `git add` в качестве аргументов. Для того чтобы добавить в свой рабочий каталог все файлы, которые вы редактировали или изменили, выполните команду `git add` с опцией `-A` (которая означает all — все). Это может быть полезно, если вы отредактировали много файлов и не хотите вводить каждое имя файла по отдельности в командную строку.

ЗАПОМНИТЕ КОМАНДУ

git add <имя файла>

Добавляет один файл в промежуточную область

git add <имя файла> <имя файла> ...

Добавляет несколько файлов в промежуточную область

git add -A

Добавляет в промежуточную область все файлы в рабочем каталоге, которые были отредактированы или изменены

Как упоминалось в *главе 2*, промежуточная область позволяет вам выбирать, какие обновленные файлы (или изменения) будут включены в ваш следующий коммит. Общее правило — объединять связанные изменения в группу. Это позволяет вам сохранять ваши коммиты более организованными. Как вы увидите в следующем разделе, каждый коммит также имеет связанное с ним сообщение о коммите. Обычно оно содержит описание того, что было обновлено в конкретном коммите.

Первый шаг процесса позволяет вам очень точно определить, что вы включаете в коммит. Это означает, что вы можете редактировать множество файлов в своем проекте, но вам не обязательно сохранять их все в одном коммите. Рассмотрим пример 3.2 проекта Book.

Пример 3.2 проекта Book

Проект Book, над которым я работаю, состоит из 10 глав, представленных 10 файлами. Представьте себе ситуацию, когда я работаю над главами 1–3 (поэтому я редактирую chapter_one.txt, chapter_two.txt и chapter_three.txt).

Если я решу, что работа, которую я проделала над главой 2, готова к коммиту (сохранению), но работу над главами 1 и 3 я не хочу включать в следующий коммит, то я могу добавить обновленную версию главы 2 в промежуточную область без добавления туда обновленных версий файлов глав 1 и 3. Это означает, что только изменения в файле главы 2 будут включены в мой следующий коммит и официально "сохранены" в локальном репозитории.

На примере 3.2 проекта Book вы убедились, что промежуточная область дает вам полный контроль над тем, как выглядят сохраненные версии вашего проекта (ваши коммиты).

В проекте Rainbow файл rainbowcolors.txt — это первый файл, который вы добавите в промежуточную область. Это означает, что при добавлении данного файла будет создан файл index (который представляет промежуточную область в каталоге .git). Как вы узнали из *главы 2*, этот файл не появится, пока вы не добавите какой-либо файл в промежуточную область. Файл index сохраняется в двоичном формате, а это означает, что фактическое содержимое выглядит для человека тарабарщиной и его нелегко понять. Для наших целей достаточно понимать, что он представляет собой промежуточную область. Выполните действия 3.2, чтобы добавить файл rainbowcolors.txt в промежуточную область и создать файл index.

ДЕЙСТВИЯ 3.2

1. Для того чтобы вы могли видеть создаваемый файл index, в окне файловой системы должен отображаться каталог вашего проекта `rainbow` с включенным просмотром скрытых файлов и каталогов и видимым содержимым каталога .git.
2. `rainbow $ git add rainbowcolors.txt`

3
```
rainbow $ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
  new file: rainbowcolors.txt
```

(На ветке main

Коммитов пока нет

Изменения, включаемые в коммит:

(используйте "git rm --cached <file>..." для отмены)

новый файл: rainbowcolors.txt))

4 Используя окно файловой системы, просмотрите содержимое каталога .git и найдите вновь созданный файл index.

На что обратить внимание:

- вы добавили файл rainbowcolors.txt в промежуточную область, и на шаге 3 он указан в строках `Changes to be committed:` ("Изменения, включаемые в коммит:"). Посмотрите, как это выглядит на диаграмме 3.2.

ДИАГРАММА 3.2



Каталог проекта `rainbow` после добавления файла rainbowcolors.txt в промежуточную область.

Файл rainbowcolors.txt теперь находится как в рабочем каталоге, так и в промежуточной области. Это связано с тем, что команда `git add` не *перемещает* файл из рабочего каталога в промежуточную область. Она *копирует* туда файл из рабочего каталога.

Когда файл rainbowcolors.txt оказался в промежуточной области, можно переходить ко второму шагу процесса коммита, который заключается в фактическом выполнении коммита в сопровождении связанного сообщения.

Выполнение коммита

Важно отметить, что *commit* (коммит) — это и глагол, и существительное. В Git глагол "коммит" означает сохранение чего-либо, а существительное — версию нашего проекта. Итак, сделать коммит — значит сохранить версию проекта.

Для того чтобы сделать коммит, введите команду `git commit` и передайте опцию `-m` (которая означает *message*, т. е. сообщение), введя сообщение в кавычках. Сообщение обычно содержит краткое описание изменений, внесенных вами в эту версию проекта.

ЗАПОМНИТЕ КОМАНДУ

git commit -m "<сообщение>"

Создает новый коммит с сообщением о коммите.

Рассмотрим пример сообщения о коммите в примере 3.3 проекта Book.

Пример 3.3 проекта Book

Если бы в моем проекте Book я работала над главой 2, добавила в промежуточную область только этот файл и захотела сделать коммит только с обновлениями главы 2, я могла бы сопроводить коммит сообщением "Обновлена глава 2".

Имейте в виду, что у разных людей и команд могут быть разные правила относительно того, что включать в сообщение о коммите. Если вы работаете над проектом Git с другими людьми, вам следует проконсультироваться со своими коллегами, чтобы точно знать, что они ожидают увидеть в этих сообщениях.

В проекте Rainbow в качестве сообщения коммита вы просто передаете имя цвета, который добавляете в проект, чтобы мы могли легко представить коммиты на диаграммах визуализации. Поскольку первым цветом, который вы добавили в список цветов радуги, был красный, в действиях 3.3 вы сделаете коммит с сообщением "red" ("красный").

ДЕЙСТВИЯ 3.3

1
```
rainbow $ git commit -m "red"
[main (root-commit) c26d0bc] red
1 file changed, 1 insertion(+)
create mode 100644 rainbowcolors.txt
```

На что обратить внимание:

- в выводе команды `git commit` показаны первые семь символов хеша красного коммита, который в этой книге — `c26d0bc`. Первые семь символов хеша вашего коммита будут другими, поскольку, как вы узнали из раздела "Что такое коммит?", хеши коммитов уникальны.

На диаграмме 3.3 вы можете видеть красный коммит в истории коммитов с первыми семью символами хеша коммита.

ДИАГРАММА 3.3



Каталог проекта `rainbow` после того, как вы сделаете красный коммит.

В предыдущей главе вы также узнали, что после добавления нового неотслеживаемого файла в промежуточную область и включения его в коммит он становится отслеживаемым файлом, поскольку Git теперь знает о нем. Поэтому файл rainbowcolors.txt стал отслеживаемым файлом.

Остальная часть вывода, следующего за командой `git commit`, не очень важна для наших целей. Она просто дает больше информации о том, что изменилось в этом коммите, и, по моему опыту, большинству пользователей Git это не обязательно изучать подробно.

Теперь, когда вы сделали первый коммит в репозитории `rainbow`, давайте посмотрим, какую информацию можно найти об этом коммите в истории коммитов.

Просмотр списка коммитов

Для того чтобы просмотреть список коммитов в истории коммитов, выполните команду `git log`. Команда `git log` выводит список коммитов в локальном репозитории в обратном хронологическом порядке. Она отображает четыре части информации о каждом коммите:

1. Хеш коммита.
2. Имя автора и адрес электронной почты.

3. Дата и время коммита.
4. Сообщение коммита.

ЗАПОМНИТЕ КОМАНДУ

git log

Выводит список коммитов в обратном хронологическом порядке.

ПРИМЕЧАНИЕ

Если вывод команды `git log` выходит за пределы окна командной строки, для просмотра остальных коммитов необходимо нажать клавишу ввода (<Enter> или <Return>) или использовать клавишу <↓>. Для того чтобы выйти из команды, введите Q и нажмите клавишу ввода. Сейчас у вас есть только один коммит, поэтому вывод `git log` будет очень коротким. Однако по мере добавления коммитов в проект Rainbow выходные данные `git log` будут становиться длиннее, и вам придется воспользоваться упомянутыми клавишами.

Выполните действия 3.4, чтобы попрактиковаться в просмотре списка коммитов.

ДЕЙСТВИЯ 3.4

1
```
rainbow $ git log
commit c26d0bc371c3634ab49543686b3c8f10e9da63c5 (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

На что обратить внимание:

- вывод `git log` показывает, что на данный момент у вас есть только один коммит — красный;
- полный хеш красного коммита в этой книге — `c26d0bc371c3634ab49543686b3c8f10e9da63c5`. Хеш коммита в вашем проекте будет другим;
- имя и адрес электронной почты автора красного коммита должны совпадать с именем и адресом, которые вы указали в *главе 1* при настройке переменных конфигурации Git `user.name` и `user.email`;
- отображаются дата и время коммита;
- под датой находится сообщение коммита, которое в данном случае состоит из одного слова `red`.

На диаграммах визуализации я буду представлять коммиты в виде кругов. Каждый коммит будет отображаться цветом, используемым в сообщении о коммите, и помечаться его полным именем или аббревиатурой (например, R для красного комми-

та). Имейте в виду, что каждый из этих кружков представляет собой отдельный коммит, который вы можете увидеть в выводе `git log`. Это соответствие наглядно показано на рис. 3.1.



Рис. 3.1. В этой книге коммиты представлены на диаграммах кружками

Есть еще одна деталь, которая присутствует в выводе `git log` после выполнения действий 3.4 и которую мы не рассмотрели. Рядом с хешем коммита в круглых скобках вы можете увидеть текст `HEAD -> main`. Здесь `main` — это ветка; в следующей главе вы поймете, что такое ветки и как с ними работать, а также узнаете, что такое `HEAD`.

Краткое содержание главы

В этой главе были представлены два этапа создания коммита, а именно добавление файлов в промежуточную область и выполнение коммита с сообщением. Вы продвинулись вперед и сделали свой первый коммит в репозитории `rainbow`. В этом вам помогала команда `git status`, которая вводит информацию о состоянии файлов в рабочем каталоге и промежуточной области.

Вы также узнали, как просмотреть список коммитов в локальном репозитории с помощью команды `git log`. В выводе `git log` вы увидели упоминание `main` в скобках рядом с хешем коммита. В *главе 4* вы узнаете, что `main` — это *ветка*, и выясните, почему и как в Git используются ветки.

[Глава 4]

Ветки

В предыдущей главе вы узнали о том, что такое коммит, и сделали свой первый коммит в репозитории `rainbow`.

В этой главе вы узнаете, что такое ветки и почему мы их используем. Вы продолжите делать коммиты в репозитории `rainbow` и увидите, как это повлияет на ветки вашего проекта. Наконец, вы создадите новую ветку и научитесь переключаться на нее. Кроме того, в ходе создания нескольких коммитов в репозитории `rainbow` вы узнаете о неизмененных и измененных файлах и о том, как коммиты связаны друг с другом.

Состояние локального репозитория

В *главе 2* мы построили диаграмму Git, включающую четыре важные области Git: рабочий каталог, промежуточную область, историю коммитов и локальный репозиторий. На диаграмме 4.1 представлена диаграмма Git, отражающая состояние репозитория `rainbow` в начале этой главы.

ДИАГРАММА 4.1



Диаграмма Git, показывающая состояние репозитория в начале *главы 4*, с единственным коммитом — красным (`red`).

ПРИМЕЧАНИЕ

С этого момента диаграммы состояния будут отображать только имя цвета в коммите или сокращение имени. Я больше не буду включать первые семь символов хеша коммита.

Для того чтобы сосредоточиться на истории коммитов, я собираюсь представить новую диаграмму, которая называется диаграммой репозитория. Она состоит только из схематического изображения истории коммитов репозитория, а также соответствующих веток и ссылок. Локальный репозиторий представлен прямоугольником с именем репозитория в левом верхнем углу. На диаграмме 4.2 показано текущее состояние репозитория `rainbow` в форме диаграммы репозитория.

ДИАГРАММА 4.2



Текущее состояние репозитория `rainbow` с единственным коммитом — красным (`red`).

Почему мы используем ветки?

Прежде чем мы займемся детальным изучением веток в Git, я хочу объяснить, почему они так полезны. Есть две основные причины использовать ветки:

- возможность работать над одним и тем же проектом по-разному;
- возможность нескольким людям одновременно работать над одним проектом.

Ветку можно рассматривать как линию развития. Проект Git может иметь несколько веток (или линий развития). Каждая из этих веток представляет собой автономную версию проекта. Разные проекты Git могут использовать ветки по-разному, в зависимости от потребностей людей, работающих над проектом.

Одним из распространенных сценариев работы с ветками является поддержка одной официальной главной линии разработки (основной или первичной ветки) и создание вторичных веток, называемых *тематическими* (topic) или *функциональными* (feature) ветками, которые используются для работы только над определенной частью проекта. Эти тематические ветки недолговечны; в конечном итоге они объ-

единяются или включаются обратно в основную ветку, а затем удаляются. Два процесса, которые вы можете использовать для интеграции одной ветки в другую, называются *слиянием* и *перебазированием*. Мы рассмотрим их более подробно в *главах 5, 9–12*. (Да, это большие темы!)

Модель с одной основной линией развития — это подход, который применяется в проекте Rainbow. Пример 4.1 проекта Book поможет вам лучше понять, как разделение на ветки находит применение в реальных проектах.

Пример 4.1 проекта Book

Предположим, официальной веткой моего проекта Book является ветка `main`. Я не хочу добавлять главу в официальную линию разработки, пока мой редактор не проверит и не одобрит ее. Таким образом, каждый раз, когда я работаю над главой, я могу создать дополнительную ветку, работать над ней, а затем отправить ее на проверку моему редактору. Как только он одобрит работу, которую я проделала в этой вторичной ветке, я смогу объединить эту ветку с веткой `main`.

Если в какой-то момент я решу работать над книгой с соавтором, каждый из нас сможет работать над своими вторичными ветками. Тогда и только тогда, когда работа над определенной вторичной веткой будет одобрена как другим автором, так и редактором, она будет считаться готовой к объединению с веткой `main`.

Теперь, когда у вас есть представление о том, почему мы используем ветки, давайте более детально разберемся в том, как они работают в Git.

Что такое ветки в Git?

Ветки в Git — это перемещаемые указатели на коммиты. Когда вы получаете список коммитов в локальном репозитории с помощью команды `git log`, вы можете увидеть информацию о том, какие ветки указывают на те или иные коммиты. Выполните действия 4.1, чтобы увидеть, как это выглядит в репозитории `rainbow`.

ДЕЙСТВИЯ 4.1

1
```
rainbow $ git log
commit c26d0bc371c3634ab49543686b3c8f10e9da63c5 (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

На что обратить внимание:

- в выводе команды `git log` рядом с хешем коммита в круглых скобках вы видите `HEAD -> main`.

Ветка или ветки, которые появляются в круглых скобках рядом с конкретным хешем коммита в выводе `git log`, являются ветками, указывающими на этот коммит.

ПРИМЕЧАНИЕ

`HEAD` (заглавными буквами) не является веткой. Мы обсудим, что это такое, в *разд. "Что такое HEAD?" далее в этой главе.*

В репозитории `rainbow` ветка `main` указывает на красный коммит. Это показано на диаграмме 4.3.

ДИАГРАММА 4.3



Ветка `main` указывает на красный коммит в репозитории `rainbow`.

Далее в этой главе вы продолжите делать коммиты в ветке `main` и увидите, как эта ветка будет меняться, указывая на новые коммиты.

ПРИМЕЧАНИЕ

Ветка `main` репозитория `rainbow` является локальной. В первой части этой книги, когда вы работаете только с локальным репозиторием в проекте Rainbow, вы будете иметь дело лишь с локальными ветками. Во второй части книги, когда мы познакомимся с удаленными репозиториями, вы узнаете о концепции удаленных ветвей и ветвей с удаленным отслеживанием.

Для того чтобы лучше понять концепцию ветвей как подвижных указателей на коммиты, выполните действия 4.2.

ДЕЙСТВИЯ 4.2

1. В окне файловой системы перейдите в каталог проекта `rainbow`.
2. Отобразите все скрытые файлы и каталоги проекта `rainbow`. Дополнительная информация о том, как просмотреть скрытые файлы и каталоги, представлена в *разд. "Просмотр содержимого каталогов" главы 1.*

3. В каталоге проекта `rainbow` в окне файловой системы перейдите в раздел .git > refs > heads > main.
4. Откройте файл main. В macOS файл автоматически откроется в базовом текстовом редакторе TextEdit. В Microsoft Windows вы можете открыть файл с помощью простого текстового редактора Блокнот.

   Вы будете использовать TextEdit и Блокнот только для просмотра содержимого некоторых файлов в каталоге .git. Эти текстовые редакторы не имеют отношения к текстовому редактору, который вам понадобится для работы с файлами в проекте Rainbow.

На что обратить внимание:

- на шаге 4 внутри файла main вы увидите хеш красного коммита в вашем репозитории `rainbow`.

На шаге 3 действий 4.2, чтобы добраться до основного файла, вы перешли в каталог .git, каталог refs, а затем в каталог heads. Сокращение refs означает *references* (ссылки). В каталоге head хранится файл для каждой локальной ветки вашего локального репозитория. На данный момент у вас есть только одна локальная ветка `main`, поэтому в этом каталоге находится только один файл. Можно сказать, что в этом файле хранится "голова" (head) ветки — другими словами, последний коммит в этой ветке.

Теперь, когда вы получили представление о том, что такое ветки и как они работают, давайте кратко рассмотрим соглашения об именах для основной ветки.

Немного истории Git: *master* и *main*

Обычно, когда вы инициализируете локальный репозиторий, используя только команду `git init` без параметров, Git по умолчанию создает ветку с именем `master`. Однако в последние годы слово master приобрело негативную окраску, связанную с историей рабовладения, и считается недостаточно политкорректным, поэтому бóльшая часть сообщества Git решила перейти на использование `main` (или других имен) в качестве имени ветки по умолчанию.

Вот почему в *главе 2* при инициализации репозитория `rainbow` вы использовали команду `git init` с опцией `-b` и передавали значение `main`. Само по себе слово main ни в коем случае не является особенным; вы могли бы дать первой ветке любое другое имя, передав иное значение команде `git init -b`. Когда вы сделали первый коммит в репозитории `rainbow`, это действие обновило ветку `main`, чтобы она указывала на первый коммит. В проекте Rainbow ветка `main` является основной линией развития.

В процессе обучения работе с Git вы встретите множество учебных ресурсов, которые по-прежнему используют ветку `master`. Важно понимать, что в этой ветке нет ничего особенного; это просто имя по умолчанию для первой ветки, созданной в Git.

На этом этапе вы готовы добавить новый цвет в файл rainbowcolors.txt и сделать еще один коммит. Но прежде чем мы продолжим, я хотела бы рассказать о нескольких дополнительных состояниях, в которых может находиться файл в проекте Git.

Неизмененные и измененные файлы

В *главе 2* я рассказала о делении файлов на неотслеживаемые и отслеживаемые. Git знает о существовании файла rainbowcolors.txt, поскольку он был включен в коммит и, следовательно, является отслеживаемым файлом.

В свою очередь, отслеживаемые файлы в рабочем каталоге могут находиться в одном из двух состояний. *Неизмененные файлы* — это файлы в рабочем каталоге, которые не редактировались с момента последнего коммита. После того как файл в рабочем каталоге отредактирован (и сохранен в текстовом редакторе), он становится *измененным*. С момента вашего последнего коммита вы не редактировали файл rainbowcolors.txt, следовательно, это неизмененный файл.

> **ПРИМЕЧАНИЕ**
>
> Для того чтобы Git знал, что файл был отредактирован, он должен быть сохранен в текстовом редакторе. Если вы внесли изменения в файл, но не сохранили эти изменения в текстовом редакторе, Git будет рассматривать его как неизмененный файл.

В *главе 3* вы узнали о команде `git status`, которая показывает состояние рабочего каталога и промежуточной области, а также разницу между ними. Команда `git status` фактически показывает список всех измененных файлов и сообщает, были ли они добавлены в промежуточную область. Однако она не перечисляет файлы, оставшиеся без изменений.

Выполните действия 4.3, чтобы с помощью команды `git status` увидеть, что rainbowcolors.txt является неизмененным файлом, затем отредактируйте и сохраните его, и снова выполните `git status`, чтобы увидеть, как изменится вывод команды.

ДЕЙСТВИЯ 4.3

1. ```
   rainbow $ git status
   On branch main
   nothing to commit, working tree clean
   ```
   (На ветке main
   нет ничего для коммита, рабочее дерево пустое)

2. В папке проекта `rainbow` в текстовом редакторе откройте файл rainbowcolors.txt, добавьте текст "Оранжевый — второй цвет радуги." в строке 2 и сохраните файл.
   ```

3
```
rainbow $ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
directory)
   modified: rainbowcolors.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

(На ветке main
есть изменения, не подготовленные для коммита:
(используйте "git add <file>..." для обновления содержимого коммита)
(используйте "git restore <file>..." для отмены изменений в рабочем каталоге)
изменен: rainbowcolors.txt
никаких изменений не добавлено в коммит (используйте "git add" и/или "git commit -a"))

На что обратить внимание:

- на шаге 1 файл rainbowcolors.txt является неизмененным. Он еще не указан в выводе `git status`;
- на шаге 3 файл rainbowcolors.txt является измененным файлом. Теперь он указан в выводе `git status`;
- файл rainbowcolors.txt не подготовлен для коммита; другими словами, он не был добавлен в промежуточную область.

Вы только что видели, как файл rainbowcolors.txt превратился из неизмененного файла в измененный, когда вы отредактировали его и сохранили изменения. На рис. 4.1 показан пример того, как теперь должен выглядеть ваш файл rainbowcolors.txt.

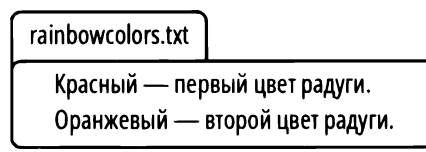


Рис. 4.1. Файл rainbowcolors.txt после добавления строки об оранжевом цвете

Выполнив действия 4.4, вы добавите файл rainbowcolors.txt в промежуточную область, чтобы его можно было включить в следующий коммит, и увидите, как изменится вывод `git status`.

ДЕЙСТВИЯ 4.4

1. `rainbow $ git add rainbowcolors.txt`
2. `rainbow $ git status`

```
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
   modified:   rainbowcolors.txt
```

(На ветке main
Изменения для включения в коммит:
(используйте "git restore --staged <file>..." для отмены)
изменен: rainbowcolors.txt)

На что обратить внимание:

- файл rainbowcolors.txt подготовлен для коммита; другими словами, он был добавлен в промежуточную область.

Вы только что завершили первый шаг процесса коммита, который заключается в добавлении файлов в промежуточную область. Далее вы сделаете еще один коммит и посмотрите, как это повлияет на вашу ветку `main`.

Выполнение коммитов в ветке

Пришло время сделать второй коммит в репозитории `rainbow`. На этот раз вы добавите оранжевый цвет, поэтому сообщение о коммите будет состоять из слова orange (оранжевый). Выполните действия 4.5, чтобы сделать коммит.

ПРИМЕЧАНИЕ

Если вывод команды `git log` выходит за пределы размера окна командной строки, необходимо нажать клавишу <Enter> (<Return>) или использовать клавишу <↓>, чтобы просмотреть остальные коммиты. Для выхода из команды необходимо ввести `Q`.

ДЕЙСТВИЯ 4.5

1. `rainbow $ git commit -m "orange"`

```
[main 7acb333] orange
1 file changed, 2 insertions(+), 1 deletion(-)
```

(1 файл изменен, 2 вставки (+), 1 удаление (–))

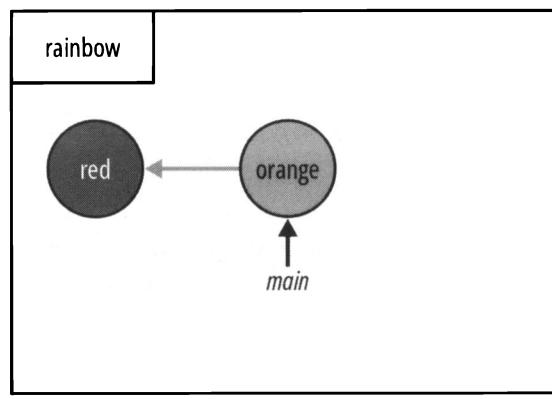
2
```
rainbow $ git log
commit 7acb333f08e12020efb5c6b563b285040c9dba93 (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange

commit c26d0bc371c3634ab49543686b3c8f10e9da63c5
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

На что обратить внимание:

- вы сделали новый коммит — "оранжевый". В репозитории `rainbow` в этой книге хеш для оранжевого коммита — `7acb333f08e12020efb5c6b563b285040c9dba93`. Хеш вашего коммита будет другим;
- текст `HEAD -> main` в круглых скобках теперь появился рядом с оранжевым коммитом.

На диаграмме 4.4 показано состояние репозитория `rainbow` после выполнения действий 4.5.

ДИАГРАММА 4.4



Репозиторий `rainbow` после того, как вы сделали оранжевый коммит.

На что обратить внимание:

- появился второй коммит, оранжевый;
- оранжевый коммит указывает на красный коммит;
- ветка `main` указывает на оранжевый коммит.

На диаграмме 4.4 вы можете видеть серую стрелку, указывающую от оранжевого коммита к красному коммиту. Эта серая стрелка представляет *родительскую ссылку*. Каждый коммит, кроме самого первого в репозитории, имеет родительский коммит (некоторые коммиты могут иметь более одного родителя; мы рассмотрим это в *главе 5*). Родительским коммитом оранжевого коммита является красный коммит; вот почему оранжевый коммит указывает на красный.

Эти родительские ссылки описывают, как коммиты связаны друг с другом. Понимание этих ссылок позволяет визуализировать историю коммитов и отслеживать, какая работа была проделана в тех или иных ветках.

ПРИМЕЧАНИЕ

На диаграммах серые стрелки используются для обозначения родительских ссылок, а черные стрелки — для обозначения указателей ветвей.

Для того чтобы проверить, какой коммит является родительским для данного коммита, вы можете использовать команду `git cat-file` с опцией `-p` и передать хеш коммита: `git cat-file -p <хеш_коммита>`. Большинство пользователей Git, вероятно, не будут использовать эту команду в своей повседневной работе, но поскольку это хороший инструмент обучения, будет полезно взглянуть на нее в деле.

Выполните действия 4.6, чтобы получить хеш родителя оранжевого коммита. Для этого вам нужно будет передать хеш оранжевого коммита команде `git cat-file -p`. Самый простой способ получить хеш конкретного коммита — просмотреть список коммитов, созданный командой `git log`, и скопировать весь хеш коммита в командной строке. В качестве альтернативы вы можете просмотреть выходные данные команды `git commit` в действиях 4.5 и передать первые семь символов хеша коммита, который там показан.

ДЕЙСТВИЯ 4.6

1. Получите хеш для оранжевого коммита (вы можете скопировать его из вывода `git log` в разд. 4.5). Вы должны передать этот хеш коммита в качестве аргумента команде `git cat-file -p` на шаге 2 данного руководства. Вы можете скопировать и вставить весь хеш коммита или просто ввести первые семь символов, как показано здесь.

2.
```
rainbow $ git cat-file -p 7acb333
tree 407fe6a858cd7f157405e013a088fdc1c61f0a40
parent c26d0bc371c3634ab49543686b3c8f10e9da63c5
author annaskoulikari gitlearningjourney@gmail.com
1645260127 +0100
committer annaskoulikari <gitlearningjourney@gmail.com>
1645260127 +0100

orange
```

На что обратить внимание:

- в выводе `git cat-file -p` вы можете видеть, что рядом со словом `parent` (родитель) приведен хеш красного коммита в этой книге. В вашем случае вы получите хеш вашего красного коммита.

К настоящему моменту вы сделали второй коммит в своем репозитории `rainbow` и на диаграмме 4.4 заметили, что когда вы делаете коммит в ветке, указатель ветки перемещается, указывая на последний коммит. Далее мы рассмотрим создание новых веток, чтобы вы могли работать над другими направлениями разработки.

Создание ветки

На данный момент в вашем репозитории `rainbow` есть только одна локальная ветка под названием `main`. Для того чтобы просмотреть ветки в локальном репозитории, вы можете использовать команду `git branch`. Для того чтобы создать новую ветку, вы можете передать этой команде имя еще не существующей ветки. Обратите внимание, что имена веток не могут содержать пробелы.

ЗАПОМНИТЕ КОМАНДУ

`git branch`

Выводит список локальных веток.

`git branch <имя_новой_ветки>`

Создает новую ветку.

Как и в случае с сообщениями коммита, как обсуждалось в *главе 3*, если вы работаете над проектом с другими людьми, вам следует проверить, существуют ли какие-либо правила, определяющие названия веток. Поскольку в проекте Rainbow нет конкретных правил именования веток, вы будете использовать обобщенное имя `feature` для ветки, которую вы создадите во врезке действий 4.7. Однако имейте в виду, что в реальном проекте Git имя ветки обычно будет более информативным и относится к функции или теме, над которой вы работаете.

ДЕЙСТВИЯ 4.7

1.
```
rainbow $ git branch
* main
```
2.
```
rainbow $ git branch feature
```
3.
```
rainbow $ git branch
  feature
* main
```

4 rainbow $ **git log**

```
commit 7acb333f08e12020efb5c6b563b285040c9dba93 (HEAD -> main,
feature)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange

commit c26d0bc371c3634ab49543686b3c8f10e9da63c5
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

5 С помощью окна файловой системы перейдите в .git > refs > heads, чтобы увидеть, какие файлы там сейчас находятся.

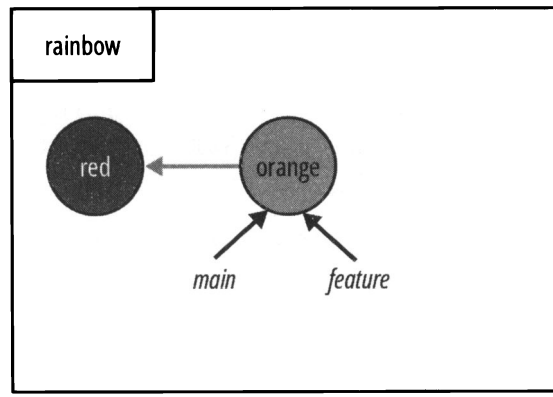
6 Откройте файл feature. Файл должен содержать хеш коммита.

На что обратить внимание:

- на шаге 2 вы создали новую ветку под названием `feature`, которая указывает на оранжевый коммит.

Состояние репозитория `rainbow` после выполнения действий 4.7 показано на диаграмме 4.5.

ДИАГРАММА 4.5



Репозиторий `rainbow` после создания ветки `feature`.

На диаграмме 4.5 вы можете видеть, что теперь есть две стрелки, обозначающие ветки `main` и `feature`, и обе они указывают на оранжевый коммит. Новая ветка изначально будет указывать на коммит, который вы использовали при создании вет-

ки. В этом случае вы можете сказать, что вы "создали ветку `feature` из ветки `main`". Вот почему ветки `main` и `feature` теперь указывают на один и тот же коммит.

В выводе `git log` из действий 4.7 в скобках рядом с оранжевым коммитом вы должны увидеть `HEAD -> main, feature`. Как вы знаете, `main` и `feature` — это ветки, но что такое `HEAD`?

Что такое *HEAD*?

В любой момент времени вы просматриваете конкретную версию своего проекта. Таким образом, вы находитесь в определенной ветке, которая указывает на коммит. `HEAD` — это просто указатель, который сообщает вам, в какой ветке вы находитесь. Имя `HEAD` всегда пишется заглавными буквами, но это всего лишь соглашение, а не аббревиатура.

> **ПРИМЕЧАНИЕ**
>
> Может случиться так, что вы окажетесь в коммите, на который не указывает ветка. Git называет это "отсоединенным состоянием `HEAD`". Мы рассмотрим его подробнее в *разд. "Проверка коммитов" главы 5.*

Выполните действия 4.8, чтобы узнать, что такое `HEAD`, просмотрев каталог .git.

> **ДЕЙСТВИЯ 4.8**
>
> 1. В окне файловой системы перейдите в rainbow > .git > HEAD.
> 2. Откройте файл HEAD. Его содержимое должно состоять из единственной строки `ref: refs/heads/main`.

На что обратить внимание:

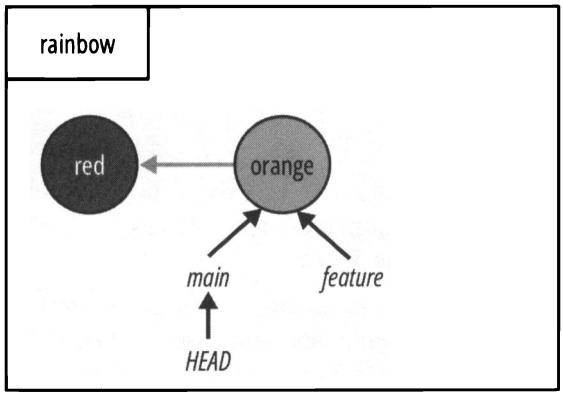
- файл HEAD содержит строку `ref: refs/heads/main`, которая является ссылкой на основной файл, представляющий ветку `main`.

Это состояние показано на диаграмме 4.6.

На диаграмме 4.6 стрелка от `HEAD` указывает на `main`. Это означает, что вы сейчас находитесь в ветке `main`.

> **ПРИМЕЧАНИЕ**
>
> `HEAD` (заглавными буквами) не следует путать с каталогом head, который можно найти в .git > refs > heads. В каталоге head хранится файл для каждой локальной ветки вашего локального репозитория, а `HEAD` указывает, в какой ветке вы находитесь, ссылаясь на один из файлов внутри каталога head. Вы можете отличить их, потому что `HEAD` всегда пишется заглавными буквами (и моноширинным шрифтом в этой книге).

ДИАГРАММА 4.6



В репозитории `rainbow` `HEAD` указывает на `main`.

Другой способ узнать, в какой ветке вы сейчас находитесь, — посмотреть выходные данные либо команды `git branch`, либо команды `git log`. В выводе `git branch` рядом с веткой, в которой вы находитесь, будет стоять звездочка. Если вы посмотрите на результат выполнения действий 4.7, вы увидите, что звездочка находится рядом с веткой `main`. В выводе `git log` слово `HEAD` внутри круглых скобок укажет на ветку, в которой вы находитесь.

Вы успешно создали новую ветку и готовы начать с ней работать, но на данный момент вы все еще находитесь в ветке `main`. Далее вы переключите ветки, переместив указатель `HEAD` на новую ветку `feature`.

Переключение веток

Для того чтобы приступить к работе с другой веткой (или направлением разработки) проекта Git, вам необходимо переключиться на эту ветку.

В настоящее время у вас есть две ветки: `main` и `feature`. Но, как вы только что видели, создание ветки в Git не означает, что вы автоматически окажетесь на этой ветке. Вы должны явно указать Git, что хотите переключиться на другую ветку. Это можно сделать с помощью команды `git switch` или `git checkout`, передав имя ветки, на которую вы хотите переключиться.

ПРИМЕЧАНИЕ

Если у вас версия Git старше версии 2.23, вам недоступна команда `git switch`, и придется использовать команду `git checkout`. Команда `git checkout` доступна всем пользователям Git.

ЗАПОМНИТЕ КОМАНДУ

git switch <имя_ветки>

Переключает ветку.

git checkout <имя_ветки>

Переключает ветку.

Единственное назначение команды `git switch` — переключение веток, но команда `git checkout` может делать больше. Подробнее о ней будет рассказано в *разд. "Проверка коммитов" главы 5*.

С этого момента в последующих разделах книги мы будем использовать команду `git switch`, поскольку это специализированная команда, добавленная в последние версии Git именно для переключения веток. Однако вы всегда можете использовать вместо нее команду `git checkout`, поскольку они эквивалентны.

Команда `git switch` (или `git checkout`) при переключении веток выполняет три действия:

1. Изменяет указатель `HEAD`, чтобы он указывал на ветку, на которую вы переключаетесь.
2. Заполняет промежуточную область *снимком коммита*, на который вы переключаетесь.
3. Копирует содержимое промежуточной области в рабочий каталог.

ПРИМЕЧАНИЕ

Если вам нужно вспомнить, что такое промежуточная область и рабочий каталог, вернитесь к *разд. "Области Git" главы 2*.

Короче говоря, когда вы меняете ветки, вы в конечном итоге меняете рассматриваемый коммит при условии, что две ветки указывают на два разных коммита. На данный момент обе ветки в репозитории `rainbow` указывают на один и тот же коммит, поэтому произойдет только первое действие, и коммит, на котором вы находитесь, не изменится. В *главе 5* вы рассмотрите пример, в котором выполняются все три вышеперечисленных действия.

Выполните действия 4.9, чтобы переключиться на ветку `feature`.

ДЕЙСТВИЯ 4.9

1
```
rainbow $ git branch
  feature
* main
```
2
```
rainbow $ git switch feature
Switched to branch 'feature'
```

(Переключено на ветку 'feature')

3
```
rainbow $ git branch
* feature
  main
```

4
```
rainbow $ git log
commit 7acb333f08e12020efb5c6b563b285040c9dba93 (HEAD ->
feature, main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange

commit c26d0bc371c3634ab49543686b3c8f10e9da63c5
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

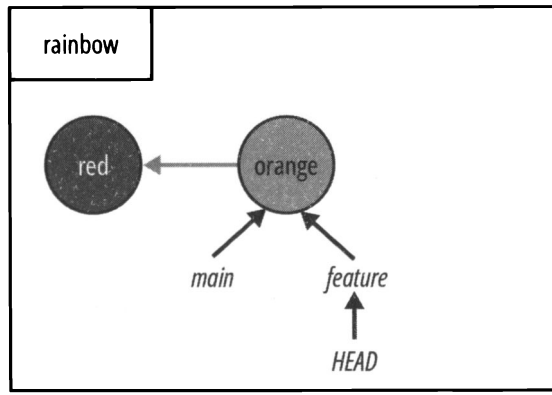
5 В окне файловой системы перейдите в rainbow > .git > HEAD и откройте файл HEAD в новом окне. Вы увидите, что содержимое файла изменилось и теперь относится к ветке функций refs/heads/feature.

На что обратить внимание:

- на шаге 4 выходные данные команды `git log` говорят о том, что `HEAD` указывает на ветку `feature`;
- вы переключились на ветку `feature`, поэтому сейчас вы будете работать именно с ней.

Эти наблюдения проиллюстрированы на диаграмме 4.7.

ДИАГРАММА 4.7



Репозиторий `rainbow` после переключения на ветку `feature`.

Как видите, переключение веток изменило содержимое файла HEAD в каталоге .git. Это показано на рис. 4.2.

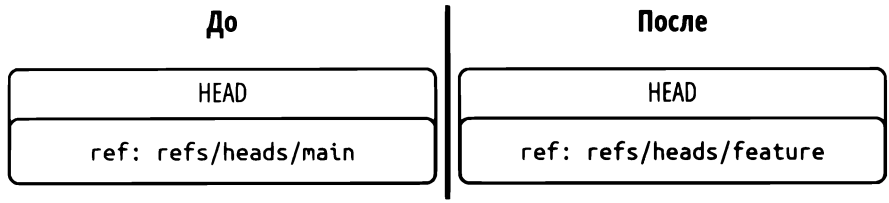


Рис. 4.2. Содержимое файла HEAD до и после переключения с ветки `main` на ветку `feature` в репозитории `rainbow`

Итак, вы переключились на ветку `feature`. Теперь посмотрим, что происходит, когда вы над ней работаете.

Работа в отдельной ветке

Теперь вы находитесь в ветке `feature`, и, выполнив действия 4.10, вы добавите желтый цвет в репозиторий `rainbow`.

ДЕЙСТВИЯ 4.10

1. В каталоге проекта rainbow в текстовом редакторе в файле rainbowcolors.txt добавьте текст "Желтый — третий цвет радуги." в строке 3. Затем сохраните файл.
2. `rainbow $ git add rainbowcolors.txt`
3. `rainbow $ git commit -m "yellow"`

```
[feature fc8139c] yellow
 1 file changed, 2 insertions(+), 1 deletion(-)
```

4. `rainbow $ git log`

```
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (HEAD ->
feature)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow

commit 7acb333f08e12020efb5c6b563b285040c9dba93 (main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange
```

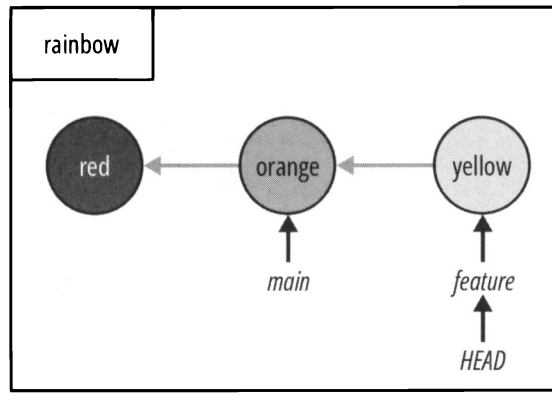
```
commit c26d0bc371c3634ab49543686b3c8f10e9da63c5
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

На что обратить внимание:

- ветка `feature` указывает на последний коммит — желтый;
- ветка `main` по-прежнему указывает на оранжевый коммит.

Эти наблюдения проиллюстрированы на диаграмме 4.8.

ДИАГРАММА 4.8



Репозиторий `rainbow` после создания желтого коммита.

Как упоминалось ранее, когда вы делаете коммит, именно ветка, в которой вы сейчас находитесь, обновляется, указывая на новый коммит. Ветки `main` и `feature` больше не указывают на один и тот же коммит, поскольку ветка `feature` была обновлена и теперь указывает на новый желтый коммит. В то же время `HEAD` продолжает указывать на ветку `feature`.

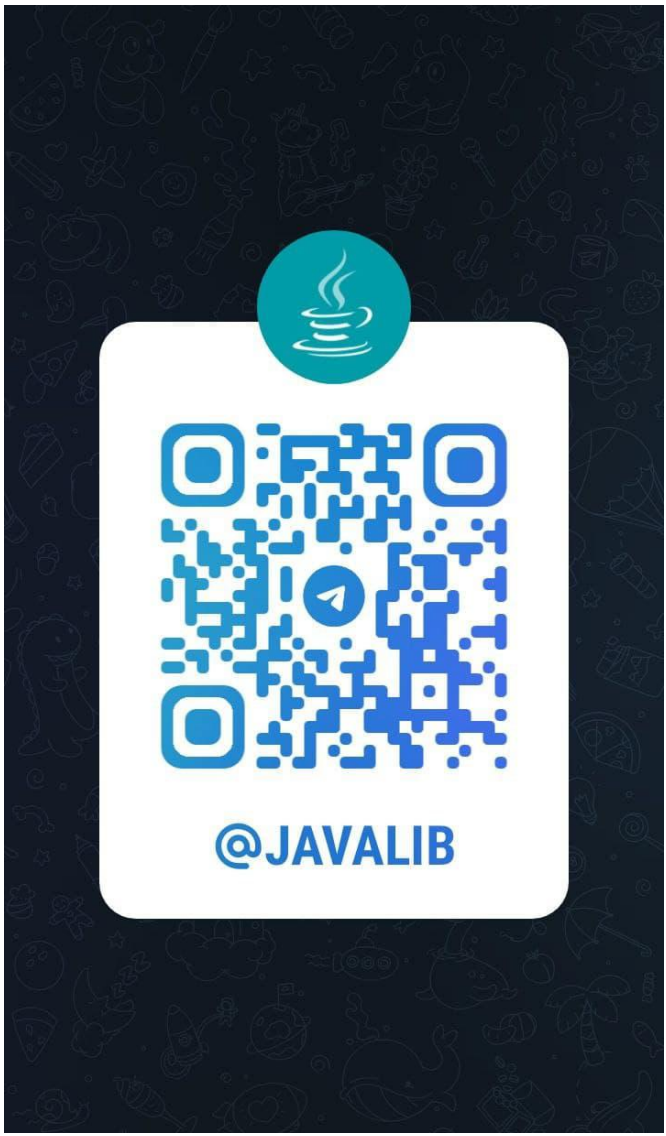
Краткое содержание главы

В этой главе была представлена концепция веток как различных направлений разработки и показано, как на практике они выступают в роли подвижных указателей на коммиты. Мы выяснили, почему нужны разные ветки, и вы узнали, как составлять список веток, создавать и изменять ветки. Мы установили, почему первая ветка в репозитории Rainbow называется `main`, и выяснили, почему в других учебных ресурсах, посвященных Git, вы можете встретить ветку под названи-

ем `master`. Вы также узнали, что `HEAD` — это указатель на ветку, в которой вы сейчас находитесь.

Продолжая делать коммиты, вы наблюдали, как отслеживаемые файлы меняют состояние с неизмененного на измененное при редактировании. Вы также заметили, что именно ветка, в которой вы находитесь, перемещается, указывая на последний сделанный вами коммит в локальном репозитории, и вы увидели, как коммиты связаны друг с другом через родительские ссылки.

Теперь, когда вы научились использовать ветки для одновременной работы над разными направлениями проекта, мы перейдем к *главе 5*, чтобы научиться объединять эти разные направления путем слияния.



[Глава 5]

Слияние

В предыдущей главе вы узнали о ветках, и мы обсудили, как они позволяют работать в разных направлениях над одним и тем же проектом и сотрудничать с другими людьми.

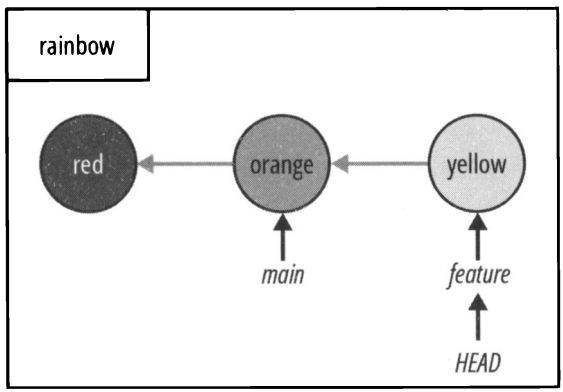
В этой главе вы узнаете об интеграции изменений из одной ветки в другую. В Git есть два способа сделать это: слияние и перебазирование. Мы рассмотрим перебазирование в *главе 11*; сейчас же мы сосредоточимся на слиянии. В этой главе будут представлены два типа слияний (ускоренное слияние и трехстороннее слияние), и вы выполните ускоренное слияние.

В этой главе вы также узнаете о том, как Git защищает вас от потери изменений, не включенных в коммит, как модификация веток может поменять содержимое рабочего каталога и как напрямую просматривать коммиты.

Состояние локального репозитория

В начале этой главы у вас должно быть три коммита и две ветки в репозитории `rainbow`, и вы должны находиться на ветке `feature`. Текущее состояние репозитория `rainbow` показано на диаграмме 5.1.

ДИАГРАММА 5.1



Репозиторий `rainbow` в начале *главы 5* с тремя коммитами и двумя ветками.

Что такое слияние?

В *главе 4* вы создали свою первую ветку `feature` и начали работать над ней. Ветки — мощная функция Git, и здорово, что они позволяют нам работать над разными частями проекта независимо. Но как потом объединить результат с веткой `main`, когда работа будет завершена?

Слияние (merging) в Git — это один из способов интегрировать в ветку изменения, внесенные в другой ветке. При любом слиянии есть одна ветка, которую вы объединяете, называемая *исходной веткой* (source branch), и одна ветка, в которую вы объединяете, называемая *целевой веткой* (target branch). Исходная ветка содержит изменения, которые будут интегрированы в целевую ветку. Целевая ветка получает изменения, и, следовательно, только она изменяется в операции слияния. Давайте посмотрим, как использовать слияние, на примере 5.1 проекта Book.

Пример 5.1 проекта Book

Предположим, в моем проекте Book мы с редактором договорились, что каждый раз, когда я начинаю работать над новой главой, я буду создавать для нее ответвление от основной ветки. Я также согласна со своим редактором, что я могу объединять второстепенную ветку с основной только после того, как он проверит мою работу на этой ветке.

Допустим, я решила поработать над главой 4. Я могу отделить ветку с названием `chapter_four` от основной ветки, поработать на этой ветке, а затем, когда я сочту, что добилась значительного прогресса, отправить свою работу на проверку редактору. Как только он одобрит изменения, я смогу объединить ветку `chapter_four` с веткой `main` моего локального репозитория.

Пример 5.1 проекта Book говорит о том, что при использовании веток также важно научиться объединять результаты своей работы. Далее мы рассмотрим различные типы слияний.

Типы слияний

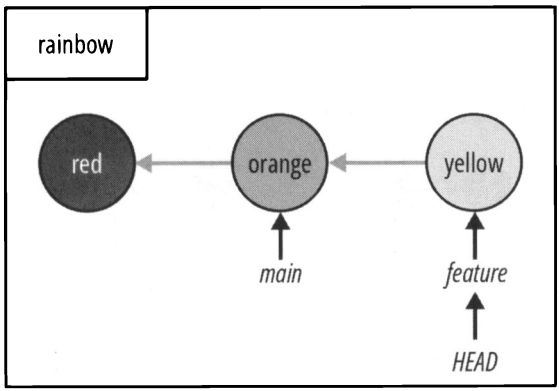
Существует два типа слияний:

- ускоренные слияния;
- трехсторонние слияния.

Фактором, от которого зависит, по какому типу произойдет слияние исходной ветки с целевой, является наличие или отсутствие факта, разошлись ли истории развития двух веток. Историю развития ветки можно проследить, пройдя по родительским ссылкам коммитов.

В *главе 4* было сказано, что на диаграммах репозитория серая стрелка, указывающая от одного коммита к другому, указывает на родительскую ссылку (т. е. стрелка указывает назад от дочернего коммита к родительскому). На диаграмме 5.2 вы можете видеть, что в репозитории `rainbow` родителем оранжевого коммита является красный коммит. По той же логике оранжевый коммит является родителем желтого коммита.

ДИАГРАММА 5.2



Родительские ссылки в репозитории `rainbow`.

История развития ветки начинается с коммита, на который она указывает, и продолжается назад по цепочке коммитов. На диаграмме 5.2 вы можете видеть, что история разработки ветки `main` состоит из оранжевого и красного коммитов, а история разработки ветки `feature` — из желтого, оранжевого и красного коммитов.

Для того чтобы проиллюстрировать разницу между ускоренным и трехсторонним слиянием, давайте начнем с рассмотрения примера ускоренного слияния в примере 5.2 проекта Book.

В этом примере мы видим, что *ускоренное слияние* — это тип слияния, который происходит, когда истории развития веток, участвующих в слиянии, не разошлись; другими словами, когда можно достичь целевой ветки, перейдя по родительским ссылкам, которые составляют историю коммитов исходной ветки. Во время ускоренного слияния Git берет указатель целевой ветки и перемещает его в коммит исходной ветки.

Пример 5.2 проекта Book

Предположим, что в ветке `main` репозитория моего проекта Book есть два коммита: A и B (рис. 5.1).

Далее предположим, что я создаю ветку `chapter_six` для работы над главой 6 моей книги и добавляю коммиты C, D и E в ветку `chapter_six` (рис. 5.2).

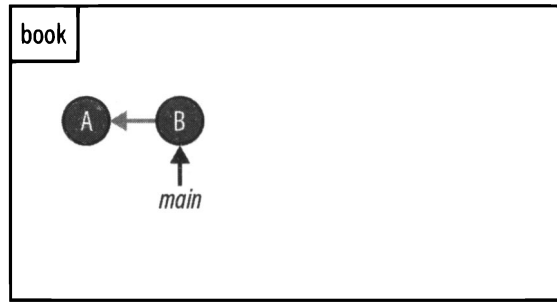


Рис. 5.1. История коммитов репозитория `book` с одной веткой `main`

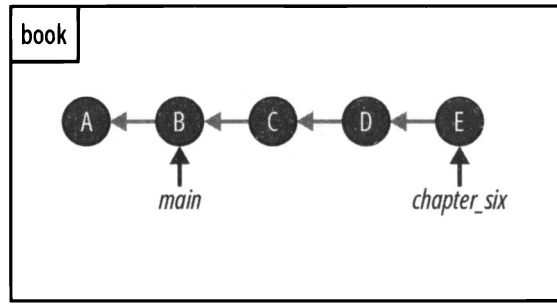


Рис. 5.2. История коммитов репозитория `book` после работы над веткой `chapter_six`

Если я пройду по родительским ссылкам ветки `main` назад, я увижу, что ветка состоит из коммитов A и B. Другими словами, история развития ветки `main` состоит из коммитов A и B. С другой стороны, история разработки ветки `chapter_six` состоит из коммитов A, B, C, D и E.

Если мы можем добраться до одной ветки через историю коммитов другой ветки, мы говорим, что истории развития ветвей *не разошлись*. Если я пройду по родительским ссылкам из ветки `chapter_six`, которая указывает на коммит E, назад, я доберусь до ветки `main`, которая указывает на коммит B. Следовательно, ветки `main` и `chapter_six` не разошлись.

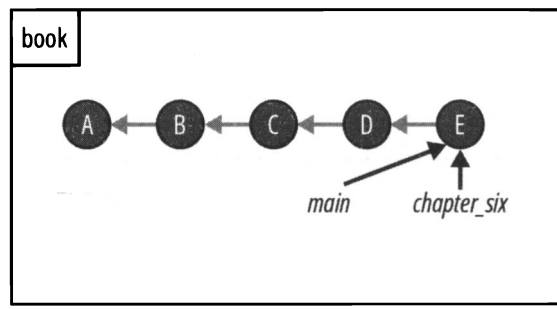


Рис. 5.3. История коммитов после объединения ветки `chapter_six` с веткой `main` репозитория `book`

Если бы я теперь объединила ветку `chapter_six` с веткой `main`, произошло бы ускоренное слияние. Во время ускоренного слияния указатель ветки `main` переместится вперед и будет указывать на коммит, на который указывает ветка `chapter_six`, т. е. коммит E (рис. 5.3).

В данном примере слияния `chapter_six` — это исходная ветка, а `main` — целевая. На рис. 5.3 показано, что указатель ветки `main` просто переместился вперед от коммита B к коммиту E. Вот почему такие виды слияний образно называют ускоренной перемоткой вперед (fast-forward merge).

Далее мы рассмотрим *трехстороннее слияние* на примере 5.3 проекта Book.

Этот пример говорит нам о том, что трехстороннее слияние происходит, когда истории развития веток, участвующих в слиянии, расходятся. Истории разработки расходятся, когда невозможно достичь целевой ветки, следуя истории коммитов исходной ветки. В этом случае, когда вы объединяете исходную ветку с целевой, Git выполняет трехстороннее слияние, создавая коммит слияния, чтобы связать две истории разработки вместе; затем он перемещает указатель целевой ветки на коммит слияния.

Пример 5.3 проекта Book

Предположим, что последние два коммита в ветке `main` моего репозитория `book` — это коммиты F и G (рис. 5.4).

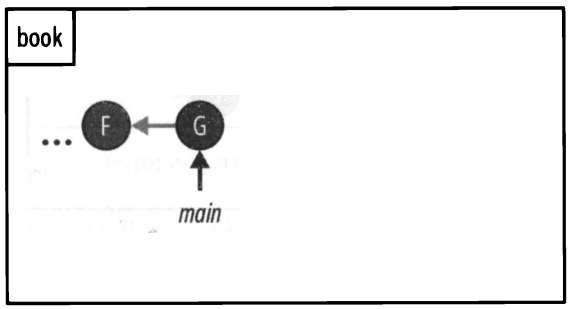


Рис. 5.4. История коммитов репозитория `book` с двумя последними коммитами в ветке `main`

Теперь предположим, что я решила создать ветку `chapter_eight` для работы над главой 8 моей книги и делаю коммиты H, I и J. Однако в то же время я вношу некоторые изменения в ветку `main`, и теперь она указывает на коммит L. Это показано на рис. 5.5.

На рис. 5.5 вы можете видеть, что история разработки ветки `chapter_eight` состоит из коммитов F, G, H, I и J. С другой стороны, история ветки `main` состоит из коммитов F, G, K и L. Невозможно пройти по родительским ссылкам (представленным серыми стрелками) ветки `chapter_eight` назад, чтобы достичь коммита, на который

указывает ветка `main`, т. е. коммита L. В этой ситуации мы говорим, что истории развития веток *разошлись*.

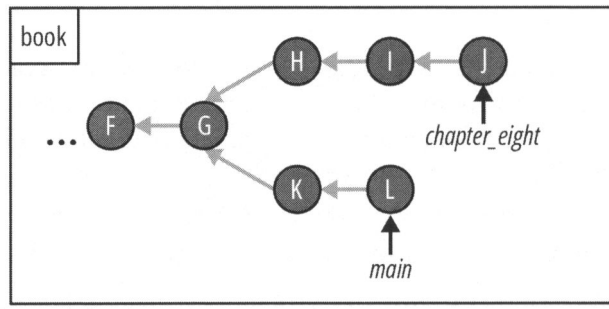


Рис. 5.5. Пример истории коммитов, в которой изменения были внесены в ветки `main` и `chapter_eight`

Если я захочу объединить ветку `chapter_eight` с веткой `main`, это не может быть быстрым слиянием, потому что невозможно просто переместить указатель ветки вперед, чтобы объединить эти две истории разработки. Вместо этого будет создан *коммит слияния* (назовем его коммитом M), чтобы связать две истории разработки вместе (рис. 5.6). Коммит слияния — это коммит, имеющий более одного родителя. Он служит примером *трехстороннего слияния* (three-way merge).

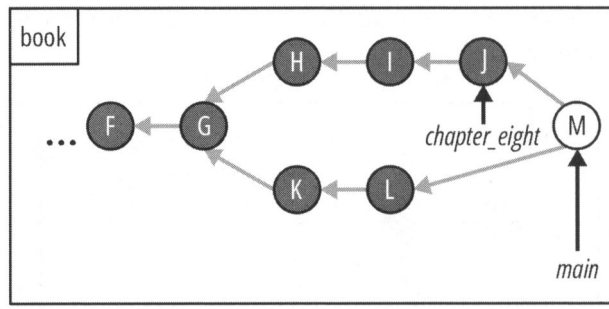


Рис. 5.6. История коммитов после объединения ветки `chapter_eight` с веткой `main`

На рис. 5.6 коммит M указывает на коммиты J и L. Причина, по которой этот вид слияния называется трехсторонним, заключается в том, что для выполнения слияния Git рассматривает два коммита, на которые указывают ветки, участвующие в слиянии (в случае репозитория `book` это коммиты J и L), а также коммит, который является общим предком этих двух коммитов, которым в данном случае является коммит G. Вот почему слияние "трехстороннее".

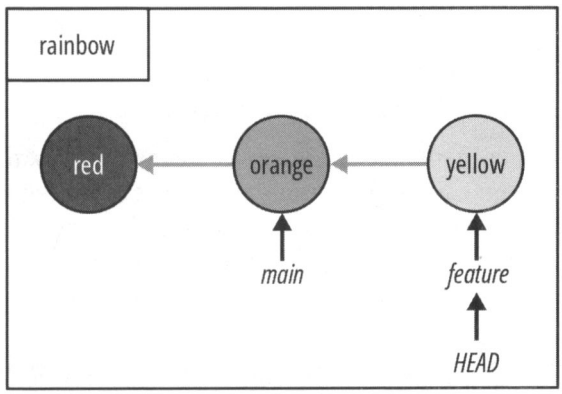
Трехсторонние слияния — это более сложный сценарий, при котором могут возникнуть *конфликты слияния*. Они возникают, когда вы объединяете две ветки, в которых в одни и те же части одного и того же файла (файлов) были внесены разные изменения, или если в одной ветке был удален файл, который редактировался в другой ветке.

Глава 9 этой книги посвящена более подробному объяснению трехсторонних слияний, а в *главе 10* вы узнаете больше о конфликтах слияний. В каждой главе вы будете рассматривать практический пример. В этой главе вы попрактикуетесь в выполнении ускоренного слияния в проекте Rainbow.

Выполнение ускоренного слияния

Для отработки навыка ускоренного слиянии в репозитории `rainbow` вы объедините ветку `feature` с веткой `main`. В данном случае ветка `feature` — исходная, а ветка `main` — целевая. На диаграмме 5.3 видно, что вы можете добраться до ветки `main`, следуя истории коммитов ветки `feature`; следовательно, это будет ускоренное слияние.

ДИАГРАММА 5.3



В репозитории `rainbow` истории развития веток `main` и `feature` не разошлись.

Процесс слияния состоит из двух этапов:

1. Переключитесь на ветку, в которую вы хотите выполнить слияние (целевую ветку).
2. Введите команду `git merge` и передайте ей имя объединяемой ветки (исходная ветка).

ЗАПОМНИТЕ КОМАНДУ

`git merge <имя_ветки>`

Интегрирует изменения из одной ветки в другую.

Далее вы узнаете, как выполнить первое слияние. В процессе вы усвоите еще два важных урока о Git: во-первых, Git защищает вас от потери всей работы, проделанной в файлах, которые вы не добавили в коммиты, и, во-вторых, изменение веток может привести к изменению содержимого рабочего каталога.

Переключение на целевую ветку

Первым шагом при слиянии является переключение на целевую ветку. Вы собираетесь объединить `feature` с `main`, поэтому вам нужно переключиться на ветку `main`. Вспомните три вещи, которые делает команда `git switch` (или `git checkout`) для переключения на ветку:

1. Изменяет указатель `HEAD`, чтобы он указывал на ветку, на которую вы переключаетесь.
2. Заполняет промежуточную область всеми файлами и каталогами, входящими в коммит, на который вы переключаетесь.
3. Копирует содержимое промежуточной области в рабочий каталог.

Как показывают эти три шага, если ветки указывают на разные коммиты, изменение веток также меняет содержимое вашего рабочего каталога. Вскоре вы увидите это на практике, но сначала обсудим другую важную особенность, о которой я недавно упоминала.

Git защищает вас от потери незафиксированных изменений

Как вы только что узнали, переключение веток меняет содержимое вашего рабочего каталога. Но что, если в вашем рабочем каталоге есть модифицированные файлы (другими словами, файлы, которые вы редактировали), которые вы еще не включили в коммит? Потеряете ли вы всю работу, проделанную в этих файлах, если переключите ветку? К счастью, нет! Git защищает вас от потери незафиксированных изменений.

Если Git обнаружит, что переключение веток приведет к потере изменений в вашем рабочем каталоге, он не позволит вам переключить ветки и выдаст сообщение об ошибке. Однако это происходит только в том случае, если модифицированные файлы, не вошедшие ни в один коммит, содержат изменения, конфликтующие с веткой, на которую вы переключаетесь.

Для того чтобы проиллюстрировать, почему это важно, давайте рассмотрим пример 5.4 проекта Book.

Пример 5.4 проекта Book

Предположим, я хочу поработать над двумя разными подходами к главе 5 в моем репозитории `book`. Это означает, что я буду работать над файлом chapter_five.txt. Я отделяю от ветки `main` две дочерние ветки: `chapter_five_approach_a` и `chapter_five_approach_b`.

Сначала я перехожу в ветку `chapter_five_approach_a` и работаю над вариантом A. Делаю пару коммитов. В этих коммитах я редактирую файл chapter_five.txt.

Затем я решаю переключиться на ветку `chapter_five_approach_b`, чтобы поработать над вариантом B в моем текстовом редакторе. Находясь в ветке `chapter_five_approach_b`, я тщательно редактирую файл chapter_five.txt, но забываю сделать какие-либо коммиты в ветке (другими словами, забываю правильно сохранить свою работу).

В какой-то момент я решаю вернуться к ветке `chapter_five_approach_a`, чтобы вспомнить, что я сделала в этой ветке. Если бы Git просто позволил мне переключать ветки, то версия файла chapter_five.txt в моем рабочем каталоге, над которым я работала в ветке `chapter_five_approach_b`, была бы заменена версией файла из ветки `chapter_five_approach_a`, и я потеряла бы все изменения, над которыми работала в варианте B, потому что забыла сделать для них коммит.

К счастью, Git не позволит этому случиться. Вместо того чтобы покорно заменить файлы, он предупредит меня о том, что в рабочем каталоге есть изменения, и напомнит о необходимости сделать коммит, чтобы я не потеряла их.

Для того чтобы рассмотреть практический пример редактирования файла в проекте Rainbow и увидеть, как Git защищает вас от потери незафиксированных изменений, выполните действия 5.1.

ДЕЙСТВИЯ 5.1

1.
```
rainbow $ git status
On branch feature
nothing to commit, working tree clean
```

2. В папке проекта rainbow с помощью текстового редактора в файле rainbowcolors.txt добавьте текст "Зеленый — четвертый цвет радуги." в строке 4. Затем сохраните файл.

3.
```
rainbow $ git status
On branch feature
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
directory)
    modified: rainbowcolors.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

(...
изменения не добавлены в коммит (используйте "git add" и/или "git commit -a"))

4
```
rainbow $ git switch main
error: Your local changes to the following files would be
overwritten by checkout:
    rainbowcolors.txt
Please commit your changes or stash them before you switch
branches.
Aborting
```

(ошибка: ваши локальные изменения в следующих файлах будут переписаны при выходе: rainbowcolors.txt

Пожалуйста, внесите ваши изменения в коммит или сохраните их перед переключением веток.

Прерывание выполнения)

На что обратить внимание:

- на шаге 1 вывод `git status` показывает, что в рабочем каталоге нет измененных файлов;
- на шаге 3 выходные данные `git status` указывают, что файл rainbowcolors.txt является измененным файлом в рабочем каталоге;
- на шаге 4 вы видите, что Git не позволяет вам переключить ветки. Он предупреждает вас, что ваши локальные изменения в следующих файлах будут перезаписаны при смене ветки, и предлагает вам сделать коммит изменений или сохранить их в другом месте, прежде чем переключать ветки.

Если бы Git позволил вам переключить ветки, то версия файла rainbowcolors.txt в рабочем каталоге, в которой упоминаются красный, оранжевый, желтый и зеленый цвета, была бы заменена версией файла rainbowcolors.txt из оранжевого коммита, на который указывает ветка `main`, и в котором упоминаются только красный и оранжевый цвета. Это означает, что вы потеряете *всю* работу, которую вы проделали, добавив зеленый цвет в свой список цветов.

Вместо этого Git защищает вас от потери вашей работы в файле rainbowcolors.txt, предупреждая о необходимости сделать коммит изменений.

ПРИМЕЧАНИЕ

Git не помешает вам переключать ветки, если вы вносите изменения в файлы, не сохраняя их в текстовом редакторе, поскольку эти файлы будут считаться неизмененными. Поэтому не забывайте сохранять файлы в текстовом редакторе, если закончили работу над ними!

Теперь вы продолжите выполнять первое слияние в проекте Rainbow. В этом случае предположим, что вы еще не готовы добавлять записи о зеленом цвете. Выполните действия 5.2 и удалите предложение об этом цвете, чтобы в вашем рабочем каталоге не было измененных файлов.

ДЕЙСТВИЯ 5.2

1. В каталоге проекта `rainbow` в текстовом редакторе удалите строку "Зеленый — четвертый цвет радуги." из файла rainbowcolors.txt и сохраните его. Обязательно удалите все лишние строки и пробелы, которые вы могли добавить.
2.
```
rainbow $ git status
On branch feature
nothing to commit, working tree clean
```

На что обратить внимание:

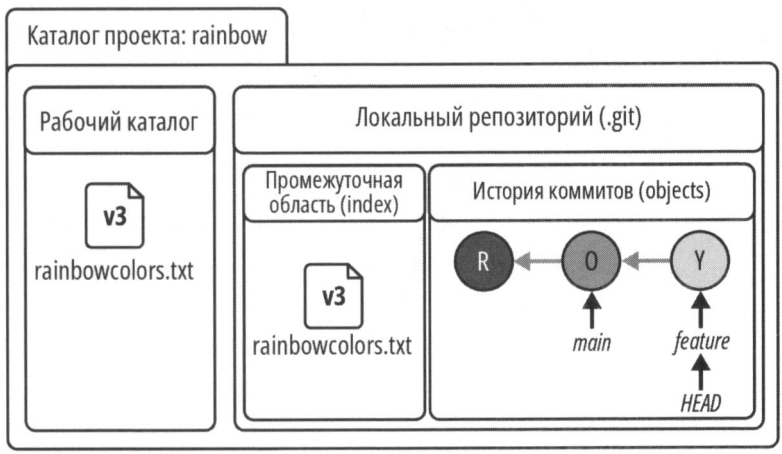
- команда `git status` сообщает, что в вашем рабочем каталоге больше нет измененных файлов.

Далее вам предстоит выполнить первый шаг слияния, т. е. переключиться на ветку, в которую вы будете выполнять слияние. Пока вы это делаете, вы станете свидетелем того, как переключение ветвей меняет файлы в вашем рабочем каталоге.

Переключение веток меняет файлы в рабочем каталоге

Для того чтобы увидеть, как переключение ветвей меняет файлы в рабочем каталоге, взгляните на диаграмму 5.4, в которой используется схема Git, представленная в *главе 2*, чтобы показать, как в данный момент выглядят различные области каталога проекта `rainbow`.

ДИАГРАММА 5.4



Каталог проекта `rainbow` перед переключением с ветки `feature` на `main`.

ПРИМЕЧАНИЕ

На диаграммах, где недостаточно места для включения полного имени коммита, я буду использовать сокращенную форму имени: например, R для красного (red), O для оранжевого (orange) и т. д. Полный список сокращений, используемых для каждого коммита, приведен в табл. П1 в *предисловии*.

На что обратить внимание:

- версия файла rainbowcolors.txt в рабочем каталоге и промежуточной области — это версия, в которой упоминаются красный, оранжевый и желтый цвета. Она представлен как версия 3 (v3) файла.

Теперь выполните действия 5.3, чтобы переключиться с ветки `feature` на ветку `main` репозитория `rainbow` (которая указывает на другой коммит), и посмотрите, как изменится содержимое вашего рабочего каталога.

ДЕЙСТВИЯ 5.3

1. Откройте файл rainbowcolors.txt в окне текстового редактора и поместите его рядом с окном командной строки, чтобы вы могли видеть их оба при выполнении предстоящих команд. Посмотрите на содержимое файла rainbowcolors.txt.
2.
```
rainbow $ git switch main
Switched to branch 'main'
```

(Переключен на ветку main)

3. Посмотрите на содержимое файла rainbowcolors.txt.
4.
```
rainbow $ git log
commit 7acb333f08e12020efb5c6b563b285040c9dba93 (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
 Date:  Sat Feb 19 09:42:07 2022 +0100

     orange

commit c26d0bc371c3634ab49543686b3c8f10e9da63c5
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

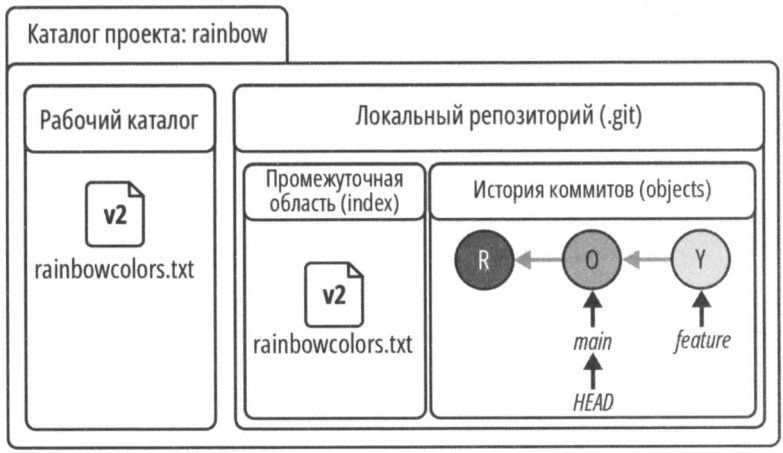
На что обратить внимание:

- в версии файла rainbowcolors.txt, который был открыт в вашем текстовом редакторе до того, как вы переключились на ветку `main`, упоминались красный, оран-

жевый и желтый цвета. Версия файла rainbowcolors.txt, которая находится в вашем текстовом редакторе после переключения на основную ветку, упоминает только красный и оранжевый цвета. Этот факт отражен на диаграмме 5.5;

- на шаге 4 в выходных данных команды `git log` видны только красный и оранжевый коммиты. Там больше не показан желтый коммит.

ДИАГРАММА 5.5



Каталог проекта `rainbow` после переключения с ветки `feature` на `main`.

На что обратить внимание:

- вы находитесь на ветке `main`, и она указывает на оранжевый коммит, который включает версию файла rainbowcolors.txt с упоминанием красного и оранжевого цветов. Этот файл представлен как версия 2 (v2);
- версия 3 файла rainbowcolors.txt заменена версией 2 файла как в промежуточной области, так и в рабочем каталоге.

Вы только что наблюдали, как переключение ветвей меняет содержимое рабочего каталога. Прежде чем перейти ко второму шагу слияния, давайте кратко коснемся того, почему в выводе `git log` на шаге 4 действий 5.3 отображаются только красный и оранжевый коммиты.

Просмотр списка всех коммитов

В *разд. "Просмотр списка коммитов" главы 3* я упоминала, что команда `git log` показывает список коммитов в обратном хронологическом порядке. Однако на самом деле она демонстрирует только список коммитов, доступных по родительским ссылкам из коммита, в котором вы находитесь при выполнении команды. Для того чтобы просмотреть список коммитов *всех* ветвей вашего локального репозитория, вы должны ввести команду `git log` с опцией `--all`.

ЗАПОМНИТЕ КОМАНДУ

git log --all

Показывает список коммитов в обратном хронологическом порядке для всех веток локального репозитория.

Выполните действия 5.4, чтобы увидеть список коммитов для всех ветвей вашего локального репозитория.

ДЕЙСТВИЯ 5.4

1
```
rainbow $ git log --all
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (feature)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow

commit 7acb333f08e12020efb5c6b563b285040c9dba93 (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange

commit c26d0bc371c3634ab49543686b3c8f10e9da63c5
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

На что обратить внимание:

- в выводе `git log --all` показаны красный, оранжевый и желтый коммиты.

Теперь, когда вы находитесь в ветке `main`, можно перейти ко второму этапу слияния.

Использование команды *git merge* для слияния

Выполните действия 5.5, чтобы завершить слияние, передав имя объединяемой ветки (исходной ветки `feature`) команде `git merge`.

ДЕЙСТВИЯ 5.5

1 Откройте файл rainbowcolors.txt в окне текстового редактора рядом с окном командной строки, чтобы вы могли видеть их оба при выполнении предстоящих команд. Посмотрите содержимое файла rainbowcolors.txt.

2 rainbow $ **git merge feature**

```
Updating 7acb333..fc8139c
Fast-forward
 rainbowcolors.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

(Обновление 7acb333..fc8139c
Ускоренное слияние
1 файл изменен, 2 вставки, 1 удаление)

3 Посмотрите содержимое файла rainbowcolors.txt.

4 rainbow $ **git log**

```
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (HEAD -> main,
feature)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow

commit 7acb333f08e12020efb5c6b563b285040c9dba93
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange

commit c26d0bc371c3634ab49543686b3c8f10e9da63c5
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

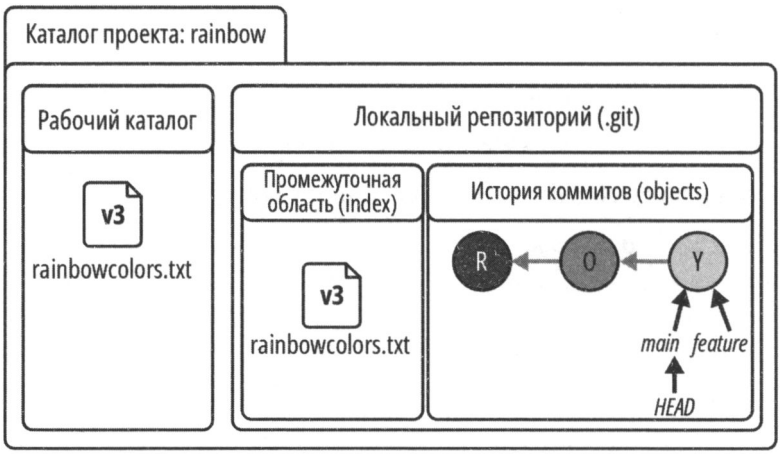
На что обратить внимание:

- в выводе `git merge` упоминаются обновление `7acb333..fc8139c` и `Fast-forward`. Это говорит о том, что Git обновил коммит, на который указывает ветка `main`, а также о том, что это было ускоренное слияние. Имейте в виду, что хеши коммитов в вашем выводе команды будут отличаться от тех, что приведены в этой книге;
- вывод `git log` говорит о том, что `main` указывает на желтый коммит. В текстовом редакторе вы можете увидеть, что ваш рабочий каталог содержит версию файла rainbowcolors.txt, который является частью желтого коммита;

- вы объединили ветку `feature` с веткой `main`, но она все еще существует, поскольку не была удалена автоматически.

Все эти наблюдения проиллюстрированы на диаграмме 5.6.

ДИАГРАММА 5.6



Рабочий каталог и локальный репозиторий после слияния ветки `feature` с `main`.

Вы объединили ветку `feature` с `main`, но в Git процедура слияния ветки не удаляет исходную ветку. Вы должны явно удалить ветку, если больше не хотите ее использовать. На данный момент вы оставите ветку `feature`, а в *главе 8* научитесь удалять ветки. Следующая тема, которую я хочу здесь затронуть, — это проверка коммитов.

Проверка коммитов

В *главе 4* я упоминала, что команду `git checkout` можно использовать для переключения ветвей, а также для выполнения других действий. Еще одна операция, которую вы можете сделать с помощью команды `git checkout`, — это проверить коммиты.

На данный момент вы находитесь на ветке `main`, которая указывает на желтый коммит. Но что если вы хотите просмотреть более старую версию своего проекта? Например, что если вы хотите увидеть состояние вашего проекта на стадии оранжевого коммита?

На данный момент нет ветки, указывающей на оранжевый коммит, поэтому вы не можете просмотреть его, переключившись на другую ветку. Вместо этого для просмотра коммита введите команду `git checkout` и передайте ей хеш оранжевого коммита.

ЗАПОМНИТЕ КОМАНДУ

`git checkout <хеш_коммита>`

Просмотр коммита.

Когда вы это сделаете, команда `git checkout` выполнит три действия, аналогичные описанным в *главе 4* и ранее в этой главе:

1. Изменит указатель `HEAD`, чтобы он указывал на коммит, на который вы переключаетесь.
2. Заполнит промежуточную область всеми файлами и каталогами, являющимися частью коммита, на который вы переключаетесь.
3. Скопирует содержимое промежуточной области в рабочий каталог.

Основное различие между этими шагами и шагами, упомянутыми ранее, заключается в том, что на шаге 1 указатель `HEAD` будет указывать непосредственно на коммит, а не на ветку. Это означает, что вы будете находиться в состоянии, которое Git пугающе называет *состоянием отсоединенного HEAD* (detached HEAD state). Оно позволяет вам просмотреть любой коммит — или, другими словами, любую версию вашего проекта — во всем вашем репозитории.

Как показывают эти шаги, просмотр коммитов изменяет содержимое рабочего каталога так же, как и переключение ветвей.

ПРИМЕЧАНИЕ

Не рекомендуется вносить какие-либо изменения в репозиторий, пока он находится в состоянии отсоединенного `HEAD` (т. е. пока он не указывает на ветку). Обычно вы будете делать коммиты в ветках, потому что ветки легче запомнить и на них легче ссылаться, чем при использовании хешей коммитов, а также потому, что Git был разработан для операций с ветками. Поэтому для просмотра коммита обычно создают новую ветку, которая будет указывать на этот коммит, и переключаются на нее, благодаря чему удается избежать отсоединенного состояния `HEAD`.

Выполните действия 5.6, чтобы просмотреть оранжевый коммит и увидеть, каково это — находиться в состоянии отсоединенного `HEAD`.

ДЕЙСТВИЯ 5.6

1. Получите хеш коммита для вашего оранжевого коммита (вы можете скопировать его из вывода `git log`). Обязательно используйте хеш коммита на шаге 2 этой инструкции.
2.
```
rainbow $ git checkout 7acb333f08e12020efb5c6b563b285040c9dba93
Note: switching to '7acb333f08e12020efb5c6b563b285040c9dba93'.

You are in 'detached HEAD' state. You can look around, make ex-
perimental changes and commit them, and you can discard any com-
mits you make in this state without impacting any branches by
switching back to a branch.
```

(Вы находитесь в состоянии отсоединенного HEAD. Вы можете осмотреться, внести экспериментальные изменения и зафиксировать их, а также отменить любые коммиты, сделанные в этом состоянии, не затрагивая ни одну ветку, переключившись обратно на ветку.)

```
If you want to create a new branch to retain commits you create,
you may do so (now or later) by using -c with the switch command.
Example:

  git switch -c <new-branch-name>
```

(Если вы хотите создать новую ветку для сохранения созданных вами коммитов, вы можете сделать это (сейчас или позже), используя -c с командой switch. Пример:

git switch -c <имя_новой_ветки>)

```
Or undo this operation with:

  git switch -
```

(Или отменить эту операцию с помощью:

...)

```
Turn off this advice by setting config variable
advice.detachedHead to false

HEAD is now at 7acb333 orange
```

(Для отключения этого совета задайте переменной конфигурации advice.detachedHead значение false)

3

```
rainbow $ git log --all
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (main, feature)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow

commit 7acb333f08e12020efb5c6b563b285040c9dba93 (HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange
```

```
commit c26d0bc371c3634ab49543686b3c8f10e9da63c5
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

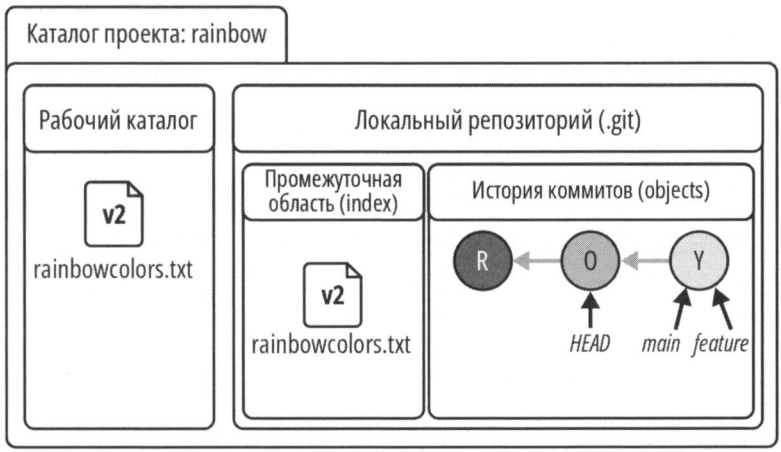
4 Просмотрите содержимое файла rainbowcolors.txt.

На что обратить внимание:

- на шаге 2 выходные данные `git checkout` сообщают вам, что вы находитесь в состоянии отсоединенного `HEAD`. Вывод также относится к модификации команды `git switch`, которую мы кратко рассмотрим в следующем разделе;
- на шаге 3 выходные данные `git log` сообщают, что `HEAD` теперь указывает на оранжевый коммит;
- версия файла rainbowcolors.txt в вашем рабочем каталоге — это версия, которая является частью оранжевого коммита, где упоминаются только красный и оранжевый цвета (представленные версией v2).

Эти наблюдения проиллюстрированы на диаграмме 5.7.

ДИАГРАММА 5.7



Рабочий каталог и локальный репозиторий после просмотра оранжевого коммита и перехода в состояние отсоединенного `HEAD`.

ПРИМЕЧАНИЕ

Может показаться пугающим, что содержимое рабочего каталога меняется каждый раз, когда вы переключаете ветки или напрямую извлекаете коммит, но помните, что вы ничего не потеряли. Все ваши коммиты надежно сохраняются в истории коммитов, и вы всегда можете переключиться на другую ветку или проверить другой коммит, чтобы увидеть, что вы просматривали раньше.

Вы только что увидели, что значит обращаться к коммиту напрямую, а не извлекать ветку. Теперь выполните действия 5.7, чтобы переключиться обратно на ветку `main` и выйти из состояния отсоединенного `HEAD`.

ДЕЙСТВИЯ 5.7

1.
```
rainbow $ git switch main
Previous HEAD position was 7acb333 orange
Switched to branch 'main'
```

(Предыдущая позиция HEAD была 7acb333 orange
Переключена на ветку main)

2.
```
rainbow $ git log
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (HEAD -> main,
feature)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow

commit 7acb333f08e12020efb5c6b563b285040c9dba93
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange

commit c26d0bc371c3634ab49543686b3c8f10e9da63c5
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

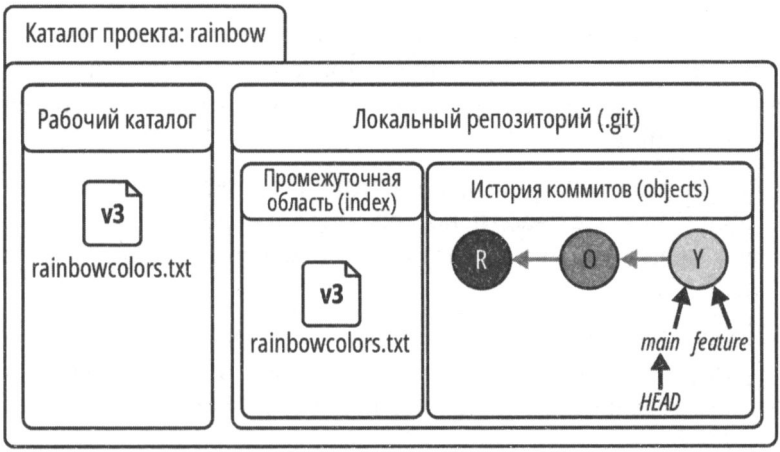
3. Посмотрите на содержимое файла rainbowcolors.txt.

На что обратить внимание:

- версия файла rainbowcolors.txt в вашем рабочем каталоге — это версия, которая является частью желтого коммита, содержащего упоминания красного, оранжевого и желтого цветов (представленные версией v3);
- вы вернулись на ветку `main`.

Текущее состояние показано на диаграмме 5.8.

ДИАГРАММА 5.8



Рабочий каталог и локальный репозиторий после переключения на ветку `main` и выхода из состояния отсоединенного `HEAD`.

Создание ветки с переключением на нее

В *главе 4* вы научились создавать новую ветку с помощью команды `git branch` и переходить на эту ветку с помощью команды `git switch` (или `git checkout`).

На шаге 2 действий 5.6 в выводе `git checkout` говорилось: "Если вы хотите создать новую ветку для сохранения подготовленных вами коммитов, вы можете сделать это (сейчас или позже), используя команду `switch` с опцией `-c`. Пример: `git switch -c <имя_новой_ветки>`".

Это связано с тем, что на самом деле можно использовать команду `git switch` или `git checkout`, чтобы создать ветку и переключиться на нее за один раз. Если вы предпочитаете команду `git switch`, то дополните ее опцией `-c` (от слова create — создать), а если вы ввели команду `git checkout`, необходимо добавить опцию `-b`.

ЗАПОМНИТЕ КОМАНДУ

`git switch -c <имя_новой_ветки>`

Создает новую ветку и переключается на нее.

`git checkout -b <имя_новой_ветки>`

Создает новую ветку и переключается на нее.

Вы попрактикуетесь в использовании этой команды в *разд. "Подготовка к выполнению запроса на включение" главы 12*, когда будете создавать новую ветку.

Краткое содержание главы

В этой главе вы узнали, что такое слияние и почему мы его делаем. Я представила два типа слияний — ускоренное слияние и трехстороннее слияние — и объяснила, как тип слияния, которое будет выполнено, зависит от истории развития ветвей, участвующих в слиянии.

Наконец, вы сами выполнили ускоренное слияние в репозитории `rainbow`, узнав в процессе, что Git защищает вас от потери незафиксированных изменений, что изменение ветвей может изменить содержимое рабочего каталога, и научились просматривать коммиты.

Эта глава завершает первую часть книги, в которой вы работали исключительно с локальными репозиториями. Далее, в *главе 6*, вы подготовите свою учетную запись службы хостинга и данные аутентификации, чтобы начать работу с удаленными репозиториями.

[Глава 6]

Хостинговые сервисы и аутентификация

В предыдущей главе вы узнали о слиянии и о том, как эта функция Git позволяет интегрировать изменения из одной ветки в другую.

До этого момента мы обсуждали работу лишь с локальными репозиториями, а вы работали только с репозиторием `rainbow`, который является локальным. Эта глава знаменует собой начало второй части книги, в которой вы будете работать со службами хостинга и удаленными репозиториями.

В этой главе вы выберете службу хостинга и подготовите данные аутентификации, которые вы будете использовать для подключения к удаленным репозиториям на этой службе хостинга по протоколу HTTPS (hypertext transfer protocol secure — безопасный протокол передачи гипертекста) или SSH (secure shell — безопасная оболочка). Информация, которая вам понадобится для выполнения этих задач, содержится в этой главе, а также в ресурсах репозитория Learning Git (**https://github.com/gitlearningjourney/learning-git**).

> **ПРИМЕЧАНИЕ**
>
> Если вы работаете в компании, которая использует Git и службу хостинга, возможно, у вас уже есть учетная запись хостинга, использующая адрес электронной почты вашей компании. Однако для выполнения упражнений из этой книги я рекомендую выбрать личную учетную запись на хостинге, а не учетную запись компании. Это связано с тем, что ваша компания могла настроить дополнительные параметры в учетной записи службы хостинга, что может усложнить упражнения, описанные далее в книге.

Если вы уже работаете со службой хостинга, которую хотите задействовать для выполнения упражнений из этой книги, и у вас уже настроены данные аутентификации для подключения к удаленным репозиториям по защищенному протоколу, то вы можете пропустить эту главу и перейти к *главе 7*, чтобы узнать больше о том, как создавать удаленные репозитории и отправлять в них обновления.

Если вы еще не остановили свой выбор на конкретном хостинге, не настроили данные аутентификации для определенного протокола или хотите узнать больше о доступных протоколах, продолжайте читать эту главу.

Услуги хостинга и удаленные репозитории

В *главе 2* говорилось о том, что существует два типа репозиториев: локальные и удаленные. Локальные репозитории находятся на вашем компьютере, а удаленные репозитории размещаются на хостинге в облаке.

Я также упомянула, что под хостинговыми услугами мы понимаем компании, которые предоставляют хостинг (размещение) проектам, использующим Git. В этой книге я расскажу о трех основных службах хостинга Git: GitHub, GitLab и Bitbucket.

Для того чтобы передать данные между локальным репозиторием и удаленным репозиторием на сервере хостинга, вам необходимо подключиться и пройти аутентификацию с помощью протокола SSH или HTTPS. Начиная с *главы 7*, вы будете передавать данные в удаленный репозиторий и получать их из него, выполняя такие команды, как `git push`, `git clone`, `git fetch` и `git pull`. Для того чтобы успешно выполнить эти команды и подключиться к удаленному репозиторию, вам необходимо заранее подготовить данные аутентификации для протокола, который вы предпочитаете.

В этой главе сначала вы выберете службу хостинга, а затем настроите данные аутентификации для подключения к удаленным репозиториям через HTTPS или SSH.

Настройка учетной записи службы хостинга

Как упоминалось ранее, если вы уже работаете с хостинговой службой и уверены, что будете использовать ее для упражнений в этой книге, вы можете просто войти в свою учетную запись на веб-сайте хостинговой службы.

Если вы никогда не работали с хостингом, вам придется выбрать какой-то один из них и создать учетную запись. Три основных хостинговых сервиса — это GitHub, GitLab и Bitbucket. Если вы не имеете конкретных предпочтений и не знаете, что выбрать, я рекомендую остановиться на GitHub, поскольку он является самым популярным и именно его я использую для вывода командной строки в примерах проекта Rainbow в этой книге.

Выполните действия 6.1, чтобы выбрать услугу хостинга, и войдите в систему.

ДЕЙСТВИЯ 6.1

1. Выберите хостинг, на котором вы хотите создать удаленный репозиторий, и войдите на сайт. Либо войдите в систему с имеющимися учетными данными, либо создайте учетную запись. Примеры этой книги основаны на работе с GitHub.

Теперь, когда вы выбрали службу хостинга и вошли в свою учетную запись, следующим шагом нужно выбрать протокол для подключения к удаленным репозиториям и подготовить данные аутентификации для этого протокола.

Настройка учетных данных аутентификации

Когда вы создаете удаленный репозиторий, внести в него изменения можно двумя способами:

1. Авторизовавшись на хостинге через его сайт и внося изменения непосредственно там.
2. Внеся изменения в локальный репозиторий и загрузив эти изменения в удаленный репозиторий на хостинге.

В обоих случаях вам необходимо будет пройти *аутентификацию* — подтвердить свою личность. Аутентификация определяет, кто имеет право входить в вашу учетную запись службы хостинга и загружать изменения в удаленный репозиторий.

В первом случае вы будете вводить свое имя пользователя или адрес электронной почты и пароль для аутентификации и входа в учетную запись на обычном вебсайте службы хостинга.

А как быть во втором случае? Как ваша служба хостинга узнает, что она должна позволять вам загружать файлы из локального репозитория в удаленный репозиторий?

Ответ заключается в том, что вам придется пройти аутентификацию по протоколу, который вы выберете. В этой главе мы рассмотрим протоколы HTTPS и SSH. Вам нужно настроить только один из них, и вы можете выбрать тот, который вам больше нравится.

> **ПРИМЕЧАНИЕ**
>
> В *главе 7* вы выполните процедуру создания удаленного репозитория для вашего проекта Rainbow, в ходе которой будут сформированы два URL-адреса: URL-адрес HTTPS и URL-адрес SSH. Вам нужно будет сохранить URL-адрес протокола, который вы решите использовать.

Сначала мы рассмотрим протокол HTTPS. Если вы хотите узнать только о протоколе SSH, можете пропустить следующий раздел и сразу перейти к *разд. "Работа с протоколом SSH"*. Надо сказать, вам вовсе не обязательно хорошо разбираться в протоколах, чтобы подключаться к удаленному репозиторию.

Если вы не решили, какой протокол выбрать, для выполнения упражнений из этой книги я рекомендую настроить протокол HTTPS, поскольку этот процесс немного проще, да и примеры в этой книге будут использовать протокол HTTPS. Какой бы протокол вы ни выбрали сейчас, вы всегда можете настроить другой в своей будущей работе с Git, так что это не критическое решение.

Работа с протоколом HTTPS

Протокол HTTPS использует имя пользователя и какой-либо пароль (или учетные данные аутентификации), чтобы организовать безопасное подключение к удален-

ным репозиториям. Раньше все службы хостинга позволяли вам вводить пароль, предназначенный для входа в свою учетную запись на сайте хостинга (который мы будем называть паролем учетной записи), также и для аутентификации HTTPS. Однако GitHub и Bitbucket больше не позволяют этого; теперь они требуют от вас создания отдельных учетных данных для аутентификации через HTTPS.

В GitHub учетные данные аутентификации называются *токеном личного доступа* (personal access token). В Bitbucket они называются *паролем приложения* (app password). С GitLab вы по-прежнему можете просто использовать пароль своей учетной записи для аутентификации. В табл. 6.1 представлен обзор трех наиболее распространенных служб хостинга и учетных данных, необходимых для аутентификации через HTTPS.

Таблица 6.1. *Услуги хостинга и необходимые учетные данные для аутентификации через HTTPS*

Служба хостинга	Имя пользователя	Пароль
GitHub	Адрес e-mail или имя пользователя	Личный токен доступа
GitLab	Адрес e-mail или имя пользователя	Пароль учетной записи
Bitbucket	Адрес e-mail или имя пользователя	Пароль приложения

Выполните действия 6.2, чтобы подготовить учетные данные для аутентификации.

ДЕЙСТВИЯ 6.2

1. Если вы предпочитаете GitHub или Bitbucket и выбрали протокол HTTPS, перейдите в репозиторий Learning Git (**https://github.com/gitlearningjourney/learning-git**) для получения дополнительной информации о настройке учетных данных аутентификации. Вы можете пропустить следующий раздел.

   Если вы выбрали протокол SSH, перейдите к *разд. "Работа с протоколом SSH"*.

Работа с протоколом SSH

Протокол SSH использует пару публичных и частных ключей SSH, что позволяет вам безопасно подключаться к удаленным репозиториям. Вот три основных шага для настройки доступа по SSH:

1. Создайте пару ключей SSH на своем компьютере.
2. Добавьте закрытый ключ SSH к агенту SSH.
3. Добавьте открытый ключ SSH в учетную запись службы хостинга.

Выполните действия 6.3, чтобы пройти три шага настройки доступа по SSH.

ДЕЙСТВИЯ 6.3

1. Если вы предпочитаете протокол SSH, перейдите в репозиторий Learning Git (**https://github.com/gitlearningjourney/learning-git**) для получения дополнительной информации о настройке учетных данных аутентификации по SSH.

ПРИМЕЧАНИЕ

Разглашение вашего закрытого ключа SSH представляет собой серьезную угрозу безопасности. Обращайтесь с ним как с паролем и никому не сообщайте.

Итак, вы подготовили данные аутентификации для подключения к удаленным репозиториям через HTTPS или SSH, и теперь можете перейти к следующей главе, чтобы создать первый удаленный репозиторий для проекта Rainbow.

Краткое содержание главы

В этой главе вы выбрали службу хостинга Git для выполнения остальных упражнений в этой книге, и подготовили данные аутентификации, которые вам понадобятся для подключения к удаленным репозиториям через HTTPS или SSH. В *главе 7* вы узнаете о создании удаленных репозиториев и начнете изучать способы совместной работы над проектами Git.

[Глава 7]

Начало работы с удаленным репозиторием

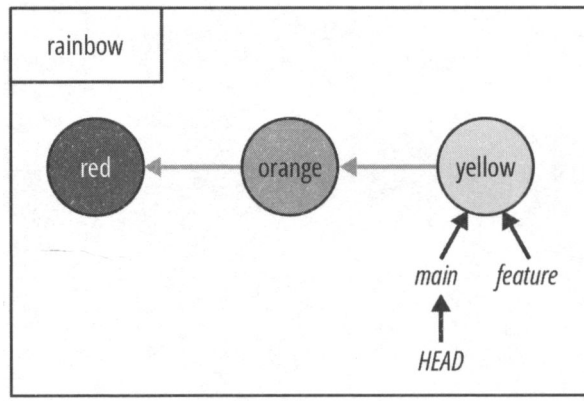
В предыдущей главе мы рассмотрели выбор службы хостинга для размещения удаленного репозитория, а также создание учетной записи и настройку данных аутентификации для безопасного подключения к удаленным репозиториям по протоколу HTTPS или SSH.

В этой главе мы рассмотрим различные способы работы с локальными и удаленными репозиториями Git и разберемся, чем полезны удаленные репозитории. Вы создадите удаленный репозиторий для проекта Rainbow и загрузите в него некоторые данные. Дополнительные материалы, которые помогут вам в работе с этой главой, доступны в репозитории Learning Git (**https://github.com/gitlearningjourney/learning-git**).

Состояние локального репозитория

В начале этой главы у вас должны быть три коммита и две ветки в репозитории `rainbow`, и вы должны находиться на ветке `main`. Текущее состояние репозитория Rainbow показано на диаграмме 7.1.

ДИАГРАММА 7.1



Состояние репозитория `rainbow` в начале *главы 7*.

Два способа начать работу над проектом Git

В *главе 2* вы узнали, что существуют два типа репозиториев: локальные репозитории, которые хранятся на компьютере, и удаленные репозитории, которые размещаются на хостинге. Вы можете начать работу над проектом с помощью Git из локального или удаленного репозитория. До сих пор в проекте Rainbow вы использовали первый подход, поскольку начинали работу над своим проектом из локального репозитория. В *главе 8* вы познакомитесь с практическим примером начала работы над проектом Git из удаленного репозитория. В этом разделе я предоставлю краткий обзор каждого подхода.

Начало работы из локального репозитория

Для того чтобы начать работу над проектом Git из локального репозитория, необходимо сначала создать локальный репозиторий на компьютере с помощью команды `git init` и сделать хотя бы один коммит. Далее вам необходимо создать удаленный репозиторий на хостинге. Наконец, вы можете загрузить данные из локального репозитория в удаленный репозиторий.

В Git принято использовать термин *"отправка"* (push)[1] для обозначения процесса выгрузки данных из локального репозитория в удаленный репозиторий. Для этой операции предназначена команда `git push`.

ЗАПОМНИТЕ КОМАНДУ

`git push`

Отправка данных в удаленный репозиторий.

Именно этот подход вы будете использовать в проекте Rainbow в данной главе; он схематически показан на рис. 7.1.

Рис. 7.1. Начало работы над проектом Git из локального репозитория

[1] В обиходе часто говорят просто *"сделать пуш"* или *"запушить"*. — *Прим. перев.*

Пример ситуации, когда вы можете начать работу над проектом Git из локального репозитория, — если на вашем компьютере есть проект, над которым вы работаете некоторое время, но каталог этого проекта не является репозиторием — другими словами, сначала вы не прибегали к Git для контроля версий, а затем решили, что нужно начать контролировать версии с помощью Git. В этом случае вы должны выполнить команду `git init` для инициализации локального репозитория на вашем компьютере, сделать первоначальный коммит, а затем пройти остальные шаги для создания удаленного репозитория и загрузки в него данных.

Другой способ начала работы над проектом Git — на первом шаге создать удаленный репозиторий.

Начало работы из удаленного репозитория

Для того чтобы начать работу над проектом Git из удаленного репозитория, вы можете либо найти существующий удаленный репозиторий, с которым хотите работать, либо создать новый удаленный репозиторий на хостинге. Затем необходимо *клонировать* (т. е. скопировать) удаленный репозиторий на свой компьютер, в результате чего будет создан локальный репозиторий. Именно этот подход мы рассмотрим в *главе 8*, и он показан на рис. 7.2.

Рис. 7.2. Начало работы над проектом Git из удаленного репозитория

Примером начала работы над проектом из удаленного репозитория может служить ситуация, когда ваш друг работает над проектом, имеющим удаленный репозиторий, и просит вас внести в него свой вклад. В этом случае вы попросите своего друга рассказать вам, где найти удаленный репозиторий, а затем клонируете (скопируете) его на свой компьютер и начнете над ним работать.

До этого момента в проекте Rainbow вы работали исключительно с локальным репозиторием. Далее в этой главе вы создадите удаленный репозиторий и загрузите в него данные. Но сначала я хочу кратко остановиться на взаимодействии между локальными и удаленными репозиториями.

Взаимодействие между локальными и удаленными репозиториями

Локальные и удаленные репозитории действуют независимо. Когда дело доходит до работы с ними, важно понимать, что никакое взаимодействие между ними не происходит автоматически. Другими словами, никакие обновления из локального репозитория не будут автоматически поступать в удаленный репозиторий, и наоборот, никакие обновления из удаленного репозитория не будут автоматически поступать в локальный репозиторий.

Между ними нет живой связи. Любые изменения в любом репозитории будут результатом явного выполнения ваших команд. В этой и последующих главах вы узнаете, что это за команды.

Прежде чем перейти к созданию удаленного репозитория, давайте обсудим, почему вам вообще нужно его создать.

Почему нужны удаленные репозитории?

Есть три основные причины, почему удаленные репозитории полезны и важны при работе над проектом Git. Они позволяют:

- легко создавать резервную копию вашего проекта где-либо, кроме вашего компьютера;
- организовать доступ к проекту Git с нескольких компьютеров;
- организовать совместную работу с другими людьми над проектами Git.

Пример 7.1 проекта Book демонстрирует ситуацию, когда вам необходимо создать удаленный репозиторий для проекта Git.

Пример 7.1 проекта Book

Предположим, у меня на компьютере есть только локальный репозиторий для моего проекта Book и нет удаленного репозитория. Если кто-то украдет мой ноутбук или выйдет из строя жесткий диск, я потеряю всю работу, проделанную над книгой. Удаленные репозитории — хороший механизм резервного копирования.

Удаленный репозиторий также позволит мне получить доступ к моему проекту с других компьютеров. Представьте, что у меня есть два ноутбука — один дома и один в офисе, на котором я работаю в течение дня, — и мой проект Book хранится на домашнем ноутбуке. Если у меня есть удаленный репозиторий, я также могу получить доступ к проекту Book со своего рабочего ноутбука. Тогда я могу либо внести в него изменения непосредственно на сайте хостинга, либо клонировать удаленный репозиторий на свой рабочий ноутбук, чтобы создать локальный репозиторий. Дополнительным преимуществом является то, что я могу показать коллегам свои достижения в работе над книгой.

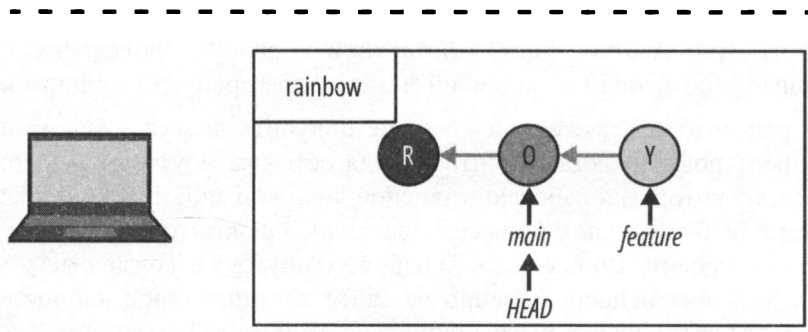
Удаленный репозиторий также позволяет мне легко сотрудничать с соавторами книги. Я могу предоставить доступ к удаленному репозиторию любому, кто хочет увидеть мою работу или внести в нее свой вклад. Другие люди смогут либо клонировать удаленный репозиторий на свой компьютер, чтобы создать локальный репозиторий, либо просто просматривать содержимое удаленного репозитория непосредственно на веб-сайте службы хостинга. В данном случае мне нужен удаленный репозиторий для совместной работы над проектом Book с моим соавтором и редактором.

На примере 7.1 проекта Book я наглядно показала, почему работа с удаленными репозиториями так удобна. Теперь, когда вы убедились в этом, перейдем к созданию удаленного репозитория. Представьте, что вы рассказали другу о своем опыте работы над проектом Rainbow, и он захотел посмотреть, что у вас получилось. Для того чтобы предоставить ему такую возможность, вам необходимо создать удаленный репозиторий. Давайте сделаем это прямо сейчас.

Создание удаленного репозитория с данными

В настоящее время проект Rainbow состоит из одного локального репозитория `rainbow`, показанного на диаграмме 7.2.

ДИАГРАММА 7.2



Проект Rainbow в настоящее время состоит из одного локального репозитория, который называется `rainbow`.

ПРИМЕЧАНИЕ

Начиная с этого момента диаграмма репозитория будет расширена: в верхней части появится область, отражающая состояние удаленного репозитория, как показано на диаграмме 7.2.

Ваш друг хочет посмотреть, над чем вы работаете. Для того чтобы он смог это сделать, вам необходимо создать удаленный репозиторий и загрузить в него данные. Этот процесс состоит из трех шагов:

1. Создайте удаленный репозиторий на хостинге.
2. Добавьте подключение к удаленному репозиторию в локальном репозитории.
3. Отправьте данные из локального репозитория в удаленный.

В следующих разделах мы подробно рассмотрим каждый шаг, и попутно я познакомлю вас с новыми типами веток.

Создание удаленного репозитория

Во время создания удаленного репозитория на сервере хостинга вы даете ему *имя проекта удаленного репозитория*, а служба хостинга предоставляет *URL-адрес удаленного репозитория*. В этот адрес автоматически включается имя проекта удаленного репозитория. Как вы узнали из *главы 6*, служба хостинга сгенерирует для вашего репозитория два URL-адреса — для доступа по протоколам SSH и HTTPS. Вам понадобится адрес для того протокола, который вы выбрали ранее.

Процесс создания удаленного репозитория полностью выполняется на сайте хостинга. Советую внимательно ознакомиться с документацией вашего хостинга для получения подробной информации о конкретных действиях, которые необходимо выполнить. Дополнительные ресурсы можно найти в репозитории Learning Git (**https://github.com/gitlearningjourney/learning-git**).

Здесь я дам лишь некоторые общие рекомендации. Как упоминалось ранее, при создании удаленного репозитория необходимо указать имя проекта. Для проекта Rainbow это будет `rainbow-remote`.

ПРИМЕЧАНИЕ

Обычно удаленный и локальный репозитории имеют одно и то же имя. Но чтобы вам было легче различать их, в проекте Rainbow мы дадим им разные имена.

Вам также придется выбрать, будет ли удаленный репозиторий общедоступным или частным. Это параметр, который вы можете настроить на хостинге для каждого репозитория. *Общедоступный репозиторий* (public repository) открыт для любого пользователя в Интернете. *Частный репозиторий* (private repository) виден только лицам, имеющим к нему доступ. В обоих случаях вы можете внести свой вклад в репозиторий, только если вам предоставлен доступ. Поскольку репозиторий, над которым вы работаете в этой книге, предназначен лишь для учебных целей, я рекомендую вам сделать его частным. Таким образом, вы можете контролировать, кто

может его просматривать (например, вы можете предоставить доступ другу, который хочет посмотреть, над чем вы работаете).

Некоторые службы хостинга могут спросить, хотите ли вы добавить какие-либо файлы при создании репозитория, например файл README или .gitignore. Для целей этой книги вам не следует добавлять какие-либо файлы; если добавление задано в настройках, обязательно отключите его. Поскольку вы будете загружать в удаленный репозиторий локальные данные, необходимо, чтобы удаленный репозиторий изначально был пустым.

Если служба хостинга предоставляет возможность выбора лицензии, вы можете игнорировать этот шаг (т. е. вообще не выбирать лицензию), поскольку это всего лишь учебное упражнение.

Наконец, некоторые службы хостинга могут попросить вас ввести значение имени ветки по умолчанию. Для упражнений из этой книги вы можете либо оставить это значение пустым, либо ввести имя `main`.

Теперь выполните действия 7.1, чтобы создать удаленный репозиторий.

ДЕЙСТВИЯ 7.1

1. Войдите в свою учетную запись на сайте выбранного вами хостинга.
2. Создайте удаленный репозиторий. Для получения дополнительной информации перейдите в репозиторий Learning Git (**https://github.com/gitlearningjourney/learning-git**) или обратитесь непосредственно к документации вашей службы хостинга.

   При создании репозитория для этого упражнения:

   - в качестве имени репозитория используйте `rainbow-remote`;
   - вы можете сделать репозиторий общедоступным или частным. Рекомендую сделать его частным.
   - не добавляйте никаких файлов. Например, не добавляйте файлы README или .gitignore;
   - если вас попросят указать имя ветки по умолчанию, можете оставить поле пустым или задать имя `main`.
3. После завершения шагов по созданию удаленного репозитория найдите URL-адрес удаленного репозитория. Служба хостинга предоставит две версии URL-адреса: одну для HTTPS и одну для SSH. В следующем упражнении вам понадобится URL-адрес протокола, который вы выбрали и настроили в *главе 6*. В примерах в этой книге используются два URL-адреса удаленного репозитория:

   - HTTPS: **https://github.com/gitlearningjourney/rainbow-remote.git**;
   - SSH: **git@github.com:gitlearningjourney/rainbow-remote.git**.

На что обратить внимание:

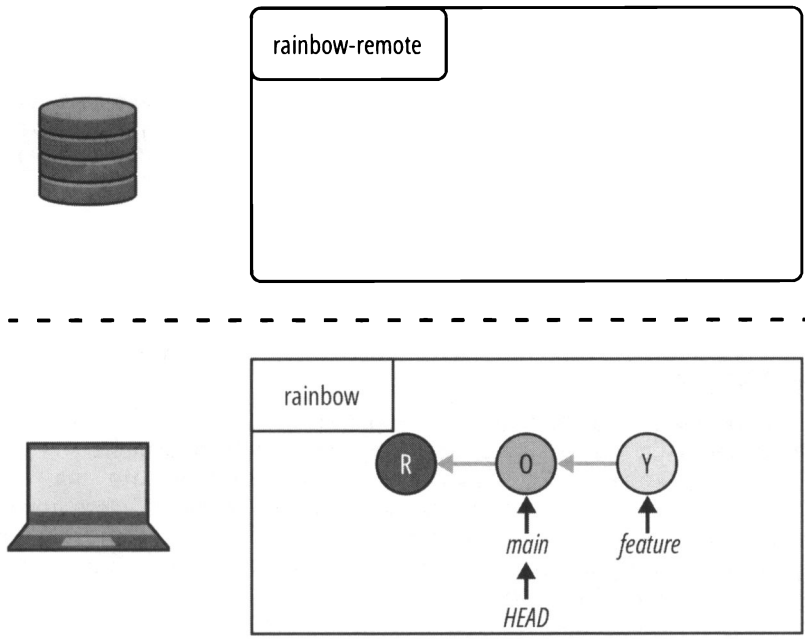
- вы создали удаленный репозиторий под названием `rainbow-remote`, но пока в нем нет данных.

Состояние проекта Rainbow после завершения вышеуказанных действий показано на диаграмме 7.3.

ПРИМЕЧАНИЕ

С этого момента на диаграмме репозитория мы используем прямоугольник с обычными углами в 90 градусов для обозначения локального репозитория и прямоугольник с закругленными углами для удаленного репозитория.

ДИАГРАММА 7.3



Проект Rainbow после создания пустого удаленного репозитория `rainbow-remote`.

Вы создали удаленный репозиторий на своем хостинге, но, как показывает эта диаграмма, в настоящее время он пуст. Создание удаленного репозитория на хостинге не загружает в него никакие данные. Для того чтобы загрузить данные, вам необходимо сначала настроить подключение к удаленному репозиторию в вашем локальном репозитории. Давайте посмотрим, как это сделать.

Настройка подключения к удаленному репозиторию

Локальный репозиторий может взаимодействовать с удаленным репозиторием, если между ними установлено соединение. У этого соединения есть свое имя, которое мы будем называть *коротким именем удаленного репозитория* или просто

коротким именем (shortname). Локальный репозиторий может иметь соединения с несколькими удаленными репозиториями, хотя это случается не очень часто.

Если локальный репозиторий был инициализирован локально, для настройки соединения с удаленным репозиторием необходимо явно связать URL-адрес удаленного репозитория с коротким именем удаленного репозитория. Для этого следует выполнить команду `git remote add`, передав ей короткое имя и следом URL-адрес удаленного репозитория.

ЗАПОМНИТЕ КОМАНДУ

`git remote add <короткое_имя> <URL>`

Добавляет подключение к удаленному репозиторию с именем `<короткое_имя>` по адресу `<URL>`.

ПРИМЕЧАНИЕ

В примерах этой книги задействован URL-адрес HTTPS; однако вы можете выбирать между SSH и HTTPS в зависимости от того, какой протокол вы предпочитаете. Обязательно используйте URL-адрес, предоставленный вашей службой хостинга, который будет отличаться от URL-адреса, показанного в примерах этой книги.

После того как соединение с удаленным репозиторием сохранено в локальном репозитории, вы сможете подключиться к удаленному репозиторию, указав в командной строке короткое имя, а не URL-адрес.

ПРИМЕЧАНИЕ

Когда вы клонируете удаленный репозиторий для создания локального репозитория, Git автоматически добавляет соединение с удаленным репозиторием с коротким именем `origin` (источник) по умолчанию. Вы увидите это на практике в *разд. "Клонирование удаленного репозитория" главы 8*. Очень часто для удаленного репозитория указывают короткое имя `origin`, даже если репозиторий сначала был создан и инициализирован локально, поскольку это короткое имя по умолчанию для репозитория при его клонировании.

Это похоже на то, как в *разд. "Немного истории Git: master и main" главы 4* мы говорили про инициализацию репозитория с помощью команды `git init` без дополнительных параметров. Git по умолчанию создает ветку с именем `master`. Опять же, в терминах `origin` или `master` нет ничего особенного; это просто текущие соглашения об именах по умолчанию, используемые Git.

Репозиторий `rainbow` был инициализирован локально, поэтому вам придется явно добавить подключение к удаленному репозиторию в настройки локального репозитория. Прежде чем продолжить, рассмотрим некоторые дополнительные полезные команды.

ПРИМЕЧАНИЕ

В официальной документации Git подключение к удаленному репозиторию, сохраненное в настройках локального репозитория, обозначается просто словом remote.

Для того чтобы просмотреть список подключений к удаленным репозиториям, доступных в локальном репозитории по короткому имени, выполните команду `git remote`. Если вы передадите команде `git remote` опцию `-v` (которая означает verbose — подробный), то она выведет список подключений к удаленным репозиториям, хранящихся в локальном репозитории, по коротким именам, а также их URL-адреса.

ЗАПОМНИТЕ КОМАНДУ

git remote

Выводит список подключений к удаленному репозиторию, хранящихся в локальном репозитории, в виде коротких имен.

git remote -v

Выводит список подключений к удаленному репозиторию, хранящихся в локальном репозитории, в виде коротких имен и URL-адресов.

Выполните действия 7.2 и свяжите свой URL-адрес с коротким именем `origin` в репозитории `rainbow`. На шаге 4 обязательно вводите всю команду одной строкой без переноса.

ДЕЙСТВИЯ 7.2

1. В окне файловой системы перейдите в rainbow > .git > config. Откройте файл config. Вы увидите, что на данный момент в файле не указано ни одного подключения к удаленным репозиториям. Другими словами, вы не увидите в файле никаких упоминаний об URL-адресе или коротком имени вашего удаленного репозитория.
2. Перейдите на сайт своего хостинга и скопируйте URL-адрес удаленного репозитория для выбранного вами протокола (SSH или HTTPS). Это адрес потребуется вам на шаге 4.
3. `rainbow $ git remote`
4. `rainbow $ git remote add origin https://github.com/gitlearningjourney/rainbow-remote.git`
5. `rainbow $ git remote`

   ```
   origin
   ```
6. `rainbow $ git remote -v`

   ```
   origin https://github.com/gitlearningjourney/rainbow-remote.git (fetch)
   origin https://github.com/gitlearningjourney/rainbow-remote.git (push)
   ```
7. Закройте файл config и откройте его снова в другом окне. Вы увидите, что в файле указано одно соединение с удаленным репозиторием.

Запись должна выглядеть примерно так:

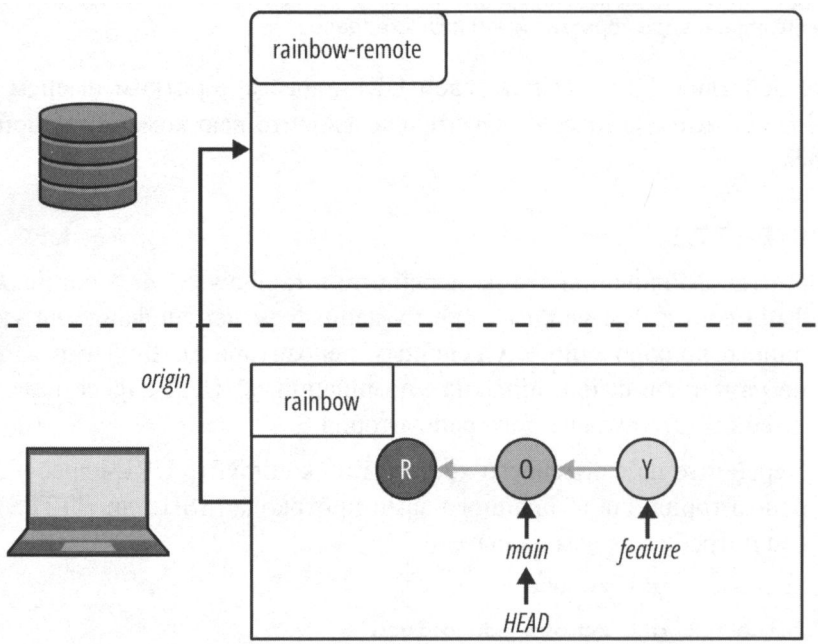
```
[remote "origin"]
  url = https://github.com/gitlearningjourney/rainbow-remote.git
  fetch = +refs/heads/*:refs/remotes/origin/*
```

На что обратить внимание:

- на шаге 3 в выводе `git remote` нет коротких имен;
- на шаге 5 в выводе `git remote` указано одно короткое имя — `origin`.

Добавление этого подключения отражено на диаграмме 7.4.

ДИАГРАММА 7.4



Проект Rainbow после добавления подключения к удаленному репозиторию `rainbow-remote` в вашем репозитории `rainbow`.

На что обратить внимание:

- репозиторий `rainbow` имеет короткое имя `origin`, связанное с URL-адресом удаленного репозитория;
- в репозитории `rainbow-remote` по-прежнему нет данных.

На диаграмме 7.4 вы можете видеть, что стрелка, обозначающая короткое имя, хранящееся в репозитории `rainbow` и относящееся к репозиторию `rainbow-remote`, указывает только в одном направлении. Это связано с тем, что соединение между локальным репозиторием и удаленным репозиторием инициируется всегда от

локального репозитория к удаленному, а не наоборот. В локальном репозитории можно найти список всех удаленных репозиториев, к которым настроено соединение; однако в удаленном репозитории вы не найдете список локальных репозиториев, которые могут соединяться с этим удаленным репозиторием.

Также обратите внимание: добавление записи о соединении с удаленным репозиторием в локальный репозиторий не означает, что какие-либо данные из локального репозитория были загружены в удаленный. Для того чтобы загрузить данные в удаленный репозиторий, вы должны отправить (push) ветку в удаленный репозиторий. Этот процесс загрузит все коммиты, являющиеся частью отправляемой ветки.

Для того чтобы лучше разобраться, что происходит, когда вы выполняете этот последний шаг, нужно рассмотреть несколько других типов веток.

Знакомьтесь: удаленные ветки и ветки удаленного отслеживания

В *главе 4* вы узнали о ветках, которые являются подвижными указателями на коммиты. До сих пор вы работали только с локальными ветками. Когда вы отправляете локальную ветку в удаленный репозиторий, вы создаете *удаленную ветку* (remote branch). Удаленная ветка — это ветка в удаленном репозитории.

Удаленные ветки *не* обновляются автоматически, когда вы делаете новые коммиты в локальных ветках. Вам необходимо явно передать коммиты из локальной ветки в удаленную. Каждая удаленная ветка (о которой знает локальный репозиторий) также имеет *ветку удаленного отслеживания* (remote-tracking branch). Это ссылка в локальном репозитории на коммит удаленной ветки, указанный в последний раз, когда происходило какое-либо сетевое соединение с удаленным репозиторием. Для удобства эту ссылку можно рассматривать как закладку.

Вы можете настроить связь отслеживания между локальной и удаленной ветками, указав, какую удаленную ветку должна отслеживать локальная ветка. Эта связь является ссылкой на *восходящую ветку* (upstream branch). В некоторых случаях Git автоматически задает ссылку на восходящую ветку, но в остальных случаях вам придется указать ее явно.

Когда вы отправляете результаты своей работы из локальной ветки в удаленную, Git должен знать, в какую удаленную ветку ее следует отправить. Если для локальной ветки определена вышестоящая ветка, вы можете использовать `git push` без аргументов, и Git автоматически отправит работу в нужную ветку. Однако, если для локальной ветки, над которой вы работаете, не определена восходящая ветка, вам нужно будет указать, в какую удаленную ветку следует отправить данные при вводе команды `git push`. (Если вы этого не сделаете, то получите сообщение об ошибке.)

В этой главе вы узнаете, как указать удаленную ветку, в которую вы хотите отправить данные с помощью команды `git push`. В *главе 9* вы узнаете, как определить восходящую ветку.

Теперь вы готовы выполнить третий шаг создания удаленного репозитория: отправку локальной ветки в удаленный репозиторий.

Отправка в удаленный репозиторий

Для того чтобы отправить локальную ветку в удаленный репозиторий, нужно ввести команду `git push` и передать ей короткое имя удаленного репозитория и имя ветки, которую вы хотите отправить. Для репозитория `rainbow` вы будете использовать короткое имя `origin`, а ветка, которую вы отправите, называется `main`.

ЗАПОМНИТЕ КОМАНДУ

git push <короткое_имя> <имя_ветки>

Передает содержимое <имя_ветки> в удаленный репозиторий <короткое_имя>.

После выполнения команды `git push` произойдут две вещи:

1. В вашем удаленном репозитории будет создана удаленная ветка.
2. В вашем локальном репозитории будет создана ветка удаленного отслеживания.

ПРИМЕЧАНИЕ

Для того чтобы отправить данные в удаленный репозиторий, ваш компьютер должен быть подключен к Интернету и иметь доступ по SSH или HTTPS к выбранной вами службе хостинга.

В *главе 4* вы использовали команду `git branch` для вывода списка всех локальных веток. Теперь вы будете выполнять команду `git branch` с опцией `--all` для получения списка всех локальных веток и веток удаленного отслеживания в вашем локальном репозитории.

ЗАПОМНИТЕ КОМАНДУ

git branch --all

Список локальных веток и веток удаленного отслеживания.

Выполните действия 7.3, чтобы отправить свою ветку в удаленный репозиторий.

ДЕЙСТВИЯ 7.3

1. В окне файловой системы перейдите в rainbow > .git > refs и просмотрите каталоги, находящиеся внутри каталога refs. Там должны быть размещены два вложенных каталога: heads и tags.
2.
```
rainbow $ git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
```

```
Compressing objects: 100% (5/5), done.
Writing objects: 100% (9/9), 747 bytes | 373.00 KiB/s, done.
Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To github.com:gitlearningjourney/rainbow-remote.git
 * [new branch] main -> main
```

3 В окне файловой системы еще раз просмотрите каталоги, находящиеся внутри каталога refs. Теперь там должно быть три вложенных каталога: heads, tags и remotes.

4 Перейдите в репозиторий `rainbow-remote` на сайте хостинга и обновите страницу. Перейдите на страницу, на которой перечислены ваши коммиты. Теперь вы должны увидеть там три своих коммита.

5
```
rainbow $ git branch --all
  feature
* main
  remotes/origin/main
```

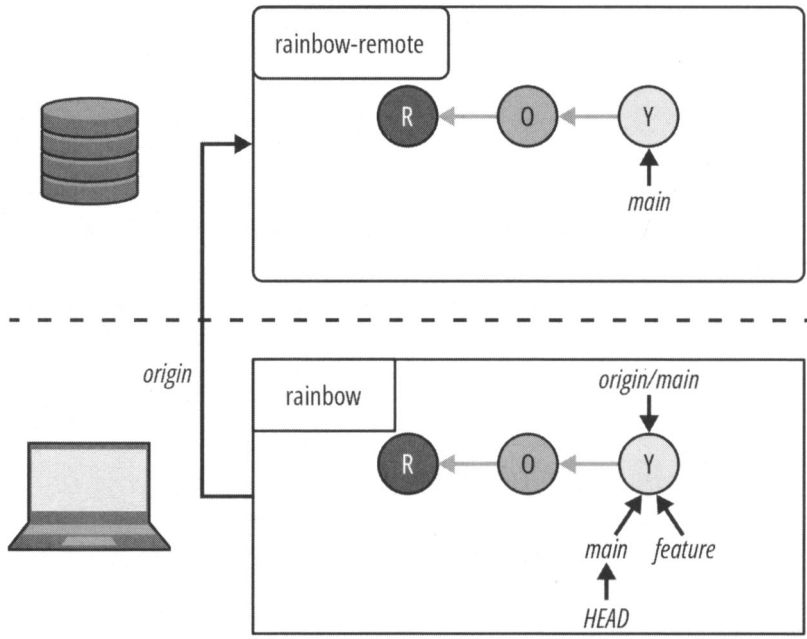
6
```
rainbow $ git log
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (HEAD -> main,
origin/main, feature)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow

commit 7acb333f08e12020efb5c6b563b285040c9dba93
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange

commit c26d0bc371c3634ab49543686b3c8f10e9da63c5
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:23:18 2022 +0100

    red
```

На что обратить внимание:

- на шаге 2 выходные данные команды `git push` показывают, что вы отправили свою ветку в удаленный репозиторий. Не имеет значения, если числа в вашем выводе команды немного отличаются от книги;

- на шаге 3 вы видите, что в каталоге refs появился новый каталог под названием remotes. Внутри каталога remotes находится каталог с именем origin, а внутри этого каталога — файл с именем main. Эта цепочка вложенных каталогов представляет собой новую ветку удаленного отслеживания `origin/main`;
- на шаге 4 вы видите, что в репозитории `rainbow-remote` есть удаленная ветка `main`. Если вы зайдете на свой хостинг, то найдете там три коммита локальной ветки `main`;
- в удаленном репозитории нет ветки `feature`. Все эти наблюдения отражены на диаграмме 7.5.

ДИАГРАММА 7.5



Проект Rainbow после отправки локальной ветки `main` в удаленный репозиторий.

ПРИМЕЧАНИЕ

Для таких команд, как `git commit`, `git push` и `git merge`, не имеет значения, если ваши выходные данные немного отличаются от выходных данных, показанных в этой книге. То же самое касается и других команд, которые вы будете использовать в последующих главах этой книги, таких как `git clone`, `git fetch` и `git pull`.

Нужно заметить, что когда вы отправляете определенную ветку в удаленный репозиторий, туда загружаются данные только из этой ветки. Вы отправили в удаленный репозиторий ветку `main`, но ветка `feature` не была отправлена в удаленный репозиторий.

На данный момент в вашем репозитории `rainbow` ветки `main` и `feature` имеют одинаковые коммиты. Другими словами, их истории развития, которые можно проследить, проходя по коммитам и родительским ссылкам в обратном направлении, состоят из одних и тех же коммитов. Поэтому на диаграмме 7.5 не так очевидно, что ветка `feature` отсутствует в удаленном репозитории. Однако в реальных проектах ветки часто имеют разную историю разработки и, следовательно, состоят из разных коммитов. Это означает, что если вы не отправите определенную ветку в удаленный репозиторий, у вас не будет некоторых коммитов в вашем удаленном репозитории.

Если вы хотите отправить свою ветку `feature` в удаленный репозиторий, вам придется сделать это явно. Для этого выполните действия 7.4.

ДЕЙСТВИЯ 7.4

1.
```
rainbow $ git switch feature
Switched to branch 'feature'
```

(Переключено на ветку 'feature')

2.
```
rainbow $ git push origin feature
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature' on GitHub by
visiting:
remote: https://github.com/gitlearningjourney/rainbow- remote/
pull/new/feature
remote:
To github.com:gitlearningjourney/rainbow-remote.git
 * [new branch]      feature -> feature
```

3.
```
rainbow $ git branch --all
* feature
  main
  remotes/origin/feature
  remotes/origin/main
```

4. Перейдите в репозиторий `rainbow-remote` на своем хостинге и обновите страницу. Посмотрите список веток.

5.
```
rainbow $ git log
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (HEAD ->
feature, origin/main, origin/feature, main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    Yellow
```

```
commit 7acb333f08e12020efb5c6b563b285040c9dba93
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange
```

ПРИМЕЧАНИЕ

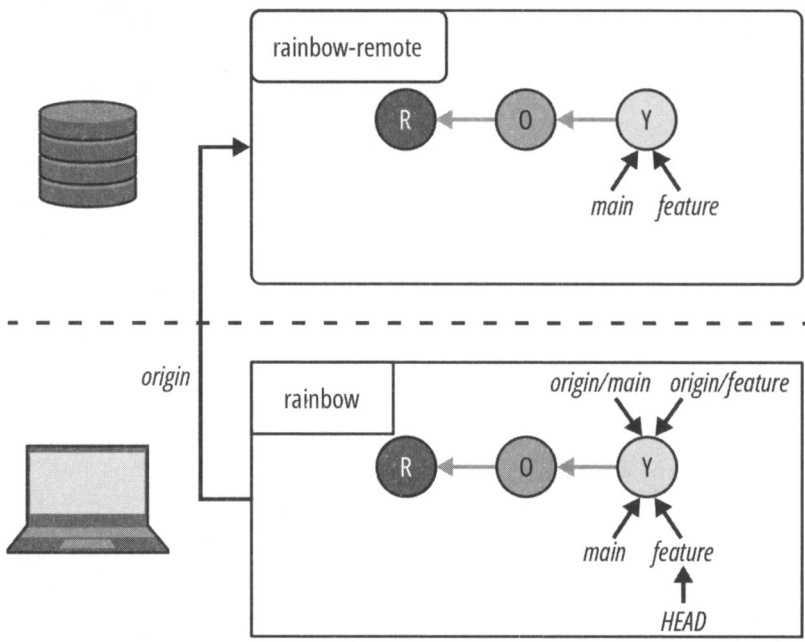
Начиная с этого момента в книге, вывод команды `git log` во врезке действий будет отображать только несколько последних коммитов, сделанных в данном репозитории.

На что обратить внимание:

- на шаге 3 вы видите, что в репозитории `rainbow` есть две ветки удаленного отслеживания: `origin/main` и `origin/feature`;
- на шаге 4 вы видите, что в репозитории `rainbow-remote` есть две удаленные ветки: `main` и `feature`.

Текущее состояние репозиториев показано на диаграмме 7.6.

ДИАГРАММА 7.6



Проект Rainbow после отправки локальной ветки `feature` в удаленный репозиторий.

Итак, вы создали удаленный репозиторий и отправили в него данные. В *главе 6* я упоминала, что есть два способа внесения изменений в удаленный репозиторий:

1. Авторизоваться на хостинге через его сайт и внести изменения непосредственно там.

2. Внести изменения в локальный репозиторий и загрузить эти изменения в удаленный репозиторий на хостинге.

В этой главе мы рассмотрели пример второго способа, но краткого обзора заслуживает и первый.

Работа с удаленным репозиторием непосредственно на хостинге

К этому моменту вы узнали, как использовать локальный репозиторий для создания коммитов (*см. главу 3*), работы с ветками (*см. главу 4*) и выполнения базового слияния (*см. главу 5*).

Однако для выполнения этих действий (и многих других, подобных этим) также можно использовать пользовательский интерфейс веб-сайта хостинговой службы. Например, на сайте хостинга вы можете делать коммиты непосредственно в удаленном репозитории, создавать удаленные ветки и, как вы увидите в *главе 12*, можете объединять ветки с помощью функции, называемой запросом на внесение изменений (pull request).

Поскольку эта книга посвящена обучению работе с Git в командной строке, я не буду описывать здесь выполнение соответствующих действий на сайте службы хостинга. Однако все эти действия подробно описаны в документации вашего хостинга. Функциональные возможности сайта хостинга могут пригодиться коллегам, с которыми вы будете работать в будущем и которые не знают, как использовать Git в командной строке.

Теперь, когда вы изучили основы работы с локальными и удаленными репозиториями, создали удаленный репозиторий для проекта Rainbow и добавили в него несколько веток, следующим шагом вашего обучения станет начало совместной работы над проектом.

Краткое содержание главы

В этой главе рассказано, что такое удаленные репозитории, почему мы их используем, а также описаны три шага по созданию удаленного репозитория с данными в нем, если вы начинаете работу над проектом с локального репозитория. Вы настроили удаленный репозиторий на своем хостинге, добавили соединение с удаленным репозиторием в настройки локального репозитория и отправили данные из локального репозитория в удаленный. Попутно вы узнали о назначении удаленных веток, веток удаленного отслеживания и восходящих веток.

Далее, в *главе 8*, ваш друг начнет помогать вам в работе над проектом, клонируя удаленный репозиторий.

[Глава 8]

Клонирование и локальное сохранение

В предыдущей главе вы узнали об удаленных репозиториях и создали один из них для проекта Rainbow.

В этой главе вы начнете моделировать совместную работу с другом над проектом Rainbow. Вы научитесь клонировать удаленные репозитории и узнаете, чем это отличается от локальной инициализации репозиториев. Вы также узнаете больше об определении восходящих веток, удалении веток и получении данных из удаленного репозитория.

ПРИМЕЧАНИЕ

Некоторые текстовые редакторы, интегрированные с Git, позволяют вам задавать определенные настройки для Git. В этой главе предполагается, что вы не задавали какие-либо специальные настройки, которые могли бы привести к отклонению Git от поведения по умолчанию.

Например, в Visual Studio Code есть функция автозагрузки (`git.autofetch`), которая периодически получает изменения из удаленного репозитория. (О получении информации вы узнаете позже в этой главе.) Эта функция отключена по умолчанию, и чтобы вы могли правильно выполнять упражнения из этой главы, она должна оставаться отключенной. Если вы включили эту функцию ранее, отключите ее, прежде чем продолжить.

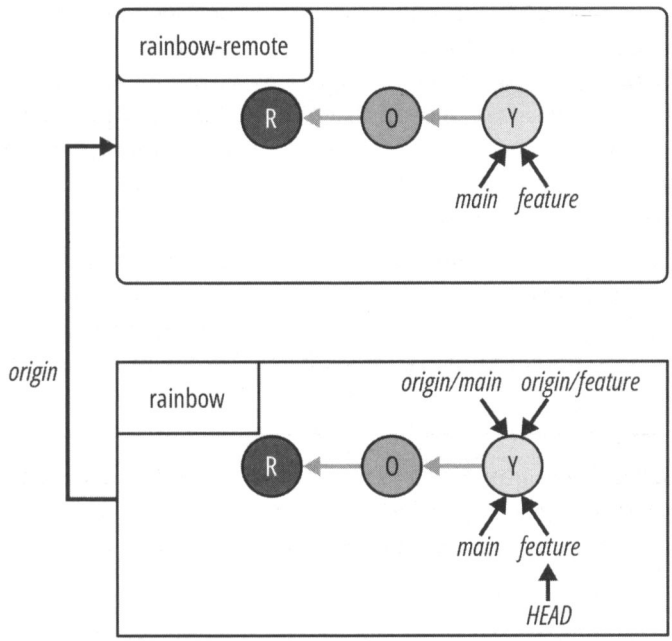
Если вы никогда не устанавливали какие-либо специальные настройки Git для своего текстового редактора, вам не нужно об этом беспокоиться.

Состояние локального и удаленного репозиториев

В начале этой главы у вас должны быть локальный репозиторий `rainbow` и удаленный репозиторий `rainbow-remote`. Они должны быть синхронизированы, т. е. со-

держать одни и те же коммиты и ветки. Ожидаемое текущее состояние двух репозиториев показано на диаграмме 8.1.

ДИАГРАММА 8.1



Проект Rainbow в начале *главы 8* с локальным репозиторием `rainbow` и удаленным репозиторием `rainbow-remote`.

Клонирование удаленного репозитория

В предыдущей главе вы создали удаленный репозиторий на хостинге, чтобы показать своему другу, как вы работаете над проектом Rainbow. Теперь давайте представим, что ваш друг решил помочь вам в работе над проектом. Если он собирается внести свой вклад в проект, ему нужно будет клонировать (т. е. скопировать) ваш удаленный репозиторий. В следующем разделе я опишу, как вы будете моделировать этот сценарий.

В начале книги вы узнали, что Git — полезный инструмент для совместной работы. Клонирование удаленных репозиториев является важной частью совместной работы с другими людьми над проектом Git, поскольку оно позволяет им работать с собственной копией репозитория на своем компьютере. Клонировать удаленный репозиторий может неограниченное количество людей (при условии, что им предоставлен к нему доступ).

Давайте для иллюстрации взглянем на пример 8.1 проекта Book.

Пример 8.1 проекта Book

Предположим, я изначально планировала работать над своим проектом Book самостоятельно, но позже передумала и решила, что хочу работать с соавтором.

Моему соавтору потребуется клонировать удаленный репозиторий, чтобы у него на компьютере появилась собственная копия проекта Book и он мог начать работать со мной над книгой. Для клонирования соавтору потребуется иметь учетную запись в той службе хостинга, на которой размещен удаленный репозиторий. Затем мне придется предоставить соавтору доступ к редактированию репозитория, чтобы он смог комментировать и вносить свой вклад независимо от того, является репозиторий общедоступным или частным.

Если в какой-то момент я захочу, чтобы мой редактор проверил готовую работу, нам нужно будет выполнить те же действия: он создаст учетную запись на хостинге, затем я предоставлю редактору доступ к репозиторию и сообщу адрес. После этого редактор сможет клонировать репозиторий на свой компьютер для проверки файлов.

Пример 8.1 проекта Book показывает, почему клонирование репозиториев является такой важной частью совместной работы над любым проектом Git. Теперь я расскажу, как вы будете имитировать опыт совместной работы с кем-то еще над проектом Rainbow.

Моделирование совместной работы

Обычно, если над одним проектом работают два человека, каждый из них имеет локальный репозиторий на своем компьютере и каждый вносит свой вклад в один удаленный репозиторий. Учитывая, что у вас может не оказаться двух компьютеров или другого человека, который помог бы вам выполнить упражнения по совместной работе в этой и последующих главах, вам нужно клонировать удаленный репозиторий на свой компьютер и создать второй локальный репозиторий, который вы назовете `friend-rainbow`, чтобы отличать его от репозитория `rainbow`.

Затем вы сделаете вид, что этот второй локальный репозиторий находится на компьютере вашего друга. Отныне, когда я упоминаю вашего друга, выполняющего какое-то действие, вы сами будете выполнять это действие в репозитории `friend-rainbow`.

После того как ваш воображаемый друг клонирует удаленный репозиторий, вы будете работать с двумя локальными репозиториями. Я рекомендую работать с двумя отдельными окнами файловой системы, по одному для каждого каталога проекта. Я также рекомендую работать с двумя отдельными окнами командной строки (или встроенными терминалами), по одному для каждого репозитория, вместо того чтобы переходить из каталога в каталог в одном окне командной строки.

В Git словом *"клонирование"* принято называть процесс копирования удаленного репозитория на компьютер с целью создания локального репозитория. Для этого предназначена команда `git clone`. Для того чтобы клонировать удаленный репозиторий, вы вводите команду `git clone` и передаете ей URL-адрес удаленного репозитория и, при необходимости, имя каталога проекта.

ЗАПОМНИТЕ КОМАНДУ

git clone <URL> <имя_каталога>

Клонирует удаленный репозиторий.

Если вы не передадите имя каталога проекта, то локальному репозиторию будет присвоено имя проекта удаленного репозитория. Например, если вы не передадите имя каталога проекта при клонировании репозитория `rainbow-remote`, то локальный репозиторий вашего друга также будет называться `rainbow-remote`. Поскольку это совпадение сделает учебное упражнение очень запутанным, нужно будет передать команде `git clone` имя каталога `friend-rainbow` в качестве аргумента для создания локального репозитория с таким именем.

Команда `git clone` выполняет следующие действия:

1. Создает каталог проекта внутри текущего каталога.
2. Создает (инициализирует) локальный репозиторий.
3. Загружает все данные из удаленного репозитория.
4. Добавляет подключение к клонированному удаленному репозиторию; присвоенное по умолчанию короткое имя `origin` будет находиться в новом локальном репозитории.

Для того чтобы сразу увидеть создаваемый репозиторий, вы можете клонировать его в каталог рабочего стола `desktop`. Это позволит вам сразу увидеть новый каталог проекта на рабочем столе вашего компьютера.

Выполните действия 8.1, чтобы клонировать удаленный репозиторий. На шаге 3 обязательно вводите всю команду в одной строке без переноса.

ДЕЙСТВИЯ 8.1

1. Откройте новое окно командной строки и перейдите в каталог desktop.
2. Перейдите на сайт службы хостинга и скопируйте URL-адрес удаленного репозитория для выбранного вами протокола: SSH или HTTPS. Этот адрес понадобится на шаге 3 данного руководства.
3.
```
desktop $ git clone https://github.com/gitlearningjourney/
rainbow-remote.git friend-rainbow
Cloning into 'friend-rainbow'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
```

```
Receiving objects: 100% (9/9), done.
Resolving deltas: 100% (1/1), done.
remote: Total 9 (delta 1), reused 9 (delta 1), pack-reused 0
```

4. Найдите вновь созданный каталог проекта `friend-rainbow` в своей файловой системе или на рабочем столе.

На что обратить внимание:

- вы клонировали удаленный репозиторий на свой компьютер и создали второй локальный репозиторий под названием `friend-rainbow`;
- вы не находитесь в каталоге friend-rainbow в командной строке, поскольку клонирование репозитория Git не означает, что вы автоматически переходите в него.

Выполните действия 8.2, чтобы перейти в новый каталог проекта `friend-rainbow` для изучения его содержимого и настройки.

ДЕЙСТВИЯ 8.2

1. `desktop $ cd friend-rainbow`
2. Откройте каталог проекта `friend-rainbow` в новом окне текстового редактора.
3. `friend-rainbow $ git remote -v`

```
origin https://github.com/gitlearningjourney/rainbow-remote.git
(fetch)
origin https://github.com/gitlearningjourney/rainbow-remote.git
(push)
```

4. `friend-rainbow $ git branch --all`

```
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/feature
  remotes/origin/main
```

5. `friend-rainbow $ git log`

```
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (HEAD -> main,
origin/main, origin/feature, origin/HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow

commit 7acb333f08e12020efb5c6b563b285040c9dba93
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

    orange
```

На что обратить внимание:

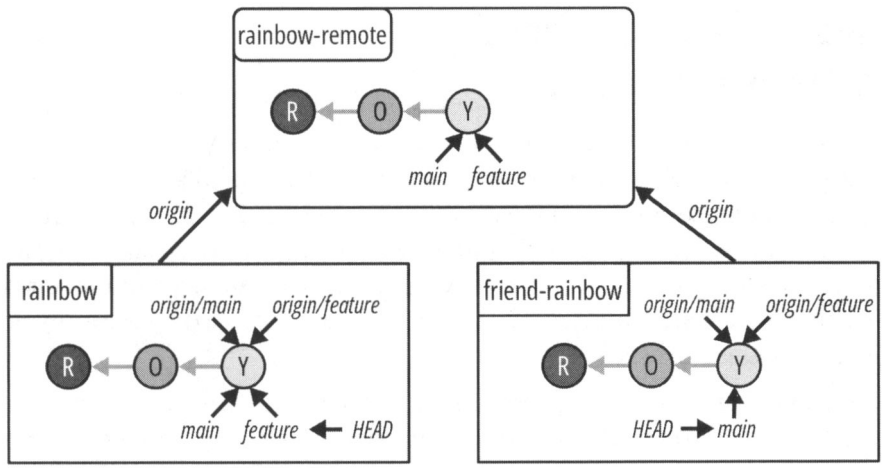
- на шаге 1 вы выполнили команду `cd` для перехода к репозиторию `friend-rainbow` в командной строке. Это необходимо, поскольку клонирование репозитория не означает, что вы автоматически переходите в его каталог;
- на шаге 3 в выводе `git remote` уже указано короткое имя `origin` удаленного репозитория;
- на шаге 4 выходные данные команды `git branch` сообщают, что есть указатель с именем `origin/HEAD`, который указывает на ветку удаленного отслеживания `origin/main`, и что у вас есть ветка удаленного отслеживания `origin/feature`. Вы также можете видеть, что локальная ветка `feature` отсутствует.

Все предыдущие наблюдения отражены на диаграмме 8.2.

ПРИМЕЧАНИЕ

С этого момента я расширю изображение диаграммы, чтобы отобразить состояние двух локальных репозиториев, `rainbow` и `friends-rainbow`, а также удаленного репозитория `rainbow-remote`.

ДИАГРАММА 8.2



Состояние проекта Rainbow после того, как ваш друг клонировал удаленный репозиторий и создал локальный репозиторий под названием `friend-rainbow`.

В следующих разделах мы углубимся в детали процесса клонирования и ответим на следующие вопросы:

- Что такое указатель `origin/HEAD`?
- Почему в новом локальном репозитории `friend-rainbow` нет локальной ветки `feature`?
- Почему для нового локального репозитория было автоматически создано короткое имя `origin`?

Что такое *origin/HEAD*?

Ранее вы заметили, что в репозитории `friend-rainbow` есть указатель `origin/HEAD`. Что это?

Когда вы клонируете репозиторий, Git обязан знать, в какой ветке он должен находиться после завершения клонирования. Указатель `origin/HEAD` указывает на нужную ветку. В проекте Rainbow `origin/HEAD` указывает на ветку `main`, поэтому ваш друг, клонировавший репозиторий, находится на ветке `main` репозитория `friends-rainbow`.

> **ПРИМЕЧАНИЕ**
>
> Для простоты я не буду показывать указатель `origin/HEAD` на диаграммах состояния проекта.

Для сравнения обратите внимание, что в репозитории `rainbow` вы в данный момент находитесь на ветке `feature`, а в репозитории `friend-rainbow` локальная ветка `feature` даже не существует. Вы узнаете, почему так получилось, в следующем разделе.

Клонирование репозиториев и различных типов веток

В выводе команды `git log`, полученном на шаге 5 действий 8.2, вы можете видеть, что в репозитории `friend-rainbow` нет ссылки на локальную ветку `feature`, но при этом есть ссылка на ветку удаленного отслеживания `origin/feature`. Это связано с тем, что при клонировании репозитория команда `git clone` создаст ветки удаленного отслеживания для всех веток, присутствующих в данный момент в клонируемом удаленном репозитории, но только одну локальную ветку — это будет ветка, на которую указывает `origin/HEAD`.

Для того чтобы ваш друг мог работать над веткой `feature`, он должен переключиться на нее. После этого Git создаст локальную ветку `feature` на основе того, куда указывала ветка удаленного отслеживания.

Выполните действия 8.3, чтобы смоделировать переключение вашего друга на ветку `feature`.

ДЕЙСТВИЯ 8.3

1
```
friend-rainbow $ git branch --all
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/feature
  remotes/origin/main
```

2
```
friend-rainbow $ git switch feature
branch 'feature' set up to track 'origin/feature'.
Switched to a new branch 'feature'
```

(ветка 'feature' настроена для отслеживания origin/feature.
Переключено на новую ветку 'feature')

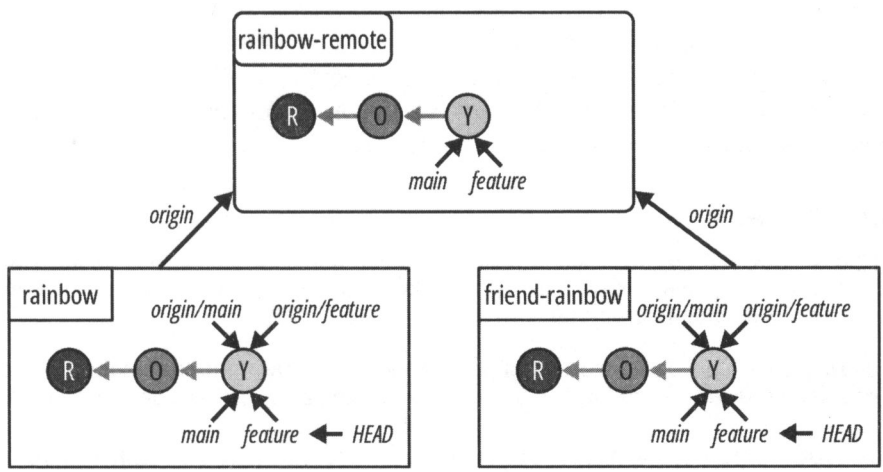
3
```
friend-rainbow $ git branch --all
* feature
  main
  remotes/origin/HEAD -> origin/main
  remotes/origin/feature
  remotes/origin/main
```

На что обратить внимание:

- на шаге 3 в выводе `git branch --all` вы увидите, что появилась новая локальная ветка `feature` и ваш друг находится на ней.

Это состояние отражено на диаграмме 8.3.

ДИАГРАММА 8.3



Проект Rainbow после того, как ваш друг переключился на ветку `feature`.

Итак, вы узнали, как создавать новые локальные ветки на основе веток удаленного отслеживания, которые вы скачали из удаленного репозитория, и как переходить на новые ветки. Далее давайте обсудим, почему в репозитории, `friend-rainbow` уже есть короткое имя origin, связанное с подключением к удаленному репозиторию.

Короткое имя *origin*

В *главе 7* вы узнали, что для связи между локальным и удаленным репозиториями необходимо указать короткое имя соединения, которое хранится в настройках локального репозитория. Для того чтобы работать с удаленным репозиторием, вам нужно было явно связать его URL с коротким именем с помощью команды `git remote add <короткое_имя> <URL>`. Это связано с тем, что вы создали репозиторий `rainbow` локально с помощью команды `git init`, и он еще не взаимодействовал с удаленным репозиторием.

Однако на шаге 3 действий 8.2 в выводе команды `git remote` вы увидели, что у вас уже есть короткое имя `origin`. Это означает, что в репозитории `friend-rainbow` уже есть URL-адрес удаленного репозитория, связанный с коротким именем `origin`. Дело в том, что локальный репозиторий вашего друга изначально не был создан локально; его напрямую клонировали из удаленного репозитория. Во время клонирования URL-адрес удаленного репозитория был связан с коротким именем в локальном репозитории, а `origin` — это короткое имя по умолчанию, которое Git связывает с удаленным репозиторием при его клонировании.

Теперь, когда вы знаете немного больше о процессе клонирования, нужно подробнее поговорить о ветках. В *главе 4* вы узнали, как создавать ветки и переключаться между ними. Далее вы узнаете, как их удалять[1].

Удаление веток

Основная причина удаления веток — стремление сохранить упорядоченную и чистую структуру проекта Git. Прежде чем удалять ветку, необходимо тщательно убедиться, что вы либо объединили ее с другой веткой, либо уверены, что вам никогда не понадобится работа, которая находится только в этой ветке.

> **ПРИМЕЧАНИЕ**
>
> Когда вы удаляете ветку с коммитами, которые не являются частью какой-либо другой ветки, вы не удаляете коммиты, являющиеся частью этой ветки. Коммиты все еще существуют в вашей истории коммитов.
>
> Однако теперь до них нелегко добраться, поскольку для них нет простой ссылки на ветку, и они не являются частью истории развития какой-либо существующей ветки.

Для того чтобы продемонстрировать, как работает удаление ветки, предположим, что вашему другу больше не нужна ветка `feature`. Для того чтобы полностью уда-

[1] К сожалению, в русском языке термин "удаленный" имеет два значения: объект с дистанционным доступом и объект, который был удален. Обычно пользователи Git так и говорят: "удаление удаленной ветки" или "удаление удаленного репозитория", и какие-либо общепринятые синонимы для обозначения процедуры удаления отсутствуют. Поэтому, пожалуйста, внимательно следите за контекстом предложений, а мы постараемся сопровождать перевод уточнениями, чтобы избежать неоднозначности. — *Прим. перев.*

лить ветку проекта, вам необходимо "стереть" ветку в удаленном репозитории, ветку удаленного отслеживания и локальную ветку. Для удаления ветки в удаленном репозитории и ветки удаленного отслеживания нужно выполнить команду `git push <короткое_имя> -d <имя_ветки>` (где `-d` означает delete — удалить). С помощью этой команды вы, по сути, отдаете удаленному репозиторию распоряжение "стереть" ветку, имя которой передали в команду.

ЗАПОМНИТЕ КОМАНДУ

git push <короткое_имя> -d <имя_ветки>

Удаляет указанную ветку удаленного репозитория и связанную с ней ветку удаленного отслеживания.

ПРИМЕЧАНИЕ

Вы также можете удалить ветку удаленного репозитория прямо на сайте хостинга, но имейте в виду, что это не приведет к удалению ветки удаленного отслеживания. Мы рассмотрим этот процесс в *разд. "Удаление ненужных веток" главы 12.*

Для того чтобы удалить *локальную* ветку, выполните команду `git branch` с опцией `-d` и передайте имя ветки, которую хотите удалить.

ЗАПОМНИТЕ КОМАНДУ

git branch -d <имя_ветки>

Удаляет локальную ветку.

Выполните действия 8.4, чтобы удалить ветку от имени вашего друга. Учтите, что вы не можете находиться в ветке, когда удаляете ее, поэтому вам (действуя от имени друга) придется переключиться с ветки `feature` на ветку `main` в репозитории `friend-rainbow`, чтобы удалить локальную ветку `feature`.

ДЕЙСТВИЯ 8.4

1.
```
friend-rainbow $ git branch --all
* feature
  main
  remotes/origin/HEAD -> origin/main
  remotes/origin/feature
  remotes/origin/main
```

2.
```
friend-rainbow $ git push origin -d feature
To github.com:gitlearningjourney/rainbow-remote.git
 - [deleted]         feature
```

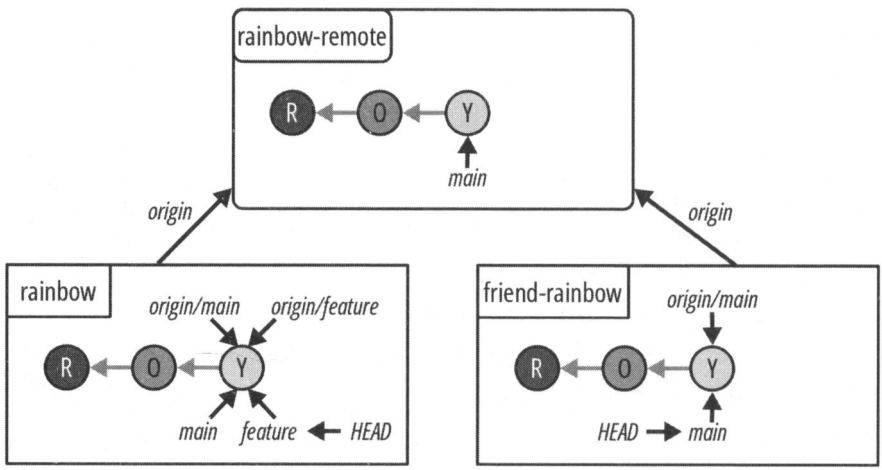
3. Перейдите в репозиторий `rainbow-remote` на своем хостинге, обновите страницу и просмотрите список веток. Ветки `feature` больше не должно быть.

```
4 friend-rainbow $ git branch --all
  * feature
    main
    remotes/origin/HEAD -> origin/main
    remotes/origin/main
5 friend-rainbow $ git switch main
  Switched to branch 'main'
  Your branch is up to date with 'origin/main'.
6 friend-rainbow $ git branch -d feature
  Deleted branch feature (was fc8139c).
7 friend-rainbow $ git branch --all
  * main
    remotes/origin/HEAD -> origin/main
    remotes/origin/main
```

На что обратить внимание:

- на шаге 2 вы удалили ветку `feature` удаленного репозитория и ветку удаленного отслеживания `origin/feature`;
- на шаге 3 вы увидите, что в удаленном репозитории `rainbow-remote` нет ветки `feature`;
- на шаге 6 вы удалили локальную ветку `feature`;
- на шаге 7 вывод команды `git branch --all` указывает на то, что в репозитории `friend-rainbow` нет локальной ветки `feature`.

ДИАГРАММА 8.4



Проект Rainbow после того, как ваш друг удалил ветку `feature` из удаленного репозитория, ветку удаленного отслеживания `origin/feature` и локальную ветку `feature`.

Диаграмма 8.4 иллюстрирует состояние локального и удаленного репозиториев после внесения этих изменений.

Как видите, в репозитории `rainbow` все еще есть ветка `feature` и ветка удаленного отслеживания `origin/feature`. Тот факт, что ваш друг удалил ветки `feature` и `origin/feature` в своем репозитории `friend-rainbow`, не влияет на ваш репозиторий `rainbow`. Эти ветки в репозитории `rainbow` вы удалите позже в данной главе, а пока давайте узнаем, как организовать совместную работу, когда ваш друг начинает вносить свой вклад в проект Rainbow.

Совместная работа и ветки Git

Прежде чем начать сотрудничать с другими участниками проекта Rainbow, имеет смысл обсудить с ними, какими должны быть некоторые соглашения относительно работы с Git. Например, в *главе 3* я упоминала, что у некоторых команд могут быть правила относительно того, что включать в сообщение коммита, а в *главе 4* вы узнали, что у некоторых команд также могут быть соглашения о том, как они называют свои ветки.

У команд также могут быть правила управления ветками. Например, могут существовать соглашения о том:

- какие ветки можно объединять друг с другом;
- когда необходимо создать ветки;
- каким должен быть процесс проверки работы, сделанной в ветке.

Распространенное правило, с которым вы можете столкнуться в проекте Git, звучит примерно так: отдельные пользователи должны работать только над своими ветками и избегать работы с чужими. Это помогает избежать конфликтов слияния (которые мы рассмотрим в *главе 10*). Мы не будем применять это правило к проекту Rainbow, чтобы упростить задачу и позволить вам сосредоточиться на изучении новых понятий, а не погружаться в детали работы с ветками. Однако имейте в виду, что вы всегда можете установить свои правила или столкнуться с другими правилами ветвления в будущих проектах Git, над которыми вы работаете. Важно обсудить с вашими коллегами соглашения о ветвлениях и хорошо понимать, что такое ветки и как работа в локальных и удаленных репозиториях с другими людьми влияет на ветки.

Далее мы рассмотрим несколько примеров сотрудничества в проекте Rainbow.

Создание коммита в локальном репозитории

В этом разделе ваш друг добавит запись о зеленом цвете в файл rainbowcolors.txt и сделает коммит в ветке `main` своего локального репозитория. Помните, что теперь вы выступаете в роли друга, а это значит, что вы будете работать в его каталоге friend-rainbow. Как было сказано ранее, с каталогом проекта друга нужно работать

в отдельном окне файловой системы и открывать файл в отдельном окне текстового редактора.

Выполните действия 8.5, чтобы смоделировать, как ваш друг делает коммит.

ПРИМЕЧАНИЕ

В выводе команды `git log` на шаге 4 в действиях 8.5 вы являетесь автором как желтого, так и зеленого коммита. Дело в том, что вы изображаете друга, вносящего вклад в проект Rainbow. Однако на самом деле, если кто-то другой сделал зеленый коммит, то он будет значиться автором коммита, и его имя и адрес электронной почты будут отображены в выходных данных. Помните об этом, читая остальные главы этой книги.

ДЕЙСТВИЯ 8.5

1. Откройте файл rainbowcolors.txt в каталоге проекта `friend-rainbow` в текстовом редакторе, добавьте запись "Зеленый — четвертый цвет радуги." в строке 4 и сохраните файл.
2. `friend-rainbow $ git add rainbowcolors.txt`
3. `friend-rainbow $ git commit -m "green"`

```
[main 6987cd2] green
 1 file changed, 2 insertions(+), 1 deletion(-)
```

4. `friend-rainbow $ git log`

```
commit 6987cd2996e245ec24ee9c5ea99874f0a01a31cd (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 11:49:03 2022 +0100

    green

commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (origin/main,
origin/HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow
```

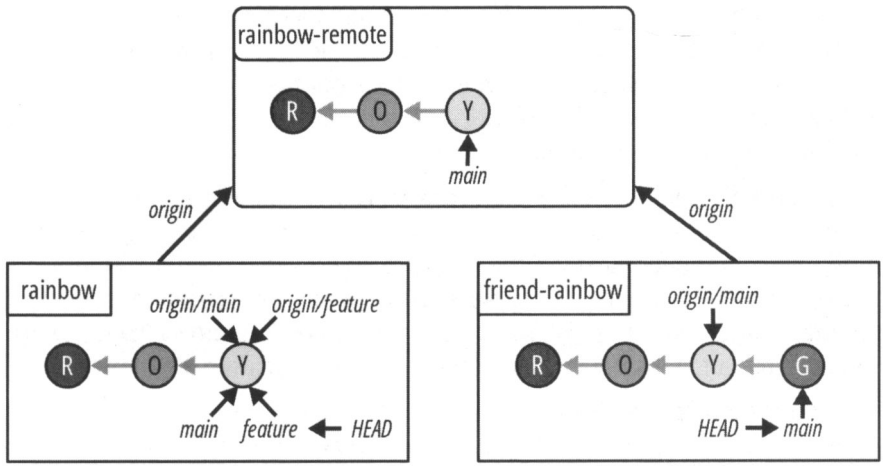
5. Перейдите в репозиторий `rainbow-remote` на сайте хостинга и обновите страницу. Зеленого коммита еще не будет.

На что обратить внимание:

- на шаге 4 выходные данные команды `git log` указывают, что в репозитории `friend-rainbow`:
  - локальная ветка `main` обновлена и теперь указывает на зеленый коммит;
  - ветка удаленного отслеживания `origin/main` по-прежнему указывает на желтый коммит.

Диаграмма 8.5 иллюстрирует изменения в репозитории `friend-rainbow`.

ДИАГРАММА 8.5



Проект Rainbow после того, как ваш друг сделал зеленый коммит в ветке `main` репозитория `friend-rainbow`.

Как видите, зеленый коммит еще не появился в репозитории `remote-rainbow`. Это связано с тем, что удаленные репозитории не обновляются автоматически; работа, выполненная в локальном репозитории, должна быть явно передана в удаленный репозиторий.

Обратите также внимание, что ветка удаленного отслеживания `origin/main` по-прежнему указывает на желтый коммит. Дело в том, что, как упоминалось в *главе 7*, ветки удаленного отслеживания представляют состояние веток в удаленном репозитории. Поскольку ваш друг еще не отправил свои изменения в удаленный репозиторий, удаленная ветка `main` не обновилась, и, следовательно, ветка удаленного отслеживания `origin/main` также не обновилась.

На данный момент у вас нет возможности сохранить зеленый коммит в репозитории `rainbow`, поскольку обновление одного локального репозитория изменениями, внесенными в другой локальный репозиторий, представляет собой двухэтапный процесс. Сначала локальный репозиторий с изменениями должен явно передать эти изменения в удаленный репозиторий. Затем локальный репозиторий, не имеющий изменений, должен явно получить и интегрировать изменения из удаленного репозитория.

Далее вы увидите, как ваш друг отправит проделанную им работу в удаленный репозиторий.

Отправка в удаленный репозиторий

В *главе 7* вы узнали, что когда вы отправляете работу из локальной ветки в удаленную, Git должен каким-то образом узнать, какую удаленную ветку вы имеете в виду. Если для локальной ветки определена восходящая ветка, вы можете использовать команду `git push` без аргументов, и Git автоматически отправит работу в эту ветку. Однако, если для локальной ветки, над которой вы работаете, не определена восходящая ветка, вам нужно будет указать, в какую удаленную ветку следует отправить работу, при вводе команды `git push`.

В прошлый раз, когда вы использовали команду `git push` в репозитории `rainbow`, вы передали короткое имя и имя ветки. Это пришлось сделать потому, что вы не определили восходящую ветку для своей локальной ветки. Напомним, что восходящая ветка — это удаленная ветка, которую отслеживает конкретная локальная ветка. Когда вы клонируете репозиторий, восходящие ветки *автоматически* настраиваются для веток, существующих в клонированном репозитории.

Для того чтобы проверить, определены ли восходящие ветки, можно выполнить команду `git branch` с опцией `-vv` (что означает very verbose — очень многословно). Эта команда также сообщает вам, находится ли локальная ветка впереди или позади восходящей ветки, если она определена.

ЗАПОМНИТЕ КОМАНДУ

git branch -vv

Выводит список локальных веток и их восходящих веток, если они есть.

Определение восходящих веток дает два основных преимущества:

1. Если вы получили (загрузили) коммиты удаленного репозитория, можете проверить, находится ли ваш репозиторий впереди удаленного репозитория или отстает от него, выполнив команду `git branch -vv` или `git status`. (Мы рассмотрим процесс получения изменений в следующем разделе этой главы.)
2. Вы можете упростить команды, используемые с удаленным репозиторием, поскольку вам не нужно указывать имя ветки и короткое имя удаленного репозитория. Например, вы можете выполнить команду `git push`, не передавая ей никаких аргументов.

Вы увидите оба этих преимущества в предстоящих примерах действий.

Поскольку ваш друг клонировал свой репозиторий, в качестве восходящей ветки для его локальной ветки `main` уже указана удаленная ветка `main`. Выполните действия 8.6, чтобы узнать, как ваш друг может проверить состояние удаленного репозитория.

ДЕЙСТВИЯ 8.6

1
```
friend-rainbow $ git branch -vv
* main 6987cd2 [origin/main: ahead 1] green
```

2
```
friend-rainbow $ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

(На ветке main
Ваша ветка опережает origin/main на 1 коммит.
(используйте "git push" для публикации локального коммита)
ожидающих коммитов нет, рабочее дерево пустое)

На что обратить внимание:

- на шаге 1 выходные данные `git branch -vv` показывают, что восходящей веткой, настроенной для локальной ветки `main`, является удаленная ветка `main` в удаленном репозитории с коротким именем `origin` и что локальная ветка `main` опережает удаленную на один коммит;
- на шаге 2 вывод `git status` тоже сообщает, что локальная ветка `main` опережает восходящую ветку `origin/main` на один коммит.

Затем выполните действия 8.7, чтобы узнать, как ваш друг может просто выполнить команду `git push` без каких-либо аргументов для обновления удаленного репозитория.

ДЕЙСТВИЯ 8.7

1
```
friend-rainbow $ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 295 bytes | 295.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local
object.
To github.com:gitlearningjourney/rainbow-remote.git
   fc8139c..6987cd2 main -> main
```

2
```
friend-rainbow $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

3 `friend-rainbow $ git log`

```
commit 6987cd2996e245ec24ee9c5ea99874f0a01a31cd (HEAD -> main,
origin/main, origin/HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 11:49:03 2022 +0100

    green

commit fc8139cbf8442cdbb5e469285abaac6de919ace6
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow
```

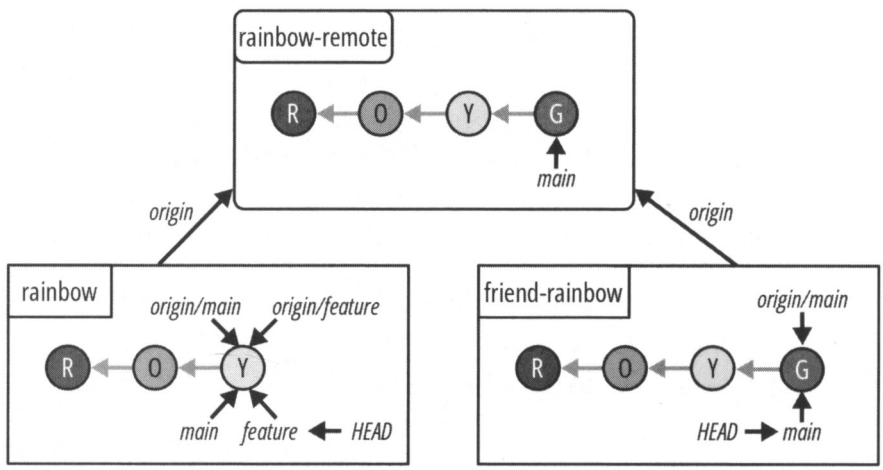
4 Перейдите в репозиторий `rainbow-remote` на вашем хостинге и обновите страницу. Там должен появиться зеленый коммит.

На что обратить внимание:

- на шаге 3 выходные данные `git log` показывают, что в репозитории `friend-rainbow` ветка удаленного отслеживания `origin/main` обновилась и теперь указывает на зеленый коммит;
- на шаге 4 вы видите, что в репозитории `rainbow-remote` удаленная ветка `main` была обновлена и теперь указывает на зеленый коммит.

Эти изменения отражены на диаграмме 8.6.

ДИАГРАММА 8.6



Проект Rainbow после того, как ваш друг отправил зеленый коммит в удаленный репозиторий.

Обратите внимание, что в репозитории `rainbow` еще нет зеленого коммита, и что локальная ветка `main` и ветка удаленного отслеживания `origin/main` в этом репозитории по-прежнему указывают на желтый коммит.

Далее вам нужно сделать так, чтобы локальная ветка `main` вашего репозитория `rainbow` синхронизировалась с веткой `main` репозитория `rainbow-remote` и репозитория `friends-rainbow`. Для этого вам нужно освоить *получение* данных (операция fetch).

Добавление изменений из удаленного репозитория

Причина, по которой локальная ветка `main` и ветка удаленного отслеживания `origin/main` в репозитории `rainbow` по-прежнему указывают на желтый коммит, заключается в том, что локальные репозитории не обновляются автоматически новыми данными из удаленных репозиториев. Точно так же, как вам необходимо предпринять явные действия для обновления удаленного репозитория изменениями в локальном репозитории, вам необходимо предпринять явные действия для обновления локальных веток и веток удаленного отслеживания изменениями из удаленного репозитория.

Добавление изменений из удаленной ветки в локальную — это двухэтапный процесс: сначала вы получаете (скачиваете) изменения из удаленного репозитория, затем интегрируете эти изменения в локальную ветку локального репозитория. Начнем с изучения первого шага.

Получение изменений из удаленного репозитория

В Git принято использовать термин *"получение"* (fetch или fetching) для обозначения процесса загрузки данных из удаленного репозитория в локальный репозиторий; для этого предназначена команда `git fetch`. Команда `git fetch` загружает все необходимые коммиты для обновления всех веток удаленного отслеживания в локальном репозитории, чтобы отразить состояние удаленных веток в указанном удаленном репозитории. Если короткое имя удаленного репозитория не указано в качестве аргумента команды `git fetch`, по умолчанию будет задействован удаленный репозиторий с коротким именем, если только для текущей ветки не определена вышестоящая ветка.

ЗАПОМНИТЕ КОМАНДУ

`git fetch <короткое_имя>`

Загружает данные из удаленного репозитория `<короткое_имя>`.

git fetch

Загружает данные из удаленного репозитория с коротким именем `origin` (по умолчанию).

Команда `git fetch` влияет только на ветки удаленного отслеживания. Она не затрагивает локальные ветки. Другими словами, она только получает (загружает) данные; на самом деле она не вносит изменения в какие-либо локальные ветки. Следовательно, когда вы получите данные из удаленного репозитория, в вашем рабочем каталоге ничего не изменится.

Вы внесете полученные изменения в следующем разделе, а пока выполните действия 8.8, чтобы получить данные из удаленного репозитория.

ДЕЙСТВИЯ 8.8

1. Перейдите в каталог проекта `rainbow` в окне командной строки.
2.
```
rainbow $ git log --all
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (HEAD ->
feature, origin/main, origin/feature, main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow

commit 7acb333f08e12020efb5c6b563b285040c9dba93
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 09:42:07 2022 +0100

   orange
```
3.
```
rainbow $ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), 275 bytes | 91.00 KiB/s, done.
From github.com:gitlearningjourney/rainbow-remote
   fc8139c..6987cd2 main -> origin/main
```
4.
```
rainbow $ git log --all
commit 6987cd2996e245ec24ee9c5ea99874f0a01a31cd (origin/main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 11:49:03 2022 +0100

    green
```

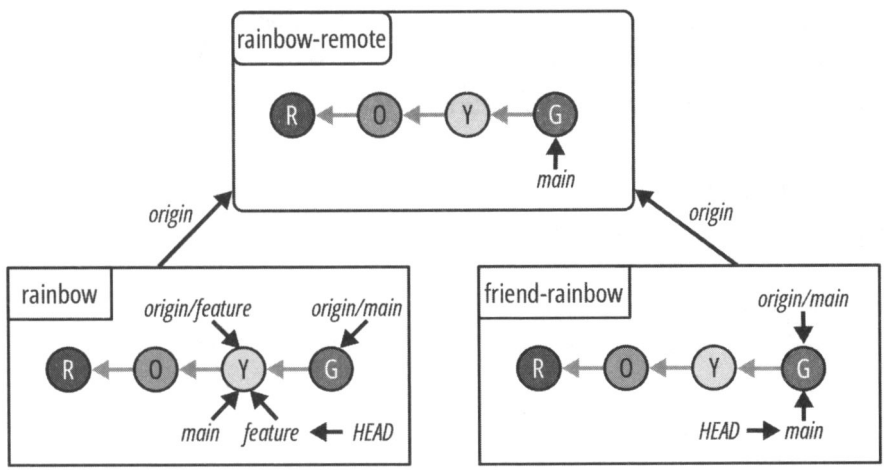
```
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (HEAD ->
feature, origin/feature, main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow
```

На что обратить внимание:

- на шаге 2 выходные данные `git log` показывают, что:
  - локальная ветка `main` и ветка удаленного отслеживания `origin/main` указывают на желтый коммит;
- на шаге 4 выходные данные `git log` показывают, что:
  - ветка удаленного отслеживания `origin/main` указывает на зеленый коммит;
  - локальная ветка `main` по-прежнему указывает на желтый коммит.

Новое состояние веток показано на диаграмме 8.7.

ДИАГРАММА 8.7



Проект Rainbow после получения данных из репозитория `rainbow-remote` в репозиторий `rainbow`.

Вы только что видели, как команда `git fetch` обновила ветки удаленного отслеживания в локальном репозитории, соответствующие удаленным веткам, существующим в удаленном репозитории. Теперь вы готовы ко второму шагу процесса: внесению изменений из удаленного репозитория в локальную ветку.

Внесение изменений в локальную ветку

После того как вы получили изменения из удаленного репозитория и обновили ветки удаленного отслеживания в локальном репозитории, необходимо обновить локальную ветку. Как вы помните, в *главе 5* было сказано, что Git предоставляет два способа внесения изменений: слияние и перебазирование. Мы рассмотрим перебазирование в *главе 11*, а пока продолжим использовать слияние.

В *разд. "Типы слияний" главы 5* вы узнали о двух видах слияний: ускоренных и трехсторонних. При ускоренном слиянии, которое вы выполнили в *главе 5*, вы объединили ветку под названием `feature` с веткой `main`. И `feature`, и `main` были локальными ветками репозитория `rainbow`. В этом разделе вы объедините ветку удаленного отслеживания `origin/main` с локальной веткой `main` репозитория `rainbow`. Типы веток, участвующих в слиянии, различны, но процесс слияния прежний. Это снова будет ускоренное слияние.

Как вы узнали из *главы 5*, при выполнении слияния вы должны находиться в ветке, в которую выполняется слияние, — в данном случае это будет ветка `main` репозитория `rainbow`. Поэтому в действиях 8.9 вы сначала переключитесь на ветку `main`, прежде чем выполнять слияние.

Для того чтобы интегрировать коммиты, которые были в ветке `main` удаленного репозитория, в вашу локальную ветку `main`, вы воспользуетесь командой `git merge`. На этот раз вы укажете имя ветки удаленного отслеживания `origin/main`.

Выполните действия 8.9 для слияния веток.

ДЕЙСТВИЯ 8.9

1.
```
rainbow $ git switch main
Switched to branch 'main'
```

2.
```
rainbow $ git merge origin/main
Updating fc8139c..6987cd2
Fast-forward
 rainbowcolors.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

3.
```
rainbow $ git log
commit 6987cd2996e245ec24ee9c5ea99874f0a01a31cd (HEAD -> main,
origin/main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 11:49:03 2022 +0100

    green
```

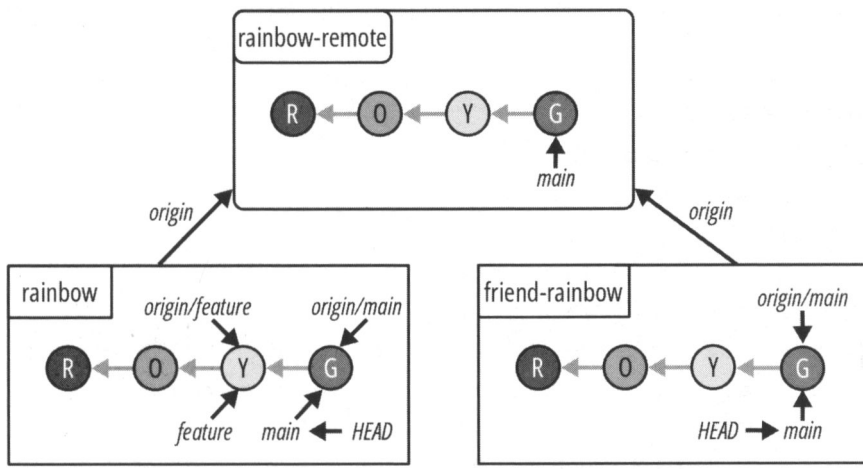
```
commit fc8139cbf8442cdbb5e469285abaac6de919ace6 (origin/feature,
feature)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 10:09:59 2022 +0100

    yellow
```

На что обратить внимание:

- на шаге 3 выходные данные команды `git log` говорят о том, что локальная ветка `main` указывает на зеленый коммит.

Текущее состояние веток показано на диаграмме 8.8.

ДИАГРАММА 8.8



Проект Rainbow после слияния ветки удаленного отслеживания `origin/main` с локальной веткой `main` в репозитории `rainbow`.

Сейчас вы увидели, как обновляется локальная ветка после завершения слияния. На диаграмме 8.8 вы также можете видеть, что в репозитории `rainbow` все еще присутствует ветка `feature`. Далее мы выполним еще несколько удалений веток, чтобы навести порядок в репозитории `rainbow`.

Удаление веток (продолжение)

В репозитории `rainbow` у вас остались локальная ветка `feature` и ветка удаленного отслеживания `origin/feature`. Для простоты в дальнейшем вы будете работать с веткой `main`, поэтому теперь вы можете удалить лишние ветки.

В *разд. "Удаление веток" ранее в этой главе* вы вводили команду `git push <короткое_имя> -d <имя_ветки>` для "стирания" удаленной ветки `feature` и ветки удаленного отслеживания `origin/feature` в репозитории `friend-rainbow` за один раз. Теперь удаленной ветки `feature` больше нет, поэтому, чтобы "стереть" ветку удаленного отслеживания `origin/feature` в репозитории `rainbow`, вы можете выполнить команду `git fetch` с опцией `-p` (которая означает prune — "срезать"). Эта команда "сотрет" все ветки удаленного отслеживания, соответствующие удаленным веткам, которые были "стерты" в удаленном репозитории.

ЗАПОМНИТЕ КОМАНДУ

git fetch -p

"Стирает" ветки удаленного отслеживания, соответствующие "стертым" удаленным веткам, и загружает данные из удаленного репозитория.

Для того чтобы удалить локальную ветку `feature`, необходимо ввести команду, которую вы уже встречали ранее в этой главе. Выполните действия 8.10, чтобы удалить ветку.

ДЕЙСТВИЯ 8.10

1
```
rainbow $ git branch --all
  feature
* main
  remotes/origin/feature
  remotes/origin/main
```
2
```
rainbow $ git fetch -p
From github.com:gitlearningjourney/rainbow-remote
 - [deleted]         (none)     -> origin/feature
```
3
```
rainbow $ git branch --all
  feature
* main
  remotes/origin/main
```
4
```
rainbow $ git branch -d feature
Deleted branch feature (was fc8139c).
```

(Ветка feature удалена)

5
```
rainbow $ git branch --all
* main
  remotes/origin/main
```

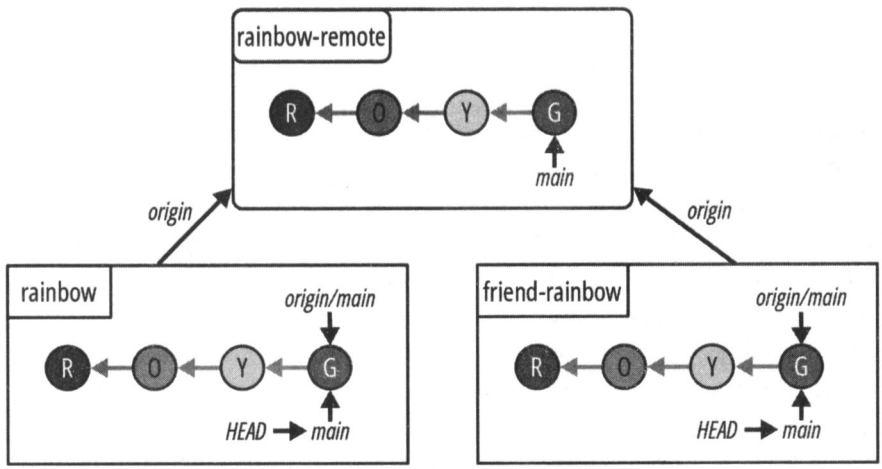
На что обратить внимание:

- на шаге 2 вы "стерли" ветку удаленного отслеживания `origin/feature`;

- на шаге 4 вы "стерли" локальную ветку `feature`;
- на шаге 5 выходные данные команды `git branch --all` указывают, что у вас больше нет локальной ветки `feature` или ветки удаленного отслеживания `origin/feature` в репозитории `rainbow`.

Текущее состояние репозиториев показано на диаграмме 8.9.

ДИАГРАММА 8.9



Проект `rainbow` после удаления локальной ветки `feature` и ветки удаленного отслеживания `origin/feature` в репозитории `rainbow`.

Краткое содержание главы

В этой главе вы начали моделировать совместную работу с другими людьми над проектом Git. Ваш воображаемый "друг" клонировал удаленный репозиторий `rainbow` и создал второй локальный репозиторий, который вы назвали `friend-rainbow`. В ходе клонирования вы заметили, что `origin` — это короткое имя по умолчанию, которое Git связывает с удаленным репозиторием при его клонировании.

После того как ваш друг сделал зеленый коммит в репозитории `friend-rainbow` и отправил его в удаленный репозиторий, вы узнали о процессе обновления локальной ветки изменениями из удаленной ветки в удаленном репозитории, который заключается в получении изменений из удаленного репозитория и последующем внесении этих изменений в локальные ветки. Вы внесли полученные изменения в локальном репозитории `rainbow`, выполнив ускоренное слияние. Далее, в *главе 9*, вы узнаете больше о другом виде слияния, упомянутом в *главе 5*, — трехстороннем слиянии.

[Глава 9]

Трехсторонние слияния

В *главе 8* вы узнали о клонировании репозиториев и о том, как происходит совместная работа с другими людьми в проекте Git. Вы рассмотрели пример, когда ваш друг вносит свой вклад в проект Rainbow, отправляет работу в удаленный репозиторий, а вы получаете изменения и выполняете слияние, чтобы синхронизировать ваши репозитории.

Все слияния, которые вы делали до этого момента в проекте Rainbow, происходили по ускоренной схеме. В этой главе вы узнаете, как выполнить трехстороннее слияние. Заодно вы также рассмотрите пример определения восходящих веток и увидите, что происходит, когда вы несколько раз редактируете файлы в рабочем каталоге между коммитами. Наконец, вы узнаете об извлечении данных из удаленного репозитория, и мы обсудим, чем извлечение данных отличается от получения.

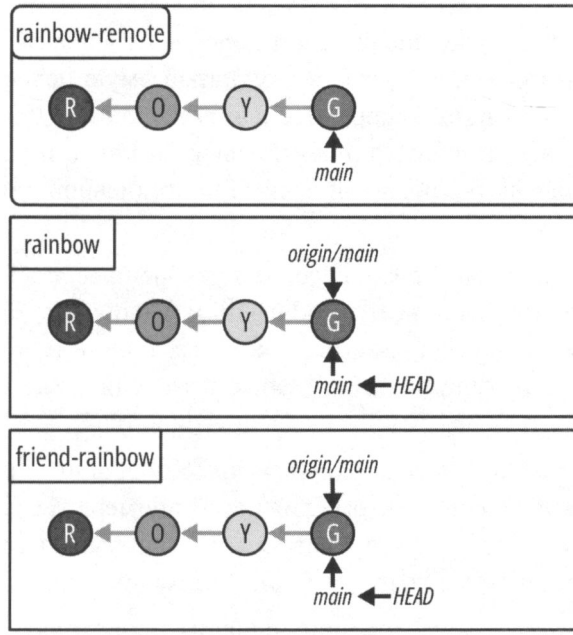
Состояние локальных и удаленных репозиториев

В начале этой главы у вас должно быть два локальных репозитория под названием `rainbow` и `friend-rainbow` и один удаленный репозиторий под названием `rainbow-remote`. Все три репозитория должны быть синхронизированы; другими словами, они должны содержать одни и те же коммиты и ветки. Я рекомендую вам продолжать использовать два отдельных окна файловой системы и окна командной строки для репозиториев `rainbow` и `friend-rainbow` при работе с примерами в этой главе, как описано в *разд. "Моделирование совместной работы" предыдущей главы*.

Диаграмма 9.1 показывает текущее состояние локальных и удаленных репозиториев в проекте Rainbow со всеми коммитами, сделанными в *главах 1–8*.

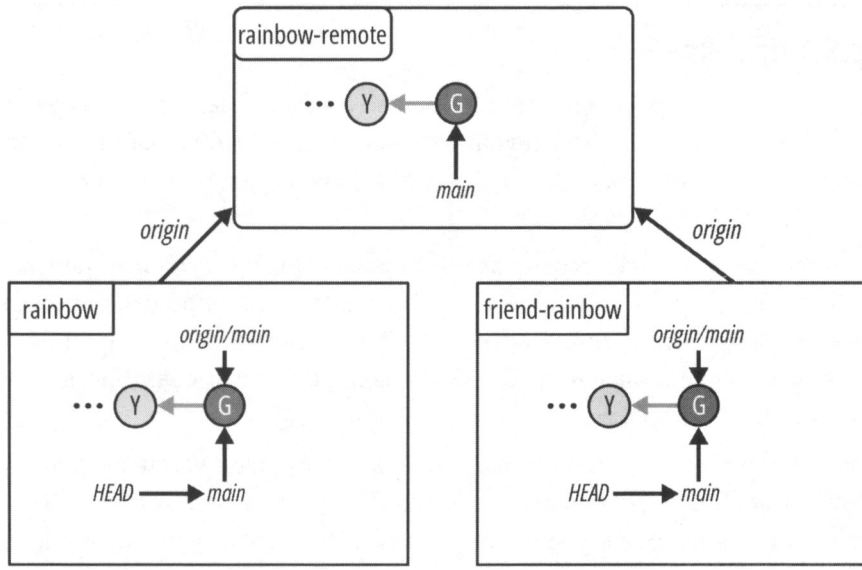
Для того чтобы сосредоточиться на коммитах, которые вы сделаете в этой главе, с этого момента я упрощу диаграммы и буду показывать только два последних коммита, которые являются частью ветки `main` во всех репозиториях: желтый коммит и зеленый коммит. Это представление показано на диаграмме 9.2.

ДИАГРАММА 9.1



Проект Rainbow в начале *главы 9* со всеми изменениями, внесенными после *главы 1*.

ДИАГРАММА 9.2



Упрощенное представление проекта Rainbow в начале *главы 9*, показывающее только два последних коммита в ветке `main` во всех репозиториях.

Почему трехсторонние слияния важны?

В *главе 5* я объясняла, что слияние означает перенос изменений, сделанных в одной ветке, называемой исходной, в другую ветку, называемую целевой. Вы узнали, что существуют два типа слияний: ускоренное и трехстороннее. До сих пор вы выполняли только ускоренные слияния, но важно также изучить и трехсторонние слияния, поскольку они являются обычной частью повседневной деятельности пользователя Git.

Трехсторонние слияния немного сложнее, чем ускоренные, поскольку они создают *коммиты слияния* и могут привести к конфликтам слияний. *Конфликты слияния* возникают при объединении двух веток, в которых в одни и те же части одного и того же файла (файлов) были внесены разные изменения, или если в одной ветке был удален файл, который редактировался в другой ветке.

В *главе 10* мы обсудим конфликты слияния более подробно и рассмотрим пример трехстороннего слияния с конфликтом слияния. В примере трехстороннего слияния в этой главе ваш друг будет редактировать файл, отличный от того, который редактируете вы, и поэтому у вас не возникнет конфликтов слияния.

Как вы узнали из *главы 5*, трехсторонние слияния происходят, когда истории развития веток, участвующих в слиянии, расходятся, — другими словами, когда невозможно достичь целевой ветки, проследив историю коммитов (родительские ссылки) исходной ветки. Давайте посмотрим, как может возникнуть такая ситуация, на примере 9.1 проекта Book.

Пример 9.1 проекта Book

Предположим, что во время работы над проектом Book мы с моим соавтором решили одновременно создать ответвление от ветки `main`, чтобы работать над разными главами. Я создаю ветку `chapter_five`, а мой соавтор — ветку `chapter_six` (рис. 9.1).

Каждый из нас работает над своей главой независимо. Мой соавтор первым заканчивает работу над веткой `chapter_six` и приступает к слиянию своей работы с веткой `main` и отправке обновленной ветки `main` в удаленный репозиторий. На рис. 9.2 в виде коммита C представлена работа, проделанная моим соавтором, и показано состояние всех репозиториев.

Когда я закончу работу над веткой `chapter_five`, которая показана как коммит D, я тоже захочу объединить свою работу с веткой `main`. Но мой соавтор сообщает мне, что он уже добавил работу в удаленную ветку `main`, поэтому сначала я должна обновить мою локальную ветку `main`, внеся в нее работу, которую мой соавтор добавил в удаленную ветку `main` (рис. 9.3).

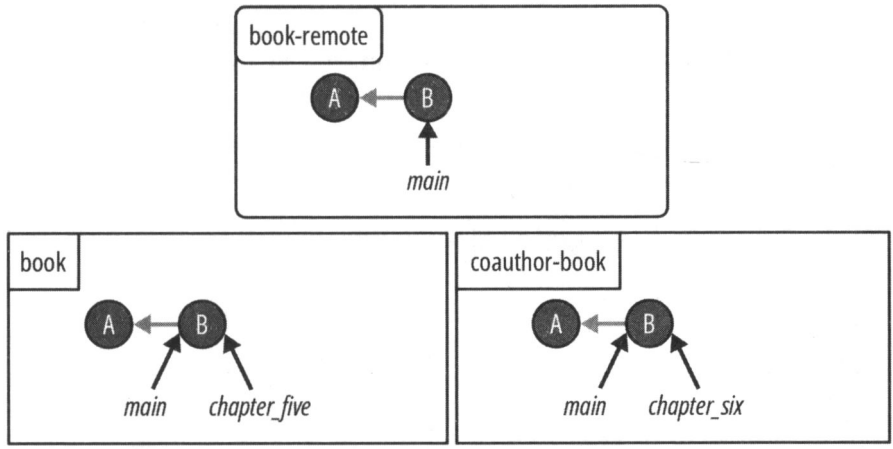


Рис. 9.1. Проект Book после того, как мы с соавтором создали ветки для работы над разными главами

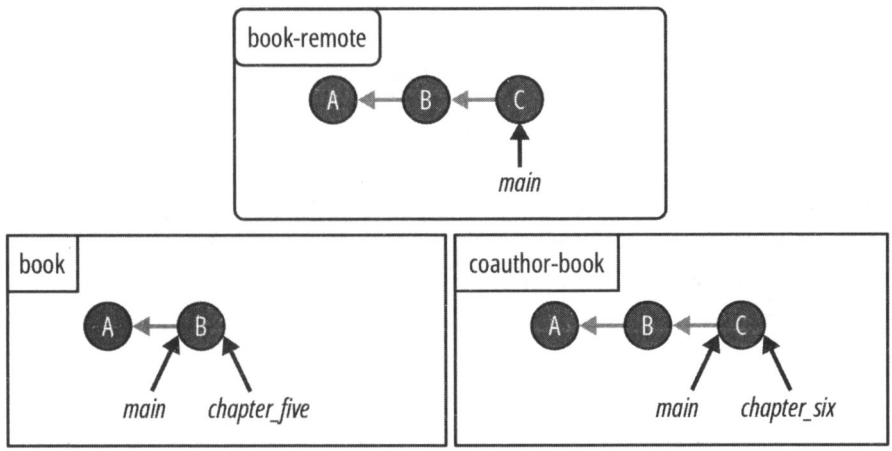


Рис. 9.2. Проект Book после того, как мой соавтор отправил свои изменения в удаленный репозиторий

На рис. 9.3 вы можете видеть, что история разработки локальной ветки `main` репозитория `book` состоит из коммитов A, B и C, а история разработки ветки `chapter_five` состоит из коммитов A, B и D. Поскольку невозможно проследить историю развития ветки `chapter_five`, чтобы добраться до ветки `main`, это означает, что истории развития этих веток разошлись. Далее у меня есть несколько вариантов.

Один из вариантов — объединить мою локальную ветку `chapter_five` с локальной веткой `main`, что будет трехсторонним слиянием, а затем отправить обновленную ветку `main` в удаленный репозиторий.

Другой вариант — выполнить слияние в удаленном репозитории с помощью функции службы хостинга, называемой *запросом на включение изменений* (pull request).

Вы узнаете о запросах на включение в *главе 12*, а пока предположим, что я решила пойти по первому пути и выполнить трехстороннее слияние в своем локальном репозитории.

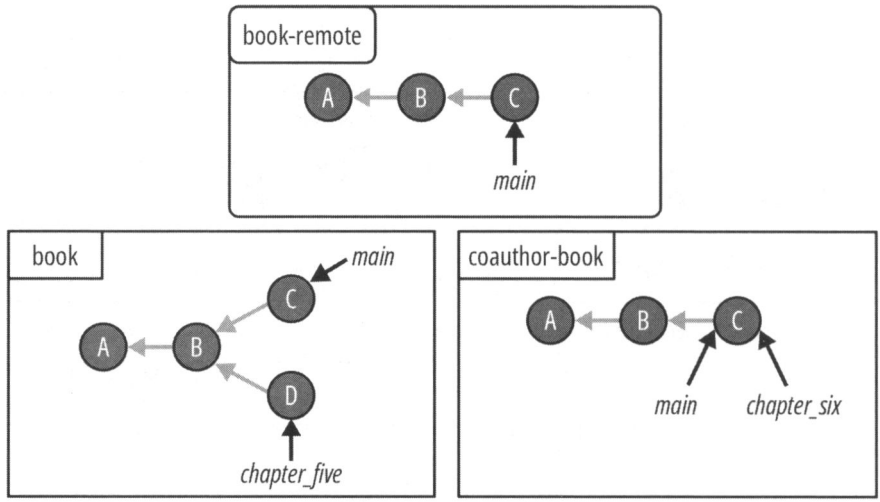


Рис. 9.3. Проект Book после того, как я внесу изменения в свою локальную ветку `chapter_five`, а также получу и интегрирую изменения из удаленной ветки `main`

На рис. 9.4 проиллюстрировано состояние репозиториев, когда я интегрирую ветку `chapter_five` в ветку `main` посредством трехстороннего слияния. Коммит слияния, полученный в результате трехстороннего слияния, обозначен буквой M.

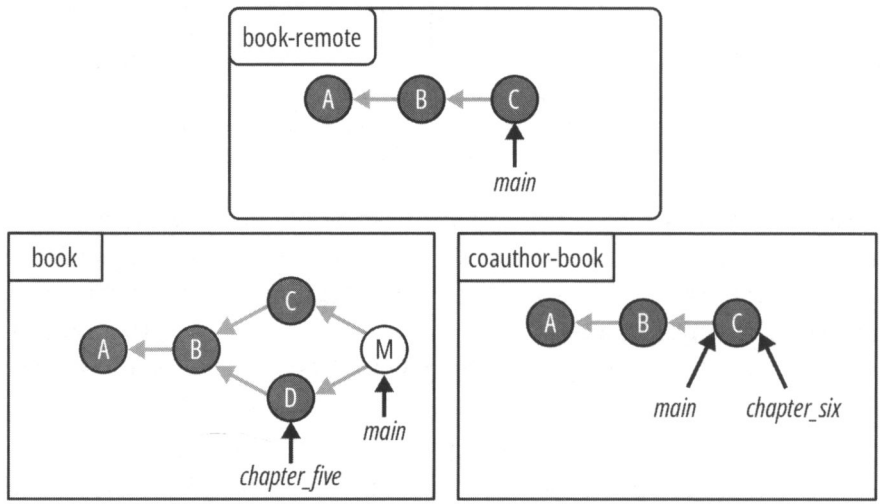


Рис. 9.4. Проект Book после объединения ветки `chapter_five` с веткой `main` посредством трехстороннего слияния

Как вы можете видеть в примере 9.1 проекта Book, трехсторонние слияния создают коммиты слияния, т. е. коммиты, которые могут иметь более одного родительского коммита.

ПРИМЕЧАНИЕ

Некоторым людям не нравятся трехсторонние слияния, поскольку они считают, что коммиты слияний усложняют историю коммитов. Для того чтобы избежать трехстороннего слияния, можете использовать процесс перебазирования, о котором вы узнаете в *главе 11*.

Далее вы создадите в проекте Rainbow ситуацию, в которой вам придется выполнить трехстороннее слияние. Попутно вы узнаете об определении восходящих веток и некоторых характеристиках измененных файлов в рабочем каталоге.

Подготовка сценария трехстороннего слияния

Сначала выполните действия 9.1, чтобы начать перечислять цвета, которые *не* являются частью радуги, в новом файле с именем othercolors.txt в каталоге проекта `rainbow`.

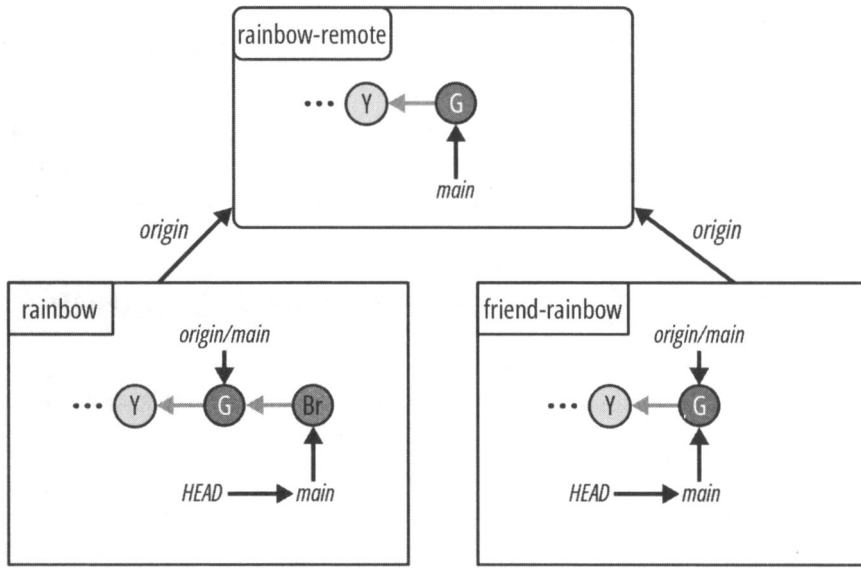
ДЕЙСТВИЯ 9.1

1. В папке проекта `rainbow` в текстовом редакторе создайте новый файл с именем othercolors.txt. Введите текст "Коричневый не является цветом радуги." в строке 1 файла и сохраните его.
2.
```
rainbow $ git add othercolors.txt
```
3.
```
rainbow $ git commit -m "brown"
[main 7f0a87a] brown
 1 file changed, 1 insertion(+)
 create mode 100644 othercolors.txt
```
4.
```
rainbow $ git log
commit 7f0a87a318e50638eec50a484bf8dfa76b76d08e (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 12:46:29 2022 +0100

    brown

commit 6987cd2996e245ec24ee9c5ea99874f0a01a31cd (origin/main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 11:49:03 2022 +0100

    green
```

На что обратить внимание:

- вы сделали коричневый коммит в репозитории `rainbow`. Результат показан на диаграмме 9.3.

ДИАГРАММА 9.3



Проект Rainbow после внесения коричневого коммита в репозиторий `rainbow`.

Далее, прежде чем отправить коммит в удаленный репозиторий, я расскажу об определении восходящих веток.

Определение восходящих веток

В *главах 7* и *8* я упоминала, что когда вы передаете работу из локальной ветки в удаленную, Git нужен способ узнать, в какую удаленную ветку вы хотите передать работу. Если для локальной ветки, над которой вы работаете, не определена восходящая ветка, вам придется явно указать, в какую удаленную ветку следует отправить изменения, при вводе команды `git push`. Если для локальной ветки определена вышестоящая ветка, вы можете использовать `git push` без аргументов, и Git автоматически отправит работу в эту ветку.

Вы также узнали, что восходящие ветки автоматически настраиваются при клонировании репозитория, но не при локальной его инициализации. Репозиторий `rainbow` был инициализирован локально, и вы еще не определили ни одной восходящей ветки.

Для того чтобы избежать указания короткого имени удаленного репозитория и ветки при каждом выполнении команды `git push` в ветке `main` вашего репозитория

rainbow, вы можете назначить вышестоящую ветку для ветки main, а затем просто вводить команду git push без аргументов.

> **ПРИМЕЧАНИЕ**
>
> После того как для локальной ветки определена восходящая ветка, вы также можете использовать без аргументов и другие команды, такие как git pull. О команде git pull вы узнаете в конце этой главы.

Для того чтобы настроить восходящую ветку, нужно выполнить команду git branch с опцией -u, которая является сокращением от --set-upstream-to. Имя удаленной ветки передают в качестве аргумента, указав короткое имя удаленного репозитория, косую черту, а затем имя удаленной ветки (например, origin/main). Затем вы увидите, как можно использовать git push без каких-либо дополнительных аргументов.

> **ЗАПОМНИТЕ КОМАНДУ**
>
> ***git branch -u <короткое_имя>/<имя_ветки>***
>
> Определяет восходящую ветку для текущей локальной ветки.

Для того чтобы проверить, определена ли восходящая ветка, пригодится команда git branch -vv, о которой вы узнали в *главе 7*.

Выполните действия 9.2, чтобы определить восходящую ветку для локальной ветки main.

> **ДЕЙСТВИЯ 9.2**
>
> 1 rainbow $ **git branch -vv**
>
> ```
> * main 7f0a87a brown
> ```
>
> 2 rainbow $ **git branch -u origin/main**
>
> ```
> branch 'main' set up to track 'origin/main'.
> ```
>
> 3 rainbow $ **git branch -vv**
>
> ```
> * main 7f0a87a [origin/main: ahead 1] brown
> ```

На что обратить внимание:

- на шаге 1 выходные данные команды git branch -vv показывают, что для локальной ветки main не назначена восходящая ветка;
- на шаге 2 выходные данные команды явно указывают, что была настроена восходящая ветка (branch 'main' set up to track 'origin/main');
- на шаге 3 выходные данные команды git branch -vv показывают, что ветка main удаленного репозитория с коротким именем origin была настроена в качестве восходящей ветки для локальной ветки main.

Итак, вы указали восходящую ветку для своей локальной ветки `main`. Теперь выполните действия 9.3, чтобы применить команду `git push` без каких-либо аргументов.

ДЕЙСТВИЯ 9.3

1.
```
rainbow $ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 310 bytes | 310.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:gitlearningjourney/rainbow-remote.git
6987cd2..7f0a87a main -> main
```

2.
```
rainbow $ git log
commit 7f0a87a318e50638eec50a484bf8dfa76b76d08e (HEAD -> main,
origin/main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 12:46:29 2022 +0100

    brown

commit 6987cd2996e245ec24ee9c5ea99874f0a01a31cd
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 11:49:03 2022 +0100

    green
```

3. Перейдите в репозиторий `rainbow-remote` на своем хостинге и обновите страницу. Вы должны увидеть коричневый коммит.

На что обратить внимание:

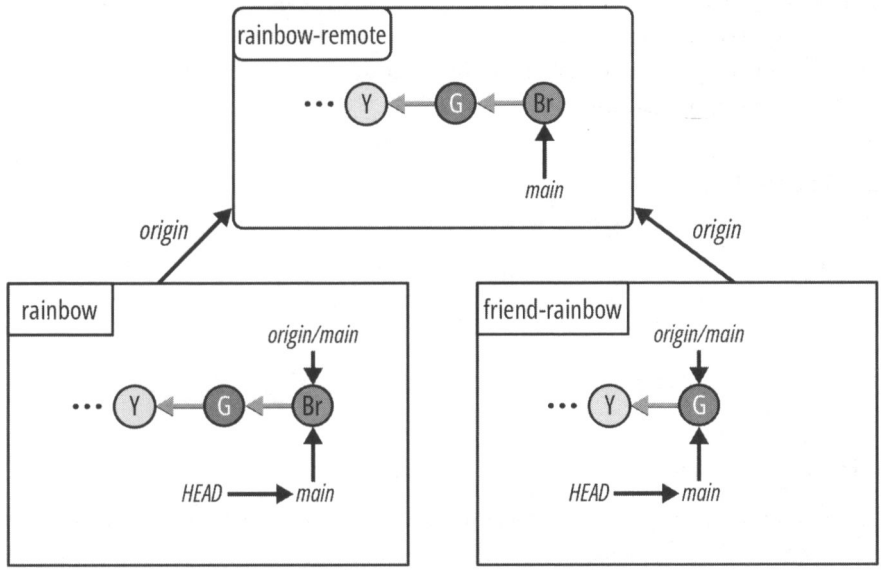
- вы отправили свою локальную ветку `main` в удаленный репозиторий.

Текущее состояние после этой операции показано на диаграмме 9.4.

Вы только что узнали, как определять восходящие ветки в локальном репозитории, сделали коричневый коммит в репозитории `rainbow` и отправили его в удаленный репозиторий. Для того чтобы возникла ситуация, когда вам придется провести трехстороннее слияние, две ветки должны иметь разные истории развития.

Далее ваш друг продолжит работу над локальной веткой `main` в своем локальном репозитории, не получая изменений, которые вы отправили в удаленную ветку `main`, вследствие чего локальная ветка `main` в репозитории `friend-rainbow` и ветка `main` в репозитории `rainbow-remote` разойдутся.

ДИАГРАММА 9.4



Проект Rainbow после отправки локальной ветки `main` в репозиторий `rainbow-remote`.

Пока ваш друг работает на локальной ветке `main`, вы также узнаете о некоторых характеристиках измененных файлов в рабочем каталоге и о том, что происходит, когда вы несколько раз редактируете файл между коммитами.

Редактирование одного файла несколько раз между коммитами

До сих пор вы редактировали файлы только один раз и добавляли их в промежуточную область. Однако важно понимать, что если вы добавите файл в промежуточную область, а затем внесете в него еще одно изменение, Git интерпретирует это как новую версию файла и пометит файл как измененный. Если вы потом захотите, чтобы последняя версия файла была включена в ваш следующий коммит, вам придется снова добавить обновленную версию файла в промежуточную область.

Для того чтобы увидеть это в действии, представьте, что ваш друг решил добавить голубой цвет в список цветов в rainbowcolors.txt и поместить отредактированный файл в свою промежуточную область, но допустил опечатку, написав: "Глбой — это пятый цвет радуги". Заметив свою оплошность, он снова отредактирует файл, чтобы исправить опечатку, и вы увидите, что файл нужно будет добавить в промежуточную область во второй раз, чтобы последние изменения были включены в следующий коммит.

В будущих диаграммах состояния проекта я буду использовать диаграмму Git, которая впервые предстала перед вами в *главе 2*, но увеличу масштаб репозитория `friend-rainbow`, чтобы показать характерные детали. Сначала выполните действия 9.4, чтобы проверить состояние рабочего каталога и промежуточной области `friend-rainbow` на данный момент.

ДЕЙСТВИЯ 9.4

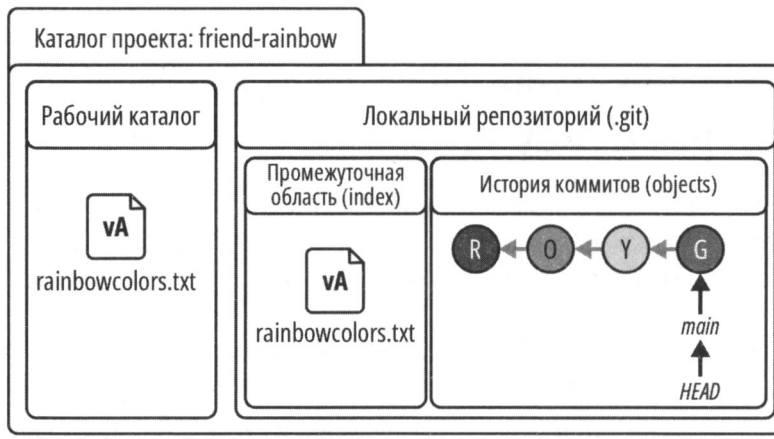
1
```
friend-rainbow $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

На что обратить внимание:

- в выводе команды `git status` сказано, что выполнять коммит не нужно, рабочее дерево очищено, что указывает на совпадение рабочего каталога и промежуточной области. В рабочем каталоге нет измененных файлов, а в промежуточной области нет новых файлов, ожидающих коммита.

Это состояние отображено на диаграмме 9.5.

ДИАГРАММА 9.5



Текущее состояние каталога проекта `friend-rainbow`.

На что обратить внимание:

- версия файла rainbowcolors.txt в рабочем каталоге и промежуточной области упоминает красный, оранжевый, желтый и зеленый цвета. Мы обозначим ее как версию A (vA).

Далее, выполняя действия 9.5, ваш друг отредактирует файл rainbowcolors.txt и добавит голубой цвет.

ДЕЙСТВИЯ 9.5

1. Откройте файл rainbowcolors.txt в каталоге проекта `friend-rainbow` в окне текстового редактора, добавьте текст "Глбой — это пятый цвет радуги." в строке 5 и сохраните файл. Это предложение содержит преднамеренную опечатку в учебных целях.

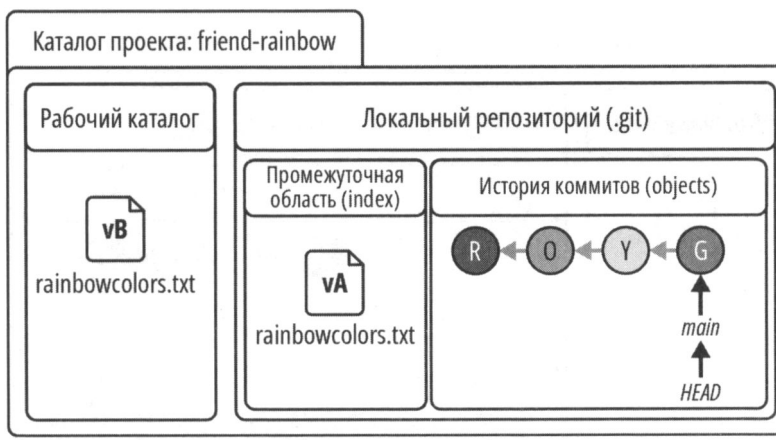
2.
```
friend-rainbow $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be  committed)
  (use "git restore <file>..." to discard changes in working
directory)
    modified: rainbowcolors.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

На что обратить внимание:

- на шаге 2 вывод `git status` указывает, что теперь в рабочем каталоге есть один измененный файл, который еще не подготовлен для коммита.

Это наблюдение отражено на диаграмме 9.6.

ДИАГРАММА 9.6



Каталог проекта `friend-rainbow` после того, как ваш друг отредактировал файл rainbowcolors.txt в рабочем каталоге.

На что обратить внимание:

- версия файла rainbowcolors.txt в промежуточной области по-прежнему vA; она не изменилась;
- версия файла rainbowcolors.txt в рабочем каталоге изменилась: в файле упоминаются красный, оранжевый, желтый, зеленый и голубой цвета. Мы представляем ее как версию B (vB).

Теперь выполните действия 9.6 и посмотрите, что произойдет, когда ваш друг добавит файл в промежуточную область.

ДЕЙСТВИЯ 9.6

1 `friend-rainbow $ git add rainbowcolors.txt`

2 `friend-rainbow $ git status`

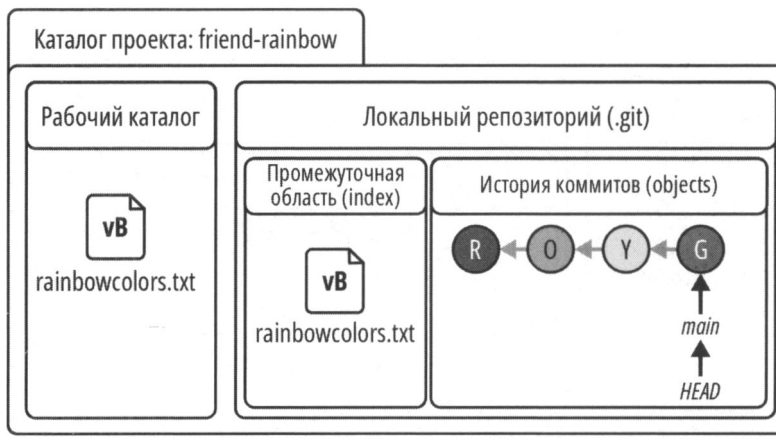
```
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified: rainbowcolors.txt
```

На что обратить внимание:

- на шаге 2 выходные данные команды `git status` указывают, что теперь в рабочем каталоге есть один измененный файл, который также подготовлен для коммита.

Это состояние показано на диаграмме 9.7.

ДИАГРАММА 9.7



Каталог проекта `friend-rainbow` после того, как ваш друг добавил измененный файл rainbowcolors.txt в промежуточную область.

На что обратить внимание:

- версия файла rainbowcolors.txt в промежуточной области изменилась с vA на vB;
- версия файла rainbowcolors.txt в промежуточной области такая же, как и в рабочем каталоге.

Ваш друг добавил обновленную версию файла rainbowcolors.txt в промежуточную область, и она готова к коммиту. Теперь выполните действия 9.7, чтобы увидеть, что произойдет, когда ваш друг вернется, чтобы снова отредактировать файл и исправить опечатку, изменив слово "Глбой" на "Голубой".

ДЕЙСТВИЯ 9.7

1. В папке проекта `friend-rainbow` в текстовом редакторе исправьте опечатку в строке 5 файла rainbowcolors.txt, чтобы предложение гласило: "Голубой — пятый цвет радуги.". Сохраните файл.

2.
```
friend-rainbow $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
   modified: rainbowcolors.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
directory)
   modified: rainbowcolors.txt
```

На что обратить внимание:

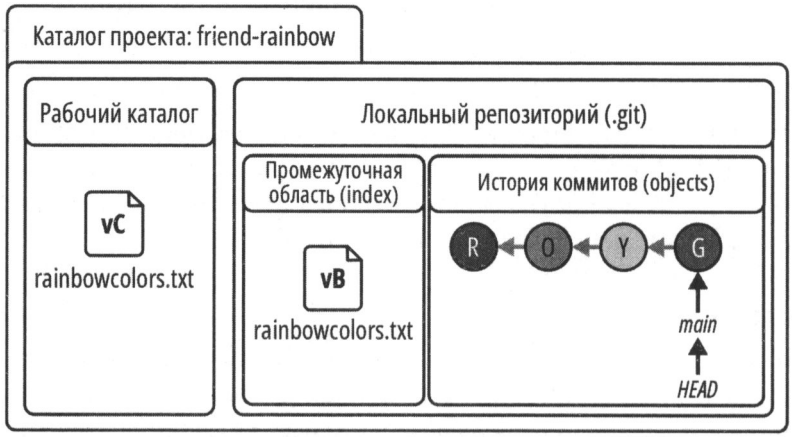
- в выводе команды `git status` на шаге 2 одна версия файла rainbowcolors.txt указана как измененный файл, подготовленный для коммита, а другая версия файла rainbowcolors.txt указана как измененный файл, не подготовленный для коммита.

Эта ситуация отражена на диаграмме 9.8.

На что обратить внимание:

- версия файла rainbowcolors.txt в промежуточной области — vB. Это версия файла с опечаткой;
- версия файла rainbowcolors.txt в рабочем каталоге изменилась с vB на версию C (vC). Файл vC — это версия без опечатки.

ДИАГРАММА 9.8



Каталог проекта `friend-rainbow` после того, как ваш друг отредактирует файл rainbowcolors.txt во второй раз.

Из этих наблюдений видно, что если файл с определенным именем подготовлен для коммита, это *не означает*, что он автоматически обновляется с учетом любых других изменений, которые вы вносите в него. Вам необходимо явно добавлять каждую обновленную версию файла в промежуточную область, чтобы включить последние изменения в следующий коммит. Выполните действия 9.8, чтобы увидеть это на практике.

ДЕЙСТВИЯ 9.8

1. `friend-rainbow $ git add rainbowcolors.txt`
2. `friend-rainbow $ git status`

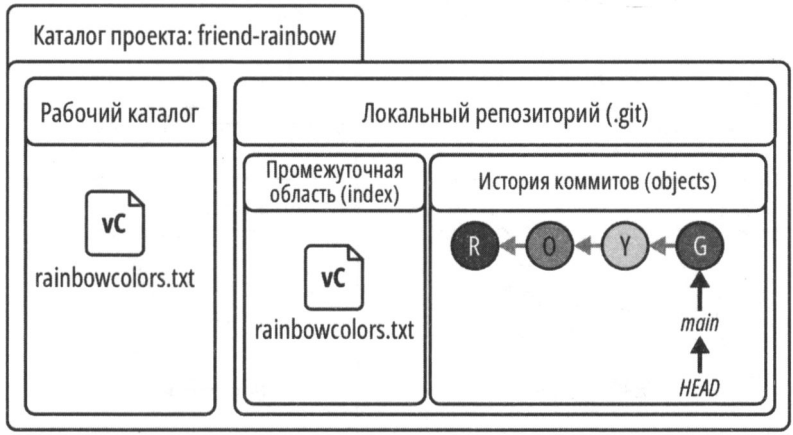
```
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
   modified: rainbowcolors.txt
```

На что обратить внимание:

- в выводе команды `git status` на шаге 2 файл rainbowcolors.txt указан как измененный файл, подготовленный для коммита. Больше нет измененных файлов, которые можно добавить в промежуточную область.

Обновленное состояние показано на диаграмме 9.9.

ДИАГРАММА 9.9



Каталог проекта `friend-rainbow` после того, как ваш друг снова добавил файл rainbowcolors.txt в промежуточную область.

На что обратить внимание:

- версия файла rainbowcolors.txt в промежуточной области изменилась с vB на vC; следовательно, в промежуточную область добавлена последняя версия файла rainbowcolors.txt.

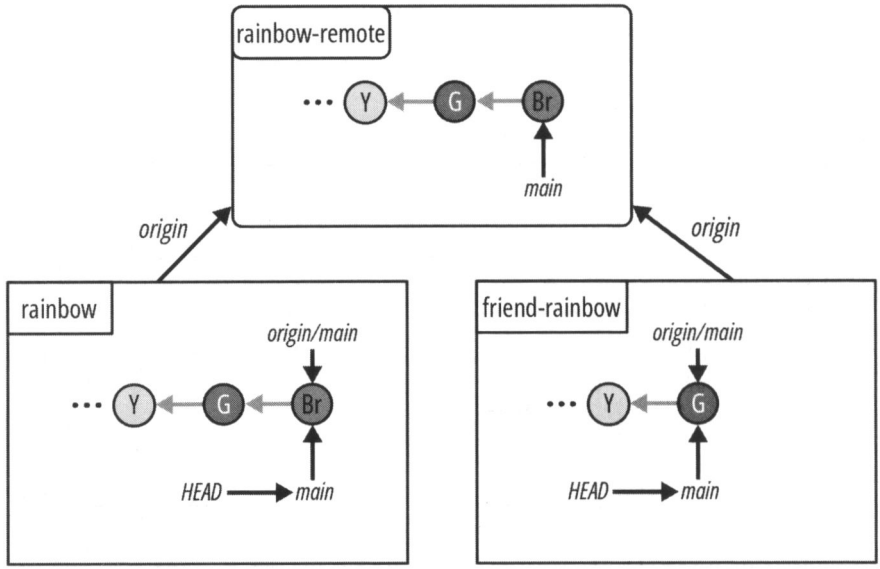
Затем ваш друг завершит выполнение синего коммита и попытается отправить свою работу в удаленный репозиторий. Однако, поскольку его локальная ветка `main` не синхронизирована с удаленной веткой `main`, он столкнется с ошибкой. Вы увидите это в следующем разделе.

Совместная работа с коллегами над разными файлами

На диаграмме 9.10 вы можете видеть, что ваш друг не получил коричневый коммит, который вы сделали в ветке `main` репозитория `rainbow`, и не отправил его в удаленный репозиторий.

Локальная ветка `main` в репозитории `friend-rainbow` не синхронизирована с удаленной веткой `main`. Ваш друг готов сделать голубой коммит и поделиться своей работой, но когда он попытается отправить свою работу в удаленный репозиторий, он получит сообщение об ошибке. Давайте посмотрим, как это произойдет, выполнив действия 9.9.

ДИАГРАММА 9.10



Текущее состояние локального и удаленного репозиториев.

ДЕЙСТВИЯ 9.9

1
```
friend-rainbow $ git commit -m "blue"
[main 342bbfc] blue
 1 file changed, 2 insertions(+), 1 deletion(-)
```

2
```
friend-rainbow $ git push
To github.com:gitlearningjourney/rainbow-remote.git
 ! [rejected] main -> main (fetch first)
error: failed to push some refs to 'github.com:gitlearningjou
rney/rainbow-remote.git'
```

(Ошибка: не удалось отправить обновления в 'github.com:gitlearningjourney/rainbow-remote.git')

```
hint: Updates were rejected because the remote contains work that
you do
hint: not have locally. This is usually caused by
hint: another epository pushing
hint: to the same ref. You may want
hint: to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help'
for details.
```

(Подсказка: обновления отвергнуты, потому что удаленный репозиторий содержит работу, которой нет в локальном репозитории. Обычно это случается, когда в то же место отправлено обновление из другого репозитория. Возможно, вам нужно сначала получить удаленные изменения (т. е. 'git pull ...'), а затем попробовать повторить отправку. Для получения инструкций воспользуйтесь командой 'git push --help'.)

3 `friend-rainbow $ git log`

```
commit 342bbfc96bb03053f23ea7f7564ca207c58ceab2 (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 13:00:56 2022 +0100

    blue

commit 6987cd2996e245ec24ee9c5ea99874f0a01a31cd (origin/main,
origin/HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 11:49:03 2022 +0100

    green
```

На что обратить внимание:

- на шаге 2 выходные данные команды `git push` показывают, что ваш друг получил сообщение об ошибке. Он не может отправить свои изменения в удаленный репозиторий.

Истории разработки локальной и удаленной версий ветки `main` разошлись, и Git не может объединять изменения с помощью простого ускоренного слияния. Добавление голубого коммита и расхождение историй разработки проиллюстрированы на диаграмме 9.11.

На что обратить внимание:

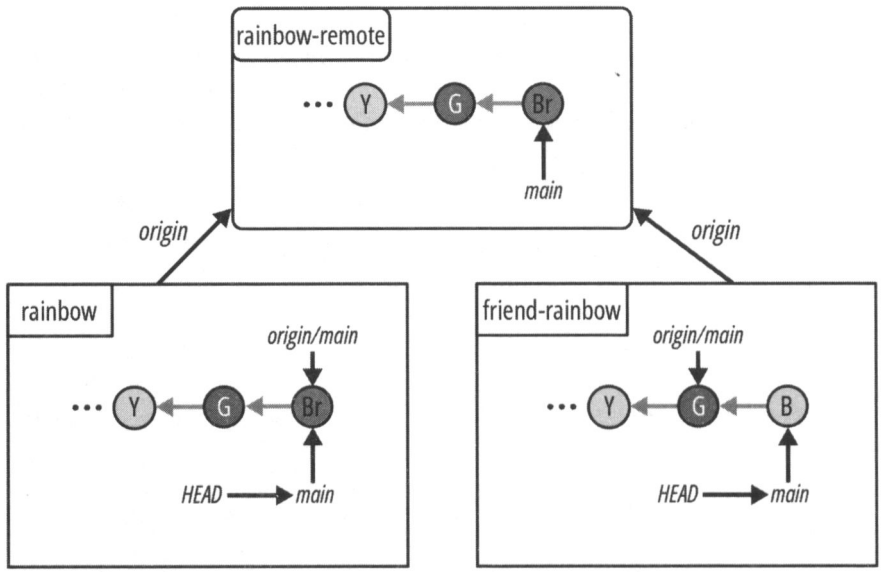
- в репозитории `friend-rainbow` локальная ветка `main` указывает на голубой коммит;
- в репозитории `rainbow-remote` ветка `main` указывает на коричневый коммит.

Давайте подробнее рассмотрим сообщение об ошибке, которое получил ваш друг:

Ошибка: не удалось отправить обновления в 'github.com:gitlearningjourney/rainbow-remote.git'.

Подсказка: обновления отвергнуты, потому что удаленный репозиторий содержит работу, которой нет в локальном репозитории. Обычно это случается, когда в то же место отправлено обновление из другого репозитория. Возможно, вам нужно сначала получить удаленные изменения (т. е. 'git pull ...'), а затем попробовать повторить отправку. Для получения инструкций воспользуйтесь командой 'git push --help'.

ДИАГРАММА 9.11



Проект Rainbow после того, как ваш друг сделал голубой коммит в репозитории `friend-rainbow`.

Git сообщает вашему другу, что в удаленной ветке `main` есть коммиты, которые он еще не извлек. Он сообщает вашему другу, что ему необходимо извлечь удаленную работу и интегрировать ее в свою локальную ветку `main`, прежде чем он сможет сделать отправку в удаленный репозиторий. Обратите внимание, что извлечение (pull) похоже на получение (fetch), но с некоторыми отличиями; мы обсудим их позже в этой главе. На данный момент, чтобы интегрировать удаленные изменения, ваш друг собирается выполнить трехстороннее слияние.

Трехстороннее слияние на практике

В *главе 8* вы узнали, что включение изменений из удаленного репозитория представляет собой двухэтапный процесс:

1. Сначала вы получаете (fetch) изменения из удаленного репозитория.
2. Затем вы интегрируете изменения в ветку локального репозитория.

Ваш друг собирается выполнить те же шаги прямо сейчас, за исключением того, что на шаге 2 он в конечном итоге выполнит трехстороннее слияние вместо быстрого слияния, поскольку истории развития двух веток, участвующих в слиянии, разошлись.

При выполнении трехстороннего слияния Git создаст коммит слияния. Для этого он войдет в текстовый редактор командной строки. Вам необходимо научиться работать с этим текстовым редактором.

Представляем Vim — текстовый редактор командной строки

Для написания сообщений о коммите Git по умолчанию использует текстовый редактор в окне командной строки под названием Vim. Если выполняется коммит (как обычный, так коммит слияния) и сообщение коммита не указано, Git запустит Vim в окне командной строки. В этой книге вы еще не встречались с Vim, поскольку каждый раз, когда вы делали коммит с помощью команды `git commit` в проекте Rainbow, вы передавали опцию `-m` и явно указывали сообщение коммита.

> **ПРИМЕЧАНИЕ**
>
> Git позволяет изменить используемый по умолчанию текстовый редактор командной строки с Vim на другой текстовый редактор. Например, вы можете настроить запуск того же текстового редактора, который вы используете для редактирования файлов в своем проекте.

Vim может показаться пугающим, потому что редактирование текста в окне командной строки выглядит непривычно, и если вы не знаете некоторых основ Vim, пользоваться им будет затруднительно. Давайте освоим основные навыки работы с Vim.

Когда Git запускает Vim в командной строке, у вас есть выбор: ввести текст, отредактировать текст или утвердить и сохранить его. В случае трехстороннего слияния Git создаст черновик сообщения о коммите по умолчанию для коммита слияния, а Vim представит его вам (рис. 9.5).

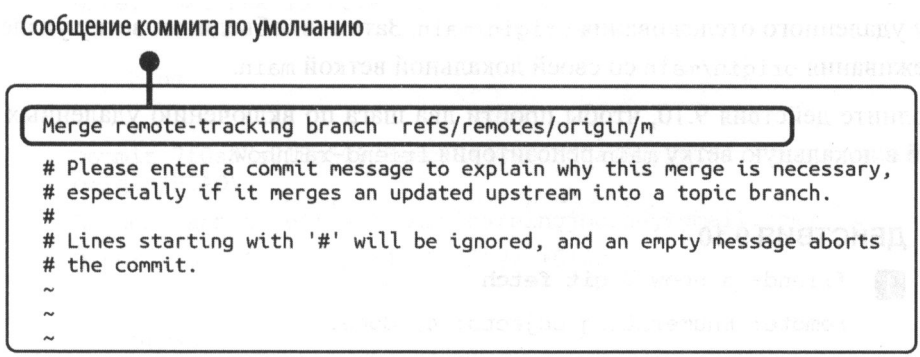


Рис. 9.5. Пример сообщения о коммите по умолчанию, составленного Git и представленного в текстовом редакторе Vim в командной строке

В качестве ознакомительного примера мы просто рассмотрим, как принять и сохранить сообщение коммита по умолчанию. Для этого вам необходимо нажать клавишу <Esc>, ввести двоеточие (:), затем последовательно нажать клавиши <W>, <Q> и <Enter> по порядку, как показано на рис. 9.6. В Vim буква `w` означает write (запись), т. е. сохранение файла, а буква `q` означает quit, т. е. выход. (Если что-то пойдет не так, вы можете нажать клавишу <Esc>, чтобы повторить попытку.)

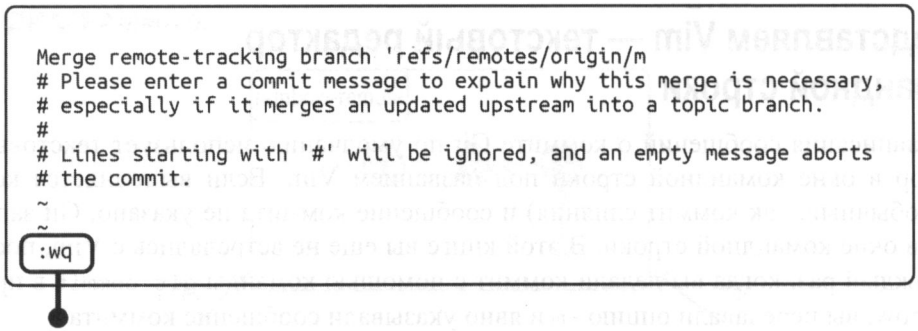


Рис. 9.6. Команда для сохранения файла и выхода из Vim

Эта команда сохранит сообщение о коммите и закроет текстовый редактор командной строки Vim. Эти инструкции будут повторяться во всех последующих действиях, требующих выхода из Vim.

К этому моменту вы получили первые навыки работы с Vim, а ваш друг готов выполнить два шага, чтобы включить удаленные изменения в свою локальную ветку.

Выполнение трехстороннего слияния

Прочитав сообщение об ошибке Git, ваш друг теперь собирается получить изменения из удаленной ветки `main` в свой локальный репозиторий. Это действие обновит ветку удаленного отслеживания `origin/main`. Затем он объединит ветку удаленного отслеживания `origin/main` со своей локальной веткой `main`.

Выполните действия 9.10, чтобы пройти два шага по включению удаленных изменений в локальную ветку `main` репозитория `friend-rainbow`.

ДЕЙСТВИЯ 9.10

1
```
friend-rainbow $ git fetch
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 290 bytes | 145.00 KiB/s, done.
From github.com:gitlearningjourney/rainbow-remote
   6987cd2..7f0a87a main -> origin/main
```

2
```
friend-rainbow $ git merge origin/main
```

3 После выполнения команды `git merge` будет запущен редактор Vim. Сообщением о коммите по умолчанию, которое Git приготовит для вас, будет либо `Merge remote-tracking branch 'refs/remotes/origin/main'` (Объединить ветку удаленного отслеживания 'refs/remotes/origin/main'), либо `Merge remote-tracking branch 'origin/main'` (Объединить ветку удаленного отслеживания 'origin/main'). Для того чтобы принять сообщение по умолчанию в редакторе Vim и выйти из редактора, необходимо нажать клавишу <Esc>, ввести **:wq**, а затем нажать клавишу <Enter>. Вы увидите следующий результат:

```
Merge made by the 'ort' strategy.
 othercolors.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 othercolors.txt
friend-rainbow $ git log
commit 225839938563c7458af81daca7beb782dfcbfb27 (HEAD -> main)
Merge: 342bbfc 7f0a87a
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 13:05:46 2022 +0100
  Merge remote-tracking branch 'refs/remotes/origin/main'
commit 342bbfc96bb03053f23ea7f7564ca207c58ceab2
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:  Sat Feb 19 13:00:56 2022 +0100

    blue

commit 7f0a87a318e50638eec50a484bf8dfa76b76d08e (origin/main,
origin/HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 12:46:29 2022 +0100

    brown
```

На что обратить внимание:

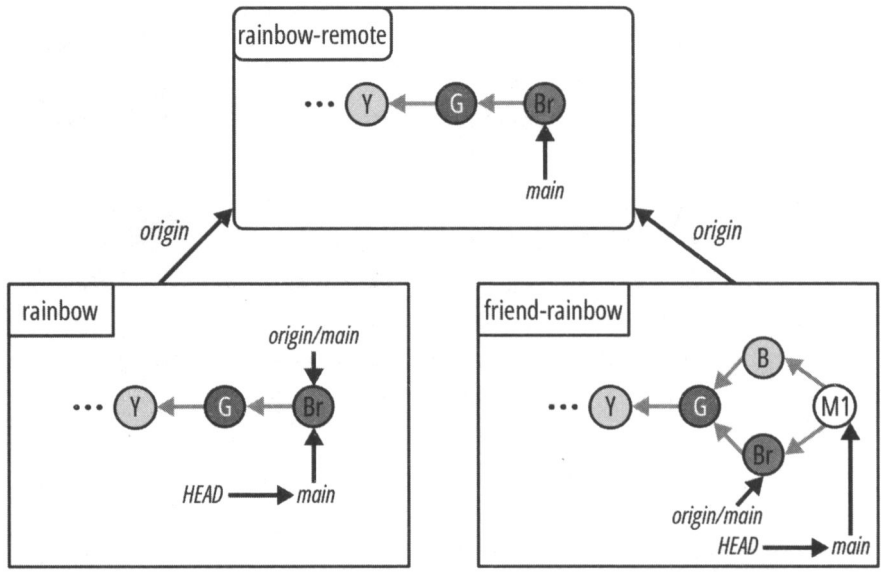
- на шаге 3 в выводе `git merge` указано, что слияние выполнено по стратегии "ort". Это говорит о том, что произошло трехстороннее, а не ускоренное слияние (если у вас более старая версия Git, в выводе может быть указано, что слияние было выполнено по "рекурсивной" стратегии; это тоже верно);
- Git подготовил для вас сообщение о коммите по умолчанию: либо `Merge remote-tracking branch 'refs/remotes/origin/main'` (Объединить ветку уда-

ленного отслеживания 'refs/remotes/origin/main'), либо `Merge remote-tracking branch 'origin/main'` (Объединить ветку удаленного отслеживания 'origin/main'). Оба варианта верны;

- на шаге 4 выходные данные команды `git log` показывают коммит слияния `2258399` и в поле `Merge` (слияние) перечисляются два родительских коммита: `342bbfc` (голубой коммит) и `7f0a87a` (коричневый коммит). Хеши ваших коммитов будут отличаться от тех, что представлены в этой книге, поскольку хеши коммитов уникальны.

На диаграмме 9.12 показано состояние проекта Rainbow после выполнения действий 9.10.

ДИАГРАММА 9.12



Проект Rainbow после того, как ваш друг получил коричневый коммит и выполнил трехстороннее слияние.

На что обратить внимание:

- В репозитории `friend-rainbow` коммит слияния обозначен как M1, и показано, как он связывает две истории разработки.

Коммит слияния M1 имеет два родительских коммита. Напомню, что в *главе 4* вы использовали команду `git cat-file -p <хеш_коммита>` для просмотра родителей коммита. В следующем наборе действий ваш друг воспользуется этой командой, передав ей хеш коммита слияния M1, чтобы увидеть его родительские коммиты (представленные соответствующими хешами коммитов). Для того чтобы получить хеш коммита слияния M1, вы можете взять выходные данные команды `git log`. Помните, что вы должны использовать *свой* хеш коммита слияния M1, а не хеш коммита из этой книги (поскольку хеши коммита уникальны).

После этого ваш друг отправит свои изменения в удаленный репозиторий, чтобы убедиться, что он обновлен. Выполните действия 9.11, чтобы проделать все вышесказанное.

ДЕЙСТВИЯ 9.11

1. Получите хеш для коммита слияния M1 (вы можете скопировать его из вывода `git log` в действиях 9.10). Вы должны передать этот хеш коммита в качестве аргумента команде `git cat-file -p` на шаге 2 данного руководства. Вы можете скопировать и вставить весь хеш коммита или просто ввести первые семь символов, как показано ниже.

2.
```
friend-rainbow $ git cat-file -p 2258399
tree 45330906e6041a0cd07849617a25443a9a5b08bd
parent 342bbfc96bb03053f23ea7f7564ca207c58ceab2
parent 7f0a87a318e50638eec50a484bf8dfa76b76d08e
author annaskoulikari <gitlearningjourney@gmail.com> 1645272346
+0100
committer annaskoulikari <gitlearningjourney@gmail.com>
1645272346 +0100

Merge remote-tracking branch 'refs/remotes/origin/main'
```

3.
```
friend-rainbow $ git push
Enumerating objects: 9, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 586 bytes | 586.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local
object.
To github.com:gitlearningjourney/rainbow-remote.git
   7f0a87a..2258399 main -> main
```

4.
```
friend-rainbow $ git log
commit 225839938563c7458af81daca7beb782dfcbfb27 (HEAD -> main,
origin/main, origin/HEAD)
Merge: 342bbfc 7f0a87a
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 13:05:46 2022 +0100

    Merge remote-tracking branch 'refs/remotes/origin/main'
```

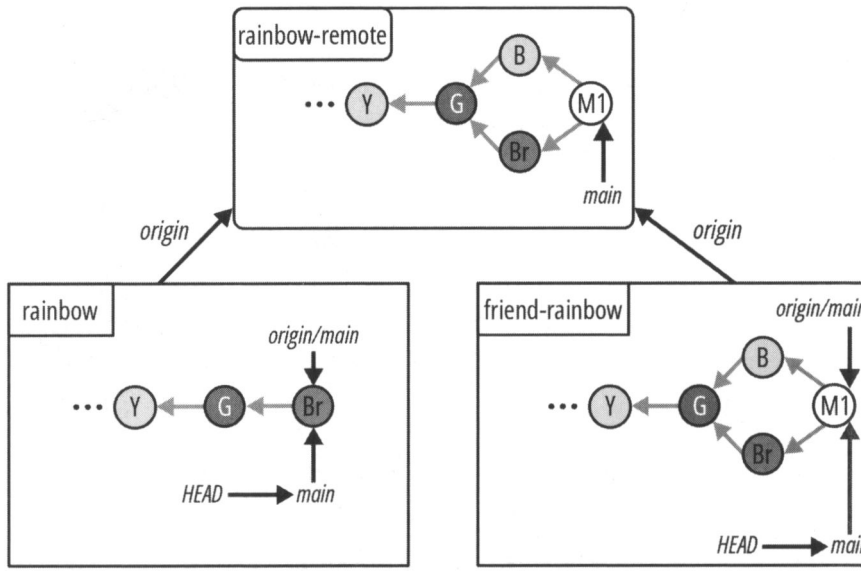
```
commit 342bbfc96bb03053f23ea7f7564ca207c58ceab2
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 13:00:56 2022 +0100

    blue
```

5 Перейдите в репозиторий `rainbow-remote` на своем хостинге и обновите страницу. Там должен появиться коммит слияния.

На что обратить внимание:

- на шаге 2 выходные данные команды `git-cat file -p` показывают два родительских коммита: `342bbfc96bb03053f23ea7f7564ca207c58ceab2` (голубой коммит) и `7f0a87a318e50638eec50a484bf8dfa76b76d08e` (коричневый коммит);
- на шаге 5 вы видите, что коммит слияния находится в удаленном репозитории. Эти наблюдения отражены на диаграмме 9.13.

ДИАГРАММА 9.13



Проект Rainbow после того, как ваш друг отправил свои изменения в удаленный репозиторий.

Коммит слияния M1 теперь находится в репозиториях `friend-rainbow` и `rainbow-remote`. Следующим шагом будет синхронизация репозитория `rainbow` с удаленным репозиторием, а также обновление локальной ветки `main` коммитом слияния. Для этого вы узнаете, как использовать Git.

Извлечение изменений из удаленного репозитория

До сих пор в проекте Rainbow, когда вы хотели обновить свой локальный репозиторий изменениями из удаленного репозитория, вы делали это в два этапа: сначала вы получали данные из удаленного репозитория (с помощью команды `git fetch`), а затем объединяли данные (с помощью команды `git merge`) в локальную ветку. Извлечение данных позволяет сделать и то и другое за один раз.

В Git мы используем термин *"извлечение"*, или pull, для обозначения процесса извлечения данных из удаленного репозитория и их интеграции в ветку локального репозитория за один раз; для этого предназначена команда `git pull`. Если для вашей локальной ветки не определена вышестоящая ветка, вы должны указать короткое имя удаленного репозитория и имя ветки, которую вы хотите обновить. Если у вас определена восходящая ветка, можете просто выполнить `git pull` без каких-либо аргументов.

ЗАПОМНИТЕ КОМАНДУ

`git pull <короткое_имя> <имя_ветки>`

Извлечение изменений из удаленного репозитория `<короткое_имя>` и слияние с указанной веткой `<имя_ветки>`.

`git pull`

Если для текущей ветки определена восходящая ветка, то извлечение изменений из удаленного репозитория и слияние с текущей веткой.

Есть еще одна вещь, которую вам нужно знать о команде `git pull`. Как вы знаете, в Git есть два способа интеграции изменений: слияние и перебазирование. Какой метод задействует команда `git pull`, будет зависеть от того, разошлись ли истории развития веток, и если да, от того, какой вариант вы выберете при вводе команды:

- если истории разработки локальной и удаленной веток в `git pull` не разошлись, то по умолчанию произойдет ускоренное слияние;
- если истории разработки локальной и удаленной веток в `git pull` разошлись, вы должны сообщить Git, как вы хотите интегрировать изменения — путем слияния или перебазирования (в противном случае вы получите сообщение об ошибке). Для того чтобы указать Git на необходимость интеграции изменения путем слияния, вы должны передать команде параметр `--no-rebase`. Для того чтобы выбрать интеграцию изменения путем перебазирования, вы должны передать команде параметр `--rebase`.

Как показано на рис. 9.7, команда `git pull` фактически сочетает в себе команду `git fetch` и команду `git merge` или `git rebase`.

Рис. 9.7. Команда `git pull` извлекает и интегрирует изменения за один раз

Теперь вопрос в том, когда вам следует получать и интегрировать изменения в два этапа, используя команду `git fetch`, а затем либо команду `git merge`, либо команду `git rebase`, и когда вам следует выполнить оба этих шага за один раз, просто используя команду `git pull`?

Пользователи Git обычно используют команду `git pull`, когда истории разработки локальной и удаленной веток не расходятся, и поэтому происходит простое ускоренное слияние. Если истории разработки локальной и удаленной веток разошлись, пользователи Git часто предпочитают использовать команду `git fetch`, а затем отдельно выбирать между перебазированием и слиянием. Выполняя процесс в два этапа, они дают себе больше времени, чтобы посмотреть, что изменится в их локальной ветке, и подготовиться к процессу интеграции изменений.

В этой книге мы будем следовать этой общепринятой практике и, следовательно, будем применять только команду `git pull`, когда быстрое слияние обновит вашу локальную ветку.

Выполните действия 9.12, чтобы попрактиковаться в использовании команды `git pull`, перенося изменения из удаленной ветки `main` в локальную ветку `main` репозитория `rainbow`.

ДЕЙСТВИЯ 9.12

1
```
rainbow $ git pull
remote: Enumerating objects: 9, done.
Remote: Counting objects: 100% (8/8), done.
Remote: Compressing objects: 100% (3/3), done.
Remote: Total 5 (delta 1), reused 5 (delta 1), pack-reused 0
Unpacking objects: 100% (5/5), 566 bytes | 141.00 KiB/s, done.
From github.com:gitlearningjourney/rainbow-remote
   7f0a87a..2258399 main -> origin/main
Updating 7f0a87a..2258399
Fast-forward
 rainbowcolors.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

2
```
rainbow $ git log
commit 225839938563c7458af81daca7beb782dfcbfb27 (HEAD -> main,
origin/main)
Merge: 342bbfc 7f0a87a
```

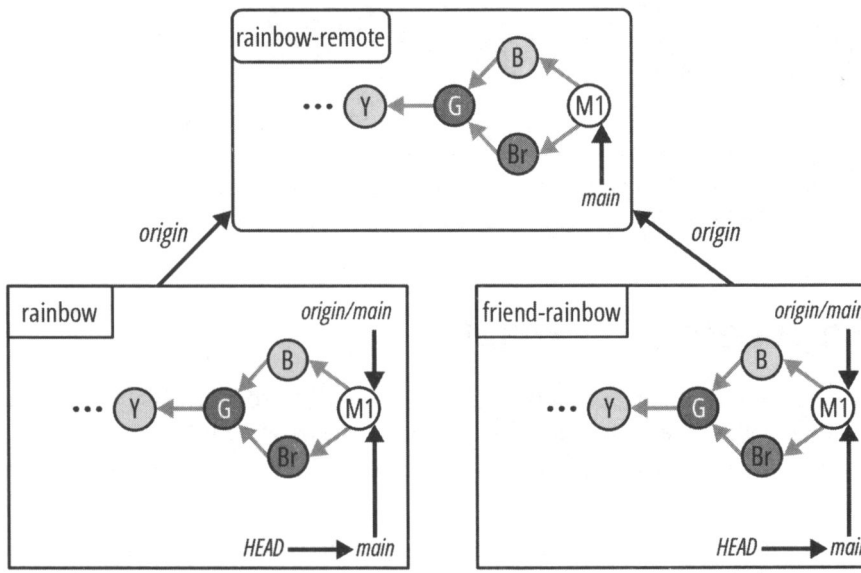
```
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 13:05:46 2022 +0100

    Merge remote-tracking branch 'refs/remotes/origin/main'

commit 342bbfc96bb03053f23ea7f7564ca207c58ceab2
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 13:00:56 2022 +0100

    blue
```

На что обратить внимание:

- первая половина вывода `git pull` на шаге 1 показывает, что выполняется получение (fetch) данных. Вторая половина вывода указывает на то, что было выполнено ускоренное слияние;
- вывод `git log` на шаге 2 показывает, что теперь у вас есть голубой коммит и коммит слияния в репозитории `rainbow`, а ветка удаленного отслеживания `origin/main` и ветка `main` обновлены, чтобы указывать на коммит слияния.

Диаграмма 9.14 иллюстрирует эти наблюдения.

ДИАГРАММА 9.14



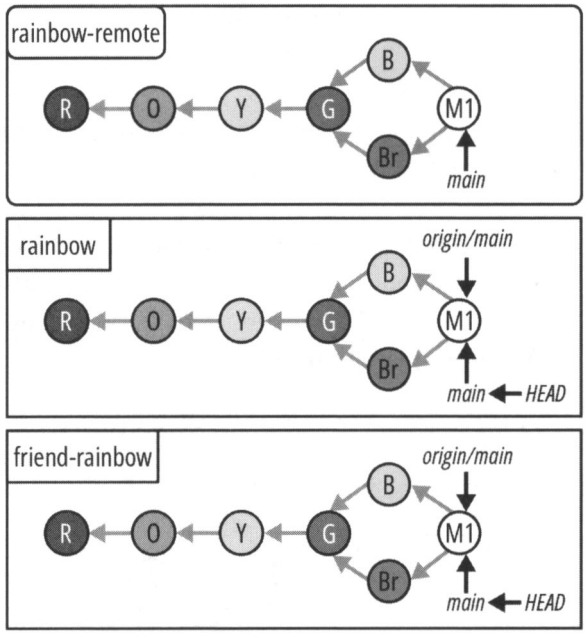
Проект Rainbow после переноса изменений из удаленного репозитория в ветку `main` репозитория `rainbow`.

Все три репозитория в проекте Rainbow теперь синхронизированы.

Состояние локальных и удаленных репозиториев

На диаграмме 9.15 показано состояние локальных и удаленных репозиториев в проекте Rainbow со всеми коммитами, сделанными с *главы 1* до конца текущей главы.

ДИАГРАММА 9.15



Состояние проекта Rainbow в конце *главы 9* со всеми изменениями, внесенными после *главы 1*.

Краткое содержание главы

В этой главе вы узнали о трехсторонних слияниях. Для того чтобы увидеть, как они работают, вы внесли изменения в проект Rainbow, которые привели к тому, что истории локальной ветки `main` в репозитории `friend-rainbow` и удаленной ветки `main` разошлись.

В процессе создания этих расходящихся историй вы узнали, как определять восходящие ветки, и увидели, что после их определения можно выполнять такие команды, как `git push`, без указания каких-либо дополнительных аргументов.

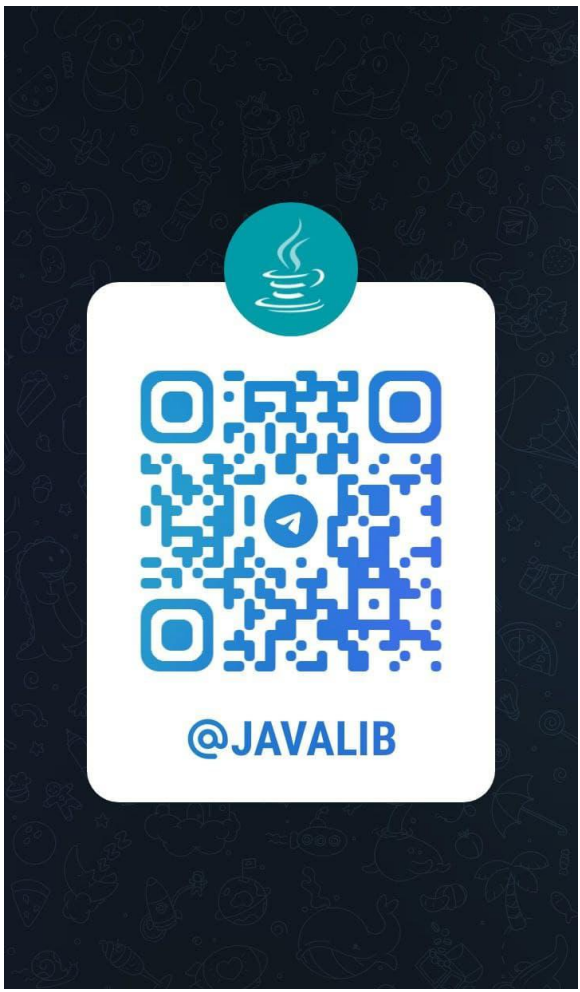
Вы также узнали важную деталь о многократном редактировании файлов в рабочем каталоге между коммитами: каждый раз, когда вы вносите изменения в файл и со-

храняете его, Git интерпретирует это как новую версию файла и помечает файл как модифицированный. Для того чтобы включить последнюю версию файла в следующий коммит, вам нужно будет добавить эту версию файла в промежуточную область.

Когда пришло время выполнить трехстороннее слияние, вы узнали о текстовом редакторе командной строки Vim и научились сохранять предложенное Git сообщение коммита и выходить из Vim. В конце процесса слияния вы увидели, как коммит слияния связал воедино две расходящиеся истории разработки.

Наконец, вы узнали о процессе извлечения (pull) и о том, что он позволяет выполнить два этапа интеграции изменений из удаленного репозитория в локальный репозиторий за один раз.

Трехстороннее слияние, которое вы выполнили в этой главе, не имело никаких конфликтов слияния. В *главе 10* вы узнаете о конфликтах слияния и способах их разрешения.



[Глава 10]

Конфликты слияния

В *главе 9* вы выполнили трехстороннее слияние, при котором не возникло никаких конфликтов слияния.

В этой главе вы узнаете о конфликтах слияния, о том, как они возникают и как их разрешить, рассмотрев практический пример трехстороннего слияния, вызывающего конфликт.

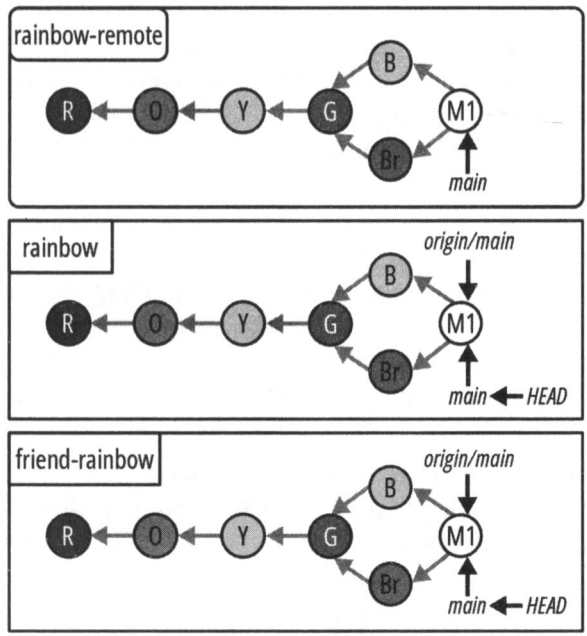
Состояние локальных и удаленных репозиториев

В начале этой главы у вас должны быть два локальных репозитория под названием `rainbow` и `friend-rainbow` и один удаленный репозиторий под названием `rainbow-remote`. Все три репозитория должны быть синхронизированы, с одинаковыми коммитами и ветками. Как обычно, я рекомендую вам открыть два отдельных окна файловой системы и командной строки для работы с репозиториями `rainbow` и `friend-rainbow`.

На диаграмме 10.1 показано состояние локальных и удаленных репозиториев в проекте Rainbow со всеми коммитами, сделанными в *главах 1–9*.

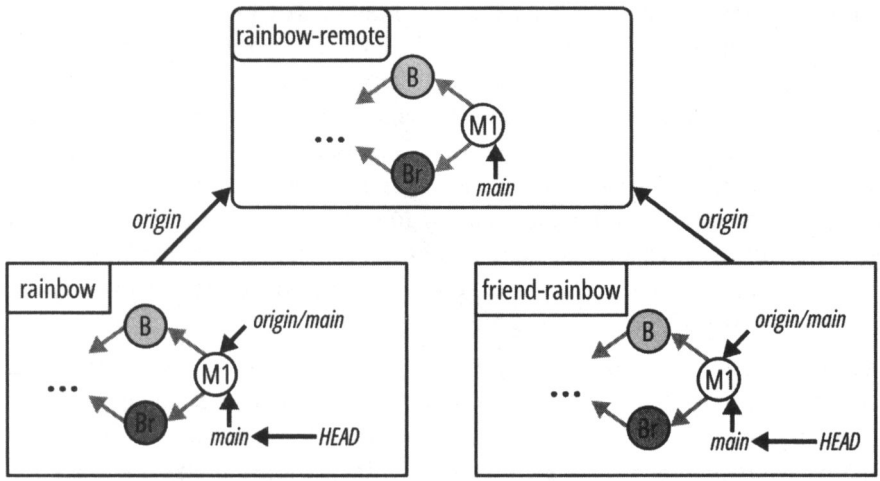
Для того чтобы сосредоточиться на коммитах, которые вы сделаете в данной главе, с этого момента я упрощу диаграммы состояния и буду показывать только три последних коммита, которые являются частью основной ветки во всех репозиториях: голубой коммит, коричневый коммит и коммит слияния M1. Это представление показано на диаграмме 10.2.

ДИАГРАММА 10.1



Проект Rainbow в начале *главы 10* со всеми изменениями, внесенными после *главы 1*.

ДИАГРАММА 10.2



Упрощенное представление проекта Rainbow в начале *главы 10*, показывающее только три последних коммита в основной ветке во всех репозиториях.

Что такое конфликты слияния

В *главе 9* вы узнали, что происходит, когда вы работаете над проектом одновременно с кем-то еще, но каждый из вас вносит изменения в разные файлы: вы редактировали файл othercolors.txt, а ваш друг редактировал файл rainbowcolors.txt. Это означало, что вы могли выполнить трехстороннее слияние без каких-либо конфликтов слияния.

Теперь я расскажу, что происходит, когда вы хотите объединить работу, в которой возникает конфликт слияния. Как вы узнали из предыдущей главы, конфликты слияния возникают, когда вы объединяете две ветки, в которых в одни и те же части одного и того же файла (файлов) были внесены разные изменения, или если в одной ветке был удален файл, который редактировался в другой ветке. В таких случаях Git не может автоматически объединить файлы, поэтому вам придется делать это вручную.

Конфликты слияния могут возникнуть как в процессе слияния, так и в процессе перебазирования. В этой главе мы рассмотрим пример конфликта при слиянии, а в *главе 11* обсудим пример конфликта слияния, возникающего в проекте Rainbow при перебазировании.

Имейте в виду, что возникновение конфликтов слияния — это нормально, и если они возникают, это не означает, что кто-то сделал ошибку во время работы над проектом. Разрешение конфликтов слияния — обычная часть работы над проектами Git. Ознакомьтесь с примером 10.1 проекта Book, где возникает именно такая заурядная ситуация.

Пример 10.1 проекта Book

Давайте попробуем представить пару сценариев, в которых я могу столкнуться с конфликтами слияния в моем проекте Book. Как вы знаете, проект состоит из 10 текстовых файлов, по одному на каждую главу книги, с именами chapter_one.txt, chapter_two.txt и т. д. Моя основная линия разработки — это ветка `main`, и вместе со своим редактором и соавтором я решила, что всякий раз, когда я работаю над главой, я буду создавать ответвление от ветки `main`; затем, после того как они оба одобрят работу, которую я проделала над этой веткой, я могу снова объединить ее с веткой `main`. Я также согласна со своим соавтором, что только один из нас должен работать над каждой главой в любой момент времени, и таким образом мы преимущественно можем избежать конфликтов слияния.

Но теперь давайте представим, что произойдет, если мы с моим соавтором случайно ошибемся. Мы оба одновременно отредактируем главу 3 и оба создадим ответвления от ветки `main` для работы над этой главой. Я создаю ветку `chapter_three`, а мой соавтор — ветку `chapter_three_coauthor`. В данном случае мы оба собираемся редактировать один и тот же файл chapter_three.txt.

Теперь предположим, что мой соавтор первым успевает объединить свою работу из ветки `chapter_three_coauthor` с веткой `main`. Когда я попытаюсь объединить свою

ветку chapter_three с самой последней версией ветки main, я обнаружу, что мне предстоит не только выполнить трехстороннее слияние, но и разрешить возникающий при этом конфликт слияния.

Это связано с тем, что версия файла chapter_three.txt в моей ветке chapter_three и версия файла в ветке main были отредактированы по-разному с тех пор, как я создала свою рабочую ветку вне ветки main. Git не может автоматически объединить работу. Мне нужно принять решение, как именно должна выглядеть окончательная версия файла chapter_three.txt, который будет включен в коммит слияния.

Это один из сценариев, в котором я могу столкнуться с конфликтами слияния. Так происходит, когда один и тот же файл был отредактирован по-разному в двух ветках, участвующих в слиянии.

Далее давайте рассмотрим другой сценарий, в котором может возникнуть конфликт слияния. Допустим, я и мой соавтор пришли к выводу, что глава 10 никуда не годится. Однако мы снова неправильно поняли друг друга относительно того, что нам следует с этим делать.

Мы случайно оба одновременно создаем ответвления от ветки main: я создаю ветку chapter_ten, а мой соавтор создает ветку chapter_ten_coauthor. В своей ветке chapter_ten я решила удалить файл chapter_ten.txt, потому что считаю, что нам следует просто целиком удалить эту главу из нашей книги. Однако мой соавтор, работающий над веткой chapter_ten_coauthor, вместо этого решает отредактировать файл chapter_ten.txt, чтобы сделать его лучше.

Опять же, мой соавтор успевает первым объединить свою ветку с веткой main. Когда я соберусь объединить свою ветку с самой современной веткой main посредством трехстороннего слияния, я снова столкнусь с конфликтом слияния. Это еще один сценарий, в котором Git не может автоматически объединить работу; он не может определить, следует ли удалить файл chapter_ten.txt или сохранить отредактированную версию. Напомню, что эта ситуация возникает, когда я объединяю две ветки, где в одной ветке был отредактирован определенный файл, а в другой ветке тот же файл был удален.

Пример 10.1 проекта Book демонстрирует несколько сценариев, в которых могут возникнуть конфликты слияния. Теперь давайте посмотрим, как их можно разрешить.

Как разрешить конфликты слияния

Когда случаются конфликты слияния, в каждом из задействованных файлов вы увидите набор специальных маркеров, указывающих, где происходят конфликты. Эти маркеры, называемые маркерами конфликта, состоят из семи левых угловых скобок (<<<<<<<), семи знаков равенства (=======) и семи правых угловых скобок (>>>>>>>), а также ссылок на ветки, участвующие в слиянии. На рис. 10.1 показан пример обозначения. Над строкой знаков равенства вы увидите содержимое целевой ветки, а под ней — содержимое исходной ветки.

```
<<<<<<< HEAD
{Content of the target branch}
=======
{Content of the source branch}
>>>>>>>refs/remote/origin/main
```

Рис. 10.1. Пример маркеров конфликта, которые появляются при возникновении конфликта слияния

Процесс разрешения конфликтов слияния состоит из двух шагов:

1. Решите, что оставить, отредактируйте содержимое и удалите маркеры конфликта.
2. Добавьте файлы, которые вы отредактировали, в промежуточную область и сделайте коммит изменений.

Мы подробно рассмотрим каждый из этих шагов, когда вы попрактикуетесь в разрешении конфликтов слияния в проекте Rainbow далее в этой главе. Для того чтобы подготовиться к этому, вам необходимо создать ситуацию с разной историей развития в ваших локальных репозиториях.

Подготовка сценария конфликта слияния

Для того чтобы подготовить основу для трехстороннего слияния с конфликтом, вам и вашему другу придется внести разные изменения в одну и ту же часть одного и того же файла. Сначала выполните действия 10.1, чтобы добавить синий цвет (`indigo`) в список цветов в файле rainbowcolors.txt в вашем репозитории `rainbow` и отправить обновленную ветку `main` в удаленный репозиторий.

ДЕЙСТВИЯ 10.1

1. В каталоге проекта `rainbow` в текстовом редакторе добавьте "Синий — шестой цвет радуги." в строке 6 файла rainbowcolors.txt. Затем сохраните файл.
2.
```
rainbow $ git add rainbowcolors.txt
```
3.
```
rainbow $ git commit -m "indigo"
[main 9b0a614] indigo
 1 file changed, 2 insertions(+), 1 deletion(-)
```
4.
```
rainbow $ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 326 bytes | 326.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:gitlearningjourney/rainbow-remote.git
   2258399..9b0a614 main -> main
```

5 rainbow $ **git log**

```
commit 9b0a61461c8e8d74ed358e65b2662e3697b94de6 (HEAD -> main,
origin/main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 08:36:11 2022 +0100

    indigo

commit 225839938563c7458af81daca7beb782dfcbfb27
Merge: 342bbfc 7f0a87a
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 13:05:46 2022 +0100

    Merge remote-tracking branch 'refs/remotes/origin/main'
```

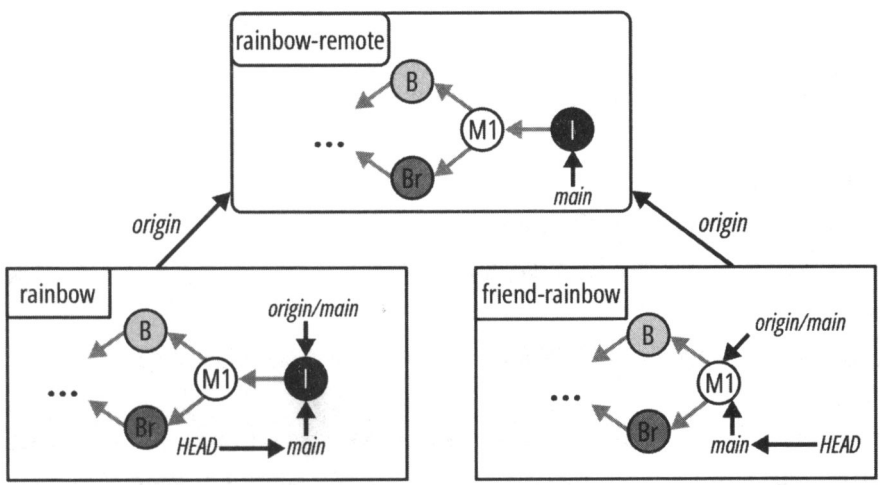
6 Перейдите в репозиторий `rainbow-remote` на своем хостинге и обновите страницу. Вы должны увидеть там синий коммит.

На что обратить внимание:

- в репозиториях `rainbow` и `rainbow-remote` имеется коммит `indigo`.

Эти изменения отражены на диаграмме 10.3.

ДИАГРАММА 10.3



Проект Rainbow после того, как вы сделаете синий коммит в своем репозитории `rainbow` и отправите обновленную ветку `main` в удаленный репозиторий.

Теперь, чтобы создать ситуацию, в которой у вас возникнет конфликт слияния, в действиях 10.2 ваш друг добавит фиолетовый цвет в ту же строку в файле

rainbowcolors.txt и сделает коммит, не извлекая предварительно внесенные вами изменения из репозитория `rainbow-remote` в свою локальную ветку `main`.

ДЕЙСТВИЯ 10.2

1. В каталоге проекта `friend-rainbow` в текстовом редакторе добавьте в файл rainbowcolors.txt строку "Фиолетовый — седьмой цвет радуги." в строке 6. Сохраните файл.
2.
```
friend-rainbow $ git add rainbowcolors.txt
```
3.
```
friend-rainbow $ git commit -m "violet"
[main 6ad5c15] violet
 1 file changed, 2 insertions(+), 1 deletion(-)
```
4.
```
friend-rainbow $ git log
commit 6ad5c15f033b68ad27f2c9bce8bfa93329b3c23e (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 08:41:25 2022 +0100

    violet

commit 225839938563c7458af81daca7beb782dfcbfb27 (origin/main,
origin/HEAD)
Merge: 342bbfc 7f0a87a
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sat Feb 19 13:05:46 2022 +0100

    Merge remote-tracking branch 'refs/remotes/origin/main'
```

На что обратить внимание:

- ваш друг сделал фиолетовый коммит.

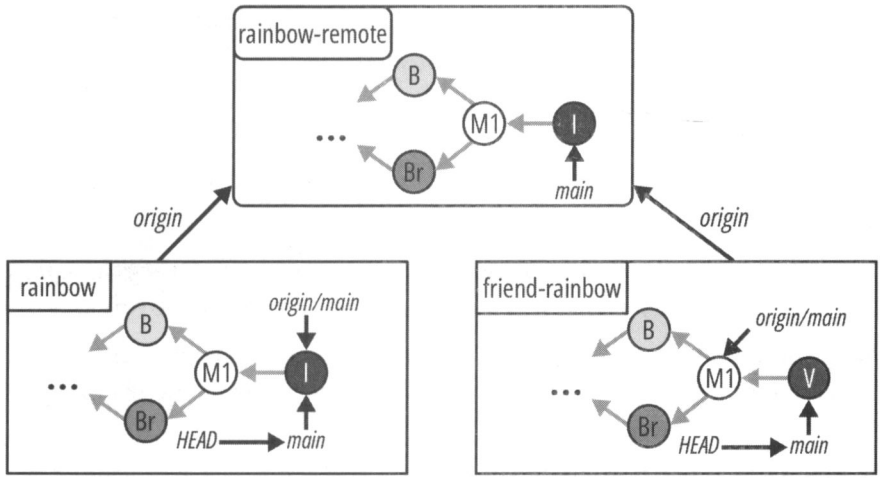
Текущее состояние показано на диаграмме 10.4.

На что обратить внимание:

- в репозитории `friend-rainbow` локальная ветка `main` указывает на фиолетовый коммит;
- в репозитории `rainbow` локальная ветка `main` указывает на синий коммит.

На этом этапе вашему другу нужно будет получить ваши изменения из удаленного репозитория и интегрировать их, прежде чем он сможет отправить свои изменения в удаленный репозиторий. Как вы узнали из *разд. "Отправка в удаленный репозиторий" главы 8*, после получения ваших изменений ваш друг может использовать команду `git status`, чтобы проверить, не отличается ли его локальная ветка `main` от удаленной ветки `main`. Выполните действия 10.3, чтобы увидеть это на примере.

ДИАГРАММА 10.4



Проект Rainbow после того, как ваш друг сделает фиолетовый коммит в репозитории `friend-rainbow` без предварительного извлечения последних изменений из ветки `main` в удаленном репозитории.

ДЕЙСТВИЯ 10.3

1
```
friend-rainbow $ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), 322 bytes | 53.00 KiB/s, done.
From github.com:gitlearningjourney/rainbow-remote
   c5941f8..6f2ea44 main -> origin/main
```

2
```
friend-rainbow $ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

nothing to commit, working tree clean
```

3
```
friend-rainbow $ git log --all
commit 6ad5c15f033b68ad27f2c9bce8bfa93329b3c23e (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 08:41:25 2022 +0100

    violet
```

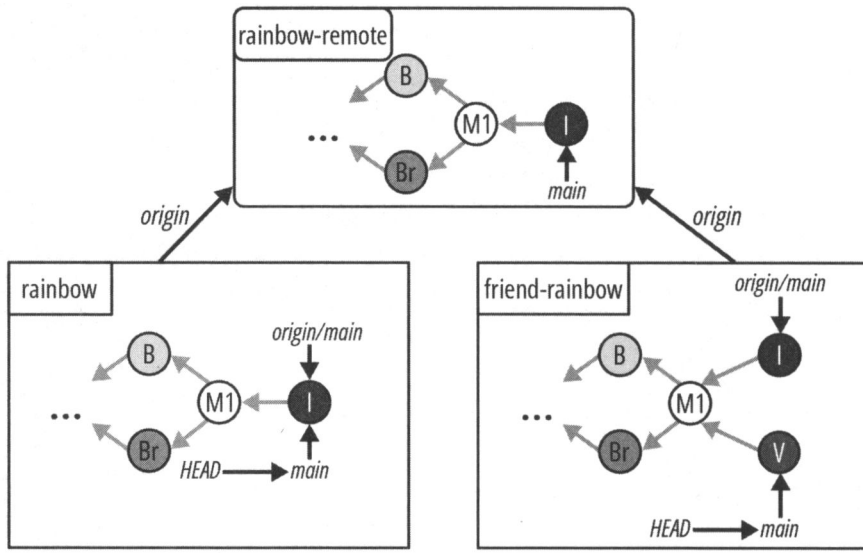
```
commit 9b0a61461c8e8d74ed358e65b2662e3697b94de6 (origin/main,
origin/HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 08:36:11 2022 +0100

    indigo
```

На что обратить внимание:

- на шаге 2 выходные данные команды `git status` сообщают, что ваша ветка и `origin/main` разошлись и имеют 1 и 1 разных коммитов соответственно;
- в репозитории `friend-rainbow` ветка удаленного отслеживания `origin/main` указывает на синий коммит; однако удаленные изменения еще не были объединены с локальной веткой `main`.

Эти наблюдения отражены на диаграмме 10.5.

ДИАГРАММА 10.5



Проект Rainbow после того, как ваш друг извлечет изменения из удаленного репозитория в репозиторий `friend-rainbow`.

Затем ваш друг собирается выполнить трехстороннее слияние и тем самым включить последние изменения из удаленной ветки `main` в свою локальную ветку `main`, чтобы иметь возможность отправить обновленную ветку `main` в удаленный репозиторий. Прежде чем он выполнит слияние и разрешит конфликты слияния в репозитории `friend-rainbow`, давайте подробнее разберемся, что влечет за собой этот процесс.

Процесс разрешения конфликтов слияния

Как упоминалось ранее в этой главе, процесс разрешения конфликтов слияния состоит из двух шагов. В этом разделе мы рассмотрим их более подробно. Напомню, что на первом шаге вы должны решить, что сохранить, отредактировать содержимое файла и удалить маркеры конфликта, а на втором — добавить все изменения в промежуточную область и сделать коммит. Далее мы рассмотрим оба шага по порядку.

Шаг 1

После того как вы выполните команду `git merge` в ситуации, когда возникает конфликт слияния, Git обнаружит конфликт и вставит маркеры конфликта, указывающие расположение конфликтующего содержимого. Первый шаг в разрешении конфликтов слияния — решить, что именно сохранить, отредактировать содержимое файлов, в которых возникают конфликты, и удалить маркеры конфликта. После того как вы закончите вносить необходимые изменения, вы также должны обязательно сохранить файл или файлы в текстовом редакторе. В проекте Rainbow конфликт слияния будет возникать только в одном файле, но в других проектах Git, над которыми вы работаете, конфликты слияния могут возникать в нескольких файлах.

Для того чтобы разрешить конфликт слияния в проекте Rainbow, ваш друг сохранит и предложение о синем цвете, и предложение о фиолетовом цвете. Он хочет перечислить цвета в правильном порядке, поэтому сначала поместит предложение о синем цвете (которое в конечном итоге окажется в строке 6), а за ним предложение о фиолетовом цвете (оно окажется в строке 7). На рис. 10.2 показан пример того, как будет выглядеть файл rainbowcolors.txt до и после того, как ваш друг внесет это изменение.

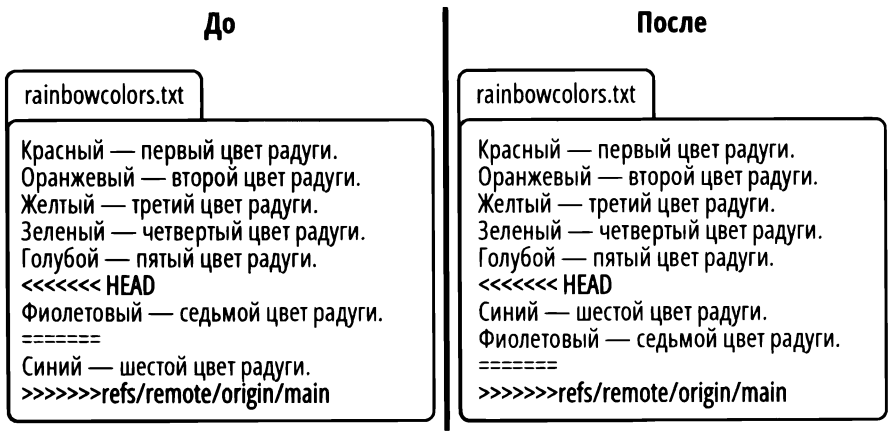


Рис. 10.2. Содержимое файла rainbowcolors.txt до и после того, как ваш друг решит, что сохранить, и отредактирует содержимое

Затем вашему другу нужно будет удалить маркеры конфликта из файла rainbowcolors.txt (рис. 10.3) и сохранить файл.

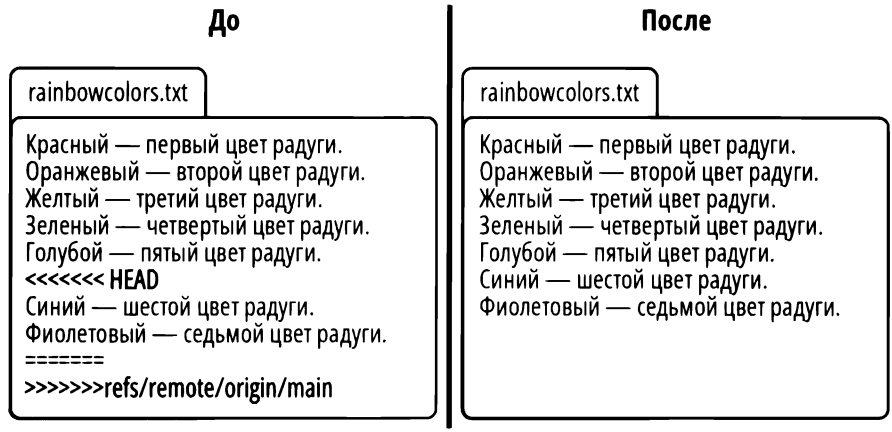


Рис. 10.3. Файл rainbowcolors.txt до и после того, как ваш друг удалит маркеры конфликта

Шаг 2

После того как ваш друг закончил редактирование контента, он готов ко второму шагу: добавлению обновленных файлов в промежуточную область и выполнению коммита. В проекте Rainbow есть только один файл с конфликтом слияния, поэтому в промежуточную область нужно будет добавить лишь один файл. Однако, как упоминалось ранее, часто возникают конфликты слияния в нескольких файлах, и в этом случае вам придется добавить в промежуточную область все измененные файлы.

На рис. 10.4 показан пример того, как будет выглядеть история коммитов в репозитории `friend-rainbow` до и после того, как ваш друг завершит слияние. Это слияние приведет к появлению коммита слияния M2, поскольку это трехстороннее слияние.

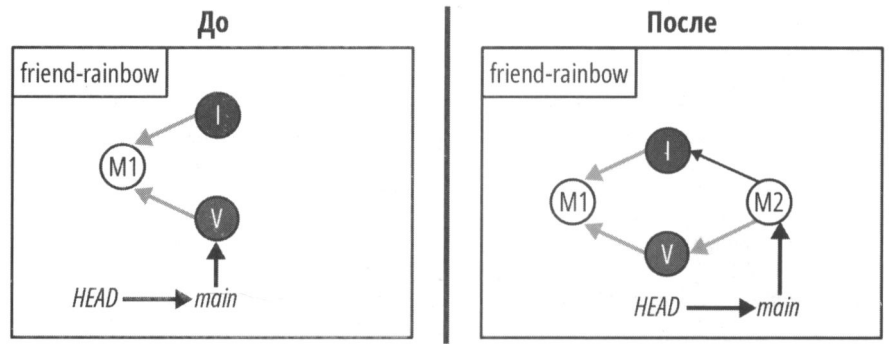


Рис. 10.4. История коммитов до и после того, как ваш друг совершил трехстороннее слияние

Итак, вы знакомы с двумя этапами разрешения конфликтов слияния. Теперь кратко обсудим, что делать, если вы решите, что не хотите продолжать слияние.

Прекращение слияния

Если в какой-то момент во время слияния с конфликтами вы решите, что не хотите продолжать интеграцию двух веток, можете прервать слияние, используя команду `git merge` с опцией `--abort`. Она вернет все ваши файлы в состояние, в котором они находились до слияния.

ЗАПОМНИТЕ КОМАНДУ

git merge --abort

Прерывает процесс слияния и возвращает в состояние до слияния.

Теперь вы знаете, как разрешать конфликты слияния, прерывать слияние и разрешать конфликты. Далее мы рассмотрим пример разрешения конфликтов слияния на практике.

Разрешение конфликтов слияния на практике

Представьте, что ваш друг собирается выполнить действия 10.4, чтобы сделать трехстороннее слияние и разрешить возникшие конфликты. На протяжении всей операции слияния он будет использовать команду `git status` для просмотра информации о процессе разрешения конфликта.

ДЕЙСТВИЯ 10.4

1
```
friend-rainbow $ git merge origin/main
Auto-merging rainbowcolors.txt
CONFLICT (content): Merge conflict in rainbowcolors.txt
Automatic merge failed; fix conflicts and then commit the result.
```

(КОНФЛИКТ (содержимое): конфликт слияния в rainbowcolors.txt

Автоматическое слияние не удалось; устраните конфликт, затем сделайте коммит результата)

2
```
friend-rainbow $ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
```

```
Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified: rainbowcolors.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

(Ваша ветка и 'origin/main' разошлись,
и имеют 1 и 1 разных коммитов каждая соответственно.
(выполните "git pull" для слияния удаленной ветки с вашей)

У вас есть не слитые пути.
(исправьте конфликт и введите "git commit")
(используйте "git merge --abort" для отмены слияния)

Не слитые пути:
(используйте "git add <file>..." чтобы отметить решение)

В коммит не внесены изменения (используйте "git add" и/или "git commit -a"))

3. Сделайте глубокий вдох, перейдите в каталог проекта `friend-rainbow` в текстовом редакторе и просмотрите файл rainbowcolors.txt. Найдите маркеры конфликта и конфликтующее содержимое.
4. Выполните шаг 1 разрешения конфликтов слияния, т. е. выберите, что именно следует сохранить, отредактируйте содержимое и удалите маркеры конфликта. В вашем случае вы сохраните весь контент, т. е. изменения из фиолетового коммита и изменения из синего коммита. Текст о синем цвете будет расположен в строке 6, а текст о фиолетовом цвете — в строке 7. Обязательно сохраните файл rainbowcolors.txt, когда закончите вносить изменения.
5. `friend-rainbow $ git add rainbowcolors.txt`
6. `friend-rainbow $ git status`

```
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)
```

```
Changes to be committed:
        modified: rainbowcolors.txt

(...
Все конфликты устранены, но слияние не завершено.
 (выполните "git commit" для завершения слияния)
Изменения, ожидающие коммита:
        ...)
```

7 `friend-rainbow $ git commit -m "merge commit 2"`

```
[main f10f972] merge commit 2
```

8 `friend-rainbow $ git log`

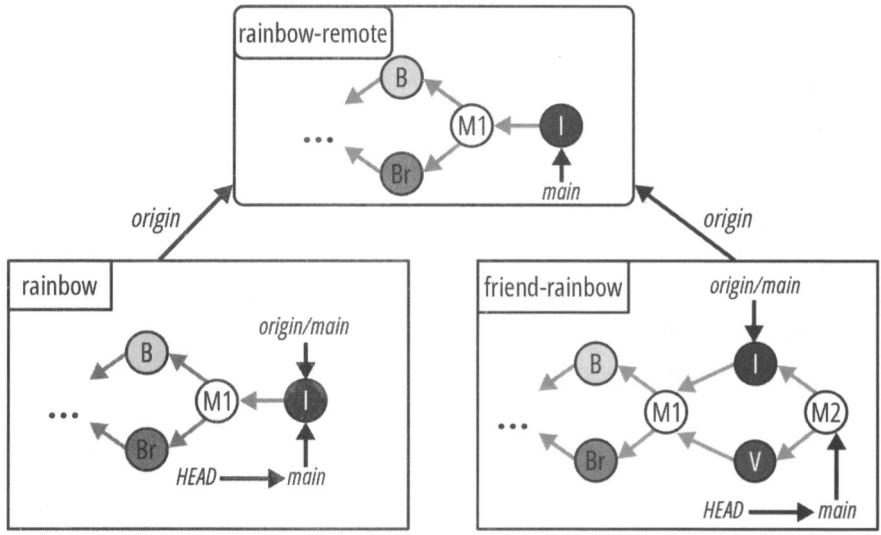
```
commit f10f9725e3319af840a3d891ca8950436a219eb0 (HEAD -> main)
Merge: 6ad5c15 9b0a614
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 09:11:06 2022 +0100

    merge commit 2

commit 6ad5c15f033b68ad27f2c9bce8bfa93329b3c23e
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 08:41:25 2022 +0100

    violet

commit 9b0a61461c8e8d74ed358e65b2662e3697b94de6 (origin/main,
origin/HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 08:36:11 2022 +0100

    indigo
```

На что обратить внимание:

- на шаге 8 выходные данные команды `git log` показывают:
  - локальная ветка `main` указывает на коммит слияния 2;
  - двумя родителями коммита слияния 2 являются `6ad5c15` (фиолетовый коммит) и `9b0a614` (синий коммит). Помните, что хеши коммитов в вашем репозитории `friend-rainbow` будут отличаться от хешей в этой книге, поскольку хеши коммитов уникальны.

Эти наблюдения отражены на диаграмме 10.6, где новый коммит слияния обозначен как M2.

ДИАГРАММА 10.6



Проект Rainbow после того, как ваш друг разрешит конфликты слияния в репозитории `friend-rainbow` и выполнит еще одно слияние.

Ваш друг успешно выполнил трехстороннее слияние и разрешил конфликты слияния. Далее мы обсудим важность регулярного доступа к удаленному репозиторию.

Оставайтесь на связи с удаленным репозиторием

При возникновении конфликтов слияния интеграция изменений из одной ветки в другую занимает больше времени. В примере проекта Rainbow был только один файл с небольшим конфликтом слияния. Однако в реальных проектах может быть намного больше файлов, требующих разрешения гораздо более сложных конфликтов слияния. Очень важно быть в курсе изменений, внесенных в соответствующие ветки удаленного репозитория, поскольку это помогает ограничить количество потенциальных конфликтов слияния, которые вам, возможно, придется разрешать позже.

Всякий раз, когда вы создаете новую ветку, рекомендуется использовать самую последнюю версию соответствующей удаленной ветки в удаленном репозитории. Часто это будет ветка `main` (или любая другая ветка, которую ваша команда использует в качестве основного направления разработки). Однако в зависимости от рабочего процесса Git вашей команды это может быть и другая ветка.

Если вы работаете над существующей веткой самостоятельно, рекомендуется обновить вашу ветку с учетом изменений, внесенных в соответствующую ветку в

удаленном репозитории, путем слияния. Если возникнет ситуация, когда вы работаете над той же веткой, что и кто-то другой, вы должны обязательно извлечь и объединить все изменения, сделанные в этой ветке в удаленном репозитории, прежде чем продолжить работу над ней.

Мы выяснили, почему нужно постоянно быть в курсе последних событий. Давайте убедимся, что ваш друг перенес работу, которую он проделал в своей локальной основной ветке, в удаленный репозиторий, и что вы синхронизировали свою локальную ветку `main` в репозитории `rainbow` с удаленной веткой `main`.

Синхронизация репозиториев

Для того чтобы все репозитории были синхронизированы, вашему другу нужно будет отправить новые коммиты из своей локальной ветки `main` в удаленный репозиторий, а вам нужно будет перенести все изменения в вашу локальную ветку `main` репозитория `rainbow`. Выполните действия 10.5, чтобы сделать это прямо сейчас.

ДЕЙСТВИЯ 10.5

1
```
friend-rainbow $ git push
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 614 bytes | 614.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local
object.
To github.com:gitlearningjourney/rainbow-remote.git
9b0a614..f10f972 main -> main
```

2
```
friend-rainbow $ git log
commit f10f9725e3319af840a3d891ca8950436a219eb0 (HEAD -> main,
origin/main, origin/HEAD)
Merge: 6ad5c15 9b0a614
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 09:11:06 2022 +0100

    merge commit 2

commit 6ad5c15f033b68ad27f2c9bce8bfa93329b3c23e
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 08:41:25 2022 +0100
    violet
```

3 Перейдите в окно командной строки, где вы находитесь в репозитории `rainbow`, и выполните следующую команду.

4
```
rainbow $ git pull
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (4/4), done.
Unpacking objects: 100% (6/6), 594 bytes | 99.00 KiB/s, done.
remote: Total 6 (delta 2), reused 6 (delta 2), pack-reused 0
From github.com:gitlearningjourney/rainbow-remote
9b0a614..f10f972 main -> origin/main
Updating 9b0a614..f10f972
Fast-forward
rainbowcolors.txt | 5 ++++-
 1 file changed, 4 insertions(+), 1 deletion(-)
```

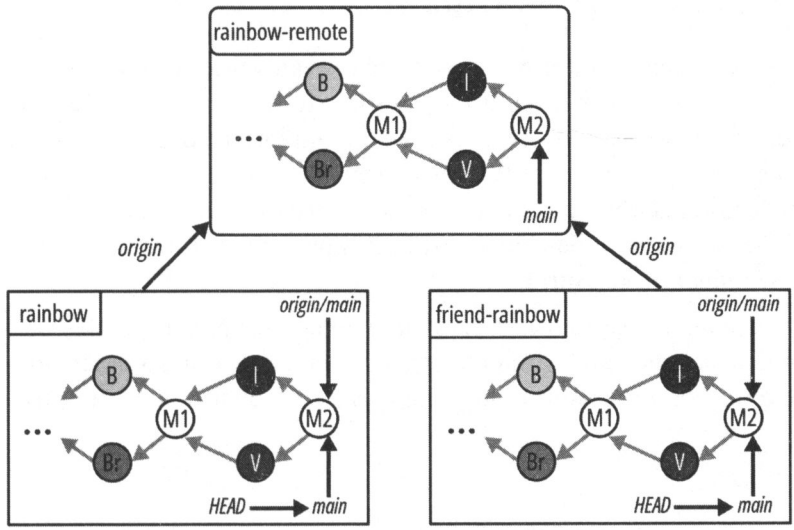
5
```
rainbow $ git log
commit f10f9725e3319af840a3d891ca8950436a219eb0 (HEAD -> main,
origin/main)
Merge: 6ad5c15 9b0a614
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 09:11:06 2022 +0100

    merge commit 2

commit 6ad5c15f033b68ad27f2c9bce8bfa93329b3c23e
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 08:41:25 2022 +0100

    violet
```

Все три репозитория в проекте Rainbow теперь синхронизированы, как показано на диаграмме 10.7.

На что обратить внимание:

- в репозитории `rainbow-remote` ветка `main` указывает на коммит слияния M2;
- в репозитории `rainbow` локальная ветка `main` и ветка удаленного отслеживания `origin/main` указывают на коммит слияния M2;
- в репозитории `friend-rainbow` ветка удаленного отслеживания `origin/main` обновлена и теперь указывает на коммит слияния M2.

ДИАГРАММА 10.7

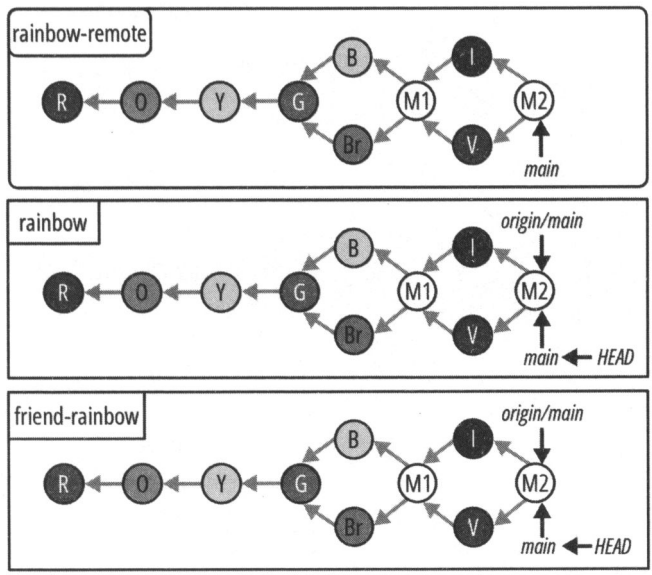


Проект Rainbow после того, как ваш друг отправил изменения в удаленный репозиторий, а вы перенесли их оттуда в свой репозиторий `rainbow`.

Состояние всех репозиториев

На диаграмме 10.8 показано состояние всех репозиториев в проекте Rainbow со всеми коммитами, сделанными с *главы 1* до конца этой главы.

ДИАГРАММА 10.8



Проект Rainbow в конце *главы 10* со всеми изменениями, внесенными после *главы 1*.

Краткое содержание главы

В этой главе вы узнали о конфликтах слияния, которые возникают при объединении двух веток, в которых в одни и те же части одного и того же файла (файлов) были внесены разные изменения, или если в одной ветке был удален файл, который отредактирован в другой ветке. Вы также узнали о двух шагах по разрешению конфликтов: решение, какой контент оставить, удаление маркеров конфликта и сохранение файлов, а затем добавление отредактированных файлов в промежуточную область и выполнение коммита.

Конфликты слияния могут возникнуть либо во время трехстороннего слияния, либо в процессе перебазирования. До сих пор мы говорили только о слиянии; в *главе 11* вы узнаете о перебазировании и потренируетесь на практическом примере в проекте Rainbow.

[Глава 11]

Перебазирование

Как вы уже знаете, Git предлагает два основных способа интеграции изменений из одной ветки в другую: слияние и перебазирование. В *главах 5* и *9* вы узнали об ускоренном слиянии и трехстороннем слиянии соответственно. В *главе 10* вы узнали о конфликтах слияния и выяснили, что они могут возникнуть как во время трехстороннего слияния, так и во время перебазирования.

В этой главе вы узнаете о перебазировании и ознакомитесь с примером. Я расскажу о пяти этапах перебазирования и покажу, как разрешать конфликты слияния во время этого процесса. Мы рассмотрим, чем отличаются слияние и перебазирование, и в каких случаях предпочтительно выбирать перебазирование вместо слияния. Я также представлю "золотое правило перебазирования", которое поможет вам определить, когда лучше избегать перебазирования. Кроме того, вы узнаете немного больше про промежуточную область и про то, как использовать ее в качестве временного пространства для систематизации изменений, которые вы намерены включить в свой следующий коммит.

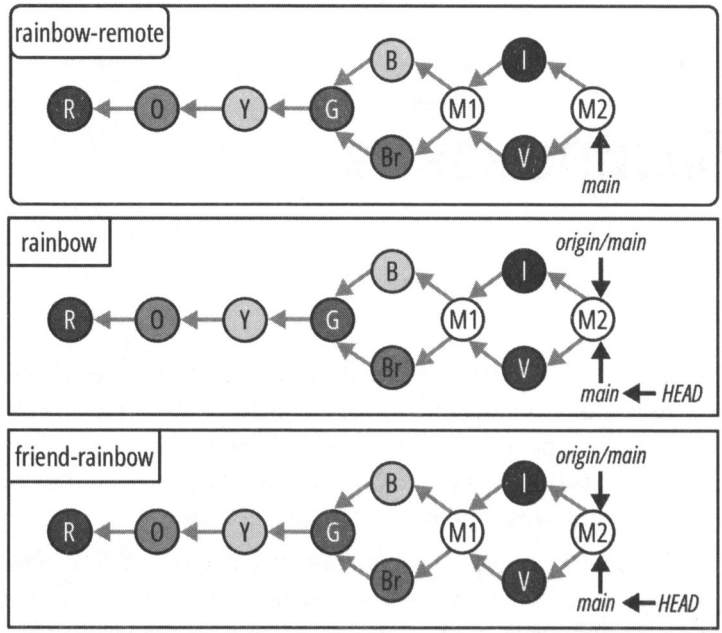
Состояние всех репозиториев

В начале этой главы у вас должны быть два локальных репозитория под названием `rainbow` и `friend-fainbow` и один удаленный репозиторий под названием `rainbow-remote`. Все три репозитория должны содержать одни и те же коммиты и ветки. Как обычно, при работе с примерами в этой главе я рекомендую вам использовать два отдельных окна файловой системы и окна командной строки для репозиториев `rainbow` и `friend-rainbow`.

На диаграмме 11.1 показано состояние локальных и удаленных репозиториев в проекте Rainbow со всеми коммитами, сделанными в *главах 1–10*.

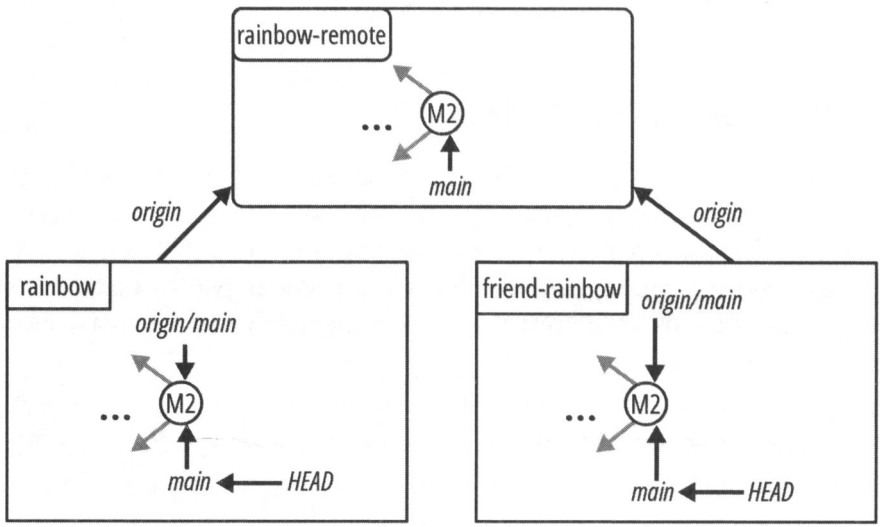
Для того чтобы сосредоточиться на коммитах, которые вы сделаете в этой главе, с этого момента я упрощу диаграммы и буду показывать только последний коммит, сделанный в ветке `main` во всех репозиториях, т. е. коммит слияния M2. Новое представление показано на диаграмме 11.2.

ДИАГРАММА 11.1



Состояние проекта Rainbow в начале *главы 11* со всеми изменениями, внесенными после *главы 1*.

ДИАГРАММА 11.2



Упрощенное представление проекта Rainbow в начале *главы 11*, показывающее только последний коммит в основной ветке во всех репозиториях.

Интеграция изменений в Git

До сих пор нашим единственным способом интеграции изменений в Git было слияние. Вы увидели на примерах, что ускоренное слияние просто перемещает указатель целевой ветки, чтобы указать на последний коммит, тогда как трехстороннее слияние создает коммит слияния, который связывает истории разработки исходной и целевой веток между собой (и иногда может привести к конфликтам слияния).

Если вы посмотрите на состояние различных репозиториев на диаграмме 11.1, вы увидите, что до зеленого коммита у вас была линейная история проекта, но после зеленого коммита история коммитов становится нелинейной из-за коммитов, созданных трехсторонним слиянием. Некоторые команды и отдельные разработчики предпочитают вести линейную историю проекта, поскольку считают ее более организованной и простой. В вашем распоряжении есть процесс перебазирования, позволяющий избежать трехсторонних слияний и коммитов слияния, а также поддерживать линейную историю проекта.

Перебазирование берет на себя всю работу, которую вы проделали и зафиксировали в коммитах одной ветки, и повторно применяет работу к другой ветке, создавая совершенно новые коммиты. Это выглядит так, будто вы создали ветку из совершенно другого коммита, отличного от исходного коммита, из которого она была создана.

Для того чтобы выполнить перебазирование, вам необходимо находиться на той ветке, которую вы хотите перебазировать. Введите команду `git rebase` и передайте ей имя ветки, в которую хотите выполнить перебазирование.

ЗАПОМНИТЕ КОМАНДУ

git rebase <имя_ветки>

Повторно применяет коммиты поверх другой ветки.

Поскольку при перебазировании создаются совершенно новые коммиты, это означает, что изменяется история коммитов. Важно быть осторожным при использовании операций Git, которые делают это. Мы обсудим ситуации, когда не рекомендуется перебазировать ветку, в *разд. "Золотое правило перебазирования" далее в этой главе*. Однако сначала я собираюсь показать вам, как можно применить перебазирование в проекте Git, а вы отработаете эту процедуру на практическом примере в проекте Rainbow. Но сначала мы разберемся, зачем вообще может понадобиться перебазирование.

Зачем применяют перебазирование?

Для того чтобы разобраться, как перебазирование помогает избежать трехсторонних слияний и сохранить линейную историю проекта, давайте вернемся к приме-

ру 9.1 проекта Book. В этой главе мы продолжим работу с ним в примере 11.1 проекта Book.

Пример 11.1 проекта Book

Продолжая пример сценария из *главы 9*, предположим, что мы с соавтором создаем разные ветки на основе ветки `main`, когда она указывает на коммит B, и собираемся работать над разными главами. Я создаю ветку `chapter_five` (глава 5), а мой соавтор — ветку `chapter_six` (глава 6), как показано на рис. 11.1.

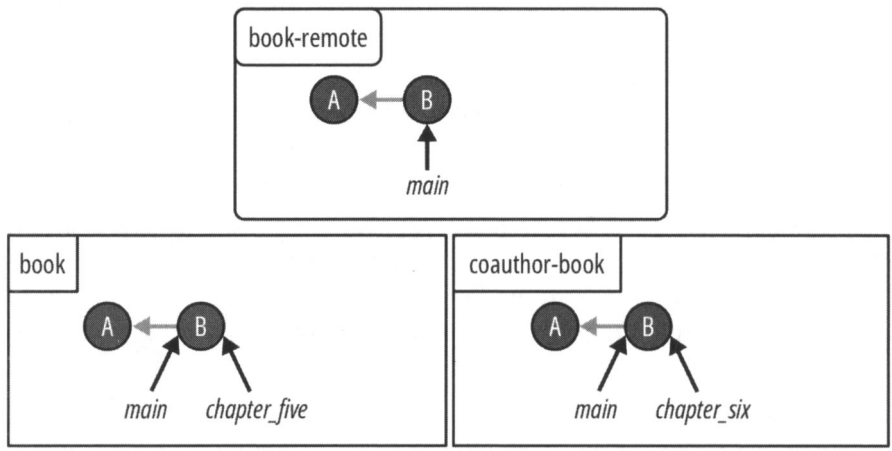


Рис. 11.1. Пример проекта Book, когда мы с соавтором создаем ветки для работы над разными главами

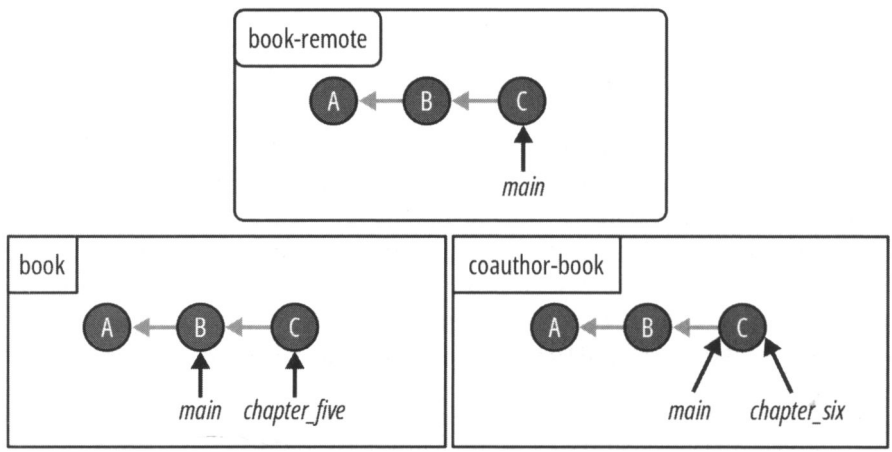


Рис. 11.2. Проект Book после того, как мой соавтор отправил свои изменения в удаленный репозиторий

Мой соавтор работает над главой 6 книги и делает коммит C. Он сливает свою работу с веткой `main` своего локального репозитория и отправляет обновленную ветку

`main` в удаленный репозиторий. В то же время я работаю над главой 5 и делаю коммит D в своем локальном репозитории. Однако я еще не успела сделать слияние и не поделилась своей работой. Эти изменения проиллюстрированы на рис. 11.2.

Когда я решу, что мои изменения в главе 5 готовы к слиянию с моей локальной веткой `main`, у меня появится выбор дальнейших действий. Один из вариантов — получить изменения удаленной ветки `main` из удаленного репозитория, а затем выполнить трехстороннее слияние, чтобы объединить `chapter_five` с `main`, что в конечном итоге приведет к появлению коммита слияния (обозначенного как коммит M на рис. 11.3). Этот вариант рассмотрен в *главе 9*.

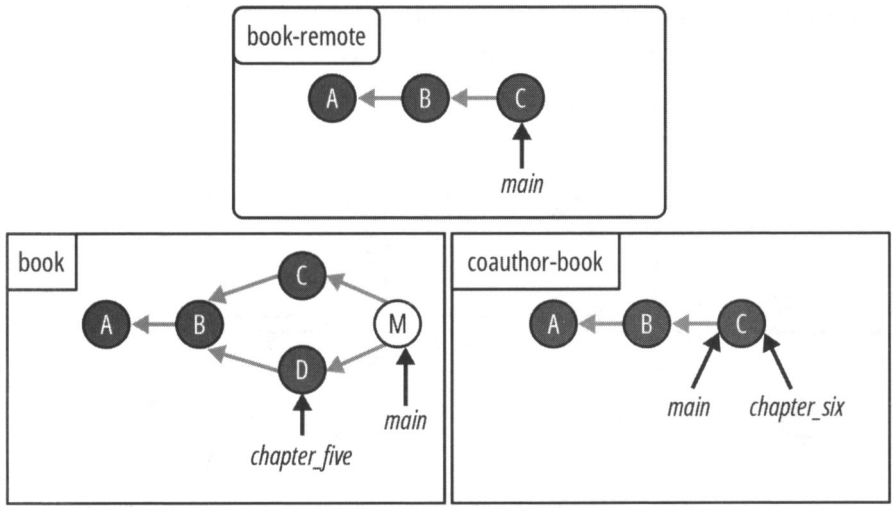


Рис. 11.3. Состояние проекта Book, если бы я интегрировала изменения из удаленного репозитория с помощью трехстороннего слияния

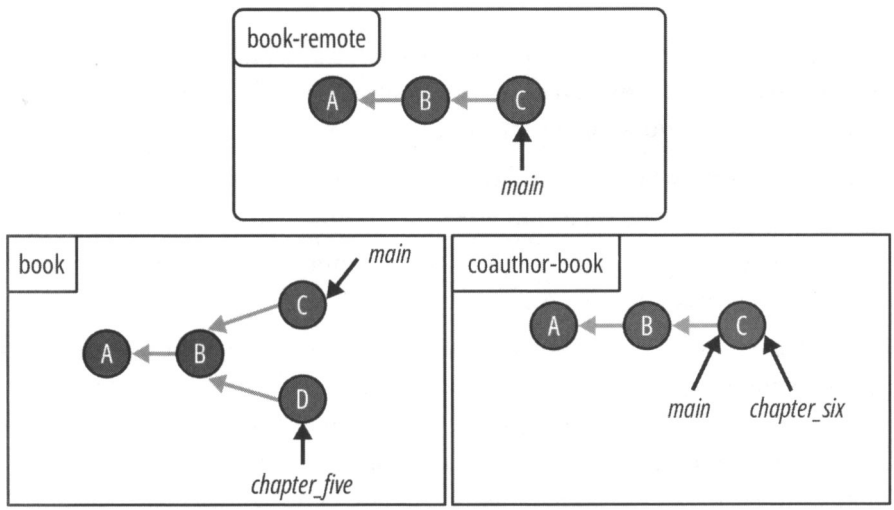


Рис. 11.4. Проект Book после получения изменений из удаленной основной ветки

Другой вариант — получить изменения из удаленного репозитория, вследствие чего моя локальная ветка `main` обновится до момента коммита C, с намерением впоследствии перебазировать мою ветку. На рис. 11.4 показано состояние проекта Book после того, как я получила изменения из удаленной ветки `main`.

Затем я могу перебазировать свою ветку `chapter_five` поверх обновленной ветки `main`. Как было сказано ранее, при перебазировании Git возьмет все коммиты, которые я сделала в ветке `chapter_five`, и повторно применит их к ветке `main`, создавая совершенно новые коммиты. В этом примере единственный коммит, который я сделала в ветке `chapter_five`, отличающий ее от ветки `main`, — это коммит D. При перебазировании ветки будет создан новый коммит D′ (рис. 11.5). Апостроф (′) в обозначении указывает, что это перебазированный коммит.

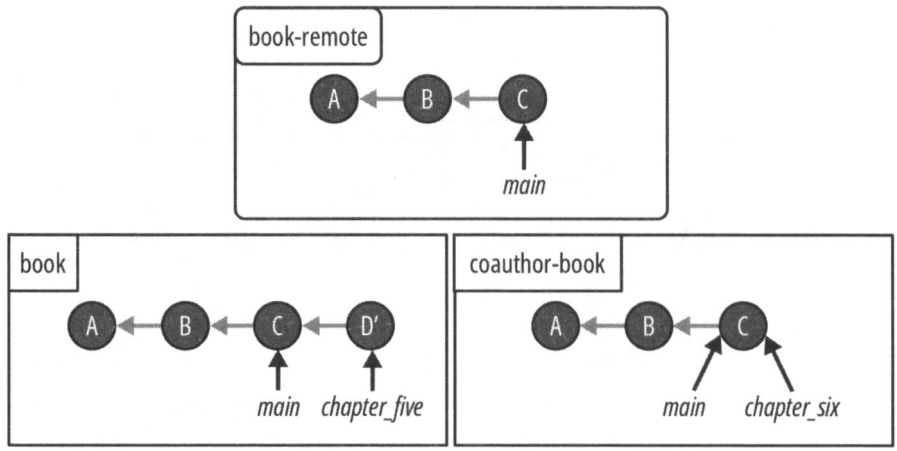


Рис. 11.5. Проект Book после того, как я интегрировала изменения из удаленного репозитория, перебазировав ветку `chapter_five` на ветку `main`

Коммит D′ представляет собой новую версию коммита D, созданную в процессе перебазирования. Он содержит все изменения, которые я внесла в исходный коммит D, но это совершенно новый коммит с новым хешем.

После перебазирования ветки `chapter_five` я могу объединить ее с веткой `main` с помощью простого ускоренного слияния (рис. 11.6).

Перебазируя свою ветку, я могу поддерживать линейную историю коммитов и избегать дополнительных коммитов слияния.

Обратите внимание, что исходный коммит D все еще существует в истории коммитов. В то же время он больше не является частью ни одной из веток репозитория `book`, поэтому он не показан на рис. 11.6. Перебазирование ветки не удаляет коммиты в этой ветке; оно просто пересоздает их. Старые коммиты все еще существуют в истории коммитов.

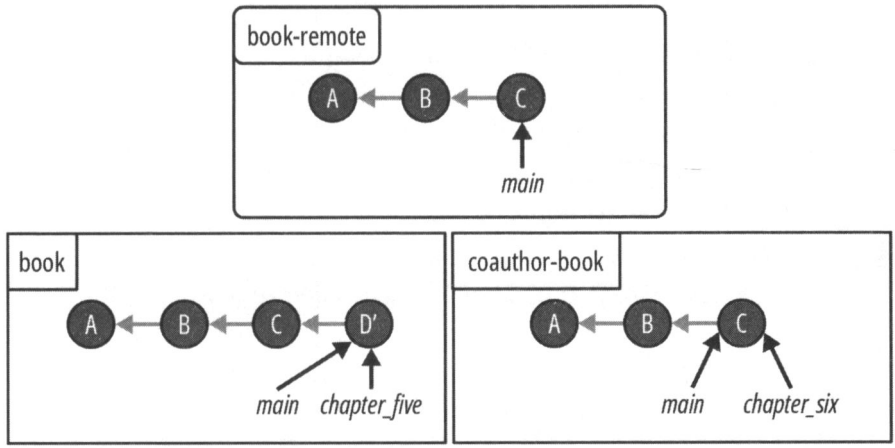


Рис. 11.6. Проект Book после объединения веток `chapter_five` и `main` с помощью простого ускоренного слияния

Мы разобрали гипотетический пример того, как перебазирование применяется для поддержания линейной истории проекта. Далее мы создадим в проекте Rainbow ситуацию, в которой у вас есть расходящиеся истории двух веток, и потренируемся выполнять перебазирование.

Подготовка примера перебазирования

Для того чтобы попрактиковаться в перебазировании, вам придется создать разные истории. Вы сделаете один коммит в репозитории `rainbow` и отправите его в удаленный репозиторий. Далее ваш друг, не получая изменений из удаленного репозитория, сделает два коммита в своем репозитории `friend-rainbow`. Затем он получит изменения из удаленного репозитория и решит перебазировать свою ветку.

Выполните действия 11.1, чтобы сделать коммит в репозитории `rainbow`.

ДЕЙСТВИЯ 11.1

1. Откройте файл othercolors.txt в каталоге проекта `rainbow` в окне текстового редактора. Добавьте в него текст "Серый — это не цвет радуги." в строке 2 и сохраните файл.
2. `rainbow $ git add othercolors.txt`
3. `rainbow $ git commit -m "gray"`

   ```
   [main 6f2cf36] gray
    1 file changed, 2 insertions(+), 1 deletion(-)
   ```
4. `rainbow $ git push`

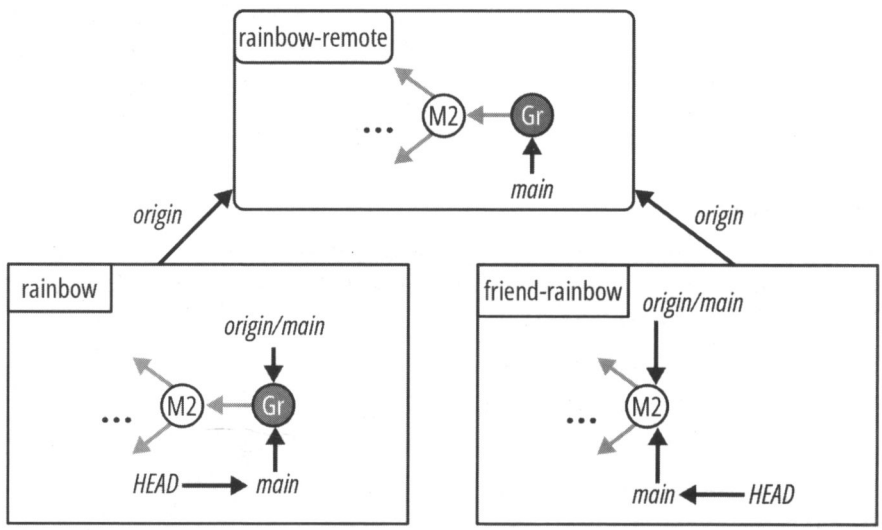
   ```
   Enumerating objects: 5, done.
   Counting objects: 100% (5/5), done.
   ```

```
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 327 bytes | 327.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:gitlearningjourney/rainbow-remote.git
   f10f972..6f2cf36 main -> main
```

5 `rainbow $ git log`

```
commit 6f2cf3698e6bf9078e8e0340ec9948f590405091 (HEAD -> main,
origin/main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 09:27:58 2022 +0100

    gray

commit f10f9725e3319af840a3d891ca8950436a219eb0
Merge: 6ad5c15 9b0a614
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 09:11:06 2022 +0100

    merge commit 2
```

ДИАГРАММА 11.3



Проект Rainbow после того, как вы сделали серый коммит в локальной ветке `main` репозитория `rainbow` и отправили его в удаленный репозиторий.

На что обратить внимание:

- вы сделали серый коммит и отправили его в удаленный репозиторий.

Текущее состояние проекта показано на диаграмме 11.3.

Затем ваш друг внесет некоторые изменения в репозиторий `friend-rainbow` в той же ветке, что приведет к расхождению историй между вашей веткой `main` и веткой `main` вашего друга. Эта ситуация также дает хорошую возможность познакомиться с некоторыми полезными функциями промежуточной области.

Удаление файлов из промежуточной области

В этом разделе ваш друг внесет изменения в оба файла в репозитории `friend-rainbow` и добавит их в промежуточную область. Но затем он спохватится, что хотел сделать два отдельных коммита для разных частей работы. Поэтому ему придется *отменить подготовку* (unstage) файла в промежуточной области.

Давайте взглянем на пример 11.2 проекта Book, в котором мы вернемся к сценарию, предложенному в примере 3.2 проекта Book, чтобы понять, почему нам может потребоваться отменить подготовку файла.

Пример 11.2 проекта Book

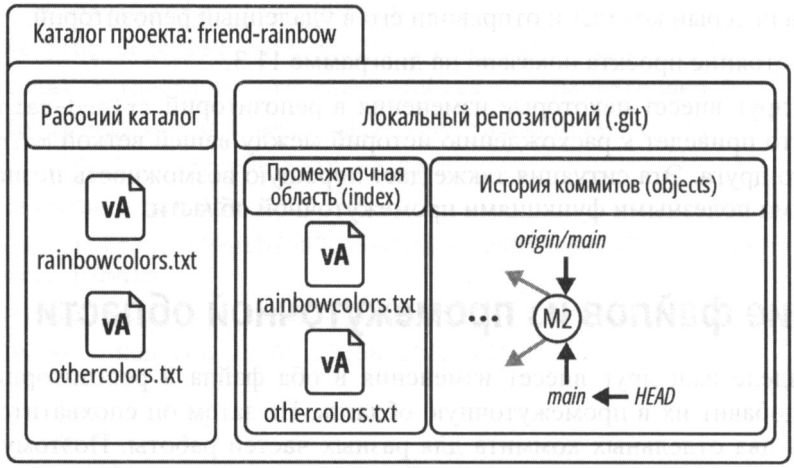
Предположим, я работаю над главами 1, 2 и 3 своей книги, редактируя файлы, соответствующие этим главам: chapter_one.txt, chapter_two.txt и chapter_three.txt. Я добавляю все файлы глав в промежуточную область, и это означает, что все изменения, которые я внесла в главы 1, 2 и 3, будут сохранены в следующем коммите. Я планирую, что мой редактор проверит все мои изменения.

Однако прежде чем сделать коммит, я внезапно прихожу к выводу, что на самом деле к проверке редактором готовы только изменения из главы 2. Другими словами, мне нужны лишь изменения, сделанные в файле chapter_two.txt, который будет включен в мой следующий коммит. Поскольку промежуточная область представляет собой своего рода черновик, где я могу добавлять и удалять измененные файлы, чтобы сформировать следующий коммит, я могу удалить измененные файлы chapter_one.txt и chapter_three.txt из этой области, так что в мой следующий коммит будут включены только изменения в файле chapter_two.txt. Это означает, что мой редактор будет просматривать лишь эти изменения.

Теперь, когда у вас есть представление о том, почему так важно уметь удалять файлы из промежуточной области, вы готовы попрактиковаться в удалении файла из промежуточной области проекта Rainbow.

Для того чтобы лучше понять, что происходит в следующем примере, мы сосредоточимся на репозитории `friend-rainbow` и диаграммах состояния Git в знакомой вам форме из *главы 2*. Текущее состояние репозитория `friend-rainbow` показано на диаграмме 11.4.

ДИАГРАММА 11.4



Каталог проекта `friend-rainbow` до того, как ваш друг отредактировал какие-либо файлы.

На что обратить внимание:

- текущие версии файлов rainbowcolors.txt и othercolors.txt, находящиеся в рабочем каталоге и промежуточной области, представлены как версия A (vA).

Далее ваш друг собирается внести некоторые изменения в файлы. Для этого нужно от его имени выполнить действия 11.2.

ДЕЙСТВИЯ 11.2

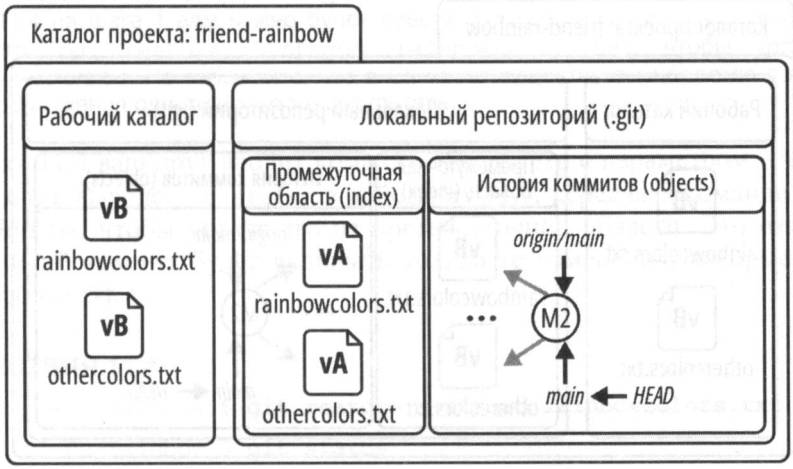
1. В каталоге проекта `friend-rainbow` в текстовом редакторе внесите следующие изменения:
   - в файле othercolors.txt добавьте текст "Черный не является цветом радуги." в строке 2 и сохраните файл;
   - в файле rainbowcolors.txt добавьте текст "Это все цвета радуги." в строке 8 и сохраните файл.
2.
```
friend-rainbow $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
directory)
    modified: othercolors.txt
    modified: rainbowcolors.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

На что обратить внимание:

- ваш друг отредактировал файлы rainbowcolors.txt и othercolors.txt, и они указаны как измененные файлы, которые не были добавлены в промежуточную область.

Обновленное состояние показано на диаграмме 11.5.

ДИАГРАММА 11.5



Каталог проекта `friend-rainbow` после того, как ваш друг изменил файлы rainbowcolors.txt и othercolors.txt в рабочем каталоге.

На что обратить внимание:

- измененные версии файлов rainbowcolors.txt и othercolors.txt, которые редактировал ваш друг, представлены как версия B (vB);
- версии файлов othercolors.txt и rainbowcolors.txt в промежуточной области (vA) отличаются от версий в рабочем каталоге (vB).

Далее, выполнив действия 11.3, ваш друг добавит обновленные файлы в промежуточную область, чтобы включить их в следующий коммит.

ДЕЙСТВИЯ 11.3

1. `friend-rainbow $ git add rainbowcolors.txt othercolors.txt`
2. `friend-rainbow $ git status`

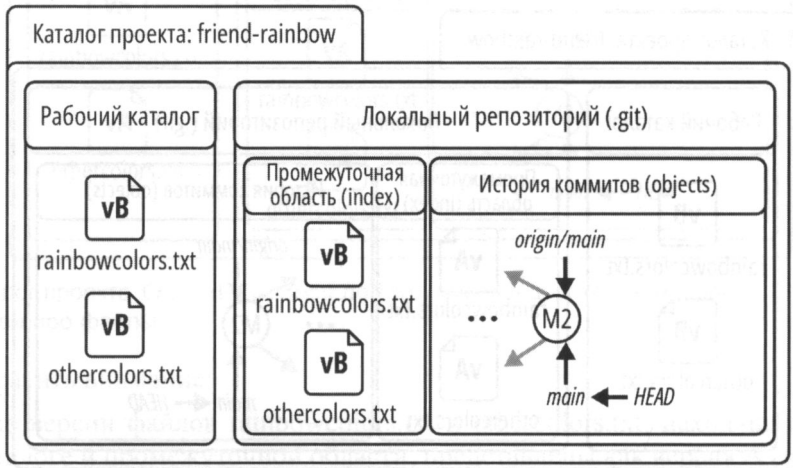
```
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   othercolors.txt
    modified:   rainbowcolors.txt
```

На что обратить внимание:

- ваш друг добавил измененные файлы rainbowcolors.txt и othercolors.txt в промежуточную область.

Это состояние показано на диаграмме 11.6.

ДИАГРАММА 11.6



Каталог проекта `friend-rainbow` после того, как ваш друг добавил обновленные версии файлов rainbowcolors.txt и othercolors.txt в промежуточную область.

На что обратить внимание:

- версии файлов othercolors.txt и rainbowcolors.txt в промежуточной области изменились с vA на vB.

Теперь предположим, что ваш друг решил создать *отдельные* коммиты для работы, которую он проделал, добавив черный цвет в список нерадужных цветов, и для работы, которую он проделал с файлом цветов радуги.

В *разд. "Знакомство с промежуточной областью" главы 2* я упоминала, что промежуточная область похожа на черновик, где вы можете добавлять и удалять измененные файлы, чтобы сформировать окончательное содержимое следующего коммита. До сих пор вы использовали команду `git add` для добавления файлов в промежуточную область. Теперь вы увидите, как удалить измененный файл из промежуточной области или, другими словами, изменить версию файла в промежуточной области. В выводе команды `git status` в перечне действий 11.3 Git предоставляет инструкции по удалению файла из промежуточной области: `(use "git restore --staged <file>..." to unstage)` (для удаления из промежуточной области выполните "git restore --staged <файл>..."). Как следует из подсказки, для удаления файлов из промежуточной области применяется команда `git restore` с опцией `--staged`, которой передают разделенные пробелами имена файлов, подлежащих удалению из промежуточной области.

ЗАПОМНИТЕ КОМАНДУ

git restore --staged <имя_файла>

Восстанавливает файл в промежуточной области до предыдущей версии.

ПРИМЕЧАНИЕ

Если у вас версия Git старше версии 2.23, вам будет недоступна команда `git restore`. В выводе Git может быть предложено использовать команду `git reset`, которая тоже позволяет удалить файл из резервной копии. При выполнении действий 11.4 на шаге 1 вам нужно будет ввести `git reset HEAD rainbowcolors.txt` вместо `git restore --staged rainbowcolors.txt`, чтобы удалить файл rainbowcolors.txt. Не забывайте, что в Git часто существует множество разных способов достижения одного и того же результата.

В действиях 11.4 ваш друг вводит команду `git resore` с параметром `--staged`, как было указано в выводе `git status` в действиях 11.3, передавая команде имя файла rainbowcolors.txt, чтобы убрать его из промежуточной области. Это означает, что его следующий коммит будет включать только те изменения, которые он внес в файл othercolors.txt.

ДЕЙСТВИЯ 11.4

1. `friend-rainbow $ **git restore --staged rainbowcolors.txt**`
2. `friend-rainbow $ **git status**`

```
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified: othercolors.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <файл>..." to discard changes in working
directory)
    modified: rainbowcolors.txt
```

(Изменения, которые будут в коммите:

(введите "git restore --staged <файл>..." для отмены размещения)

...

Изменения, которые не попадут в коммит:

(введите "git add <file>..." для обновления состава коммита)

(введите "git restore <файл>..." для отмены изменений в рабочем каталоге)

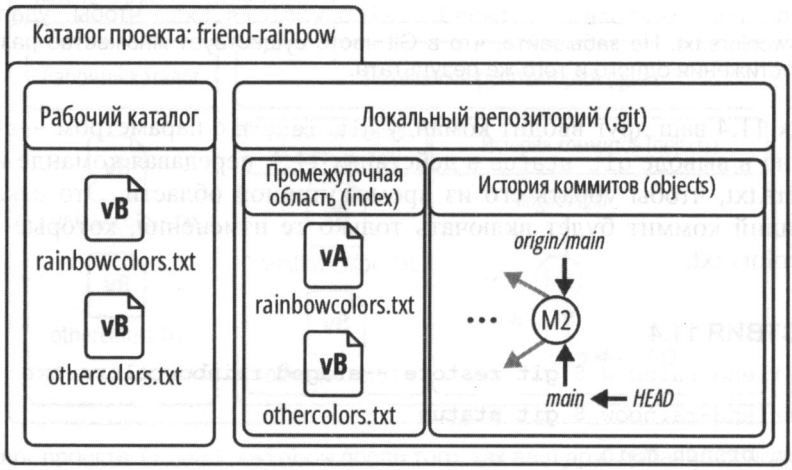
...)

На что обратить внимание:

- вывод `git status` показывает, что обновленная версия файла othercolors.txt все еще находится в промежуточной области, в то время как обновленная версия файла rainbowcolors.txt удалена оттуда.

Новое положение дел отражено на диаграмме 11.7.

ДИАГРАММА 11.7



Каталог проекта `friend-rainbow` после того, как ваш друг отменил размещение файла rainbowcolors.txt в промежуточной области.

На что обратить внимание:

- ваш друг удалил файл rainbowcolors.txt, поэтому версия файла в промежуточной области вернулась к vA;
- обновленная версия othercolors.txt (vB) все еще находится в состоянии разработки;
- версия rainbowcolors.txt с последними изменениями вашего друга (vB) все еще находится в рабочем каталоге.

Далее, выполняя действия 11.5, ваш друг сделает коммит, включающий в себя только изменения в файле othercolors.txt.

ДЕЙСТВИЯ 11.5

1
```
friend-rainbow $ git commit -m "black"
[main 29bdadd] black
 1 file changed, 2 insertions(+), 1 deletion(-)
```

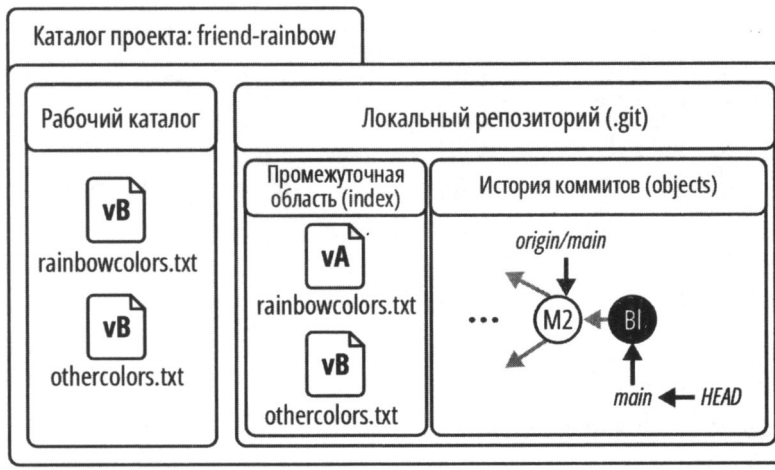
2
```
friend-rainbow $ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)
```

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
directory)
    modified: rainbowcolors.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

3 `friend-rainbow $ git log`

```
commit 29bdadd50ddea41c75b476e776b6204a555b3d54 (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:07:38 2022 +0100

    black

commit f10f9725e3319af840a3d891ca8950436a219eb0 (origin/main,
origin/HEAD)
Merge: 6ad5c15 9b0a614
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 09:11:06 2022 +0100

    merge commit 2
```

ДИАГРАММА 11.8



Каталог проекта `friend-rainbow` после того, как ваш друг сделал черный коммит.

На что обратить внимание:

- ваш друг создал черный коммит;

- на шаге 2 в выводе `git status` упоминается, что файл rainbowcolors.txt является измененным файлом в рабочем каталоге.

Диаграмма 11.8 показывает состояние каталога проекта `friend-rainbow` после выполнения действий 11.5.

Теперь ваш друг выполнит действия 11.6, чтобы добавить изменения, внесенные им в файл rainbowcolors.txt, в промежуточную область и сделает еще один коммит.

ДЕЙСТВИЯ 11.6

1 `friend-rainbow $ git add rainbowcolors.txt`

2
```
friend-rainbow $ git commit -m "rainbow"
[main 51dc6ec] rainbow
 1 file changed, 1 insertion(+), 1 deletion(-)
```

3
```
friend-rainbow $ git log
commit 51dc6ecb327578cca503abba4a56e8c18f3835e1 (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:10:11 2022 +0100

    rainbow

commit 29bdadd50ddea41c75b476e776b6204a555b3d54
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:07:38 2022 +0100

    black

commit f10f9725e3319af840a3d891ca8950436a219eb0 (origin/main,
origin/HEAD)
Merge: 6ad5c15 9b0a614
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 09:11:06 2022 +0100

    merge commit 2
```

На что обратить внимание:

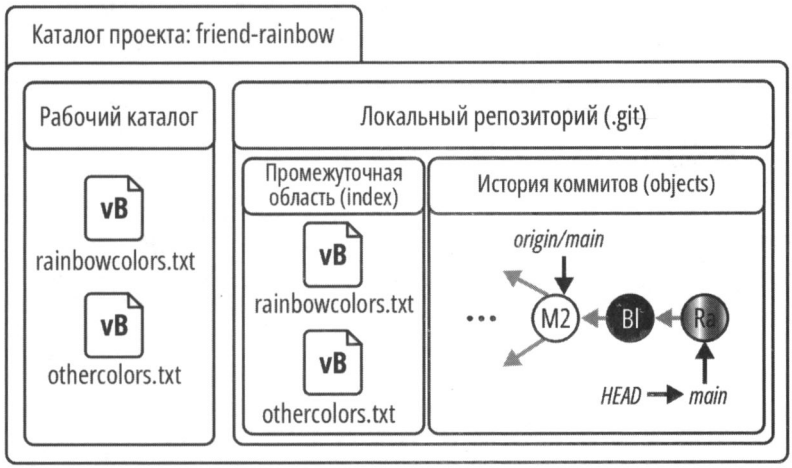
- ваш друг сделал радужный коммит.

Результат этого действия показан на диаграмме 11.9.

На что обратить внимание:

- у файла rainbowcolors.txt в промежуточной области теперь версия vB, которая является частью радужного коммита.

ДИАГРАММА 11.9



Каталог проекта `friend-rainbow` после того, как ваш друг добавил файл rainbowcolors.txt в промежуточную область и сделал коммит.

Вы только что наблюдали, как ваш друг добавлял файлы и удалял их из промежуточной области, чтобы создать именно те коммиты, которые он хотел. Локальная ветка `main` в репозитории `rainbow` и локальная ветка `main` в репозитории `friend-rainbow` теперь имеют разную историю развития. Прежде чем ваш друг продолжит пример с перебазированием, ему необходимо получить всю работу, которую вы отправили в удаленную ветку `main`, из удаленного репозитория.

Подготовка к перебазированию

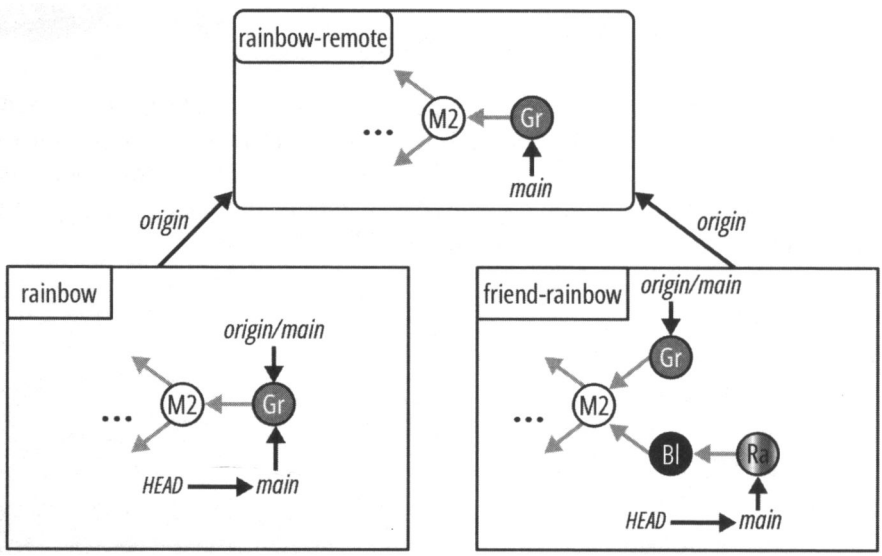
Как вы видели ранее в этой главе в примере 11.1 проекта Book, чтобы перебазировать свою ветку, вам сначала нужно получить всю работу, проделанную в той ветке, на которую вы хотите перебазироваться. Итак, готовясь к перебазированию своей ветки, ваш друг намеревается получить последние обновления из удаленного репозитория. Для этого он выполняет действия 11.7.

ДЕЙСТВИЯ 11.7

1
```
friend-rainbow $ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 307 bytes | 153.00 KiB/s, done.
From github.com:gitlearningjourney/rainbow-remote
f10f972..6f2cf36 main -> origin/main
```

2
```
friend-rainbow $ git log --all
commit 51dc6ecb327578cca503abba4a56e8c18f3835e1 (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:10:11 2022 +0100

    rainbow

commit 29bdadd50ddea41c75b476e776b6204a555b3d54
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:07:38 2022 +0100

    black

commit 6f2cf3698e6bf9078e8e0340ec9948f590405091 (origin/main,
origin/HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 09:27:58 2022 +0100

    gray
```

ДИАГРАММА 11.10



Проект Rainbow после того, как ваш друг обновил репозиторий `friend-rainbow`, получив вашу работу из удаленного репозитория.

На что обратить внимание:

- ваш друг получил серый коммит из удаленного репозитория, и ветка удаленного отслеживания `origin/main` была обновлена, чтобы указывать на нее.

Эти события показаны на диаграмме 11.10.

Ветка удаленного отслеживания `origin/main` в репозитории `friend-rainbow` представляет собой последнюю версию удаленной ветки `main`. Теперь ваш друг готов перебазировать свою локальную ветку `main` на ветку удаленного отслеживания `origin/main`. Далее мы рассмотрим этапы самого процесса перебазирования.

Пять этапов процесса перебазирования

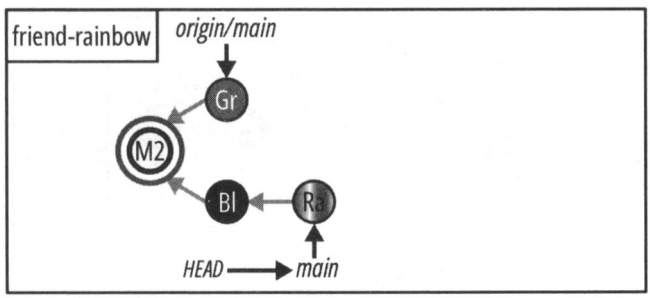
В этом разделе вы пройдете пять этапов процесса перебазирования. Для того чтобы наглядно представить этапы, я покажу диаграммы состояния проекта, демонстрирующие практический пример, который вы будете выполнять позже в этой главе, когда ваш друг перебазирует свою локальную ветку `main` в репозитории `friend-rainbow` на ветку удаленного отслеживания `origin/main`.

Для того чтобы инициировать процесс перебазирования, вы воспользуетесь командой `git rebase`. Затем Git выполнит пять этапов самого процесса; единственная причина, по которой вам может понадобиться вмешаться, — если возникнут какие-либо конфликты слияния. Мы обсудим эту ситуацию более подробно немного позже в *разд. "Конфликты перебазирования и слияния" далее в той главе.*

Этап 1: поиск общего предка

На первом этапе процесса Git определит общего предка двух веток, участвующих в перебазировании: ветки, в которой вы находитесь, и ветки, на которую вы перебазируетесь.

ДИАГРАММА 11.11



Этап 1: нахождение общего предка веток, участвующих в перебазировании (здесь — коммит слияния M2).

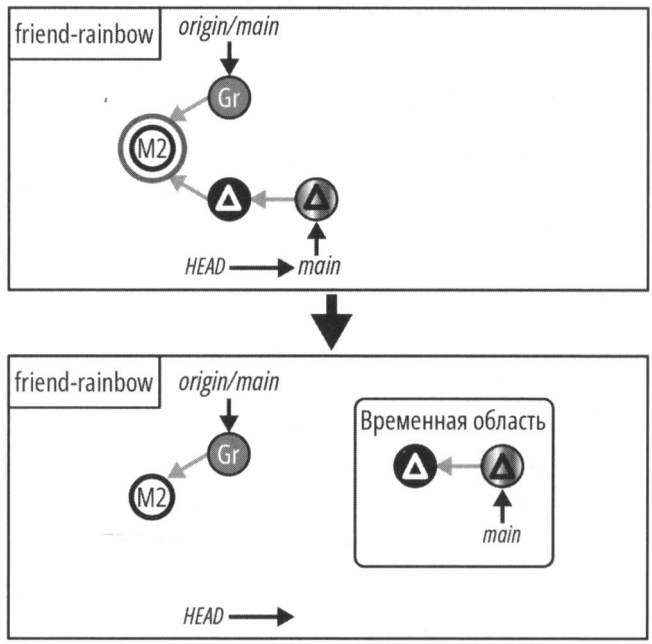
В примере проекта Rainbow ветка вашего друга находится на локальной ветке `main` в репозитории `friend-rainbow`, а перебазироваться он будет на ветку удаленного отслеживания `origin/main`. Общим предком будет коммит слияния M2, как показано на диаграмме 11.11.

Этап 2: сохранение информации о ветках, участвующих в перебазировании

На этапе 2 Git сохранит изменения, внесенные каждым коммитом ветки, в которой вы находитесь, во временную область. В этой временной области также будет сохранена дополнительная информация, например, о том, в какую ветку вы выполняете перебазирование и куда она указывала, когда вы инициировали перебазирование.

В примере проекта Rainbow изменения, внесенные черным и радужным коммитами, будут сохранены во временной области вместе с информацией об удаленной ветке `main`, как показано на диаграмме 11.12.

ДИАГРАММА 11.12



Этап 2: сохранение информации о перебазировании во временной области.

ПРИМЕЧАНИЕ

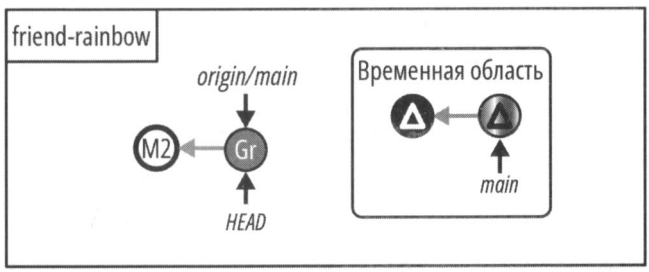
На следующих диаграммах треугольники (△) используются для обозначения изменений, внесенных коммитами.

Этап 3: сброс указателя *HEAD*

На этапе 3 Git сбросит `HEAD`, чтобы он указывал на тот же коммит, что и ветка, в которую вы перебазируетесь.

В примере проекта Rainbow он сбросит `HEAD` на тот же коммит, на который указывает ветка удаленного отслеживания `origin/main`, т. е. на серый коммит, как показано на диаграмме 11.13.

ДИАГРАММА 11.13



Этап 3: сброс `HEAD` на тот же коммит, что и ветка, на которую вы перебазируетесь (здесь — серый коммит).

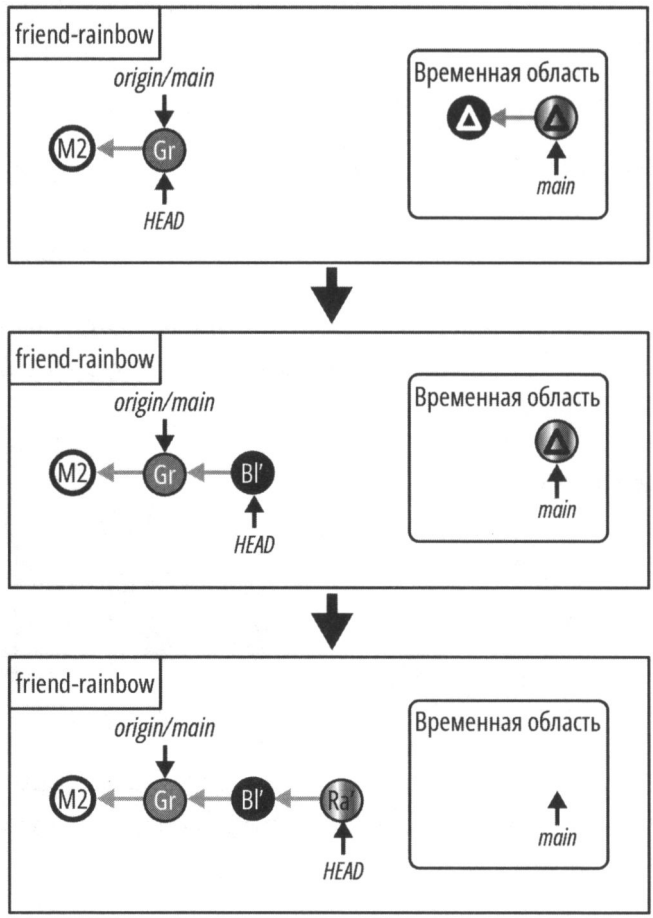
Этап 4: применение изменений и выполнение коммита

На этапе 4 Git по очереди применит набор изменений из каждого коммита, делая новый коммит после применения каждого набора.

В примере проекта Rainbow сначала применяются изменения, внесенные черным коммитом, и создается новый коммит, а затем применяются изменения, внесенные радужным коммитом, и создается еще один новый коммит, как показано на диаграмме 11.14.

На диаграмме 11.14 новые черный и радужный коммиты представлены, соответственно, как Bl′ и Ra′ (с апострофом), поскольку это совершенно новые коммиты.

ДИАГРАММА 11.14



Этап 4: применение изменений и выполнение соответствующих новых коммитов (здесь — черный и радужный коммиты).

Этап 5: переключение на перебазированную ветку

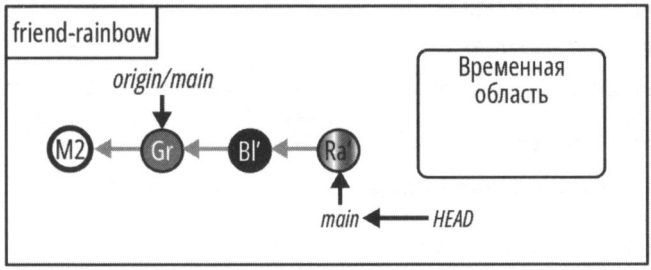
На этапе 5 Git сделает так, чтобы ветка, которую вы перебазировали, указывала на последний коммит, который он повторно применял, и проверит эту ветку, чтобы `HEAD` указывал на нее.

В примере проекта Rainbow ветка `main` будет указывать на новый радужный коммит, как показано на диаграмме 11.15.

На этом мы завершаем рассмотрение пяти этапов процесса перебазирования. Как было сказано в начале этого раздела, Git самостоятельно выполняет весь процесс; все, что вам нужно сделать, — это запустить его с помощью команды `git rebase`. Я также отметила, что единственная ситуация, когда вам нужно участвовать в про-

цессе перебазирования, — это если Git столкнется с конфликтами слияния. Мы кратко рассмотрим данный сценарий в следующем разделе.

ДИАГРАММА 11.15



Этап 5: переключение на ветку, на которую вы перебазировались.

Конфликты перебазирования и слияния

В *главе 10* вы узнали о конфликтах, которые возникают при слиянии двух веток, в которых в одни и те же файлы были внесены разные изменения, или если в одной ветке был удален файл, который редактировался в другой ветке.

Git выполнит весь процесс перебазирования самостоятельно, если только не столкнется с конфликтами слияния. В таком случае вы должны вмешаться и разрешить их. Процесс разрешения конфликтов слияния при перебазировании аналогичен процессу трехстороннего слияния с некоторыми небольшими отличиями.

При разрешении конфликтов в трехстороннем слиянии все конфликты слияния отображаются одновременно; как только вы разрешите все конфликты и добавите все обновленные файлы в промежуточную область, вы сделаете окончательный коммит слияния. Напротив, в процессе перебазирования, поскольку Git применяет изменения из каждого коммита одно за другим, он приостанавливает процесс каждый раз, когда сталкивается с конфликтами слияния в любом повторно примененном коммите. Это означает, что при перебазировании вам, возможно, придется разрешать конфликты слияния несколько раз, в зависимости от того, сколько коммитов содержат конфликты слияния.

После разрешения конфликтов слияния в конкретном коммите вам нужно добавить обновленные файлы в промежуточную область, а затем дать указание Git возобновить процесс перебазирования, введя команду `git rebase` с опцией `--continue`. Git продолжит перебазирование остальных коммитов. Вам не нужно явно делать какие-либо коммиты, как это происходит при трехстороннем слиянии с конфликтами.

Как и в случае со слиянием, если в какой-либо момент процесса разрешения конфликтов слияния при перебазировании вы решите, что не хотите продолжать про-

цесс перебазирования, вы можете остановить или прервать процесс, используя команду `git rebase` с опцией `--abort`. Это вернет все ваши файлы в состояние, в котором они находились до перебазирования.

ЗАПОМНИТЕ КОМАНДУ

`git rebase --continue`

Продолжает процесс перебазирования после разрешения конфликтов слияния.

`git rebase --abort`

Останавливает процесс перебазирования и возвращает файлы в состояние до перебазирования.

Теперь, когда вы узнали, чего ожидать при перебазировании, пришло время попрактиковаться с проектом Rainbow.

Перебазирование ветки на практике

От имени вашего друга выполните действия 11.8, где он перебазирует ветку `main` в своем локальном репозитории на ветку удаленного отслеживания `origin/main`.

ДЕЙСТВИЯ 11.8

1. В каталоге проекта `friend-rainbow` запишите хеши коммитов для черного и радужного коммитов. Для этого пригодится команда `git log`, которая добавляет в свои выходные данные хеши коммитов. В примере из этой книги хеши коммитов такие:

   черный (black): `29bdadd50ddea41c75b476e776b6204a555b3d54`

   радужный (rainbow): `51dc6ecb327578cca503abba4a56e8c18f3835e1`

2. ```
   friend-rainbow $ git rebase origin/main
   Auto-merging othercolors.txt
   CONFLICT (content): Merge conflict in othercolors.txt
   error: could not apply 29bdadd... black
   hint: Resolve all conflicts manually, mark them as resolved with
   hint: "git add/rm <conflicted_files>", then run "git rebase --
   continue".
   hint: You can instead skip this commit: run "git rebase --skip".
   hint: To abort and get back to the state before "git rebase",
   run "git rebase --abort".
   Could not apply 29bdadd... black
   ```
   ```

(КОНФЛИКТ (содержимое): конфликт слияния в othercolors.txt

ошибка: невозможно применить 29bdadd... black

подсказка: разрешите все конфликты вручную, пометьте их как разрешенные командой "git add/rm <конфликтующие_файлы>", затем выполните "git rebase --continue". Вместо этого вы можете пропустить этот коммит: "git rebase --skip". Для прерывания операции и возврата к исходному состоянию выполните команду "git rebase --abort")

3
```
friend-rainbow $ git status
interactive rebase in progress; onto 6f2cf36
Last command done (1 command done):
pick 29bdadd black
Next command to do (1 remaining command):
   pick 51dc6ec rainbow
  (use "git rebase --edit-todo" to view and edit)
You are currently rebasing branch 'main' on 'bcb1dc0'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
        both modified: othercolors.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

(...

Выполняется перебазирование ветки 'main' на 'bcb1dc0'.

(устраните конфликты и выполните "git rebase --continue")

(выполните "git rebase --skip", чтобы пропустить изменение)

(выполните "git rebase --abort", чтобы вернуться к исходной ветке)

Не объединены пути:

(выполните "git restore --staged <файл>..." для удаления файла из промежуточной области)

(выполните "git add <file>...", чтобы отметить решения)

изменены оба: othercolors.txt

В коммит не внесены изменения (выполните "git add" and/or "git commit -a")

На что обратить внимание:

- операция перебазирования была прервана, поскольку при применении изменений, включенных в черный коммит, Git столкнулся с конфликтом слияния;
- на шаге 2 команда `git rebase` подсказывает вам, что необходимо разрешить все конфликты вручную, пометив их как разрешенные с помощью `git add/rm <конфликтующие_файлы>`, затем запустить `git rebase --continue`;
- на шаге 3 команда `git status` также показывает информацию о перебазировании и о том, в каких файлах необходимо разрешить конфликты слияния.

Далее ваш друг выполнит действия 11.9, чтобы применить шаги, о которых вы узнали в *главе 10*, для разрешения конфликтов слияния. Затем он выполнит команду `git rebase --continue`.

ДЕЙСТВИЯ 11.9

1. Выполните шаг 1 разрешения конфликтов слияния, т. е. выберите, что именно следует сохранить, отредактируйте содержимое и удалите маркеры конфликта. Вам необходимо сохранить все изменения из обеих веток. Оставьте предложение "Серый не является цветом радуги." выше предложения "Черный не является цветом радуги.". Обязательно сохраните файл othercolors.txt, когда закончите вносить изменения.

2. ```
   friend-rainbow $ git add othercolors.txt
   ```

3. ```
   friend-rainbow $ git status
   interactive rebase in progress; onto 6f2cf36
   Last command done (1 command done):
      pick 29bdadd black
   Next command to do (1 remaining command):
      pick 51dc6ec rainbow
     (use "git rebase --edit-todo" to view and edit)
   You are currently rebasing branch 'main' on 'bcb1dc0'.
     (all conflicts fixed: run "git rebase --continue")

   Changes to be committed:
     (use "git restore --staged <file>..." to unstage)
           modified: othercolors.txt
   ```

4. ```
   friend-rainbow $ git rebase --continue
   ```
   ```

5 Примите сообщение коммита по умолчанию в редакторе Vim и выйдите из редактора. Как вы узнали из *главы 9*, для этого необходимо нажать клавишу <Esc>, ввести **:wq** и нажать <Enter>.

Вы увидите следующий результат:

```
[detached HEAD e055f2b] black
 1 file changed, 3 insertions(+), 1 deletion(-)
Successfully rebased and updated refs/heads/main.
```

(Ветка refs/heads/main успешно перебазирована и обновлена.)

6
```
friend-rainbow $ git log
commit 7c09136bcbfdd9f638ed13c6653e06451579d21c (HEAD -> main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:10:11 2022 +0100

    rainbow

commit e055f2bc66aed1f3627041900a8c825c7a875206
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:07:38 2022 +0100

    black

commit 6f2cf3698e6bf9078e8e0340ec9948f590405091 (origin/main,
origin/HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 09:27:58 2022 +0100

    gray
```

На что обратить внимание:

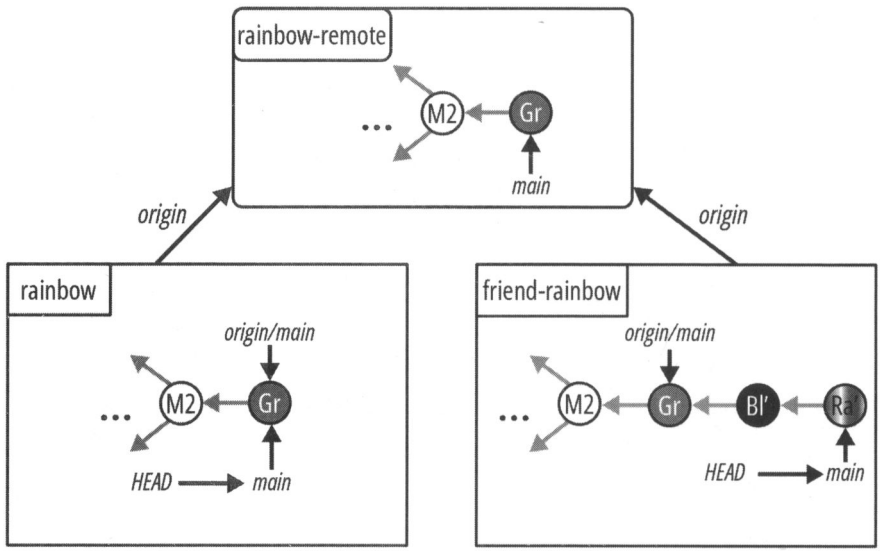
- на шаге 3 вывод `git status` сообщит вам, что все конфликты устранены и можно продолжить: `all conflicts fixed: run "git rebase --continue"`;
- на шаге 6 выходные данные `git log` показывают, что процесс перебазирования создал новые коммиты — радужный и черный. Хеши коммитов новые, как и метки времени (дата и время каждого коммита).

Результат этого процесса показан на диаграмме 11.16.

На что обратить внимание:

- новые черный и радужный коммиты обозначены как Bl′ и Ra′ (с апострофом);
- в репозитории `friend-rainbow` серый коммит, новый черный коммит и новый радужный коммит образуют линейную историю проекта. Другими словами, коммита слияния нет.

ДИАГРАММА 11.16



Проект Rainbow после того, как ваш друг перебазирует свою локальную ветку `main` на ветку удаленного отслеживания `origin/main`.

На рис. 11.7 сравниваются хеши старых и новых коммитов, черного и радужного, из этой книги. Хеши коммитов в ваших репозиториях будут отличаться от указанных в этой книге, поскольку хеши коммитов уникальны.

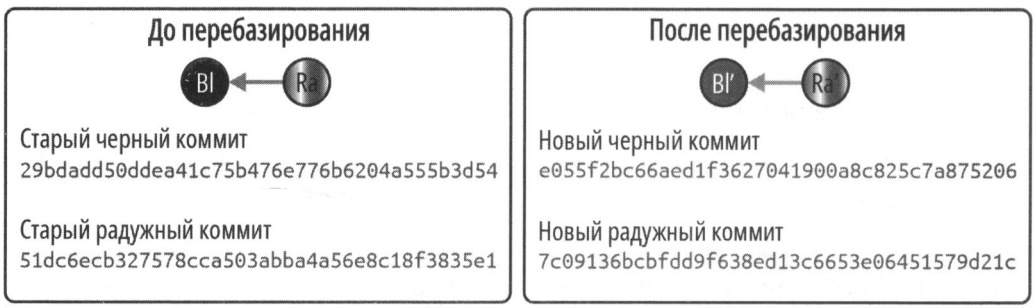


Рис. 11.7. Хеши старых коммитов, черного и радужного, отличаются от хешей новых коммитов

Из этого упражнения вы узнали, что перебазирование переписывает историю, и это подводит нас к золотому правилу перебазирования.

Золотое правило перебазирования

Как вы видели в примере проекта Rainbow, при перебазировании создаются совершенно новые коммиты. Это означает, что перебазирование меняет историю коммитов. Вы всегда должны быть осторожны при изменении истории коммитов, поскольку это может привести к осложнениям в вашем проекте, особенно когда вы работаете с другими людьми.

Золотое правило перебазирования гласит, что *вы не должны перебазировать ветку, на которой могли основываться работы других людей*. Например, если вы отправили ветку в удаленный репозиторий, она считается публичной веткой. Это означает, что другие соавторы также могут работать над этой веткой в своих локальных репозиториях или переносить работу в эту ветку в удаленном репозитории.

В этом случае следует воздержаться от перебазирования ветки. Для того чтобы понять важность золотого правила перебазирования, давайте взглянем на пример 11.3 проекта Book, в котором мы вернемся к ситуации, обсуждавшейся в примере 11.1 проекта Book.

Пример 11.3 проекта Book

В примере 11.1 проекта Book я продемонстрировала ситуацию, когда в ветке `main` было два коммита (A и B), когда я создала ветку `chapter_five`, а мой соавтор создал ветку `chapter_six`. Впоследствии мой соавтор объединил коммит C из своей ветки `chapter_six` в свою ветку `main` и отправил обновленную ветку `main` в удаленный репозиторий. В то же время я добавила коммит D в ветку `chapter_five` моего локального репозитория. Этот первоначальный сценарий показан на рис. 11.8.

Теперь предположим, что я помещаю ветку `chapter_five` в удаленный репозиторий, а затем, в нарушение золотого правила перебазирования, я также извлекаю коммит C из удаленного репозитория и обновляю свою локальную ветку `main`, потому что планирую перебазировать свою ветку `chapter_five` на последнюю версию ветки `main`. Эта ситуация изображена на рис. 11.9.

Я не осознаю своей ошибки и реализую план по перебазированию ветки `chapter_five` в ветку `main`. На рис. 11.10 показано состояние удаленного и локального репозиториев после перебазирования. Коммит D′ в репозитории `book` представляет собой повторно созданный коммит D после перебазирования.

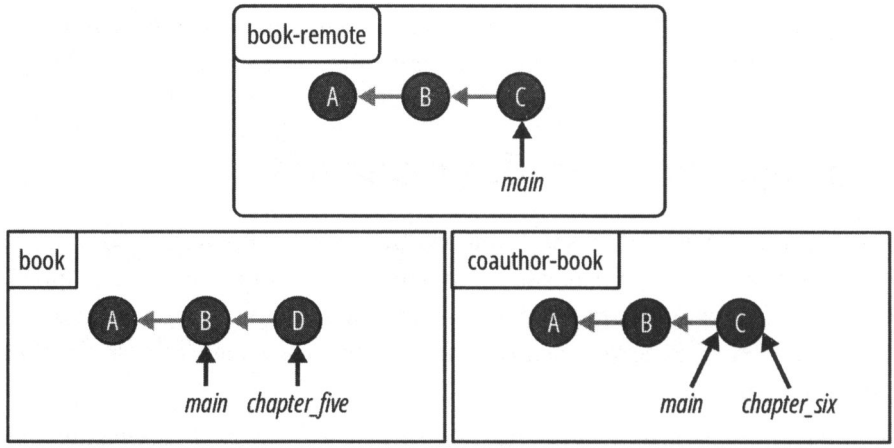


Рис. 11.8. Состояние проекта Book, когда локальная ветка `chapter_five` и удаленная ветка `main` разошлись

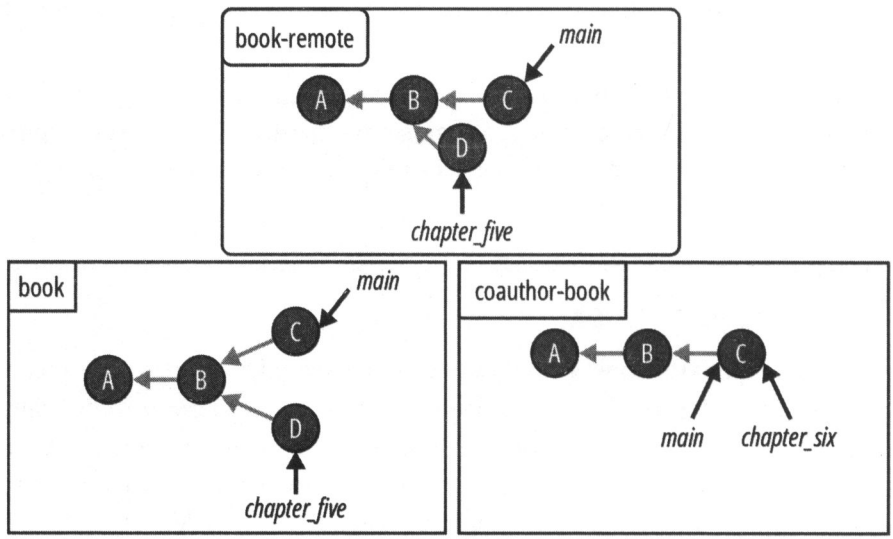


Рис. 11.9. Проект Book после того, как я отправила ветку `chapter_five` в удаленный репозиторий и получила коммиты в ветке `main`

Я еще не отправила обновленную ветку `chapter_five` в удаленный репозиторий. Теперь предположим, что мой соавтор видит удаленную ветку `chapter_five` в удаленном репозитории и решает внести в нее свой вклад. Он загружает ветку `chapter_five` в свой соавторский репозиторий `coauthor-book`, добавляет в нее коммиты E и F, а затем отправляет их в удаленный репозиторий. Эта ситуация проиллюстрирована на рис. 11.11.

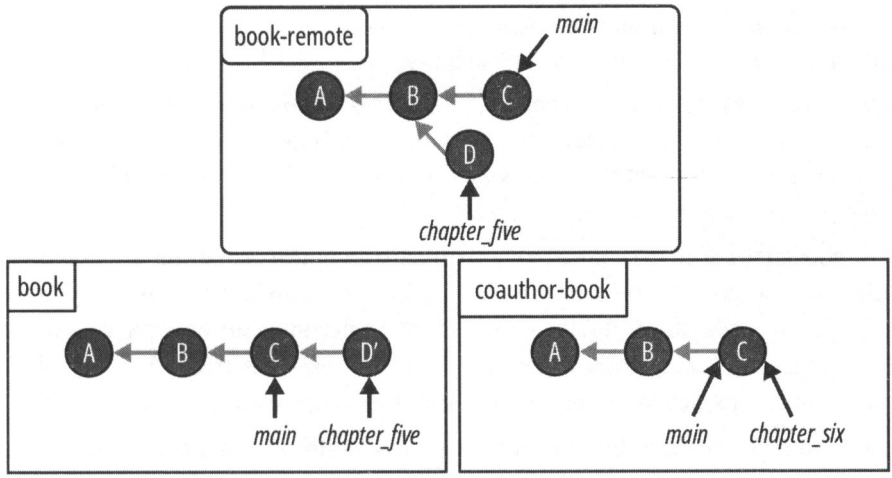


Рис. 11.10. Проект Book после перебазирования ветки `chapter_five` в ветку `main`

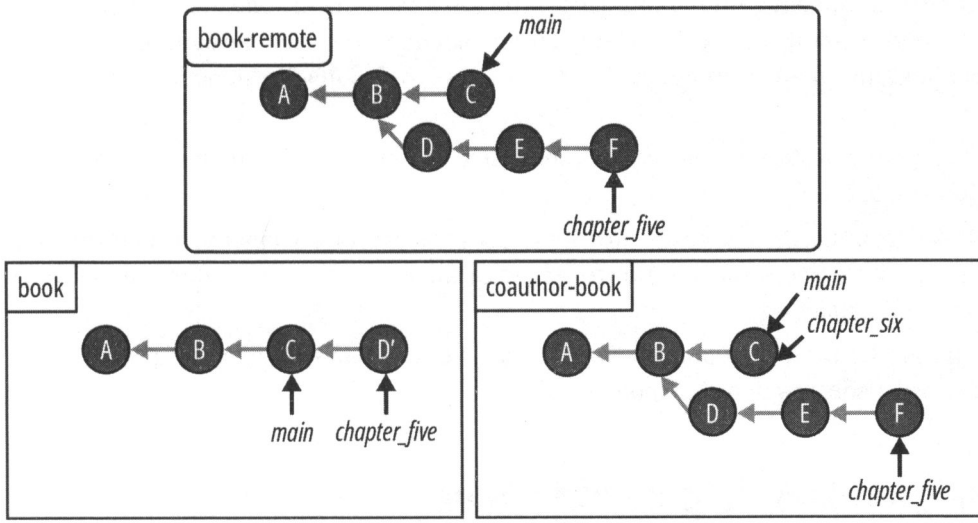


Рис. 11.11. Проект Book после того, как мой соавтор внес свой вклад в удаленную ветку `chapter_five`

Когда я попытаюсь отправить свою ветку `chapter_five` в удаленный репозиторий, я получу следующее сообщение об ошибке:

```
error: failed to push some refs to 'github.com:gitlearningjourney/rainbow-
remote.git'
hint: Updates were rejected because the tip of your current branch is
behind its remote counterpart. Integrate the remote changes (e.g. 'git
pull ...') before pushing again. See the 'Note about fast-forwards' in
'git push --help' for details.
```

(ошибка: не удалось отправить некоторые ссылки на 'github.com:gitlearningjourney/rainbow-remote.git'

подсказка: обновления были отклонены, поскольку окончание вашей текущей ветки отстает от удаленного аналога. Интегрируйте удаленные изменения (например, 'git pull ...') перед повторной отправкой. Подробности см. в разд. 'Примечание об ускоренном коммите' в 'git push --help'.)

Здесь Git сообщает мне, что я не могу отправить свои изменения в удаленный репозиторий. Как вы можете видеть на рис. 11.11, истории коммитов удаленной ветки `chapter_five` и локальной ветки `chapter_five` больше не содержат одни и те же коммиты: удаленная ветка `chapter_five` состоит из коммитов A, B, D, E и F, в то время как локальная ветка `chapter_five` состоит из коммитов A, B, C и D′.

Это щекотливая ситуация, потому что сейчас у меня и моего соавтора нет простого способа решить эту проблему. Нам придется общаться, чтобы решить проблему и придумать, как мы можем устранить различия между нашими ветками.

Пример 11.3 проекта Book показывает, почему вы должны следовать золотому правилу перебазирования, если собираетесь воспользоваться командой `git rebase`. В заключение можно отметить, что вы можете безопасно перебазировать ветку, если:

- у вас есть локальная ветка, которая никогда не была отправлена в удаленный репозиторий;
- у вас есть локальная ветка, которую вы поместили в удаленный репозиторий, и вы на 100% уверены, что никто не основывал на ней свою работу и не вносил в нее свой вклад.

Вот и все. Если есть вероятность, что над веткой работал кто-то другой, то рекомендуется избегать перебазирования.

Синхронизация репозиториев

Для того чтобы убедиться, что репозитории проекта Rainbow синхронизированы, вашему другу придется отправить свои изменения в удаленный репозиторий, а вам перенести изменения в репозиторий `rainbow`. Для того чтобы сделать это сейчас, выполните действия 11.10.

ДЕЙСТВИЯ 11.10

1
```
friend-rainbow $ git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
```

```
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 652 bytes | 652.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local
object.
To github.com:gitlearningjourney/rainbow-remote.git
   6f2cf36..7c09136 main -> main
```

2 `friend-rainbow $ **git log**`

```
commit 7c09136bcbfdd9f638ed13c6653e06451579d21c (HEAD -> main,
origin/main, origin/HEAD)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:10:11 2022 +0100

    rainbow
commit e055f2bc66aed1f3627041900a8c825c7a875206
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:07:38 2022 +0100

    black
```

3 Перейдите в окно командной строки, где вы находитесь в репозитории `rainbow`, чтобы выполнить команду из шага 4.

4 `rainbow $ **git pull**`

```
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 1), reused 6 (delta 1), pack-reused 0
Unpacking objects: 100% (6/6), 632 bytes | 105.00 KiB/s, done.
From github.com:gitlearningjourney/rainbow-remote
   6f2cf36..7c09136 main -> origin/main
Updating 6f2cf36..7c09136
Fast-forward
 othercolors.txt | 4 +++-
 rainbowcolors.txt | 2 +-
 2 files changed, 4 insertions(+), 2 deletions(-)
```

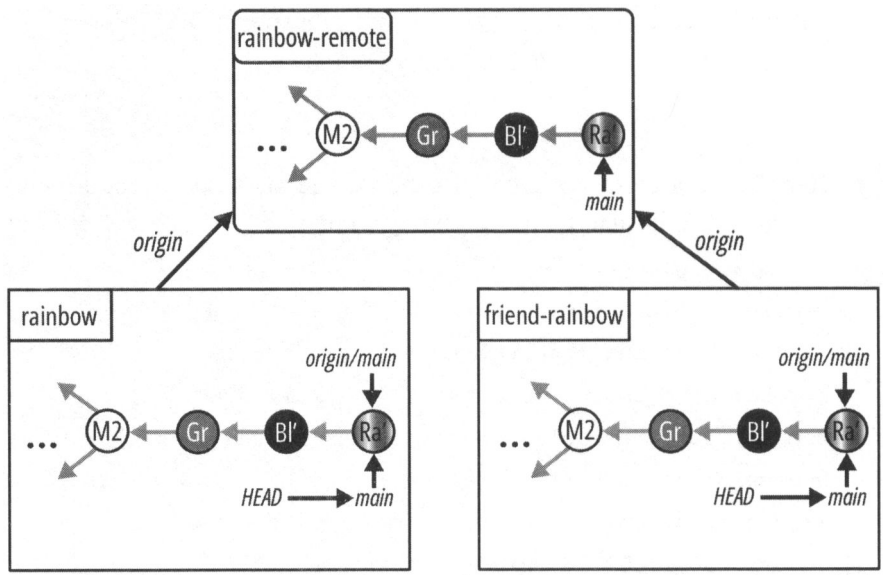
5 `rainbow $ **git log**`

```
commit 7c09136bcbfdd9f638ed13c6653e06451579d21c (HEAD -> main,
origin/main)
```

```
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:10:11 2022 +0100

    rainbow

commit e055f2bc66aed1f3627041900a8c825c7a875206
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:07:38 2022 +0100

    black
```

Все три репозитория в проекте Rainbow теперь синхронизированы, как показано на диаграмме 11.17.

ДИАГРАММА 11.17

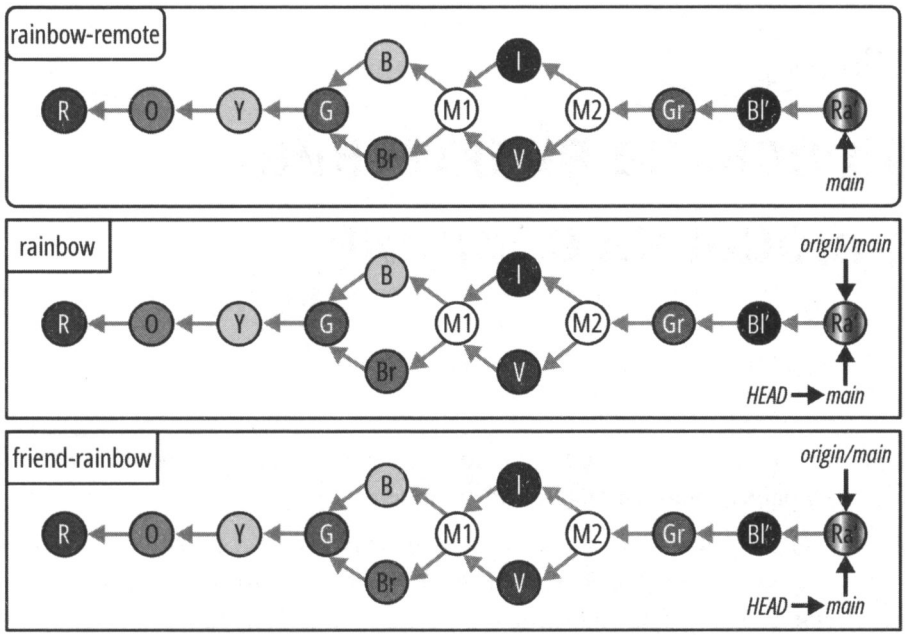


Проект Rainbow после того, как ваш друг отправил свою локальную ветку `main` в удаленный репозиторий, а вы извлекли данные из удаленного репозитория, чтобы обновить локальную ветку `main`.

Состояние локальных и удаленных репозиториев

Диаграмма 11.18 показывает текущее состояние локальных и удаленных репозиториев в проекте Rainbow со всеми коммитами, сделанными с *главы 1* до конца этой главы.

ДИАГРАММА 11.18



Состояние проекта Rainbow в конце *главы 11* со всеми изменениями, внесенными ачиная с *главы 1*.

Краткое содержание главы

В этой главе вы узнали о перебазировании — втором способе интеграции изменений из одной ветки в другую в Git — и о том, как его следует применять, чтобы избежать трехэтапных слияний и коммитов слияний. Вы узнали о пяти этапах процесса перебазирования, который выполняет Git, затем попрактиковались в перебазировании на примере проекта Rainbow. В ходе процесса вы также практиковались в разрешении конфликтов слияния.

Поскольку при перебазировании перезаписывается история коммитов, существует золотое правило перебазирования, которое гласит, что вы не должны перебазировать ветки, на которых могли основываться изменения ваших партнеров по проекту. Кроме того, вы узнали, как отменять изменения, т. е. удалять измененные файлы из промежуточной области, чтобы вы могли точно настроить содержимое предстоящего коммита.

К этому моменту вы умеете интегрировать изменения из одной ветки в другую при работе с локальным репозиторием посредством слияния или перебазирования. Далее, в *главе 12*, мы рассмотрим полезный инструмент для совместной работы над проектами Git, который позволяет удаленно интегрировать изменения, называемый запросом на включение.

[Глава 12]

Запросы на включение (запросы на слияние)

До сих пор мы рассматривали только интеграцию изменений из одной ветки в другую при работе с локальным репозиторием.

В этой главе вы узнаете о *запросах на включение* (pull request) — полезном инструменте для совместной работы, который позволяет интегрировать изменения из одной ветки в другую в удаленном репозитории. Попутно вы также научитесь удобному приему, позволяющему проще определять восходящие ветки для новых локальных веток.

Дополнительные ресурсы, которые помогут вам в работе с этой главой, доступны в репозитории книги (**https://github.com/gitlearningjourney/learning-git**).

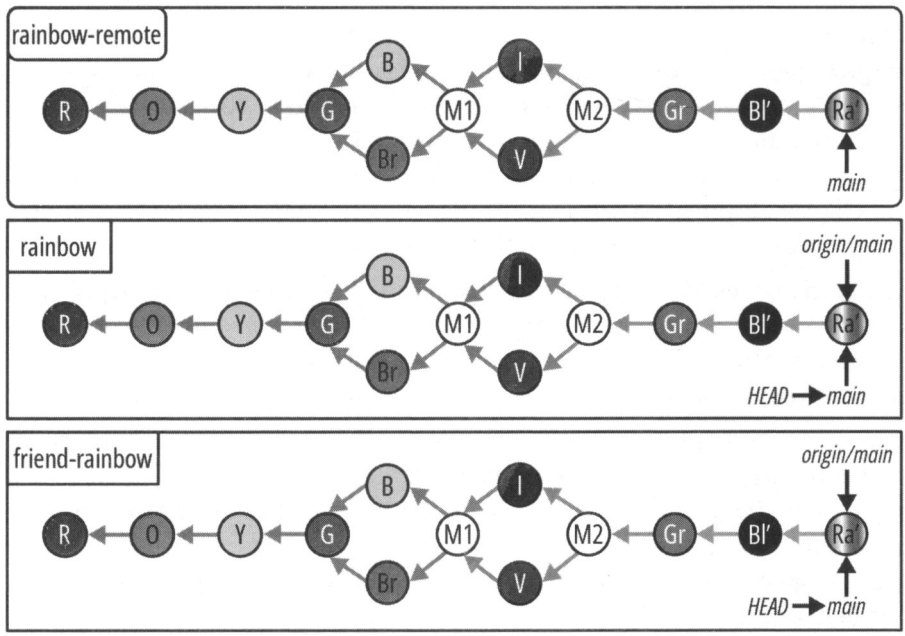
Состояние локальных и удаленных репозиториев

В начале этой главы у вас должны быть два локальных репозитория под названием `rainbow` и `friend-rainbow` и один удаленный репозиторий под названием `rainbow-remote`. Все три репозитория должны быть синхронизированы, с одинаковыми коммитами и ветками.

На диаграмме 12.1 показано состояние локальных и удаленных репозиториев в проекте Rainbow со всеми коммитами, сделанными в *главах 1–11*.

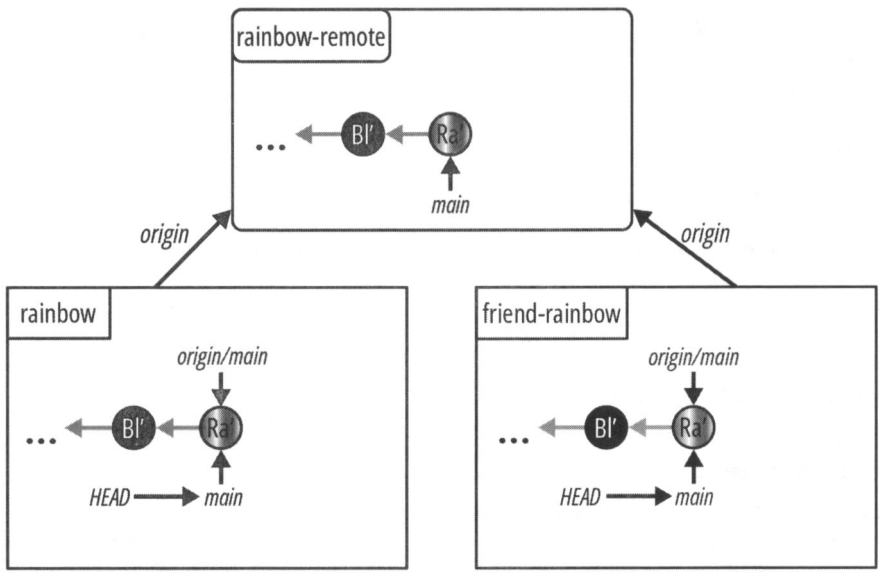
Для того чтобы сосредоточиться на коммитах, которые вам предстоит сделать в этой главе, с этого момента я упрощу диаграммы состояния проекта и буду показывать только два последних коммита, которые являются частью ветки `main` во всех репозиториях: черный (black) и радужный (rainbow). Это представление показано на диаграмме 12.2.

ДИАГРАММА 12.1



Состояние проекта Rainbow в начале *главы 12* со всеми изменениями, внесенными после *главы 1*.

ДИАГРАММА 12.2



Упрощенное представление проекта Rainbow в начале *главы 12*, показывающее только два последних коммита в основной ветке во всех репозиториях.

Знакомство с запросами на включение

Запрос на включение (pull request, также часто называемый *запросом на слияние*, или merge request) — это функция, предлагаемая службой хостинга. Она позволяет вам делиться работой, которую вы выполнили в ветке, со своими партнерами по проекту, потенциально собирать отзывы об этой работе и, наконец, интегрировать работу в проект удаленно на хостинге. Хотя запросы на включение являются функцией не Git, а хостинга, на котором размещаются проекты, использующие Git, они настолько полезны в повседневной работе с Git, что я не могу обойти их вниманием.

Запросы на включение можно интегрировать путем слияния или перебазирования, но по умолчанию (и чаще всего) используется вариант слияния, поэтому в примере в этой главе мы остановимся на нем. В оставшейся части главы я буду называть процесс интеграции запроса на включение "слиянием запроса на включение".

Когда вы создаете запрос на включение, можно сказать, что вы "открываете" запрос на включение. Когда запрос на включение будет рассмотрен и одобрен, а соответствующее слияние выполнено, вы "закрываете" его. Вы также можете закрыть запрос на включение досрочно, если передумаете и захотите удалить его из списка открытых запросов (например, если запрос на включение не будет одобрен руководителем проекта).

Процесс запроса на включение можно разделить на девять шагов:

1. Создайте ветку в локальном репозитории.
2. Добавьте свою работу, сделав коммиты в ветке.
3. Отправьте ветку в удаленный репозиторий.
4. Создайте (или откройте) запрос на включение в службе хостинга.
5. Проверьте запрос на включение и, возможно, включите в него отзывы других людей.
6. Получите одобрение запроса на включение.
7. Выполните слияние запроса на включение.
8. Если это была тематическая ветка (функциональная ветка), "сотрите" удаленную ветку.
9. Извлеките изменения, чтобы синхронизировать локальный репозиторий с удаленным репозиторием, и выполните очистку, удалив локальную ветку и ветку удаленного отслеживания.

ПРИМЕЧАНИЕ

Если между ветками запроса на включение возникают конфликты слияния, вы не сможете его выполнить. Сначала необходимо эти конфликты разрешить. Некоторые службы хостинга предоставляют поддержку для разрешения конфликтов слияния на своем веб-сайте; однако чаще всего конфликты слияния разрешаются в локальном репозитории с помощью процесса, описанного в *главе 10*.

Прежде чем мы перейдем к примеру, давайте кратко отметим некоторые моменты, которые вам нужно иметь в виду при работе со службой хостинга.

Особенности службы хостинга

Конкретные шаги по созданию запросов на включение и управлению ими с помощью GitHub, GitLab и Bitbucket различаются, и вам, возможно, придется обратиться к документации вашего хостинга, чтобы выполнить некоторые упражнения, описанные в этой главе. Дополнительные ресурсы можно найти в репозитории книги (**https://github.com/gitlearningjourney/learning-git**).

В зависимости от хостинга могут различаться даже термины: в частности, в то время как GitHub и Bitbucket используют термин *"запрос на включение"* (pull request), GitLab называет ту же функцию *запросом на слияние* (merge request). Помните об этих различиях при выполнении действий с репозиториями в этой главе.

> **ПРИМЕЧАНИЕ**
>
> Несмотря на то что в названии присутствует слово "pull", запросы на включение на самом деле не связаны с командой `git pull`.

Наконец, в *главе 6*, когда мы рассматривали выбор службы хостинга и настройку доступа по протоколу HTTPS или SSH, я рекомендовала вам использовать личную учетную запись службы хостинга, а не учетную запись компании. Это связано с тем, что в службе хостинга можно настроить дополнительные параметры для процесса создания и утверждения запроса на включение. Например, вы (или компания, в которой вы работаете) можете выставить определенные требования для создания запроса на включение или установить ограничения на то, кто может утверждать запросы на включение, или сколько человек в команде должны одобрить запрос на включение, прежде чем для него будет выполнено слияние.

В примерах в этой книге предполагается, что на хостинге не задано никаких дополнительных настроек. Однако о них следует помнить, если вы используете учетную запись компании, поскольку могут существовать дополнительные требования и ограничения, которые влияют на вашу способность создавать запросы на включение и управлять ими.

Далее мы обсудим, зачем вам вообще могут понадобиться запросы на включение.

Зачем нужны запросы на включение?

Запросы на включение облегчают общение и совместную работу над проектами Git, предоставляя простой механизм проверки работы. У них есть полезная функция комментирования, которая позволяет вам и вашим коллегам добавлять комментарии к определенным строкам в файлах проекта, отвечать на эти комментарии и начинать обсуждения. Это помогает организовать процесс проверки.

Поскольку запросы на включение полностью управляются в пользовательском интерфейсе службы хостинга, они также позволяют людям, не использующим Git, оставлять отзывы о проектах Git — не обязательно иметь навыки работы с Git, чтобы комментировать запрос на включение! Пример 12.1 проекта Book продемонстрирует нам практическую ситуацию, в которой будет полезен запрос на включение.

Пример 12.1 проекта Book

Предположим, я создаю ответвление от ветки `main` под названием `chapter_nine` для работы над главой 9 моей книги. Я делаю два коммита в этой ветке — W и X, а затем отправляю свою локальную ветку `chapter_nine` в удаленный репозиторий, создавая удаленную ветку `chapter_nine`. Состояние локального и удаленного репозиториев показано на рис. 12.1.

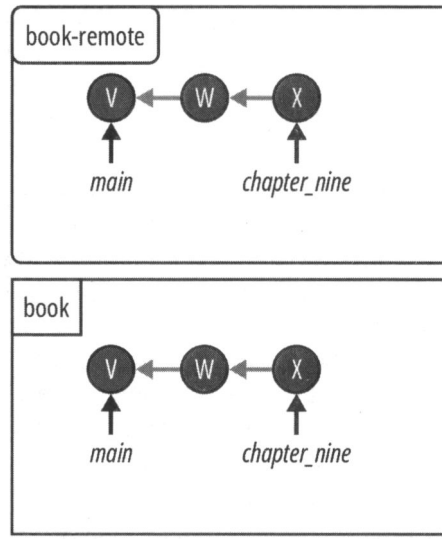


Рис. 12.1. Проект Book после отправки ветки `chapter_nine` в удаленный репозиторий

Как вы помните, я договорилась со своим редактором, что он должен проверить мою работу, прежде чем я добавлю ее в ветку `main` через слияние. Это означает, что ему необходимо просмотреть новую ветку `chapter_nine`. Один из вариантов — клонировать удаленный репозиторий на свой локальный компьютер и проверить там мой обновленный файл. Однако это не дает редактору простого способа предоставить мне отзывы и комментарии. Кроме того, давайте предположим, что мой редактор еще не научился использовать Git и поэтому ему будет сложно выполнить клонирование и работать в локальном репозитории.

Вместо этого я открою моему редактору доступ к удаленному репозиторию, а затем сделаю в удаленном репозитории запрос на включение, чтобы объединить удаленную ветку `chapter_nine` с удаленной веткой `main`. Я могу либо отправить своему редактору URL-адрес запроса на включение, либо просто попросить его перейти в

удаленный репозиторий и найти запрос на включение под названием "Обновления главы 9".

Затем мой редактор может воспользоваться функцией комментирования, чтобы задавать вопросы и оставлять отзывы. Предположим, он заметил в главе неточность, которую мне нужно исправить, поэтому он оставляет комментарий в запросе на включение и сообщает мне о необходимости просмотреть его отзыв. Устранив проблему в главе в моем локальном репозитории, я делаю еще один коммит в локальной ветке `chapter_nine` (коммит Y) и отправляю его в удаленный репозиторий. Это действие автоматически обновит удаленную ветку `chapter_nine` и, следовательно, автоматически обновит запрос на включение. На рис. 12.2 показано обновленное состояние локального и удаленного репозиториев.

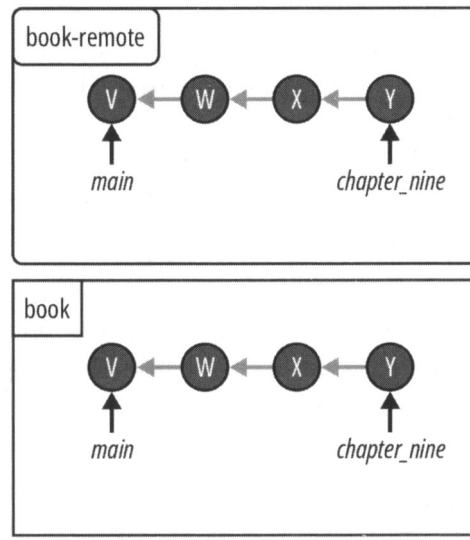


Рис. 12.2. Проект Book после того, как я сделаю коммит Y в ветке `chapter_nine` и отправлю в удаленный репозиторий

Тот факт, что я сделала запрос на включение, позволяет моему редактору предоставить мне отзыв о работе, а мне — поделиться своими обновлениями после того, как я учту его замечания.

Пример 12.1 проекта Book показал вам, почему запросы на включение так полезны. Далее вы узнаете, как выполняется слияние, инициированное запросом на включение.

Слияние в ходе запроса на включение

Как вы знаете, в Git есть два типа слияний: *ускоренное слияние* происходит, когда истории развития исходной и целевой веток не расходятся; в противном случае происходит *трехстороннее слияние* и появляется коммит слияния. Вы рассмотрели

примеры выполнения ускоренного и трехстороннего слияний в *главах 5* и *9* соответственно.

По умолчанию удаленное слияние отличается от локального. На большинстве хостингов удаленное слияние с запросом на включение происходит *без ускоренного перехода вперед* (non-fast-forward). При этом варианте слияния, даже если истории разработки исходной и целевой веток не разошлись, коммит слияния все равно будет создан.

Слияние, выполненное без ускоренного перехода вперед, иногда называют *явным слиянием* (explicit merge), потому что оно с помощью коммита слияния всегда явно показывает, где произошло слияние.

Настройки службы хостинга можно изменить, чтобы выполнялся другой вариант слияния (или другим способом). Однако, поскольку вариант без ускоренного перехода вперед является наиболее распространенным, именно его мы и рассмотрим в этой книге. Давайте вернемся к сценарию из примера 12.1 проекта Book и продолжим его в примере 12.2 того же проекта, чтобы увидеть, как обычно работают запросы на слияние.

Пример 12.2 проекта Book

В примере 12.1 проекта Book говорилось о том, как я сделала запрос на включение под названием "Обновления главы 9", чтобы мой редактор мог проверить работу, которую я выполнила в ветке `chapter_nine` и собираюсь объединить с веткой `main`. На рис. 12.3 показано состояние проекта Book на данный момент.

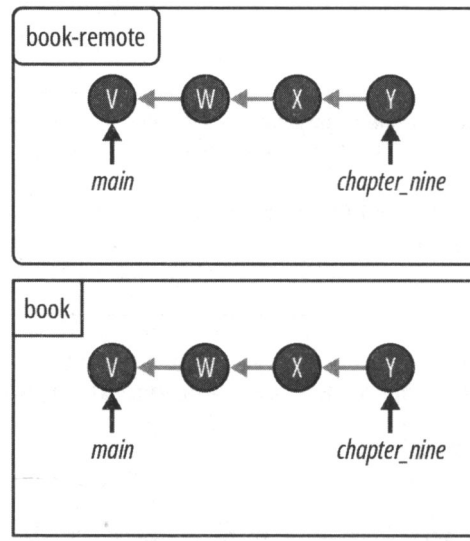


Рис. 12.3. Состояние локального и удаленного репозиториев в проекте Book

После того как мой редактор проверил последнюю работу, которую я отправила в удаленную ветку `chapter_nine`, представленную коммитом Y, у него больше нет

замечаний. Поэтому он нажимает специальную кнопку на веб-сайте хостинга, чтобы одобрить запрос на включение.

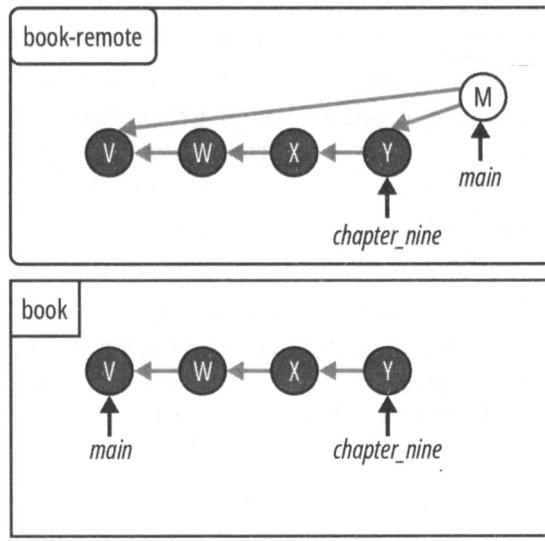


Рис. 12.4. Проект Book после подтверждения моего запроса на включение и слияния ветки `chapter_nine` с веткой `main`

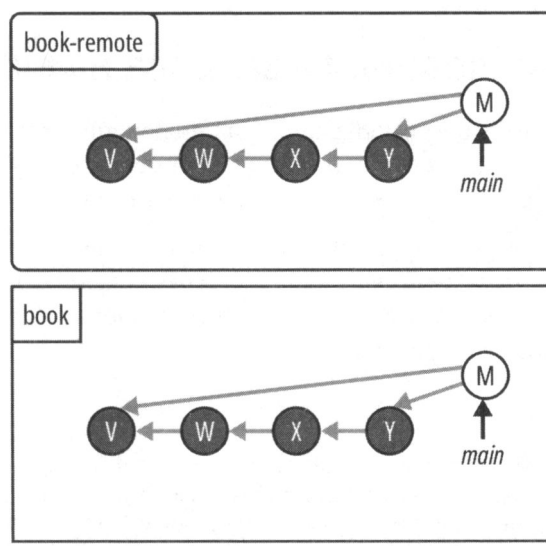


Рис. 12.5. Проект Book после удаления ветки `chapter_nine` и переноса изменений из удаленной ветки `main` в мою локальную ветку `main`

Это означает, что я могу объединить удаленную ветку `chapter_nine` с удаленной веткой `main`, нажав свою кнопку на веб-сайте хостинга для слияния по запросу на включение. Поскольку у меня установлены настройки по умолчанию, это будет слияние без ускоренного перехода вперед. Несмотря на то что истории разработки

удаленной ветки `chapter_nine` и удаленной ветки `main` не разошлись, все равно будет создан коммит слияния. На рис. 12.4 он показан как коммит M.

Родителями коммита слияния M являются коммит V, на который указывала ветка `main` перед слиянием, и коммит Y — последний коммит в ветке `chapter_nine`.

Последние действия, которые мне нужно предпринять, — это удалить ветку `chapter_nine` и перенести самую последнюю версию ветки `main` из удаленного репозитория в свой локальный репозиторий, чтобы моя локальная ветка `main` синхронизировалась с удаленным репозиторием. Это показано на рис. 12.5.

Родителями коммита слияния M являются коммит V, на который указывала ветка `main` перед слиянием, и коммит Y — последний коммит в ветке `chapter_nine`.

Последние действия, которые мне нужно предпринять, — это удалить ветку `chapter_nine` и перенести самую последнюю версию ветки `main` из удаленного репозитория в свой локальный репозиторий, чтобы моя локальная ветка `main` синхронизировалась с удаленным репозиторием. Это показано на рис. 12.5.

Итак, вы узнали, как происходит завершение запроса на включение в примере проекта Book 12.2. Теперь вернемся к проекту Rainbow, чтобы попрактиковаться в выполнении процесса запроса на включение.

Подготовка к выполнению запроса на включение

Для того чтобы рассмотреть пример запроса на включение, вы начнете с выполнения шагов 1 и 2 процесса запроса на включение (описанного в *разд. "Знакомство с запросами на включение" ранее в этой главе*) в репозитории `rainbow`.

При выполнении действий 12.1 вы будете вводить команду `git switch` с опцией `-c`, которая описана в *разд. "Создание ветки с переключением на нее" главы 5*, чтобы создать новую ветку под названием `topic` и немедленно переключиться на нее. Затем вы добавите в нее свою работу.

ПРИМЕЧАНИЕ

Причина, по которой вы будете использовать общее имя `topic` для новой ветки, заключается в следующем. Как сказано в *разд. "Почему мы используем ветки?" главы 4*, ветки, созданные для работы над определенной частью проекта в Git, принято называть *тематическими* (topic) или *функциональными* (feature). В реальном проекте Git название ветки обычно содержит краткое описание функции или темы, над которой вы работаете.

ДЕЙСТВИЯ 12.1

1
```
rainbow $ git switch -c topic
Switched to a new branch 'topic'
```

2. Откройте в окне файловой системы каталог проекта `rainbow`. В текстовом редакторе откройте файл othercolors.txt и в строке 4 добавьте текст "Розовый не является цветом радуги.", затем сохраните файл.

3. `rainbow $ git add othercolors.txt`

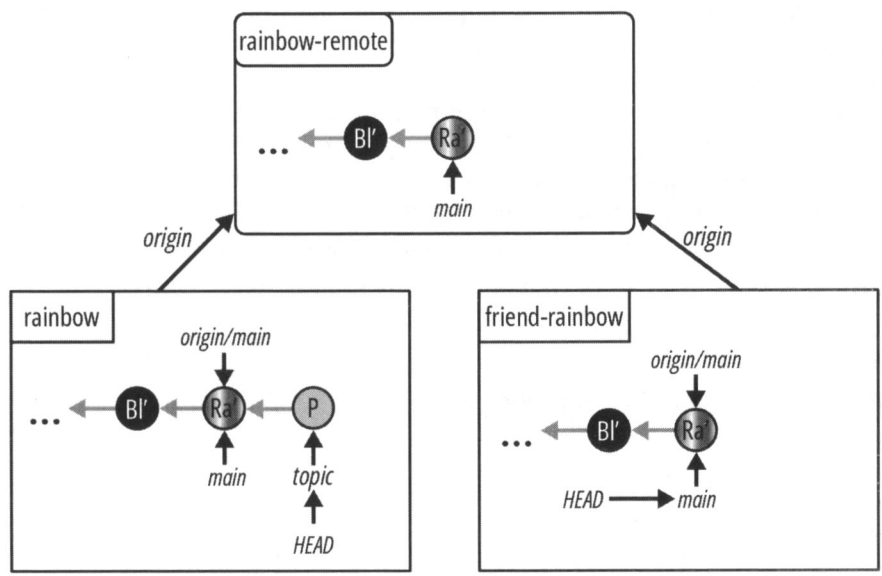
4. `rainbow $ git commit -m "pink"`

```
[topic 4c35a5c] pink
 1 file changed, 1 insertion(+), 1 deletion(-)
```

5. `rainbow $ git log`

```
commit 4c35a5c02c3dc03f044cbdfdbb0ae55161af6a86 (HEAD -> topic)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Jul 3 14:16:01 2022 +0200

    pink

commit 7c09136bcbfdd9f638ed13c6653e06451579d21c (origin/main,
main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:10:11 2022 +0100

    rainbow
```

ДИАГРАММА 12.3



Проект Rainbow после внесения розового коммита в ветку темы в репозитории `rainbow`.

На что обратить внимание:

- вы создали розовый коммит в ветке `topic` репозитория `rainbow`.

Это показано на диаграмме 12.3.

Ветка `topic` — это новая локальная ветка. Для нее не определена *восходящая ветка*. (Напомню, что восходящая ветка — это удаленная ветка, которая отслеживает конкретную локальную ветку; вы узнали об этой концепции в *главе 7*, а определение восходящей ветки было дано в *главе 9*.)

Далее вы выполните шаг 3 процесса запроса на включение, который заключается в отправке вашей работы в удаленный репозиторий. Попутно я покажу вам более простой способ определения восходящих веток.

Более простой способ определения восходящих веток

В *главе 9* вы узнали, как использовать команду `git branch -u <короткое_имя>/<имя_ветки>` для определения восходящей ветки. Но теперь, когда мы приближаемся к концу книги, я открою вам небольшой секрет. Пользователи Git очень часто забывают определить восходящие ветки. Или не настраивают их просто из-за лени.

Но есть полезный трюк, к которому пользователи Git любят прибегать для определения восходящих веток, когда впервые помещают новую ветку в удаленный репозиторий. Если вы попытаетесь выполнить команду `git push` без каких-либо аргументов в ветке, для которой не определена восходящая ветка, Git выдаст предупреждение в выводе команды. В предупреждении будет указана команда для определения ветки (`git push --set-upstream <короткое_имя> <имя_ветки>`), а также предложено короткое имя и имя ветки, которые вы, возможно, захотите использовать (Git предлагает сделать так, чтобы удаленная ветка имела то же имя, что и локальная). Если вы скопируете и вставите эту команду в окно командной строки и выполните ее, то сможете решить две задачи за раз: вы отправите локальную ветку в удаленный репозиторий и определите для нее вышестоящую ветку.

Выполните действия 12.2, чтобы опробовать этот прием на практике.

ДЕЙСТВИЯ 12.2

1
```
rainbow $ git branch -vv
  main 7c09136 [origin/main] rainbow
* topic 4c35a5c pink
```
2
```
rainbow $ git push
fatal: The current branch topic has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin topic
```

(Критическая ошибка: текущая ветка topic не имеет восходящей ветки.

Для отправки текущей ветки и определения вышестоящей выполните команду

git push --set-upstream origin topic)

3 Скопируйте команду `git push --set-upstream origin topic` из вывода на шаге 2 и вставьте в командную строку.

4
```
rainbow $ git push --set-upstream origin topic
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 314 bytes | 314.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local
object.
remote:
remote: Create a pull request for 'topic' on GitHub by visiting:
remote: https://github.com/gitlearningjourney/rainbow-remote/
pull/new/topic
remote:
To github.com:gitlearningjourney/rainbow-remote.git
 * [new branch]      topic -> topic
branch 'topic' set up to track 'origin/topic'.
```

5
```
rainbow $ git branch -vv
  main 7c09136 [origin/main] rainbow
* topic 4c35a5c [origin/topic] pink
```

6
```
rainbow $ git log
commit 4c35a5c02c3dc03f044cbdfdbb0ae55161af6a86 (HEAD -> topic,
origin/topic)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Jul 3 14:16:01 2022 +0200

    pink

commit 7c09136bcbfdd9f638ed13c6653e06451579d21c (origin/main,
main)
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Feb 20 10:10:11 2022 +0100

    rainbow
```

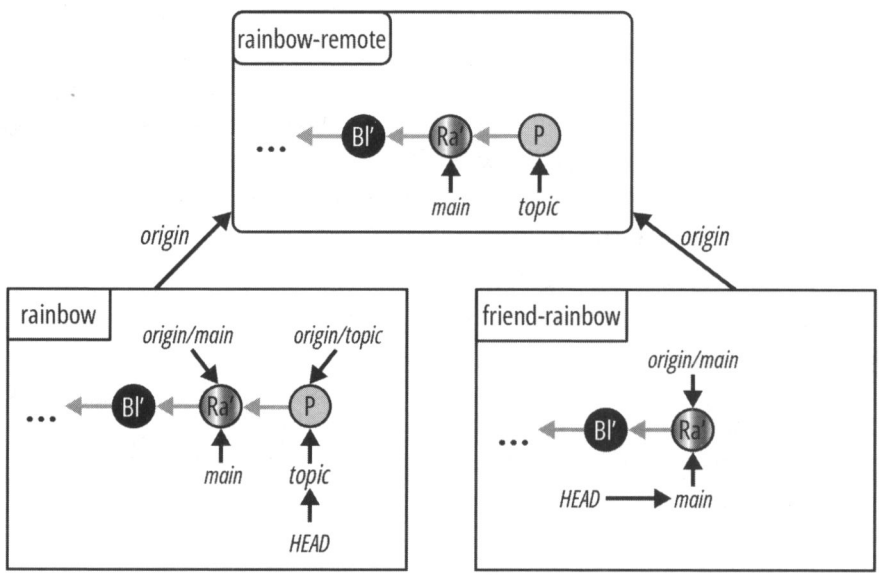
7 Перейдите в репозиторий `rainbow-remote` на своем хостинге и обновите страницу. Вы должны увидеть созданную вами новую ветку `topic`, и если вы выберете просмотр коммитов в ветке темы, вы также увидите розовый коммит.

На что обратить внимание:

- на шаге 2 выходные данные команды `git push` выдают предупреждение о том, что ветка `topic` не имеет восходящей ветки. Там также содержится рекомендация выполнить команду `git push --set-upstream origin topic`, чтобы отправить текущую ветку и установить удаленную ветку в качестве восходящей;
- на шаге 4 выходные данные команды `git push` рекомендуют вам сделать запрос на включение для новой ветки по адресу `https://github.com/gitlearningjourney/rainbow-remote/pull/new/topic`.

Текущее состояние репозиториев после выполнения действий 12.2 показано на диаграмме 12.4.

ДИАГРАММА 12.4



Проект Rainbow после того, как вы отправили ветку `topic` в репозиторий `rainbow-remote` и одновременно определили вышестоящую ветку для локальной ветки `topic`.

Вы только что увидели, как при помощи команды `git push` можно легко определить вышестоящую ветку, одновременно отправляя ветку в удаленный репозиторий. Далее вы выполните шаг 4 процесса запроса на включение.

ПРИМЕЧАНИЕ

Этот трюк предполагает, что вы согласны с тем, чтобы восходящая ветка в удаленном репозитории имела то же самое имя, что и локальная ветка. Если это не так, вам, возможно, придется отредактировать рекомендованную команду `git push`, чтобы указать альтернативную удаленную ветку, которую вы хотите установить в качестве восходящей ветки.

Создание запроса на включение для службы хостинга

Когда вы создаете запрос на включение, вам необходимо определить исходную и целевую ветки. В примере проекта Rainbow `topic` — это исходная ветка, а `main` — целевая ветка. Другими словами, вы сливаете `topic` с `main`. Для того чтобы создать запрос на включение, вам придется использовать пользовательский интерфейс вашего хостинга. Дополнительные ресурсы можно найти в репозитории книги (**https://github.com/gitlearningjourney/learning-git**).

На веб-странице создания запроса на включение у вас будет возможность ввести информацию о запросе. Единственное обязательное поле для большинства услуг хостинга — это заголовок. Как и в случае с сообщениями коммита и названиями веток, если вы работаете над проектом Git в составе команды, вам следует уточнить у коллег, приняты ли у них правила составления заголовков запросов на включение. В примере проекта Rainbow в этой книге вы будете использовать заголовок "Добавление розового цвета", а все остальные поля оставите пустыми.

В GitLab и Bitbucket при создании запроса на включение вы также можете выбрать опцию автоматического удаления ненужной ветки после успешного слияния в ходе обработки запроса на включение. Вам решать, хотите ли вы выбрать эту опцию (если она доступна на вашем хостинге). В любом случае после выполнения запроса на включение вы должны убедиться, что ненужная ветка удалена (или она все еще существует), поскольку это шаг 8 в процессе запроса на включение.

Теперь выполните действия 12.3, чтобы создать запрос на включение.

ДЕЙСТВИЯ 12.3

1. Перейдите в удаленный репозиторий на своем хостинге. Следуйте инструкциям, чтобы создать запрос на включение для слияния ветки `topic` (исходной) с `main` (целевой). Служба хостинга может предварительно заполнить поле заголовка запроса на включение сообщением вашего последнего коммита, которое в вашем случае состоит из единственного слова "pink" (розовый).

Измените его на более понятный "Добавление розового цвета". Дополнительные ресурсы можно найти в репозитории книги (**https://github.com/gitlearningjourney/learning-git**).

Теперь, когда вы создали свой первый запрос на включение, пришло время перейти к шагам 5 и 6 процесса запроса на включение, проверить запрос и утвердить его.

Рассмотрение и утверждение запроса на включение

Запросы на включение дают возможность участникам проекта просмотреть вашу работу, внести в нее свой вклад или просто одобрить ее. Пользовательский интерфейс службы хостинга обычно предоставляет возможность просмотра строк, которые были изменены в каждом модифицированном файле, и того, как они были изменены, используя цвета и символы для отображения этой информации. На рис. 12.6 показано, как служба хостинга может указать, что вы добавили в строку 4 файла othercolors.txt предложение о розовом цвете.

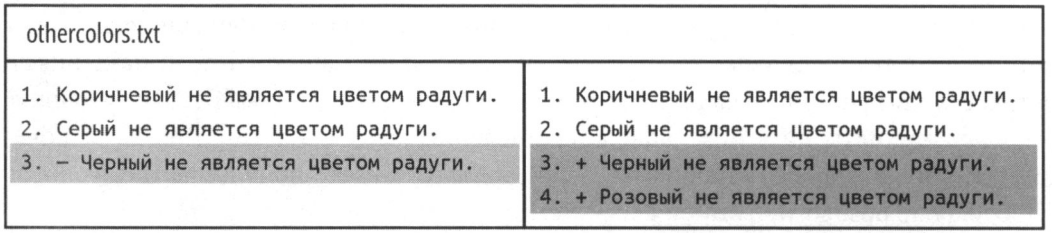


Рис. 12.6. Пример того, как служба хостинга отображает файлы, измененные в запросе на включение

Когда соавторы просматривают вашу работу, они могут оставить комментарии к запросу на включение. Если в вашем запросе на включение есть комментарии, вы можете просмотреть их, возможно, внести изменения и отправить один или несколько дополнительных коммитов в ветку, которую предстоит объединить с целевой веткой. Это автоматически обновит запрос на включение новыми коммитами.

Соавтор также может перенести ветку в свой локальный репозиторий и внести в нее дополнительные изменения, добавив коммиты и отправив их в удаленный репозиторий. Опять же, запрос на включение автоматически обновится с добавлением дополнительных коммитов.

Предположим, в примере проекта Rainbow вы сообщили своему другу, что сделали запрос на включение, и попросили его проверить изменения. Ваш друг перейдет к запросу на включение, ознакомится с ним, решит, что он доволен внесенными вами изменениями, и одобрит запрос на включение.

Обычно, если ваш коллега просматривает запрос на включение, он входит в *свою* учетную запись на службе хостинга, затем проверяет работу и одобряет запрос, нажав кнопку **Approve** (Одобрить) в пользовательском интерфейсе службы хостинга. Однако, поскольку вы имитируете действия двух человек, эта кнопка может не отображаться в вашем пользовательском интерфейсе. В этом случае вам просто придется сделать вид, что ваш друг одобряет запрос на включение.

Выполните действия 12.4, чтобы имитировать ситуацию, когда ваш друг просматривает и одобряет запрос на включение.

ДЕЙСТВИЯ 12.4

1. Представьте, что эти действия выполняет ваш друг, и перейдите к разделу запросов на включение на сайте хостинга.
2. В разделе запросов на включение найдите файлы, которые были изменены. При необходимости ознакомьтесь с инструкциями хостинга, чтобы узнать, как это делается.
3. "Одобрите" запрос от имени друга.

Как только запрос на включение будет одобрен (если не в буквальном смысле, то в переносном), вы можете перейти к шагу 7 процесса запроса на включение.

Слияние в ходе запроса на включение

Теперь, снова действуя от своего имени, вы можете вернуться к службе хостинга и выполнить действия по слиянию запроса на включение. Обычно это просто переход к запросу на включение в пользовательском интерфейсе и выбор соответствующей кнопки. Выполните действия 12.5, чтобы осуществить слияние вашего запроса на включение прямо сейчас.

ДЕЙСТВИЯ 12.5

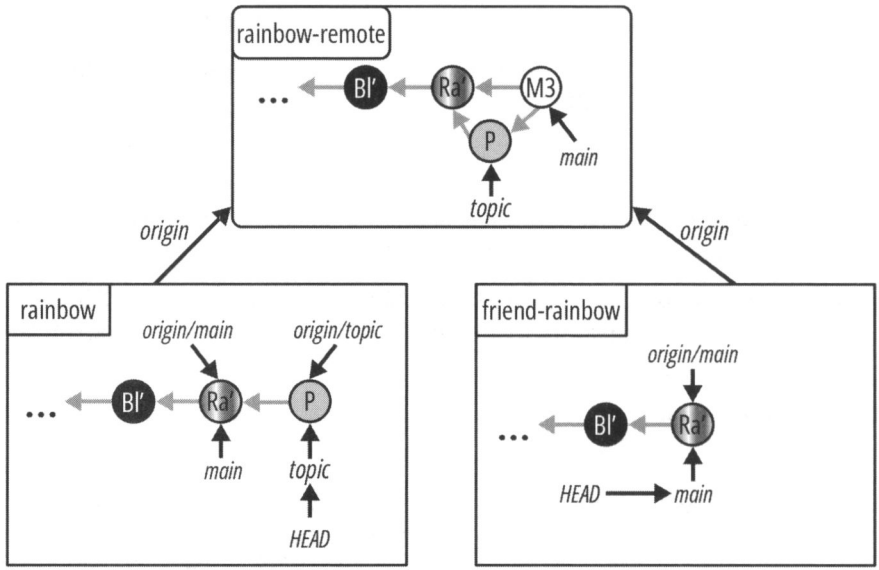
1. Следуйте инструкциям вашей службы хостинга, чтобы выполнить слияние вашего запроса на включение.
2. Просмотрите список коммитов в удаленном репозитории. Найдите последний коммит слияния и выберите его. Запишите хеш коммита, а также хеши двух его родительских коммитов.

На что обратить внимание:

- вы объединили удаленную ветку `topic` с удаленной веткой `main` и создали коммит слияния.

Результат этих действий представлен на диаграмме 12.5; новый коммит слияния помечен как M3.

ДИАГРАММА 12.5



Проект Rainbow после слияния запроса на включение в удаленном репозитории.

Как упоминалось ранее в этой главе, это слияние выполняется с использованием опции по умолчанию, исключающей ускоренное слияние. Это означает, что даже несмотря на то, что истории развития удаленной ветки `topic` и удаленной ветки `main` не разошлись, все равно будет создан коммит слияния.

На диаграмме 12.5 вы можете видеть, что удаленная ветка `topic` все еще находится в репозитории `rainbow-remote`, а локальная ветка `topic` и ветка удаленного отслеживания `origin/topic` остаются в репозитории `rainbow`. На восьмом шаге процесса запроса на включение нам нужно избавиться от удаленной ветки `topic`.

Удаление ненужных веток

Если ветка, которую вы поместили в запрос на включение, является тематической, обычно ее удаляют сразу после слияния запроса на включение, поскольку можно предположить, что работа над этой веткой завершена. Для новых частей работы вы создадите новые ветки и опять пройдете процесс запроса на включение. Это позволяет упорядочить удаленный репозиторий и не загромождать его старыми ветками.

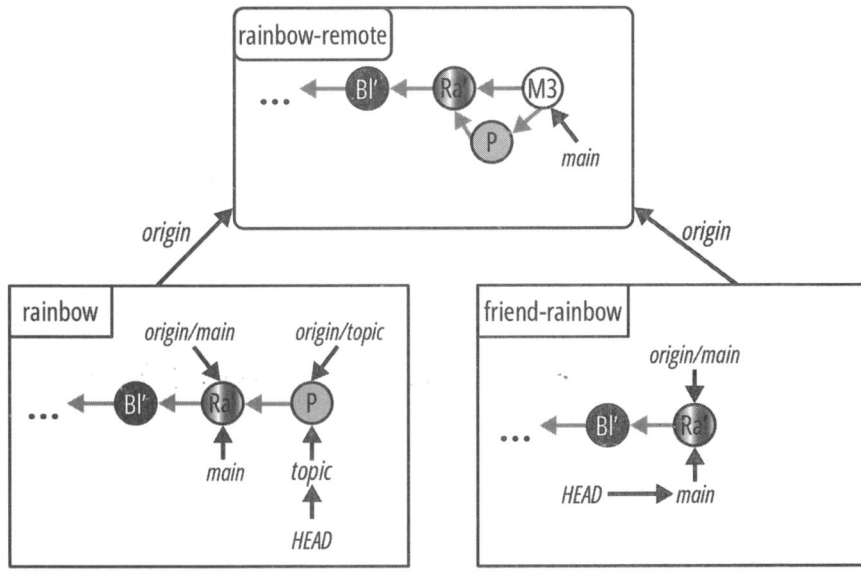
Если при создании запроса на включение ваша служба хостинга предлагала вам возможность автоматического удаления ветки после слияния запроса на включение, то ветка `topic` в удаленном репозитории к этому моменту может отсутствовать. Если это не так или вы решили не выбирать эту опцию, выполните действия 12.6, чтобы убрать ветку `topic` из удаленного репозитория.

ДЕЙСТВИЯ 12.6

1. Перейдите в свой удаленный репозиторий на своем хостинге и удалите ветку `topic`. При необходимости обратитесь к документации службы хостинга, чтобы узнать, как это сделать.

Диаграмма 12.6 иллюстрирует состояние репозиториев после удаления ветки `topic` на хостинге.

ДИАГРАММА 12.6



Проект Rainbow после удаления ветки `topic` из репозитория на хостинге.

Как видите, локальные ветки `main` в обоих локальных репозиториях теперь не синхронизированы с веткой `main` в удаленном репозитории. Кроме того, локальная ветка `topic` и ветка удаленного отслеживания `origin/topic` все еще находятся в репозитории `rainbow`. Далее вы убедитесь, что все должны извлекать последние изменения из удаленного репозитория, чтобы основная линия разработки проекта Rainbow — ветка `main` — была актуальной в обоих локальных репозиториях. Вы

также очистите репозиторий `rainbow`, удалив локальную ветку `topic` и ветку удаленного отслеживания `origin/topic`.

Синхронизация и очистка локальных репозиториев

Последний шаг процесса запроса на включение — синхронизация ваших локальных репозиториев с удаленным репозиторием. Сначала вы синхронизируете репозиторий `rainbow`, а поскольку для ветки `topic` было выполнено слияние, вы также удалите локальную ветку `topic` и ветку удаленного отслеживания `origin/topic`.

Напомним, что в *главе 8* вы использовали команду `git fetch` с опцией `-p` (от слова prune — обрезка) для избавления от всех ветвей удаленного отслеживания, соответствующих ветвям, которые были "стерты" в удаленном репозитории. На этот раз вы добавите опцию `-p` к команде `git pull`, и она будет иметь тот же эффект. Выполните действия 12.7, чтобы синхронизировать репозиторий `rainbow`.

ДЕЙСТВИЯ 12.7

1
```
rainbow $ git switch main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

2
```
rainbow $ git pull -p
From github.com:gitlearningjourney/rainbow-remote
 - [deleted]         (none)     -> origin/topic
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
Unpacking objects: 100% (1/1), 626 bytes | 626.00 KiB/s, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
   7c09136..2f833d6 main -> origin/main
Updating 7c09136..2f833d6
Fast-forward
 othercolors.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

3
```
3 rainbow $ git branch -d topic
Deleted branch topic (was 4c35a5c).
```

(Удалена ветка topic)

4
```
rainbow $ git log
commit 2f833d6fa783882c5f832da9e1eafe6d405d3468 (HEAD -> main,
origin/main)
Merge: 7c09136 4c35a5c
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Mon Jul 4 05:50:08 2022 +0200

    Merge pull request #1 from gitlearningjourney/topic

    Adding the color pink

commit 4c35a5c02c3dc03f044cbdfdbb0ae55161af6a86
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Jul 3 14:16:01 2022 +0200

    pink
```

На что обратить внимание:

- вы удалили локальную ветку `topic` и ветку удаленного отслеживания `origin/topic`;
- вы обновили локальную ветку `main`;
- коммит слияния (M3) содержит сообщение о коммите и описание, которые автоматически создаются службой хостинга. В примере в этой книге сообщение о коммите — это `Merge pull request #1 from gitlearningjourney/topic` (*Запрос на слияние № 1 из gitlearningjourney/topic*), а описание коммита — `Adding the color pink` (*Добавление розового цвета*), которое было заголовком созданного вами запроса на включение. Каждая служба хостинга может иметь несколько отличающийся шаблон, по которому они составляют это сообщение коммита по умолчанию;
- родителями коммита слияния M3 являются `7c09136` (радужный коммит) и `4c35a5c` (розовый коммит). Как обычно, хеши коммитов в вашем репозитории будут отличаться от хешей в этой книге, поскольку хеши коммитов уникальны.

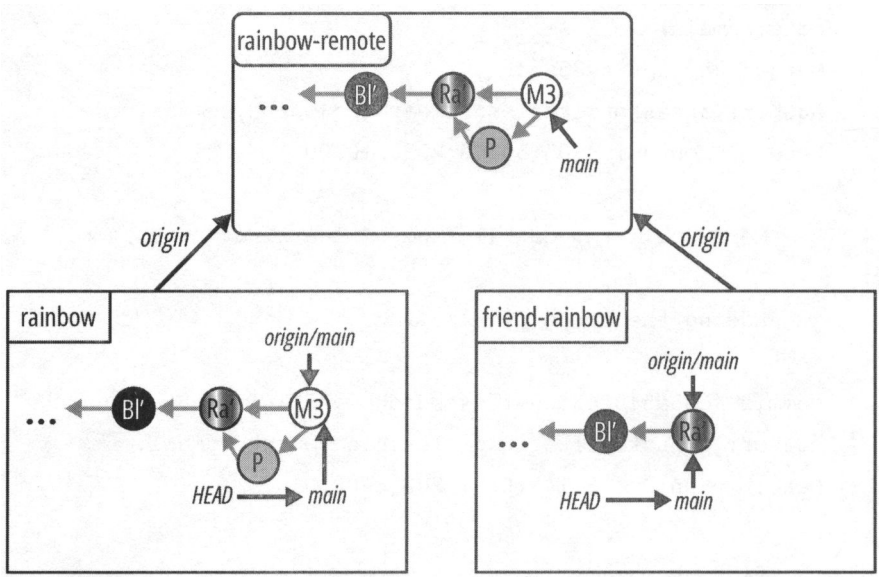
Состояние проекта Rainbow после действий 12.7 показано на диаграмме 12.7.

На что обратить внимание:

- в репозитории `rainbow` ветка `main` указывает на последний коммит слияния M3.

Далее, выполняя действия 12.8, ваш друг синхронизирует свою локальную ветку `main` с изменениями из удаленного репозитория.

ДИАГРАММА 12.7



Проект Rainbow после того, как вы перенесли изменения из удаленной ветки `main` в локальную ветку `main` репозитория `rainbow` и удалили локальную ветку `topic` и ветку удаленного отслеживания `origin/topic`.

ДЕЙСТВИЯ 12.8

1 `friend-rainbow $ git pull`

```
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 1), reused 2 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 831 bytes | 166.00 KiB/s, done.
From github.com:gitlearningjourney/rainbow-remote
   7c09136..2f833d6  main       -> origin/main
Updating 7c09136..2f833d6
Fast-forward
 othercolors.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

2 `friend-rainbow $ git log`

```
commit 2f833d6fa783882c5f832da9e1eafe6d405d3468 (HEAD -> main,
origin/main, origin/HEAD)
```

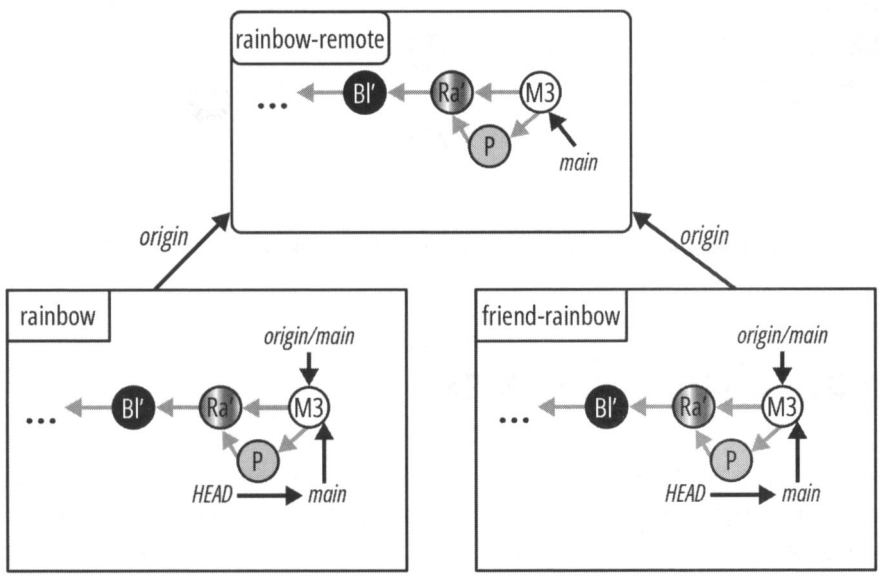
```
Merge: 7c09136 4c35a5c
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Mon Jul 4 05:50:08 2022 +0200

    Merge pull request #1 from gitlearningjourney/topic

    Adding the color pink

commit 4c35a5c02c3dc03f044cbdfdbb0ae55161af6a86
Author: annaskoulikari <gitlearningjourney@gmail.com>
Date:   Sun Jul 3 14:16:01 2022 +0200

    pink
```

На что обратить внимание:

- ваш друг обновил свою локальную ветку `main`.

Это показано на диаграмме 12.8.

ДИАГРАММА 12.8

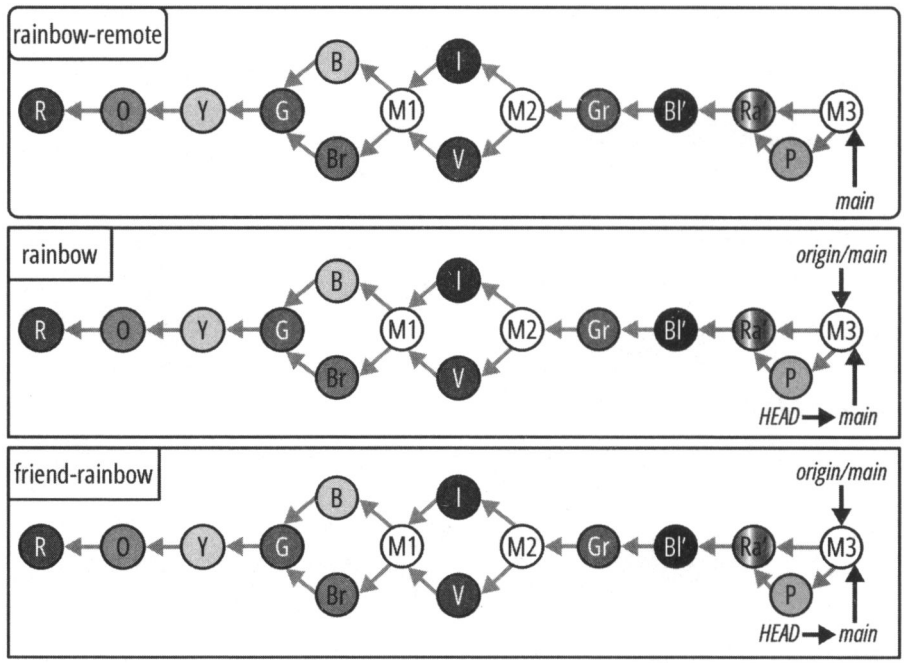


Проект Rainbow после того, как ваш друг перенес изменения из удаленной ветки `main` в локальную ветку `main` в репозитории `friend-rainbow`.

Состояние локальных и удаленных репозиториев

Вы подошли к концу обучения работе с Git. На диаграмме 12.9 показано состояние локальных и удаленных репозиториев в проекте `rainbow` со всеми коммитами, сделанными с *главы 1* до конца *главы 12*.

ДИАГРАММА 12.9



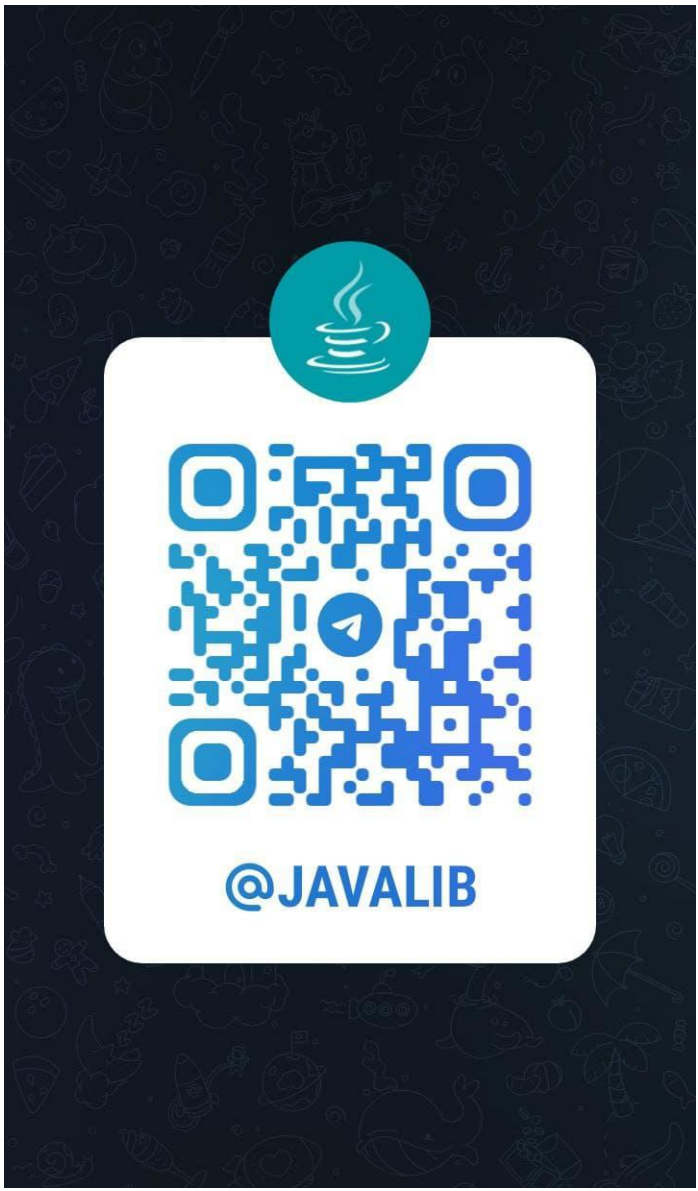
Проект Rainbow в конце *главы 12* со всеми изменениями, внесенными после *главы 1*.

Краткое содержание главы

В этой главе вы узнали о запросах на включение. Мы обсудили, почему они являются полезным инструментом для совместной работы над проектами Git с несколькими людьми, поскольку они облегчают общение и процесс проверки. Вы выполнили все девять шагов процесса запроса на включение на практическом примере проекта Rainbow. Работая над этим примером, вы стали свидетелями того, как по умолчанию запросы на включение объединяются в режиме обычного (не ускорен-

ного) слияния, которое создает коммит слияния даже в случае, когда истории веток, участвующих в слиянии, не разошлись. Вы также узнали о простом трюке, позволяющем Git сгенерировать команду, которую можно скопировать и применить для отправки ветки в удаленный репозиторий и определения для нее восходящей ветки за один раз.

Прочтите эпилог, чтобы ознакомиться с некоторыми заключительными мыслями, и обязательно загляните в приложения, где вы найдете полезный справочный материал.



Эпилог

Поздравляю — вы завершили знакомство с Git! Я надеюсь, что эта книга помогла вам в вашем путешествии по увлекательному миру Git и теперь у вас есть прочное понимание основ работы системы контроля версий.

Хотя мы и подошли к концу книги, это только начало вашего пути в Git. Ваш следующий шаг — начать использовать Git для контроля версий ваших проектов. По ходу дела вы научитесь применять гораздо больше возможностей, существующих в мире Git. Вам также необходимо выяснить, какой рабочий процесс Git подойдет вам или вашей команде, или, если вы пришли в действующую команду, узнать, как устроен в ней рабочий процесс Git.

Теперь, когда вы прошли весь путь обучения, работая над проектом Rainbow, начиная с *главы 1* и заканчивая *главой 12*, вы можете вернуться к содержанию конкретной главы или попробовать выполнить некоторые практические упражнения еще раз. В этом случае смело обращайтесь к *приложению 1*, чтобы узнать, как подготовить рабочее состояние проекта Rainbow, которое вам понадобится, чтобы начать работу с произвольной главы, не проходя предыдущие этапы.

Наконец, если эта книга оказалась для вас полезной, не стесняйтесь поделиться ею с кем-нибудь еще, чтобы она помогла им в изучении Git.

[Приложение 1]

Подготовка исходного состояния проекта в начале главы

Эта книга основана на практических обучающих примерах, и предполагается, что вы будете знакомиться с ними по порядку, от *главы 1* до *главы 12*. На протяжении всей книги вы будете работать над проектом Rainbow и узнаете, как устроен Git.

Однако в некоторых ситуациях может возникнуть необходимость или желание начать с определенной главы, не выполняя все предыдущие действия. Например:

- вы выполнили упражнения всей книги один раз и хотите повторить их, начиная с конкретной главы;
- что-то пошло не так в проекте Rainbow в предыдущей главе, вам не удалось это исправить, и вы хотите продолжить работу с новой главы так, будто ничего не случилось.

В этом случае вы можете воспользоваться инструкциями в соответствующем разделе этого приложения, чтобы воссоздать проект Rainbow приблизительно в том состоянии, которое должно быть в начале главы, с которой вы хотели бы поработать. Вы сформируете состояние репозиториев, необходимое для продолжения работы над проектом Rainbow из выбранной вами главы. Для получения дополнительной информации и подробных объяснений шагов каждого последующего действия обратитесь к содержанию соответствующей главы.

ПРИМЕЧАНИЕ

В *главах 9–11* вы не будете заново создавать все коммиты, сделанные в предыдущих главах. Вы только заново создадите коммит, аналогичный последнему коммиту, сделанному перед главой, с которой вы хотите начать. Это означает, что в ваших репозиториях проекта Rainbow будет меньше коммитов, чем в репозиториях проекта Rainbow, показанных на диаграммах непосредственно в самой главе. Имейте это в виду при работе с главами.

Все инструкции предполагают, что вы хотя бы один раз выполнили действия, описанные в *главе 1*, на компьютере, который используете для упражнений проекта Rainbow. Сюда входят установка Git, выбор текстового редактора и настройка переменных `user.name` и `user.email`. Если это не так, вам нужно будет начать с выполнения необходимых действий в *разд. "Предварительная настройка для всех глав" далее в этом приложении*.

В инструкциях для репозиториев используются те же имена, что и в остальной части книги: `rainbow`, `friend-rainbow` и `rainbow-remote`. Если у вас уже есть эти репозитории на вашем локальном компьютере и на вашем хостинге, вам придется использовать немного другие имена, например `rainbow1`, `friend-rainbow1` и `rainbow-remote1`. Я рекомендую вам придерживаться похожих названий, поскольку эти имена упоминаются на протяжении каждой главы.

Предварительная настройка для всех глав

Если у вас не установлен Git или не готов к использованию текстовый редактор, выполните действия П1.1. В противном случае вы можете пропустить этот шаг.

ДЕЙСТВИЯ П1.1

1. Перейдите в репозиторий книги по адресу **https://github.com/gitlearningjourney/learning-git** и следуйте инструкциям по скачиванию Git для вашей операционной системы.
2. Выберите текстовый редактор, который вам нравится. Если на вашем компьютере еще нет текстового редактора, скачайте его. Дополнительная информация приведена в *разд. "Подготовка текстового редактора" главы 1*.

Затем вы должны присвоить переменным конфигурации `user.name` и `user.email` соответствующие значения. Если вы еще этого не сделали, выполните действия П1.2. Дополнительную информацию о переменных конфигурации Git вы найдете в *разд. "Настройка конфигураций Git" главы 1*. Обязательно укажите свое имя и адрес электронной почты вместо заполнителей.

ДЕЙСТВИЯ П1.2

1. `$ git config --global user.name "<ваше_имя>"`
2. `$ git config --global user.email "<ваш_email>"`

Предварительная настройка для главы 2

Выполните действия П1.3, чтобы подготовить исходное состояние для работы, начиная с *главы 2*.

ДЕЙСТВИЯ П1.3

1. Откройте окно командной строки в приложении, которое вы предпочитаете.
2. `$ cd desktop`
3. `desktop $ mkdir rainbow`
4. `desktop $ cd rainbow`
5. Откройте каталог проекта `rainbow` в окне Проводника файловой системы.

Предварительная настройка для главы 3

Выполните действия П1.4, чтобы подготовить исходное состояние репозитория `rainbow` для работы, начиная с *главы 3*.

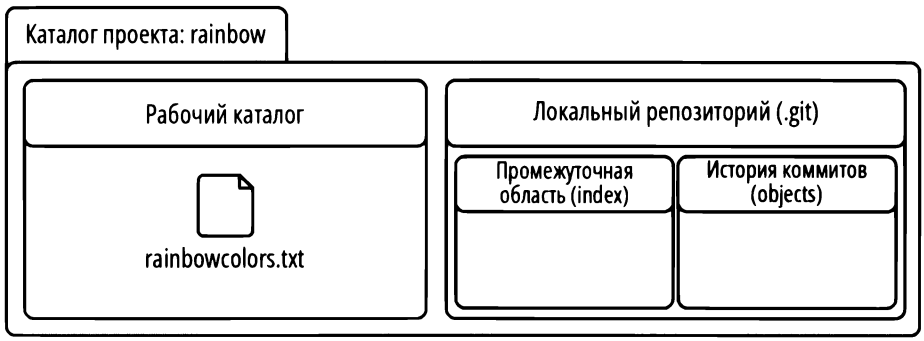
ДЕЙСТВИЯ П1.4

1. Откройте окно командной строки в приложении, которое вы предпочитаете.
2. `$ cd desktop`
3. `desktop $ mkdir rainbow`
4. `desktop $ cd rainbow`
5. `rainbow $ git init -b main`

   ```
   Initialized empty Git repository in
   /Users/annaskoulikari/desktop/rainbow/.git/
   ```
6. Запустите текстовый редактор и создайте файл с именем rainbowcolors.txt внутри рабочего каталога.
7. Добавьте в файл rainbowcolors.txt строку 1 "Красный — это первый цвет радуги." и сохраните файл.

В конце этого процесса созданный вами репозиторий `rainbow` будет содержать файл rainbowcolors.txt в рабочем каталоге, как показано на диаграмме П1.1.

ДИАГРАММА П1.1



Воссозданный репозиторий `rainbow`, чтобы начать работать с *главы 3*, минуя *главу 2*.

Предварительная настройка для главы 4

Выполните действия П1.5, чтобы подготовить исходное состояние репозитория `rainbow` для работы, начиная с *главы 4*.

ДЕЙСТВИЯ П1.5

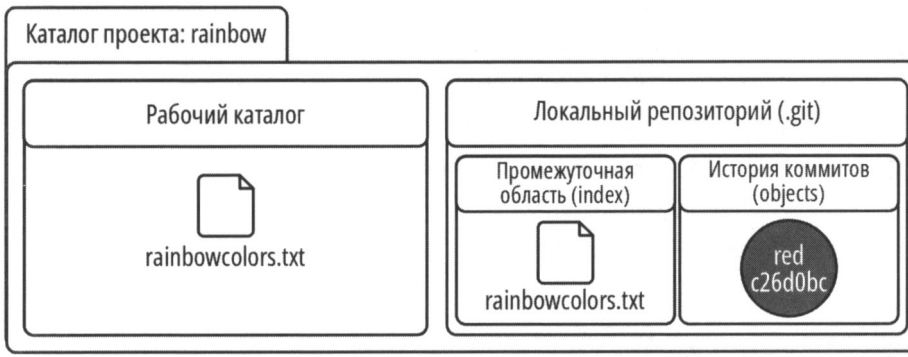
1. Откройте окно командной строки в приложении, которое вы предпочитаете.
2. `$ cd desktop`
3. `desktop $ mkdir rainbow`
4. `desktop $ cd rainbow`
5. `rainbow $ git init -b main`

   ```
   Initialized empty Git repository in
   /Users/annaskoulikari/desktop/rainbow/.git/
   ```
6. Запустите текстовый редактор и создайте файл с именем rainbowcolors.txt внутри рабочего каталога.
7. Добавьте в файл rainbowcolors.txt строку 1 "Красный — это первый цвет радуги." и сохраните файл.
8. `rainbow $ git add rainbowcolors.txt`
9. `rainbow $ git commit -m "red"`

   ```
   [main (root-commit) c26d0bc] red
    1 file changed, 1 insertion(+)
    create mode 100644 rainbowcolors.txt
   ```

В конце этого процесса созданный вами радужный репозиторий будет содержать красный коммит, как показано на диаграмме П1.2.

ДИАГРАММА П1.2



Воссозданный репозиторий `rainbow`, чтобы начать работать с *главы 4*, минуя *главу 3*.

Предварительная настройка для главы 5

Выполните действия П1.6, чтобы подготовить исходное состояние репозитория `rainbow` для работы, начиная с *главы 5*.

ДЕЙСТВИЯ П1.6

1. Откройте окно командной строки в приложении, которое вы предпочитаете.
2. `$ cd desktop`
3. `desktop $ mkdir rainbow`
4. `desktop $ cd rainbow`
5. `rainbow $ git init -b main`

   ```
   Initialized empty Git repository in
   /Users/annaskoulikari/desktop/rainbow/.git/
   ```
6. Запустите текстовый редактор и создайте файл с именем rainbowcolors.txt внутри рабочего каталога.
7. Добавьте в файл rainbowcolors.txt строку 1 "Красный — это первый цвет радуги." и сохраните файл.

8. `rainbow $ git add rainbowcolors.txt`
9. `rainbow $ git commit -m "red"`

```
[main (root-commit) c26d0bc] red
 1 file changed, 1 insertion(+)
 create mode 100644 rainbowcolors.txt
```

10. Добавьте в файл rainbowcolors.txt строку 2 "Оранжевый — это второй цвет радуги." и сохраните файл.
11. `rainbow $ git add rainbowcolors.txt`
12. `rainbow $ git commit -m "orange"`

```
[main 7acb333] orange
 1 file changed, 2 insertions(+), 1 deletion(-)
```

13. `rainbow $ git branch feature`
14. `rainbow $ git switch feature`

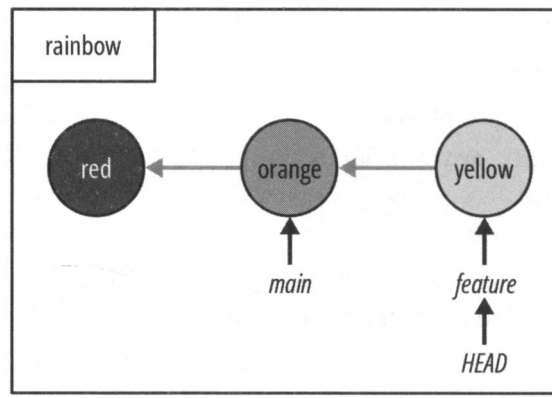
```
Switched to branch 'feature'
```

15. Добавьте в файл rainbowcolors.txt строку 3 "Желтый — это третий цвет радуги." и сохраните файл.
16. `rainbow $ git add rainbowcolors.txt`
17. `rainbow $ git commit -m "yellow"`

```
[feature fc8139c] yellow
 1 file changed, 2 insertions(+), 1 deletion(-)
```

В конце этого процесса созданный вами репозиторий `rainbow` будет содержать красный, оранжевый и желтый коммиты, как показано на диаграмме П1.3.

ДИАГРАММА П1.3



Воссозданный репозиторий `rainbow`, чтобы начать работать с *главы 5*, минуя *главу 4*.

Предварительная настройка для глав 6 и 7

Выполните действия П1.7, чтобы подготовить исходное состояние репозитория `rainbow` для начала работы с *главы 6* или *7*.

ДЕЙСТВИЯ П1.7

1. Откройте окно командной строки в приложении, которое вы предпочитаете.
2. `$ cd desktop`
3. `desktop $ mkdir rainbow`
4. `desktop $ cd rainbow`
5. `rainbow $ git init -b main`

```
Initialized empty Git repository in
/Users/annaskoulikari/desktop/rainbow/.git/
```

6. Запустите текстовый редактор и создайте файл с именем rainbowcolors.txt внутри рабочего каталога.
7. Добавьте в файл rainbowcolors.txt строку 1 "Красный — это первый цвет радуги." и сохраните файл.
8. `rainbow $ git add rainbowcolors.txt`
9. `rainbow $ git commit -m "red"`

```
[main (root-commit) c26d0bc] red
 1 file changed, 1 insertion(+)
 create mode 100644 rainbowcolors.txt
```

10. Добавьте в файл rainbowcolors.txt строку 2 "Оранжевый — это второй цвет радуги." и сохраните файл.
11. `rainbow $ git add rainbowcolors.txt`
12. `rainbow $ git commit -m "orange"`

```
[main 7acb333] orange
 1 file changed, 2 insertions(+), 1 deletion(-)
```

13. `rainbow $ git branch feature`
14. `rainbow $ git switch feature`

```
Switched to branch 'feature'
```

15. Добавьте в файл rainbowcolors.txt строку 3 "Желтый — это третий цвет радуги." и сохраните файл.
16. `rainbow $ git add rainbowcolors.txt`
17. `rainbow $ git commit -m "yellow"`

```
[feature fc8139c] yellow
 1 file changed, 2 insertions(+), 1 deletion(-)
```

18
```
rainbow $ git switch main
Switched to branch 'main'
```

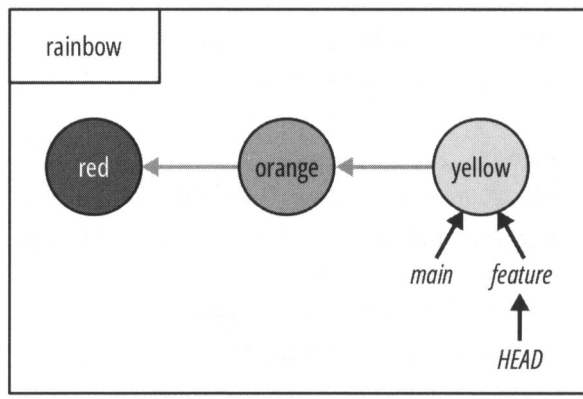
19
```
rainbow $ git merge feature
Updating 7acb333..fc8139c
Fast-forward
 rainbowcolors.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

ПРИМЕЧАНИЕ

Если вы используете этот раздел, чтобы начать с *главы 7*, и у вас еще нет учетной записи службы хостинга и/или данных аутентификации, настроенных для доступа по HTTPS или SSH, вам необходимо перейти к *главе 6* и выполнить упражнения из нее, прежде чем продолжить. Инструкций этого раздела недостаточно для выполнения упражнений *главы 7*.

В конце этого процесса созданный вами репозиторий `rainbow` будет содержать красный, оранжевый и желтый коммиты, как показано на диаграмме П1.4.

ДИАГРАММА П1.4



Воссозданный репозиторий `rainbow`, чтобы начать работать с *главы 6* или *7*, минуя *главу 5*.

Предварительная настройка для главы 8

Выполните действия П1.8, чтобы подготовить исходное состояние репозитория `rainbow` для начала работы с *главы 8*. На шаге 24 обязательно введите всю команду в одной строке.

ДЕЙСТВИЯ П1.8

1. Откройте окно командной строки в приложении, которое вы предпочитаете.
2. `$ cd desktop`
3. `desktop $ mkdir rainbow`
4. `desktop $ cd rainbow`
5. `rainbow $ git init -b main`

```
Initialized empty Git repository in
/Users/annaskoulikari/desktop/rainbow/.git/
```

6. Запустите текстовый редактор и создайте файл с именем rainbowcolors.txt внутри рабочего каталога.
7. Добавьте в файл rainbowcolors.txt строку 1 "Красный — это первый цвет радуги." и сохраните файл.
8. `rainbow $ git add rainbowcolors.txt`
9. `rainbow $ git commit -m "red"`

```
[main (root-commit) c26d0bc] red
 1 file changed, 1 insertion(+)
 create mode 100644 rainbowcolors.txt
```

10. Добавьте в файл rainbowcolors.txt строку 2 "Оранжевый — это второй цвет радуги." и сохраните файл.
11. `rainbow $ git add rainbowcolors.txt`
12. `rainbow $ git commit -m "orange"`

```
[main 7acb333] orange
 1 file changed, 2 insertions(+), 1 deletion(-)
```

13. `rainbow $ git branch feature`
14. `rainbow $ git switch feature`

```
Switched to branch 'feature'
```

15. Добавьте в файл rainbowcolors.txt строку 3 "Желтый — это третий цвет радуги." и сохраните файл.
16. `rainbow $ git add rainbowcolors.txt`
17. `rainbow $ git commit -m "yellow"`

```
[feature fc8139c] yellow
 1 file changed, 2 insertions(+), 1 deletion(-)
```

18. `rainbow $ git switch main`

```
Switched to branch 'main'
```

19
```
rainbow $ git merge feature
Updating 7acb333..fc8139c
Fast-forward
 rainbowcolors.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

20 Если у вас еще нет учетной записи службы хостинга и/или данных аутентификации, настроенных для доступа по HTTPS или SSH, вам необходимо перейти к *главе 6*, чтобы завершить эту часть процесса прямо сейчас. Затем продолжите эту цепочку действий.

21 Войдите в свою учетную запись хостинга.

22 Создайте удаленный репозиторий. Для получения дополнительной информации о том, как это сделать, перейдите в репозиторий книги (**https://github.com/gitlearningjourney/learning-git**) или воспользуйтесь документацией вашей службы хостинга.

При создании репозитория для этого упражнения:

- в качестве имени репозитория используйте `rainbow-remote`;
- вы можете сделать репозиторий общедоступным или частным; рекомендую сделать его частным;
- не добавляйте в новый репозиторий никаких файлов; например, не добавляйте файл README или .gitignore;
- если вас попросят указать имя ветки по умолчанию, можете оставить поле пустым или установить для него значение `main`.

23 Выполнив шаги по созданию удаленного репозитория, найдите его URL-адрес. Если вы не знаете, где его найти, обратитесь к документации вашего хостинга.

Доступны две версии URL-адреса: одна для доступа по HTTPS и одна для доступа по SSH. В примерах в этой книге встречаются два URL-адреса удаленного репозитория:

- HTTPS: **https://github.com/gitlearningjourney/rainbow-remote.git**;
- SSH: **git@github.com:gitlearningjourney/rainbow-remote.git**.

Скопируйте URL-адрес протокола, который вы выбрали для использования. В следующих шагах везде, где вы видите мой URL-адрес, вы должны использовать вместо него свой URL-адрес.

24
```
rainbow $ git remote add origin
https://github.com/gitlearningjourney/rainbow-remote.git
```

25
```
rainbow $ git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
```

```
Writing objects: 100% (9/9), 747 bytes | 373.00 KiB/s, done.
Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To github.com:gitlearningjourney/rainbow-remote.git
 * [new branch]      main -> main
```

26 `rainbow $ git switch feature`

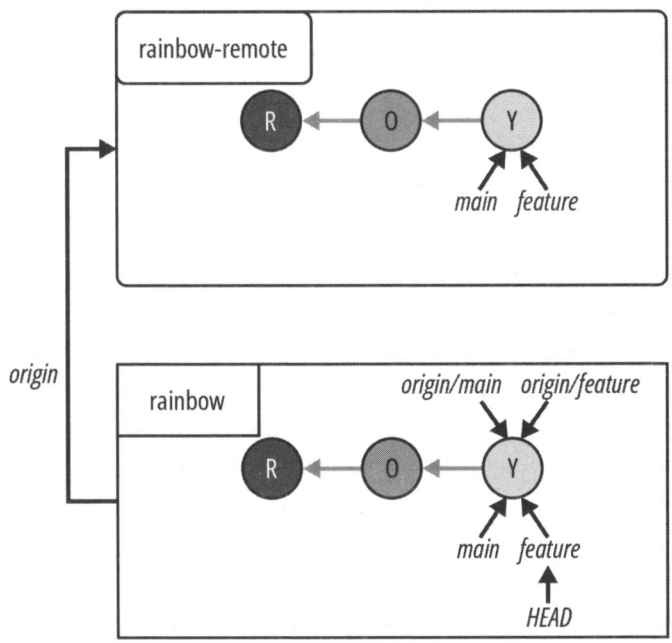
```
Switched to branch 'feature'
```

27 `rainbow $ git push origin feature`

```
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature' on GitHub by
visiting:
remote: https://github.com/gitlearningjourney/rainbow-
remote/pull/new/feature
remote:
To github.com:gitlearningjourney/rainbow-remote.git
 * [new branch]      feature -> feature
```

В конце этого процесса созданный вами проект Rainbow будет содержать красный, оранжевый и желтый коммиты во всех репозиториях, как показано на диаграмме П1.5.

ДИАГРАММА П1.5



Воссозданный проект Rainbow, позволяющий начать работу с *главы 8*, минуя *главу 7*.

Предварительная настройка для главы 9

Выполните действия П1.9, чтобы подготовить исходное состояние проекта Rainbow для начала работы с *главы 9*. На шагах 14 и 17 обязательно введите всю команду в одной строке.

ДЕЙСТВИЯ П1.9

1. Откройте окно командной строки в приложении, которое вы предпочитаете.
2. `$ cd desktop`
3. `desktop $ mkdir rainbow`
4. `desktop $ cd rainbow`
5. `rainbow $ git init -b main`

   ```
   Initialized empty Git repository in
   /Users/annaskoulikari/desktop/rainbow/.git/
   ```
6. Запустите текстовый редактор и создайте файл с именем rainbowcolors.txt внутри рабочего каталога проекта `rainbow`.
7. Добавьте в файл rainbowcolors.txt следующий текст:

   "Красный — это первый цвет радуги.

   Оранжевый — это второй цвет радуги.

   Желтый — это третий цвет радуги.

   Зеленый — это четвертый цвет радуги."

   и сохраните файл.
8. `rainbow $ git add rainbowcolors.txt`
9. `rainbow $ git commit -m "green"`

   ```
   [main (root-commit) 4e59074] "green"
    1 file changed, 4 insertions(+)
    create mode 100644 rainbowcolors.txt
   ```
10. Если у вас еще нет учетной записи службы хостинга и/или данных аутентификации, настроенных для доступа по HTTPS или SSH, вам необходимо перейти к *главе 6*, чтобы завершить эту часть процесса прямо сейчас. Затем продолжите эту цепочку действий.
11. Войдите в свою учетную запись хостинга.
12. Создайте удаленный репозиторий. Для получения дополнительной информации о том, как это сделать, перейдите в репозиторий книги (**https://github.com/gitlearningjourney/learning-git**) или воспользуйтесь документацией вашей службы хостинга.

При создании репозитория для этого упражнения:

- в качестве имени репозитория используйте `rainbow-remote`;
- вы можете сделать репозиторий общедоступным или частным; рекомендую сделать его частным;
- не добавляйте в новый репозиторий никаких файлов; например, не добавляйте файл README или .gitignore;
- если вас попросят указать имя ветки по умолчанию, можете оставить поле пустым или установить для него значение `main`.

13 Выполнив шаги по созданию удаленного репозитория, найдите его URL-адрес. Если вы не знаете, где его найти, обратитесь к документации вашего хостинга.

Доступны две версии URL-адреса: одна для доступа по HTTPS и одна для доступа по SSH. В примерах в этой книге встречаются два URL-адреса удаленного репозитория:

- HTTPS: **https://github.com/gitlearningjourney/rainbow-remote.git**;
- SSH: **git@github.com:gitlearningjourney/rainbow-remote.git**.

Скопируйте URL-адрес протокола, который вы выбрали для использования. В следующих шагах везде, где вы видите мой URL-адрес, вы должны использовать вместо него свой URL-адрес.

14
```
rainbow $ git remote add origin
https://github.com/gitlearningjourney/rainbow-remote.git
```

15
```
rainbow $ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 311 bytes | 311.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:gitlearningjourney/rainbow-remote.git
 * [new branch] main -> main
```

16 Откройте новое окно командной строки, чтобы перейти в каталог рабочего стола и начать имитировать действия вашего друга.

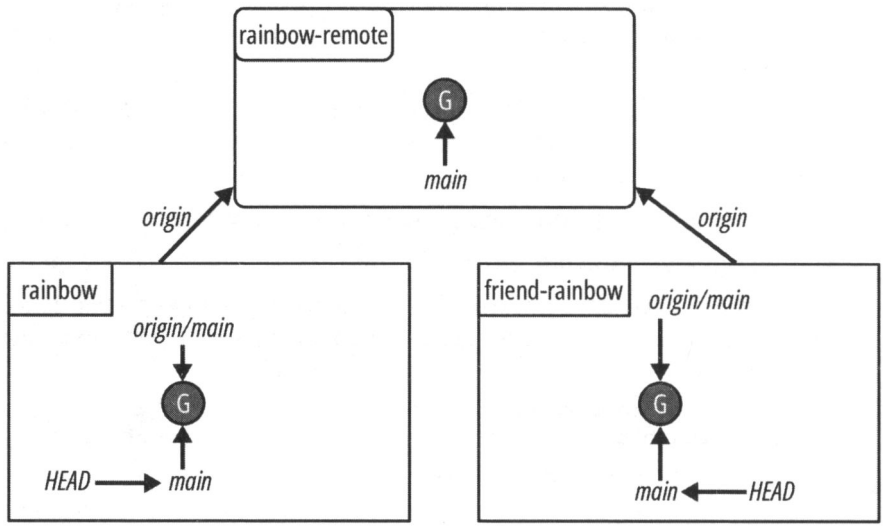
17
```
desktop $ git clone
https://github.com/gitlearningjourney/rainbow-remote.git friend-
rainbow
Cloning into 'friend-rainbow'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

18 `desktop $ cd friend-rainbow`

19 Откройте каталог friend-rainbow в новом окне файловой системы.

В конце этого процесса минимальная структура созданного вами проекта Rainbow будет содержать единственный зеленый коммит во всех репозиториях, как показано на диаграмме П1.6.

ДИАГРАММА П1.6



Воссозданный проект Rainbow, позволяющий начать работу с *главы 9*, минуя *главу 8*.

Предварительная настройка для главы 10

Выполните действия П1.10, чтобы подготовить исходное состояние проекта Rainbow для начала работы с *главы 10*. На шагах 15 и 19 обязательно введите всю команду в одной строке.

ДЕЙСТВИЯ П1.10

1 Откройте окно командной строки в приложении, которое вы предпочитаете.

2 `$ cd desktop`

3 `desktop $ mkdir rainbow`

4 `desktop $ cd rainbow`

5. `rainbow $ git init -b main`

   ```
   Initialized empty Git repository in
   /Users/annaskoulikari/desktop/rainbow/.git/
   ```

6. Запустите текстовый редактор и создайте файлы с именами rainbowcolors.txt и othercolors.txt внутри рабочего каталога проекта `rainbow`.

7. Добавьте в файл rainbowcolors.txt следующий текст

   "Красный — это первый цвет радуги.

   Оранжевый — это второй цвет радуги.

   Желтый — это третий цвет радуги.

   Зеленый — это четвертый цвет радуги.

   Голубой — это пятый цвет радуги."

   и сохраните файл.

8. Добавьте в файл othercolors.txt текст "Коричневый не является цветом радуги." и сохраните файл.

9. `rainbow $ git add rainbowcolors.txt othercolors.txt`

10. `rainbow $ git commit -m "fake merge commit 1"`

    ```
    2 files changed, 6 insertions(+)
    create mode 100644 othercolors.txt
    create mode 100644 rainbowcolors.txt
    ```

11. Если у вас еще нет учетной записи службы хостинга и/или данных аутентификации, настроенных для доступа по HTTPS или SSH, вам необходимо перейти к *главе 6*, чтобы завершить эту часть процесса прямо сейчас. Затем продолжите эту цепочку действий.

12. Войдите в свою учетную запись хостинга.

13. Создайте удаленный репозиторий. Для получения дополнительной информации о том, как это сделать, перейдите в репозиторий книги (**https://github.com/gitlearningjourney/learning-git**) или воспользуйтесь документацией вашей службы хостинга.

    При создании репозитория для этого упражнения:

    - в качестве имени репозитория используйте `rainbow-remote`;
    - вы можете сделать репозиторий общедоступным или частным; рекомендую сделать его частным;
    - не добавляйте в новый репозиторий никаких файлов; например, не добавляйте файл README или .gitignore.
    - если вас попросят указать имя ветки по умолчанию, можете оставить поле пустым или установить для него значение `main`.

14 Выполнив шаги по созданию удаленного репозитория, найдите его URL-адрес. Если вы не знаете, где его найти, обратитесь к документации вашего хостинга.

Доступны две версии URL-адреса: одна для доступа по HTTPS и одна для доступа по SSH. В примерах в этой книге встречаются два URL-адреса удаленного репозитория:

- HTTPS: **https://github.com/gitlearningjourney/rainbow-remote.git**;
- SSH: **git@github.com:gitlearningjourney/rainbow-remote.git**.

Скопируйте URL-адрес протокола, который вы выбрали для использования. В следующих шагах везде, где вы видите мой URL-адрес, вы должны использовать вместо него свой URL-адрес.

15
```
rainbow $ git remote add origin
https://github.com/gitlearningjourney/rainbow-remote.git
```

16
```
rainbow $ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 394 bytes | 394.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:gitlearningjourney/rainbow-remote.git
  * [new branch] main -> main
```

17
```
rainbow $ git branch -u origin/main
branch 'main' set up to track 'origin/main'.
```

18 Откройте новое окно командной строки, чтобы перейти в каталог рабочего стола и начать имитировать действия вашего друга.

19
```
desktop $ git clone
https://github.com/gitlearningjourney/rainbow-remote.git friend-
rainbow
Cloning into 'friend-rainbow'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

20
```
desktop $ cd friend-rainbow
```

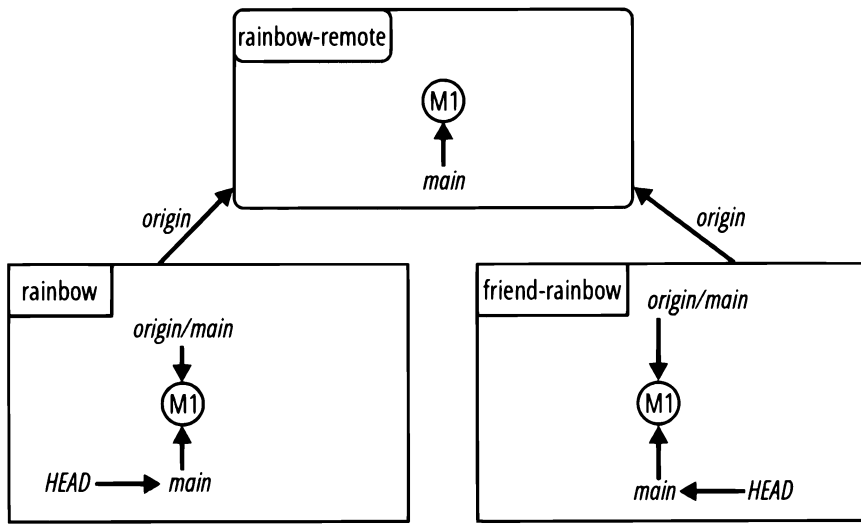
21 Откройте каталог friend-rainbow в новом окне файловой системы.

В конце этого процесса минимальная структура созданного вами проекта Rainbow будет содержать единственный коммит — фиктивный коммит слияния 1 (M1), во всех репозиториях, как показано на диаграмме П1.7.

ПРИМЕЧАНИЕ

Последним коммитом, созданным в *главе 9*, был коммит слияния 1 (M1), у которого было два родителя. Ваш коммит будет первым и единственным в репозиториях, поэтому у него не будет родительских коммитов. Следовательно, это фиктивный коммит слияния.

ДИАГРАММА П1.7



Воссозданный проект Rainbow, позволяющий начать работу с *главы 10*, минуя *главу 9*.

Предварительная настройка для главы 11

Выполните действия П.11, чтобы подготовить исходное состояние проекта Rainbow для работы, начиная с *главы 11*. На шагах 15 и 19 обязательно введите всю команду в одной строке.

ДЕЙСТВИЯ П.11

1. Откройте окно командной строки в приложении, которое вы предпочитаете.
2. `$ cd desktop`

3. `desktop $ mkdir rainbow`
4. `desktop $ cd rainbow`
5. `rainbow $ git init -b main`

```
Initialized empty Git repository in
/Users/annaskoulikari/desktop/rainbow/.git/
```

6. Запустите текстовый редактор и создайте файлы с именами rainbowcolors.txt и othercolors.txt внутри рабочего каталога проекта `rainbow`.
7. Добавьте в файл rainbowcolors.txt следующий текст

   "Красный — это первый цвет радуги.

   Оранжевый — это второй цвет радуги.

   Желтый — это третий цвет радуги.

   Зеленый — это четвертый цвет радуги.

   Голубой — это пятый цвет радуги.

   Синий — это шестой цвет радуги.

   Фиолетовый — это седьмой цвет радуги."

   и сохраните файл.
8. Добавьте в файл othercolors.txt текст "Коричневый не является цветом радуги." и сохраните файл.
9. `rainbow $ git add rainbowcolors.txt othercolors.txt`
10. `rainbow $ git commit -m "fake merge commit 2"`

```
[main (root-commit) 32fa0b7] fake merge commit 2
 2 files changed, 8 insertions(+)
 create mode 100644 othercolors.txt
 create mode 100644 rainbowcolors.txt
```

11. Если у вас еще нет учетной записи службы хостинга и/или данных аутентификации, настроенных для доступа по HTTPS или SSH, вам необходимо перейти к *главе 6*, чтобы завершить эту часть процесса прямо сейчас. Затем продолжите эту цепочку действий.
12. Войдите в свою учетную запись хостинга.
13. Создайте удаленный репозиторий. Для получения дополнительной информации о том, как это сделать, перейдите в репозиторий книги (**https://github.com/gitlearningjourney/learning-git**) или воспользуйтесь документацией вашей службы хостинга.

    При создании репозитория для этого упражнения:

    - в качестве имени репозитория используйте `rainbow-remote`;
    - вы можете сделать репозиторий общедоступным или частным; рекомендую сделать его частным;

- не добавляйте в новый репозиторий никаких файлов; например, не добавляйте файл README или .gitignore;
- если вас попросят указать имя ветки по умолчанию, можете оставить поле пустым или установить для него значение `main`.

14 Выполнив шаги по созданию удаленного репозитория, найдите его URL-адрес. Если вы не знаете, где его найти, обратитесь к документации вашего хостинга.

Доступны две версии URL-адреса: одна для доступа по HTTPS и одна для доступа по SSH. В примерах в этой книге встречаются два URL-адреса удаленного репозитория:

- HTTPS: **https://github.com/gitlearningjourney/rainbow-remote.git**;
- SSH: **git@github.com:gitlearningjourney/rainbow-remote.git**.

Скопируйте URL-адрес протокола, который вы выбрали для использования. В следующих шагах везде, где вы видите мой URL-адрес, вы должны использовать вместо него свой URL-адрес.

15
```
rainbow $ git remote add origin
https://github.com/gitlearningjourney/rainbow-remote.git
```

16
```
rainbow $ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 413 bytes | 413.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:gitlearningjourney/rainbow-remote.git
* [new branch] main -> main
```

17
```
rainbow $ git branch -u origin/main
branch 'main' set up to track 'origin/main'.
```

18 Откройте новое окно командной строки, чтобы перейти в каталог рабочего стола и начать имитировать действия вашего друга.

19
```
desktop $ git clone
https://github.com/gitlearningjourney/rainbow-remote.git friend-
rainbow
Cloning into 'friend-rainbow'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

20 `desktop $ cd friend-rainbow`

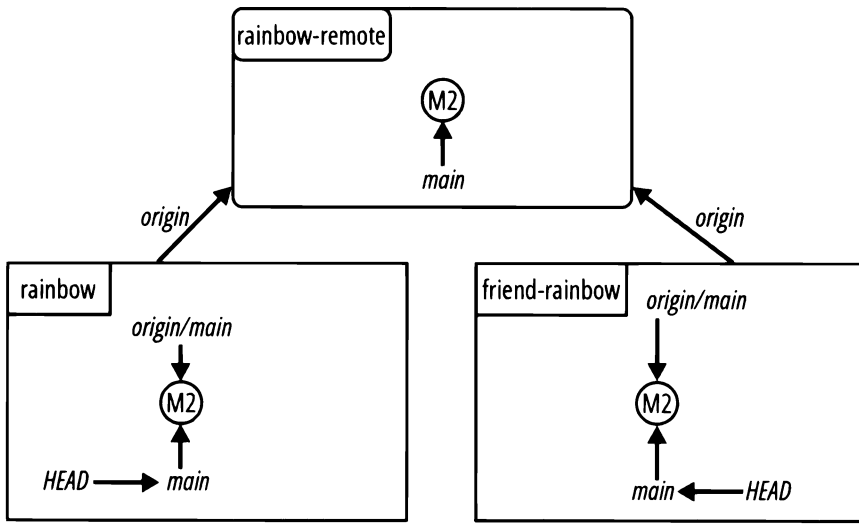
21 Откройте каталог friend-rainbow в новом окне файловой системы.

В конце этого процесса минимальная структура созданного вами проекта Rainbow будет содержать единственный фиктивный коммит слияния 2 (M2) во всех репозиториях, как показано на диаграмме П1.8.

ПРИМЕЧАНИЕ

Последним коммитом, созданным в *главе 10*, был коммит слияния 2 (M2), у которого было два родителя. Ваш коммит будет первым и единственным в репозиториях, поэтому у него не будет родительских коммитов. Следовательно, это фиктивный коммит слияния.

ДИАГРАММА П1.8



Воссозданный проект Rainbow, позволяющий начать работу с *главы 11*, минуя *главу 10*.

Предварительная настройка для главы 12

Выполните действия П1.12, чтобы подготовить исходное состояние проекта Rainbow для работы, начиная с *главы 12*. На шагах 15 и 19 обязательно введите всю команду в одной строке.

ДЕЙСТВИЯ П1.12

1. Откройте окно командной строки в приложении, которое вы предпочитаете.
2. `$ cd desktop`
3. `desktop $ mkdir rainbow`
4. `desktop $ cd rainbow`
5. `rainbow $ git init -b main`

   ```
   Initialized empty Git repository in
   /Users/annaskoulikari/desktop/rainbow/.git/
   ```
6. Запустите текстовый редактор и создайте файлы с именами rainbowcolors.txt и othercolors.txt внутри рабочего каталога проекта `rainbow`.
7. Добавьте в файл rainbowcolors.txt следующий текст

   "Красный — это первый цвет радуги.

   Оранжевый — это второй цвет радуги.

   Желтый — это третий цвет радуги.

   Зеленый — это четвертый цвет радуги.

   Голубой — это пятый цвет радуги.

   Синий — это шестой цвет радуги.

   Фиолетовый — это седьмой цвет радуги.

   Это все цвета радуги."

   и сохраните файл.
8. Добавьте в файл othercolors.txt следующий текст

   "Коричневый не является цветом радуги.

   Серый не является цветом радуги.

   Черный не является цветом радуги." и сохраните файл.
9. `rainbow $ git add rainbowcolors.txt othercolors.txt`
10.
    ```
    rainbow $ git commit -m "fake merge commit 2"
    [main (root-commit) 32fa0b7] fake merge commit 2
     2 files changed, 11 insertions(+)
     create mode 100644 othercolors.txt
     create mode 100644 rainbowcolors.txt
    ```
11. Если у вас еще нет учетной записи службы хостинга и/или данных аутентификации, настроенных для доступа по HTTPS или SSH, вам необходимо перейти к *главе 6*, чтобы завершить эту часть процесса прямо сейчас. Затем продолжите эту цепочку действий.

12 Войдите в свою учетную запись хостинга.

13 Создайте удаленный репозиторий. Для получения дополнительной информации о том, как это сделать, перейдите в репозиторий книги (**https://github.com/gitlearningjourney/learning-git**) или воспользуйтесь документацией вашей службы хостинга.

При создании репозитория для этого упражнения:

- в качестве имени репозитория используйте `rainbow-remote`;
- вы можете сделать репозиторий общедоступным или частным; рекомендую сделать его частным;
- не добавляйте в новый репозиторий никаких файлов; например, не добавляйте файл README или .gitignore;
- если вас попросят указать имя ветки по умолчанию, можете оставить поле пустым или установить для него значение `main`.

14 Выполнив шаги по созданию удаленного репозитория, найдите его URL-адрес. Если вы не знаете, где его найти, обратитесь к документации вашего хостинга.

Доступны две версии URL-адреса: одна для доступа по HTTPS и одна для доступа по SSH. В примерах в этой книге встречаются два URL-адреса удаленного репозитория:

- HTTPS: **https://github.com/gitlearningjourney/rainbow-remote.git**;
- SSH: **git@github.com:gitlearningjourney/rainbow-remote.git**.

Скопируйте URL-адрес протокола, который вы выбрали для использования. В следующих шагах везде, где вы видите мой URL-адрес, вы должны использовать вместо него свой URL-адрес.

15
```
rainbow $ git remote add origin
https://github.com/gitlearningjourney/rainbow-remote.git
```

16
```
rainbow $ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 445 bytes | 445.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:gitlearningjourney/rainbow-remote.git
* [new branch] main -> main
```

17
```
rainbow $ git branch -u origin/main
branch 'main' set up to track 'origin/main'.
```

18 Откройте новое окно командной строки, чтобы перейти в каталог рабочего стола и начать имитировать действия вашего друга.

19 `desktop $ git clone https://github.com/gitlearningjourney/rainbow-remote.git friend-rainbow`

```
Cloning into 'friend-rainbow'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

20 `desktop $ cd friend-rainbow`

21 Откройте каталог friend-rainbow в новом окне файловой системы.

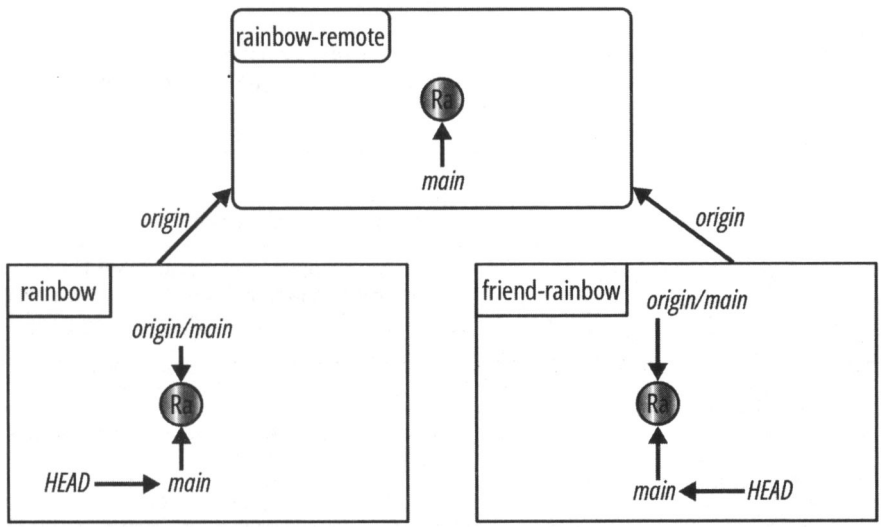
В конце этого процесса минимальная структура созданного вами проекта Rainbow будет содержать единственный радужный коммит во всех репозиториях, как показано на диаграмме П1.9.

ПРИМЕЧАНИЕ

Радужный коммит в начале *главы 12* — это перебазированный радужный коммит, поэтому на диаграммах в этой главе он помечен как Ra'.

Радужный коммит, полученный в результате выполнения действий П1.9, является обычным коммитом. Помните об этом, работая над главой. Просмотрите *приложение 3*, в котором кратко показаны визуальные обозначения книги, если вы хотите освежить знания.

ДИАГРАММА П1.9



Воссозданный проект Rainbow, позволяющий начать работу с *главы 12*, минуя *главу 11*.

[Приложение 2]

Краткий справочник по командам

Глава 1	
`clear`	Очистить экран командной строки
`pwd`	Показать путь к текущему рабочему каталогу
`ls`	Список видимых файлов и каталогов
`ls -a`	Список скрытых и видимых файлов и каталогов
`cd <путь_к_каталогу>`	Сменить рабочий каталог
`mkdir <имя_каталога>`	Создать каталог
`git config --global --list`	Перечислить переменные в глобальном файле конфигурации Git и их значения
`git config --global user.name "<имя>"`	Установить свое имя в глобальном файле конфигурации Git
`git config --global user.email "<email>"`	Установить свой адрес электронной почты в глобальном файле конфигурации Git

Глава 2	
`Git init`	Инициализировать репозиторий Git
`git init -b <имя_ветки>`	Инициализировать репозиторий Git и установить имя начальной ветки: `<имя_ветки>`

Глава 3	
`git status`	Показать состояние рабочего каталога и промежуточной области
`git add <имя_файла>`	Добавить один файл в промежуточную область
`git add <имя_файла> <имя_файла> ...`	Добавить несколько файлов в промежуточную область

Глава 3 *(продолжение)*	
`git add -A`	Добавить все файлы в рабочем каталоге, которые были отредактированы или изменены, в промежуточную область
`git commit -m "<сообщение>"`	Создать новый коммит с сообщением о коммите
`git log`	Показать список коммитов в обратном хронологическом порядке

Глава 4	
`git branch`	Вывести список локальных веток
`git branch <имя_новой_ветки>`	Создать ветку
`git switch <имя_новой_ветки>`	Переключить ветку
`git checkout <имя_новой_ветки>`	Переключить ветку

Глава 5	
`git merge <имя_ветки>`	Интегрировать изменения из одной ветки в другую ветку
`git log --all`	Показать список коммитов в обратном хронологическом порядке для всех веток локального репозитория
`git checkout <хеш_коммита>`	Проверить коммит
`git switch -c <имя_новой_ветки>`	Создать новую ветку и переключиться на нее
`git checkout -b < имя_новой_ветки>`	Создать новую ветку и переключиться на нее

Глава 7	
`git push`	Загрузить данные в удаленный репозиторий
`git remote add <короткое_имя> <URL>`	Добавить подключение к удаленному репозиторию с именем `<короткое_имя>` по адресу `<URL>`
`git remote`	Вывести список подключений к удаленному репозиторию, хранящихся в локальном репозитории, отсортированный по короткому имени
`git remote -v`	Вывести список подключений к удаленному репозиторию, хранящихся в локальном репозитории с короткими именами и URL-адресами
`git push <короткое_имя> <имя_ветки>`	Загрузить контент из `<имя_ветки>` в удаленный репозиторий `<короткое_имя>`
`git branch --all`	Вывести список локальных веток и веток удаленного отслеживания

Глава 8	
`git clone <URL> <имя_каталога>`	Клонировать удаленный репозиторий
`git push <короткое_имя> -d <имя_ветки>`	"Стереть" удаленную ветку и связанную с ней ветку удаленного отслеживания
`git branch -d <имя_ветки>`	Удалить локальную ветку
`git branch -vv`	Перечислить локальные ветки и их вышестоящие ветки, если они есть
`git fetch <короткое_имя>`	Загрузить данные из удаленного репозитория `<короткое_имя>`
`git fetch`	Загрузить данные из удаленного репозитория `origin`
`git fetch -p`	"Стереть" ветки удаленного отслеживания, соответствующие "стертым" удаленным веткам, и загрузить данные из удаленного репозитория

Глава 9	
`git branch -u <короткое_имя>/ <имя_ветки>`	Определить вышестоящую ветку для текущей локальной ветки
`git pull <короткое_имя> <имя_ветки>`	Получить и интегрировать изменения из удаленного репозитория `<короткое_имя>` для указанной ветки `<имя_ветки>`
`git pull`	Если для текущей ветки определена восходящая ветка, извлечь и интегрировать изменения из этой восходящей ветки

Глава 10	
`git merge --abort`	Остановить процесс слияния и вернуться в состояние до слияния

Глава 11	
`git rebase <имя_ветки>`	Повторно применить коммиты поверх другой ветки
`git restore --staged <имя_файла>`	Восстановить файл до другой версии файла в промежуточной области
`git rebase --continue`	Продолжить процесс перебазирования после разрешения конфликтов слияния
`git rebase --abort`	Остановить процесс перебазирования и вернуться в состояние до перебазирования

[Приложение 3]

Условные обозначения

Коммиты

На диаграммах в этой книге коммиты представлены кружками. Обычные коммиты в проекте Rainbow имеют один сплошной цвет и содержат полное имя или сокращение имени (единственным исключением является радужный коммит, который содержит все цвета радуги). Пример обычных коммитов показан на рис. П3.1.

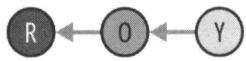


Рис. П3.1. Пример обычных коммитов проекта Rainbow с использованием цвета и сокращений имен коммитов, чтобы отличать их друг от друга

Если обычный коммит создается заново из-за операции перебазирования (обсуждаемой в *главе 11*), мы добавляем апостроф к имени повторного коммита, чтобы отличить его от исходного коммита. Пример такого коммита показан на рис. П3.2.

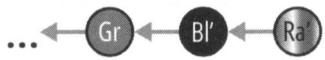


Рис. П3.2. Пример коммитов, созданных при перебазировании в проекте Rainbow (с апострофами в сокращениях, чтобы отличать их от исходных коммитов)

На протяжении всей книги коммиты слияния обозначаются белым кружком с толстой черной рамкой и буквой M (иногда они могут также содержать цифры). Пример коммита слияния показан на рис. П3.3.

Рис. П3.3. Пример коммита слияния

Обычные коммиты в проекте Book представлены синими кружками, где буквы алфавита используются для того, чтобы отличать их друг от друга. Пример коммита в проекте Book показан на рис. П3.4.

Рис. П3.4. Пример обычного проекта Book, в котором используются буквы алфавита, чтобы отличать коммиты друг от друга

Диаграммы Git

На диаграмме Git, которая впервые представлена в *главе 2*, изображен один каталог проекта Git. Он состоит из четырех областей Git: рабочий каталог, промежуточная область, история коммитов и локальный репозиторий. Рабочий каталог и промежуточная область могут содержать файлы, а история коммитов может содержать коммиты и ветки. Ветки и указатель `HEAD` (ссылка) представлены черными стрелками, а родительские ссылки между коммитами — серыми стрелками. Пример диаграммы Git показан на рис. П3.5.

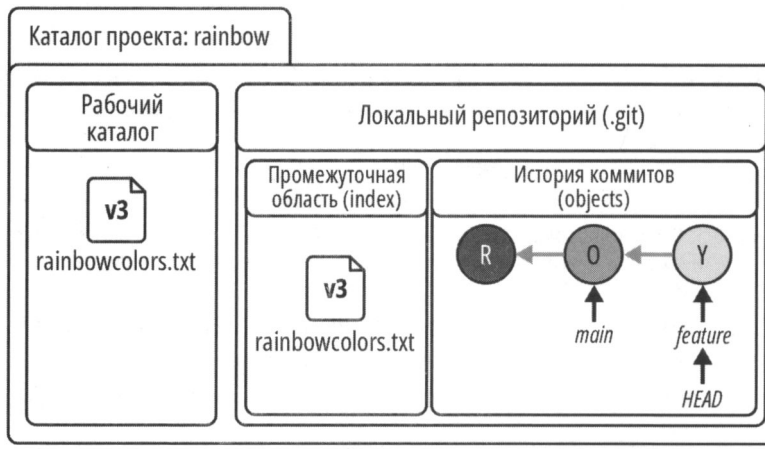


Рис. П3.5. Пример диаграммы Git с представлением рабочего каталога, промежуточной области, истории коммитов и локального репозитория, показывающей состояние проекта Rainbow в начале *главы 5*

Диаграммы репозитория

Диаграмма репозитория представляет либо один репозиторий, либо несколько репозиториев. Локальные репозитории всегда обозначены обычными прямоугольниками. Удаленные репозитории обозначены прямоугольниками со скругленными углами. Репозитории могут содержать коммиты и ветки. Диаграмма репозитория применяется, начиная с *главы 4*.

На рис. П3.6 вы можете видеть пример диаграммы с одним локальным репозиторием.

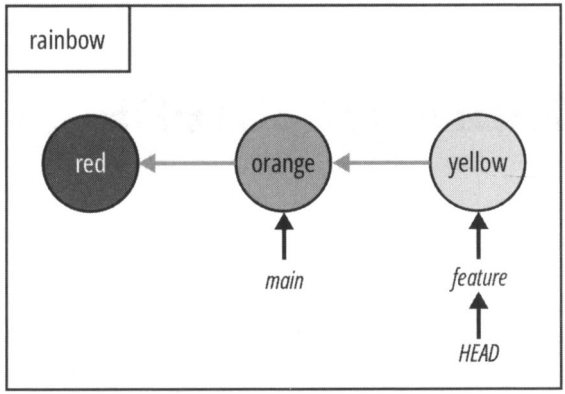


Рис. П3.6. Пример диаграммы с одним локальным репозиторием, отражающей состояние проекта Rainbow в конце *главы 4*

На рис. П3.7 показан пример диаграммы с двумя локальными репозиториями и одним удаленным репозиторием.

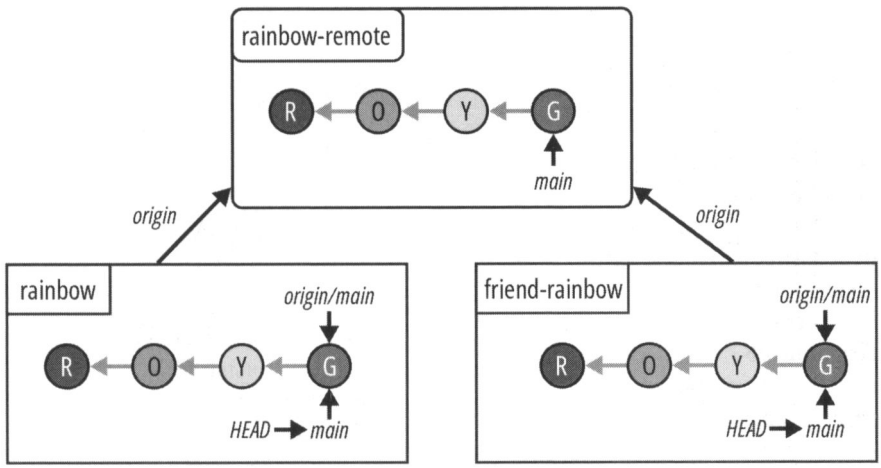


Рис. П3.7. Пример диаграммы с двумя локальными репозиториями и одним удаленным репозиторием, отражающей состояние проекта Rainbow в конце *главы 8*

Предметный указатель

Об изображении на обложке

Животное на обложке книги — золотистолобая листовка (*Chloropsis aurifrons*), обитающая на полуострове Индостан, на юго-западе Китая и в Юго-Восточной Азии.

Эта разновидность Листовок представляет собой небольшую ярко-зеленую птицу с черной головой и горлом, украшенную оранжево-золотым пятном на лбу. Длинные острые клювы и языки с кисточками на концах помогают этим птицам питаться насекомыми с коры и листьев деревьев. Этот представитель фауны — обычный обитатель тропических лесов и садов, в которых питается насекомыми, фруктами и даже — когда парит, как колибри, — нектаром.

Эта птица очень привязана к своей территории, и агрессивна по отношению к другим птицам и, как известно, способна нападать на крупных животных и людей, если чувствует угрозу. *Chloropsis aurifrons* размножается весной, строя небольшое чашеобразное гнездо из палочек и листьев. Самка откладывает два-три яйца, которые самец насиживает в течение двух недель. Прежде чем вылететь, птенцы остаются в гнезде еще две недели.

Изучаем Git

Прочитав книгу, вы изучите основы работы Git и сможете уверенно использовать этот популярный инструмент контроля версий.

Информация дается в простой и наглядной форме на практических примерах и упражнениях и вводится постепенно, чтобы не увязнуть в неизвестных терминах и понятиях.

Книга идеально подходит для всех, кому необходимо использовать Git для личных или профессиональных проектов: студентов курсов по программированию, начинающих разработчиков, специалистов по данным, технических писателей и многих других!

Вы узнаете, как:

- **Загрузить Git и инициализировать локальный репозиторий**
- **Добавлять файлы в промежуточную область и делать коммиты**
- **Создавать, переключать и удалять ветки**
- **Объединять и перемещать ветки**
- **Работать с удаленными репозиториями, включая клонирование, отправку, извлечение и выборку**
- **Использовать запросы на включение для совместной работы**

«Это чрезвычайно увлекательное, доступное и в то же время очень подробное руководство для тех, кто хочет получить хорошие практические знания Git».

— **Роберт С. Мартин (дядя Боб)**, автор бестселлера «Чистый код»

«Анна написала книгу, которая нужна сообществу Git. Вы можете изучить Git по этой книге легко и просто».

— **Бен Штрауб**, разработчик и соавтор книги «Pro Git!»