

- [About](#)
 - [Branching and Merging](#)
 - [Small and Fast](#)
 - [Distributed](#)
 - [Data Assurance](#)
 - [Staging Area](#)
 - [Free and Open Source](#)
 - [Trademark](#)
- [Documentation](#)
 - [Reference](#)
 - [Book](#)
 - [Videos](#)
 - [External Links](#)
- [Downloads](#)
 - [GUI Clients](#)
 - [Logos](#)
- [Community](#)

This book is available in [English](#).

Full translation available in

[български език](#),
[Deutsch](#),
[Español](#),
[Français](#),
[Ελληνικά](#),
[فارسی](#),
[Nederlands](#),
[Русский](#),
[Slovenščina](#),
[Tagalog](#),
[Українська](#),
[українська](#).

Partial translations available in

[Čeština](#),
[Македонски](#),
[Polski](#),
[Српски](#),
[Ўзбекча](#),
[українська](#).

Translations started for

[azərbaycan dili](#),
[Беларуская](#),
[فارسی](#),
[Indonesian](#),
[Italiano](#),
[Bahasa Melayu](#),
[Português \(Brasil\)](#),
[Português \(Portugal\)](#),
[Svenska](#),
[Türkçe](#).

The source of this book is [hosted on GitHub](#).
Patches, suggestions and comments are welcome.

[Chapters ▼](#)

1. [Введение](#)
 - 1.1.1 [О системе контроля версий](#)
 - 1.1.2 [Краткая история Git](#)
 - 1.1.3 [Основы Git](#)
 - 1.1.4 [Командная строка](#)
 - 1.1.5 [Установка Git](#)
 - 1.1.6 [Первоначальная настройка Git](#)
 - 1.1.7 [Как получить помощь?](#)
 - 1.1.8 [Заключение](#)
 2. [Основы Git](#)
 - 2.1.1 [Создание Git-репозитория](#)
 - 2.1.2 [Запись изменений в репозиторий](#)
 - 2.1.3 [Просмотр истории коммитов](#)
 - 2.1.4 [Операции отмены](#)
 - 2.1.5 [Работа с удалёнными репозиториями](#)
 - 2.1.6 [Работа с метками](#)
 - 2.1.7 [Псевдонимы в Git](#)
 - 2.1.8 [Заключение](#)
 3. [Ветвление в Git](#)
 - 3.1.1 [О ветвлении в двух словах](#)
 - 3.1.2 [Основы ветвления и слияния](#)
 - 3.1.3 [Управление ветками](#)
 - 3.1.4 [Работа с ветками](#)
 - 3.1.5 [Удалённые ветки](#)
 - 3.1.6 [Перебазирование](#)
 - 3.1.7 [Заключение](#)
 4. [Git на сервере](#)
 - 4.1.1 [Протоколы](#)
 - 4.1.2 [Установка Git на сервер](#)
 - 4.1.3 [Генерация открытого SSH ключа](#)
 - 4.1.4 [Настраиваем сервер](#)
 - 4.1.5 [Git-демон](#)
 - 4.1.6 [Умный HTTP](#)
 - 4.1.7 [GitWeb](#)
 - 4.1.8 [GitLab](#)
 - 4.1.9 [Git-хостинг](#)
 - 4.1.10 [Заключение](#)
 5. [Распределенный Git](#)
 - 5.1.1 [Распределенный рабочий процесс](#)
 - 5.1.2 [Участие в проекте](#)
 - 5.1.3 [Сопровождение проекта](#)
 - 5.1.4 [Заключение](#)
 6. [GitHub](#)
 - 6.1.1 [Настройка и конфигурация учетной записи](#)
 - 6.1.2 [Внесение собственного вклада в проекты](#)
 - 6.1.3 [Сопровождение проекта](#)
 - 6.1.4 [Управление организацией](#)
 - 6.1.5 [Scripting GitHub](#)
 - 6.1.6 [Заключение](#)
 7. [Инструменты Git](#)
 - 7.1.1 [Выбор ревизии](#)
 - 7.1.2 [Интерактивное индексирование](#)
 - 7.1.3 [Прибережение и очистка](#)
 - 7.1.4 [Подпись результатов вашей работы](#)
 - 7.1.5 [Поиск](#)
 - 7.1.6 [Исправление истории](#)
 - 7.1.7 [Раскрытие тайн reset](#)
 - 7.1.8 [Продвинутое слияние](#)
 - 7.1.9 [Rebase](#)
 - 7.1.10 [Обнаружение ошибок с помощью Git](#)
 - 7.1.11 [Подмодули](#)
 - 7.1.12 [Создание пакетов](#)
 - 7.1.13 [Замена](#)
 - 7.1.14 [Хранилище учётных данных](#)
 - 7.1.15 [Заключение](#)
 8. [Настройка Git](#)
 - 8.1.1 [Конфигурация Git](#)
 - 8.1.2 [Атрибуты Git](#)
 - 8.1.3 [Хуки в Git](#)
 - 8.1.4 [Пример принудительной политики Git](#)
 - 8.1.5 [Заключение](#)
 9. [Git и другие системы контроля версий](#)
 - 9.1.1 [Git как клиент](#)
 - 9.1.2 [Миграция на Git](#)
 - 9.1.3 [Заключение](#)
 10. [Git изнутри](#)
 - 10.1.1 [Сантехника и Фарфор](#)
 - 10.1.2 [Объекты Git](#)
 - 10.1.3 [Ссылки в Git](#)
 - 10.1.4 [Pack-файлы](#)
 - 10.1.5 [Спецификации ссылок](#)
 - 10.1.6 [Протоколы передачи данных](#)
 - 10.1.7 [Уход за репозиторием и восстановление данных](#)
 - 10.1.8 [Переменные среды](#)
 - 10.1.9 [Заключение](#)
- A1. [Appendix A: Git в других окружениях](#)
 - A1.1.1 [Графические интерфейсы](#)
 - A1.1.2 [Git в Visual Studio](#)
 - A1.1.3 [Git в Visual Studio Code](#)
 - A1.1.4 [Git в Eclipse](#)
 - A1.1.5 [Git в IntelliJ / PyCharm / WebStorm / PhpStorm / RubyMine](#)
 - A1.1.6 [Git в Sublime Text](#)
 - A1.1.7 [Git в Bash](#)
 - A1.1.8 [Git в Zsh](#)
 - A1.1.9 [Git в Powershell](#)
 - A1.1.10 [Заключение](#)
 - A2. [Appendix B: Встраивание Git в ваши приложения](#)
 - A2.1.1 [Git из командной строки](#)
 - A2.1.2 [Libgit2](#)
 - A2.1.3 [JGit](#)
 - A2.1.4 [go-git](#)
 - A2.1.5 [Dulwich](#)
 - A3. [Appendix C: Команды Git](#)
 - A3.1.1 [Настройка и конфигурация](#)
 - A3.1.2 [Клонирование и создание репозитория](#)
 - A3.1.3 [Основные команды](#)
 - A3.1.4 [Ветвление и слияния](#)
 - A3.1.5 [Совместная работа и обновление проектов](#)
 - A3.1.6 [Осмотр и сравнение](#)
 - A3.1.7 [Отладка](#)
 - A3.1.8 [Внесение исправлений](#)
 - A3.1.9 [Работа с помощью электронной почты](#)
 - A3.1.10 [Внешние системы](#)
 - A3.1.11 [Администрирование](#)
 - A3.1.12 [Низкоуровневые команды](#)

2nd Edition

3.2 Ветвление в Git - Основы ветвления и слияния

Основы ветвления и слияния

Давайте рассмотрим простой пример рабочего процесса, который может быть полезен в вашем проекте. Ваша работа построена так:

1. Вы работаете над сайтом.
2. Вы создаете ветку для новой статьи, которую вы пишете.
3. Вы работаете в этой ветке.

В этот момент вы получаете сообщение, что обнаружена критическая ошибка, требующая скорейшего исправления. Ваши действия:

1. Переключиться на основную ветку.
2. Создать ветку для добавления исправления.
3. После тестирования слить ветку содержащую исправление с основной веткой.
4. Переключиться назад в ту ветку, где вы пишете статью и продолжить работать.

Основы ветвления

Предположим, вы работаете над проектом и уже имеете несколько коммитов.

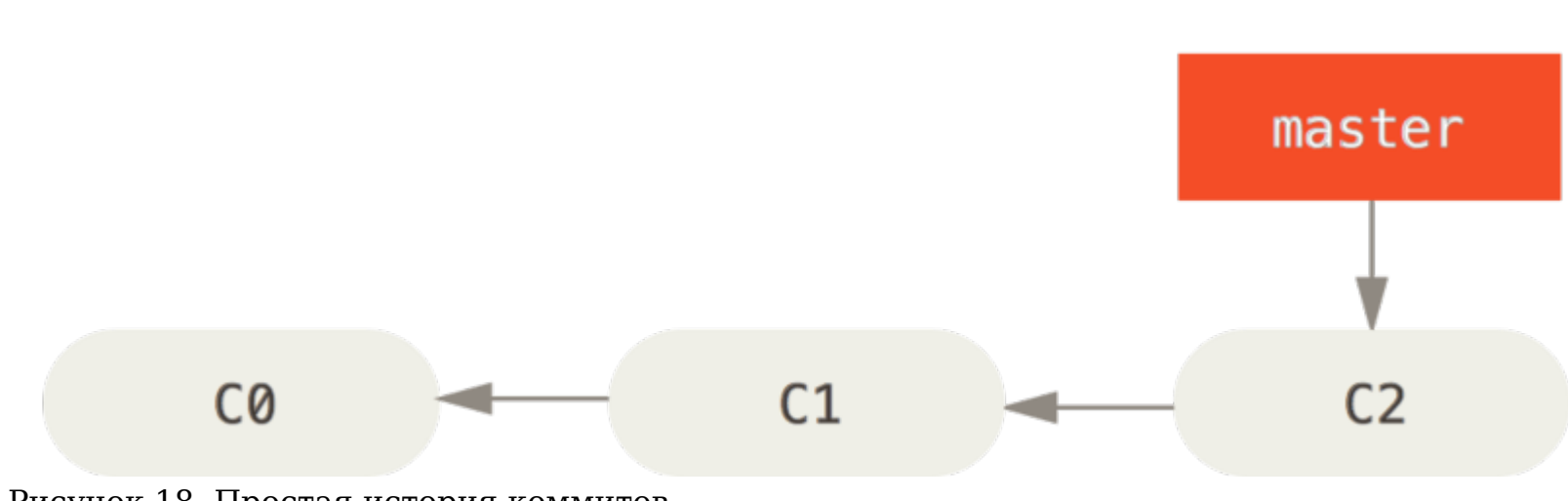


Рисунок 18. Простая история коммитов

Вы решаете, что теперь вы будете заниматься проблемой #53 из вашей системы отслеживания ошибок. Чтобы создать ветку и перейти к ней, можно выполнить команду `git checkout` с параметром `-b`:

```
$ git checkout -b iss53
Switched to a new branch "iss53"
```

Это тоже самое что и:

```
$ git branch iss53
$ git checkout iss53
```

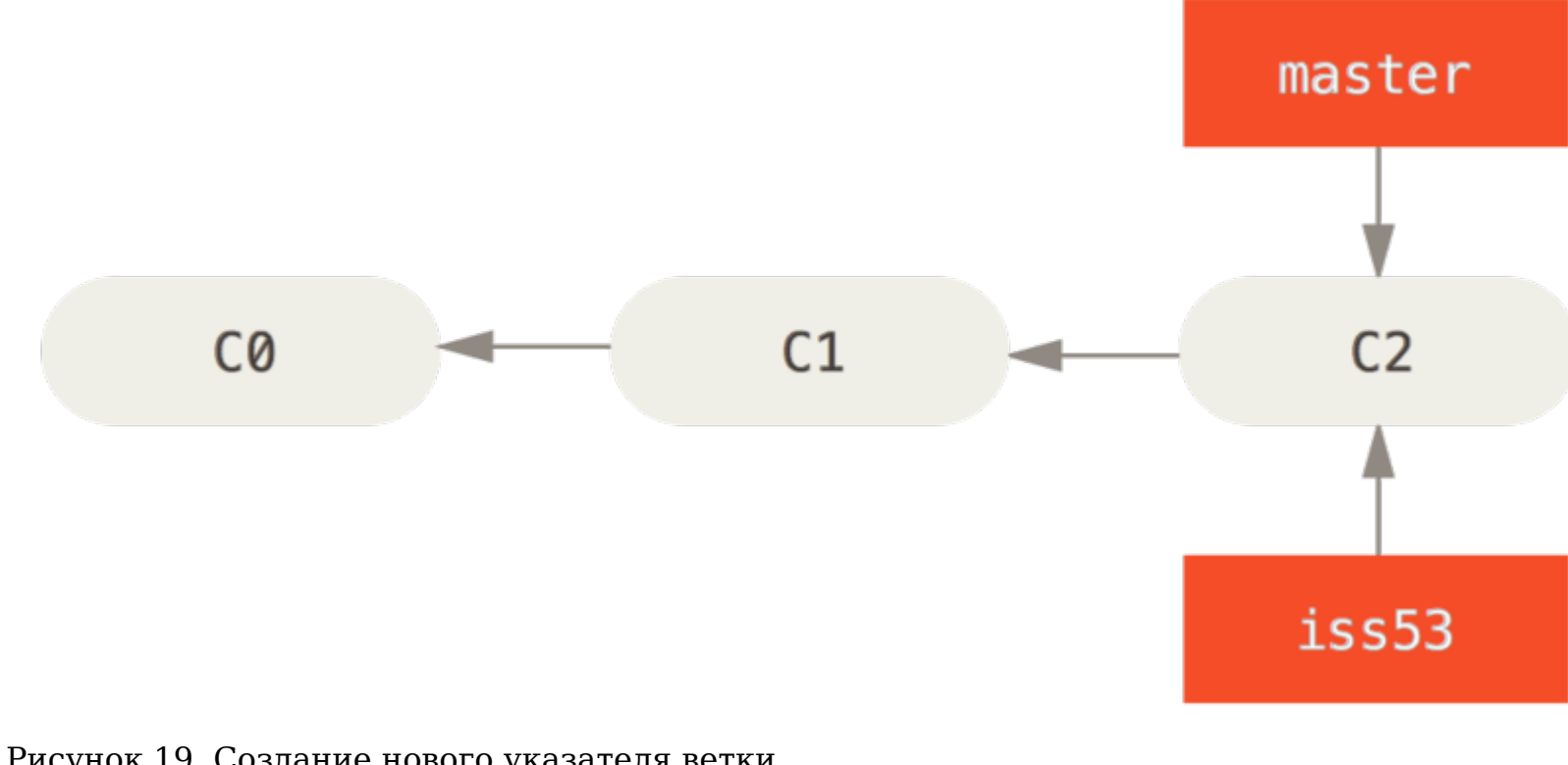


Рисунок 19. Создание нового указателя ветки

Вы работаете над своим сайтом и делаете коммиты. Это приводит к тому, что ветка `iss53` движется вперед, так как вы переключились на нее ранее (недо указывает на нее).

```
$ vim index.html
$ git commit -a -m 'added a new footer [issue 53]'
```

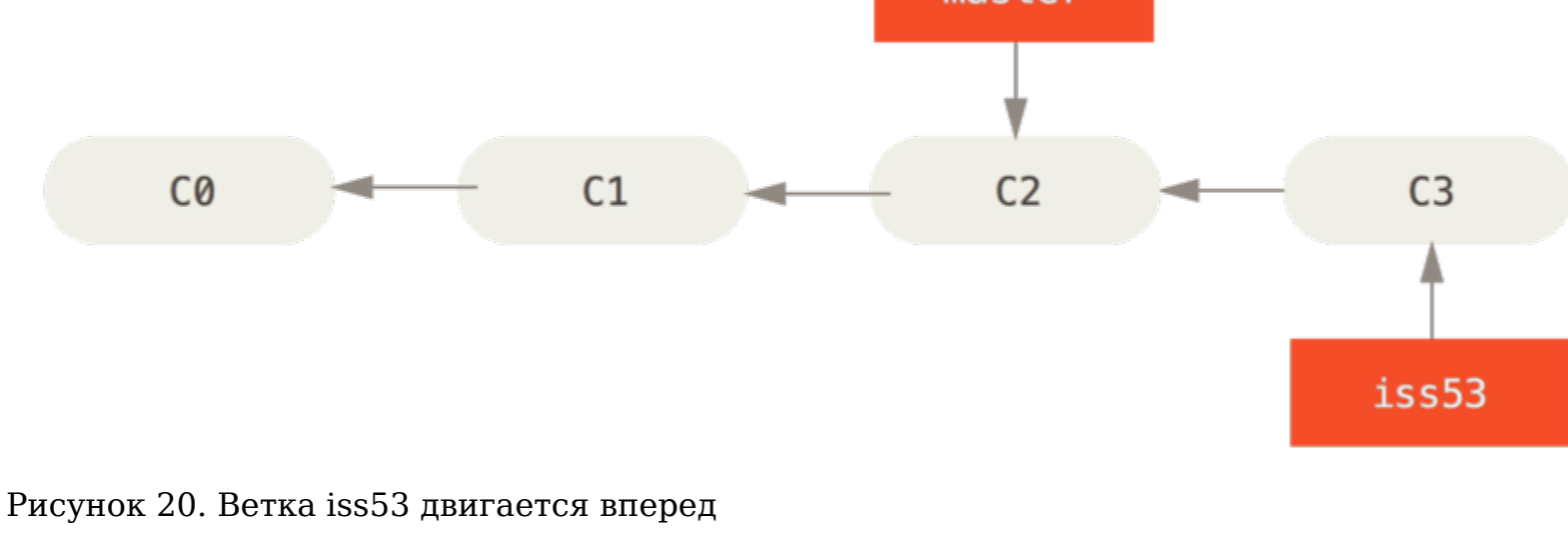


Рисунок 20. Ветка `iss53` двигается вперед

Тут вы получаете сообщение об обнаружении уязвимости на вашем сайте, которую нужно немедленно устранить. Благодаря Git, не требуется размещать это исправление вместе с тем, что вы сделали в `iss53`. Вам даже не придется прилагать усилий, чтобы откатить все эти изменения для начала работы над исправлением. Все, что вам нужно — переключиться на ветку `master`.

Но перед тем как сделать это — имейте в виду, что если ваш рабочий каталог либо область подготовленных файлов содержат изменения, не попавшие в коммит и конфликтующие с веткой, на которую вы хотите переключиться, то Git не позволит вам переключить ветки. Лучше всего переключаться из чистого рабочего состояния проекта. Есть способы обойти это (спрятать (stash) или исправить (amend) коммиты), но об этом мы поговорим позже в главе [Прибережение и очистка](#). Теперь предположим, что вы зафиксировали все свои изменения и можете переключиться на ветку `master`:

```
$ git checkout master
Switched to branch 'master'
```

С этого момента ваш рабочий каталог имеет точно такой же вид, какой был перед началом работы над проблемой #53, и вы можете сосредоточиться на работе над исправлением. Важно запомнить: когда вы переключаете ветки, Git возвращает состояние рабочего каталога к тому виду, какой он имел в момент последнего коммита в эту ветку. Он добавляет, удаляет и изменяет файлы автоматически, чтобы состояние рабочего каталога соответствовало тому, когда был сделан последний коммит.

Теперь вы можете перейти к написанию исправления. Давайте создадим новую ветку для исправления, в которой будем работать, пока не закончим исправление.

```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ vim index.html
$ git commit -a -m 'fixed the broken email address'
[hotfix 1fb7853] fixed the broken email address
1 file changed, 2 insertions(+)
```

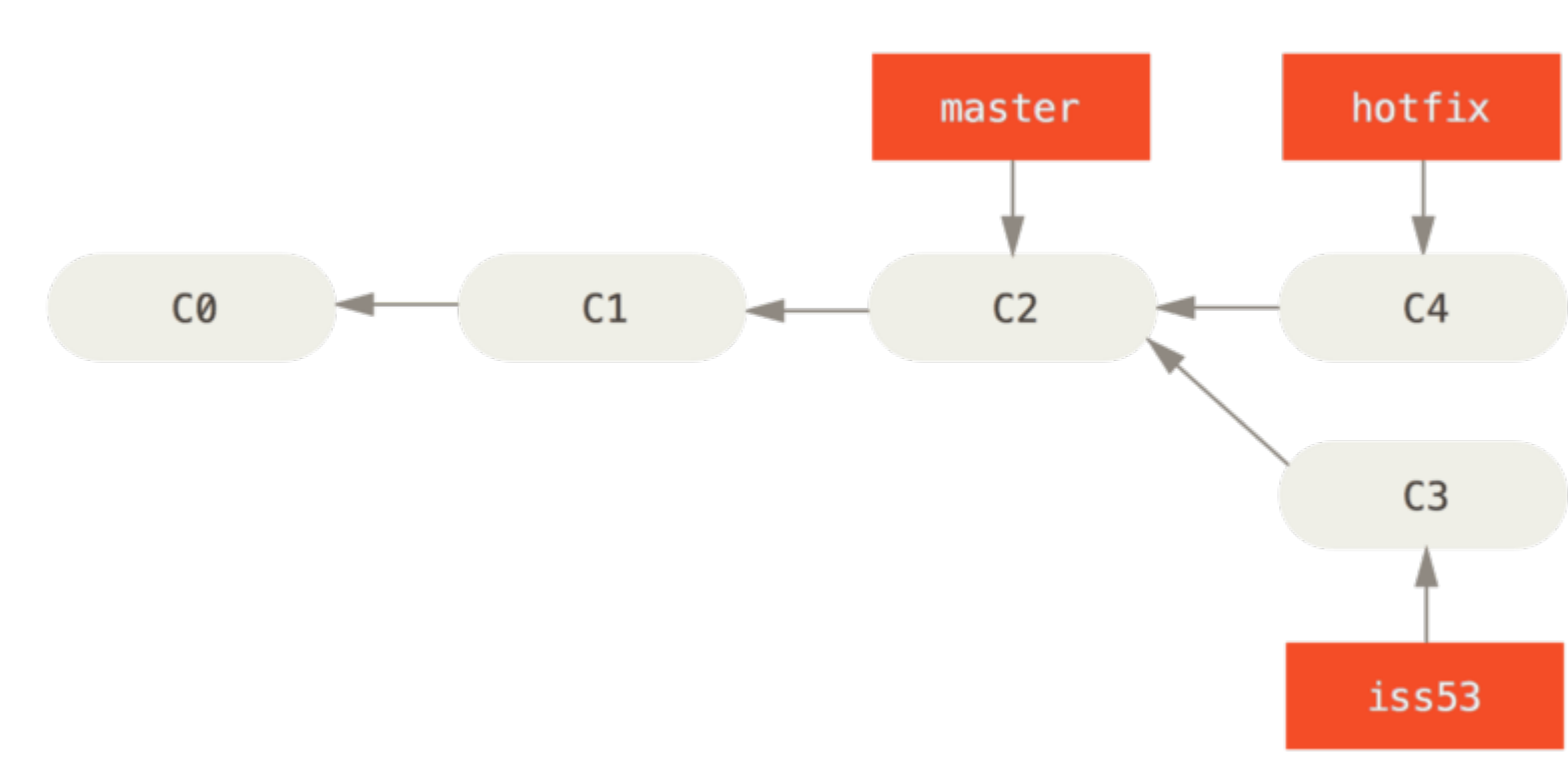


Рисунок 21. Ветка `hotfix` основана на ветке `master`

Вы можете прогнать тесты, чтобы убедиться, что ваше исправление делает именно то, что нужно. И если это так — выполнить слияние ветки `hotfix` с веткой `master` для включения изменений в продукт. Это делается командой `git merge`:

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
1 file changed, 2 insertions(+)
```

Заметили фразу “fast-forward” в этом слиянии? Git просто переместил указатель ветки вперед, потому что коммит `c4`, на который указывает слитая ветка `hotfix`, был прямым потомком коммита `c2`, на котором вы находились до этого. Другими словами, если коммит сливается с тем, до которого можно добраться двигаясь по истории прямо, Git упрощает слияние просто перенося указатель ветки вперед, так как нет расхождений в изменениях. Это называется “fast-forward”.

Теперь ваши изменения включены в коммит, на который указывает ветка `master`, и исправление можно внедрять.

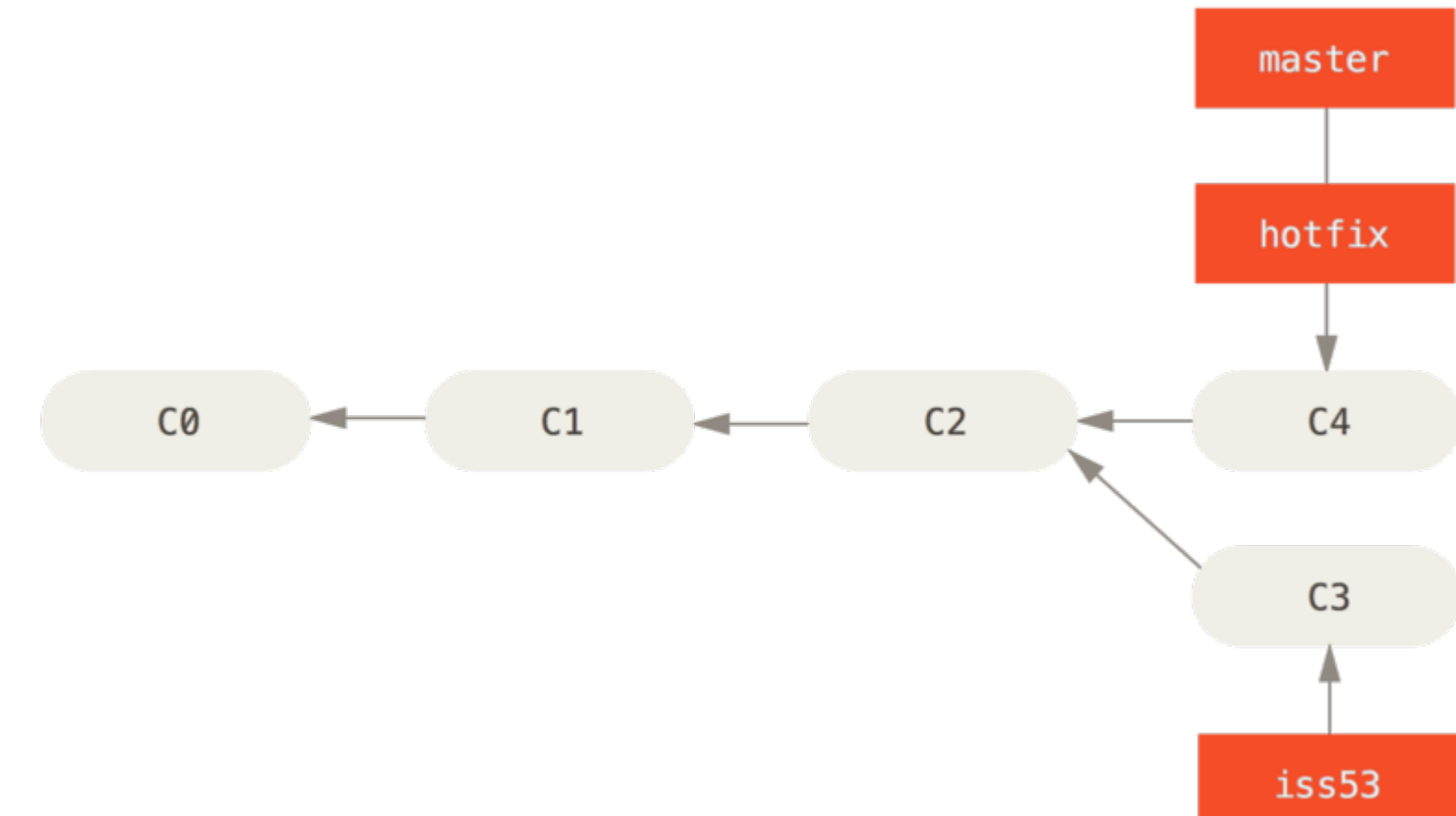


Рисунок 22. `master` перемотан до `hotfix`

После внедрения вашего архиважного исправления вы готовы вернуться к работе над тем, что были вынуждены отложить. Но сначала нужно удалить ветку `hotfix`, потому что она больше не нужна — ветка `master` указывает на то же самое место. Для удаления ветки выполните команду `git branch -d`:

```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).
```

Теперь вы можете переключиться обратно на ветку `iss53` и продолжить работу над проблемой #53:

```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'finished the new footer [issue 53]'
[iss53 ad82d7a] finished the new footer [issue 53]
1 file changed, 1 insertion(+)
```

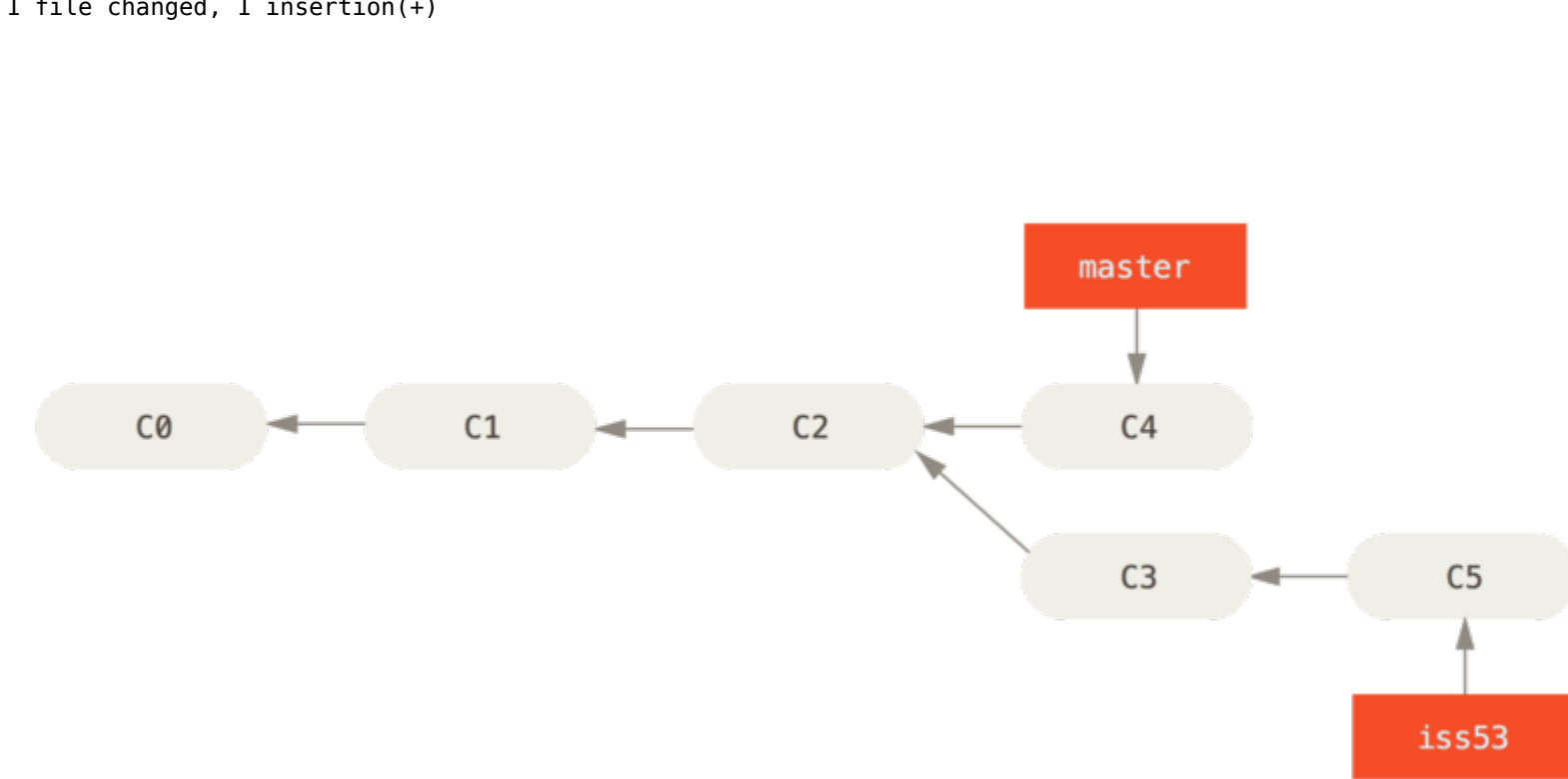


Рисунок 23. Продолжение работы над `iss53`

Стоит обратить внимание на то, что все изменения из ветки `hotfix` не включены в вашу ветку `iss53`. Если их нужно включить, вы можете влить ветку `master` в вашу ветку `iss53` командой `git merge master`, или же вы можете отложить слияние этих изменений до завершения работы, и затем влить ветку `iss53` в `master`.

Основы слияния

Предположим, вы решили, что работа по проблеме #53 закончена и её можно влить в ветку `master`. Для этого нужно выполнить слияние ветки `iss53` точно так же, как вы делали это с веткой `hotfix` ранее. Все что нужно сделать — переключиться на ветку, в которую вы хотите включить изменения, и выполнить команду `git merge`:

```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
 index.html | 1 +
1 file changed, 1 insertion(+)
```

Результат этой операции отличается от результата слияния ветки `hotfix`. В данном случае процесс разработки отделился в более ранней точке. Так как коммит, на котором мы находимся, не является прямым родителем ветки, с которой мы выполняем слияние, Git придётся немного потрудиться. В этом случае Git выполняет простое трёхстороннее слияние используя последние коммиты объединяемых веток и общего для них родительского коммита.

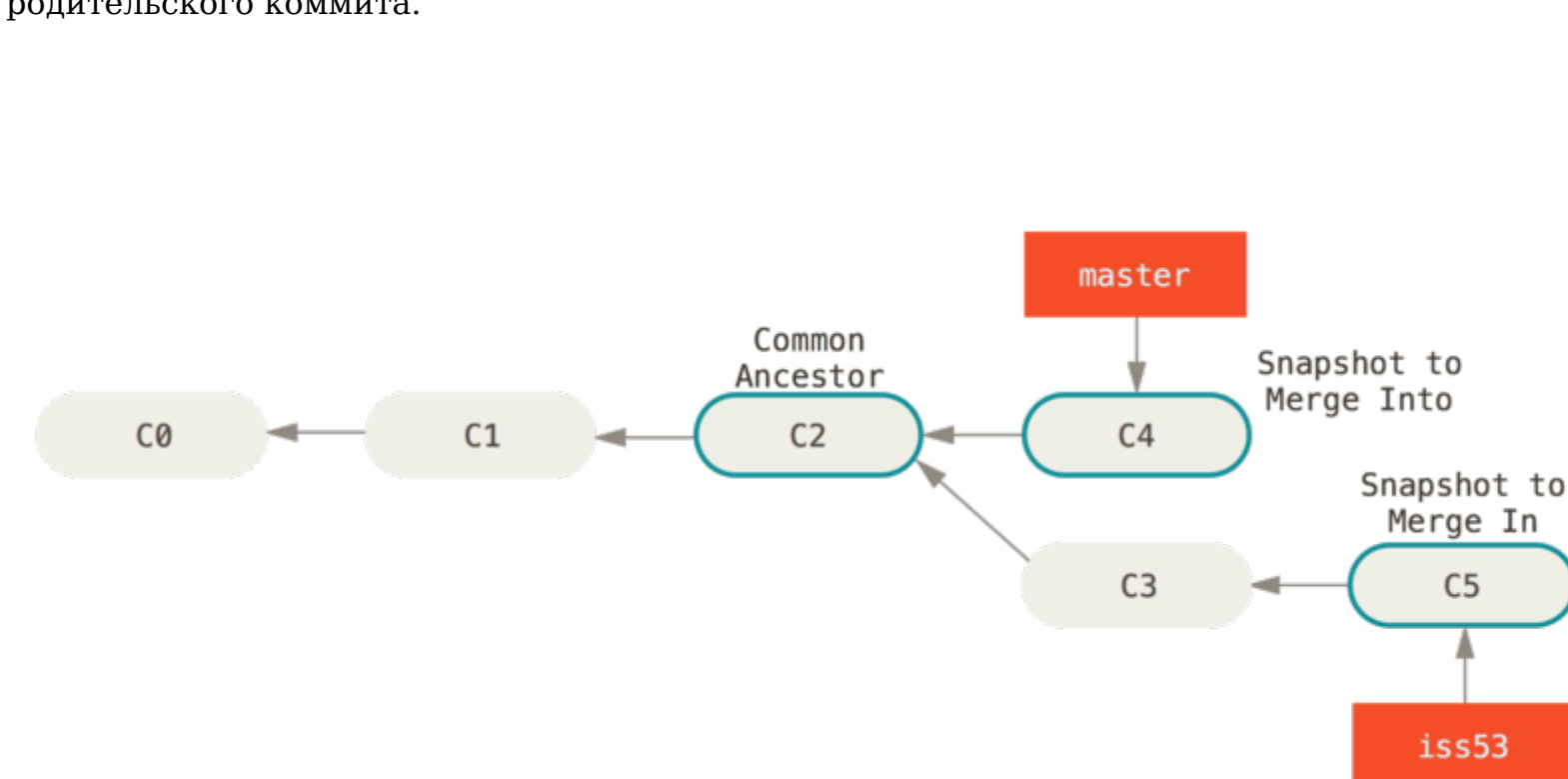


Рисунок 24. Использование трёх снимков при слиянии

Вместо того, чтобы просто передвинуть указатель ветки вперёд, Git создаёт новый результирующий снимок трёхстороннего слияния, а затем автоматически делает коммит. Этот особый коммит называют коммитом слияния, так как у него более одного предка.

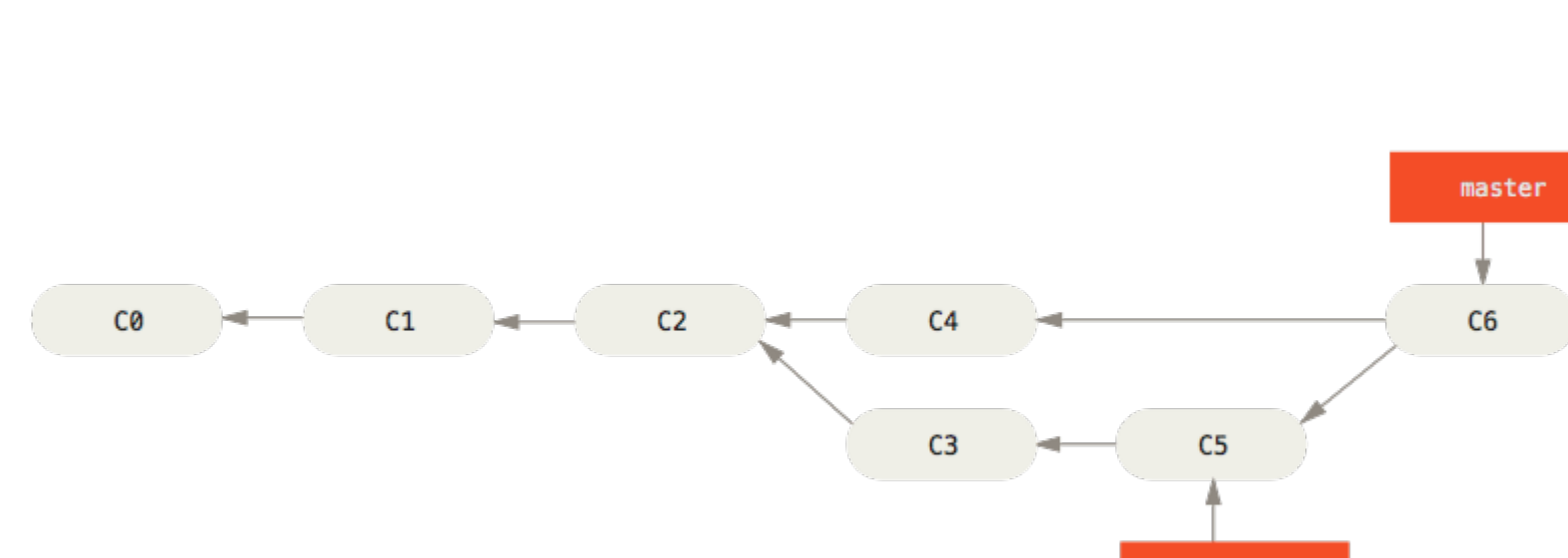


Рисунок 25. Коммит слияния

Теперь, когда изменения слиты, ветка `iss53` больше не нужна. Вы можете закрыть задачу в системе отслеживания ошибок и удалить ветку:

```
$ git branch -d iss53
```

Основные конфликты слияния

Иногда процесс не проходит гладко. Если вы изменили одну и ту же часть одного и того же файла по-разному в двух объединяемых ветках, Git не сможет их чисто объединить. Если ваше исправление ошибки #53 потребовало изменить ту же часть файла что и `hotfix`, вы получите примерно такое сообщение о конфликте слияния:

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Git не создал коммит слияния автоматически. Он остановил процесс до тех пор, пока вы не разрешите конфликт. Чтобы в любой момент после появления конфликта увидеть, какие файлы не объединены, вы можете запустить `git status`:

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
```

```
Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   index.html
```

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

Всё, где есть неразрешённые конфликты слияния, перечисляется как неслитое. В конфликтующие файлы Git добавляет специальные маркеры конфликтов, чтобы вы могли исправить их вручную. В вашем файле появился раздел, выглядящий примерно так:

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```

Это означает, что версия из HEAD (вашей ветки `master`, поскольку именно её вы извлекли перед запуском команды слияния) — это верхняя часть блока (всё, что над =====), а версия из вашей ветки `iss53` представлена в нижней части. Чтобы разрешить конфликт, придётся выбрать один из вариантов, либо объединить содержимое по-своему. Например, вы можете разрешить конфликт, заменив весь блок следующим:


```
<div id="footer">
please contact us at email.support@github.com
</div>
```

В этом разрешении есть немного от каждой части, а строки <<<<<<, ===== и >>>>>> полностью удалены. Разрешив каждый конфликт во всех файлах, запустите `git add` для каждого файла, чтобы отметить конфликт как решённый. Добавление файла в индекс означает для Git, что все конфликты в нём исправлены.

Если вы хотите использовать графический инструмент для разрешения конфликтов, можно запустить `git mergetool`, которое проведет вас по всем конфликтам:

```
$ git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge araxis bc3 codecompare vimdiff emerge
Merging:
index.html

Normal merge conflict for 'index.html':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (opendiff):
```

Если вы хотите использовать инструмент слияния не по умолчанию (в данном случае Git выбрал `opendiff`, поскольку команда запускалась на Mac), список всех поддерживаемых инструментов представлен вверху после фразы “one of the following tools.” Просто введите название инструмента, который хотите использовать.

Note Мы рассмотрим более продвинутые инструменты для разрешения сложных конфликтов слияния в разделе [Продвинутое слияние](#).

После выхода из инструмента слияния Git спросит об успешности процесса. Если вы ответите скрипту утвердительно, то он добавит файл в индекс, чтобы отметить его как разрешенный. Теперь можно снова запустить `git status`, чтобы убедиться в отсутствии конфликтов:

```
$ git status
On branch master
All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:
  modified:   index.html
```

Если это вас устраивает и вы убедились, что все файлы, где были конфликты, добавлены в индекс — выполните команду `git commit` для создания коммита слияния. Комментарий к коммиту слияния по умолчанию выглядит примерно так:

```
Merge branch 'iss53'

Conflicts:
  index.html
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#   .git/MERGE_HEAD
# and try again.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#   modified:   index.html
#
```

Если вы считаете, что коммит слияния требует дополнительных пояснений — опишите как были разрешены конфликты и почему были применены именно такие изменения, если это не очевидно.

[prev](#) | [next](#)
[About this site](#)
Patches, suggestions, and comments are welcome.
Git is a member of [Software Freedom Conservancy](#)