Система контроля версий Git

Ветвление

О ветвлении

- Почти каждая система контроля версий (СКВ) в какой-то форме поддерживает ветвление. Используя ветвление, Вы отклоняетесь от основной линии разработки и продолжаете работу независимо от неё, не вмешиваясь в основную линию. Во многих СКВ создание веток — это очень затратный процесс, часто требующий создания новой копии директории, что может занять много времени для большого проекта.
- Ветвление Git очень легковесно: операция создания ветки выполняется почти мгновенно, переключение между ветками туда-сюда, обычно, также быстро. В отличие от многих других СКВ, Git поощряет процесс работы, при котором ветвление и слияние выполняется часто, даже по несколько раз в день. Понимание и владение этой функциональностью дает вам уникальный и мощный инструмент, который может полностью изменить привычный процесс разработки.

О ветвлении в двух словах

- Для точного понимания механизма ветвлений, необходимо вернуться назад и изучить то, как Git хранит данные. Как вы можете помнить, Git не хранит данные в виде последовательности изменений, он использует набор снимков (**snapshot**).
- Когда вы делаете коммит, Git сохраняет его в виде объекта, который содержит указатель на снимок (**snapshot**) подготовленных данных. Этот объект так же содержит имя автора и email, сообщение и указатель на коммит или коммиты непосредственно предшествующие данному (его родителей): отсутствие родителя для первоначального коммита, один родитель для обычного коммита, и несколько родителей для результатов слияния двух и более веток.

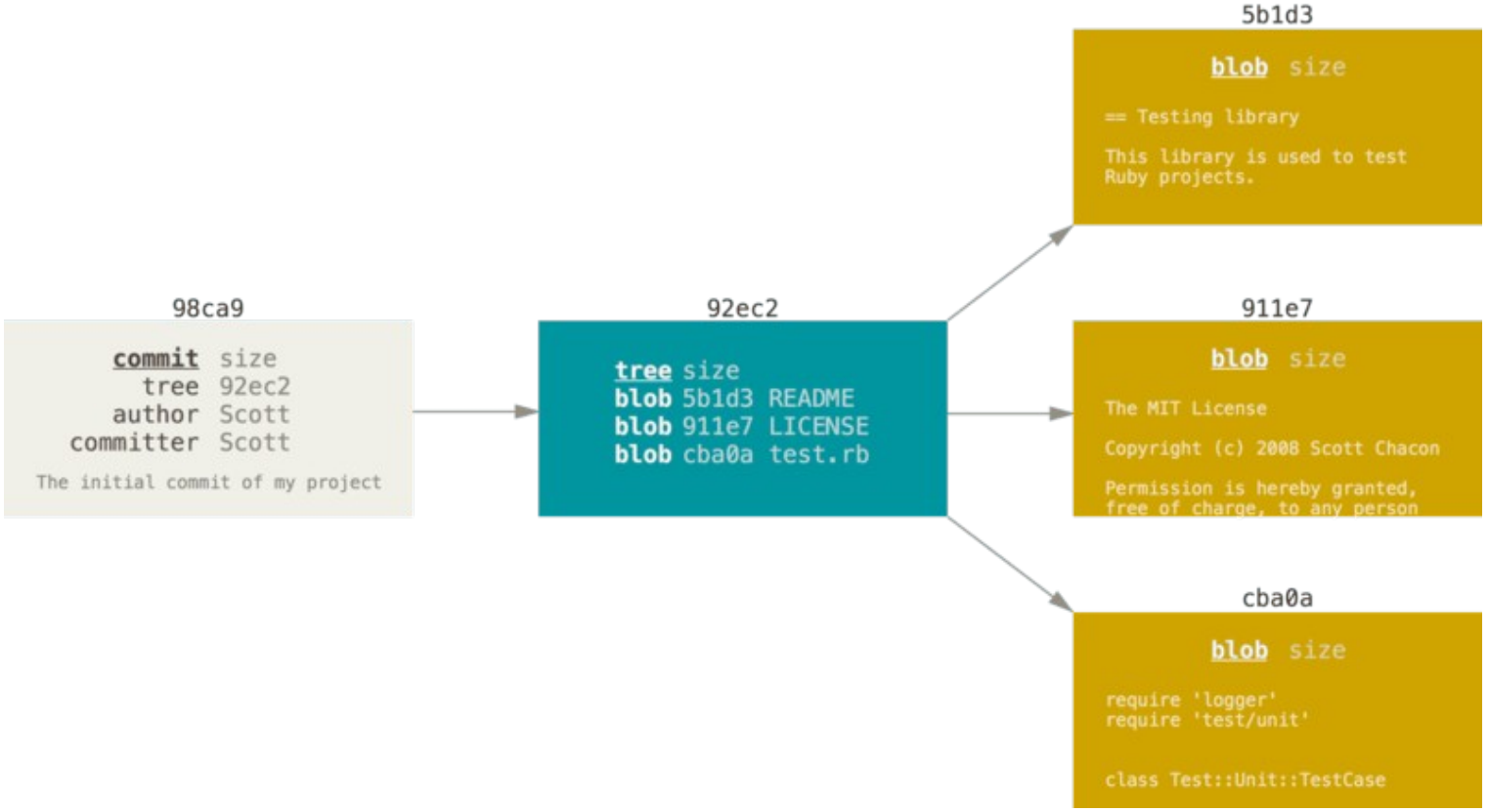
БЛОБ

Предположим, в вашей рабочей директории есть три файла и вы добавляете их все в индекс и создаёте коммит. Во время индексации вычисляется контрольная сумма каждого файла (SHA-1), затем каждый файл сохраняется в репозиторий (**блоб** — большой бинарный объект), а контрольная сумма попадёт в индекс:

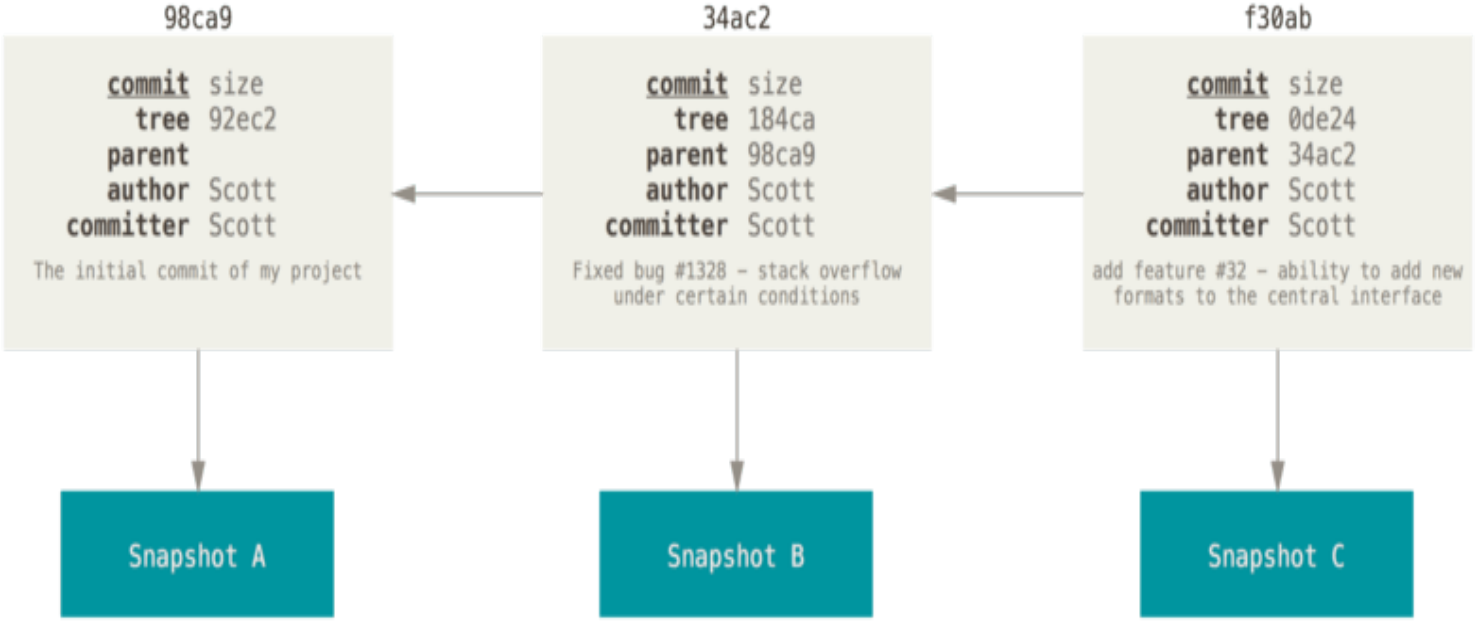
- $ git add README test.rb LICENSE
- $ git commit -m 'initial commit of my project'

Когда вы создаёте коммит командой git commit, Git вычисляет контрольные суммы каждого подкаталога (в нашем случае, только основной каталог проекта) и сохраняет его в репозитории как объект дерева каталогов. Затем Git создаёт объект коммита с метаданными и указателем на основное дерево проекта для возможности воссоздать этот снимок в случае необходимости.

Коммит и его дерево

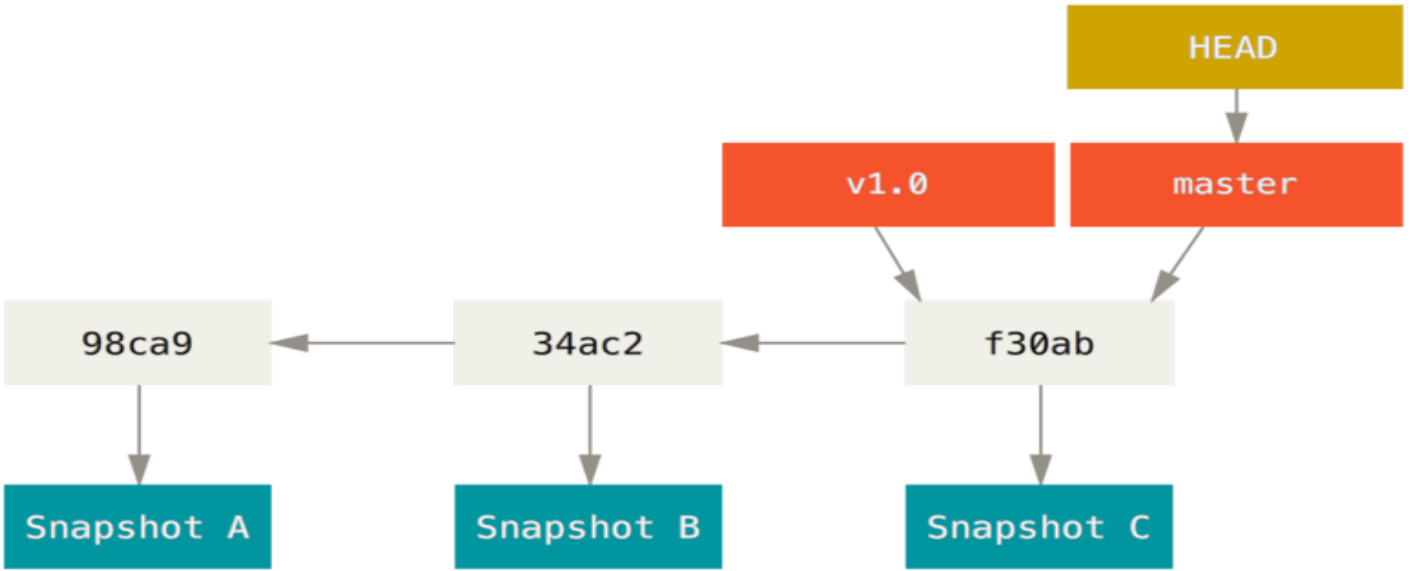


Коммит и его родители



Если вы сделаете изменения и создадите ещё один коммит, то он будет содержать указатель на предыдущий коммит.

Ветка и история коммитов

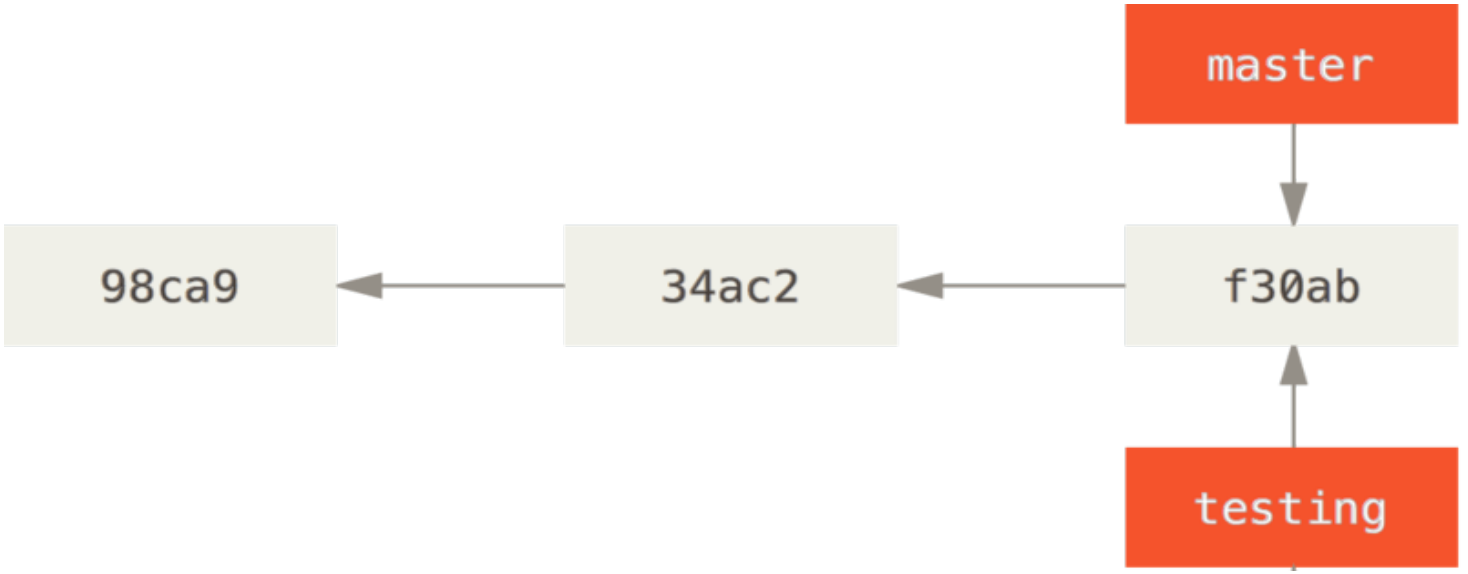


Ветка в Git — это простой перемещаемый указатель на один из таких коммитов. По умолчанию, имя основной ветки в Git — master. Как только вы начнёте создавать коммиты, ветка master будет всегда указывать на последний коммит. Каждый раз при создании коммита указатель ветки master будет передвигаться на следующий коммит автоматически.

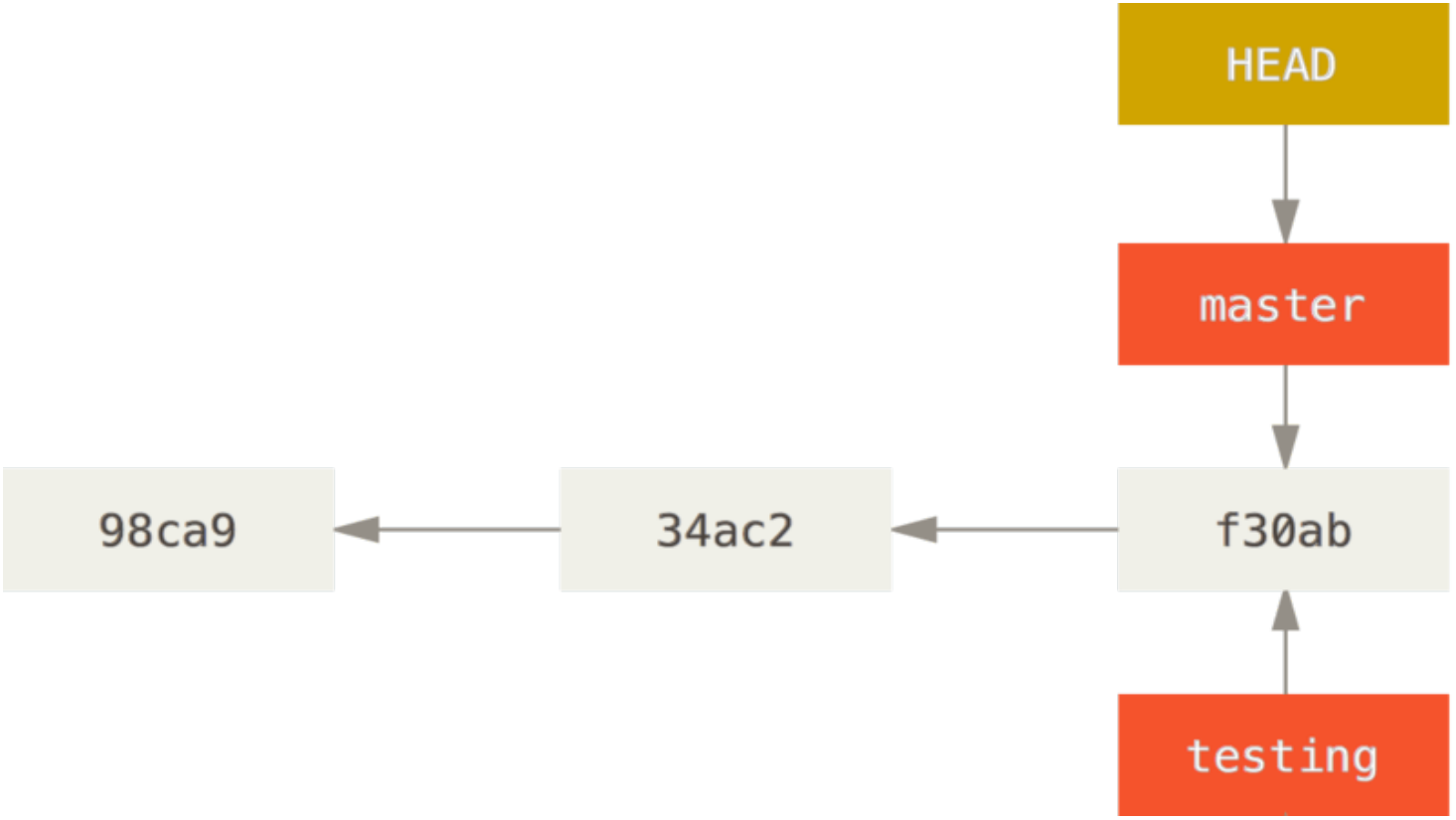
Создание новой ветки

- Что же на самом деле происходит при создании ветки? Всего лишь создаётся новый указатель для дальнейшего перемещения. Допустим вы хотите создать новую ветку с именем testing. Вы можете это сделать командой **git branch**:
  - $ git branch testing
- В результате создаётся новый указатель на текущий коммит. Как Git определяет, в какой ветке вы находитесь? Он хранит специальный указатель HEAD. В Git — это указатель на текущую локальную ветку. В нашем случае мы все еще находимся в ветке master. Команда **git branch** только создаёт новую ветку, но не переключает на неё.

Две ветки указывают на одну последовательность коммитов



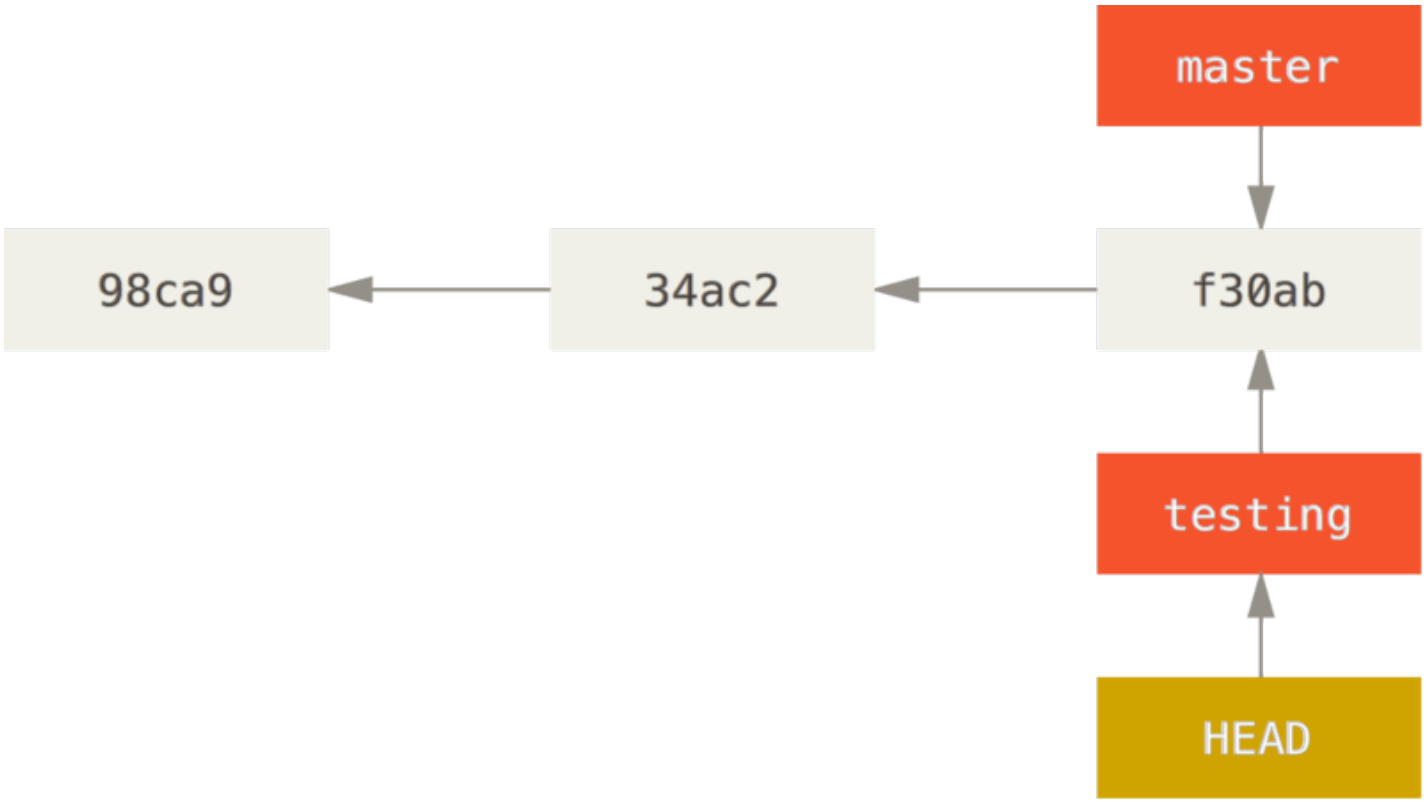
HEAD указывает на ветку



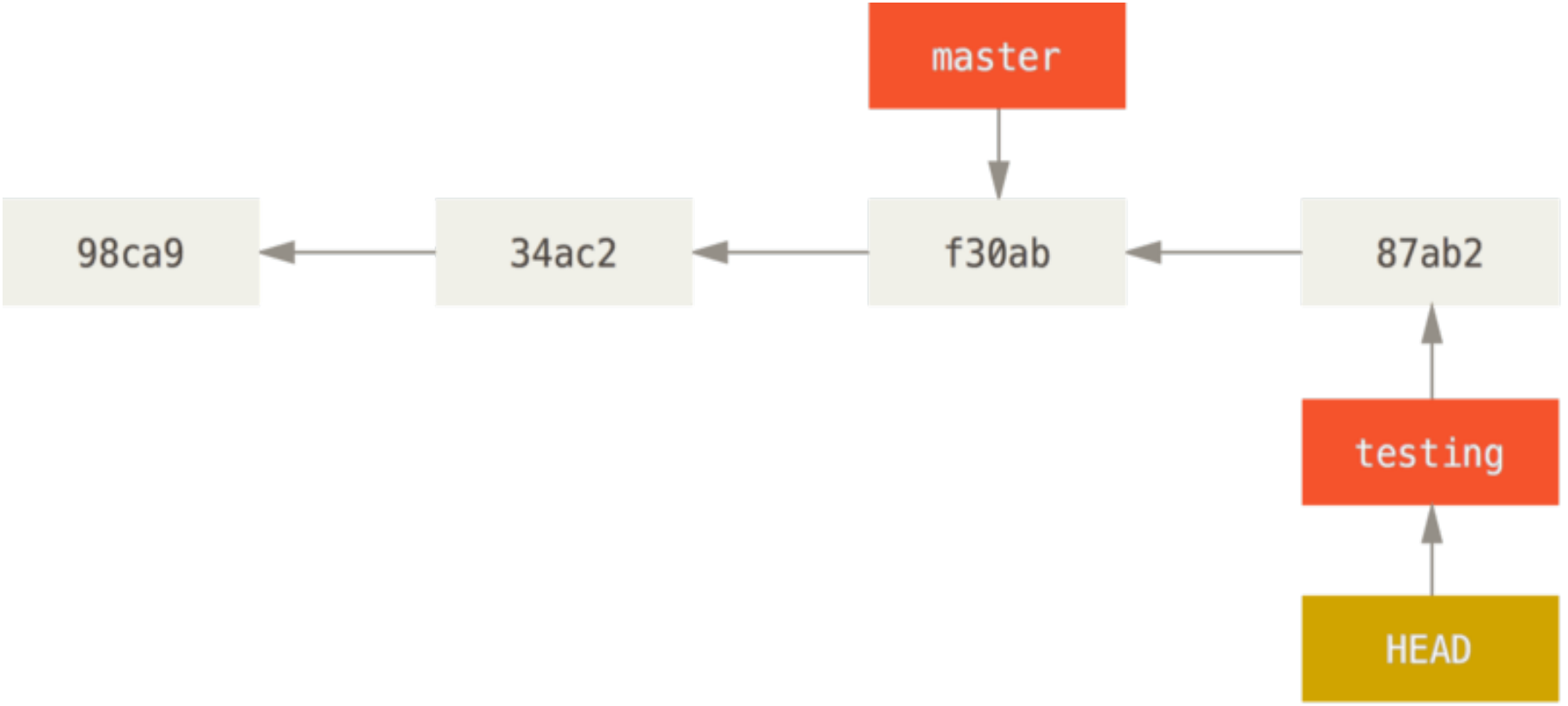
Переключение веток

- Для переключения на существующую ветку выполните команду **git checkout**. Давайте переключимся на ветку testing:
  - $ git checkout testing
- Какой в этом смысл? Давайте сделаем ещё один коммит:
  - $ nano test.rb
  - $ git commit -a -m 'made a change'
- Указатель на ветку testing переместился вперёд, а master указывает на тот же коммит, где вы были до переключения веток командой **git checkout**. Давайте переключимся назад на ветку master:
  - $ git checkout master

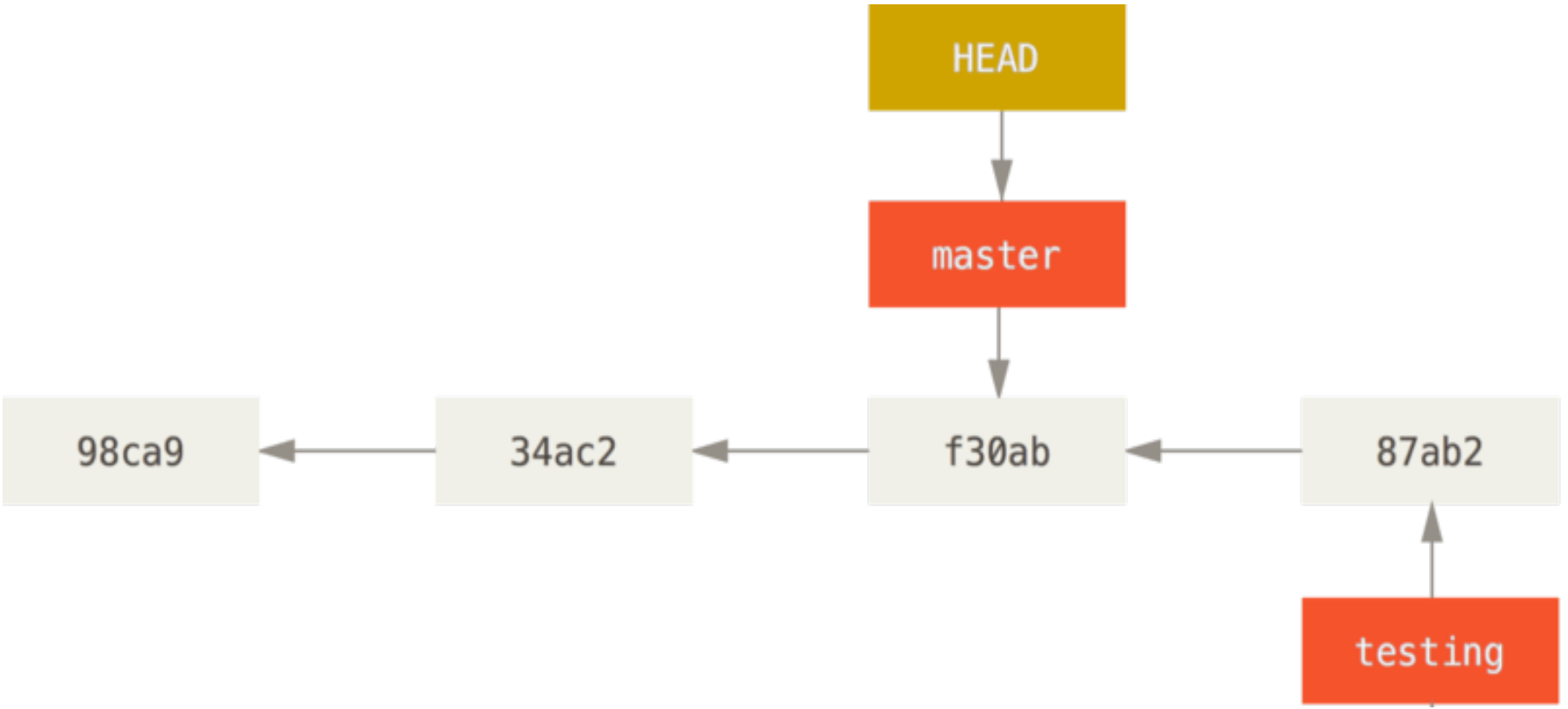
HEAD указывает на текущую ветку



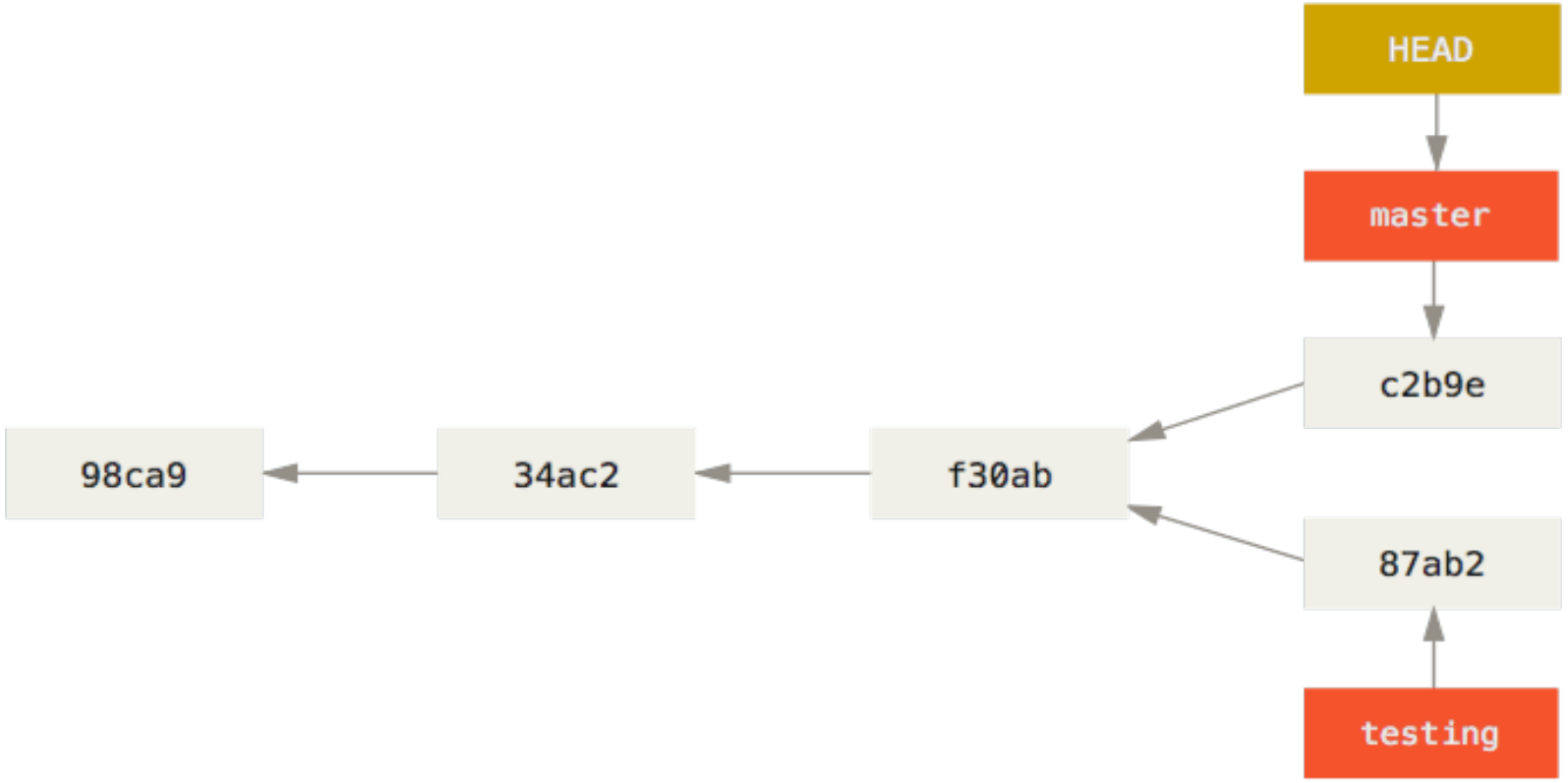
Указатель HEAD переместился вперед после коммита



HEAD переместился на master



Разветвленная история



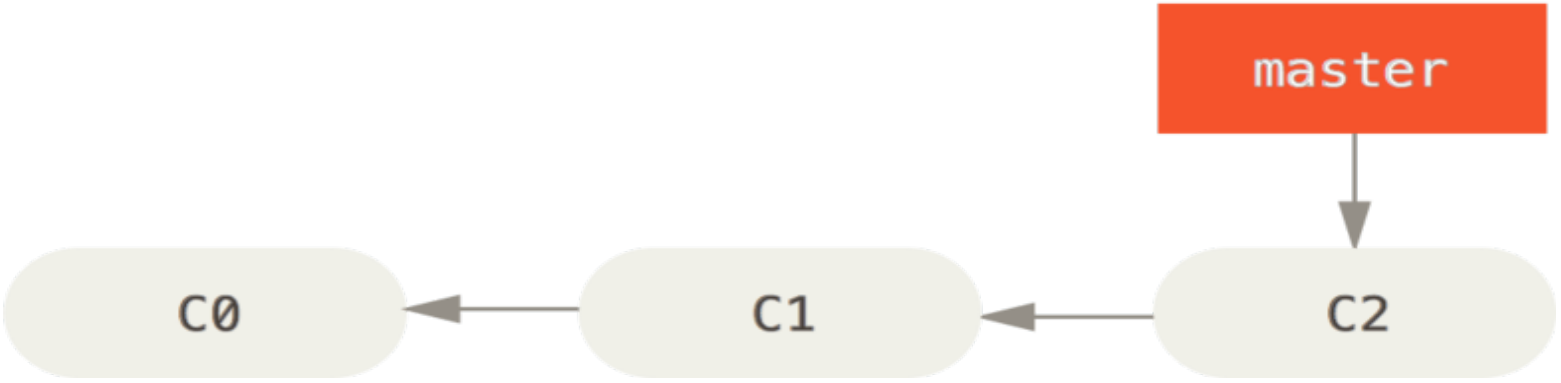
Операции создания и переключения

- Создать новую ветку:
  - $ git branch new_branch
- Переключится на другую ветку
  - $ git checkout other_branch
- Создать ветку и сразу переключится:
  - $ git checkout -b new_branch2
- Посмотреть ветки с последними коммитами:
  - $ git branch -v
- Посмотреть можно и через git log, например:
  - $ git log --oneline --decorate --graph --all.

Основы ветвления и слияния

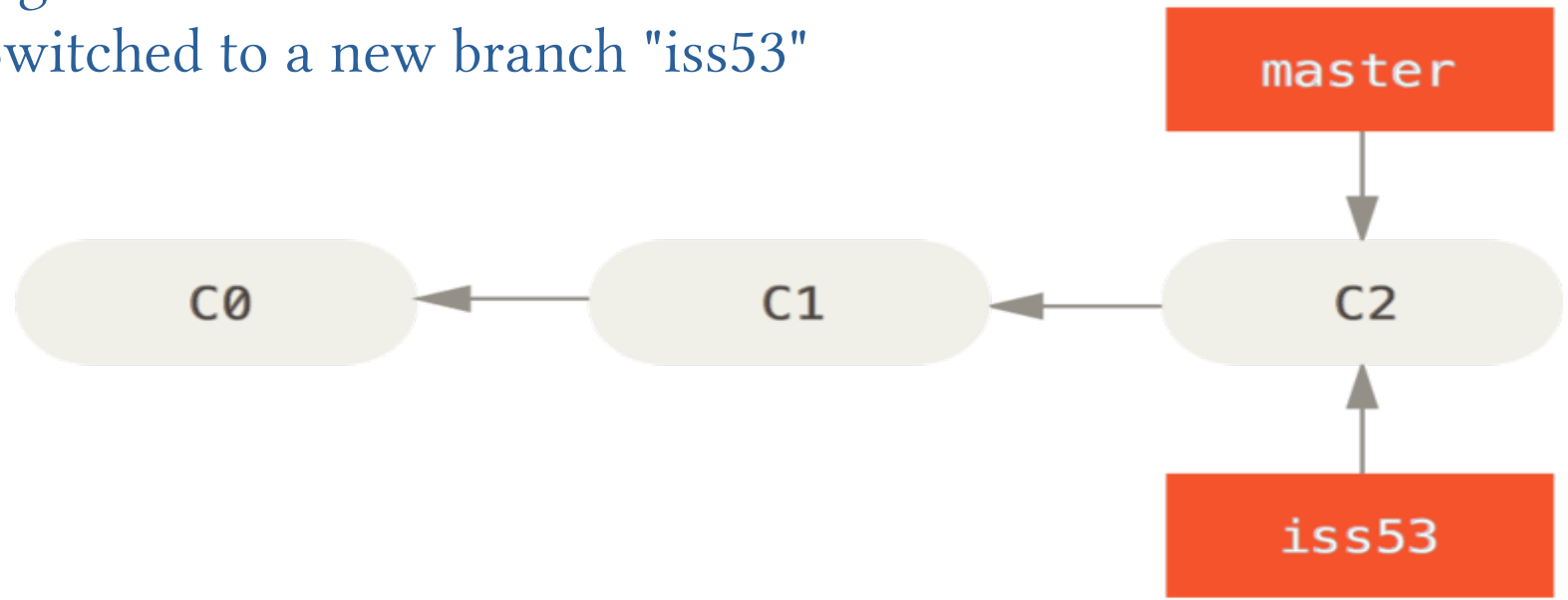
- Давайте рассмотрим простой пример рабочего процесса, который может быть полезен в вашем проекте. Ваша работа построена так:
    1. Вы работаете над сайтом.
    2. Вы создаете ветку для новой статьи, которую вы пишете.
    3. Вы работаете в этой ветке.
- В этот момент вы получаете сообщение, что обнаружена 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.

Простая история коммитов



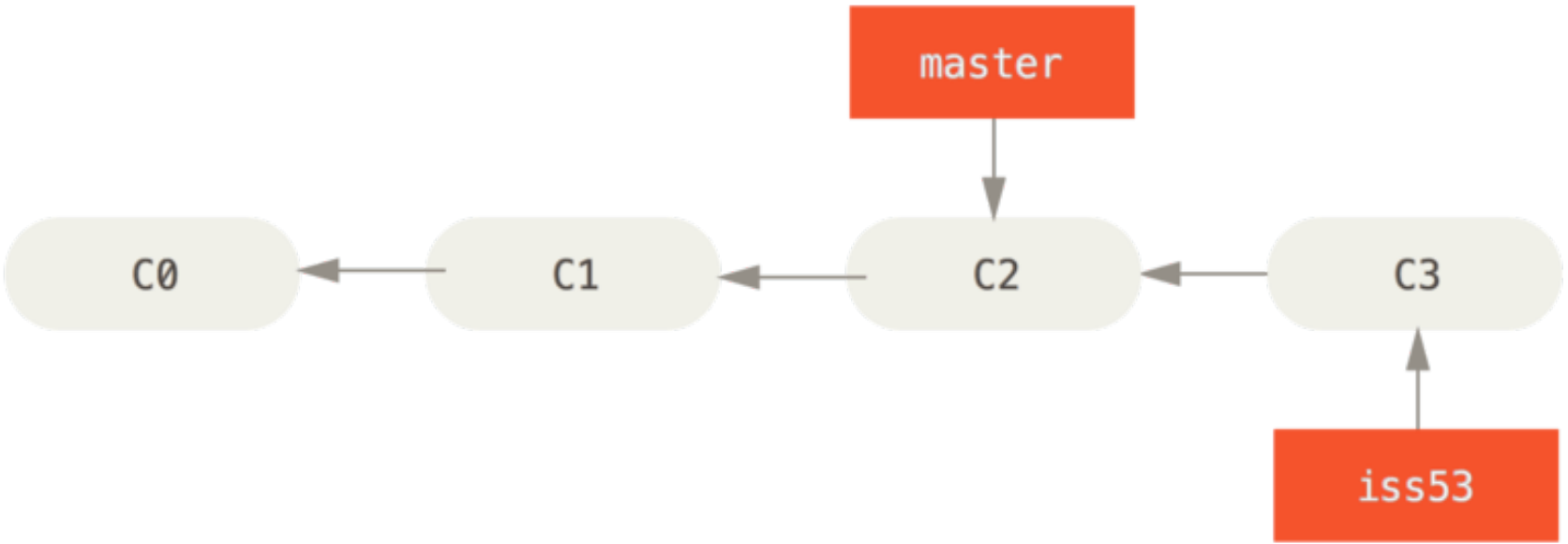
Создание нового указателя ветки

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



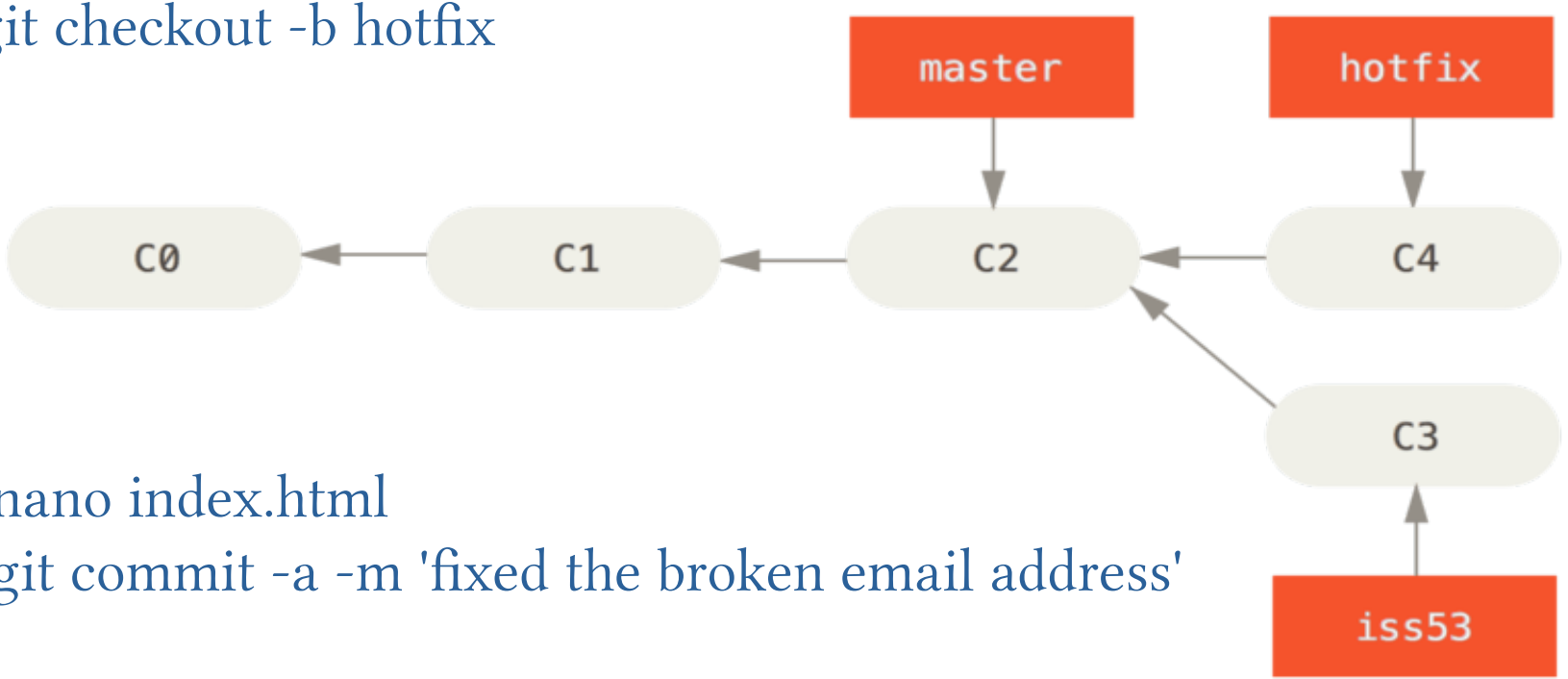
Ветка iss53 двигается вперед

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



Ветка hotfix основана на master

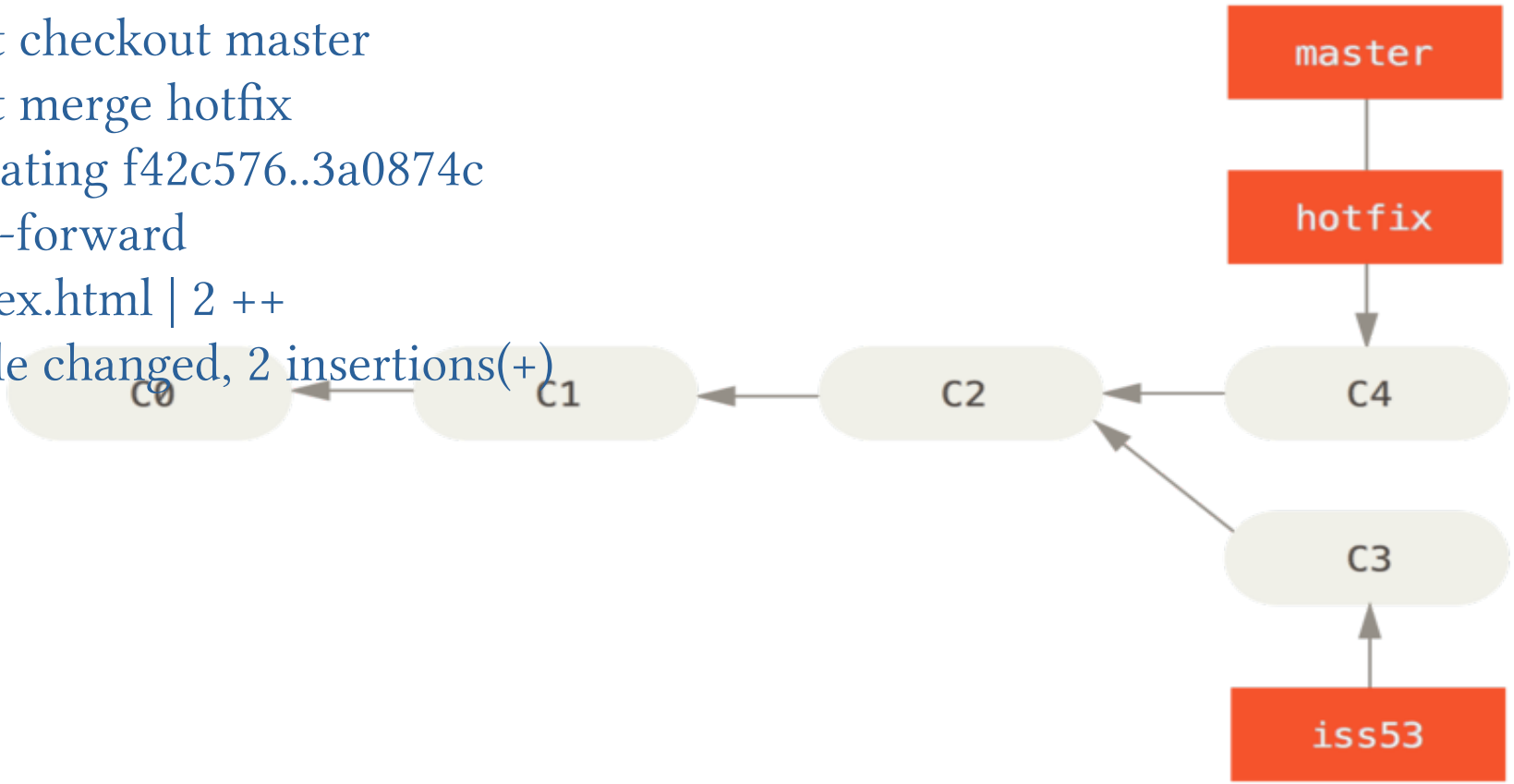
$ git checkout master
$ git checkout -b hotfix



$ nano index.html
$ git commit -a -m 'fixed the broken email address'

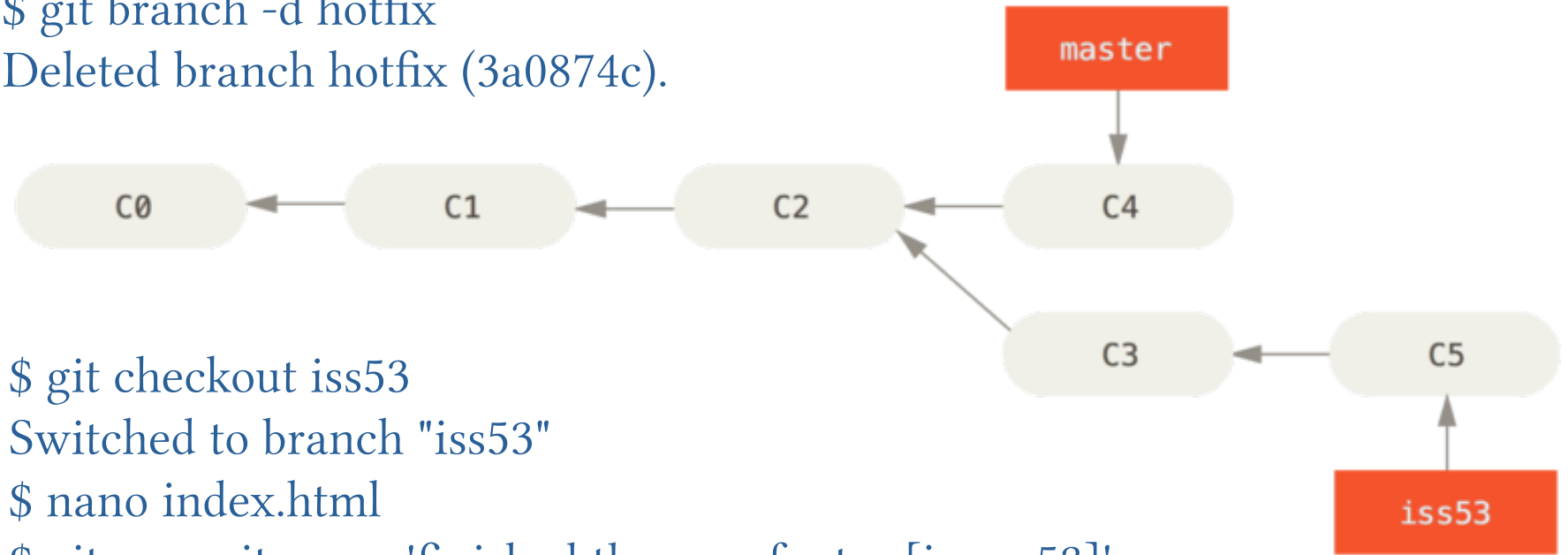
Master перемотан до hotfix

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



Продолжение работы над iss53

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



$ git checkout iss53
Switched to branch "iss53"
$ nano 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(+)

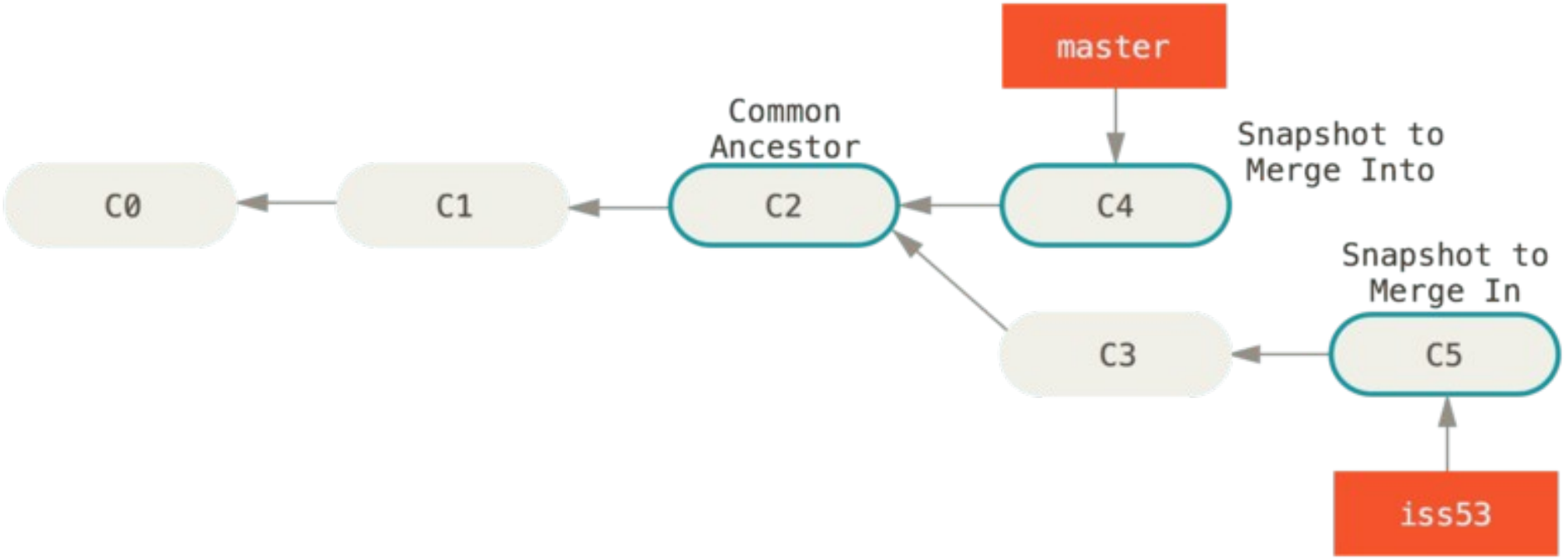
Основы слияния

- 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:

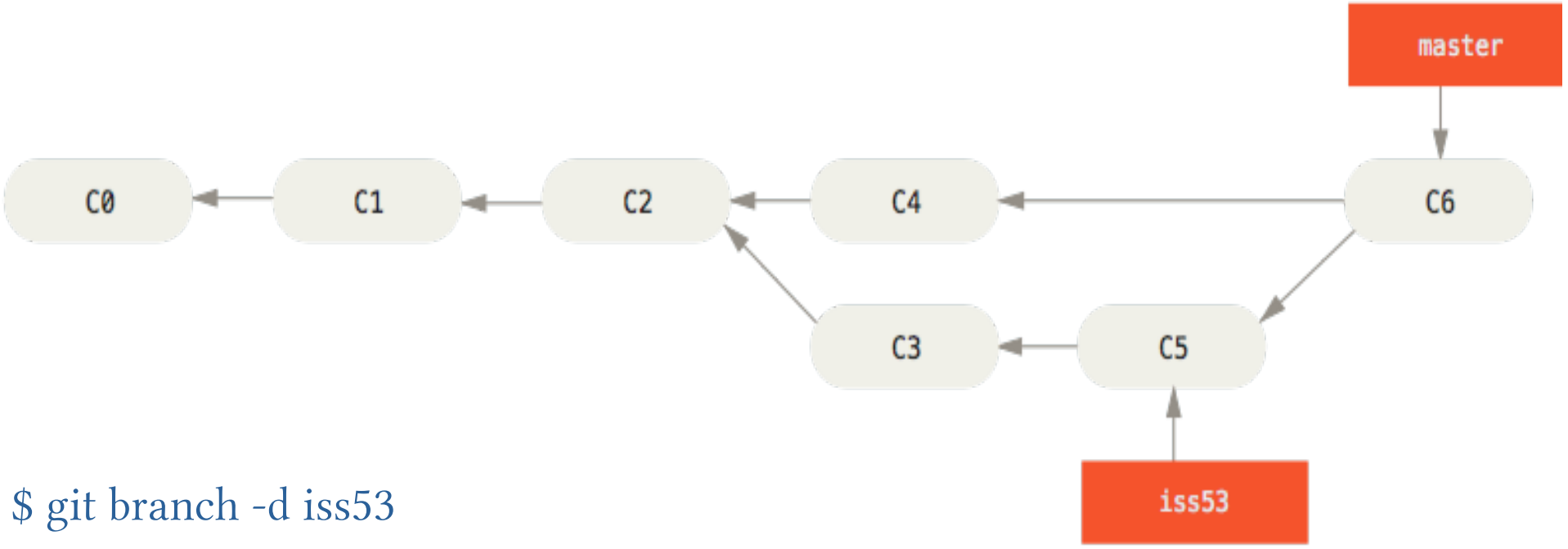
```
$ git checkout master
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```

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

Использование трех снимков при слиянии



Коммит слияния



$ 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
```

Операции слияния и удаления

- Слиение ветки new_branch в ветку мастер:
  - $ git checkout master
  - $ git merge new_branch
- Удаление ветки:
  - $ git branch -d new_branch #которая была смержена
  - $ git branch -D new_branch #удалить принудительно
- Работа с удаленным сервером:
  - $ git push origin new_branch #запушить на сервер
  - $ git push origin --delete new_branch #удалить ветку с сервера

Работа с метками

- Есть два вида меток (тегов): легковесные и аннотированные:
  - $ git tag v1.0.0 #создание легковестной метки
  - $ git tag -a v1.2.0 -m 'my version 1.2.0' #создание аннотированной метки
  - $ git push origin v1.0.0 #запушит на удаленный сервер метку
  - $ git push origin --tags. #запушит на удаленный сервер все метки
- Удалить метку:
  - $ git tag -d v1.0.0 #локально
  - $ git push origin --delete v.1.0.0 #с удаленного сервера
- Просмотр меток:
  - $ git tag #все метки
  - $ git show v1.2.0 #просмотр информации о метке