Дорожная карта

0. Приступаем
1. Введение в Git
2. Начало работы с Git
3. Просмотр истории
4. Ветвление 👈
5. Слияние
6. Отмена изменений
7. Рабочий процесс

Дорожная карта (продолжение)

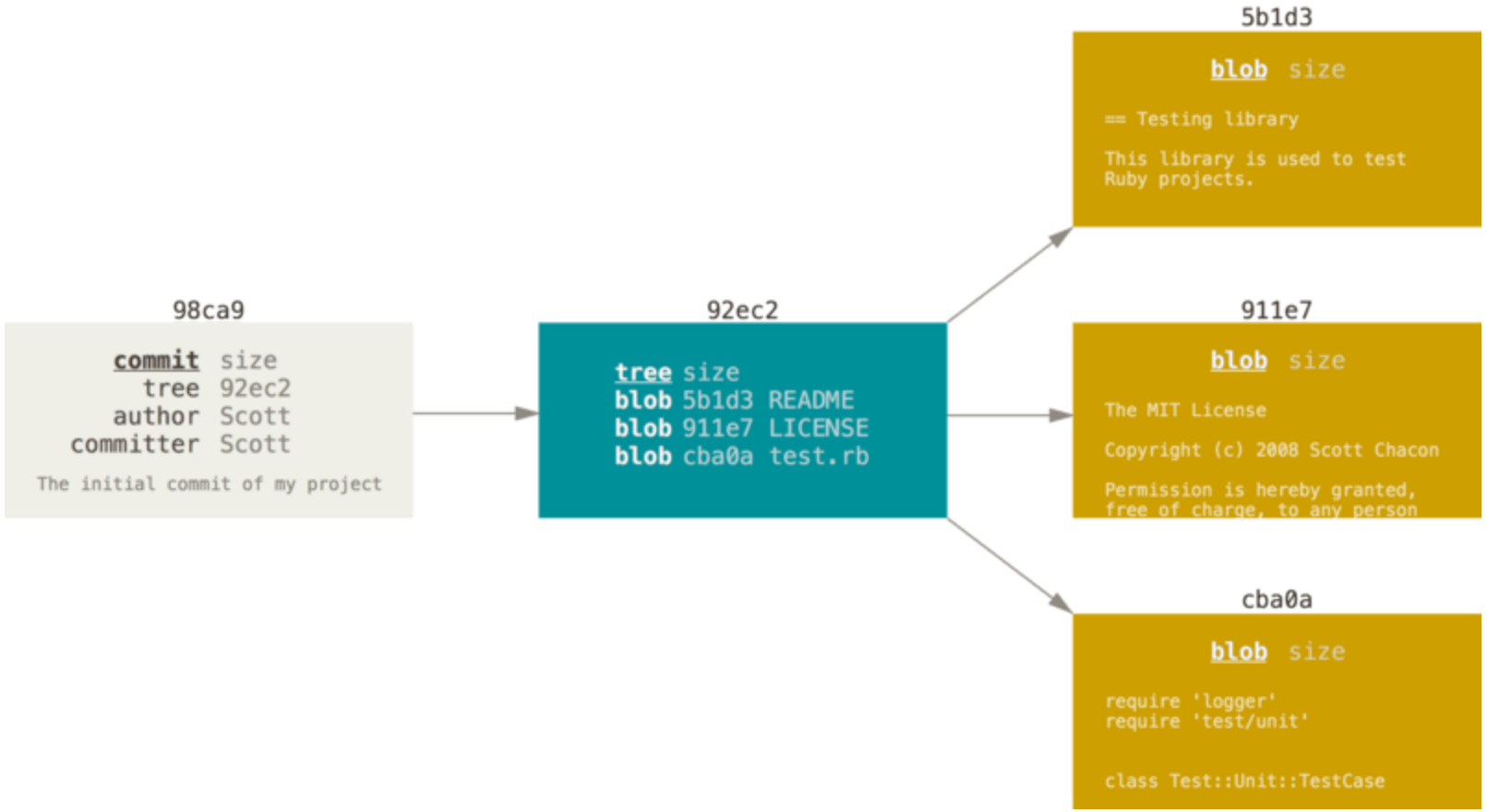
8. Работа в команде
9. Метки
10. Последние штрихи
11. Завершаем

Основы ветвления

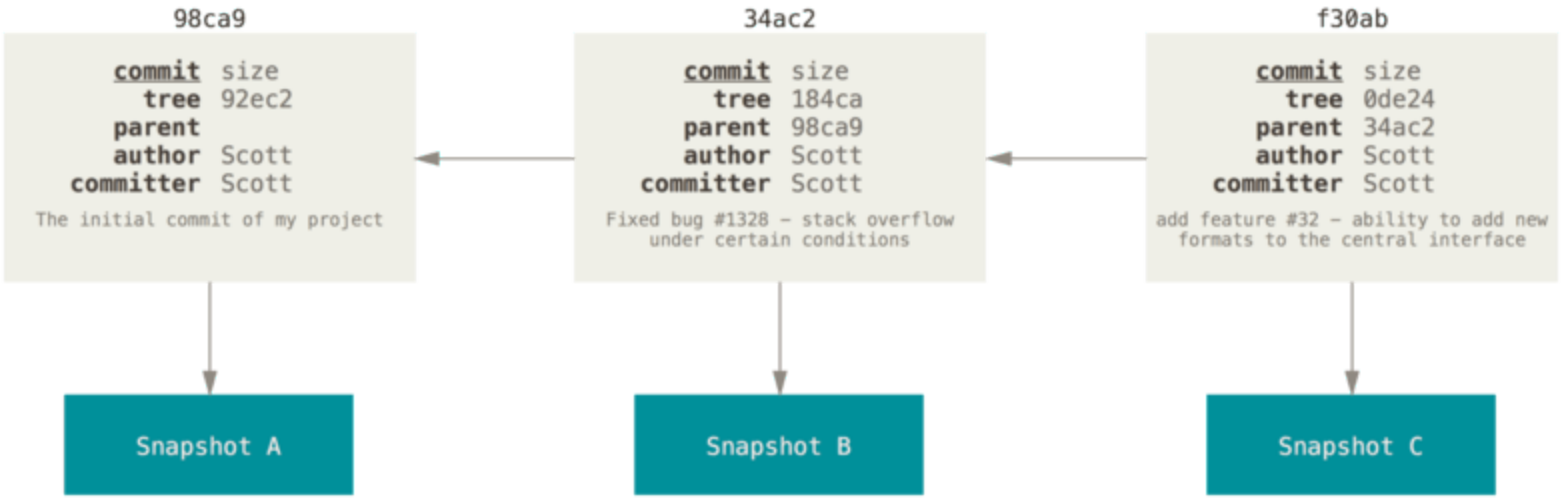
- Почти каждая система контроля версий (СКВ) в какой-то форме поддерживает ветвление.
- Во многих СКВ создание веток - это очень затратный процесс.
- В Git ветвление очень легковесно.
- Git поощряет процесс работы, при котором ветвление и слияние выполняется часто.

Как Git хранит данные

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

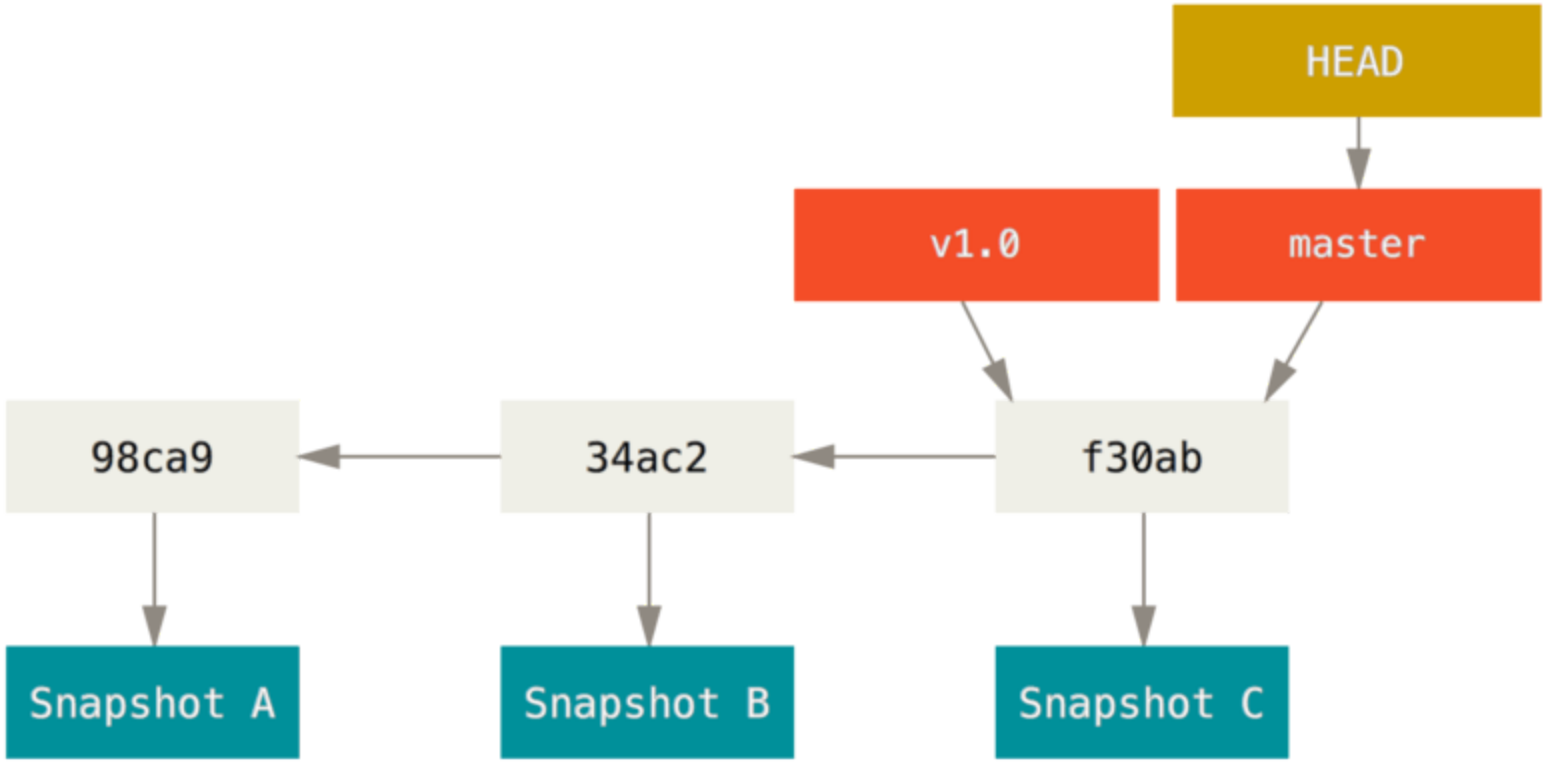


История коммитов



Ветка в Git

Ветка (branch) в Git - это легко перемещаемый указатель на один из этих коммитов. Имя основной ветки по умолчанию в Git - `master`.



Пример рабочего процесса

Ваша работа построена так:

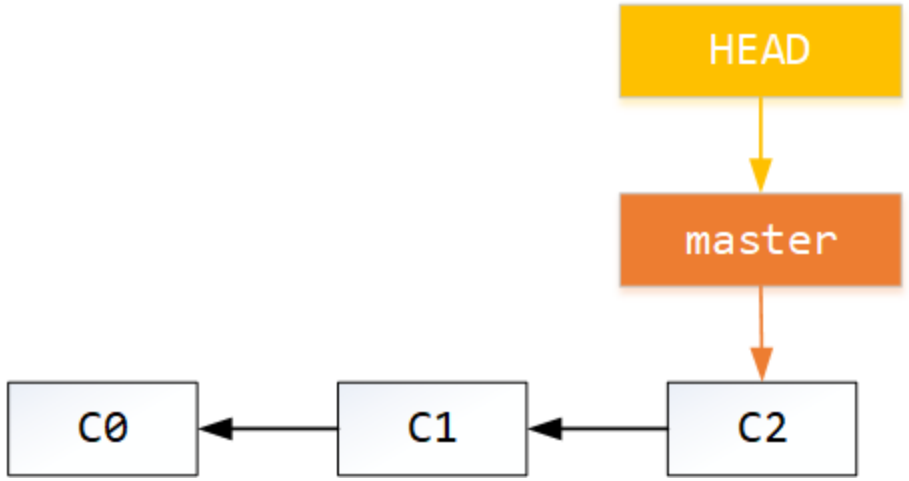
1. Вы работаете над проектом.
2. Вы создаете ветку для тестирования.
3. Вы работаете в этой ветке.

Приходит сообщение о критической ошибке, требующей немедленного 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 на ветку тестирования, и закончить тесты.

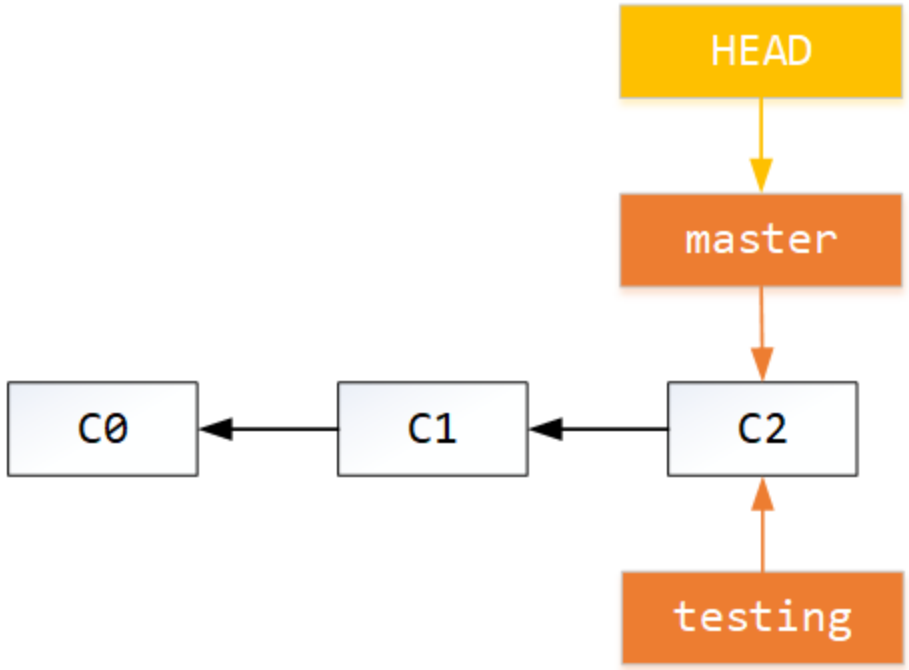
Пример → Текущая ветка master

```
$ git log --oneline --decorate
f30ab (HEAD, master) add feature #32 - ability to add new
34ac2 fixed bug #1328 - stack overflow under certain conditions
98ca9 initial commit of my project
```



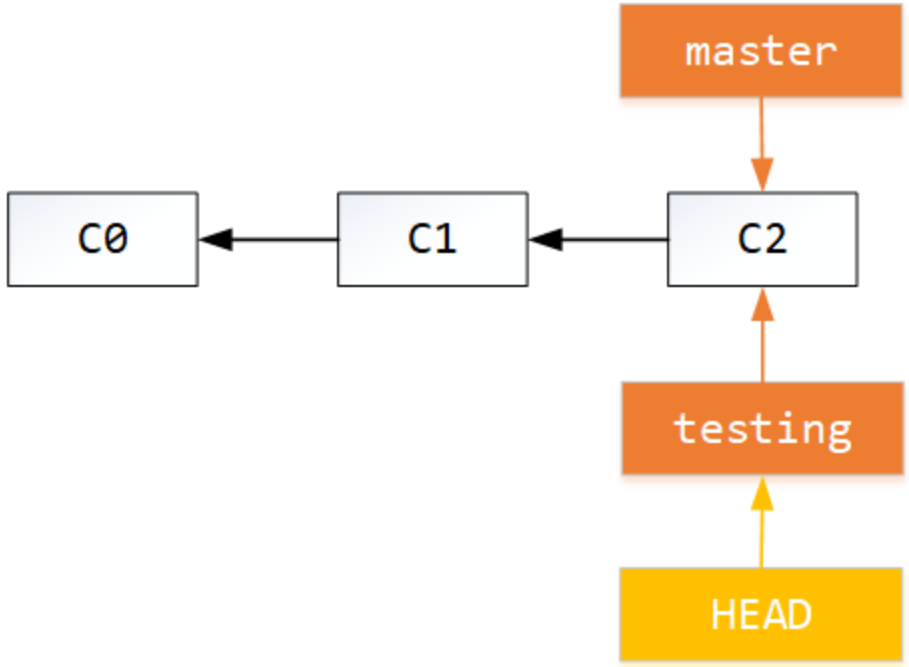
Пример → Создание нового указателя ветки testing

```
$ git branch testing
```



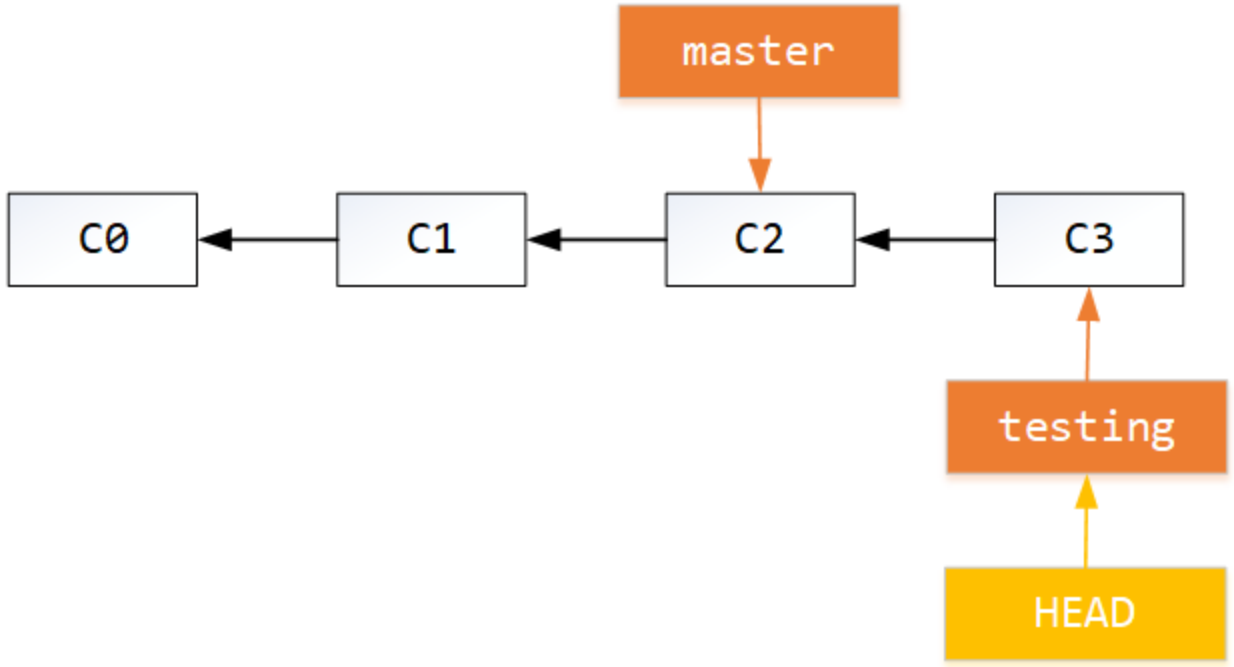
Пример → Переключение ветки на testing

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



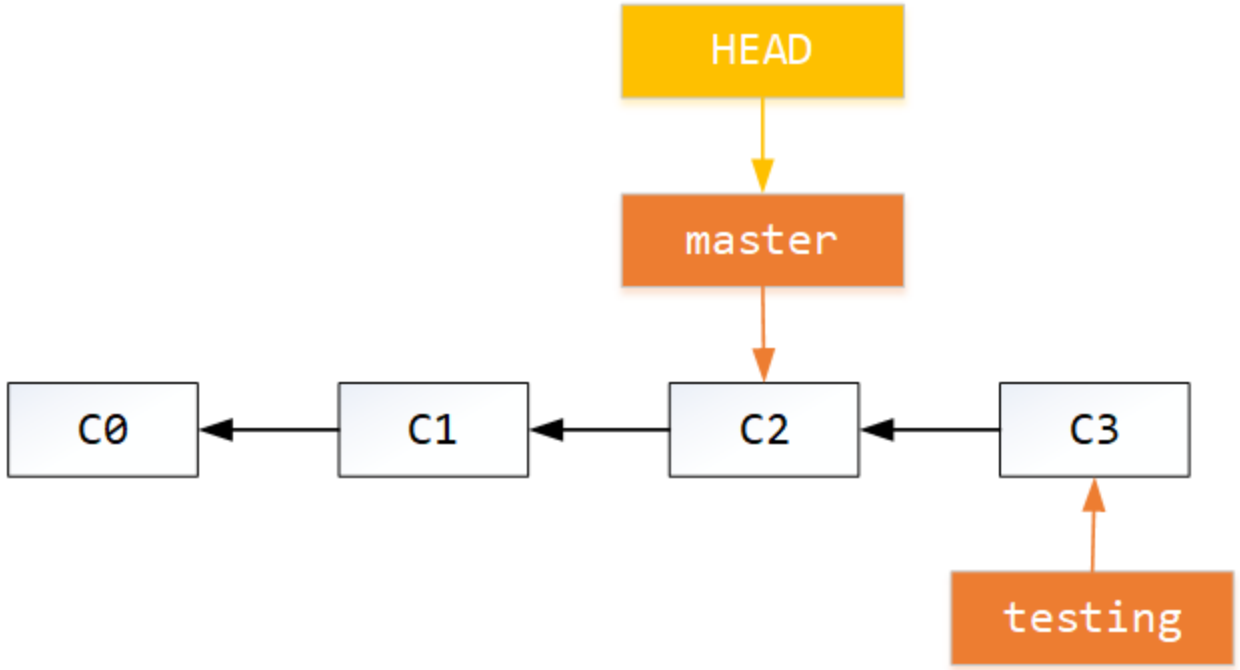
Пример → Указатель на ветку HEAD переместился вперёд после коммита

```
$ vim test.rb
$ git commit -a -m 'add unittests'
```



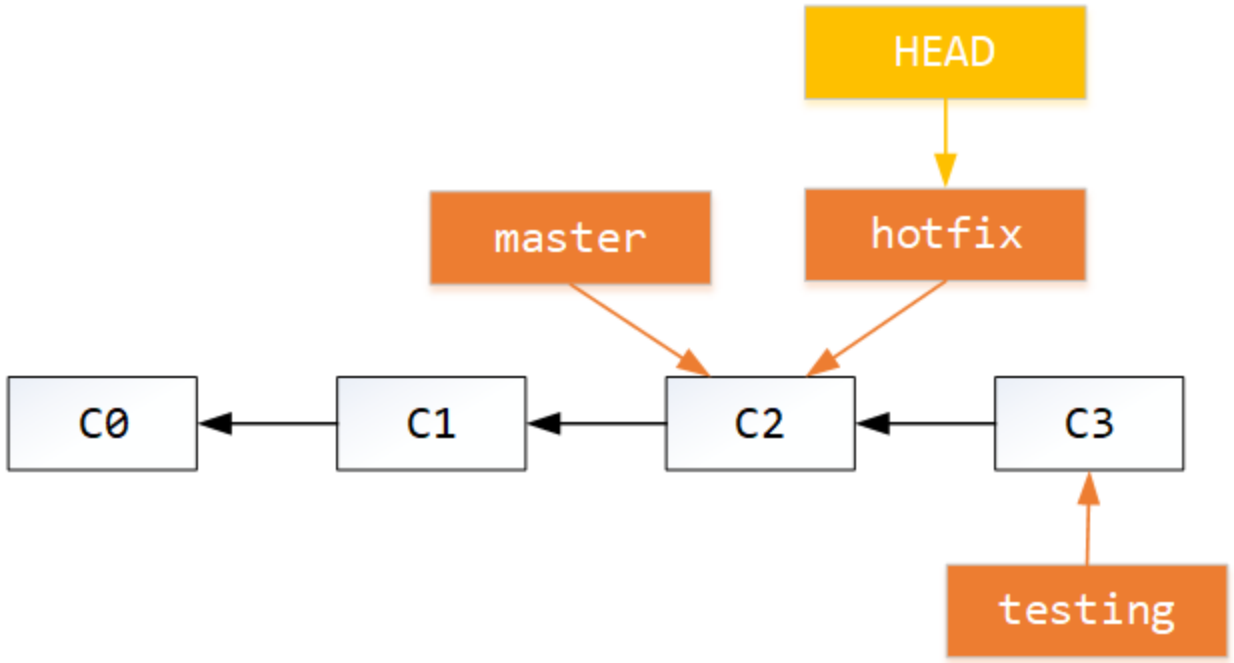
Пример → HEAD перемещается когда вы делаете checkout

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



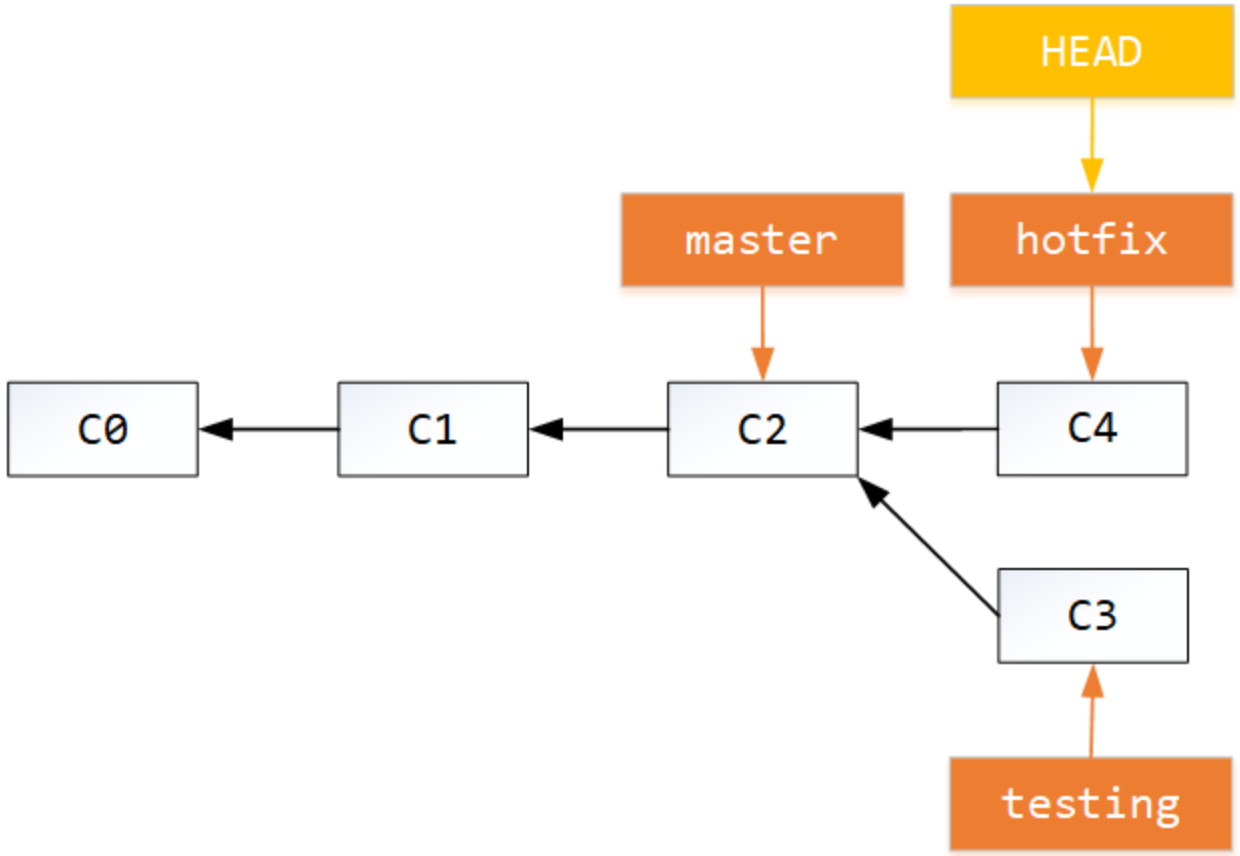
Пример → Ветка hotfix основана на ветке master

```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
```



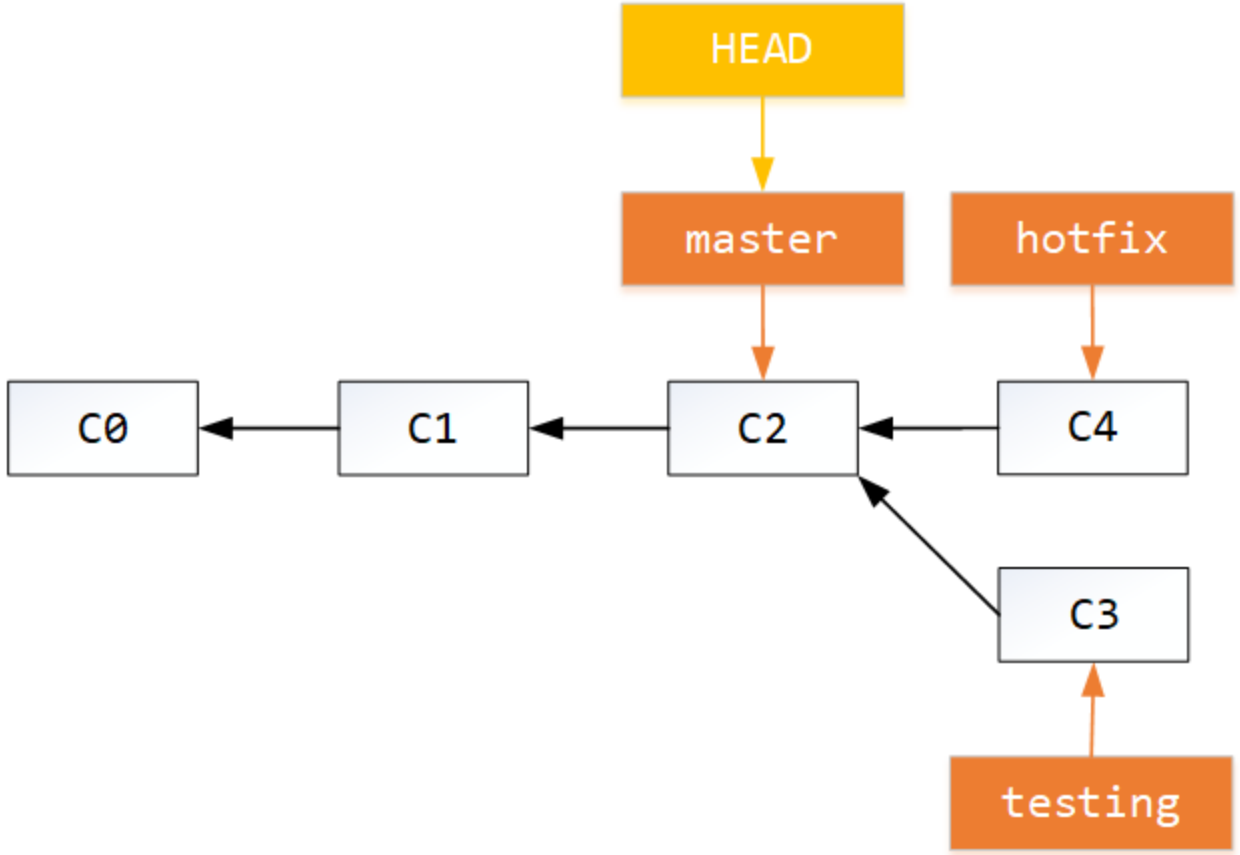
Пример → Ветка hotfix содержит новые изменения

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



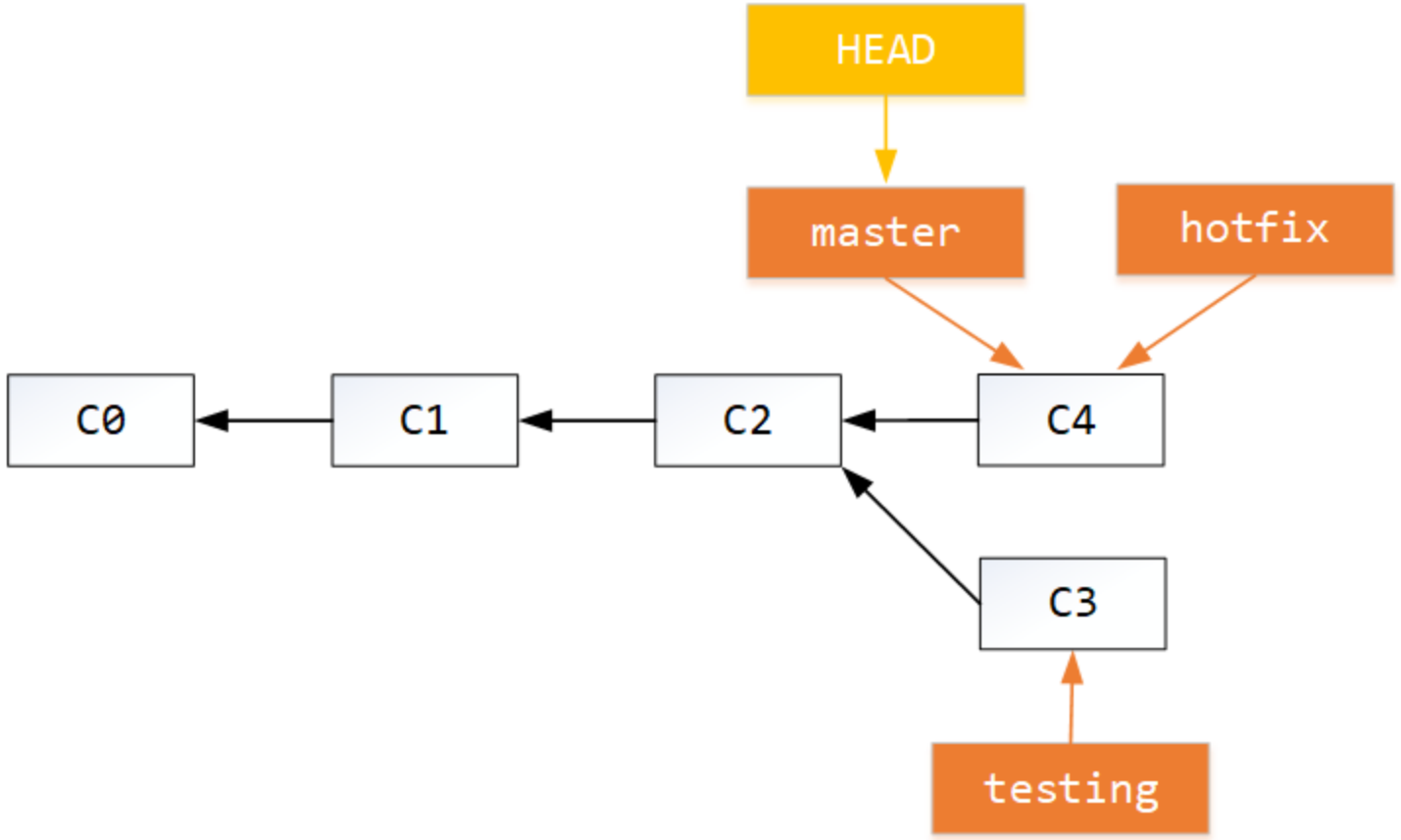
Пример → Переключение ветки на master

```
$ git checkout master
```



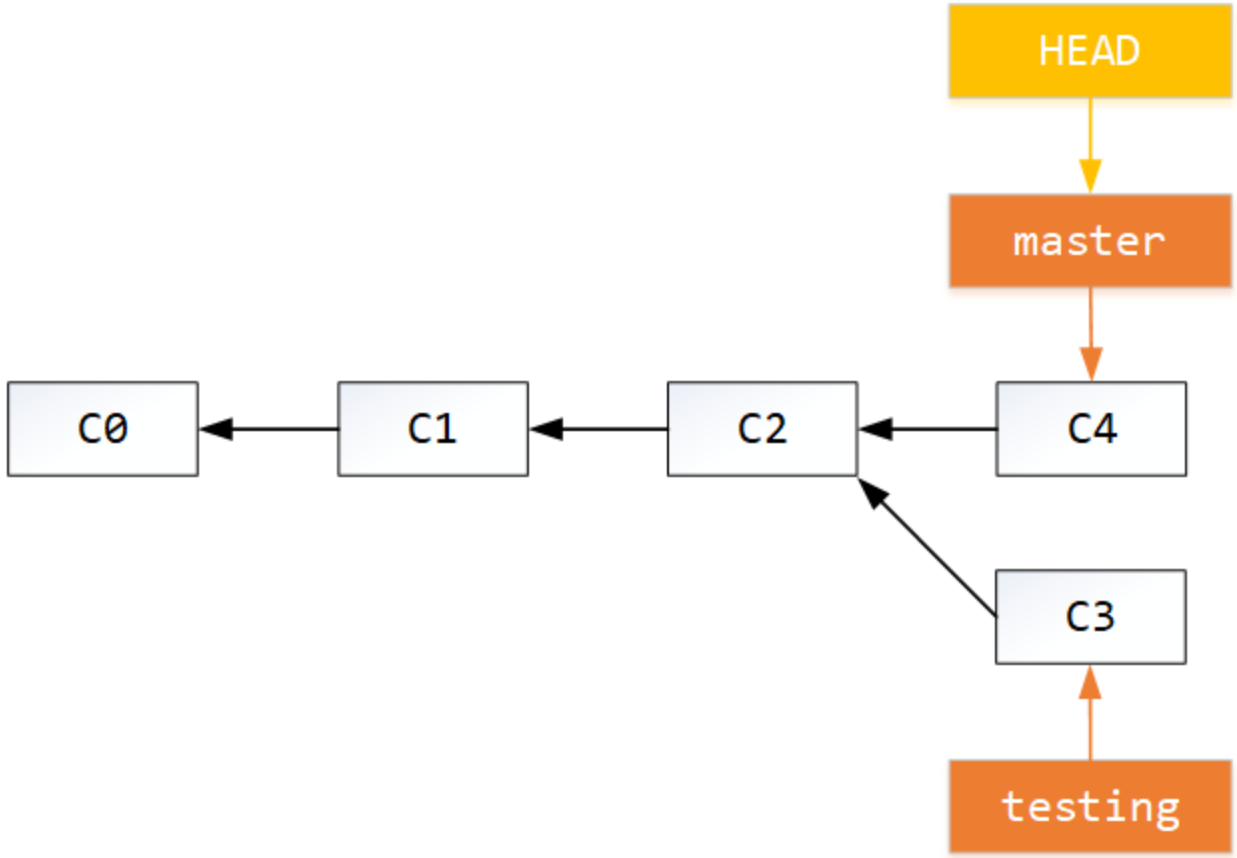
Пример → master перемотан до hotfix

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



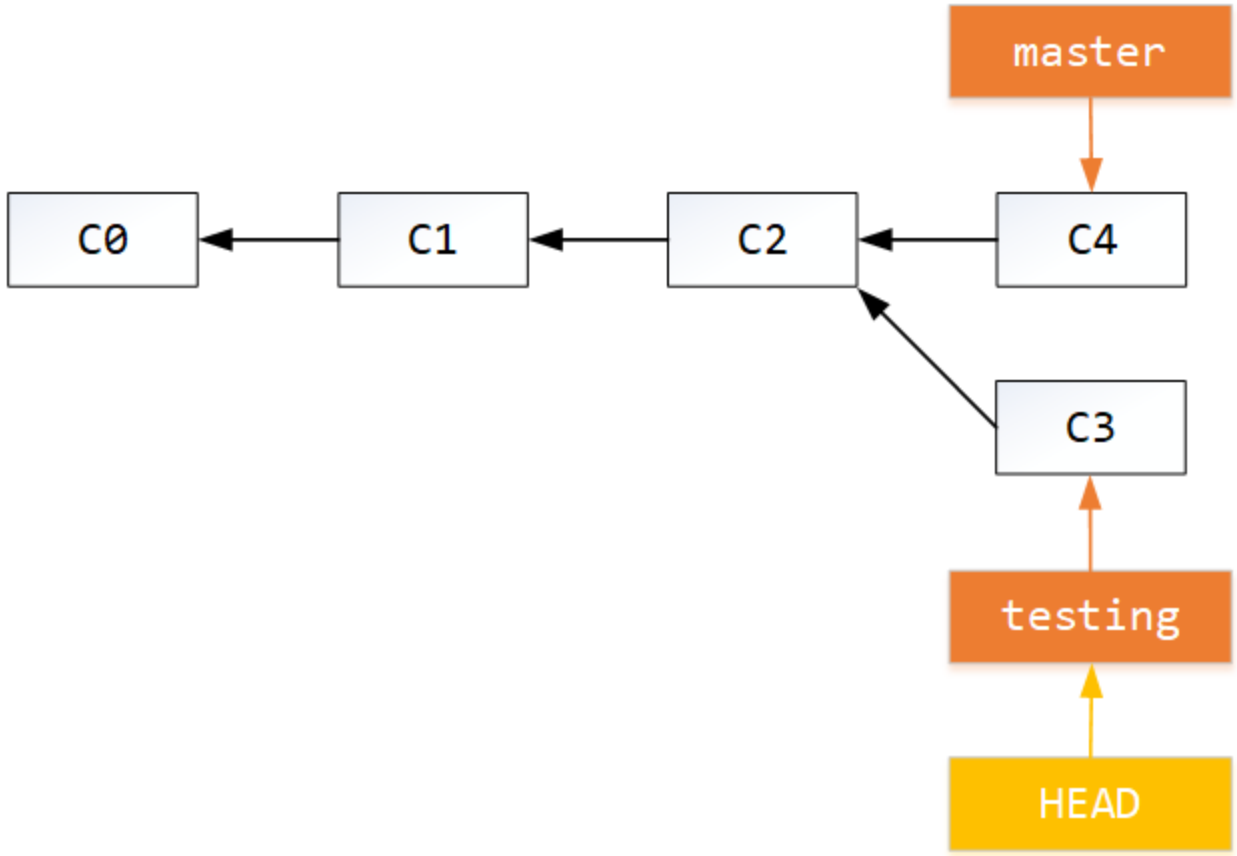
Пример → Удаление ветки hotfix

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



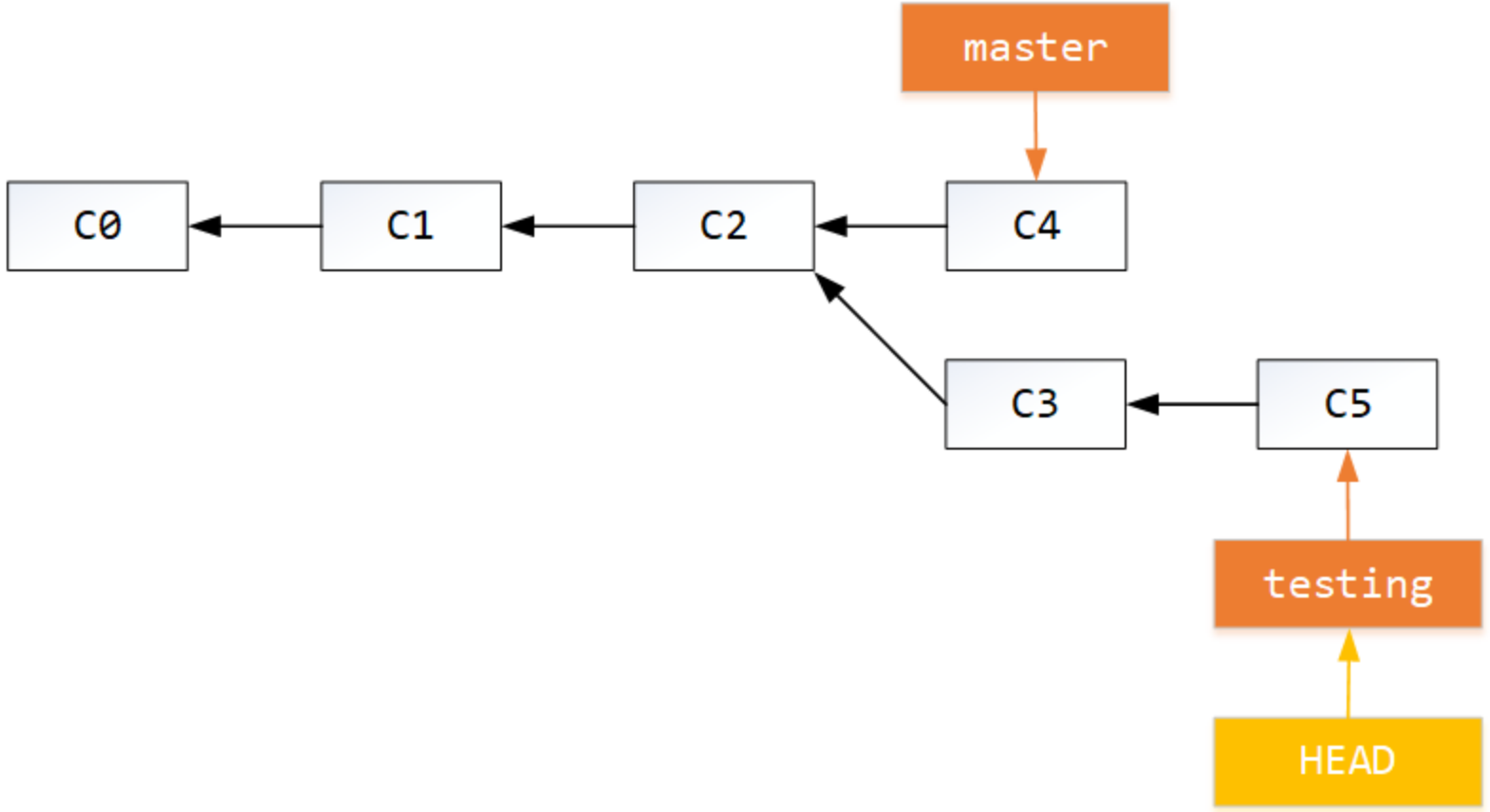
Пример → Переключение на testing

```
$ git checkout testing
Switched to branch "testing"
```



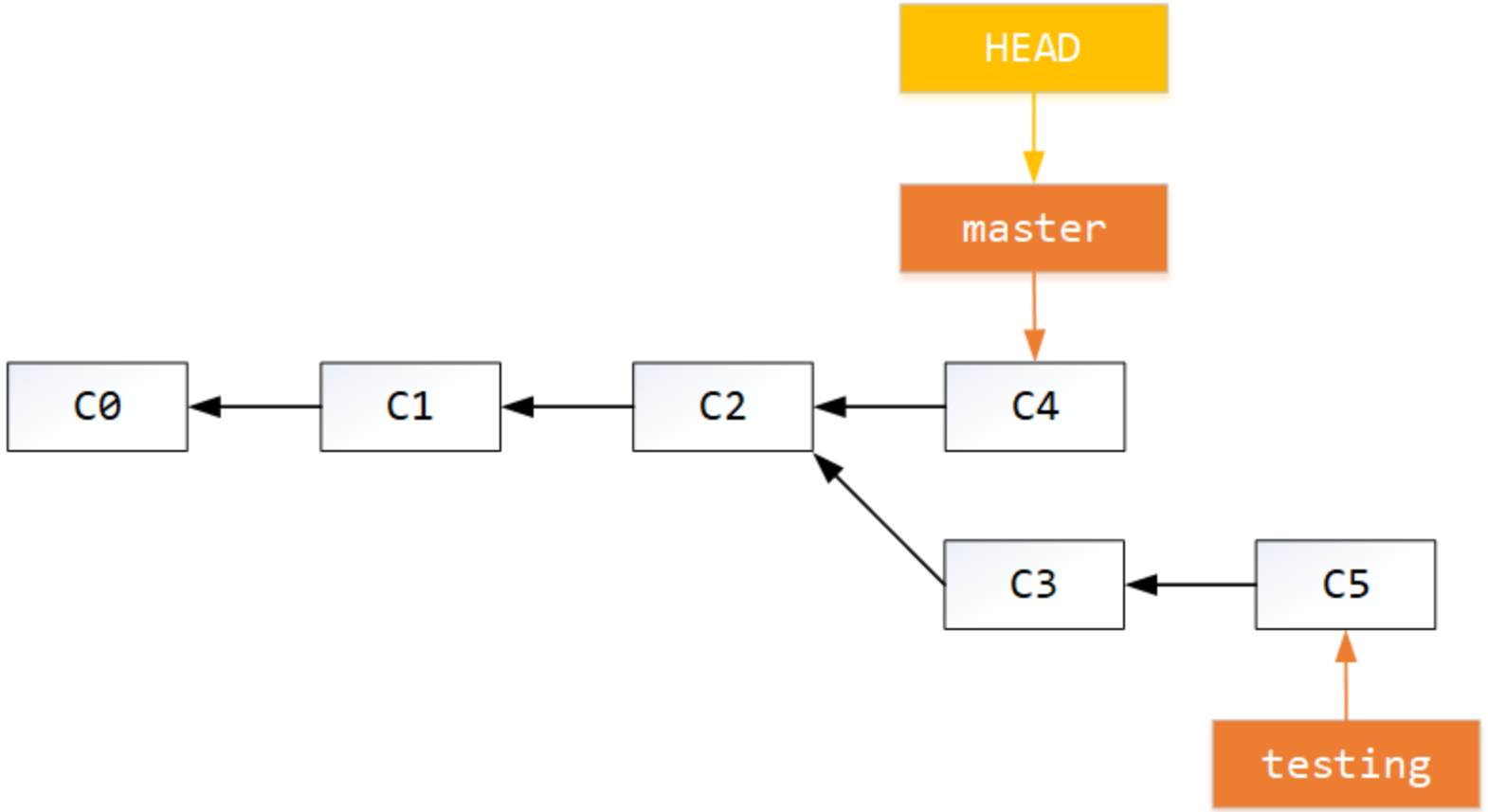
Пример → Продолжение работы над testing

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



Пример → Переключение на master

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



Управление ветками

- Получить список веток
- Создать ветку
- Переключить ветку
- Удалить ветку
- Переименовать ветку

Получить список веток

Команда `git branch` без параметров или `git branch --list` отображает список *локальных* веток. Текущая ветка отмечена символом `*`.

```
$ git branch
  hotfix
* master
  testing
```

Параметр `-v` (`--verbose`) показывает информацию последнего коммита для каждой ветки.

```
$ git branch -v
  hotfix 782fd34 add scott to the author list in the readmes
* master  7a98805 Merge branch 'testing'
  testing  93b412c fix javascript issue
```

Получить список веток (продолжение)

Опция `-a` или `--all` позволяет отобразить список *всех* веток (локальных + удалённых):

```
$ git branch --all
* master
  remotes/origin/master
```

Получить список веток (продолжение)

Опция `--merged` показывает в списке ветки, которые уже влиты в текущую:

```
$ git branch --merged
  hotfix
* master
```

Опция `--no-merged` показывает в списке ветки, которые ещё *не* влиты в текущую:

```
$ git branch --no-merged
  testing
```

Создать ветку

Команда `git branch <branch> [commit]` позволяет создать ветку с именем `branch` на текущем или указанном коммите `commit`.

Пример:

```
$ git branch
* master

$ git branch jira_234

$ git branch
  jira_234
* master
```

Переключить ветку

Команда `git checkout <branch>` позволяет переключиться на указанную ветку `branch`. Она обновляет содержимое рабочей папки согласно последнему коммиту в этой ветке.

```
$ git checkout jira_234
Switched to branch 'jira_234'

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

Удалить ветку

Команда `git branch -d <branch>` *безопасно* удаляет указанную ветку:

```
$ git branch -d jira_234
error: The branch 'jira_234' is not fully merged.
If you are sure you want to delete it, run 'git branch -D jira_234'.
```

Команда `git branch -D <branch>` *гарантированно* удаляет указанную ветку:

```
$ git branch -D jira_234
Deleted branch jira_234 (was 7d7abf4).
```

Переименовать ветку

Команда `git branch -m <newname>` переименовывает текущую ветку в `<newname>`.

```
git branch -m <newname>
git branch -m <oldname> <newname>
```

Практика 💪

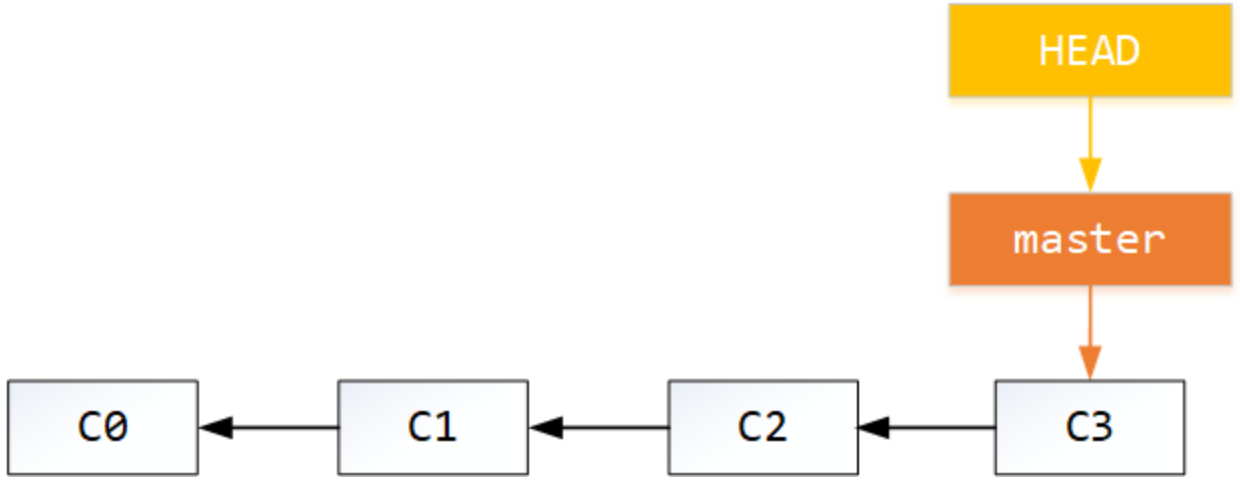
1. Получите список локальных веток.
2. Получите список всех веток.
3. Создайте локальную ветку `test` и переключитесь на неё.
4. Добавьте несколько файлов в рабочей директории и создайте новый коммит.
5. Посмотрите историю коммитов
6. Переключитесь на ветку `master` (`main`). Убедитесь, что файлы созданные на шаге 4 отсутствуют. Почему так?
7. Отобразите список всех коммитов всех веток в виде графа (`git log --graph --oneline --decorate --all`)

Отсоединённый указатель

Отсоединённый указатель - это такое состояние, когда указатель HEAD ссылается не на ветку, а непосредственно на коммит.

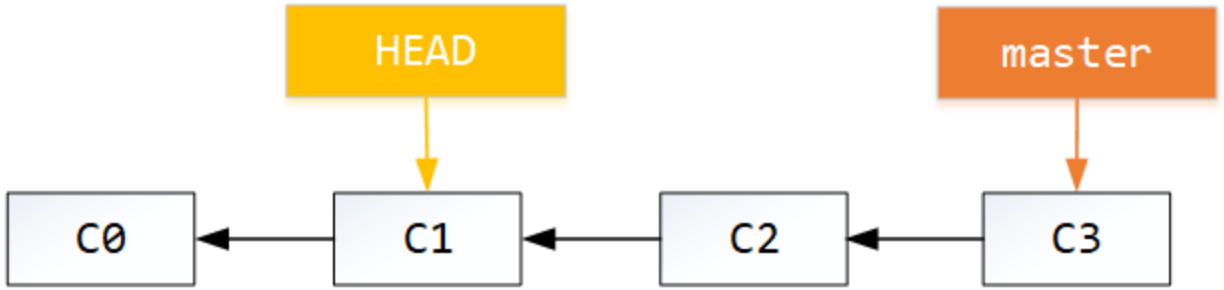
Пример → Начальное состояние

```
$ git log --oneline --decorate --graph --all
- beb06f0 (HEAD -> master) C3
- 40a5902 C2
- 2074654 C1
- c9d2e7e C0
```

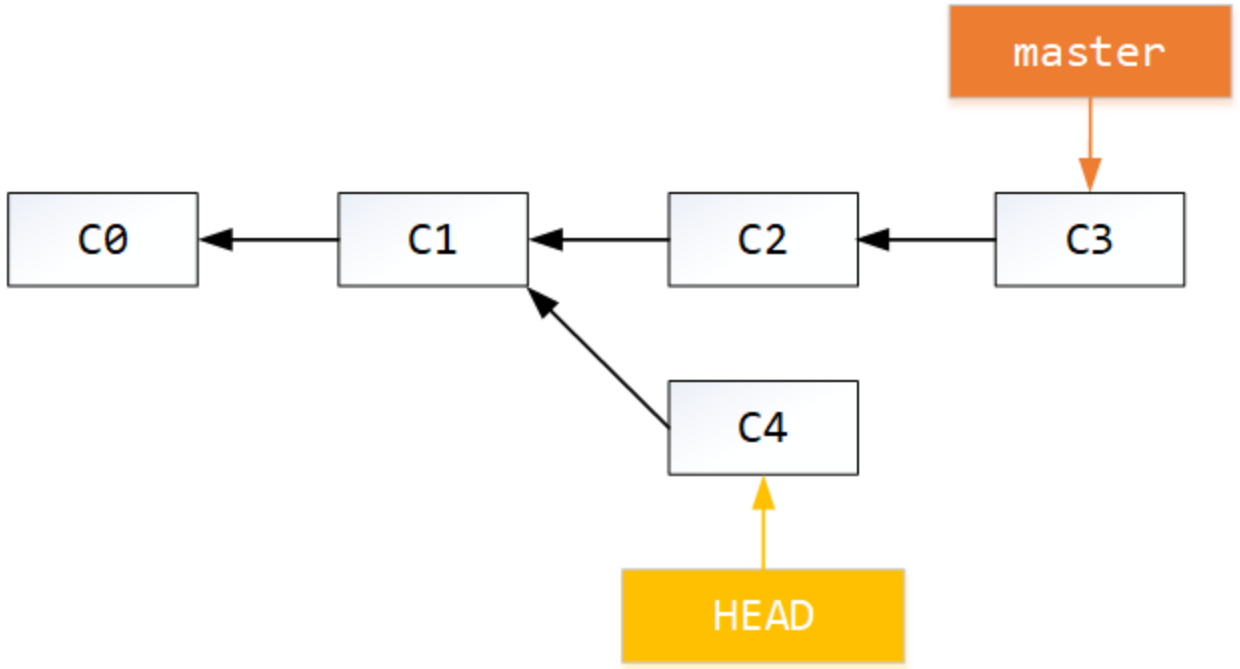


Пример → Изменение указателя HEAD на произвольный коммит

```
$ git checkout 2074654
Note: checking out '2074654'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

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

  git checkout -b <new-branch-name>

HEAD is now at 2074654 C1
```



```
$ git commit -m "C4"
[detached HEAD 215b848] C4
 1 file changed, 1 insertion(+)
```




```
$ git log --oneline --decorate --graph --all
- 215b848 (HEAD) C4
| * beb06f0 (master) C3
| * 40a5902 C2
|/
- 2074654 C1
```

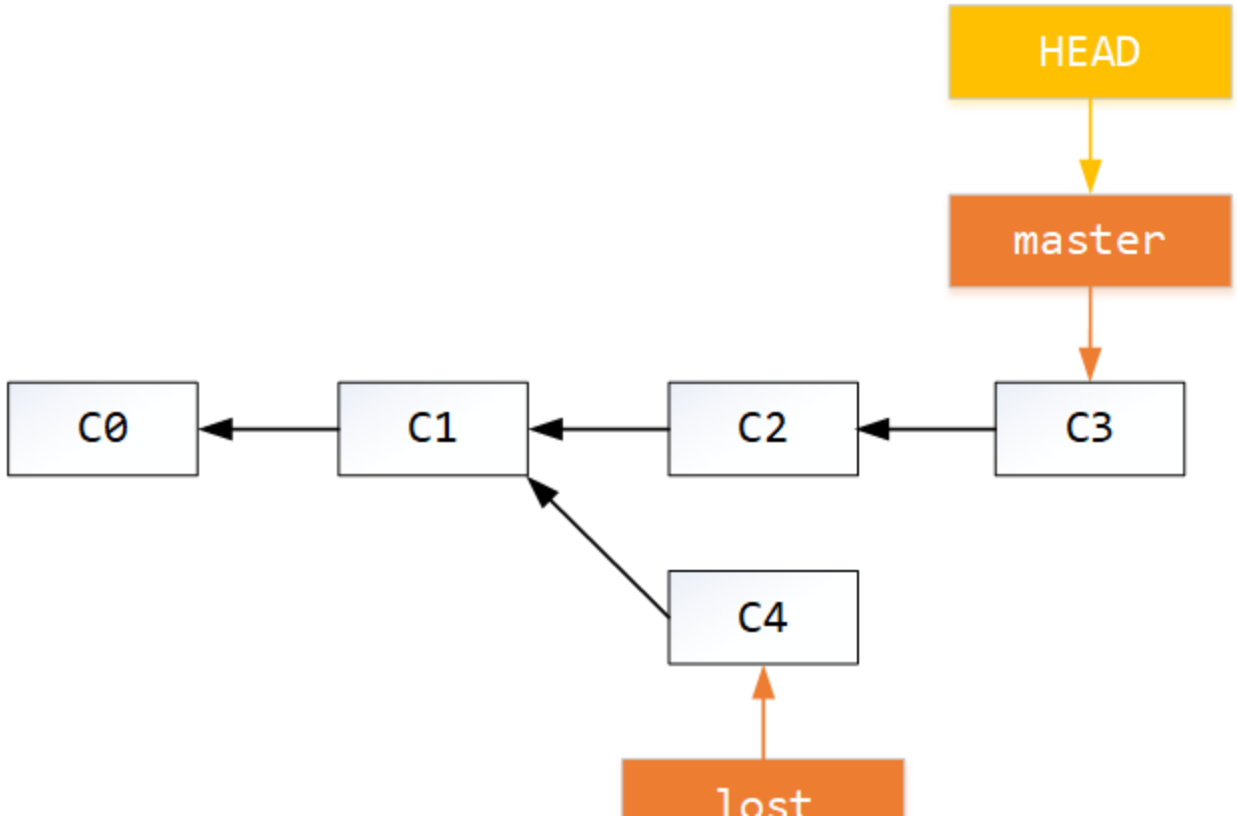
Пример → Выход из detached HEAD

```
$ git checkout master
Warning: you are leaving 1 commit behind, not connected to
any of your branches:

  215b848 C4

If you want to keep it by creating a new branch, this may be a good time
to do so with:

 git branch <new-branch-name> 215b848

Switched to branch 'master'
```

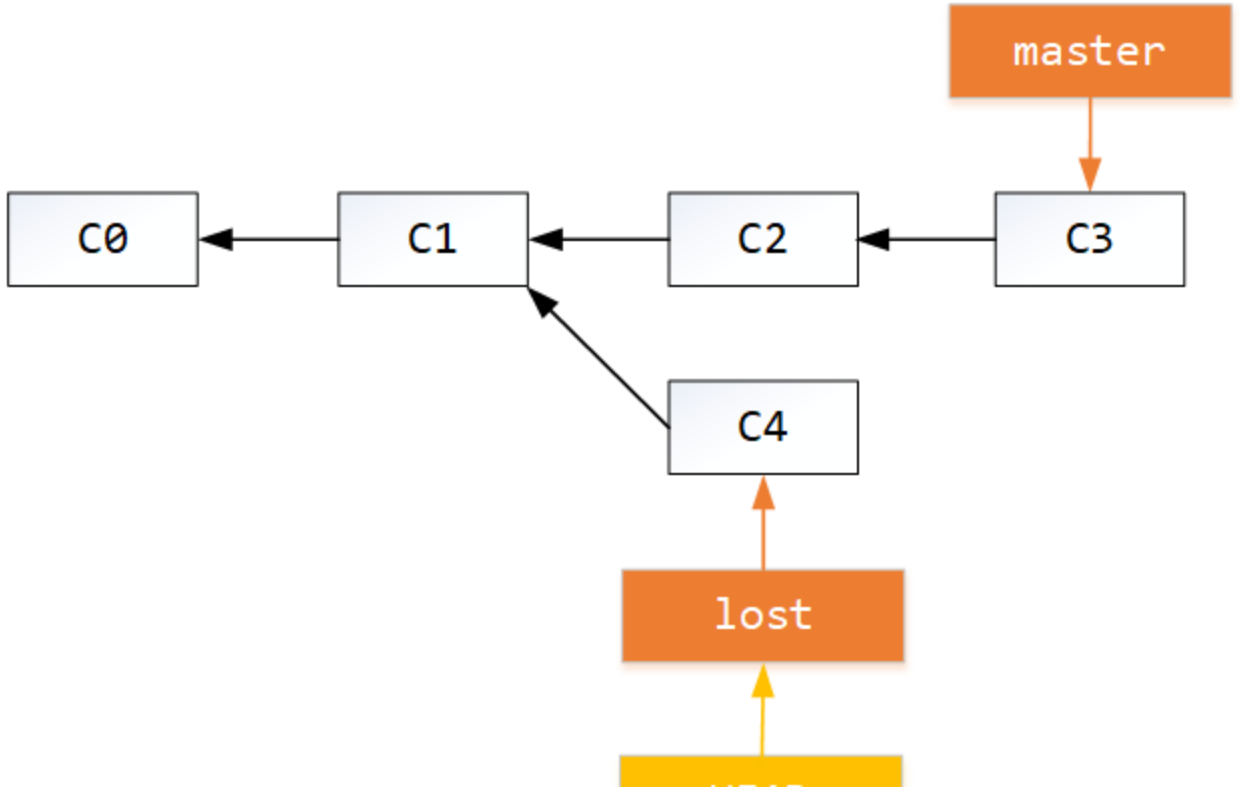
```
$ git log --oneline --decorate --graph --all
- beb06f0 (HEAD -> master) C3
- 40a5902 C2
- 2074654 C1
- c9d2e7e C0
```

Пример – создание ветки на произвольном коммите

```
$ git branch lost 215b848
$ git log --oneline --decorate --graph --all
- 215b848 (lost) C4
| * beb06f0 (HEAD -> master) C3
| * 40a5902 C2
|/
- 2074654 C1
- c9d2e7e C0
```



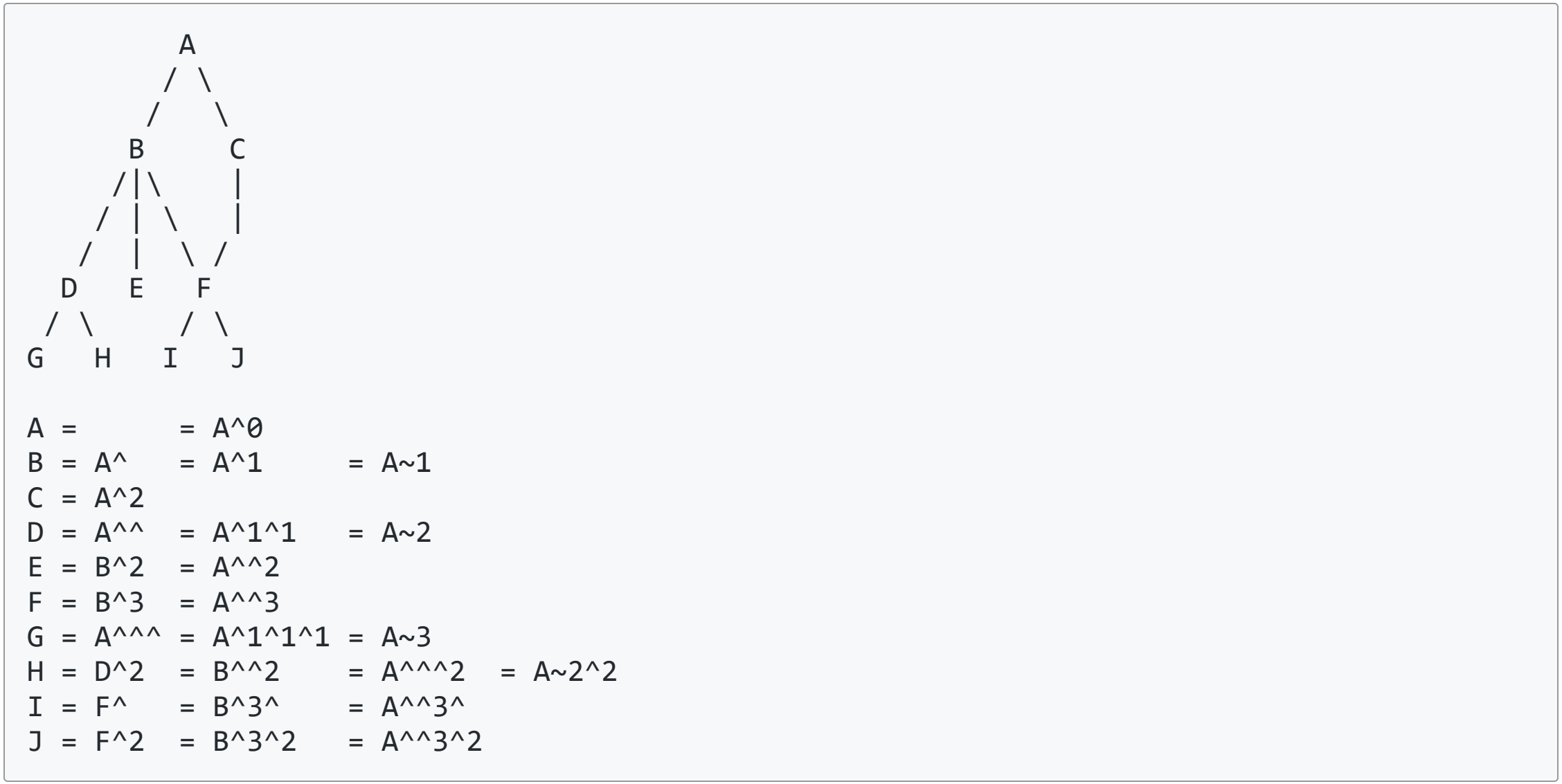
```
$ git checkout lost
Switched to branch 'lost'
$ git log --oneline --decorate --graph --all
- 215b848 (HEAD -> lost) C4
| * beb06f0 (master) C3
| * 40a5902 C2
|/
- 2074654 C1
- c9d2e7e C0
```



Пример → Просмотр истории изменения HEAD

```
$ git reflog
215b848 (HEAD -> lost) HEAD@{0}: checkout: moving from master to lost
beb06f0 (master) HEAD@{1}: checkout: moving from 215b8480640ed73fa1f43586f561ecfe27aa4ba3 to master
215b848 (HEAD -> lost) HEAD@{2}: commit: C4
2074654 HEAD@{3}: checkout: moving from master to 2074654
beb06f0 (master) HEAD@{4}: commit: C3
40a5902 HEAD@{5}: commit: C2
2074654 HEAD@{6}: commit: C1
c9d2e7e HEAD@{7}: commit (initial): C0
```

Ссылки на коммиты



Практика 💪

1. Переключитесь на ветку `master`
2. Посмотрите историю
3. Переключитесь на третий коммит с конца `master`, используя `HEAD~3`
    - Получили состояние detached HEAD?
4. Сделайте изменения и создайте новый коммит
    - Посмотрите граф истории для всех веток

Практика (продолжение)

5. Переключитесь на ветку `master`
    - Посмотрите граф истории для всех веток
    - Виден ли ваш последний созданный коммит? Почему?
6. Найдите ссылку на коммит в выводе `git reflog`
7. Создайте ветку на этом коммите
    - Посмотрите граф истории для всех веток
    - Виден ли этот коммит теперь?

Осиротелые коммиты

"Осиротелые" коммиты не имеют родительских коммитов.

Необходимы, чтобы вести несколько независимых историй в одном репозитории, например:

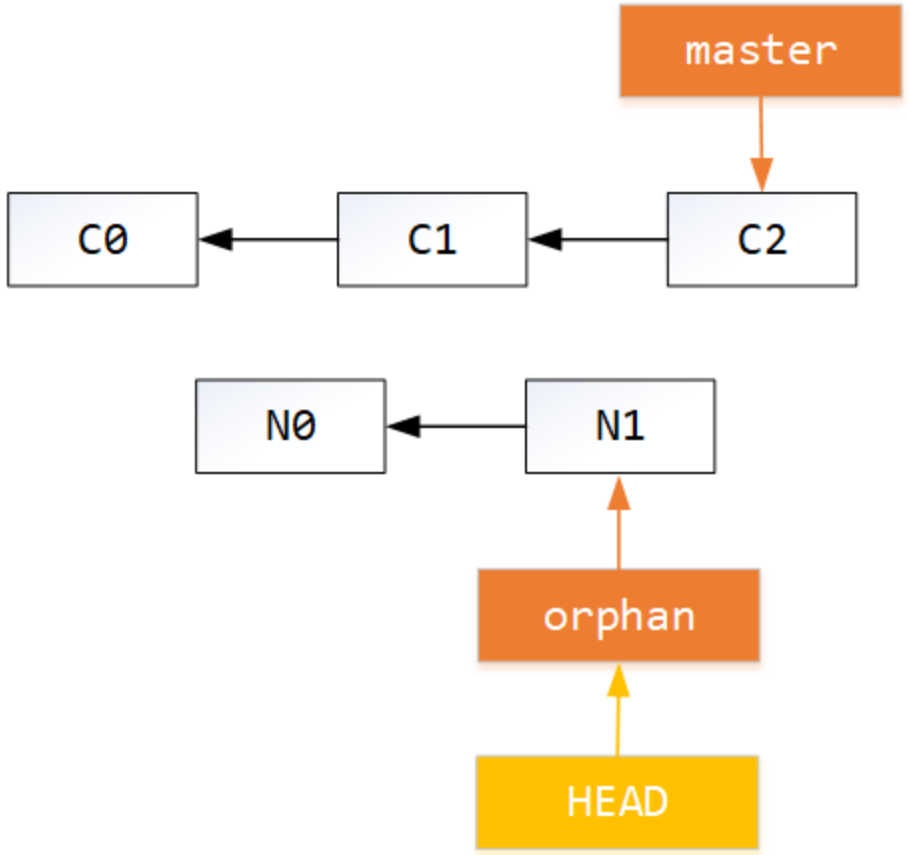
- вспомогательная конфигурация проекта в отдельной ветке
- несколько проектов в одном репозитории
- документация и программный код разделены

Создаются с помощью команды

```
git checkout --orphan <name>
git rm -rf .
<do work>
git add <your files>
git commit -m 'Initial commit'
```

Изначально файлы с прошлого коммита добавлены в индекс.

Осиротелые коммиты (продолжение)



Заключение

Ветка в Git - это именнованый указатель на коммит.

`HEAD` - указатель на текещую ветку или текущий коммит (состояние *detached HEAD*).

Команда `git branch` позволяет управлять ветками:

- посмотреть список веток
- создать ветку
- удалить ветку
- переименовать ветку
- другое (см. справку)

Команда `git checkout` позволяет переключиться на указанную ветку или коммит (состояние *detached HEAD*).

Осиротелые коммиты позволяет иметь несколько корней у репозитория.

Ваши вопросы ?