

Основы Git



Андрей
Борю



Андрей Борю

Principal DevOps Engineer, Snapcart





План занятия

1. [История коммитов](#)
2. [Операции отмены](#)
3. [Работа с удаленными репозиториями](#)
4. [Работа с тегами \(метками\)](#)
5. [Киллер фича – ветвления](#)
6. [Итоги](#)
7. [Домашнее задание](#)



История коммитов

Просмотр истории коммитов

Если в папке клонированного проекта вы запустите команду **git log**, увидите следующий вывод.

```
$ git clone git@github.com:netology-code/devops-homeworks.git
```

```
$ git log
```

```
commit 473b6e578b23ee685ac057676eecda9c4568c59f (HEAD -> master, origin/master)
```

```
Author: andrey.borue <andrey.borue@gml.com>
```

```
Date: Sat Apr 18 12:18:22 2020 +0800
```

```
Homework Intro draft
```

```
commit 188f7a0a1e285ae906a49a1c8b98cc3606d9a364
```

```
Author: andrey.borue <andrey.borue@gml.com>
```

```
Date: Sat Apr 18 12:13:36 2020 +0800
```

```
First commit
```

История: патч

Полезным аргументом является **--patch**, который показывает разницу (выводит патч), внесенную в каждый коммит. Вы можете ограничить количество записей в выводе команды; используйте параметр **-1** для вывода только одной записи:

```
$ git log --patch -1 (или git log -p -1)
```

```
commit 473b6e578b23ee685ac057676eecda9c4568c59f (HEAD -> master, origin/master)
```

```
Author: andrey.borue <andrey.borue@gml.com>
```

```
Date: Sat Apr 18 12:18:22 2020 +0800
```

```
Homework Intro draft
```

```
diff --git a/intro/README.md b/intro/README.md
```

```
new file mode 100644
```

```
index 0000000..9992ca8
```

```
--- /dev/null
```

```
+++ b/intro/README.md
```

```
@@ -0,0 +1,3 @@
```

```
+# Домашнее задание к занятию «1.1. Введение в DevOps»
```

```
+
```

```
### Задание №1 - Установка PyCharm и плагинов к нему
```

История: статистика

Опция **--stat** отображает аналогичную информацию, но содержит разницу для каждой записи. Удобно использовать для код ревью или быстрого просмотра серии внесенных изменений.

\$ git log --stat -2

```
commit 473b6e578b23ee685ac057676eecda9c4568c59f (HEAD -> master, origin/master)
```

```
Author: andrey.borue <andrey.borue@gml.com>
```

```
Date: Sat Apr 18 12:18:22 2020 +0800
```

```
Homework Intro draft
```

```
intro/README.md | 3 +++
```

```
1 file changed, 3 insertions(+)
```

```
commit 188f7a0a1e285ae906a49a1c8b98cc3606d9a364
```

```
Author: andrey.borue <andrey.borue@gml.com>
```

```
Date: Sat Apr 18 12:13:36 2020 +0800
```

```
First commit
```

```
README.md | 3 +++
```

```
1 file changed, 3 insertions(+)
```

История: форматированный вывод

Следующей полезной опцией является **--pretty**. Эта опция меняет формат вывода.

```
$ git log --pretty=oneline (или short, full, fuller)
```

```
473b6e578b23ee685ac057676eecda9c4568c59f (HEAD -> master, origin/master) Homework Intro draft
```

```
188f7a0a1e285ae906a49a1c8b98cc3606d9a364 First commit
```

Интересной опцией является **format**, которая позволяет указать формат для вывода данных.

```
$ git log --pretty=format:"%h - %an, %ar : %s"
```

```
473b6e5 - andrey.borue, 72 minutes ago : Homework Intro draft
```

```
188f7a0 - andrey.borue, 77 minutes ago : First commit
```


Опции форматирования

- %H** Хеш коммита
- %h** Сокращенный хеш коммита
- %T** Хеш дерева
- %t** Сокращенный хеш дерева
- %P** Хеш родителей
- %p** Сокращенный хеш родителей
- %an** Имя автора
- %ae** Электронная почта автора
- %ad** Дата автора (формат даты можно задать опцией --date=option)
- %ar** Относительная дата автора
- %cn** Имя коммитера
- %ce** Электронная почта коммитера
- %cd** Дата коммитера
- %cr** Относительная дата коммитера
- %s** Содержание



Автор и коммитер

Автор – это человек, изначально сделавший работу.

Коммитер – это человек, который последним применил эту работу.

Другими словами, если вы создадите патч для какого-то проекта, а один из основных членов команды этого проекта применит этот патч, вы оба получите статус участника – вы как автор и основной член команды как коммитер.

Опции git log

--p	Показывает патч для каждого коммита.
--stat	Показывает статистику измененных файлов для каждого коммита.
--shortstat	Отображает только строку с количеством изменений/вставок/удалений для команды --stat.
--name-only	Показывает список измененных файлов после информации о коммите.
--name-status	Показывает список файлов, которые добавлены/изменены/удалены.
--abbrev-commit	Показывает только несколько символов SHA-1 чек-суммы вместо всех 40.
--relative-date	Отображает дату в относительном формате (например, "2 weeks ago") вместо стандартного формата даты.
--graph	Отображает ASCII граф с ветвлениями и историей слияний.
--pretty	Показывает коммиты в альтернативном формате. Возможные варианты опций: oneline, short, full, fuller и format (с помощью последней можно указать свой формат).
--oneline	Сокращение для одновременного использования опций --pretty=oneline --abbrev-commit.

Ограничение вывода git log

-(n)	Показывает только последние n коммитов.
--since, --after	Показывает только те коммиты, которые были сделаны после указанной даты: "2008-01-15" или же относительную дату, например "2 years 1 day 3 minutes ago".
--until, --before	Показывает только те коммиты, которые были сделаны до указанной даты.
--author	Показывает только те коммиты, в которых запись author совпадает с указанной строкой.
--committer	Показывает только те коммиты, в которых запись committer совпадает с указанной строкой.
--grep	Показывает только коммиты, сообщение которых содержит указанную строку.
-S	Показывает только коммиты, в которых изменение в коде повлекло за собой добавление или удаление указанной строки.
--no-merges	Не отображать коммиты слияния (мержа).



Операции отмены



Внимание!

Операции отмены - это одна из редких областей Git, где неверными действиями можно необратимо удалить результаты своей работы.

Очень важно понимать, что когда вы вносите правки в последний коммит, вы не столько исправляете его, сколько заменяете новым, который полностью его перезаписывает. В результате всё выглядит так, будто первоначальный коммит никогда не существовал, а так же он больше не появится в истории вашего репозитория.

Очевидно, смысл изменения коммитов в добавлении незначительных правок в последние коммиты и, при этом, в избежании засорения истории сообщениями вида “Ой, забыл добавить файл” или “Исправление грамматической ошибки”.

git commit --amend

Отмена может потребоваться, если вы сделали коммит слишком рано, например, забыв добавить какие-то файлы или комментарий к коммиту. Если вы хотите переделать коммит — внесите необходимые изменения, добавьте их в индекс и сделайте коммит ещё раз, указав параметр **--amend**.

```
$ git commit -m 'initial commit'
```

```
$ git add forgotten_file
```

```
$ git commit --amend
```

В итоге получится единый коммит — второй коммит заменит результаты первого.

Отмена индексации файла

Команда, которой вы определяете состояние этих областей, также подсказывает вам как отменять изменения в них. Например, вы изменили два файла и хотите добавить их в разные коммиты, но случайно выполнили команду **git add *** и добавили в индекс оба. Как исключить из индекса один из них? Команда **git status** напомнит вам.

```
$ git add *
```

```
$ git status
```

```
On branch master
```

```
Changes to be committed:
```

```
(use "git reset HEAD <file>..." to unstage)
```

```
renamed:    README.md -> README
```

```
modified:   CONTRIBUTING.md
```


Отмена индексации файла

Команда может показаться несколько странной, но это работает. Файл CONTRIBUTING.md изменен, но больше не добавлен в индекс.

```
$ git reset HEAD CONTRIBUTING.md
```

```
Unstaged changes after reset:
```

```
M    CONTRIBUTING.md
```

```
$ git status
```

```
On branch master
```

```
Changes to be committed:
```

```
(use "git reset HEAD <file>..." to unstage)
```

```
renamed: README.md -> README
```

```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git checkout -- <file>..." to discard changes in working directory)
```

```
modified: CONTRIBUTING.md
```



Внимание!

Команда `git reset` **может** быть опасной если вызвать её с параметром `--hard`. В приведенном примере файл не был затронут, следовательно команда относительно безопасна.

Отмена изменений в файле

Если вы поняли, что не хотите сохранять свои изменения файла CONTRIBUTING.md, вы можете отменить изменения — вернуть к тому состоянию, которое было в последнем коммите.

```
$ git status
```

```
...
```

```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git checkout -- <file>..." to discard changes in working directory)
```

```
modified: CONTRIBUTING.md
```

```
$ git checkout -- CONTRIBUTING.md
```

```
$ git status
```

```
On branch master
```

```
Changes to be committed:
```

```
(use "git reset HEAD <file>..." to unstage)
```


```
renamed: README.md -> README
```



Внимание!

Важно понимать, что **git checkout -- <file>** — опасная команда.

Все локальные изменения в файле пропадут — Git просто заменит его версией из последнего коммита. Ни в коем случае не используйте эту команду, если вы не уверены, что изменения в файле вам не нужны.



Работа с удаленными репозиториями

Просмотр удалённых репозиториев

Чтобы просмотреть список настроенных удалённых репозиториев, вы можете запустить команду **git remote**. Если вы клонировали репозиторий, то увидите как минимум **origin** — имя по умолчанию, которое Git даёт серверу, с которого производилось клонирование.

Вы можете также указать ключ **-v**, чтобы просмотреть адреса для чтения и записи, привязанные к репозиторию.

```
$ git clone git@github.com:netology-code/devops-homeworks.git
```

```
$ cd devops-homeworks
```

```
$ git remote
```

```
origin
```

```
$ git remote -v
```

```
origin git@github.com:netology-code/devops-homeworks.git (fetch)
```

```
origin git@github.com:netology-code/devops-homeworks.git (push)
```

Несколько удалённых репозиториев

Если у вас больше одного удалённого репозитория, команда выведет их все. Для репозитория с несколькими настроенными удалёнными репозиториями в случае совместной работы нескольких пользователей, вывод команды может выглядеть так:

```
$ git remote -v
```

```
bakkdoor https://github.com/bakkdoor/grit (fetch)
```

```
bakkdoor https://github.com/bakkdoor/grit (push)
```

```
cho45 https://github.com/cho45/grit (fetch)
```

```
cho45 https://github.com/cho45/grit (push)
```

```
defunkt https://github.com/defunkt/grit (fetch)
```

```
defunkt https://github.com/defunkt/grit (push)
```

```
koke git://github.com/koke/grit.git (fetch)
```

```
koke git://github.com/koke/grit.git (push)
```

```
origin git@github.com:mojombo/grit.git (fetch)
```

```
origin git@github.com:mojombo/grit.git (push)
```

Добавление удалённых репозиториев

Чтобы добавить удалённый репозиторий и присвоить ему имя (shortname), просто выполните команду **git remote add <shortname> <url>**

```
$ git remote
```

```
origin
```

```
$ git remote add pb https://github.com/paulboone/ticgit
```

```
$ git remote -v
```

```
origin      https://github.com/schacon/ticgit (fetch)
```

```
origin      https://github.com/schacon/ticgit (push)
```

```
pb          https://github.com/paulboone/ticgit (fetch)
```

```
pb          https://github.com/paulboone/ticgit (push)
```


Получение изменений

Теперь вместо указания полного пути вы можете использовать **pb**. Например, если вы хотите получить изменения, которые есть у другого, но нет у вас, вы можете выполнить команду **git fetch pb**.

```
git fetch [remote-name]
```

```
$ git fetch pb
```

```
remote: Counting objects: 43, done.  
remote: Compressing objects: 100% (36/36), done.  
remote: Total 43 (delta 10), reused 31 (delta 5)  
Unpacking objects: 100% (43/43), done.  
From https://github.com/paulboone/ticgit  
* [new branch]    master    -> pb/master  
* [new branch]    ticgit    -> pb/ticgit
```



git fetch

Данная команда связывается с указанным удалённым проектом и забирает все те данные проекта, которых у вас ещё нет.

После того как вы выполнили команду, у вас должны появиться ссылки на все ветки из этого удалённого проекта, которые вы можете просмотреть или слить в любой момент.



git pull

Как правило, извлекает (**fetch**) данные с сервера, с которого вы изначально клонировали, и автоматически пытается слить (**merge**) их с кодом, над которым вы в данный момент работаете.

Отправка изменений

```
git push <remote-name> <branch-name>
```

```
$ echo "test" >> README.md
```

```
$ git add README.md
```

```
$ git commit -m "test commit"
```

```
[master 46906d6] test commit
```

```
1 file changed, 1 insertion(+)
```

```
$ git push
```

```
Enumerating objects: 5, done.
```

```
Counting objects: 100% (5/5), done.
```

```
Delta compression using up to 12 threads
```

```
Compressing objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 297 bytes | 297.00 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:netology-code/devops-homeworks.git
```

```
473b6e5..46906d6 master -> master
```

Просмотр удалённого репозитория

```
$ git remote show origin
```

```
* remote origin
```

```
URL: https://github.com/my-org/complex-project
```

```
Fetch URL: https://github.com/my-org/complex-project
```

```
Push URL: https://github.com/my-org/complex-project
```

```
HEAD branch: master
```

```
Remote branches:
```

master	tracked
dev-branch	tracked
markdown-strip	tracked
issue-43	new (next fetch will store in remotes/origin)
issue-45	new (next fetch will store in remotes/origin)
refs/remotes/origin/issue-11	stale (use 'git remote prune' to remove)

```
Local branches configured for 'git pull':
```

```
dev-branch merges with remote dev-branch
```

```
master merges with remote master
```

```
Local refs configured for 'git push':
```

dev-branch	pushes to dev-branch	(up to date)
markdown-strip	pushes to markdown-strip	(up to date)
master	pushes to master	(up to date)

Переименование и удаление

Для переименования удалённого репозитория можно выполнить **git remote rename**. Например, если вы хотите переименовать **pb** в **paul**.

Если по какой-то причине вы хотите удалить удаленный репозиторий, вы можете использовать **git remote rm**.

```
$ git remote
```

```
origin
```

```
pb
```

```
$ git remote rename pb paul
```

```
$ git remote
```

```
origin
```

```
paul
```

```
$ git remote rm paul
```

```
$ git remote
```

```
origin
```



Работа с тегами (метками)



Что такое теги

Как и большинство VCS, Git имеет возможность пометить определённые моменты в истории как важные.

Как правило, эта функциональность используется для отметки моментов выпуска версий (v1.0, и т.п.).

Просмотр списка тегов

```
$ git tag
```

```
v0.1
```

```
v1.3
```

```
$ git tag -l 'v1.8.5*' (-l это сокращение к --list)
```

```
v1.8.5
```

```
v1.8.5-rc0
```

```
v1.8.5-rc1
```

```
v1.8.5-rc2
```

```
v1.8.5-rc3
```

```
v1.8.5.1
```

```
v1.8.5.2
```

Виды тегов

- Легковесные
- Аннотированные

Аннотированные теги

```
$ git tag -a v0.0 -m 'init version'
```

```
$ git tag
```

```
v0.0
```

```
$ git show v0.0
```

```
tag v0.0
```

```
Tagger: andrey.borue <andrey.borue@gml.com>
```

```
Date: Sat Apr 18 16:14:28 2020 +0800
```

```
init version
```

```
commit 46906d6836ea1f649788579e1c7f16afecc38b01 (HEAD -> master, tag: v0.0,  
origin/master)
```

```
Author: andrey.borue <andrey.borue@gml.com>
```

```
Date: Sat Apr 18 15:38:19 2020 +0800
```

```
test commit
```

```
diff --git a/README.md b/README.md
```

```
...
```

Легковесные теги

```
$ git tag v0.2
```

```
$ git tag
```

```
v0.0
```

```
v0.2
```

```
$ git show v0.2
```

```
commit 46906d6836ea1f649788579e1c7f16afecc38b01 (HEAD -> master, tag:
v0.2, tag: v0.1, tag: v0.0, origin/master)
```

```
Author: andrey.borue <andrey.borue@gmail.com>
```

```
Date: Sat Apr 18 15:38:19 2020 +0800
```

```
test commit
```

```
diff --git a/README.md b/README.md
```

```
...
```



Тегирование существующего коммита

```
$ git tag -a v1.2 9fceb02
```



Обмен тегами

```
$ git push origin v0.0
```

```
Enumerating objects: 1, done.
```

```
Counting objects: 100% (1/1), done.
```

```
Writing objects: 100% (1/1), 160 bytes | 160.00 KiB/s, done.
```

```
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To github.com:netology-code/devops-homeworks.git
```

```
* [new tag]      v0.0 -> v0.0
```



Удаление локального тега

```
$ git tag -d v0.2
```

```
Deleted tag 'v0.2' (was 46906d6)
```

Удаление тега из удалённого репозитория

```
$ git push origin :refs/tags/v0.0
```

```
To github.com:netology-code/devops-homeworks.git
```

```
- [deleted]      v0.0
```

```
$ git push origin --delete <tagname>
```


Переход на тег

git checkout v0.0

Note: switching to 'v0.0'.

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 switching back to a branch.

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>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable `advice.detachedHead` to false
HEAD is now at 46906d6 test commit

Создание ветки от тега

```
$ git checkout -b version0 v0.0
```

Switched to a new branch 'version0'

ИЛИ

```
$ git switch -c version0 v0.0
```

Switched to a new branch 'version0'

Псевдонимы (алиасы) команд

```
$ git config --global alias.co checkout
```

```
$ git config --global alias.br branch
```

```
$ git config --global alias.ci commit
```

```
$ git config --global alias.st status
```

```
$ git config --global alias.unstage 'reset HEAD --'
```

```
$ git config --global alias.last 'log -1 HEAD'
```

```
$ git config --global alias.visual '!gitk'
```



Промежуточный итог



Чему мы научились

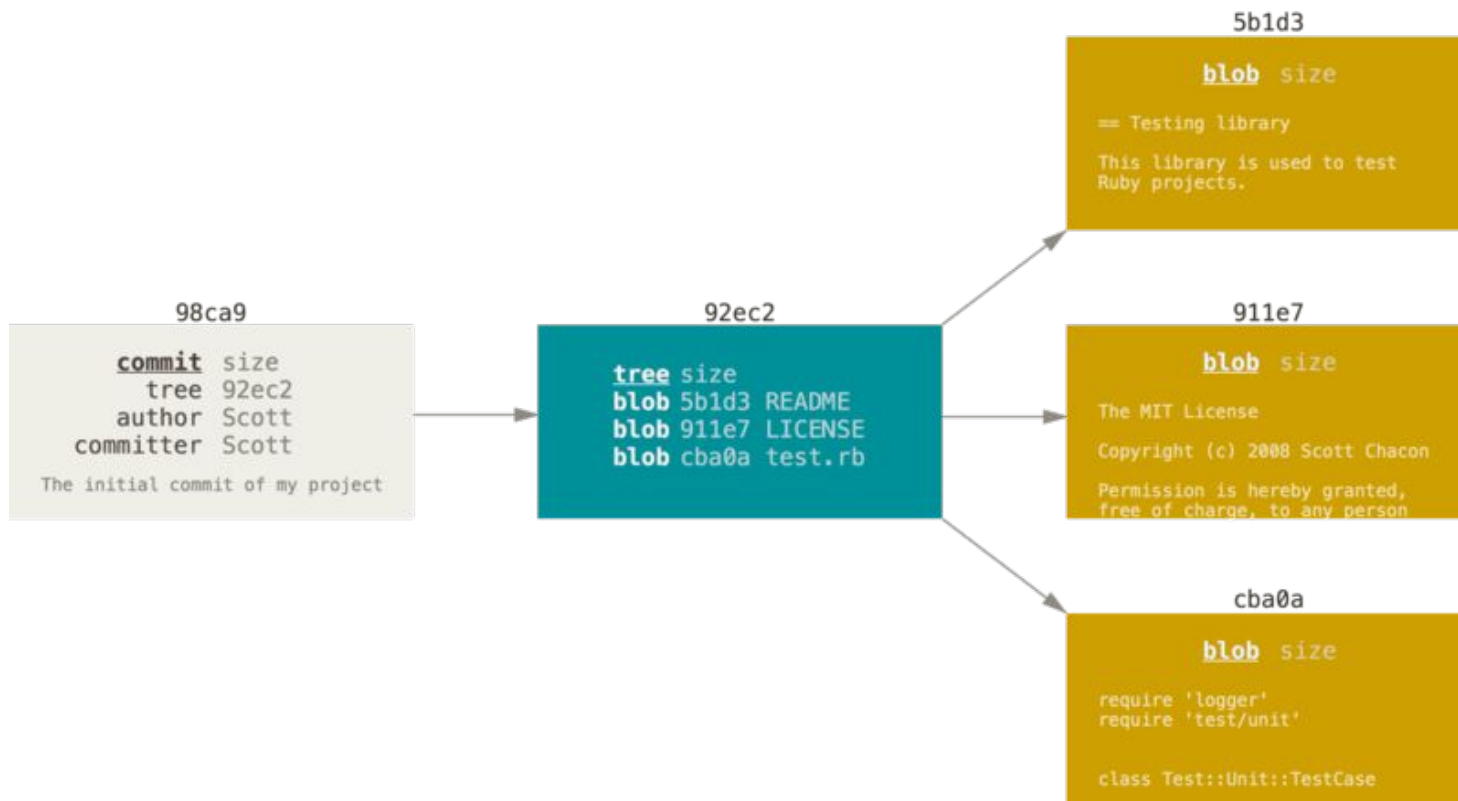
- Создавать и клонировать репозиторий
- Вносить изменения
- Индексировать и фиксировать эти изменения
- Просматривать историю всех изменений в репозитории
- Помечать коммиты тегами



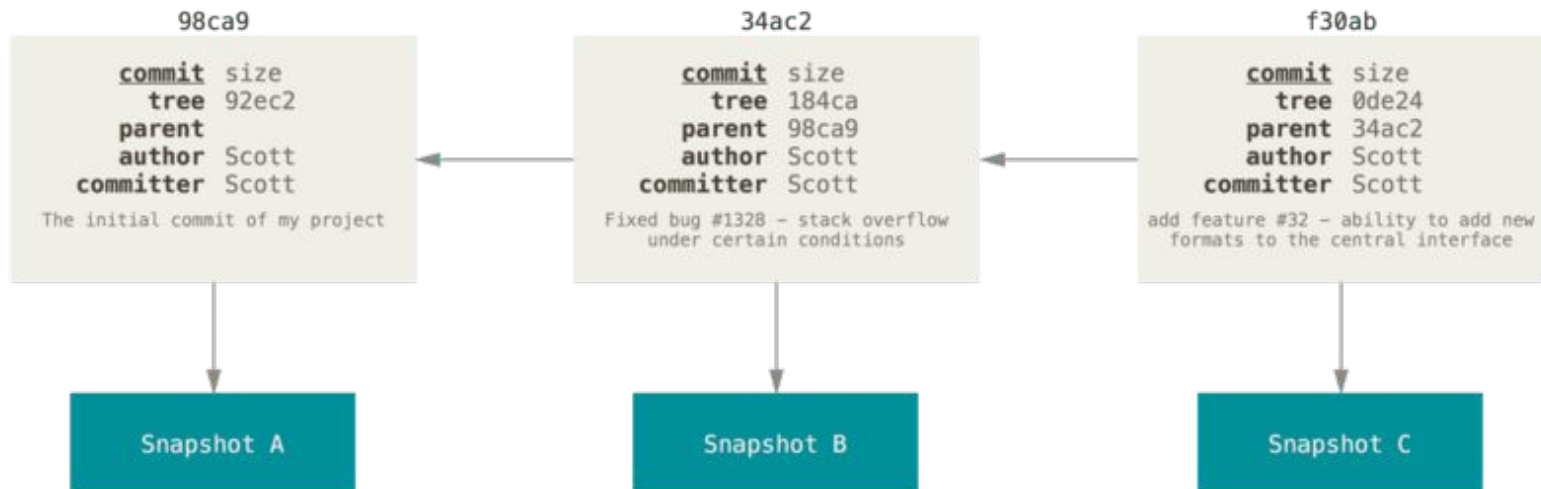
Киллер фича – ветвления

Дерево коммита

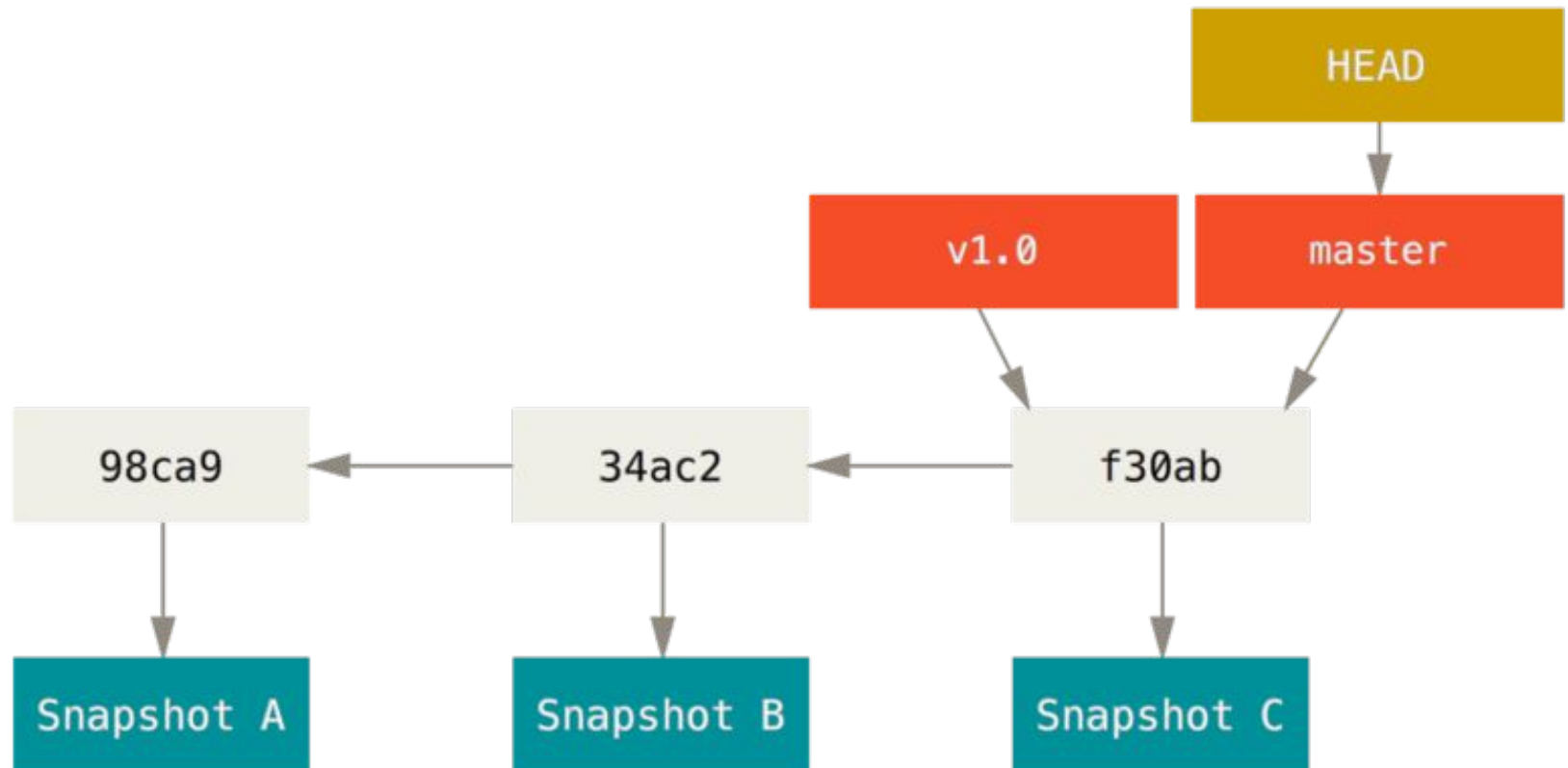
```
$ git add README test.rb LICENSE  
$ git commit -m 'The initial commit of my project'
```



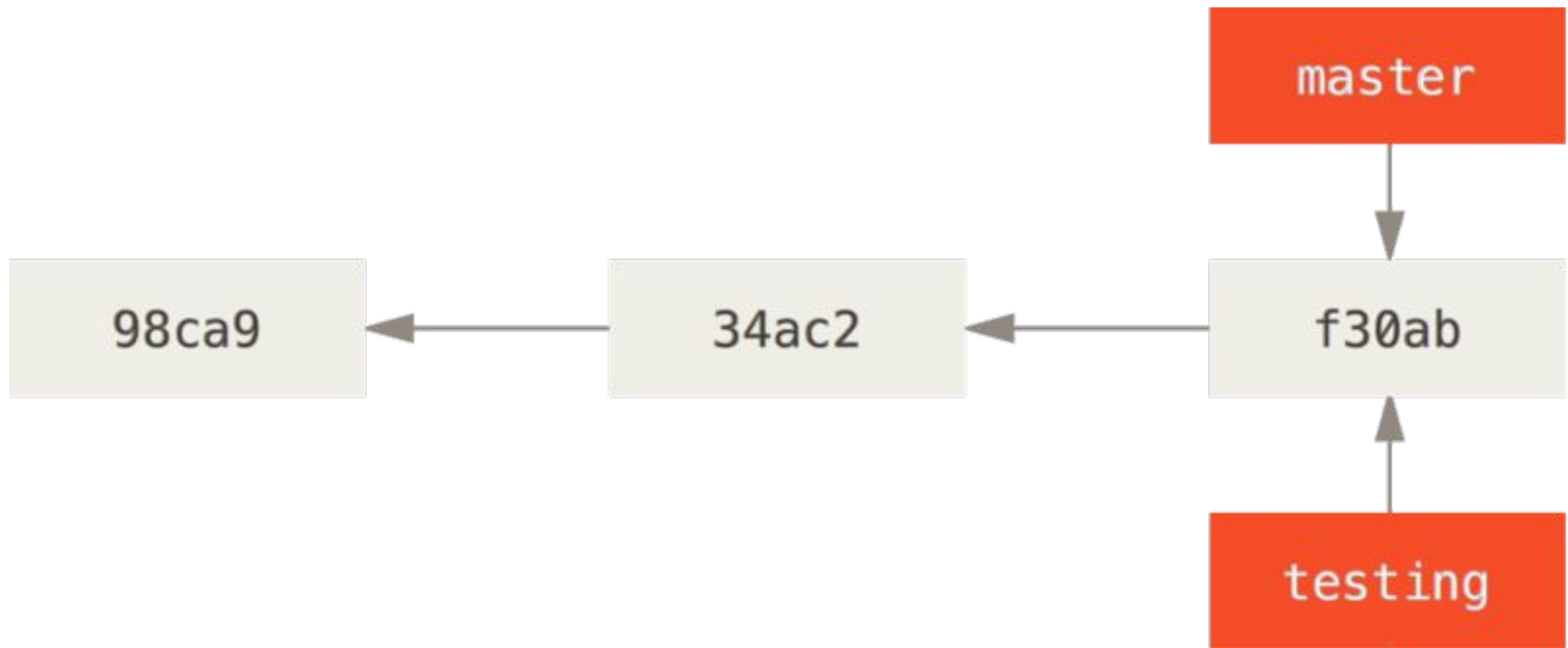
Коммит и его родители



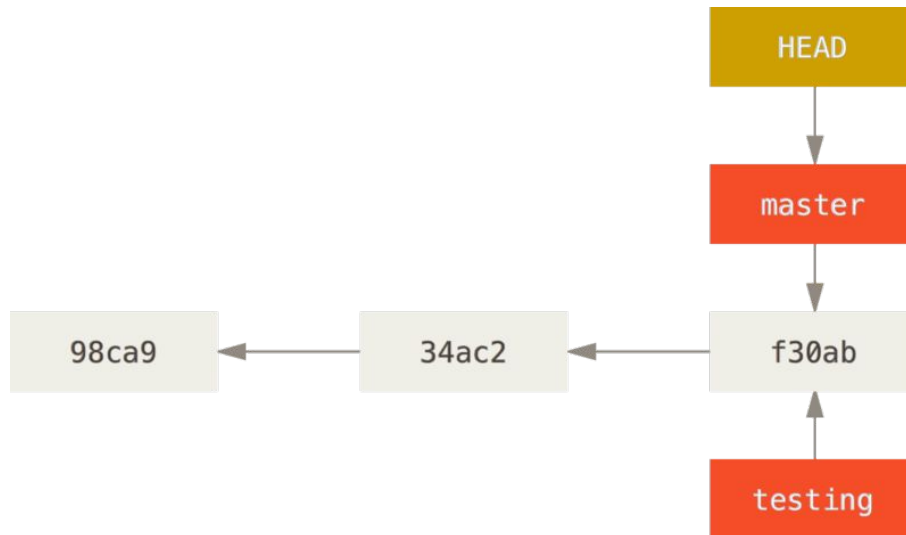
Ветка, тег и история коммитов



Две ветки указывают на одну и ту же историю



HEAD указывает на ветку мастер

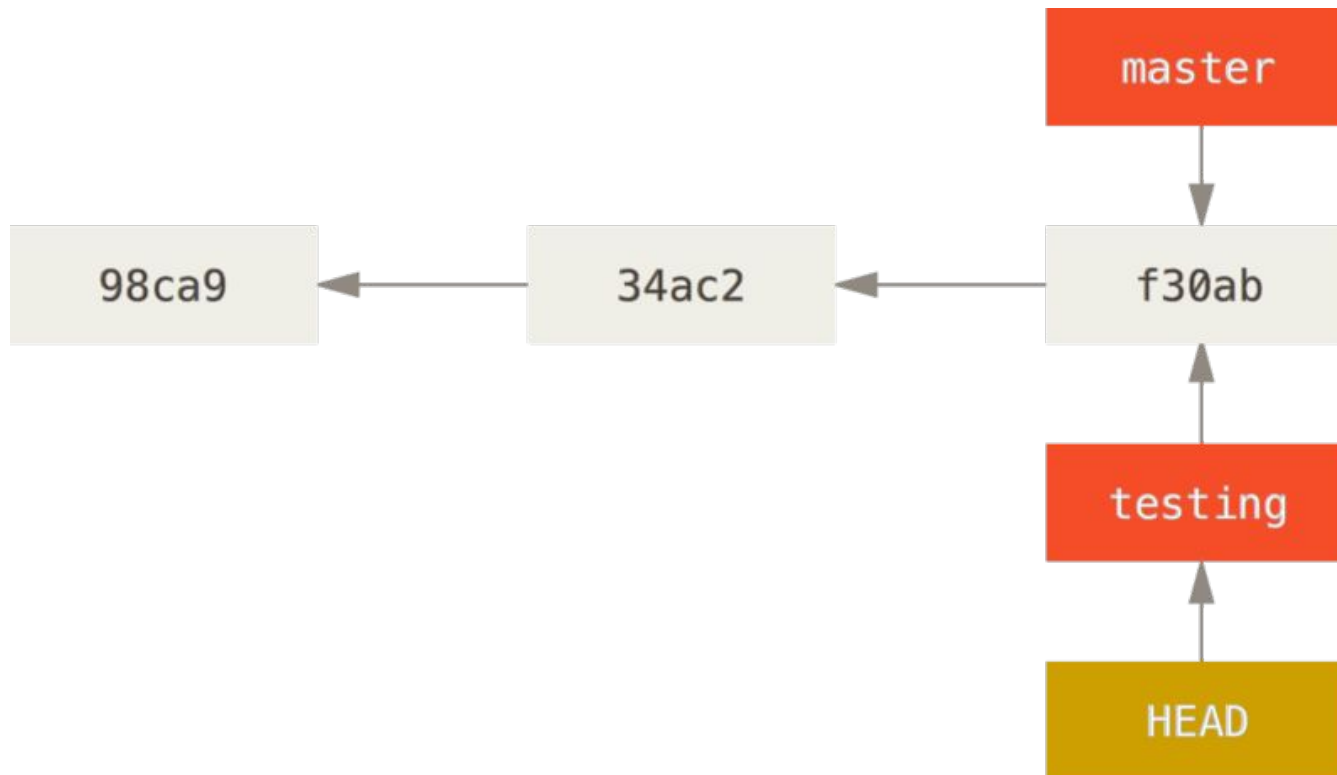


\$ git log --oneline --decorate

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

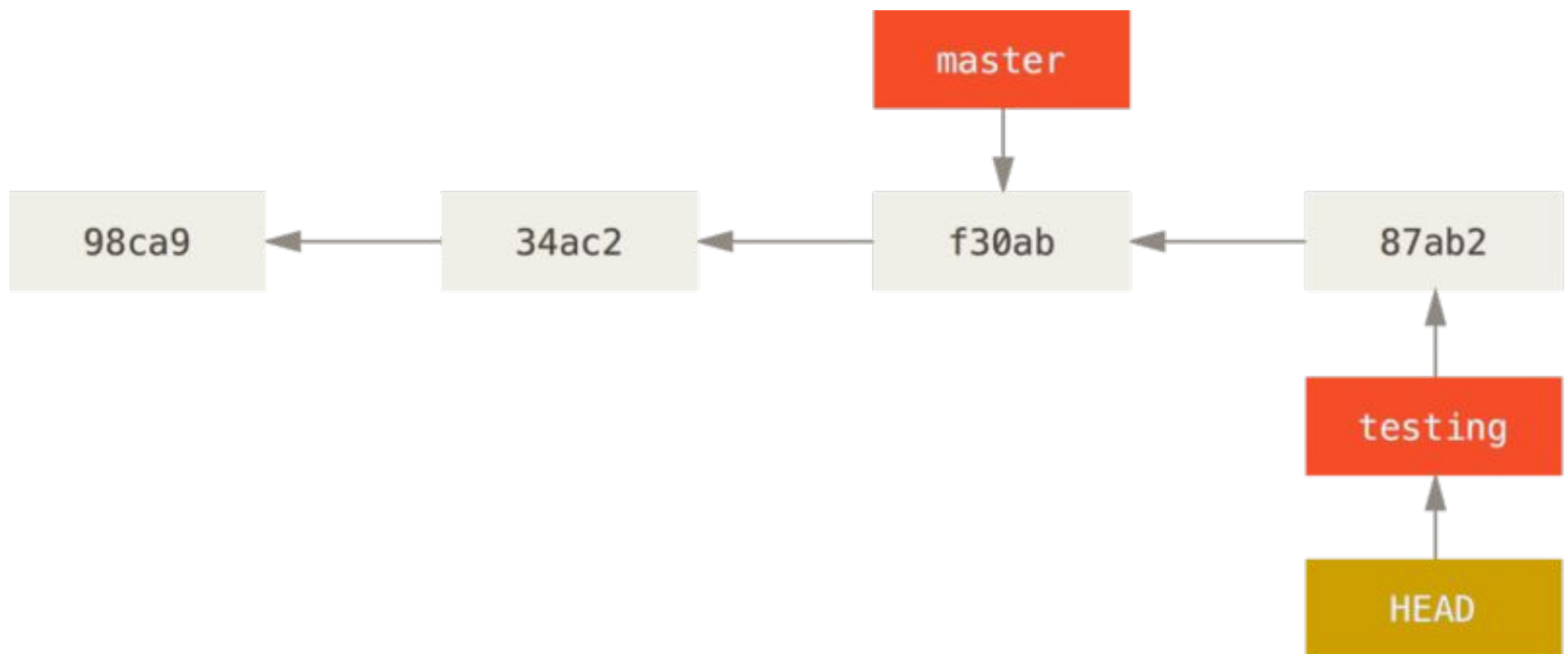
HEAD указывает на текущую ветку

```
$ git checkout testing
```



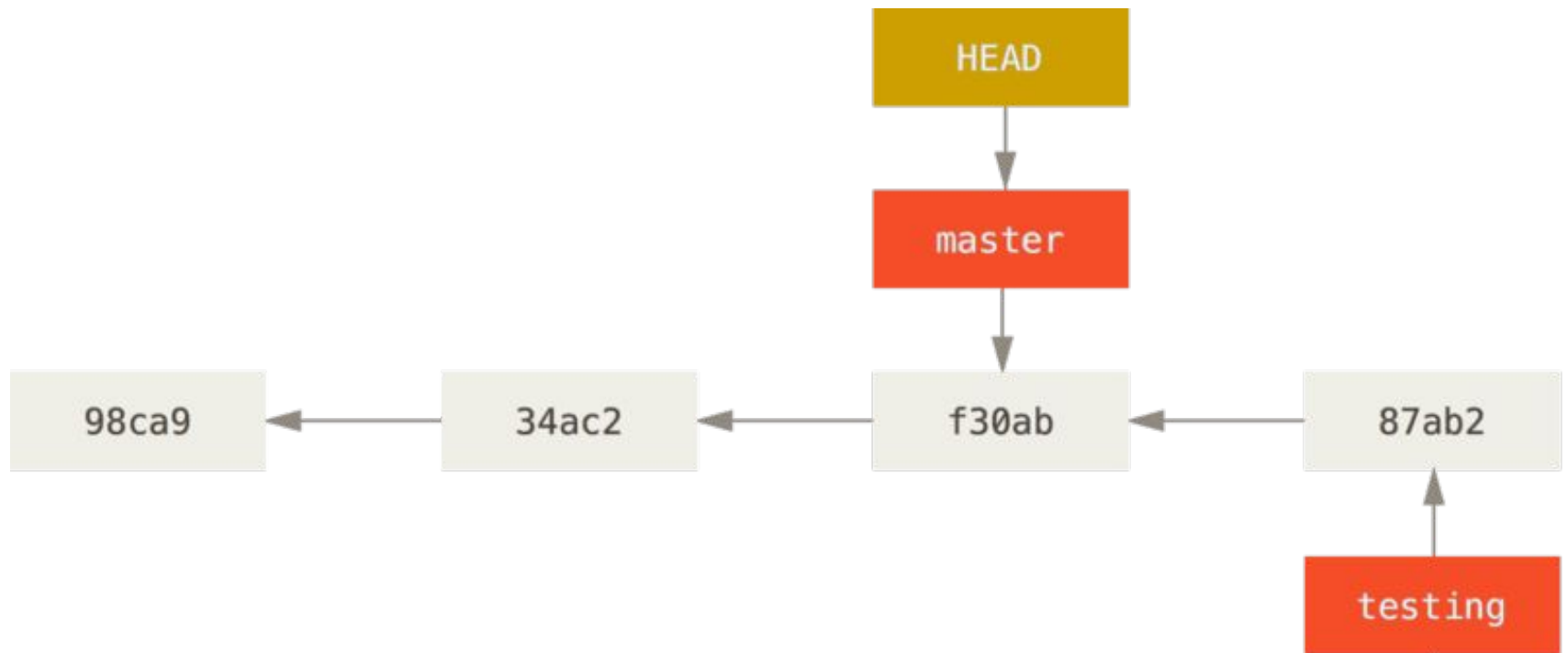
HEAD переместился вперед


```
$ echo "new line" >> test.rb  
$ git commit -a -m 'made a change'
```



HEAD перемещается во время checkout

```
$ git checkout master
```





Переключение веток меняет файлы в рабочем каталоге

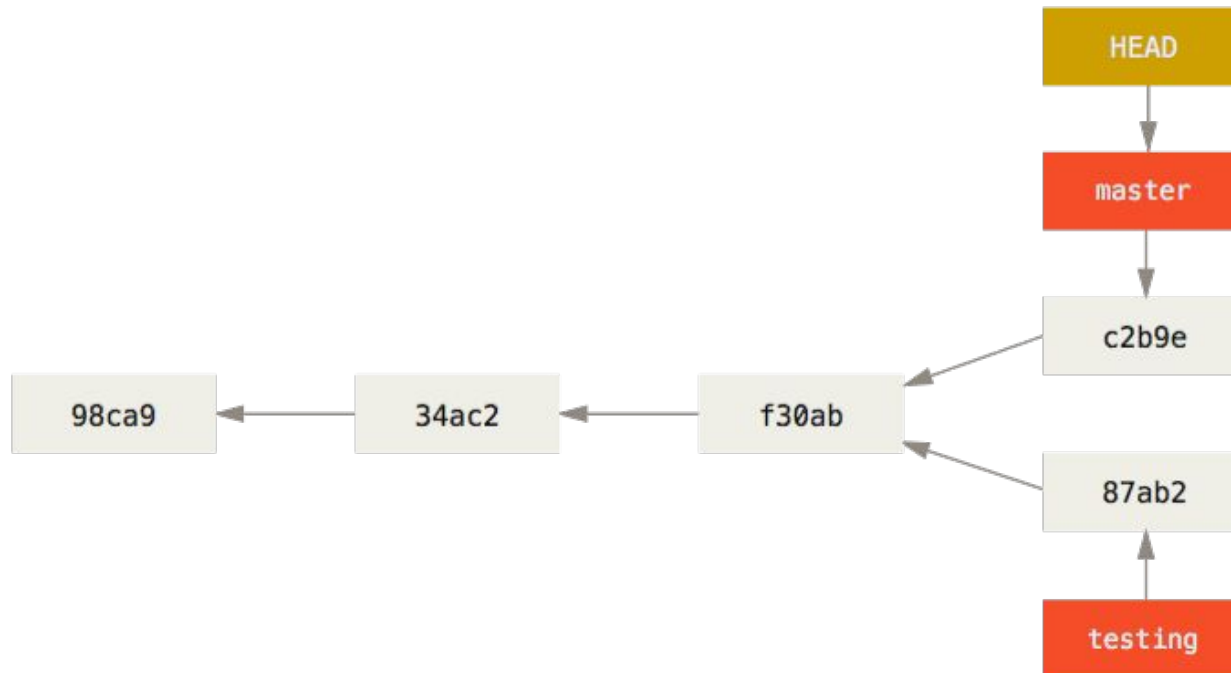
Важно запомнить, что при переключении веток в Git происходит изменение файлов в рабочей директории.

Если вы переключаетесь на старую ветку, то рабочий каталог будет выглядеть так же, как выглядел на момент последнего коммита в ту ветку.

Если Git по каким-то причинам не может этого сделать — он не позволит вам переключиться вообще.

Разветвлённая история

```
$ echo "second new line" >> test.rb  
$ git commit -a -m 'made a change in master'
```



Разветвлённая история

```
$ git log --oneline --decorate --graph --all
```

```
* c2b9e (HEAD, master) made other changes
```

```
| * 87ab2 (testing) made a change
```

```
|/
```

```
* f30ab add feature #32 - ability to add new formats to the
```

```
* 34ac2 fixed bug #1328 - stack overflow under certain conditions
```

```
* 98ca9 initial commit of my project
```



Создание ветки и переключение на нее

```
$ git checkout -b new_branch
```

или

```
$ git switch -c new_branch
```



Итоги



Чему мы научились

- Работать с историей коммитов
- Отменять изменения
- Работать с удаленными репозиториями
- Создавать и использовать теги
- Узнали как работает ветвление



Домашнее задание



Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и
пишите отзыв о лекции!**

Андрей Борю



[andreyborue](https://t.me/andreyborue)



[andreyborue](https://netology.ru/andreyborue)



[andreyborue](https://t.me/andreyborue)