

Инструменты Git



Андрей
Борю



Андрей Борю

Principal DevOps Engineer, Snapcart





План занятия

1. [Выбор ревизии](#)
2. [Интерактивное индексирование](#)
3. [Stash и clean](#)
4. [Поиск](#)
5. [Исправление истории](#)
6. [Продвинутый merge](#)
7. [Отмена слияния](#)
8. [Другие полезные инструменты](#)
9. [Итоги](#)
10. [Домашнее задание](#)



Выбор ревизии

Сокращенный SHA-1

Git понимает, какой коммит имеется в виду по нескольким первым символам его хеша, если указанная часть SHA-1 имеет в длину по крайней мере четыре символа и однозначна.

```
$ git log --pretty=oneline
```

```
6dab2a9293a146fdb64dd8405f1ea95faba3cd3f (HEAD -> master, origin/master) Git 03 homework  
87332c922e28ef59537f8ef1d3fce2c3a598ebcf Git 02 homework  
4247ce32eec2c02d07dde1e7c514554d9be475ea Git 01 homework
```

```
$ git log --abbrev-commit --pretty=oneline
```

```
6dab2a9 (HEAD -> master, origin/master) Git 03 homework  
87332c9 Git 02 homework  
4247ce3 Git 01 homework
```

```
$ git show 87332c922e28ef59537f8ef1d3fce2c3a598ebcf
```

```
$ git show 87332c922e28ef59537f
```

```
$ git show 8733
```

Ссылки на ветки

Вы можете использовать имя ветки в любой команде Git, которая ожидает коммит или значение SHA-1.

```
$ git show ca82a6dff817ec66f44342007202690a93763949
```

```
$ git show topic1
```

Что бы узнать SHA-1 объекта можно использовать команду **rev-parse**.

```
$ git rev-parse topic1
```

```
ca82a6dff817ec66f44342007202690a93763949
```

RefLog – журнал ссылок

Чтобы просмотреть свой локальный журнал ссылок, используйте команду **git reflog**.

```
$ git reflog
```

```
6dab2a9 (HEAD -> master, origin/master) HEAD@{0}: commit: Git 02 homework
```

```
87332c9 HEAD@{1}: commit: Git 02 homework
```

```
4247ce3 HEAD@{2}: commit: Git 01 homework
```

```
85088cb HEAD@{3}: commit: Intro homework
```

```
46906d6 (tag: v0.0, origin/version0.0, version0.0, version0) HEAD@{4}: checkout: moving from version0.0 to master
```

Если вы хотите увидеть какой была HEAD вашего репозитория N шагов назад, то вы можете использовать ссылку **@{n}**.

```
$ git show HEAD@{5}
```

```
$ git show master@{yesterday}
```

```
$ git log -g master
```

```
$ git show HEAD@{2.months.ago}
```

Ссылка на предков

Ещё один популярный способ указать коммит — это использовать её родословную.

```
$ git log --pretty=format:'%h %s' --graph
```

```
* 734713b fixed refs handling, added gc auto, updated tests
* d921970 Merge commit 'phedders/rdocs'
|\
| * 35cfb2b Some rdoc changes
* | 1c002dd added some blame and merge stuff
|/
* 1c36188 ignore *.gem
* 9b29157 add open3_detach to gemspec file list
```

```
$ git show HEAD^
```

```
commit d921970aadf03b3cf0e71becdaab3147ba71cdef
```

```
$ git show d921970^
```

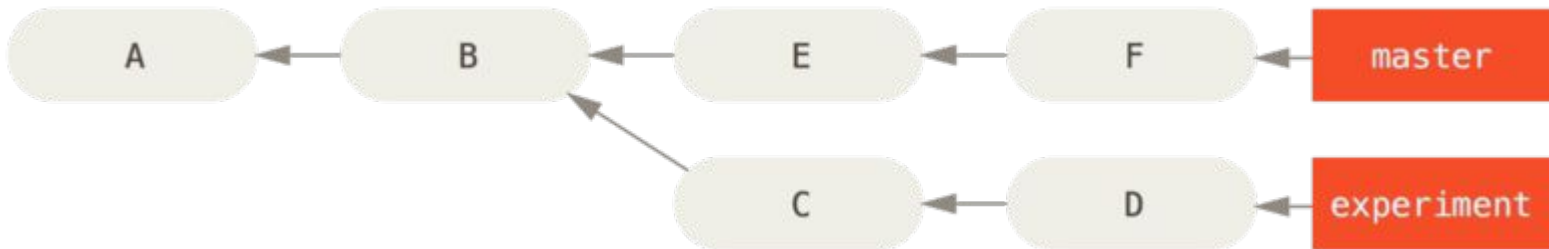
```
$ git show d921970^2
```

```
$ git show HEAD~3
```

```
$ git show HEAD^^^
```

```
$ git show HEAD~3^2
```


Диапазон коммитов: две точки



```
$ git log master..experiment
```

```
D
```

```
C
```

```
$ git log experiment..master
```

```
F
```

```
E
```

```
$ git log origin/master..HEAD
```

```
$ git log origin/master..
```

Множество точек

Исключение коммитов при помощи **--not** или **^**.
Эквивалентные команды:

```
$ git log refA..refB  
$ git log ^refA refB  
$ git log refB --not refA
```

Все коммиты, доступные из **refA** и **refB**, но не доступные из **refC**:

```
$ git log refA refB ^refC  
$ git log refA refB --not refC
```

Три точки

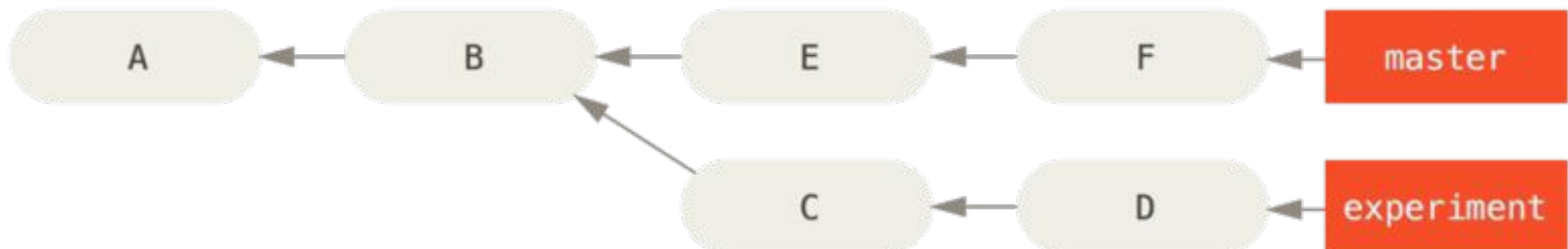
Какие коммиты есть либо в ветке **master**, либо в **experiment**, но не в обеих сразу:


```
$ git log master...experiment
```

```
F  
E  
D  
C
```

```
$ git log --left-right master...experiment
```

```
< F  
< E  
> D  
> C
```





Интерактивное индексирование

Интерактивное индексирование

Интерактивное индексирование полезно, если вы изменили множество файлов, а затем решили, что хотите чтобы эти изменения были в нескольких маленьких понятных коммитах, а не в одном большом и запутанном.

<https://git-scm.com/book/ru/v2/Инструменты-Git-Интерактивное-индексирование>

```
$ git add -i
```

```
    staged  unstaged path
```

```
1:  unchanged    +0/-1 TODO
2:  unchanged    +1/-1 index.html
3:  unchanged    +5/-1 lib/simplegit.rb
```

```
*** Commands ***
```

```
1: status  2: update  3: revert  4: add untracked
5: patch   6: diff    7: quit   8: help
```

```
What now>
```



Stash и clean (прибережение и очистка)

Прячем наработки

\$ git status

Changes to be committed:

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

modified: index.html

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: lib/simplegit.rb

\$ git stash

Saved working directory and index state \

"WIP on master: 049d078 added the index file"

HEAD is now at 049d078 added the index file

(To restore them type "git stash apply")

\$ git status

On branch master

nothing to commit, working directory clean

Восстановление изменений

\$ git stash list

```
stash@{0}: WIP on master: 049d078 added the index file  
stash@{1}: WIP on master: c264051 Revert "added file_size"  
stash@{2}: WIP on master: 21d80a5 added number to log
```

\$ git stash apply

\$ git stash apply stash@{2}

\$ git stash apply

```
# On branch master  
# Changed but not updated:  
#   (use "git add <file>..." to update what will be committed)  
#  
#   modified:   index.html  
#   modified:   lib/simplegit.rb  
#
```


Восстановление изменений

\$ git stash apply --index

On branch master

Changes to be committed:

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

modified: index.html

Changed but not updated:

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

modified: lib/simplegit.rb#

\$ git stash list

stash@{0}: WIP on master: 049d078 added the index file

stash@{1}: WIP on master: c264051 Revert "added file_size"

stash@{2}: WIP on master: 21d80a5 added number to log

\$ git stash drop stash@{0}

Dropped stash@{0} (364e91f3f268f0900bc3ee613f9f733e82aaed43)

\$ git stash pop

Дополнительные параметры

Не прятать то, что вы уже добавили в индекс командой **git add**:

```
$ git stash --keep-index
```

Также спрятать все неотслеживаемые файлы:

```
$ git stash --include-untracked
```

Запустить интерактивный режим:

```
$ git stash --patch
```

Создание изменений из спрятанных изменений:

```
$ git stash branch testchanges
```

Очистка рабочей директории

Если вы хотите не прятать некоторые из изменений или файлов в вашей рабочей директории, а просто избавиться от них. Команда **git clean** сделает это для вас.

```
$ git clean
```

Основные аргументы:

-n, --dry-run	Ничего не удалять, только показать что будет удалено.
-f, --force	Действительно удалить, а не только показать.
-d	Удалять директории.
-q, --quit	Тихий режим
-x	Удалить даже то, что в .gitignore
-i, --interactive	Интерактивный режим



Поиск

Git grep

Git поставляется с командой **grep**, которая позволяет легко искать в истории коммитов или в рабочем каталоге по строке или регулярному выражению.

-n, --line-number	Отобразить номера строк.
-c, --count	Покажет количество совпадений в файлах.
-p, --show-function	Показать метод или функцию, в котором присутствует совпадение.
--and, --or, --not	Логические операторы.
--break	Добавить пустые строки между результатами.
--heading	Отобразить имена файлов надо группой найденных строк.

И многое другое - **\$ git grep --help**

Поиск в журнале изменений

Команда **git log** обладает мощными инструментами для поиска определённых коммитов по содержимому их сообщений или содержимому сделанных в них изменений.

Например, если вы хотите найти, когда была добавлена константа **ZLIB_BUF_MAX**, то вы можете с помощью опции **-S** попросить Git показывать только те коммиты, в которых была добавлена или удалена эта строка.

```
$ git log -SZLIB_BUF_MAX --oneline
```

```
e01503b zlib: allow feeding more than 4GB in one go
```

```
ef49a7a zlib: zlib can only process 4GB at a time
```

Поиск строки в истории

```
$ git log -L :git_deflate_bound:zlib.c
```

```
commit ef49a7a0126d64359c974b4b3b71d7ad42ee3bca
```

```
Author: Junio C Hamano <gitster@pobox.com>
```

```
Date: Fri Jun 10 11:52:15 2011 -0700
```

```
    zlib: zlib can only process 4GB at a time
```

```
diff --git a/zlib.c b/zlib.c
```

```
--- a/zlib.c
```

```
+++ b/zlib.c
```

```
@@ -85,5 +130,5 @@
```

```
-unsigned long git_deflate_bound(z_stream strm, unsigned long size)
```

```
+unsigned long git_deflate_bound(git_zstream *strm, unsigned long size)
```

```
{
```

```
-    return deflateBound(strm, size);
```

```
+    return deflateBound(&strm->z, size);
```

```
}
```

```
commit 225a6f1068f71723a910e8565db4e252b3ca21fa
```

```
...
```

```
$ git log -L '/unsigned long git_deflate_bound/',/^}/:zlib.c
```



Исправление истории



Изменение последнего коммита

Чтобы изменить сообщение вашего последнего коммита используйте amend:

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

Изменение сообщений нескольких коммитов

```
$ git rebase -i HEAD~3
```

```
pick f7f3f6d changed my name a bit
```

```
pick 310154e updated README formatting and added blame
```

```
pick a5f4a0d added cat-file (это самый последний коммит)
```

```
# Rebase 710f0f8..a5f4a0d onto 710f0f8
```

```
# Commands:
```

```
# p, pick = use commit
```

```
# r, reword = use commit, but edit the commit message
```

```
# e, edit = use commit, but stop for amending
```

```
# s, squash = use commit, but meld into previous commit
```

```
# f, fixup = like "squash", but discard this commit's log message
```

```
# x, exec = run command (the rest of the line) using shell
```

```
#
```

```
# These lines can be re-ordered; they are executed from top to bottom.
```

```
# If you remove a line here THAT COMMIT WILL BE LOST.
```

```
# However, if you remove everything, the rebase will be aborted.
```

```
# Note that empty commits are commented out
```

Изменение сообщений нескольких коммитов

```
edit f7f3f6d changed my name a bit  
pick 310154e updated README formatting and added blame  
pick a5f4a0d added cat-file
```

```
$ git rebase -i HEAD~3
```

Stopped at f7f3f6d... changed my name a bit

You can amend the commit now, with

```
git commit --amend
```

Once you're satisfied with your changes, run

```
git rebase --continue
```

Переупорядочивание коммитов

Вы также можете использовать интерактивное перебазирование для переупорядочивания или полного удаления коммитов. Если вы хотите удалить коммит и изменить порядок, в котором были внесены два оставшихся, то измените скрипт перебазирования с такого:

```
pick f7f3f6d changed my name a bit  
pick 310154e updated README formatting and added blame  
pick a5f4a0d added cat-file
```

На такой:

```
pick 310154e updated README formatting and added blame  
pick f7f3f6d changed my name a bit
```

Объединение коммитов

Если вместо “**pick**” или “**edit**” вы укажете “**squash**”, Git применит изменения из текущего и предыдущего коммитов и предложит вам объединить их сообщения:

```
pick f7f3f6d changed my name a bit
squash 310154e updated README formatting and added blame
squash a5f4a0d added cat-file
```

Когда вы сохраните скрипт и выйдете из редактора, Git применит изменения всех трёх коммитов и запросит сообщение для объединенного коммита:

```
# This is a combination of 3 commits.
# The first commit's message is:
changed my name a bit
# This is the 2nd commit message:
updated README formatting and added blame
# This is the 3rd commit message:
added cat-file
```


Разбиение коммитов

```
pick f7f3f6d changed my name a bit
edit 310154e updated README formatting and added blame
pick a5f4a0d added cat-file
```

```
$ git reset HEAD^
$ git add README
$ git commit -m 'updated README formatting'
$ git add lib/simplegit.rb
$ git commit -m 'added blame'
$ git rebase --continue
```

```
$ git log -4 --pretty=format:"%h %s"
```

```
1c002dd added cat-file
9b29157 added blame
35cfb2b updated README formatting
f3cc40e changed my name a bit
```



Продвинутый инструмент: filter-branch

Удаление файла из каждого коммита

```
$ git filter-branch --tree-filter 'rm -f passwords.txt' HEAD
```

```
Rewrite 6b9b3cf04e7c5686a9cb838c3f36a8cb6a0fc2bd (21/21)  
Ref 'refs/heads/master' was rewritten
```

Опция **--tree-filter** выполняет указанную команду после переключения на каждый коммит и затем повторно фиксирует результаты.

Изменение корневой директорией проекта

Предположим, вы выполнили импорт из другой системы управления исходным кодом и получили в результате поддиректории, которые не имеют никакого смысла (**trunk**, **tags** и так далее). Если вы хотите сделать поддиректорию **trunk** корневой директорией для каждого коммита, команда **filter-branch** может помочь вам в этом:

```
$ git filter-branch --subdirectory-filter trunk HEAD
```

```
Rewrite 856f0bf61e41a27326cdae8f09fe708d679f596f (12/12)  
Ref 'refs/heads/master' was rewritten
```

Глобальное изменения email'a

Если вы забыли выполнить **git config** для настройки своего имени и email перед началом работы или хотите открыть исходные коды вашего рабочего проекта и везде изменить email, вы можете его изменить сразу в нескольких коммитах с помощью команды **filter-branch**. Используйте опцию **--commit-filter**:

```
$ git filter-branch --commit-filter '  
    if [ "$GIT_AUTHOR_EMAIL" = "schacon@localhost" ];  
    then  
        GIT_AUTHOR_NAME="Scott Chacon";  
        GIT_AUTHOR_EMAIL="schacon@example.com";  
        git commit-tree "$@";  
    else  
        git commit-tree "$@";  
    fi' HEAD
```



Продвинутый merge

Прерывание слияния

Чтобы отменить попытку слияния, используйте **git merge --abort**.

```
$ git merge mbranch
```

```
Auto-merging hello.rb
```

```
CONFLICT (content): Merge conflict in hello.rb
```

```
Automatic merge failed; fix conflicts and then commit the result.
```

```
$ git status -sb
```

```
## master
```

```
UU hello.rb
```

```
$ git merge --abort
```

```
$ git status -sb
```

```
## master
```

Если по каким-то причинам вы обнаружили себя в ужасном состоянии и хотите просто начать всё сначала, вы можете также выполнить **git reset --hard HEAD**

```
$ git reset --hard HEAD
```

Игнорирование пробельных символов

Если множество конфликтов слияния вызваны пробельными символами, то можно прервать слияние и запустить его снова, но на этот раз с опцией **-Xignore-all-space** или **-Xignore-space-change**.

\$ git merge -Xignore-all-space whitespace

Auto-merging hello.rb

Merge made by the 'recursive' strategy.

hello.rb | 2 +-
+ 1 - 1

1 file changed, 1 insertion(+), 1 deletion(-)

Ручное слияние

Во время состояния конфликта слияния можно получить три версии файла для дальнейшего их объединения:

```
$ git show :1:hello.rb > hello.common.rb  
$ git show :2:hello.rb > hello.ours.rb  
$ git show :3:hello.rb > hello.theirs.rb
```

Если вы хотите что-то более суровое, то можете также воспользоваться служебной командой **ls-files -u** для получения SHA-1 хешей для каждого из этих файлов.

```
$ git ls-files -u  
100755 ac51efdc3df4f4fd328d1a02ad05331d8e2c9111    1    hello.rb  
100755 36c06c8752c78d2aff89571132f3bf7841a7b5c3    2    hello.rb  
100755 e85207e04dfdd5eb0a1e9febbc67fd837c44a1cd    3    hello.rb
```

Ручное слияние

Когда в нашей рабочей директории присутствует содержимое всех трёх состояний, мы можем вручную исправить их, чтобы устранить проблемы с пробельными символами и повторно выполнить слияние с помощью команды **git merge-file**

```
$ dos2unix hello.theirs.rb
dos2unix: converting file hello.theirs.rb to Unix format ...

$ git merge-file -p \
  hello.ours.rb hello.common.rb hello.theirs.rb > hello.rb
```

Если перед коммитом можно посмотреть различия между состояниями при помощи **git diff**.

```
$ git diff --ours
$ git diff --theirs -w
```

git clean поможет удалить ненужные более дополнительные файлы.

```
$ git clean -f
Removing hello.common.rb
Removing hello.ours.rb
Removing hello.theirs.rb
```

Checkout в конфликтах

В качестве значения опции **--conflict** вы можете указывать **diff3** или **merge**.

```
$ git checkout --conflict=merge hello.rb  
$ git checkout --conflict=diff3 hello.rb
```

В случае **diff3** будет больше информации:.

```
def hello  
  <<<<<< ours  
    puts 'hola world'  
  ||||| base  
    puts 'hello world'  
  =====  
    puts 'hello mundo'  
  >>>>>> theirs  
end  
  
hello()
```


История при слиянии

Полезный инструмент — команда **git log**. Она поможет вам получить информацию о том, что могло привести к возникновению конфликтов.

```
$ git log --oneline --left-right HEAD...MERGE_HEAD
```

```
< f1270f7 update README  
< 9af9d3b add a README  
< 694971d update phrase to hola world  
> e3eb223 add more tests  
> 7cff591 add testing script  
> c3ffff1 changed text to hello mundo
```

Если мы добавим опцию **--merge** к команде **git log**, она покажет нам коммиты, в которых изменялся конфликтующий в данный момент файл.

```
$ git log --oneline --left-right --merge
```

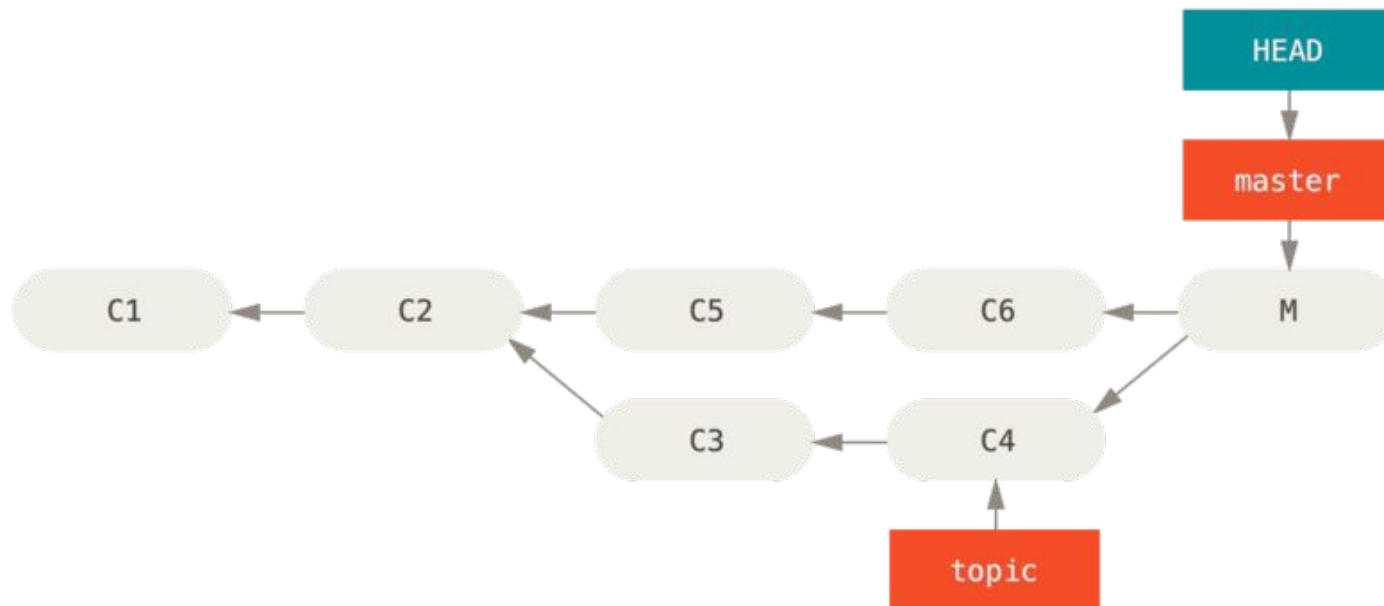
```
< 694971d update phrase to hola world  
> c3ffff1 changed text to hello mundo
```



Отмена слияния

Случайный merge

Допустим, вы начали работать в тематической ветке, случайно слили ее в **master**, и теперь ваша история коммитов выглядит следующим образом.

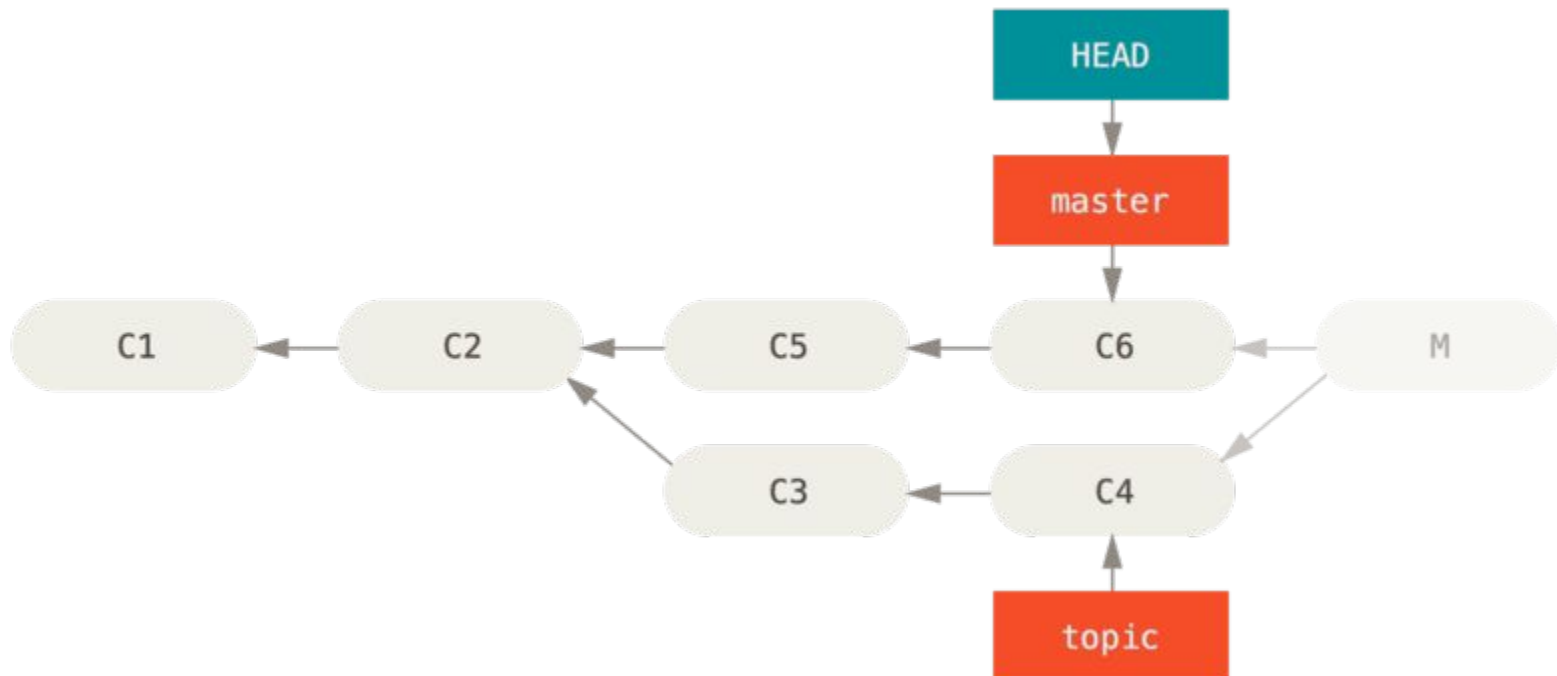


Есть два подхода к решению этой проблемы, в зависимости от того, какой результат вы хотите получить.

Исправление ссылок

Если нежелаемый коммит слияния существует только в вашем локальном репозитории, то простейшее и лучшее решение состоит в перемещении веток так, чтобы они указывали туда куда вам нужно.

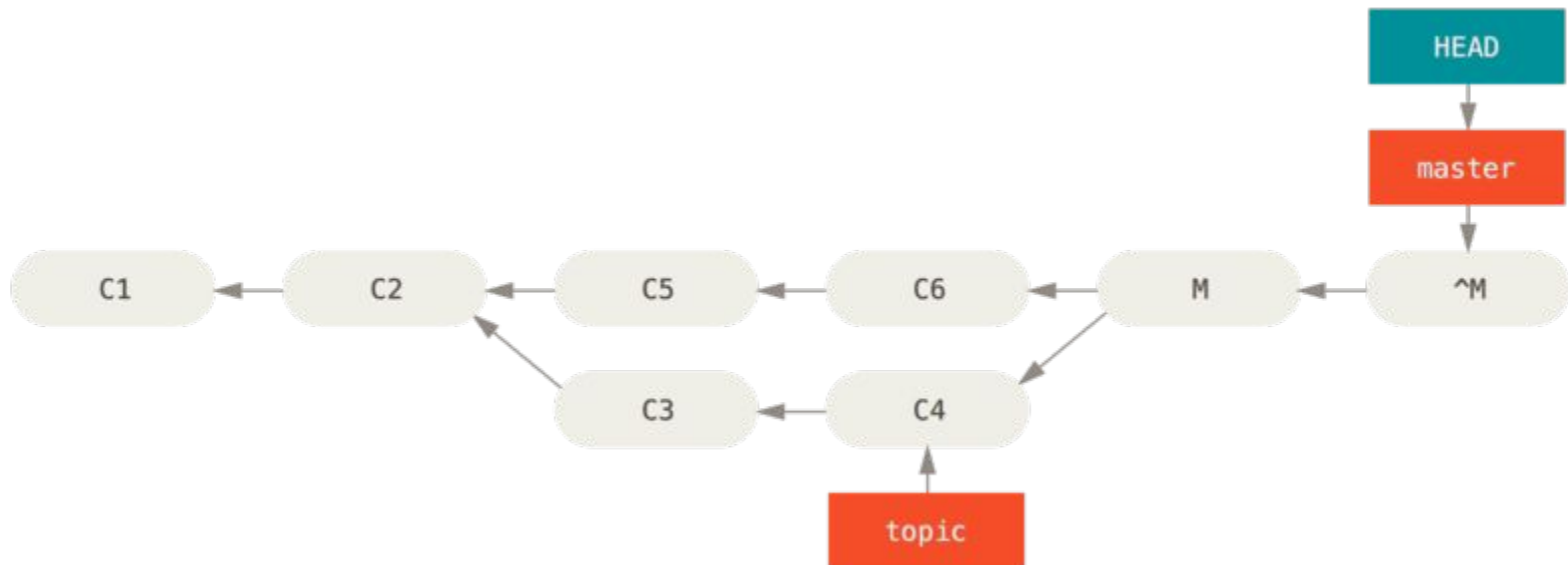
```
$ git reset --hard HEAD~
```



Отмена коммита

```
$ git revert -m 1 HEAD
```

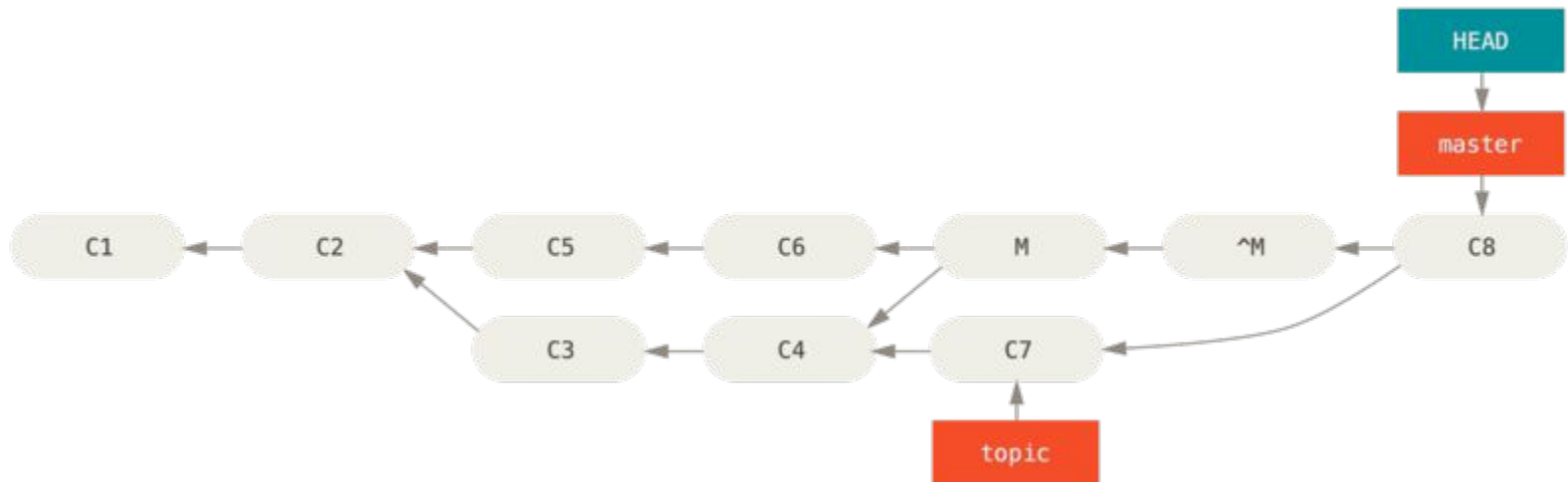
```
[master b1d8379] Revert "Merge branch 'topic'"
```



История с плохим слиянием

\$ git merge topic

Already up-to-date.

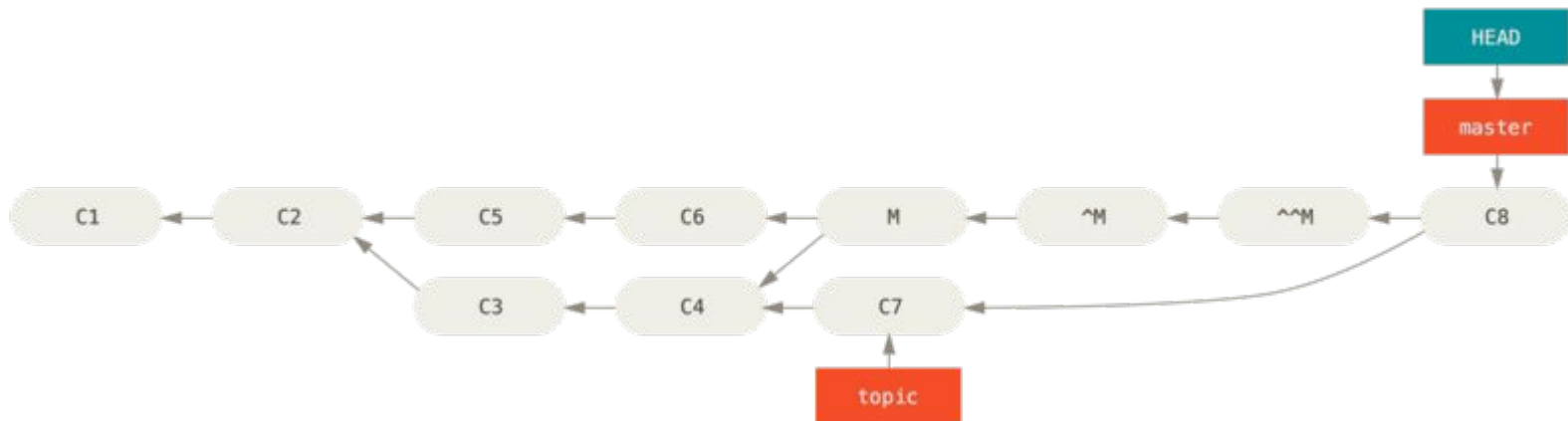



История после отмененного коммита слияния

```
$ git revert ^M
```

```
[master 09f0126] Revert "Revert "Merge branch 'topic'""
```

```
$ git merge topic
```





Другие полезные инструменты

Rerere

Функциональность **git rerere** — частично скрытый компонент Git. Её имя является сокращением для “reuse recorded resolution” (“повторное использование сохранённых разрешений конфликтов”). Как следует из имени, эта функциональность позволяет попросить Git запомнить то, как вы разрешили некоторую часть конфликта, так что в случае возникновения такого же конфликта, Git сможет его разрешить автоматически.

Включается этот функционал так:

```
$ git config --global rerere.enabled true
```

Подробнее:

<https://git-scm.com/book/ru/v2/Инструменты-Git-Rerere>

Blame

Если вы обнаружили ошибку в вашем коде и хотите знать, когда она была добавлена и почему, то в большинстве случаев аннотация файла будет лучшим инструментом для этого.

```
$ git blame -L 12,16 simplegit.rb
^4832fe2 (Scott Chacon 2008-03-15 10:31:28 -0700 12) def show(tree = 'master')
^4832fe2 (Scott Chacon 2008-03-15 10:31:28 -0700 13)   command("git show #{tree}")
^4832fe2 (Scott Chacon 2008-03-15 10:31:28 -0700 14) end
^4832fe2 (Scott Chacon 2008-03-15 10:31:28 -0700 15)
9f6560e4 (Scott Chacon 2008-03-17 21:52:20 -0700 16) def log(tree = 'master')
```

С опцией **-C** для отслеживания первоисточников фрагментов файла:

```
$ git blame -C -L 141,143 GITPackUpload.m
f344f58d GITServerHandler.m (Scott 2009-01-04 143) {
70befddd GITServerHandler.m (Scott 2009-03-22 144)   //NSLog(@"GATHER COMMI
ad11ac80 GITPackUpload.m   (Scott 2009-03-24 145)
```

Бинарный поиск

Аннотирование файла помогает, если вы знаете, где находится проблема и можете начать исследование с этого места. Если вы не знаете, что сломано, а с тех пор как код работал, были сделаны десятки или сотни коммитов, вы вероятно воспользуетесь командой **git bisect**. Эта команда выполняет бинарный поиск по истории коммитов для того, чтобы помочь вам как можно быстрее определить коммит, который создал проблему.

```
$ git bisect start
```

```
$ git bisect bad
```

```
$ git bisect good v1.0
```

```
Bisecting: 6 revisions left to test after this
```

```
[ecb6e1bc347ccec5f9350d878ce677feb13d3b2] error handling on repo
```

```
▪
```

Бинарный поиск: поиск хорошего коммита

Git выяснил, что произошло около 12 коммитов между коммитом, который вы отметили как последний хороший коммит (v1.0), и текущим плохим коммитом, и выгрузил вам один из середины.

Пусть в данном коммите проблема не проявляется, вы сообщаете об этом Git с помощью `git bisect good` и продолжаете ваше путешествие:

```
$ git bisect good
```

```
Bisecting: 3 revisions left to test after this
```

```
[b047b02ea83310a70fd603dc8cd7a6cd13d15c04] secure this thing
```

```
$ git bisect bad
```

```
Bisecting: 1 revisions left to test after this
```

```
[f71ce38690acf49c1f3c9bea38e09d82a5ce6014] drop exceptions table
```

Бинарный поиск: хороший коммит найден

Этот коммит хороший и теперь Git имеет всю необходимую информацию для определения того, где была внесена ошибка. Он сообщает SHA-1 первого плохого коммита и отображает некоторую информацию о коммите и файлах, которые были изменены в этом коммите, так, чтобы проще было разобраться что же случилось.

\$ git bisect good

```
b047b02ea83310a70fd603dc8cd7a6cd13d15c04 is first bad commit  
commit b047b02ea83310a70fd603dc8cd7a6cd13d15c04
```

```
Author: PJ Hyett <pjhyett@example.com>
```

```
Date: Tue Jan 27 14:48:32 2009 -0800
```

```
secure this thing
```

```
:040000 040000 40ee3e7821b895e52c1695092db9bdc4c61d1730
```

```
f24d3c6ebcfc639b1a3814550e62d60b8e68a8e4 M config
```

\$ git bisect reset

Бинарный поиск: автоматизация

В действительности, если у вас есть скрипт, который будет возвращать 0 если проект находится в рабочем состоянии и любое другое число в обратном случае, то вы можете полностью автоматизировать **git bisect**. Сперва, вы снова сообщаете границы бинарного поиска, указывая известные плохие и хорошие коммиты. Вы можете сделать это, передавая их команде **bisect start** — первым аргументом известный плохой коммит, а вторым известный хороший коммит.

```
$ git bisect start HEAD v1.0
```

```
$ git bisect run test-error.sh
```

Подмодули

Часто при работе над одним проектом, возникает необходимость использовать в нем другой проект. Возможно, это библиотека, разрабатываемая сторонними разработчиками или вами, но в рамках отдельного проекта, и используемая в нескольких других проектах. Типичная проблема, возникающая при этом — вы хотите продолжать работать с двумя проектами по отдельности, но при этом использовать один из них в другом.

```
$ git submodule add https://github.com/chaconinc/DbConnector
```

Подробнее:

<https://git-scm.com/book/ru/v2/Инструменты-Git-Подмодули>

Чему мы научились

- Как работать с полными и сокращенными хэшами ревизий.
- Работе с диапазоном коммитов.
- Как “прятать” изменения локально и удалять все лишние файлы.
- Поиску виновных в ошибках, поиск текста в коде и истории коммитов.
- Как отменять коммиты и исправлять историю.



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)