

Источник:

01. Git — инструмент для совместной работы с нуля и до регламента в команде — Александр Васильев

<https://www.youtube.com/watch?v=XfpNNPo5ypk>

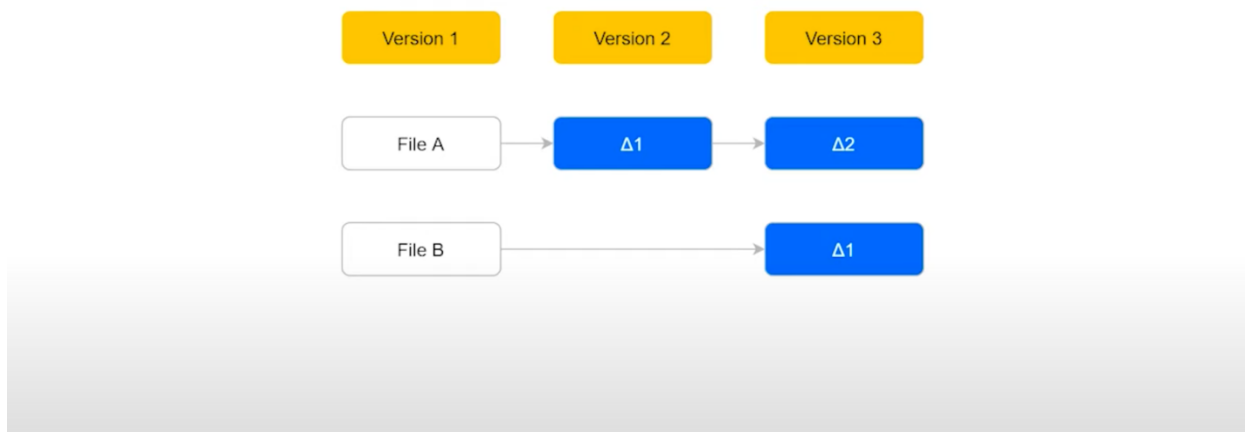
Сегодня повествование пойдет про систему Git – систему, без которой сложно представить мир современной разработки. Итак, что же это такое и зачем он нужен.

Система контроля версий



Мы можем, конечно, хранить наши файлы в отдельных папочках, именовать их версия 1, версия 2, но это не дает нам какой-то гибкости. А системы, наоборот, дают. Мы можем отслеживать изменения, смотреть кто, когда и какие изменения вносил, а также вести параллельную разработку.

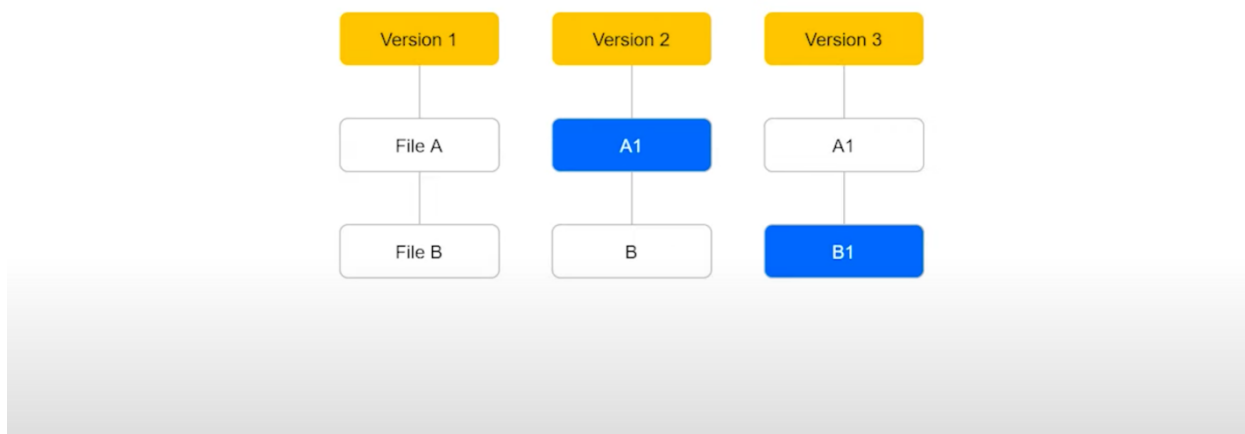
Другие системы хранят диффы файлов



Системы у нас бывают двух видов: первые хранят диффы файлов, то есть только то, что изменилось, и для каждого нового файла создается такая цепочка.

Например, в Версии 2 у нас изменяется файл и для него записывается его дельта. И для File B ничего не записывается.

git хранит файлы целиком



Git же отличается от этого: он хранит файл целиком и если меняется File A, то создается новый файл, а Версия 2 ссылается на него. Он ссылается по хэшам, на это мы обязательно посмотрим дальше. А File B не изменился и в Версии 2 записана ссылка на его старую версию.

Что внутри?

Что же у нас внутри?

Blob

Blob e92bc..

Content of readme.md

В объекте типа **blob** содержится длина файла и его содержимое

Внутри Git состоит из blob-ов – это бинарные объекты, в них содержится содержимое наших файлов и их длина.

Tree

Tree 78tf4..

blob: e92bc readme.md
tree: 69g78 lib

В объекте типа **tree** хранится список записей, который соответствует иерархии файловой системы

Деревья – это файловая структура нашего проекта. В них содержится список blob-ов, то есть ссылки на эти blob-ы и список дочерних деревьев, то есть вложенных папок.

Commit

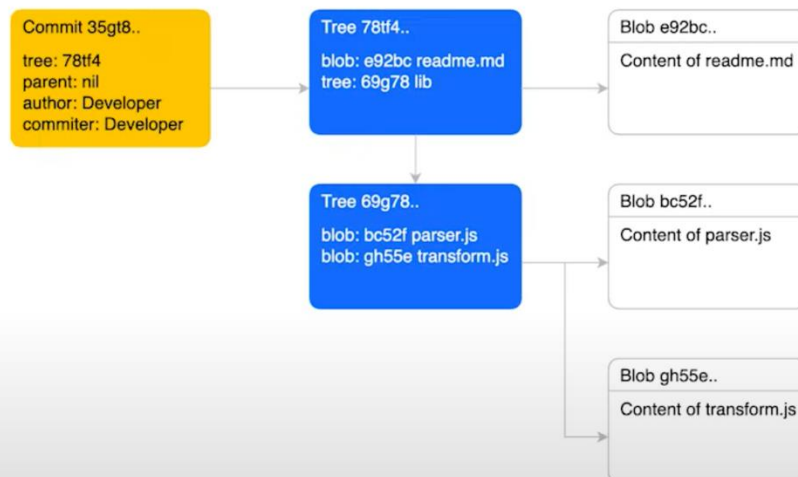
Commit 35gt8..

tree: 78tf4
parent: nil
author: Developer
committer: Developer

В объекте типа **commit** хранится ссылка на объект tree и ссылку на родительский коммит

И кирпичики наших хранилищ это commit-ы. В коммитах уже больше информации, там записываются ссылки на деревья, на корневое дерево, ссылка на родительский коммит, автор и коммитер, а также дата.

Blob, tree, commit



И вот так у нас выглядит структура нашего коммита в конечном итоге. Коммит ссылается на корневое дерево, а дерево в свою очередь ссылается на другие деревья и блобы. Если посмотри на эту схему, то тут видно, что у нас есть корневая папка, в которой лежит файл readme и одна подпапка, в которой 2 js файла.

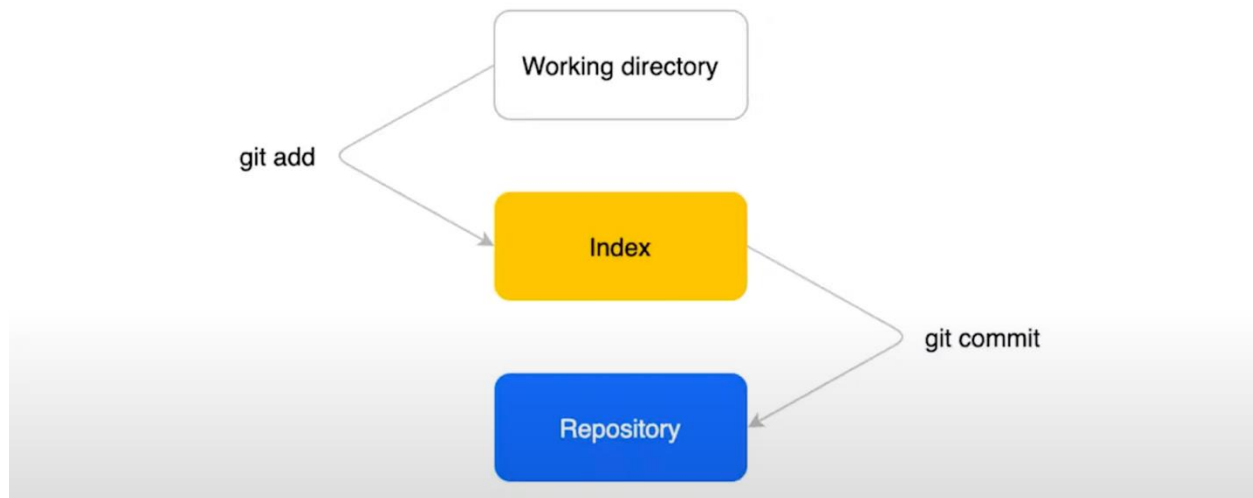
Однонаправленный список коммитов



И коммиты в свою очередь объединяются в однонаправленный список от более новых к старым, где в каждом новом коммите есть ссылка на его родителя.

Состояния

Три состояния



Итак, файлы у нас могут находиться в git-е в трех состояниях: первое это рабочая директория. По сути, это слепок хранилища на текущую версию, и мы можем вносить в нее изменения, при этом никак не затрагивая само хранилище. С помощью команды **git add** мы вводим файлы в индексированное состояние. Это состояние обозначает, что файлы помечены, как готовые для фиксации в репозитории. И с помощью команды **git commit** мы их фиксируем.

Давайте посмотрим на это вживую!

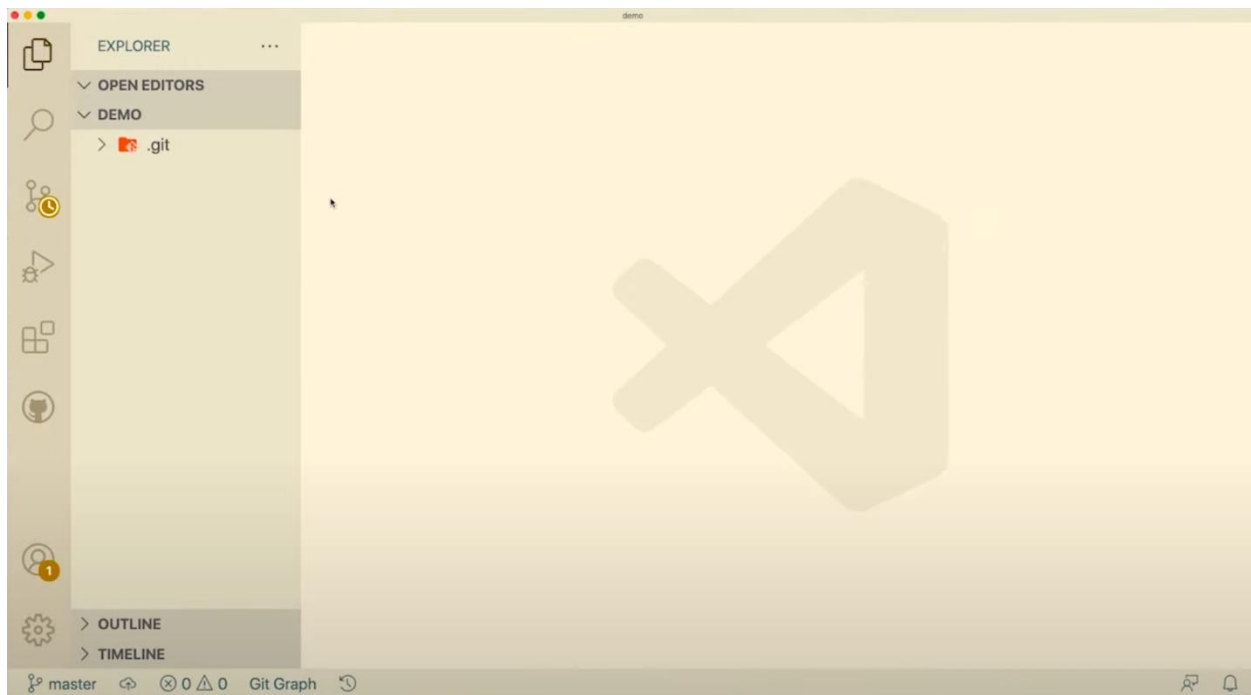
Для начала создадим репозиторий командой **git init** и назовем его **demo**.

git init demo

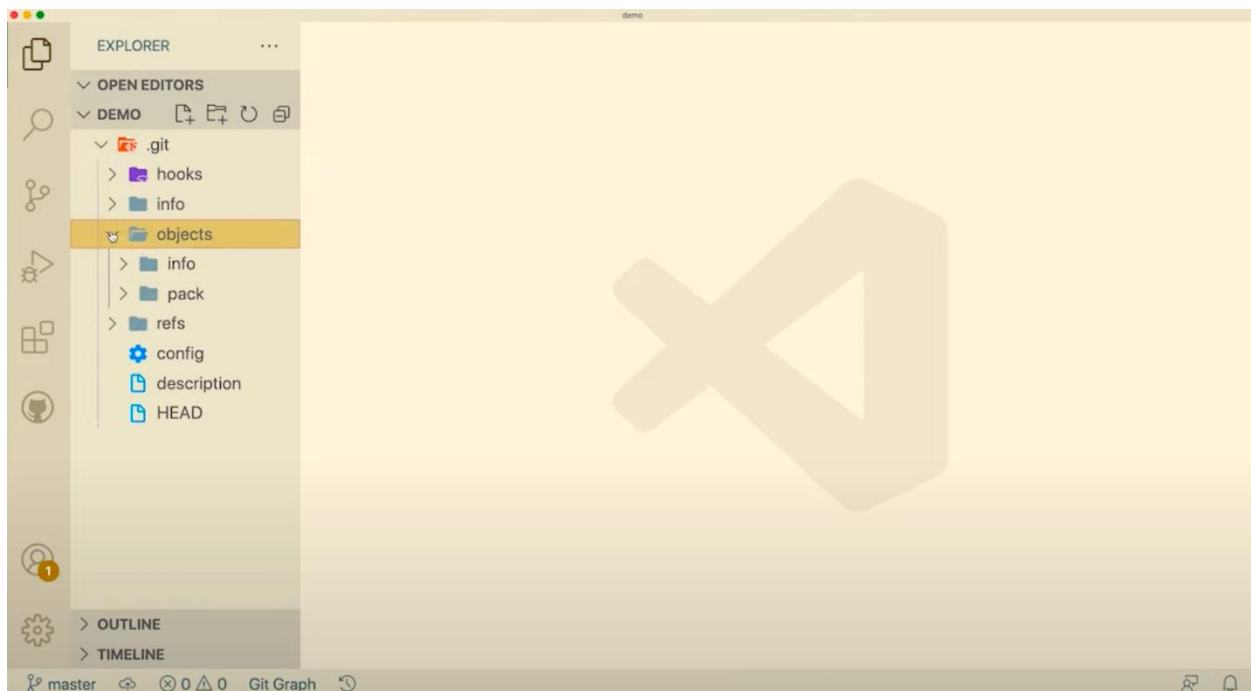
Далее откроем его в редакторе (открываем demo).

code .

Итак, мы сможем увидеть, что у нас есть внутри одна папка **.git** – она нужна для нужд гита, он там хранит объекты и так далее.



Мы заглянем внутрь нее и нас интересует здесь папка objects. В ней он как раз таки и хранит все наши объекты, о которых я говорил. Сейчас тут две пустые папки и нас они пока не интересуют.



Давайте пока создадим какой-нибудь файл.

Создаем файл, пишем туда к примеру: «hello shri», и называем его readme.

echo 'hello shri' > readme.md

И с помощью команды **git status** мы всегда сможем посмотреть в каком состоянии находится у нас репозиторий.

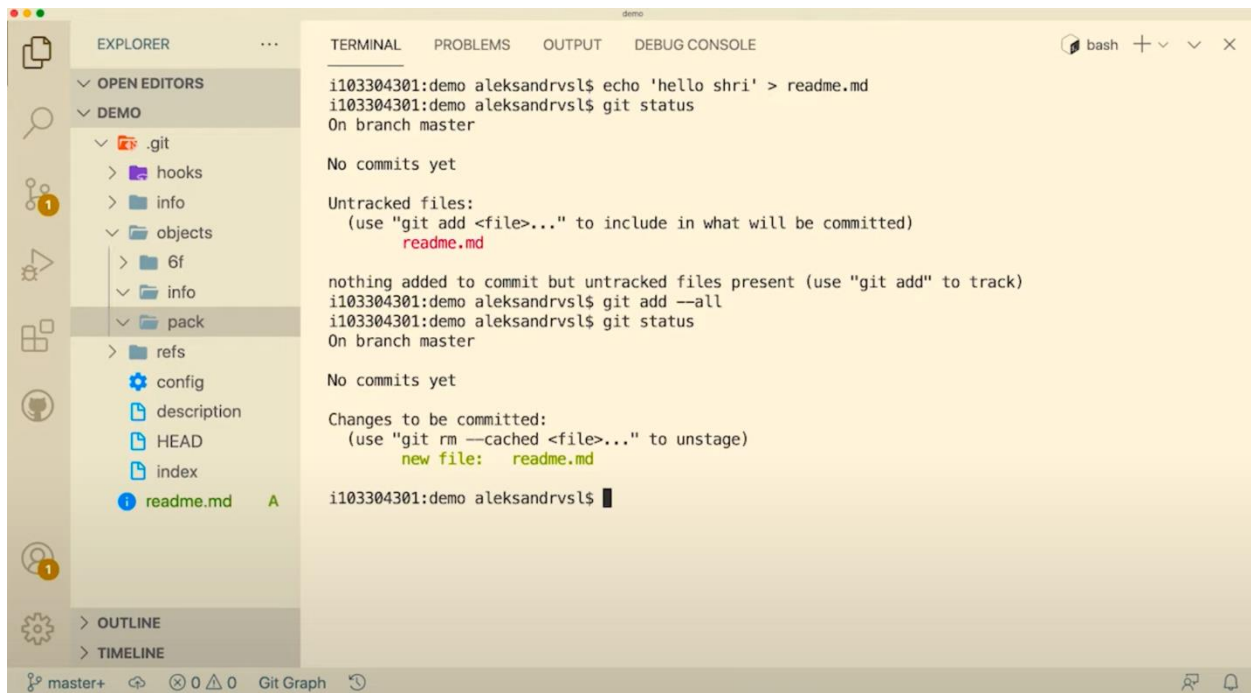


Выполняем **git status** и видим, что у нас здесь появился этот файл, подсвечен он красным цветом и это означает, что он находится в рабочей директории или он в данный момент не отслеживаем.

С помощью команды **git add** можем перенести его уже в индекс, то есть в индексированное состояние. Я могу выполнить эту команду с разными параметрами, к примеру, указать какой-то паттерн, либо просто указать имя файла, который я хочу добавить, либо указать флаг **--all** и он добавит все не отслеживаемые изменения в индекс. Выполним эту команду.

git add --all

Смотри на статус (команда **git status**) – теперь у нас файл помечен зеленым цветом – это означает, что он у нас в индексе.



И последней командой мы его добавим в репозиторий, то есть создадим первый наш коммит (команда **git commit**). Так, эту команду мы можем выполнить либо с флагом **m** и в кавычках указать что именно мы добавили, либо просто можем выполнить команду **git commit**, откроется редактор.

git commit

Например, у меня сейчас по умолчанию открылось папо, на самом деле это можно настроить. Есть команда еще **git config**. Про нее можно отдельно потом почитать, там очень много настроек, в том числе и выбор редактора, в котором мы будем вводить сообщение.

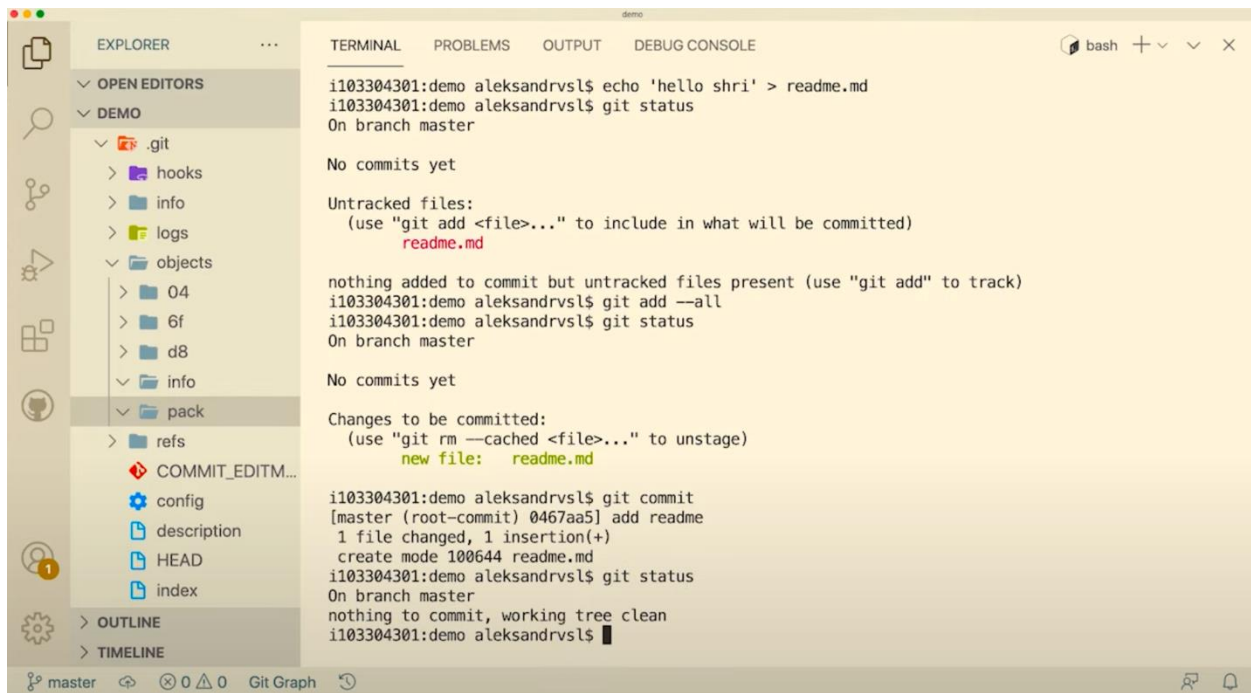
Теперь мы должны ввести сообщение, и оно должно быть максимально понятное и отображать суть внесенных нами изменений. В данном случае мы добавили файл **readme**. Так и запишем.

add readme

Тут можно конечно пропустить еще несколько строчек и добавить описание более подробно, но у нас просто это начальный коммит и мы напишем просто **initial commit**.

initial commit

После сохраняем и выходим отсюда. Смотрим **git status** (выполняем команду **git status**).



```
i103304301:demo aleksandrsvl$ echo 'hello shri' > readme.md
i103304301:demo aleksandrsvl$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  readme.md

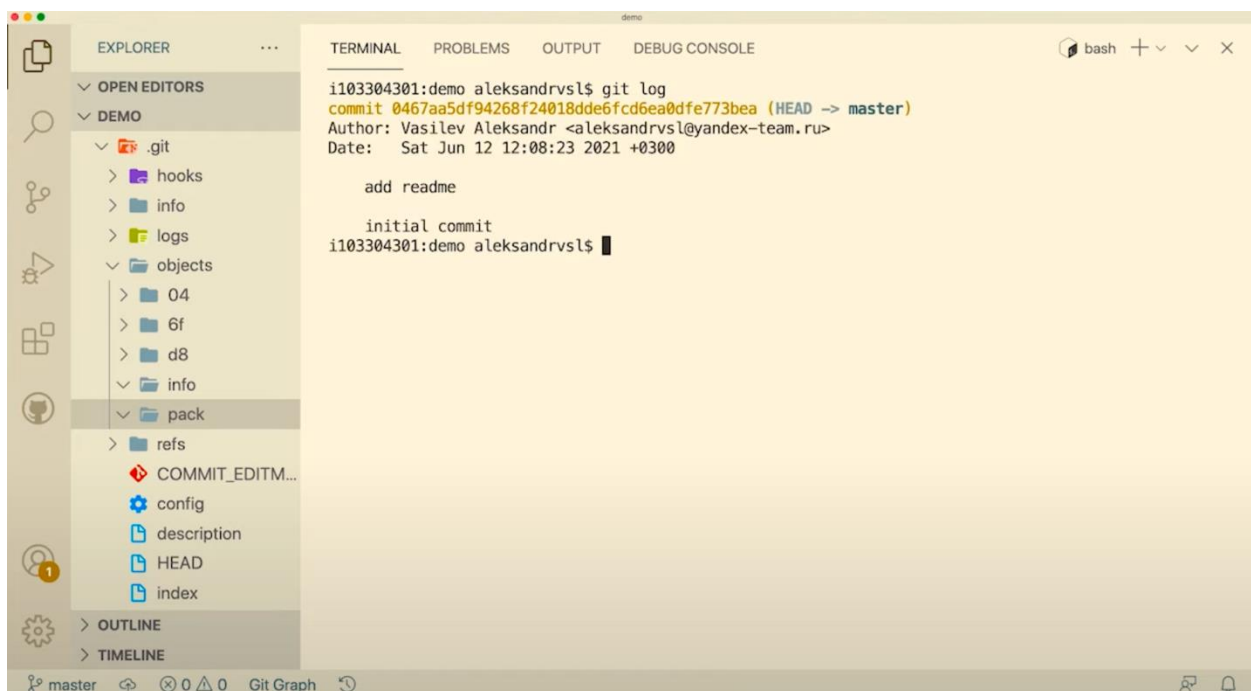
nothing added to commit but untracked files present (use "git add" to track)
i103304301:demo aleksandrsvl$ git add --all
i103304301:demo aleksandrsvl$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
  new file:   readme.md

i103304301:demo aleksandrsvl$ git commit
[master (root-commit) 0467aa5] add readme
1 file changed, 1 insertion(+)
 create mode 100644 readme.md
i103304301:demo aleksandrsvl$ git status
On branch master
nothing to commit, working tree clean
i103304301:demo aleksandrsvl$
```

У нас теперь рабочая директория чистая и у нас добавился коммит. С помощью команды **git log** мы можем посмотреть на этот коммит. Вот он.



```
i103304301:demo aleksandrsvl$ git log
commit 0467aa5df94268f24018dde6fcd6ea0dfe773bea (HEAD -> master)
Author: Vasilev Aleksandr <aleksandrsvl@yandex-team.ru>
Date: Sat Jun 12 12:08:23 2021 +0300

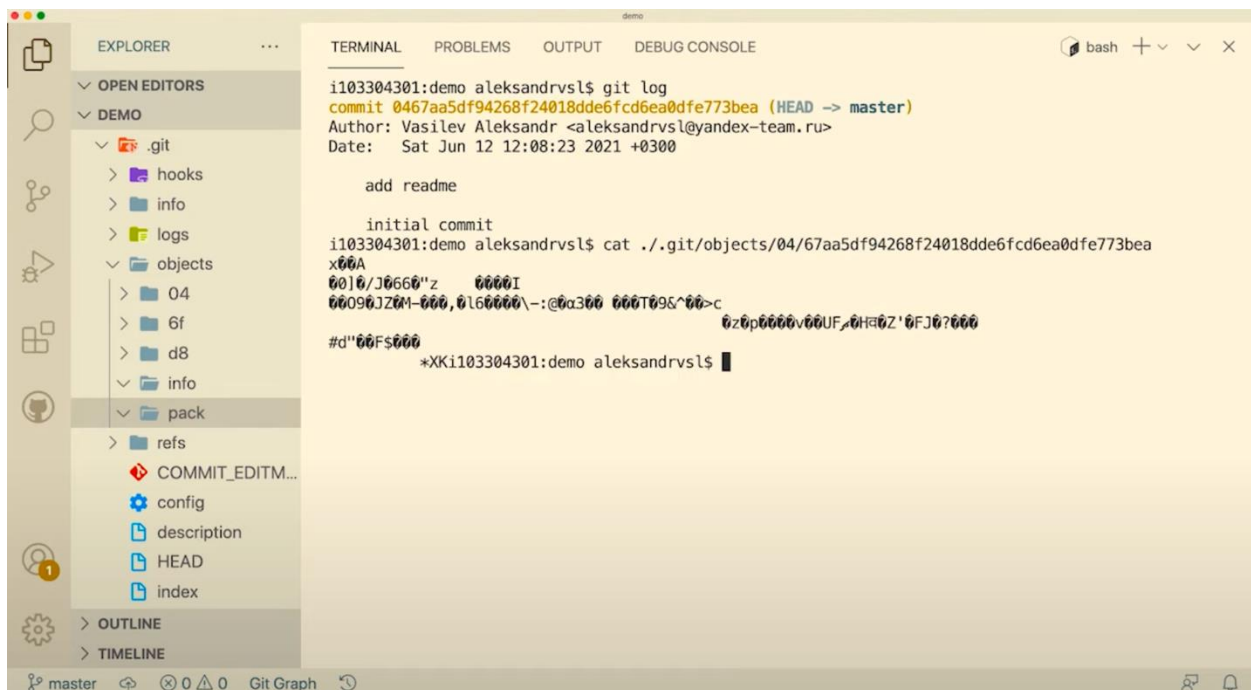
    add readme

    initial commit
i103304301:demo aleksandrsvl$
```

Это единственный коммит пока что в нашем репозитории. И вы могли заметить наверное, что у нас в папке `objects` появились некоторые объекты. Это как раз и есть те объекты, в которых `git` хранит всю информацию. Там лежат наши блобы, деревья и коммиты.

Давайте посмотрим на первый такой файл. Мы его выведем в консоль командой **cat**.

cat ./git/objects/04/...

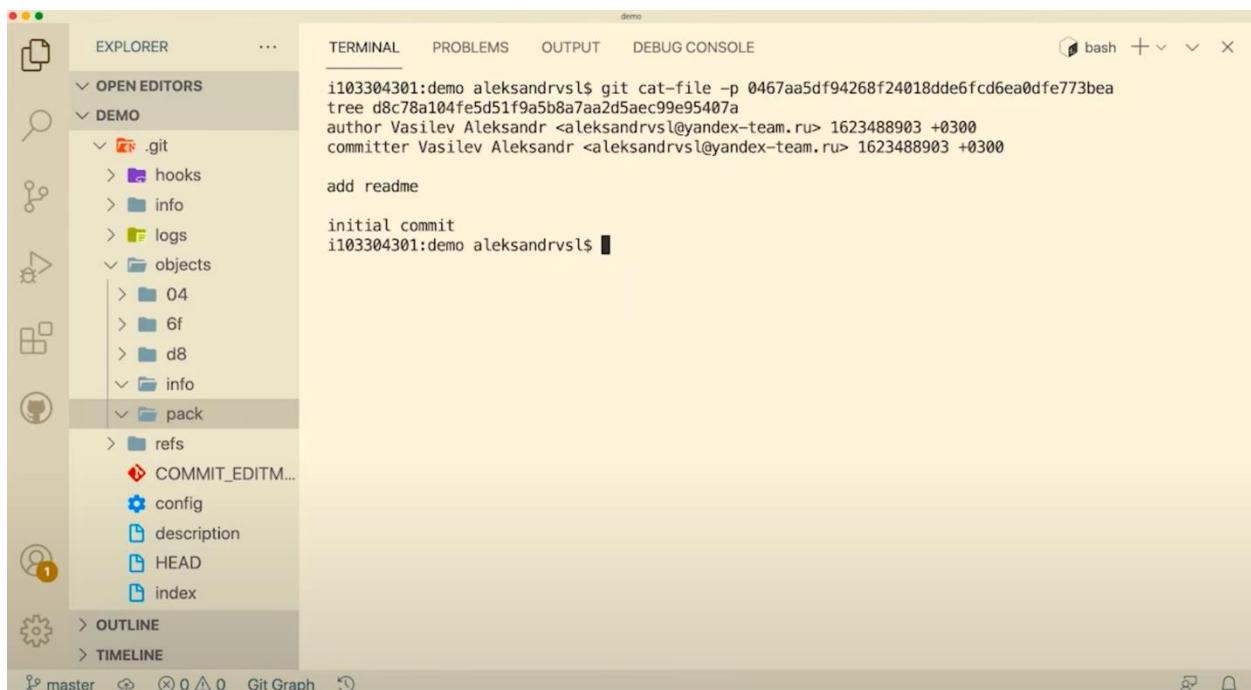


```
i103304301:demo aleksandrsl$ git log
commit 0467aa5df94268f24018dde6fcd6ea0dfe773bea (HEAD -> master)
Author: Vasilev Aleksandr <aleksandrsl@yandex-team.ru>
Date: Sat Jun 12 12:08:23 2021 +0300

    add readme

    initial commit
i103304301:demo aleksandrsl$ cat ../.git/objects/04/67aa5df94268f24018dde6fcd6ea0dfe773bea
x00A
00]0/06660"z 0000I
00090JZ0M-000,0160000\--:@0a300 000T09S^00>c
0z0p0000v00UF,0H0Z'0FJ07000
#d"00F$000
*Xi103304301:demo aleksandrsl$
```

Он нам вывел какую-то непонятную информацию, потому что все файлы там бинарные. У нас есть команда, чтобы прочитать эти бинарные файлы. Давайте вернем сейчас предыдущую команду, и команда эта называется **git cat-file**. У нее есть флаг **-p**, который обозначает слово pretty – красивый, то есть чтобы вывод был красивым.



```
i103304301:demo aleksandrsl$ git cat-file -p 0467aa5df94268f24018dde6fcd6ea0dfe773bea
tree d8c78a104fe5d51f9a5b8a7aa2d5aec99e95407a
author Vasilev Aleksandr <aleksandrsl@yandex-team.ru> 1623488903 +0300
committer Vasilev Aleksandr <aleksandrsl@yandex-team.ru> 1623488903 +0300

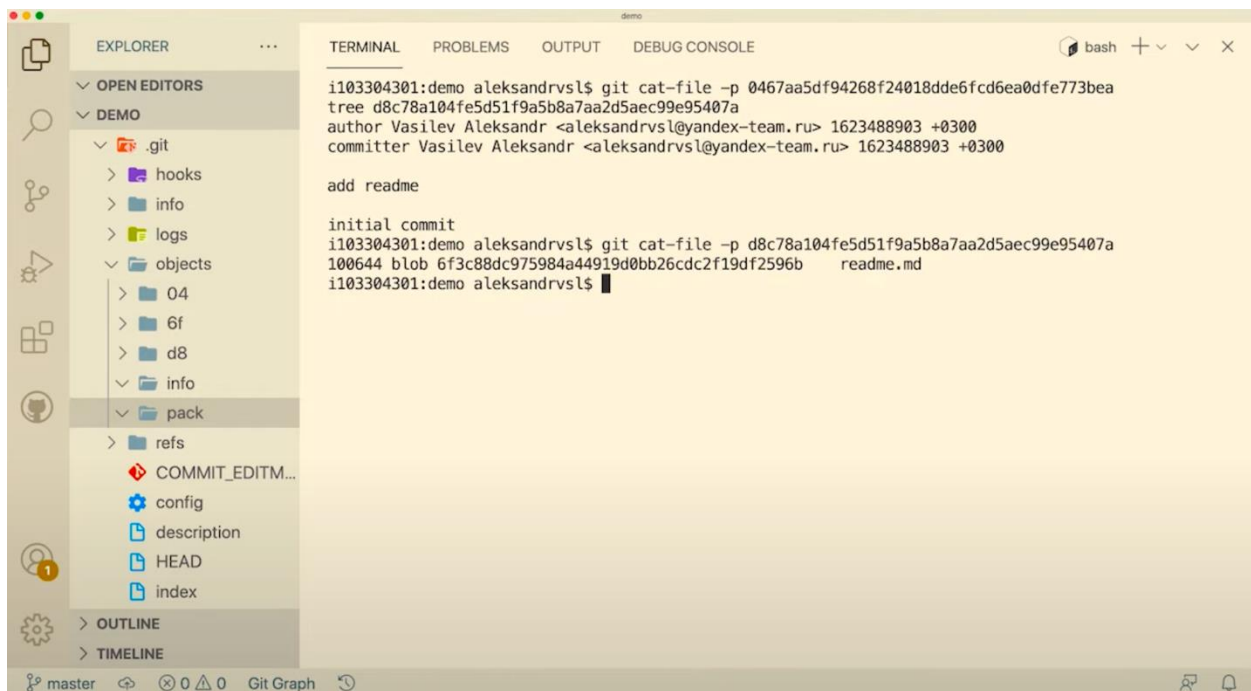
    add readme

    initial commit
i103304301:demo aleksandrsl$
```

Все остальное мы удаляем. И вот в эту команду мы должны передать уже хэш целиком, без пути.

Так, мы попали на коммит, здесь у нас, как говорилось ранее, видна ссылка на дерево, есть автор, коммитер – сейчас это я, и они одинаковые. На самом деле они могут отличаться. Я постараюсь сказать, когда они будут отличаться, чуть позже. А сейчас мы можем посмотреть таким же образом на дерево.

Выбираем его и точно также, при помощи команды **git cat-file -p** .

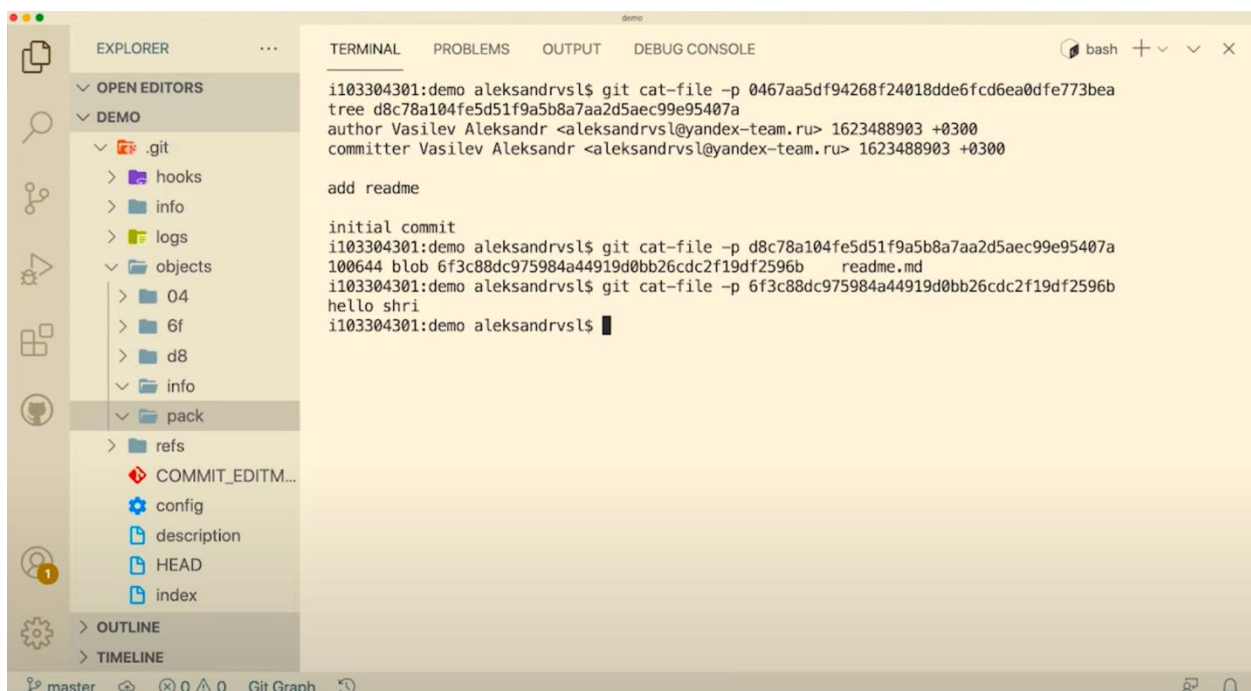


```
i103304301:demo aleksandrsl$ git cat-file -p 0467aa5df94268f24018dde6fcd6ea0dfe773bea
tree d8c78a104fe5d51f9a5b8a7aa2d5aec99e95407a
author Vasilev Aleksandr <aleksandrsl@yandex-team.ru> 1623488903 +0300
committer Vasilev Aleksandr <aleksandrsl@yandex-team.ru> 1623488903 +0300

add README

initial commit
i103304301:demo aleksandrsl$ git cat-file -p d8c78a104fe5d51f9a5b8a7aa2d5aec99e95407a
100644 blob 6f3c88dc975984a44919d0bb26cdc2f19df2596b    README.md
i103304301:demo aleksandrsl$
```

Дерево у нас небольшое, там записан всего один файл – это наш `README`. Но их может быть гораздо больше. Это просто список, перечисленный здесь размер файла, тип, хэш и его название. Так, и последний это собственно наш файл. Мы на него тоже посмотрим (при помощи предыдущей команды). Как мы помним, там должно храниться содержимое нашего файла `README`.



```
i103304301:demo aleksandrsl$ git cat-file -p 0467aa5df94268f24018dde6fcd6ea0dfe773bea
tree d8c78a104fe5d51f9a5b8a7aa2d5aec99e95407a
author Vasilev Aleksandr <aleksandrsl@yandex-team.ru> 1623488903 +0300
committer Vasilev Aleksandr <aleksandrsl@yandex-team.ru> 1623488903 +0300

add README

initial commit
i103304301:demo aleksandrsl$ git cat-file -p d8c78a104fe5d51f9a5b8a7aa2d5aec99e95407a
100644 blob 6f3c88dc975984a44919d0bb26cdc2f19df2596b    README.md
i103304301:demo aleksandrsl$ git cat-file -p 6f3c88dc975984a44919d0bb26cdc2f19df2596b
hello shri
i103304301:demo aleksandrsl$
```

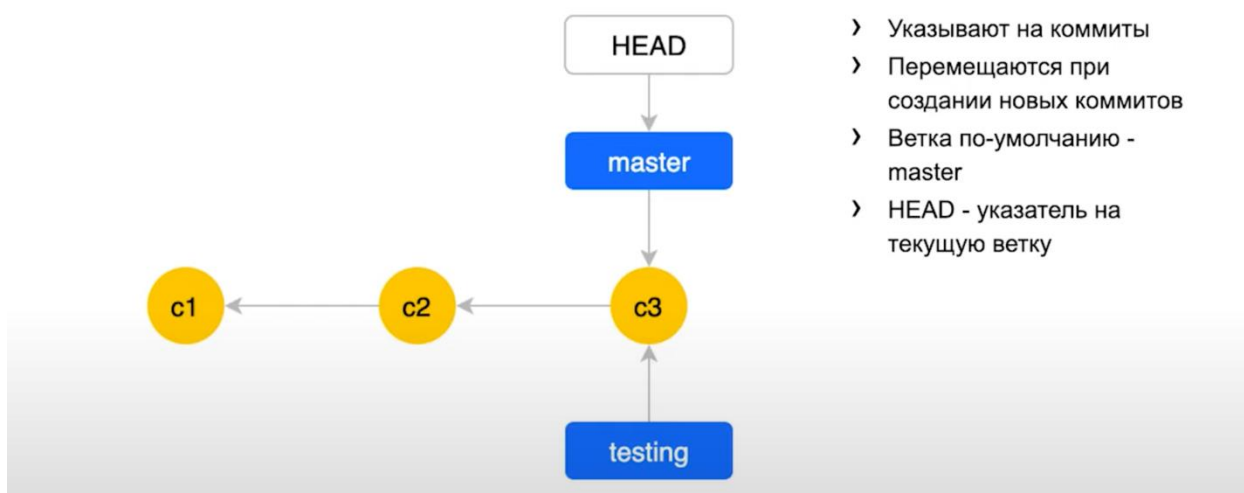
Так и есть.

Еще пару слов про то, как формируются у нас эти папки (названия папок). Git хэширует содержимое файла, первые 2 символа хэша он берет в качестве имени для папки, а остальную часть в качестве имени файла. Таким образом он обходит ограничения некоторых файловых систем на количество файлов в папках.

Ветки и теги

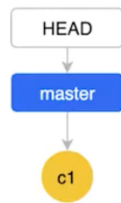
Далее у нас ветки. Как раз таки я говорил про параллельную работу и механизм ветвления в гите помогает нам организовывать эту параллельную работу.

Ветки



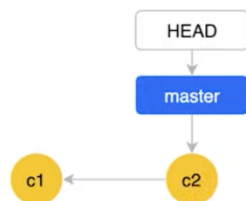
По умолчанию в гите создается ветка **master**. Ветка – это указатель на коммит, а **head** это указатель на текущую ветку, или где мы находимся в настоящий момент.

git merge feature [--abort]



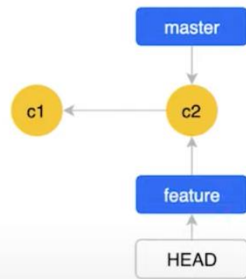
Представим такую ситуацию, вот у нас есть репозиторий с одним коммитом, прямо как мы только что сделали. Мы добавляем туда еще один.

git merge feature [--abort]



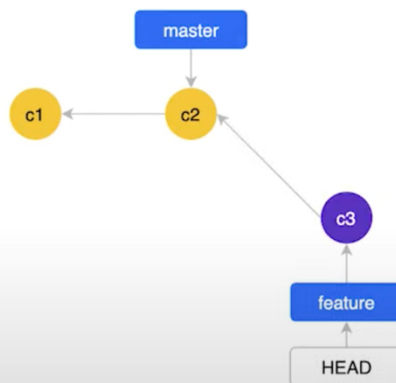
Ветка **master** передвигается на него вместе с указателем на эту ветку **head**. После чего мы решаем, что нам необходимо создать какой-то новый функционал, новый модуль. И для этого мы создадим новую ветку.

git merge feature [--abort]



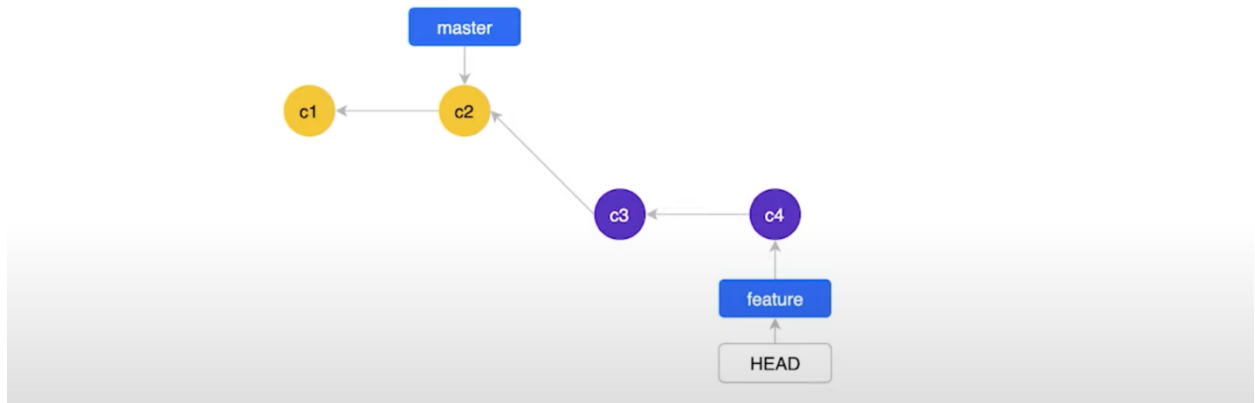
Создаем новую ветку и переключаемся на нее. Ветка называется **feature**. У нас появляется новый указатель.

git merge feature [--abort]



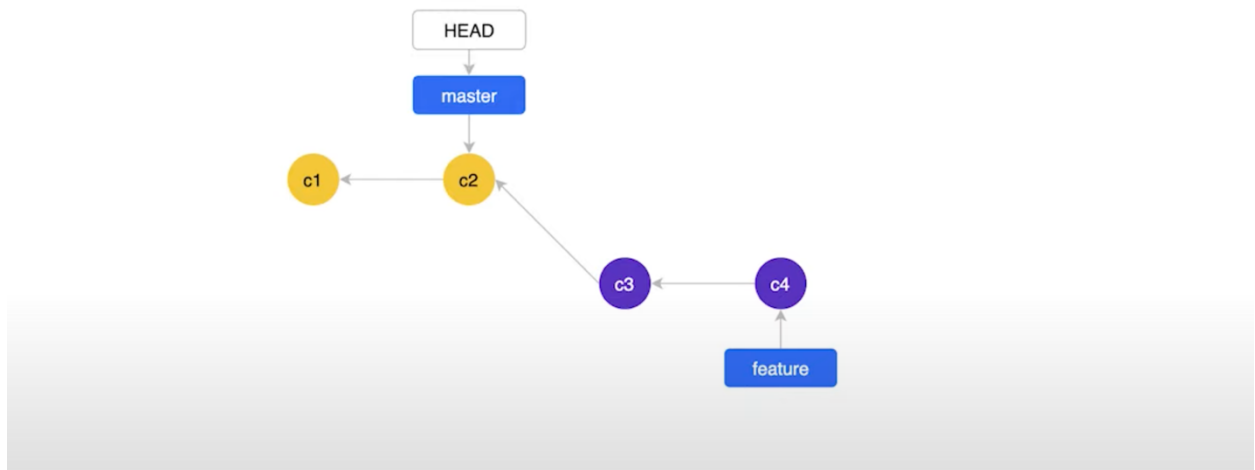
И добавляем туда изменения: одно изменение, второе.

git merge feature [--abort]



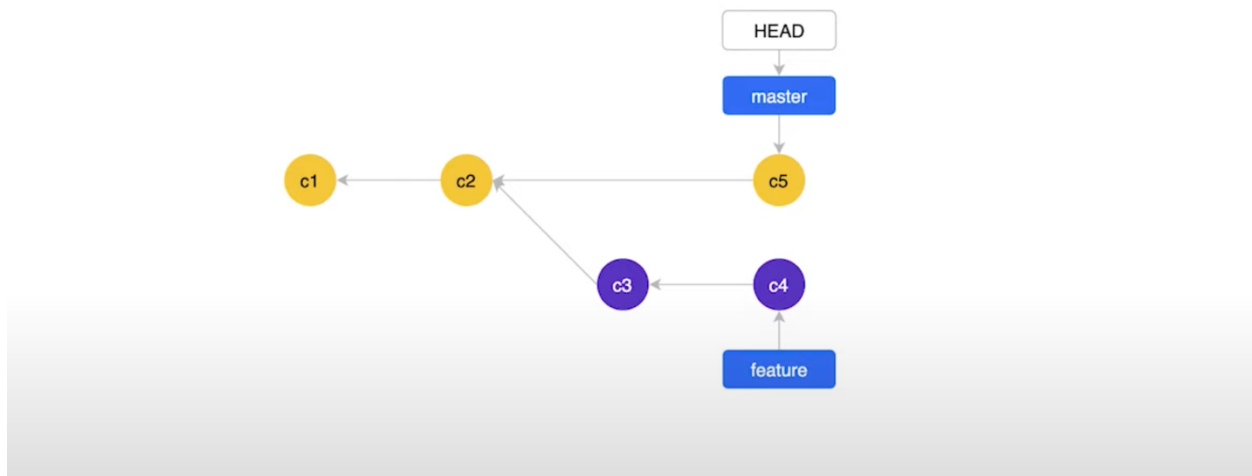
После чего, мы вдруг обнаруживаем, что в ветке **master** необходимо внести какой-то hotfix. Поэтому мы переключаемся на эту ветку – **head** перепрыгивает на него.

git merge feature [--abort]



И добавляем этот hotfix в новый коммит.

git merge feature [--abort]



И у нас получилась вот такая вот параллельная структура, поэтому это и называется параллельная работа.

Давайте посмотрим на примере.

У нас есть репозиторий, давайте выполним команду **git log**. Здесь у нас 2 коммита и мы должны сейчас добавить новую ветку и внести в нее какие-то изменения.

Мы можем добавить новую ветку командой **git branch**. Назовем эту ветку **feature**.

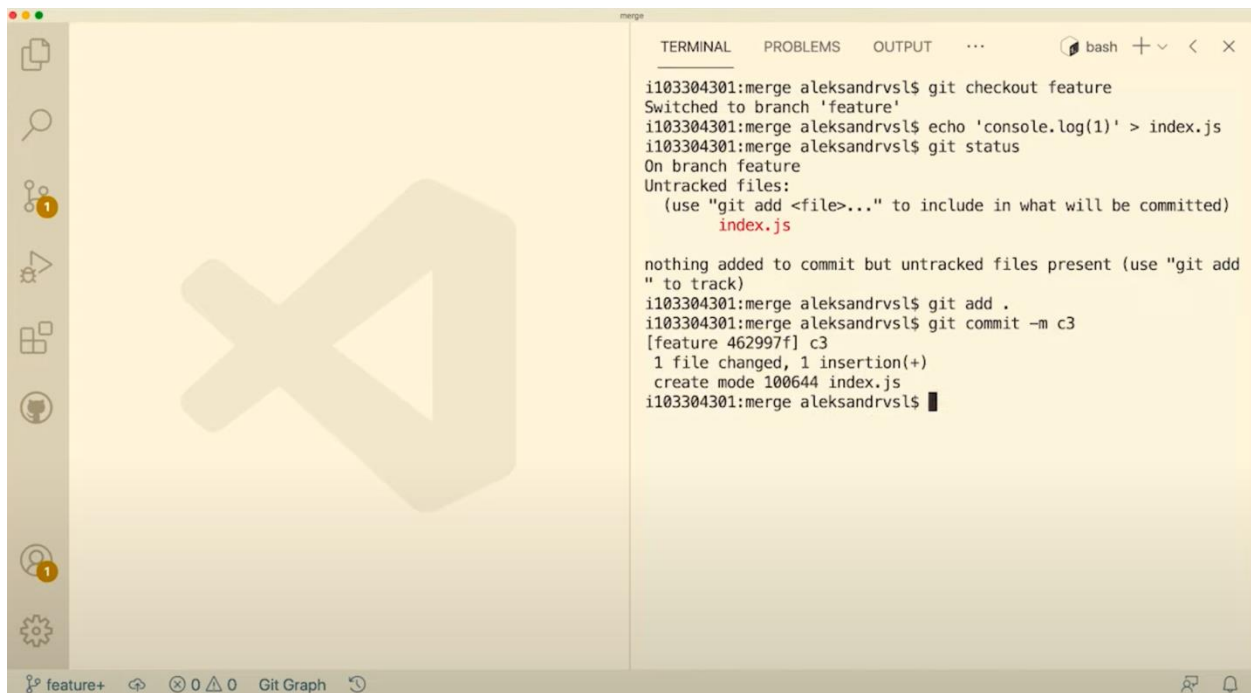
git branch feature

Теперь давайте перейдем в эту ветку командой **git checkout feature**.

И начнем вносить какие-то изменения, например, добавим новый файл, напишем туда `console.log(1)` и назовем его `index.js`

```
echo 'console.log(1)' > index.js
```

Далее проверяем, что он у нас добавился (командой **git status**), добавляем его в индекс и отправляем в репозиторий командой **git commit**.



```
i103304301:merge aleksandrsvls$ git checkout feature
Switched to branch 'feature'
i103304301:merge aleksandrsvls$ echo 'console.log(1)' > index.js
i103304301:merge aleksandrsvls$ git status
On branch feature
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.js

nothing added to commit but untracked files present (use "git add" to track)
i103304301:merge aleksandrsvls$ git add .
i103304301:merge aleksandrsvls$ git commit -m c3
[feature 462997f] c3
1 file changed, 1 insertion(+)
create mode 100644 index.js
i103304301:merge aleksandrsvls$
```

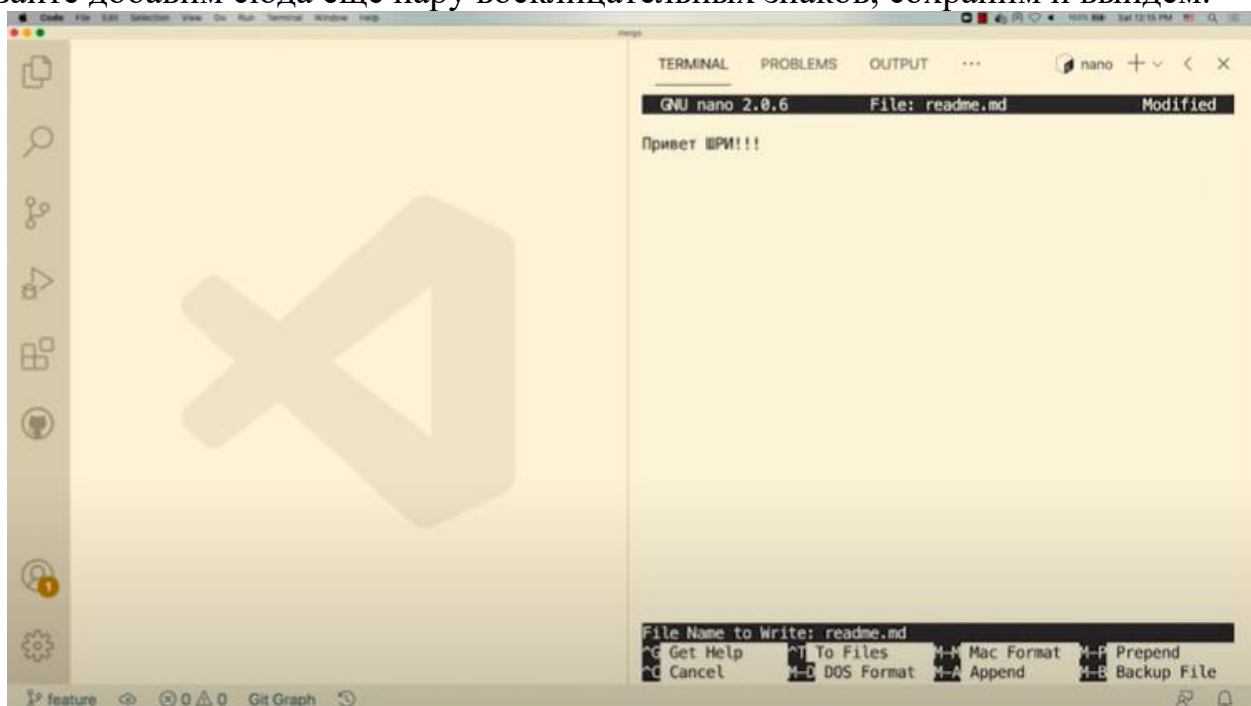
В данном случае мы просто укажем флаг **-m** и напомним сюда **c3**.

Вспоминаем, что у нас тут должен лежать еще какой-то файл (проверяем командой **ls**).

Это readme. Давайте его отредактируем и внесем еще один коммит.

nano readme.md

У нас написано: «Привет ШРИ!», мне кажется недостаточно эмоционально. Давайте добавим сюда еще пару восклицательных знаков, сохраним и выйдем.

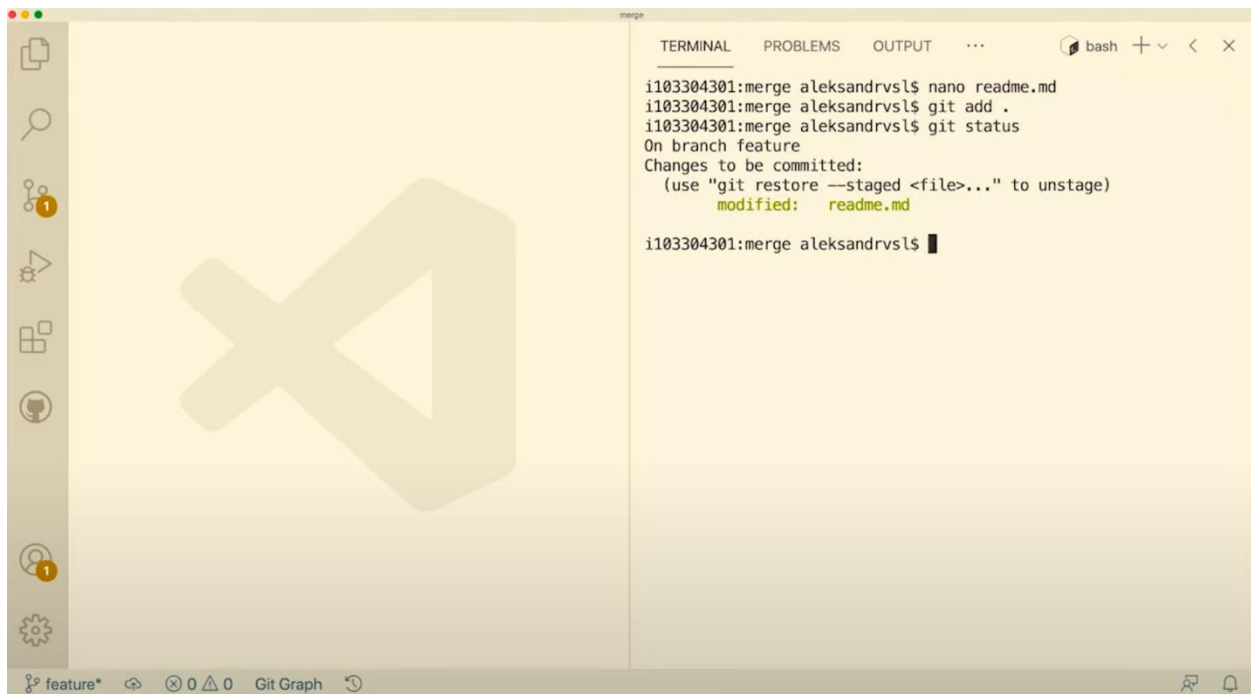


```
GNU nano 2.0.6 File: readme.md Modified

Привет ШРИ!!!

File Name to Write: readme.md
^G Get Help ^T To Files ^H Mac Format ^R Prepend
^C Cancel ^O DOS Format ^A Append ^B Backup File
```

Добавим его снова в индекс, проверим и закоммитим – **c4**.



```
i103304301:merge aleksandrsl$ nano readme.md
i103304301:merge aleksandrsl$ git add .
i103304301:merge aleksandrsl$ git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   readme.md

i103304301:merge aleksandrsl$
```

Теперь, чтобы у нас получилось также как на слайде мы должны переключиться в ветку **master** и добавить туда еще одно изменение. Переключаемся в ветку **master** командой **git checkout master**, и редактируем тот же самый файл.

nano readme.md

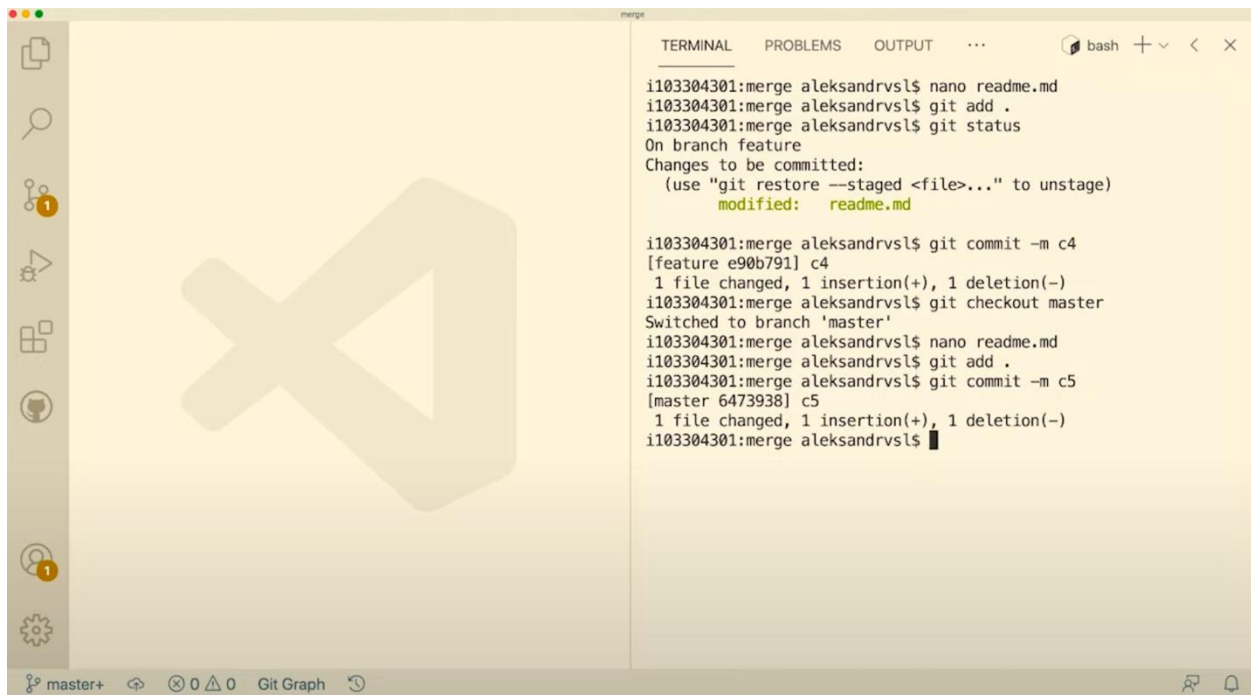
Здесь, к примеру, мы напишем «Привет Яндекс!».



```
GNU nano 2.0.6      File: readme.md      Modified
Привет Яндекс!

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
Y Yes
N No      C Cancel
```

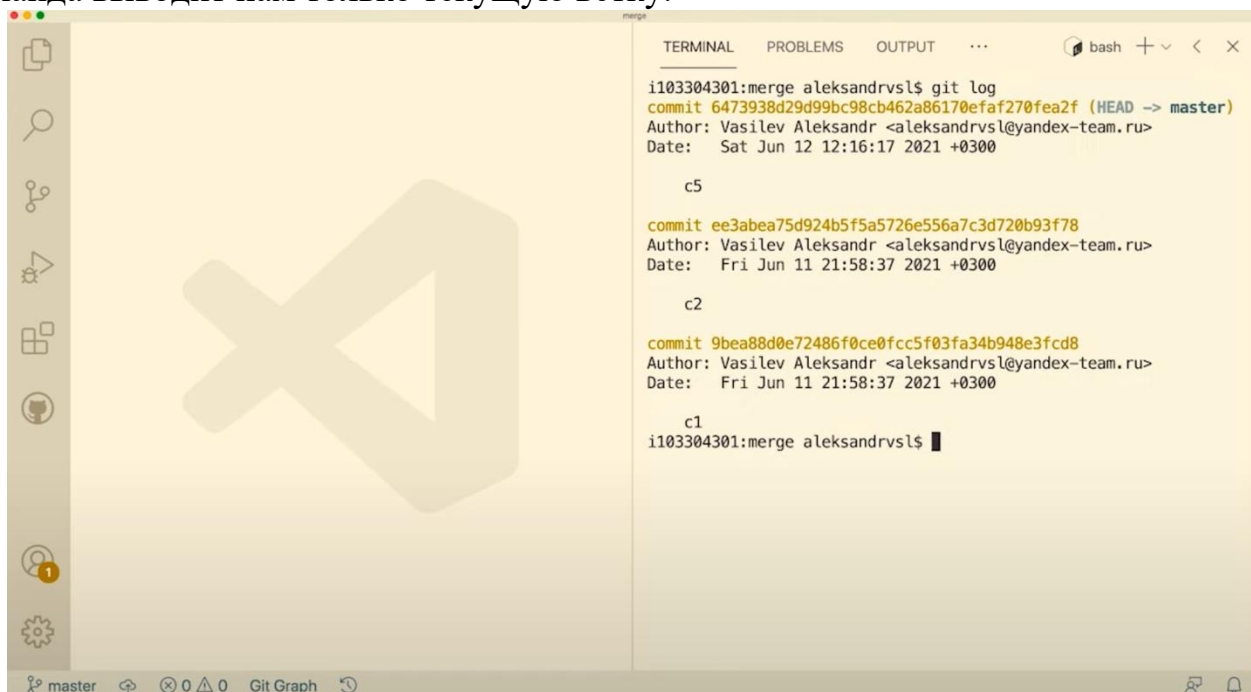
Сохраняем, выходим и добавляем новый коммит. Сначала вносим файл в индекс. После коммитим с коммитом **c5**.



```
i103304301:merge aleksandrsl$ nano readme.md
i103304301:merge aleksandrsl$ git add .
i103304301:merge aleksandrsl$ git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   readme.md

i103304301:merge aleksandrsl$ git commit -m c4
[feature e90b791] c4
 1 file changed, 1 insertion(+), 1 deletion(-)
i103304301:merge aleksandrsl$ git checkout master
Switched to branch 'master'
i103304301:merge aleksandrsl$ nano readme.md
i103304301:merge aleksandrsl$ git add .
i103304301:merge aleksandrsl$ git commit -m c5
[master 6473938] c5
 1 file changed, 1 insertion(+), 1 deletion(-)
i103304301:merge aleksandrsl$
```

Итак, мы тут выполнили очень много всяких разных действий и порой бывает сложно понять, где мы сейчас находимся и тем более вспомнить что мы делали до этого. Поэтому командой **git log** мы можем посмотреть, что там, но по умолчанию эта команда выводит нам только текущую ветку.



```
i103304301:merge aleksandrsl$ git log
commit 6473938d29d99bc98cb462a86170efaf270fea2f (HEAD -> master)
Author: Vasilev Aleksandr <aleksandrsl@yandex-team.ru>
Date: Sat Jun 12 12:16:17 2021 +0300

    c5

commit ee3abea75d924b5f5a5726e556a7c3d720b93f78
Author: Vasilev Aleksandr <aleksandrsl@yandex-team.ru>
Date: Fri Jun 11 21:58:37 2021 +0300

    c2

commit 9bea88d0e72486f0ce0fcc5f03fa34b948e3fcd8
Author: Vasilev Aleksandr <aleksandrsl@yandex-team.ru>
Date: Fri Jun 11 21:58:37 2021 +0300

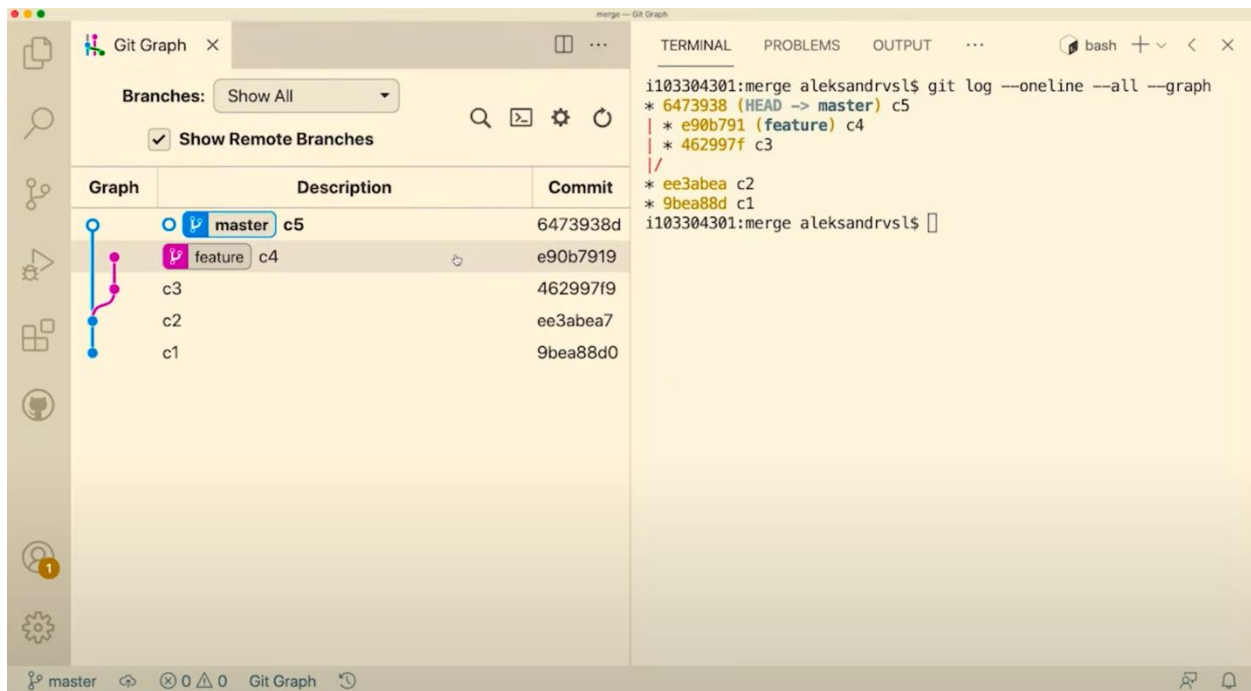
    c1
i103304301:merge aleksandrsl$
```

Здесь у нас всего 3 коммита – **c1**, **c2**, **c5**. Мы можем выполнить эту команду с несколькими флагами, чтобы увидеть всю историю. Давайте добавим несколько флагов, например **--oneline** – это сделает наши коммиты в консоли более компактными. Добавим еще флаг **--all**, чтобы вывести все ветки и еще добавим флаг **--graph**, чтобы увидеть граф нашего репозитория.



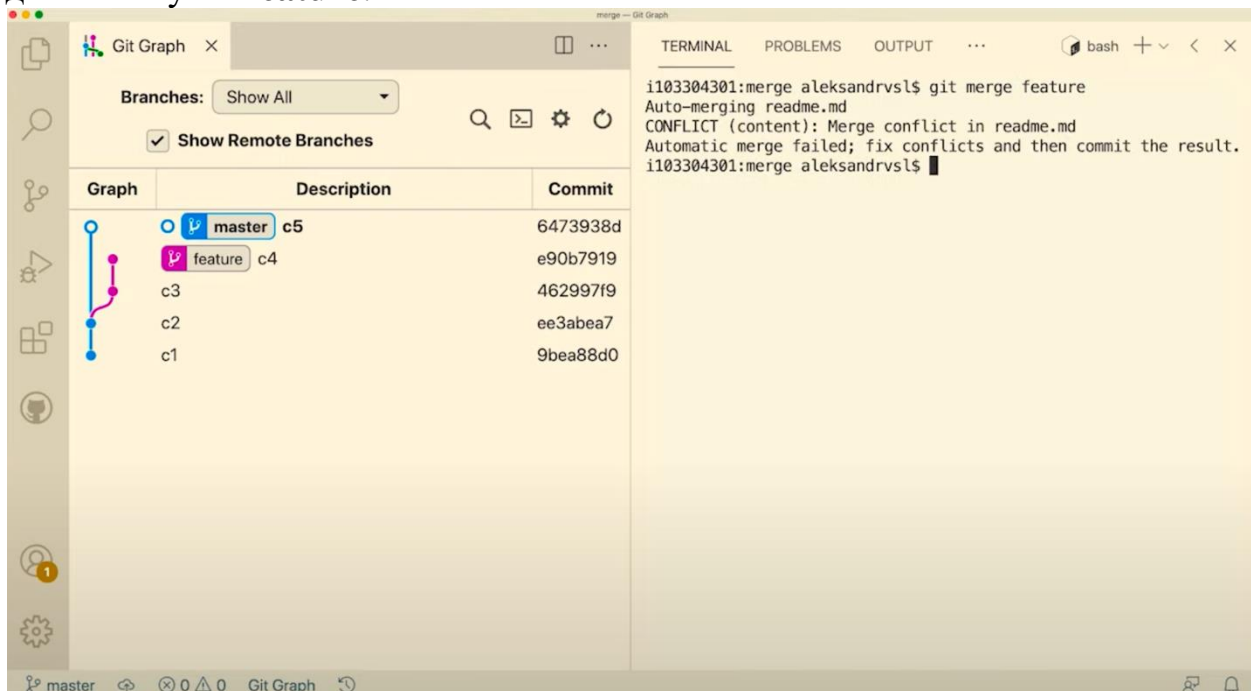
Здесь мы как раз таки и видим нашу параллельную историю, **head** указывает на **master**, также у нас есть ветка **feature**. Собственно картина такая же, как была на слайде. Единственное что, довольно проблематично постоянно вбивать эту команду, поэтому в дальнейшем мы можем использовать инструмент git graph (нижняя панель), который является плагином для нашего редактора.





Здесь точно такая же история. Единственное что **head** тут указывается кружочком.

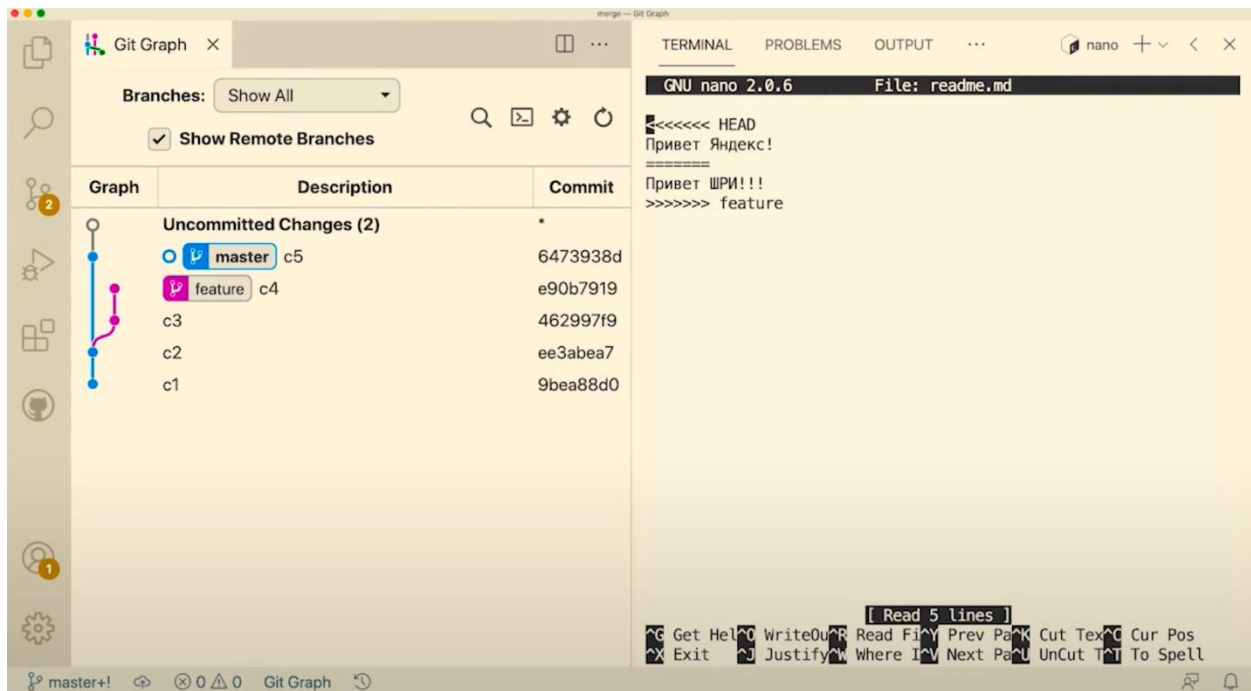
Теперь мы будем выполнять слияние наших веток. Выполняется это командой **git merge**. И здесь мы должны указать ветку, которую мы хотим влить в нашу ветку – в данном случае **feature**.



Git падает с ошибкой – это не случайно, потому что мы изменили файл `readme` в одном и том же месте в двух разных ветках.

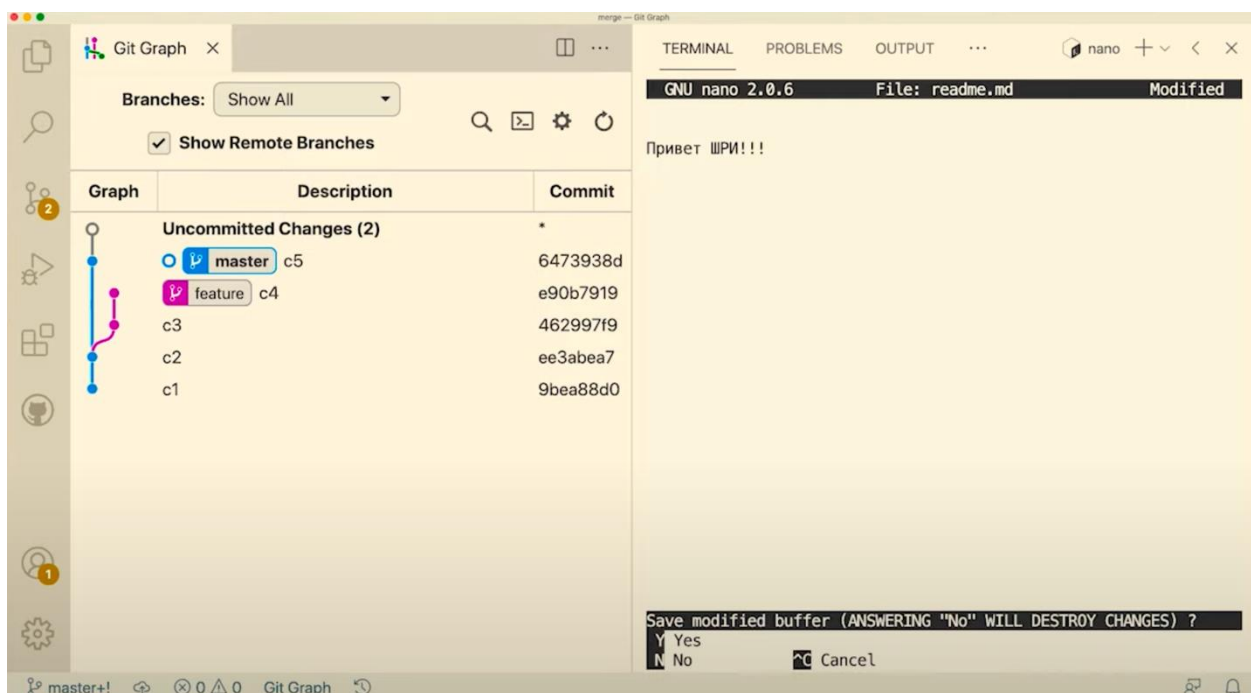
Такое будет периодически происходить и с Вами. Нам надо сейчас решить этот конфликт. Для этого давайте посмотрим, что у нас в этом файлике есть (следующей командой).

nano readme.md

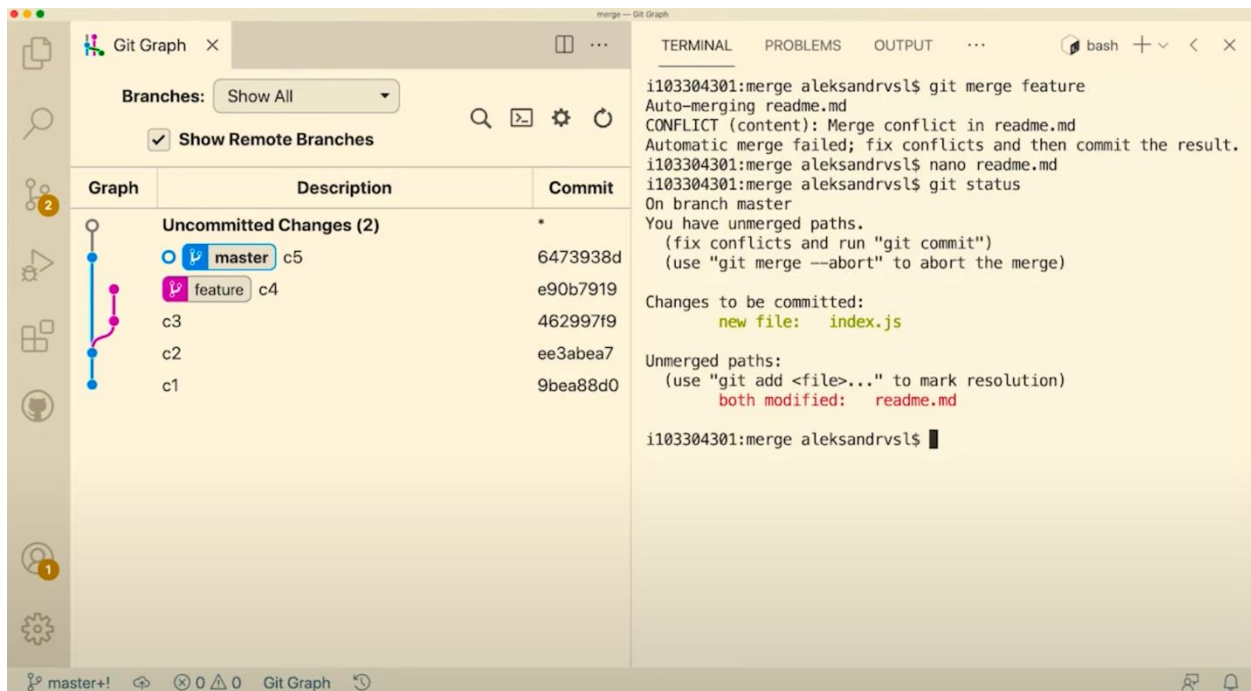


Здесь git добавил специальные конструкции, которыми он помечает конфликты. Наша задача сейчас эти конфликты решить. Мы должны удалить все эти конструкции и выбрать, какое изменение будет для нас актуальным.

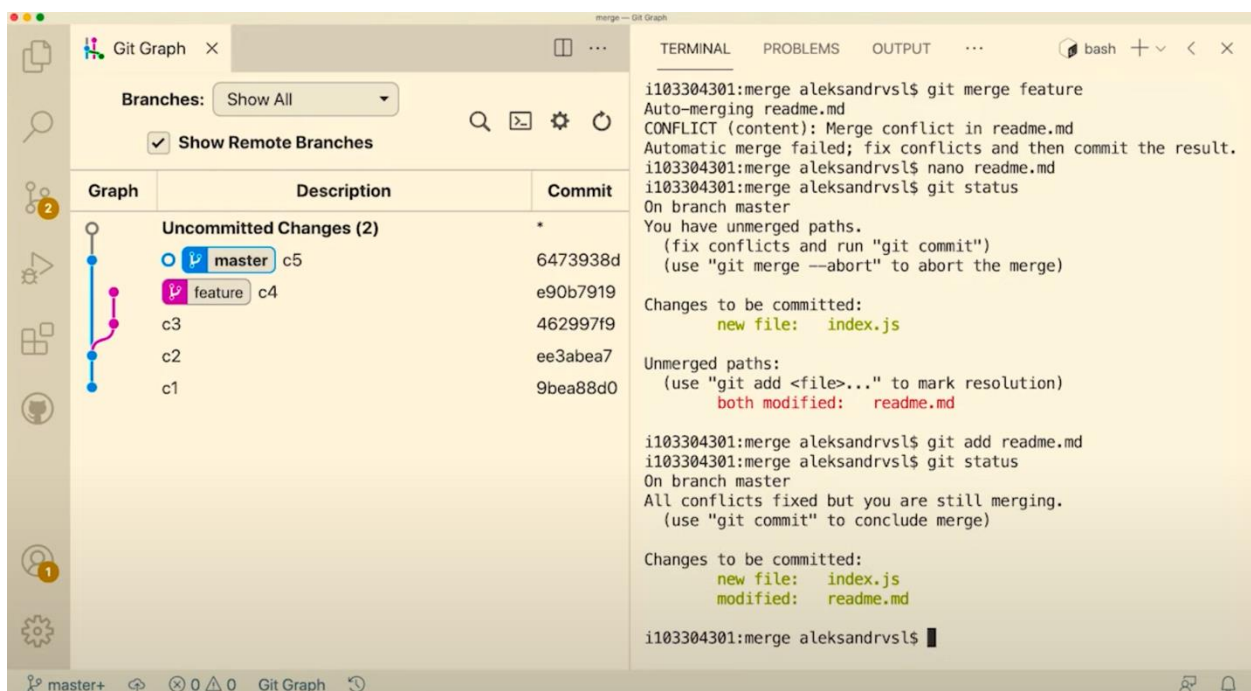
Давайте удалим все конструкции и сохраним.



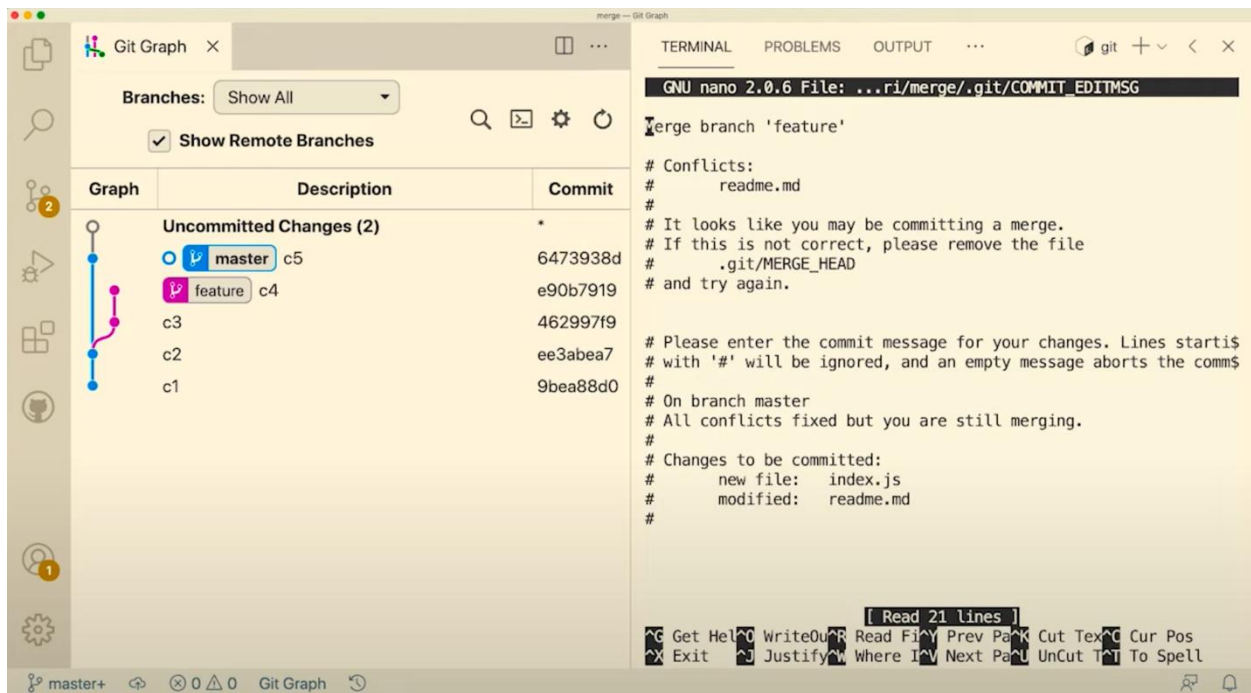
Выполним команду **git status**.



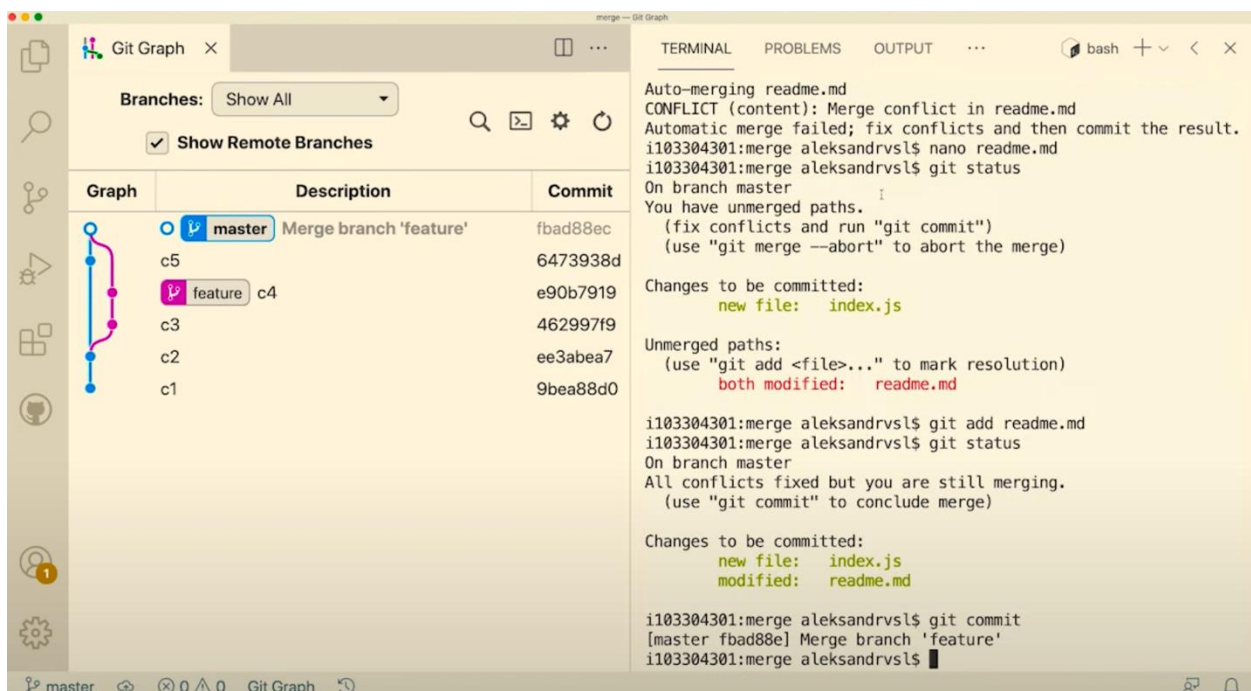
И git нам дает подсказки, когда мы выполняем эту команду. К примеру, чтобы нам закончить решение конфликтов мы должны написать команду **git commit**, либо можем вообще выйти из режима слияния веток, выполнив команду **git merge --abort**. Мы решили сейчас конфликт, поэтому добавляем наш файл в индекс.



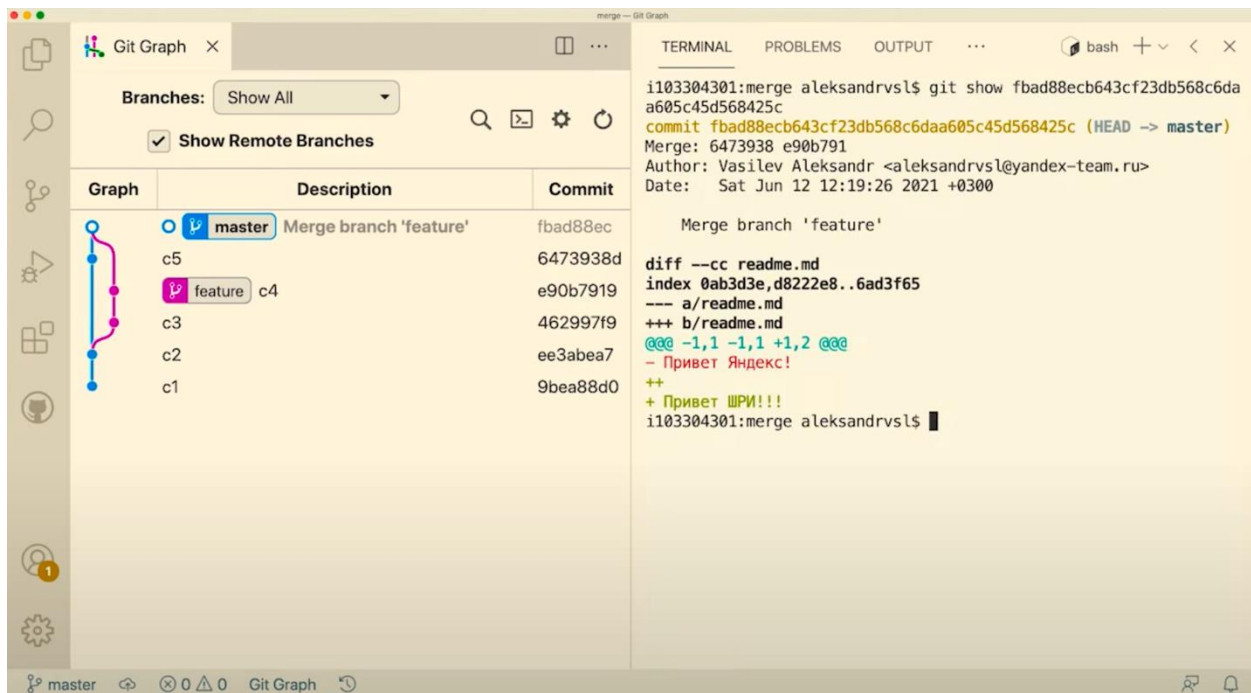
И командой **git commit** завершаем слияние веток.



Git опять нам открывает редактор и предлагает свое сообщение. А именно **merge branch feature**. Собственно то, что мы сейчас сделали. Нас это сообщение устраивает, поэтому просто выходим отсюда.



И мы видим слева на графе, что у нас появился новый коммит. Этот коммит называется коммит слияния и у него сейчас 2 родителя – один из ветки **feature**, другой из ветки **master**. Мы можем посмотреть на этот коммит командой **git show**. Давайте сначала с помощью **git log** возьмем хэш этого коммита, и с помощью команды **git show** выведем его.



Здесь указаны изменения, которые произошли и, что более важно, указано, что это коммит слияния с двумя родителями.

Таким образом мы изучили первый вариант объединения веток.

Еще одним вариантом будет команда **git merge**.