Практика работы с Git и GitHub

Подготовка к работе

- Установите Git и SourceTree для своей ОС
  - https://www.sourcetreeapp.com/

При работе с Windows, Git можно скачать по ссылке:

- https://git-scm.com/download

Если работаем на Windows, то используйте программу Git Bash, которая запускает терминал.

Устанавливать SourceTree необязательно и вы можете использовать любой другой подобный инструмент

Обязательная настройка

Необходимо настроить информацию об авторе, иначе, без этих данных Git не позволит нам делать коммиты Используйте свое имя, фамилию и адрес эл. почты.

```
$ git config --global user.name "Artem Starostenko"
$ git config --global user.email artemstarostenko65@gmail.com
```

❗ Важно! Указанные данные должны совпадать с именем, фамилией и электронной почтой учетной записи на GitHub

Начинаем отслеживать историю

- Создайте пустую директорию `practice-git` и перейдите в нее
- Инициализируйте репозиторий, используя команду `git init`
- Убедитесь, что создалась директория `.git`

Добавим файл

1. Создаем текстовый файл:
   - `echo "My first line in project" > file.txt`
2. Выполним `git status`.
   - Почему файл помечен, как *untracked*?
3. Добавляем файл в индекс:
   - `git add file.txt`
4. Снова выполним `git status`
5. Смотрим проиндексированные изменения:
   - `git diff --cached`

Сохраним изменения в истории

1. Создаем коммит:
   - `git commit -m "My first commit"`
2. Проверьте, что коммит есть в истории:
   - `git log`
3. Посмотрите информацию о последнем коммите:
   - `git show`

Добавим еще один коммит

1. Добавим строчку в существующий файл:
    - `echo "London is the capital of GB" >> file.txt`
2. Посмотрим непроиндексированные изменения `git diff`
    - Сравните вывод с результатом `git diff --cached`
3. Создадим коммит: `git commit -am "Capital added"`
    - Удалось ли нам сделать коммит без добавления в Index? Или мы автоматически добавили изменения перед коммитом?
4. Проверим в истории, сделали ли мы коммит:
    - `git log`

Исправьте предыдущий коммит

В последнем коммите мы зафиксировали добавление строки "*London is the capital of GB*" в файл `file.txt`, но мы забыли добавить в файл, что "*Moscow is the capital of Russia*".

Исправьте последний коммит (не добавляйте новый) и в сообщении коммита укажите, что была добавлена не одна столица, а несколько.

> Если затрудняетесь - ответы даны на следующем слайде

Ответы

- Добавим новую строку:

```
echo "Moscow is the capital of Russia" >> file.txt
```

- Добавим изменения в *stage*:

```
git add file.txt
```

- Добавим обновленный *stage* и изменим сообщение:

```
git commit --amend
```

Смотрим историю

Убедитесь, что коммита по-прежнему два и нам действительно удалось изменить последний коммит, не создавая новый:

- `git log`

```
commit 66bca8c1208dae1a067c805fb4ad7d970eaabd89
Author: Artem Starostenko <artemstarostenko65@gmail.com>
Date:   Tue Nov 22 12:23:32 2016 +0300

    Capitals added

commit bf79e90c5a4d5e82171ba6bc81c847e958015846
Author: Artem Starostenko <artemstarostenko65@gmail.com>
Date:   Tue Nov 22 12:17:50 2016 +0300

    My first commit
```

Проверяем изменения в последнем коммите

Используйте `git show`, чтобы посмотреть изменения:

```
commit 66bca8c1208dae1a067c805fb4ad7d970eaabd89
Author: Artem Starostenko <artemstarostenko65@gmail.com>
Date:   Tue Nov 22 12:23:32 2016 +0300

    Capitals added

diff --git a/file.txt b/file.txt
index ee2ae0c..ddc9cc9 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1,3 @@
 My first line in project
+London is a capital of GB
+Moscow is a capital of Russia
```

Добавим Францию

Добавьте в конец файла `file.txt` строку "*Paris is the capital of France*" и зафиксируйте эти изменения, сделав коммит. В сообщении коммита укажите "*Another capital was added*".

Используйте команды `git log` и `git show`, чтобы проверить сделанный коммит.

Отменим второй коммит

- Найдите hash второго коммита из `git log`
- Используйте команду `git revert 1f829c0` чтобы отменить коммит.
  - ℹ️ Используйте первые символы своего хеша (рекомендуется указывать первые 7 символов)
  - Также можно воспользоваться `HEAD~1`
- Если все правильно, то должен возникнуть конфликт

Решаем конфликт

- Смотрим, где у нас произошел конфликт командой `git status`
- Смотрим конфликтующие строки командой `cat file.txt`

```
My first line in project
<<<<<<< HEAD
London is the capital of GB
Moscow is the capital of Russia
Paris is the capital of France
=======
>>>>>>> parent of 1f829c0... Capital added
```

Git помечает специальными маркерами строки, в которых произошел конфликт

Какие изменения относятся к отменяемому коммиту?

Посмотрим, какие строки нам нужно удалить при решении конфликта, чтобы отменить изменения второго коммита.

- Сделаем `git show HEAD~1`

```
@@ -1 +1,3 @@
 My first line in project
+London is the capital of GB
+Moscow is the capital of Russia
```

Решение конфликта

1. Отменить изменения коммита, удалив строки *London...* и *Moscow*

```
My first line in project
<<<<<<< HEAD
Paris is the capital of France
=======
>>>>>>> parent of 1f829c0... Capital added
```

2. Привести содержимое файла в нужное состояние, удалив маркеры Git

```
My first line in project
Paris is the capital of France
```

Продолжаем отменять коммит

1. Сообщим Git, что мы разрешили конфликт - добавим файл в *stage* командой `git add file.txt`
2. Продолжим процесс отмены коммита - `git revert --continue`
3. Проверим, что создался новый коммит - `git log`
4. Посмотрим, что поменялось в последнем коммите - `git show`

Вопрос на самопроверку

❓ Почему в коммите, который мы отменяем, было добавлено две строчки, а конфликт распространился на три?

Работаем с .gitignore

1. Создайте два файла с одинаковым расширением, например так:

```
echo "this is a garbage file" | tee garbage1.tmp garbage2.tmp
```

2. Проверьте, что Git видит эти файлы командой `git status`
3. Создайте файл `.gitignore` и укажите в нем, чтобы Git игнорировал все файлы с расширением `.tmp`, так как мы не хотим их отслеживать
4. Снова сделайте команду `git status` - файлы с расширением `.tmp` должны исчезнуть из вывода
5. Закоммитьте `.gitignore`

Задание на понимание Index

Добейтесь, чтобы в выводе команды `git status` файл `file.txt` присутствовал в двух секциях одновременно: *Changes to be committed* и *Changes not staged for commit.*

Сравните вывод команд `git diff` и `git diff --cached`

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   file.txt

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:   file.txt
```

Продолжение работы

Результат нужно закоммитить, чтобы рабочая директория была чистой:

```
$ git status
On branch master
nothing to commit, working tree clean
```

Работа с ветками

- Создание и удаление веток
- Мердж веток
- Перемещение (rebase) веток
- Клонирование удаленного репозитория
- Pull и Push в удаленный репозиторий

Создадим ветку

1. Создайте ветку с именем **first** - `git branch first`
2. Посмотрите список локальных веток - `git branch` (звездочкой будет помечена текущая ветка)

```
$ git branch
  first
* master
```

3. Переключитесь на ветку **first** - `git checkout first`

Создадим еще одну ветку

1. Создадим и переключимся на ветку одной командой (сравните, как было с веткой **first**) - `git checkout -b second`
2. Снова проверим текущую ветку, но теперь командой `git status`
3. И снова посмотрим список локальных веток - `git branch`

Изменим ветку second

- Создайте новый файл, например так:

```
$ echo "Let's change the branch" > branch.txt
```

- Закоммитьте изменения с сообщением "*Branch was changed*"

Сделаем коммит на первой ветке

Переключитесь на ветку **first**, создайте файл `first.txt` с любым содержимым и закоммитьте изменения.

> Постарайтесь сделать самостоятельно. Ответы даны на следующем слайде.

Подсказка

Надо переключиться, создать файл и закоммитить изменения:

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

$ echo "And this branch too" > first.txt

$ git add first.txt

$ git commit -m "First branch changed"
[first 2e54f37] First branch changed
 1 file changed, 1 insertion(+)
 create mode 100644 first.txt
```

Смотрим результат

Посмотрим граф изменений - `git log --all --decorate --oneline --graph`

Создайте *alias* для данной команды, она нам еще пригодится:

```
$ git config --global alias.g 'log --all --decorate --oneline --graph'
```

```
$ git g

* 2e54f37 (HEAD -> first) First branch changed
| * f279210 (second) Branch changed
|/
* 17d8431 (master) .gitignore added
* f9e503c Revert "Capitals added"
* de140fc France added
* 66bca8c Capitals added
* bf79e90 My first commit
```

Смерджим ветки

- Перейдем в ветку **master** и проверим наличие файлов:

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

$ ls
```

- Выполним слияние ветки **first** (обратите внимание - у нас появился файл `first.txt`)

```
$ git merge first
Updating 17d8431..2e54f37
Fast-forward
first.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 first.txt

$ ls
  first.txt
```

Вопрос по слиянию

Заметьте, что при слиянии ***не*** *создалось дополнительного коммита*, переместился только указатель ветки **master**.

❓ Как называется данный тип слияния?

```
* 2e54f37 (HEAD -> master, first) First branch changed
| * f279210 (second) Branch changed
|/
* 17d8431 .gitignore added
* f9e503c Revert "Capitals added"
* de140fc France added
* 66bca8c Capitals added
* bf79e90 My first commit
```

😉 Ответ есть в лекции...

Сделаем rebase ветки second

- Перейдем на ветку **second**:

```
$ git checkout second

Switched to branch 'second'
```

- Выполним ребейз:

```
$ git rebase master

First, rewinding head to replay your work on top of it...
Applying: Branch changed
```

Ветка second поменяла свое основание

Что получилось в итоге:

```
$ git g

* b77f3f6 (HEAD -> second) Branch changed
* 2e54f37 (master, first) First branch changed
* 17d8431 .gitignore added
* f9e503c Revert "Capitals added"
* de140fc France added
* 66bca8c Capitals added
* bf79e90 My first commit
```

- Линейная история
- Ветки **master** и **first** совпадают
- Ветка **second** на 1 коммит опережает **master**

Удаляем ветку

- Удалите ветку **first**:

```
$ git branch -d first

Deleted branch first (was 2e54f37).
```

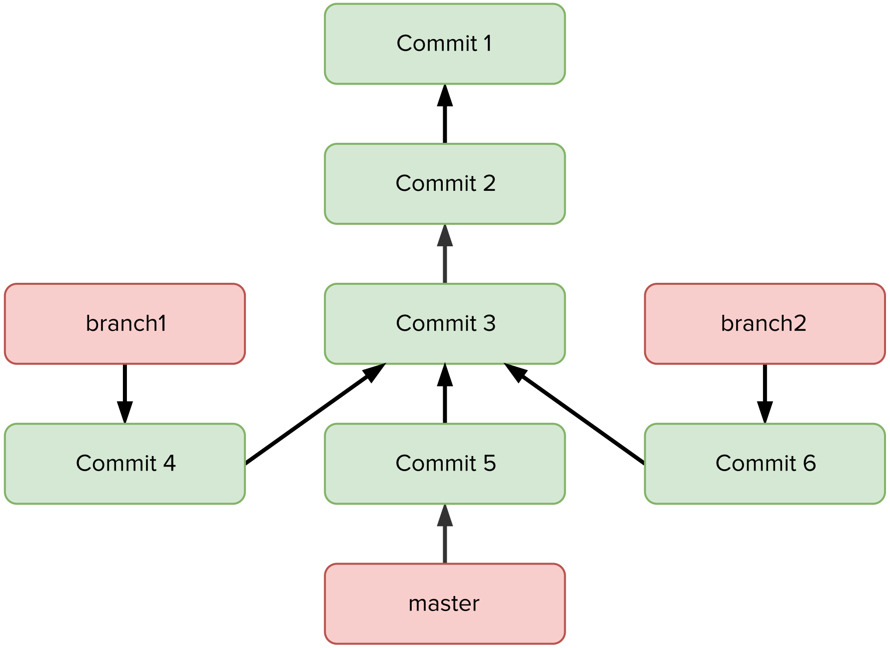
- Проверьте результат:

```
$ git branch

  master
* second
```

Задание

- Создайте новую директорию `task-branches`
- В ней создайте новый репозиторий Git
- Воссоздайте следующую схему коммитов и веток:



Регистрация на GitHub

Зарегистрируйтесь на GitHub

Убедитесь, что поля с именем, фамилией и адресом электронной почты заполнены

❗ Важно! Должно совпадать с именем, фамилией и электронной почтой настроенными в `git` (консольной утилите)

ℹ️ Рекомендуется подключить двухфакторную аутентификацию, чтобы обезопасить ваш аккаунт. Добавить двухфакторную аутентификацию можно, перейдя в **Settings** –> **Security**

Регистрация на GitHub

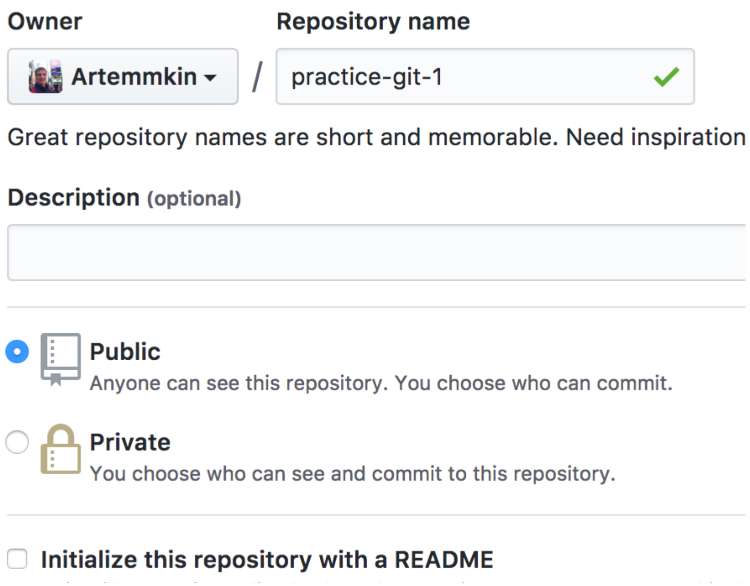
Добавьте SSH-ключ для своего аккаунта, чтобы можно было работать с удаленным репозиторием через SSH.

Инструкции по созданию пары ключей для вашей ОС можно посмотреть здесь.

Инструкции по добавлению публичного ключа для своего аккаунта можно посмотреть здесь.

Скопируйте свои репозитории на GitHub

Создайте пустой GitHub-репозиторий **practice-git-1** для нашего первого локального репозитория (`practice-git`).



Скопируйте свои репозитории на GitHub

Перейдем в папку practice-git, где выполняли первые задания, и настроим информацию об удаленном репозитории:

```
$  git remote add origin git@github.com:<UserName>/practice-git-1.git
```

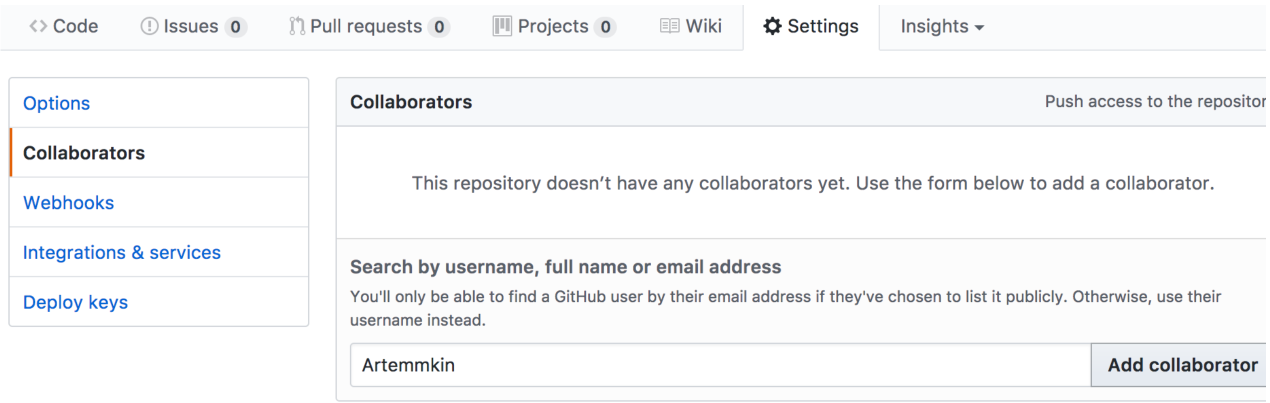
Пушим все ветки локального репозитория в удаленный:

```
$ git push origin -u --all
```

Опция -u позволяет задать для локальной ветки дефолтную удаленную ветку для синхронизации, так что команды git pull/push будут работать на данной локальной ветке без дополнительных аргументов.

Скопируйте свои репозитории на GitHub

- Создайте GitHub репозиторий `practice-git-2` для задания с ветками (`task-branches`).
- Запушьте туда изменения по аналогии с предыдущим репозиторием.
- Добавьте любого коллегу из чата в *Collaborators*. Для этого зайдите в настройки репозитория:

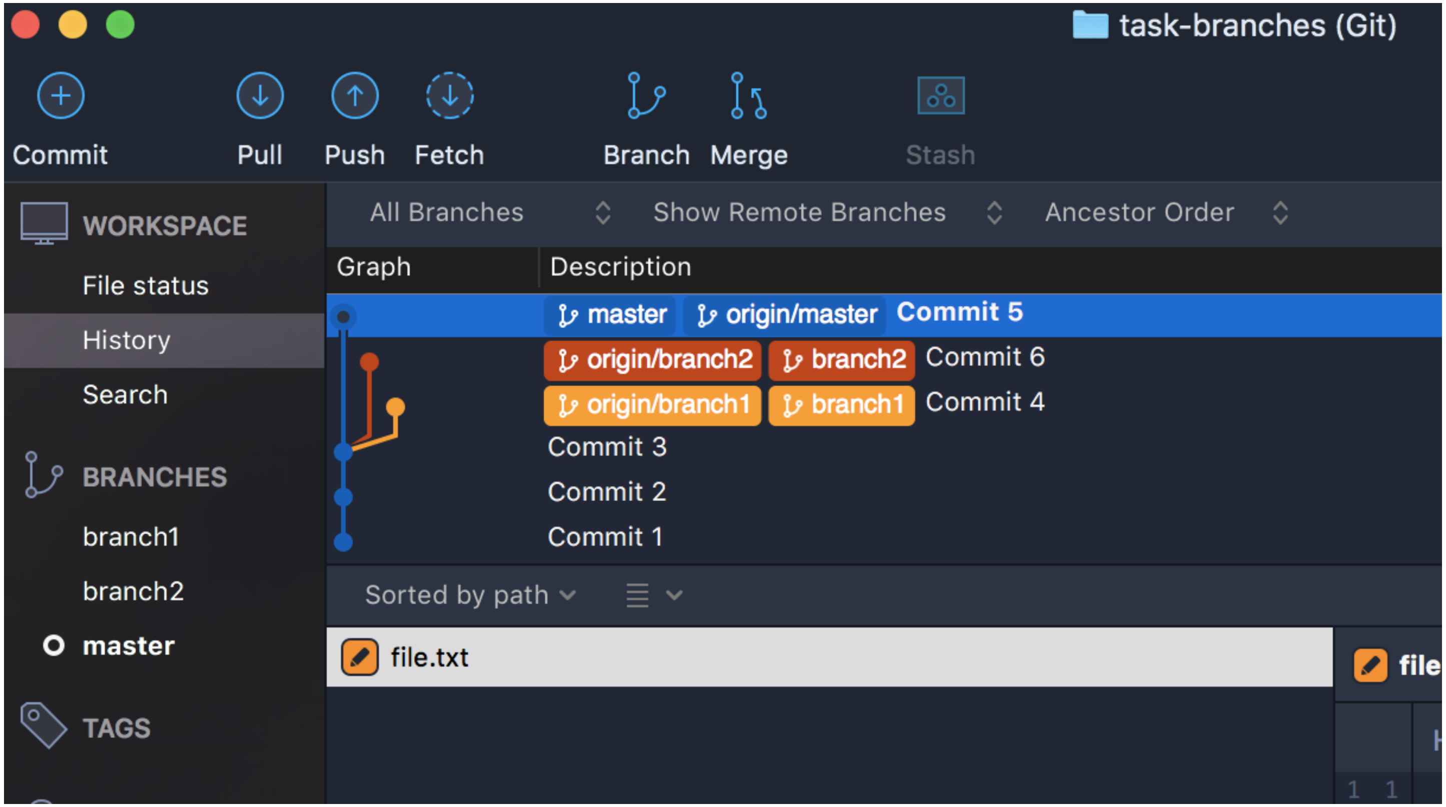


SourceTree

Если вы используете другой графический клиент и описываемой функциональности в нем нет, то можете пропустить часть задания связанной с графическими клиентами. Или сделать задания по аналогии.

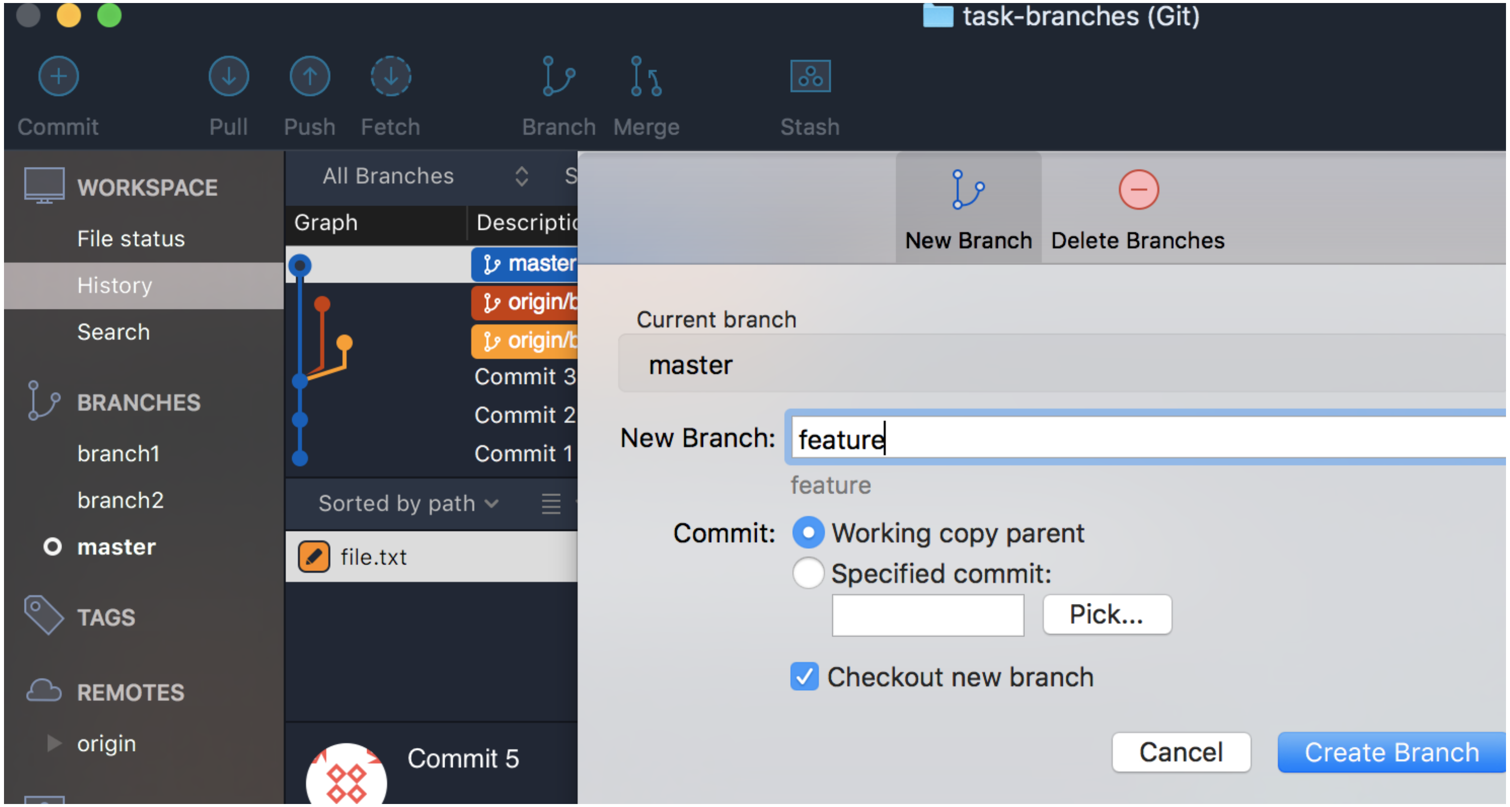
SourceTree

Добавляем проект task-branches в SourceTree. Два раза кликаем на ветку master, чтобы в нее перейти:



Создаем ветку feature

Нажмите кнопку **Branch**, чтобы создать новую ветку:



Проверим

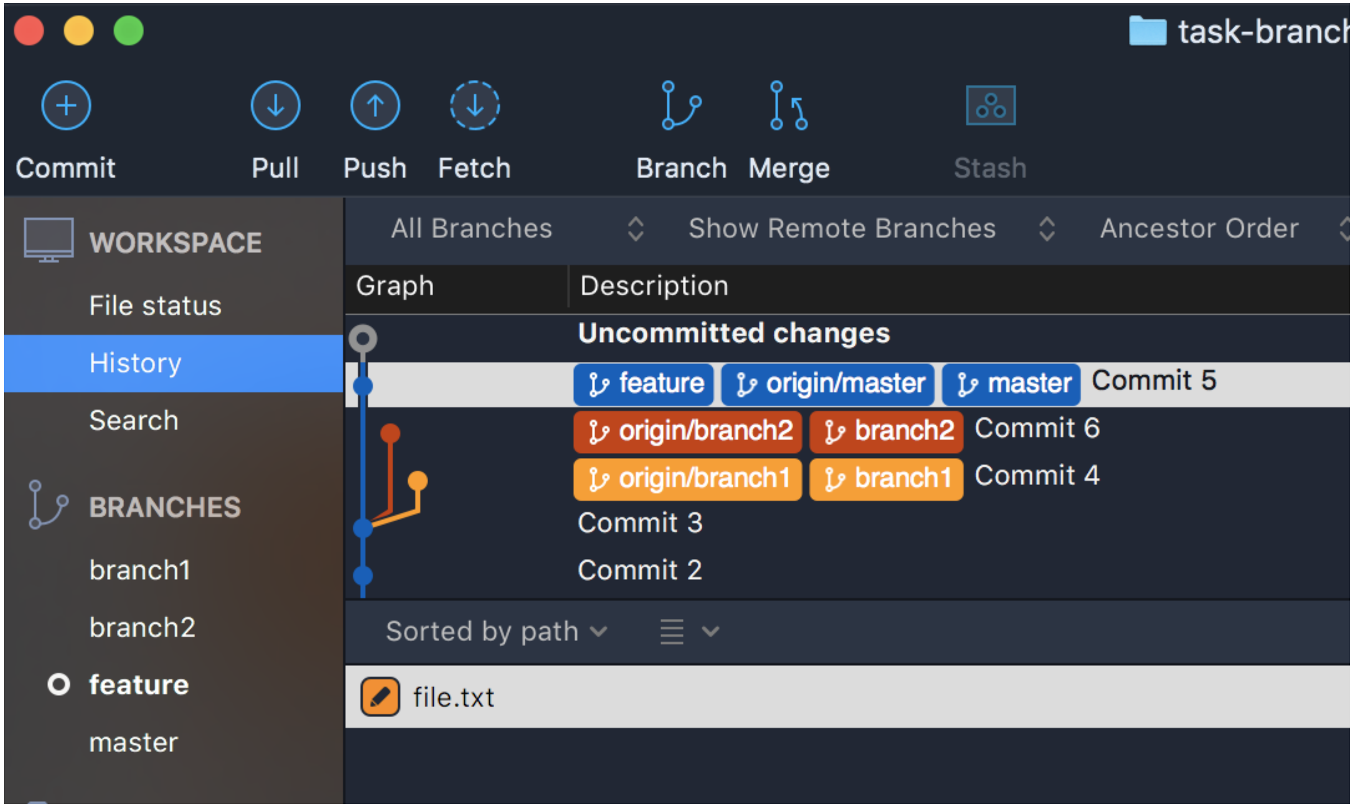
Проверяем в терминале, что ветка создалась:

```
$ git branch

  branch1
  branch2
* feature
  master
```

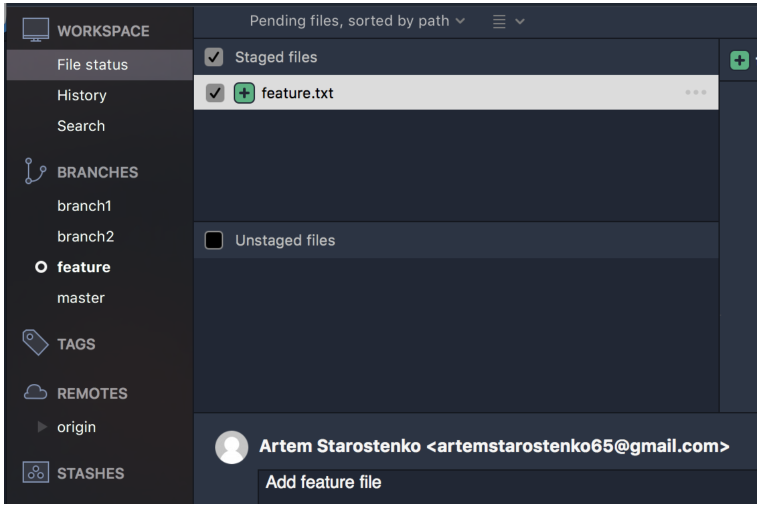
Делаем коммит из SourceTree

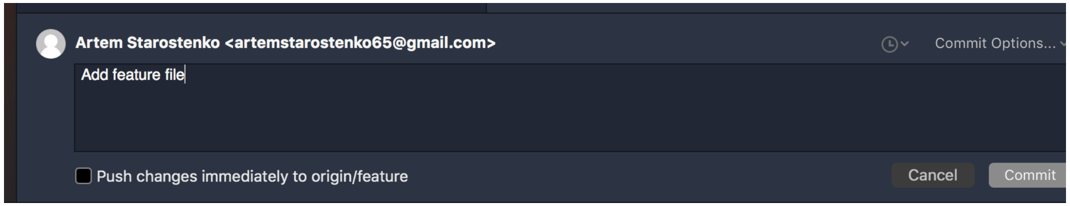
Создайте пустой файл `feature.txt`. В SourceTree отобразится информация о незакомиченных изменениях. Нажмите **Commit**, чтобы сделать коммит.



Делаем коммит из SourceTree

Ставим галку напротив файла `feature.txt` в секции *Unstaged* - файл будет переведен в *Stage*. И нажмите **Commit** под сообщением:





Добавление изменений по частям

Перейдите на ветку `feature`, кликнув по ней два раза в SourceTree. В терминале вы должны видеть, с какой веткой вы работаете. Для этого пользователям Linux и Mac можно воспользоваться темами oh my zsh.

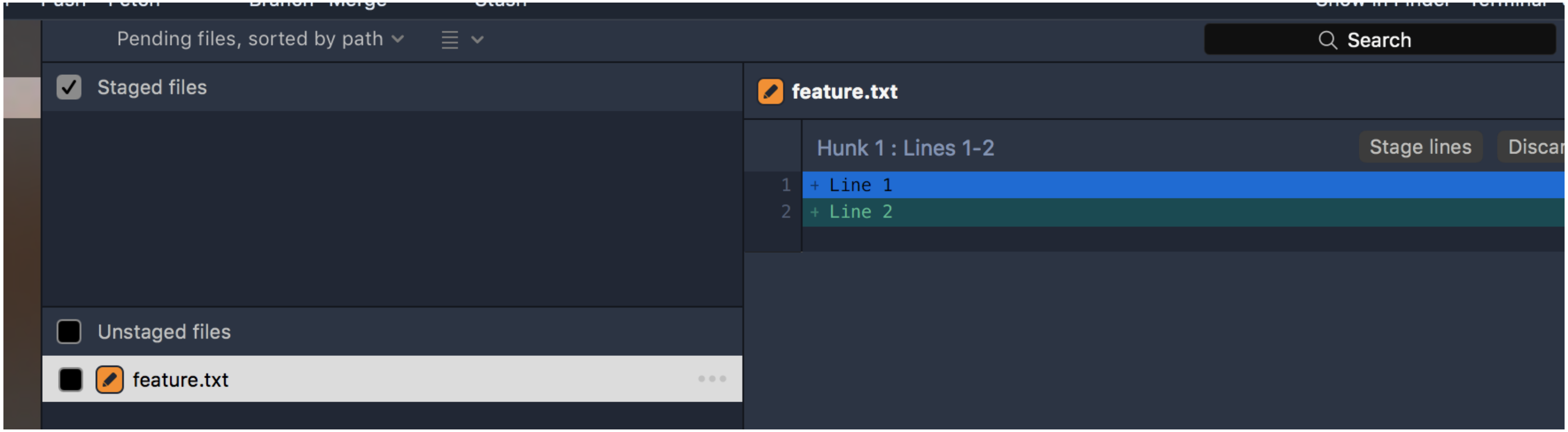
```
user@laptop:~/task-branches [feature]$
```

Добавьте в файл `feature.txt` две любые строки. Например:

```
$ echo "Line 1" >> feature.txt
$ echo "Line 2" >> feature.txt
```

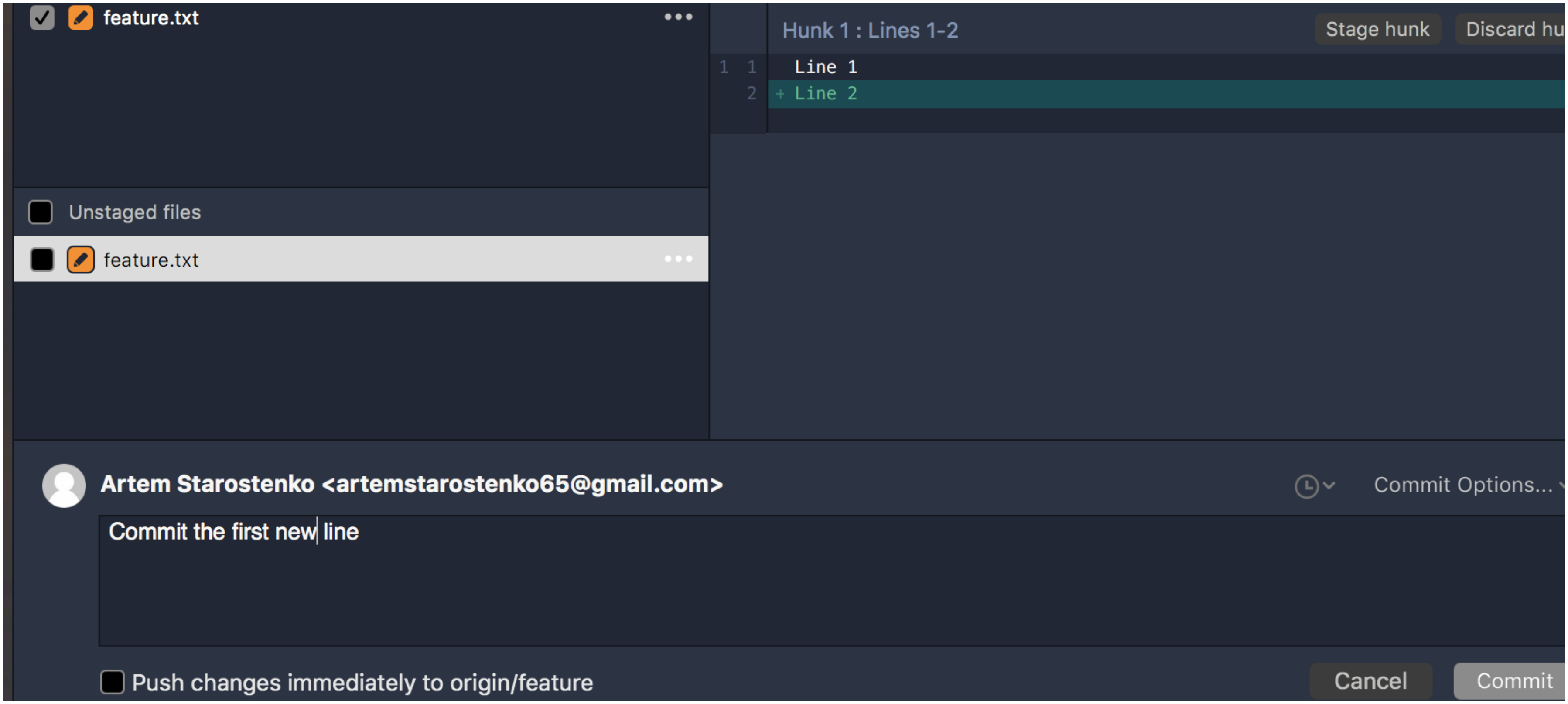
Добавление изменений по частям

Нажмите **Commit** в SourceTree. Добавьте в *Stage* первую добавленную строку. Для этого кликните на нее и нажмите **Stage lines**:



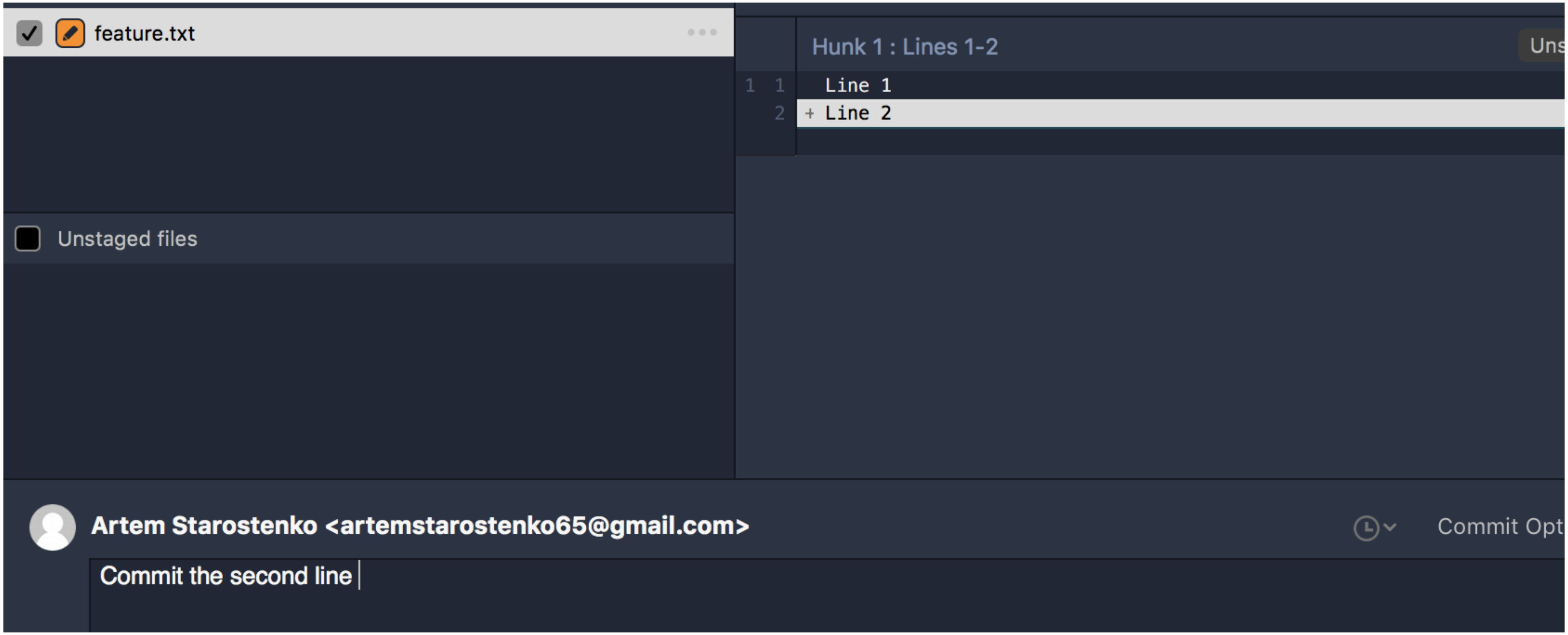
Добавление изменений по частям

Напишите сообщение, что была закомичена первая строка и сделайте коммит:



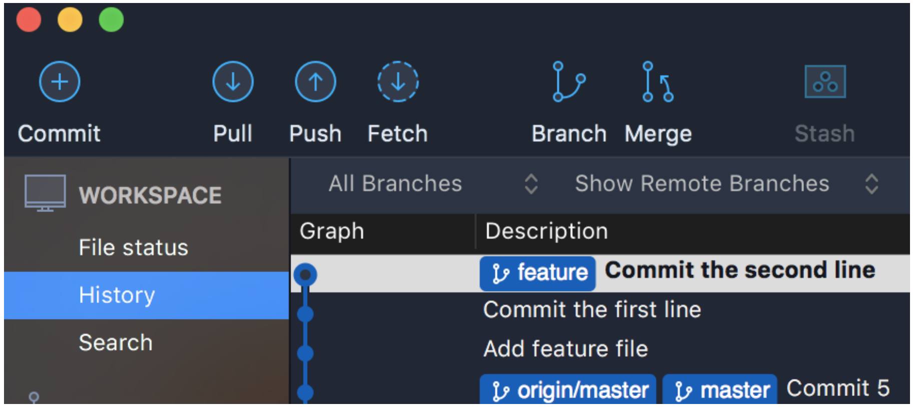
Добавление изменений по частям

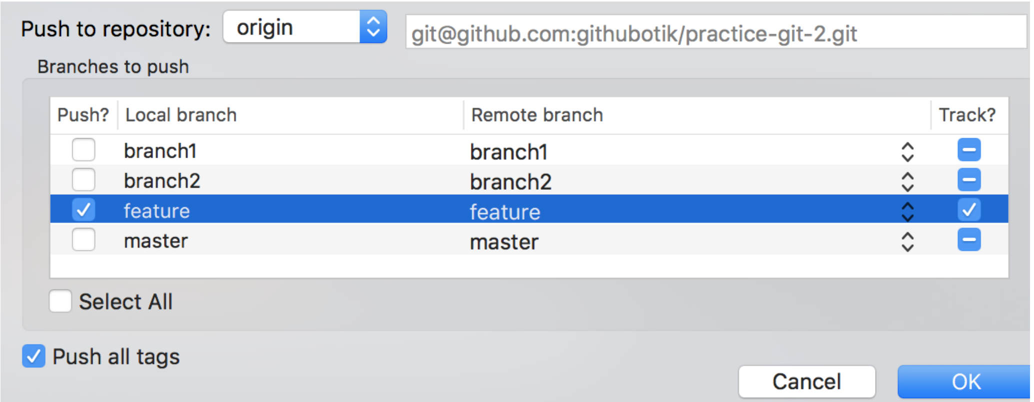
Теперь сделайте по аналогии коммит для второй добавленной строки.



Создание PR

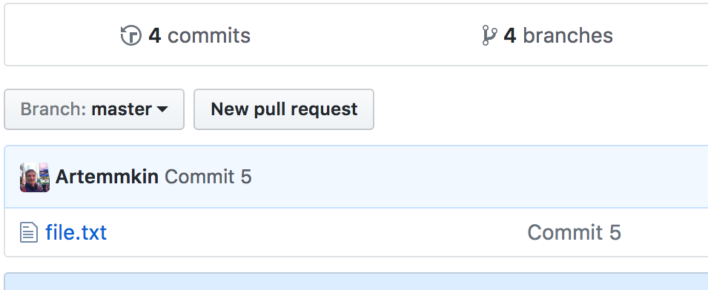
Сделайте push ветки `feature` в удаленный репозиторий на GitHub (нажав на кнопку **Push**):

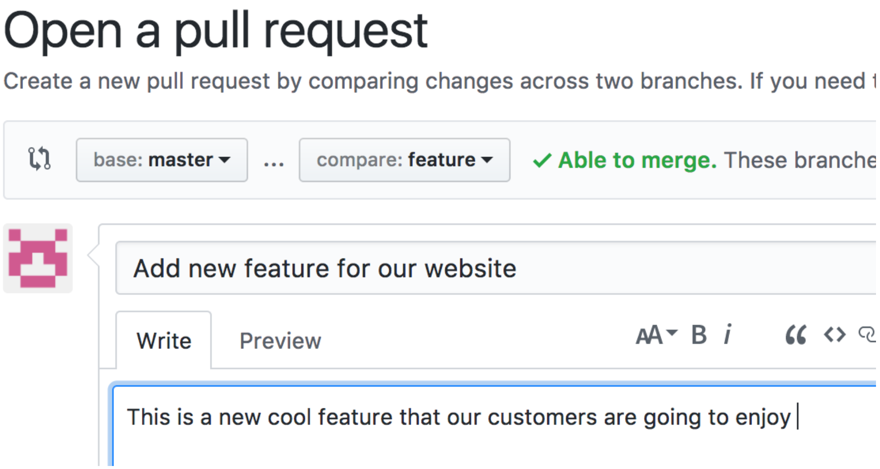




Создание PR

Перейдите на страницу GitHub вашего репозитория `practice-git-2`. Создайте новый запрос на мердж ветки **feature** в **master**. Не забудь добавить Reviewers (см. след. слайд).





Создание PR

Добавьте в Reviewers коллегу, чтобы попросить его посмотреть ваши наработки. Попросите еще кого-нибудь из нашей учебной группы сделать ревью. Вы, в свою очередь, можете сделать ревью (добавить комментарии) их наработок. После получения одобрений хотя бы от одного из Reviewers (необязательно ждать преподавателя) **сделайте merge**.