

# Работа с системой контроля версий

Рассмотрим работу с системой git в три этапа: вариант, как можно создать новый проект с настроенным удаленным репозиторием; какие команды и правила используются при работе с git; как добавить git в уже существующий проект, чтобы не переписывать его с нуля.

## Введение в git. Создание нового проекта

Для того, чтобы создать пустой удаленный репозиторий (пространство проекта на удаленном сервере) для пока что не созданного проекта, заходим на главную страницу <https://github.com> (регистрируемся там, если требуется), после чего на главной нажимаем кнопку “Start a project”.



Откроется страница настроек репозитория. Необходимо указать имя проекта, описание (не обязательно), указать будет он публичным или приватным, а также можно отметить галками те дополнительные файлы конфигураций вместе с которыми следует создать репозиторий. Здесь это файл README.md с описанием проекта (пока что будет пустой), файл .gitignore с настройками тех файлов, которые не будут попадать из вашего локального репозитория (папка проекта на компьютере) на удаленный (GitHub). Как правило это какой-то пользовательский кеш, либо какие-то ваши личные данные, либо любая другая информация, которую вы хотите хранить в папке проекта, но которая не должна быть в вашем удаленном репозитории. Можно также указать лицензию (подробнее об этом можно почитать по ссылке “Learn more” в этом пункте). Выберем здесь .gitignore для

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner \* Repository name \*

SemyonovE / HelloGit ✓

Great repository names are short and memorable. Need inspiration? How about [legendary-giggle](#)?

Description (optional)

This is an app for git tutorial

Public Anyone on the Internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file This is where you can write a long description for your project. [Learn more](#).

Add .gitignore Choose which files not to track from a list of templates. [Learn more](#).

.gitignore template: Swift ▾

Choose a license A license tells others what they can and can't do with your code. [Learn more](#).

License: MIT License ▾

This will set `main` as the default branch. Change the default name in your [settings](#).

[Create repository](#)

языка Swift (в выпадающем списке), остальное по желанию и далее нажмем кнопку “Create repository”.

После нас перебросят на страницу проекта в GitHub. Мы увидим, что уже есть 1 коммит (запись в истории изменений проекта) с файлами, добавленными при инициализации репозитория, видим что есть у нас ветка master (или main, суть у них одна - главная ветка).

The screenshot shows a GitHub repository page for 'SemyonovE/HelloGit'. At the top, there are buttons for 'main' (selected), '1 branch', '0 tags', 'Go to file', 'Add file', and a green 'Code' button. Below this is a table of files: '.gitignore' (Initial commit, now), 'LICENSE' (Initial commit, now), and 'README.md' (Initial commit, now). To the right, sections for 'About', 'Releases', and 'Packages' are visible, each with a 'Create a new release' or 'Publish your first package' link.

Если нажать на яркую зелёную кнопку "Code", откроется окно в котором мы можем выбрать то, как хотим получить созданный удаленно проект на своем компьютере, то есть стянуть себе локальную копию. Чтобы у нас осталась привязка к гиту, а не просто мы загрузили файлы проекта (Download ZIP), надо скопировать адрес нашего удаленного репозитория в текстовом поле (кнопка справа от ссылки). В моём случае это ссылка <https://github.com/SemyonovE>HelloGit.git>

Теперь нужно открыть на компьютере терминал (или другое приложение, которое вы используете вместо терминала, в моих примерах это iTerm2) и перейти внутри терминала в ту папку, куда вы хотите стянуть (загрузить) свой проект. Перейти можно используя команду 'cd' и указать вручную тот путь, куда нужно перейти, относительно текущей папки (по умолчанию мы находимся в папке нашего пользователя). Либо можно просто написать cd и перетащить в терминал необходимую папку (её путь пропишется автоматически и будет абсолютным, то есть прямо из корня вашего диска). Останется только нажать Enter, чтобы увидеть,

Last login: Wed Mar 31 19:39:25 on ttys006  
apple ~ > cd Desktop  
apple ~/Desktop > Identity and

что наш путь сменился (перед курсором терминала). Я стяну удаленный проект на рабочий стол.

Теперь, когда мы в нужной директории, вспоминаем про скопированный нами путь к удаленному репозиторию и набираем команду 'git clone ВАША\_ССЫЛКА'. В моём случае это команда:

**git clone https://github.com/SemyonovE>HelloGit.git**

Если до этого вы не работали с гитом, то вам предложат ввести свой логин с сайта GitHub, а затем и пароль. Если при вводе пароля вы не увидите вводимые

```
apple ~/Desktop > git clone https://github.com/SemyonovE>HelloGit.git
Cloning into 'HelloGit'...
remote: Enumerating objects: 5, done.  Name OnboardingScreen.swift
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.  default - Swift Source
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.  Location Relative to Group
apple ~/Desktop > OnboardingScreen.swift
```

вами символы, это нормально. Просто введите пароль и нажмите Enter. Он просто может не отображаться, чтобы его никто не увидел. Это стандартная процедура при вводе пароля в

терминале. После выполнения команды мы должны увидеть, что проект с удаленного репозитория был загружен.

Осталось перейти в саму папку проекта, чтобы уже там работать с системой

```
apple ~ ~/Desktop > cd HelloGit          git дальше. Пишем 'cd  
apple ~ ~/De>HelloGit ⚡ main > █ ИМЯ_ВАШЕГО_ПРОЕКТА' и нажимаем  
Enter, в моем случае:
```

## cd HelloGit

Для того, чтобы убедиться что всё как надо (проект чистый, без изменений и

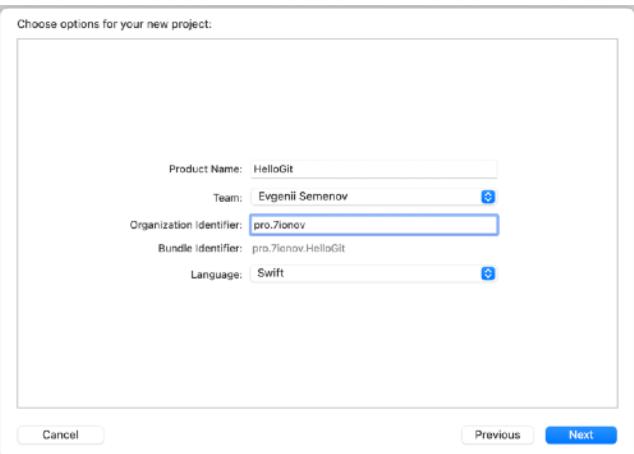
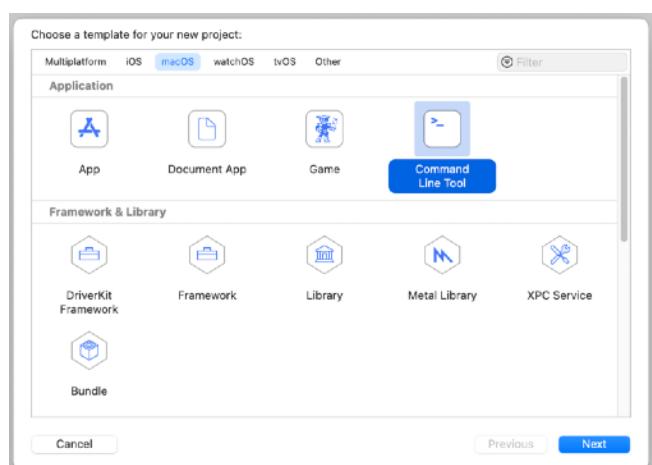
```
apple ~ ~/De>HelloGit ⚡ main > git status  
On branch main  
Your branch is up to date with 'origin/main'.  
nothing to commit, working tree clean
```

сейчас мы находимся на ветке master) мы можем вызвать команду 'git status' (статус/состояние нашего локального репозитория)

## git status

Теперь у нас на компьютере есть копия удаленного проекта. Для того, чтобы

наполнить этот проект какими-то данными, нам достаточно их разместить в папке этого проекта. Для начала создадим наш тестовый проект. В Xcode выбираем File>New>Project и создаем приложение командной строки под MacOs (для изучения возможностей гит нам этого варианта будет вполне достаточно).



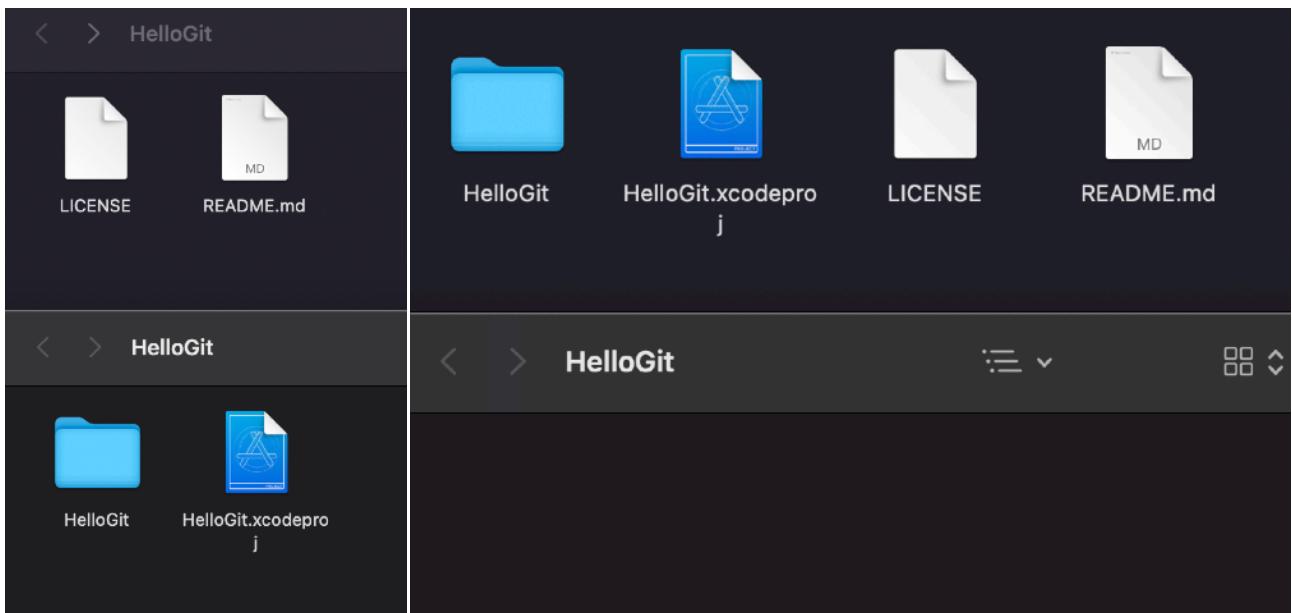
Указываем название нашего проекта. Лучше, если оно будет совпадать с именем проекта, который мы создали на GitHub: но, если имя будет отличаться, не страшно. Указываем команду (аккаунт разработчика), идентификатор (любой уникальный набор разрешенных символов, обычно в виде обратного url-адреса, пока мы не собираемся загружать наш проект в магазин приложений, он может быть совершенно любым) и сохраняем. Место сохранения xcode-проекта, лучше указать отличное от директории, содержащей наш git-проект (проект в локальном репозитории) с таким же именем, иначе эта папка будет перезаписана. В моём случае git-проект находится на рабочем столе, поэтому я могу сохранить свой xcode-проект, скажем, в папку "Загрузки".

После сохранения xcode-проекта мы перенесем всё его содержимое в папку git-проекта, поэтому Xcode сейчас можно закрыть. Открываем в Finder две папки. Сверху у меня папка git-проекта, то место, куда мне необходимо перенести только

что созданный xcode-проект, а ниже папка самим этим проектом. Выделяем все файлы в папке xcode-проекта и переносим в папку git-проекта.

Было:

Стало:



Теперь можно удалить папку xcode-проекта (она теперь пустая) и

запустить .xcodeproj-файл уже из папки git-проекта (так как теперь у нас нет двух папок проекта для удобства буду дальше называть его просто проект). Убедимся, что мы ничего не сломали, наш проект запускается в Xcode и работает. Для этого достаточно собрать (cmd+B) или запустить (cmd+R) проект. Видим, что структура файлов нашего

проекта не сломалась, ошибок нет, значит мы всё сделали верно. Теперь давайте вернёмся к терминалу и посмотрим, что изменилось. Снова используем команду проверки состояния проекта:

## git status

Видим информацию о том, что в проекте добавились две сущности, давайте закоммитим (создадим commit, то есть сохраним в истории) эти изменения, чтобы они никуда не пропали. Сейчас файлы выделены красным, потому что просто поместить их в папку проекта мало, нам нужно также подготовить их для коммита (добавить в индекс, или иначе говоря отслеживать). В терминале мы видим подсказку, что для

```
apple ~ ~/De>HelloGit M P main > git status
On branch main
Your branch is up to date with 'origin/main'.

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

nothing added to commit but untracked files present (use "git add" to track)
```

этого нужно использовать команду 'git add <file>...' и указать те файлы, что мы ходим подготовить для коммита. В самом деле мы можем вызвать команду

```
git add HelloGit.xcodeproj HelloGit/
```

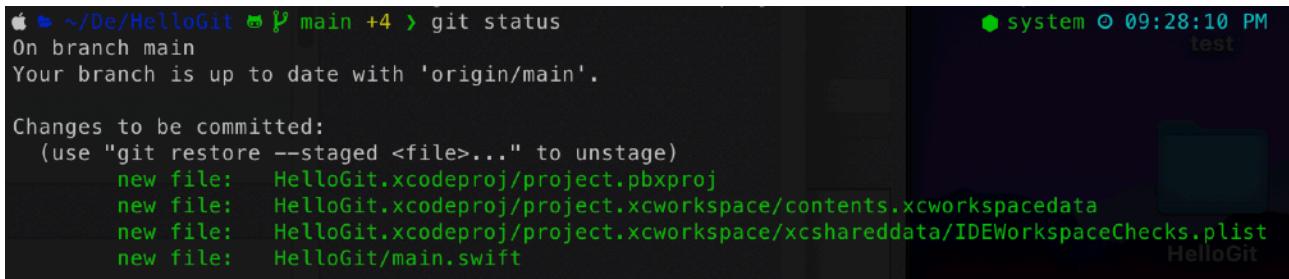
указав таким образом все добавленные файлы через пробел, но если таких файлов будет много, можно просто использовать команду

```
git add .
```

Тогда все новые или измененные файлы будут подготовлены к коммиту. Теперь, если вызвать

```
git status
```

мы увидим все те файлы, что были подготовлены и они уже будут написаны зеленым цветом (у вас терминал может выводить без выделения цветом).



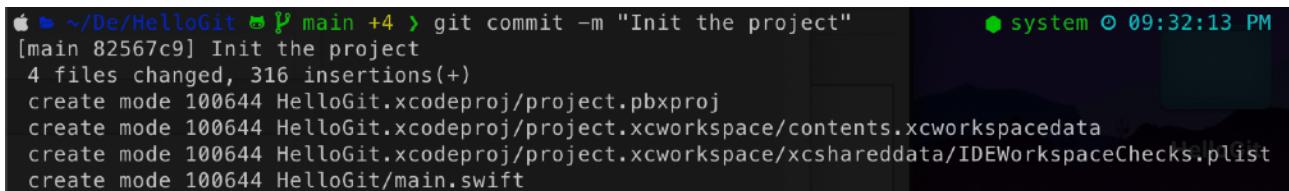
```
apple ~ ~/De/HelloGit main +4 > git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   HelloGit.xcodeproj/project.pbxproj
    new file:   HelloGit.xcodeproj/project.xcworkspace/contents.xcworkspacedata
    new file:   HelloGit.xcodeproj/project.xcworkspace/xcshareddata/IDEWorkspaceChecks.plist
    new file:   HelloGit/main.swift
```

Чтобы нам сделать первый коммит и сохранить все изменения в истории master ветки нужно выполнить команду 'git commit -m "ЧТО\_ИЗМЕНИЛОСЬ"'. В моём случае:

```
git commit -m "Init the project"
```

Здесь в кавычках указывается описание того, что именно было добавлено/изменено.



```
apple ~ ~/De/HelloGit main +4 > git commit -m "Init the project"
[master 82567c9] Init the project
4 files changed, 316 insertions(+)
create mode 100644 HelloGit.xcodeproj/project.pbxproj
create mode 100644 HelloGit.xcodeproj/project.xcworkspace/contents.xcworkspacedata
create mode 100644 HelloGit.xcodeproj/project.xcworkspace/xcshareddata/IDEWorkspaceChecks.plist
create mode 100644 HelloGit/main.swift
```

Теперь, если снова вызвать команду

```
git status
```

мы увидим, что у нас есть 1 изменение (1 коммит), которое есть у нас локально, но мы пока не отправили его в наш удаленный репозиторий. Чтобы это сделать (опять же мы видим подсказку), нужно использовать команду

```
git push
```

Здесь также, возможно, потребуется ввести логин/пароль от сайта GitHub.

```
apple ~ ~/De/HelloGit $ git push
Username for 'https://github.com': SemyonovE
Password for 'https://SemyonovE@github.com':
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (10/10), 3.15 KiB | 3.15 MiB/s, done.
Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/SemyonovE>HelloGit.git
  6ff135f..82567c9  main -> main
```

Изменения успешно отправлены. Теперь, если вернуться на сайт <https://github.com/> в папку проекта (или обновить страницу, если вы её не закрывали), мы увидим изменения в проекте.

This is an app for git tutorial

Readme

MIT License

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

Swift 100.0%

Теперь перейдём к самому проекту и посмотрим, какие возможности нам предоставляет Git. Работу будем проводить на тестовом проекте в котором рассмотрим базовый набор команд, необходимые для командной разработки. Возможные внештатные ситуации на проекте рассмотрены не будут.

Смоделируем ситуацию, у нас на тестовом проекте будет два разработчика, мы будем играть роли обоих. Так у нас получится увидеть преимущество git-системы при работе в команде. Для начала нам необходимо выстроить git-flow, то есть то, по каким правилам мы и другие разработчики должны будут работать в git-системе, чтобы всё было единообразно и по правилам. У нас сейчас есть ветка master (или main), обычно её используют для хранения конечных, протестированных версий нашего кода, для разработки же нам нужно создать дополнительную ветку. Там мы сможем менять код как угодно, пока не наведем в нем порядок, чтобы весь код можно было отправить в master для актуализации текущей версии проекта. Чтобы создать ветку в нашем репозитории можно выполнить команду 'git branch ИМЯ\_ВЕТКИ'. Пока что повремените с выполнением этой команды для своего проекта, дальше будет пояснение почему. В моем случае подобная команда бы имела вид

## git branch develop

В этом случае мы бы создали ветку, но чтобы вся наша работа была в дальнейшем именно в ней нам пришлось бы также на нее переключиться (сменить текущую ветку разработки, сейчас это master). Для этого достаточно вызвать команду

'git checkout ИМЯ\_ВЕТКИ'. Или для моего примера это была бы команда

## git checkout develop

Но, если мы хотим создать ветку и сразу переключиться на неё, можно использовать команду 'git checkout -b ИМЯ\_ВЕТКИ'. Давайте используем подобную команду для тестового проекта, а именно

```
git checkout -b develop
```

Мы увидим сообщение, что ветка создана и мы также переключились на неё.

```
apple ➜ ~/De/HelloGit git [main] > git checkout -b develop
Switched to a new branch 'develop'
```

Чтобы увидеть все наши ветки, можно использовать команду 'git branch -a'. Здесь мы увидим локальные ветки и ветки на удалённом репозитории (про HEAD я расскажу позже).

```
* develop
  main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
(END)
```

Чтобы выйти из режима просмотра, достаточно нажать клавишу 'q' на клавиатуре (важно, чтобы был выбран именно английский язык в качестве языка ввода).

Теперь нашу созданную ветку нужно отправить в удалённый репозиторий, потому что она будет общей для всех разработчиков. Используем команду

```
git push origin develop
```

```
apple ➜ ~/De/HelloGit git [develop] > git push origin develop
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:     https://github.com/SemyonovE>HelloGit/pull/new/develop
remote:
To https://github.com/SemyonovE>HelloGit.git
 * [new branch]      develop -> develop
```

Здесь origin имя нашего удаленного репозитория, оно такое по умолчанию (удаленных репозиториев может быть несколько), а в конце мы указываем какую именно ветку мы хотим отправить. Если не указывать ветку, будет отправляться всё, что есть в нашем локальном репозитории.

Далее наш git-flow будет построен следующим образом:

- Любой разработчик создает себе ветку на некоторую задачу 'feature/ИМЯ\_ЗАДАЧИ' из ветки develop, поработав в этой ветке и доделав задачу он будет мерджить свою ветку (вливать свой код) обратно в develop.

- Когда наступит время проверить весь код, который сейчас в разработке, мы создадим дополнительную ветку `release`. В ней код будет тестироваться независимо от ветки `develop`, в которой разработка может продолжаться без ожидания окончания тестирования. Также здесь мы будем заводить ветки с багами '`bugfix/ИМЯ_БАГА`', если они появились во время разработки.
- Когда код протестирован и все баги закрыты мы будем вливать ветку `release` в ветку `master`. Теперь там будет доступна последняя актуальная версия нашего проекта.

Приступим к работе над самим приложением. Наше приложение будет без какой-то UI части, мы сконцентрируемся именно на системе версионирования нашего кода, а не что там конкретно есть, но тем не менее создадим несколько условно-реальный проект. Нам необходимо сделать приложение, которое будет состоять из 2 экранов (абстрактно, в нашем случае просто классы с выводом информации в консоль):

- Первым экраном будет онбординг (знакомство с приложением), на котором будет заголовок, описание и кнопка запустить приложение.
- Вторым экраном будет главная, на которой просто будет отображаться список некоторых новостей (просто зашьем в приложение несколько вымышленных новостей и будем выводить их в консоль).

Для того, чтобы начать разработку, одному из программистов нужно подготовить базу, каркас нашего проекта, чтобы дальше остальные уже на его основе могли дополнять проект своими фичами. Включаемся в роль первого разработчика и создаём себе отдельную ветку на задачу добавления таких компонентов в наше приложение как `Label` (текстовая метка) и `Button` (кнопка).

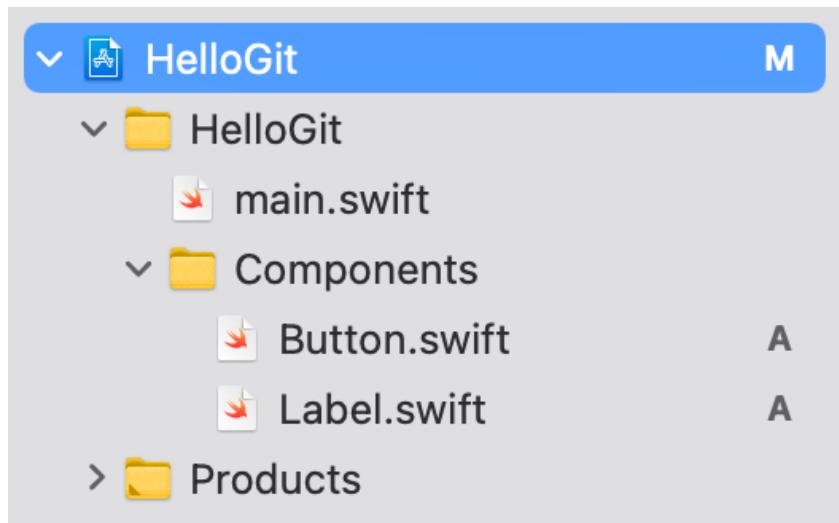
**git checkout -b feature/components**

```
Apple ~ ~/De/HelloGit git ⌘ develop > git checkout -b feature/components
Switched to a new branch 'feature/components'
```

Обратите внимание на наименование ветки, оно состоит из 2 компонент, первый это тип ветки (`feature` - означает какую-то новую функциональность), в нашем случае это будут:

- `feature` для задач из ветки `develop`.
- `bugfix` для багов из ветки `release`.

Теперь, когда ветка создана, разработчик открывает Xcode и делает своё дело, реализует заявленные компоненты. Заодно организует в проекте структуру папок.



```

class Button {

    private(set) var title: String
    private(set) var backgroundColor: String
    private var closure: (() -> Void)?

    init(title: String, backgroundColor: String = "clear") {
        self.title = title
        self.backgroundColor = backgroundColor
    }

    func setTitle(_ text: String) {
        title = text
    }

    func setBackgroundColor(_ color: String) {
        backgroundColor = color
    }

    func addAction(_ closure: @escaping () -> Void) {
        self.closure = closure
    }

    func touch() {
        closure?()
    }
}

class Label {

    private(set) var title: String
    private(set) var textColor: String

    init(title: String, textColor: String = "black") {
        self.title = title
        self.textColor = textColor
    }

    func setTitle(_ text: String) {
        title = text
    }

    func setTextColor(_ color: String) {
        textColor = color
    }
}

```

Два простых класса, которые имитируют текстовое представление и кнопку, которую можно “нажать”, вызвав соответствующий метод.

Теперь первый разработчик (мы) идем в терминал и коммитит эти изменения

```

git status
git add .

```

```
git commit -m "Added button and label components"
```

```

apple ~ ~/De/HelloGit $ feature/components > git status
On branch feature/components
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:  HelloGit/Components/Button.swift
    new file:  HelloGit/Components/Label.swift
  31.03.2021.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:  HelloGit.xcodeproj/project.pbxproj
    modified:  HelloGit/Components/Button.swift
    modified:  HelloGit/Components/Label.swift

apple ~ ~/De/HelloGit $ feature/components +2 !3 > git add .
apple ~ ~/De/HelloGit $ feature/components +3 > git commit -m "Added button and label components"

[feature/components bb2fe40] Added button and label components
 3 files changed, 85 insertions(+)
 create mode 100644 HelloGit/Components/Button.swift
 create mode 100644 HelloGit/Components/Label.swift
apple ~ ~/De/HelloGit $ feature/components > git status
On branch feature/components
nothing to commit, working tree clean

```

Тут разработчик замечает, что забыл избавиться от стандартного кода в файле main.swift

```
print("Hello, World!")
```

Не обязательно делать новый коммит, если мы еще никуда не отправили нашу ветку. Мы

можем сделать дополнение последнего коммита. Для этого удаляем стандартный код и вносим дополнение к предыдущему коммиту с помощью команд

```
git status
git add .
git commit --amend
```

```

Added button and label components

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed Mar 31 22:52:54 2021 +0700
#
# On branch feature/components
# Changes to be committed:
#   modified:  HelloGit.xcodeproj/project.pbxproj
#   new file:  HelloGit/Components/Button.swift
#   31.03.2021.
#   modified:  HelloGit/Components/Label.swift
#   modified:  HelloGit/main.swift
#
~
```

В таком случае у нас откроется информация о прошлом коммите, чтобы мы при желании её изменили (скорее всего эта информация открылась в редакторе vim, инструкцию по работе с ним можно найти в интернете). Допустим, мы не будем ничего менять и просто выйдем,

не меняя описание прошлого коммита. Вводим с клавиатуры ':q' (если открылся vim) и нажимаем Enter.

Мы успешно изменили предыдущий коммит, теперь нужно отправить эти изменения на удалённый репозиторий, чтобы наш код был проверен и если с ним всё хорошо, чтобы он был влит в develop. Отправим нашу ветку с помощью уже известной команды

```

apple ~ ~/De/HelloGit $ feature/components !1 > git status
On branch feature/components
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:  HelloGit/main.swift

no changes added to commit (use "git add" and/or "git commit -a")
apple ~ ~/De/HelloGit $ feature/components !1 > git add .
apple ~ ~/De/HelloGit $ feature/components +1 > git commit --amend
[feature/components 30774a9] Added button and label components
Date: Wed Mar 31 22:52:54 2021 +0700
 4 files changed, 85 insertions(+), 3 deletions(-)
 create mode 100644 HelloGit/Components/Button.swift
 create mode 100644 HelloGit/Components/Label.swift

```

### git push origin feature/components

```

apple ~ ~/De/HelloGit $ feature/components > git push origin feature/components
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 1.36 KiB | 1.36 MiB/s, done.
Total 9 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
remote:
remote: Create a pull request for 'feature/components' on GitHub by visiting:
remote:     https://github.com/SemyonovE/HelloGit/pull/new/feature/components
remote:
To https://github.com/SemyonovE/HelloGit.git
 * [new branch]      feature/components -> feature/components

```

## Перейдём на сайт GitHub в наш проект

The screenshot shows a GitHub repository page for a project named 'HelloGit'. At the top, there's a green banner indicating 'feature/components had recent pushes 1 minute ago' and a 'Compare & pull request' button. Below the banner, the repository details show 'main' branch, 3 branches total, and 0 tags. A commit from 'SemyonovE' titled 'Init the project' is listed, along with other initial commits for files like 'HelloGit.xcodeproj', '.gitignore', 'LICENSE', and 'README.md'. The README file contains the text 'This is an app for git tutorial'. On the right side of the page, there are sections for 'About', 'Releases', 'Packages', and 'Languages', all currently empty or showing no activity.

Видим, что совсем недавно была отправлена наша ветка с компонентами и мы можем создать pull request (команду на добавление нашего кода в другую ветку), нажав на кнопку “Compare & pull request”. Либо мы можем сделать это нажав на “branches” и после, напротив нужной ветки уже нажать “New pull request”.

The screenshot shows the 'branches' section of the GitHub repository. It lists three branches: 'main' (the default branch), 'feature/components', and 'develop'. The 'feature/components' branch is currently selected, as indicated by its blue color and the fact that it is the active branch. There are two buttons next to each branch: 'New pull request' and edit/refresh icons. The 'feature/components' branch has 1 commit and the 'develop' branch has 0 commits.

Прежде чем мы создадим ПР (pull request, иногда говорят МР - merge request, это одно и то же, но в разных git-системах называется по разному), нам надо описать, что было сделано более детально (если это необходимо, обычно достаточно описания из коммита в заголовке) и выбрать ту ветку, в которую вы хотим влить (смержить) наш код. Выбираем ветку `develop` в качестве ветки назначения (слева, куда указывает стрелка от нашей текущей ветки).

The screenshot shows the 'Create pull request' dialog. It displays the base branch as 'develop' and the compare branch as 'feature/components'. A green checkmark indicates that the branches are 'Able to merge'. The main text area contains the message 'Added button and label components'. Below the message, there are 'Write' and 'Preview' tabs, a rich text editor toolbar, and a comment input field. At the bottom, there's a note about attaching files and a large 'Create pull request' button.

Ниже будет информация про все наши изменения, в том числе и системные изменения нашего проекта, которые произошли сами собой, их сделал сам Xcode. Нажимаем создать и видим информацию о нашем запросе на вливание кода. Другой разработчик или ревьюер (проверяющий) с проекта теперь займет и посмотрит наши изменения (можно его уведомить, что ему есть что посмотреть,

либо в будущем, на реальных проектах при создании запроса на слияние вы будете указывать ревьюера который будет проверять ваш код).

### Added button and label components #1

The screenshot shows a GitHub pull request page. At the top, a green button says "Open" and a message says "SemyonovE wants to merge 1 commit into develop from feature/components". Below this are tabs for Conversation (0), Commits (1), Checks (0), and Files changed (4). A comment from "SemyonovE" says "commented now" with "No description provided.". Below the comment is a list item "-o Added button and label components" with a timestamp "30774a9". A note below says "Add more commits by pushing to the feature/components branch on SemyonovE/HelloGit." A large green box contains CI status: "Continuous integration has not been set up" (using GitHub Actions and several other apps) and "This branch has no conflicts with the base branch" (merging can be performed automatically). A green button at the bottom of the box says "Merge pull request". Below the box is a comment input field with "Leave a comment" placeholder and "Attach files by dragging & dropping, selecting or pasting them." buttons. At the bottom right are "Close pull request" and "Comment" buttons.

Если с кодом всё хорошо, ревьюер, который проверял (ревьюил) наш код, нажимает кнопку “Merge pull request” и потом “Confirm merge”. После чего наш код попадет в указанную ветку (в данном случае develop).

### Added button and label components #1

The screenshot shows a GitHub pull request page after merging. A purple button says "Merged" and a message says "SemyonovE merged 1 commit into develop from feature/components" with a timestamp "27 seconds ago". Below this are tabs for Conversation (0), Commits (1), Checks (0), and Files changed (4). A comment from "SemyonovE" says "commented 6 minutes ago" with "No description provided.". Below the comment is a list item "-o Added button and label components" with a timestamp "30774a9". At the bottom of the list is a message "-o SemyonovE merged commit e73e82a into develop 27 seconds ago" with a "Revert" button to its right.

Если вдруг что-то пошло не так, у нас есть кнопка “Revert” (в правом нижнем углу), чтобы отменить слияние. Но обычно это скорее внештатная ситуация, чем что-то обычное, чем придется пользоваться.

# Работа над проектом. Базовые команды при работе с git

Возвращаемся в наш терминал, переключаемся на ветку develop и подтягиваем все ее изменения. Сделать это можно с помощью команды

```
git checkout develop  
git pull origin develop
```

```
apple ~ ~/De/HelloGit git [P] feature/components > git checkout develop          ● system  
Switched to branch 'develop'  
apple ~ ~/De/HelloGit git [P] develop > git pull origin develop           ● system  
hint: Pulling without specifying how to reconcile divergent branches is  
hint: discouraged. You can squelch this message by running one of the following  
hint: commands sometime before your next pull:  
hint:  
hint:   git config pull.rebase false  # merge (the default strategy)  
hint:   git config pull.rebase true   # rebase  
hint:   git config pull.ff only     # fast-forward only  
hint:  
hint: You can replace "git config" with "git config --global" to set a default  
hint: preference for all repositories. You can also pass --rebase, --no-rebase,  
hint: or --ff-only on the command line to override the configured default per  
hint: invocation.  
From https://github.com/SemyonovE/HelloGit  
 * branch            develop    -> FETCH_HEAD  
Updating 82567c9..e73e82a  
Fast-forward  
  HelloGit.xcodeproj/project.pbxproj | 16 ++++++  
  HelloGit/Components/Button.swift  | 39 ++++++  
  HelloGit/Components/Label.swift  | 30 ++++++  
  HelloGit/main.swift             |  3 ---  
 4 files changed, 85 insertions(+), 3 deletions(-)  
 create mode 100644 HelloGit/Components/Button.swift  
 create mode 100644 HelloGit/Components/Label.swift
```

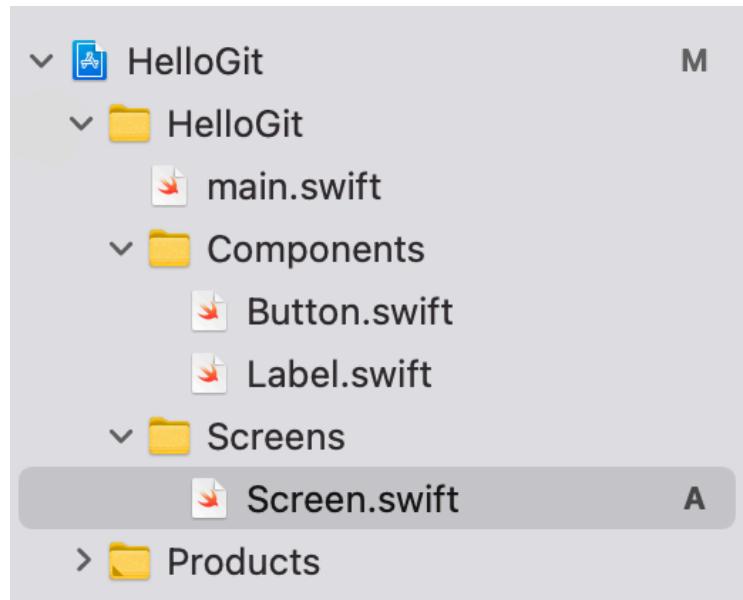
Здесь pull - получить, обратная команда, команде push - отправить.

Далее, всё тот же, основной разработчик, роль которого мы играем продолжает работу над проектом. Теперь нам нужно создать базовый класс экрана, чтобы его можно было использовать для создания первого экрана онбординга. Создаем новую ветку из текущей develop ветки

```
git checkout -b feature/onboarding
```

```
apple ~ ~/De/HelloGit git [P] develop > git checkout -b feature/onboarding  
Switched to a new branch 'feature/onboarding'
```

Теперь можно перейти в Xcode и продолжить работу над проектом. Создаем дополнительный файл с классом Screen и дополняем нашу структуру папок для новой сущности.



```
class Screen {

    var title: String
    var backgroundColor: String = "clear"

    init(title: String = "") {
        self.title = title

        didLoad()
        didShow()
    }

    func didLoad() {
        print("Screen \(title) loaded")
    }

    func didShow() {
        print("Screen \(title) showed")
    }
}
```

В системе контроля версий предпочтительно работать небольшими шагами, чтобы потом проще было разбираться в том, что из себя представляет отдельное изменение (коммит) в истории и нам проще было поправить ошибку (обратить изменения), если мы вдруг её допустили. Мы завершили работу по каркасу нашего экрана. Это будет некоторый абстрактный экран (как мы помним, мы не собираемся его отображать на самом деле), у которого будет два метода, срабатывающих когда экран загрузится (`didLoad`) и когда экран отобразиться (`didShow`), а также будут два свойства заголовок и цвет фона. Сделанные изменения теперь следует сохранить. Сперва нам нужно подготовить наши изменения и проверить, что всё готово

```
git add .
git status
```

```

apple ~ ~/De/HelloGit git (feature/onboarding) > git add .
apple ~ ~/De/HelloGit git (feature/onboarding +2) > git status
On branch feature/onboarding
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   HelloGit.xcodeproj/project.pbxproj
    new file:   HelloGit/Screens/Screen.swift

```

Далее делаем сам коммит

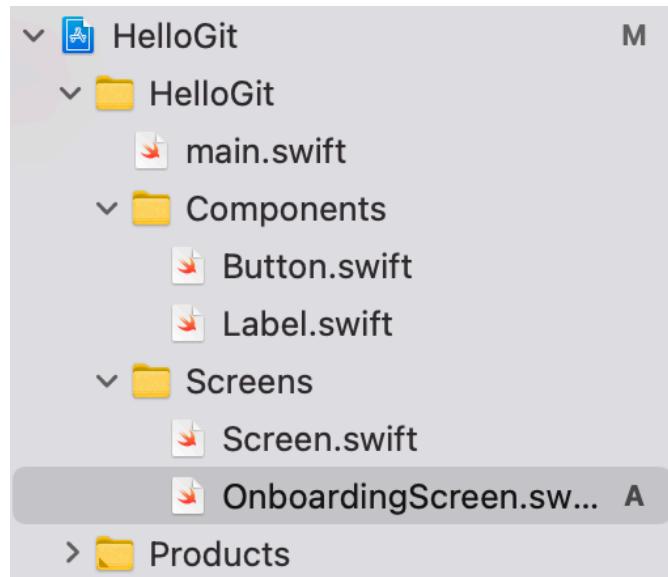
`git commit -m "Added Screen and changed folders hierarchy"`

```

apple ~ ~/De/HelloGit git (feature/onboarding +2) > git commit -m "Added Screen and changed folders hierarchy"
[feature/onboarding 2518ed4] Added Screen and changed folders hierarchy
  2 files changed, 41 insertions(+)
  create mode 100644 HelloGit/Screens/Screen.swift

```

Можем продолжить работу над проектом и добавить уже приветственный экран (онбординг).



```

class OnboardingScreen: Screen {

    private let titleLabel = Label(title: "Welcome my app")
    private let subtitleLabel = Label(
        title: "This app will help you to learn git",
        textColor: "light-gray"
    )
    private let launchButton = Button(
        title: "Launch",
        backgroundColor: "green"
    )

    private var onLaunchButton: (Button) -> Void
    var mainScreen: Screen?
}

```

```

init(title: String, onLaunchButton: @escaping (Button) -> Void) {
    self.onLaunchButton = onLaunchButton
    super.init(title: title)
}

override func didLoad() {
    super.didLoad()

    launchButton.addAction {
        // TODO: Launch main screen
    }
}

override func didShow() {
    super.didShow()

    onLaunchButton(launchButton)
}
}

```

Всё, что будет иметь приветственный экран это заголовок, подзаголовок и кнопку “Launch” для перехода на главный экран. Здесь также мы укажем, что нам нужна ссылка на этот главный экран (mainScreen), который будет добавлен позже и замыкание (onLaunchButton), через которое мы получим кнопку “Launch” в основном коде. Во время создания экрана мы передаем ему заголовок и замыкание для перехода. После чего сработают стандартные методы, описанного нами класса Screen, где добавляется действие для нажатия кнопки и кнопка передается в замыкание. Укажем в файле main.swift то, как следует использовать онбординг в программе.

```

var onboardingLaunchButton: Button?
let onboarding = OnboardingScreen(title: "Onboarding") {
    button in

        onboardingLaunchButton = button
}
}

guard let onboardingLaunchButton = onboardingLaunchButton else {
    fatalError("Has no button for launch app")
}

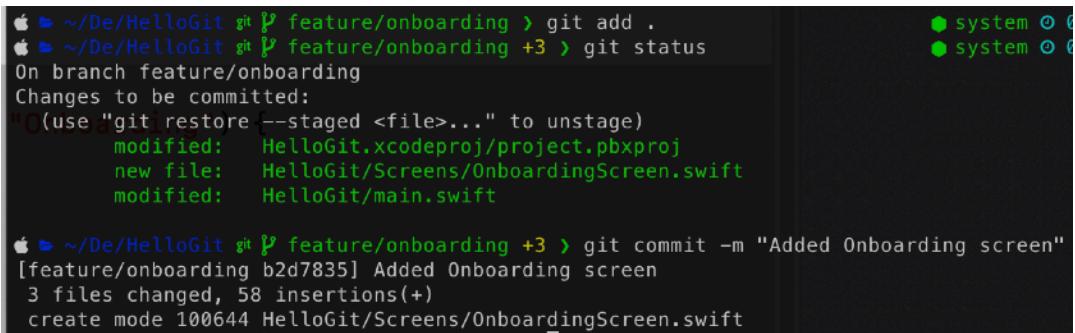
print("USER TOUCHES NEXT BUTTON")
onboardingLaunchButton.touch()

```

Здесь мы создаем переменную, в которой будет храниться ссылка на нашу кнопку, далее создаем приветственный экран, указывая заголовок и замыкание в котором нам будет передана кнопка для запуска приложения. Всё, что от нас требуется, сохранить ссылку на эту кнопку в ранее созданную переменную. После нужно убедиться, что такая кнопка у нас будет, иначе мы просто не сможем перейти на другой экран, поэтому в противном случае вызываем fatalError. Последним шагом мы выводим в консоль информацию о том, что пользователь нажимает на кнопку и вызываем у неё соответствующий метод touch(). Запустим и проверим, всё ли у нас работает корректно.

Итак, экран был загружен и показан, после чего пользователь нажал на кнопку. Так как при нажатии на кнопку мы пока ничего не описали (у нас стоит комментарий TODO, о том, что функционал нужно будет доделать позже, когда появится сам главный экран) следовательно перехода на другой экран пока что нет. Давайте пока сохраним в истории наши промежуточные изменения

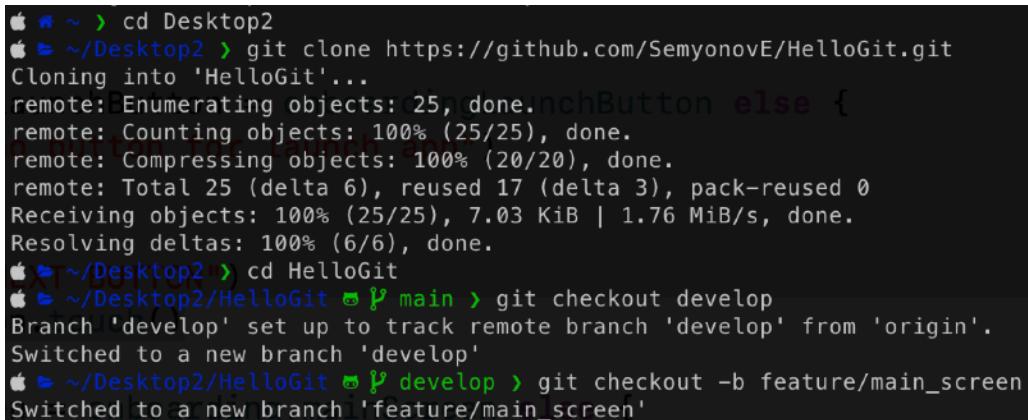
```
git add .
git status
git commit -m "Added Onboarding screen"
```



A screenshot of a macOS terminal window. The command `git add .` is run, followed by `git status`, which shows changes to be committed: modified files `HelloGit.xcodeproj/project.pbxproj`, `HelloGit/Screens/OnboardingScreen.swift`, and `HelloGit/main.swift`. Finally, `git commit -m "Added Onboarding screen"` is run, creating a new commit [feature/onboarding b2d7835] with the message "Added Onboarding screen".

Представим, что пока первый разработчик разбирается с экраном приветствия на проект подключается второй разработчик, который будет заниматься главным экраном. Чтобы всё было максимально правдоподобно, я открою отдельное окно терминала и создам отдельную папку Desktop2 в папке пользователя, как будто это разработчик, работающий с другого компьютера и у него проект будет уже на его рабочем столе. Стягивая себе проект разработчик увидит что есть ветки master и develop, он уже прочитал наши правила работы по git-flow, поэтому переключается на последнюю актуальную для разработки ветку (develop) и создает свою ветку для добавления в приложение главного экрана

```
cd Desktop2
git clone https://github.com/SemyonovE/HelloGit.git
cd HelloGit
git checkout develop
git checkout -b feature/main_screen
```



A screenshot of a macOS terminal window. It starts with `cd Desktop2`, then `git clone https://github.com/SemyonovE/HelloGit.git`, which clones the repository into the `HelloGit` directory. The output shows the progress of cloning objects and compressing them. After cloning, `cd HelloGit` is run, followed by `git checkout develop`, which sets up the local `develop` branch to track the remote `develop` branch from `origin`. Finally, `git checkout -b feature/main_screen` is run to switch to the new local `feature/main_screen` branch.

Смотрим с чем нам предстоит работать. Уже созданы компоненты экранов, но пока не создано ни одного экрана (первый разработчик сделал все локально на своем компьютере, поэтому второй про эти изменения не знает). Идем к первому разработчику и спрашиваем, есть ли у него уже готовый каркас для экрана, на что получаем положительный ответ и номер коммита в котором он реализован. Первый разработчик также может узнать номер этого коммита воспользовавшись командой 'git log' у которой есть множество настроек (не буду останавливаться на ней подробно, так как возможностей у неё действительно много). Выведем номер коммита (просто уникальный хэш, по которому можно идентифицировать любое изменение в нашей истории), как давно были сделаны изменения и какую информацию мы указали в описании коммита.

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

```
b2d7835 - 14 minutes ago : Added Onboarding screen
2518ed4 - 32 minutes ago : Added Screen and changed folders hierarchy
e73e82a - 3 days ago : Merge pull request #1 from SemyonovE/feature/components
30774a9 - 3 days ago : Added button and label components
82567c9 - 3 days ago : Init the project
6ff135f - 3 days ago : Initial commit
```

Но второму разработчику, который хочет получить себе каркас экрана мало просто знать номер коммита, ему также необходимо иметь к нему доступ. Тогда первый разработчик просто должен отправить все свои изменения на сервер, а второй стянуть их себе. Сначала выполним команды первого разработчика по отправке кода

```
git push origin feature/onboarding
```

```
apple:~/De/HelloGit git [?] feature/onboarding > git push origin feature/onboarding
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 8 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (15/15), 2.45 KiB | 2.45 MiB/s, done.
Total 15 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'feature/onboarding' on GitHub by visiting:
remote: https://github.com/SemyonovE>HelloGit/pull/new/feature/onboarding
remote:
To https://github.com/SemyonovE>HelloGit.git
 * [new branch]      feature/onboarding -> feature/onboarding
```

Указываем то, какая ветка и куда будет отправлена. Теперь второй разработчик может стянуть себе все изменения с удаленного репозитория. Но сначала нам надо узнать, как точно называется ветка с нужным коммитом. Можно просто спросить у первого разработчика, либо посмотреть информацию об удаленном репозитории в терминале

```
git ls-remote origin
```

или

## git remote show origin

```
apple ~ ~/Desktop2>HelloGit git feature/main_screen > git ls-remote origin
82567c950f9fcc2409e0bc109fb7c8430867654d HEAD
e73e82a5c5ed4514803a36204a96c35572ad5cfc refs/heads/develop
30774a9b7e8b442eb8831aead39aa249f8828a73 refs/heads/feature/components
b2d783566f4eaab97721497336ccb9158b28ce5d refs/heads/feature/onboarding
82567c950f9fcc2409e0bc109fb7c8430867654d refs/heads/main
30774a9b7e8b442eb8831aead39aa249f8828a73 refs/pull/1/head
apple ~ ~/Desktop2>HelloGit git feature/main_screen > git remote show origin
* remote origin
  Fetch URL: https://github.com/SemyonovE/HelloGit.git
  Push URL: https://github.com/SemyonovE/HelloGit.git
  HEAD branch: main
  Remote branches:
    develop           tracked
    feature/components tracked
    feature/onboarding tracked
    main              tracked
  Local branches configured for 'git pull':
    develop merges with remote develop
    main    merges with remote main
  Local refs configured for 'git push':
    develop pushes to develop (up to date)
    main    pushes to main   (up to date)
```

Осталось получить необходимую ветку. Мы уже знаем команду pull, которая стянет необходимую нам ветку со всеми коммитами на компьютер. Однако, если мы применим ее находясь на ветке feature/main\_screen то все изменения будут применены к текущей ветке. Нам бы этого не хотелось, потому что никакой приветственный экран нам в ветке главного экрана не нужен. Есть альтернатива команде pull (команда fetch), которая позволяет стянуть данные, но не применять их для текущей ветки. Тем самым мы просто получим ветку feature/onboarding в ее локальную копию (с таким же именем), оставив ветку feature/main\_screen без изменений.

## git fetch origin feature/onboarding

```
apple ~ ~/Desktop2>HelloGit git feature/main_screen > git fetch origin feature/onboarding
From https://github.com/SemyonovE/HelloGit
 * branch            feature/onboarding -> FETCH_HEAD
 * [new branch]      feature/onboarding -> origin/feature/onboarding
apple ~ ~/Desktop2>HelloGit git feature/main_screen > git log --graph --oneline --all
```

Воспользуемся другим вариантом команды git log, чтобы посмотреть, как теперь выглядят наши изменения на проекте.

## git log --graph --oneline --all

```
* b2d7835 (origin/feature/onboarding) Added Onboarding screen
* 2518ed4 Added Screen and changed folders hierarchy
* e73e82a (HEAD -> feature/main_screen, origin/develop, develop) Merge pull request #1 from SemyonovE/feature/components
| \
| * 30774a9 (origin/feature/components) Added button and label components
| /
| * 82567c9 (origin/main, origin/HEAD, main) Init the project
* 6ff135f Initial commit
```

Откроется дерево нашей истории коммитов с названиями веток, номерами коммитов, описанием и самое главное мы увидим визуально в каком порядке и какие изменения были сделаны первым разработчиком на проекте.

Видно как проект был начат (самый низ дерева), после чего отделилась ветка с компонентами и была влита обратно в develop, где в том числе находимся сейчас и мы (ветка feature/main\_screen). Скопируем здесь номер необходимого нам коммита, либо нам его передал первый разработчик ранее и закроем просмотр дерева (выйти из просмотра можно по прежнему нажав 'q' при английской раскладке).

Теперь, когда у нас есть нужная ветка локально и мы знаем номер необходимого коммита, нам нужно забрать себе коммит первого разработчика с наработками по каркасу экрана. Сделать это можно специальной командой

```
git cherry-pick 2518ed4
```

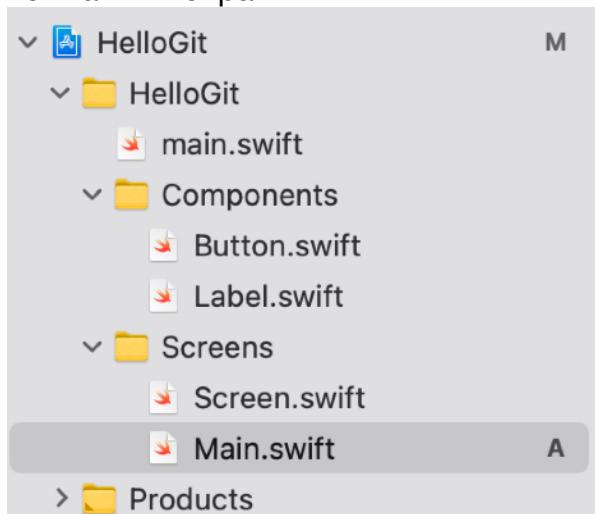
```
Apple ➜ ~/Desktop2/HelloGit git [feature/main_screen] > git cherry-pick 2518ed4
[feature/main_screen 9516e38] Added Screen and changed folders hierarchy
  Date: Sat Apr 3 13:26:02 2021 +0700
  2 files changed, 41 insertions(+)
  create mode 100644 HelloGit/Screens/Screen.swift
```

Если вновь вызвать просмотр дерева истории

```
git log --graph --oneline --all
```

```
* 9516e38 (HEAD -> feature/main_screen) Added Screen and changed folders hierarchy
| * b2d7835 (origin/feature/onboarding) Added Onboarding screen
| * 2518ed4 Added Screen and changed folders hierarchy
|/
* e73e82a (origin/develop, develop) Merge pull request #1 from SemyonovE/feature/components
| \
| * 30774a9 (origin/feature/components) Added button and label components
|/
* 82567c9 (origin/main, origin/HEAD, main) Init the project
* 6ff135f Initial commit
```

Мы увидим, что полностью клонировали себе нужный коммит с файлом каркаса и его описание. Существуют и другие возможности забрать наработки из других веток и даже команда cherry-pick не так проста, но более детально этот вопрос можно изучить отдельно. Со спокойной душой второй разработчик садится и добавляет в приложение главный экран.



```

class Main: Screen {

    private let titleLabel = Label(title: "My super git app")

    override func didShow() {
        super.didShow()

        print("Loading news")
        // TODO: Load news
    }
}

```

Как видите у нас в проекте нет экрана онбординга, потому что он остался в ветке первого разработчика и это правильно, ведь он не имеет отношения к главному экрану. А вот без файла Screen.swift у нас бы не получилось реализовать главный экран.

Сам экран имеет заголовок и печать в консоль информации о том, что загружаются новости. Никаких новостей пока нет, поэтому снова оставляем себе на будущее комментарий, что их нужно не забыть добавить и коммитим изменения

```

git add .
git status
git commit -m "Added main screen"
git push origin feature/main_screen

```

```

$ ~/Desktop2>HelloGit git feature/main_screen !1 ?1 > git add .
$ ~/Desktop2>HelloGit git feature/main_screen +2 > git status
On branch feature/main_screen
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   HelloGit.xcodeproj/project.pbxproj
    new file:   HelloGit/Screens/Main.swift
$ ~/Desktop2>HelloGit git feature/main_screen +2 > git commit -m "Added main screen"
[feature/main_screen c2c2091] Added main screen
 2 files changed, 24 insertions(+)
  create mode 100644 HelloGit/Screens/Main.swift
$ ~/Desktop2>HelloGit git feature/main_screen > git push origin feature/main_screen
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 8 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (14/14), 2.00 KiB | 2.00 MiB/s, done.
Total 14 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'feature/main_screen' on GitHub by visiting:
remote:     https://github.com/SemyonovE/HelloGit/pull/new/feature/main_screen
remote:
To https://github.com/SemyonovE/HelloGit.git
 * [new branch]      feature/main_screen -> feature/main_screen

```

И отправляем их на ревью

The screenshot shows a GitHub repository page. At the top, there are two pull requests: one from 'feature/onboarding' to 'main' (merged 20 minutes ago) and another from 'feature/main\_screen' to 'main' (merged 1 minute ago). Below these, the 'main' branch summary shows 5 branches and 0 tags. A commit list for 'SemyonovE' shows initial commits for files like 'HelloGit.xcodeproj', '.gitignore', 'LICENSE', and 'README.md'. The 'README.md' file content is: "HelloGit\nThis is an app for git tutorial". On the right side, there are sections for 'About', 'Releases', 'Packages', and 'Languages'.

Выбираем создание ПР для нужной ветки. Не забываем также сменить ветку назначения на develop, если там указана другая.

[Open a pull request](#)

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

The screenshot shows the GitHub pull request creation interface. It displays a comparison between the 'develop' branch (base) and the 'feature/main\_screen' branch (compare). A green checkmark indicates that the branches are 'Able to merge'. The comment area is empty, and the 'Create pull request' button is at the bottom right.

Дальше также ничего нового. Первый разработчик смотрит код и вливает его если всё ок. Но что, если код содержит неточности? Второму разработчику нужно внести изменения в проект, закоммитить их и снова отправить на удаленный репозиторий. Давайте для примера поменяем заголовок главного экрана на более содержательный.

```
private let titleLabel = Label(title: "Hello Git App")
```

И отправляем наши изменения, предварительно их закоммитив

```
git add .
git status
git commit -m "Fixed main screen title for review"
git push origin feature/main_screen
```

```

apple ~ ~/Desktop2/HelloGit $ feature/main_screen > git add .
apple ~ ~/Desktop2/HelloGit $ feature/main_screen +1 > git status
On branch feature/main_screen
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   HelloGit/Screens/Main.swift

apple ~ ~/Desktop2/HelloGit $ feature/main_screen +1 > git commit -m "Fixed main screen title for review"
[feature/main_screen 48f0b21] Fixed main screen title for review
 1 file changed, 1 insertion(+), 1 deletion(-)
apple ~ ~/Desktop2/HelloGit $ feature/main_screen > git push origin feature/main_screen
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 512 bytes | 512.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/SemyonovE/HelloGit.git
  c2c2091..48f0b21  feature/main_screen -> feature/main_screen

```

Не обязательно каждый раз вызывать `git status`, здесь это сделано для примера, в целях самоконтроля, что мы ничего не забыли. Перед отправкой тоже полезно посмотреть статус, чтобы убедиться, что мы закоммитили все необходимые изменения.

Далее на GitHub нам не нужно будет дополнительного что-то делать. Изменения автоматически подтянутся в созданный нами ПР.

### Feature/main screen #3

**SemyonovE** commented 4 minutes ago

No description provided.

**SemyonovE** added 3 commits 2 hours ago

- Added Screen and changed folders hierarchy 9516e38
- Added main screen c2c2091
- Fixed main screen title for review 48f0b21

Add more commits by pushing to the `feature/main_screen` branch on **SemyonovE/HelloGit**.

Continuous integration has not been set up  
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch  
Merging can be performed automatically.

Merge pull request

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Close pull request Comment

Теперь всё корректно, и ветка может быть влита. Как бы переключаемся на роль первого разработчика и нажимаем “Merge pull request”, далее нас устроит стандартный комментарий для коммита, содержащего слияние одной ветки в другую, поэтому нажимаем “Confirm merge”. Далее первый разработчик может стянуть себе изменения develop ветки (там теперь появился главный экран), чтобы закончить экран приветствия, добавив переход на главный. Мы можем выполнить команду 'git fetch', как делали раньше, потому как собираемся стягивать не ту ветку в которой находимся

**git fetch origin develop**

```
apple ~ ~/De/HelloGit git feature/onboarding > git fetch origin develop
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 20 (delta 4), reused 19 (delta 4), pack-reused 0
Unpacking objects: 100% (20/20), 3.02 KiB | 309.00 KiB/s, done.
From https://github.com/SemyonovE/HelloGit
 * branch            develop    -> FETCH_HEAD
   e73e82a..fbc473f  develop    -> origin/develop
```

Однако теперь нам не достаточно просто получить изменения удаленного репозитория. Нужно именно заменить локальную ветку develop удаленной. В git мы оперируем указателями, на самом деле находясь сейчас на ветке feature/onboarding мы не находимся на некотором физическом ответвлении нашего репозитория, данное имя ветки это лишь указатель на те несколько коммитов, которые мы делали находясь в текущей ветке. На самом деле наша локальная ветка сейчас просто указывает на коммит, который был раньше, и чтобы локальная ветка develop стала такой же длинной как удаленная (включала все последние коммиты), нам достаточно просто сдвинуть указатель локальной ветки develop на самый последний коммит удаленной ветки develop. Если мы снова посмотрим дерево коммитов

**git log --graph --oneline --all**

```
* fbc473f (origin/develop) Merge pull request #3 from SemyonovE/feature/main_screen
|\ 
| * 48f0b21 (origin/feature/main_screen) Fixed main screen title for review 743
| * c2c2091 Added main screen
| * 9516e38 Added Screen and changed folders hierarchy
|/
| * b2d7835 (HEAD -> feature/onboarding, origin/feature/onboarding) Added Onboarding screen 51
| * 2518ed4 Added Screen and changed folders hierarchy 28.03
|/
* e73e82a (develop) Merge pull request #1 from SemyonovE/feature/components
|\ 
| * 30774a9 (origin/feature/components, feature/components) Added button and label components
|/
* 82567c9 (origin/main, origin/HEAD, main) Init the project
* 6ff135f Initial commit
(END)
```

то увидим, что текущий указатель (HEAD), который мы переключаем, прыгая с ветки на ветку, находится в другой ветке, давайте переключимся на ветку develop, чтобы обновить ее.

## git checkout develop

```
Apple ➜ ~/De/HelloGit git ⌂ feature/onboarding > git checkout develop
Switched to branch 'develop'
```

И снова выполним команду отображения дерева коммитов

```
git log --graph --oneline --all
```

```
* fbc473f (origin/develop) Merge pull request #3 from SemyonovE/feature/main_screen
|\ 
| * 48f0b21 (origin/feature/main_screen) Fixed main screen title for review
| * c2c2091 Added main screen
| * 9516e38 Added Screen and changed folders hierarchy
|
| * b2d7835 (origin/feature/onboarding, feature/onboarding) Added Onboarding screen
| * 2518ed4 Added Screen and changed folders hierarchy
|
* e73e82a (HEAD -> develop) Merge pull request #1 from SemyonovE/feature/components
|
| * 30774a9 (origin/feature/components, feature/components) Added button and label components
|
* 82567c9 (origin/main, origin/HEAD, main) Init the project
* 6ff135f Initial commit
(END)
```

Видим, что текущий указатель поменялся и находится на том коммите, куда указывает локальная ветка develop. Но удаленный develop (origin/develop) находится немного выше. Все, что нам надо сделать, это протянуть локальный develop в самый верх ветки origin/develop. Для этого воспользуемся командой rebase (перебазировать). Хотя ее возможности несколько шире, сейчас она хорошо подходит для нашей цели.

```
git rebase origin/develop
```

```
Apple ➜ ~/De/HelloGit git ⌂ develop > git rebase origin/develop
Successfully rebased and updated refs/heads/develop.
```

И снова посмотрим на дерево коммитов

```
git log --graph --oneline --all
```

```
* fbc473f (HEAD -> develop, origin/develop) Merge pull request #3 from SemyonovE/feature/main_screen
|\ 
| * 48f0b21 (origin/feature/main_screen) Fixed main screen title for review
| * c2c2091 Added main screen
| * 9516e38 Added Screen and changed folders hierarchy
|
| * b2d7835 (origin/feature/onboarding, feature/onboarding) Added Onboarding screen
| * 2518ed4 Added Screen and changed folders hierarchy
|
* e73e82a Merge pull request #1 from SemyonovE/feature/components
|
| * 30774a9 (origin/feature/components, feature/components) Added button and label components
|
* 82567c9 (origin/main, origin/HEAD, main) Init the project
* 6ff135f Initial commit
(END)
```

Отлично, теперь наша локальная копия develop ветки знает обо всех актуальных изменениях в коде. Другой способ, как можно было добиться того же результата, это сначала переключиться на ветку develop

### git checkout develop

```
apple ~ ~/De/HelloGit git (develop) > git rebase origin/develop
Successfully rebased and updated refs/heads/develop.
```

А после вызвать команду pull, в результате которой мы бы не только загрузили обновления origin/develop, но и сразу же применили их к текущей ветке

### git pull origin develop

```
apple ~ ~/De/HelloGit git (develop) > git pull origin develop
hint: Pulling without specifying how to reconcile divergent branches is
hint: discouraged. You can squelch this message by running one of the following
hint: commands sometime before your next pull:
hint:
hint:   git config pull.rebase false # merge (the default strategy)
hint:   git config pull.rebase true  # rebase
hint:   git config pull.ff only    # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
From https://github.com/SemyonovE/HelloGit
 * branch            develop      -> FETCH_HEAD
Updating e73e82a..fbc473f
Fast-forward
 HelloGit.xcodeproj/project.pbxproj | 16 ++++++
>HelloGit/Screens/Main.swift        | 20 ++++++
>HelloGit/Screens/Screen.swift     | 29 ++++++
3 files changed, 65 insertions(+)
 create mode 100644 HelloGit/Screens/Main.swift
 create mode 100644 HelloGit/Screens/Screen.swift
```

### git log --graph --oneline --all

```
* fbc473f (HEAD -> develop, origin/develop) Merge pull request #3 from SemyonovE/feature/main_screen
| \
| * 48f0b21 (origin/feature/main_screen) Fixed main screen title for review 74346a
| * c2c2091 Added main screen 28.03.201
| * 9516e38 Added Screen and changed folders hierarchy 51f4ae
| /
| * b2d7835 (origin/feature/onboarding, feature/onboarding) Added Onboarding screen 28.03.201
| * 2518ed4 Added Screen and changed folders hierarchy
| / \
| * e73e82a Merge pull request #1 from SemyonovE/feature/components
| \
| * 30774a9 (origin/feature/components, feature/components) Added button and label components
| /
| * 82567c9 (origin/main, origin/HEAD, main) Init the project
| * 6ff135f Initial commit
(END)
```

Результат будет точно таким же: мы обновим локальную ветку develop. Теперь возвращаемся в ветку, где мы работаем на экраном приветствия

## git checkout feature/onboarding

```
Apple ➜ ~/De/HelloGit git 🍭 develop > git checkout feature/onboarding
Switched to branch 'feature/onboarding'
```

Нам нужно добавить переход на главный экран, но информация о нем хранится в совершенно другой ветке. Как же нам объединить и тот и текущий код. Можно вспомнить про команду cherry-pick, но не всегда стоит рассматривать её как самый верный вариант. Стягивая в текущую ветку все коммиты другой мы просто наводим беспорядок и потом не будет понятно кто отвечал за ту или иную реализацию в коде. На самом деле мы можем просто смержить ветки и сделать это без ПР. Почему же мы не будем создавать ПР для этого, спросите вы. Все просто. Мы будем мерджить код в другом направлении (с точки зрения приоритетов веток), нежели раньше. Мы вольем код, который есть в ветке develop в свою ветку feature/onboarding. И здесь нам не нужно чье-то разрешение, потому что в develop находится уже проверенный код, а наша ветка это наше личное дело. Вот когда мы будем влиять все изменения обратно в develop, тогда без ПР уже не обойтись. Но сейчас давайте заберем себе весь код, что есть сейчас в develop ветке.

## git merge develop

```
Apple ➜ ~/De/HelloGit git 🍭 feature/onboarding > git merge develop
Auto-merging HelloGit.xcodeproj/project.pbxproj
CONFLICT (content): Merge conflict in HelloGit.xcodeproj/project.pbxproj
Automatic merge failed; fix conflicts and then commit the result.
```

И мы сталкиваемся с первой проблемой: у нас есть конфликты. Всё дело в том, что мы в текущей ветке добавили экран приветствия, а в develop ветке есть код с добавлением в проект главного экрана. Не всегда у нас получится все слить просто, без лишней работы, в таком случае нам нужно разрешить все конфликты. Что это за конфликты? Это все те места в коде и настройках нашего проекта, за объединение которых git не хочет брать на себя ответственность. Он не может догадаться действительно ли теперь должно стать два экрана, или это на самом деле один экран, просто его кто-то переименовал. Чтобы увидеть, какие файлы сейчас нужно поправить мы можем вызвать

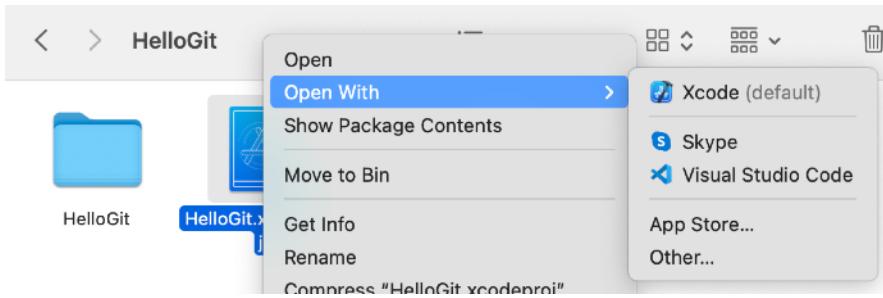
## git status

```
Apple ➜ ~/De/HelloGit git 🍭 feature/onboarding merge ~1 +1 > git status
On branch feature/onboarding
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:
  new file:   HelloGit/Screens/Main.swift

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:  HelloGit.xcodeproj/project.pbxproj
```

Видим, что у нас HelloGit.xcodeproj (файл нашего проекта) был изменен в обеих ветках. И теперь наша задача подсказать git как должен этот файл выглядеть в окончательном варианте. Можем просто открыть этот файл в любом текстовом редакторе.



У меня установлен VSCode, поэтому воспользуюсь им.

```
/* Begin PBXBuildFile section */
29763D14261840B300322122 /* Screen.swift in Sources */ = {isa = PBXBuildFile; fileRef = 29763D13261840B300322122 /* Screen.swift */; }
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<< HEAD (Current Change)
29763D172618426600322122 /* OnboardingScreen.swift in Sources */ = {isa = PBXBuildFile; fileRef = 29763D162618426600322122 /* Onboardi
=====
29763D1F2618535700322122 /* Main.swift in Sources */ = {isa = PBXBuildFile; fileRef = 29763D1E2618535700322122 /* Main.swift */; }
>>>>> develop (Incoming Change)
2980170D2614BBA200971AD8 /* main.swift in Sources */ = {isa = PBXBuildFile; fileRef = 2980170C2614BBA200971AD8 /* main.swift */; }
298017182614D18000971AD8 /* Button.swift in Sources */ = {isa = PBXBuildFile; fileRef = 298017162614D18000971AD8 /* Button.swift */; }
298017192614D18000971AD8 /* Label.swift in Sources */ = {isa = PBXBuildFile; fileRef = 298017172614D18000971AD8 /* Label.swift */; }
/* End PBXBuildFile section */

/* Begin PBXCopyFilesBuildPhase section */
298017072614BBA200971AD8 /* CopyFiles */ = [
    isa = PBXCopyFilesBuildPhase;
    buildActionMask = 2147483647;
    dstPath = /usr/share/man/man1/;
    dstSubFolderSpec = 0;
    files = (
        );
    runOnlyForDeploymentPostprocessing = 1;
];
/* End PBXCopyFilesBuildPhase section */

/* Begin PBXFileReference section */
29763D13261840B300322122 /* Screen.swift */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.swift; path = .
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<< HEAD (Current Change)
29763D162618426600322122 /* OnboardingScreen.swift */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.swift;
=====
29763D1E2618535700322122 /* Main.swift */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.swift; path = Ma
>>>>> develop (Incoming Change)
298017092614BBA200971AD8 /* HelloGit */ = {isa = PBXFileReference; explicitFileType = "compiled.mach-o.executable"; includeInIndex = 0
2980170C2614BBA200971AD8 /* main.swift */ = {isa = PBXFileReference; lastKnownFileType = sourcecode.swift; path = main.swift; sourceType
298017182614D18000971AD8 /* Button.swift */ = {isa = PBXFileReference; lastKnownFileType = sourcecode.swift; path = Button.swift; sou
```

По коду будут отображаться вставки вида

```
<<<<< HEAD
Какой-то код
=====
Какой-то другой код
>>>>> develop
```

Соответственно в одном месте в файле проекта мы имеем две вариации кода (из каждой ветки) и нам нужно определиться что мы оставим. Код в верхней части это код нашей текущей ветки (feature/onboarding), а снизу код из develop ветки, который мы пытаемся вмержить. Так как нам нужно оставить оба экрана, нам надо удалить системные подписи, оставив только код каждого из кусков. Но в будущем вам могут попасться ситуации, когда два разработчика поменяли один и тот же код, и оставить следует только одну версию. Тут уже нужно просто принять решение, в зависимости от ситуации (скорее всего оставить более свежую реализацию).

```

/* Begin PBXBuildFile section */
29763D14261840B300322122 /* Screen.swift in Sources */ = {isa = PBXBuildFile; fileRef = 29763D13261840B300322122 /* Screen.swift */; }
29763D172618426600322122 /* OnboardingScreen.swift in Sources */ = {isa = PBXBuildFile; fileRef = 29763D162618426600322122 /* Onboardi
29763D1F2618535700322122 /* Main.swift in Sources */ = {isa = PBXBuildFile; fileRef = 29763D1E2618535700322122 /* Main.swift */; }
2980170D2614BBA200971AD8 /* main.swift in Sources */ = {isa = PBXBuildFile; fileRef = 2980170C2614BBA200971AD8 /* main.swift */; }
298017182614D18000971AD8 /* Button.swift in Sources */ = {isa = PBXBuildFile; fileRef = 298017162614D18000971AD8 /* Button.swift */; }
298017192614D18000971AD8 /* Label.swift in Sources */ = {isa = PBXBuildFile; fileRef = 298017172614D18000971AD8 /* Label.swift */; }

/* End PBXBuildFile section */

/* Begin PBXCopyFilesBuildPhase section */
298017072614BBA200971AD8 /* CopyFiles */ = {
    isa = PBXCopyFilesBuildPhase;
    buildActionMask = 2147483647;
    dstPath = /usr/share/man/man1/;
    dstSubfolderSpec = 0;
    files = (
    );
    runOnlyForDeploymentPostprocessing = 1;
};

/* End PBXCopyFilesBuildPhase section */

/* Begin PBXFileReference section */
29763D13261840B300322122 /* Screen.swift */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.swift; path =
29763D162618426600322122 /* OnboardingScreen.swift */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.swift;
29763D1E2618535700322122 /* Main.swift */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.swift; path = Ma
298017092614BBA200971AD8 /* HelloGit */ = {isa = PBXFileReference; explicitFileType = "compiled.mach-o.executable"; includeInIndex = 0
2980170C2614BBA200971AD8 /* main.swift */ = {isa = PBXFileReference; lastKnownFileType = sourcecode.swift; path = main.swift; sourcefir
298017162614D18000971AD8 /* Button.swift */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.swift; path =
298017172614D18000971AD8 /* Label.swift */ = {isa = PBXFileReference; fileEncoding = 4; lastKnownFileType = sourcecode.swift; path = L

/* End PBXFileReference section */

```

Удаляем все системные включения о конфликтах, оставляя сам код (у меня таких ошибок было 4, у вас количество может отличаться) и возвращаемся в терминал. Мы исправили все конфликты, так как они все были в одном файле, теперь можно создать коммит с вливанием.

```

git add .
git status
git commit -m "Merged develop into feature/onboarding"

```

```

apple:~/De/HelloGit $ git feature/onboarding merge ~1 +1 > git add .
apple:~/De/HelloGit $ git feature/onboarding merge +2 > git status
On branch feature/onboarding
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
  modified:   HelloGit.xcodeproj/project.pbxproj
  new file:   HelloGit/Screens/Main.swift

apple:~/De/HelloGit $ git feature/onboarding merge +2 > git commit -m "Merged develop into feature/onboarding"
[feature/onboarding beeaa807] Merged develop into feature/onboarding

```

**git log --graph --oneline --all**

```

* beeaa807 (HEAD -> feature/onboarding) Merged develop into feature/onboarding
| \
| * fbc473f (origin/develop, develop) Merge pull request #3 from SemyonovE/feature/main_screen
| |
| * 48f0b21 (origin/feature/main_screen) Fixed main screen title for review
| * c2c2091 Added main screen
| * 9516e38 Added Screen and changed folders hierarchy
| /
* b2d7835 (origin/feature/onboarding) Added Onboarding screen
* 2518ed4 Added Screen and changed folders hierarchy
| /
* e73e82a Merge pull request #1 from SemyonovE/feature/components
| \
| * 30774a9 (origin/feature/components, feature/components) Added button and label components
| /
* 82567c9 (origin/main, origin/HEAD, main) Init the project
* 6ff135f Initial commit
(END)

```

Видим на схеме, что мы всё сделали, как хотели.

Итак, подготовительные работы закончены, сделаем переход на главный экран. Изменим тело замыкания в файле OnboardingScreen.swift

```
launchButton.addAction {  
    [weak self] in  
  
    self?.mainScreen = Main(title: "Main")  
}
```

и проверим в файле main.swift, что у нас все получилось

```
guard let main: Screen = onboarding.mainScreen else {  
    fatalError("Main doesn't exist")  
}
```

Отлично, видим, что главный экран загружен, показан и есть загрузка новостей, но пока без самой реализации. Коммитим изменения и отправляем на ПР.

```
Screen Onboarding loaded  
Screen Onboarding showed  
USER TOUCHES NEXT BUTTON  
Screen Main loaded  
Screen Main showed  
Loading news  
Program ended with exit code: 0
```

```
git add .  
git status  
git commit -m "Used main screen for creating launch flow"  
git push origin feature/onboarding
```

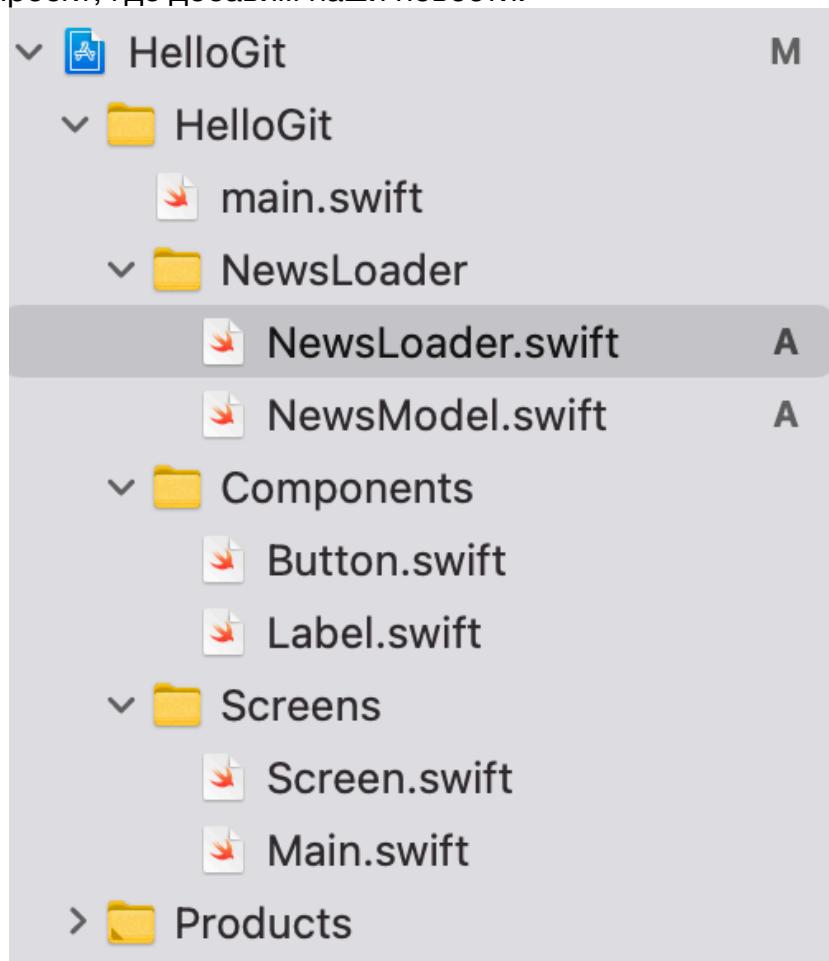
```
Apple ➜ ~/De/HelloGit git (feature/onboarding !2) > git add .  
Apple ➜ ~/De/HelloGit git (feature/onboarding +2) > git status  
On branch feature/onboarding  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    modified:   HelloGit/Screens/OnboardingScreen.swift  
    modified:   HelloGit/main.swift  
  
Apple ➜ ~/De/HelloGit git (feature/onboarding +2) > git commit -m "Used main screen for creating launch flow"  
[feature/onboarding c4b8b12] Used main screen for creating launch flow  
 2 files changed, 9 insertions(+), 3 deletions(-)  
Apple ➜ ~/De/HelloGit git (feature/onboarding) > git push origin feature/onboarding  
Enumerating objects: 25, done.  
Counting objects: 100% (25/25), done.  
Delta compression using up to 8 threads  
Compressing objects: 100% (12/12), done.  
Writing objects: 100% (12/12), 1.55 KiB | 1.55 MiB/s, done.  
Total 12 (delta 4), reused 0 (delta 0), pack-reused 0  
remote: Resolving deltas: 100% (4/4), completed with 3 local objects.  
To https://github.com/SemyonovE/HelloGit.git  
 b2d7835..c4b8b12  feature/onboarding -> feature/onboarding
```

Создаем ПР по старой схеме и меняем роль на второго разработчика. Наша задача убедиться, что все в ПР хорошо и влить его. Тем временем, после создания главного экрана этот разработчик приступил к подгрузке новостей. Создадим для этого отдельную ветку

```
git checkout -b feature/news_loader
```

```
apple ~ ~/Desktop2>HelloGit git (feature/main_screen) > git checkout -b feature/news_loader
Switched to a new branch 'feature/news_loader'
```

и перейдем в проект, где добавим наши новости.



Добавляем модель, описывающую каждую новость (заголовок, дата, автор и текст) и описание новости, которое мы будем выводить в консоль.

```
struct NewsModel: CustomStringConvertible {  
  
    let title: String  
    let text: String  
    let date: String  
    let author: String  
  
    var description: String {  
        ""  
        -----  
        \(title) | \(date)  
        -----  
        \(text)  
        -----  
        \(author)  
        -----  
        ""  
    }  
}
```

Далее добавляем класс загрузчика новостей с зашитыми новостями.

```
enum NewsLoader {  
  
    static private let news: [NewsModel] = [  
        NewsModel(title: "Внимание",  
                  text: "Сегодня гололёд! Будьте осторожны!!!",  
                  date: "28.03.2021",  
                  author: "Андрей Фролов"),  
        NewsModel(title: "Британские учёные",  
                  text: "В африке обнаружен новый вид белок!",  
                  date: "27.03.2021",  
                  author: "Кирилл Тунгас"),  
        NewsModel(title: "И земля содрогнулась",  
                  text: "В городе Энске произошло землетрясение 8  
баллов.",  
                  date: "20.03.2021",  
                  author: "Алексей Иванов")  
    ]  
  
    static func load() -> [NewsModel] {  
        return news  
    }  
}
```

Возвращаемся в файл Main.swift и там, где мы оставляли комментарий реализуем подгрузку новостей

```
NewsLoader.load()  
.sorted { $0.date < $1.date }  
.forEach { print($0) }
```

Загружаем новости, сортируем их по дате и выводим все новости в консоль. Так как в текущей ветке у нас нету флоу (последовательности вызова экранов) самого приложения, можно в main.swift просто добавить создание главного, чтобы проверить работу загрузки новостей.

```
let main = Main(title: "Main")
```

```

Screen Main loaded
Screen Main showed
Loading news
-----
И земля содрогнулась | 20.03.2021
-----
В городе Энске произошло землетрясение 8
баллов.

Алексей Иванов
-----
-----
Британские учёные | 27.03.2021
-----
В африке обнаружен новый вид белок!

Кирилл Тунгас
-----
-----
Внимание | 28.03.2021
-----
Сегодня гололёд! Будьте осторожны!!!

```

All Output

Все корректно, не забудем удалить строку создания экрана, потому что она была для теста и закоммитим, а после отправим наши изменения.

```

git add .
git status
git commit -m "Added news loader"
git push origin feature/news_loader

```

```

apple:~/Desktop2>HelloGit git:~feature/news_loader !2 ?1 > git add .
apple:~/Desktop2>HelloGit git:~feature/news_loader +4 > git status
On branch feature/news_loader
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
    modified:   HelloGit.xcodeproj/project.pbxproj
    new file:   HelloGit/NewsLoader/NewsLoader.swift
    new file:   HelloGit/NewsLoader/NewsModel.swift
    modified:   HelloGit/Screens/Main.swift

apple:~/Desktop2>HelloGit git:~feature/news_loader +4 > git commit -m "Added news loader"
[feature/news_loader 4aa5790] Added news loader
 4 files changed, 79 insertions(+), 1 deletion(-)
  create mode 100644 HelloGit/NewsLoader/NewsLoader.swift
  create mode 100644 HelloGit/NewsLoader/NewsModel.swift
apple:~/Desktop2>HelloGit git:~feature/news_loader > git push origin feature/news_loader
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 8 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 1.82 KiB | 1.82 MiB/s, done.
Total 10 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'feature/news_loader' on GitHub by visiting:
remote:     https://github.com/SemyonovE/HelloGit/pull/new/feature/news_loader
remote:
To https://github.com/SemyonovE/HelloGit.git
 * [new branch]      feature/news_loader -> feature/news_loader

```

Создаем ПР как мы делали это раньше.

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

 Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Но что мы видим. Вливание не может быть произведено автоматически, потому что у нас есть конфликты. Это связано с тем, что другой (первый) разработчик прежде уже залил изменения экрана приветствия, а мы их не подтянули, добавив в проект новые файлы. Ничего страшного. Мы можем создать PR и пойдем исправим ситуацию у себя локально, после чего уведомим удаленный репозиторий об этих изменениях.

## Added news loader #5

 Open SemyonovE wants to merge 1 commit into [develop](#) from [feature/news\\_loader](#) 

 Conversation 0  Commits 1  Checks 0  Files changed 4

 SemyonovE commented now Owner  ...

No description provided.

  Added news loader 4aa5790

---

Add more commits by pushing to the [feature/news\\_loader](#) branch on [SemyonovE>HelloGit](#).

  This branch has conflicts that must be resolved  
Use the [web editor](#) or the [command line](#) to resolve conflicts. [Resolve conflicts](#)

**Conflicting files**  
HelloGit.xcodeproj/project.pbxproj

[Merge pull request](#)  You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Видим файл, который конфликтует, это всё тот же файл проекта. Стянем себе актуальную версию develop ветки переключившись на нее заранее.

```
git checkout develop  
git pull origin develop
```

```
Apple ~ ~/Desktop2/HelloGit git (feature/news_loader) > git checkout develop ● system 0 0  
Switched to branch 'develop'  
Your branch is behind 'origin/develop' by 9 commits, and can be fast-forwarded.  
(use "git pull" to update your local branch)  
Apple ~ ~/Desktop2/HelloGit git (develop :9) > git pull origin develop ● system 0 0  
hint: Pulling without specifying how to reconcile divergent branches is  
hint: discouraged. You can squelch this message by running one of the following  
hint: commands sometime before your next pull:  
hint:  
hint:   git config pull.rebase false # merge (the default strategy)  
hint:   git config pull.rebase true # rebase  
hint:   git config pull.ff only    # fast-forward only  
hint:  
hint: You can replace "git config" with "git config --global" to set a default  
hint: preference for all repositories. You can also pass --rebase, --no-rebase,  
hint: or --ff-only on the command line to override the configured default per  
hint: invocation.  
From https://github.com/SemyonovE/HelloGit  
 * branch            develop    -> FETCH_HEAD  
Updating e73e82a..0352106  
Fast-forward  
HelloGit.xcodeproj/project.pbxproj | 20 ++++++Main.swift | 20 ++++++  
HelloGit/Screens/Main.swift        | 20 ++++++  
HelloGit/Screens/OnboardingScreen.swift | 42 ++++++  
HelloGit/Screens/Screen.swift      | 29 ++++++  
HelloGit/main.swift               | 18 ++++++  
5 files changed, 129 insertions(+)  
create mode 100644 HelloGit/Screens/Main.swift  
create mode 100644 HelloGit/Screens/OnboardingScreen.swift  
create mode 100644 HelloGit/Screens/Screen.swift
```

Это второй способ, который мы обсуждали, когда знакомились с командой `rebase` выше. Просто переключиться на ветку `develop` и обновить ее согласно изменениям на удаленном репозитории. Теперь вернемся на ветку `feature/news_loader` и вольем в нее все то, что есть в `develop`, разрешив при этом конфликты.

```
/* Begin PBXSourcesBuildPhase section */  
298017052614BBA200971AD8 /* Sources */ = {  
    isa = PBXSourcesBuildPhase;  
    buildActionMask = 2147483647;  
    files = (  
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes  
=<<<<< HEAD (Current Change)  
293D97AE26186C1A00D7AB08 /* NewsModel.swift in Sources */,  
=====  
29763D172618426600322122 /* OnboardingScreen.swift in Sources */,  
>>>>> develop (Incoming Change)  
29763D1F2618535700322122 /* Main.swift in Sources */,  
2980170D2614BBA200971AD8 /* main.swift in Sources */,  
298017182614D18000971AD8 /* Button.swift in Sources */,  
293D97AD26186C1A00D7AB08 /* NewsLoader.swift in Sources */,  
298017192614D18000971AD8 /* Label.swift in Sources */,  
29763D14261840B300322122 /* Screen.swift in Sources */,  
);  
runOnlyForDeploymentPostprocessing = 0;  
};  
/* End PBXSourcesBuildPhase section */
```

У меня конфликт только один, решаю его по той же схеме, что и выше. Оставляя обе строчки по каждой из блоков. Перед тем, как коммитить слияние веток лучше запустить проект и проверить, что все было сделано верно и ошибок нет.

```

Screen Onboarding loaded
Screen Onboarding showed
USER TOUCHES NEXT BUTTON
Screen Main loaded
Screen Main showed
Loading news
-----
И земля содрогнулась | 20.03.2021
-----
В городе Энске произошло землетрясение 8
баллов.

Алексей Иванов
-----
Британские учёные | 27.03.2021
-----
В африке обнаружен новый вид белок!

Кирилл Тунгас
-----
Внимание | 28.03.2021

```

All Output

Все экраны отобразились, новости загрузились. Коммитим изменения и отправляем на удаленный репозиторий.

```

git add .
git status
git commit -m "Merged develop into feature/news_loader"
git push origin feature/news_loader

```

```

$ ~/Desktop2/HelloGit git merge ~1 +2 > git add .
$ ~/Desktop2/HelloGit git merge +3 > git status
On branch feature/news_loader
All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:
  modified:   HelloGit.xcodeproj/project.pbxproj
  new file:   HelloGit/Screens/OnboardingScreen.swift
  modified:   HelloGit/main.swift

$ ~/Desktop2/HelloGit git merge +3 > git commit -m "Merged develop into feature/news_loader"
[feature/news_loader 4e5a098] Merged develop into feature/news_loader
$ ~/Desktop2/HelloGit git push origin feature/news_loader
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 8 threads  Hunk 1 : Lines 7-14
Compressing objects: 100% (6/6), done.    objects = {
Writing objects: 100% (6/6), 912 bytes | 912.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/SemyonovE/HelloGit.git
  4aa5790..4e5a098  feature/news_loader -> feature/news_loader

```

Возвращаемся в GitHub и видим, что теперь ветка с новостями может быть влита в develop.

## Added news loader #5

[Open](#) SemyonovE wants to merge 2 commits into `develop` from `feature/news_loader`

Conversation 0 Commits 2 Checks 0 Files changed 4

SemyonovE commented 8 minutes ago

No description provided.

Owner ...

Added news loader 4aa5790

Merged develop into feature/news\_loader 4e5a098

Add more commits by pushing to the `feature/news_loader` branch on [SemyonovE>HelloGit](#).

Continuous integration has not been set up  
GitHub Actions and [several other apps](#) can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch  
Merging can be performed automatically.

[Merge pull request](#) You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

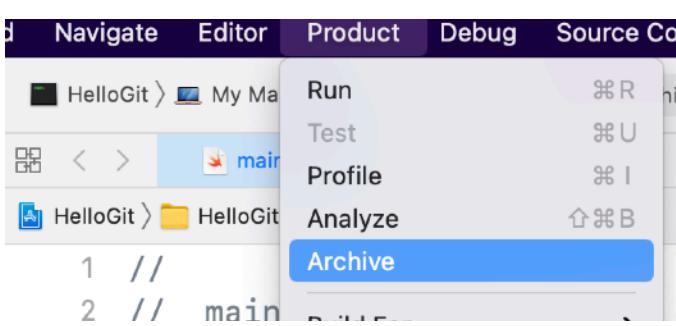
Снова переключаемся на первого разработчика. Он у нас будет главнее, так как начинал этот проект и ему поручено отдать текущую реализацию на тестирование.

```
git checkout develop
git pull origin develop
git checkout -b release
git push origin release
```

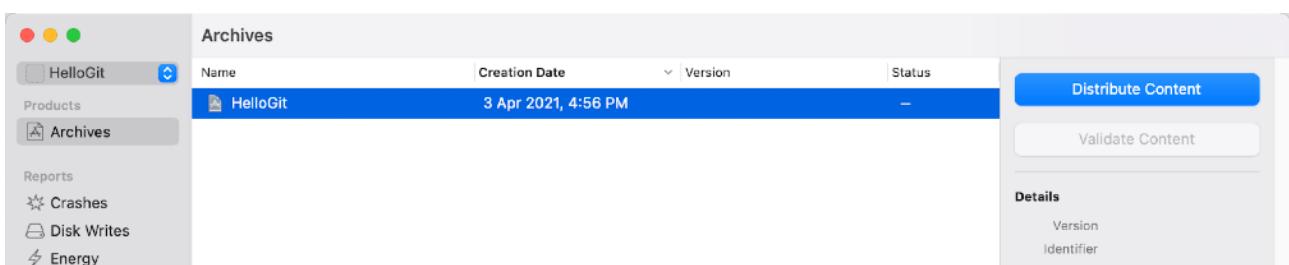
```

Apple Macbook Pro: ~ /De/HelloGit git (feature/onboarding) > git checkout develop
Switched to branch 'develop'                                     ● system 0 04:1
Apple Macbook Pro: ~ /De/HelloGit git (develop) > git pull origin develop
hint: Pulling without specifying how to reconcile divergent branches is
hint: discouraged. You can squelch this message by running one of the following
hint: commands sometime before your next pull:
hint:
hint:   git config pull.rebase false # merge (the default strategy)
hint:   git config pull.rebase true # rebase
hint:   git config pull.ff only    # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 18 (delta 3), reused 16 (delta 3), pack-reused 0
Unpacking objects: 100% (18/18), 3.85 KiB | 303.00 KiB/s, done.
From https://github.com/SemyonovE/HelloGit
 * branch            develop      -> FETCH_HEAD
   fbc473f..4e3ab11  develop      -> origin/develop
Updating fbc473f..4e3ab11
Fast-forward
HelloGit.xcodeproj/project.pbxproj | 20 ++++++-----+
HelloGit/NewsLoader/NewsLoader.swift | 30 ++++++-----+
HelloGit/NewsLoader/NewsModel.swift | 30 ++++++-----+
HelloGit/Screens/Main.swift         |  4 +--+
HelloGit/Screens/OnboardingScreen.swift | 42 ++++++-----+
HelloGit/main.swift                | 18 ++++++-----+
6 files changed, 143 insertions(+), 1 deletion(-)
create mode 100644 HelloGit/NewsLoader/NewsLoader.swift
create mode 100644 HelloGit/NewsLoader/NewsModel.swift
create mode 100644 HelloGit/Screens/OnboardingScreen.swift
Apple Macbook Pro: ~ /De/HelloGit git (develop) > git checkout -b release
Switched to a new branch 'release'                                ● system 0 04:5
Apple Macbook Pro: ~ /De/HelloGit git (release) > git push origin release
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'release' on GitHub by visiting:
remote:   https://github.com/SemyonovE/HelloGit/pull/new/release
remote:
To https://github.com/SemyonovE/HelloGit.git
 * [new branch] release -> release

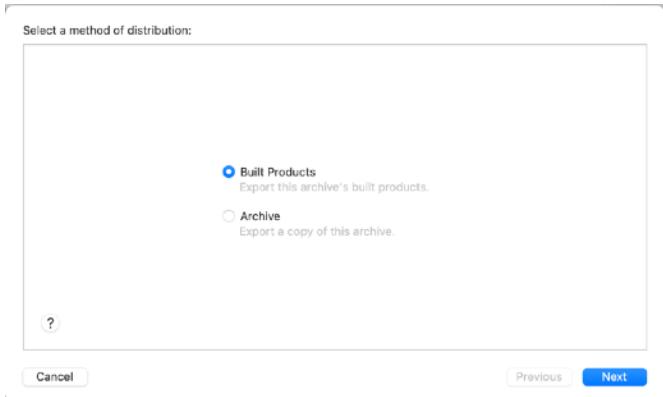
```



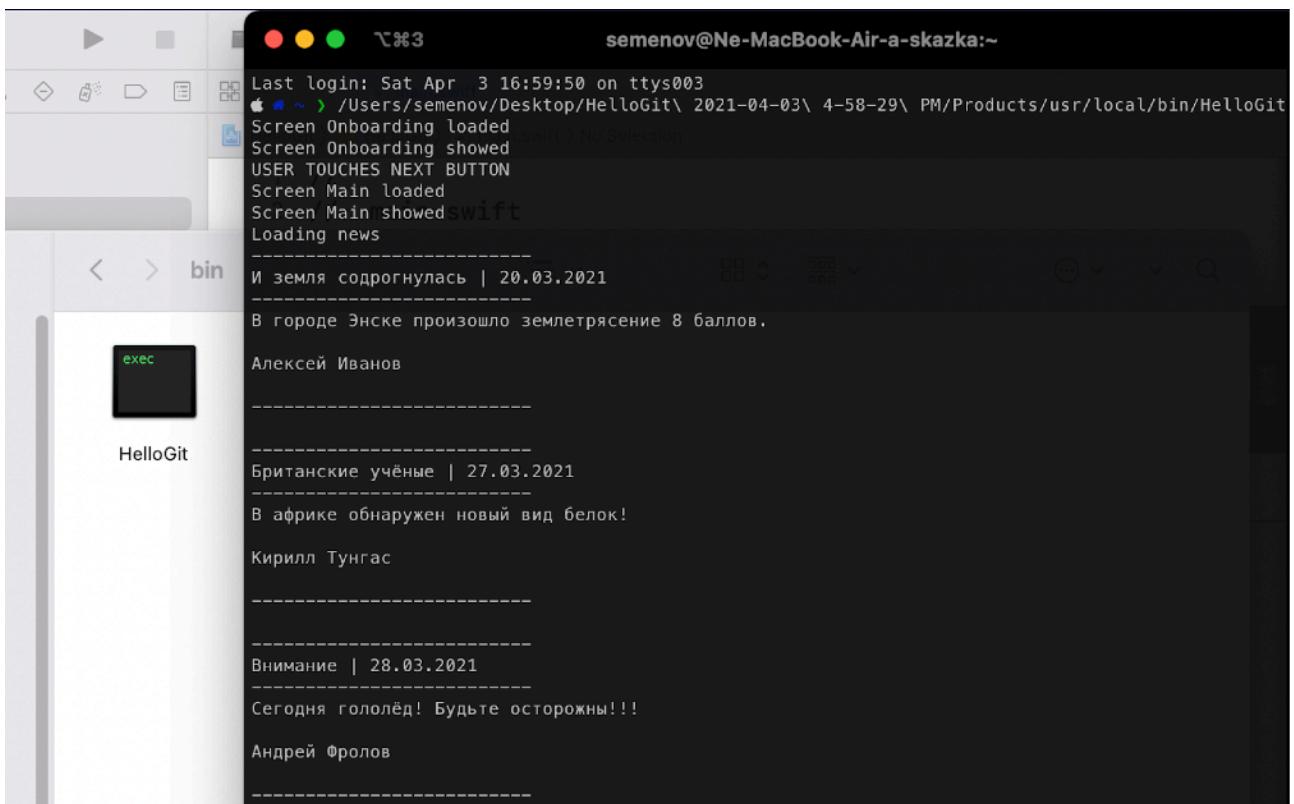
Теперь можно сделать сборку нашего приложения для тестировщика. Так как у нас простое консольное приложение, давайте рассмотрим для примера, как бы это можно было сделать. Открываем проект в Xcode и в панели меню выбираем Product>Archive. В появившемся окне нажимаем “Distribute Content”



Укажем, что нам нужна именно сборка нашего проекта и выберем место его сохранения, например рабочий стол.



В появившейся папке на рабочем столе, где-то глубоко в иерархии папок можно найти наше консольное приложение и запустить его. Лучше сделать это не щелкнув дважды по приложению, а перетянув его в терминал. Тогда в конце выполнения приложение не закроется и мы сможем увидеть результат его работы (отображение списка новостей).



Красота, все работает! На мгновение представим, что мы тот самый тестировщик, и допустим нам кажется ошибочным то, как выводится информация о нажатии пользователя на кнопку. Или по техническому заданию должно было быть иначе, но разработчики не обратили на это внимание. Говорим разработчикам поправить.

Так как этот функционал делал первый разработчик, ему его и исправлять. Он уже находится на нужной release ветке, поэтому ничего стягивать на этот раз дополнительного не нужно. Нужно создать отдельную ветку с багом и поправить вывод в консоль.

```
git checkout -b bugfix/user_touches_log
```

```
apple ~ ~/De/HelloGit git > release > git checkout -b bugfix/user_touches_log
Switched to a new branch 'bugfix/user_touches_log'
```

Далее поправим сам вывод в проекте.

```
print("The user clicks the Next button")
```

Закоммитим изменения и отправим на удаленный репозиторий.

```
git commit -m "Fixed user touches log text"  
git push origin bugfix/user_touches_log
```

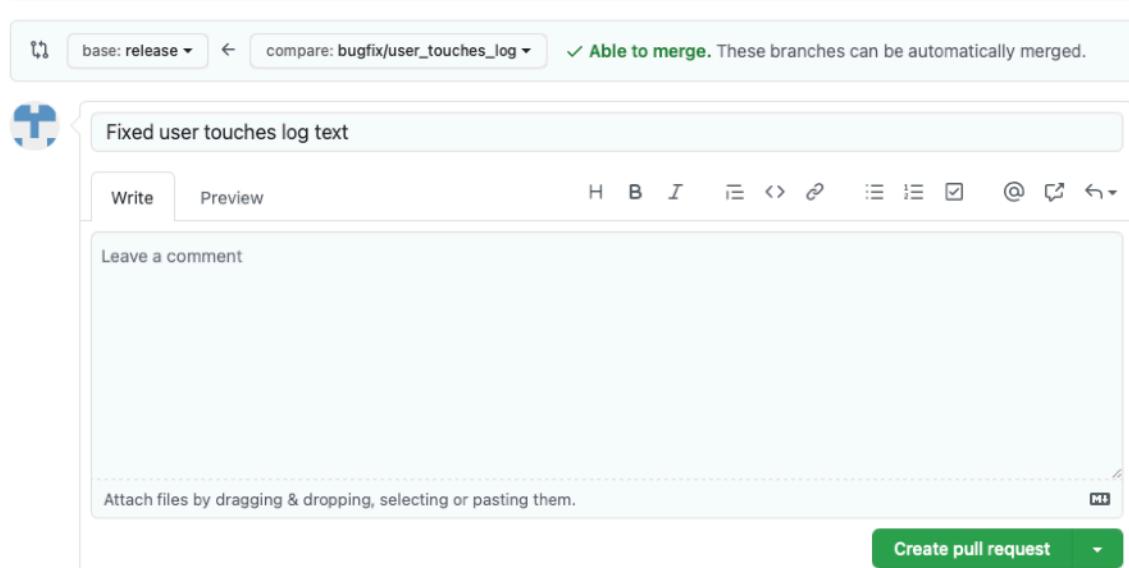
```
apple ~ ~/Desktop>HelloGit $ P bugfix/user_touches_log > git add .          ● system ⌂ 05:06:16 PM  
apple ~ ~/Desktop>HelloGit $ P bugfix/user_touches_log +1 > git status           ● system ⌂ 05:07:17 PM  
On branch bugfix/user_touches_log  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    modified:   HelloGit/main.swift  
  
apple ~ ~/Desktop>HelloGit $ P bugfix/user_touches_log +1 > git commit -m "Fixed user touches log text"          Added  
[bugfix/user_touches_log dba3079] Fixed user touches log text  
  1 file changed, 1 insertion(+), 1 deletion(-)          Merge  
apple ~ ~/Desktop>HelloGit $ P bugfix/user_touches_log > git push origin bugfix/user_touches_log            orig  
Enumerating objects: 7, done.          orig  
Counting objects: 100% (7/7), done.          Initial  
Delta compression using up to 8 threads  
Compressing objects: 100% (4/4), done.  
Writing objects: 100% (4/4), 397 bytes | 397.00 KiB/s, done.  
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0  
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.  
remote:  
remote: Create a pull request for 'bugfix/user_touches_log' on GitHub by visiting:  
remote:     https://github.com/SemyonovE/HelloGit/pull/new/bugfix/user_touches_log  
remote:  
To https://github.com/SemyonovE/HelloGit.git  
 * [new branch]      bugfix/user_touches_log -> bugfix/user_touches_log
```

Снова создаем ПР и отдаем его проверить второму разработчику (других у нас на проекте нет, иначе бы мы могли выбирать или отправить ведущему разработчику, но так как мы и есть тот ведущий разработчик на проекте, мы можем даже сами ревьюить свой код, хотя на реальном проекте так лучше не делать).

Обратите внимание, что здесь влиять мы будем именно в release ветку, потому что в develop ветке разработка может продолжаться и там будет уже новый код, не предназначенный для сборки.

#### Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



Хорошо, код влит. Стягиваем изменения в ветки release с удаленного репозитория

```
git checkout release  
git pull origin release
```

```
apple ~ ~/Desktop/HelloGit git bugfix/user_touches_log > git checkout release  
Switched to branch 'release'  
apple ~ ~/Desktop/HelloGit git release > git pull origin release  
hint: Pulling without specifying how to reconcile divergent branches is  
hint: discouraged. You can squelch this message by running one of the following  
hint: commands sometime before your next pull:  
hint:  
hint:   git config pull.rebase false # merge (the default strategy)  
hint:   git config pull.rebase true # rebase  
hint:   git config pull.ff only    # fast-forward only  
hint:  
hint: You can replace "git config" with "git config --global" to set a default  
hint: preference for all repositories. You can also pass --rebase, --no-rebase,  
hint: or --ff-only on the command line to override the configured default per  
hint: invocation.  
remote: Enumerating objects: 1, done.  
remote: Counting objects: 100% (1/1), done.  
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (1/1), 641 bytes | 320.00 KiB/s, done.  
From https://github.com/SemyonovE/HelloGit  
 * branch            release    -> FETCH_HEAD  
   4e3ab11..7510e8d  release    -> origin/release  
Updating 4e3ab11..7510e8d  
Fast-forward  
 HelloGit/main.swift | 2 +-  
  1 file changed, 1 insertion(+), 1 deletion(-)
```

и снова отправляем сборку на тест. Допустим теперь всё отлично и тестировщик дает отмашку влиять код в главную ветку.

Возвращаемся в GitHub и создаем ПР уже из release в master. Всегда нужно использовать ПР при вливании кода в ветку рангом (приоритетом) выше. В данном случае у нас master - самая главная ветка, после нее идет ветка release, следом develop, а затем уже равнозначно все остальные ветки фич и багов. Поэтому несмотря на то, что код уже протестирован, корректно будет влить его через ПР, который будет скорее фиктивный.

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

 base: main ▾ ← compare: release ▾ ✓ Able to merge. These branches can be automatically merged.

Код влит и теперь все кто захотят скачать наше приложение будут скачивать уже более свежую версию.

Спокойно возвращаемся в терминал и продолжаем работу над проектом. Для этого прежде всего актуализируем код develop ветки, просто вмержив в нее ветку master (можно без ПР, потому что мы льем код из более главной ветки). После чего отправим эти изменения на удаленный репозиторий

```
git checkout main  
git pull origin main  
git checkout develop  
git pull origin develop  
    git merge main  
git push origin develop
```

```
MacBook-Pro:HelloGit SemyonovE % git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.    Onboarding Screen
MacBook-Pro:HelloGit SemyonovE % git pull origin main
hint: Pulling without specifying how to reconcile divergent branches is
hint: discouraged. You can squelch this message by running one of the following
hint: commands sometime before your next pull:      Argument 2 {button in...}
hint:
hint:     git config pull.rebase false # merge (the default strategy)
hint:     git config pull.rebase true  # rebase
hint:     git config pull.ff only    # fast-forward only
hint:                                         Padding
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 617 bytes | 308.00 KiB/s, done.
From https://github.com/SemyonovE/HelloGit
 * branch           main      -> FETCH_HEAD
   82567c9..f25411c  main      -> origin/main
Updating 82567c9..f25411c
Fast-forward
HelloGit.xcodeproj/project.pbxproj      | 52 ++++++-----+
HelloGit/Components/Button.swift        | 39 ++++++-----+
HelloGit/Components/Label.swift         | 30 ++++++-----+
HelloGit/NewsLoader/NewsLoader.swift    | 30 ++++++-----+
HelloGit/NewsLoader/NewsModel.swift     | 30 ++++++-----+
HelloGit/Screens/Main.swift            | 22 ++++++-----+
HelloGit/Screens/OnboardingScreen.swift| 42 ++++++-----+
HelloGit/Screens/Screen.swift          | 29 ++++++-----+
HelloGit/main.swift                   | 17 ++++++-----+
9 files changed, 290 insertions(+), 1 deletion(-)
create mode 100644 HelloGit/Components/Button.swift
create mode 100644 HelloGit/Components/Label.swift
create mode 100644 HelloGit/NewsLoader/NewsLoader.swift
create mode 100644 HelloGit/NewsLoader/NewsModel.swift
create mode 100644 HelloGit/Screens/Main.swift
create mode 100644 HelloGit/Screens/OnboardingScreen.swift
create mode 100644 HelloGit/Screens/Screen.swift
MacBook-Pro:HelloGit SemyonovE % git checkout develop
Switched to branch 'develop'
MacBook-Pro:HelloGit SemyonovE % git pull origin develop
hint: Pulling without specifying how to reconcile divergent branches is
hint: discouraged. You can squelch this message by running one of the following
hint: commands sometime before your next pull:
hint:
hint:     git config pull.rebase false # merge (the default strategy)
hint:     git config pull.rebase true  # rebase
hint:     git config pull.ff only    # fast-forward only
hint:                                         Padding
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
From https://github.com/SemyonovE/HelloGit
 * branch           develop      -> FETCH_HEAD
Already up to date.
MacBook-Pro:HelloGit SemyonovE % git merge main
Updating 4e3ab11..f25411c
Fast-forward
HelloGit/main.swift | 2 ++
1 file changed, 1 insertion(+), 1 deletion(-)
MacBook-Pro:HelloGit SemyonovE % git push origin develop
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/SemyonovE/HelloGit.git
  4e3ab11..f25411c  develop -> develop
```

Получилась длинная цепочка действий, но каждый раз она будет одинаковой (если, конечно, не будет конфликтов при слиянии веток) и делать это придется только после обновления приложения. Теперь мы отправили все правки в удаленную ветку `develop` и можно уведомить остальных разработчиков подтянуть себе эти изменения.

Возвращаемся к проекту. Допустим теперь у нас есть задача к следующему релизу: добавить на экране приветствия еще подзаголовок и поменять цвет кнопки на синий. Заводим новую ветку и вносим изменения.

```
git checkout -b feature/onboard_redesign
```

```
apple ~ ~/Desktop/HelloGit git (feature/onboard_redesign) > git checkout -b feature/onboard_redesign
Switched to a new branch 'feature/onboard_redesign'
```

```
private let secondSubtitleLabel = Label(
    title: "And you can read news! :)",
    textColor: "light-gray"
)
private let launchButton = Button(
    title: "Launch",
    backgroundColor: "blue"
)
```

Но допустим, пока мы вносим эти изменения нам прилетает срочная задача от тестировщика. Оказалось, когда мы отдавали в релиз приложение, тестировщик не заметил, что новости отображаются не совсем красиво. А ждать до следующего релиза, пока это будет поправлено не дело. Люди уже качают наше приложение и видят невнятное описание новостей. Нужно срочно исправить. Когда исправлять баг нужно сразу в релизной версии (в ветке `master`), нам нужно создать `hotfix` ветку, чтобы избежать лишней бюрократии и максимально быстро влить правки. Но у нас еще не доделана текущая задача, мы добавили все что надо, но не протестирували, на что у нас нет времени. Для таких случаев в `git` есть специальное хранилище промежуточного кода, чтобы не делать недоделанный коммит. Называется это хранилище `stash` (стеш). Проверим состояние `git` и отправим все изменения в стеш

```
git status
git stash push -m "Onboard redesign"
git status
```

```
apple ~ ~/Desktop/HelloGit git (feature/onboard_redesign) > git status
On branch feature/onboard_redesign
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   HelloGit/Screens/OnboardingScreen.swift

no changes added to commit (use "git add" and/or "git commit -a")
apple ~ ~/Desktop/HelloGit git (feature/onboard_redesign) !1 > git stash push -m "Onboard redesign"
Saved working directory and index state On onboard_redesign: Onboard redesign
apple ~ ~/Desktop/HelloGit git (feature/onboard_redesign) *1 > git status
On branch feature/onboard_redesign
nothing to commit, working tree clean
```

Переключаемся на `master` ветку, заводим `hotfix` ветку

```
git checkout main  
git checkout -b hotfix/news_description
```

```
Apple ~ ~/Desktop/HelloGit git [P feature/onboard_redesign *1 > git checkout main  
Switched to branch 'main'  
Your branch is up to date with 'origin/main'.  
Apple ~ ~/Desktop/HelloGit git [P main *1 > git checkout -b hotfix/news_description  
Switched to a new branch 'hotfix/news_description'
```

И ВНОСИМ ИЗМЕНЕНИЯ В ОПИСАНИЕ НОВОСТЕЙ

```
var description: String {  
    ....  
    -----  
    \((title) | \((date)  
    -----  
    \((text)  
  
    Автор: \((author)  
  
    ....  
}
```

Коммитим изменения, отправляем на удаленный репозиторий и создаем ПР.

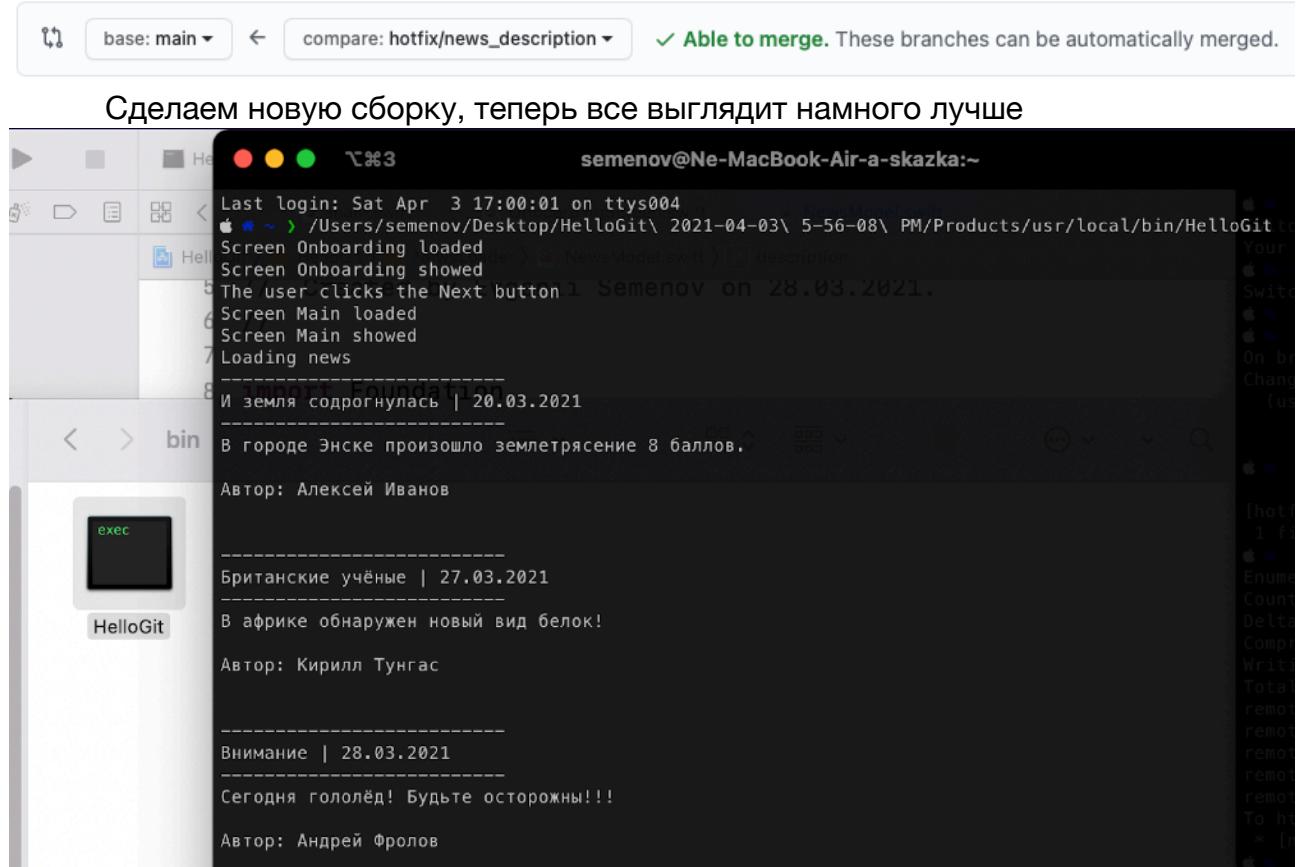
```
git add .  
git status  
git commit -m "Fixed news description"  
git push origin hotfix/news_description
```

```
Apple ~ ~/Desktop/HelloGit git [P hotfix/news_description *1 > git add .  
Apple ~ ~/Desktop/HelloGit git [P hotfix/news_description *1 +1 > git status  
On branch hotfix/news_description  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    modified:   HelloGit/NewsLoader/NewsModel.swift  
  
Apple ~ ~/Desktop/HelloGit git [P hotfix/news_description *1 +1 > git commit -m "Fixed news description"  
[hotfix/news_description 3c7d930] Fixed news description  
1 file changed, 1 insertion(+), 2 deletions(-)  
Apple ~ ~/Desktop/HelloGit git [P hotfix/news_description *1 > git push origin hotfix/news_description  
Enumerating objects: 9, done.  
Counting objects: 100% (9/9), done.  
Delta compression using up to 8 threads  
Compressing objects: 100% (5/5), done.  
Writing objects: 100% (5/5), 485 bytes | 485.00 KiB/s, done.  
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0  
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.  
remote:  
remote: Create a pull request for 'hotfix/news_description' on GitHub by visiting:  
remote:     https://github.com/SemyonovE/HelloGit/pull/new/hotfix/news_description  
remote:  
To https://github.com/SemyonovE/HelloGit.git  
 * [new branch]      hotfix/news_description -> hotfix/news_description
```

Хотфикс мы вливаем сразу в мастер. Иногда все же хотфикс создают в ветке release, чтобы была возможность протестировать код перед отправкой, но у нас все достаточно просто и понятно, можно сразу лить в master.

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



Можно вернуться к недоделанной задаче и продолжить ее выполнение.

`git checkout feature/onboard_redesign`

```
apple ~ ~/Desktop/HelloGit git hotfix/news_description *1 > git checkout feature/onboard_redesign
Switched to branch 'feature/onboard_redesign'
```

Чтобы вернуться к закешированному коду нужно вспомнить какие его части были сохранены, ведь может оказаться, что их будет несколько. Воспользуемся командой

`git stash list`

```
stash@{0}: On onboard_redesign: Onboard redesign
(END)
```

У нас хранится только один слепок кода, поэтому, чтобы забрать этот код из сохраненного стеша у нас есть два пути: извлечь его из стеша, удалив в этом хранилище или просто скопировать, но чтобы он на всякий случай остался в стеше. В первом случае нам следует использовать

## git stash pop stash@{0}

```
apple ~ ~/Desktop/HelloGit git feature/onboard_redesign *1 > git stash pop stash@{0}
On branch feature/onboard_redesign
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   HelloGit/Screens/OnboardingScreen.swift

no changes added to commit (use "git add" and/or "git commit -a")
Dropped stash@{0} (b0ba5796a522f8d88f73546feeb352c8bd2d3ed3)
```

Тогда наш код окажется в текущей ветке и будет удален из стеша, а для второго случая можно использовать команду

## git stash apply stash@{0}

```
apple ~ ~/Desktop/HelloGit git feature/onboard_redesign *1 > git stash apply stash@{0}
On branch feature/onboard_redesign
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   HelloGit/Screens/OnboardingScreen.swift

no changes added to commit (use "git add" and/or "git commit -a")
```

Мы можем применить изменения не очищая их в стеше. Я воспользуюсь первым способом, потому что мне хранить эти изменения в дальнейшем не нужно.

Тестируем, коммитим, отправляем, создаем ПР и влияем код в develop.

## git add .

git commit -m "Changed button color and added second subtitle on  
onboarding screen"

git push origin feature/onboard\_redesign

```
apple ~ ~/Desktop/HelloGit git feature/onboard_redesign !1 > git add .
apple ~ ~/Desktop/HelloGit git feature/onboard_redesign +1 > git commit -m "Changed button color and
added second subtitle on onboarding screen"
[feature/onboard_redesign 35f5238] Changed button color and added second subtitle on onboarding s
creen
1 file changed, 3 insertions(+), 1 deletion(-)
apple ~ ~/Desktop/HelloGit git feature/onboard_redesign > git push origin feature/onboard_redesign
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 571 bytes | 571.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
remote:
remote: Create a pull request for 'feature/onboard_redesign' on GitHub by visiting:
remote:     https://github.com/SemyonovE/HelloGit/pull/new/feature/onboard_redesign
remote:
To https://github.com/SemyonovE/HelloGit.git
 * [new branch]      feature/onboard_redesign -> feature/onboard_redesign
```

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



Также нужно не забыть влить новые правки из master ветки в ветку develop.

**git checkout main  
git pull origin main**

```
Apple ~ ~/Desktop/HelloGit git P feature/onboard_redesign > git checkout main ● system
Switched to branch 'main'
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)
Apple ~ ~/Desktop/HelloGit git P main :2 > git pull origin main ● system
hint: Pulling without specifying how to reconcile divergent branches is
hint: discouraged. You can squelch this message by running one of the following
hint: commands sometime before your next pull:
hint:
hint:   git config pull.rebase false # merge (the default strategy)
hint:   git config pull.rebase true # rebase
hint:   git config pull.ff only    # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
From https://github.com/SemyonovE/HelloGit
 * branch           main      -> FETCH_HEAD
Updating f25411c..b072ca5
Fast-forward
  HelloGit/NewsLoader/NewsModel.swift | 3 +-
  1 file changed, 1 insertion(+), 2 deletions(-)
  title: "Welcome my app"
  1 file changed, 1 insertion(+), 1 deletion(-)
  le: "This app will help you to learn
```

**git checkout develop  
git pull origin develop**

```
Apple ~ ~/Desktop/HelloGit git P main > git checkout develop ● system
Switched to branch 'develop'
Apple ~ ~/Desktop/HelloGit git P develop > git pull origin develop ● system
hint: Pulling without specifying how to reconcile divergent branches is
hint: discouraged. You can squelch this message by running one of the following
hint: commands sometime before your next pull:
hint:
hint:   git config pull.rebase false # merge (the default strategy)
hint:   git config pull.rebase true # rebase
hint:   git config pull.ff only    # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 674 bytes | 337.00 KiB/s, done. Q Search
From https://github.com/SemyonovE/HelloGit
 * branch           develop      -> FETCH_HEAD
   f25411c..32edb6e  develop      -> origin/develop
Updating f25411c..32edb6e
Fast-forward
  HelloGit/Screens/OnboardingScreen.swift | 14t+++--"Welcome my app"
  1 file changed, 13 insertions(+), 1 deletion(-)
  le: "This app will help you to learn
```

```
git merge main
```

```
apple ~ ~/Desktop/HelloGit git [P] develop > git merge main
Merge made by the 'recursive' strategy.
HelloGit/NewsLoader/NewsModel.swift | 3 +-- launch app")
1 file changed, 1 insertion(+), 2 deletions(-)
```

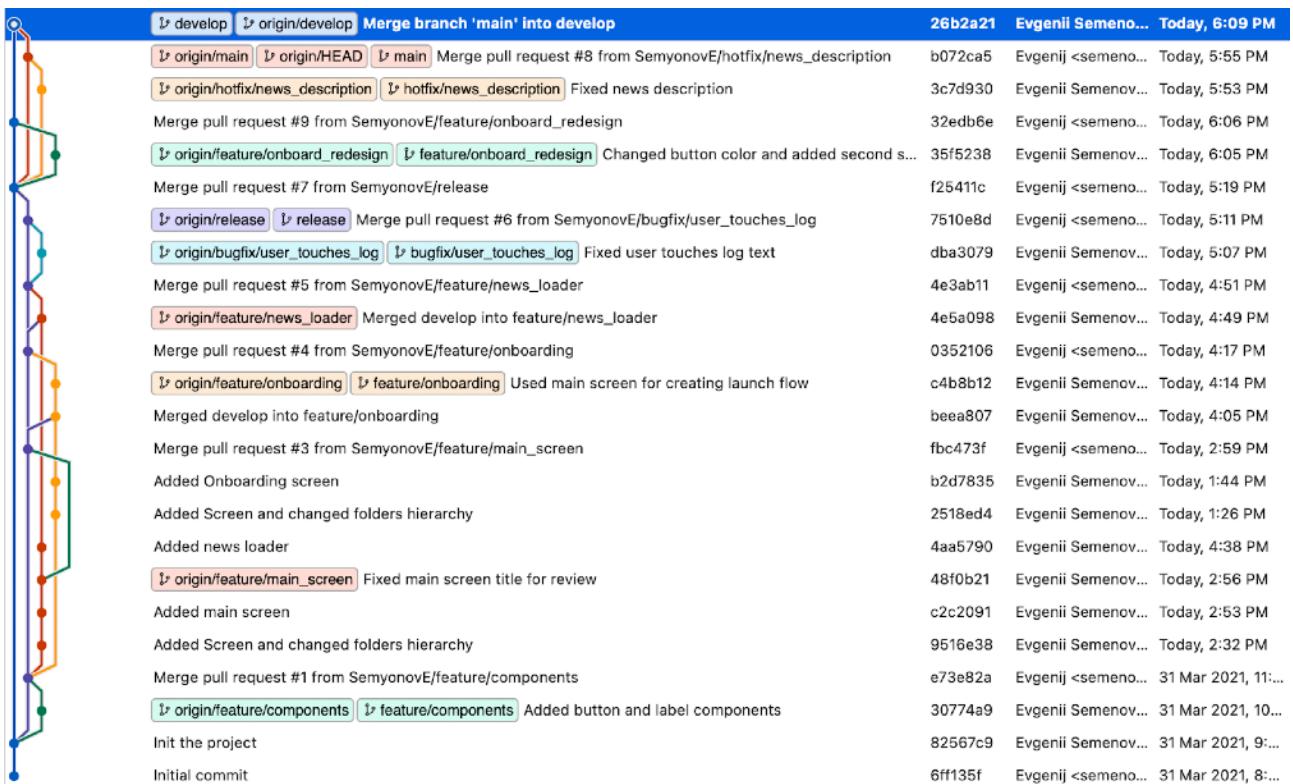
Иногда, когда мы будем мерджить код из ветки в ветку нам будет предложено задать описание этому слиянию. В этом случае чаще всего описание оставляют стандартное и выйти из редактора описания (если у вас открылось описание в терминале) можно набрав ':q' и нажав Enter.

```
git push origin develop
```

```
apple ~ ~/Desktop/HelloGit git [P] develop > git push origin develop
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 362 bytes | 362.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/SemyonovE/HelloGit.git
  32edb6e..26b2a21  develop -> develop
```

Все, мы завершили работы по проекту и рассмотрели основные процессы, которые происходят на проектах. Также мы рассмотрели некоторые дополнительные команды, облегчающие разработку. Дальше все будет повторяться примерно в том же ключе. Я специально не рассматривал внештатные ситуации на проекте, когда например мы влили в develop нерабочий код, или случайно влили сразу develop в ветку master, но обычно в таком случае данные проблемы будут разрешать старшие разработчики, либо удаленный репозиторий будет настроен так, что у вас не будет прав на создание нежелательных вливаний кода.

Также для работы с git обычно используют какую-то программу-оболочку, которая позволяет делать все операции быстрее и проще. Но знание того, как это работает изнутри, должно быть у разработчика и он должен уметь управлять git через консоль, потому что бывают ситуации когда и оболочка не поможет. Сейчас мы прошли вот такой вот длинный путь, отражу дерево изменений в программе Sourcetree, это бесплатная программа, которую использую в своей работе я.



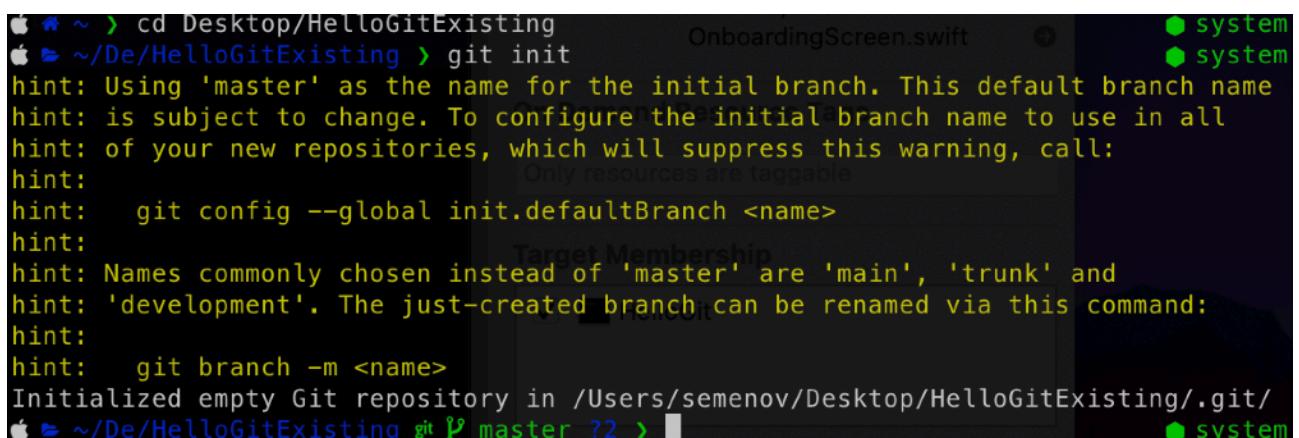
			26b2a21	Evgenii Semeno...	Today, 6:09 PM
origin/main	origin/HEAD	main	b072ca5	Evgenij <semeno...	Today, 5:55 PM
origin/fix/news_description	fix/news_description	Merge pull request #8 from SemyonovE/news_description	3c7d930	Evgenii Semenov...	Today, 5:53 PM
Merge pull request #9 from SemyonovE/feature/onboard_redesign			32edb6e	Evgenij <semeno...	Today, 6:06 PM
origin/feature/onboard_redesign	feature/onboard_redesign	Changed button color and added second s...	35f5238	Evgenii Semenov...	Today, 6:05 PM
Merge pull request #7 from SemyonovE/release			f25411c	Evgenij <semeno...	Today, 5:19 PM
origin/release	release	Merge pull request #6 from SemyonovE/bugfix/user_touches_log	7510e8d	Evgenij <semeno...	Today, 5:11 PM
origin/bugfix/user_touches_log	bugfix/user_touches_log	Fixed user touches log text	dba3079	Evgenii Semenov...	Today, 5:07 PM
Merge pull request #5 from SemyonovE/feature/news_loader			4e3ab11	Evgenij <semeno...	Today, 4:51 PM
origin/feature/news_loader	Merged develop into feature/news_loader		4eb0a98	Evgenii Semenov...	Today, 4:49 PM
Merge pull request #4 from SemyonovE/feature/onboarding			0352106	Evgenij <semeno...	Today, 4:17 PM
origin/feature/onboarding	feature/onboarding	Used main screen for creating launch flow	c4b8b12	Evgenii Semenov...	Today, 4:14 PM
Merged develop into feature/onboarding			beea807	Evgenii Semenov...	Today, 4:05 PM
Merge pull request #3 from SemyonovE/feature/main_screen			fbc473f	Evgenij <semeno...	Today, 2:59 PM
Added Onboarding screen			b2d7835	Evgenii Semenov...	Today, 1:44 PM
Added Screen and changed folders hierarchy			2518ed4	Evgenii Semenov...	Today, 1:26 PM
Added news loader			4aa5790	Evgenii Semenov...	Today, 4:38 PM
origin/feature/main_screen	feature/main_screen	Fixed main screen title for review	48f0b21	Evgenii Semenov...	Today, 2:56 PM
Added main screen			c2c2091	Evgenii Semenov...	Today, 2:53 PM
Added Screen and changed folders hierarchy			9516e38	Evgenii Semenov...	Today, 2:32 PM
Merge pull request #1 from SemyonovE/feature/components			e73e82a	Evgenij <semeno...	31 Mar 2021, 11:...
origin/feature/components	feature/components	Added button and label components	30774a9	Evgenii Semenov...	31 Mar 2021, 10:...
Init the project			82567c9	Evgenii Semenov...	31 Mar 2021, 9:...
Initial commit			6ff135f	Evgenij <semeno...	31 Mar 2021, 8:...

# Внедрение git в существующий проект

Мы рассмотрели вариант, когда проект создаётся с нуля и для него создаётся свой пустой репозиторий. Но иногда оказывается, что проект уже начал, над ним ведется работа, но у нас прежде не было никакой истории изменений, потому что для проекта не был настроен git. В таком случае нам следует поступить немного другим способом, чтобы создать удаленный репозиторий и связать с ним наш локальный проект.

Для начала нужно проинициализировать (создать) пустой репозиторий внутри папки xcode-проекта. Для этого достаточно перейти в него в терминале, так же как мы делали это ранее (`cd ПУТЬ_В_ПАПКУ_ПРОЕКТА`) и вызвать в терминале команду

```
git init
```

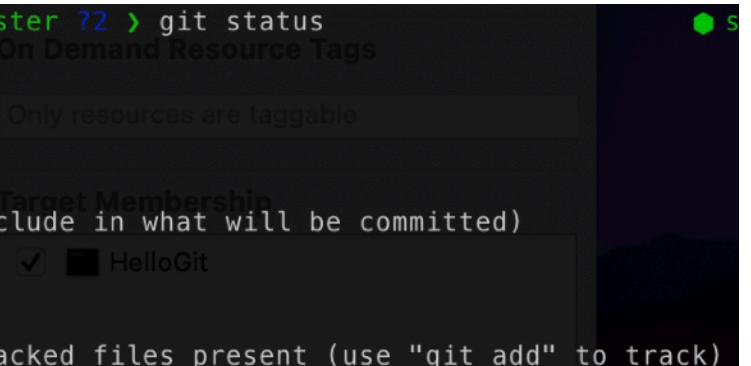


```
cd Desktop/HelloGitExisting
~/De/HelloGitExisting > git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:   Only resources are taggable
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/semenov/Desktop/HelloGitExisting/.git/
~/De/HelloGitExisting git: master ? 2 >
```

Далее, если вызвать команду

### git status

мы увидим что весь наш проект виден в системе, но пока не подготовлен к первому коммиту.



```
apple ~ ~/De>HelloGitExisting git [master ?2] > git status
On branch master
On Demand Resource Tags
Only resources are taggable
Untracked files:
(use "git add <file>..." to include in what will be committed)
  HelloGit.xcodeproj/
  HelloGit/
nothing added to commit but untracked files present (use "git add" to track)
```

Прежде чем мы сделаем первый коммит, вспомним про файл `.gitignore`, который мы ранее получили автоматически на сайте GitHub. Сейчас за нас этот файл нам никто не создаст, но он будет полезен нам в будущем, чтобы мы не сохраняли в репозиторий лишний мусор или личную информацию. Поэтому нам нужно где-то раздобыть этот файл. В интернете есть множество примеров, каждый программист может создать свой такой файл, потому как он просто состоит из списка фильтров, которые будут использованы для проверки всех папок и файлов внутри проекта прежде чем мы их закоммитим. Лишние, которые мы указали в этом файле просто не будут учтены (не будут добавлены в индекс).

Готовый пример такого файла можно найти по ссылке <https://github.com/github/gitignore/blob/master/Swift.gitignore>.

Выделяем все содержимое файла по ссылке и возвращаемся к нашему терминалу. Напишем две команды: для создания и открытия нового файла `.gitignore` в папке нашего проекта

```
touch .gitignore
open .gitignore
```



```
apple ~ ~/De>HelloGitExisting git [master ?2] > touch .gitignore
apple ~ ~/De>HelloGitExisting git [master ?3] > open .gitignore
```

В открывшемся окне вставляем весь скопированный код, сохраняем и закрываем текстовые редактор, который был открыт. Всё, мы успешно добавили данный файл в наш проект.

На самом деле вы можете не увидеть этот файл в папке проекта, потому что он системный и будет скрыт. Если вы хотите убедиться, что файл действительно есть, то переключить показ/скрытие системных файлов можно с помощью сочетания клавиш 'cmd+shift+.' (точка - клавиша с буквой 'Ю'). Либо мы снова можем вызвать команду

### git status

и увидеть, что добавился файл `.gitignore`.

```

apple ~ ~/De/HelloGitExisting git P master ?3 > git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    HelloGit.xcodeproj/
    HelloGit/

nothing added to commit but untracked files present (use "git add" to track)

```

Дальше всё как обычно. Используем

**git add .**

для подготовки файлов к коммиту и команду

**git commit -m "Init commit"**

для создания первого коммита со всеми данными вашего проекта. Сообщение конечно же может отличаться от предложенного, но обычно пишут что-то такое.

```

apple ~ ~/De/HelloGitExisting git P master +5 > git commit -m "Init commit"      ● system ○ 10:03:32 PM
[master (root-commit) e7978c9] Init commit
 5 files changed, 406 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 HelloGit.xcodeproj/project.pbxproj
 create mode 100644 HelloGit.xcodeproj/project.xcworkspace/contents.xcworkspacedata
 create mode 100644 HelloGit.xcodeproj/project.xcworkspace/xcshareddata/IDEWorkspaceChecks.plist
 create mode 100644 HelloGit/main.swift

```

На нашем компьютере всё готово к отправке кода. Далее необходимо создать пустой репозиторий проекта на сайте GitHub, чтобы было куда отправлять проект. Будем действовать по уже известной нам схеме. Заходим на главную GitHub и нажимаем “Start a project” (также можно найти кнопку “New” на странице

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner *	Repository name *
SemyonovE	HelloGitExisting

Great repository names are short and memorable. Need inspiration? How about [ubiquitous-engine](#)?

Description (optional)

Public  
Anyone on the internet can see this repository. You choose who can commit.

Private  
You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file  
This is where you can write a long description for your project. [Learn more](#).

Add .gitignore  
Choose which files not to track from a list of templates. [Learn more](#).

Choose a license  
A license tells others what they can and can't do with your code. [Learn more](#).

**Create repository**

вашего профиля, делают они аналогичное). При создании репозитория нам нужно снова указать его название и, если нужно, описание, после чего решить будет он приватным или публичным. И наконец блок с конфигурацией при инициализации. На этот раз не будем выбирать ни одного пункта, потому что нам нужно создать полностью пустой репозиторий (все необходимые настройки уже должны быть в нашем локальном репозитории).

Так как репозиторий пустой, нам будут предложены те команды, которые необходимо выполнить, чтобы наполнить этот репозиторий.

Quick setup — if you've done this kind of thing before

Set up in Desktop or  HTTPS  SSH <https://github.com/SemyonovE>HelloGitExisting.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a `README`, `LICENSE`, and `.gitignore`.

...or create a new repository on the command line

```
echo "# HelloGitExisting" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin https://github.com/SemyonovE>HelloGitExisting.git  
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/SemyonovE>HelloGitExisting.git  
git branch -M main  
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

Возвращаемся в терминал и выполняем необходимые команды. В моём случае ветка `master` уже есть и проект уже существует. Мне лишь необходимо добавить настройку с удалённым репозиторием

**git remote add origin https://github.com/SemyonovE>HelloGitExisting.git**

Здесь `origin` - название нашего удаленного репозитория. Это стандартное название, его можно поменять, а можно вообще добавить несколько удаленных репозиториев с разными именами. Далее сделаем `push` (отправку) проекта на GitHub

**git push origin master**

или просто

**git push**

ПОТОМУ ЧТО У НАС ВСЕ РАВНО ЕСТЬ ТОЛЬКО ОДНА ВЕТКА И ОДИН РЕПОЗИТОРИЙ.

```
apple ~ ~/De/HelloGitExisting git [master] > git remote add origin https://github.com/SemyonovE>HelloGitExisting.git  
apple ~ ~/De/HelloGitExisting git [master] > git push origin master  
Enumerating objects: 11, done.  
Counting objects: 100% (11/11), done.  
Delta compression using up to 8 threads  
Compressing objects: 100% (10/10), done.  
Writing objects: 100% (11/11), 4.00 KiB | 4.00 MiB/s, done.  
Total 11 (delta 0), reused 0 (delta 0), pack-reused 0  
To https://github.com/SemyonovE>HelloGitExisting.git  
 * [new branch]      master -> master  
apple ~ ~/De/HelloGitExisting git [master] >
```

Обновив страницу проекта на GitHub мы увидим, что он успешно загружен в удаленный репозиторий.

The screenshot shows a GitHub repository page for 'HelloGit'. At the top, there are buttons for 'master' (selected), '1 branch', '0 tags', 'Go to file', 'Add file', and a green 'Code' button. Below this is a table of commits:

Author	Commit Message	Time Ago
SemyonovE	Init commit	e7978c9 5 minutes ago
	HelloGit.xcodeproj	Init commit
	HelloGit	Init commit
	.gitignore	Init commit

Below the commits is a light blue bar with the text 'Help people interested in this repository understand your project by adding a README.' and a green 'Add a README' button. To the right, sections include 'About' (no description, website, or topics provided), 'Releases' (no releases published, 'Create a new release'), 'Packages' (no packages published, 'Publish your first package'), and 'Languages' (Swift 100.0%).

Дальше остается только настроить git-flow (создать необходимые общие ветки и описать правила) и работать над проектом уже согласно нему.

Примеры кода доступны по ссылке <https://github.com/SemyonovE/Swift.CourseUIKit>