



Спринт 3. Git

Яндекс Практикум

В этой шпаргалке вы найдёте основные команды, которые изучили в уроках про Git. Они пригодятся, когда вы будете сдавать проект на ревью, создавать ветку, готовить изменения, делать пул-реквест.



Некоторые термины и команды из списка не были затронуты на уроках. Мы добавили их сюда, чтобы вы были готовы к новым задачам, ведь Git'ом вы будете пользоваться и в дальнейшем, когда станете работать над реальными проектами.



Скачайте документ, чтобы обращаться к нему при необходимости, пока не доведёте навыки до автоматизма.

[Глоссарий](#)

[Список базовых команд](#)

[Pull Request](#)

[.gitignore](#)

[Что писать в сообщении коммита](#)

[Сообщение коммита должно быть конкретным и информативным](#)

[Используйте обезличенные глаголы в начале сообщения](#)

[Не пишите слишком длинные сообщения](#)

[Начинайте сообщение с заглавной буквы](#)

[Markdown: язык разметки](#)

[Заголовки](#)

[Выделение текста](#)

[Списки](#)

[Ссылки](#)

[Изображения](#)

Код

Цитаты

Горизонтальная линия

Глоссарий

Термин	Описание
Репозиторий (repository)	Хранилище вашего проекта. Включает все файлы и историю изменений.
Клонирование (clone)	Создание локальной копии существующего репозитория из удалённого источника.
Коммит (commit)	Зафиксированное состояние проекта в определённый момент времени. Говорят: «закомить эти изменения»
Ветка (branch)	Независимая линия разработки. В ней можно работать над новыми функциями или исправлениями, и это не повлияет на основной проект. Говорят: <i>брэнч, ветка.</i>
Слияние (merge)	Процесс интеграции изменений из одной ветки в другую. Говорят: <i>мердж, смерджить.</i>
Пул-реквест (pull request)	Запрос на внесение изменений из вашей ветки в основную ветку проекта на платформах вроде GitHub. Еще называют: <i>ПР (PR).</i>
Пул (pull)	Интеграция изменений из удалённого репозитория в локальную ветку. Говорят: <i>спулить ветку.</i>
Пуш (push)	Отправка локальных изменений в удалённый репозиторий. Говорят: <i>запушить в ветку.</i>

Термин	Описание
Индекс (index)	<p>Промежуточная область, в которую добавляются изменения перед выполнением коммита.</p> <p>Говорят: <i>проиндексируйте изменения.</i></p>
Фетч (fetch)	<p>Команда для получения изменений из удалённого репозитория без интеграции в локальную ветку.</p> <p>Говорят: <i>зафетчить репозиторий.</i></p>
Тег (tag)	<p>Метка для определённых коммитов. Чаще всего используется для маркировки версий релизов.</p> <p>Говорят: <i>поставь релизный тег.</i></p>
Стэш (stash)	<p>Временное сохранение изменений, которые не готовы к коммиту. Нужно, чтобы можно было чисто переключиться на другую ветку.</p> <p>Говорят: <i>застешить изменения.</i></p>
Конфликт слияния (merge conflict)	<p>Ситуация, когда Git не может автоматически объединить изменения из разных веток из-за противоречий между ними.</p> <p>Говорят: <i>пофиксить конфликты.</i></p>
Форк (fork)	<p>Копия чужого репозитория в вашем аккаунте на GitHub. Позволяет разрабатывать проект независимо.</p> <p>Говорят: <i>форкни этот проект.</i></p>
Ревью (review)	<p>Проверка и обсуждение кода в пул-реквесте другими разработчиками. Обычно проводят перед слиянием проекта.</p>

Список базовых команд

Команда	Описание
<code>git init</code>	Инициализировать новый git-репозиторий в текущей директории.
<code>git clone <repository-url></code>	Клонировать репозиторий из удалённого источника на локальный компьютер.
<code>git status</code>	Показать состояние изменений в рабочей директории и индексе.
<code>git add <file></code>	Добавить файл в индекс (подготовить к коммиту).
<code>git init</code>	Инициализировать новый git-репозиторий в текущей директории.
<code>git commit -m "commit message"</code>	Создать коммит (фиксацию изменений) с сообщением.
<code>git push [remote-name] [branch-name]</code>	Отправить коммиты в удалённый репозиторий. Обычно удалённый репозиторий имеет имя <code>origin</code>
<code>git push</code>	Отправить коммиты в удалённый репозиторий текущей ветки, если ранее команда была вызвана с именем удалённого репозитория.
<code>git pull [remote-name] [branch-name]</code>	Получить и интегрировать изменения из удалённого репозитория.
<code>git pull</code>	Получить и объединить изменения из удалённого репозитория с текущей веткой.
<code>git branch</code>	Показать список веток.
<code>git branch <branch-name></code>	Создать новую ветку.
<code>git checkout <branch-name></code>	Переключиться на указанную ветку.
<code>git checkout -b <branch-name></code>	Создать и переключиться на указанную ветку.
<code>git merge <branch></code>	Слить указанную ветку с текущей.
<code>git diff [branch1] [branch2]</code>	Показать различия между ветками.
<code>git log</code>	Просмотреть историю коммитов.
<code>git stash</code>	Сохранить изменения в стеке временных изменений.
<code>git stash pop</code>	Применить сохранённые изменения из стека.
<code>git fetch [remote-name]</code>	Получить изменения из удалённого репозитория, не объединяя их с локальными.
<code>git remote add <remote-name> <repository-url></code>	Добавить новый удалённый репозиторий под указанным именем.

Команда	Описание
<code>git tag</code>	Показать список тегов.
<code>git tag <tag-name></code>	Создать новый тег.
<code>git config --global user.name "Ваше Имя"</code>	Установить имя пользователя.
<code>git config --global user.email "ваш_email@example.com"</code>	Установить email.
<code>git config --list</code>	Показать все настройки.

Pull Request

Создание пул-реквеста (Pull Request) — это предложения изменений в коде, которые затем могут быть рассмотрены и приняты в основной репозиторий.

Ниже приведён алгоритм создания пул-реквеста в GitHub, однако аналогичные шаги применимы и к другим платформам, таким как GitLab или Bitbucket:

1. Создайте новую ветку

- Создайте новую ветку для изменений. Замените `<имя_новой_ветки>` на название, отражающее суть ваших изменений.

```
git checkout -b <имя_новой_ветки>
```

2. Внесите изменения

- Внесите изменения в код или содержимое репозитория. Это может быть исправление ошибки, добавление новой функциональности и тому подобное.

3. Добавьте изменения в индекс и сделайте коммит

- Добавьте изменённые файлы в индекс Git:

```
git add .
```

- Зафиксируйте изменения, создав коммит:

```
git commit -m "Краткое описание ваших изменений"
```

4. Отправьте изменения в ваш проект на GitHub

- Отправьте вашу ветку с изменениями в форкнутый репозиторий на GitHub:

```
git push origin <имя_новой_ветки>
```

5. Создайте пул реквест

- Перейдите на страницу репозитория на GitHub.
- Вы увидите кнопку Compare & pull request напротив вашей ветки. Нажмите на неё.
- Укажите краткое описание ваших изменений. При необходимости укажите любую дополнительную информацию или ссылки, помогающие понять суть изменений.
- Нажмите Create pull request.

После того, как вы отправите пул-реквест, ревьюер рассмотрит ваши изменения. Этот процесс так и называется «ревью», от английского `code review`.

Возможно, ревьюер попросит внести дополнительные корректировки.

Если ваш пул-реквест одобряют, можете объединить изменения с основной веткой репозитория.

`.gitignore`

Файл `.gitignore` используется в Git для исключения ненужных файлов и директорий из отслеживания. Для разработчиков и разработчиц iOS это особенно важно, чтобы избегать добавления временных файлов, личной информации и других ненужных данных в репозиторий.

Создание файла `.gitignore`

- Создайте в корне вашего проекта файл с именем `.gitignore`.
- Добавьте в него правила для игнорирования.

Примеры правил в `.gitignore`:

- Игнорировать все файлы в папке: `logs/`

- Игнорировать один файл: `config.env`
- Игнорировать все файлы с определённым расширением: `.log`
- Игнорировать все файлы в конкретной директории: `/temp/*`
- Не игнорировать файлы `.gitignore` и `.md`: `!.gitignore` , `!*.md`

Для простых проектов существуют заготовленные шаблоны.

- Собрать такой шаблон можно в gitignore.io
- По ссылке можно посмотреть сгенерированный шаблон.



Create useful .gitignore files for your project

macOS x

Xcode x

Swift x

CocoaPods x

SwiftPackageManager x

Carthage x

Create

Что писать в сообщении коммита

В сообщениях к коммитам в Git, принято следовать определённым правилам, чтобы история изменений была понятна всем пользователям.

Сообщение коммита должно быть конкретным и информативным

КРУТО: Добавлен алгоритм автоматического сохранения формы.

КРУТО: Исправлены ошибки в код-стайле ViewController.

НЕ ОЧЕНЬ: Исправлены ошибки.

НЕ ОЧЕНЬ: Fix.

НЕ ОЧЕНЬ: Правка замечания.

Используйте обезличенные глаголы в начале сообщения

КРУТО: Исправлена опечатка в названии переменной.

КРУТО: Добавлена история измерений на главный экран.

НЕ ОЧЕНЬ: Исправил(а) опечатку в названии переменной.

Не пишите слишком длинные сообщения

Стремитесь к тому, чтобы первая строка сообщения была не более 50 символов.

Начинайте сообщение с заглавной буквы

Это не только вопрос стиля, это также улучшает читаемость списка коммитов.

КРУТО: Добавлен новый раздел в README.

НЕ ОЧЕНЬ: добавлен ещё один раздел в README.



Использовать русский язык или английский для описания изменений — обычно это правило отдельной команды разработки. Используйте язык, принятый в вашей команде.

Markdown: язык разметки

Markdown — это язык разметки, с помощью которого форматируют текст. Используя простые символы, такие как `#` для заголовков или `*` для выделения текста курсивом, вы можете структурировать текст, делая его наглядным, читаемым и с удобной навигацией.

В репозиториях в GitHub в разметке Markdown часто описывают файлы README.md и дополнительную документацию. Описание пул-реквестов и комментарии в них тоже поддерживают Markdown.

Ниже приведены примеры синтаксиса.

Заголовки

- Для создания заголовков используются символы `#`. Количество символов `#` соответствует уровню заголовка (от `#` до `#####`, от h1 до h6 в HTML).

```
# Заголовок 1
## Заголовок 2
```


Выделение текста

- **Курсив** — обрамите текст одиночными звездочками или нижними подчеркиваниями: `курсив*` или `_курсив_`.
- **Жирный** — используйте двойные звездочки или нижние подчеркивания: `*жирный*` или `__жирный__`.
- **Жирный курсив** — комбинируйте оба метода: `**жирный курсив**` или `__жирный курсив__`.

Списки

- Ненумерованные списки создаются с помощью `*`, `-` и `+`

- Пункт 1
- Пункт 2
- Пункт 3

- Нумерованные списки начинаются с числа и точки.

1. Первый пункт
2. Второй пункт
3. Третий пункт

Ссылки

- Чтобы создать ссылку, используйте квадратные скобки для текста ссылки и круглые скобки для URL: `[Practicum](https://practicum.yandex.ru)`.

Изображения

- Почти так же, как и ссылки, но добавьте впереди восклицательный знак: `![альтернативный текст](url_изображения)`.

Код

Используйте обратные кавычки (```) для `inline кода` и тройные обратные кавычки (`````) для блоков кода:

- Inline: ``код``.

- Блок кода.

```
```swift  
print("Hello world!")
```
```

Цитаты

- Для цитат используйте `>`:

```
> Это цитата.
```

Горизонтальная линия

- Чтобы поставить разделитель, используйте три и более звездочки (`***`), дефисы (`--`) или нижние подчеркивания (`_`).

Яндекс Практикум