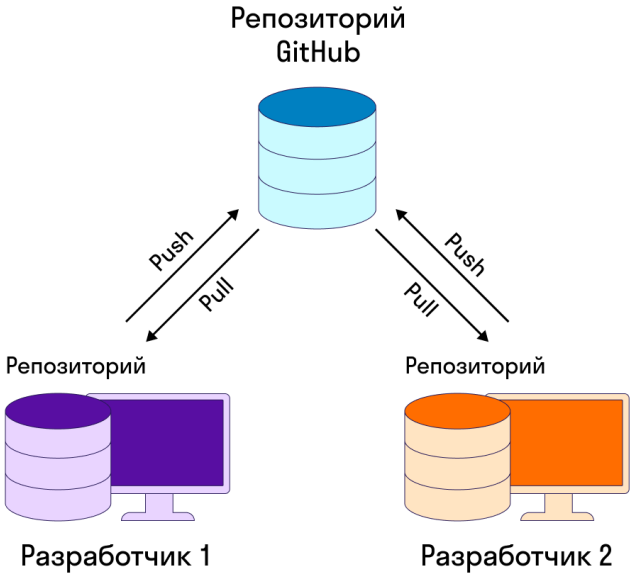
skypro

Продвинутый Git. Шпаргалка

GitHub

GitHub — это бесплатный (для одиночного использования) сервис, в котором хранят свои проекты большинство компаний и разработчиков. Основан на системе контроля версий Git и позволяет разработчикам хранить свои проекты в облаке, следить за изменениями, делать пул-реквесты, создавать ветви и сливать их с основной веткой кода.



Push — отправка новых коммитов на удаленный репозиторий, например в GitHub. Эта операция выполняется командой `git push`.

Pull — обновление локальной ветки за счет коммитов на удаленном репозитории. Эта операция выполняется командой `git pull`.

Для работы с GitHub вам понадобится:

1. Создать аккаунт на github.com.
2. Привязать GitHub-аккаунт по персональному токену или SSH-ключу.

> **HTTPS-токен** (или personal access token) для GitHub — это специальная строка символов, которая используется вместо пароля при работе с репозиториями GitHub через протокол HTTPS. Этот токен добавляет дополнительный уровень безопасности и предоставляет доступ к вашим репозиториям и ресурсам на GitHub без необходимости использования вашего фактического пароля.

> **SSH-ключ** — это идентификатор, с помощью которого удаленные компьютеры могут обмениваться данными.

3. Создать репозиторий на GitHub.
4. Связать локальный и удаленный репозитории.

Связываем репозиторий в GitHub с локальным Git-репозиторием с помощью командной строки

Чтобы посмотреть URL-адрес каждого подключения, добавьте к команде `git remote` флаг `-v`:

```
git remote -v
```

Если вы хотите удалить существующую связь локального репозитория с удаленным, воспользуйтесь командой:

```
git remote remove имя_remote
```

Чтобы установить связь локального репозитория с удаленным по ссылке (HTTPS или SSH), воспользуйтесь командой:

```
git remote add имя_remote ссылка_на_удаленный_репозиторий
```

> Обратите внимание, что имя `remote` можно задать какое угодно. Локальный репозиторий может быть связан одновременно с несколькими удаленными по разным `remote`. Стандартное имя для `remote` — `origin`.

Команда `git push` используется для выгрузки содержимого локального репозитория в удаленный репозиторий. Она позволяет передать коммиты из локального репозитория в удаленный.

Пример использования команды:

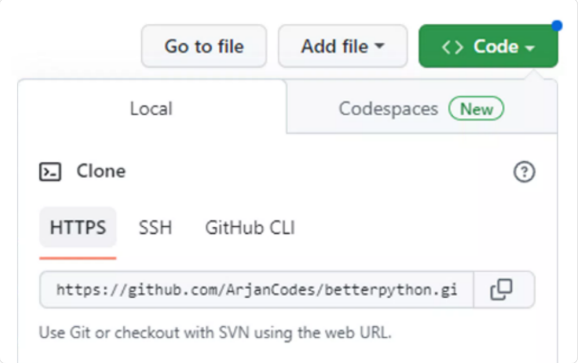
```
git push remote branch
```

Клонирование

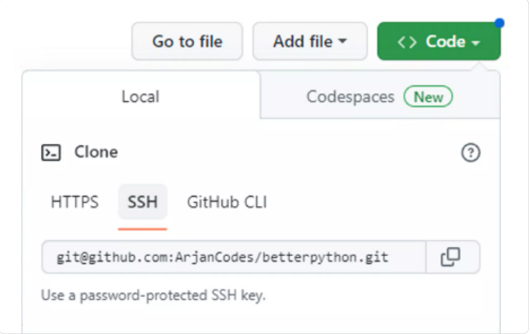
```
git clone git@github.com:ТУТ_ВАШЕ_ГИТХАБ_ИМЯ/project.git
```

Обратите внимание

Если вы делаете привязку по персональному токену, то для клонирования репозитория используйте ссылку **HTTPS**:



Если вы делаете привязку по SSH-ключу, то для клонирования репозитория используйте ссылку **SSH**:



README.md

> **README.md** — это текстовый файл в формате Markdown, который содержит краткое описание вашего проекта.

Что должно быть в README.md:

1. Заголовок и краткое описание.
2. Установка и использование.
3. Примеры использования.
4. Документация и ссылки.
5. Лицензия.

Язык разметки Markdown

> **Markdown** — это язык разметки, который позволяет легко форматировать текст в стиле подобном HTML, широко используется для написания README.md-файлов на GitHub и других платформах разработки.

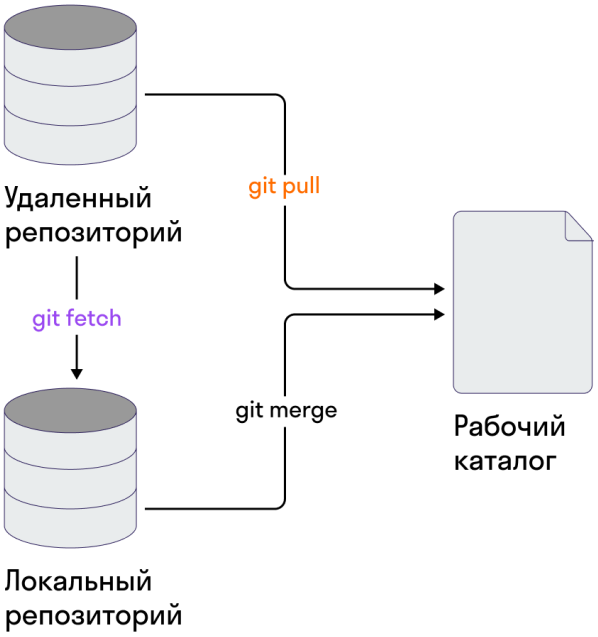
Основные элементы Markdown для README.md

1. **Заголовки.** Обозначаются символом `#`. Например, `# Заголовок` создаст заголовок верхнего уровня.
2. **Списки.** Маркированные списки создаются с помощью `*`, `-` или `+`, а нумерованные списки — с помощью числовых значений.
3. **Ссылки.** Ссылки создаются в следующем формате: `[текст ссылки](URL)`. Например, `[GitHub](<https://github.com>)` создаст ссылку на GitHub.
4. **Изображения.** Изображения вставляются так же, как и ссылки, но с добавлением символа `!` в начале: `![alt текст](URL_изображения)`.
5. **Код.** Код внутри текста отмечается обратными кавычками (`` ` ``). Для блочного кода используется три обратные кавычки (```` ``` ````).
6. **Выделение текста.** Текст можно выделять жирным (`**текст**`) или курсивом (`*текст*`).

Основные команды при работе с GitHub

Команда `git fetch` — собирает коммиты из целевой ветки удаленного репозитория, которых нет в текущей ветке, и сохраняет их в локальном репозитории, но не сливает их в текущую ветку.

Команда `git pull` — сливает любые внесенные коммиты в ветку, в которой вы сейчас работаете.



GitFlow

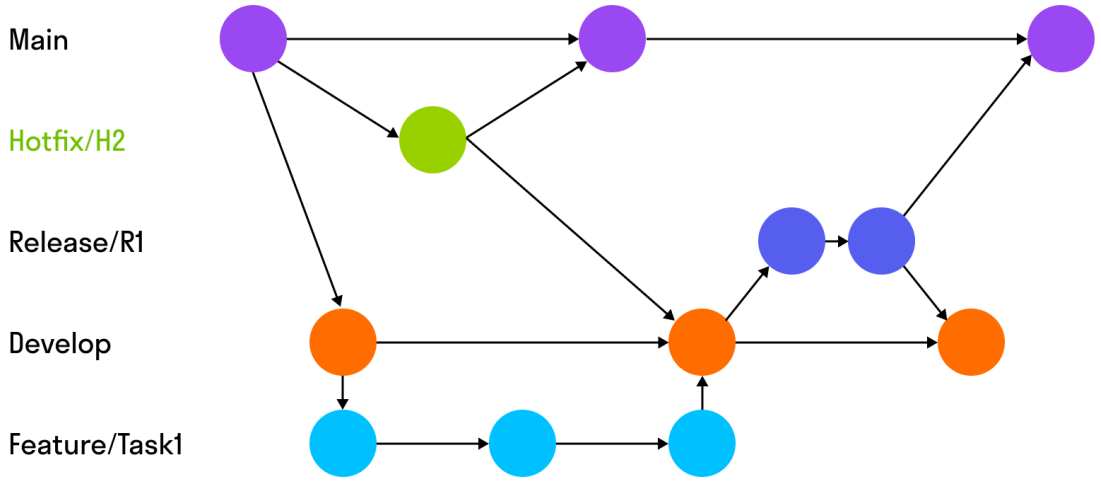
Ветка (branch) — последовательность коммитов, расположенных в хронологическом порядке.

GitFlow — методика работы с Git, в которой определяется, какие виды веток необходимы проекту и как выполнять слияние между ними.

Описание веток

- `Main` (`Master`) — основная ветка, которая выкатывается на продакшен, то есть это рабочая версия продукта, доступная пользователям.
- `Develop` — рабочая ветка, куда попадают все принятые изменения.
- `Feature/Task1` — функциональная ветка для внедрения фичи; берется из ветки `develop`.
- `Release/R1` — ветка для подготовки релиза.
- `Hotfix/H2` — ветка, которая используется для внесения срочных исправлений; берется от ветки `main`.

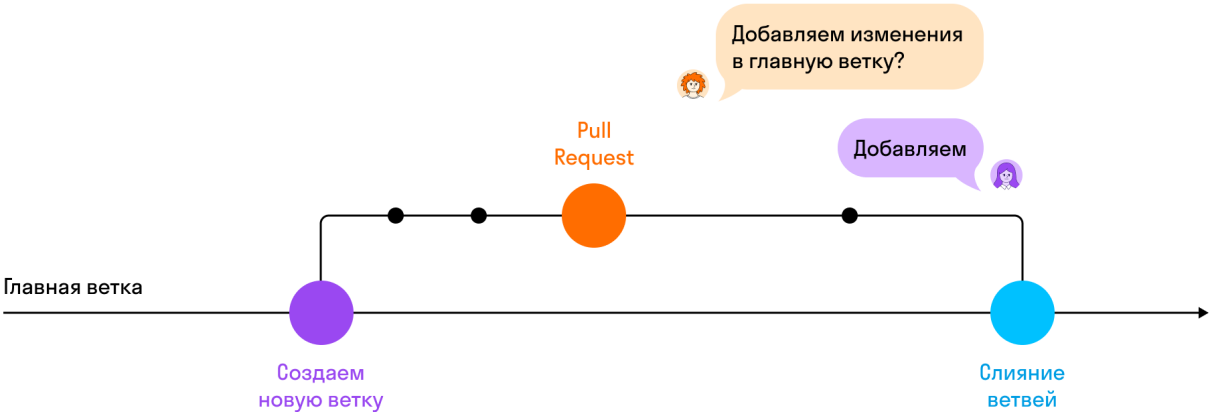
Описание процесса



1. Создается ветка `develop` от ветки `main`.
2. В ветке `develop` происходит разработка новых функций. Каждая функция создается в отдельной ветке `feature/task1` и после завершения работ сливается с веткой `develop`.
3. Когда функциональность готова к релизу, создается ветка `release/R1` от ветки `develop`. После завершения работ ветка `release/R1` сливается с веткой `main` и помечается тегом с номером версии. Также изменения, полученные в `release/R1`, сливаются в ветку `develop`.
4. Если в процессе эксплуатации выявляются ошибки, создается ветка `hotfix/H2` от ветки `main`. В нее вносятся только исправления ошибок, и после завершения работ ветка `hotfix/H2` сливается с веткой `main` и `develop`.

Pull Request

Pull Request (PR) — это механизм, используемый в Git, который позволяет предложить изменения в проект, внесенные в ветке вашего репозитория, чтобы они могли быть включены в основную (или любую другую) ветку проекта.



Порядок работы с pull request

1. Создайте новую ветку для ваших изменений. Название ветки обычно отражает характер изменений.
2. Внесите необходимые изменения в вашей ветке. Это может быть добавление нового функционала, исправление ошибок или любые другие изменения.
3. После завершения изменений отправьте pull request из вашей ветки в основную ветку проекта. В pull request вы можете описать характер внесенных изменений и почему они важны.
4. Другие участники проекта могут просматривать ваш pull request, комментировать код и предлагать изменения. Вы можете обсуждать изменения и вносить коррективы.
5. После того как изменения прошли обсуждение и проверку, они могут быть включены в основную ветку проекта.
6. После внесения изменений в основную ветку вы можете удалить вашу временную ветку.

Команда git branch

Команда `git branch` позволяет создавать, просматривать, переименовывать и удалять ветки.

- Показать список существующих веток в вашем репозитории:

```
git branch
```

- Создать новую ветку с указанным именем, но не переключать на нее:

```
git branch branch_name
```

- Удалить указанную ветку.
  Это безопасная операция, Git не позволит удалить ветку с неслитыми изменениями.

```
git branch -d branch_name
```

- Принудительное удаление ветки, даже если в ней есть неслитые изменения:

```
git branch -D branch_name
```

- Переименование текущей ветки:

```
git branch -m new_branch_name
```

- Вывести список удаленных веток:

```
git branch -a
```

Команда git checkout

Команда `git checkout` в Git используется для переключения между ветками и восстановления рабочего каталога до состояния определенного коммита или ветки.

- Переключение между ветками:

```
git checkout branch_name
```

- Создание новой ветки и переключение на нее:

```
git checkout -b new_branch_name
```

Команда git push

Команда `git push` в Git используется для отправки изменений из локального репозитория в удаленный репозиторий. Это позволяет вам обновлять удаленный репозиторий после внесения изменений в вашей локальной ветке.

Эта команда отправляет локальную ветку (указанную `branch_name`) в удаленный репозиторий с именем `origin`:

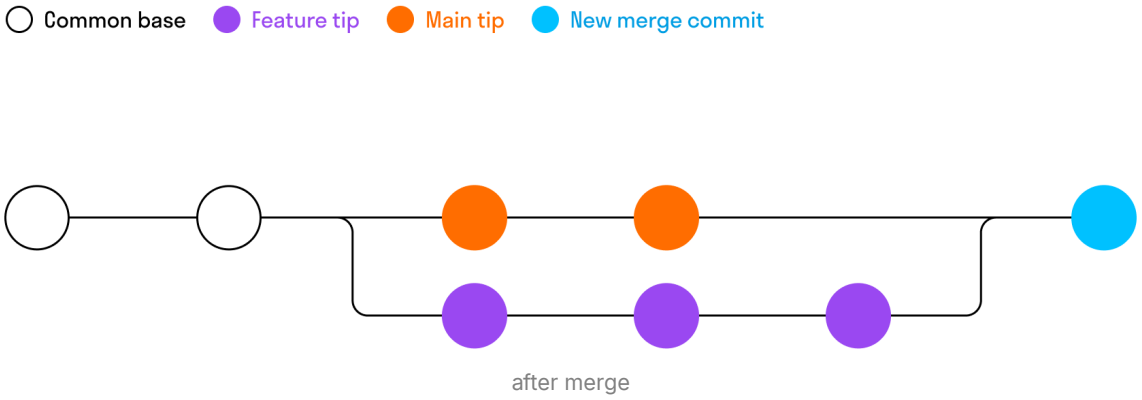
```
git push origin branch_name
```

Слияния: merge и rebase

Merge используется для объединения изменений из одной ветки в другую. Команда `git merge` позволяет соединить две ветки, например основную ветку `main` и новую функциональную ветку.

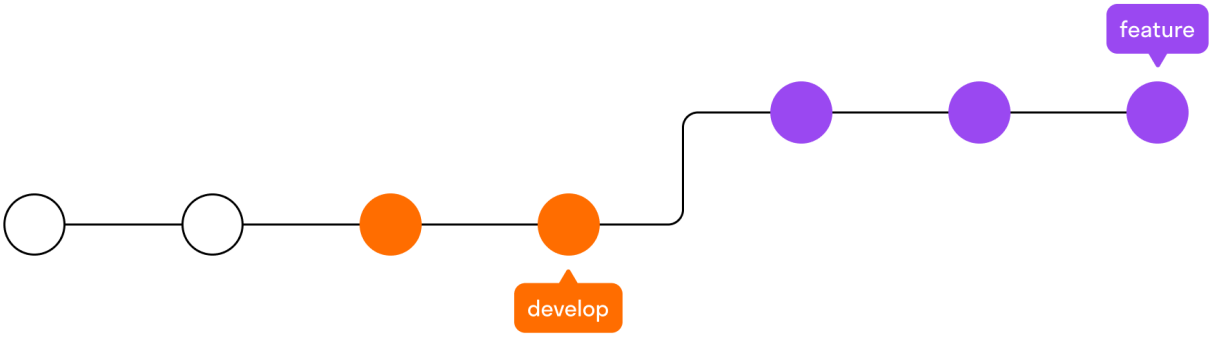
У нас есть ветка `main` и новая ветка `feature`, которую мы хотим объединить с `main`. Когда мы выполняем `git merge feature`, Git автоматически объединяет изменения из `feature` с текущей веткой, создавая новый коммит слияния.

```
git checkout main
git merge feature
```



after merge

Rebase также объединяет изменения, но он делает это путем пересоздания истории. Команда `git rebase` берет изменения из одной ветки и вставляет их в другую. В результате история становится линейной, без дополнительных коммитов слияния.




```
git checkout feature
git rebase main
```

Rebase создает новые коммиты на основе изменений из `main` и переносит ваши коммиты поверх них.