Рекомендации по ведению GIT

1. Создание и настройка репозитория

- Лид создаёт публичный репозиторий на https://github.com/, с названием `Team-<Номер команды>-<Тема проекта>` .

> Примеры хороших названий репозиториев:
>
> - Team-01-Forms
> - Team-02-Presentations
> - Team-03-SiteBuilder
> - Team-04-GraphicsEditor
> - Team-05-FitnessTrainer

- Добавляем в Collaborators, наставника дипломной практики @taraswww777 с правами (Write)
- Добавляем в Collaborators, остальных членов команды с правами (Write)
- Все добавленные участники должны проверить доступ к репозиторию
- В корне репозитория заводим `README.md` файл который заполняем по образцу:

```
# Команда <Номер коменды>. Проект - <Название проекта>.

Состав участников и их роли
```

Работа с ветками

Определены следующие основные ветки:

- `main` — ветка, соответствующая **рабочему продакшен-окружению** (production).
- `develop` — основная **рабочая ветка** для интеграции функций.
- `release/*` — ветки-кандидаты в релиз (release candidate).
- `feature/*` — ветки для разработки новых функций (фич).

Политика защиты веток в репозитории

Ниже описаны правила работы с основными ветками репозитория. Требования направлены на обеспечение стабильности кода и контроль качества изменений.

1. Ветка `main`

Правила внесения изменений:

- Только через Pull Request (PR).
- Запрещён `force push` (включая `--force-with-lease`).

Требования к PR:

- Обязательны **2 одобрения** (approvals) от ревьюверов.
- Вливание (merge) может выполнить **только лид команды**.
- **Обязательно** успешное прохождение всех этапов CI-пайплайна.

Цель: гарантия того, что в продакшен попадает только проверенный и согласованный код.

---

2. Ветка `develop`

Правила внесения изменений:

- Только через Pull Request (PR).
- Запрещён `force push`.

Требования к PR:

- Обязательно **1 одобрение** (approval) от ревьювера.
- **Обязательно** успешное прохождение всех этапов CI-пайплайна.

Цель: обеспечение базового контроля качества при интеграции новых функций.

---

3. Ветки `release/*`

Правила внесения изменений:

- Только через Pull Request (PR).
- Запрещён `force push`.

Требования к PR:

- Обязательны **2 одобрения** (approvals) от ревьюверов.
- Вливание (merge) может выполнить **только лид команды**.
- **Обязательно** успешное прохождение всех этапов CI-пайплайна.

Цель: максимальная защита релизных веток от ошибок и непроверенных изменений.

---

4. Ветки `feature/*`

Правила внесения изменений:

- **Нет ограничений** на способ внесения изменений.

- Разрешены:
  - прямые коммиты;
  - `force push` ;
  - локальные слияния без PR.

Цель: предоставление разработчикам свободы при реализации новых функций без избыточного контроля на этапе разработки.

Общие примечания

NaN. **Одобрения (approvals)**

```
- Одобрение должно быть дано ревьювером, не являющимся автором PR.

- В случае спорных правок требуется дополнительное обсуждение.
```

NaN. **CI-пайплайн**

```
- PR не может быть влитым, пока хотя бы один этап CI не пройден.

- При обновлении PR (новые коммиты) CI перезапускается автоматически.
```

NaN. **Force push**

```
- Запрет действует на уровне настроек репозитория (branch protection rules).

- Попытки `force push` будут отклонены системой.
```

NaN. **Роли**

```
- **Лид команды** имеет особые права на вливание в `main` и `release/*`.

- Остальные участники команды могут создавать PR и одобрять изменения в рамках своих полномочий.
```

Процесс: создание фичи

1. **Создание ветки**
   Из ветки `develop` создаётся ветка формата `feature/<issueNumber>-<shortDescription>`, например: `feature/123-add-login-form`

Рекомендация: используйте понятный короткий дескриптор после номера задачи.

2. **Разработка**
   В ветке `feature/...` реализуется функционал. Регулярно делайте коммиты с понятными сообщениями.
3. **Подготовка к PR**
   По завершении работы:
   - убедитесь, что код соответствует стандартам проекта;
   - проверьте отсутствие конфликтов с текущей версией `develop`.
4. **Создание Pull Request (PR)**
   Откройте PR из ветки `feature/123-add-login-form` в ветку `develop`.

   Важно: в названии PR должна быть ссылка на issue в формате `#<issueNumber>` (например, `#123`). Этого

достаточно — дополнительное описание в теле PR на данном этапе не требуется.

5. **Рецензирование**
   - коллеги оставляют комментарии и замечания;
   - внесите исправления в ту же ветку `feature/...`.
6. **Обновление ветки перед слиянием**
   Перед финальным слиянием выполните:

```
git fetch
git rebase origin/develop
```

Зачем: это синхронизирует вашу ветку с актуальными изменениями из `develop` и минимизирует конфликты.

7. **Слияние (merge)**
   После прохождения всех проверок (**quality gates**) и одобрения PR:
   - владелец PR выполняет слияние в `develop`;
   - ветка `feature/...` удаляется (опционально, по правилам команды).

Процесс: устранение конфликтов при `git rebase` (для веток `feature/*`)

1. Обнаружение конфликта

При выполнении `git rebase` Git может приостановить операцию из-за конфликта. Система выведет список конфликтующих файлов.

Проверьте статус:

```
git status
```

Файлы с конфликтами будут отмечены как *unmerged* или *both modified*.

2. Открытие конфликтующих файлов

Откройте файлы, указанные в статусе, в **любом удобном редакторе** (текстовом редакторе, IDE или специализированном инструменте для работы с Git).

3. Анализ и разрешение конфликта

В файле появятся специальные маркеры:

```
<<<<<<< HEAD
# Ваши изменения в текущей ветке
=======
# Изменения из целевой ветки
>>>>>>> <имя-целевой-ветки>
```

Ваши действия:

1. Проанализируйте оба варианта изменений.
2. Выберите, какие правки оставить:
   - только свои;
   - только чужие;
   - комбинированную версию (вручную объедините нужные части).
3. Удалите все маркеры конфликта
4. Сохраните файл после внесения правок.

Для разрешения конфликтов можно использовать:

- Свой текущий редактор кода

- IntelliJ IDEA Community Edition - подойдёт и для более сложных сценариев работы с GIT

4. Фиксация разрешения

После правки каждого конфликтующего файла добавьте его в индекс:

```
git add <путь-к-файлу>
```

5. Продолжение rebase

После разрешения всех конфликтов продолжите операцию:

```
git rebase --continue
```

Git продолжит применять коммиты. Если возникнут новые конфликты — повторите шаги 3–5.

6. Отмена rebase (при необходимости)

Если не удаётся разрешить конфликты или нужно прервать процесс:

```
git rebase --abort
```

Это вернёт ветку в состояние до начала `rebase`.

7. Проверка результата

Убедитесь, что rebase завершён успешно:

```
git status
```

Ожидаемый результат: чистый статус (нет unmerged файлов).

8. Отправка изменений в удалённый репозиторий

После успешного rebase отправьте изменения:

```
git push --force-with-lease origin <ваша-feature-ветка>
```

Важно: используйте `--force-with-lease` вместо простого `--force` для безопасной перезаписи ветки.

Рекомендации

1. **Регулярное обновление**
   Перед началом работы обновляйте свою ветку:

```
git fetch
git rebase origin/develop
```

2. **Коммуникация**
   Если работаете с общей `feature`-веткой, согласуйте rebase с коллегами.
3. **Резервное копирование**
   Перед сложным rebase создайте резервную ветку:

```
git branch backup-<имя-ветки>
```

4. **Тестирование после rebase**
   После завершения:

- запустите тесты;
- проверьте работоспособность функционала.

5. **Использование инструментов**

   Для сложных конфликтов рекомендуется применять:

   - встроенные средства IDE (трёхпанельные diff/merge-вьюеры).