Семинар #2: git, продолжение. Практика.

Как сдавать задачи

Для сдачи ДЗ вам нужно создать репозиторий на GitLab (если он ещё не создан) под названием `devtools-homework`. Структура репозитория должна иметь вид:

```
├── seminar2_more_git/
│   ├── 01.txt
│   ├── 02.sh
│   ├── 03.sh
│   ├── 04.txt
│   ├── 05a.sh
│   └── ...
└── ...
```

Для каждой задачи необходимо создать bash-скрипт, содержащий решение, и отправить его в ваш репозиторий. Название скрипта должно соответствовать номеру задачи (например `02.sh`, `05a.sh`). Если в задаче используется интерактивное перебазирования, то помимо скрипта вам нужно будет отправить файл `git-rebase-todo`. Название этого файла тоже должно начинаться с номера задачи, например `05b-git-rebase-todo.txt`. Если в задаче требуется что-то иное, это будет указано в конце самой задачи.

Подготовка. Генерация репозитория, который будет использоваться в задачах.

Для решения некоторых задач вам потребуется создать простой репозиторий, состоящий из четырёх коммитов, с помощью bash-скрипта. Для этого сделайте следующее:

1. Зайдите в репозиторий v.biryukov/devtools_course, пройдите в папку `seminar2_more_git/practice`, откройте файл `create_animals.sh` и скачайте его.
2. Скопируйте скрипт в папку, в которой вы будете его запускать. Убедитесь, что папка, куда вы копируете скрипт, не является частью другого репозитория, а также не содержит внутри себя папку `animals`.
3. Откройте терминал и зайти в папку, содержащую скрипт

   ```
   $ cd имяпапки
   ```

4. Добавьте скрипту права на исполнение

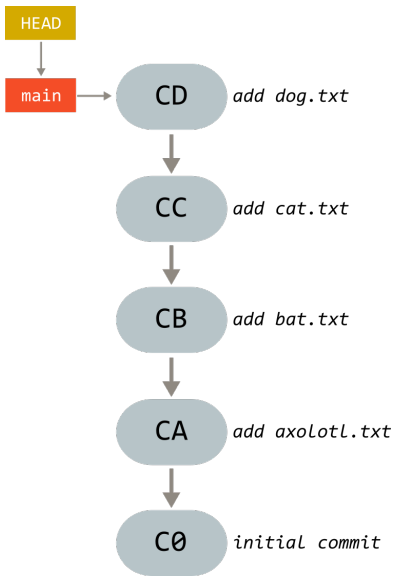
   ```
   $ chmod +x create_animals.sh
   ```

5. Запустите скрипт, он создаст папку `animals` с репозиторием внутри

   ```
   $ ./create_animals.sh
   ```

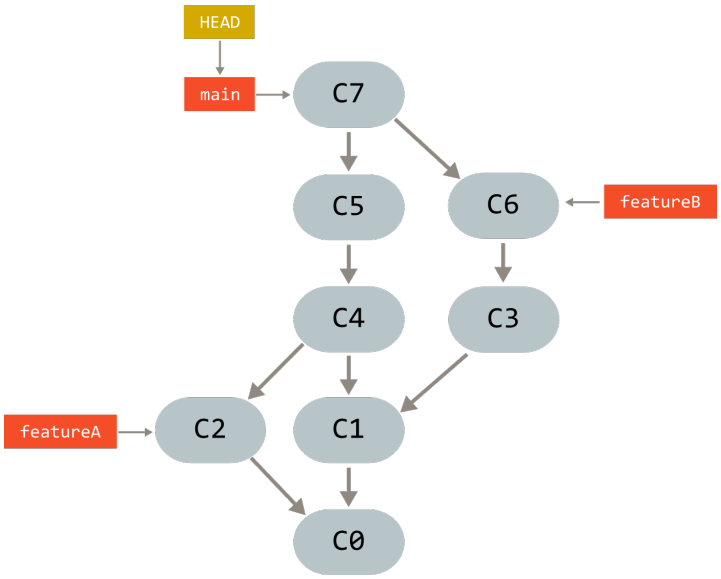
6. Перейдите в папку `animals` и убедитесь, что репозиторий создался корректно, напечатав дерево коммитов:

   ```
   $ cd animals
   $ git log --oneline --graph --all
   ```



Задача 1. Способы адресации коммитов

Пусть граф коммитов выглядит так, как это изображено на рисунке:



Коммит `C4` – это коммит слияния ветки **`featureA`** в ветку **`main`**, а `C7` – это коммит слияния **`featureB`** в **`main`**. После этого вызвали команду **`git log`** вот так:

```
$ git log --oneline --all --graph
*  cc17ea1 (HEAD -> main) C7
|\
| * 9e18dac (featureB) C6
| * eace0bf C3
* | 00d7b4b C5
* |   38d17da C4
|\ \
| |/
|/|
| * 4ed9ec3 (featureA) C2
* | 3599341 C1
|/
* 7afecc8 C0
```

Для каждого пункта ниже напишите команду или выражение, которое идентифицирует указанный коммит.

1. Хэш-адрес коммита `C0`
2. Адрес коммита `C0`, используя символы относительной адресации (∼ и ^) от текущего положения **`HEAD`**.
3. Адрес коммита `C0`, используя относительную адресацию от указателя ветки **`main`**.
4. Хэш-адрес коммита `C3`
5. Адрес коммита `C3`, используя относительную адресацию от указателя ветки **`featureB`**.
6. Адрес коммита `C3`, используя относительную адресацию от указателя **`HEAD`**.
7. Хэш-адрес коммита `C2`
8. Адрес коммита `C2`, используя относительную адресацию от указателя ветки **`featureA`**.
9. Адрес коммита `C2`, используя относительную адресацию от указателя **`HEAD`**.

Для того чтобы сдать эту задачу, создайте файл **`01.txt`** со всеми ответами и отправьте его в ваш репозиторий для сдачи заданий.

Задача 2. Отмена индексации

Предположим, что вы создали новый файл и добавили его в область индексирования, используя команды:

```
$ touch emu.txt
$ git add emu.txt
```

Однако, затем передумали и хотите убрать этот файл из области индексирования. Выполните команду, которая бы убирала этот файл из области индексирования, но оставляла бы файл в рабочей папке. Добавлять новые коммиты в этой задаче нельзя.

Задача 3. Отмена коммита

Предположим, что вы создали новый файл и добавили его в область индексирования и в локальный репозиторий:

```
$ touch emu.txt
$ git add emu.txt
$ git commit -m "CE: add emu.txt"
```

Однако, затем передумали и хотите отменить коммит. При этом вы хотите, чтобы файл `emu.txt` продолжал находиться и в рабочей папке и в области индексирования. Выполните команду, которая отменит коммит, оставив рабочую папку и область индексирования неизменными.

Задача 4. Отмена `reset`

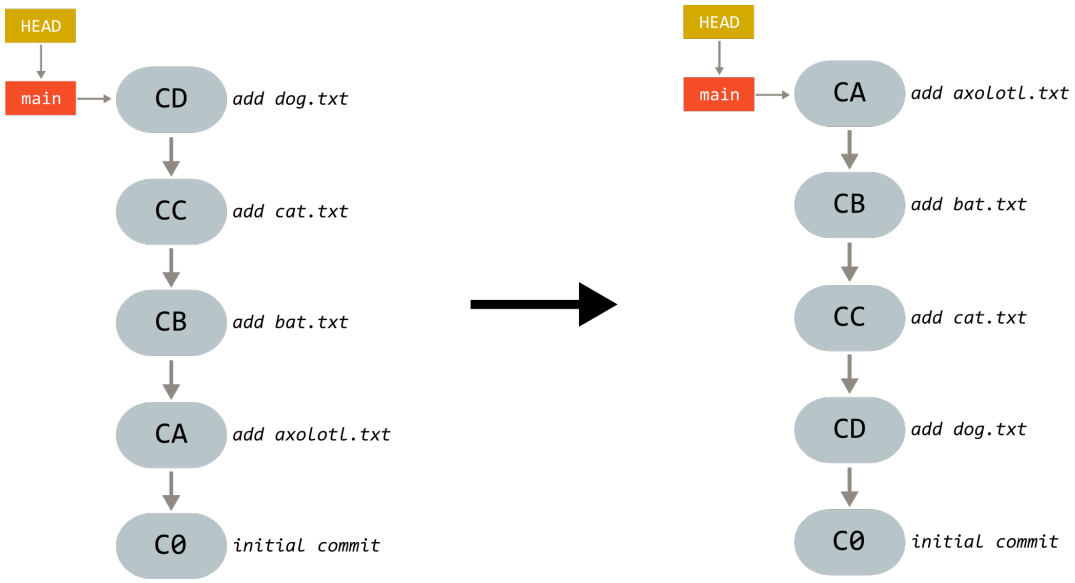
Предположим, что вы решили удалить все коммиты, кроме первого и, находясь на коммите `CD` (*add dog*), выполнили:

```
$ git reset --hard HEAD~4
```

Но потом передумали, и решили вернуть всё обратно. Что для этого нужно сделать?
Для того, чтобы сдать эту задачу вам нужно создать текстовый файл `04.txt`, в котором нужно будет описать словами алгоритм действий для восстановления после `reset`.

Задача 5. Обратить коммиты

В этом задании вам нужно обратить коммиты как это показано на изображении. При этом создавать файлы вручную нельзя, можно только использовать команды `git`.



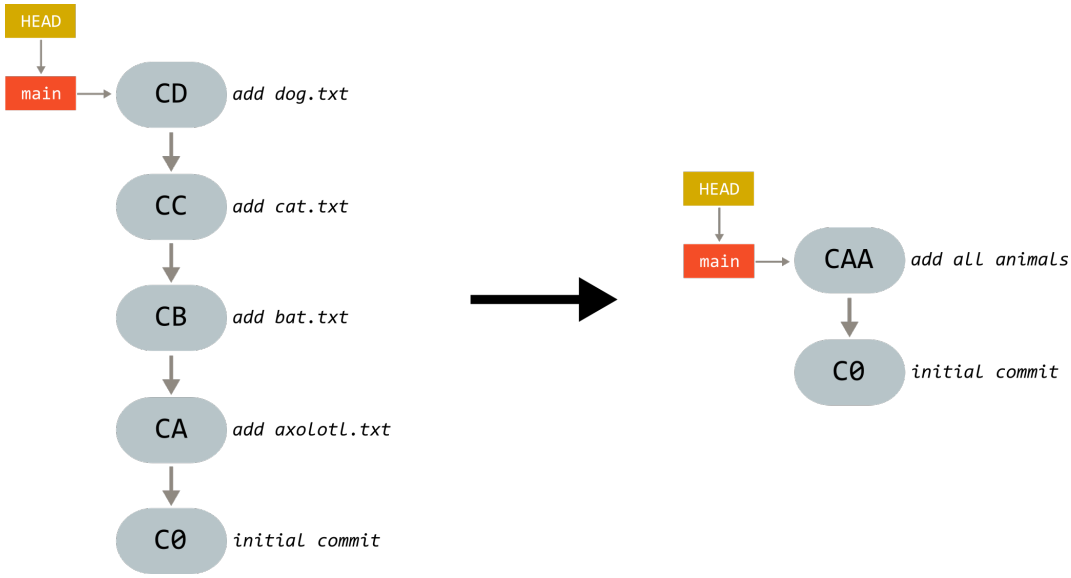
Решите эту задачу двумя способами:

(a) Создав новую ветку на начальном коммите и добавляя коммиты с помощью `git cherry-pick`. После этого нужно будет удалить старую ветку и переименовать новую.

(b) Используя интерактивное перебазирование.

Задача 6. Объединить коммиты

Пересоздайте репозиторий с помощью скрипта `create_animals.sh`. В этом задании вам нужно объединить все коммиты в один как это показано на изображении. При этом создавать файлы вручную нельзя, можно только использовать команды `git`.



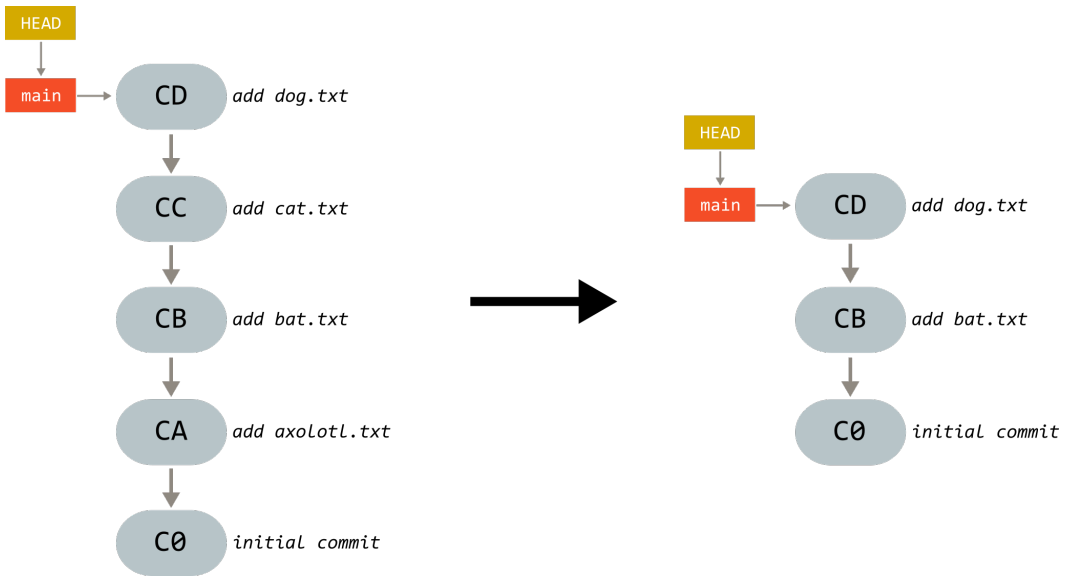
Решите эту задачу двумя способами:

(a) Используя `reset`.

(b) Используя интерактивное перебазирование.

Задача 7. Удалить коммиты

Пересоздайте репозиторий с помощью скрипта `create_animals.sh`. В этом задании вам нужно удалить два коммита, как это показано на изображении. При этом создавать или удалять файлы вручную нельзя, можно только использовать команды `git`.



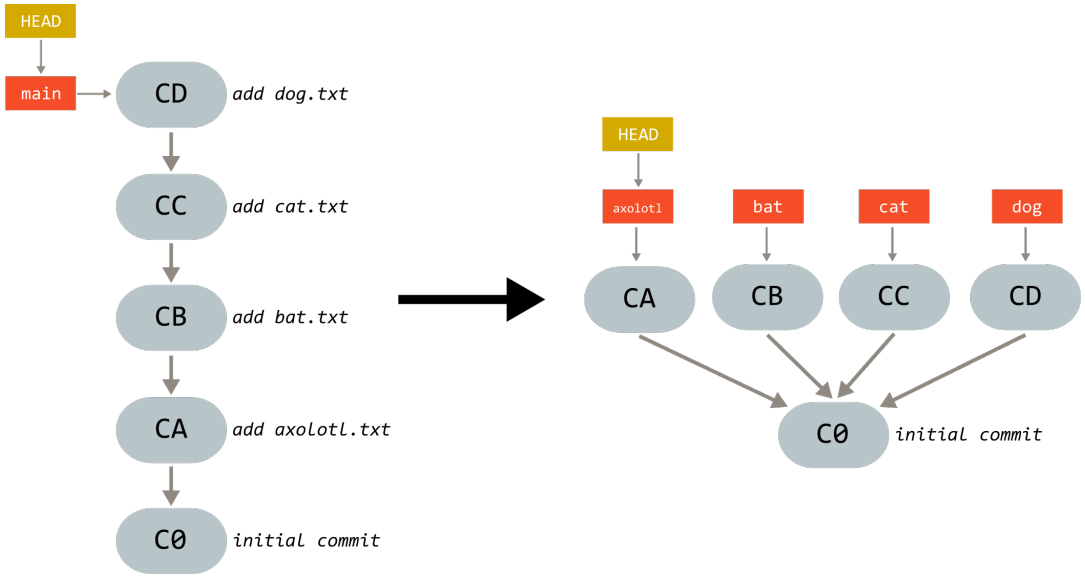
Решите эту задачу двумя способами:

(a) Используя `cherry-pick`.

(b) Используя интерактивное перебазирование.

Задача 8. Перераспределить коммиты

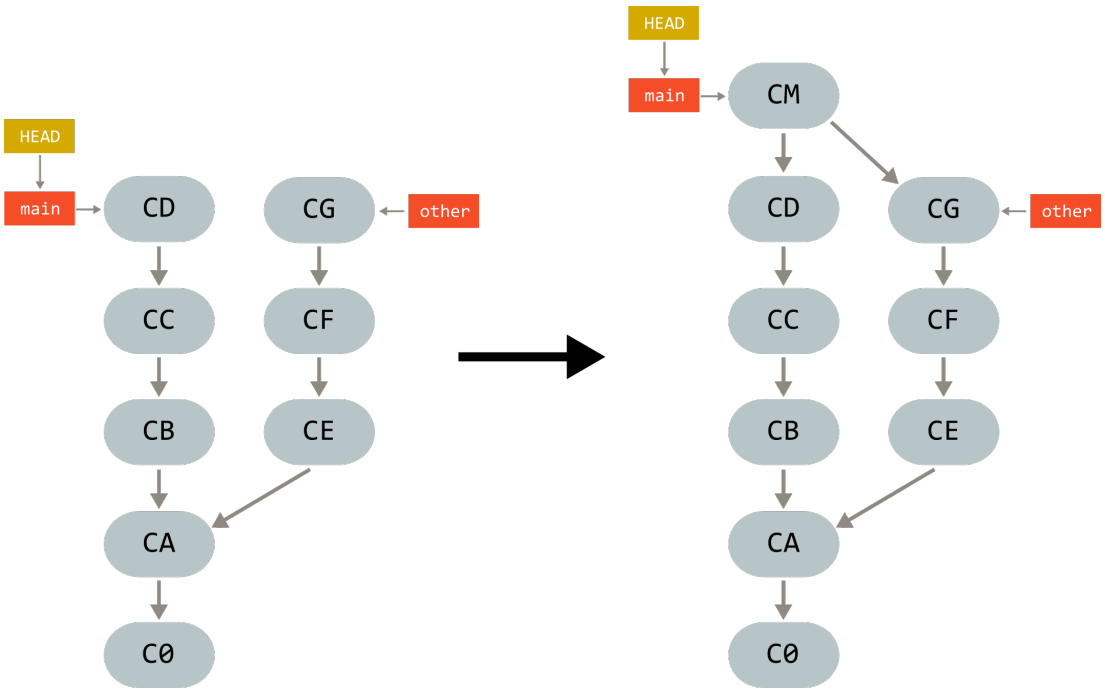
Пересоздайте репозиторий с помощью скрипта `create_animals.sh`. В этом задании вам нужно перераспределить коммиты в новые ветки, как это показано на изображении. При этом создавать или удалять файлы вручную нельзя, можно только использовать команды `git`.



Задача 9. Отмена слияния

Создайте новый репозиторий с помощью скрипта `create_branch_animals.sh` (его можно скачать оттуда же, где находился предыдущий скрипт). В этом репозитории будет 2 ветки: `main` и `other`. Предположим, что вы сделали слияние с помощью команд:

```
$ git switch main
$ git merge other -m "CM: merging"
```

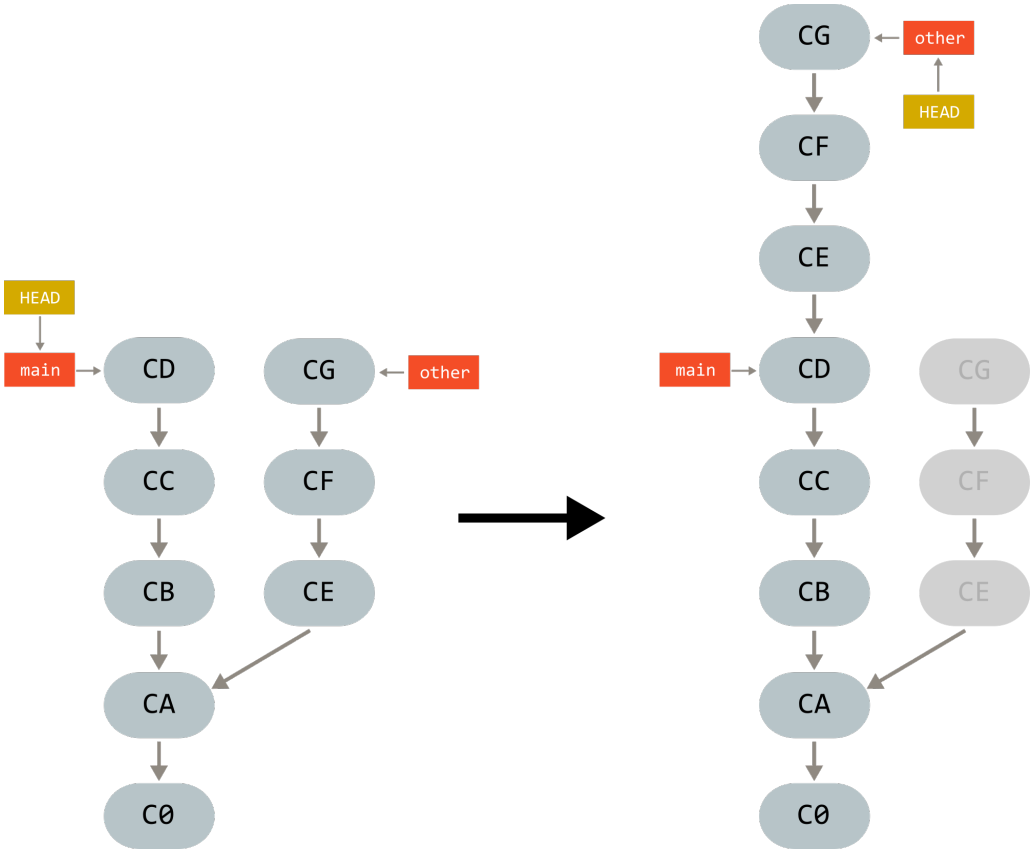


Однако потом передумали и хотите отменить слияние. Напишите команды `git`, которые бы отменяли слияние и возвращали бы всё как прежде.

Задача 10. Отмена перебазирования

Создайте новый репозиторий с помощью скрипта `create_branch_animals.sh`. В этом репозитории есть 2 ветки: `main` и `other`. Предположим, что вы сделали перебазирование ветки `other` на ветку `main` с помощью команд:

```
$ git switch other
$ git rebase main
```



Однако потом передумали и хотите отменить перебазирование. Напишите команды `git`, которые бы отменяли перебазирование и возвращали бы всё как прежде.

Задача 11. Отмена `push`

Для этой задачи вам нужно:

1. Создать новый локальный репозиторий на вашем компьютере с помощью скрипта `create_animals.sh`.
2. Создать новый пустой репозиторий на GitLab.
3. Скопировать ссылку на GitLab репозиторий (протокол SSH).
4. Добавить новый remote на вашем локальном репозитории.

   ```
   $ git remote add имя ссылка
   ```

   Имя можно выбрать любое, но часто выбирают `origin`.
5. Проверьте, что новый remote добавился:

   ```
   $ git remote -v
   ```

6. Отправьте ветки `main` и `other` вашего локального репозитория на удалённый репозиторий GitLab.

   ```
   $ git push -u origin main other
   ```

Предположим, что вы создали новый файл и добавили его в область индексирования, в локальный репозиторий, а затем и в удалённый репозиторий.

```
$ touch emu.txt
$ git add emu.txt
$ git commit -m "CE: add emu.txt"
$ git push
```

Однако затем передумали и хотите отменить коммит. Вы хотите, чтобы изменения пропали везде – как на вашем компьютере, так и в последнем состоянии удалённого репозитория. Решите эту задачу двумя способами:

- Используя `git push`.
- Используя `git revert`.

Задача 12. Игнорирование

Вы работаете над проектом на языке C и используете Git для контроля версий. В процессе разработки, компиляции и работы в разных средах создаются файлы, которые не должны попадать в репозиторий. А именно, в репозиторий не должно попадать следующее:

- **Логи проекта**
  Папки с названием `logs`, в какой бы директории проекта они не находились.
- **Служебные файлы операционных систем**
  Файлы с названиями `Thumbs.db` и `.DS_Store`, в какой бы директории проекта они не находились. Это скрытые служебные файлы, которые генерируются операционными системами Windows и macOS соответственно.
- **Файлы IDE**
  Папки с названием `.vscode` и `.idea`, в какой бы директории проекта они не находились. Эти папки генерируются при использовании IDE Visual Studio и IDE от JetBrains соответственно.
- **Файлы, создаваемые при компиляции**
  Любые файлы с расширениями `.exe`, `.a`, `.lib .dll`, `.o`, `.so`, в какой бы директории они не находились.
- **Папка сборки**
  Папка `build` в корневой директории проекта. Но в поддиректориях папка с таким названием игнорироваться не должна.
- **Конфигурационные файлы**
  Файлы с расширением `.cfg`, в какой бы директории проекта они не находились. За исключением файла `config/settings.cfg`, который не должен игнорироваться.

Напишите файл `.gitignore` для такого проекта. Протестируйте этот файл на одном из созданных ранее репозиториев. Для сдачи этого задания поместите файл `.gitignore` в ваш репозиторий для сдачи заданий.

Задача 13. Форк

Зайдите на GitLab и сделайте форк репозитория mipt-hsse.gitlab.yandexcloud.net/v.biryukov/branch_animals. Клонируйте форк на свой компьютер. Создайте новую ветку на коммите `HEAD`. Добавьте в этой ветке новый коммит, содержащий файл с новым животным (на ваш выбор). Отправьте изменения на ваш форк. Сделайте merge request, чтобы добавить изменения в изначальный репозиторий (`v.biryukov/branch_animals`).
Для сдачи этого задания убедитесь, что ваш форк является публичным.

Задача 14. Тэги

Создайте 2 новых тэга в репозитории `branch_animals`: один на коммите `CA` – `v0.0.1`, второй на коммите `CD` – `v0.1.0`. Проверьте, что тэги созданы:

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

Для сдачи этого задания отправьте теги на удалённый сервер на GitLab.