**Вычислитель отличий**

**Источник - проект с сайта hexlet.io.**

Необходимо создать консольное приложение для поиска отличий в конфигурационных файлах.
Проект необходимо разместить на GitHub.
Код должен быть проверен линтером (выбрать самостоятельно)
Код должен быть протестирован.

### Возможности утилиты:

- Поддержка разных форматов
- Генерация отчета в виде plain text, pretty и json

### Пример использования:

```
$ gendiff --format plain first-config.ini second-config.ini
Setting "common.setting2" deleted.
Setting "common.setting4" added with value "blah blah".
Setting "group1.baz" changed from "bas" to "bars".
Section "group2" deleted.
```

После выполнения каждого шага **необходимо** связаться со мной, предварительно загрузив код в репозиторий.

**Шаг 1.**

### Первым шагом в этом проекте нужно вывести справочную информацию по нашей утилите:

```
gendiff -h

  Usage: gendiff [options] <firstConfig> <secondConfig>

  Compares two configuration files and shows a difference.

  Options:

    -h, --help            output usage information
    -V, --version         output the version number
    -f, --format [type]   Output format
```

Можно использовать стороннюю библиотеку для построения интерфейсов командной строки.
На этом шаге тесты не требуются.

**Шаг 2.**

## Сравнение плоских файлов (JSON)

Ниже пример того, что должно получиться по результату данного шага:

```
$ gendiff before.json after.json

{
    host: hexlet.io
  + timeout: 20
  - timeout: 50
  - proxy: 123.234.53.22
  + verbose: true
  - follow: false
}
```

Отсутствие плюса или минуса говорит о том, что ключ присутствует в обоих файлах и его значения совпадают. Во всех остальных ситуациях ключ был либо удален, либо добавлен, либо изменен. В примере выше ключ `timeout` был изменен, `proxy` удален, а `verbose` добавлен.
Порядок вывода не важен, главное чтобы в случае изменения значения одного ключа, обе строчки находились рядом.
Сам диф между исходными структурами должен быть представлен в виде AST. AST в данном случае описывает то, что произошло с каждым ключем в дифе, например, был ли он добавлен, изменен или удален. Удобство AST в том, что очень сильно упрощается рендеринг, так как он работает уже с вычисленной разницей

```
{
  "host": "hexlet.io",
  "timeout": 50,
  "proxy": "123.234.53.22",
  "follow": false
}
```

after.json:

```
{
  "timeout": 20,
  "verbose": true,
  "host": "hexlet.io"
}
```

Требования:

- Сравниваются данные, а не строки файлов
- Две строчки дифа, отвечающие за изменение поля (пример с timeout), должны находиться рядом. Порядок не важен.
- С этого момента подразумевается, что разработка ведется через тесты.
- Должна быть возможность использовать утилиту как библиотеку и как консольную программу

**Шаг 3.**

Сравнение плоских файлов (yaml)

Добавить поддержку файлов yaml

Требование:

- Вынесите код отвечающий за парсинг в собственный модуль `parsers`.

Для парсинга yaml можно использовать стороннюю библиотеку.

**Шаг 4. (по желанию)**

Сравнение плоских файлов (INI)

Добавить поддержку файлов ini

**Шаг 5.**

Сравнение сложных файлов.

Форматы `yaml` и `json` имеют рекурсивную структуру. Другими словами, значение по определенному ключу само может быть `json` или `yaml`. Подобную вложенность можно получить и в `ini` файлах, для этого используется особый, описанный в вики, синтаксис.
Пример дифа для вложенных структур:

```
{
    common: {
      + follow: false
        setting1: Value 1
```

```
      - setting2: 200
      - setting3: true
      + setting3: {
            key: value
        }
        setting6: {
            key: value
          + ops: vops
        }
      + setting4: blah blah
      + setting5: {
            key5: value5
        }
    }
    group1: {
      + baz: bars
      - baz: bas
        foo: bar
      - nest: {
            key: value
        }
      + nest: str
    }
  - group2: {
        abc: 12345
    }
  + group3: {
        fee: 100500
    }
}
```

before.json:

```
{
  "common": {
    "setting1": "Value 1",
    "setting2": "200",
    "setting3": true,
    "setting6": {
      "key": "value"
    }
  },
  "group1": {
    "baz": "bas",
    "foo": "bar",
    "nest": {
```

```
      "key": "value"
    }
  },
  "group2": {
    "abc": "12345"
  }
}
```

after.json

```
{
  "common": {
    "follow": false,
    "setting1": "Value 1",
    "setting3": {
      "key": "value"
    },
    "setting4": "blah blah",
    "setting5": {
      "key5": "value5"
    },
    "setting6": {
      "key": "value",
      "ops": "vops"
    }
  },

  "group1": {
    "foo": "bar",
    "baz": "bars",
    "nest": "str"
  },

  "group3": {
    "fee": "100500"
  }
}
```

На этом шаге необходимо разделить процесс построения дифа и рендеринг. Сам диф между исходными структурами должен быть представлен в виде AST. AST в данном случае описывает то, что произошло с каждым ключем в дифе, например, был ли он добавлен, изменен или удален. Удобство AST в том, что очень сильно упрощается рендеринг, так как он работает уже с вычисленной

разницей. Добавлять новые рендеринги становится проще, и они не дублируют логику построения дифа.

**Шаг 6**
**Плоский формат вывода**

Часто утилиты предоставляют вывод результата своих действий в разных форматах. Это бывает нужно при интеграции с другими системами или просто для удобства восприятия. Диф между файлами тоже можно отображать разными способами:

```
# Данный вывод просто пример, он не является отражением данных из других шагов
$ gendiff --format plain before.json after.json

Property 'timeout' was updated. From 50 to 20
Property 'proxy' was removed
Property 'common.setting4' was removed
Property 'common.setting5' was removed
Property 'common.setting2' was added with value: 200
Property 'common.setting6.ops' was added with value: 'vops'
Property 'common.sites' was added with value: 'hexlet.io'
Property 'group1.baz' was updated. From 'bars' to 'bas'
Property 'group3' was removed
Property 'verbose' was added with value: true
Property 'group2' was added with value: [complex value]
```

- Если новое значение свойства является составным, то пишется `[complex value]`
- если свойство вложенное, то отображается весь путь до корня, а не только с учетом родителя, например выше это: `common.setting6.ops`.

Ссылки

- **Паттерн: Фабрика**