

# Проверка заданий.

Буленок В.Г.

Студент входит на сайт, используя учётную запись на github. При первом входе для него создаётся репозиторий для сохранения выполненных заданий и результатов тестирования заданий. После входа студент видит свои оценки, замечания преподавателя, результаты тестирования заданий. Использование github репозитория для сохранения результатов даёт возможность преподавателю и студенту совместно работать с выполняемыми заданиями, преподаватель и студент могут вносить изменения в код, преподаватель может ставить задания перед студентом, через механизм issues, отслеживать историю развития проекта. Описания действий при тестировании мы сохраняем в репозитории в формате JSON. В этой статье мы рассмотрим программу проводящую тестирование и сохраняющую результат тестирования в репозитории. Способ организации работы со студентами мы заимствовали из курса информатики университета Гарварда <https://cs50.harvard.edu>. Исходный код писали сами, и считаем, что всякий преподаватель должен сам писать код своих инструментов, а не пользоваться готовыми системами, например, moodle. Для написания статьи был выбран стиль литературного программирования. Цель этого показать, как мы реализовали наши инструменты, чтобы другие преподаватели могли использовать наш опыт при создании своих, мы считаем, что преподаватель должен сам создавать инструменты для автоматизации работы со студентами а не брать готовые.

## 1 Анализ задания

Цель программы проверить работу студента и загрузить её вместе с результатом тестирования в репозиторий на github.

### 1.1 Описание проверки

Описание проверки сохраняем в репозитории <https://github.com/{orgTest}/psets>. Например, для задний на 2021 года создаём ветку 2021, для каждого задания создаём в ней файл. Ниже приведён пример файла `hello.js`. Структура файла:

```
{ "files":["hello.js"], "output":"Hello, World! "run":  
"hello.js"} 
```

Свойство `files` содержит список файлов, отправляемых на проверку. Свойство `output` содержит ожидаемый вывод из программы. Свойство `run` содержит имя основного файла.

## 1.2 Запуск программы

Для запуска программы переходим в терминале в каталог с заданием и выполняем команду, например, `check204 2021/hello` Аргумент – имя ветки и название задания (имя файла без расширения).

## 2 Структура

```
2 <check204.js 2>≡  
  #!/usr/bin/env node  
  <загрузка модулей 8a>  
  <константы 8b>  
  ;(async function main()  
  {  
      await checkArgs();  
      const psetName = process.argv[2];  
      const test = await getTask(psetName);  
      await filesExists(test);  
      await testOutput(test);  
      if(local)  
          process.exit(0);  
      await createDir(test);  
      await cloneRepo(orgTest, psetName, test);  
  })()  
  <проверка аргументов командной строки 3>  
  <закачивание описания 4a>  
  <тестирование задания 4b>  
  <отправка на проверку 6b>  
Uses orgTest 8b.
```

## 3 Реализация

### 3.1 Проверка аргументов

Если пользователь не указал ветку и название задания, то выдаётся сообщение об ошибке. Если пользователь указал неверный адрес, то выдаётся сообщение об ошибке.

3 *⟨проверка аргументов командной строки 3⟩*≡ (2)

```
async function checkArgs()
{
    if(process.argv.length < 3 || process.argv.length > 4)
    {
        console.log(chalk.red('Использование: \n
                                check204 branch/problem \n
                                или\n
                                check204 branch/problem local\n'));
        process.exit(1);
    }
    const url2 = 'https://raw.githubusercontent.com/${orgTest}/pset
    const url_exists = await urlExists(url2);
    if(!url_exists)
    {
        console.log(chalk.red("Неверный адрес задания"));
        process.exit(2);
    }
    if(process.argv.length == 4)
    {
        if(process.argv[3] == 'local')
            local = true;
        else
        {
            console.log('Использование: \n
                        check204 branch/problem local\n')
            process.exit(2);
        }
    }
};
```

Uses chalk 8a, orgTest 8b, and urlExists 8a.

## 3.2 Закачивание описания

4a  $\langle \text{закачивание описания 4a} \rangle \equiv$  (2)

```
async function getTask(psetName)
{
    try
    {
        const url2 = 'https://raw.githubusercontent.com/${orgTest}/pset
        const res = await fetch(url2,
        {
            headers:
            {
                "Accept": "application/json"
            }
        })
        const res1 = await res.json();
        return res1;
    } catch(err)
    {
        console.log(err);
    }
};
```

Uses `fetch` 8a and `orgTest` 8b.

## 3.3 Тестирование программы

4b  $\langle \text{тестирование задания 4b} \rangle \equiv$  (2)  
 $\langle \text{проверка наличия файлов 5} \rangle$   
 $\langle \text{проверка правильности вывода из программы 6a} \rangle$

### 3.3.1 Проверка наличия файлов

5  $\langle \text{проверка наличия файлов 5} \rangle \equiv$  (4b)

```
async function filesExists(test)
{
    const files = await fsPromises.readdir(process.cwd());
    for(file of test.files)
    {
        if(files.includes(file))
        {
            console.log(chalk.green('Файл ${file} присутствует'));
            resTest += 'Файл ${file} присутствует.\n';
        }
        else
        {
            console.log(chalk.red('Файл ${file} отсутствует'));
            process.exit(1)
        }
    }
}
```

Uses chalk 8a and fsPromises 8a.

### 3.3.2 Тест вывода из программы

6a  $\langle \text{проверка правильности вывода из программы 6a} \rangle \equiv$  (4b)

```
async function testOutput(test)
{
    const { stdout, stderr } = await exec(`node ${test.run}`);
    if(!stdout.localeCompare(test.output))
    {
        console.log(chalk.green('Вывод из программы верный'));
        resTest += '# Вывод из программы верный\n';
    }
    else
    {
        console.log(chalk.red('Ошибка'));
        resTest += "# Ошибка\n";
        console.log(chalk.red('Программа выдаёт'));
        resTest += '# Программа выдаёт\n';
        resTest += '<pre>\n';
        resTest += stdout.toString();
        resTest += '</pre>\n';
        console.log(stdout);
        console.log(stderr);
        console.log(chalk.red('Правильный вывод'));
        resTest += '# Правильный вывод\n';
        resTest += "<pre>";
        resTest += test.output;
        resTest += "</pre>"
        console.log(test.output);
        console.log(stderr);
    }
}
```

Uses chalk 8a and exec 8a.

### 3.4 Отправка на проверку

6b  $\langle \text{отправка на проверку 6b} \rangle \equiv$  (2)

$\langle \text{создание временного каталога 7a} \rangle$

$\langle \text{клонирование каталога 7b} \rangle$

### 3.4.1 Создание временного каталога

7a  $\langle \text{создание временного каталога } 7a \rangle \equiv$  (6b)

```
async function createDir(test)
{
    await fsPromises.rmdir('/tmp/test', {recursive: true});
    console.log("Folder /tmp/test deleted");
    await fsPromises.mkdir('/tmp/test', {recursive: true});
    console.log("Folder create dir");
}
```

Uses fsPromises 8a.

### 3.4.2 Клонирование каталога

7b  $\langle \text{клонирование каталога } 7b \rangle \equiv$  (6b)

```
async function cloneRepo(org, psetName, test)
{
    const readlineSync = require('readline-sync')
    var username = readlineSync.question('Github username:');
    const origin = `https://${username}@github.com/${org}/${username}`;
    const {stdout, stderr} = await exec(`git clone ${origin} -b main`);
    console.log(stdout);
    console.log(stderr);
    await exec(`git checkout -b ${psetName}`, {cwd: '/tmp/test'})
    for(let file of test.files)
    {
        await fsPromises.copyFile(path.join(process.cwd(), file),
        path.join('/tmp/test', file));
    }
    const fileName = `/tmp/test/${psetName.split('/')[1]}.md`;
    console.log(fileName);
    await fsPromises.writeFile(fileName, resTest);
    await exec(`git config user.name test && git config user.email test@example.com`);
    await exec(`git push -f ${origin} ${psetName}`, {cwd: '/tmp/test'})
}
```

Uses exec 8a, fsPromises 8a, and path 8a.

## 3.5 Загрузка модулей

8a  $\langle \text{загрузка модулей } 8a \rangle \equiv$  (2)

```
const path = require('path');
const chalk = require('chalk');
const util = require('util');
const urlExists = util.promisify(require('url-exists'));
const fetch = require('node-fetch');
const fsPromises = require('fs').promises;
const exec = util.promisify(require('child_process').exec);
```

Defines:

chalk, used in chunks 3, 5, and 6a.  
exec, used in chunks 6a and 7b.  
fetch, used in chunk 4a.  
fsPromises, used in chunks 5 and 7.  
path, used in chunk 7b.  
urlExists, used in chunk 3.  
util, never used.

## 3.6 Задание констант

orgTest - имя организации в которой репозиторий описания задания.

orgCheck - имя организации где сохраняем результат.

8b  $\langle \text{константы } 8b \rangle \equiv$  (2)

```
const orgTest = 'cs204';
const orgCheck = 'cs204check';
var resTest = '';
var local = false;
```

Defines:

orgCheck, never used.  
orgTest, used in chunks 2–4.

## 4 Установка в домашний каталог

Настроим на установку в каталог /npm.

8c  $\langle \text{установка } 8c \rangle \equiv$  8d>

```
mkdir ~/npm
npm config set prefix ~/npm
export PATH="$HOME/npm/bin:PATH"
```

Последняя строчка в .profile, чтобы система находила программы.  
Перезагрузить.

8d  $\langle \text{установка } 8c \rangle + \equiv$  <8c

```
npm -global install check204
```



## 4.1 Makefile

Сборку проекта осуществляет утилита make. В этой части мы создаем Makefile, управляющий сборкой. Утилита make начинает читать этот файл с конца, поэтому такой порядок действий выбран в файле. Результат работы утилиты make – это текст test.pdf, и файлы – программы check204, эти цели заданы в первой строке. Здесь используются утилиты noweave, notangle из пакета noweb, pdflatex генерирует pdf файл из tex файла.

9  $\langle$ Makefile 9 $\rangle \equiv$

```
res: doc/check204.pdf  bin/check204
```

```
bin/check204: src/main.nw
```

```
    notangle -Rcheck204.js src/main.nw > bin/check204
```

```
doc/check204.pdf: src/check204.tex  tmp/main.tex
```

```
    pdflatex -output-directory=doc src/check204.tex
```

```
    pdflatex -output-directory=doc src/check204.tex
```

```
# Утилита noweave генерирует файл main.tex из отдельных файлов  
# формата noweb.
```

```
tmp/main.tex: src/main.nw  src/Makefile.nw
```

```
    noweave -n -latex -index -autodefs c  src/main.nw \  
    src/Makefile.nw > tmp/main.tex
```

```
# Эту команду выполняем в случае, если были изменения в файле  
#Makefile.nw, notangle генерирует новый Makefile
```

```
new: src/Makefile.nw
```

```
    notangle -t8 -RMakefile src/Makefile.nw > Makefile
```

```
view:
```

```
    evince doc/check204.pdf &
```

```
clean:
```

```
    rm doc/check204.pdf bin/check204
```

# Содержание

<b>1</b>	<b>Анализ задания</b>	<b>1</b>
1.1	Описание проверки . . . . .	1
1.2	Запуск программы . . . . .	2
<b>2</b>	<b>Структура</b>	<b>2</b>
<b>3</b>	<b>Реализация</b>	<b>3</b>
3.1	Проверка аргументов . . . . .	3
3.2	Закачивание описания . . . . .	4
3.3	Тестирование программы . . . . .	4
3.3.1	Проверка наличия файлов . . . . .	5
3.3.2	Тест вывода из программы . . . . .	6
3.4	Отправка на проверку . . . . .	6
3.4.1	Создание временного каталога . . . . .	7
3.4.2	Клонирование каталога . . . . .	7
3.5	Загрузка модулей . . . . .	8
3.6	Задание констант . . . . .	8
<b>4</b>	<b>Установка в домашний каталог</b>	<b>8</b>
4.1	Makefile . . . . .	9