Основы программной инженерии

Лабораторная работа 1

черновик
версия 0.6

Целью лабораторной работы является знакомство с системой управления версиями Git и системой ведения проекта GitLab.

В ходе выполнения лабораторной работы студенты должны приобрести навыки работы

1. с командами git для локальной работы,
2. с некоторыми компонентами (issues, wiki) GitLab.

1. Условие задачи

Исходными данными для выполнения лабораторной работы является исходный код программы, которая решает некоторую задачу. Программа состоит из модуля, реализующего основные функции, и основной программы. Основная программа содержит функцию, которая формирует массив с тестовыми данными, и вызов функций из модуля. Программа успешно работает на простых тестовых данных, т.е. находится в том состоянии, когда ее можно поместить под версионный контроль.

В программе допущена ошибка. Эту ошибку вам надо найти и исправить.

Допущенная в программе ошибка не связана с вводом некорректных данных или с передачей в функции массивов нулевой длины. При поиске ошибки отталкивайтесь от предложенной реализации основной функции.

Выполнение лабораторной работы включает следующие крупные шаги:

1. Чтение от начала и до конца методических указаний, проработка непонятных моментов, подготовка плана (для себя) выполнения лабораторной работы.
2. Создание локального репозитория; помещение исходной программы под версионный контроль.
3. Разработка теста, для которого задача решается неверно; добавление теста к программе; фиксация изменений.
4. Составление отчета (issue) об ошибке.
5. Исправление ошибки; фиксация изменений; закрытие отчета об ошибке.
6. Подготовка отчета о проделанной работе (с использованием wiki). Заполнение отчета происходит в течение всего времени выполнения лабораторной работы.

Замечания.

1. Команды, приведенные в данном руководстве, нужно набирать в консоли самостоятельно (!), а не копировать из руководства! Скопированную команду git может понять неправильно из-за проблем с кодировкой.
2. Комментарии к фиксациям изменений нужно писать на английском языке и при этом стараться делать их максимально понятными, но короткими.
3. Приводимые в примерах имена файлов, выдача команд и т.п. могут отличаться от имен файлов, выдачи команд и т.п. на вашем компьютере.

2. Создание локального репозитория

Запустите командную оболочку MSYS.

После запуска командной оболочки вы будете находиться в так называемой «домашней папке». Узнать имя папки, в которой вы находитесь, можно с помощью команды pwd, а проанализировать ее содержимое с помощью команды ls.

```
$ pwd
/c/msys64/home/student

$ ls –a
. .. .bash_history .bash_logout .bash_profile и др.
```

Пояснения.

1. В Linux разделителем имен в пути является символ “/” (в Windows – “\”).
2. В оболочке MSYS для именования дисков используется путь “/c/” вместо “c:\”, которая используется в Windows.
3. Ключ “-a” команды ls «заставляет» ее выводить все имена, в том числе и те, которые начинаются с символа “.”.
4. “.” – короткое имя текущей папки (в нашем случае “/c/mys64/home/student”). “..” – короткое имя «родительской» папки (в нашем случае “/c/mys64/home”). Имена, которые начинаются с точки, в Linux как правило обозначают имена служебных папок или файлов и не отображаются.
5. Удобно использовать команду ls со следующим набором ключей “-lah” (“l” –табличная выдача, “a” – отображение скрытых файлов/папок, “h” – отображение размера файлов в «нормальном» виде).

Вам необходимо создать отдельную папку для выполнения лабораторной работы в домашней папке или перейти в ту папку, которая вам удобна. Для создания папки используется команда mkdir, для перехода по папкам - cd. Последняя может работать как с абсолютными путями, так и с относительными.

```
// Создадим папку для выполнения лабораторной работы
$ mkdir work

$ ls
work

// Перейдем в папку work, используя относительный путь
$ cd work

$ pwd
/c/msys64/home/student/work

// Вернемся в «домашнюю» папку
```

```
$ cd ..

$ pwd
/c/msys64/home/student

// Перейдем в папку work, используя абсолютный путь
$ cd /c/msys64/home/student/work

$ pwd
/c/msys64/home/student/work
```

Замечание.
Чтобы попасть в «домашнюю» папку из любой другой папки можно воспользоваться командой cd ~ .

Для создания локального репозитория воспользуемся командой “git init”. Прежде, чем выполнить эту команду, убедитесь, что вы находитесь в папке work. Если это не так, перейдите в папку work.

```
$ git init
// При успехе должно появиться
Инициализирован пустой репозиторий Git в /home/student/work/.git/

// Убедимся, что папка теперь непустая
$ ls –a

. .. .git
```

3. Подготовка к работе

Проверьте задано ли имя пользователя и адрес электронной почты.

```
// Проверим задано ли имя пользователя и e-mail
$ git config --list

core.repositoryformatversiion=0
core.filemode=false
...
```

Если в выдаче отсутствуют переменные user.name и user.email, значит имя и адрес электронной почты не заданы. Установите значение для этих переменных (это также нужно сделать, если имя пользователя или адрес электронной почты не соответствуют вашим). В качестве имени пользователя укажите логин, который вы используете для входа в GitLab. В качестве адреса электронной почты, укажите адрес, который указывали при регистрации в GitLab.

```
$ git config user.name xyz
$ git config user.email xyz@xyz
$ git config --list
```

```
core.repositoryformatversiion=0
core.filemode=false
...
user.name=xyz
user.email=xyz@xyz
```

Поместите в папку для выполнения лабораторной работы исходный код программы из архива.

```
// Скопируем в эту папку исходный код программы любым удобным способом

// Убедимся, что все получилось
$ pwd

/c/msys64/home/student/work

$ ls

iarray.py main.py
```

Убедитесь в том, что программа работает на тестовом примере.

После запуска программы в папке лабораторной работы work, скорее всего, появится папка __pycache__, которая содержит байт-код Python.

```
$ ls

__pycache__ iarray.py main.py
```

В репозитории следует хранить только исходные тексты (а не исполняемые или объектные файлы). Чтобы сообщить git какие файлы не нужно отслеживать, воспользуемся специальным файлом, который называется .gitignore. В нашем случае этот файл должен располагаться непосредственно в папке рабочей копии.

В папке рабочей копии (в данном примере c:\msys64\home\student\work) с помощью текстового редактора Notepad++ создайте файл, который будет содержать следующую строку

```
**/__pycache__/**
```

(pycahe начинается с двух подряд идущих символов подчеркивания и ими же заканчивается). Файл сохраните под именем .gitignore («имя» файла начинается с точки, никакого расширения у файла быть не должно).

Замечание.

Для упрощения создания файла .gitignore в командной оболочке можно использовать команду touch .gitignore, а уже после его создания отредактировать его с помощью vim или Notepad++.

```
$ git status

На ветке master
Начальный коммит
Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено...

        .gitignore
        iarray.py
        main.py

ничего не добавлено в коммит, но есть неотслеживаемые файлы ...

$ git add .gitignore

$ git status

На ветке master
Начальный коммит
Изменения, которые будут включены в коммит:
  (используйте «git reset HEAD <файл>...» чтобы убрать из индекса)

        новый файл:    .gitignore

Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено...

        iarray.py
        main.py

// Зафиксируем изменения
$ git commit –m ".gitignore was added."
[master (корневой коммит) 5dc5e86] .gitignore was added.
```

Команда commit отображает информацию о фиксации: имя ветки, в которой выполнена фиксация (***master***), контрольную сумму фиксации (***5dc5e86***), сколько файлов было изменено, статистику по добавленным/удаленным строкам.

git для идентификации ревизий использует значение хэша фиксации. Главная причина этого - git децентрализованная система контроля версий и поэтому монотонной сквозной нумерации фиксации в ней быть просто не может.

После этого добавьте под версионный контроль саму программу.

```
// Добавим остальные файлы под версионный контроль
$ git add iarray.py main.py

$ git status
```

```
На ветке master
Изменения, которые будут включены в коммит:
  (используйте «git reset HEAD <файл>...», чтобы убрать из индекса)

        новый файл:    iarray.py
        новый файл:    main.py

// Благодаря .gitignore, папка __pycache__ проигнорирована.

// Зафиксируем изменения
$ git commit –m "Initial version of program was added."
```

4. Выполнение лабораторной работы

Нетривиальные моменты и результат выполнения каждого задания из данного раздела должны быть отражены в отчете.

4.1. Назначение программы

Опишите назначение программы и каждой ее функции.

4.2. Поиск ошибки и ее описание

- Найдите тест, на котором программа будет работать неверно. Оформите его как функцию, которая не принимает никаких параметров, а возвращает пару список, количество элементов. Добавьте вызов новой тестовой функции (вызов старой тестовой процедуры оставьте, потому что количество тестов должно только увеличиваться). Сохраните ваши изменения.
  Замечание.
  На данном этапе необходимо только добавить в программу тест, демонстрирующий, что основной алгоритм работает неверно. Исправлять сам программный код не нужно.

- Проанализируйте ваши изменения с помощью команд git status и git diff. Результат анализа приведите в отчете.
  Замечание.
  Вам необходимо уметь объяснить преподавателю результат, который выдает команда diff.

- Зафиксируйте ваши изменения с помощью команды git commit. В отчете приведите номер ревизии и комментарий, которым вы сопроводили фиксацию.

- Создайте отчет об ошибке (issue) в GitLab:
  - Войдите в GitLab (после этого вы окажетесь на странице ваших проектов).
  - Щелкните по имени проекта, созданного для курса «Основы программной инженерии».

- Щелкните по ссылке “Issues” в верней части страницы.
- Создайте отчет об ошибке (кнопка “New Issue”).
  Обязательно нужно заполнить поля “Title” (заголовок) и “Description” (полное описание). При заполнении этих полей вспомните на какие моменты обращалось внимание на семинаре. В полном описании обязательно укажите номер ревизии, в которой ошибка была найдена. Назначьте ошибку на себя.
  Замечание.
  Для описания полей “Title”, “Description” используйте русский язык.

4.3. Исправление ошибки

- Исправьте ошибку и убедитесь в правильности ваших исправление.

- Перед фиксацией изменений проанализируйте их с помощью команд git status и git diff. Результаты анализа приведите в отчете.

- Зафиксируйте ваши изменения.

- Закройте отчет об ошибке. При этом в комментарии укажите суть исправлений, приведите номер ревизии.

4.4. Анализ истории

- Проанализируйте историю изменений с помощью команды git log.

- Что изменится в выдаче этой команды, если вы добавите параметр “—name-status”?

- С помощью какого параметра, можно анализировать историю не за весь период, а, например, между двумя определенными ревизиями?

- Каким образом можно сравнить две версии одного и того же файла, но разных ревизий?

- Научитесь анализировать историю изменений с помощью GitLab.

4.5. Работа с wiki в GitLab

Войдите в GitLab (вы окажетесь на странице ваших проектов). Щелкните по имени проекта, созданного для курса «Основы программной инженерии». Щелкните по ссылке “Wiki” в верней части страницы.

Замечание.
Страницы wiki хранятся в отдельном репозитории (адрес этого репозитория можно посмотреть на странице “Wiki” -> “Git Access”). С этим репозиторием можно работать с помощью команд git.

В поле “Content” наберите текст “[Лабораторная работа 1 (знакомство с wiki)](lab_01_test)” (текст набирается без кавычек), затем нажмите на кнопку “Create page”. После этого вы увидите главную страницу вашей Wiki, на которой будет размещена ссылка на только что созданную страницу lab_01_test.

Перейдите на эту страницу и в поле “Content” вставьте текст, из таблицы 1. Сохраните страницу.

Сравните представление текста в таблице 1 и на странице в Wiki. Выполните задания, которые приведены в конце страницы. Результаты их выполнения должны быть отражены в отчете.

Для получения дополнительной справочной информации воспользуйтесь ссылкой “Markdown”, которая становится доступной при редактировании страницы, или источником [4].

Таблица 1.

````
Для создания страницы достаточно записать конструкцию вида `[название ссылки](имя страницы)`.

# 1. Оформление текста

Параграф - это несколько предложений, которые "примыкают" друг к другу (т.е. между ними не должно быть пустой строки).
Это все еще первый параграф, хотя это предложение расположено на новой строке.

Это уже второй параграф. Между параграфами - пустая строка.

Вот таким способом *можно* **менять** ~~стиль~~ шрифта.

# 2. Оформление кода на каком-нибудь языке

```c
int test(int a)
{
    printf("%d\n", a);
}
```

```
Здесь внутри *форматирование* **wiki** не работает.
```
# 3. Ссылки на разные ресурсы.

Чтобы все заработало указанные объекты должны уже существовать в вашем репозитории или GitLab.

#1 - ссылка на issue

!1 - ссылка на merge request
````

```
XYZ - ссылка на номер ревизии (XYZ нужно поменять но номер ревизии из вашего репозитория)

[файл](.gitignore)

URL-адрес можно вставить просто https://www.google.com, а можно сделать [красиво](https://www.google.com)

# 4. Оформление списков и таблиц

1. Элемент списка 1
2. Элемент списка 2
 * подсписок 1
 * подсписок 2 (размер вложенности имеет значение как в python)
2. Элемент списка 3 (в начале не ошибка, а особенность)
4. Элемент списка 4

|Столбец 1|Столбец 2|
|---------|---------|
|1        |2        |

# 5. Что нужно изучить самостоятельно

1. Как добавлять рисунки.
2. Как сделать оглавление.
3. Как добавить ссылки для перехода между страницами wiki.
4. Как писать комментарии.
5. Стили оформления таблиц.
```

5. FAQ

Проблема.
При выполнении команды git XYZ выдается сообщение «fatal: Не найден git репозитоий (или один из его каталогов): .git» или «fatal: This operation must be run in a work tree»
Решение.
Вы находитесь вне папки рабочей копии или внутри папки .git. Перейдите в папку рабочей копии.

6. Литература

1. Pro Git (https://git-scm.com/book/ru/v2)
2. Unix shell: абсолютно первые шаги (https://habrahabr.ru/post/267825/)
3. Документация по языку разметки markdown в Gitlab (https://docs.gitlab.com/ce/user/markdown.html)