

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
дисциплины «Программирование на Python»

Выполнила:
Кубанова Ксения Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А.

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Тема: Лабораторная работа 1.1. Исследование основных возможностей Git и GitHub

Цель: исследовать базовые возможности системы контроля версий Git и веб-сервиса для хостинга IT-проектов GitHub.

Теоретический материал

Системы контроля версий

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

Локальные системы контроля версий

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже, директорию с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели. Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.

Централизованные системы контроля версий

Следующая серьёзная проблема, с которой сталкиваются люди, — это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища.

Распределённые системы контроля версий

Здесь в игру вступают распределённые системы контроля версий (РСКВ). В РСКВ (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени) — они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.

Git

Основное отличие Git от любой другой СКВ (включая Subversion и её собратьев) — это подход к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы (CVS, Subversion, Perforce, Bazaar и т. д.) представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени (обычно это называют контролем версий, основанным на различиях).

В Git для всего вычисляется хеш-сумма, и только потом происходит сохранение. В дальнейшем обращение к сохранённым объектам происходит по этой хеш-сумме. Это значит, что невозможно изменить содержимое файла или директории так, чтобы Git не узнал об этом. Данная функциональность встроена в Git на низком уровне и является неотъемлемой частью его философии. Вы не потеряете информацию во время её передачи и не получите повреждённый файл без ведома Git.

Когда вы производите какие-либо действия в Git, практически все из них только добавляют новые данные в базу Git. Очень сложно заставить систему удалить данные либо сделать что-то, что нельзя впоследствии отменить. Как и в любой другой СКВ, вы можете потерять или испортить свои изменения, пока они не зафиксированы, но после того, как вы зафиксируете снимок в Git, будет очень сложно что-либо потерять, особенно, если вы регулярно синхронизируете свою базу с другим репозиторием.

Порядок выполнения работы

1. Создание общедоступного репозитория. Была выбрана лицензия MIT и язык C++.

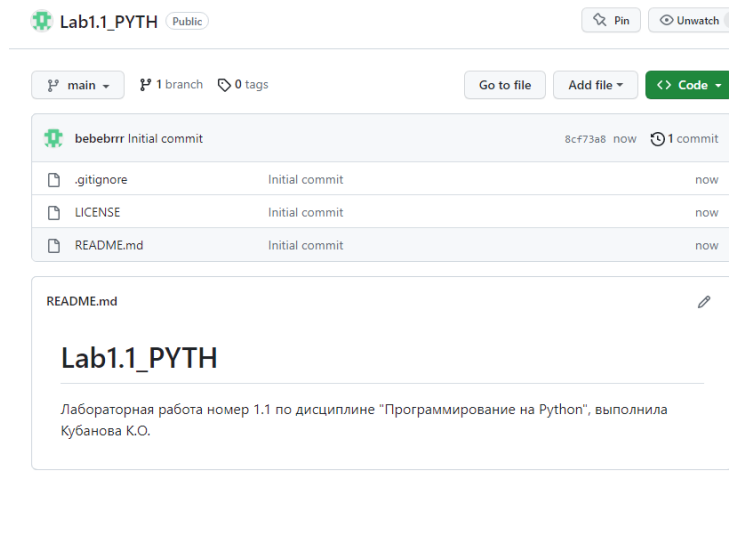


Figure 1. Созданный репозиторий

2. Выполнение клонирования репозитория на компьютер. Для данного действия в терминале была введена команда «git clone “ссылка”».

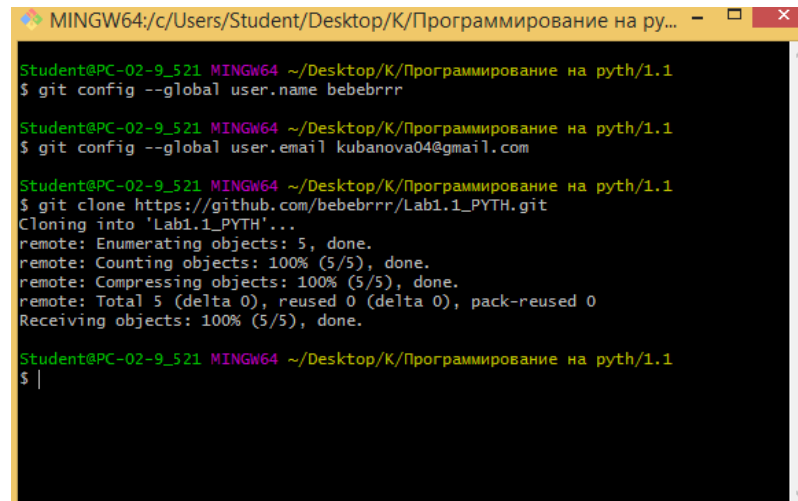
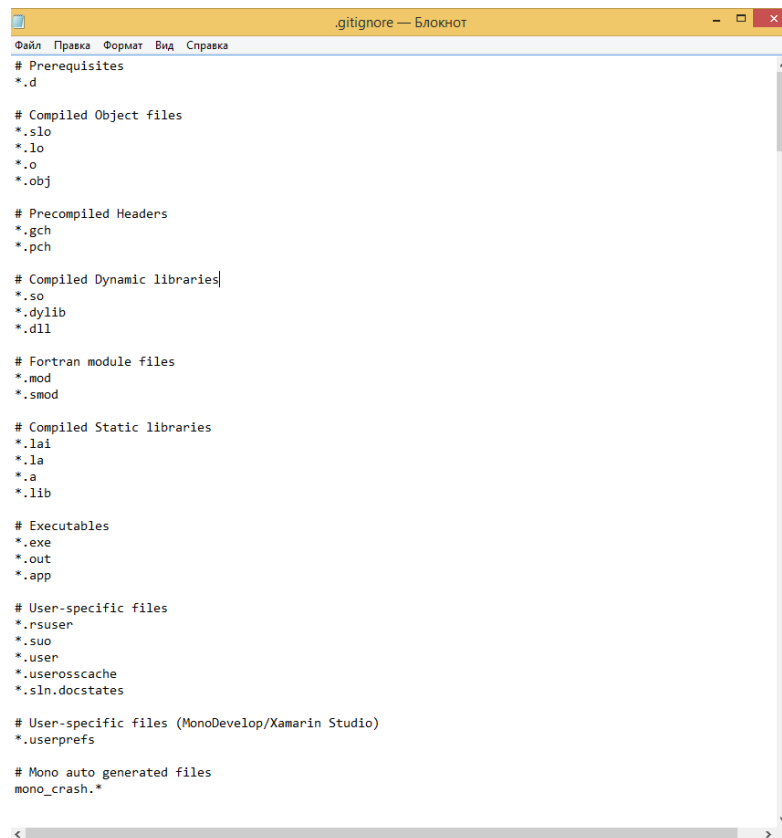


Figure 2. Клонирование

3. Дополнение .gitignore. Для этого были скопированы данные из соответствующих txt файлов в предложенных ссылках теоретического материала.

A screenshot of a text editor window titled ".gitignore — Блокнот". The window contains a .gitignore file with various patterns for ignoring files. The patterns are grouped by comments: "# Prerequisites", "# Compiled Object files", "# Precompiled Headers", "# Compiled Dynamic libraries", "# Fortran module files", "# Compiled Static libraries", "# Executables", "# User-specific files", "# User-specific files (MonoDevelop/Xamarin Studio)", and "# Mono auto generated files".

```
.gitignore — Блокнот
Файл Правка Формат Вид Справка

# Prerequisites
*.d

# Compiled Object files
*.slo
*.lo
*.o
*.obj

# Precompiled Headers
*.gch
*.pch

# Compiled Dynamic libraries
*.so
*.dylib
*.dll

# Fortran module files
*.mod
*.smod

# Compiled Static libraries
*.lai
*.la
*.a
*.lib

# Executables
*.exe
*.out
*.app

# User-specific files
*.rsuser
*.suo
*.user
*.userosscache
*.sln.docstates

# User-specific files (MonoDevelop/Xamarin Studio)
*.userprefs

# Mono auto generated files
mono_crash.*
```

Figure 3. Дополнение .gitignore

4. Добавление информации о создателе. Для этого следовало дополнить файл README.md на определённом языке программирования.
5. Создание 7 коммитов. В txt файле была написана небольшая программа и каждое обновление фиксировалось коммитами.

```
MINGW64/c/Users/Student/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git commit -m "Инициализация переменных"
[main 2251fcb] Инициализация переменных
1 file changed, 1 insertion(+)

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git add 1.1.txt

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git commit -m "8воd a"
[main b68eae] 8воd a
1 file changed, 3 insertions(+)

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git add 1.1.txt

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git commit -m "8воd b"
[main ad6b77c] 8воd b
1 file changed, 2 insertions(+), 1 deletion(-)

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git add 1.1.txt

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git commit -m "8воd c"
[main 4d67f7e] 8воd c
1 file changed, 2 insertions(+)

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git add 1.1.txt

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git commit -m "8воd d"
[main e235032] 8воd d
1 file changed, 2 insertions(+)

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git add 1.1.txt

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git commit -m "Нахождение суммы введенных переменных"
[main 235a9ef] Нахождение суммы введенных переменных
1 file changed, 3 insertions(+), 1 deletion(-)

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git push
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 4 threads
Compressing objects: 100% (24/24), done.
Writing objects: 100% (24/24), 2.12 KiB | 1.06 MiB/s, done.
Total 24 (delta 14), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (14/14), done.
To https://github.com/bebebrrr/lab1.1_PYTH.git
7c6c73e..235a9ef main -> main

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$
```

Figure 4. 7 коммитов

6. Добавление файла README. Для этого в консоли использовались команды “git add”, “git push”.

```
Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH
(main)
$ git add README.md

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH (main)
$ git commit -m "Обновленный"
[main f41f2a0] Обновленный
1 file changed, 2 insertions(+)

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 532 bytes | 532.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/bebebrrr/lab1.1_PYTH.git
235a9ef..f41f2a0 main -> main

Student@PC-02-9_521 MINGW64 ~/Desktop/К/Программирование на pyth/1.1/lab1.1_PYTH (main)
$
```

Figure 5. Добавление файла README

Вопросы для защиты работы

1. Что такое СКВ и каково ее назначение?

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

2. В чем недостатки локальных и централизованных СКВ?

Недостатки локальных и централизованных систем контроля версий (СКВ) могут варьироваться в зависимости от конкретной реализации и требований проекта или организации. Некоторые общие недостатки таких систем могут включать в себя:

1) Ограниченность доступа: Локальные СКВ могут ограничивать доступ к коду только на уровне локальной сети, что может быть неудобно для удаленных сотрудников или распределенных команд. Централизованные СКВ также могут ограничивать доступ к коду, особенно если сервер с центральным репозиторием находится внутри организации.

2) Недоступность при сбоях: Локальные СКВ могут столкнуться с проблемами доступности при отказе сервера или проблемах с сетью. Централизованные СКВ также могут стать недоступными, если сервер с центральным репозиторием выходит из строя или возникают проблемы с сетью.

3) Отсутствие отслеживания изменений оффлайн: Локальные СКВ могут не предоставлять возможности для отслеживания изменений, сделанных в репозитории в оффлайн режиме, до их отправки на сервер. Это может вызывать проблемы, если разработчик работает без подключения к сети и забывает отправить изменения.

4) Отсутствие гранулярного контроля доступа: Централизованные СКВ могут не обеспечивать достаточно гранулярного контроля доступа к репозиторию. Это означает, что все разработчики, имеющие доступ к центральному серверу, имеют доступ ко всему коду проекта, что может привести к возможности несанкционированного доступа или нежелательным изменениям.

3. К какой СКВ относится Git?

Относится к классу распределенных СКВ.

4. В чем концептуальное отличие Git от других СКВ?

Каждый участник имеет полную копию репозитория.

5. Как обеспечивается целостность хранимых данных в Git?

С помощью хэш-суммы.

6. В каких состояниях могут находиться файлы в Git? Как связаны эти состояния?

Измененные, ожидающие фиксации, зафиксированные. Связь прослеживается в том, что это три последовательных состояния файлов при переходе.

7. Что такое профиль пользователя в GitHub?

публичная страница GitHub

8. Какие бывают репозитории в GitHub?

публичные, приватные.

9. Укажите основные этапы модели работы с GitHub.

Upseteam, origin, local, working copy.

10. Как осуществляется первоначальная настройка Git после установки?

Проверка версии: git version

Ввод имени и почты:

```
git config --global user.name <YOUR_NAME>
```

```
git config --global user.email <EMAIL>
```

11. Опишите этапы создания репозитория в GitHub.

new repositories в правом верхнем углу. Далее создание, т.е. имя, описание, вид, выбор .gitignore и LICENSE.

12. Какие типы лицензий поддерживаются GitHub при создании репозитория?

| License | License keyword |
|-----------------------------------|-----------------|
| Academic Free License v3.0 | afl-3.0 |
| Apache license 2.0 | apache-2.0 |
| Artistic license 2.0 | artistic-2.0 |
| Boost Software License 1.0 | bsl-1.0 |
| BSD 2-clause "Simplified" license | bsd-2-clause |

| | |
|---|--------------------|
| BSD 3-clause "New" or "Revised" license | bsd-3-clause |
| BSD 3-clause Clear license | bsd-3-clause-clear |
| BSD 4-clause "Original" or "Old" license | bsd-4-clause |
| BSD Zero-Clause license | 0bsd |
| Creative Commons license family | cc |
| Creative Commons Zero v1.0 Universal | cc0-1.0 |
| Creative Commons Attribution 4.0 | cc-by-4.0 |
| Creative Commons Attribution ShareAlike 4.0 | cc-by-sa-4.0 |
| Do What The F*ck You Want To Public License | wtfpl |
| Educational Community License v2.0 | ecl-2.0 |
| Eclipse Public License 1.0 | epl-1.0 |
| Eclipse Public License 2.0 | epl-2.0 |
| European Union Public License 1.1 | eupl-1.1 |
| GNU Affero General Public License v3.0 | agpl-3.0 |
| GNU General Public License family | gpl |
| GNU General Public License v2.0 | gpl-2.0 |
| GNU General Public | gpl-3.0 |

| | |
|---|------------|
| License v3.0 | |
| GNU Lesser General Public License family | lgpl |
| GNU Lesser General Public License v2.1 | lgpl-2.1 |
| GNU Lesser General Public License v3.0 | lgpl-3.0 |
| ISC | isc |
| LaTeX Project Public License v1.3c | lppl-1.3c |
| Microsoft Public License | ms-pl |
| MIT | mit |
| Mozilla Public License 2.0 | mpl-2.0 |
| Open Software License 3.0 | osl-3.0 |
| PostgreSQL License | postgresql |
| SIL Open Font License 1.1 | ofl-1.1 |
| University of Illinois/NCSA Open Source License | ncsa |
| The Unlicense | unlicense |
| zLib License | zlib |

13. Как осуществляется клонирование репозитория GitHub?

Зачем нужно клонировать репозиторий?

git clone, для создания репозитория на локальном компьютере.

14. Как проверить состояние локального репозитория Git?

Git status

15. Как изменяется состояние локального репозитория Git после выполнения следующих операций: добавления/изменения файла в локальный репозиторий Git; добавления нового/ измененного файла под версионный контроль с помощью команды git add ; фиксации (коммита)

изменений с помощью команды `git commit` и отправки изменений на сервер с помощью команды `git push` ?

После добавления/изменения файла в локальный репозиторий Git файл будет находиться в рабочем каталоге.

Файл не будет автоматически добавлен в индекс (staging area) для фиксации.

Добавление нового/измененного файла под версионный контроль с помощью команды `git add`:

Команда `git add` используется для добавления файлов в индекс (staging area) для фиксации.

После выполнения команды `git add`, файлы будут добавлены в индекс и будут готовы для фиксации (коммита).

Фиксация (коммит) изменений с помощью команды `git commit`:

Команда `git commit` используется для фиксации изменений в локальном репозитории Git.

После выполнения команды `git commit`, изменения будут зафиксированы и сохранены в истории репозитория.

Отправка изменений на сервер с помощью команды `git push`:

Команда `git push` используется для отправки изменений из локального репозитория на удаленный сервер Git.

После выполнения команды `git push`, изменения будут отправлены на сервер и будут доступны для других пользователей.

16. У Вас имеется репозиторий на GitHub и два рабочих компьютера, с помощью которых Вы можете осуществлять работу над некоторым проектом с использованием этого репозитория. Опишите последовательность команд, с помощью которых оба локальных репозитория, связанных с репозиторием GitHub будут находиться в синхронизированном состоянии. Примечание: описание необходимо начать с команды `git clone` .

`git clone <URL репозитория> <путь к локальной папке>`

17. GitHub является не единственным сервисом, работающим с Git. Какие сервисы еще Вам известны? Приведите сравнительный анализ одного из таких сервисов с GitHub.

GitLab: GitLab предлагает платформу для управления репозиториями Git и предоставляет инструменты для непрерывной интеграции и развертывания (CI/CD), отслеживания проблем и управления задачами. В отличие от GitHub, GitLab является полностью открытым исходным кодом и позволяет развернуть инстанс на собственных серверах.

Bitbucket: Bitbucket предлагает управление репозиториями Git, а также репозиториями Mercurial. В отличие от GitHub, Bitbucket предоставляет бесплатные закрытые репозитории для небольших команд.

18. Интерфейс командной строки является не единственным и далеко не самым удобным способом работы с Git. Какие Вам известны программные средства с графическим интерфейсом пользователя для работы с Git? Приведите как реализуются описанные в лабораторной работе операции Git с помощью одного из таких программных средств.

Существует несколько программных средств с графическим интерфейсом для работы с Git.

Sourcetree: Sourcetree является удобным графическим клиентом для Git и Mercurial. Он предоставляет интуитивно понятный интерфейс для выполнения основных операций Git, таких как клонирование репозитория, создание веток, коммиты, слияния, откат изменений и просмотр истории изменений. Он также интегрируется с различными сервисами хостинга репозитория, такими как GitHub, Bitbucket и GitLab.

Пример реализации операций Git с помощью Sourcetree:

- Клонирование репозитория: Вы можете просто ввести URL репозитория и выбрать место для клонирования.

- Создание ветки: Sourcetree предлагает интуитивный интерфейс для создания и переключения между ветками.

- Коммиты: Вы можете просмотреть список изменений, выбрать файлы для коммита и добавить комментарий перед выполнением коммита.

- Слияние: Sourcetree предоставляет возможность выполнить слияние веток с помощью интуитивного интерфейса.

Вывод: Git нужен, чтобы записывать состояние проекта, ходить туда-сюда по сохранениям, хранить свой код онлайн, делиться кодом с друзьями.