

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Основы кроссплатформенного программирования

Отчет по лабораторной работе №2

**Исследование
основных возможностей Git и GitHub**

Выполнил студент группы

ИВТ-б-о-21-1

Назаров Н.Ю. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил доцент

Кафедры инфокоммуникаций, старший
преподаватель

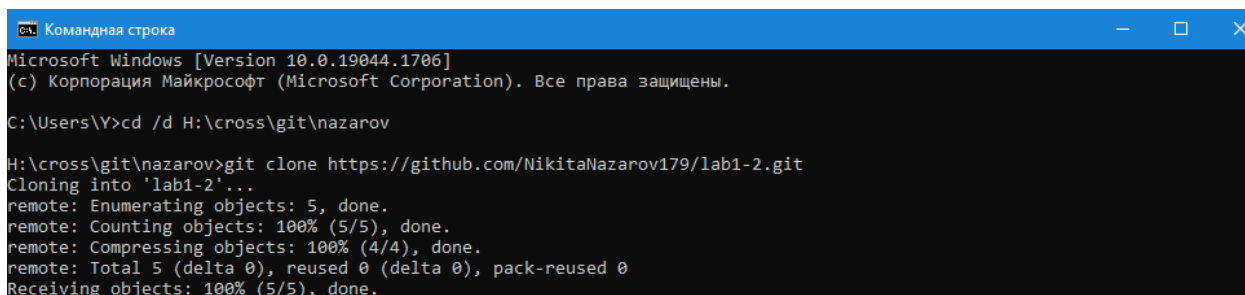
Воронкин Р.А.

(подпись)

Тема: исследование возможностей Git для работы
с локальными репозиториями.

Цель работы: исследовать базовые возможности системы
контроля версий Git для работы с локальными репозиториями.

Выполнение работы.

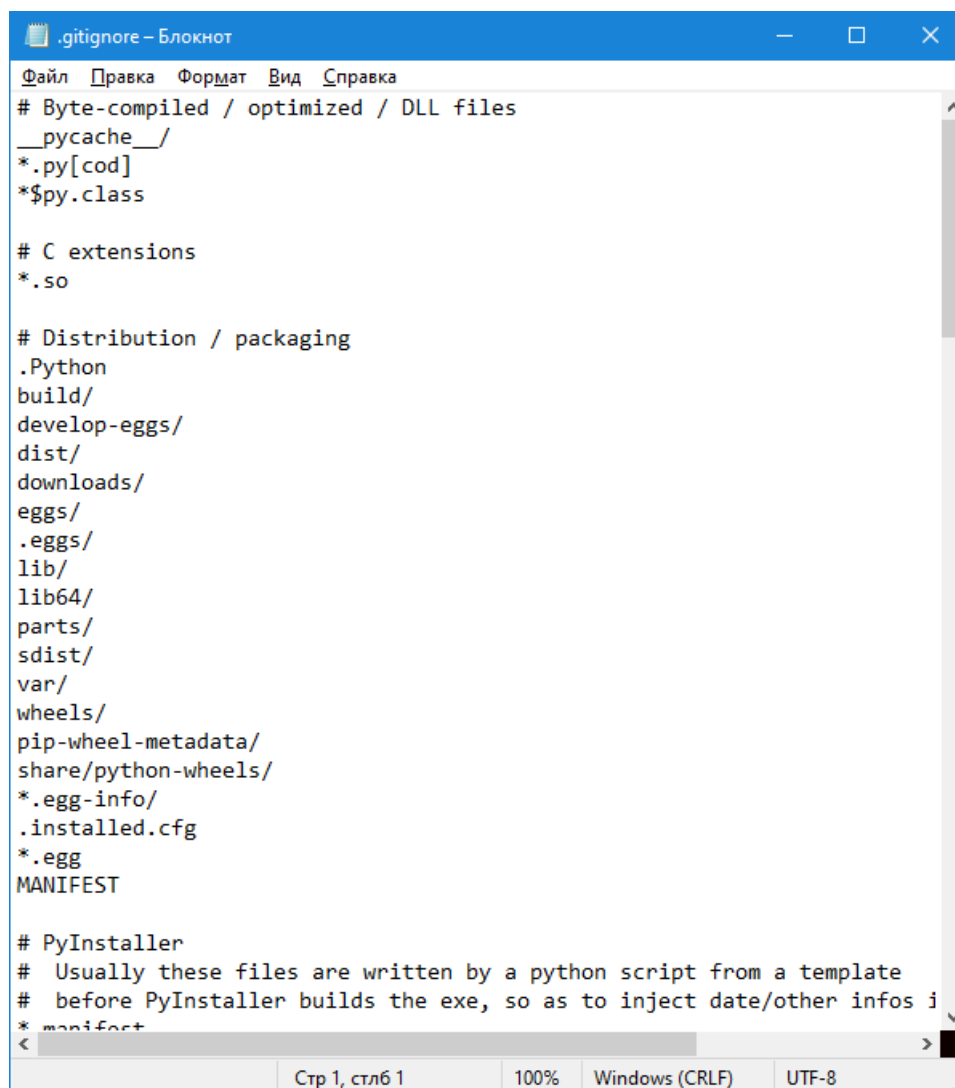


```
Командная строка
Microsoft Windows [Version 10.0.19044.1706]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Y>cd /d H:\cross\git\nazarov

H:\cross\git\nazarov>git clone https://github.com/NikitaNazarov179/lab1-2.git
Cloning into 'lab1-2'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 1 – клонирование репозитория на устройство



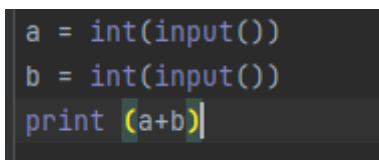
```
.gitignore – Блокнот
Файл  Правка  Формат  Вид  Справка
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
pip-wheel-metadata/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# PyInstaller
# Usually these files are written by a python script from a template
# before PyInstaller builds the exe, so as to inject date/other infos i
*manifest
```

Рисунок 2 – редактирование файла .gitignore



```
a = int(input())
b = int(input())
print (a+b)
```

Рисунок 3 – первоначальный вариант программы

```
H:\cross\git\nazarov\lab1-2>git commit -m "1_variant_programmi"
[main ccefe5d] 1_variant_programmi
8 files changed, 39 insertions(+)
create mode 100644 .idea/.gitignore
create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
create mode 100644 .idea/lab1-2.iml
create mode 100644 .idea/misc.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/vcs.xml
create mode 100644 lab1-2.py
```

Рисунок 4 – коммит

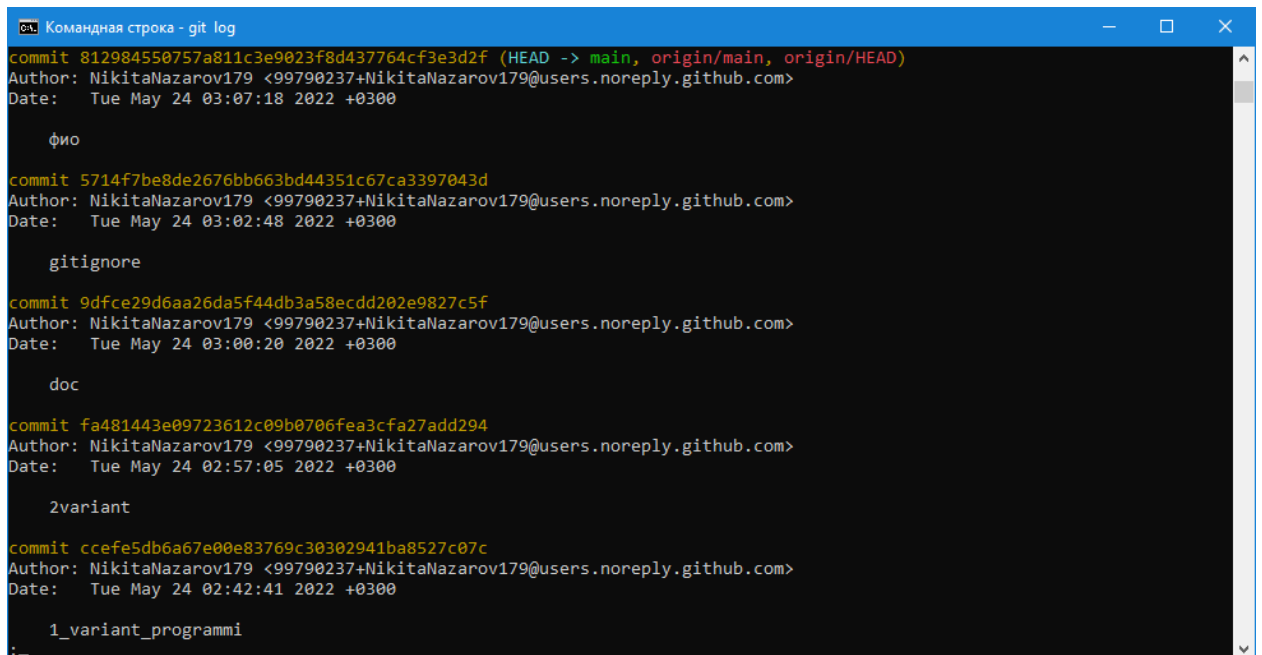
```
a = int(input())
b = int(input())
print (a+b)/(a*b)
print (a)
print (b)
```

Рисунок 5 – измененный вариант программы

```
H:\cross\git\nazarov\lab1-2>git commit -m "2variant"
[main fa48144] 2variant
3 files changed, 5 insertions(+), 2 deletions(-)
create mode 100644 .idea/.name

H:\cross\git\nazarov\lab1-2>
```

Рисунок 6 – коммит



```
Командная строка - git log
commit 812984550757a811c3e9023f8d437764cf3e3d2f (HEAD -> main, origin/main, origin/HEAD)
Author: NikitaNazarov179 <99790237+NikitaNazarov179@users.noreply.github.com>
Date: Tue May 24 03:07:18 2022 +0300

    фюо

commit 5714f7be8de2676bb663bd44351c67ca3397043d
Author: NikitaNazarov179 <99790237+NikitaNazarov179@users.noreply.github.com>
Date: Tue May 24 03:02:48 2022 +0300

    gitignore

commit 9dfce29d6aa26da5f44db3a58ecdd202e9827c5f
Author: NikitaNazarov179 <99790237+NikitaNazarov179@users.noreply.github.com>
Date: Tue May 24 03:00:20 2022 +0300

    doc

commit fa481443e09723612c09b0706fea3cfa27add294
Author: NikitaNazarov179 <99790237+NikitaNazarov179@users.noreply.github.com>
Date: Tue May 24 02:57:05 2022 +0300

    2variant

commit ccefe5db6a67e00e83769c30302941ba8527c07c
Author: NikitaNazarov179 <99790237+NikitaNazarov179@users.noreply.github.com>
Date: Tue May 24 02:42:41 2022 +0300

    1_variant_programmi
:
```

Рисунок 7 – результат выполнения команды git log

```
Командная строка
H:\cross\git\nazarov\lab1-2>git show HEAD
commit 812984550757a811c3e9023f8d437764cf3e3d2f (HEAD -> main, origin/main, origin/HEAD)
Author: NikitaNazarov179 <99790237+NikitaNazarov179@users.noreply.github.com>
Date: Tue May 24 03:07:18 2022 +0300

    фиио

diff --git a/.idea/.name b/.idea/.name
index 6d83af1..42061c0 100644
--- a/.idea/.name
+++ b/.idea/.name
@@ -1,1 @@
- lab1-2.py
\ No newline at end of file
+README.md
\ No newline at end of file
diff --git a/README.md b/README.md
index 65b36d2..d49db2a 100644
--- a/README.md
+++ b/README.md
@@ -1,3 @@
-# lab1-2
\ No newline at end of file
+# lab1-2
+Назаров Никита Юрьевич
+ИБТ-6-о-21-1
\ No newline at end of file
H:\cross\git\nazarov\lab1-2>
```

Рисунок 8 – результат выполнения команды git log с определенными условиями

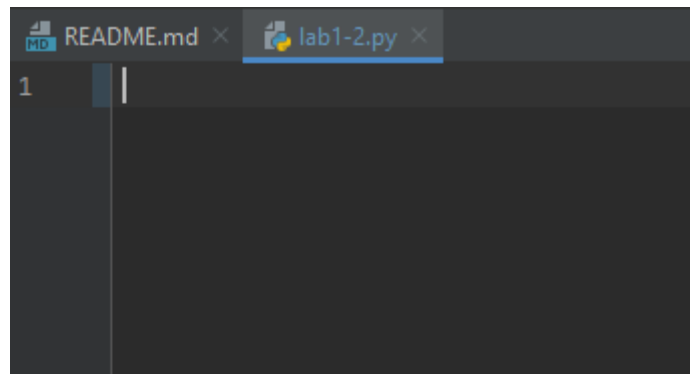


Рисунок 9 – удаление всех строк кода

```
H:\cross\git\nazarov\lab1-2>git checkout -- lab1-2.py
```

Рисунок 10 – команда git checkout

```
a = int(input())
b = int(input())
print (a+b)/(a*b)
print (a)
print (b)
```

Рисунок 11 – результат выполнения предыдущей команды

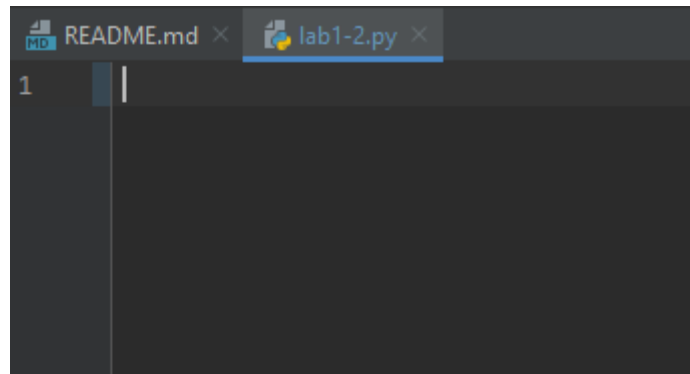


Рисунок 12 – удаление всех строк кода


```
H:\cross\git\nazarov\lab1-2>git reset --hard HEAD~1  
HEAD is now at 8129845 фю  
H:\cross\git\nazarov\lab1-2>
```

Рисунок 13 – откат версии

A screenshot of a code editor with two tabs: 'README.md' and 'lab1-2.py'. The 'lab1-2.py' tab is active, showing the following Python code:

```
a = int(input())  
b = int(input())  
print (a+b)/(a*b)  
print (a)  
print (b)
```

Рисунок 14 - результат отката



Auto DevOps

It will automatically build, test, and deploy your application based on a predefined CI/CD configuration.

Learn more in the [Auto DevOps documentation](#)

Enable in settings

main


lab1-2 / +

Find file

Web IDE

Download

Clone



фио

NikitaNazarov179 authored 38 minutes ago

81298455

README

MIT License

Add CHANGELOG

Add CONTRIBUTING

Add Kubernetes cluster

Set up CI/CD

Configure Integrations

Name	Last commit	Last update
.idea	фио	38 minutes ago
doc	doc	45 minutes ago
.gitignore	gitignore	43 minutes ago
LICENSE	Initial commit	1 week ago
README.md	фио	38 minutes ago
lab1-2.py	2variant	48 minutes ago

Рисунок 15 – репозиторий на gitlab

<https://gitlab.com/NikitaNazarov179/lab1-2>

Вывод: команда `git -checkout <FileName>` удаляет изменения произошедшие с файлом в репозитории до коммита.

Контрольные вопросы и ответы на них:

Вопросы для защиты работы.

1. Как выполнить историю коммитов в Git? Какие существуют дополнительные опции для просмотра истории коммитов?

Наиболее простой и в то же время мощный инструмент для этого — команда `git log`. По умолчанию, без аргументов, `git log` выводит список коммитов созданных в данном репозитории в обратном хронологическом порядке. То есть самые последние коммиты показываются первыми.

Одна из опций, когда вы хотите увидеть сокращенную статистику для каждого коммита, вы можете использовать опцию `–stat`.

Вторая опция (одна из самых полезных аргументов) является `-p` или `-- patch`, который показывает разницу (выводит патч), внесенную в каждый коммит. Так

же вы можете ограничить количество записей в выводе команды; используйте параметр `-2` для вывода только двух записей (пример команды `git log -p -2`).

Третья действительно полезная опция это `--pretty`. Она меняет формат вывода. Существует несколько встроенных вариантов отображения. Опция `oneline` выводит каждый коммит в одну строку, что может быть очень удоб-

ным если вы просматриваете большое количество коммитов. К тому же, опции short, full и fuller делают вывод приблизительно в том же формате, но с меньшим или большим количеством информации соответственно.

Наиболее интересной опцией является format, которая позволяет указать формат для вывода информации. Особенно это может быть полезным, когда вы хотите сгенерировать вывод для автоматического анализа — так как вы указываете формат явно, он не будет изменен даже после обновления Git.

Для опции git log --pretty=format существуют различного рода опции для изменения формата отображения.

2. Как ограничить вывод при просмотре истории коммитов?

Для ограничения может использоваться функция git log <n>, где n число записей.

Также, существуют опции для ограничения вывода по времени, такие как --since и --until, они являются очень удобными. Например, следующая команда покажет список коммитов, сделанных за последние две недели:

```
git log --since=2.weeks
```

Это команда работает с большим количеством форматов — вы можете указать определенную дату вида 2008-01-15 или же относительную дату, например 2 years 1 day 3 minutes ago.

Также вы можете фильтровать список коммитов по заданным параметрам. Опция --author дает возможность фильтровать по автору коммита, а опция --grep (показывает только коммиты, сообщение которых содержит указанную строку) искать по ключевым словам в сообщении коммита. Функция -S показывает только коммиты, в которых изменение в коде повлекло за собой добавление или удаление указанной строки.

3. Как внести изменения в уже сделанный коммит?

Внести изменения можно с помощью команды git commit --amend

Эта команда берёт индекс и применяет его к последнему коммиту. Если после последнего коммита не было никаких проиндексированных изменений (например, вы запустили приведённую команду сразу после предыдущего коммита), то состояние проекта будет абсолютно таким же и всё, что мы изменим, это комментарий к коммиту.

Для того, чтобы внести необходимые изменения - нам нужно проиндексировать их и выполнить команду `git commit --amend`.

```
git commit -m 'initial commit' git add forgotten_file
```

```
git commit --amend
```

Эффект от выполнения этой команды такой, как будто мы не выполнили предыдущий коммит, а еще раз выполнили команду `git add` и выполнили коммит.

4. Как отменить индексацию файла в Git?

Например, вы изменили два файла и хотите добавить их в разные коммиты, но случайно выполнили команду `git add *` и добавили в индекс оба. Как исключить из индекса один из них? Команда `git status` напомним вам:

Прямо под текстом «Changes to be committed» говорится: используйте `git reset HEAD <file>` для исключения из индекса.

5. Как отменить изменения в файле?

С помощью команды `git checkout -- <file>`.

6. Что такое удаленный репозиторий Git?

Удалённый репозиторий это своего рода наше облако, в которое мы сохраняем те или иные изменения в нашей программе/коде/файлах.

7. Как выполнить просмотр удаленных репозиториях данного локального репозитория?

Для того, чтобы просмотреть список настроенных удалённых репозиториях, необходимо запустить команду `git remote`.

Также можно указать ключ `-v`, чтобы просмотреть адреса для чтения и записи, привязанные к репозиторию. Пример: `git remote -v`

8. Как добавить удаленный репозиторий для данного локального репозитория?

Для того, чтобы добавить удалённый репозиторий и присвоить ему имя (shortname), просто выполните команду `git remote add <shortname> <url>`.

9. Как выполнить отправку/получение изменений с удаленного репозитория?

Если необходимо получить изменения, которые есть у Пола, но нету у вас, вы можете выполнить команду `git fetch <Название репозитория>`. Важно отметить, что команда `git fetch` забирает данные в ваш локальный репозиторий, но не сливает их с какими-либо вашими наработками и немодифицирует то, над чем вы работаете в данный момент. Вам необходимо вручную слить эти данные с вашими, когда вы будете готовы.

Если ветка настроена на отслеживание удалённой ветки, то вы можете использовать команду `git pull` чтобы автоматически получить изменения из удалённой ветки и слить их со своей текущей. Выполнение `git pull`, как правило, извлекает (fetch) данные с сервера, с которого вы изначально клонировали, и автоматически пытается слить (merge) их с кодом, над которым вы в данный момент работаете.

Чтобы отправить изменения на удалённый репозиторий необходимо отправить их в удалённый репозиторий. Команда для этого действия простая: `git push <remote-name> <branch-name>`.

10. Как выполнить просмотр удаленного репозитория?

Для просмотра удалённого репозитория, можно использовать команду `git remote show <remote>`.

11. Каково назначение тэгов Git?

Теги - это ссылки указывающие на определённые версии кода/написанной программы. Они удобны чтобы в случае чего вернуться к нужному моменту. Также при помощи тегов можно помечать важные моменты.

12. Как осуществляется работа с тэгами Git?

Просмотреть наличие тегов можно с помощью команды: `git tag`.

А назначить (указать, добавить тег) можно с помощью команды `git tag -a v1.4(версия изначальная) -m "Название"`.

С помощью команды `git show` вы можете посмотреть данные тега вместе с коммитом: `git show v1.4`.

Отправка тегов, по умолчанию, команда `git push` не отправляет теги на удалённые сервера. После создания теги нужно отправлять явно на удалённый сервер. Процесс аналогичен отправке веток — достаточно выполнить команду `git push origin <tagname>`. Для отправки всех тегов можно использовать команду `git push origin tags`.

Для удаления тега в локальной репозитории достаточно выполнить команду `git tag -d <tagname>`. Например, удалить созданный ранее лёгкий тег можно следующим образом: `git tag -d v1.4-lw`

Для удаления тега из внешнего репозитория используется команда `git push origin --delete <tagname>`.

Если вы хотите получить версии файлов, на которые указывает тег, то вы можете сделать `git checkout` для тега пример: `git checkout -b version2 v2.0.0`.

13. Самостоятельно изучите назначение флага `--prune` в командах `git fetch` и `git push`. Каково назначение этого флага?

`Git fetch --prune` команда получения всех изменений с репозитория GitHub.

В команде `git push --prune` удаляет удалённые ветки, у которых нет локального аналога.

Вывод: исследовал базовые возможности системы контроля версий `git` для работы с локальными репозиториями. Также, благодаря созданию тегов и

пункту 7 лабораторной работы после изменения файлов освоил возможность отката к заданной версии.