

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 ПРАКТИЧЕСКОЙ РАБОТЕ №3**  
**дисциплины «Программирование на Python»**

Выполнил:  
Касимов Асхаб Арсенович  
2 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Информатика и вычислительная  
техника», очная форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Роман Александрович

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

Цель: исследование базовых возможностей по работе с локальными и удаленными ветками Git.

Порядок выполнения работы:

1. Создал репозиторий

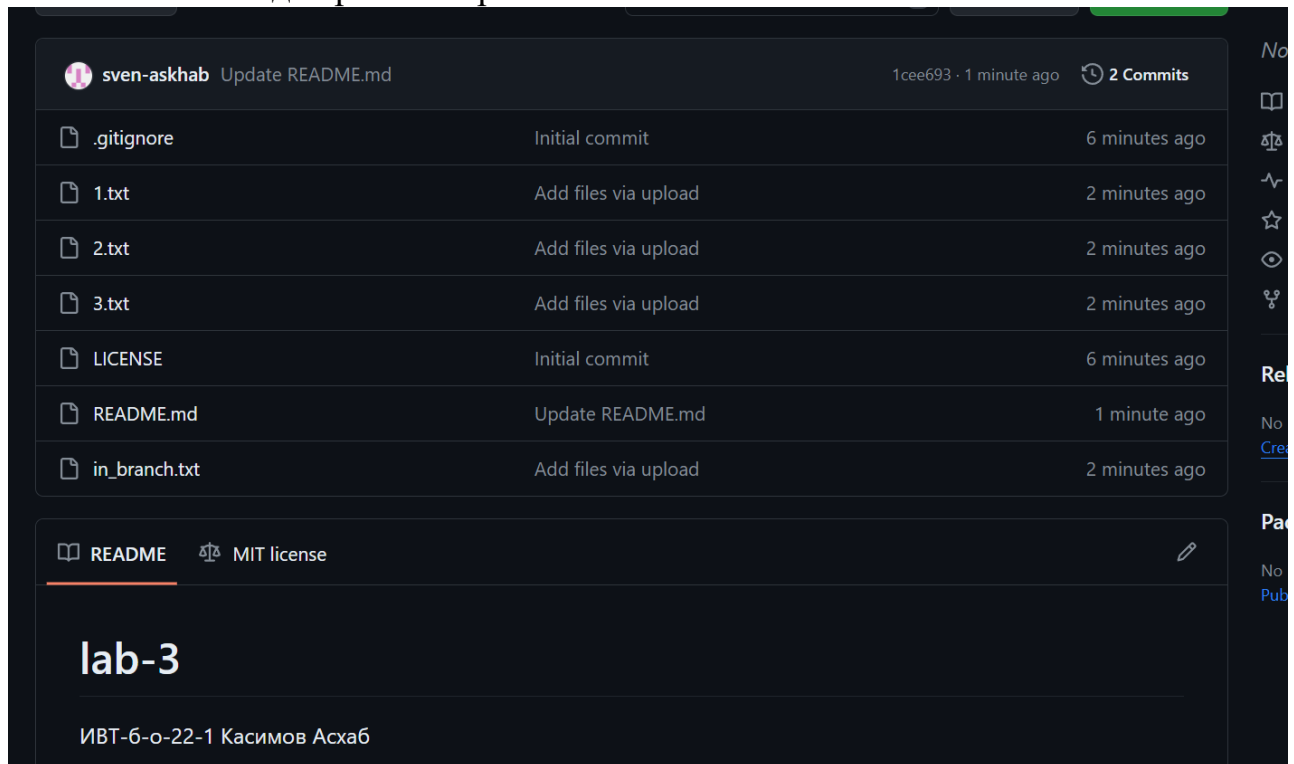


Рисунок 1. Созданный репозиторий

2. Копировал репозиторий

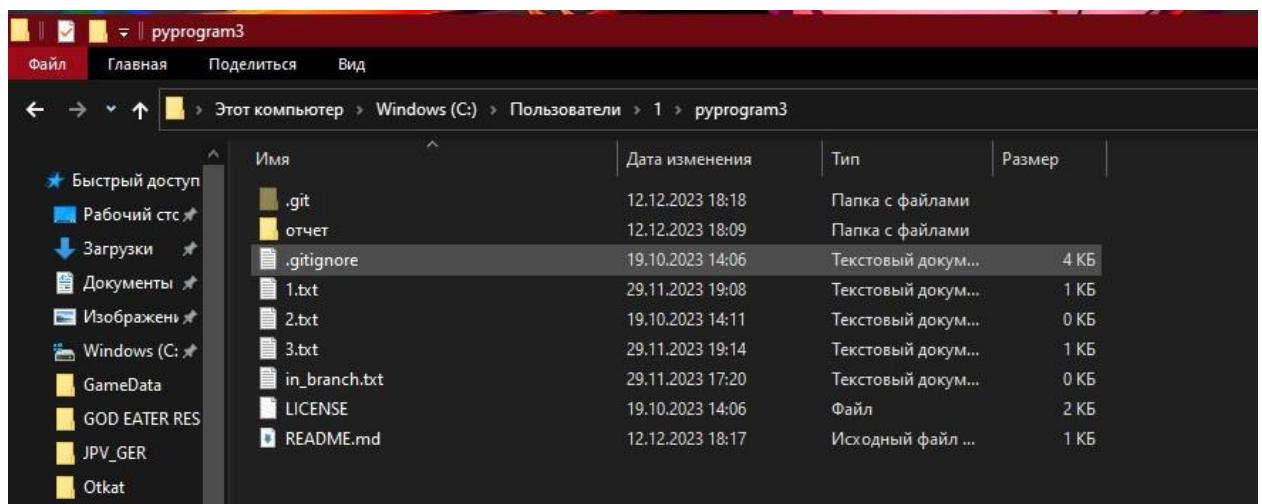


Рисунок 2. Размещение репозитория на компьютер

3. Изменил файл .gitignore

```
158 # and can be added to the global gitignore or merged into this file. For a more nuclear
159 # option (not recommended) you can uncomment the following to ignore the entire idea folder.
160 #.idea/
161 **/.DS_Store
162 .vscode|
```

Use `Control + Shift + m` to toggle the `tab` key moving focus. Alternatively, use `esc` then `tab` to move to the next interaction

Рисунок 3. Измененный файл .gitignore

#### 4. Клонировал репозиторий

```
1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2 (main)
$ git clone https://github.com/schacon/simplegit-progit
Cloning into 'simplegit-progit'...
remote: Enumerating objects: 13, done.
remote: Total 13 (delta 0), reused 0 (delta 0), pack-reused 13
Receiving objects: 100% (13/13), done.
Resolving deltas: 100% (3/3), done.
```

Рисунок 4. Клонированный репозиторий в Git

#### 5. Выполнение команд




 1.txt	1.txt changed
 2.txt	add 2.txt file
 3.txt	add 3.txt file

Рисунок 5. Создал 3 файла и закоммитил их

```
1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram3 (main)
$ git branch
* main
  my_first_branch
```

Рисунок 6. Создал новую ветку

 in_branch.txt	first_branch
---	--------------

Рисунок 7. Перешел на новую ветку и создал файл

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram3 (main)
$ git checkout my_first_branch
Switched to branch 'my_first_branch'

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram3 (my_first_branch)
$ touch in_branch.txt

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram3 (my_first_branch)
$ git add .

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram3 (my_first_branch)
$ git commit -m "first_branch"
[my_first_branch 93b2c2e] first_branch
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 in_branch.txt

```

Рисунок 8. Слил ветки

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram3 (main)
$ git branch -d my_first_branch
Deleted branch my_first_branch (was 93b2c2e).

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram3 (main)
$ git branch -d new_branch
Deleted branch new_branch (was d9d8f2c).

```

Рисунок 9. Удалил ветки my\_first\_branch и new\_branch

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram3 (main)
$ git branch branch_1

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram3 (main)
$ git branch branch_2

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram3 (main)
$ git branch
  branch_1
  branch_2
* main

```

Рисунок 10. Создал ветки branch\_1 и branch\_2

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)

```

Рисунок 11. Выполнение команды git remote -v

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git remote
origin

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git remote add pb https://github.com/paulboone/ticgit

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
pb      https://github.com/paulboone/ticgit (fetch)
pb      https://github.com/paulboone/ticgit (push)

```

Рисунок 12. Выполнение команд git remote; git remote add pb https://github.com/paulboone/ticgit; git remote -v

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git fetch pb
remote: Enumerating objects: 43, done.
remote: Counting objects: 100% (22/22), done.
remote: Total 43 (delta 22), reused 22 (delta 22), pack-reused 21
Unpacking objects: 100% (43/43), 5.99 KiB | 31.00 KiB/s, done.
From https://github.com/paulboone/ticgit
* [new branch]      master      -> pb/master
* [new branch]      ticgit      -> pb/ticgit

```

Рисунок 13. Выполнение команды git fetch pb

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git remote show origin

* remote origin
Fetch URL: https://github.com/schacon/ticgit
Push URL: https://github.com/schacon/ticgit
HEAD branch: master
Remote branches:
  master tracked
  ticgit tracked
Local branch configured for 'git pull':
  master merges with remote master
Local ref configured for 'git push':
  master pushes to master (up to date)

```

Рисунок 14. Выполнение команды git remote show origin

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git pull
Already up to date.

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git remote rename pb paul
Renaming remote references: 100% (2/2), done.

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git remote
origin
paul

```

Рисунок 15. Выполнение команд git remote rename pb paul; git pull; git remote

```
1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git remote remove paul

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git remote
origin
```

Рисунок 16. Выполнение команд git remote remove paul; git remote

```
1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git tag

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git tag -l "v1.8.5*"

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git tag -a v1.4 -m "my version 1.4"

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git tag
v1.4
```

Рисунок 17. Создание тега

```
1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git show v1.4
tag v1.4
Tagger: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Wed Oct 11 18:58:59 2023 +0300

my version 1.4

commit 847256809a3d516cd3db8f81859401110fe8d945 (HEAD -> master, tag: v1.4, origin/master, origin/HEAD)
Author: Jeff Welling <Jeff.Welling@gmail.com>
Date: Tue Apr 26 17:29:17 2011 -0700

    Added note to clarify which is the canonical TicGit-ng repo

diff --git a/README.mkd b/README.mkd
index ab92035..9ea9ff9 100644
--- a/README.mkd
+++ b/README.mkd
@@ -1,3 +1,6 @@
-Note: the original TicGit author has pulled all the TicGit-ng changes into his repository, creating a potentially confusing situation. The schacon TicGit repo, this one, is not consistently m
aintained. For up to date TicGit-ng info and code, check the canonical TicGit-ng repository at https://github.com/jeffwelling/ticgit
+
+## TicGit-ng ##
+
+This project provides a ticketing system built on Git that is kept in a
```

Рисунок 18. Выполнение команды git show v1.4

```
1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2/ticgit (master)
$ git tag -d v1.4
Deleted tag 'v1.4' (was 8ff2d5b)
```

Рисунок 19. Удаление тега

6. Создал не менее 7 коммитов и 4 тегов

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2 (main)
$ git log --graph --pretty=oneline --abbrev-commit
* 85dee31 (HEAD -> main, tag: v1.3, origin/main, origin/HEAD) FinalProgr
* 60c4df7 NewCommit6
* 6550469 (tag: v1.2) NewCommit5
* ad13295 NewCommit4
* 422018e (tag: v1.1) NewCommit3
* 9180323 NewCommit2
* ef5070c (tag: v1.0) New Commit1
* 86ca520 New Commit1
* de0c4eb Initial commit

```

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2 (main)
$ git log
commit 85dee3104ae5a14efeb7385098d05dee0b94f46e (HEAD -> main, tag: v1.3, origin/main, origin/HEAD)
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Thu Oct 12 13:59:13 2023 +0300

    FinalProgr

commit 60c4df798f575d755460faa318e091a0ae840f65
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Thu Oct 12 13:56:28 2023 +0300

    NewCommit6

commit 6550469ed20bb4c80488ce2035c2775f4338dae5 (tag: v1.2)
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Thu Oct 12 13:53:24 2023 +0300

    NewCommit5

commit ad13295d9ad601116382837a9df731e2a6a19a4a
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Thu Oct 12 13:51:59 2023 +0300

    NewCommit4

commit 422018e5121643a966843c502a7d5cb8141f16d6 (tag: v1.1)
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Thu Oct 12 13:46:15 2023 +0300

    NewCommit3

commit 9180323dc926793c4a524ca364daaef561916323
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Thu Oct 12 13:42:01 2023 +0300

    NewCommit2

commit ef5070cfc1d125744b4eb75d7af816d36ec034cb (tag: v1.0)
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Wed Oct 11 21:36:21 2023 +0300

    New Commit1

commit 86ca520bf6461c2de9445ba28867eb96c5b93730
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Wed Oct 11 21:33:28 2023 +0300

    New Commit1

```

Рисунок 20. Все созданные коммиты и теги

7. Просмотрел содержимое коммитов командой git show

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2 (main)
$ git show HEAD
commit 85dee3104ae5a14efeb7385098d05dee0b94f46e (HEAD -> main, tag: v1.3, origin/main, origin/HEAD)
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Thu Oct 12 13:59:13 2023 +0300

    FinalProgr

diff --git a/proga/main.py b/proga/main.py
index 42b68fd..4288439 100644
--- a/proga/main.py
+++ b/proga/main.py
@@ -3,4 +3,5 @@ chisl = int(input("Введите число: "))
count = 0
while chisl > 0:
    count = count + 1
-    var_1 = var_1 // 10
\ No newline at end of file
+    chisl = chisl // 10
+print("Количество цифр равно:", count)
\ No newline at end of file

```

Рисунок 21. Просмотр последнего коммита

```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2 (main)
$ git show HEAD~1
commit 60c4df798f575d755460faa318e091a0ae840f65
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Thu Oct 12 13:56:28 2023 +0300

    NewCommit6

diff --git a/proga/main.py b/proga/main.py
index 62d63a9..42b68fd 100644
--- a/proga/main.py
+++ b/proga/main.py
@@ -2,4 +2,5 @@ print("<Количество цифр в числе>")
chisl = int(input("Введите число: "))
count = 0
while chisl > 0:
-    count = count + 1
\ No newline at end of file
+    count = count + 1
+    var_1 = var_1 // 10
\ No newline at end of file

```

Рисунок 22. Просмотр предпоследнего коммита



```

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram2 (main)
$ git show 9180323dc926793c4a524ca364daaef561916323
commit 9180323dc926793c4a524ca364daaef561916323
Author: Akse1Sukub <akse1.sukubov@yandex.ru>
Date: Thu Oct 12 13:42:01 2023 +0300

    NewCommit2

diff --git a/proga/.idea/.name b/proga/.idea/.name
new file mode 100644
index 0000000..11a5d8e
--- /dev/null
+++ b/proga/.idea/.name
@@ -0,0 +1 @@
+main.py
\ No newline at end of file
diff --git a/proga/main.py b/proga/main.py
index e16def4..8451eca 100644
--- a/proga/main.py
+++ b/proga/main.py
@@ -1,2 @@
-print("<Количество цифр в числе>")
\ No newline at end of file
+print("<Количество цифр в числе>")
+chisl = int(input("Введите число: "))
\ No newline at end of file

```

Рисунок 23. Просмотр коммита по ссылке

8. Освоил возможность отката к заданной версии

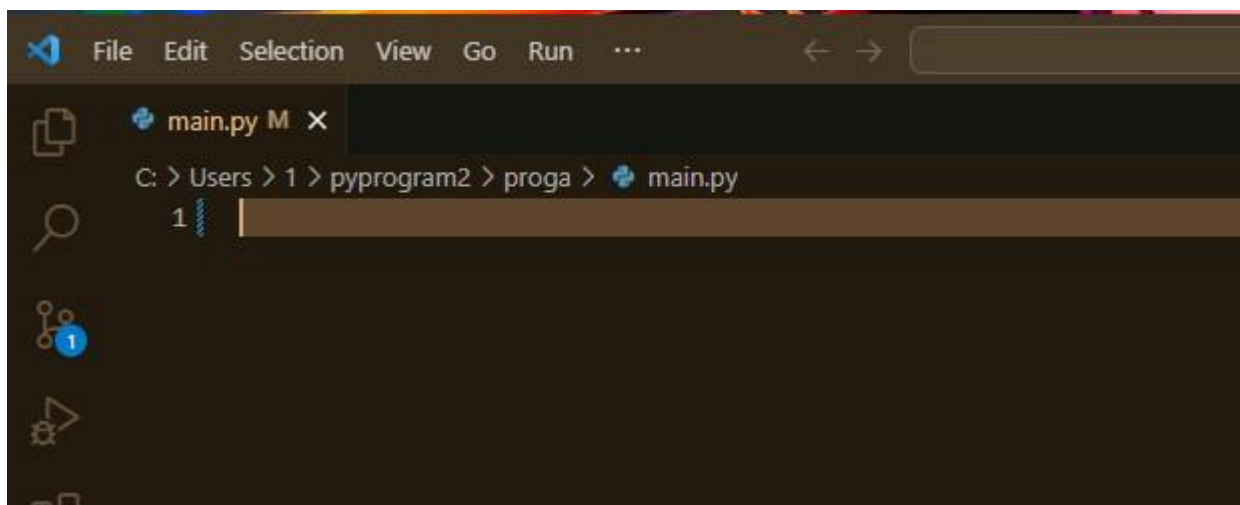
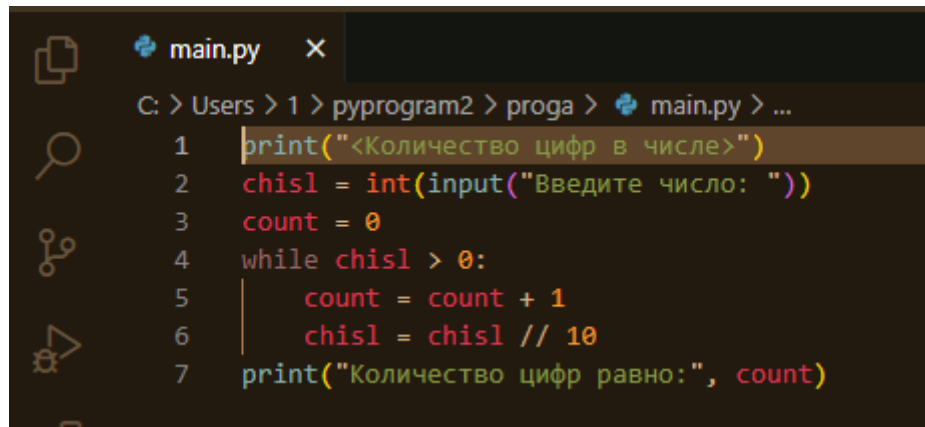


Рисунок 24. Удаление программы

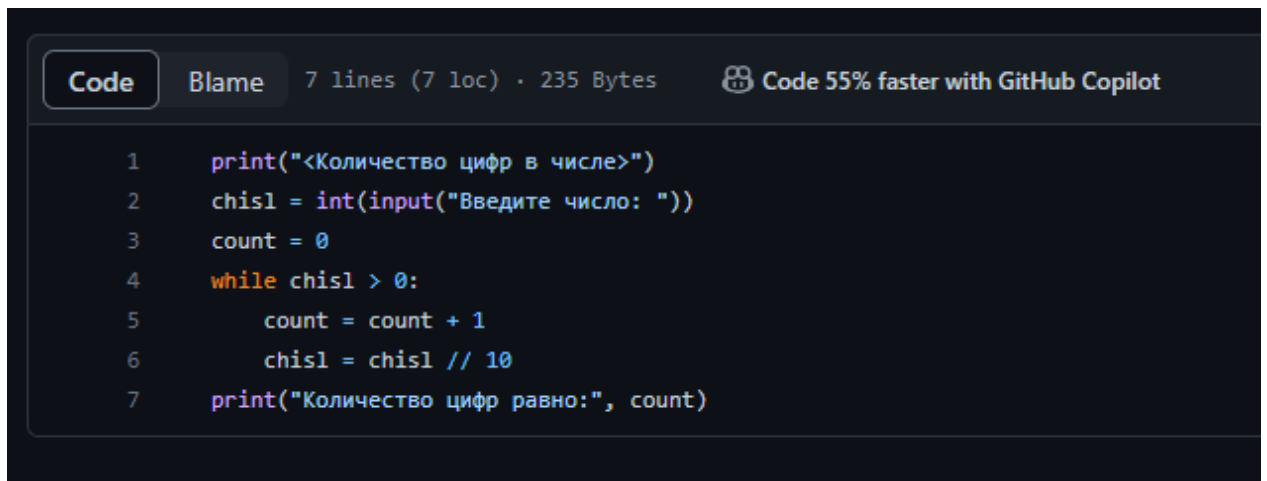


```
main.py x
C: > Users > 1 > pyprogram2 > proga > main.py > ...
1 print("<Количество цифр в числе>")
2 chisl = int(input("Введите число: "))
3 count = 0
4 while chisl > 0:
5     count = count + 1
6     chisl = chisl // 10
7 print("Количество цифр равно:", count)
```

Рисунок 25. Вернул программу с помощью команды `git checkout -- main.py`

Рисунок 26. Сделал коммит с удаленной программой

Рисунок 27. Вернул программу с помощью команды `git reset --hard HEAD~1`



```
Code Blame 7 lines (7 loc) · 235 Bytes Code 55% faster with GitHub Copilot
1 print("<Количество цифр в числе>")
2 chisl = int(input("Введите число: "))
3 count = 0
4 while chisl > 0:
5     count = count + 1
6     chisl = chisl // 10
7 print("Количество цифр равно:", count)
```

## Ответы на контрольные вопросы:

1. Как выполнить историю коммитов в Git? Какие существуют дополнительные опции для просмотра истории коммитов? После того, как вы создали несколько коммитов или же клонировали репозиторий с уже существующей историей коммитов, вероятно Вам понадобится возможность посмотреть, что было сделано — историю коммитов. Одним из основных и наиболее мощных инструментов для этого является команда `git log`. Команда `git log` имеет очень большое количество опций для поиска коммитов по разным критериям. Рассмотрим наиболее популярные из них. Одним из самых полезных аргументов является `-p` или `--patch`, который показывает разницу (выводит патч), внесенную в каждый коммит. Если вы хотите увидеть сокращенную статистику для каждого коммита, вы можете использовать опцию `--stat`. Следующей действительно полезной опцией является `--pretty`. Эта опция меняет формат вывода. Существует несколько встроенных вариантов отображения. Опция `oneline` выводит каждый коммит в одну строку, что может быть очень удобным 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.

2. Как ограничить вывод при просмотре истории коммитов? В дополнение к опциям форматирования вывода, команда `git log` принимает несколько опций для ограничения вывода — опций, с помощью которых можно увидеть определенное подмножество коммитов. Одна из таких опций — это опция `-n`, которая показывает только последние два коммита. В действительности вы можете использовать `-n`, где `n` — это любое натуральное число и представляет собой `n` последних коммитов. На практике вы не будете часто использовать эту опцию, потому что Git по умолчанию использует постраничный вывод, и вы будете видеть только одну страницу за раз. Опции для ограничения вывода по времени, такие как `--since` и `--until`, являются

очень удобными. Опция `--author` дает возможность фильтровать по автору коммита, а опция `--grep` искать по ключевым словам в сообщении коммита. Следующим действительно полезным фильтром является опция `-S`, которая принимает аргумент в виде строки и показывает только те коммиты, в которых изменение в коде повлекло за собой добавление или удаление этой строки. Последней полезной опцией, которую принимает команда `git log` как фильтр, является путь. Если вы укажете каталог или имя файла, вы ограничите вывод только теми коммитами, в которых были изменения этих файлов. Эта опция всегда указывается последней после двойного тире ( `--` ), чтобы отделить пути от опций

3. Как внести изменения в уже сделанный коммит? Отмена может потребоваться, если вы сделали коммит слишком рано, например, забыв добавить какие-то файлы или комментарий к коммиту. Если вы хотите переделать коммит — внесите необходимые изменения, добавьте их в индекс и сделайте коммит ещё раз, указав параметр `--amend`.

4. Как отменить индексацию файла в Git? Использовать `git reset HEAD` ... для исключения из индекса.

5. Как отменить изменения в файле? Использовать `git checkout --` для возвращения к версии из последнего коммита.

6. Что такое удаленный репозиторий Git? Удалённые репозитории представляют собой версии вашего проекта, сохранённые в интернете или ещё где-то в сети.

7. Как выполнить просмотр удаленных репозиториях данного локального репозитория? Для того, чтобы просмотреть список настроенных удалённых репозиториях, вы можете запустить команду `git remote`. Она выведет названия доступных удалённых репозиториях. Если вы клонировали репозиторий, то увидите как минимум `origin` — имя по умолчанию, которое Git даёт серверу, с которого производилось клонирование.

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` .

9. Как выполнить отправку/получение изменений с удаленного репозитория? Для получения данных из удалённых проектов, следует выполнить `git fetch [remote-name]`. Когда вы хотите поделиться своими наработками, вам необходимо отправить их в удалённый репозиторий. Команда для этого действия простая: `git push` .

10. Как выполнить просмотр удаленного репозитория? Если хотите получить побольше информации об одном из удалённых репозиториях, вы можете использовать команду `git remote show` . Она выдаёт URL удалённого репозитория, а также информацию об отслеживаемых ветках.

11. Каково назначение тэгов Git? Как и большинство СКВ, Git имеет возможность помечать определённые моменты в истории как важные. Как правило, эта функциональность используется для отметки моментов выпуска версий (v1.0, и т. п.). Такие пометки в Git называются тегами.

12. Как осуществляется работа с тэгами Git? Просмотреть список имеющихся тегов в Git можно очень просто. Достаточно набрать команду `git tag` (параметры `-l` и `--list` опциональны). Создание аннотированного тега в Git выполняется легко. Самый простой способ — это указать - а при выполнении команды `tag`. 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` . Для удаления тега в локальном репозитории достаточно выполнить команду `git tag -d` . Если вы хотите получить версии файлов, на которые указывает тег, то вы можете сделать `git checkout` для тега. Однако, это переведёт репозиторий в состояние «detached HEAD», которое имеет ряд неприятных побочных эффектов.

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

предоставленного `git help fetch`: `--prune` используется для удаления ссылок удаленного отслеживания, которые больше не существуют в удаленном репозитории, а из описания, предоставленного `git help push`: `--prune` используется для удаления ветвей на удаленном репозитории, для которых нет аналога в локальном репозитории. Вывод: в результате выполнения работы были исследованы возможности Git для работы с локальными репозиториями.