

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

Институт перспективной инженерии  
Департамент цифровых, робототехнических систем и электроники

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**  
**дисциплины**  
**«Основы кроссплатформенного программирования»**  
**Вариант**

Выполнил:  
Кравчук Мирослав Витальевич  
2 курс, группа ИТС-б-о-23-1,  
11.03.02 «Инфокоммуникационные  
технологии и системы связи», очная  
форма обучения

---

(подпись)

Проверил:  
Ассистент департамента цифровых,  
робототехнических систем и  
электроники Хацукова А.И

---

(подпись)

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

Ставрополь, 2024 г.

## **Тема: Исследование возможностей Git для работы с локальными репозиториями**

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

### **Конспект теоретического материала:**

Полезные опции для `git log --pretty=format` отображает наиболее полезные опции для

изменения формата.

Опция

Описания вывода

%H

Хеш коммита

%h

Сокращенный хеш коммита

%T

Хеш дерева

%t

Сокращенный хеш дерева

%P

Хеш родителей

%p

Сокращенный хеш родителей

%an

Имя автора

%ae

Электронная почта автора

%ad

Дата автора (формат даты можно задать опцией `--date=option`)

%ar

Относительная дата автора

%cn

Имя коммитера

%ce

Электронная почта коммитера

%cd

Дата коммитера

%cr

Относительная дата коммитера

%s

Содержание

Вам, наверное, интересно, какая же разница между автором и коммитером. Автор — это человек, изначально сделавший работу, а коммитер — это человек, который последним применил эту работу. Другими словами, если вы создадите патч для какого-то проекта, а один из основных членов команды этого проекта применит этот патч, вы оба получите статус участника — вы как автор и основной член команды как коммитер.

Опции `oneline` и `format` являются особенно полезными с опцией `--graph` команды `log`. С этой опцией вы сможете увидеть небольшой граф в формате ASCII, который показывает текущую ветку и историю слияний.

Для того, чтобы внести вклад в какой-либо Git-проект, вам необходимо уметь работать с удалёнными репозиториями. Удалённые репозитории представляют собой версии вашего проекта, сохранённые в интернете или ещё где-то в сети. У вас может быть несколько удалённых репозиториях, каждый из которых может быть доступен для чтения или для чтения-записи.

Взаимодействие с другими пользователями предполагает управление удалёнными репозиториями, а также отправку и получение данных из них. Управление репозиториями включает в себя как умение добавлять новые, так и умение удалять устаревшие репозитории, а также умение управлять различными удалёнными ветками, объявлять их отслеживаемыми или

нет и так далее. В данном разделе мы рассмотрим некоторые из этих навыков.

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

Как и большинство СКВ, Git имеет возможность пометать 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. В этом разделе вы узнаете, как посмотреть имеющиеся теги, как создать новые или удалить существующие, а также какие типы тегов существуют в Git.

Git использует два основных типа тегов: легковесные и аннотированные.

Легковесный тег — это что-то очень похожее на ветку, которая не изменяется — просто указатель на определённый коммит.

А вот аннотированные теги хранятся в базе данных Git как полноценные объекты. Они имеют контрольную сумму, содержат имя автора, его e-mail и дату создания, имеют комментарий и могут быть подписаны и проверены с помощью GNU Privacy Guard (GPG). Обычно рекомендуется создавать аннотированные теги, чтобы иметь всю перечисленную информацию; но если вы хотите сделать временную метку или по какой-то причине не хотите сохранять остальную информацию, то для этого годятся и легковесные.

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.

## Порядок выполнения работы:


1. Изучил теоретический материал.
2. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и выбранный язык программирования.

### Создайте новый репозиторий

Репозиторий содержит все файлы проекта, включая историю изменений. У вас уже есть репозиторий проекта в другом месте? [Импортируйте репозиторий.](#)

Обязательные поля отмечены звездочкой (\*).

Владелец \*

 miron2314 ▾

Название репозитория \*

/ demo 2

✓ Ваш новый репозиторий будет создан как demo-2.

Имя репозитория может содержать только буквы ASCII, цифры и символы -, \_ и .

Отличные названия репозитория короткие и запоминающиеся. Нужно вдохновение? Как насчет **переработанного мема?**

Описание (необязательно)

Исследование возможностей Git для работы с локальными репозиториями



Общедоступный

Этот репозиторий может видеть любой пользователь Интернета. Вы сами выбираете, кто может совершать коммиты.



Приватный

Вы сами выбираете, кто может просматривать этот репозиторий и фиксировать его в нем.

Инициализируйте этот репозиторий с помощью:



Добавьте файл README

Здесь вы можете написать подробное описание вашего проекта. [Узнайте больше о README.](#)

Добавить файл .gitignore


.шаблон gitignore: Отсутствует ▾

Выберите файлы, которые не следует отслеживать, из списка шаблонов. [Узнайте больше об игнорировании файлов.](#)

Выберите лицензию

Лицензия: Отсутствует ▾

Лицензия сообщает другим пользователям, что они могут и чего не могут делать с вашим кодом. [Узнайте больше о лицензиях.](#)

Это позволит установить  main в качестве ветки по умолчанию. Измените имя по умолчанию в ваших [настройках](#).



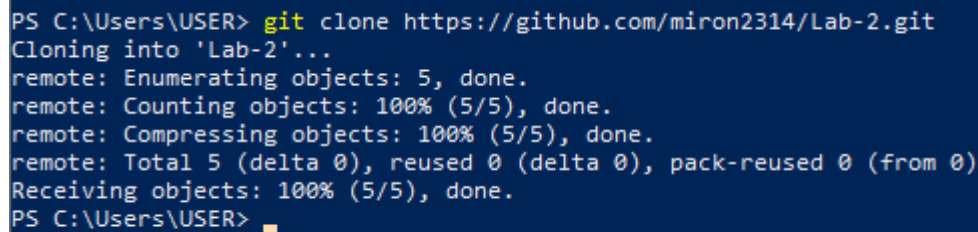
Вы создаете публичный репозиторий в своем личном кабинете.

Создать репозиторий

Рисунок 1. Создание репозитория

3.Проработал примеры лабораторной работы. Отразил вывод на консоли при выполнении команд git в отчете для лабораторной работы.

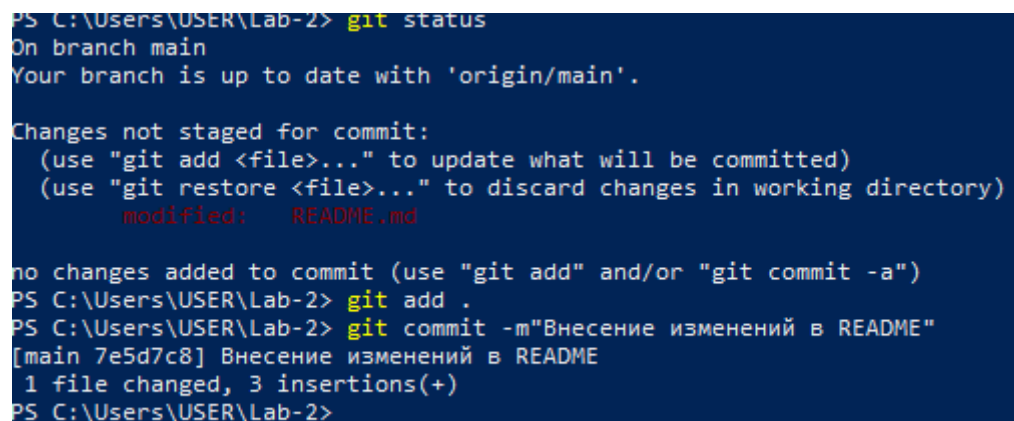
4.Выполнил клонирование созданного репозитория на рабочий компьютер.



```
PS C:\Users\USER> git clone https://github.com/miron2314/Lab-2.git
Cloning into 'Lab-2'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (5/5), done.
PS C:\Users\USER>
```

Рисунок 2. Клонирование репозитория

5. Добавил в файл README.md информацию о дисциплине, группе и ФИО студента, выполняющего лабораторную работу.



```
PS C:\Users\USER\Lab-2> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\USER\Lab-2> git add .
PS C:\Users\USER\Lab-2> git commit -m"Внесение изменений в README"
[main 7e5d7c8] Внесение изменений в README
 1 file changed, 3 insertions(+)
PS C:\Users\USER\Lab-2>
```

Рисунок 3. Добавление информации в файл

6.Написал небольшую программу на выбранном языке программирования. Фиксировал изменения при написании программы в локальном репозитории. Сделано не менее 7 коммитов, отмеченных не менее 3 тэгами.

```

PS C:\Users\USER\Lab-2> git tag -a v1.1 -m "добавление тегов"
PS C:\Users\USER\Lab-2> git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 415 bytes | 415.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/miron2314/Lab-2.git
    f1f1abd..a59a5b3  main -> main
PS C:\Users\USER\Lab-2> git push origin v1.1
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 190 bytes | 190.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/miron2314/Lab-2.git
    * [new tag]          v1.1 -> v1.1
PS C:\Users\USER\Lab-2>

```

Рисунок 4. Добавление тегов

7. Просмотрел историю хранилища командой git log. Добавил скриншот консоли с выводом в отчет по лабораторной работе.

```

Initial commit
PS C:\Users\USER\Lab-2> git log --graph --pretty=oneline --abbrev-commit
* cbe28b8 (HEAD -> main, origin/main, origin/HEAD) добавление счетчика
* a07ece2 (tag: v1.3) рандомайзер
* e0ebee4 (tag: v1.2) добавление програм
* a59a5b3 (tag: v1.1) добавление списка
* f1f1abd добавление программы
* 7e5d7c8 Внесение изменений в README
* 342bcc8 Initial commit
PS C:\Users\USER\Lab-2>

```

Рисунок 5. Использование команды git log

8.Просмотрел содержимое коммитов командой git show. Отобразил результаты работы команд в отчете.

```

PS C:\Users\USER\Lab-2> git show head
commit cbe28b8c0a00162b247b3166b0f0785cceb1ec4d (HEAD -> main, origin/main, origin/HEAD)
Author: miron2314 <nikolaidichenko@gmail.com>
Date: Thu Sep 26 23:16:58 2024 +0300

    добавление счетчика

diff --git "a/\321\201\321\207\320\265\321\202\321\207\320\270\320\272.py" "b/\321\201\321\207\320\265\321\202\321\207\320\270\320\272.py"
new file mode 100644
index 0000000..3709a33
--- /dev/null
+++ "b/\321\201\321\207\320\265\321\202\321\207\320\270\320\272.py"
@@ -0,0 +1,8 @@
+def word_counter():
+    text = input("Enter some text: ")
+    words = text.split()
+    count = len(words)
+
+    print("Word count:", count)
+
+word_counter()

```

Рисунок 6. Использование команды git show head

```

PS C:\Users\USER\Lab-2> git show head~1
commit a07ece2cc888a9ad3eee6a66eab7fee5b3b350f3 (tag: v1.3)
Author: miron2314 <nikolaidichenko@gmail.com>
Date: Thu Sep 26 22:38:34 2024 +0300

    рандомайзер

diff --git a/main2.py b/main2.py
index 079d257..ac2fc90 100644
--- a/main2.py
+++ b/main2.py
@@ -1,5 +1,15 @@
-def get_vowels(String):
-    return [each for each in String if each in "aeiou"]
+get_vowels("animal") # [a, i, a]
+get_vowels("sky") # []
+get_vowels("Football") # [o, o, a]
\ No newline at end of file
+import random
+
+def dice_simulator():
+    num_dice = int(input("Enter the number of dice: "))
+    rolls = []
+
+    for _ in range(num_dice):
+        roll = random.randint(1, 6)
+        rolls.append(roll)
+
+    print("Rolls:", rolls)
+
+dice_simulator()
diff --git a/time.py b/time.py
new file mode 100644
index 0000000..bf73934
--- /dev/null
+++ b/time.py
@@ -0,0 +1,4 @@
+str1 = "Python Programming"
+str2 = "IM A PROGRAMMER"
+print(str1.upper()) #PYTHON PROGRAMMING
+print(str2.lower()) #im a programmer
\ No newline at end of file

```

Рисунок 7. Использование команды get show head~1



```

PS C:\Users\USER\Lab-2> git show head~2
commit e0abee4334d58a0f3b8b164fdc9ff741ebe83140 (tag: v1.2)
Author: miron2314 <nikolaidichenko@gmail.com>
Date: Thu Sep 26 21:59:47 2024 +0300

    добавление програм

diff --git a/main2.py b/main2.py
new file mode 100644
index 0000000..079d257
--- /dev/null
+++ b/main2.py
@@ -0,0 +1,5 @@
+def get_vowels(String):
+    return [each for each in String if each in "aeiou"]
+get_vowels("animal") # [a, i, a]
+get_vowels("sky") # []
+get_vowels("football") # [o, o, a]
\ No newline at end of file
diff --git a/main3.py b/main3.py
new file mode 100644
index 0000000..9be69ac
--- /dev/null
+++ b/main3.py
@@ -0,0 +1,3 @@
+n=5
+string="Hello World "
+print(string * n)
\ No newline at end of file

```

Рисунок 8. Использование команды git show head~2

```

PS C:\Users\USER\Lab-2> git show cbe28b8
commit cbe28b8c0a00162b247b3166b0f0785cceb1ec4d (HEAD -> main, origin/main, origin/HEAD)
Author: miron2314 <nikolaidichenko@gmail.com>
Date: Thu Sep 26 23:16:58 2024 +0300

    добавление счетчика

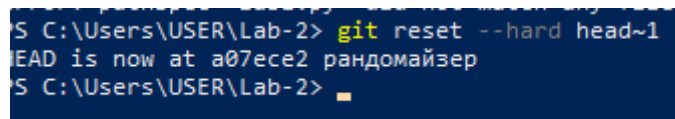
diff --git "a/\321\201\321\207\320\265\321\202\321\207\320\270\320\272.py" "b/\321\201\321\207\320\265\321\202\321\207\320\270\320\272.py"
new file mode 100644
index 0000000..3709a33
--- /dev/null
+++ "b/\321\201\321\207\320\265\321\202\321\207\320\270\320\272.py"
@@ -0,0 +1,8 @@
+def word_counter():
+    text = input("Enter some text: ")
+    words = text.split()
+    count = len(words)
+
+    print("Word count:", count)
+
+word_counter()
\ No newline at end of file

```

Рисунок 9. Использование команды git show для коммита с хэшем

9. Освоил возможность отката к заданной версии. Удалил весь код из одного из файлов программы репозитория и сохранил этот файл.

Удалил все несохраненные изменения в файле командой: git checkout сделал выводы об изменении содержимого файла программы после выполнения отката. Отразил эти выводы в отчете.



```
S C:\Users\USER\Lab-2> git reset --hard head~1
HEAD is now at a07ece2 рандомайзер
S C:\Users\USER\Lab-2>
```

Рисунок 10. Откат состояния хранилища

**Вывод:** в результате использования команды `git reset --hard` отменяется последние коммиты и сбрасывается рабочая копия до определенного состояния. `Git checkout` позволяет управлять содержимым рабочего каталога и возвращаться к прошлым состояниям файлов и веток.

10. Зафиксировал сделанные изменения.

11. Добавил отчет по лабораторной работе в формате PDF в папку `doc` репозитория. Зафиксируйте изменения.

12. Отправил изменения из локального репозитория в удаленный репозиторий GitHub.

13. Проконтролировал изменения, произошедшие в репозитории GitHub.

14. Самостоятельно изучил работу с сервисами GitLab или BitBucket. Создал репозиторий на одном из этих сервисов. Создал зеркало репозитория на GitHub с использованием созданного репозитория.