Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ О ПРАКТИЧЕСКОЙ** **РАБОТЕ № 3**

Разработка приложений с git

Тема

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Преподаватель |  |  |  |  |  | П.В.Пересунько |
|  |  |  |  | подпись, дата |  | инициалы, фамилия |
| Студент |  | КИ19-16/2б |  |  |  | С.Е.Шпаков |
|  |  | номер группы, зачетной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2019

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc25285173)

[1 Цель и задачи 3](#_Toc25285174)

[1.1 Цель 3](#_Toc25285175)

[1.2 Задачи 3](#_Toc25285176)

[2 Описание варианта задания 3](#_Toc25285177)

[3 Ход выполнения 4](#_Toc25285178)

[4 Выводы 8](#_Toc25285179)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 9](#_Toc25285180)

# 1 Цель и задачи

## 1.1 Цель

Изучить принципы работы с системой контроля версий Git. Научиться создавать репозиторий, делать фиксации, откатывать файлы к старой версии и пользоваться командой bisect.

## 1.2 Задачи

Для выполнения практической работы необходимо выполнить следующие задачи:

1. изучить основные принципы работы с git;
2. установить имя пользователя и почту в git;
3. создать репозиторий;
4. сделать несколько коммитов вместе с реализацией программы по

варианту;

1. показать логи стандартные, однострочные и однострочные с датой;
2. вернуться к последнему коммиту, отбросив все изменения после него

(как в примере из лекции № 4 с последней попыткой сделать рефакторинг);

1. перейти к старому коммиту и сделать скриншот или листинг состояния

одного из измененных файлов;

1. оформить отчет по практической работы;
2. ответить на вопросы и выполнить дополнительные задания.

# 2 Описание варианта задания

Имеется алфавит A, состоящий только из трёх букв – «O», «E», «W» и «M». Для каждого символа исходного алфавита A соответствие поставлена комбинация символов из кодового алфавита B:

1. **«**O**»** – кодируется символом «..»;
2. **«**E**»** – «кодируется комбинацией «.\_»;
3. **«**W**»** – кодируется комбинацией «\_.»;
4. **«**M**»** – «кодируется комбинацией «\_\_».

Необходимо раскодировать строку, состоящую из букв кодового алфавита.

# 3 Ход выполнения

Для выполнения задания необходимо подготовить git (в дальнейшем «гит») и репозиторий. В первую очередь, чтобы поладить с гитом, необходимо ему представится и дать свои контактные данные. Для этого необходимо использовать команду “git config –global user.\* {input}”, где на месте \* находится name или email для соответствующей части представления (рисунок 1).

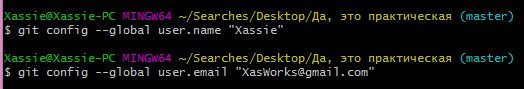


Рисунок 1 – Настройка имени пользователя и почты в git

Чтобы начать работу, необходимо создать или инициализировать репозиторий. Делается это простой командой “git init” (рисунок 2), при запуске git bash в необходимом месте или перемещении туда вручную как в командой строке.

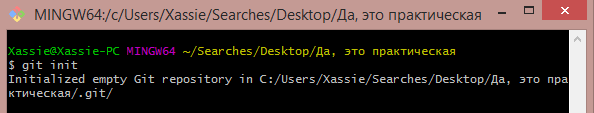


Рисунок 2 – Создание (инициализация) репозитория

Теперь можно приступить к реализации программы. Алфавиты А и В, для удобства, помещены в словарь, где каждый элемент В соответствует элементу А, а сам словарь помещен в константу. В основной функции мы первым делом показываем нашу вежливость и приветствуем мир. Дальше нам необходим ввод зашифрованного сообщения, который реализуется через интерфейс командной строки. Дальше мы ищем зашифрованные буквы и если в конце концов н остается нерасшифрованных символов в веденной строке, то выводим ответ. Проверку ввода можно опустить, так как неизвестные символы так же, как и неверно зашифрованная строка, приведут к одной ошибке «Impossible to decode». Таким образом мы получили программу, реализованную по варианту задания (листинг 1).

Листинг 1 – Декодер, реализованный по заданию

import argparse

ALPHABET = {

'..': 'O',

'.\_': 'E',

'\_.': 'W',

'\_\_': 'M',

}

def main():

print('Hello world!')

parser = argparse.ArgumentParser(description="Give a line to decode")

parser.add\_argument('n', type=str, help='Coded message')

R = parser.parse\_args()

msg = R.n

dc = ''

res = ''

for i in msg:

dc += i

if dc in ALPHABET:

res += ALPHABET[dc]

dc = ''

if not dc:

print(res)

else:

print('Impossible to decode')

if \_\_name\_\_ == '\_\_main\_\_':

main()

После создания файла, в процессе реализации программы, а так же после того, как программа была закончена мы делаем коммиты. Для этого мы показываем какой файл нужно закоммитить “git add {filename}” и непосредственно коммитим, добавляя комментарий “git commit –m {comment}” (рисунок 3). Если для комментария недостаточно одной строки, то “-m {comment}” можно опустить, тогда откроется редактор, где мы и напишем комментарий.

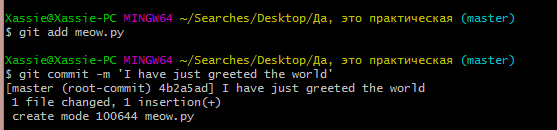


Рисунок 3 – Коммит в гите

В какой-то момент нам станет интересно что именно мы закоммитили. Это можно посмотреть в логах с помощью команды “git log”. Если мы хотим вывод оформить, чтобы он выглядел проще и удобнее, то можно использовать опциональный аргумент “—pretty”. Ему можно придать значение “oneline” (название говорит за себя). Можно оформить самому с помощью “format”. Возможных аргументов много, но здесь будут использоваться дата, имя, хэш и комментарий - %ad, %cn, %H, %s соответственно (рисунок 4).

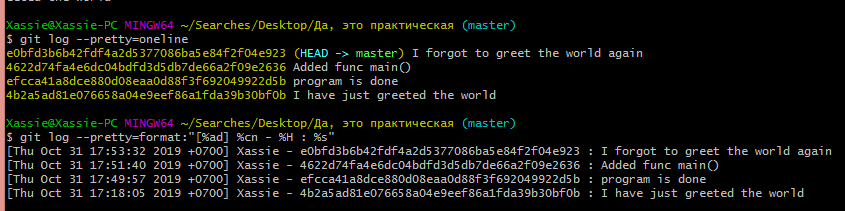


Рисунок 4 – Логи гита

Все мы можем напортачить. Если файл перестал работать, то можно откатить изменения с помощью команды “git checkout” (рисунок 5). Если после этого добавить хэш одного из предыдущих коммитов, то файл примет вид файла в тот коммит.

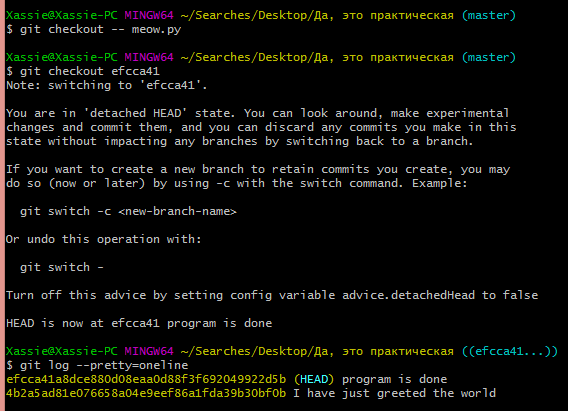


Рисунок 5 – Откат файла с помощью checkout

В текущем коммите файл выглядел похоже на финальный, но всё же немного отличался (листинг 2).

Листинг 2 – Программа в одном из ранних коммитов

import argparse

ALPHABET = {

'..' : 'O',

'.\_' : 'E',

'\_.' : 'W',

'\_\_' : 'M',

}

parser = argparse.ArgumentParser(description="Give a line to decode")

parser.add\_argument('n', type=str, help='Coded message')

R = parser.parse\_args()

msg = R.n

dc = ''

res = ''

for i in msg:

dc += i

if dc in ALPHABET:

res += ALPHABET[dc]

dc = ''

if not dc:

print(res)

else:

print('Impossible to decode')

# 4 Выводы

В результате работы были рассмотрены основы обращения с гитом, разобраны его основные функции, реализована программа дешифратор, а также был поприветствован мир.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. СТО 4.2-07-2014 Система менеджмента качества Общие требования к построению, изложению и оформлению документов учебной деятельности. – Взамен СТО 4.2–07–2012 ; введ. 09.01.2014, - Красноярск : ИПК СФУ, 2014 – 60 с.