

# **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

**Markdown**

Анастасия Павловна Баранова, НБИбд-01-21

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>4</b>	<b>Контрольные вопросы</b>	<b>11</b>
<b>5</b>	<b>Выводы</b>	<b>19</b>
	<b>Список литературы</b>	<b>20</b>

## Список иллюстраций

3.1	Создание учётной записи на github. . . . .	6
3.2	Установка git-flow. . . . .	6
3.3	Установка gh. . . . .	6
3.4	Базовая настройка git. . . . .	7
3.5	Базовая настройка git. . . . .	7
3.6	Базовая настройка git. . . . .	7
3.7	Создание ключа ssh. . . . .	7
3.8	Создание ключа ssh. . . . .	8
3.9	Создание ключа gpg. . . . .	8
3.10	Добавление GPG ключа в GitHub. . . . .	9
3.11	Добавление GPG ключа в GitHub. . . . .	9
3.12	Добавление GPG ключа в GitHub. . . . .	9
3.13	Настройка автоматических подписей коммитов git. . . . .	9
3.14	Настройка gh. . . . .	10
3.15	Создание репозитория курса на основе шаблона. . . . .	10
3.16	Создание репозитория курса на основе шаблона. . . . .	10
3.17	Настройка каталога курса. . . . .	10
3.18	Настройка каталога курса. . . . .	10

# 1 Цель работы

Целью данной работы является изучение идеологии и применения средств контроля версий и освоение умений работать с git.

## 2 Задание

- Создать базовую конфигурацию для работы с git.
- Создать ключ SSH.
- Создать ключ PGP.
- Настроить подписи git.
- Зарегистрироваться на Github.
- Создать локальный каталог для выполнения заданий по предмету.

### 3 Выполнение лабораторной работы

Создаю учетную запись на github и заполняю основные данные (рис. 3.1).

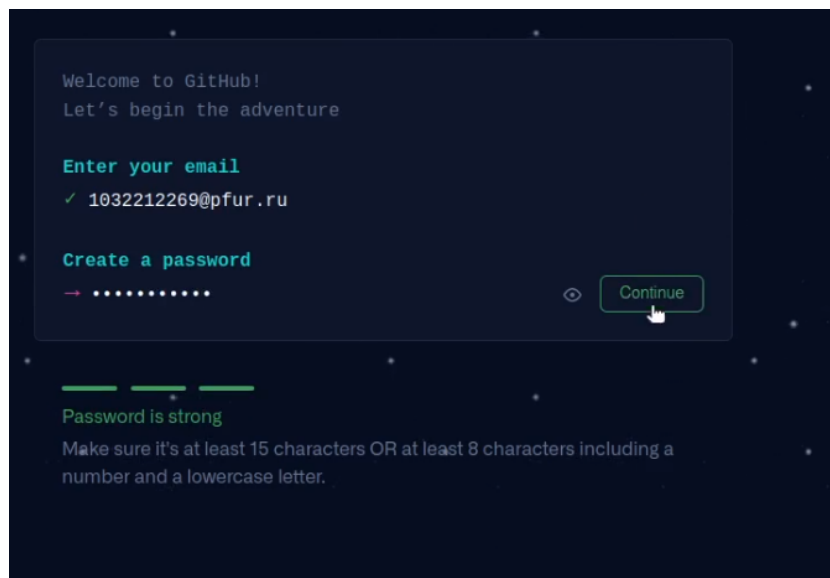


Рис. 3.1: Создание учётной записи на github.

Устанавливаю git-flow (рис. 3.2).

```
anastasia@Anastasia-PC:~/tutorial$ sudo apt-get install git-flow  
[sudo] Passwort für anastasia:
```

Рис. 3.2: Установка git-flow.

Устанавливаю gh (рис. 3.3).

```
anastasia@Anastasia-PC:~/tutorial$ sudo apt install gh  
[sudo] Passwort für anastasia:
```

Рис. 3.3: Установка gh.

Задаю имя и email владельца репозитория (рис. 3.4).

```
anastasia@Anastasia-PC: ~  
anastasia@Anastasia-PC:~$ git config --global user.name "Анастасия Баранова"  
anastasia@Anastasia-PC:~$ git config --global user.email "1032212269@pfur.ru"
```

Рис. 3.4: Базовая настройка git.

Настрою utf-8 в выводе сообщений git (рис. 3.5).

```
anastasia@Anastasia-PC:~$ git config --global core.quotepath false
```

Рис. 3.5: Базовая настройка git.

Настрою верификацию и подписание коммитов git. Зададам имя начальной ветки (будем называть её master), параметр autocrlf, параметр safecrlf (рис. 3.6).

```
anastasia@Anastasia-PC:~/tutorial$ git config --global init.defaultBranch master  
anastasia@Anastasia-PC:~/tutorial$ git config --global core.autocrlf input  
anastasia@Anastasia-PC:~/tutorial$ git config --global core.safecrlf warn  
anastasia@Anastasia-PC:~/tutorial$
```

Рис. 3.6: Базовая настройка git.

Создам ключ ssh по алгоритму rsa с ключём размером 4096 бит (рис. 3.7).

```
anastasia@Anastasia-PC:~/tutorial$ ssh-keygen -C "Анастасия Баранова <1032212269@pfur.ru>"  
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/anastasia/.ssh/id_rsa): pub  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in pub  
Your public key has been saved in pub.pub  
The key fingerprint is:  
SHA256:M7FgyjfgbS4AXuxSxEkHF7zMNI60JTtoLlMmpMOBy4dM Анастасия Баранова <1032212269@pfur.ru>  
The key's randomart image is:  
+---[RSA 3072]-----+  
|+ B+++.  
|+BoBo+  
|*o*+E = .  
|==* @ . o  
|.o+. = S  
|... + . o  
| . .  
| .  
+---[SHA256]-----+
```

Рис. 3.7: Создание ключа ssh.

Создам ключ ssh по алгоритму ed25519 (рис. 3.8).

```

anastasia@Anastasia-PC:~/tutorial$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/anastasia/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/anastasia/.ssh/id_ed25519
Your public key has been saved in /home/anastasia/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:jTla4KLzesdLz0FIeFufaGnDGSWcc6WSJ/f/CKvuPps anastasia@Anastasia-PC
The key's randomart image is:
+--[ED25519 256]--+
|      . . . . .      |
|      . ++..       |
|      . + *O+      |
|      + * # O      |
|      . + S + .    |
|      . . * O .    |
|      O .O . . .   |
|      O..OO ... O O|
|      .O....O+E+. .|
+-----[SHA256]-----+

```

Рис. 3.8: Создание ключа ssh.

Генерирую ключ gpg (рис. 3.9).

```

anastasia@Anastasia-PC:~/tutorial$ gpg --full-generate-key
gpg (GnuPG) 2.2.19; Copyright (C) 2019 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Bitte wählen Sie, welche Art von Schlüssel Sie möchten:
(1) RSA und RSA (voreingestellt)
(2) DSA und Elgamal
(3) DSA (nur signieren/beglaubigen)
(4) RSA (nur signieren/beglaubigen)
(14) Vorhandener Schlüssel auf der Karte
Ihre Auswahl? 1
RSA-Schlüssel können zwischen 1024 und 4096 Bit lang sein.
Welche Schlüssellänge wünschen Sie? (3072) 4096
Die verlangte Schlüssellänge beträgt 4096 Bit
Bitte wählen Sie, wie lange der Schlüssel gültig bleiben soll.
0 = Schlüssel verfällt nie
<n> = Schlüssel verfällt nach n Tagen
<n>w = Schlüssel verfällt nach n Wochen
<n>m = Schlüssel verfällt nach n Monaten
<n>y = Schlüssel verfällt nach n Jahren
Wie lange bleibt der Schlüssel gültig? (0)
Schlüssel verfällt nie
Ist dies richtig? (j/N)

```

Рис. 3.9: Создание ключа gpg.

Добавляю GPG ключ в GitHub. Вывожу список ключей и копирую отпечаток приватного ключа (рис. 3.10 3.11).



```

anastasia@Anastasia-PC:~/tutorial$ gpg --list-secret-keys --keyid-format LONG
/home/anastasia/.gnupg/pubring.kbx
-----
sec   rsa4096/A380A490DB872F35 2022-04-20 [SC]
      845899AE8414BF0C121431E0A380A490DB872F35
uid    [uneingeschränkt] apparanova <1032212269@pfur.ru>
ssb    rsa4096/9F88A0954484B719 2022-04-20 [E]

```

Рис. 3.10: Добавление GPG ключа в GitHub.

```

anastasia@Anastasia-PC:~/tutorial$ gpg --armor --export A380A490DB872F35 | xclip -sel clip
anastasia@Anastasia-PC:~/tutorial$

```

Рис. 3.11: Добавление GPG ключа в GitHub.

Перехожу в настройки GitHub (<https://github.com/settings/keys>), нажимаю на кнопку New GPG key и вставляю полученный ключ в поле ввода (рис. 3.12).

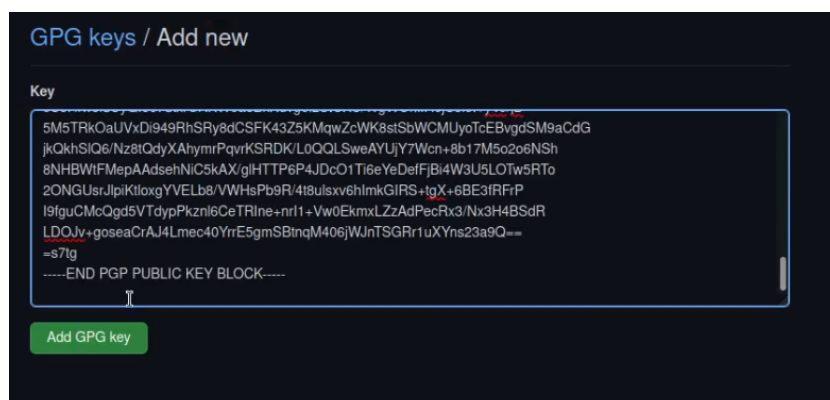


Рис. 3.12: Добавление GPG ключа в GitHub.

Настрою автоматические подписи коммитов git. Используя введённый email, укажу Git применять его при подписи коммитов (рис. 3.13).

```

anastasia@Anastasia-PC:~/tutorial$ git config --global user.signingkey A380A490DB872F35
anastasia@Anastasia-PC:~/tutorial$ git config --global commit.gpgsign true
anastasia@Anastasia-PC:~/tutorial$ git config --global gpg.program $(which gpg2)
anastasia@Anastasia-PC:~/tutorial$

```

Рис. 3.13: Настройка автоматических подписей коммитов git.

Настрою gh. Авторизуюсь (рис. 3.14).

```

anastasia@Anastasia-PC:~/tutorial$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? SSH
? Generate a new SSH key to add to your GitHub account? No
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: 39C9-7932
Press Enter to open github.com in your browser...
✓ Authentication complete.
- gh config set -h github.com git_protocol ssh
✓ Configured git protocol
✓ Logged in as apbaranova

```

Рис. 3.14: Настройка gh.

Создам репозиторий курса на основе шаблона (рис. 3.15 3.16).

```

anastasia@Anastasia-PC:~/tutorial$ mkdir -p ~/work/study/2021-2022/"Операционные системы"
anastasia@Anastasia-PC:~/tutorial$ cd ~/work/study/2021-2022/"Операционные системы"
anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы$ gh repo create study_2021-2022_os-intro -
-template=yamadharma/course-directory-student-template --public
✓ Created repository apbaranova/study_2021-2022_os-intro on GitHub

```

Рис. 3.15: Создание репозитория курса на основе шаблона.

```

anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы$ git clone --recursive git@github.com:apb-
ranova/study_2021-2022_os-intro.git os-intro

```

Рис. 3.16: Создание репозитория курса на основе шаблона.

Настрою каталог курса (рис. 3.17 3.18).

```

anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы$ cd ~/work/study/2021-2022/"Операционные с
истемы"/os-intro
anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы/os-intro$ rm package.json
anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы/os-intro$ make COURSE=os-intro
anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы/os-intro$ git add .
anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы/os-intro$ git commit -am 'feat(main): make
course structure'

```

Рис. 3.17: Настройка каталога курса.

```

anastasia@Anastasia-PC:~/work/study/2021-2022/Операционные системы/os-intro$ git push

```

Рис. 3.18: Настройка каталога курса.

## 4 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?

Ответ:

Система контроля версий (VCS) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Для примеров в этой книге мы будем использовать исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа. Если вы графический или веб-дизайнер и хотели бы хранить каждую версию изображения или макета — а этого вам наверняка хочется — то пользоваться системой контроля версий будет очень мудрым решением. даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь, вы всё испортите или потеряете файлы, всё можно будет легко восстановить. Вдобавок, накладные расходы за всё, что вы получаете, будут очень маленькими.

2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.

Ответ: Хранилище-система, которая обеспечивает хранение всех существовавших вариантов файлов Commit-фиксация изменений История-список преды-

душих ревизий Рабочая копия-копия другой ветки Команде `commit` можно передать сообщение, описывающее изменения в ревизии. Она также записывает идентификатор пользователя, текущее время и временную зону, плюс список измененных файлов и их содержимого. Сообщение, описывающее изменения, определяется через опцию `-m`, или `- message`. Можно также вводить сообщения, состоящие из нескольких строк; в большинстве оболочек вы можете сделать это оставив открытую кавычку в конце строки. `commit -m` “добавлен первый файл.

3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.

Ответ: Системы контроля версий. Централизованная система контроля версий Subversion и децентрализованная система контроля версий Mercurial. Существуют СКВ централизованные, в которых имеется один репозиторий, в который собираются изменения со всех рабочих копий разработчиков, и децентрализованные, когда репозиторий много, и они могут обмениваться изменениями между собой. Централизованные СКВ - репозиторий один. У каждого разработчика своя рабочая копия. Время от времени разработчик может затягивать к себе в рабочую копию новые изменения из репозитория, или проталкивать свои изменения из своей рабочей копии в репозиторий. Прочие особенности централизованных СКВ зависят от реализации.

4. Опишите действия с VCS при единоличной работе с хранилищем.

Ответ: Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, т. н. «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии

или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, новая версия помещается в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время. Сервер может использовать т. н. дельта-компрессию — такой способ хранения документов, при котором сохраняются только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Некоторые системы (например, ClearCase) поддерживают сохранение версий обоих видов: большинство версий сохраняется в виде дельт, но периодически (по специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

##### 5. Опишите порядок работы с общим хранилищем VCS.

Ответ: Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, т. н. «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, новая версия помещается в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время. Сервер может использовать т. н. дельта-компрессию — такой способ хранения документов, при котором сохраняются только изменения между последовательными версиями,

что позволяет уменьшить объём хранимых данных. Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Некоторые системы (например, ClearCase) поддерживают сохранение версий обоих видов: большинство версий сохраняется в виде дельт, но периодически (по специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

#### 6. Каковы основные задачи, решаемые инструментальным средством git?

Ответ: Устанавливает единственную новую команду, git. Все возможности предоставляются через подкоманды этой команды. Вы можете просмотреть краткую справку командой help. Некоторые идеи группируются по темам, используйте help topics для списка доступных тем. Одна из функций системы контроля версий — отслеживать кто сделал изменения. В распределенных системах для этого требуется идентифицировать каждого автора уникально в глобальном плане. Большинство людей уже имеют такой идентификатор: email адрес. Bazaar достаточно умен, чтобы автоматически создавать email адрес из текущего имени и адреса хоста. Основные задачи: создание ветки, размещение веток, просмотр изменений, фиксация изменений, сообщение из текстового редактора, выборочная фиксация, удаление зафиксированных изменений, игнорирование файлов, просмотр истории, статистика ветки, контроль файлов и каталогов, ветвление, объединение веток, публикация ветки.

#### 7. Назовите и дайте краткую характеристику командам git.

Ответ: Обновление рабочей копии По мере внесения изменений в проект рабочая копия на компьютере разработчика стареет, расхождение её с основной версией проекта увеличивается. Это повышает риск возникновения конфликтных изменений (см. ниже). Поэтому удобно поддерживать рабочую копию в

состоянии, максимально близком к текущей основной версией, для чего разработчик выполняет операцию обновления рабочей копии (update) насколько возможно часто (реальная частота обновлений определяется частотой внесения изменений, зависящей от активности разработки и числа разработчиков, а также временем, затрачиваемым на каждое обновление — если оно велико, разработчик вынужден ограничивать частоту обновлений, чтобы не терять время). Модификация проекта Разработчик модифицирует проект, изменяя входящие в него файлы в рабочей копии в соответствии с проектным заданием. Эта работа производится локально и не требует обращений к серверу VCS. Фиксация изменений Завершив очередной этап работы над заданием, разработчик фиксирует (commit) свои изменения, передавая их на сервер (либо в основную ветвь, если работа над заданием полностью завершена, либо в отдельную ветвь разработки данного задания). VCS может требовать от разработчика перед фиксацией обязательно выполнить обновление рабочей копии. При наличии в системе поддержки отложенных изменений (shelving) изменения могут быть переданы на сервер без фиксации. Если утверждённая политика работы в VCS это позволяет, то фиксация изменений может проводиться не ежедневно, а только по завершении работы над заданием; в этом случае до завершения работы все связанные с заданием изменения сохраняются только в локальной рабочей копии разработчика.

#### 8. Приведите примеры использования при работе с локальным и удалённым репозиториями.

Ответ: Мы создаем новую ветку выполнив `git init` в уже созданном каталоге: `% mkdir tutorial % cd tutorial % ls -a ./ ../ % pwd /home/mbp/work/bzr.test/tutorial % % git init % ls -aF ./ ../ .git/ %` Мы обычно обращаемся к веткам на нашем компьютере просто передав имя каталога содержащего ветку. `bzr` также поддерживает доступ к веткам через `http` и `sftp`, например: `git log http://bazaar-vcs.org git // git.dev/ git log sftp://bazaarvcs.org/bzr/bzr.dev/` Установив для `git` плагины можно также осуществлять доступ к веткам с использованием `rsync`. Команда `status`

показывает какие изменения были сделаны в рабочем каталоге с момента последней ревизии: `% git status modified: foo` `bzr status` скрывает неинтересные файлы, которые либо не менялись, либо игнорируются. Также команде `status` могут быть переданы необязательные имена файлов, или каталогов для проверки. Команда `diff` показывает изменения в тексте файлов в стандартном формате `diff`. Вывод этой команды может быть передан другим командам, таким как `"patch"`, `"diffstat"`, `"filterdiff"` и `"colordiff"`: `% git diff == added file 'hello.txt' — hello.txt`  
`1970-01-01 00:00:00 +0000 +++ hello.txt 2005-10-18 14:23:29 +0000`  
6.2. Указания к лабораторной работе  
75 `@@ -0,0 +1,1 @@ +hello world`  
Команде `commit` можно передать сообщение описывающее изменения в ревизии. Она также записывает идентификатор пользователя, текущее время и временную зону, плюс список измененных файлов и их содержимого. `git commit -m "добавлен первый файл"`  
Если вы передадите список имен файлов, или каталогов после команды `commit`, то будут зафиксированы только изменения для переданных объектов. Например: `bzr commit -m "исправления документации" commit.py`  
Если вы сделали какие-либо изменения и не хотите оставлять их, используйте команду `revert`, что бы вернуться к состоянию предыдущей ревизии. Многие деревья с исходным кодом содержат файлы которые не нужно хранить под контролем версий, например резервные файлы текстового редактора, объектные файлы и собранные программы. Вы можете просто не добавлять их, но они всегда будут обнаруживаться как неизвестные. Вы также можете сказать `git` игнорировать их добавив их в файл `.ignore` в корне рабочего дерева. Для получения списка файлов которые игнорируются и соответствующих им шаблонов используйте команду `ignored: % ignored config.h ./config.h configure.in ~ *~`  
Команда `bzr log` показывает список предыдущих ревизий. Команда `log -forward` делает тоже самое, но в хронологическом порядке, показывая более поздние ревизии в конце может контролировать файлы и каталоги, отслеживая переименования и упрощая их последующее объединение: `% mkdir src % echo 'int main() {}' > src/simple.c % add src added src added src/simple.c % status added: src/ src/simple.c`  
**bzr remove** удаляет



файл из под контроля версий, но может и не удалять рабочую копию файла<sup>2</sup>. Это удобно, когда вы добавили не тот файл, или решили, что файл на самом деле не должен быть под контролем версий. `% rm -r src % remove -v hello.txt ? hello.txt`  
`% status removed: hello.txt src/ src/simple.c unknown: hello.txt` Часто вместо того что бы начинать свой собственный проект, выхотите предложить изменения для уже готового проекта. Что бы сделать это вам нужно получить копию готовой ветки. Так как эта копия может быть потенциальной новой веткой. Если две ветки разошлись (обе имеют уникальные изменения) тогда `merge` — это подходящая команда для использования. Объединение автоматически вычислит изменения, которые существуют на объединяемой ветке и отсутствуют в локальной ветке и попытается объединить их с локальной веткой. `git merge URL`.

#### 9. Что такое и зачем могут быть нужны ветви (branches)?

Ответ: Часто вместо того что бы начинать свой собственный проект, вы хотите предложить изменения для уже готового проекта. Что бы сделать это вам нужно получить копию готовой ветки. Так как эта копия может быть потенциальной новой веткой эта команда называется `branch`: Управление версиями `git branch`  
`cd git.dev` Эта команда копирует полную историю ветки и после этого вы можете делать все операции с ней локально: просматривать журнал, создавать и объединять другие ветки.

#### 10. Как и зачем можно игнорировать некоторые файлы при `commit`?

Ответ: Нет проблем если шаблон для игнорирования подходит для файла под контролем версий, или вы добавили файл, который игнорируется. Шаблоны не имеют никакого эффекта на файлы под контролем версий, они только определяют показываються неизвестные файлы, или просто игнорируются. Файл `git.rignore` обычно должен быть под контролем версий, что бы новые копии ветки видели такие же шаблоны: `git add . gitignore git commit -m "Добавлены шаблоны для игнорирования"`. Многие деревья с исходным кодом содержат файлы, которые не

нужно хранить под контролем версий, например, резервные файлы текстового редактора, объектные файлы и собранные программы. Вы можете просто не добавлять их, но они всегда будут обнаруживаться как неизвестные. Вы также можете сказать bazaar игнорировать их добавив их в файл в корне рабочего дерева. Этот файл содержит список шаблонов файлов, по одному в каждой строке. Обычное содержимое может быть таким: \*.o \*~ \*.tmp \*.py [ со ] Если шаблон содержит слеш, то он будет сопоставлен с полным путем начиная от корня рабочего дерева; иначе он сопоставляется только с именем файла. Таким образом пример выше игнорирует файлы с расширением .o во всех подкаталогах, но пример ниже игнорирует только config.h в корне рабочего дерева и HTML файлы в каталоге doc/: ./config.h doc/\*.html Для получения списка файлов которые игнорируются и соответствующих им шаблонов используйте команду git ignored : \$ git ignored config.h ./config.h configure.in~ \*~ \$

## 5 Выводы

В ходе выполнения данной лабораторной работы я изучила идеологию и применение средств контроля версий и освоила умения по работе с git.

# Список литературы

1. Лекция Системы контроля версий