

# **Лабораторная работа №2**

**Операционные системы**

Лазева Диана Анатольевна НБИбд-04-22

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольные вопросы	12
4	Выводы	20
	Список литературы	21

## Список иллюстраций

2.1	Зададим имя и email владельца репозитория . . . . .	6
2.2	Настроим utf-8 в выводе сообщений git . . . . .	6
2.3	Зададим имя начальной ветки (будем называть её master) . . . .	6
2.4	Параметр autocrlf и параметр safecrlf . . . . .	7
2.5	SSH . . . . .	7
2.6	PGP . . . . .	8
2.7	Выводим список ключей и копируем отпечаток приватного ключа	9
2.8	Используя введенный email, укажите Git применять его при подписи коммитов: . . . . .	10
2.9	авторизация . . . . .	10
2.10	создание репозитория . . . . .	10
2.11	настройка каталога курса . . . . .	11

## Список таблиц

# 1 Цель работы

Изучить идеологию и применение средств контроля версий. Освоить умения по работе с git.

## 2 Выполнение лабораторной работы

Задание

Создать базовую конфигурацию для работы с git.

Создать ключ SSH.

Создать ключ PGP.

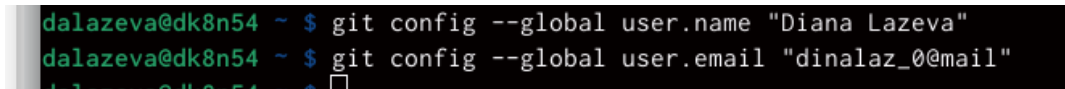
Настроить подписи git.

Зарегистрироваться на Github.

Создать локальный каталог для выполнения заданий по предмету.

Последовательность выполнения работы

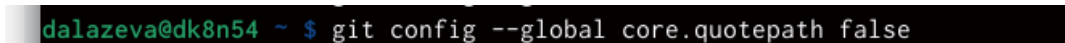
Базовая настройка git (рис. [2.1])



```
dalazeva@dk8n54 ~ $ git config --global user.name "Diana Lazeva"
dalazeva@dk8n54 ~ $ git config --global user.email "dinalaz_0@mail"
dalazeva@dk8n54 ~ $
```

Рис. 2.1: Зададим имя и email владельца репозитория

(рис. [2.2])



```
dalazeva@dk8n54 ~ $ git config --global core.quotePath false
dalazeva@dk8n54 ~ $
```

Рис. 2.2: Настроим utf-8 в выводе сообщений git

(рис. [2.3])



```
dalazeva@dk8n54 ~ $ git config --global init.defaultBranch master
dalazeva@dk8n54 ~ $
```

Рис. 2.3: Зададим имя начальной ветки (будем называть её master)

(рис. [2.4])

```
dalazeva@dk8n54 ~ $ git config --global core.autocrlf input
dalazeva@dk8n54 ~ $ git config --global core.safecrlf warn
```

Рис. 2.4: Параметр autocrlf и параметр safecrlf

Создаем ключи ssh (рис. [2.5])

```
dalazeva@dk8n54 ~ $ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/d/a/dalazeva/.ssh/id_rsa):
/afs/.dk.sci.pfu.edu.ru/home/d/a/dalazeva/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /afs/.dk.sci.pfu.edu.ru/home/d/a/dalazeva/.ssh/id_rsa
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/d/a/dalazeva/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:oxQIuIy8ss57FejQYLafQ/a+htx07AwToiwjPWNod58 dalazeva@dk8n54
The key's randomart image is:
+---[RSA 4096]-----+
|o.+                |
|.* o .            |
|*.* = .           |
|+O B + .          |
|*oX.=.= S         |
|==.B.X...         |
|..o = +E          |
|o o               |
|. +o              |
+----[SHA256]-----+
dalazeva@dk8n54 ~ $ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/d/a/dalazeva/.ssh/id_ed25519):
/afs/.dk.sci.pfu.edu.ru/home/d/a/dalazeva/.ssh/id_ed25519 already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /afs/.dk.sci.pfu.edu.ru/home/d/a/dalazeva/.ssh/id_ed25519
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/d/a/dalazeva/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:laIZeRIOJtyDSCd2sUaxmeKkmZoPIRVg49HIyId2YYQ dalazeva@dk8n54
The key's randomart image is:
+--[ED25519 256]--+
|*@0%* .           |
|*E@BBo o .        |
|.+++ . = o o       |
|++o  * o           |
|* .   o S          |
|oo                 |
|+                  |
```

Рис. 2.5: SSH

Создаем ключи pgp (рис. [2.6])

```
dalazeva@dk8n54 ~ $ gpg --full-generate-key
gpg (GnuPG) 2.2.40; Copyright (C) 2022 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Выберите тип ключа:
  (1) RSA и RSA (по умолчанию)
  (2) DSA и Elgamal
  (3) DSA (только для подписи)
  (4) RSA (только для подписи)
  (14) Имеющийся на карте ключ
Ваш выбор? 1
длина ключей RSA может быть от 1024 до 4096.
Какой размер ключа Вам необходим? (3072) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
  0 = не ограничен
  <n> = срок действия ключа - n дней
  <n>w = срок действия ключа - n недель
  <n>m = срок действия ключа - n месяцев
  <n>y = срок действия ключа - n лет
Срок действия ключа? (0) 0
Срок действия ключа не ограничен
Все верно? (y/N) y

GnuPG должен составить идентификатор пользователя для идентификации ключа.

Ваше полное имя: dinalaz
Адрес электронной почты: dinalaz_0@mail.ru
Примечание:
Вы выбрали следующий идентификатор пользователя:
"dinalaz <dinalaz_0@mail.ru>"

Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход? O
Необходимо получить много случайных чисел. Желательно, чтобы Вы
в процессе генерации выполняли какие-то другие действия (печать
на клавиатуре, движения мыши, обращения к дискам); это даст генератору
случайных чисел больше возможностей получить достаточное количество энтропии.
Необходимо получить много случайных чисел. Желательно, чтобы Вы
в процессе генерации выполняли какие-то другие действия (печать
на клавиатуре, движения мыши, обращения к дискам); это даст генератору
случайных чисел больше возможностей получить достаточное количество энтропии.
```

Рис. 2.6: PGP

Добавление PGP ключа в GitHub(рис. [2.7])



```

dalazeva@dk8n54 ~ $ gpg --list-secret-keys --keyid-format LONG

gpg: проверка таблицы доверия
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: глубина: 0 достоверных: 2 подписанных: 0 доверие: 0-, 0q, 0n, 0m, 0f, 2u
/afs/.dk.sci.pfu.edu.ru/home/d/a/dalazeva/.gnupg/pubring.kbx
-----
sec   rsa4096/9C610729B2B0B224 2023-02-17 [SC]
      6204F228EC2CD1D3BFED7C419C610729B2B0B224
uid           [ абсолютно ] Dinalaz <dinalaz_0@mail.ru>
ssb   rsa4096/F659DB6DE6B10FB7 2023-02-17 [E]

sec   rsa4096/A7ED85FDECE71B0D 2023-02-17 [SC]
      502100FB5F18C165D6348AB2A7ED85FDECE71B0D
uid           [ абсолютно ] dinalaz <dinalaz_0@mail.ru>
ssb   rsa4096/D8FB8A7DB128BDDF 2023-02-17 [E]

dalazeva@dk8n54 ~ $ gpg --armor --export <PGP Fingerprint> | xclip -sel clip

bash: синтаксическая ошибка рядом с неожиданным маркером «|»
dalazeva@dk8n54 ~ $ gpg --armor --export D8FB8A7DB128BDDF | xclip -sel clip

```

Рис. 2.7: Выводим список ключей и копируем отпечаток приватного ключа

## GPG keys / Add new

Title

1234567

Key

```

-----BEGIN PGP PUBLIC KEY BLOCK-----
mQINBQYKAAQIACwAAAAIBQYKAAQIACwAAAAIBQYKAAQIACwAAAAIBQYKAAQIACw
Zs+pQwonW0iecytDYreskldIEZihy56UcNUVSLChbhaJO/WceKGXMAf0xwhX9shx
XNCEVxfK5Y/QCADabjxqUZ3hEI0e4aCTQnV0yVI0tDdgGR+pdKGvzeeCMWLT4lqe
q06r5Ya230phWiXFDwegmR9i6xGTcsizLxb9XEbWLDAG8tPjfxGPKI089MQq2BLL
BpTGrUjqZThcu6LVhH33ZiOezVsRyz/MkZwomIKLF6hw7BNnh3VYqyQEwQ7O518
gRP84EGO/g1H6U0C9uZa0y4LSvczlhYsp9t0qUn1MZFI6WaHtK8+wnq7QkSCC7Bu
im8guQ76bzfWH8Wy7b97sQvuSAQsQFnzmkUx++B9dE+FJE0/pCbH
=jP3H
-----END PGP PUBLIC KEY BLOCK-----

```

Add GPG key

(рис. [??])

Настройка автоматических подписей коммитов git (рис. [2.8])

```
dalazeva@dk8n54 ~ $ git config --global user.signingkey D8FB8A7DB128BDDF
dalazeva@dk8n54 ~ $ git config --global commit.gpgsign true
dalazeva@dk8n54 ~ $ git config --global gpg.program $(which gpg2)
```

Рис. 2.8: Используя введённый email, укажите Git применять его при подписи  
КОММИТОВ:

Настройка gh (рис. [2.9])

авторизация

Рис. 2.9: авторизация

Создание репозитория курса на основе шаблона (рис. [2.10])

создание репозитория

Рис. 2.10: создание репозитория

Настройка каталога курса рис. [2.11])

```

dalazeva@dk8n54 ~/work/study/2022-2023/Операционные системы
dalazeva@dk8n54 ~/work/study/2022-2023/Операционные системы/os-intro
$ rm package.json
rm: невозможно удалить 'package.json': Нет такого файла или каталога
dalazeva@dk8n54 ~/work/study/2022-2023/Операционные системы/os-intro
$ echo os-intro > COURSE
dalazeva@dk8n54 ~/work/study/2022-2023/Операционные системы/os-intro
$ make
make: Цель «all» не требует выполнения команд.
dalazeva@dk8n54 ~/work/study/2022-2023/Операционные системы/os-intro
$ git add .
dalazeva@dk8n54 ~/work/study/2022-2023/Операционные системы/os-intro
$ git commit -am 'feat(main): make course structure'
[master 9c913d2] feat(main): make course structure
16 files changed, 898 insertions(+), 38 deletions(-)
create mode 100644 labs/lab01/presentation/presentation.html
create mode 100644 labs/lab01/presentation/presentation.pdf
create mode 100644 labs/lab02/report/image/auth.png
create mode 100644 labs/lab02/report/image/dobavl1.png
create mode 100644 labs/lab02/report/image/dobavl2.png
create mode 100644 labs/lab02/report/image/gpg.png
create mode 100644 labs/lab02/report/image/keys.png
create mode 100644 labs/lab02/report/image/kod.png
create mode 100644 labs/lab02/report/image/master.png
create mode 100644 labs/lab02/report/image/name.png
create mode 100644 labs/lab02/report/image/parametry.png
create mode 100644 labs/lab02/report/image/podpisi.png
create mode 100644 labs/lab02/report/image/sozdanie.png
create mode 100644 labs/lab02/report/report.docx
create mode 100644 labs/lab02/report/report.pdf
dalazeva@dk8n54 ~/work/study/2022-2023/Операционные системы/os-intro
$ git push
git@github.com: Permission denied (publickey).
fatal: Не удалось прочитать из внешнего репозитория.

Удостоверьтесь, что у вас есть необходимые права доступа
и репозиторий существует.
dalazeva@dk8n54 ~/work/study/2022-2023/Операционные системы/os-intro $

```

Рис. 2.11: настройка каталога курса

## 3 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? 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. Хранилище-система, которая обеспечивает хранение всех существовавших вариантов файлов Commit-фиксация изменений История-список предыдущих ревизий Рабочая копия-копия другой ветки Команде commit можно передать сообщение, описывающее изменения в

реvisions. 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. 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 репозитория 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. 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. е. «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, но-

вая 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. 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. дельт компрессию — такой способ хранения документов, при котором сохраняются только измене-

ния 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? 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. 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 версии, 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. 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 +00006.2. 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~ *~ log` 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. `gitmerge URL`.

9. Что такое и зачем могут быть нужны ветви (branches)? 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`? 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 показывающиеся 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 `bzr` игнорировать их добавив их в файл в корне рабочего дерева. Этот файл содержит список шаблонов файлов, по одному в каждой строчке. 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~ ~ $
```

## 4 Выводы

Я изучила идеологию и применение средств контроля версий, а так же освоила умения по работе с git.

## **Список литературы**