

# **Отчет по лабораторной работе №2**

**Группа НБИбд-02-22**

Сидоренко Максим Алексеевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>4</b>	<b>Контрольные вопросы</b>	<b>16</b>
<b>5</b>	<b>Отвеы на вопросы</b>	<b>17</b>
<b>6</b>	<b>Вывод</b>	<b>25</b>

## Список иллюстраций

3.1	Базовые конфиги . . . . .	7
3.2	utf-8 в выводе сообщений git . . . . .	7
3.3	Называние начальной ветки . . . . .	8
3.4	autocrlf . . . . .	8
3.5	safecrlf . . . . .	8
3.6	по алгоритму rsa . . . . .	8
3.7	по алгоритму ed25519 . . . . .	8
3.8	Создал ключи pgr . . . . .	9
3.9	Выбор параметров . . . . .	9
3.10	Выбор параметров . . . . .	9
3.11	Регистрация аккаунта . . . . .	10
3.12	Заполнение профиля . . . . .	10
3.13	Вывод списка ключей . . . . .	11
3.14	Копирование отпечатка . . . . .	11
3.15	Переход в GitHub new Gpg key . . . . .	11
3.16	Переход в GitHub new Gpg key . . . . .	11
3.17	Переход в GitHub new Gpg key . . . . .	12
3.18	Настройка автоматических подписей коммитов git . . . . .	12
3.19	Настройка gh . . . . .	12
3.20	Настройка gh . . . . .	12
3.21	Создание репозитория курса . . . . .	13
3.22	Создание репозитория курса . . . . .	13
3.23	Создание репозитория курса взятие ключа ssh . . . . .	13
3.24	перешел и удалил лишние файлы . . . . .	14
3.25	Создал необходимые каталоги . . . . .	14
3.26	Отправка файлов на сервер . . . . .	14
3.27	Отправка файлов на сервер . . . . .	14
3.28	Проверка . . . . .	15

## Список таблиц

# 1 Цель работы

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

1) Я запустил терминал и установил ПО git и gh

```
masidorenkor@masidorenkor:~$ sudo apt install git
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Следующие пакеты устанавливались автоматически и больше не требуются:
  libflashrom1 libftdi1-2 liblvm13
Для их удаления используйте «sudo apt autoremove».
Будут установлены следующие дополнительные пакеты:
  git-man liberror-perl
Предлагаемые пакеты:
  git-daemon-run | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb git-cvs
  git-mediawiki git-svn
Следующие НОВЫЕ пакеты будут установлены:
  git git-man liberror-perl
Обновлено 0 пакетов, установлено 3 новых пакетов, для удаления отмечено 0 пакетов, и 8
пакетов не обновлено.
Необходимо скачать 4 121 кВ архивов.
После данной операции объём занятого дискового пространства возрастёт на 20,9 МВ.
Хотите продолжить? [Д/н]
```

```
masidorenkor@masidorenkor:~$ sudo apt install gh
Чтение списков пакетов... Готово
```

2) Я произвел базовую конфигурацию для работы с git

```
masidorenkor@masidorenkor:~$ git config --global user.name "MaximSidorenkoNB12"
masidorenkor@masidorenkor:~$ git config --global user.email "justforrudn@mail.ru"
```

Рис. 3.1: Базовые конфиги

3) Настроил utf-8 в выводе сообщений git

```
masidorenkor@masidorenkor:~$ git config --global user.name "justforrudn@mail.ru"
masidorenkor@masidorenkor:~$ git config --global core.quotepath false
masidorenkor@masidorenkor:~$
```

Рис. 3.2: utf-8 в выводе сообщений git

4) Задал имя начальной ветки (буду называть её master)

```
masidorenkor@masidorenkor:~$ git config --global init.defaultBranch master
```

Рис. 3.3: Называние начальной ветки

- 5) Ввел параметры autocrlf и safecrlf

```
masidorenkor@masidorenkor:~$ git config --global core.autocrlf input
```

Рис. 3.4: autocrlf

```
masidorenkor@masidorenkor:~$ git config --global core.safecrlf warn
```

Рис. 3.5: safecrlf

- 6) Создал ключи ssh по алгоритму rsa с ключём размером 4096 бит и по по алгоритму ed25519

```
masidorenkor@masidorenkor:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
```

Рис. 3.6: по алгоритму rsa

```
[SSH255]
masidorenkor@masidorenkor:~$ ssh-keygen -t ed25519
```

Рис. 3.7: по алгоритму ed25519

- 7) Создал ключи pgr при следующих действиях: сгенерировал ключ, Из предложенных опций выбирал: тип RSA and RSA, размер 4096, выбрал срок действия; значение по умолчанию — 0 (срок действия не истекает никогда), GPG запросил личную информацию, которая сохранится в ключе, ввел свое имя и адрес электронной почты, При вводе email убедился, что он соответствует адресу, используемому на GitHub, в комментарии ничего не вводил



```

masldorenkor@masldorenkor:~$ gpg --full-generate-key
gpg (GnuPG) 2.2.27; Copyright (C) 2021 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.

```

Рис. 3.8: Создал ключи pgr

```

Выберите тип ключа:
  (1) RSA и RSA (по умолчанию)
  (2) DSA и Elgamal
  (3) DSA (только для подписи)
  (4) RSA (только для подписи)
  (14) Именющийся 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:

```

Рис. 3.9: Выбор параметров

```

GnuPG должен составить идентификатор пользователя для идентификации ключа.
Ваше полное имя: MaxIm
Адрес электронной почты: justforrudn@mail.ru
Примечание:
Вы выбрали следующий идентификатор пользователя:
"MaxIm <jjustforrudn@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. 3.10: Выбор параметров

8) Создал аккаунт на GitHub, заполнил профиль

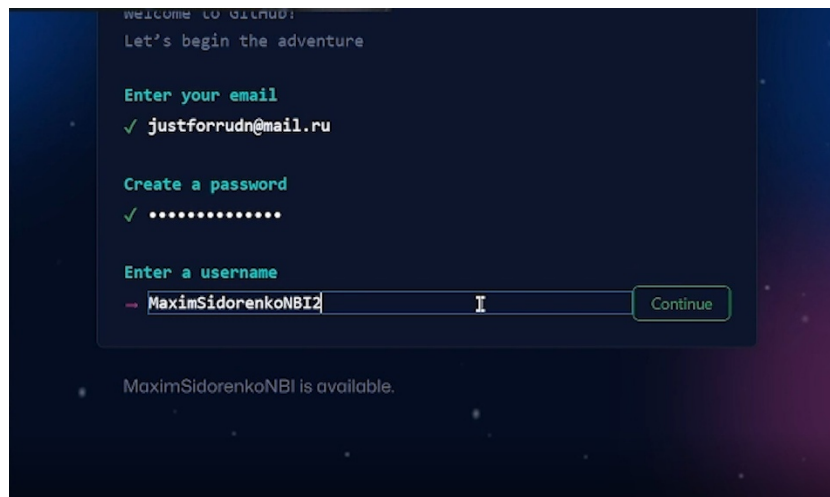


Рис. 3.11: Регистрация аккаунта

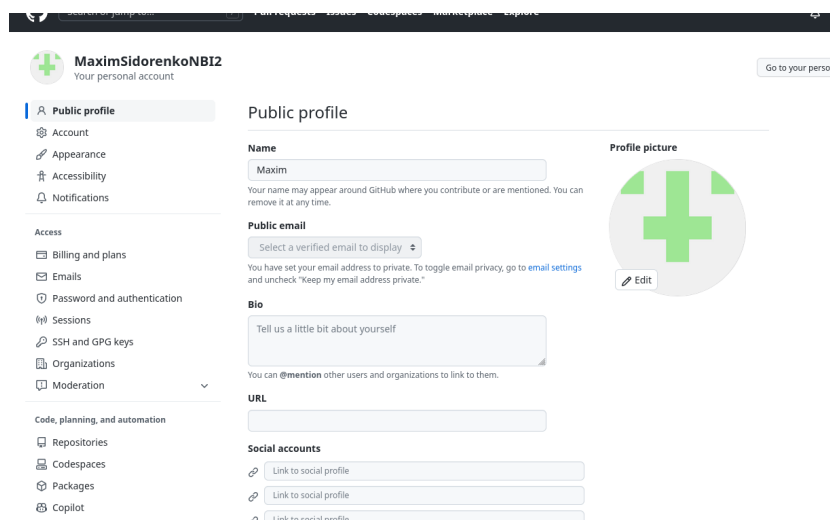


Рис. 3.12: Заполнение профиля

- 9) Добавил PGP ключ в GitHub при следующих действиях: Вывел список ключей и скопировал отпечаток приватного ключа, Скопировал мой сгенерированный PGP ключ в буфер обмена, перешел в настройки GitHub (<https://github.com/settings/keys>), нажал на кнопку New GPG key и вставил полученный ключ в поле ввода.

```
masidorenkor@masidorenkor:~$ gpg --list-secret-keys --keyid-format LONG
/home/masidorenkor/.gnupg/pubring.kbx
-----
sec   rsa4096/A060935245E37B97 2023-02-14 [SC]
      6875F2AA62A462A210AB0C44A060935245E37B97
uid    [ абсолютно ] Maxim <justforrudn@mail.ru>
ssb    rsa4096/B2F417616F89326B 2023-02-14 [E]
masidorenkor@masidorenkor:~$
```

Рис. 3.13: Вывод списка ключей

```
masidorenkor@masidorenkor:~$ gpg --armor --export A060935245E37B97 | xclip -sel clip
```

Рис. 3.14: Копирование отпечатка

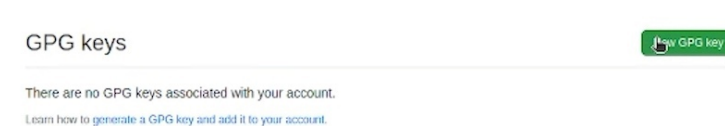


Рис. 3.15: Переход в GitHub new Gpg key

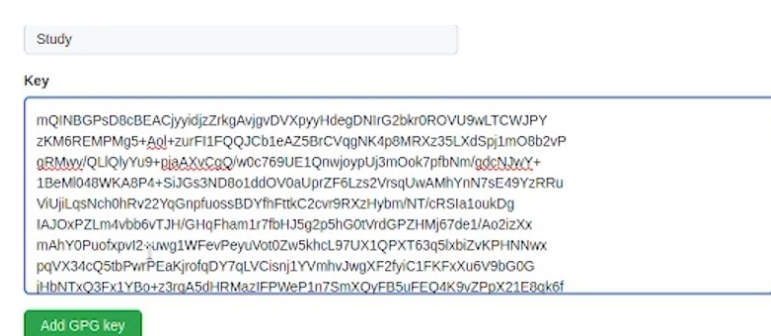


Рис. 3.16: Переход в GitHub new Gpg key

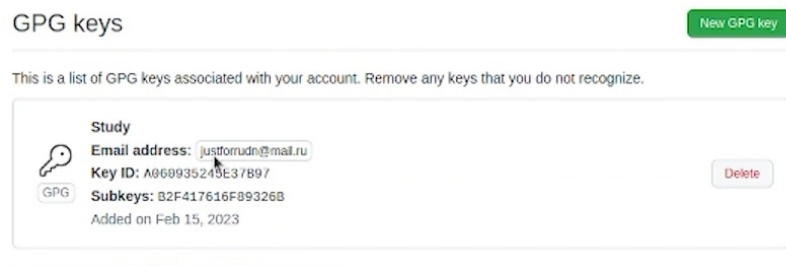


Рис. 3.17: Переход в GitHub new Gpg key

- 10) Настроил автоматические подписи коммитов git, Использовал мою почту от GitHub email, указал Git применять его при подписи коммитов:

```
masidorenkor@masidorenkor:~$ gpg --armor --export A060935245E37B97 | xclip -sel clip
masidorenkor@masidorenkor:~$ git config --global user.signingkey A060935245E37B97
masidorenkor@masidorenkor:~$ git config --global commit.gpgsign true
masidorenkor@masidorenkor:~$ git config --global user.signingkey justforrudn@mail.ru
masidorenkor@masidorenkor:~$ git config --global gpg.program $(which gpg2)
```

Рис. 3.18: Настройка автоматических подписей коммитов git

- 11) Настроил gh, Для начала авторизовался, Утилита задала несколько наводящих вопросов. Авторизовался через браузер

```
masidorenkor@masidorenkor:~$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? [Use arrows to move, type to filter]
> Login with a web browser
  Paste an authentication token
```

Рис. 3.19: Настройка gh

```
Press Enter to open browser in your browser...
Gtk-Message: 01:53:01.985: Failed to load module "can
Gtk-Message: 01:53:01.992: Failed to load module "can
✓ Authentication complete. Press Enter to continue..
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as MaximSidorenkoNB12
```

Рис. 3.20: Настройка gh

- 12) Создал репозиторий курса на основе шаблона <https://github.com/yamadharm/course-directory-student-template>

```
masidorenko@masidorenko:~$ mkdir -p ~/work/study/2022-2023/"Операционные системы"
masidorenko@masidorenko:~$ ls
snap  Видео  Загрузки  Музыка  'Рабочий стол'
work  Документы  Изображения  Общедоступные  Шаблоны
```

Рис. 3.21: Создание репозитория курса

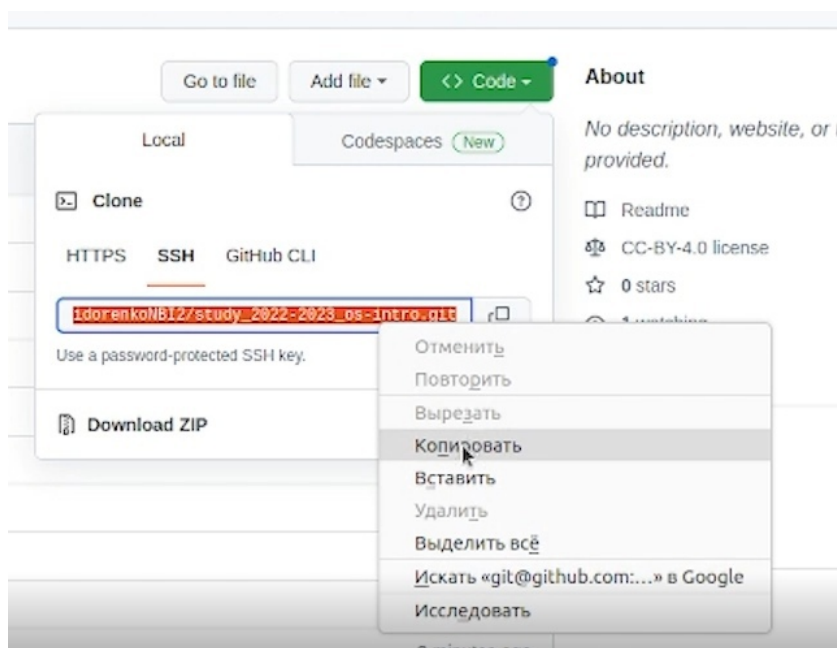


Рис. 3.22: Создание репозитория курса

```
masidorenko@masidorenko:~/work/study/2022-2023/Операционные системы$ git clone --recursive git@github.com:MaximSidorenkoNB12/study_2022-2023_os-intro.git
Клонирование в «study_2022-2023_os-intro»...
```

Рис. 3.23: Создание репозитория курса взятие ключа ssh

- 13) Настроил каталог курса, а именно перешёл в каталог курса, Удалите лишние файлы, Создал необходимые каталоги, отправил файлы на сервер

```
masidorenkor@masidorenkor:~$ cd ~/work/study/2022-2023/"Операционные системы"/study_2022-2023_os-intro
masidorenkor@masidorenkor:~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro$ rm package.json
```

Рис. 3.24: перешел и удалил лишние файлы

```
os-intro$ echo os-intro > COURSE
masidorenkor@masidorenkor:~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro$ make
```

Рис. 3.25: Создал необходимые каталоги

```
masidorenkor@masidorenkor:~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro$ git add .
masidorenkor@masidorenkor:~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro$ git commit -am 'feat(main): make course structure'
```

Рис. 3.26: Отправка файлов на сервер

```
masidorenkor@masidorenkor:~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro$ git push
```

Рис. 3.27: Отправка файлов на сервер

14) Проверил целостность репозитория

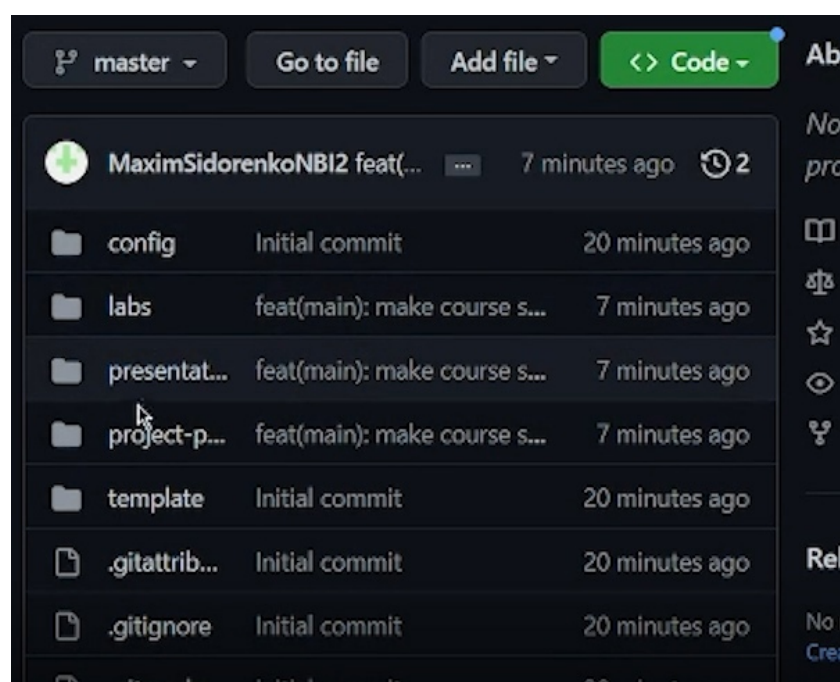


Рис. 3.28: Проверка



## 4 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.
4. Опишите действия с VCS при единоличной работе с хранилищем.
5. Опишите порядок работы с общим хранилищем VCS.
6. Каковы основные задачи, решаемые инструментальным средством git?
7. Назовите и дайте краткую характеристику командам git.
8. Приведите примеры использования при работе с локальным и удалённым репозиториями.
9. Что такое и зачем могут быть нужны ветви (branches)?
10. Как и зачем можно игнорировать некоторые файлы при commit?



## 5 Ответы на вопросы

- 1) 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) 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 ревизии. Она также записывает идентификатор пользователя, текущее время и временную зону, плюс список изменённых 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) 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. Децентрализованные не имеют определенный сервер и владельца
- 4) 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) 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 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 сохра-

нение 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) 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 ограничить 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) 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 файла2. 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) 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) 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 `bzi` 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~ ~ $
```



## 6 Вывод

После проделанной данной работы я изучил идеологию и применение средств контроля версий. Освоил умения по работе с git.