Отчёт по лабораторной работе №1. Управление версиями.

Предмет: научное программирование

Александр Сергеевич Баклашов

Содержание

List of Figures

1 Цель работы

– Изучить идеологию и применение средств контроля версий.

– Освоить умения по работе с git. [1]

2 Теоретическое введение

Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется. В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию изменённых файлов, а производить так называемую дельтакомпрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В

зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом. Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить. В отличие от классических, в распределённых системах контроля версий центральный репозиторий не является обязательным. Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд. [1]

3 Выполнение лабораторной работы

3.1 Установка gh в Fedora Linux

1. Установим gh в Fedora Linux (рис. 3.1)

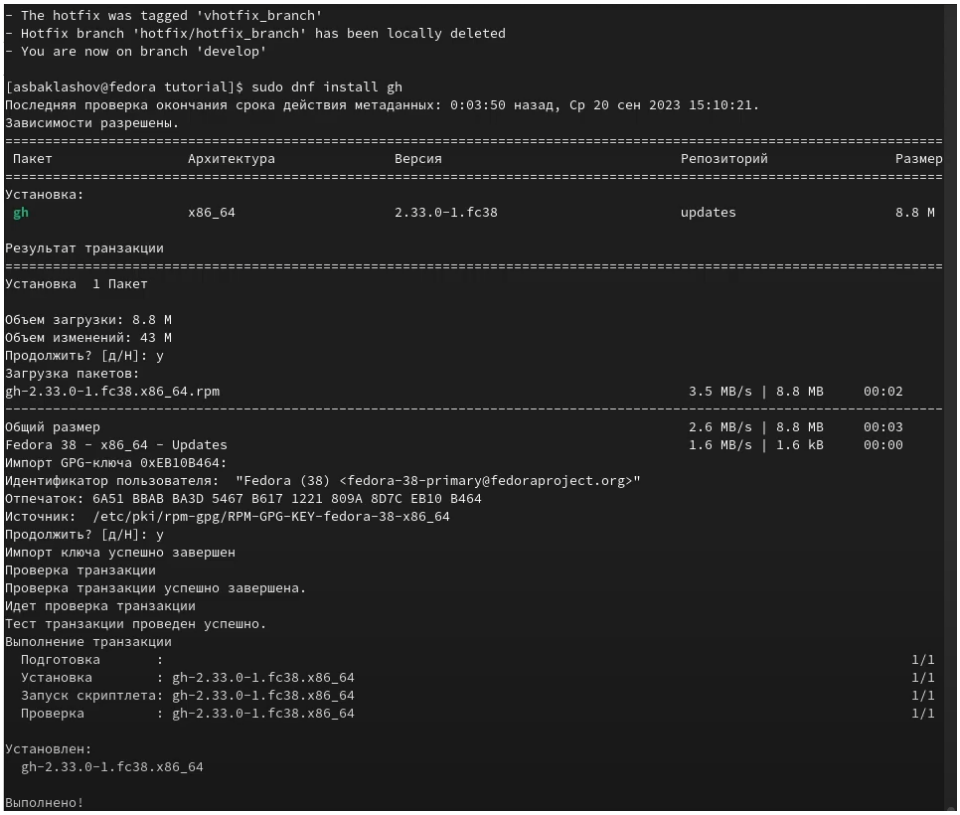

Figure 3.1: Установка gh

3.2 Базовая настройка git

2. Зададим имя и email владельца репозитория (рис. 3.2)

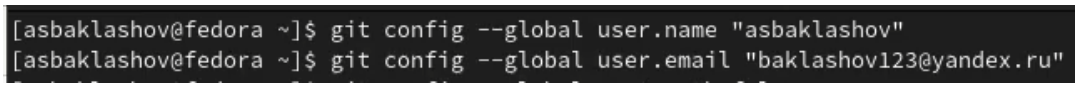

Figure 3.2: Имя и email

3. Зададим имя начальной ветки, параметр autocrlf, параметр safecrlf. (рис. 3.3)

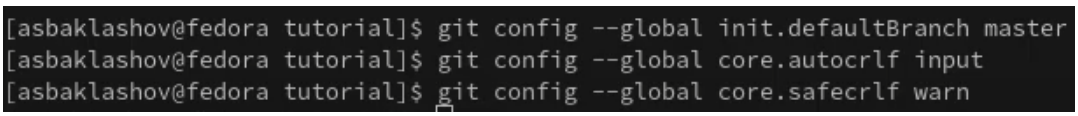

Figure 3.3: Задача параметров

3.3 Создание ключей ssh

4. Создадим ключ ssh по алгоритму rsa с ключём размером 4096 бит. (рис. 3.4)

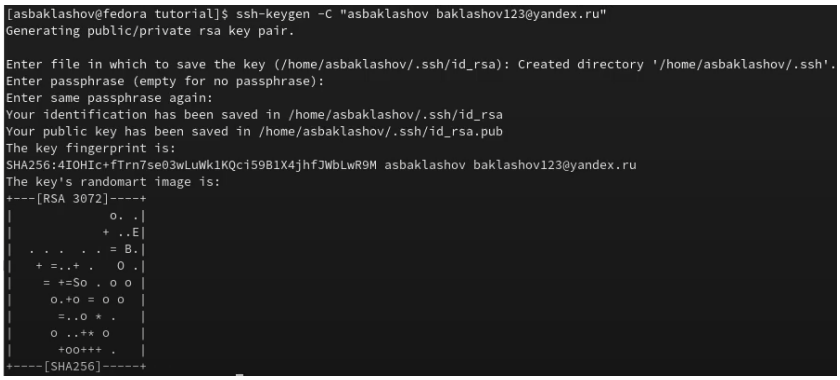

Figure 3.4: ssh rsa

5. Создадим ключ ssh по алгоритму ed25519 (рис. 3.5)

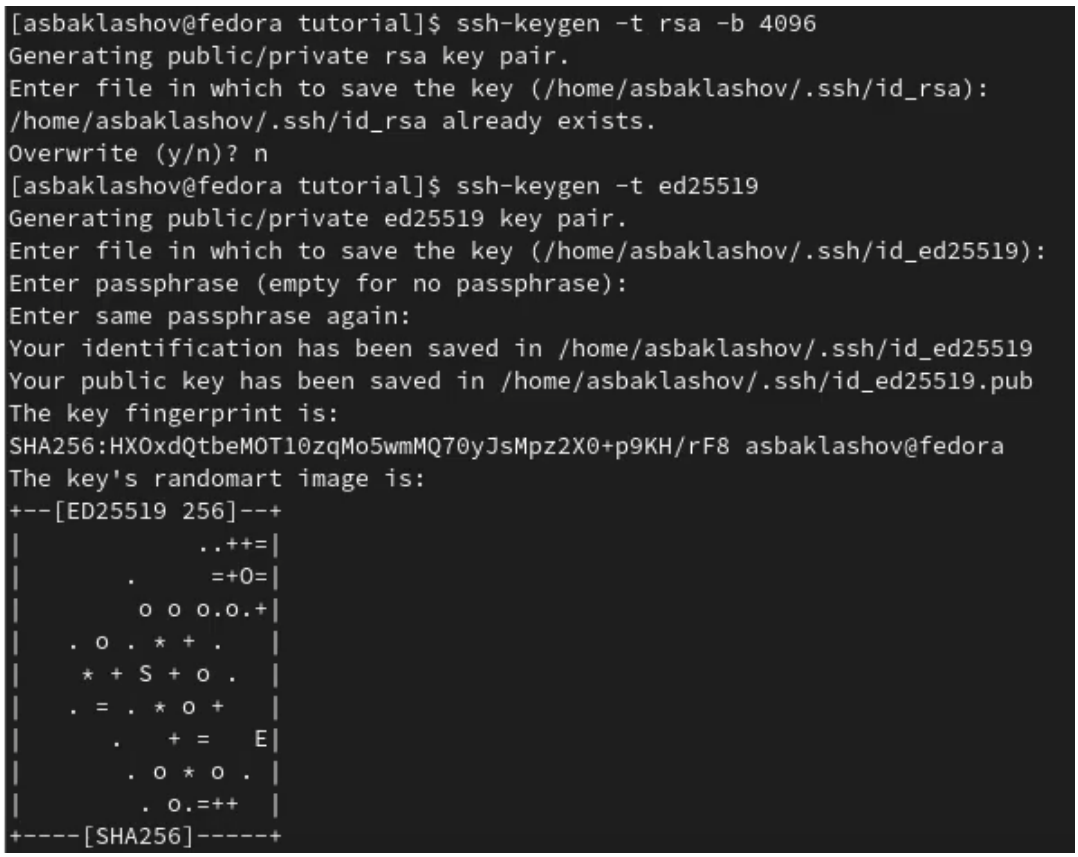

Figure 3.5: ssh ed25519

3.4 Создание ключа GPG

6. Сгенерируем ключ с определёнными параметрами (рис. 3.6)

```
[asbaklashov@fedora tutorial]$ gpg --full-generate-key
gpg (GnuPG) 2.4.0; 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.

gpg: создан каталог '/home/asbaklashov/.gnupg'
gpg: создан щит с ключами '/home/asbaklashov/.gnupg/pubring.kbx'
Выберите тип ключа:
   (1) RSA and RSA
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
   (9) ECC (sign and encrypt) *default*
  (10) ECC (только для подписи)
  (14) Existing key from card
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: asbaklashov
Адрес электронной почты: baklashov123@yandex.ru
Примечание:
Вы выбрали следующий идентификатор пользователя:
    "asbaklashov <baklashov123@yandex.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.
gpg: /home/asbaklashov/.gnupg/trustdb.gpg: создана таблица доверия
gpg: создан каталог '/home/asbaklashov/.gnupg/openpgp-revocs.d'
gpg: сертификат отзыва записан в '/home/asbaklashov/.gnupg/openpgp-revocs.d/ECC4FE491E0501C5137C09C927FFD23FC0645AB9.r
ev'.
открытый и секретный ключи созданы и подписаны.

pub   rsa4096 2023-09-20 [SC]
      ECC4FE491E0501C5137C09C927FFD23FC0645AB9
uid                      asbaklashov <baklashov123@yandex.ru>
sub   rsa4096 2023-09-20 [E]

[asbaklashov@fedora tutorial]$ 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:   1  подписанных:   0  доверие: 0-, 0q, 0n, 0m, 0f, 1u
/home/asbaklashov/.gnupg/pubring.kbx
------------------------------------
sec   rsa4096/27FFD23FC0645AB9 2023-09-20 [SC]
      ECC4FE491E0501C5137C09C927FFD23FC0645AB9
uid              [  абсолютно ] asbaklashov <baklashov123@yandex.ru>
ssb   rsa4096/B3D03979F42FA72E 2023-09-20 [E]
```

Figure 3.6: Ключ GPG

3.5 Добавление GPG ключа в GitHub

7. Выведем список ключей и копируем отпечаток приватного ключа (рис. 3.7)

```
[asbaklashov@fedora tutorial]$ 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:   1  подписанных:   0  доверие: 0-, 0q, 0n, 0m, 0f, 1u
/home/asbaklashov/.gnupg/pubring.kbx
------------------------------------
sec   rsa4096/27FFD23FC0645AB9 2023-09-20 [SC]
      ECC4FE491E0501C5137C09C927FFD23FC0645AB9
uid                 [  абсолютно ] asbaklashov <baklashov123@yandex.ru>
ssb   rsa4096/B3D03979F42FA72E 2023-09-20 [E]
```

Figure 3.7: GPG

8. Скопируем сгенерированный GPG ключ в буфер обмена (рис. 3.8)

```
[asbaklashov@fedora tutorial]$ gpg --armor --export 27FFD23FC0645AB9 | xclip -sel clip
```

Figure 3.8: Копирование GPG

9. Перейдём в настройки GitHub, нажмём на кнопку New GPG key и вставим полученный ключ в поле ввода. (рис. 3.9)

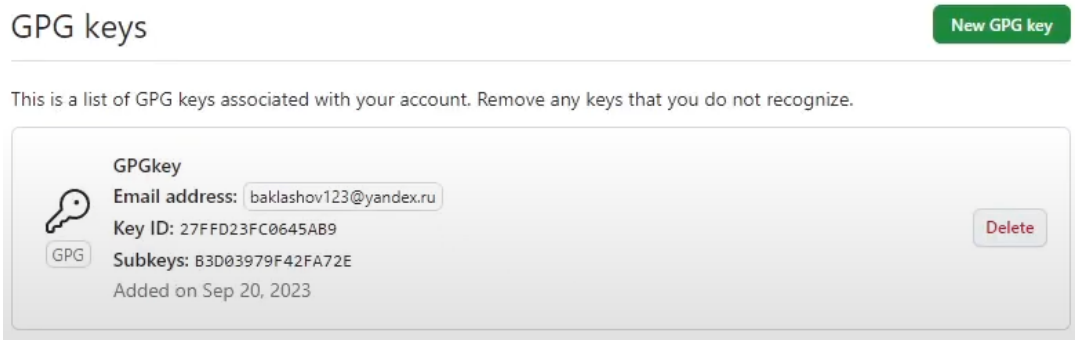

Figure 3.9: Добавление GPG

3.6 Настройка автоматических подписей коммитов git

10. Используя введённый email, укажем Git применять его при подписи коммитов (рис. 3.10)

```
[asbaklashov@fedora tutorial]$ git config --global user.signingkey 27FFD23FC0645AB9
[asbaklashov@fedora tutorial]$ git config --global commit.gpgsign true
[asbaklashov@fedora tutorial]$ git config --global gpg.program $(which gpg2)
```

Figure 3.10: Подписи

3.7 Настройка gh

11. Совершим настройку gh (рис. 3.11)

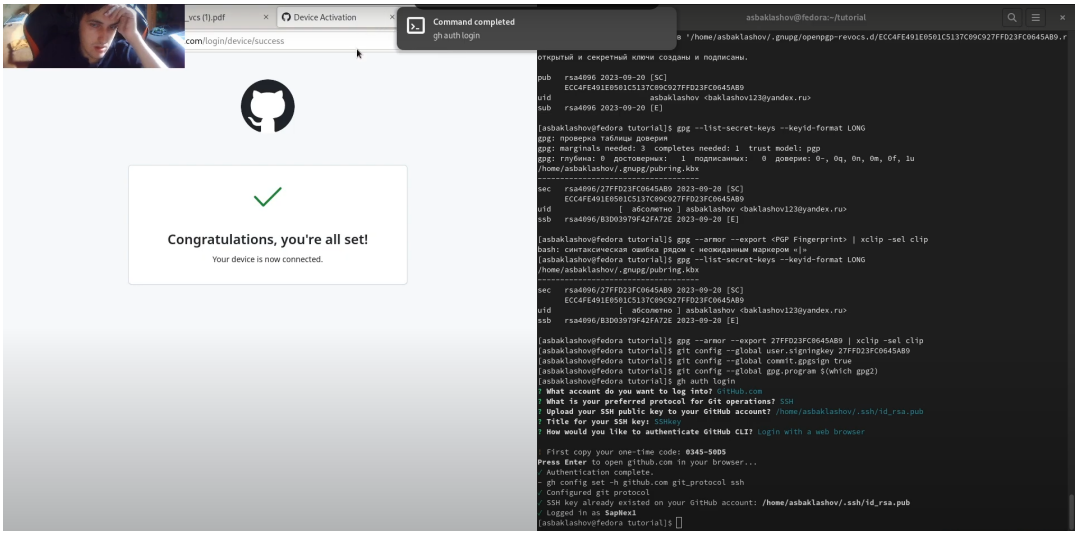

Figure 3.11: gh

3.8 Шаблон для рабочего пространства

12. Создадим репозиторий курса на основе шаблона (рис. 3.12)

```
[asbaklashov@fedora SciProg]$ git clone --recursive git@github.com:SapNex1/study_2021-2022_os-intro.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 27 (delta 1), reused 11 (delta 0), pack-reused 0
Получение объектов: 100% (27/27), 16.93 КиБ | 22.00 КиБ/с, готово.
Определение изменений: 100% (1/1), готово.
Подмодуль «template/presentation» (https://github.com/yamadharma/academic-presentation-markdown-template.git) зарегист
рирован по пути «template/presentation»
Подмодуль «template/report» (https://github.com/yamadharma/academic-laboratory-report-template.git) зарегистрирован по
 пути «template/report»
Клонирование в «/home/asbaklashov/work/study/2023-2024/SciProg/os-intro/template/presentation»...
remote: Enumerating objects: 82, done.
remote: Counting objects: 100% (82/82), done.
remote: Compressing objects: 100% (57/57), done.
remote: Total 82 (delta 28), reused 77 (delta 23), pack-reused 0
Получение объектов: 100% (82/82), 92.90 КиБ | 51.00 КиБ/с, готово.
Определение изменений: 100% (28/28), готово.
Клонирование в «/home/asbaklashov/work/study/2023-2024/SciProg/os-intro/template/report»...
remote: Enumerating objects: 101, done.
remote: Counting objects: 100% (101/101), done.
remote: Compressing objects: 100% (70/70), done.
remote: Total 101 (delta 40), reused 88 (delta 27), pack-reused 0
Получение объектов: 100% (101/101), 327.25 КиБ | 198.00 КиБ/с, готово.
Определение изменений: 100% (40/40), готово.
Submodule path 'template/presentation': checked out 'b1be3800ee91f5809264cb755d316174540b753e'
Submodule path 'template/report': checked out '1d1b61dcac9c287a83917b82e3aef11a33b1e3b2'
```

Figure 3.12: Репозиторий

13. Настроим каталог курса (рис. 3.13)

```
[asbaklashov@fedora SciProg]$ cd os-intro/
[asbaklashov@fedora os-intro]$ rm package.json
[asbaklashov@fedora os-intro]$ make COURSE=os-intro
bash: make: команда не найдена...
Установить пакет «make», предоставляющий команду «make»? [N/y] y


 * Ожидание в очереди...
 * Загрузка списка пакетов....
Следующие пакеты должны быть установлены:
 gc-8.2.2-3.fc38.x86_64 Garbage collector for C and C++
 guile22-2.2.7-7.fc38.x86_64    A GNU implementation of Scheme for application extensibility
 make-1:4.4.1-1.fc38.x86_64     A GNU tool which simplifies the build process for users
Продолжить с этими изменениями? [N/y] y


 * Ожидание в очереди...
 * Ожидание аутентификации...
 * Ожидание в очереди...
 * Загрузка пакетов...
 * Запрос данных...
 * Проверка изменений...
 * Установка пакетов...

[asbaklashov@fedora os-intro]$ make COURSE=os-intro
```

Figure 3.13: Каталог

14. Отправим файлы на сервер (рис. 3.14)

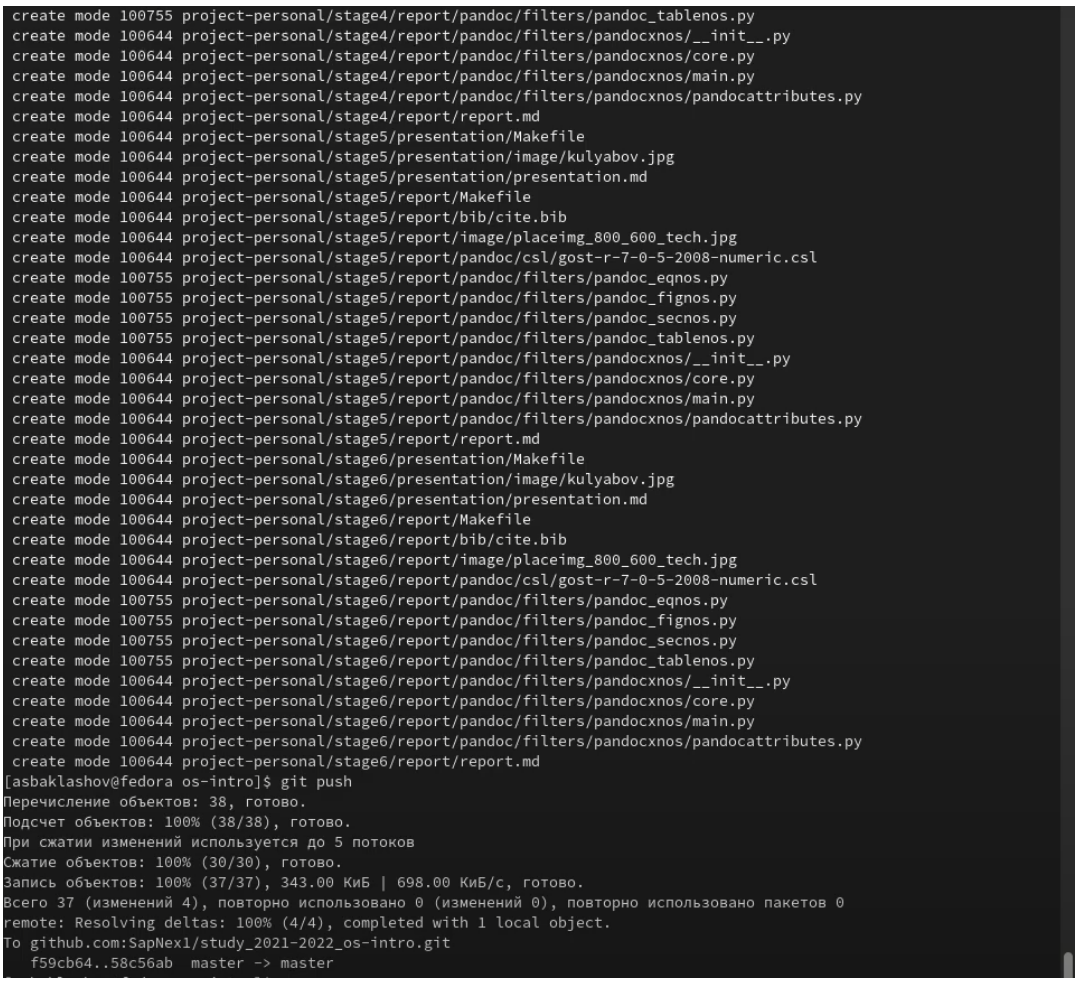

Figure 3.14: Отправка файлов

4 Вывод

В ходе данной лабораторной работы я изучил идеологию и применение средств контроля версий, а также освоил умения по работе с git.

5 Контрольные вопросы

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

Системы контроля версий (Version Control Systems, VCS), также известные как системы управления версиями, представляют собой программные инструменты и методологии, которые используются для управления изменениями в коде, документах и других файловых ресурсах в течение времени. Они предназначены для следующих задач:

Отслеживание изменений: Системы контроля версий позволяют пользователям записывать изменения, внесенные в файлы, вместе с комментариями, которые описывают характер изменений. Это позволяет легко понимать, что и когда было изменено.

Совместная работа: VCS обеспечивают средства для совместной работы нескольких разработчиков над одним проектом. Они могут одновременно вносить изменения в файлы и объединять их в общий репозиторий.

История и восстановление: VCS сохраняют историю изменений, что позволяет восстанавливать предыдущие версии файлов и даже возвращаться к состоянию проекта на более ранние этапы его развития.

Ответвление и слияние (Branching and Merging): VCS позволяют создавать отдельные ветки (branches) проекта, чтобы работать над новыми функциями или исправлениями ошибок без влияния на основную версию. Позже эти ветки могут быть слияны (merged) в основную ветку.

Отслеживание авторства и ответственности: VCS фиксируют, кто и когда внес

изменения в файлы, что помогает отслеживать авторство и ответственность за код.

Резервное копирование и восстановление данных: VCS предоставляют средства для создания резервных копий проекта, что обеспечивает защиту данных от потери или повреждения.

Ревизии и метки: Важными элементами VCS являются ревизии и метки. Ревизии представляют собой номера, присвоенные определенным состояниям репозитория, позволяя быстро находить нужную версию. Метки (tags) используются для пометки определенных моментов в истории, таких как релизы.

Работа с несколькими репозиториями: VCS также позволяют работать с несколькими удаленными репозиториями, что полезно для совместной разработки, открытого исходного кода и интеграции с другими инструментами и платформами.

Системы контроля версий, такие как Git, Subversion (SVN), Mercurial, и другие, широко используются в различных областях, включая разработку программного обеспечения, управление проектами, написание документации и многие другие, где важно отслеживание и управление изменениями в файловых ресурсах.

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

5.0.1 Хранилище (Repository):

Определение: Хранилище (репозиторий) в системе контроля версий (VCS) представляет собой центральное место, где хранится вся история изменений проекта и текущее состояние файлов.

Роль: Основная роль хранилища - сохранение истории изменений, отслеживание версий файлов, и предоставление возможности для совместной работы над проектом. Хранилище может быть локальным (на вашем компьютере), удаленным (на сервере), или распределенным (комбинация локального и удаленного).

5.0.2 Commit (Коммит):

Определение: Коммит в VCS представляет собой акт сохранения изменений в репозитории. Это фиксирует текущее состояние файлов в проекте.

Роль: Коммиты используются для отслеживания изменений в проекте и создания точек в истории. Они содержат описание изменений и могут включать в себя измененные, добавленные или удаленные файлы.

5.0.3 История (History):

Определение: История в VCS представляет собой запись всех прошлых коммитов и изменений, сделанных в проекте. Она включает в себя информацию о том, кто, когда и что изменил в проекте.

Роль: История позволяет просматривать, анализировать и возвращаться к предыдущим версиям проекта. Это полезно для отслеживания эволюции проекта, поиска ошибок и управления версиями.

5.0.4 Рабочая копия (Working Copy):

Определение: Рабочая копия в VCS представляет собой локальную копию проекта, с которой вы работаете на своем компьютере. Она содержит файлы проекта в их текущем состоянии.

Роль: Рабочая копия позволяет вам вносить изменения в файлы, просматривать текущее состояние проекта и подготавливать коммиты. Она обновляется из репозитория при получении новых изменений и сохраняется в репозиторий при выполнении коммитов.

Отношения между этими понятиями следующие: Рабочая копия представляет собой вашу локальную рабочую среду, где вы вносите изменения в файлы проекта. Вы создаете коммиты (commit) для фиксации изменений в вашей рабочей копии. Эти коммиты сохраняются в репозитории (repository) и составляют историю изменений проекта. Таким образом, рабочая копия, коммиты и история

взаимосвязаны и образуют основу работы с системой контроля версий.

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

Централизованные и децентрализованные системы контроля версий (VCS) представляют разные методологии управления версиями и совместной работой.

Вот их основные характеристики и различия:

5.0.5 Централизованные системы контроля версий (Centralized VCS):

Архитектура: В централизованных VCS существует один центральный сервер, на котором хранится вся история и версии проекта. Клиенты взаимодействуют с этим центральным сервером для доступа к файлам и истории.

Работа в изоляции: Когда разработчик хочет внести изменения, он создает копию (чаще называемую рабочей копией) из центрального репозитория, работает над изменениями и отправляет их обратно на сервер.

Примеры: Примерами централизованных систем контроля версий являются Subversion (SVN) и CVS (Concurrent Versions System).

5.0.6 Децентрализованные системы контроля версий (Distributed VCS):

Архитектура: В децентрализованных VCS каждый разработчик имеет свой собственный локальный репозиторий, который содержит всю историю проекта. Эти локальные репозитории могут взаимодействовать друг с другом, обмениваясь изменениями.

Работа в изоляции: Каждый разработчик может работать над своими изменениями в локальном репозитории, фиксировать их и отправлять на удаленные

репозитории, когда это необходимо. Это позволяет разработчикам работать в изоляции и независимо друг от друга.

Примеры: Примерами децентрализованных систем контроля версий являются Git и Mercurial.

Различия:

Архитектура: Основное различие заключается в архитектуре. В централизованных системах существует один центральный сервер, а в децентрализованных - множество локальных репозиториев.

Работа в изоляции: Децентрализованные системы позволяют разработчикам работать над изменениями в изоляции, не завися от доступности центрального сервера, что делает их более гибкими.

Отсутствие единой точки отказа: В децентрализованных системах отсутствует единая точка отказа, так как каждый разработчик имеет полную копию истории проекта.

Скорость и эффективность: Децентрализованные системы, как Git, часто считаются более быстрыми и эффективными при обмене данными и работе с историей проекта.

Примеры децентрализованных систем контроля версий включают Git и Mercurial, а примеры централизованных систем контроля версий включают Subversion (SVN) и CVS. Каждая из них имеет свои преимущества и недостатки, и выбор зависит от конкретных потребностей проекта и команды разработчиков.

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

Если вы работаете с системой контроля версий (VCS) в одиночку, то процесс работы будет более простым и менее зависящим от взаимодействия с другими разработчиками. Вот основные действия, которые вы выполняете при единоличной работе с хранилищем VCS:

4.1. Инициализация репозитория:

Создайте новый репозиторий или склонируйте существующий, если вы начинаете работу с существующим проектом. Создание рабочей копии (Working

Copy):

После инициализации репозитория у вас появляется локальная рабочая копия, которая является вашей средой для работы над проектом.

4.2. Добавление файлов и изменений:

Добавьте файлы и директории в вашу рабочую копию, которые вы хотите отслеживать и контролировать с помощью VCS. Вносите изменения в файлы, как обычно, с учетом вашего процесса разработки.

4.3. Фиксация изменений (Commit):

Периодически фиксируйте изменения, которые вы сделали в вашей рабочей копии, с помощью коммитов. Каждый коммит создает запись о состоянии проекта в текущий момент. Просмотр истории и переход к предыдущим версиям:

Вы можете просматривать историю ваших коммитов и переходить к предыдущим версиям проекта, если вам это нужно.

4.4. Работа в изоляции:

Вы можете продолжать работать в изоляции, не взаимодействуя с другими разработчиками. Все ваши коммиты сохраняются локально в вашей рабочей копии.

4.5. Работа с ветками (по желанию):

Если нужно, вы можете создавать и переключаться между ветками, чтобы разрабатывать разные функции или исправлять ошибки в изоляции.

4.6. Подготовка к резервному копированию (Backup):

Поскольку ваша рабочая копия и история хранятся локально, важно регулярно создавать резервные копии репозитория для защиты от потери данных. Отправка изменений на удаленный репозиторий (по желанию):

Если у вас есть удаленный репозиторий (например, на GitHub или GitLab), вы можете отправлять изменения на него, чтобы создать резервную копию проекта и/или совместно работать с другими разработчиками.

4.7. Соблюдение лучших практик VCS:

Вы можете использовать стандартные практики VCS, такие как хорошие ком-

ментарии к коммитам, создание веток для новых функций и т. д., даже при индивидуальной работе.

При единоличной работе с VCS, вы все равно можете получить множество преимуществ, таких как история изменений, возможность отката к предыдущим версиям, защита данных и удобное отслеживание прогресса проекта. Эти преимущества могут быть особенно полезными, если у вас есть несколько проектов или если вы хотите сотрудничать с другими разработчиками в будущем.

5. Опишите порядок работы с общим хранилищем VCS.

Порядок работы с общим хранилищем VCS включает в себя:

- Инициализацию или клонирование репозитория.
- Создание и переключение веток (по желанию).
- Внесение и изменение файлов.
- Фиксацию изменений (коммиты) с описанием.
- Получение изменений из центрального репозитория.
- Отправку изменений на сервер.
- Решение конфликтов (по необходимости).
- Соблюдение структуры и стандартов.
- Обсуждение и обратная связь (по желанию).
- Анализ истории и версий.
- Резервное копирование и безопасность.
- Завершение и управление проектом.
- Совместную работу и обмен знаниями (по желанию).

Этот порядок обеспечивает совместную разработку и управление версиями проекта в команде разработчиков.

6. Каковы основные задачи, решаемые инструментальным средством git?

Основные задачи, решаемые инструментом Git:

Управление версиями: Отслеживание изменений в файлах и создание точек в истории проекта.

Совместная разработка: Возможность совместной работы нескольких разработчиков над одним проектом.

Отслеживание изменений: Просмотр и анализ истории изменений, включая авторство и временные метки.

Управление конфликтами: Разрешение конфликтов при слиянии изменений от разных разработчиков.

Резервное копирование: Создание резервных копий и обеспечение безопасности данных проекта.

Ветвление и слияние: Создание отдельных веток для разработки новых функций и их последующее слияние в основную ветку.

Удобство и эффективность: Улучшение процесса разработки и управления проектами.

Сотрудничество и обмен знаниями: Совместная работа над проектами и обмен опытом и знаниями в команде разработчиков.

7. Назовите и дайте краткую характеристику командам git.

git init: Инициализирует новый локальный репозиторий Git в текущей директории.

git clone: Клонирует удаленный репозиторий Git на локальный компьютер.

git add: Добавляет изменения из рабочей копии в индекс для подготовки к коммиту.

git commit: Создает новый коммит, фиксируя текущее состояние проекта в истории.

git pull: Получает изменения с удаленного репозитория и автоматически сливает их с текущей веткой.

git push: Отправляет локальные изменения на удаленный репозиторий.

git branch: Показывает список веток и создает новые ветки.

git checkout: Переключает текущую ветку на другую или создает новую ветку.

git merge: Сливает изменения из одной ветки в другую.

git status: Показывает статус текущей ветки и изменения в рабочей копии.

git log: Отображает историю коммитов.

git diff: Показывает разницу между версиями файлов или коммитами.

git stash: Скрывает текущие изменения для временной работы на другой ветке.

git remote: Управляет удаленными репозиториями.

git fetch: Получает изменения из удаленного репозитория без их автоматического слияния.

git revert: Создает новый коммит, отменяющий изменения из предыдущего коммита.

git reset: Отменяет коммиты и изменения в истории репозитория.

git rm: Удаляет файлы из индекса и рабочей копии.

git tag: Управляет метками (тегами) для пометки коммитов.

git config: Устанавливает настройки Git, такие как имя пользователя и адрес электронной почты.

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

Инициализация локального репозитория:

git init

Клонирование удаленного репозитория:

git clone https://github.com/user/repo.git

Добавление и фиксация изменений:

git add file.txt

git commit -m “Добавил новый файл”

Получение изменений с удаленного репозитория:

git pull origin main

Отправка локальных изменений на удаленный репозиторий:

git push origin main

Просмотр истории коммитов:

git log

Создание и переключение веток:

git branch feature-branch git checkout feature-branch

Слияние изменений из одной ветки в другую:

git checkout main git merge feature-branch

Создание и переключение на новую ветку без коммита:

git checkout -b new-branch

Удаление файлов из индекса и рабочей копии:

git rm file.txt

Откат к предыдущему коммиту:

git reset –hard HEAD^

Создание и переключение на тег (метку):

git tag v1.0

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

Ветви (branches) в системе управления версиями, такой как Git, представляют собой параллельные линии разработки, которые позволяют разработчикам работать над различными функциональными частями проекта независимо друг от друга.

Они могут быть нужны для:

Изоляции задач: Разработчики могут работать над разными задачами или функциями в отдельных ветвях, не мешая друг другу.

Параллельной разработки: Ветви позволяют одновременно разрабатывать и тестировать новые функции или исправления ошибок.

Тестирования: Ветви позволяют создавать изолированные среды для тестирования, не затрагивая основной код.

Управления версиями: Каждая ветвь может представлять новую версию проекта или стабильную ветвь для релизов.

Совместной разработки: Ветви облегчают совместную работу нескольких разработчиков над одним проектом.

Экспериментов: Ветви позволяют проводить эксперименты и не беспокоиться о влиянии на основной код.

Таким образом, ветви обеспечивают гибкость и эффективность в управлении разработкой проекта, позволяя разработчикам работать более организованно и безопасно.

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

Вы можете игнорировать некоторые файлы при коммите в Git с помощью файла .gitignore. Этот файл содержит шаблоны для файлов и директорий, которые вы хотите исключить из контроля версий. Зачем это полезно:

Исключение временных и сгенерированных файлов: Например, вы можете игнорировать файлы, создаваемые вашей средой разработки или временные файлы, чтобы не засорять репозиторий.

Игнорирование настроек и конфиденциальных данных: Вы можете исключить файлы с настройками, содержащими пароли, ключи API и другие конфиденциальные данные, чтобы они не попали в репозиторий.

Уменьшение объема репозитория: Игнорирование больших бинарных файлов, медиафайлов или других данных, которые не должны храниться в репозитории, помогает уменьшить его размер.

Предотвращение конфликтов: Игнорирование файлов, которые могут изменяться разными способами на разных компьютерах, помогает предотвратить конфликты при слиянии.

Чтобы использовать .gitignore, создайте файл .gitignore в корневой директории вашего репозитория и добавьте в него шаблоны файлов и директорий, которые вы хотите игнорировать.

После того как .gitignore создан и настроен, Git будет игнорировать указанные файлы и директории при коммите.

6 Библиография

1. Лабораторная работа №1. Управление версиями. - 12 с. [Электронный ресурс]. М. URL: Лабораторная работа №1. Управление версиями. (Дата обращения: 20.09.2023).