Отчет по лабораторной работе №2

Операционные системы

Дмитрий Павлович Стрижов

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Задание

1. Создать базовую конфигурацию для работы с git.
2. Создать ключ SSH.
3. Создать ключ PGP.
4. Настроить подписи git.
5. Создать локальный каталог для выполнения заданий по предмету.

3 Теоретическое введение

Системы контроля версий. Общие понятия

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 дельта-компрессию — 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.

Примеры использования git

```
Система контроля версий Git представляет собой набор программ командной строки. Д
Благодаря тому, что Git является распределённой системой контроля версий, резервн
```

Основные команды git

```
Перечислим наиболее часто используемые команды git.


Создание основного дерева репозитория:


git init
```

Получение обновлений (изменений) текущего дерева из центрального репозитория:

```
git pull
```

Отправка всех произведённых изменений локального дерева в центральный репозиторий

```
git push
```

Просмотр списка изменённых файлов в текущей директории:

```
git status
```

Просмотр текущих изменений:

```
git diff
```

Сохранение текущих изменений:

добавить все изменённые и/или созданные файлы и/или каталоги:

```
git add .
```

добавить конкретные изменённые и/или созданные файлы и/или каталоги:

```
git add имена_файлов
```

удалить файл и/или каталог из индекса репозитория (при этом файл и/или катало

```
git rm имена_файлов
```

Сохранение добавленных изменений:

сохранить все добавленные изменения и все изменённые файлы:

```
git commit -am 'Описание коммита'
```

сохранить добавленные изменения с внесением комментария через встроенный реда

```
git commit
```

создание новой ветки, базирующейся на текущей:

```
git checkout -b имя_ветки
```

переключение на некоторую ветку:

```
git checkout имя_ветки
```

(при переключении на ветку, которой ещё нет в локальном репозитории, она

отправка изменений конкретной ветки в центральный репозиторий:

```
git push origin имя_ветки
```

слияние ветки с текущим деревом:

```
git merge --no-ff имя_ветки
```

Удаление ветки:

удаление локальной уже слитой с основным деревом ветки:

```
git branch -d имя_ветки
```

принудительное удаление локальной ветки:

```
git branch -D имя_ветки
```

удаление ветки с центрального репозитория:

```
git push origin :имя_ветки
```

Стандартные процедуры работы при наличии центрального репозитория

Работа пользователя со своей веткой начинается с проверки и получения изменений и

```
git checkout master
git pull
git checkout -b имя_ветки
```

Затем можно вносить изменения в локальном дереве и/или ветке.

После завершения внесения какого-то изменения в файлы и/или каталоги проекта необ

```
git status
```

При необходимости удаляем лишние файлы, которые не хотим отправлять в центральный

Затем полезно просмотреть текст изменений на предмет соответствия правилам ведени

```
git diff
```

Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы, из

```
git add …
git rm …
```

Если нужно сохранить все изменения в текущем каталоге, то используем:

```
git add .
```

Затем сохраняем изменения, поясняя, что было сделано:

```
git commit -am "Some commit message"
```

Отправляем изменения в центральный репозиторий:

```
git push origin имя_ветки
```

или

```
git push
```

Работа с локальным репозиторием

Создадим локальный репозиторий.

Сначала сделаем предварительную конфигурацию, указав имя и email владельца репози

```
git config --global user.name "Имя Фамилия"
git config --global  user.email "work@mail"
```

Настроим utf-8 в выводе сообщений git:

```
git config --global quotepath false
```

Для инициализации локального репозитория, расположенного, например, в каталоге ~/

```
cd
mkdir tutorial
cd tutorial
git init
```

После это в каталоге tutorial появится каталог .git, в котором будет храниться ис

Создадим тестовый текстовый файл hello.txt и добавим его в локальный репозиторий:

```
echo 'hello world' > hello.txt
git add hello.txt
git commit -am 'Новый файл'
```

Воспользуемся командой status для просмотра изменений в рабочем каталоге, сделанн

```
git status
```

Во время работы над проектом так или иначе могут создаваться файлы, которые не тр

```
curl -L -s https://www.gitignore.io/api/list
```

Затем скачать шаблон, например, для C и C++

```
curl -L -s https://www.gitignore.io/api/c >> .gitignore
curl -L -s https://www.gitignore.io/api/c++ >> .gitignore
```

Работа с сервером репозиториев

Для последующей идентификации пользователя на сервере репозиториев необходимо сге

```
ssh-keygen -C "Имя Фамилия <work@mail>"
```

Ключи сохраняться в каталоге ~/.ssh/.
Существует несколько доступных серверов репозиториев с возможностью бесплатного р
Для работы с ним необходимо сначала завести на сайте https://github.com/ учётную
Для этого зайти на сайт https://github.com/ под своей учётной записью и перейти в

После этого выбрать в боковом меню GitHub setting>SSH-ключи и нажать кнопку Добав

```
cat ~/.ssh/id_rsa.pub | xclip -sel clip
```

Вставляем ключ в появившееся на сайте поле.

После этого можно создать на сайте репозиторий, выбрав в меню , дать ему название

Для загрузки репозитория из локального каталога на сервер выполняем следующие ком

```
git remote add origin
  ssh://git@github.com/<username>/<reponame>.git
```

```
git push -u origin master
```

Далее на локальном компьютере можно выполнять стандартные процедуры для работы с

Базовая настройка git

Первичная настройка параметров git

Зададим имя и email владельца репозитория:

```
git config --global user.name "Name Surname"
git config --global user.email "work@mail"
```

Настроим utf-8 в выводе сообщений git:

```
git config --global core.quotepath false
```

Настройте верификацию и подписание коммитов git.

Зададим имя начальной ветки (будем называть её master):

```
git config --global init.defaultBranch master
```

Учёт переносов строк

В разных операционных системах приняты разные символы для перевода строк:

- Windows: \r\n (CR и LF);
- Unix: \n (LF);
- Mac: \r (CR).

Посмотреть значения переносов строк в репозитории можно командой:

```
git ls-files --eol
```

Параметр autocrlf

Настройка core.autocrlf предназначена для того, чтобы в главном репозитории в

Настройка core.autocrlf с параметрами true и input делает все переводы строк

core.autocrlf true: конвертация CRLF->LF при коммите и обратно LF->CRLF п

core.autocrlf input: конвертация CRLF->LF только при коммитах (используют

Варианты конвертации

Таблица 1.: Варианты конвертации для разных значений параметра core.autocrlf

git commit	LF -> LF	LF -> LF	LF -> CRLF
	CR -> CR	CR -> CR	CR -> CR
	CRLF -> CRLF	CRLF -> LF	CRLF -> CRLF
git checkout	LF -> LF	LF -> LF	LF -> CRLF
	CR -> CR	CR -> CR	CR -> CR
	CRLF -> CRLF	CRLF -> CRLF	CRLF -> CRLF

Установка параметра:

Для Windows

```
git config --global core.autocrlf true
```

Для Linux

```
git config --global core.autocrlf input
```

Параметр safecrlf

Настройка core.safecrlf предназначена для проверки, является ли окончаний стр

core.safecrlf true: запрещается необратимое преобразование lf<->crlf. Пол

core.safecrlf warn: печать предупреждения, но коммиты с необратимым перех

Установка параметра:

```
git config --global core.safecrlf warn
```

Создание ключа ssh

Общая информация

Алгоритмы шифрования ssh

Аутентификация

В SSH поддерживается четыре алгоритма аутентификации по открытым ключам:

DSA:

размер ключей DSA не может превышать 1024, его следует отключить;

RSA:

следует создавать ключ большого размера: 4096 бит;

ECDSA:

ECDSA завязан на технологиях NIST, его следует отключить;

Ed25519:

используется пока не везде.

Симметричные шифры

Из 15 поддерживаемых в SSH алгоритмов симметричного шифрования, безопасны

chacha20-poly1305;

aes*-ctr;

aes*-gcm.

Шифры 3des-cbc и arcfour потенциально уязвимы в силу использования DES и

Шифр cast128-cbc применяет слишком короткий размер блока (64 бит).

Обмен ключами

Применяемые в SSH методы обмена ключей DH (Diffie-Hellman) и ECDH (Ellipt
Hellman) можно считать безопасными.

Из 8 поддерживаемых в SSH протоколов обмена ключами вызывают подозрения т

ecdh-sha2-nistp256;

ecdh-sha2-nistp384;

ecdh-sha2-nistp521.

Не стоит использовать протоколы, основанные на SHA1.

Файлы ssh-ключей

По умолчанию пользовательские ssh-ключи сохраняются в каталоге ~/.ssh в домаш

Убедитесь, что у вас ещё нет ключа.

Файлы закрытых ключей имеют названия типа id_<алгоритм> (например, id_dsa, id

По умолчанию закрытые ключи имеют имена:

id_dsa

id_ecdsa

id_ed25519

id_rsa

Открытые ключи имеют дополнительные расширения .pub.

По умолчанию публичные ключи имеют имена:

id_dsa.pub

id_ecdsa.pub

id_ed25519.pub

id_rsa.pub

При создании ключа команда попросит ввести любую ключевую фразу для более над

Сменить пароль на ключ можно с помощью команды:

ssh-keygen -p

Создание ключа ssh

Ключ ssh создаётся командой:

ssh-keygen -t <алгоритм>

Создайте ключи:

по алгоритму rsa с ключём размером 4096 бит:

ssh-keygen -t rsa -b 4096

по алгоритму ed25519:

ssh-keygen -t ed25519

При создании ключа команда попросит ввести любую ключевую фразу для более надёжно

Сменить пароль на ключ можно с помощью команды:

ssh-keygen -p

Добавление SSH-ключа в учётную запись GitHub

Скопируйте созданный SSH-ключ в буфер обмена командой:

```
xclip -i < ~/.ssh/id_ed25519.pub
```

Откройте настройки своего аккаунта на GitHub и перейдем в раздел SSH and GPC keys
Нажмите кнопку ew SSH key.
Добавьте в поле Title название этого ключа, например, ed25519@hostname.
Вставьте из буфера обмена в поле Key ключ.
Нажмите кнопку Add SSH key.

Верификация коммитов с помощью PGP

Как настроить PGP-подпись коммитов с помощью gpg.

Общая информация

Коммиты имеют следующие свойства:
    author (автор) — контрибьютор, выполнивший работу (указывается для справки);
    committer (коммитер) — пользователь, который закоммитил изменения.
Эти свойства можно переопределить при совершении коммита.
Авторство коммита можно подделать.
В git есть функция подписи коммитов.
Для подписывания коммитов используется технология PGP (см. Работа с PGP).
Подпись коммита позволяет удостовериться в том, кто является коммитером. Авторств

Создание ключа

Генерируем ключ

```
gpg --full-generate-key
```

Из предложенных опций выбираем:

тип RSA and RSA;

размер 4096;

выберите срок действия; значение по умолчанию — 0 (срок действия не истекает

GPG запросит личную информацию, которая сохранится в ключе:

Имя (не менее 5 символов).

Адрес электронной почты.

При вводе email убедитесь, что он соответствует адресу, используемому на

Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить

Экспорт ключа

Выводим список ключей и копируем отпечаток приватного ключа:

```
gpg --list-secret-keys --keyid-format LONG
```

Отпечаток ключа — это последовательность байтов, используемая для идентификации б

Формат строки:

```
sec   Алгоритм/Отпечаток_ключа Дата_создания [Флаги] [Годен_до]
      ID_ключа
```

Экспортируем ключ в формате ASCII по его отпечатку:

```
gpg --armor --export <PGP Fingerprint>
```

Добавление PGP ключа в GitHub

Копируем ключ и добавляем его в настройках профиля на GitHub (или GitLab).

Скопируйте ваш сгенерированный PGP ключ в буфер обмена:

```
gpg --armor --export <PGP Fingerprint> | xclip -sel clip
```

Перейдите в настройки GitHub (https://github.com/settings/keys), нажмите на кнопк

Подписывание коммитов git

Подпись коммитов при работе через терминал:

```
git commit -a -S -m 'your commit message'
```

Флаг -S означает создание подписанного коммита. При этом может потребоваться ввод
ключа.

Настройка автоматических подписей коммитов git

Используя введёный email, укажите Git применять его при подписи коммитов:

```
git config --global user.signingkey <PGP Fingerprint>
git config --global commit.gpgsign true
git config --global gpg.program $(which gpg2)
```

Проверка коммитов в Git

GitHub и GitLab будут показывать значок Verified рядом с вашими новыми коммитами.

Режим бдительности (vigilant mode)

На GitHub есть настройка vigilant mode.
Все неподписанные коммиты будут явно помечены как Unverified.
Включается это в настройках в разделе SSH and GPG keys. Установите метку на Flag

4 Выполнение лабораторной работы

4.1 Создание базовой конфигурации для работы с git

Задаем имя и почту пользователя(рис. 4.1).

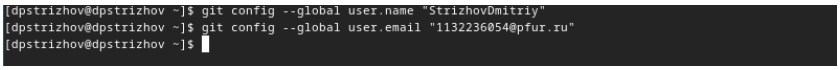

Рис. 4.1: Имя и почта пользователя

Настраиваем прочие настройки(utf-8, autocrlf, safecrlf, имя начальной ветки) (рис. 4.2).

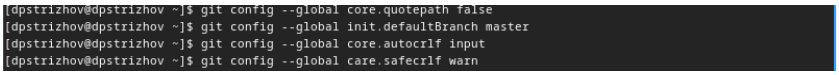

Рис. 4.2: Прочие настройки

4.2 Создание ключа SSH

Создаем ключ SSH по алгоритмам rsa размером 4096 и ed25519 (рис. 4.3, 4.4).

Рис. 4.3: Создание ключа SSH по алгоритму rsa размером 4096

Рис. 4.4: Создание ключа SSH по алгоритму ed25519

4.3 Создание ключа PGP

Создаем ключ PGP (рис. 4.5).

Рис. 4.5: Генерация ключа

Выводи список ключей и копируем отпечаток приватного ключа (рис. 4.6).

Рис. 4.6: Вывод списка ключей

Копируем ключ в буфер обмена (рис. 4.7).

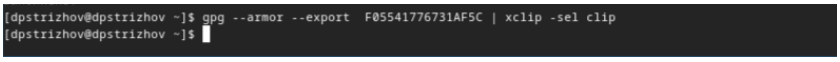

Рис. 4.7: Копирование ключа

Добавляем ключ на github (рис. 4.8).

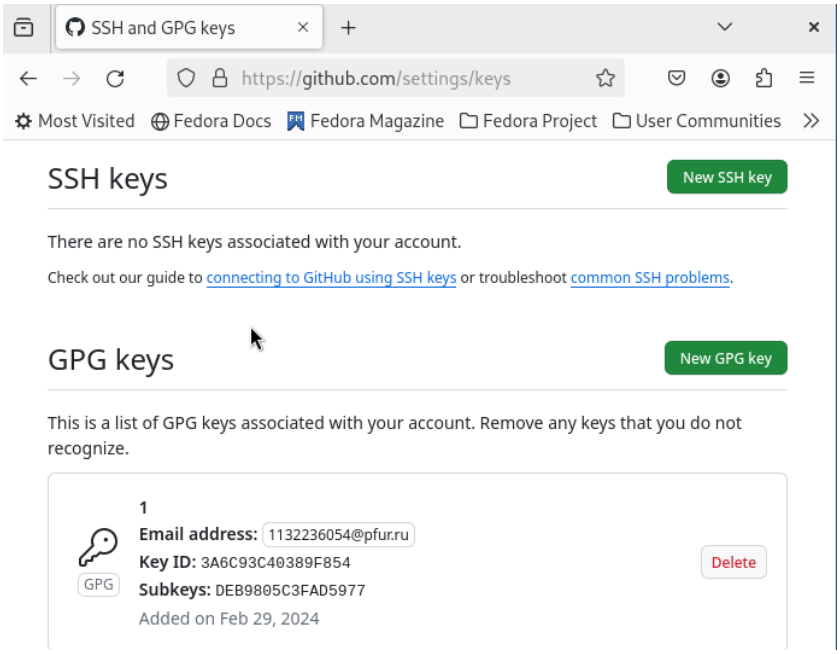

Рис. 4.8: Ключ на github

4.4 Настройка подписей git

Настраиваем подписи (рис. 4.9).

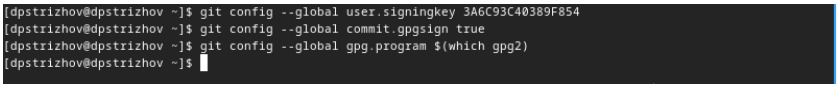

Рис. 4.9: Настройка подписей

4.5 Создание локального каталога для выполнения заданий по предмету

Заходим в github (рис. 4.10).

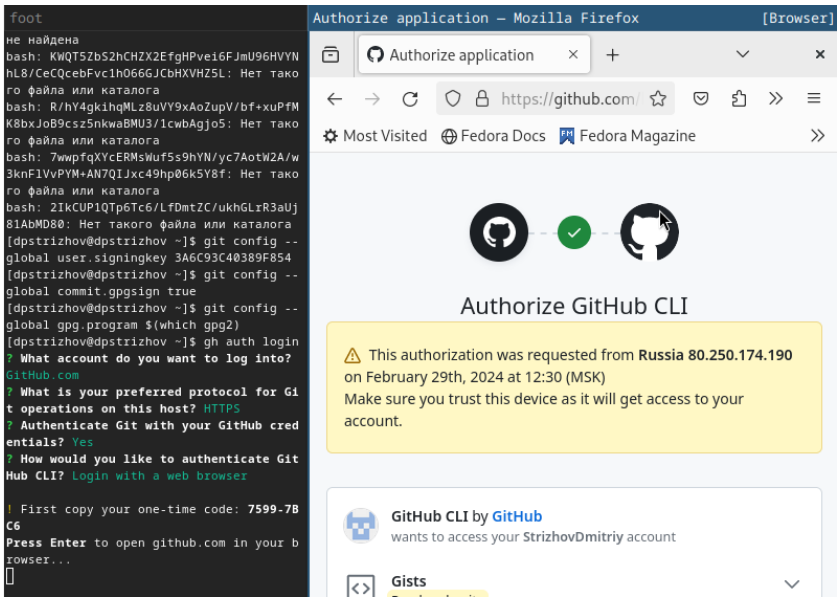

Рис. 4.10: Авторизация в github

Создаем шаблон для рабочего пространства(рис. 4.11, 4.12, 4.13).

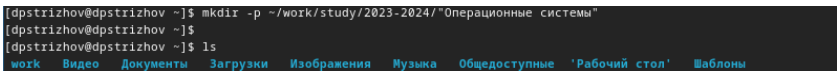

Рис. 4.11: Создание необходимых каталогов

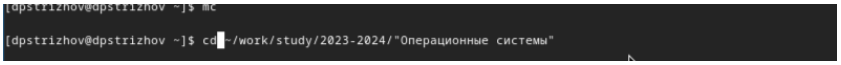

Рис. 4.12: Переход в нужный католог

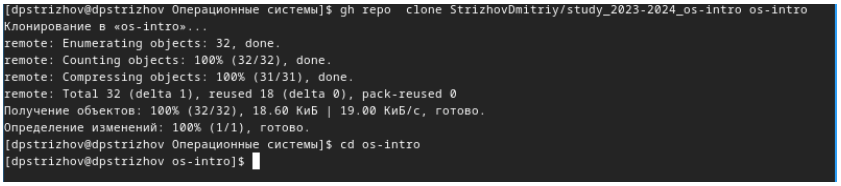

Рис. 4.13: Копирование репозитория

Удаляем ненужные файлы (рис. 4.14).

Рис. 4.14: Удаление ненужных файлов

Создаем необходимые катологи (рис. 4.15).

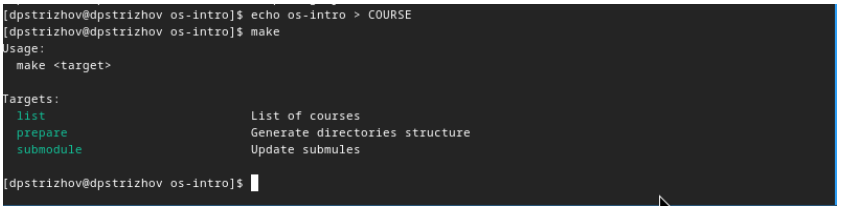

Рис. 4.15: Создание необходимых катологов

Отправляем на github (рис. 4.16).

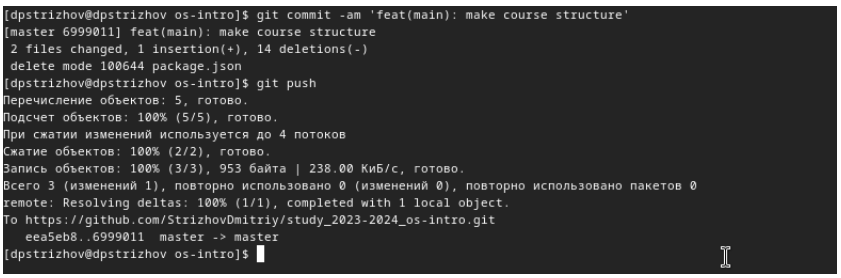

Рис. 4.16: Отправлка на github

5 Выводы

За время выполнения лабораторной работы я изучил идеологию и применение средств контроля версий, а также освоил умения по работе git.

Список литературы

Введение в Git: настройка и основные команды. Ссылка: https://selectel.ru/blog/tutorials/git-setup-and-common-commands/