

Лабораторная работа №2

Дисциплина - операционные системы

Волгин Иван Алексеевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Выводы	14
6	Контрольные вопросы	15

Список иллюстраций

4.1	Установка git и gh	8
4.2	Задаем имя и email владельца репозитория и настраиваем utf-8	8
4.3	Задаем имя начальной ветки и параметры autocrlf и safecrlf.	9
4.4	Создаем ключ ssh по алгоритму rsa.	9
4.5	Создаем ssh ключ по алгоритму ed25519.	9
4.6	Создание ключа pgr.	10
4.7	Личная информация, запрошенная GPG.	10
4.8	Выводим список ключей в терминал.	11
4.9	Копируем ключ PGP в буфер обмена.	11
4.10	Вставляем PGP ключ на GitHub.	11
4.11	Настройка автоматических подписей git.	11
4.12	Авторизуемся и отвечаем на вопросы.	12
4.13	Авторизуемся через броузер.	12
4.14	Создаем директорию и переходим в нужный файл.	13
4.15	Создаем на гитхабе репозиторий на основе шаблона.	13
4.16	Клонируем репозиторий себе в систему.	13

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 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 отличие от классических, в распределённых системах контроля версий центральный репозиторий не является обязательным.

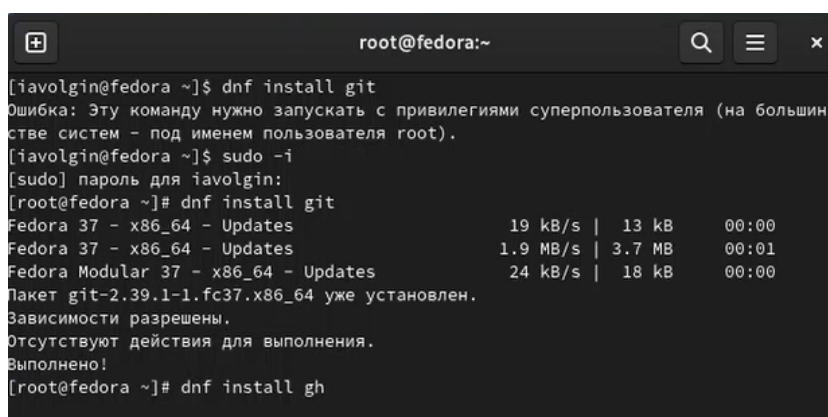
Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд.

Примеры использования git

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

4 Выполнение лабораторной работы

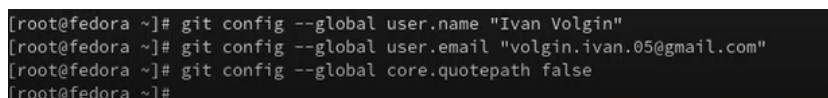
Для начала устанавливаем git и gh с помощью команд `dnf install git`, `dnf install gh` (рис. 4.1).



```
root@fedora:~  
[iavolgin@fedora ~]$ dnf install git  
Ошибка: Эту команду нужно запускать с привилегиями суперпользователя (на большин  
стве систем - под именем пользователя root).  
[iavolgin@fedora ~]$ sudo -i  
[sudo] пароль для iavolgin:  
[root@fedora ~]# dnf install git  
Fedora 37 - x86_64 - Updates          19 kB/s | 13 kB      00:00  
Fedora 37 - x86_64 - Updates          1.9 MB/s | 3.7 MB     00:01  
Fedora Modular 37 - x86_64 - Updates 24 kB/s | 18 kB      00:00  
Пакет git-2.39.1-1.fc37.x86_64 уже установлен.  
Зависимости разрешены.  
Отсутствуют действия для выполнения.  
Выполнено!  
[root@fedora ~]# dnf install gh
```

Рис. 4.1: Установка git и gh

Далее нам нужно задать имя (`git config --global user.name "Name Surname"`) и email (`git config --global user.email "work@mail"`) владельца репозитория и настроить utf-8 в выводе сообщений git (`git config --global core.quotePath false`) (рис. 4.2)



```
[root@fedora ~]# git config --global user.name "Ivan Volgin"  
[root@fedora ~]# git config --global user.email "volgin.ivan.05@gmail.com"  
[root@fedora ~]# git config --global core.quotePath false  
[root@fedora ~]#
```

Рис. 4.2: Задаем имя и email владельца репозитория и настраиваем utf-8

Затем задаем имя начальной ветки (будем называть её master) (`git config --global`

init.defaultBranch master), параметр autocrlf (git config –global core.autocrlf input) и параметр safecrlf (git config –global core.safecrlf warn) (рис. 4.3)

```
[root@fedora ~]# git config --global init.defaultBranch master
[root@fedora ~]# git config --global core.autocrlf input
[root@fedora ~]# git config --global core.safecrlf warn
```

Рис. 4.3: Задаем имя начальной ветки и параметры autocrlf и safecrlf.

Создаем ключи ssh. Первый по алгоритму rsa с ключом размером 4096 бит (ssh-keygen -t rsa -b 4096) (рис. 4.4), и второй по алгоритму ed25519 (ssh-keygen -t ed25519) (рис. 4.5)

```
[root@fedora ~]# ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
```

Рис. 4.4: Создаем ключ ssh по алгоритму rsa.

```
[root@fedora ~]# ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
```

Рис. 4.5: Создаем ssh ключ по алгоритму ed25519.

Далее генерируем ключ pgr (gpg –full-generate-key) и из предложенных опций выбираем тип RSA and RSA, размер 4096 и бесконечный срок действия (рис. 4.6)

```
root@fedora:~  
[root@fedora ~]# gpg --full-generate-key  
gpg (GnuPG) 2.3.8; 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.  
  
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)
```

Рис. 4.6: Создание ключа gpg.

Далее GPG запросит информацию, которая будет храниться в ключе: имя, адрес электронной почты и комментарий(рис. 4.7)

```
Ваше полное имя: Volgi Ivan Alekseevich  
Адрес электронной почты: volgin.ivan.05@gmail.com  
Примечание:  
Вы выбрали следующий идентификатор пользователя:  
  "Volgi Ivan Alekseevich <volgin.ivan.05@gmail.com>"  
Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход? O
```

Рис. 4.7: Личная информация, запрошенная GPG.

Далее нам нужно добавить PGP ключ в GitHub. Для этого выводим список ключей в терминал (`gpg --list-secret-keys --keyid-format LONG`) (рис. 4.8) и копируем PGP ключ в буфер обмена (`gpg --armor --export | xclip -sel clip`) (рис. 4.9). Затем переходим в настройки GitHub, нажимаем кнопку New PGP key и вставляем полученный ключ в поле ввода (рис. 4.10)

```
[root@fedora ~]# 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
/root/.gnupg/pubring.kbx
-----
sec   rsa4096/F294E77B30BA86A8 2023-02-16 [SC]
      B7724396E534D6AFEB812FF1F294E77B30BA86A8
uid           [ абсолютно ] Volgi Ivan Alekseevich <volgin.ivan.05@gmail.co
m>
ssb   rsa4096/7A57985FA041C61A 2023-02-16 [E]

[root@fedora ~]#
```

Рис. 4.8: Выводим список ключей в терминал.

```
[root@fedora ~]# gpg --armor --export F294E77B30BA86A8 | xclip -sel clip
[root@fedora ~]#
```

Рис. 4.9: Копируем ключ PGP в буфер обмена.

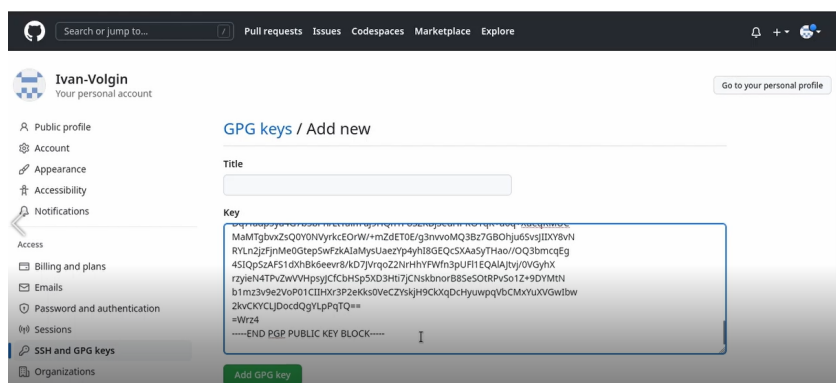


Рис. 4.10: Вставляем PGP ключ на GitHub.

а затем настраиваем автоматические подписи коммитов git (рис. 4.11). `git config --global user.signingkey git config --global commit.gpgsign true git config --global gpg.program $(which gpg2)`

```
root@fedora:~
[root@fedora ~]# git config --global user.signkey F294E77B30BA86A8
[root@fedora ~]# git config --global commit.gpgsign true
[root@fedora ~]# git config --global gpg.program $(which gpg2)
[root@fedora ~]#
```

Рис. 4.11: Настройка автоматических подписей git.

Далее мы должны настроить gh. Для начала авторизовываемся (gh auth login) (рис. 4.12). Затем система задает несколько наводящих вопросов и авторизуемся через браузер (рис. 4.13).

```
[iavolgin@fedora ~]$ 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. 4.12: Авторизуемся и отвечаем на вопросы.

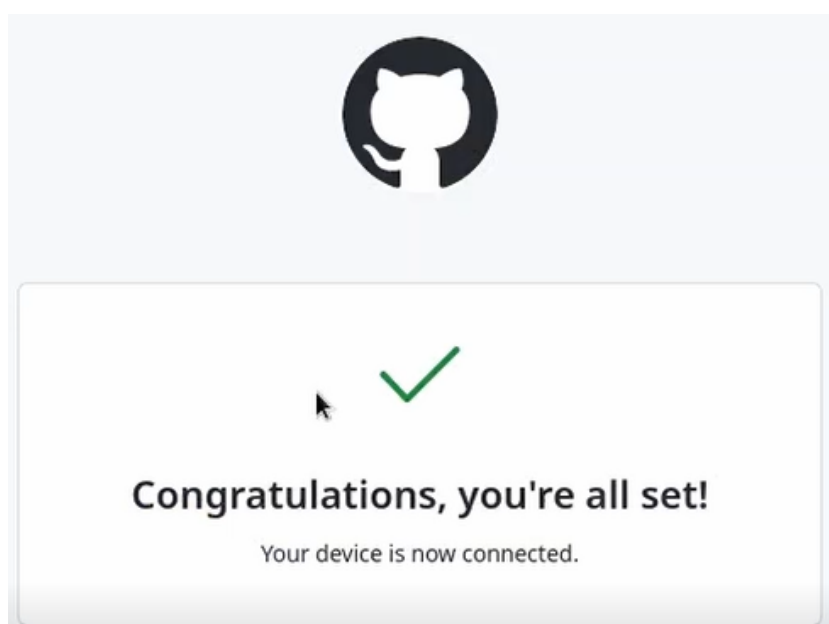


Рис. 4.13: Авторизуемся через браузер.

После этого создаём репозиторий курса на основе шаблона. `mkdir -p ~/work/study/2022-2023/“Операционные системы”` (создаем в файловой системе директорию) (рис. 4.14) `cd ~/work/study/2022-2023/“Операционные системы”` (переходим в файл “Операционные системы”) (рис. 4.14)

`gh repo create study_2022-2023_os-intro --template=yamadharma/course-directory-student-template --public` (создаем на гитхабе репозиторий на основе шаблона)

(рис. 4.15).

`git clone --recursive git@github.com:/study_2022-2023_os-intro.git os-intro` (клонировем репозиторий себе в систему) (рис. 4.16).

```
[iavolgin@fedora ~]$ mkdir -p ~/work/study/2022-2023/"Операционные системы"
[iavolgin@fedora ~]$ cd ~/work/study/2022-2023/"Операционные системы"
[iavolgin@fedora Операционные системы]$
```

Рис. 4.14: Создаем директорию и переходим в нужный файл.

```
[iavolgin@fedora Операционные системы]$ gh repo create study_2022-2023_os-intro
--template=yamadharma/course-directory-student-template --public
✓ Created repository Ivan-Volgin/study_2022-2023_os-intro on GitHub
[iavolgin@fedora Операционные системы]$
```

Рис. 4.15: Создаем на гитхабе репозиторий на основе шаблона.

```
[root@fedora ~]# git clone --recursive https://github.com/Ivan-Volgin/study_2022-2023_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 | 2.12 МБ/с, готово.
Определение изменений: 100% (1/1), готово.
Подмодуль «template/presentation» (https://github.com/yamadharma/academic-presentation-markdown-template.git) зарегистрирован по пути «template/presentation»
Подмодуль «template/report» (https://github.com/yamadharma/academic-laboratory-report-template.git) зарегистрирован по пути «template/report»
Клонирование в «/root/os-intro/template/presentation»...
remote: Enumerating objects: 82, done.
remote: Counting objects: 100% (82/82), done.
remote: Compressing objects: 100% (57/57), done.
```

Рис. 4.16: Клонировем репозиторий себе в систему.

Далее настраиваем каталог курса. Переходим в каталог (`cd ~/work/study/2022-2023/“Операционные системы”/os-intro`), удаляем лишние файлы (`rm package.json`), создаем необходимые каталоги (`echo os-intro > COURSE`, `make`) и отправляем все файлы на сервер (`git add .`, `git commit -am 'feat(main): make course structure'`, `git push`). Часть видео с выполнением этих команд у меня не записалась.

5 Выводы

В ходе выполнения лабораторной работы я изучил идеологию и применение средств контроля версий, так же освоил умения по работе с git.

6 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? Система управления версиями (VCS) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое. Применяются они для хранения полной истории изменений, совместной работы команды над одним проектом, хранения полной информации о каждом изменении (кто и когда).
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище – репозиторий, в котором хранятся все документы, история их изменений и прочая информация. Commit – отслеживание изменений, сохраняет разницу в изменениях. История – хранилище всех изменений, позволяющее в любой момент вернуться к прежней версии проекта. Рабочая копия – копия проекта, основанная на версии из хранилища, зачастую последней.
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. Централизованные VCS – одно основное хранилище проекта, из которого каждый пользователь может брать себе для работы нужную копию файлов и после их изменения возвращать обратно. Децентрализованные VCS – личный вариант репозитория каждого пользователя, есть возможность забирать и добавлять

изменения из любого репозитория.

4. Опишите действия с VCS при единоличной работе с хранилищем. Сначала подключается удаленный репозиторий. Затем вносятся изменения и отправляются обратно на сервер.
5. Опишите порядок работы с общим хранилищем VCS. Пользователь перед началом работы должен взять нужную ему версию проекта из хранилища, затем внести изменения и отправить их обратно на сервер, тем самым создав новую версию проекта. Старые версии тоже сохраняются.
6. Каковы основные задачи, решаемые инструментальным средством git? Хранение всех версий проекта, истории изменений и упрощение командной работы.
7. Назовите и дайте краткую характеристику командам 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 каталог остаётся в локальной директории): • `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 редактор: • `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 будет создана и связана с удалённой) • 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`

8. Приведите примеры использования при работе с локальным и удалённым репозиториями. При работе с локальным репозиторием нам может понадобиться получить обновления из центрального репозитория, тогда мы будем должны использовать команду `git pull`. Аналогично, чтобы внести уже проработанные изменения в удаленный репозиторий мы должны использовать команду `git push`.
9. Что такое и зачем могут быть нужны ветви (branches)? Каждая ветвь представляет собой полную копию материнской ветви, но изменения которые на ней происходят не отображаются на материнской ветви (потом можно провести слияние). Они нужны для удобства работы разных разработчиков или отделов разработки над одним проектом. Так они не мешают друг другу.
10. Как и зачем можно игнорировать некоторые файлы при commit? При работе могут создаваться файлы, которые не нужно отправлять в удаленный репозиторий. Чтобы избежать их попадания туда можно добавить шаблоны игнорируемых при добавлении в репозиторий типов файлов.