Лабораторная работа №2

Отчёт

Дмитриев Владимир Константинович

Содержание

# Цель работы

Изучить идеологию и применение средств контроля версий. Освоить умения по работе с git [@tuis]

# 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.

# Выполнение лабораторной работы

Для начала установим git. В моём случае он уже установлен (рис. [-@fig:001])

|  |
| --- |
| Установка git |

Установка git

Теперь установим gh (рис. [-@fig:002])

|  |
| --- |
| Установка gh |

Установка gh

Далее, зададим имя для владельца репозитория. В данном случае это моё имя (рис. [-@fig:003])

|  |
| --- |
| Указание имени |

Указание имени

Теперь зададим почту. Я задал почту, на которую у меня зарегистрирован аккаунт на github (рис. [-@fig:004])

|  |
| --- |
| Указание почты |

Указание почты

Настроим кодировку utf8 в выводе сообщений git (рис. [-@fig:005])

|  |
| --- |
| Настройка кодировки utf8 |

Настройка кодировки utf8

Зададим имя начальной ветки, настроим параметры autocrlf и safecrlf (рис. [-@fig:006])

|  |
| --- |
| Настройка git |

Настройка git

Создадим ключ RSA размером 4096 бит (рис. [-@fig:007])

|  |
| --- |
| Создание ключа RSA |

Создание ключа RSA

Теперь создадим ключ по алгоритму ed22519 (рис. [-@fig:008])

|  |
| --- |
| Создание ключа ed22519 |

Создание ключа ed22519

Теперь создадим ключ gpg. Выбираем из предложенных вариантов первый тип (RSA and RSA), размер ключа задаём 4096 бит и делаем срок действия ключа неограниченным (рис. [-@fig:009])

|  |
| --- |
| оздание ключа pgp (1) |

оздание ключа pgp (1)

После нас попросят ввести свои данные. Мы вводим имя и адрес электронной почты. После этого соглашаемся с генерацией ключа (рис. [-@fig:010])

|  |
| --- |
| оздание ключа pgp (2) |

оздание ключа pgp (2)

Далее, выводим список pgp ключей (рис. [-@fig:011])

|  |
| --- |
| Список pgp ключей |

Список pgp ключей

Копируем наш ключ в буфер обмена (рис. [-@fig:012])

|  |
| --- |
| Копирование ключа |

Копирование ключа

Вставляем этот ключ на гитхаб, и задаём ему имя. Я выбрал имя Sway (рис. [-@fig:013])

|  |
| --- |
| Вставка ключа в GitHub |

Вставка ключа в GitHub

Теперь производим настройку автоматических подписей (рис. [-@fig:014])

|  |
| --- |
| Настройка автоматических подписей коммитов git |

Настройка автоматических подписей коммитов git

После, нам нужно авторизоваться в github с помощью gh. Мы выбираем сайт для авторизации (GitHub.com), после выбираем предпочитаемый протокол (SSH), публичный SSH ключ (id\_rsa.pub), и имя для ключа (Sway). В качестве способа авторизации выбираем авторизацию через браузер (рис. [-@fig:015])

|  |
| --- |
| Авторизация в gh |

Авторизация в gh

Теперь создаём рабочую директорию курса и переходим в неё (рис. [-@fig:016])

|  |
| --- |
| Создание рабочей директории и переход в неё |

Создание рабочей директории и переход в неё

Далее, создаём репозиторий для лабораторных работ из шаблона (рис. [-@fig:017])

|  |
| --- |
| Сознание репозитория курса |

Сознание репозитория курса

И клонируем его к себе на компьютер (рис. [-@fig:018])

|  |
| --- |
| Клонирование репозитория |

Клонирование репозитория

Переходим в него с помощью cd и удаляем ненужные файлы (package.json) и создаём необходимые каталоги, записав в файл COURSE строку os-intro (это наш текущий курс) и прописываем make prepare для того, чтобы нужные нам каталоги создались (рис. [-@fig:019])

|  |
| --- |
| Удаление ненужных файлов и использование make |

Удаление ненужных файлов и использование make

Теперь добавляем нашу папку для отправки (рис. [-@fig:020])

|  |
| --- |
| Использование git add |

Использование git add

Делаем коммит, в котором указываем, что мы сделали структуру курса (рис. [-@fig:021])

|  |
| --- |
| Использование git commit |

Использование git commit

И отправляем файлы на сервер GitHub с помощью команды push (рис. [-@fig:022])

|  |
| --- |
| Использование git push |

Использование git push

# Выводы

Была произведена установка git, проведена его первоначальная настройка, были созданы ключи для авторизации и подписи, а также создан репозиторий курса из предложенного шаблона

# Ответы на контрольные вопросы

1. Системы контроля версий – это системы, в которых мы можем хранить свои проекты и выкладывать их обновления, контролируя релизы и каждые внесённые изменения. Эти системы нужны для работы над проектами, чтобы иметь возможность контролировать версии проектов и в случае командной работы контролировать изменения, внесённые всеми участниками. Также, VCS позволяют откатываться на более ранние версии
2. Хранилище – репозиторий, в нём хранятся все файлы проекта и все его версии  
   commit – внесённые изменения в репозитории  
   история – это история изменений файлов проекта  
   рабочая копия – копия, сделанная из версии репозитория, с которой непосредственно работает сам разработчик
3. Централизованные системы контроля версий имеют один центральный репозиторий, с которым работают все разработчики. Примером является CVS, который является уже устаревшей системой.  
   В децентрализованных системах же используется множество репозиториев одного проекта у каждого из разработчиков, при этом репозитории можно объединять брать из каждого только то, что нужно. Примером является знакомый нам Git
4. Создаётся репозиторий, и разрабатывается проект. При внесении изменений файлы отправляются на сервер
5. Разработчик клонирует репозиторий к себе на компьютер, и после внесения изменений выгружает их на сервер в качестве отдельной версии. После этого разработчики с более высокими правами могут, например, объединить его версию с текущей
6. Хранение файлов проекта, а также обеспечение командной работы, и контроль за версиями проекта
7. git clone – клонирует проект с сервера на компьютер  
   git add – добавляет папку для выгрузки на сервер  
   git commit – фиксирует изменения репозитория  
   git push – выгружает изменения на сервер  
   git pull – получить изменения с сервера  
   git rm – удалить файл  
   git status – получить статус репозитория
8. С локальным: git commit -am “added files” – создаёт коммит С удалённым: git push – загрузить данные на удалённый сервер
9. Ветки – это несколько независимых копий проекта, в каждой из которых ведётся разработка какой-то конкретной функции, при этом ветки существуют параллельно. Они нужны, когда нужно параллельно вести разработку нескольких функций, а в конце их можно объединить в одну
10. Игнорировать файлы можно, внося их в файл .gitignore. Игнорировать файлы нужно, когда их не нужно добавлять в репозиторий. Например, это могут быть файлы виртуального окружения (venv)

# Список литературы