Лабораторная работа № 2

Первоначальна настройка git.

Ермаков Алексей

Содержание

# 1 Цель работы

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

# 2 Задание

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 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. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями.  
Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.

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 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

Стандартные процедуры работы при наличии центрального репозитория

Работа пользователя со своей веткой начинается с проверки и получения изменений из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений):  
  
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 необходимо разместить их в центральном репозитории. Для этого необходимо проверить, какие файлы изменились к текущему моменту:  
  
git status  
  
При необходимости удаляем лишние файлы, которые не хотим отправлять в центральный репозиторий.  
  
Затем полезно просмотреть текст изменений на предмет соответствия правилам ведения чистых коммитов:  
  
git diff  
  
Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы, изменения которых нужно сохранить. Для этого используем команды добавления и/или удаления с нужными опциями:  
  
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 репозитория:  
  
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 ~/tutorial, необходимо ввести в командной строке:  
  
cd   
mkdir tutorial  
cd tutorial  
git init  
  
После это в каталоге tutorial появится каталог .git, в котором будет храниться история изменений.  
  
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, сделанных с момента последней ревизии:  
  
git status  
  
Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл .gitignore с помощью сервисов. Для этого сначала нужно получить список имеющихся шаблонов:  
  
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

# 4 Выполнение лабораторной работы

Установка программного обеспечения(рис. 1).

Установка git

Установим git:  
  
dnf install git

Установка gh

Fedora:  
  
dnf install gh

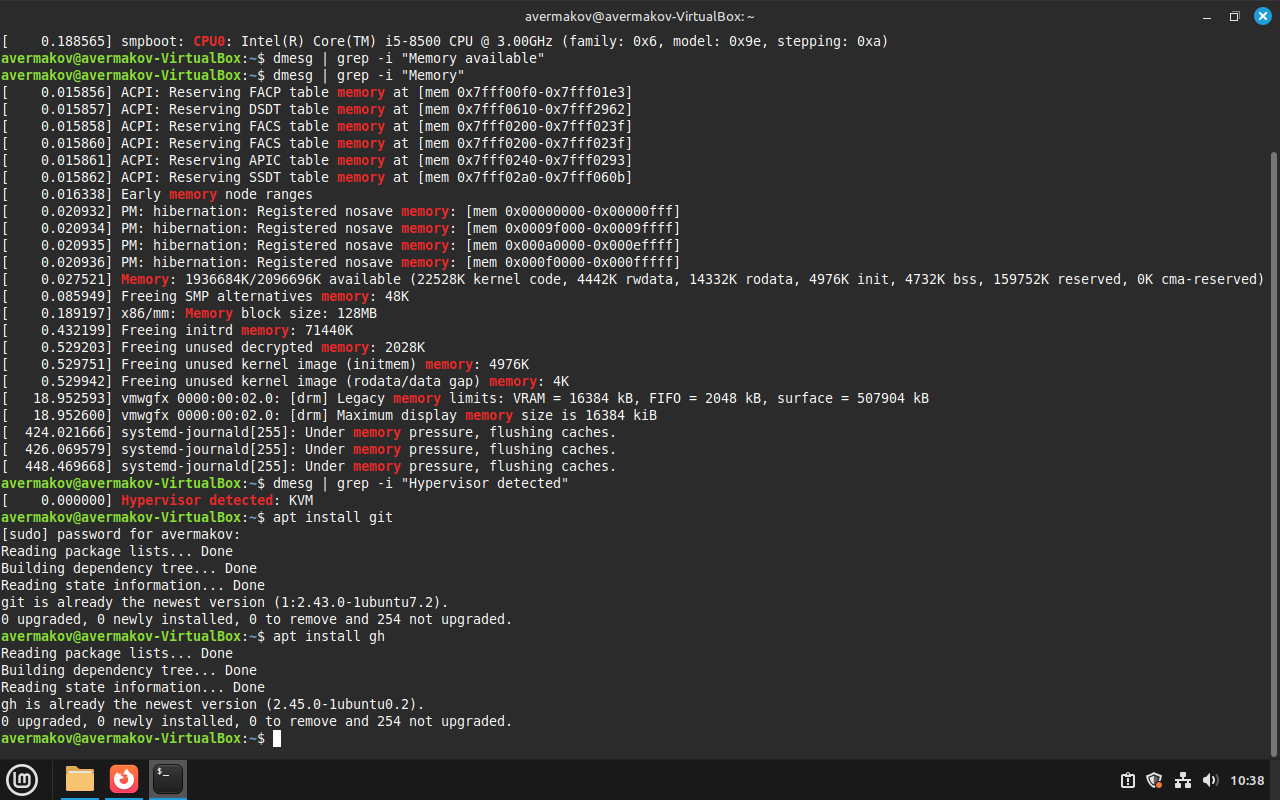


Рис. 1: Установка программного обеспечения git и gh

Базовая настройка git (рис. 2).

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 (см. Верификация коммитов git с помощью GPG).  
  
Зададим имя начальной ветки (будем называть её master):  
  
git config --global init.defaultBranch master  
  
Параметр autocrlf:  
  
git config --global core.autocrlf input  
  
Параметр safecrlf:  
  
git config --global core.safecrlf warn

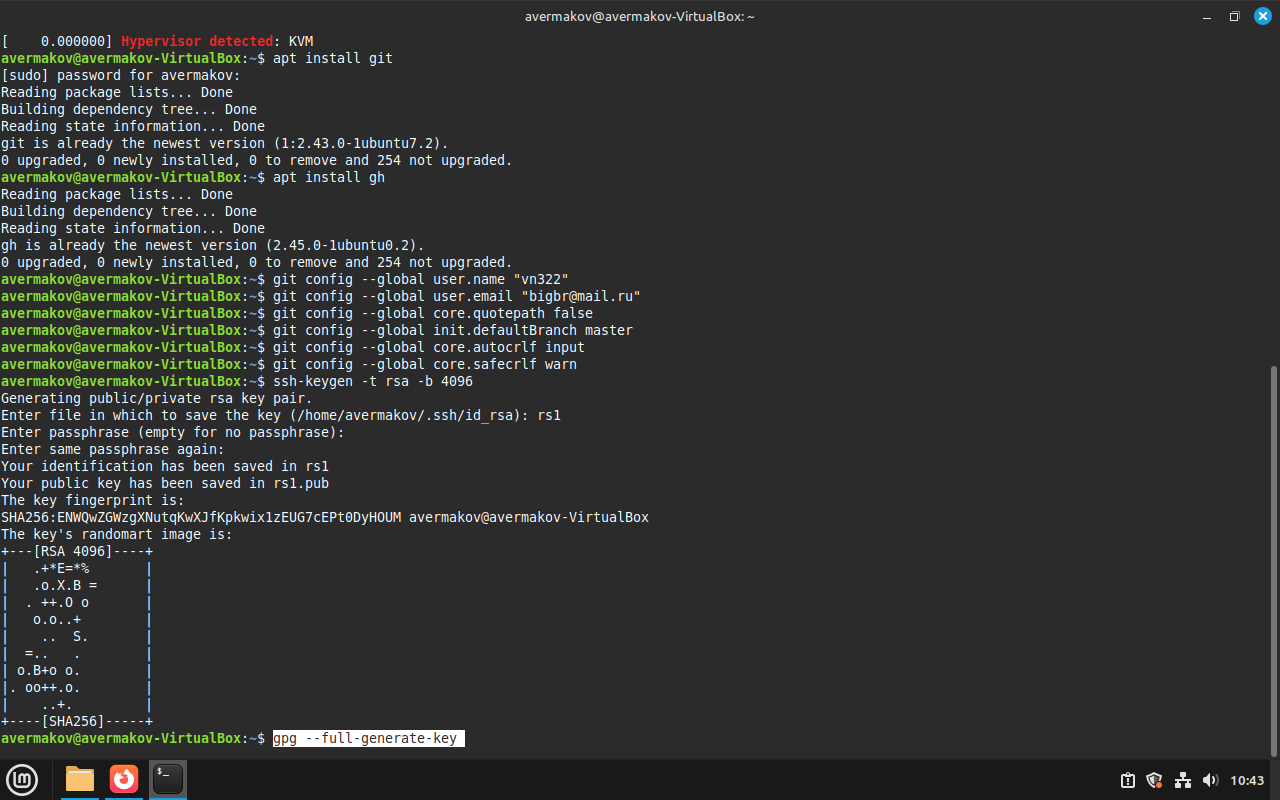


Рис. 2: Базовая настройка git

Создайте ключи ssh

по алгоритму rsa с ключём размером 4096 бит:  
  
ssh-keygen -t rsa -b 4096  
  
по алгоритму ed25519:  
  
ssh-keygen -t ed25519

Создайте ключи pgp

Генерируем ключ  
  
gpg --full-generate-key

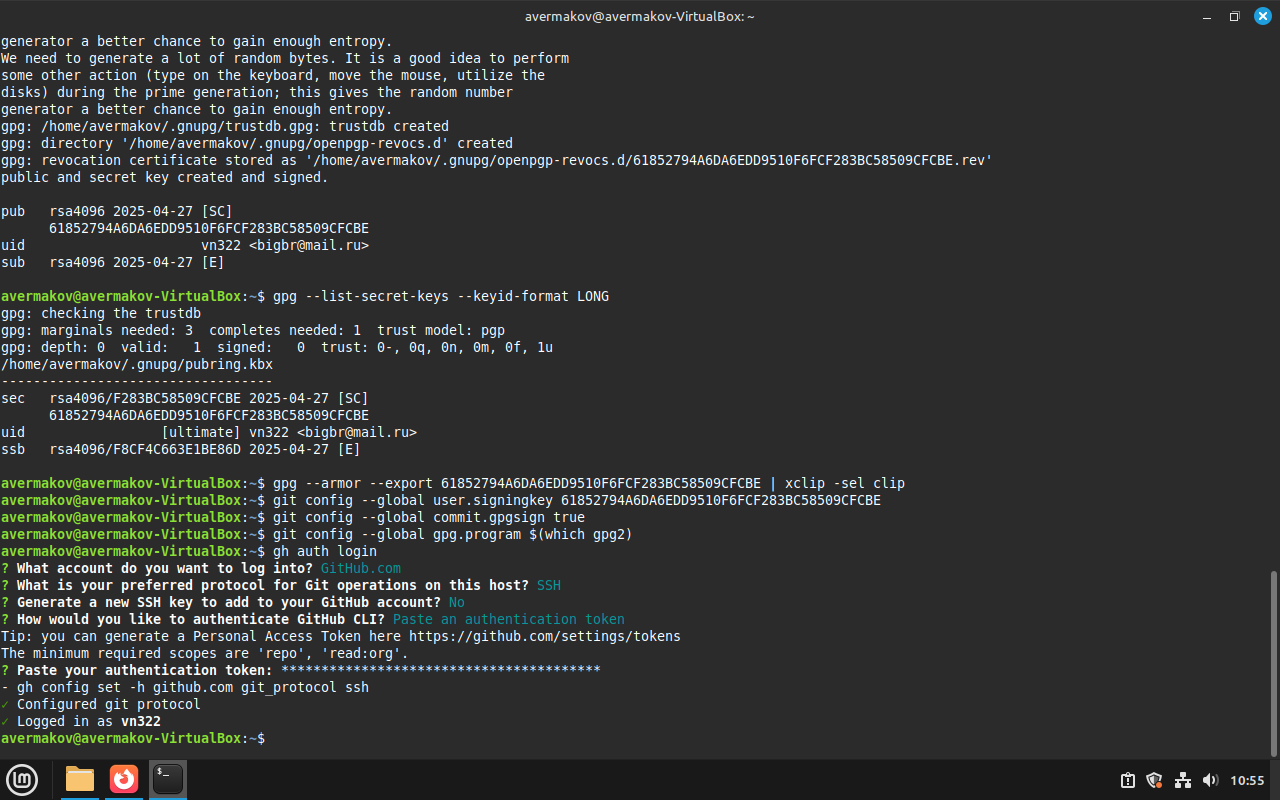


Рис. 3: Базовая настройка git

Настройка каталога курса(рис. 4).

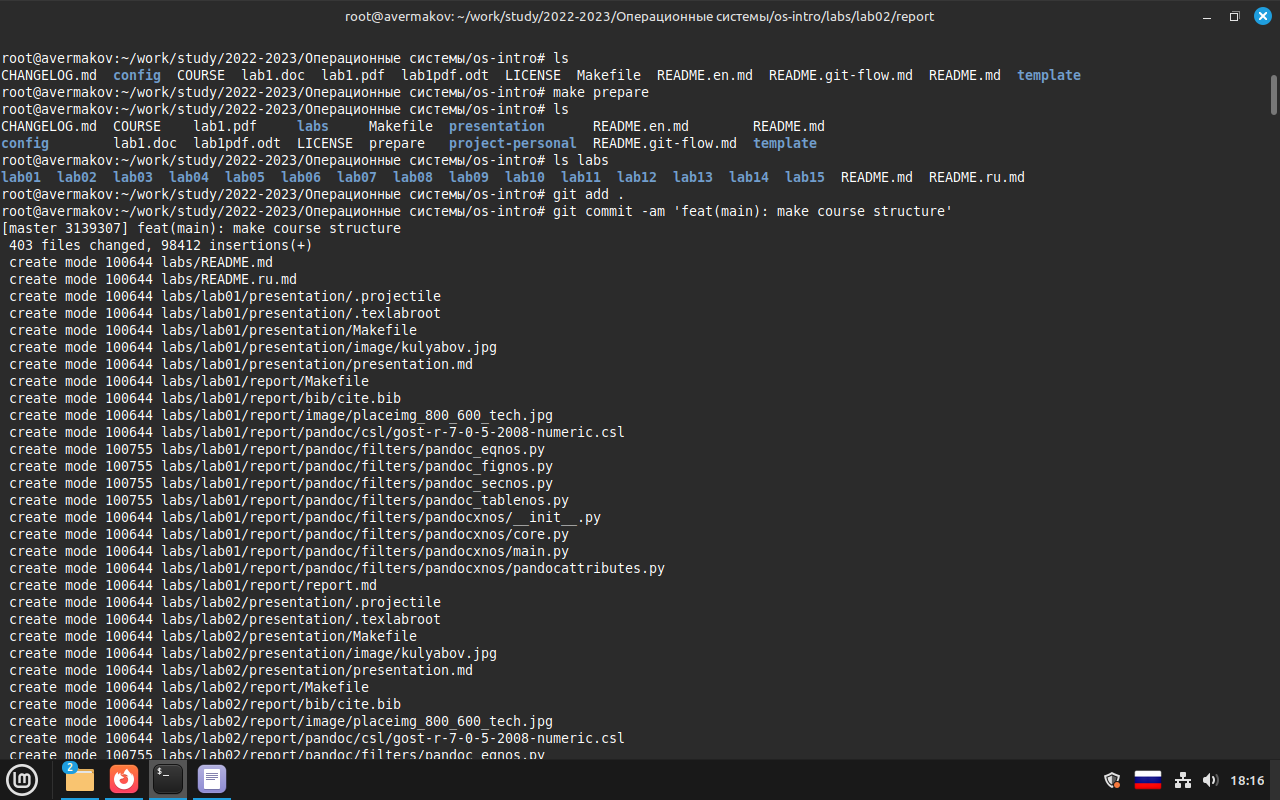


Рис. 4: Настройка каталога курса

# 5 Контрольные вопросы

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

Система контроля версий - программное обеспечение для облегчения работы с изменяющейся информацией.

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.

Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.  
  
Хранилище - репозиторий, хранилище версий, в нем хранятся все документы вместе с их историей, и другой служебной информацией.  
Commit - отслеживание изменений, сохранение разницы в рабочих изменениях.  
История - сохранение всех изменений в проекте и при необходимости возможность обратиться к старым данным.  
Рабочая копия - копия проекта, связанная с репозиторием, текущее состояние файлов проекта, основанное на их последней версии из хранилища.  
  
Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.  
  
Централизованные VCS - одно основное хранилище всего проекта, каждый пользователь копирует себе необходимые файлы из репозитория и изменяет их, а потом может добавить обратно. Примеры: - Subversion; - CVS; - TFS; - VAULT; - AccuRev;  
  
Децентрализованные VCS - У каждого пользователя свой вариант репозитория, есть возможность добавлять и забирать версии из любого репозитория. Например: - Git; - Mercurial; - Bazaar;

Обычно используются централизованные системы контроля версий, с одним общим репозиторием.

Опишите действия с VCS при единоличной работе с хранилищем. Сначала нужно создать удалённый репозиторий, после - подключить его. Затем по мере того, как проект будет выполняться, нужно отправлять данные на сервер.  
  
Опишите порядок работы с общим хранилищем VCS. В начале при помощи последовательности команд пользователь получает нужную ему версию данных. Потом он работает с ними, вносит некоторые изменения, и уже после этого он может разместить новую версию в хранилище. При этом предыдущие версии не удаляются из хранилища, что является очень удобным - к ним можно вернуться в любой момент.  
  
Каковы основные задачи, решаемые инструментальным средством git?  
  
Хранить информацию обо всех изменениях, производимых в проекте.  
Обеспечить командную работу.  
  
Назовите и дайте краткую характеристику командам git.

git init - Создание основного дерева репозитория

git pull - Получение обновлений (изменений) текущего дерева из центрального репозитория

git push - Отправка всех произведённых изменений локального дерева в центральный репозиторий

git status - Просмотр списка изменённых файлов в текущей директории изменений

git add . / git add / git rm - Сохранение текущих изменений

git commit / git commit -am “описание коммита” - Сохранение добавленных изменений

git checkout -b имя\_ветки - Создание новой ветки, базирующейся на текущей

git checkout имя\_ветки - Переключение на некоторую ветку

git push origin имя\_ветки - Отправка изменений конкретной ветки в центральный репозиторий

git merge –no-ff имя\_ветки - Слияние ветки с текущим деревом

git branch -d имя\_ветки - Удаление локальной уже слитой с основным деревом ветки

git branch -D имя\_ветки - Принудительное удаление локальной ветки

git push origin :имя\_ветки - Удаление ветки с центрального репозитория.

Приведите примеры использования при работе с локальным и удалённым репозиториями. git push -all (push origin master/любой branch)  
  
Что такое и зачем могут быть нужны ветви (branches)? Ветвь - один из параллельных участков истории проекта в одном хранилище. Все ветви исходят из одной версии - точки ветвления. Обычно ветви делятся на master и trunk. Между ветками возможно и слияние. Ветки нужны для разработки новых функций.  
  
Как и зачем можно игнорировать некоторые файлы при commit? Игнорировать некоторые файлы можно прописав шаблон .gitignore специально для игнорируемых файлов. Зачем это нужно? Чтобы в репозиторий не попали "лишние" файлы, которые неминуемо будут возникать при работе над проектом. Это могут быть временные файлы, объектные файлы.

# 6 Выводы

В данной работе познакомились с системой контроля версий git. Создали клон учебного репозитория создали и отредактировали файлы с последующей отправкой в репозиторий и проверкой их наличия. Визуальный интерфейс выглядит привлекательнее, но может и в командной строке есть свои плюсы.

# Список литературы

1. GNU Bash Manual [Электронный ресурс]. Free Software Foundation, 2016. URL: https://www.gnu.org/software/bash/manual/.
2. Newham C. Learning the bash Shell: Unix Shell Programming. O’Reilly Media, 2005. 354 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Robbins A. Bash Pocket Reference. O’Reilly Media, 2016. 156 с.
5. Таненбаум Э. Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 874 с.
6. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с. {#refs} :::