Лабораторная Работа №3

# Изучение системы контроля версий Git.

### Git

Чтобы забыть о проблемах централизованных систем контроля версий, были созданы децентрализованные системы контроля версий (ДСКВ или DVCS – Distributed Version Control System).

Как следует из названия, в таких системах клиенты не просто забирают последнюю (или выбранную, но всегда одну) версию проекта: на самом деле они полностью копируют его содержание и историю. Таким образом, такой подход имеет два больших преимущества перед VCS:

* В любой момент времени разработчик может посмотреть состояние проекта в какой-то момент времени даже без наличия доступа к центральному серверу;
* Каждая копия проекта это, по сути дела, полная резервная копия проекта. Таким образом, если сервер выйдет из строя, то любой разработчик сможет восстановить его состояние на любой другой машине.



Схема 2 – Децентрализованная система контроля версий

## Git

В качестве системы контроля версий для данной лабораторной работы будем использовать git. Цикл работы с git выглядит следующим образом:

1. git clone – создаёт копию проекта на локальной машине разработчика;
2. работа непосредственно с файлами (в default или, в другой ветке – git checkout branch\_name)
3. git pull – забрать изменения, которые были сделаны разработчиками во время, прошедшее с момента git clone
4. git add – добавляем файлы в тот планируемый commit
5. git status – просмотр всех изменённых и добавленных файлов
6. git commit – создание пакета со всеми staged файлами
7. git push – отправка пакета на сервер

Если у вас возникают вопросы в процессе работы с git, то вы можете воспользоваться командой git --help, которая позволит вам узнать наиболее часто встречающиеся функции; также вы можете воспользоваться справочником по отдельным командам. Так, в частности, команда git checkout --help откроет браузер с информацией об использовании checkout, вместе с параметрами и примерами.

# Zenity

Утилита **zenity** – это средство создания диалоговых окон в режиме командной строки. Следует отметить, что на самом деле диалоговые окна создаются средствами Gtk+, поэтому в системе должны быть установлены соответствующие библиотеки. От аналогичных программ, таких как **dialog** и **whiptail**, **zenity** отличают средства реализации GUI-элементов.

Применение **zenity** для отдельных команд в интерактивном режиме не столь эффективно, как при написании сценариев. В сценариях командной оболочки часто требуется взаимодействие с пользователем, чтобы сообщить некую информацию, например, о возникновении "нестандартной" ситуации. Также требуется отображать информацию о состоянии выполнения операции, длящейся продолжительное время. Кроме этого, иногда сценарию необходимо получить некоторую информацию от пользователя: выбор варианта ответа на заданный вопрос, выбор файла из предложенного списка и т.д. Всё это можно организовать с помощью **zenity**.

Необходимо уточнить, что после закрытия диалогового окна, **zenity** возвращает числовой код завершения операции:

* **0** - означает, что пользователь нажал в диалоговом окне кнопку "**OK**" или "**Закрыть**" (**Сlose**);
* **1** - означает, что пользователь нажал кнопку "**Отмена**" (**Cancel**) или воспользовался функциями (кнопками) окна, чтобы закрыть его;
* **-1** - сообщает о том, что операция завершилась с ошибкой;
* **5** - диалоговое окно было закрыто после истечения интервала таймаута.

## Создание диалогового окна для вывода сообщений

Чтобы начать использовать **zenity** на практике не требуется обладать особыми знаниями или умениями, достаточно просто познакомиться с различными опциями (ключами), позволяющими в полной мере использовать возможности этой программы.

В **zenity** определены четыре типа диалоговых окон для вывода сообщений:

* ошибка (ключ --**error**);
* информация (ключ **--info**);
* вопрос (ключ **--question**);
* предупреждение (ключ **--warning**).

## Установка zenity:

sudo **apt**-**get install zenity**

**Создание простейшего окна при помощи zenity.**

Задание:

1. Изучить базовую функциональность git и zenity.
2. Склонировать репозиторий, находящийся по адресу:
3. Используя zenity, реализовать следующую функциональность (исходя из своего варианта):

|  |  |  |  |
| --- | --- | --- | --- |
| № Варианта | Операция | Описание | Файл |
| 1 | Add | Сложение | add.sh |
| 2 | Sub | Умножение | sub.sh |
| 3 | Mul | Вычитание | mul.sh |
| 4 | Div | Деление | div.sh |
| 5 | Mod | Остаток от деления | mod.sh |
| 6 | Not | Отрицание | neg.sh |
| 7 | And | Конъюнкция | conj.sh |
| 8 | Or | Дизъюнкция | dis.sh |
| 9 | Pow | Возведение в квадрат | pow.sh |
| 10 \* | Sqrt | Извлечение корня | sqrt.sh |
| 11 \* | Exp | Возведение в степень | exp.sh |
| 12 \* | Log | Взятие логарифма | log.sh |
| 13 \* | Cos | Взятие косинуса от числа (радианы) | bc | cos.sh |
| 14 \* | Sin | Взятие синуса от числа (радианы) | bc | sin.sh |
| 15 \*\* | Main dialog | Создание файла-точки входа | main.sh |

1. Создать новую ветку, где название ветки формируются по типу задание-операция (для варианта 1: 1-add).
2. Внести изменения в свою ветку и занести их результаты на сервер.
3. Результат выполнения работы предъявить преподавателю.

# Контрольные вопросы:

1. Что такое Система Контроля Версий?
2. Перечислите основные команды git.
3. Расскажите подробнее о следующей команде:
   1. clone
   2. checkout
   3. branch
   4. commit
   5. push
   6. merge
4. Что такое zenity? Какие возможности он имеет?
5. \* Как бы вы реализовали это задание иначе?
6. \* Можно ли переименовать файл conj.sh в файл con.sh? Объяснить, если да, то какие последствия будет иметь такая операция, если нет, то почему.