**Министерство образования Республики Беларусь**

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

лабораторная работа

на тему

«**Основы системы контроля версий Git**»

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: |  | Козяков Андрей Игоревич, магистрант группы №815042 |  |
| Проверил: |  | Алексеев Виктор Федорович,  канд.техн.наук, доцент |  |

Минск 2018

**Тема: «Основы системы контроля версий Git»**

**Цель работы:** познакомиться с системой контроля версий Git, изучить возможности ее использования. После изучения теоретического материала выполнить тестовые задания для усвоения материала.

**Необходимые знания:** базовое знание команд консоли.

**1 Общие теоретические сведения**

Рассмотрим базовую теорию для того чтобы Вы могли выполнить эту лабораторную работу, а именно узнаем, что такое системы контроля версий, зачем они нужны, и почему так важны.

**1.1 Что такое система контроля версий**

Что такое "система контроля версий", и почему это важно? Система контроля версий – это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Для контроля версий файлов в этой книге в качестве примера будет использоваться исходный код программного обеспечения, хотя на самом деле вы можете использовать контроль версий практически для любых типов файлов.

Если вы графический или web-дизайнер и хотите сохранить каждую версию изображения или макета (скорее всего, захотите), система контроля версий (далее СКВ) – как раз то, что нужно. Она позволяет вернуть файлы к состоянию, в котором они были до изменений, вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда, и многое другое. Использование СКВ также значит в целом, что, если вы сломали что-то или потеряли файлы, вы спокойно можете всё исправить. В дополнение ко всему вы получите всё это без каких-либо дополнительных усилий.

**1.2 Что такое локальная система контроля версий**

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже, директорию с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.



Рисунок 1.1 – Локальный контроль версий.

Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами. Даже популярная операционная система Mac OS X предоставляет команду rcs, после установки Developer Tools. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

**1.3 Централизованные системы контроля версий**

Следующая серьёзная проблема, с которой сталкиваются люди, — это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как: CVS, Subversion и Perforce, имеют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.

Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать ЦСКВ, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус — это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми он работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё — всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Локальные СКВ страдают от той же самой проблемы — когда вся история проекта хранится в одном месте, вы рискуете потерять всё.



Рисунок 1.2 – Централизованный контроль версий.

**1.3 Децентрализованные системы контроля версий**

Здесь в игру вступают децентрализованные системы контроля версий (ДСКВ). В ДСКВ (таких как Git, Mercurial, Bazaar или Darcs), клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени): они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.



Рисунок 1.3 – Децентрализованный контроль версий.

Более того, многие ДСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно, в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

**2 Основы Git**

Что же такое Git, если говорить коротко? Очень важно понять эту часть материала, потому что если вы поймёте что такое Git и основы того, как он работает, тогда, возможно, вам будет гораздо проще его использовать. Пока вы изучаете Git, попробуйте забыть всё, что вы знаете о других СКВ, таких как Subversion и Perforce; это позволит вам избежать определённых проблем при использовании утилиты. Git хранит и использует информацию совсем иначе по сравнению с другими системами, даже несмотря на то, что интерфейс пользователя достаточно похож, и понимание этих различий поможет вам избежать путаницы во время использования.

**2.1 Снимки, а не различия**

Основное отличие Git’а от любой другой СКВ (Subversion и её собратья включительно), это подход Git’а к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы (CVS, Subversion, Perforce, Bazaar и т.д.) представляют информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени.



Рисунок 2.1 – Хранение данных относительно первоначальной версии каждого из файлов.

Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git’а к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git’е, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как, скажем, **поток снимков**.



Рисунок 2.2 – Хранение данных как снимков проекта во времени.

Это очень важное отличие между Git и почти любой другой СКВ. Git переосмысливает практически все аспекты контроля версий, которые были скопированы из предыдущего поколения большинством других систем. Это делает Git больше похожим на миниатюрную файловую систему с удивительно мощными утилитами, надстроенными над ней, нежели просто на СКВ. Когда мы будем рассматривать управление ветками в Ветвление в Git, мы увидим, какие преимущества вносит такой подход к работе с данными в Git.

Когда вы производите какие-либо действия в Git, практически все из них только добавляют новые данные в базу Git. Очень сложно заставить систему удалить данные либо сделать что-то, что нельзя впоследствии отменить. Как и в любой другой СКВ, вы можете потерять или испортить свои изменения, пока они не закоммичены, но после того, как вы закоммитите снимок в Git, будет очень сложно что-либо потерять, особенно, если вы регулярно синхронизируете свою базу с другим репозиторием.

Это самая важная вещь, которую нужно запомнить о Git, если вы хотите, чтобы остаток процесса обучения прошёл гладко. Git имеет три основных состояния, в которых могут находиться ваши файлы: зафиксированном (committed), изменённом (modified) и подготовленном (staged). “Зафиксированный” значит, что файл уже сохранён в вашей локальной базе. К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы. Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит.

Мы подошли к трём основным секциям проекта Git: Git-директория (Git directory), рабочая директория (working directory) и область подготовленных файлов (staging area).

Область подготовленных файлов — это файл, располагающийся в вашей Git-директории, в нём содержится информация о том, какие изменения попадут в следующий коммит. Эту область ещё называют “индекс”, однако называть её stage-область также общепринято.

Базовый подход в работе с Git выглядит так:

1. Вы изменяете файлы в вашей рабочей директории.
2. Вы добавляете файлы в индекс, добавляя тем самым их снимки в область подготовленных файлов.
3. Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в вашу Git директорию.

**3 Установка Git**

Для установки Git в Windows также имеется несколько способов. Официальная сборка доступна для скачивания на официальном сайте Git’а. Просто перейдите на страницу <http://git-scm.com/download/win>, и загрузка запустится автоматически. Обратите внимание, что это проект, называемый Git для Windows (другое название msysGit), который отделён от самого Git; для получения дополнительной информации о нём перейдите на <http://msysgit.github.io/>.

Другой простой способ установки Git — установить GitHub для Windows. Его установщик включает в себя утилиты командной строки и GUI Git’а. Он также корректно работает с Powershell, обеспечивает чёткое сохранение учётных данных и правильные настройки CRLF. Вы познакомитесь с этими вещами подробнее несколько позже, здесь же отметим, что они будут вам необходимы. Вы можете загрузить GitHub для Windows с сайта [http://windows.github.com](http://windows.github.com/).

**3.1 Первоначальная настройка Git**

Теперь, когда Git установлен в вашей системе, самое время настроить среду для работы с Git’ом под себя. Это нужно сделать только один раз — при обновлении версии Git’а настройки сохранятся. Но, при необходимости, вы можете поменять их в любой момент, выполнив те же команды снова.

В состав Git’а входит утилита git config, которая позволяет просматривать и настраивать параметры, контролирующие все аспекты работы Git’а, а также его внешний вид.

Первое, что вам следует сделать после установки Git’а, — указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Git’е содержит эту информацию, и она включена в коммиты, передаваемые вами, и не может быть далее изменена:

$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com

Опять же, если указана опция --global, то эти настройки достаточно сделать только один раз, поскольку в этом случае Git будет использовать эти данные для всего, что вы делаете в этой системе. Если для каких-то отдельных проектов вы хотите указать другое имя или электронную почту, можно выполнить эту же команду без параметра --global в каталоге с нужным проектом.

Многие GUI-инструменты предлагают сделать это при первом запуске.

Если вы хотите проверить используемую конфигурацию, можете использовать команду git config --list, чтобы показать все настройки, которые Git найдёт:

$ git config --list

user.name=John Doe

user.email=johndoe@example.com

color.status=auto

color.branch=auto

color.interactive=auto

color.diff=auto

...

**3.2 Создание Git-репозитория**

Если вы собираетесь начать использовать Git для существующего проекта, то вам необходимо перейти в директорию проекта и в командной строке ввести

$ git init

Эта команда создаёт в текущей директории новую поддиректорию с именем .git, содержащую все необходимые файлы репозитория — основу Git-репозитория. На этом этапе ваш проект ещё не находится под версионным контролем.

**3.3 Определение состояния файлов**

Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда git status. Если вы выполните эту команду сразу после клонирования, вы увидите что-то вроде этого:

$ git status

On branch master

nothing to commit, working directory clean

Это означает, что у вас чистый рабочий каталог, другими словами – в нем нет отслеживаемых измененных файлов. Git также не обнаружил неотслеживаемых файлов, в противном случае они бы были перечислены здесь. Наконец, команда сообщает вам на какой ветке вы находитесь и сообщает вам, что она не расходится с веткой на сервере. Пока что это всегда ветка ``master``, ветка по умолчанию; в этой главе это не важно. В [Ветвление в Git](https://git-scm.com/book/ru/v2/ch00/ch03-git-branching) будут рассмотрены ветки и ссылки более детально.

Предположим, вы добавили в свой проект новый файл, простой файл README. Eсли этого файла раньше не было, и вы выполните git status, вы увидите свой неотслеживаемый файл вот так:

$ echo 'My Project' > README

$ git status

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

README

nothing added to commit but untracked files present (use "git add" to track)

Понять, что новый файл README неотслеживаемый можно по тому, что он находится в секции ``Untracked files`` в выводе команды ``status``. Статус ``Untracked files``, по сути, означает, что Git видит файл, отсутствующий в предыдущем снимке состояния (коммите); Git не станет добавлять его в ваши коммиты, пока вы его явно об этом не попросите. Это предохранит вас от случайного добавления в репозиторий сгенерированных бинарных файлов или каких-либо других, которые вы и не думали добавлять. Мы хотели добавить README, так давайте сделаем это.

2 Примеры создания веб-сервера

Рассмотрим элементарные примеры создания простого веб-сервера на чистом Node.JS и используя фреймворк Express. А также попытаемся написать CRUD запросы (GET, POST, DELETE, UPDATE) как пример работы с маршрутизацией и понять разницу между написания на чистом Node.JS и используя Express.

Пометка. В интернете есть настолько хорошие ресурсы, что я бы хотел Вам порекомендовать их пройти как практику примера для Node.JS и Express.

2.1 Используя Node.JS (чистый)

Пункты 1 – 3: <https://nodeguide.ru/doc/dailyjs-nodepad/>

Хороший пример работы с Node.JS (желательно пройти этот пример): <https://metanit.com/web/nodejs/1.1.php>

2.2 Используя Express

Официальная документация и пример работы с REST маршрутизацией: <https://expressjs.com/ru/guide/routing.html>

Хороший пример простой реализации REST для Express:

* + 1. <https://habr.com/company/ruvds/blog/321104/>
    2. <https://habr.com/post/193458/#p1>

Так же есть много видео уроков на YouTube.

И помните, что лучшее как источник является официальная документация для Node.JS и для Express.

**3 Практика**

**3.1 Контрольные вопросы**

* 1. Что такое Node.JS?
  2. С помощью чего можно устанавливать модули для Node.JS?
  3. Для чего нужен Node.JS?
  4. Как называет подход для организации API?
  5. Какие популярные запросы существуют в HTTP?
  6. Что такое Express?
  7. Зачем нужен Express?

**3.2 Порядок выполнения работы**

* 1. Изучение теоретические сведения.
  2. Ознакомиться с полезными источниками из презентации.
  3. Ответить на контрольные вопросы.
  4. Выполнить просто задание.

**3.3 Задание**

* 1. Написать и запустить простой веб-сервер с запросами GET, UPDATE, POST, DELETE на Node.JS.
  2. Написать и запустить простой веб-сервер с запросами GET, UPDATE, POST, DELETE на Express.

**Памятка:** пользоваться ресурсами можно не только из раздела с теорией, а также интернетом ввиду популярности данной и темы, и большого источника с информацией.