Лабораторная работа №2

# Цель работы: Изучение системы контроля версий Mercurial.

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

Mercurial — это современная распределенная система контроля версий с открытым кодом. Эта система — замена для более ранних систем вроде Subversion. У нее два основных назначения:

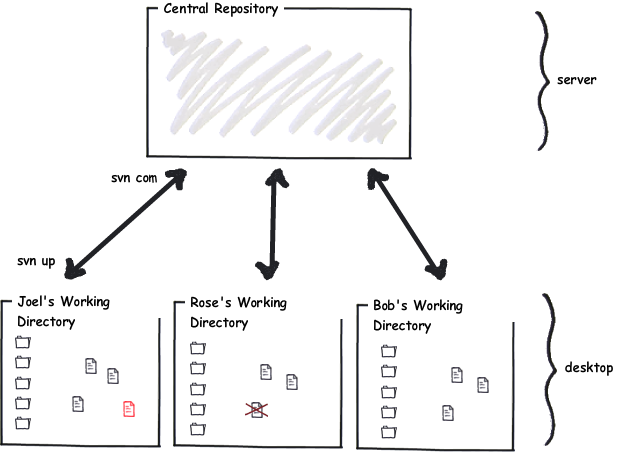
1. Она хранит все предыдущие версии каждого файла
2. Она может объединить разные версии вашего кода, то есть сотрудники могут независимо работать над кодом и затем объединять свои изменения.

Система Mercurial написана на Python, хотя чувствительные к производительности части (например, своя реализация diff) выполнены в качестве модулей-расширений на C. Mercurial первоначально была написана для Linux, позже портирована под Windows, Mac OS X и большинство Unix-систем. Репозитории Mercurial управляются при помощи утилиты командной строки hg, но есть и GUI–интерфейсы.

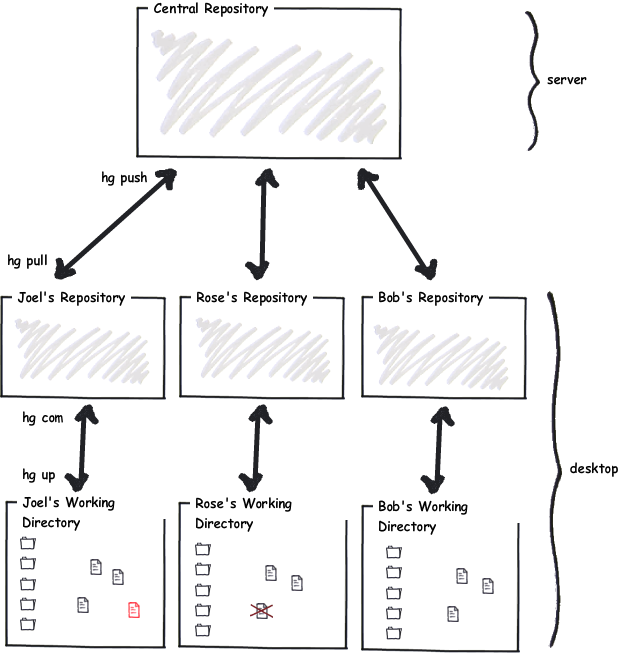
При работе с Subversion, когда вы вносите новый код в репозиторий, его получают все. Так как весь новый код, который пишет программист, содержит баги и/или недоделки, то разработчик встаёт перед следующим выбором:

* Вносить неработающий (не до конца рабочий) код (невозможно при работе с реальным заказчиком);
* Придерживать новый код до момента его окончательной отладки (всё время).

На практике, команда, использующая Subversion, часто днями или неделями ничего не добавляет в репозиторий. В таких командах новички боятся залить что-то в репозиторий из-за опасения поломать проект, и возвращаются к старым методам передачи работы с проектом через архивы. Вот простая иллюстрация работы с Subversion:



При работе с Mercurial *у каждого разработчика свой собственный репозиторий, живущий у него на компьютере:*



*Mercurial разделяет момент внесения кода в репозиторий и момент получения этого кода всеми остальными.*

И это означает, что вы можете коммитить (hg com) (hg commit), но все остальные не получат ваши изменения. Когда у вас накопятся изменения, которые вас устраивают, вы проталкиваете (hg push) их в главный репозиторий.

В Subversion мыслят **ревизиями**. Ревизия — это то, как выглядит вся файловая система в определенный момент времени. В Mercurial вы мыслите наборами изменений (**changesets**). Набор изменений — это четкий список изменений между двумя соседними ревизиями.

Когда нам нужно сделать слияние, Subversion смотрит на обе ревизии —измененный собственно разработчика и измененный код, который уже находится на сервере — и пытается угадать, как слепить их вместе в одно целое (слияние). Обычно Subversion это не удается, и получаются длинные списки конфликтов («merge conflicts»), которые на самом деле не конфликты, а просто места, в которых система не смогла разобраться в наших изменениях. Такие конфликты разрешаются вручную. Аналогичная система и в Mercurial, только merge conflicts встречаются реже.

## Основные команды Mercurial.

Большинство людей работают с Mercurial через интерфейс командной строки. Так можно работать в Windows, Unix, и Mac. Команда для Mercurial — это hg. Для того, чтобы выполнить эту команду, необходимо

sudo apt-get install mercurial

Если выполнить hg без параметров, то вы получите список наиболее часто используемых команд. Вы можете также попробовать выполнить hg help для получения полного списка команд.

Для того чтобы воспользоваться преимуществами контроля версий, вам нужен репозиторий. Репозиторий хранит все предыдущие версии всех ваших файлов. На самом деле, для экономии места на диске, все предыдущие версии не будут храниться — будет храниться только компактный список изменений.

hg init создает репозиторий. В результате выполнения этой команды появляется новый каталог .hg:

Это и есть репозиторий! Это каталог со всем, что нужно Mercurial для работы. Настройки, предыдущие версии файлов, теги и прочее.  Вам почти всегда не стоит возиться с этим каталогом самостоятельно. Если мы захотим добавить в него все исходные файлы, то нужно всего лишь выполнить hg add.

В данной лабораторной работе выполнять hg init нет необходимости, достаточно сделать hg clone.

hg clone делает полную копию всего репозитория.

hg add помечает файлы как запланированные для добавления в репозиторий. Файлы на самом деле не будут добавлены до тех пор, пока вы не зафиксируете изменения.

Остался еще один шаг… вам нужно зафиксировать ваши изменения. Какие изменения? Факт добавления всех этих файлов.

Почему вы вынуждены фиксировать изменения? При работе с Mercurial фиксирование изменений означает: «Эй, вот как файлы сейчас выглядят, так их и запомни, пожалуйста». Это как создание копии всего каталога… каждый раз, когда у вас есть что-то измененное, и оно вам типа нравится, то вы фиксируете изменения.

hg commit сохраняет текущее состояние всех файлов в репозиторий.

В случае такой ошибки, которая вам вероятно всего встретится первый раз при работе с репозиторием:

error: abort: no username supplied (see "hg help config")

вам необходимо будет добавить файл hgrc (именно с таким именем, без расширения) в .hg директорию. Содержание файла должно будет быть примерно таким:

[ui]

username = Your Name <your@mail>

После этого сохранение должно заработать. Тем ни менее, Mercurial ещё запустит редактор для того, чтобы вы могли напечатать комментарий к изменениям. Вам нужно просто написать что-нибудь, что напомнит вам о сделанных изменениях. Это делается для того, чтобы вы могли просмотреть историю изменений.

hg log отображает историю изменений, зафиксированных в репозитории.

Благодаря Mercurial, если вам не нравится то, что вы сделали, вы можете выполнить удобную команду hg revert, которая немедленно вернет ваш каталог к виду, в котором он был в момент последнего коммита.

hg revert возвращает измененные файлы к зафиксированному в репозитории виду.

Используйте аргумент --all, если вы хотите вернуть все файлы к прежнему состоянию.

Таким образом, когда вы работаете над исходным кодом, используя Mercurial, вы:

1. Делаете изменения
2. Оцениваете, подходят ли они
3. Если подходят, то выполняете commit
4. Если не подходят, то выполняете revert
5. GOTO 1

Со временем вы можете забыть, на чем остановились и что поменяли с момента прошлого коммита. Mercurial отслеживает все это для вас. Все что вам нужно, это выполнить hg status и Mercurial даст вам список измененных файлов.

hg status отображает список измененных файлов, добавляя значок в начале каждой строки. Этот значок сообщает о том, что же произошло. M означает «Modified» — файл был изменен. ! означает отсутствие — файл должен быть здесь, но куда-то делся. ? означает что, состояние не определено — Mercurial ничего не знает про этот файл.

Кроме hg status есть ещё команда hg diff, которая сообщит вам, что произошло с файлом с прошлого коммита.

hg diff показывает, что изменилось в файле.

Теперь, вы наверняка хотите положить ваши изменения на сервер. Это делается в пять этапов:

1. команда hg pull
2. команда hg merge
3. тестирование
4. команда hg commit
5. команда hg push

hg pull подтягивает (pull) все наборы изменений из другого хранилища в текущий. Заметьте, именно из другого хранилища.

hg merge делает объединение двух изменений в одно большое. В рамках этой лабораторной работы, если вы всё делали правильно, вам не нужно будет делать эту операцию.

hg push проталкивает свежие изменения из текущего репозитория в центральный.

Также есть некоторые другие популярные команды, которые вряд ли пригодятся вам для выполнения данной лабораторной работы, однако могут значительно облегчить вам жизнь в случае необходимости.

hg remove помечает файлы как запланированные для удаления из репозитория. Файлы на диске не будут удалены до тех пор, пока вы не зафиксируете изменения.

hg cat отображает содержимое любого файла для любой ревизии.

Для того чтобы увидеть как файл выглядел раньше, можно просто указать нужный номер набора изменений (changeset из лога) при помощи аргумента -r («revision», то есть ревизия). Если у файла сложное содержимое или большая длина, а изменился он лишь немного, то можно использовать команду hg diff с аргументом r для вывода разницы между двумя ревизиями.

hg update приводит рабочий каталог к состоянию, как у заданной ревизии.

Однако всё время смотреть изменения при помощи командной строки довольно неудобно. Кроме этого, иногда бывает необходимость в создании центрального репозитория (например, на сервере). Проще всего создать центральный репозиторий при помощи встроенного в Mercurial веб-сервера. Вам нужно лишь создать репозиторий при помощи hg init и открыть к нему доступ при помощи hg serve. По умолчанию репозиторий будет доступен на 8000-ом порту.

hg serve запускает веб-сервер и делает текущий репозиторий доступным в сети Интернет.

## Краткие основы tcl.

\*Подробная информация: http://zetcode.com/gui/tcltktutorial/

Для выполнения задания вам понадобится tcl – скриптовый язык высокого уровня. Для того, чтобы запустить программу на tcl, её не нужно компилировать – достаточно вызвать её при помощи утилиты командной строки tlcsh (соответвенно tcl и sh -- шелл):

dzmitry@dzmitry-VirtualBox:~/VCS-labs/Labs/tcl$ tclsh array\_sort.tcl

Для выполнения задания вам нужно знать основные команды языка:

|  |  |
| --- | --- |
| Пример функции | Краткое описание |
| set x "Hello, World!" | Присваивание строки переменной х |
| puts "A simple string: $x\n" | Вывод на экран |
| set where [file dirname [info script]]  source [file join $where hello.tcl]  source [file join $where doClear.tcl]  … | Добавить .tcl файл в другой .tcl файл |
| if {… } {…} | Если-тогда выражение |
| foreach name [array names copy] {  puts "$name is $copy($name)"  } | Вывод массива на экран |

Таким образом, алгоритм создания простейшей программы на tcl:

set x "Hello, World!"

puts "A simple string: $x\n"

Эти строки поместить в файл с расширением .tcl

Выполнить dzmitry@dzmitry-VirtualBox:~/VCS-labs/Labs/tcl$ tclsh hello.tcl

Однако, команда tlcsh работают только в командной строке, что и следует из её названия. В данной же лабораторной работе вам будет предложено дописать функциональность калькулятора. Это можно сделать, используя процедуру wish. Например, в лабораторной работе, файл с точкой входа приложения нужно будет выполнить следующим образом:

dzmitry@dzmitry-VirtualBox:~/VCS-labs/Labs/tcl$ wish calc.tcl

# Задание:

Часть 1.

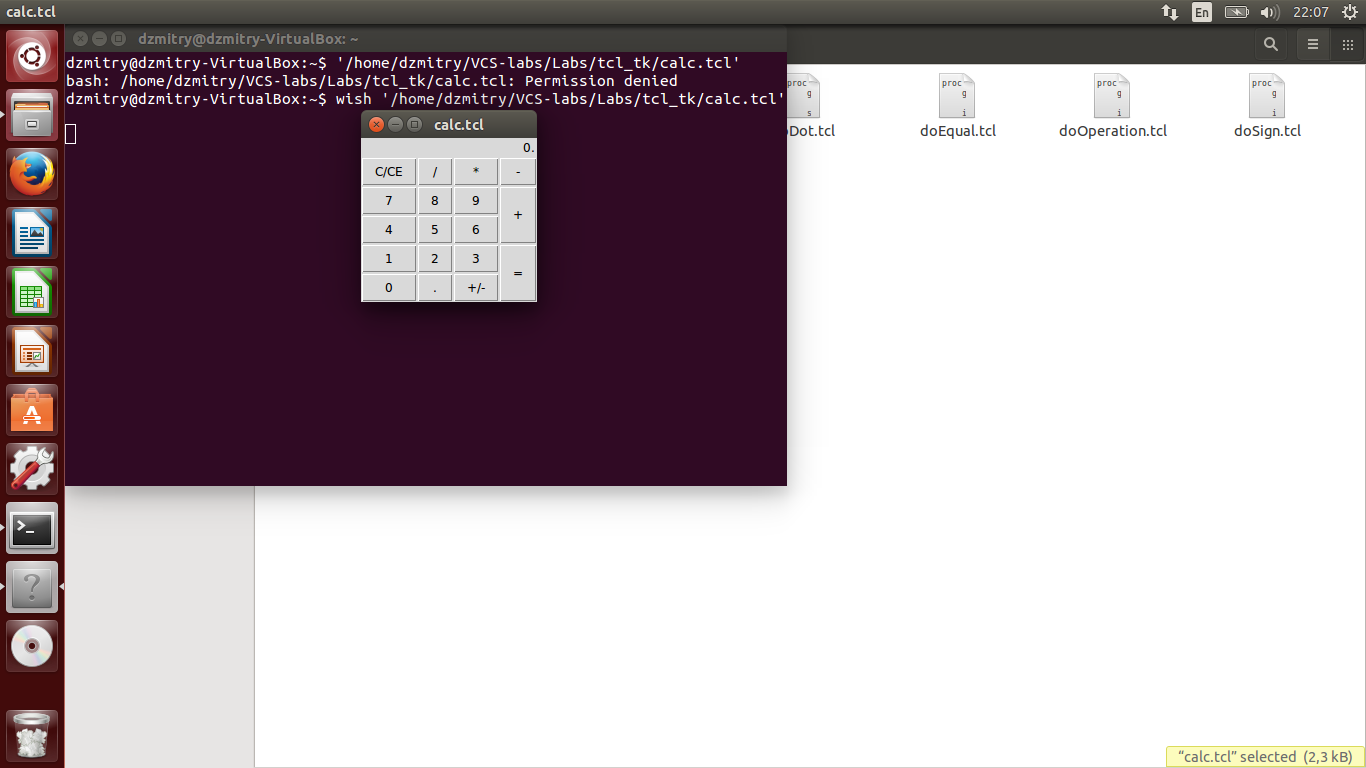
1. Установите tk на рабочую машину при помощи команды

sudo apt-get install tk8.5

1. Склонируйте репозиторий, который находится по адресу, указанному преподавателем, к себе на локальную машину
2. Создайте в локальном репозитории файл со своим именем на языке tcl, который выводил бы в консоль вашу имя и фамилию. Используйте в качестве имени файла свою имя и фамилию (Ivan\_Ivanov.tcl).
3. Используя команды mercurial, добавьте этот файл в удалённый репозиторий
4. В файле all.tcl добавьте строку, которая вызывала бы ваш файл. В итоге должен получиться файл, вызвав который, преподаватель сможет увидеть имена всех, кто дошёл до этой части задания.

Часть 2.

1. Откройте папку calc. Обратите внимание на уже существующие файлы: они все понадобятся вам для работы.
2. Восстановите исходную функциональность калькулятора. Если вы всё сделали правильно, то в конце вы должны получить примерно такое окно:



1. В репозитории вы найдёте код, в котором не достаёт переменных, данных, цифр, атрибутов и так далее. Восстановив его, вы получите рабочее приложение – калькулятор на tcl. Варианты:
   1. calc.tcl (set и source части)
   2. calc.tcl (grid часть)
   3. doEqual.tcl (применить операцию)
   4. doOperation.tcl (вызвать операцию)
   5. doSign.tcl (операции со знаком)
   6. doAppend.tcl (сделать правильное сравнение строк)
   7. doDot.tcl (С/СЕ)
   8. doClear.tcl (применить точку)
2. После выполнения задания добавьте изменения на сервер.
3. \* Попытайтесь выполнить команду hg serve и посмотрите на результат её выполнения, которые можно внести в отчёт.