Мы разрабатываем систему VCS на языке Python – это система контроля версий, которая позволяет отслеживать изменения в проекте: изменения в коде программы, а так же появление новых файлов или удаление старых.

Наша система контроля версий обладает следующим функционалом:

* Система авторизации
* Система прав
* Пользовательские функции для работы с проектом (commit, push, pull, load и т.д.)
* Ветвление

Рассмотрим более детально каждый из пунктов.

**Система авторизации**

При запуске нашей программы, первой запускается функция авторизации. Она предлагает войти в систему, используя уже существующий аккаунт или создать новый. Если пользователя с введённым логином не существует, то программа выдаст ошибку авторизации и предложит создать новый аккаунт.

Для хранения информации об аккаунтах пользователей используется следующая структура:

{ username\_1: password; username\_2: password; }

Которая находится по следующему пути /global/bin/users.txt.

**Ветвление**

Ветвление в нашей системе контроля версий – это своеобразный механизм, позволяющий пользователю быстро переключаться между задачами и решать их (в пределах проекта). По умолчанию, проект, и все его изменения, хранятся в ветке master ( /project\_name/master/ ). Но, при возникновении необходимости изменить что-либо в проекте, при этом не меняя уже имеющийся код, пользователь может создать ещё одну ветку и сохранять изменения проекта уже в этой ветке. После внесения изменений (одного или нескольких коммитов и последующих пушей. Про них рассказано в пункте Пользовательские функции) пользователь может слить все сделанные изменения из своей ветки в ветку master, при этом, при возникновении конфликтной ситуации, будет выводиться сообщение о невозможности слияния текущей ветки с меткой master( слияние всегда происходит с веткой master ). В случае, если между файлами не возникает никаких конфликтных ситуаций, в ветку master добавляются все изменения из ветки, которую хотят слить. Далее, вызываются функции commit и push и производится изменение стеков в папках global и local.

**Система прав**

Наша система контроля версий использует систему прав для проверки доступа пользователя к определённому функционалу. Для работы используется 2 структуры:

users\_rights\_for\_project – используется для хранения списка пользователей, которые обладают тем или иным уровнем доступа к функционалу VCS в данном проекте.

{ project\_name: { master: [ admin, ivan, dima, denis ]; branch1: [admin, ivan, dima, petya] } }

users\_requests – используется для хранения приглашений для пользователя на вступление в тот или иной проект.

{username: { master: [ [project\_owner1, prj\_1] ]; branch1: [project3\_owner, prj\_3] }

**Принцип работы системы прав**

* Пользователь заходит в свой проект и вызывает функцию добавления новых пользователей add\_users\_to\_project, затем он может добавить нового пользователя или в весь проект (во все ветки), или в какую-то ветку проекта. Выбрав один из вариантов, пользователю выводится список всех пользователей, из которого он выбирает тех, кого хочет добавить. Выбрав и подтвердив свой выбор, в структуру users\_requests каждому выбраному пользователю отправляется приглашение вида { branch: [ project\_owner, prj ] }.
* Когда в систему заходит пользователь, то сразу же вызывается функция, проверяющая, есть ли у этого пользователя какие-нибудь приглашения, и если есть, то он должен выбрать, какие из запросов принять, а какие нет. Независимо от того, какие запросы выберет пользователь, все запросы на его имя будут удалены. Когда пользователь выберет, в какие проекты он хочет вступить, и подтвердит свой выбор, откроется файл со структурой users\_rights\_for\_projects, в которую будет, по выбранным в приглашении проектам и веткам, добавлено его имя.

Аналогично функции add\_users\_to\_project, работает функция del\_users\_from\_project, которая предлагает удалить пользователей из всего проекта или из какой-то ветки, а затем, выбрав и подтвердив свой выбор, из структуры users\_rights\_for\_projects будут удалены выбранные пользователи из соответствующих веток.

Уровень доступа определяется двумя функциями: have\_user\_high\_lvl\_of\_rights и have\_user\_some\_lvl\_of\_rights. Высшим уровнем пользователя обладает только админ и создатель проекта, то есть первая функция возвращает True, если имя пользователя находится в нулевом или первом элементе. Вторая ф-ция возвращает True для выбранной ветки, если в выбранной ветке есть его имя.

При создании ветки, 2-им элементом будет обозначено имя пользователя, создавшего эту ветку.

**Функции для работы с проектом**

После авторизации и проверки приглашений, пользователю предлагают следующий набор функций:

* choose - выбрать свой проект
* make – создать новый проект
* load – загрузить проект из локальной директории
* exit – выйти

choose - выводит на экран информацию обо всех проектах, доступных пользователю и принимает на вход имя проекта, который хочет открыть пользователь.

Make – предлагает создать новый проект и, в случае, если проект будет создан, автоматически переходит в этот проект.

Load – эта функция позволять загрузить проект (при условии, что пользователь обладает необходимым уровнем доступа), которой он, например, удалил.

Exit – предлагает пользователю выйти из аккаунта, а затем выйти из VCS совсем.

**Функции, доступные после выбора проекта**

После того, как пользователь выберет проект, будет автоматически выбрана одна из доступных веток.

* change\_branch – выводит список веток, доступных пользователю и предлагает выбрать одну из них.
* make\_branch – функция для создания новой ветки.
* set\_project – функция для смены проекта.
* commit – функция для фиксирования изменений в проекте в папке local в стеке (структура для фиксирования изменений).
* push – функция для отправки изменений в папку global в текущую ветку. Так же в стек в папку global в выбранной ветке будут занесены соответствующие изменения.
* show projects – выводит список всех проектов, доступных пользователю.
* update – функция, сравнивающая изменения в стеках в папках local и global в выбранной ветке и, если есть какие-то изменения, то обновляет проект пользователя в папке local до последней версии.
* log – функция, позволяющая узнать историю коммитов
* del\_last\_commit – функция, удаляющая последний коммит в папке local в выбранной ветке.
* merge – Функция для слияния текущей ветки с веткой master

**Смысл большей части функций понятен из описания, поэтому далее мы более подробно разберём более сложные функции и то, как они работают.**

**Стек. Принцип работы функций commit и push**

Для записи изменений в проекте, используется специальная структура – стек. Причём, в каждой ветке находится свой стек, и изменения, сделанные в определённой ветке, после коммита будут занесены только в стек выбранной ветки. Стек выглядит следующим образом:

Наш стек – это список из словарей. Каждый словарь – это сделанный коммит. В каждом словаре содержится 3 ключа: ‘user’ – ключ, по которому хранится имя пользователя, сделавшего коммит; ‘date\_time’ – ключ, по которому хранится дата и время коммита; ‘changes’ – ключ, по которому находятся все изменения, сделанные за коммит.

[ **{** **‘user’:** username**;** **‘date\_time’:** date\_time**; ‘changes’:** {} **}** ]

**Ключ ‘changes’ в стеке**

По этому ключу находится словарь, в котором в качестве ключей, используются пути к файлам (путь начинается с папки Project), а по этим ключам находится список из двух элементов: спецификатор и словарь. Всё это выглядит следующим образом:

**‘changes’: { path\_to\_changed\_file:** [ qualifier, {} ]

В качестве спецификотора выступают символы: ‘+’, ‘-’ и ‘…’. ‘+’ означает, что это новый файл, которого не было в прошлых коммитах. ‘-’ – означает, что файл был удалён. ‘…’ – означает, что файл уже был создан, но в нём имеются какие-то изменения.

Спецификатор находится в списке на нулевом месте. На первом расположен ещё один словарь, в котором в качестве ключей выступают номера строк, которые изменили. На нулевом месте в этой структуре находится ещё один спецификатор для строки

**{ string\_number:** [ **qualifier**, ] **}**

По номерам строк может находиться список из двух или трёх элементов. Если спецификатор был ‘+’ или ‘-’, тогда список будет состоять из двух элементов. Вторым элементом будет строка, которую добавили (Спецификатор ‘+’)/удалили (Спецификатор ‘-‘). Если же такая строка была изменена (Спецификатор ‘…’), то вторым элементом будет строка до изменений, а третьим – строка после изменений.

Для наглядности, приведём пример:

Пусть у пользователя XXX проект выглядел следующим образом:



А затем он сделал следующие изменения 19.07.2016 в 13 45:



Удалил файл ***2.txt*** , создал файл ***3.txt*** и изменил файл ***1.txt*** и сделал коммит.

Вот так будет выглядеть последний элемент стека после коммита:

{ **‘user’**: **‘**XXX**’ ;**

**‘date\_time’: ’**19 07 2016 13 45 30**’ ;**

**‘changes’:** **{ ‘Project/local/My project/master/1.txt’:** [ **‘**…**’,**

{ **2:** [ **‘**…**’,** **‘**45**’, ‘**456**’** ]

**3:** [ **‘**-**’, ‘**6**’** ] }

] **;**

**‘Project/local/My project/master/2.txt’:** [ **‘**-**’,**

{ **1:** [ **‘**-**’,** **‘**ab**’** ]

**2:** [ **‘**-**’, ‘**c**’** ] }

] **;**

**‘Project/local/My project/master/3.txt’:** [ **‘**+**’,**

{ **1:** [ **‘**+**’, ‘**xy**’** ]

**2:** [ **‘**+**’, ‘**z**’** ] }

]

**}**

}

Стек:

[

**{** **‘user’:** username**;** **‘date\_time’:** date\_time**;**

**‘changes’: { path\_to\_changed\_file:** [ qualifier, **{ string\_number:** [ **qualifier**, … ] **}** ]

**}**

]

После выполнения команды commit, все, сделанные нами, изменения были зафиксированы в локальном стеке. Теперь мы можем запушить эти изменения в папку global командой push. Поле её выполнения все изменения будут так же произведены и в папек global соответствующего проекта.

**Функция log**

В нашей VCS имеется возможность посмотреть историю коммитов, которые были запушены.

log/log --simple - выводит подробную историю коммитов (показывает изменения внутри файлов)

log --name-only – выводит краткую историю коммитов (только имена изменённых файлов)

log --reverse – выводит подробную историю коммитов в обратном порядке

log --after date – выводит подробную историю коммитов после определённой даты date.

В зависимости от формата написания даты, функция работает следующим образом:

yyyy/mm/dd HH:MM – выводит историю коммитов после определённых даты и времени,

yyyy/mm/dd – выводит историю коммитов после определённой даты,

HH:MM - выводит историю коммитов за сегодня после HH:MM,

all HH:MM - выводит историю коммитов, которые были сделаны после HH:MM во все дни.

log --before date – выводит подробную историю коммитов до определённой даты (формат даты см. log --after)

log --author user – выводит подробную историю коммитов, сделанных юзером с именем user

Принцип работы функции:

Функция log выгружает данные из файла .stack.txt, который лежит в директории …/global/project\_name/branch\_name, и записывает их в список из словарей, затем выводит на экран имя юзера, сделавшего коммит, время коммита и пару: спецификатор и укороченный путь к файлу. Если спецификатор файла ‘…’, то функция также выводит изменения в файле (номер строки, спецификатор и, если спецификатор ‘+’ или ‘-’, то строку, если ‘…’, то две строки: первая – как было, вторая – как стало). Каждый коммит отделяется строкой из ‘#’.

**Функция merge**

Принцип работы функции merge заключается в том, что при её вызове происходит поиск изменений между веткой master и текущей веткой. Алгоритм нахождения изменений аналогичен алгоритму в функции commit. Далее цикл проходит по каждому файлу, в котором были изменения.

Алгоритм обнаружения конфликтов следующий: изначально по ветке master создаётся пустой файл **(\*)** с таким же именем, как и у текущего файла, но в скобках дописывается название ветки, с которой происходит слияние, и дата/время слияния. Воспользовавшись функцией, которая находит изменения, можно определить, в каких строчках конфликт.

Рассмотрим следующие условия:

- Если строка была непустой и изменилась, то - конфликт.

- Если строка была пустой или не существовала вовсе, то конфликта нету.

После проверки очередных строчек, в **(\*)** заносится строка из изменённого файла. Если в итоге конфликты не были обнаружены, то старая версия файла по ветке master удаляется, а **(\*)** переименовывается в соответсвии с тем файлом, иначе, если хотя бы один конфликт был обнаружен, то по ветке master будут находиться обе версии этого файла.

**Функция update**

Данная функция работает аналогично команде push, но только за старую версию принимается проект в папке local, а за новую - проект в папке global.