**4 СЦЕНАРИЙ ВЗАИМОДЕЙСТВИЯ ПОЛЬЗОВАТЕЛЯ С СИСТЕМОЙ**

В данном разделе рассмотрены примеры работы с Subversion как централизованной системой контроля версий, а также Mercurial и Git как децентрализованной. Данный раздел может быть использован в качестве вспомогательного материала для выполнения практических заданий.

**4.1 Работа в Subversion**

Для получения рабочей директории на свой компьютер необходимо выполнить команду svn checkout. Стоит отметить, что практически в каждой системе контроля версий (и во всех рассматриваемых в данном лабораторном практикуме) есть такая команда – именно с неё обычно начинается вся работа над проектами. В случае с svn есть удобное сокращение от checkout – команда co, пример работы которого можно наблюдать на рисунке 4.1.

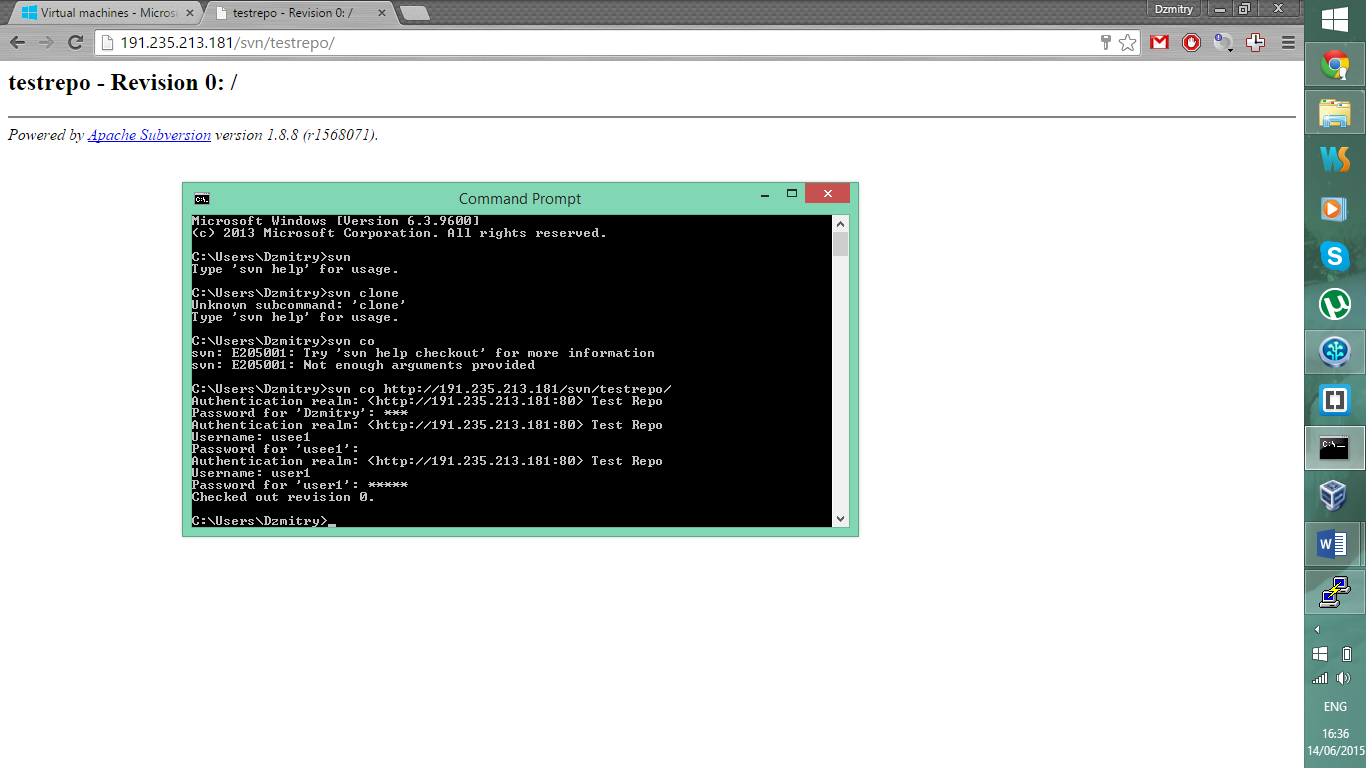


Рисунок 4.1 – Работа команды svn checkout

После выполнения данной операции, если эта ревизия была первой, то в файловой системе должна появится новая папка, в которой будет скрытая директория .svn (как, например, на рисунке 4.2), в которой содержатся все файлы, необходимые для Subversion.

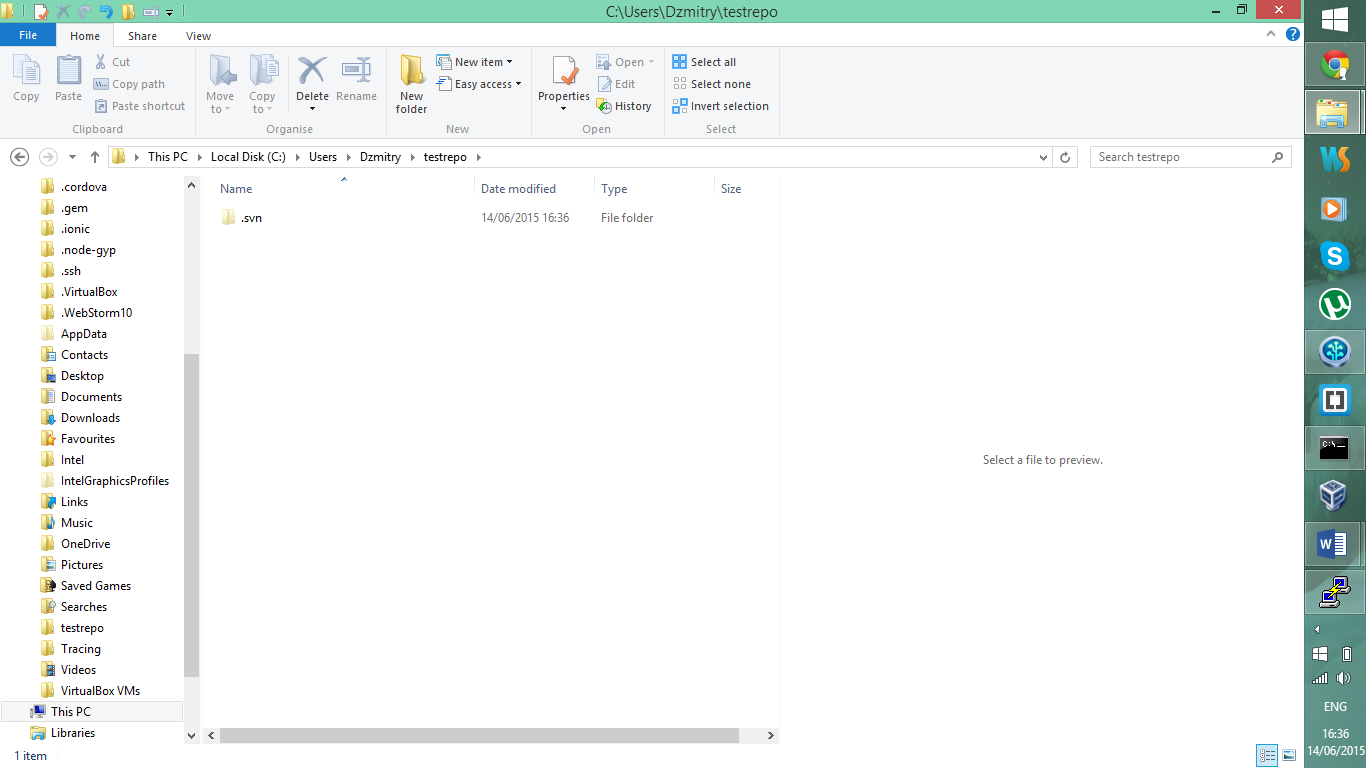


Рисунок 4.2 – Ожидаемый результат работы команды svn checkout

Добавление файлов в Subversion осуществляется при помощи команды svn add. Для того, чтобы продемонстрировать правильность выполнения команды, необходимо добавить в рабочий каталог файл, в данном случае – main.h и выполнить команду svn commit. У данной функции есть сокращение – svn ci, предполагаемый пример работы которой продемонстрирован на рисунке 4.3.

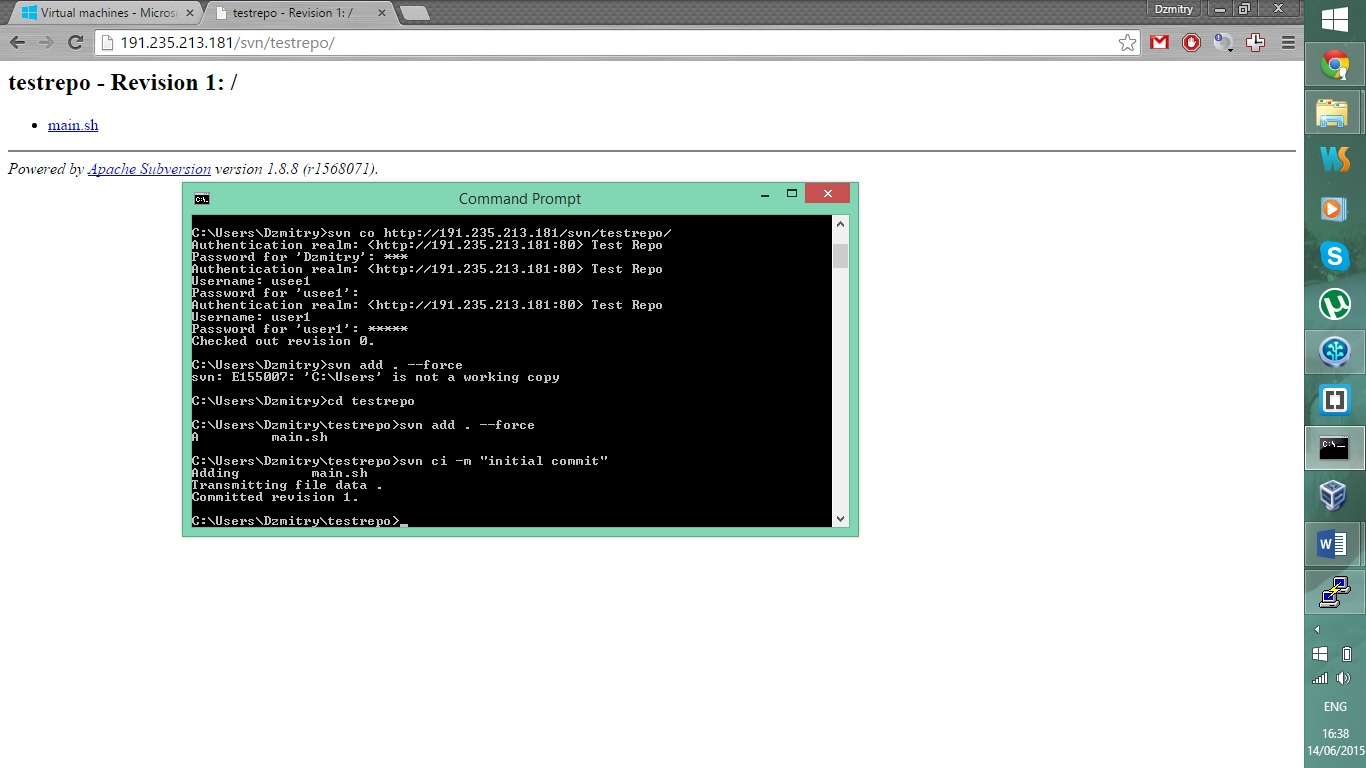


Рисунок 4.3 – Общий вид окна с работой команды svn ci

После этого можно заходить на сайт, который работает параллельно с самой svn, и посмотреть результат операции. Так, сайт должен показать текущую версию Subversion, текущую ревизию и файлы, находящиеся в ней. Работающий сайт под Apache Subversion (порт 80) можно наблюдать на рисунке 4.4.

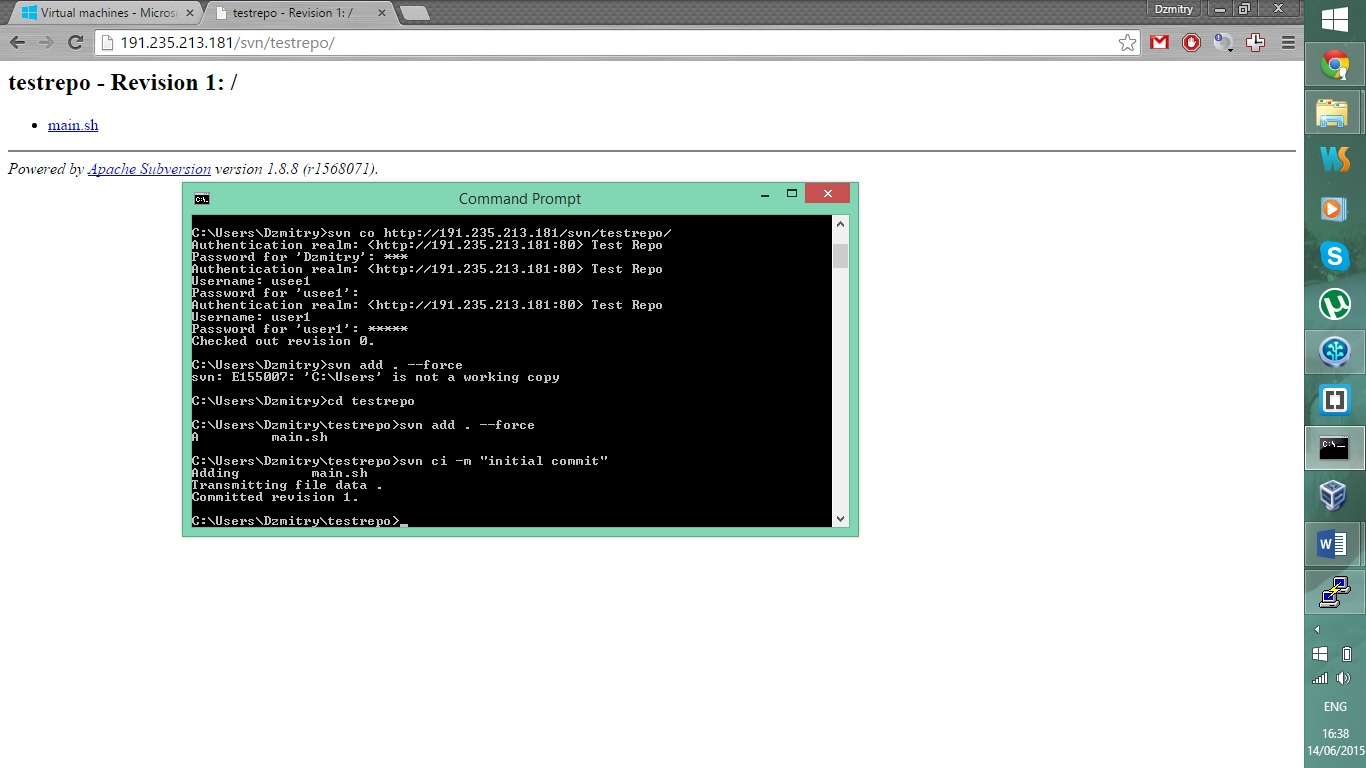


Рисунок 4.4 – Ожидаемый результат работы команды svn ci

Таким образом, зная всего лишь команды svn checkout, svn add и svn ci, можно работать с репозиторием под управлением Subversion. Именно поэтому Subversion считается самой лёгкой системой для изучения и дальнейшей разработки; тем ни менее, при выполнении более сложных действий и при работе в команде, состоящей из большого количества людей, может быть лучше воспользоваться другими системами контроля версий.

**4.2 Работа в Mercurial**

Рассмотрим работу с Mercurial на примере работы с репозиторием через SourceTree. Данная программа была выбрана потому, что она является самым популярным средством (после командной строки) для работы с Mercurial и Git. Также следует отметить, что традиционно активные пользователи Windows пользуются в основном графическими интерфейсами, поэтому, после изучения основных функций той или иной СКВ, разработчики часто переходят на более простые инструменты.

Начало работы с Mercurial (также, как и в случае с любой другой системой контроля версий), можно начать с команды clone. Для работы с данной СКВ используется утилита hg, поэтому полная команды будет выглядеть как hg clone <адрес\_репозитория> <путь\_на\_локальном\_компьютере>. Вывод команды hg clone и пример полной версии команды можно наблюдать на рисунке 4.5; при работе из консоли результат должен быть таким же.

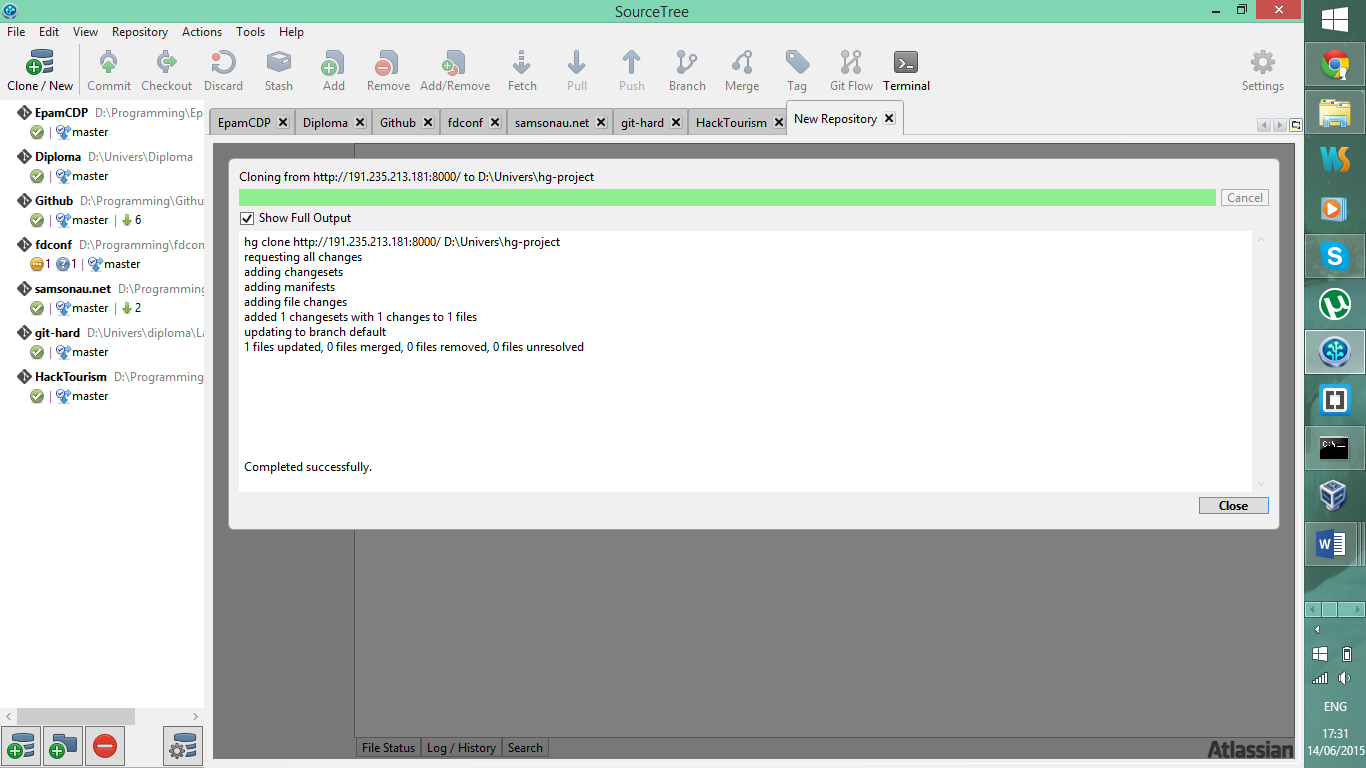


Рисунок 4.5 – Результаты работы команды hg clone в SourceTree

В результате данной операции должна появиться папка .hg, с содержимым, близким к тому, что показан на рисунке 4.6.

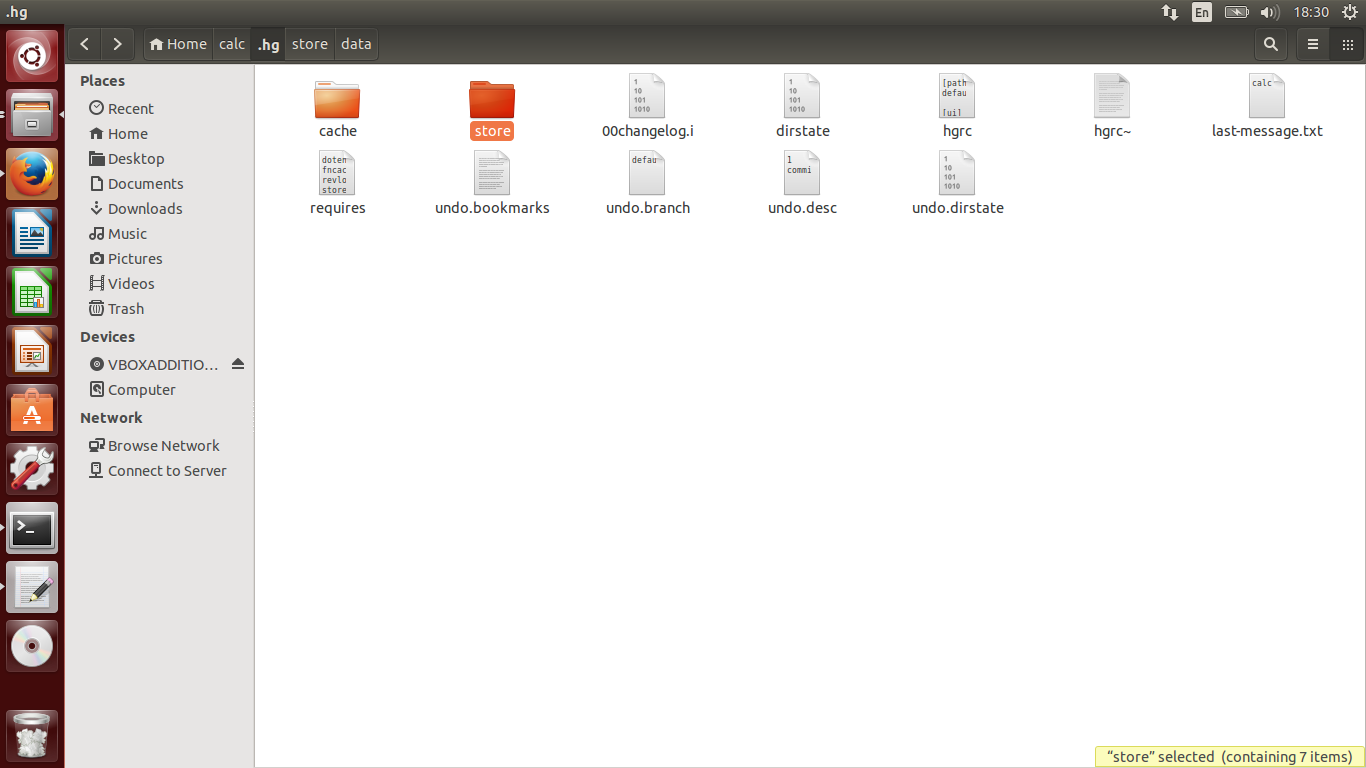


Рисунок 4.6– Содержимое .hg-папки после выполнения hg clone

В результате выполнения данной команды мы получим готовую к работе директорию с сервера. После этого нужно будет последовательно выполнить команды hg add (чтобы добавить новые файлы), hg commit (чтобы получить пакет изменений, готовых к отправке на сервер) и hg push (чтобы отправить данные изменения). Как видим, вся разница состоит в том, что появилась ещё одна команда hg push. Такое разделение позволяет держать коммиты не отправляя их на сервер до тех пор, пока они не понадобятся. Ожидаемые результаты всех трёх операций можно увидеть на рисунке 4.7.

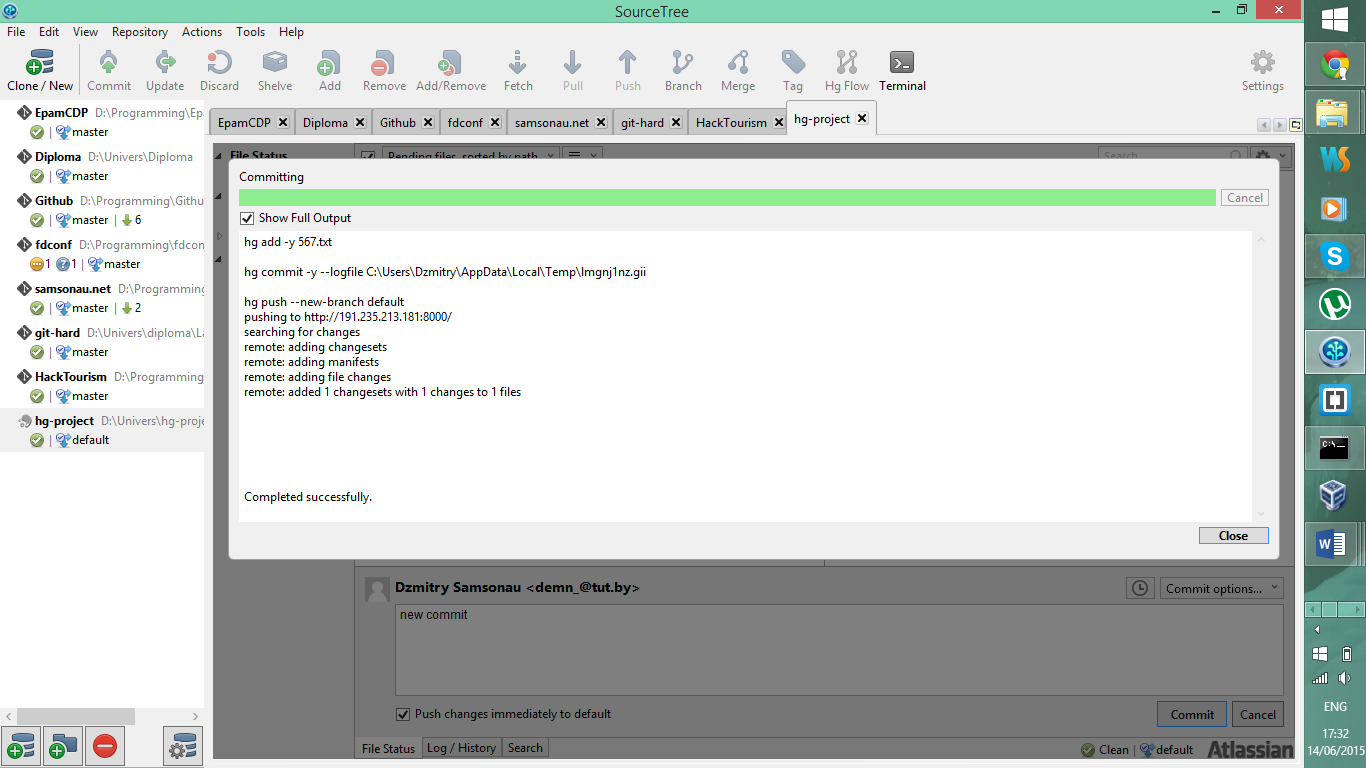


Рисунок 4.7 – Ожидаемые результаты работы команд add, commit и push

При этом на сервере мы сможем наблюдать все запросы к данной директории (в данном случае через командную строку, как на рисунке 4.8).

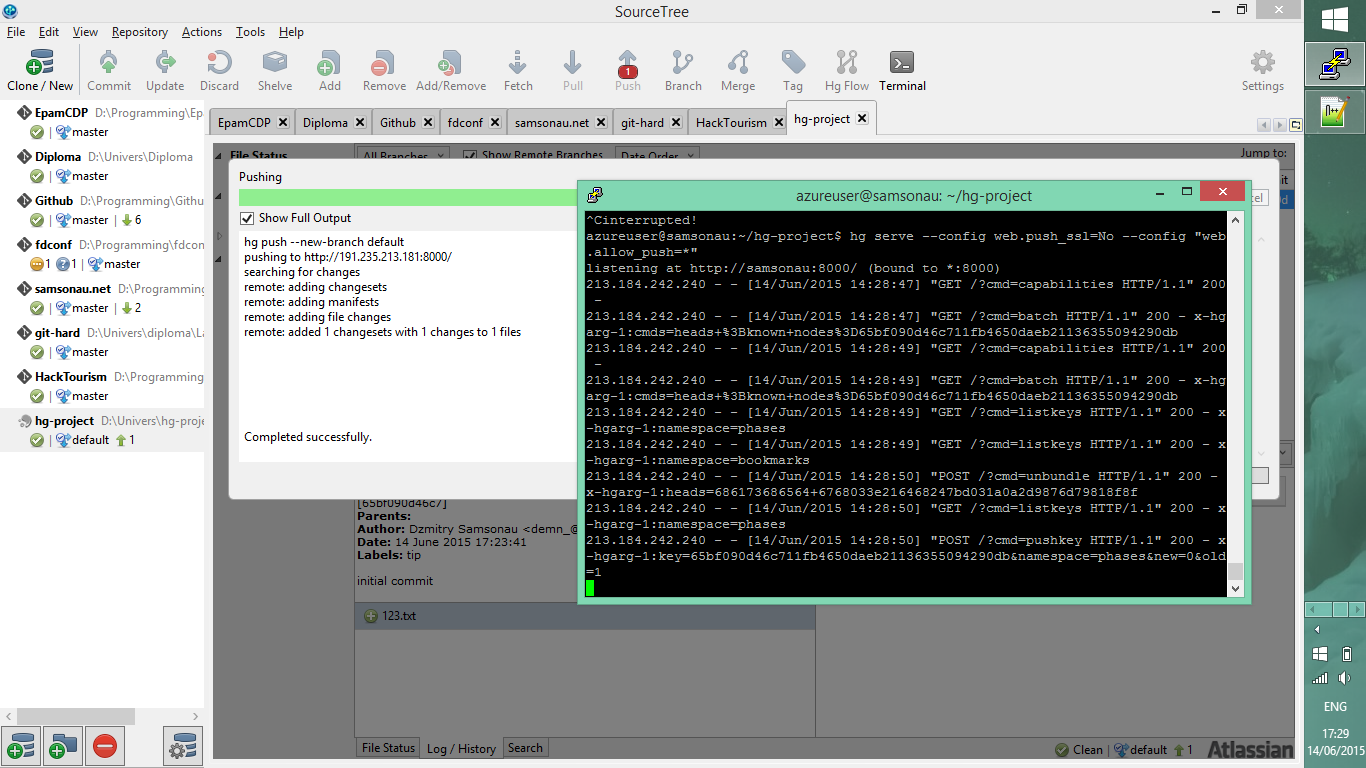


Рисунок 4.8 – Просмотр запросов в PuTTY-агенте

Следует отметить, что в отличие от Subverison, в Mercurial структура собственной папки СКВ (а именно .hg) более понятна. Так, в частности, по пути .hg/store/data можно посмотреть все список всех файлов, которые хранятся в системе. Разумеется, они закодированы и просмотреть их содержимое не удастся, однако можно визуально оценить правильность выполнения коммита по наличию или отсутствию файлов. Пример содержимого папки data после выполнения git push продемонстрирован на рисунке 4.9.

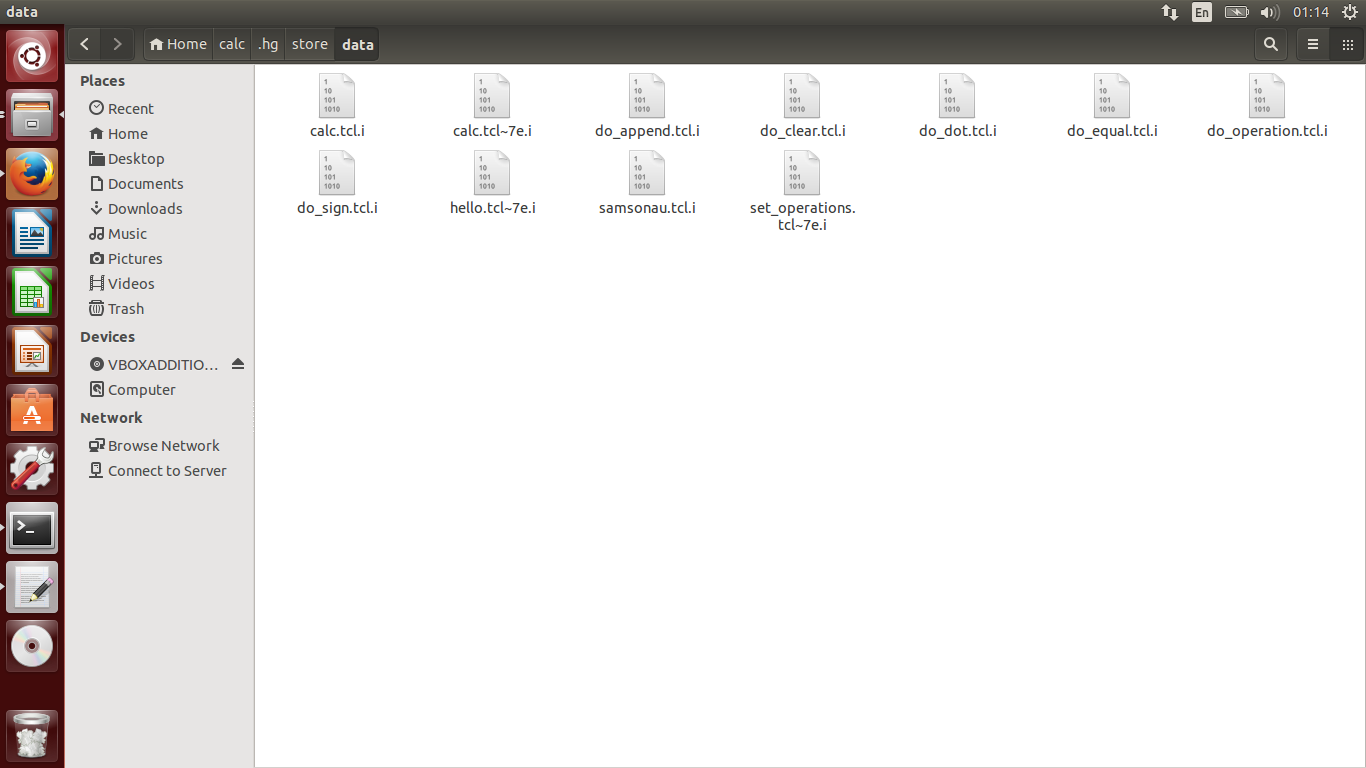


Рисунок 4.9 – Просмотр содержимого папки .hg

Кроме этого, все изменения можно посмотреть на сайте (рисунок 4.10), который доступен по адресу сервера и специальному HTTP порту (в случае с сервером Microsoft Azure – 8000).

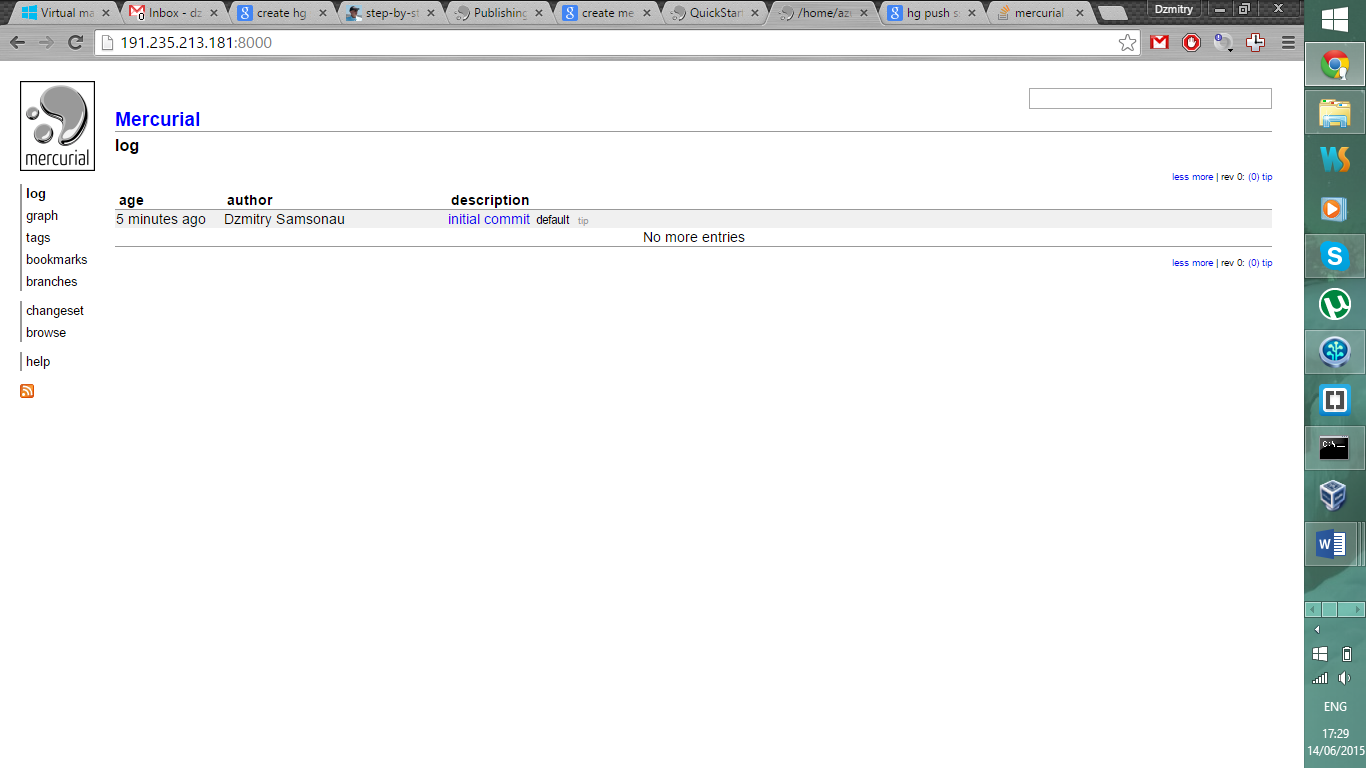


Рисунок 4.10 – Просмотр всех изменений в репозитории

Кроме обычной функции просмотра коммита, на сайте есть также возможности отображения графа изменений, выбора ветвей, в которых нужно посмотреть активность, а также многие другие, полный список которых доступен во вкладке help.

Таким образом, работа с Mercurial похожа на работу с SVN, с той лишь внешней разницей, что центральный сервер может быть установлен даже на машине разработчика. Внутри различий гораздо больше, в их число входят и разница между ревизиями и версиями файлов, другая реализация функции нахождения разницы между двумя файлами, а также вопросы авторизации пользователей. Все они находятся за рамками данных лабораторных работ, поэтому в сценарий взаимодействия пользователя с системой эти различия не входят.

**4.3 Работа в системе Git**

Работа в системе Git приблизительно похожа на работу в системе Mercurial. В нём также есть функции clone, add, commit и push, которые делают те же функции. Поэтому было бы логично рассмотреть другие сценарии взаимодействия пользователя с Git системой.

Одним из таких сценариев является работа с ветками. В Git это команда checkout. Полный синтаксис этой команды выглядит как git checkout <имя\_ветки>, а ожидаемый результат её выполнения показан на рисунке 4.11. На первый взгляд может показаться, что ничего не изменилось, однако в файловой системе все файлы уже поменяли своё содержимое на то, что было в ветке bstu.

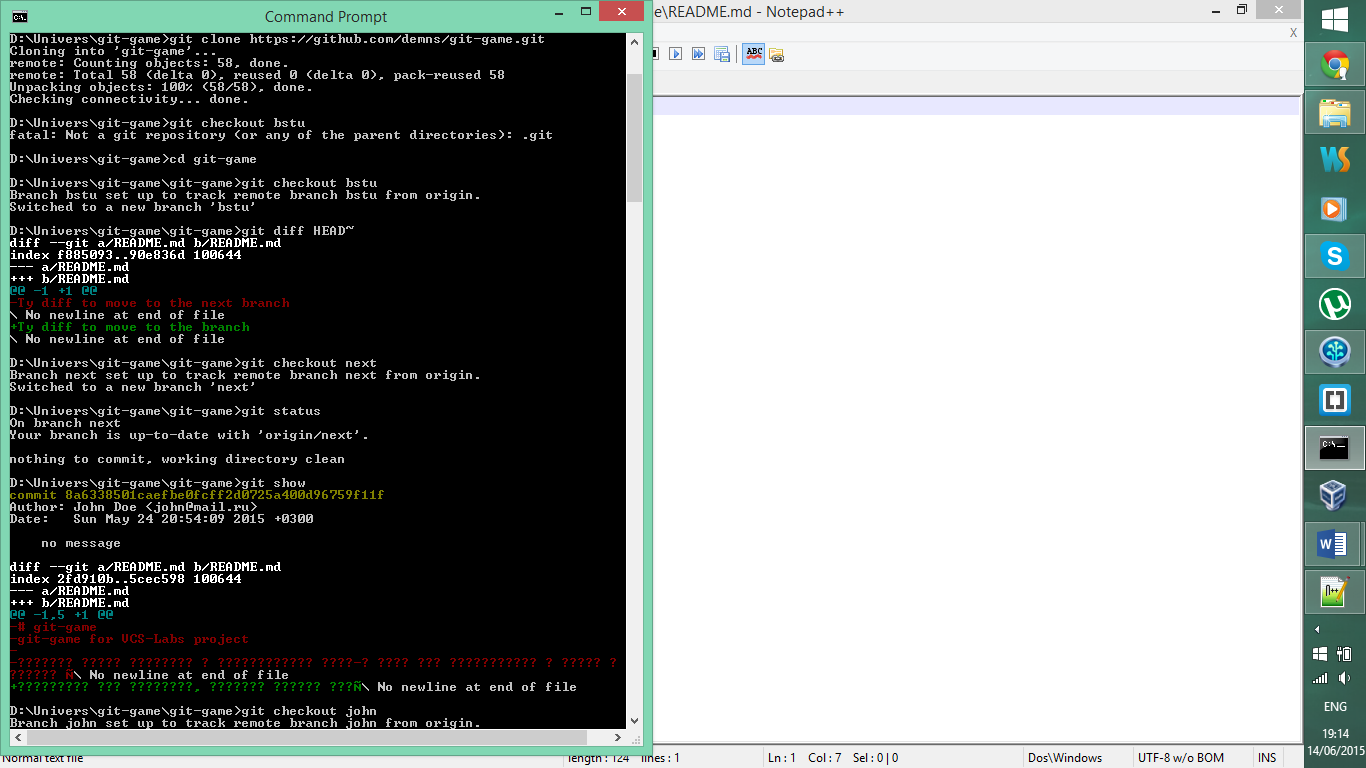


Рисунок 4.11 – Выполнение команды git checkout

Далее можно использовать при разработке команду git diff. Эта команда показывает разницу между коммитами, причём в зависимости от параметров, передаваемых этой функции, можно выбрать самые разнообразные коммиты. Так, если выбрать в качестве аргументов два hash-кода коммита, то можно увидеть разницу между двумя абсолютно любыми правками. Если же выбрать HEAD~, то можно увидеть разницу между текущим и предыдущим коммитом. Именно это функция является самой популярной в случае с git diff; ожидаемый результат показан на рисунке 4.12.

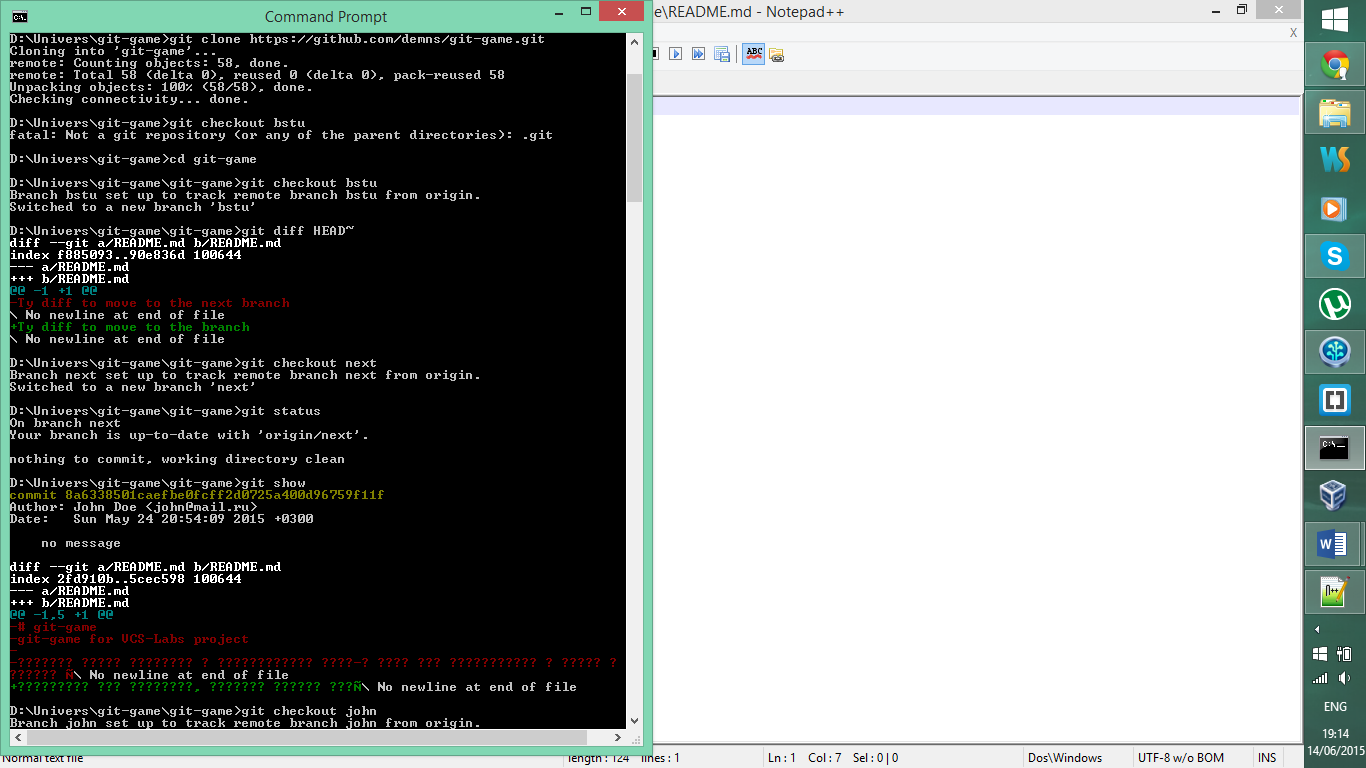


Рисунок 4.12 – Выполнение команды git diff

В данном случае можно увидеть, что в файле README.md (который по умолчанию является описанием директории в git-репозитории), было сделано изменение: в более новой ревизии было убрано слово next.

Теперь, переключившись на ветку next, нам захочется посмотреть кто и что в ней делал последним. Это делается через команду git show, которая показывает всю доступную информацию о верхнем (HEAD) коммите. В список такой информации входит имя и почтовый ящик человека, который совершил коммит, дата его создания (именно создания, а не отправки на сервер), сообщение, с которым был отправлен данный коммит (при его наличии), а также изменения, которые были сделаны в нём. К слову, последнее отображается при помощи той же команды git diff (как можно видеть на рисунке 4.13), только применяется она не ко всему коммиту целиком, а только к изменённым файлам. Учитывая то, что эта операция в Git является очень простой, она выполняется практически без затрат производительности.

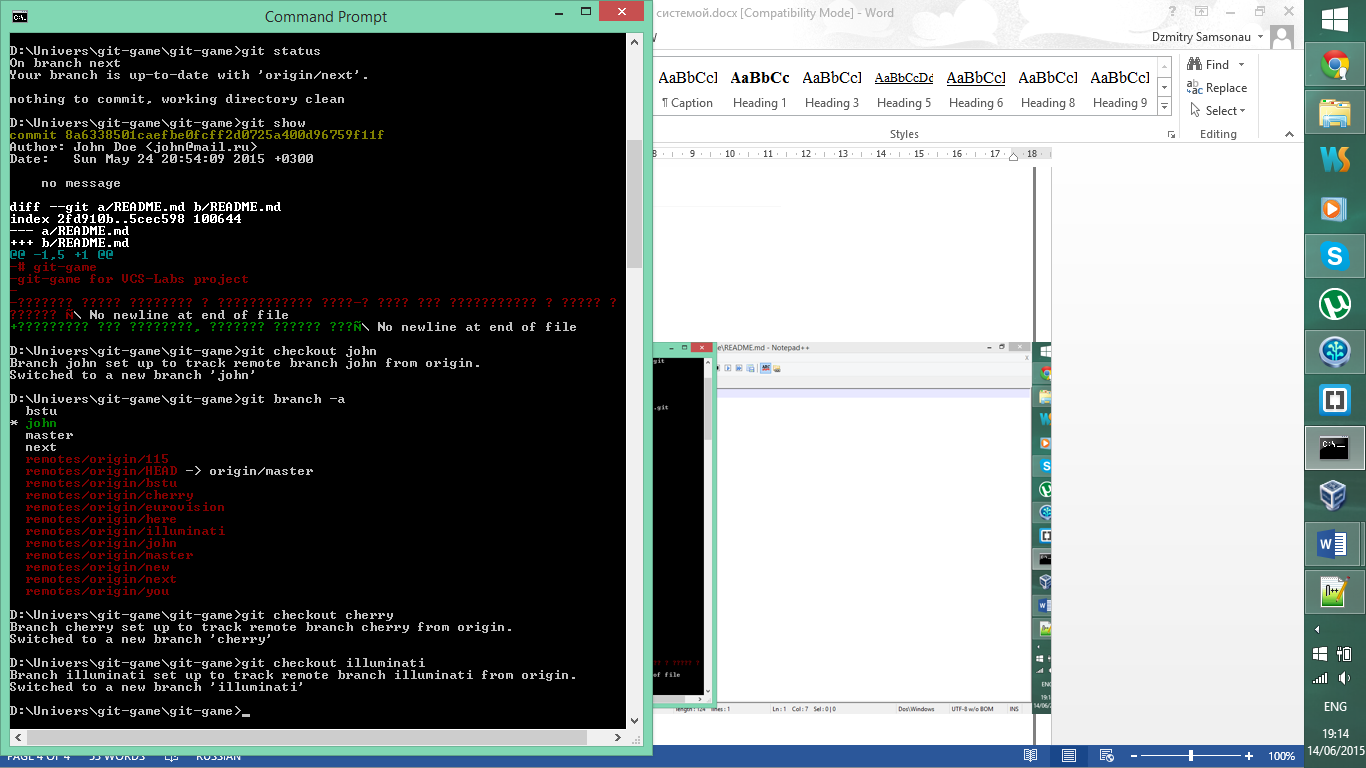


Рисунок 4.13 – Выполнение команды git show

Также в больших проектах бывает не лишним посмотреть, а какие же вообще ветки есть в репозитории. Это делается командой git branch с параметром –a (пример на рисунке 4.14).

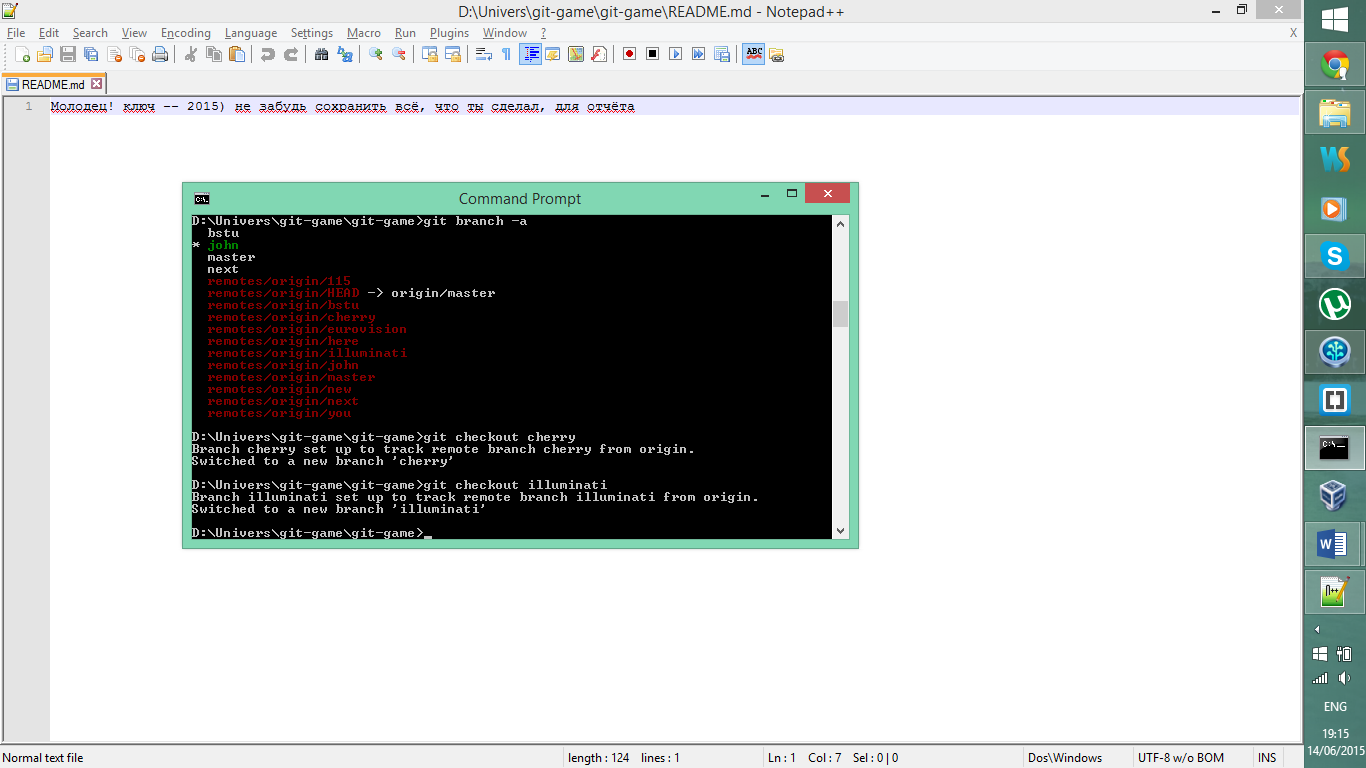


Рисунок 4.14 – Выполнение команды git branch

Следует отметить, что все рассмотренные команды в каждой СКВ – это далеко не полный список. Тем ни менее, в рамках цикла из 4-х лабораторных работ количество представленных выше команд достаточно для того, чтобы понять общий смысл каждой из систем.