**3 РАЗРАБОТКА ИНФОРМАЦИОННОГО ОБЕСПЕЧЕНИЯ**

Для проведения лабораторного практикума были выбраны Subversion, Mercurial и Git в качестве систем контроля версий и tcl/tk, zenity и HTML в качестве примеров для написания программ или разметки.

**3.1 Обмен данными в системах контроля версий**

В данном дипломном проекте при первоначальной разработке использовался аккаунт Microsoft Azure. В связи с тем, что качество интернет-соединения было достаточно невысокое (по сравнению с требованиями Azure), и оно на настоящий момент не может предоставить скорости, которой будет достаточно для качественного отображения удалённого рабочего стола, было принято решение тестировать и проводить отладку программного комплекса лабораторного практикума с использованием программы PuTTY, которая взаимодействует с удалённым сервером через SSH (без использования удалённого рабочего стола).

SSH (англ. Secure Shell — «безопасная оболочка») — сетевой протокол прикладного уровня, позволяющий производить удалённое управление операционной системой и туннелирование TCP-соединений (например, для передачи файлов). Схож по функциональности с протоколами Telnet и rlogin, но, в отличие от них, шифрует весь трафик, включая и передаваемые пароли. SSH допускает выбор различных алгоритмов шифрования. SSH-клиенты и SSH-серверы доступны для большинства сетевых операционных систем.

SSH позволяет безопасно передавать в незащищённой среде практически любой другой сетевой протокол. Таким образом, можно не только удалённо работать на компьютере через командную оболочку, но и передавать по шифрованному каналу звуковой поток или видео. Также SSH может использовать сжатие передаваемых данных для последующего их шифрования.

Кроме этого, при разработке использовался SSH-туннель — туннель, создаваемый посредством SSH-соединения и используемый для шифрования туннелированных данных. Он используется для того, чтобы обезопасить передачу данных в Интернете (аналогичное назначение имеет IPsec). При пересылке через SSH-туннель незашифрованный трафик любого протокола шифруется на одном конце SSH-соединения и расшифровывается на другом.

Все эти меры предосторожности были нужны потому, что аккаунт в Azure является платным, а в случае перехвата злоумышленниками пары логин-пароль и последующего несанкционированного доступа к аккаунту разработчика, поддержка Azure-аккаунта закрывается (однако, злоумышленник может потратить достаточно большое количество средств, которые всё ещё будут на счету разработчика).

Кроме непосредственно разработки и взаимодействия с удалённой ОС необходимо было использовать сетевой протокол для общения с системами контроля версий. Все используемые в лабораторном практикуме системы контроля версий умеют работать с четырьмя сетевыми протоколами для передачи данных: локальный, Secure Shell (SSH), HTTP и собственный (для каждой из систем, SVN, Git или Mercurial). Важно понимать, что за исключением протокола HTTP, все эти протоколы требуют, чтобы система контроля версий была установлена и работала на сервере.

Базовым является локальный протокол, при использовании которого удалённый репозиторий — другой каталог на диске. Наиболее часто он используется, если все члены команды имеют доступ к общей файловой системе, например, к NFS, или, что менее вероятно, когда все работают на одном компьютере. Последний вариант не столь хорош, поскольку все копии вашего репозитория находятся на одном компьютере, делая возможность потерять всё более вероятной.

Наиболее часто используемый транспортный протокол — это SSH. Причина этого в том, что доступ по SSH уже есть на многих серверах, а если его нет, то его очень легко настроить. Кроме того, SSH — единственный из сетевых протоколов, предоставляющий доступ и на чтение, и на запись. Два других сетевых протокола (HTTP и Git) в большинстве случаев дают доступ только на чтение, поэтому даже если они вам доступны, вам всё равно понадобится SSH для записи. К тому же SSH это протокол с аутентификацией, и благодаря его распространённости обычно его легко настроить и использовать.

Огромный плюс протоколов HTTP и HTTPS в простоте их настройки. По сути, всё, что необходимо сделать — поместить пустой репозиторий внутрь каталога с HTTP документами и установить перехватчик post-update. В таком случае любой, имеющий доступ к веб-серверу, на котором был размещён репозиторий, может его склонировать.

Одним из примеров собственных протоколов является Git-протокол. Так, вместе с Git'ом поставляется специальный демон, который слушает порт 9418 и предоставляет сервис, схожий с протоколом SSH, но абсолютно без аутентификации. Чтобы использовать Git-протокол для репозитория, необходимо создать файл git-daemon-export-ok, иначе демон не будет работать с этим репозиторием, но следует помнить, что в протоколе отсутствуют средства безопасности. Соответственно, любой репозиторий в Git'е может быть либо доступен для клонирования всем, либо не доступен никому. Как следствие, обычно нельзя отправлять изменения по этому протоколу. При этом можно открыть доступ на запись, но из-за отсутствия авторизации в этом случае кто угодно, зная URL вашего проекта, сможет его изменить.

При составлении лабораторного практикума был использован протокол SSH, как наиболее популярный, доступный и действенный способ соединения.

**3.2 Форматы файлов при использовании систем контроля версий**

В данном цикле лабораторных работ используются такие системы контроля версий, в которых при сохранении новых версий на сервере используется дельта-компрессия: система находит отличия новой версии от предыдущей и записывает только их, избегая дублирования данных. Таким образом, на сервере будут лежать уже сжатые файлы; прочитать из них информацию без предварительной переработки не получится.

Теперь рассмотрим ситуацию на других клиентах.

В случае системы контроля версий Git в каталоге .git находится вся нужная информация про файлы и папки, а в корне самой директории – непосредственно текущая версия репозитория. Подробный пример можно увидеть на рисунке 3.1.

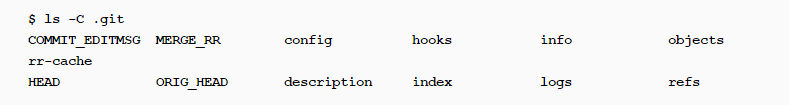


Рисунок 3.1 – Содержание папки .git на клиенте

В папке с базой данных объектов хранятся преимущественно каталоги, состоящие из двух символов – первых двух букв хэша sha1 объекта, хранящегося в Git. Внутри лежат сжатые и закодированные данные про объекты, хранящиеся в Git. Пример содержимого папки object можно увидеть на рисунке 3.2.

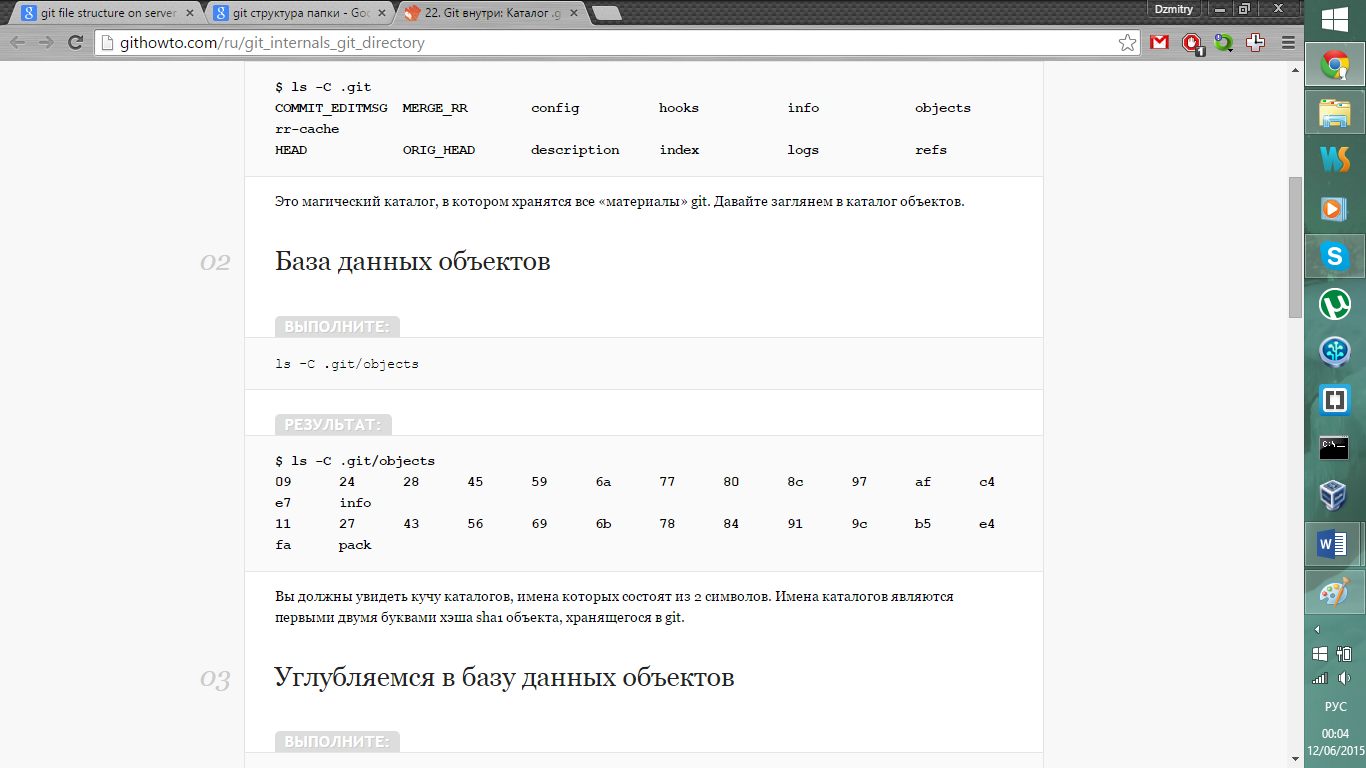


Рисунок 3.2 – Содержание папки object на клиенте

При работе с Mercurial структура каталога будет другой. Так, когда Mercurial отслеживает изменения файла, он сохраняет историю этого файла в объекте метаданных называемом filelog. Каждая запись в filelog содержит достаточно информации чтобы восстановить одну ревизию отслеженного файла. Filelog'и хранятся в виде файлов в папке .hg/store/data. Они содержат два вида информации: данные о ревизиях и индексы, помогающие Mercurial эффективно искать ревизии.

Для большого или имеющего длинную историю файла filelog хранится раздельно в файлах с данными (расширение .d) и с индексом (расширение .i). Для маленьких файлов с небольшой историей ревизионные данные и индекс хранятся в едином файле с расширением .i. Связь между файлом в рабочей директории и filelog'ом, который отслеживает его историю в хранилище, показана на рисунке 3.3.

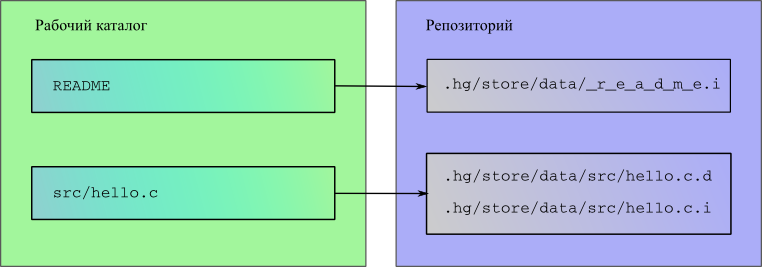


Рисунок 3.3 – Связь между файлами в рабочей директории и filelog'ом

Для сбора и хранения информации об отслеживаемых файлах Mercurial использует структуру называемую манифест. Каждый раздел в манифесте содержит информацию о файлах, входящих в один набор изменений. В разделе записано, какие файлы присутствуют в наборе изменений, ревизия каждого файла и некоторые другие части метаданных файла.

Для хранения внутренней информации Subversion также использует собственную папку – папку .svn. В этой папке хранятся системные файлы и копии рабочих файлов до внесения нами правок. Когда запускается команда status, система SVN сверяет наши файлы с образцами, которые хранятся внутри папки .svn, поэтому для выявления изменений не нужно подключаться к удаленному серверу.

**3.3 Установка системы контроля версий Git на сервер**

В данном разделе описано, как установить Git-систему на виртуальный сервер Windows Azure Ubuntu 14.04 LTS. Несмотря на специфичность, данные шаги можно воспроизвести на любом другом сервере.

В связи с тем, что использовался удалённый сервер, на него нужно зайти при помощи PuTTY-клиента. Данный процесс показан на рисунке 3.4.

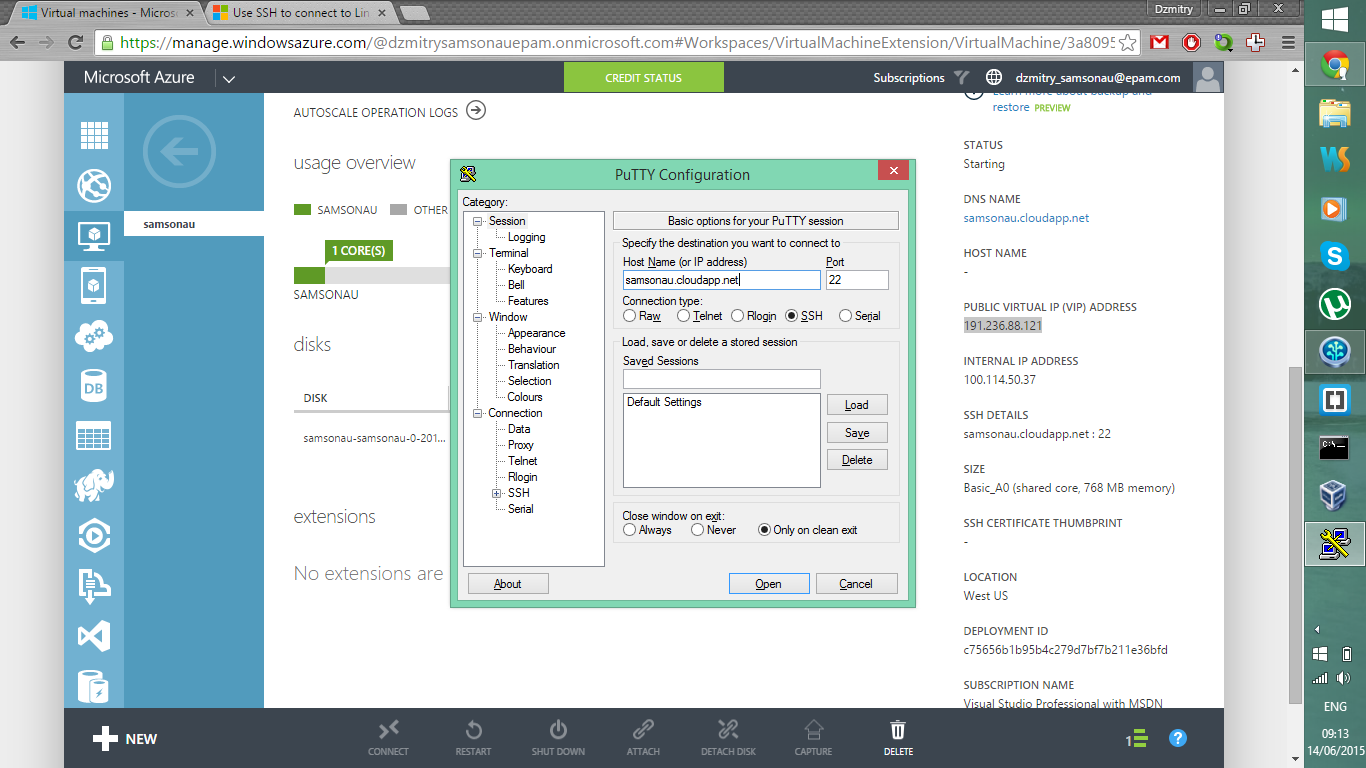


Рисунок 3.4 – Запуск PuTTY-агента

После успешного запуска и перехода на сервер, появляются несколько диагностических сообщений, после чего запускается консоль, как на рисунке 3.5.



Рисунок 3.5 – Вход на сервер

После успешного входа приступаем к установке. В случае с Git, пакет, который нам нужно установить, так и называется – git. На самом деле, это пакет git-core, который был переименован в git. Команда для установки данного пакета показана на рисунке 3.6.

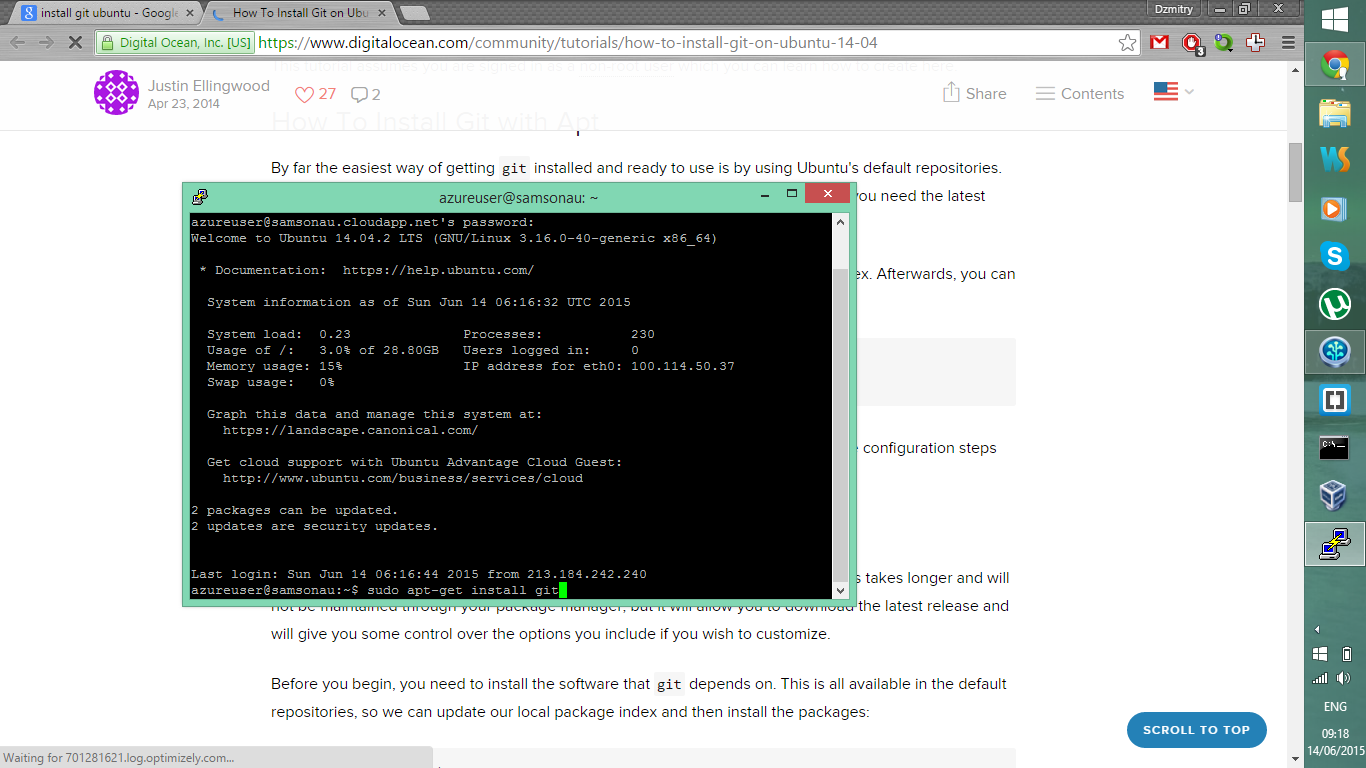


Рисунок 3.6 – Выполнение команды для установки Git на сервере

После успешного завершения установки Git, создаём пользователя с таким же именем. Это делается для того, чтобы не загрязнять другие аккаунты данными с git; традиционно под пользователем git находятся только ключи, директории и конфигурационные файлы. Пример добавления пользователя в Ubuntu можно увидеть на рисунке 3.7.



Рисунок 3.7 – Создание данных для нового пользователя git

Так как данный сервер будет использоваться только как git-хранилище, переходим к пользователю git. Операция, необходимая для этого и её результат показаны на рисунке 3.8.

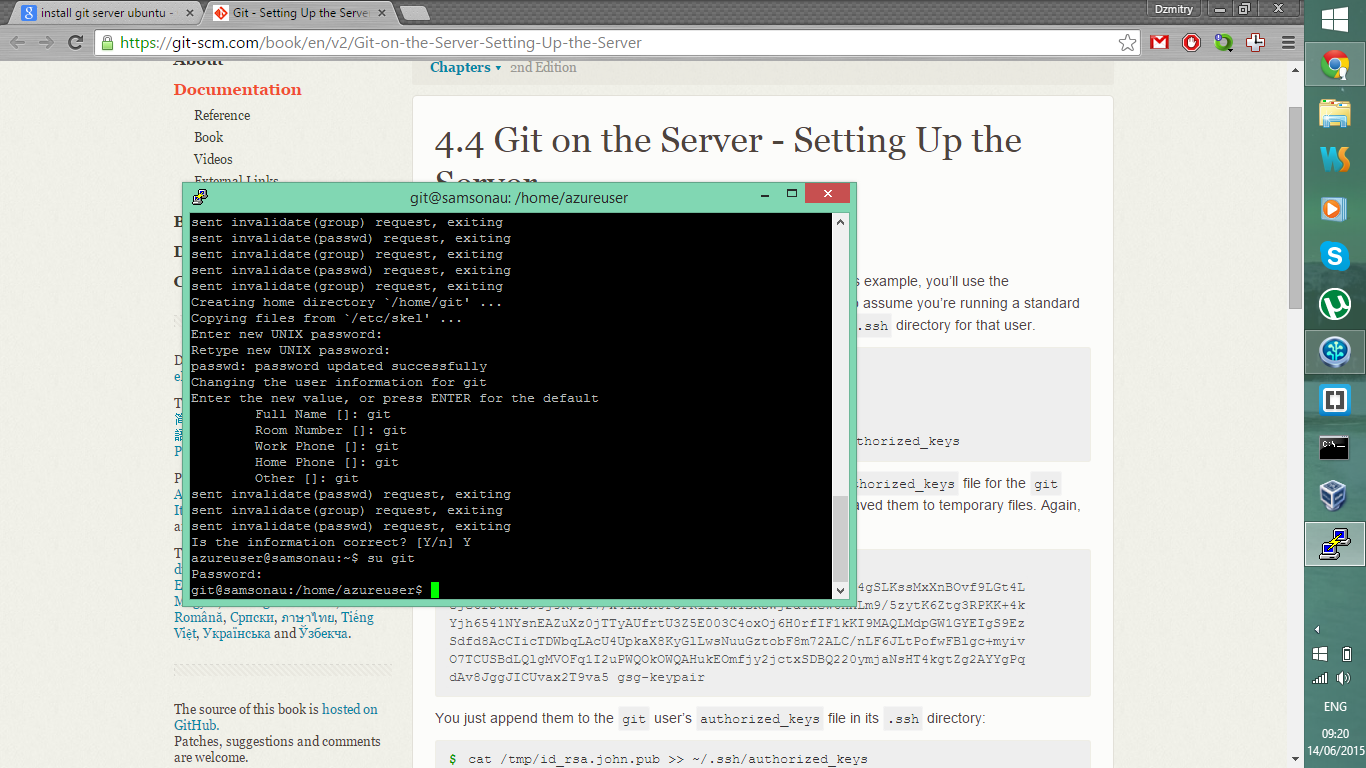


Рисунок 3.8 – Переход на работу с пользователем git

С Git, как было замечено ранее, можно работать через собственный протокол, через и через HTTP. Далее будет показана возможность взаимодействия с Git через SSH, поэтому нам понадобятся ssh-rsa ключи, которые нужно добавить в скрытую папку .ssh. Добавление файла, в котором будут храниться ключи, показано на рисунке 3.9.

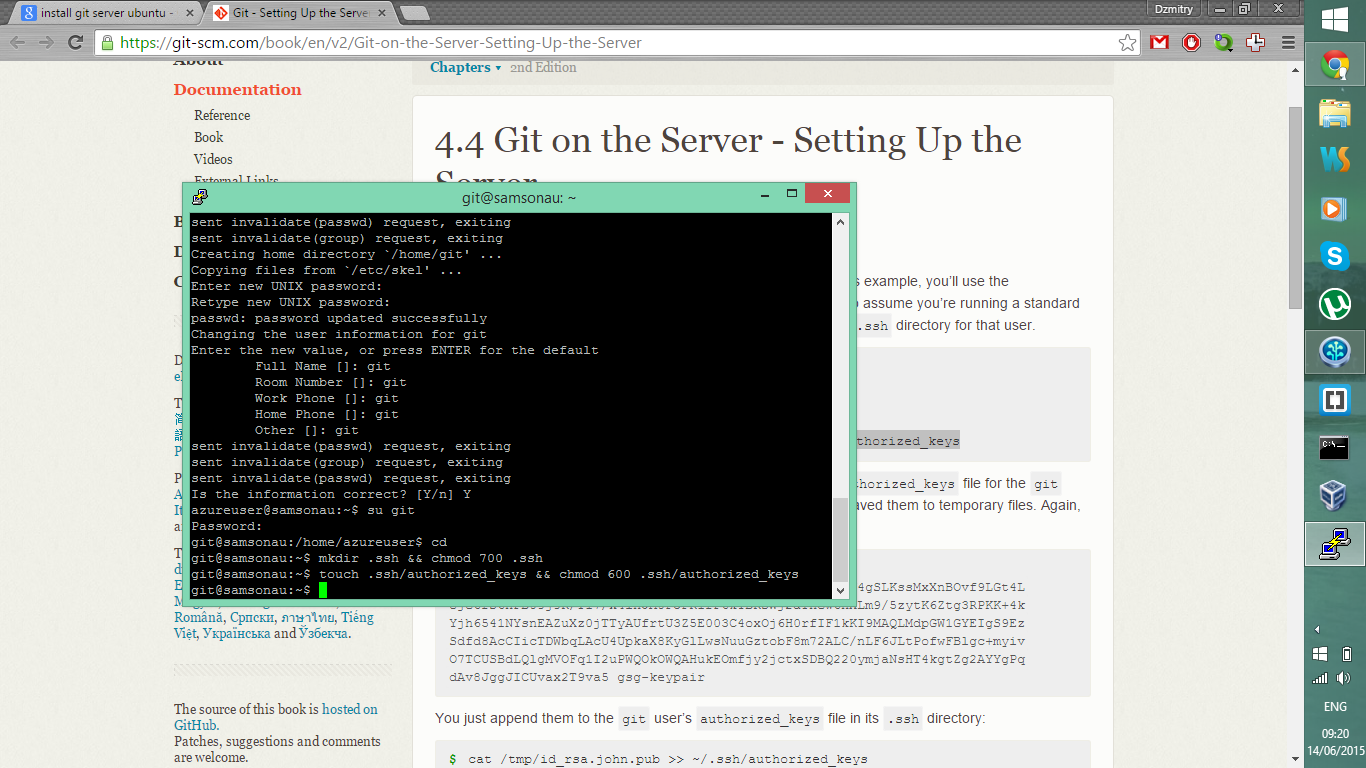


Рисунок 3.9 – Добавление файла для ключей

Далее необходимо сгенирировать ключ. Самый простой способ сделать это в Windows – это PuTTY Key Generator, который поставляется совместно с бесплатной программой SourceTree. Пример генерации ключа для файла authorized\_keys можно увидеть на рисунке 3.10.

В authorized\_keys находится список открытых ключей (RSA/DSA), который размещён на сервере (т.е. в той системе куда осуществляется подключение). Открытый ключ – это такой ключ, который может быть опубликован и используется для проверки подлинности подписанного документа, а также для предупреждения мошенничества со стороны заверяющего лица в виде отказа его от подписи документа. Открытый ключ подписи вычисляется, как значение некоторой функции от закрытого ключа, но знание открытого ключа не дает возможности определить закрытый ключ. При этом главное свойство пары открытый-закрытый ключ заключается в том, что по секретному ключу легко вычисляется открытый ключ, но по известному открытому ключу практически невозможно вычислить секретный.

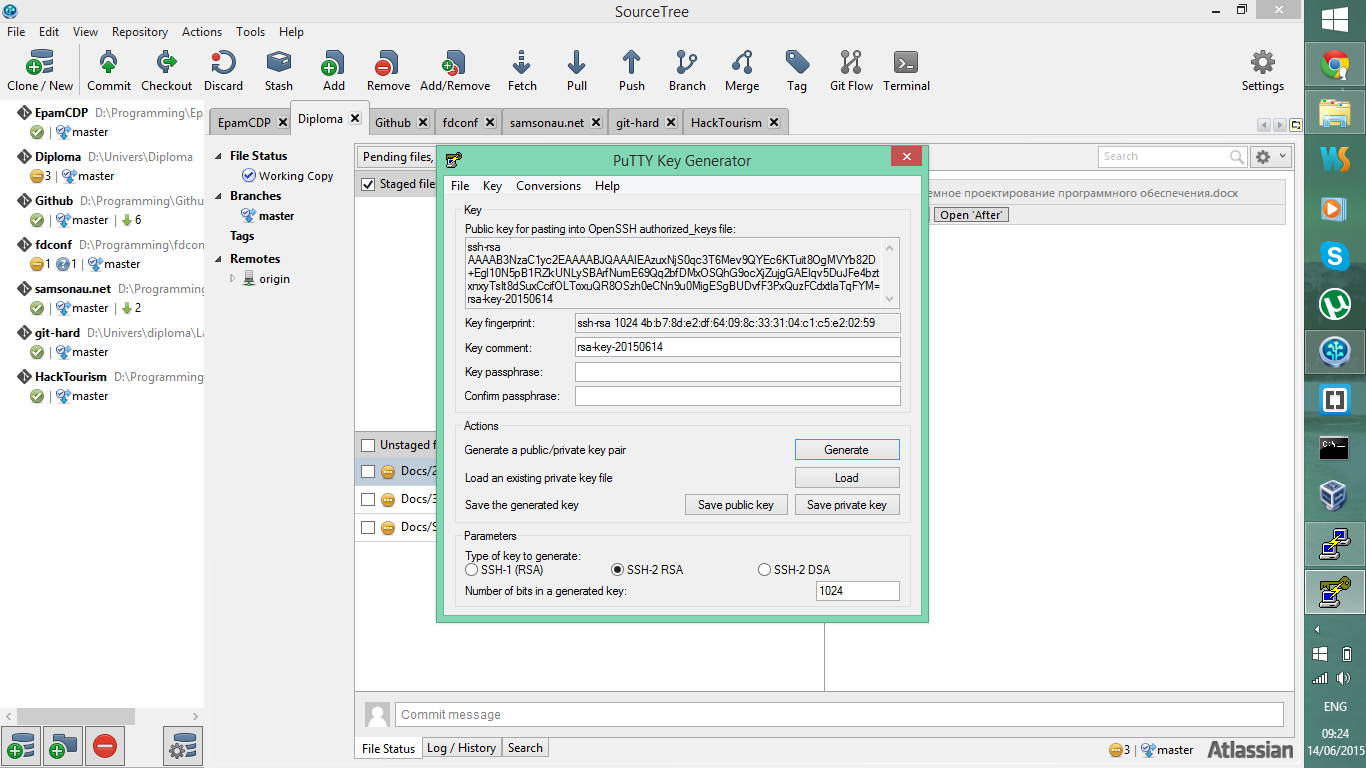


Рисунок 3.10 – Создание ключа в PuTTY Hey Generator

После генерации строки с открытым ключом её нужно добавить в файл на сервере. Один из способов сделать это – на рисунке 3.11.

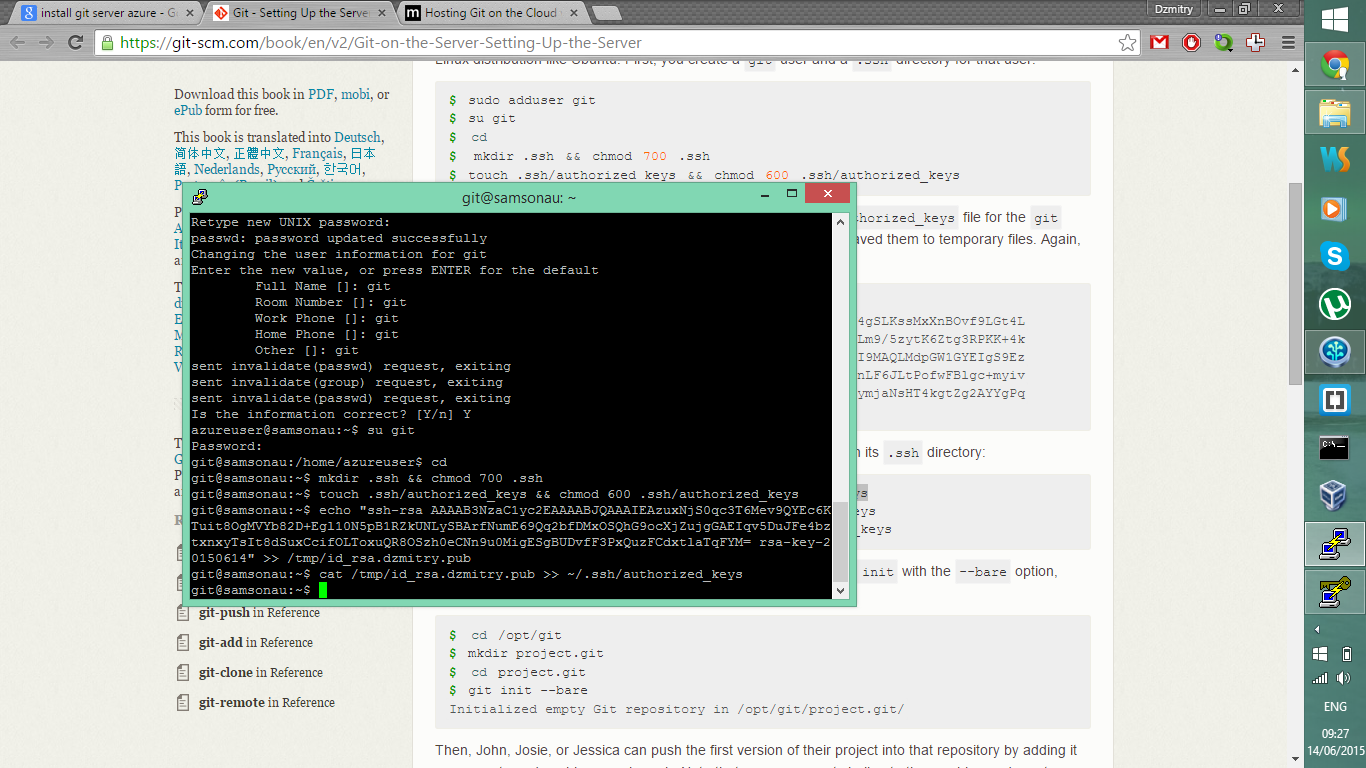


Рисунок 3.11 – Добавление ключа в файл

Далее проверим правильность заполнения файла. Это также можно было сделать при помощи консоли, однако лучше установить пакет mc, который даст возможность пользования программной Midnight Commander, который является визуальным файловым менеджером. Результат работы команды просмотра файла, вызванной нажатием клавиши F3, показан на рисунке 3.12.

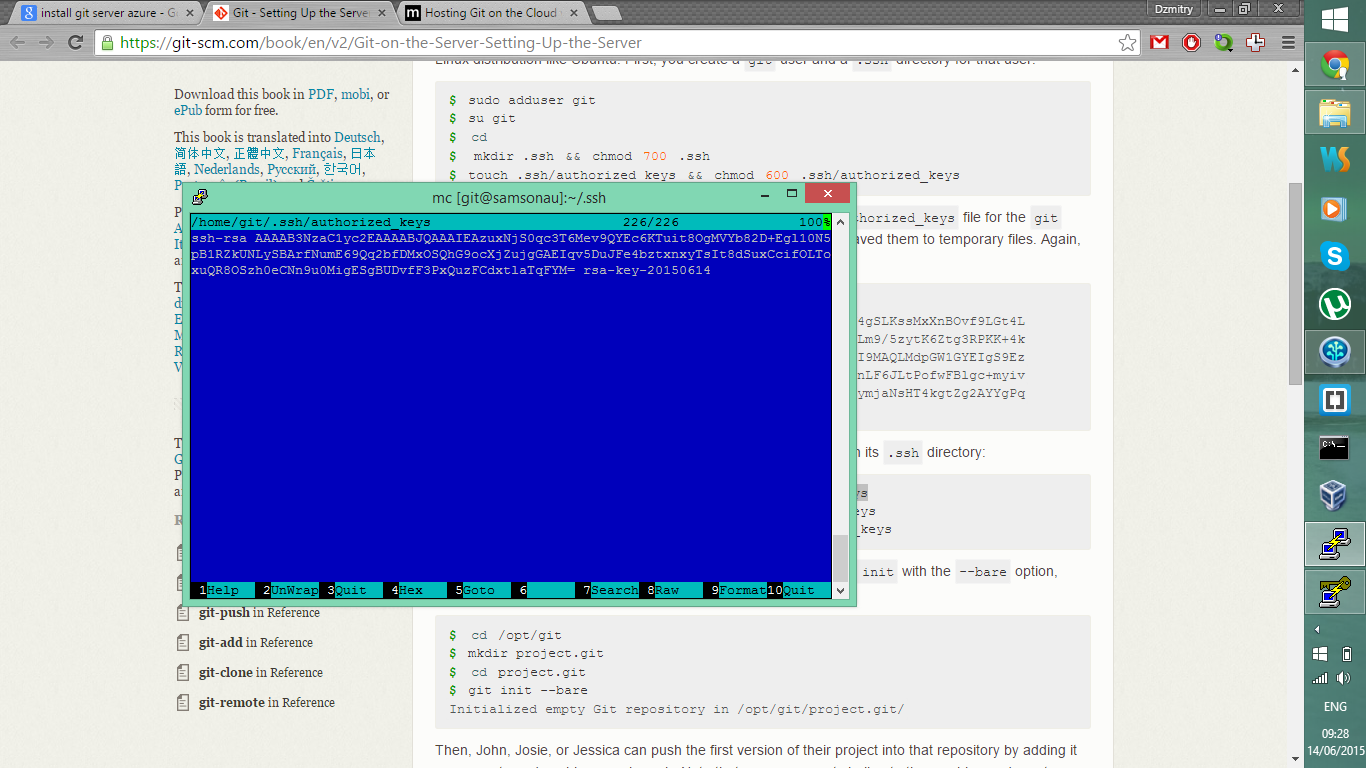


Рисунок 3.12 – Проверка правильности выполнения команды

Если всё сделано правильно, то теперь можно создать центральный Git-репозиторий на сервере. Следует отметить, что сделать это с клиентской машины не представляется возможным: для каждого нового проекта это нужно будет делать самостоятельно при помощи сервера. Именно поэтому в лабораторных практикумах, для облегчения задачи администраторов, нужно будет создать всего 2 Git-репозитория на каждую группу. Пример создания можно увидеть на рисунке 3.13.

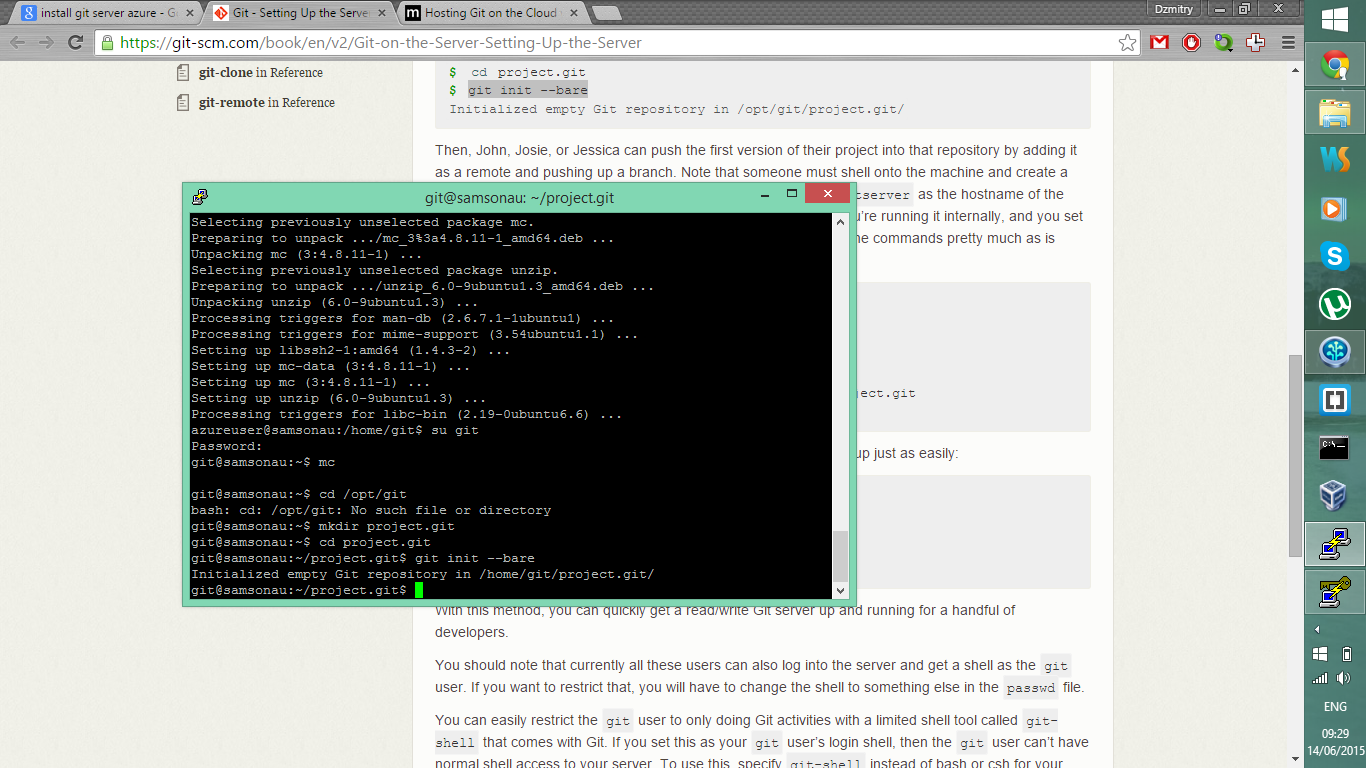


Рисунок 3.13 – Создание пустого репозитория на сервере

На этом работа на сервере завершена. Теперь посмотрим, как можно взаимодействовать с данным репозиторием на клиенте. Для начала первому учащемуся нужно будет инициализировать репозиторий и сделать первый коммит. Пример такого коммита и процесс инициализации показан на рисунке 3.14.

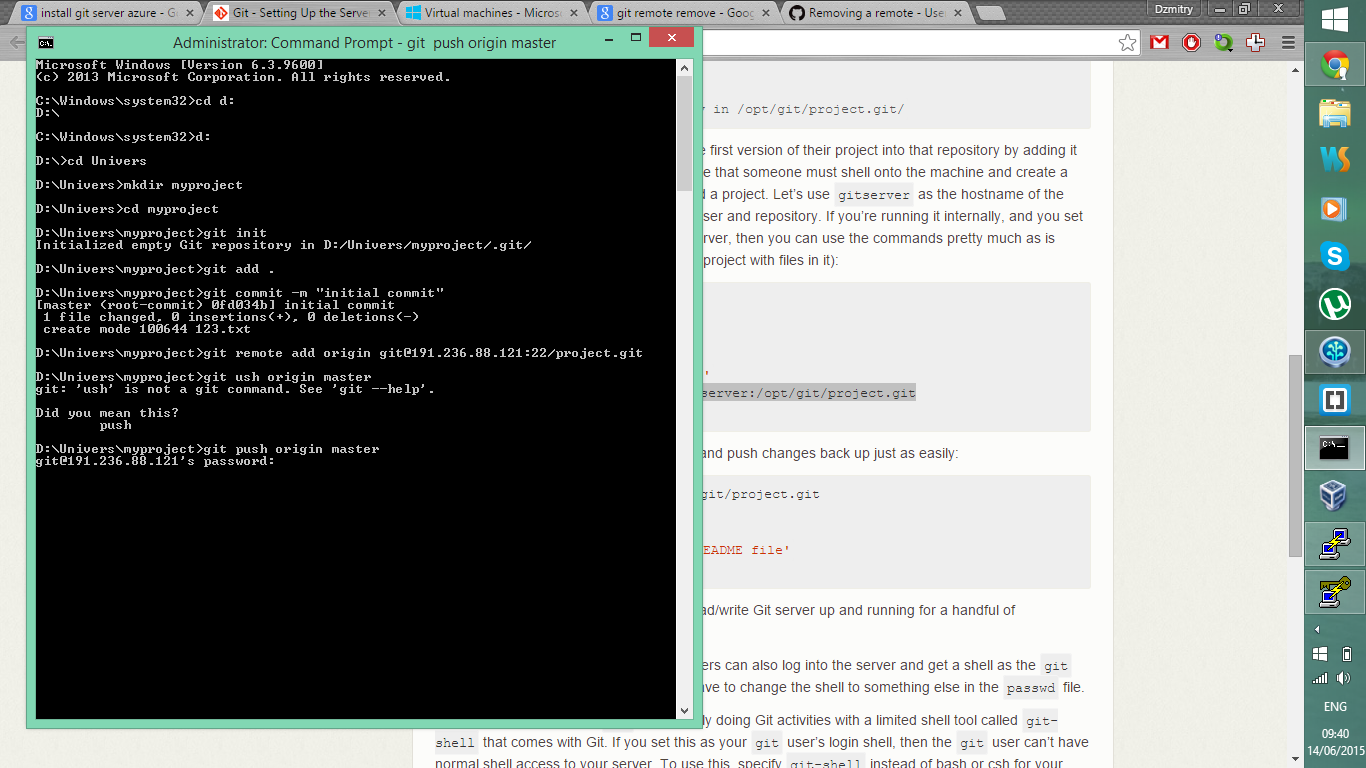


Рисунок 3.14 – Инициализация репозитория на клиенте

После этого, зная IP-адрес сервера, на него можно добавить первый коммит. Пример добавления remote, команды push и результаты их работы представлены на рисунке 3.15.

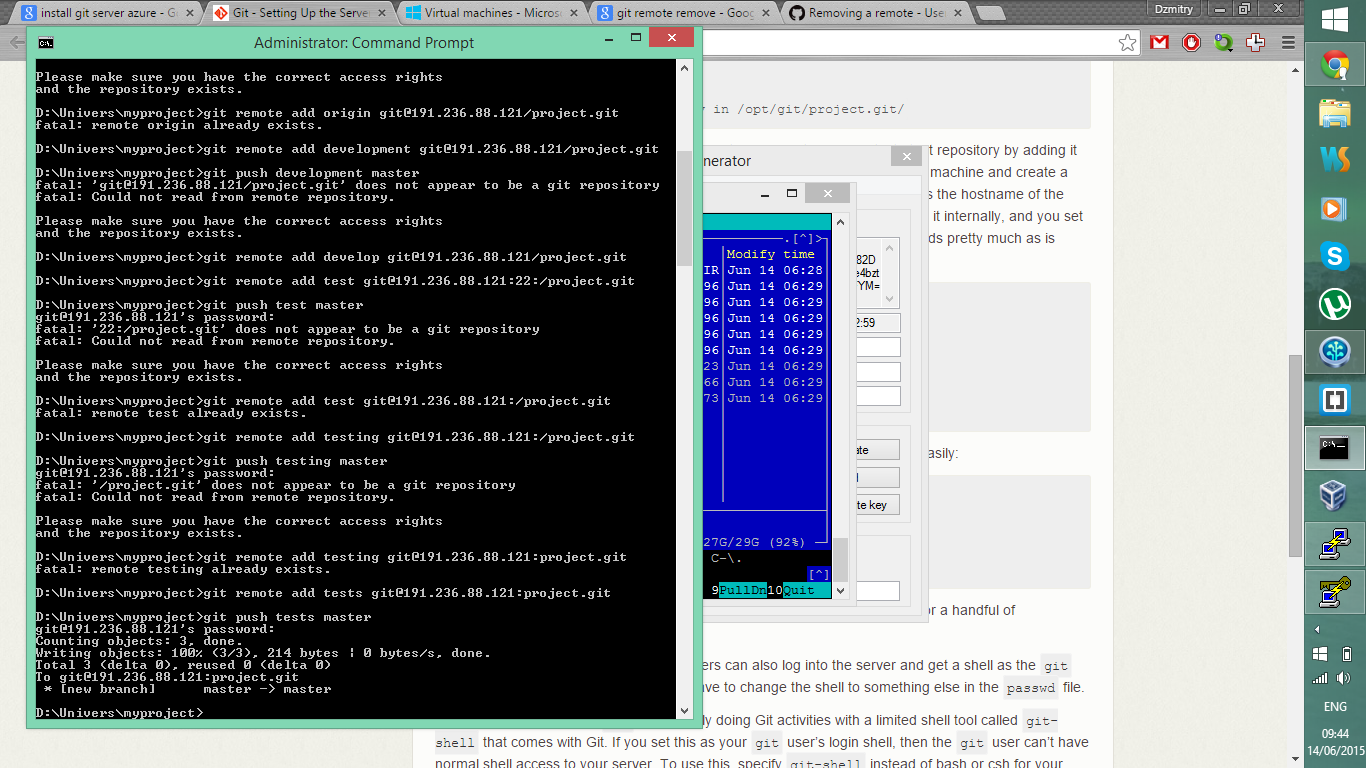


Рисунок 3.15 – Добавление изменений на сервер

Однако, взаимодействие через консоль не даст возможности оперативно посмотреть все изменения, добавить новую ветку, откатиться на старую версию и так далее. Команды Git достаточно сложны, и запомнить их все не представляется возможным. Поэтому, для проверки правильности отправления можно воспользоваться программой SourceTree, которая предоставляет визуальный интерфейс для Git и Mercurial репозиториев. Пример команды clone в данной программе представлен на рисунке 3.16.

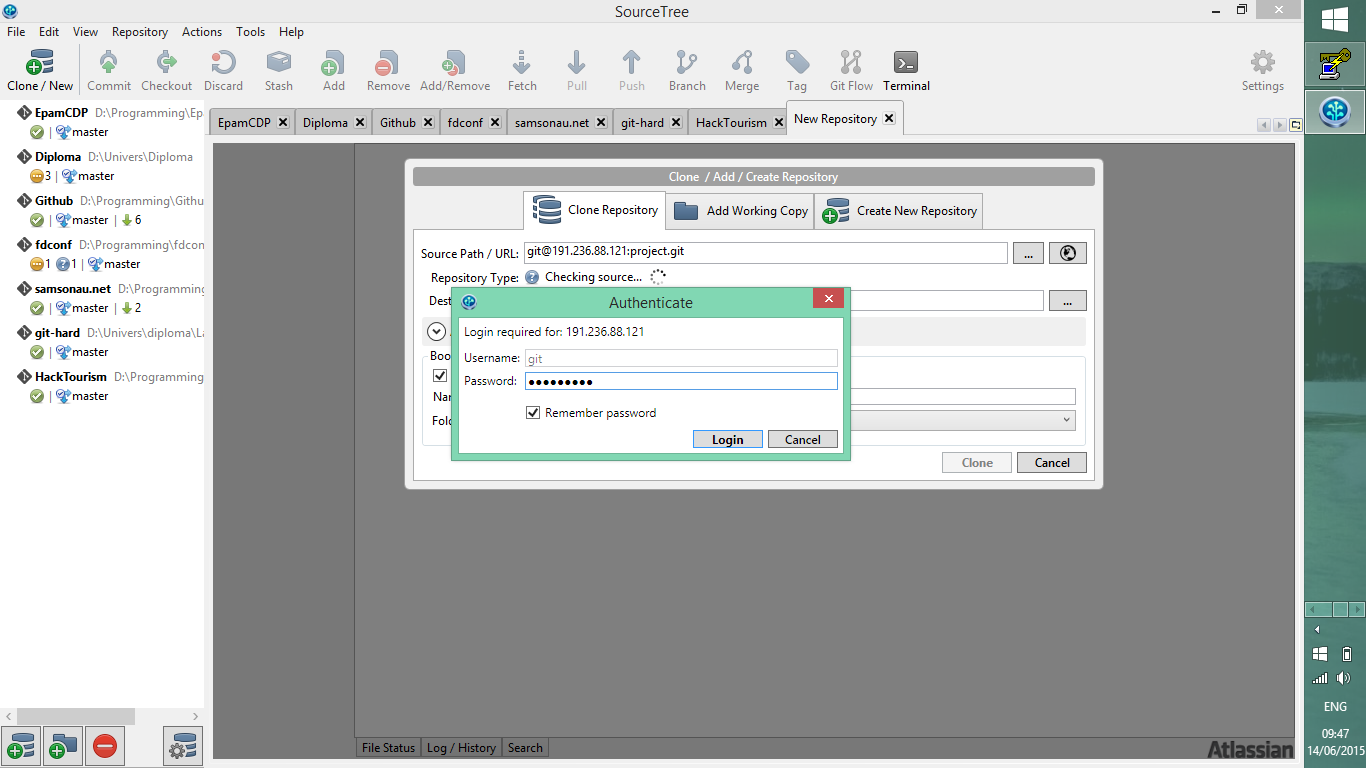


Рисунок 3.16 – Выполнение команды clone в SourceTree

На рисунке 3.16 было выполнено клонирования репозитория на рабочую машину. Результаты выполнения этой команды и её лог показаны на рисунке 3.17.

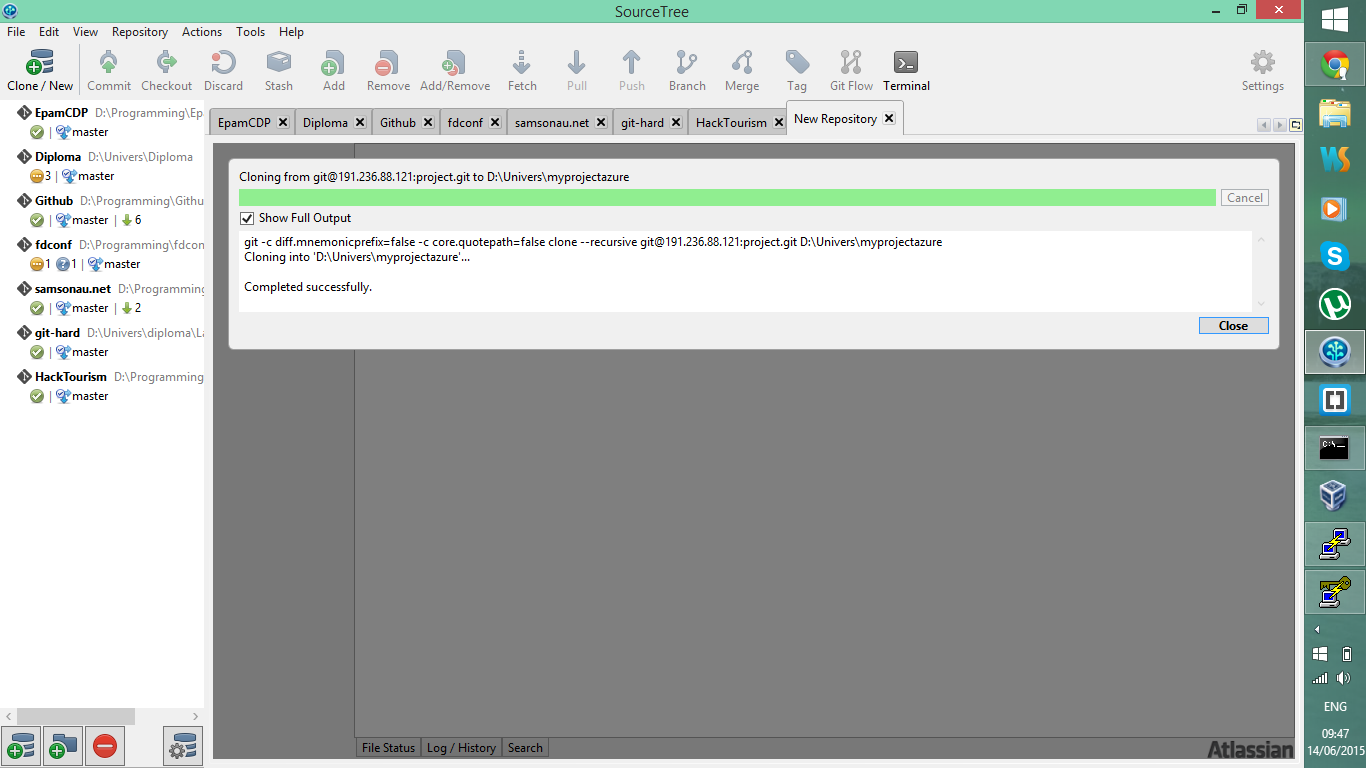


Рисунок 3.17 – Просмотр лога команды clone в SourceTree

Если всё сделано верно, то репозиторий, созданный на предыдущих шагах будет выглядеть, как на рисунке 3.18.

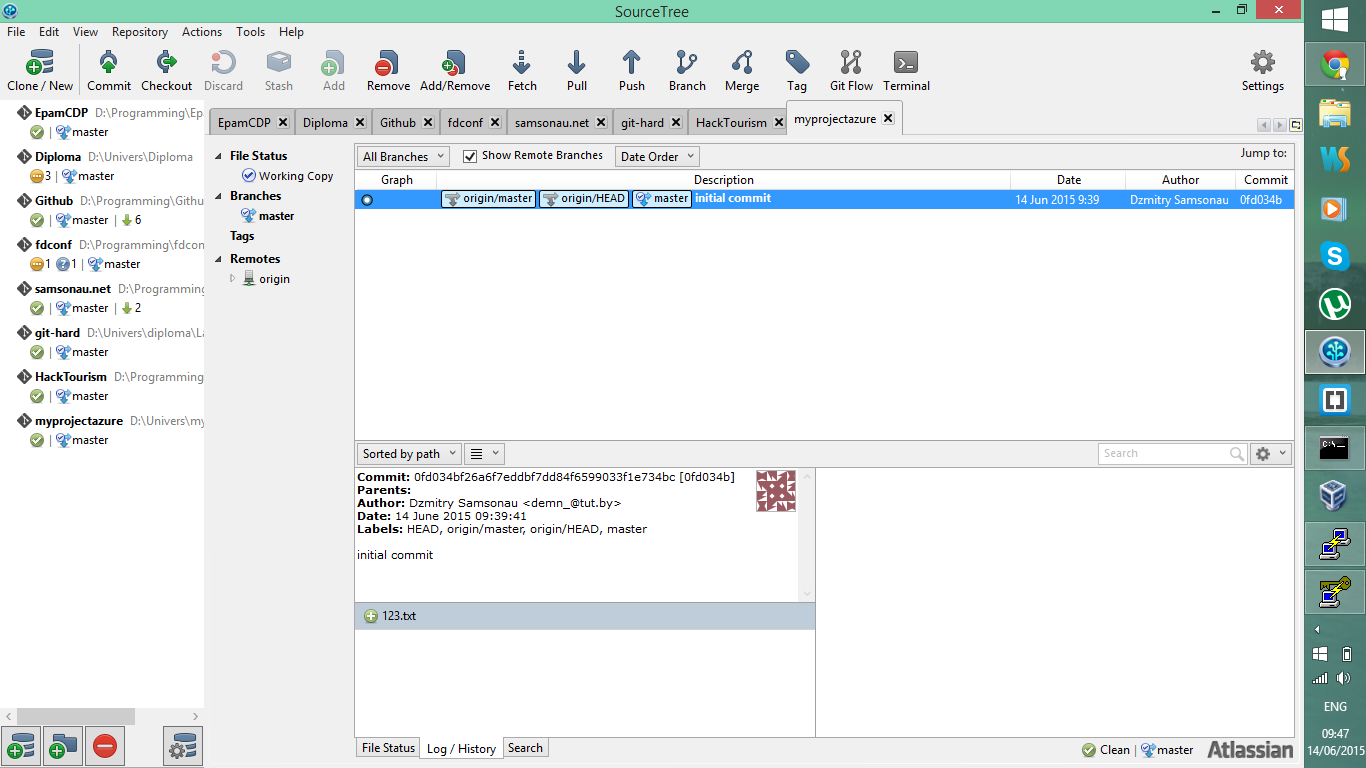


Рисунок 3.18 – Графический вид репозитория в SourceTree

В целях проверки правильности инициализации репозитория на сервере и проверки возможности работы с ним через графический интерфейс, создадим новую ветку, добавим в неё изменения и посмотрим, добавится ли она на сервере. Диалог создания новой ветки представлен на рисунке 3.19.

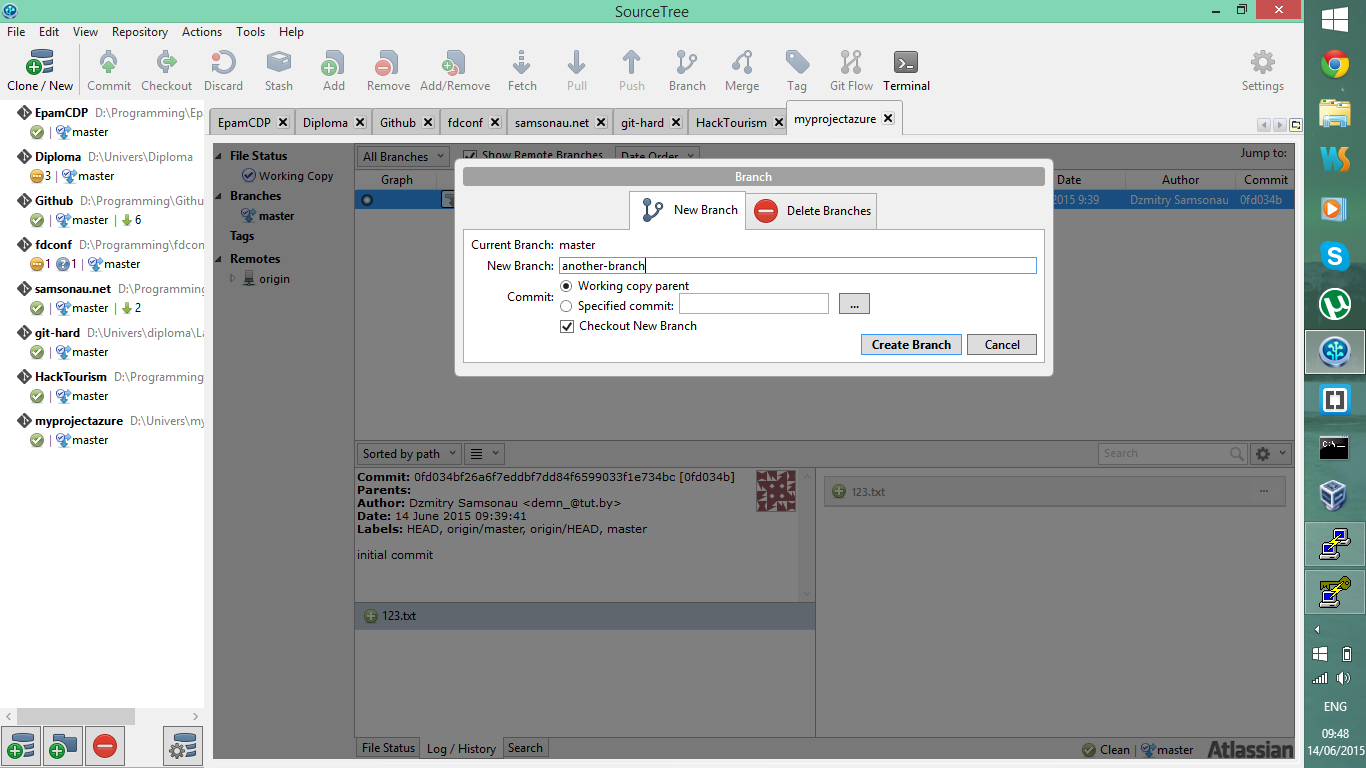


Рисунок 3.19 – Создание ветки another-branch в SourceTree

Теперь, добавив файл 456.txt в текущий репозиторий, попробуем добавить изменения на сервер. Диалог добавления (вместе с комментарием к коммиту) представлен на рисунке 3.20.

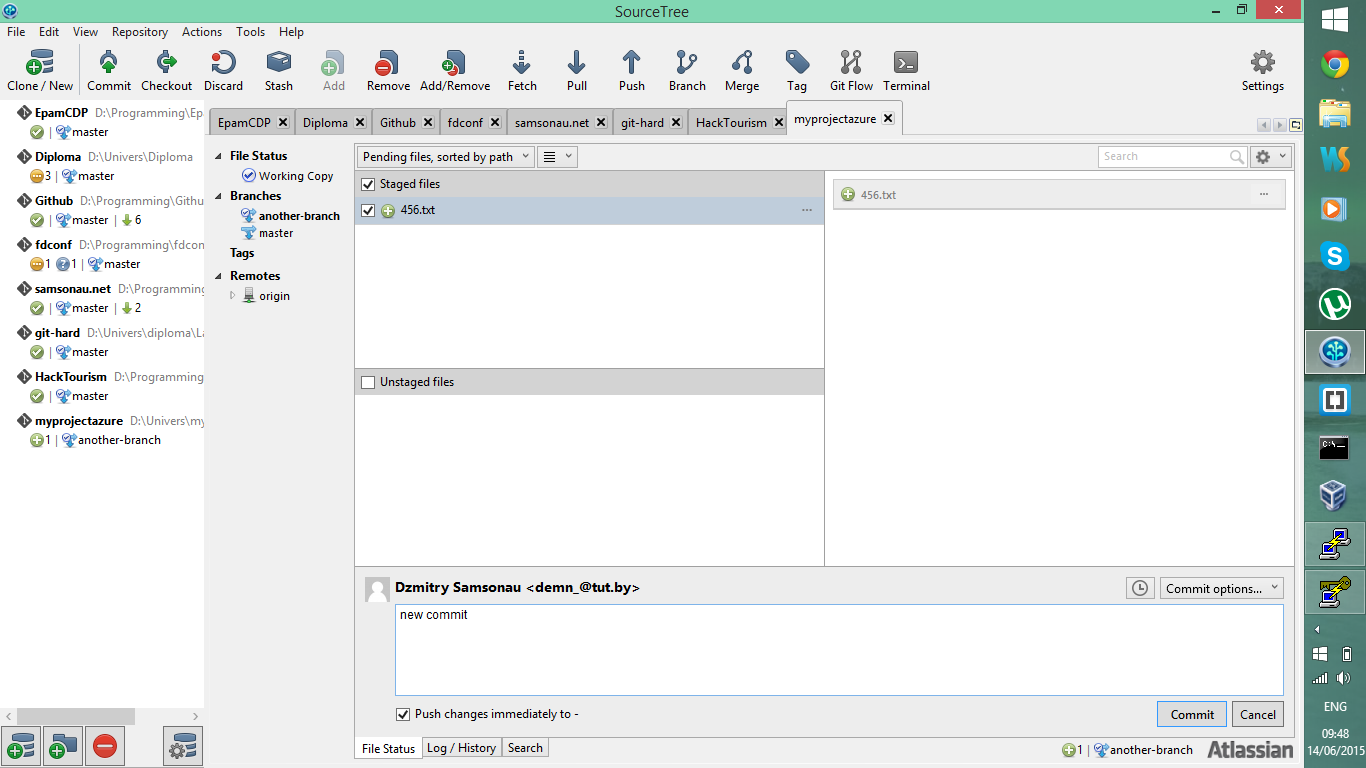


Рисунок 3.20 – Добавление файла и коммит в SourceTree

Если всё сделано правильно, то лог функций git commit и git push, выполненных последовательно, будет выглядеть, как на рисунке 3.21.

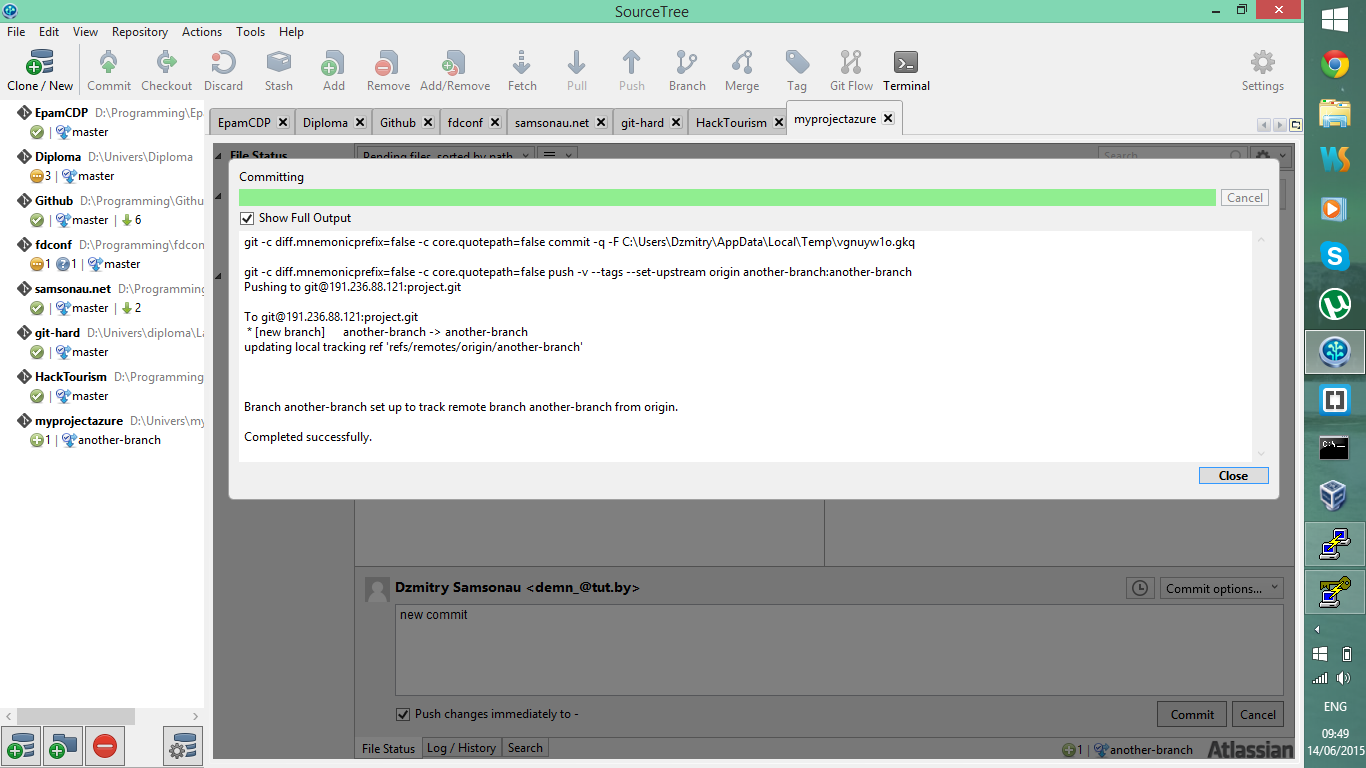


Рисунок 3.21 – Полный лог функций commit и push в SourceTree

Изменения на сервере можно посмотреть по пути project.git/refs/heads. На рисунке 3.22 показано правильное состояние рабочей директории.



Рисунок 3.22 – Проверка наличия ветки на сервере

Установка Mercurial очень похожа на процесс установки Git, однако есть несколько отличий.

В первую очередь они заключается в пакете установки (для Mercurial -- sudo apt-get install mercurial-server) и имени пользователя (hg).

Чтобы все ключи пользователей, которые имеют доступ к серверу по ssh, хранились в одном месте, то в файл /etc/ssh/sshd\_config нужно добавить строчку вида AuthorizedKeysFile /etc/ssh/keys/%u.pub, которая будет означать, что файлы ключей должны храниться в папке /etc/ssh/keys/ и иметь вид имя\_пользователя.pub.

Ключи пользователей, которые должны иметь доступ к серверу необходимо поместить в папку /etc/mercurial-server/keys/users, а ключи администраторов — /etc/mercurial-server/keys/root.

Также следует отметить, что для Mercurial существует команда hg serve, которая создаст сайт, на который можно будет заходить и смотреть текущие изменения в репозитории. Т.е. в принципе можно не устанавливать Mercurial-сервер, а сделать какой-либо из репозиториев учащихся главным. Для Git такой команды и, следовательно, возможности, не существует.

**3.4 Установка Subversion на сервер**

В данном разделе будет описано, как установить Subversion-систему. В этом примере также будем использовать сервер Ubuntu 14.04 LTS.

Для установки SVN необходимо сначала установить все необходимые библиотеки. Это делается командой sudo apt-get install subversion apache2 libapache2-svn apache2-utils. После этого необходимо создать новую папку и новый репозиторий (в данном примере -- testrepo). Это делается при помощи последовательного выполнения команд sudo mkdir -p /svn/repos/ и sudo svnadmin create /svn/repos/testrepo.

Далее нужно подменить владельца данного файла при помощи sudo chown -R www-data:www-data /svn/repos/testrepo.

Теперь создадим файл testrepo.conf по пути /etc/apache2/sites-available/ и добавим в него строки таким же образом, как показано на рисунке 3.23.

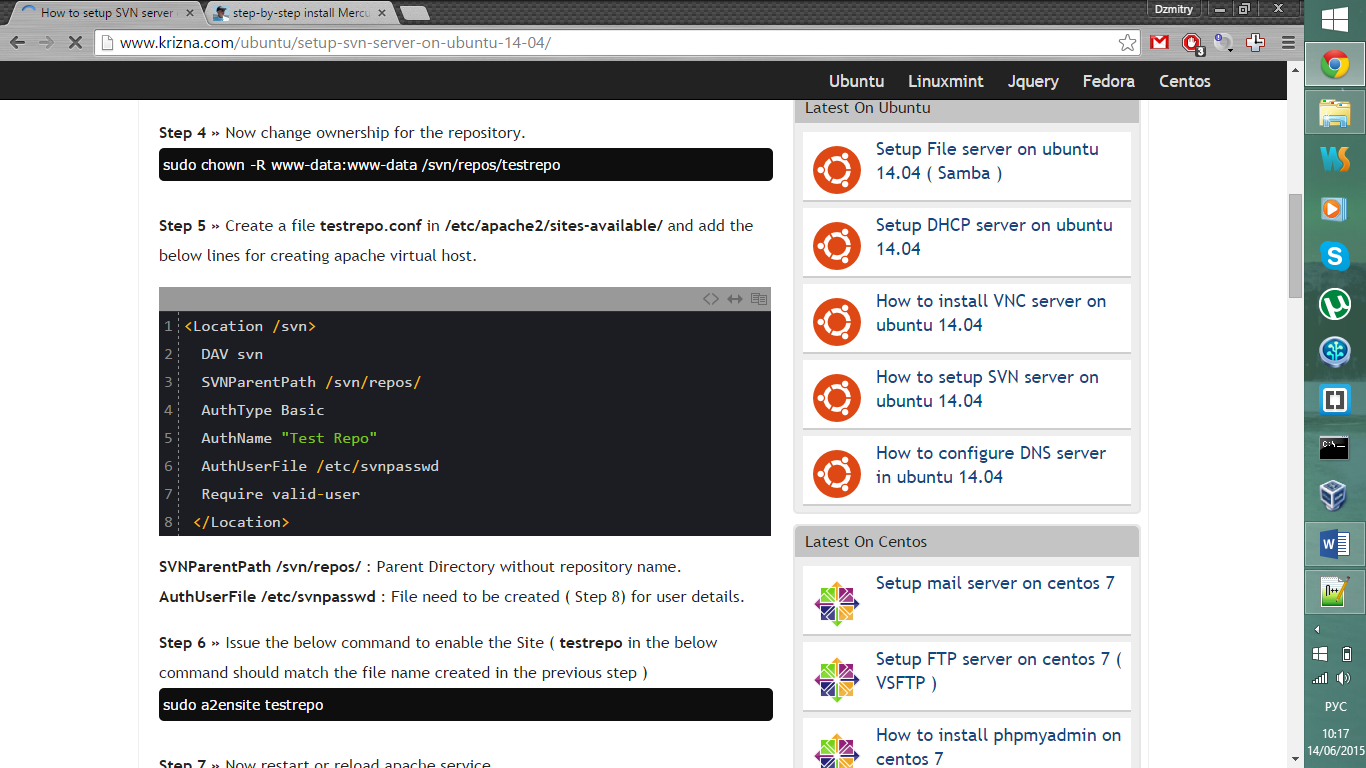


Рисунок 3.23 – Содержимое файла testrepo.conf

В этом файле строка SVNParentPath /svn/repos/ – это родительская папка без имени репозитория, а AuthUserFile /etc/svnpasswd – файл, который нужно создать для добавления данных пользователя.

Теперь можно добавить сайт, на котором можно будет просматривать текущие изменения в репозитории, выполнив sudo a2ensite testrepo. Далее необходимо будет перезапустить Apache-сервер, через команду sudo service apache2 reload.

Когда мы вызываем a2ensite, скрипт ищет в каталоге /etc/apache2/sites-available/ файл виртуального хоста mysite.ltd, если файл был найден, скрипт создаёт символическую ссылку на одноименный «файл» в каталоге /etc/apache2/sites-enabled/. После активации виртуального хоста остаётся просто послать команду, которая заставит вебсервер перечитать файлы конфигурации.

Добавление данных в файл с доступом делается следующим образом:

− sudo htpasswd -cm /etc/svnpasswd user1 – для добавления первого пользователя;

− sudo htpasswd -m /etc/svnpasswd user2 – для добавления второго и последующих.

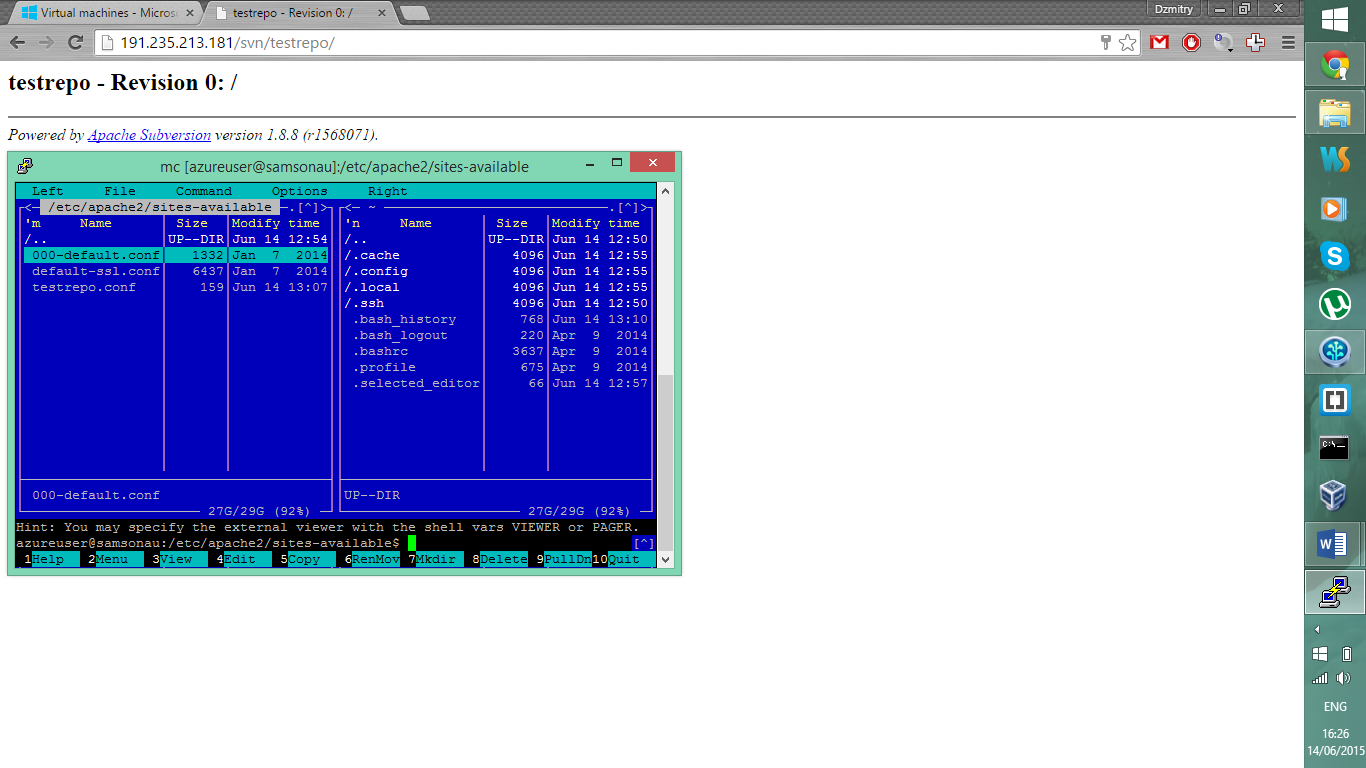


Рисунок 3.24 – Проверка правильности установки Apache Subversion

После этого можно запустить браузер по адресу http://тут\_адрес\_сервера/svn/testrepo и проверить правильность работы сервера. Если сервер настроен правильно, то результат будет схож с тем, что можно увидеть на рисунке 3.24. Если сайт не загрузился – стоит просто перезагрузить сначала apache-сервер, а затем и саму машину.

Если сайт загрузился, значит, можно начинать непосредственную работу с репозиторием.