# Работа №1

# Приложение 1

# «Системы контроля версий»

При разработке программного обеспечения необходимо оперировать исходными кодами программ. В процессе работы исходные коды постоянно модифицируются, что добавляет программному обеспечению новые возможности, однако это может привнести в исходный код дополнительные ошибки. Разумным решением является фиксация каких-то стабильных версий программного обеспечения, чтобы при необходимости была возможность откатиться обратно к стабильной версии.

Дополнительную сложность при разработке ПО представляет групповая разработка. Практически любое программное обеспечение разрабатывается группой программистов, и далеко не всегда удается разделить их работу таким образом, чтобы участки программного кода, с которыми они работают, никак не пересекались. Во всех групповых проектах существует необходимость одновременного редактирования одного и того же файла исходных кодов несколькими разными людьми.

Для решения этих задач были разработаны системы контроля версий.

**Система контроля версий** — это специальное программное обеспечение, обеспечивающее работу с постоянно изменяющейся информацией. В задачи системы контроля версий входит хранение истории изменений и обеспечение одновременной работы нескольких разработчиков.

В рамках систем контроля версий используется следующая терминология:

**Репозиторий** — хранилище документов вместе с историей их изменения.

**Версия, ревизия** — зафиксированное состояние документов, которому присвоена определенная дата и уникаьный номер. Как правило версия снабжается комментарием, описывающим сделанные относительно прошлой версии изменения.

**Основная версия (HEAD) —** последняя ревизия, зафиксированная в репозитории. Как правило, работа ведется именно с ней.

**Рабочая копия** — набор документов, с которым производится работа. Рабочая копия не содержит истории документов. Она может относиться как основной версии, так и к более ранней.

**Обновление (check-out)** — это получение от репозитория новой рабочей копии.

**Фиксация (commit)** — внесение набора изменений и формирование новой версии (ревизии) в репозитории.

**Слияние (merge)** — происходит при фиксации в том случае, если обнаруживается несколько разных правок от разных разработчиков в одном документе. Как правило, система контроля версий в состоянии безошибочно объединить несколько правок в разных частях документа. В противном случае возникает конфликт:

**Конфликт** — это ситуация, возникающая при внесении нескольких правок в один и тот же участок программного кода, при которой система контроля версий не может самостоятельно произвести слияние. В таком случае система предоставляет пользователям возможность осуществить слияние вручную. Пользователь может принять только один набор изменений, либо объединить их вручную, проанализировав добавленный код.

Также в процессе разработки зачастую возникает ситуация, когда внесение каких-то изменений не укладывается в один набор изменений, а добавление в основную версию сырых наработок недопустимо. В таком случае применяют ветвление.

**Ветвь (branch)** — это независимое направление разработки. Изменения, вносимые в одну ветвь, не влияют на другую. При создании ветви образуются две параллельные версии, имеющий общую историю до момента ветвления и отдельные — после.

Когда набор изменений в ветви приведет к формированию стабильной рабочей версии, можно осуществить слияние ветви с другой, стабильной ветвью.

Слияние в таком случае происходит так же, как и обычно, так как отдельная ветвь рассматривается всего лишь как большой набор изменений.

На сегодняшний день существует большое количество систем контроля версий. Они разделяются на две большие группы:

1. Централизованные системы. В таких системах один из узлов — сервер — хранит у себя репозиторий, а клиенты при работе с репозиторием получают лишь рабочие копии. При необходимости, пользователь может запросить историю версий и привести свою рабочую копию к состоянии какой-то версии, существовавшей в прошлом. По окончании работы пользователь фиксирует набор изменений, создавая таким образом новую версию. К преимуществам централизованной системы можно отнести механизм **блокировки**, позволяющий выдать одному пользователю эксклюзивный доступ к части исходных кодов, чтобы избежать путаницы при их правке.

   Среди известных на сегодняшний момент централизованных систем контроля версий можно выделить следующие:

   - **Concurrent Version System (CVS)**. Является относительно старой системой, которая тем не мене до сих пор используется на некоторых проектах.
   - **Subversion (SVN)**. Система контроля версий, выпущенная для замены устаревшей CVS. Обладает всем присущим ей функционалом, а также устраняет ряд недоработок.
   - **Team Foundation Server** — продукт от компании Microsoft, предназначенный для автоматизации совместной разработки программного обеспечения. Содержит в себе, в том числе, централизованную систему контроля версий.

2. Распределенные системы. В таких системах пользователь получает от сервера не рабочую копию, а копию репозитория, с которой работает локально. Далее пользователь работает только со своей копией, внося в нее необходимые изменения, фиксируя новые версии, а также, при необходимости, производя ветвления. При необходимости фиксации сделанных изменений пользователь отправляет их на сервер, где происходит слияние двух репозиториев в один. Среди распределенных систем контроля версий можно выделить следующие:

- **BitKeeper**. Проприетарная система контроля версий, разработанная в 1998 году для обслуживания исходных кодов системы GNU/Linux. В 2005 году владельцы данной системы запретили ее бесплатное использование, после чего сообщество Linux разработало свои распределенные системы контроля версий.
- **Git**. Система контроля версий, разработанная сообществом разработчиков ядра ОС Linux для своих нужд на замену системы BitKeeper. На сегодняшний день является самой популярной системой контроля версий и используется для разработки широкого круга программного обеспечения для различных платформ.
- **Mercurial**. Система контроля версий, разрабатываемая параллельно с Git. Данная система сходна с Git архитектурно, отличается несколько более понятным синтаксисом, имеет отличия во внутренней структуре, но меньше распространена.

В рамках распределенной системы контроля версий вводятся следующие понятия:

**Клонирование (clone)** — создание новой локальной копии удаленного репозитория.

**Получение репозитория (pull)** — обновление существующей локальной копии удаленного репозитория.

**Отправка репозитория (push)** — отправка своего локального репозитория на сервер для объединения с удаленным и фиксации сделанных изменений на удаленном сервере. Как правило, push жестко контролируется, требует аутентификации пользователя и происходит гладко только при отсутствии конфликтов. При наличии конфликтов пользователь должен выполнить pull, разрешить все конфликты и лишь затем выполнять push.

В рамках данного курса будет производиться работа с системой контроля версий Git. Домашняя страница этой системы — сайт **http://git-scm.com/**. На странице проекта можно бесплатно скачать как сам клиент git, написанный в виде консольного приложения, так и графические клиенты git для различных платформ, а также ознакомиться со справочным руководством для данной системы: http://git-

scm.com/book/ru/v2. Данная система является кросс-платформенной и доступна для всех настольных операционных систем, используемых на данный момент.

Для операционных систем MS Windows рекомендуется применение графического клиента git: **Tortoise Git**. Он представляет собой удобную обертку на консольный клиент, и должен быть установлен после официального. Tortoise Git доступен по адресу http://tortoisegit.org/.

## Пример работы с Git

Рассмотрим работу с Git на примере консольных команд. Все графические оболочки, так или иначе, вызывают консольные команды git, поэтому обладают схожей терминологией.

Для запуска консольного режима следует запустить команду **Git CMD** или **Git Bash** из меню «Пуск», либо воспользоваться контекстным меню в необходимой папке: нажать правой кнопкой мыши на пустом месте и выбрать пункт «**Git CMD Here**» или «**Git Bash Here**».

Для работы с git используется одноименная консольная команда «`git`», в качестве параметров которой указывается необходимое действие. Перед началом работы пользователю необходимо либо создать новый пустой репозиторий в текущей папке командой `git init`:

```
c:\_progs\git> git init test
Initialized empty Git repository in c:/_progs/git/test/.git/
```

либо склонировать существующий репозиторий с удаленного сервера в текущую папку командой `git clone`:

```
c:\_progs\git> git clone https://example.com/git/test.git
Cloning into 'test'...
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

В процессе клонирования git отображает ход процесса и выводит справочные сведения о репозитории. В случае, если удаленный репозиторий пуст, git выдаст соответствующее предупреждение:

```
c:\_progs\git> git clone https://example.com/git/test.git
Cloning into 'test'...
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
```

Все остальные команды git выполняются из папки репозитория. В приведенном примере необходимо перейти в склонированную папку, используя команду `cd`:

```
c:\_progs\git> cd test
c:\_progs\git\test>
```

В случае, если у вас уже есть клон локального репозитория, и вам нужно лишь получить свежие правки, достаточно выполнить команду `git pull`. Она обновит локальный репозиторий до того же состояния, в котором находится удаленный. Обычно эта команда выполняется первой после начала работы с уже имеющейся копией репозитория. В случае отсутствия каких-то изменений система выдаст сообщение:

```
c:\_progs\git\test>git pull
Already up-to-date.
```

При наличии удаленных изменений вывод команды будет похож на `git clone`:

```
c:\_progs\git\test> git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://example.com/git/test
   c6e6dc1..65fe65e  master     -> origin/master
Updating c6e6dc1..65fe65e
Fast-forward
```

```
 DESCRIPTION.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

## Состояние репозитория

Находясь в папке репозитория, можно в любой момент посмотреть его текущее состояние при помощи команды `git status`. В случае репозитория, который только что был склонирован, сообщение будет выглядеть следующим образом:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

При наличии локальных изменений, не зафиксированных нигде, сообщение будет выглядеть следующим образом:

```
c:\_progs\git\test> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

        modified:   DESCRIPTION.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

При наличии новой локальной версии, не добавленной в удаленный репозиторий, сообщение о статусе будет выглядеть следующим образом:

```
c:\_progs\git\test> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
```

```
nothing to commit, working directory clean
```

По умолчанию git находится в ветви **master**. Это название не имеет специального назначения, а просто сложилось исторически. Удаленный репозиторий, с которого производилось клонирование, обозначается ключевым словом **origin**. Для просмотра информации об удаленном репозитории существует команда `git remote -v`:

```
c:\_progs\git\test>  git remote -v
origin  https://example.com/git/test.git (fetch)
origin  https://example.com/git/test.git (push)
```

Для того, чтобы просмотреть историю версий репозитория, существует команда `git log`. Она отображает дату изменения в хронологическом порядке, а также приводит комментарий к каждой ревизии:

```
c:\_progs\git\test> git log
commit e2a36929ce18b7f30837df3f352b336ced0e0b4c
Author: karankevich <anton_7c3@mail.ru>
Date:   Wed Jun 29 14:35:11 2016 +0300

    Мелкая правка

commit 65fe65e17ef0b68d195f274c8fce430ee1f53a1a
Author: karankevich <anton_7c3@mail.ru>
Date:   Wed Jun 29 14:29:49 2016 +0300

    Изменение кодировки описания Windows1251 -> UTF8

commit c6e6dc1f8df64a2caa73ff90f9571e4987fece19
Author: Karankevich <anton_7c3@mail.ru>
Date:   Wed Jun 29 14:15:00 2016 +0300

    Начальный коммит в тестовый репозиторий
```

Вывод команды `git log` показывает номер ревизии после слова `commit`. В

классических системах контроля версий, таких как SVN, номер ревизии является обычным числом. Система git ориентирована на ветвления и распределенную разработку, поэтому присвоить всем версиям репозитория единую и сквозную нумерацию не представляется возможным. Вместо этого каждой ревизии в качестве номера присваивается длинная уникальная последовательность символов — хэш.

При работе с номерами ревизий, например, когда необходимо откатиться на какую то из прошлых версий, не обязательно указывать хэш целиком. Достаточно указать первые несколько символов, которые бы однозначно определяли номер. В приведенном примере номер первой ревизии можно сократить до `c6e6dc` и даже до `c6`.

## Добавление и отслеживание изменений

После того, как были сделаны какие-либо изменения, вывод команды `git status` даст об этом понять:

```
c:\_progs\git\test> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

        modified:   DESCRIPTION.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Также эта команда дает подсказку о том, что для фиксации новой ревизии необходимо указать, какие файлы будут в ней участвовать, вызвав команду `git add`:

```
c:\_progs\git\test> git add DESCRIPTION.txt
```

После этого вывод команды статуса будет выглядеть так:

```
c:\_progs\git\test> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   DESCRIPTION.txt
```

На данном этапе можно зафиксировать сделанные изменения командой `git commit`. Команда вызовет текстовый редактор, в котором нужно будет записать комментарий к данной фиксации.

```
c:\_progs\git\test> git commit
[master 3457362] Revision 3.5
 1 file changed, 4 insertions(+), 1 deletion(-)
```

Более короткий вариант вышеприведенных команд — вызов команды «`git commit -a`». Она фиксирует новую ревизию, автоматичеси добавляя к ней все файлы, в которых были сделаны изменения. Однако, эта команда не добавит новые файлы, о которых репозиторий еще не знает. Для их добавления все же придется вызвать `git add`.

После этого изменения зафиксированы в локальном репозитории, что может быть видно командой `git log`:

```
c:\_progs\git\test> git log
commit d1ed234eecb0d28e44b779f82b53453db9668923
Author: Karankevich <anton_7c3@mail.ru>
Date:   Wed Jun 29 14:35:53 2016 +0300

    Очередные мелкие правки

commit e2a36929ce18b7f30837df3f352b336ced0e0b4c
Author: karankevich <anton_7c3@mail.ru>
Date:   Wed Jun 29 14:35:11 2016 +0300
```

```
    Мелкая правка

commit 65fe65e17ef0b68d195f274c8fce430ee1f53a1a
Author: karankevich <anton_7c3@mail.ru>
Date:   Wed Jun 29 14:29:49 2016 +0300

    Изменение кодировки описания Windows1251 -> UTF8

commit c6e6dc1f8df64a2caa73ff90f9571e4987fece19
Author: Karankevich <anton_7c3@mail.ru>
Date:   Wed Jun 29 14:15:00 2016 +0300

    Начальный коммит в тестовый репозиторий
```

Однако, данные изменения сделаны **локально**, о чем говорит вывод команды `git log`:

```
c:\_progs\git\test> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

Для того, чтобы отправить свои изменения на удаленный сервер, необходимо вызвать команду `git push`. Команда требует написания, в какую ветвь удаленного репозитория следует внести изменения, поэтому нужно вызвать команду `git push origin master`:

```
c:\_progs\git\test> git push origin master
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 297 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
```

```
To https://niiret.ru/git/test.git
   5c371ec..3457362  master -> master
```

Довольно распространена ситуация, когда локальный репозиторий отстает от исходного (origin) на некоторое количество версий. В таком случае команда git push выдаст соответствующее предупреждение:

```
c:\_progs\git\test> git push origin master
To https://niiret.ru/git/test.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://niiret.ru/git/test.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Выходом в такой ситуации является применение команды `git pull`, которая прочитает все изменения в удаленном репозитории, произведет слияние и приведет репозиторий к тому же состоянию, что и удаленный, плюс сделанные локальные правки и добавленные версии. Такой репозиторий можно отправлять через `git push`.

## Сводная таблица команд git

Можно кратко перечислить основные команды git в порядке их применения в виде таблицы:

| **Команда** | **Назначение** |
|---|---|
| `git init` | Создание нового пустого репозитория. |
| `git clone` | Создание локальной копии удаленного репозитория. |
| `git pull` | Получение свежей версии уже склонированного репозитория. |
| `git status` | Текущее состояние локального репозитория, его отличие от удаленного, наличие незафиксированных правок. |
| `git log` | Отображение хронологии репозитория. |

| **Команда** | **Назначение** |
|---|---|
| `git add` | Добавление файла для фиксации следующей версии — нового или уже существующего с новыми изменениями. |
| `git commit` | Фиксация новой версии (ревизии). |
| `git commit -a` | Быстрая фиксация новой версии с автоматическим добавлением всех файлов, в которых были сделаны изменения. Не требует предварительного вызова git add, однако не добавляет новые файлы. |
| `git push` | Отправка изменений на удаленный репозиторий. Рекомендуется применять команду в формате «git push origin master». |

Служебной командой git является команда `git config`. Она позволяет оперировать различными настройками, как глобальными, так и настройками конкретной рабочей копии.

Диаграмма работы с git представлена на рисунке 1.

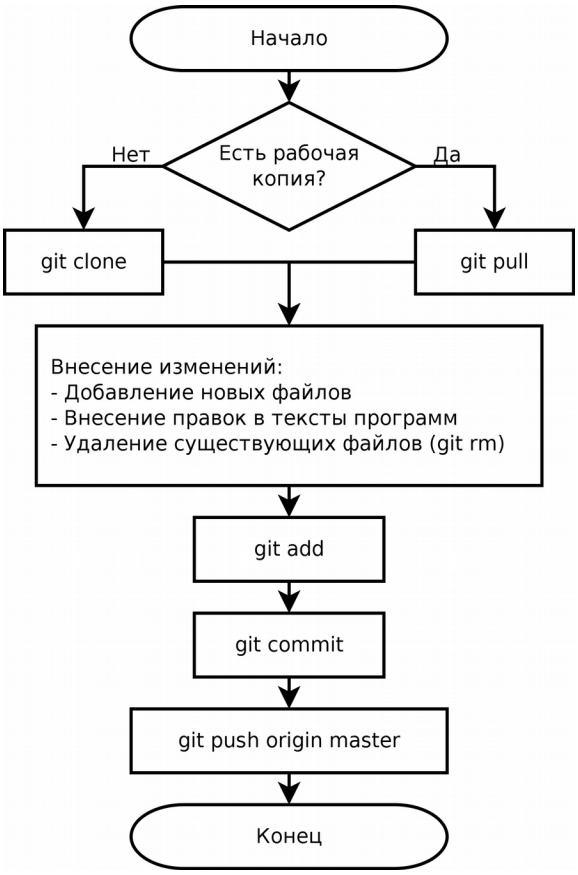


Рисунок 1 - Диаграмма работы с Git

Среди рекомендуемых клиентов для платформы MS Windows можно выделить следующие:

**SourceTree** (сайт **http://www.sourcetreeapp.com/**). Является бесплатным клиентом Git и Mercurial для платформ MS Windows 7 и новее, а также Mac OS X 10.7 или новее.

**SmartGit** (сайт **http://www.syntevo.com/smartgit/**). Является коммерческим клиентом git, однако доступен бесплатно для некоммерческого применения в течение ограниченного времени.

**TortoiseGit** (сайт **https://tortoisegit.org/**). Представляет собой доволнение к проводнику MS Windows, позволяющее вызывать команды git через контекстное меню (появляющееся при нажатии правой кнопки мыши). Требует предварительной установки консольного клиента git, который можно скачать с сайта **http://git-scm.com/.**

## Пример работы с Tortoise Git

Использование консольного клиента git оправдано в достаточно простых случаях, либо при отсутствии графического интерфейса. Для разработки в среде MS Windows с использованием Git одним из лучших решений является использование графической обертки под названием Tortoise Git. Она представляет собой команды git, встроенные в контекстное меню проводника Windows. Аналогичная обертка существует для системы SVN. На рисунке 2 показано контекстное меню проводника в операционной системе, в которой установлены Tortoise Git и Tortoise SVN.

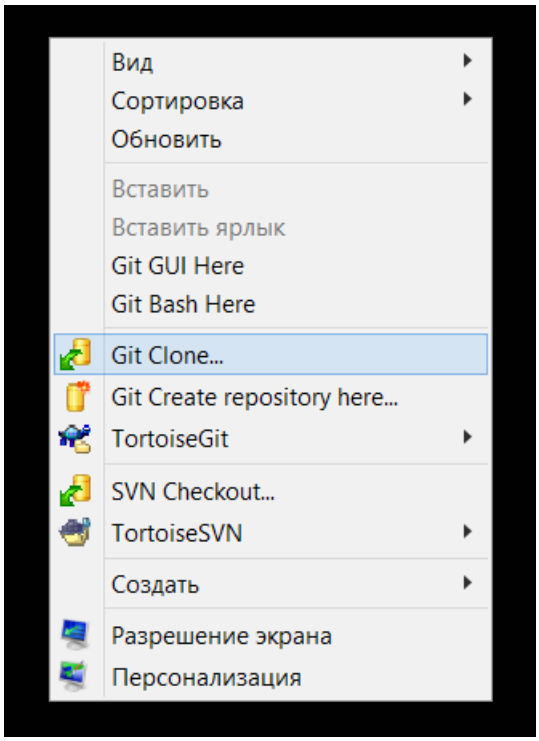


Рисунок 2 - Контекстное меню проводника с Tortoise Git

Работа с git при помощи Tortoise Git как правило начинается с вызова пункта меню «Git Clone...». В появившемся окне (рисунок 3) необходимо указать URL удаленного репозитория. Папка, в которую будет склонирован репозиторий (в приведенном примере — `c:\_progs\test\git\test`) не должна существовать, так как она будет создана в процессе клонирования. При необходимости можно поменять имя и путь создаваемой папки.

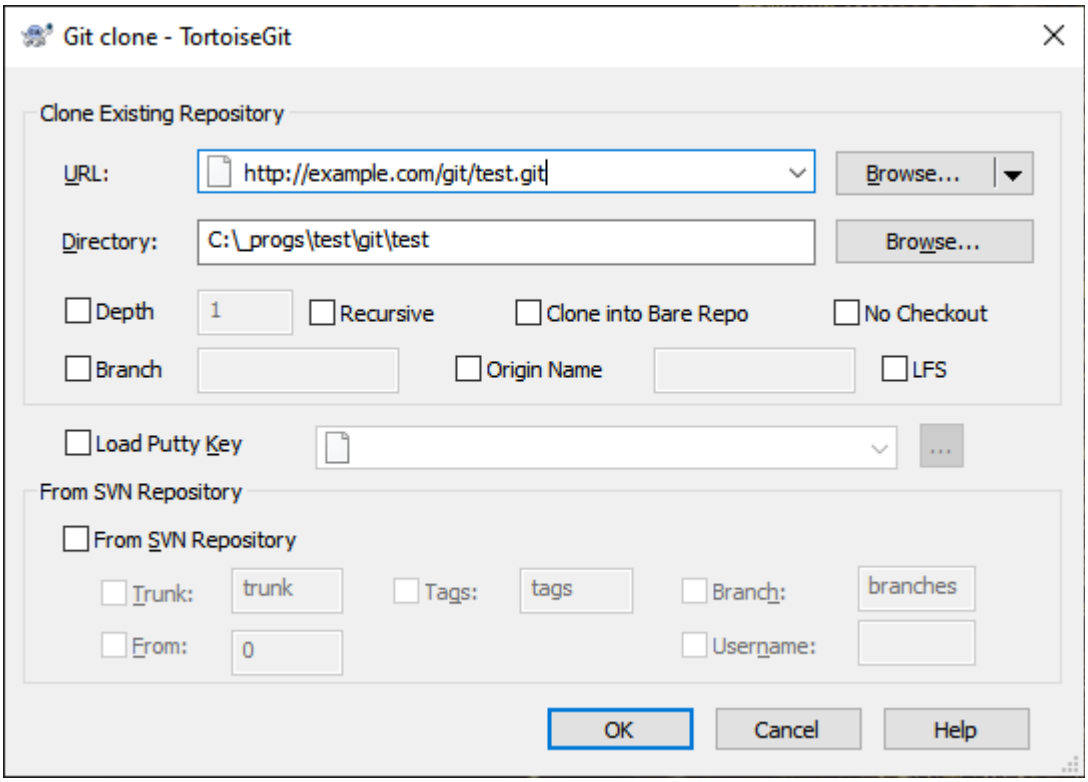


Рисунок 3 - Окно команды Git clone

Также система Tortoise Git автоматически добавляет значки в левом нижнем углу к тем файлам, которые находятся под контролем версий, то есть обслуживаются локальным репозиторием. На рисунке 4 показан набор файлов в репозитории, расположенных в папке test после клонирования.

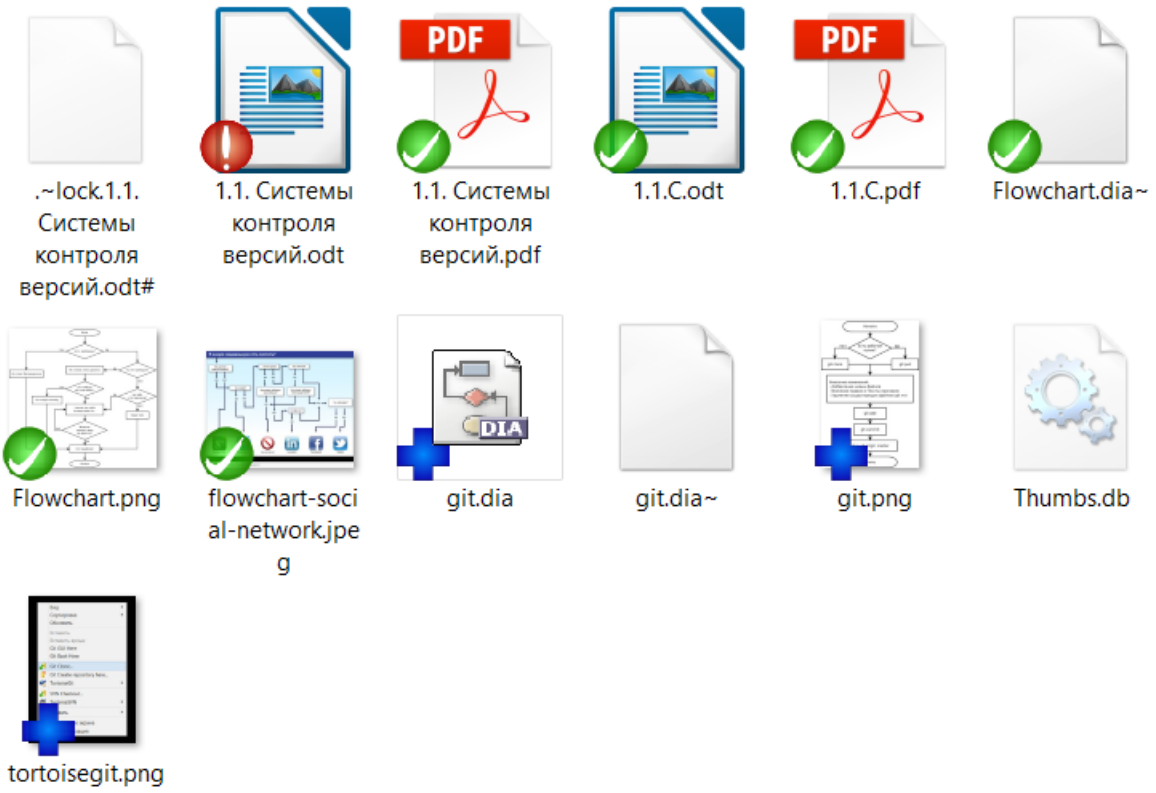


Рисунок 4 - Файлы под контролем версий в Tortoise Git

В папке можно выделить несколько типов файлов:

- Файлы, не находящиеся под контролем версий. Такие файлы не отмечаются значком в левом нижнем углу, либо отмечаются знаком вопроса в круге (на рисунке не показано).
- Файлы, соответствующие текущей версии. Такие файлы отмечаются зеленой галочкой. Они не содержат каких либо правок относительно текущей версии репозитория.
- Файлы, содержащие правки. Такие файлы отмечаются красным восклицательным знаком. Они содержат набор изменений относительно текущей версии, которые могут быть зафиксированы при помощи команды `commit`, либо отменены при помощи команды `revert`.
- Файлы, отмеченные для добавления. Такие файлы отмечены знаком «Плюс» и будут добавлены к следующей версии, зафиксированной в репозитории.

Следует отметить, что статусы файлов, отображаемые в виде иконок,

соответствуют только **локальному** репозиторию, поэтому состояние значка в углу иконки файла может не соответствовать фактическому состоянию данного файла на сервере.

При необходимости внести файлы в систему контроля версий следует выбрать его, вызвать контекстное меню и в подменю «**Tortoise Git**» выбрать пункт **Add**. Рядом с файлом появится значок плюсика, который означает, что файл будет добавлен в ближайшую версию.

Для фиксации версии следует вызвать из контекстного меню команду «**Git Commit -> Master**». Данная команда вызовет меню создания новой версии, где будут указаны файлы, участвующие в создании версии (как новые, так и старые с исправлениями), а также поле ввода комментария. Добавление версии без комментария запрещено. На рисунке 5 показана процедура создания новой версии.

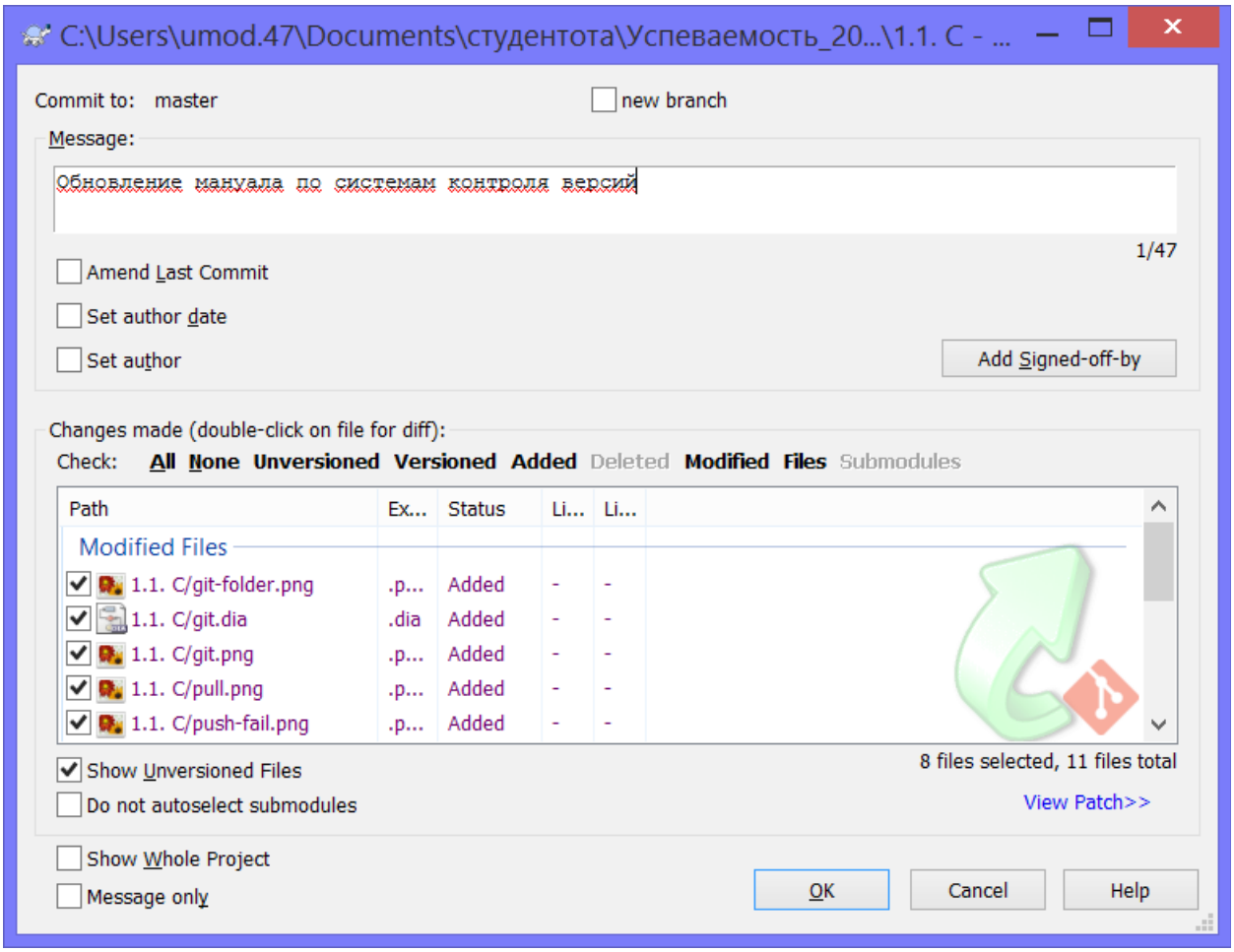


Рисунок 5 - Создание новой версии в Tortoise Git

При создании новой версии можно посмотреть список файлов, подвергшихся изменениям

Для синхронизации с удаленной рабочей копией существует пункт меню «**Git Sync**», который вызывает окно синхронизации изменений. Среди кнопок в этом окне следует выделить следующие:

- **Pull**. Используется для получения изменений из удаленного репозитория и их слияния с локальной версией. Текстовый лог, отображаемый в окне, выводит то же самое, что вывела бы одноименная консольная команда. При успешном получении изменений в логе работы команды будет написано **Success** (рисунок 6):

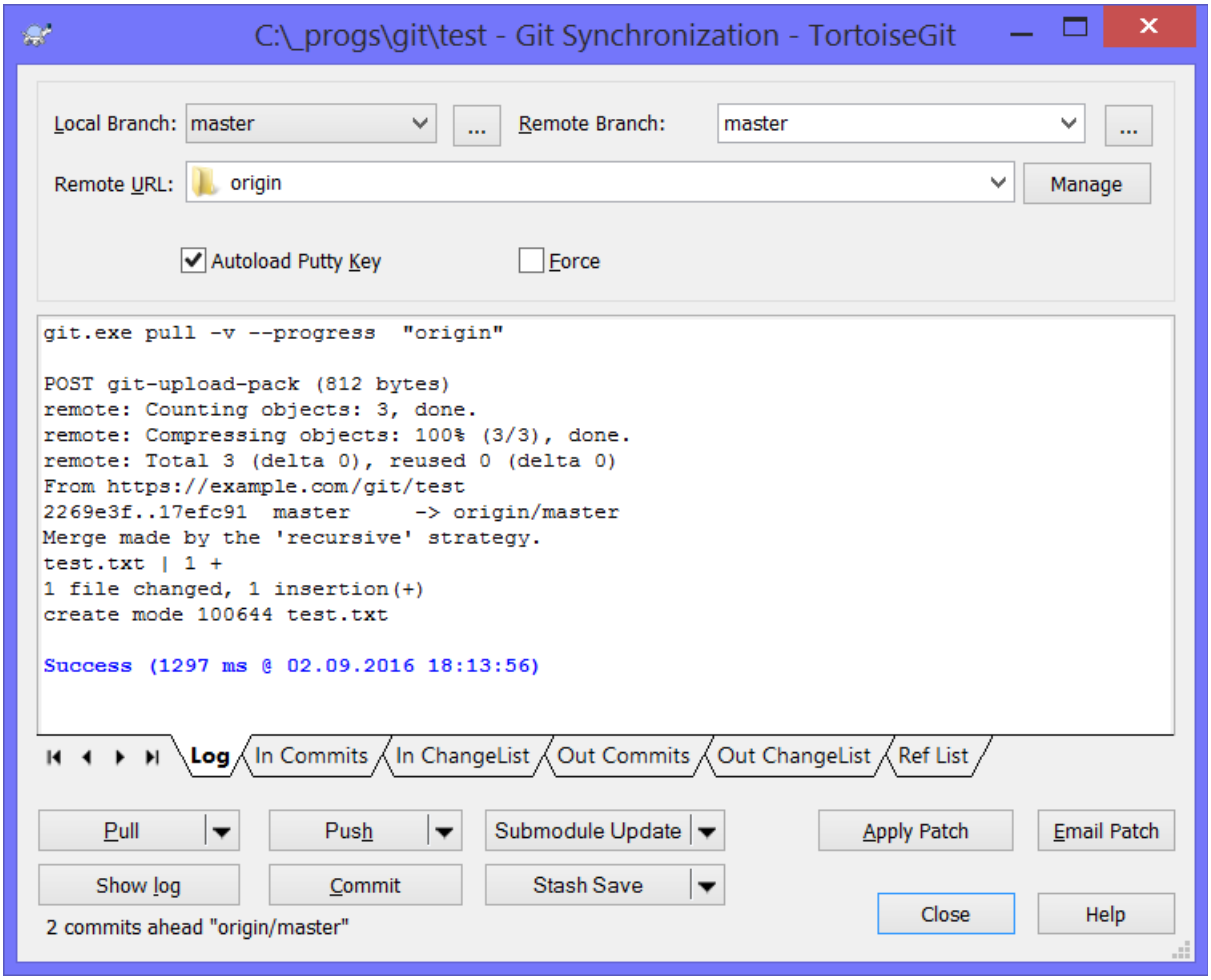


Рисунок 6 - Команда Git Pull в Tortoise Git

• **Push**. Используется для отправки изменений в удаленный репозиторий.

При использовании команды Push возможна ситуация, когда удаленный репозиторий содержит изменения, не учтенные в локальном репозитории. В таком случае вывод Push будет содержать сообщение об ошибке (рисунок 7):

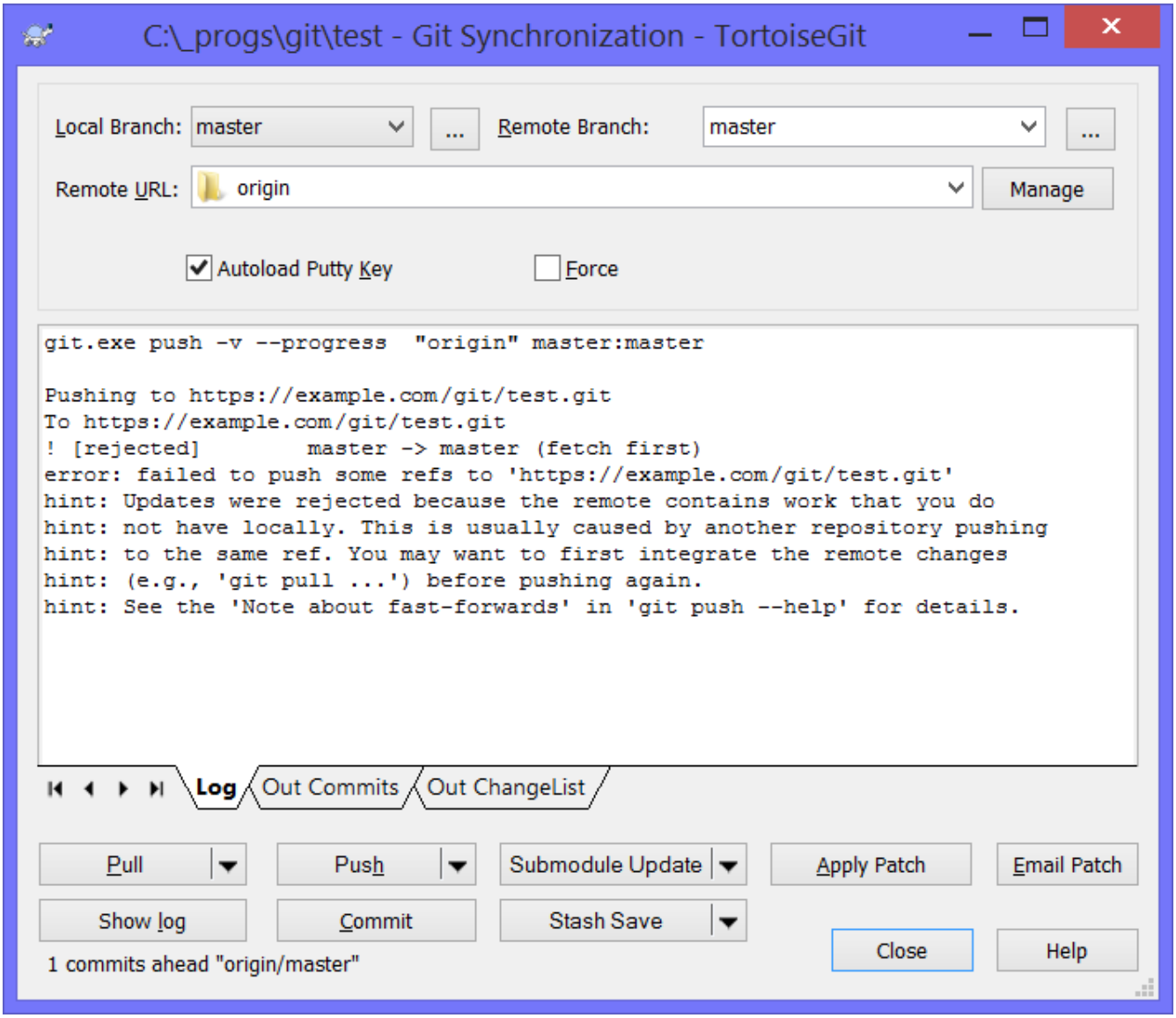


Рисунок 7 - Ошибка при работе Git Push при наличии изменений в удаленном репозитории

В таком случае необходимо сначала выполнить команду Git Pull для получения удаленных изменений и затем выполнить Git Push. При правильной работе Git Push также напишет **Success** в конце лога работы (рисунок 8):

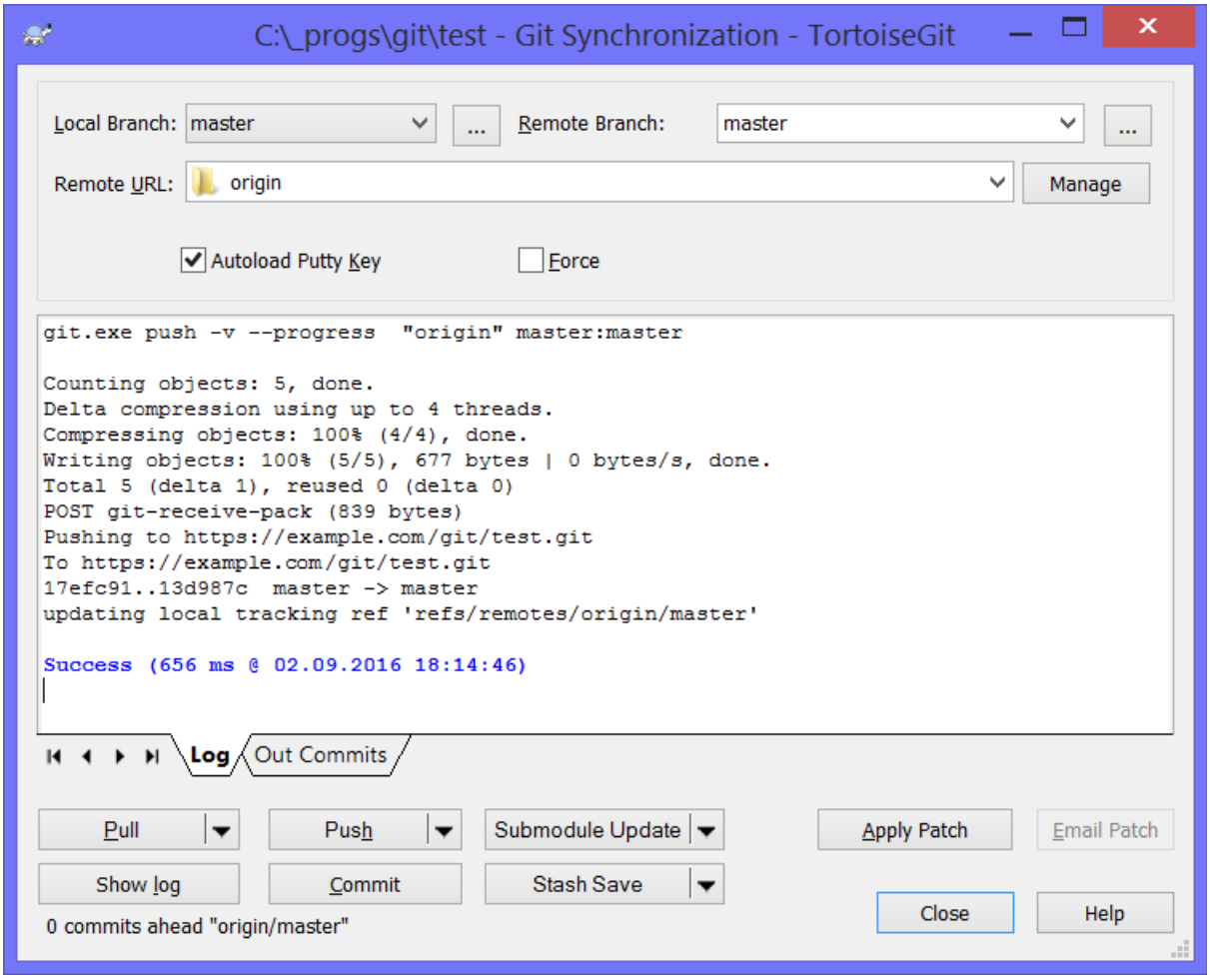


Рисунок 8 - Правильная работа команды Git Push

Внизу окна синхронизации выводится информация о том, на сколько версий отличается локальный репозиторий и исходный. Можно считать, что все изменения зафиксированы успешно, если после проведения синхронизации с помощью Pull или Push ниже кнопки «Show log» написано «`0 commits ahead “origin/master”`».