**ГРУППА К8-224**



**СЕРВИС ПО ПОИСКУ ВЕЧЕРИНОК. ИНСТРУКЦИЯ ПО ИСПОЛЬЗОВАНИЮ GITHUB**

|  |
| --- |
|  |
| «PartySurfing» |

**Москва 2014 - 2015**

**Содержание**

[1. Установка Git под Windows 3](#_Toc415660195)

[2. Установка Git под Linux 4](#_Toc415660196)

[2.1. Установка из исходников 4](#_Toc415660197)

[2.2. Установка в Linux 5](#_Toc415660198)

[3. Начальная подготовка к работе 6](#_Toc415660199)

[4. Начало работы с Git 7](#_Toc415660200)

[5. Проверка состояния репозитория 10](#_Toc415660201)

[6. Отмена локальных изменений 14](#_Toc415660202)

[7. Отмена проиндексированных изменений 16](#_Toc415660203)

[8. Отмена коммита 17](#_Toc415660204)

[9. Удаление файлов 18](#_Toc415660205)

[10. Отправка изменений в удаленный репозиторий 19](#_Toc415660206)

[11. Извлечение изменений из удаленного репозитория 20](#_Toc415660207)

[12. Небольшая шпаргалка 21](#_Toc415660208)

[13. Приложение 25](#_Toc415660209)

История изменений

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Версия | Изменения | Дата | Автор | Подпись |
| 1.0 | Инструкция | 17.10.2014 | Сиволоб Иван |  |
| 2.0 | Редактирование | 20.12.2014 | Македонская Евгения |  |

# Установка Git под Windows

Cкачайте с http://msysgit.github.io/ файл .exe и просто запустите его. Будет установлена как консольная (GitBash) так и графическая (GitGUI) версии.

# Установка Git под Linux

## [Установка из исходников](http://git-scm.com/book/ru/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git#Установка-из-исходников)

Если есть возможность, то, как правило, лучше установить Git из исходных кодов, поскольку так вы получите самую свежую версию. Каждая новая версия Git'а обычно включает полезные улучшения пользовательского интерфейса, поэтому получение последней версии — часто лучший путь, если, конечно, вас не затрудняет установка программ из исходников. К тому же, многие дистрибутивы Linux содержат очень старые пакеты. Для установки Git'а вам понадобятся библиотеки, от которых он зависит: curl, zlib, openssl, expat и libiconv. Например, если в вашей системе менеджер пакетов — yum (Fedora), или apt-get (Debian, Ubuntu), можно воспользоваться следующими командами, чтобы разрешить все зависимости:

$ yum install curl-devel expat-devel gettext-devel \

openssl-devel zlib-devel

или

$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext \

libz-dev libssl-dev

Установив все необходимые библиотеки, можно идти дальше и скачать последнюю версию с сайта Git'а:

http://git-scm.com/download

Теперь скомпилируйте и установите:

$ tar -zxf git-1.7.2.2.tar.gz

$ cd git-1.7.2.2

$ make prefix=/usr/local all

$ sudo make prefix=/usr/local install

После этого вы можете скачать Git с помощью самого Git'а, чтобы получить обновления:

$ git clone git://git.kernel.org/pub/scm/git/git.git

## [Установка в Linux](http://git-scm.com/book/ru/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git#Установка-в-Linux)

Если вы хотите установить Git под Linux как бинарный пакет, это можно сделать, используя обычный менеджер пакетов вашего дистрибутива. Если у вас Fedora, можно воспользоваться yum'ом:

$ yum install git-core

Если же у вас дистрибутив, основанный на Debian, например, Ubuntu, попробуйте apt-get:

$ apt-get install git

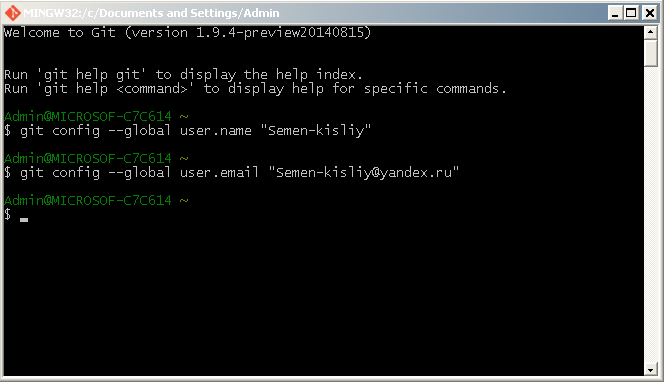
После завершения установки запустите консольную версию Git и можно приступать к работе.

(Если у вас иной менеджер пакетов, то скорее всего вы и сами знаете как установить Git.)

# Начальная подготовка к работе

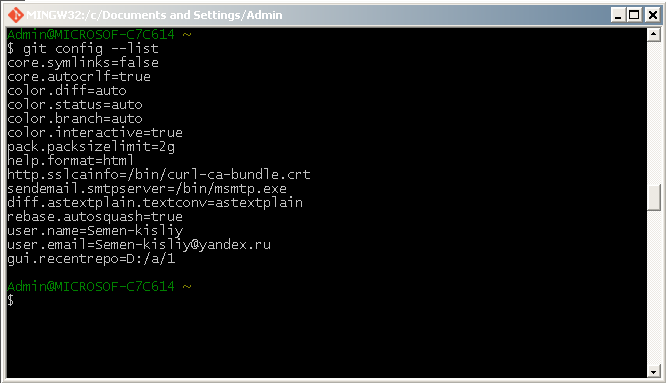
Если вы ещё ни разу не пользовались Git, то сначала нужно задать имя пользователя и пароль:

$ git config --global user.name "Ваше имя"

$ git config --global user.email "Ваш\_email@example.ru"

Если вы хотите проверить настройки, введите:

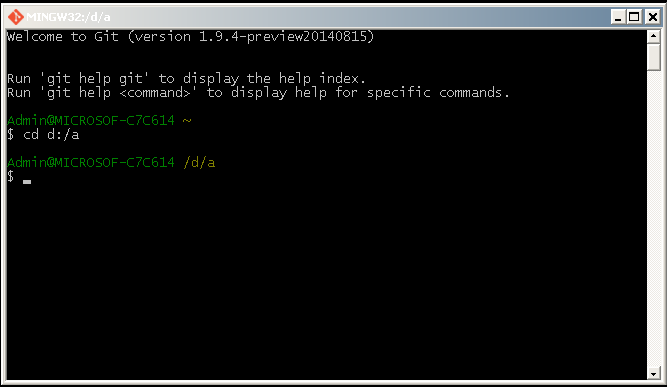
$ git config –list



# Начало работы с Git

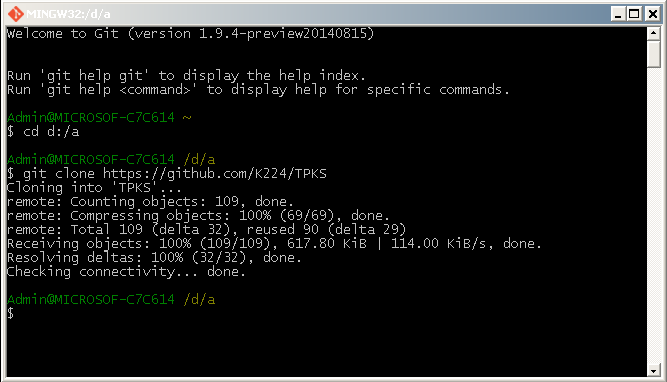
Чтобы включиться в работу над проектом склонируйте имеющийся удаленный репозиторий. Для этого в консоли зайдите в каталог, в котором хотите разместить файлы проекта, например:

$ cd d:/a/



Затем введите команду:

$ git clone <https://github.com/K224/TPKS>



Теперь переходим с помощью $ cd TPKS в раздел TPKS. В нём находятся локальные файлы скопированного репозитория. При изменении или добавлении новых файлов необходимо их сначала проиндексировать с помощью команды:

$ git add <file name>

А затем зафиксировать изменения:

$ git commit –m “Здесь описываем внесённые изменения”

(внутри кавычек лаконично комментируется, что именно измененилось)

Например, вручную или, используя touch hello.php, создаём в каталоге файл hello.php и вводим команды:

$ git add hello.php

$ git commit –m “hello.php file has been added”

Для упрощения добавления файлов можно использовать Glob шаблоны. Они представляют собой упрощенные регулярные выражения используемые командными интерпретаторами. Символ \* соответствует 0 или более символам; последовательность [abc] — любому символу из указанных в скобках (в данном примере a, b или c); знак вопроса (?) соответствует одному символу; [0-9] соответствует любому символу из интервала (в данном случае от 0 до 9).

Например,

$ git add \*.php

Добавит все *php* файлы в индексацию.

Также команду $ git commit можно передать с параметром –a, тогда автоматически проиндексируется каждый уже отслеживаемый на момент коммита файл, позволяя вам обойтись без $ git add:

$ git commit -a -m 'added new files'

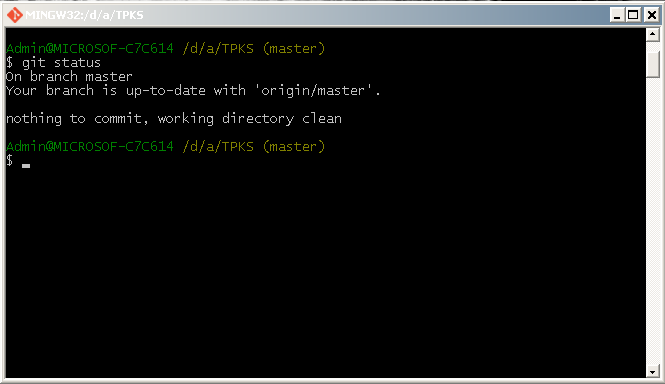
(перед коммитом $ git add писать уже не нужно )

Можно использовать параметр –v , тогда в комментарий будет также помещена дельта изменений между коммитами, и можно будет увидеть всё, что сделано.

# Проверка состояния репозитория

Для проверки состояния файлов используется команда:

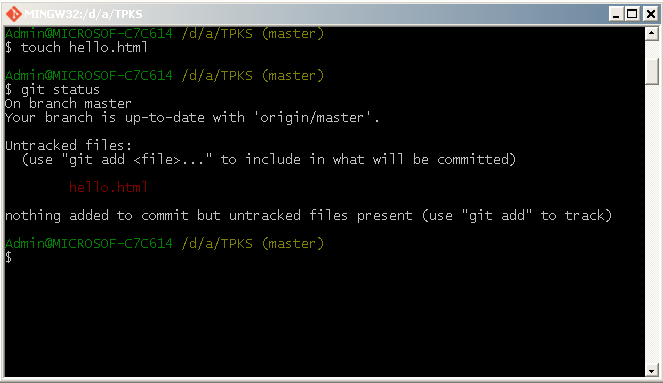
$ git status



Если проверить состояние сразу после коммита команда сообщит, что коммитить сейчас нечего. Это означает, что в репозитории хранится текущее состояние рабочего каталога, и нет никаких изменений, ожидающих записи.

При добавлении или изменении файлов $ git status будет выводить соответствующие сообщения с подсказками в скобках, о том, что делать в конкретном случае:

Проверка истории репозитория

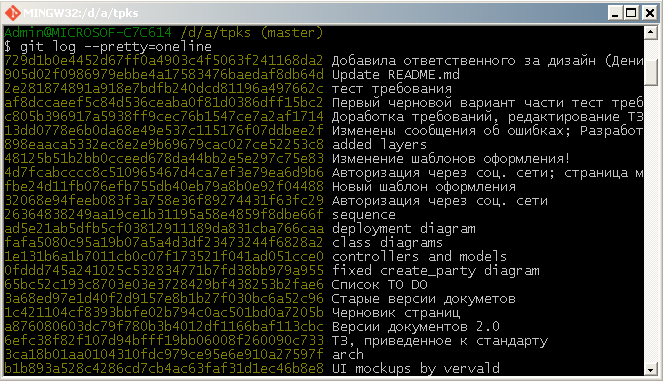


Для получения списка изменений можно использовать:

$ git log

При этом выводится информация обо всех совершенных коммитах (последний коммит будет в списке первым.) Но часто коммитов бывает очень много и пользоваться $ git log становится не удобно. Тогда можно применить форматирование:

$ git log --pretty=oneline

(вывод коммитов в одну строку)

Или например:

$ git log --pretty=oneline --max-count=2

$ git log --pretty=oneline --since='5 minutes ago'

$ git log --pretty=oneline --until='5 minutes ago'

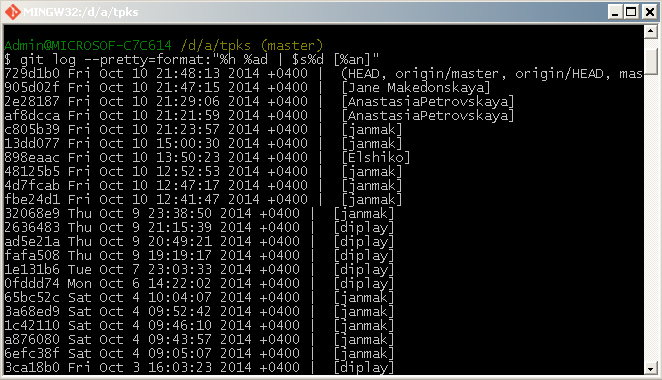
$ git log --pretty=oneline --author=<your name>

(здесь интуитивно понятно, что делают эти параметры)

Или ещё можно использовать:

$ git log --pretty=format:"%h %ad | %s%d [%an]"

(непосредственное применение форматирования)



--pretty="..." — определяет формат вывода.

Некоторые параметры формата:

* %h — укороченный хэш коммита
* %H —хэш коммита
* %d — дополнения коммита («головы» веток или теги)
* %ad — дата коммита
* %s — комментарий
* %an — имя автора
* %as — электронная почта автора

--graph — отображает дерево коммитов в виде ASCII-графика

--date=short — сохраняет формат даты коротким и симпатичным

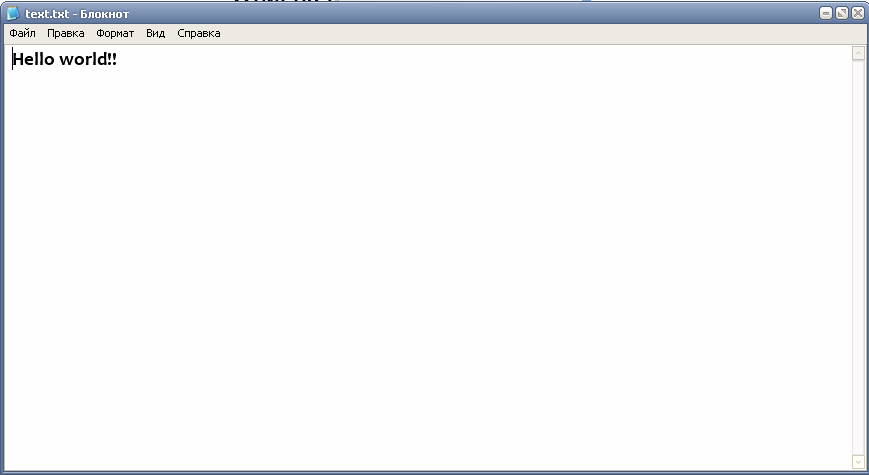
Для визуализации истории коммитов, можно попробовать программу gitk:

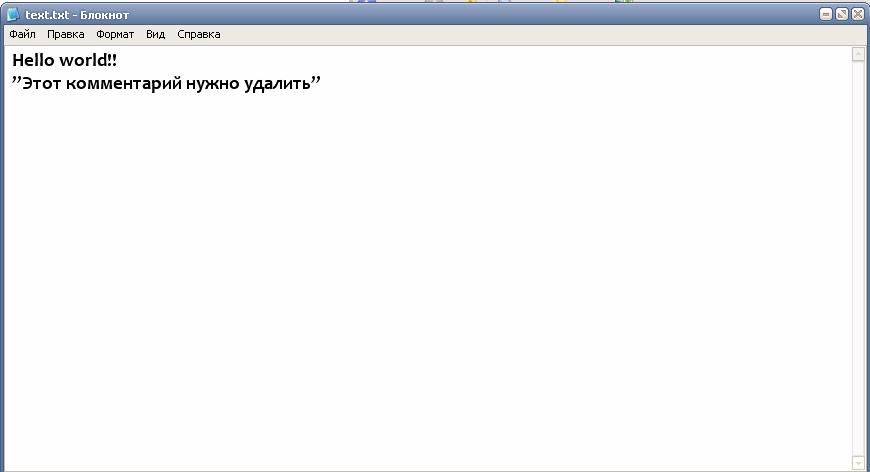
$ gitk

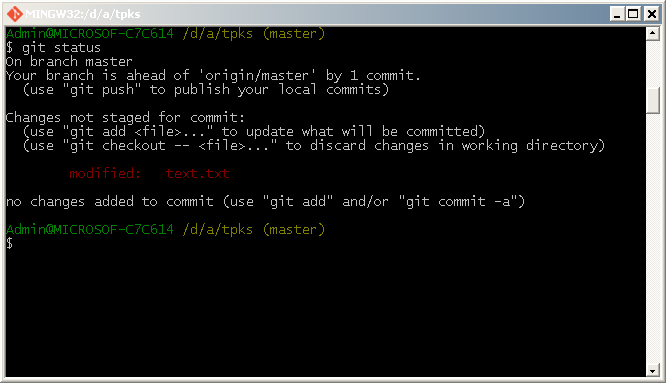
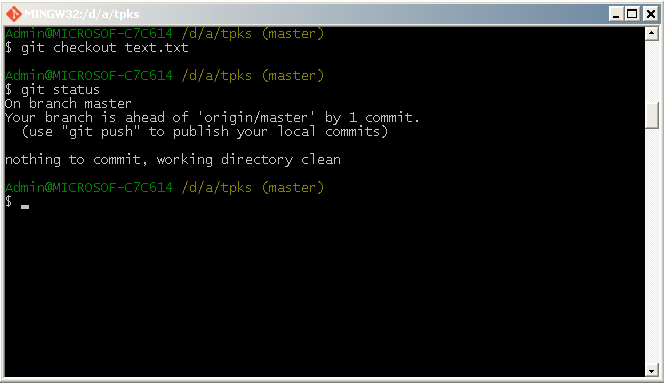
# Отмена локальных изменений

Если вы внесли нежелательные изменения в файл проекта (добавили ненужный комментарий, плохой код и т.д.), но не проиндексировали его и желаете убрать изменения, выполните:

$ git checkout <filename>

Например, был фай text.txt:



А вы добавили в него строку ”Этот комментарий нужно удалить” и хотите убрать её, то:

# Отмена проиндексированных изменений

Если вы внесли изменение в файл и проиндексировали его (уже выполнили, например, $ git add hello.html):

$ git reset HEAD hello.html

Команда reset сбрасывает буферную зону к HEAD (указателю на текущий коммит). Это очищает буферную зону от изменений, которые мы только что проиндексировали.

Команда reset (по умолчанию) не изменяет рабочий каталог. Поэтому рабочий каталог все еще содержит нежелательный комментарий. Мы можем использовать команду checkout из предыдущего урока, чтобы удалить нежелательные изменения в рабочем каталоге.

# Отмена коммита

Последний ненужный коммит отменяется командой:

$ git revert HEAD --no-edit

Если нужно отменить не последний коммит, вместо HEAD нужно указать известное хэш-значение коммита.

# Удаление файлов

Для того чтобы удалить файл из Git'а, вам необходимо удалить его из отслеживаемых файлов (точнее, удалить его из вашего индекса) а затем выполнить коммит. Это позволяет сделать команда git rm, которая также удаляет файл из вашего рабочего каталога, так что вы в следующий раз не увидите его как не проиндексированный.

Если вы изменили файл и уже проиндексировали его, вы должны использовать принудительное удаление с помощью параметра -f.

$ git rm -f

Чтобы удалить файл из индекса, но оставить его при этом в рабочем каталоге (в папке на диске):

$ git rm --cached readme.txt

# Отправка изменений в удаленный репозиторий

После того, как все необходимые изменения будут сделаны в вашем рабочем каталоге локально и будет произведён коммит, данные можно отправлять в удаленный репозиторий (на сервер github.com):

$ git push origin master

(где origin – имя удаленного репозитория по умолчанию, master – имя отправляемой ветки)

# Извлечение изменений из удаленного репозитория

В то время как локально вы работаете над своей частью проекта, ваши коллеги делают то же самое в своих рабочих каталогах, также локально, и время от времени они тоже будут отправлять свои изменения в удаленный репозиторий. Для того чтобы ваш рабочий каталог всегда оставался ”свежим”, необходимо извлекать новые данные:

$ git pull origin master

# [Небольшая шпаргалка](http://git-scm.com/book/ru/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git#Установка-из-исходников)

Создать новый репозиторий:

$ git init project-name

Клонировать репозиторий с удаленной машины:

$ git clone git@bitbucket.org:afiskon/hs-textgen.git

Добавить файл в репозиторий:

$ git add text.txt

Удалить файл:

$ git rm text.txt

Текущее состояние репозитория (изменения, неразрешенные конфликты и тп):

$ git status

Сделать коммит:

$ git commit -a -m "Commit description"

Сделать коммит, введя его описание с помощью $EDITOR:

$ git commit -a

Замерджить все ветки локального репозитория на удаленный репозиторий:

$ git push origin

Аналогично предыдущему, но делается пуш только ветки master:

$ git push origin master

Запушить текущую ветку, не вводя целиком ее название:

$ git push origin HEAD

Замерджить все ветки с удаленного репозитория:

$ git pull origin

Аналогично предыдущему, но накатывается только ветка master:

$ git pull origin master

Накатить текущую ветку, не вводя ее длинное имя:

$ git pull origin HEAD

Скачать все ветки с origin, но не сливать их в локальный репозиторий:

$ git fetch origin

Аналогично предыдущему, но только для одной заданной ветки:

$ git fetch origin master

Начать работать с веткой some\_branch (уже существующей):

$ git checkout -b some\_branch origin/some\_branch

Создать новую ветку (ответвится от текущего):

$ git branch some\_branch

Переключиться на другую ветку (из тех, с которыми уже работаем):

$ git checkout some\_branch

Получаем список веток, с которыми работаем:

$ git branch # звездочкой отмечена текущая ветвь

Просмотреть все существующие ветви:

$ git branch -a # | grep something

Слить some\_branch в текущую ветку:

$ git merge some\_branch

Удалить ветку (после мерджа):

$ git branch -d some\_branch

Просто удалить ветку (тупиковая ветвь):

$ git branch -D some\_branch

Последние изменения:

$ git log

История конкретного файла:

$ git log file.txt

Аналогично предыдущему, но с просмотром сделанных изменений:

$ git log -p file.txt

История с именами файлов и псевдографическим изображением бранчей:

$ git log --stat --graph

Изменения, сделанные в заданном коммите:

$ git show d8578edf8458ce06fbc5bb76a58c5ca4a58c5ca4

Посмотреть, кем в последний раз правилась каждая строка файла:

$ git blame file.txt

Удалить бранч из репозитория на сервере:

$ git push origin :branch-name

Откатиться к конкретному коммиту (хэш смотрим в «git log»):

$ git reset --hard d8578edf8458ce06fbc5bb76a58c5ca4a58c5ca4

Аналогично предыдущему, но файлы на диске остаются без изменений:

$ git reset --soft d8578edf8458ce06fbc5bb76a58c5ca4a58c5ca4

Попытаться обратить заданный commit (но чаще используется branch/reset + merge):

$ git revert d8578edf8458ce06fbc5bb76a58c5ca4a58c5ca4

Просмотр изменений (суммарных, а не всех по очереди, как в «git log»):

$ git diff # подробности см в "git diff --help"

Используем vimdiff в качестве программы для разрешения конфликтов (mergetool) по умолчанию:

$ git config --global merge.tool vimdiff

Отключаем диалог «какой mergetool вы хотели бы использовать»:

$ git config --global mergetool.prompt false

Разрешение конфликтов (когда оные возникают в результате мерджа):

$ git mergetool

Создание тэга:

$ git tag some\_tag # за тэгом можно указать хэш коммита

Удаление untracked files:

$ git clean -f

«Упаковка» репозитория для увеличения скорости работы с ним:

$ git gc

Следует отметить, что Git позволяет использовать короткую запись хэшей. Вместо «d8578edf8458ce06fbc5bb76a58c5ca4a58c5ca4» можно писать «d8578edf» или даже «d857».

# [Приложение](http://git-scm.com/book/ru/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%A3%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-Git#Установка-из-исходников)

Веб-сервис системы Git:

[*https://github.com/*](https://github.com/)

Для более полной и подробной информации о системе Git – перевод книги Pro Git:

[*http://git-scm.com/book/ru/Введение*](http://git-scm.com/book/ru/Введение)

Хороший вводный курс:

[*http://githowto.com/ru*](http://githowto.com/ru)

Очень короткое введение в самые основные команды:

[*http://rogerdudler.github.io/git-guide/index.ru.html*](http://rogerdudler.github.io/git-guide/index.ru.html)