

Записки программиста

Блог о программировании, а также электронике, радио, и всяком таком

Моя шпаргалка по работе с Git

28 декабря 2011

Некоторое время назад я открыл для себя Git. И знаете, я проникаю. То есть, по-настоящему проникаю. Теперь я использую Git не только на работе (где я с ним, собственно, познакомился), но и для своих проектов, которые я стал хранить на BitBucket. Последний начал [поддерживать Git относительно недавно](#). В отличие от GitHub, BitBucket позволяет совершенно бесплатно создавать как открытые, так и закрытые репозитории.

В чем состоит отличие Git от Subversion?

Главное отличие Git от [Subversion](#) заключается в том, что Git — *распределенная* система контроля версий. Звучит ужасающе, но на практике это означает очень простую вещь. Каждый разработчик держит у себя на диске отдельный репозиторий. Обратите внимание — не *копию* репозитория, не *некоторые ветки*, а тупо отдельный и при этом абсолютно полноценный репозиторий.

Пока мы работаем в рамках своего репозитория, все происходит в точности, как в Subversion. Мы коммитим и откатываем изменения, создаем, мержим и удаляем ветки, разрешаем конфликты и т.д. Помимо этого, предусмотрены команды для работы с репозиториями на удаленных машинах. Например, «git push» означает мерж локальных изменений в удаленный репозиторий, а «git pull» — наоборот, мерж изменений из удаленного репозитория в локальный. Обмен данными по сети обычно происходит с использованием протокола SSH.

В результате имеем:

- Git присущи все те же преимущества от использования VCS, что мы получаем в Subversion.
- Git дает нам нормальное шифрование «из коробки», безо всяких танцев с бубнами, как в случае с Subversion.
- Если сервер с «главным» репозиторием, куда пушит свои изменения все разработчики (хотя формально в Git нет никакого «главного» репозитория), вдруг прилет — ничего страшного. Делаем коммиты в локальный репозиторий и ждем, когда сервер вернется.
- Даже если сервер доступен, все равно удобнее сделать пятак локальных коммитов, а затем отправить их на сервер одним пушем.
- Сервер *вообще* не нужен. Вы можете использовать Git только локально. И не обязательно для работы с исходниками. Например, можно использовать Git для того, чтобы иметь возможность откатиться к предыдущим версиям файлов (каких-нибудь электронных таблиц) или вернуть случайно удаленные.
- Git не раскидывает по каталогам служебную информацию (помните «.svn»?), вместо этого она хранится только в корне репозитория.
- Git очень очень моден (хотя это далеко не единственная распределенная система контроля версий, например, есть Mercurial и Darcs), в связи с чем растет число разработчиков, использующих его. Как следствие, используя Git, легче получить помощь на каком-нибудь форуме или собрать команду разработчиков, знакомых с этой VCS.
- Существует множество полезных утилит для работы с Git — Qgit, gitk, gitweb и другие. «Из коробки» есть импорт и экспорт в/из Subversion/CVS.
- Git поддерживает многие хостинги репозиториев ([GitHub](#), [BitBucket](#), [SourceForge](#), [Google Code](#), ...) — есть из чего выбрать.
- Большой популярностью пользуется GitHub. Используя Git, вы увеличиваете вероятность того, что кто-то захочет безвозмездно написать патч для вашего open source проекта.

Пример использования Git

Я использовал Git при написании программы из заметки [Генерация почти осмысленных текстов на Haskell](#), снял под своей любимой FreeBSD. Вот как примерно выглядела моя работа с Git.

В первую очередь необходимо поставить Git:

```
pkg_add -r git
```

Затем создаем пару ssh ключей, если не создавали ее ранее:

```
ssh-keygen
cat ~/.ssh/id_rsa.pub
```

Заходим на BitBaket, создаем git-репозиторий под новый проект, а в свойствах аккаунта прописываем свой открытый ssh-ключ. Затем клонируем репозиторий:

```
cd ~/projects/haskell
git clone git@bitbucket.org:afiskon/hs-textgen.git
cd hs-textgen
```

Делаем какие-то изменения:

```
echo test > TODO.TXT
```

Добавляем новый файл в репозиторий и делаем коммит:

```
git add TODO.TXT
git commit -a
```

Поскольку я не указал описание коммита, запускается [редактор VIM](#), с помощью которого я и ввожу описание. Затем я отправляю все сделанные мною изменения на BitBaket:

```
git push origin
```

Допустим, теперь я хочу сделать некоторые изменения в проекте, но не уверен, выйдет ли из этого что-то хорошее. В таких случаях создается новая ветка:

```
git branch new_feature
git checkout new_feature
```

Работаем с этой веткой. Если ничего хорошего не вышло, возвращаемся к основной ветке (она же «trunk» или «ствол»):

```
git checkout master
```

Если вышло что-то хорошее, мержим ветку в master (о разрешении конфликтов рассказано в следующем параграфе):

```
git commit -a # делаем коммит всех изменений в new_feature
git checkout master # переключаемся на master
git merge new_feature # мержим ветку new_feature
```

Не забываем время от времени отправлять наш код на BitBucket:

```
git push origin
```

Если мы правим код с нескольких компьютеров, то перед началом работы не забываем «накатить» в локальный репозиторий последнюю версию кода:

```
git pull origin
```

Работа в команде мало чем отличается от описанного выше. Только каждый программист должен работать со своей веткой, чтобы не мешать другим программистам. Одна из классических ошибок при начале работы с Git заключается в push'e *всех веток*, а не только той, с которой вы работаете. Вообще я бы советовал первое время перед выполнением каждого push делать паузу с тем, чтобы подумать, что и куда сейчас уйдет. Для большей безопасности советую при генерации ssh-ключей указать пароль. Тогда каждый запрос пароля со стороны Git будет для вас сигналом «Эй, ты делаешь что-то, что затронет других».

Для работы с Git под Windows можно воспользоваться клиентом [TortoiseGit](#). Если память не подводит, для работы ему нужен [Git for Windows](#). Для генерации ключей можно воспользоваться утилитой [PuTTYGen](#). Только не забудьте экспортировать открытый ключ в правильном формате, «Conversions → Export OpenSSH key».

Следует отметить, что мне лично TortoiseGit показался не слишком удобным. Возможно, это всемо лишь дело привычки, но мне кажется с помощью удаленной или еще в чем-то вполне возможны. Если вы видите в этом разделе ошибку, отпишитесь, пожалуйста, в комментариях.

Шпаргалка по командам

В этом параграфе приведена сухая шпаргалка по командам Git. Я далеко не спец в этой системе контроля версий, так что ошибки в терминологии или еще в чем-то вполне возможны. Если вы видите в этом разделе ошибку, отпишитесь, пожалуйста, в комментариях.

Создать новый репозиторий:

```
git init project-name
```

Если вы планируете клонировать его по ssh с удаленной машины, также скажите:

```
git config --bool core.bare true
```

... иначе при git push вы будете получать странные ошибки вроде:

```
Refusing to update checked out branch: refs/heads/master
By default, updating the current branch in a non-bare repository
is denied, because it will make the index and work tree inconsistent
with what you pushed, and will require 'git reset --hard' to match
the work tree to HEAD.
```

Клонировать репозиторий с удаленной машины:

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

Если хотим пушить один код в несколько репозиториев:

```
git remote add remotename git@gitlab.example.ru:repo.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 SEDITOR:

```
git commit -a
```

Замержить *все ветки* локального репозитория на удаленный репозиторий (аналогично вместо origin можно указать и remotename, см выше):

```
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
```

Замержить 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 мержить 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 новый бранч (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 ветка
```

Просмотреть все существующие ветки:

```
git branch -a # | grep something
```

Замержить some_branch в текущую ветку:

```
git merge some_branch
```

Удалить бранч (после мержа):

```
git branch -d some_branch
```

Просто удалить бранч (тупиковая ветвь):

```
git branch -D some_branch
```

История изменений:

```
git log
```

История изменений в обратном порядке:

```
git log --reverse
```

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:

```
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
```

Отображаем табы как 4 пробела, например, в «git diff»:

```
git config --global core.pager 'less -x4'
```

Создание *глобального файла ignore*:

```
git config --global core.excludesfile ~/.gitignore_global
```

Разрешение конфликтов (когда оные возникают в результате мержа):

```
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
```

Иногда требуется создать копию репозитория или перенести его с одной машины на другую. Это делается примерно так:

```
mkdir -p /tmp/git-copy
cd /tmp/git-copy
git clone --bare git@example.com:afiskon/cpp-opengl-tutorial1.git
cd cpp-opengl-tutorial1.git
git push --mirror git@example.com:afiskon/cpp-opengl-tutorial1.git
```

Следует отметить, что Git позволяет использовать короткую запись хэшей. Вместо «d8578edf8458ce06fbc5bb76a58c5ca4a58c5ca4» можно писать «d8578ed» или даже «d857».

Дополнение: Также в 6-м пункте [Мини-заметок номер 9](#) приводится пример объединения коммитов с помощью [git rebase](#), а в 10-м пункте [Мини-заметок номер 11](#) вы найдете пример объединения двух репозиториев в один без потери истории.

Работа с сабмодулями

Более подробно сабмодули и зачем они нужны объясняется в заметке [Простой кроссплатформенный OpenGL-проект на C++](#). Здесь упомянем самое главное.

Добавить сабмодуль:

```
git submodule add https://github.com/glfw/glfw glfw
```

Инициализация сабмодулей:

```
git submodule init
```

Обновление сабмодулей, например, если после [git pull](#) поменялся коммит, на который смотрит сабмодуль:

```
git submodule update
```

Удаление сабмодуля производится так:

- Скажите `git rm --cached имя_субмодуля`;
- Удалите соответствующие строочки из файла .gitmodules;
- Также пройдите соответствующую секцию в .git/config;
- Сделайте коммит;
- Удалите файлы сабмодуля;
- Удалите каталог .git/modules/имя_субмодуля;

Дополнительные материалы

В качестве источников дополнительной информации я бы рекомендовал следующие:

- [Why Git is Better than X](#);
- [Хабрастатья о Почему Git?](#);
- [Книга «Волшебство Git» на русском языке](#);

Как обычно, любые замечания, дополнения и вопросы категорически приветствуются. И кстати, с наступающим вас!

Дополнение: [Практика работы с системами контроля версий](#)

Метки: [Разработка](#).

Понравился пост? Узнайте, как можно [поддержать развитие этого блога](#).

Также подпишитесь на [RSS](#), [ВКонтакте](#), [Twitter](#) или [Telegram](#).

Комментарии к этой записи в настоящее время закрыты.

29 Комментариев

Записки программиста

Политика конфиденциальности Disqus


Войти

❤️ Рекомендовать 15

🐦 Твитнуть

📧 Поделиться

Новое в начале

Максим • 4 года назад


Александр, большое Вам спасибо!

Именно с Вашей помощью я начал использовать GIT в разработке. Даже теперь не понимаю как это я столько лет прожил без этого. Правда стоит учитывать что разработку в Linux я начал относительно недавно и соответственно довольно быстро пришел к GIT.

Когда то давно был дефолт. Но это был ужас и под windows, вспоминать об этом не хочу.

👍 🐦 📧

📧 Поделиться

Дима • 5 лет назад

• Коротко о себе

Меня зовут Александр, позывной любительского радио R2AUK. Днем я обычный программист, а ночью превращаюсь в радиолюбителя-коротковолновика. Забавно я пишу об интересующих меня вещах и временами — просто о жизни.

Вы можете следить за обновлениями этого блога с помощью [RSS](#), [ВКонтакте](#), [Twitter](#) или [Telegram](#). Если вам нравится данный сайт, возможно, вы захотите поддержать его на [Patreon](#).

Мой контактный e-mail — mail@rax.mn. Если вы хотите мне написать, прощу предварительно ознакомиться с [этим FAQ](#). Если у вас технический вопрос, просьба адресовать его на форум [forum.devzen.ru](#).

Поиск

Популярные заметки

- Моя шпаргалка по работе с Git, 8416 просмотров за месяц
- Моя шпаргалка по работе в Vim, 7365 просмотров за месяц
- Начало работы с PostgreSQL, 2219 просмотров за месяц
- Шпаргалка по использованию умных указателей в C++, 2053 просмотра за месяц
- Выходим на радиолюбительские диапазоны 2 м и 70 см, 1950 просмотров за месяц
- Пишем пол микроконтроллеры STM32 в Arduino IDE, 1853 просмотра за месяц
- Шпаргалка по основным инструциям ассемблера x86/x64, 1553 просмотра за месяц
- Быстрое вхождение в Kubernetes, 1552 просмотра за месяц
- Настройка фреймла с помощью iptables за пять минут, 1542 просмотра за месяц
- Знакомство с компараторами на примере Matplotlib, 1484 просмотра за месяц
- Травим плату перекисью водорода с лимонной кислотой, 1386 просмотров за месяц
- Основы сборки проектов на C/C++ при помощи CMake, 1330 просмотров за месяц
- Сеты и примеры ячеек, которые помогут вам в освоении нового языка программирования, 1305 просмотров за месяц
- Установки и настройка OpenVPN в Ubuntu Linux за 5 минут, 1295 просмотров за месяц
- Поятка по virtualenv и изолированным проектам на Python, 1266 просмотров за месяц
- Потоковая репликация в PostgreSQL и пример фреймворка, 1261 просмотр за месяц
- Как спроектировать схему базы данных, 1230 просмотров за месяц
- Пяточка по отладке при помощи GDB, 1061 просмотр за месяц
- Устанавливаем связь из Prometheus и Grafana, 1022 просмотра за месяц

Копирование представленных на данном сайте материалов любыми способами не возбраняется. Указание ссылки на оригинал приветствуется. © 2009–2020 Записки программиста

LiveInternet

СЕРВИС СТАТИСТИКИ