Записки программиста

Блог о программировании, операционных системах, СУБД, девайсах, сетях, алгоритмах и пр

Моя шпаргалка по работе с Git

28 декабря 2011

Некоторое время назад я открыл для себя Git. И знаете, я проникся. То есть, по-настоящему проникся. Теперь я использую Git не только на работе (где я с ним, собственно, познакомился), но и для своих проектиков, 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 бранчи*, 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, мерджим и удаляем бранчи, разрешаем конфликты и тд. Помимо этого, предусмотрены команды для работы с репозиториями на удаленных машинах. Например, «git push» означает мердж локальных изменений в удаленный репозиторий, а «git pull» — наоборот, мердж 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 пушат свои изменения все разработчики (хотя формально в Git нет никакого «главного» репозитория), вдруг прилег — ничего страшного. Делаем коммиты в локальный репозиторий и ждем, когда сервер вернется.
- Даже если сервер доступен, все равно удобнее сделать пяток 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. А в этой заметке рассказывается, как хранить git-репозиторий на флешке.
- Git не раскидывает по каталогам служебную информацию (помните «.svn»?), вместо этого она хранится только в корне репозитория.
- Git нынче 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 поддерживают 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 OpenSource проекта.

Пример использования Git

Я использовал Git при написании программы из заметки Генерация почти осмысленных текстов на Haskell, сидя 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 БитБакет, 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 БитБакет:

```
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, мерджим 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 # мерджим 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'е *всех веток*, а не только той, с которой вы работали. 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 изменяет, для нормальной работы ему нужен MSysGit. 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 каким-то глючноватым и вообще не слишком удобным. Возможно, это всего лишь дело привычки, но мне кажется намного удобнее работать с Git из консоли, чем с помощью контекстного меню в Проводнике. Так что по возможности я бы советовал работать с Git в Юниксах. В крайнем случае можно поставить виртуальную машину, установить под ней FreeBSD (безо всяких GUI) и работать в этой виртуальной машине.

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
```

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 мерджить 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 ветвь

Просмотреть все существующие ветви:

git branch -a # | grep something

Замерджить some_branch в текущую ветку:

git merge some_branch

Удалить бранч (после мерджа):

git branch -d some_branch

Просто удалить бранч (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».

Дополнительные материалы

В качестве источников дополнительной информации я бы рекомендовал следующие:

- Why Git is Better than X;
- Хабростатья «Почему Git?»;
- Хабровопрос о преимуществах Git и других DVCS;
- Книга «Pro Git» на русском языке;
- Книга «Волшебство Git» на русском языке;
- How To Create a Remote Shared Git Repository;

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

Дополнение: Практика работы с системами контроля версий

Метки: Разработка.

Подпишись через RSS, E-Mail, Google+, Facebook, Vk или Twitter.

Понравился пост? Поделись с другими: (необходимо включить JS)

Для отображения комментариев необходимо включить JavaScript!

- # Коротко о себе

  Привет! Меня зовут Александр. Здесь я пишу об интересующих меня вещах и временами — просто о жизни.

  Вы можете следить за обновлениями этого блога с помощью RSS, E-Mail, Google+, Facebook, ВКонтакте или Twitter.

  Связаться со мной можно, отправив письмо на *mail@eax.me*. Если у вас вопрос и он не срочный, то желательно предложить его в темах слушателей к ближайшему выпуску DevZen.

- Поиск

- # Популярные заметки

  - Моя шпаргалка по работе с Git, 15955 просмотров за месяц
  - Итак, вы решили стать программистом, 7198 просмотров за месяц
  - Начало работы с PostgreSQL, 4794 просмотра за месяц
  - Моя шпаргалка по работе в Vim, 4133 просмотра за месяц
  - Советы и примеры задач, которые помогут вам в освоении нового языка программирования, 3557 просмотров за месяц
  - Краткая шпаргалка по сочетаниям клавиш в IntelliJ IDEA, 2929 просмотров за месяц
  - Краткий обзор GUI-фреймворков для Java и мое первое простенькое GUI-приложение на Swing, 2278 просмотров за месяц
  - Непрерывная интеграция с Jenkins, 2248 просмотров за месяц
  - Пример простейшей многопоточной программы на WinAPI, 1742 просмотра за месяц
  - Памятка по регулярным выражениям, 1629 просмотров за месяц
  - Некоторые интересные отличия PostgreSQL от MySQL, 1452 просмотра за месяц
  - Redis и области его применения, 1053 просмотра за месяц