Создание системы Push-to-Deploy с git

11 Ноябрь 2017 1 комментарий

Впервые я настраивал Push-to-Deploy систему используя git и puppet для стороннего проекта пару лет назад. Она работала так хорошо, что я начал использовать опыт процесса разработки NMC на всех новых проектах, начиная с прошлого года. Она предлагает простоту модели «Push-to-deploy», которая была полностью внедрена на Heroku, с полным контролем и гибкостью в среде операционной системы

Я начал думать о следующем шаге этой системы на Didsum, и использовать ее для более чем для простой публикации проектов. Модель Push-to -____ является мощной и простой в использовании, если вы знаете как компоненты взаимодействуют друг с другом. **В этой статье я покажу как настроить Push-to-Deploy систему с нуля.**

Подготовка репозиториев

Начнем с того, что разместим репозиторий git для разработки, удаленный репозиторий git и deploy директорию в том же локальном каталоге.

```
mkdir push-to-deploy
cd push-to-deploy
mkdir {development,remote,deploy}
```

Отлично, теперь настроим удаленный git репозиторий. На настоящем проекте это будет директория на боевом сервере. Существует специальная настройка для репозиториев git, целью которых является получение push-запросов от разработчиков, их называют «чистыми» репозиториями. Вы можете узнать больше о «чистых» репозиториях, но для нас их цель - просто получить push-запросы

```
cd remote
git init --bare
cd ...
```

Восхитительно, давайте настроим наш репозиторий git для разработки. (Вы также можете скопировать один из ваших git-проектов в папку разработки.)

```
cd development
git init
echo "Hello, world." >> file.txt
git add file.txt
git commit -m 'First commit.'
```

Теперь у нас есть репозиторий git для разработки с первым коммитом. Последним этапом нашей подготовки станет - регистрация «удаленного» репозитория. Если вы никогда не работали с удаленными репозиториями, есть руководство на сайте git.

```
git remote add production ../remote
git push production master
```

Ура, вы только сделали push вашего первого коммита с разработки на чистый, "удаленный" репозиторий. Теперь мы готовы настроить push-to-deploy.

Настройка Push-to-Deploy

Теперь, когда наш удаленный репозиторий настроен, мы готовы написать сценарий для того, что он будет делать, когда он получит push-запрос Перейдем к папке **hooks** нашего удаленного репозитория. **Перехватчики** - это скрипты, которые запускаются при определенных событиях.

```
cd ../remote/hooks
touch post-receive
chmod +x post-receive
```

Перехватчик, который нам нужен для push-to-deploy, - **post-receive**. Он запускается после того, как завершился процесс пуша коммита. В приведенных выше командах мы создаем файл сценария **post-receive** с установкой времени последнего изменения файла и удостоверяемся, что это исполняемый файл.

Затем откройте файл сценария в предпочитаемом текстовом редакторе. Скопируйте содержимое ниже:

```
#!/usr/bin/env ruby
# post-receive

# 1. Read STDIN (Format: "from_commit to_commit branch_name")
from, to, branch = ARGF.read.split " "

# 2. Only deploy if master branch was pushed
if (branch =~ /master$/) == nil
    puts "Received branch #{branch}, not deploying."
    exit
end

# 3. Copy files to deploy directory
deploy_to_dir = File.expand_path('../deploy')
`GIT_WORK_TREE="#{deploy_to_dir}" git checkout -f master`
puts "DEPLOY: master(#{to}) copied to '#{deploy_to_dir}'"

# 4.TODO: Deployment Tasks
# i.e.: Run Puppet Apply, Restart Daemons, etc
```

Давайте пройдемся по всем шагам

1. Когда git запускает **post-receive**, данные о том, что было получено, предоставляются сценарию через STDIN. Он содержит три аргумента, разделенных пробелами: предыдущий идентификатор коммита HEAD, новый идентификатор коммита HEAD и имя ветки, которая была выбрана для внесения изменений. Мы считываем эти значения и присваиваем им переменные из, в, и ветки, соответственно.
2. Наша цель - автоматизировать **push-to-deploy**. Предположим, что рабочие процессы, которые поддерживают работу проекта находятся в ветке master, тогда нам нужно выйти из этого сценария до выката, если ветка с которой выполнили команду **push**, не является веткой **master**
3. Первым шагом публикации проекта является "проверка", в основном экспорт или копирование файлов из ветки **master** в директорию где находится наш рабочий проект. (Помните, что в этой демонстрации это придуманный «разворачивающий» каталог, на деле это может быть */var/www* или там где вы разместили его .)
4. Теперь, когда наша директория с деплоем обновлена, мы можем запустить необходимые задачи для развертывания. Это могут быть применение сценариев **Puppet** (я скоро напишу пост об этом сценарии), перезапуск веб-сервера или сервера приложения, очистка файлов кэша, перекомпиляция и т. д. Неважно, какие шаги вам обычно нужно делать вручную после обновления файлов вашего проекта, автоматизируйте их здесь!

Сохраните свой **post-receive**, и давайте проверим его!

Тестирование с помощью пуша

Мы можем проверить наш сценарий вручную, создав новый коммит в нашем каталоге разработки и запушив его:

```
cd ../../development
echo "New line." >> file.txt
git add file.txt
git commit -m 'Testing push-to-deploy'
git push production master
```

На выходе команды git push вы должны увидеть строки, начинающиеся с «remote:». Эти строки являются результатом нашего сценария post-receive:

```
Already on 'master'
DEPLOY: master($TO_ID) copied to 'push-to-deploy/deploy'
```

Первая строка представляет собой "кричащий" вывод команды git checkout на шаге 3, мы можем ее игнорировать. Вторая строка - из команды puts, также из шага 3 в нашем сценарии post-receive.

Теперь директория, в которую публикуем проект, должна быть заполнена и обновлена:

```
ls ../deploy
diff file.txt ../deploy/file.txt
```

Чудесно, не так ли?

Тестирование без использования метода push

Когда вы работаете с перехватчиком post-receive, это раздражает, а также портит историю коммитов вашего проекта и пушит при каждом изменении. К счастью, так как это всего лишь сценарий, мы можем подделать его из командной строки.

```
cd ../remote
git log -2 --format=oneline --reverse
```

Во-первых, нам нужно получить идентификаторы наших последних двух коммитов. Команда git log, выше, даст нам эти два идентификатора в том порядке, в котором вы захотите заменить переменные $ FROM_ID и $ TO_ID, соответственно.

```
echo "$FROM_ID $TO_ID master" | ./hooks/post-receive
```

Этот метод упрощает настройку ваших post-receive перехватчиков, позволяя вам быстро перебирать сценарий и выполнять его повторно.

Следующие шаги

В этой статье мы рассмотрели, как настроить **push-to-deploy** с **git**. Для реального проекта ваши «удаленные» и «публикуемые» папки обычно размещают на сервере, а не локально. Детали и правильная настройка SSH были бы уже не в рамках этой статьи(обратите внимание на себя: я тоже должен писать в конфигурации SSH!).

Здесь вы можете определить, какие действия нужно автоматизировать! Удачных пушей!

git

Like

Popular posts

Our readers also like these articles

Понимание нативных методов javascript массивов

22 Август 2016 0 комментариев

CMS 1С-Битрикс - копирование файлов

8 Октябрь 2016 0 комментариев

Команды, важные в работе с Git

24 Сентябрь 2016 0 комментариев

Комментарии

1 комментарий

Anton — 15 November в 11:14

Чоткий статья жиесть

Leave your comment

Your opinion is important to us

Type your e-mail

email@email.com

What's your name?

Your name

Commentary

text here

Leave a commentary