Git-Хуки

Как и многие другие системы контроля версий, Git предоставляет возможность запуска пользовательских скриптов в случае возникновения определенных событий. Такие действия называются хуками и разделяются на две группы: серверные и клиентские. Если хуки на стороне клиента запускаются такими операциями как слияние или создание коммита, то на стороне сервера они инициируются сетевыми операциями, такими как получение отправленного коммита. Хуки часто используются для широкого круга задач.

### **Установка хука**

Хуки хранятся в подкаталоге hooks относительно основного каталога Git. Для большинства проектов это .git/hooks. Когда вы инициализируете новый репозиторий командой git init, Git наполняет каталог hooks примерами скриптов, большинство из которых готовы к использованию, при этом каждый из них содержит документацию по используемым входным данным. Все примеры представлены в виде шелл скриптов, содержащими код на Perl, но вы можете использовать любой язык для написания скриптов — главное правильно именовать исполняемые файлы. Если вы решите использовать какой-либо из предустановленных скриптов, то достаточно его просто переименовать, убрав суффикс .sample.

Для подключения собственного скрипта достаточно задать ему соответствующее имя, поместить в подкаталог hooks основного каталога Git и сделать его исполняемым. Далее, мы рассмотрим наиболее часто используемые хуки.

### **Клиентские Хуки**

Для клиента существует множество различных хуков. В этой главе они разделены на хуки уровня коммита, уровня e-mail и прочие.

| Примечание | Необходимо отметить, что клиентские хуки **НЕ** копируются при клонировании репозитория. Если вы намерены использовать такие скрипты для обеспечения соблюдения политики, то вам следует использовать серверные хуки; например [Пример принудительной политики Git](https://git-scm.com/book/ru/v2/ch00/r_an_example_git_enforced_policy). |
| --- | --- |

#### **Хуки уровня коммита**

Первые четыре хука работают во время создания коммитов.

Первым запускается pre-commit хук, до того как вы напечатаете сообщение коммита. Он используется для проверки данных перед созданием коммита и позволяет увидеть если вы что-то забыли, запустить тесты, или выполнить другую необходимую проверку кода. Создание коммита будет отменено если выполнение хука завершится с кодом отличным от нуля. Пропустить выполнение хука можно с помощью git commit --no-verify. С помощью этого хука можно проверять стиль кода (запустить lint или аналог), проверять наличие пробелов в конце строк (именно это делает стандартный хук) или проверять наличие документации для новых методов.

Хук prepare-commit-msg запускается до вызова редактора сообщения коммита, но после создания стандартного сообщения. Это позволяет вам изменить стандартное сообщение коммита до того, как автор коммита увидит его. Хук принимает несколько параметров: путь к файлу, содержащему сообщение коммита, тип коммита и SHA-1-хеш, если текущий коммит является исправлением существующего. Для обычных коммитов этот хук бесполезен, однако находит своё применение для коммитов, где сообщение генерируется автоматически, например, для сообщений на основе шаблонов, коммитов слияния, сжимаемых и исправляемых коммитов. Его можно использовать для программного заполнения шаблона коммита необходимой информацией.

Хук commit-msg принимает один параметр — путь к временному файлу, содержащему указанное разработчиком сообщение коммита. Если скрипт завершается с ненулевым кодом, то Git отменяет создание коммита, поэтому вы можете использовать этот хук для валидации состояния проекта или сообщения коммита до того как он будет создан. В последнем разделе этой главы мы покажем как использовать этот хук для проверки сообщения коммита на соответствие заданному шаблону.

Хук post-commit запускается после того, как коммит создан. Он не принимает никаких параметров, но вы можете легко получить информацию о последнем коммите выполнив git log -1 HEAD. Обычно, этот скрипт используется для уведомлений или чего-то подобного.

#### **Хуки для рабочего процесса на основе E-mail**

Для рабочего процесса на основе e-mail на стороне клиента можно задать три хука. Все они вызываются командой git am, поэтому если вы не используете её в своём рабочем процессе, то можете смело перейти к следующему разделу. Если вы получаете по почте патчи, подготовленные командой git format-patch, то найдёте здесь немного полезной информации.

В первую очередь запускается хук applypatch-msg. Он принимает единственный аргумент: имя временного файла, содержащее предлагаемое сообщение коммита. Git отменит патч если этот скрипт завершится с ненулевым кодом. Этот хук можно использовать для проверки формата сообщения или для его нормализации, если ваш скрипт умеет редактировать сообщение коммита.

Следующим запускается хук pre-applypatch. Здесь всё немного запутанно: хук запускается *после* применения патча, но перед созданием коммита, что позволяет проверить состояние кода до создания коммита. В этот момент можно запустить тесты или другим способом проверить состояние проекта. Если что-то пропущено или тесты не пройдены, скрипт должен завершиться с ненулевым кодом, что остановит выполнение команды git am, а коммит не будет создан.

Последним запускается хук post-applypatch, который вызывается уже после того как коммит создан. Вы можете его использовать для уведомления группы или автора патча о его применении. С помощью этого хука вы не можете прервать процесс применения патча.

#### **Прочие хуки на стороне клиента**

Хук pre-rebase выполняется при попытке перебазирования и может остановить процесс вернув ненулевой код. Его можно использовать для запрета перебазирования уже отправленных коммитов. Git устанавливается с примером такого скрипта, однако он делает некоторые допущения, которые могут не соответствовать вашему рабочему процессу.

Хук post-rewrite запускается командами, которые заменяют коммиты: git commit --amend и git rebase (но не git filter-branch). Его единственный аргумент — команда, которая инициировала перезапись, а список перезаписанных изменений передаётся через stdin. Его применение практически аналогично хукам post-checkout и post-merge.

После успешного выполнения git checkout запускается хук post-checkout; его можно использовать для настройки рабочего каталога в соответствии с требованиями проекта. Например, перемещение в рабочий каталог больших бинарных файлов, которые не должны отслеживаться, автогенерация документации и тому подобное.

Хук post-merge запускается после успешного выполнения команды merge. Его можно использовать для восстановления данных в рабочем каталоге, которые Git не может отслеживать, такие как права доступа. Так же этот хук может проверять наличие внешних по отношению к Git файлов, которые вы захотите скопировать при внесении изменений.

Хук pre-push выполняется во время работы команды git push: после обновления удалённых ссылок, но до непосредственной отправки данных. Он принимает название и путь удалённого репозитория как параметры, а список изменений для отправки через stdin. Его можно использовать для валидации набора изменений до их реальной отправки (ненулевой код отменяет отправку изменений).

Время от времени, как часть нормальной работы, Git выполняет сборку мусора вызовом команды git gc --auto. Хук pre-auto-gc вызывается непосредственно перед выполнением операции сборки мусора и может быть использован для уведомления о её запуске или для её отмены, если сейчас не самое подходящее для этого время.

### **Хуки на сервере**

В дополнение к хукам на стороне клиента, как системный администратор вы можете использовать несколько важных хуков на сервере для вашего проекта, тем самым обеспечив выполнение практически любой политики. Эти скрипты выполняются до и после отправки на сервер. Pre-хуки могут возвращать ненулевой код в любой момент, что отменит передачу и отправит сообщение об ошибке клиенту; таким образом вы можете реализовать сколь угодно сложную политику.

#### **pre-receive**

Хук pre-receive запускается первым при старте получения данных от клиента. Он получает на stdin список отправленных изменений и если завершается ненулевым кодом, то ни одно из них принято не будет. Этот хук можно использовать для того, чтобы убедиться что все изменения можно применить методом перемотки вперёд, а так же для проверки прав доступа.

#### **update**

Хук update очень похож на pre-receive, за исключением того, что он выполняется для каждой ветки, которую отправитель пытается обновить. Если отправитель пытается отправить изменения в несколько веток, то pre-receive хук будет вызван однократно, а update выполнен для каждой изменяемой ветки. Вместо чтения из stdin, хук принимает три аргумента: название ссылки (ветка), SHA-1-хеш, на который указывала ссылка до отправки, и SHA-1-хеш коммита, отправляемого пользователем. Если скрипт завершается ненулевым кодом, то отклоняются все изменения только для текущей ветки, при этом изменения для других веток всё ещё могут быть применены.

#### **post-receive**

Хук post-receive вызывается после окончания всего процесса и может быть использован для обновления других сервисов или уведомления пользователей. Он принимает на stdin те же данные, что и хук pre-receive. Использовать его можно, например, для e-mail рассылки, для уведомления сервера непрерывной интеграции или обновления системы управления задачами — разобрав сообщение коммита, можно определить необходимость создания, изменения или закрытия каких либо задач. Этот хук не может прервать процесс, но клиент остаётся подключённым пока он не завершится, поэтому избегайте выполнения длительных операций.

| Подсказка | Если вы пишете сценарий/хук, который другие должны будут прочитать, используйте длинные версии параметров командной строки; через шесть месяцев вы будете нас благодарить. |
| --- | --- |

### 

### 

### 

### **Примеры использования**

**Автоматический запуск тестов** (pre-commit):

| #!/bin/sh  npm test |
| --- |

**Проверка сообщения коммита** (commit-msg):

| #!/bin/sh  if ! grep -q "^[A-Z]+-[0-9]+" "$1"; then  echo "Commit message must contain JIRA ticket (e.g., PROJ-123)"  exit 1  fi |
| --- |