Rapor Edilecek Hooklar

pre-commit

`git-commit` tarafından çağrılır. Eğer kullanılmaması gerekirse `--no-verify` seçeneği ile komut çalıştırılmadan geçilebilir. Parametre almaz. Yorum yapılmadan ve hatta yorum logları tutulmadan önce çalıştırılır. Başarılı sayılabilmesi için sonuç değerinin `0` olması gerekir.

```
  #!/bin/sh
  # verilen dizinden dosya silinmesini engelleyen sh betiği
  folder=dont_delete

  res=$(git diff --cached --name-status | grep -P "D\t${folder}/" | sed
s_^D[[:space:]]\[^/]*\/__)

  if test -z "$res"; then exit 0; fi

  echo "HATA: $folder dizininden aşağıdaki dosyalar silinmeye çalışılıyor:"
  for i in $res; do; echo " -> $i"; done

  echo "Commit engellendi!"
  exit 1
```

post-commit

`git-commit` tarafından çağırılır. Parametre almaz, `commit` yapıldıktan sonra çalıştırılır. Uyarı amaçlıdır ve `commit`'in sonucunu etkilemez.

```
#!/bin/sh

echo '######## UYARI ########'
echo '-> Bu projede çalışanların dikkatine: '
echo '   * ikinci bir emre kadar izinleriniz iptal edilmiştir'
```

prepare-commit-msg

`pre-commit` hook'undan sonra çağrılır. Tetiklendiği zaman bir text editörü içerisinde yorum mesajı üretir. Başarılı sonuç dönmesi için `0` dönmesi gerekir. Eğer sıfırdan farklı bir sonuç dönerse `git commit` komutu iptal edilir. `commit`'lerde otomatik olarak oluşturulması istenen mesajlar için sıkça kullanılır.

`prepate-commit-msg` scripti 1-3 aralığında parametre alır.

1. Basılacak mesajı içeren geçici dosyanın adı. Bu dosya değiştirilerek onay mesajı değiştirilebilir
2. `commit` türü. `message`, `template`, `merge`, `squash` türlerinde olabilir. `message` türü için -m veya -F parametresiyle, `template` için -T parametresiyle kullanılır.

3. SHA1 türünde uygun yorum hashi de alabilmektedir. Bu hashi kullanabilmek için `-c`, `-C` veya `--ammend` parametrelerini girmek gerekmektedir.

```
  #!/usr/bin/env ruby
  # Grup motivesi arttırmak için commit'in sonuna bir söz ekler
  msg_file = ARGV[0]
  content  = File.read(msg_file)
  QUOTE_OF_DAY = '"Yarını iyileştirmenin tek yolu bugün neyi yanlış yaptığını
bilmektir!" ~Robin Sahrman~'.freeze

  content += "\n" + QUOTE_OF_DAY

  File.write(msg_file, content)
```

commit-msg

Bu kanca `git-commit` veya `git-merge` tarafından çağrılır. `--no-verify` seçeneği ile geçilebilir. Önerilen `commit` log mesajı tarafından tek parametre ile çalıştırılır. Sıfır harici bir değer ile sonlandırılması durumunda `commit` iptal edilir.

```
  #!/usr/bin/env ruby

  # yasakli kelimeleri içeren commit mesajlarını engelleyen ruby betiği
  dosya_adi         = ARGV[0]
  dosya             = File.read(dosya_adi)
  yasakli_kelimeler = ["yasakli", "kelime", "at"]

  yasakli = nil
  yasakli_kelimeler.any? { |v| yasakli = v if icerik.include? v }

  if yasakli
    puts "HATA: Herkese açık bir projede \"#{yasakli}\" gibi kelimelerin
kullanılması uygun değildir!"
    puts "Commit iptal ediliyor!"
    exit 1
  end
```

pre-receive

`git-receive-pack` tarafından çağrılır. Biri depoya birşey yüklemek için `git push` çalıştırdığında her commit için ayrı ayrı çalışır. Uzak sunucu da bulunmalıdır. Argüman almaz fakat ancak her bir `ref` için güncellenmek üzere standart girdiyle formatın bir satırını alır. Bu satır `<eski-değer> SP <yeni-değer> SP <ref-adı> LF` şeklinde değerlerden oluşur. `<eski-değer>` `ref` içerisinde saklanan eski nesne adı, `<yeni-değer>` `ref` içerisinde bulunan yeni nesne adını, `<ref-adı>` da `ref`'in tam adını içerir.

```
#!/bin/bash

while read eski_sha yeni_sha refname
do
  if git diff "$eski_sha" "$yeni_sha" | grep -qE '^\+(<<<<<<<|>>>>>>>)'; then
    echo "Conflict'e sebep olacak işaretler bulundu $(basename "$refname")."
    git diff "$eski_sha" "$yeni_sha" | grep -nE '^\+(<<<<<<<|>>>>>>>)'
    exit 1
  fi
done
```

update

pre-receive'den sonra git receive-pack tarafından "ref adı, eski-sha1, yeni-sha1" değerleriyle çağırılır. Uzaktak, depo ref'i güncellemeden önce update hook çalıştırılır. Eğer sıfır dönerse ref güncellemesi başarısız sayılır ve iptal edilir.

```
#!/usr/bin/env ruby
branch      = ARGV[0]
eski_commit = ARGV[1]
yeni_commit = ARGV[2]

if eski_commit == yeni_commit
  puts "HATA: #{branch}, içerisinde son commit isimleri aynı: #{eski_commit}"
  puts "İşlem başarısız.."
  exit 1
end
```

```
#!/usr/bin/env ruby
branch      = ARGV[0]
eski_commit = ARGV[1]
yeni_commit = ARGV[2]

puts "#{branch}, #{eski_commit}'ten #{yeni_commit}'e taşınıyor..."
```

post-update

git dosya eklediğinde veya dosyaları güncellediğinde git-receive-pack tarafından çağırılır. Tüm ref'ler güncellendikten sonra uzak depoda bir kez çalıştırılır. Her biri gerçekten güncellenen ref adı olan değişken sayıda parametre alır. Birincil amacı bildirimdir. git'in aldığı paketleri etkileyemez.

Güncelleme sonrasında, push edilen HEAD'in ne olduğu içinde bulunmasına rağmen orijinal ve değiştirilmiş değerlerin içeriği olmadığı için log tutulması konusunda zayıf bir kaynaktır.

pre-auto-gc

git gc --auto tarafından çağırılır. Parametre almaz. Depoları temizlemeden önce bazı kontrolleri yapmak için kullanılır. Otomatik yeniden paketleme işlemini durdurmak için, uygun bir mesaj verdikten sonra sıfır olmayan bir durumdan çıkmalıdır.

```
#!/bin/sh
# batarya kontrolu. Sadece bilgisayarımda sarj aleti takılı mı kontrolü
yapıyor

if test "$(cat /sys/class/power_supply/AC/online 2>/dev/null)" = 1
then
  exit 0
fi

echo "Auto packing deferred; not on AC"
exit 1
```

pre-push

Uzaktaki durumu kontrol ettikten sonra, ancak herhangi bir şey push yapılmadan önce `git push` taradından çağırılır. Sıfır olmayan bir sonuç dönerse, hiçbir şey push yapılmaz. 2 tane parametre alır. Birincisi parametre gönderme yapılan uzak sunucunun adıdır. İkincisi parametre gönderme yapılan URL'dir. Eğer gnerme yapılan uzak sunucuya isim verilmezse iki parametre de eşit olacaktır.

Commitler hakkındaki bilgiler, `<yerel ref> <yerel sha1> <uzak_sunucu ref> <uzak_sunucu sha1>` şeklinde standart girdiden verilir.

```
#!/bin/bash
# nopush anahtar kelimesini içeren commitleri alarak son kontrol sağlatır
word=nopush
COMMITS=$(git log --grep "$word" --format=format:%H)

if [ "$COMMITS" ]; then exitmaybe=1; fi

if [ $exitmaybe -ne 1 ]; then exit 0; fi

while true
do
  echo "Commitleri yollamak istediğinize emin misiniz?[E/h] "
  read -r REPLY <&1
  case $REPLY in
    [Ee]* ) break;;
    [Hh]* ) exit 1;;
    * ) echo "Evet için e, Hayır için h kullaniniz";;
  esac
done
```

pre-rebase

`git-rebase` tarafından çağırılır. Bir dalda tekrar `rebase` yapılmasını önler. Bir veya iki parametreyle çağırılabilir. İlk parametre, `fork`'un ters sırada elde edilmiş halidir. İkinci parametre ise `rebase` edilmiş daldır ve geçerli dalı yeniden birleştirirken ayarlanmaz.

```
#!/bin/sh
basebrach="$1"
topic="refs/heads/$2"
# topic tamamen merge edildi mi kontrol eder
if test -z `git rev-list --pretty=oneline ^master "$topic"`
then
  echo >&2 "$topic is fully merged to master"
  exit 1 ;
fi
```

post-receive

post-update'e benzer tavırlar sergiler. Farkları ise, isimlerinin yanı sıra tüm referansların hem eski hem de yeni değerlerini barındırır.

```
#!/bin/sh
# git çalışma ağacının yolunu değiştir

echo "Hook çalışıyor..."
export GIT_WORK_TREE=/home/tayak/proje/www/
git checkout -f
rm -rf /home/tayak/proje/www/temp
```

post-checkout

git-checkout çalışma ağacını güncelledikten sonra çalıştırılır. Üç parametre alır: önceki HEAD'in ref'i, yeni HEAD'in ref'i ve dalın değişip değişmediği bilgisi. Sonucu git-checkout'u etkilemez.

post-merge

Yerel bir depoda bir git çekme işlemi yapıldığında git-merge tarafından çağırılır. Tek parametre alır, yapılan birleşmenin squash birleşmesi olup olmadığını belirten bir durum bayrağı. Depoları birleştirme başarısız olursa çalıştırılmaz. Ayrıca post_merge sonucu birleştirme sonucunu etkileyemez.

rebase

Squash birleştirmesi ve onarım işlemi için, ezilen edilen commitler, ezme işlemine yeniden yazılmış şekilde aktarılır.
commitler kesi olarak rebase tarafından işlendiği sırada listelenir.

pre-applypatch

git-am tarafından çağrılır. Parametre almaz. Düzeltme eki uygulandıktan sonra ancak bir işlem yapılmadan önce çağrılır. Eğer dönen sonuç sıfır değilse, çalışma ağacı yapılan yamalardan sonra commit edilmez. Mevcut çalışma ağacını denetlemek ve geçerli testi geçemezse bir commiti reddetmek için kullanılabilir.

applypatch-msg

`git-am` tarafından çağrılır. Önerilen işlem günlüğü iletisini tutan dosyanın adı tek bir parametre alır. Sıfır olmayan sonuç dönerse yama uygulamadan önce `git am`'in iptaline sebep olur.

sendemail-validate

`git-send-email` tarafından çağrılır. Gönderilecek e-postayı tutan dosyanın adı tek bir parametre alır. Sıfır olmayan bir durumdan çıkmak, git gönder e-postasının herhangi bir e-posta göndermeden önce iptal edilmesine neden olur.

fsmonitor-watchman

`core.fsmonitor` yapılandırma seçeneği .git/hooks/fsmonitor-watchman olarak ayarlandığında çağırılır. İki parametre alır, birinci parametre versiyon bilgisi, ikinci parametre ise zaman bilgisidir. Zaman bilgisi 1 Ocak 1970'den başlayarak şimdiye kadar geçen nanosaniyeler şeklinde tutulur. Çalışma yolları, çalışma dizinin köküne göre olmalı ve tek bir NUL ile ayrılmalıdır.

Çıkış durumu, git'i aramasını sınırlamak için kancadaki verileri kullanıp kullanmayacağını belirler. Hata durumunda, tüm dosya ve klasörleri doğrulamak için yapılan değişikler geri alınacaktır.