applypatch-msg

“git am” komutu ile ilişkilendirilir. commit mesajı dosyasını düzenleyebilir ve genellikle bir yapılan eklemelerin iletisini bir projenin standart standartlarına göre doğrulamak veya etkin biçimde biçimlendirmek için kullanılır. Sıfır olmayan bir çıkış durumu commiti iptal eder.

1 parametre alır: Önerilen commit mesajını içeren dosyanın adı

pre-applypatch

“git am” komutu ile ilişkilendirilmiştir. Sistemde eksik görülen bir kısma yapılan eklemeler uygulandıktan sonra, değişiklikler yapılmadan önce çağrılır. Sıfır olmayan bir durumdan çıkmak değişiklikleri kabul edilmeyen bir durumda bırakacaktır. Değişiklikleri yapmadan önce ağacın durumunu kontrol etmek için kullanılabilir.

Parametre almaz.

post-applypatch

“git am” komutu ile ilişkilendirilmiştir. Sistemde eksik görülen bir kısma yapılan eklemeler uygulandıktan ve commit edildikten sonra çalıştırılır. Commit edildikten sonra çalıştırıldığı için işlemi iptal edemez ve çoğunlukla bildirimler oluşturmak için kullanılır.

Parametre almaz.

pre-commit

“git commit” komutu ile ilişkilendirilmiştir. Bu hook, önerilen commit mesajını almadan önce çağrılır. Mesaj yerine commitin kendisini kontrol etmek için kullanılır.

Parametre almaz.

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

# yasaklı klasörününe ekleme yapılmamasını engelleyen betik
# eklenen dizin isimlerini al

folder = 'yasakli/'

files = %x[git diff --cached --name-status | grep -P "^A\t#{folder}"]

# eşleşme yoksa doğru olarak dön
exit 0 if files.empty?

# eklenen dosya isimlerini ayıkla
yasaklilar = files.split("\n").map do |file_name|
  file_name.split("#{folder}").last
end

puts "Commit engellendi!!"
puts "Aşağıdaki dosyaları silip tekrar deneyiniz:"
yasaklilar.each { |f| puts " * #{folder}#{f}" }
```

```
exit 1
```

prepare-commit-msg

“git commit” komutu ile ilişkilendirilmiştir. Sadece geçerli mesaj editörünü başlatmadan önce, varsayılan commit mesajını aldıktan sonra çağrılır. Sıfır olmayan bir çıkış commiti iptal eder. Bu, mesajı engellenemeyecek şekilde düzenlemek için kullanılır.

1 ila 3 arasında parametre alır: 1 mesajın bulunduğu geçici alan, 2 commit in türü (git commit ten sonra -m, -t girildiyse yada merge işlemi yapıldıysa bu bilgi olur) 3- SHA1 git commit -c olarak çağırıldıysa commitin özet değeri

```ruby
#!/usr/bin/env ruby
# commit dosyasında yorum satırı hariç satırların md5
# değerini alarak yorumun başına ekler
require 'digest'

commit_msg_file = ARGV[0]

text = File.readlines(commit_msg_file).map { |l|
  l.gsub!(/^[#].*/, '')
}.join

text = "md5: " + Digest::MD5.hexdigest(text) + "\n" + text

File.write(commit_msg_file, text)
```

commit-msg

“git commit” komutu ile ilişkilendirilmiştir. Bir standarda uygunluğu sağlamak veya herhangi bir kritere göre reddetmek için, düzenlendikten sonra mesajı ayarlamak için kullanılabilir. Sıfır olmayan bir değerle çıkarsa commiti iptal edebilir.

1 parametre alır: Önerilen mesajı tutan dosya.

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

# committeki yasakli kelimeleri sansürleyen ruby betiği
file_name  = ARGV[0]
content    = File.read(file_name)

forbidden_words = ["yasak1", "yasak2"]

forbidden_words.each { |f_word| content.gsub!(f_word, '*' * f_word.size) }

File.write(file_name, content)
```

post-commit

“git commit” komutu ile ilişkilendirilmiştir. Asıl commit yapıldıktan sonra çağrılır. Bu nedenle commiti bozamaz. Özellikle bildirimlere izin vermek için kullanılır.

Parametre almaz.

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

puts '---------------HATIRLATMA---------------'
puts "Yaptığınız commit'i push etmeyi unutmayın!"
puts '----------------------------------------'
```

pre-rebase

“git rebase” komutu ile ilişkilendirilmiştir. Bir dal(branch) rebase edilirken çağrılır. Temel olarak rebase durdurmak için kullanılır.

2 parametre alır: İlk parametre, dizinin çatallandığı ters yönlendirmedir. İkinci parametre, yeniden dallanmakta olan daldır ve geçerli dalı yeniden birleştirirken ayarlanmaz.

Örneğin, havuzunuzdaki rebasing'i tamamen reddetmek istiyorsanız, aşağıdaki rebase öncesi komut dosyasını kullanabilirsiniz:

```
#!/bin/sh
# Disallow all rebasing
echo "pre-rebase: Rebasing is dangerous. Don't do it."
exit 1
```

post-checkout

“git checkout” ve “git clone” komutları ile ilişkilendirilmiştir. Genel olarak koşulları doğrulamak, farklılıkları göstermek ve gerekirse ortamı yapılandırmak için kullanılır.

3 parametre alır: Önceki HEAD referansı, yeni HEAD referansı, branch checkout (1) veya file checkout (0) olduğunu gösteren bayrak

A common problem with Python developers occurs when generated .pyc files stick around after switching branches. The interpreter sometimes uses these .pyc instead of the .py source file. To avoid any confusion, you can delete all .pyc files every time you check out a new branch using the following post-checkout script:

```
#!/usr/bin/env python
import sys, os, re
from subprocess import check_output
# Collect the parameters
previous_head = sys.argv[1]
new_head = sys.argv[2]
is_branch_checkout = sys.argv[3]
if is_branch_checkout == "0":
print "post-checkout: This is a file checkout. Nothing to do."
sys.exit(0)
print "post-checkout: Deleting all '.pyc' files in working directory"
for root, dirs, files in os.walk('.'):
for filename in files:
ext = os.path.splitext(filename)[1]
if ext == '.pyc':
```

```
os.unlink(os.path.join(root, filename))
```

post-merge

“git merge” ya da “git pull” komutları ile ilişkilendirilmiştir. Merge’den(birleştirmeden) sonra çağrılır. Bu nedenle merge iptal edilemez. Git’in işleyemediği izinleri veya diğer veri türlerini kaydetmek veya uygulamak için kullanılabilir.

1 parametre alır: yapılan merge(birleştirme) işleminin squash birleştirme olup olmadığını belirten bir durum bayrağı

https://gist.github.com/nnja/5ce68f0312caf6607bba15dde2c28f02

```
#!/usr/bin/env python
import sys
import subprocess

diff_requirements = 'git diff ORIG_HEAD HEAD --exit-code -- requirements.txt'

exit_code = subprocess.call(diff_requirements.split())
if exit_code == 1:
    print 'The requirements file has changed! Remember to install new
dependencies.'
else:
    print 'No new dependencies.'
```

pre-push

“git push” komutu ile ilişkilendirilmiştir. Bir itmenin gerçekleşmesini önlemek için kullanılabilir.

2 parametre alır: Uzak hedefin adını ve konumunu sağlayan iki parametre ile çağrılır.

http://blog.ricardofilipe.com/post/git-hook-push-master

```
#!/bin/bash
protected_branch='master'
current_branch=$(git symbolic-ref HEAD | sed -e
's,.*/\(.*\),\1,')

if [ $protected_branch = $current_branch ]
       then
              read -p "You're about to push to master. What's
the meaning of life? > " -n 2 -r < /dev/tty
              echo
              #if echo $REPLY | grep -E '^[Yy]$' > /dev/null
              if echo $REPLY | grep -E '42' > /dev/null
              then
                     exit 0 # carry on
              fi
              exit 1 # exit and go read a book
else
```

```
    exit 0
fi
```

pre-receive

Uzak repodaki “git-receive-pack” komutu ile ilişkilendirilir. Uzak depodaki referansları güncellemeye başlamadan hemen önce pre-receive hook başlatılır. Çıkış durumu, güncellemenin başarısını veya başarısızlığını belirler.

Parametre almaz.

```
#!/bin/bash
# check each branch being pushed
while read old_sha new_sha refname
do
  if git diff "$old_sha" "$new_sha" | grep -qE '^\+.*\s+$'; then
    echo "HATA: Satır sonunda boşluk karakteri var!"
    git diff "$old_sha" "$new_sha" | grep -nE '^\+.*\s+$'
    exit 1
  fi
done
```

update

Uzak repodaki “git-receive-pack” komutu ile ilişkilendirilir. Uzak depodaki ref güncellemeden hemen önce, update hook çağrılır. Çıkış durumu, ref güncellemesinin başarısını veya başarısızlığını belirler.

3 tane parametre alır. Güncellenen ref adı, eski nesne adı, yeni nesne adı

```
#!/usr/bin/env python
import sys
branch = sys.argv[1]
old_commit = sys.argv[2]
new_commit = sys.argv[3]
print "Moving '%s' from %s to %s" % (branch, old_commit, new_commit)
# Abort pushing only this branch
# sys.exit(1)
```

The above update hook simply outputs the branch and the old/new commit hashes. When pushing more than one branch to the remote repository, you’ll see the print statement execute for each branch.

post-receive

Uzak repodaki “git-receive-pack” komutu ile ilişkilendirilir. Tüm referanslar güncellendikten sonra uzak depoda bir kez çalıştırılır. Güncellemelerden sonra çağrıldığından işlemi iptal edemez.

Parametre almaz, ancak stdin üzerinden "<old-value> <new-value> <ref-name>" şeklinde bilgi alır.

post-update

Uzak repodaki “git-receive-pack” komutu ile ilişkilendirilir. Tüm referanslar gönderildikten sonra yalnızca bir kez çalıştırılır. Bu bağlamda post-receive hook a benzer, ancak eski veya yeni değerleri almaz. Daha çok, itilen referanslar için bildirimleri uygulamak için kullanılır.

güncellenen ref adlarının her biri için değişken sayıda parametre alır.

pre-auto-gc

“git gc –auto” komutu ile ilişkilendirilir. Depoları otomatik olarak temizlemeden önce bazı kontrolleri yapmak için kullanılır.

Parametre almaz.

post-rewrite

“git commit --amend, git-rebase” komutları ile ilişkilendirilir. Bu, git komutları zaten işlenmiş verileri yeniden yazarken çağrılır. Parametrelere ek olarak, "<old-sha1> <new-sha1>" şeklinde stdin içindeki karakter dizileri alır.

1 parametre alır: İlişkilendirilen komutun adı (amend ya da rebase)