

فصل دوازدهم

غواصی به پروژه

مدیریت: شعب

فصل گذشته ، ما Issues را کشف کردیم و از آنها برای برنامه ریزی پروژه استفاده کردیم. ما همچنین یاد گرفتیم که چگونه تعهدات خود را به موضوعات پیوند دهیم تا بتوانیم هر تغییری را در پروژه خود دنبال کنیم. روش کار ما ساده بود: یک مسئله را انتخاب کنید ، تعهدی را ایجاد کنید که بتواند آن را برطرف کند و به GitHub سوق دهید. سپس مسئله حل و فصل شد. اما این روش کار در بیشتر پروژه های دنیای واقعی چندان سازگار نیست. پتانسیل پیچ خوردگی خیلی زیاد است.

اگر برای حل مسئله به بیش از یک تعهد نیاز دارید ، چه می کنید؟ اگر سایر اعضای تیم تعهدی را اعمال کنند که شامل تغییراتی در همان پرونده هایی است که شما روی آنها کار می کنید ، فشار می آورد؟ چگونه مطمئن شویم که متعهدین تحت فشار واقعاً مسئله را برطرف می کنند؟ همه اینها بخشی از دلایلی هستند که چرا ایجاد تغییرات مستقیم در پروژه توصیه نمی شود ، حتی اگر به تنهایی کار کنید.

همانطور که در فصل گذشته گفتیم ، بستن شماره توسط کلمات کلیدی در پیام متعهد بسیار جالب است ، اما باید با آن بسیار مراقب باشید. فقط شما کارهایتان را دیده اید ، و ممکن است این مسئله برطرف نشود. یا ممکن است اشکالات جدیدی را در این پروژه معرفی کند. به همین دلیل بهتر است شخص دیگری قبل از پذیرفتن تغییرات ، کد شما را بررسی کند. این بخشی است که ما در این فصل می خواهیم در مورد آن صحبت کنیم. ابتدا با رایج ترین جریان کاری GitHub (نحوه کار بیشتر تیم ها در GitHub) آشنا می شوید و سپس قصد داریم با مفهوم Branches آشنا شویم.

اما قبل از شروع این فصل ، این نکته کوچکی است که همیشه باید به خاطر بسپارید: "شما اشتباه می کنید. زمان زیادی بنابراین باید حتماً از حداکثر امنیت استفاده کنید. " بیا بریم!

گردش کار GitHub

در این بخش ، در مورد متداول ترین روشی که توسعه دهندگان از GitHub استفاده می کنند صحبت خواهیم کرد. به خاطر داشته باشید که هر تیم روش انجام کار خود را دارد ، اما هرکدام از این روش های کار با الهام از جریان کاری اساسی که می خواهیم ارائه دهیم.

واقعیت کوچک در مورد اشتباه بودن را به خاطر دارید؟ این احتمال همه جا از اشتباهات به همین دلیل است که شما باید این جریان کاری GitHub را دنبال کنید ، بنابراین حتی اگر خطایی رخ دهد ، بازتاب آن را به شیوه ای کنترل شده جدا می کنید. روش کار ما از فصل قبل این بود که همه کارها را مستقیماً به پروژه اصلی متعهد کنیم و این بسیار خطرناک است. پروژه اصلی

بیشتر اوقات خط تولید است ، نسخه ای که مشتریان می بینند و از آن استفاده می کنند. بنابراین ، این نسخه باید بسیار تمیز باشد و باید همیشه قابل بهره برداری باشد. اگر هر خطایی راهی نسخه اصلی شود ، مشتری ها اشکالات را تجربه می کنند و باعث اختلال در هر یک از اعضای تیم می شود.

یکی از راه های حل این مسئله ، ایجاد یک کپی از پروژه اصلی و کار بر روی این کپی است. هر تغییری که در این نسخه ایجاد کنید روی پروژه اصلی تأثیر نمی گذارد ، بنابراین هیچ یک از اشتباهات شما نمی تواند بر روی مشتری تأثیر بگذارد. و هنگامی که شما (و افراد دیگر) کاملاً مطمئن هستید که تغییرات ایجاد شده برای حل مسئله است ، می توانید آن تغییرات را در نسخه اصلی تولید کنید.

به آن نسخه های اصلی پروژه شاخه گفته می شود و مفهوم تولید مثل تغییر در شاخه دیگر ادغام نامیده می شود. شما می توانید شاخه های زیادی را که دوست دارید ایجاد کنید و می توانید بین آنها تجارت کنید. وقتی برای اولین بار مخزن ایجاد می کنید ، یک شعبه جدید برای شما ایجاد می کند. "استاد" نامیده می شود اکثر توسعه دهندگان نسخه اصلی یا تولیدی خود را به صورت کارشناسی ارشد قرار می دهند و فقط هنگامی که کاملاً مطمئن هستند که انجام این کار صحیح است ، تغییرات را در آنجا ایجاد می کنند.

درست مانند شاخه های درخت ، شاخه گیت می تواند دارای تغییرات زیادی باشد ، به این معنی که حتی می توانید شاخه های جدید دیگری از شاخه های دیگری غیر از استاد ایجاد کنید ، حتی حفظ چنین معماری دشوار است. بیشتر اوقات ، هنگام کار روی یک موضوع ، شاخه ای ایجاد می کنید و پس از برطرف شدن مشکل ، آن را حذف می کنید.

برای این که همه این موارد را به چشم بیاوریم ، می خواهیم در مورد گردش کار پیش فرض یا رایج GitHub اطلاعات کسب کنیم. همانطور که می دانید همه چیز باید از یک موضوع شروع شود. ما این فصل گذشته را قبلاً پوشانده ایم ، بنابراین شما با این موضوع آشنا هستید. بنابراین ، ما در مورد هر یک از مراحل بعدی گردش کار صحبت خواهیم کرد.

وقتی قصد دارید با ایجاد تغییر در کد ، یک مسئله را حل کنید ، ابتدا باید یک نسخه از نسخه فعلی پروژه را تهیه کنید: یک شاخه جدید ایجاد کنید.

سپس ، طبق معمول ، تغییرات خود را انجام داده و وضعیت پروژه را مرتب می شوید. شما می توانید هر تعداد تعهدات مورد نیاز خود را انجام دهید. این شعبه اصلی را تحت تأثیر قرار نمی دهد. همچنین می توانید تعهدات خود را به GitHub فشار دهید تا کد شما دیده شود.

سپس شاخه خود را به استاد اصلی پیوند می دهید ، تا دیگران بتوانند تغییرات را مقایسه کرده و کد شما را مرور کنند. به این پیوند PullRequest گفته می شود: شما درخواست می کنید که تعهدات خود را در مورد شاخه استاد اعمال کنید.

سپس اعضای دیگر تیم می توانند کد شما را مرور کنند و در مورد GitHub اظهار نظر کنند. شما تا زمانی که همه مشکلات برطرف نشود ، تعهدات بیشتری را برای پرداختن به آن نظرات وارد خواهید کرد

اگر هر یک از طرفین (توسعه دهندگان ، مدیران ، آزمایش کنندگان یا مشتری) موافقت کنند که تغییرات شما اشکالی ندارد و مسئله مورد نظر را حل می کند ، درخواست کشش پذیرفته می شود. این بدان معناست که هر تعهدی که در شعبه خود انجام داده اید در شعبه استاد اعمال می شود. سپس می توانید شاخه ای را که ایجاد کرده اید حذف کنید.

و همین شاید تعجب کنید که تفاوت آن با استفاده مستقیم از استاد چیست.

این بسیار متفاوت است زیرا اشتباهات و حذفیات قبل از اعمال تغییر در نسخه تولید گرفتار می شوند. این بدان معنی است که تعداد اشکالات تولید به حداقل می رسد. همچنین این امکان را برای اعضای مختلف تیم شما فراهم می کند که تغییرات را قبل از اعمال درخواست بررسی کنند ، این یک روش استاندارد برای کار در شرکت های دارای فناوری برتر است. بسته بندی تغییرات در یک درخواست کشش نیز مشکل چندین نفر را تحت فشار قرار می دهد که متعهد به حل مشکلات مختلف در همان زمان هستند. ورود به سیستم تاریخچه را تمیز نگه می دارد.

شما ممکن است وسوسه شوید که درخواستهای کشش را فقط وقتی احساس می کنید که با کار خود انجام داده اید باز کنید. مگر اینکه کارهایی که انجام داده اید بسیار کوچک و سر راست باشد ، صبر نکنید قبل از افتتاح روابط عمومی.

با کار PR در مراحل اولیه توسعه ، می توانید قبل از ایجاد تغییرات خیلی زیاد ، بازخورد دریافت کنید. این برای مبتدیان بسیار مفید است ، به خصوص به دلیل این که پیروی از مسیر اشتباه از ابتدا به اصلاح شما زمان زیادی می برد و آرزو می کنید که زودتر به شما صحیح گفته شده باشد. باز کردن درخواست کشیدن به معنای انجام کار نیست. این بدان معناست که شما در مورد اعمال تعهدات از یک شاخه به شاخه دیگر فکر می کنید.

برای خلاصه کردن همه این مراحل ، می توانید یک شکل کمی در شکل 1-12 پیدا کنید.

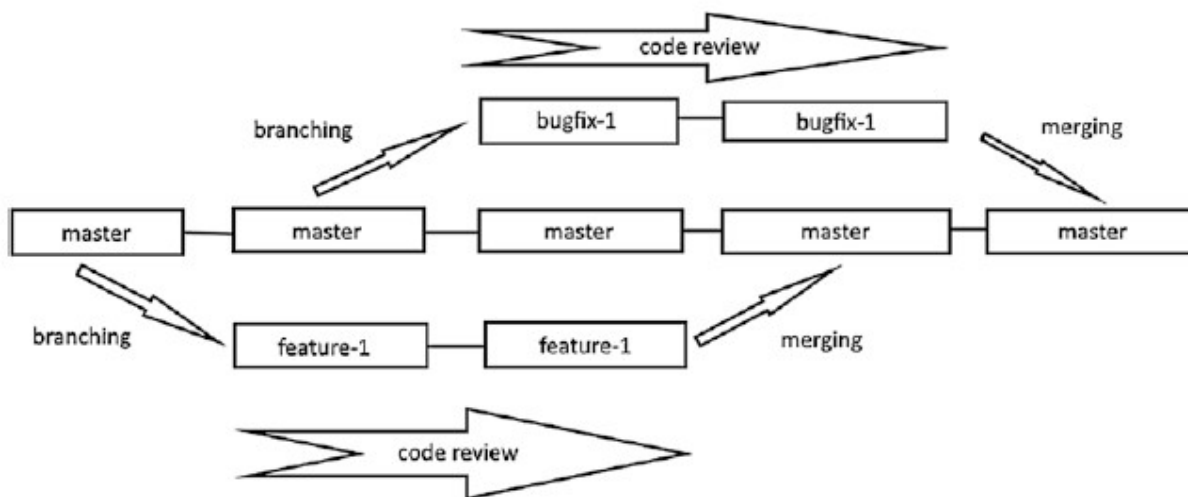


Figure 12-1. Basic Git workflow

همانطور که می بینید ، می توانیم از هر شعبه ای در پروژه خود شعبه ایجاد کنیم. گیت در ابتدای مخزن شعبه ای به نام استاد برای ما ایجاد کرد. سپس می توانیم شاخه های بیشتری ایجاد کنیم (مثلاً یک شاخه اشکال یا یک شاخه ویژگی) برای ایجاد تغییر در شاخه اصلی.

Branches

همانطور که قبلاً گفتیم ، شعب اصلی ترین ویژگی بررسی کد هستند. شما باید قبل از انتشار کار خود روی شعبه خود کار کنید ، تا از تغییرات دیگر افراد ناراحت نشوند. به عبارت ساده ، یک شعبه فقط یک نسخه مستقل از پروژه شما در یک زمان خاص است. بیا ببینیم که چطور کار می کنند و اجازه دهید برخی از آنها را ایجاد و حذف کنیم.

منطق پشت شاخه ها ساده است: وضعیت فعلی پروژه را بگیرید و یک نسخه از آن تهیه کنید. در این نسخه می توانید تغییرات خود را بدون آسیب رساندن به افراد دیگر انجام دهید. شما می توانید از شعب استفاده کنید تا کانال های متمایز توزیع داشته باشید یا فقط چیزهای جدیدی را با این پروژه امتحان کنید.

هنگام ایجاد مخزن ، به طور پیش فرض شعبه می گیرید: master. هنگام کار روی پروژه های بسیار کوچک ، این شاخه کافی است؛ اما بیشتر پروژه ها برای به دست آوردن بهترین نتیجه به شعب بیشتری احتیاج دارند. اول ، آنها به یک شاخه تولید نیاز دارند ، که در آن مشتریان بتوانند آخرین نسخه پایدار نرم افزار را بدست آورند. این شاخه استاد است. شاخه تولید فقط زمانی به روز می شود که پروژه مطمئن باشد پایدار است زیرا این شاخه انتشار است. سپس ، شاخه توسعه وجود دارد ، که در آن همه پیشرفت ثبت شده و تمام تعهدات مورد آزمایش قرار می گیرد.

شما بیشتر در شاخه توسعه کار خواهید کرد زیرا جایی که بیشتر سرگرم کننده است. سرانجام ، شاخه های پچ کوتاه وجود دارد که شما برای ایجاد تعهدات خود قبل از ادغام آنها در شعبه توسعه ایجاد می کنید. آن شاخه های لکه دار با درخواست کشش زندگی می کنند و می میرند. وقتی مشکلی را حل می کنید و بعد از آن حذف می شوید ، یکی را ایجاد می کنید.

برای خلاصه کردن کمی ، (بیشتر اوقات) سه نوع شاخه خواهید داشت:

- شعبه تولید ، جایی که نسخه های پایدار پروژه خود را منتشر می کنید
- شعبه توسعه ، جایی که آخرین نسخه خود را آزمایش خواهید کرد
- پچ کردن شعبه ، جایی که در مورد مسائل خود کار خواهید کرد

در صورت وجود مشکل فوری بسیار فوری که نیاز به حل سریع دارد ، هرگز به طور مستقیم به تولید یا شاخه توسعه متعهد نخواهید شد. برای به روزرسانی آن شعب ، از درخواستهای کشش استفاده می کنید تا تغییرات بررسی و آزمایش شوند. برخی از شرکت ها وجود دارند که هر توسعه دهنده فقط مستقیماً به شعبه توسعه متعهد می شود ، اما این کار بسیار ضد کارایی است زیرا اگر یک اشکال کشف شود ، آنها نمی دانند کدام تعهد را معرفی کرده است. همچنین ، توسعه دهنده را وادار می کند تا تعهدات "یک-آن-همه-آن" را فشار دهد ، که یک الگوی است. انجام همه تعهدات ، تعهدی است که سعی می کند در همان زمان ، هرگونه مسأله را برطرف کند ، برای مثال ، تعهدی که اشکال را برطرف کرده و همزمان ویژگی جدیدی را معرفی کند. این عمل اغلب به دلیل تنبلی توسعه دهندگان ایجاد می شود زیرا آنها نمی خواهند شعبه جدیدی را برای موضوع دیگر ایجاد کنند. این باعث می شود درخواست های بسیار بد جابجایی ایجاد شود و ردیابی پیشرفت پروژه را دشوار کند. همچنین این یک چالش بزرگ را برای آزمایش کنندگان ایجاد می کند زیرا نمی دانند نسخه پایدار کدام نسخه است. این یک ایده همه جانبه است؛ حتی با پروژه های کوچک خود این کار را نکنید. ممکن است ایجاد و حذف شاخه ها در همه زمانها خسته کننده به نظر برسد ، اما بهترین کار در هنگام کار با آن است.

نکته ای که باید در مورد شاخه های گیت به خاطر بسپارید این است که آنها فقط اشاراتی به تعهدات هستند. به همین دلیل ایجاد و حذف آنها بسیار سریع است. به یاد داشته باشید وقتی درمورد اینکه چگونه Git تعهدات خود را در لینک های زنجیر شده ذخیره می کند ، به یاد دارید؟ خوب ، یک شعبه فقط مرجع یکی از آن تعهدات است. تعهد حاوی اطلاعاتی درباره نویسنده ، تاریخ ، عکس فوری و از همه مهمتر نام تعهد قبلی است. نام تعهد قبلی والدین خوانده می شود و هر تعهدی جز اولین آن حداقل مشهود است. بنابراین ، هر تعهد با مورد قبلی مرتبط است تا بتوانیم تاریخ تغییر پروژه را از نو بازآفرینی کنیم.

در حال حاضر ، شما فقط شاخه پیش فرض به نام master دارید و آخرین تعهد پروژه خود را ارجاع می دهید. برای ایجاد یک تعهد جدید ، Git مرجع را بررسی می کند و اطلاعاتی را که متعهد به ایجاد پیوند بین تعهد جدید و قبلاً ارجاع شده است

استفاده می کند. بنابراین ، هر بار که مرتکب شوید ، مرجع به سمت تعهد جدید حرکت می کند و چرخه ادامه می یابد. بنابراین ، یک شاخه فقط اشاره ای به تعهدی است که به عنوان والدین شخص بعدی طراحی شده است.

اما چگونه Git می داند که در کدام شاخه هستیم؟ خوب ، از مرجع دیگری به نام HEAD استفاده می کند که تعهد فعلی را ارجاع می دهد. اگر در شعبه هستید ، HEAD آخرین تعهد آن شعبه را ارجاع می دهد. اما اگر نسخه قبلی را بررسی می کنید (مانند زمانی که از "git checkout <commit_name>" استفاده می کردیم) ، سرارجاعات مربوط به HEAD را مرتکب می شوید و شما در وضعیتی به نام "HEAD جدا شده" قرار دارید.

در بیشتر مواقع ، می توانید از HEAD به عنوان مرجع شعبه فعلی استفاده کنید و هر تعهدی که ایجاد می کنید از آخرین متعهد در آن شعبه به عنوان والدین استفاده می کند.

وقتی شاخه ای را به شاخه دیگری می پیوندید ، تعهد جدیدی ایجاد می شود که دارای دو والد است: یکی از والدین از هر شعبه ، بنابراین می توانید نوع commit را با توجه به تعداد والدین خود تشخیص دهید:

- بدون پدر و مادر: اولین تعهد
- یکی از والدین: در یک شعبه مرتکب عادی شوید
- والدین متعدد: تعهدی که با ادغام شعب ایجاد شده است

ایجاد شعبه

اکنون که درباره شاخه ها چیزهای زیادی می دانید ، بیایید یکی ایجاد کنیم! این بسیار آسان است؛ شما فقط باید از دستور "شاخه git" استفاده کنید که به دنبال آن نام شعبه است. بخاطر داشته باشید که نام شعبه فقط باید حاوی مقادیر الفبایی و خط و خط یا زیر باشد. هیچ فضایی مجاز نیست.

\$ git branch <name>

به عنوان مثال ، بیایید یک شعبه توسعه برای پروژه خود ایجاد کنیم بیایید آن را "devlop" بگذاریم. در اینجا نحوه انجام این کار آورده شده است:

\$ git branch develop

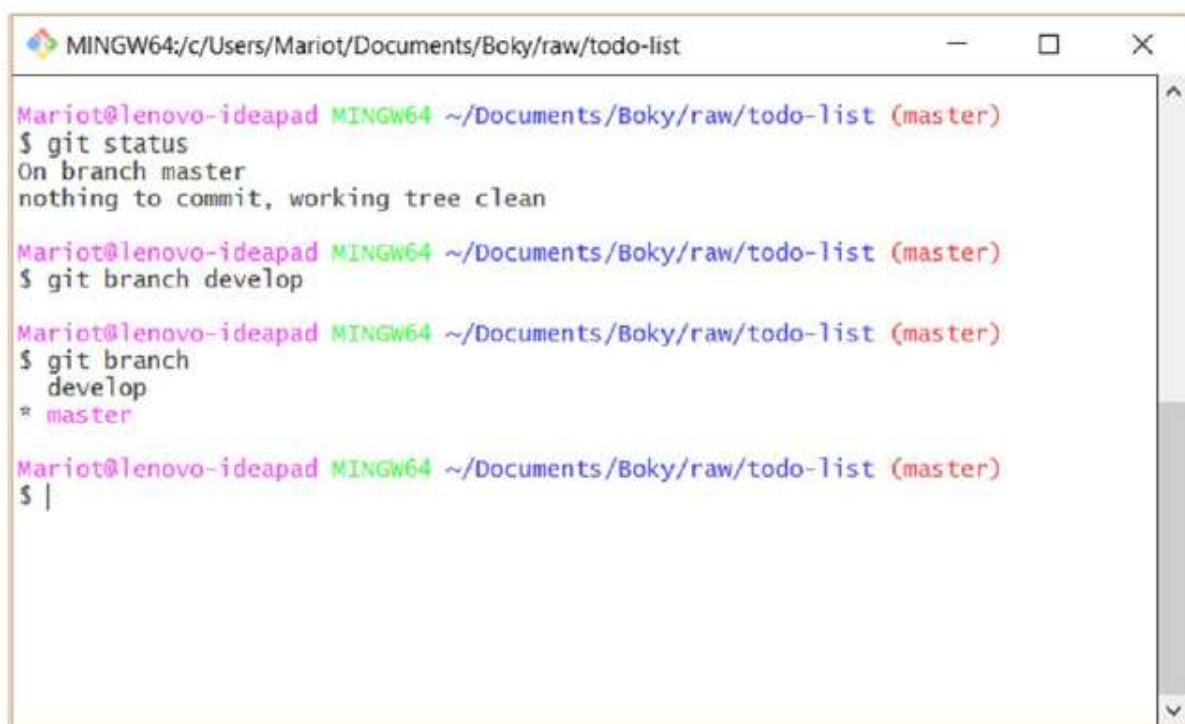
پس از اجرای آن فرمان ، متوجه خواهید شد که هیچ چیزی در پروژه شما تغییر نکرده است. دلیل این امر این است که ایجاد شعبه فقط به معنای ایجاد ارجاع به آخرین تعهد شعبه فعلی و چیز دیگری نیست. برای شروع کار با یک شاخه ، باید به آن بروید.

انتقال به شاخه دیگری

ما شاخه توسعه خود را ایجاد کردیم و اکنون زمان آن رسیده که به آن برویم. اما مشکل اینجاست: نامی را که به شعبه دادیم فراموش کردیم. حال ممکن است شخصی به ما پیشنهاد دهد که می توانیم به عقب برگردیم و به بخش قبلی نگاه کنیم تا نام را جستجو کنیم. اما من ایده بهتری دارم: لیست تمام شعب فعلی ما. برای این کار کفایت دستور git branch را بدون هیچ پارامتری اجرا کنید.

\$ git branch

این دستور به شما شاخه هایی را می دهد که در حال حاضر در اختیار دارید و یک ستاره کوچک در کنار آن قرار دارد که در حال حاضر در آن هستید (HEAD). شکل 12-2 را برای نمونه ای از لیست شعبات بررسی کنید.



```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git status
On branch master
nothing to commit, working tree clean

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git branch develop

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git branch
  develop
* master

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ |
```

Figure 12-2. List of branches in our project

شما متوجه خواهید شد که ما هنوز در شاخه کارشناسی ارشد هستیم زیرا چیزی جز ایجاد شعبه نکرده ایم. حالا بیایید به آن تغییر دهیم.

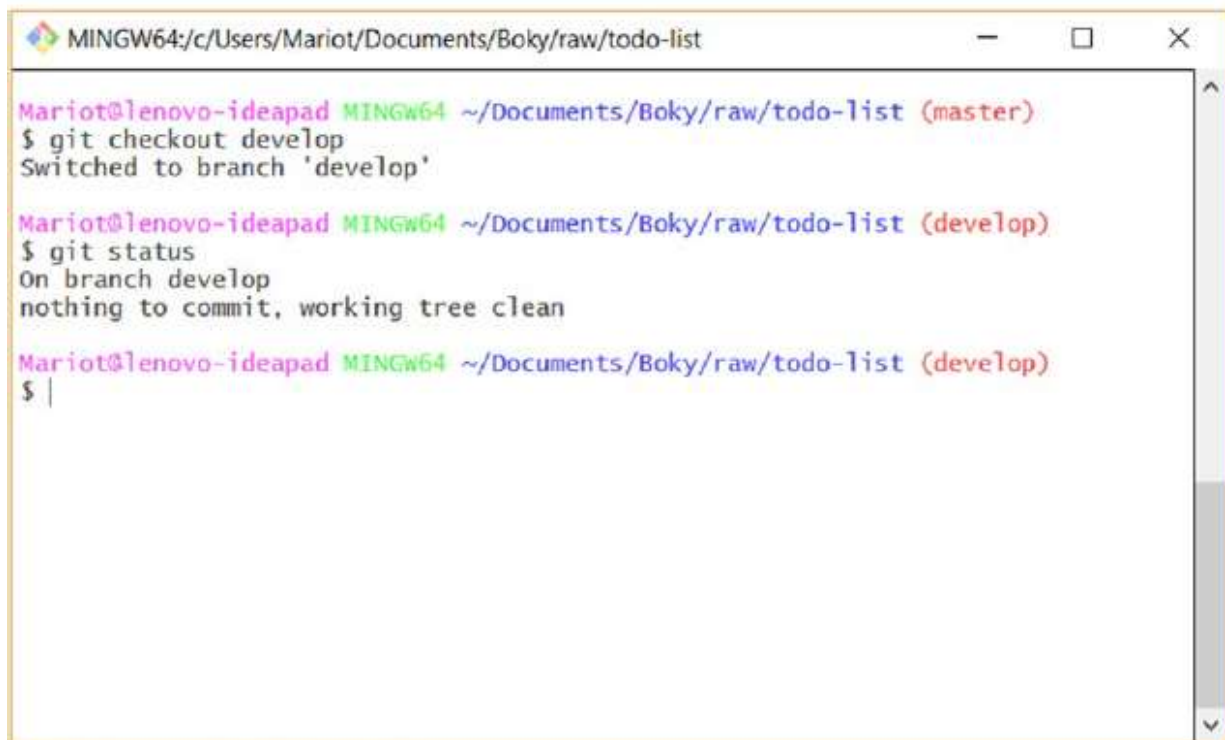
شما قبلاً دستور جابجایی بین نسخه ها را می دانید. خوب ، ما از همان دستور برای حرکت بین شاخه ها استفاده خواهیم کرد. به سادگی از "پرداخت git" با نام شعبه به عنوان پارامتر استفاده کنید.

\$ git checkout <name>

بنابراین ، اگر بخواهیم به شاخه توسعه تغییر دهیم ، باید مجبور شویم اجرا کنیم:

\$ git checkout develop

پس از بررسی شعبه جدید ، یک پیام تأیید از Git دریافت خواهید کرد و همچنین می توانید نتیجه وضعیت git را بررسی کنید تا مطمئن شوید. شکل 12-3 نتیجه آن دستورات را نشان می دهد.



```
MINGW64:/c:/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git checkout develop
Switched to branch 'develop'

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git status
On branch develop
nothing to commit, working tree clean

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ |
```

Figure 12-3. Switching branches

حذف یک شعبه

شما از ایجاد شاخه آزمایش لذت می برید؟ خوب زمان آن است که آن را حذف کنیم زیرا ما در حال حاضر یک شعبه آزمایش داریم: توسعه دهید. این جایی است که ما شاخه های وصله خود را ادغام خواهیم کرد و تمام آزمایشات در آنجا انجام خواهد شد. می توانید با بررسی "حذف شاخه بعد از ادغام روابط عمومی" هنگام ایجاد درخواست Pull ، یک شاخه تحت فشار را حذف کنید ، به معنی شاخه ای که در مخزن از راه دور وجود دارد.

با این کار شاخه از راه دور پاک می شود اما شاخه های محلی شما بدون تغییر خواهند بود. شما باید شاخه های محلی خود را به صورت دستی پاک کنید.

برای حذف یک شاخه ، فقط از همان دستور استفاده کنید تا یکی را ایجاد کنید اما با گزینه "d-" استفاده کنید.

\$ git branch -d <name>

بنابراین ، برای حذف شاخه آزمایش خود ، از آن استفاده خواهیم کرد

\$ git branch -d testing

درست مانند یک شاخه درخت واقعی ، شما شاخه Git را که اکنون در آن ایستاده اید ، قطع نمی کنید. قبل از حذف شاخه ، شعبه دیگری را بررسی کنید. و به همین دلیل ، شما نمی توانید کمتر از یک شعبه در یک پروژه داشته باشید. اگر به هر حال امتحان کنید ، خطایی مانند مورد نشان داده شده در شکل 4-12 دریافت خواهید کرد.



```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git checkout master
Switched to branch 'master'

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git branch testing

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git checkout testing
Switched to branch 'testing'

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (testing)
$ git branch -d testing
error: Cannot delete branch 'testing' checked out at 'C:/Users/Mariot/Documents/Boky/raw/todo-list'

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (testing)
$ |
```

Figure 12-4. *Deleting current branch*

بنابراین ، شما باید master را بررسی کنید یا قبل از حذف شاخه آزمایش ، شعبه ای را توسعه دهید. اگر این کار را به درستی انجام داده اید ، باید نتیجه ای مانند معدن دریافت کنید ، همانطور که در شکل 12-5 نشان داده شده است.



```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (testing)
$ git checkout master
Switched to branch 'master'

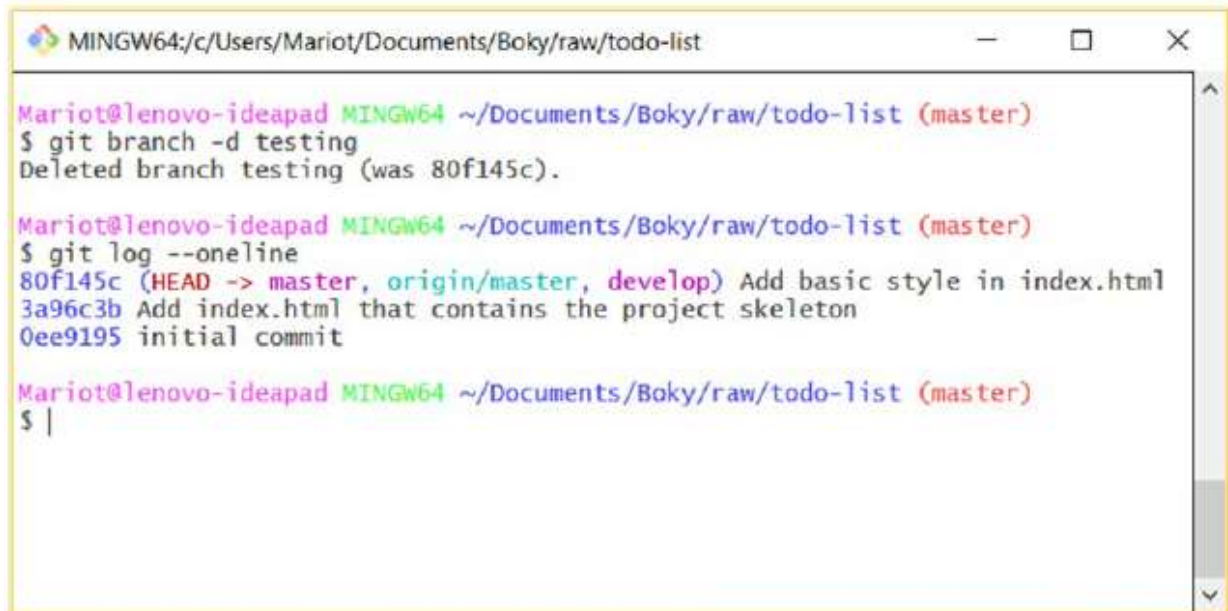
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git branch -d testing
Deleted branch testing (was 80f145c).

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ |
```

Figure 12-5. *Deleting of a branch (we hardly knew ye)*

به پیام تأیید توجه داشته باشید ، آن را به شما می دهد نام SHA-1 شعبه شما فقط حذف شده است. از آنجا که شاخه ای که ایجاد کردیم و حذف کردیم هیچ تعهدی نداشت ، فقط به آخرین تعهد شعبه فعلی اشاره داشت. برای تأیید این موضوع ، باید سابقه تاریخ را بررسی کنیم.

دستور لیست git log را اجرا کنید تا لیستی از جدیدترین تعهدات ، دقیقاً مانند شکل 12-6 دریافت کنید.



```
MINGW64:/c:/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git branch -d testing
Deleted branch testing (was 80f145c).

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git log --oneline
80f145c (HEAD -> master, origin/master, develop) Add basic style in index.html
3a96c3b Add index.html that contains the project skeleton
0ee9195 initial commit

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ |
```

Figure 12-6. Commit name check

خواهید دید که آخرین نام متعهد و نام شعبه یکسان است؛ به این دلیل است که ما هیچ کاری در شعبه خود نکرده ایم. همچنین می توانید روی سابقه تاریخی که شاخه ها از کجا منشا می گیرند ، مشاهده کنید. در این مثال ، شاخه توسعه از تعهد f145c80 سرچشمه می گیرد. این والدین شعبه است

شاخه های ادغام

ما در این فصل در مورد ادغام شعب زیادی صحبت کردیم اما یک ادغام واحد را انجام نداده ایم. بیایید آن را تغییر دهیم. بیایید تصور کنیم که می خواهید با افزودن چند قطعه اطلاعات ، پرونده README پروژه را بهبود بخشید. این کار قبلاً در موضوعات GitHub ما ذکر شده است ، بنابراین مشکلی در این زمینه وجود ندارد.

مرحله بعدی ایجاد یک شاخه جدید از شاخه توسعه است تا بتوانیم بعداً آنها را ادغام کنیم. شما باید شعبه جدیدی را از شعبه توسعه ایجاد کنید نه اینکه موضوع ایجاد شود زیرا ما تا زمانی که همه چیز به درستی آزمایش نشود ، ما به شاخه استاد لمس نخواهیم کرد. اگر همه چیز روشن و تمیز باشد ، شاخه توسعه را در شاخه استاد ادغام خواهیم کرد.

پس روشن است ، بگذارید شعبه جدیدی را که در آن کار خواهیم کرد ایجاد کنیم. بیایید آن را "improve-readme-description" بگذاریم. فراموش نکنید که قبل از ایجاد شعبه جدید از آن ، شاخه توسعه یافته را بررسی کنید.

```
$ git checkout develop
$ git branch improve-readme-description
```

اکنون که شعبه ایجاد شده است ، به آن سوئیچ کنید تا بتوانیم شروع به کار کنیم. برای تغییر به شعبه جدید ، فقط از دستور پرداخت استفاده کنید.

```
$ git checkout improve-readme-description
```

کامل! اکنون شعبه ای به نام "بهبود-خواندن-توصیف" داریم که از شاخه توسعه سرچشمه می گیرد. ما آنقدر شاخه ها را دوست داریم که یک شاخه از یک شعبه ایجاد کردیم!

حالا بیاییم کار کنیم پرونده README.MD را باز کرده و محتوای آن را به آن تغییر دهید.

```
# TODO list
```

```
A simple app to manage your daily task:
```

```
It uses HTML5 and CSS3.
```

```
## Features
```

```
* List of daily tasks
```

اکنون پرونده را مرحله بندی کرده و آماده انجام آن شوید. من به شما امکان می دهم پیام متعهد را انتخاب کنید ، اما فراموش نکنید که برای مسئله ای که می خواهید حل کنید ارجاع دهید! مراحل بعدی بدین ترتیب است

```
$ git add README.md
```

```
$ git commit
```

هیچ چیز جدیدی در اینجا نیست به عنوان هر دستور مانند هر شاخه است. تنها تغییر جزئی این است که نام شعبه در توضیحات مربوط به تعهد متفاوت است. شما می توانید آن را در نتیجه من نشان داده شده در شکل 7-12 مشاهده کنید.

همانطور که در شکل مشاهده می کنید ، HEAD اکنون به آخرین تعهد شعبه جدید ما اشاره دارد. این بدان معناست که هر تعهدی که ایجاد خواهیم کرد آن را به عنوان والدین خواهیم داشت. همچنین متوجه خواهید شد که استاد و توسعه شاخه تغییر نکرده اند. به این دلیل است که ما فقط در شعبه تازه ایجاد شده کار می کردیم.

اکنون که از رفع مشکل خود راضی هستیم ، اجازه دهید شاخه را به شاخه توسعه ادغام کنیم تا بتوانیم آن را آزمایش کنیم. برای ادغام شاخه خود در توسعه ، ابتدا باید بررسی کنیم. بنابراین ، با استفاده از دستور `git checkout checkout` حرکت کنید.

```
$ git checkout develop
```

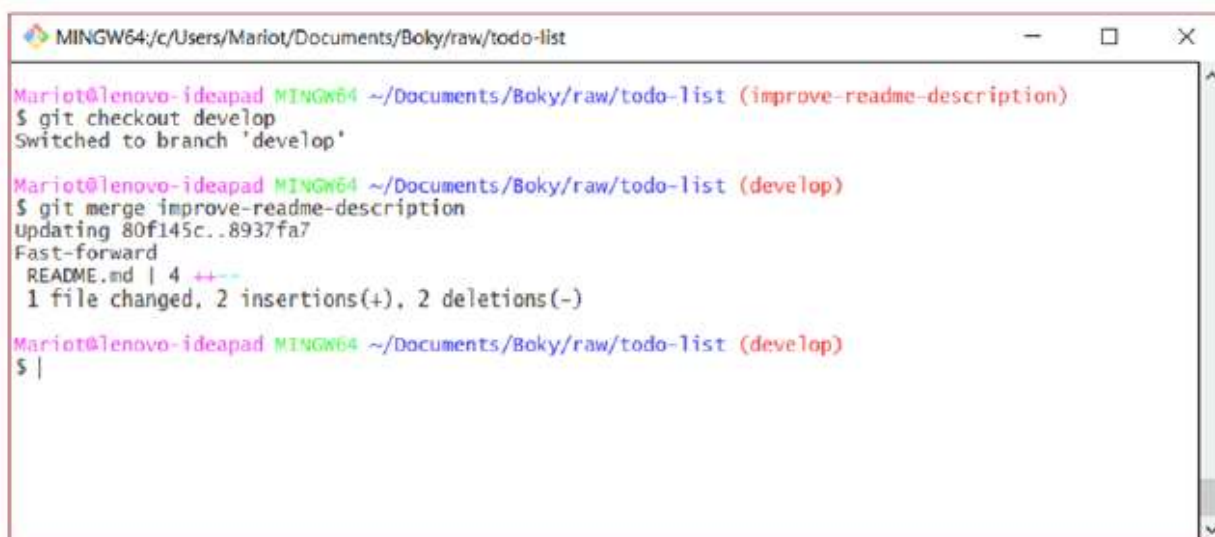
اکنون سعی خواهیم کرد که شاخه را در بخش توسعه یافته ادغام کنیم. ادغام فقط به معنای بازتولید کلیه تعهدات در یک شاخه در شاخه دیگر است. بنابراین ، ما از دستور `git merge` استفاده می کنیم و به دنبال آن نام شاخه برای ادغام استفاده می شود.

```
$ git merge <name>
```

از آنجا که ما به دنبال ادغام "improve-readme-description" در "توسعه" هستیم ، دستور ما برای اجرای در شاخه توسعه است.

```
$ git merge improve-readme-description
```

این دستور تعهدات شما را از "improve-readme-description" به "develop." بازآفرینی می کند. بنابراین ، شما به عنوان تأیید تعهد نتیجه مشابهی خواهید گرفت. شکل 9-12 را برای مثال بررسی کنید.



```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (improve-readme-description)
$ git checkout develop
Switched to branch 'develop'

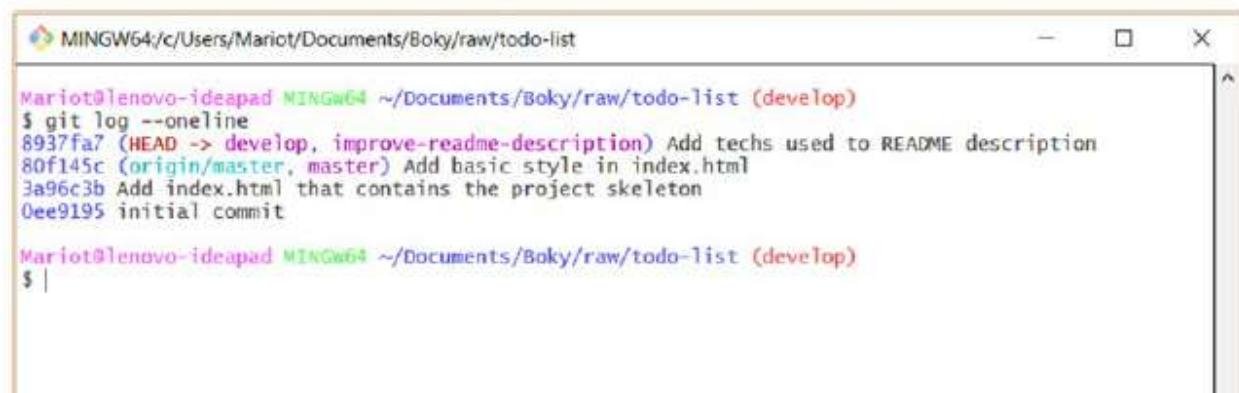
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git merge improve-readme-description
Updating 80f145c..8937fa7
Fast-forward
 README.md | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ |
```

Figure 12-9. Merge result

بباید بررسی دقیق اطلاعات مربوط به آنچه اتفاق افتاده است را بررسی کنیم. بعد از اجرای "`git log --online`" که در شکل 10-12 نشان داده شده است ، نتیجه مشابهی را با معدن خود دریافت خواهید کرد.

همانطور که می بینید ، HEAD اکنون به پیشرفت اشاره دارد زیرا این شاخه بررسی شده است. همچنین می توانید توجه داشته باشید که توسعه و بهبود-توصیف-توصیف اکنون به همان تعهد اشاره دارد. به دلیل ادغام است



```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git log --oneline
8937fa7 (HEAD -> develop, improve-readme-description) Add techs used to README description
80f145c (origin/master, master) Add basic style in index.html
3a96c3b Add index.html that contains the project skeleton
0ee9195 initial commit
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ |
```

Figure 12-10. History log after merge

اولین ادغام خود را تبریک می گویم! دفعه دیگر خیلی آسان نخواهد بود (اشاره: ادغام اختلافات ، هنگامی که همان خط کد در تعهدات مختلف تغییر یافته باشد ، ظاهر می شوند)

Pushing a branch to remote

شعب نه تنها برای کار در محل ساخته می شوند ، بلکه می توانید با فشار دادن آنها به مخزن از راه دور ، آنها را به جهانیان نیز منتشر کنید. به عنوان مثال ، اجازه دهید شاخه توسعه ما را به سمت GitHub سوق دهیم تا همه بتوانند پیشرفت ما را ببینند.

دستور سوق دادن شاخه به ریموت ، درست همان چیزی است که در یک فصل قبلی آموخته ایم ، فشار فشرده است. دستور است

```
$ git push <remote_name> <branch_name>
```

نام از راه دور تغییر نکرده است. هنوز "origin" است این شعبه است که این بار متفاوت است. به جای master ، ما می خواهیم شاخه توسعه را تحت فشار قرار دهیم. بنابراین ، دستور خواهد بود

```
$ git push origin develop
```

از آنجا که قبلاً به راه دور سوق داده اید ، نتیجه نشان داده شده در شکل 11-12 برای شما آشناست.

```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git push origin develop
Username for 'https://github.com': mtsitoara
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 412 bytes | 206.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/mtsitoara/todo-list/pull/new/develop
remote:
To https://github.com/mtsitoara/todo-list.git
 * [new branch]      develop -> develop

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ |
```

Figure 12-11. Pushing to a remote branch

اگر برای بررسی صفحه پروژه خود به GitHub برگردید، در مورد ایجاد درخواست های کشش نیز دکمه تماس با عمل را خواهید داشت. اکنون آنها را نادیده بگیرید و به جای آن بین شاخه استاد خود حرکت کنید و یکی را توسعه دهید. بعد از تحت فشار قرار دادن شاخه جدید می توانید شکل 12-12 را برای نمونه ای از صفحه پروژه بررسی کنید.

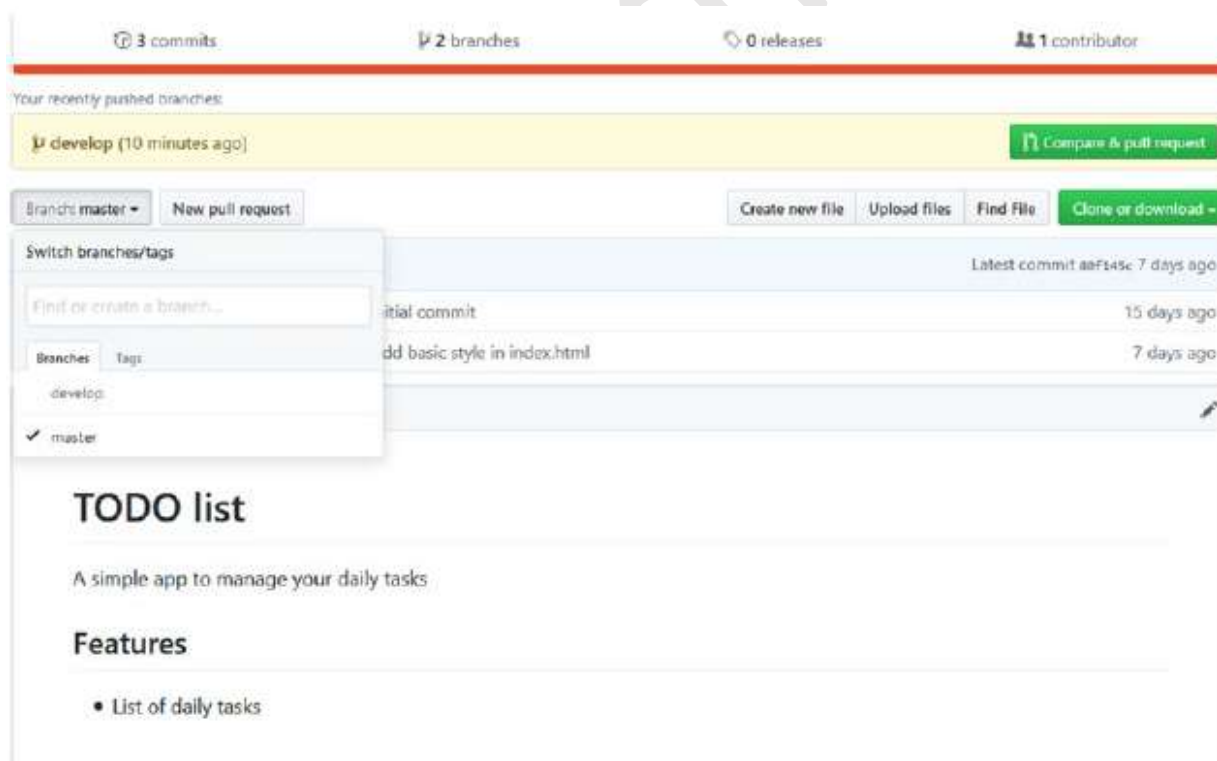


Figure 12-12. Our new project page

الان همه چیز مربوط به شعب است. اکنون می دانید چگونه آنها را ایجاد ، ادغام و حذف کنید. و از همه مهمتر ، شما دانش بنیادی در مورد گردش کار GitHub دارید: یک شاخه ایجاد کنید ، روی آن شاخه کار کنید و یک درخواست کشش ایجاد کنید.

اکنون ، ممکن است از خود پرسید: "اما آیا به ما قول نمی دهید بررسی کد کنید و درخواستها را جلب کنید؟ آیا ما حتی از گردش کار استفاده کردیم؟" کاملاً حق با شماست. ما از گردش کار استفاده نمی کنیم زیرا ما از روش مستقیم استفاده کردیم: مستقیماً با شاخه ها پیام می زنید.

در یک پروژه در دنیای واقعی ، شما متعهد نخواهید شد و مانند گذشته کار ما را مستقیماً به استاد یا شاخه توسعه فشار می آورد. در عوض ، از Pull Request برای ادغام شاخه ها با یکدیگر استفاده خواهید کرد. به این ترتیب ، کار شما می تواند قبل از اینکه بتوانید آنها را در شاخه توسعه یا کارشناسی ارشد ادغام کنید ، توسط همکارانتان بررسی شود.

خلاصه

این فصل به آنچه که Git را به ابزاری قدرتمند برای مدیریت پروژه تبدیل می کند ، پرداخته شده است:

شاخه ها. شعبه ها در یک پیشرفت سریع ضروری هستند زیرا احتمالاً همزمان در بسیاری از موضوعات کار خواهید کرد. نگه داشتن همه آن تغییرات در همان مکان ، تلاشی برای فاجعه است. به عنوان مثال ، شما باید در یک محیط تمیز برای رفع اشکال یا معرفی یک ویژگی شروع کنید. تلاش برای انجام هر دو در همان زمان خطر ابتلا به اشکالات بیشتر را به طور جدی افزایش می دهد.

راه اصلی اصلی این فصل ، اهمیت استفاده از یک گردش کار در هنگام توسعه با Git است. و آن دسته از کارها همه از شاخه ها استفاده می کنند تا انواع مختلف کار لازم برای حل مسئله پاک را جدا کنند.

ما نحوه ایجاد ، بررسی و حذف شاخه ای را مشاهده کرده ایم. حال بیا بید در مورد Pull Requests و Code Code بیشتر بدانیم ، بنابراین می توانیم در شاخه استاد خود تغییراتی را پیشنهاد کنیم!

فصل سیزدهم

مدیریت بهتر پروژه:

Pull Requests

در آخرین فصل ، ما درباره گردش کار معمولی GitHub می آموزیم. اکثر شرکت ها برای کار روزانه خود از تنوع این گردش کار استفاده می کنند. ما همچنین در مورد شاخه ها و نحوه استفاده از آنها چیزهای زیادی آموختیم. اما یک چیز وجود دارد که ما فرصتی برای بررسی آن نداریم: چگونه این دو مفهوم را با هم ترکیب کنیم. پاسخ ساده است: درخواستها و بررسیهای کد را بکشید.

در فصل قبل دلایل زیادی وجود داشت که چرا استفاده از رویکرد سنتی برای مدیریت کد (همه متعهد به همان شاخه هستند) یک ایده بد است. اما از آنجا که ما در پروژه فعلی خود به تنهایی کار می کنیم ، هنوز این ناراحتی را نمی بینیم. اما آنها در اینجا هستند و زمان زیادی برای حل و فصل می گیرند. بنابراین ، به من اعتماد کنید ، بهتر است جریان کار را دنبال کنید.

در این فصل نحوه اجرای روند کاری که در فصل قبل به ما ارائه شده است به شما نشان خواهیم داد که ما از شعب تازه ایجاد شده برای معرفی تغییرات در شعب قدیمی تر استفاده خواهیم کرد ، همچنین در مورد بررسی کد و نحوه مدیریت آنها می آموزید.

چرا از درخواستهای Pull استفاده می کنیم؟

بسیاری از توسعه دهندگان که یک جریان کاری خاص را دنبال نمی کنند می گویند که این اتلاف وقت است زیرا زمان توسعه ارزشمند را از شما می گیرد. یک حقیقت در آن بیانیه وجود دارد زیرا پیروی از گردش کار به معنای انتظار برای بررسی سایر افراد برای کد شماست. اما باید در نظر داشته باشید که در حالی که منتظر بررسی هستید ، مجبور نیستید کاری انجام دهید ، زیرا می توانید مستقیماً ادامه دهید و مسئله دیگری را حل کنید! به همین دلیل است که شعب در سیستم های VersionControl بسیار قدرتمند هستند. می توانید همزمان روی چندین موضوع کار کنید.

با گردش کار می توانید شروع به کار بر روی یک موضوع کنید ، برخی ایده ها یا راهنمایی های لازم را از همسالان خود بخواهید و سپس در انتظار پاسخ ها روی یک موضوع دیگر کار کنید. پس از دریافت بازخورد لازم ، می توانید به کار بر روی شماره اول ادامه دهید و این روند را تا زمانی که همه مسائل برطرف نشود تکرار کنید. همچنین استفاده از گردش کار به شما امکان می دهد کار را روی یک مسئله شروع کنید حتی اگر هنوز اطلاعاتی در مورد کارهایی که باید انجام دهید کامل نیست. می توانید روی یک مسئله کار کنید و اطلاعات بیشتری را در اواسط آن متوقف کنید. و نکته آخر: با خواندن مجدد شخص دیگری کد شما بهترین راه برای کاهش اشکالات است. زمانی که با تعقیب اشکالات در همه جا بدست می آورید بیشتر از زمانی است که با تعهد مستقیم به استاد کسب می کنید.

گردش کاری GitHub همچنین روشی ارجح برای همکاری همکاران OpenSource است.

این بسیار زشت خواهد بود اگر کسی بتواند تعهدات خود را مستقیماً به شعبه و بدون بررسی بررسی کند. در عوض ، هر مشارکت کننده یک کلون کار پروژه دارد و می تواند تغییراتی را پیشنهاد دهد که سایر همکاران در این زمینه بررسی و گفتگو کنند.

بنابراین ، در پایان ، کار با استفاده از گردش کار GitHub بهترین روش کار است و استفاده از آن باعث رفع اشکالات شما می شود. و همانطور که در فصل گذشته دیدیم ، استفاده از شاخه ها تنها اولین قدم است ، بنابراین برای تکمیل گردش کار باید از Pull Requests استفاده کنید.

ببایید در مورد آنها بیشتر بدانیم!

بررسی اجمالی در مورد Pull Requests

درخواست های Pull ، همان اندازه که مفید هستند ، یک مفهوم نسبتاً آسان برای درک هستند. ارسال درخواست Pull یا PR فقط درخواست مجوز برای اعمال کلیه تعهدات موجود در یک شعبه در شعبه دیگر است. اما خیلی سریع حرکت می کنیم قبل از یادگیری در مورد درخواستهای Pull ، باید یاد بگیریم که "pull" چیست.

pull

در اصطلاحات Git ، کشش درست برعکس فشار است (اگر حدس می زنید که به آن پنج امتیاز دهید). فشار شعبه شما را می گیرد و تمام تعهدات خود را در یک شعبه از راه دور کپی می کند و اگر هنوز در سرور وجود نداشته باشد این شعبه را ایجاد می کند. کشیدن فقط همین است ، اما به عقب: به شاخه ای از راه دور نگاه می کند و تعهدات موجود در آن را به مخزن محلی شما کپی می کند. این فقط تعویض تعهدات است: اگر از محلی به محلی از راه دور باشد ، فشار دهید و اگر از راه دور به محلی است ، بکشید. نحو بسیار مشابه است:

```
$ git pull <remote_name> <branch_name>
```

به عنوان مثال ، اگر می خواهید از شعبه master در GitHub تعهداتی دریافت کنید ، باید ضمن بررسی شاخه master ، دستور را اجرا کنید:

```
$ git pull origin master
```

قبل از اجرای هر دستور ، اطمینان داشته باشید که همیشه در شاخه مربوط به روشی باشید که می کشید. بنابراین ، در این حالت ، شما باید استاد را قبل از اجرای کشش git بررسی کنید. پس از اجرای دستور ، نتیجه ای مانند من دریافت خواهید کرد همانطور که در شکل 1-13 نشان داده شده است.

```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git checkout master
Switched to branch 'master'

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git pull origin master
From https://github.com/mtsitoara/todo-list
* branch      master      -> FETCH_HEAD
Already up to date.

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ |
```

Figure 13-1. Pulling master from origin

از آنجا که شما همان وظایف را در مخزن محلی و On GitHub خود دارید ، هیچ اتفاقی نیفتاد. اما به محض شروع کار با افراد دیگر ، مجبور خواهید بود که شاخه های آنها را روی دستگاه محلی خود بکشید تا تغییرات آنها را مرور کنید یا به سادگی تغییرات را در GitHub بررسی کنید.

در اصل این است! کشیدن فقط کپی تعهدات از یک شعبه از راه دور به یک شعبه محلی است. و نگران نباشید ، به زودی فرصتهای بیشتری برای بازی کردن با git pull خواهید داشت.

What does a PR do

حالا که اطلاعات بیشتری در مورد کشیدن می دانیم ، تصویری واضح تر از این داریم که یک درخواست Pull چیست.

PR فقط درخواست مجوز برای انجام عمل کشش در یک مخزن از راه دور را دارد. اما کشیدن یک شاخه برای کامل شدن عمل کافی نیست: شما همچنین باید شاخه ها را با هم ادغام کنید.

به یاد دارید وقتی یک شاخه پچ را به شعبه توسعه ادغام کردیم؟ روابط عمومی فقط درخواست مجوز برای انجام این کار را می دهد. شما می توانید هر کاری را که می خواهید با شعبه های محلی خود انجام دهید ، اما وقتی با شعب بالادست (شعب موجود در مخزن از راه دور) سر و کار دارید ، باید کمی از تمدن استفاده کنید و ابتدا اجازه بخواهید. این اطمینان را می دهد که هر گونه رفع در شعب اصلی به درستی آزمایش و بررسی می شود.

بنابراین ، برای جمع آوری آن ، یک Pull Request درخواستی است که شما برای گرفتن GitHub برای انجام آن اعمال می کنید: شاخه وصله خود را بکشید و آن را با یک شاخه دیگر ادغام کنید. به عنوان مثال ، در پروژه ما ، در حال حاضر سه شعبه محلی (استاد ، توسعه و بهبود-خواندن-توصیف) و دو شاخه از راه دور (استاد و توسعه) داریم. اگر ما تعهدات جدیدی

را برای بهبود-توصیف خواندن قایل شدیم و می خواستیم آنرا با توسعه ادغام کنیم ، PR را باز می کردیم. پس از پذیرش روابط عمومی ، GitHub اقدامات زیر را انجام می دهد: شاخه بهبود-خواندن-توصیف را بکشید و سپس آن را با شاخه توسعه ادغام کنید. ممکن است از خود پرسید: "اگر انتهای یک درخواست Pull برای ادغام یک شاخه است ، چرا آنرا درخواست ادغام نمی نامید؟" خوب ، بسیاری از افراد (از جمله سایر خدمات میزبانی Git مانند GitLab) آنرا Merge Request می نامند. یعنی همین حرف در این کتاب از دو اصطلاح به صورت متقابل استفاده خواهیم کرد.

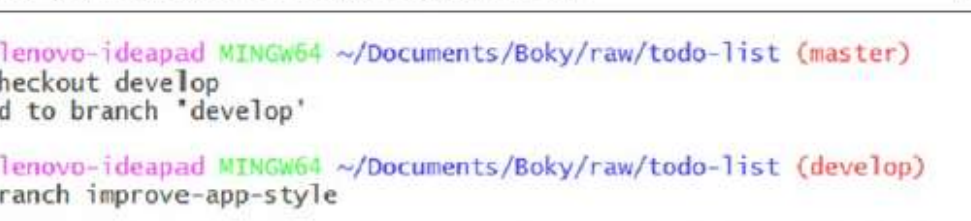
یک درخواست Pull ایجاد کنید

بریم به کسب و کار برسیم! ایجاد یک روابط عمومی جدید بسیار آسان است. شما فقط به دو شاخه احتیاج دارید: یکی برای کار کردن و دیگری برای ادغام. بیایید این کار را انجام دهیم! اول ، اجازه دهید مسئله ای ایجاد کنیم که روی آن کار کنیم. بنابراین به GitHub بروید و موضوعی با عنوان "بهبود سبک برنامه" ایجاد کنید. بله ، ما قبلاً مسئله مشابهی داشتیم ، اما از آنجایی که ما این مسئله را قبلاً حل کرده ایم ، قصد داریم موضوع جدیدی را باز کنیم. بازیافت مجدد مسائل ایده خوبی نیست زیرا پیگیری پیشرفت شما را دشوار خواهد کرد.

بعد از ایجاد مسئله ، وقت آن است که به هر ترمینال خود برگردید زیرا هر روابط عمومی با یک شعبه شروع می شود. ما می خواهیم شعبه ای با عنوان "بهبود برنامه" را از آخرین شاخه توسعه ، که ایجاد می شود ، ایجاد کنیم. همانطور که در فصل گذشته دیدیم ، نحوه ایجاد یک شاخه جدید از دیگری بررسی کردن شعبه منبع و اجرای دستور ایجاد شاخه است. بنابراین ، ما باید آن دستورات را یکی پس از دیگری اجرا کنیم:

```
$ git checkout develop
$ git branch improve-app-style
$ git checkout improve-app-style
```

بعد از اجرای آن سه فرمان ، می توانید با شعبه جدید درست مانند شکل 13-2 ، شاخه جدید خود را پیدا کنید.



The screenshot shows a Windows terminal window with the title bar "MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list". The terminal content is as follows:

```
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git checkout develop
Switched to branch 'develop'

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git branch improve-app-style

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git checkout improve-app-style
Switched to branch 'improve-app-style'

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (improve-app-style)
$ |
```

Figure 13-2. *Creation of a new branch*

در شعبه تازه ایجاد شده ما ، بگذارید روی مسئله کار کنیم. Openindex.html را باز کرده و محتویات آن را جایگزین کنید

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>TODO list</title>
        <style>
            h1 {
                text-align:center;
            }
            h3 {
                text-transform: uppercase;
            }
        
```



```

    ul {
        margin: 0;
        padding: 0;
    }
    ul li {
        cursor: pointer;
        position: relative;
        padding: 12px 8px 12px 40px;
        background: #eee;
        font-size: 18px;
        transition: 0.2s;
    }
    ul li:nth-child(odd) {
        background: #f9f9f9;
    }
    ul li:hover {
        background: #ddd;
    }
</style>
</head>
<body>
    <h1>TODO list</h1>
    <h3>Todo</h3>
    <ul>
        <li>Buy a hat for the bat</li>
        <li>Clear the fogs for the frogs</li>
        <li>Bring a box to the fox</li>
    </ul>
    <h3>Done</h3>
    <ul>
        <li>Put the mittens on the kittens</li>
    </ul>
</body>
</html>

```

سپس پرونده را مرحله بندی کرده و آماده انجام آن شوید. چیز ساده ای را به عنوان یک پیام متعهد ، بدون نیاز به مراجعه به موضوع ، قرار دهید. بعداً این کار را خواهیم کرد به عنوان یک پیام متعهد ، می توانید به سادگی قرار دهید: "تغییرات اساسی رنگ را در ردیف آیتم ها اضافه کنید." طبق معمول ، پیام تأیید را مطابق شکل 3-13 پس از تعهد دریافت خواهید کرد.



```
MINGW64:/c:/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (improve-app-style)
$ git add index.html
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (improve-app-style)
$ git commit
[improve-app-style a739045] Add basic color changes on item rows
1 file changed, 17 insertions(+), 4 deletions(-)
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (improve-app-style)
$ git status
On branch improve-app-style
nothing to commit, working tree clean
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (improve-app-style)
$ |
```

Figure 13-3. Commit confirmation

اکنون زمان آن رسیده که آن را به سمت GitHub سوق دهیم. همانطور که قبلاً دیده ایم، باید از دستور `git push` استفاده کنیم و به دنبال آن نام از راه دور و نام شعبه انجام می شود. بنابراین دستور خواهد بود

```
$ git push origin improve-app-style
```

بعد از اینکه شعبه خود را به GitHub فشار دادید، یک پیام تأیید آشنا دیگر دریافت خواهید کرد. می توانید شکل 4-13 را برای نمونه ای از این موضوع بررسی کنید.



```
MINGW64:/c:/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (improve-app-style)
$ git push origin improve-app-style
Username for 'https://github.com': mtsitoara
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 516 bytes | 516.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'improve-app-style' on GitHub by visiting:
remote:   https://github.com/mtsitoara/todo-list/pull/new/improve-app-style
remote:
To https://github.com/mtsitoara/todo-list.git
 * [new branch]      improve-app-style -> improve-app-style

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (improve-app-style)
$ |
```

Figure 13-4. Pushing the branch to GitHub

همانطور که در پیام تأیید مشاهده می کنید ، Git پیوندی را برای دنبال کردن به شما نشان می دهد تا بتوانید درخواست Pull را ایجاد کنید اما اجازه دهید PR را با یک روش دیگر مستقیماً در GitHub ایجاد کنیم.

به صفحه پروژه خود بروید و به دنبال چیزهای مختلف در ارائه باشید. پس از push اخیر به شعبه جدید ، صفحه پروژه شما می بایست مانند آنچه در شکل 13-5 نشان داده شده است ، باشد.

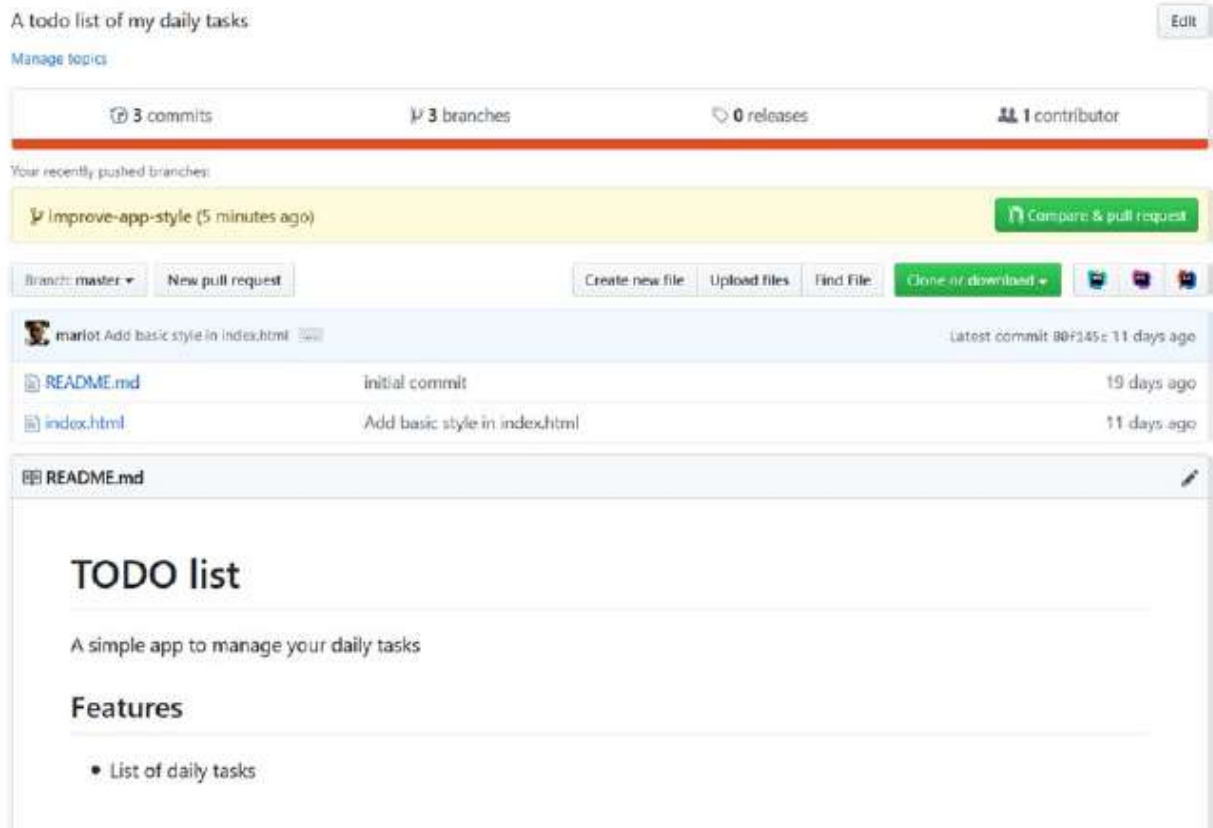


Figure 13-5. Project page after a recent push

همانطور که مشاهده می کنید ، یک تماس جدید برای اقدام در صفحه ، درست بالای لیست شعب وجود دارد. این نام شاخه ای را که اخیراً ایجاد کرده اید نشان می دهد و یک دکمه بزرگ برای ایجاد روابط عمومی است. برای ادامه روی دکمه کلیک کنید؛ درست مانند شکل 13-6 باید به فرم ایجاد Pull Request برسید.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: develop + compare: improve-app-style ✓ Able to merge. These branches can be automatically merged.

Add basic color changes on item rows

Write Preview

- Change item background color
- Change background color on item hover

Fix #7

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

1 commit 1 file changed 0 commit comments 1 contributor

Commits on Jul 25, 2019

mariot Add basic color changes on item rows a719045

Figure 13-6. Pull request creation form

می توانید توجه داشته باشید که فرم ایجاد روابط عمومی بسیار شبیه به فرم ایجاد مسئله است. در سمت راست ، می توانید همان اطلاعات را درباره تکالیف و برجسب ها پیدا کنید. آنها دقیقاً مشابه کار می کنند در پایین صفحه ، می توانید تعهدی را که توسط Pull Request اعمال می شود ، مشاهده کنید. و اگر به پایین بروید ، تفاوت بین نسخه ها را پیدا خواهید کرد.

شکل 13-7 را برای نمونه ای از این موضوع بررسی کنید.

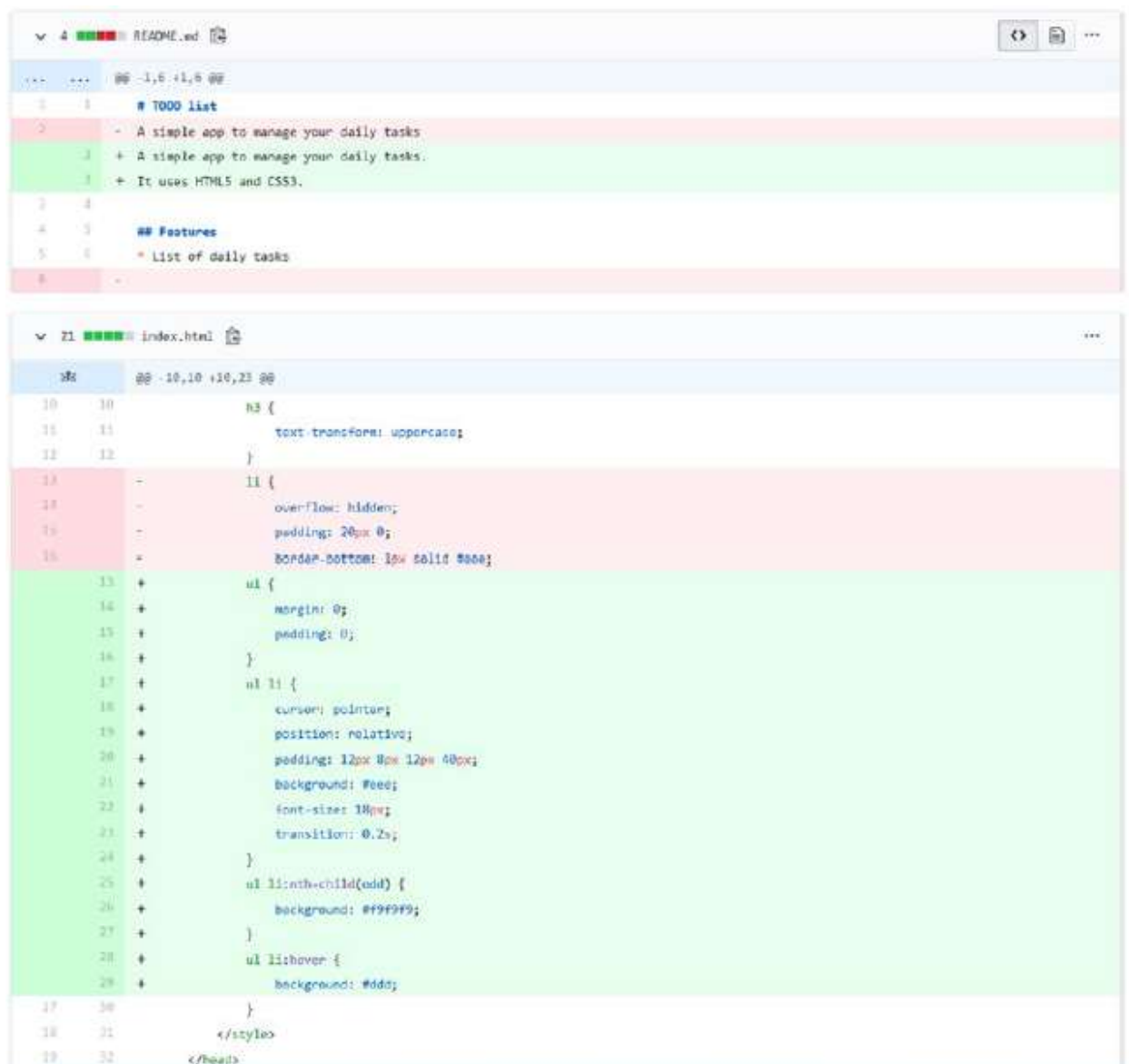


Figure 13-7. Differences between versions

اما ممکن است از خود پرسید که چرا دو تعهد وجود دارد که باید اعمال شود. به دلیل شعبه هدف است. اگر شکل 6-13 را از نزدیک بررسی کنید، می فهمید که شاخه اصلی روابط عمومی استاد است. این همان چیزی نیست که ما می خواهیم، زیرا ما شاخه توسعه را هدف قرار می دهیم. پیش بروید و شاخه پایه را تغییر دهید تا توسعه یابد. پس از تغییر آن، صفحه بارگیری مجدد می شود، و نتیجه متفاوتی می گیرید، که در شکل 8-13 نشان داده شده است.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: develop + compare: improve-app-style ✓ Able to merge. These branches can be automatically merged.

Add basic color changes on item rows

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers: No reviews

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

1 commit 1 file changed 0 commit comments 1 contributor

Commits on Jul 25, 2019

mariot Add basic color changes on item rows a730845

Figure 13-8. Pull Request on develop

وقتی آن را تغییر می دهید ، توجه کنید که نام PR نیز تغییر کرده است. دلیل این است که نام PR آخرین پیام متعهد را به عنوان یک نام پیش فرض در نظر می گیرد. اما اگر بخواهید می توانید آن را تغییر دهید ، به ویژه اگر چندین تعهد در یک روابط عمومی داشته باشید. یک مورد را در مورد نام روابط عمومی به یاد داشته باشید: باید به عنوان پیام های متعهد کاملاً واضح و روشن باشد. نام روابط عمومی شما باید به این سؤال پاسخ دهد: "اگر من این را ادغام کنم ، این روابط عمومی چه خواهد کرد؟" بنابراین از نام و توضیحات PR خود خوب مراقبت کنید تا داوران بتوانند بدون خواندن کد خود مشکل شما را حل کنند و بدانند.

می توانید توضیحات PR خود را در جعبه متن توضیحات گسترش دهید و از ارائه اطلاعات بیشتر در مورد تغییرات دریغ نکنید. شما باید کلمات کلیدی را برای بستن مسائل در آنجا قرار دهید. شکل 9-13 برای نمونه ای از این موضوع بررسی کنید.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base: develop ← compare: improve-app-style → ✓ Able to merge. These branches can be automatically merged.

Add basic color changes on item rows

Write Preview

AA B i “ < > ↻

• Change item background color

• Change background color on item hover

Fix #7


Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers

No reviews

Assignees

 mbsitoara

Labels

enhancement

Projects

None yet

Milestone

No milestone


1 commit

1 file changed

0 commit comments

1 contributor

Commits on Jul 25, 2019

 marlot

Add basic color changes on item rows

a77984

Figure 13-9. A completed Pull Request

پس از آماده شدن ، روی "ایجاد درخواست کشیدن" کلیک کنید تا آن را انجام دهید. شما به صفحه ای شبیه به صفحه نشان داده شده در شکل 10-13 خواهید رسید.

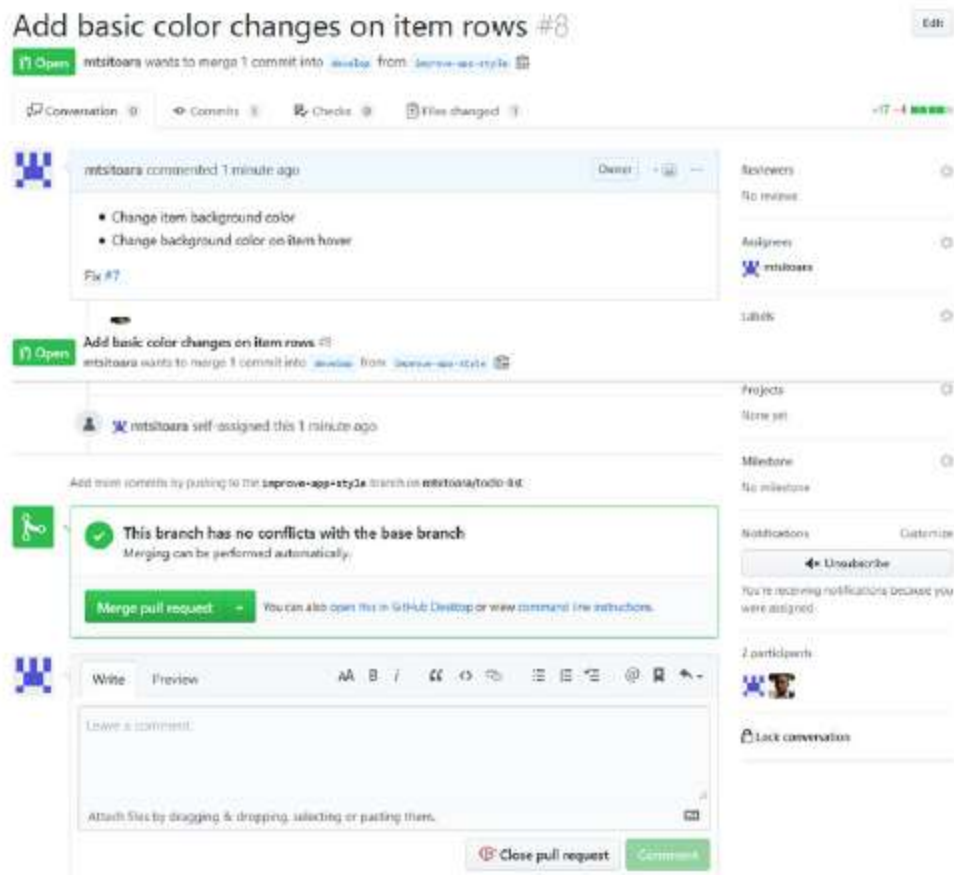


Figure 13-10. Your new Pull Request

باز هم ، این دیدگاه بسیار شبیه به همتای Issues است ، حتی شماره PR شماره Issues را دنبال می کند. تنها تفاوت دکمه ادغام درخواست کشش است. اگر روی این دکمه کلیک کنید ، PR پذیرفته می شود و شاخه ها ادغام می شوند. اما این کار را انجام ندهید! بیا بید قبل از ادغام با PR ما بازی کنیم.

بررسی کد

بررسی کد یکی از بهترین ویژگی های GitHub است. مدت ها پیش رفته است که روزهایی که شما مجبورید یک جلسه یک به یک را با Tech Lead خود برنامه ریزی کنید تا بتوانند کد شما را بررسی کنند. دیگر نیازی به ارسال زنجیره های طولانی ایمیل الکترونیکی (با یک لیست طولانی از افراد اذیت شده در لیست سی سی) برای هر درخواست تغییر در کد نیست. اکنون ، همه چیز در GitHub انجام می شود. اجازه بدید ببینم!

یک بررسی کد ارائه دهید

در شکل 9-13، شما نگاهی اجمالی به روند بررسی Code دارید. همه تغییرات انجام شده در پرونده ها را در مقایسه با نسخه فعلی مشاهده کردید، اما هنوز نمی توانید با آنها ارتباط برقرار کنید.

در این بخش یاد می گیرید که چگونه کد همکاران خود را مرور کنید. می توانید در شکل 10-13 مشاهده کنید که صفحه روابط عمومی، مانند صفحه Issues، بخش های زیادی دارد. برای شروع بررسی کد، باید روی "پرونده ها تغییر کرد" کلیک کنید. سپس به صفحه ای شبیه به صفحه نشان داده شده در شکل 11-13 خواهید رسید.

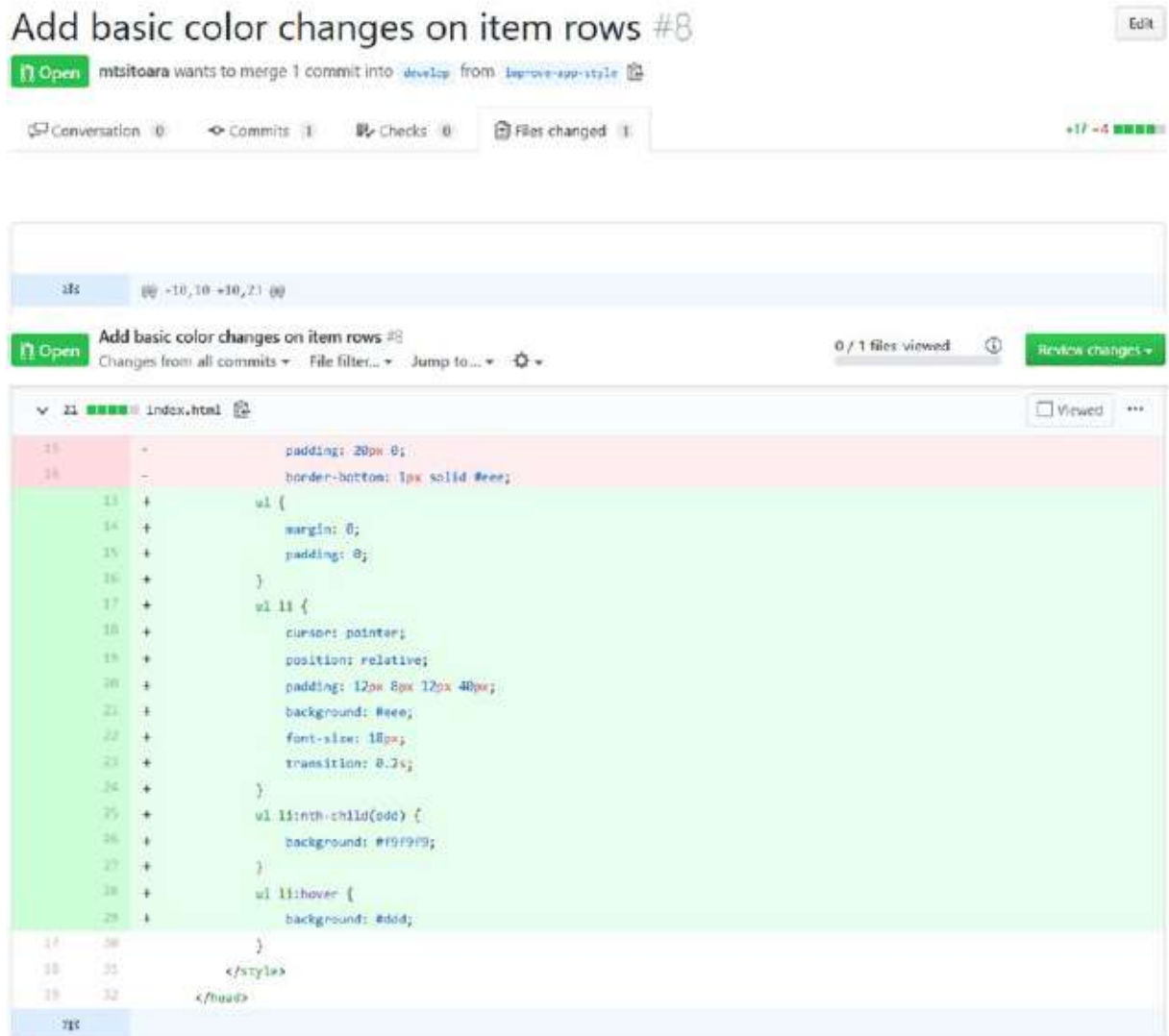


Figure 13-11. The Code Review section

این دیدگاه باید نتایج git diff را به شما یادآوری کند، زیرا در اصل همین موضوع است. تفاوت بین نسخه ها را با جزئیات به شما نشان می دهد، بدین معنی که می بینید چه چیزی اضافه شده، حذف شده یا جایگزین شده است.

نظر بدهید

اکنون ، بیایید وانمود کنیم که این کد را مرور خواهیم کرد. در حین بررسی کد ، می توانید در مورد کل تغییرات یا یک کد خاص نظر دهید. برای مثال ، بگذارید در مورد تعریف "ul li" در خط 17 نظر بگذاریم.

همانطور که در اطراف مکان نما خود را بر روی بررسی کد تغییر دهید ، آیکون "plus" را دنبال کنید. این بدان معنی است که شما می توانید در آنجا اظهار نظر کنید. مکان نما را روی خط 17 قرار دهید و هنگامی که نماد "plus" نشان داد ، روی آن کلیک کنید. مانند بخش شکل 12-13 ، یک بخش نظر کوچک باز می شود.

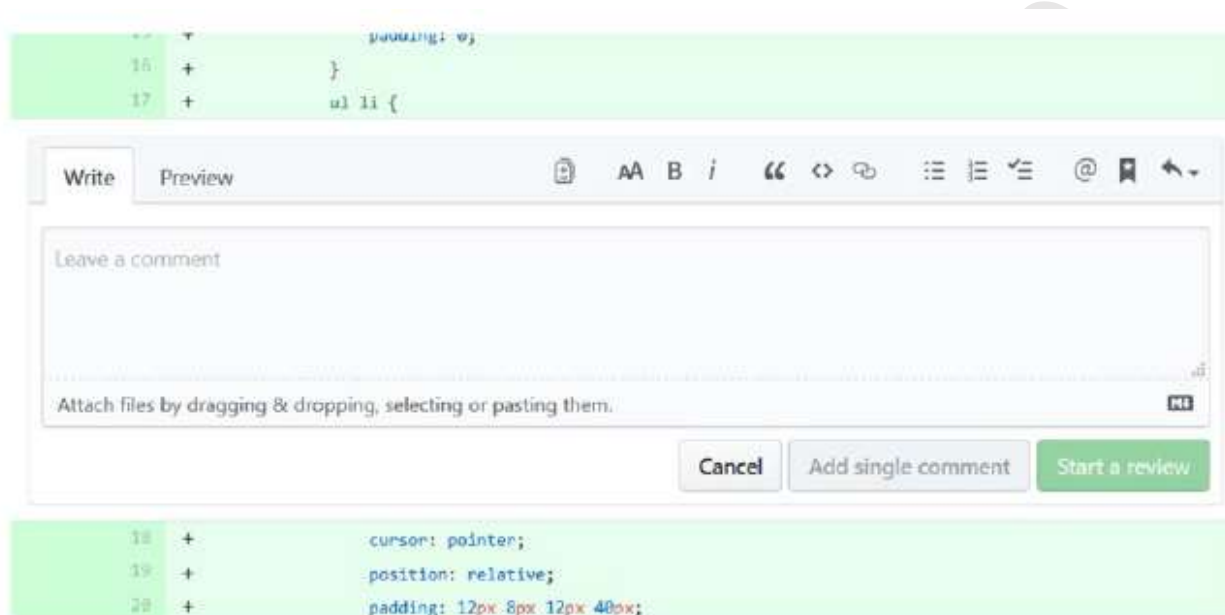


Figure 13-12. A code review on a line

مثل همیشه ، شما می توانید با کمک نحو Markdown ، انواع نظرات را در این بخش بیان کنید. برای این مثال ، ما می خواهیم این نظر را ارائه دهیم: "موارد لیست را برای UX پاک کننده غیرقابل انتخاب کنید. از "انتخاب کاربر: هیچ" استفاده کنید. " قبل از ترک نظر ، دقیقاً مانند شکل 13-13 باید پیش نمایش را بررسی کنید.



اگر از نظر خود راضی هستید ، بر روی "شروع بازیابی" کلیک کنید تا به مرحله بعدی بروید. این نظر در صفحه نقد نمایش داده می شود ، و دقیقاً مانند نتیجه ای که در شکل 13-14 نشان داده شده است ، یک دکمه پاسخی در مورد نظر نیز وجود خواهد داشت.



Figure 13-14. The posted comment

با استفاده از این دکمه ، توسعه دهنده می تواند قبل از شروع کار مجدد در PR ، نظر خود را با نظر بررسی کند. در صورت تمایل می توانید اظهارنظرهای بیشتری انجام دهید ، زیرا نظرات در اصل همان چیزی هستند که یک بررسی کد را تشکیل می دهند. اگر راضی هستید ، روی دکمه "پایان بررسی خود" در بالای صفحه کلیک کنید. شما مانند بخش دیگری که در شکل 13-15 نشان داده شده است ، دوباره با یک بخش کوچک استقبال می شوید.

The screenshot shows the GitHub review interface. At the top, there are tabs for 'Write' and 'Preview'. Below the tabs is a rich text editor with various formatting icons (bold, italic, quote, code, link, list, etc.). The main area is a large text box for leaving a comment, with a placeholder text 'Leave a comment'. Below the text box is a message: 'Attach files by dragging & dropping, selecting or pasting them.' with a small icon. Underneath, there are three radio buttons for selecting the type of review: 'Comment' (selected), 'Approve', and 'Request changes'. Each radio button has a description: 'Submit general feedback without explicit approval.', 'Submit feedback and approve merging these changes.', and 'Submit feedback that must be addressed before merging.' respectively. At the bottom, there is a green button labeled 'Submit review' and a status indicator '1 pending comment'.

Figure 13-15. Finishing the review

پس از اتمام بررسی ، شما سه گزینه دریافت خواهید کرد: اظهار نظر ، تأیید یا درخواست تغییرات. از آنجا که این PullRequest خود ما است ، ما نمی توانیم تغییراتی را در این مورد تأیید یا درخواست کنیم ، بنابراین فقط گزینه پیش فرض را انتخاب می کنیم ، که بازخورد کلی در مورد تغییرات است. بیایید بگذاریم: "فراموش نکنید که مرورگرهای مختلف را در نظر بگیرید" به عنوان نظر و بررسی را ارسال کنید. شما یک بار دیگر به صفحه جزئیات روابط عمومی مطابق شکل 13-16 باز خواهید گشت.



Figure 13-16. Your completed Code Review

صفحه جزئیات PR نظرات مختلفی را که توسط داور و همچنین نظرات کلی برای کل PR ارائه می شود به شما نشان می دهد. بیاپید این نظرات را حل کنیم.

یک درخواست را به روز کنید

نظری که از طرف ناظر باقی مانده است پیشنهاد می کند که قبل از پذیرش روابط عمومی ما باید برخی از کدها را تغییر دهیم بنابراین ، بیاپید این کار را انجام دهیم! بیاپید PR را با فشار آوردن تعهدات جدید به شاخه وصله به روز کنیم.

index.html را باز کنید و محتوای آن را به این زیر تغییر دهید:

```
type html>
>
head>
  <meta charset="utf-8">
  <title>TODO list</title>
  <style>
    h1 {
      text-align:center;
    }
    h3 {
      text-transform: uppercase;
    }
    ul {
      margin: 0;
      padding: 0;
    }
    ul li {
      cursor: pointer;
      position: relative;
      padding: 12px 8px 12px 40px;
      background: #eee;
      font-size: 18px;
      transition: 0.2s;
      -webkit-user-select: none;
      -moz-user-select: none;
      -ms-user-select: none;
      user-select: none;
    }
    ul li:nth-child(odd) {
      background: #f9f9f9;
    }
    ul li:hover {
      background: #ddd;
    }
  </style>
</head>
```

```

    </style>
</head>
<body>
  <h1>TODO list</h1>

  <h3>Todo</h3>
  <ul>
    <li>Buy a hat for the bat</li>
    <li>Clear the fogs for the frogs</li>
    <li>Bring a box to the fox</li>
  </ul>

  <h3>Done</h3>
  <ul>
    <li>Put the mittens on the kittens</li>
  </ul>
</body>
</html>

```

یک بار دیگر پرونده را مرحله بندی کنید و پروژه را با این پیام انجام دهید: "موارد لیست را انتخاب نکنید." سپس ، شاخه را دوباره به GitHub فشار دهید. اگر در این تمرین گم شدید ، بخش قبلی را بررسی کنید. بعد از اینکه شعبه را PUSH دادید ، دوباره به صفحه روابط عمومی برگردید. در صفحه جزئیات توضیحات جدید را مشاهده خواهید کرد. شکل 12-17 را برای نمونه ای از این موضوع بررسی کنید.

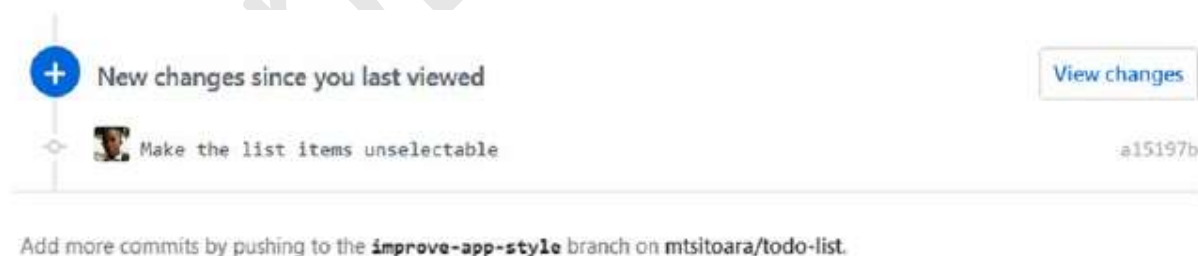


Figure 13-17. New changes detected by GitHub

بعد از انجام هر کاری که انجام دهید ، GitHub PR را به روز می کند تا تغییرات اعمال شده در شعبه را منعکس کند. برای دیدن تغییرات جدید روی "نمایش تغییرات" کلیک کنید. شما یک بار دیگر به صفحه Code Review وارد خواهید شد اما با کمی پیچ و خم: فقط متوجه تغییرات جدید به معنی تغییراتی خواهید شد که هنوز ندیده اید. این باعث می شود که برای نظر نویسنده پیشرفت PR را آسان تر کند.

از آنجا که ما هیچ نظر اضافی نداریم ، پیش بروید و روی "بررسی نهایی" کلیک کنید و سپس یک نظر کلی ارائه دهید در یک محیط کار ، شما کد خود را مرور نمی کنید ، بنابراین انتخاب تأیید در دسترس خواهد بود. اما از آنجا که ما به تنهایی کار می کنیم ، فقط یک نظر کلی مانند "کار خوب" بدهید از آنجا که توسعه دهنده بسیار سخت کار کرده است نظر کلی دقیقاً مانند شکل 13-18 در صفحه جزئیات PR ظاهر می شود.

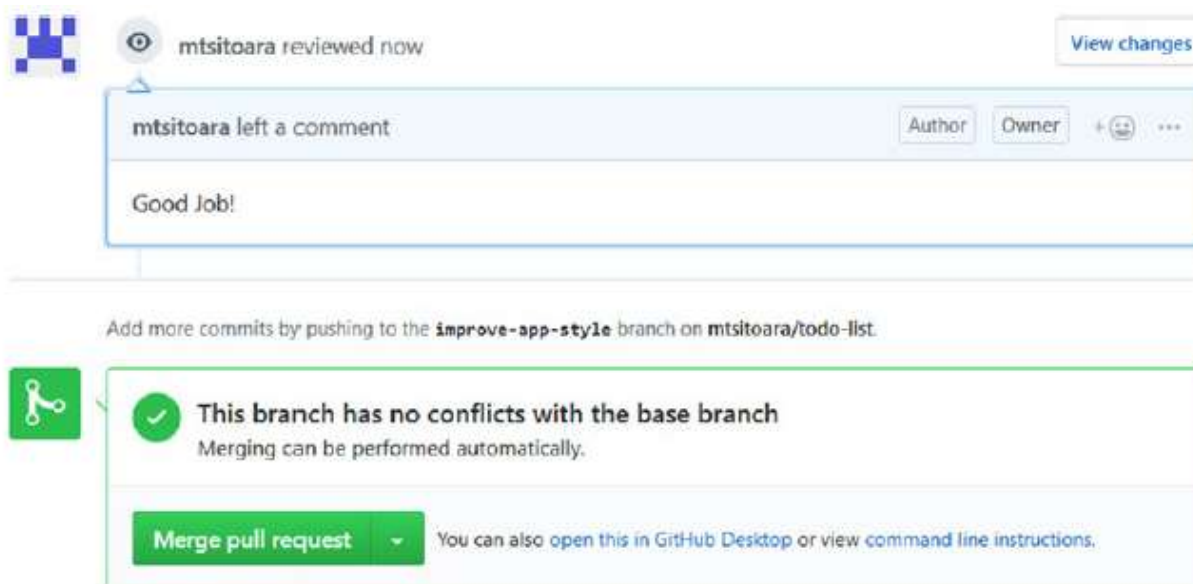


Figure13-18. A final comment has been made

اکنون می توانیم با خیال راحت شعبه خود را به شاخه پایه ادغام کنیم زیرا کد ما به درستی بررسی می شود. برای پذیرش و ادغام PR ، روی دکمه بزرگ سبز کلیک کنید. قبل از ادغام شعبه از شما درخواست تأیید خواهد شد. پس از تأیید آن ، شعب ادغام می شوند و PR بسته می شود. حتی می توانید شاخه منبع را در صورت تمایل ، درست همانطور که شکل 13-19 نشان می دهد ، حذف کنید.



Figure 13-19. Pull Request accepted

این که آیا می خواهید شاخه را حذف کنید یا خیر ، به عهده شماست. بعضی اوقات ، تیم ها شاخه ها را حذف نمی کنند تا اینکه یک آزمایش کننده تأیید کند که همه چیز خوب است.

"اما چرا مسئله من به طور خودکار بسته نمی شود؟" شما این را می پرسید زیرا تعمیر در شاخه توسعه ، که شاخه پیش فرض آن نیست. فقط رفع اشکال ادغام شده در شاخه پیش فرض (کارشناسی ارشد) به طور خودکار مسائل را بسته می کند. اما از آنجایی که شما در مورد آن مسئله نگران هستید ، قبل از اینکه به فصل بعد برویم کمی ورزش کنید.

خلاصه

تبریک می گویم که اولین PR خود را پذیرفته اید! (اما اگر خودتان آنها را نپذیرید چشمگیرتر خواهد بود). این فصل بسیار طولانی بوده است ، اما برای بهره مندی از ویژگی های عالی GitHub باید آن را کاملاً درک کنید. برای مشکلات بعدی خود ، به جای تعهد مستقیم به استاد ، PR را باز کنید.

و به یاد داشته باشید که در بیشتر تنظیمات حرفه ای ، تعهد بر روی استاد نه تنها دلسرد می شود بلکه به طور پیش فرض توسط GitHub انکار می شود. هر تغییر باید از یک درخواست Pull باشد.

شما باید در حال حاضر با استفاده از PR راحت باشید. اگر نه ، بخش های اول این فصل را دوباره بخوانید. تنها نکته ای که باید به خاطر بسپارید این است که یک درخواست کشش فقط یک روش جالب برای درخواست مجوز برای اعمال تعهدات بر روی یک شعبه است.

ممکن است اکنون سؤالاتی پیش بیاید: "اگر کسی قبل از اتمام روابط عمومی من تغییراتی را در شاخه پایه اعمال کند؟" ، "اگر شخص دیگری همان پرونده من را تغییر دهد؟" یا "اگر من وظیفه داشته باشم چه می شود؟ اگر من مشغول کار با PR هستم ، مسئله دیگری را حل کنید؟" این سؤالات در واقع بسیار مهم هستند؛ به همین دلیل ما آنها را در فصل بعدی پوشش خواهیم داد. ما با ادغام اختلافات و چگونگی حل آنها مقابله خواهیم کرد. اما قبل از یادگیری چگونگی حل آنها ، خواهیم آموخت که چگونه به طور کلی از آنها جلوگیری کنیم! بیا بریم!