

# قسمت سوم

## کار تیمی با گیت

### فصل چهاردهم

### درگیری ها

فصل گذشته ما را به دنیای شگفت انگیز MergeRequests معرفی کرده است شما باید بدانید که کاربرد آنها چیست و چرا استفاده از آنها ایده خوبی است. حتی اگر آنها یک مفهوم نسبتاً ساده برای درک آن باشند ، با برخی کاستی ها همراه هستند که نادیده گرفتن آنها دشوار است.

شما پیشتر با سفر خود به پایان رسیده اید ، مسیری طولانی را طی کرده اید ، اما هنوز چیزهایی وجود دارد که باید قبل از ادامه مسیر خودتان همه را یاد بگیرید. شما باید یاد بگیرید که در طول مسیر چه مشکلی دارید. ما در این فصل قصد داریم در مورد آن مشکلات صحبت کنیم. ابتدا نحوه بررسی ادغام شاخه ها را مجدداً بررسی خواهیم کرد و سپس مشکلاتی را که احتمالاً در حرفه خود مرتفع کرده اید ، ارائه خواهیم داد. سرانجام ، ما راه حل های متداول برای این مشکلات را خواهیم دید. از تعارض نترسید زیرا آنها به راحتی حل می شوند؛ آنها فقط آزار دهنده هستند

### چگونه یک ادغام کار می کند

بگذارید کمی به عقب برگردیم و به اصول اولیه برگردیم: ادغام چه کاری انجام می دهد؟ یک ادغام هر یک از ارتکابها را در یک شاخه انجام می دهد و آنها را در مورد دیگری اعمال می کند. درست است؟ خوب ، یک ادغام به خوبی برنامه ریزی شده بیشتر مواقع بدون مشکل پیش می رود. اما حتی اگر آخرین جزئیات را برنامه ریزی کنید ، چیزی وجود دارد که نمی توانید کنترل کنید: آنچه دیگران انجام می دهند.

فراموش نکنید که Git یک سیستم کنترل نسخه توزیع شده است ، به این معنی که هر مشارکت کننده نسخه ای از پروژه خود را دارد و می تواند هر کاری را برای مخزن محلی خود انجام دهد. هر کس می تواند هر پرونده را تغییر دهد زیرا هیچ "فصل پرونده" مانند برخی از VCS وجود ندارد. این بدان معنی است که مواردی وجود دارد که چندین نفر در همان پرونده تغییراتی ایجاد کرده اند. با هم آوردن همه این تغییرات ، ادغام لازم است. قبل از رفتن به قسمت بعدی ، باید یک چیز را به خاطر

بسیارید: فقط وقتی مطمئن باشید که تعهدات در آن شعبه نهایی است ، یک شاخه را ادغام کنید. ادغام شاخه ای که شامل کار ناتمام است ، هدف از انشعاب-داشتن سابقه ای واضح را خراب می کند. باز کردن درخواست بیرون کشیدن حتی اگر نمی خواهید در واقع ادغام شاخه ها خوب باشد ؛ اما در واقع ادغام آنها ناقص نیست.

همانطور که قبلاً گفتیم ، ادغام با یک شاخه آغاز می شود. بیشتر اوقات ، این شاخه ای است که شما هنوز در مخزن محلی خود ندارید. بنابراین ، شما باید آن را از مبدأ بکشید (نام پیش فرض مخزن از راه دور).

## Pulling

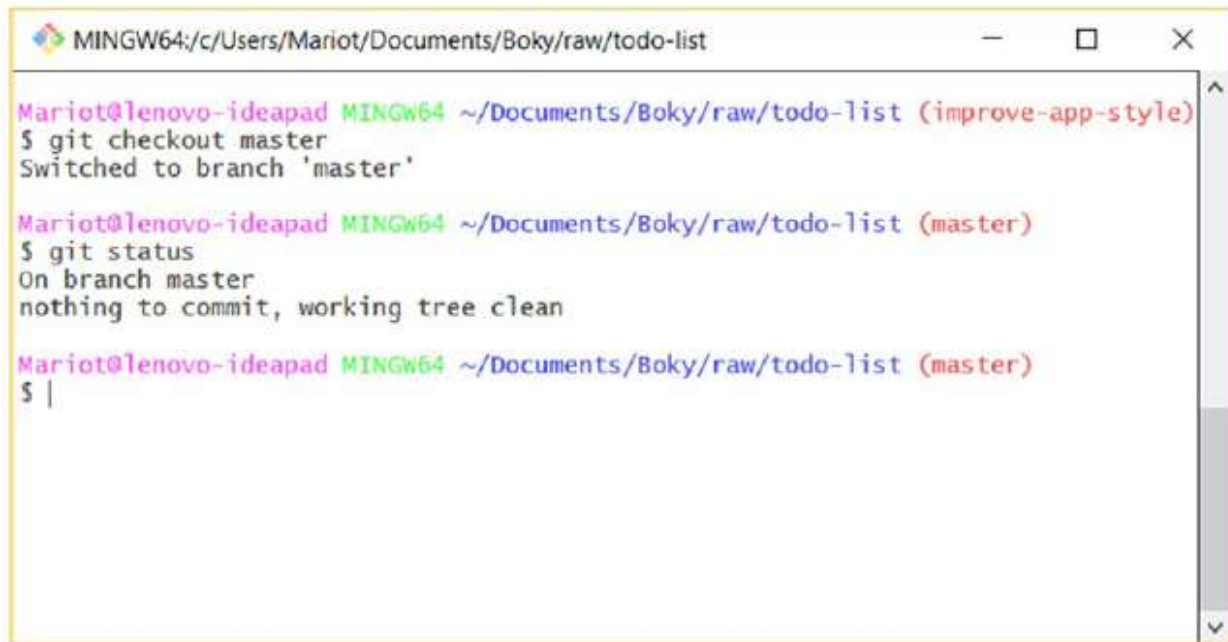
بیا یاد بگیریم دوباره فرمان کشیدن را مرور کنیم. کشیدن به معنای کپی کردن یک شاخه از راه دور در مخزن محلی است. به عنوان مثال ، ما یک شاخه را در توسعه و master ادغام کرده ایم اما کاری به شعب محلی ما انجام نداده است. این بدان معناست که ما در جدول زمانی تاریخ "behind هستیم" زیرا در مخزن از راه دور تعهدی وجود دارد که ما نداریم.

در حقیقت ، کلمه "behind" کمی غلط است زیرا همانطور که ما تأسیس کردیم ، هر مخزن مستقل است و هیچ مخزن مرکزی در Git وجود ندارد. ما تصمیم گرفتیم که یک مخزن استاد راه دور داشته باشیم زیرا کار در تیم ها آسان تر می شود. اما ، به طور دقیق ، شما می توانید تعهدات خود را به عنوان مورد علاقه خود مبادله کنید. مفهوم "پشت" فقط برای آسانتر کردن زندگی توسعه دهندگان ابداع شد.

بیا یاد سعی کنیم master را به مخزن محلی خود بکشیم. به یاد داشته باشید که قبل از انجام مراحل بعدی در این فصل باید تمرین را از فصل آخر (ادغام توسعه در استاد) به پایان برسانید. ابتدا شعبه استاد محلی خود را بررسی کنید و مطمئن شوید که تمیز است.

```
$ git checkout master  
$ git status
```

اگر در فهرست کار خود هیچ کاری خنده دار نکردید ، باید همان نتیجه را در شکل 14-1 دریافت کنید: یک فهرست راهنما.



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

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)
$ |
```

**Figure 14-1.** A clean directory is needed before a pull

اکنون ، اجازه دهید قبل از هرگونه تغییر ، گزارش تاریخ را بررسی کنیم.

```
$ git log --online
```

این منجر به خروجی تاریخچه شاخه اصلی خواهد شد. تغییرات اخیر ما را اعمال نمی کند زیرا این تغییرات در حال حاضر فقط در مخزن راه دور هستند. سابقه کارشناسی ارشد تاریخ باید شبیه نمونه ای باشد که در شکل 14-2 نشان داده شده است.



```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git log --online
80f145c (HEAD -> master, origin/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 (master)
$ |
```

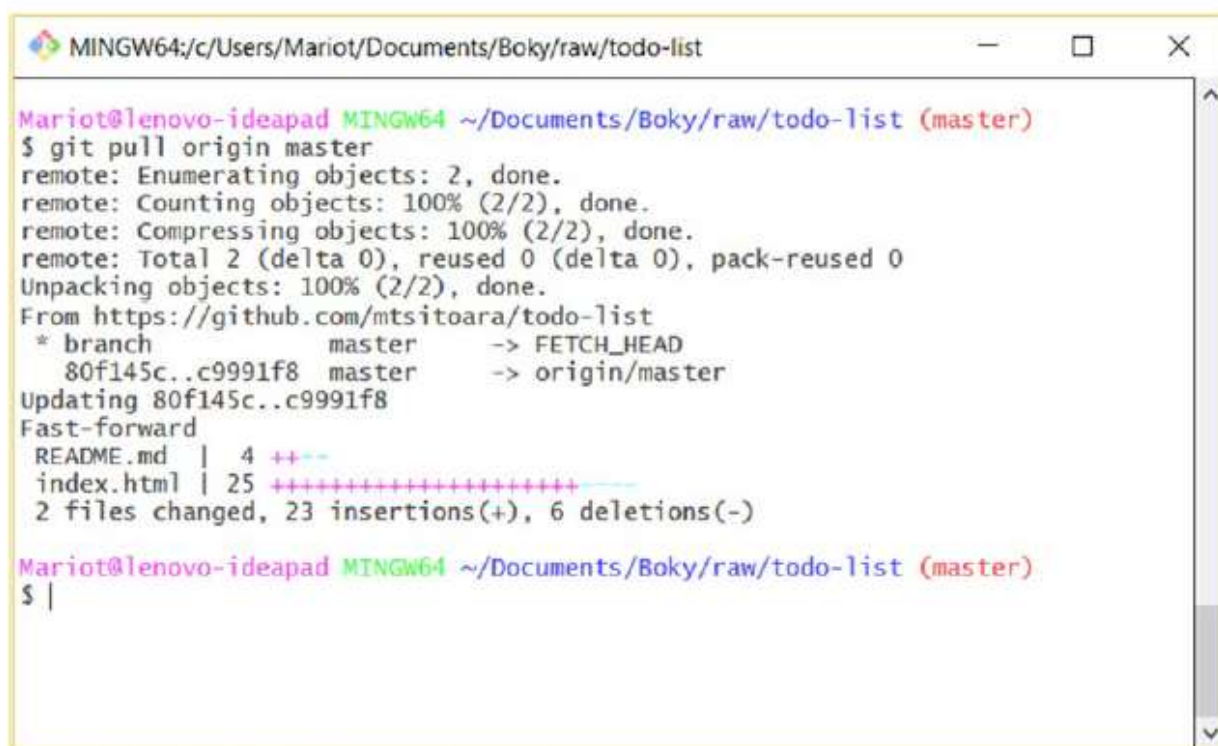
**Figure 14-2.** The history log before the pull

همانطور که در شکل 2-13 مشاهده می کنید ، HEAD در آخرین تعهد شعبه در حال حاضر (و بیشتر اوقات ، اینگونه خواهد بود) نشان داده شده است. با توجه به این نتیجه ، شعبه استاد محلی ما و شاخه استاد از راه دور در یک سطح قرار دارند ، به این معنی که آنها دارای همان تعهدات هستند. ما می دانیم که این درست نیست زیرا ما تغییری را در استاد از راه دور ایجاد کرده ایم. Gitrepository محلی ما این را نمی داند زیرا ما هنوز هیچ تعهدی از سرور دریافت نکرده ایم. بیا این کار را اینجا دهیم.

همانطور که در آخرین فصل مشاهده کردیم ، دستور فرمان pull and push را به همان روش انجام می دهیم: فقط باید نام مخزن از راه دور و نام شعبه از راه دور را به عنوان پارامترها عبور دهید. بنابراین دستور خواهد بود

```
$ git pull origin master
```

پس از اجرای این کد در یک فهرست کار تمیز ، نتیجه نشان داده شده در شکل 3-14 را دریافت خواهید کرد.



```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git pull origin master
remote: Enumerating objects: 2, done.
remote: Counting objects: 100% (2/2), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (2/2), done.
From https://github.com/mtsitoara/todo-list
 * branch                master       -> FETCH_HEAD
   80f145c..c9991f8      master       -> origin/master
Updating 80f145c..c9991f8
Fast-forward
 README.md | 4 ++--
 index.html | 25 ++++++
 2 files changed, 23 insertions(+), 6 deletions(-)

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

**Figure 14-3.** Pulling master from origin

ادغام سریع به جلو

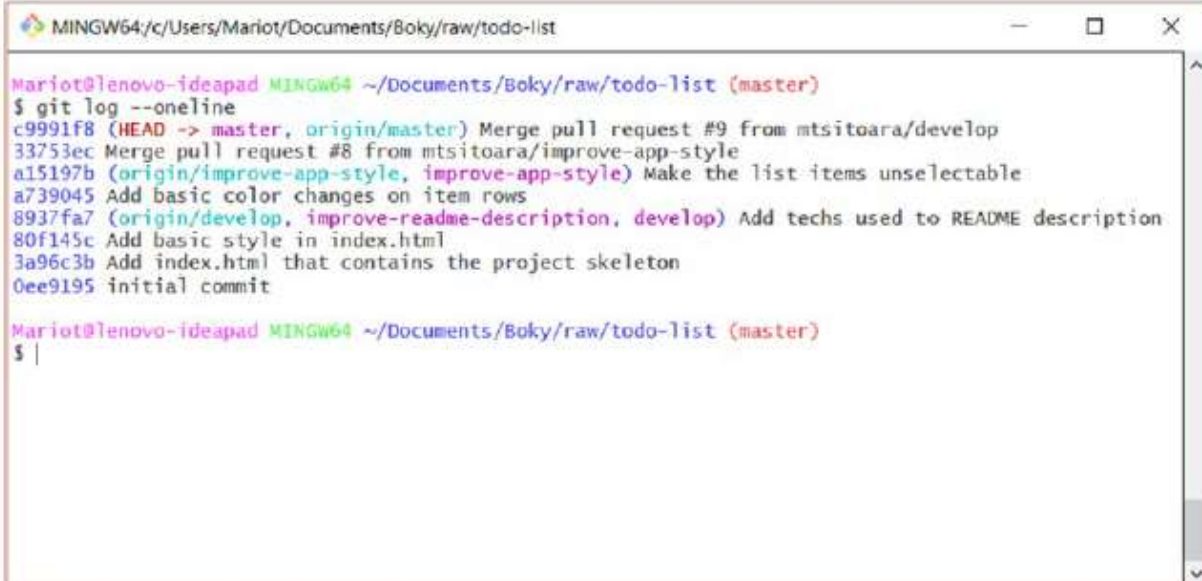
بعد از اینکه استاد را از مبدأ بیرون کشیدید ، خلاصه ای از این عملیات را دریافت می کنید. شما تعداد فایل‌های تغییر یافته و نوع ادغام انجام شده را مشاهده خواهید کرد. در اینجا ، نوع "سریع به جلو" است و ساده ترین نوع ادغام است. حرکت سریع به معنای این است که تعهدات مربوط به شعبه از راه دور در همان جدول زمانی که شعبه محلی انجام شده بود ، بنابراین Git فقط باید HEAD را به آخرین تعهد شعبه اصلی منتقل می کند. به یاد داشته باشید وقتی درمورد تعهدات درمورد روابط والدین و فرزندان با دیگران صحبت کردیم؟ اگر Git ارتباط بین کمیسیون را پیدا کند که شاخه اول و شاخه ای که باید

ادغام شود ، یک ادغام سریع به جلو انجام می شود ، به این معنی که فقط یک حرکت اشاره گر لازم است ، که باعث می شود Git بسیار سریع شود. شما همیشه باید سعی کنید از سریع استفاده کنید و به عنوان روشی برای ادغام استفاده کنید زیرا این کار ساده تر و مهمتر از همه پاک ترین کارنامه تاریخ است.

در مورد گزارش تاریخچه صحبت می کنیم ، اجازه می دهیم تا تغییرات را که از سرور دریافت کردیم ، بررسی کنیم. بار دیگر ، از گزینه "online--" استفاده کنید تا نتیجه زیباتر بگیرید.

```
$ git log --online
```

Result



```
MINGW64~/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git log --online
c9991f8 (HEAD -> master, origin/master) Merge pull request #9 from mtsitoara/develop
33753ec Merge pull request #8 from mtsitoara/improve-app-style
a15197b (origin/improve-app-style, improve-app-style) Make the list items unselectable
a739045 Add basic color changes on item rows
8937fa7 (origin/develop, improve-readme-description, develop) Add techs used to README description
80f145c 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 14-4.** History log after pulling from origin

شما تعهدات اضافی گرفتید! تعهدات شعبه از راه دور در شعبه محلی شما ادغام شدند. اکنون ، شاخه ارشد محلی شما به همان تعهد شعبه مبدا اشاره دارد.

بگذارید همه این موارد را باز کنیم. ابتدا ، در مورد رنگ شاخه ها صحبت خواهیم کرد. شاخه های سبز شاخه های محلی شما هستند ، در حالی که شاخه های قرمز از راه دور هستند. Remote همچنین دو نام دارد زیرا نام آنها با نام مخزن از راه دور ترکیب می شود.

می بینید که بهبود-خواندن-توصیف ، توسعه و منشاء / توسعه در یک سطح هستند. ما می دانیم که این صحیح نیست زیرا ما توسعه را از GitHub تغییر داده ایم. Gitwon نمی داند که تغییرات تا زمانی که شاخه توسعه را از مبدا بیرون نکشید ، رخ داده است.

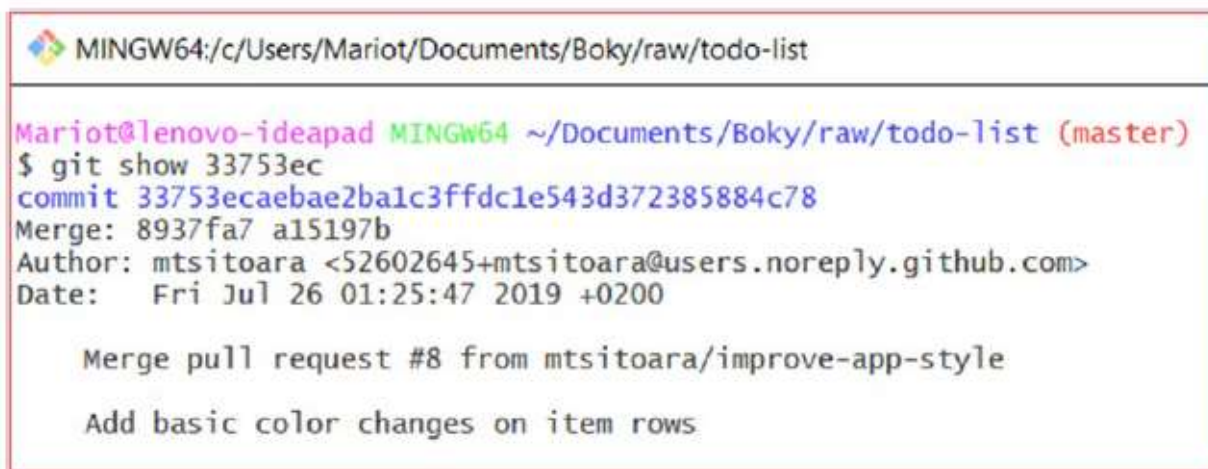
متوجه خواهید شد که تعهدی که در مورد این تاریخ انجام نداده اید ، یعنی " Merge pull request #8 from " mtsitoara/improve-app-style" وجود دارد که به آنها ادغام می شوند و توسط Git ایجاد می شوند که دو یا چند تعهد

را با هم ادغام می کنید. در پروژه ما ، سبک بهبود برنامه را در توسعه ادغام کرده و سپس به استاد تبدیل شده است. هر یک از این ادغامها یک تعهد ادغام ایجاد می کنند.

درست مانند تعهدات عادی ، می توانید با استفاده از دستور `git show` اطلاعات بیشتری در مورد آن نشان دهید. بیایید اولین تعهد ادغام را نشان دهیم.

```
$ git show 33753ec
```

این یک نمای آشنا برای ما خواهد بود: نمای `intel`. شما باید همان نتیجه را در شکل 14-5 دریافت کنید.

A screenshot of a terminal window with a red border. The title bar shows the path 'MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list'. The terminal content shows the command '\$ git show 33753ec' and its output: 'commit 33753ecaebae2ba1c3ffdc1e543d372385884c78', 'Merge: 8937fa7 a15197b', 'Author: mtsitoara <52602645+mtsitoara@users.noreply.github.com>', 'Date: Fri Jul 26 01:25:47 2019 +0200', 'Merge pull request #8 from mtsitoara/improve-app-style', and 'Add basic color changes on item rows'.

```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git show 33753ec
commit 33753ecaebae2ba1c3ffdc1e543d372385884c78
Merge: 8937fa7 a15197b
Author: mtsitoara <52602645+mtsitoara@users.noreply.github.com>
Date: Fri Jul 26 01:25:47 2019 +0200

Merge pull request #8 from mtsitoara/improve-app-style

Add basic color changes on item rows
```

**Figure 14-5.** The detailed view of a merge commit

این دیدگاه جالب نیست زیرا فقط والدین متعهد و کاربرانی را که ادغام شده اند نشان می دهد. یک نکته که باید به یاد داشته باشید این است که مرتکبین و ادغام می توانند افراد مختلفی باشند. و شما باید کلمات کلیدی خود را برای حل مسائل در پیام ادغام متعهد به جای پیام های متعهد قرار دهید. بیشتر اوقات ، تعهد کافی نخواهد بود برای حل یک مشکل؛ بنابراین ، آن کلمات کلیدی را در پیام درخواست `pull` قرار دهید تا مسئله در هنگام ادغام شاخه بسته شود.

تاریخچه ثبت شده در شکل 14-4 بسیار زیبا است ، اما در واقع مفهوم شاخه ها و ادغام ها را نشان نمی دهد. یک نمودار مناسب تر است ، و یک پارامتر برای آن در دستور `git log` وجود دارد. پارامتر `"graph--"` است و برای بدست آوردن بهترین نتیجه باید از آن با `"-"` آنلاین استفاده کنید.

```
$ git log --oneline --graph
```

این دستور نمودارهای ساده مانند نمونه نشان داده شده در شکل 14-6 را تولید می کند.



```

MINGW64~/c:/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git log --oneline --graph
* c9991f8 (HEAD -> master, origin/master) Merge pull request #9 from mtsitoara/develop
|
| * 33753ec Merge pull request #8 from mtsitoara/improve-app-style
| |
| | * a15197b (origin/improve-app-style, improve-app-style) Make the list items unselectable
| | * a739045 Add basic color changes on item rows
| | /
| * 8937fa7 (origin/develop, improve-readme-description, develop) Add techs used to README description
| /
* 80f145c 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 14-6.** The history graph of our project

همانطور که مشاهده می کنید نمودار گزارش تاریخیچه دقیق تری از پروژه ما ارائه می دهد. هر ستاره مانند همیشه نشان دهنده یک تعهد است. اما نوع جدیدی از عناصر در این نمودار نشان داده شده است: شاخه ها. می بینید که ما از شاخه کارشناسی ارشد فاصله گرفتیم و شاخه توسعه را ایجاد کردیم که به نوبه خود به شاخه بهبود برنامه تغییر پیدا کرد. ما دو تعهد را در آن شاخه فشار دادیم و سپس آنرا ادغام کردیم تا پس از آن توسعه یابد ، ما برای ادغام در کار ادغام شدیم.

وقتی روی پروژه ای کار می کنید که از شاخه های زیادی استفاده می کند و غالباً (همانطور که باید) ادغام می شوید، بهتر است از نمای گرافیک به عنوان واضح تر از نمای سنتی استفاده کنید. همچنین، رنگ ها بسیار زیبا هستند.

برای یک سابقه کار بسیار متمیز، پیشنهاد می‌کنم شاخه محل، بهبود-خواندن-توصیف را حذف کنید.

```
$ git branch -D improve-readme-description
```

حذف یک شعبه در حال حاضر ادغام شده خطر کمی را به همراه دارد. اما بسیاری از توسعه دهندگان معمولاً این کار را نمی کنند در صورتی که بعداً دوباره به این کار بپردازند. بیشتر اوقات، این اتفاق نمی افتد. یک قانون خوب این است که شاخه ها را هنگامی حذف کنید که مطمئن باشید برای آزمایش چیزی نیازی به بررسی مجدد آن ندارید.

کاری که ما در اینجا انجام داده ایم ساده ترین شکل ادغام است: یک حرکت سریع. اما به یاد داشته باشید که بعد از اینکه از شاخه ای جدا شدید (مانند کارهایی که ما استادانه انجام داده ایم و توسعه یافته ایم) ، شما در منطقه کاملاً جداگانه قرار دارید. شما از شعب دیگر به روزرسانی نمی کنید ، مگر اینکه آنها را از آنها بخواهید. این همچنین بدان معنی است که سایر شعب به طور مستقل از شعبه شما ارزیابی می کنند. تا زمانی که درخواست برداشت از شعبه را کردید ، ممکن است قبلاً تغییر کرده باشد. به عنوان مثال ، چندین مشارکت کننده می توانند شاخه های جدیدی را ایجاد کرده و درمورد مسائل خود کار کنند. آنها به طور همزمان انجام نمی شوند ، بنابراین هر روابط عمومی یکی پس از دیگری پذیرفته می شوند. اینجا است که مشکل شروع می شود: در حالی که شما روی مسئله خود کار می کنید ، شاخه هدف شما خارج از نفوذ شما تغییر خواهد کرد. واقعاً ، که با آنها کار می کنید ممکن است تا زمانی که با تغییرات خود به پایان رسید تغییر کند. شاید چندین نفر در پرونده

های مشابه خود پرونده های مشابه را تغییر دهند. این اتفاق در حرفه شما بسیار اتفاق می افتد ، و بارها ، روابط عمومی به خوبی مطابق آنچه در این فصل انجام داده ایم ، پیش نمی روند. این مشکلات "تضاد" خوانده می شوند و حل کردن برای سفر Git شما ضروری است. بیایید این کار را انجام دهیم!

## ادغام اختلافات

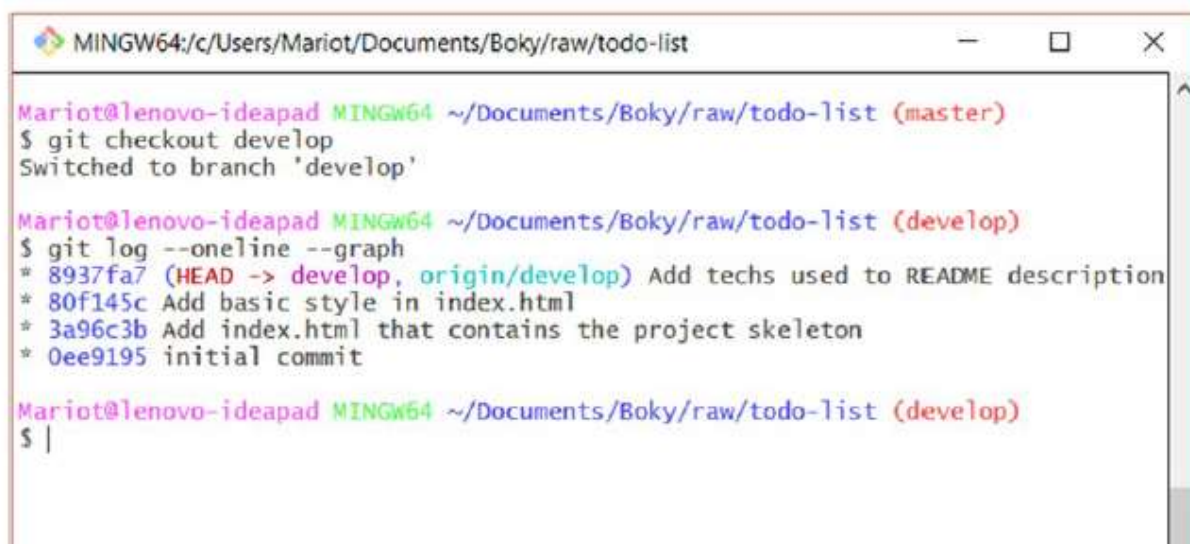
بهترین راه برای درک ادغام اختلافات ایجاد یکی است. بنابراین ، بیایید پروژه ما را خراب کنیم! ابتدا شعبه توسعه محلی ما را بررسی کنید. از آنجا که ما به این شاخه لمس نکرده ایم ، هنوز هم باید تمیز باشد.

```
$ git checkout develop
```

اولین کاری که می خواهیم انجام دهیم بررسی گزارش تاریخ است.

```
$ git log --oneline --graph
```

شما همان نتیجه قبلی را می گیرید زیرا ما هنوز از مبدأ خارج نشده ایم. این نتیجه در شکل 14-7 نشان داده شده است.



```
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 log --oneline --graph
* 8937fa7 (HEAD -> develop, origin/develop) Add techs used to README description
* 80f145c 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 14-7.** Develop history log before pull

هیچ چیز جالب اینجا نیست ، فقط یک ورود قدیمی قدیمی و بدون هیچ مشکلی. از آنجایی که ما شاخه توصیف بهبود-خواندن را حذف کردیم ، هیچ شاخه ای در سابقه تاریخ توسعه باقی نمی ماند.

این گزارش می گوید توسعه و origin/develop در یک حالت قرار دارند. اما این درست نیست زیرا ما آن را از GitHub تغییر دادیم. اما ما به جای بیرون آمدن از مبدأ ، ابتدا می خواهیم در شاخه خود تغییراتی ایجاد کنیم ، تغییراتی که باعث ایجاد تضاد با تغییرات از مبدأ خواهد شد.

index.html را باز کرده و محتوای آن را با کد زیر جایگزین کنید:



```

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

      h3 {
        text-transform: capitalize;
      }
      li {
        overflow: hidden;
        padding: 22px 0;
        border-bottom: 2px solid #eee;
      }
    </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>

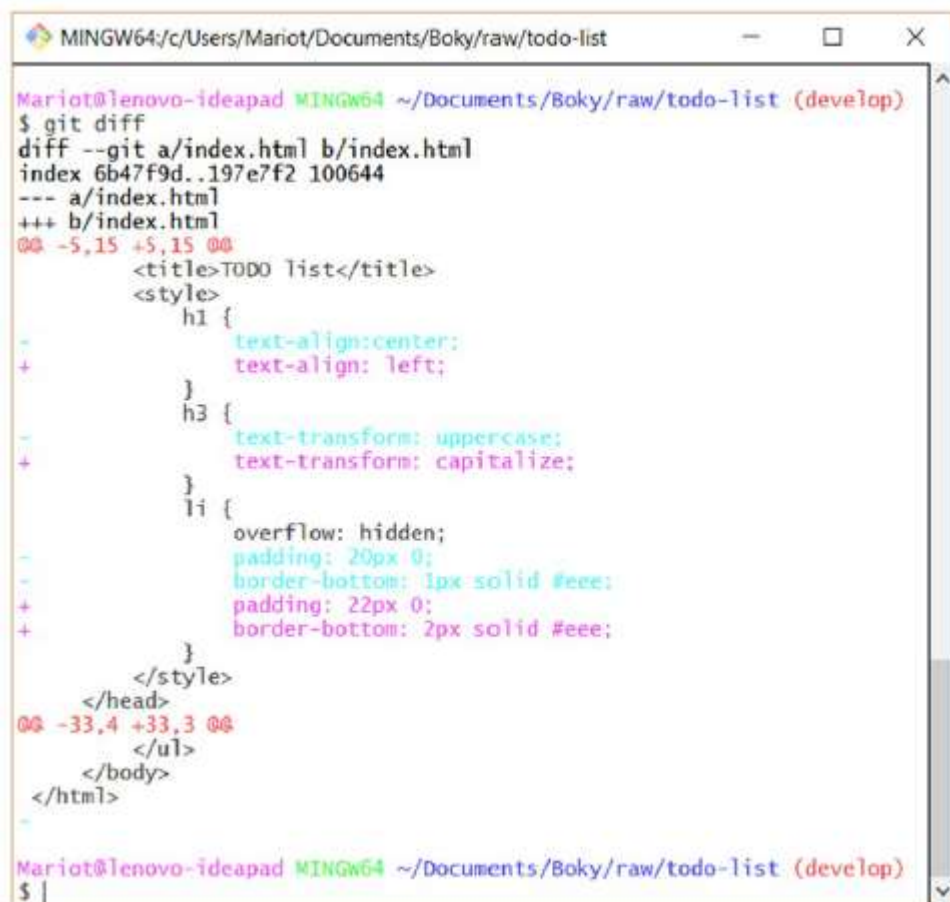
    <h3>Done</h3>
    <ul>
      <li>Put the mittens on the kittens</li>
    </ul>
  </body>
</html>

```

git diff را برای بررسی تغییرات خود اجرا کنید. ما فقط تغییرات کوچک ایجاد کردیم ، بنابراین نباید یک معامله بزرگ باشد ، درست است؟

```
$ git diff
```

نتیجه برای ما بسیار آشناست زیرا ما آن را همیشه در GitHub و در نمایش git می بینیم. نتیجه شما باید مانند من باشد که در شکل 8-14 نشان داده شده است.



```
MINGW64: c:/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git diff
diff --git a/index.html b/index.html
index 6b47f9d..197e7f2 100644
--- a/index.html
+++ b/index.html
@@ -5,15 +5,15 @@
<title>TODO List</title>
<style>
  h1 {
-    text-align: center;
+    text-align: left;
  }
  h3 {
-    text-transform: uppercase;
+    text-transform: capitalize;
  }
  li {
-    overflow: hidden;
-    padding: 20px 0;
-    border-bottom: 1px solid #eee;
+    padding: 22px 0;
+    border-bottom: 2px solid #eee;
  }
</style>
</head>
@@ -33,4 +33,3 @@
</ul>
</body>
</html>
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$
```

**Figure 14-8.** Difference between develop and the working directory

هیچ چیز جدیدی در اینجا نیست. بیاید پرونده تغییر یافته را به قسمت صفحه اضافه کنیم و پروژه فعلی را انجام دهیم.

```
$ git add index.html
```

نکته آیا باز کردن ویرایشگر متن برای هر مرتکب خسته کننده است؟ خوب، اگر عجله دارید می توانید از آن پرش کنید. برای انجام پروژه ضمن رد کردن مرحله انتشار پیام متعهد، می توانید پیام متعهد را به عنوان پارامتر منتقل کنید:

```
$ git commit -m "<commit_message>"
Don't forget the `-m`!
$ git commit -m "Change CSS to introduce conflicts"
```

این نتیجه ای را که قبلاً ندیده ایم به دست می آورد. همانطور که در شکل 9-14 مشاهده می کنید نتیجه استاندارد می گیریم زیرا هنوز درگیری وجود ندارد.

A screenshot of a terminal window titled 'MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list'. The prompt is 'Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)'. The user enters '\$ git commit -m "Change CSS to introduce conflicts"'. The output shows '[develop c5d8f8e] Change CSS to introduce conflicts' and '1 file changed, 4 insertions(+), 5 deletions(-)'. The prompt returns to 'Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)' with '\$ |' on the next line.

```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git commit -m "Change CSS to introduce conflicts"
[develop c5d8f8e] Change CSS to introduce conflicts
1 file changed, 4 insertions(+), 5 deletions(-)
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ |
```

**Figure 14-9.** The commit that will introduce conflicts

برای ایجاد درگیری ، ما باید تعهدی را که برای توسعه در نظر گرفتیم بدست آوریم وقتی شاخه ای را در آن ادغام کردیم.

## Pulling commits from origin

ما قبلاً فرمان pull را در عمل مشاهده کرده ایم ، اما در این سناریو ، کمی مشکل از آن خواهیم داشت: ما همان پرونده را در تعهدات مختلف تغییر دادیم. این باعث ایجاد اختلافات می شود و ما باید قبل از تکمیل کشش ، این موارد را برطرف کنیم. به یاد داشته باشید ، فقط به معنی کپی کردن تعهدات از راه دور در مخزن محلی خود باشید.

بباید با کشیدن مستقیم توسعه از مبدأ شروع کنیم. باز هم ، این دستور بسیار شبیه به فرمان push است. شما فقط به مخزن از راه دور و نام شعبه نیاز دارید.

```
$ git pull origin develop
```

نتیجه ای که می گیریم با همه چیزهایی که قبلاً دیده ایم بسیار متفاوت است. به جای نتیجه یک عمل کامل ، ما یک درگیری گرفتیم و بین دو state گیر افتاده ایم. می توانید شکل 10-14 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 (develop)
$ git pull origin develop
From https://github.com/mtsitoara/todo-list
* branch          develop    -> FETCH_HEAD
  8937fa7..33753ec develop    -> origin/develop
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.

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

**Figure 14-10.** Merge conflict during the pull command

بگذارید این نتیجه را یک به یک باز کنیم. اول ، ما URL را داریم که برای کشیدن استفاده می شود ، بنابراین هیچ چیز جالب اینجا نیست.

بعد ، اولین کاری که توسط Git انجام می شود ، این عمل "واکشی" نام دارد و نقش آن کپی کردن شعبه انتخاب شده از راه دور تا مخزن محلی است. سپس این شاخه در یک ذخیره موقت به نام FETCH\_HEAD ذخیره می شود. دقیقاً مانند HEAD به آخرین تعهدی که ما روی آن کار می کنیم ، FETCH\_HEAD به نوعی شعبه ای که تازه از مبدأ خارج شده ایم ، اشاره دارد. اقدام بعدی یکپارچگی اساسی است ، درست مانند آنچه قبلاً دیده ایم. ما شعبه راه دور را برداشتیم و وقت آن است که آن را با شعبه فعلی ادغام کنیم. عمل جزئیات ادغام را انجام می دهد: شاخه ها توسعه می یابند و منشاء / توسعه می یابند. حتی تعهدی را که استفاده می شود را مشخص می کند. نام تعهدات شما متفاوت خواهد بود ، اما برای تأیید اولین تعهد ، فقط باید بررسی تعهد را بررسی کنید::

```
$ git log --oneline
```

نام commit را در مورد دوم تا آخرین مرتکب همانطور که در شکل 14-11 نشان داده شده است پیدا خواهید کرد.

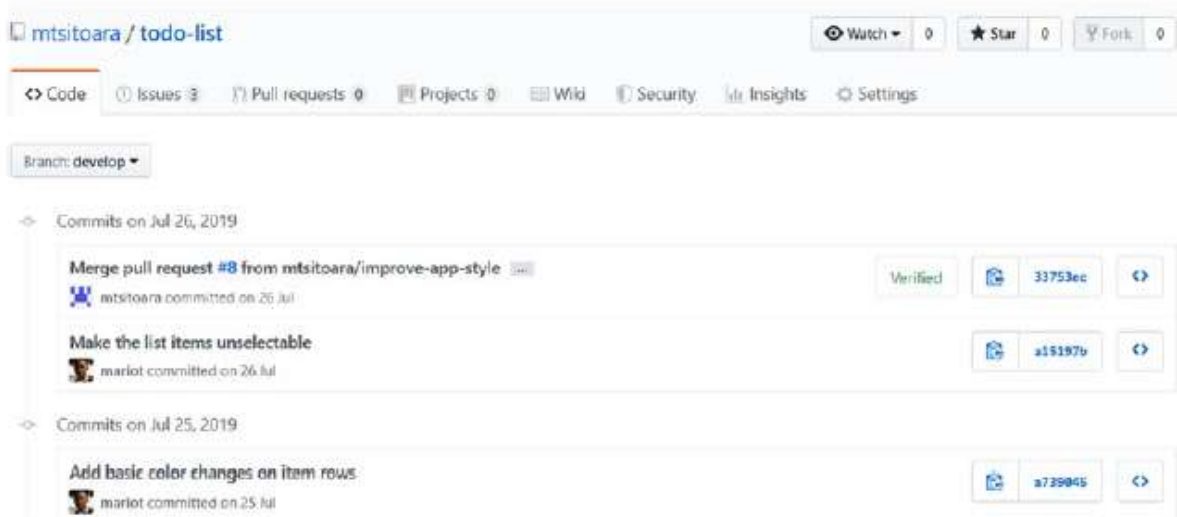
```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop|MERGING)
$ git log --oneline
c5d8f8e (HEAD -> develop) Change CSS to introduce conflicts
8937fa7 Add techs used to README description
80f145c Add basic style in index.html
3a96c3b Add index.html that contains the project skeleton
0ee9195 initial commit

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

**Figure 14-11.** The second to the last commit will be used for the merge

توجه داشته باشید که این ادغام از تعهد گذشته استفاده نخواهد کرد زیرا این تعهدی است که ما روی آن کار می کنیم ، تعهدی که تغییرات را ایجاد کرده است.

شکل 10-14 همچنین به یک تعهد دیگر برای ادغام اشاره دارد و می توانید این تعهد را در مبدأ / توسعه پیدا کنید. به صفحه پروژه خود در GitHub بروید و شاخه توسعه را انتخاب کنید تا گزارش تاریخچه شاخه از راه دور را ببینید. همچنین می توانید به طور مثال با لینک GitHub خود مانند <https://github.com/mtsitoara/todo-list/commits/develop> ، به عنوان مثال به آن دسترسی پیدا کنید. شما مانند گذشته در تصویر 12-14 نمایی از آخرین تعهدات دریافت خواهید کرد.



**Figure 14-12.** The commits on origin/develop

همانطور که مشاهده می کنید ، commit دوم که در شکل 13-10 به آن اشاره شده است آخرین تعهد شاخه از راه دور است ، تعهدی که با ادغام قبلی ما در GitHub ایجاد شده است. برای به دست آوردن اطلاعات بیشتر ، می توانید روی آن کلیک کرده و جزئیات مربوط به این تعهد را دریافت کنید. برای نمونه ای از این گزینه ، شکل 14 را بررسی کنید.



**Figure 14-13.** More info on the merge commit

می توانید در شکل 14-13 مشاهده کنید که این تعهد دارای دو والد است. زیرا این تعهدی است که با ادغام دو شاخه ایجاد شده است. همچنین می توانید ببینید که یکی از والدین در شکل 13-10 نیز ارجاع شده است زیرا این آخرین ارتکابی بود که قبل از اینکه شاخه ها را در GitHub ادغام کنیم تحت فشار قرار گرفته است.

برگردیم به شکل 14-10. در بخش بعدی نتیجه ، گیت تلاش می کند تا شاخه ها را "ادغام خودکار" کند ، به این معنی که سعی کرده است به طور خودکار شاخه ها را ادغام کند. این حالت به آسانی پیش می رود که پرونده های مختلف یا قسمت های مختلف پرونده ها توسط شاخه ها برای ادغام تغییر یافته باشند. اما از آنجا که درگیری پیدا کرد ، ادغام شکست خورده است. و حل ما بر عهده ماست.

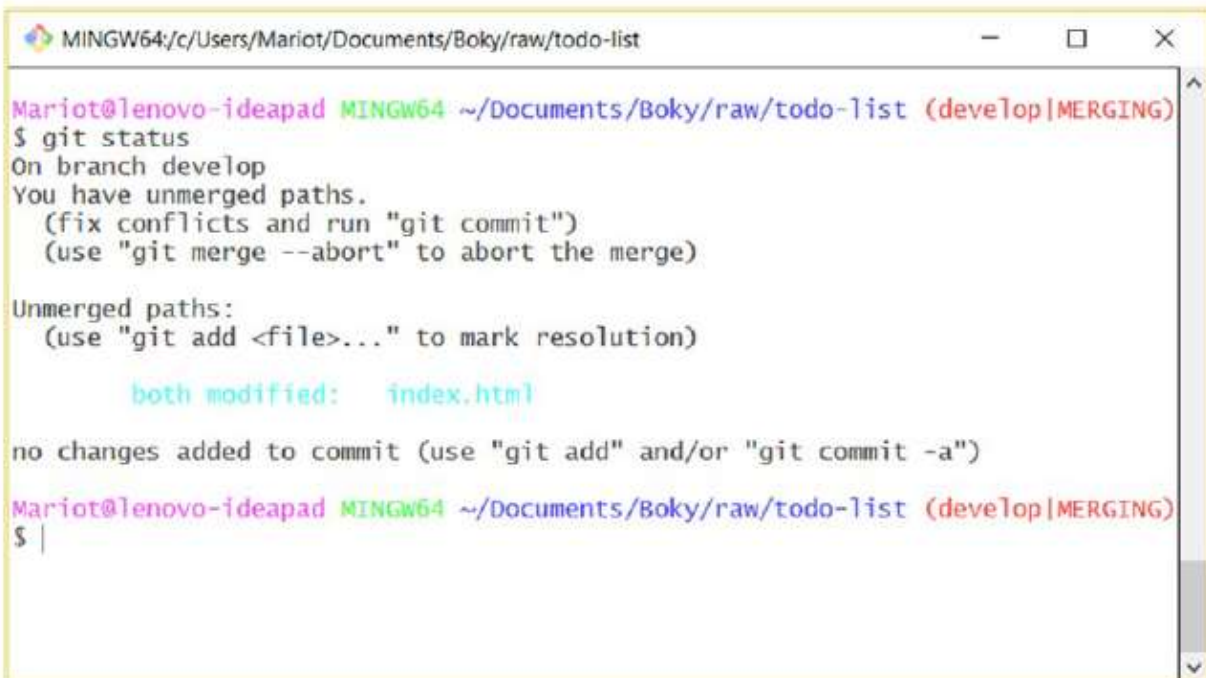
Git سعی کرد تا شاخه توسعه محلی ما را با FETCH\_HEAD ادغام کند ، اما از آنجا که هر دو شاخه شامل تغییراتی در همان قسمت های `indind.html` هستند ، باید تصمیم بگیرید که کدام تغییرات را باید حفظ کنید. در بخش بعدی خواهیم دید که چگونه این کار را انجام دهیم.

آخرین اطلاعاتی که باید از شکل 14-10 به آن توجه داشته باشید وضعیتی است که مخزن محلی ما در آن قرار دارد. اگر به قسمت سمت چپ کنسول نگاه کنید ، می فهمید که مخزن در حالت "توسعه | ادغام" به جای شاخه "توسعه" استاندارد. این بدان معناست که هنوز اختلافات حل نشده در پروژه وجود دارد و ادغام (و با تمديد ، کشش) هنوز انجام نشده است. برای کسب اطلاعات بیشتر در مورد وضعیت فعلی مخزن می توانید وضعیت را بررسی کنید.

```
$ git status
```

این نتیجه جدیدی به شما می دهد که قبلاً آن را ندیده ایم ، که در شکل 14-14 نشان داده شده است.





```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop|MERGING)
$ git status
On branch develop
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop|MERGING)
$ |
```

**Figure 14-14. Status of the merge**

این نتیجه قابل خواندن بسیار آسان است و مشاوره خوبی برای مراحل بعدی ارائه می دهد. اول، کارهایی را که باید انجام دهیم بعد به ما می گوید: درگیری ها را حل کنیم و پروژه را مرتکب شویم. در بسیاری از مواقع، این ایده خوبی است زیرا می توانیم در شعبه محلی برای حل و فصل مناقشاتی که می دانیم بوجود می آیند کار کنیم. به عنوان مثال، ما می توانیم این ادغام را قطع کنیم، تعهدی را که درگیری ایجاد کرده است برگردانیم و دوباره بکشیم. پس از آن ما بدون درگیری با یکدیگر ادغام خواهیم کرد. اما این برای ما بسیار آسان و معقول است، بنابراین، بیایید این کار را به سختی انجام دهیم!

بعد، ما لیستی از پرونده های مربوط به ادغام را در اختیار داریم. در اینجا، `onlyindex.html` مربوط می شود و در هر دو شاخه اصلاح شده است. بیایید آن را باز کنیم تا شاهد درگیری ها باشیم.

شما تغییرات بزرگی را در آن مشاهده خواهید کرد، همانطور که در شکل 14-15 و شکل 14-16 نشان داده شده است.

```
index.html x
C:\Users\Markit\Documents\Boky\raw\todo-list> index.html >...
11     text-transform: capitalize;
12   }
13   Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
14   <<<<<<< HEAD (Current Change)
15     li {
16       overflow: hidden;
17       padding: 22px 0;
18       border-bottom: 2px solid #eee;
19     }
20     ul {
21       margin: 0;
22       padding: 0;
23     }
24     ul li {
25       cursor: pointer;
26       position: relative;
27       padding: 12px 8px 12px 40px;
28       background: #eee;
29       font-size: 18px;
30       transition: 0.2s;
31       -webkit-user-select: none;
32       -moz-user-select: none;
33       -ms-user-select: none;
34       user-select: none;
35     }
36     ul li:nth-child(odd) {
37       background: #f9f9f9;
38     }
39     ul li:hover {
40       background: #ddd;
41     }
42   >>>>>>> 33753ecaebae2ba1c3ffdc1e543d372385884c78 (Incoming Change)
43   </style>
44 </head>
```

**Figure 14-15.** *index.html* in Visual Studio Code



آن مناطق با خط "=====" جدا می شوند، که کد را از دو شاخه نشان می دهد. بخش اول کدی است که در شعبه فعلی خود دارید. بخش دوم کد مربوط به شاخه ای است که می خواهید ادغام کنید.

بنابراین ، ما دو پرونده متناقض در پرونده خود داریم. اول توسعه کد و دوم کد مربوط به مبدا / توسعه است. برای حل و فصل اختلاف ادغام ، ما باید پرونده را ویرایش کنیم زیرا فقط یک تغییر دارد. این بدان معنا نیست که شما باید بین دو تغییر انتخاب کنید ، این بدان معنی است که در پایان فقط یک نفر باقی می ماند ؛ در صورت لزوم می توانید آنها را ادغام کنید.

در مورد ما ، بهتر است بیشتر قسمت دوم را حفظ کنیم زیرا ما قبلاً آن تغییرات را بررسی کرده ایم و پذیرفته ایم. اما همچنین مواردی وجود دارد که می توانیم از قسمت اول آن را حفظ کنیم. بنابراین ، بهترین دوره عمل ، کپی کردن کد مورد نیاز ما از اسایرتان های اول کپی کردن آن در قسمت دوم است. کد سپس تبدیل می شود

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

```

        h3 {
            text-transform: capitalize;
        }
<<<<<< HEAD
        li {
            overflow: hidden;
            padding: 22px 0;
            border-bottom: 2px solid #eee;

=====
        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;
            overflow: hidden;
        }
        ul li:nth-child(odd) {
            background: #f9f9f9;
        }
        ul li:hover {
            background: #ddd;
>>>>>> 33753ecaebae2ba1c3ffdc1e543d372385884c78
        }
    </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>

```

همانطور که می بینید ، ما فقط از یک قسمت یک خط را کپی کرده ایم ، زیرا قسمت دوم تقریباً کامل شده است. اکنون زمان تمیز کردن پرونده مربوط به قسمت غیر ضروری است. ابتدا می توانیم قسمت اول تعارض کد را حذف کنیم (بین < /> >> /> i> و << /> ) زیرا دیگر به آنها احتیاجی نداریم. بنابراین ما می توانیم خط باقی مانده را حذف کنیم (<<<<<<< ) زیرا معنایی ندارد که دیگر داشته باشیم. پرونده سپس تبدیل می شود

```

<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>TODO list</title>
    <style>
      h1 {
        text-align: left;
      }
      h3 {
        text-transform: capitalize;
      }
      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;
        overflow: hidden;
    }
    ul li:nth-child(odd) {
        background: #f9f9f9;
    }
    ul li:hover {
        background: #ddd;
    }
</style>
</head>

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

```

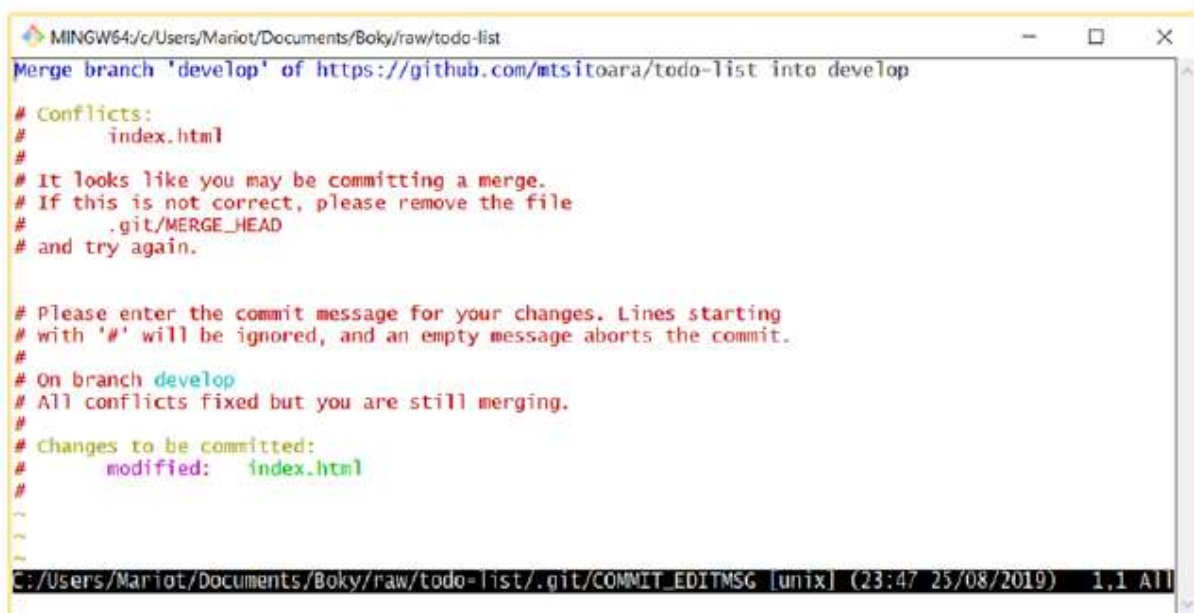
پرونده به حالت عادی برگشته است! با ادغام کدهای متناقض و بیشتر از آن سه خط بزرگ. اکنون می توانید روند ادغام را ادامه دهید. اگر مرحله بعد را فراموش کردید ، می توانید وضعیت git را دوباره اجرا کنید (با شکل 13-14 را بررسی کنید). بنابراین ، اکنون که پرونده آماده است ، باید آنرا مرحله بندی کنیم.

```
$ git add index.html
```

پس از آن ، شما باید طبق معمول این پروژه را انجام دهید.

```
$ git commit
```

با دیدن پیام پیام متعهد آشنا خواهید شد اما با کمی پیچ و خم: پیام متعهد از قبل نوشته خواهد شد. شکل 14-17 را برای نمونه ای از این موضوع بررسی کنید.



```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Merge branch 'develop' of https://github.com/mtsitoara/todo-list into develop

# Conflicts:
#   index.html
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#   .git/MERGE_HEAD
# and try again.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch develop
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#   modified:   index.html
#
~
~
C:/Users/Mariot/Documents/Boky/raw/todo-list/.git/COMMIT_EDITMSG [unix] (23:47 25/08/2019) 1,1 All
```

**Figure 14-17.** The default commit message

البته ، شما همیشه می توانید پیام متعهد را تغییر دهید ، اما من پیشنهاد می کنم پیش فرض را ترک کنید مگر اینکه شما یک راهنمایی شخصی یا شرکتی را دنبال کنید. می توانید پیام متعهد را ذخیره کنید و حرکت کنید. اگر به نتیجه دستور (نگاه کنید به شکل 13-18) نگاه می کنید ، خواهید دید که دوباره به شاخه توسعه باز می گردید و دیگر در حالت "ادغام" نیستید.



```
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop|MERGING)
$ git add index.html

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop|MERGING)
$ git commit
[develop d116e1b] Merge branch 'develop' of https://github.com/mtsitoara/todo-list into develop

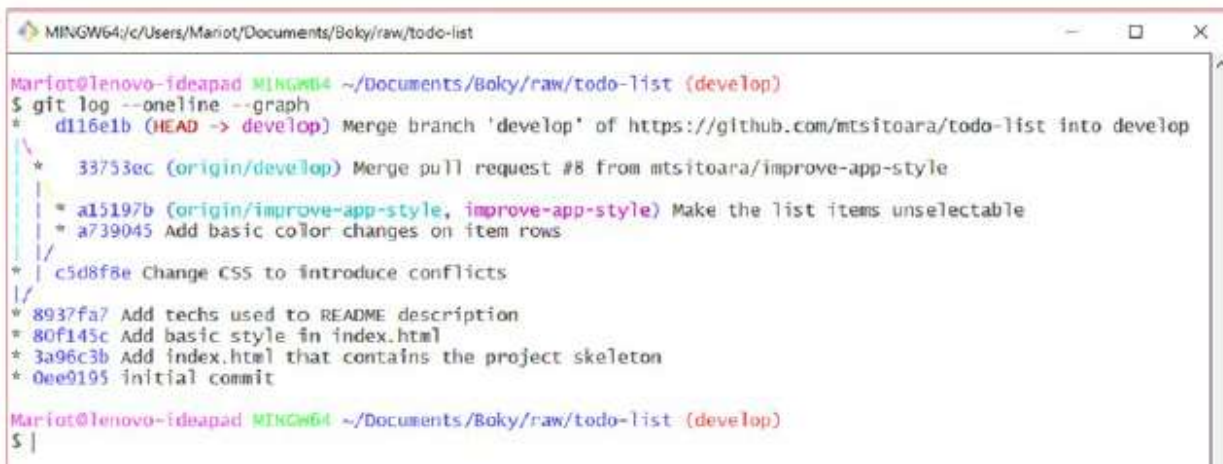
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ |
```

**Figure 14-18.** Back to normal state

همچنین می توانید با بررسی گزارش تاریخ ، بررسی کنید که ادغام به پایان رسیده است یا خیر. اطمینان حاصل کنید که برای نتیجه زیبا گزینه گرافیکی اضافه کنید.

```
$ git log --oneline --graph
```

این یک تصویری خیره کننده را نشان می دهد که در شکل 14-19 نمایش داده شده است.



```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git log --oneline --graph
* d116e1b (HEAD -> develop) Merge branch 'develop' of https://github.com/mtsitoara/todo-list into develop
* 33753ec (origin/develop) Merge pull request #8 from mtsitoara/improve-app-style
* a15197b (origin/improve-app-style, improve-app-style) Make the list items unselectable
* a739045 Add basic color changes on item rows
* c5d8f8e Change CSS to introduce conflicts
* 8937fa7 Add techs used to README description
* 80f145c 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 14-19.** The recent history of our project

می توانید بر روی آن نمودار مشاهده کنید که وقتی شاخه مبدا / توسعه را ادغام می کنیم ، تمام تاریخچه آن را وارد می کنیم. بنابراین ، به نظر می رسد که ما یک شاخه از یک شعبه داریم. در پروژه های بزرگ گیت ، تمام وقت اتفاق می افتد.

## خلاصه

این بزرگترین فصل کتاب است. تبریک می گویم برای رسیدن به آنجا! ما دیدیم که چگونه می توان کد را از یک سرور از راه دور بیرون کشید و چگونگی حل تعارض را هنگامی که همان منطقه کد توسط دو شاخه مختلف اصلاح شده است ، حل می کنیم.

راه اصلی اصلی در مورد کشیدن این است که در واقع دو دستور یکی پس از دیگری اجرا می شوند:

- واکنشی ، که شاخه از راه دور را در یک شعبه موقت کپی می کند
- ادغام ، که شاخه موقت را به یکی از شاخه های فعلی ادغام می کند ، اما هنگامی که دو شاخه دارای یک کد هستند ، درگیری با هم اختلاف می کند. برای حل این درگیری ها ، باید پرونده مربوطه را مجدداً باز کنید و تصمیم بگیرید کد را نگه دارید. سپس ، بقیه اساسی است: مرحله بندی و انجام.

درگیری های ادغام یکی از مواردی است که آزار دهنده است اما متأسفانه در حرفه شما اتفاق می افتد ، بنابراین یادگیری چیزهای زیادی در مورد آنها مهم است. و از آنجا که آنها آزار دهنده هستند ،

ما می خواهیم در فصل بعد نحوه کاهش ظاهر آنها را بیاموزیم.

# فصل پانزدهم

## اطلاعات بیشتر درباره

# Conflicts

فصل آخر شدید بود ، مگه نه؟ ما در مورد آنچه درگیری های ادغام هستند صحبت می کنیم و چه موقع اتفاق می افتد. ما همچنین دیدیم که چگونه آنها را به صورت دستی حل کنیم. نگران نباشید ، این فصل هضم بسیار ساده تر خواهد بود. ما می خواهیم در مورد چگونگی سوق دادن شاخه خود به ریموت پس از درگیری ادغام صحبت کنیم. همچنین ، ما می خواهیم استراتژی هایی را برای اتخاذ برای کاهش تعداد درگیری هایی که ممکن است رخ دهد ، ببینیم. بیا بریم!

### Pushing after a conflict resolution

همانطور که در فصل های قبلی دیدیم ، فشار دادن به معنی کپی کردن تعهدات محلی ما به یک شعبه از راه دور است. این بدان معناست که هر تعهدی که در محلی داشته باشیم در مخزن راه دور اعمال می شود.

ما در قسمت آخر دیدیم که یک عمل کششی فقط دو عملی است که یکی پس از دیگری انجام می شود: یک عمل واکشی است که شاخه از راه دور را در یک مکان موقت کپی می کند و یکپارچه سازی را انجام می دهد که شاخه موقت را به محلی متصل می کند و از زمان کشیدن و فشار اقدامات فقط یکسان هستند اما در جهات مختلف ، به همان روش کار می کند تا شعبه محلی شما را به سمت منشأ سوق دهد.

بنابراین یک عمل فشار نیز به دو بخش تقسیم می شود: یک کپی از شعبه محلی شما به ریموت و ادغام شاخه ها. تنها تفاوت بین اعمال فشار و کشش فقط موضوعی است که بازیگر عملکرد را انجام می دهد: شما یا سرور.

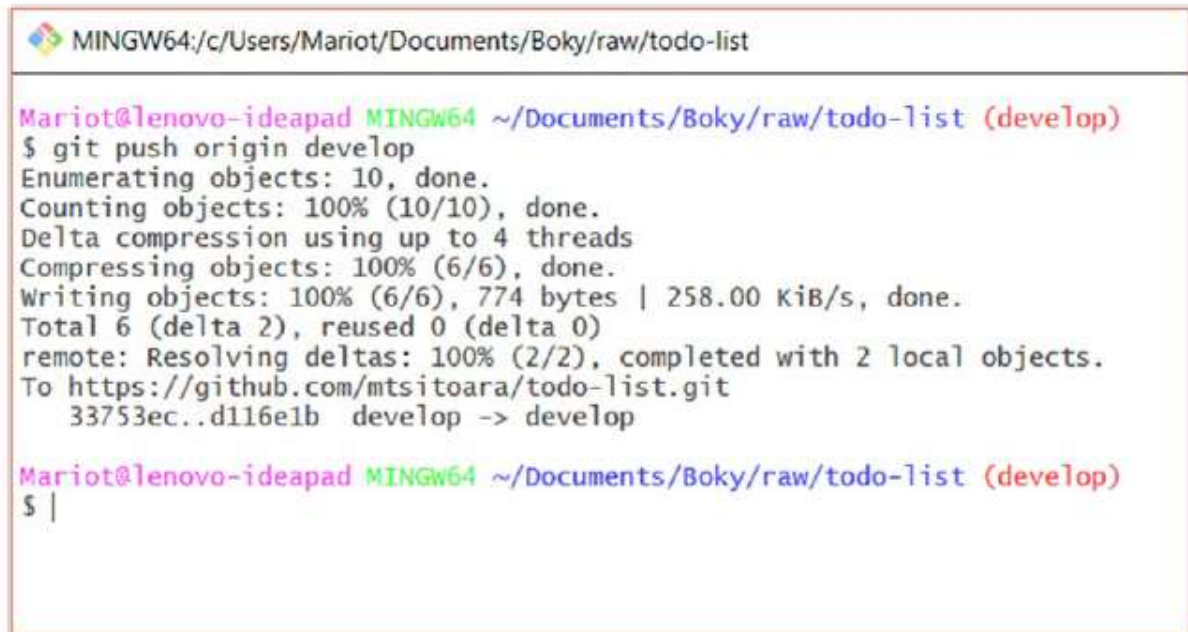
در شرایط عادی ، فشار به آسانی پیش می رود زیرا ادغام به طور خودکار با استفاده از "سریع جلو" انجام می شود. وقتی تعهدات مربوط به شعب محلی شما می تواند مستقیماً با تعهدات موجود در شعبه از راه دور در ارتباط باشد ، حرکت سریع به جلو امکان پذیر است.

به عنوان مثال ، اضافه کردن تعهدات یکی پس از دیگری بر روی شاخه کارشناسی ارشد ما (مانند آنچه تاکنون انجام داده ایم) و سپس فشار آوردن آنها منجر به ادغام سریع می شود ، نیازی به ایجاد یک ادغام نیست.

در شرایط ما ، این اتفاق خواهد افتاد و همچنین ما فقط به شعبه توسعه خود تعهدات جدیدی افزودیم. و ما مشکلی نخواهیم داشت مگر اینکه ما یا شخص دیگری در گذشته برویم و تاریخ را تغییر دهیم. هرگز سعی نکنید این کار را انجام دهید. گفته شد ، اجازه دهید با استفاده از دستور معمول ، شاخه توسعه ما را تحت فشار قرار دهیم.

```
$ git push origin develop
```

همانطور که انتظار می رود ، نتیجه معمول را در شکل 1-15 نشان خواهیم داد.



```
MINGW64:/c:/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (develop)
$ git push origin develop
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 774 bytes | 258.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/mtsitoara/todo-list.git
    33753ec..d116e1b  develop -> develop

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

**Figure 15-1. Pushing our develop branch**

در پایان ، پس از کشیدن و ادغام تغییرات ، شاخه بازگشت به مبدأ نباید منجر به رفتار غیر منتظره ای شود. مگر اینکه کسی تاریخ را تغییر دهد.

## تغییرات را قبل از ادغام بررسی کنید

قبل از تلاش برای ادغام ، مهمترین کاری که شما می توانید انجام دهید بررسی تمام تغییراتی است که شعبه شما معرفی خواهد کرد. این یک قدم اساسی است که نباید از آن چشم پوشی کرد زیرا باعث می شود ساعت های بیشمار نبرد با Git صرفه جویی شود.

## محل شعبه را بررسی کنید

اولین موردی که باید مطمئن شوید موقعیت مکانی شماست. برای ادغام دو شاخه با هم ، باید شعبه مورد نظر را بررسی کنید. به عنوان مثال ، اگر شما قصد دارید ادغام را به استاد تبدیل کنید ، باید ابتدا دومی را بررسی کنید بنابراین کد خواهد بود (در حال حاضر دستور دوم را اجرا نمی کنید):

```
$ git checkout master
```

```
$ git merge develop
```

## بررسی شاخه-Diff

بررسی تفاوت فقط برای تعهدات محفوظ نیست! همچنین می توانید از آن برای بررسی اختلافات بین دو شاخه استفاده کنید که در شرایط ظریف مانند ادغام بسیار مفید است. دستور نسبتاً ساده است:

```
$ git diff branch1..branch2
```

به دو نقطه بین دو نام شاخه توجه کنید. این اختلافات بین دو شاخه را از نظر نمای آشنا نشان می دهد. بیا باید توسعه را با استاد مقایسه کنیم:

```
$ git diff master..develop
```

نتیجه در مقایسه با تعهدات بسیار شبیه به نتایج متفاوت ما است. شکل 15-2 را برای چنین نمونه ای بررسی کنید.



```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (master)
$ git diff master..develop
diff --git a/index.html b/index.html
index 2d27723..391ef94 100644
--- a/index.html
+++ b/index.html
@@ -5,10 +5,10 @@
<title>TODO list</title>
<style>
  h1 {
-    text-align: center;
+    text-align: left;
  }
  h3 {
-    text-transform: uppercase;
+    text-transform: capitalize;
  }
  ul {
    margin: 0;
@@ -25,6 +25,7 @@
-moz-user-select: none;
-ms-user-select: none;
user-select: none;
+overflow: hidden;
  }
  ul li:nth-child(odd) {
    background: #f9f9f9;
@@ -50,4 +51,3 @@
</ul>
</body>
</html>
```

**Figure 15-2.** Differences between branches

اگر تغییرات زیادی ایجاد کرده اید و نمی خواهید خیلی دور شوید ، می توانید آن تغییرات را در GitHub مشاهده کنید. فقط شعبه را فشار داده و یک درخواست Pull را باز کنید!

ادغام را درک کنید



ما قبلاً مفاهیم زیادی را در مورد Git Merges مشاهده کرده ایم ، اما اجازه می دهیم آنها را مرور کنیم تا دید واضح تری از این ویژگی دریافت کنیم. همانطور که قبلاً دیدیم ، ادغام عبارت است از ترکیب دو شاخه یا به عبارت دقیق تر ریختن شاخه به شاخه دیگر. از هر شاخه دیگری می توان شاخه هایی تشکیل داد و وقتی شاخه ای ایجاد شد ، از والدین خود مستقل می شود. تغییرات زمان انجام شده برای هر یک از شاخه ها روی دیگری تأثیر نمی گذارد. بیایید شرایطی را تصور کنیم که یک شاخه کودک ایجاد می کنید و در آن شاخه جدید متعهد می شوید. وقتی زمان ادغام فرا رسید ، چندین موقعیت ممکن است به وجود آید. اگر شاخه اصلی تغییر نکرده باشد (هیچ تعهدی انجام نشده است) و شما سعی در ادغام دارید ، یک ادغام "سریع به جلو" رخ خواهد داد. یک ادغام "سریع به جلو" از لحاظ فنی ادغام شده اما فقط یک تغییر مرجع در Git است. به یاد داشته باشید که تعهدات Git مانند لیست های زنجیر شده رفتار می کنند ، به این معنی که یک تعهد شامل یک اشاره به موارد قبلی است. در واقع ، اگر والدین تغییر نکرده اند ، Git فقط مرجع را به سمت والدین (به دنبال لیست زنجیر شده) به جلو هدایت می کند و آخرین تعهد در شاخه کودک آخرین تعهد شعبه والدین می شود. به بیان ساده تر ، Git تعهدات موجود در شاخه کودک را به شاخه والدین اضافه می کند. این ساده ترین نوع "ادغام" بلکه غیر معمول ترین است ، مگر اینکه به تنهایی کار کنید.

## کاهش درگیری

ما آخرین فصل را دیدیم که حل اختلافات می تواند دردناک باشد و همچنین بسته به اندازه آنها می تواند زمان زیادی را ببرد. بنابراین ، برای ما سودمند خواهد بود که ظاهر آنها را به حداقل برسانیم. ما در این بخش قصد داریم استراتژی های اتخاذ برای محدود کردن درگیری ها را ببینیم.

داشتن یک گردش کار خوب

در صورت استفاده از یک گردش کار خوب ، بسیاری از مشکلاتی که در Git و GitHub با آنها روبرو خواهید شد ، قابل جلوگیری است. ما قبلاً در فصل های گذشته رایج ترین گردش کار Git را مشاهده کرده ایم اما اجازه دهید دوباره آن را مرور کنیم.

اولین چیزی که باید بخاطر بسپارید تعهد مستقیم به شعب اصلی شما ندارید. به عبارت ساده تر: هر تغییری که قصد دارید به استاد خود وارد شوید یا شاخه هایی را توسعه دهید باید با ادغام انجام شود. و هر یک از ادغام ها باید توسط Pull Request معرفی شوند.

به این ترتیب ، هنگام کار روی آن می توانید بازخورد خود را دریافت کنید. همچنین این روش به آزمایش کنندگان امکان پیگیری تغییرات پروژه را می دهد. شما باید همیشه از PR استفاده کنید تا حتی اگر به تنهایی کار کنید ، در شعب اصلی تغییری ایجاد کند. این کار باعث می شود سابقه تاریخی بسیار واضح تر و تمیزتری نسبت به پیام های متعهد ساده انجام شود.

هر درخواست Pull باید حل مسئله را به عنوان یک هدف داشته باشد. بنابراین ، روابط عمومی باید فقط یک کار را انجام دهد ، چه رفع اشکال ، یک پیشنهاد ویژگی یا تغییرات مستندات. وسوسه نشوید چندین مسئله را با یک روابط عمومی حل کنید. Do-it-all Pull Request دستور العمل های مناسب برای ادغام درگیری ها هستند. یک مورد که غالباً توسط توسعه دهندگان نادیده گرفته می شود ، پایان بخش خط و قالب بندی پرونده است. همانطور که در فصل 3 دیدیم ، سیستم عامل های مختلف از انتهای خط مختلف استفاده می کنند. لازم است تیم شما در مورد اینکه کدام یک از آنها را برای هر پروژه استفاده می کند ، بحث کند. اکثر تیم ها از انتهای خط خطی یونیکس استفاده می کنند که کاربران ویندوز باید مشتری Git خود را بر این اساس پیکربندی کنند. در مورد قالب بندی ، این به تیم شما بستگی دارد ، اما تنها قانون این است که همه شما باید برای یک تورفتگی و بازده خط از یک قالب یکسان استفاده کنید.

## سقط یک ادغام

بسیاری از درگیری های ادغام شما ناشی از برخورد کد نیستند. بسیاری از آنها از تفاوت های قالب بندی و فضای سفید ناشی می شوند. به عنوان مثال ، یک فضای برگشت ناگهانی یا تعداد فاصله های دنداندار می تواند اختلافاتی را ایجاد کند حتی اگر کد تغییر نکرده باشد.

هنگامی که با این نوع درگیری ها روبرو شد ، بهترین حرکت فقط برای متوقف کردن ادغام ، عقب انداختن اختلافات قالب بندی خود است و سپس دوباره سعی کنید ادغام شوید. همانطور که قبلاً دیدید ، دستور سقط یک ادغام است

```
$ git merge --abort
```

این هیچ یک از تعهدات شما را از بین نمی برد ، این ادغام را فقط لغو می کند و شما در وضعیت فعلی خود خواهید ماند.

## استفاده از یک ابزار Git visual

هنگام استفاده از یک ویرایشگر متن ساده ، حل و فصل یک درگیری ممکن است دشوار باشد زیرا بیشتر اوقات ، این باعث می شود که طرح رنگ کد را خراب کند. یک راه حل ساده برای آن استفاده از ابزارهای تخصصی برای Git است. آنها می توانند پسوند IDE یا ابزاری مخصوص برای Git باشند.

## خلاصه

این فصل یک یادآوری ساده درباره ادغام ها و نحوه استفاده آنها بود. ما انواع مختلف ادغام Git و موقعیت هایی را که می توانند در آنها ظاهر شوند ، دیدیم. ما همچنین بررسی کردیم که چگونه ادغام کار می کند و هدف چیست: تعهدات را از یک شاخه به شاخه دیگر بریزید.

موارد اصلی برای به خاطر سپردن روش های مختلفی برای کاهش درگیری ادغام است. شما ممکن است هرگز از شر آنها خلاص نشوید ، اما پیروی از این دستگاه ها ظاهر آنها را به حداقل می رساند.

ما در سفر Git پیشرفت های زیادی داشته ایم ، اما این کار را با استفاده از کنسول های ساده و خسته کننده انجام داده ایم. وقت آن است که رنگ بیشتری را در پروژه های Git ما قرار دهیم ، بنابراین بیایید در مورد GUI اطلاعات کسب کنیم!

# فصل شانزدهم

## ابزار Git GUI

در فصل های قبل ، ما بسیاری از مهمترین Git features و مفاهیم را دیده ایم. ما در مورد تعهدات ، شاخه ها ، درخواست های کشش و ادغام یاد گرفته ایم. با استفاده از این مفاهیم ، شما در حال حاضر می توانید تقریباً هر چیزی را در Git انجام دهید. تنها یک مشکل کوچک: فقط از پنجره ترمینال یا کنسول استفاده کرده ایم. در این فصل ، شما هیچ مفهوم یا ویژگی جدیدی را یاد نخواهید گرفت ، شما فقط یاد می گیرید که چگونه می توانید آنچه را که قبلاً با سبک می شناسید بکار بگیرید. ابتدا می خواهیم ابزارهای پیش فرض موجود با Git را مورد بررسی قرار دهیم ، سپس در مورد IDE هایی که ادغام Git دارند بیشتر بدانید. و در آخر به برخی از ابزارهای تخصصی که مخصوص Git ساخته شده اند نگاهی بیندازید.

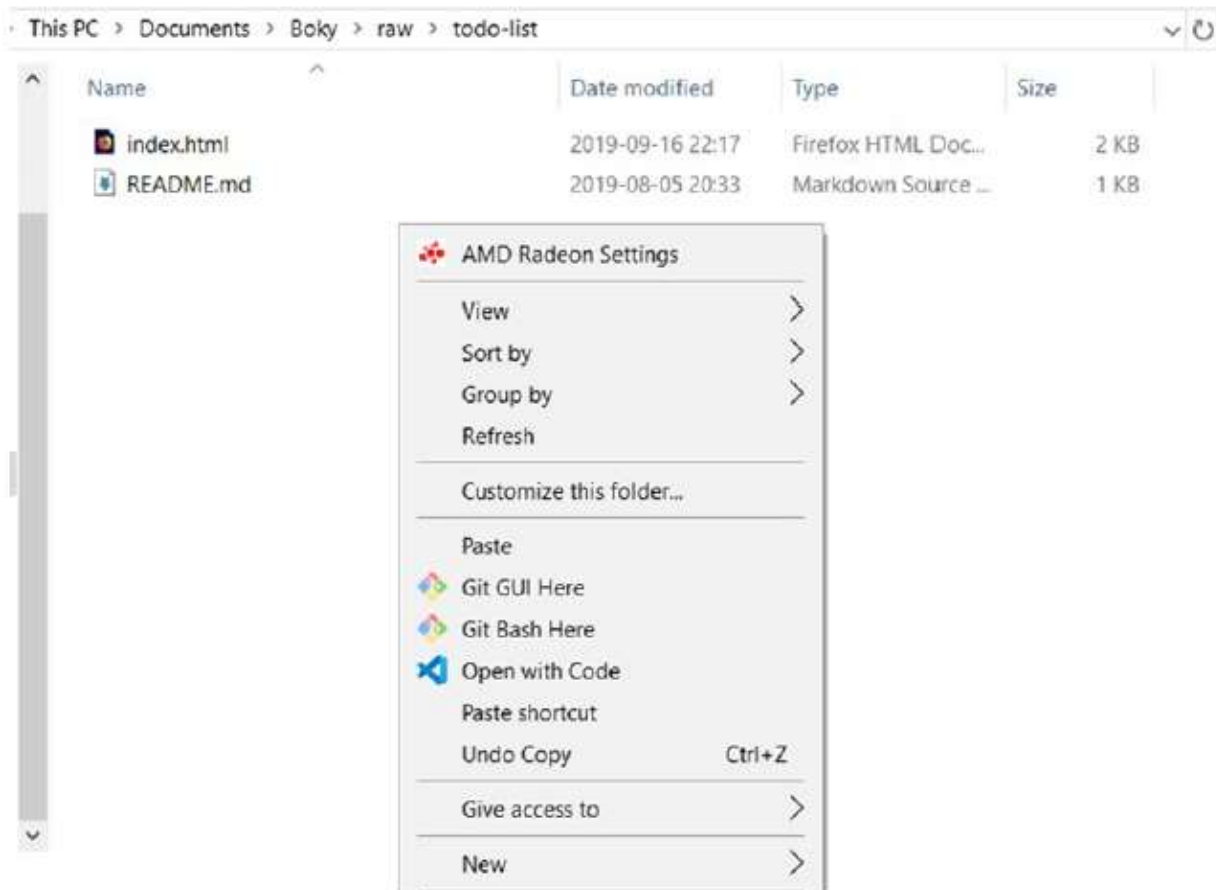
### ابزارهای پیش فرض

اگر مراحل نصب را از فصل 3 دنبال کرده اید ، در حال حاضر آن ابزارها را روی رایانه خود نصب کرده اید. اگر نه ، می توانید به راحتی آنها را در فروشگاه نرم افزاری معمولی ما قرار دهید. این ابزارهای پیش فرض با Git ارسال می شوند تا رابط کاربری گرافیکی بسیار ساده ای را در اختیار کاربران قرار دهند تا مخازن خود را مرور کنند و تعهدات خود را تهیه کنند. آنها تقریباً برای هر سیستم عامل موجود هستند ، بنابراین نگران نباشید ، آنها در دسترس شما هستند. آنها به دلایل تاریخی و به دلیل ساخت داخل گیت در این کتاب ارائه شده اند.

### تعهد: git-GUI

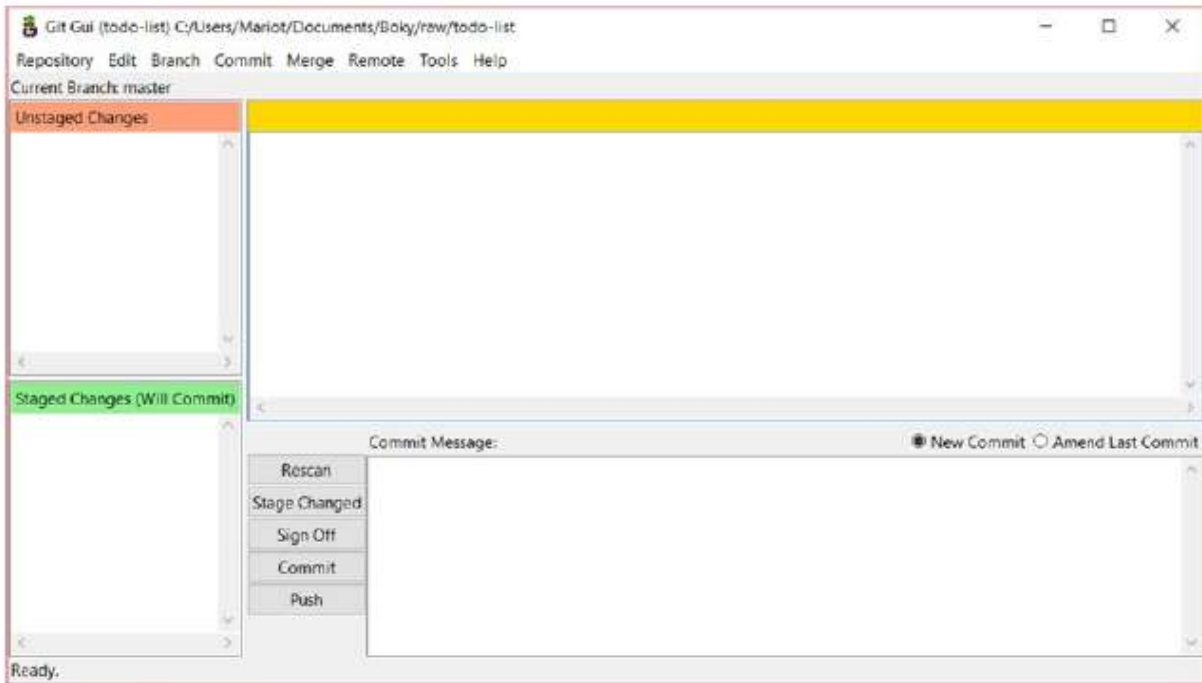
اولین ابزاری که می خواهیم ببینیم با نام git-GUI است و یک رابط تعهد گرافیکی برای Git است. شما از آن برای انجام پروژه خود و بررسی تغییرات پیشنهادی استفاده خواهید کرد. می توانید اطلاعات بیشتر در مورد آن را در <https://git-scm.com/docs/git-gui> پیدا کنید.

می توانید همانطور که گیت Bash را باز می کنید ، مانند خط فرمان ، فهرست زمینه یا صفحه شروع ، آن را باز کنید. انتخاب کنید هر کدام بهترین گزینه برای شما باشد. در ویندوز و سیستم عامل های مبتنی بر دبیان ، می توانید با حرکت به دایرکتوری مخزن و با کلیک راست روی یک فضای خالی ، یک Git GUI را باز کنید. انجام این کار نتیجه ای مشابه شکل 1-16 به شما می دهد.



**Figure 16-1.** Windows context menu

همانطور که مشاهده می کنید ، می توانید Git GUI و Git Bash را در آنجا باز کنید. پیش بروید و Git GUI را انتخاب کنید. یک پنجره برنامه کوچک دریافت خواهید کرد که جزئیات وضعیت فهرست فعلی کار شما را نشان می دهد. پنجره در شکل 16-2 ارائه شده است.



**Figure 16-2. Git GUI interface**

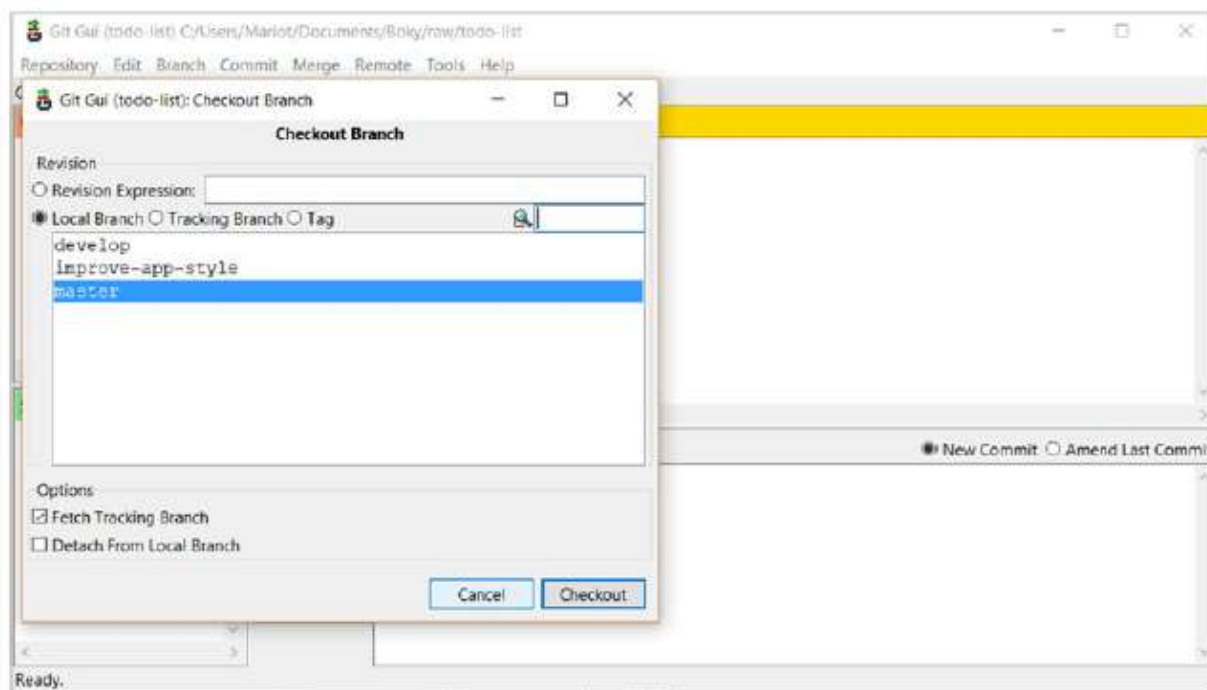
و اگر نمی خواهید از فهرست زمینه استفاده کنید یا نمی توانید ، با باز کردن یک ترمینال در محل مخزن Git خود و اجرای دستور زیر می توانید آن را باز کنید:

```
$ git gui
```

رابط کاربری Git GUI بسیار سبک و شهودی است. و برای هر سیستم عامل یکسان است بنابراین همه در خانه احساس می کنند. به چهار بخش تقسیم می شود:

- بالا سمت چپ لیستی از پرونده های ویرایش شده است که هنوز مرحله بندی نشده اند.
- پایین سمت چپ لیستی از پرونده های مرحله بندی شده است.
- بالا سمت راست نمای متفاوت است.
- پایین سمت راست یک متن متن پیام متعهد است.

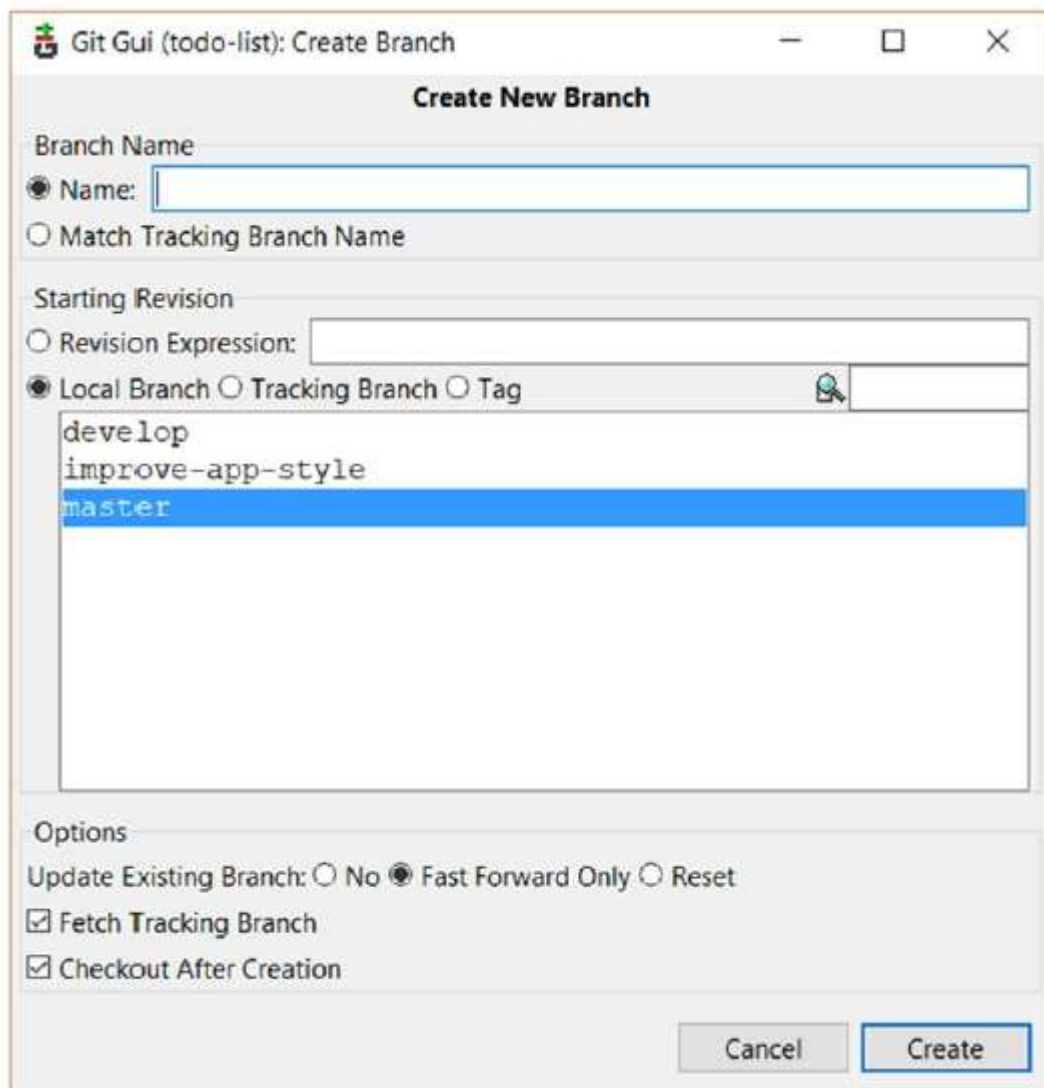
و از آنجا که ما در پروژه خود چیزی را تغییر نداده ایم همه چیز خالی است. بنابراین ، بایید پروژه خود را با تعهدات اضافی اشتباه بگیریم. ابتدا مطمئن شویم که ما در شاخه استاد هستیم و سپس یک شاخه جدید از آن ایجاد می کنیم. به منوی "شاخه" بروید و "checkout ..." را انتخاب کنید. پنجره انتخاب نشان داده شده در شکل 16-3 را باز خواهد کرد.



**Figure 16-3.** Choosing a branch to check out

متوجه خواهید شد که وقتی مکان نما شما روی یک شعبه معلق می شود ، اطلاعات مربوط به آخرین تعهد آن ظاهر می شود. به شما کمک می کند تا یک شاخه مناسب را پیدا کنید ، اما در صورت داشتن نام های خوب شعبه لازم نیست. شعبه استاد را بررسی کنید و سپس با انتخاب "ایجاد ..." در فهرست "شاخه" ، یک مورد جدید ایجاد کنید. پنجره ایجاد شعبه را نشان داده شده در شکل 16-4 دریافت خواهید کرد.





**Figure 16-4.** Creating of a new branch

اولین قسمت ورودی مهمترین است: نام شعبه جدید شما. شاخه را "سبک های جداگانه و کد" نامگذاری کنید. ورودی دوم ورودی انتخابی است که در آن باید انتخاب کنید که می خواهید شاخه را از کجا ایجاد کنید. در شرایط ما ، ما می خواهیم شعبه جدیدی از شعبه استاد محلی خود ایجاد کنیم. بنابراین "شاخه محلی" را انتخاب کنید و "استاد" را انتخاب کنید. بخش سوم گزینه هایی است که توصیه می کنم گزینه های پیش فرض را نگه دارید. با گزینه های پیش فرض ، Git آخرین تعهدات را در شاخه راه دور (ردیابی) واگذار می کند و سپس شعبه جدید را بررسی می کند.

اکنون می توانید برای دیدن نتیجه بر روی "ایجاد" کلیک کنید. خواهید دید که جعبه پیام کوچک در بالا سمت چپ اکنون "جداگانه کد و سبک" را به عنوان شاخه فعلی لیست می کند. برای اینکه به شما چشم انداز بدهیم ، در اینجا معادلاتی از آنچه ما فقط انجام داده ایم:

```
$ git checkout master  
$ git branch -b separate-code-and-styles
```

اکنون که در یک شاخه صحیح قرار داریم ، می توانیم روی تعهد خود عمل کنیم. هنگام بحث درباره گردش کار Git ، قانون طلایی ما را به خاطر دارید؟ هر تعهد باید حل مسئله را به عنوان یک هدف داشته باشد. به شما اجازه خواهیم داد این مسئله را ایجاد کنید.

تمرین: یک مسئله ایجاد کنید

به شماره های GitHub بروید.

موضوعی با عنوان "کد و سبک های جداگانه" ایجاد کنید.

به شماره issue توجه کنید.

اکنون ما آماده هستیم تا commit کنیم! یک پرونده جدید با نام "style.css" در مخزن خود ایجاد کنید و در این کد جایگذاری کنید:

```
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;
}

```

Then, open "index.html" and change its content to

```

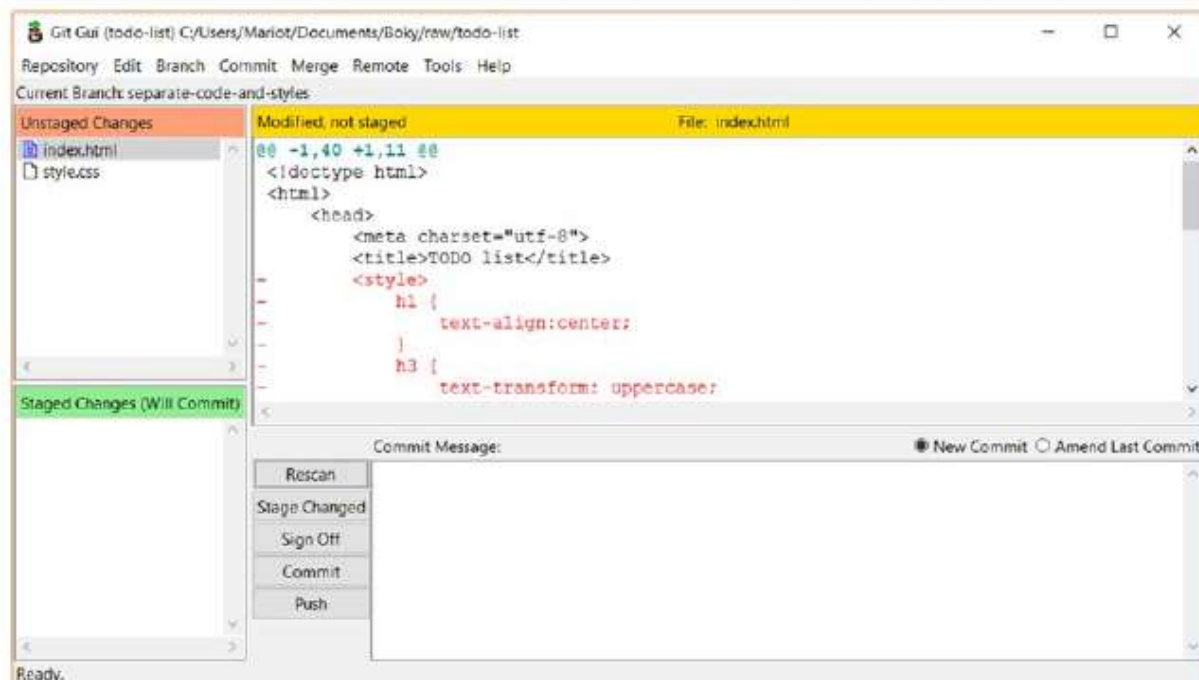
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>TODO list</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>TODO list</h1>

    <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>

```

دو پرونده را ذخیره کنید و برای دیدن نتیجه به Git GUI امیدوار باشید. خواهید دید ... هیچ چیز جدیدی نیست! زیرا Git GUI هنوز از تغییرات ما آگاه نیست. برای دیدن تغییرات ، روی "Rescan" در نزدیکی کادر پیام کلیک کنید. شما نتیجه نشان داده شده در شکل 5-16 را دریافت خواهید کرد.



**Figure 16-5. Changes shown in Git GUI**

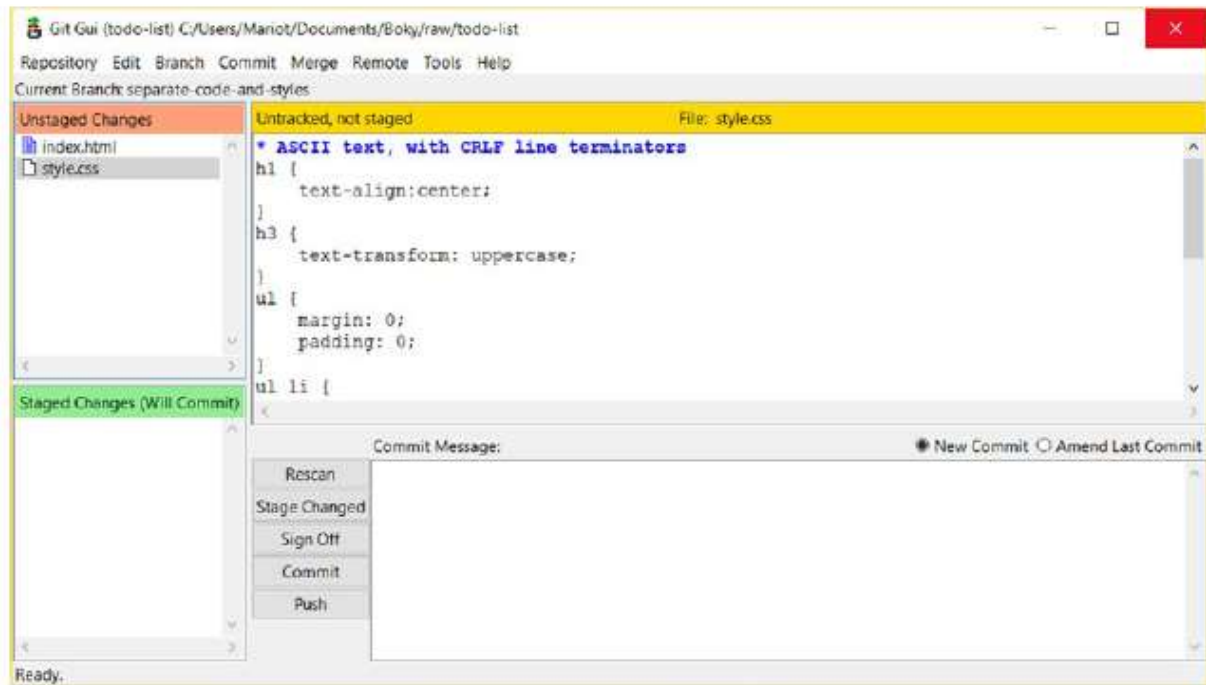
حالا ما تغییرات خود را داریم! شما می توانید لیست پرونده های اصلاح شده را در قسمت بالا سمت چپ Git GUI ، جایی که پرونده های غیرفعال وجود دارد ، مشاهده کنید. متوجه خواهید شد که پرونده ها دارای نمادهای مختلف هستند:

- نماد پرونده خالی برای یک پرونده جدید (هرگز مرتکب نشده است)
- نماد پرونده برای یک پرونده اصلاح شده (قبل از این بخشی از ارتکاب آن بوده است)
- آ "؟" نماد یک پرونده حذف شده (قبلاً بخشی از یک تعهد بوده است)

آیا آن منظره چیزی را به شما یادآوری نمی کند؟ خوب ، این وضعیت وضعیت است ، البته! کلیک بر روی "Rescan" همان اجرای این دستور در ترمینال است:

```
$ git status
```

در اینجا ، "index.html" را اصلاح کردیم و "style.css" ایجاد کردیم. اگر روی نام پرونده ها کلیک کنید (نه نمادها ؛ هنوز روی نمادها کلیک نکنید) ، تغییر نمای را مشاهده خواهید کرد. شکل 16-6 را برای نمونه ای از نتیجه ای که پس از کلیک بر روی style.css می گیرید بررسی کنید.



**Figure 16-6. Diff on the newly created style.css file**

مطمئنناً سریعتر از اجرای "git Diff" است! همچنین ، اگر فایل‌های تغییر یافته زیادی دارید ، کار را آسان تر می کنید ، بنابراین کلیک کردن بر روی نام فایل معادل اجرای این دستورات است:

```
$ git diff index.html
$ git diff style.css
```

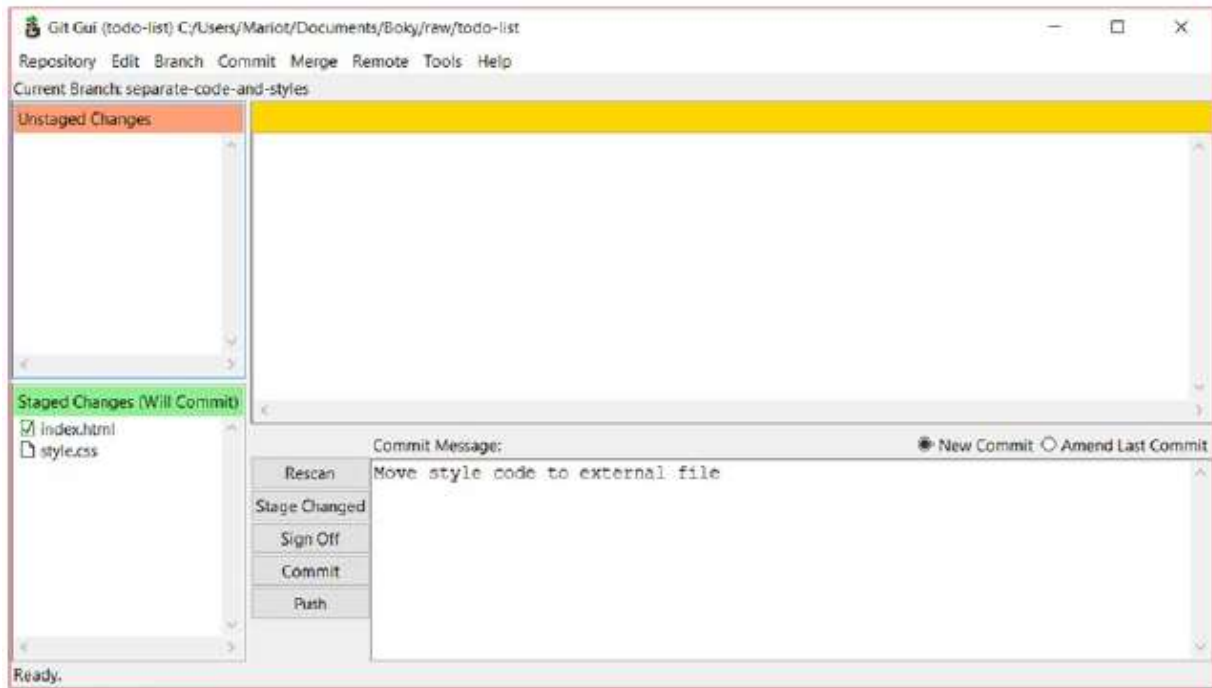
اکنون زمان آن است که پرونده های ما را برای آماده سازی برای ارتکاب مرحله آماده کنیم. نمایش و بالا بردن یک فایل واقعاً آسان است: فقط باید روی نماد آن کلیک کنید. یا می توانید پرونده هایی را که می خواهید مرحله بزنید (با کلیک کردن بر روی نام آنها) انتخاب کرده و در منوی "commit" گزینه "Stage to Commit" را انتخاب کنید. کلیک بر روی نمادهای پرونده همانند اجرای این دستورات است:

```
$ git add index.html style.css
$ git reset HEAD index.html
$ git reset HEAD style.css
```

دیدن؟ سریعتر از تایپ کردن دستورات!

بالاخره می توانیم پروژه خود را انجام دهیم! اما ابتدا مطمئن شوید که تمام پرونده هایی که ایجاد کرده اید یا تغییر یافته مرحله بندی شده اند ، به این معنی که در قسمت پایین سمت چپ قرار دارند. سپس می توانید پیام متعهد خود را در قسمت پایین سمت راست Git GUI ، درست مانند شکل 16-7 بنویسید.





**Figure 16-7. Writing of a commit message**

اکنون با آماده شدن پرونده های خود و ارسال پیام متعهد ، ما آماده هستیم تا commit کنیم. فقط بر روی دکمه "commit" در نزدیکی کادر پیام کلیک کنید. بعد از انجام این کار ، Git GUI به حالت عادی و خالی خود باز می گردد. ما از ابزار گرافیکی متعهد شده ایم! با کلیک بر روی دکمه "commit" نتیجه مشابهی با این دستور وجود دارد:

```
$ git commit -m "Move style code to external file"
```

از آنجایی که شما بهترین دانش آموز من هستید (به دیگران نگویند) ، من به شما اجازه می دهم در شعبه ما متعهد دیگری شوید.

تمرین: یک commit دیگر ایجاد کنید

README.md را باز کنید.

این خط را در انتهای پرونده اضافه کنید: "مجوز: MIT".

یک پرونده جدید با نام LICENSE ایجاد کنید.

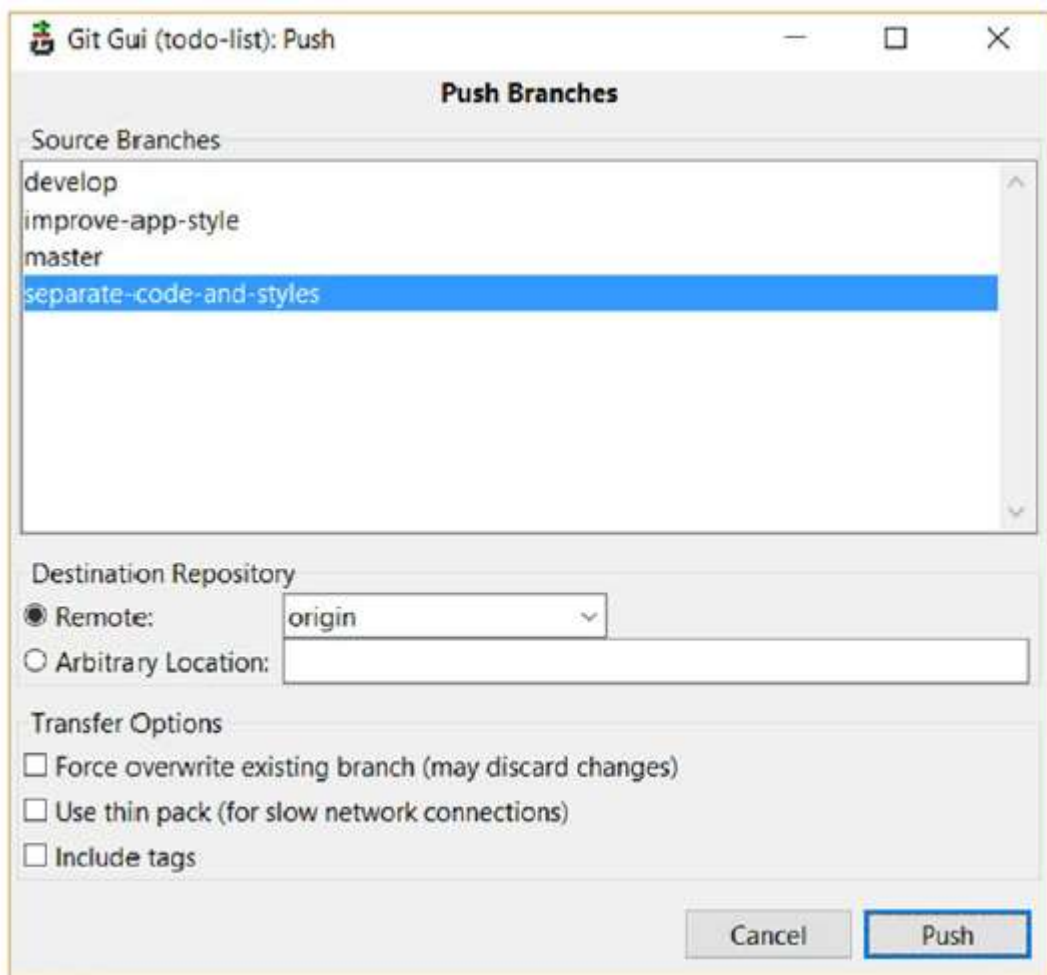
متن مجوز را از آدرس <https://choosealicense.com/licenses/mit> کپی کنید و در

پرونده LICENSE.

هر دو پرونده را مرحله کنید.

با پیام "اضافه کردن مجوز MIT" متعهد شوید.

اوه! اکنون دو شعبه جدید در شعبه جدید خود دارید و زمان آن رسیده است که آنها را به مخزن راه دور منتقل کنید. مطمئناً حدس زده اید که کدام دکمه را کلیک کنید. "push" است با کلیک بر روی آن نتیجه در شکل 8-16 به شما می دهد.

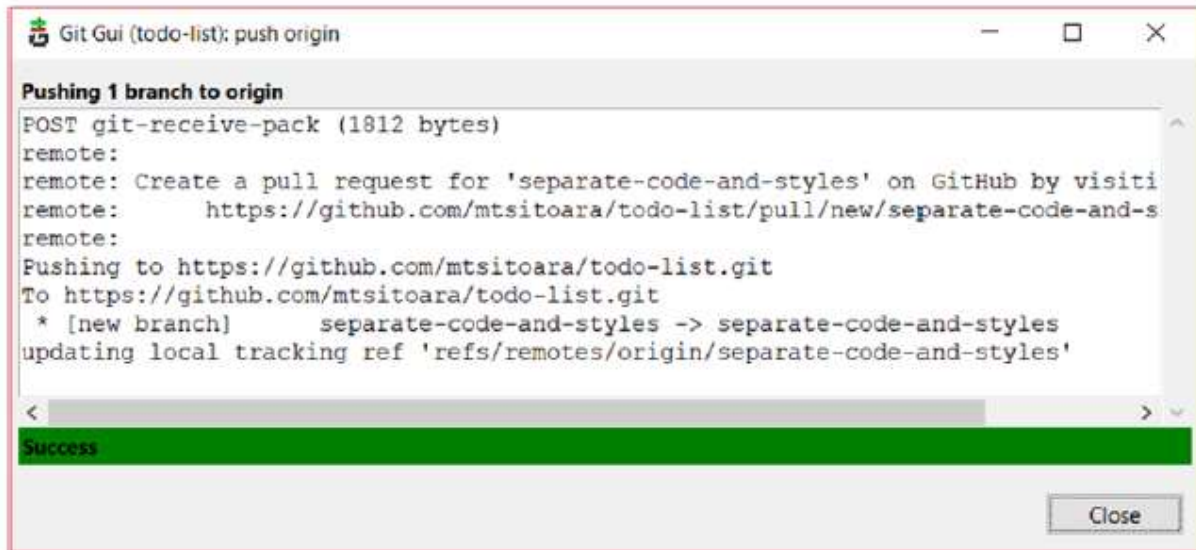


**Figure 16-8. Pushing a branch**

این یک رابط سر راست است؛ شما فقط باید شاخه ای را که می خواهید فشار دهید و محلی که می خواهید آن را فشار دهید را انتخاب کنید.

شعبه فعلی به طور پیش فرض انتخاب شده است ، بنابراین ما لازم نیست چیزی را تغییر دهیم. بخش دوم ، کشویی انتخاب مقصد است. و مجدداً ، لازم نیست چیزی تغییر دهیم زیرا فقط یک مخزن از راه دور داریم. گزینه های فعلی را نادیده بگیرید. ما آنها را در فصل بعد خواهیم دید.

برای فشار دادن کلیک کنید! اگر از یک تأیید هویت HTTPS برای ارتباط با GitHub استفاده می کنید ، از شما نام کاربری و رمز عبور GitHub خواسته می شود و سپس نتیجه نشان داده شده در شکل 9-16 را دریافت می کنید.



**Figure 16-9. Push result**

هیچ چیز جدیدی در اینجا وجود ندارد ، ما به همان نتیجه این دستور رسیدیم:  
و اینگونه است که با Git GUI متعهد می شوید! درست است ، درست است؟ و خیلی سریع این یک ابزار عالی است که می تواند زمان زیادی را در هنگام مرور نظرات صرفه جویی کند. صحبت از تعهدات ، اجازه دهید ابزار پیش فرض دیگر را ببینیم!

## مرور: gitk

در بخش قبلی ، ما در مورد ایجاد و فشار تعهدات زیادی صحبت کردیم. اکنون می خواهیم آن دسته از اقدامات را در زیستگاه طبیعی خود تجسم کنیم: مخزن. gitk ابزاری ساده برای داشتن تصویری ساده از تاریخ پروژه شما است. شما می توانید از آن به عنوان یک دستور "git log" غرق فکر کنید. اسناد بیشتر درباره gitk را می توانید در <https://git-scm.com/docs/gitk> پیدا کنید.

از آنجا که قبلاً Git-GUI را باز کرده اید ، اجازه دهید از آن برای باز کردن gitk استفاده کنیم. به سادگی "منوی تاریخچه شعبه" را از فهرست "مخزن" انتخاب کنید. با این کار gitk باز خواهد شد ، و پنجره ای را که در شکل 10-16 نشان داده شده است مشاهده خواهید کرد.

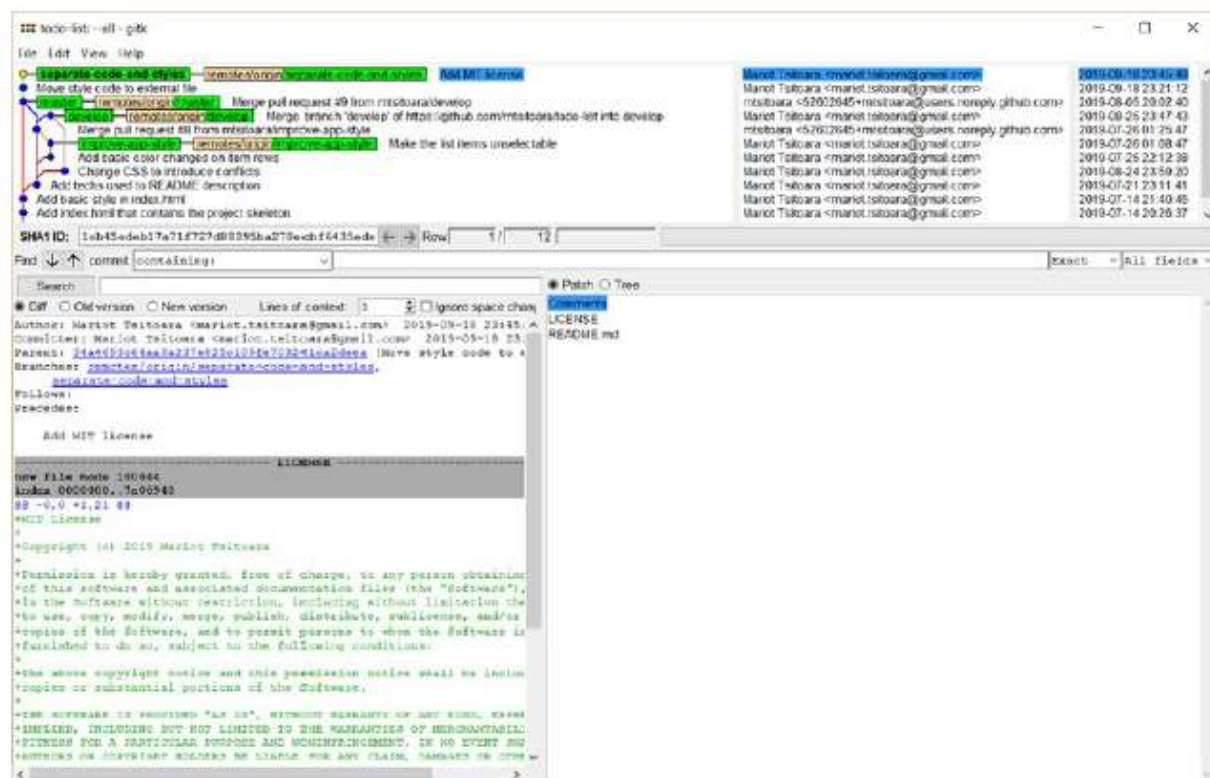


Figure 16-10. gitk interface

در بالای پنجره لیستی از کلیه تعهدات پروژه خود، از همه شعبه ها پیدا خواهید کرد. در یک نمودار گرافیکی زیبا ارائه شده است که می توانید با دستور آن را در کنسول تولید کنید:

```
$ git log --oneline --graph
```

برای کسب اطلاعات بیشتر در مورد آنها می توانید بر روی تعهدات کلیک کنید. با انتخاب یک تعهد، نماها در پایین پنجره به روز می شود. قسمت پایین سمت چپ یک نمای متفاوت است اما با پیچ و تاب: شما همچنین می توانید نسخه قدیمی یا جدید فایلها را مشاهده کنید.

قسمت پایین سمت راست لیستی از تمام پرونده های تغییر یافته در تعهد است. برای دیدن تغییرات در نمای متفاوت می توانید بر روی آنها کلیک کنید. کلیک روی تعهد معادل اجرای کد زیر است:

```
$ git show <commit_name>
```

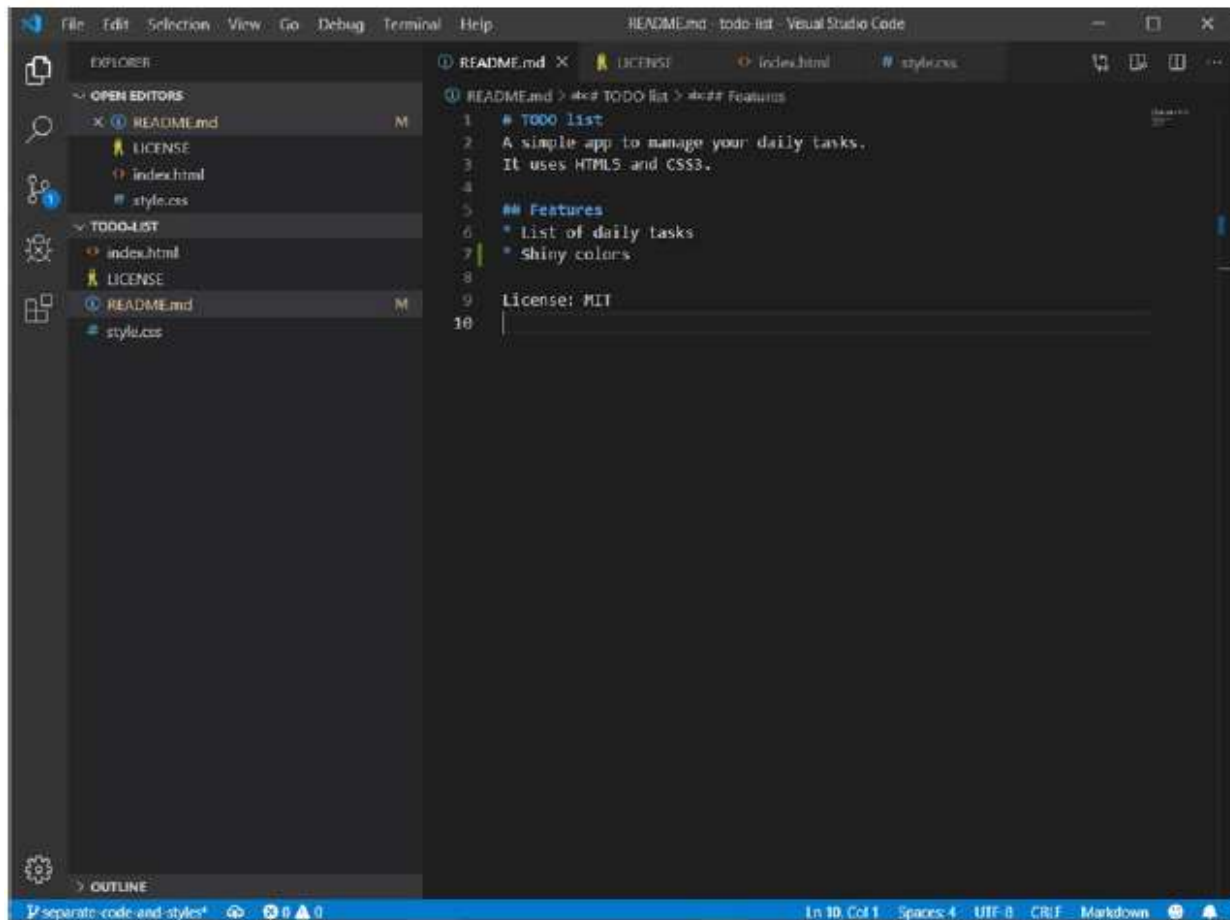
و این برای gitk، ابزار پیش فرض مرور Git است! از آنجایی که اکنون می توانید ابزارهای گرافیکی پیش فرض را مرتب و مرور کنید، اکنون زمان آن رسیده است تا شما را به ابزارهای دیگر ارائه دهید.

## ابزارهای IDE

همانطور که در بخش قبلی دیدیم ، نسبت به تایپ کردن در کنسول ، تعهد با یک ابزار گرافیکی بسیار سریع است. اما هنوز هم یک مشکل وجود دارد: شما باید محیط IntegratedDevelopment خود را ترک کرده و از آنها استفاده کنید. آیا می توانید از ابزارهای گرافیکی مستقیماً از ویرایشگر خود استفاده کنید ، خوشایند نخواهد بود؟ با بسیاری از ویرایشگران مدرن امکان پذیر است. من شما را به دو IDE محبوب که Git یکپارچه هستند ارائه می دهم تا بتوانید از آنها برای توسعه آینده خود استفاده کنید. اگر نمی خواهید از آنها استفاده کنید یا در حال حاضر عاشق ویرایشگر فعلی خود هستید ، احتمال دارد که IDE شما به اندازه کافی مدرن ، دارای ابزار یا افزونه Git یکپارچه باشد. هر IDE رابط و تجربه کاربر خاص خود را دارد ، بنابراین من در این بخش به جزئیات نمی پردازم. من فقط می خواهم به چه ویژگی هایی در دسترس باشم.

## Visual Studio Code

ویرایشگر بسیار محبوب ، ویژوال استودیو کد ، یک IDE سبک وزن است که توسط مایکروسافت پشتیبانی می شود. می توانید آن را در <https://code.visualstudio.com/> پیدا کنید. این جدید است ، بنابراین همه اسباب بازی های جدید و براق در آن ادغام شده است. و گیت در مرکز آن است. شما می توانید شکل و احساس VS Code را در شکل 11-16 مشاهده کنید.

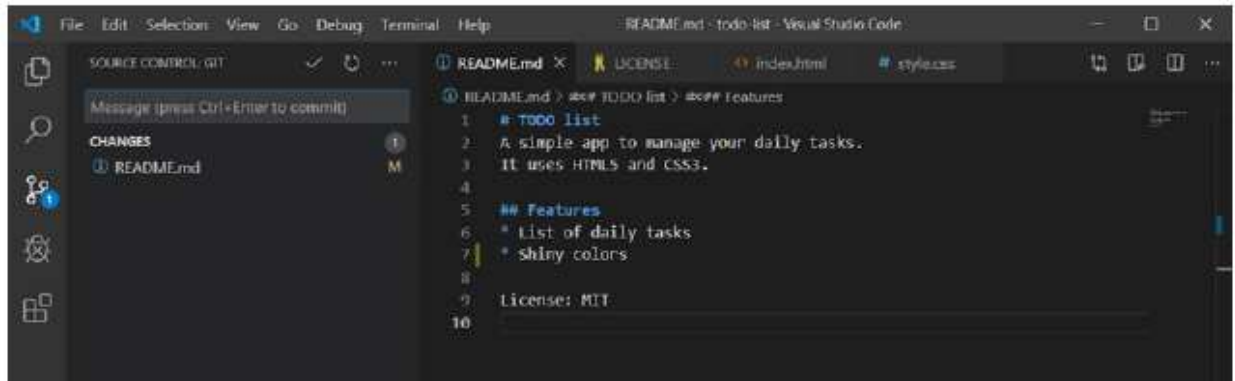


**Figure 16-11.** Visual Studio Code

این رابط کاربری مشابه با هر IDE دیگر اما با اندکی پاداش دارد: می توانید ردپای Git را در اینجا و آنجا مشاهده کنید. ابتدا، اگر یک فایل ردیابی شده را تغییر دهید (در اینجا README.md)، قسمت ویرایش شده برجسته می شود. دیگر نیازی به اجرای git diff نیست! و در سمت چپ پایین پنجره، نام شاخه فعلی را دارید. در صورت کلیک بر روی آن، می توانید شاخه ای را که می خواهید حرکت کنید یا یک شعبه جدید ایجاد کنید، انتخاب کنید. اگر پرونده های غیرقابل استفاده دارید، در نزدیکی نام شعبه شما یک علامت "\*" وجود خواهد داشت و یک نماد "M" در نزدیکی نام پرونده های مربوطه وجود دارد. اگر فایل های غیرقابل قبول فشاری را تنظیم کرده اید، علامت "+" دارید.

برای دسترسی به Git Tab که در شکل 12-16 نشان داده شده است، روی نماد Source Control کلیک کنید.



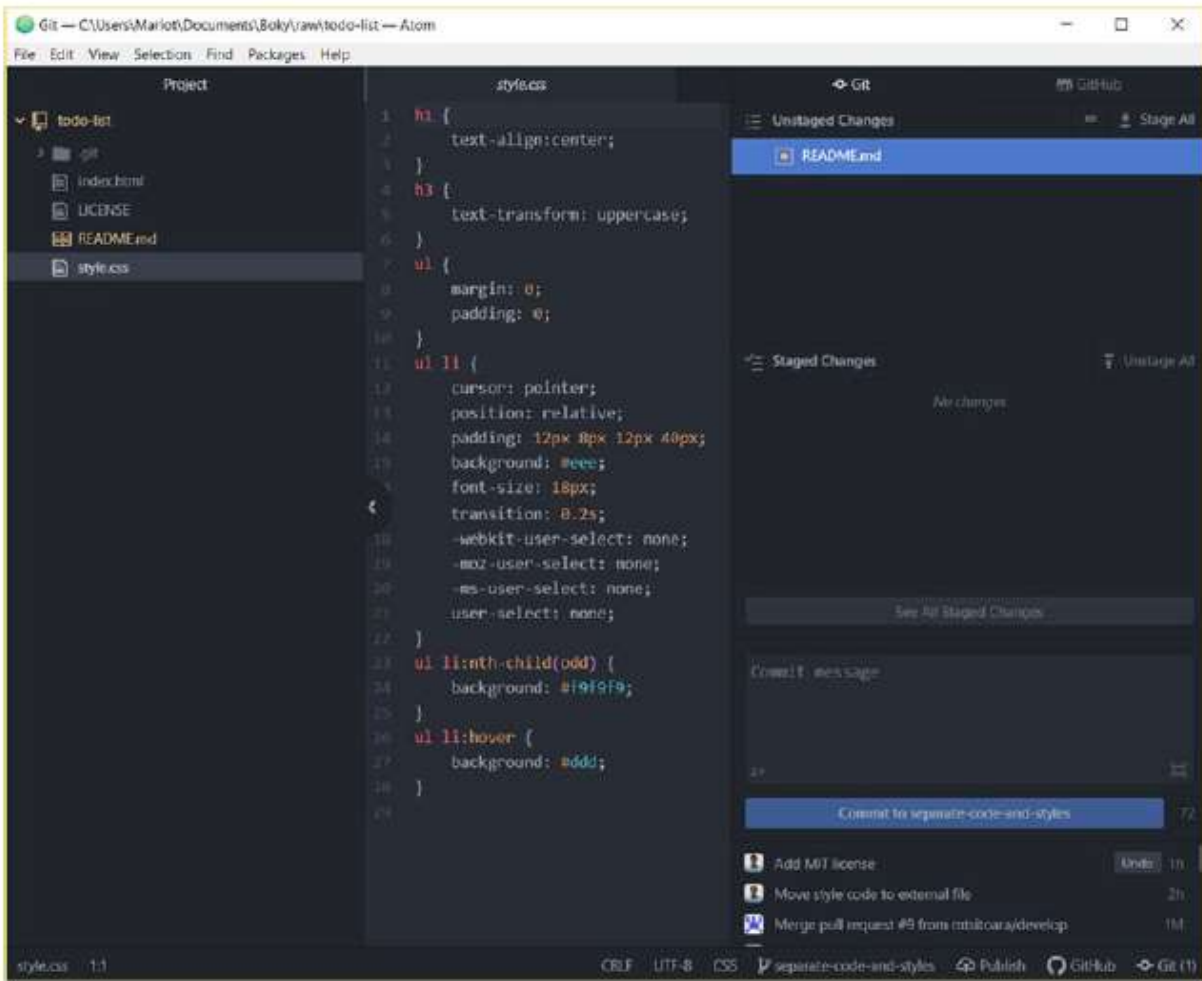


**Figure 16-12.** Source Control view

این نمای بسیار شبیه به git-gui است ، بنابراین به شما امکان می دهد خودتان آنرا کشف کنید!

اتم

Atom یک IDE است که توسط GitHub تحت push قرار گرفته است و همچنین یک انتخاب بسیار محبوب در بین توسعه دهندگان است. می توانید آن را از طریق <https://atom.io> بررسی کنید. رابط کاربری آن را در شکل 13-16 مشاهده می کنید.



**Figure 16-13.** Atom interface

این همان ویژگی های Git را به عنوان Visual Studio Code دارد اما با کمی پیچ و خم: می توانید حساب GitHub خود را به آن پیوند داده و PR را مستقیماً از ویرایشگر بسازید!

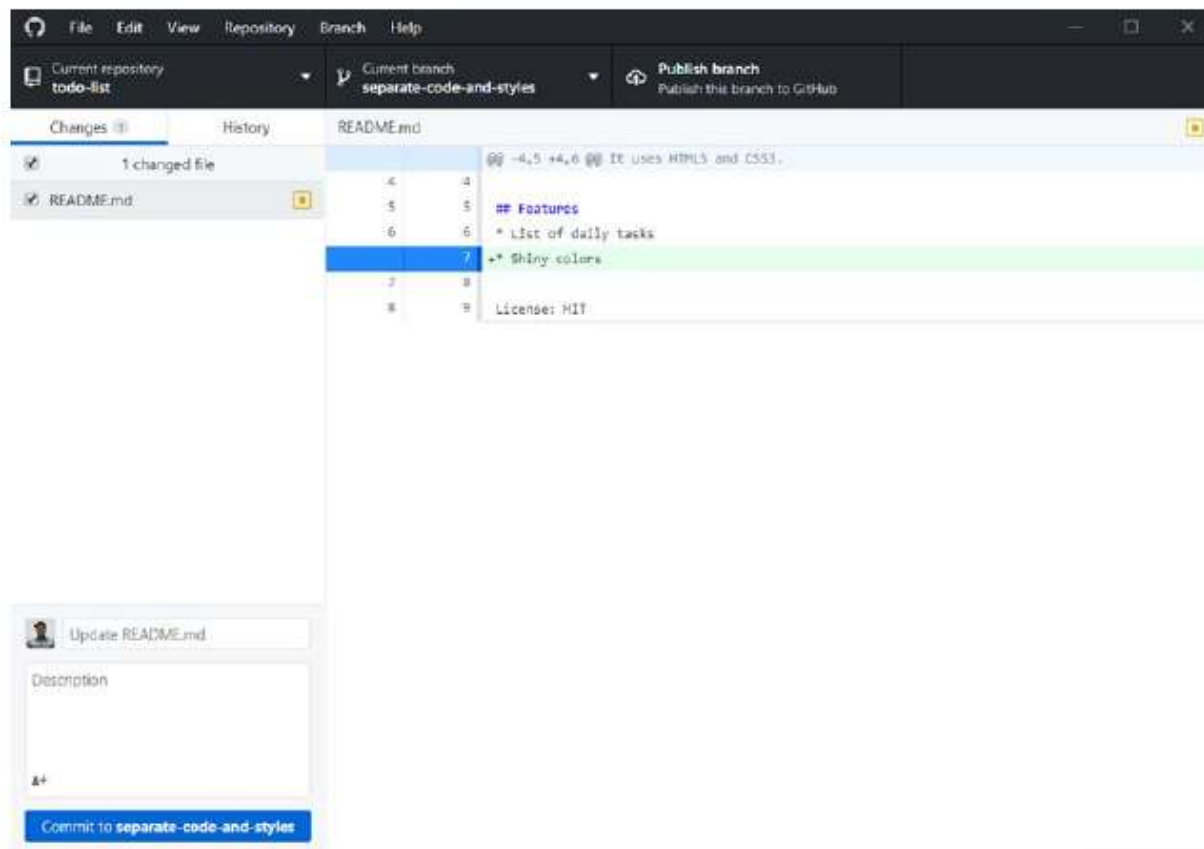
## ابزارهای تخصصی

ما ابزارهای پیش فرض Git و برخی IDE ها را دیدیم که Git یکپارچه هستند. اکنون ، بیا ببینیم برخی از ابزارهای ویژه برای Git را توسعه دهیم.

## دسک تاپ GitHub

اگر می خواهید ابزارهای پیش فرض gitk و git-GUI را دوست داشته باشید اما از رابط کاربری خود متنفر باشید بیا ببینیم از ابزارهای پیش فرض فوق العاده استفاده کنید ، اما ظاهر آنها در آن دوران مدرن عجیب است. دسک تاپ GitHub (در

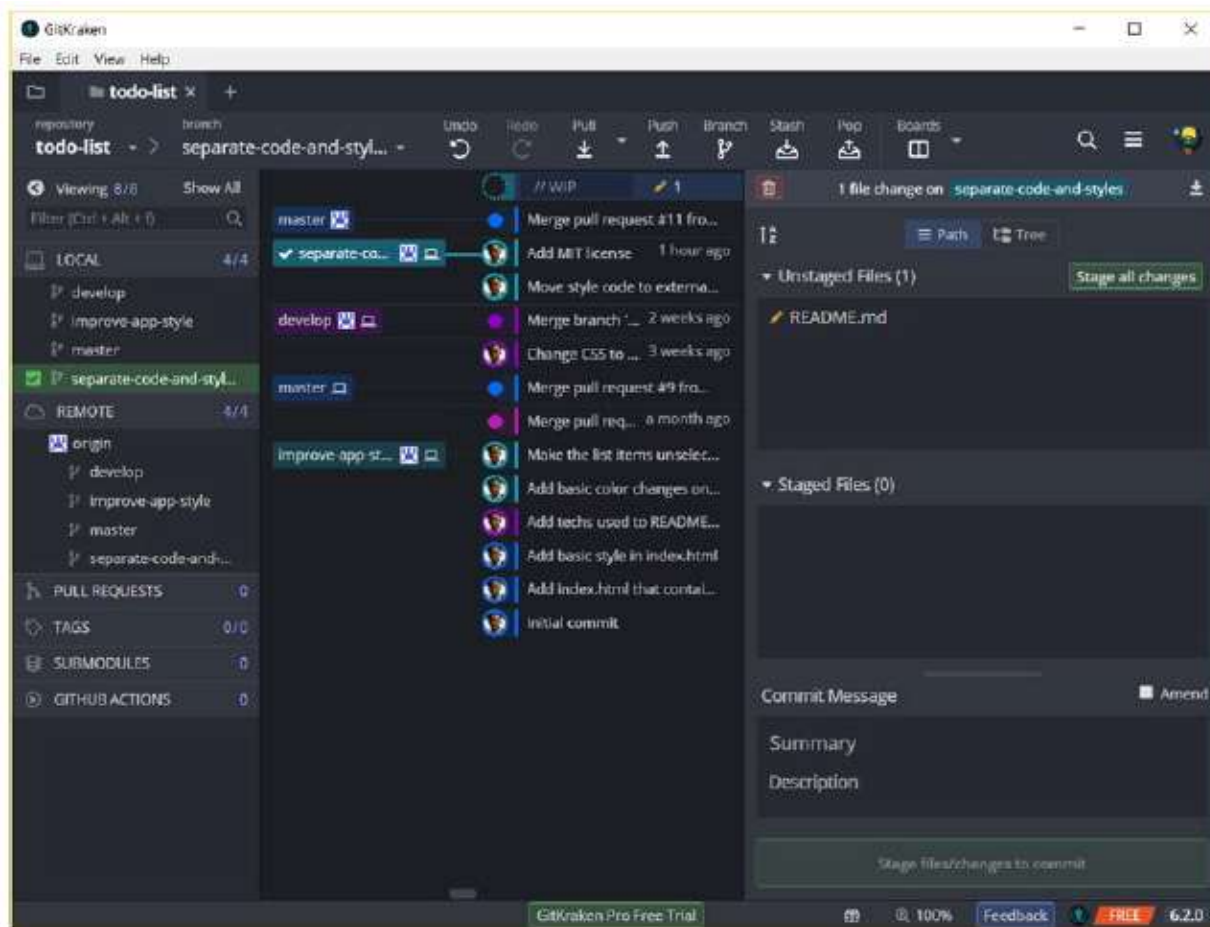
<https://desktop.github.com> / یافت شده است) برای جایگزینی این ابزارها ایجاد شده است. تمام ویژگی های آنها در یک نرم افزار ترکیب شده است. می توانید شکل 14-16 رابط کاربری GitHub را بررسی کنید.



**Figure 16-14. GitHub Desktop**

## GitKraken

GitKraken یک سرویس گیرنده Git است که توسط Axolosoftware ساخته شده است و روز به روز محبوب تر می شود. می توانید آن را در وب سایت آن به نشانی اینترنتی [www.itkraken.com](http://www.itkraken.com) دریافت کنید. این پیشرفته تر از سایر ابزارهای دیگر است زیرا هدف آن تقویت بهره وری توسعه دهنده است. حتی دارای ویرایشگر کد یکپارچه است! رابط کاربری آن را در شکل 15-16 مشاهده می کنید.



**Figure 16-15. GitKraken overview**

باز هم ، رابط کاربری مشابه سایرین است ، اما آنچه GitKraken را متمایز می کند زیبایی آن است: فوق العاده زرق و برق دارد!

## خلاصه

این فصل سرگرم کننده بود ، نه؟ ما در مورد چگونگی استفاده از یک ابزار گرافیکی برای ایجاد تعهد و مرور آنها چیزهای زیادی می آموزیم. همچنین یک ابزار جدید کشف کردیم که بتواند در یک IDE یا یک ابزار تخصصی در دسترس ما باشد. و چگونه می توانیم ابزار پیش فرض خوب قدیمی خود را فراموش کنیم؟!

ممکن است از خود پرسید که چرا از همان ابتدا از ابزار گرافیکی استفاده نمی کنید؟ به این دلیل است که استفاده از ابزاری بدون دانستن مفاهیم موجود در آنها ضد تولید و اتلاف وقت است. به من اعتماد کن ، یادگیری استفاده از ترمینال ارزشش را داشت! در مورد پایانه ها صحبت می کنیم ، بگذارید برای برخی از پیشرفته های Gitcommands به آن برگردیم!