

# فصل هفدهم

## گیت پیشرفته

در فصل گذشته ، ما یاد گرفتیم که چگونه ویژگی های اصلی Git را در یک زمینه گرافیکی انجام دهیم. اکنون ، بیایید دستورات Git دیگری را مشاهده کنید که شما به اندازه دیگران استفاده نخواهید کرد بلکه برای بهره وری بهتر قدرتمند و لازم هستند. این دستورات بسیار آسان برای یادگیری است که اگر شما با استفاده از Git اشتباه کرده اید ، برای شما مفید خواهد بود. ما می خواهیم برخی از مشکلات رایج را مشاهده کنید که مطمئناً پس از چند بار با استفاده از Git مرتفع خواهید شد. سپس ما ساده ترین راه حل آنها را خواهیم دید. این یک فصل بسیار آسان است اما ما می خواهیم برخی از ویژگی های قدرتمند Git را بیاموزیم.

### بازگرداندن

ما قبلاً دیده ایم که چگونه می توان تعهد خود را در فصل های گذشته بازگرداند. اما بیشتر اوقات ، تنها کاری که می خواهید انجام دهید بازگشت یک پرونده واحد به حالت قبلی است. این بیشتر زمانی اتفاق می افتد که مدتی است که کدگذاری می کنید تا بدانید کل استراتژی شما اشتباه بوده است.

و به جای اینکه صدها بار Cmd-Z را بزنید ، بهتر است پرونده را برگردانید. شما احتمالاً از قبل می دانید که چگونه این کار را انجام دهید زیرا Git به شما می گوید که چگونه این کار را انجام دهید پس از بررسی وضعیت git. ابتدا اجازه دهید README.MD را باز کنیم و متن دیگری را در آن اضافه کنیم.

```
# TODO list
A simple app to manage your daily tasks.
It uses HTML5 and CSS3.

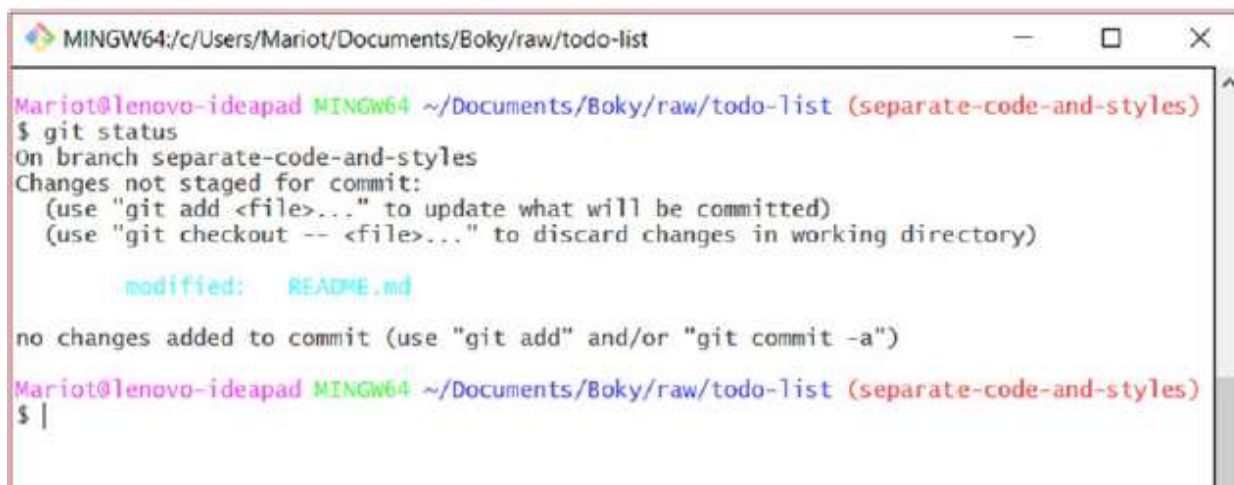
## Features
* List of daily tasks
* Pretty colors

License: MIT
```

حال ، وضعیت را می بینیم.

```
$ git status
```

طبق معمول ، وضعیت مخزن خود را مشاهده خواهید کرد (در شکل 1-17 نشان داده شده است).



```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git status
On branch separate-code-and-styles
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

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

**Figure 17-1.** Git status after a changed file

هیچ چیز جدیدی در اینجا وجود ندارد ، اما توجه خود را به دستورالعمل های نشان داده شده در بالای پرونده اصلاح شده هدایت کنید. همانطور که مشاهده می کنید ، برگرداندن پرونده به حالت قبلی فقط به معنای بررسی آن است. دستور بدین ترتیب است

```
$ git checkout -- <file>
```

این دستور از هر تغییری که در یک پرونده خاص انجام داده اید صرف نظر می کند. در استفاده از آن مراقب باشید ، زیرا کد ارزشمند را پاک نمی کند. شاید بهتر باشد از رابط کاربری گرافیکی (GUI) استفاده کنید تا بتوانید سریعاً قبل از اینکه آنها را دور بیندازید ، جزئیات دقیقی از تغییرات فعلی را بدست آورید. سعی می کنیم با دستور زیر تغییرات ما در README.md را دور بزنیم.

```
$ git checkout -- README.md
```

شما پاسخی را از این دستور دریافت نخواهید کرد ، اما اگر وضعیت git را دوباره بررسی کنید ، خواهید دید که README.md به حالت قبلی خود برگشته است.

## Stashing

بسیاری از اوقات می خواهید به حرکت بین شاخه ها پردازید اما نمی توانید زیرا فهرست کار شما کثیف است. در این زمینه ، کثیف به معنای این است که شما پرونده های تغییر ناپذیر را داشته باشید ، چه در حالت اصلاح شده یا مرحله ای. تنها راه

تغییر شعبه این است که ابتدا آنها را مرتکب شوید. اما بیشتر اوقات ، شما هنوز آماده نیستید که مرتکب شوید ، زیرا هنوز مسئله حل نشده است.

یک راه حل برای این کار ، تعهد موقت ، تغییر شاخه ، کار بر روی آن و سپس بازگشت و اصلاح تعهد موقت است. بسیاری از مشکلات به این روش وجود دارد: اول ، هنگامی که مرتکب می شوید ، فهرست کار شما پاک خواهد شد ، به این معنی که شما دیگر نمی دانید که کدام پرونده ها تغییر یافته اند. دوم ، این یک روش ساده کثیف و زشت است. به همین دلیل فرمان اصلاح ایجاد نشده است. راه حل ایده آل استفاده از تکنیکی به نام "stashing" است. Stashing به معنای گرفتن هر پرونده ردیابی اصلاح شده در WorkingDirectory است و بعداً آن را کنار می گذارد. این بدان معناست که شما یک دایرکتوری تمیز خواهید داشت و می توانید بدون اینکه مجبور به انجام تغییرات خود شوید ، در مخزن خود حرکت کنید. این تغییرات در پایگاه داده کوچکی به نام "stash" ذخیره می شوند.

شما می توانید به عنوان مخزن موقت برای تعهدات ناتمام خود فکر کنید. این به عنوان یک بانک اطلاعاتی در اولویت اول طراحی شده است ، به این معنی که آخرین تغییراتی که شما ایجاد کرده اید برای اولین بار به شما ارائه می شود. بهترین راه برای درک آن ، امتحان کردن است. بنابراین ، اجازه دهید پرونده README.MD خود را دوباره تغییر دهیم.

```
# TODO list
```

```
A simple app to manage your daily tasks.
```

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

```
## Features
```

```
* List of daily tasks
```

```
* Pretty colors
```

```
License: MIT
```

اگر وضعیت را بررسی کنید ، خواهید دید که README.md اصلاح شده اما غیرفعال است. شما همان نتیجه قبلی را می گیرید (شکل 1-17). حال فرض کنیم که در حالی که شما روی این مسئله کار می کنید ، فوری نیاز به توجه شما دارد. بدیهی است ، شما می توانید شعبه اصلی را اکنون بررسی کنید زیرا فهرست کار شما کثیف است و نمی توانید تغییرات فعلی خود را برگردانید زیرا هنوز کاملاً کامل نشده اید. راه حل این است که تغییرات فعلی خود را در جایی کاهش دهید تا بتوانید یک فهرست راهنمایی تمیز برای کار با آن داشته باشید. برای انجام این کار ، شما باید از دستور stash استفاده کنید که بسیار آسان است:

این دستور پرونده های اصلاح شده شما را برای مرحله بندی کردن آنها انجام می دهد و متعهدانه موقت را در قسمت stash ایجاد می کند تا فهرست کار شما تمیز شود. آن را امتحان کنید و نتیجه مشابهی را خواهید گرفت که در شکل 1-17-2 نشان داده شده است.



```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git stash push
Saved working directory and index state WIP on separate-code-and-styles: 1eb45ed Add MIT license
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ |
```

**Figure 17-2.** *Stashing current changes*

همانطور که مشاهده می کنید ، تغییرات ناخواسته شما یک اسم و توضیحی مانند یک تعهد معمولی داده شده است. این طبیعی است زیرا ذخیره فقط یک مخزن موقت است که فقط یک شاخه دارد. اگر وضعیت مخزن را بررسی کنید ، همانطور که در نظر گرفته شده است یک فهرست کار تمیز دریافت خواهید کرد (در شکل 17-3). و در نهایت می توانید به شاخه های دیگر بروید.



```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git stash push
Saved working directory and index state WIP on separate-code-and-styles: 1eb45ed Add MIT license
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git status
On branch separate-code-and-styles
nothing to commit, working tree clean
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ |
```

**Figure 17-3.** *A stash push produces a clean working directory*

فشار آوردن به تغییرات در این حالت باعث می شود آزادی حرکت بیشتری داشته باشید بدون اینکه کار فعلی خود را از دست بدهید. در پیشرفت سریع بسیار مفید است.

از آنجا که stash فقط یک مخزن کوچک است ، بنابراین می توانید اکثر Gitfeatures را روی آن انجام دهید ، مانند بررسی سابقه تاریخ یا مشاهده دقیق جزئیات از تغییرات. بیاید برای درک بهتر آن ، این استیشن را کشف کنیم. اول ، بگذارید با استفاده از دستور فهرست stash ، به گزارش تاریخ نشان دهیم.

```
$ git stash list
```

این امر اگرچه نمای ساده ای از تاریخچه نشان داده شده در شکل 17-4 ، شما را آشنا می کند.



```
MINGW64: c:/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git stash list
stash@{0}: WIP on separate-code-and-styles: 1eb45ed Add MIT license
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ |
```

**Figure 17-4. List of stashed changes**

همانطور که قبلاً گفتیم ، این بانک اطلاعاتی در آخرین نسخه اول کار می کند ، بنابراین اگر در فهرست کار خود تغییرات دیگری ایجاد کردیم و آنها را حذف کردیم ، آنها در بالای stash فعلی ما ظاهر می شوند. در شکل 17-4 متوجه خواهید شد که هر یک از ستونها دارای یک عدد است. تعامل با آنها ساده تر است ، برخلاف تعهدی که باید آنها را با نام آنها صدا کنید. بگذارید با استفاده از نمایش، نمایش داده شده از فرمان ، نمای تفصیلی از تغییر ایجاد شده را مشاهده کنیم.

**\$ git stash show**

این دستور ساده به شما نشان می دهد پرونده های تغییر یافته در نوک استاشن ، به معنی آخرین تغییراتی که به آن وارد شده اند. شکل 17-5 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 (separate-code-and-styles)
$ git stash show
README.md | 1 +
1 file changed, 1 insertion(+)
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ |
```

**Figure 17-5. Detailed view of the tip of the stash**

دستور `stash show` فقط توضیحات تغییرات موجود در `stash` را نشان می دهد ، اما چیز دیگری نیست. برای دیدن تغییرات ، باید `stash` را اعمال کنید. استفاده از `stash` بسیار ساده است: فقط دستور زیر را اجرا کنید.

## `$ git stash pop`

این دستور آخرین تغییرات در `stash` را انجام می دهد و آن را در شاخه فعلی اعمال می کند. و همانطور که از نام آن پیداست ، ظاهر تغییرات باعث می شود که آنها از حالت خارج شوند.

بنابراین ، اگر شما فقط یک مجموعه از تغییرات را در استار خود داشته باشید ، پس از بالا آمدن نوک ، آن خالی خواهد بود. اگر دستور قبلی را اجرا کنید ، نتیجه ای که می گیرید همان خواهد بود که تغییرات را از نو مجدداً ایجاد کرده و وضعیت را بررسی کرده اید (نشان داده شده در شکل 6-17).



```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git stash show
README.md | 1 +
1 File changed, 1 insertion(+)

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git stash pop
On branch separate-code-and-styles
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (8493c5ec10a605f41466fe4b535bb8289bd24f84)

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ |
```

**Figure 17-6. Popping the last set of changes**

ما در ابتدا دوباره برمی گردیم! اما اگر آرزو می کردیم ، می توانستیم شاخه های ساخته شده را تغییر دهیم یا به منشأ فشار بیاوریم بدون اینکه تغییرات گرانبها را از دست بدهیم.

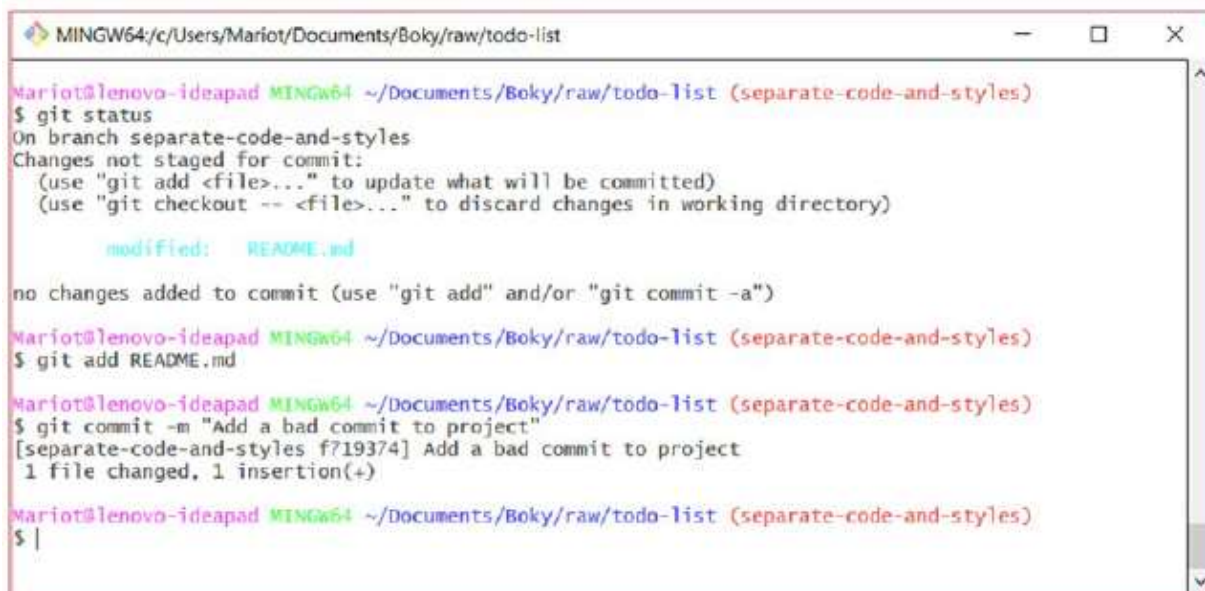
وقتی می خواهید تغییرات فعلی خود را کنار بگذارید برای تغییر سریع در جای دیگر ، `Stashing` مفید است. به عنوان یک قاعده کلی ، اگر شما نیاز به استفاده از بیش از یک مجموعه تغییر ایجاد شده داشته باشید ، در جریان کار خود اشتباهی انجام می دهید.

تنظیم مجدد

امیدوارم که شما از این ویژگی غالباً استفاده نکنید زیرا بسیار مخرب است! بعضی اوقات ، می خواهید همه کارهایی را که انجام داده اید دور کنید و روی یک صفحه تمیز کار کنید ، حتی اگر قبلاً پروژه خود را انجام داده اید. برای درک بهتر آن ، بگذارید تعهدی ایجاد کنیم و سپس آن را دور بیندازیم.



برخی از اصلاحات را در README.md انجام دهید ، آن را مرحله کنید و سپس پروژه را انجام دهید ، همانطور که در شکل 17-7 نشان داده شده است.



```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git status
On branch separate-code-and-styles
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git add README.md
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git commit -m "Add a bad commit to project"
[separate-code-and-styles f719374] Add a bad commit to project
1 file changed, 1 insertion(+)
```

**Figure 17-7.** Add a bad commit to the project

برای چشم انداز این ، بگذارید با استفاده از دستور git log ، سابقه تاریخ فعلی را بعد از این مرتکب بررسی کنیم.

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

این دستور ، آخرین اقدامات در مورد این شاخه را ، دقیقاً مانند شکل 17-8 نشان می دهد.



```
MINGW64/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git log --oneline
f719374 (HEAD -> separate-code-and-styles) Add a bad commit to project
1eb45ed (origin/separate-code-and-styles) Add MIT license
34a4693 Move style code to external file
c9991f8 (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 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 (separate-code-and-styles)
$ |
```

همانطور که می بینید ، آخرین تعهد ما در بالای گزارش ما قرار دارد. توجه کنید که مرجع HEAD به آن اشاره شده است. این بدان معنی است که تعهد بعدی (یا شعبه) ما این تعهد را به عنوان والدین انجام خواهیم داد. شما همچنین متوجه خواهید شد که مبداء از راه دور / سبک ها و کد های جداگانه تغییر نکرده اند ، به این دلیل که ما هنوز پروژه خود را تحت فشار قرار نداده ایم.

اما بگذارید تصور کنیم که شما کاملاً از آن متعهدین ناراضی هستید و می خواهید این کار را انجام دهید. تنها انتخاب شما بازگشت مجدد شعبه به حالت قبلی است. برای تنظیم مجدد پروژه ، از دستور `git reset` و به دنبال آن وضعیت پروژه برای تنظیم مجدد در آن استفاده می کنیم. برای انجام این کار باید از گزینه "-" استفاده کنید ، زیرا این یک دستور بسیار خطرناک است. به عنوان مثال ، بازگشت به همان حالت شاخه راه دور به دستور زیر نیاز دارد:

```
$ git reset --hard origin/separate-code-and-styles
```

این دستور همه چیز را پاک می کند تا پروژه دوباره به حالت قبلی برگردد. در شکل 17-9 نتیجه آن را ببینید.



```
MINGW64/c:/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git reset --hard origin/separate-code-and-styles
HEAD is now at 1eb45ed Add MIT license
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git status
On branch separate-code-and-styles
nothing to commit, working tree clean
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ |
```

**Figure 17-9.** Status of the project after a reset

تعهدات شما پس از حالت هدف ، تغییرات فعلی و پرونده های مرحله بندی شده شما حذف خواهند شد زیرا گزینه "-" هارد" همه چیز را در مسیر خود بازنویسی می کند. این خطرناک ترین فرمان `git` است و قبل از استفاده باید به سختی فکر کنید.

تنظیم مجدد فقط باید در آخرین حالت انجام شود. ترجیح دهید در صورت امکان با فقط مستقیماً به کار خود روی یک شاخه جدید برگردید. با بی احتیاطی ، تنظیم مجدد می تواند داده های شما را از بین ببرد.



## خلاصه

در این فصل به مفاهیم پیشرفته ای از Git پرداخته شده است که هنگام مواجهه با موقعیت های خاص ، برای شما مفید خواهند بود. برای بازگشت فایل به حالت قبلی و بدون تلاش زیاد ، باید از تنظیم مجدد استفاده کنید. و البته ، شما می توانید با استفاده از رابط کاربری گرافیکی نیز آن تغییرات را برگردانید. اگر نیاز به تغییر سریع متن داشته باشید ، ذخیره سازی بسیار مفید خواهد بود. و در آخر اینکه ، تنظیم مجدد سخت یک ویژگی همه جانبه است که بسیار مخرب است؛ مگر اینکه چاره دیگری نداشته باشید.

این نتیجه گیری درس ما در مورد دستورات پیشرفته Git است. بیایید اکنون به GitHub برگردیم ، تا چندین ویژگی دیگر را پیدا کنیم که می توانند در مدیریت پروژه به ما کمک کنند.

# قسمت چهارم

## منابع اضافی

### فصل هجدهم

## جزئیات بیشتر با GitHub

تقریباً هر ویژگی Git را که روزانه در فصل های قبلی از آنها استفاده خواهید کرد ، دیده ایم. حالا بیایید نگاه خود را به سمت GitHub برگردانیم ، که تاکنون فقط به عنوان یک سایت میزبان کد استفاده شده است. اما ما قبلاً مشخص کرده ایم که GitHub بسیار بیشتر از این است. می توانید از آن برای میزبانی اسناد برای نسخه پروژه و میزبان نسخه نرم افزار خود استفاده کنید. همچنین به طور عمده از آن به عنوان یک ابزار مدیریت پروژه و راهی برای اتصال به همکاران خود استفاده خواهید کرد. بیایید در مورد این ویژگی ها بیاموزیم.

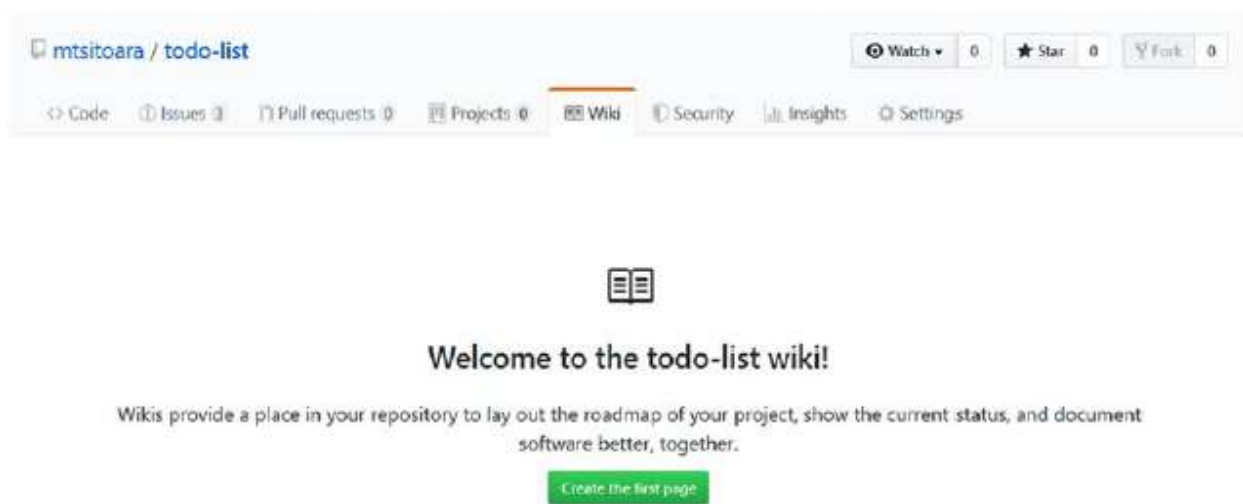
### ویکی

پروژه شما می تواند در رده خود بهترین باشد ، اما اگر افراد دیگر نمی دانند چگونه از آن استفاده کنند یا چگونه کار می کنند ، به جایی نخواهید رسید. به همین دلیل اسناد مهم است ، به خصوص در توسعه نرم افزار. GitHub یک روش خوب برای مستندسازی پروژه شما فراهم می کند: wikis.

ویکی های GitHub عمدتاً به همان روشی که محبوب ترین ویکی جهان است: ویکی پدیا کار می کنند.

هدف آن ارائه اطلاعات عمیق در مورد پروژه شما است: چه کاری را انجام می دهد که چگونه کار می کند چگونه می تواند به کسی کمک کند ...

بیا یک صفحه ویکی برای پروژه خود ایجاد کنیم تا بهتر بتوانیم آن را درک کنیم. فقط به صفحه اصلی پروژه خود بروید و روی "ویکی" کلیک کنید. شما به صفحه نشان داده شده در شکل 18-1 خواهید رسید.



**Figure 18-1.** Wiki homepage

یک دکمه بزرگ تماس با عمل را در صفحه اصلی ویکی مشاهده خواهید کرد ، بنابراین برای ایجاد اولین صفحه ویکی خود روی آن کلیک کنید. وارد صفحه ایجاد صفحه می شوید که در شکل 18-2 نمایش داده شده است.

## Create new page

The screenshot shows a web interface for creating a new page. At the top, there's a title bar with the text "Create new page". Below it is a text input field containing "Home". Underneath is a tabbed interface with "Write" and "Preview" tabs. The "Write" tab is active, showing a rich text editor toolbar with icons for bold, italic, link, unlink, list, quote, and code. To the right of the toolbar is a dropdown menu labeled "Edit mode:" with "Markdown" selected. Below the toolbar is a large text area containing the text "welcome to the todo-list wiki!". At the bottom, there's an "Edit message" section with a text input field containing "Initial Home page". A green "Save Page" button is located at the bottom right.

**Figure 18-2. Creation of a page**

همانطور که می بینید ، این یک نمایش بسیار ساده است که به سه بخش عنوان ، محتوای و پیام ویرایش تقسیم می شود. به عنوان یک عنوان صفحه وب فکر کنید ، بنابراین باید همان استانداردها را رعایت کند: باید واضح و دعوت کننده باشد. محتوا باید درست مانند README.md در Markdown نوشته شود. می توانید ویکی ها را با قالب های دیگر بنویسید ، اما Markdown گزینه پیشنهادی است زیرا بسیاری از ویرایشگران قبلاً از آن استفاده می کنند و خواندن آن بسیار ساده تر است. پیام ویرایش دقیقاً مانند پیام های متعهد است ، توضیحی ساده از تغییرات پیشنهادی شما.

محتوا را در ویکی خود تغییر دهید. به عنوان مثال:

# What is this

This is a simple app to track your daily goals

# Why another TODO app

Because that is never enough TODO apps in the world

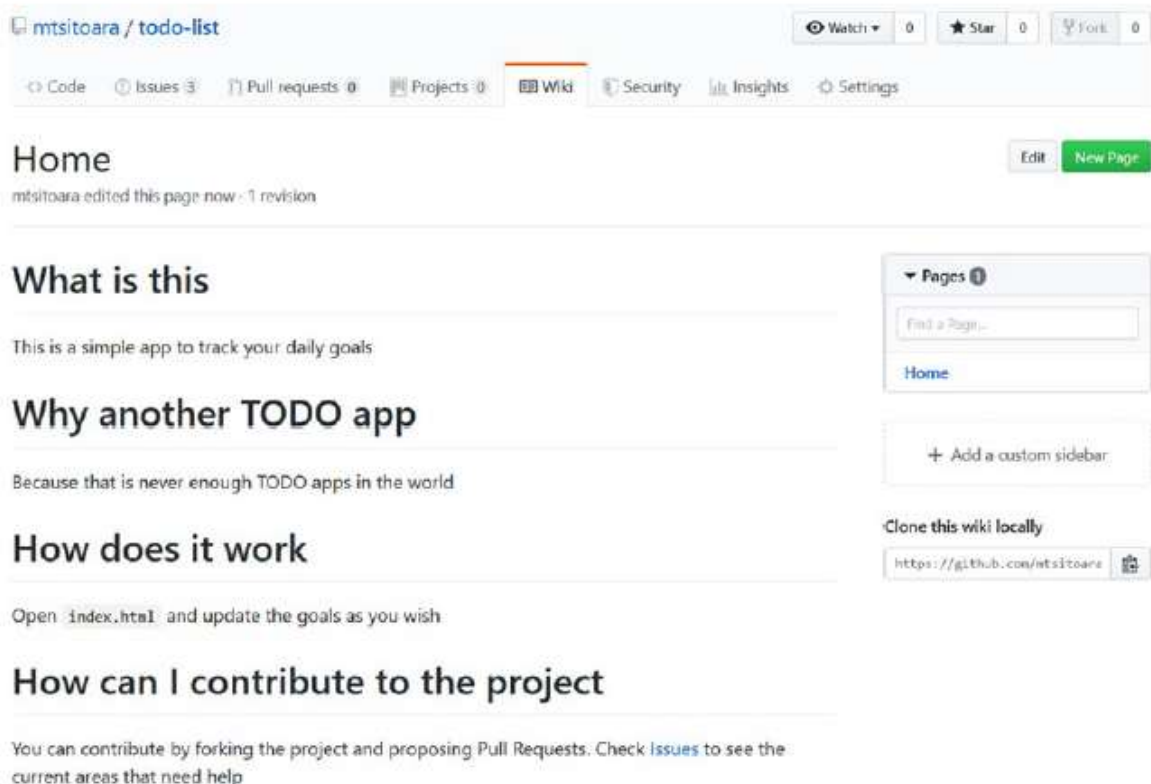
# How does it work

Open `index.html` and update the goals as you wish

# How can I contribute to the project

You can contribute by forking the project and proposing Pull Requests. Check [Issues](https://github.com/mtsitoara/issues) to see the current areas that need help

تغییرات را ذخیره کنید ، و به صفحه اصلی ویکی هدایت می شوید ، که در شکل 18-3 نشان داده شده است.



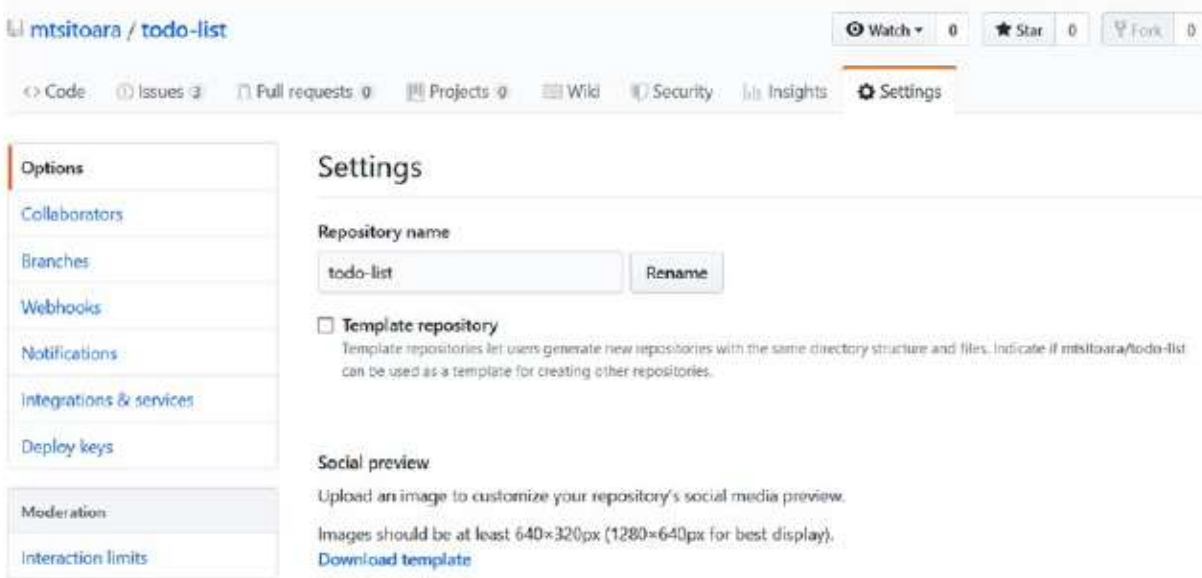
**Figure 18-3.** Wiki homepage showing the newly created wiki

همانطور که می بینید ویکی که تازه ایجاد کرده اید به طور خودکار در صفحه پروژه شما قابل مشاهده است و هر صفحه ایجاد شده در نوار کناری در سمت راست ظاهر می شود. می توانید به همان اندازه صفحات ویکی ایجاد کنید ، اما اطمینان حاصل کنید که آنها قابل درک و مفید هستند. فراموش نکنید که تصاویر و پیوندهای مربوط را اضافه کنید

## صفحات GitHub

به عبارت ساده ، GitHub Pages وب سایتی است که برای شما در GitHub میزبانی شده است. می توانید از آن برای نمایش یک پروژه ، میزبان نمونه کارها خود استفاده کنید ، یا فقط از آن به عنوان نسخه آنلاین رزومه خود استفاده کنید. یک صفحه GitHub می تواند برای حساب شخصی شما (نمونه کارها andresumé) یا برای پروژه های شما (ویترین) باشد. اگر تصمیم دارید از آن برای حساب کاربری خود استفاده کنید ، فقط می توانید یک صفحه ایجاد کنید. اما اگر ویترین پروژه های شما باشد ، می توانید برای هر یک از آنها صفحه ایجاد کنید. برای توضیح بهتر در این زمینه می توانید <https://pages.github.com/> را بررسی کنید.

بباید فرض کنیم می خواهید صفحه ای را برای نمایش پروژه لیست لیست خود ایجاد کنید. ابتدا باید به صفحه پروژه خود برگردید و روی "تنظیمات" کلیک کنید. شما به صفحه نشان داده شده در شکل 18-4 دسترسی خواهید داشت.



**Figure 18-4. Settings page**

به تنظیمات صفحات بروید ، که در شکل 18-5 نشان داده شده است.



## GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

**Source**  
GitHub Pages is currently disabled. Select a source below to enable GitHub Pages for this repository. [Learn more.](#)

None ▾

**Theme Chooser**  
Select a theme to publish your site with a Jekyll theme using the master branch. [Learn more.](#)

Choose a theme

**Figure 18-5.** GitHub Pages settings

گزینه اول یک لیست بازشو شامل محل منبع سن شما است. شما باید صفحه خود را در شاخه اصلی میزبان باشید اما برای فایل های منبع دو مکان دارید. یکی مستقیماً روی استاد است. مورد دیگر در فهرست تحت عنوان "اسناد" قرار دارد. من گزینه دوم را به عنوان واضح تر برای هر بازدید کننده توصیه می کنم. سپس ابتدا باید آن فهرست را ایجاد کنیم. با استفاده از ابزارهای GitHub یا Git ، فایلی به نام index.html را زیر دایرکتوری به نام docs ایجاد کنید. در پرونده ، فقط HTML اساسی بنویسید:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Docs</title>
  </head>
  <body>
    <h1>Docs</h1>
    <p>Example of documentation</p>
  </body>
</html>
```

این مستندات شما خواهد بود. شاخه استاد شما بنابراین باید مانند من باشد که در شکل 18-6 نشان داده شده است.



```
13 lines (12 sloc) 249 Bytes
1 <!doctype html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Docs</title>
6     <link rel="stylesheet" href="style.css" />
7   </head>
8   <body>
9     <h1>Docs</h1>
10    <p>Example of documentation</p>
11  </body>
12 </html>
```

**Figure 18-6.** Docs folder and index.html

سپس می توانیم به صفحه تنظیمات برگردیم و منبع اسناد را انتخاب کنیم. پوشه Docs را به عنوان یک منبع انتخاب کنید و صفحه بارگیری مجدد می کند و پیوندی مانند شکل 18-7 به شما نشان می دهد.

## GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is ready to be published at <https://mtsitoara.github.io/todo-list/>.

### Source

Your GitHub Pages site is currently being built from the `/docs` folder in the `master` branch. [Learn more.](#)

master branch /docs folder ▾

### Theme Chooser

Select a theme to publish your site with a Jekyll theme. [Learn more.](#)

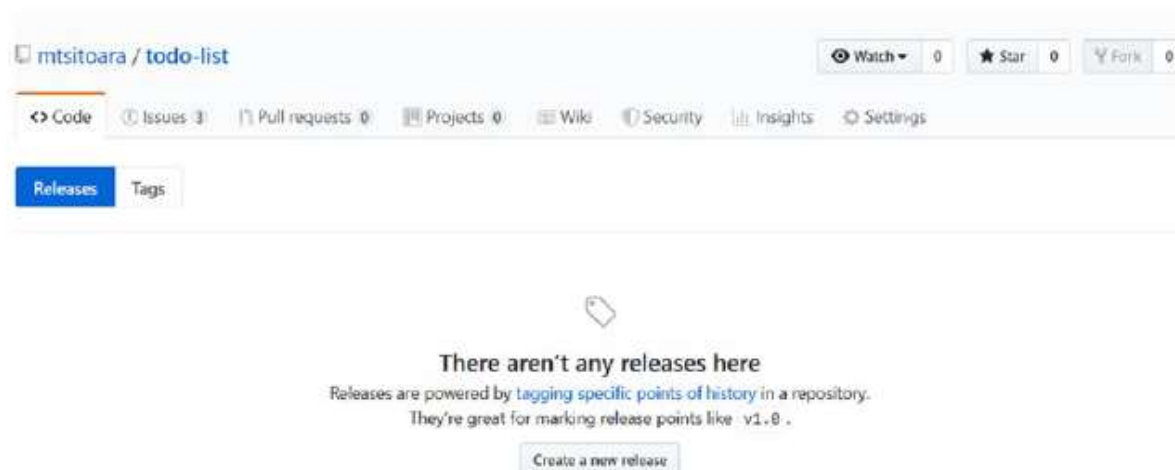
Choose a theme

**Figure 18-7.** Page published

اگر پیوندی را که به شما نشان داده شده است دنبال کنید ، منظره ای باشکوه از صفحه پروژه GitHub خود خواهید دید! پس از آن امکانات می توانند بی حد و مرز باشند زیرا می توانید صفحه خود را مانند سایر صفحات وب سایت ثابت طراحی کنید! اگر سبکی بهتر می خواهید ، <https://jekyllrb.com/> را بررسی کنید. این می تواند به شما در تولید صفحات GitHub در هر زمان کمک کند!

## منتشر شده

پروژه شما نامحدود در توسعه نخواهد ماند؛ باید دیر یا زود آزاد شود و چه سکوی بهتری برای انتشار برنامه خود نسبت به GitHub؟ خیلی راحت است دوباره به صفحه پروژه خود برگردید و روی "انتشار" کلیک کنید. صفحه اصلی نشان داده شده در شکل 18-8 را مشاهده خواهید کرد.



**Figure 18-8.** Releases page

بباید اولین نسخه یمان را ایجاد کنیم! روی دکمه فراخوانی به عمل کلیک کنید ، و نمای ایجاد نسخه ، که در شکل 18-9 نشان داده شده است ، دریافت خواهید کرد.

The image shows the GitHub 'Releases' page for creating a new release. At the top, there are two tabs: 'Releases' (active) and 'Tags'. Below the tabs, there is a 'Tag version' input field and a 'Target: master' dropdown menu. A note says 'Choose an existing tag, or create a new tag on publish'. Below this is a 'Release title' input field. There are two tabs: 'Write' (active) and 'Preview'. The main area is a large text box with the placeholder 'Describe this release'. Below the text box, there is a note 'Attach files by dragging & dropping, selecting or pasting them.' and a small icon. Below this is a large box with a downward arrow and the text 'Attach binaries by dropping them here or selecting them.' At the bottom, there is a checkbox labeled 'This is a pre-release' with a sub-note 'We'll point out that this release is identified as non-production ready.' At the very bottom, there are two buttons: 'Publish release' (green) and 'Save draft' (grey).

**Figure 18-9.** Release creation form

پر کردن فرم ساده و واضح است. نکته اصلی این است که با ریختن آنها روی فرم قبلی، دودوهای نسخه را بارگذاری کنید. از آنجا که برنامه ما در HTML است، اجازه دهید نسخه های فشرده شده شاخه استاد ما را پیوست کنیم. برنامه های قابل نصب، اجرای آن یک باینری خواهد بود. برای ما، فایل های zip و z7 خواهد بود. فراموش نکنید که در صورت نیاز هدف انتشار را تغییر دهید. گزینه پیش فرض شاخه اصلی است اما می توانید به شعبه دیگری یا یک تعهد خاص اشاره کنید! سپس فرم همان شکل خواهد بود که در شکل 10-17 نشان داده شده است.

The image shows a GitHub release creation form. At the top, there are tabs for 'Releases' and 'Tags'. Below them, a version field contains 'v0.1' and a 'Target: master' dropdown. A message states: 'Excellent! This tag will be created from the target when you publish this release.' Below this is a text field for the release name, currently containing 'initial release'. There are 'Write' and 'Preview' tabs. The 'Changes' section has a text area with the content: '- can modify todos by modifying the "index.html" file'. Below the changes is a note: 'Attach files by dragging & dropping, selecting or pasting them.' followed by a file upload area. Two files are listed: 'todo-list.7z' and 'todo-list.zip', both showing '(0.00 MB)' and a delete icon. Below the file list is a note: 'Attach binaries by dropping them here or selecting them.' At the bottom, there is a checkbox for 'This is a pre-release' with a sub-note: 'We'll point out that this release is identified as non-production ready.' Finally, there are 'Publish release' and 'Save draft' buttons.

Releases Tags

v0.1 Target: master

Excellent! This tag will be created from the target when you publish this release.

initial release

Write Preview

Changes:

- can modify todos by modifying the "index.html" file

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

todo-list.7z (0.00 MB) X

todo-list.zip (0.00 MB) X

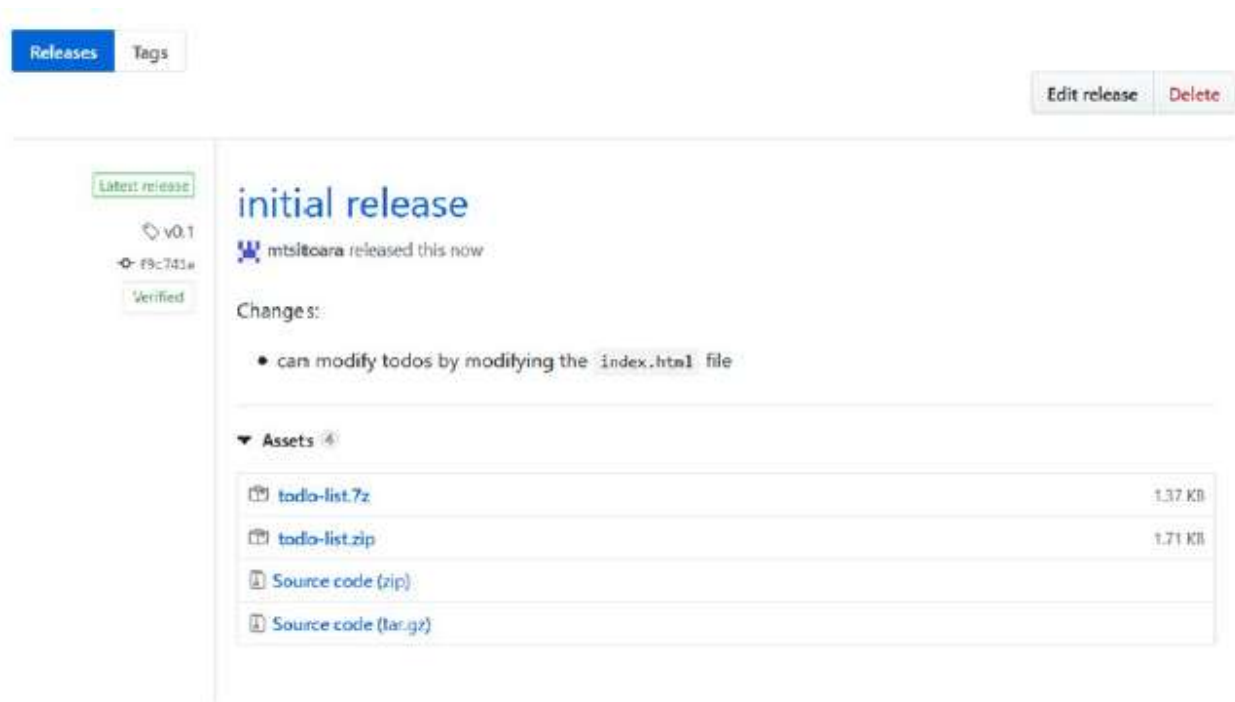
Attach binaries by dropping them here or selecting them.

☐ This is a pre-release  
We'll point out that this release is identified as non-production ready.

Publish release Save draft

**Figure 18-10.** Filled release form with binaries

برای دیدن نتیجه ، روی انتشار کلیک کنید. شما به لیست نسخهها هدایت می شوید و نسخه جدید خود را در آنجا مشاهده خواهید کرد! برای مثال مثلاً شکل 11-18 را بررسی کنید.



**Figure 18-11.** List of all the releases

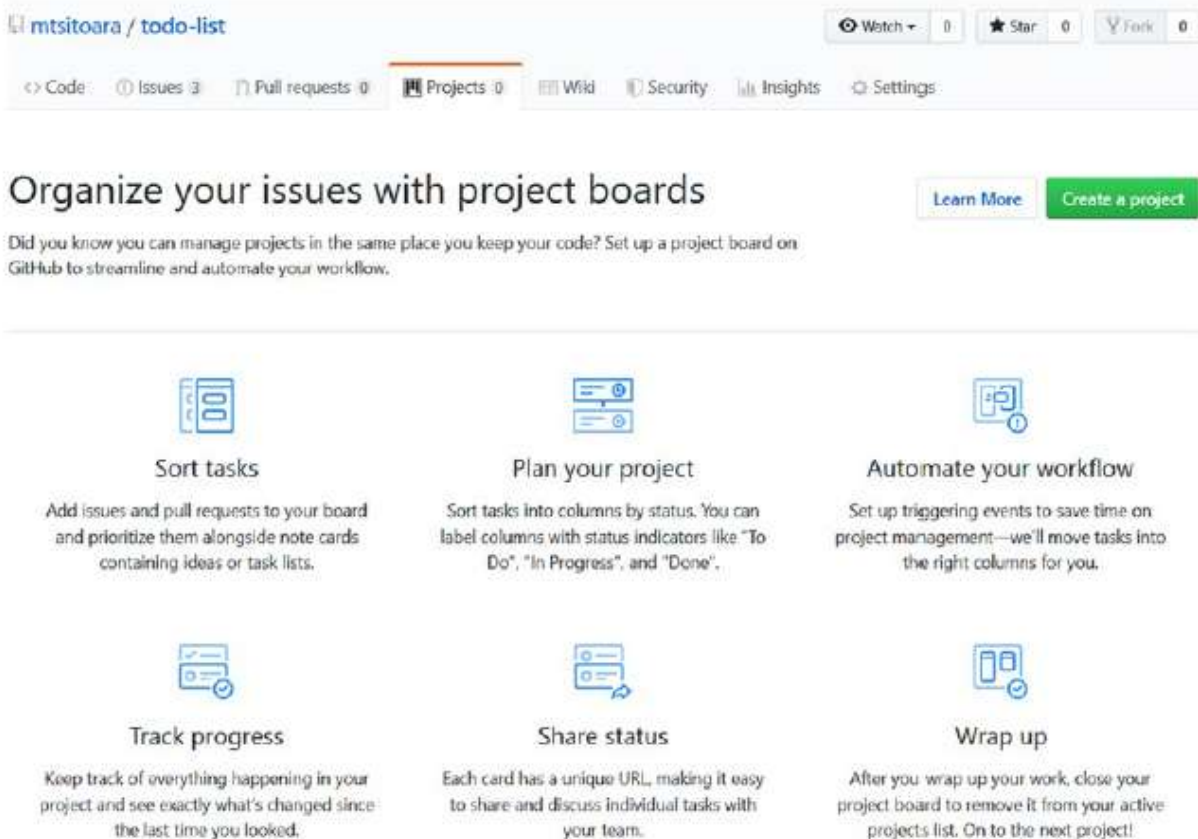
همانطور که مشاهده می کنید، GitHub به طور خودکار کد منبع را با انتشار خود نیز بسته می کند! مراقب باشید هنگام ایجاد نسخه؛ مطمئن باشید همه چیز را به درستی آزمایش کرده و مجدداً آزمایش می کنید!

## تابلوهای پروژه

Project Boards ویژگی بسیار مفیدی از GitHub است زیرا راهی برای پیگیری و سازماندهی پروژه شما فراهم می کند. به عنوان مثال، می توانید برای هر ایده جدیدی که دارید، کارت ایجاد کنید، بنابراین می توانید بعداً در مورد آنها با تیم خود بحث کنید. اما اصلی ترین کاربرد Project Boards پیگیری پیشرفت پروژه شما است. این فراتر از Issues است، زیرا شماره ها فقط یک ویژگی یا اشکال را توصیف می کنند که باید روی آن کار کنید. اما اگر هیئت مدیره روی آن کار کند یا فقط برنامه ای برای اجرای آن وجود دارد، هیئت مدیره پروژه می تواند به شما نشان دهد.

بهترین روش برای درک تابلوهای پروژه آزمایش مستقیم با آنها است. بنابراین به صفحه پروژه خود برگردید و "پروژه ها" را انتخاب کنید. پروژه خالی نشان داده شده در شکل 18-12 را دریافت خواهید کرد.





**Figure 18-12.** Projects main page

صفحه اصلی پروژه هنوز خالی است زیرا ما هیچ پروژه ای را ایجاد نکرده ایم. همچنین شرایط مختلفی را برای شما نشان می دهد که مایل به استفاده از تابلوهای پروژه هستید. برای ادامه روی "ایجاد یک پروژه" کلیک کنید. نمایی که در شکل 13-18 نشان داده شده است را دریافت خواهید کرد.

## Create a new project

Coordinate, track, and update your work in one place, so projects stay transparent and on schedule.

### Project board name

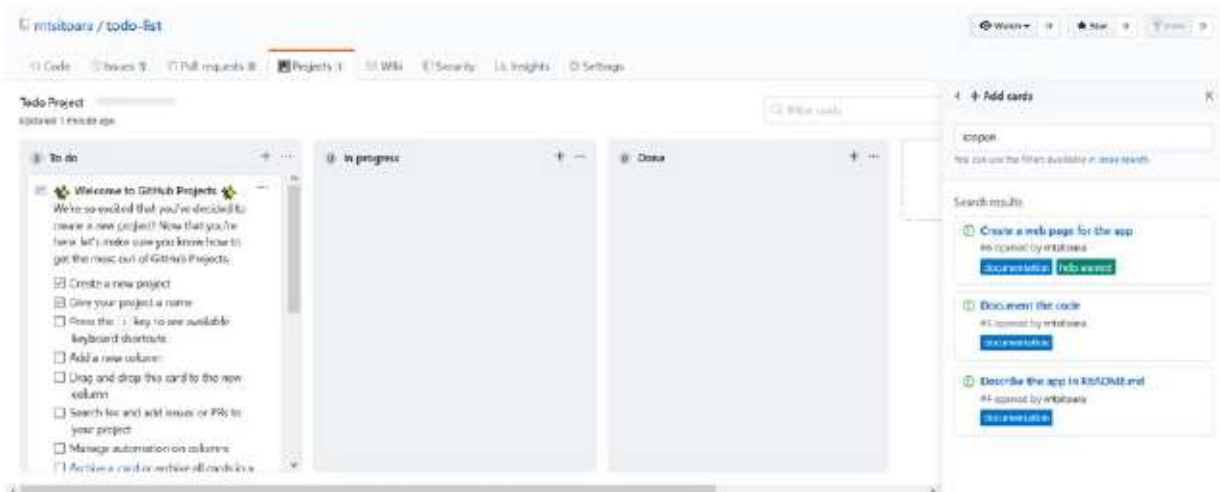
### Description (optional)

### Project template

Save yourself time with a pre-configured project board template.

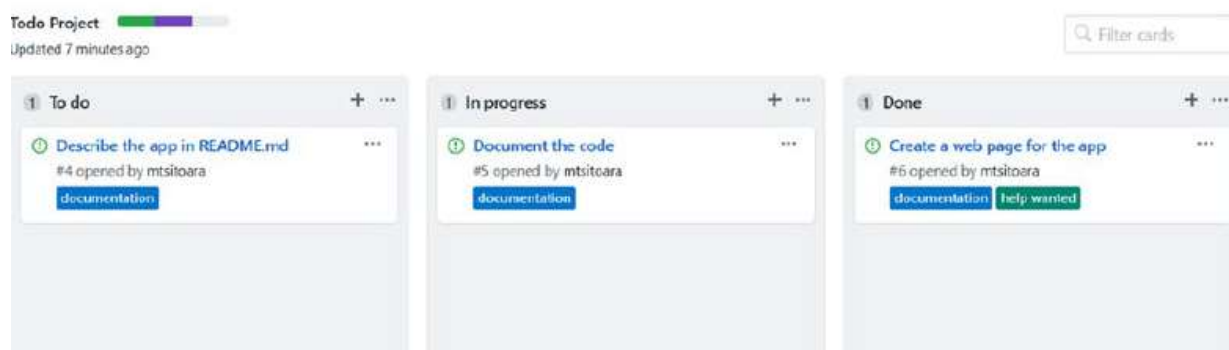
**Figure 18-13.** Creation of a project

باز هم ، این یک شکل بسیار ساده است اما توجه خود را به الگوی هدایت کنید: بسیار مهم است. به عنوان یک مبتدی ، باید از الگوی BasicKanban استفاده کنید زیرا نمونه اولیه آن است. می توانید خودتان تابلوها را ایجاد کنید ، اما در حال حاضر ، بایید به اصول اولیه پردازیم. پروژه را ایجاد کنید ، و صفحه نیمه خالی نشان داده شده در شکل 14-18 را مشاهده خواهید کرد.



**Figure 18-14.** New project created

همانطور که مشاهده می کنید ، سه صفحه ایجاد شده است: "برای انجام" ، "در حال انجام" و "انجام شد". دقیقاً مثل برنامه ما! در سمت راست صفحه می توانید لیستی از مشکلات باز ما را مشاهده کنید. آن موضوعات را به صفحه مربوطه بکشید و رها کنید. در صفحه "برای انجام" ، شما یک مثال کوچک از آنچه می توانید با هیئت های خود انجام دهید وجود دارد. این نه تنها برای شماره ها بلکه برای درخواست های کشیدن یا یادداشت های ساده است. پس از قرار دادن شماره های خود در صفحه مورد نظر ، نتیجه ای مانند شکل 15-17 دریافت خواهید کرد.

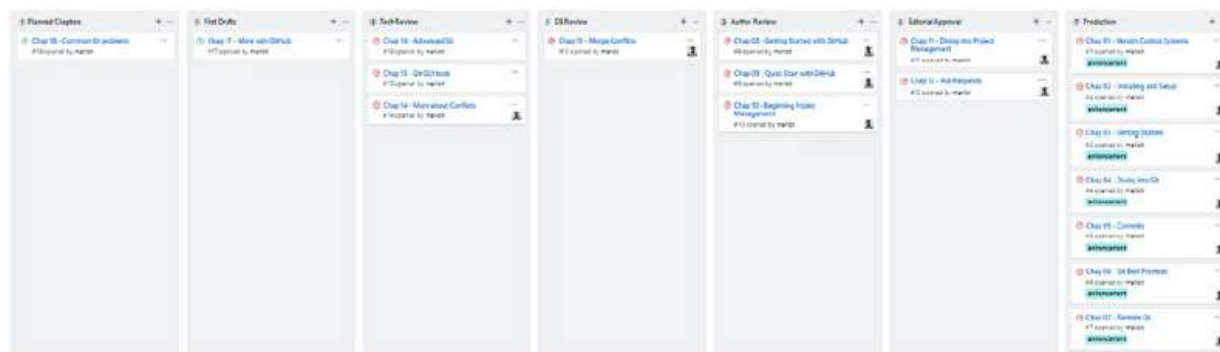


**Figure 18-15.** Our first Project Boards

یک پاداش اندک: با حرکت دادن شماره ها در صفحه ، نوار رنگی نزدیک نام پروژه تغییر خواهد کرد. این یک راه خوب برای پیگیری پیشرفت شما است!

اما تابلوهای Project بیش از یک ردیاب پیشرفت پروژه هستند! شما می توانید تابلوهای پروژه را برای بسیاری از موقعیت ها ایجاد کنید: ردیابی انتشار ، یادداشت های جلسه ، یادداشت ایده های توسعه دهنده ، بازخورد کاربر ... شما می توانید در شکل 16-17 صفحه پروژه برای این کتاب پیدا کنید که می توانید در

<https://github.com/mariot/boky/project/1>



**Figure 18-16.** Current Project Board of this book

من به شما توصیه می کنم از پروژه های تخته برای پروژه های آینده خود استفاده کنید زیرا داشتن دیدگاه روشنی از پیشرفت شما راهی مطمئن برای موفقیت است. اگر احساس بی حوصلگی می کنید ، می توانید کانبان خودکار را نیز بررسی کنید که به طور خودکار کارت را برای شما جابجا می کند! به عنوان مثال ، هر شماره جدید تحت عنوان "To Do" پر خواهد شد و هر شماره بسته به "انجام می شود" منتقل می شود.

## خلاصه

این فصل ما را برای لحظه ای کمی از Git دور کرد و ما روی GitHub تمرکز کردیم. ما دیده ایم که GitHub بیش از یک فروشگاه برای کد شماست ، اما یک ابزار کامل برای مدیریت و انتشار پروژه شما است. بعد از این فصل ، شما باید بتوانید مینی وب سایت را ببوشید و کمی از آن مستندات داشته باشید. همچنین باید اولین نسخه از برنامه خود را داشته باشید. مهمترین ویژگی که در ابتدای صفحه نشان داده شده است تخته Project است. از آنها استفاده کنید تا دید روشنی نسبت به آنچه انجام داده اید و به کجا می روید. به نظر می رسد آنها ساده هستند اما در مدیریت پروژه بسیار مفید هستند. اکنون به اصول اولیه Git و GitHub تسلط پیدا کرده اید. اما هنوز راه های موانع در مسیر شما وجود دارد: شما هنوز از آنچه در انتظار شما در محیط دنیای واقعی است مطمئن نیستید. در فصل بعد ، مشکلاتی را که مطمئناً هنگام کار با دیگران با آنها روبرو خواهید شد و چگونگی حل آنها را بررسی خواهیم کرد. گوش به زنگ باشید!

# فصل نوزدهم

## مشکلات رایج Git

ما در مورد ویژگی های اصلی و پیشرفته Git و چه زمانی از آنها استفاده کردیم. اما از آنجا که ما فقط انسان هستیم ، در سفر Git ما با مشکلات زیادی روبرو خواهیم شد. بیشتر این مشکلات نتیجه ناخواسته است ، بنابراین فقط آگاهی از وجود آنها گامی بزرگ در جهت جلوگیری از آنها است. اما اگر هنوز هم آنها را اجرا کردید ، در اینجا بهترین راه حل ها هستند!

### مخزن

مخزن ستون فقرات تجربه Git شما است. همه چیز از همان جا شروع می شود و به پایان می رسد. بسیار دشوار است ، اما با کمترین احتمال اتفاق بد رخ می دهد ، در اینجا چند نکته وجود دارد.

### شروع به کار

این بنیادی ترین "راه حل" در فصل است ، و امیدوارم که شما هرگز از آن استفاده نکنید. این راه حل اساساً راهی برای پاک کردن همه چیز و شروع آن است! این تنها زمانی باید گزینه ای باشد که مخزن از راه دور داشته باشید و بخواهید محلی خود را به دلایلی حذف کنید. دلایل انجام این کار شامل موارد زیر است

- تغییر رایانه های کاری
  - بخش های غیرقابل خواندن در هارد دیسک
  - خطاهای غیرقابل بازگشت در فهرست "git."
- برای شروع کار ، فقط باید مخزن راه دور را با دستور git-clone کlon کنید:

```
$ git clone <repository_location>
```

محل مخزن ، لینک HTTPS یا SSH به مخزن از راه دور شما است. می توانید آن را در صفحه پروژه GitHub خود پیدا کنید. کلونینگ همان تأثیرگذاری اولیه یک مخزن اما دارای یک جایزه بزرگ است: تمام تاریخچه و تعهدات نیز در مخزن محلی جدید شما کپی می شوند. و دیگر نیازی به پیوند اصلی ندارید.

## تغییر مبدأ

در شرایط عادی می خواهید نشانی اینترنتی مخزن از راه دور در طول توسعه خود یکسان باشد. اما شرایط خاصی وجود دارد که تغییر آن ضروری است:

- جابجایی بین پیوندهای HTTPS و SSH
  - انتقال مخزن به میزبان دیگر
  - اضافه کردن مخزن اختصاصی برای انتشار یا آزمایش
- اول ، اجازه دهید اطلاعات بیشتری درباره راه دورهای فعلی ما کسب کنیم. بنابراین ، از گزینه git large با گزینه "v-" استفاده کنید.

```
$ git remote -v
```

این لیستی از راه دورهای فعلی خود را به شما می دهد همانطور که در شکل 19-1 نشان داده شده است.



```
MINGW64: c:/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ git remote -v
origin https://github.com/mtsitoara/todo-list.git (fetch)
origin https://github.com/mtsitoara/todo-list.git (push)
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (separate-code-and-styles)
$ |
```

**Figure 19-1.** List of current remotes

در اینجا ، ما فقط یک "منشأ" از راه دور داریم که به یک GitHub HTTPSlink اشاره می کند. برای تغییر این لینک ، شما باید از زیرمجموعه ی set-URL استفاده کنید:

```
$ git remote set-url <remote_name> <remote_url>
```



به عنوان مثال ، اگر من برای دسترسی به GitHub می خواستم به جای HTTPS به SSH تغییر دهم ، من اجرای

```
$ git remote set-url origin git@github.com:mtsitoara/todo-list.git
```

انجام این کار به من امکان می دهد بدون ارائه نام کاربری و رمز ورود خود ، به GitHub منتقل و پیوندید. تأیید اعتبار توسط دو مجموعه کلید انجام می شود: یک کلید خصوصی که من در رایانه محلی خود نگه می دارم و یک کلید عمومی که باید آن را بارگذاری کنم. به GitHub. اگر علاقمند به استفاده از SSH برای تأیید هویت خود هستید ، لطفاً به این قسمت بروید

بسته به OperatingSystem خود (<https://help.github.com/fa/articles/which-remote-url-should-i-use>) برای اطلاعات بیشتر به GitHub کمک می کند. اگر تصمیم دارید از HTTPS استفاده کنید اما باید رمز ورود را ذخیره کنید تا مجبور نباشید آن را همیشه تایپ کنید ، می توانید از یک یاور معتبر استفاده کنید. باز هم ، بسته به سیستم عامل شما (<https://help.github.com/fa/articles/caching-your-github-password-in-git>) اطلاعات بیشتری در مورد این مورد در مورد کمک GitHub وجود دارد.

## دایرکتوری کار

شما بیشتر وقت خود را در فهرست کار می گذرانید ، و در اینجا ، چیزهای زیادی وجود ندارد که بتوانید آنها را بشکنید.

## Git diff خالی است

این خیلی زیاد پیش می آید اما خطرناک نیست. بعضی اوقات ، تغییرات زیادی ایجاد کرده اید و می خواهید تغییرات را بررسی کنید. اما وقتی git diff را اجرا می کنید ، نتیجه خالی است. وحشت نکنید! Git diff فقط فایل های اصلاح شده را نشان می دهد ، بنابراین اگر پرونده شما مرحله بندی شده است ، آن را در آنجا مشاهده نمی کنید. برای دیدن تغییرات انجام شده در پرونده های مرحله بندی شده ، باید اجرا کنید:

```
$ git diff --staged
```

## تغییر در یک پرونده را لغو کنید

وقتی از Git استفاده خواهید کرد ، این امر بسیار زیاد خواهد شد. بعضی اوقات ، شما فقط می خواهید بدون نیاز به بررسی کامل یک پرونده ، به پرونده قبلی برگردانید و سپس کد را کپی کنید. ما قبلاً این فرمان را دیده ایم:

```
$ git checkout <commit_name> -- <file_name>
```

این دستور پرونده را مطابق آنچه که در زمان اجرای تعهد بود بررسی می کند و به این ترتیب ، دایرکتوری کاری خود را تغییر می دهد که هیچ گونه تغییر ناپذیری را از دست ندهید!

## Commits

بیشتر مشکلات هنگام بروز تلاش برای اجرای پروژه فعلی شما بوجود می آیند. اما نگران نباشید ، همیشه یک راه حل ساده برای این نوع مشکلات وجود دارد. مهمترین نکته ای که باید در نظر بگیرید این است: آیا دستوراتی که از آنها استفاده می کنید مخرب هستند؟ دستوراتی مانند تنظیم مجدد یا بررسی کردن دایرکتوری کار خود را تغییر دهید ، بنابراین لطفاً اطمینان حاصل کنید که قبل از اجرای آنها می دانید که چه کاری انجام می دهید.

## Error in commit

این یک خطای اساسی در Git است. بعد از انجام کار سخت ، گاهی متوجه خواهید شد که یک خطای دستوری کمی به پیام متعهد شما رسیده است یا اینکه شما آن را برای تشکیل پرونده فراموش کرده اید. راه حل این مشکلات اصلاح تعهد است ، به این معنی که شما تعهد فوری را فسخ کرده و یک مورد جدید ایجاد خواهید کرد. دستور ساده است:

### \$git commit --and

نام تعهد تغییر خواهد کرد زیرا شما اساساً محتوای آن را تغییر می دهید. این بدان معنی است که شما نباید تعهدی را که قبلاً به شعبه ای از راه دور منتقل کرده اید اصلاح کنید ، به ویژه اگر شخص دیگری در آن شاخه کار کند. این در حال بازنویسی تاریخ است و شما هرگز نباید این کار را انجام دهید.

با این کار ، اگر تعهد خود را تحت فشار قرار داده و در شعبه تنها باشید ، می توانید تعهدی را اصلاح کرده و دوباره سعی کنید آن را فشار دهید. اما از آنجا که نام متعهد تغییر کرده است ، Git به شما امکان نمی دهد تاریخ را بدون جنگ تغییر دهید. شما باید تمام تاریخچه را در شاخه راه دور پاک کنید و آن را جایگزین خود کنید ، به این معنی که همه چیز را در شاخه راه دور رونویسی خواهید کرد. به همین دلیل است که اگر تنها در یک شعبه نیستید ، هرگز نباید تعهد را اصلاح کنید. برای فشار دادن یک شعبه با تعهدات اصلاح شده ، شما این قدرت را دارید.

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

گزینه "-f" را مجبور می کند تا همه چیز را در شاخه راه دور رونویسی کند و آن را با سابقه شاخه فعلی خود جایگزین کند.

تعهدات اصلاح فقط باید در صورت استفاده از پیام متعهد یا افزودن / حذف پرونده مورد استفاده قرار گیرد. تعهدی برای تغییر کد اصلاح نکنید.

## Undo commits

اگر به یک شعبه متعهد شده اید اما متوجه شده اید که این اشتباه است ، می توانید آن را خنثی سازی کنید ، اما تنها وقتی به شعبه ای از راه دور هل نداده اید. دستور ساده اما خطرناک است: این دستور تنظیم مجدد است. اما برخلاف تنظیم مجدد "سخت" که در آن همه چیز پاک شده است ، برای خنثی کردن تعهد اما حفظ تغییرات لازم است به تنظیم مجدد "نرم" ایجاب کنید.

```
$ git reset HEAD~ --soft
```

سپس این تعهد ناپدید می شود و گزینه ای را برای حذف تغییرات و اعمال آنها در شعبه دیگر از بین می برد. باز هم ، این در حال بازنویسی تاریخ است و اگر قبلاً به شعبه ای از راه دور سوق داده باشید نباید از آن استفاده شود.

### شاخه ها

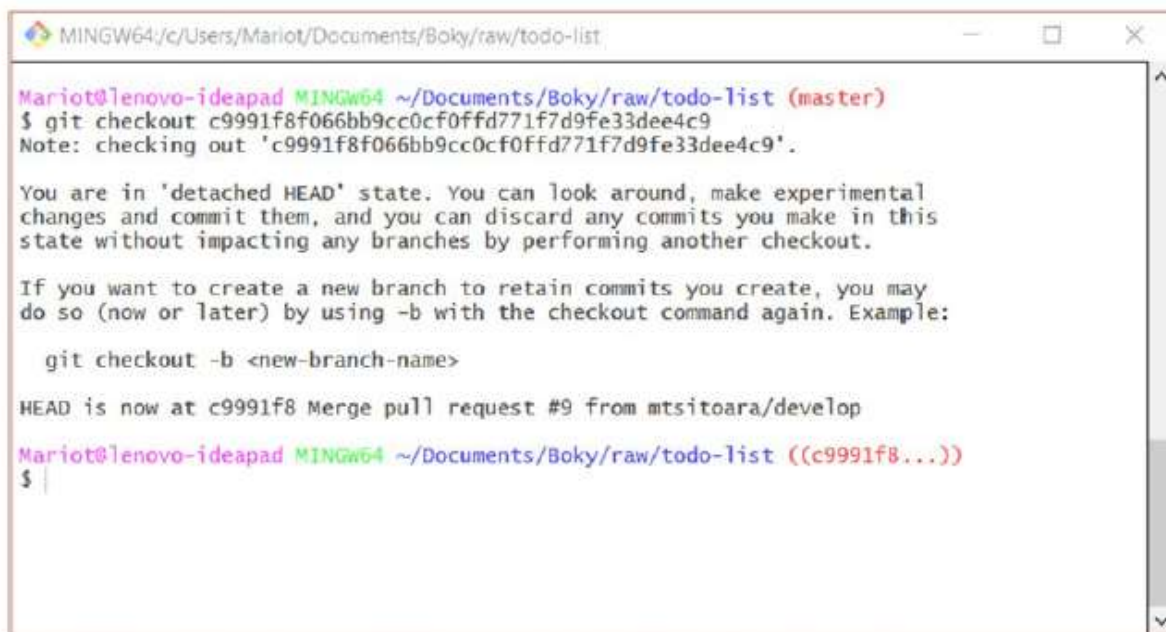
برای داشتن یک گردش کار بهینه ، باید با شعب زیادی کار کنید. هنگام کار روی یک ویژگی جدید یا رفع اشکال ، اولین گزینه شما باید ایجاد یک شاخه باشد. اما هرچه از شاخه ها راحت تر باشید ، احتمالاً جزئیاتی را فراموش خواهید کرد که می تواند منجر به مشکلات شود. در اینجا متداول ترین مشکلاتی که با Git روبرو می شوید وجود دارد.

## Detached HEAD

HEAD یک مرجع به تعهد بررسی شده فعلی است ، به این معنی که والدین نسبت به تعهدی که در آینده ایجاد خواهید کرد متعهد هستند. معمولاً HEAD به آخرین تعهد شعبه فعلی اشاره دارد. و تمام شعب و تعهدات آینده آن را به عنوان والدین خواهند داشت.

هنگامی که شعب را چک می کنید ، رئیس بین آخرین تعهدات شعب به عقب و جلو خواهد رفت. اما وقتی تعهد خاصی را پرداخت می کنید ، به وضعیتی با نام "جدا شده HEAD" وارد می شوید و این بدان معنی است که شما در وضعیتی قرار دارید که هیچ چیزی را ایجاد نمی کنید به هر چیزی وصل شود. بنابراین بی فایده است که سعی کنید در طی آن حالت مرتکب شوید زیرا هرگونه تغییر از بین می رود.

Git هنگامی که در آن حالت قرار دارید به شما می گوید (مانند شکل 19-2) بنابراین هرگز نمی توانید در آن حالت ناآگاه باشید.

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 (master)'. The user enters '\$ git checkout c9991f8f066bb9cc0cf0ffd771f7d9fe33dee4c9'. The output shows a note about checking out the commit, followed by instructions on the 'detached HEAD' state. It then shows the command 'git checkout -b <new-branch-name>' and the result 'HEAD is now at c9991f8 Merge pull request #9 from mtsitoara/develop'. The prompt changes to 'Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list ((c9991f8...))' and the user enters '\$'.

**Figure 19-2. Checking out a commit**

بنابراین بررسی یک تعهد فقط برای آزمایش چیزی روی نرم افزار شما لازم است. اگر می خواهید تعهدی را که قصد انجام آن را دارید ، بتوانید شعبه ای از آن تعهد خاص ایجاد کنید. دستور همان ایجاد شعبه از شاخه دیگر است:

```
$ git checkout -b <branch_name>
```

## روی شاخه اشتباه کار کردن

این اتفاق زیادی می افتد. اوضاع معمولاً به این صورت است: شما یک کار دریافت می کنید و آنقدر مشتاق هستید که آن را تکمیل کنید و بلافاصله شروع به کدنویسی می کنید ، وقتی متوجه می شوید که همه را روی شاخه استاد کار می کردید ، نگران نباشید ، حل این مسئله بسیار ساده است. اگر برخی از پرونده ها را در شاخه اشتباه اصلاح کردید ، می توانید مستقیماً یک شاخه جدید ایجاد کنید (و آن را بررسی کنید) تا تغییرات فعلی را در آنجا انجام دهید. دوباره همان دستور است:

```
$ git checkout -b <branch_name>
```

با ایجاد تغییرات فعلی یک شاخه جدید ایجاد می کنید و آن را بررسی می کنید. سپس می توانید پرونده های تغییر یافته خود را مرحله بندی کرده و پروژه را انجام دهید. با این حال ، اگر قبلاً شعبه را به یک مخزن از راه دور منتقل کنید ، این کار نخواهد کرد. تاریخ ، تاریخ است ، آن را تغییر ندهید. تنها راه حل این است که تعهدی را که فشار می دهید برگردانید و در تمام زندگی با آن شرم زندگی کنید.

## با شعبه والدین همگام شوید

هنگامی که شاخه ای از شاخه دیگری (معمولاً استاد) ایجاد می کنید ، تاریخچه آنها به هم پیوسته نمی شوند ، بنابراین آنچه در یک شاخه اتفاق می افتد ، هیچ تاریخچه ای در دیگری ندارد. این بدان معناست که در حالی که شما در شاخه خود کار می کنید ، افراد دیگر می توانند به این شعب متعهد شوند. و آن تعهدات در اختیار شعبه شما نخواهد بود.

اگر هنوز شاخه خود را مشغول به کار هستید اما علاقه مند به انجام این تعهدات جدید در شاخه پایه هستید ، ابتدا باید یک صفحه تمیز داشته باشید ، به این معنی که باید پروژه خود را انجام دهید (یا تغییرات فعلی خود را خنثی کنید).

سپس باید شعبه والدین را بررسی کنید ، تعهدات جدید را بکشید و دوباره به شعبه خود برگردید.

```
$ git checkout master  
$ git pull origin master  
$ git checkout <branch_name>
```

با خیال راحت در شعبه محلی خود ، می توانید شاخه والدین را جلب کنید. مفهوم ساده است: Git تعهدات فعلی شما را برمی دارد و یک شاخه جدید از نوک شاخه والدین ایجاد می کند. تعهدات شما سپس به شعبه جدید شما اعمال می شود. این مانند شما خواهد بود که شاخه ای از آخرین تعهد شعبه استاد ایجاد کنید. دستور "rebase" گفته می شود.

```
$ git rebase master
```

تعهدات master ممکن است در شاخه شما درگیری ایجاد کند ، بنابراین آماده شوید تا دستانتان کثیف شود. حل و فصل این درگیری ها با یکدیگر ادغام می شود همان چیزی است که قبلاً مشاهده کردیم: هر پرونده متضاد را باز کنید و کد مورد نظر را برای نگه داشتن انتخاب کنید. سپس می توانید آنها را stage کنید و commit شوید.

می توانید نمونه ای از تقابل تخفیف را در شکل 19-3 پیدا کنید ، که بر روی آن استاد و test\_branch هر دو اصلاح شده README.md انجام می شود.



```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (test_branch)
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: another commit
Using index info to reconstruct a base tree...
M   README.md
Falling back to patching base and 3-way merge...
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
error: Failed to merge in the changes.
hint: Use 'git am --show-current-patch' to see the failed patch
Patch failed at 0001 another commit
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (test_branch|REBASE 1/1)
$ |
```

**Figure 19-3.** Merge conflict during rebase

همانطور که می بینید ، تقریباً دقیقاً مانند هر درگیری ادغام است. و قطعاً همان است:

```
$ git add <conflicted_files>
$ git rebase --continue
```

در اینجا نیز اگر به خاطر درگیری احساس شجاعت زیادی ندارید ، می توانید تخفیف را متوقف کرده و به حالت اولیه برگردید.

```
$ git rebase --abort
```

اگر مدت طولانی در یک شعبه کار می کنید ، ایده خوبی است که هر از چندگاهی از این کار دوباره سرکشی کنید ، بنابراین شما نیز خیلی پشت شاخه والدین نیستید. البته ، می توانید با درگیری های ادغام روبرو شوید ، اما احتمالاً احتمال بیشتری وجود دارد که بیشتر به نظر برسید.

و اگر ترس از درگیری را به تأخیر انداختید ، فقط خود را برای ناکامی ها تعیین می کنید ، زیرا این درگیری ها دوباره ظاهر می شوند ، اگر به هر حال سعی کنید شاخه ها را ادغام کنید. بهتر است هر از گاهی با درگیری های کوچک با تخفیف مقابله کنید تا مجبور شوید بسیاری از پرونده های متناقض را همزمان با هم ادغام کنید.

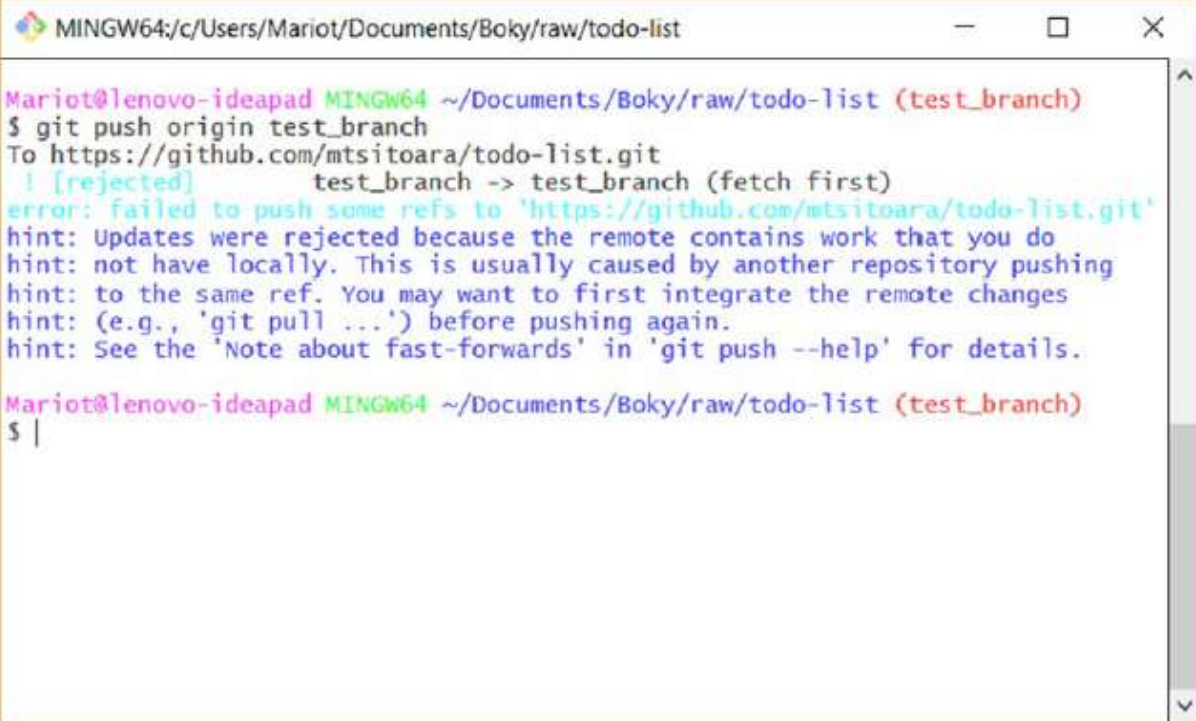


## شعب گوناگون شده اند

اگر از گردش کار بد Git استفاده می کنید ، برای شما اتفاق می افتد. همانطور که قبلاً گفتیم ، شما باید برای حل و فصل یک مسئله روی شاخه خود کار کنید ، زیرا چندین نفر مرتکب در همان شعبه ، دستور العمل مناسب برای فاجعه است.

ما می گویم که دو شاخه تغییر کرده اند که به دلیل تغییر تاریخ دیگر نمی توانید به شعبه راه دور خود فشار بیاورید. این اتفاق می افتد زمانی که شما به شعبه محلی خود متعهد شده اید ، اما افراد دیگر قبل از شما تعهدات خود را روی شاخه راه دور سوق داده اند.

اکنون وقت فشار آوردن ، Git به شما اجازه نمی دهد زیرا آخرین تعهد شعبه از راه دور بخشی از تاریخ محلی شما نیست. شما مانند خطایی که در شکل 19-4 نشان داده شده است خطایی دریافت خواهید کرد.

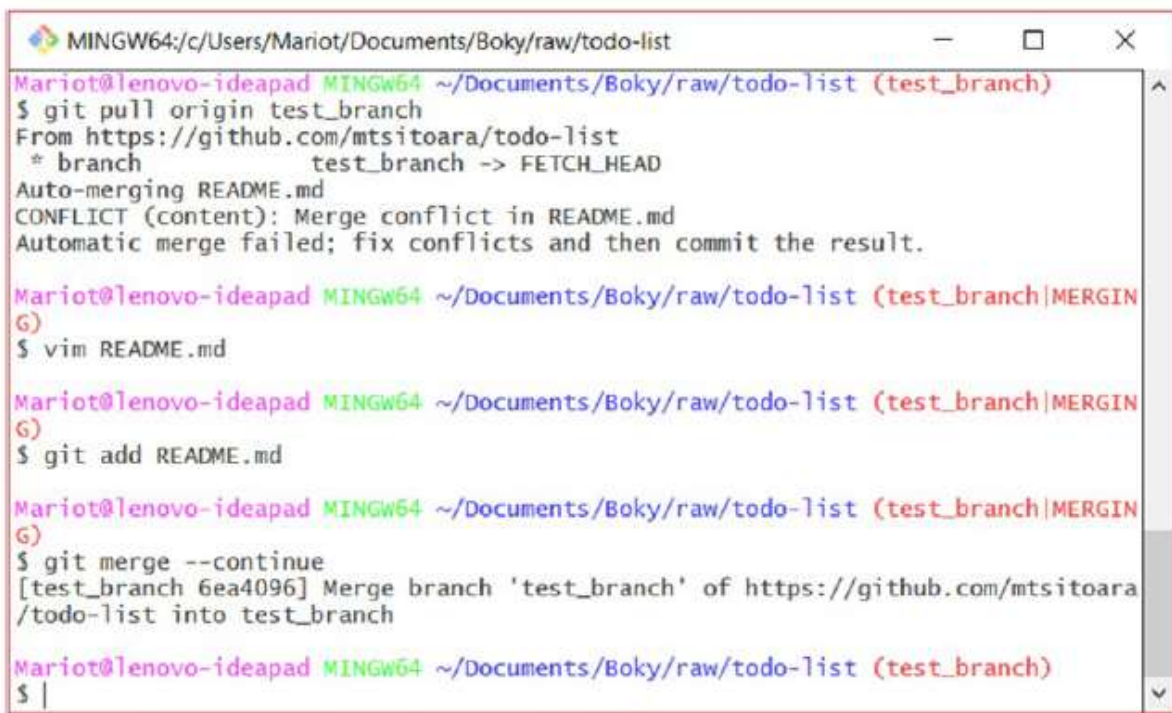
A screenshot of a terminal window titled 'MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list'. The prompt is 'Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/todo-list (test\_branch)'. The user enters '\$ git push origin test\_branch'. The output shows the push attempt to 'https://github.com/mtsitoara/todo-list.git', which is rejected. The error message states: 'error: failed to push some refs to 'https://github.com/mtsitoara/todo-list.git''. It includes hints: 'Updates were rejected because the remote contains work that you do not have locally. This is usually caused by another repository pushing to the same ref. You may want to first integrate the remote changes (e.g., 'git pull ...') before pushing again. See the 'Note about fast-forwards' in 'git push --help' for details.' The prompt returns to '\$ |'.

**Figure 19-4.** Rejected changes

معقول ترین راه حل در اینجا است: تعهدات مربوط به شاخه راه دور را بکشید و تغییرات خود را ادغام کنید. پس از آن شما می توانید تغییرات خود را در تاریخ خود داشته باشید (پس از حل اختلافات ادغام نهایی) و می توانید پس از آن فشار وارد کنید.

```
$ git pull origin <branch_name>
$ git push origin <branch_name>
```

این به شما یک سابقه زشت تاریخ می دهد ، اما حداقل همه تعهدات ذخیره می شوند. نمونه ای از این در شکل 19-5 نشان داده شده است.

A screenshot of a terminal window titled 'MINGW64:/c/Users/Mariot/Documents/Boky/raw/todo-list'. The terminal shows the following commands and output:  
1. `$ git pull origin test_branch`  
Output: `From https://github.com/mtsitoara/todo-list`  
`* branch test_branch -> FETCH_HEAD`  
`Auto-merging README.md`  
`CONFLICT (content): Merge conflict in README.md`  
`Automatic merge failed; fix conflicts and then commit the result.`  
2. `$ vim README.md`  
3. `$ git add README.md`  
4. `$ git merge --continue`  
Output: `[test_branch 6ea4096] Merge branch 'test_branch' of https://github.com/mtsitoara/todo-list into test_branch`  
5. `$`

**Figure 19-5.** Merge local and remote branch

راه حل دیگر خطرناک تر است: همه چیز را در شاخه راه دور بازنویسی کنید و تاریخ آن را جایگزین کنید. برای این کار ، باید با استفاده از گزینه "force" فشار بیاورید.

```
$ git push origin <branch_name> -f
```

این منجر به تعهدات گمشده و کشمکش می شود. هرگز این کار را نکنید. باز هم ، اگر از یک گردش کار خوب Git و GitHub استفاده کنید ، این اتفاق نمی افتد.

## خلاصه

این فصل را نوشتیم تا شما را در هنگام مواجهه با مشکلات معمول Git به راه حل صحیح بیان کنید. مطمئناً، مشکلات جدیدتر و دشوارتری را کشف خواهید کرد، اما این روش خوبی برای شروع است. نکته اصلی که باید به خاطر بسپارید این است که همیشه قبل از انجام هر کاری، خصوصاً ارتکاب، بررسی کنید که کجا هستید.

اما اگر از گردش کار مشترک Git and GitHub استفاده می کنید، به هیچ وجه ظاهر نمی شود. بنابراین، اجازه دهید در فصل بعد دوباره کشف کنیم. ما قبلاً در مورد این مورد در فصل های قبل صحبت کرده ایم، اما زمان آن رسیده است که بعد از مشاهده همه ویژگی های کاربردی ترین Git و GitHub، آن را مرور کنیم.

TopLearn.com

# فصل بیستم

## گردش کاری Git و GitHub

ما در آخرین فصل ها ، بخصوص در مورد جنبه های فنی گیت ، چیزهای زیادی آموخته ایم. اکنون می دانید که چگونه پروژه خود را به درستی نسخه کنید و چگونه با مشکلات احتمالی مواجه شوید. ما همچنین با GitHub خیلی به اصول مدیریت پروژه توجه کردیم.

اکنون زمان آن است که همه این موارد را در چشم انداز خود قرار داده و برنامه بازی کاملی را برای پروژه خود تهیه کنید. در این فصل ، شما با یک کار گردشگری با دقت ارائه شده است که شما باید برای یک پروژه موفق دنبال کنید. شما می توانید از آن به عنوان یک بخش "بهترین شیوه ها" یا یک راهنمای "چگونه-به" استفاده کنید.

### نحوه استفاده از این گردش کار

گردش کار ارائه شده در این فصل برای مبتدیان و کاربران با تجربه طراحی شده است. همچنین زمان زیادی در پروژه های منبع باز استفاده می شود ، بنابراین بسیاری از توسعه دهندگان از آن استفاده می کنند. به خاطر داشته باشید که این گردش کار به هیچ وجه در سنگ تنظیم نشده است و با توجه به دلایل ممکن ، می توان آن را جلب کرد.

پیشنهاد من این است که وقتی هنوز تازه کار هستید ، از جریان کار به صورت مذهبی پیروی کنید تا بتوانید نحوه عملکرد آن را بدست آورید و آیین های مختلفی را که باید طی کنید ، انجام دهید. وقتی کمی احساس تجربه می کنید ، می توانید جریان کاری را کمی تغییر دهید و این باعث می شود کارآمد تر شوید. اما هرگز امنیت را برای مدتی از بین نبرید. دور زدن برخی از آیین ها ممکن است مدتی را برای شما به ارمغان آورد ، اما اگر منجر به بروز اشکالات بیشتر و ادغام درگیری ها شود ، می تواند ضد تولید باشد. بعد از چند سال استفاده از Git و GitHub ، شما استاد آن می شوید و می توانید گردش کار خود را ایجاد کنید ، به شرطی که آنچه شما آورده اید ، تیم شما را کارآمدتر کند.

### گردش کار GitHub

اساسی ترین خطایی که هنگام کار با GitHub مرتکب می شوید این است که فقط به عنوان یک سرویس میزبانی کد فکر کنید ، یعنی استفاده از آن فقط برای به اشتراک گذاری کد بین همکاران خود و یا فقط آزاد کردن محصول خود برای کاربران. GitHub چنان ابزاری قدرتمند است که استفاده از آن در تمام توان خود ، یک زیاده عظیم خواهد بود.

از GitHub به عنوان ابزار اصلی مدیریت پروژه خود فکر کنید. هر اقدامی که قصد انجام آن را در پروژه خود دارید باید در GitHub ردیابی شود، بنابراین می توانید به عقب برگردید و تاریخ را درک کنید. شما فقط نمی توانید پیش بروید و برخی تغییرات را ایجاد کنید بدون اینکه به درستی مستند سازی کنید که چرا آن تغییرات را انجام می دهید. در اینجا قوانین طلایی GitHub آورده شده است.

## هر پروژه با یک پروژه شروع می شود

هنگام شروع یک پروژه جدید، باید درست پس از ایجاد مخزن، یک پروژه GitHub ایجاد کنید. شما باید این کار را در اسرع وقت انجام دهید زیرا استفاده از Project Boards بهترین راه برای جلوگیری از تکامل شما است. شما باید حداقل یک هیئت مدیره Kanban داشته باشید تا "انجام" پروژه خود را ردیابی کند. و می توانید از تابلوهای دیگر برای خنثی کردن بازخورد کاربر یا لباس پوشیدن لیستی از ایده های تصادفی خود استفاده کنید. راه اصلی اصلی این است که همیشه آنچه را که در ذهن شما می گذرد را به صورت نوشتاری نگه دارید، زیرا به احتمال زیاد بیشتر آن را فراموش خواهید کرد.

## هر عملی با یک شماره شروع می شود

موضوعات روش خوبی برای یادداشت کردن آنچه باید در پروژه شما انجام شود، است. وقتی متوجه برنامه ای در برنامه خود می شوید، اولین گزینه شما نباید این باشد که بتوانید IDE خود را برطرف کنید بلکه یک شماره را برای پیگیری آن ایجاد کنید. همین موضوع با یک ایده ویژگی پیش می رود، حتی اگر مطمئن نیستید که در آینده روی آن کار خواهید کرد. اگر تصمیم گرفتید که آن را عملی نکنید، یک شماره ایجاد کنید تا هدف خود را مستند کنید و می توانید بعد از بسته شدن آن اقدام کنید.

این آیین دلالت دارد که هر کاری که در Git محلی خود انجام می دهید باید یک مسئله را به عنوان یک هدف حل کند. بنابراین، هنگامی که روی IDE خود کار می کنید، همیشه باید از خود پرسید: "این مسئله چه عاملی را حل می کند؟" اگر جواب "هیچ کس" است، باید مسئله ای برای آن ایجاد کنید، مهم نیست که انجام این کار چقدر کوچک است.

## هیچ push مستقیمی بر روی استاد نیست

این آیین اصلی است که دنبال کردن آن بسیار سخت است اما زندگی را برای همه درگیر در یک پروژه آسانتر می کند. این ایده ساده است: هیچ کس نباید مستقیماً تعهدات خود را به شاخه استاد سوق دهد. تنها راه معرفی تغییرات استاد، ادغام سایر شاخه ها در آن است.

پیامد مستقیم این امر این است که هر تغییری که ایجاد می کنید باید قبل از ادغام در استاد، در شعبه خود باشد. بنابراین، هر ویژگی یا رفع اشکال جدید باید در یک شاخه شروع شود و در صورت آماده شدن در استاد ادغام شود. "آماده" به معنی بررسی و آزمایش صحیح است.

هرگونه ادغام در استاد نیاز به روابط عمومی دارد

از آنجا که ما نمی توانیم مستقیماً به استاد سوق دهیم و تنها انتخاب ادغام شاخه ها در آن است. اما نباید کورکورانه شاخه های شاخه ای را به استاد ادغام کنید. برای پیشنهاد تغییرات باید برای درخواست تغییرات ایجاد کنید. از این طریق ، یکی دیگر از اعضای تیم می تواند تحقیق کند. کد شما برای تأیید صحت بودن یا نبودن کد شما باید به شماره شماره هایی که روابط عمومی را در توضیحات PR برطرف می کنید ، مراجعه کنید تا هنگام پذیرش PR ، شماره ها به طور خودکار بسته شوند.

## برای مستند کردن کد خود از ویکی استفاده کنید

این ممکن است به نظر برسد اما بهترین راه برای ثبت کد شماست. پرونده README برای یک اسناد کامل کد کافی نیست (یا اقتباس شده است) ، بنابراین ویکی لازم است. به نظر می رسد کار بزرگی به نظر برسد ، اما بهترین راه نوشتن اسناد همزمان با کد است. بنابراین ، شما فقط باید هر از گاهی تغییرات کوچکی بنویسید. اگر مدت طولانی برای تصمیم به نوشتن اسناد منتظر بمانید ، دچار ناراحتی می شوید و احتمالاً اطلاعات مهم را فراموش خواهید کرد.

## گردش کار Git

بیایید اکنون در مورد Git صحبت کنیم. در حال حاضر ، شما مطمئناً همه پرکاربردترین ویژگی های Git را می شناسید. اما استفاده از آنها در لحظه مناسب بهترین راه برای جلوگیری از خطا (و درگیری) است.

### همیشه بدانید که کجا هستید

این بسیار اساسی است و بنابراین ، بسیار آسان فراموش می شود. شما همیشه باید بدانید که در کدام شاخه فعالیت می کنید قبل از انجام هرگونه تغییر یا اجرای هر دستور. اگر از IDE مدرن استفاده می کنید ، شاخه فعلی شما باید در پایین صفحه نمایش داده شود. اگر اینطور نیست ، هیچ چیز ضرب و شتم وضعیت git قابل اعتماد قدیمی را ندارد!

### قبل از هر عملی تغییرات از راه دور را بکشید

قبل از ایجاد شاخه ای از آن ، شاخه استاد راه دور را بکشید. این به شما امکان می دهد تا با همکاران خود به روز باشید و از بیشتر درگیری های ادغام خودداری کنید.

### مواظب پیام متعهد خود باشید

لطفاً برای بررسی نحوه نوشتن پیام متعهد خوب ، به فصل 5 مراجعه کنید. این روند را دست کم نگیرید زیرا این ستون اصلی ستون تاریخچه شما خواهد بود. نوشتن پیام متعهد بد ممکن است در ابتدا چند دقیقه شما را نجات دهد ، اما زمان رفع اشکال (آن خواهد آمد ، به من اعتماد کنید) ، ساعتهای بی شماری را در جستجوی متعهدی که معرفی کرده است تلف می کنید.

## تاریخ را بازنویسی نکنید

فقط این یکی از بدترین کارهایی است که می توانید هنگام استفاده از یک تیم Gitwithin انجام دهید. اگر تعهدی را تغییر دهید و مجبور کنید آن را به یک شاخه از راه دور منتقل کنید ، هر کاری که برای آن شاخه انجام می شود با تغییرات شما رونویسی می شود. این بدان معناست که اگر شخص دیگری در آن شاخه کار می کرد ، باید هر کاری را که انجام داده بودند دور بیندازند و شعبه محلی خود را دوباره تنظیم کنند. اگر واقعاً مجبور هستید این کار را انجام دهید ، مطمئن باشید که تنها فردی هستید که در آن شاخه کار می کنید.

## خلاصه

با اینکه فصل کوتاه بود! اما این بهترین راه برای داشتن یک پروژه موفق است. نکته اصلی که باید به خاطر داشته باشید این است که GitHub بسیار بیشتر از یک سرویس میزبانی کد است. شما باید از آن برای ردیابی درست تکامل پروژه و ردیابی ایده های شما یا مشتریان خود استفاده کنید. با دنبال کردن این گردش کار ، خود را برای موفقیت تعیین می کنید زیرا از بیشتر مشکلات مربوط به Git و GitHub جلوگیری خواهید کرد.

شما در حال حاضر تمام ابزارهای موفقیت در Git و GitHub را دارید! همه اینها به تخیل و شجاعت شما بستگی دارد. از این ابزارها به درستی استفاده کنید و پروژه خود را به بهترین مسیرها هدایت کنید. موفق باشید!