

# فصل دوم

## **Version Control with Git**

**کنترل نسخه با Git**

## سیستم های کنترل نسخه

این اولین پرش ما به سیستم های کنترل نسخه (VCS) است. در پایان این فصل ، شما باید در مورد Version Control ، Git و تاریخچه آن بدانید. هدف اصلی این است که بدانیم کنترل نسخه در چه شرایطی مورد نیاز است و چرا Git انتخابی ایمن است.

### کنترل نسخه چیست؟

همانطور که از نام آن پیداست ، Version Control مربوط به مدیریت چندین نسخه از یک پروژه است. برای مدیریت یک نسخه ، هر تغییر (علاوه بر این ، ویرایش یا حذف) در پرونده های یک پروژه باید ردیابی شود. نسخه کنترل هر تغییر ایجاد شده در یک پرونده (یا گروهی از پرونده ها) را ضبط می کند و راهی برای خنثی سازی یا برگشت هر تغییر را ارائه می دهد.

برای یک کنترل نسخه مؤثر ، شما باید از ابزاری به نام سیستم های کنترل نسخه استفاده کنید. آنها به شما کمک می کنند تا بین تغییرات تغییر مکان دهید و به سرعت به شما اجازه می دهد تا وقتی چیزی درست نیست به نسخه قبلی برگردید.

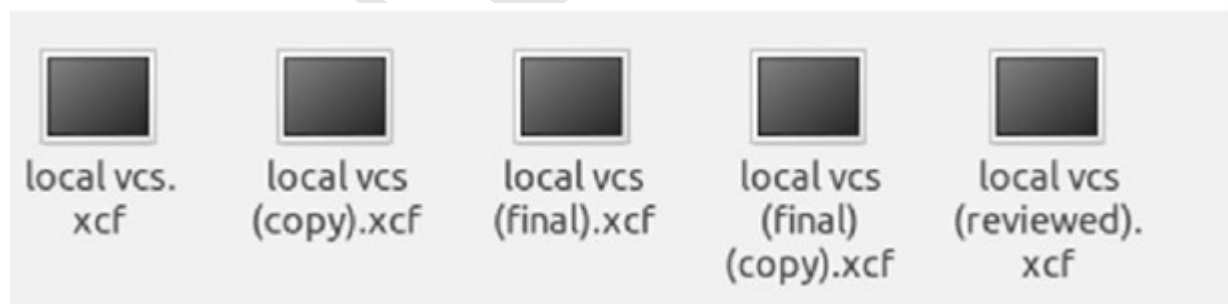
هنگامی که بیش از یک نفر در یک پروژه مشارکت می کند ، ردیابی تغییرات به کابوس تبدیل می شود و احتمال بازنویسی شخص دیگری را تغییر می دهد. با کنترل نسخه ، چندین نفر می توانند روی نسخه خود از پروژه (به نام شاخه ها) کار کنند و تنها وقتی که آنها (یا اعضای تیم دیگر) از کار راضی باشند ، آن تغییرات را در پروژه اصلی ادغام کنند.

چرا به یکی احتیاج داری؟

آیا تاکنون روی یک پروژه متنی یا کدی کار کرده اید که از شما می خواهد تغییرات خاصی که در هر پرونده ایجاد شده است را بخاطر بسپارید؟ اگر بله ، چگونه توانستید هر نسخه را کنترل و کنترل کنید؟

شاید سعی کرده باشید پرونده هایی را با پسوندهایی مانند بررسی ، "ثابت" یا "نهایی" کپی و تغییر نام دهید؟ شکل زیر نوع

VersionControl را نشان می دهد.

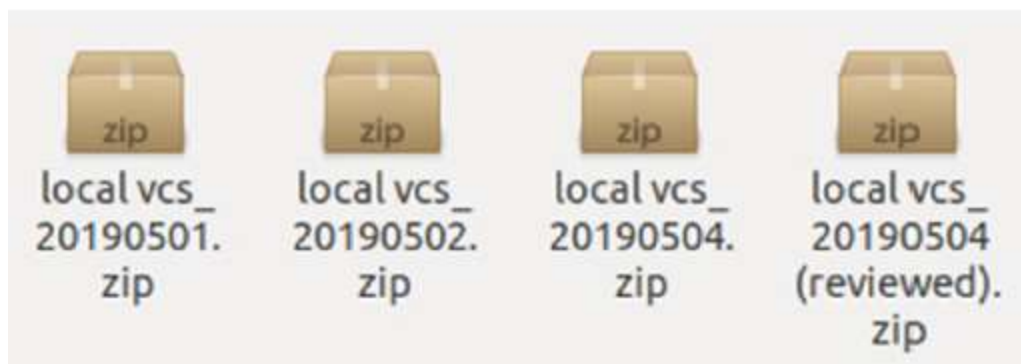


*Gimp files with suffixes like "final," "final (copy)," and "reviewed"*

شکل نشان می دهد که بسیاری از مردم برای مقابله با تغییرات پرونده انجام می دهند.

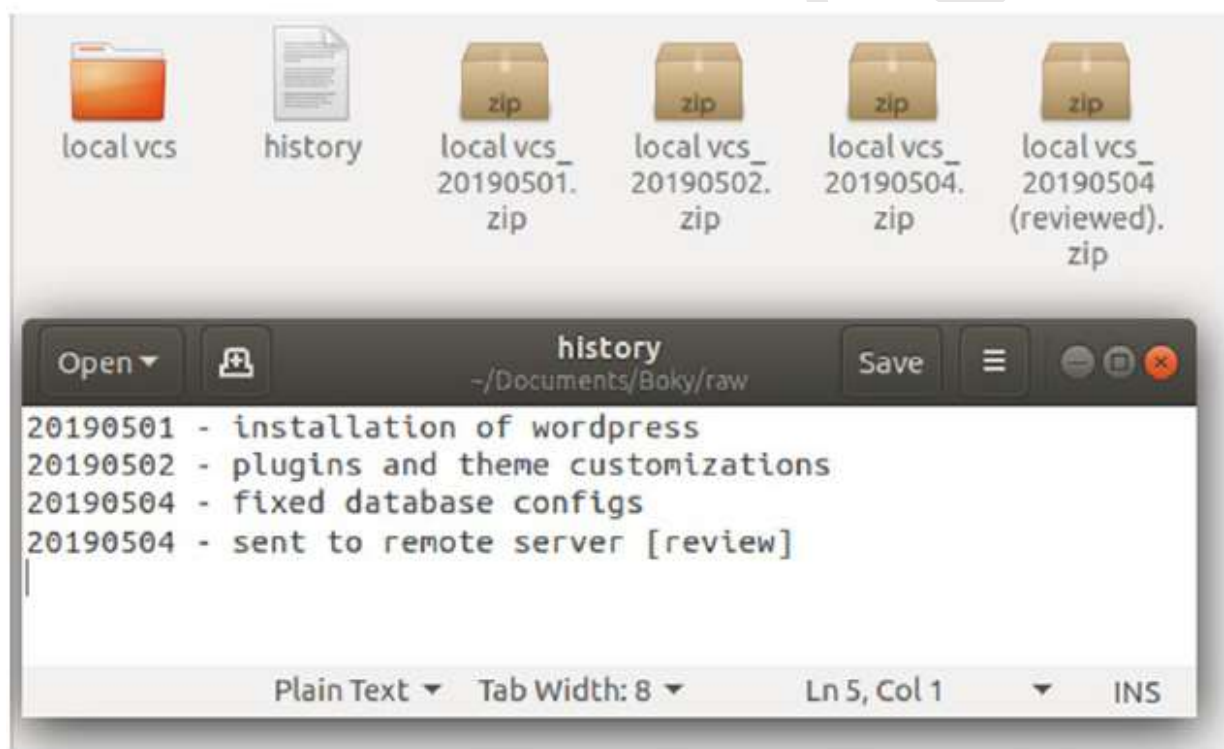
همانطور که مشاهده می کنید ، این پتانسیل را دارد که خیلی سریع دست از روی آن برود ، خیلی راحت فراموش می شود که کدام پرونده است و چه چیزی بین آنها تغییر کرده است.

برای ردیابی نسخه ها ، یک ایده این است که پرونده ها را فشرده کرده و نشان های زمانی را به نام ها اضافه کنید تا نسخه ها بر اساس تاریخ ایجاد تنظیم شوند. شکل زیر این نوع از ردیابی نسخه را نشان می دهد.



*Compressed version files sorted by dates*

به نظر می رسد راه حل نشان داده شده در شکل بالا تا زمانی که متوجه نشوید که حتی اگر نسخه ها ردیابی شوند ، نمی توانید بدانید محتویات و توضیحات هر نسخه چیست. برای برطرف کردن این وضعیت ، برخی از توسعه دهندگان از راهکاری مانند آنچه در شکل زیر نشان داده شده استفاده می کنند ، یعنی قرار دادن خلاصه تغییر هر نسخه در یک پرونده جداگانه.



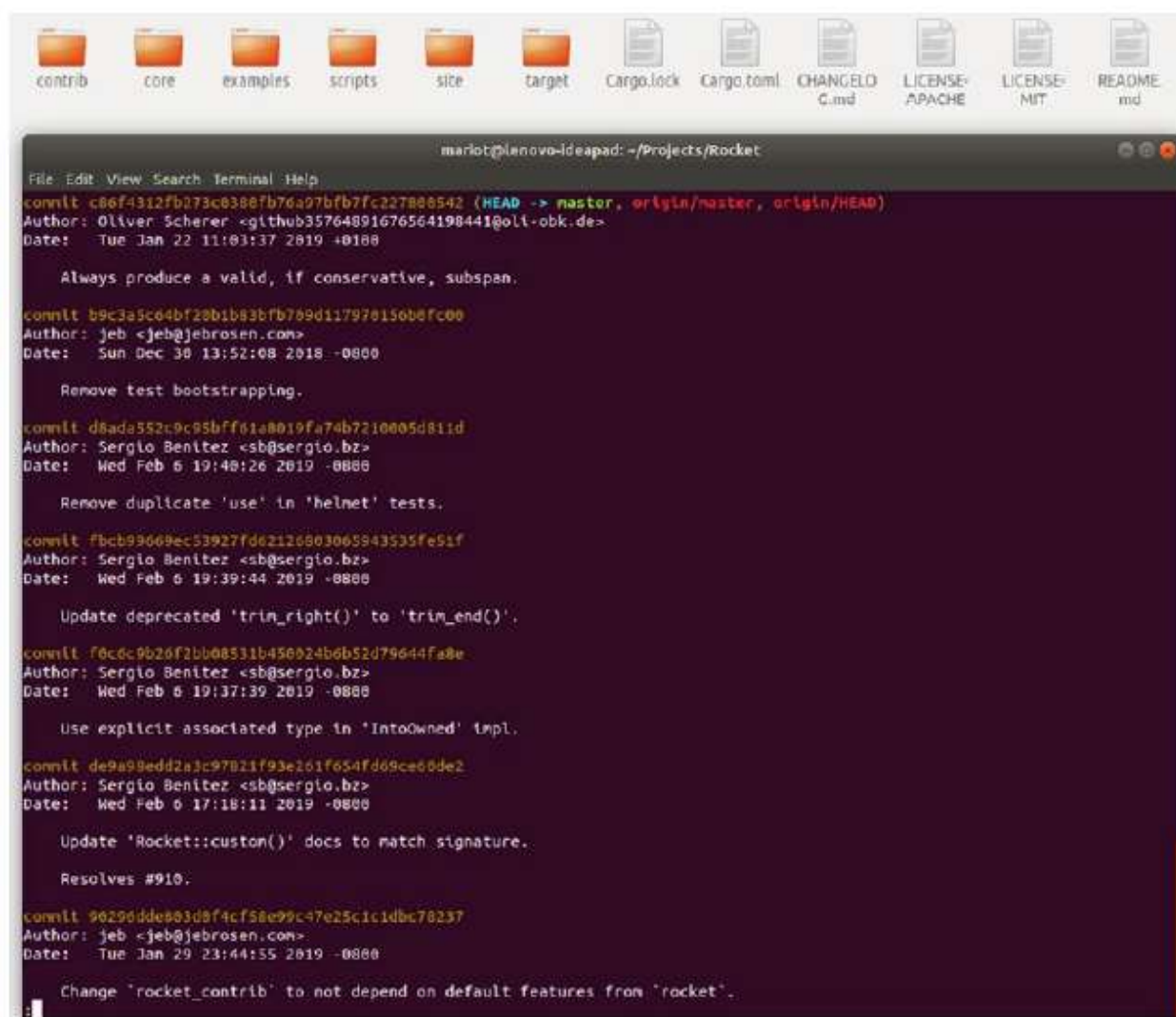
یک پرونده جداگانه که در آن هر نسخه ردیابی می شود

همانطور که در شکل بالا نشان داده شده است ، یک پرونده جداگانه با پوشه پروژه همراه با توضیحات مختصر در مورد تغییر ایجاد شده همچنین به بسیاری از پرونده های فشرده شده که شامل نسخه های قبلی پروژه هستند توجه کنید. باید این کار را بکند ، درست است؟ نه کاملاً ، شما هنوز هم راهی برای مقایسه هر نسخه و تغییر هر پرونده نیاز دارید. هیچ راهی برای انجام این کار در آن سیستم وجود ندارد. شما فقط باید هر کاری که کردید را به خاطر بسپارید. و اگر پروژه بزرگ شود ، پوشه با هر نسخه بزرگتر می شود.

وقتی برنامه نویس یا نویسنده دیگری به تیم شما بپیوندد چه اتفاقی می افتد؟ آیا پرونده ها یا نسخه هایی را که ویرایش کرده اید به یکدیگر ارسال می کنید؟ یا روی همان پوشه راه دور کار کنید؟ در مورد آخر ، چگونه می دانید چه کسی روی کدام پرونده کار می کند و چه چیزی تغییر کرده است؟

و آخر اینکه ، آیا هرگز احساس نیاز به خنثی کردن تغییری را که سالها قبل ایجاد کرده اید بدون اینکه همه چیز را در روند انجام دهید ، خنثی کرده اید؟ ctrl-Z نامحدود و قدرتمند؟ همه این مشکلات با استفاده از یک سیستم کنترل نسخه یا VCS حل می شوند. VCStracks هر تغییری که در هر پرونده از پروژه خود ایجاد کرده اید و روشی ساده برای مقایسه و برگرداندن آن تغییرات ارائه می دهد. هر نسخه از این پروژه همچنین با شرح تغییرات ایجاد شده به همراه لیستی از پرونده های جدید یا ویرایش شده همراه است. هنگامی که تعداد بیشتری از افراد به این پروژه می پیوندند ، یک VCS می تواند دقیقاً نشان دهد چه کسی یک فایل خاص را در یک زمان خاص ویرایش کرده است. همه اینها باعث می شود شما برای پروژه خود زمان ارزشمندی کسب کنید زیرا می توانید به جای گذراندن زمان برای ردیابی هر تغییری ، روی نوشتن تمرکز کنید. شکل زیر یک پروژه نسخه سازی شده توسط Git را نشان می دهد.

همانطور که در شکل زیر نشان داده شده است ، یک پروژه نسخه شده تمام راه حل هایی را که در این فصل میبینیم ترکیب می کند. توضیحات تغییر ، کار تیمی و تاریخ ویرایش وجود دارد.



```
marlot@lenovo-ideapad: ~/Projects/Rocket
File Edit View Search Terminal Help
commit c86f4312fb273c0388fb76a97bfb7fc227888542 (HEAD -> master, origin/master, origin/HEAD)
Author: Oliver Scherer <github35764891676564198441@oli-obk.de>
Date: Tue Jan 22 11:03:37 2019 +0100

    Always produce a valid, if conservative, subspan.

commit b9c3a5c64bf20b1b83bfb769d117970150b0fc00
Author: jeb <jeb@jebrosen.com>
Date: Sun Dec 30 13:52:08 2018 -0800

    Remove test bootstrapping.

commit d8ada552c9c85bfff61a8019fa74b7210005d811d
Author: Sergio Benitez <sb@sergio.bz>
Date: Wed Feb 6 19:40:26 2019 -0800

    Remove duplicate 'use' in 'helnet' tests.

commit f6cb99669ec53927fde2126803065943535fe51f
Author: Sergio Benitez <sb@sergio.bz>
Date: Wed Feb 6 19:39:44 2019 -0800

    Update deprecated 'trin_right()' to 'trim_end()'.

commit f6c6c9b26f2bb08511b450b24b0b52d79644fa8e
Author: Sergio Benitez <sb@sergio.bz>
Date: Wed Feb 6 19:37:39 2019 -0800

    Use explicit associated type in 'IntoOwned' impl.

commit de9a98edd2a3c97021f93e201f654fd09ce60de2
Author: Sergio Benitez <sb@sergio.bz>
Date: Wed Feb 6 17:18:11 2019 -0800

    Update 'Rocket::custom()' docs to match signature.

    Resolves #910.

commit 90296dde03d8f4cf58e99c47e25c1c1db70237
Author: jeb <jeb@jebrosen.com>
Date: Tue Jan 29 23:44:55 2019 -0800

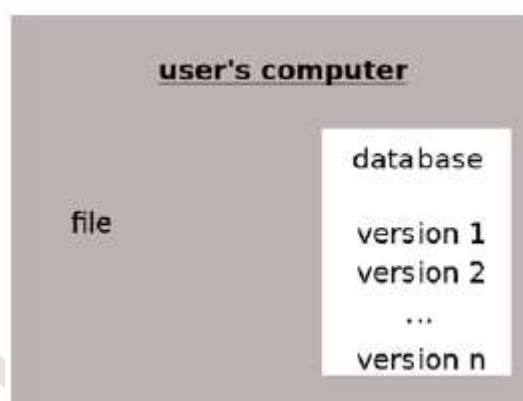
    Change 'rocket_contrib' to not depend on default features from 'rocket'.
```

## گزینه ها چیست؟

در بسیاری از سیستم های کنترل نسخه های مختلفی وجود دارد که هرکدام مزایا و کاستی های خاص خود را دارند. VCS می تواند محلی ، متمرکز یا توزیع شود.

## سیستم های کنترل نسخه محلی

این اولین VCS است که برای مدیریت کد منبع ایجاد شده است. آنها با ردیابی تغییرات ایجاد شده در پرونده ها در یک پایگاه داده واحد که در محلی ذخیره شده است ، کار کردند. این بدان معنی است که همه تغییرات در یک کامپیوتر واحد نگهداری می شدند و اگر مشکلی وجود داشت ، تمام کار از بین می رفت. این همچنین بدان معنی است که کار با یک تیم از این مسئله خارج نبود. یکی از محبوب ترین VCS های محلی Source CodeControl System یا SCCS بود که یک منبع آزاد اما بسته بود. توسعه یافته توسط AT&T ، در دهه 1970 تا زمانی که سیستم کنترل تجدید نظر یا RCS منتشر شد ، به طور مستمر مورد استفاده قرار گرفت. RCS از SCCS محبوب تر شد زیرا این برنامه از Open Source ، cross-platform و بسیار مؤثرتر برخوردار بود. RCS که در سال 1982 منتشر شد ، هم اکنون توسط پروژه گنو نگهداری می شود. یکی از اشکالات این دو محلی محلی این بود که آنها فقط در یک زمان روی یک پرونده کار می کردند. هیچ راهی برای پیگیری کل پروژه با آنها وجود نداشت. برای کمک به شما در تجسم نحوه عملکرد ، شکل زیر یک VCS محلی ساده را نشان می دهد.

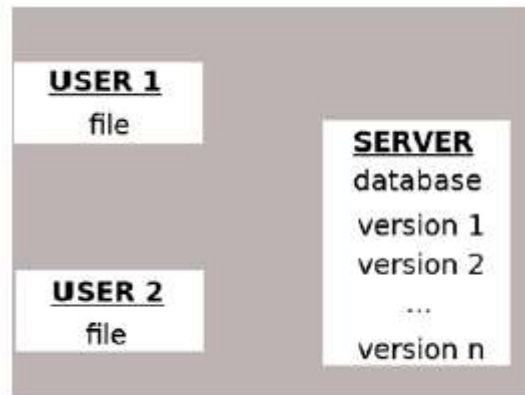


همانطور که در شکل بالا مشاهده می کنید ، همه چیز در رایانه کاربر است و فقط یک پرونده ردیابی می شود. نسخه سازی در پایگاه داده ای که توسط VCS محلی مدیریت می شود ، ذخیره می شود.

## سیستم های کنترل نسخه متمرکز

VCS متمرکز (CVCS) با ذخیره کردن تاریخ تغییر در یک سرور واحد که مشتری ها (نویسندگان) می توانند به آن متصل شوند ، کار می کند. این روش راهی برای همکاری با یک تیم و همچنین راهی برای نظارت بر سرعت کلی یک پروژه ارائه می دهد. آنها هنوز هم محبوب هستند زیرا این مفهوم بسیار ساده است و تنظیم آن بسیار آسان است. مشکل اصلی این بود که ، مانند VCS محلی ، یک خطای سرور می تواند تمام کار خود را برای تیم هزینه کند. از آنجا که پروژه اصلی در یک سرور از راه دور ذخیره شده است ، اتصال به شبکه نیز لازم است.

در شکل زیر نحوه عملکرد آن را مشاهده می کنید.



شکل بالا نشان می دهد که یک VCS متمرکز به طور مشابه با VC های محلی کار می کند ، اما پایگاه داده در یک سرور از راه دور ذخیره می شود. مشکل اصلی تیمی که از VCS متمرکز استفاده می کند اینست که به محض استفاده یک پرونده توسط شخصی ، آن پرونده قفل می شود و اعضای دیگر تیم نمی توانند روی آن کار کنند. بنابراین ، آنها مجبور بودند فقط برای تغییر یک پرونده واحد ، بین خودشان هماهنگی کنند. این تاخیرهای زیادی در توسعه ایجاد می کند و به طور کلی منبع ناامیدی زیادی برای همکاران است. و هرچه اعضای تیم بیشتر باشند ، مشکلات بیشتری بوجود می آیند.

در تلاش برای مقابله با مشکلات VCS محلی ، Concurrent Version System یا CVS توسعه داده شد. این منبع باز بود و می توانست به جای یک فایل واحد ، چندین فایل را ردیابی کند. بسیاری از کاربران همچنین می توانند همزمان با همان پرونده کار کنند ، از این رو "همزمان" در نام هستند. تمام تاریخچه در یک مخزن از راه دور ذخیره می شد و کاربران با چک کردن سرور ، تغییرات را مرتب نگه می داشتند ، یعنی کپی کردن محتویات پایگاه داده از راه دور در رایانه های محلی آنها.

Apache Subversion یا SVN در سال 2000 توسعه داده شد و می تواند همه چیزهایی باشد که CVS می تواند با یک پاداش در اختیار داشته باشد: این می تواند پرونده های غیر متنی را ردیابی کند. یکی از مهمترین مزیت های SVN این بود که به جای ردیابی گروهی از پرونده ها مانند VCS قبلی ، کل پروژه را ردیابی می کند.

بنابراین ، در اصل ردیابی فهرست به جای فایل ها است. این بدان معنی است که تغییر نام ، اضافه کردن و حذف نیز ردیابی می شوند. این باعث شده تا SVN همراه با منبع باز ، یک VCS بسیار محبوب؛ و هنوز هم امروزه به طور مستمر مورد استفاده قرار می گیرد.

## سیستم های کنترل نسخه توزیع شده

توزیع VCS تقریباً مشابه VCS متمرکز کار می کند اما با یک تفاوت بزرگ:

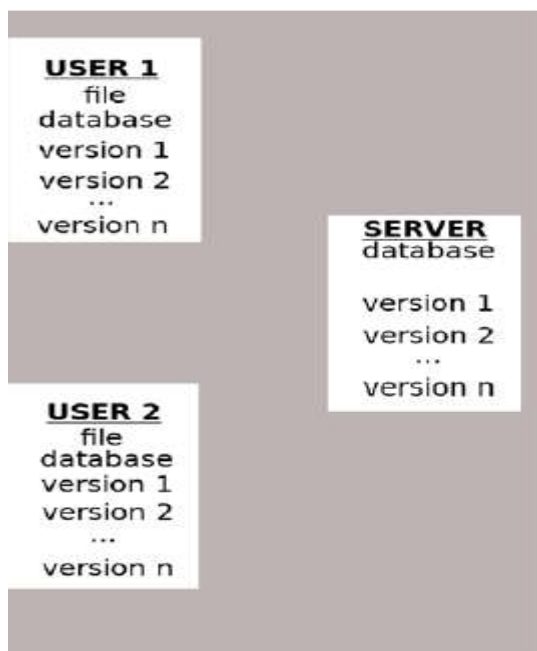
سرور اصلی وجود ندارد که تمام تاریخ را در اختیار داشته باشد. هر مشتری به جای چک کردن یک سرور واحد ، یک نسخه از مخزن (همراه با سابقه تغییر) دارد.

این امر احتمال از دست دادن همه چیز را بسیار کاهش می دهد زیرا هر مشتری دارای کلونی از پروژه است. با استفاده از یک VCS توزیع شده ، مفهوم داشتن "سرور اصلی" تار می شود زیرا هر مشتری در اصل تمام قدرت را در مخزن خود دارد. این امر مفهوم "چنگال" را در جامعه منبع آزاد تشویق می کند. Forking عمل کلون کردن یک مخزن برای ایجاد تغییرات شخصی شما و برداشت متفاوت در پروژه است. فایده اصلی چنگال این است که در صورت مناسب بودن می توانید تغییراتی را از مخازن دیگر بکشید (و دیگران می توانند همین کار را با تغییرات شما انجام دهند).

یک سیستم کنترل نسخه توزیع شده معمولاً سریعتر از سایر انواع VCS است زیرا نیازی به دسترسی به شبکه به یک سرور راه دور ندارد. تقریباً همه چیز به صورت محلی انجام می شود. همچنین تفاوت کمی در نحوه عملکرد آن وجود دارد: به جای

ردیابی تغییرات بین نسخه ها ، تمام تغییرات را به عنوان "تکه" ردیابی می کند. این بدان معنی است که آن تکه ها می توانند آزادانه بین مخازن رد و بدل شوند ، بنابراین هیچ مخزن اصلی برای نگهداری وجود ندارد.

شکل نشان می دهد که چگونه یک VCS توزیع شده کار می کند....



BitKeeper SCM یک VCS اختصاصی بود که در سال 2000 منتشر شد ، مانند SCCS در دهه 1970 ، منبع بسته بود. این یک "نسخه جامعه" رایگان بود که فاقد بسیاری از ویژگی های بزرگ BitKeeper SCM بود ، اما از آنجا که این اولین VCS توزیع شده بود ، حتی در جامعه منبع باز هم بسیار محبوب بود. این محبوبیت BitKeeper نقش بزرگی در ایجاد Git بازی می کند. اکنون پس از انتشار کد منبع خود تحت مجوز Apache در سال 2016 ، این نرم افزار منبع باز است. شما می توانید پروژه فعلی BitKeeper را در [www.bitkeeper.org](http://www.bitkeeper.org) پیدا کنید. پیشرفت کند شده است ، اما هنوز هم جامعه ای در آن نقش دارد.

Git چیست؟

به یاد داشته باشید نسخه کنترل توزیع شده اختصاصی SystemBitKeeper SCM از آخرین بخش؟ خوب ، توسعه دهندگان هسته لینوکس از آن برای پیشرفت خود استفاده کردند. تصمیم به استفاده از آن به طرز وحشیانه یک اقدام بد تلقی می شد و باعث نارضایتی بسیاری از افراد می شد.

از آنجا که این منبع بسته بود ، توسعه دهندگان سیستم کنترل نسخه مورد علاقه خود را از دست دادند. جامعه (به رهبری لینوس توروالدز) مجبور شد VCS دیگری پیدا کند و از آنجا که گزینه دیگری در دسترس نبود ، آنها تصمیم گرفتند که خودشان را ایجاد کنند. بنابراین ، گیت متولد شد. از آنجا که گیت برای جایگزینی BitKeeper SCM ساخته شد ، به طور کلی با چند ترفند مشابه کار کرد. مانند BitKeeper SCM ، Git یک سیستم کنترل نسخه توزیع شده است ، اما سریعتر است و با پروژه های بزرگ بهتر کار می کند. جامعه Git بسیار فعال است و بسیاری از مشارکت کنندگان در توسعه آن نقش دارند. می توانید اطلاعات بیشتری در مورد Git در <https://git-scm.com> پیدا کنید. ویژگی های Git و نحوه عملکرد آن در ادامه در این بخش توضیح داده شده است.



## Git چه کاری می تواند انجام دهد؟

به یاد داشته باشید تمام آن مشکلاتی که ما در ابتدای این فصل سعی کردیم آنها را حل کنیم؟ خوب ، Gitcan همه آنها را حل می کند. این حتی می تواند مشکلاتی را که شما نمی دانسته اید حل کنید! اول ، با ردیابی تغییرات عالی عمل می کند. تو می توانی

- بین نسخه ها به عقب و جلو بروید
  - تفاوت های بین این نسخه ها را مرور کنید
  - تاریخچه تغییر پرونده را بررسی کنید
  - برای ارجاع سریع نسخه خاصی را برچسب بزنید
  - Git همچنین ابزاری عالی برای کار تیمی است. تو می توانی
  - "متغیرها" را بین مخازن مبادله کنید
  - تغییرات ایجاد شده توسط دیگران را مرور کنید
- یکی از ویژگی های اصلی Git سیستم Branching آن است. شعبه یک کپی از پروژه است که می توانید بدون مخدوش کردن با مخزن روی آن کار کنید. این مفهوم مدتی است که وجود دارد ، اما با استفاده از Git بسیار سریعتر و کارآمدتر است. شعبه همچنین همراه با ادغام است که عمل کپی کردن تغییرات انجام شده در یک شعبه است
- برای بازگشت به منبع به طور کلی ، شما یک شاخه ایجاد می کنید تا ویژگی جدیدی را ایجاد کرده و آزمایش کنید و هنگامی که از کار راضی هستید ، آن شاخه را با هم ادغام کنید.
- یک مفهوم ساده نیز وجود دارد که ممکن است از آن استفاده کنید: Stashing. Stashing عملی است که با خیال راحت از بین بردن ویرایش های فعلی خود به گونه ای که محیطی تمیز دارید تا روی چیزهایی کاملاً متفاوت کار کنید. ممکن است بخواهید از بازی کردن در هنگام بازی کردن استفاده کنید یا یک ویژگی را آزمایش می کنید اما باید در اولویت کار روی یک ویژگی جدید باشید. بنابراین ، تغییرات خود را دور می کنید و شروع به نوشتن آن ویژگی می کنید. پس از اتمام کار ، می توانید تغییرات خود را پس بگیرید و آنها را در محیط کار فعلی خود اعمال کنید. به عنوان یک اشتها آور کوچک ، در اینجا برخی از دستورات Git که در این کتاب یاد خواهید گرفت آورده شده است:



```
$ git init      # Initialize a new git database
$ git clone     # Copy an existing database
$ git status    # Check the status of the local project
$ git diff      # Review the changes done to the project
$ git add       # Tell Git to track a changed file
$ git commit    # Save the current state of the project to database
$ git push      # Copy the local database to a remote server
$ git pull      # Copy a remote database to a local machine
$ git log       # Check the history of the project
$ git branch    # List, create or delete branches
$ git merge     # Merge the history of two branches together
$ git stash     # Keep the current changes stashed away to be used later
```

همانطور که مشاهده می کنید ، دستورات کاملاً توصیفی هستند. از دانستن همه آنها به صورت توصیف ، نترسین. وقتی یادگیری را به درستی شروع کنیم ، آنها را یکی یکی حفظ خواهید کرد. و شما همچنین از آنها همیشه استفاده نخواهید کرد ، بیشتر از آن برای افزودن و تعهد استفاده خواهید کرد. شما در مورد هر دستور یاد خواهید گرفت ، اما ما بر روی دستوراتی که احتمالاً در یک محیط حرفه ای از آنها استفاده خواهید کرد ، تمرکز خواهیم کرد. اما قبل از آن ، اجازه دهید کار داخلی Git را ببینیم.

## Git چگونه کار می کند؟

برخلاف بسیاری از سیستم های کنترل نسخه ، Git با Snapshots تفاوت هایی ندارد. این بدان معنی است که تفاوت بین دو نسخه از یک پرونده را ردیابی نمی کند بلکه از وضعیت فعلی پروژه عکس می گیرد.

به همین دلیل Git در مقایسه با سایر VCS های توزیع شده بسیار سریع است. همچنین به همین دلیل جابجایی بین نسخه ها و شاخه ها بسیار سریع و آسان است.

به یاد داشته باشید که چگونه یک سیستم کنترل نسخه متمرکز کار می کند؟

خوب ، گیت کاملاً برعکس. برای انجام کار نیازی به برقراری ارتباط با سرور مرکزی ندارید. از آنجا که Git یک VCS توزیع شده است ، هر کاربر دارای مخزن کاملاً مناسب با تاریخچه و تغییرات خاص خود است. بنابراین ، همه چیز بجز اشتراک گذاری تکه ها یا تغییر شکلهای بصورت محلی انجام می شود. همانطور که قبلاً گفته شد ، به سرور مرکزی نیازی نیست. اما بسیاری از توسعه دهندگان از یکی به عنوان یک کنوانسیون استفاده می کنند زیرا کار با آن راحت تر است. صحبت از اشتراک گذاری بچ ، چگونه Git می داند که تغییرات کدام یک هستند؟ وقتی گیت عکس فوری می گیرد ، یک چک را بر روی آن انجام می دهد. بنابراین ، می داند که کدام پرونده ها با مقایسه چک ها تغییر یافته اند. به همین دلیل Git می تواند تغییرات بین پرونده ها و دایرکتوری ها را به راحتی ردیابی کند ، و همچنین هر گونه فساد فایل را بررسی می کند.

ویژگی اصلی گیت سیستم "Three States" آن است. "States" ها فهرست کار ، منطقه مرحله بندی و فهرست گیت هستند:

- فهرست کار فقط عکس فعلی است که شما روی آن کار می کنید.

- منطقه مرحله بندی جایی است که پرونده های تغییر یافته در نسخه فعلی آنها ، آماده برای ذخیره در دیتابیس مشخص شده اند.

- فهرست git پایگاه داده ای است که تاریخ در آن ذخیره می شود.

بنابراین ، اساساً Git به شرح زیر کار می کند: شما پرونده ها را تغییر می دهید ، هر پرونده ای را که می خواهید در عکس فوری وارد کنید به قسمت مرحله بندی (git add) اضافه کنید ، سپس عکس فوری را بگیرید و آنها را به بانک اطلاعات اضافه کنید (git commit). برای اصطلاحات ، ما یک پرونده اصلاح شده اضافه شده به قسمت مرحله بندی "s tagged شده" و یک پرونده اضافه شده به پایگاه داده "commit" می نامیم. بنابراین ، یک پرونده از "اصلاح شده" به "مرحله بندی شده" به "commit" می رود.

## گردش کار معمولی Git چیست؟

برای کمک به شما در تجسم آنچه در این بخش در مورد آن صحبت کردیم ، در اینجا کمی نمایشی از یک گردش کار معمولی با استفاده از Git ارائه شده است. نگران نباشید اگر اکنون همه چیز را درک نکردید ، فصل های بعدی شما را تنظیم می کند.

این اولین روز کار شماست. شما وظیفه دارید نام خود را به پرونده توضیحات پروژه موجود اضافه کنید. از آنجا که این روز اول شماست ، یک توسعه دهنده ارشد برای بررسی کد شما در آنجا حضور دارد.

اولین کاری که شما باید انجام دهید این است که کد منبع پروژه را بدست آورید. از سرور خود بخواهید که سرور کجا ذخیره شود. برای این نسخه ی نمایشی ، سرور GitHub است ، به این معنی که پایگاه داده Git در یک سرور از راه دور به میزبانی GitHub ذخیره می شود و می توانید به وسیله URL یا مستقیماً در وب سایت GitHub به آن دسترسی داشته باشید. در اینجا ، ما قصد داریم از دستور clone برای به دست آوردن دیتابیس استفاده کنیم ، اما شما همچنین می توانید این پروژه را از وب سایت GitHub بارگیری کنید. یک فایل فشرده حاوی و پرونده های پروژه با تمام تاریخچه آن را دریافت خواهید کرد. بنابراین ، شما مخزن را کlon می کنید تا با استفاده از دستور "clone" کد منبع را بدست آورید.

```
git clone https://github.com/mariot/thebestwebsite.git
```

سپس Git یک کپی از مخزن را در دایرکتوری فعلی که از آن کار می کنید بارگیری می کند. پس از آن می توانید دایرکتوری جدید را وارد کنید و محتویات آن را مطابق شکل زیر بررسی کنید.

```
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw (master)
$ cd thebestwebsite/

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/thebestwebsite (master)
$ dir
gulpfile.js  LICENSE  nginx  package.json  README.md  src  yarn.lock

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

محتوای مخزن نشان داده شده است

اگر می خواهید تغییرات اخیر در پروژه را بررسی کنید ، می توانید از دستور "log" برای نمایش تاریخ استفاده کنید. شکل زیر نمونه ای از آن را نشان می دهد.

```

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/thebestwe
$ git log
commit 0cc01f912449ed913c9f48673a4b450a66951f31 (HEAD -> mas
Author: Denys Vitali <denys@denv.it>
Date:   Fri Jan 18 17:44:45 2019 +0100

    Add Hugo Theme references

    Reference: https://github.com/hugomodo/hugomodo-best-mot

commit 033eb62a526e4fffd9c73257ab37e76c9d484cd74
Author: Denys Vitali <denys@denv.it>
Date:   Thu Jan 10 10:46:28 2019 +0100

    Fix #31, add inverted-contrast mode

commit 74452d4c8cacb2dcad4431532eb99ccac4b00eac
Merge: 13e4f7e 6c3ba31
Author: Denys Vitali <denys@denv.it>
Date:   Mon Nov 12 10:12:39 2018 +0100

    Merge pull request #30 from numbermaniac/patch-1

    create -> created

commit 6c3ba31b95190fdaecf95b9af2b9d2f5554d7203
Author: numbermaniac <numbermaniac@users.noreply.github.com>
Date:   Sun Nov 11 11:22:45 2018 +1100

    create -> created

```

خوب! اکنون شما باید یک شعبه جدید ایجاد کنید تا در این زمینه کار کنید تا با پروژه دچار مشکل نشوید. با استفاده از دستور "branch" می توانید یک شاخه جدید ایجاد کرده و با دستور "checkout" آن را چک کنید.

```

git branch add-new-dev-name-to-readme
git checkout add-new-dev-name-to-readme

```

اکنون که شاخه جدید ایجاد شده است ، می توانید شروع به تغییر پرونده ها کنید. شما می توانید از هر ویرایشگر مورد نظر استفاده کنید. Git تمام تغییرات را از طریق checksums پیگیری می کند. اکنون که تغییرات لازم را ایجاد کرده اید ، وقت آن رسیده است که آنها را در قسمت صحنه بندی قرار دهید. به عنوان یک یادآوری ، منطقه مرحله بندی جایی است که کدهای اصلاح شده را آماده می کنید تا برای عکس فوری آماده شوند. اگر پرونده "README.MD" را اصلاح کردیم ، می توانیم با استفاده از دستور "add" آن را به قسمت صحنه اضافه کنیم.

```

git add README.md|

```

لازم نیست هر پرونده تغییر یافته را به قسمت صحنه اضافه کنید ، فقط مواردی را که می خواهید در عکس فوری از آنها حساب کنید. اکنون که پرونده آماده شده است ، زمان آن رسیده است که آن را put یا تغییر داده را در بانک اطلاعاتی قرار دهیم. ما این کار را با استفاده از دستور "commit" انجام می دهیم و توضیحی مختصر با آن می دهیم.

```
git commit -m "Add Mariot to the list of developers"
```

و همین تغییری که ایجاد کرده اید اکنون در بانک اطلاعاتی بوده و با خیال راحت ذخیره می شود. اما فقط بر روی کامپیوتر شما! دیگران نمی توانند کار شما را ببینند زیرا شما در مخزن شخصی خود و در شعبه ای دیگر کار کرده اید. برای نشان دادن کار خود به دیگران ، باید تعهدات خود را به سمت سرور راه دور سوق دهید. اما شما باید قبل از فشار دادن ، ابتدا کد را به dev ارشد نشان دهید. اگر آنها با آن خوب نیستند ، می توانید شاخه خود را با تصویر اصلی پروژه (که به آن شاخه اصلی گفته می شود) ادغام کنید. بنابراین ابتدا باید با استفاده از دستور "checkout" به شاخه کارشناسی ارشد بروید.

```
git checkout master
```

شما اکنون در master branch هستید ، جایی که تمام کارهای تیم در آن ذخیره می شود. اما ممکن است زمانی که شما در رفع مشکل خود کار کردید ، این پروژه تغییر کرده است ، به این معنی که ممکن است یک عضو تیم برخی از پرونده ها را تغییر داده باشد. شما باید قبل از انجام تغییرات شخصی خود ، آن تغییرات را بازیابی کنید. این خطر "درگیری" را که می تواند اتفاق بیفتد وقتی دو یا چند همکار پرونده مشابه را تغییر می دهند ، کاهش می دهد.

برای به دست آوردن تغییرات ، شما باید پروژه را از سرور راه دور (also called origin) بکشید.

```
git pull origin master
```

حتی اگر همکار دیگری همان پرونده شما را تغییر داده باشد ، خطر درگیری کم است زیرا شما فقط یک خط را تغییر داده اید. درگیری فقط وقتی ایجاد می شود که همان خط توسط چندین نفر اصلاح شده باشد. اگر شما و همکارانتان قسمت های مختلف پرونده را تغییر داده اید ، همه چیز درست است.

اکنون که با وضعیت فعلی پروژه سازگار بودیم ، وقت آن رسیده که نسخه خود را برای تسلط خود متعهد کنیم. می توانید شاخه خود را با دستور "merge" ادغام کنید.

```
git merge add-new-dev-name-to-readme
```

اکنون که این commit دوباره به master تبدیل شده است ، زمان آن رسیده است که تغییرات را به سمت سرور اصلی فشار دهید. ما این کار را با استفاده از دستور "push" انجام می دهیم.

```
git push
```

شکل زیر دستوراتی را که ما استفاده کرده ایم و نتایج نشان می دهد.



```

Mariot@lenovo-ideapad MINGw64 ~/Documents/Boky/raw/thebestwebsite (add-new-dev-name-to-readme)
$ git commit -m "Add Mariot to the list of developers"
[add-new-dev-name-to-readme 4de128e] Add Mariot to the list of developers
1 file changed, 1 insertion(+), 1 deletion(-)

Mariot@lenovo-ideapad MINGw64 ~/Documents/Boky/raw/thebestwebsite (add-new-dev-name-to-readme)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Mariot@lenovo-ideapad MINGw64 ~/Documents/Boky/raw/thebestwebsite (master)
$ git merge add-new-dev-name-to-readme
Updating 0cc01f9..4de128e
Fast-forward
 README.md | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)

Mariot@lenovo-ideapad MINGw64 ~/Documents/Boky/raw/thebestwebsite (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 316 bytes | 316.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/mariot/thebestwebsite.git
0cc01f9..4de128e master -> master

```

ساده است! و باز هم ، اگر هنوز همه چیز را نمی فهمید ، نگران نباشید. این فقط کمی نمایش نحوه استفاده از Git است. همچنین خیلی واقع بینانه نیست: هیچ مدیری به این استخدام جدید اجازه دسترسی همه جانبه به مخزن اصلی خود را مانند آن نمی دهد.

## خلاصه

این تنها نگاهی دزدکی در گیت بود. این ویژگی های بسیار قدرتمند دیگری دارد که در طول راه یاد خواهید گرفت. اما قبل از هر چیز دیگری ، در اینجا مواردی وجود دارد که باید قبل از حرکت به مرحله بعدی از خود پرسید: "چگونه Git به من در پروژه هایم کمک می کند؟" ، "کدام ویژگی ها مهمترین هستند؟" ، و "آیا Git می تواند جریان کاری من را بهبود بخشد. ؟"

پیشگام اصلی این فصل ، تفاوت بین VCS توزیع شده و متمرکز است. گردش کار تیم هایی که از CVCS استفاده می کنند کمتر ساماندهی شده است و بسیاری از توسعه دهندگان را نیز برآورده نمی کند. بنابراین ، شما باید در مورد VCS توزیع شده بیشتر بیاموزید تا با اوقات خود ادامه دهید

# فصل سوم

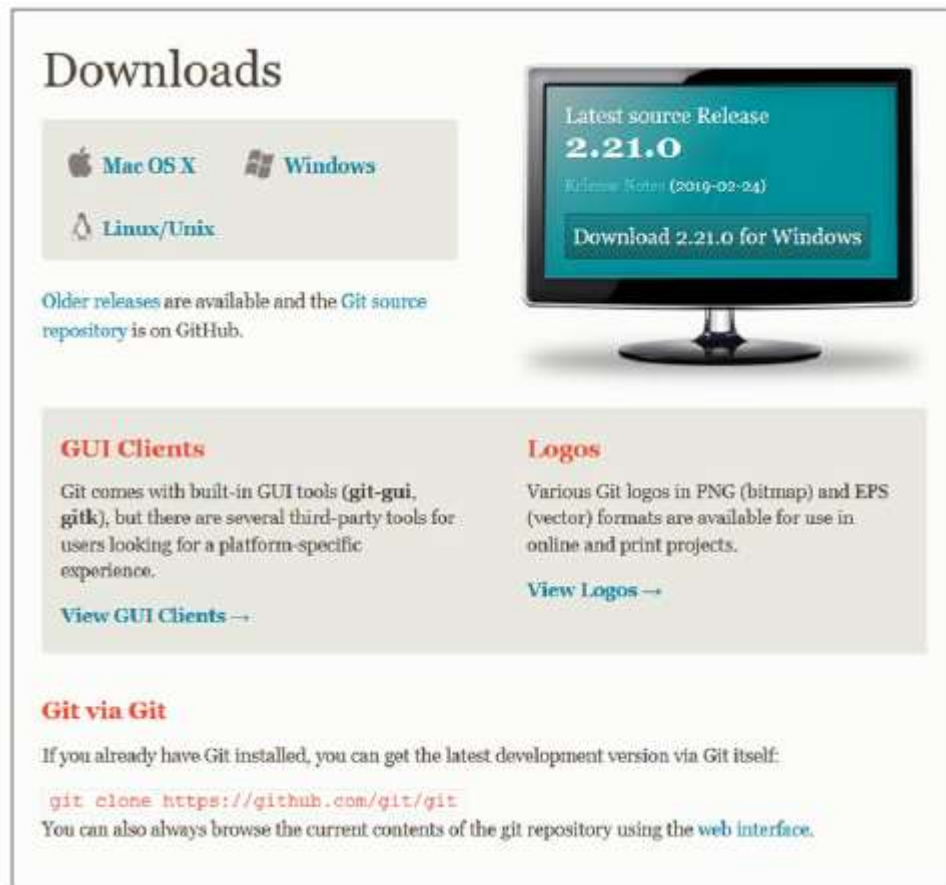
## نصب و راه اندازی

اکنون که شما با نسخه کنترل و Git چگونه کار می کنید ، می خواهیم نحوه نصب و راه اندازی آن را بیاموزیم. این فصل در مقایسه با سایر موارد کوتاه تر است زیرا تنظیم Git بسیار آسان است.

### نصب و راه اندازی

پرونده های لازم برای نصب Git در <https://git-cm.com/downloads> برای کلیه سیستم ها است.

فقط پیوند را دنبال کرده و سیستم عامل خود را انتخاب کنید. همچنین می توانید در شکل زیر مشاهده کنید که مشتری های GUI برای Git نیز در آنجا وجود دارد. قبل از اتمام قسمت سوم این کتاب ، کار گروهی با Git ، به آنجا سرزنشید. شما باید قبل از استفاده از مشتری های GUI خود را با Gitcommands آشنا کنید. در غیر این صورت ، شما وقت زیادی را برای تلاش برای حل مسئله ساده که با دستورات ساده آن ثانیه طول می کشد ، از دست می دهید.



*The download section of git-scm.com as of May 2019*

بعد از اینکه خود را با دستورات Git آشنا کردید ، می توانید یک مشتری GUI را بررسی کرده و خود را ببینید. در قسمت آخر این کتاب یک فصل در مورد مشتریان GUI وجود دارد. اما لطفا قبل از آن زمان هیچ مشتری GUI را انجام ندهید. این زمان یادگیری شما را به شدت طولانی می کند.

## Windows

نصب Git در سیستم های ویندوز بسیار آسان است. پس از باز کردن لینک (<https://gitscm.com/download/win>) ، بارگیری باید به صورت خودکار شروع شود و شما به صفحه تأیید نشان داده شده در شکل زیر خواهید رسید. اگر اینطور نیست ، فقط ساختهای مربوط به ویندوز خود را بارگیری کنید.





### Your download is starting...

You are downloading the latest (2.21.0) 64-bit version of Git for Windows. This is the most recent maintained build. It was released 2 months ago, on 2019-02-26.

If your download hasn't started, [click here to download manually](#).

### Other Git for Windows downloads

Git for Windows Setup

[32-bit Git for Windows Setup.](#)

[64-bit Git for Windows Setup.](#)

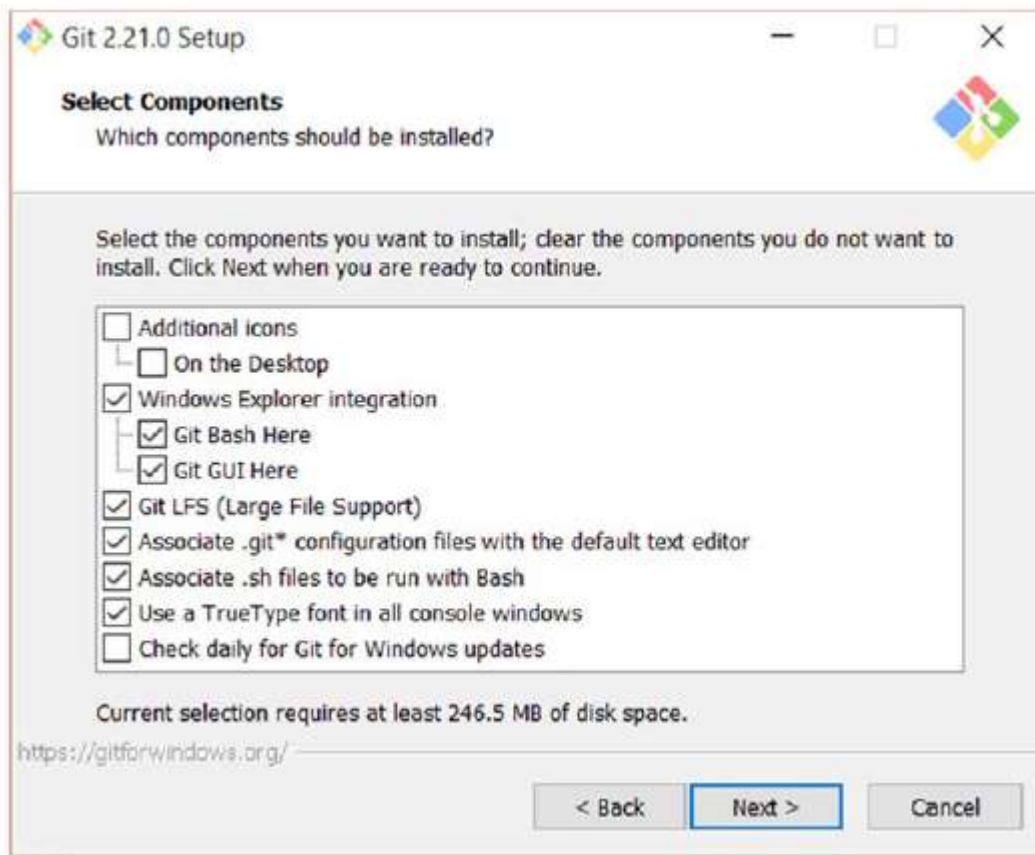
Git for Windows Portable ("thumbdrive edition")

[32-bit Git for Windows Portable.](#)

[64-bit Git for Windows Portable.](#)

The current source code release is version 2.21.0. If you want the newer version, you can build it from [the source code](#).

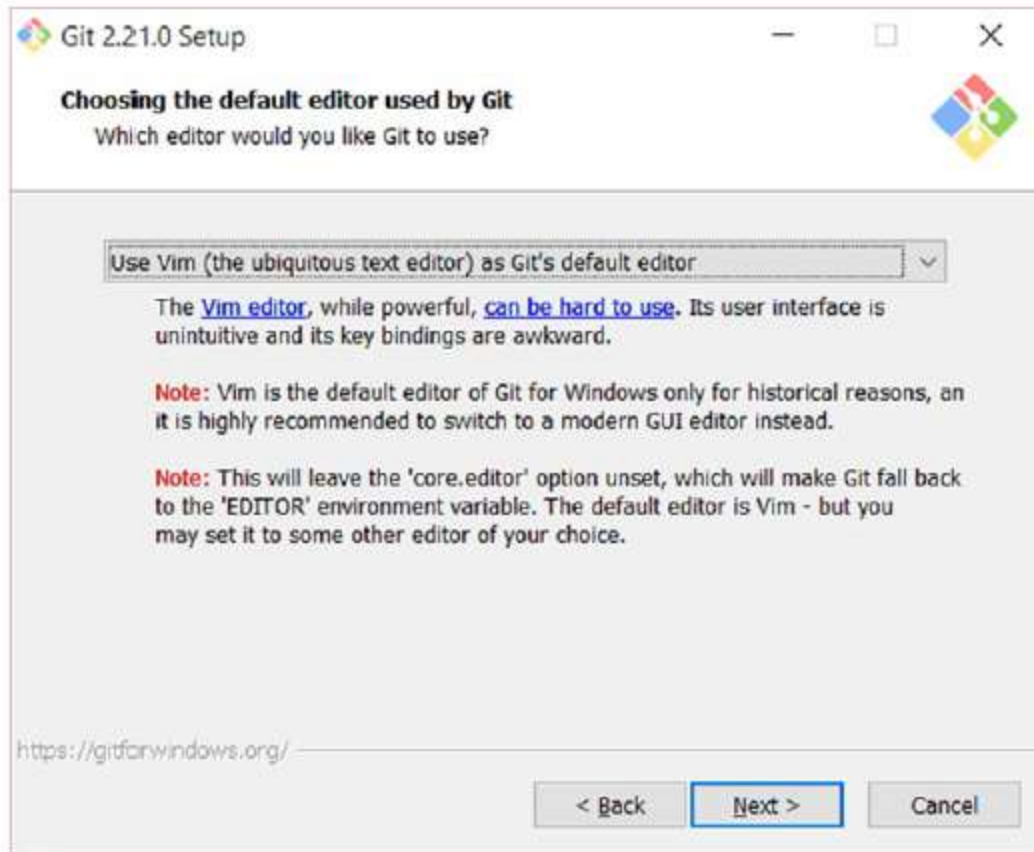
برای شروع نصب فایل exe را اجرا کنید. صفحه اول اعلامیه مجوز است که شرایط و ضوابط را بیان می کند. شما باید آن را تا آخر بخوانید (بله ، درست است). روی بعدی کلیک کنید ، و به صفحه انتخاب کامپوننت شبیه به صفحه نمایش داده شده در شکل زیر خواهید رسید. در اینجا از شما خواسته می شود که کدام مؤلفه ها را نصب کنید. توصیه می کنم گزینه های پیش فرض را روشن کنید.



در شکل بعد مشاهده می کنید که فقط باید اجزای نصب آنها را بررسی کنید.

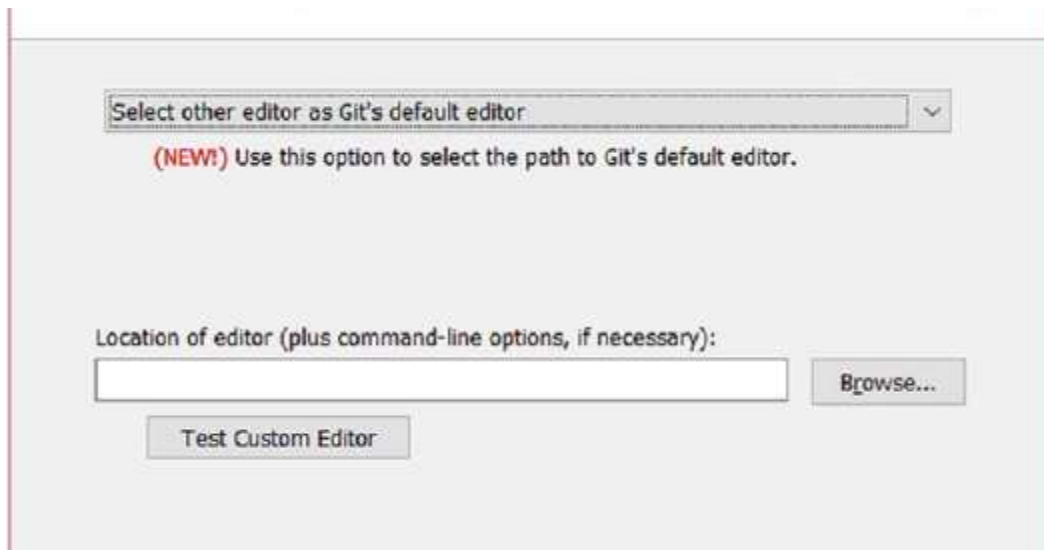
این ایده خوبی است که ادغام ویندوز اکسپلورر را بررسی کنید. به این ترتیب شما فقط باید روی یک پوشه راست کلیک کنید تا گزینه هایی را برای شروع Git در GUI پیش فرض یا خاکستر (پنجره فرمان) در منوی زمینه پیدا کنید. تمام مؤلفه های دیگر کاملاً توضیحی هستند ، بنابراین تصمیم بر عهده شماست.

بعد از انتخاب خود ، روی گزینه بعدی کلیک کنید ، و انتخاب ویرایشگر پیش فرض را مشاهده خواهید کرد که در شکل نشان داده شده است. Git برای تعریف ویرایشگر پیش فرض به شما نیاز دارد زیرا به ویرایشگر نیاز دارید تا توضیحات و نظرات را commit بنویسد.



همانطور که در شکل زیر مشاهده می کنید ، Vim به دلایل تاریخی ویرایشگر پیش فرض برای Git است.

فقط ویرایشگر متن مورد علاقه خود را از لیست کشویی انتخاب کنید. دو نفر اول ، Nano و Vim ، در کنسول یا پنجره فرمان کار می کنند ، بنابراین شما مجبور نیستید برنامه دیگری را باز کنید. در این لیست ، بسیاری از ویرایشگران محبوب مانند ++Notepad ، SublimeText ، Atom و کد ویژوال استودیو (VS) را می توانید پیدا کنید. اگر ویرایشگر شما فهرست نشده است ، می توانید آخرین گزینه را انتخاب کنید ، و یک ورودی جدید ظاهر می شود (در شکل 2-5 نشان داده شده است) بنابراین می توانید پیوندی را به پرونده اجرایی اصلی ویرایشگر ارائه دهید.



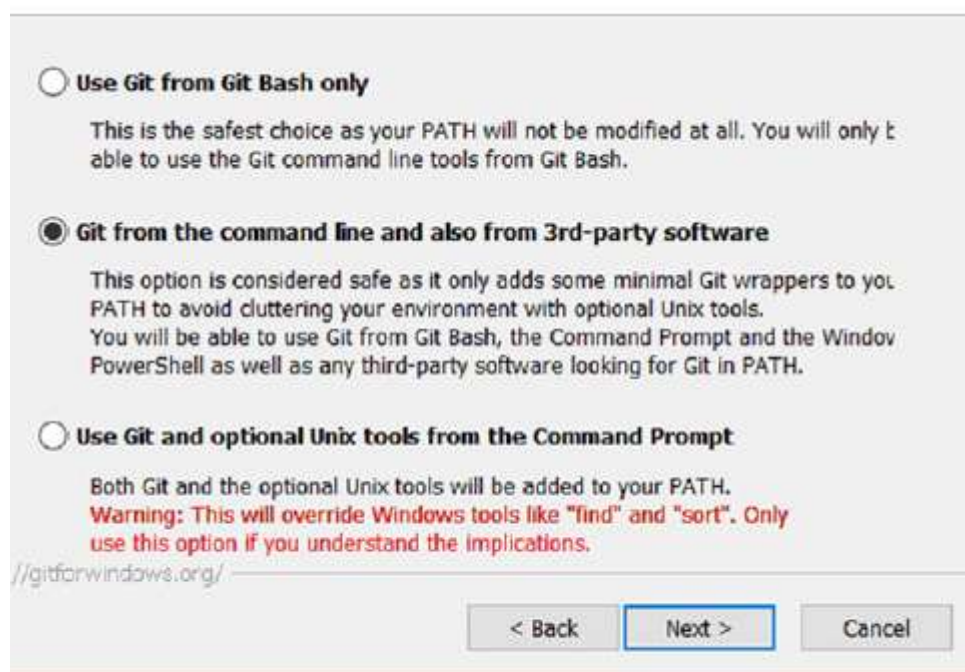
در شکل بالا ، صفحه را می توانید مشاهده کنید که اگر در لیست کشویی ذکر نشده است ، می توانید ویرایشگر سفارشی خود را تنظیم کنید. برای این کتاب تصمیم گرفتم گزینه پیش فرض را رها کنم و از Vim استفاده کنم. اگر تصمیم دارید از هر ویرایشگر دیگری استفاده کنید ، هیچ چیزی در این کتاب تغییر نمی کند. و نگران نباشید ، این انتخاب قطعی نیست ، شما هنوز هم می توانید هر زمان که می خواهید تغییر دهید.

پس از انتخاب ویرایشگر مورد علاقه خود ، می توانید به صفحه بعدی بروید ، یعنی تنظیم محیط PATH ، که در شکل بعد نشان داده شده است. path متغیری است که لیستی از فهرست هایی را که برنامه های اجرایی در ارزش آنها قرار دارند ، در خود جای می دهد. در صورت نیاز به اجرای آن در کنسول ، نیازی به تایپ کامل مسیر اجرایی نیست. فقط باید نام آن را تایپ کنید. به عنوان مثال ، برای راه اندازی Visual Studio Code از کنسول ، باید

C:\Program Files (x86)\Microsoft VS Code\bin

C:\Program Files (x86)\Microsoft VS Code\bin\code.

را تایپ کنید. فقط باید آن را راه اندازی کنم.



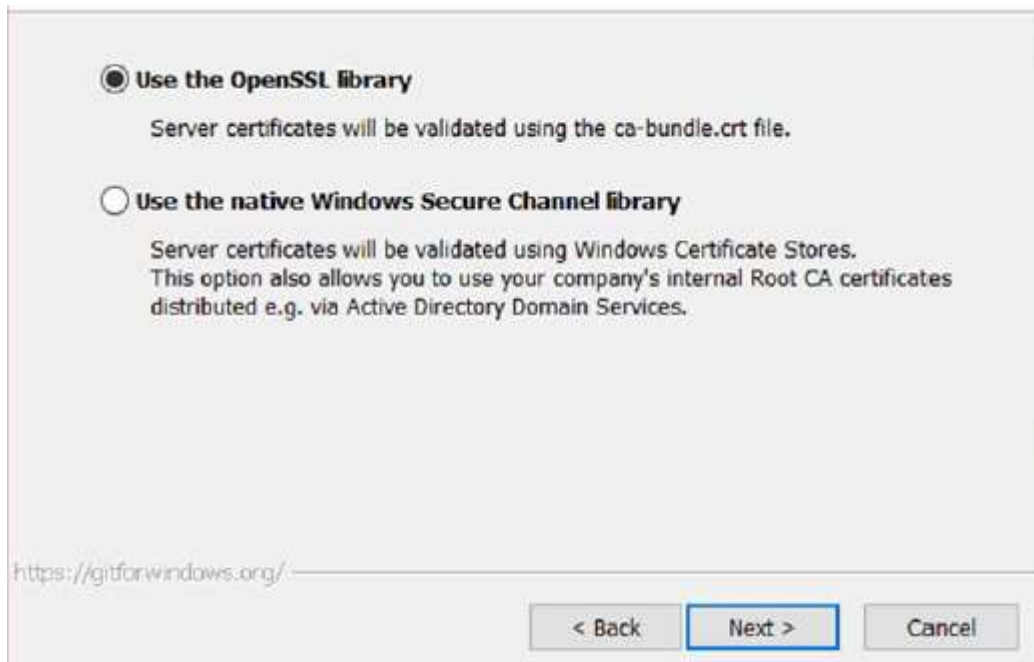
### Choosing to add Git to PATH or not

در صورت تمایل همین کار می تواند در مورد Git اعمال شود. اگر این را نمی خواهید و فقط می خواهید از Git با کنسول جدا شده خود "Git Bash" استفاده کنید ، اولین گزینه را انتخاب کنید. بنابراین ، برای استفاده از Git ، باید آن را از لیست برنامه ها یا از فهرست زمینه یک پوشه (اگر تصمیم به نصب ادغام Windows Explorer گرفتید) را راه اندازی کنید.

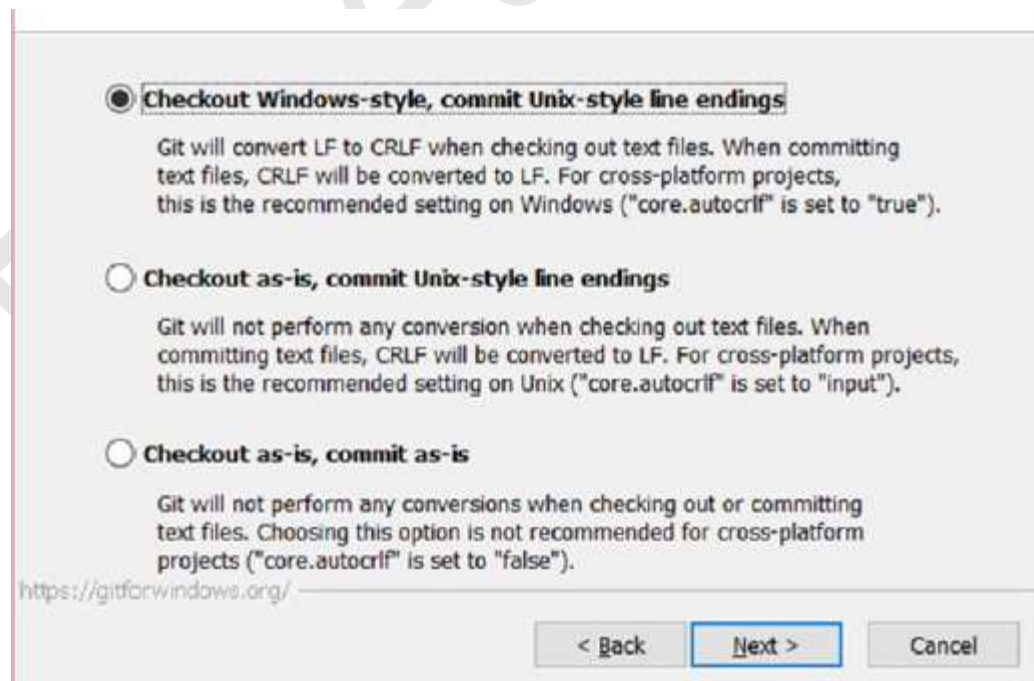
اگر می خواهید همه جا از Git استفاده کنید ، گزینه پیش فرض را اضافه کنید تا آن را به محیط PATH اضافه کنید. به این ترتیب ، ابزارهای دیگر نیز می توانند از آن استفاده کنند و شما می توانید از هر پنجره فرمان کار کنید. من این گزینه را به شدت پیشنهاد می کنم.

گزینه آخر کمی تهاجمی است. بسیاری از دستورات Git را به PATH شما اضافه می کند و برخی از ابزارهای پیش فرض ویندوز را رونویسی می کند. فقط اگر یک دلیل معتبر داشته باشید این را انتخاب کنید. به طور کلی ، شما چنین دلیلی ندارید.

گزینه ای را مانند شکل بالا انتخاب کنید و به مرحله بعدی بروید. شما به صفحه ای در مورد اتصالات HTTPS خواهید رسید ، که در شکل زیر نشان داده شده است. باید هنگام ارسال داده از طریق HTTPS ، از کدام کتابخانه استفاده کنید. بعداً در این کتاب ، شما باید به یک سرور راه دور وصل شوید (از آنجا که Git یک توزیع VCS است) تا تعهدات خود را با افراد دیگر به اشتراک بگذارید ، بنابراین باید تمام این اتصالات رمزگذاری شوند تا اطلاعات شما بیشتر تضمین شود و از سرقت آنها اطمینان حاصل شود.



فقط از گزینه پیش فرض استفاده کنید مگر اینکه دلیلی برای (خط مشی شرکت یا راه اندازی امنیت کمی خود) داشته باشید. پس از این ، به مرحله بعدی بروید که مربوط به پایان خط است. یک بار دیگر ، صفحه نمایش انتخابی است ، بنابراین شما باید مانند صفحه نمایش داده شده در شکل زیر ظاهر شوید. سیستم عامل های مختلف فایل های متنی را به طور متفاوتی کار می کنند ، به خصوص هنگام برخورد با پایان های خط. و احتمال اینکه تیمی که با آن کار خواهید کرد ، از سیستم عامل های مختلف استفاده کند. بنابراین ، Git باید قبل از به اشتراک گذاشتن تعهدات ، انتهای خط را به و از هر سبک پایان تبدیل کند.





همانطور که از ویندوز استفاده خواهید کرد ، باید گزینه پیش فرض را بررسی کنید. در صورت عدم دقت در انتهای خط ، دو گزینه دیگر آسیب های زیادی را به شما وارد می کند. بعد از انتخاب گزینه پیش فرض می توانید به مرحله بعدی بروید. مرحله بعدی انتخاب یک ترمینال پیش فرض (یا کنسول) ، شبیه ساز است. این یک صفحه انتخاب ساده مانند گذشته است که در شکل بعد نشان داده شده است. Git Bash برای کار به یک شبیه ساز کنسول نیاز دارد ، بنابراین باید یکی را انتخاب کنید. شبیه ساز پیش فرض MinTTY است ، گزینه دیگر کنسول پیش فرض ویندوز است.

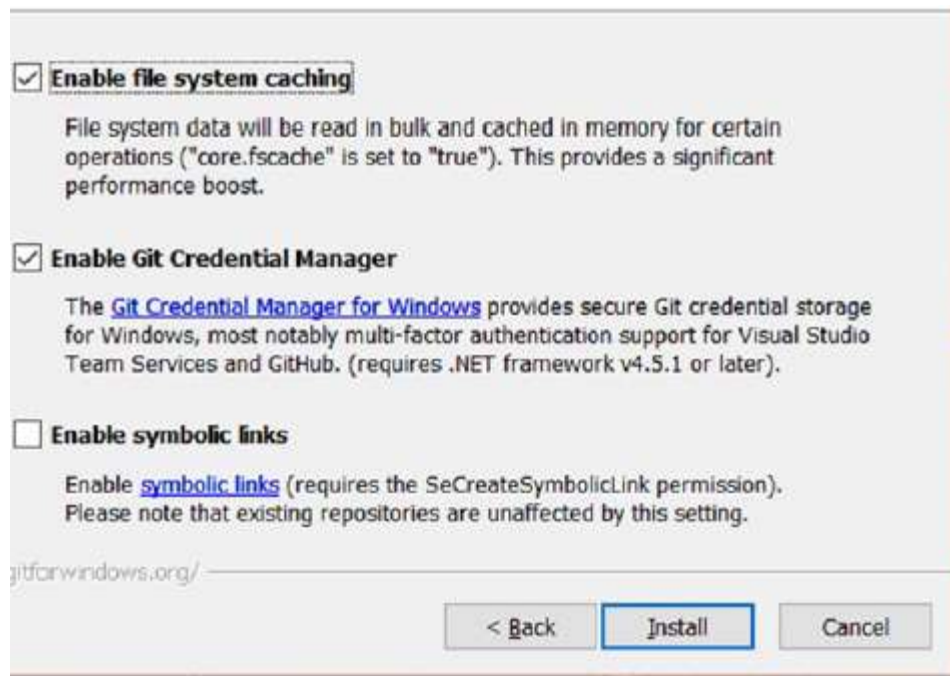


### *Choosing a terminal emulator*

من پیشنهاد می کنم گزینه پیش فرض را نگه دارید زیرا MinTTY می تواند همه کارهایی را انجام دهد که پنجره کنسول ویندوز می تواند اما از هر نظر بهتر است. برای رفتن به مرحله آخر ، روی **next** کلیک کنید.

ما اکنون در آخر بازی هستیم! این نصب تقریباً تمام شده است. فقط چند مورد برای تغییر در صفحه گزینه های اضافی. این صفحه (در شکل زیر نشان داده شده) به شما امکان می دهد برخی از ویژگی های اضافی را که با نصب Git شما عالی خواهد بود ، فعال کنید. به عنوان مثال ، مدیر Git Credential به شما کمک می کند تا به صورت ایمن به سرورهای راه دور متصل شوید و با سایر ابزارهای Git به خوبی بازی کنید.





### Configuring extra options

فقط گزینه های پیش فرض را ترک کنید مگر اینکه دلیل نداشته باشید. پس از آن ، فقط نصب را راه اندازی کنید و بگذارید تمام شود. و همین گیت در سیستم ویندوز شما نصب شده است. اما قبل از استفاده از آن ، به قسمت بعدی پرش کنید تا به درستی تنظیم شود!

مک

اگر قبلاً پیشرفت خاصی با نرم افزار Mac OS X انجام داده اید ، احتمالاً قبلاً Git را دارید زیرا با XCode نصب شده است (<https://developer.apple.com/xcode/>). می توانید بررسی کنید که آیا Gitby دستور را از کنسول خود اجرا کرده است:

### \$ git --version

باید نسخه Git را که در حال حاضر نصب شده است به شما بدهد یا اگر نصب نشده باشد ، به شما اجازه می دهد ابزارهای خط فرمان XCode را نصب کنید. اگر نصب را در آن فوریت انتخاب کنید ، Git نصب خواهد شد و می توانید از بقیه این بخش پرش کنید.

برای نصب Git در Mac خود ، فقط باید به لینک دانلود <https://git-scm.com/download/mac> بروید ، و همانطور که در شکل زیر نشان داده شده است ، بارگیری باید به طور خودکار شروع شود. پرونده بارگیری شده را اجرا کنید و نصب شروع می شود. خیلی راحت است



### Download screen for Mac

همچنین می توانید برای نصب آن از Homebrew (<https://brew.sh/>) استفاده کنید. فقط دستور را اجرا کنید:

```
$ brew install git
```

این حدود نیمی از جهان را نصب می کند اما سرانجام متوقف می شود ، و Git نصب می شود. برای سیستم عامل Mac X X ، نصب Git به آسانی انجام می شود و احتمالاً قبلاً آن را دارید.

لینوکس

اگر مرتباً از لینوکس استفاده می کنید ، احتمالاً از توزیع من نسبت به من اطلاعات بیشتری دارید. بنابراین ، نصب Git با مدیر بسته خود ممکن است یک تکه کیک برای شما باشد. برای توزیع Ubuntu- و Debian ، از APT برای نصب Git استفاده می کنید.

```
$ sudo apt-get install git
```

or

```
$ sudo apt install git (for newer systems)
```

```
$ sudo yum install git
```

or

```
$ sudo dnf install git (for newer systems)
```

## راه اندازی Git

قبل از شروع استفاده از Git ، ابتدا به کمی تنظیم نیاز دارید. احتمالاً فقط یک بار این کار را خواهید کرد زیرا تمام تنظیمات در یک پرونده جهانی خارجی ذخیره شده اند ، به این معنی که تمام پروژه های شما پیکربندی های یکسانی را به اشتراک می گذارند. همچنین راهی برای پیکربندی پروژه ها یکی یکی وجود دارد اما بعداً خواهیم دید.

از آنجا که Git یک سیستم کنترل نسخه توزیع شده است ، باید یک روز به سایر مخازن از راه دور متصل شوید. برای جلوگیری از اشتباه هرگونه هویت ، لازم است کمی درباره خودتان بگویید. نگران نباش در مورد شما یک واقعیت جالب نخواهد پرسید!

برای راه اندازی Git ، Git Bash ، (برای سیستمهای ویندوز) یا پنجره پیش فرض کنسول را باز کنید (برای سیستمهای لینوکس / macOS یا ویندوز که محیط PATH خود را اصلاح کرده است). در قسمت فرمان ، فقط نام و آدرس ایمیل خود را به Git بگویید:

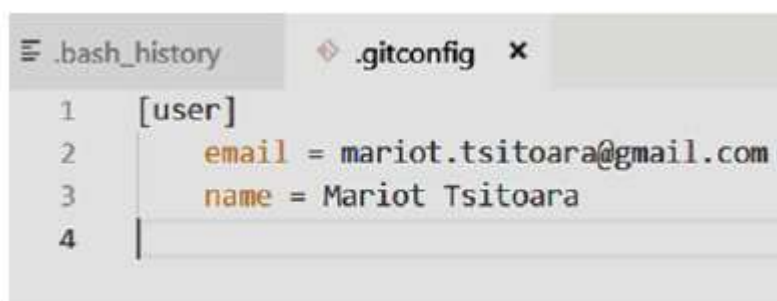
```
$ git config --global user.name "Mariot Tsitoara"
$ git config --global user.email "mariot.tsitoara@gmail.com"
```

به استدلال "جهانی" توجه کنید. این بدان معناست که راه اندازی برای همه مخازن آتی در آینده است ، بنابراین لازم نیست که در آینده مجدداً این کار را انجام دهید.

با دستور پیکربندی ، می توانید ویرایشگر پیش فرض خود را نیز تغییر دهید. شما همیشه می خواهید ویرایشگر خود را تغییر دهید زیرا شما یک نسخه جدید را پیدا کرده اید یا آن را حذف نکرده اید ، دستور پیکربندی برای کمک به شما وجود دارد. به عنوان مثال ، برای تغییر ویرایشگر پیش فرض به نانو ، تایپ می کنید.

```
$ git config --global core.editor="nano"
```

می توانید پرونده ای که پیکربندی Git خود را در پوشه خانه خود ثبت کرده است را پیدا کنید. برای ویندوز ، می توانید آن را در C: \ Users \ YourName \ .gitconfig پیدا کنید. برای لینوکس و سیستم عامل مک ، می توانید آن را در home/yourname/.gitconfig/ مطابق شکل زیر پیدا کنید.

A screenshot of a code editor window with two tabs: ".bash\_history" and ".gitconfig x". The ".gitconfig" tab is active and shows the following content:

```
1 [user]
2   email = mariot.tsitoara@gmail.com
3   name = Mariot Tsitoara
4
```

### *My .gitconfig file*

در کنار پرونده .gitconfig ، ممکن است پرونده دیگری با نام .bash\_history پیدا کنید که تمام فرمانی را که تایپ می کنید در کنسول ثبت می کند. اگر می خواهید دستور دیگری را که فراموش کرده اید ، دوباره بررسی کنید.

## خلاصه

بگذارید آنچه را که تاکنون آموخته ایم ، مرور کنیم! اول ، شما باید تا به حال Gitinstall بر روی سیستم خود داشته باشید. مراحل نصب بر روی ویندوز بسیار آسان و در مک و لینوکس آسان تر است. پیشنهاد می کنم اگر مطمئن نیستید که به آنچه نیاز دارید ، تمام گزینه های پیش فرض (حتی اگر در تصاویر قبلی نشان داده نشده باشند) را نگه دارید.

بعد ، راه اندازی وجود دارد. شما فقط باید در هر سیستمی که Git را در آن نصب کرده اید یک بار این کار را انجام دهید. Git از نام و ایمیل شما برای امضای هر عملی که انجام می دهید استفاده خواهد کرد ، بنابراین لازم است قبل از استفاده ، آنها را تنظیم کنید.

و همین شما اکنون آماده هستید تا با تمام جلال و جلال خود از Git استفاده کنید. برای شروع با پرش به Git ، به فصل بعدی بروید.

# فصل چهارم

## Getting Started

سرانجام برای شروع کار با Git آماده هستید! در این فصل ، شما باید چندین اصطلاحات و مفاهیم Git لازم برای هر پروژه را بیاموزید. سپس ، به شما وظیفه ایجاد یک پروژه ، ایجاد تغییرات در آن ، بررسی تغییرات و در نهایت حرکت بین نسخه ها را می دهد.

### مخزن

فروشگاهی است که تمام پروژه شما و کلیه تغییرات ایجاد شده در آن نگهداری می شود. شما می توانید از آن به عنوان "تغییر پایگاه داده" استفاده کنید. اما نگران نباشید این فقط یک پوشه معمولی در سیستم شما است ، بنابراین دستکاری بسیار آسان است. برای هر پروژه ای که می خواهید با Git مدیریت کنید ، باید یک مخزن برای آن تنظیم کنید. راه اندازی مخزن بسیار آسان است. فقط به پوشه مورد نظر خود ردیابی کنید و به Git بگویید که یک مخزن را در آنجا شروع کند.

بنابراین برای هر پروژه ای که می خواهید شروع کنید ، باید

- دایرکتوری حاوی پروژه خود را ایجاد کنید

- به فهرست بروید

- یک مخزن Git را اولیه کنید

دیدن؟ خیلی راحت است بیایید آن اظهارات را به دستورات تبدیل کنیم. اما ابتدا ، یک کنسول را برای تایپ دستورات ما باز می کنیم. برای کاربران لینوکس ، شما فقط باید پایانه دلخواه خود را راه اندازی کنید (Ctrl-Alt-T برای دبیان مانند دایرکتوری). برای macOS ، فقط کافی است از Cmd-Space استفاده کنید تا Spotlight را در جایی جستجو کنید که بتوانید برنامه Terminal را جستجو کنید. کاربران ویندوز می توانند دو کنسول را باز کنند: cmd و Powershell. Powershell مدرن تر و دارای دستورات شبیه به UNIX است. برای باز کردن یکی از آنها ، از Windows-R استفاده کرده و نام (cmd یا PowerShell) را تایپ کنید. توجه داشته باشید که در صورت باز بودن آنها ، باید همه این کنسول ها را در اولین نصب Git خود مجدداً راه اندازی کنید. Git for Windows همچنین از یک شبیه ساز کنسولی به نام Git Bash بهره می برد که محیطی مشابه کنسول های لینوکس و مک را فراهم می کند. اگر از ویندوز استفاده می کنید ، من اکیداً استفاده از Git Bash را پیشنهاد می کنم تا بتوانید همان تجربه را با سایر افرادی که از سیستم عامل های مختلف استفاده می کنند داشته باشید.

Git Bash (از لیست برنامه ها یا منوی متنی) را باز کنید و آن دستورات را تایپ کنید:

```
$ mkdir mynewproject
$ cd mynewproject/
$ git init
```

mkdir دستوری است که برای ایجاد فهرست استفاده می شود. برای "ایجاد فهرست" کوتاه است. cd دستوری است که برای پیمایش بین دایرکتوری ها استفاده می شود. برای "تغییر فهرست" کوتاه است.

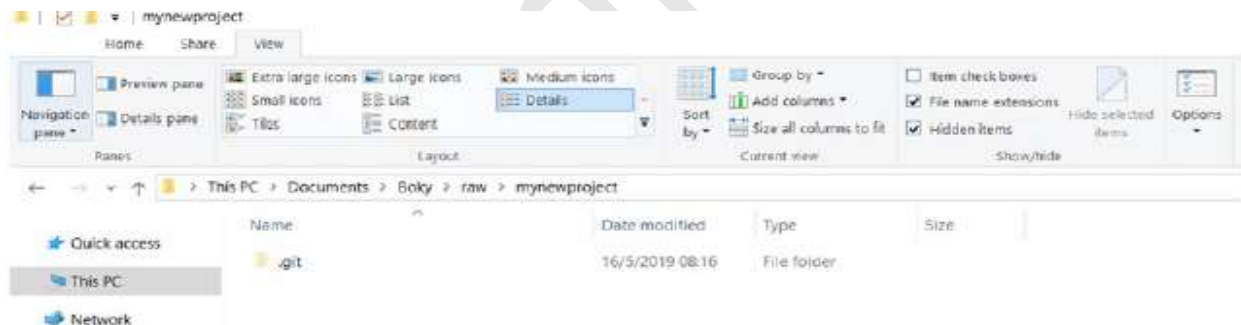
سرانجام ، شروع git برای "شروع اولیه Git" کوتاه است. بعد از شروع مخزن ، Git به شما می گوید که پایگاه داده مانند شکل چگونه ایجاد شده است.

```
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw (master)
$ mkdir mynewproject

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw (master)
$ cd mynewproject/

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/mynewproject (master)
$ git init
Initialized empty Git repository in C:/Users/Mariot/Documents/Boky/raw/mynewproject/.git/
```

Git یک دایرکتوری با نام ".git" ایجاد خواهد کرد که شامل تمام تغییرات و عکس های فوری شما می شود. اگر می خواهید آن را بررسی کنید ، باید فایل های پنهان را از تنظیمات کاوشگر پرونده خود نشان دهید. مخزن مانند دایرکتوری است که در شکل نشان داده شده است.



### *An empty repository*

و اگر دایرکتوری .git را باز کنید ، موارد دیگری پیدا خواهید کرد که بخشی از پایگاه داده Git هستند. شکل را برای مثال بررسی کنید.

Name	Date modified	Type	Size
hooks	16/5/2019 08:16	File folder	
info	16/5/2019 08:16	File folder	
objects	16/5/2019 08:16	File folder	
refs	16/5/2019 08:16	File folder	
config	16/5/2019 08:16	File	1 KB
description	16/5/2019 08:16	File	1 KB
HEAD	16/5/2019 08:16	File	1 KB

به یاد داشته باشید فصل 2 که گفته بود به جای پیگیری تغییرات بین نسخه ها ، Git عکس های فوری می گیرد؟ خوب ، تمام عکسهای فوری در فهرست ".git" ذخیره می شوند. هر عکس فوری "commit" خوانده می شود ، و ما کمی بعد از این بخش بررسی خواهیم کرد.

پرونده HEAD در این فهرست ".git" به "branch" یا براندازی فعلی پروژه ای که در آن کار می کنید اشاره می کند. شاخه پیش فرض "master" خوانده می شود ، اما دقیقاً مانند هر شاخه دیگر است. نام فقط یک کنوانسیون قدیمی است. همچنین باید بدانید که اولیه سازی تنها راه برای تهیه مخزن است. می توانید یک مخزن کامل را با تمام تاریخچه و عکس های فوری آن کپی کنید. "cloning" نامیده می شود و در فصل دیگری آن را خواهیم دید.

## دایرکتوری کار

چه در مورد منطقه خالی خارج از فهرست ".git"؟ این دایرکتوری کار نامیده می شود و پرونده هایی که روی آن کار می کنید در همان جا ذخیره می شوند. معمولاً جدیدترین نسخه شما در فهرست کارها قرار خواهد گرفت. هر پرونده ای که روی آن کار می کنید در فهرست کار است. هیچ چیز خاصی در مورد این مکان وجود ندارد ، مگر اینکه فقط پرونده ها را مستقیماً در اینجا دستکاری کنید. هرگز پرونده ها را در فهرست ".git" تغییر ندهید! Git فایل جدیدی را که در فهرست کار قرار می دهید تشخیص می دهد. و با استفاده از "وضعیت" دستور Git ، وضعیت دایرکتوری را بررسی می کنید.

## \$ git status

به عنوان مثال ، اگر یک فایل جدید با نام README.md را در WorkDirectory ایجاد کنیم ، خواهیم دید که Git می داند که پروژه تغییر کرده است. اطمینان حاصل کنید که پرونده جدید خود را در کنار آن قرار داده اید. پوشه git مانند شکل ، نه در آن!

This PC > Documents > Boky > raw > mynewproject			
Name	Date modified	Type	Size
.git	16/5/2019 20:36	File folder	
README.md	16/5/2019 20:34	Markdown Source ...	1 KB

### Creation of a new file in the Working Directory

اگر وضعیت دایرکتوری کار را بررسی کنیم ، نتیجه ای مانند آنچه در شکل بعد نشان داده شده است ، می گیریم. همانطور که در شکل بعد مشاهده می کنید ، ما هنوز هیچ commit نداریم. دلیل این است که ما هنوز در دایرکتوری کار هستیم و هنوز هیچ عکس فوری نگرفته ایم. همچنین می گوید که ما در شاخه "master" هستیم. این نام پیش فرض تنها شاخه ای است که در مرحله اولیه سازی مخزن ایجاد شده است. سپس پرونده های غیرقابل حل را دریافت می کنیم. این پرونده هایی است که ما اصلاح کردیم (در این مثال ، ایجاد شده است).



```
MINGW64:/c/Users/Mariot/Documents/Boky/raw/mynewsite
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/mynewsite (master)
$ touch README.md

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/mynewsite (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README.md

nothing added to commit but untracked files present (use "git add" to track)
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/mynewsite (master)
$ |
```

### *The status of the Working Directory*

در اصل ، آن فهرست کار: منطقه ای است که شما مستقیماً با پرونده های پروژه خود در تعامل هستید.

### منطقه صحنه

Staging Area جایی است که پرونده های شما قبل از اینکه عکسهای فوری عکسبرداری شوند ، می روند. هر پرونده ای که در فهرست کار تغییر داده اید نباید هنگام گرفتن عکس فوری از وضعیت فعلی پروژه در نظر گرفته شود. فقط پرونده هایی که در Staging Area قرار دارند ، تصویربرداری می شوند. بنابراین ، قبل از گرفتن عکس فوری از پروژه ، شما انتخاب می کنید که کدام پرونده ها را تغییر داده تا در نظر بگیرید. تغییر در یک پرونده می تواند ایجاد ، حذف یا ویرایش باشد. به عنوان آن تصور کنید که کدام پرونده ها را در عکس خانوادگی قرار دهید. برای افزودن پرونده به Staging Area ، از دستور Git "افزودن" استفاده می کنیم.

## \$ git add nameofthefile

ساده است. اگر می خواستیم README.md را که قبلاً ایجاد کردیم ، مرحله بندی کنیم ، از "git add README.md" استفاده خواهیم کرد. یا اگر چندین فایل ایجاد کردید ، می توانید یکی پس از دیگری یا با یکدیگر مانند "git add file1 file2" اضافه کنید.

بگذارید پرونده جدید ما را با استفاده از دستور مرحله تنظیم کنیم:

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

سپس وضعیت را با دستور وضعیت git بررسی می کنیم.

```
$ git status
```

اضافه کردن پرونده به قسمت مرحله بندی نتیجه قابل مشاهده ای به همراه نخواهد داشت ، اما بررسی وضعیت نتیجه ای مشابه شکل زیر به شما می دهد.

```
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/mynewsite (master)
$ git add README.md

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/mynewsite (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   README.md
```

اگر شکل بالا را بررسی کنید ، متوجه خواهید شد که پس از تهیه پرونده ، فهرست کار دوباره تمیز می شود. به این دلیل است که "وضعیت git" فقط پرونده های "غیر مرحله ای" را ردیابی می کند (پرونده های ویرایش شده برای عکس فوری مشخص نشده اند).

همانطور که در شکل بالا مشاهده می کنید ، می توانید با استفاده از دستور "git rm --cached" ، فایل را مرتفع کنید.

بعد از اینکه تمام پرونده هایی را که می خواهید تغییرات در نظر گرفته شود مرحله بندی کنید ، اکنون آماده هستید تا اولین عکس فوری خود را بگیرید!

## Commits

همانطور که در بخش قبل از این مورد صحبت کردیم ، یک commit فقط یک عکس فوری از کل پروژه در زمان معین است. Git تغییرات فردی انجام شده در پرونده ها را ضبط نمی کند. از کل پروژه عکس می گیرد. علاوه بر عکس فوری ، یک commit همچنین حاوی اطلاعاتی در مورد "نویسنده" محتوا و "مرتکب" یا کسانی است که تغییرات را در مخزن قرار داده اند.

از آنجایی که یک تعهد عکاسی از وضعیت پروژه است ، وضعیت قبلی این پروژه تعهد دیگری به نام "والدین" است. اولین تعهد توسط Git ایجاد می شود که مخزن ایجاد می شود ، و این تعهدی است که والدین ندارد. سپس همه تعهدات آینده از طریق فرزندپروری به یکدیگر مرتبط می شوند. گروه آن دسته از تعهداتی که والدین یکدیگر هستند ، "branch" نامیده می شود.

یک تعهد با نام آن مشخص می شود ، رشته ای 40 کاراکتری که با داشتن تعهد بدست می آید. این یک هش SHA1 ساده است ، بنابراین چندین تعهد با اطلاعات مشابه دارای همین نام است.

اشاره به یک تعهد خاص "head" نامیده می شود و همچنین دارای یک نام است. و head شما در حال کار بر روی "HEAD" است (بخش قبلی را ببینید).

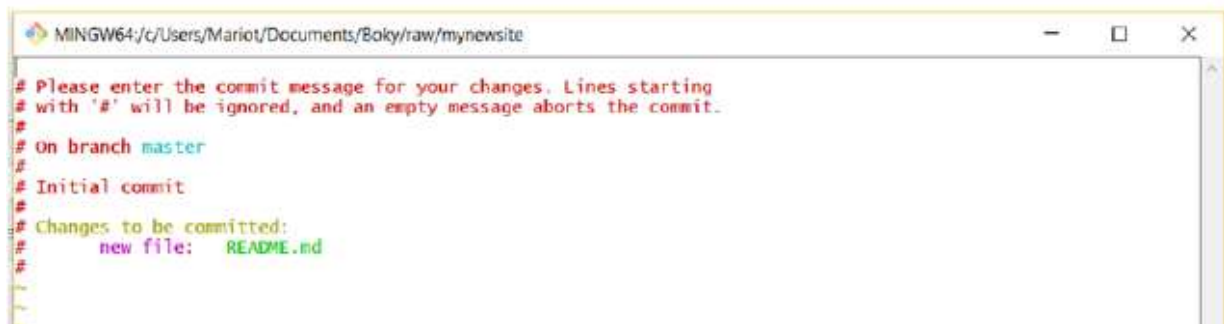
اکنون می‌توانیم پرونده‌هایی را که قبلاً تهیه کرده ایم مرتکب شویم. قبل از هر مرتکب، باید وضعیت فهرست کار و محل استقرار را بررسی کنید. اگر تمام پرونده‌هایی که می‌خواهید مرتکب شوید در منطقه مرحله بندی قرار دارند (تحت عبارت "تغییراتی که باید انجام شود")، می‌توانید مرتکب شوید. اگر اینطور نیست، باید آنها را با "git add" مرحله بزنید.

برای انجام همه تغییراتی که ایجاد کردیم، از "git commit" استفاده می‌کنیم. این یک تصویر لحظه‌ای از وضعیت فعلی پروژه ما خواهد بود.

## \$ git commit

اگر این دستور را اجرا کنیم، ویرایشگر پیش فرض ما را باز می‌کند (اگر می‌خواهید اصلاح خود را تغییر دهید فصل 3 را بررسی کنید) و از ما بخواهید برای پیغام متعهد شوید. پیام متعهد توضیحی کوتاه از آنچه در تعهد نسبت به مورد قبلی تغییر کرده است.

ویرایشگر پیش فرض من Vim است، بنابراین اگر من دستور commit را اجرا کنم، صفحه‌ای را مطابق شکل زیر مشاهده می‌کنم.



در شکل قبل می‌بینید که خط اول پرونده خالی است. که در آن شما باید پیام commit را بنویسید. پیام متعهد باید بر روی یک خط نوشته شود، اما همیشه می‌توانید سطر بیشتری از نظرات اضافه کنید. نظرات با "#" شروع می‌شود و توسط Git نادیده گرفته می‌شود. آنها فقط برای تکمیل پیام commit، برای روشن تر شدن آن استفاده می‌شوند. همچنین توجه داشته باشید که Git لیست فایل‌های تغییر یافته را در نظرات commit به طور خودکار قرار می‌دهد (همان پرونده‌هایی که با "وضعیت git" دیدید).

شما روش‌های مناسب برای نوشتن پیام‌های commit را به روش صحیح در فصل‌های بعدی یاد خواهید گرفت. اما در حال حاضر، فقط یک پیام ساده مانند "اضافه کردن README.md به پروژه" در اولین خط خالی مانند شکل بعد وارد کنید.

```
MINGW64~/c/Users/Mariot/Documents/Boky/raw/mynewsite
Add README.md with the description of the project
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:   README.md
#
```

بعد از اینکه پیام commit خود را مانند شکل بالا نوشتید ، می توانید ویرایشگر را ببندید (پس از ذخیره کردن!). سپس خلاصه ای از commit مانند شکل بعد را دریافت خواهید کرد.

```
MINGW64~/c/Users/Mariot/Documents/Boky/raw/mynewsite
Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/mynewsite (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   README.md

Mariot@lenovo-ideapad MINGW64 ~/Documents/Boky/raw/mynewsite (master)
$ git commit
[master (root-commit) 585b501] Add README.md with the description of the project
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md

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

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

خلاصه این تعهد شامل اطلاعات زیادی خواهد بود:

- branch فعلی: استاد
  - نام commit قبلی: root-commit زیرا این اولین تعهد ما است
  - نام commit: هفت حرف اول هش مرتکب
  - پیام commit
  - تعداد پرونده ها تغییر کرده است: یک پرونده
  - عملیاتی که برای هر پرونده انجام می شود: ایجاد
- اولین عکس فوری خود را گرفتیم! اگر وضعیت مخزن را بررسی کنید ، می توانید ببینید که دوباره تمیز است ، مگر اینکه برخی از پرونده ها را به صورت ناپایدار رها کنید.

## شروع سریع با Git

بنابراین ، اکنون که شما با مفهوم اصلی Git آشنا شدید ، قصد داریم آنها را در یک پروژه واقعی به کار گیریم. بیایید تصور کنیم می خواهید پوشه ای را برای نگه داشتن لیست TODO خود ایجاد کنید و می خواهیم نسخه شود تا بتوانید بررسی کنید که هر آیتم به اتمام رسیده است. برای اینکه بیشتر با Git آشنا شوید ، بدون هیچ کمکی تمرین بعدی را انجام می دهید. اگر گیر کردید ، فقط بخش های قبلی را برای راهنمایی ها بررسی کنید.

فقط اصول اساسی Git را به خاطر بسپارید:

- پرونده ها را در دایرکتوری کار تغییر دهید.
- پرونده هایی را که می خواهید برای ضبط وضعیت فعلی در قسمت مرحله بندی قرار دهید ، قرار می دهید.
- شما با یک تعهد عکس فوری از پروژه می گیرید.
- فراموش نکنید که قبل از ارتکاب پرونده هایی را که تغییر داده اید روی Staging Area قرار دهید یا بخشی از عکس فوری نباشند ، پرونده های تغییر یافته ای که شما در Staging Area قرار نداده اید ، تا زمانی که تصمیم به دور ریختن نگیرید ، در فهرست شاخه کار خواهند ماند. آنها را در نظر بگیرید با آنها را در آینده متعهد کنند
- بیایید تمرین را شروع کنیم! لطفاً آن را تا آخر تکمیل کنید و به فصل بعد بروید تا اینکه به وضوح درک کنید که Git چگونه کار می کند.

### EXERCISE: یک برنامه TODO VERSIONED

- یک مخزن جدید ایجاد کنید.
- فایلی به نام TODO.txt را در دایرکتوری ایجاد کرده و متن را وارد کنید.
- مرحله TODO.txt.
- پروژه را متعهد کنید و یک پیام کوتاه متعهد بگذارید.
- دو پرونده جدید با نام های DONE.txt و WORKING.txt ایجاد کنید.
- آن پرونده ها را مرحله بندی و مرتکب شوید.
- تغییر نام WORKING.txt به IN PROGRESS.txt.
- مقداری متن را به DONE.txt اضافه کنید.
- وضعیت فهرست را بررسی کنید.
- DONE.txt و Stage IN PROGRESS.txt.
- DONE.txt غیرفعال.
- اجرای پروژه را انجام دهید.

• وضعیت فهرست را بررسی کنید.

پس از اتمام این تمرین ، کتاب را ببندید و سعی کنید آن موارد را برای آنها توضیح دهید  
خودتان به قول خودتان:

Working Directory

Staging Area

Commit

اگر در درک این مفاهیم بیش از حد مشکل ندارید ، شما برای دستورات و مفاهیم بیشتری Git آماده هستید.

## خلاصه

این فصل برای درک شما از Git بسیار مهم است. راههای اصلی آماده سازی سه حالت است که می تواند یک پرونده باشد:

- اصلاح شده: شما پرونده ای را در دایرکتوری کار اصلاح کردید.

- Staged: شما پرونده را به Staging Area اضافه کردید تا بتواند به صورت فوری تصویربرداری شود.

- متعهد: شما یک عکس فوری از کل پروژه گرفتید (کلیه پرونده های اصلاح نشده و مرحله بندی شده).

اگر یک پرونده بخشی از تعهد قبلی بوده است و آنها را اصلاح نکرده اید ، آنها به طور خودکار بخشی از تعهد بعدی خواهند بود. یک پرونده اصلاح شده اما غیرفعال به عنوان اصلاح نشده در نظر گرفته می شود. شما باید از Git بخواهید که با نمایش آن پرونده ها ، آنها را ردیابی کند.

همچنین در مورد ارتکاب و ارتکاب پیام ها کمی یاد گرفتیم. باز کردن ویرایشگر خارجی برای نوشتن پیام های متعهد ممکن است در ابتدا کمی ناخوشایند باشد ، اما در نهایت پس از مدتی قطع آن خواهید شد.

در فصل بعد یاد خواهیم گرفت که چگونه تاریخ پروژه را بررسی کنیم و بین نسخه ها حرکت کنیم. ما همچنین در مورد نادیده گرفتن خواهیم آموخت