# A picture containing symbol, logo, clipart, electric blue Description automatically generated

**دانشگاه آزاد اسلامي واحد دماوند**

**گروه كامپيوتر**

**موضوع گزارش: مقدمه ای بر Git & Github**

**درس: روش پژوهش و ارائه**

**استاد: جناب آقای دکتر بهرنگ مسعودی‌فر**

**تهیه کننده: یاسمین آشوری**

**شماره دانشجویی: 982142068**

**نیمسال دوم سال تحصیلی 1402-1401**

**فهرست**

مقدمه..............................................................................................................................................................................................4

1. مقدمات Git

1-1.تاریخچه Git.........................................................................................................................................................................................4

1-2. چرا به Git نیاز داریم؟......................................................................................................................................................................5

1-3. Git چیست؟.......................................................................................................................................................................................6

1-4. About Version Control.........................................................................................................................................................6

Centralized Version Control Systems1-4........................................................................................................................7

Decentralized Version Control Systems1-4..................................................................................................................7

1-5.Snapshots.........................................................................................................................................................................................8

1-6.Git دارای یکپارچگی است.............................................................................................................................................................10

1-7.معرفی GitHub, GitLab, GitKraken..............................................................................................................................10

1. شروع کار با Git

2-1: ساختن یک Git Repository..................................................................................................................................................11

Initializing a Repository in an Existing Directory.2-1......................................................................................11

Commit.2-2 دقیقا به چه معناست؟................................................................................................................................................12

2-2.سه مرحله اساسی..............................................................................................................................................................................13

2-3.قوانین نوشتن Commit................................................................................................................................................................15

2-3.نمونه ای از پیام های commit خوب.........................................................................................................................................16

2-3.نمونه ای از پیام های commit بد...............................................................................................................................................16

16…………………..……..……………………………………………Undoing Things.2-4

Local.2-5 چیست؟................................................................................................................................................................................17

Remote.2-6 چیست؟..........................................................................................................................................................................18

Pushing to Your Remotes.2-6................................................................................................................................................18

Git push.2-7...................................................................................................................................../...................................................19

Cloning.2-8 یک مخزن موجود........................................................................................................................................................20

1. مقدمات Branching

Git Branching.3-1..............................................................................................................................................................................................20

Branch.3-1 دقیقا چیست؟...................................................................................................................................................................................21

Branching.3-1 به طور خلاصه..........................................................................................................................................................................22

3-2.ساختن یک Branch جدید.............................................................................................................................................................................24

Switching Branches.3-3..................................................................................................................25

Basic Branching and Merging.3-4..................................................................................................28

Basic Branching3-4........................................................................................................................28

Basic Merging.3-4..........................................................................................................................30

Basic Merge Conflicts.3-4..............................................................................................................32

Changing a branch name.3-5..........................................................................................................35

Rebasing.3-6...................................................................................................................................36

36……………...…………...……………………...…………………………The Basic Rebase.3-6

1. Contributing to a Project

Pulling.4-1......................................................................................................................................39

Forking Projects.4-2.......................................................................................................................39

Creating a Pull Request.4-3............................................................................................................40

جمع‌بندی ...............................................................................................................................................................................................................43

منابع.............................................................................................................................................................................................................................44

**مقدمه**

در این گزارش، به توضیحات اولیه برای کار با گیت می‌پردازیم و مقدمات لازم را جهت کار و همکاری با دیگر افراد پروژه، فراهم می‌کنیم.

بخش اول: در این بخش، توضیحات مقدماتی را به شما خواهیم گفت، که به درک اولیه کمک می‌کند و زمینه را برای یادگیری بیشتر فراهم می‌سازد. درمورد تاریخچه گیت، انواع ورژن‌های کنترل، ویژگی‌های گیت و دلایل استفاده از آن صحبت می‌شود.

بخش دوم: در این بخش به دستورات مهم گیت می‌پردازیم و به صورت عملی کار با گیت را یاد خواهید گرفت. درمورد نحوه نوشتن commit، ساختن مخزن گیت، مخزن محلی و مخزن remote صحبت خواهیم کرد.

بخش سوم: در این بخش به آشنایی با شاخه‌ها، نحوه انتخاب شاخه، تغییر نام شاخه، ادغام و دیگر بحث های مربوط به شاخه‌ها خواهیم پرداخت.

بخش چهارم: در این بخش نکاتی را به شما خواهیم آموخت که شما را برای همکاری با دیگر افراد بر روی پروژه‌ها آماده می‌کند. این بخش شامل دستورات Fork، pull request و دیگر دستورات مهم می‌باشد.

**1.مقدمات Git**

**1-1. تاریخچه Git**

مانند بسیاری از چیزهای بزرگ در زندگی، گیت با تخریب خلاقانه و جنجال های آتشین شروع شد. هسته لینوکس یک پروژه نرم افزاری متن ‌باز با قلمروی نسبتاً بزرگ است. در طول سالهای اولیه نگهداری هسته لینوکس، تغییرات در نرم افزار به عنوان patches[[1]](#footnote-1) و فایل های آرشیو شده آن ارسال شد. در سال 2002 پروژه هسته لینوکس با استفاده از یک DVCS اختصاصی به نام BitKeeper توسعه داده شد.

در سال 2005، رابطه تجاری بین جامعه ای که هسته لینوکس را توسعه داد و شرکتی که BitKeeper را توسعه داده بود از بین رفت و وضعیت رایگان بودن این ابزار لغو شد. این امر باعث شد تا جامعه توسعه لینوکس و به ویژه لینوس توروالدز، خالق لینوکس ابزار خود را بر اساس برخی از درس‌هایی که هنگام استفاده کردن از BitKeeper آموخته بود توسعه دهد. برخی از اهداف آن به شرح زیر بود:

• سرعت

• طراحی ساده

• پشتیبانی قوی برای توسعه غیرخطی (هزاران شاخه موازی)

• کاملا توزیع شده

• قادر به مدیریت پروژه های بزرگ مانند هسته لینوکس به طور موثر (سرعت و اندازه داده)

از زمان تولد گیت در سال 2005، گیت تکامل یافته است و به گونه ای رشد کرده است که استفاده از آن آسان است و در عین حال مواردی که در بالا به آن اشاره کردیم را رعایت کرده است. گیت برای پروژه های بزرگ بسیار کارآمد است و سیستم انشعاب آن برای توسعه غیر خطی فوق العاده است. [ 1 ]: 14

**1-2. چرا به Git نیاز داریم؟**

نیاز به یک سیستم کنترل نسخه، باعث شده است که یادگیری گیت از اهمیت بالایی برخوردار باشد.

هر پروژه نرم افزاری با یک ایده شروع می‌شود، که ایده اصلی آن در کد منبع پیاده سازی شده است. اما فایل آن چیزی است که به ما قدرت می‌دهد تا برنامه های کاربردی را توسعه دهیم، بنابراین ما باید با احتیاط با آنها رفتار کنیم. باید مطمئن شویم که تاریخچه تغییرات را حفظ می‌کنیم و امکان همکاری بین افراد را فراهم می‌‌کنیم. توسعه دهندگان و سازمان های جهانی عاشق گیت هستند. اما چه چیزی باعث محبوبیت گیت شده است؟ احتمالا بازی ‌های ویدیویی انجام داده باشید که با پیشروی در بازی، در برخی از نبردها پیروز شده و در برخی دیگر شکست می‌خورید. هر چند وقت یکبار ممکن است به بیش از یک بار تلاش برای به پایان رساندن یک چالش خاص نیاز داشته باشید. بسیاری از بازی ها به شما این امکان را می‌دهند که پیشرفت خود را ذخیره کنید. ذخیره کردن پیشرفت، یک "snapshot" از بازی همانطور که در حال حاضر است ایجاد می‌کند. خبر خوب این است که اکنون اگر با اتفاق ناگواری مواجه ‌شوید، دیگر مجبور نخواهید بود که به نقطه اول برگردید. در عوض، شما به سادگی عکس فوری ای را که قبلا گرفته‌اید دوباره بارگیری کنید.

در واقع «عکس فوری» راهی برای ثبت مجموعه‌ای از تغییرات است، بنابراین حتی اگر تغییراتی را در مجموعه‌ای از فایل‌های پروژه خود ایجاد کرده باشید، همه در یک عکس فوری ذخیره می‌شود. به این معنی که اگر اشتباهی مرتکب شدید یا شاید از نسخه فعلی پروژه راضی نباشید، می‌توانید به عکس قبلی خود برگردید. از سوی دیگر، اگر راضی هستید فقط یک عکس فوری دیگر ایجاد می‌کنید و به کار خود ادامه می‌دهید. یک سیستم کنترل نسخه مانند گیت به شما این امکان را می‌دهد که با اطمینان خاطر با توسعه دهندگان دیگر بر روی مجموعه ای از فایل ها همکاری کنید.

گیت را به ‌عنوان بانک حافظه و پلتفرم همکاری در نظر بگیرید که همه در یک واحد ساخته شده‌اند. درک کنترل نسخه، و به طور خاص، درک اینکه قادر به انجام چه کارهایی است و تأثیری که بر نحوه کار ما دارد، می‌تواند به ما کمک کند و در پروژه سازنده باشد. [ 2 ]: 58

**1-3. Git چیست؟**

بنابراین، به طور خلاصه گیت چیست؟ دانستن مفهوم آن مهم است، زیرا اگر بدانید گیت چیست و اصول کارکرد آن چیست، احتمالاً استفاده مؤثر از آن برای شما بسیار آسانتر خواهد بود. همانطور که گیت را یاد می‌گیرید، سعی کنید ذهن خود را از چیزهایی که ممکن است در مورد سایر VCS ها بدانید، پاک کنید. انجام این کار به شما کمک می‌کند از سردرگمی در هنگام استفاده از این ابزار جلوگیری کنید. حتی با وجود اینکه رابط کاربری گیت تقریباً مشابه انوع VCS است، گیت اطلاعات را به روشی بسیار متفاوت ذخیره می‌کند و درک این تفاوت ها به شما کمک می‌کند هنگام استفاده از آن سردرگم نشوید.14 :[ 1 ]

**About Version Control .1-4**

"کنترل نسخه" چیست و چرا باید به آن اهمیت دهید؟ کنترل نسخه سیستمی است که تغییرات یک فایل یا مجموعه ای از فایل ها را در طول زمان ثبت می‌کند تا بتوانید نسخه های خاصی را بعداً فراخوانی کنید. یک سیستم کنترل نسخه (VCS) ابزار بسیار عاقلانه‌ ای برای استفاده است. این سیستم شما را قادر می‌سازد که فایل های انتخاب شده را به حالت قبلی برگردانید، کل پروژه را به حالت قبلی برگردانید، تغییرات را در طول زمان مقایسه کنید، ببینید چه کسی آخرین بار چه چیزی را تغییر داده است، مشکلی را در قسمت issues مطرح کنید. استفاده از VCS همچنین به طور کلی به این معنی است که اگر بخشی از پروژه را خراب کنید یا فایل ها را گم کنید، می‌توانید به راحتی آن ها را بازیابی کنید. [ 1 ]: 10

**Centralized Version Control Systems.1-4**

مشکل اصلی این است که باید با توسعه دهندگان در سیستم های دیگر همکاری کنیم و این قابلیت مهم در نسخه های قبلی کنترل نسخه وجود نداشت پس برای مقابله با این مشکل، سیستم های کنترل نسخه متمرکز (CVCSs) توسعه یافتند.

این سیستم ها (مانند CVS، Subversion و Perforce) دارای یک سرور واحد هستند که شامل تمام فایل های نسخه شده و تعدادی clientهستند که فایل ها را از آن مکان مرکزی بررسی می‌کنند. برای چندین سال، این استاندارد محبوب برای کنترل نسخه به شمار می‌رفت. [ 1 ]: 11

A picture containing text, font, design, tripod

Description automatically generated

Figure1. Centralized version control diagram

**Distributed Version Control Systems.1-4**

اینجاست که سیستم ‌های کنترل نسخه توزیع‌ شده (DVCS) وارد عمل می‌شوند. در یک DVCS (مانند Git، Mercurial، Bazaar)، مشتریان فقط آخرین عکس فوری فایل‌ها را بررسی نمی‌کنند. به جای آن، آنها به طور کامل مخزن را از جمله تاریخچه کامل آن را منعکس[[2]](#footnote-2) می‌کنند. بنابراین، اگر هر سروری از بین برود این سیستم ‌ها از طریق آن سرور با یکدیگر همکاری می‌کنند و هر یک از مخازن client برای بازیابی آن به سرور کپی می‌شوند. هر clone در واقع یک نسخه پشتیبان کامل از تمام داده ها است.

علاوه بر این، بسیاری از این سیستم‌ها با داشتن چندین مخزن remote می‌توانند با یکدیگر کار کنند، پس می‌توانید با انواع مختلف مردم به روشهای مختلف به طور همزمان در یک پروژه همکاری کنید. این قابلیت به شما این امکان را می‌دهد که چندین نوع workflowsرا تنظیم کنید که در سیستم های متمرکز امکان پذیر نیست، مانند مدل های hierarchical. [ 1 ]: 12

A diagram of a computer database

Description automatically generated with low confidence

Figure 2. Distributed version control diagram

**Snapshots.1-5**

تفاوت عمده بین گیت و هر VCS دیگری طرز فکر گیت در مورد داده های خود است. از نظر مفهومی اکثر سیستم های دیگر اطلاعات را به عنوان لیستی از تغییرات مبتنی بر فایل ذخیره می‌کنند. این سیستم های دیگر (CVS، Subversion، Perforce، Bazaar، و غیره) اطلاعاتی را ذخیره می‌کنند که این اطلاعات را به عنوان مجموعه ای از فایل ها و تغییرات ایجاد شده در هر فایل در طول زمان در نظر می‌گیرند (که معمولاً به عنوان کنترل نسخه مبتنی بر دلتا توصیف می شود).

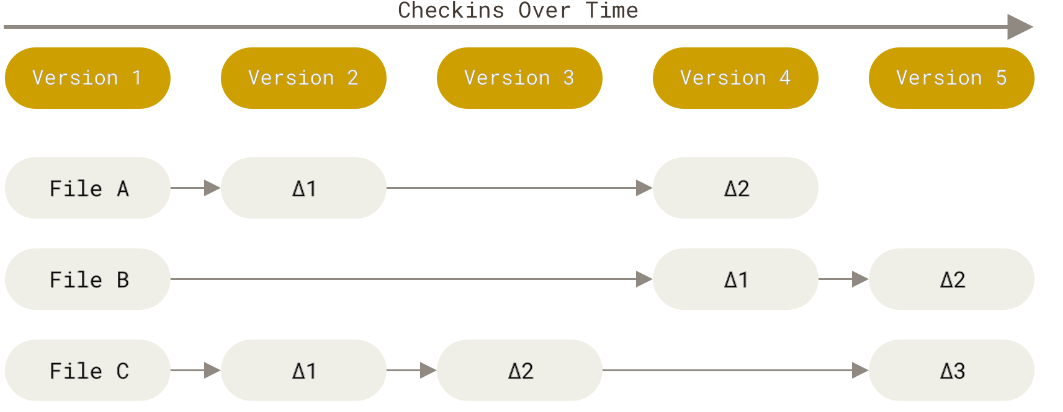


Figure3. Storing data as changes to a base version of each file

گیت داده های خود را به این شکل در نظر نمی‌گیرد یا ذخیره نمی‌کند. در عوض به داده های خود بیشتر شبیه مجموعه ای از عکس های فوری از یک فایل سیستم مینیاتوری نگاه می‌کند. با گیت، هر بار که پروژه خود را commit یا ذخیره می‌کنید، گیت اساساً تصویری از ظاهر تمام فایل‌های شما در آن لحظه می‌گیرد و مرجعی به آن عکس فوری ذخیره می‌کند. برای کارآمد بودن، اگر فایل‌ تغییر نکرده باشد، گیت فایل را دوباره ذخیره نمی‌کند، فقط یک پیوند به فایل مشابه قبلی که قبلاً ذخیره کرده است می‌دهد. و همانطور که گفتیم گیت بیشتر به داده های خود مانند جریانی از عکس های فوری فکر می‌کند.

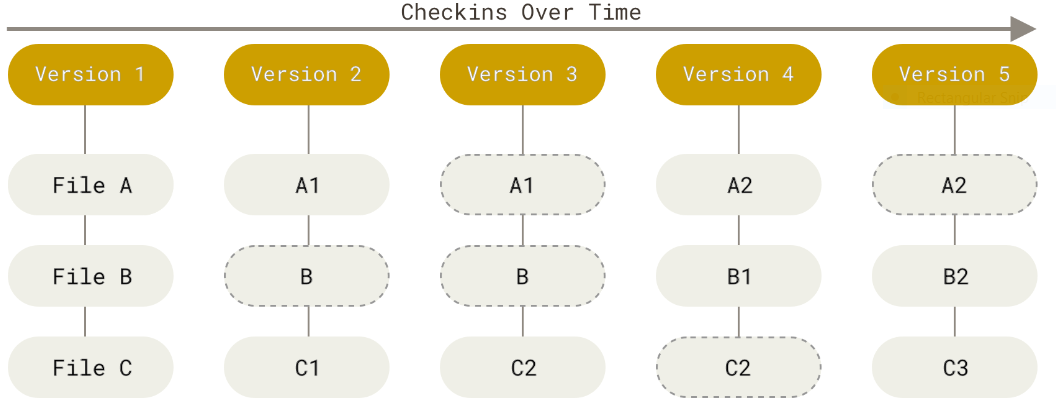


Figure4. Storing data as snapshots of the project over time

این یک تمایز مهم بین گیت و تقریباً تمام VCS های دیگر است. و گیت تقریباً در تمام جنبه‌های کنترل نسخه که اکثر سیستم‌های دیگر از نسل قبلی کپی کرده‌اند متفاوت است و باعث می‌شود گیت بیشتر شبیه یک فایل سیستم کوچک با ابزارهای فوق‌العاده قدرتمند باشد نه صرفاً یک VCS. [ 1 ]: 14

**Git.1-6 دارای یکپارچگی است**

همه چیز در آن قبل از ذخیره شدن checksummed می‌شود. این بدان معناست که تغییر محتویات هر فایل یا دایرکتوری بدون اطلاع گیت غیرممکن است. این قابلیت در پایین ترین سطوح در گیت تعبیه شده است و یکپارچگی یکی از فلسفه های ساخت آن است. مکانیزمی که برای ساخت آن استفاده شده است SHA-1 نامیده می‌شود. که یک رشته 40 کاراکتری است که از کاراکترهای هگزادسیمال (0-9 و a-f) تشکیل شده و بر اساس محتویات یک فایل یا ساختار دایرکتوری در Git محاسبه می‌شود. و شبیه به کاراکترهای پایین است و در پایگاه داده های خود همه چیز را به این شکل ذخیره می‌کند: [ 1 ]: 16

24b9da6552252987aa493b52f8696cd6d3b00373

**1-7.معرفی GitLab, GitHub, Gitkraken**

GitHub : بزرگترین میزبان برای مخازن گیت است و در واقع یک Git Hosting Service است و نقطه مرکزی همکاری میلیون ها توسعه دهنده و پروژه است. درصد زیادی از تمام مخازن گیت در GitHub میزبانی می‌شوند و بسیاری از پروژه های متن باز از آن برای میزبانی گیت، ردیابی مشکل، بررسی کد و موارد دیگر استفاده می‌کنند. بنابراین، در حالی که بخشی مستقیم از پروژه منبع باز گیت نیست، در زمانی که از گیت به طور حرفه ای استفاده می‌کنید، نیاز دارید که با GitHub تعامل داشته باشید. [ 1 ]: 166

:GitLab یک برنامه وب مبتنی بر پایگاه داده است، و یک مخزن کد متن باز و پلتفرم توسعه نرم افزار مشترک برای پروژه های بزرگ DevOps و DevSecOps است. GitLab مکانی را برای ذخیره کد آنلاین و قابلیت های ردیابی مشکل و CI/CD ارائه می دهد.

:GitKraken یک GUI یا رابط گرافیکی چند پلتفرمی قدرتمند برای گیت است و عنوان جایگزینی برای خط فرمان توسعه یافته است. به روشی بسیار ساده، می‌توانیم ردیابی کاملی از مخازن خود داشته باشیم، شاخه‌ها وtags را مشاهده کنیم، علاوه بر این می‌توانیم تمام تاریخچه کارهای خود را داشته باشیم.

[ 3 ]: Happy Developer Course

**2.شروع کار با Git**

این بخش تمام دستورات اساسی را که برای انجام اکثریت کارهایی که با گیت انجام می‌دهید، پوشش می‌دهد. در پایان، شما باید بتوانید یک مخزن را پیکربندی و initializeکنید و ردیابی فایل ها را شروع و متوقف کنید یا تغییرات را stage و commit کنید. [ 1 ]: 26

**2-1.ساختن یک Git Repository**

شما معمولاً یک مخزن گیت را به یکی از دو روش ایجاد می‌کنید:

1. می‌توانید یک دایرکتوری محلی را که در حال حاضر تحت کنترل نسخه نیست، انتخاب کنید و آن را به یک مخزن گیت تبدیل کنید، یا 2. می‌توانید یک مخزن گیت موجود را از جای دیگری cloneکنید. در هر صورت ما در نهایت با یک مخزن گیت در ماشین محلی آماده برای انجام کارهای خود هستیم. [ 1 ]: 26

**Initializing a Repository in an Existing Directory.2-1**

اگر دایرکتوری پروژه ای که دارید در حال حاضر تحت کنترل نسخه نیست و می‌خواهید آن را با گیت کنترل کنید، ابتدا باید به دایرکتوری آن پروژه بروید. اگر هرگز این کار را انجام نداده‌اید، بسته به سیستمی که اجرا می‌کنید کمی متفاوت انجام می‌شود: [ 1 ]: 26

برای Windows:

$ cd C:/Users/user/my\_project

برای macOS:

$ cd /Users/user/my\_project

و تایپ کنید:

$ git init

A screenshot of a computer

Description automatically generated with medium confidence

Figure 5. Git Init

این دستور یک subdirectory جدید به نام .git ایجاد می‌کند که حاوی تمام فایل های مخزن ضروری شما است، یک ساختار برای مخزن Git. در این مرحله، هیچ چیز در پروژه شما هنوز ردیابی نشده است. برای اطلاعات بیشتر در مورد اینکه دقیقاً چه فایل‌هایی در دایرکتوری git. ایجاد کردید، به Git Internals مراجعه کنید.

اگر می‌خواهید version-controlling فایل ها را شروع کنید (برخلاف دایرکتوری خالی) پس احتمالاً باید ردیابی آن فایل‌ها را شروع کنید و یک commit initialانجام دهید.

می‌توانید این کار را با چند دستور git add انجام دهید که فایل هایی را که می‌خواهید ردیابی کنید مشخص می‌کند و به دنبال آن یک git commit انجام می‌شود: [ 1 ]: 26

$ git add \*.c

$ git add LICENSE

$git commit -m 'Initial project version'

7

**Commit.2-2 دقیقا به چه معناست؟**

ما یادگرفتیم کهcommit به گیت، یک فرآیند دو مرحله ای است. ابتدا فایل ها را اضافه می‌کنید و سپس commit می‌کنید. اولین چیزی که باید بدانید این است که فقط آن فایل هایی که اضافه کرده‌اید commit هستند.

فرض کنید دو فایل داشتید، Checklist.md و README.md، اما فقط Checllist.md. را اضافه کرده‌اید پس گیت فقط تغییراتی را که در این فایل دارید ذخیره می‌کند. زمانی که ما commit می‌کنیم، گیت از یک الگوریتم تخصصی استفاده می‌کند تا با خیال راحت همه چیزهایی را که به حافظه آن اضافه کرده‌ایم جمع کند.

وقتی می گوییم تغییرات خود را به گیت، "commit" کردیم، ترجمه آن این است که گیت یک شی commit ایجاد می‌کند که در پوشه .git ذخیره می‌کند. این شی commit توسط یک شناسه منحصر به فرد "stamped" می‌شود. برای مثال این شناسه 513141d را دریافت می‌کنیم که در واقع یک رشته بسیار طولانی ‌تر است که حاوی اعداد و حروف است که چیزی شبیه به این است:

24b9da6552252987aa493b52f8696cd6d3b00373

اکنون یک مخزن گیت با فایل های ردیابی شده و یک commit اولیه دارید. [ 2 ]: 95

**2-2.سه مرحله اساسی**

گیت دارای سه حالت اصلی است که فایل‌های شما می‌توانند در آن‌ها قرار گیرند:

1.Modified

2.Staged

3.Committed

1. Modified به این معنی است که شما فایل را تغییر داده‌اید اما هنوز آن را در پایگاه داده خود Commit نکرده اید.
2. Staged به این معنی است که شما یک فایل اصلاح شده را در نسخه فعلی آن علامت گذاری کرده‌اید تا به عکس فوری commit بعدی بروید.
3. Commit به این معنی است که داده ها به طور ایمن در پایگاه داده محلی شما ذخیره می‌شوند.

مواردی که گفتیم ما را به سه بخش اصلی یک پروژه گیت هدایت می کند:

1.The working tree

2.The staging area

3.The Git directory

A screenshot of a computer

Description automatically generated with medium confidence

Figure 6. Workflow

working tree :Theیک single checkout از یک نسخه از پروژه است. این فایل‌ها از پایگاه داده فشرده در فهرست گیت بیرون کشیده می‌شوند و روی دیسک قرار می‌گیرند تا شما از آن استفاده کنید یا آن را تغییر دهید.

The staging area: فایلی است که عموماً در دایرکتوری گیت شما قرار دارد و اطلاعاتی را در مورد آنچه در commit بعدی شما وارد می‌شود ذخیره می‌کند. نام فنی آن در اصطلاح گیت «index» است، اما عبارت «staging area» همان معنی را نیز به خوبی می‌رساند. Git directory جایی است که گیت پایگاه داده شی را برای پروژه شما ذخیره می‌کند. این مهمترین بخش گیت است و آن چیزی است که وقتی یک مخزن را از رایانه دیگری cloneمی‌کنید کپی می‌شود.

Git workflow چیزی شبیه به این است:

1. شما فایل‌ها را در working treeخود تغییر می‌دهید.

2. شما به طور انتخابی فقط آن دسته از تغییراتی را که می‌خواهید بخشی از commit بعدی شما باشند stage می‌کنید و فقط آن تغییرات را به Git directory اضافه می‌کنید.

3. شما یک commit انجام می‌دهید که فایل‌ها را همانطور که در قسمت staging هستند انتقال دهد و آن‌ها را به‌ طور دائم در فهرست گیت شما ذخیره می‌کند. اگر نسخه خاصی از یک فایل در دایرکتوری گیت باشد، commit در نظر گرفته می‌شود. [ 1 ]: 16

**2-3.قوانین نوشتن Commit**

ابتدا دوباره مفهوم commit در گیت را مرور می‌کنیم. مخزن گیت مجموعه ای از فایل های ردیابی شده در پوشه .git یک پروژه است. به عبارت ساده، یک commit یک عکس فوری از مخزن محلی شما است. یک commit را می‌توان به عنوان یک نقطه بازرسی یا ذخیره برای پروژه در نظر گرفت. یک commit Git معمولاً پس از انجام یک مشارکت قابل توجه در پروژه انجام می‌شود و هنگامی که می‌خواهید پیشرفت خود را ذخیره کنید به کار می‌رود. هر بار که یک توسعه‌دهنده یک commit را انجام می‌دهد، این گزینه به آنها داده می‌شود که پیامی به نام commit بنویسند. از پیام ‌های Git commit برای توضیح عملکرد commit استفاده می‌شود.

برای ایجاد یک commit، یک توسعه‌دهنده باید این کارها را انجام دهد:

1.Make changes

2.Stage changes

3.Commit changes

1.ایجاد تغییرات شامل افزودن، حذف یا تغییر هر چیزی در یک شاخه محلی است. هر چیزی که در آن شاخه تغییر کرده باشد در آن چیزی که فهرست کاری نامیده می‌شود گنجانده می‌شود.

Staging changes.2 به معنای گرفتن و دریافت کردن فایل‌هایی است که تغییر داده‌اید و این فایل‌ها را به حالت "آماده برای commit" در دایرکتوری stage منتقل کنید. همچنین ممکن است برنامه‌نویسان به دایرکتوری stageبه ‌عنوان ناحیه stageیا indexاشاره کنند. شما می‌توانید هر تعداد فایلی را که می‌خواهید stageکنید، اما مهم است که توجه داشته باشید که چندین فایل با هم یک پیام commit را به اشتراک می‌گذارند. این به این معنی است که شما باید مطمئن شوید که commit‌ها به نحوی با یکدیگر مرتبط هستند. شما و همکاران تان برای مثال، می‌توانید گروهی از فایل‌ها را که همگی بر روی یک Hotfix خاصی که روی آن کار می‌کنید تمرکز کنند. برعکس، اگر هر فایل روی محدوده کاملاً متفاوتی از پروژه شما تمرکز کند، اجرای چندین فایل منطقی نخواهد بود. ایجاد یک عکس فوری از مخزن محلی شما. به این ترتیب توسعه دهندگان کار خود را ذخیره می‌کنند و یک دستور Git رایج است.

ساختار پیام های commit به شرح زیر است:

* Commit message title - limited to 72 characters
* Description- No character limit

اکثر توسعه دهندگان پیشنهاد می کنند که خلاصه پیام commit بیشتر از 50 کاراکتر نباشد و توضیحات به 72 کاراکتر محدود شود.

* تمرکز commit روی یک موضوع باشد: تمرکز commitها روی یک موضوع باعث می‌شود که تصمیم بگیرید چه فایل هایی را stageو commit کنید و خواندن تاریخچه commit را برای دیگران آسان تر می‌کند.
* از نوشتن غیر ضروری حروف بزرگ اجتناب کنید.
* املای خود را بررسی کنید و پیام commit را با نقطه تمام نکنید.
* Commit ها باید به راحتی قابل درک باشند، نه گیج کننده یا بی نظم.

**2-3.نمونه ای از پیام های commit خوب**

* Updated the left navigation with new 12.2 feature updates
* Bumped nodegit version up to 6.4.2

هر دو commitبالا هر کدام نشان دهنده یک موضوع که تغییر کرده‌اند هستند. واضح است که چه چیزی تغییر کرده است، و درک دامنه تغییر آسان است.

**2-3.نمونه ای از پیام های commit بد**

* made fixes
* Applied patch.
* Updated the left nav w/ new 12.2 updates & bumped nodegit to v6.4.2

[ 4 ]: GitKraken Course, Commits

**Undoing Things.2-4**

در هر مرحله، ممکن است بخواهید کاری را لغو کنید، در این بخش چند ابزار اساسی برای لغو تغییراتی که ایجاد کرده‌اید را بررسی می‌کنیم. باید با احتیاط از این دستور استفاده کنید زیرا همیشه نمی‌توانید برخی از undoها را لغو کنید. یکی از لغو کردن های رایج زمانی اتفاق می‌افتد که خیلی زود نهایی سازی را انجام می‌دهید و فراموش می‌کنید که چند فایل را اضافه کنید، یا پیام commit خود را اشتباه می‌نویسید.

اگر می‌خواهید commit را دوباره انجام دهید، تغییرات اضافی را که فراموش کرده اید اضافه کنید، آنها راstage کنید و با استفاده از گزینه --amend دوباره commit کنید.

$ git commit --amend

این دستور staging areaرا بررسی می‌کند و از آن برای commit استفاده می‌کند. اگر از آخرین commit خود هیچ تغییری ایجاد نکرده‌اید (به عنوان مثال، این دستور را بلافاصله پس از commit قبلی خود اجرا می کنید)، عکس فوری دقیقاً یکسان خواهد بود و تنها چیزی که تغییر می‌دهید پیام commit است.

در واقع همان ویرایشگر commit-message فعال می‌شود، اما حاوی پیام commit قبلی شما است.

می‌توانید پیام را مانند همیشه ویرایش کنید، اما commit قبلی شما را بازنویسی می‌کند. به عنوان مثال، اگر commit کنید و سپس متوجه شوید که فراموش کرده‌اید تغییرات را در فایلی که می‌خواهید به این commit اضافه کنید، میتوانید stage کنید و کاری شبیه به این انجام دهید.

$ git commit -m 'Initial commit'

$ git add forgotten\_file

$ git commit --amend

در نهایت با یک commit منفرد کار تمام می‌شود و commit دوم جایگزین نتایج اولی می‌شود. [ 1 ]: 46

**Local.2-5 چیست؟**

در ابتدا به چند مورد از ویژگی های گیت اشاره کردیم، اما یکی دیگر از مهم ترین قابلیت های گیت این است که تقریباً هر عملیاتی در گیت Localاست. اکثر عملیات ها در گیت برای کار کردن فقط به فایل‌ها و منابع محلی نیاز دارند. معمولاً هیچ اطلاعاتی از رایانه دیگری لازم نیست. از آنجایی که شما کل تاریخچه پروژه را دقیقاً در دیسک محلی خود دارید، اکثر عملیات ها تقریباً آنی به نظر می‌رسند. به عنوان مثال، برای مرور تاریخچه پروژه گیت، نیازی به رفتن به سرور، برای دریافت تاریخچه و نمایش آن نیست و تاریخچه را به سادگی از پایگاه داده محلی شما می‌خواند. این بدان معنی است که شما تاریخچه پروژه را تقریباً بلافاصله می‌بینید. اگر می‌خواهید تغییرات ایجاد شده بین نسخه فعلی یک فایل و فایل یک ماه پیش را ببینید، گیت می‌تواند فایل یک ماه پیش را جستجو کند و تغییراتی که در Local وجود دارد را محاسبه کند، به جای اینکه مجبور باشد از یک سرور remoteدرخواست کند یا نسخه قدیمی فایل را از سرور دریافت کند، این کار را به صورت محلی انجام می‌دهد. همچنین به این معنی است که اگر آفلاین هستید، نمی‌توانید کار نسبتا زیادی با گیت انجام دهید. اگر سوار هواپیما یا قطار می‌شوید و می‌خواهید کمی کار کنید، می‌توانید تا زمانی که به یک اتصال شبکه برای uploadبرسید، در ماشین محلی خود تغییرات را commit کنید. این قابلیت در دیگر کنترل نسخه ها به این شکل وجود ندارد. برای مثال در Perforce، وقتی به سرور متصل نیستید نمی‌توانید کار زیادی انجام دهید. در Subversion و CVS، می‌توانید فایل‌ها را ویرایش کنید، اما نمی‌توانید تغییراتی را در پایگاه داده خود اعمال کنید (زیرا پایگاه داده شما آفلاین است). ممکن است این موضوع زیاد مهم به نظر نرسد اما ممکن است تعجب کنید که چه تفاوت بزرگی می تواند در نهایت ایجاد کند. [ 1 ]: 10

**Remote.2-6 چیست؟**

برای اینکه بتوانید در هر پروژه ای همکاری کنید، باید بدانید که چگونه مخازن remote را مدیریت کنید. Remoteها نسخه‌ هایی از پروژه شما هستند که در اینترنت یا شبکه، میزبانی می‌شوند. شما می‌توانید چندین مخزن از آنها را داشته باشید، که هر کدام به طور کلی یا فقط خواندنی هستند یا برای خواندن/ نوشتن. همکاری با دیگران شامل مدیریت این مخازن راه دور و pull و push کردن داده‌ها است و در هنگام نیاز به اشتراک گذاری پروژه خود است. مدیریت remote ها شامل دانستن نحوه افزودن مخازن remote، حذف remoteهایی که دیگر معتبر نیستند، مدیریت شاخه های مختلف راه دور و تعریف آن ها به عنوان ردیابی یا عدم ردیابی و موارد دیگر است. در این بخش، برخی از این مهارت‌ های مدیریت از راه دور را پوشش می‌دهیم. به طور خلاصه یک remote را فقط یک کپی دیگر از تاریخچه commit خود در نظر بگیرید، به علاوه این نکته که هر تغییری که در نسخه محلی خود ایجاد می‌کنید در نهایت باید با remoteهمگام شود. دستور push به شما این امکان را می‌دهد که این کار را انجام دهید. [ 1 ]: 50

**Pushing to Your Remotes.2-6**

وقتی پروژه خود را در نقطه‌ای دارید که می‌خواهید به اشتراک بگذارید، باید آن را به سمت remote هدایت کنید. دستور این کار ساده است: git push <remote> <branch>. اگر می‌خواهید شاخه اصلی خود را به سرور اصلی خود منتقل کنید، می‌توانید این را اجرا کنید تا هر commitرا که انجام داده‌اید به سرور بازگردانید: [ 1 ]: 53

$ git push origin master

**Git push.2-7**

git push به شما این امکان را می‌دهد که هر commit جدیدی را که در مخزن محلی ایجاد کرده‌اید را به ریموت منتقل کنید. به عبارت دیگر، دستور git push به شما این امکان را می‌دهد که تغییرات محلی خود را با ریموت همگام سازی کنید.

در پایان روز، شما می‌خواهید کار را به اتمام برسانید، برخی commit ها را در یک شاخه انجام دهید و تغییرات خود را به ریموت منتقل کنید. به این ترتیب، اگر شخصی مخزن را cloneکند، همه چیز، از جمله commitهایی را که شما انجام داده‌اید و به ریموت منتقل کرده‌اید، شبیه سازی می‌کند. اما چطور باید این کار را انجام دهیم؟ قبلاً در مورد دستور git push صحبت کرده‌ایم، فرض کنید شما به تازگی یک commit را روی شاخه اصلی انجام داده‌اید و می‌خواهید آن را به ریموت انتقال دهید. شما از دستور git push استفاده می‌کنید. با این حال، از آنجایی که GitHub باید مطمئن شود که شما دسترسی مناسبی برای نوشتن در مخزن دارید، از شما می‌خواهد که loginکنید.

GitHub در اینجا نام کاربری و رمز عبور را درخواست می‌کند. حتما از نام کاربری خود در اینجا استفاده کنید. ممکن است کمی گیج کننده به نظر برسد زیرا اگرچه "رمز عبور" خواسته شده، اما آنچه واقعاً درخواست می‌کند "password access token" است. اگر به نحوی توکن خود را به اشتباه فراموش کرده‌اید، همیشه می‌توانید با استفاده از پنل «Tokens Access Personal» در پروفایل خود در GitHub، یکی دیگر را ایجاد کنید.

A picture containing text, screenshot, menu, font

Description automatically generated

Figure 7. Git Push

همچنین، وقتی توکن خود را وارد می‌کنید، در واقع آنچه را که تایپ می‌کنید، (به دلایل امنیتی) نمی‌بینید. سپس گیت یکسری کارها را برای ارسال commitهای شما به ریموت انجام می‌دهد و اگر همه چیز خوب پیش برود، هیچ خطایی نخواهید دید و عملیات با موفقیت انجام شده است. [ 2 ]: 486

**Cloning.2-8 یک مخزن موجود**

اگر می‌خواهید یک کپی از مخزن Git موجود دریافت کنید برای مثال، پروژه‌ای که می‌خواهید در آن مشارکت کنید، دستور مورد نیاز git clone است. اگر با VCS های دیگری مانند Subversion آشنا هستید، متوجه خواهید شد که دستور "clone" درست است و نه "checkout". یک کپی کامل از تقریباً تمام داده‌ های سرور دریافت می‌کند. هنگامی که git clone را اجرا می‌کنید، هر نسخه از هر فایل برای تاریخچه پروژه به طور پیش فر pull می‌شود. در واقع، اگر دیسک سرور شما دچار مشکل شود، اغلب می‌توانید از تقریباً هر یک از نسخه های clone در هر client استفاده کنید تا سرور را به حالتی که در زمان cloneبود برگردانید. [ 1 ]: 27

$ git clone https://github.com/libgit2/libgit2

**3.مقدمات Branching**

**Git Branching.3-1**

تقریباً هر VCS دارای نوعی پشتیبانی از branchاست. انشعاب به این معنی است که شما از خط اصلی توسعه دور می‌شوید و بدون به هم ریختن آن خط اصلی به انجام کار خود ادامه می‌دهید. در بسیاری از ابزارهای VCS، این یک فرآیند تا حدودی گران است، که اغلب شما را ملزم به ایجاد یک کپی جدید از فهرست کد منبع خود می‌کند، که ممکن است برای پروژه های بزرگ زمان زیادی طول بکشد.

برخی از افراد به مدل شاخه‌بندی گیت به عنوان killer feature اشاره می‌کنند و مطمئناً گیت را در جامعه VCS متمایز می‌کند. چرا گیت اینقدر خاص است؟ شیوه انشعاب‌ گیت فوق‌العاده سبک است، عملیات انشعاب را تقریباً آنی می‌سازد، و معمولاً به همان سرعت بین شاخه‌ها جا به‌ جا می‌شود. برخلاف نسخه های دیگر از VCSها، گیت workflows را تشویق می‌کند که اغلب حتی چندین بار در روز branch و mergeشوند.

درک و تسلط بر این ویژگی، یک ابزار قدرتمند و منحصر به فرد را در اختیار شما قرار می‌دهد و می‌تواند روش توسعه شما را کاملاً تغییر دهد. [ 1 ]: 63

**Branch.3-1 دقیقا چیست؟**

یک شاخه به زبان ساده اشاره‌ای به یک commit است. پس چه چیزی یک شاخه را به یک شاخه تبدیل می‌کند؟ یک commit یک عکس فوری از محتوایی است که stage کرده‌اید (یعنی فایل‌هایی که به فهرست اضافه کرده‌اید).

اگر روی یک task کار می‌کنید که در آن دو یا چند commitانجام داده‌اید، پس این commitها به هم متصل می‌شوند. یعنی هر commit بعدی شناسه commitرا که درست قبل از آن آمده را ثبت می‌کند.

A picture containing handwriting, text, font, white

Description automatically generated

Figure 8. Branch form

تصور کنید برای هر شاخه در مخزن خود یک یادداشت جداگانه دارید. هر یادداشت دارای نام شعبه و شناسه آخرین commit در آن شاخه است. وقتی روی شاخه ای commit می‌کنید، گیت ابتدا commit را ایجاد می‌کند. سپس sticky note را که نشان دهنده آن شاخه است، می‌گیرد، شناسه commit را که روی آن بود پاک می‌کند، و به مورد جدید اضافه می‌کند: یک شاخه همیشه به آخرین commit در آن شاخه اشاره می‌کند، و هر commit به نوبه خود به دیگری اشاره می‌کند. commit ( به parent commit) و یک شاخه به سادگی یک مرجع به یک commit از طریق ID آن است. این مرجع هر بار که commit دیگری را در آن شاخه انجام می‌دهید به روز می‌شود.

[ 2 ]: 180

A picture containing text, font, handwriting, diagram

Description automatically generated

Figure 9. Commit ID on Branch

**Branching 3.1 به طور خلاصه**

هنگامی که commit را با اجرای git commit ایجاد می‌کنید، گیت هر زیر شاخه (در این مورد، فقطroot project directory) را چک می‌کند و آنها را به عنوان یک شی درختی در مخزن گیت ذخیره می‌کند. سپس گیت یک شی commit ایجاد می کند که دارای ابر داده [[3]](#footnote-3) و یک اشاره‌گر است تا بتواند در صورت نیاز آن عکس فوری را دوباره ایجاد کند.

A picture containing text, screenshot, font

Description automatically generated

مخزن گیت شما اکنون شامل پنج شی است: سه حباب [[4]](#footnote-4)(هر کدام نشان دهنده محتویات یکی از سه فایل)، یک درخت که محتویات دایرکتوری را فهرست می‌کند و مشخص می‌کند نام فایل به عنوان کدام حباب ذخیره می‌شود، و یکی با اشاره گر به آن commit می‌کند. اگر تغییراتی ایجاد کنید و دوباره commit کنید commit بعدی یک اشاره گر به commit که بلافاصله قبل از آن آمده است ذخیره می‌کند.

A picture containing text, screenshot, font, rectangle

Description automatically generated

Figure 10. A commit and its tree

اگر تغییراتی ایجاد کنید و دوباره commit کنید، commit بعدی یک اشاره گر به commit، که بلافاصله قبل از آن آمده است را ذخیره می‌کند.

A screenshot of a computer

Description automatically generated with low confidence

Figure 11. Commits and their parents

یک شاخه در گیت به سادگی یک اشاره گر متحرک برای یکی از این commitها است. نام شعبه پیش فرض در گیت master است. همانطور که شروع به commit می‌کنید، یک شاخه اصلی ایجاد می‌شود که به آخرین commit که انجام داده‌اید اشاره می‌کند. هر بار که commit می‌کنید، نشانگر شاخه اصلی به طور خودکار رو به جلو حرکت می‌کند.

شاخه "Master" در گیت یک شاخه خاص نیست. دقیقاً مثل هر شاخه دیگری است.

تنها دلیلی که تقریباً هر مخزن این شاخه را دارد این است که دستور git init، آن را به ‌طور پیش‌فرض ایجاد می‌کند. [ 1 ]: 63

A picture containing text, screenshot, font, rectangle

Description automatically generated

Figure 12. A branch and its commit history

**3-2.ساختن یک Branch جدید**

وقتی یک شاخه جدید ایجاد می‌کنید چه اتفاقی می‌افتد؟ انجام این کار یک اشاره گر جدید برای حرکت در آن مسیر برای شما ایجاد می‌کند. فرض کنید می‌خواهید یک شاخه جدید به نام تست ایجاد کنید. این کار را با دستور git branch انجام می‌دهید:

$ git checkout testing

این دستور یک نشانگر جدید به همان commit که در حال حاضر در آن هستید ایجاد می‌کند.

A picture containing text, font, screenshot, line

Description automatically generated

Figure 13. Two branches pointing into the same series of commits

گیت چگونه می‌داند که در حال حاضر در چه شاخه ای هستید؟ به کمک یک اشاره گر ویژه به نام HEAD این کار را انجام می‌دهد. توجه داشته باشید که این بسیار متفاوت از مفهوم HEAD در سایر VCSهایی است که ممکن است با آنها کار کرده باشید، مانند Subversion یا CVS. در گیت، در واقع یک اشاره گر به شاخه محلی است که در حال حاضر در آن هستید. در این مورد، شما هنوز در masterهستید. دستور git branch فقط یک شاخه جدید ایجاد کرده است و چیزی به آن شاخه منتقل نشد. [ 1 ]: 65

**Switching Branches.3-3**

برای جابجایی به یک شاخه موجود، دستور git checkout را اجرا می‌کنید. در مثال زیر می‌خواهیم به شاخه آزمایشی جدید منتقل شویم:

$ git branch testing

این دستور HEAD را حرکت می‌دهد تا به شاخه testing اشاره کند.

A picture containing text, screenshot, font, rectangle

Description automatically generated

Figure 14. HEAD pointing to a branch

اهمیت این موضوع چیست؟ برای روشن شدن موضوع یک commitدیگر انجام می‌دهیم.

$ vim test.rb

$ git commit -a -m 'made a change

A picture containing text, screenshot, rectangle, line

Description automatically generated

Figure 15. The HEAD branch moves forward when a commit is made

اکنون شاخه testingبه جلو حرکت کرده است، اما شاخه اصلی هنوز به آن commitاشاره می‌کند که هنگام اجرای git checkout برای تغییر شاخه ها انجام می‌دادید. با این دستور به شاخه اصلی برمی‌گردیم.

$ git checkout master

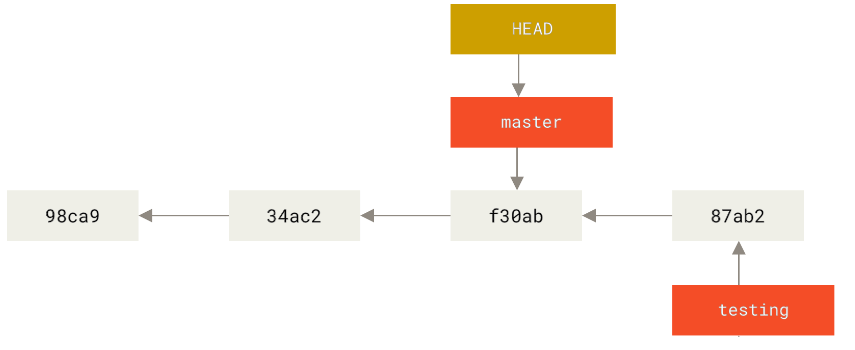


Figure 16. HEAD moves when you checkout

دستور بالا دو کار را انجام داد. 1.نشانگر HEAD را به سمت شاخه اصلی حرکت داد و2 .فایل‌های موجود در فهرست کاری شما را به عکس فوری که masterبه آن اشاره می‌کند برگرداند. این همچنین به این معنی است که تغییراتی که از این نقطه به بعد ایجاد می‌کنید از نسخه قدیمی‌تر پروژه متفاوت است. اساساً کاری را که در شاخه آزمایشی خود انجام داده‌اید به عقب برمی‌گرداند تا بتوانید به مسیر دیگری بروید.

نکته: Switching branches changes files in your working directory

توجه به این نکته مهم است که وقتی شاخه‌ها را در گیت تغییر می‌دهید، فایل‌های موجود در فهرست کاری شما تغییر می‌کنند. اگر به یک شاخه قدیمی‌تر بروید، working directoryشما برگردانده می‌شود. اگر گیت نتواند این کار را به طور تمیز انجام دهد، به هیچ وجه به شما اجازه switchنمی‌دهد. [ 1 ]: 66

**Basic Branching and Merging.3-4**

ابتدا یک مثال ساده از انشعاب و ادغام با یک workflow را که ممکن است در دنیای واقعی استفاده کنید، مرور کنیم. [ 1 ]: 70

این مراحل را دنبال خواهیم کرد:

1.Do some work on a website.

2.Create a branch for a new user story you’re working on

3.Do some work in that branch

در این مرحله، پیامی دریافت می‌کنید مبنی بر اینکه مشکل دیگری وجود دارد و نیاز به رفع فوری مشکل دارید. شما موارد زیر را انجام خواهید داد:

1.Switch to your production branch.  
2. Create a branch to add the hotfix.  
3. After it’s tested, merge the hotfix branch, and push to production.  
4.Switch back to your original user story and continue working.

**Basic Branching3-4**

ابتدا، فرض کنید که در حال کار بر روی پروژه خود هستید و چند commit از قبل در شاخه master دارید**:**

A picture containing text, screenshot, font, design

Description automatically generated

Figure 17. A simple commit history

شما تصمیم گرفته‌اید که روی issue #53 کار کنید. برای ایجاد یک شاخه جدید و switch کردن همزمان به آن می‌توانید دستور git checkout را با switch -b اجرا کنید.

$ git checkout -b iss53

Switched to a new branch "iss53"

و دستور بالا کوتاه شده این دستور است:

$ git branch iss53

$ git checkout iss53

A picture containing text, screenshot, font, design

Description automatically generated

Figure 18. Creating a new branch pointer

شما روی وب سایت خود کار می‌کنید و commitهایی را انجام می‌دهید. با انجام این کار iss53 به جلو حرکت می‌کند زیرا شما آن را checkout کرده‌اید؛ که HEAD به آن اشاره می‌کند:[ 1 ]: 70

$ vim index.html

$ git commit -a -m 'Create new footer [issue 53]' 71

**Basic Merging.3-4**

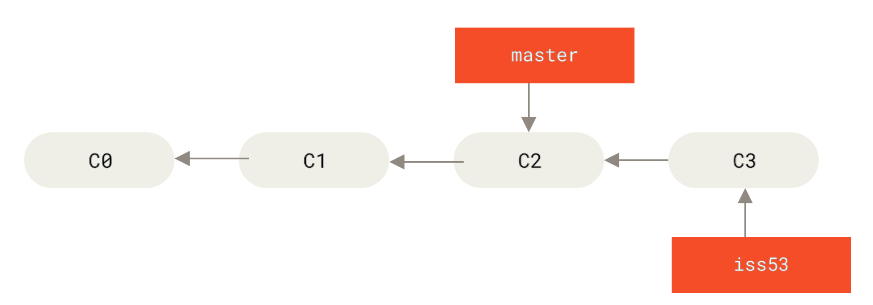


Figure 19. The iss53 branch has moved forward with your work

فرض کنید که کار 53# کامل شده و آماده ادغام در شاخه اصلی شما است. برای انجام این کار، شعبه issue #53 خود را به master ادغام خواهید کرد، دقیقاً مانند آنچه قبلاً شاخه hotfix خود را ادغام کرده اید. تنها کاری که باید انجام دهید این است که شاخه ای را که می خواهید در آن ادغام کنید بررسی کنید و سپس دستور git merge را اجرا کنید. این دستور به نظر کمی متفاوت از ادغام Hotfix است که قبلاً انجام دادید. در این مورد، سابقه توسعه شما از نقطه‌ای قدیمی‌تر فاصله گرفته است. از آنجایی که commit در شاخه ای که در آن هستید، جد مستقیم شاخه ای نیست که در آن ادغام می‌شود، گیت باید کارهایی را انجام دهد.

در این مورد، گیت یک ادغام سه ‌طرفه ساده را با استفاده از دو عکس فوری که توسط نوک شاخه‌ها و جد مشترک این دو نشان داده شده است، انجام می‌دهد:

$ git checkout master

Switched to branch 'master'

$ git merge iss53

Merge made by the 'recursive' strategy.

index.html | 1 +

1 file changed, 1 insertion(+)

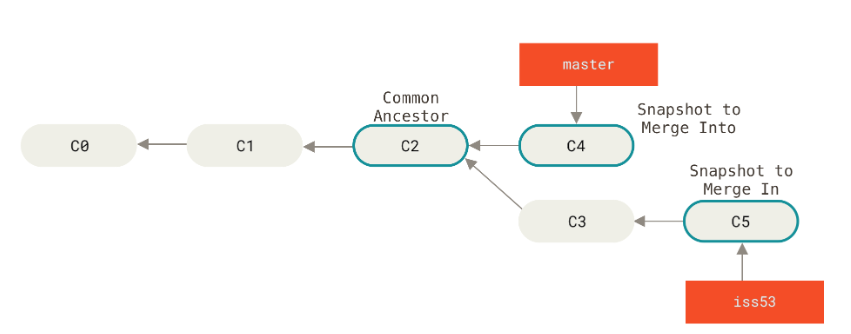


Figure20. Three snapshots used in a typical merge

به جای اینکه فقط نشانگر، شاخه را به جلو حرکت دهد، Git یک عکس فوری جدید ایجاد می‌کند که از ادغام سه طرفه حاصل می‌شود و به طور خودکار یک commit جدید ایجاد می‌کند که به آن اشاره می‌کند. و به عنوان یک merge commit نامیده می‌شود، و از این جهت که بیش از یک والد دارد، خاص است.

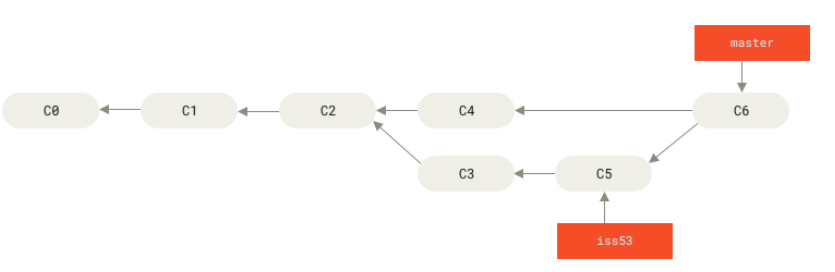


Figure21. A merge commit

اکنون که کار شما ادغام شده است، دیگر نیازی به شاخه issue #53 ندارید. می‌توانید شعبه را حذف کنید:

$ git branch -d iss53

**Basic Merge Conflicts.3-4**

گاهی اوقات، این روند به این شکل پیش نمی‌رود. اگر قسمت مشابهی از یک فایل را در دو شاخه‌ای که در حال ادغام آن هستید به طور متفاوت از یکدیگر تغییر دهید، گیت نمی‌تواند آنها را به راحتی ادغام کند. اگر راه حل شما برای issue #53 همان بخش از فایل را به عنوان شاخه Hotfix تغییر داد، یک merge conflict دریافت خواهید کرد:

$ git merge iss53

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result

گیت به طور خودکار یک merge commit جدید ایجاد نکرده است. بلکه درحالی که شما مشغول حل کردن conflictها هستید، روند را متوقف کرده است. برای اطلاع از اینکه کدام فایل‌ها در هر نقطه‌ای پس از merge به صورت unmerged باقی می‌مانند، می‌توانید status git را اجرا کنید.

$ git status  
On branch master  
You have unmerged paths.  
(fix conflicts and run "git commit")  
Unmerged paths:  
(use "git add <file>..." to mark resolution)  
both modified: index.html  
no changes added to commit (use "git add" and/or "git commit -a")

هر چیزی که دارای merge conflictsباشد و هنوز حل نشده باشد، گیت آن را به عنوان unmerged فهرست می‌کند. گیت نشانگرهای استاندارد conflict-resolutionرا به فایل هایی که دارای تداخل هستند اضافه می‌کند، بنابراین می‌توانید آنها را به صورت دستی باز کرده و تداخل‌ها را حل کنید. فایل شما حاوی بخشی است که شبیه ساختار زیر به نظر می‌رسد:

<<<<<<< HEAD:index.html  
<div id="footer">contact : email.support@github.com</div>  
=======  
<div id="footer">  
please contact us at support@github.com  
</div>  
>>>>>>> iss53:index.html

این بدان معناست که نسخه موجود در HEAD (شاخه اصلی شما، زیرا این همان چیزی بود که هنگام اجرای دستور ادغام خود بررسی کرده بودید) قسمت بالای آن بلوک است. برای حل تعارض، باید یا یک طرف یا طرف دیگر را انتخاب کنید یا خودتان آن را ادغام کنید. به عنوان مثال، ممکن است این تضاد را با جایگزین کردن کل بلوک با این کد حل کنید:

<div id="footer">

please contact us at [email.support@github.com](mailto:email.support@github.com)

</div>

پس از خروج از ابزار ادغام، Git از شما می‌پرسد که آیا ادغام موفقیت آمیز بوده است یا خیر. اگر به اسکریپت بگویید که بوده است، فایل را stageمی‌کند تا آن را به عنوان resolvedبرای شما علامت گذاری کند. می‌توانید وضعیت git را دوباره اجرا کنید تا بررسی کنید که تمام تداخل‌ها برطرف شده‌اند:

$ git status  
On branch master  
All conflicts fixed but you are still merging.  
(use "git commit" to conclude merge)  
Changes to be committed:

اگر از آن راضی هستید و تأیید می‌کنید، می‌توانید git commit را تایپ کنید تا commit ادغام نهایی شود. پیام commit به طور پیش فرض چیزی شبیه به این است: [ 1 ]: 76

Merge branch 'iss53'  
Conflicts:  
78

index.html  
# It looks like you may be committing a merge.  
# If this is not correct, please remove the file  
# .git/MERGE\_HEAD  
# and try again.  
# Please enter the commit message for your changes. Lines starting  
# with '#' will be ignored, and an empty message aborts the commit.  
# On branch master  
# All conflicts fixed but you are still merging.  
# Changes to be committed:  
# modified: index.html

**Changing a branch name.3-5**

فرض کنید شاخه‌ای دارید که bad-branch-name است و می‌خواهید نام آن را به این شاخه تغییر دهید corrected-branch-name در حالی که کل تاریخچه را حفظ می‌کنید. شما همچنین می‌خواهید نام شعبه را در remote تغییر دهید (GitHub، GitLab، سرورهای دیگر). چگونه این کار را انجام می‌دهید؟ با دستور git -move شاخه را به صورت محلی تغییر نام دهید:

$ git branch --move bad-branch-name corrected-branch-name

این تغییر نام، شعبه شما را با نام شاخه اصلاح شده جایگزین می‌کند، اما این تغییر در حال حاضر فقط محلی است. برای اینکه دیگران شاخه را ببینند، این دستور را بنویسید:

$ git push --set-upstream origin corrected-branch-name

اکنون نگاهی گذرا خواهیم داشت به جایی که اکنون هستیم:

$ git branch --all  
\* corrected-branch-name  
main  
remotes/origin/bad-branch-name  
remotes/origin/corrected-branch-name  
remotes/origin/main

توجه داشته باشید که در شعبه اصلاح شده-نام-شاخه هستید و در کنترل از راه دور در دسترس است. با این حال شاخه ای با نام بد نیز هنوز در آنجا وجود دارد، اما می‌توانید با اجرای دستور زیر آن را حذف کنید:

$ git push origin --delete bad-branch-name

اکنون نام شعبه به طور کامل با نام شعبه اصلاح شده جایگزین شده است. [ 1 ]: 80

**Rebasing.3-6**

در گیت دو راه اصلی برای یکی کردن تغییرات از یک شاخه به شاخه دیگر وجود دارد: merge و rebase. در این بخش درمورد rebasing یاد خواهیم گرفت. اینکه چیست، چگونه انجام می‌شود، چرا یک ابزار بسیار شگفت انگیز است و در چه مواردی لازم به استفاده از آن نیست. [ 1 ]: 95

**The Basic Rebase.3-6**

اگر به مثال قبلی از Basic Merging مراجعه کنید، می‌بینید که کار خود را منشعب کرده‌اید و در دو شاخه مختلف commit انجام داده‌اید.

A diagram of a flowchart

Description automatically generated with low confidence

Figure 22. Simple divergent history

ساده ترین راه برای ادغام شاخه ها، همانطور که قبلاً توضیح دادیم، دستور ادغام است. و یک ادغام سه طرفه بین دو آخرین snapshot شاخه (C3 و C4) و جدیدترین جد مشترک این دو (C2) انجام می‌دهد و یک snapshotو commit جدید ایجاد می‌کند.

A diagram of a flowchart

Description automatically generated with low confidence

Figure 23. Merging to integrate diverged work history

با این حال، راه دیگری وجود دارد: می‌توانیدpatch تغییری را که در C4 معرفی شده است بردارید و دوباره آن را در بالای C3 اعمال کنید. در گیت به این کار rebasing می‌گویند. با دستور rebase می‌توانید تمام تغییراتی را که روی یک شاخه انجام شده است بگیرید و در شاخه دیگری دوباره reply کنید. برای این مثال، شما باید شاخه experiment را بررسی کنید و سپس آن را به شرح زیر به شاخه master تغییر دهید:

$ git checkout experiment  
$ git rebase master  
First, rewinding head to replay your work on top of it...  
Applying: added staged command

این عملیات با رفتن به جد مشترک دو شاخه (شاخه‌ای که روی آن هستید و شاخه‌ای که روی آن مجدداً rebasing می‌کنید) و گرفتن تفاوت معرفی ‌شده توسط هر commit شاخه‌ای که روی آن هستید، انجام می‌شود و آن تفاوت‌ها را به صورت فایل‌های موقت ذخیره می‌کند. بازنشانی شاخه فعلی به همان commit شاخه‌ای که در حال تغییر آن هستید و در نهایت هر تغییر را به نوبه خود اعمال کنید.

A picture containing jack, design

Description automatically generated with medium confidence

Figure 24. Rebasing the change introduced in C4 onto C3

در این مرحله، می‌توانید به شاخه اصلی برگردید و یک merge سریع انجام دهید.

$ git checkout master

$ git merge experiment

A picture containing jack

Description automatically generated with medium confidence

Figure 25. Fast-forwarding the master branch

عکس فوری که توسط C4 به آن اشاره شده است دقیقاً همان عکسی است که توسط C5 در مثال ادغام به آن اشاره شده است. هیچ تفاوتی در محصول نهایی ادغام وجود ندارد، اما rebasing تاریخچه تمیزتری را ایجاد می‌کند. اگر گزارش یک شاخه rebased را بررسی کنید، مانند یک تاریخچه خطی به نظر می‌رسد. به نظر می‌رسد که همه کارها به صورت متوالی اتفاق افتاده است، حتی زمانی که در ابتدا به صورت موازی اتفاق افتاده و اغلب، این کار را انجام می‌دهید تا مطمئن شوید که commitبه طور تمیز در یک شاخه remote اعمال می‌شود.

در این حالت، کار خود را در یک شاخه انجام می‌دهید و زمانی که آماده ارسال commit خود به پروژه اصلی بودید، کار خود را به origin/master تغییر می‌دهید. به این ترتیب، maintainer نیازی به انجام یکپارچه سازی ندارد. توجه کنید که عکس فوری که توسط commit نهایی به آن اشاره می‌شود، خواه آخرین rebased commit برای یک rebase یا آخرین mergeپس از mergeباشد، همان عکس فوری است فقط تاریخچه متفاوت است. Rebasing مجدد از یک خط کاری، به خط دیگر به ترتیبی که معرفی شده است تغییر می‌کند، در حالی که ادغام نقاط پایانی را گرفته و آنها را با هم ادغام می‌کند. [ 1 ]: 95

**Contributing to a Project.4**

**Pulling.4-1**

دستوری به نام git pull وجود دارد که اساساً یک واکشی[[5]](#footnote-5) گیت است که بلافاصله پس از آن یک ادغام گیت در اکثر موارد وجود دارد. اگر شعبه ردیابی را همانطور که در بخش آخر نشان داده شد راه اندازی کرده‌اید، با تنظیم صریح آن یا با ایجاد آن توسط clone یا دستورات دیگر، git pull به دنبال سرور و شاخه‌ای است که شاخه فعلی شما ردیابی می‌کند و از آن سرور تمام تغییرات را fetch می‌کند. سپس در شعبه ریموت این کار را تکرار کنید. اکنون که حساب ما راه‌اندازی شده است، بیایید جزئیاتی را بررسی کنیم که می‌تواند برای کمک به شما در مشارکت در یک پروژه موجود مفید باشد. [ 1 ]: 94

**Forking Projects.4-2**

اگر قصد این را دارید که در پروژه‌ای موجود که به آن دسترسی ندارید، مشارکت کنید، می‌توانید پروژه را «fork» کنید. هنگامی که پروژه ای را "fork" می‌کنید، GitHub یک کپی از پروژه ایجاد می‌کند که کاملاً متعلق به شماست. به نام شما ایجاد می‌شود، و شما می‌توانید تغییرات را pull و pushکنید.

افراد می‌توانند پروژه‌ای را fork کنند و تغییرات خود را با ایجاد آنچه که pull request نامیده می‌شود به مخزن اصلی بازگردانند، که در ادامه به آن خواهیم پرداخت. این کار یک صفحه با بررسی کد ایجاد می‌کند و مالک و مشارکت‌کننده می‌توانند در مورد تغییرات پروژه صحبت کنند تا زمانی که مالک از آن راضی باشد، در این مرحله مالک می‌تواند تغییرات را ادغام کند. برای انجام پروژه، از صفحه پروژه دیدن کنید. و روی دکمه "Fork" در سمت راست بالای صفحه کلیک کنید. پس از چند ثانیه به صفحه پروژه جدید خود هدایت خواهید شد. [ 1 ]: 171

**Creating a Pull Request.4-3**

به صورت خلاصه، pull request یعنی رویدادی که در آن یک مشارکت‌کننده از یک نگهدارنده مخزن می‌خواهد تا کدی را که می‌خواهد در یک پروژه mergeکند، بررسی کند.

همچنین به عنوانmerge request شناخته می‌شود. اولین گزینه برای ایجاد درخواست آسان است کافی است URL را در پنجره مرورگر خود تایپ کنید:

A picture containing text, receipt, font, line

Description automatically generated

Figure 26. Github URL

اما چه کسی می‌خواهد URL های طولانی را تایپ کند؟ از طرف دیگر، اگر بلافاصله پس از ارسال تغییرات یک شاخه به ریموت، از مخزن GitHub خود بازدید کنید، خواهید دید که GitHub یک بنر خوب برای شروع به شما ارائه می‌دهد، بدون نیاز به تایپ. این چیزی است که به نظر می رسد:

A screenshot of a computer

Description automatically generated with medium confidence

Figure 27. Create a pull request

این بنر گذرا است، بنابراین اگر بلافاصله پس از فشار دادن به صفحه مخزن GitHub خود مراجعه کنید ممکن است آن را نبینید. بنابراین، اجازه دهید راه دیگری برای ایجاد درخواست‌ها بررسی کنیم. اگر از صفحه مخزن GitHub خود بازدید می‌کنید، باید یک برگه «pull request» را در بالا ببینید:

A screenshot of a computer

Description automatically generated with medium confidence

Figure 28. Making a pull request

A screenshot of a computer

Description automatically generated with medium confidence

Figure 29. Pull Request page

با کلیک بر روی دکمه "ایجاد pull request" در نهایت شما به صفحه ای منتقل می‌شوید که به شما امکان می‌دهد که انتخاب کنید کدام شاخه ها را می‌خواهید ادغام کنید: شاخه (منبع) و شاخه (هدف) کدام باشد.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 30. Comparing changes

صفحه "مقایسه تغییرات" به شما امکان می‌دهد شاخه هایی را که می‌خواهید ادغام کنید انتخاب کنید. در GitHub به شاخه اصلی masterمی‌گویند: یعنی شاخه ای که می‌خواهید در آن ادغام را انجام دهید. در تصویر بالا، ما master را به عنوان base و feat-b را به عنوان منبع انتخاب کردیم. با کلیک بر روی دکمه "ایجاد درخواست" به صفحه ای منتقل می‌شوید که به شما امکان می‌دهد اطلاعاتی درباره درخواست جدید خود اضافه کنید.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 31. Pull Request Information

درخواست با موفقیت ارسال شد. [ 2 ]: 523

جمع‌بندی:

در این گزارش، هر آنچه که برای شروع کار با گیت نیاز دارید شامل نحوه نوشتن دستورات گیت و همکاری با دیگر افراد پروژه را به طور خلاصه و کاربردی بیان کردیم و به صورت تصویری با تمام مراحل آشنا شدیم. همچنین برای یادگیری موارد پیشرفته‌تر می‌توانید به منابع رجوع کنید و با اطلاعات بیشتر درمورد دستورات گیت آشنا شوید.

منابع

[ 1 ]: The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress

# [ 2 ]: Head First Git by Raju Gandhi, Publisher(s): O'Reilly Media, Inc

[ 3 ]: [Happy Developer YouTube Channel](https://www.youtube.com/watch?v=rScUEZPeazY)

[ 4 ]: [Images and Content from Git Kraken Course](https://learn.gitkraken.com/courses/git-foundations)

1. پچ یا (وصله) به یک سری از تغییرات گفته می‌شود که برای بروزرسانی، تعمیر و یا بالا بردن کیفیت یک نرم‌افزار کامپیوتری به کار می‌رود.  [↑](#footnote-ref-1)
2. mirror [↑](#footnote-ref-2)
3. Metadata [↑](#footnote-ref-3)
4. Blobs [↑](#footnote-ref-4)
5. fetch [↑](#footnote-ref-5)