نرم افزار کنترل نسخه گیت (**Git**)

کنترل یا مدیریت نسخه چیست؟

منظور از کنترل نسخه[[1]](#footnote-1) عبارت است از مجموعه فرآیند‌هایی که برای پی‌گیری[[2]](#footnote-2) تغییرات اعمال شده بر یک پروژه یا مجموعه در طول دوره‌های تکاملی و رفع عیب آن به کار گرفته می‌شود. این عمل - که بیشتر در مورد اسناد متنی معنی پیدا می‌کند – شامل روشهایی برای شناسایی تغییرات اعمال شده بر هر سند در طی ادوار مختلف گسترش، عاملین آنها و همچنین امکان بازگرداند یا ارجاع به نسخه‌های قبلی در هرزمان – برای مثال بدلیل وجود ایراد در نسخه فعلی – می‌باشد.

برای مثال فرض کنید یک نویسنده مشغول نگارش کتاب جدید خود است در حین نوشتن به چند صفحه قبلتر باز گشته و بعد از مطالعه یک پاراگراف یا بخش از نوشته خود، تصمیم به تعویض آن می‌گیرد. به همین منظور پاراگراف (یا بخش) مورد نظر را پاک کرده و شروع به تایپ می‌کند. روز بعد هنگام شروع به نوشتن ادامه مطلب خود متوجه می‌شود که پاراگرافی که تغییر داده به نوعی با ادامه مطالب هم‌خوانی ندارد و بهتر است مطلب حذف شده را دوباره بازگرداند. در این حالت دو چیز ممکن است به اون کمک کنند؛ اول حافظه و توان ذهنی که بتواند مطلب را به آن شکل که بوده دوباره بازگرداند که هر‌چه دیرتر تصمیم به باز گرداندن متن گرفته شود، امکان مشابهت متن جدید با متن اولیه کمتر خواهد بود. راه حل دیگری که معمولا به ذهن افراد می‌رسد ذخیره یک نسخه کتاب و اعمال تغییرات بر روی نسخه جدید است. این عمل را می‌توان بیان‌گر مفهوم اولیه کنترل نسخه دانست.

نرم‌افزارهای کنترل(مدیریت) نسخه‌

با پیشرفتهای صورت گرفته در زمینه برنامه‌نویسی و نیاز شرکت‌ها (و حتی افراد) به ذخیره نسخه‌های مختلف از برنامه‌های توسعه داده شده خود،‌ نیاز به نرم‌افزارهایی که این کار را با دقت بالا، کمترین خطا و حجم افزوده، بصورت توزیع شده و با توانایی ادغام تغییرات اعمال شده در نسخه‌های مختلف (که معمولا توسط بیش از یک نفر صورت می‌گیرد) انجام دهند به شدت احساس می‌شد به همین دلیل در طول دوره‌های مختلف نرم‌افزارهایی برای این منظور ایجاد شدند.

در واقع منظور از نرم‌افزارهای کنترل نسخه، سیستمی است که تغییرات یک یا چند فایل را در طول زمان در خود ذخیره می‌کند تا در هنگام نیاز به نسخه مورد نظر را باز خوانی کرده و از آن استفاده نماییم. به عنوان نمونه در مثالی که بیان شد، اگر نویسنده بخواهد ورژنهای مختلف کتاب خود را در اختیار داشته باشد، استفاده از نرم‌افزارهای کنترل نسخه خواهند راه عاقلانه‌ای خواهد بود. در این صورت به راحتی می‌تواند تغییرات متون خود را در طول زمان بررسی کند و در صورت نیاز بجای ادامه دادن از نسخه فعلی به نسخه‌ای قدیمی‌تر رجوع کرده و با آن کار خود را ادامه دهد. اگر برای مثال نویسنده هر فصل از مطلب خود را در یک فایل ذخیره کند آنگاه با استفاده از این نرم‌افزارها به راحتی می‌تواند فقط یک فایل را به نسخه قبلی باز گرداند و باقی فایلها دست نخورده خواهند ماند.

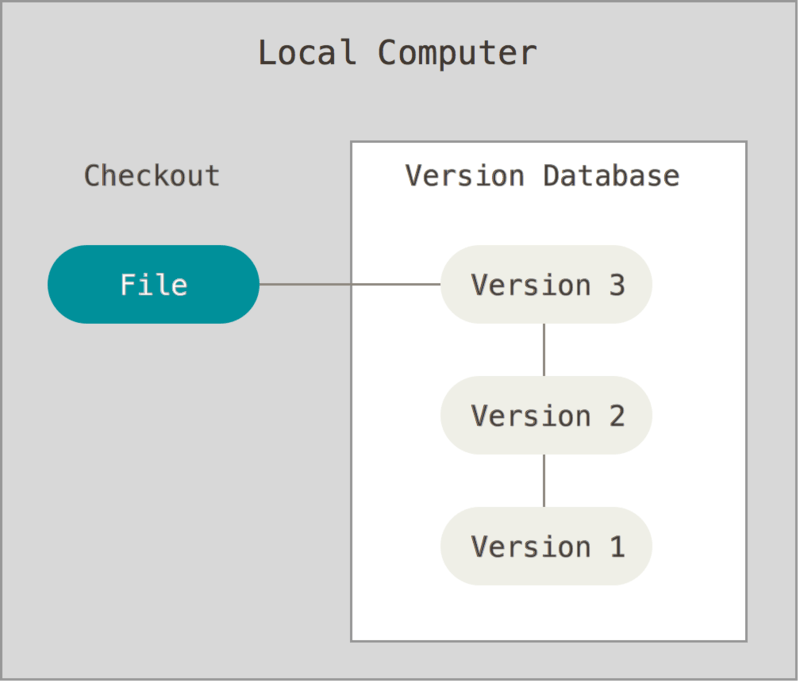
نرم‌افزارهای کنترل نسخه دارای نسلهای مختلفی هستند که در هر نسل پاسخگویی خود به نیازها مختلف کاربران پاسخ دادند. در این بین نرم‌افزارهای مختلفی هم تعریف شد که از جمله آنها ‌می‌توان به «اس وی ان»[[3]](#footnote-3)، «مرکوریال»[[4]](#footnote-4)، «بیت کیپر»[[5]](#footnote-5)، «تی اف اس»[[6]](#footnote-6) و «گیت»[[7]](#footnote-7) اشاره کرد.

نسلهای مختلف نرم‌افزارهای کنترل نسخه

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | امکان شبکه | محدوده عملیاتی | همروندی | نمونه |
| نسل اول | ندارد | یک فایل در آن واحد | قفل فایل | RSC, SCCS |
| نسل دوم | مرکزی | چند فایل | ادغام قبل از ارسال | CVS, SourceSafe, Team Foundation Server, Subversion |
| نسل سوم | توزیع شده | مجموعه تغییرات | ارسال قبل از ادغام | Git, Bazaar, Mercurial |

1. نرم‌افزارهای نسل اول (محلی)

ساده‌ترین روش کنترل نسخه که ممکن است به ذهن خطور کند، ذخیره یک کپی از فایل‌های مورد نظر در پوشه‌ای دیگر خواهد بود که در بهترین حالت نام این پوشه شامل تاریخ نسخه برداری (و یا شماره نسخه) خواهد بود. اما این رویه به شکل غیر قابل باوری مستعد خطا و اشتباه است؛ برای مثال ممکن است فرد فراموش کند در کدام پوشه مشغول کار بوده و به همین دلیل اشتباها فایل دیگری را تغییر دهد یا حذف کند یا در پوشه اشتباه کپی کند.

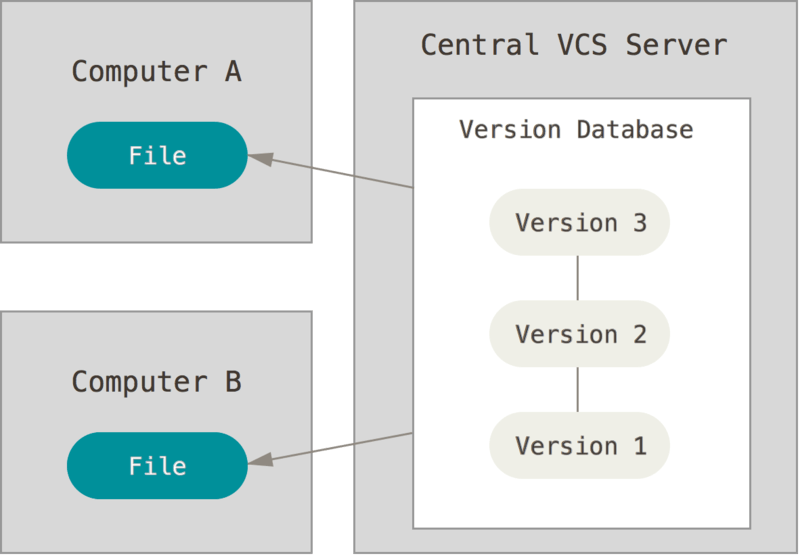


تصویر 1- سیستم کنترل نسخه محلی

به همین منظور نسل اول نرم‌افزارهای کنترل نسخه بوجود آمدند که تغییرات را در یک پایگاه داده بسیار ساده ذخیره می‌کرد. برای مثال می‌توان به نرم‌افزار محبوب RCS اشاره کرد که در حال حاضر نیز به همراه برخی از سیستم عاملها در اختیار کاربر قرار می‌گیرد.

1. نرم‌افزارهای نسل دوم (متمرکز)

بزرگترین مشکلی که کاربران در نسل اول با آن مواجه بودند نیاز آنها برای همکاری با سایر توسعه دهندگان در سایر سیستم‌ها بود. برای برآورده کردن این نیاز نرم‌افزارهای کنترل نسخه متمرکز توسعه یافتند. این سیستمها (که از آن جمله می‌توان به CSV و Perforce اشاره کرد) تشکیل شده بودند از یک سرور که تمامی نسخه‌ها را در اختیار داشت و تعدادی کاربر که نسخه مورد نظر خود را از این مرکز انتخاب ‌کرده و تغییرات خود را بر آن اعمال می‌کردند. برای سالهای متمادی این رویه به عنوان استاندارد نرم‌افزارهای کنترل نسخه مورد توجه بود.



تصویر 2- سیستم کنترل نسخه متمرکز

از مزایای این سیستم در مقابل نسل اول می‌توان به این موارد اشاره کرد:

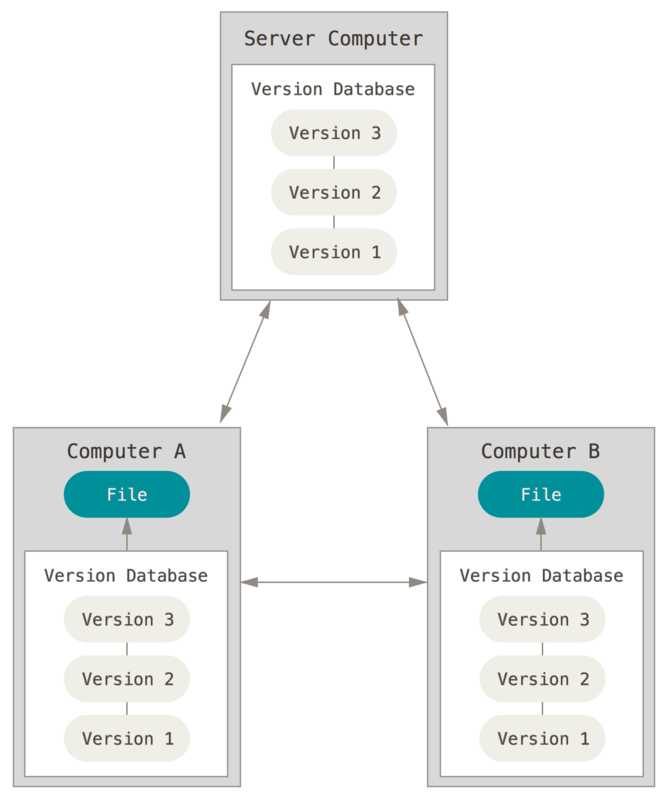
* هر کاربر تا حدودی می‌دانست که دیگران چه کاری بر روی سیستم انجام داده‌اند.
* امکان مدیریت محدوده دسترسی کاربران.
* مدیریت ساده‌تر در مقابل مدیریت بر روی هر کامپیوتر (بصورت محلی)

علاوه بر مزایای بیان شده در بالا برای این سیستم معایبی نیز برای آن وجود دارد:

* معایب وارد بر سایر سیستمهای سرور محور از جمله قطع اتصال از سرور و یا از بین رفتن فایلها در سرور.

1. نرم‌افزارهای نسل سوم (توزیع شده)

در این نسل هر کاربر یک کپی از کل مخزن[[8]](#footnote-8) را در اختیار خواهد داشت. بنابراین در صورتی که سرور به هر دلیلی از دسترس خارج شد یا فایلهای آن از دست رفت می‌توان اطلاعاتی که در اختیار یکی از کاربران هست را با آن جایگزین نمود. علاوه بر این برخی از این سیستم‌ها از چندین مخزن از راه دور[[9]](#footnote-9) استفاده می‌کنند که این امکان را به کاربران می‌دهد تا بر روی یک پروژه به صورت همزمان با گروه‌های مختلف همکاری نماید.



تصویر 3- سیستم کنترل نسخه توزیع شده

تاریخچه گیت

در سالهای ابتدایی توسعه هسته لینوکس (۱۹۹۱- ۲۰۰۲) تغییراتی که قرار بود –به منظور نگهداری و بهبود- بر روی آن اعمال شود بصورت قطعات کد[[10]](#footnote-10) فایلهای فشرده در اختیار توسعه دهنده آن (لینوس تروالدز[[11]](#footnote-11)) قرار می‌گرفت. در سال ۲۰۰۲ اقدام به استفاده از یک سیستم کنترل نسخه توزیع شده[[12]](#footnote-12) اختصاصی به نام بیت کیپر[[13]](#footnote-13) کرد.

در سال ۲۰۰۵ روابط بین انجمن[[14]](#footnote-14) توسعه دهندگان هسته لینوکس و توسعه دهندگان این نرم‌افزار تجاری تیره گردید و به همین دلیل مجوز استفاده رایگان از نرم‌افزار مربوطه لغو شد. این وضعیت انجمن توسعه دهندگان (و به خصوص لینوس تروالدز) را به سمتی سوق داد که با توجه به تجربیات خود در طول استفاده از بیت کیپر دست به کار توسعه ابزاری شوند برای توسعه هسته لینوکس به کار رود.

اهداف سیستم جدید از این قرار بود:

* سرعت.
* طراحی ساده.
* پشتیبانی قوی از توسعه غیر خطی (هزاران شاخه موازی).
* کاملا توزیع شده.
* توانایی مدیریت پروژه‌های بزرگ مانند هسته لینوکس به نحو احسن.

شاخه‌ها در گیت

تقریبا هر سیستم کنترل نسخه به نوعی از شاخه‌سازی[[15]](#footnote-15) پشتیبانی می‌کنید. شاخه‌سازی به معنی دور شدن از خط اصلی توسعه و انجام کارها بدون دستکاری نسخه اصلی است. در برخی از سیستم‌های کنترل نسخه این عمل بسیار هزینه بر است و معمولا نیازمند ایجاد یک کپی از پوشه منبع برنامه می‌باشد (که با توجه به حجم پروژه ممکن است زمان‌بر باشد).

روشی که گیت برای ایجاد شاخه‌ها به کار می‌برد کم هزینه است که باعث می‌شود تا عملیات ایجاد شاخه و همچنین جابجایی بین شاخه‌ها به سرعت انجام پذیرد. بر خلاف سایر نرم‌افزارهای کنترل نسخه، گیت گردش‌های کار[[16]](#footnote-16) را به ایجاد و ادغام مکرر (حتی چند بار در روز) ترقیب می‌کند. بنابراین یادگیری و درک این خصیصه تاثیر بسزایی در استفاده بهتر از گیت خواهد داشت.

گیت داده‌ها را به صورت یک سری از مجموعه تغییرات یا تفاوت‌ها نگه داری نمی‌کند و بح جای این کار از یک سری تصویر[[17]](#footnote-17) به منظور نگهداری تغییرات استفاده می‌کند.

وقتی شما یک تغییر را اعمال می‌کنید، گیت یک شیء[[18]](#footnote-18) شامل یک اشاره‌گر به آن تصویری که از محتوایی که تغییر داده‌اید ذخیره می‌کند. علاوه بر آن، شیء شامل نام ایجاد کننده تغییرات، ایمیل، پیامی که برای تغییرات تعریف شده و همچنین اشاره‌گری به تمامی تغییرات اعمال شده‌ای که دقیقا قبل این تغییر واقع شده‌اند (تحت عنوان والد یا والدین[[19]](#footnote-19)) در خود ذخیره می‌کند. اولین تغییرات بدون والد و تغییرات معمولی با یک والد و تغییرات ناشی از ادغام دو یا چند شاخه دو یا چند والد خواهند داشت.

برای مثال فرض کنید شما یک مخزن برای کار ایجاد کرده‌اید و سپس سه فایل ایجاد نموده و تغییرات را در سیستم ثبت می‌نمایید.

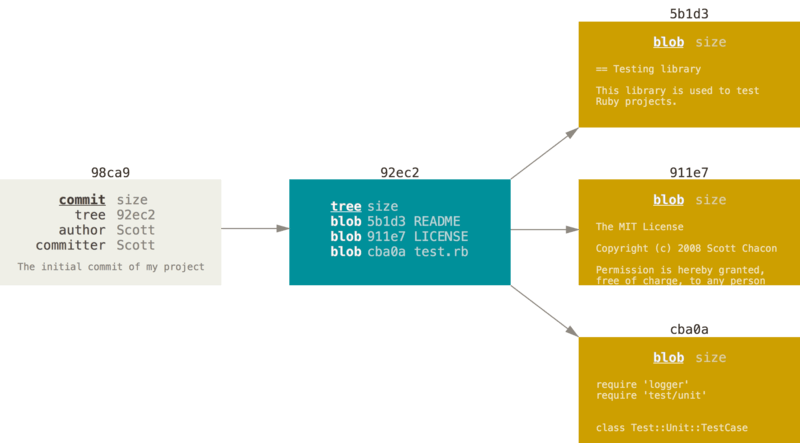
$git init

$ git add README test.rb LICENSE

$ git commit -m 'The initial commit of my project'

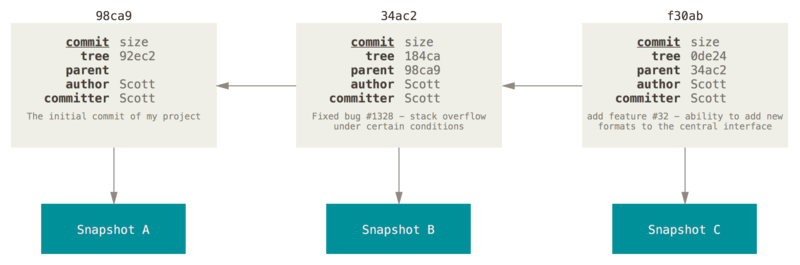
وقتی با دستور git commit تغییرات خود را اعمال می‌کنید، گیت جمع کنترلی[[20]](#footnote-20) هر زیر پوشه (در این مثال فقط پوشه ریشه پروژه بررسی می‌شود) را محاسبه کرده و این سه شیء را در مخزن ذخیره می‌کند. پس از آن گیت یک شیء کامیت شامل فراداده و اشاره‌گر به ریشه درخت پروژه ایجاد می‌کند که در صورت نیاز به کمک آن می‌تواند دوباره تصویر مورد نظر را دوباره ایجاد کند.

مخزن گیت شما در حال حاضر شامل پنج شیء است:‌ یک شیء از نوع blob به ازای هر فایل شامل محتوای آنها، یک درخت که محتوای پوشه را لیست کرده و تعیین می‌کند که محتویات مربوط به هر نام فایل در کدام شیء blob ذخیره شده است و یک شیء کامیت که شامل فرا داده‌ها و اشاره‌گری به ریشه درخت می‌باشد.



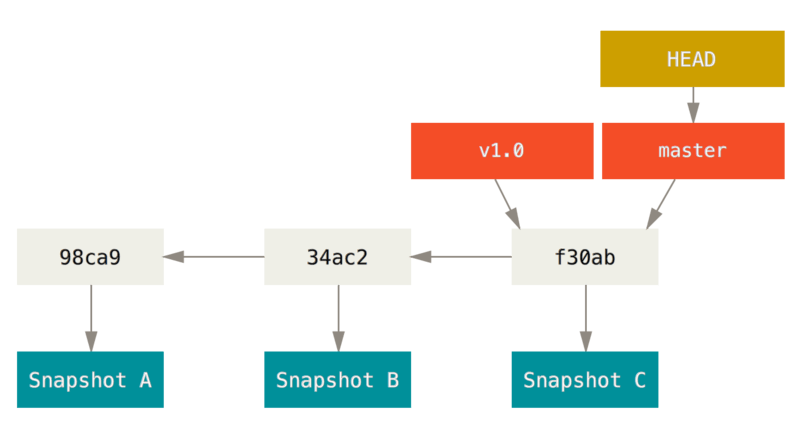
تصویر 4- یک کامیت و ساختار درختی آن

اگر دوباره تغییرات جدیدی اعمال کنید، هر کامیت یک اشاره‌گر به کامیت قبلی خود نگه خواهد داشت:



تصویر 5- کامیت و والد

یک شاخه در گیت، یک اشاره‌گر کم هزینه و متحرک است که به یکی از کامیت‌ها اشاره می‌کند. شاخه پیش‌فرض در گیت «ارباب[[21]](#footnote-21)» نامیده می‌شود. با اعمال هر تغییر یک شاخه ارباب در اختیار شما قرار می‌گیرد که به آخرین تغییرات اعمال شده اشاره می‌کند. و با هر بار اعمال تغییرات این اشاره گر به صورت اتوماتیک به جلو حرکت می‌کند.



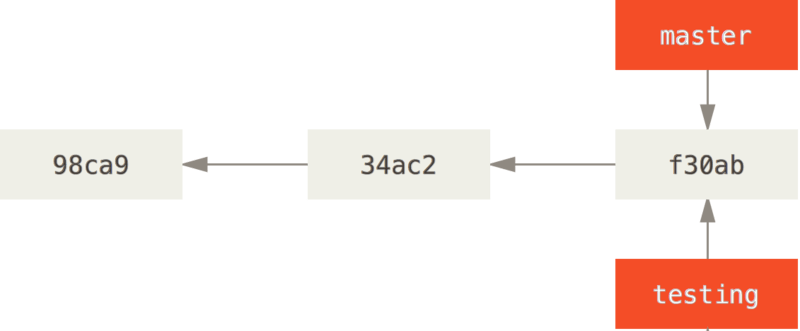
تصویر 6- یک شاخه و سوابق(تاریخچه) آن

ایجاد شاخه جدید

با ایجاد یک شاخه جدید در واقع یک اشاره‌گر جدید ایجاد می‌کند برای مثال دستور زیر باعث ایجاد شاخه‌ای با نام testing خواهد شد:

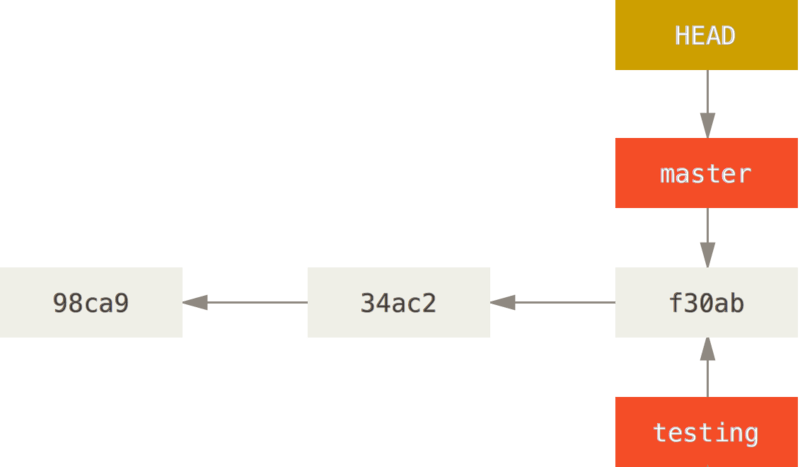
$ git branch testing

این عمل یک اشاره‌گر جدید به کامیتی که در حال حاضر بر آن قرار دارید تعریف می‌کند.



تصویر 7- دو شاخه به کامیت‌های یکسان اشاره می‌کنند

گیت برای تشخیص اینکه در حال حاضر کدام شاخه مورد استفاده قرار گرفته از یک اشاره‌گر خاص به نام «HEAD» استفاده می‌کند که به شاخه محلی که در حال حاضر انتخاب شده اشاره می‌کند (در این مثال ما هنوز در شاخه ارباب قرار داریم) دستور git branch فقط شاخه جدیدی ایجاد می‌کند ولی شاخه فعلی را عوض نمی‌کند.

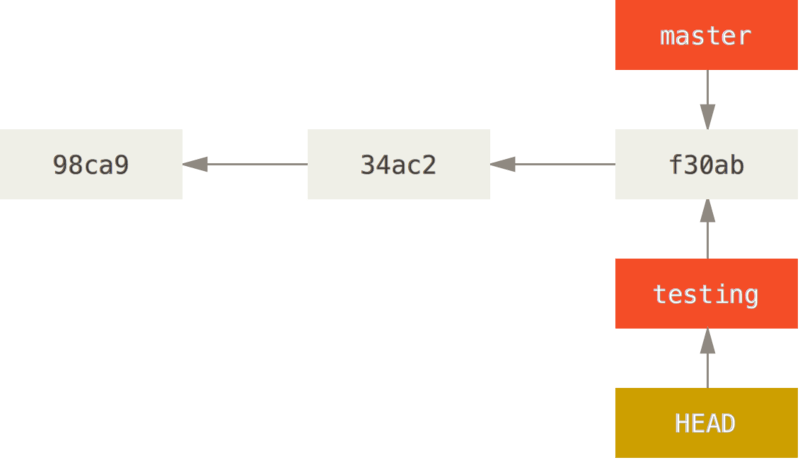


تصویر 8- اشاره‌گر HEAD به شاخه ارباب اشاره می‌کند.

جابجایی بین شاخه‌ها

برای جابجایی بین شاخه‌ها می‌توان از دستور git checkout استفاده کرد. دستور زیر باعث اشاره HEAD به شاخه testing خواهد شد.

$ git checkout testing

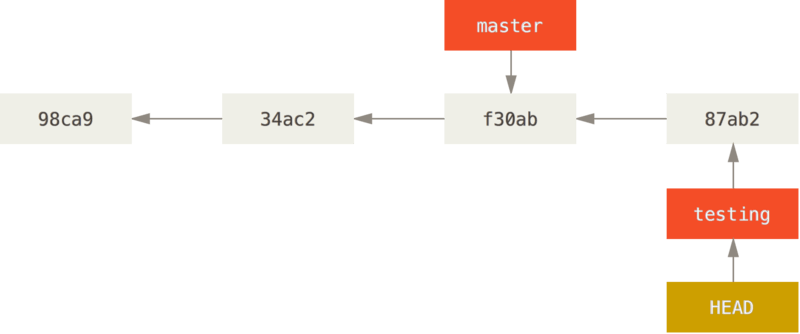


تصویر 9- اشاره‌گر HEAD به شاخه testing اشاره می‌کند.

در صورتی که تغییری در این شاخه اعمال شود این شاخه به جلو حرکت خواهد کرد.

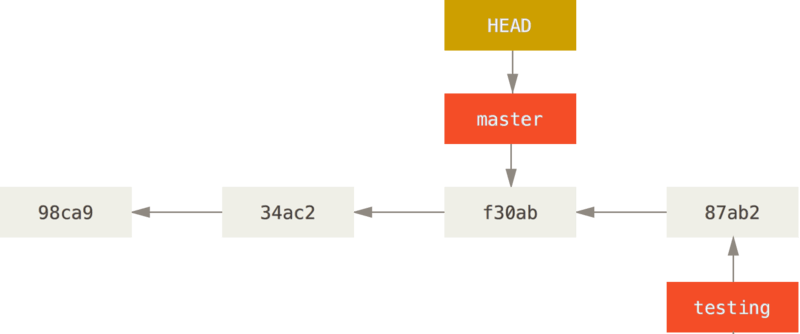
$ vim test.rb

$ git commit -a -m 'made a change'



تصویر 10- شاخه testing با اعمال تغییرات به جلو حرکت می‌کند.

در این حالت با جابجایی به شاخه ارباب دو اتفاق رخ خواهد داد:

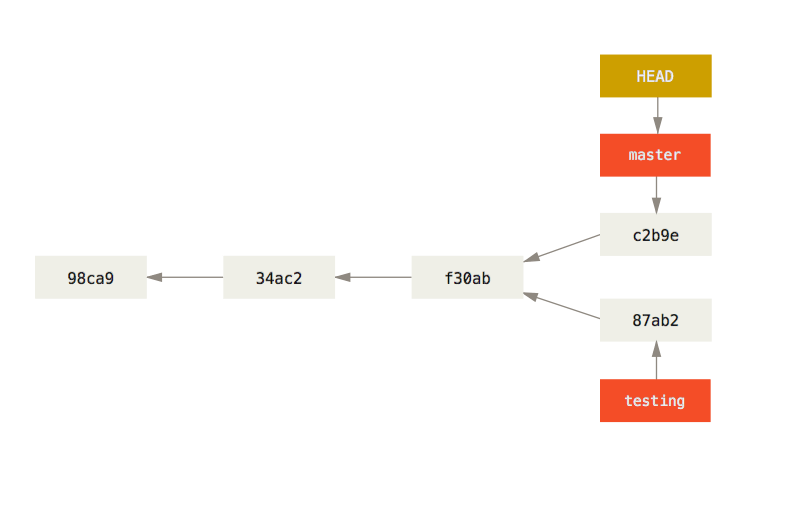


تصویر 11- اشاره‌گر HEAD با اعمال دستور checkout جابجا می‌شود.

اشاره‌گر HEAD را به عقب برده تا به شاخه ارباب اشاره کند. و همچنین تمامی فایلها را به نقطه‌ای که اشاره‌گر ارباب به آن اشاره می‌کند باز ‌می‌گرداند. این بدان معنی است که تمامی تغییراتی که از این نقطه به بعد صورت می‌پذیرد، از نسخه قدیمی پروژه جدا خواهد بود.

در هنگام جابجایی از یک شاخه به شاخه قدیمی‌تر، پوشه کار به آن نقطه بازگردانده می‌شود و اگر گیت به دلیلی نتواند این تغییر را اعمال کند اجازه جابجایی به شما نخواهد داد.

حال اگر بر روی شاخه ارباب تغییری اعمال کنیم تاریخچه دو شاخه از یک‌دیگر جدا خواهد شد. شما به راحتی می‌توانید بین این دو شاخه جابجا شده تغییرات جدید در آنها ایجاد کرده و اعمال نمایید و در نهایت –وقتی آماده بودید- آنها را در هم ادغام نمائید. این کارها با چند دستور ساده branch, checkout, commit صورت پذیرفت.

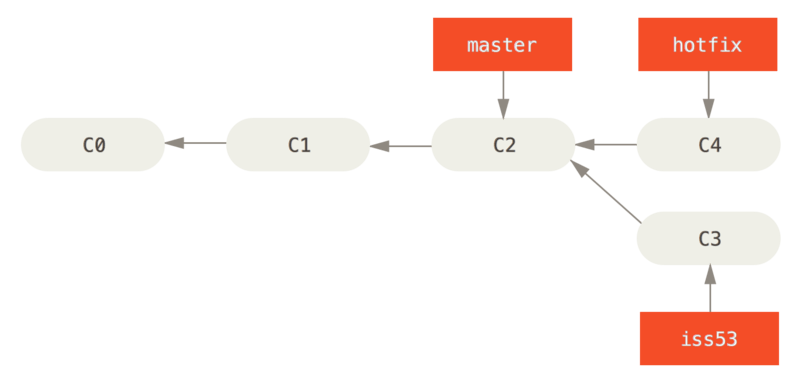


تصویر 12- تاریخچه متمایز

ادغام شاخه‌ها

در گیت ادغام به روشهای متفاوتی صورت می‌گیرد که از جمله آنها می‌توان به رو به جلو[[22]](#footnote-22)، بازگشتی[[23]](#footnote-23) و عصاره‌گیری[[24]](#footnote-24) اشاره کرد. به مثال زیر توجه کنید:

فرض کنید برای رفع ایرادی که طبق سیستم پی‌گیری ایرادات با عنوان Issue #53 نام‌گذاری شده، یک شاخه از برنچ ارباب با نام iss53 ایجاد می‌کنید. در حین کار به شما اعلام می‌شود که ایرادی در نسخه اصلی (نسخه‌ای که توسط شاخه ارباب پیش می‌رود) وجود دارد که باید به سرعت رفع شود. به همین منظور شما تغییرات خود را در شاخه محلی اعمال می‌کنید (یا شاید فقط به صورت موقت ذخیره کنید) و به شاخه اصلی رفته شاخه‌ای به نام hotfix ایجاد کرده و مشکل مربوطه را بر روی آن رفع می‌نمایید.



تصویر 13- دو شاخه iss53 و hotfix از شاخه اصلی منشعب شده‌اند.

حال باید تغییرات شاخه hotfix را در شاخه اصلی اعمال نمایید. برای این کار از دستور ادغام استفاده می‌کنیم و عملیات به صورت زیر خواهد بود:

$ git checkout master

$ git merge hotfix

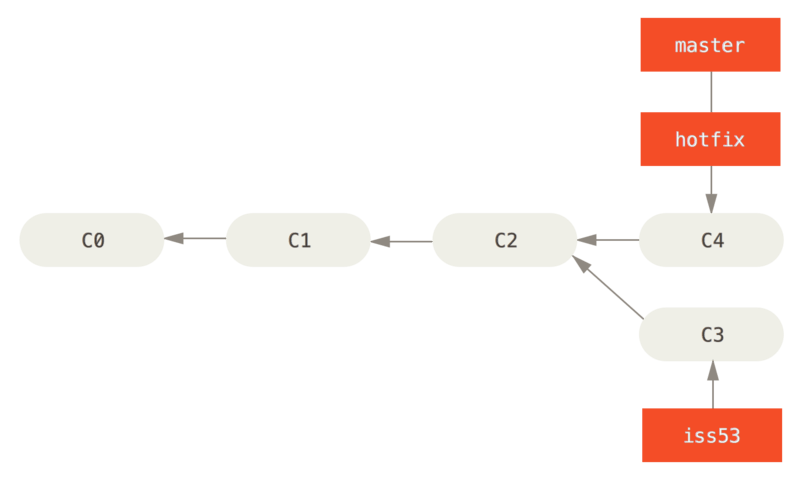
Updating f42c576..3a0874c

Fast-forward

index.html | 2 ++

1 file changed, 2 insertions(+)

در اعمال این ادغام – که به آن رو به جلو می‌گویند – به دلیل اینکه کامیت مورد نظر (C4) در مسیر مستقیم به آخرین کامیت شاخه اصلی (C2) منتهی می‌شود لذا به راحتی و فقط با جابجایی اشاره‌گر شاخه اصلی به کامیت تغییرات در ان اعمال می‌شود و نیازی به کار اضافه‌ای نیست.



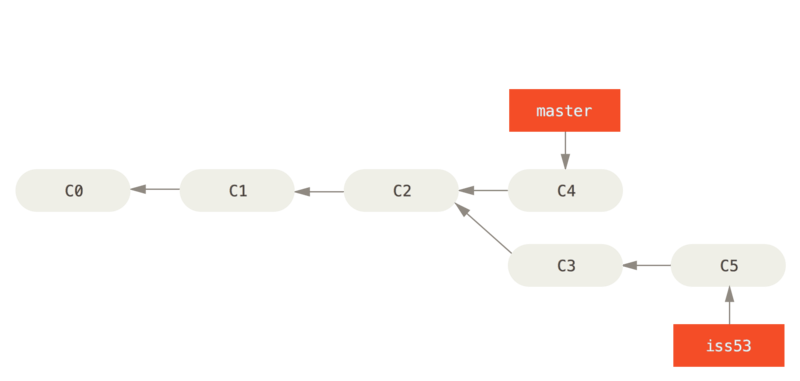
تصویر 14- حرکت شاخه اصلی به سمت شاخه hotfix

از آنجایی که تغییرات در شاخه اصلی اعمال شده،‌ شما می‌توانید شاخه hotfix را حذف کنید.

$ git branch -d hotfix

Deleted branch hotfix (3a0874c).

حال به شاخه iss53 بازمی‌گردیم و کار خود را در آن شاخه ادامه می‌دهیم و پس از اتمام تغییرات را اعمال می‌کنیم.



تصویر 15- ادامه کار در شاخه iss53

با اتمام رفع ایراد حال باید آنها را در شاخه اصلی اعمال کنیم. برای این کار لازم است تا تغییرات را در شاخه اصلی اعمال نماییم. به این منظور باید شاخه اصلی را انتخاب کرده و سپس دستور ادغام را ارسال نماییم.

$ git checkout master

Switched to branch 'master'

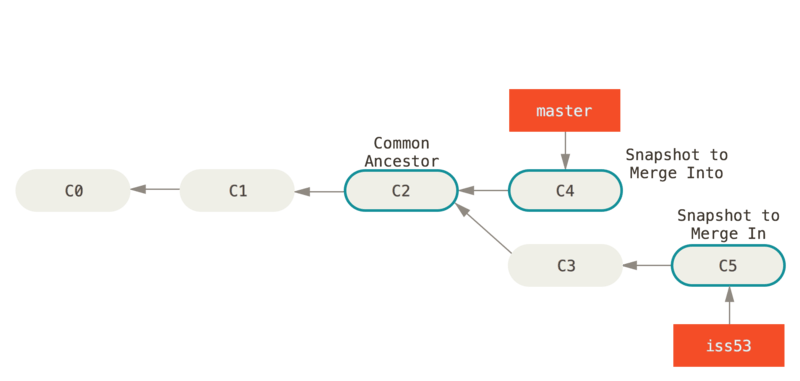
$ git merge iss53

Merge made by the 'recursive' strategy.

index.html | 1 +

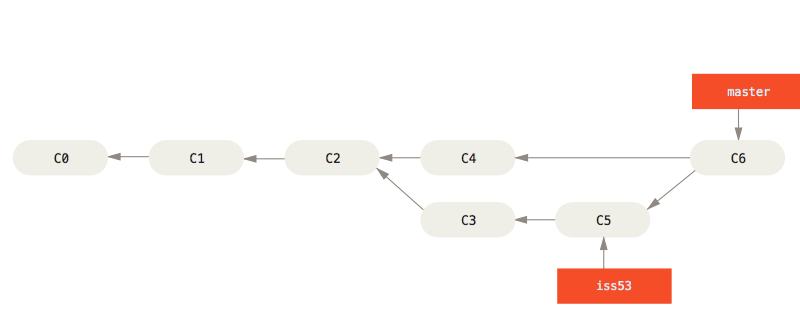
1 file changed, 1 insertion(+)

تفاوتی که بین این ادغام و ادغام قبلی وجود دارد، این است که در اینجا تاریخچه دو شاخه در نقطه‌ای از یک دیگر متمایز شده است. به همین دلیل گیت از یک روش ادغام سه جهته[[25]](#footnote-25) ساده استفاده می‌کند تا بوسیله تصاویر دو کامیت نزدیک‌ترین جد[[26]](#footnote-26) مشترک را پیدا کند.



تصویر 16- سه تصویر که برای ایجاد ادغام استفاده می‌شوند.

سپس تصویر جدیدی ایجاد می‌کند که نتیجه‌ی این ادغام سه جهته خواهد بود.



تصویر 17- کامیت ادغام شده.

بر خلاف برخی دیگ از نرم‌افزارهای کنترل نسخه مانند subversion (نسخه ۱.۵ و قبل از آن) –که تشخیص نقطه اشتراک توسط خود فرد انجام می‌شد– گیت بصورت خودکار نزدیک‌ترین والد (جد) مشترک بین دو کامیت را پیدا کرده و ادغام را بر مبنای آن انجام می‌دهد.

مناقشه در ادغام[[27]](#footnote-27)

اگر تغییرات مشابهی در دو شاخه بر روی یک فایل ایجاد شود، ادغام آنها به راحتی که در بالا توضیح داده شده نخواهد بود. فرض کنید در مثال قبل در دو شاخه hotfix و iss53 هردو تغییری در بخش یکسانی از یک فایل داشته باشند، آنگاه گیت در هنگام ادغام دچار مناقشه می‌شود.

$ git merge iss53

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

در این حالت گیت کامیت جدید را ایجاد نکرده و تا حل و فصل مناقشات[[28]](#footnote-28) پروسه را موقتا متوقف می‌نماید. در این حالت برای اطلاع از اینکه پروژه در چه فایلهایی دچار مناقشه شده می‌توان از دستور git status استفاده نمود. هر موردی که از دچار مناقشه باشد بعنوان ادغام نشده[[29]](#footnote-29) در لیستی نمایش داده می‌شود.

$ 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")

گیت نشانگرهای استانداردی برای نمایش محدوده مناقشه در فایل مربوطه قرار می‌دهد:

<<<<<<< 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 نشانگر‌ها بیانگر این است که تغییرات در نسخه‌ای که HEAD به آن اشاره می‌کند (در اینجا شاخه اصلی)، شامل قسمت بالایی (هر چیزی که بالای ======= قرار دارد) است در حالی که تغییرات در شاخه iss53 شامل مواردی است که در قسمت پایین آمده. برای رفع مناقشه کافیست تغییر مورد نظر خود را از بین دو بخش انتخاب کرده یا حتی بجای آنها چیز دیگری قرار دهید. برای مثال بخش مناقشه را با این کد جایگزین می‌کنیم که بخشی از دو تغییر را داشته و همچنین خطوط شامل <<<<<<< ، ======= و >>>>>>> را حذف می‌کنیم.

<div id="footer">

please contact us at email.support@github.com

</div>

بعد از اینکه مناقشات هر فایل رفع گردید می‌توان با استفاده از دستور git add به همراه نام فایل آنرا به تغییرات اضافه می‌کنیم که به معنی رفع مناقشه است. بعد از رفع مشکل یک فایل می‌توان با دستور git status وضعیت جدید را بررسی کرد و در صورتی که ایرادی وجود نداشت می‌توان با دستور git commit روند ادغام را ادامه داد.

$ git status

On branch master

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

modified: index.html

در صورت اجرای دستور git commit پیام پیش‌فرض بصورت زیر خواهد بود که در صورت تمایل می‌توانید متن آنرا تغییر دهید:

Merge branch 'iss53'

Conflicts:

index.html

#

# It looks like you may be committing a merge.

# If this is not correct, please remove the file

# .git/MERGE\_HEAD

# and try again.

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

# On branch master

# All conflicts fixed but you are still merging.

#

# Changes to be committed:

# modified: index.html

#

شاخه‌های راه دور

شاخه‌های راه دور[[30]](#footnote-30) به شاخه‌های گفته می‌شود که بر روی سرور گیت مورد نظر قرار دارند. از آنجایی که معمولا توسعه نرم‌افزار توسط تیم صورت می‌گیرد، لذا افراد مختلف در سیستم‌های مختلف مشغول توسعه سیستم خواهند بود که نیازمند دسترسی به مخازن مشترک دارند. این مخازن معمولا در یک سرور که افراد گروه به آن دسترسی دارند قرار می‌گیرند که هر مخزن حاوی یک یا چند شاخه مربوط به خود است. افراد گروه با دستور git clone یک نمونه از این مخزن را در سیستم خود قرار می‌دهند و می‌توانند بر روی آن تغییرات خود را اعمال کنند. بعد از دریافت کپی از مخزن مورد نظر تنها شاخه‌ای که به صورت محلی در اختیار هست و می‌توان از آن استفاده کرد، شاخه اصلی خواهد بود و بقیه شاخه‌ها (در صورت وجود) بصورت مستقیم در دسترس نیستند.

شاخه‌هایی در مخزن دارای ارجاع[[31]](#footnote-31) به شاخه‌های راه دور هستند و آنها را پی‌گیری می‌نمایند، و در هر ارتباط با سرور بروز می‌شوند در نمایش شاخه‌ها آنها را به شکل (branch)/(remote) نمایش داده می‌شوند. برای مثال اگر بخواهیم بدانیم وضعیت برنچ اصلی در سرور به چه صورت است، تنها کافی است بعد از بروز رسانی شاخه origin/master[[32]](#footnote-32) را بررسی نماییم.

واکشی و کشیدن

برای به روز رسانی شاخه‌های مرجع[[33]](#footnote-33) می‌توان از دستور واکشی[[34]](#footnote-34) استفاده کرد. این دستور داده‌های موجود روی شاخه های مرجع را با استفاده از داده‌های سرور اصلی به‌روز می‌کند.

Git fetch

در شاخه‌های محلی که شاخه‌های مرجع را پی‌گیری می‌کنند، با اجرای دستور کشیدن[[35]](#footnote-35) علاوه بر بروز رسانی شاخه‌های مرجع شاخه محلی مورد نظر را نیز به‌روز خواهد کرد و درصورتی که تاریخچه شاخه مورد نظر از شاخه مرجع جدا شود در حالت عادی تغییرات محلی به بصورت عادی در تغییرات مرجع ادغام می‌شود.

نشاندن

برای اشتراک گذاری شاخه‌های محلی از دستور نشاندن[[36]](#footnote-36) استفاده می‌شود.

git push <remote> <branch>

در مثال زیر شاخه servefix به اشتراک گذاشته شده است.

$ git push origin serverfix

Counting objects: 24, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (15/15), done.

Writing objects: 100% (24/24), 1.91 KiB | 0 bytes/s, done.

Total 24 (delta 2), reused 0 (delta 0)

To https://github.com/schacon/simplegit

\* [new branch] serverfix -> serverfix

از این پس هر یک از همکاران که پروژه را واکشی کند، شاخه مورد نظر در شاخه‌های مرجع وی قرار خواهد داشت:

$ git fetch origin

remote: Counting objects: 7, done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 3 (delta 0), reused 3 (delta 0)

Unpacking objects: 100% (3/3), done.

From https://github.com/schacon/simplegit

\* [new branch] serverfix -> origin/serverfix

ولی برای ایجاد تغییرات در آن باید یک شاخه محلی برای آن ایجاد نماید. این دستور یک شاخه محلی از شاخه مرجع می‌نماید.

$ git checkout -b serverfix origin/serverfix

Branch serverfix set up to track remote branch serverfix from origin.

Switched to a new branch 'serverfix'

پی‌گیری شاخه‌ها

ایجاد یک شاخه محلی از شاخه‌های راه دور به صورت خودکار امکان پی‌گیری آن را فراهم می‌آورد. (شاخه ایجاد شده را شاخه پی‌گیر[[37]](#footnote-37) و شاخه‌ی راه دور را شاخه بالادست[[38]](#footnote-38) می‌نامند). از آنجایی که شاخه پی‌گیر در ارتباط مستقیم با شاخه بالادست هستند، هرگاه بر روی آنها دستور git pull اجرا شود، گیت می‌داند که از کدام سرور آنها را به‌روز نماید.

در حالت عادی وقتی از یک شاخه مرجع شاخه‌ای محلی ایجاد شود به صورت خودکار آنرا پی‌گیری می‌کند. ولی در صورتی که شاخه‌ای محلی ایجاد شده باشد که بخواهیم شاخه‌ای مرجع را بعنوان شاخه بالا دستی آن معرفی نماییم، از دستور زیر استفاده می‌کنیم:

$ git branch -u origin/serverfix

Branch serverfix set up to track remote branch serverfix from origin.

این دستور شاخه مرجع origin/serverfix را بعنوان شاخه بالادست شاخه محلی فعلی (در اینجا serverfix) تعیین می‌کند.

حذف شاخه‌ها

برای حذف شاخه محلی تنها کافیست دستور زیر را بر روی آن اجرا کنیم:

$ git branch –d <branch>

برای حذف شاخه مرجع از دستور زیر استفاده می‌کنیم:

$ git push origin --delete serverfix

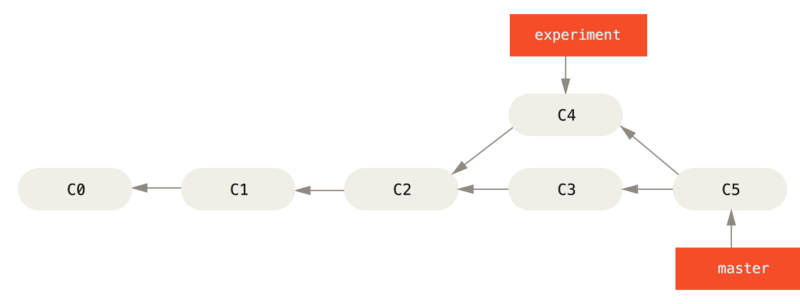
To https://github.com/schacon/simplegit

- [deleted] serverfix

در حذف شاخه مرجع در واقع اشاره‌گر مربوط به آن حذف می‌شود و خود شاخه دست نخورده باقی می‌ماند تا زمانی که عملیات زباله روبی[[39]](#footnote-39) صورت پذیرد. در این صورت اگر بصورت ناخواسته شاخه‌ای حذف شد به راحتی قابل بازیابی خواهد بود.

تغییر مبنا

در گیت دو روش برای یکپارچه سازی وجود دارد؛ ادغام[[40]](#footnote-40)، و تغییر مبنی[[41]](#footnote-41). همانطور که قبلا توضیح داده شد به هنگام ادغام –که ساده‌ترین روش یکپارچه سازی است– دو شاخه در صورتی که تفاوتی در تاریخچه نداشته باشند، فقط اشاره‌گر در شاخه به سمت اشاره‌گر شاخه مقصد حرکت خواهد کرد و در صورتی که جدایی در تاریخچه وجود داشته باشد، تصویر جدیدی بر مبنای سه تصویر (تصویر آخرین کامیت هر شاخه و تصویر نزدیکترین جد (والد) مشترک) با دو والد –که به دو تصویر آخر شاخه‌ها اشاره دارد– ایجاد خواهد شد.



تصویر 18- یکپارچه سازی تاریخچه به کمک ادغام.

راه دیگر این است که تغییرات ایجاد شده در شاخه فعلی را گرفته و بعد از تغییرات شاخه مورد نظر اعمال کنیم. به این کار در گیت تغییر مبنی گفته می‌شود که با دستور rebase قابل انجام است:

$ git checkout experiment

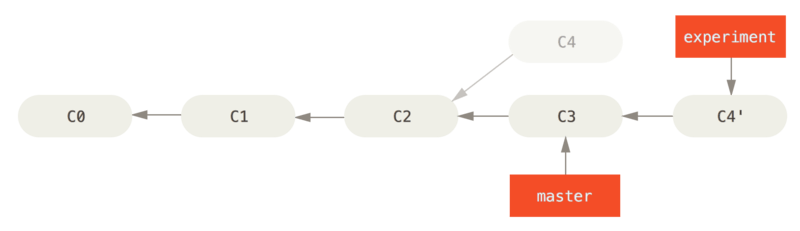
$ git rebase master

First, rewinding head to replay your work on top of it...

Applying: added staged command

نحوه کار به اینگونه است که با اجرای این دستور گیت روی شاخه فعلی به عقب رفته و تمامی تغییرات محلی را تک‌به‌تک در یک فایل موقت ذخیره می‌نماید که کامیت مورد نظر را حذف می‌کند تا جایی که دو شاخه از نظر تاریخچه برابر شوند. آنگاه مانند ادغام رو به جلو تمامی تغییرات شاخه مورد نظر را اعمال می‌نماید. بعد از آن دوباره تغییرات محلی را به همان ترتیب در ادامه آن قرار می‌دهد. در صورتی که در حین تغییر مبنی مناقشه‌ای پیش بیاید عملیات همان‌جا متوقف می‌شود تا مناقشه توسط کاربر رفع شود. و پس از آن با دستور rebase --continue عملیات ادامه پیدا می‌کند.

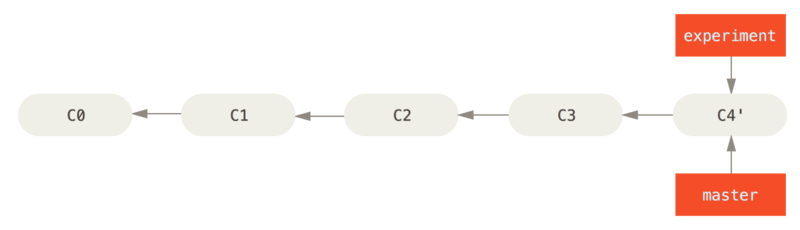
در این حالت با اجرای دستور ادغام روی شاخه اصلی ادغام از نوع رو به جلو انجام خواهد شد.



تصویر 19- تغییر مبنی تغییرات ایجاد شده در C4 به C3 .

$ git checkout master

$ git merge experiment



تصویر 20- ادغام رو به جلو در شاخه اصلی.

در اینجا اشاره‌گر دقیقا به همانجایی اشاره می‌کند که در مثال ادغام، به C5 اشاره می‌کرد به این معنی که از نظر تغییرات تفاوتی وجود نخواهد داشت ولی تاریخچه فعالیتها به صورت خطی و مرتب خواهد ماند.

منابع:

1. <http://ericsink.com/vcbe/html/history_of_version_control.html>
2. [https://git-scm.com/book/en/v2/](https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control)

1. Version Control [↑](#footnote-ref-1)
2. Track [↑](#footnote-ref-2)
3. Apache Subversion (SVN). [↑](#footnote-ref-3)
4. Mercurial [↑](#footnote-ref-4)
5. BitKeeper [↑](#footnote-ref-5)
6. Microsoft Team Foundation Server [↑](#footnote-ref-6)
7. Git [↑](#footnote-ref-7)
8. مخزن (Repository) به محلی گفته می‌شود که تمامی فایلهای که باید مورد پی‌گیری قرار گیرند در آن قرار خواهند گرفت. مخزن ساختار درختی فایلها را نیز پی‌گیری می‌کند. [↑](#footnote-ref-8)
9. Remote [↑](#footnote-ref-9)
10. Patch [↑](#footnote-ref-10)
11. Linus Torvalds [↑](#footnote-ref-11)
12. DVCS [↑](#footnote-ref-12)
13. BitKeeper [↑](#footnote-ref-13)
14. Community [↑](#footnote-ref-14)
15. Branching [↑](#footnote-ref-15)
16. Workflows [↑](#footnote-ref-16)
17. Snapshot [↑](#footnote-ref-17)
18. Object [↑](#footnote-ref-18)
19. Parent or parents [↑](#footnote-ref-19)
20. Checksum [↑](#footnote-ref-20)
21. Master [↑](#footnote-ref-21)
22. Fast Forward. [↑](#footnote-ref-22)
23. Recurcive. [↑](#footnote-ref-23)
24. Squash [↑](#footnote-ref-24)
25. Three-way merge. [↑](#footnote-ref-25)
26. Ancestor. [↑](#footnote-ref-26)
27. Merge Conflict. [↑](#footnote-ref-27)
28. Resolve Conflics. [↑](#footnote-ref-28)
29. Unmerged. [↑](#footnote-ref-29)
30. Remote Branches. [↑](#footnote-ref-30)
31. Reference. [↑](#footnote-ref-31)
32. کلمه origin همانند کلمه master بعنوان موارد پیشفرض تعریف می‌شوند و قابل تغییر هستند. [↑](#footnote-ref-32)
33. Reference Branches. [↑](#footnote-ref-33)
34. Fetch. [↑](#footnote-ref-34)
35. Pull. [↑](#footnote-ref-35)
36. Push. [↑](#footnote-ref-36)
37. Tracking Branch. [↑](#footnote-ref-37)
38. Upstream Branch. [↑](#footnote-ref-38)
39. Garbage Collection. [↑](#footnote-ref-39)
40. Merge. [↑](#footnote-ref-40)
41. Rebase. [↑](#footnote-ref-41)