عنوان: **#8 آموزش سیستم مدیریت کد Git**
نویسنده: حسام امامی
تاریخ: ۱۳۹۱/۰۷/۰۲ ۲۰:۳۹
آدرس: www.dotnettips.info
برچسب‌ها: Git

در این بخش به بررسی چگونگی ایجاد branchها و همچنین نحوه‌ی merge کردن آن‌ها خواهیم پرداخت.

:Branch

در این مقاله به بررسی شاخه‌ها و همچنین ضرورت ایجاد آن‌ها پرداخته شده است. جهت ایجاد یک شاخه می‌توان از دستور زیر استفاده کرد:

```
git branch  [branch name]
```

توجه کنید که دستور فوق تنها یک شاخه را ایجاد می‌کند؛ اما همچنان git در شاخه جاری باقی می‌ماند.
همچنین جهت مشاهده شاخه‌های ایجاد شده از دستور زیر استفاده می‌شود:

```
git branch
```

شاخه جاری، با یک علامت * در کنار آن مشخص می‌شود:

در حالت پیش‌فرض، تمامی عملیات در git، در شاخه master انجام می‌گیرد. برای تعویض و رجوع به شاخه ایجاد شده می‌توان از دستور checkout استفاده کرد. همانطور که قبلا گفته شد، یکی دیگر از کاربردهای این دستور تعویض شاخه‌ها است:

```
git checkout [branch name]
```

همچنین می‌توان به صورت همزمان هم شاخه جدید ایجاد کرد و هم به این شاخه جدید سوئیچ نمود:

```
git checkout -b [branch name]
```

تذکر:

در صورتیکه working tree تقریبا clean نباشد، یعنی تغییراتی در فایل‌ها صورت گرفته باشد که این تغییرات هنوز در repository ذخیره نشده باشند، git امکان تعویض شاخه را نخواهد داد. علت تقریبا به این جهت است که در مواردی git می‌تواند برخی تفاوتها را نادیده بگیرد؛ مثلا اگر فایلی در شاخه‌ی دیگر وجود نداشته باشد. در این حالت سه راهکار پیش روی کاربر است:

۱) حذف تغییرات
۲) ذخیره تغییرات در repository
۳) استفاده از stash

دو مورد نخست مشخص هستند و استفاده از stash در ادامه همین مقاله آورده شده است.

برای حذف یک شاخه ایجاد شده از دستور زیر استفاده می‌شود:

```
git branch -d [branch name]
```

در این حالت نباید در شاخه‌ای باشیم که قصد حذف آن را داریم. همچنین اگر تغییرات در شاخه والد موجود نباشند، git هشداری را مبنی بر آن‌که «شاخه دارای تغییراتی است که در صورت حذف آن از بین می‌روند» به کاربر می‌دهد. در این حالت اگر مسر به انجام حذف باشیم، دستور فوق را این بار با D- به کار می‌بریم. بنابراین جهت جلوگیری از اشتباه بهتر است دستور حذف ابتدا با d انجام شود و در صورت نیاز از D استفاده شود.
برای تغییر نام یک شاخه از دستور زیر استفاده می‌شود:

```
git branch -m [old name][new name]
```

ادغام شاخه‌ها:
معمولا بعد از آن‌که ویرایش فایل‌ها در یک شاخه به پایان رسید و فایل‌های نهایی تولید شدند، باید این فایل‌ها را در شاخه‌ای دیگر مثلا master قرار داد. برای این منظور، از دستور merge استفاده می‌شود. در هنگام merge باید در شاخه مقصد قرار داشت؛ یعنی در همان شاخه‌ای که قرار است فایل‌های شاخه‌ای دیگر با آن ادغام شوند.
برای ادغام یک شاخه به شاخه دیگر از دستور زیر استفاده می‌شود:

```
git merge [branch name]
```

نکته مهم:
در git دو نوع ادغام وجود دارد:
۱) fast forward
۲) real merge
حالت اول زمانی اتفاق می‌افتد که در شاخه والد، commit جدیدی ثبت نشده باشد. در این حالت در هنگام merge، اشاره‌گر آخرین فرزند والد، به اولین commit در شاخه‌ی فرزند اشاره می‌کند و دقیقا مانند یک زنجیر دو شاخه به هم متصل می‌شوند. اما اگر در شاخه والد بعد از تشکیل شاخه فرزند commit هایی صورت گرفته باشد، ما یک real merge خواهیم داشت.

تداخل یا conflict:
در هنگام merge کردن شاخه‌ها گاهی این مساله به وجود می‌آید که فایل‌هایی که قرار است تغییرات آن‌ها با هم ادغام شوند، به گونه‌ای ویرایش شده‌اند که git نمی‌تواند عمل merge را انجام دهد. به عنوان مثال تصور کنید فایلی دارای ۱۰ خط است. در شاخه والد خطوط ۱ و ۴ و در شاخه فرزند خطوط ۲ و ۴ ویرایش شده‌اند. git برای ادغام فایل، برای خطوط ۱ و ۲ دچار مشکلی نیست؛ زیرا خط یک را از شاخه والد و خط ۲ را از شاخه فرزند بر می‌دارد. اما برای خط ۴ چه کار کند؟ git نمی‌تواند تصمیم بگیرد که داده نهایی از خط شماره ۴ فرزند است و یا والد. به همین جهت در این‌جا ما یک merge conflict داریم. برای رفع این مشکل یا می‌توان با استفاده از دستور زیر از انجام merge صرفنظر کرد:

```
git merge --abort
```

و یا به صورت دستی و یا با استفاده از برخی از ابزارهای موجود، اقدام به رفع دوگانگی فایل‌ها کرد. بعد از رفع conflictها با دستور:

```
git merge --continue
```

می‌توان ادامه ادغام را خواستار شد.

Stash:

در هنگام توضیح چگونگی تعویض شاخه‌ها، به مطلبی به نام stash اشاره شد. Stash در واقع مکان جدایی در git است که از آن به عنوان محلی جهت ذخیره‌سازی موقت تغییرات استفاده می‌شود. عملکرد stash مانند commit می‌باشد. با این تفاوت که SHA-1 منحصر به فردی برای آن در نظر گرفته نمی‌شود. بنابراین stash محلی است که به طور موقت می‌تواند تغییرات فایل‌ها را ذخیره کند.

برای ایجاد یک stash از دستور زیر استفاده می‌شود:

```
git stash save "[stash name]"
```

همچنین جهت مشاهده تمامی stashهای ذخیره شده از دستور زیر می‌توان استفاده کرد:

```
git stash list
```

در صورت اجرای این دستور، همانطور که در شکل زیر مشخص است، هر stash توسط یک شماره به صورت:

```
stash@{number}
```

مشخص می‌شود.

برای مشاهده تغییرات در یک stash از دستور زیر استفاده میشود:

```
git stash show stahs@{[number]}
```

همچنین در صورتیکه جزئیات بیشتری مورد نیاز باشد، می‌توان p- را قبل از شماره stash به دستور فوق اضافه کرد.

در صورتیکه بخواهید stash ایجاد شده را حذف کنید، می‌توانید از دستور زیر استفاده کنید:

```
git stash Drop [stash name]
```

همچنین می‌توان با دستور زیر کل stashهای موجود را حذف نمود:

```
git stash clear
```

برای اعمال تغییرات با استفاده از stash می‌توان از دو دستور استفاده کرد:

۱) pop : در این حالت همانند ساختار پشته، آخرین stash اعمال و از لیست stashها حذف می‌شود.

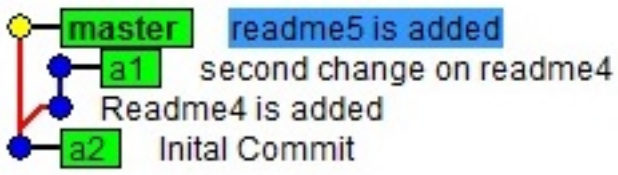
۲) apply : در این دستور، در صورتیکه شماره stash ذکر شود، آن stash اعمال می‌شود. در غیر این صورت، آخرین stash استفاده خواهد شد. تفاوت این دستور با دستور فوق در این است که در اینجا stash بعد از استفاده حذف نمی‌گردد.

دستور rebase:

عملکرد این دستور برای بسیاری از افراد چندان واضح و مشخص نیست و نمی‌توانند تفاوت آن را با دستور merge به خوبی دریابند. برای درک بهتر این موضوع سناریوی زیر را در نظر بگیرید:

تصور کنید شما در حال توسعه یک برنامه هستید و هر از چندگاهی نیاز پیدا می‌کنید تا باگ‌های ایجاد شده در برخی از فایل‌های قبلی خود را رفع کنید. برای این منظور شما برای هر فایل، شاخه‌ای جدید ایجاد کرده و طی چند مرحله، هر فایل را اصلاح می‌کنید. سپس شاخه ایجاد شده را در شاخه اصلی ادغام می‌کنید. حال تصور کنید که تعداد این فایل‌ها افزایش یافته و مثلا به چند صد عدد برسد. در این حالت شما دارای تعداد زیادی شاخه هستید که تا حدود زیادی سوابق فایل‌های شما را دچار پیچیدگی می‌کنند. در این حالت شاید بهتر باشد که دارای یک فایل سابقه خطی باشیم. بدین معنا که بعد از merge سوابق، شاخه اصلی شما به گونه‌ای در خواهد آمد که انگار هیچ وقت شاخه‌های اضافی وجود نداشته‌اند و تمام تغییرات برای هر فایل پشت سر هم و در شاخه اصلی اتفاق افتاده‌اند. برای این منظور می‌توانید از دستور rebase استفاده کنید.

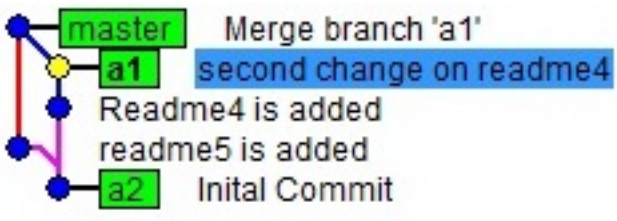
به مثال‌های زیر و شکل شاخه‌ها بعد از اعمال دستورات merge و rebase توجه کنید :



در شاخه master فایل readme5 اضافه شده و در شاخه a2 فایل readme4 اضافه شده و بعد تغییری در آن ذخیره شده است

```
$ git log --oneline
d6ed8ef Merge branch 'a1'
e01afc2 readme5 is added
5182a64 second change on readme4
31d9419 Readme4 is added
183b405 Inital Commit
```

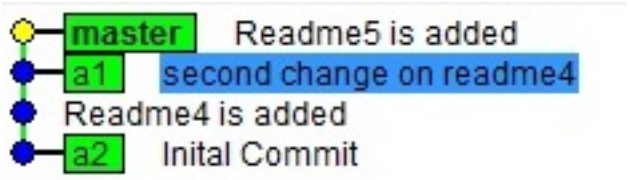
شاخه a1 در master ادغام شده است



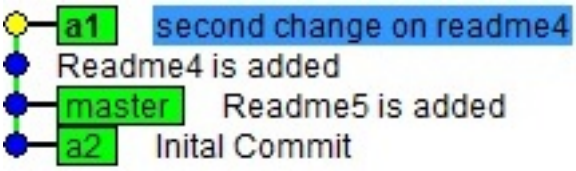
شکل درختی شاخه‌ها پس از ادغام

```
$ git log --oneline
1144826 second change on readme4
4cf90f6 Readme4 is added
184b399 Readme5 is added
183b405 Inital Commit
```

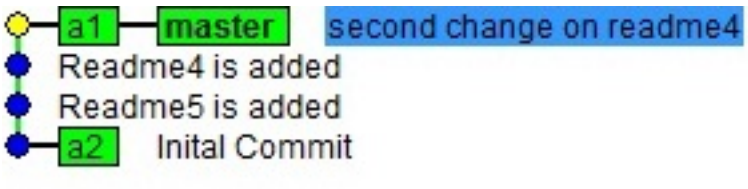
در شکل فوق از دستور rebase استفاده شده است



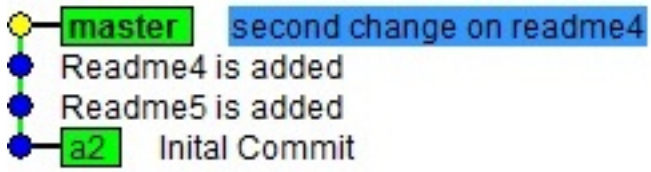
شکل شاخه‌ها بعد از اعمال rebase



همانطور که مشاهده می‌شود با سوئیچ به شاخه master هنوز head در محل قبلی خود است



با اعمال دستور ادغام، head به محل آخرین commit منتقل می‌شود



اکنون می‌توان شاخه a1 را حذف کرد. همانطور که دیده می‌شود، به نظر می‌رسد این شاخه هیچگاه وجود نداشته است.

تذکر:
بعد از انجام دستور rebase باید از دستور merge استفاده کرد. زیرا هر شاخه برای خود head جداگانه‌ای دارد. بعد از اجرای این فرمان، هنوز head در شاخه مقصد به آخرین فرمان خود اشاره می‌کند. در آخرین فرمان، شاخه‌ای اضافه شده، بنابراین اجرای دستور merge حالت fast forward را پیاده می‌کند و head به آخرین commit منتقل می‌شود.

تذکر:

همانطور که مشاهده کردید، دستور rebase به صورت فوق سوابق شاخه را از بین می‌برد. بنابراین نباید از این دستور برای شاخه‌های عمومی یعنی آنهایی که دیگران تغییرات آنها را دنبال می‌کنند استفاده کرد.
شکل استفاده از این دستور به صورت زیر است:

```
git rebase [destination branch]
```

یا

```
git rebase [destination][source]
```

همانند دستور merge این دستور نیز ممکن است سبب ایجاد تداخل شود و برای رفع این موضوع باید مانند merge عمل کرد؛ این دستور نیز دارای دو اصلاح کننده abort-- و continue-- می‌باشد

تذکر مهم :

به تفاوت محل درج ادغام‌ها در merge و rebase توجه کنید.

دستور cherry-pick :
با استفاده از این فرمان می‌توان یک یا چند commit را از شاخه‌ای برداشته و در شاخه‌ی دیگری اعمال کنیم. در واقع دستور cherry-pick همانند بخشی از دستور rebase است. با این تفاوت که rebase در واقع چندین cherry-pick را یک‌جا انجام می‌دهد. البته در cherry-pick هر commit بدون تغییر باقی می‌ماند.
بیشترین کاربرد این دستور برای اعمال patchها و رفع باگ‌ها در یک شاخه است. این دستور به صورت زیر استفاده می‌شود:

```
git cherry-pick [branch name]
```