عنوان: **#2 آموزش سیستم مدیریت کد Git**
نویسنده: حسام امامی
تاریخ: ۱۳۹۱/۰۵/۱۲ ۱۶:۵۵
آدرس: www.dotnettips.info
برچسب‌ها: Git

در ادامه آموزش Git، به بررسی مفاهیم مورد استفاده در این سیستم مدیریت کد می‌پردازیم. البته ذکر این نکته ضروری است که ممکن است برخی از تعاریف زیر، برای افرادی که تا کنون با اینگونه سیستم‌ها کار نکرده‌اند، مبهم باشد. اما مشکلی نیست؛ زیرا در دروس بعدی کار با Git، به صورت عملی، این مفاهیم به شکل دقیق‌تر و کاربردی‌تر بیان می‌شوند. هدف در اینجا تنها ایجاد یک تصویر کلی از نحوه کار سیستم‌های مدیریت کد توزیع شده است.

تعاریف زیر هر چند برای Git نوشته شده‌اند، اما می‌توانند در بقیه DVCSها نیز کاربرد داشته باشند.

Commit:

بعد از آن که برنامه نویسان از صحت کدهای خود مطمئن شدند، برای ثبت وضعیت فعلی باید آن‌ها را commit کنند. با این کار یک نسخه جدید از فایل‌ها ایجاد می‌شود. به این ترتیب امکان بازگشت به نقطه فعلی درآینده به وجود خواهد آمد.

Pushing:

بعد از انجام عملیات Commit، معمولا برنامه نویسان می‌خواهند کدهای نوشته شده را با دیگران به اشتراک بگذارند. این کار به وسیله عملیات Pushing صورت می‌گیرد. بنابراین pushing عبارت است از عملی که با استفاده از آن داده‌ها از یک Repository به Repository دیگر جهت به اشتراک گذاری انتقال می‌یابد. معمولا به این مخزن Upstream Repository می‌گویند. Upstream Repository یک مخزن عمومی برای تمامی برنامه نویسانی است که تغییرات فایل‌های خود را در آنجا push می‌کنند.

Pulling:

عملیات Pushing تنها نیمی از آن چیزی است که برنامه نویسان برای حفظ به روز بودن کدهای خود به آن احتیاج دارند. در بسیاری از موارد آن‌ها نیاز دارند تا تغییرات فایل‌ها و آخرین به روز رسانی‌ها را نیز دریافت کنند. این کار در دو مرحله متفاوت انجام می‌شود:

1) بازیابی داده‌ها از مخزن عمومی (fetch)

2) الحاق داده‌های دریافت شده با داده‌های فعلی

معمولا در بسیاری از سیستم‌های مدیریت کد، چون به هر دوی این عملیات توامان نیاز است، با یک دستور هر دو کار انجام می‌شود. به مجموع عملیات فوق Pulling گویند.

Branchها (شاخه‌ها):

Branch و یا همان شاخه، به ما این امکان را می‌دهد که بتوانیم برای قسمت‌های مختلف یک پروژه که روند تولید آن‌ها با هم ارتباط مستقیمی ندارند، سوابق فایلی متفاوتی را ایجاد کنیم.

به عنوان مثال تصور کنید که در یک پروژه سه تیم متفاوت وجود دارد

1) تیم توسعه برنامه

2) تیم تست و اشکال یابی

3) واحد گرافیکی

در این حالت منطقی است به جای آن که سوابق فایل‌ها برای همه یکسان باشد، هر تیم، شاخه مخصوص به خود را داشته باشد، تا تنها تغییرات فایل‌های مربوطه را پیگیری کند و در نهایت بعد از آن که از صحت کار خود مطمئن شد، آن را در یک شاخه اصلی برای استفاده دیگر تیم‌ها قرار دهد.

در Git شاخه اصلی master نام دارد و فایل‌ها به صورت پیش فرض در این شاخه قرار داده می‌شوند. استاندارد کار بر آن است که در شاخه master تنها فایل‌های نهائی قرار گیرند.

:Merging

به عملیات ادغام دو یا چند شاخه با یکدیگر Merging گفته می‌شود. در بعضی موارد، در روند توسعه یک برنامه نیاز است که شاخه‌هایی جهت مدیریت بهتر کد ایجاد شود. اما بعد از توسعه این قسمت ها، می‌توان شاخه‌های ایجاد شده را با هم ادغام نمود تا تغییرات فایل‌ها در یک شاخه قرار گیرند. مثلا در یک تیم توسعه فرض کنید دو گروه وجود دارند که کدهای مربوط به دسترسی داده را می‌نویسند و هر دو را در یک شاخه فایل‌های خود، نگهداری می‌کنند. گروه اول بر روی کلاس‌های انتزاعی و گروه دوم بر روی کلاس‌های عملی کار می‌کنند. به منظور اینکه گروه دوم به اشتباه کلاس‌های انتزاعی را که هنوز کامل نیستند پیاده سازی نکند، دو شاخه از شاخه اصلی ایجاد می‌شود و هر گروه در شاخه‌ای مجزا قرار می‌گیرد. گروه اول تنها کلاس‌های انتزاعی را در شاخه مشترک قرار می‌دهد که کار آنها تمام شده باشد و گروه دوم تنها همان کلاس‌ها را پیاده سازی و در شاخه مشترک می‌گذارد. بعد از آنکه کار این دو بخش پایان گرفت می‌توان هر سه شاخه را در یک شاخه مثلا بخش کدهای دسترسی داده قرار داد.

البته عملیات Merging می تواند باعث ایجاد مشکلی به نام Conflict شود که خوشبختانه Git روش هایی را برای مدیریت این مشکل دارد که در مقالات بعد به آن اشاره خواهد شد.

:Locking

با استفاده از این کار می‌توان مانع تغییر یک فایل توسط برنامه نویسان دیگر شد. معولا Locking به 2 صورت است

1)Strict Locking

2) Optimistic Locking

در روش اول بعد از آن که فایلی قفل شد همان کسی که فایل را قفل کرده تنها امکان تغییر آن را خواهد داشت؛ که البته این روش مناسب سیستم‌های توزیع شده نیست.

در روش دوم فرض بر این است که تغییراتی را که هر کس بر روی فایل می‌دهد، به گونه‌ای باشد که هنگام ادغام این تغییرات، اختلالی ایجاد نشود. یعنی وظیفه بر عهده مصرف کننده فایل است که آگاهی داشته باشد چگونه فایل را تغییر دهد. هنگامی که فایلی به این روش قفل می‌شود، اگر در حین تغییر فایل توسط ما، شخص دیگری فایل را تغییر داده باشد و آن را pull کرده باشد ما در زمان push فایل با خطا مواجه می‌شویم. سیستم از ما می‌خواهد که ابتدا تغییرات فایل را pull کنیم و سپس فایل را push نمائیم. در هنگام pull اگر برنامه نویسی قوانین تغییرات فایل را رعایت نکرده باشد، ممکن است اعمال تغییرات با خطا همراه گردد.

تعاریف فوق بخشی از مفاهیم اولیه مورد نیاز Git بود. اما ما در ادامه به بررسی objectهای Git و همچنین نحوه ذخیره سازی و مدیریت فایل‌ها در این سیستم مدیریت کد خواهیم پرداخت.

نظرات خوانندگان

نویسنده: پژمان
تاریخ: ۱۳۹۱/۰۵/۱۲ ۱۸:۱۸

سلام؛ خسته نباشید. با تشکر.

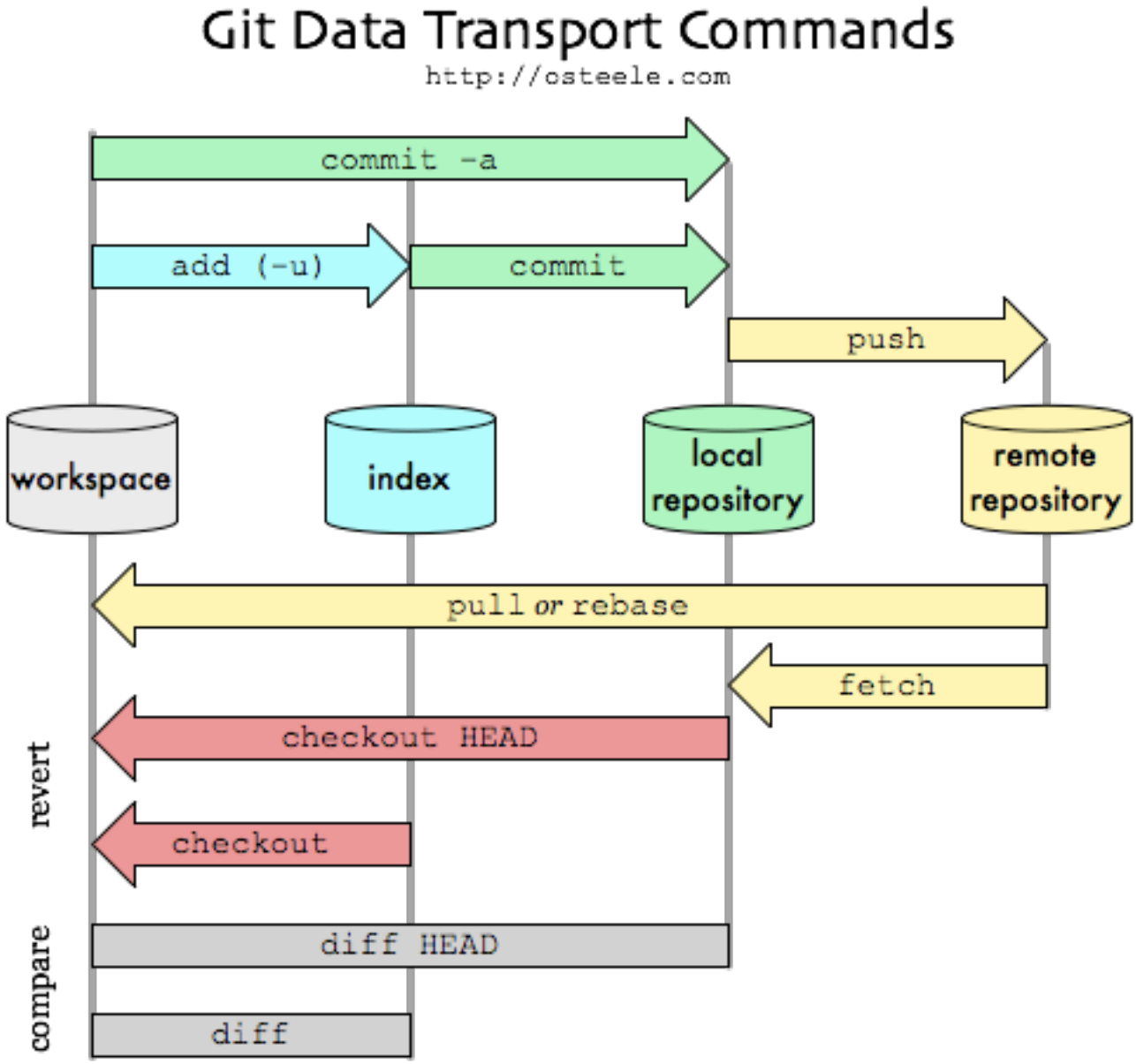
من قبلا با svn کار کردم. به نظر می‌رسه که در git این commit به مخزن local است نه مخزن اصلی یا upstream در اینجا. درسته؟

نویسنده: حسام امامی
تاریخ: ۱۳۹۱/۰۵/۱۲ ۱۹:۵۹

با سلام بله شما ابتدا باید در مخزن محلی commit را انجام دهید بعد در صورتی بخواهید، می‌توانید اطلاعات را در مخزن upstream، push کنید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۹ ۹:۲۱

گردش کاری توضیح داده شده به صورت تصویری :


Git Data Transport Commands
http://osteele.com
commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
checkout
diff HEAD
diff
revert
compare