**گیت**

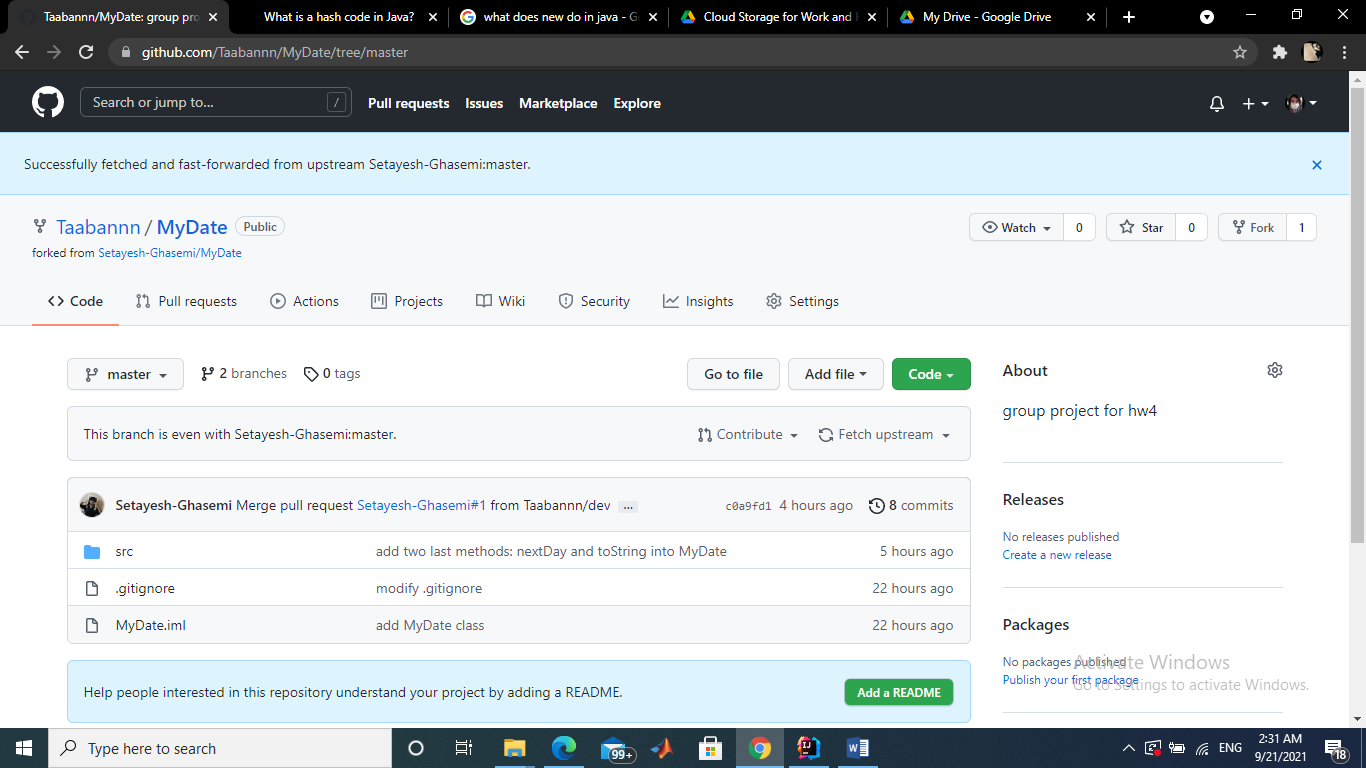
1 - از git stash command زمانی استفاده می­کنیم که بخواهیم وضعیت فعلی تغییرات را ذخیره نماییم. در واقع در روند کار هنوز به جایی نرسیده­ایم که از git commit –m استفاده نماییم، اما می­خواهیم تغییرات اعمال شده تا اینجا را ذخیره کرده تا در فرصت بعدی آن را تکمیل نماییم. با استفاده از این دستور، اصلاحات مدنظر ما در جایی دیگر ذخیره می­گردد و هنوز HEAD به commit قبلی اشاره می­کند. با استفاده از git stash list می­توانیم به لیست stash های قبلی دست یابیم. با استفاده از دستور git stash show می­توانیم وضعیت فایل در state مدنظرمان را ببینیم و برای تغییرات موقتی می­توانیم از git stash apply استفاده نماییم.

2 – با استفاده از git checkout و با داشتن شناسه­ی کامیت، می­توانیم تغییرات کامیت مدنظر را مشاهده نماییم. (محدودیتی نیز در انتخاب کامیت­ها وجود ندارد.) با استفاده از دستور git revert مقدار HEAD به commit یکی مانده به آخر در تاریخچه ی کامیت ها اشاره می­کند. با استفاده از دستور git reset –hard HEAD~1 تمام تغییرات ایجاد شده توسط آخرین کامیت به طور کلی از بین می رود ودر این حالت HEAD به کامیت یکی مانده به اخری اشاره می­کند، در حالی که با استفاده از دستور git reset –soft HEAD~1 تغییرات ایجاد شده توسط آخرین کامیت حفظ می­شود ودر این حالت نیز HEAD به کامیت یکی مانده به اخری اشاره می­کند.

3 – با استفاده از دستور git rebase در صورتیکه نام شاخه­ای در جلوی آن ذکر شود، به طور اتوماتیک قبل از انجام کار دیگری به آن شاخه switch می­کند و در غیر اینصورت در همان شاخه­ی فعلی باقی می­ماند. در صورتی که عملیات merge شاخه­ها موفقیت آمیز نباشد با استفاده از دستور git rebase --continue می توان مشکل را حل نمود یا با استفاده از دستور git rebase --skip می توان کامیتی را که با شکست مواجه شده bypass نمود و یا با استفاده از git rebase --abort می توان فایل­های git/rebase. را پاک نمود و با استفاده از git rebase --onto می­توانیم تغییرات دلخواه را در درخت نمایش دهنده­ی گیت ایجاد نماییم.

4 – در واقع git workflow یک دستورالعمل یا توصیه نامه برای چگونگی استفاده از گیت به منظور افزایش بازدهی (و یکدست بودن تغییرات ایجاد شده توسط کسانی که به گیت برای ویرایش فایلها دسترسی دارند.) است. به عنوان مثال یه بخش مرکزی برای merge نمودن شاخه­های فرعی با شاخه­ی اصلی ایجاد می­نماییم. اینگونه تمامی کاربران به یک ورژن از فایلها دسترسی دارند و بعد از اعمال تغییرات توسط بخش مرکزی هربار آن را آپدیت می­کنند. این گونه conflict ها را راحت­تر می توان مدیریت نمود زیرا هر کاربر برای merge کردن شاخه­ی کاری خود با شاخه­ی اصلی مجبور است تا هربار برای merge نمودن به بخش مرکزی یک pull request بفرستد و پس از بررسی­های انجام شده توسط بخش مرکزی آن شاخه­ی فرعی با شاخه­ی اصلی merge می­شود.

5 – در شکل 1 نتیجه­ی کار و repository مشترک را می­بینید. (در این کلاس پیاده­سازی توابع toString و nextDay با من بوده است.)



شکل 1