**گردش کار (Workflow) در Git**

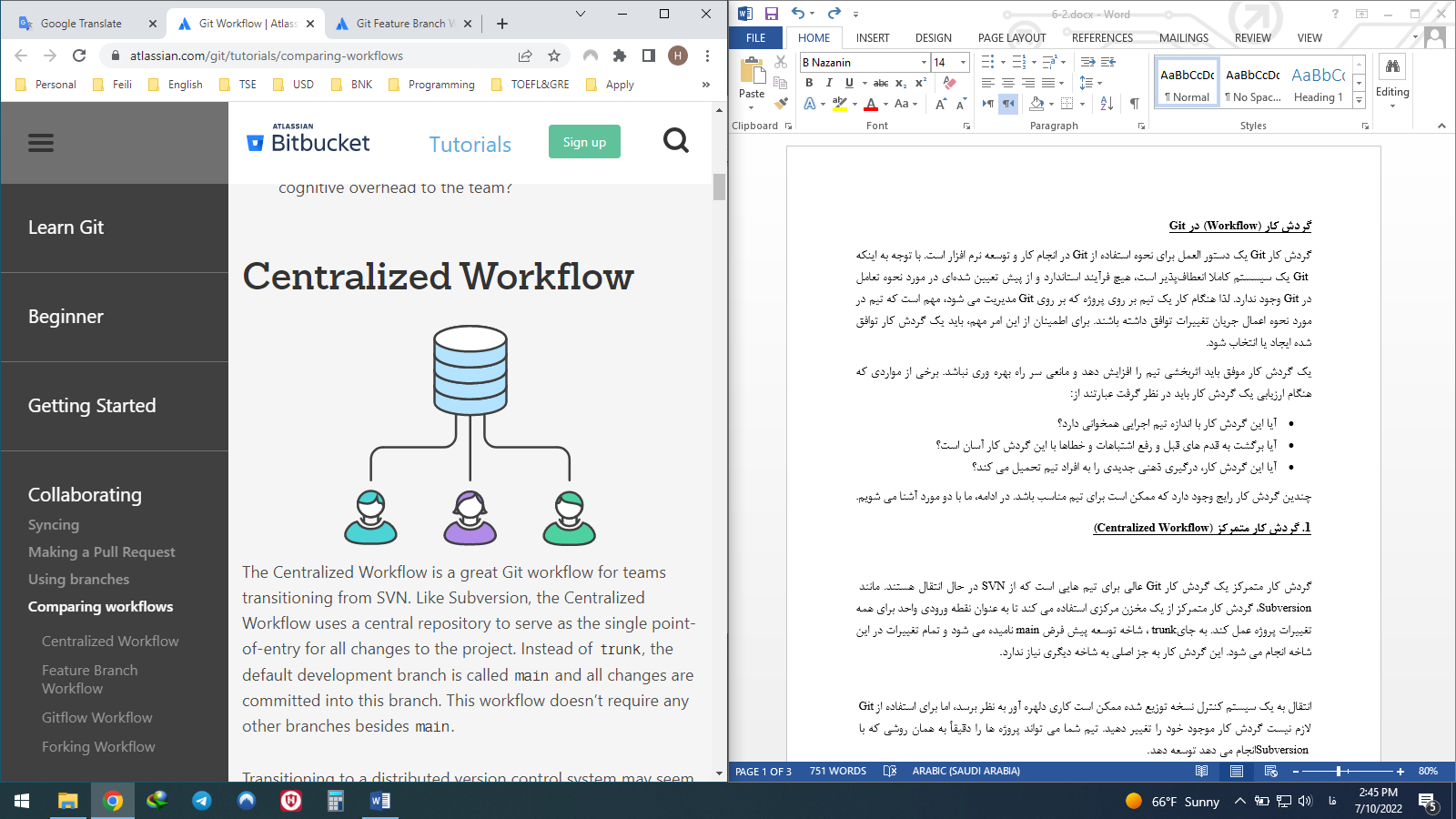
گردش کار Git یک دستور العمل برای نحوه استفاده از Git در انجام کار و توسعه نرم افزار است. با توجه به اینکه Git یک سیسستم کاملا انعطاف‌پذیر است، هیچ فرآیند استاندارد و از پیش تعیین شده­ای در مورد نحوه تعامل در Git وجود ندارد. لذا هنگام کار یک تیم بر روی پروژه که بر روی Git مدیریت می شود، مهم است که تیم در مورد نحوه اعمال جریان تغییرات توافق داشته باشند. برای اطمینان از این امر مهم، باید یک گردش کار توافق شده ایجاد یا انتخاب شود.

یک گردش کار موفق باید اثربخشی تیم را افزایش دهد و مانعی سر راه بهره وری نباشد. برخی از مواردی که هنگام ارزیابی یک گردش کار باید در نظر گرفت عبارتند از:

* آیا این گردش کار با اندازه تیم اجرایی همخوانی دارد؟
* آیا برگشت به قدم های قبل و رفع اشتباهات و خطاها با این گردش کار آسان است؟
* آیا این گردش کار، درگیری ذهنی جدیدی را به افراد تیم تحمیل می کند؟

چندین گردش کار رایج وجود دارد که ممکن است برای تیم مناسب باشد. در ادامه، ما با دو مورد آشنا می شویم.

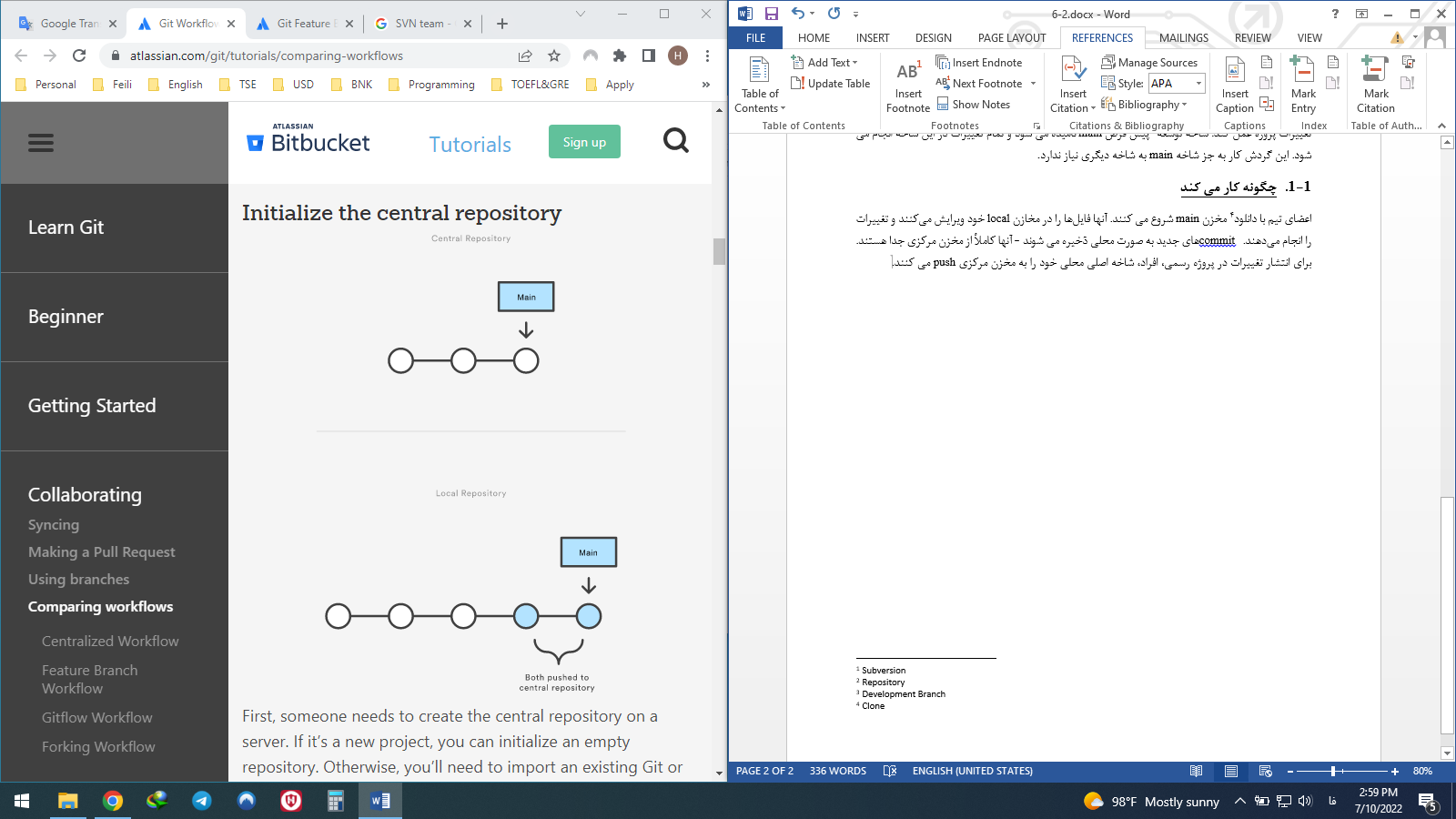
**1. گردش کار متمرکز (Centralized Workflow)**



یک گردش کار متمرکز معمولاً برای تیم هایی که از SVN[[1]](#footnote-1)به Git مهاجرت کردند و همچنین تیم های کوچک مناسب است. گردش کار متمرکز از یک مخزن[[2]](#footnote-2) مرکزی استفاده می کند تا به عنوان نقطه ورودی واحد برای همه تغییرات پروژه عمل کند. شاخه توسعه[[3]](#footnote-3) پیش فرض main نامیده می شود و تمام تغییرات در این شاخه انجام می شود. این گردش کار به جز شاخه main به شاخه دیگری نیاز ندارد.

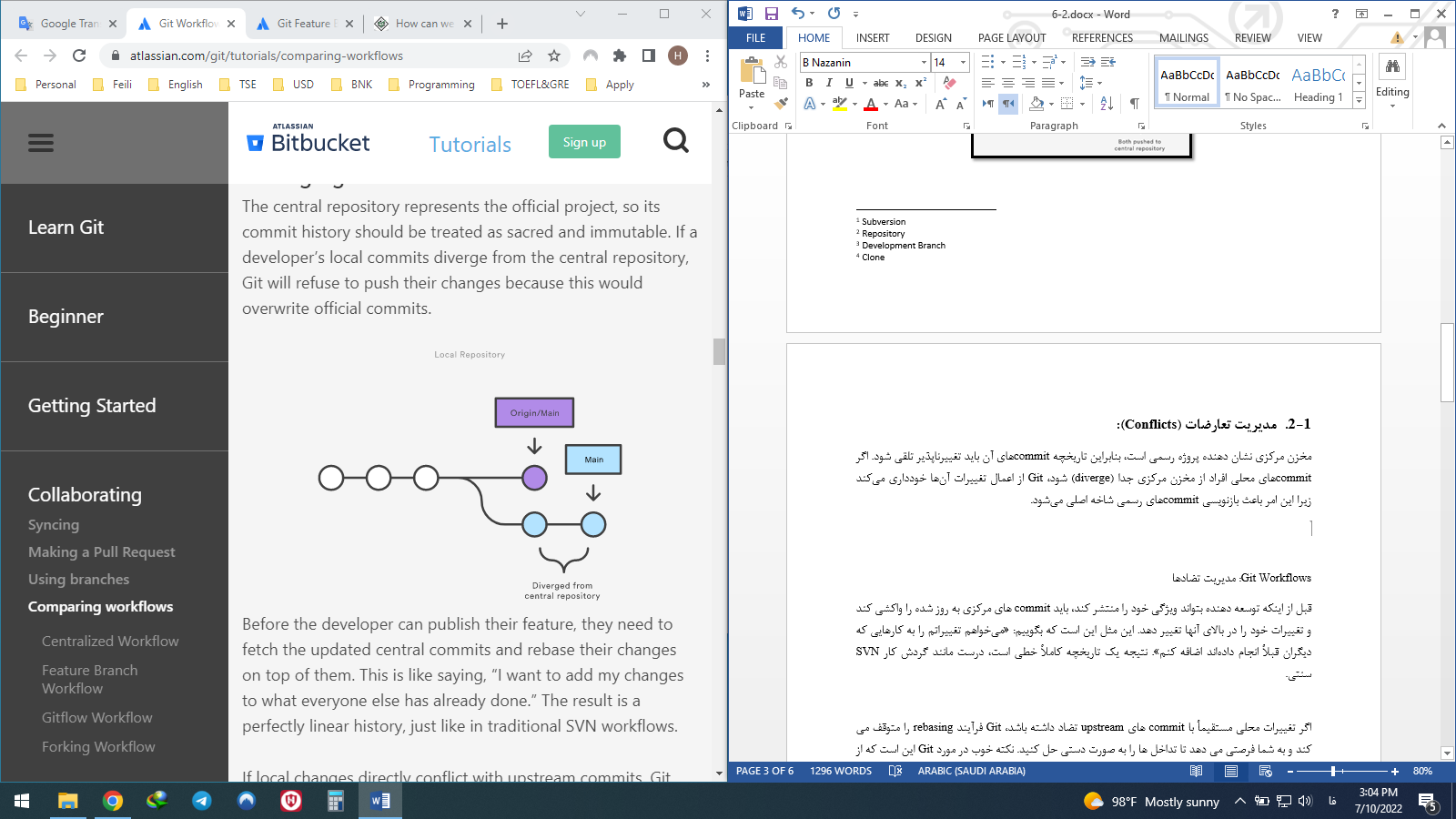
* 1. **چگونه کار می کند**

اعضای تیم با دانلود[[4]](#footnote-4) مخزن main شروع می کنند. آنها فایل‌ها را در مخازن local خود ویرایش می‌کنند و تغییرات را انجام می‌دهند. commit های جدید به صورت محلی ذخیره می شوند - آنها کاملاً از مخزن مرکزی جدا هستند. برای انتشار تغییرات در پروژه رسمی، افراد، شاخه اصلی محلی خود را به مخزن مرکزی push می کنند.



* 1. **مدیریت تداخلات (Conflicts):**

مخزن مرکزی نشان دهنده پروژه رسمی است، بنابراین تاریخچه commitهای آن باید تغییرناپذیر تلقی شود. اگر commitهای محلی افراد از مخزن مرکزی جدا (diverge) شود، Git از اعمال تغییرات آن‌ها خودداری می‌کند زیرا این امر باعث بازنویسی commit‌های رسمی شاخه اصلی می‌شود.



قبل از اینکه فرد بتواند کار خود را منتشر کند، باید commitهای مرکزی به روز شده را fetch کند و تغییرات خود را بالای آنها اعمال کند. یعنی بگوییم: «می‌خواهم تغییراتم را به کارهایی که دیگران قبلاً انجام داده‌اند اضافه کنم» که نتیجه آن یک تاریخچه کاملاً خطی است.

اگر تغییرات local مستقیماً با commitهای upstream تضاد داشته باشد، Git فرآیند rebasing را متوقف می کند و به شما فرصتی می دهد تا تداخل ها را به صورت دستی حل کنید. نکته خوب در مورد Git این است که از همان status git و دستورات git add برای ایجاد commit و حل تداخلات استفاده می کند. این امر مدیریت ادغام ها برای افراد جدید آسان می کند.

* 1. **مثال از گردش کار متمرکز**

یک مثال کلی در مورد نحوه همکاری یک تیم کوچک معمولی با استفاده از این گردش کار را مشاهده خواهیم کرد.

John روی feature خود کار می کند و می تواند در مخزن local خود ویژگی‌هایی را با استفاده از فرآیند Git commit توسعه دهد: edit، stage، و commit. به یاد داشته باشید از آنجایی که این دستورات commit های محلی را ایجاد می کنند، جان می تواند این فرآیند را هر چند بار که بخواهد بدون نگرانی در مورد آنچه در مخزن مرکزی می گذرد تکرار کند.

هم چنین Mary در حال کار بر روی feature در مخزن محلی شخصی با استفاده از همان فرآیند edit، stage، و commit است. او مانند John، اهمیتی نمی‌دهد که در مخزن مرکزی چه می‌گذرد، و واقعاً برایش مهم نیست که John در مخزن محلی‌اش چه می‌کند، زیرا همه مخازن محلی خصوصی هستند.

پس از آن John ویژگی خود را منتشر می کند. هنگامی که جان ویژگی خود را تمام کرد، باید commitهای محلی خود را در مخزن مرکزی منتشر کند تا سایر اعضای تیم بتوانند به آن دسترسی داشته باشند. او می تواند این کار را با دستور git push انجام دهد. از آنجایی که مخزن مرکزی از زمانی که John آن را دریافت کرده است، به‌روزرسانی نشده است، این انتشار منجر به هیچ تداخلی نمی‌شود و push همانطور که انتظار می‌رود کار خواهد کرد.

حال Mary سعی می کند feature خود را منتشر کند اما از آنجایی که تاریخچه محلی او از مخزن مرکزی جدا شده است، Git با یک پیام خطای نسبتاً واضح درخواست را رد می کند:

error: failed to push some refs to '/path/to/repo.git'  
hint: Updates were rejected because the tip of your current branch is behind  
hint: its remote counterpart. Merge the remote changes (e.g. 'git pull')  
hint: before pushing again.  
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

این مشکل Mary را از ارسال commit رسمی باز می دارد. او باید به‌روزرسانی‌های John را در مخزن خود بیاورد، آنها را با تغییرات محلی خود ادغام کند و سپس دوباره امتحان کند.

حال Mary باید commitهای خودر را در بالای تعهد(های) John اعمال کند. Mary می تواند با دستور زیر برای گنجاندن تغییرات بالادستی در مخزن خود استفاده کند:

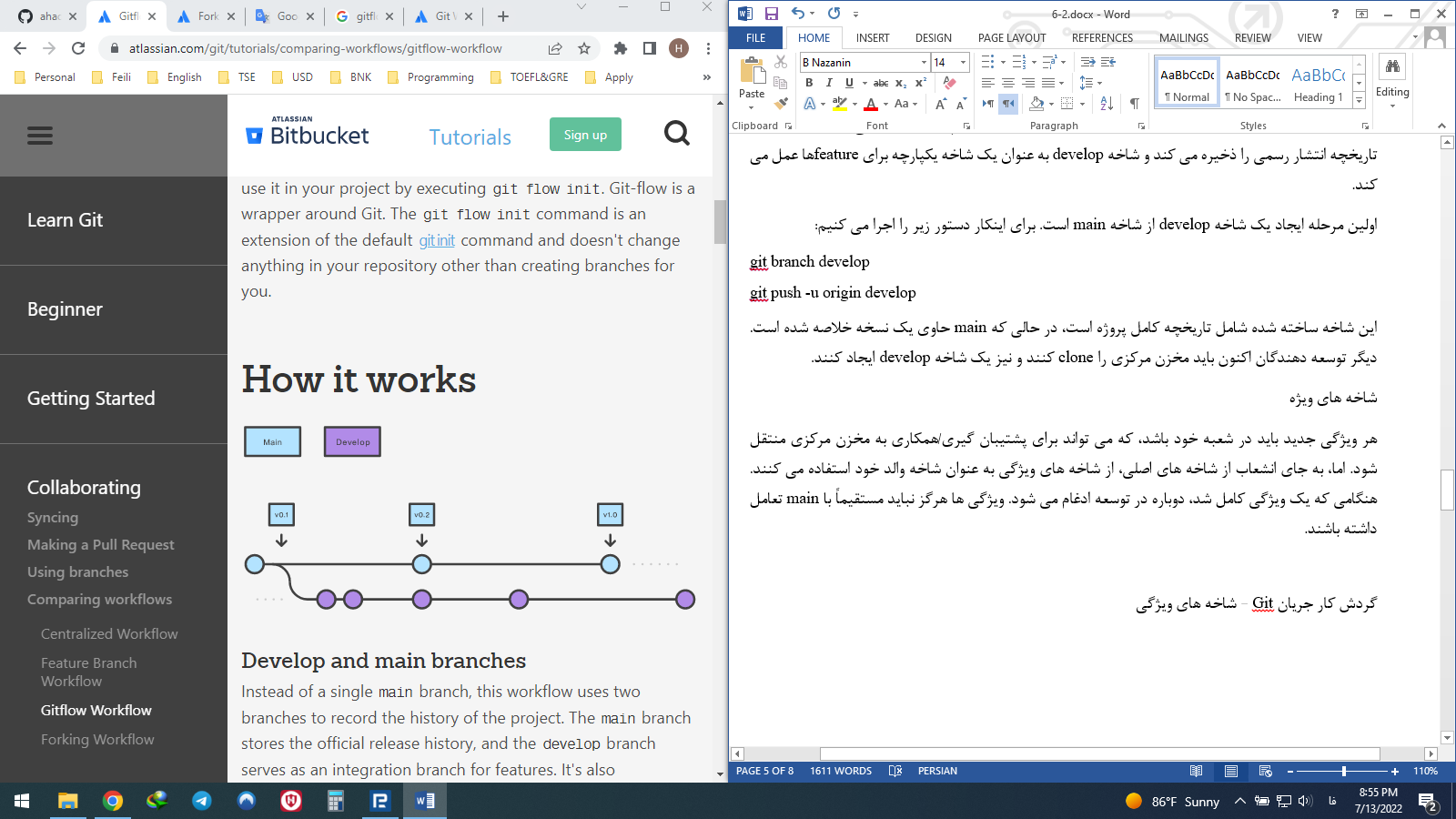
git pull --rebase origin main

عبارت rebase به Git می‌گوید که تمام commitهای Mary را پس از همگام‌سازی با تغییرات مخزن مرکزی به ابتدای شاخه اصلی منتقل کند. اگر عبارت rebase را فراموش کنید، pull همچنان کار می کند، اما هر بار که نیاز به همگام سازی با مخزن مرکزی باشد، با یک "merge commit" اضافی مواجه می شوید. برای این نوع گردش کار، همیشه بهتر است به جای ایجاد یک "merge commit" ، rebase کنید.

**2. گردش کار Feature Branch**

ابتدا باید gitflow نصب گردد. برای اینکار در سیستم‌های OSX، باید git-flow install brew را اجرا کنید. در ویندوز باید git-flow را دانلود و نصب کنید. پس از نصب git-flow می توانید با اجرای git flow init از آن در پروژه خود استفاده کنید. این دستور چیزی را در مخزن شما تغییر نمی‌دهد به جز ایجاد شاخه‌ها در آن.

به جای یک شاخه main، این گردش کار از دو شاخه برای ثبت تاریخچه پروژه استفاده می کند. شاخه main تاریخچه انتشار رسمی را ذخیره می کند و شاخه develop به عنوان یک شاخه یکپارچه برای featureها عمل می کند.



اولین مرحله ایجاد یک شاخه develop از شاخه main است. برای اینکار دستور زیر را اجرا می کنیم:

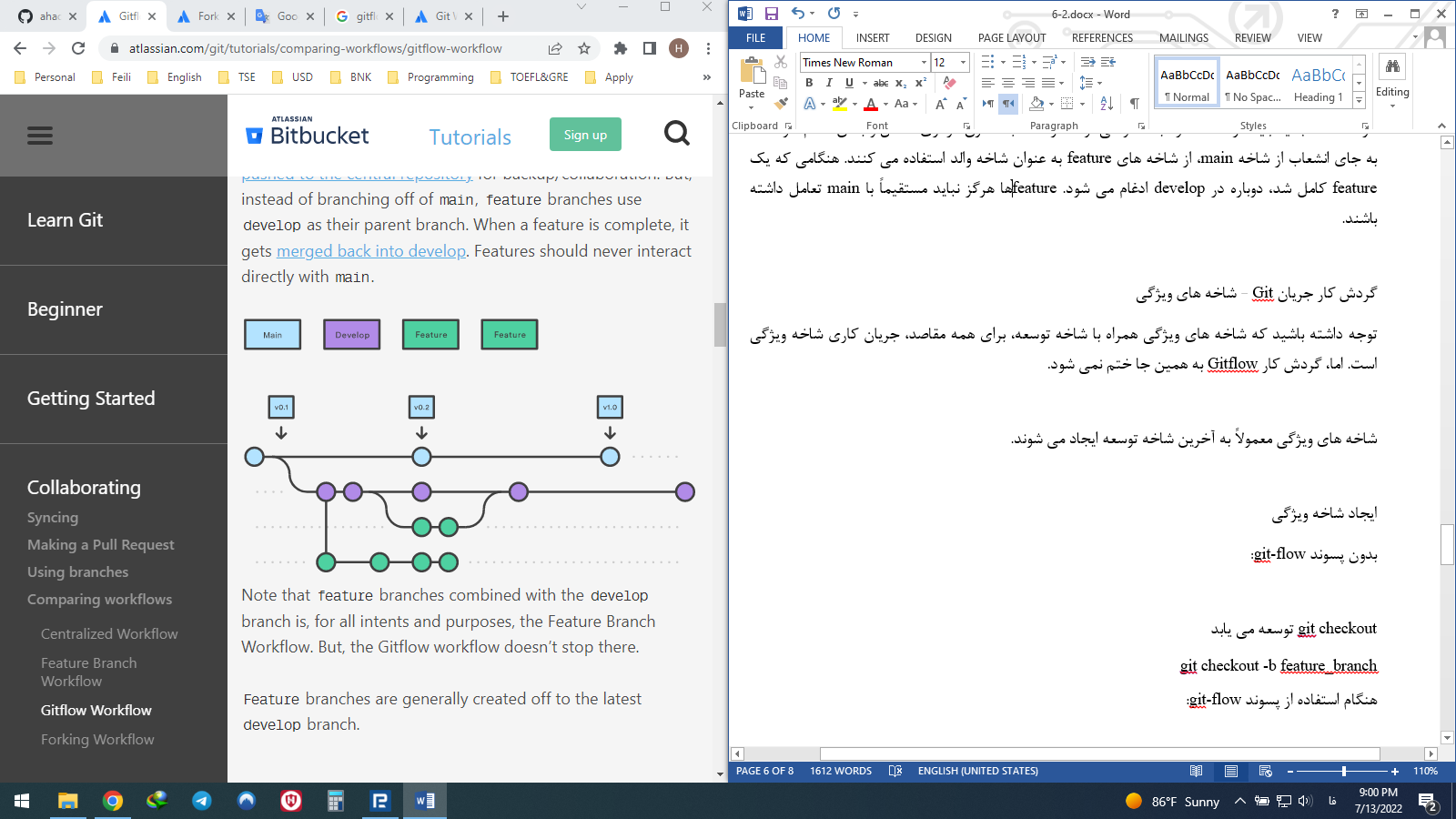
git branch develop

git push -u origin develop

این شاخه ساخته شده شامل تاریخچه کامل پروژه است، در حالی که main حاوی یک نسخه خلاصه شده است. دیگر توسعه دهندگان اکنون باید مخزن مرکزی را clone کنند و نیز یک شاخه develop ایجاد کنند.

**2.1- شاخه های Feature**

هر feature جدید باید در شاخه خود باشد، و می تواند در ادامه به مخزن مرکزی منتقل و با آن ادغام شود. اما، به جای انشعاب از شاخه main، از شاخه های feature به عنوان شاخه والد استفاده می کنند. هنگامی که یک feature کامل شد، دوباره در develop ادغام می شود. featureها هرگز نباید مستقیماً با main تعامل داشته باشند.



توجه داشته باشید که شاخه های feature همراه با شاخه develop، بسته به هدف می توانند به سرانجام نرسند و متوقف شوند. اما، گردش کار اصلی Gitflow ادامه خواهد داشت. همچنین شاخه های feature معمولاً از آخرین شاخه develop ایجاد می شوند.

**2.2- ایجاد شاخه feature**

باید در git-flow عبارت زیر را اجرا کرد:

git checkout develop  
git checkout -b feature\_branch

در صورت استفاده از extension git-flow:

git flow feature start feature\_branch

**2.3- سرانجام شاخه feature**

وقتی کار روی یک feature با اتمام می رسد، گام بعدی ادغام شاخه آن در شاخه develop است. برای این کار عبارت زیر را اجرا میکنیم:

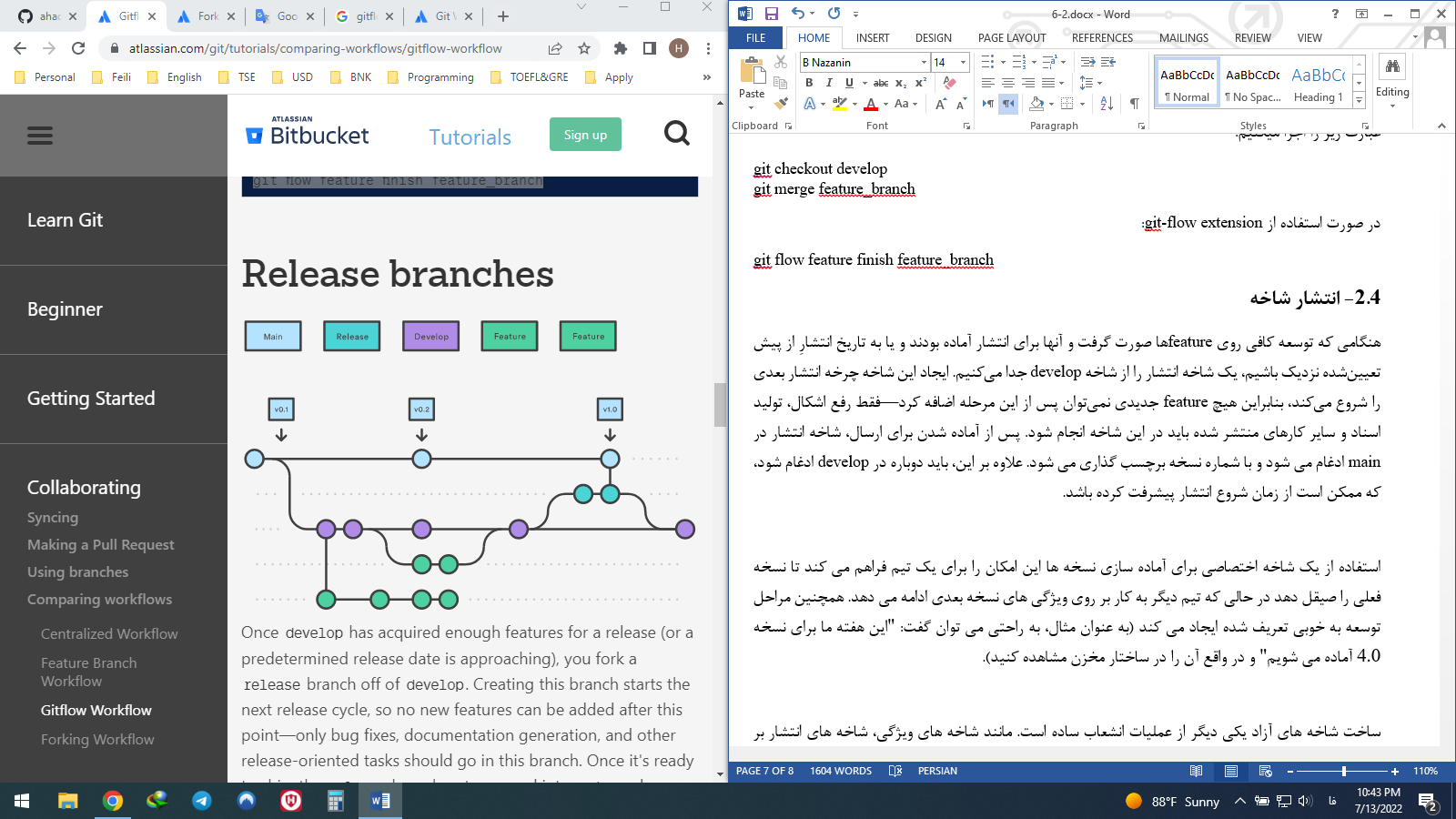
git checkout develop  
git merge feature\_branch

در صورت استفاده از extension git-flow:

git flow feature finish feature\_branch

**2.4- انتشار شاخه**

هنگامی که توسعه کافی روی featureها صورت گرفت و آنها برای انتشار آماده بودند و یا به تاریخ انتشارِ از پیش تعیین‌شده نزدیک باشیم، یک شاخه انتشار را از شاخه develop جدا می‌کنیم. ایجاد این شاخه چرخه انتشار بعدی را شروع می‌کند، بنابراین هیچ feature جدیدی نمی‌توان پس از این مرحله اضافه کرد—فقط رفع اشکال، تولید اسناد و سایر کارهای منتشر شده باید در این شاخه انجام شود. پس از آماده شدن برای ارسال، شاخه انتشار در main ادغام می شود و با شماره نسخه برچسب گذاری می شود. علاوه بر این، باید دوباره در develop ادغام شود، که ممکن است از زمان شروع انتشار پیشرفت کرده باشد.



استفاده از یک شاخه اختصاصی برای آماده سازی نسخه ها این امکان را برای یک تیم فراهم می کند تا نسخه فعلی را چکش کاری کند، در حالی که تیم دیگر به کار بر روی feature های نسخه بعدی ادامه می دهد. ساخت شاخه های انتشار یکی دیگر از عملیات انشعاب ساده است. مانند شاخه های feature، شاخه های انتشار بر اساس شاخه develop هستند. با استفاده از روش های زیر می توان یک شاخه انتشار جدید ایجاد کرد.

git checkout develop  
git checkout -b release/0.1.0

در صورت استفاده از extension git-flow:

$ git flow release start 0.1.0  
Switched to a new branch 'release/0.1.0'

هنگامی که نسخه مورد نظر آماده ارسال شد، آن را در main ادغام کرده و توسعه می دهد، سپس شاخه انتشار حذف می شود. حال برای تکمیل شاخه انتشار، از روش های زیر استفاده کنید:

git checkout main  
git merge release/0.1.0

در صورت استفاده از extension git-flow:

git flow release finish '0.1.0'

**2.5- شاخه Hotfix**

از شاخه های hotfix برای وصله و بررسی مشکلات نسخه های تولید شده استفاده می شود. شاخه های Hotfix بسیار شبیه به شاخه های انتشار و شاخه های feature هستند با این تفاوت که آنها به جای مشتق شدن از develop از main جدا می شوند. این تنها شاخه ای است که باید مستقیماً از main جدا شود. به محض اینکه مشکلات مرتفع شد، باید آن را در دو بخش اصلی و توسعه (یا شاخه نسخه فعلی) ادغام کنید، و main باید با شماره نسخه به روز شده برچسب گذاری شود.

داشتن یک خط توسعه اختصاصی برای رفع اشکال به تیم شما اجازه می‌دهد بدون وقفه در بقیه جریان کار یا انتظار برای چرخه انتشار بعدی، به مشکلات رسیدگی کنید. می توانید شاخه های hotfix را به عنوان شاخه های انتشار موقتی در نظر بگیرید که مستقیماً با main کار می کنند. یک شاخه Hotfix را می توان با استفاده از روش های زیر ایجاد کرد:

git checkout main  
git checkout -b hotfix\_branch

در صورت استفاده از extension git-flow:

$ git flow hotfix start hotfix\_branch

مشابه به پایان رساندن یک شاخه انتشار، یک شاخه Hotfix به دو بخش اصلی و توسعه ادغام می شود.

git checkout main  
git merge hotfix\_branch  
git checkout develop  
git merge hotfix\_branch  
git branch -D hotfix\_branch

$ git flow hotfix finish hotfix\_branch

1. Subversion [↑](#footnote-ref-1)
2. Repository [↑](#footnote-ref-2)
3. Development Branch [↑](#footnote-ref-3)
4. Clone [↑](#footnote-ref-4)