گردش کار Gitflow یک گردش کار در Git است که به توسعه ی مداوم نرم افزار و پیاده سازی روندهای DevOpsکمک می کند. اولین بار وینسنت دریسن(Vincent Driessen) ، این گردش کار را مطرح کرد و جامعه برنامه نویسان، به صورت گسترده استفاده کردند. گردش کار Gitflow یک روش شاخه گرفتن سختگیرانه را تعریف می کند که حول انتشار محصول تعریف شده است. این امر یک چهارچوب مقاوم برای مدیریت پروژه های بزرگ تر را برای ما فراهم می کند. شما می توانید در مقاله ی ۴ گردش کار درGit ، مطالب بیشتری در مورد این گردش کار، در سکان آکادمی بخوانید.

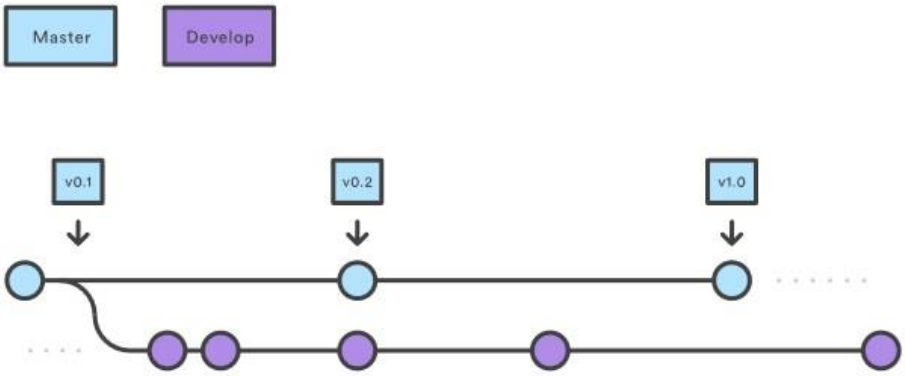
Gitflowبرای پروژه هایی که چرخه ی انتشار محصول برنامه ریزی شده ای دارند و همچنین برای پیاده سازی روند تحویل مداوم- (Continuous Delivery) که یک best practiceدر حوزه ی DevOps است- بسیار مناسب است. این گردش کار به هیچ مفهوم یا دستور جدیدی فراتر از آنچه برای گردش کار Feature Branch مورد نیاز است، نیاز ندارد Gitflow .به جای اضافه کردن مفهوم جدید، به هر شاخه، یک نقش متفاوت می دهد و مشخص می کند که این شاخه ها کی و چگونه باید با یکدیگر تعامل داشته باشند. علاوه بر شاخه یfeature ، گردش کار Gitflow از شاخه های مختلفی برای آماده سازی، نگهداری و ثبت نسخه های منتشر شده استفاده می کند. در عین حال شما می توانید از تمام ویژگی های گردش کار شاخه ی feature مثل درخواستpull ، کار کردن مستقل توسعه دهندگان و همکاری بهینه تر بین توسعه دهندگان بهره ببرید.

آماده سازی

Gitflowدر واقع فقط یک ایده ی انتزاعی از گردش کار در Gitاست. این گردش کار مشخص می کند که چه نوع شاخه هایی ساخته شود و چگونه این شاخه ها را با هم merge کنیم. ما در این مقاله اهداف این شاخه ها را شرح خواهیم داد. مجموعه ابزار git-flowیک ابزار خط فرمان واقعی است که به نصب نیاز دارد. فرآیند نصب git-flowآسان است و پکیج های این ابزار برای سیستم عامل های مختلفی در دسترس است. در سیستم های OSX شما می توانید از دستور brew install git-flowاستفاده کنید. همچنین در ویندوز شما می توانید آن را دانلود و نصب کنید. پس از نصب git-flowمی توانید از آن در پروژه های خود با اجرای دستور git flow initاستفاده کنید.

Git-flowیک ابزار توسعه داده شده روی Git است و دستور git flow initیک افزونه از دستور اصلی git initاست که تنها تغییری که در مخزن شما ایجاد می کند ساختن شاخه ها برای شما است.

گردش کار Gitflow چگونه کار می کند؟



شاخه های developوmaster

به جای یک شاخه یmaster ، این گردش کار از دو شاخه برای ثبت تاریخچه ی پروژه استفاده می کند. شاخه ی master تاریخچه ی نسخه های رسمی پروژه را نشان می دهد که به صورت رسمی عرضه شده اند. شاخه ی develop به عنوان یک بستر برای گرد هم آوریfeature ها عمل می کند. شاخه ی develop همچنین این امکان را به ما می دهد که به تمام commit ها شماره ی نسخه ی محصول را به عنوان برچسب اختصاص دهیم.اولین قدم، ایجاد شاخه ی develop است. یک راه آسان برای توسعه دهنده، ایجاد یک شاخه ی خالی با نام develop در Git خود و push کردن آن به سرور است. این کار با دستور زیر قابل انجام است:

```
git branch develop
git push -u origin develop
```

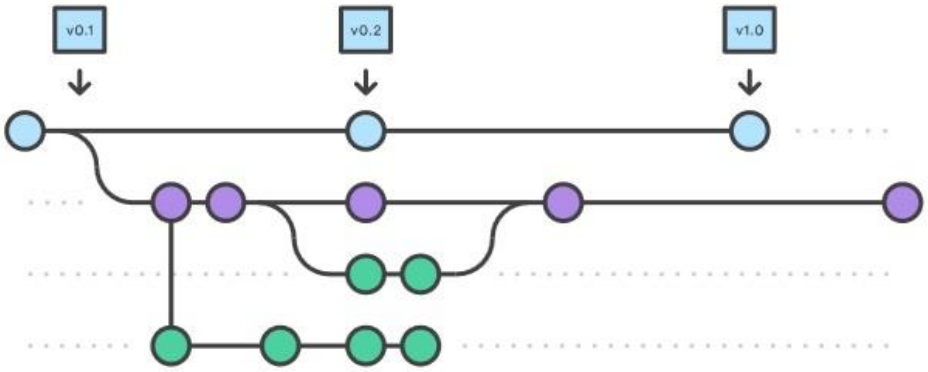
در حالی که شاخه ی master نسخه ی خلاصه شده ای از تاریخچه ی پروژه را نگهداری می کند، این شاخه تمام تاریخچه ی پروژه را شامل خواهد شد. اکنون توسعه دهنده های دیگر باید ازمخزن مرکزی، cloneبگیرند و یک شاخه ی قابل ردیابی برای develop ایجاد کنند:

```
$ git flow init
Initialized empty Git repository in ~/project/.git/
No branches exist yet. Base branches must be created now.
Branch name for production releases: [master]
Branch name for "next release" development: [develop]
How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
$ git branch
* develop
```

master

شاخه های feature

هر feature جدید می بایست در شاخه ی خود قرار داشته باشد. این شاخه می تواند به مخزن مرکزی برای تهیه ی نسخه پشتیبان (backup) یا به منظور همکاری با توسعه دهنده های دیگر، pushشود؛ اما به جای ایجاد شاخه ی جدید ازmaster ، شاخه های feature از شاخه ی develop به عنوان پدر استفاده می کنند. هنگامی که یک feature کامل می شود، در شاخه ی develop بابقیه merge می شود. شاخه های feature هرگز نباید به صورت مستقیم با شاخه ی master ارتباط داشته باشند.



شاخه های feature همراه با شاخه ی develop همان گردش کار Feature Branch را تداعی می کند؛ اما گردش کار Gitflow همین جا متوقف نمی شود شاخه های feature به طور معمول، از به روزترین نسخه ی شاخه ی develop گرفته می شوند.

ساخت یک شاخه یfeature

با استفاده از دستور زیر می توان یک شاخه ی feature جدید ساخت:

```
git checkout develop
git checkout -b feature_branch
```

اگر از افزونه ی git-flowاستفاده می کنید امکان استفاده از دستور زیر را نیز دارید:

```
git flow feature start feature_branch
```

اکنون می توانید مثل قبل با Git کار کنید.

پایان یک شاخه یfeature

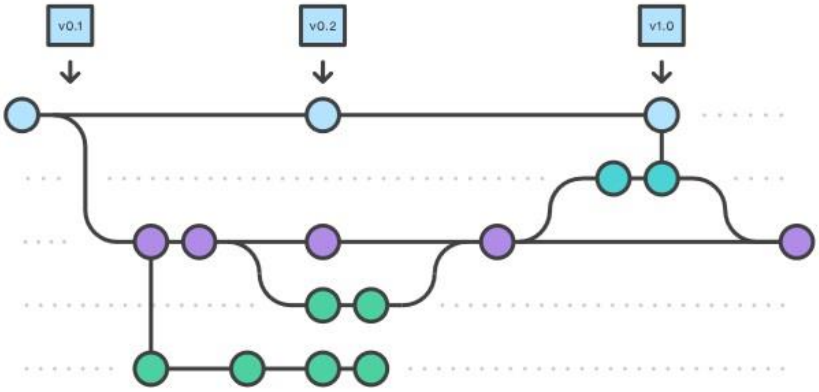
وقتی که توسعه ی یک feature به اتمام می رسد، گام بعدی merge شاخه ی feature_branch در شاخه ی develop است.این کار با دستور زیر قابل انجام شدن است:

```
git checkout develop
git merge feature_branch
```

همچنین با افزونه ی git-flowمی توانید از دستور زیر استفاده کنید:

```
git flow feature finish feature_branch
```

شاخه های انتشار(Release Branches)



هنگامی که شاخه یdevelop, feature های کافی را برای انتشار نسخه جدید محصول کسب کرد (یا تاریخ انتشار از قبل مشخص شده، نزدیک است)، شما یک شاخه ی انتشار از شاخه ی develop می گیرید. ایجاد این شاخه چرخه ی انتشار جدیدی را آغاز می کند، پس هر feature جدیدی باید بعد از انتشار اضافه شود. در این مرحله فقط رفع نقص، تهیه ی مستند پروژه و باقی کار های مربوط به انتشار محصول در این شاخه انجام می شوند. وقتی که محصول آماده ی ارائه است، شاخه ی انتشار در شاخه یmaster ، ادغام می شود و برچسب شماره ی نسخه روی آن می خورد. علاوه بر این باید شاخه ی انتشار در شاخه ی develop نیز merge شود.

استفاده از یک شاخه ی مجزا برای انتشار محصول این امکان را می دهد که در حالی که یک تیم رویfeature های دیگر برای انتشار بعدی کار می کنند، تیم دیگری این نسخه را برای انتشار نهایی آماده کنند. این کار همچنین فازبندی مراحل توسعه محصول را واضح و روشن می سازد.ساخت یک شاخه ی انتشار نیز مانند ساخت هر شاخه ی دیگری ساده است. این شاخه از شاخه ی develop گرفته می شود.

```
git checkout develop
git checkout -b release/0.1.0
```

همچنین این کار با استفاده از افزونه ی git-flowبه صورت زیر انجام می شود:

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

شاخه ی انتشار پس از merge در master و develop پاک می شود .merge شاخه ی انتشار در شاخه ی develop بسیار مهم است، زیرا ممکن است به روزرسانی های حیاتی ای به شاخه ی انتشار اضافه شده باشند و این تغییرها می بایست در اختیارfeature های جدید قرار داشته باشند. اگر سازمان شما به بررسی و مرور کد، اهمیت می دهد، بهترین موقعیت برای درخواست pull اینجا است.

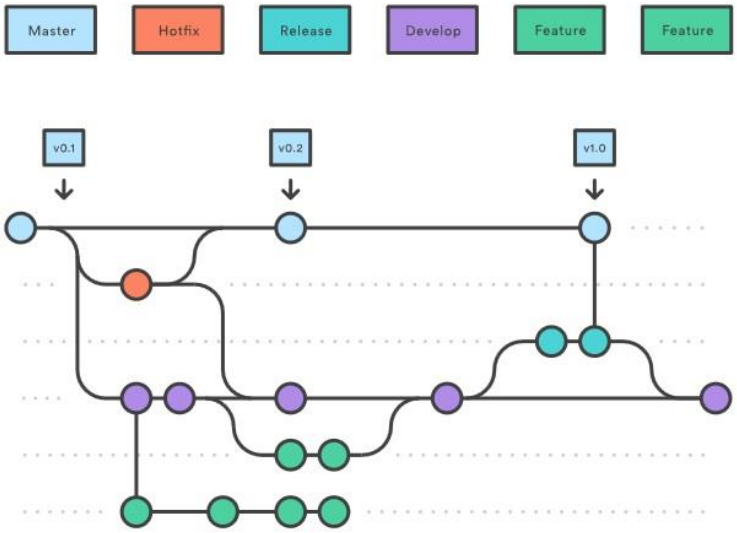
برای اتمام شاخه ی انتشار از دستورهای زیر استفاده کنید:

```
git checkout master
git merge release/0.1.0
```

دستور بالا با استفاده از افزونه ی git-flowبه صورت زیر در می آید:

```
git flow release finish '0.1.0'
```

شاخه هایHotfix



شاخه های hotfix یا نگهداری، برای رفع سریع نقص های نسخه های منتشر شده استفاده می شوند. شاخه های hotfix بسیار به شاخه های انتشار شبیه اند با این تفاوت که شاخه های hotfix به جای develop از شاخه ی master گرفته می شوند. این شاخه ها تنها شاخه ای هستند که باید به طور مستقیم از شاخه ی master گرفته شوند. به محض اینکه رفع نقص انجام شد این شاخه باید در هر دو شاخه ی master وdevelop، با هم merge شود و شاخه ی master باید یک برچسب با شماره نسخه ی جدید دریافت کند.

داشتن یک روند جدا برای رفع نقص ها به تیم شما این امکان را می دهد تا بدون مزاحمت برای سایر تیم ها، مشکل ها را حل کنند یا تا چرخه ی انتشار محصول بعدی صبر کنند.

یک شاخه ی hotfix با دستورهای زیر قابل ایجاد است:

```
git checkout master
git checkout -b hotfix_branch
```

همچنین با استفاده از افزونه ی git-flowمی توانید از دستور زیر استفاده کنید:

```
$ git flow hotfix start hotfix_branch
```

مشابه با شاخه ی انتشار، این شاخه، پس از پایان در هر دو شاخه ی develop وmaster با هم merge می شود:

```
git checkout master
git merge hotfix_branch
git checkout develop
git merge hotfix_branch
git branch -D hotfix_branch
```

با استفاده از افزونه ی :git-flow

```
$ git flow hotfix finish hotfix_branch
```

یک نمونه از گردش کار

یک مثال کامل از گردش کار Feature Branch به صورت زیر است. فرض می کنیم یک مخزن با شاخه ی master ساخته شده است.

```
git checkout master
git checkout -b develop
git checkout -b feature_branch
# work happens on feature branch
git checkout develop
git merge feature_branch
git checkout master
git merge develop
git branch -d feature_branch
```

علاوه بر روند feature و انتشار، یک مثال hotfix به صورت زیر است:

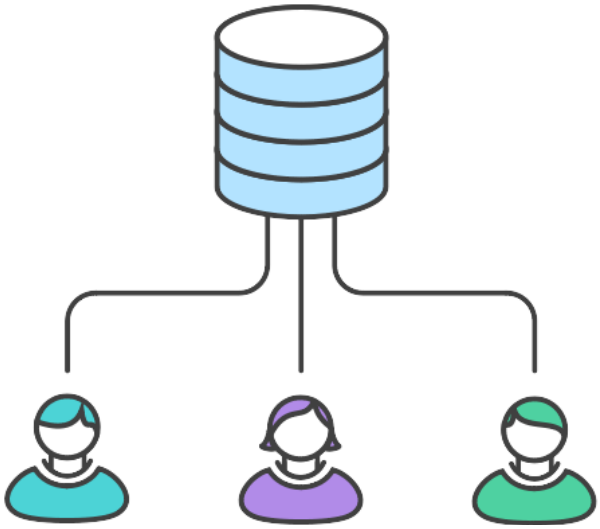
```
git checkout master
git checkout -b hotfix_branch
# work is done commits are added to the hotfix_branch
git checkout develop
git merge hotfix_branch
git checkout master
git merge hotfix_branch
```

علاوه بر امروزه افراد زیادی برای کنترل نحوه‌ی پیشبرد پروژه و مدیریت تیم خود از نرم‌افزار گیت (Git) استفاده می‌کنند. گیت روش‌های مختلف و انعطاف‌پذیری را برای مدیریت و پیشبرد پروژه‌ها فراهم آورده است. هر یک از این روش‌ها به اصطلاح گردشِ‌کار یا workflow نامیده می‌شود. به طور کلی، گردش‌کار استانداردی وجود ندارد که بتوان آن را برای همه‌ی پروژه‌ها و تیم‌ها توصیه کرد و مهم‌ترین چیز در تعیین روش گردش‌کار، توافق و همراهی همه‌ی اعضای تیم است. با این حال گیت چندین گردش‌کار پیش‌فرض را ارائه داده است که ممکن است برای تیم شما نیز مناسب بوده و بتوانید از آن بهره ببرید. در این مقاله می‌خواهیم این روش‌های پیش‌فرض را بررسی نموده و به شما در انتخاب یک گردش‌کار مناسب یاری رسانیم.توجه داشته باشیدکه معرفی این روش‌ها بیشتر با هدف ارائه‌ی یک راهنما و آشنایی با حالات ممکن پیشبرد پروژه صورت می‌گیرد و هرگز نباید به آن‌ها به عنوان دستورالعمل‌های محض و قوانین سخت نگاه کرد.

گردش‌کار موفق گیت چیست؟

در هنگام تعیین یک روش گردش‌کار، بسیار مهم است که فرهنگ تیم خود را در نظر داشته باشید. گردش‌کار قرار است عملکرد تیم شما را بهبود ببخشد نه این‌که بار مضاعفی بر دوش اعضای تیم گذاشته و مانعی در مقابل بهره‌وری آن‌ها شود. سایر موارد مهمی که باید در تعیین و انتخاب گردش‌کار مد نظر قرار دهید در سه پرسش زیر مشخص می‌شوند:آیا مقیاس گردش‌کار با اندازه‌ی تیم متناسب است؟آیا اصلاح و جبران اشتباهات در این گردش‌کار آسان است؟آیا این گردش‌کار بار روانی غیرضروری و اضافه‌ای را به تیم تحمیل نمی‌کند؟

گردش‌کار متمرکز(Centralized)



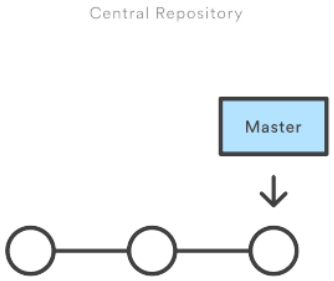
گردش‌کار متمرکز یک گردش‌کار عالی برای تیم‌هایی است که به تازگی از SVN به گیت آمده‌اند. در این روش مشابه آنچه در SVN صورت می‌گرفت، یک مخزن مرکزی وجود دارد که تمام تغییرات پروژه از طریق آن به کد اصلی وارد می‌شوند. به جای آنچه که تحت عنوان تنه (Trunk) در SVN می‌شناسیم، در گردش کار متمرکز یک شاخه اصلی به نام مَستر (Master)در نظر گرفته شده است که تمام تغییرات پروژه بر روی آن پیاده‌سازی می‌شوند. در این گردش‌کار، به جز شاخه‌ی مستر، وجود شاخه‌های دیگر ضروری نیست.انتقال به یک سیستم ورژن کنترل توزیع‌شده می‌تواند کار دشوار و استرس‌آوری به نظر برسد اما اگر از SVN به گیت آمده‌اید، جایی برای نگرانی وجود ندارد زیرا در گردش‌کار متمرکز، تیم شما می‌تواند دقیقاً به همان روش SVN به کار خود ادامه دهد.با این حال، گیت نسبت به SVN مزایایی نیز دارد که می‌تواند گردش‌کار قدرتمندتری را برای شما فراهم آورد. نخست این‌که گیت یک کپی از کل پروژه را در اختیار هر یک از دولوپرهای تیم قرار می‌دهد. به این ترتیب هر دولوپر می‌تواند فارغ از تغییرات بالادستی، تغییرات مورد نظر خود را در این

مخزن محلی ذخیره و اعمال کند تا در زمان مناسب به نسخه‌ی اصلی پروژه اضافه شوند.دوم اینکه امکان دسترسی به مدل قدرتمند انشعاب و ادغام (branching and merging model)گیت را خواهید داشت. برخلافSVN ، شاخه‌های گیت در ادغام و اشتراک‌گذاری تغییرات بین مخزن‌ها از ساز و کار Fail-safe بهره می‌برند Fail-safe .ساز و کاری است که طی آن در صورتی که بخشی از پروژه به درستی کار نکند، سایر بخش‌ها در کمترین حد ممکن با این مشکل درگیر می‌شوند و یا در حالت آرمانی هیچ تأثیری از آن نمی‌پذیرند.گردش‌کار متمرکز در استفاده از مخزن میزبان که دولوپرها فرآیندهای push و pull را در آن انجام می‌دهند با سایر گردش‌کارها مشابه است اما در مقایسه با آن‌ها الگوهای درخواست pull و forking تعریف شده‌ای ندارد. همان‌طور که پیش از این اشاره شد، گردش‌کار متمرکز بیشتر برای تیم‌هایی که از SVN به گیت مهاجرت کرده‌اند و تیم‌های کوچک‌تر مناسب است.

نحوه‌ی کار گردش‌کار متمرکز

برای شروع کار هر دولوپر باید یک کپی محلی از پروژه‌ی اصلی را برای خود ایجاد کند. به این کار تکثیر کردن (Cloning) می‌گویند. هر یک از دولوپرها می‌تواند در کپی مخصوص خود، تغییرات مورد نظرش را اعمال نماید، درست مانند آنچه که در SVN صورت می‌گرفت. این تغییرات به طور محلی در کپی‌هایی که در اختیار دولوپرهای تیم است (یعنی در مخزن‌های محلی آن ها) ذخیره می‌شود و اعمال آن‌ها بر روی نسخه‌ی اصلی پروژه تا هر زمانی که بخواهند به تعویق می‌افتد.برای اعمال تغییرات بر روی پروژه‌ی اصلی، هر یک از دولوپرها باید شاخه‌ی مستر مخزن محلی خود را به مخزن مرکزی پروژه ارسال یا به اصطلاح push کند. این عمل معادل Commit در SVN است با این تفاوت که فقط مواردی که پیش از این در شاخه‌ی مستر مرکزی وجود نداشته‌اند را پیاده‌سازی می‌کند. در ادامه به شرح دقیق‌تر این گردش‌کار می‌پردازیم.

ایجاد و مقداردهی اولیه‌ی مخزن مرکزی



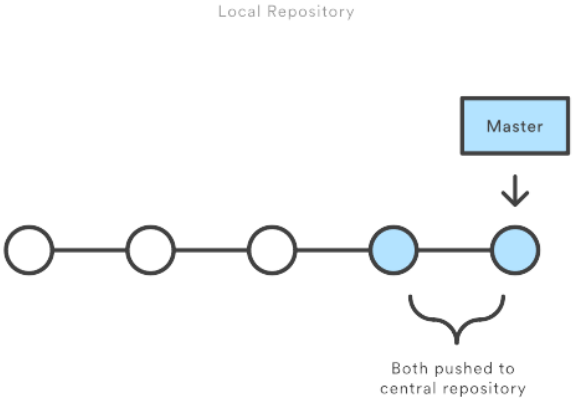
برای شروع کار، در مورد پروژه‌ی جدید می‌توانید یک مخزن خالی ایجاد کنید تا در آینده فایل‌های پروژه را به آن اضافه کنید. در غیر این صورت باید مخزن گیتی که از قبل موجود است یا SVN پروژه‌ی خود را وارد نمایید. مخازن مرکزی حتماً باید از نوع bare باشند (و روی فایل‌های موجود در آن به طور مستقیم کاری انجام نشود). با دستور زیر می توان یک مخزن bare ایجاد کرد:

```
ssh user@host git init --bare /path/to/repo.git
```

دقت کنید که در این دستور نام کاربری(ssh user) ، آی‌پی سرور میزبان (host) مورد نظر خود و آدرس ایجاد و ذخیره مخزن (path/to/repo.git) را به درستی وارد کنید. توجه داشته باشید که پسوند git. باید بلافاصله بعد از نام مخزن بیاید تا مشخص شود که مخزن از نوع bare است.

مخزن مرکزی میزبانی شده

مخزن‌های مرکزی اغلب از طریق سرویس‌های شخص ثالث میزبانی گیت مانند Bitbucket Cloud یا Bitbucket Serverایجاد می‌شوند. فرآیند ایجاد مخزن bare که در بالا توضیح داده شد نیز توسط همین سرویس میزبانی انجام می‌شود. پس از آن، سرویس میزبانی آدرسی را برای شما ایجاد می‌کند تا بتوانید از مخزن محلی خود به مخزن مرکزی دسترسی داشته باشید.



تکثیر (Clone) مخزن مرکزی

در مرحله‌ی بعد، هر دولوپر باید یک کپی از کل پروژه را برای خود ایجاد کند. این مرحله تکثیر یا clone کردن نام دارد و از طریق دستور git clone انجام می‌شود:

```
git clone ssh://user@host/path/to/repo.git
```

همزمان با تکثیر یک مخزن، گیت با فرض این‌که بعدها قرار است تعامل بیشتری با مخزن مادر داشته باشید، به صورت خودکار میانبری تحت عنوان origin را برای شما ایجاد می‌کند.

ایجاد و اعمال تغییرات

هر دولوپر در مخزن محلی خود می‌تواند طبق روند استاندارد گیت، تغییراتی را به وجود آورد. این روند استاندارد سه مرحله‌ی ویرایش(edit) ، نمایش (stage) و ثبت (commit) را شامل می‌شود. برای انجام این مراحل از دستورات زیر استفاده کنید:

```
git status # View the state of the repo
git add <some-file> # Stage a file
git commit # Commit a file</some-file>
```

دستور git status اطلاعات پروژه را برای شما نمایش می‌دهد. دستور git add یک تغییر ایجاد شده را به مرحله‌ی نمایش می‌فرستد. مرحله‌ی نمایش به منزله‌ی ثبت موقت یک تغییر است بدون این‌که نیاز باشد تمام تغییرات محلی ایجاد شده را در فایل ثبت نمایید. با دستور git commit می‌توانید تغییری که ایجاد نموده‌اید را بر روی پروژه ثبت کنید.از آنجا که این دستورات در مخزن محلی (و نه مخزن مرکزی) اعمال می‌شوند، بدون هیچ‌گونه نگرانی می‌توانید هرچند بار که لازم باشد آن‌ها را تکرار کنید. به عنوان مثال اگر بخواهید یک فیچر بزرگ را ویرایش کنید، با استفاده از این ویژگی می‌توانید آن را در قالب چندین بخش کوچک‌تر ویرایش نموده و تغییرات را مرحله به مرحله ایجاد نمایید.

انتقال تغییرات ثبت شده‌ی جدید به مخزن مرکزی

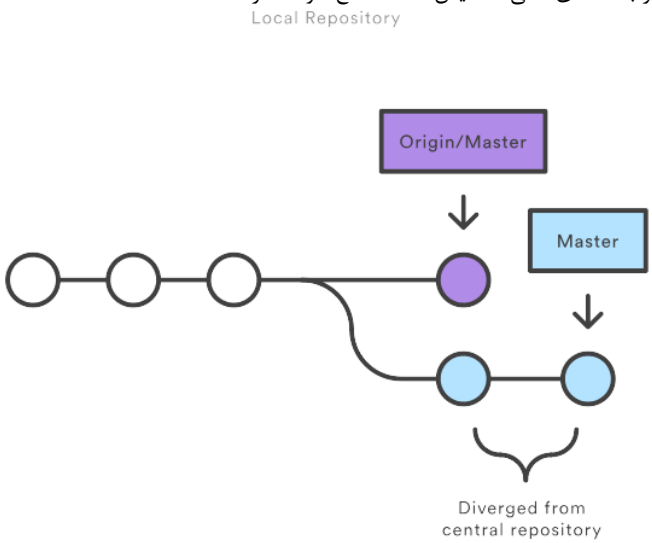
بعد از این‌که هر دولوپر تغییرات مورد نظر را در مخزن محلی خود ثبت نمود وقت آن است که برای به اشتراک گذاشتن این تغییرات با سایر دولوپرها، آن‌ها را به مخزن مرکزی پروژه منتقل کرده یا به اصطلاح آن‌ها را پوش (Push) کند. این کار با استفاده از دستور زیر صورت می‌گیرد:

git push origin master

همان‌طور که گفتیم، با این دستور می‌توانید تغییرات ثبت شده‌ی جدید را به مخزن مرکزی پروژه منتقل کنید. اما ممکن است قبل از شما دولوپر دیگری تغییراتی را به مخزن مرکزی منتقل کرده باشد که با تغییرات ثبت شده‌ی شما مغایرت (Conflict) داشته باشد. در این صورت گیت با پیامی این مغایرت را به شما اطلاع می‌دهد. در چنین موقعیتی پیش از هر چیز دستور git pull باید اجرا شود. در ادامه به شرح دقیق‌تر این موقعیت خواهیم پرداخت.

مدیریت مغایرت‌ها

می‌توان گفت که مخزن مرکزی هر پروژه در واقع نسخه‌ی رسمی آن به حساب می‌آید و تاریخچه‌ی تغییرات ثبت شده در آن باید به نوعی مقدس و غیرقابل‌تغییر در نظر گرفته شود. پس اگر تغییرات ثبت شده‌ی محلی دولوپری با آنچه که قبلاً در مخزن مرکزی ثبت شده است در تضاد باشد، تغییرات ثبت شده‌ی قبلی به عنوان معیار در نظر گرفته می‌شوند و گیت از پذیرش و ثبت تغییرات جدیدی که آن‌ها را مغایر با نسخه‌ی فعلی تشخیص دهد، امتناع خواهد کرد.



از این روی، قبل از این‌که دولوپری تغییرات مورد نظر خود را به مخزن مرکزی انتقال دهد باید آخرین به‌روزرسانی این مخزن را دانلود نموده و تغییرات محلی خود را با نسخه‌ی به‌روزشده‌ی پروژه هماهنگ کند تا مغایرتی وجود نداشته باشد. این کار باعث می‌شود که تاریخچه‌ی تغییرات پروژه یک روند کاملاً خطی داشته باشد، درست مانند آنچه در SVN وجود داشت.اگر تغییرات جدید با تغییرات ثبت شده‌ی بالادستی در تضاد باشد، گیت روند ثبت تغییرات جدید در مخزن مرکزی را متوقف نموده و این امکان را به شما می‌دهد که در همان زمان به صورت دستی تضادها را برطرف کنید.نکته‌ی جالب در مورد گیت این است که هم برای ایجاد تغییرات و هم برای رفع مغایرت‌ها می‌توان از دستورات git status و git addاستفاده کرد. این امکان، کار مدیریت ادغام کدها را برای دولوپرها ساده‌تر می‌کند. علاوه بر این، اگر دولوپرها همزمان با انتقال تغییرات به مخزن مرکزی موفق به رفع مغایرت‌ها نشوند، این امکان برای آن‌ها فراهم است که فرآیند انتقال تغییرات را متوقف نموده و پس از رفع مشکلات دوباره اقدام به انتقال کنند.

یک مثال

در ادامه می‌خواهیم در قالب یک مثال نشان دهیم که چگونه یک تیم کوچک می‌تواند از گردش کار متمرکز استفاده نماید. در این سناریو دو دولوپر به نام‌های جواد و مریم روی فیچرهای جداگانه‌ای کار می‌کنند و در نهایت حاصل کار خود را در مخزن مرکزی به اشتراک می‌گذارند.

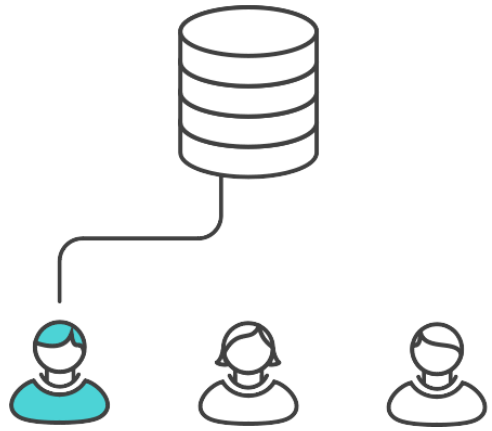
الف- جواد بر روی فیچر مورد نظر خود کار می‌کند

جواد در مخزن محلی خود، طی روند استاندارد گیت (ویرایش، نمایش و ثبت)، تغییرات مورد نظر خود را ایجاد و ثبت می‌کند. توجه کنید که این تغییرات فقط در مخزن محلی جواد (یعنی بر روی نسخه‌ای از پروژه که او در کامپیوتر خود ذخیره کرده است) اعمال می‌شود. او می‌تواند هر چند بار که نیاز باشد فرآیند ویرایش، نمایش و ثبت را تکرار کند زیرا این تغییرات فعلاً در مخزن مرکزی اعمال نمی‌شوند.

ب- مریم هم بر روی فیچر مورد نظر خود کار می‌کند

همزمان که جواد مشغول کار خود است، مریم در مخزن محلی خود مشغول کار بر روی فیچر دیگری است. او نیز از روند استاندارد ویرایش، نمایش و ثبت استفاده می‌کند و هر چند بار که نیاز باشد این روند را تکرار می‌کند. توجه داشته باشید که تا وقتی مریم و جواد در مخزن‌های محلی خود مشغول کار هستند، کار هیچ‌یک از آن‌ها هیچ تأثیری بر روی کار دیگری ندارد زیرا مخزن‌های محلی کاملاً خصوصی بوده و به جز خود دولوپر کسی به آن دسترسی ندارد.

پ- جواد فیچر مورد نظر خود را منتشر می‌کند

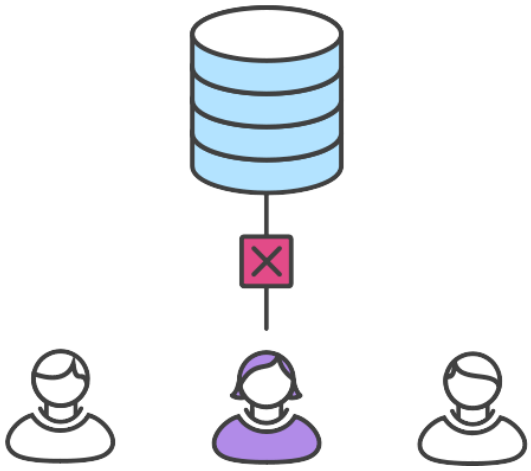


هنگامی که جواد توسعه فیچر مورد نظر خود را به پایان رساند باید آن را از مخزن محلی خود به مخزن مرکزی پروژه منتقل کند تا سایر دولوپرهای تیم نیز به آن دسترسی داشته باشند. او برای این کار می‌تواند از دستور git push به صورت زیر استفاده کند:

```
git push origin master
```

توجه داشته باشید که origin اتصال راه‌دور (remote) به مخزن مرکزی است. درست همان زمانی که جواد پروژه را برای خود به اصطلاح Clon کرد (یعنی یک کپی از پروژه را در مخزن محلی خود ایجاد کرد)، این اتصال راه‌دور هم برای دسترسی‌های بعدی توسط گیت ایجاد شد. جواد با شناسه‌ی master از گیت می‌خواهد کاری کند که شاخه‌ی مستر مخزن مرکزی شبیه شاخه‌ی مستر مخزن محلی او شود. از آنجا که قبل از جواد کسی تغییری در مخزن مرکزی ایجاد نکرده است، همان‌طور که انتظار می‌رود، تمام تغییرات مورد نظر جواد بدون هیچ مشکلی در مخزن مرکزی ثبت می‌شود.

ت- مریم سعی می‌کند فیچر مورد نظر خود را منتشر کند



پس از این‌که جواد تغییرات مورد نظر خود را با موفقیت به مخزن مرکزی انتقال داده و ثبت نمود، مریم نیز قصد دارد فیچری که توسعه داده است را در مخزن مرکزی ثبت نماید. بنابراین دستور زیر را وارد می‌کند:

```
git push origin master
```

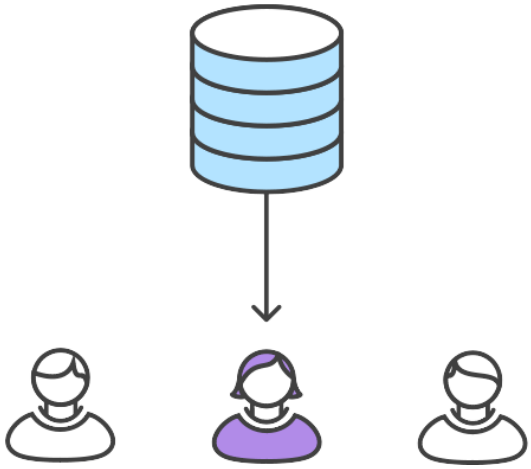
اما از آن‌جا که تاریخچه‌ی تغییرات مخزن محلی او با تاریخچه‌ی تغییرات مخزن مرکزی متفاوت است، گیت این درخواست را نمی‌پذیرد و خطای زیر را نمایش می‌دهد:

```
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.
```

گیت مانع بازنویسی پروژه در مخزن مرکزی می‌شود. مریم پیش از هر چیز باید مخزن محلی خود را به‌روزرسانی کند تا تغییراتی که جواد ایجاد نموده بود به مخزن محلی او منتقل و ادغام شود. حالا مریم می‌تواند برای انتقال فیچر مورد نظر خود به مخزن مرکزی دوباره اقدام کند.

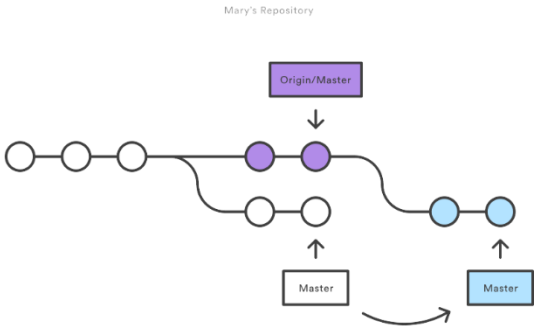
ث- مریم دوباره سعی می‌کند تغییرات مورد نظر خود را اعمال کند



مریم می‌تواند از دستور git pull برای ادغام تغییرات بالادستی در مخزن محلی خود استفاده کند.

```
git pull --rebase origin master
```

مثل دستورsvn update ، این دستور تمام تغییرات ثبت شده‌ی بالادستی را به مخزن محلی منتقل نموده و سعی می‌کند تا آن‌ها را با تغییرات ثبت‌شده‌ی محلی ادغام نماید. گزینه‌ی rebase– به گیت می‌گوید بعد از همگام‌سازی تغییرات با مخزن مرکزی، تمام تغییرات ثبت‌شده‌ی مخزن مرکزی را به مخزن محلی مریم انتقال دهد. اگر از این گزینه استفاده نکنید باز هم دستور pull کار می‌کند ولی هر بار که نیاز به همگام‌سازی با مخزن مرکزی باشد باید از دستور merge commit استفاده کنید. در مورد این گردش‌کار، بهتر است همیشه به جای دستور merge commit از rebase بهره ببرید.

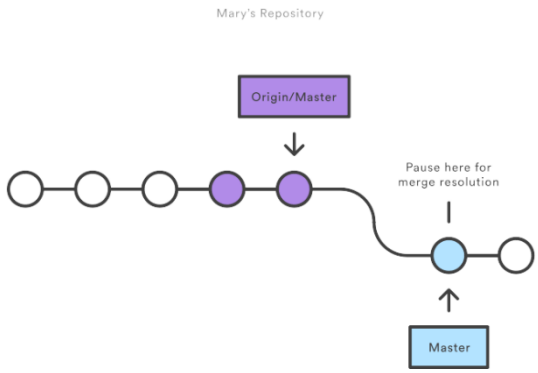


ج- مریم مغایرت‌ها را برطرف می‌کند

وقتی از گزینه‌ی rebase استفاده می‌کنید، تغییرات یک به یک به شاخه‌ی مستر مرکزی انتقال پیدا می‌کنند. بنابراین به جای حل کردن حجم بزرگی از مغایرت‌ها و تعارضات احتمالی با دستورmerge commit ، می‌توانید مورد به مورد آن‌ها را برطرف نمایید. این امکان باعث می‌شود که تمرکز بیشتری داشته باشید. علاوه بر این، تاریخچه‌ی پروژه‌ی تمیزی نیز خواهید داشت. واضح و تمیز بودن تاریخچه‌ی پروژه به نوبه‌ی خود سبب می‌شود که راحت‌تر محل ایجاد باگ‌ها را تشخیص دهید و در صورت لزوم با کمترین تأثیر بر پروژه، تغییرات را به حالت قبل برگردانید.

اگر مریم و جواد بر روی فیچرهای جداگانه کار کنند، احتمال به وجود آمدن مغایرت بسیار اندک خواهد بود. اما اگر چنین مغایرتی رخ دهد، گیت فرآیند rebasing را درست در همان موردی که مغایرت وجود دارد متوقف نموده و پیغام زیر را به همراه مجموعه‌ای از دستورالعمل‌های مرتبط نمایش خواهد داد:

```
CONFLICT (content): Merge conflict in <some-file>
```



نکته‌ی جالب در مورد گیت این است که هر کسی می‌تواند مغایرت‌های ادغام مربوط به خود را برطرف کند. به عنوان مثال، مریم می‌تواند با دستور git status متوجه شود که مشکل در کجا رخ داده است. فایل‌های حاوی مغایرت ذیل عنوان Unmerged paths نمایش داده می‌شوند:

```
# Unmerged paths:
# (use "git reset HEAD <some-file>..." to unstage)
# (use "git add/rm <some-file>..." as appropriate to mark resolution)
#
# both modified: <some-file>
```

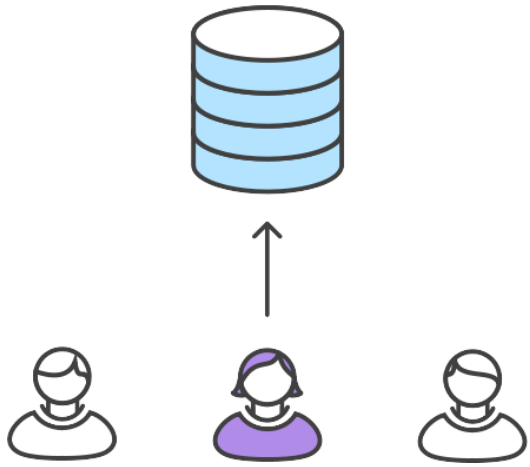
مریم بعد از ویرایش فایل(های) مورد نظر و هنگامی که به نتیجه‌ی رضایت‌بخشی دست پیدا کرد، فایل(ها) را به مرحله‌ی نمایش فرستاده و بعد از آن اجازه می دهد که git rebase ادامه‌ی کار را انجام دهد.

```
git add <some-file>
git rebase –continue
```

این تمام چیزی است که انجام می‌شود. گیت یک به یک به سراغ تغییرات بعدی رفته و هر زمان که مغایرتی پیش بیاید چرخه‌ی فوق را تکرار می‌کند.اگر در این مرحله سردرگم شدید و نمی‌دانستید چه کاری باید بکنید، آرام باشید و فقط دستور زیر را اجرا کنید تا به همان جایی بروید که شروع کردید:

```
git rebase --abort
```

چ- مریم فیچر خود را با موفقیت منتشر می‌کند



بعد از همگام‌سازی مخزن محلی با مخزن مرکزی، مریم با استفاده از دستور زیر می‌تواند تغییرات مورد نظر خود را با موفقیت منتشر کند:

```
git push origin master
```

همان‌طور که دیدیم، با استفاده از تعداد اندکی از دستورات گیت می‌توان محیط توسعه‌ی سنتی Subversion را شبیه‌سازی کرد. این برای تیم‌هایی که از SVN به گیت مهاجرت می‌کنند ویژگی بسیار خوبی است اما ماهیت توزیع‌شده‌ی گیت در آن مطرح نمی‌شود.

سایر گردش‌کارهای متداول در گیت

گردش‌کار متمرکز در واقع بخشی از دیگر گردش‌کارها در گیت است. محبوب‌ترین گردش‌کارهای گیت به نوعی دارای یک مخزن مرکزی هستند که دولوپرها می‌توانند کدهایی را به آن منتقل نموده یا از آن دریافت کنند. در ادامه به صورت خلاصه به بعضی از پرطرفدارترین گردش‌کارها در گیت خواهیم پرداخت. این گردش‌کارها امکانات بیشتری را در زمینه‌ی مدیریت شاخه‌ها در توسعه‌ی فیچرها، اصلاحات فوری و انتشار نهایی در اختیار کاربران قرار می‌دهند.

گردش‌کارFeature branching

گردش‌کار Feature branching در واقع یک بسط منطقی از گردش‌کار متمرکز است. نگرش اصلی حاکم بر این گردش‌کار این است که تمام فیچرها به صورت متمرکز و در یک شاخه‌ی جداگانه توسعه یابند. این مجزا کردن فیچرها از بقیه‌ی کد سبب می‌شود که چندین دولوپر بتوانند بر روی یک فیچر خاص کار کنند بدون این‌که برای کد اصلی مشکلی ایجاد شود. وجود این دیدگاه همچنین باعث می‌شود که در شاخه‌ی مستر هیچ‌وقت شاهد کدهای خراب نباشیم که مزیت بزرگی برای محیط‌های ادغام مداوم (continuous integration environments)است.

گردش‌کار Gitflow

گردش‌کار Gitflow نخستین بار در سال ۲۰۱۰ در یک پست وبلاگی منتشر شد و بسیار مورد توجه قرار گرفت. این گردش‌کار در حقیقت یک مدل دقیق شاخه‌سازی برای مدیریت و انتشار پروژه است. مفاهیم و دستورات مورد نیاز این گردش‌کار چیزی فراتر از مفاهیم و دستورات Feature branching نیست ولی در عوض نقش‌های بسیار مشخصی را به شاخه‌های مختلف اختصاص می‌دهد و مشخص می‌کند که هر شاخه چه زمانی و چگونه با شاخه‌ی اصلی ادغام شود.

گردش‌کار Forking

گردش‌کار Forking از سایر گردش‌کارهایی که در این آموزش مطرح شد متفاوت است. در این گردش‌کار به جای این‌که یک مخزن سمت سرور واحد به عنوان مخزن مرکزی وجود داشته باشد، به ازای هر دولوپر یک مخزن سمت سرور وجود دارد. یعنی هر دولوپر نه یک مخزن، بلکه دو مخزن دارد: یک مخزن محلی شخصی و یک مخزن سمت سرور عمومی.

چند نکته و دستورالعمل

هیچ گردش‌کاری وجود ندارد که بتوان آن را برای همه‌ی تیم‌ها و پروژه‌ها توصیه کرد. همان‌طور که پیش از این اشاره کردیم، نکته‌ی مهم این است که گردش‌کار گیت بتواند بهره‌وری تیم را افزایش دهد. همچنین علاوه بر فرهنگ تیم، یک گردش‌کار مناسب باید متمم فرهنگ کسب و کار نیز باشد. امکانات گیت مانند شاخه‌ها و تگ‌ها باید مکمل روند کسب و کار شما باشند نه مزاحم آن. در ادامه چند دستورالعمل مهم در انتخاب گردش‌کار مناسب را بیان می‌کنیم:

عمر شاخه‌ها را تا حد امکان کوتاه کنید

هرچقدر شاخه‌ها زمان طولانی‌تری جدا از شاخه‌ی اصلی باشند، ریسک ایجاد مغایرت‌ها و مشکلات اجرای کدها بیشتر می‌شود. شاخه‌های کم‌عمر می‌توانند فرآیند ادغام و اجرای راحت‌تری را به ارمغان آورند.

بازگشت‌ها را به حداقل رسانده و روند آن را آسان کنید

مهم است که گردش‌کار مورد نظر شما بتواند مانع از ادغام کدهایی شود که بعدها مجبور شوید آن‌ها را به حالت قبل برگردانید. به عنوان مثال گردش‌کاری که در آن هر شاخه قبل از ادغام با شاخه‌ی مستر مورد آزمون قرار می‌گیرد، از این نظر، گردش‌کار مناسبی است. البته به هر حال گاهی چنین اتفاقی می‌افتد و مجبور خواهید بود که کد را به حالت قبل برگردانید. بنابراین باید گردش‌کاری را انتخاب کنید که این فرآیند بازگشت در آن به سادگی امکان‌پذیر بوده و در کار سایر اعضای تیم اخلال ایجاد نکند.

از یک برنامه‌ی انتشار مشخص پیروی کنید

یک گردش‌کار مناسب باید چرخه‌ی انتشار نرم‌افزار را تکمیل کند. به عنوان مثال اگر روزانه در چندین نوبت انتشار را انجام می‌دهید، لازم است به فکر پایداری شاخه‌ی مستر باشید. در حالی که اگر دیر به دیر انتشار را انجام می‌دهید بهتر است از تگ‌های گیت برای برچسب زدن به شاخه‌های مختلف استفاده کنید.

جمع بندی

در این مقاله ما گردش کار Gitflow را بررسی کردیم Gitflow .یکی از چند گردش کاری است که تیم شما می تواند از آن تبعیت کند. این گردش کار برای تیم هایی که برنامه ریزی توسعه محصول را بر اساس انتشار آن انجام می دهند بسیار مناسب است. همچنین این گردش کار، روندهایی مخصوص رفع نقص محصول منتشر شده تعریف می کند و این روند را شفاف می کند.

روند کلی گردش کار Gitflow به صورت زیر است:

۱. شاخه ی develop از شاخه ی master گرفته می شود.
۲. شاخه ی انتشار از شاخه ی develop گرفته می شود.
۳. شاخه های feature از شاخه ی develop گرفته می شوند
٤. وقتی که کار روی یک شاخه ی feature تمام می شود، این شاخه در شاخه ی merge ، developمی شود.
٥. وقتی که کار روی شاخه ی انتشار تمام می شود، این شاخه در هر دو شاخه ی develop و masterبا هم merge می شود.
٦. اگر نقص در شاخه ی master شناسایی شود، یک شاخه ی hotfix از شاخه ی master گرفته می شود.
۷. وقتی که نقص در شاخه ی hotfix رفع شد، این شاخه در هر دو شاخه ی develop و master ادغام می شود.