Git stash.1

این دستور فایل های ویرایش شده را به طور موقت ذخیره میکند. Stash در لغت به معنی ذخیره کردن است. در گیت نیز هر تغییری که کامیت نشده باشد را ذخیره میکند.

در مواقعی که تغییرات خود را به دلایلی نخواهیم کامیت کنیم به عنوان مثال قبل از تکمیل نهایی پروژه مایل به کامیت کردن آن نباشیم میتوانیم از این دستور استفاده کنیم و به طور موقت این موارد را ذخیره کنیم و بعد در صورت نیاز آن را restore کنیم.

Git reset.2

زمانی که قصد داریم کامیت های برنچ خصوصی خودمان و یا تغییرات کامیت نشده را از بین ببریم و یا بتوانیم head جاری را به یک وضعیت مشخص ریست کنیم. میتوان وضعیت فایل های خاصی و یا یک برنچ را ریست کرد.

این تغییرات برای مواقعی که در ریموت push نشده باشند مفید است.

با دستور زیر میتوان قسمت های موردنظر را انتخاب و آن ها را revert یا unstage کرد:

git reset (--patch | -p) [tree-ish] [--] [paths]

در صورتی که فایل هایی را به اشتباه استیج کنیم که مربوط به آن کامیت نیستند نیز با استفاده از این دستور میتوانیم فایل تغییر یافته را از استیج به دایرکتوری کاری انتقال دهیم.

git reset HEAD FILE-TO-UNSTAGE

Git checkout

با این دستور و وارد کردن هش کامیت مربوط میتوانیم به ورژنی از کامیت ها که مدنظر هست برویم ولی این روش کامیت های جدید یا اطلاعات جدید را در سرور تغییر نمی دهد.

همچنین میتوانیم برای رفتن به برنچ موردنظر از این دستور استفاده کنیم.

Git revert

این دستور را میتوان به عنوان یک undo شناخت. به جای حذف کامیت از تاریخچه پروژه این دستور تغییرات ایجاد شده توسط کامیت را برمیگرداند و محتوای نهایی را در قالب یک کامیت جدید ثبت میکند. این کار جلوی از دست رفتن تاریخچه گیت را میگیرد.

این دستور در مواقعی که بخواهیم به آخرین ورژن از تغییرات برگردیم مناسب تر است. در مواقعی که به چند کامیت قبل تر برگردیم امکان وقوع conflict وجود دارد.

Git Merge.3

ادغام کردن باعث میشود که محتوای شاخه مبدا را برداریم و با شاخه مقصد یکپارچه کنیم. در این فرآیند تنها شاخه مقصد

تغییر می یابد و سابقه شاخه مبدا دست نخورده باقی می ماند. از مزایای این روش حفظ کردن سابقه کامل و ترتیب زمانی

و همچنین چارچوب شاخه می باشد. شیوه ارسال تغییرات از پایین رو به بالاست.

از معایب این روش سابقه کامیت ها به دلیل وجود تعداد زیادی از کامیت های merge آلوده است و دیباگ کردن دشوار می شود.

Git Rebase

روشی برای یکپارچه سازی تغییرات یک شاخه در شاخه دیگر است. در این روش همه تغییرات در یک Patch منفرد فشرده میشود و سپس در شاخه مقصد ادغام میشود.

Rebase کردن برخلاف merge کردن باعث از بین رفتن سابقه تغییرات میشود زیرا کل کار را از یک شاخه به شاخه دیگر انتقال میدهد. در این فرآیند سابقه شاخه ناخواسته حذف میشود. شیوه ارسال تغییرات از سلسله مراتب بالاتر به بخش های پایین تر است.

از مزایای این روش منظم شدن یک سابقه پیچیده است. دستکاری یک کامیت منفرد آسان است. از ایجاد نویز کامیت در ریپورت های شلوغ با شاخه های زیاد جلوگیری میکند. کامیت های میانی از طریق تبدیل شدن به یک کامیت منفرد پاک میشوند که برای تیم های devOps بسیار مفید است.

Rebase کردن یک ریپوزیتوری عمومی در موارد کار تیمی میتواند کار خطرناکی باشد. اگر به طرز نادرستی rebase کنیم و ناخواسته کل سابقه را بازنویسی کنیم میتواند منجر به مشکلات جدی شود از این رو باید اطمینان حاصل کنیم دقیقا میخواهیم چه کاری انجام دهیم.

GitFlow.4

به طور کلی میتوانیم gitFlow را یک ایده مدیریتی برای محیط های توسعه در نظر بگیریم که روشی برای ساخت انواع برنچ ها محسوب شده و همچنین طریقه مرج کردن آن ها با هم را مشخص میکند.

در انجام یک پروژه به جای استفاده از برنچ های مختلف با یک برنچ مستر نیز میتوانیم از برنچ develop و master استفاده کنیم. برنچ master در واقع نسخه لانچ شده پروژه بوده و برنچ develop همان نسخه ای است که تغییرات نهایی پروژه و تمام فیچرها روی آن قرار گرفته و ابتدا تست میشوند و در نهایت با برنچ مستر مرج میشوند.

بعد ار هر بار لانچ کردن با استفاده از تگ ها باید برنچ مستر را ورژن بندی نمود.

روند کلی گردش کار GitFlow به صورت زیر است:

1. شاخه develop از شاخه مستر گرفته میشود.
2. شاخه انتشار از شاخه develop گرفته میشود.
3. شاخه های feature از شاخه develop گرفته میشود.
4. وقتی که کار روی یک شاخه feature تمام میشود این شاخه در شاخه develop مرج میشود.
5. وقتی که کار روی شاخه انتشار تمام میشود این شاخه در هر دو شاخه ی develop و master با هم مرج میشود.
6. اگر نقص در شاخه مستر شناسایی شود یک شاخه hotfix از شاخه مستر گرفته میشود.
7. وقتی که نقص در شاخه hotfix رفع شد این شاخه در هر دو شاخه ی develop و master ادغام میشود.