برای استفاده از گیت میتوان دستورات مربوط به گیت را در ترمینال وارد کرد. برای کار با گیت ابزارهای گرافیکی هم موجود هستند اما کار با ترمینال سرعت و قدرت بیشتری دارد. اولین کاری که بعد از نصب گیت باید انجام دهید، شناساندن خودتون به گیت هستش. برای اینکار از دستور config استفاده میشود. دستور config وظیفه ایجاد و تغییر optionها بروی پروژه را دارد. زمانی پروژه جدیدی ایجاد میکنید میتوانید با دستور init به گیت بفهمانید که تغییرات اعمال شده بروی این پروژه باید تحت کنترل باشند. زمانی که با دستور init یک پروژه را به گیت فهماندید، گیت آن پروژه را به یک مخزن(repository) تبدیل میکند. مخزن پروژه ای است که git تغییرات آن را کنترل میکند.

Git help ‘command’

این دستور توضیحاتی درباره command مورد نظر میدهد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git config user.name “sajjad”

با استفاده از این دستور تنها در پروژه جاری نام شمارو set میکند

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git config --global user.name “sajjad”

این دستور به صورت global نام شما را در تمام پروژه ها set میکند

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

بعد از دستورات باید ایمیل خود را نیز وارد کنید :

Git config -- global user.email [sajjad@gamil.com](mailto:sajjad@gamil.com)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git config –list

نمایش optionها و configهایی که set کرده اید ماننده اسم و ایمیل

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git init

ساخت یک repository خالی برای پروژه ی شما . بعد از زدن این دستور خود گیت پوشه ی جدیدی به نام .git را میسازد که config های این پروژه را در آن ذخیره میکند

=====================================================

زمانی که تغییری در پروژه شما ایحاد میشود میتوانید با کمک دستور git status تغییرات رو ببینید. بعد از اینکه گیت تغییرات پروژه شما رو فهمید اگر از تغییرات راضی بودید باید اقدام به ثبت تغییرات کنید. در گیت سه مرحله برای ثبت تغییرات وجود دارد. مرحله اول untracked یا modified است که نشاندهنده تغییرات یک فایل یا ایحاد شدن یک فایل است. بعد با دستور git add باید تغییرات رو به مرحله دوم یعنی staged ارسال کنید. این مرحله نشان دهنده رضایت شما از تغییرات ایجاد شده است. و در آخر برای ثبت کردن تغییرات باید با دستور git commit پروژتون رو به مرحله commited بفرستید. زمانی که به این مرحله رسیدید، پروژه شما در حالت ثبات قرار میگیره.

Untracked files به معنی این است که این فایل تازه به وجود آماده است و تاکنون git آن را دنبال نمیکرده است

Modified به فایل هایی گفته میشونده که git آنهارا دنبال میکرده است (از قبل وجود داشته اند) اما تغییر کرده اند و لازم است آنها را ثبت کنیم



Git add name\_file

Gti commit -m ‘massage’

=====================================================

زمانی که تعداد فایل ها در مرحله اول زیاد بود به جای نوشتن اسم تمام فایل ها میتونید با استفاده از A- به گیت بفهمونید که تمام فایل ها رو به مرحله staged منتقل کنه همچنین میتونید برای راحتی از کاراکتر . نقطه هم استفاده کنید. در این قسمت میبینید که چطور میشه با استفاده از دستور reset فایل ها که داخل مرحله staged هستنید رو به مرحله اول برگردونید. همچنین با روش wildcard برای ارسال فایل ها از مرحله اول به مرحله دوم آشنا میشید

Git add name\_file1 name\_fiel2

Git add -A ======> git add .

انتقال فایل ها از مرحله اول به مرحله دوم

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git reset name\_file

با استفاده از این دستور فایل انتخابی از مرحله staged به مرحله اول برمیگرد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Wildcard (عبارات باقاعده) مثلا :

Git add \*.txt

این دستور به این معنی است که فقط تمام فایل هایی که پسوند txt دارند را به حالت staged منتقل کن

=====================================================

بعد از اینکه چندین commit در گیت ایجاد کردید به احتمال زیاد دلتون میخواد که بتونید یک تاریخچه از تغییراتی که داشتید رو ببینید. برای اینکار میتونید از دستور log در گیت استفاده کنید. به صورت پیشفرض اگر هیچ آپشنی به دستور ندید، تمام commitها از جدید به قدیمی نشان داده میشود. این دستور یک هش چهل کاراکتری از نوع SHA-1 به همراه، name, email و تاریخ و پیغام هر commit را نشان میدهد. هر commit دارای یک هش است که این هش بین تمام commitها یکتا بوده و با استفاده از آن میتوان در تاریخچه تغییرات حرکت کرد. همچین دستور log در git آپشنهای زیادی داره که با استفاده از اونها میتونید شکل نمایش log رو تغییر بدین. در این قسمت با دستور log در گیت آشنا میشید

Git log

Git log –oneline

Git log –stat

هر سه ی این دستورات تمام commit های مارا نمایش میدهد اما هرکدام با جزئیات متفاوت

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git log –patch ===> git log -p

این هم ماننده 3 تای بالایی میباشد با این تفاوت که تغییرات هر commit نسبت به commit قبل را به صورت کامل نمایش میدهد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HEAD همیشه به آخرین commit اشاره دارد

=====================================================

branchها در واقع یک کپی از برنامه شما هستند. زمانی که بخواید به پروژتون یک ویژگی جدیدی اضافه کنید به جای کار کردن روی سورس پروژه اصلی با استفاده از گیت میتونید یک branch از برنامه خودتون ایجاد کنید. با این کار گیت یک کپی کامل از سورس پروژه شما رو کپی میکنه و شما میتونید روی اون branch کار کنید و اگر از تغییرات راضی بودید میتونید اون branch رو با سورس پروژه اصلی که به صورت پیشفرض روی branch master هستش ترکیب کنید. اما اگر از تغییرات راضی نبودید میتونید خیلی راحت اون branch رو حذف کنید بدون اینکه تغییری روی سورس اصلی برنامتون اعمال بشه

Git branch ===> git branch –list

نمایش لیست branch ها

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git branch name\_branch

برنچی با نام دلخواه شما (name\_branch) ساخته میشود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git checkout name\_branch

وارد برنچ مورد نظر شما میشود (تغییر branch)

=====================================================

برای تغییر نام branch میتونید از آپشن m استفاده کنید. برای ترکیب کردن دو branch مختلف میتونید از دستور merge استفاده کنید. برای حذف کردن branch دو آپشن مختلف وجود دارد. اگر branch قبلا merge شده بودید میتونید از آپشن d استفاده کنید در غیر اینصورت میتونید از آپشن D استفاده کنید.

Git branch -m name\_origin name\_new

تغییر نام برنچ از name\_origin به name\_new

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git checkout master

Git merge name\_branch

برای اینکه تغییراتی که در برنچی اعمال کردیم را بر روی برنچ مستر بیاوریم باید ابتدا وارد master بشوید بعد با زدن دستور دوم اینکار انجام میشود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git branch -d name\_branch

برنچی که مرج شده است را با استفاده از -d پاک میکند

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git branch -D name\_branch

برنچی که هنوز مرج نشده است با این دستورپاک می‌شود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

git push origin --delete service/CS-5199

برنچی که بر روی remote پوش شده است پاک میشود

=====================================================

دستور checkout کاربرد متفاوتی دارد. قبلا دیدید که با استفاده از checkout میتونستید branch های مختلفی رو فعال کنید. در این جلسه با کاربرد دیگری از checkout آشنا میشید. با استفاده از این دستور میتونید بین commit ها حرکت کنید و پروژتون رو به commit خاصی برگردونید. دقت کنید که دستور checkout تغییرات رو حذف نمیکنه و بلکه فقط پروژه ما رو به زمان خاصی برمیگردونه تا ما بتونیم وضعیت پروژمون رو در commit خاصی ببینیم. در این حالت تغییراتی که ایجاد میکنید، تاثیری در سورس اصلی نخواهد داشت و این حالت read-only میباشد

Git checkout hash\_commit

هنگاهیکه hash کامیت خود را پیدا کردید با استفاده از این دستور به آن کامیت میرود .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

هنگامیکه وارد commit مورد نظر شدیم حال قصد تغییر دادن وضعیت را داریم .میتوانیم این کار را انجام دهیم و وضعیت تغییر یافته را کامیت کنیم اما با بازگشت به HEAD آن کامیت از بین میرود . اما گیت پیغامی به شما نمایش میدهد که همچین اتفاقی افتاده است و بهترین موقع برای نگه داشتن آن کامیت الآن میباشد شما باید برنچ جدید بسازید همراه با hash آن کامیت

مثلا همچین پیغامی نمایش میدهد



وقتی آن کار را انجام دهید و با مرج کردن میتوانید آن کامیت را نگه دارید

Git checkout main/master

بعد از اینکه وارد کامیت مورد نظر شدید برای خروج از کامیت و رفتن به آخرین کامیت از این دستور استفاده میکنیم

=====================================================

در این قسمت با دستور revert آشنا میشید. این دستور به شما اجازه میده که تغییراتی که در commit خاصی ایجاد کردید رو حذف کنید. این دستور نسبت به دستور checkout قدرت بیشتری دارد. دقت کنید که دستور revert در گیت یک commit را حذف نمیکنه بلکه یک commit جدید ایجاد کرده و تاثیرات commit قبلی را حذف میکند.

Git revert abort hash\_commit

تاثیرات کامیت با هش مورد نظر را حذف میکند . البته این دستور بد درد آخرین commitها میخورد چرا که اگر بخواهید تاثیرات کامیت های قبلتر را ازبین ببرید به confilict میخوریم

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git revert –abort

هنگامیکه متوجه شدیم که این revert به confilict میخورد با این دستور میتوانیم revert را متوقف کنیم

=====================================================

در این قسمت با دستور reset در گیت آشنا میشوید. دستور reset نسبت به دو دستور قبلی revert, checkout قدرت بیشتری دارد. این دستور واقعا توانایی حذف را دارد به همین خاطر در استفاده از این دستور دقت کنید. این دستور در سه سطح مختلف کار میکند. سطح اول زمانی است که شما از آپشن soft استفاده میکنید. در این سطح دستور reset اقدام به حذف commit ها و انتقال فایل ها به حالت staged میکند. سطح دوم که با mixed مشخص میشود commit ها را حذف کرده و فایل ها را به بخش untracked/modified منتقل میکند. و سطح آخر که با hard مشخص میشود تمام commit ها و فایل ها را حذف میکند.



به طور پیش فرض –soft میباشد

Git reset hash\_commit –soft

Git reset hash\_commit –mixed

Git reset hash\_commit –hard

=====================================================

فایل gitignore در گیت مشخص کننده مواردی است که گیت باید آنها را نادیده بگیرد. زمانی در حال کار بروی پروژه ای هستید متوجه میشوید که یکسری فایل ها و دایرکتوری ها توسط سیستم عامل و یا IDE ساخته شده اند. این فایل ها نباید در کنار سورس اصلی پروژه در گیت قرار بگیرند. به همین دلیل در فایل gitignore لیستی از فایل ها و دایرکتوری هایی که باید توسط گیت نادیده گرفته شوند نوشته میشنود و گیت دیگر این موارد را شناسایی نمیکند. دقت کنید که نوشتن نام فایل ها و دایرکتوری هایی که قبلا توسط گیت شناسایی شده اند بی تاثیر است.

سایت gitignore.io

=====================================================

github یک پلتفرم برای اشتراک گذاری پروژه هایی است که از git استفاده میکنند. با استفاده از github میتونید پروژه های خودتان را به بقیه برنامه نویس ها نشان داده و از کمک آنها بهره مند شوید. تاکنون ابزارها و تکنولوژی ها متعددی در github با کمک بقیه برنامه نویس ها ساخته شده اند. در حال حاضر بیش از 40 میلیون برنامه نویس و 100 میلیون پروژه در گیتهاب وجود دارد.

Git remote add name url\_git

با استفاده از این دستور میتوانیم یک remote برای repository خود با نام name ایجاد کنیم مثلا :

Git remote add sajjad <https://github.com/mr-moshtaghi/test.git>

Git push sajjad

اکنون اگر دستور git push sajjad را بزنیم repository local ما به آن آدرس git متصل میشود و تمام تغییرات را بر روی آن repository remote اعمال میکند

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git remote -v

Remote هایی که برای این repository local وجود دارند را نشان میدهد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git remote rm name\_remote

حذف remote مورد نظر

=====================================================

Git checkout -b name\_branch

این دستور یک نوع shortcut میباشد به این صورت که هم برنچ ساخته میشود و همزان به برنچ مورد نظر switch میشود

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git push name\_remote name\_branch

برنچ مورد نظر را به remote مورد نظر push میکند

=====================================================

issue به شما اجازه میده که با صاحب پروژه صحبت کنید. میتونید در رابطه با اضافه کردن یک ویژگی جدید یا برطرف کردن یک باگ با صاحب مخزن صحبت کنید.

=====================================================

 فایل README حاوی یکسری اطلاعات در رابطه با پروژه شما است. همچنین میتونید با زبان Markdown فایل های README رو استایل دهی کنید.

=====================================================

fork یک کپی از مخزن شخص دیگری است که گیتهاب برای شما ایجاد میکند. با استفاده از fork میتونید روی پروژهای دیگه تغییر ایجاد کرده و در توسعه آنها مشارکت کنید. هدف دیگه fork، استفاده از پروژه دیگران به عنوان پایه و اساس پروژه خودتون است. به عنوان مثال اگه شما یک باگ در یک پروژه پیدا کردید میتونید ابتدا اون پروژه رو fork کنید، مشکل رو برطرف کنید و در آخر با یک pull request اون رو به پروژه اصلی بفرستید

=====================================================

در این قسمت با merge conflict و نحوه رفع این مشکل در گیت آشنا میشید. این مشکل زمانی رخ میده که یک فایل در دو branch مختلف تغییر کرده باشه و شما زمانی که سعی کنید این دو branch رو با هم ترکیب کنید با این مشکل روبرو میشید. برای رفع این مشکل باید دو تا branch رو باهم مقایسه کنید و تغییرات یکی از branch ها رو انتخاب کنید و اون یکی رو پاک کنید.

البته در pycharm قابلیت ادغام وجود دارد

زمانی با یک تیم در حال کار روی یک پروژه در گیتهاب هستید ممکنه فایلی که توسط شما تغییر کرده، همزمان توسط کس دیگه ای هم تغییر کرده باشه. در این حالت اگر بخواید branch خودتون رو با branch master پروژه merge کنید، گیت به شما ارور میده. برای حل این مشکل ابتدا با استفاده از دستور git pull تمام تغییرات داخل پروژه رو میارید داخل local خودتون و بعد میتونید تصمیم بگیرید که کدوم تغییر باید بمونه.

=====================================

git restore settings.py

با استفاده از دستور بالا میتوانید فایل settings.py را در لکال خود نگه داریم و نگذاریم بالا برود