**آموزش git :**

**Version control systerm** : یک نرم فزار هست که روی سیستم ما به صورت لوکال نصب میشه و وظیفش این هست که تغییرات فایل های ما رو ردیابی و مدیریت کند.

گیت یکی از کنترل ورژن ها هست. ( ما کنترل ورژن های دیگری هم داریم.)

**Git Hub** : گیت و گیت هاب دو موضوع جدا از هم هست. گیت کاری به اینترنت نداره و فقط یه نرم افزار هست اما گیت هاب این قابلیت رو بهمون میده که نسخه گیتی که روی سیستم به صورت لوکال داریم میتونیم در فضای اینترنت قرارش بدیم.

**نصب git :**

برای نصب گیت نکته خاصی ندارد و موارد انتخابی را به صورت دیفالت تغییری نمیدیم و فقط next میزنیم.

حال اگر با کلیک راست git bash را باز کنیم یک پنچره شبیه به cmd باز میشود. ( این پنچره با cmd تفاوت هایی دارد. در این پنچره دستورات شبیه به دستوارت لینوکس هست ولی در cmd دستورات تفاوت دارد با دستورات لینوکس)

**Git GUI** : در واقع مخزنی که داریم رو میتونیم به صورت گرافیکی نمایش دهیم.

برای اینکه بتونیم ببینیم گیت روی سیستم نصب هست در cmd دستور زیر رو وارد میکنیم.

git – version

اضافه کردن گیت به فایل های پروژه :

برای این کار باید یک ریپازیتوری ایجاد کنیم. ریپازیتوری یعنی محفظه، یعنی گیت میاد یک محفظه ایجاد میکنه تا پروژه های مختلفی که روی گیت هست با هم قاطی نشوند.

از طریق cmd وارد پوشه پروژه خودمون میشیم. و دستور git init میزنیم که جواب زیر رو برامون برمیگردنه :

Initialized empty Git repository in ادرس پروژه ای که مقصد ما هست

که به معنی این هست که یک مخزن خالی گیت در ادرس پروژه ما ایجاد شده است.

حال اگر داخل پوشه پروژه خودمون بریم یک پوشه به نام git . ایجاد شده که hide هم هست. که داخل پوشه یک سری فایل هست که گیت برای مدیریت پروژه ما به اونها نیاز داره.

**دستور بعدی git status هست :** به این معنی هست که وضعیت پروژه من چی هست.

مثلا میگه که روی چه branch هستیم وضعیت commit ها رو بهمون نشون میده. و

**Untracked files :** که نام فایل هایی که ردیابی نمیشوند به ما با رنگ قرمز نمایش میده. یعنی داره میگه من این سه تا فایل رو اصلا نمیشناسم. اگر میخوای اینها رو برات ردیابی و مدیریت کنم باید بیای و از git add استفاده کنی.

نکته **:** اگر در ادرس پروژه که گیت روی آن نصب نشده دستور git status رو بزنیم پیغام زیر را خواهیم دید:

fatal : not a git repository (or any of the parent directories): .git

و به این معنی هست که میگه فایل .git اصلا وجود نداره داخل این پوشه که من بخوام وضعیت گیت رو بهت نشون بدم.

**دستور git add :**

با این دستور میتونیم فایلی رو به گیت معرفی کنیم تا شروع به مدیریت اون فایل کنه.

git add نام فایل

وقتی این دستور رو مینویسیم چیزی رو برای ما بر نمیگردونه. حال اگر git status رو بزنیم :

Changes to be commited:

(use “git rm –cached <file>…” to unstage)

new file: file1.text

untracked files:

(use “git add <file>…” to include in what will be committed)

file2.txt

file3.txt

در بخش اول میگه یک دونه فایل داری که این رو الان میشناسم فقط کاری که باید کنی این هست که کامیتش کنی.

و در بخش دوم فایل هایی که ردیابی نمیشن ) untracked ) رو به ما نمایش میده. بعد از اینکه دستور git add رو میزنیم فایل ما وارد ناحیه stage میشه. بعد از stage باید فایل ها رو وارد ریپازیتوری کنیم.

**و با دستور “git rm –cached <file> میتونیم فایل رو از stage بیرون بیارم.**

دستور git add . : این نقطه به پوشه جاری اشاره میکنه و میگه که هرانچه در این پوشه هست ( اعمم از پوشه ها و فایل ها ) همه رو وارد stage کن.

**دستور git commit :**

این دستور مقادیری که داخل stage هست رو برامون میاره داخل ریپازیتوری.

git commit –m “ متن مورد نظر برای کامیت”

اگر این دستور را اجرا کنیم خطای زیر رو خواهیم دید:

Author identity unknown …..

به این معنی که نویسنده این کامیت مشخص نشده است و در ادامه ی خطا به ما راهنمایی میکنه که با یک ایمیل و یوزرنیم نویسنده را به گیت معرفی کنیم.

git config –global user.email [you@example.com](mailto:you@example.com)

git config –global user.name “your Name”

این کار برای اینکه مشخص کنیم کامیت را کدام یک از اعضای تیم ثبت کرده مفید است.

ایمیل خودمون رو وارد میکنیم. و یوزر نیم هم دلبخواه است.

برای اینکه یوزرنیم و ایمیلی که ست کرده ایم رو ببینیم از دستورات زیر استفاده میکنیم:

git config user.name

git config user.email

حالا اگر کامیت رو بزنیم یک کد هش رو به ما نشون میده. (هر کامیتی داری یک کد هش شده است). و تعداد فایل هایی که کامیت شده اند را هم بهمون نشون میده.

**دستور git log :**

این دستور کامیت هایی که انجام شده رو بهمون نشون میده. نویسنده کامیت هم بهمون نشون میده و تاریخ کامیت به همراه متن کامیت هم بهمون نشون میده.

**نکته :** حال اگر در فایل هایی که کامیت شده اند تغیری ایجاد شود و ما git status رو بزنیم به صورت فایل های modified بهمون نمایش میده.

**نکته :** فرض کنیم چند فایل تغییر یافته و ما هنگام کامیت کردن یکی از فایل ها رو که باید داخل اون کامیت میبود رو فراموش کرده ایم که اضافه کنیم. و در واقع میخوایم کامیت رو اصلاح کنیم.

برای رفع این مشکل دوباره git add . رو میزنیم تا تمام فایل های باقی مانده به stage بروند. حالا از دستور زیر استفاده میکنیم.

git commit –amend

حال یک متن برای ما نمایش میدهد:

Waiting for your editor to close the file …

میگه که منتظرم تغییراتت رو توی ادیتورت اعمال کنی و سیو کنی. ( توی مراحل نصب ما مشخص کردیم که از چه ادیتوری میخوایم استفاده کنیم.) بعد از این متن ادیتوری که در مرحله نصب مشخص کرده ایم برایمان باز میشه. و یک تب به نام COMMIT\_EDITMSG برامون باز میشه که در بالای صفحه متن کامیت رو ببینیم و همچنین فایل هایی که تغییر کرده و قرار هست اضافه بشه رو ببینیم که اینها قابل ویرایش هستند. بعد از ویرایش این فایل سیو میکنیم. و میبندیمش

کامیت ویرایش شده رو میتونیم داخل cmd ببینیم.

**نکته** : این دستور فقط به کامیت اخر اشاره میکنه و نمیتونیم یک کامیت خاص رو ویرایش کنیم.

**دستور git log –oneline :** مثل دستور git log هست با این تفاوت که خلاصه ای از کامیت ها رو به ما نمایش میده.

**استفاده از Git Gui :**

زمانی که این نرم افزار رو باز میکنیم یک گزینه به نام Open Existing Repository ( یعنی ریپازیتوری که در حال حاضر وجود داره رو میخایم باز کنیم.) اگر که روش کلیک کنیم و بعد از اون باید فایل پروژه ای که به گیت وصل هست رو انتخاب کنیم. که وضعت فایل ها رو به ما نمایش میده. نرم افزار مشابه GitKraken هست که همین کار رو برامون میکنه. یا اکشتنسن Git History که در vs code هست.

**فایل git ignore :** ممکنه نخوایم یک سری از فایل های پروژه رو داخل گیت قرار بدیم. مثلا ماژول هایی که داخل پروژه هست و حجمش هم زیاد هست.

برای اینکار نیاز هست که یک فایل در پوشه پروژه به اسم git ignore ایجاد کنیم.

توجه : این فایل فقط پسوند داره و یعنی به اینصورت باید اون رو ایجاد کنیم:

.gitignore

حالا داخل این فایل میتونیم نام فایل هایی که میخوایم ازشون صرف نظر بشه رو بنویسیم.

و بعد این فایل رو هم کامیت میکنیم. و روی گیت قرار میدیم.

**شاخه یا branch :**

ایجاد یک شاخه و سویچ کردن بین شاخه ها :

**دستور git branch :** لیست شاخه های ما رو نشون میده. و روی هر شاخه ای که باشیم کنار اسم اون شاخه یک \* میگزاره.

برای ایجاد یک شاخه جدید از دستور بالا استفاده میکنیم و بعد از دستور اسم شاخه مورد نظرمون رو مینویسیم. مثلا :

git branch bugfix

**دستور git switch :** با این دستور میتونیم به شاخه مورد نظرمون سویچ کنیم. مثلا:

git switch bugfix

حالا پروژه ما تا الان هر چیزی که بوده میاد روی شاخه bugfix. حالا اگر تغییراتی روی فایل ها داشته باشیم و کامیت کنیم تغییرات روی شاخه bugfix اعمال میشه.

نکته: اگر برگردیم روی شاخه ای که از قبل بودیم دیگه اون تغییراتی که روی شاخه bugfix اعمال کردیم رو نخواهیم دید.

و : روی هر شاخه ای که باشیم و دستور git log رو بزنیم کامیت های که ایجاد کردیم ( هر چیزی که قبل از این کامیت وجود داشته چه در این شاخه و چه در شاخه اصلی) همه رو برامون برمیگردونه.

اگر در یک شاخه تغییراتی داشتیم و بدون اینکه کامیت کنیم به شاخه دیگری برویم تغییرات برای ما به اون شاخه میاد و میتونیم روی اون شاخه کامیت کنیم.

**دستور git log –all –graph :** این دستور کامیت ها رو به صورت گرافیکی برامون نمایش میده.

**دستور git switch –c bugfix2 :** این دستور هم شاخه bugfix2 رو ایجاد میکنه و هم سویج میکنه روی این شاخه.

**دستور git branch -m bugfix3:** این دستور برای تغییر نام شاخه هست، با این دستور روی هر شاخه ای که باشیم نام اون شاخه رو میتونیم عوض کنیم. ( در این مثال نام به bugfix3 تغییر خواهد کرد.)

**دستور git branch –d bugfix3 :** این دستور برای حذف شاخه استفاده میشه.

**نکته:** روی شاخه ای که باشیم نمیتونیم اون شاخه رو حذف کنیم و باید بریم روی یک شاخه دیگه و بعد اون شاخه رو حذف کنیم.

**ارور :**

error: the branch ‘bugfix3’ is not fully merged.

If you are sure you want to delete it, run ‘git branch –D bugfix3’.

این ارور میگه روی شاخه bugfix3 یک سری تغییرات هست که مرج نشده. اگر مطمعنی که میخوای حذفش کنی به جای –d از –D استفاده کن.

**Merge :**

برای مرج کردن شاخه ممکن است دو حالت پیش آید:

**Fast Forward :** توی این حالت از شاخه اصلی به شاخه دیگری رفتیم و روی اون شاخه تغییراتی داشتیم و حالا میخوایم مرج کنیم ( نکته: روی شاخه اصلی کامیتی نداشتیم)

توی این حالت هر کامیتی که در شاخه فرعی درست کردیم میاد بالا و در شاخه master قرار میگیره.

برای مرج کردن یک شاخه باید روی شاخه بالای باشیم و بعد شاخه پایینی رو مرج کنیم.

**دستور git merge bugfix :** این دستور کامیت های شاخه bugfix رو میاره روی شاخه بالایی ( مثلا master).

**حالت دوم :** توی این حالت از شاخه اصلی به شاخه دیگری رفتیم و روی اون شاخه تغییراتی داشتیم و حالا میخوایم مرج کنیم با این تفاوت که روی شاخه اصلی هم کامیت هایی رو داشتیم.

توی این حالت یک کامیت جدید ایجاد میشه. و توی این حالت ممکنه کانفلیکت هم رخ بده.

توی این حالت هم از همون دستور git merge استفاده میکنیم. بعد از مرج کردن یک کامیت جدید ایجاد میشه که اسمش رو خودش میگزاره مثلا: merge branch bugfix2

**دستور git commit –am “description commit” :** این دستور git add و git commit رو با هم انجام میده. و به جای description commit هم باید متن کامیت قرار بدیم.

توی حالت دوم موقع مرج کردن ممکنه با پیغام زیر روبه رو بشیم:

Auto-merging file1.txt

CONFLICT (content): Merge coflict in file1.text

Automatic merge failed; fix conflicts and then commit the result.

میگه مرج شما توی file1.txt تداخل داره. برو اونها رو فیکس کن و در نهایت یک کامیت جدید ایجاد کن.

نکته : بعد از فیکس کردن تداخل باید یک کامیت جدید ایجاد کنیم. و بعد مرج میکنیم.

**دستور:**

**git diff =>** Compare Staging Area and Working Directory

این دستور Staging Area رو با Working Directory مقایسه میکنه.

Working Directory : محیطی که داریم تغییرات رو روی فایل هامون انجام میدیم. ( مثلا با ادیتور vscode داریم یه سری تغییرات رو روی فایل ها میدیم. به این محیط میگن working Directory.

Staging Area هم محیطی هست که بعد از زدن دستور git add فایل اونجا قرار میگیرن.

**دستور git diff :** تغییرات stage و working directory مقایسه و برامون نمایش میده.

**دستور git diff –staged :**  این دستور میاد آخرین کامیت رو با استیج مقایسه میکنه.

**دستور git diff HEAD :** میاد اخرین کامیت رو با working Directory مقایسه میکنه.

**مقایسه دو کامیت :** برای این کار از دستورgit diff به صورت زیر استفاده میکنیم:

**git diff commit1..commit2**

**نکته:** بین هر دو کامیتی که بخایم میتونیم مقایسه رو انجام بدیم.

**و :** در دستور بالا نام کامیت رو قرار نمیدیم بلکه کد هش اون کامیت رو قرار میدیم.

**و:** در دستور بالا میتونیم به جای کامیت، نام شاخه ها رو قرار بدیم و دو شاخه رو با هم مقایسه کنیم.

**برگشتن به یک کامیت خاص:**

برای این کار از دستور زیر استفاده میکنیم:

git checkout هش کامیت

حالا اگر فایل ها رو ببینیم اخرین تغییراتی که داریم مربوط به کامیتی که بهش checkout کردیم هست.

دستور git switch master رو اگر بزنیم علاوه بر اینکه به شاخه master میریم موقعیت ما به اخرین کامیت هم میره، یعنی اگر خاستیم برگردیم به آخرین کامیت میتونیم از این دستور هم استفاده کنیم.

**دستور git checkout HEAD~2 :** این دستور میگه که head روی هر کامیتی که قرار داره دو تا کامیت برو قبل تر.

**دستور git restore <file> :** این دستور تغییراتی که ایجاد کردم و در working storge هست رو پاک میکنه به جای <file> باید نام فایلی که میخوایم برگرده به حالت قبل ( آخرین کامیت ) رو وارد کنیم.

**دستور git restore –stage <file> :** این دستور فایل هایی که با دستور git add وارد استیج کردیم رو از استیج بیرون میاره.

میخایم تغییرات مورد نظرمون رو از یک کامیت مورد نظر بگیریم: برای این کار از دستور زیر استفاده میکنیم:

git restore –source هش کامیت مورد نظر نام فایل مورد نظر

این دستور اطلاعات فایل مورد نظر رو برمیگرونه به اطاعاتی که این فایل در کامیت مورد نظر داشته است.

یا از دستور زیر به جای دستور بالا استفاده کنیم:

git restore –source HEAD~5 نام فایل مورد نظر

**دستور reset :**

با این دستور میتونیم کامیت ها رو پاک کنیم.

دو نوع ریست وجود دارد:

حالت اول تغییرات working directory رو برامون نگه میداره.

حالت دوم که بهش hard هم میگن که working directory هم پاک میکنه.

git reset هش کامیت

اگر این دستور رو بزنیم برمیگرده به کامیتی که بهش اشاره کردیم و تمامی کامیت هایی که بعد از اون نوشتیم رو حذف میکنه.

ولی اگر به فایل ها نگاهی بندازیم میبینیم که تغییری نکردن.

git rest –hard هش کامیت

برمیگرده به اون کامیتی که بهش اشاره کردیم با این تفاوت که اطلاعات فایل ها هم برمیگرده به کامیت مورد نظر. یعنی working directory هم حذف میشه.

git revert هش کامیت

این دستور برمیگرده به کامیتی که بهش اشاره کردیم و اطلاعات اخرین کامیت رو با کامیتی که بهش اشاره کردیم مرج میکنه و یک کامیت جدید ایجاد میکنه.

نکته : کایمت های بعد از خودش رو حذف نمیکنه.

**دستور Stash :**

فرضا ما روی یک پروژه داریم کار میکنیم و یک دفعه مجبور میشیم یک باگ رو سریع رفع کنیم و تغییراتی که اعمال کردیم برای کامیت شدن آماده نیستن هنوز. در اینجا تغییراتی که برای کامیت شدن آماده نشدن هنوز stash ( ذخیره) میکنیم .

یعنی وقتی stash میکنیم هر آنچه که در working directory هست رو برمیداره و ذخیره میکنه، بعد میایم باگ رو رفع میکنیم و بعد کدهای خودمون رو از stash میگیریم.

کارایی دیگر stash : فرضا داریم روی یک شاخه کار میکنیم و الان میخایم برگردیم به شاخه اصلی اینجا ممکنه دو اتفاق بیفته اگر تغییراتی که در شاخه فرعی دارم تداخلی با اطلاعات شاخه اصلی داشته باشه اینجا اجازه نمیده گیت که سوییچ اتفاق بیفته و میگه اول تغییراتت رو کامیت کن بعد بیا رو شاخه اصلی.

اگر تداخل نداشته باشیم اون تغییرات شاخه فرعی هم همراه سویچ میاد روی شاخه اصلی.

اینجا هم stash میکنیم و بعد سویچ میکنیم که تغییرات همراه ما به شاخه اصلی نیان.

**دستور git stash save یا git stash :** هر آنچه که در working directory هست رو برمیداره و سیو میکنه. یک هش برامون ایجاد میکنه و یک پیام. ( متن این پیام رو میتونیم خودمون تعیین کنیم.)

**دستور git stash pop :** برای برگردوندن اطلاعاتی که در stash هست از این دستور استفاده میکنیم.

دستور pop بعد از استفاده اطلاعات داخل stash رو حذف میکنه ولی ممکنه ما بخوایم این تغییرات رو در شاخه دیگه ای هم استفاده کنیم و بعد از استفاده از pop نخوایم satsh حذف بشه برای این کار از apply استفاده میکنیم.

git stash apply

حالا میخوایم چند stash داشته باشیم. برای اینکار از متنی که میتونیم برای stash بنویسیم استفاده میکنیم.

**دستور git stash list :** این دستور لیست stash های ما رو نمایش میده.

**دستور git stash apply** stash@{1} :

این دستور اطلاعات stash ی که به نامش اشاره کرده ایم ( در اینجا stash@{1} ) رو برامون میاره.

**دستور git stash drop** stash@{2} : این دستور stash ی که بهش اشاره کرده ایم رو پاک میکنه.

**دستور git stash clear :** این دستور همه stash های ما رو پاک میکنه.

**آشنایی با Git Hub :** برای اینکه افراد مختلفی بتوانند به ریپازیتوری ما وصل بشن و اون رو ببینن یا تغییر بدن باید ریپازیتوری رو در بستر وب قرار بدیم برای این کار از git hub استفاده میکنیم.

**دستور git clone :** برای دانلود سورس کدها از گیت هاب از git clone استفاده میکنیم. این دستور مربوط به گیت هست نه گیت هاب.

نکته: ریپازیتوری های private رو میتونیم کاری کنیم چند نفر اون رو ببینن.

برای انتقال ریپازیتوری به گیت هاب دو حالت اتفاق می افته:

یک : زمانی که ریپازیتوری جدیدی درست کردیم و تازه میخایم پروژه رو ایجاد کنیم در این حالت بعد از ایجاد ریپازیتوری در گیت هاب، یک فولدر در جای مورد نظر ایجاد میکنیم و داخلش دستور git init رو میزنیم و بعد از اون پوشه خالی رو میتونیم بیاریم داخل گیت هاب و حالا بعد از این وقتی تغییراتی در ریپازیتوری لوکال انجام میدیم با دستور push اون رو میفرستیم بالا.

دو: زمانی که پروژه از قبل داریم و داخل گیت هم باهاش کار کردیم، کامیت های خودمون رو داریم، شاخه های خودمون رو داریم و حالا میخایم بزاریمش روی گیت هاب، برای این حالت هم از دستور git push مستقیم استفاده میکنیم.

**دستور git remote :** ریموت در واقع یک مقصد رو به ما نشون میده، ما وقتی میخایم پروژه رو بزاریم بالا وقتی میخایم push بزنیم این ریپازیتوری کجا باید بره قرار بگیره، باید اول مشخص کنیم وقتی git push میزنیم پروژه و فایل های ما روی کدام ریپازیتوری قرار بگیره، برای همین از دستور git remote و بعدش آدرس ریپازیتوری استفاده میکنیم. این دستور در واقع ریموت هایی که برای پروژه داریم رو برمیگردونه.

**وصل کردن پروژه به گیت هاب با ریموت :**

git remote add <name> <url>

url که مقصد ما هست، name هم یک برچسب هست که هرکجا از پروژه بخایم url صدا بزنیم name رو به جاش صدا میزنیم.

نکته: Name دلبخواه هست.

**دستور git remote –v :** این دستور url ای که برای pull و push استفاده میشه رو بهمون نشون میده.

**دستور git push :**

git push <remote> <branch>

بعد از دستور git push اول نام remote رو وارد میکنیم و بعد نام شاخه.

بعد از درست کردن شاخه هم باید اون شاخه رو push کنیم تا بتونیم در گیت هاب اون رو ببینیم.

git push <remote> <branch>

Readme.md : این فایل برای نوشتن توضیحاتی در مورد ریپازیتوری در گیت هاب است، پسوند این فایل md هست، اگر markdown رو سرچ کنیم نحوه نوشتن در این فایل را خواهیم دید.

نحوه ایجاد Readme : اگر این فایل رو در ریپازیتوری نداشته باشیم اول در پروژه یک فایل با این نام درست میکنیم و اطلاعات لازم رو داخلش مینویسیم.

**سرویس github page :** این سرویس به ما این قابلیت رو میده که یک سایت استاتیک برای ریپازیتوری خودمون داشته باشیم. استاتیک یعنی فقط html ، css و javascript هست و زبان بک اند نداره. که میتونیم داکیومنت اضافه کنیم، دموی پروژه رو داخلش قرار بدیم.

برای این کار وارد setting ریپازیتوری میشیم، روی منوی pages کلیک میکنیم، باید فایل های مورد نیاز برای نمایش سایت استاتیک رو ساخته باشیم و در این قسمت اون رو انتخاب کنیم.

**Remote Tracking Branch :** در واقع یک اشاره گر هست، که به محل شاخه ما توی ریپازیتوری گیت هاب اشاره میکنه، حالا برای اینکه گیت توی ریپازیتوریمون بتونه تشخیص بده که master الان به کدام کامیت داره اشاره میکنه توی گیت هاب، از origin/master استفاده میکنه،

برای اینکه به این ریموت برنج ها دسترسی داشته باشیم از دستور زیر استفاده میکنیم :

git branch –r

این به ریموت برنج های ما اشاره میکنه،

فرض کنید پروژه ای رو از گیت هاب clone کرده ایم و این پروژه در گیت هاب دارای چند شاخه باشد حال اگر بعد از clone کردن دستور git branch رو روی پروژه اجرا کنیم فقط شاخه master رو به ما نمایش میده، یعنی فقط برنج اصلی،

حال اگر دستور remoter branch –r رو بزنیم میبینیم که ریموت های همه برنج ها رو نشون میده، حال اگر بخوایم روی شاخه های دیگه سویچ کنیم از همون دستور switch استفاده میکنیم، و اگر این شاخه در گیت هاب وجود داشته باشه به صورت خودکار روی این برنج سویچ میشیم.

نکته: فرضا ما میخواهیم کامیتی را از لوکال به ریپازیتوری خود push کنیم اما هم تیمی ما چند کامیت ایجاد کرده و اونها رو push کرده، پس الان شاخه ای که ما داریم چند کامیت از شاخه ای که در git hub هست عقب تر است،

اینجا برای اینکه به مشکل نخوریم باید قبل از اینکه تغییرات خودمون رو push کنیم باید کامیت های شاخه در گیت هاب رو دریافت کنیم،

**دستور git fetch :** این دستور مانند git pull برای گرفتن اخرین تغییرات روی گیت هاب است، با این تفاوت که دستور git pull تغییرات رو مستقیما به working directory ما میاره ولی دستور git fetch تغییرات رو میاره روی local Repository.

Local repository محیطی هست بعد از stage area.

git fetch <remote> <branch>

اگر بخوایم تغییرات یک شاخه خاص رو بگیرم اسم شاخه رو مینویسیم ولی اگر ننویسیم همه تغییرات رو برمیگردونه.

اگر از دستور git fetch استفاده کردیم برای اینکه تغییرات رو بیاریم به working directory از دستور git checkout <remote> <namebranch>

استفاده میکنیم.

نکته: زمانی که fetch اتفاق افتاده برای اینکه بتونیم کامیت هایی که روی گیت هاب داشتیم ولی روی پروژه لوکال خودمون نداشتیم رو مشاهده کنیم از دستور :

git log –oneline –all

استفاده میکنیم.

حالا میتونیم اون تغییراتی که داشتیم رو بررسی کنیم اگر تغییرات اوکی بودن از git pull استفاده میکنیم git pull مرج رو برامون انجام میده،

**دستور git pull :** در واقع git fetch رو انجام میده به اضافه اینکه مرج هم انجام میده.

git pull <remote> <namebrach>

نکته : در تمام جاهایی که دستورات به این صورت هست که ریموت و نام شاخه رو از ما میخواد اگر نام ریموت origin باشه لازم نیست نام ریموت رو بنویسیم و اگر namebranch رو ننویسیم دستور برای شاخه ای که روش هستیم اتفاق می افته. مثلا در دستور بالا فرضا ما روی شاخه master هستیم و دستور رو به صورت زیر مینویسیم :

git pull

این دستور میاد از شاخه mater اخرین تغییرات رو pull میگیره.

**کانفلیکت :**

زمانی که کانفلیکتی وجود داشته باشه و از دستور git pull استفاده میکنیم به ما میگه که کانفلیکت وجود داره و باید این کانفلیکت رو برطرف کنی.

**گیت هاب :**

زمانی که ریپازیتورای از نوع private درست میکنیم اگر فرد دیگری بخواد وارد این ریپازیتوری بشه با ارور 404 مواجه خواهد شد. مگر اینکه مشخص کنیم چه کسانی حق دسترسی داشته باشند.

حالا میخوایم یک اکانتی رو به ریپازیتوری اضافه کنیم که بتونه پروژه ما رو ببینه، تغییراتی رو خواست اضافه کنه و پوش کنه.

برای این کار وارد setting ریپازیتوری میشیم، وارد منوی Manage access میشیم، روی Add people کلیک میکنیم، و باید username یا fullname یاemail فرد مورد نظر را اضافه میکنیم.

بعد از اینکار یک درخواست برای فردی که به پروژه اضافه کرده ایم ارسال میشود که بیا و توی پروژه شریک شو، و اون فرد باید این رو تائید کنه.

نکته : setting مربوط به ریپازیتوری فقط در دسترس فرد owner هست کسی که ریپازیتوری رو ساخته و اشخاص دعوت شده به پروژه دسترسی به setting ریپازیتوری رو ندارند.

فرضا در ادامه مثال بالا شخصی که به پروژه دعوت کرده ایم یک شاخه جدید درست کرده و تغییراتی روی پروژه اعمال و push کرده و از هم تیمی ها میخواهد که تغییرات اون رو مرج کنند.

وقتی که اون شخص برنج جدید رو میسازه تغییرات رو اضافه و کامیت میکنه، حالا که میخوان اون تغییرات را push کنه ( برنج هم اینجا میخواهد push بشه) ارور زیر رو خواهد دید:

fatal : The current branch categories has on upstream branch.

To push the curent branch and set the remote as upstream, use

git push –set-upstream origin categories

این ارور میگه شاخه categries رو روی گیت خودت اضافه کردی ولی توی ریموت این شاخه رو نداری. و اگر میخوای این رو push کنی به صورت خط اخر دستور رو بنویس. که به جای اینکه از دستور خط اخر متن ارور استفاده کنیم میتونیم از دستور زیر هم استفاده کنیم:

git push –u origin categories

**pull request :**

وظیفش این هست که زمانیکه یک شاخه جدید رو ایجاد میکنیم و تغییرات رو اعمال کردیم و میخوایم به بقیه بگیم بیان این شاخه رو ببینن و مرج رو انجام بدن، ( به نحوی اطلاع رسانی انجام میده برای مرج کردن).

این موضوع مربوط به گیت هاب هست نه گیت.

وقتی که یک شاخه جدید رو اضافه میکنیم و قرارش میدیم روی گیت هاب بالای ریپازیتوری یک دکمه به نام Compare & pull request اضافه میشه. بعد از زدن این دکمه صفحه ای باز میشود که بالای صفحه مشخص میکنیم که چه شاخه ای قرار است با چه شاخه ای مرج شود و بعد از انتخاب شاخه به ما نشون میده که آیا دارای کانفلیکت هستیم یا خیر. بعد یک متن باید براش در نظر بگیریم که در حالت پیش فرض متن کامیت ما هست . از طرفی یک کامنت باید براش در نظر بگیریم. میتونیم کامل توضیح بدیم که توی این شاخه ای که میخایم اضافه کنیم چه ویژگی هایی اضافه شده. بعد از این روی دکمه create pull request کلیک میکنیم. این درخواست برای مدیر پروژه ارسال میشه شاخه ما رو pull میکنه و اگر اوکی بود اون رو مرج میکنه با شاخه اصلی.

**Pull request زمانی که تداخل وجود داشته باشد :**

اگر تداخل در شاخه ای که میخواهیم برای مرج اطلاع رسانی کنیم وجود داشته باشد هنگام ساخت pull request به ما میگه که کانفلیک وجود داره ولی با این وجود میتونیم درخواست رو ارسال کنیم.

**تغییر وضعیت ریپازیتوری از private به public و بلعکس :**

برای این کار وارد settings در ریپازیتوری میشیم و قسمت Options در این صفحه در قسمت Danger Zone روی دکمه Change visibility کلیک میکنیم و در اینجا میتونیم سطح دسترسی رو تغییر بدیم.

ما میتونیم در گیت هاب یک سری قوانین برای شاخه ها داشته باشیم، برای ریپازیتوری private نیاز هست تا اکانت خریداری بشه ولی برای دسترسی public یک سری امکانات رو میتونیم داشته باشیم.