**بسم الله الرحمن الرحیم**

git config --global user.name “[firstname lastname]”

set a name that is identifiable for credit when review version history

git config --global user.email “[valid-email]”

set an email address that will be associated with each history marker

git status

show modified files in working directory, staged for your next commit

git init

initialize an existing directory as a Git repository

git add [file]

add a file as it looks now to your next commit (stage)

git commit -m “[descriptive message]”

commit your staged content as a new commit snapshot

git log

show all commits in the current branch’s history

دستور git log –oneline تمام Commit ها را به صورت یک خط می آورد

برای اینکه فایلی را مستقیماَ add و سپس commit کنیم از دستور ترکیبی زیر استفاده می کنیم

Git commit -a -m “Message”

**روش برگرداندن فایل از stage به working directory :**

با استفاده از دستور git reset head می توان یک فایل را از stage به حالت Working directory برد

**روش برگرداندن فایل قبل از مرحله stage به آخرین commit :**

برای این کار از دستور زیر استفاده می کنیم

Git restore <file name>

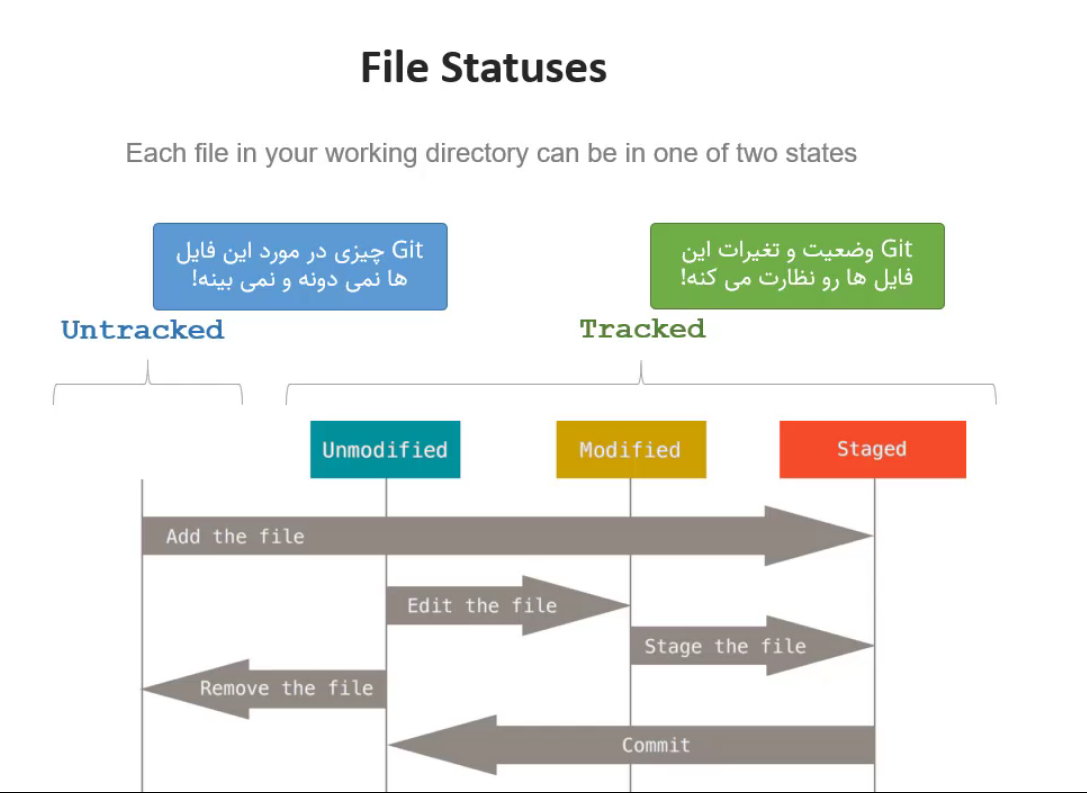
**-**

**بسم الله الرحمن الرحیم**

**درس : آشنایی با وضعیت فایل ها و Life cycle**

**وضعیت فایل ها در Git :**

هر فایل در Git دو وضعیت به صورت زیر دارد



**Untrack :**

اگر فایلی را داشته باشیم که بخواهیم آن را از حالب Untrack به حالت track که Stage است ببریم از دستور زیر استفاده می کنیم

Git add نام فایل

اگر به جای نام فایل علامت . بزنیم کل فایل ها به حالت Stage می رود

**Unmodified :**

فایل به Git معرفی شده ولی از آن به بعد هیچ تغییری به آن اضافه نشده است

**Modified :**

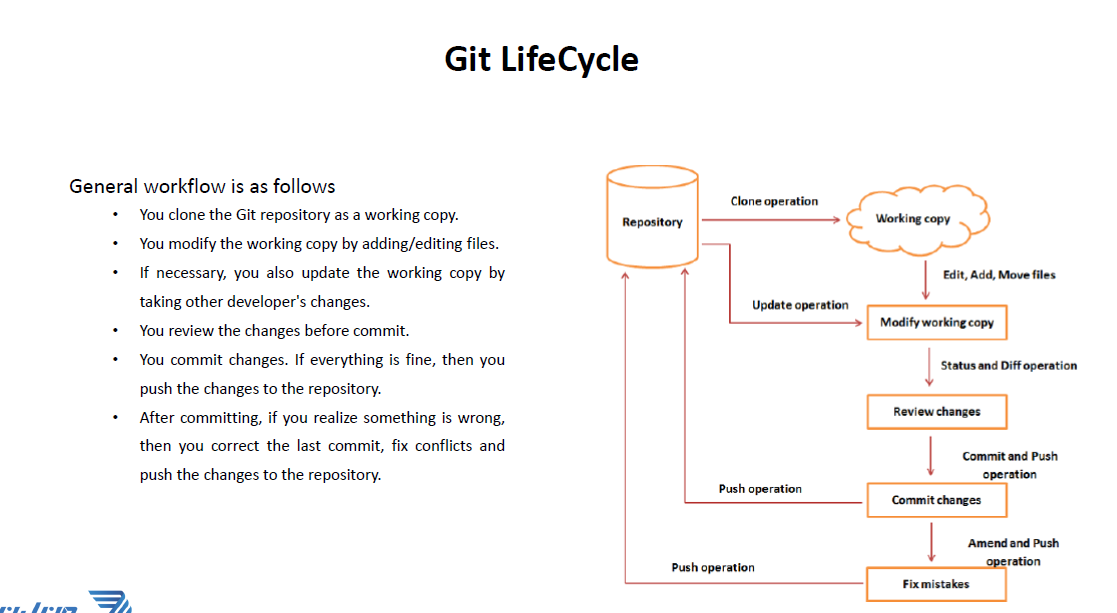
فایل Modified یک فایل unmodified است که با هر تغییری به حالت modified تغییر پیدا می کند

**Stage :**

هر فایلی که بر روی آن Add به کار می رود به حالت Stage می رود که منتظر است یک Commit بر روی آن زده می شود

**Commit :**

می توان بوسیله آن یک نسخه ثبت کنیم در Branch که داریم و نسخه هایی که ثبت می شوند قابل برگشت هستند



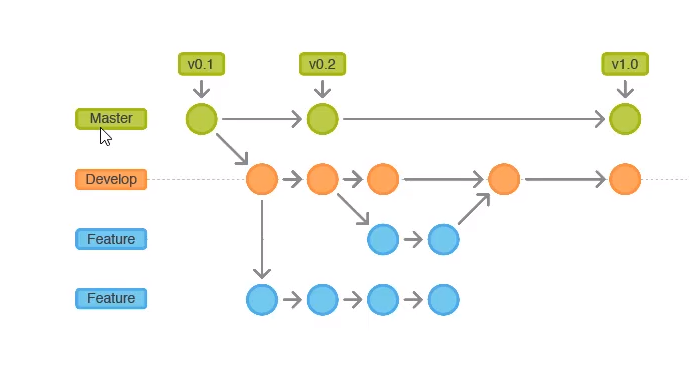
Read This

<https://www.javatpoint.com/git-terminology>

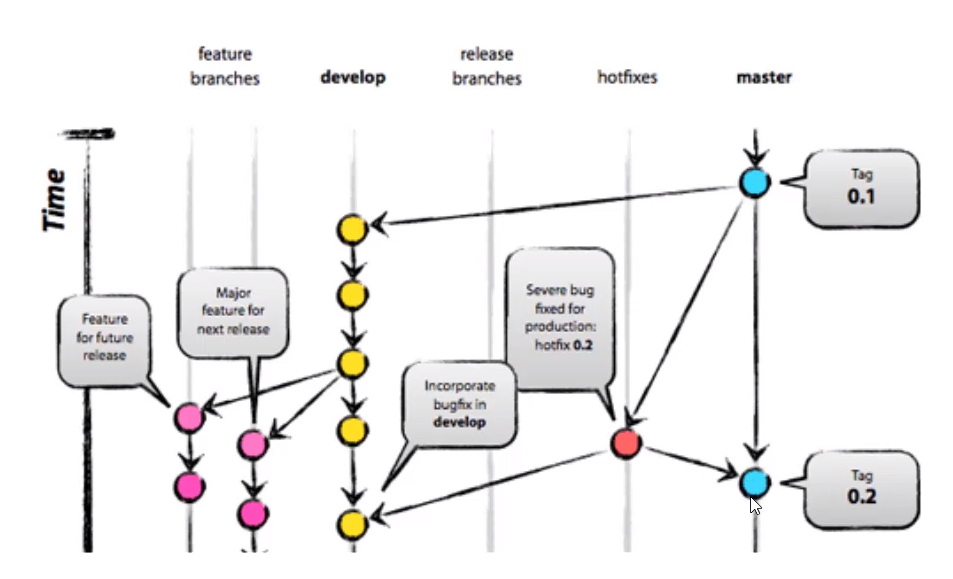
**بسم الله الرحمن الرحیم**

**درس : ایجاد و مدیریت انشعاب ها در پروژه**

هنگامی که یک پروؤه را با Git init به Git معرفی می کنیم برای ما یک شاخه Master می سازد ولی به دلیل کار Developer و سایر برنامه نویسان نیاز است که شاخه های دیگری را تولید کنیم و پس از اتمام کار برنامه نویسان آن را با شاخه Master ادغام کنیم



معمولاّ آخرین نسخه نرم افزار بر روی شاخه Master قرار دارد



برای دیدن Branch ها در یک پروژه از دستور

Git branch

اگر بخواهیم تمام branch ها را ببینیم و حتی branch های online را ببینیم از دستور زیر استفاده می‌کنیم

Git branch -a

اگر بخواهیم یک branch را بسازیم و مستقیم به آن switch کنیم از دستور زیر استفاده می‌کنیم

Git checkout -b <branch name>

استفاده میکنیم

توجه شود که از هر شاخه ای branch جدید را بسازیم آن شاخه مبنای اصلی برای branch جدید است

ساختن یک Branch جدید :

Git Branch Branch name

وقتی یک Branch جدید می سازیم از آخرین commit که انجام می‌شود آن Branch را در نظر می‌گیرد

جهت رفتن از یک Branch به یک Branch جدید از دستورbranch name git checkout استفاده می کنیم

برای پاک کردن یک Branch از دستور git branch -d branch name استفاده می کنیم

اگر بخواهیم یک Branch را در داخل یک Branch دیگر Merge کنیم ابتدا به Branch مقصد می‌رویم سپس با دستور

git merge merge source آن Branch را ادغام می کنیم

برای تغییر نام یک branch از دستور زیر استفاده می‌کنیم

Git branch -m <branch name> <new branch name>

**بسم الله الرحمن الرحیم**

**درس : رفع تداخل ها (Fix conflit) :**

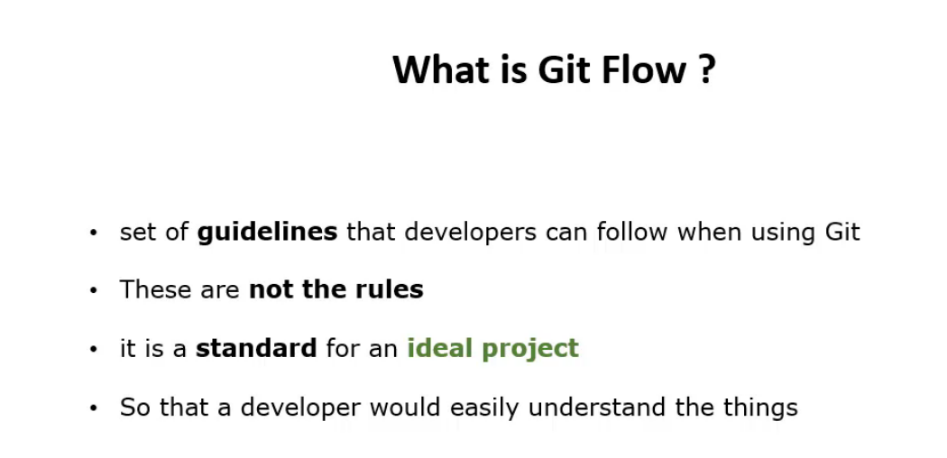
اگر دو شاخه بر روی یک موضوع تغییرات را ایجاد کرده باشند در هنگام merge کردن پیغام conflict می دهد که با مراجعه به اصل برنامه و پاک کردن قسمتی که مورد نیاز نمی باشد می توان خطا را برطرف نمود و سپس مجدد فایل را add و سپس commit کرد

**بسم الله الرحمن الرحیم**

**درس : مدل برنچینگ بهینه (Git Flow):**

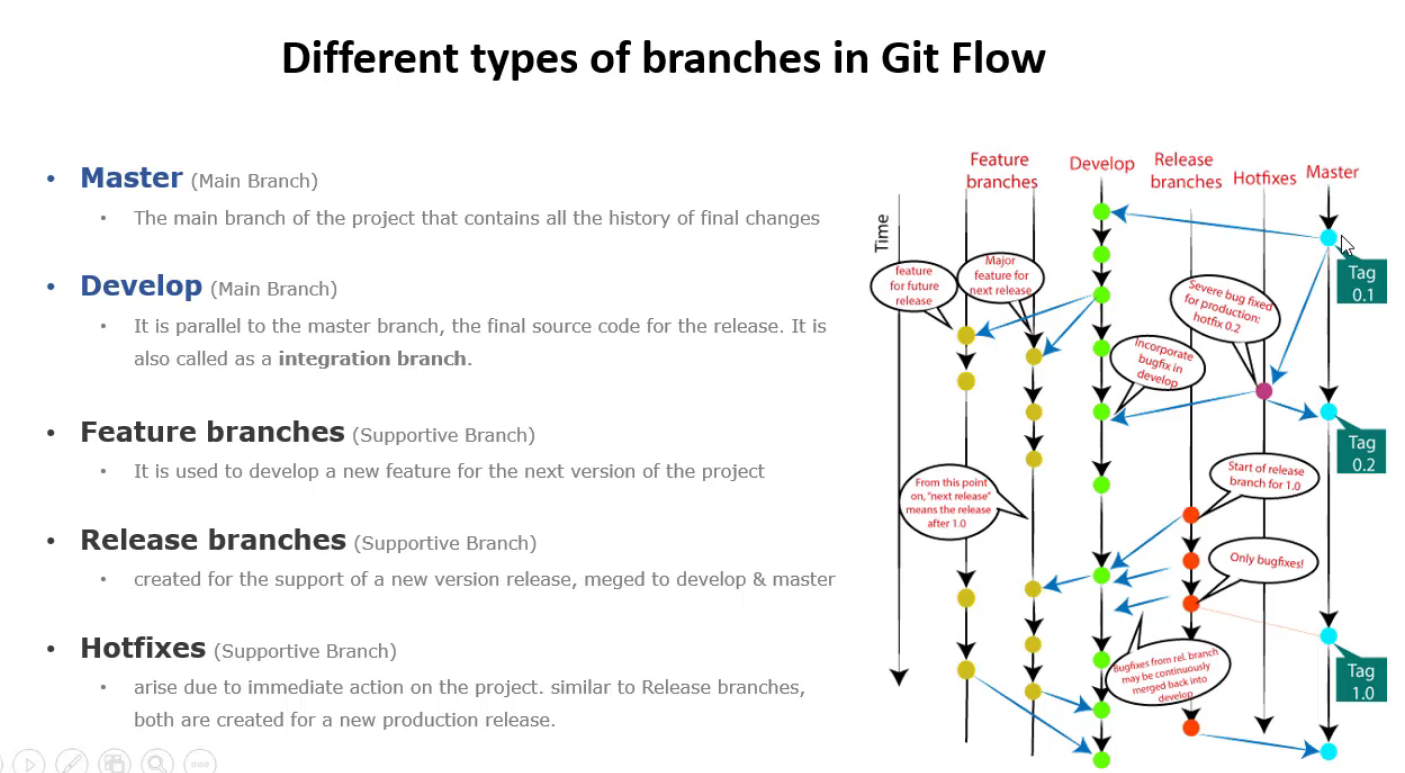
Git flow مجموعه استانداردها برای استفاده از git است

Git flow قانون نیست ولی یک استاندارد است که روند سورس کد را به سمت مدل ایده آل می‌برد



مدل قبلی Branchig برا اساس Feature نیست و نمی توان به نقطه خاصی اشاره کرد

در Branchig مدل Git flow ما پنج نوع Feature داریم

1. Master که Branch دائمی است حاوی آخرین نسخه از نرم افزار است که توسط کاربران استفاده می‌شود این یک Branch اصلی است
2. Developer که Branch دائمی است این Branch حاوی آخرین تغییرات توسط Developer ها است که در Release بعدی ارائه به کاربر می شود و جزء Branch های اصلی هستند
3. Feature که Branch پشتیبان است صرفاّ در زمانی که ما یک Feature را توسعه می دهیم به کار می رود بعد از توسعه یا برطرف کردن Feature می توان آن را با Develope یکی (merge) نمود و پاک کرد
4. Realease حاوی تعدادی زیادی از Feature Branch هستند Branch پشتیبان است که آخرین نسخه نرم افزار است که در Develope یا Master قرار میگیرد
5. Hot fix که Branch پشتیبان است زمانی که نسخه ای را منتشر کردیم و اتفاقی می افتد که نیاز به اقدام سریع داریم برای رفع آن یک Hot fix Branch ایحاد می کنیم
6. 

http://www.javatpont/git-flow

**بسم الله الرحمن الرحیم**

**درس :ثبت و مشاهده تغییرات در Git :**

برای مشاهده تغییرات در یک فایل از دستور زیر استفاده می کنیم

Git show (ID)

که در آن ID به محل commit مورد نظر ما اشاره دارد

برای مشاهده و مقایسه تغییرات در دو Branch مختلف از دستور زیر استفاده می کنیم

Git diff (source branch name)..(Target branch name)

با استفاده از افزونه Git History در vscode می توان تاریخچه را مشاهده کرد

با استفاده از دستور git diff می توان تغییرات یک فایل commit نشده با مخزن را ببینیم

با استفاده از دستور git diff –staged می‌توان تغییرات یک فایل commit شده با مخزن را ببینیم

**بسم الله الرحمن الرحیم**

**درس : برگرداندن و ریست کردن پروژه به یک کامیت خاص :**

برای برگرداندن پروزه به یک Commit خاص از دستور زیر استفاده می کنیم

Git reset –hard (id)

اگر از –hard استفاده کنیم پروژه به طور کلی ریست می شود و تمام فایل های اضافه پاک می‌شود

اگر از –hard استفاده نکنیم پروژه فایل ها را ریست نمی کند و نگه می دارد

**بسم الله الرحمن الرحیم**

**درس : آموزش stashing در git :**

این ابزار برای نگه داشتن پروژه های نیمه کاره که شرایط commit را ندارند از stashing استفاده می‌شودپ

برای اینکار ابتدا پروژه را add کرده و سپس با استفاده از دستور git stash آن را stash می کنیم

برای استخراج لیست stash ها از دستور git stash list استفاده می کنیم

برای پاک کردن یک branch از دستور git branch -D branch name استفاده می‌کنیم برای پاک کردن آن نباید در داخل آن Branch بود

Git stash drop شماره ID برای یک stash مورد نظر باعث می شود که آن stash پاک گردد

برای مشاهده یک stash که داشتیم کار میکردیم ببینیم از دستورID stash git show

برای برگشتن به یک stash از دو روش استفاده می‌کنیم

1. Git stash apply که با این دستور کلیه تغییرات اعمال می شود ولی stash قبلی می‌ماند
2. Git stash pop stash قبلی از بین می رود

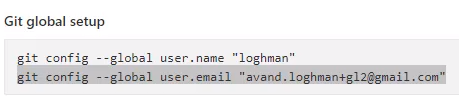
برای پاک کردن لیست تمام stash ها از دستور git stash clear

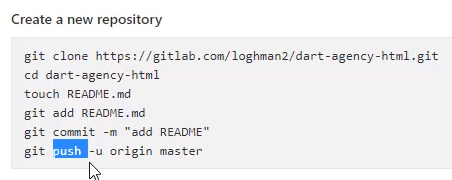
می توان از دستو ر git stash save “Notic” برای stash کردن استفاده نمود

Stash ها را در Branch های دیگر هم می توان به کار برد

**بسم الله الرحمن الرحیم**

**درس : معرفی config و استفاده از gitlab :**







**پاک کردن یک فایل با استفاده از git**

برای اینکار از دستور git rm <Filename> استفاده می کنیم

اگر یک فایل را بخواهیم از مخزن پاک کنیم و لی اصل فایل بماند از دستور زیر استفاده می کنیم

git rm –cached <Filename>

اگر بخواهیم فایل ها را از مخزن پاک کنیم که تو در تو هستند از دستور زیر استفاده می کنیم

git rm –cached -r <Filename>

**تغییر نام یک فایل با استفاده از git :**

Git mv (old file name) (new file name)

**با استفاده از این دستور می توان یک فایل را به پوشه فرستاد**

Git mv (old file name) dir/directory name

\

**تا استفاده از این دستور می توان یک فایل را به Working directory برد**

Git reset –mixed (code commited)

فتوجه به این نکته ضروری است که این دستور تنها فایل را به آخرین Commit می برد

اگر بخواهیم فایل را به آخرین تغییرات ببریم از دستور زیر استفاده می‌کنیم

git checkout .

**نادیده گرفتن بعضی فایل ها توسط Git :**

اگر بخواهیم بعضی فایل ها توسط Git دنبال نشود در ابتدا یک فایل به نام .gitignore را می‌سازیم

برای این کار از دستور .gitignore استفاده می‌کنیم و سپس نام فایل مورد نظر را در آن اضافه می‌کنیم به این نکته توجه شود که این فایل هم add و هم commit شود

اگر بیک فولدر داشتیم و در داخل آن تعدادی فایل برای اینکه بتوانیم آن فولدر و فایل های آن را دنبال نکنیم در داخل فولدر .gitignore نام آن فولدر را می‌رنیم و در انتهای آن علامت / را قرار می‌دهیم

اگر بخواهیم گروهی از فایل ها را دنبال نکند در داخل فایل .gitignore ابتدا علامت \* را زده و سپس . و پسوند فایل را می‌نویسیم

اگر بخواهیم که فایلی را که در مرحله گروهی نسبت به عدم دنبال کردن آن اقدام کنیم در ابتدا در فایل .gitignore علامت ! را زده و سپس نام فایل با پسوند مورد نظر را وارد می‌کنیم

در صورتیکه بخواهیم یک دسته ای از فایل ها که نیاز به دنبال کردن ندارند ولی توسط برنامه های کامپایلری ایجاد می شوند و این فایل ها مورد نیاز نیستند را دنبال نکنیم از سایت .gitignore استفاده می کنیم و محتویات فایل تولید شده را اضافه به فایل .gitignore اضافه می‌کنیم

اگر فایلی را از قبل توسط Git دنبال کنیم و سپس آن را به .gitignore اضافه کنیم git همچنان به دنبال کردن آن ادامه می دهد

دلیل این موضوع این است که قبلا یکبار آن را add کرده ایم برای این که بتوانیم آن را دنبال نکنیم باید در ابتدا آن را از داخل مخزن خارج کنیم

اگر بخواهیم تعداد مشخصی از commit ها را از آخرین commit ببینیم می توانیم از دستور زیر استفاده کنیم

Git log -Number of commit

Git log –oneline -number of commit

**ارسال پروژه به مخزن آنلاین :**

اضافه کردن یک آدرس به ریموت برای ارسال پروژه به مخزن آنلاین :

Git remote add <favorite name (origin) > url

جهت دیدن نام ریموت مورد نظر از دستور

Git remote

برای قطع ارتباط با یک مخزن از دستور زیر استفاده می کنیم

git remote remove <favorite name (origin) >

جهت دیدن دستور url ها از دستور زیر استفاده می‌کنیم

Git remote -v

برای فرستادن یک پروژه از مخزن داخلی به مخزن آنلاین از دستور زیر استفاده می کنیم

Git push -u <favorite name (origin) > Branch name

اگر بخواهیم تمام شاخه ها را به یک مخزن آنلاین ارسال کنیم

Git push -u <favorite name (origin) > --all

اگر پروژه ای را داشته باشیم و آن پروژه توسط شخص دیگری تغییر کرده باشد برای اعمال تغییرات پروژه بر روی پروژه اصلی از دستور زیر استفاده می کنیم

Git pull <favorite name (origin) >

توجه به این نکته ضروری می باشد که پس از pull کردن با شاخه مورد نظر مستقیم merge می شود

پس از تغییرات در سیستم خودمان مجدداّ آن را push می‌کنیم

**دستور fech :**

این دستور باعث می شود که تغییرات مستقیما در پروژه به صورتMerge در نیاید بلکه در ابتدا به همان branch مورد نظر که به صورت remote است برود و سپس بتوان آن را merge کرد توجه شود که حتماّ از شاخه origin عملیات merge صورت پذیرد

**ساخت یک Branch در محیطhub git :**

اگر یک Branch در محیط git بسازیم می توانیم آن را با pull و یا fetch به مخزن داخلی خودمان انتقال دهیم ولی همچنان در لیست Branch ها نیست ولی اگر checkout را بزنیم به داخل آن branch می رویم و مستقیم به مخزن داخلی اضافه می شود

**پاک کردن یک Branch از داخل git hub :**

Git push <favorite name (origin) > --delete <branch name>

توجه شود که صرفا از git hub پاک می شود درس 30 07:24

**انتقال یک پروژه از روی یک مخزن آنلاین به یک سیستم :remo**

برای اینکار از دستور زیر استفاده می‌کنیم

Git clone <git online address>

پس از اجرای این دستور فقط Branch master انتقال می یابد برای انتقال سایر branch ها از دستور زیر استفاده می کنیم

Git checkout <branch name>

**Fork :**

اگر در یک پروژه که open source است اگر همه بخواهند روی آن کار کنند برای اینکه کسی بتواند روی این پروزه کار کند باید آن را fork کند

توجه شود که این پروژه صرفاّ بر روی مخزن پروژه ما می رود

16:18