**Git-Tutorial**

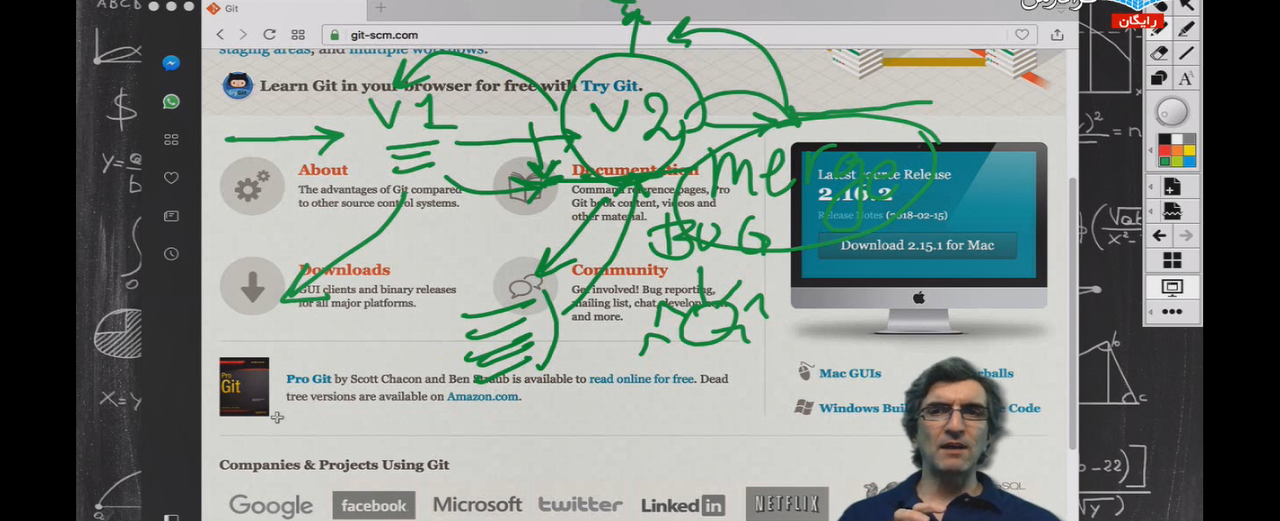
**Faradars (Jadi) :**

**Video1 :**

**گیت رو آقای لینوس توروالدز نوشته (هم گیت رو و هم کرنل لینوکس رو ایشون نوشته ☺)**

**گیت در واقع یک distributed version control system(VCS) است و مثلا وقتی یک مقاله داریم مینویسیم یا داریم یک سایت طراحی میکنیم ، به چه مشکلی بر میخوریم ؟‌ خب یکسری فایل داریم میریزمشون در یک دایرکتوری بعدش همه رو کپی میکنیم تو یه new\_Folder ، بعدش که کارمون تموم شد یه پروژه میسازیم به اسم Final وبعد همه رو اون تو کپی میکنیم . بعد از یه مدت پروژه رو میخواهیم Edit کنیم پس حالا دیگه به فولدرمون تاریخ هم اضافه میکنیم که مثلا در فلان تاریخ ادیت و ویرایش شده !اما ..... بعد از یه مدت حتما گیج میشیم. مثلا اگه ۲ تا برنامه نویس باشیم که داریم روی یه پروژه کار میکنیم با این distributed ها مشکلمون حل میشه.**

**توجه : کاری که گیت میکنه اینه که پروژه ها رو قسمت قسمت میکنه . در واقع میاد میپرسه که آقا هر موقع رسیدی به version 1 در پروژه ات . حالا بیا به من هم بگو بدونم.اگه مثلا در حین اینکه داری version2 ر مینویسی ، رییست گفت بروتو همون version 1 و یکچیزی رو تصحیح کن گیج نمیشی و به گیت میگی برو به ورژن قبلی و این رو در سرور کپی کن .و اما اگه به یه باگ برخوردی به گیت میگی که برو به ورژن ۱ و یک نسخه رفع باگ مینویسی و مشکل رو حل میکنی و بعدش اون رو با ورژن ۲ merge کن و حالا دیگه ورژن ۲ ایرادی نداره . دلیلی که گیت خوبه اینه که اجازه میده پروژه شما شاخه شاخه بشه. و اسم و ورژن بخورن.**



**خب برای شروع (البته بعد از نصب گیت ) ، باید یک فایل را انتخاب کنیم (ما اینجا به دایرکتوری tmp میرویم ) و فایل گیت خودمون رو اونجا بسازیم . مثال :‌**

**cd /tmp**

**وبعد برای اینکه یک فایل بسازیم دستور مقابل رو میزنیم :**

**mkdir project**

**خب حالا باید برویم به همان فایل project .پس میزنیم :**

**cd project**

**خب حالا اول برای اینکه بخواهیم با گیت کار کنیم دستور مقابل رو میزنیم‌:**

**git init**

**به ما این پیغام رو میده ‌:**

|  |
| --- |
| **Initialized empty Git repository in C:/Users/asus/AppData/Local/Temp/.git/**  **واگر که دوباره git init بزنیم میگه :** |
| **Reinitialized existing Git repository in C:/Users/asus/AppData/Local/Temp/project/.git/** |

**جایی که ریخته اینجاست :**

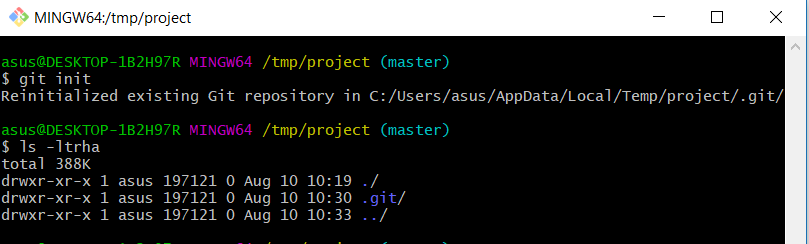


**که به جای existing ابتدا به ما میگه empty اما اینجا یه بار دیگه initialize ش کرده بودم .**

**وقتی کد مقابل رو بزنیم به ما میگه که چی رو ساخته :**

**ls –ltrha**

**و پیغام زیر رو نشون میده :**



**خب حالا میزنیم کد مقابل رو تا یه دایرکتوری .git برای ما در اون مکان درست کنه که البته ابتدا hidden هست و باید unhide ش کنیم .که البته یکسری فایل درست میکنه که ما به اونها هیچ کاری نداریم :**

**ls –ltrh .git**

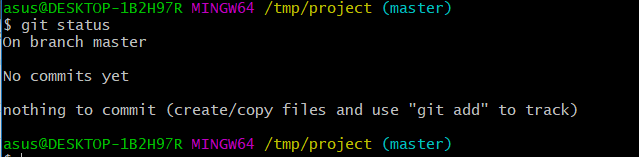
**توجه :‌با دستور clear هم کل صفحه را پاک میکنیم .**

**پس وقتی git init رو میزنیم گیت به ما میگه که از این به بعد این دایرکتوری تحت نظر گیت هست .**

**یک دستور خیلی معروف و مهم که وضعیت رو به ما میگه اینه :‌**

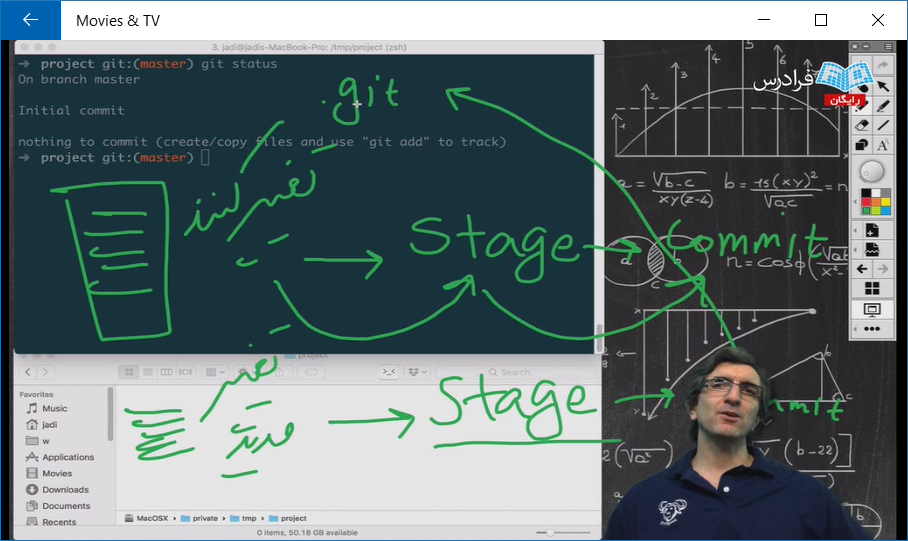
**git status**

**و پیغام زیر رو به ما نشون میده :**



**که داره میگه که شما روی branch master هستید و هیچ commit ای ندارید . در واقع میگه شما یه دایرکتوری دارید و فایل ها اینجا هستند و میونند تغییر کنند و وقتی شما فایل ها رو تغییر میدید اونها میروند به حالت stage(صحنه ، لبه) . یعنی آماده یه چیزی میشوند به اسم commit کردن .در واقع با زدن دستور status گیت میفهمه که این فایل ها رفته اند به حالت stage و با یه دستور دیگه هم commit میشوند و در واقع میروند به آرشیو گیت(همان .git) و یادش میمونه که این فایله stateاش این بود ها.**

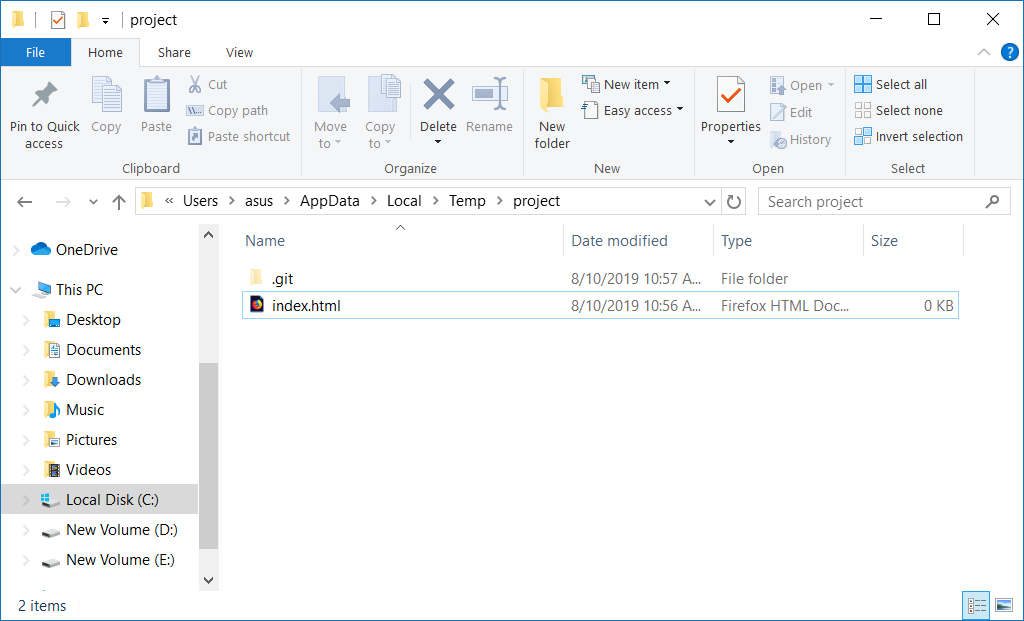
**Video2 :**



**خب حالا یک index.html در project ( یعنی در کنار .git)میسازیم .با دستور مقابل :**

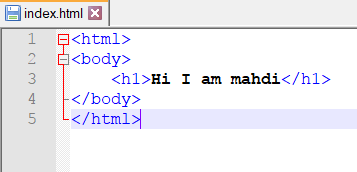
**touch index.html**

**و این طور میشود :‌**

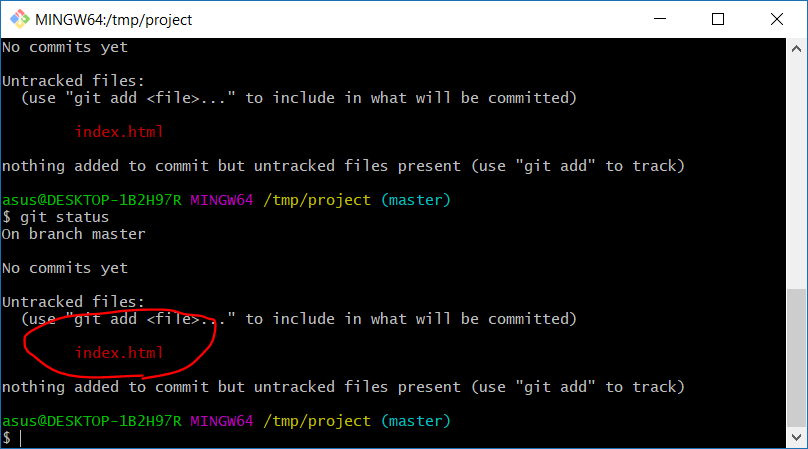


**اون رو با یک تکست ادیتور باز میکنیم :**

**و یک متن به زبان html رو در اون مینویسیم :**

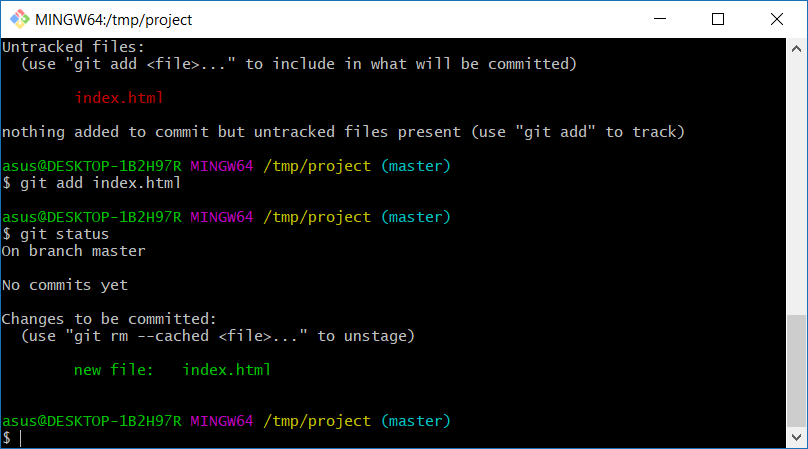


**حالا اگه git status رو بزنیم ، به ما پیام زیر را میدهد :**



**که داره میگه که یک فایل untracked داری ، یعنی فایلی که گیت اون رو دنبال نمیکنه و پیشنهاد هم داده که برای اینکه به گیت اون رو اضافه کنی از دستور git add استفاده کن .**

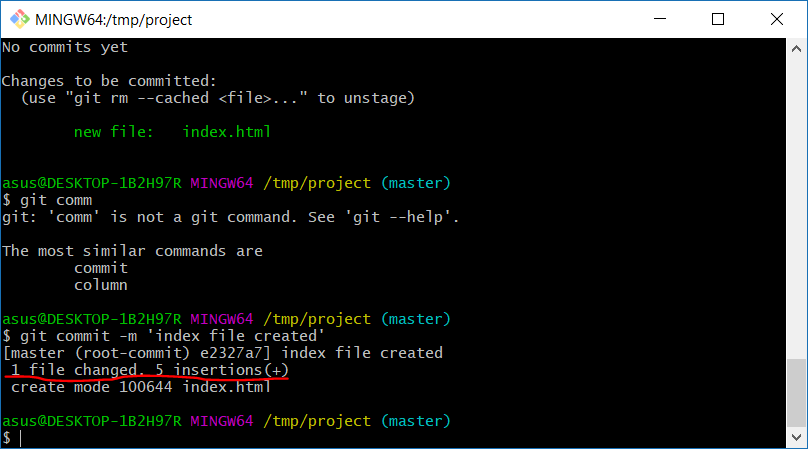
**پس ما هم git add رو میزنیم وبعد status میگیریم ، پیغام زیر رو نشونمون میده :‌**



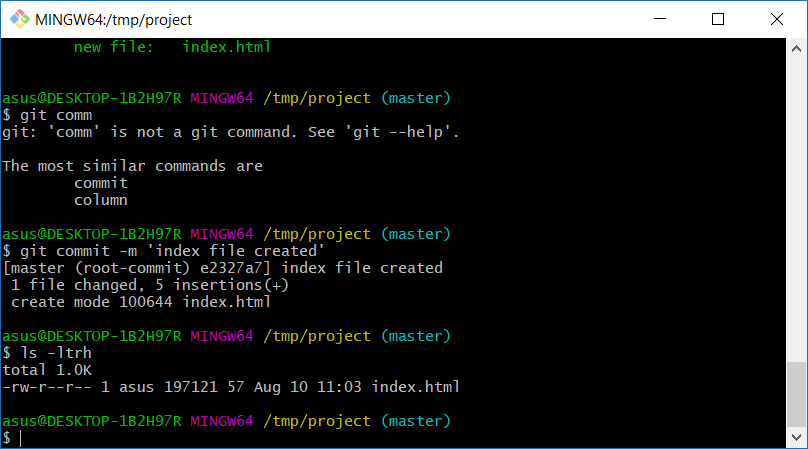
**که این دفعه میگه حالا یه new file داری .**

**حالا باید دستور git commit –m ’ …’ رو هم بزنیم تا یه توضیحی مینویسیم که این چی هست؟ m ، مخفف message است. این commit را برای این مینویسیم تا بعدا هرکی خواست log را ببیند گیج نشود که این فایل index.html چی هست .**

**پس git commit –m ‘ ‘ را زدیم و بعدش گفت باید بگی که کی هستی ما هم user.email و user.name رو بهش دادم تا در config –global اون رو بنویسه .**

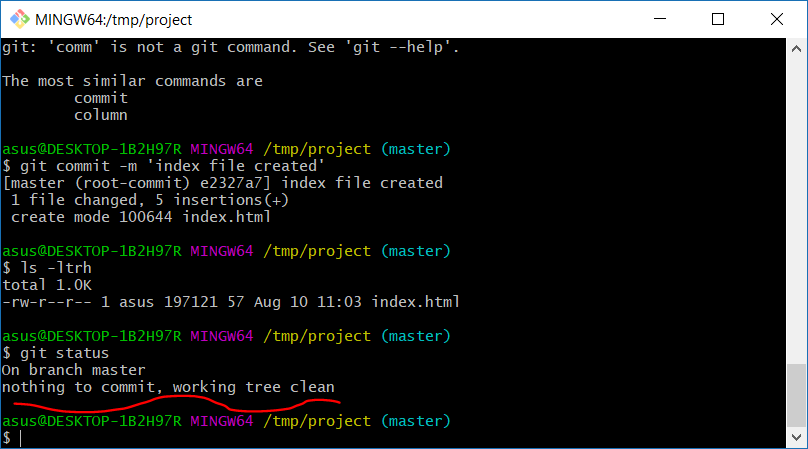


**حالا اگه بزنیم : ls –ltrh ، پیام زیر رو نشون میده :**



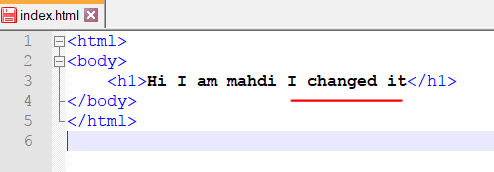
**حالا git status را میزنیم :**

**پیغام زیر رو نشون میده :**

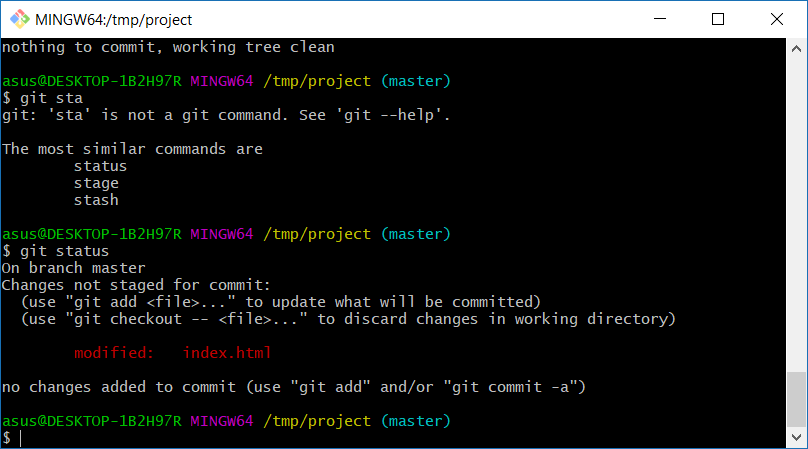


**و در واقع داره میگه هیچی عوض نشده .**

**حالا برای اینکه فایل رو عوض کنیم index.html رو با تکست ادیتور باز میکنیم و اون رو یه کوچولو دستکاری میکنیم :‌**



**حالا git status میزنیم ، پیغام زیر رو نشون میده :**



**که داره میگه index.html شما تغییر کرده و به اصطلاح modify شده. اینجا modify قرمز رنگ هست.**

**پس حالا دوباره index.html رو add میکنیم :**

**به این شکل :**

**git add index.html یا git add “\*.html” یا git add -A**

**نکته :**

**توجه : ستاره(\*) به معنای هر چیزی میباشد .**

**توجه : -A زدیم که یعنی همه فایل ها (All) .**

**پس ابتدا add میکنیم و بعد status میگیریم ، میبینیم که modify سبز رنگ شده است.و بعد هم حتما commit میکنیم .**

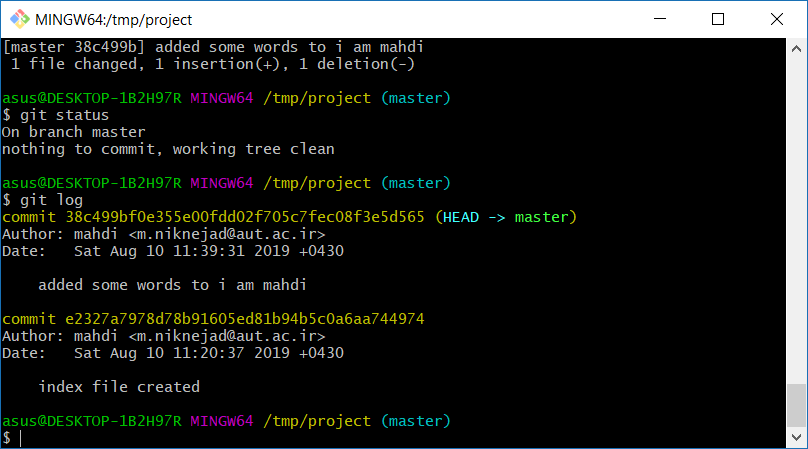
**Video 3 :**

**خب ، یه دستور داریم به اسم روبرو :**

**git log**

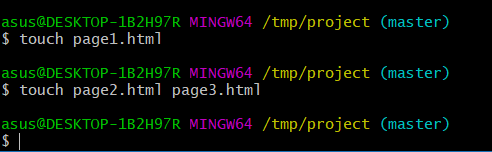
**این برای این است که commit هامون رو(یا log هامون رو ببینیم)**

**هر commit ، یک کد uique پیدا میکند .منظور‌:**



**در ضمن توجه کنید که commit ها را بر اساس آخری به اولی sort میکنه و نشون میده .**

**حالا میخواهیم مثلا پروژه را بیشترش کنیم و توسعه بدیم و برای این کار ، ۳ تا فایل .html دیگه میسازیم با همون touch**



در همه آنها مینویسیم مثلا :

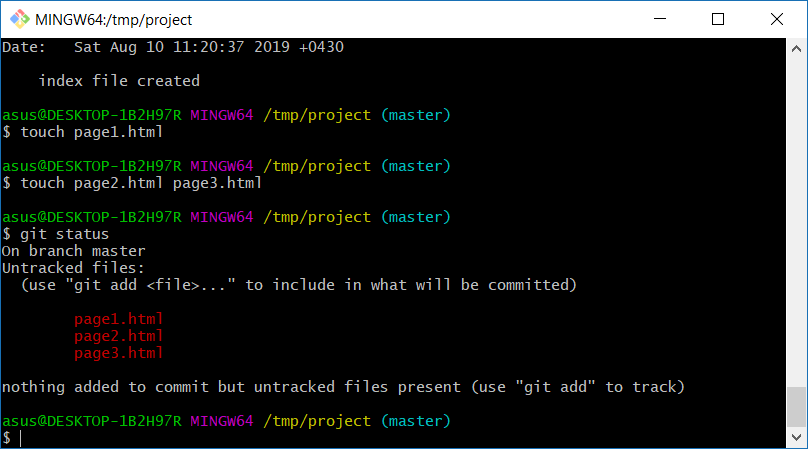
**You are on page 1**

**You are on page 2**

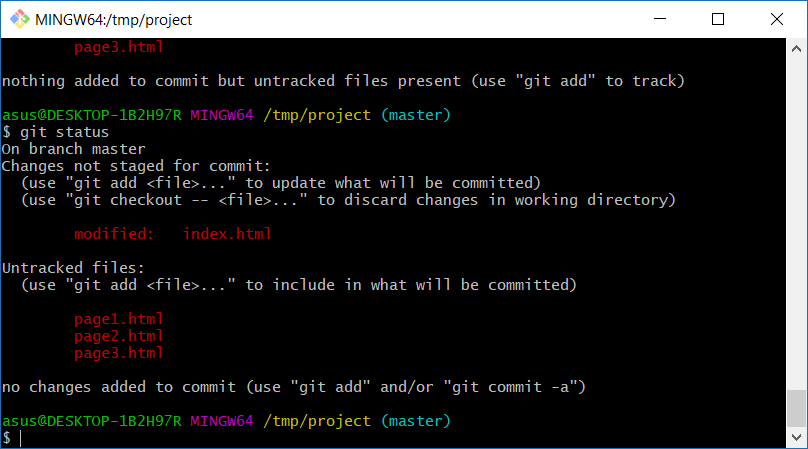
**You are on page 3**

**حالا میزنیم : git status**

**نتیجه زیر را به ما نشون میده :**



**حالا اگر همینجا index.html را هم عوض کنیم،این طور میشود ‌:**

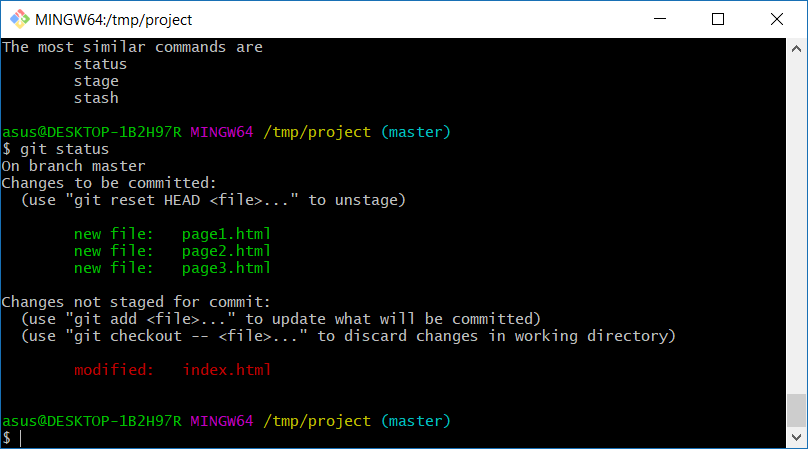


**حالا برای اینکه تمیز کار کنیم ، اول page ها را اول add میکنیم به این شکل :**

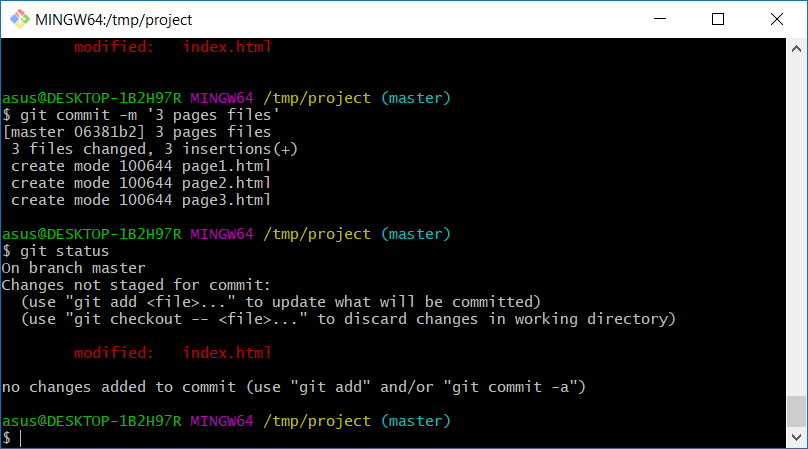
**نکته :**

**git add “page\*”**

**که این طوری تمام page ها add میشوند .اما index هنوز تغییری نکرده. منظور :**



**حالا اول این ۳ تا را commit کنیم :**

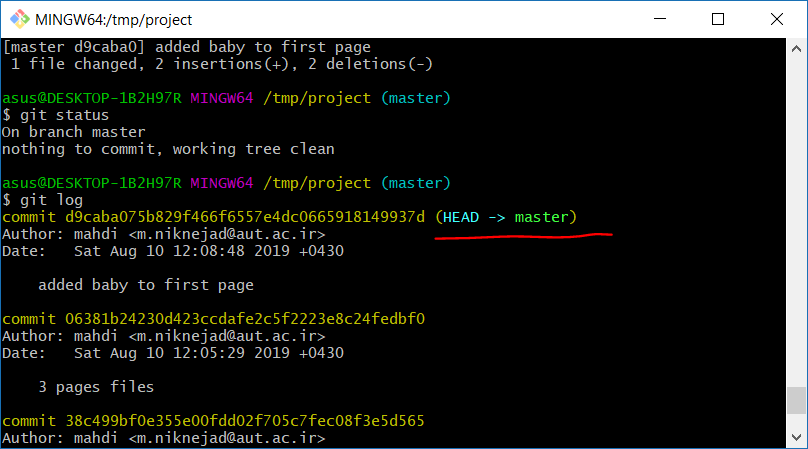


**و بعدش status میزنیم ، میبینیم که هنوز index مانده و باید اول add اش کنیم و ...**

**پس add اش میکنیم .**

**پس کلا اینطوریه : add -> commit**

**حالا status وlog رو زدیم (مدام باید این کار ها رو بکنیم).در log آن چیزی که نوشته جلویش Head یعنی که داریم الآن روی آن کار میکنیم . منظور :**



**توجه کنید که چقدر commit اضافه کردیم ، گیت به ما این امکان رو میده که به هر کدوم برگردیم .که جلوتر میبینیم .**

**Video 4 :‌**

**گفته بودیم که گیت به ما این اجازه رومیده که در commit هامون جلو عقب بریم و git log رو دیدیم ولی اگه بخواهیم واقعا جلو عقب بریم :‌بیاییم ببینیم گیت تغییرات ما رو چه طور نگاه میکنه و یادش نگه میداره .**

**خب برای مثال میزنیم توی page1 که یه چیزی رو عوض کنیم (مثلا مینویسیم : -editted ). حالا دوباره باید اون رو add کنیم . اما یه کار جالب میکنیم :**

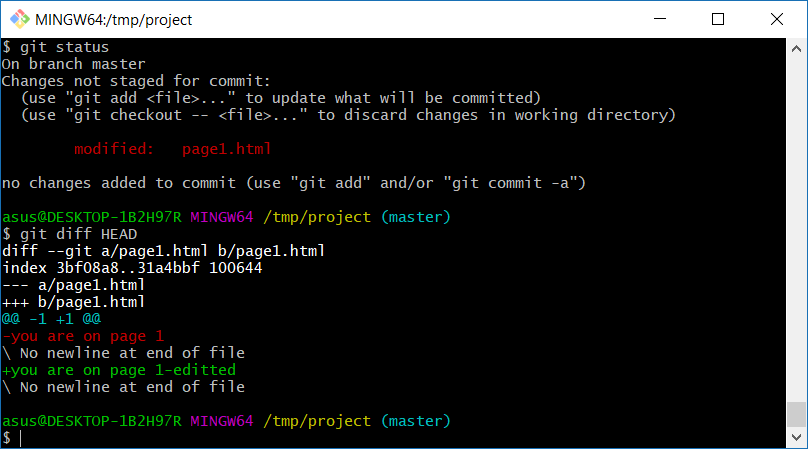
**میزنیم :**

**git diff HEAD**

**این HEAD همان HEAD ای است که در بالاترین (جدیدترین) commit در log با هم دیدیم .**

**diff می اید و تغییرات (تفاوت)فایل ها رو به ما نشون میده .به ما توضیحاتی رو ارایه میده . در ضمن این یک دستور لینوکسی است.**

**یک نکته : توجه کنید که HEAD را میشود ، بعدا جایش را عوض کرد. الان آخرین commit است .**



**a به فایل قدیم و b به فایل جدید اشاره میکند. –(منفی) نشان دهنده قدیم است و +(مثبت)نشان دهنده جدید است .**

**حالا برای مثال page 3 را هم عوض میکنیم و دوباره diff را میزنیم :**

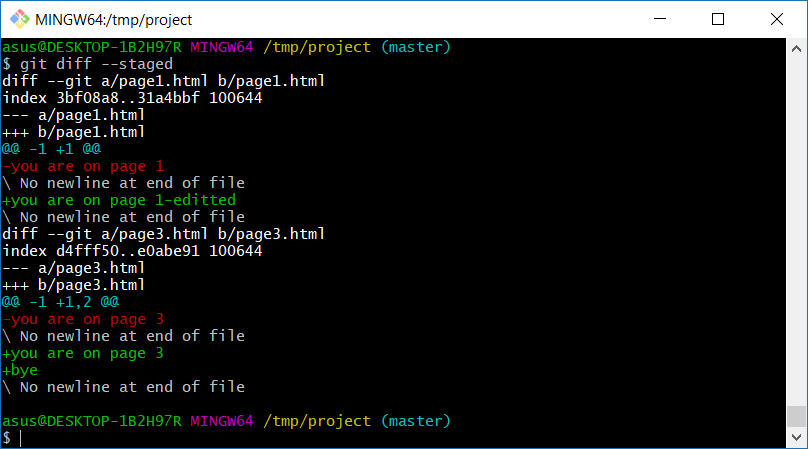
**مثلا در آخر page 3 ، bye را اضافه میکنیم.**



**حالا status میزنیم و بعد add –A را میزنیم خب حالا باید commit کنیم اما یه لحظه مشکوک میشیم و دستور زیر رو میزنیم:**

**git diff --staged**

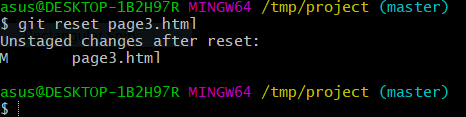
**نشون بده اونهایی که تو stage هستند ، چی شون عوض شده ؟ (توجه کنید که قبلش add –A رو زده باشید و اگرنه هیچ اتفاقی نمی افته)**



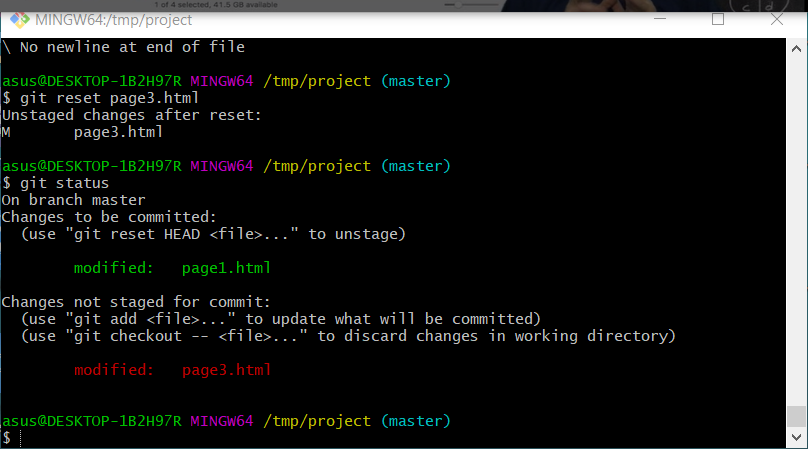
**توجه کنید که میتونیم یه چیزی رو از stage بیاوریم بیرون . مثلا الآن دقت میکنیم و میبینیم که تغییری که در page 3 دادیم خوب نبوده و باید عوض بشه . پس دستور مقابل رو میزنیم :‌**

**git reset page3.html**

**در پاسخ این رو به ما نشون میده :**



**خب حالا اگه که status بزنیم ، پایین رو به ما نشون میده :**



**که یعنی page1 هنوز commit نشده و page3 اصلا هنوز add نشده و یعنی از حالت stage دراومده .**

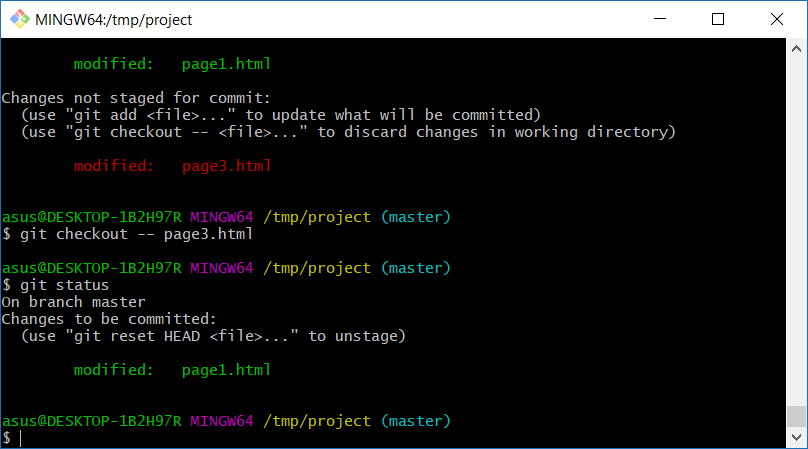
**و حالا اگه commit کنیم ، فقط page1 درست میشه.**

**یه چیز دیگه هم داریم که اگه فایلی رو خراب کرده باشیم و میخواستم کلا برگردونیمش به حالت قبلی ، دستور مقابل رو مینویسیم :**

**git checkout -- page3.html**

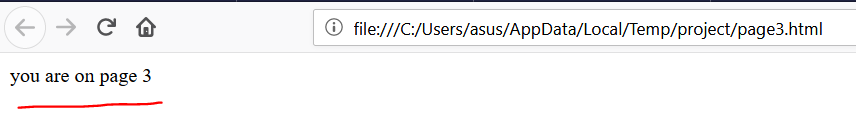
**توجه کنید که checkout به معنای چک کردن آخر خرید است یا مثلا وقتی ازهتل بیرون می آییم.**

**پس دستور را زدیم و بعدش status را هم زدیم . نتیجه :‌**



**داره میگه فقط یه چیزی برای commit داری. و در واقع page3 رو کلا برگردوند به همون حالت اول قبل از تغییر دادن . ؟؟؟**

**یعنی الان اگه page3 رو بریم ببینیم دیگه کلمه bye که به اون اضافه کردیم رو نداره .**

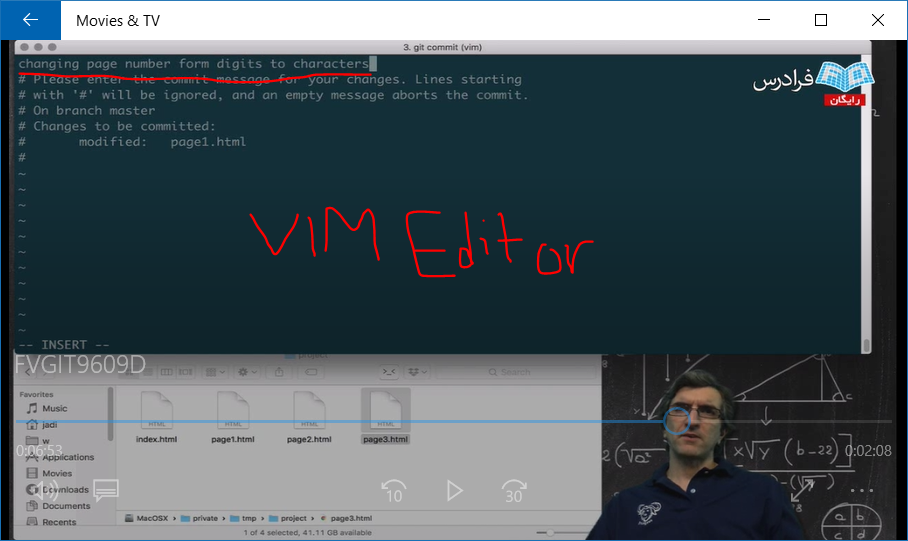


**حالا فقط یه commit خالی میزنیم .**

**git commit**

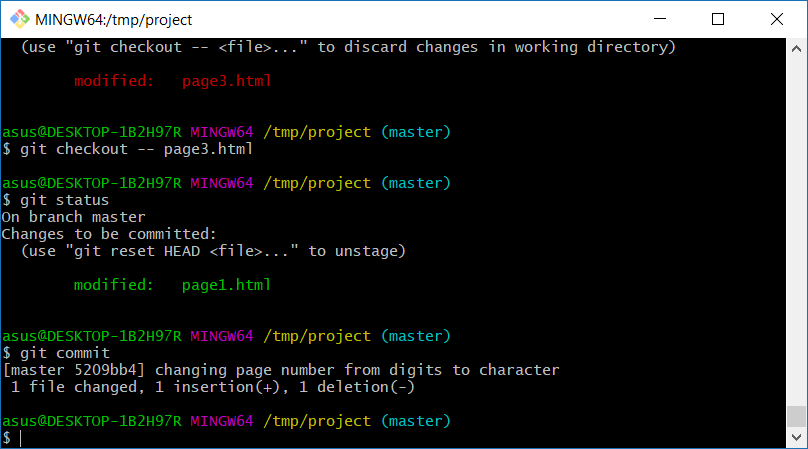
**این طوری وارد یک editor میشویم و پیغام را آنجا مینویسیم . مثلا مینویسیم :**

**changing page number from digits to character**

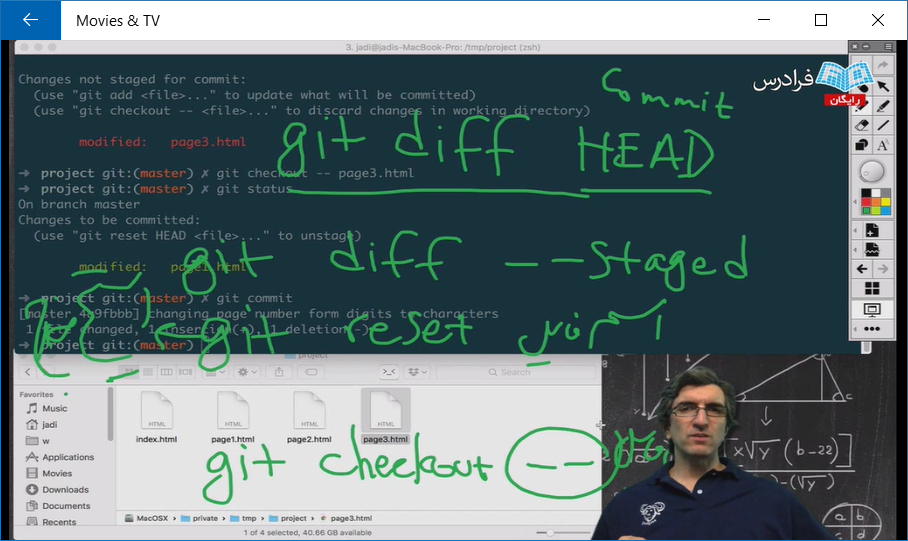


**و حالا برای اینکه بیاییم بیرون از صفحه ادیتور میزنیم : esc + :wq**

**و حالا این رو داریم :**



**خلاصه ‌:**



**در واقع checkout ، آن فایل رو از جایی که HEAD هست، می اورد بیرون و جایگزین وضع فعلی اش میکنه در واقع تغییراتی که انجام دادیم رو بی خیال میشه .**

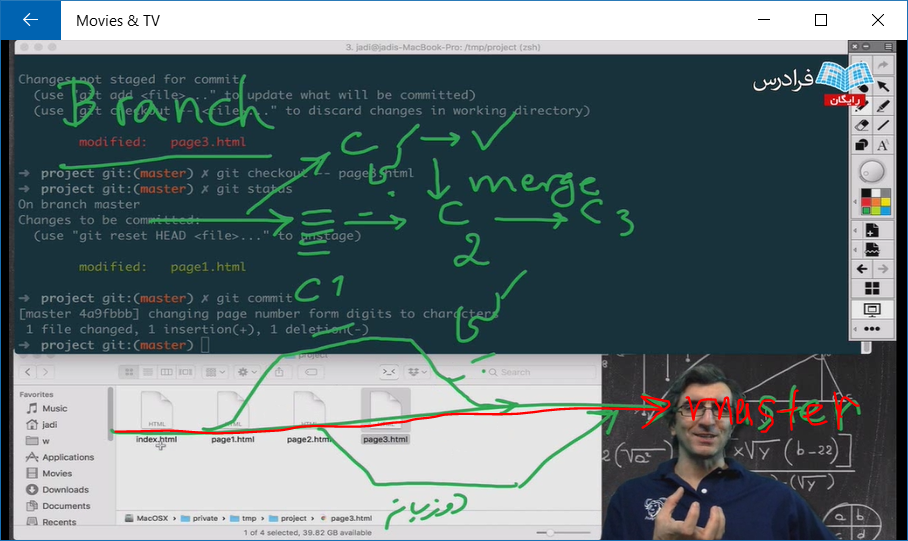
**Video 5 :‌**

**این جلسه میخواهیم راجع به Branch (شاخه) ها صحبت کنیم .**

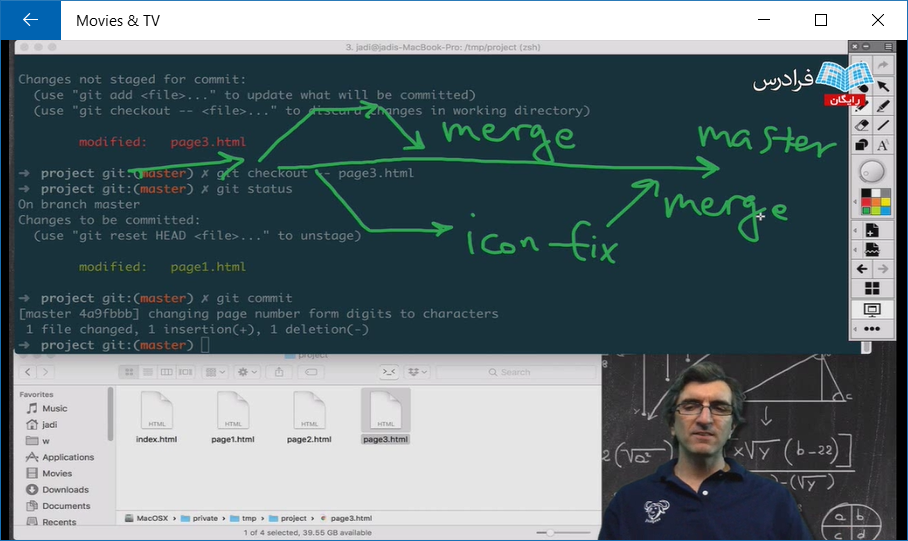
**مثلا داریم یک پروژه(بانک) مینویسیم و هی commit میکنیم ، میرسیم به commit 3 و رییس میگوید باید یه چیزی(کپچا) اضافه کنیم پس ما – که تا حالا داشتیم خطی میرفتیم جلو ، - حالا برمیگردیم و یک branch جدید اضافه میکنیم و پروژه رو ادامه میدهیم و وقتی کپچا درست شد با commit 3 ، میکس یا merge میکنیم .**

**خوبی این حالت اینه که میتونیم قسمت های مختلف پروژه رو جدا کنیم .**

**توجه :‌اسم شاخه اصلی master است .**



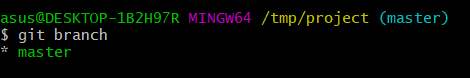
**اما اگر ۲ نفر هستیم که روی یک پروژه کار میکنیم ، هر کدام یک شاخه از branch master ، جدا میکینم و کارهای خودمان را میکنیم و بعد با master ، merge میکنیم .**



**حالا دستور مقابل رو میزنیم :‌**

**git branch**

**نتیجه :‌**



**ستاره \* ای که کنار master نوشته یعنی ما الان روی این branch هستیم .**

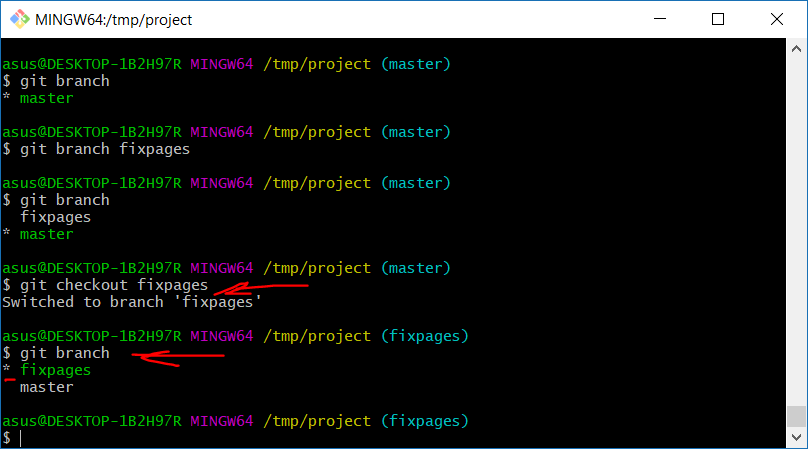
**برای اینکه یک branch جدید داشته باشیم ‌:دستور مقابل را میزنیم : برای مثال شاخه ای به اسم fixpages میخواهیم:**

**git branch fixpages**

**ما الان فقط این شاخه رو ساختیم و حالا برای اینکه برویم داخلش ،‌ از checkout (بکش بیرون) استفاده میکنیم . منظور‌:**

**git checkout fixpages**

**حالا اگه git branch را بزنیم میگه که در fixpages هستیم .منظور‌:**



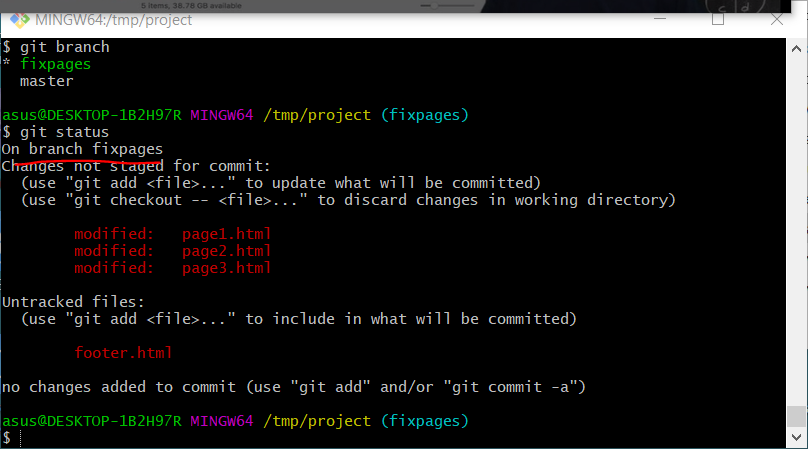
**حالا هر ۳ تا page رو درست میکنیم و تغییر میدهیم (در همه تگ های html و body رو اضافه میکنیم )**

**و در آخر هم یک footer.html هم با touch میسازیم . آنرا با ادیتور باز میکنیم و مینویسیم :**

**<hr>**

**Funsite for fun**

**حالا اگه status بزنیم ، نتیجه زیر بدست میاد :**



حالا add –A رو میزنیم :

**تا دقیقه ۶ از video E رو دیدم .**

حالا می آییم و همه را add میکنیم و فقط footer راreset میکنیم . حالا یکسری را commit میکنیم وبعد اون footer را کامیت میکنیم .

اگر یه فایل در یه branch بسازیم که تو اون یکی برنچ نیست وقتی جا به جا میشویم اون ناپدید میشه .

اما وقتی **git merge**  را میزنیم همه چی میاد این ور و حالا اون Footer.html را هم داریم این ور .

برای اینکه یک فایل را از گیت حذف کنیم از **git rm footer.html** استفاده میکنیم .

توجه : برای پاک کردن یک branch ، دستور مقابل را میزنیم :

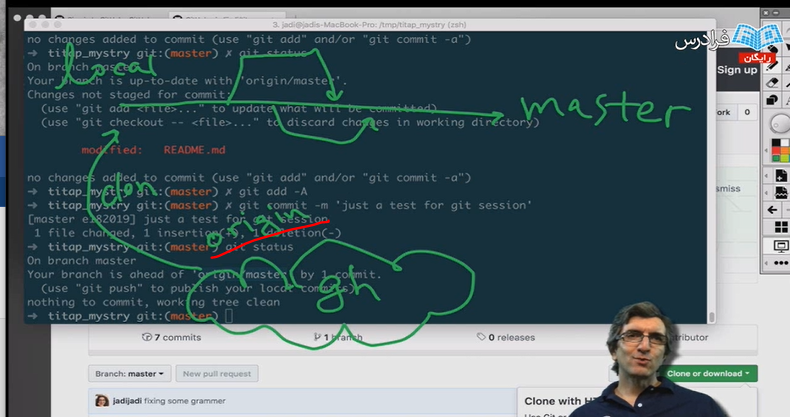
**git branch -d <branch name>**

توجه : وقتی یک برنچ را با دیگری مرج کردیم حالا میشود آنرا پاک کرد .

حالا : remote ها

میتونیم فایل خود را به یک جایی در شبکه مثلا push کنیم و یا از همان جا pull کنیم.

باید ابتدا clone کنیم . وقتی clone کنیم و عوض کنیم چیزی رو میگه که شما از origin/master 1 کامیت جلوتر هستی.



حالا برای اینکه push کنیم:

**git push origin master**

اما اگه فایل روی گیت هاب یا گیت لب عوض بشه باید pull کنیم .

**git pull origin master**

اگر –u بگذاریم در push یا pull اگه فقط git push یا فقط git pullبنویسیم خودش از oigin به master میبره .

به این منابع میگن : **repository**

برای push کردن به یه فایلی در یه شرکت باید اجازه دسترسی بگیریم یعنی :pull request بدیم احتمالا ؟؟

بررسی conflict ها :

میشه یه remote اضافه کنیم:

**git remote add origin** [**http://github.com/test-repo/**](http://github.com/test-repo/)

توجه کنید که میشه 2 نفر که دارند روی یه پروژه کار میکنند یک فایل رو همزمان عوض کنند .

در remote میشه اینکار رو کرد و این خطرناک است و باعث conflict میشود .

یعنی 2 نفر دارند در master یه فایل رو با هم عوض میکنند .

این جوری اگه push کنیم ، پیغام ما reject میخوره .

یعنی میگه اول pull کن و بعد push کن .

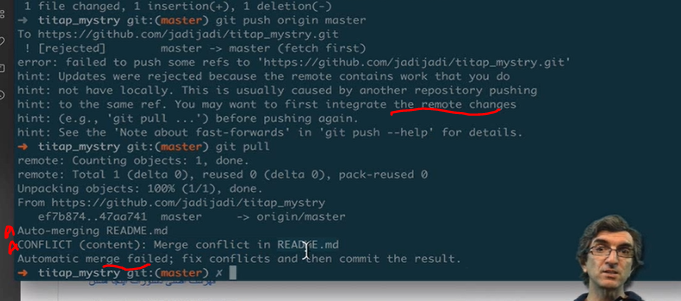
چون –u را زدیم از این به بعد میزینم فقط :

git push

git pull

اما وقتی فایل ها فرق کنه و ما هم pull کینم خود git میفهمه میگه من خودم auto merge میکنم .

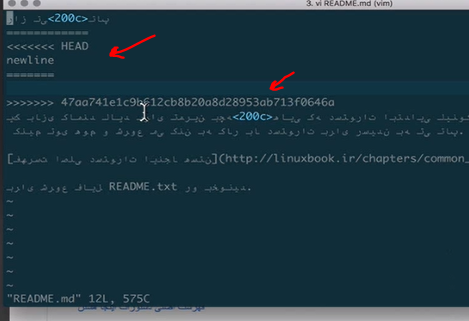
اما میگه بعدش که من به conflict خوردم .



میگه unmerged path

both modified

اگه همون فایل که اینجا READMEبود رو با vi باز کنیم :



میگه که شما >>>>>>آمدی و این کار را کردی و آن کاری که دوستت در سرور کرده اینه . اینم هشش است :

<<<<<<< Hash code

پس ما آمدیم و فایل را ادیت کردیم حالا باید commit اش کنیم .

حالا درست میکنه ولی میگه که branch تو از origin جلوتره .

البته چون هر 2 ی ما رفتیم و یک جای readme را عوض کردیم به مشکل خوردیم و اگرنه خود گیت میفهمه .

پس زود کدتون رو merge کنید push کنید ببینید که به مشکل میخورید یا نه

حالا:

درس جدید : تگ زدن

الان یه پروژه رو تا یه جا جلو رفتیم و خوبه دیگه میگیم تگ ورژن 1.0 را بزن روش که هر موقع خواستیمش بهمون بدهدش .

پس برای تگ زدن به ورژن ها :

**git tag**

مثلا میزنیم :

**git tag –a v1.0 –m “first tag “**

-a برای اینه که annotation اضافه کنیم .

توجه : برای اینکه ببینیم که متن یک commit چی بوده نویسنده اش کی بوده و اینها میشه تین رو زد :

البته قبلا از git log میریم ببینیم که شماره هش کامیت را برمیداریم و بعد :

**git show jnrji76eb** مثلا

و حالا برای اینکه آنجا ی کامیت را یه تگ بزنیم :

**git tag –a v0.6 –m “before “**

حالا 2تا تگ داریم

دقت کنید که لزومی به نوشتن ورژن نیست فقط v1.9 . مثلا هر چی میشه باشه ولی مرسومه که ورژن رو تگ میزنن.

در ضمن میشه نوشت :

**git tag –l “v\*”**

یعنی هر چی تگ با v شروع میشه رو به من نشون بده .

در ضمن میشه tag رو این جوری هم نگاه کرد :

**git show <name of tag version> مثلا : git show v1.0**

وقتی تگ میزنیم و بعدش push میکنیم هیچی رو اضافه نمیکنه اما برای اینکه push کنه باید بزنیم :

**git push origin v0.6**

یعنی روی origin ، ورژن 0.6 را پوش کن.

**git push origin --tags**

توجه : برای اینکه جلوی کاری رو بگیریم از ^C استفاده میکنیم .

حالا میشه تگ ها رو هم checkout کرد اما در آن تغییری رخ نمیدهد .

وقتی برنامه ها دارند release میدهند اون آخر میان و tag میزنن .

حالا برای اینکه یک تگ و یا کامیت را امضا کنیم :

git اجازه میده که شما امضای دیجیتال داشته باشد تا کسی نتونه بگه این مال منه و دزدی کنه

به pgp و gpg مراجعه کنید اینها رمزنگاری های مرسوم دنیا ی امروز هستند

**pretty good privacy**

رمز نگاری های کلیدی :

2 تا کلید جفت داریم که هر چیزی رو با کلید private درست میکنیم و بعد کلید public را به ملت میدهیم .

پس از gpg استفاده میکنیم برای اینکه کلید بسازیم :

**gpg --list-keys**

این طوری تمام کلید هایمان را لیست میکند و نشان میدهد

برای درست کردن کلید :

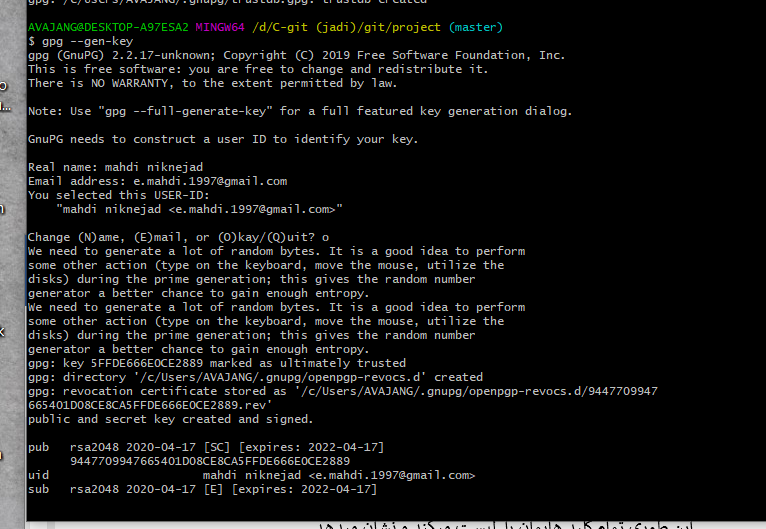
**gpg --gen-key**

یعنی generate کردن یک key

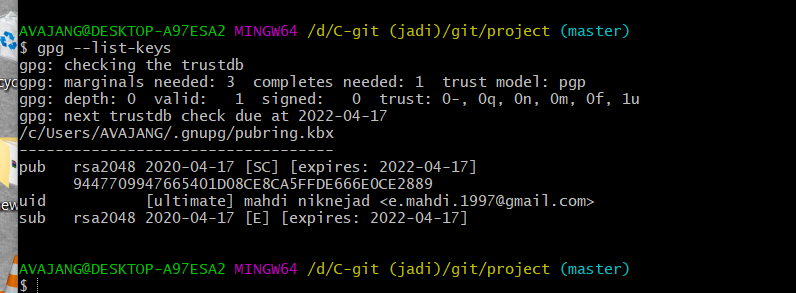
توجه کنید که اینجا من که این کد ها رو زدم و اسم و ایمیل را گرفت (بدون اینترنت)

و بعد خود دستگاهم از من یه رمز خواست و من هم دادم :

13760821mahdi



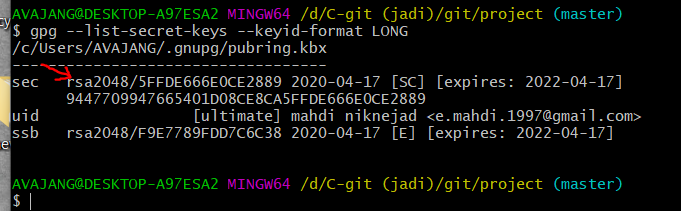
حالا یه کلید داریم .



میتوانیم بزنیم :

**gpg --list-secret-keys --keyid-format LONG**

با این شد :



در اونجا که علامت زدم : بعد از slash

5FFDE666E0CE2889

حالا میزنیم :

**$ git config --global user.signingKey 5FFDE666E0CE2889**

یعنی به گیت میگیم اگه خواستی از طرف من چیزی رو امضا کنی این طوری امضا کن با این رمز .

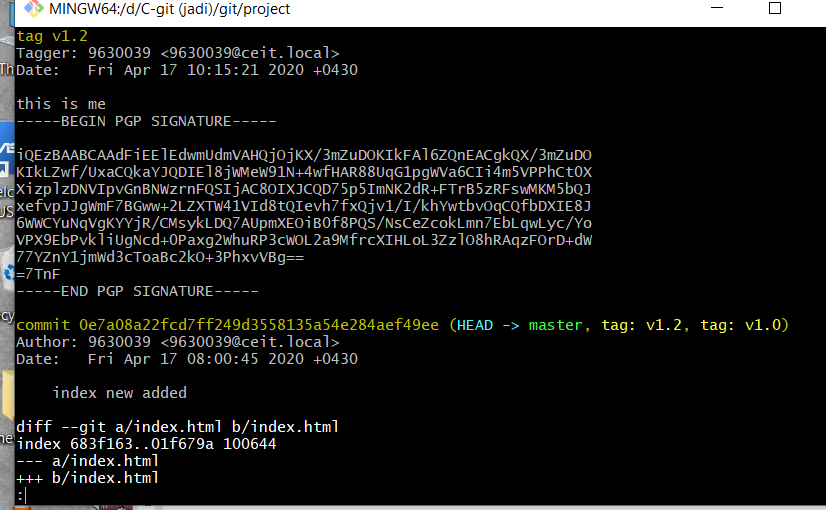
حالا میشه کارای قبل را کرد ولی این بار امضا هم کرد .

می آییم و یک تگ جدید میزنیم و به جای –a میزینم : -s که یعنی برو به signkey .

**$ git tag -s v1.0 -m "this is me "**

حالا اگه :

**git show v1.2**



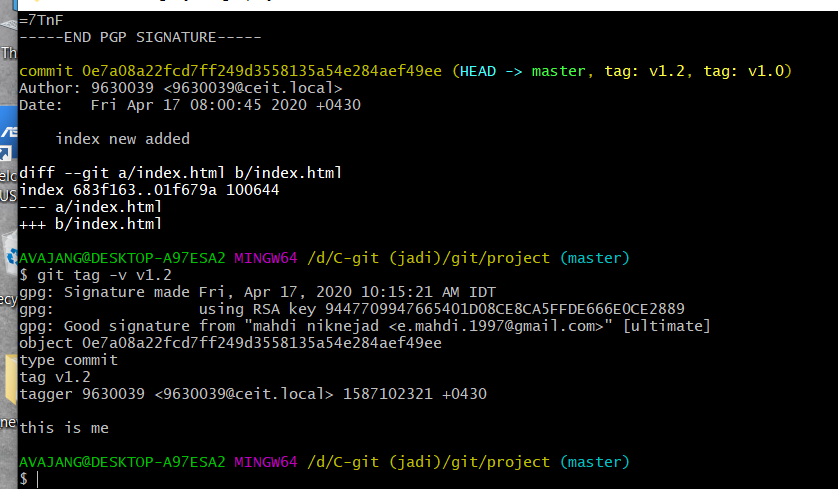
به اون پیام بزرگه اون بالا میگن : امضای دیجیتال

حالا برای اینکه تست کنیم که واقعا باز میکنه یا نه ؟

میزنیم :

**git tag –v v1.2**

برای verify کردن



اگر بخواهیم کامیت را هم امضا کنیم باید :

**git commit –S –m “new commit”**

با S بزرگ

درس جدید :

توجه : جادی یه بار زد less + folan chiz . less به چه معنایی هست ؟

نکته ریز : برای اینکه بدانی فلان چیز یعنی چی ؟

میزنی :

**git help blame** مثلا

blame کردن یعنی برو مقصر را پیدا کن

debug کردن با گیت در واقع این طوریه

یک مثال :

**git blame <name of file> -L8(this is line number)**

or

-L8 ,L10

از خط 8 تا 10 را نشان میدهد که چه اتفاقاتی افتاده

پس وقتی bug دیدید یا یه خط خوب و باحال دیدید از blame استفاده میکنیم .

یه دستور داریم به نام binary search commit با مخفف bisect

میخواهیم ببینیم که این باگ رد کدام کامیت ایجاد شده :

git bisect start

start , good , bad

خود گیت داره جلو و عقب میره . تا مشکل پیدا بشه .

درس آخر :

برای اینکه یک چیزی مثل گیتلب local داشته باشیم :

میشه برنامه گیتلب رو دانلود کنیم .

fork کردن یعنی یه پروژه رو بکن برای من .

میتونیم تو این تغییرات بدهیم . میتونیم حالا clone اش کنیم تغییر بدیم و پوش کنیم و اینها .

حالا می آییم و به صاحب اصلی پروژه میگیم که یعنی درخواست میدیم pull request میدیم که بیا کامیت مرا هم push کن رو مال خودت

میزنیم :

New Merge request

در گیتهاب pull requestداشتیم .

پایان ☺