به نام خدا

**آموزش Git**

منبع : Lynda

**فهرست مطالب**

[**فیلم 1 2**](#_Toc47760148)

[**فیلم 2 2**](#_Toc47760149)

[**فیلم 3 2**](#_Toc47760150)

[**مقدمه 2**](#_Toc47760151)

[**فیلم 4 2**](#_Toc47760152)

[**تاریخچه VCS 2**](#_Toc47760153)

[**فیلم 5 3**](#_Toc47760154)

[**درباره distributed version control 3**](#_Toc47760155)

[**فیلم 6 4**](#_Toc47760156)

[**فیلم 7 و 8 و 9 4**](#_Toc47760157)

[**فیلم 10 4**](#_Toc47760158)

[**فیلم 11 6**](#_Toc47760159)

[**فیلم 12 6**](#_Toc47760160)

[**فیلم 13 7**](#_Toc47760161)

[**initializing a repository 7**](#_Toc47760162)

[**فیلم 14 7**](#_Toc47760163)

[**فیلم 15 8**](#_Toc47760164)

[**فیلم 16 9**](#_Toc47760165)

[**در مورد نوشتن commit message 9**](#_Toc47760166)

[**فیلم 17 11**](#_Toc47760167)

[**فیلم 18 12**](#_Toc47760168)

[**فیلم 19 14**](#_Toc47760169)

[**فیلم 20 16**](#_Toc47760170)

[**فیلم 21 17**](#_Toc47760171)

[**فیلم 22 20**](#_Toc47760172)

[**فیلم 23 22**](#_Toc47760173)

[**]ویژه[ اضافه کردن یک پروژه آماده به ریپوزیتوری موجود در Github : 23**](#_Toc47760174)

فیلم 1

tick

فیلم 2

tick

فیلم 3

**مقدمه**

Git :

* keeps track of changes
* version 1,2,…
* Version Control System **VCS** - distributed version control
* Source code management SCM- use VCS for management
* open source
* created by Linus Torvalds
* Cross platform
* faster than other source controls before it

فیلم 4

**تاریخچه VCS**

* SCCS
  + source code control system
  + close source
  + **save snapshot of changes**
* RCS
  + cross-platform
  + open source
  + faster than SCCS
  + **نحوه کارکردش را نفهمیدم**
* CVS
  + concurrent version system
  + open source
  + **more than 1 user can work on the same file in the sam time**
* SVN
  + open source
  + faster than CVS
  + **take snapshot of directory not just file**
  + **نحوه کارکردش را دقیقا نفهمیدم**
* BitKeeper SCM
  + close source
  + در ابتدا برای لینوکس استفاده میشده اما بعد از چند وقت از رایگان بودن (بخشی اش رایگان بوده) در می آید و لینوکس هم دیگر از ان استفاده نمیکند و به سراغ گیت میرود .
  + **نحوه کارکردش را دقیقا نفهمیدم**
* Git
  + replaced instead of BitKeeper to manage linux kernel source code
  + ….

فیلم 5

درباره distributed version control

* هر کدام از vcs هایی که جلسه قبل صحبت شد ، یک central code repository model دارند و وظیفه ما است که آنرا آپدیت نگه داریم با توجه به submit های جدید و ... اما گیت این طور نیست
* each of team users maintain their own repositories instead of working from a cntral repository
* changes are stored as “change sets”
  + track changes not versions
  + دقیقا نفهمیدم
* no single master repository
* no need to communicate with a cntral server
  + no network needed
  + faster
  + کمی توضیحات دیگر داده شد که مهم نبود ولی واضح هم نبود و نفهمیدم

فیلم 6

**چه کسی از Git استفاده میکند ؟**

* anyone who tracks edits
  + review history log of changes
  + view differnces between versions
  + retrieve old versions
* anyone who wants to share changes with collaborators
* programmers

**چه کسی از گیت استفاده نمیکند ؟**

* anyone who wants to track non-text file
  + images,videos,musics و ....

فیلم 7 و 8 و 9

نصب گیت – tick

کارکردن با git bash – tick

با **which git** میشه فهمید که گیت کجا ذخیره شده است

همجنین با **git --version**  هم میشه فهمید که ورژنی که از گیت نصب کردیم چی هست

فیلم 10

**Git configuration**



commands :

* **System**
  + **git config –system**
* **User**
  + **git config –global**
* **Project**
  + **git config**

**تمرین ها :**

**ابتدا کامند های کانفیگ را میزنیم :**

**> git config --global user.name “mahdi niknejad”**

**> git config --global user.email “m.niknejad@aut.ac.ir”**

**> git config --list**

**user.name=mahdi niknejad**

**user.email=m.niknejad@aut.ac.ir**

**> git config user.name**

**mahdi niknejad**

**> git config user.email**

**m.niknejad@aut.ac.ir**

**و بعد میرویم تا فایل .gitconfig را پیدا کنیم :**

**> cd ~**

**> ls –la (you can see .gitconfig in the list)**

**> cat .gitconfig**

**[user]**

**name = mahdi niknejad**

**email = m.niknejad@aut.ac.ir**

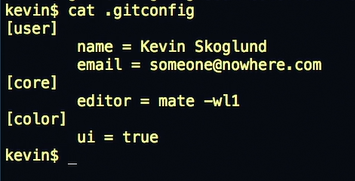
**signingKey = 5FFDE666E0CE2889**

**ست کردن ادیتور دیفالت :**

**> git config --global core.editor “nano” (or “vim” , “emacs” , “notepad” , “mate” , ... )**

**> git config --global color.ui true**

**> cat .gitconfig**



فیلم 11

**ظاهرا در لینوکس ، git completion نداریم ( گرچه در ویندوز به صورت دیفالت هست ! ) و باید ست کرد که اینجا آموزش داده شده است .**

فیلم 12

**در مورد git help :**

**> git help**

**> git help add**

**در اینجا میخواهیم ببینیم که add چه میکند . در لینوکس ، یک صفحه در همان command-line باز میشود ( صفحه Git manual) که توضیحاتی نوشته و در ویندوز ، در browser می آید .**

**برای جلو رفتن در صفحه Git manual ، از f استفاده میکنیم**

**برای عقب رفتن در صفحه Git manual ، از b استفاده میکنیم**

**برای خارج شدن هم از q استفاده میکنیم**

**این صفحه همان man هست . مثلا میزدیم :**

**man ls**

**یا**

**man git-add (= git help add)**

فیلم 13

**initializing a repository**

**کامندی که برای initialize کردن نیاز داریم تا بتوانیم از گیت برای یک فایل استفاده کنیم ، git init است .**

فیلم 14

**میتوانیم در آن پروژه که ساختیم بزنیم ls –la و میبینیم :**

**> ls -la**

**total 4**

**drwxr-xr-x 1 AVAJANG 197121 0 Aug 3 07:59 ./**

**drwxr-xr-x 1 AVAJANG 197121 0 Aug 3 07:59 ../**

**drwxr-xr-x 1 AVAJANG 197121 0 Aug 3 09:43 .git/**

**(توجه : اگر ls خالی بزنیم چیزی نشان نمیدهد چون hidden هستند)**

**همانطور که میبینید دایرکتوری .git ساخته شده که تمام track ها در آن ذخیره میشود**

**برای دیدن داخل دایرکتوری .git میزنیم :**

**> ls –la .git**

**total 11**

**drwxr-xr-x 1 AVAJANG 197121 0 Aug 3 09:43 ./**

**drwxr-xr-x 1 AVAJANG 197121 0 Aug 3 07:59 ../**

**-rw-r--r-- 1 AVAJANG 197121 130 Aug 3 09:43 config**

**-rw-r--r-- 1 AVAJANG 197121 73 Aug 3 07:59 description**

**-rw-r--r-- 1 AVAJANG 197121 23 Aug 3 07:59 HEAD**

**drwxr-xr-x 1 AVAJANG 197121 0 Aug 3 07:59 hooks/**

**drwxr-xr-x 1 AVAJANG 197121 0 Aug 3 07:59 info/**

**drwxr-xr-x 1 AVAJANG 197121 0 Aug 3 07:59 objects/**

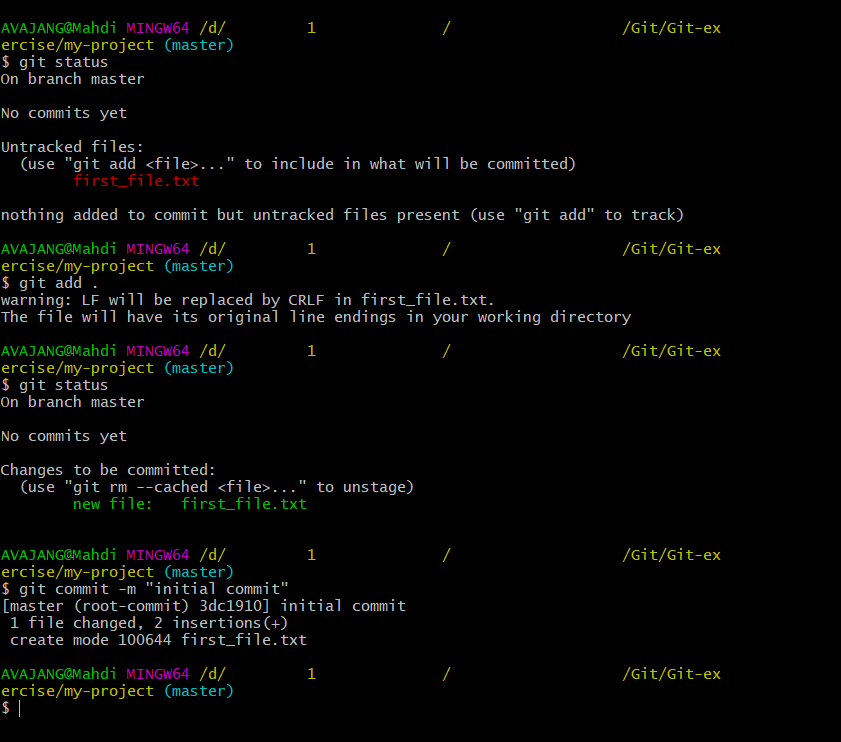
**drwxr-xr-x 1 AVAJANG 197121 0 Aug 3 07:59 refs/**

فیلم 15

یک فایل متنی first\_file.txt در فولدر پروژه میسازیم ( با همان touch ) و بعد میزنیم git status و میبینیم که untracked file داریم و حالا باید git add . بزنیم و بعدش git commit –m “message” بزنیم. حالا دیگه این فایل tracked شده است .

پس کار هایی که کردیم :

* make changes
* add changes
* commit changes



فیلم 16

در مورد نوشتن commit message

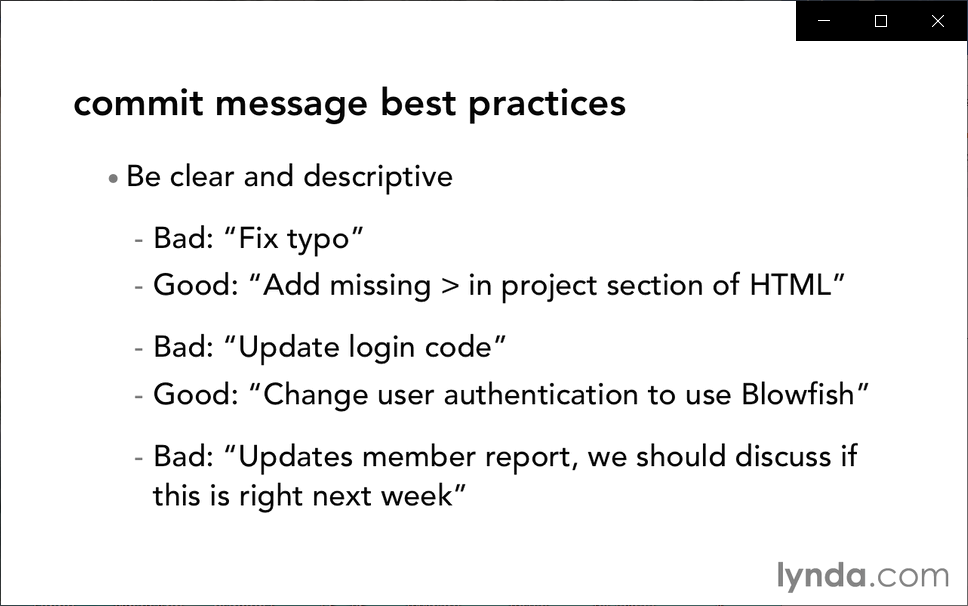
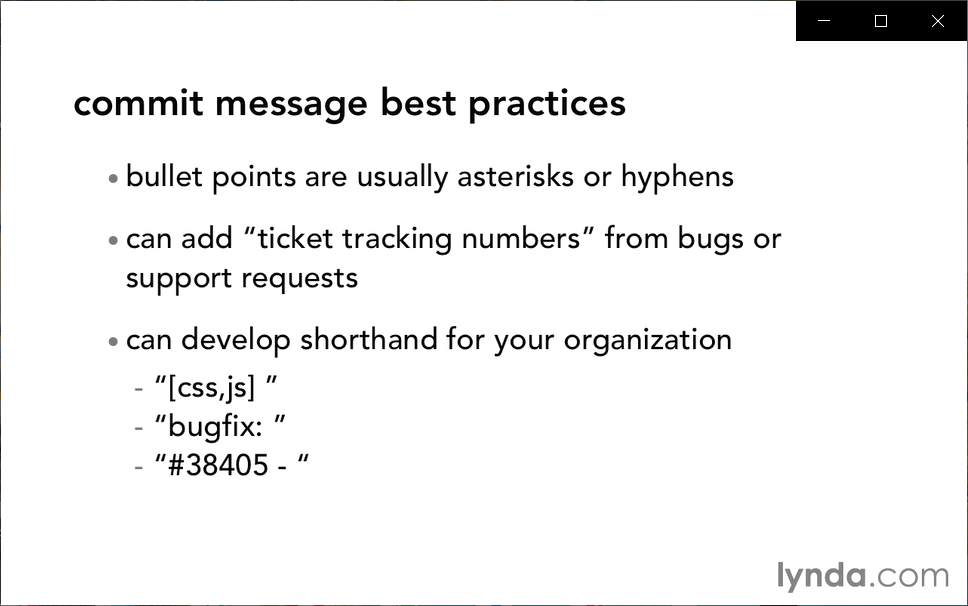
best practices :

* یک خطی و کوتاه ( کمتر از 50 کاراکتر ) ولی کامل
* نوشتن جملات به **زمان حال** ، نه گذشته !
  + مثلا :

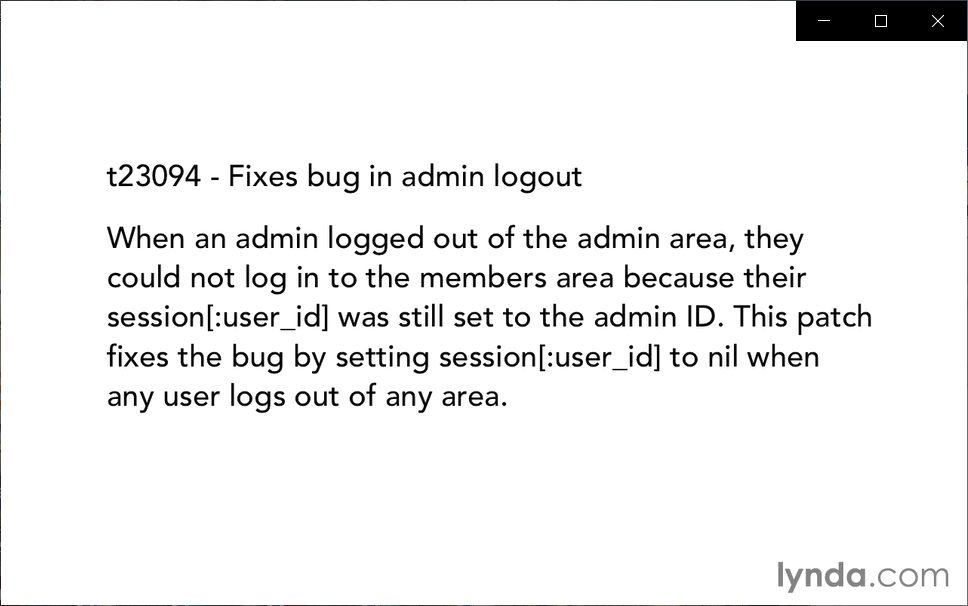
fix bug correct

fixes bug correct

~~fixed bug~~ incorrect



مثالی از یک commit خوب :



فیلم 17

برای دیدن commit ها(ی اخیر) باید از git log استفاده کنیم .

> git log

commit 3dc191018c913956fe2f419cff442ec128b96abd (HEAD -> master)

Author: mahdi niknejad <m.niknejad@aut.ac.ir>

Date: Mon Aug 3 10:11:20 2020 +0430

initial commit

برای اینکه فقط n تا(محدود تا) commit را ببینیم از git log –n 3 مثلا استفاده میکنیم.

برای اینکه از یک تاریخی به بعد ببینیم که این پروژه چه تغییراتی داشته و کامیت هایش چه بوده ، می آییم

و از git log --since=2019-08-12 مثلا استفاده میکنیم .

و همچنین برای اینکه ببینیم تا چه تاریخی ، چه تغییرات و کامیت هایی داشته ایم در پروژه می آییم

و از git log --until=2019-08-12 مثلا استفاده کنیم.

برای اینکه ببینیم کی کامیت ها را زده است و مثلا سرچ کنیم نویسنده کامیت را از git log --author=”Ali” مثلا استفاده میکنیم.

برای پیدا کردن یک کلمه یا حرف در متن کامیت از git log --grep=”com” مثلا استفاده میکنیم .

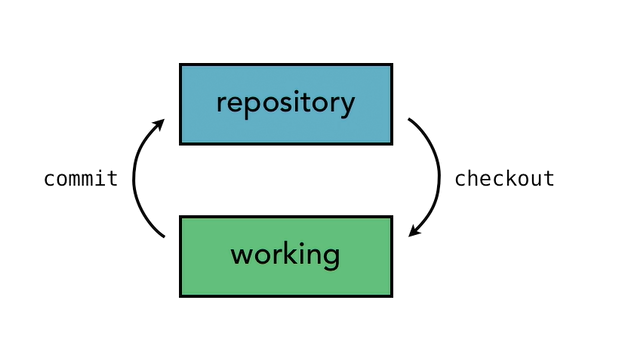
فیلم 18

**Git concepts & architectures**

**معماری two-tree :**

2 تا درخت داریم : repository , working

برای اینکه فایل ها را از repository به working ، ببریم و روی آنها کار کنیم از **checkout** استفاده میکنیم و برای اینکه تغییرات را به repository بیریم ، فایل ها را **commit** میکنیم.



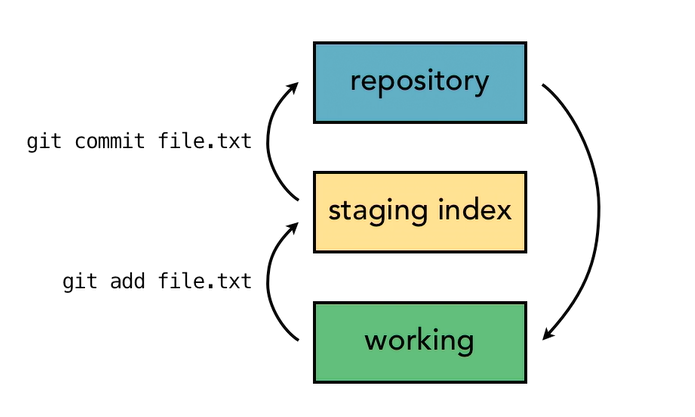
**git commit**

**git checkout**

**معماری three-tree :**

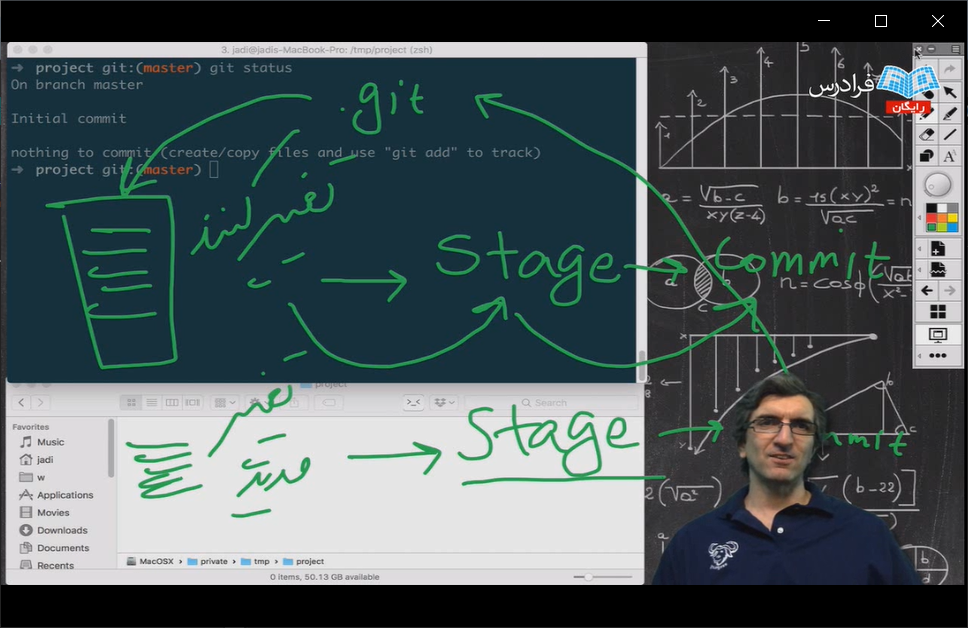
در اینجا به غیر از 2 تا بخش قبلی ، یک درخ دیگر برای staging هم داریم . یعنی در ابتدا ، تغییرات را add میکنیم و بعد آنها را commit میکنیم . علت وجود add این هست که مثلا میخواهیم 10 تا فایل مختلف را به repository اضافه کنیم ، اما در این مرحله میخواهیم فقط یکی از اینها را اضافه کنیم و بقیه هنوز کار دارند و باید رویشان کار شود و یعنی فقط یکی را میخواهیم کامیت کنیم وحتی ممکن هست که چند تا فایل با هم یک تغییر نیاز داشته باشند و باید روی آنها اعمال شود پس همه را ابتدا add میکنیم و بعد آنهایی که میخواهیم را commit میکنیم.

مرحله برگشت هم داریم که می آییم از repository به working می آییم



طبق توضیحات جادی :

یک دایرکتوری داریم که فایلهای درونش میتوانند تغییر کنند . وقتی تغییر کردند به وضعیت stage(لبه) میروند و آماده commit شدن میشوند . در واقع یک سری فایل استاندارد در دایرکتوری داشتیم حالا بعد از کامیت کردن دوباره فایل های جدید داریم که دوباره استاندارد هستند . حالا این فایل جدید کامیت شده به آرشیو .git میرود .

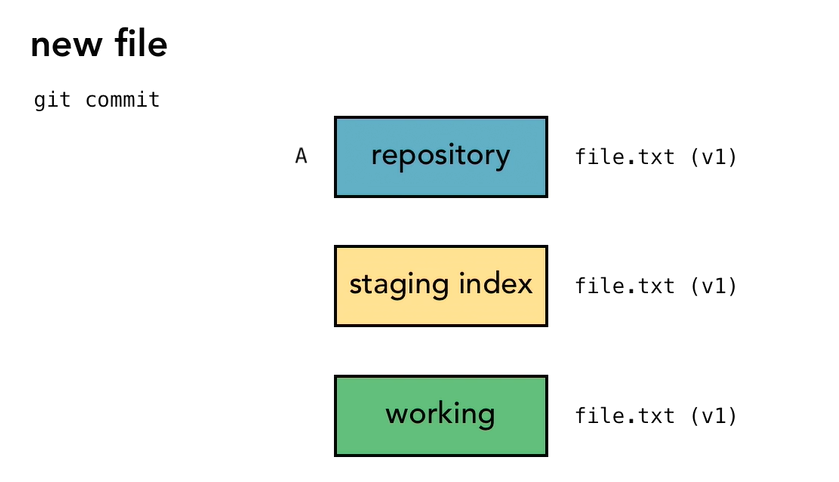


فیلم 19

**Git workflow**

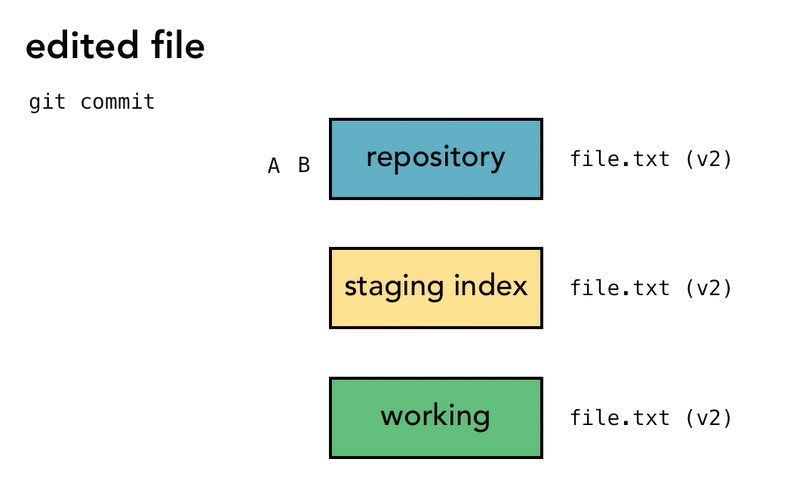
* ابتدا فرض کنید file.txt را در working داریم (مثلا با touch آنرا ساخته ایم ). حالا تغییراتی روی این فایل میدهیم یا نمیدهیم ولی آنرا add میکنیم با git add file.txt و حالا file.txt به stage رفته است . (البته برای اینکه همه فایل های untracked در یک دایرکتوری را اضافه کنیم میتوانیم از git add . استفاده کنیم.

و حالا باید تغییرات را commit کنیم با git commit . حالا دیگه فایل ما tracked شده است .



* حالا فرض کنید میخواهیم همان file.txt را edit کنیم .

یعنی ورژن 2 از فایلمان را داریم در working . و حالا باید تغییرات را add کنیم و بعد هم commit.



توجه : حالا با استفاده از git log میتوانیم تغییرات را ببینیم یعنی میتوانیمA B را ببینیم .

فیلم 20

ما در اینجا از اسنپ شات هایA , B , … استفاده کردیم ما در اصل اینها هر کدام مفهومی دارند . اینها **change sets** هستند .

Git می آید و برای هر change set ، یک **checksum** تولید میکند .

checksum دیتا را به یک عدد تبدیل میکند .

دیتا های یکسان ، checksum های یکسانی دارند .

تغییر دیتا ، checksum را هم تغییر میدهد

Git برای اینکه checksum را تولید کند ، از SHA-1 (hash algorithm) استفاده میکند

**checksum ، معمولا به این صورت هست : 40-character hexadecimal string یعنی از 0-9 و a-f تشکیل شده است . مثلا :**

**5c15e8bd540c113cd2d9eac6f64cacbc5ff6fe9c**

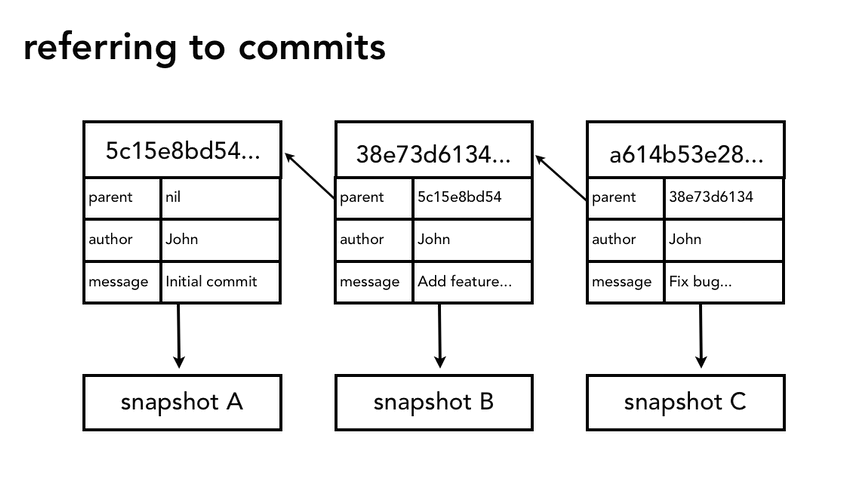
قبلا هم در git log ، این SHA را دیده بودیم در commit-id :



* حالا فرض کنید که 3 تا snapshot داریم : A,B,C ، میخواهیم ببینیم که این SHA به چه مواردی بستگی دارد ؟

commit-id و parent-commit و author of commit و commit-message

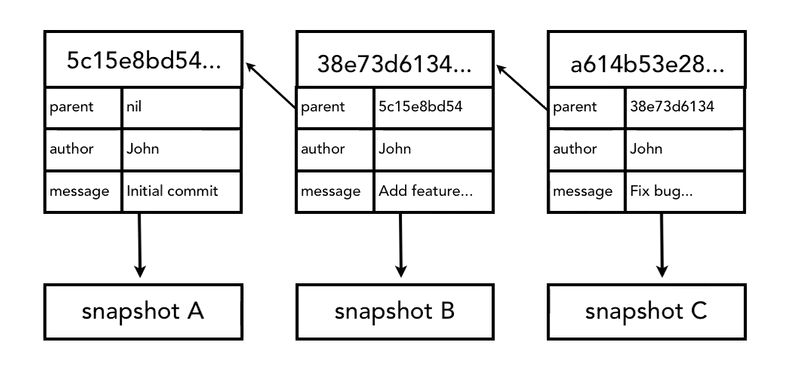
همانطور که میبینید ، snapshot B ، که کامیت بعدی A هست ؛ parent اش همان id عه A هست .



فیلم 21

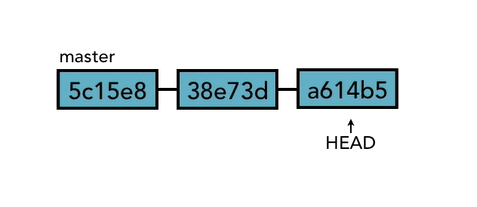
**HEAD pointer :**

* اشاره گری است که به یک commit خاص هست .
* pointer to **“tip” of current branch** in repository **--** اشاره گر به نوک برنچ جاری در ریپوزیتوری
* last state of repository – what was last checked out
* points to parent of next commit – where writing comits takes place



توجه : برنچ دیفالتی که داریم روی ان کار میکنیم ، master هست

فرض کنیم که 6 رقم اول SHA را به عنوان SHA در نظر بگیریم . داریم :



HEAD را دیدیم که به کجا اشاره میکند .

فرآیند commit کردنها و SHA ها : (sequence)

اول HEAD به 5c15e8 اشاره میکند و بعد کامیت جدید میکنیم و

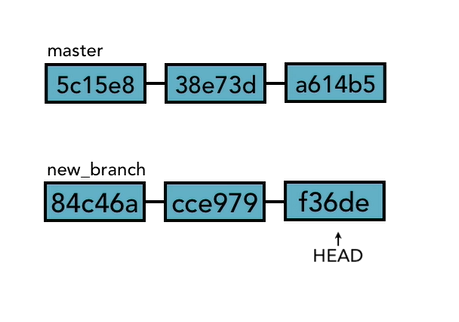
حالا HEAD به 38e73d اشاره میکند و بعد کامیت جدید میکنیم و

حالا HEAD به a614b5 ، اشاره میکند .

حالا میخواهیم یک برنچ جدید داشته باشیم . با استفاده از git branch <<name>>

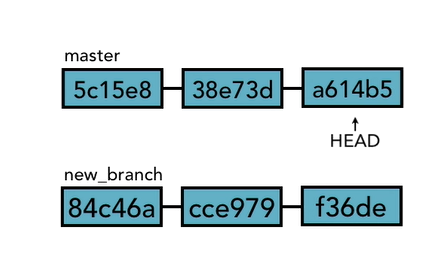
( و همچنین میتوانیم با git branch –d <<name>> ، آن برنچ جدید را حذف کنیم )

این طوری HEAD ، به اولین کامیت برنچ جدید اشاره میکند و قس علی هذا ..... تا میرسد به آخرین کامیت این برنچ جدید



و بعد با استفاده از git checkout ، میتوانیم بین branch ها switch کنیم و روی دیگری برویم .

وقتی دوباره برگردیم ، HEAD به آخرین کامیت master برمیگردد



خب دوباره به my-project میرویم و ls –la میزنیم . در آنجا یک فایل .git داشتیم .

به دایرکتوری .git میرویم و ls –la میزنیم .

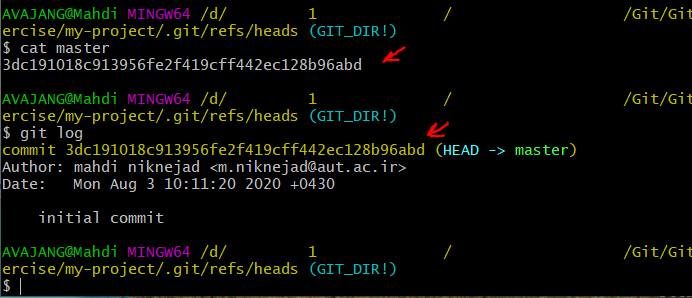
در داخل آن ، فایل HEAD را میبینیم . آنرا cat میکنیم تا ببینیم درونش چی هست

**ref: refs/heads/master**

میبینیم که به اخرین commit اشاره نمیکند بلکه به آخرین branch اشاره میکند و درون همان master ، tip پیاده سازی شده است که میبینیم :

پس به refs میرویم با cd و بعد ls –la میزنیم :

و بعد به heads میرویم و بعد هم master را cat میکنیم و میبینیم :



در اینجا همان commit-id ای که مال آخرین کامیت ما بوده را میبینم . وقتی git log را بزنیم ، میبینیم که همان هست دقیقا .

و اصلا میتوانیم بزنیم : git log HEAD

فیلم 22

**Adding files**

در ابتدا git status را میزنیم که یک گزارش وضعیت میدهد از تفاوت ها بین

working , staging , repository

وقتی فایلی اضافه نکرده باشیم یا تغییر نداده باشیم ، پیام میدهد که

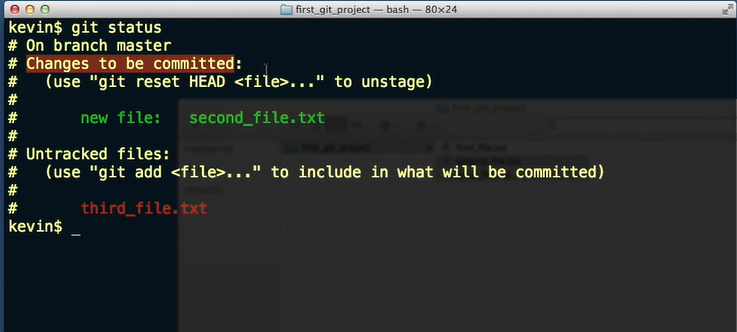
nothing to commit(working directory clean)

یعنی چیزی در حالت stage نداریم تا کامیت شود و یعنی همه چیز که در working هست ، در repository هم هست .

حالا چند فایل را اضافه میکنیم و حالا به ما پیام میدهد که untracked file داریم و باید ابتدا add شوند و به stage بروند .

پس میزنیم : git add . تا تمام فایل های این دایرکتوری add شوند (البته میشد دونه دونه هم add کرد ( که در اون صورت خواهیم داشت مثلا :

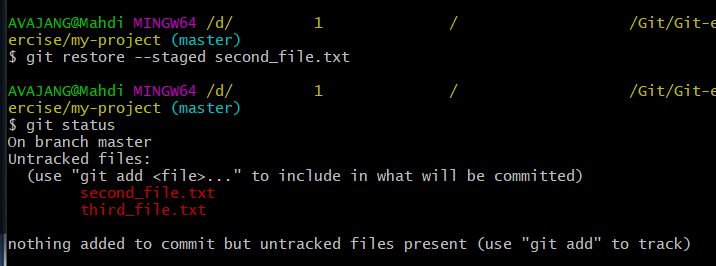
(یکر از 2 تا فایل را add کرده ایم فقط ) -->



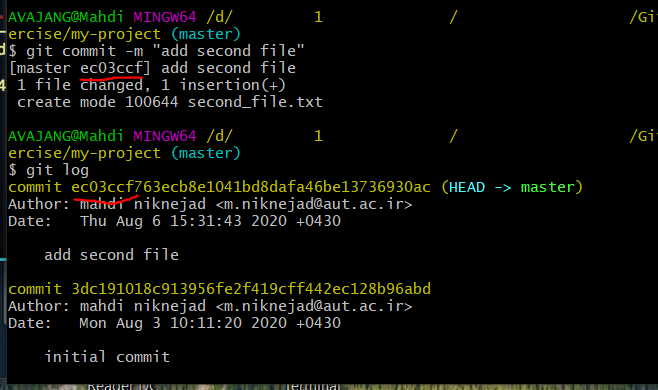
اینجا میگوید که second\_file رفته به stage و third\_file هنوز untracked هست و باید با add برود به stage .

در ضمن برای اینکه یک فایل را از stage خارج کنیم(unstage) باید از

git reset HEAD second\_file.txt و یا git restore --staged second\_file.txt استفاده کنیم .



خب حالا second\_file.txt را add میکنیم و بعد آنرا commit میکنیم و یک پیغام به ما میدهد :



و حالا بعد از اینکه دوباره git status بزنیم ، فقط third\_file.txt به عنوان untracked میماند .

حالا آنرا هم add و commit میکنیم .

فیلم 23

**Editing files**

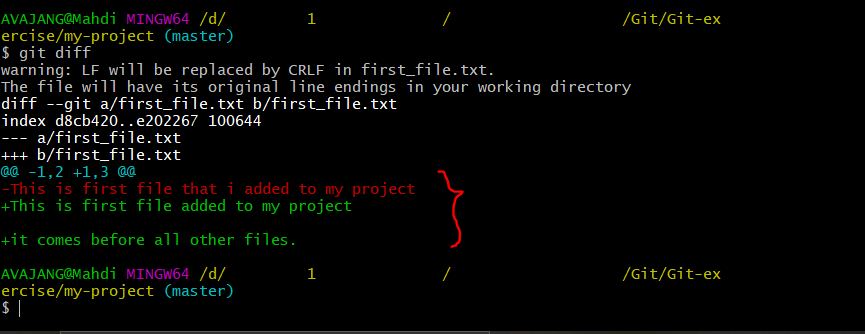
تغییری در first\_file.txt ایجاد میکنیم . git status را میزنیم. git add . را میزنیم اما قبل از git commit کردن ، second\_file وthird\_file را هم تغییر میدهیم . و git status میزنیم .میبینیم که first\_file در همان staging هست و حالا second را هم add میکنیم و آن هم سبز میشود مثل first و بعد این 2 تا را commit میکنیم . حالا فقط third مانده . و آنرا هم همین کارهارو باهاش میکنیم.

فیلم 24

**دیدن تغییرات با diff**

**با استفاده از git diff**

در first تغییراتی میدهیم و بعد با git diff تغییرات را میبینیم . در واقع با این کامند تغییرات و تفاوت ها differences را بین نسخه جدید و فدیم فایل میبینیم.



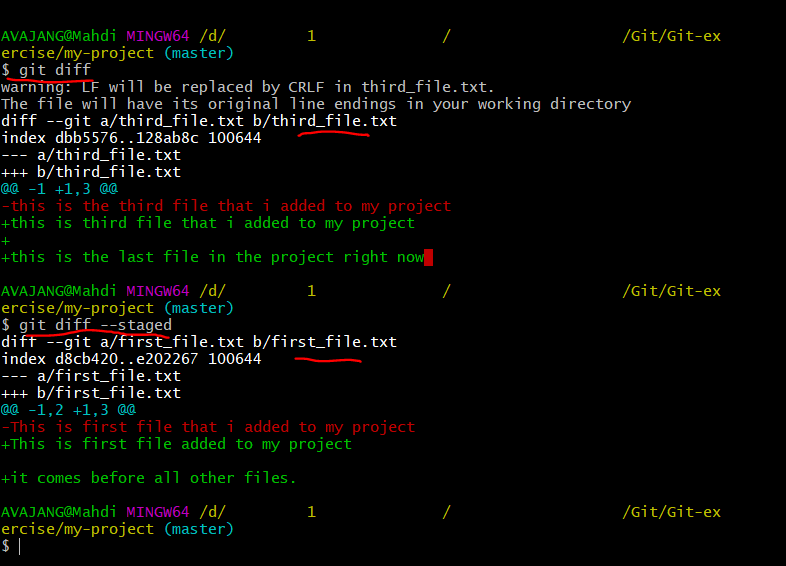
یک تغییر دیگر هم در third\_file میدهیم . و انرا هم diff میکنیم اون هم به ترتیب در لیست می آید و تغییراتش را نشان میدهد . برای اینکه فقط تغییرات یک فایل را ببینیم میتوانیم بزنیم مثلا : git diff first\_file.txt

فیلم 25

**دیدن staged changes**

حالا فقط first را add میکنیم و بعد git diff میزنیم. و به ما تغییرات third\_file را نشان میدهد

اما اگر git diff --staged بزنیم ، تغییرات first را نشان میدهد چرا که فقط first هست که add شده و به stage رفته است



توجه : git diff --cached هم معنی git diff --staged را میدهد .

حالا third را هم add میکنیم و بعد git diff میزنیم و میبینیم که هیچ اتفاقی نمی افتد در واقع تغییرات را تا قبل از رفتن به stage نشان میدهد و حالا برای دیدن تغییرات باید از git diff --staged استفاده کنیم و هر دو را هم میبینیم.

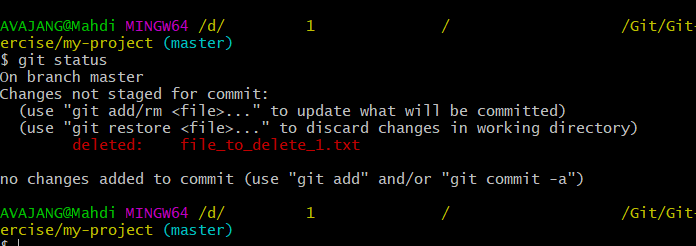
فیلم 26

Deleting files

راه 1

2 تا فایل جدید ایجاد میکنیم و بعد add و commit و در نهایت یکی را delete میکنیم و به سطل آشغال می اندازیم .

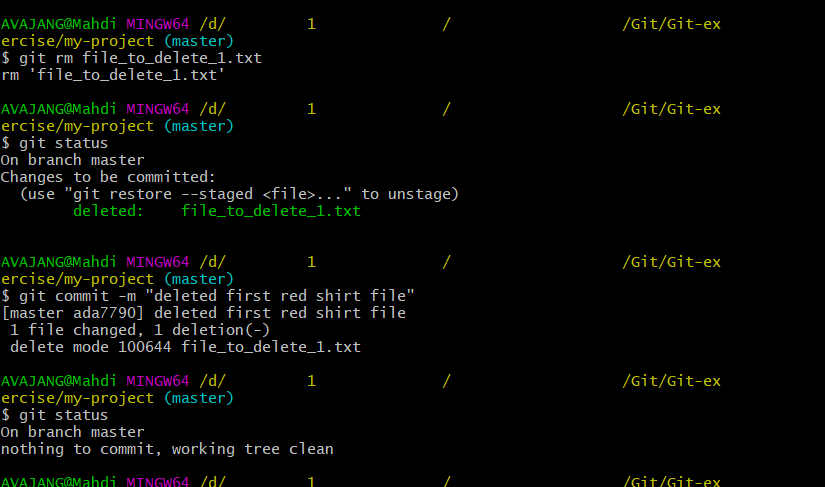
حالا وقتی git status میزنیم ، میبینیم :



حالا برای اینکه واقعا این فایل را delete کنیم ، از remove استفاده میکنیم یعنی :

git rm file\_to\_delete\_1.txt

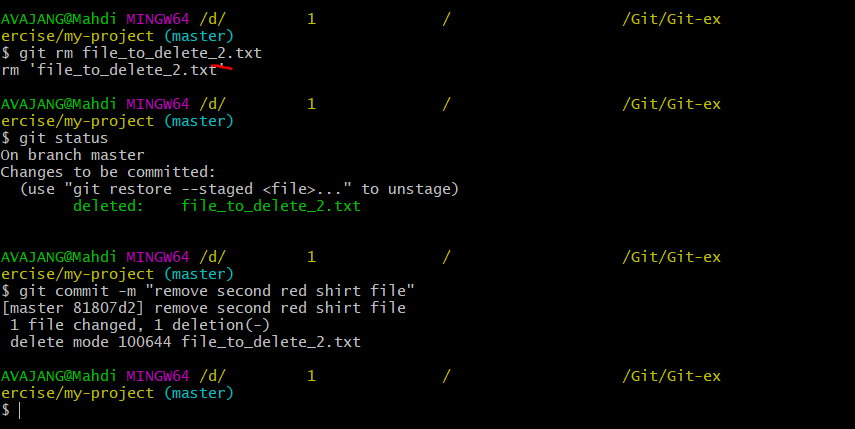
و بعد به ما میگوید که خب delete کردم و رفتم به stage و حالا اماده commit کردنه



راه 2

اما اگر بخواهیم از همان ابتدا از گیت حذفش کنیم نه با درگ کردن و انداختن در سطل آشغال چه کنیم ؟

میتوانیم مستقیم هم از git rm استفاده کنیم.



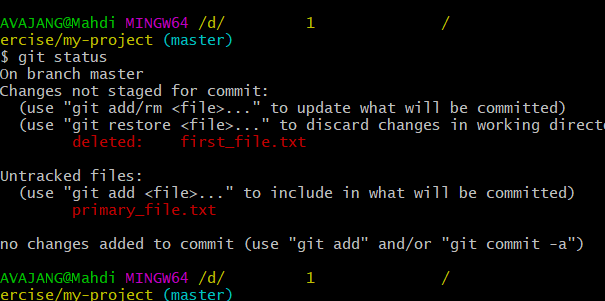
فیلم 27

Move and Rename Files

راه 1

Rename

فایل first\_file را Rename میکنیم . و git status میزینم ومیبینیم :



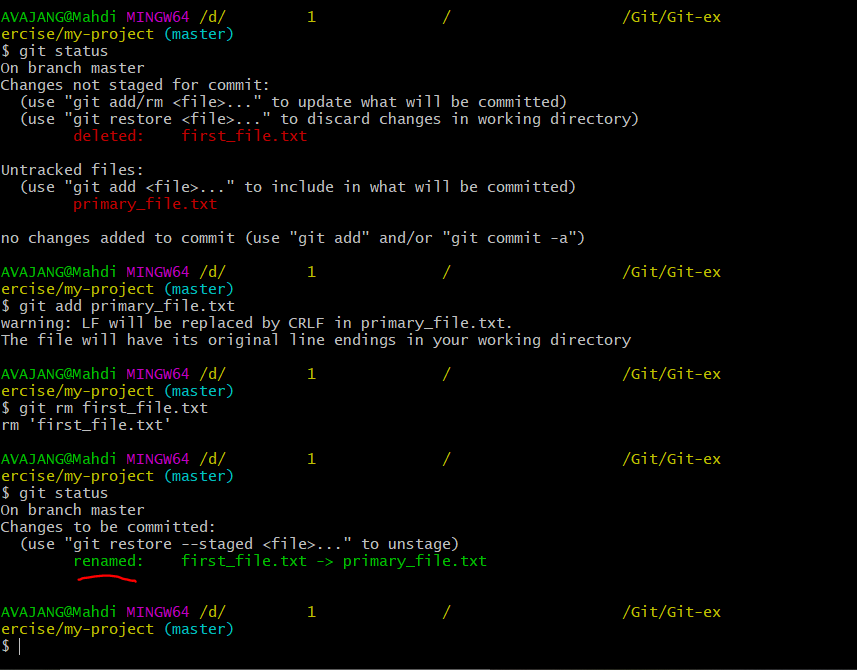
که یعنی یک فایل delete شده و یکی اضافه شده .

حالا باید کار های زیر را انجام بدهیم :

**git add <<new file>>**

**git rm <<old file>>**

و با git status داریم :



وقتی 50 درصد یک دیتایی یکسان باشند ، خود گیت میفهمد و rename تلقی میکند .

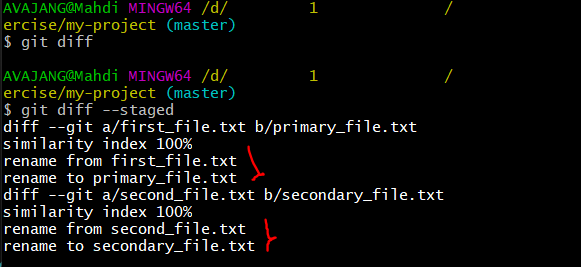
راه 2

**Move**

البته میتوانیم از دستورات لینوکسی هم استفاده کنیم مثل mv به عنوان move

git mv second\_file.txt secondary\_file.txt

و حالا git diff میزنیم :



]ویژه[ اضافه کردن یک پروژه آماده به ریپوزیتوری موجود در Github :

برای اینکه یک پروژه آماده را روی گیت ببریم :

1. Create a new repository on GitHub. You can also add a gitignore file, a readme and a licence if you want
2. Open Git Bash
3. Change the current working directory to your local project.
4. Initialize the local directory as a Git repository.  
   **git init**
5. Add the files in your new local repository. This stages them for the first commit.  
   **git add .**
6. Commit the files that you’ve staged in your local repository.  
   **git commit -m "initial commit"**
7. Copy the https url of your newly created repo
8. In the Command prompt, add the URL for the remote repository where your local repository will be pushed.  
     
   **git remote add origin remote <<repository URL>>**  
     
   **git remote -v**
9. Push the changes in your local repository to GitHub.  
     
   **git push -f origin master**

**توجه :**

**برای اینکه یک پروژه ای که فایل .Readme.md دارد در گیتهاب را استفاده کنیم باید در ابتدا آنرا**

**git pull <address> کنیم چون یک فایلی در آن ریپوزیتوری هست به نام readme که در working ما نیست پس باید اول با هم سینک شوند و بعد ادامه دهیم .**