فصل 1- (از 10:05)

گام 1- نصب git از git-scm.com

گام 2- نصب vs code از <https://code.visualstudio.com>

گام 3- نصب اکستنشن git history در vsCode

کارهای تکمیلی:

نصب اکستنشن material theme برای داشتن themeهای مورد علاقه در vsCode

نصب اکشتنشن material theme Icons در کنار فایلهایی که ایجاد می کنیم یک آیکن قرار می دهد.

اجرای اولین بار یک پروژه در vsCode:

برای اجرای اولین بار یک پروژه در vsCode، فولدر آن را به درون ادیتور vsCode دراگ می کنیم.

یا

File | Open Folder … | انتخاب فولدر | Select Folder

Repository یا REPO: پروژه ای که در آن از git استفاده شده را Repository گویند. (فولدری که در آن .git اضافه شده است.)

در Repository، git به طور کامل بر روی پروژه کنترل دارد.

COMMITS: می توان گفت هر commit یک نقطه save است.

قبل از هر commit دو مرحله وجود دارد:

* انتخاب فایلهایی که می خواهیم commit کنیم. (modified)
* STAGE کردن فایل هایی که می خواهیم commit کنیم. (staged)

Commit کردن یعنی فایلهایی را که در stage قرار دارند را در قالب یک تغییر ذخیره کنیم.

در کامپیوتر، دستورات را به دو طریق می توان انجام داد:

* رابط گرافیکی
* Command Line

مثلا برای ایجاد یک فولدر به دو طریق می توان عمل کرد:

* راست کلیک | new folder (رابط گرافیکی)
* در cmd:

>mkdir | Enter

git به صورت پیشفرض ابزاری command line است.

برای اینکه بفهمیم git نصب شده یا خیر؟ (در cmd):

>git

برای دانستن نسخه git نصب شده:

>git –version

برای ایجاد Repository: (تبدیل یک پروژه (فولدر) به Repository)

>git init

برای انجام commit:

>git commit

برای تنظیم نام:

>git config --global user.name "Mehdi Raeisi"

برای تنظیم email:

>git config --global user.email [mehdira3000@gmail.com](mailto:mehdira3000@gmail.com)

برای مشاهده نام تنظیم شده:

>git config --global user.name

برای مشاهده email تنظیم شده:

>git config --global user.email

هر commit حاوی یک امضاء (name و email شخص) است.

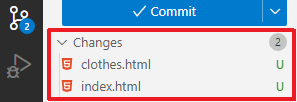
--global یعنی می خواهم این config در همه پروژه های من ثبت شود.

برای انجام کارهای git در vsCode:

 (Source Control) (سمت چپ صفحه) | Initialize Repository =>

اگر دستور git init اجرا نشده باشد؛ اجرا می شود.

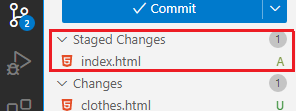
در  (Source Control)، فایلهایی که تغییر یافته اند در تب Changed نمایش داده می شوند.



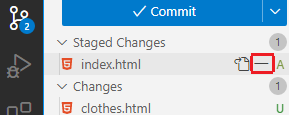
چون در مرحله modify هستیم و برای اینکه آنها را commit کنیم باید ابتدا آنها را به مرحله stage ببریم. برای این منظور روی دکمه  (stage changes) کلیک می کنیم.



بعد از کلیک روی دکمه  (stage changes)، فایل انتخابی به تب staged changes منتقل می شوند.

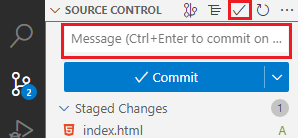


برای بازگرداندن فایل انتخابی به تب قبلی changes (حالت modified) روی دکمه  (Unstage Changes) کلیک می کنیم.



برای commit کردن فایل stage شده؛ دو کار باید انجام دهیم

* نوشتن یک عنوان یا پیغام در باکس message
* کلیک روی آیکن  (commit)



بعد از انجام commit و در زمانی که چیزی برای commit وجود ندارد؛ اصطلاحا گفته می شود branch ما empty یا clean است.

## Git History Extension:

Commit های ایجاد شده را کجا می توان دید؟

نصب اکستنشن git history باعث اضافه شدن آیکن (Git View History) در بالا و سمت راست صفحه شده که سوابق commit ها را نمایش می دهد.

در  (Source Control):

(Untracked): نشانگر فایلی ست که تاکنون commit نشده.



 (Index Added): با کلیک روی آیکن  (State Changes) مربوط به فایلی که تاکنون commit نشده؛ آیکن فایل به  تغییر می یابد. که این یعنی فایل به مرحله staged رسیده.

 (Modified): بعد از ایجاد تغییر مجدد بر روی فایل commit شده؛ آیکن آن به شکل  تغییر می یابد.

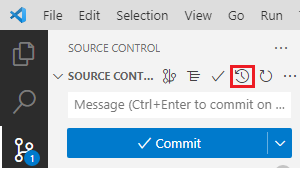
Branch: در git اصطلاح branch یعنی لیستی از commit ها.

در یک پروژه می توان چندین branch داشت.

با شروع یک پروژه، در ابتدا تنها یک branch اصلی داریم؛ که نام آن master است. اصطلاحا می گوییم که در head برنچ master هستیم. یعنی کدها مربوط به زمان آخرین commit است.

روش برگرداندن کدها به یکی از commit های پیشین:

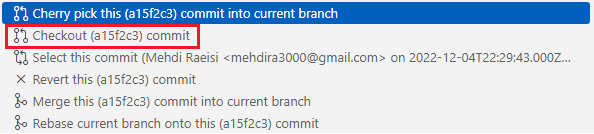
* کلیک روی آیکن (Git View History)



* در پنل  (Git History)، کلیک روی آیکن  (more) از commit مورد نظر.



* انتخاب گزینه Checkout … commit

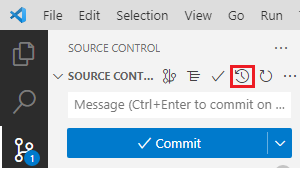


Merge کردن چند commit آخر:

ممکن است زمانهایی متوجه شویم که کد نوشته شده درست است ولی ترجیح می دهیم که مثلا سه commit آخر به یک commit تغییر پیدا کنند.

جهت انجام این کار:

* کلیک روی آیکن (Git View History)



* در پنل  (Git History)، کلیک روی آیکن  (Soft reset to this commit)



=> مشاهده پیغام "All affected files will be merged and kept in local workspace"   
با معنی “همه فایل های تحت تاثیر، merege شده و در local workspace نگهداری می شوند."

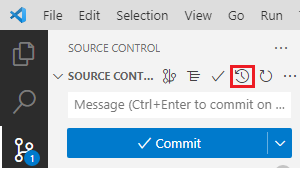
* کلیک روی دکمه OK

=> با انجام این کار، commit های آخر پاک می شوند ولی تغییرات همچنان باقیست.

و مثلا سه commit آخر، به یک commit کلی تر تبدیل می شود.

حذف چند commit آخر:

* کلیک روی آیکن (Git View History)



* در پنل  (Git History)، کلیک روی آیکن  (Hard reset to this commit)



* کلیک روی دکمه OK

=> حذف commit های بعد از commit انتخابی

# Branch:

با ایجاد یک repository، به صورت پیش فرض یک branch به نام master ایجاد می شود.

## روش ایجاد یک branch جدید:

master  (گوشه سمت چپ و پایین صفحه) | + Create new branch … | تایپ نام در branch name | Enter

## روش انتقال بین branch ها:

branch name (گوشه سمت چپ و پایین صفحه) | انتخاب branch مورد نظر (از لیست بالای صفحه)

## روش حذف یک branch:

Ctrl + Shift + p | تایپ عبارت: >git: Delete Branch | Enter | انتخاب branch

## merge کردن دو branch:

Ctrl + Shift + p | تایپ عبارت: >git: merge branch | Enter | انتخاب branch (جهت merge با branch جاری)

## انتقال commit از یک branch به branch دیگر:

در پنل  (Git History)، انتخاب یک commit | + branch (در commit انتخابی) | تایپ نام branch انتخابی | Enter

=> ایجاد یک branch جدید حاوی commit انتخابی و commitهای قبل تر آن.

## تداخل merge ها:

Accept current change: تغییرات ما اعمال می شود.

Accept incoming change: تعییرات branch دیگر اعمال می شود.

C یعنی وجود confilict

# gitignore:

برای اینکه git با فایلهای انتخابی ما کاری نداشته باشد و تغییرات آنها را دنبال نکند.

* ایجاد یک فایل جدید با نام: .gitignore
* تایپ نام file ها و folder هایی که نمی خواهیم git آنها را رد گیری کند.

# **github.com**

push کردن: انتقال تغییرات از local repository به repository موجود در github

pull کردن: گرفتن تغییرات از repository موجود در github

clone کردن: انتقال یک repository از github به سیستم.

fork کردن: همکاری در پروژه ای که permission آن را نداریم.

Fetch کردن: اگر تغییرات جدیدی در github به وجود آمده باشد؛ آن را به repository کامپیوتر خود ببریم و تغییرات خود را نیز push کنیم.

**Push کردن پروژه:** (انتقال پروژه از سیستم local به github.com)

(در سایت github.com) کلیک روی دکمه +  (بالا و سمت راست صفحه) | New Repository | نوشتن نام در باکس Repository name | انتخاب public یا private بودن پروژه (پروژه های private با ip ایران ساپورت نمی شود.) | Create repository   
| در کادر با نوشته "”**Quick setup — if you’ve done this kind of thing before:**

* https (پیشفرض است و آن را تغییر نمی دهیم)
* کپی کردن آدرس لینک نمایش داده شده: **1** (در اینجا) <https://github.com/mehdira3000/myRepositoryName.git>

| (درون vsCode) در terminal تایپ می کنیم: (روش اول)

> git push ‘**1** آدرس کپی شده در بالا’ ‘برنچی که می خواهیم منتقل کنیم’

که در اینجا می شود:

> git push https://github.com/mehdira3000/myRepositoryName.git master

| (اگر بار اول باشد که پروژه وارد github.com می کنیم) Sign in with your … | Authorize Git CredentialManager | وارد کردن پسورد  
| confirm => حال با refresh کردن browser در سایت github.com مشاهده می شود که repository از کامپیوتر به github منتقل شده.

#### **روش دوم نوشتن کد git push:**

* گام اول- ایجاد یک remote:

> git remote add origin <https://github.com/mehdira3000/myGitPracticing3.git>

* origin: نام مستعار لینک است. یعنی remote اصلی. (یک نام رایج است)
* http://github.com/... .git: آدرس کپی شده در سایت github.com **1**
* گام دوم- به جای نوشتن آدرس لینک کپی شده (**1**) (در روش قبلی)، می نویسیم origin:

> git push origin master

* master: نام برنچی است که می خواهیم push کنیم.

#### **روش سوم اجرای push:**

با ایجاد remote (در روش دوم)، از این به بعد (در پایین و سمت چپ vsCode)، با کلیک بر روی دکمه  ُSynchronize Changes)، این کار انجام می شود.

روش چهارم اجرای push:

Ctrl + Shift + p | تایپ می کنیم: >git: push | Enter

## **روش پاک کردن یک remote موجود:**

> git remote rm origin

**Pull کردن پروژه:** (انتقال repository از github.com به کامپیوتر ما)

(در سایت github.com) انتخاب Repository | Code | کپی کردن لینک https://github.com/.../myRepositoryName.git

| (درون vsCode) File | New Window (برای داشتن یک vsCode جدید) |  (Clone Git Repository …) | paste کردن آدرس لینک کپی شده | Clone from URL … | تعیین محلی که می خواهیم repository در آن قرار گیرد | Select Repository Location | OPEN

نکته- در اینجا بعد از انجام هر تغییری، برای push کردن تغییرات، با کلیک روی آیکن  ُSynchronize Changes)، در github، دیگر نیازی ساخت remote نیست.

# برای logout کردن از اکانت github:

(درون Control Panel) User Account | Manage your credentials | Window Credential | (در لیست Generic Credentials) انتخاب ‘git <https://github.com>’ | Remove => پاک شدن اطلاعات هویتی مربوط به github از سیستم ما

## **روش دادن permeation ایجاد تغییرات به سایر accountها در github:**

(درون github) ورود به Repository | Settings | Manage Accounts | وارد کردن پسورد | Invite a collaborator | جستجو و وارد کردن اکانت شخص مورد نظر | کلیک روی Add myAccountName to anotherAccountName (برای ارسال دعوتنامه) | Pending Invite => لینک را به شخص می دهیم تا با مراجعه به لینک، دعوتنامه را ببیند.

## **روش دیدن تغییرات Repository توسط سایر accountهای روی github:**

(در vsCode)  (ُSynchronize Changes) (پایین و سمت چپ صفحه) | OK => اعمال تغییرات اعلام شده در github روی پروژه در کامپیوتر ما

اینکه همه کاربران بتوانند روی master branch تغییرات ایجاد کرده و آن را commit کنند؛ جالب نیست و هر کاربری باید branch خود را داشته باشد.

(در github) صفحه را refresh می کنیم. | کلیک روی نام branch جاری (در اینجا master)  | Compare & Pull Branch | gallery added (نوشتن توضیحات) | Create pull request | (و برای گرفتن کد) Merge Pull Request | Confirm Merge

=> **Merge شدن دو branch**

# **GitHub Desktop**

دانلود از: desktop.github.com

برای اضافه کردن repository:

File | Add Local Repository | …

برای clone کردن:

File | Clone repository | …

برای ایجاد repository جدید:

File | New repository | …

برای login و logout کردن:

File | Option | …

اصطلاحات پر کاربرد:

push کردن: انتقال تغییرات از local repository به repository موجود در github

pull کردن: گرفتن تغییرات از repository موجود در github

clone کردن: انتقال یک repository از github به سیستم.

fork کردن: همکاری در پروژه ای که permission آن را نداریم.

Fetch کردن: اگر تغییرات جدیدی در github به وجود آمده باشد؛ آن را به repository کامپیوتر خود ببریم و تغییرات خود را نیز push کنیم.

تنظیمات git در سه سطح قابل انجام است:

* سطح system: تنظیمات تنها روی repository های سیستم ما اعمال می شود.
* سطح global: تنظیمات روی repositoryهای current user اعمال خواهد شد.
* سطح local: تنظیمات تنها روی repository جاری اعمال خواهد شد.

برای فهمیدن اینکه git نصب شده یا خیر:

>git

برای فهمیدن ورژن git نصب شده:

>git –version

برای تنظیم نام:

>git config --global user.name "Mehdi Raeisi"

برای تنظیم email:

>git config --global user.email [mehdira3000@gmail.com](mailto:mehdira3000@gmail.com)

برای مشاهده نام تنظیم شده:

>git config --global user.name

برای مشاهده email تنظیم شده:

>git config --global user.email

برای مشاهده تمامی تنظیمات انجام شده:

>git config --list

برای ایجاد یک repository خالی: (فولدر جاری به عنوان repository در نظر گرفته می شود.)

>git init

برای ایجاد یک repository خالی با نام انتخابی: (فولدری با نام انتخابی به عنوان repository در نظر گرفته می شود.)

>git init myRepositoryName

برای انتقال repository از github به سیستم:

>git clone <https://github.com/mehdira/git-crash-course.git>

برای انتقال repository از github به سیستم، با تعیین نامی جدید:

>git clone <https://github.com/mehdira/git-crash-course.git> newRepositoryName

برای add کردن فایل جدید یا تغییر کرده به stage:

>git add myFileName.format

برای add کردن همه فایل هایی که در پروژه تغییر کرده اند به stage: (. یا \*)

>git add .

OR

>git add \*

OR

>git add -A

برای حذف فایل انتخابی در stage:

>git rm myFileName.format

برای برگرداندن فایل تغییر یافته از stage به حالت قبل از git add یا git rm:

>git reset HEAD myFileName.format

برای تایید تغییرات موجود در stage:

>git commit -m “my about text”

برای تغییر و ویرایش پیام آخرین commit:

>git ommit –amend -m “new my about text”

برای نمایش فایلهای در دست تغییر و فایلهای موجود در ناحیه stage که هنوز commit نشده اند:

>git status

برای نمایش git status به صورت خلاصه:

>git status -s

برای ایجاد یک branch جدید:

>git branch -M myBranchName

برای نمایش branch های موجود در یک repository:

>git branch

برای حذف یک branch موجود:

>git branch -d myBranchName

>git branch -D myBranchName

برای جابجایی بین branch ها:

>git checkout myBranchName

برای ایجاد یک branch جدید و انتقال به آن:

>git checkout -b myNewBranchName

برای برگرداندن به وضعیت اولیه در فایلهایی که به اشتباه merge شده اند:

>git checkout -m fileName.format

برای مشاهده لیستی از remote های مربوط به repository:

>git remote

برای حذف یک remote از repository:

>git remote rm remoteName

برای دسترسی به repository موجود در github از remote استفاده می شود:

>git remote add remoteName url

مثال-

> git remote add origin <https://github.com/mehdira3000/myGitPracticing3.git>

* origin: نام مستعار لینک است. یعنی remote اصلی. (یک نام رایج است)
* http://github.com/... .git: آدرس کپی شده در سایت github.com **1**

برای انتقال پروژه (repository) از کامپیوتر به github (با استفاده از دستور remote):

>git push -u remoteName myBranchName

مثال-

>git push -u origin master

برای انتقال پروژه (repository) از کامپیوتر به github (بدون استفاده از دستور remote):

> git push https://github.com/mehdira3000/myRepositoryName.git master

باید قبل از استفاده از دستور git push، منبع و قسمت بالایی git تنظیم شود:

>git push –set-upstream shortName branchName

مثال-

>git push –set-upstream origin featureBranch

برای گرفتن تغییرات از repository موجود در github:

>git pull remoteName

مثال-

>git pull origin

برای دانلود تغییرات سایر اعضای تیم از github:

>git fetch

>git fetch origin main

این دستور اطلاعات مربوط به همه commitها، رفرنس ها و ... را دانلود می کند. بنابراین قبل از اعمال این تغییرات در repository کامپیوتر خود، می توان آنها را مرور کرد.

برای ذخیره کردن موقت فایلهای ویرایش شده commit نشده:

>git stash

برای دیدن لیست فایلهای stash(دخیره موقت) شده:

>git stash list

برای اضافه کردن یک stash به branch:

>git stash apply

با دستورات git stash apply و git stash pop، می توان تغییرات ذخیره شده را دوباره اعمال کرد.

هر دو فرمان تغییرات ذخیره شده در آخرین stash را دوباره اعمال می کنند.

یک stash apply تغییرات را مجددا اعمال می کند درحالیکه stash pop تغییرات را از stash حذف کرده و مجددا آنها را روی نسخه کاری اعمال می کند.

برای مشاهده همه commit های قبلی branch جاری (با شروع از آخرین commit):

>git log

برای مشاهده همه commit های قبلی branch مورد نظر (با شروع از آخرین commit):

>git log branchName

برای مشاهده همه commit های قبلی از همه branch ها (با شروع از آخرین commit):

>git log –all

برای مشاهده همه commit های قبلی حاصل از merge شدن branch ها (با شروع از آخرین commit):

>git log –merges

برای مشاهده log های commitهاست و به جای این دستور، استفاده از دستور git log توصیه می شود:

>git whatchanged

برای مشاهده خلاصه ای از دستور git log:

و نیز برای دانستن اینکه چه کسی روی چه چیزی کار کرده است:

>git shortlog

برای مشاهده جزئیات یک commit خاص:

>git show commitHashCode

مثال-

>git show 1fa88e2e12cff9b0fa08bc5169c81739054cfa11

در مقایسه با دستور git log، دستور git show جزئیات مربوط به یک commit خاص را نمایش می دهد.

>git show

برای ادغام یا merge کردن branch انتخابی در branch جاری:

>git merge branchName

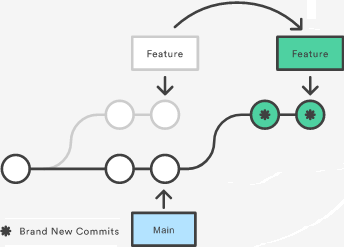
برای ادغام یا merge کردن branch انتخابی در branch جاری، با این شرط که تاریخچه commit های branch انتخابی در branch اصلی وارد نشود:

>git merge branchName –squash

برای ادغام یا merge کردن دو branch در یک branch (عملکرد آن مشابه git merge است.)

>git rebase base

Rebasing فرآیند ترکیب یا جابجایی دنباله ای از commitها در بالای یک commit پایه جدید است. git rebase فرآیند خطی ادغام است.



برای پیدا کردن commitهای بد:

گام 1- شروع کار

>git bisect start

گام 2- معرفی یک commit خوب به git:

>git bisect good a123

گام 3- معرفی یک commit بد به git:

>git bisect bad z123

برای انتخاب یک commit از یک branch و اعمال آن به branch های دیگر:

>git cherry-pick commitHashCode

برای ترکیب چند فایل در قالب یک فایل واحد:

>git archive --format zip HEAD > archive-HEAD.zip

دستور git archive چند فایل را در یک فایل واحد ترکیب می کند. مانند ابزار zip است یعنی می تواند فایلها را از حالت فشرده خارج کرده و به هرکدام از فایلها به طور جداگانه دسترسی بدهد.

یک فایل zip از بازبینی فعلی ایجاد می کند.

هنگام گرفتن کدها با دستور git pull معمولا باید rebase (و نه merge) را اجرا کنیم. این کار از merge های متعدد جلوگیری می کند.

>git pull –rebase

برای بررسی سطر به سطر محتوای یک فایل و تشخیص اینکه چه کسی هر تغییر را ایجاد کرده:

>git blame myFileName.format

مثال-

>git blame index.html

برای ایجاد یک tag جدید:

>git tag -a tagName

تگها در git مفید هستند و از آنها می توان برای مدیریت انتشار استفاده کرد. tag را می توان مانند یک branch بدون تغییر در نظر گرفت. اهمیت آن هنگام انتشار عمومی خیلی بیشتر است.

برای بررسی امضای gpg مربوط به commit ها: (GPG یا GNU Privacy Guard)

>git verify-commit <commit>

مثال-

>git verify-commit f309e6f3c44241dbf94b54c801e821d0b692d624

برای تایید tag:

>git verify-tag <tag>

مثال-

برای مقایسه فایل کاری با نسخه آن در git:

>git diff HEAD filename.format

مثال-

>git diff HEAD index.html

برای مقایسه دو branch:

>igt diff branch1 branch2

برای داشتن برنامه ای گرافیکی برای اجرای دستورات git:

>git citool

برای تغییر نام فایل در git:

>git mv oldFileName.format newFileName.format

مثال-

>git mv index.html firstPage.html

برای داشتن اطلاعات و راهنمایی مربوط به دستورات git:

>git help gitCommand

مثال-

>git help push

برای حذف فایلهایی که هنوز با git add به repository اضافه نشده اند:

>git clean

برای تنظیم ide پیشفرض git:

>git config --global core.editor myIDE

مثال-

>git config --global core.editor emacs

برای ایجاد alias برای دستورات پرکاربرد git:

>git config --global alias.myAlias myGitCommand

مثال-

>git config --global alias.i init

توسط alias می توانیم برای دستورات پرکاربرد خود، نامهای مستعاری تعریف کرده و با آنها دستورات خود را اجرا کنیم.

برای مشاهده درست داکومنتهای فارسی در دستور git status:

>git config --global core.quotepath false