«به نام خدا»



****

زمستان 1397

محتوای فنی یازدهمین نبرد هوش مصنوعی شریف

«بخش نخست: راهنمای Git»

مهران حسین‌زاده

فهرست مطالب

[**1.** **گیت چیست؟** 3](#_Toc748142)

[**2.** **شروع به کار با سایت GitHub** 3](#_Toc748143)

[**2.1.** **عضویت و ورود به سایت** 3](#_Toc748144)

[**2.2.** **ساختن repository** 3](#_Toc748145)

[**3.** **نصب git** 4](#_Toc748146)

[**4. اتصال repository و پوشة محل کد به هم** 4](#_Toc748147)

[**5.** **مفاهیم و دستورات مهم git** 5](#_Toc748148)

[**5.1.** **staging area** 5](#_Toc748149)

[**5.2.** **local repository** 6](#_Toc748150)

[**5.3.** **remote repository** 6](#_Toc748151)

[**5.4.** **stashing** 10](#_Toc748152)

[**5.5.** **branching** 12](#_Toc748153)

[**5.5.1.** **git branch** 12](#_Toc748154)

[**5.5.2.** **git checkout** 12](#_Toc748155)

[**5.5.3.** **git merge** 13](#_Toc748156)

[**6.** **مدیریت و مشاهدة تغییرات و وضیعت git** 13](#_Toc748157)

[**6.1.** **git log** 13](#_Toc748158)

[**6.2.** **gitk** 13](#_Toc748159)

[**6.3.** **git status** 13](#_Toc748160)

[**7.** **ابزارهای صفحة repository در GitHub** 14](#_Toc748161)

[**7.1.** **سربرگ Code** 14](#_Toc748162)

[**7.2.** **سربرگ Wiki** 14](#_Toc748163)

[**7.3.** **سربرگ Insights** 14](#_Toc748164)

[**8.** **امکانات git در IDEها** 14](#_Toc748165)

[**9.** **لینک‌های مفید و منابع** 15](#_Toc748166)

# **گیت چیست؟**

git ابزاری برای تسهیل مشارکت در انجام پروژه‌های کامپیوتری و مدیریت تغییرات و نسخه‌های مختلف آن است. با استفاده از git می‌توانید در قالب گروه‌های کوچک و بزرگ به انجام پروژه‌ها بپردازید، با امکانات آن از فعالیت‌ها و ریز‌تغییرات اعمال شده توسط هم‌گروهی‌هایتان اطلاع یابید، مسیر اجرای پروژه را شاخه‌بندی کنید و کار خود را با اطمینان خاطر با آن‌ها ادغام کنید. علاوه بر این، ابزار git حتی در پروژه‌های تک‌نفره نیز برای نگهداری تغییرات نسخه های مختلف کدها و مدیریت آن‌ها مفید است.

# **شروع به کار با سایت GitHub**

## **عضویت و ورود به سایت­­**

برای شروع به کار با گیت لازم است تا ابتدا حساب کاربری خود در سایت github.com را بسازید و وارد آن شوید. این کار به ‌راحتی در صفحة اصلی این سایت قابل انجام است.

## **ساختن repository**

repository در گیت، محلی برای ساختن و مدیریت یک پروژة جدید و تعریف افراد مشارکت‎کننده در آن است. برای ساختن یک repository جدید کافی است تا در صفحة شخصی خود و در بخش Repositories از دکمة New استفاده کنید.

سپس با انتخاب یک نام و توضیح اختیاری برای پروژه مورد نظرتان می‌توانید کار را شروع کنید. هم‌چنین در این صفحه تنظیماتی برای private یا public بودن repository و نیز اضافه شدن فایل‌هایی مثل .gitignore و README وجود دارد که دربارة آن جلوتر صحبت خواهیم کرد.

پس از ساختن repository و ورود به صفحة‌ آن، می‌توانید سایر اعضای گروه خود در github را از بخش Collaborators در زیر تنظیمات Settings به کمک نام کاربری آن‌ها یا آدرس ایمیل‌شان به این repository دعوت کنید. افرادی که به این شکل اضافه شوند، امکاناتی نظیر clone کردن پروژة گیت، دیدن فایل‌های آن و نیز اعمال تغییرات در فایل‌ها یا ساختار git را دارند؛ البته سازندة repository می‌تواند این امکانات یا دسترسی‌ها را در تنظیمات مربوط به repository تغییر دهد.

پس از ساختن repository لازم است تا محل(directory) مربوط به پروژه در کامپیوتر خود را به این repository متصل کنیم. رویة‌ کلی به این صورت است که شما می‌توانید در این محل تغییرات خود بر پروژة اصلی را اعمال کنید و سپس این تغییرات را با استفاده از دستورات git به اطلاع دیگر همکارانتان در پروژه نیز برسانید.

# **نصب git**

برای کار با repository در سیستم خودتان، ابتدا باید git را نصب کنید. برای نصب گیت بر روی سیستم‌عامل خود می‌توانید از این لینک استفاده کنید:

<https://git-scm.com/download>

# **اتصال repository و پوشة محل کد به هم**

در این مرحله کافی است تا در صفحة اصلی repository و از سربرگ Code لینک repository را کپی کرده و سپس در محل آدرس مورد نظرتان در ترمینال دستور clone را به صورت زیر وارد کنید:

git clone “your repository’s link”

اکنون همه چیز برای شروع کار آماده است. با این کار فایل‌های مربوط به پروژه به همراه برخی فایل‌های تنظیمات git به محل پروژه‌تان اضافه می‌شوند که مهم‌ترین آن‌ها عبارتند از:

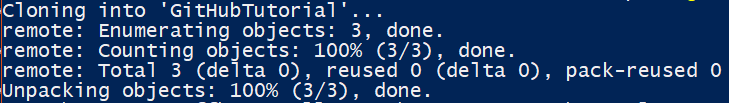
* .git: این پوشه که به صورت خودکار ساخته می‌شود و حاوی تنظیمات مخصوص git است، نشان‌دهندة آن است که direcotory مورد نظر یک git repository است(این پوشه در محل پروژة شما hidden است) .
* .gitignore: در هر پروژه ممکن است فایل‌هایی وجود داشته باشند که به علت حجم زیاد (مثل کتابخانه‌ها) یا مسائل امنیتی (مثل اطلاعات مربوط به استفاده از apiها در پروژه) و یا بی‌ارتباط بودن به کد پروژه (مثل فایل‌های .exe یا .o) نخواهیم در repository قرار داده شوند. این کار با اضافه کردن اسم یا الگوی این فایل‌ها یا directory‌ها به لیست موجود در فایلی به این نام ممکن است (این فایل نیز در محل پوشة پروژه hidden است) .

در لینک زیر می‌توانید الگوهای مفیدی برای استفاده در فایل .gitignore بسته به زبان برنامه‌نویسی خود، IDE مورد استفاده‌تان، سیستم‌عامل یا platformهای پرکاربرد پیدا کنید:

<https://www.gitignore.io>

* readme: این فایل به نوعی توضیح کلیت پروژه یا توضیحاتی درباره نحوة استفاده یا کارکرد بخش‌هایی از آن است که توصیه می‌شود در repository قرار داده شود.

از این جا به بعد یک repository را clone کرده و ادامه توضیحات را با آن جلو می‌بریم:



شکل 1 خروجی clone

# **مفاهیم و دستورات مهم git**

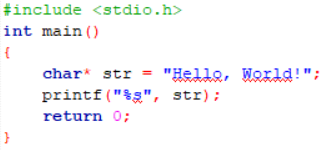
در این بخش به معرفی مفاهیم و دستورات مهم git می‌پردازیم. دقت کنید که تمام دستورات زده شده در ترمینال تنها در دیرکتوری مربوط به repository (که حاوی پوشه .git است) معنا دارند.

## **staging area**

staging area حافظه‌ای است که فایل‌ها یا پوشه‌هایی را که در directory کاری شما ساخته یا ویرایش شده‌اند و آمادة ثبت برای اضافه شدن به git هستند در خود نگه می‌دارد. برای اینکه ایجاد یا تغییر یک فایل یا پوشه‌ را در staging area اعمال کنیم از دستور add استفاده می‌کنیم؛ مثلا:

git add “your created/modified file or directory”

در ادامة کار با پروژة clone شده در قسمت قبل، فایل های HelloWorld.c و HelloWorld2.c را که دارای کد زیر هستند add می‌کنیم:



شکل 2 code فایل‌های HelloWorld.c و HelloWorld2.c



شکل 3 اضافه کردن فایل‌های کد به staging area

همچنین دستور زیر به صورت خودکار تمام فایل‌ها یا پوشه‌های جدید یا ویرایش شده را اضافه می‌کند:

git add .

طبیعی است که فایل‌ها یا پوشه‎های قرار گرفته در لیست .gitignore با دستورات بالا اضافه نمی‌شوند.

همچنین می‌توان با دستور زیر یک فایل اضافه شده به staging area را از آن حذف کرد؛ در این صورت در commit بعدی (که در بخش بعد توضیح داده‌خواهد شد) این فایل از این repository حذف خواهد شد:

git rm --cached “your created/modified file or directory”

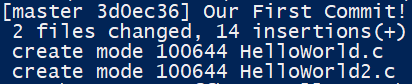
## **local repository**

فایل‌های اضافه شده به staging area باید در پایگاه‌داده (database) گیت ذخیره شوند تا بتوان تغییر اضافه شده در مرحله جدید را تحت عنوان commit اضافه کرد. local repository محلی برای نگهداری همین تغییرات اعمال شده، در کنار تغییرات احتمالی ایجاد شده توسط سایر مشارکت‌کنندگان پروژه است. با استفاده از دستور commit می‌توان فایل‌هایی را که با دستور add به staging area اضافه شده‌اند به local repository اضافه کرد. در هر commit اطلاعاتی نظیر تاریخ commit، پدیدآورندة آن و پیام آن ذخیره می‌شود. بسیار مهم است که commit message که با دستور زیر هنگام commit کردن معرفی می‌شود، توضیح مناسبی دربارة ماهیت تغییرات یا کدهای جدید اضافه شده با این commit داشته باشد:

git commit –m “your commit message”

در این جا تغییرات stage شده در مرحله قبل را commit می‌کنیم:





شکل 4 commit کردن و خروجی آن

## **remote repository**

remote repository نسخه‌ای از پروژة شماست که در سرورهای تحت وب git ذخیره شده‌اند؛ این همان جایی است که می‌توانید کد خود را با دیگر collaborator ها به اشتراک بگذارید. چنانچه پروژة خود را با روش clone ایجاد کرده باشید، نام این remote repository به صورت پیش‌فرض origin است. این نام را می‌توانید با دستور زیر مشاهده کنید:

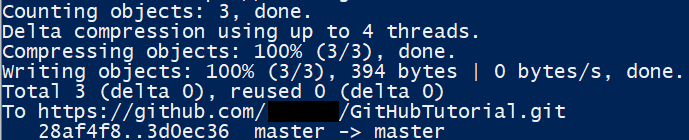
git remote

کار با remote repository دو بخش دارد:

* اضافه کردن تغییرات commit شدة خودتان به remote repository (که در local repository شما ذخیره شده‌اند): برای این منظور از دستور push استفاده می‌شود؛ که در آن origin نام remote repository است (دربارة branch در بخش بعدی توضیح داده شده است):

git push origin “your branchname, like master”

push کردن commit اعمال شده در مرحلة قبل با دستور git push origin master:



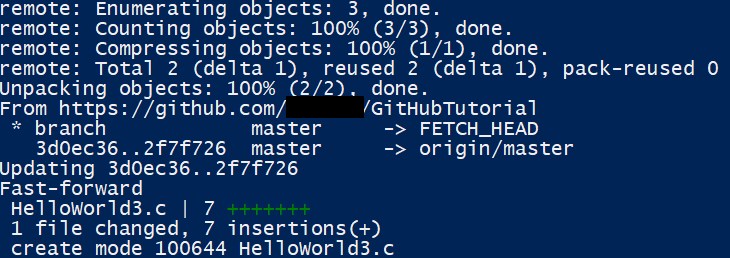
شکل 5 خروجی push

* گرفتن تغییرات commit شده در remote که هنوز در local شما ثبت نشده‌اند: برای این کار نیز از دستور pull استفاده می‌کنیم، که در آن نام branch و remote repository مانند دستور قبل معرفی می‌شوند:

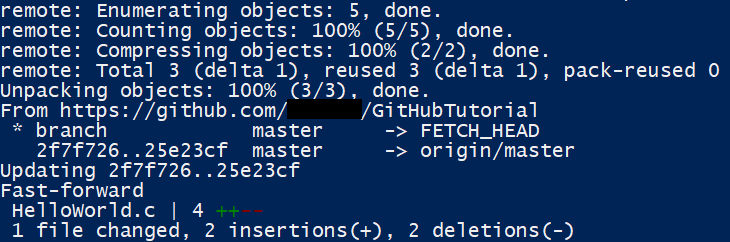
git pull origin “your branchname, like master”

در حقیقت عمل pull ترکیب دو عمل fetch و merge است؛ به این ترتیب که ابتدا در fetch تغییرات جدید اعمال شده در remote را به local می‌آورد و سپس در دستور merge آن‌ها را با وضعیت branch شما در local ادغام می‌کند. این ادغام گاهی اوقات به صورت اضافه کردن خط کدهایی است که توسط دیگران در remote ذخیره شده‌اند؛ اینگونه تغییرات را خود ابزار git به صورت خودکار مدیریت می‌کند. اما گاهی اوقات این ادغام باعث ایجاد conflict می‌شود؛ به این معنا که تغییرات اعمال شده در remote با نسخة موجود در local شما تضادی دارند که نیاز است به صورت آگاهانه و توسط کاربر دربارة آن تصمیم گرفته شود. در این باره جلوتر بیشتر صحبت خواهیم کرد.

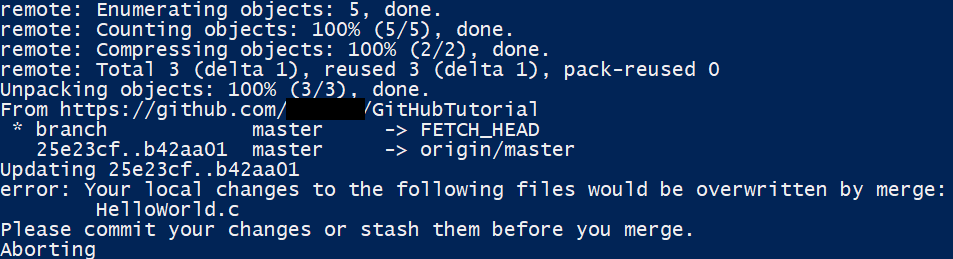
برای بررسی دقیق‌تر، pull را در سه حالت زیر درنظر می‌گیریم (تمام شکل‌ها خروجی دستور git pull origin master هستند):

* مشارکت‌کنندة دیگر در پروژه، تنها فایل‌ها یا directory‌های جدیدی را اضافه و commit کرده‌است؛ در این حالت با pull این فایل‌ها و directoryهای جدید اضافه می‌شوند. در مثال زیر، فایل HelooWorld3.c توسط شخص دیگری push شده است:

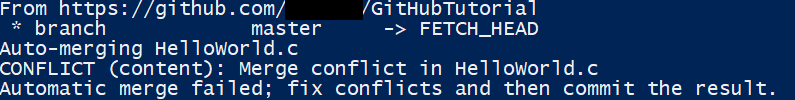
شکل 6 نتیجه pull و اضافه شدن فایل جدید HelloWorld3.c

* در این حالت، فایل HelloWorld.c توسط شخص دیگری ویرایش شده است (تبدیل نام متغیر str به HelloWorld\_str )، اما در working directory تغییری نسبت به آخرین وضعیت remote repository ایجاد نشده‌است. در این حالت git عمل auto-merge را انجام داده و ویرایش انجام‌شده در working directory اعمال می‌شود:

شکل 7 نتیجه pull و تغییر نام متغیر str (auto-merge)

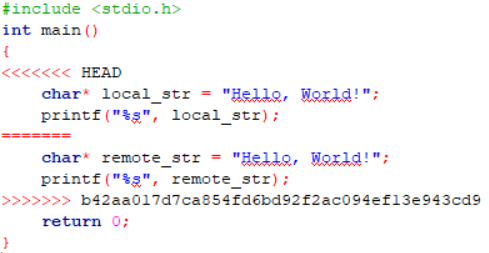
* در این حالت فرض کنید که کاربر دیگری نام متغیر را به remote\_str تبدیل کرده است و سپس این تغییر را commit و push کرده است؛ اما هم‌زمان کاربری که دستور pull را اجرا کرده‌است نام این متغیر را در working directory خود در حال حاضر به local\_str تغییر داده است. در این حالت نتیجه pull توسط این کاربر به صورت زیر است:

شکل 8 نتیجه pull با وجود تغییرات ذخیره نشده

**هرگاه یک کاربر تغییرات local ذخیره نشده‌ای داشته‌باشد، قبل از pull کردن این تغییرات باید stash یا commit شوند**. ( درباره stash در بخش های جلوتر توضیح داده خواهد شد) پس از add کردن و commit کردن تغییرات اعمال شده خروجی زیر داده می‌شود:

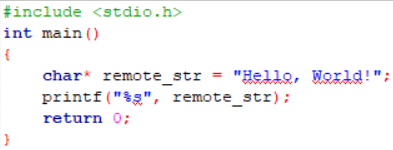
شکل 9 نتیجه pull پس از commit کردن local changes

برخلاف حالت قبل که وضعیت working directory دقیقا منطبق با آخرین state ذخیره شده از پروژه در remote repository بود، اینجا working directory حاوی تغییرات ذخیره‌نشده‌ای است که باید قبل از pull، با stash یا commit تعیین تکلیف شوند. پس از حل این مسئله با commit، خروجی pull بالا به این معنی است که git نتوانسته‌است به صورت خودکار، تغییرات اعمال شده را با حالت فعلی local ادغام کند (اگر تغییرات local کاربری که pull را اجرا کرده‌است مستقل از تغییرات اعمال شده به وسیلة pull بودند، همة این فرایند ادغام به صورت خودکار انجام می‌شد)؛ در این مرحله فایل HelloWorld.c برای این کاربر به صورت زیر است:



شکل 10 وضعیت فایل HelloWorld.c در working directory بعد از pull

در این فایل بخش‌هایی که git نتوانسته‌است آن‌ها را به‌صورت خودکار merge کند، با علامت‌های مخصوصی مشخص شده‌اند؛ تغییرات local از علامت <<<<<<< تا ======= و تغییرات حاصل از push از ======= تا >>>>>>> هستند. حال تصمیم برای انتخاب یکی از آن‌ها به عهدة خود کاربر است تا با ویرایش منطقی کد و مقایسه با کد pull شده نسخه نهایی را انتخاب کند. همچنین بعد از resolve کردن این conflict در فایل موردنظر، این تغییر نهایی باید commit و push شود.



شکل 11 فایل HelloWorld.c پس از ادغام دستی

## **stashing**

همانطور که در بخش قبل اشاره شد، در فرایند pull لازم است تا تغییرات local ذخیره‌نشده، ابتدا در جایی ذخیره شوند و پس از آن عمل pull انجام شود. یک راه برای این کار، مطابق بالا، commit کردن آن‌هاست؛ اما خیلی اوقات به دلایل منطقی تغییراتی که تاکنون در local خود اعمال کرده‌ایم هنوز کامل نیستند یا ایراداتی دارند که برای رفع آن‌ها و commit کردن آن‌ها به pull نیاز داریم. در این حالت می‌توان از stash استفاده کرد (از stash می‌توان برای switch کردن کار بین branchهای مختلف نیز استفاده کرد). stashها در واقع در یک پشته (stack) از تغییرات ناتمام (unfinished changes) ذخیره می‌شوند؛ این تغییرات نسبت به آخرین commit در این پشته ذخیره می‌شوند و می‌توانند به انتخاب کاربر در هر زمانی مجددا اعمال شوند. دستورات مرتبط با stashing عبارت‌اند از:

* stash کردن:

git stash

این دستور تغییرات ذخیره نشده را وارد پشته می‌کند.

* دیدن لیست stash ها:

git stash list

این دستور لیستی از همة stashهای موجود در پشته را نشان می‌دهد که هریک از آن‌ها با یک نام و توضیح مشخص شده‌است.

* حذف یک stash از پشته

git stash drop “stash\_name”

دستور بالا stashی را که با stash\_name مشخص شده است از پشته بیرون می‌اندازد (اسم stashها از لیستی که در قسمت قبل معرفی شد، قابل دیدن است). همچنین استفاده از این دستور به صورت git stash drop آخرین stash اضافه شده به پشته را پاک می‌کند.

* apply کردن stash

git stash apply “stash\_name”

دستور بالا stashی را که با stash\_name مشخص شده است، روی working directory اعمال کرده و باز می‌گرداند. همچنین استفاده از این دستور به صورت git stash apply به صورت خودکار آخرین stash اضافه شده به پشته را اعمال می‌کند.

* pop

git stash pop

این دستور ترکیب دو دستور git stash apply و git stash drop است.

بنابراین، pull کردن با وجود داشتن تغییرات local ذخیره‌نشده‌ای که نمی‌خواهیم commitشان کنیم، به صورت مجموعه دستورات زیر قابل انجام است:

git stash

git pull

git stash pop

پس از آن، رفع کردن conflictهای احتمالی مشابه توضیحات قبل است.

## **branching**

همانطور که گفته‌شد، git برای ذخیرة پروژه، commitها را در پایگاه‌دادة موجود در remote repository ذخیره می‌کند. branch در واقع یک اشاره‌گر (pointer) به یک commit خاص در repository است؛ commit ها در هر branch به صورت linked list در ادامة همین اشاره‌گر اضافه می‌شوند. از branch می‌توان برای تقسیم کردن و مدیریت ماهیت تغییرات ایجاد شده در repository استفاده کرد. به صورت پیش‌فرض، هر repository یک branch به نام master دارد که branch اصلی پروژه است. دستورهای مهم برای کار با branch عبارتند از:

### **git branch**

از این دستور می‌توان به دو صورت استفاده کرد:

* نشان دادن branch فعلی:

git branch

* ایجاد یک branch جدید:

git branch “your new branchname”

### **git checkout**

در ساختار ذخیرة commitها، اشاره‌گری به نام HEAD وجود دارد که محل آخرین commit در remote را نشان می‌دهد؛ به این ترتیب commit جدید به صورت پیوندی در ادامة همین commit می‌آید. این دستور اشاره‌گر HEAD را به محل آخرین commit موجود در branch خاصی تغییر می‌دهد. به این ترتیب commit های جدید در این branch جدید ثبت می‌شوند.

git checkout “branchname”

### **git merge**

مطابق آنچه اشاره شد، با branch در واقع از یک commit به بعد، دنبالة‌ commitهای جدید می‌تواند از branch قبلی جدا شود تا در مسیری جدید این تغییرات ثبت شوند. این امر می‌تواند مثلا برای اضافه کردن یک ویژگی جدید به پروژة اصلی باشد؛ به همین منظور کدهای مربوط به اعمال این ویژگی و تست آن در یک branch جدید نوشته می‌شود. اما در نهایت و با تکمیل کار در این branch، لازم است تا این تغییرات به مسیر اصلی پروژه (مثلا در master) اضافه شوند؛ به طور مثال، برای ادغام branch با master branch لازم است تا پس از checkout کردن به branch مورد نظر در آن از دستور زیر استفاده شود:

git merge master

به‌طور کلی می‌توان برای اعمال تغییر جدیدی در پروژه که جدای از مسیر اصلی پروژه است یا وظایف مختلفی که بین افراد مختلف تقسیم شده‌اند، از محل‌های شروع تصمیم برای اضافه کردن این ویژگی‌ها یک branch جدید ایجاد کرد؛ سپس با checkout کردن به آن branch می‌توان commitهای مربوط به آن را در این مسیر مجزا از branch اصلی انجام داد؛ پس از اتمام این ویرایش‌ها و اطمینان از درستی آن‌ها می‌توان آن را در master ادغام کرد.

# **مدیریت و مشاهدة تغییرات و وضیعت git**

## **git log**

این دستور یک log از اطلاعات همة commitهای اعمال شده می‌دهد. همچنین، به هر commit یک نام داده می‌شود که از طریق این دستور می‌توان این نام‌ها را نیز مشاهده کرد.

git log

## **gitk**

gitk ابزاری است برای رصد وضعیت شماتیک branchها و commitها. با این ابزار می‌توان علاوه بر نام و سازندة هر commit و تاریخ ایجاد آن، در هر commit دلخواه، نسخة قدیمی و اولیة فایل‌های تغییر کرده در طی آن commit یا تغییرات ایجاد شده با این commit ( diff ) را نیز مشاهده کرد.

## **git status**

این دستور گزارشی از وضعیت فعلی repository می‌دهد؛ اطلاعات نشان داده شده در خروجی این دستور عبارتند از:

* branch فعلی که در آن هستیم
* وضعیت up-to-date بودن یا نبودن local repository با remote repository
* تغییرات stage شده که نیازمند commit هستند
* تغییرات stage نشده
* فایل‌هایی که به git اضافه نشده اند - untracked files

# **ابزارهای صفحة repository در GitHub**

در این بخش به معرفی اجمالی تعدادی از امکانات مفید صفحة repository در GitHub می‌پردازیم:

## **سربرگ Code**

در این سربرگ می‌توانید بدون نیاز به clone کردن پروژه در سیستم خود، فایل‌های مربوط به پروژه را مشاهده یا جست‌و‌جو کنید یا فایل‌های جدیدی اضافه کنید.

## **سربرگ Wiki**

از سربرگ Wiki در صفحة اصلی repository می‌توانید مستنداتی دربارة مراحل انجام پروژه، مسئولیت‌ها، وضعیت فعلی پروژه، نحوة اجرای پروژه یا .... تهیه کنید. این مستندات در بخش pages از همین سربرگ قابل مشاهده و ویرایش هستند. همچنین در این سربرگ می‌توان لینک مربوط به این wikiها را دریافت و مشابه قبل clone کرد تا بتوان به این فایل‌ها دسترسی داشت.

## **سربرگ Insights**

این بخش اطلاعات آماری خلاصه‌ای دربارة میزان مشارکت افراد مختلف در repository و commitها بر حسب زمان می‌دهد.

# **امکانات git در IDEها**

با گسترش کاربرد git در پروژه‌های کامپیوتری مهم و بزرگ، بسیاری از IDEها دستورات مهم و پرکاربرد git یا امکانات مناسب برای merge کردن کدها و رفع conflictها را با رابط‌های گرافیکی مناسب یا میانبرها در خود تعبیه کرده‌اند. به عنوان مثال، برای آشنایی با این روش‌ها در محصولات شرکت JetBrains می‌توانید مستندات آن را در لینک زیر پیدا کنید:

<https://www.jetbrains.com/help/idea/using-git-integration.html>

# **لینک‌های مفید و منابع**

* **لینک‌های مفید**
* <https://git-scm.com/download/>

(راهنمای نصب git)

* <https://gitexplorer.io/>

(سایتی مفید برای آشنایی با دستور مرتبط با کاربرد مورد نظرتان در git)

* <https://gitignore.io/>

(سایتی مفید برای آشنایی با الگوهای مورد استفاده در تهیة فایل .gitignore)

* <https://www.jetbrains.com/help/idea/using-git-integration.html>

(مستند شرکت JetBrains برای استفاده از ابزارهای git در IDEهای این شرکت)

* **منابع**
* <https://guides.github.com/activities/hello-world/>
* <https://git-scm.com/book/en/v2/Getting-Started-Git-Basics/>
* <https://git-scm.com/docs/git-commit/>
* <https://git-scm.com/book/en/v1/Git-Tools-Stashing/>
* <https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging/>
* <https://clubmate.fi/git-removing-files-from-the-staging-area-and-the-tree/>