

«به نام خدا»



یازدهمین نبرد  
هوش مصنوعی  
شریف  
**SHARIF**  
**AI CHALLENGE**  
دانشگاه صنعتی شریف  
دانشکده مهندسی کامپیوتر  
**2019**

زمستان ۱۳۹۷

محتوای فنی یازدهمین نبرد هوش مصنوعی شریف

«بخش نخست: راهنمای Git»

مهران حسینزاده

## فهرست مطالب

۱. گیت چیست؟ ..... ۳
۲. شروع به کار با سایت **GitHub** ..... ۳
  - ۲.۱. عضویت و ورود به سایت ..... ۳
  - ۲.۲. ساختن **repository** ..... ۳
  ۳. نصب **git** ..... ۴
  ۴. اتصال **repository** و پوشه محل کد به هم ..... ۴
  ۵. مفاهیم و دستورات مهم **git** ..... ۵
    - ۵.۱. **staging area** ..... ۵
    - ۵.۲. **local repository** ..... ۶
    - ۵.۳. **remote repository** ..... ۶
    - ۵.۴. **stashing** ..... ۱۰
    - ۵.۵. **branching** ..... ۱۲
      - ۵.۵.۱. **git branch** ..... ۱۲
      - ۵.۵.۲. **git checkout** ..... ۱۲
      - ۵.۵.۳. **git merge** ..... ۱۳
  ۶. مدیریت و مشاهده تغییرات و وضعیت **git** ..... ۱۳
    - ۶.۱. **git log** ..... ۱۳
    - ۶.۲. **gitk** ..... ۱۳
    - ۶.۳. **git status** ..... ۱۳
  ۷. ابزارهای صفحه **repository** در **GitHub** ..... ۱۴
    - ۷.۱. سربرگ **Code** ..... ۱۴
    - ۷.۲. سربرگ **Wiki** ..... ۱۴
    - ۷.۳. سربرگ **Insights** ..... ۱۴
  ۸. امکانات **git** در **IDE** ها ..... ۱۴
  ۹. لینک‌های مفید و منابع ..... ۱۵

## ۱. 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 نسخه‌های مختلف کدها و مدیریت آن‌ها مفید است.

## ۲. شروع به کار با سایت GitHub

### ۲.۱. عضویت و ورود به سایت

برای شروع به کار با گیت لازم است تا ابتدا حساب کاربری خود در سایت [github.com](https://github.com) را بسازید و وارد آن شوید. این کار به راحتی در صفحه اصلی این سایت قابل انجام است.

### ۲.۲. ساختن repository

repository در گیت، محلی برای ساختن و مدیریت یک پروژه جدید و تعریف افراد مشارکت‌کننده در آن است. برای ساختن یک repository جدید کافی است تا در صفحه شخصی خود و در بخش Repositories از دکمه 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 درباره آن جلوتر صحبت خواهیم کرد.

پس از ساختن repository و ورود به صفحه 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 پروژه گیت، دیدن فایل‌های آن و نیز اعمال تغییرات در فایل‌ها یا ساختار git را دارند؛ البته سازنده 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. رویه کلی به این صورت است که شما می‌توانید در این محل تغییرات خود بر پروژه

اصلی را اعمال کنید و سپس این تغییرات را با استفاده از دستورات git به اطلاع دیگر همکارانتان در پروژه نیز برسانید.

### ۳. نصب git

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

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

### ۴. اتصال repository و پوشه محل کد به هم

در این مرحله کافی است تا در صفحه 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، نشان‌دهنده آن است که directory مورد نظر یک git repository است (این پوشه در محل پروژه 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 پوشه پروژه hidden است).
- در لینک زیر می‌توانید الگوهای مفیدی برای استفاده در فایل .gitignore بسته به زبان برنامه‌نویسی خود، IDE مورد استفاده‌تان، سیستم عامل یا platformهای پرکاربرد پیدا کنید:

<https://www.gitignore.io>

- **readme**: این فایل به نوعی توضیح کلیت پروژه یا توضیحاتی درباره نحوه 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:

```
Cloning into 'GitHubTutorial'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

شکل ۱ خروجی clone

## ۵. 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.

### ۵.۱ staging area

staging area حافظه‌ای است که فایل‌ها یا پوشه‌هایی را که در directory کاری شما ساخته یا ویرایش شده‌اند و آماده 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؛ مثلاً:

```
git add "your created/modified file or directory"
```

در ادامه کار با پروژه clone شده در قسمت قبل، فایل‌های HelloWorld.c و HelloWorld2.c را که دارای کد زیر هستند add می‌کنیم:

```
#include <stdio.h>
int main()
{
    char* str = "Hello, World!";
    printf("%s", str);
    return 0;
}
```

شکل ۲ code فایل‌های HelloWorld.c و HelloWorld2.c

```
git add HelloWorld.c HelloWorld2.c
```

شکل ۳ 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 داده خواهد شد) این فایل از این repository حذف خواهد شد:

```
git rm --cached "your created/modified file or directory"
```

## ۵.۲. local repository

فایل‌های اضافه شده به staging area باید در پایگاه داده (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، پدیدآورنده آن و پیام آن ذخیره می‌شود. بسیار مهم است که commit message که با دستور زیر هنگام commit کردن معرفی می‌شود، توضیح مناسبی درباره 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:

```
git commit -m "Our First Commit!"
[master 3d0ec36] Our First Commit!
2 files changed, 14 insertions(+)
create mode 100644 HelloWorld.c
create mode 100644 HelloWorld2.c
```

شکل ۴ commit کردن و خروجی آن

## ۵.۳. remote repository

remote repository نسخه‌ای از پروژه 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 پروژه خود را با روش clone ایجاد کرده باشید، نام این remote repository به صورت پیش فرض origin است. این نام را می‌توانید با دستور زیر مشاهده کنید:

```
git remote
```

کار با remote repository دو بخش دارد:

- اضافه کردن تغییرات commit شده 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 (درباره branch در بخش بعدی توضیح داده شده است):

```
git push origin "your branchname, like master"
```

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

```
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 394 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/[redacted]/GitHubTutorial.git
28af4f8..3d0ec36 master -> master
```

شکل ۵ 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 نسخه موجود در local شما تضادی دارند که نیاز است به صورت آگاهانه و توسط کاربر درباره آن تصمیم گرفته شود. در این باره جلوتر بیشتر صحبت خواهیم کرد.

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

◀ مشارکت‌کننده دیگر در پروژه، تنها فایل‌ها یا `directory`های جدیدی را اضافه و `commit` کرده‌است؛ در این حالت با `pull` این فایل‌ها و `directory`های جدید اضافه می‌شوند. در مثال زیر، فایل `HelloWorld3.c` توسط شخص دیگری `push` شده است:

```
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 2 (delta 1), reused 2 (delta 1), pack-reused 0
Unpacking objects: 100% (2/2), done.
From https://github.com/[redacted]/GitHubTutorial
* branch          master      -> FETCH_HEAD
  3d0ec36..2f7f726  master      -> origin/master
Updating 3d0ec36..2f7f726
Fast-forward
 HelloWorld3.c | 7 +++++++
 1 file changed, 7 insertions(+)
 create mode 100644 HelloWorld3.c
```

شکل ۶ 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:

```
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/[redacted]/GitHubTutorial
* branch          master      -> FETCH_HEAD
  2f7f726..25e23cf  master      -> origin/master
Updating 2f7f726..25e23cf
Fast-forward
 HelloWorld.c | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
```

شکل ۷ 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 کرده است 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:

```
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/[redacted]/GithubTutorial
* branch          master      -> FETCH_HEAD
   25e23cf..b42aa01 master      -> origin/master
Updating 25e23cf..b42aa01
error: Your local changes to the following files would be overwritten by merge:
       Helloworld.c
Please commit your changes or stash them before you merge.
Aborting
```

شکل ۸ نتیجه pull با وجود تغییرات ذخیره نشده

هرگاه یک کاربر تغییرات local ذخیره نشده‌ای داشته باشد، 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:

```
From https://github.com/[redacted]/GithubTutorial
* branch          master      -> FETCH_HEAD
Auto-merging Helloworld.c
CONFLICT (content): Merge conflict in Helloworld.c
Automatic merge failed; fix conflicts and then commit the result.
```

شکل ۹ نتیجه pull پس از commit کردن local changes

برخلاف حالت قبل که وضعیت working directory دقیقاً منطبق با آخرین state ذخیره شده از پروژه در remote repository بود، اینجا working directory حاوی تغییرات ذخیره نشده‌ای 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 نتوانسته است به صورت خودکار، تغییرات اعمال شده را با حالت فعلی local ادغام کند (اگر تغییرات local کاربری که pull را اجرا کرده است مستقل از تغییرات اعمال شده به وسیله pull بودند، همه این فرایند ادغام به صورت خودکار انجام می‌شد)؛ در این مرحله فایل HelloWorld.c برای این کاربر به صورت زیر است:

```
#include <stdio.h>
int main()
{
    <<<<<<< HEAD
    char* local_str = "Hello, World!";
    printf("%s", local_str);
    =====
    char* remote_str = "Hello, World!";
    printf("%s", remote_str);
    >>>>>>> b42aa017d7ca854fd6bd92f2ac094ef13e943cd9
    return 0;
}
```

شکل ۱۰ وضعیت فایل HelloWorld.c در working directory بعد از pull

در این فایل بخش‌هایی که git نتوانسته است آن‌ها را به صورت 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 عهده 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.

```
#include <stdio.h>
int main()
{
    char* remote_str = "Hello, World!";
    printf("%s", remote_str);
    return 0;
}
```

شکل ۱۱ فایل HelloWorld.c پس از ادغام دستی

#### ۵.۴. stashing

همانطور که در بخش قبل اشاره شد، در فرایند pull لازم است تا تغییرات local ذخیره نشده، 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 مجدداً 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 همه stash‌های موجود در پشته را نشان می‌دهد که هر یک 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 ذخیره نشده‌ای که نمی‌خواهیم commit شان کنیم، به صورت مجموعه دستورات زیر قابل انجام است:

```
git stash  
git pull  
git stash pop
```

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

## ۵.۵. branching

همانطور که گفته شد، git برای ذخیره پروژه، commit ها را در پایگاه داده موجود در remote repository ذخیره می‌کند. branch در واقع یک اشاره گر (pointer) به یک commit خاص در repository است؛ commit ها در هر branch به صورت linked list در ادامه همین اشاره گر اضافه می‌شوند. از branch می‌توان برای تقسیم کردن و مدیریت ماهیت تغییرات ایجاد شده در repository استفاده کرد. به صورت پیش فرض، هر repository یک branch به نام master دارد که branch اصلی پروژه است. دستورهای مهم برای کار با branch عبارتند از:

### ۵.۵.۱. 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"
```

### ۵.۵.۲. git checkout

در ساختار ذخیره commit ها، اشاره گری به نام HEAD وجود دارد که محل آخرین commit در remote را نشان می‌دهد؛ به این ترتیب commit جدید به صورت پیوندی در ادامه همین commit می‌آید. این دستور اشاره گر 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"
```

## ۵.۵.۳. git merge

مطابق آنچه اشاره شد، با branch در واقع از یک commit به بعد، دنباله commit های جدید می تواند از branch قبلی جدا شود تا در مسیری جدید این تغییرات ثبت شوند. این امر می تواند مثلاً برای اضافه کردن یک ویژگی جدید به پروژه اصلی باشد؛ به همین منظور کدهای مربوط به اعمال این ویژگی و تست آن در یک branch جدید نوشته می شود. اما در نهایت و با تکمیل کار در این branch، لازم است تا این تغییرات به مسیر اصلی پروژه (مثلاً 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
```

به طور کلی می توان برای اعمال تغییر جدیدی در پروژه که جدای از مسیر اصلی پروژه است یا وظایف مختلفی که بین افراد مختلف تقسیم شده اند، از محل های شروع تصمیم برای اضافه کردن این ویژگی ها یک branch جدید ایجاد کرد؛ سپس با checkout کردن به آن branch می توان commit های مربوط به آن را در این مسیر مجزا از branch اصلی انجام داد؛ پس از اتمام این ویرایش ها و اطمینان از درستی آن ها می توان آن را در master ادغام کرد.

## ۶. مدیریت و مشاهده تغییرات و وضعیت git

## ۶.۱. git log

این دستور یک log از اطلاعات همه commit های اعمال شده می دهد. همچنین، به هر commit یک نام داده می شود که از طریق این دستور می توان این نام ها را نیز مشاهده کرد.

```
git log
```

## ۶.۲. gitk

gitk ابزاری است برای رصد وضعیت شماتیک branch ها و commit ها. با این ابزار می توان علاوه بر نام و سازنده هر commit و تاریخ ایجاد آن، در هر commit دلخواه، نسخه قدیمی و اولیه فایل های تغییر کرده در طی آن commit یا تغییرات ایجاد شده با این commit (diff) را نیز مشاهده کرد.

## ۶.۳. git status

این دستور گزارشی از وضعیت فعلی repository می دهد؛ 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

## ۷. ابزارهای صفحه repository در GitHub

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

### ۷.۱. 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.

### ۷.۲. سربرگ Wiki

از سربرگ Wiki در صفحه اصلی repository می‌توانید مستنداتی درباره مراحل انجام پروژه، مسئولیت‌ها، وضعیت فعلی پروژه، نحوه 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.

### ۷.۳. سربرگ Insights

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

## ۸. 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>

## ۹. 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 تهیه 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/>