بسمه تعالی

محتوای فنی

یازدهمین نبرد هوش مصنوعی شریف

راهنمای گیت

بهمن و اسفند 1397

گیت چیست؟

GitHub 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 ازفعالیت‌ها 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 برای پروژه‌های تکی 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 صفحه اصلی این سایت قابل انجام است.

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 مدنظرتان 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 درباره‌ی 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 کد 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.

برای این کار باید مراحل زیر طی شود:

نصب Git

برای نصب گیت بر روی سیستم‌عامل خود می‌توانید از این لینک استفاده کنید:

https://git-scm.com/download

اتصال Repository و پوشه محل کد به هم

برای این کار 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”
```

با این کار فایل‌های مربوط به پروژه به همراه برخی فایل‌های تنظیمات git به آن دیرکتوری اضافه می‌شوند که مهم‌ترین آن‌ها عبارتند از:

- .git : این پوشه که به صورت خودکار ساخته می‌شود و حاوی تنظیمات مخصوص git است، نشان‌دهنده‌ی آن است که دیرکتوری مورد نظر یک git repository 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 این فایل‌ها یا دیرکتوری‌ها به لیست موجود در فایلی به این نام ممکن است.
- readme : این فایل به نوعی توضیح کلیت پروژه یا توضیحاتی درباره نوه‌ی 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 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 دیکتوری مربوط به repository (که حاوی پوشه git. است.) معنا دارند.

staging area

این بخش شامل فایل‌هایی است که در directory کاری شما ساخته یا ویرایش شده‌اند و آماده‌ی ثبت برای اضافه شدن به git هستند. برای اعمال ایجاد یا تغییر فایل یا پوشه‌ای به staging area از دستور add استفاده می‌شود؛ مثلا:

```
git add “your created/modified file or directory”
```

در ادامه مثال مطرح شده در بالا، فایل های HelloWorld.c و HelloWorld2.c که دارای کد زیر هستند، add می‌کنیم:

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

شکل 2 code فایل‌های HelloWorld.c و HelloWorld2.c

```
git add HelloWorld.c 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 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، نیاز دارند تا در دیتابیس 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 repo اضافه کرد. در هر commit اطلاعاتی نظیر تاریخ commit، پدیدآورنده‌ی آن و پیام آن ذخیره می‌شود. بسیار مهم است که commit message که با دستور زیر هنگام commit کردن معرفی می‌شود، توضیح مناسبی درباره‌ی 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 4 commit کردن و خروجی آن

remote repository

remote repository نسخه‌ی پروژه شماست که در سرورهای تحت اینترنت git ذخیره شده‌اند؛ این همان جایی است که می‌توانید کد خود را با دیگر collaborator ها به اشتراک بگذارید. چنانچه پروژه‌ی 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 شده‌ی 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 5خروجی 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 می آورد 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 نسخه‌ی موجود در local شما تضادی دارند که نیاز است به صورت آگاهانه و توسط کاربر درباره‌ی آن تصمیم گرفته شود. در این باره جلوتر بیشتر صحبت خواهیم کرد.

برای توضیح بیشتر، pull را در 3 حالت زیر ادامه توضیح می‌دهیم (تمام اشکال خروجی دستور git pull origin master هستند):

- مشارکت‌کننده‌ی دیگر در پروژه، تنها فایل‌ها یا دیرکتوری‌های جدیدی را اضافه و commit کرده‌است؛ در این حالت، با pull این فایل‌ها و دیرکتوری‌های جدید اضافه می‌شوند. در مثال زیر، فایل HelooWorld3.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 +++++++
```

لکشد 6 هجیتpull دیدج لیاف زدش هفاضا وHelloWorld3.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:
- در این حالت فرض کنید که کاربر 2 نام متغیر را به remote_str تبدیل کرده است و سپس این تغییر را commit و push کرده است؛ اما هم‌زمان کاربر 1 نام این متغیر را در

```
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 8 خروجی دستور pull هنگامی که تغییرات دیگری در ریپازیتوری وجود دارد

```
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 7 خروجی دستور pull هنگامی که تغییرات در ریپازیتوری remote (auto-merge) شده باشد

working directory خود در حال حاضر به local_str تغییر داده است. در این حالت نتیجه pull توسط کاربر 1 به صورت زیر است:

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:

```
From https://github.com/[redacted]/GitHubTutorial
 * branch            master     -> FETCH_HEAD
Auto-merging HelloWorld.c
```

شکل 9 خروجی دستور pull از commit نکردن local changes

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 کاربر 1 مستقل از تغییرات اعمال شده به وسیله push بودند، همه‌ی این فرایند ادغام به صورت خودکار انجام می‌شد)؛ در این مرحله فایل HelloWorld.c برای کاربر 1 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 10 وضعیت فایل HelloWorld.c در working directory کاربر 1

در این بخش‌هایی که توسط git نتوانسته اند، 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 عهده‌ی خود کاربر است تا با ویرایش منطقی کد و مقایسه با کد push شده نسخه نهایی را انتخاب کند.

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

شکل 11فایل 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 اینکار 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 استک از تغییرات ناتمام (unfinished changes) ذخیره می‌شوند؛ این تغییرات نسبت به آخرین commit در این استک 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 استک می‌کند.

- دیدن لیست stash ها:

```
git stash list
```

این دستور، لیستی از همه‌ی stashهای موجود در stack می‌دهد که هریک از آن‌ها با یک نام و توضیح مشخص‌شده‌است.

- حذف یک stash از استک

```
git stash drop “stash_name”
```

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

- apply کردن stash

```
git stash apply “stash_name”
```

دستور بالا، stashای را که با stash_name مشخص شده است، روی working directory اعمال کرده و بازمی‌گرداند. همچنین استفاده از این دستور به صورت git stash apply به صورت خودکار آخرین stash اضافه شده به استک را اعمال می‌کند.

- pop

```
git stash pop
```

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

بنابراین، برای 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 ذخیره‌ی پروژه commit ها را در دیتابیس‌های موجود در remote repository ذخیره می‌کند. branch در واقع یک پوینتر به یک commit خاص در repository است؛ commit ها در هر branch به صورت linked list در ادامه‌ی همین pointer اضافه می‌شوند. از branch می‌توان برای تقسیم کردن و مدیریت ماهیت تغییرات ایجاد شده در repository ایجاد 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 ذخیره commitها، پوینتری 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، پوینتر HEAD را به محل آخرین commit موجود در branch خاصی تغییر پیدا می‌کند. 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 commit ای به بعد، دنباله‌ی commit های 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 پروژه‌ی اصلی باشد؛ به همین منظور کدهای مربوط به اعمال این ویژگی و تست آن در یک branch جدید انجام 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 انجام داد؛ پس از اتمام این ویرایش‌ها و اطمینان از درستی آن‌ها می‌توان به master آن را merge کرد.

مدیریت و مشاهده‌ی تغییرات و وضعیت git

git log

این دستور یک log از اطلاعات همه commit های 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 سازنده‌ی هر commit و تاریخ ایجاد آن، در هر commit دلخواه، نسخه قدیمی و اولیه 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 صفحه repository در GitHub

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

سربرگ Code

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

سربرگ Insights

این بخش اطلاعات آماری خلاصه‌ای درباره‌ی میزان مشارکت افراد مختلف در repository و commitها و کلیات آماری آن‌ها به تفکیک 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 مناسبی 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