



گیت

تهیه کننده: امیر حلاجی بیدگلی

VCS ها چیستند؟

ورژن کنترل سیستمی است که به منظور کنترل و پیگیری تغییرات اعمال شده در پروژه‌های نرم افزاری مورد استفاده قرار می‌گیرد. سیستم‌های کنترل ورژن ماهیتی مانند دیتابیس دارند، با این تفاوت که اطلاعات درون آن شامل نسخه‌های گوناگون از پروژه و داده‌های مربوط به هر از آن‌ها است. این دیتابیس داده به برنامه نویسان این امکان را می‌دهند تا هر زمانی که بخواهند، ورژن مورد نظر از پروژه خود را ذخیره و یا فراخوانی کنند.

مزیت استفاده از VCS ها چیست؟

قبل از پرداختن به جواب این سوال، به سوال های زیر پاسخ دهید.

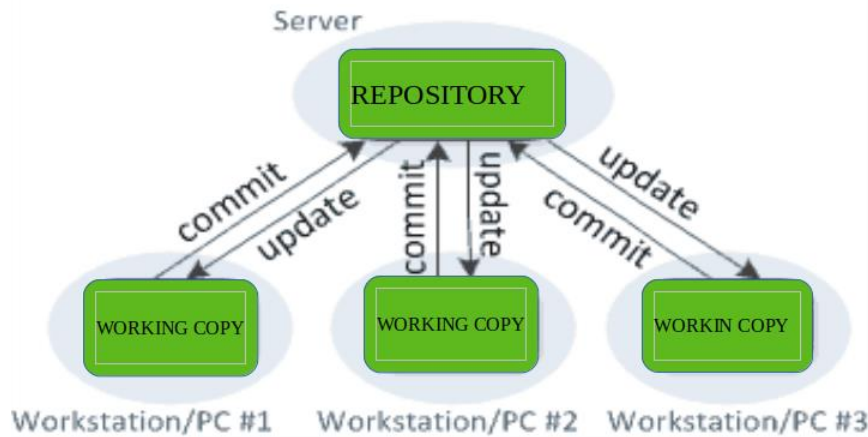
- ❖ تا به حال شده در کد یا برنامه خود تغییری به وجود آورده باشید که بخوانید آن را به حالت(های) قبل بازگردانید؟
- ❖ تا به حال شده که بخوانید نسخه های مختلف از یک برنامه را نگه دارید؟
- ❖ آیا تا به حال خواسته اید که نیاز به یک مرور کلی از تاریخچه کد خود داشته باشید؟
- ❖ آیا تا به حال شده کد خود را گم کنید؟
- ❖ آیا تا به حال شده که بخوانید کد خود را به اشتراک بگذارید؟
- ❖ آیا تا به حال براتون پیش اومده که بخوانید در کد دیگران تغییر ایجاد کنید؟
- ❖ آیا تا به حال براتون پیش اومده که با دوستتون به صورت گروهی روی پروژه ای کار کنید و هر کدام در حین قبول کردن بخش های مختلف پروژه نیاز به کد دیگری داشته باشید؟

به طور خلاصه تمام مشکلات بالا، با VCS ها حل میشوند.

انواع VCS ها و مقایسه آنها

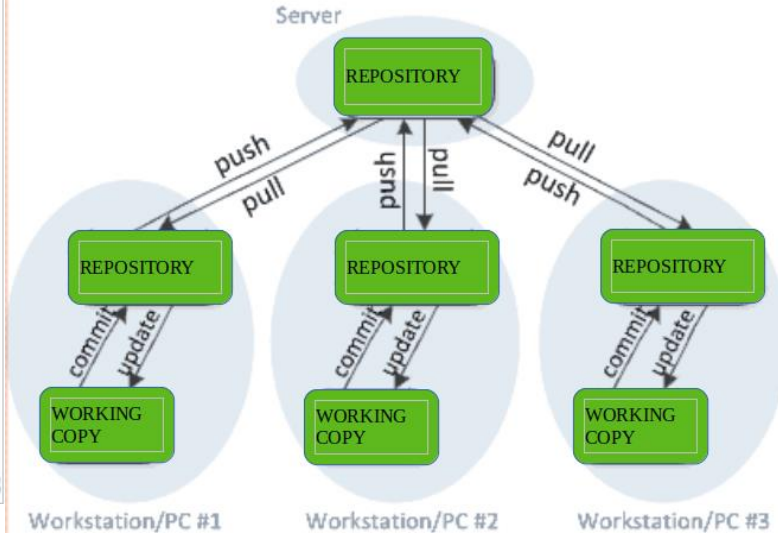
Centralized
(CVCS)

Centralized version control



Distributed
(DVCS)

Distributed version control

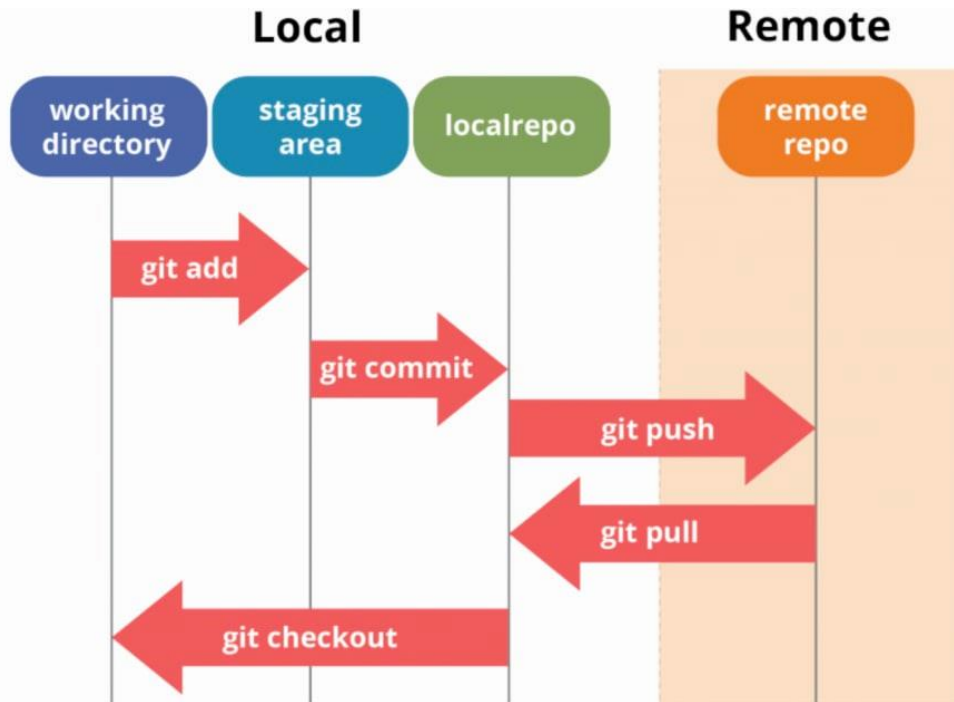


گیت‌هاب و گیت لب چیستند؟ با گیت چه تفاوت‌هایی دارند؟

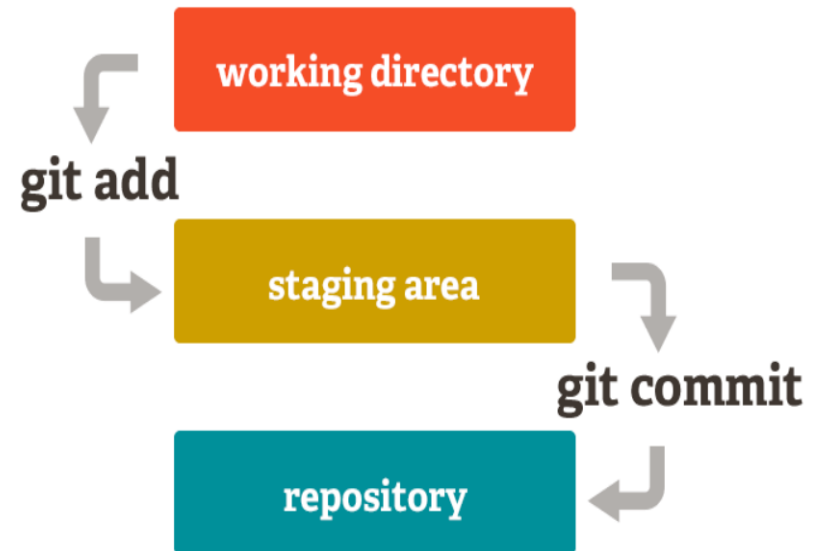
Git vs GitHub Comparison

GIT	GITHUB
Installed locally	Hosted in the cloud
First released in 2005	Company launched in 2008
Maintained by The Linux Foundation	Purchased in 2018 by Microsoft
Focused on version control and code sharing	Focused on centralized source code hosting
Primarily a command-line tool	Administered through the web
Provides a desktop interface named Git Gui	Desktop interface named GitHub Desktop
No user management features	Built-in user management
Minimal external tool configuration features	Active marketplace for tool integration
Competes with Mercurial, Subversion, IBM, Rational Team Concert and ClearCase	Competes with Atlassian Bitbucket and GitLab
Open source licensed	Includes a free tier and pay-for-use tiers

خلاصہ کار



Picture(1)



Picture(2)

کامند های گیت

1- git init

این دستور یک ریپازیتوری خالی ایجاد میکند. این دستور در واقع اولین گام برای ساختن یک ریپازیتوری است.

.....

2-git add <file or directory name>

این دستور فایل یا دایرکتوری مورد نظر شما را به Stage Area اضافه میکند. اگر میخواهید تمام فایل های خود را اضافه کنید، کافیست که بعد از add، **نقطه (.)** بگذارید..

.....

3-git status

با این دستور پی میبریم که فایل ها در کدام یک از وضعیت ها قرار دارند، به طور مثال اگر پس از دستور (**git add .**) این کامند را استفاده کنیم، تمامی فایل ها به رنگ سبز نمایش داده میشوند و این به این معنی است که این فایل ها وارد Stage Area شده اند؛ اگر فایل ها وارد Stage Area نشده باشند، با رنگ قرمز نمایش داده میشوند.

4-git commit -m "your commit message"

با این دستور تمام تغییرات در ریپازیتوری ذخیره می شوند.(دستور کامیت). پس از این دستور با دستور (git status) استعلام بگیرید و می بینید که عبارت (nothing to commit, working tree clean) ظاهر میشود، به این معنا که فایلی برای کامیت موجود نیست.

5-git commit -a -m "message"

با این دستور فایل شما در یک مرحله وارد ریپازیتوری میشود، یعنی کامند های شماره ۲ و ۴ را باهم انجام می دهید.البته این به شرطی است که حتما قبل از آن یک بار فایلتان را به Stage Area اضافه کرده باشید.

6-git rm --cached <file name>

این دستور فایل مورد نظر شما را از Stage Area حذف می کند.

7-git rm --cached -r .

این دستور کلیه فایل ها را از Stage Area حذف می کند

نکته: اگر از این دستور استفاده کنید، دیگر نمیتوانید با استفاده از دستور شماره ۵ مستقیم کامیت کنید؛ بلکه باید دوباره با استفاده از شماره ۲ ابتدا فایل(های) مورد نظرتان را به Stage Area اضافه کنید و سپس کامیت کنید.

.....

8-git diff

برای مشاهده آخرین تغییرات بین زمان حال و آخرین اضافه شدن به Stage ، از این دستور استفاده کنید.

.....

9-git log

این دستور تاریخچه کامیت های انجام شده را به ترتیب آخرین کامیت به شما نمایش میدهد. مواردی مانند نام نویسنده، ایمیل و تاریخ و پیام کامیت مورد نظر به شما نمایش داده می شود. با دستور (git log --online) میتوانید تاریخچه کامیت های خود را خط به خط مشاهده کنید. یا مثلاً اگر دستور (git log -2) را استفاده کنید، دو کامیت آخر به شما نمایش داده می شوند.

نکته: ممکن است اسم و ایمیل شما نمایش داده نشود. برای حل این مشکل چه باید کرد؟

✓ راه حل

```
10-git config --global user.name "your name"
```

```
11-git config --global user.email "your email"
```

این دو دستور را بهتر است قبل از ایجاد کردن ریپازیتوری اجرا کنید که اسم و ایمیلتان را ثبت کنید. در اینصورت در تاریخچه کامیتهایتان، اسم و ایمیلی را که ثبت کرده‌اید، مشاهده خواهید کرد.

بخش دوم دستورات

ریست کردن - بخش اول

ممکن است که پس از تغییراتی روی تعدادی از فایل ها ایجاد کرده باشید متوجه شوید که تغییرات اشتباه بوده اند و بخواهید آن ها را به حالت قبل برگردانید؛ در واقع در این حالت، وارد Stage Area نشده اید.

12- `git checkout --<file name>`

برای همه به جای `<file name>` از نقطه (.) استفاده کنید.

نکته: این دستور مانند `Ctrl + z` عمل می کند و فقط مال زمانی است که تغییراتتان در مرحله ابتدایی قرار دارد و وارد Stage Area ، و یا کامیت نشده اند.

ادامه بخش دوم دستورات

ریست کردن - بخش دوم

در این مرحله فرض کنید که فایل های تغییر داده شده را به Stage Area اضافه کرده اید و حال میخواهید تغییراتتان را از آنجا حذف کنید. درست است که این دستور باعث برگشت از Stage Area میشود ولی همچنان میتوان از دستور شماره ۵ استفاده کرد.

13- git reset HEAD <file name>

```
free@amir MINGW64 ~/Desktop/New folder (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   hello.py

free@amir MINGW64 ~/Desktop/New folder (master)
$ git reset HEAD hello.py
Unstaged changes after reset:
M       hello.py

free@amir MINGW64 ~/Desktop/New folder (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   hello.py

no changes added to commit (use "git add" and/or "git commit -a")
```

ادامه بخش دوم دستورات

ریست کردن-بخش سوم

در این مرحله فرض کنید که با دستوری مشابه دستور شماره ۴ یا ۵ کامیت انجام داده اید و حالا به هر دلیلی پشیمان شدید و میخواهید تغییرات پروژه را به یک کامیت خاص برگردانید. برای اینکار باید ID کامیت ها را بدانید

نکته: برای دانستن ID کامیت ها کافیست از دستور شماره ۹ استفاده کنید.

14-git reset --hard <commit-id>

```
free@amir MINGW64 ~/Desktop/New folder (master)
$ git log
commit 7a3f2a5d85e7f4b127ccc9b80b1454f5456d88c9 (HEAD -> master)
Author: amirhallaji <a.hallaji.b@gmail.com>
Date: Wed Sep 16 10:46:33 2020 +0430

    second.cpp added to the project

commit 20a96908e0e87d4563453e5ed2d7b0f540c6f102
Author: amirhallaji <a.hallaji.b@gmail.com>
Date: Wed Sep 16 10:43:07 2020 +0430

    message

commit 0c847ad105f0d6891ee20488da14918a826275fc
Author: amirhallaji <a.hallaji.b@gmail.com>
Date: Wed Sep 16 10:20:32 2020 +0430

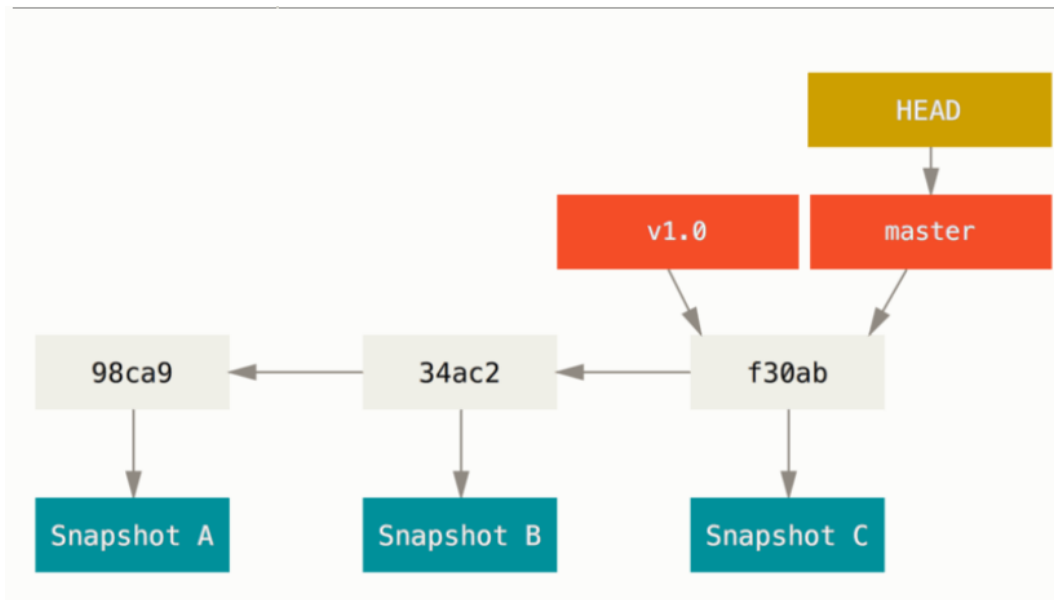
    initial commit

free@amir MINGW64 ~/Desktop/New folder (master)
$ git reset --hard 20a96908e0e87d4563453e5ed2d7b0f540c6f102
HEAD is now at 20a9690 message
```

برنچ ها (Branches)

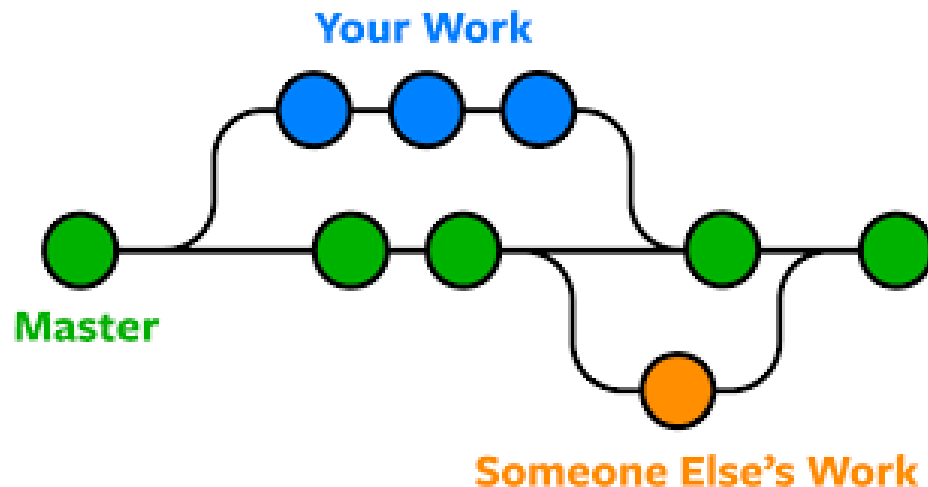
برنچ در لغت به معنی شاخه و یا شعبه می باشد و طبق تعریف سایت مرجع گیت، هر برنچ یک نشانگر سبک وزن قابل انتقال به یکی از کامیت ها است.

برنچ پیش فرض، برنچ مستر نام دارد. هنگامی که شما کامیت انجام می دهید، به شما یک برنچ مستر داده شده که به آخرین کامیت شما اشاره میکند



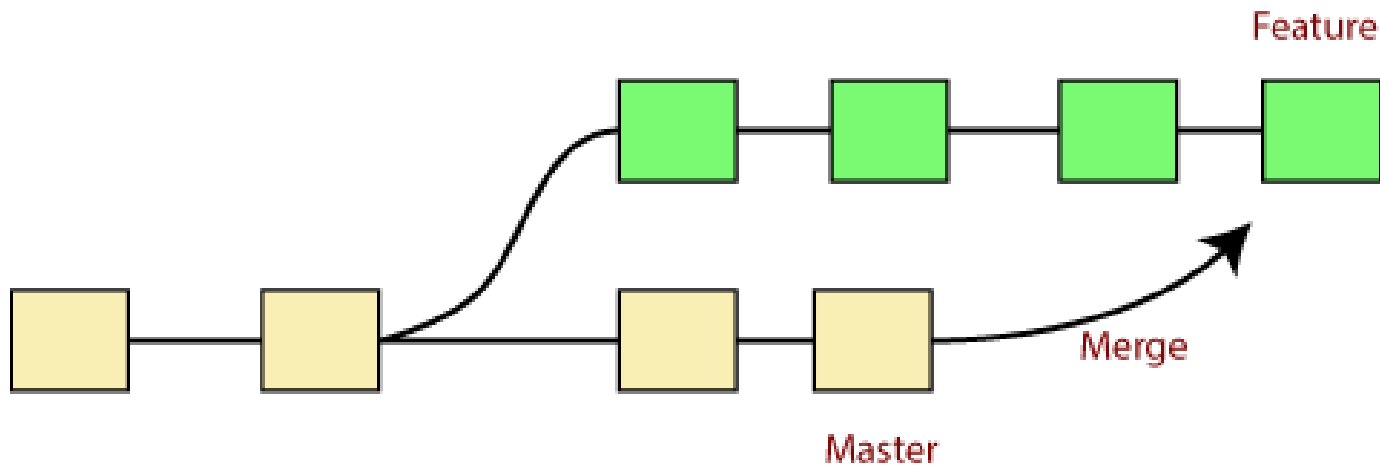
کاربرد برنچ چیست؟

فرض کنید که در حال توسعه یک اپلیکیشن می باشید و نسخه های متفاوتی از آن را تهیه کرده اید. بنا بر نیاز، شما باید هر ورژن را به صورت جداگانه نگه دارید. لذا الان که گیت بلدیم، از برنچ استفاده میکنیم و نسخه های مختلف را در برنچ های مختلف نگهداری میکنیم. با این اقدام، به طور مثال میتوانیم نسخه اولیه پروژه خود را حفظ کرده و به توسعه ورژن دوم پردازیم.



مفهوم merge یا ادغام

فرض کنید که با یکی از دوستان خود در حال توسعه یک اپلیکیشن هستید. شما کد بخش های فرانت و دوستان کدهای بخش بک را میزنند. در انتها برای اینکه پروژه کامل شود، این دو بخش باید باهم ادغام (merge) شوند.



بخش سوم-کامند های branch

14- git branch -h

با این دستور تمام کامند های مربوط به branch را مشاهده می کنید.

15- git branch <name>

با این دستور یک branch با نام name می سازید.

16- git branch -a

با این دستور لیست تمام branch ها را مشاهده می کنید.

ادامه بخش سوم

17- git checkout <name>

با این دستور از branch فعلی به branch ای با نام name منتقل می شوید.

18- git branch -d <name>

با این دستور، branch با نام name را حذف می کنید.

ادامه

19- git checkout -b <name>

این دستور هم شاخه ای با نام `name` می سازد و به صورت همزمان به این شاخه سوییچ نیز میکند.

نکته: تغییراتی که در هر `branch` اعمال میکنید، فقط در همان `branch` اعمال می شوند. به طور مثال اگر یک شاخه با نام `dev` و یک شاخه اصلی (`master`) نیز داشته باشید، اگر در شاخه `dev` تغییراتی اعمال کنید و به شاخه `master` بازگردید، پروژه به حالت قبل بازمیگردد. (این یکی از خوبی های `branch` است)

.....

20- git merge <name>

نکته: برای اینکه ببینیم این ادغام به درستی صورت پذیرفته است، می توانیم از دستور `git log --graph` استفاده کنیم و لاگ ها را مشاهده کنیم.

بخش چهارم - کار با گیت‌هاب

برای اینکه بخواهیم پروژه خود را در گیت‌هاب قرار بدهیم، ابتدا باید یک ریپازیتوری بسازید. سپس برای ریموت زدن به پروژه گیت‌هاب، از دستور زیر استفاده کنید.

21- `git remote add origin <repository link>`

.....

22- `git push -u origin master`

این دستور کامیت شما را در `master branch` پوش می‌کند. با اینکار انگار کامیت‌هایتان را آپلود کرده‌اید.

.....

23- `git pull origin`

این دستور، تمام فایل‌ها و فولدرهای حاوی پروژه شما با پروژه موجود در ریپازیتوری گیت‌هابتان سینک می‌شود. (دریافت پروژه)

ادامه بخش چهارم

24- git clone <repository link>

با این دستور پروژه موجود در ریپازیتوری را دانلود می کنید.
اکنون پس از دانلود کردن پروژه، هر تغییری را که اعمال کنید، با دستورات قبلی نظیر (git status or git diff) مشاهده خواهید کرد.

بخش پنجم دستورات

فرض کنید که در پروژه‌مان فایل‌ها و فولدرهایی داریم که نمی‌خواهیم به وسیله گیت دنبال شوند. برای اینکار ابتدا باید فایلی به نام `.gitignore` در پروژه‌مان بسازیم.

25- touch .gitignore

فرض کنید که می‌خواهیم تمام فایل‌های فرمت `txt` را ایگنور کنیم؛ همچنین اگر بخواهیم گیت فولدری را دنبال نکند کارهای زیر را انجام می‌دهیم:

```
Untracked files:
(use "git add <file>..." to include in what will be committed)

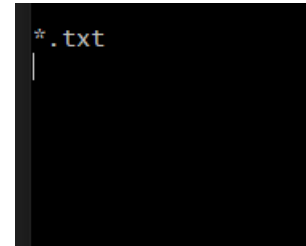
.gitignore
hello.py

free@amir MINGW64 ~/Desktop/New folder (master)
$ git add .
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory.

free@amir MINGW64 ~/Desktop/New folder (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   .gitignore
        modified:   hello.py

free@amir MINGW64 ~/Desktop/New folder (master)
$
```



```
*.txt
```

محتوای فایل

`.gitignore`

دنبال نکردن (untrack) کردن فایل یا دایرکتوری در گیت

ممکن است که بخواهیم فایل یا فولدری را که قبلاً track می‌شد را untrack کنیم؛ به طور مثال می‌خواهیم دایرکتوری‌ای به نام Cpp داریم که می‌خواهیم از این به بعد توسط گیت دنبال نشود. به این منظور ابتدا باید در فایل .gitignore عبارت Cpp/ را بنویسیم؛ سپس دستور . git rm --cached -r را اجرا کنیم؛ سپس دستور . git add را فراخوانی کنیم؛ اکنون اعلام بگیرید تا ببینید درست انجام شده یا خیر.

```
free@amir MINGW64 ~/Desktop/New folder (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   .gitignore
        new file:   Cpp/second.cpp
        new file:   Cpp/test.cpp
        modified:   hello.py

free@amir MINGW64 ~/Desktop/New folder (master)
$
```

1

```
free@amir MINGW64 ~/Desktop/New folder (master)
$ cat .gitignore
*.txt
Cpp/

free@amir MINGW64 ~/Desktop/New folder (master)
$ git rm --cached -r .
rm '.gitignore'
rm 'Cpp/second.cpp'
rm 'Cpp/test.cpp'
rm 'hello.py'

free@amir MINGW64 ~/Desktop/New folder (master)
$ git add .
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory.

free@amir MINGW64 ~/Desktop/New folder (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   .gitignore
        modified:   hello.py

free@amir MINGW64 ~/Desktop/New folder (master)
$ |
```

2