

**دانشکده ریاضی و علوم کامپیوتر**

**برنامه‌سازی پیشرفته و کارگاه**

**آشنایی با گیت**

استاد درس

دکتر مهدی قطعی

استاد دوم

بهنام یوسفی مهر

نگارش

صالح ملازاده ، محمدرضا شیخ الاسلامی

بهار ۱۴۰۳

**فهرست**

[Version Control چیه و چرا مهمه ؟ 3](#_Toc190089228)

[شروع کار با گیت 6](#_Toc190089229)

[گیر افتادی؟ یه نگاهی به کمک‌های گیت بنداز! 7](#_Toc190089230)

[تنظیمات اولیه گیت 8](#_Toc190089231)

[CRLF چیه و چرا انقدر معروفه؟ 9](#_Toc190089232)

[اولین قدم: راه‌اندازی یه مخزن گیت 10](#_Toc190089233)

[: .gitignoreدوست گیت برای نادیده گرفتن فایل‌ها 11](#_Toc190089234)

[git status : وضعیت پروژه رو بررسی کن! 11](#_Toc190089235)

[Staging Area چیه؟ 14](#_Toc190089236)

[git commit : تغییراتت رو ثبت کن ! 17](#_Toc190089237)

[git log : دیدن تاریخچه کامیت ها و برگشت به گذشته ! 25](#_Toc190089238)

[git revert : ماشین زمان گیت ! 28](#_Toc190089239)

[ویدیو برای یادگیری بهتر 30](#_Toc190089240)

Version Control چیه و چرا مهمه ؟

اگه تاحالا روی یه پروژه‌ی کدنویسی کار کرده باشید، احتمالاً تجربه‌ی اینو داشتید که یه تغییراتی ایجاد کنید و بعدش بفهمید که اوضاع خراب شده! یا شاید خواستید روی یه قابلیت جدید کار کنید ولی نگران بودید که تغییراتتون کدهای قبلی تون رو خراب کنن . **Version Control** اینجاست که وارد بازی می‌شه!

مشکلات کار بدون Version Control :

**۱- عدم امکان بازگشت به نسخه‌های قبلی:** اگه تغییری ایجاد کنید که باعث خراب شدن کد بشه، بدون ورژن کنترل عملاً راهی برای برگشتن به نسخه‌ی سالم قبلی نداری، مگر اینکه خودت به‌صورت دستی از پروژه نسخه‌های مختلف ذخیره کرده باشی که کار سخت و گیج‌کننده‌ایه.

**۲- مدیریت سخت فایل‌ها:** معمولاً افراد بدون ورژن کنترل، از روش‌هایی مثل کپی کردن فایل‌ها با اسم های مختلف استفاده می کنن مثلا project\_final\_v2\_fixed\_really\_final که هم باعث سردرگرمی میشه و هم احتمال از دست رفتن اطلاعات رو بالا میبره.

**۳- مشکلات همکاری تیمی:** اگه چند نفر همزمان روی یه پروژه کار کنن، بدون ورژن کنترل ممکنه فایل‌ها رو روی همدیگه بنویسن یا تغییرات یکدیگر رو از دست بدن. در بهترین حالت، باید کلی هماهنگ کنن که کی روی کد کار کنه تا مشکلی بوجود نیاد.

**۴- عدم مستندسازی تغییرات:** وقتی تغییراتی در پروژه انجام بدید، اگه نسخه‌ای از قبل نداشته باشید یا جایی ثبت نکنید که چرا این تغییر رو دادید، بعداً ممکنه یادتون بره که این تغییر چه دلیلی داشته و روند توسعه رو سخت کنه.

**۵- ریسک از بین رفتن پروژه:** اگه کامپیوترتون خراب بشه یا فایل‌ها به هر دلیلی پاک بشن، بدون داشتن نسخه‌های متعدد در جای امن، ممکنه کل زحماتتون هدر بره.

Version Control System (VCS) چیه؟

سیستم کنترل نسخه (VCS) یه ابزاریه که کمک می‌کنه تغییراتتون توی کدهاتون رو مدیریت کنید. با این سیستم، می‌تونید نسخه‌های مختلف از پروژه‌اتون رو ذخیره کنید، تغییرات رو دنبال کنید و هر وقت لازم شد، به یه نسخه‌ی قبلی برگردید. این یعنی دیگه لازم نیست کلی فایل با اسم‌های عجیب‌وغریب مثل final\_version\_v2\_fixed\_really\_final درست کنید!

چرا Version Control مهمه؟

**۱- برگشت‌پذیری:** اگه یه تغییری زدید که باعث خراب شدن پروژه شد، می‌توندی خیلی راحت به یه نسخه‌ی قبلی برگردید.

**۲- همکاری تیمی:** وقتی چند نفر روی یه پروژه کار می‌کنن، VCS باعث می‌شه که تغییرات همزمان بدون تداخل مدیریت بشن و بتونید متوجه بشید هر تغییر روی پروژه توسط چه شخصی انجام داده شده.

**۳- مستندسازی تغییرات:** هر تغییری که انجام می‌دید، با یه توضیح ثبت می‌شن که بعداً بفهمید چرا این تغییر رو دادید.

**۴- مدیریت شاخه‌ها:** می‌تونید یه شاخه جدید برای ویژگی‌های جدید درست کنید و بعد از تست، اون رو به پروژه‌ی اصلی اضافه کنید.

نحوه عملکرد Version Control به زبان ساده

Version Control به این صورت کار می‌کنه که هر بار تغییراتی در فایل‌های پروژه ایجاد می‌کنید، اون تغییرات در قالب یک نسخه (commit) ثبت می‌شن. هر commit یه پیام توضیحی داره که مشخص می‌کنه چه تغییری انجام شده. بعد از ثبت، این تغییرات در یک مخزن (repository) ذخیره می‌شن و هر وقت لازم باشه، می‌تونید به نسخه‌های قبلی برگردید.

برای مدیریت بهتر، می‌تونی شاخه‌های (branches) مختلفی ایجاد کنید تا روی ویژگی‌های جدید کار کنید، بدون اینکه به نسخه‌ی اصلی آسیب بزنید. در نهایت، می‌تونید تغییرات رو با نسخه‌ی اصلی (main branch) ترکیب کنی. **نترسید !** همه ی این واژه ها رو در ادامه توضیح میدیم

Git چیه و چه ربطی به VCS داره؟

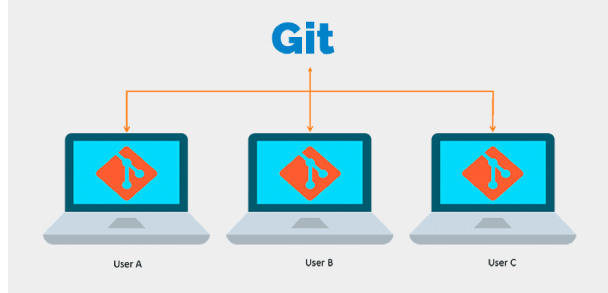
Git یکی از محبوب‌ترین سیستم‌های کنترل نسخه‌ است که به‌صورت توزیع‌شده کار می‌کنه. یعنی

هر کس یه نسخه‌ی کامل از پروژه رو روی سیستم خودش داره و نیازی نیست همش به یه سرور مرکزی وابسته باشی. حالا ورژن کنترل ها یا همون سیستم های کنترل نسخه به ۲ دسته اصلی **سیستم های متمرکز**  و **سیستم های توزیع شده** تقسیم میشن که در موضوع بعدی تفاوتشون رو بهتر درک می کنید !

تفاوت بین سیستم‌های کنترل ورژن متمرکز و توزیع‌شده

سیستم‌های متمرکز و **توزیع‌شده مثل** ( Git) از نظر کارایی یه هدف دارن، ولی فرقشون توی نحوه‌ی مدیریت داده‌هاست.توی سیستم‌های متمرکز، یه سرور مرکزی داریم که آخرین نسخه‌های پروژه روی اون ذخیره میشه. یعنی همه‌ی اعضای تیم با یه نسخه‌ی واحد از پروژه کار می‌کنن که روی اون سرور قرار داره. اما توی سیستم‌های توزیع‌شده مثل Git، هر کسی یه کپی کامل از کل پروژه روی سیستم خودش داره. این یعنی حتی اگه اینترنت نداشته باشید، باز هم می‌تونید روی کدتتون کار کنید و به جای یه سرور مرکزی، هر عضو تیم یه نسخه‌ی کامل از مخزن رو روی سیستم خودش داره.

وقتی توسعه‌دهنده‌ها از Git استفاده می‌کنن، هر کسی یه کپی از کل پروژه داره که شامل همه‌ی تغییراتی میشه که از روز اول تا الان اتفاق افتاده.

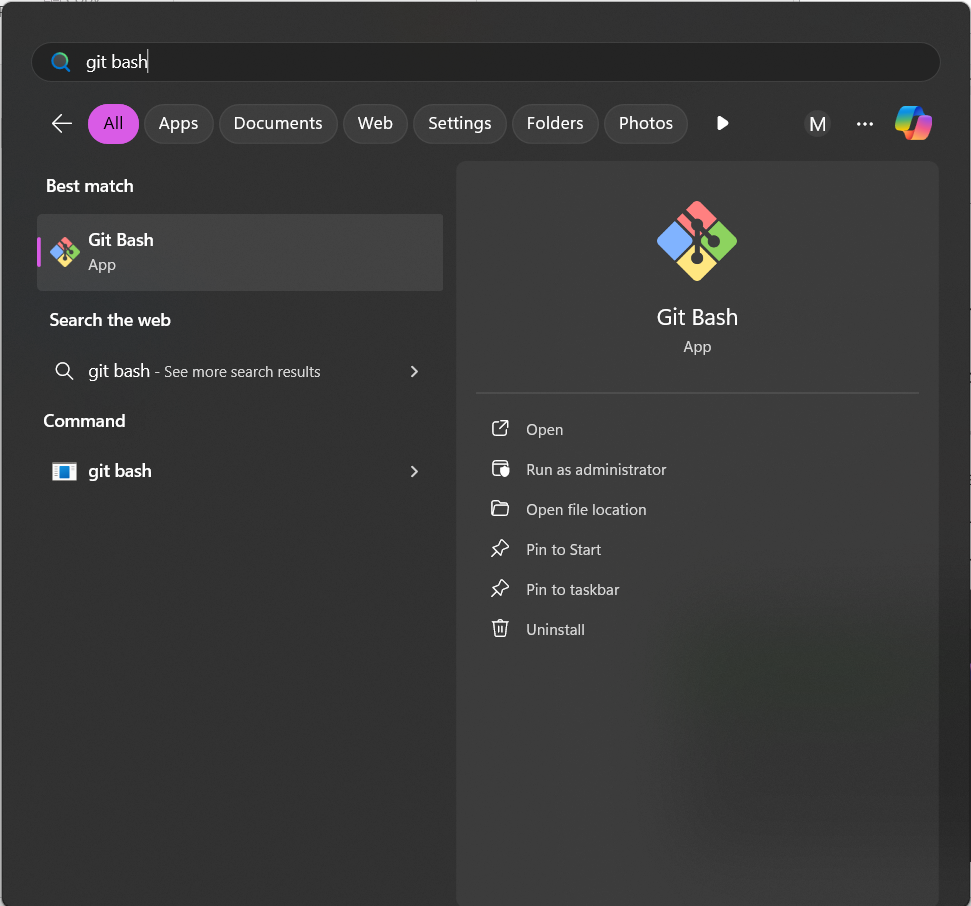
توی بخش بعدی کار کردن با گیت رو شروع می کنیم !

شروع کار با گیت

اولین دستوری که می‌خوایم باهم دیگه امتحان کنیم، دستور دریافت ورژن گیت عه. این دستور رو ممکنه خیلی وقت ها ازش استفاده کنید هم برای اینکه ببینید گیت به خوبی روی سیستمتون نصبه یا اینکه صرفا چه نسخه‌ای از گیت رو در حال حاضر روی سیستمتون دارید.

برای اینکار کافیه توی یک Command Prompt دستور **“git –v”** رو اجرا کنیم.

برای همراه شدن با داک، بهتره که اگه توی ویندوز هستین، **Git Bash** رو اجرا کنید (معمولا همراه با نصب گیت، روی ویندوزتون نصب می‌شه) و اگر با لینوکس یا مک هستین، کافیه که Terminal اتون رو اجرا کنید. در پایین، تصویر فضای Git Bash رو مشاهده میکنید.





حالا وقتشه که دستور زیر رو اجرا کنید برای اینکه ورژن گیت خودتون رو بفهمید :

❯ git -v

git version 2.45.2

همونطور که میبینید، Git ورژن 2.45.2 روی سیستم نصبه.

گیر افتادی؟ یه نگاهی به کمک‌های گیت بنداز!

گیت یه دستور داره که هروقت گیرافتادین یا چیزی یادتون رفت، سریع می‌تونید ازش کمک بگیرید!

برای اینکار باید دستور زیر رو اجرا کنید:

❯ git -h

بخشی از خروجی ای که ممکنه ببینید به صورت زیره:

usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]

          [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

          [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]

          [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

          [--config-env=<name>=<envvar>] <command> [<args>]

پایین تر پرکاربردی‌ترین دستورات رو هم براتون نشون داده:

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

  clone     Clone a repository into a new directory

  init      Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

  add       Add file contents to the index

  mv        Move or rename a file, a directory, or a symlink

  restore   Restore working tree files

  rm        Remove files from the working tree and from the index

اینجوری هروقت مطمئن نبودین که باید چیکار میکردین یا دستوری رو یادتون رفت، به راحتی می‌تونید از خود گیت کمک بگیرید!

تنظیمات اولیه گیت

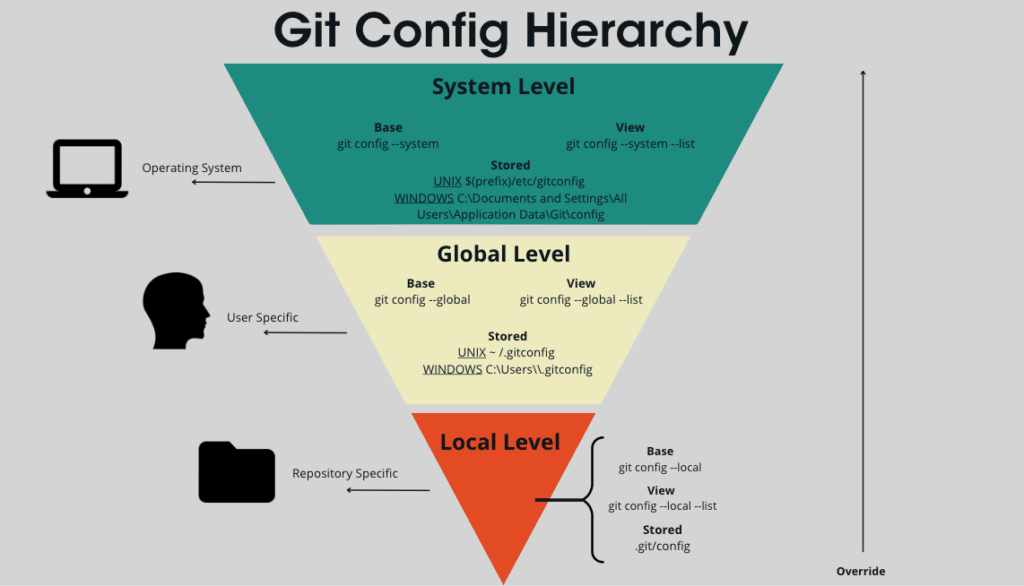
خب، حالا که مطمئن شدی گیت رو روی سیستمت نصب کردی، اولین چیزی که باید انجام بدی اینه که خودت رو به گیت معرفی کنی. چرا؟ چون گیت می‌خواد بدونه چه کسی داره این تغییرات رو انجام می‌ده. در ادامه متوجه می‌شی که چرا باید اطلاعاتت رو به گیت بدی!

برای این کار دو تا تنظیم مهم داریم: اسم و ایمیل.

برای اینکه اسمت رو به گیت بگی باید دستور زیر رو اجرا کنی:

❯ git config --global user.name "Jesper"

شاید بپرسی “--global” یعنی چی؟ تو می‌تونی تنظیمات گیت رو بر اساس سطح‌بندی های متفاوتی انجام بدی. در اینجا ما می‌خوایم تنظیماتمون صرفا برای یوزر فعلی تغییر کنه.



برای ثبت ایمیل هم دستور زیر رو اجرا کنید:

❯ git config --global user.email "molioo1298@gmail.com"

بد نیست که اشاره‌ای هم به ویرایشگر متن(Text Editor) پیش‌فرض گیت هم داشته باشیم. در ادامه کار با گیت، خیلی وقتا برای اینکه بتونی یه کاری رو با گیت به انجام برسی، مجبوری می‌شی که یه متن رو ادیت کنی. حالا اینجا به گیت میگیم که ما دوست داریم با چه ویرایشگری کار کنیم:

❯ git config --global core.editor "code --wait"

به عنوان مثال من اینجا به گیت گفتم که من می‌خوام متن‌هایی که باید ادیت بزنم رو با VS Code انجام بدم. شماها می‌تونین از ویرایشگرهای دیگه‌ای مثل Vim یا Nano هم استفاده کنید؛ چیزی رو انتخاب کنید که بیشتر از همه باهاش راحتین! ویرایشگر پیش‌فرض گیت وقتی بهش ویرایشگر پیشنهادیمون رو نمی‌گیم، vi عه. در ادامه کار، با vi بیشتر آشنا میشین!

تنظیمات بعدی‌ای که باید انجام بدین و خیلی مهمه، نحوه درنظر گرفتن پایان خط توسط گیت عه. شاید بگی یعنی چی و چه فایده‌ای داره؟ در ادامه این مورد رو می‌خوایم بررسی کنیم با هم!

CRLF چیه و چرا انقدر معروفه؟

خب، CRLF به نوعی روش برای نشون دادن پایان خط تو فایل‌های متنی گفته می‌شه. این عبارت از دو حرف تشکیل شده:

CR (Carriage Return) : یه کاراکتر که می‌گه برو به ابتدای خط؛ همون “\r” معروفه!

LF (Line Feed) : یه کاراکتر که می‌گه برو به خط بعدی. اینو همه می‌شناسیم؛ “\n”!

اما داستان جایی جذاب می‌شه که سیستم‌عامل‌ها درباره این موضوع اختلاف دارن.

ویندوز از هر دو کاراکتر یعنی CRLF برای پایان خط استفاده می‌کنه. این یعنی وقتی خط تموم می‌شه، هم به ابتدای خط برمی‌گرده و هم می‌ره به خط بعد اما لینوکس و مک‌های جدید فقط از LF استفاده می‌کنن. این یعنی نیازی نمی‌بینن اول به ابتدای خط برگردن و مستقیم می‌رن به خط بعد.

اگه یه فایل متنی بین ویندوز و لینوکس رد و بدل کنی، ممکنه به مشکل بخوری. چون ویندوز می‌خواد CRLF ببینه، اما لینوکس فقط با LF کار می‌کنه. نتیجه؟ فایل ممکنه بهم بریزه یا وقتی تو ادیتور بازش می‌کنی، همه‌چی درهم و نامرتب بشه.

پس اگه روی ویندوز داری گیت رو تنظیم می‌کنی، باید autocrlf رو true قرار بدی:

❯ git config --global core.autocrlf true

اگه روی مک یا لینوکسی، باید autocrlf رو input قرار بدی:

❯ git config --global core.autocrlf input

اینجوری وقتی که دو نفر از دو سیستم عامل متفاوت میخوان روی یه پروژه کار کنن، به مشکلی برنمی‌خورن.

هر وقت که مطمئن نبودی باید چه چیزایی رو تنظیم کنی یا اینکه ببینی دیگه چه شخصی سازی هایی گیت داره، میتونی از دستور **"git –h config"** استفاده کنی.

خب بهت خسته نباشید می‌گم! تو گیت رو برای استفاده آماده کردی و می‌تونی اولین پروژه‌ات رو همین الان شروع کنی!

در ادامه قراره با پایه‌ای ترین دستورات گیت آشنا بشیم و کلی بهمون خوش بگذره!

اولین قدم: راه‌اندازی یه مخزن گیت

راه‌اندازی یه مخزن (Repository) تو گیت، مثل آماده کردن یه دفترچه یادداشت برای پروژه‌تونه. قراره هر چیزی که تغییر می‌دی، تو این مخزن ذخیره و پیگیری بشه. برای این کار، باید چند قدم ساده برداری:

اول از همه، یه فولدر برای پروژه‌ت بساز. این کار رو با دستور زیر انجام می‌دی:

❯ mkdir repo

اینجا repo اسم فولدرته. می‌تونی هر اسمی که دوست داری بذاری.

حالا با دستور زیر وارد فولدری که ساختی شو:

❯ cd repo

اینجاست که گیت وارد ماجرا می‌شه! دستور git init رو بزن:

❯ git init

که باید خروجی شبیه این ببینی:

Initialized empty Git repository in /tmp/repo/.git/

حالا شاید بپرسی خب چه اتفاقی افتاد؟ این دستور باعث می‌شه گیت تو همون فولدر شروع به کار کنه. گیت یه پوشه مخفی به اسم .**git** می‌سازه که همه چیز (تاریخچه تغییرات، تنظیمات و ...) اونجا ذخیره می‌شه. این فولدر رو میتونی با دستور زیر ببینی:

❯ ls -la

total 0

drwxr-xr-x  3 mkmaster mkmaster  60 Feb  8 14:35 **.**

drwxrwxrwt 15 root     root     380 Feb  8 14:35 ..

drwxr-xr-x  7 mkmaster mkmaster 200 Feb  8 14:35 **.git**

حالا فولدر پروژه‌ت یه "مخزن گیت" شده و آماده‌ست که تغییراتت رو دنبال کنه.

: .gitignoreدوست گیت برای نادیده گرفتن فایل‌ها

وقتی با گیت کار می‌کنی، ممکنه فایل‌هایی تو پروژه داشته باشی که نمی‌خوای گیت اونا رو دنبال کنه؛ مثل فایل‌های موقت، فایل‌های بزرگ، یا تنظیمات محلی. اینجاست که **.gitignore**وارد می‌شه! برای شروع فقط کافیه که .gitignore رو بسازیم. جلوتر راجب محتواش و چجوری پر کردنش حرف میزنیم:

❯ touch .gitignore

git status : وضعیت پروژه رو بررسی کن!

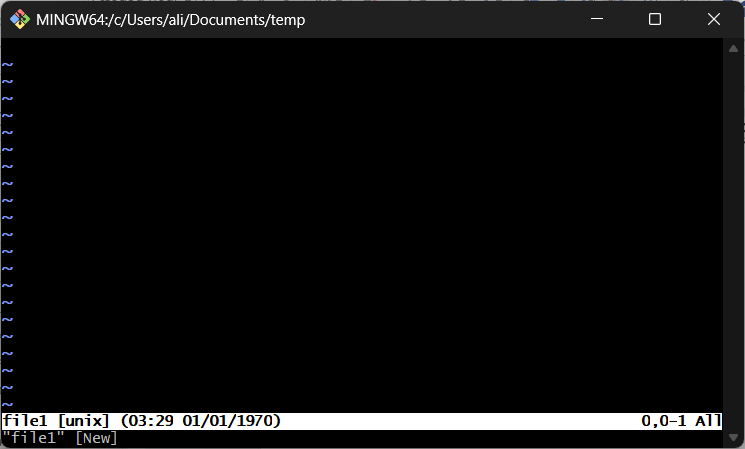
دستور git status بهت نشون می‌ده که وضعیت فعلی پروژه‌ت چطوره و چه تغییراتی آماده کامیت شدن هستن. این دستور به طور خاص به موارد زیر اشاره می‌کنه:

* فایل‌هایی که اضافه شدن یا تغییر کردن
* فایل‌هایی که هنوز به گیت اضافه نشده‌اند (untracked)
* فایل‌هایی که تغییرات‌شون ثبت شده ولی هنوز کامیت نشده‌اند
* فایل‌هایی که از مرحله آماده‌سازی (staging area)خارج شدن

خب برای اینکه بیشتر با این دستور آشنا بشیم، بیاید یه فایل متنی به فولدرمون اضافه کنیم. برای اینکار دستور زیر رو اجرا کنید:

❯ vi file1

وقتی که این دستور رو اجرا کردید احتمالا با صحنه زیر مواجه میشید:



نترسید! صفحه‌ای که میبینید فضای vi عه. یه ویرایشگر متن که قبل‌تر بهش اشاره کردیم. خب برای اینکه بتونید داخلش چیزی بنویسید باید ابتدا وارد حالت ویرایش بشید، برای اینکار کافیه که i رو روی کیبوردتون فشار بدید.



همونطور که بالا میبینید، vi در حالت INSERT قرار گرفته است. حالا کافیه که متنی که می‌خواید رو تایپ کنید. وقتی که کارِتون تموم شد، دکمه esc رو بزنید. حالا روی کیبورد ":wq" رو تایپ کنید:



حالا enter بزنید تا از محیط vi خارج بشید و تغییرات رو روی فایل اعمال کنید. با دستور cat می‌تونید محتوای فایل رو نگاه کنید:

❯ cat file1

خب تا اینجا ما یه فایل متنی به فولدر پروژه‌امون اضافه کردیم. حالا وقتشه که دستور git status رو اجرا کنیم:

❯ git status

که خروجی زیر رو باید بهتون بده:

On branch master

No commits yet

Untracked files:

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

       .gitignore

       file1

nothing added to commit but untracked files present (use "git add" to track)

همونطور که می‌بینی، گیت داره می‌گه فایل های file1 و .gitignore همگی Untracked ان. Untracked files به فایل‌هایی گفته می‌شه که توی پروژه‌ت وجود دارن، اما گیت هنوز اونا رو دنبال نمی‌کنه. این یعنی گیت هیچ نسخه‌ای از اون فایل‌ها رو ذخیره نکرده و تغییرات‌شون رو پیگیری نمی‌کنه. وقتی یه فایل جدید به پوشه پروژه اضافه کنی و هنوز با دستور git add به مرحله آماده‌سازی (staging area) نبرده باشی، اون فایل تو لیست Untracked Files نمایش داده می‌شه. چند لحظه دیگه دستور git add رو اجرا می‌کنیم اما قبلش بهتره به نسخه کوتاه شده git status (short status) نگاهی داشته باشیم:

برای اینکه یه خروجی خلاصه شده از git status داشته باشیم؛ کافیه که دستور رو با "-s" اجرا کنیم:

❯ git status -s

که خروجی ای به صورت زیر داره:

?? .gitignore

?? file1

"??" یعنی فایل‌ها untracked هستن.

خب حالا با دستور add می‌تونید گیت رو وادار کنید که فایل های untracked رو شروع به پیگیری کنه:

❯ git add .gitignore file1

حالا بیاید یه بار دیگه git status رو اجرا کنیم:

❯ git status

On branch master

No commits yet

Changes to be committed:

 (use "git rm --cached <file>..." to unstage)

       new file:   .gitignore

       new file:   file1

یا خروجی کوتاه شده اون:

❯ git status -s

A .gitignore

A file1

"A" یعنی فایل‌ها added شدن و آماده‌ان برای کامیت. جلو تر با مفهوم کامیت و اینکه چی هست بیشتر آشنا می‌شی ولی تا الان تو فایل هاتو وارد مرحله آماده‌سازی (staging area) کردی! خب این یعنی چی؟

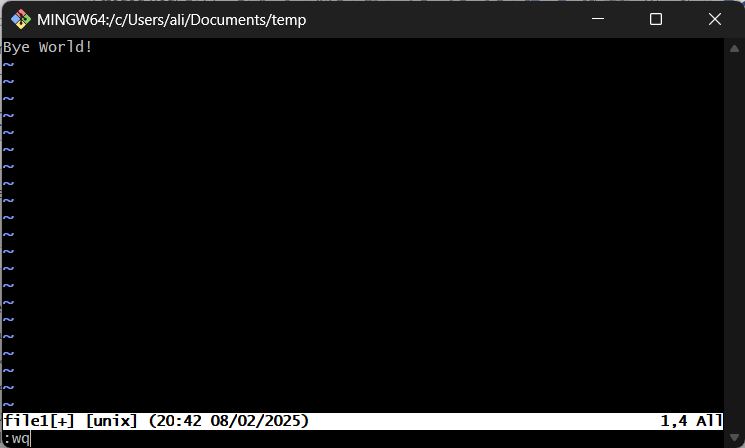
Staging Area چیه؟

Staging Area یا Indexیکی از مفاهیم اصلی در گیت هست که بهت اجازه می‌ده قبل از ذخیره‌ی تغییرات در تاریخچه پروژه (commit)، تغییرات رو مرتب و بررسی کنی. این فضا جاییه که فایل‌های انتخاب‌شده توسط کاربر برای کامیت بعدی قرار می‌گیرن.می‌تونی بهش به عنوان یه فضای موقت نگاه کنی که برات تغییرات فایل‌های پروژه رو که آماده کامیت شدن هستن رو نگه می‌داره. شاید بپرسی که کاربردش چیه؟

بهت این امکان رو می‌ده که مشخص کنی کدوم فایل‌ها یا تغییرات باید به کامیت بعدی اضافه بشن و کدوم‌ها اضافه نشن. خیلی وقتا تو نیاز پیدا می‌کنی که کدهایی که از وجودشون مطمئنی رو کامیت کنی و به بقیه کاری نداشته باشی!

خب حالا بیاید محتوای فایل file1 رو تغییر بدیم:

❯ vi file1



حالا که محتویات file1 رو تغییر دادی، بیاید ببینیم خروجی git status چی می‌تونه باشه:

❯ git status

On branch master

No commits yet

Changes to be committed:

 (use "git rm --cached <file>..." to unstage)

       new file:   .gitignore

       new file:   file1

Changes not staged for commit:

 (use "git add <file>..." to update what will be committed)

 (use "git restore <file>..." to discard changes in working directory)

       modified:   file1

فایل file1 تغییر کرده و گیت بهت می‌گه که این تغییرات هنوز کامیت نشده. برای ثبت تغییرات باید دوباره از git add استفاده کنی. اما قبلش بیاید یه چیز دیگه هم رو با هم امتحان کنیم.

بیاید یه فایل دیگه بسازیم، اونو add کنیم و بعدش فایل جدید رو حذف کنیم؛ به نظرتون چه چیزی توی خروجی git status امون قراره باشه؟

خب بیاید اینکارو انجام بدیم:

❯ touch file2

❯ git add file2

❯ rm file2

touch یه دستور ساده برای ساخت یه فایل خالی جدیده. حالا بیاید git status بگیریم:

❯ git status

On branch master

No commits yet

Changes to be committed:

 (use "git rm --cached <file>..." to unstage)

       new file:   .gitignore

       new file:   file1

       new file:   file2

Changes not staged for commit:

 (use "git add/rm <file>..." to update what will be committed)

 (use "git restore <file>..." to discard changes in working directory)

       modified:   file1

       deleted:    file2

deleted نشون می‌ده که فایل file2 حذف شده، ولی هنوز این تغییرات به گیت اعلام نشده. خب برای اینکه این تغییرات رو وارد مرحله آماده‌سازی کنیم و به گیت بگیم که دوباره پیگیریشون کنه باید همچنان از دستور add استفاده کنیم؛ شاید باورتون نشه ولی برای اعلام تغییرات حذف هم باید از دستور git add استفاده کنیم!

❯ git add file2 file1

❯ git status

On branch master

No commits yet

Changes to be committed:

 (use "git rm --cached <file>..." to unstage)

       new file:   .gitignore

       new file:   file1

حالا همچی آماده‌اس که اولین کامیتتون رو ثبت کنید!

git commit : تغییراتت رو ثبت کن !

حالا که Staging Area رو آماده کردید، وقتشه که تغییراتتون رو Commit کنید!

یه کامیت توی گیت مثل یه عکس Snapshot) ) از پروژه‌ات توی یه لحظه خاصه . وقتی یه کامیت میزنی، گیت وضعیت فعلی پروژه رو (یعنی همه فایل‌های ردیابی‌شده‌ی پروژه) ذخیره می‌کنه و یه شناسه منحصربه‌فرد بهش اختصاص میده که بهش **Commit Hash** میگن. این شناسه یه مقدار SHA-1 هستش که به صورت ساده معمولاً یه رشته ۴۰ کاراکتری از اعداد و حروفه که گیت به صورت خودکار تولیدش می‌کنه که باعث می‌شه هر کامیت یکتا و تغییرناپذیر باشه.

در خیلی از موارد می‌تونید از بخشی از این شناسه برای ارجاع دادن به کامیت استفاده کنید و مثلاً اولین 7 کاراکتر از شناسه کامیت معمولاً کافیه.

هر کامیت شامل چیه؟

* کدها و تغییرات ثبت‌شده) فایل‌هایی که توی Staging Area بودن(
* نام و ایمیل نویسنده کامیت
* تاریخ و زمان کامیت
* پیام توضیحی درباره تغییرات
* شناسه منحصر‌به‌فرد (Commit Hash)

چجوری کامیت کنیم ؟

برای اینکه کامیت انجام بدید مثل عکس پایین عبارت git commit -m رو بنویسید و بعدش داخل **""**

پیام مربوط به کامیت خودتون رو که یک توضیحی از تغییراتی هست که انجام دادید بنویسید.

❯ git commit -m "First commit"

❌❌❌❌️ از اینجا به بعد امکان داره خروجی که دریافت می کنید با خروجی که داخل داک هستش فرق بکنه . حالا ممکنه براتون سوال شده باشه که کدوم قسمتش ممکنه تفاوت داشته باشه ؟

**شناسه ی کامیت یا کامیت هش** ! چون این هش به صورت تصادفی و یکتا هستش برای هر کامیت

حالا بعد از انجام دستور کامیت با خروجی ای به صورت زیر رو به رو میشید:

[master (root-commit) 1b9ab90] First commit

2 files changed, 2 insertions(+)

create mode 100644 .gitignore

create mode 100644 file1

تو خط اول خروجی ۷ کاراکتر اول **شناسه ی کامیت** که در اینجا 1b9ab90 هست رو می بینید و بعدش هم پیام کامیت یا همون ( Commit message ) که خودتون نوشتینش

و تو خط های بعدی هم می بینید که ۲ تا فایل قبلی که تو Staging Area اضافه کرده بودید الان دیگه کامیت شدن . حواستون باید به ۲ مورد باشه :

1. فقط فایل‌هایی که به Staging Area با (git add) اضافه کردید توی کامیت ثبت می‌شن!
2. اگه فایلی رو تغییر داده باشید ولی git add نزده باشید، اون توی کامیت نمیاد و همچنان توی وضعیت "تغییرکرده" باقی می‌مونه.

حالا می خوایم یه فایل خالی جدید درست کنیم . این کار رو با دستور touch انجام میدیم و بعد از دستور touch اسم فایل که اینجا **temp.log** هست رو می نویسیم

❯ touch temp.log

بعدش از دستور **git status** برای فهمیدن وضعیت فایل هامون استفاده می کنیم :

❯ git status

و بعدش با خروجی زیر رو به رو میشید :

On branch master

Untracked files:

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

       temp.log

nothing added to commit but untracked files present (use "git add" to track)

پس این فایل جدیدی که درست کردیم هنوز توسط گیت پیگیری نمیشه یا به اصطلاح **untracked** هست پس باید اون رو به Staging Area ببریم . این کار رو همون طور که یاد گرفتید با git add انجام میدیم ولی این دفعه از دستور ی با یه کم تغییر استفاده می کنیم :

❯ git add .

شاید براتون سوال بشه که این دستور چه فرقی می کنه و چه زمانی ازش استفاده می کنیم ؟

دستور git add . همه ی فایل‌های تغییر‌ یافته (مثل فایل‌های جدید، تغییر‌یافته و حذف‌شده) رو چه در مسیر فعلی و چه در زیرشاخه‌هاش باشن به منطقه استیج (Staging Area) اضافه می‌کنه

حالا برای اینکه بفهمیم وضعیت فایل هامون چطوره از **-s** **git status** استفاده می کنیم :

❯ git status -s

A  temp.log

اوپس ! ما اینجا یه اشتباهی انجام دادیم . شاید براتون جای سوال باشه که چه اشتباهی ؟

عجله نکنید ! قبلش بیاید دربار ه ی فایل های **.log** یذره صحبت کنیم که اصلا چی هستن ؟

فایل‌های**.log** معمولا فایل‌های لاگ (گزارش) هستن که برنامه‌ها و سرویس‌ها برای ذخیره اطلاعات اجرایی خودشون ایجاد می‌کنن. این اطلاعات می‌تونه شامل خطاها، پیام‌های دیباگ، درخواست‌های کاربران و... باشه. این فایل‌ها معمولا نیازی به ورژن کنترل ندارن چون خود گیت اون‌ها رو ذخیره نمی‌کنه و هر بار که پروژه اجرا بشه، این فایل‌ها ممکنه تغییر کنن یا حذف بشن. به همین خاطر، بهتره اون‌ها رو داخل **.gitignore** بذاریم تا به اشتباه کامیت نشن.

**.gitignore** یه فایل متنی توی ریپازیتوری گیت هست که مشخص می‌کنه کدوم فایل‌ها نباید توسط گیت رهگیری (track) بشن. این یعنی وقتی git add رو می‌زنید، گیت اون فایل‌ها رو در نظر نمی‌گیره و داخل ریپازیتوری ذخیره نمی‌کنه. فایل‌های .**gitignore** فقط روی فایل‌های جدید تأثیر دارن یعنی اگه قبلا یه فایل رو استیج کرده باشید، حتی اگه بعدا توی .gitignore بذاریدش، همچنان توی گیت خواهد بود! پس برای حذفش از گیت ولی نگه داشتنش توی سیستم خودتون، دستور زیر رو بزنید:

❯ git rm --cached temp.log

تا خروجی زیر رو ببینید :

rm 'temp.log'

حالا **git status** رو بزنید تا وضعیت فایل ها رو بررسی کنیم

❯ git status -s

?? temp.log

با توجه به **??** متوجه میشیم که فایلمون دیگه از محدوده ی بررسی گیت خارج شده.

حالا برای اینکه دیگه این فایل های **.log** توسط گیت بررسی نشن باید اسم اون ها رو تو فایل **.gitignore** بنویسیم

پس با دستور **vi .gitignore** میریم برای ادیت کردن فایل **.gitignore** ای که اول داک ساخته بودیم

❯ vi .gitignore

بعد از زدن این دستور تو خط اول **\*.log** رو می نویسیم تا همه ی فایل هایی که پسوند .log دارن دیگه توسط git بررسی نشن !

\*.log

~

~

~

:wq

حالا به جای جالب ماجرا میرسیم ! اگر **git status** رو بزنید تا وضعیت فایل ها رو ببینید خروجی زیر رو می بینید :

❯ git status

On branch master

Changes not staged for commit:

 (use "git add <file>..." to update what will be committed)

 (use "git restore <file>..." to discard changes in working directory)

       modified:   .gitignore

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

دیگه خبری از فایل های **.log** نیست و فقط فایل **.gitignore** ای که ادیت کردیم رو داره بهمون نشون میده ! پس حالا برای استیج شدن **git add .** رو می زنیم و بعد از اون هم **git status** رو می زنیم تا وضعیت فایل هامون رو ببینیم

❯ git add .

❯ git status

بعد از اون هم خروجی زیر رو می بینید :

On branch master

Changes to be committed:

 (use "git restore --staged <file>..." to unstage)

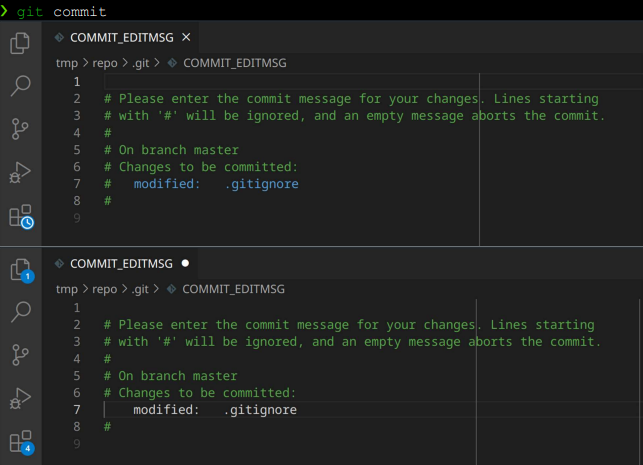
       modified:   .gitignore

حالا باید فایل stage شده رو کامیت کنیم . ما تا الان هر موقع **git commit** رو استفاده کردیم بعدش از -m استفاده کردیم تا پیام مربوط به کامیتمون رو بنویسیم ! حالا اگر این کار رو نکنیم چه اتفاقی میفته ؟

اگه فقط git commit بزنم چی میشه؟ 🤔

اگه -m "پیام کامیت" رو نزنید، گیت خودش یه ویرایشگر متنی باز می‌کنه تا پیام کامیت رو اونجا بنویسید. با توجه به اینکه اول داک , ویرایشگر پیش فرض خودتون رو به vs code تغییر دادید یک پنجره تو vs code براتون باز میشه و توش می تونید پیام کامیت خودتون رو بنویسید و بعدش

Ctrl + S رو بزنید تا Save بشه و بعدش vs code رو ببندید



بعدش با خروجی پایین رو به رو میشید که نشون میده Commit به درستی انجام شده

[master 182c1f6]        modified:   .gitignore

1 file changed, 1 insertion(+), 1 deletion(-)

حالا چرا کامیت مهمه؟

1. **ردیابی تغییرات:** می‌تونی بفهمید چه چیزی تغییر کرده، چه کسی تغییرش داده و چه زمانی.
2. **همکاری تیمی:** چندین نفر می‌تونن همزمان روی پروژه کار کنن، بدون اینکه کار همدیگه رو خراب کنن.
3. **بازگشت به نسخه‌های قبلی:** اگه یه مشکلی پیش بیاد، می‌تونید خیلی راحت به یه نسخه قدیمی‌تر برگردی.

تو قسمت بعدی یاد می گیریم که چجوری **خوب** کامیت کنیم تا هم خودمون و هم افراد تیممون بفهمن هر تغییر چه تأثیری روی پروژه گذاشته و چه چیزی تغییر کرده.

بهترین روش‌های کامیت زدن

* **تغییرات مرتبط رو توی یک کامیت بزار**

یه کامیت باید فقط یه مجموعه تغییرات مرتبط رو شامل بشه. مثلاً، اگه دو تا باگ جداگونه رو درست می‌کنید، بهتره برای هر کدوم یه کامیت جدا بزنید. این کار باعث می‌شه هم تیمی‌هاتون راحت‌تر تغییرات رو دنبال کنن و اگه لازم شد، یه تغییر رو بدون مشکل برگردونن.

* زود‌به‌زود کامیت کن!

اگه مرتب کامیت بزنید، هر کامیت کوچیک و تمیز می‌مونه و فقط تغییرات مرتبط رو شامل می‌شه. این کار کمک می‌کنه که راحت‌تر تغییراتتون رو با بقیه به اشتراک بذارید. در عوض، اگه دیر‌به‌دیر

و یه کامیت خیلی بزرگ بزنید دنبال کردن تغییرات سخت می‌شه.

* قبل از کامیت، کدتو تست کن!

هیچ‌وقت چیزی رو کامیت نکنید که فکر می‌کنید درسته! قبل از اینکه کامیت بزنید، کدتون رو خوب تست کنید که مطمئن بشید همه‌چی اونطوری که باید کار می‌کنه و مشکلی ایجاد نمی‌کنه. این قضیه مخصوصاً وقتی که کدتو با بقیه به اشتراک می‌ذاری خیلی مهم‌تر می‌شه.

* پیام کامیت درست و حسابی بنویس!

1. **پیام کامیت باید خلاصه و واضح باشه:** همیشه یک توضیح کوتاه از تغییرات بدید. اگر نیاز بود، می‌تونید توضیحات بیشتری در قسمت بدنه پیام بنویسید.
2. **از زمان حال استفاده کنید**: پیام‌های کامیت باید به صورت زمان حال نوشته بشن. مثلاً به جای "اضافه کرد" از "اضافه می‌کنم" استفاده کنید.
3. **پیام باید دقیق و مشخص باشه:** پیام باید به طور خاص توضیح بده که دقیقاً چه چیزی تغییر کرده و چرا این تغییرات ضروری بوده.
4. **از پیام‌های توضیحی برای تغییرات بزرگ استفاده کنید:** اگر تغییرات پیچیده‌ای انجام دادید یا نیاز به توضیحات بیشتری بود، می‌تونید از ویرایشگر متنی برای نوشتن پیام استفاده کنید. برای این کار از دستور git commit بدون -m استفاده کنید تا وارد ویرایشگر بشید و پیام طولانی‌تری بنویسید.

چرا نوشتن پیام کامیت خوب ضروریه؟

یه پیام کامیت خوب باعث می‌شه که تاریخچه‌ی پروژه قابل فهم و قابل مدیریت باشه. وقتی تیمی روی یه پروژه کار می‌کنید، یا حتی خودتون بعد از چند هفته سراغ کد برید، یه پیام کامیت واضح و درست حسابی می‌تونه نجاتتون بده و Debugging رو راحت تر می کنه .

git log : دیدن تاریخچه کامیت ها و برگشت به گذشته !

اگه توی پروژه‌اتون کلی کامیت زدید و می‌خواید ببینید چه تغییراتی انجام دادید یا اینکه چه کسی کامیت‌ها رو زده، و چه زمانی این اتفاق افتاده، دستور **git log** به دردتون می‌خوره. این دستور تاریخچه کامیت‌های پروژه رو به ترتیب معکوس زمانی نمایش می‌ده یعنی جدیدترین کامیت اول نشون داده میشه و این دستور کمک می‌کنه تغییرات رو ردیابی کنید.

چجوری از Git Log استفاده کنیم ؟

برای استفاده از قابلیت های گفته شده , اول توی ترمینال عبارت **git log** رو بنویسید :

❯ git log

و بعدش با خروجی پایین روبه رو میشید :

بیاید با هم ببینیم این اطلاعات چه چیزی رو نشون میدن !

commit 182c1f6fd71206127cd02f62f935a51139b36e26 (**HEAD** -> **master**)

Author: Jesper <molioo1298@gmail.com>

Date:   Sat Feb 8 14:52:03 2025 +0330

           modified:   .gitignore

commit 1b9ab90c82d0301b222998520edc4fa42734f526

Author: Jesper <molioo1298@gmail.com>

Date:   Sat Feb 8 14:44:22 2025 +0330

   First commit

(q)

1. اولین چیزی که می بینید یه رشته ۴۰ کاراکتری نارنجی رنگ هستش که همون شناسه کامیت

یا همون ( Commit Hash ) هست که هر کامیت رو متمایز می کنه

1. تو خط بعدی نویسنده یا (Author ) هر کامیت رو نشون میده که در واقع کسی هست که این تغییرات رو ثبت کرده
2. تو خط بعدی زمانی که کامیت انجام شده رو نشون میده
3. و در نهایت تو خط آخر توضیحی که موقع کامیت نوشتیم یا همون پیام کامیت

( Commit message ) رو نشون میده

شخصی سازی خروجی Git Log

اگر خروجی git log براتون زیادی شلوغ یا خوانا نیست، می‌تونید با یه سری گزینه (flags) شخصی سازیشون کنید

1. برای اینکه یه نمایش خلاصه شده ببنید عبارت--oneline رو بعد از git log بنویسید و خروجی مثل پایین می بینید :

❯ git log --oneline

182c1f6 (**HEAD** -> **master**)        modified:   .gitignore

1b9ab90 First commit

تو این حالت فقط ۷ کاراکتر اول Commit hash رو بهمون نمایش میده و مقابلش هم پیام مربوط به اون کامیت رو نشونمون میده

1. اگر فقط چند تا از آخرین کامیت‌ها رو می‌خواید ببینید، دستور پایین رو بزنید :



1.  اگر بخواید فقط کامیت های یه نفر خاص رو ببینید می تونید از دستور پایین استفاده کنید :

حالا می خوایم چند تا تغییرات انجام بدیم ولی قبلش با زدن دستور ls می تونید لیست فایل های موجود داخل پوشه اتون رو ببنید

❯ ls

file1  temp.log

پس الان ۲ تا فایل به نام های "file1" و "temp.log" داخل پوشه اتون موجوده

حالا با زدن دستور rm می تونید یه فایل رو پاک کنید که اینجا ما file1 رو پاک می کنیم

❯ rm file1

بعد از اون برای اینکه git بفهمه که ما این فایل رو حذف کردیم باید اون رو به Staging area اضافه کنیم که این کار رو همون طور که یاد گرفتید با git add . انجام میدیم :

❯ git add .

بعد از اون طبق معمول باید تغییراتمون رو commit کنیم :

❯ git commit -m "Second commit"

[master 9f1db9f] Second commit

1 file changed, 1 deletion(-)

همون طور که می بینید کامیت جدید با Commit hash ۷ رقمی که به صورت 9f1db9f هست ثبت شده . الان با زدن دستور git log --onelineمی تونید لیست کامیت ها رو ببینید :

❯ git log --oneline

9f1db9f (**HEAD** -> **master**) Second commit

182c1f6         modified:   .gitignore

1b9ab90 First commit

اگر دقت کنید می بینید که کامیت جدیدی که با هش 9f1db9f زدیم بالاتر از همه در لیست کامیت ها قرار گرفته .

الان دوباره دستور ls رو بزنید تا لیست فایل ها رو بتونید ببینید :

همون طور که می بینید file1 از لیست فایل ها حذف شده

❯ ls

temp.log

git revert : ماشین زمان گیت !

خب، فرض کنید که یه کامیت اشتباه انجام دادید و می‌خواید اون تغییرات رو برگردونید، ولی نمی‌خواید تاریخچه‌ی گیت رو خراب کنید. اینجاست که git revert به دادتون می‌رسه!

git revert چیکار می‌کنه؟

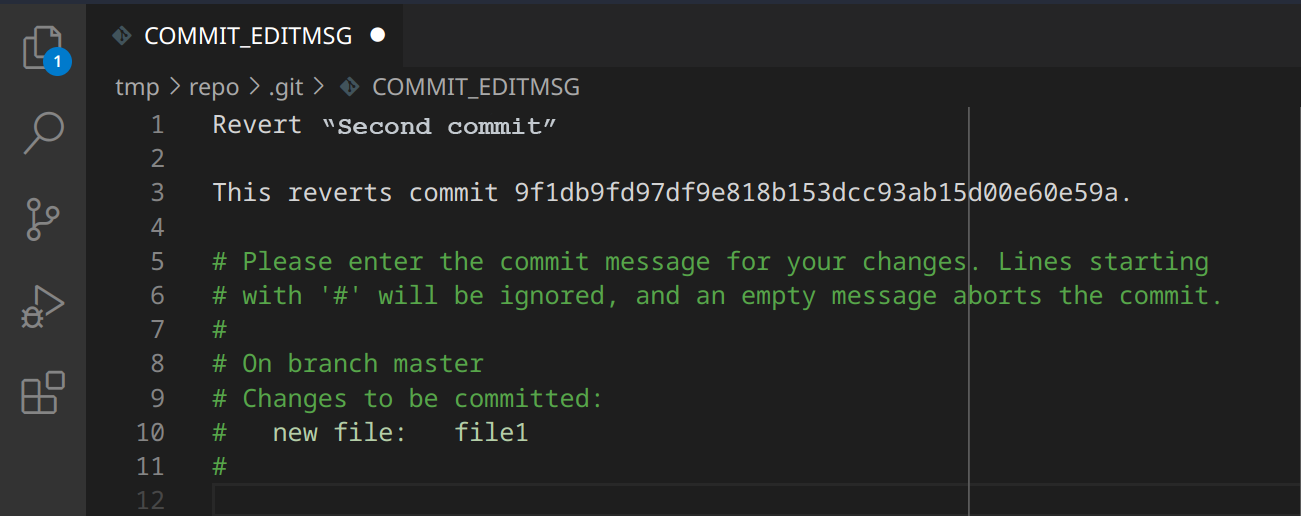
**git revert** در واقع یه کامیت جدید می‌سازه که تغییرات یه کامیت خاص رو برعکس می‌کنه! یعنی اگه یه فایل رو توی یه کامیت حذف کرده باشید، git revert یه کامیت جدید می‌سازه که اون فایل رو برمی‌گردونه.

حواستون باشه! git revert خود کامیت رو پاک نمی‌کنه، فقط تغییراتش رو خنثی می‌کنه.

چجوری از Git Revert استفاده کنیم ؟

برای استفاده از این دستور اول باید شناسه اون کامیت ای که می خواید تغییراتش رو خنثی کنید پیدا کنید . برای این کار می تونید از git log استفاده کنید. دقت کنید لازم نیست همه ی ۴۰ کاراکتر رو کپی کنید و می تونید از همون ۷ کاراکتر اول استفاده کنید . بعد از اینکه ۷ کاراکتر اول شناسه اون کامیت رو پیدا کردید دستور git revert رو بنویسید و بعد اش اون شناسه کامیت رو مثل کامند پایین بنویسید .

❯ git revert 9f1db9f

بعد از زدن این دستور، گیت یه ویرایشگر باز می‌کنه و ازتون می‌خواد که پیام کامیت جدید رو بنویسید.

بعد از اون Ctrl + S رو بزنید و vs code رو ببندید تا این کامیت ذخیره بشه . حالا گیت یه کامیت جدید ایجاد می‌کنه که تغییرات اون کامیت مشخص شده رو برعکس می‌کنه. بعد از اون با خروجی پایین رو به رو میشید که نشون میده این کامیت جدید ثبت شده و کامیت هشش به صورت **4410d50** هست.

[master 4410d50] Revert "Second commit"

1 file changed, 1 insertion(+)

create mode 100644 file1

حالا با زدن git log می تونید هیستوری کامیت هاتون رو ببینید :

❯ git log

commit 4410d50cdc74756a9848cba62fb4ebea7bca51bf (**HEAD** -> **master**)

Author: Jesper <molioo1298@gmail.com>

Date: Sat Feb 8 14:54:49 2025 +0330

Revert "Second commit"

This reverts commit 9f1db9fd97df9e818b153dcc93ab15d00e60e59a.

commit 9f1db9fd97df9e818b153dcc93ab15d00e60e59a

Author: Jesper <molioo1298@gmail.com>

Date:   Sat Feb 8 14:54:43 2025 +0330

   Second commit

commit 182c1f6fd71206127cd02f62f935a51139b36e26

Author: Jesper <molioo1298@gmail.com>

Date:   Sat Feb 8 14:52:03 2025 +0330

           modified:   .gitignore

commit 1b9ab90c82d0301b222998520edc4fa42734f526

Author: Jesper <molioo1298@gmail.com>

Date:   Sat Feb 8 14:44:22 2025 +0330

   First commit

همون طور که می بینید کامیت اول همون کامیت revert هستش که تازه انجامش دادیم .

حالا یبار دیگه دستور ls رو اجرا کنید:

❯ ls

file1  temp.log

همونطور که می‌بینید فایلی که پاک کرده بودیم بازگردونی شد!

ویدیو برای یادگیری بهتر

اگر می خواید بهتر مفاهیم و کامند ها رو یاد بگیرید می تونید ویدیو های جادی تو فرادرس رو هم ببینید !

[Git - GitHub - GitLab - Faradars](https://faradars.org/courses/fvgit9609-git-github-gitlab)