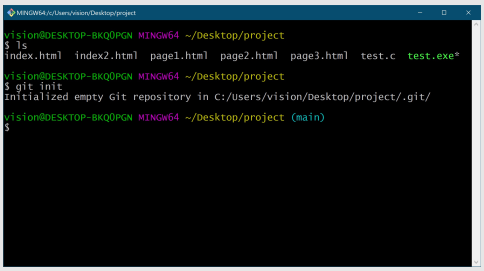
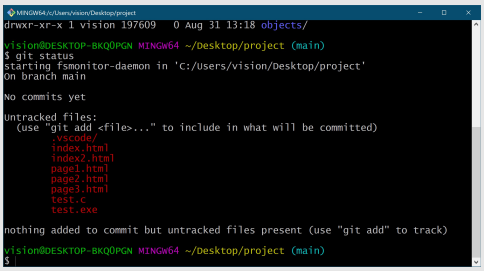
نکته ۱

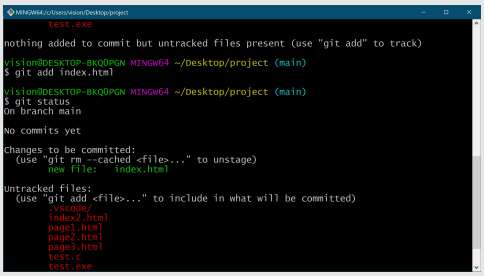
اولین مرحله نصب Git از طریق لینک می‌باشد. گیت یک سیستم کنترل نسخه توزیع شده است. در واقع Git یک برنامه‌ای است که در هر بار اجرا به فایل‌های پیکربندی شده با خودش می‌اندازد و با توجه به تغییر و تحول در آن‌ها نسبت به آن تصمیم می‌گیرد. برای پیکربندی یک دایرکتوری توسط Git ابتدا باید دستور `git init` اجرا شود. این دستور محل دایرکتوری مورد نظر را برای کنترل نسخه توسط Git آماده می‌کند. با اجرای این دستور پوشه‌ای به نام `.git` در همان دایرکتوری ساخته می‌شود. برای بررسی شرایط دایرکتور پس از پیکر بندی، دستور `git status` استفاده می‌شود.



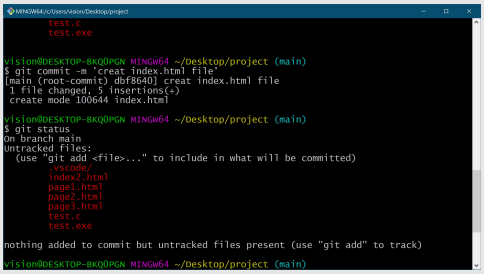
فایل‌های موجود در دایرکتوری می‌توانند تغییر کنند و در این هنگام این فایل‌ها را می‌توان به مرحله Stage[1] برد. یعنی فایل‌ها آماده `commit` کردن می‌شوند. اگر بخواهیم در هر بار اضافه کردن فایل و تغییر و تحول دایرکتوری و فایل‌های مربوط به آن `git` این اتفاقات را پیگردی کند باید پس از هر تغییر و تحول با دستور `git status` تمامی تغییر و تحولات را به `stage` بیاوریم.



با دستور `git add # filename` فایل مورد را نظر به `git` می‌شناسانیم و از این پس تمام تغییر و تحولات آن را پیگردی خواهد کرد.



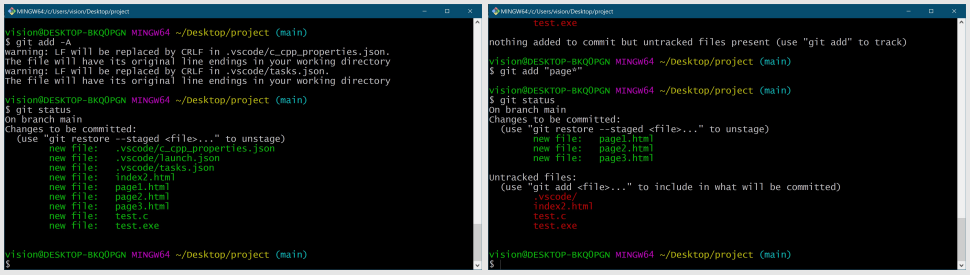
سپس با دستور `git commit -m 'message` کامیت می‌کنیم.



[1] صحنه

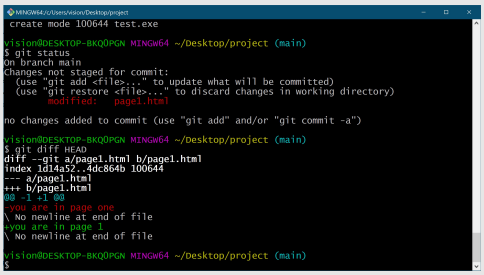
نکته ۲

دستوری به صورت `git log` تمام پیشینه کامیت‌های ما را نشان می‌دهد. حال اگر تعداد فایل‌ها زیاد باشد می توان از دستور `git add -A` برای ارسال تمام فایل‌ها به استیج استفاده کرد و یا با دستور `git add "page*"` صرفا فایل‌هایی را که با اسم page شروع می‌شوند را به استیج ببریم.

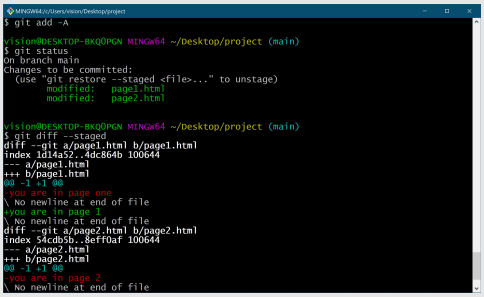


نکته ۳

اگر یکی از فایل‌های موجود در دایرکتوری را تغییر دهیم می‌توانیم ببینیم که `git status` به این تغییر پی برده است. با استفاده از دستور `git diff HEAD` تغییرات آخر کامیت را نشان خواهد داد. `HEAD` به هد آخرین کامیت ما اشاره دارد.



اگر بعد از تغییرات فایل‌ها و انتقال آن ها به محیط استیج نسبت به تغییرات اعمال کرده‌امان در فایل‌ها مشکوک شویم می‌توانیم با دستور `git diff --staged` تمامی این تغییرات را ببینیم.

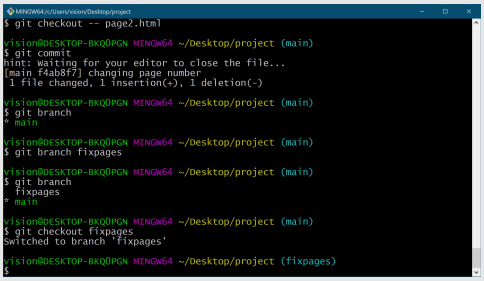


همچنین می‌توان برای نمایش خلاصه تغییرات در فایل‌هایمان از دستور `git diff --stat` استفاده کرد. برای خارج کردن فایلی از استیج می‌توان دستور `git reset #filename` و یا `git reset HEAD #filename` را اجرا کرد. اگر فایلی جدیداً ساخته شده باشد و به اشتباه علی‌رغم خواسته ما وارد استیج شده باشد می‌توان از دستور `git rm --cached #filename` برای خارج کردن **فایل جدید ساخته شده** به جای دستور `git reset` استفاده کرد. اگر بخواهیم فایلی را که خراب کرده‌ایم و می‌خواستیم آن را به ورژن قبلی‌اش برگردانیم می‌توانیم از دستورهای `git chechkout -- #filename` و یا `git checkout #filename` استفاده کنیم. در واقع این دستور تمامی تغییرات اجرا شده در فایل را به حالت قبل از تغییرات بر می‌گرداند.

نکته ۴

دستور `git branch` شاخه‌های پروژه را نشان می‌دهد. برای اضافه کردن شاخه‌ای به پروژه باید دستور `git branch # branch name` را به کار برد. در واقع این شاخه ساخته شده از شاخه `master` یا همان `main` در ورژن‌های جدید ساخته شده است. برای تغییر شاخه باید دستور

`git checkout # branch name` را به کار برد.



در نهایت برای تلفیق شاخه‌های جانبی با شاخه اصلی باید دستور `git merge # branch name` را استفاده کرد.

نکته ۵

با دستور `git rm # file name` می‌توان یک فایل را هم از دایرگتوری و هم git حذف کرد. با اسفاده از دستور `git branch -d #` `branch name` می‌توان شاخه ایجاد شده را پاک کرد.

نکته ۶

با استفاده از دستور `git clone # project web address` می‌توان پروژه‌های هاست شده در github ، gitlab را به عنوان پروژه خودمان بر روی هارد درایومان ذخیره کنیم. با استفاده از دستور `git push origin master` می‌توان پروژه انجام شده را به مخزن کد آنلاین github فرستاد و تغییرات را نیز بر روی نسخه origin انجام داد. همچنین می‌توان با استفاده از دستور `git pull origin master` تغییرات صورت گرفته بر روی پروژه را به صورت محلی بر روی دایرکتوری خودمان نیز داشته باشیم.

نکته ۷

می‌توان یک ریموت برای پروژه در نظر گرفت. دستور `git remote add # remote name(or origin) # repo address` این عمل را انجام می‌دهد.

نکته ۸

برای نمایش کامیت‌های مختلف صورت گرفته با جزییات می‌توان دستور `git show # commit id` را به کار برد. گاهی پس از دولوپ کردن پروژه‌ای نیاز به ایجاد تگ برای پروژه است به این صورت که برای توسعه‌دهنده نیازی به به خاطر سپردن شاخه‌های پروژه نداشته باشد. دستور `git tag` تگ‌های ایجاد شده را نمایش می دهد. برای ساختن یک تگ باید `git tag -a #version number -m 'message'` را می‌توان به کار برد. حتی می‌توان تگ‌هایی با ورژن‌های قدیمی را نیز برای یک کامیت مشخص اضافه کرد که به این منظور دستور `git tag -a #version number #commit id -m 'message'` به کار می‌رود. تگ‌ها به صورت عادی پوش نمی‌شوند لذا برای این امر می‌توان دستورهای `git push origin #tag name` و `git push origin --tags` استفاده کرد.

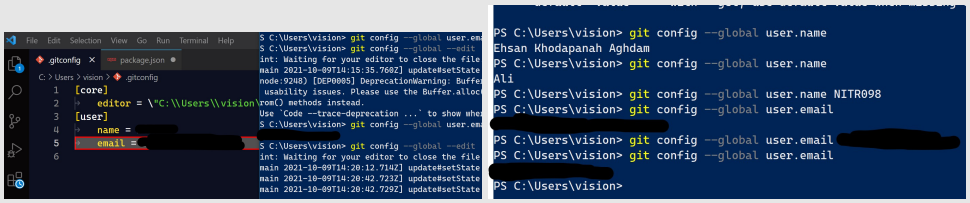
نکته ۹

در گیت می‌توان با دستور `git help #action name` به راهنمای هر کاری که در git بخواهیم انجام دهیم دسترسی داشته باشیم. برای پیدا کردن تغییر دهنده هر خط کد در یک پروژه در هنگام دیباگینگ می‌توان از دستور `git blame -L #line number` استفاده کرد.

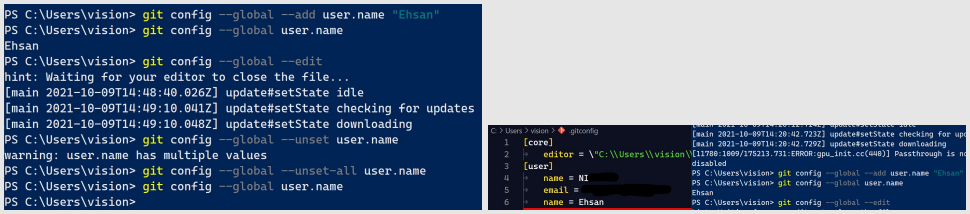
نکته ۱۰

یکی از مراحل مهم بعد از نصب git کانفیگ کردن آن است. هدف از این کار این است که اگر پروژه‌ای را به صورت ریموت دریافت کرده و تغییراتی در آن دادیم این تغییرات با نام و یوزر و ایمیل و حتی امضای دیجیتال(در صورت موجود بودن) شناخته شود. با دستور `git config` می‌توان به گزینه‌های قابل کافیگ گیت دسترسی پیدا کرد. در گیت ۳ نوع گیت کانفیگ لوکیشن[۲] با نام‌های global، system و local می‌باشد. اگر بخواهیم تمام اطلاعات گیت برای پروژه‌های مختلف روی سیستم شخصی‌مان یکی باشد حالت اول و اگر تعداد مختلفی یوزر بر روی کامپیوتر باشد و بخواهیم اطلاعاتشان یکی باشد حالت دوم و اگر بخواهیم کانفیگ گیت برای هر پروژه متفاوت باشد، گزینه سوم را باید پیکربندی کنیم. دستورات `git config --global user.name "your name"` و `git config --global user.email "your email"` به همین منظور استفاده می‌شوند. اگر بخواهیم تغییرات را در ادیتور از پیش تعیین شده توسط ما برای گیت انجام دهیم باید دستور `git config`

`--edit --global` را اجرا و تغییرات را در ادیتور اعمال و آن فایل را که در پوشه `gitconfig`. پروژه است، پیکربندی می‌کنیم.



گزینه‌های اضافی دیگری نیز در گیت کانفیگ مثلا آپشن `--add` در دستور `git config --global --add user.name "another name"` را می‌توان برای اضافه کردن نام دیگری نیز به عنوان یوزرها در نظر گرفت اما عملا گیت صرفا آخرین نام اضافه شده به یوزرها را برای خود انتخاب می‌کند. حال برای پاک کردن نام‌ها می‌توان از آپشن `--unset` استفاده کرد اما این کار با خطا مواجه می‌شود زیرا این آپشن صرفا می‌تواند یک متغیر (در این جا همان نام اضافی در یوزر) را پاک کند به همین منظور از دستور `git config --global --unset-all user.name` برای پاک کردن تمام یوزرها استفاده می‌شود البته روش دیگر باز کردن فیل کانفیگ در ادیتور و تغییر یوزر هم می‌توانست باشد.



برای نمایش تمامی پارمترهای `gitconfig` می‌توان از دستور `git config --global --list` استفاده کرد.
نکته مهم محل ذخیره‌سازی فایل‌های گیت کانفیگ است که در حالت global این فایل در پوشه یوزر کامپیوتر و در حالت system در پوشه `etc` واقع در محل نصب گیت واقع شده است. در حالت local نیز این فایل در پوشه پنهان[۳] `.git`. در محل پروژه پیگردی شده با گیت است.

نکته ۱۱
دایرکتوری که توسط گیت پیگردی می‌شود شامل پوشه پنهان `.git`. است که این پوشه محتویاتی دارد که در ادامه تعدادی از آن‌ها را می‌گوییم.

فایل `HEAD` یک اشاره‌گر به آخرین قسمتی که در پروژه قرار داریم، است. در فایل `gitconfig` اطلاعات کانفیگ محلی پروژه ثبت شده است. در پوشه `refs` تگ‌های پروژه و HEAD pointers به ترتیب در زیرپوشه‌های `tags` و `heads` در این محل قرار می‌گیرند. همچنین اگر پروژه شامل شاخه‌های دیگری باشد نیز در زیرپوشه `heads` قرار می‌گیرد.
برای ریست کردن تغییرات و رفتن به حالت خاصی در یک کامیت می‌توان از دستور `git reset --soft #commit id` استفاده کرد. این دستور صرفاً تمامی تغییرات اعمال شده و قرار گرفته در repository را به تغییرات همان `commit id` مذکور می‌برد. یعنی حتی اگر فایلی هم تولید شده باشد آن فایل‌ها در دایرکتوری موجود خواهند بود اما تغییرات بعد از این کامیت را به حالت استیج خواهد برد. دستور دیگر `git reset --mixed #commit id` همانند دستور قبل است با این تفاوت که تغییرات بعد از کامیت اعمالی را از استیج نیز خارج می‌کند. دستور `git reset --hard #commit id` همانند دستورات قبل است اما تمامی فایل‌های بعد از `commit id` را به طور کل از دایرکتوری پاک می‌کند.

نکته ۱۲
یکی از ویژگی‌های دستور `checkout` برای ایجاد شاخه‌های مجازی است. مثلاً فرض کنید ۳ کامیت داریم و با دستور `git checkout #2nd commit id` اشاره‌گر HEAD را به ابتدای کامیت دوم می‌رود و تمام تغییرات بعد از آن و فایل‌های ایجاد شده دیگر در دسترس این اشاره‌گر و یوزر نیست. در این حالت گیت برای کامیت‌های بعد از کامیت دوم یک شاخه مجازی در نظر می‌گیرد. برای برگشتن به حالت عادی به سادگی

[۲] Git Config File Location
[۳] Hidden

می‌توان با دستور `git checkout main` به آخرین کامیت برگشت و دوباره تمامی تغییرات و فایل‌ها در دایرکتوری ظاهر می‌شوند.
دستور `git log --oneline` پیشینه کامیت‌ها را به طور خلاصه در یک سطر نمایش می‌دهد. همچنین می‌توان به صورت پیچیده‌تر `git log --all --decorate --graph --oneline` را هم به کار برد که در واقع در آپشن‌های به کار رفته از چپ به راست ابتدا نشانگر توضیح مختصر در یک خط، نمایش در یک نمودار سلسله مراتبی، هر کامیت مربوط به کدام شاخه و در نهایت تاریخچه تمام شاخه‌های این پروژه را بیان می‌کند.

نکته ۱۳
دستور `git revert #commit id` تمام تغییراتی که با `commit id` شناسایی می‌شود را به طور کامل پاک می‌کند و با انجام این عمل یک کامیت به عنوان این پاکسازی هم به کامیت‌های موجود می‌افزاید.

نکته ۱۴
برای گرفتن کمک و خواندن نحوه فراخوانی دستورات می‌توان به صورت `git help #command name` و یا `git #command name --help` از مستندات محلی گیت برای راهنمایی استفاده کرد.

نکته ۱۵
خلاصه‌نویسی دستوراتی مانند `git log --oneline --graph --decorate --all` با تعریف یک دستور جدید در گیت با توجه به اینکه دسترسی به این دستور جدید در گیت چه سطحی باشد، به صورت `git config --global alias.{command name} "command "action"` می‌تواند انجام گیرد. البته `golabl--` به معنای دسترسی این دستور در سطح یک یوزر است و نه صرفا در حد یک پروژه. برای همین مورد نمایش پیشینه گیت با آپشن‌های مختلف می‌توان دستوری به نام `hist` به صورت `git config --global alias.hist "log --oneline --graph --decorate --all"` تعریف کرد و نحوه فراخوانی دستور هم به صورت `git hist` می‌باشد.

این برگه تقلب در تاریخ ۱۸ مهر ۱۴۰۰ و به کوشش احسان خداپناه اقدم برای استفاده شخصی‌اش حروف‌چینی شده است.