



## معرفی پروژه

### اهداف پروژه

- هدف این پروژه ساخت یک نرم‌افزار کنترل نسخه (Version Control System) است. نمونه‌های مختلفی برای این نرم‌افزار وجود دارد. ابزار گیت (Git) که در کارگاه کامپیوتر با آن آشنا شده‌اید، معروف‌ترین و پرستفاده‌ترین نرم‌افزار کنترل نسخه است.
- در این پروژه انتظار می‌رود که با دانشی که تا به اینجا در درس مبانی برنامه‌سازی کسب کرده‌اید، یک نسخه ساده‌سازی شده از این ابزار را پیاده کنید.
- انتظار می‌رود در فرآیند توسعه این پروژه از یک سیستم مدیریت نسخه، Git، استفاده کنید و پروژه را بر بستر یک مخزن Github، نگهداری کنید. در این مورد لازم است تغییرات خود را در دوره‌های کوتاه مدت commit کنید تا تاریخچه خوبی از تغییراتی که در طول پروژه ایجاد کرده‌اید، داشته باشید.

### ابزار کنترل نسخه چیست؟

ابزار کنترل نسخه، ابزاری برای ردیابی و مدیریت تغییرات سورس کد یک نرم‌افزار است. این ابزار تاریخچه هر گونه تغییر در کد را در یک پایگاه داده خاص نگهداری می‌کند. قدرت این ابزار هنگامی مشخص می‌شود که توسعه‌دهنده نرم‌افزار در فرآیند توسعه به یک اشتباه در فرآیند توسعه برخورد می‌کند و یا اینکه نیاز به بررسی تاریخچه تغییرات پروژه پیدا می‌کند. به واسطه امکاناتی که این ابزار به ما می‌دهد، بازگرداندن و یا مشاهده تغییرات گذشته به سادگی امکان پذیر خواهد بود و فرآیند توسعه نرم‌افزار با چالش‌های بسیار کمتری مواجه خواهد شد.

### توانایی‌های لازم برای پروژه

در فرآیند توسعه پروژه به همه آنچه تا کنون در درس آموخته‌اید، نیاز خواهید داشت. توانایی‌های زیر به طور مستقیم در توسعه پروژه مورد نیاز خواهند بود:

- توانایی کار با فایل.



- توانایی کار با رشته و اعمال عملیات مختلف روی آنها.
- کار با struct ها و طراحی ساختار کلی برنامه به واسطه آنها.
- کار با پوینتر: تخصیص، مدیریت و آزاد سازی حافظه.
- آشنایی با filesystem لینوکس و دسترسی فایل‌ها و نحوه ذخیره‌سازی آنها.



## توضیح بخش‌های مختلف پروژه

### انتخاب نام مناسب برای پروژه

در پیاده‌سازی پروژه باید یک کامند جایگزین برای git انتخاب کنید و پروژه را به واسطه آن پیاده‌سازی کنید. ما در ادامه از کامند neogit استفاده می‌کنیم. شما می‌توانیم در پیاده‌سازی پروژه از این کامند یا هر نام دلخواه دیگری استفاده کنید.



## تنظیمات اولیه پروژه

با استفاده از این دستور، یک شخص می‌تواند username و email خود را برای تمام پروژه‌هایی که دارد، set بکند. توجه شود که با این دستور username و gmail تمام پروژه‌ها override می‌شود.

```
neogit config --global user.name " "
```

```
neogit config --global user.email " "
```

**نکته:** در صورتی که آپشن global- در کامند نباشد، این موارد صرفاً برای خود پروژه تغییر می‌کند.

با استفاده از دستور زیر می‌توانیم به یک دستور دیگر که در پروژه موجود است یک نام نسبت دهیم و پس از آن برای استفاده از آن دستور از کامند زیر استفاده کنیم

```
neogit config (--global) alias.<alias-name> "a command"
```

برای مثال با اجرای دستور زیر می‌توانی از دستور neogit arc برای اضافه کردن همه فایل‌های تغییر یافته در پوشه src استفاده کرد.

```
neogit config alias.arc "git add src/"
```

خطاها:

- در صورتی که کامند نسبت داده شده معتبر نباشد، باید خطای مناسب در خروجی چاپ بشود



## ایجاد مخزن گیت

این دستور مخزن git را در پوشه‌ای که در آن اجرا می‌شود، راه‌اندازی می‌کند. به واسطه این دستور که یک پوشه پنهان با نام neogit ساخته می‌شود و تمام اطلاعات لازم برای کنترل ورژن و همچنین تنظیمات اولیه پروژه در آن نگهداری می‌شود.

```
neogit init
```

خطاها:

- در صورتی که پوشه پنهان با نام neogit در خود پوشه‌ای که کامند در آن اجرا می‌شود و یا یکی از پوشه‌های بالاتر آن وجود داشته باشد، خطای مناسب در خروجی چاپ بشود.
- نکته:** در صورتی که دستورهای بعدی در پوشه‌ای که در آن یا پوشه‌های بالاتر آن مخزن گیت ایجاد نشده باشد، باید خطای مناسب در خروجی نمایش داده شود.

**دستور add**

```
neogit add [file address or directory address]
```

در صورتی که آرگومان ورودی دستور یک فایل باشد، آن فایل به حالت stage می‌رود. در صورتی که یک پوشه به عنوان ورودی داده بشود، باید تمام فایل‌های تغییر یافته داخل آن پوشه به حالت stage بروند.

**نکته:** اگر یک فایل در حالت stage قرار داشته باشد، با دوباره اجرا شدن این دستور برای آن فایل تغییر به وجود نخواهد آمد. در صورتی که پوشه یا فایل مد نظر وجود نداشته باشد، خطای مناسب در خروجی نمایش داده شود.

```
neogit add s*c
```

این دستور باید بتواند فایل‌ها و یا پوشه‌هایی که نامشان به واسطه یک wildcard مشخص شده را به حالت stage ببرد.

**نکته:** تنها لازم است wildcard ستاره تنها برای یک کلمه پیاده بشود.

```
neogit add -f <file1> <file2> <dir1>
```

در صورتی که از آپشن -f در این دستور استفاده شود، می‌توان چندین فایل یا پوشه را با هم به حالت stage برد. در صورتی که تعدادی از این فایل یا پوشه‌ها وجود نداشتند، برای آنها در خروجی خطای مناسب نمایش داده بشود و دستور برای سایرین بدون مشکل اجرا بشود.

```
neogit add -n <depth>
```

**امتیازی**

```
neogit add -n <depth> (depth > 1)
```

این دستور لیستی از تمام فایل‌ها و دایرکتوری‌ها داخل پوشه را باید در خروجی نمایش دهد و مشخص کند که هر کدام آیا در حالت stage قرار دارد یا خیر. مقدار depth مشخص می‌کند که تا چند لایه در پوشه پیش برویم.

**امتیازی**

```
neogit add -redo
```

در صورتی که این دستور اجرا بشود، تمام فایل‌هایی که ابتدا در حالت stage قرار



داشته‌اند و سپس با اعمال تغییر به حالت unstage رفته‌اند را به حالت stage برمی‌گرداند.

**دستور reset**

`neogit rest [file address or directory address]`

در صورتی که آرگومان ورودی دستور یک فایل باشد، آن فایل به حالت `unstage` می‌رود. در صورتی که یک پوشه به عنوان ورودی داده بشود، باید تمام فایل‌های `stage` آن پوشه به حالت `unstage` بروند.

**موارد امتیازی:**

- پیاده‌سازی wildcard مانند دستور `add`
- پیاده‌سازی آپشن `-f` مانند دستور `add`
- در صورتی که داخل پوشه، پوشه دیگری وجود داشته باشد، محتویات آن پوشه نیز `unstage` بشوند

`neogit reset -undo`

این دستور، آخرین فایل یا فایل‌هایی که به واسطه دستور `add` به حالت `stage` رفته‌اند را به حالت `unstage` می‌برد.

**امتیازی:** قابلیت تکرار کردن این دستور تا ۱۰ مرحله



**دستور status****neogit status**

این دستور وضعیت فایل‌های داخل پوشه‌ای را که در آن دستور اجرا می‌شود به نمایش می‌گذارد. در هر خط خروجی اسم فایل نمایش داده شود و در کنار آن به کد وضعیت قرار می‌گیرد. این کد وضعیت به صورت XY است. X دو مقدار + و - را می‌پذیرد و که به معنای بودن یا نبودن فایل در حالت stage است. Y نیز حالات زیر را دارد:

- M: محتویات فایل تغییر یافته است.
  - A: فایل به پروژه اضافه شده است.
  - D: فایل پاک شده است.
  - T (امتیازی): نوع دسترسی‌های فایل عوض شده باشد. مثلاً دسترسی فایل از ۷۵۳ به ۴۴۴ تغییر پیدا کرده باشد.
- در صورتی که فایل تغییری پیدا نکرده باشد، نیازی نیست در خروجی این دستور نمایش داده شود.
- امتیازی:** با اجرای دستور وضعیت تمام فایل‌های داخل مخزن پروژه بررسی شود و در خروجی نمایش داده شود.

**دستور commit****neogit commit -m [commit message]**

با اجرای این دستور تغییرات همه فایل‌های داخل وضعیت stage در مخزن پروژه commit می‌شوند. در صورتی که هیچ فایل در حالت stage نباشد، commit ساخته نخواهد شد.

نکات:

- در صورتی که پیغام commit دارای whitespace باشد، باید پیغام بین دو double quote باشد.
- پیغام commit نباید بیشتر از ۷۲ کاراکتر باشد و در صورت بیشتر بودن خطای مناسب در خروجی چاپ شود.
- در صورتی که پیغام commit مشخص نشده بود، خطای مناسب در خروجی نمایش داده شود.
- به هر commit موفق یک id منحصر به فرد باید نسبت داده شود و در خروجی id ، زمان و پیغام commit در خروجی نمایش داده شود.

**neogit set -m "shortcut message" -s shortcut-name**

با اجرای این دستور یک shortcut برای پیغام ایجاد می‌شود و می‌توان برای commit با آن پیغام از shortcut استفاده کرد. در صورت اجرای دستور بالا، عملکرد دو دستور -s shortcut -m "shortcut message" neogit commit و یکسان خواهد بود. در صورتی که shortcut وجود نداشته باشد، خطای مناسب در خروجی نمایش داده شود.

**neogit replace -m "new shortcut message" -s shortcut-name**

با اجرای این دستور محتوای پیغام shortcut تغییر می‌کند. در صورتی که shortcut وجود نداشته باشد، خطای مناسب در خروجی نمایش داده شود.

**neogit remove -s shortcut-name**

با اجرای این دستور shortcut از قبل مشخص شده حذف خواهد شد.



در صورتی که shortcut وجود نداشته باشد، خطای مناسب در خروجی نمایش داده شود.



## دستور log

### neogit log

این دستور تمام commit های داخل مخزن گیت را به ترتیب تاریخ از جدید به قدیم نمایش می‌دهد. در هر log مربوط به یک کامیت باید موارد زیر وجود داشته باشد:

- تاریخ و ساعت commit
- پیغام commit
- شخصی که commit را انجام داده است
- id یا hash مربوط به commit
- branch که در آن commit انجام شده است.
- تعداد فایل‌هایی که commit شده‌اند

### neogit log -n [number]

این دستور به تعداد مشخص شده commit آخر را نمایش می‌دهد.

### neogit log -branch [branch-name]

این دستور تاریخچه commit هایی را نشان می‌دهد که در branch مشخص شده انجام شده‌اند.

در صورتی که branch-name وجود نداشته باشد، در خروجی خطای مناسب نمایش داده شود.

### neogit log -author [author-name]

این دستور commit هایی را که نویسنده‌شان در ورودی داده شده است، در خروجی از جدیدی به قدیم نمایش می‌دهد.

### neogit log -since <date>

### neogit log -before <date>

این دو دستور به ترتیب تاریخچه commit ها را از تاریخ مشخص شده به بعد و از تاریخ مشخص شده به قبل از جدید به قدیم در خروجی نمایش می‌دهند.



```
neogit log -search <word>
```

این دستور تاریخچه تمام commit هایی که پیغامشان شامل کلمه ورودی است، در خروجی از جدید به قدیم نمایش می‌دهد.

### موارد امتیازی:

- پشتیبانی جست و جو از حالت wildcard در یک کلمه.
- پشتیبانی جست و جو از چند کلمه (در صورتی که هر کدام از کلمات داخل پیغام بود، در خروجی نمایش داده شود).

**دستور branch**

```
neogit branch <branch-name>
```

این دستور از آخرین commit پروژه یک branch با نام مشخص شده می‌سازد. در صورتی که branch با نام مشخص شده وجود داشته باشد، خطای مناسب در خروجی نمایش داده شود.

**نکته:** branch اصلی هر مخزن master نام دارد.

```
neogit branch
```

این دستور لیست تمام branch‌های مخزن را نمایش می‌دهد.



## دستور checkout

```
neogit checkout <branch-name>
```

این دستور وضعیت فایل‌های داخل پروژه را به آخرین commit از branch مشخص شده می‌برد. در صورتی که تغییری commit نشده وجود داشته باشد، دستور نباید کار کند و در خروجی خطای مناسب نمایش داده شود.

```
neogit checkout <commit-id>
```

این دستور وضعیت فایل‌های داخل پروژه را به commit مشخص شده می‌برد. نکات قابل توجه:

- در صورتی که تغییری commit نشده وجود داشته باشد، دستور نباید کار کند و در خروجی خطای مناسب نمایش داده شود.
- هنگامی که به یک commit در گذشته checkout می‌کنیم، نمی‌توانیم تغییر در فایل‌ها ایجاد کنیم و آن‌ها را commit کنیم. به عبارت دیگر تنها محل مجاز commit کردن HEAD پروژه است.

```
neogit checkout HEAD
```

این دستور وضعیت فایل‌های داخل پروژه را به HEAD پروژه برمی‌گرداند.

### امتیازی

```
neogit checkout HEAD-n
```

این دستور وضعیت فایل‌های داخل پروژه را به n امین commit قبل از HEAD می‌برد.

# فهرست

نکات قابل توجه

۲	
۳	توضیح بخش‌های مختلف پروژه
۳	دستور <code>revert</code>
۴	دستور <code>tag</code>
۵	دستور <code>tree</code>
۶	دستور <code>stahs</code>
۸	دستور <code>pre-commit</code>
۱۰	دستور <code>grep</code>
۱۱	دستور <code>diff</code>
۱۳	دستور <code>merge</code>





## توضیح بخش‌های مختلف پروژه

### دستور revert

```
neogit revert [-e] [-m] <commit-id>
```

این دستور تغییرات انجام شده توسط کامیت داده‌شده در ورودی را برمی‌گرداند و یک کامیت جدید با تغییرات بازگردانده شده ایجاد می‌کند. فلگ e برای تغییر در پیغام آن کامیت می‌باشد. همچنین فلگ m برای مشخص نمودن parent number در زمانی که مرجع انجام شده باشد و بخواهیم مشخص کنیم در کدام طرف مرجع این کامیت انجام شده است، استفاده می‌شود.

**نکته:** پیاده‌سازی فلگ e امتیازی است.

```
neogit revert -n [-m] <commit-id>
```

به واسطه فلگ n تغییرات کامیت داده شده در ورودی بدون ثبت کامیت جدید برمی‌گردد.

```
neogit revert HEAD-X <commit-id> [-e]
```

تغییرات انجام شده به X امین کامیت قبلی برگشته و کامیت جدیدی با تغییرات بازگردانده شده ایجاد می‌شود.



## دستور tag

```
neogit tag -a <tag-name> [-m <message>] [-c <commit-id>] [-f]
```

این دستور یک tag با نام <tag-name> می‌سازد. اگر اسم tag تکراری بود، باید ارور مناسب دهید. در صورتی که از فلگ f استفاده شود، تگ جدید را می‌توان overwrite کرد. فلگ اختیاری m یک پیغام را به تگ اضافه می‌کند. در غیر این صورت پیغام تگ خالی در نظر گرفته خواهد شد. فلگ اختیاری c مشخص می‌کند که تگ بر روی کدام کامیت قرار می‌گیرد. در صورتی که از فلگ c استفاده نشده بود، تگ را برای کامیت فعلی که روی آن قرار داریم در نظر خواهیم گرفت.

**نکته:** توجه شود که برای هر تگ، باید نام تگ، commit-id، فرد تگ کننده، تاریخ و زمان ساخت تگ و پیغام تگ نگهداری شود.

```
neogit tag
```

این دستور لیست تمام تگ‌ها را به صورت صعودی و به ترتیب حروف الفبا نمایش می‌دهد.

```
neogit tag show <tag-name>
```

این دستور اطلاعات تگی که در <tag-name> مشخص می‌شود را نمایش می‌دهد. این اطلاعات شامل نام تگ، آیدی کامیت commit آن تگ، نام فرد تگ کننده، تاریخ و زمان ساخت تگ و در صورت وجود، message آن تگ. برای نمونه:

```
neogit tag show v1.0a
tag v1.0a
commit 81912ba
Author: John Appleseed <john.appleseed@sharif.edu>
Date: Sun Jan 21 19:26:10 2023
Message: version 1.0a
```

**دستور tree****neogit tree**

با اجرای این دستور یک ساختار درختی از تاریخچه کامیت‌ها به همراه برنچی که در آن بوده‌اند در خروجی نمایش داده می‌شود.

```
1e
|
43
| \
f4 23
| |
l1 54
| /
34
|
23
```

در مثال بالا پس از دو کامیت یک برنچ از برنچ اصلی ساخته شده است و پس از دو کامیت در هر برنچ با برنچ اصلی مرچ شده است و نهایتاً یک کامیت دیگر در برنچ اصلی زده شده است.

**نکته:** در گره‌های درخت کافی است دو کاراکتر اول آیدی کامیت را قرار دهید.  
**امتیازی:** نمایش درخت با بیش از دو برنچ و درخت‌های با ساختارهای پیچیده.



## دستور stash

قابلیت stash به کاربر اجازه می‌دهد که تغییرات انجام شده روی محتوای دایرکتوری مورد نظر را ذخیره کند و از دست ندهد و در عین حال به یک نسخه شده commit از آن فایل‌ها برگردد. در واقع از دستور stash neogit زمانی استفاده می‌کنیم که می‌خواهیم استیت کنونی فایل‌های خود را ذخیره کنیم و در عین حال به یک نسخه تمیز برگردیم. این دستور تغییرات لوکال دایرکتوری را ذخیره می‌کند و سپس تمام دایرکتوری را به حالتی که در کامیت HEAD داشته‌است برمی‌گرداند.

توصیه می‌شود [این لینک](#) را مطالعه کنید.

```
neogit stash push [-m <message>]
```

این دستور تغییرات لوکال اعمال‌شده را در یک stash entry جدید ذخیره می‌کند، و وضعیت پوشه کنونی را به کامیت HEAD بازمی‌گرداند. کاربر باید بتواند در صورت تمایل با آپشن -m توضیحاتی برای stash ذخیره کند.

```
neogit stash list
```

این دستور تمام stash entry هایی که هم‌اکنون داریم را لیست می‌کند. stash entry ها به شکل stack ذخیره می‌شوند، یعنی هرگاه از دستور push استفاده می‌کنیم در صدر لیست یک stash entry ساخته می‌شود. stash entry ها از بالای لیست و از صفر شماره گذاری می‌شوند. به ازای هر stash entry بایستی index آن (جایگاه آن در لیست)، نام branch که هنگام ساخت stash entry روی آن بودیم، و همچنین message آن stash entry نمایش داده شود.

```
neogit stash show <stash-index>
```

این دستور مانند دستور diff عمل می‌کند، و باید diff بین محتوای stash شده و کامیتی که آن موقع stash کردن روی آن بودیم نمایش داده شود.

```
neogit stash pop
```

این دستور یک stash entry را از بالای لیست stash ها برمی‌دارد، از لیست پاک می‌کند و آن را روی وضعیت فعلی دایرکتوری اعمال می‌کند. لازم به ذکر است که این عمل می‌تواند باعث بروز conflict شود لذا انتظار ما مانند دستور merge است. همانطور که گفته شد تغییرات را اعمال کنید ولی می‌توانید به صورت امتیازی، ها conflict را هندل کنید و در صورت وجود کانفلیکت، stash را از بالای لیست پاک نکنید تا زمانی که کاربر کانفلیکت را هندل کند و سپس دستور neogit stash drop را بزند. همچنین به صورت



امتیازی می‌توانید این را پیاده کنید که کاربر بتواند به صورت اختیاری با ایندکس، stash مدنظر خود را مشخص کند و لزوماً استش بالای لیست اعمال نشود.  
**نکته:** پشتیبانی بیش از یک stash entry امتیازی است.

### امتیازی

```
neogit stash branch <branch-name>
```

این دستور یک شاخه با نام اشاره‌شده با شروع از کامیتی که از آنجا stash ساخته شده‌است می‌سازد و سپس تغییرات stash entry را روی آن اعمال می‌کند (اما کامیت نمی‌کند) و stash entry را حذف می‌کند و شاخه جدید را out check می‌کند. طبیعتاً stash entry ای که بالای لیست است مدنظر است اما می‌توانید به صورت امتیازی این قابلیت را پیاده کنید که بتوان های stash دیگر را هم با استفاده از index آنها استفاده کرد.

```
neogit stash clear
```

این دستور تمامی stash entry ها را پاک می‌کند.

### امتیازی

```
neogit stash drop
```

یک stash entry را پاک می‌کند. به صورت پیشفرض بالای لیست اما می‌توانید به صورت امتیازی این قابلیت را پیاده کنید که بتوان با index مشخص کرد که کدام stash entry پاک شود.



## دستور pre-commit

این دستور بر روی فایل‌هایی که در مرحله‌ی stage قرار دارند اجرا شده و همانطور که از اسمش معلوم است، نقش بررسی فایل‌ها پیش از commit کردن را دارد. برای این بخش از مفهومی به نام hook استفاده می‌کنیم. hook همان قراردادهایی هستند که ما برای بررسی فایل‌ها بر روی دستور pre-commit قرار می‌دهیم. در ادامه لیستی از این hook ها در اختیار شما قرار داده شده که فقط پیاده کردن بخشی از آن‌ها اجباری می‌باشد.

```
neogit pre-commit hooks list
```

این دستور تمام hook های موجود را به کاربر نشان می‌دهد.

```
neogit pre-commit applied hooks
```

این دستور تمام hook هایی که در دستور چک خواهند شد را به کاربر نشان می‌دهد.

```
neogit pre-commit add hook <hook-id>
```

این دستور hook مورد نظر را به لیست hook هایی که چک می‌شوند اضافه می‌کند.

```
neogit pre-commit remove hook <hook-id>
```

این دستور hook نوشته شده را از لیست hook هایی که چک می‌شوند حذف می‌کند.

```
neogit pre-commit
```

این دستور بر روی تمامی فایل‌های stage شده ران می‌شود و hook های مناسب هر کدام را بر روی آن اجرا می‌کند و در نهایت برای تمامی فایل‌ها خروجی به فرمت زیر تولید شود:

```
“File name”:  
“hook-id1” .....PASSED  
“hook-id2” .....FAILED  
“hook-id3” .....SKIPPED
```

- PASSED: فایل مورد نظر hook را با موفقیت به انجام رساند.
- FAILED: فایل مورد نظر hook را با موفقیت به انجام نرساند.
- SKIPPED: hook مورد نظر برای این فرمت از فایل نبوده است.

**امتیازی:** نتایج تست‌های hook به صورت رنگی با رنگ مناسب در خروجی نمایش



داده شوند.

**نکته:** با اضافه شدن hook به پروژه در صورت fail شدن یکی از hook لازم است که از کامیت شدن تغییرات توسط کاربر جلوگیری کنید و یا با پرسش از کاربر از انجام کامیت با وجود hook پاس نشده مطمئن شوید.

لیست hook ها:

توضیحات	file format	hook-id
در فایل‌های c و cpp کامنت TODO و در فایل تکست کلمه TODO نباشد.	.cpp .c .txt	todo-check
در انتهای فایل‌ها whitespace اضافه نباشد.	.cpp .c .txt	eof-blank-space
فرمت فایل مورد نظر معتبر باشد.	7 valid formats	format-check
برای پرانتز، کروشه، براکت به ازای هر کاراکتر باز یک کاراکتر بسته وجود داشته باشد.	.cpp .c .txt	balance-braces
چک کردن دندان‌گذاری در متن به سبک <a href="#">K &amp; R</a> یا <a href="#">Allman</a>	.cpp .c	indentation-check
چک کردن ارورهای static مانند error compile در کد.	.cpp .c	static-error-check
سایز فایل موجود از ۵ مگابایت بیشتر نباشد.	all	file-size-check
کاراکترهای موجود در فایل از ۲۰۰۰۰ بیشتر نباشد.	.cpp .c .txt	character-limit
مدت زمان فایل‌های صوتی و تصویری از ۱۰ دقیقه بیشتر نباشد.	.mp۴ .wav .mp۳	time-limit

**نکته:** از موارد بالا لازم است ۴ مورد را پیاده‌سازی کنید. موارد اضافه‌تر امتیازی به حساب خواهند آمد.

### امتیازی

```
neogit pre-commit -u
```

هنگامی که دستور pre-commit با این پرچم ران شود در پایان pre-commit hook هایی که FAILED شده‌اند و در جدول برای آن‌ها قابلیت تصحیح وجود دارد، خود به خود به فرمت صحیح مورد نظر تغییر (در صورت امکان) پیدا می‌کنند.

**نکته:** پیاده‌سازی این دستور برای هر کدام از هوک‌های زیر امتیازی خواهد بود:

- eof-blank-spac

- balance-braces

- indentation-check

### امتیازی

```
neogit pre-commit -f <file1> <file2>
```

در این دستور فرایند pre-commit فقط برای فایل‌های مورد نظر اجرا می‌شود. در صورت وجود نداشتن یا stage نبودن فایل، خطای مناسب در خروجی نمایش داده شود.



## دستور grep

```
neogit grep -f <file> -p <word> [-c <commit-id>] [-n]
```

با اجرای این دستور کلمه مدنظر در فایل مشخص شده جست و جو می‌شود و در خروجی خطوطی از فایل که دارای کلمه بودند نمایش داده می‌شوند. در صورتی که کامیت با فلگ c مشخص شود، جست و جو در کامیت مشخص شده انجام خواهد شد. با استفاده از فلگ n در کنار هر خط خروجی شماره خط در فایل نیز باید نمایش داده شود.

**امتیازی:** در خطوط خروجی کلمه مدنظر با رنگ جداگانه‌ای نمایش داده شود،

**امتیازی:** پشتیبانی از wildcard در کلمه جست و جو شده.



**دستور diff**

```
neogit diff -f <file1> <file2> -line1 <begin-end> -line2 <begin-end>
```

این دستور دو فایل داده شده را با هم خط به خط مقایسه می‌کند. اگر آپشن‌های لاین به دستور داده شود شما باید از شروع خط اول فایل اول تا پایان آن را با شروع خط اول فایل دوم تا پایان خط آن را با هم مقایسه کنید. در صورت نبود آپشن line شما باید تمام فایل را با فایل دیگر مقایسه کنید. در نهایت هریک از آپشن‌های 1-line و 2-line فایل مربوطه به طور کامل در نظر گرفته میشود (از خط اول فایل تا آخر). در صورتی که خط پایانی داده شده از تعداد خط‌های موجود در یک فایل بیشتر بود باید تا انتهای فایل diff انجام شود. در هنگام بررسی diff بین دو فایل، فایل‌ها خط به خط مقایسه می‌شوند و null space های بین خطوط اعم از space, enter, tab صرف نظر می‌شوند. در صورت تفاوت میان دو خط، در خروجی به شکل زیر نمایش می‌دهیم:

```
««««««
<file1_name>-<line_number>
FOP is cool :)
<file2_name>-<line_number>
AP is very cool :)))
»»»»»»
```

مثلا اگر در فایل اول داشته باشیم:

```
FOP is cool :)
AP .....
```

و در فایل دوم داشته باشیم:

```
FOP is cool :)

AP .....
```

این دو فایل با هم تفاوتی ندارند زیرا null space در نظر گرفته نمی‌شود.  
**امتیازی:** رنگ گذاری، کافی است خط‌های مربوط به فایل اول و خط‌های مربوط به فایل دوم به دلخواه با رنگ‌های متمایز نشان داده شوند.  
**امتیازی:** در صورتی که دو خط تنها در یک کلمه متفاوت بودند (تعریف کلمه را به صورت عبارتی در نظر بگیرید که قبل و بعد آن اسپیس است و خود شامل اسپیس نمی‌باشد) آن را به صورت زیر نمایش دهد:



```
«««««  
<file1_name>-<line_number>  
Dars FOP »Shirin« ast  
<file2_name>-<line_number>  
Dars FOP »Talkh« ast  
»»»»»
```

```
neogit diff -c <commit1-id> <commit2-id>
```

در این دستور شما صرفاً باید بین فایل‌های دو commit مقایسه‌ای انجام دهید. اگر فایلی در commit ای بود و در دیگری نبود باید نمایش داده شود. این مورد برای هر commit ای باید جداگانه بررسی شود. اگر در دو کامیت فایلی با نام مشترک و دایرکتوری مشابه موجود بود باید به درون آن فایل‌ها رفته و دستور diff کامل بر روی دو فایل را انجام دهد و مانند دستور diff فایل‌ها در صورت وجود اختلاف این تفاوت‌ها را نشان بدهد. این کار باید به ازای تمام فایل‌هایی که ویژگی گفته شده را دارند انجام شود.

## دستور merge

```
neogit merge -b <branch1> <branch2>
```

با اجرای این دستور head های دو برنچ با یکدیگر مرج میشوند و تبدیل به یک برنچ جدید خواهند شد. مرج دو برنچ به این شکل است که اگر فایل‌ها دارای اسم متفاوت و در دایرکتوری‌های متفاوتی باشند آن‌ها اضافه خواهند شد ولی اگر در دو برنچ فایلی وجود داشت که اسم یکسان و دارای آدرس نسبی یکسانی بودند دستور diff بر روی آن‌ها اجرا شود.

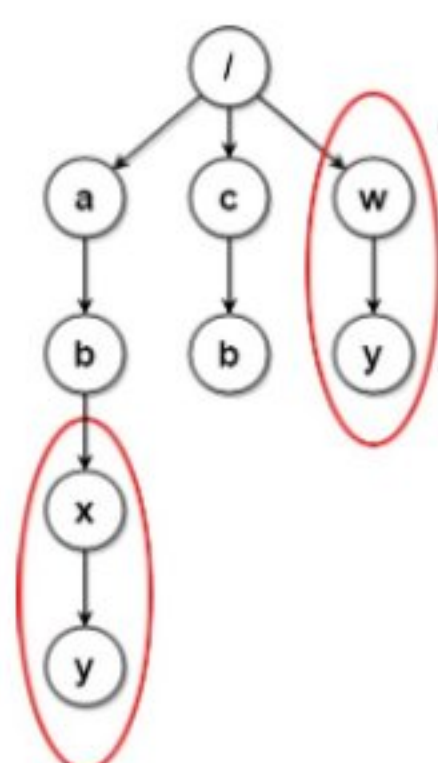
سپس پس از اعمال دستور زیر خروجی را به شکل زیر نمایش دهد:

```
«««««file1_absolute_path»»»»»
<file1_name>-<line_number_which_conflict_happened>
FOP jazzab ast
«««««file2_absolute_path»»»»»
<line_number_which_conflict_happened>
Kheir. AP besar jazzab tar ast
```

### امتیازی

```
neogit merge -b <branch1> <branch2> -clean
```

اگر دو برنچ را مرج کردیم و متوجه شدیم دو فایل یکسان (محتوا و اسم یکسانی دارند) ولی در مسیرهای متفاوتی قرار دارند برای کاهش ارتفاع درخت بیاید فایل با ارتفاع بیشتر را حذف کنید یعنی پس از اجرای این دستور دو گره x و y حذف خواهند شد (درخت‌های خیلی پیچیده در تحویل بررسی نخواهد شد).



**کانفلیکت (امتیازی):** اگر موقع merge کردن، دو فایل در تعدادی خط conflict خوردند، به ازای هر خط conflict باید محتویات هر دو خط و نام و آدرس فایل را با فرمت زیر به کاربر داده و به کاربر اجازه دهید با وارد کردن عدد ۱ یا ۲ انتخاب کند کدام خط



## در merge نهایی قرار بگیرد:

```
<file address>  
<branch1>-<line-number>:  
<line1>  
<branch2>-<line-number>:  
<line2>
```

به جای `<branch ۱>` و `<branch ۲>` نام branch ها را قرار داده و به جای `line-number` شماره خطی که در آن تفاوت رخ داده را بنویسید.

کاربر در هر `conflict` میتواند عبارت `edit` نوشته و به صورت دستی، چیزی که جایگزین آن خط می‌شود را تایپ کند.

کاربر باید بتواند با وارد کردن `quit` از فرایند `resolve` کردن `conflict` خارج شوند. در صورت خارج شدن از فرایند `conflict` کلیه تغییرات ناشی از `merge` از بین رفته و وضعیت فایل‌ها به حالت اولیه خود برمی‌گردد.

**امتیازی:** کلمات متفاوتی که در هر فایل `conflict` وجود دارد را به رنگ‌های مختلف نشان دهید. یعنی برای مثال، اگر در فایل اول عبارت `Hello World` و در فایل دوم عبارت `Hi World` بود، `Hello` در فایل اول را با رنگ قرمز و `Hi` در فایل دوم را با رنگ آبی نشان دهید.