EJERCICIO GIT

Para hacer la práctica, no olvides hacer captura de pantalla y adjuntarla en los mismos puntos en los que yo he adjuntado la mía.

Primero, descarga e instala Visual Studio Code

Crea una carpeta en el escritorio llamada new git Project. Inicia en ella un repositorio de GIT (Recuerda que también lo podemos clonar con el comando Git clone). Crea dentro de ella 3 carpetas llamadas “index.html”, “css” y “js”. Después crear un fichero con cada código( el de html llamado también index y los de css y javascript llamados app) y guardar dicho fichero en su respectiva carpeta (recuerda usar el comando git status para ver que está todo OK). Ahora tendremos 3 ficheros en nuestro Working Directory. Mediante el comando git add vamos a pasar el fichero index.html al Staging área. Usa el comando Git Status para ver como el fichero en html ha desaparecido del directorio de trabajo y está en el staging index.

Dato a saber: con el comando git rm --cached fichero devolveríamos el fichero subido al staging index al working directory.

Ahora subimos el resto de archivos de nuestro directorio de trabajo al staging área. Lo podemos hacer uno a uno o usando el comando git add . (sube todo lo que tengamos en el directorio de trabajo al staging área).

Hacer un git commit. Se debe abrir VSCode indicando los 3 cambios a ser comiteados. En línea 1 escribir Initial Commit (Porque es el primerísimo commit). Guarda y cierra VS Code.

En GIT deberíais estar así. Aparecerá el SHA de nuestro commit.

Texto

Descripción generada automáticamente

Código HTML EN INDEX.HTML

**<!doctype html>**

<html lang="en">

<head>

<meta charset="utf-8">

<title>Blog Project</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta name="description" content="">

<link rel="stylesheet" href="css/app.css">

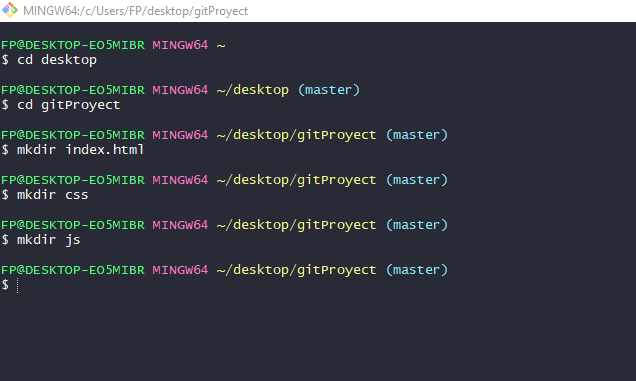
</head>

<body>

<script src="js/app.js"></script>

</body>

</html>



**PARTE 2**

Ahora modificamos el fichero HTML introduciendo en “body” lo siguiente:

<header>

<h1>Expedition</h1>

</header>

Una vez lo hemos guardado (recuerda ver git status para ver si se ha modificado) debemos pasarlo del working directory al staging index (qué comando era?). Una vez lo tengamos hacer el segundo commit con el mensaje Add header to blog.

Ahora vamos a probar el comando Git Diff. Este comando nos permite ver cambios que se hayan hecho en los ficheros pero que aún no hayan sido comiteados, por lo que seguirán en el staging index. Vamos a cambiar Expedition por Adventure en nuestro fichero html. Guardamos en VSCode e iniciamos comando Git Diff.

Deberíamos ver lo siguiente:

Texto

Descripción generada automáticamente

Al final la información es la misma que en el comando git log -p, porque al final git log usa git diff.

Lo añadimos al staging área y hacemos commit para dejarlo en el repositorio con nombre change adventure

Ahora con el comando Touch vamos a añadir un Word a nuestro directorio de trabajo llamado Project.docx. Si este documento lo almacenamos en una carpeta llamada .gitignore, cuando vayamos a hacer un gitt add. Nos subirá todo lo que tengamos en nuestro directorio al staging área salvo lo que esté dentro de la carpeta .gitignore. Aparecerá lo siguiente al hacer git status:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ahora, vamos a probar el comando Git Tag. Éste sirve para añadir información a los commits, por ejemplo sirve para indicar qué Sha es la versión 1.0. Para probarlo, introducimos el comando git tag -a v1.0. Esto afectará al commit más reciente. Se abrirá VSCode e introduciremos en la línea 1 el mensaje “Ready for content”. Para comprobar que lo hemos hecho bien, al introducir sólo el comando git tag aparecerá v1.0. Ahora, prueba a eliminarlo usando el comando git tag -d v1.0.

Ahora, vamos a hacer un git tag a un commit antiguo. Para ello, veremos los Sha de nuestros commits haciendo git log –oneline. Vamos a poner el tag “Testing git tag” al commit add header to blog.

Branching

Ahora vamos con las ramas de nuestro proyecto. Cuando un commit nuevo es añadido a nuestro repositorio, la rama Master, que es la que estamos usando en nuestro caso, se mueve hasta ahí. El término “head” indica la rama en la que estamos trabajando. Usando a secas el comando git branch aparecen las ramas que tenemos en nuestro proyecto.

Ahora, vamos a crear otra rama que se llame sidebar. Al crearla, veremos que seguimos trabajando en la rama Master. Para cambiar de rama, usaremos el comando git checkout sidebar. Aparecerá lo siguiente:

Texto

Descripción generada automáticamente

Al usar este comando, veremos que Git elimina todos los archivos y directorios del directorio de trabajo que Git está rastreando, almacenándolos igualmente en el repositorio, por lo que no se pierde nada. Es decir, básicamente se lleva todo a la nueva rama. Ahora, para ver en qué situación estamos, un git log --oneline --decorate

Ahora, haciendo git branch aparecerán todas nuestras ramas, marcándonos como activa la rama sidebar. Por último, eliminamos la rama sidebar. Para ello, tenemos que salir de esta rama com un git checkout, ya que git no deja eliminar una rama si estamos en ella, o si hay algún commit que está sólo en esa rama. Para eliminarla, usaremos el comando git branch -d sidebar.

Ahora, vamos a corregir el fichero html para que esté de esta forma y lo comiteamos.

**<!doctype html>**

<html lang="en">

<head>

<meta charset="utf-8">

<title>Blog Project</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta name="description" content="">

<link rel="stylesheet" href="css/app.css">

</head>

<body>

<header>

<h1>Expedition</h1>

</header>

<div class="container">

<main>

</main>

</div>

<footer>

Made with ♥ @ Udacity

</footer>

<script src="js/app.js"></script>

</body>

</html>

Además, eliminaremos la carpeta .gitignore de nuestro directorio de trabajo.

Al hacer git status, debe aparecer que estamos en branch master y con nothing to commit, working tree clean.

El paso siguiente va a ser empezar a trabajar con varias ramas haciendo a su vez varios cambios.

Primero, modificamos el fichero CSS introduciendo lo siguiente:

body {

background-color: #00cae4;

}

Guardamos, hacemos git status para comprobar que lo hayamos modificado y pasamos al staging área y comiteamos. Como nombre le pondremos set background colour for page.

Haciendo un git log --oneline --decorate deberíamos estar así:

Texto

Descripción generada automáticamente

Acto seguido, creamos la rama sidebar para el commit “add starting html structure”. Deberíamos estar así:

Texto

Descripción generada automáticamente

Si abrimos nuestro fichero CSS en VSCode nos aparecerá en BLANCO! Sin los cambios que hemos hecho. Hasta que no cambiemos a la rama principal no veremos nuestro commit hecho, por lo que ahora no vemos el cambio de color que hemos hecho.

Ahora, modificamos el fichero html incluyendo lo siguiente:

<div class="container">

<main>

</main>

</div>

*<!-- start of new content -->*

<aside>

<h2>About Me</h2>

<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eos, debitis earum molestias veniam suscipit aliquam totam exercitationem tempore neque vitae. Minima, corporis pariatur facere at quo porro beatae similique! Odit.</p>

</aside>

*<!-- end of new content -->*

<footer>

Made with ♥ @ Udacity

</footer>

Comiteamos y guardamos con el mensaje aside changes.

Ahora, cambiamos a la rama master (recuerda el comando git checkout). También, volvemos a abrir el fichero html para cambiar la línea h1, yo he incluido adventure max. Ahora lo comiteamos con el mensaje Improve site heading for SEO.

A continuación, volvemos a cambiar a la rama sidebar. El contenido que hemos añadido a la rama Master no está visible en sidebar!.

Abrimos el fichero html y hacemos algún cambio en el elemento <aside>. Yo he añadido mi película favorita. Guardamos y comiteamos con el mensaje “improvge site heading dor SEO”.

Ahora, para terminar viendo cambios que podemos hacer, vamos a crear una rama llamada footer y cambiar a la vez. Para ello, utilizamos el comando git checkout -b footer master. Haciendo un git log --oneline --decorate debería aparecer lo siguiente:

Texto

Descripción generada automáticamente

Abrimos el fichero html y en footer introducimos lo siguiente:

<section>

<h3 class="visuallyhidden">Social Links</h3>

<a class="social-link" href="https://twitter.com/udacity">

<img src="img/social-twitter.png" alt="Twitter">

</a>

<a class="social-link" href="https://www.instagram.com/udacity/">

<img src="img/social-instagram.png" alt="Instagram">

</a>

<a class="social-link" href="https://plus.google.com/+Udacity">

<img src="img/social-google.png" alt="Google Plus">

</a>

</section>

Comiteamos con el mensaje Add links to social media

En este momento, tenemos varios cambios en varias ramas. Para poder ver todas las ramas y los cambios hechos, usamos el comando

git log --oneline --decorate --graph --all

Os tiene que aparecer lo siguiente

Texto

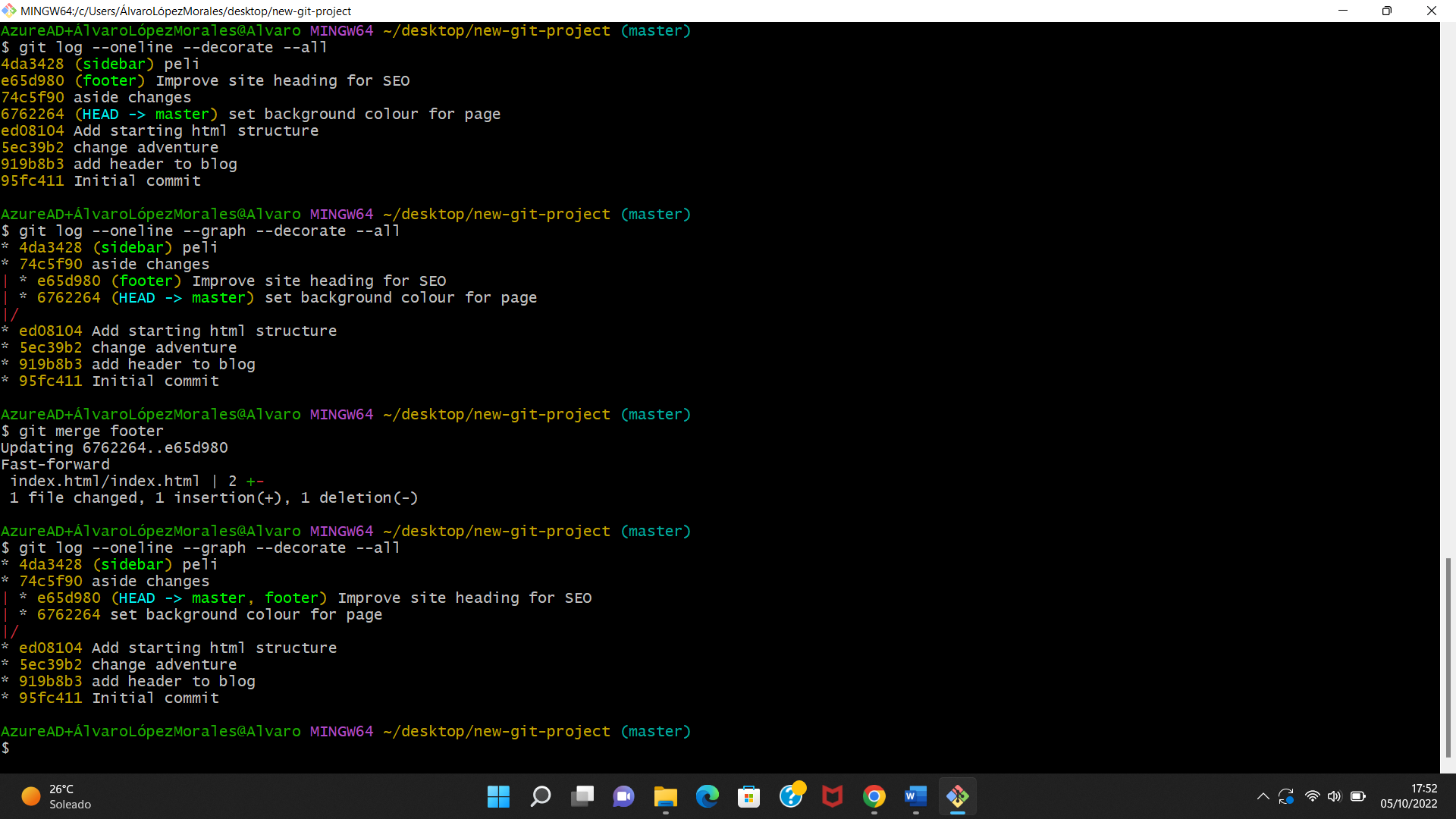
Descripción generada automáticamente

**PARTE 3**

Vamos a empezar con el concepto Merging. Básicamente este significa combinar 2 ramas. Importante: Merging implica hacer un commit. Como la rama de Footer está ahora mismo por delante de la rama Master, hacer un merge de footer a master causará un fast-forward merge. Una combinación de avance rápido simplemente moverá la rama de la que acabamos de salir hacia adelante hasta que apunte a la misma confirmación a la que apunta la otra rama.

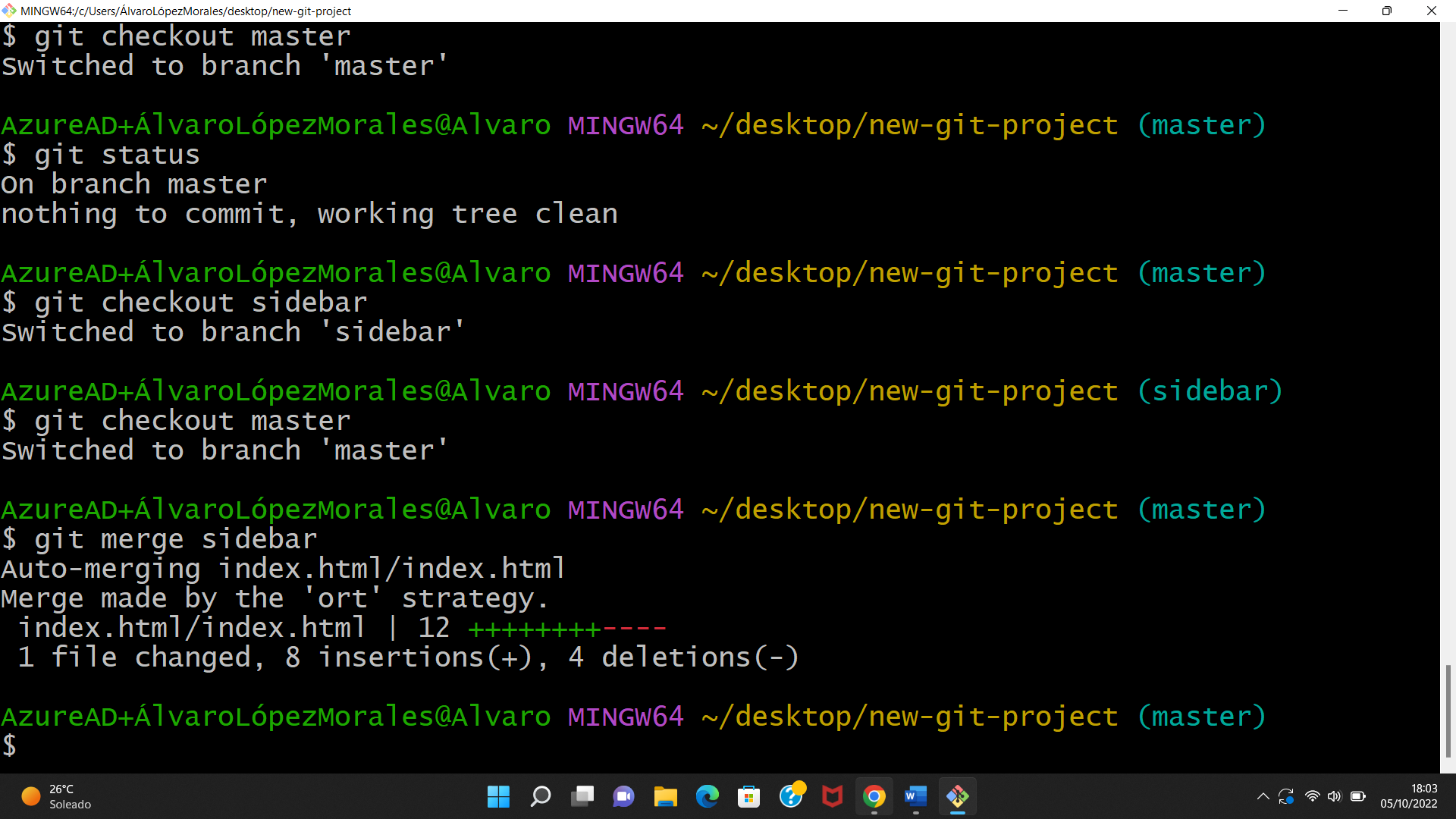
Si nos equivocamos al hacer un merge, podemos volver atrás usando el comando siguiente: *git reset --hard HEAD^*

Ahora nos situamos en la rama master, e introducimos el comando git merge footer. Os tiene que aparecer lo siguiente haciendo un git log --oneline –graph --decorate –all



Ahora, probamos que el fichero HTML está igual tanto si estamos en la rama master como si estamos en la rama footer. Para ello, primero lo abrimos estando en la rama Master, luego cambiamos a la rama footer y así comprobamos que el fichero permanece idéntico.

Acto seguido, vamos a hacer un merge con la rama Sidebar. Para ello, primero comprobamos que estamos en la rama Master y usamos el comando git merge sidebar. Os tiene que aparecer esto, cerrando VSCode (recordad que se abre porque al hacer un merge estáis haciendo un commit) y cerrándolo aceptando el mensaje automático que os salga.



MERGE CONFLICT

Vamos a probar qué ocurre cuando tenemos un conflicto entre 2 ramas a la hora de hacer un merge. Esto pasará, por ejemplo, cuando cambiemos el mismo encabezado en 2 ramas. ¿Cómo sabe Git con cuál quedarse? No lo sabe.

En nuestro caso, vamos a forzar un conflicto de merging.

**Forcing A Merge Conflict!**

Remember that a merge conflict occurs when *the exact same line(s) are changed in separate branches*. Let's alter the page's heading on two different branches. So Let's:

* change the heading on the master branch
* create a heading-update branch *that's located on the commit right before the recently modified master branch*
* change the *same* heading
* switch back to the master branch
* merge in the heading-update branch

**Change Heading On Branch 1**

Since the master branch is just a regular ol' branch like all the rest, let's just alter the heading while we're on the master branch. So change the <h1> heading from whatever you have it to something else. For me, the heading is currently "Adventure" on line 13, and I'm changing it to "Quest".

Once you've made the change, save the file and commit it to the repository.

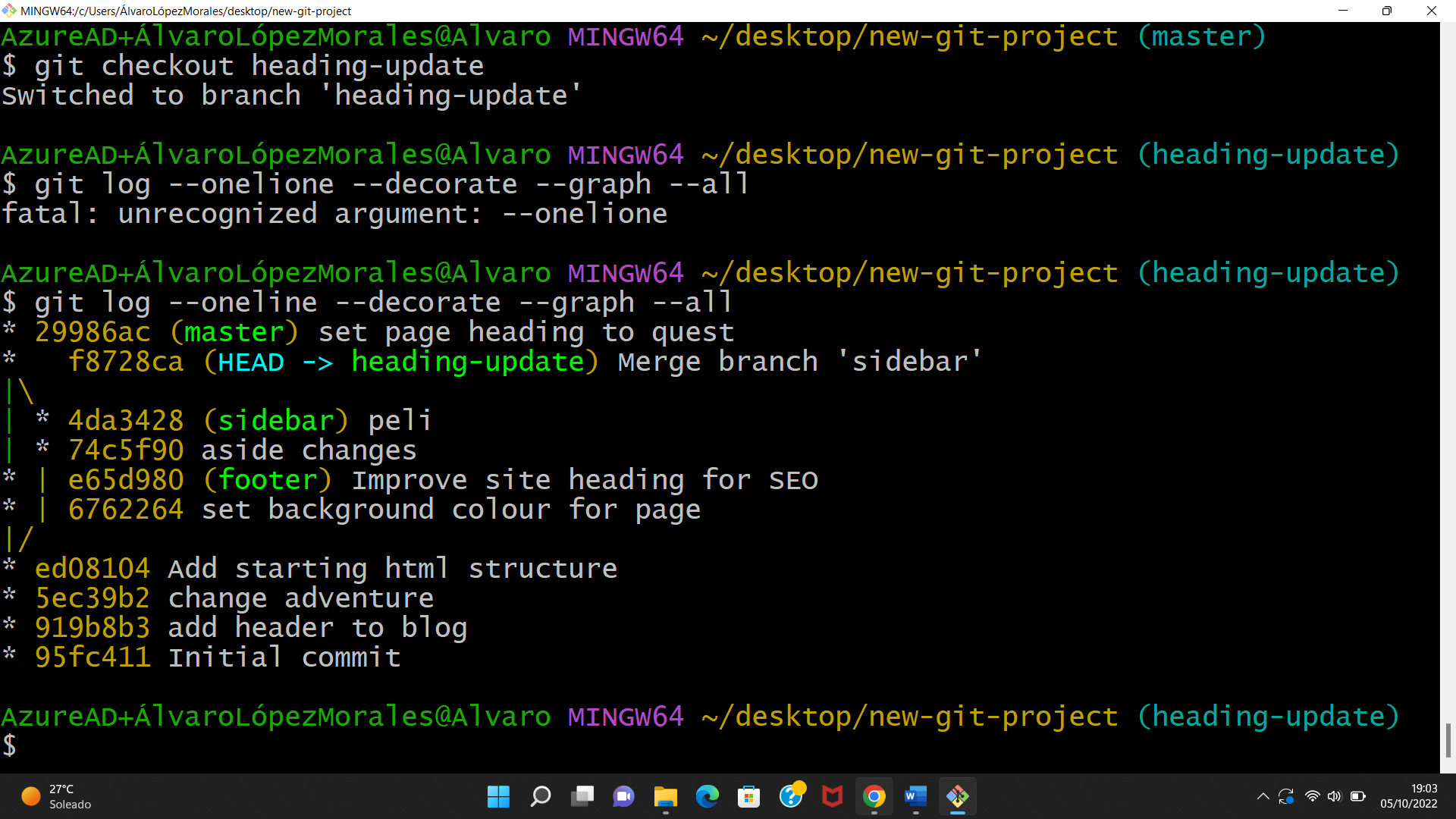
**Change Heading On Branch 2**

Now we need to create a different branch and update the heading on that branch.

Now this is important, we need to create a branch that's *not branching from the master branch*. If we make a change that branches *off* of the master branch, then *that* change will be "ahead" of this one and Git will just use that change instead of the one we just made on master. So we need to put the branch "in the past".

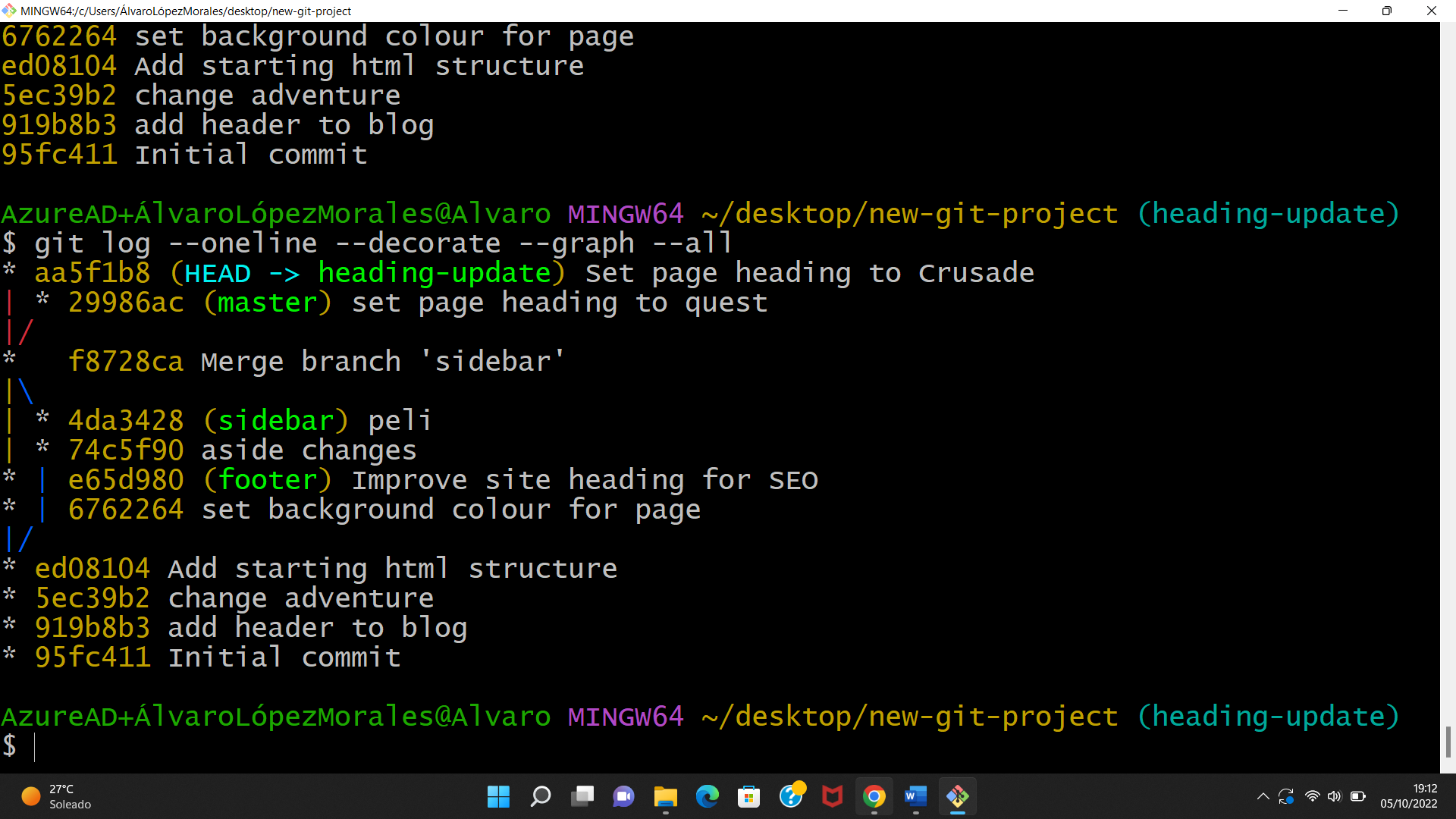
Let's just create a branch that's on the commit right *before* the most recent one. So use Git log to get the previous commit's SHA and create a branch on that commit.

Deberíais estar así:



Ahora que ya tenemos la nueva rama creada, vamos a cambiar la misma línea que en la rama master en el fichero HTML. En este caso, cambiamos de nuevo la línea 13, cambiando adventure por Crusade y hacemos commit con el mensaje “Set page heading to Crusade”.

Os tiene que aparecer lo siguiente al hacer el comando git log --oneline --decorate --graph –all:



A continuación, cambiamos a la rama master y hacemos un merge con la rama heading-update. Tenéis que ver lo siguiente:

Texto

Descripción generada automáticamente

**Merge Conflict Output Explained**

The output that shows in the Terminal is:

$ git merge heading-update

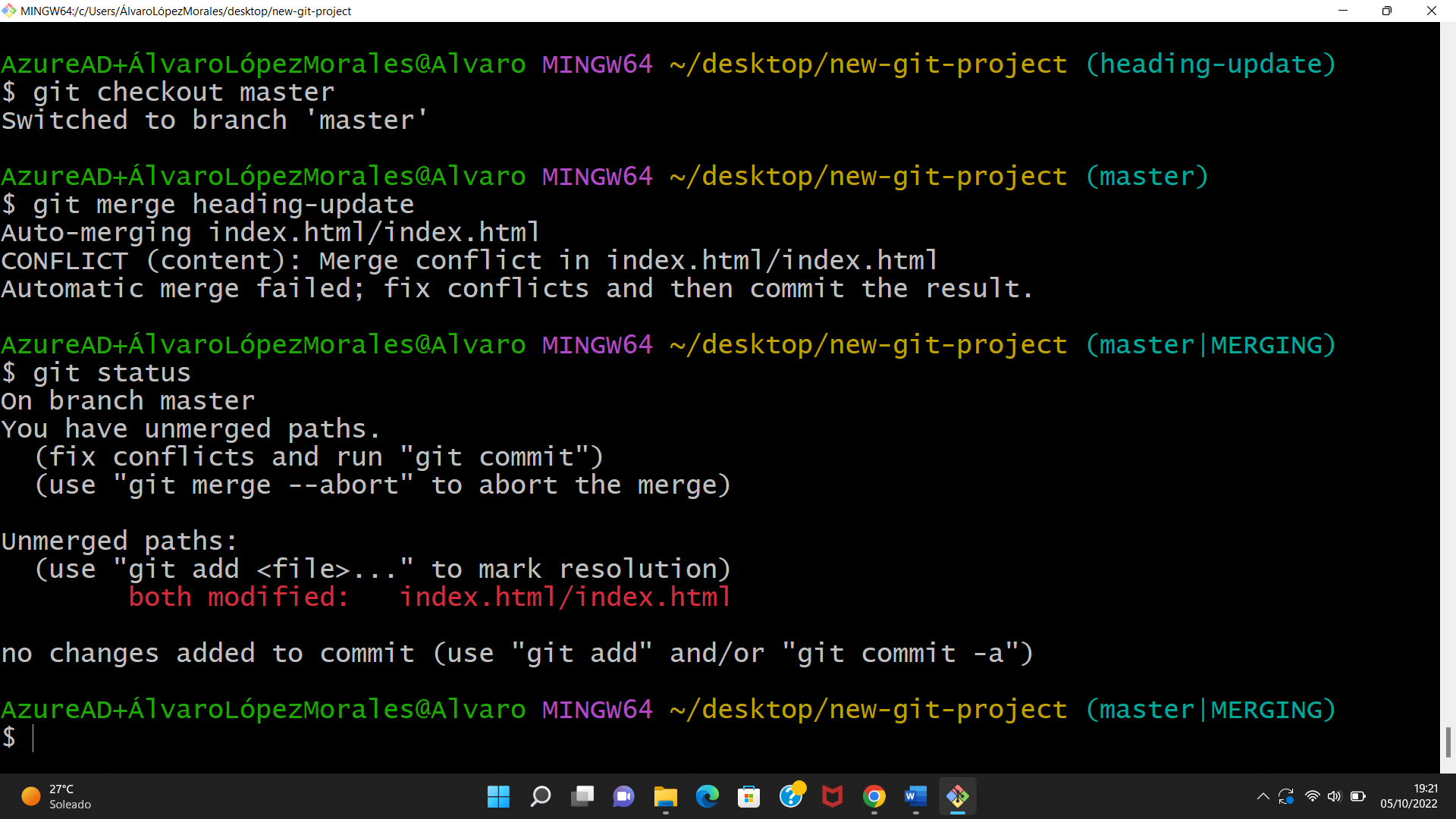
Auto-merging index.html

CONFLICT (content): Merge conflict **in** index.html

Automatic merge failed; fix conflicts and **then** commit the result.

Notice that right after the git merge heading-update command, it tries merging the file that was changed on both branches (index.html), but that there was a conflict. Also, notice that it tells you what happened - "Automatic merge failed; fix conflicts and then commit the result".

Al hacer ahora git status vemos que nos indica dónde está el problema:



Si abrimos el fichero html ahora mismo en nuestro editor, también nos indicará el problema:

Captura de pantalla de computadora

Descripción generada automáticamente

El editor nos marca lo siguiente:

<<<<<<< HEAD everything below this line (until the next indicator) shows you what's on the current branch

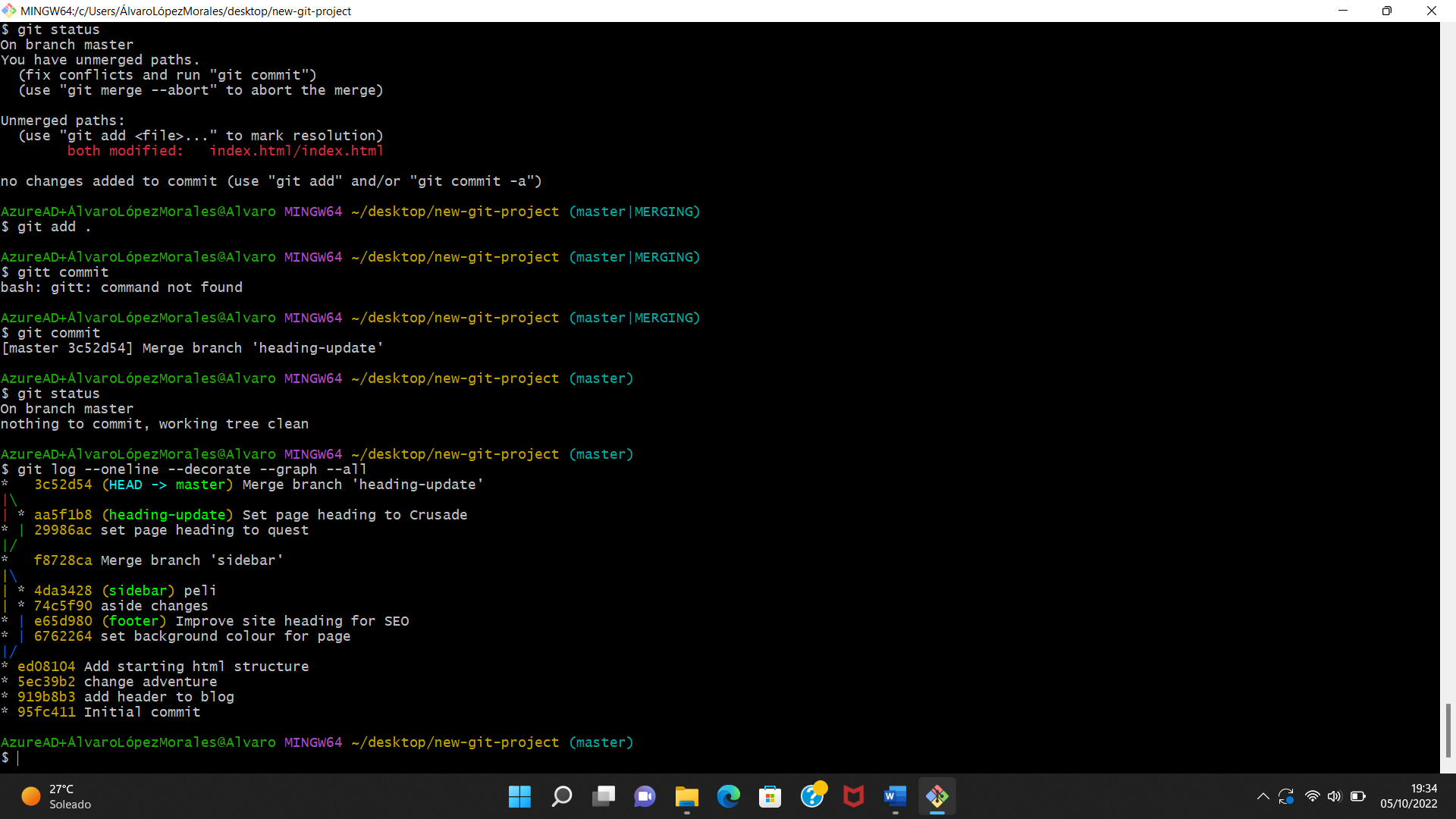
||||||| merged common ancestors everything below this line (until the next indicator) shows you what the original lines were

======= is the end of the original lines, everything that follows (until the next indicator) is what's on the branch that's being merged in

>>>>>>> heading-update is the ending indicator of what's on the branch that's being merged in (in this case, the heading-update branch)

Para solucionar el conflicto de Merging, eliminamos todo lo que nos marca fallo del html. A continuación, cambiamos el header a Advneturous Quest y guardamos. Sólo nos queda añadirlo al staging index y comitearlo.

Tenéis que estar en este punto ahora:



Por último, vamos a hacer pruebas con los comandos git commit --amend, que sirve para alterar el commit más reciente, el comando git revert, que revierte cambios en el commit que queramos (poniendo el nº de commit) y el comando git reset, el cual elimina commits, a su vez cargándoselos del repositorio.

Con el directorio de trabajo limpio (recuerda git status) el comando git --amend te permite poner un nuevo mensaje de commit a nuestro commit más reciente. Por otro lado, este comando también permite incluir cambios o ficheros que se te hayan olvidado en el último commit.

Poniendo directamente el comando git commit--amend (tras haber comprobado que está todo OK mediante git status) se nos abrirá directamente VSCode con el mensaje que hemos introducido en nuestro último commit. Modificamos (yo he incluido una S más), guardamos y cerramos. Os debería aparecer lo siguiente:

Texto

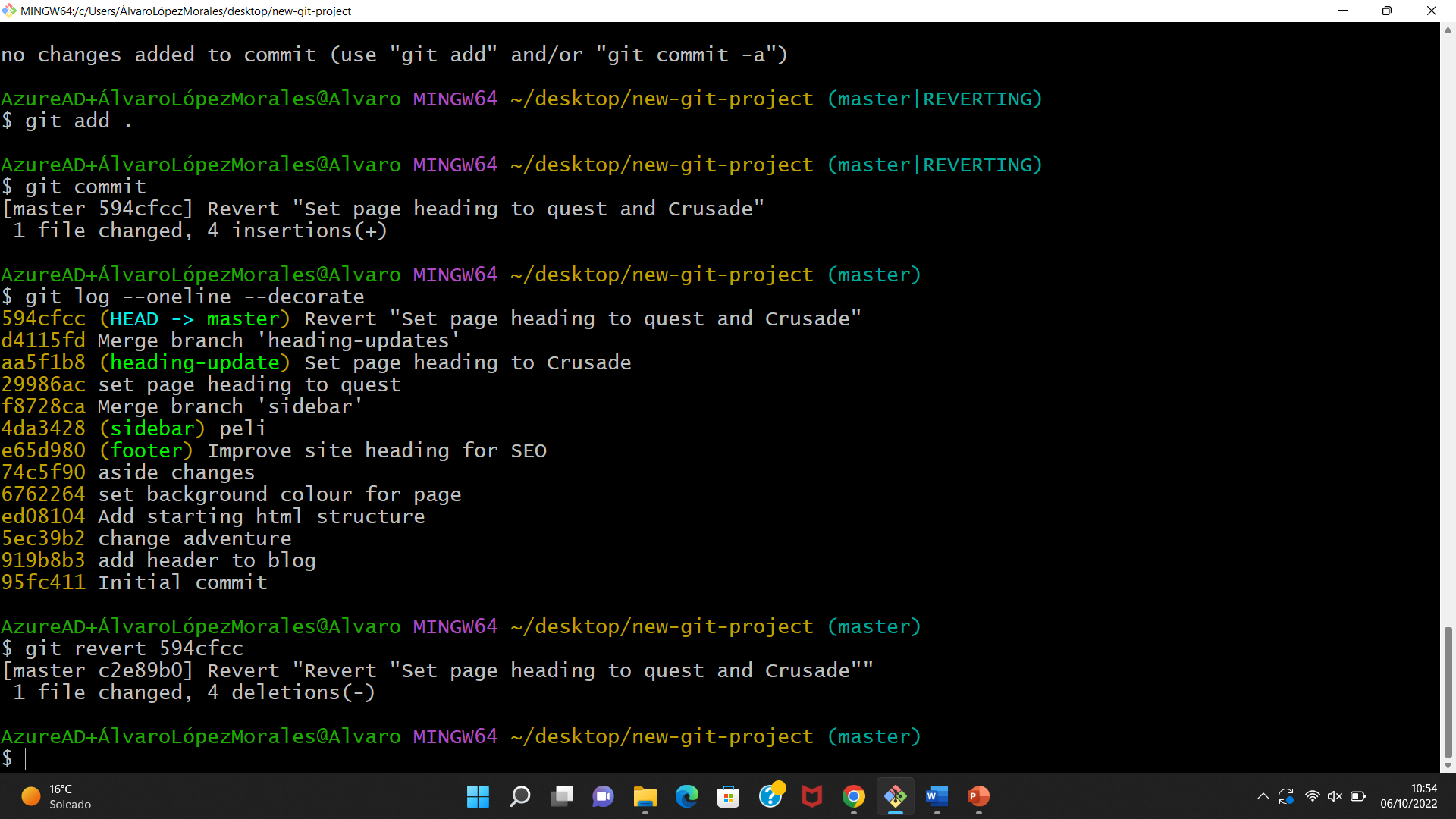
Descripción generada automáticamente

Ahora vamos a modificar el fichero html y subirlo al staging index. Posteriormente, introducimos el comando git commit --amend. Yo he cambiado el mensaje “adventurous Quest” por aventura. Haciendo esto, hemos modificado el fichero y lo hemos lanzado a nuestro repositorio sin hacer un n uevo commit, simplemente alterando el anterior.

Texto

Descripción generada automáticamente

Acto seguido vamos a probar el comando git revert (+número de sha). Modificamos el encabezado del html a “set page heading to quest and crusade”. Comiteamos y al usar el comando git revert +sha aparecerá lo siguiente:



Como vemos, Git ha revertido nuestro cambio hecho.

Vamos a ver ahora para qué sirve el comando git reset. Aparentemente, parece cercano a git revert, pero, la gran diferencia entre ellos es que git revert crea un nuevo commit que sustituye al anterior, mientras que el comando git reset ELIMINA commits. Ojo, elimina el commit de nuestro repositorio, así que hay que tener cuidado.

Por seguridad, Git guarda todo durante 30 días hasta que elimina algo por completo, así que en caso de que cometamos un error y eliminemos algo que no queremos, mediante el comando git reflog podríamos acceder al contenido eliminado.

Caracteres de GIT para referenciar commits antiguos:

the parent commit – the following indicate the parent commit of the current commit

HEAD^

HEAD~

HEAD~1

the grandparent commit – the following indicate the grandparent commit of the current commit

HEAD^^

HEAD~2

the great-grandparent commit – the following indicate the great-grandparent commit of the current commit

HEAD^^^

HEAD~3

La diferencia principal entre ^ y ~ (comando alt 126) es cuando un commit es creado por un merge. Un commit creado con un merge tiene 2 “padres”. En su caso, el comando ^ identifica al primer padre mientras que ^2 se refiere al segundo padre. El primer padre es la rama en la que estábamos cuando indicamos el comando git merge, mientras que el segundo padre es la rama con la que hemos hecho el merge. Vamos a verlo conm un ejemplo. Esta es la situación en la que nos tenemos que encontrar haciendo un git log:

Texto

Descripción generada automáticamente

HEAD ^ hace referencia al commit 594cfcc

HEAD ~1 es también el commit 594cfcc

HEAD^^ es el commit d4115fd

HEAD ~2 es también el commit d4115fd

HEAD ^^^ es el commit 29986ac

HEAD ~3 es también el commit 29986ac

HEAD^^^2 es el commit aa5f1b8

¿A qué commit hará referencia el comando HEAD ~6? 🡪 Hará referencia al commit 6762264

EL COMANDO GIT RESET (importante, este comando elimina commits)

Puede ser usado para:

* Mover el Head y el punto de la rama en la que estemos al commit referenciado
* Eliminar commits
* Mover cambios que hayan sido comiteados al staging index (volver atrás)
* Eliminar cambios comiteados del staging index

La forma en la Git interpreta cómo usamos el comando git reset es mediante el uso de --mixed, --soft, --hard.

Consejo 🡪 Antes de eliminar nada, crear una rama extra para guardar ahí todo, para poderlo recuperar en caso de que nos interese.

En nuestro caso, creamos una nueva rama llamada backup 🡪 git branch backup

Ahora, vamos a probar el comando git reset --mixed.

Introducimos el comando git reset --mixed HEAD^ 🡪 os tiene que aparecer lo siguiente:

Texto

Descripción generada automáticamente

Con el comando que he introducido se verá afectado el commit en el que está ubicado el HEAD, por lo que en mi caso será el commit con Sha c2e89b0.

Haciendo un git status debéis comprobar quer los cambios no se han stageado (Se han sacado del staging index:

Texto

Descripción generada automáticamente

A continuación, vamos a probar el comando git reset --soft HEAD^ Este comando cogerá los cambios hechos en el commit anterior y los moverá al staging index. Os tiene que aparecer esto:

Texto

Descripción generada automáticamenteLo interesante es la primera parte “changes to be commited”. La segunda parte aparece ahí porque no he utilizado la gama backup, por lo que arrastro el comando git reset --mixed.

Ahora probamos el comando git reset --hard HEAD^. Este seguirá afectando al mismo commit que los anteriores, puesto que no hemos movido el HEAD y seguimos en la rama Master:

Texto

Descripción generada automáticamente

Como veis, he restablecido los cambios anteriores (por lo que ya no me aparecen modificaciones en git status) y al hacer el git reset --hard HEAD^ lo que he hecho es eliminar los cambios hechos en el commit 594cfcc Revert “Set page heading to quest and crusade”, que, como veréis en la siguiente captura, ha desaparecido.

Texto

Descripción generada automáticamente