Clase 3 repaso git. taller

Escenario 1: Confirmación de archivos

Paso 1: Git Init

Para almacenar un directorio bajo control de versiones, necesita crear un repositorio. Con Git, inicializa un repositorio en el directorio de nivel superior para un proyecto.

Tarea

Como se trata de un proyecto nuevo, es necesario crear un nuevo repositorio. Utilice el `git init ↩` comando para crear un repositorio.

ProTip

Después de inicializar un repositorio, se crea un nuevo subdirectorio oculto llamado *.git* . Este subdirectorio contiene los metadatos que usa Git para almacenar su información. Si está interesado en los detalles, utilice la línea de comando para explorar el contenido.

Paso 2: estado de Git

Cuando un directorio es parte de un repositorio, se denomina *directorio de trabajo* . Un directorio de trabajo contiene la última versión descargada del repositorio junto con los cambios que aún no se han confirmado. Mientras trabaja en un proyecto, todos los cambios se realizan en este directorio de trabajo.

Puede ver qué archivos han cambiado entre su directorio de trabajo y lo que se ha enviado previamente al repositorio mediante el comando `git status ↩`.

El resultado de este comando se denomina "estado del árbol de trabajo".

Protip

Git "no rastrea" todos los archivos hasta que se indique lo contrario. Los detalles de cómo se tratan en el siguiente paso.

Paso 3 - Git Add

Para guardar o confirmar archivos en su repositorio de Git, primero debe agregarlos al *área de prueba* . Git tiene tres áreas, un directorio de trabajo, un área de preparación y el repositorio en sí. Los usuarios mueven, también conocido como promover, los cambios del directorio de trabajo a un área de preparación antes de enviarlos al repositorio.

Uno de los enfoques clave con Git es que las confirmaciones están enfocadas, son pequeñas y frecuentes. El área de preparación ayuda a mantener este flujo de trabajo al permitirle promover solo ciertos archivos a la vez en lugar de todos los cambios en su directorio de trabajo.

Tarea

Use el comando `git add <file|directory>` para agregar *hello-world.js* al área de preparación.

Si realiza un cambio adicional después de agregar un archivo al área de preparación, el cambio no se reflejará hasta que agregue el archivo nuevamente.

Protip

Como se describe en el Paso 2, el `git status ↩` comando le permite ver el estado tanto del directorio de trabajo como del área de preparación en cualquier momento.

Paso 4 - Compromiso de Git

Una vez que se ha agregado un archivo al área de preparación, se debe enviar al repositorio. El comando `git commit -m "commit message"` mueve los archivos de la etapa de pruebas al repositorio y registra la hora / fecha, el autor y un mensaje de confirmación que se puede usar para agregar contexto y razonamiento adicionales a los cambios, como un número de informe de error.

Solo se confirmarán los cambios agregados al área de preparación, no se incluirán los archivos en el directorio de trabajo que no se hayan preparado.

Tarea

Úselo `git commit -m "<commit message>"` para confirmar el archivo en etapas.

Protip

A cada confirmación se le asigna un hash SHA-1 que le permite volver a consultar la confirmación en otros comandos.

Paso 5: Ignorar Git

A veces, hay archivos o directorios particulares que nunca desea confirmar, como la configuración de desarrollo local. Para ignorar estos archivos, crea un archivo *.gitignore* en la raíz del repositorio.

El archivo *.gitignore le* permite definir comodines para los archivos que desea ignorar, por ejemplo * .tmp ignorará todos los archivos con la extensión .tmp.

Cualquier archivo que coincida con un comodín definido no se mostrará en una `git status` salida y se ignorará al intentar el `git add` comando.

Tarea

Agregue y *confirme* un archivo *.gitignore* en el repositorio para ignorar todos los archivos * .tmp.

Protip

El `.gitignore` debe estar comprometido con el repositorio para garantizar que las reglas se apliquen en diferentes máquinas.

Escenario 2: Confirmación de cambios

Paso 1: estado de Git

Como se discutió en el escenario anterior, `git status` nos permite ver los cambios en el directorio de trabajo y el área de preparación en comparación con el repositorio.

Dado que el repositorio actual se está ejecutando, se `git status ↵` muestra que se ha realizado un cambio en nuestro directorio de trabajo a un archivo previamente comprometido, commit.js, pero aún no se ha movido al área de prueba.

Paso 2 - Git Diff

El comando le `git diff ↩` permite comparar los cambios en el directorio de trabajo con una versión previamente confirmada. De forma predeterminada, el comando compara el directorio de trabajo y la confirmación *HEAD* .

Si desea comparar con una versión anterior, proporcione el hash de confirmación como parámetro, por ejemplo `git diff <commit>` . La comparación con las confirmaciones generará los cambios para todos los archivos modificados. Si desea comparar los cambios en un solo archivo, proporcione el nombre como un argumento como `git diff committed.js` .

Protip

De forma predeterminada, la salida está en formato de diferencias combinadas. El comando `git difftool` cargará una herramienta externa de su elección para ver las diferencias.

Paso 3 - Git Add

Al igual que en el escenario anterior, para realizar un cambio, primero se debe realizar mediante `git add` comando.

Tarea

Para proceder en la primera etapa, sus cambios usando `git add`

Protip

Si cambia el nombre o elimina archivos, debe especificar estos archivos en el comando agregar para que se rastreen. Alternativas que puede usar `git mv` y `git rm` para que git realice la acción e incluyen la actualización del área de ensayo.

Paso 4: diferencias por etapas

Una vez que los cambios estén en el área de preparación, no se mostrarán en la salida de `git diff ↩`. De forma predeterminada, `git diff` solo comparará el directorio de trabajo y no el área de preparación.

Para comparar los cambios en el área de ensayo con la confirmación anterior, proporcione el parámetro de ensayo `git diff --staged ↩`. Esto le permite asegurarse de haber realizado correctamente todos sus cambios.

Paso 5 - Registro de Git

El comando le `git log ↩` permite ver el historial del repositorio y el registro de confirmación.

Protip

El formato de la salida del registro es muy flexible. Por ejemplo, para generar cada confirmación en una sola línea, el comando es `git log --pretty=format:"%h %an %ar - %s" ↩`. Se pueden encontrar más detalles en la página de manual de git log a la que se accede usando `git log --help`

Paso 6 - Git Show

Si bien git log le dice el autor y el mensaje de la confirmación, para ver los cambios realizados en la confirmación, debe usar el comando `git show ↩`

Al igual que con otros comandos, por defecto mostrará los cambios en la confirmación HEAD. Úselo `git show <commit-hash>` para ver cambios anteriores.

Escenario 3: trabajar de forma remota

Paso 1 - Git Remote

Los repositorios remotos le permiten compartir cambios desde o hacia su repositorio. Las ubicaciones remotas son generalmente un servidor de compilación, una máquina de miembros del equipo o una tienda centralizada como Github.com. Los controles remotos se agregan usando el `git remote` comando con un nombre *descriptivo* y la ubicación remota, generalmente una URL HTTPS o una conexión SSH, por ejemplo, *https://github.com/OcelotUproar/ocelite.git* o *git@github.com : / OcelotUproar / ocelite. git* .

El nombre descriptivo le permite hacer referencia a la ubicación en otros comandos. Su repositorio local puede hacer referencia a varios repositorios remotos diferentes según su escenario.

Tarea

Este entorno tiene una ubicación de repositorio remoto de */ s / remote-project / 1* . Usando `git remote` , agregue esta ubicación remota con el *origen del* nombre .

Protip

Si usa `git clone` , discutido en un escenario futuro, entonces la ubicación desde la que está clonando se agregará automáticamente como un control remoto con el *origen del* nombre .

Paso 2 - Git Push

Cuando esté listo para compartir sus confirmaciones, debe *enviarlas* a un repositorio remoto a través de `git push`. Un flujo de trabajo típico de Git sería realizar varias confirmaciones pequeñas a medida que completa una tarea y la envía a un control remoto en puntos relevantes, como cuando la tarea está completa, para garantizar la sincronización del código dentro del equipo.

El `git push` comando va seguido de dos parámetros. El primer parámetro es el nombre descriptivo del repositorio remoto que definimos en el primer paso. El segundo parámetro es el nombre de la rama. De forma predeterminada, todos los repositorios de git tienen una rama *maestra* en la que se trabaja el código.

Tarea

Empuje las confirmaciones en la rama maestra al origen remoto.

Paso 3 - Git Pull

Donde le `git push` permite enviar sus cambios a un repositorio remoto, `git pull` funciona a la inversa. `git pull` le permite sincronizar los cambios de un repositorio remoto en su versión local.

Los cambios del repositorio remoto se fusionan automáticamente en la rama en la que está trabajando actualmente.

Tarea

Extraiga los cambios del control remoto a su rama maestra.

En el siguiente paso, exploraremos qué cambios se han realizado.

Paso 4 - Registro de Git

Como se describe en el escenario anterior, puede usar el `git log ↩` comando para ver el historial del repositorio. El `git show ↩` comando le permitirá ver los cambios realizados en cada confirmación.

En este ejemplo, el resultado de `git log ↩` muestra una nueva confirmación de " DifferentUser@JoinScrapbook.com " con el mensaje "Fix for Bug # 1234". La salida de `git show ↩` resalta las nuevas líneas agregadas al archivo en verde.

Protip

Use el comando `git log --grep="#1234" ↩` para encontrar todas las confirmaciones que contienen # 1234

Paso 5 - Git Fetch

El comando `git pull` es una combinación de dos comandos diferentes `git fetch` y `git merge`. Fetch descarga los cambios del repositorio remoto en una rama separada llamada *remotes / <remote-name> / <remote-branch-name>* . Se puede acceder a la sucursal usando `git checkout` .

El uso `git fetch` es una excelente manera de revisar los cambios sin afectar su rama actual. El formato de nomenclatura de las sucursales es lo suficientemente flexible como para que pueda tener varios controles remotos y sucursales con el mismo nombre y cambiar fácilmente entre ellos.

El siguiente comando fusionará los cambios recuperados en master.

```
git merge remotes/<remote-name>/<remote-branch-name> master
```

Cubriremos la fusión con más detalle en un escenario futuro.

Tarea

Se han realizado cambios adicionales en el repositorio de *origen* . Use `git fetch` para descargar los cambios y luego verifique la rama para verlos.

Protip

Puede ver una lista de todas las ramas remotas usando el comando `git branch -r ↩`

Escenario 4: deshacer cambios

Paso 1: Git Checkout

Cuando se trabaja con Git, un escenario común es deshacer los cambios en su directorio de trabajo. El comando `git checkout` reemplazará todo en el directorio de trabajo a la última versión confirmada.

Si desea reemplazar todos los archivos, utilice un punto (.) Para indicar el directorio actual; de lo contrario, una lista de los directorios / archivos separados por espacios.

Tarea

Úselo `git checkout` para borrar cualquier cambio en el directorio de trabajo.

Paso 2 - Restablecimiento de Git

Si está en medio de una confirmación y ha agregado archivos al área de preparación, pero luego cambió de opinión, deberá usar el `git reset` comando. `git reset` moverá los archivos desde el área de preparación al directorio de trabajo. Si desea restablecer todos los archivos, utilice un . para indicar el directorio actual; de lo contrario, enumere los archivos separados por espacios.

Esto es muy útil cuando intenta mantener sus confirmaciones pequeñas y enfocadas, ya que puede mover archivos fuera del área de preparación si ha agregado demasiados.

Tarea

Mueva los cambios de la etapa de pruebas al directorio de trabajo usando `git reset`

Paso 3 - Git Reset Hard

A `git reset --hard` combinará git reset y git checkout en un solo comando. El resultado serán los archivos eliminados del área de preparación y el directorio de trabajo volverá al estado de la última confirmación.

Tarea

Elimine los cambios tanto del área de ensayo como del directorio de trabajo usando `git reset`

Protip

El uso `HEAD` borrará el estado de la última confirmación, el uso le `git reset --hard <commit-hash>` permite volver a cualquier estado de confirmación. Recuerde, HEAD es un alias para el último hash de confirmación de la rama.

Paso 4 - Git Revert

Si ya ha confirmado archivos, pero se dio cuenta de que cometió un error, el comando le `git revert` permite deshacer las confirmaciones. El comando creará una nueva confirmación que tiene el efecto inverso de la reversión de la confirmación.

Si no ha introducido sus cambios, `git reset HEAD~1` tendrá el mismo efecto y eliminará la última confirmación.

Tarea

Úselo `git revert` para revertir los cambios en la última confirmación.

Tenga en cuenta que esto abrirá una sesión del editor de Vim para crear un mensaje de confirmación para cada confirmación. Para guardar el mensaje de confirmación y salir de vim, escriba el comando *:wq* para cada sesión de Vim.

Protip

La motivación detrás de la creación de nuevas confirmaciones es que reescribir el historial en Git es un anti-patrón. Si ha enviado sus confirmaciones, debe crear nuevas confirmaciones para deshacer los cambios, ya que otras personas podrían haber realizado confirmaciones mientras tanto.

Paso 5 - Git Revert

Para revertir múltiples confirmaciones a la vez, usamos el carácter ~ para significar menos. Por ejemplo, HEAD ~ 2 son dos confirmaciones de la cabeza. Esto se puede combinar con los personajes ... para decir entre dos confirmaciones.

Tarea

Utilice el comando `git revert HEAD...HEAD~2 ↩` para revertir las confirmaciones entre HEAD y HEAD ~ 2.

Protip

Puede utilizar el comando `git log --oneline ↩` para obtener una descripción general rápida del historial de confirmaciones.

Escenario 5: solución de conflictos de fusión

Paso 1 - Git Merge

El `git fetch` comando descarga los cambios en una rama separada que se puede verificar y fusionar. Durante una fusión, Git intentará combinar automáticamente las confirmaciones.

Cuando no existan conflictos, la fusión se 'adelantará rápidamente' y no tendrá que hacer nada. Si existe un conflicto, recuperará un error y el repositorio estará en un estado de fusión.

Tarea

En su entorno, se han obtenido los cambios de un repositorio remoto.

Ahora necesita fusionar los cambios de origen / maestro.

Esto resultará en un conflicto de fusión. El conflicto indica que la fusión falló porque ambos repositorios agregaron el archivo. Resolveremos esto en los siguientes pasos siguientes.

Protip

Al mantener las confirmaciones pequeñas y enfocadas, reduce la probabilidad de un conflicto de fusión.

El comando `git pull` es una combinación de `fetch` y `merge` .

Paso 2: Visualización del conflicto

Cuando ocurre un conflicto, los cambios tanto del local como del remoto aparecerán en el mismo archivo en formato Unix diff. Este es el mismo formato que usa *git diff* .

Para leer el formato, los cambios locales aparecerán en la parte superior entre <<<<<<< *HEAD* y ======= con los cambios remotos debajo entre ======= y >>>>> >> *mandos a distancia / origen / maestro* .

Para resolver el conflicto, los archivos deben editarse para que coincidan con nuestro estado final deseado. Demostraremos esto en el siguiente paso.

Protip

Git admite diferentes herramientas de combinación visual y de línea de comandos para facilitar la resolución de conflictos. El comando `git mergetool` lanzará una herramienta externa, somos grandes fanáticos dekdiff3 .

Paso 3: resolución de conflictos

La forma más sencilla de solucionar un conflicto es elegir la versión local o remota usando `git checkout --ours staging.txt` o `git checkout --theirs staging.txt ↩`. Si necesita tener más control, puede editar manualmente los archivos como de costumbre.

Una vez que los archivos están en el estado deseado, ya sea manualmente o usando git checkout, entonces necesita preparar y confirmar los cambios. Al confirmar, se creará un mensaje de confirmación predeterminado con detalles de la combinación y qué archivos están en conflicto.

Tarea

Resolver el conflicto seleccionando los cambios remotos y completar la combinación usando `git add` seguido de `git commit`.

Protip

Si desea revertir en medio de una fusión e intentarlo de nuevo, utilice el comando `git reset --hard HEAD;` para volver a su estado anterior.

Úselo `git commit --no-edit` cuando desee usar el mensaje de confirmación predeterminado.

Paso 4: avance no rápido

Para simular una combinación de avance no rápido, se ha producido lo siguiente.

1) El desarrollador A extrae los últimos cambios del desarrollador B.
2) El desarrollador B confirma los cambios en su repositorio local.
3) El desarrollador A confirma cambios no conflictivos en su repositorio local.
4) El desarrollador A extrae los últimos cambios del desarrollador B.

En este escenario, Git no puede adelantar los cambios del Desarrollador B porque el Desarrollador A ha realizado varios cambios.

Cuando esto suceda, Git intentará fusionar automáticamente los cambios. Si no existen conflictos, la fusión se completará y se creará una nueva confirmación para indicar que la fusión está ocurriendo en ese momento.

El mensaje de confirmación predeterminado para fusiones es "Fusionar rama"' de Estas confirmaciones pueden ser útiles para indicar puntos de sincronización entre repositorios, pero también producen un registro de confirmaciones ruidoso. En el siguiente paso, investigaremos enfoques alternativos.

Tarea

Extraiga los cambios del repositorio remoto y use el mensaje de confirmación predeterminado con el comando a continuación.

```
git pull --no-edit origin master ↩
```

Puedes ver las confirmaciones con `git log --all --decorate --oneline ↩`

Paso 5 - Git Rebase

Los mensajes de confirmación de fusión pueden ser útiles para indicar puntos de sincronización, pero también pueden producir mucho ruido. Por ejemplo, si está trabajando contra sucursales locales y no ha presionado, entonces esta información adicional no tiene sentido y es confusa para otros desarrolladores que buscan en el repositorio.

Para resolver esto, puede usar en `git rebase` lugar de `git merge` . Una rebase desenrollará los cambios que ha realizado y reproducirá los cambios en la rama, aplicando sus cambios como si hubieran ocurrido todos en la misma rama. El resultado es un historial y un gráfico limpios para la fusión.

Importante Como `rebase` reproducirá los cambios en lugar de fusionarse, cada confirmación tendrá un nuevo ID de hash. Si usted u otros desarrolladores han empujado / retirado el repositorio, cambiar el historial puede perder las confirmaciones. Como tal, no debe reajustar las confirmaciones que se han hecho públicas, por ejemplo, presionar las confirmaciones y luego reajustar las confirmaciones anteriores de una rama diferente. El resultado serán confirmaciones previamente públicas con diferentes ID de hash. Se pueden encontrar más detalles enLos peligros de volver a basarse .

Paso 6: Cambio de base de las solicitudes de extracción

Este enfoque también se aplica cuando se trabaja con sucursales remotas y se puede aplicar al emitir una solicitud de extracción mediante:

```
git pull --rebase
```

Esto actuará como si hubiera realizado una solicitud de extracción antes de cada una de sus confirmaciones.

Escenario 6: experimentos con ramas

Paso 1 - Rama de Git

Las ramas se crean en base a otra rama, generalmente maestra. El comando `git branch <new branch name> <starting branch>` toma una rama existente y crea una rama separada para trabajar. En este punto, ambas ramas son idénticas.

Para cambiar a una rama, usa el `git checkout <new branch name>` comando.

Tarea

Cree y realice el pago de una nueva rama llamada 'new_branch'

Protip

El comando `git checkout -b <new branch name>` creará y comprobará la rama recién creada.

Paso 2: enumerar las sucursales

Para listar todas las ramas use el comando `git branch ↩`.

El argumento adicional `-a` incluirá ramas remotas mientras que la inclusión `-v` incluirá el mensaje de confirmación HEAD para la rama

Tarea

Enumere todas las ramas con su último mensaje de confirmación usando `git branch -va ↩`

Paso 3: fusionar con el maestro

Se ha realizado un compromiso con la nueva rama. Para fusionar esto en *master* , primero necesitaría verificar la rama de destino, en este caso master, y luego usar el comando 'git merge' para fusionar las confirmaciones de una rama.

Tarea

Fusiona las confirmaciones en tu nueva rama de nuevo en master.

Paso 4 - Empujar ramas

Como hemos discutido en cursos anteriores, si desea enviar una rama a un control remoto, use el comando `git push <remote_name> <branch_name>`

Paso 5 - Limpiar ramas

Limpiar las ramas es importante para eliminar la cantidad de ruido y confusión. Para eliminar una rama, debe proporcionar el argumento `-d` , por ejemplo `git branch -d <branch_name>`

Tarea

Ahora que la rama se ha fusionado en master, ya no es necesaria. Elimine su nueva rama para mantener su repositorio limpio y comprensible.

Escenario 7: búsqueda de errores

Paso 1 de 4

Git difiere dos confirmaciones

El `git diff` comando es el más simple para comparar lo que ha cambiado entre confirmaciones. Dará salida a las diferencias entre las dos confirmaciones.

Ejemplo

Puede visualmente dos confirmaciones proporcionando las dos confirmaciones hash-ids o punteros (blobs)

```
git diff HEAD~2 HEAD ↩
```

Registro de Git

Si bien le `git log` ayuda a ver los mensajes de confirmación, por defecto no muestra lo que realmente cambió. Afortunadamente, el comando es extremadamente flexible y las opciones adicionales brindan información útil sobre el historial del repositorio.

Ejemplos de

Para ver la descripción general de las confirmaciones en una vista corta, use el comando `git log --oneline ↩`

Para generar la información de confirmación con las diferencias de lo que cambió, debe incluir el indicador *-p* como `git log -p ↩`

Esto generará el historial completo. Puede filtrarlo con varias opciones diferentes. El *-n <número>* especifica un límite de confirmaciones para mostrar desde HEAD. Por ejemplo, `git log -p -n 2 ↩` muestra HEAD y HEAD ~ 1.

Si conoce el período de tiempo, puede usar un período de tiempo entre confirmaciones antes de una fecha en particular usando *--since = "hace 2 semanas"* y _-- hasta = "hace 1 día".

Como con la mayoría de los comandos de Git, podemos generar un rango de confirmaciones usando `HEAD...HEAD~1` como se muestra en la terminal.

El uso del comando `git log --grep="Initial" ↩` generará todas las confirmaciones que incluyan la palabra "Inicial" en su mensaje de confirmación. Esto es útil si etiqueta confirmaciones con números de seguimiento de errores.

Git bisect toma una serie de pasos, ejecute los pasos para ver los resultados.

Pasos

1. Para entrar en el modo bisecto, usa el comando `git bisect start ↩`.
2. Una vez en el modo bisecto, define su pago actual como mal uso `git bisect bad ↩`. Esto indica que contiene el problema que está buscando para ver cuándo se introdujo.
3. Hemos definido dónde ocurrió una mala confirmación, ahora necesitamos definir cuándo se utilizó la última confirmación buena conocida `git bisect good HEAD~5 ↩`. En este caso, fue hace cinco confirmaciones.
4. El paso 3 verificará la confirmación entre las malas y las buenas. Luego puede verificar la confirmación, ejecutar pruebas, etc. para ver si existe el error. En este ejemplo, puede verificar el contenido usando `cat list.html ↩`
5. Esta confirmación se ve bien ya que todo tiene etiquetas HTML correctas. Le decimos a Git que estamos felices de usar `git bisect good ↩`. Esto verificará automáticamente la confirmación en medio de la última confirmación buena conocida, como se define en el paso 5 y nuestra confirmación incorrecta.
6. Como hicimos antes, debemos verificar si la confirmación es buena o mala. `cat list.html ↩`
7. A esta confirmación le faltan etiquetas HTML. El uso `git bisect bad ↩` finalizará la búsqueda y generará el ID de confirmación relacionado.

El resultado es que en lugar de buscar cinco confirmaciones, solo buscamos dos. En una escala de tiempo mucho mayor, la *bisectriz* puede ahorrarle un tiempo significativo.

Git Blame

Si bien no es deseable tener una cultura de "culpa", puede ser útil saber quién trabajó en ciertas secciones del archivo para ayudar con las mejoras en el futuro. Aquí es donde `git blame` puede ayudar.

`git blame <file>` muestra la revisión y el autor que modificó por última vez cada línea de un archivo.

Ejemplo

Ejecutar la culpa en un archivo dará como resultado quién tocó por última vez cada línea.

```
git blame list.html ↩
```

Si conocemos las líneas que nos interesan, entonces podemos usar el parámetro *-L* para proporcionar un rango de líneas a la salida.

```
git blame -L 6,8 list.html ↩
```

Escenario 8: ser exigente con Git

Paso 1 - Recolección de cerezas

En este escenario, tenemos una serie de confirmaciones en la *rama* _new que ha creado dos archivos html. En este escenario, solo nos preocupan los cambios en uno de los archivos, pero si fusionamos la rama, fusionaríamos las cinco confirmaciones y los cambios no deseados.

Para fusionar confirmaciones individuales usamos el `git cherry-pick <hash-id|ref>` comando. Esto se comporta de manera similar a la *fusión* , si no existen conflictos, la confirmación se fusionará automáticamente.

Tarea

Siga los pasos a continuación para fusionar las tres confirmaciones que nos interesan. Queremos reproducir las confirmaciones en el orden en que ocurrieron en el repositorio original.

1. Elija la confirmación con el mensaje "Confirmación inicial, sin elementos"
2. Selecciona la confirmación con el mensaje "Lista inicial"
3. Elija la confirmación con el mensaje "Agregar elementos finales a la lista"

Protip

Hemos usado HEAD anteriormente para indicar la punta de la rama actual. Puede hacer referencia a través de ramas usando la sintaxis _~ # _. Por ejemplo, se `new_branch~3` refiere a la penúltima confirmación en la rama, en este caso tiene el mensaje de confirmación "Confirmación inicial, sin elementos"

Paso 2: resolución del conflicto de recolección de cerezas

De la misma manera, la fusión puede generar conflictos, al igual que la selección selectiva. Resuelve los conflictos de la misma manera que al fusionar una rama, ya sea arreglando manualmente los archivos o seleccionando los *suyos* o los *nuestros a* través de `git checkout` .

Si siente que ha cometido un error, puede detener y revertir la selección utilizando `git cherry-pick --abort`

Tarea

Elija la creación de la segunda lista usando `git cherry-pick new_branch~1 ↩`

Esto resultará en un conflicto de fusión. Resuelve el conflicto usando `git checkout` y selecciona la confirmación elegida.

Continúe cuando esté listo donde completaremos la selección de cerezas.

Paso 3 - Continuación de la recolección de cerezas después de un conflicto

Una vez que se hayan resuelto los conflictos, puede continuar con la selección de cereza usando el comando `git cherry-pick --continue` .

De manera similar al uso de la *combinación* , la resolución de una selección selectiva resultará en una confirmación.

Tarea

Complete la selección de cereza agregando primero el elemento previamente en conflicto y luego usando la opción *--continuar* .

```
git add list2.html ↩
```

```
git cherry-pick --continue ↩
```

En este punto, aparecerá el editor predeterminado, en este caso vim, que le permitirá editar el mensaje de confirmación seleccionado para incluir detalles del conflicto y cómo se resolvió. Para guardar y salir de vim, escriba *: wq*

Escenario 9: reescritura del historial

Paso 1 de 4

Modificación de mensajes de confirmación

La reescritura del historial de repositorios se realiza usando `git rebase -interactive`. Al poner rebase en modo interactivo, tiene más control sobre los cambios que desea realizar. Después de iniciar en modo interactivo, se le dan seis comandos para realizar en cada confirmación en el repositorio. Al usar el editor que se abre, de forma predeterminada Vim, define qué acciones desea realizar en cada confirmación.

En este ejemplo queremos cambiar el compromiso. Para ponerlo en este estado, necesitamos cambiar la palabra "seleccionar" al lado de la confirmación para que coincida con la acción que desea realizar según la lista que se muestra en la ventana de Vim, en este caso "reescribir".

En este ejemplo queremos cambiar el mensaje de confirmación.

Comienzo

Para comenzar, debemos ingresar al modo Interactive Rebase usando `git rebase --interactive --root ↩`

Seleccione el modo interactivo

Para comenzar con Vim puede ser un poco confuso, para editar texto primero debe escribir, lo `i` que lo pondrá en "modo de inserción".

Queremos editar el error tipográfico "comit" en el primer mensaje de confirmación "Comit inicial de la lista". Para el compromiso, cambie la palabra "pick" para que coincida con el comando que queremos realizar en el compromiso, en este caso "reescribir".

Para guardar y salir presione la `esc` tecla y luego `:wq` . Esto abrirá otra ventana del editor de Vim.

Cambio de mensaje

Nuevamente usando Vim, edite el mensaje de confirmación para cambiar "comit" a "confirmar". Después de guardar y salir de Vim, verá la salida de Git cambiando la confirmación. Úselo `git log --oneline ↩` para ver el mensaje de confirmación actualizado.

Protip

El argumento *--root le* permite reajustar todas las confirmaciones en el repositorio, incluida la primera confirmación.

Una alternativa más rápida para cambiar el último mensaje de confirmación es usar `git commit --amend ↩` y realizar el cambio usando Vim.

Squash se compromete

Se ha realizado una serie de 8 confirmaciones diferentes en su entorno local. En ese momento, estos compromisos tenían sentido, sin embargo, ahora deben ser solo un compromiso. Usando Rebase necesitamos *aplastar* las confirmaciones juntas.

Usando `git rebase --interactive HEAD~8`, tenemos el rango de confirmaciones desde HEAD hasta el último 0. Para aplastar necesitamos una confirmación base en la que todo se aplastará. Como tal, deje la primera confirmación como "pick" pero cambie el resto a "squash".

Después de guardar, tendrá la oportunidad de editar. De forma predeterminada, el mensaje de confirmación de Git será una combinación de los mensajes de confirmación previamente aplastados.

Tarea

Cuando ingresamos a Interactive Rebase podemos especificar que queremos modificar las 8 confirmaciones anteriores usando `git rebase --interactive HEAD~8 ↩`

En la etapa anterior usamos *reescribir* . Aquí queremos usar *calabaza* . Queremos aplastar 8 confirmaciones en una, si etiquetamos todas las confirmaciones como aplastadas, obtendríamos el error "No se puede 'aplastar' sin una confirmación previa", ya que no hay una confirmación base para aplastar todo.

Para aplastar las confirmaciones, debemos dejar la primera confirmación como nuestra base y etiquetar el siguiente 7 con *squash* .

Confirmar mensaje

Al guardar y salir de Vim, se nos muestra una nueva ventana de Vim que enumera una combinación de los 8 mensajes de confirmación en la rebase.

Después de guardar el mensaje de confirmación, se modificará el historial. Puedes ver esto usando `git log --oneline ↩`

Reordenar confirmaciones

Reordenar las confirmaciones puede ayudar a construir una mejor imagen del orden lógico en el que se completó el trabajo.

Tarea

Queremos reordenar nuestras dos últimas confirmaciones. Usar *HEAD ~ 2* nos permite modificarlos.

```
git rebase --interactive HEAD~2 ↩
```

Usando Vim, simplemente reordene las líneas, guarde y salga, y las confirmaciones coincidirán con el orden.

Compromiso dividido

Al igual que con el aplastamiento de las confirmaciones, a veces es útil dividir los cambios de las confirmaciones para mantenerlos enfocados y facilitar la selección o la reversión.

Dividir confirmaciones es un proceso de dos etapas. Primero necesitamos definir qué compromiso queremos dividir y luego necesitamos definir cómo queremos que se vean los nuevos compromisos.

Definición de compromiso para dividir

Aquí queremos dividir la confirmación anterior. Entramos en modo rebase usando `git rebase --interactive HEAD~1 ↩`

Al igual que con el rebase anterior, necesitamos cambiar la tarea para *editar*

Ahora estamos en un estado de edición interactiva del historial. Git registrará todos los cambios y el resultado final se aplicará al repositorio.

Dividir confirmaciones

Después de definir que queremos *editar* el compromiso, ahora estamos en un estado que nos permite cambiar el historial.

1. Como queremos dividir una confirmación existente, primero debemos eliminarla usando `git reset HEAD^ ↩`.
2. La confirmación se ha eliminado pero los archivos aún existen. Ahora podemos realizar las confirmaciones como deseamos anteriormente, como dos acciones separadas.

Ejecuta los comandos:

```
git add file3.txt ↩
git commit -m "File 3" ↩
git add file4.txt ↩
git commit -m "File 4" ↩
```

Ahorro

Una vez satisfecho con el estado del repositorio, le decimos a Git que continúe con la rebase y actualice el repositorio con *--continue* .

```
git rebase --continue ↩
```

Puedes ver la salida y las dos nuevas confirmaciones usando `git log --oneline ↩`

GRACIAS

Línea de atención al ciudadano: 018000 910270
Línea de atención al empresario: 018000 910682

@SENAcomunica

www.sena.edu.co