UNIVERSIDAD GALILEO

MAESTRIA INTELIGENCIA DE NEGOCIOS

CIENCIA DE DATOS EN PYTHON

SECCION L

# Investigación GIT

Osiel Gutierrez Herrera

Carne: 21000958

# Contenido

# ¿Cómo puedo comprobar el estado de un repositorio?

Cuando esté utilizando Git, con frecuencia querrá verificar el **estado** de su repositorio. Para ello, ejecute el comando `git status`, que muestra una lista de los archivos que se han modificado desde la última vez que se guardaron los cambios.

---

Usted ha sido puesto en el `dental`repositorio. Úselo `git status`para descubrir qué archivo(s) se han cambiado desde la última vez que se guardó. ¿Qué archivo(s) se enumeran?

Instrucciones

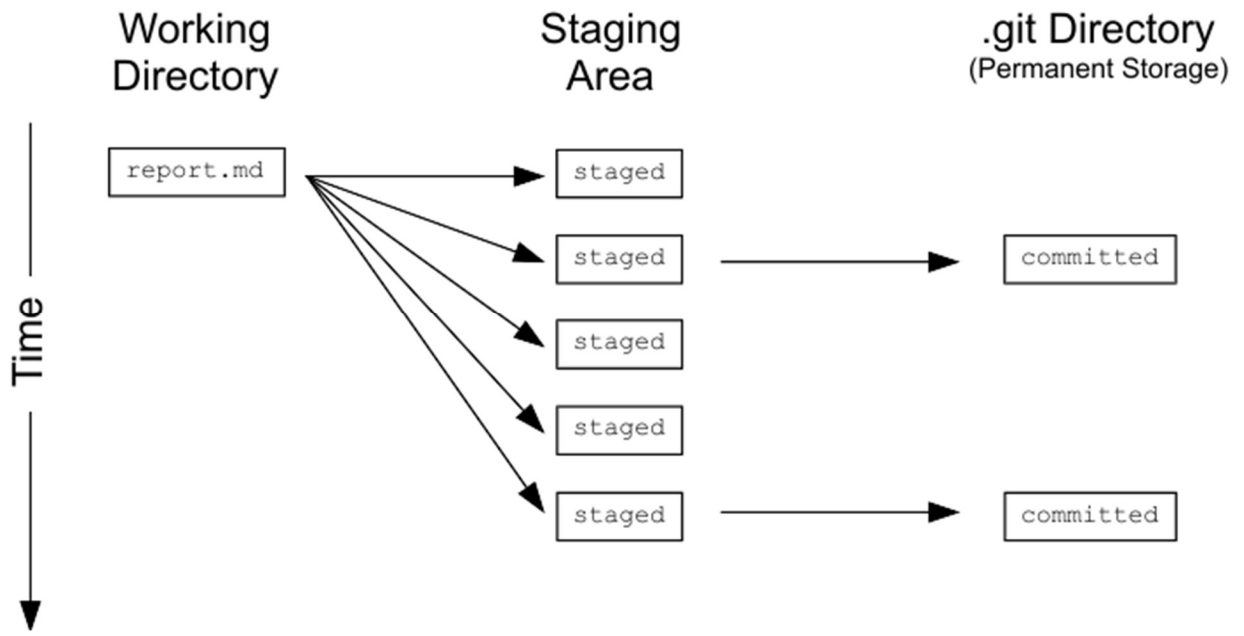
**50 PX**

Respuestas posibles

- `data/summer.csv`.
- `report.txt`.
- Ninguno de los anteriores.
- Los dos anteriores.

# ¿Cómo puedo saber lo que he cambiado?

Git tiene un **área** de preparación en la que almacena archivos con cambios que desea guardar y que aún no se han guardado. Poner archivos en el área de preparación es como poner cosas en una caja, mientras que **confirmar** esos cambios es como poner esa caja en el correo: puede agregar más cosas a la caja o sacar cosas con la frecuencia que desee, pero una vez que las coloca en el correo, no puede hacer más cambios.




`git status`le muestra qué archivos están en esta área de preparación y qué archivos tienen cambios que aún no se han colocado allí. Para comparar el archivo tal como está actualmente con lo que guardó por última vez, puede usar `git diff filename`. `git diff`sin ningún nombre de archivo le mostrará todos los cambios en su repositorio, mientras `git diff directory`que le mostrará los cambios en los archivos en algún directorio.

Instrucciones

**100 EXP**

Usted ha sido puesto en el `dental`repositorio. Úselo `git diff`para ver qué cambios se han realizado en los archivos.

# ¿Qué hay en una diferencia?

Una **diferencia** es una visualización formateada de las diferencias entre dos conjuntos de archivos. Git muestra diferencias como esta:

```
diff --git a/report.txt b/report.txt
index e713b17..4c0742a 100644
--- a/report.txt
+++ b/report.txt
@@ -1,4 +1,5 @@
-# Seasonal Dental Surgeries 2017-18
+# Seasonal Dental Surgeries (2017) 2017-18
+# TODO: write new summary
```

Esta espectáculos:

- El comando utilizado para producir la salida (en este caso, `diff --git`). En él, `a`y `b`son marcadores de posición que significan "la primera versión" y "la segunda versión".
- Una línea de índice que muestra claves en la base de datos interna de cambios de Git. Los exploraremos en el próximo capítulo.
- `--- a/report.txt`y `+++ b/report.txt`en el que las líneas que se *eliminan* tienen el prefijo `-`y las líneas que se agregan tienen el prefijo `+`.
- Una línea que comienza con `@@`eso indica *dónde* se están realizando los cambios. Los pares de números son `start line`y `number of lines`(en esa sección del archivo donde ocurrieron los cambios). Esta salida diferencial indica cambios que comienzan en la línea 1, con 5 líneas donde antes había 4.
- Una lista línea por línea de los cambios `-`mostrando las eliminaciones y `+`las adiciones (también hemos configurado Git para mostrar las eliminaciones en rojo y las adiciones en verde). Las líneas que *no han* cambiado a veces se muestran antes y después de las que sí lo han hecho para dar contexto; cuando aparecen, *no* tienen ni `+`ni `-`frente a ellos.

Las herramientas de programación de escritorio como **RStudio** pueden convertir diferencias como esta en una visualización de cambios lado a lado más legible; también puede usar herramientas independientes como **DiffMerge** o **WinMerge** .

---

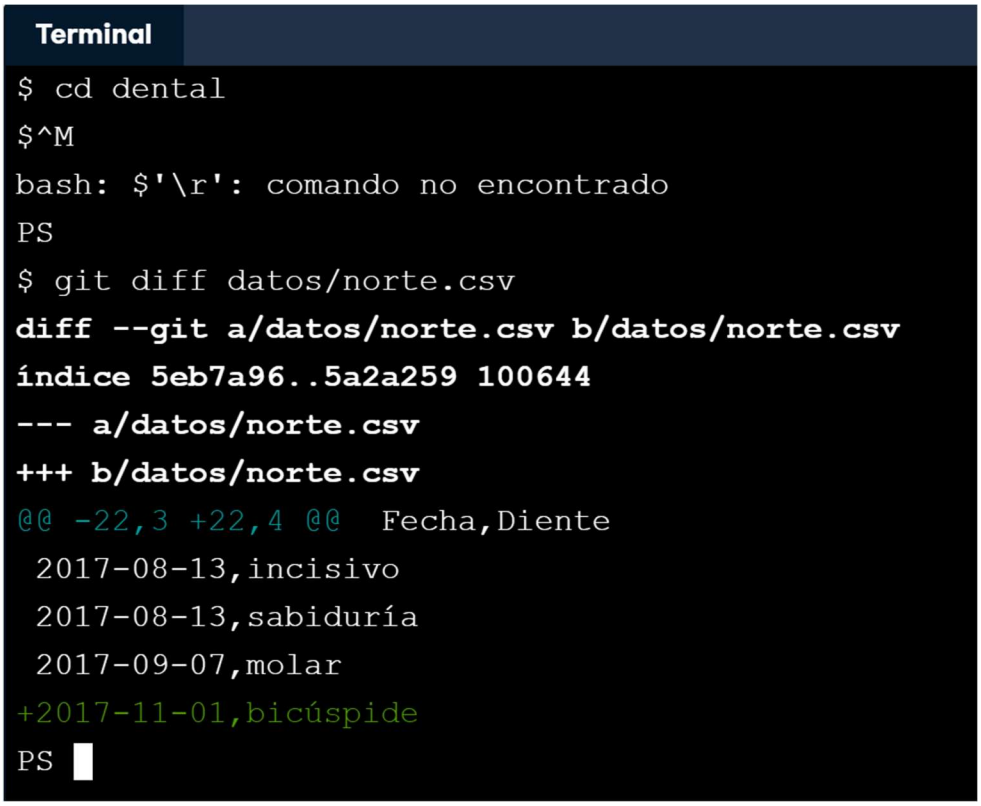
Usted ha sido puesto en el `dental`repositorio. Use `git diff data/northern.csv`para ver los cambios en ese archivo. ¿Cuántas líneas se han agregado o eliminado?

Instrucciones

**50 PX**

Respuestas posibles

- Ninguna.
- 1.
- 2.
- 20




Adicionado en verde.

# ¿Cuál es el primer paso para guardar los cambios?

Confirmas los cambios en un repositorio de Git en dos pasos:

1. Agregue uno o más archivos al área de ensayo.
2. Comprometer todo en el área de preparación.

Para agregar un archivo al área de preparación, use `git add filename`.

Instrucciones1/2

**50 PX**

- Usted ha sido puesto en el `dental`repositorio. Úselo `git add`para agregar el archivo `report.txt`al área de ensayo.
- 2 Use otro comando de Git para verificar el estado del repositorio.

# ¿Cómo puedo saber lo que se va a cometer?

Para comparar el estado de sus archivos con los del área de ensayo, puede utilizar `git diff -r HEAD`. La `-r`bandera significa "comparar con una revisión en particular", y `HEAD`es un atajo que significa "la confirmación más reciente".

Puede restringir los resultados a un solo archivo o directorio usando `git diff -r HEAD path/to/file`, donde la ruta al archivo es relativa a donde se encuentra (por ejemplo, la ruta desde el directorio raíz del repositorio).

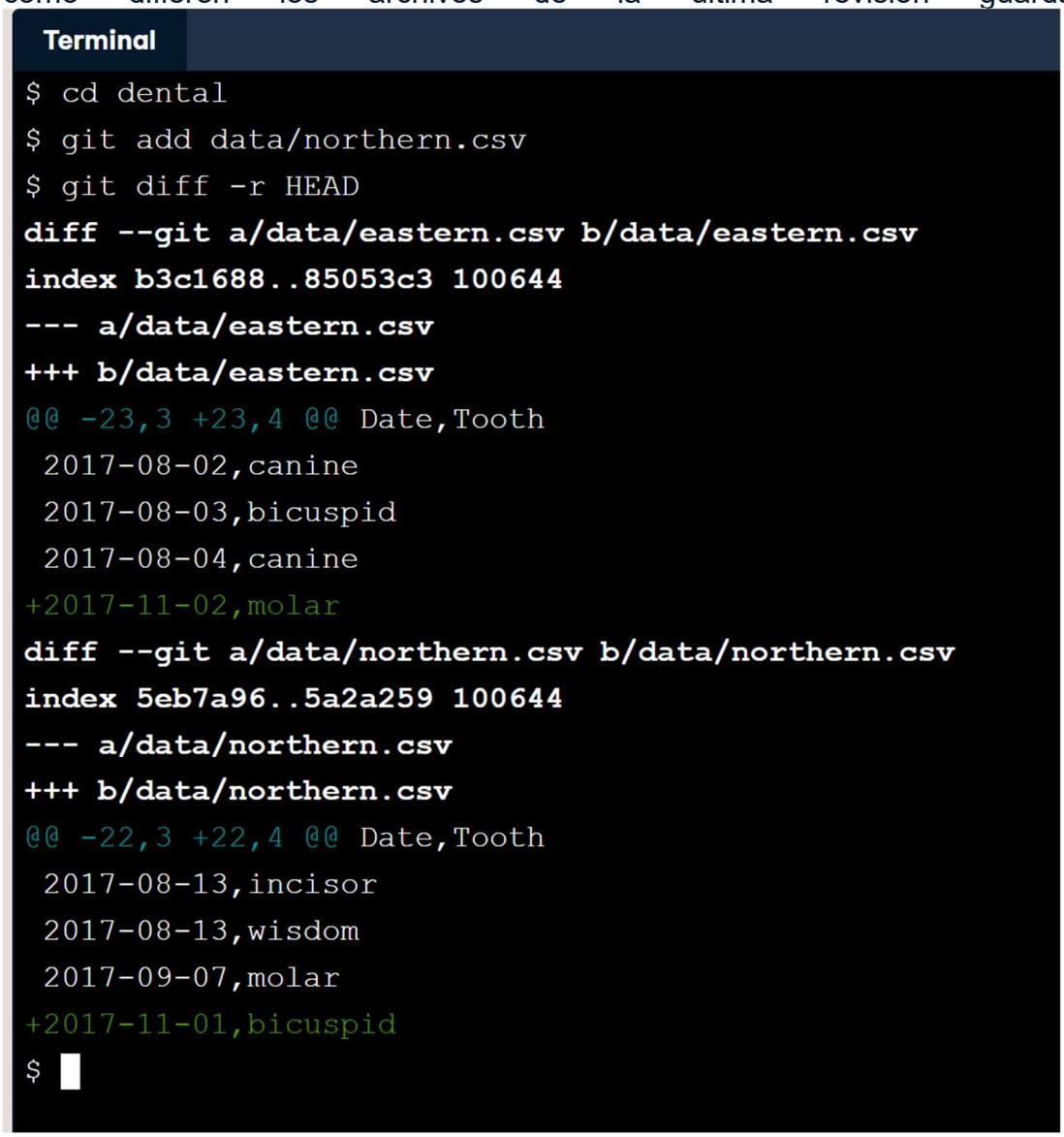
Exploraremos otros usos de `-r`y `HEAD`en el próximo capítulo.

Instrucciones1/3

**30 PX**

- 1Ha sido colocado en el `dental`repositorio, donde `data/northern.csv`se ha agregado al área de preparación. Use `git diff`con `-r`y un argumento para ver

cómo difieren los archivos de la última revisión guardada.




**Tomar Pista (-9 XP)**

- 2Use un solo comando de Git para ver los cambios en el archivo que se ha preparado (y *solo* ese archivo).

```
$ git diff -r HEAD data/northern.csv
diff --git a/data/northern.csv b/data/northern.csv
index 5eb7a96..5a2a259 100644
--- a/data/northern.csv
+++ b/data/northern.csv
@@ -22,3 +22,4 @@ Date,Tooth
 2017-08-13,incisor
 2017-08-13,wisdom
 2017-09-07,molar
+2017-11-01,bicuspid
```

- 3`data/eastern.csv`aún no se ha agregado al área de preparación. Use un comando Git para hacer esto ahora.

# Interludio: ¿cómo puedo editar un archivo?

Unix tiene una variedad desconcertante de editores de texto. En este curso, algunas veces usaremos uno muy simple llamado Nano. Si escribe `nano filename`, se abrirá `filename`para editarlo (o crearlo si aún no existe). Luego puede moverse con las teclas de flecha, eliminar caracteres con la tecla de retroceso, etc. También puede realizar algunas otras operaciones con combinaciones de teclas de control:

- Ctrl-K: eliminar una línea.
- Ctrl-U: anula la eliminación de una línea.
- Ctrl-O: guarda el archivo ('O' significa 'salida').
- Ctrl-X: salir del editor.

Instrucciones

**100 EXP**

Ejecute `nano names.txt`para editar un nuevo archivo en su directorio de inicio e ingrese las siguientes cuatro líneas:

```
Lovelace
Hopper
Johnson
```

```
Wilson
```

Para guardar lo que ha escrito, presione Ctrl-O para escribir el archivo, luego Enter para confirmar el nombre del archivo, luego Ctrl-X y Enter para salir del editor.

# ¿Cómo confirmo los cambios?

Para guardar los cambios en el área de preparación, utilice el comando `git commit`. Siempre guarda todo lo que está en el área de preparación como una unidad: como verá más adelante, cuando desea deshacer cambios en un proyecto, deshace todo un compromiso o nada.

Cuando confirma cambios, Git requiere que ingrese un **mensaje** de registro . Esto tiene el mismo propósito que un comentario en un programa: le dice a la siguiente persona que examine el repositorio por qué hizo un cambio.

De forma predeterminada, Git inicia un editor de texto que le permite escribir este mensaje. Para mantener las cosas simples, puede usar `-m "some message in quotes"`en la línea de comando para ingresar un mensaje de una sola línea como este:

```
git commit -m "Program appears to have become self-aware."
```

Si accidentalmente escribe mal un mensaje de confirmación, puede cambiarlo usando la `--amend`bandera.

```
git commit --amend - m "new message"
```

Instrucciones1/2

**50 PX**

- 1Se le ha colocado en el `dental`repositorio y `report.txt`se le ha agregado al área de preparación. Use un comando Git para verificar el estado del repositorio.

Git status

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   report.txt
```

2. Confirme los cambios en el área de preparación con el mensaje "Agregando una referencia".

```
$ git commit -m "Adding a reference."
[master 258ec63] Adding a reference.
 1 file changed, 3 insertions(+)
```

# ¿Cómo puedo ver el historial de un repositorio?

El comando `git log` se utiliza para ver el **registro** de la historia del proyecto. Las entradas de registro se muestran primero las más recientes y se ven así:

```
commit 043070548738119599 3bac9c21512ccfb511056d
Author: Rep Loop <repl@datacamp.com>
Date:   Wed Sep 20 13:42:26 2017 +0000

    Added year to report title.
```

La `commit` línea muestra una ID única para la confirmación llamada **hash** ; exploraremos esto más a fondo en el próximo capítulo. Las otras líneas le dicen quién hizo el cambio, cuándo y qué mensaje de registro escribieron para el cambio.

Cuando ejecuta `git log`, Git usa automáticamente un buscapersonas para mostrar una pantalla de salida a la vez. Presiona la barra espaciadora para bajar una página o la tecla 'q' para salir.

---

Estás en el directorio `dental`, que es un repositorio de Git. Use un solo comando de Git para ver el historial del repositorio. ¿Cuál es el mensaje en la primera entrada del registro (que se muestra al final)?

*Tenga en cuenta que es posible que no todas las entradas estén visibles en la primera pantalla y que es posible que deba consultar páginas adicionales para ver la primera entrada.*

Instrucciones

**50 PX**

Respuestas posibles

- "Archivo de informe de resumen agregado".

- "Archivos de datos CSV estacionales agregados"
- "Error solucionado y resultados regenerados".
- "Recordatorio añadido para citar fuentes de financiación".

# ¿Cómo puedo ver el historial de un archivo específico?

El registro completo de un proyecto puede ser abrumador, por lo que a menudo es útil inspeccionar solo los cambios en archivos o directorios particulares. Puede hacer esto usando `git log path`, donde `path`es la ruta a un archivo o directorio específico. El registro de un archivo muestra los cambios realizados en ese archivo; el registro de un directorio muestra cuándo se agregaron o eliminaron archivos en ese directorio, en lugar de cuándo se cambió el contenido de los archivos del directorio.

---

Usted ha sido puesto en el `dental`repositorio. Úselo `git log`para mostrar solo los cambios realizados en `data/southern.csv`. ¿Cuántos ha habido?

Instrucciones

**50 PX**

Respuestas posibles

- 0.
- 1.
- 2.
- 3.

# ¿Cómo escribo un mejor mensaje de registro?

Escribir un mensaje de registro de una línea con `git commit -m "message"` es lo suficientemente bueno para cambios muy pequeños, pero sus colaboradores (incluido usted mismo en el futuro) apreciarán más información. Si ejecuta `git commit` *sin* `-m "message"` , Git inicia un editor de texto con una plantilla como esta:

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is up-to-date with 'origin/master'.
#
# Changes to be committed:
#       modified:   skynet.R
#
```

Las líneas que comienzan con `#` son comentarios y no se guardarán. (Están ahí para recordarle lo que se supone que debe hacer y qué archivos ha cambiado). Su mensaje debe ir en la parte superior y puede ser tan largo y detallado como desee.

Instrucciones

**100 EXP**

Instrucciones

**100 EXP**

Se le ha colocado en el `dental` repositorio y `report.txt` se le ha agregado al área de preparación. Los cambios `report.txt` ya se han realizado. Use `git commit` *sin* `-m` para confirmar los cambios. Se abrirá el editor Nano. Escriba un mensaje significativo y use Ctrl+O y Enter para guardar, y luego Ctrl+X para salir del editor.

# ¿Cómo almacena Git la información?

Puede preguntarse qué información se almacena en cada confirmación que realiza. Git usa una estructura de tres niveles para esto.

1. Una **confirmación** contiene metadatos como el autor, el mensaje de confirmación y la hora en que ocurrió la confirmación. En el siguiente diagrama, la confirmación más reciente se encuentra en la parte inferior ( `feed0098`), debajo de sus confirmaciones principales.
2. Cada confirmación también tiene un **árbol** , que rastrea los nombres y las ubicaciones en el repositorio cuando ocurrió esa confirmación. En la confirmación más antigua (superior), había dos archivos rastreados por el repositorio.
3. Para cada uno de los archivos enumerados en el árbol, hay un **blob** . Contiene una instantánea comprimida del contenido del archivo cuando ocurrió la confirmación (blob es la abreviatura

de *objeto binario grande* , que es un término de la base de datos SQL para "puede contener datos de cualquier tipo"). En el compromiso medio, `report.md`y `draft.md`se cambiaron, por lo que los blobs se muestran junto a ese compromiso. `data/northern.csv`no cambió en esa confirmación, por lo que el árbol se vincula al blob de la confirmación anterior. La reutilización de blobs entre confirmaciones ayuda a acelerar las operaciones comunes y minimiza el espacio de almacenamiento.

# ¿Qué es un hash?

Cada envío a un repositorio tiene un identificador único llamado **hash** (ya que se genera al ejecutar los cambios a través de un generador de números pseudoaleatorios llamado **función hash** ). Este hash normalmente se escribe como una cadena hexadecimal de 40 caracteres como `7c35a3ce607a14953f070f0f83b5d74c2296ef93`, pero la mayoría de las veces, solo tiene que darle a Git los primeros 6 u 8 caracteres para identificar la confirmación a la que se refiere.

Los hashes son los que permiten a Git compartir datos de manera eficiente entre repositorios. Si dos archivos son iguales, se garantiza que sus hashes serán los mismos. Del mismo modo, si dos confirmaciones contienen los mismos archivos y tienen los mismos ancestros, sus valores hash también serán los mismos. Por lo tanto, Git puede decir qué información debe guardarse y dónde al comparar hashes en lugar de comparar archivos completos.

---

Use `cd`para ir al `dental`directorio y luego ejecute `git log`. ¿Cuáles son los primeros cuatro caracteres del hash de la confirmación más reciente?

```
$ cd dental
$ git log
commit da2604ea65c1404631c9b268c492d87654850e0f
Author: Rep Loop <repl@datacamp.com>
Date:   Thu Jul 15 17:05:49 2021 +0000

    Added year to report title.

commit bebb127aac82d90eef2145f14db83d2064128430
Author: Rep Loop <repl@datacamp.com>
Date:   Thu Jul 15 17:05:49 2021 +0000

    Adding fresh data for western region.

commit 29a59811032295e8bb3ddbbde5bb1e0ca59fb0c6
Author: Rep Loop <repl@datacamp.com>
Date:   Thu Jul 15 17:05:49 2021 +0000
```

Instrucciones

**50 PX**

Respuestas posibles

- camac
- 2e1b
- 2e95

Ninguna de las anteriores.

# ¿Cómo puedo ver una confirmación específica?

Para ver los detalles de una confirmación específica, utilice el comando `git show`con los primeros caracteres del hash de la confirmación. Por ejemplo, el comando `git show 0da2f7`produce esto:

```
commit 0da2f7ad11664ca9ed933c1ccd1f3cd24d481e42
Author: Rep Loop <repl@datacamp.com>
Date:   Wed Sep 5 15:39:18 2018 +0000

    Added year to report title.

diff --git a/report.txt b/report.txt
index e713b17..4c0742a 100644
--- a/report.txt
+++ b/report.txt
@@ -1,4 +1,4 @@
-# Seasonal Dental Surgeries 2017-18
+# Seasonal Dental Surgeries (2017) 2017-18

 TODO: write executive summary.
```

La primera parte es la misma que la entrada de registro que muestra `git log`. La segunda parte muestra los cambios; al igual que con `git diff`, las líneas que eliminó el cambio tienen el prefijo `-`, mientras que las líneas que agregó tienen el prefijo `+`.

---

Usted ha sido puesto en el `dental`directorio. Úselo `git log`para ver los hash de las confirmaciones recientes y, luego, `git show`con los primeros dígitos de un hash para ver la confirmación más reciente. ¿Cuántos archivos cambió?

Recordatorio: presione la barra espaciadora para desplazarse hacia abajo a través `git log`de la salida y `q`para salir de la pantalla paginada.

Instrucciones

**50 PX**

Respuestas posibles

- Ninguna.
- 1.

- 2.
- 4.

# ¿Cuál es el equivalente de Git de una ruta relativa?

Un hash es como una ruta absoluta: identifica una confirmación específica. Otra forma de identificar una confirmación es usar el equivalente de una ruta relativa. La etiqueta especial `HEAD`, que vimos en el capítulo anterior, siempre se refiere a la confirmación más reciente. Entonces , la etiqueta `HEAD~1`se refiere a la confirmación anterior, mientras que `HEAD~2`se refiere a la confirmación anterior, y así sucesivamente.

Tenga en cuenta que el símbolo entre `HEAD`y el número es una tilde `~`, *no* un signo menos `-`, y que no puede haber espacios antes o después de la tilde.

---

Estás en el `dental`repositorio. Con un solo comando de Git, muestre la confirmación realizada justo antes de la más reciente. ¿Cuál de los siguientes archivos cambió?

Instrucciones

**50 PX**

Respuestas posibles

- `report.txt`.
- `data/western.csv`.
- Los dos anteriores.
- Ninguno de los anteriores.

# ¿Cómo puedo ver quién cambió qué en un archivo?

`git log`muestra el historial general de un proyecto o archivo, pero Git puede brindar aún más información. El comando `git annotate file`muestra quién hizo el último cambio en cada línea de un archivo y cuándo. Por ejemplo, las primeras tres líneas de salida de se `git annotate report.txt`ven así:

```
04307054        (  Rep Loop     2017-09-20 13:42:26 +0000       1)# Seasonal De
5e6f92b6        (  Rep Loop     2017-09-20 13:42:26 +0000       2)
5e6f92b6        (  Rep Loop     2017-09-20 13:42:26 +0000       3)TODO: write e
```

Cada línea contiene cinco elementos, con los elementos dos a cuatro encerrados entre paréntesis. Al inspeccionar la primera línea, vemos:

1. Los primeros ocho dígitos del hash, `04307054`.
2. El autor, `Rep Loop`.
3. La hora de la confirmación, `2017-09-20 13:42:26 +0000`.
4. El número de línea, `1`.
5. El contenido de la línea, `# Seasonal Dental Surgeries (2017) 2017-18`.

---

Estás en el `dental`repositorio. Use un solo comando para ver los cambios en `report.txt`. ¿Cuántos conjuntos diferentes de cambios se han realizado en este archivo (es decir, cuántos valores hash ***distintos*** aparecen en la primera columna de la salida)?

Instrucciones

**50 PX**

Respuestas posibles

- 1.
- 3.
- 4.
- 7.

# ¿Cómo puedo ver qué cambió entre dos confirmaciones?

`git show`con un ID de confirmación muestra los cambios realizados *en* una confirmación en particular. Para ver los cambios *entre* dos confirmaciones, puede usar `git diff ID1..ID2`, where `ID1`e `ID2`identificar las dos confirmaciones que le interesan, y el conector `..`es un par de puntos. Por ejemplo, `git diff abc123..def456`muestra las diferencias entre las confirmaciones `abc123`y `def456`, mientras que `git diff HEAD~1..HEAD~3`muestra las diferencias entre el estado del repositorio una confirmación en el pasado y su estado tres confirmaciones en el pasado.

---

Estás en el `dental`repositorio. Úselo `git diff`para ver las diferencias entre su estado actual y su estado dos confirmaciones anteriores. ¿Cuál de los siguientes archivos ha cambiado?

Instrucciones

**50 PX**

Respuestas posibles

- `data/western.csv`.
- `report.txt`.
- `data/southern.csv`.
- `report.txt`y `data/western.csv`.
- `report.txt`y `data/southern.csv`.

# ¿Cómo agrego nuevos archivos?

Git no realiza un seguimiento de los archivos de forma predeterminada. En cambio, espera hasta que lo haya usado `git add`al menos una vez antes de comenzar a prestar atención a un archivo.

En el diagrama que vio al comienzo del capítulo, los archivos sin seguimiento no tendrán una mancha y no se enumerarán en un árbol.

Los archivos no rastreados no se beneficiarán del control de versiones, por lo que para asegurarse de que no se pierda nada, `git status`siempre le informará sobre los archivos que están en su repositorio pero que (todavía) no están siendo rastreados.

Instrucciones1/3

**30 PX**

- 1Estás en el `dental`repositorio. Úselo `git status`para encontrar los archivos que aún no se están rastreando.

  **Tomar Pista (-9 XP)**

- 2Úselo `git add`para agregar el nuevo archivo al área de ensayo.
- 3Úselo `git commit`para guardar los archivos preparados con el mensaje "Comenzando a rastrear fuentes de datos".

---

Mirando el diagrama, ¿qué archivos cambiaron en la confirmación más reciente de este repositorio?

responde la pregunta

**50XP**

Respuestas posibles

- `data/northern.csv`

  imprenta1

- `report.md`

  imprenta2

- `draft.md`

  imprenta3

# ¿Cómo le digo a Git que ignore ciertos archivos?

El análisis de datos a menudo produce archivos temporales o intermedios que no desea guardar. Puede indicarle que deje de prestar atención a los archivos que no le interesan creando un archivo en el directorio raíz de su repositorio llamado `.gitignore` y almacenando una lista de patrones **comodín** que especifican los archivos a los que no desea que Git preste atención. Por ejemplo, si `.gitignore`contiene:

```
build
*.mpl
```

entonces Git ignorará cualquier archivo o directorio llamado `build`(y, si es un directorio, todo lo que contenga), así como cualquier archivo cuyo nombre termine en `.mpl`.

---

¿Cuál de los siguientes archivos *no* sería ignorado por un `.gitignore`que contenía las líneas:

```
pdf
*.pyc
backup
```

responde la pregunta

**50XP**

Respuestas posibles

- `report.pdf`

  imprenta1

- `bin/analyze.pyc`

  imprenta2

- `backup/northern.csv`

  imprenta3

- Ninguna de las anteriores.

# ¿Cómo puedo eliminar archivos no deseados?

Git puede ayudarte a limpiar los archivos que le has dicho que no quieres. El comando `git clean -n` le mostrará una lista de archivos que están en el repositorio, pero cuyo historial Git no está rastreando actualmente. Un comando similar `git clean -f` eliminará esos archivos.

*Use este comando con cuidado:* `git clean` solo funciona en archivos sin seguimiento, por lo que, por definición, su historial no se ha guardado. Si los elimina con `git clean -f`, desaparecerán para siempre.

Instrucciones1/3

**30 PX**

- 1Estás en el `dental` repositorio. Úselo `git status` para ver el estado de su repositorio.
- 2`backup.log` parece ser un archivo sin seguimiento y es uno que no necesitamos. Deshagámonos de eso. Úselo `git clean` con la bandera adecuada para eliminar archivos no deseados.
- 3Úselo `ls` para listar los archivos en su directorio de trabajo actual. `backup.log` ya no debería estar ahí!

# ¿Cómo puedo ver cómo está configurado Git?

Como la mayoría de las piezas de software complejas, Git le permite cambiar su configuración predeterminada. Para ver cuáles son las configuraciones, puede usar el comando `git config --list`con una de tres opciones adicionales:

- `--system`: configuración para cada usuario en esta computadora.
- `--global`: ajustes para cada uno de tus proyectos.
- `--local`: ajustes para un proyecto específico.

Cada nivel anula al anterior, por lo que **la configuración local** (por proyecto) tiene prioridad sobre **la configuración global** (por usuario), que a su vez tiene prioridad sobre la **configuración del sistema** (para todos los usuarios de la computadora).

---

Estás en el `dental`repositorio. ¿Cuántos valores de configuración local se establecen para este repositorio?

Instrucciones

**50 PX**

Respuestas posibles

- Ninguna.
- 3.
- 4.
- 7.

# ¿Cómo puedo cambiar mi configuración de Git?

La mayoría de las configuraciones de Git deben dejarse como están. Sin embargo, hay dos que debe configurar en cada computadora que use: su nombre y su dirección de correo electrónico. Estos se registran en el registro cada vez que realiza un cambio y, a menudo, se utilizan para identificar a los autores del contenido de un proyecto para dar crédito (o culpar, según las circunstancias).

Para cambiar un valor de configuración para todos sus proyectos en una computadora en particular, ejecute el comando:

```
git config --global setting value
```

Con este comando, especifica lo `setting`que desea cambiar y lo `value`que desea configurar. Los ajustes que identifican su nombre y dirección de correo electrónico son `user.name`y `user.email`, respectivamente.

Instrucciones

**100 EXP**

Cambie la dirección de correo electrónico ( `user.email`) configurada para el usuario actual para *todos los* proyectos a `rep.loop@datacamp.com`.

```
$ git config --global user.email rep.loop@datacamp.com
```

# ¿Cómo puedo cometer cambios de forma selectiva?

No tienes que poner todos los cambios que has hecho recientemente en el área de preparación a la vez. Por ejemplo, suponga que está agregando una

función `analysis.R` y detecta un error en `cleanup.R`. Después de haberlo arreglado, desea guardar su trabajo. Dado que los cambios en `cleanup.R`no están directamente relacionados con el trabajo que está haciendo en `analysis.R`, debe guardar su trabajo en dos confirmaciones separadas.

La sintaxis para organizar un solo archivo es `git add path/to/file`.

Si comete un error y accidentalmente prepara un archivo que no debería tener, puede quitar las adiciones `git reset HEAD`e intentarlo de nuevo.

Instrucciones1/2

**50 PX**

- 1Desde la salida de `git status`a la derecha, verá que se cambiaron dos archivos; `data/northern.csv`y `data/eastern.csv`. Registre solo los cambios realizados en `data/northern.csv`.

```
$ git add data/northern.csv
```

2.Confirme esos cambios con el mensaje "Agregando datos de la región norte".

```
$ git commit --amend -m "Adding data from northern region."
```

# ¿Cómo reorganizo los archivos?

Las personas a menudo guardan su trabajo cada pocos minutos cuando usan un editor de texto de escritorio. De manera similar, es común usarlo `git add`periódicamente para guardar los cambios más recientes en un archivo en el área de ensayo. Esto es particularmente útil cuando los cambios son experimentales y es posible que desee deshacerlos sin saturar el historial del repositorio.

Instrucciones1/2

**50 PX**

- 1Estás en el `dental`repositorio. Se utiliza `git status`para comprobar el estado del repositorio.

```
$ cd dental
$ git add data/northern.csv
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   data/northern.csv

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directo
)

        modified:   data/northern.csv
```

- 2.Parece que `data/northern.csv` ya se han realizado algunos cambios, pero hay cambios nuevos que aún no se han realizado. Úselo `git add` para volver a organizar estos últimos cambios `data/northern.csv`.

```
$ git add data/northern.csv
```

# ¿Cómo puedo deshacer cambios en archivos no preparados?

Suponga que ha realizado cambios en un archivo y luego decide que desea **deshacerlos** . Su editor de texto puede hacer esto, pero una forma más confiable es dejar que Git haga el trabajo. El comando:

```
git checkout -- filename
```

descartará los cambios que aún no hayan sido escenificados. (El guión doble `--` debe estar allí para separar el `git checkout` comando de los nombres del archivo o archivos que desea recuperar).

*Use este comando con cuidado:* una vez que descarte los cambios de esta manera, desaparecerán para siempre.

Instrucciones

**100 EXP**

Está en el `dental` repositorio, donde se organizaron todos los cambios en los `.csv` archivos `data`. `git status` espectáculos que `data/northern.csv` se cambiaron

nuevamente después de su puesta en escena. Utilice un comando de Git para deshacer los cambios en el archivo `data/northern.csv`.

```
$ git checkout -- data/northern.csv
```

# ¿Cómo puedo deshacer los cambios en los archivos preparados?

Al comienzo de este capítulo, vio que `git reset`eliminará los archivos que previamente preparó usando `git add`. Al combinar `git reset`con `git checkout`, puede deshacer cambios en un archivo en el que realizó cambios. La sintaxis es la siguiente.

```
git reset HEAD path/to/file
git checkout -- path/to/file
```

(Quizás se pregunte por qué hay dos comandos para restablecer los cambios. La respuesta es que deshacer un archivo y deshacer los cambios son casos especiales de operaciones de Git más poderosas que aún no ha visto).

Instrucciones1/2

**50 PX**

- 1Úselo `git reset`para eliminar el archivo `data/northern.csv` (y *solo* ese archivo).

  **Tomar Pista (-15 XP)**

- 2Úselo `git checkout --`para deshacer los cambios desde la última confirmación de `data/northern.csv`.

# ¿Cómo restauro una versión antigua de un archivo?

Anteriormente vio cómo usar `git checkout`para deshacer los cambios que realizó desde la última confirmación. Este comando también se puede usar para retroceder aún más en el historial de un archivo y restaurar versiones de ese archivo desde una confirmación. De esta manera, puede pensar en comprometerse como guardar su trabajo y **verificar** como cargar esa versión guardada.

La sintaxis para restaurar una versión anterior toma dos argumentos: el hash que identifica la versión que desea restaurar y el nombre del archivo.

Por ejemplo, si `git log`muestra esto:

```
commit ab8883e8a6bfa873d44616a0f356125dbaccd9ea
Author: Author: Rep Loop <repl@datacamp.com>
Date:   Thu Oct 19 09:37:48 2017 -0400

    Adding graph to show latest quarterly results.

commit 2242bd761bbeafb9fc82e33aa5dad966adfe5409
Author: Author: Rep Loop <repl@datacamp.com>
Date:   Thu Oct 16 09:17:37 2017 -0400

    Modifying the bibliography format.
```

luego `git checkout 2242bd report.txt`reemplazaría la versión actual de `report.txt`con la versión que se confirmó el 16 de octubre. Tenga en cuenta que esta es la misma sintaxis que usó para deshacer los cambios no preparados, excepto `--`que se reemplazó por un hash.

Restaurar un archivo no borra nada del historial del repositorio. En su lugar, el acto de restaurar el archivo se guarda como otra confirmación, porque es posible que más adelante desee deshacer lo que deshizo.

Una cosa más: hay otra característica `git log`que será útil aquí. Pasar `-`seguido de un número restringe la salida a esa cantidad de confirmaciones. Por ejemplo, `git log -3 report.txt`le muestra las últimas tres confirmaciones relacionadas con `report.txt`.

Instrucciones1/4

**25 PX**

- 1El contenido actual de `data/western.csv`se muestra en la terminal. Úselo `git log -2`para enumerar los dos últimos cambios en ese archivo.

  **Tomar Pista (-7 XP)**

- 2Úselo `git checkout`con los primeros caracteres de un hash para restaurar la versión `data/western.csv`que tiene el mensaje de confirmación `"Adding fresh data for southern and western regions."`.

- ```
  $ git checkout 29a5981 data/western.csv
  ```
- 3Úselo `cat data/western.csv`para mostrar los contenidos actualizados.
- 
- 4Confirme la versión restaurada de `data/western.csv`y asegúrese de incluir un mensaje.
  ```
  $ git commit -m "hola"
  [master 7db861f] hola
   1 file changed, 3 deletions(-)
  ```

# ¿Cómo puedo deshacer todos los cambios que he hecho?

Hasta ahora, ha visto cómo deshacer cambios en un solo archivo a la vez usando `git reset HEAD path/to/file`. A veces querrá deshacer los cambios en muchos archivos.

Una forma de hacer esto es dar `git reset`un directorio. Por ejemplo, `git reset HEAD data`eliminará cualquier archivo del `data`directorio. Aún mejor, si no proporciona ningún archivo o directorio, se eliminará todo. Aún mejor, `HEAD`es el compromiso predeterminado para quitar el escenario, por lo que simplemente puede escribir `git reset`para quitar el escenario de todo.

De manera similar `git checkout -- data`, restaurará los archivos en el `data`directorio a su estado anterior. No puede dejar el argumento del archivo completamente en blanco, pero recuerde de **Introducción a Shell para Data Science** que puede hacer referencia al directorio actual como `.`. Entonces `git checkout -- .`revertirá todos los archivos en el directorio actual.

Instrucciones1/2

**50 PX**

- 1Úselo `git reset`para eliminar todos los archivos del área de preparación.

  **Tomar Pista (-15 XP)**

- 2Úselo `git checkout`para volver a poner esos archivos en su estado anterior. Use el nombre del directorio `.`para referirse a "todos los archivos en o debajo de este directorio", y sepárelo del comando con `--`.
  ```

# ¿Qué es una sucursal?

Si no usa el control de versiones, un flujo de trabajo común es crear diferentes subdirectorios para contener diferentes versiones de su proyecto en diferentes estados, por ejemplo `development`y `final`. Por supuesto, entonces siempre terminas con `final-updated`y `final-updated-revised`también. El problema con esto es que se vuelve difícil determinar si tiene la versión correcta de cada archivo en el subdirectorio correcto y corre el riesgo de perder el trabajo.

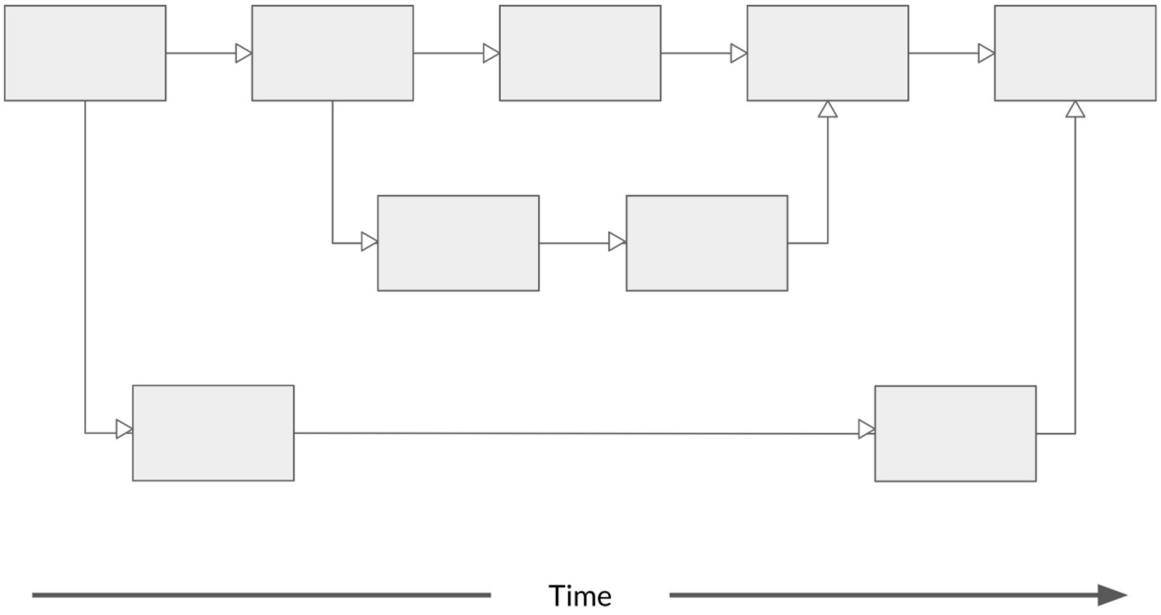
Una de las razones por las que Git es popular es su soporte para crear **ramas** , lo que le permite tener múltiples versiones de su trabajo y le permite rastrear cada versión sistemáticamente.

Cada rama es como un universo paralelo: los cambios que realice en una rama no afectan a otras ramas (hasta que las vuelva a **fusionar** ).

Nota: el Capítulo 2 describió la estructura de datos de tres partes que usa Git para registrar el historial de un repositorio: *blobs* para archivos, *árboles* para los estados guardados de los repositorios y *compromisos* para registrar los cambios. Las ramas son la razón por la que Git necesita árboles y confirmaciones: una confirmación tendrá dos padres cuando las ramas se fusionen.

---

En el siguiente diagrama, cada cuadro es una confirmación y las flechas apuntan a la siguiente confirmación ("secundaria"). ¿Cuántas fusiones se han producido?




responde la pregunta

**50XP**

Respuestas posibles

- Ninguna

  imprenta1
- 1

  imprenta2
- 2

  imprenta3
- 3

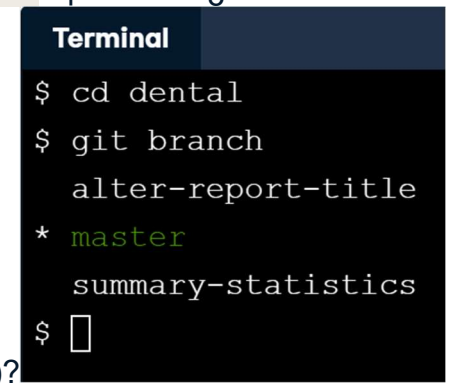
  imprenta4

# ¿Cómo puedo ver qué ramas tiene mi repositorio?

De forma predeterminada, cada repositorio de Git tiene una rama llamada `master` (que es la razón por la que ha estado viendo esa palabra en la salida de Git en lecciones anteriores). Para enumerar todas las sucursales en un repositorio, puede ejecutar el comando `git branch`. La sucursal en la que se encuentra actualmente se mostrará con una `*`al lado de su nombre.

---

Estás en el `dental`repositorio. ¿Cuántas sucursales hay en este repositorio (incluyendo `master`)?




Instrucciones

**50 PX**

Respuestas posibles

- Ninguna.
- 1.
- 2.
- 3.

# ¿Cómo puedo ver las diferencias entre sucursales?

Las ramas y las revisiones están estrechamente conectadas, y los comandos que funcionan en las últimas suelen funcionar en las primeras. Por ejemplo, así como `git diff revision-1..revision-2` muestra la diferencia entre dos versiones de un repositorio, `git diff branch-1..branch-2` muestra la diferencia entre dos ramas.

---

Estás en el `dental` repositorio. ¿Cuántos archivos en la `summary-statistics` rama son diferentes de sus equivalentes en la `master` rama?

Instrucciones

**50 PX**

Respuestas posibles

- Ninguna.
- 1.
- 3.
- 8.

# ¿Cómo puedo cambiar de una sucursal a otra?

Anteriormente utilizó `git checkout` con un hash de confirmación para cambiar el estado del repositorio a ese hash. También puede usar `git checkout` con el nombre de una sucursal para cambiar a esa sucursal.

Dos notas:

1. Cuando ejecuta `git branch`, pone un `*` al lado del nombre de la sucursal en la que se encuentra actualmente.
2. Git solo te permitirá hacer esto si todos tus cambios se han confirmado. Puede evitar esto, pero está fuera del alcance de este curso.

En este ejercicio, también utilizará `git rm`. Esto elimina el archivo (al igual que el comando de shell `rm`) y luego organiza la eliminación de ese archivo con `git add`, todo en un solo paso.

Instrucciones1/6

**16 PX**

- 1 Estás en la `master` rama del `dental` repositorio. Cambiar a la `summary-statistics` sucursal.

```
$ git checkout summary-statistics
Switched to branch 'summary-statistics'
```

**Tomar Pista (-4 XP)**

- 2 Úselo `git rm` para eliminar `report.txt`.

```
$ git rm report.txt
rm 'report.txt'
```

- 3 Confirme su cambio con `-m "Removiendo informe"` como un mensaje.

```
$ git commit -m "Removing reporte"
[summary-statistics 566f9f5] Removing reporte
 1 file changed, 7 deletions(-)
 delete mode 100644 report.txt
```

- 4 Utilícelo `ls` para comprobar que se ha ido.

```
$ ls
bin  data  results
```

- 5 Vuelve a la `master` sucursal.

```
$ git checkout master
Switched to branch 'master'
```

- 6Úselo `ls`para asegurarse de que `report.txt`todavía está allí.

```
$ ls
bin  data  report.txt  results
$
```

# ¿Cómo puedo crear una sucursal?

Podría esperar que lo usaría `git branch`para crear una rama y, de hecho, esto es posible. Sin embargo, lo más común que desea hacer es crear una rama y luego cambiar a esa rama.

En el ejercicio anterior, solía `git checkout branch-name`cambiar a una sucursal. Para crear una rama y luego cambiar a ella en un solo paso, agregue una `-b`bandera, llamando a `git checkout -b branch-name`,

El contenido de la nueva rama es inicialmente idéntico al contenido de la original. Una vez que comience a realizar cambios, solo afectarán a la nueva rama.

Instrucciones1/4

**25 PX**

- 1Estás en la `master`rama del `dental`repositorio. Cree una nueva rama llamada `deleting-report`.

```
$ git checkout -b deleting-report
Switched to a new branch 'deleting-report'
$
```

**Tomar Pista (-7 XP)**

- 2Úselo `git rm report.txt`para eliminar el informe.

```
$ git rm report.txt
rm 'report.txt'
```

- 3Confirme sus cambios con un mensaje de registro.

```
$ git commit -m
error: switch `m' requires a value
usage: git commit [<options>] [--] <pathspec>...
```

- 4Úselo `git diff`con los argumentos apropiados para comparar la `master`rama con el nuevo estado de la `deleting-report`rama.

```
git diff master..deleting-report
```

# ¿Cómo puedo fusionar dos sucursales?

La ramificación te permite crear universos paralelos; **la fusión** es cómo los vuelves a unir. Cuando fusiona una rama (llámela origen) con otra (llámela destino), Git incorpora los cambios realizados en la rama de origen en la rama de destino. Si esos cambios no se superponen, el resultado es una nueva confirmación en la rama de destino que incluye todo lo de la rama de origen (los siguientes ejercicios describen lo que sucede si hay *conflictos* ).

Para fusionar dos sucursales, ejecuta `git merge source destination` (sin `..`entre los dos nombres de sucursal). Git abre automáticamente un editor para que pueda escribir un mensaje de registro para la fusión; puede mantener su mensaje predeterminado o completar algo más informativo.

Instrucciones1/1

**100 EXP**

- 1Estás en la `master`rama del `dental`repositorio. Combine los cambios *de* la `summary-statistics`rama (el origen) en la `master`rama (el destino) con el mensaje "Fusionando estadísticas de resumen".

**Terminal**

```
$ cd dental
$ gir merge --no-edit summary-statistics master
bash: gir: command not found
$ git merge --no-edit summary-statistics master
Merge made by the 'recursive' strategy.
 bin/summary         | 4 ++++
 results/summary.txt | 2 ++
 2 files changed, 6 insertions(+)
 create mode 100755 bin/summary
 create mode 100644 results/summary.txt
$
```

# ¿Qué son los conflictos?

A veces, los cambios en dos ramas entran en conflicto entre sí: por ejemplo, las correcciones de errores pueden tocar las mismas líneas de código, o los análisis en dos ramas diferentes pueden agregar registros nuevos (y diferentes) a un archivo de datos de resumen. En este caso, Git depende de ti para reconciliar los cambios en conflicto.

---

El archivo `todo.txt`inicialmente contiene estas dos líneas:

```
A) Write report.
B) Submit report.
```

Creas una rama llamada `update`y modificas el archivo para que sea:

```
A) Write report.
B) Submit final version.
C) Submit expenses.
```

Luego vuelve a la `master`rama y elimina la primera línea, de modo que el archivo contenga:

```
B) Submit report.
```

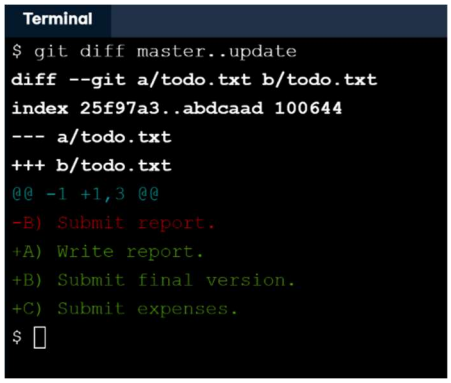
Cuando intenta fusionar `update`y `master`, ¿qué conflictos informa Git? Puede usar `git diff master..update`para ver la diferencia entre las dos ramas.

Instrucciones

**50 PX**

Respuestas posibles

- Solo la línea B, ya que es la única que cambia en ambos ramales.
- Líneas A y B, ya que se eliminó una y se modificó la otra.
- Líneas B y C, ya que se cambió una y se eliminó la otra.
- Las tres líneas, ya que todas se agregaron, eliminaron o cambiaron.




# ¿Cómo puedo fusionar dos ramas con conflictos?

Cuando hay un conflicto durante una combinación, Git le dice que hay un problema y, al ejecutarse `git status` después de la combinación, le recuerda qué archivos tienen conflictos que debe resolver imprimiendo `both modified:` junto a los nombres de los archivos.

Dentro del archivo, Git deja marcadores que se ven así para indicarle dónde ocurrieron los conflictos:

```
<<<<<<< destination-branch-name
...changes from the destination branch...
=======
...changes from the source branch...
>>>>>>> source-branch-name
```

En muchos casos, el nombre de la sucursal de destino se `HEAD` debe a que se fusionará con la sucursal actual. Para resolver el conflicto, edite el archivo para eliminar los marcadores y realice cualquier otro cambio que sea necesario para conciliar los cambios, luego confirme esos cambios.

Instrucciones1/5

**20 PX**

- 1Estás en la `master`rama del `dental`repositorio. Combine los cambios *de* la `alter-report-title`rama (el origen) en la `master`rama (el destino).

```
Terminal
$ cd dental
$ git branch
  alter-report-title
* master
  summary-statistics
$ git merge alter-report-title master
Auto-merging report.txt
CONFLICT (content): Merge conflict in report.txt
Automatic merge failed; fix conflicts and then commit the result.
$
```

**Tomar Pista (-6 XP)**

- 2Úselo `git status`para ver qué archivo tiene conflictos.

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   report.txt

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

- 3Resulta que `report.txt`tiene algunos conflictos. Use `nano report.txt`para abrirlo y eliminar algunas líneas para que solo se mantenga el segundo título. Guarde su trabajo con Ctrl+O y Enter, y luego salga del editor con Ctrl+X. Puede eliminar fácilmente líneas enteras con Ctrl+K.
- 4Agregue el archivo fusionado al área de ensayo.
- 5Confirme sus cambios con un mensaje de registro.

```
git commit -m "Reconciling"
```

# ¿Cómo puedo crear un nuevo repositorio?

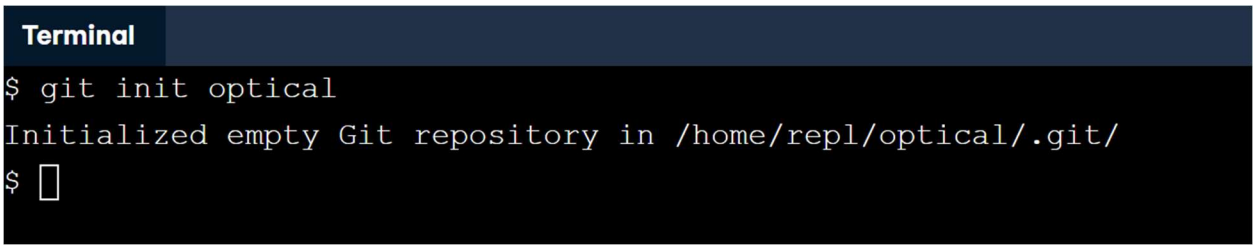
Hasta ahora, ha estado trabajando con repositorios preexistentes. Si desea crear un repositorio para un nuevo proyecto en el directorio de trabajo actual, simplemente puede decir `git init project-name`, donde "nombre del proyecto" es el nombre que desea que tenga el directorio raíz del nuevo repositorio.

Una cosa que *no* debes hacer es crear un repositorio Git dentro de otro. Si bien Git permite esto, la actualización de **repositorios anidados** se vuelve muy complicada muy rápidamente, ya que debe decirle a Git en cuál de los dos `.git`directorios se almacenará la actualización. Ocasionalmente, los proyectos muy grandes necesitan hacer esto, pero la mayoría de los programadores y analistas de datos lo intentan. para evitar caer en esta situación.

Instrucciones

**100 EXP**

Use un solo comando para crear un nuevo repositorio de Git llamado `optical`en su directorio actual.




# ¿Cómo puedo convertir un proyecto existente en un repositorio Git?

Los usuarios experimentados de Git inician instintivamente nuevos proyectos mediante la creación de repositorios. Sin embargo, si es nuevo en Git o trabaja con personas que lo son, a menudo querrá convertir proyectos existentes en repositorios. Hacerlo es simple, solo ejecuta:

```
git init
```

en el directorio raíz del proyecto, o:

```
git init /path/to/project
```

desde cualquier otro lugar de su computadora.

Instrucciones1/2

**50 PX**

- 1Estás en el directorio `dental`, que aún no es un repositorio de Git. Use un solo comando para convertirlo en un repositorio de Git.

```
Terminal
$ pwd
/home/repl/dental
$ git init
Initialized empty Git repository in /home/repl/dental/.git/
$
```

- **Tomar Pista (-15 XP)**
- 2Consulta el estado de tu nuevo repositorio.

```
Terminal
$ pwd
/home/repl/dental
$ git init
Initialized empty Git repository in /home/repl/dental/.git/
$ git status
On branch master

Initial commit

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

        bin/
        data/
        report.txt
        results/

nothing added to commit but untracked files present (use "git add" to trac
k)
$
```

# ¿Cómo puedo crear una copia de un repositorio existente?

A veces, se unirá a un proyecto que ya se está ejecutando, heredará un proyecto de otra persona o continuará trabajando en uno de sus propios proyectos en una nueva máquina. En cada caso, **clonará** un repositorio existente en lugar de crear uno nuevo. La clonación de un repositorio hace exactamente lo que sugiere el nombre: crea una copia de un repositorio existente (incluido todo su historial) en un nuevo directorio.

Para clonar un repositorio, utilice el comando `git clone URL`, donde `URL`identifica el repositorio que desea clonar. Esto normalmente será algo como

```
https://github.com/datacamp/project.git
```

pero para esta lección, usaremos un repositorio en el sistema de archivos local, por lo que solo puede usar una ruta a ese directorio. Cuando clonas un repositorio, Git usa el nombre del repositorio existente como el nombre del directorio raíz del clon, por ejemplo:

```
git clone /existing/project
```

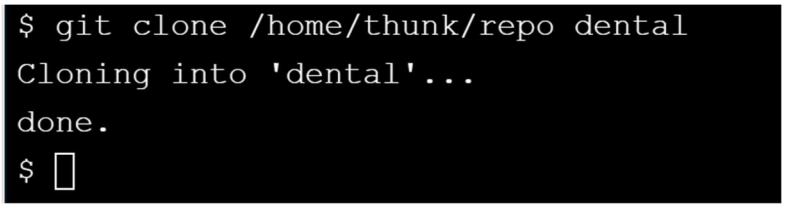
creará un nuevo directorio llamado `project`dentro de su directorio de inicio. Si desea llamar al clon de otra manera, agregue el nombre del directorio que desea al comando:

```
git clone /existing/project newprojectname
```

Instrucciones

**100 EXP**

Acaba de heredar el proyecto de análisis de datos dentales de un colega, quien le dice que todo su trabajo está en un repositorio en formato `/home/thunk/repo`. Use un solo comando para clonar este repositorio para crear un nuevo repositorio llamado `dental`dentro de su directorio de inicio.



# ¿Cómo puedo averiguar dónde se originó un repositorio clonado?

Cuando clonas un repositorio, Git recuerda dónde estaba el repositorio original. Lo hace almacenando un **control remoto** en la configuración del nuevo repositorio. Un control remoto es como un marcador de navegador con un nombre y una URL.

Si utiliza un servicio de alojamiento de repositorios git en línea como GitHub o Bitbucket, una tarea común sería clonar un repositorio de ese sitio para que funcione localmente en su computadora. Entonces la copia en el sitio web es el control remoto.

Si está en un repositorio, puede listar los nombres de sus controles remotos usando `git remote`.

Si desea obtener más información, puede usar `git remote -v`(para "detallado"), que muestra las URL del control remoto. Tenga en cuenta que "URL" es plural: es posible que un control remoto tenga varias URL asociadas con diferentes propósitos, aunque en la práctica, cada control remoto casi siempre se combina con una sola URL.

---

Estás en el `dental`repositorio. ¿Cuántos controles remotos tiene?

Instrucciones

**50 PX**

Respuestas posibles

- Ninguna.
- 1.
- 2.

# ¿Cómo puedo definir los controles remotos?

Cuando clonas un repositorio, Git crea automáticamente un llamado remoto `origin` que apunta al repositorio original. Puede agregar más controles remotos usando:

```
git remote add remote-name URL
```

y elimine los existentes usando:

```
git remote rm remote-name
```

Puede conectar dos repositorios de Git cualquiera de esta manera, pero en la práctica, casi siempre conectará repositorios que comparten algún ancestro común.

Instrucciones

- **100 EXP**
- Estás en el `dental`repositorio. Agregue `/home/thunk/repo`como un control remoto llamado `thunk`a él.

```
$ git remote add thunk /home/thunk/repo
```

# ¿Cómo puedo extraer cambios de un repositorio remoto?

Git realiza un seguimiento de los repositorios remotos para que pueda **extraer** cambios de esos repositorios y **enviarlos** .

Recuerde que el repositorio remoto suele ser un repositorio en un servicio de hospedaje en línea como GitHub. Un flujo de trabajo típico es que extrae el trabajo de sus colaboradores del repositorio remoto para tener la última versión de todo, hace algo de trabajo usted mismo y luego envía su trabajo al remoto para que sus colaboradores tengan acceso a él.

La extracción de cambios es sencilla: el comando `git pull remote branch` obtiene todo lo que se encuentra `branch`en el repositorio remoto identificado por `remote` y lo fusiona con la rama actual de su repositorio local. Por ejemplo, si está en la `quarterly-report`rama de su repositorio local, el comando:

```
git pull thunk latest-analysis
```

obtendría los cambios de `latest-analysis`la sucursal en el repositorio asociado con el control remoto llamado `thunk` y los fusionaría en su `quarterly-report`sucursal.

Instrucciones

**100 EXP**

Estás en la `master`rama del repositorio `dental`. Extraiga los cambios de la `master`rama llamada `origin`.

```
$ git pull origin master
From /home/thunk/repo
 * branch            master     -> FETCH_HEAD
Updating bebb127..0ae302b
Fast-forward
 report.txt | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
$
```

# ¿Qué sucede si trato de extraer cuando tengo cambios sin guardar?

Así como Git le impide cambiar de rama cuando tiene trabajo sin guardar, también le impide obtener cambios de un repositorio remoto cuando hacerlo podría sobrescribir las cosas que ha hecho localmente. La solución es simple: confirme sus cambios locales o retírelos, y luego intente extraerlos nuevamente.

Instrucciones1/3

**30 PX**

- 1Estás en el repositorio dental, que fue clonado desde un remoto llamado `origin`. Úselo `git pull`para traer cambios desde ese repositorio.

**Terminal**

```
$ cd dental
$ git pull
Updating da2604e..0ae302b
error: Your local changes to the following files would be overwritten by
erge:
        report.txt
Please, commit your changes or stash them before you can merge.
Aborting
$
```

**Tomar Pista (-9 XP)**

- 2Descarte los cambios en su repositorio.

**Solution**

```
git checkout -- .
```

```
$ git checkout -- .
```

- 3Vuelva a intentar el `git pull`.

```
$ $ git pull
Updating da2604e..0ae302b
Fast-forward
 report.txt | 2 ++
 1 file changed, 2 insertions(+)
$
```

# ¿Cómo puedo enviar mis cambios a un repositorio remoto?

El complemento de `git pull`is `git push`, que envía los cambios que ha realizado localmente a un repositorio remoto. La forma más común de usarlo es:

```
git push remote-name branch-name
```

que empuja el contenido de su rama `branch-name` a una rama con el mismo nombre en el repositorio remoto asociado con `remote-name`. Es posible usar diferentes nombres de sucursales en su extremo y en el extremo del control remoto, pero hacer esto rápidamente se vuelve confuso: casi siempre es mejor usar los mismos nombres para las sucursales en todos los repositorios.

Instrucciones1/3

**30 PX**

- 1Estás en la `master`rama del `dental`repositorio, que tiene un remoto llamado `origin`. Has cambiado `data/northern.csv`; añádelo al área de preparación.

- Terminal

```
$ cd dental
$ git add data/northern.csv
$
```

**Tomar Pista (-9 XP)**

- 2Confirme sus cambios con el mensaje "Se agregaron más datos del norte".

**Solution**

```
git commit -m "Added more northern data."
```

```
$ git commit -m "Added more northern data."
[master 7804271] Added more northern data.
 1 file changed, 1 insertion(+)
```

- 3Empuje sus cambios al repositorio remoto `origin`, especificando la `master`rama.

**Solution**

```
git push origin master
```

```
$ git push origin master
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 371 bytes | 0 bytes/s, done.
Total 4 (delta 3), reused 0 (delta 0)
To /home/thunk/repo
   0ae302b..7804271  master -> master
```

# ¿Qué sucede si mi push entra en conflicto con el trabajo de otra persona?

Sobrescribir tu propio trabajo por accidente es malo; sobrescribir la de otra persona es peor.

Para evitar que esto suceda, Git no le permite enviar cambios a un repositorio remoto a menos que haya fusionado el contenido del repositorio remoto en su propio trabajo.

En este ejercicio, realizó y confirmó cambios en el `dental`repositorio localmente y desea enviar sus cambios a un repositorio remoto.

Instrucciones1/3

**30 PX**

- 1Úselo `git push`para enviar esos cambios al repositorio remoto `origin`, especificando la `master`rama.

```
$ git push origin master
```

**Terminal**

```
$ cd dental
$ git add data/northern.csv
$ git commit -m "Adding a record"
[master 6f2c32f] Adding a record
 1 file changed, 1 insertion(+)
$ git push origin master
To /home/thunk/repo
 ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to '/home/thunk/repo'
hint: Updates were rejected because the tip of your current branch is behi
nd
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

**Tomar Pista (-9 XP)**

- 2Para evitar que sobrescriba el trabajo remoto, Git se ha negado a ejecutar su inserción. Úselo `git pull`para actualizar su repositorio con `origin`. Se abrirá un editor del que puede salir con Ctrl+X.

**Solution**

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

```
$ git pull --no-edit origin master
From /home/thunk/repo
 * branch            master     -> FETCH_HEAD
Already up-to-date.
```

- 3Ahora que ha fusionado el estado del repositorio remoto en su repositorio local, intente presionar nuevamente.

```
$ git push origin master
Counting objects: 2, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 326 bytes | 0 bytes/s, done.
Total 2 (delta 1), reused 0 (delta 0)
To /home/thunk/repo
   7804271..7cb5f8e  master -> master
$
```

Anexos