Contenido

Git - Conceptos básicos

Sistema de control de versiones

Version Control System (VCS) es un software que ayuda a los desarrolladores de software a trabajar juntos y mantener un historial completo de su trabajo.

A continuación, se enumeran las funciones de un VCS:

- Permite a los desarrolladores trabajar simultáneamente.
- No permite sobrescribir los cambios de los demás.

- Mantiene un historial de cada versión.

Los siguientes son los tipos de VCS:

- Sistema de control de versiones centralizado (CVCS).
- Sistema de control de versiones distribuido/descentralizado (DVCS).

En este capítulo, nos concentraremos solo en el sistema de control de versiones distribuido y especialmente en Git. Git cae bajo el sistema de control de versiones distribuidas.

Sistema de control de versiones distribuido

El sistema de control de versiones centralizado (CVCS) utiliza un servidor central para almacenar todos los archivos y permite la colaboración en equipo. Pero el mayor inconveniente de CVCS es su único punto de falla, es decir, la falla del servidor central. Desafortunadamente, si el servidor central se cae durante una hora, entonces, durante esa hora, nadie puede colaborar en absoluto. E incluso en el peor de los casos, si el disco del servidor central se corrompe y no se ha realizado una copia de seguridad adecuada, perderá todo el historial del proyecto. Aquí, el sistema de control de versiones distribuidas (DVCS) entra en escena.

Los clientes de DVCS no solo revisan la última instantánea del directorio, sino que también reflejan completamente el repositorio. Si el servidor deja de funcionar, el repositorio de cualquier cliente se puede volver a copiar en el servidor para restaurarlo. Cada pago es una copia de seguridad completa del repositorio. Git no depende del servidor central y es por eso que puedes realizar muchas operaciones cuando estás desconectado. Puede confirmar cambios, crear sucursales, ver registros y realizar otras operaciones cuando no esté conectado. Requiere conexión a la red solo para publicar sus cambios y tomar los últimos cambios.

Ventajas de Git

Gratis y de código abierto

Git se publica bajo la licencia de código abierto de GPL. Está disponible gratuitamente a través de Internet. Puede usar Git para administrar proyectos inmobiliarios sin pagar un solo centavo. Como es un código abierto, puede descargar su código fuente y también realizar cambios según sus requisitos.

Rápido y pequeño

Como la mayoría de las operaciones se realizan localmente, brinda un gran beneficio en términos de velocidad. Git no depende del servidor central; por eso, no hay necesidad de interactuar con el servidor remoto para cada operación. La parte central de Git está escrita en C, lo que evita los gastos generales de tiempo de ejecución asociados con otros lenguajes de alto nivel. Aunque Git refleja todo el repositorio, el tamaño de los datos en el lado del cliente es pequeño. Esto ilustra la eficiencia de Git para comprimir y almacenar datos en el lado del cliente.

Copia de seguridad implícita

Las posibilidades de perder datos son muy raras cuando hay varias copias de ellos. Los datos presentes en cualquier lado del cliente reflejan el repositorio, por lo tanto, se pueden usar en caso de falla o corrupción del disco.

Seguridad

Git utiliza una función hash criptográfica común llamada función hash segura (SHA1) para nombrar e identificar objetos dentro de su base de datos. Cada archivo y confirmación se suma a la verificación y se recupera mediante su suma de verificación en el momento de la finalización de la compra. Implica que es imposible cambiar el archivo, la fecha y el mensaje de confirmación y cualquier otro dato de la base de datos de Git sin conocer Git.

Sin necesidad de hardware potente

En el caso de CVCS, el servidor central debe ser lo suficientemente potente para atender las solicitudes de todo el equipo. Para equipos más pequeños, no es un problema, pero a medida que crece el tamaño del equipo, las limitaciones de hardware del servidor pueden ser un cuello de botella en el rendimiento. En el caso de DVCS, los desarrolladores no interactúan con el servidor a menos que necesiten enviar o recibir cambios. Todo el trabajo pesado ocurre en el lado del cliente, por lo que el hardware del servidor puede ser muy simple.

Ramificación más fácil

CVCS utiliza un mecanismo de copia económico. Si creamos una nueva rama, copiará todos los códigos en la nueva rama, por lo que lleva mucho tiempo y no es eficiente. Además, la eliminación y fusión de sucursales en CVCS es complicada y requiere mucho tiempo. Pero la gestión de sucursales con Git es muy sencilla. Solo lleva unos segundos crear, eliminar y fusionar sucursales.

Terminologías de DVCS

Repositorio local

Cada herramienta VCS proporciona un lugar de trabajo privado como copia de trabajo. Los desarrolladores realizan cambios en su lugar de trabajo privado y, después de la confirmación, estos cambios pasan a formar parte del repositorio. Git va un paso más allá al proporcionarles una copia privada de todo el repositorio. Los usuarios pueden realizar muchas operaciones con este repositorio, como agregar archivos, eliminar archivos, renombrar archivos, mover archivos, confirmar cambios y muchas más.

Directorio de trabajo y área de ensayo o índice

El directorio de trabajo es el lugar donde se desprotegen los archivos. En otros CVCS, los desarrolladores generalmente realizan modificaciones y confirman sus cambios directamente en el repositorio. Pero Git usa una estrategia diferente. Git no rastrea todos y cada uno de los archivos modificados. Cada vez que confirma una operación, Git busca los archivos presentes en el área de

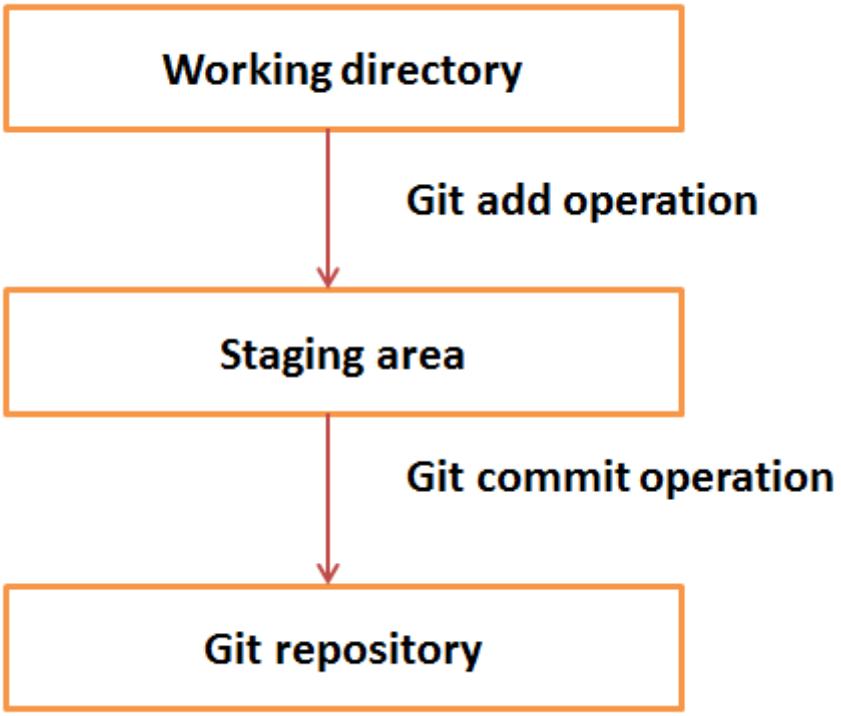
preparación. Solo los archivos presentes en el área de preparación se consideran para la confirmación y no todos los archivos modificados.

Veamos el flujo de trabajo básico de Git.

Paso 1 : modifica un archivo del directorio de trabajo.

Paso 2 : agrega estos archivos al área de preparación.

Paso 3 : realiza una operación de confirmación que mueve los archivos del área de preparación. Después de la operación de inserción, almacena los cambios de forma permanente en el repositorio de Git.



Suponga que modificó dos archivos, a saber, "sort.c" y "search.c" y desea dos confirmaciones diferentes para cada operación. Puede agregar un archivo en el área de preparación y confirmar. Después de la primera confirmación, repita el mismo procedimiento para otro archivo.

```
# First commit
[bash]$ git add sort.c

# adds file to the staging area
[bash]$ git commit –m “Added sort operation”

# Second commit
[bash]$ git add search.c

# adds file to the staging area
[bash]$ git commit –m “Added search operation”
```

Blobs

Blob significa **B** inary **L** arge **Object** . Cada versión de un archivo está representada por blob. Un blob contiene los datos del archivo, pero no contiene ningún metadato sobre el archivo. Es un archivo binario, y en la base de datos de Git, se nombra como hash SHA1 de ese archivo. En Git, los archivos no se direccionan por nombres. Todo está dirigido al contenido.

Trees

Tree es un objeto, que representa un directorio. Contiene blobs y otros subdirectorios. Un árbol es un archivo binario que almacena referencias a blobs y árboles que también se nombran como hash **SHA1** del objeto de árbol.

Commits

Commit contiene el estado actual del repositorio. Una confirmación también se nombra mediante el hash **SHA1 .** Puede considerar un objeto de confirmación como un nodo de la lista vinculada. Cada objeto de confirmación tiene un puntero al objeto de confirmación principal. Desde una confirmación determinada, puede retroceder mirando el puntero principal para ver el historial de la confirmación. Si una confirmación tiene varias confirmaciones principales, esa confirmación en particular se ha creado mediante la fusión de dos ramas.

Branches

Las ramas se utilizan para crear otra línea de desarrollo. De forma predeterminada, Git tiene una rama maestra, que es la misma que la troncal en Subversion. Por lo general, se crea una rama para trabajar en una nueva característica. Una vez que se completa la función, se fusiona nuevamente con la rama maestra y eliminamos la rama. Cada rama está referenciada por HEAD, que apunta a la última confirmación en la rama. Cada vez que realiza una confirmación, HEAD se actualiza con la última confirmación.

Tags

La etiqueta asigna un nombre significativo con una versión específica en el repositorio. Las etiquetas son muy similares a las ramas, pero la diferencia es que las etiquetas son inmutables. Significa que la etiqueta es una rama, que nadie tiene la intención de modificar. Una vez que se crea una etiqueta para una confirmación en particular, incluso si crea una nueva confirmación, no se actualizará. Por lo general, los desarrolladores crean etiquetas para lanzamientos de productos.

Clone

La operación de clonación crea la instancia del repositorio. La operación de clonación no solo verifica la copia de trabajo, sino que también refleja el repositorio completo. Los usuarios pueden realizar muchas operaciones con este repositorio local. La única vez que interviene la red es cuando se sincronizan las instancias del repositorio.

Pull

La operación de extracción copia los cambios de una instancia de repositorio remoto a uno local. La operación de extracción se utiliza para la sincronización entre dos instancias de repositorio. Esto es lo mismo que la operación de actualización en Subversion.

Push

La operación Push copia los cambios de una instancia de repositorio local a una remota. Esto se usa para almacenar los cambios de forma permanente en el repositorio de Git. Esto es lo mismo que la operación de confirmación en Subversion.

HEAD

HEAD es un puntero, que siempre apunta a la última confirmación en la rama. Cada vez que realiza una confirmación, HEAD se actualiza con la última confirmación. Los jefes de las ramas se almacenan en **el directorio .git/refs/heads/** .

```
[CentOS]$ ls -1 .git/refs/heads/
master

[CentOS]$ cat .git/refs/heads/master
570837e7d58fa4bccd86cb575d884502188b0c49
```

Revision

La revisión representa la versión del código fuente. Las revisiones en Git están representadas por confirmaciones. Estos compromisos se identifican mediante hashes seguros **SHA1 .**

URL

La URL representa la ubicación del repositorio de Git. La URL de Git se almacena en el archivo de configuración.

```
[tom@CentOS tom_repo]$ pwd
/home/tom/tom_repo

[tom@CentOS tom_repo]$ cat .git/config
[core]
repositoryformatversion = 0
filemode = true
bare = false
logallrefupdates = true
[remote "origin"]
url = gituser@git.server.com:project.git
fetch = +refs/heads/*:refs/remotes/origin/*
```

Git - Configuración del entorno

Antes de poder usar Git, debe instalar y realizar algunos cambios de configuración básicos. A continuación se muestran los pasos para instalar el cliente Git en Ubuntu y Centos Linux.

Instalación de Cliente Git

Si está utilizando la distribución GNU/Linux base de Debian, entonces el comando **apt-get** hará lo necesario.

```
[ubuntu ~]$ sudo apt-get install git-core
[sudo] password for ubuntu:

[ubuntu ~]$ git --version
git version 1.8.1.2
```

Y si está utilizando una distribución GNU/Linux basada en RPM, utilice el comando **yum como se indica.**

```
[CentOS ~]$
su -
Password:

[CentOS ~]# yum -y install git-core

[CentOS ~]# git --version
git version 1.7.1
```

Personalizar el entorno de Git

Git proporciona la herramienta de configuración de git, que le permite establecer variables de configuración. Git almacena todas las configuraciones globales en el archivo **.gitconfig** , que se encuentra en su directorio de inicio. Para establecer estos valores de configuración como globales, agregue la opción **--global** y, si omite la opción **--global** , sus configuraciones son específicas para el repositorio de Git actual.

También puede configurar la configuración de todo el sistema. Git almacena estos valores en el **archivo /etc/gitconfig** , que contiene la configuración para cada usuario y repositorio en el sistema. Para establecer estos valores, debe tener derechos de root y usar la opción **--system** .

Cuando el código anterior se compila y ejecuta, produce el siguiente resultado:

Configuración de nombre de usuario

Git utiliza esta información para cada confirmación.

```
[jerry@CentOS project]$ git config --global user.name "Jerry
Mouse"
```

Configuración de identificación de correo electrónico

Git utiliza esta información para cada confirmación.

```
[jerry@CentOS project]$ git config --global user.email
"jerry@tutorialspoint.com"
```

Evite fusionar confirmaciones para extraer

Obtiene los últimos cambios de un repositorio remoto y, si estos cambios son divergentes, Git crea confirmaciones de combinación de forma predeterminada. Podemos evitar esto a través de la siguiente configuración.

```
jerry@CentOS project]$ git config --global
branch.autosetuprebase always
```

Resaltado de color

Los siguientes comandos habilitan el resaltado de colores para Git en la consola.

```
[jerry@CentOS project]$ git config --global color.ui true

[jerry@CentOS project]$ git config --global color.status auto

[jerry@CentOS project]$ git config --global color.branch auto
```

Configuración del editor predeterminado

De forma predeterminada, Git usa el editor predeterminado del sistema, que se toma de la variable de entorno VISUAL o EDITOR. Podemos configurar uno diferente usando git config.

```
[jerry@CentOS project]$ git config --global core.editor vim
```

Configuración de la herramienta de combinación predeterminada

Git no proporciona una herramienta de combinación predeterminada para integrar cambios conflictivos en su árbol de trabajo. Podemos configurar la herramienta de combinación predeterminada habilitando las siguientes configuraciones.

```
[jerry@CentOS project]$ git config --global merge.tool
vimdiff
```

Listado de configuraciones de Git

Para verificar la configuración de Git del repositorio local, use el comando **git config –list** como se indica a continuación.

```
[jerry@CentOS ~]$ git config --list
```

El comando anterior producirá el siguiente resultado.

```
user.name=Jerry Mouse
user.email=jerry@tutorialspoint.com
push.default=nothing
branch.autosetuprebase=always
color.ui=true
color.status=auto
color.branch=auto
core.editor=vim
```

```
merge.tool=vimdiff
```

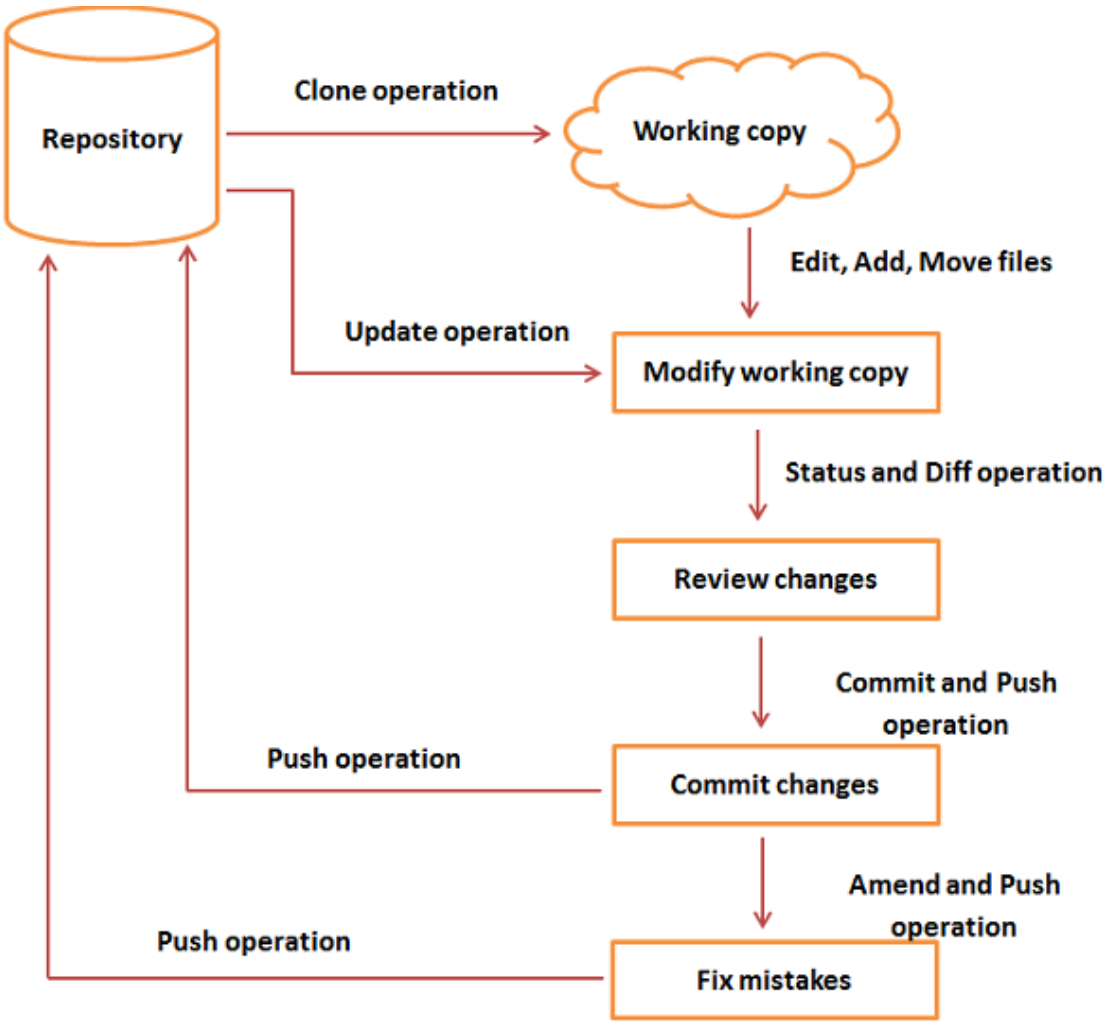
Git - Ciclo de vida

En este capítulo, discutiremos el ciclo de vida de Git. En capítulos posteriores, cubriremos los comandos de Git para cada operación.

El flujo de trabajo general es el siguiente:

- Clona el repositorio de Git como una copia de trabajo.
- Usted modifica la copia de trabajo agregando/editando archivos.
- Si es necesario, también actualiza la copia de trabajo tomando los cambios de otros desarrolladores.
- Revisas los cambios antes de confirmar.
- Confirmas cambios. Si todo está bien, envía los cambios al repositorio.
- Después de la confirmación, si se da cuenta de que algo anda mal, entonces corrige la última confirmación y envía los cambios al repositorio.

A continuación, se muestra la representación pictórica del flujo de trabajo.



Git - Crear operación

En este capítulo, veremos cómo crear un repositorio Git remoto; de ahora en adelante, nos referiremos a él como Servidor Git. Necesitamos un servidor Git para permitir la colaboración en equipo.

Crear nuevo usuario

```
# add new group
[root@CentOS ~]# groupadd dev

# add new user
[root@CentOS ~]# useradd -G devs -d /home/gituser -m -s
/bin/bash gituser

# change password
[root@CentOS ~]# passwd gituser
```

El comando anterior producirá el siguiente resultado.

```
Changing password for user gituser.
New password:
Retype new password:
passwd: all authentication token updated successfully.
```

Crear un repositorio básico

Inicialicemos un nuevo repositorio usando el comando **init seguido de la opción --bare** . Inicializa el repositorio sin un directorio de trabajo. Por convención, el repositorio básico debe llamarse **.git** .

```
[gituser@CentOS ~]$ pwd
/home/gituser

[gituser@CentOS ~]$ mkdir project.git

[gituser@CentOS ~]$ cd project.git/

[gituser@CentOS project.git]$ ls

[gituser@CentOS project.git]$ git --bare init
Initialized empty Git repository in /home/gituser-
m/project.git/

[gituser@CentOS project.git]$ ls
branches config description HEAD hooks info objects refs
```

Generar par de claves RSA pública/privada

Recorramos el proceso de configuración de un servidor Git, la utilidad **ssh-keygen** genera un par de claves RSA pública/privada, que usaremos para la autenticación del usuario.

Abra una terminal e ingrese el siguiente comando y simplemente presione enter para cada entrada. Después de completar con éxito, creará un directorio **.ssh** dentro del directorio de inicio.

```
tom@CentOS ~]$ pwd
/home/tom

[tom@CentOS ~]$ ssh-keygen
```

El comando anterior producirá el siguiente resultado.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tom/.ssh/id_rsa):
Press Enter Only
Created directory '/home/tom/.ssh'.
Enter passphrase (empty for no passphrase): ---------------->
Press Enter Only
Enter same passphrase again: ------------------------------->
Press Enter Only
Your identification has been saved in /home/tom/.ssh/id_rsa.
Your public key has been saved in /home/tom/.ssh/id_rsa.pub.
The key fingerprint is:
df:93:8c:a1:b8:b7:67:69:3a:1f:65:e8:0e:e9:25:a1 tom@CentOS
The key's randomart image is:
+--[ RSA 2048]----+
|  |
|  |
|  |
|
.
|
| Soo |
| o*B. |
| E = *.= |
| oo==. . |
| ..+Oo
|
+-----------------+
```

ssh-keygen ha generado dos claves, la primera es privada (es decir, id_rsa) y la segunda es pública (es decir, id_rsa.pub).

Nota: nunca comparta su CLAVE PRIVADA con otras personas.

Adición de claves a authorized_keys

Supongamos que hay dos desarrolladores trabajando en un proyecto, a saber, Tom y Jerry. Ambos usuarios han generado claves públicas. Veamos cómo usar estas claves para la autenticación.

Tom agregó su clave pública al servidor usando el comando **ssh-copy-id** como se indica a continuación:

```
[tom@CentOS ~]$ pwd
/home/tom
```

```
[tom@CentOS ~]$ ssh-copy-id -i ~/.ssh/id_rsa.pub
gituser@git.server.com
```

El comando anterior producirá el siguiente resultado.

```
gituser@git.server.com's password:
Now try logging into the machine, with "ssh
'gituser@git.server.com'", and check in:
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't
expecting.
```

De manera similar, Jerry agregó su clave pública al servidor mediante el comando ssh-copy-id.

```
[jerry@CentOS ~]$ pwd
/home/jerry

[jerry@CentOS ~]$ ssh-copy-id -i ~/.ssh/id_rsa
gituser@git.server.com
```

El comando anterior producirá el siguiente resultado.

```
gituser@git.server.com's password:
Now try logging into the machine, with "ssh
'gituser@git.server.com'", and check in:
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't
expecting.
```

Enviar cambios al repositorio

Hemos creado un repositorio básico en el servidor y permitimos el acceso a dos usuarios. A partir de ahora, Tom y Jerry pueden enviar sus cambios al repositorio agregándolo como un control remoto.

El comando Git init crea el directorio **.git** para almacenar metadatos sobre el repositorio cada vez que lee la configuración del archivo **.git/config .**

Tom crea un nuevo directorio, agrega el archivo README y confirma su cambio como confirmación inicial. Después de la confirmación, verifica el mensaje de confirmación ejecutando el comando **git log .**

```
[tom@CentOS ~]$ pwd
/home/tom

[tom@CentOS ~]$ mkdir tom_repo

[tom@CentOS ~]$ cd tom_repo/

[tom@CentOS tom_repo]$ git init
Initialized empty Git repository in /home/tom/tom_repo/.git/

[tom@CentOS tom_repo]$ echo 'TODO: Add contents for README' >
README
```

```
[tom@CentOS tom_repo]$ git status -s
?? README

[tom@CentOS tom_repo]$ git add .

[tom@CentOS tom_repo]$ git status -s
A README

[tom@CentOS tom_repo]$ git commit -m 'Initial commit'
```

El comando anterior producirá el siguiente resultado.

```
[master (root-commit) 19ae206] Initial commit
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 README
```

Tom verifica el mensaje de registro ejecutando el comando git log.

```
[tom@CentOS tom_repo]$ git log
```

El comando anterior producirá el siguiente resultado.

```
commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Tom envió sus cambios al repositorio local. Ahora es el momento de enviar los cambios al repositorio remoto. Pero antes de eso, tenemos que agregar el repositorio como remoto, esta es una operación de una sola vez. Después de esto, puede enviar los cambios al repositorio remoto de forma segura.

Nota : de forma predeterminada, Git envía solo a las ramas coincidentes: para cada rama que existe en el lado local, el lado remoto se actualiza si ya existe una rama con el mismo nombre allí. En nuestros tutoriales, cada vez que insertamos cambios en la rama **maestra de origen** , use el nombre de rama adecuado según sus requisitos.

```
[tom@CentOS tom_repo]$ git remote add origin
gituser@git.server.com:project.git

[tom@CentOS tom_repo]$ git push origin master
```

El comando anterior producirá el siguiente resultado.

```
Counting objects: 3, done.
Writing objects: 100% (3/3), 242 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new branch]
master -> master
```

Ahora, los cambios se confirmaron con éxito en el repositorio remoto.

Git - Operación de clonación

Tenemos un repositorio básico en el servidor Git y Tom también lanzó su primera versión. Ahora, Jerry puede ver sus cambios. La operación Clonar crea una instancia del repositorio remoto.

Jerry crea un nuevo directorio en su directorio de inicio y realiza la operación de clonación.

```
[jerry@CentOS ~]$ mkdir jerry_repo

[jerry@CentOS ~]$ cd jerry_repo/

[jerry@CentOS jerry_repo]$ git clone
gituser@git.server.com:project.git
```

El comando anterior producirá el siguiente resultado.

```
Initialized empty Git repository in
/home/jerry/jerry_repo/project/.git/
remote: Counting objects: 3, done.
Receiving objects: 100% (3/3), 241 bytes, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
```

Jerry cambia el directorio al nuevo repositorio local y enumera el contenido de su directorio.

```
[jerry@CentOS jerry_repo]$ cd project/

[jerry@CentOS jerry_repo]$ ls
README
```

Git - Realizar cambios

Jerry clona el repositorio y decide implementar operaciones básicas de cadenas. Así que crea el archivo string.c. Después de agregar el contenido, string.c se verá así:

```
#include <stdio.h>

int my_strlen(char *s)
{
   char *p = s;

   while (*p)
      ++p;

   return (p - s);
}

int main(void)
{
```

```
   int i;
   char *s[] =
   {
      "Git tutorials",
      "Tutorials Point"
   };

   for (i = 0; i < 2; ++i)

   printf("string lenght of %s = %d\n", s[i],
my_strlen(s[i]));

   return 0;
}
```

Compiló y probó su código y todo funciona bien. Ahora, puede agregar con seguridad estos cambios al repositorio.

La operación de agregar Git agrega un archivo al área de ensayo.

```
[jerry@CentOS project]$ git status -s
?? string
?? string.c

[jerry@CentOS project]$ git add string.c
```

Git muestra un signo de interrogación antes de los nombres de los archivos. Obviamente, estos archivos no son parte de Git y es por eso que Git no sabe qué hacer con estos archivos. Es por eso que Git muestra un signo de interrogación antes de los nombres de los archivos.

Jerry ha agregado el archivo al área de almacenamiento, el comando de estado de git mostrará los archivos presentes en el área de preparación.

```
[jerry@CentOS project]$ git status -s
A string.c
?? string
```

Para confirmar los cambios, usó el comando git commit seguido de la opción –m. Si omitimos la opción –m. Git abrirá un editor de texto donde podemos escribir un mensaje de compromiso de varias líneas.

```
[jerry@CentOS project]$ git commit -m 'Implemented my_strlen
function'
```

El comando anterior producirá el siguiente resultado:

```
[master cbe1249] Implemented my_strlen function
1 files changed, 24 insertions(+), 0 deletions(-)
create mode 100644 string.c
```

Después de comprometerse a ver los detalles del registro, ejecuta el comando git log. Mostrará la información de todas las confirmaciones con su ID de confirmación, autor de confirmación, fecha de confirmación y **hash SHA-1** de confirmación.

```
[jerry@CentOS project]$ git log
```

El comando anterior producirá el siguiente resultado:

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Git - Revisar cambios

Después de ver los detalles de la confirmación, Jerry se da cuenta de que la longitud de la cadena no puede ser negativa, por eso decide cambiar el tipo de retorno de la función my_strlen.

Jerry usa el comando **git log para ver los detalles del registro.**

```
[jerry@CentOS project]$ git log
```

El comando anterior producirá el siguiente resultado.

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function
```

Jerry usa el comando **git show** para ver los detalles de la confirmación. El comando git show toma el ID de confirmación **SHA-1** como parámetro.

```
[jerry@CentOS project]$ git show
cbe1249b140dad24b2c35b15cc7e26a6f02d2277
```

El comando anterior producirá el siguiente resultado:

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function


diff --git a/string.c b/string.c
new file mode 100644
index 0000000..187afb9
--- /dev/null
+++ b/string.c
@@ -0,0 +1,24 @@
+#include <stdio.h>
```

```
+
+int my_strlen(char *s)
+{
   +
   char *p = s;
   +
   +
   while (*p)
   + ++p;
   + return (p -s );
   +
}
+
```

Cambia el tipo de retorno de la función de int a size_t. Después de probar el código, revisa sus cambios ejecutando el comando **git diff .**

```
[jerry@CentOS project]$ git diff
```

El comando anterior producirá el siguiente resultado:

```
diff --git a/string.c b/string.c
index 187afb9..7da2992 100644
--- a/string.c
+++ b/string.c
@@ -1,6 +1,6 @@
#include <stdio.h>

-int my_strlen(char *s)
+size_t my_strlen(char *s)
{
   char *p = s;
   @@ -18,7 +18,7 @@ int main(void)
};
for (i = 0; i < 2; ++i)
{
   - printf("string lenght of %s = %d\n", s[i],
my_strlen(s[i]));
   + printf("string lenght of %s = %lu\n", s[i],
my_strlen(s[i]));
   return 0;
}
```

Git diff muestra el signo **'+'** antes de las líneas, que se agregaron recientemente y **'-'** para las líneas eliminadas.

Git - Confirmar cambios

Jerry ya ha confirmado los cambios y quiere corregir su última confirmación. En este caso, la operación de **modificación de git** ayudará. La operación de modificación cambia la última confirmación, incluido su mensaje de confirmación; crea una nueva ID de confirmación.

Antes de modificar la operación, comprueba el registro de confirmación.

```
[jerry@CentOS project]$ git log
```

El comando anterior producirá el siguiente resultado.

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Jerry confirma los nuevos cambios con la operación de modificación y visualiza el registro de confirmación.

```
[jerry@CentOS project]$ git status -s
M string.c
?? string

[jerry@CentOS project]$ git add string.c

[jerry@CentOS project]$ git status -s
M string.c
?? string

[jerry@CentOS project]$ git commit --amend -m 'Changed return
type of my_strlen to size_t'
[master d1e19d3] Changed return type of my_strlen to size_t
1 files changed, 24 insertions(+), 0 deletions(-)
create mode 100644 string.c
```

Ahora, git log mostrará un nuevo mensaje de confirmación con una nueva ID de confirmación:

```
[jerry@CentOS project]$ git log
```

El comando anterior producirá el siguiente resultado.

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530
```

```
Initial commit
```

Git - Operación de inserción

Jerry modificó su última confirmación mediante la operación de modificación y está listo para impulsar los cambios. La operación Push almacena datos de forma permanente en el repositorio de Git. Después de una operación de inserción exitosa, otros desarrolladores pueden ver los cambios de Jerry.

Ejecuta el comando git log para ver los detalles de la confirmación.

```
[jerry@CentOS project]$ git log
```

El comando anterior producirá el siguiente resultado:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t
```

Antes de la operación de inserción, quiere revisar sus cambios, por lo que usa el comando **git show** para revisar sus cambios.

```
[jerry@CentOS project]$ git show
d1e19d316224cddc437e3ed34ec3c931ad803958
```

El comando anterior producirá el siguiente resultado:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t

diff --git a/string.c b/string.c
new file mode 100644
index 0000000..7da2992
--- /dev/null
+++ b/string.c
@@ -0,0 +1,24 @@
+#include <stdio.h>
+
+size_t my_strlen(char *s)
+
{
   +
   char *p = s;
   +
   +
   while (*p)
   + ++p;
   + return (p -s );
   +
```

```
}
+
+int main(void)
+
{
   + int i;
   + char *s[] =
   {
      + "Git tutorials",
      + "Tutorials Point"
      +
   };
   +
   +
   +
   for (i = 0; i < 2; ++i)
   printf("string lenght of %s = %lu\n", s[i],
my_strlen(s[i]));
   +
   +
   return 0;
   +
}
```

Jerry está contento con sus cambios y está listo para impulsarlos.

```
[jerry@CentOS project]$ git push origin master
```

El comando anterior producirá el siguiente resultado:

```
Counting objects: 4, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 517 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
19ae206..d1e19d3 master -> master
```

Los cambios de Jerry se han enviado con éxito al repositorio; ahora otros desarrolladores pueden ver sus cambios realizando una operación de clonación o actualización.

Git - Operación de actualización

Modificar función existente

Tom realiza la operación de clonación y encuentra una nueva cadena de archivo.c. Quiere saber quién agregó este archivo al repositorio y con qué propósito, por lo que ejecuta el comando **git log .**

```
[tom@CentOS ~]$ git clone gituser@git.server.com:project.git
```

El comando anterior producirá el siguiente resultado:

```
Initialized empty Git repository in /home/tom/project/.git/
```

```
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (4/4), done.
Receiving objects: 100% (6/6), 726 bytes, done.
remote: Total 6 (delta 0), reused 0 (delta 0)
```

La operación Clonar creará un nuevo directorio dentro del directorio de trabajo actual. Cambia el directorio al directorio recién creado y ejecuta el comando **git log .**

```
[tom@CentOS ~]$ cd project/

[tom@CentOS project]$ git log
```

El comando anterior producirá el siguiente resultado:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Después de observar el registro, se da cuenta de que Jerry agregó el archivo string.c para implementar operaciones básicas con cadenas. Tiene curiosidad sobre el código de Jerry. Así que abre string.c en el editor de texto e inmediatamente encuentra un error. En la función my_strlen, Jerry no usa un puntero constante. Entonces, decide modificar el código de Jerry. Después de la modificación, el código se ve de la siguiente manera:

```
[tom@CentOS project]$ git diff
```

El comando anterior producirá el siguiente resultado:

```
diff --git a/string.c b/string.c
index 7da2992..32489eb 100644
--- a/string.c
+++ b/string.c
@@ -1,8 +1,8 @@
#include <stdio.h>
-size_t my_strlen(char *s)
+size_t my_strlen(const char *s)
{
   - char *p = s;
   + const char *p = s;
   while (*p)
   ++p;
}
```

Después de la prueba, confirma su cambio.

```
[tom@CentOS project]$ git status -s
M string.c
?? string

[tom@CentOS project]$ git add string.c

[tom@CentOS project]$ git commit -m 'Changed char pointer to
const char pointer'
[master cea2c00] Changed char pointer to const char pointer
1 files changed, 2 insertions(+), 2 deletions(-)

[tom@CentOS project]$ git log
```

El comando anterior producirá el siguiente resultado:

```
commit cea2c000f53ba99508c5959e3e12fff493b
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 08:32:07 2013 +0530

Changed char pointer to const char pointer


commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530
Initial commit
```

Tom usa el comando git push para impulsar sus cambios.

```
[tom@CentOS project]$ git push origin master
```

El comando anterior producirá el siguiente resultado:

```
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 336 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
d1e19d3..cea2c00 master -> master
```

Agregar nueva función

Mientras tanto, Jerry decide implementar la funcionalidad **de comparación de cadenas** . Entonces modifica string.c. Después de la modificación, el archivo se ve de la siguiente manera:

```
[jerry@CentOS project]$ git diff
```

El comando anterior producirá el siguiente resultado:

```
index 7da2992..bc864ed 100644
--- a/string.c
+++ b/string.c
30Git Tutorials
@@ -9,9 +9,20 @@ size_t my_strlen(char *s)
return (p -s );
}
+char *my_strcpy(char *t, char *s)
+
{
   +
   char *p = t;
   +
   + while (*t++ = *s++)
   + ;
   +
   +
   return p;
   +
}
+
int main(void)
{
   int i;
   +
   char p1[32];
   char *s[] =
   {
      "Git tutorials",
      "Tutorials Point"
      @@ -20,5 +31,7 @@ int main(void)
      for (i = 0; i < 2; ++i)
      printf("string lenght of %s = %lu\n", s[i],
my_strlen(s[i]));
      +
      printf("%s\n", my_strcpy(p1, "Hello, World !!!"));
      +
      return 0;
   }
}
```

Después de la prueba, está listo para impulsar su cambio.

```
[jerry@CentOS project]$ git status -s
M string.c
?? string

[jerry@CentOS project]$ git add string.c

[jerry@CentOS project]$ git commit -m "Added my_strcpy
function"
[master e944e5a] Added my_strcpy function
1 files changed, 13 insertions(+), 0 deletions(-)
```

Antes de la operación de inserción, verifica la confirmación al ver los mensajes de registro.

```
[jerry@CentOS project]$ git log
```

El comando anterior producirá el siguiente resultado:

```
commit e944e5aab74b26e7447d3281b225309e4e59efcd
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:41:42 2013 +0530

Added my_strcpy function


commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Jerry está contento con los cambios y quiere impulsar sus cambios.

```
[jerry@CentOS project]$ git push origin master
```

El comando anterior producirá el siguiente resultado:

```
To gituser@git.server.com:project.git
! [rejected]
master -> master (non-fast-forward)
error: failed to push some refs to
'gituser@git.server.com:project.git'
To prevent you from losing history, non-fast-forward updates
were rejected
Merge the remote changes before pushing again. See the 'Note
about
fast-forwards' section of 'git push --help' for details.
```

Pero Git no le permite a Jerry impulsar sus cambios. Porque Git identificó que el repositorio remoto y el repositorio local de Jerry no están sincronizados. Debido a esto, puede perder la historia del proyecto. Para evitar este lío, Git falló en esta operación. Ahora, Jerry primero tiene que actualizar el repositorio local y, solo después, puede impulsar sus propios cambios.

Obtener los últimos cambios

Jerry ejecuta el comando git pull para sincronizar su repositorio local con el remoto.

```
[jerry@CentOS project]$ git pull
```

El comando anterior producirá el siguiente resultado:

```
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From git.server.com:project
d1e19d3..cea2c00 master -> origin/master
First, rewinding head to replay your work on top of it...
Applying: Added my_strcpy function
```

Después de la operación de extracción, Jerry comprueba los mensajes de registro y encuentra los detalles de la confirmación de Tom con el ID de confirmación **cea2c000f53ba99508c5959e3e12fff493ba6f69 .**

```
[jerry@CentOS project]$ git log
```

El comando anterior producirá el siguiente resultado:

```
commit e86f0621c2a3f68190bba633a9fe6c57c94f8e4f
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:41:42 2013 +0530

Added my_strcpy function


commit cea2c000f53ba99508c5959e3e12fff493ba6f69
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 08:32:07 2013 +0530

Changed char pointer to const char pointer


commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530
Initial commit
```

Ahora, el repositorio local de Jerry está completamente sincronizado con el repositorio remoto. Para que pueda impulsar sus cambios de manera segura.

```
[jerry@CentOS project]$ git push origin master
```

El comando anterior producirá el siguiente resultado:

```
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 455 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
```

```
To gituser@git.server.com:project.git
cea2c00..e86f062 master -> master
```

Git - Operación de almacenamiento

Suponga que está implementando una nueva función para su producto. Su código está en progreso y de repente llega una escalada del cliente. Debido a esto, debe dejar de lado el trabajo de su nueva función durante unas horas. No puede confirmar su código parcial y tampoco puede descartar sus cambios. Por lo tanto, necesita un espacio temporal, donde pueda almacenar sus cambios parciales y luego confirmarlos.

En Git, la operación de almacenamiento toma los archivos rastreados modificados, los cambios por etapas y los guarda en una pila de cambios inconclusos que puede volver a aplicar en cualquier momento.

```
[jerry@CentOS project]$ git status -s
M string.c
?? string
```

Ahora, desea cambiar de sucursal para escalar clientes, pero no desea comprometerse en lo que ha estado trabajando todavía; así que guardarás los cambios. Para insertar un nuevo alijo en su pila, ejecute el comando **git stash .**

```
[jerry@CentOS project]$ git stash
Saved working directory and index state WIP on master:
e86f062 Added my_strcpy function
HEAD is now at e86f062 Added my_strcpy function
```

Ahora, su directorio de trabajo está limpio y todos los cambios se guardan en una pila. Verifiquémoslo con el comando **git status .**

```
[jerry@CentOS project]$ git status -s
?? string
```

Ahora puede cambiar de sucursal de manera segura y trabajar en otro lugar. Podemos ver una lista de cambios ocultos usando el comando **git stash list .**

```
[jerry@CentOS project]$ git stash list
stash@{0}: WIP on master: e86f062 Added my_strcpy function
```

Supongamos que ha resuelto la escalada del cliente y está de vuelta en su nueva función buscando su código a medio hacer, simplemente ejecute el comando **git stash pop** para eliminar los cambios de la pila y colocarlos en el directorio de trabajo actual.

```
[jerry@CentOS project]$ git status -s
?? string

[jerry@CentOS project]$ git stash pop
```

El comando anterior producirá el siguiente resultado:

```
# On branch master
# Changed but not updated:
```

```
# (use "git add ..." to update what will be committed)
# (use "git checkout -- ..." to discard changes in working
directory)
#
#
modified: string.c
#
# Untracked files:
# (use "git add ..." to include in what will be committed)
#
#
string
no changes added to commit (use "git add" and/or "git commit
-a")
Dropped refs/stash@{0}
(36f79dfedae4ac20e2e8558830154bd6315e72d4)

[jerry@CentOS project]$ git status -s
M string.c
?? string
```

Git - Operación de movimiento

Como sugiere el nombre, la operación de movimiento mueve un directorio o un archivo de una ubicación a otra. Tom decide mover el código fuente al directorio **src** . La estructura de directorio modificada aparecerá de la siguiente manera:

```
[tom@CentOS project]$ pwd
/home/tom/project

[tom@CentOS project]$ ls
README string string.c

[tom@CentOS project]$ mkdir src

[tom@CentOS project]$ git mv string.c src/

[tom@CentOS project]$ git status -s
R string.c -> src/string.c
?? string
```

Para que estos cambios sean permanentes, tenemos que enviar la estructura de directorios modificada al repositorio remoto para que otros desarrolladores puedan verlo.

```
[tom@CentOS project]$ git commit -m "Modified directory
structure"

[master 7d9ea97] Modified directory structure
1 files changed, 0 insertions(+), 0 deletions(-)
rename string.c => src/string.c (100%)
```

```
[tom@CentOS project]$ git push origin master
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 320 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
e86f062..7d9ea97 master -> master
```

En el repositorio local de Jerry, antes de la operación de extracción, mostrará la estructura de directorios anterior.

```
[jerry@CentOS project]$ pwd
/home/jerry/jerry_repo/project

[jerry@CentOS project]$ ls
README string string.c
```

Pero después de la operación de extracción, la estructura del directorio se actualizará. Ahora, Jerry puede ver el directorio **src** y el archivo presente dentro de ese directorio.

```
[jerry@CentOS project]$ git pull
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From git.server.com:project
e86f062..7d9ea97 master -> origin/master
First, rewinding head to replay your work on top of it...
Fast-forwarded master to
7d9ea97683da90bcdb87c28ec9b4f64160673c8a.

[jerry@CentOS project]$ ls
README src string

[jerry@CentOS project]$ ls src/
string.c
```

Git - Operación de cambio de nombre

Hasta ahora, tanto Tom como Jerry usaban comandos manuales para compilar su proyecto. Ahora, Jerry decide crear un Makefile para su proyecto y también le da un nombre propio al archivo “string.c”.

```
[jerry@CentOS project]$ pwd
/home/jerry/jerry_repo/project

[jerry@CentOS project]$ ls
README src

[jerry@CentOS project]$ cd src/
```

```
[jerry@CentOS src]$ git add Makefile

[jerry@CentOS src]$ git mv string.c string_operations.c

[jerry@CentOS src]$ git status -s
A Makefile
R string.c -> string_operations.c
```

Git muestra **R** antes del nombre del archivo para indicar que se ha cambiado el nombre del archivo.

Para la operación de confirmación, Jerry usó una marca que hace que git commit detecte automáticamente los archivos modificados.

```
[jerry@CentOS src]$ git commit -a -m 'Added Makefile and
renamed strings.c to
string_operations.c '

[master 94f7b26] Added Makefile and renamed strings.c to
string_operations.c
1 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 src/Makefile
rename src/{string.c => string_operations.c} (100%)
```

Después de confirmar, envía sus cambios al repositorio.

```
[jerry@CentOS src]$ git push origin master
```

El comando anterior producirá el siguiente resultado:

```
Counting objects: 6, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 396 bytes, done.
Total 4 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
7d9ea97..94f7b26 master -> master
```

Ahora, otros desarrolladores pueden ver estas modificaciones actualizando su repositorio local.

Git - Operación de eliminación

Tom actualiza su repositorio local y encuentra el binario compilado en el directorio **src** . Después de ver el mensaje de confirmación, se da cuenta de que Jerry agregó el binario compilado.

```
[tom@CentOS src]$ pwd
/home/tom/project/src

[tom@CentOS src]$ ls
Makefile string_operations string_operations.c

[tom@CentOS src]$ file string_operations
string_operations: ELF 64-bit LSB executable, x86-64, version
1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.18, not stripped
```

```
[tom@CentOS src]$ git log
commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary
```

VCS se utiliza para almacenar solo el código fuente y no los archivos binarios ejecutables. Entonces, Tom decide eliminar este archivo del repositorio. Para más operaciones, utiliza el comando **git rm** .

```
[tom@CentOS src]$ ls
Makefile string_operations string_operations.c

[tom@CentOS src]$ git rm string_operations
rm 'src/string_operations'

[tom@CentOS src]$ git commit -a -m "Removed executable
binary"

[master 5776472] Removed executable binary
1 files changed, 0 insertions(+), 0 deletions(-)
delete mode 100755 src/string_operations
```

Después de confirmar, envía sus cambios al repositorio.

```
[tom@CentOS src]$ git push origin master
```

El comando anterior producirá el siguiente resultado.

```
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 310 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
29af9d4..5776472 master -> master
```

Git - Corregir errores

Errar es humano. Entonces, cada VCS proporciona una función para corregir errores hasta cierto punto. Git proporciona una función que podemos usar para deshacer las modificaciones que se han realizado en el repositorio local.

Supongamos que el usuario accidentalmente hace algunos cambios en su repositorio local y luego quiere deshacer estos cambios. En tales casos, la operación de **reversión** juega un papel importante.

Revertir cambios no confirmados

Supongamos que Jerry modifica accidentalmente un archivo de su repositorio local. Pero él quiere deshacer su modificación. Para manejar esta situación, podemos usar el comando **git checkout .** Podemos usar este comando para revertir el contenido de un archivo.

```
[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git status -s
M string_operations.c

[jerry@CentOS src]$ git checkout string_operations.c

[jerry@CentOS src]$ git status –s
```

Además, podemos usar el comando **git checkout** para obtener un archivo eliminado del repositorio local. Supongamos que Tom elimina un archivo del repositorio local y queremos recuperar este archivo. Podemos lograr esto usando el mismo comando.

```
[tom@CentOS src]$ pwd
/home/tom/top_repo/project/src

[tom@CentOS src]$ ls -1
Makefile
string_operations.c

[tom@CentOS src]$ rm string_operations.c

[tom@CentOS src]$ ls -1
Makefile

[tom@CentOS src]$ git status -s
D string_operations.c
```

Git muestra la letra **D** antes del nombre del archivo. Esto indica que el archivo se eliminó del repositorio local.

```
[tom@CentOS src]$ git checkout string_operations.c

[tom@CentOS src]$ ls -1
Makefile
string_operations.c

[tom@CentOS src]$ git status -s
```

Nota : podemos realizar todas estas operaciones antes de confirmar.

Eliminar cambios del área de ensayo

Hemos visto que cuando realizamos una operación de agregar, los archivos se mueven del repositorio local al área de estado. Si un usuario modifica accidentalmente un archivo y lo agrega al área de preparación, puede revertir sus cambios usando el comando **git checkout .**

En Git, hay un puntero HEAD que siempre apunta a la última confirmación. Si desea deshacer un cambio del área preparada, puede usar el comando git checkout, pero con el comando checkout, debe proporcionar un parámetro adicional, es decir, el puntero HEAD. El parámetro de puntero de confirmación

adicional indica al comando git checkout que restablezca el árbol de trabajo y también que elimine los cambios por etapas.

Supongamos que Tom modifica un archivo de su repositorio local. Si vemos el estado de este archivo, mostrará que el archivo se modificó pero no se agregó al área de ensayo.

```
tom@CentOS src]$ pwd
/home/tom/top_repo/project/src
# Unmodified file

[tom@CentOS src]$ git status -s

# Modify file and view it's status.
[tom@CentOS src]$ git status -s
M string_operations.c

[tom@CentOS src]$ git add string_operations.c
```

El estado de Git muestra que el archivo está presente en el área de preparación, ahora retírelo usando el comando git checkout y vea el estado del archivo revertido.

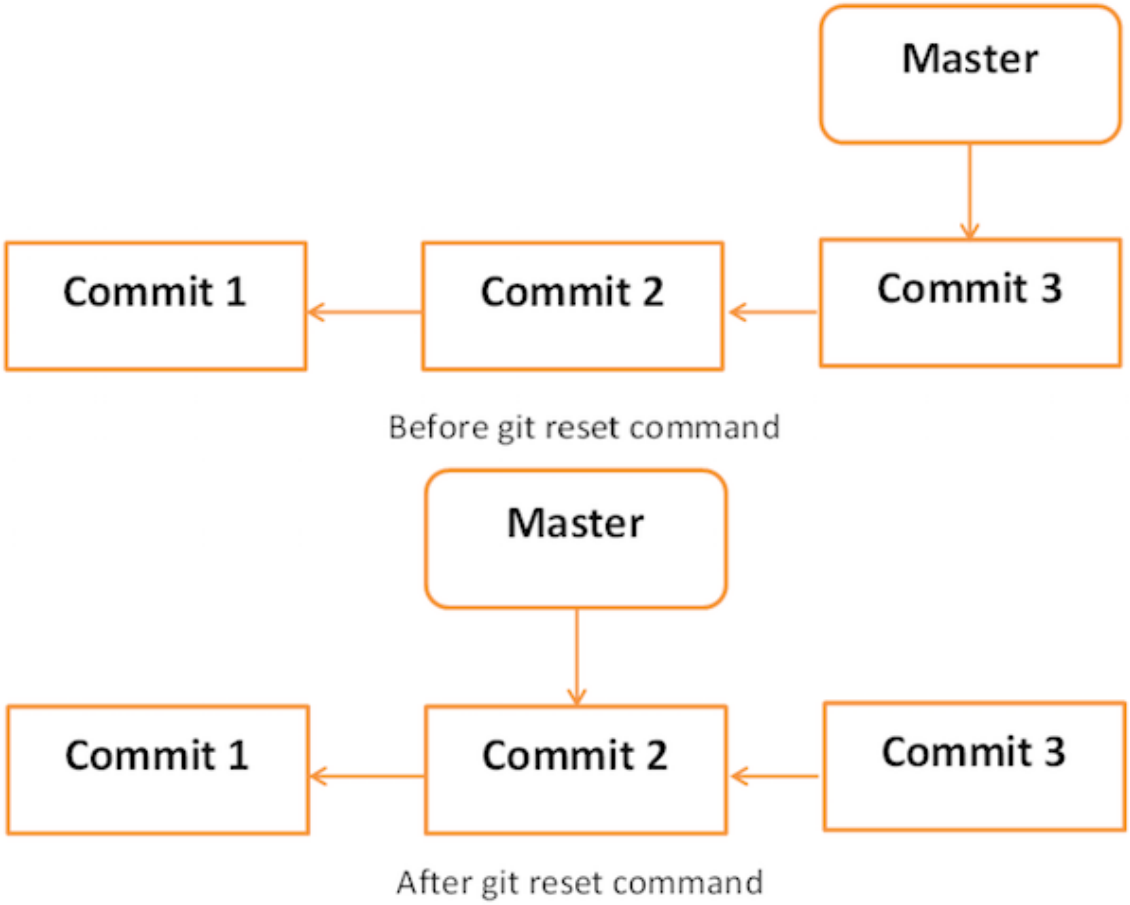
```
[tom@CentOS src]$ git checkout HEAD -- string_operations.c

[tom@CentOS src]$ git status -s
```

Mueva el puntero HEAD con Git Reset

Después de hacer algunos cambios, puede decidir eliminar estos cambios. El comando de restablecimiento de Git se usa para restablecer o revertir los cambios. Podemos realizar tres tipos diferentes de operaciones de reinicio.

El siguiente diagrama muestra la representación pictórica del comando de reinicio de Git.



Soft

Cada rama tiene un puntero HEAD, que apunta a la última confirmación. Si usamos el comando de restablecimiento de Git con la opción --soft seguida de la ID de confirmación, solo restablecerá el puntero HEAD sin destruir nada.

El archivo .git/refs/heads/master almacena el ID de confirmación del puntero HEAD. Podemos verificarlo usando el comando **git log -1 .**

```
[jerry@CentOS project]$ cat .git/refs/heads/master
577647211ed44fe2ae479427a0668a4f12ed71a1
```

Ahora, vea la ID de confirmación más reciente, que coincidirá con la ID de confirmación anterior.

```
[jerry@CentOS project]$ git log -2
```

El comando anterior producirá el siguiente resultado.

```
commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary


commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Jerry Mouse <jerry@tutorialspoint.com>
```

```
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary
```

Restablezcamos el puntero HEAD.

```
[jerry@CentOS project]$ git reset --soft HEAD~
```

Ahora, simplemente restablecemos el puntero HEAD una posición. Revisemos el contenido de **.git/refs/heads/master file** .

```
[jerry@CentOS project]$ cat .git/refs/heads/master
29af9d45947dc044e33d69b9141d8d2dad37cc62
```

Se cambió el ID de confirmación del archivo, ahora verifíquelo viendo los mensajes de confirmación.

```
jerry@CentOS project]$ git log -2
```

El comando anterior producirá el siguiente resultado.

```
commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary


commit 94f7b26005f856f1a1b733ad438e97a0cd509c1a
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 10:08:01 2013 +0530

Added Makefile and renamed strings.c to string_operations.c
```

mixed

Git reset con la opción --mixed revierte los cambios del área de preparación que aún no se han confirmado. Revierte los cambios solo desde el área de preparación. Los cambios reales realizados en la copia de trabajo del archivo no se ven afectados. El reinicio de Git predeterminado es equivalente al reinicio de git, mixto.

hard

Si usa la opción --hard con el comando de reinicio de Git, borrará el área de preparación; restablecerá el puntero HEAD a la última confirmación del ID de confirmación específico y también eliminará los cambios del archivo local.

Comprobemos el ID de confirmación.

```
[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git log -1
```

El comando anterior producirá el siguiente resultado.

```
commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
```

```
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary
```

Jerry modificó un archivo agregando un comentario de una sola línea al comienzo del archivo.

```
[jerry@CentOS src]$ head -2 string_operations.c
/* This line be removed by git reset operation */
#include <stdio.h>
```

Lo verificó usando el comando git status.

```
[jerry@CentOS src]$ git status -s
M string_operations.c
```

Jerry agrega el archivo modificado al área de ensayo y lo verifica con el comando git status.

```
[jerry@CentOS src]$ git add string_operations.c
[jerry@CentOS src]$ git status
```

El comando anterior producirá el siguiente resultado.

```
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
#
modified: string_operations.c
#
```

El estado de Git muestra que el archivo está presente en el área de ensayo. Ahora, reinicie HEAD con la opción --hard.

```
[jerry@CentOS src]$ git reset --hard
577647211ed44fe2ae479427a0668a4f12ed71a1

HEAD is now at 5776472 Removed executable binary
```

El comando de restablecimiento de Git tuvo éxito, lo que revertirá el archivo del área de preparación y eliminará cualquier cambio local realizado en el archivo.

```
[jerry@CentOS src]$ git status -s
```

El estado de Git muestra que el archivo se ha revertido del área de ensayo.

```
[jerry@CentOS src]$ head -2 string_operations.c
#include <stdio.h>
```

El comando principal también muestra que la operación de reinicio también eliminó los cambios locales.

Git - Operación de etiquetas

La operación de etiquetas permite dar nombres significativos a una versión específica en el repositorio. Supongamos que Tom y Jerry deciden etiquetar el código de su proyecto para que luego puedan acceder a él fácilmente.

Crear etiquetas

Etiquetemos el HEAD actual usando el comando **git tag .** Tom proporciona un nombre de etiqueta con la opción -a y proporciona un mensaje de etiqueta con la opción -m.

```
tom@CentOS project]$ pwd
/home/tom/top_repo/project

[tom@CentOS project]$ git tag -a 'Release_1_0' -m 'Tagged
basic string operation code' HEAD
```

Si desea etiquetar una confirmación en particular, use la ID de COMMIT adecuada en lugar del puntero HEAD. Tom usa el siguiente comando para insertar la etiqueta en el repositorio remoto.

```
[tom@CentOS project]$ git push origin tag Release_1_0
```

El comando anterior producirá el siguiente resultado:

```
Counting objects: 1, done.
Writing objects: 100% (1/1), 183 bytes, done.
Total 1 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new tag]
Release_1_0 -> Release_1_0
```

Ver etiquetas

Tom creó etiquetas. Ahora, Jerry puede ver todas las etiquetas disponibles usando el comando de etiqueta de Git con la opción –l.

```
[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git pull
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (1/1), done.
From git.server.com:project
* [new tag]
Release_1_0 -> Release_1_0
Current branch master is up to date.

[jerry@CentOS src]$ git tag -l
Release_1_0
```

Jerry usa el comando show de Git seguido de su nombre de etiqueta para ver más detalles sobre la etiqueta.

```
[jerry@CentOS src]$ git show Release_1_0
```

El comando anterior producirá el siguiente resultado:

```
tag Release_1_0
Tagger: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 13:45:54 2013 +0530
```

```
Tagged basic string operation code


commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary

diff --git a/src/string_operations b/src/string_operations
deleted file mode 100755
index 654004b..0000000
Binary files a/src/string_operations and /dev/null differ
```

Eliminar etiquetas

Tom usa el siguiente comando para eliminar etiquetas tanto del repositorio local como del remoto.

```
[tom@CentOS project]$ git tag
Release_1_0

[tom@CentOS project]$ git tag -d Release_1_0
Deleted tag 'Release_1_0' (was 0f81ff4)
# Remove tag from remote repository.

[tom@CentOS project]$ git push origin :Release_1_0
To gituser@git.server.com:project.git
- [deleted]
Release_1_0
```

Git - Operación de parches

Patch es un archivo de texto, cuyo contenido es similar a Git diff, pero junto con el código, también tiene metadatos sobre confirmaciones; por ejemplo, ID de confirmación, fecha, mensaje de confirmación, etc. Podemos crear un parche a partir de confirmaciones y otras personas pueden aplicarlas a su repositorio.

Jerry implementa la función strcat para su proyecto. Jerry puede crear una ruta de su código y enviársela a Tom. Luego, puede aplicar el parche recibido a su código.

Jerry usa el comando **format-patch de** Git para crear un parche para la última confirmación. Si desea crear un parche para una confirmación específica, use **COMMIT_ID** con el comando format-patch.

```
[jerry@CentOS project]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git status -s
M string_operations.c
?? string_operations
```

```
[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m "Added my_strcat function"

[master b4c7f09] Added my_strcat function
1 files changed, 13 insertions(+), 0 deletions(-)

[jerry@CentOS src]$ git format-patch -1
0001-Added-my_strcat-function.patch
```

El comando anterior crea archivos **.patch** dentro del directorio de trabajo actual. Tom puede usar este parche para modificar sus archivos. Git proporciona dos comandos para aplicar parches , **git am** y **git apply** , respectivamente. **Git apply** modifica los archivos locales sin crear una confirmación, mientras que **git am** modifica el archivo y también crea una confirmación.

Para aplicar un parche y crear una confirmación, use el siguiente comando:

```
[tom@CentOS src]$ pwd
/home/tom/top_repo/project/src

[tom@CentOS src]$ git diff

[tom@CentOS src]$ git status –s

[tom@CentOS src]$ git apply 0001-Added-my_strcat-
function.patch

[tom@CentOS src]$ git status -s
M string_operations.c
?? 0001-Added-my_strcat-function.patch
```

El parche se aplicó con éxito, ahora podemos ver las modificaciones usando el comando **git diff .**

```
[tom@CentOS src]$ git diff
```

El comando anterior producirá el siguiente resultado:

```
diff --git a/src/string_operations.c
b/src/string_operations.c
index 8ab7f42..f282fcf 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,5 +1,16 @@
#include <stdio.h>
+char *my_strcat(char *t, char *s)
diff --git a/src/string_operations.c
b/src/string_operations.c
index 8ab7f42..f282fcf 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,5 +1,16 @@
```

```
#include <stdio.h>
+char *my_strcat(char *t, char *s)
+
{
   +
   char *p = t;
   +
   +
   +
   while (*p)
   ++p;
   +
   while (*p++ = *s++)
   + ;
   + return t;
   +
}
+
size_t my_strlen(const char *s)
{
   const char *p = s;
   @@ -23,6 +34,7 @@ int main(void)
   {
```

Git - Gestión de sucursales

La operación de sucursales permite crear otra línea de desarrollo. Podemos usar esta operación para bifurcar el proceso de desarrollo en dos direcciones diferentes. Por ejemplo, lanzamos un producto para la versión 6.0 y es posible que deseemos crear una rama para que el desarrollo de las características de 7.0 se pueda mantener separado de las correcciones de errores de 6.0.

Crear Branch

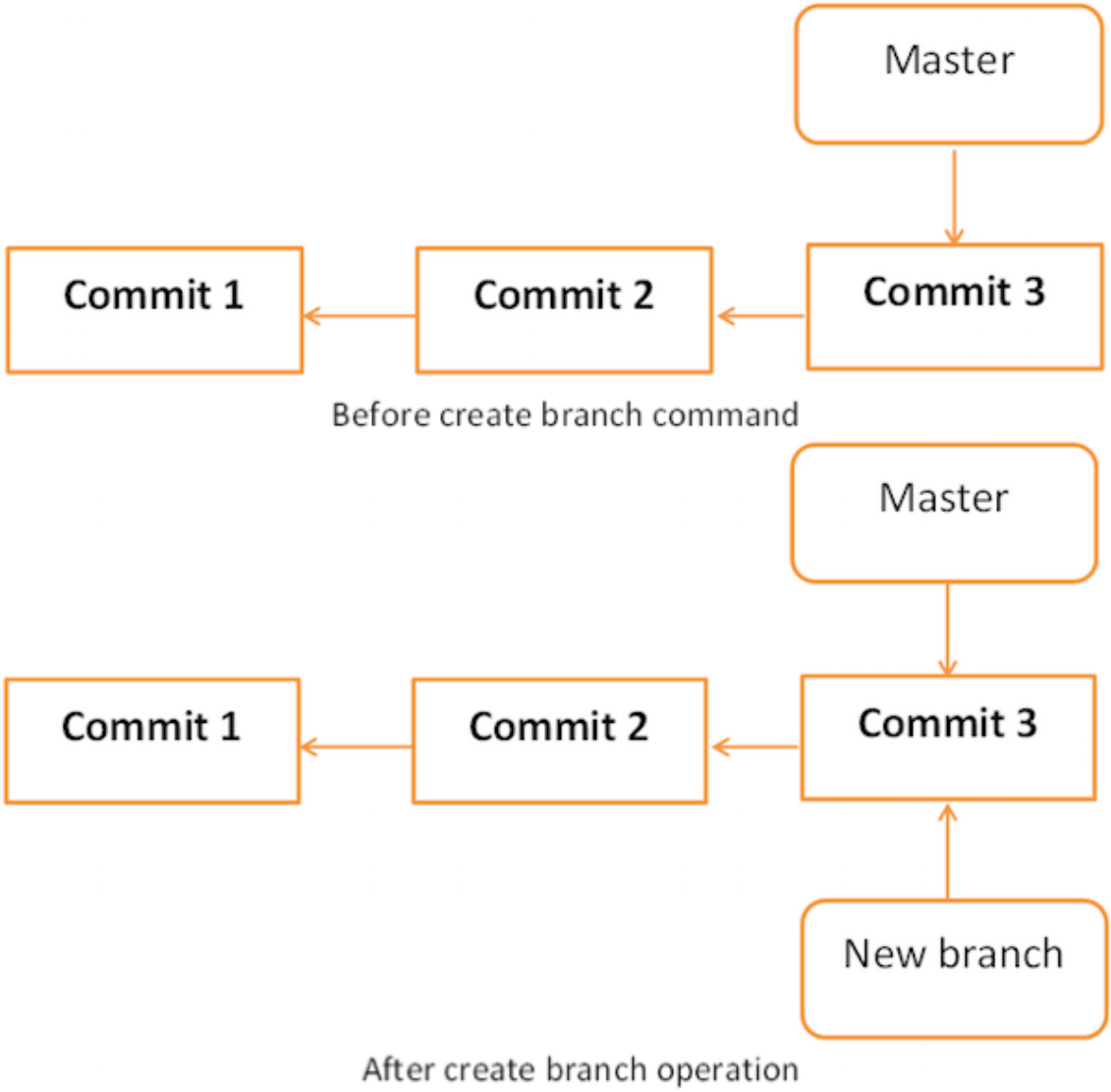
Tom crea una nueva rama usando el comando git branch <branch name>. Podemos crear una nueva sucursal a partir de una existente. Podemos usar una confirmación o etiqueta específica como punto de partida. Si no se proporciona ningún ID de confirmación específico, la rama se creará con HEAD como punto de partida.

```
[jerry@CentOS src]$ git branch new_branch

[jerry@CentOS src]$ git branch
* master
new_branch
```

Se crea una nueva rama; Tom usó el comando git branch para listar las ramas disponibles. Git muestra un asterisco antes de la rama actualmente desprotegida.

La representación pictórica de la operación de creación de rama se muestra a continuación:



Cambiar entre Branches

Jerry usa el comando git checkout para cambiar entre sucursales.

```
[jerry@CentOS src]$ git checkout new_branch
Switched to branch 'new_branch'
[jerry@CentOS src]$ git branch
master
* new_branch
```

Acceso directo para crear y cambiar de Branch

En el ejemplo anterior, hemos usado dos comandos para crear y cambiar ramas, respectivamente. Git proporciona la opción **–b** con el comando de pago; esta operación crea una nueva rama e inmediatamente cambia a la nueva rama.

```
[jerry@CentOS src]$ git checkout -b test_branch
Switched to a new branch 'test_branch'

[jerry@CentOS src]$ git branch
master
```

```
new_branch
* test_branch
```

Eliminar una Branch

Se puede eliminar una rama proporcionando la opción –D con el comando git branch. Pero antes de eliminar la rama existente, cambie a la otra rama.

Jerry está actualmente en **test_branch** y quiere eliminar esa rama. Así que cambia de rama y borra la rama como se muestra a continuación.

```
[jerry@CentOS src]$ git branch
master
new_branch
* test_branch

[jerry@CentOS src]$ git checkout master
Switched to branch 'master'

[jerry@CentOS src]$ git branch -D test_branch
Deleted branch test_branch (was 5776472).
```

Ahora, Git mostrará solo dos ramas.

```
[jerry@CentOS src]$ git branch
* master
new_branch
```

Cambiar el nombre de una Branch

Jerry decide agregar soporte para caracteres anchos en su proyecto de operaciones de cadenas. Ya ha creado una nueva sucursal, pero el nombre de la sucursal no es apropiado. Así que cambia el nombre de la sucursal usando la opción **–m** seguida del nombre de la **sucursal anterior y el nombre** de la **nueva sucursal** .

```
[jerry@CentOS src]$ git branch
* master
new_branch

[jerry@CentOS src]$ git branch -m new_branch wchar_support
```

Ahora, el comando git branch mostrará el nuevo nombre de la rama.

```
[jerry@CentOS src]$ git branch
* master
wchar_support
```

Combinar dos Branches

Jerry implementa una función para devolver la longitud de cadena de cadena de caracteres anchos. Nuevo, el código aparecerá de la siguiente manera:

```
[jerry@CentOS src]$ git branch
master
* wchar_support
```

```
[jerry@CentOS src]$ pwd
/home/jerry/jerry_repo/project/src

[jerry@CentOS src]$ git diff
```

El comando anterior produce el siguiente resultado:

```
t a/src/string_operations.c b/src/string_operations.c
index 8ab7f42..8fb4b00 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,4 +1,14 @@
#include <stdio.h>
+#include <wchar.h>
+
+size_t w_strlen(const wchar_t *s)
+
{
   +
   const wchar_t *p = s;
   +
   +
   while (*p)
   + ++p;
   + return (p - s);
   +
}
```

Después de la prueba, confirma y envía sus cambios a la nueva rama.

```
[jerry@CentOS src]$ git status -s
M string_operations.c
?? string_operations

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m 'Added w_strlen function to
return string lenght of wchar_t
string'

[wchar_support 64192f9] Added w_strlen function to return
string lenght of wchar_t string
1 files changed, 10 insertions(+), 0 deletions(-)
```

Tenga en cuenta que Jerry está enviando estos cambios a la nueva rama, razón por la cual usó el nombre de la rama **wchar_support** en lugar de la rama **principal** .

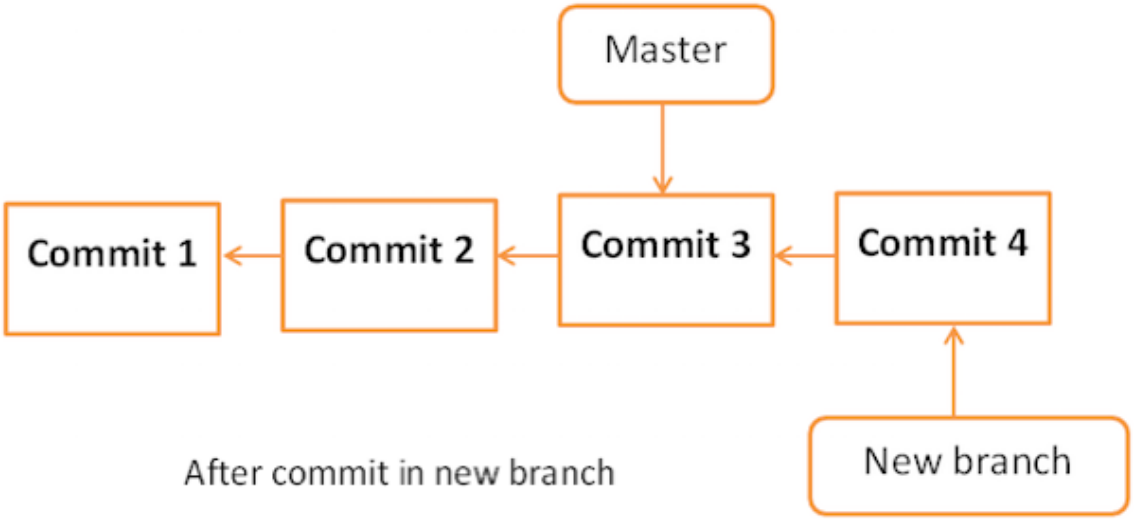
```
[jerry@CentOS src]$ git push origin wchar_support    <---
Observer branch_name
```

El comando anterior producirá el siguiente resultado.

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 507 bytes, done.
```

```
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new branch]
wchar_support -> wchar_support
```

Después de confirmar los cambios, la nueva rama aparecerá de la siguiente manera:



Tom siente curiosidad por saber qué está haciendo Jerry en su rama privada y consulta el registro de la rama **wchar_support** .

```
[tom@CentOS src]$ pwd
/home/tom/top_repo/project/src

[tom@CentOS src]$ git log origin/wchar_support -2
```

El comando anterior producirá el siguiente resultado.

```
commit 64192f91d7cc2bcdf3bf946dd33ece63b74184a3
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 16:10:06 2013 +0530

Added w_strlen function to return string lenght of wchar_t
string


commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary
```

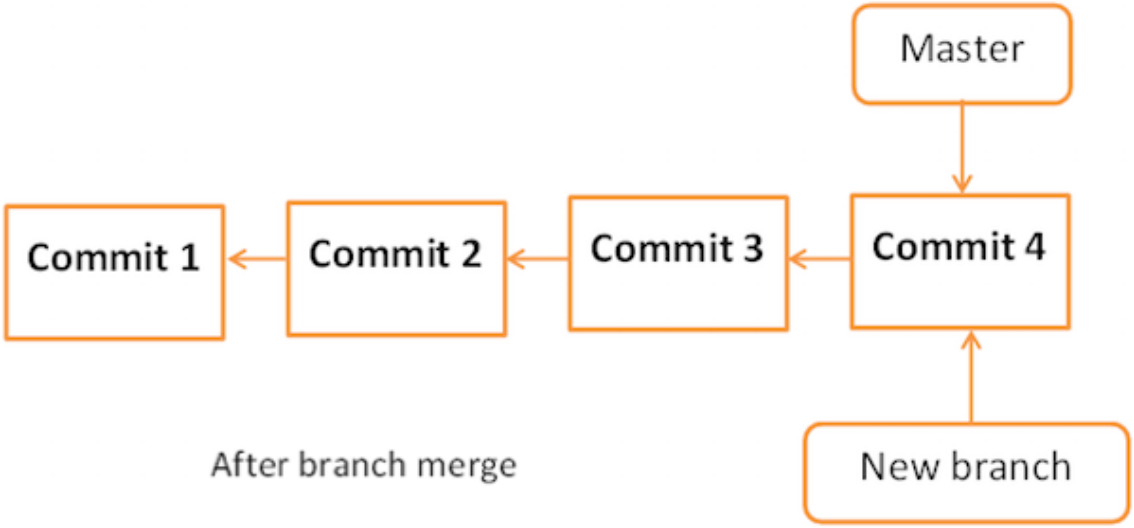
Al ver los mensajes de confirmación, Tom se da cuenta de que Jerry implementó la función strlen para caracteres anchos y quiere la misma funcionalidad en la rama principal. En lugar de volver a implementar, decide tomar el código de Jerry fusionando su rama con la rama principal.

```
[tom@CentOS project]$ git branch
* master
```

```
[tom@CentOS project]$ pwd
/home/tom/top_repo/project

[tom@CentOS project]$ git merge origin/wchar_support
Updating 5776472..64192f9
Fast-forward
src/string_operations.c | 10 ++++++++++
1 files changed, 10 insertions(+), 0 deletions(-)
```

Después de la operación de fusión, la rama maestra aparecerá de la siguiente manera:



Ahora, la rama **wchar_support** se fusionó con la rama principal. Podemos verificarlo viendo el mensaje de confirmación o viendo las modificaciones realizadas en el archivo string_operation.c.

```
[tom@CentOS project]$ cd src/

[tom@CentOS src]$ git log -1

commit 64192f91d7cc2bcdf3bf946dd33ece63b74184a3
Author: Jerry Mouse
Date: Wed Sep 11 16:10:06 2013 +0530

Added w_strlen function to return string lenght of wchar_t
string

[tom@CentOS src]$ head -12 string_operations.c
```

El comando anterior producirá el siguiente resultado.

```
#include <stdio.h>
#include <wchar.h>
size_t w_strlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

```
   while (*p)
      ++p;

   return (p - s);
}
```

Después de la prueba, envía sus cambios de código a la rama maestra.

```
[tom@CentOS src]$ git push origin master
Total 0 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
5776472..64192f9 master -> master
```

Reorganizar Branches

El comando Git rebase es un comando de combinación de ramas, pero la diferencia es que modifica el orden de las confirmaciones.

El comando merge de Git intenta poner las confirmaciones de otras ramas encima del HEAD de la rama local actual. Por ejemplo, su rama local tiene confirmaciones A->B->C->D y la rama de combinación tiene confirmaciones A->B->X->Y, luego git merge convertirá la rama local actual en algo como A->B–>C–>D–>X–>Y

El comando Git rebase intenta encontrar el ancestro común entre la rama local actual y la rama fusionada. Luego envía las confirmaciones a la rama local modificando el orden de las confirmaciones en la rama local actual. Por ejemplo, si su rama local tiene confirmaciones A->B->C->D y la rama de combinación tiene confirmaciones A->B->X->Y, entonces Git rebase convertirá la rama local actual en algo como A- >B->X->Y->C->D.

Cuando varios desarrolladores trabajan en un solo repositorio remoto, no puede modificar el orden de las confirmaciones en el repositorio remoto. En esta situación, puede usar la operación de reorganización para poner sus confirmaciones locales encima de las confirmaciones del repositorio remoto y puede impulsar estos cambios.

Git - Manejo de conflictos

Realizar cambios en la rama wchar_support

Jerry está trabajando en la rama **wchar_support** . Cambia el nombre de las funciones y después de probar, confirma sus cambios.

```
[jerry@CentOS src]$ git branch
 master
* wchar_support
[jerry@CentOS src]$ git diff
```

El comando anterior produce el siguiente resultado:

```
diff --git a/src/string_operations.c
b/src/string_operations.c
index 8fb4b00..01ff4e0 100644
```

```
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,7 +1,7 @@
#include <stdio.h>
#include <wchar.h>
-size_t w_strlen(const wchar_t *s)
+size_t my_wstrlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

Después de verificar el código, confirma sus cambios.

```
[jerry@CentOS src]$ git status -s
M string_operations.c

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m 'Changed function name'
[wchar_support 3789fe8] Changed function name
1 files changed, 1 insertions(+), 1 deletions(-)

[jerry@CentOS src]$ git push origin wchar_support
```

El comando anterior producirá el siguiente resultado:

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 409 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
64192f9..3789fe8 wchar_support -> wchar_support
```

Realizar cambios en la rama principal

Mientras tanto, en la rama maestra, Tom también cambia el nombre de la misma función y envía sus cambios a la rama maestra.

```
[tom@CentOS src]$ git branch
* master
[tom@CentOS src]$ git diff
```

El comando anterior produce el siguiente resultado:

```
diff --git a/src/string_operations.c
b/src/string_operations.c
index 8fb4b00..52bec84 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,7 +1,8 @@
#include <stdio.h>
#include <wchar.h>
-size_t w_strlen(const wchar_t *s)
+/* wide character strlen fucntion */
+size_t my_wc_strlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

Después de verificar diff, confirma sus cambios.

```
[tom@CentOS src]$ git status -s
M string_operations.c

[tom@CentOS src]$ git add string_operations.c

[tom@CentOS src]$ git commit -m 'Changed function name from
w_strlen to my_wc_strlen'
[master ad4b530] Changed function name from w_strlen to
my_wc_strlen
1 files changed, 2 insertions(+), 1 deletions(-)

[tom@CentOS src]$ git push origin master
```

El comando anterior producirá el siguiente resultado:

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 470 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
64192f9..ad4b530 master -> master
```

En la rama **wchar_support** , Jerry implementa la función strchr para cadenas de caracteres anchas. Después de la prueba, confirma y envía sus cambios a la rama **wchar_support** .

```
[jerry@CentOS src]$ git branch
master
* wchar_support
[jerry@CentOS src]$ git diff
```

El comando anterior produce el siguiente resultado:

```
diff --git a/src/string_operations.c
b/src/string_operations.c
index 01ff4e0..163a779 100644
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@ -1,6 +1,16 @@
#include <stdio.h>
#include <wchar.h>
+wchar_t *my_wstrchr(wchar_t *ws, wchar_t wc)
+
{
   +
   while (*ws)
   {
      +
      if (*ws == wc)
      +
      return ws;
      +
      ++ws;
```

```
      +
   }
   + return NULL;
   +
}
+
size_t my_wstrlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

Después de verificar, confirma sus cambios.

```
[jerry@CentOS src]$ git status -s
M string_operations.c

[jerry@CentOS src]$ git add string_operations.c

[jerry@CentOS src]$ git commit -m 'Addded strchr function for
wide character string'
[wchar_support 9d201a9] Addded strchr function for wide
character string
1 files changed, 10 insertions(+), 0 deletions(-)

[jerry@CentOS src]$ git push origin wchar_support
```

El comando anterior producirá el siguiente resultado:

```
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 516 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To gituser@git.server.com:project.git
3789fe8..9d201a9 wchar_support -> wchar_support
```

Abordar conflictos

Tom quiere ver qué está haciendo Jerry en su rama privada, por lo que intenta extraer los últimos cambios de la rama **wchar_support** , pero Git cancela la operación con el siguiente mensaje de error.

```
[tom@CentOS src]$ git pull origin wchar_support
```

El comando anterior produce el siguiente resultado:

```
remote: Counting objects: 11, done.
63Git Tutorials
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (8/8), done.
From git.server.com:project
* branch
wchar_support -> FETCH_HEAD
Auto-merging src/string_operations.c
CONFLICT (content): Merge conflict in src/string_operations.c
Automatic merge failed; fix conflicts and then commit the
result.
```

Resolver conflictos

Del mensaje de error, está claro que hay un conflicto en src/string_operations.c. Ejecuta el comando git diff para ver más detalles.

```
[tom@CentOS src]$ git diff
```

El comando anterior produce el siguiente resultado:

```
diff --cc src/string_operations.c
index 52bec84,163a779..0000000
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@@ -1,8 -1,17 +1,22 @@@
#include <stdio.h>
#include <wchar.h>
++<<<<<<< HEAD
+/* wide character strlen fucntion */
+size_t my_wc_strlen(const wchar_t *s)
++=======
+ wchar_t *my_wstrchr(wchar_t *ws, wchar_t wc)
+
{
   +
   +
   while (*ws)
   {
      if (*ws == wc)
      +
      return ws;
      +
      ++ws;
      +
   }
   + return NULL;
   +
}
+
+ size_t my_wstrlen(const wchar_t *s)
++>>>>>>>9d201a9c61bc4713f4095175f8954b642dae8f86
{
   const wchar_t *p = s;
```

Como tanto Tom como Jerry cambiaron el nombre de la misma función, Git se encuentra en un estado de confusión y le pide al usuario que resuelva el conflicto manualmente.

Tom decide mantener el nombre de la función sugerido por Jerry, pero mantiene el comentario agregado por él, tal como está. Después de eliminar los marcadores de conflicto, git diff se verá así.

```
[tom@CentOS src]$ git diff
```

El comando anterior produce el siguiente resultado.

```
diff --cc src/string_operations.c
diff --cc src/string_operations.c
index 52bec84,163a779..0000000
--- a/src/string_operations.c
+++ b/src/string_operations.c
@@@ -1,8 -1,17 +1,18 @@@
#include <stdio.h>
#include <wchar.h>
+ wchar_t *my_wstrchr(wchar_t *ws, wchar_t wc)
+
{
   +
   while (*ws)
   {
      +
      if (*ws == wc)
      +
      return ws;
      +
      ++ws;
      +
   }
   + return NULL;
   +
}
+
+/* wide character strlen fucntion */
- size_t my_wc_strlen(const wchar_t *s)
+ size_t my_wstrlen(const wchar_t *s)
{
   const wchar_t *p = s;
```

Como Tom ha modificado los archivos, primero debe confirmar estos cambios y, a partir de entonces, puede extraer los cambios.

```
[tom@CentOS src]$ git commit -a -m 'Resolved conflict'
[master 6b1ac36] Resolved conflict

[tom@CentOS src]$ git pull origin wchar_support.
```

Tom ha resuelto el conflicto, ahora la operación de extracción tendrá éxito.

Git - Diferentes plataformas

GNU/Linux y Mac OS usan salto **de línea (LF)** o nueva línea como carácter de final de línea, mientras que Windows usa la combinación de **salto de línea y retorno de carro (LFCR)** para representar el carácter de final de línea.

Para evitar confirmaciones innecesarias debido a estas diferencias en los finales de línea, debemos configurar el cliente de Git para que escriba el mismo final de línea en el repositorio de Git.

Para el sistema Windows, podemos configurar el cliente Git para convertir los finales de línea al formato **CRLF** durante la verificación y volver a convertirlos al

formato **LF** durante la operación de confirmación. Los siguientes ajustes harán lo necesario.

```
[tom@CentOS project]$ git config --global core.autocrlf true
```

Para GNU/Linux o Mac OS, podemos configurar el cliente Git para convertir finales de línea de **CRLF** a **LF** mientras realiza la operación de pago.

```
[tom@CentOS project]$ git config --global core.autocrlf input
```

Git - Repositorios en línea

GitHub es un servicio de alojamiento web para proyectos de desarrollo de software que utiliza el sistema de control de revisión Git. También tiene su aplicación GUI estándar disponible para descargar (Windows, Mac, GNU/Linux) directamente desde el sitio web del servicio. Pero en esta sesión, solo veremos la parte CLI.

Crear repositorio de GitHub

Vaya a github.com . Si ya tiene la cuenta de **GitHub** , inicie sesión con esa cuenta o cree una nueva. Siga los pasos del sitio web github.com para crear un nuevo repositorio.

Operación de empuje

Tom decide usar el servidor de **GitHub .** Para iniciar un nuevo proyecto, crea un nuevo directorio y un archivo dentro de él.

```
[tom@CentOS]$ mkdir github_repo

[tom@CentOS]$ cd github_repo/

[tom@CentOS]$ vi hello.c

[tom@CentOS]$ make hello
cc hello.c -o hello

[tom@CentOS]$ ./hello
```

El comando anterior producirá el siguiente resultado:

```
Hello, World !!!
```

Después de verificar su código, inicializa el directorio con el comando git init y confirma sus cambios localmente.

```
[tom@CentOS]$ git init
Initialized empty Git repository in
/home/tom/github_repo/.git/

[tom@CentOS]$ git status -s
?? hello
?? hello.c
```

```
[tom@CentOS]$ git add hello.c

[tom@CentOS]$ git status -s
A hello.c
?? hello

[tom@CentOS]$ git commit -m 'Initial commit'
```

Después de eso, agrega la URL del repositorio de **GitHub** como un origen remoto y envía sus cambios al repositorio remoto.

```
[tom@CentOS]$ git remote add origin
https://github.com/kangralkar/testing_repo.git

[tom@CentOS]$ git push -u origin master
```

La operación Push le pedirá el nombre de usuario y la contraseña de **GitHub .** Después de una autenticación exitosa, la operación tendrá éxito.

El comando anterior producirá el siguiente resultado:

```
Username for 'https://github.com': kangralkar
Password for 'https://kangralkar@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 214 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/kangralkar/test_repo.git
 * [new branch]      master -> master
 Branch master set up to track remote branch master from
origin.
```

A partir de ahora, Tom puede enviar cualquier cambio al repositorio de **GitHub .** Puede usar todos los comandos discutidos en este capítulo con el repositorio de **GitHub .**

Operación de tracción

Tom envió con éxito todos sus cambios al repositorio de **GitHub .** Ahora, otros desarrolladores pueden ver estos cambios realizando una operación de clonación o actualizando su repositorio local.

Jerry crea un nuevo directorio en su directorio de inicio y clona el repositorio de **GitHub** usando el comando git clone.

```
[jerry@CentOS]$ pwd
/home/jerry

[jerry@CentOS]$ mkdir jerry_repo

[jerry@CentOS]$ git clone
https://github.com/kangralkar/test_repo.git
```

El comando anterior produce el siguiente resultado:

```
Cloning into 'test_repo'...
remote: Counting objects: 3, done.
```

```
remote: Total 3 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (3/3), done.
```

Verifica el contenido del directorio ejecutando el comando ls.

```
[jerry@CentOS]$ ls
test_repo

[jerry@CentOS]$ ls test_repo/
hello.c
```