GIT LAB Guide

**www.zippyops.com**

172-172, 5th floor Old Mahabalipuram Road

(Above Axis Bank-PTC Bus Stop)  
Thuraipakkam  
Chennai 600097

**🖂**[**zippyops@gmail.com**](mailto:zippyops@gmail.com)

**✆+91 7010585768**

Contents

[Prerequisites 1](#_Toc500078088)

[Installation 1](#_Toc500078089)

[Install Git with Yum 1](#_Toc500078090)

[Set Up Git 2](#_Toc500078091)

[Create a Bare Repository 2](#_Toc500078092)

[Setup a repo in github 2](#_Toc500078093)

[Generate Public/Private RSA Key Pair 4](#_Toc500078094)

[Adding Keys to GITHUB 5](#_Toc500078095)

[Push Changes to the Repository 6](#_Toc500078096)

[Git - Clone Operation 7](#_Toc500078097)

[Git - Perform Changes 8](#_Toc500078098)

[Git - Review Changes 9](#_Toc500078099)

[Git - Commit Changes 11](#_Toc500078100)

[Git - Push Operation 12](#_Toc500078101)

[Git - Update Operation 13](#_Toc500078102)

[Add New Function 15](#_Toc500078103)

[Fetch Latest Changes 17](#_Toc500078104)

[Git - Stash Operation 18](#_Toc500078105)

[Git - Move Operation 19](#_Toc500078106)

[Git - Rename Operation 20](#_Toc500078107)

[Git - Delete Operation 21](#_Toc500078108)

[Git - Fix Mistakes 22](#_Toc500078109)

[Revert Uncommitted Changes 22](#_Toc500078110)

[Remove Changes from Staging Area 23](#_Toc500078111)

[Move HEAD Pointer with Git Reset 24](#_Toc500078112)

[Soft 25](#_Toc500078113)

[mixed 26](#_Toc500078114)

[hard 26](#_Toc500078115)

[Conclusion 27](#_Toc500078116)

[Git - Tag Operation 28](#_Toc500078117)

[Create Tags 28](#_Toc500078118)

[View Tags 28](#_Toc500078119)

[Delete Tags 29](#_Toc500078120)

[Git - Patch Operation 29](#_Toc500078121)

[Git - Managing Branches 31](#_Toc500078122)

[Create Branch 32](#_Toc500078123)

[Switch between Branches 33](#_Toc500078124)

[Shortcut to Create and Switch Branch 33](#_Toc500078125)

[Delete a Branch 33](#_Toc500078126)

[Rename a Branch 34](#_Toc500078127)

[Merge Two Branches 34](#_Toc500078128)

[Rebase Branches 37](#_Toc500078129)

[Perform Changes in wchar\_support Branch 37](#_Toc500078130)

[Perform Changes in Master Branch 38](#_Toc500078131)

[Tackle Conflicts 40](#_Toc500078132)

[Resolve Conflicts 40](#_Toc500078133)

# Prerequisites

Before you begin with this guide, there are a few steps that need to be completed first.

* You will need a CentOS 7 with latest update.
* Create a non-root user that has sudo privileges
* Create 2 normal users as user-a and user-b
* Create a github account at https://github.com/

# Installation

The two most common ways to install Git will be described in this section. Each option has their own advantages and disadvantages, and the choice you make will depend on your own needs. For example, users who want to maintain updates to the Git software will likely want to use yum to install Git, while users who need features presented by a specific version of Git will want to build that version from source. We are going to use YUM installation

## Install Git with Yum

The easiest way to install Git and have it ready to use is to use CentOS's default repositories. This is the fastest method, but the Git version that is installed this way may be older than the newest version available

Use yum, CentOS's native package manager, to search for and install the latest Git package available in CentOS's repositories:

sudo yum install git

If the command completes without error, you will have git downloaded and installed. To double-check that it is working correctly, try running Git's built-in version check:

git --version

If that check produced a Git version number, then you can now move on

# Set Up Git

Now that you have git installed, you will need to submit some information about yourself so that commit messages will be generated with the correct information attached. To do this, use the git config command to provide the name and email address that you would like to have embedded into your commits

Login as user-a and perform below steps

git config --global user.name "Your Name"

git config --global user.email "you@example.com"

To confirm that these configurations were added successfully, we can see all of the configuration items that have been set by typing:

git config --list

This configuration will save you the trouble of seeing an error message and having to revise commits after you submit them.

Login as user-b and repeat the above steps

# Create a Bare Repository

[user-a@gitlab ~]$ pwd

/home/user-a

[user-a@gitlab ~]$ mkdir project.git

[user-a@gitlab ~]$ cd project.git/

[user-a@gitlab project.git]$ ls -l

total 0

[user-a@gitlab project.git]$ git --bare init

Initialized empty Git repository in /home/user-a/project.git/

[user-a@gitlab project.git]$ ls

branches config description HEAD hooks info objects refs

[user-a@gitlab project.git]$

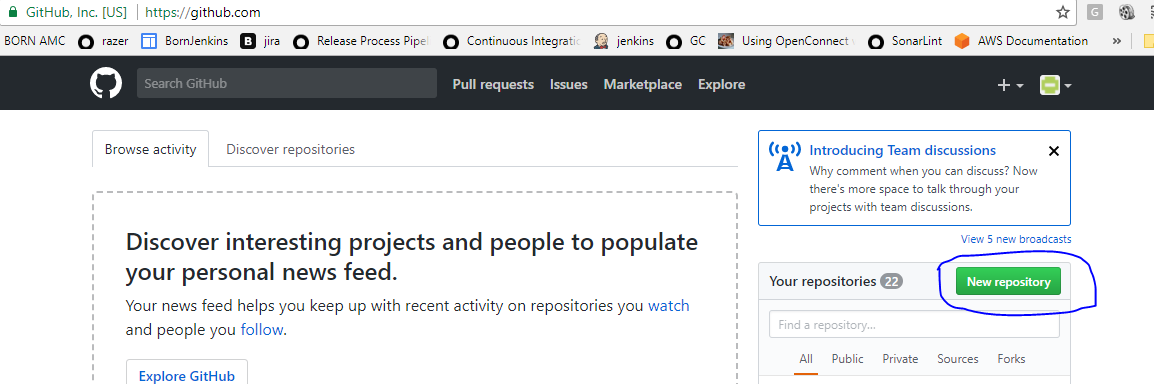
The above steps can be used to create a repository in local machine and this will act as git server.

But in our lab we are going to use the repository created in GitHub.

# Setup a repo in github

Login to github.com

Click on New repository

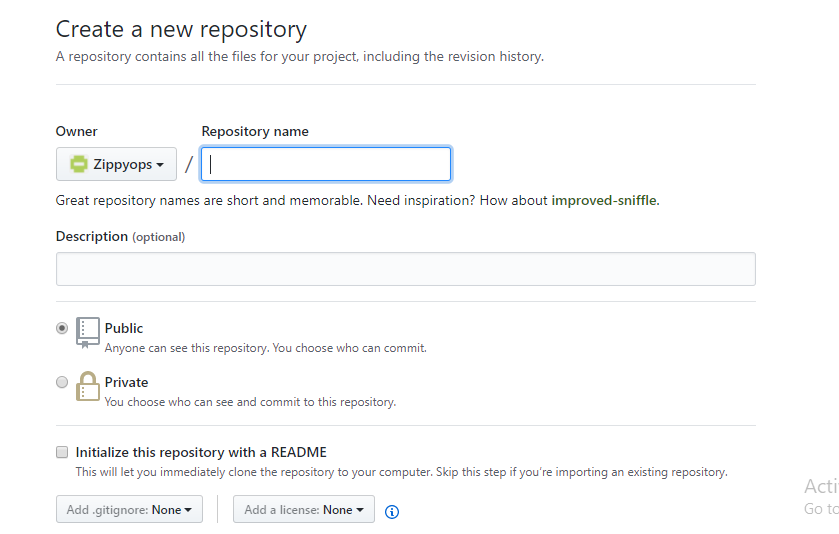


Give a unique name for repository

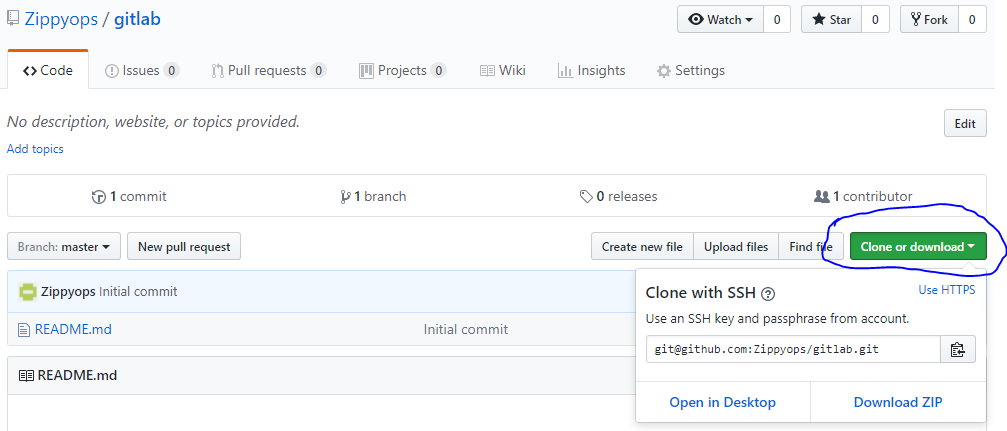
Provide a Description for the repository

Select “**Public”**

Click on **“Create repository”**



Click on **Clone or download** to get the git clone url



# Generate Public/Private RSA Key Pair

You need to create public/private RSA key pair for authentication

Login to your server as user-a

ssh-keygen utility generates public/private RSA key pair

[user-a@CentOS ~]$ ssh-keygen

After successful completion, it will create a .ssh directory inside the home directory

Generating public/private rsa key pair.

Enter file in which to save the key (/home/user-a/.ssh/id\_rsa): **Press Enter Only**

Created directory '/home/user-a/.ssh'.

Enter passphrase (empty for no passphrase): **---------------> Press Enter Only**

Enter same passphrase again: **------------------------------> Press Enter Only**

Your identification has been saved in /home/user-a/.ssh/id\_rsa.

Your public key has been saved in /home/user-a/.ssh/id\_rsa.pub.

The key fingerprint is:

df:93:8c:a1:b8:b7:67:69:3a:1f:65:e8:0e:e9:25:a1 user-a@CentOS

The key's randomart image is:

+--[ RSA 2048]----+

| |

| |

| |

|

.

|

| Soo |

| o\*B. |

| E = \*.= |

| oo==. . |

| ..+Oo

|

+-----------------+

ssh-keygen has generated two keys, first one is private (i.e., id\_rsa) and the second one is public (i.e., id\_rsa.pub).

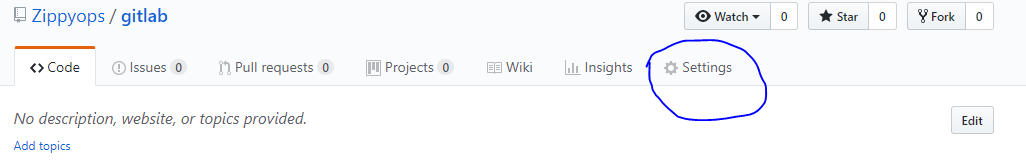
Note: Never share your PRIVATE KEY with others.

Repeat the above step for user-b

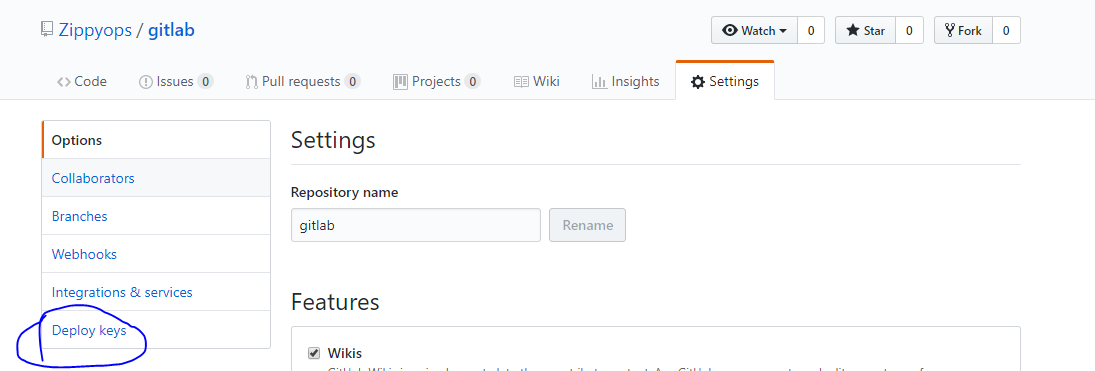
# Adding Keys to GITHUB

Login to Github and select the repository created earlier

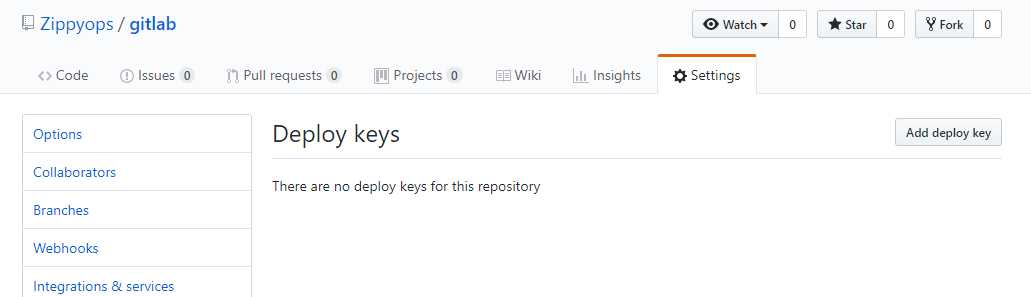
Click on **settings**



Then click on **Deploy keys**



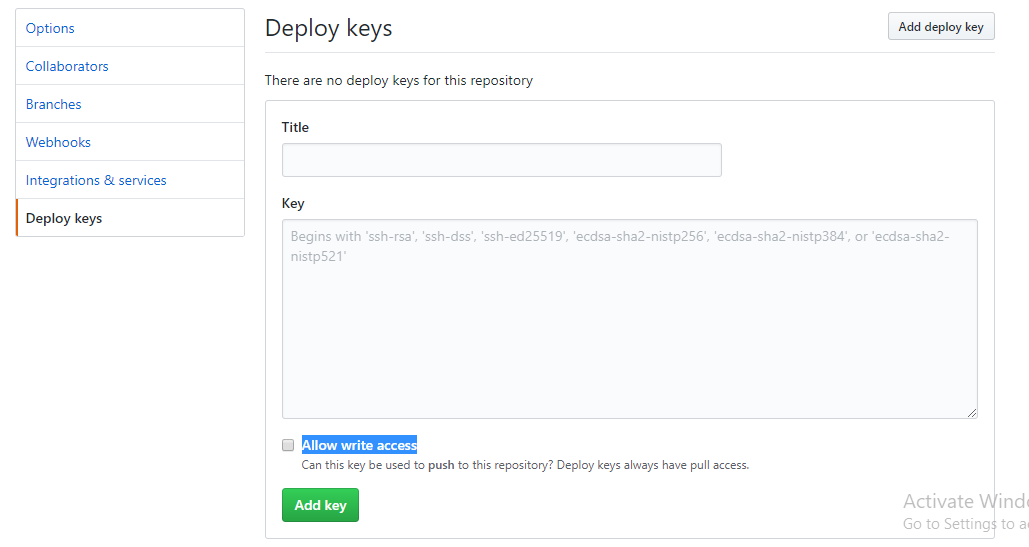
Click on **Add deploy key**



Give title as user-a

Copy the content of id\_rsa.pub (created in previous step) for user-a and paste it under **Key**

Select **Allow write access** andClick on **Add key**



Repeat the above steps for user-b

# Push Changes to the Repository

We have created a bare repository on the server and allowed access for two users. From now on, user-a and user-b can push their changes to the repository by adding it as a remote.

Git init command creates .git directory to store metadata about the repository every time it reads the configuration from the .git/config file.

user-a creates a new directory, adds test.txt file, and commits his change as initial commit. After commit, he verifies the commit message by running the git log command.

[user-a@gitlab ~]$ pwd

/home/user-a

[user-a@gitlab ~]$ mkdir user-a\_repo

[user-a@gitlab ~]$ cd user-a\_repo/

[user-a@gitlab user-a\_repo]$ git init

Initialized empty Git repository in /home/user-a/user-a\_repo/.git/

[user-a@gitlab user-a\_repo]$ echo 'TODO: Add contents for testfile' > test.txt

[user-a@gitlab user-a\_repo]$ git status -s

?? test.txt

[user-a@gitlab user-a\_repo]$ git add .

[user-a@gitlab user-a\_repo]$ git status -s

A test.txt

[user-a@gitlab user-a\_repo]$ git commit -m 'Initial commit'

The above command will produce the following result.

[master (root-commit) 35ff72a] Initial commit

1 file changed, 1 insertion(+)

create mode 100644 test.txt

user-a checks the log message by executing the git log command.

[user-a@gitlab user-a\_repo]$ git log

commit 35ff72afbbc575e6586789c6737121aa9f5998ee

Author: usera <usera@zippyops.com>

Date: Fri Dec 1 01:52:15 2017 -0500

Initial commit

User-a committed his changes to the local repository. Now, it’s time to push the changes to the remote repository. But before that, we have to add the repository as a remote, this is a one-time operation. After this, he can safely push the changes to the remote repository. (The clone URL which we have earlier)

**Note:** By default, Git pushes only to matching branches: For every branch that exists on the local side, the remote side is updated if a branch with the same name already exists there. In our tutorials, every time we push changes to the **origin master** branch, use appropriate branch name according to your requirement.

[user-a@gitlab user-a\_repo]$ git remote add origin git@github.com:Zippyops/gitlab.git

[user-a@gitlab user-a\_repo]$ git push origin master

The above command will produce the following result.

[user-a@gitlab user-a\_repo]$ git push origin master

Counting objects: 3, done.

Writing objects: 100% (3/3), 241 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To git@github.com:Zippyops/gitlab.git

\* [new branch] master -> master

Now, the changes are successfully committed to the remote repository.

# Git - Clone Operation

We have a bare repository on the Git server and user-a also pushed his first version. Now, user-b can view his changes. The Clone operation creates an instance of the remote repository.

user-b creates a new directory in his home directory and performs the clone operation.

[user-b@gitlab ~]$ mkdir user-b\_repo

[user-b@gitlab ~]$ cd user-b\_repo/

[user-b@gitlab user-b\_repo]$ git clone [git@github.com:Zippyops/gitlab.git](mailto:git@github.com:Zippyops/gitlab.git)

The above command will produce the following result.

[user-b@gitlab user-b\_repo]$ git clone git@github.com:Zippyops/gitlab.git

Cloning into 'gitlab'...

Warning: Permanently added the RSA host key for IP address '192.30.253.112' to the list of known hosts.

remote: Counting objects: 3, done.

remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0

Receiving objects: 100% (3/3), done.

User-b changes the directory to new local repository and lists its directory contents.

[user-b@gitlab user-b\_repo]$ cd gitlab/

[user-b@gitlab gitlab]$ ls

test.txt

# Git - Perform Changes

User-b clones the repository and decides to implement basic string operations. So he creates string.c file. After adding the contents, string.c will look like as follows:

vi string.c

#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",

"zippyops"

};

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

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

return 0;

}

[user-b@gitlab gitlab]$ gcc string.c -o string

[user-b@gitlab gitlab]$ ls -l

total 20

-rwxrwxr-x 1 user-b user-b 8552 Dec 1 02:10 string

-rw-rw-r-- 1 user-b user-b 319 Dec 1 02:08 string.c

-rw-rw-r-- 1 user-b user-b 32 Dec 1 02:03 test.txt

He compiled and tested his code and everything is working fine. Now, he can safely add these changes to the repository.

gcc string.c -o string

Git add operation adds file to the staging area.

[user-b@gitlab gitlab]$ git status -s

?? string

?? string.c

[user-b@gitlab gitlab]$ git add string.c

Git is showing a question mark before file names. Obviously, these files are not a part of Git, and that is why Git does not know what to do with these files. That is why, Git is showing a question mark before file names.

User-b has added the file to the stash area, git status command will show files present in the staging area.

[user-b@gitlab gitlab]$ git status -s

A string.c

?? string

To commit the changes, he used the git commit command followed by –m option. If we omit –m option. Git will open a text editor where we can write multiline commit message.

[user-b@gitlab gitlab]$ git commit -m 'Implemented my\_strlen function'

[master 781517b] Implemented my\_strlen function

1 file changed, 27 insertions(+)

create mode 100644 string.c

After commit to view log details, he runs the git log command. It will display the information of all the commits with their commit ID, commit author, commit date and SHA-1 hash of commit.

[user-b@gitlab gitlab]$ git log

commit 781517bf4e9aed9e4ddd33e5785984f2c5258c6f

Author: userb <userb@zippyops.com>

Date: Fri Dec 1 02:14:26 2017 -0500

Implemented my\_strlen function

commit 35ff72afbbc575e6586789c6737121aa9f5998ee

Author: usera <usera@zippyops.com>

Date: Fri Dec 1 01:52:15 2017 -0500

Initial commit

# Git - Review Changes

After viewing the commit details, user-b realizes that the string length cannot be negative, that’s why he decides to change the return type of my\_strlen function.

User-b uses the **git log** command to view log details.

[user-b@gitlab gitlab]$ git log

commit 781517bf4e9aed9e4ddd33e5785984f2c5258c6f

Author: userb <userb@zippyops.com>

Date: Fri Dec 1 02:14:26 2017 -0500

Implemented my\_strlen function

commit 35ff72afbbc575e6586789c6737121aa9f5998ee

Author: usera <usera@zippyops.com>

Date: Fri Dec 1 01:52:15 2017 -0500

Initial commit

User-b uses the git show command to view the commit details. The git show command takes SHA-1 commit ID as a parameter.

[user-b@gitlab gitlab]$ git show 781517bf4e9aed9e4ddd33e5785984f2c5258c6f

commit 781517bf4e9aed9e4ddd33e5785984f2c5258c6f

Author: userb <userb@zippyops.com>

Date: Fri Dec 1 02:14:26 2017 -0500

Implemented my\_strlen function

diff --git a/string.c b/string.c

new file mode 100644

index 0000000..455e1de

--- /dev/null

+++ b/string.c

@@ -0,0 +1,27 @@

+#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",

+ "zippyops"

+ };

+

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

+

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

+

+ return 0;

+}

He changes the return type of the function from int to size\_t. After testing the code, he reviews his changes by running the git diff command.

[user-b@gitlab gitlab]$ git diff

diff --git a/string.c b/string.c

index 455e1de..ce4665e 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;

@@ -21,7 +21,7 @@ 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 = %1u\n", s[i], my\_strlen(s[i]));

return 0;

}

Git diff shows '+' sign before lines, which are newly added and '−' for deleted lines.

# Git - Commit Changes

User-b has already committed the changes and he wants to correct his last commit. In this case, git amend operation will help. The amend operation changes the last commit including your commit message; it creates a new commit ID.

Before amend operation, he checks the commit log.

[user-b@gitlab gitlab]$ git log

commit 781517bf4e9aed9e4ddd33e5785984f2c5258c6f

Author: userb <userb@zippyops.com>

Date: Fri Dec 1 02:14:26 2017 -0500

Implemented my\_strlen function

commit 35ff72afbbc575e6586789c6737121aa9f5998ee

Author: usera <usera@zippyops.com>

Date: Fri Dec 1 01:52:15 2017 -0500

Initial commit

[user-b@gitlab gitlab]$

User-b commits the new changes with -- amend operation and views the commit log.

[user-b@gitlab gitlab]$ git status -s

M string.c

?? string

[user-b@gitlab gitlab]$ git add string.c

[user-b@gitlab gitlab]$ git status -s

M string.c

?? string

[user-b@gitlab gitlab]$ git commit --amend -m 'Changed return type of my\_strlen to size\_t'

[master 92d8b72] Changed return type of my\_strlen to size\_t

1 file changed, 27 insertions(+)

create mode 100644 string.c

Now, git log will show new commit message with new commit ID:

[user-b@gitlab gitlab]$ git log

commit 92d8b72718ff679342c9ed39a07f592fc743d059

Author: userb <userb@zippyops.com>

Date: Fri Dec 1 02:14:26 2017 -0500

Changed return type of my\_strlen to size\_t

commit 35ff72afbbc575e6586789c6737121aa9f5998ee

Author: usera <usera@zippyops.com>

Date: Fri Dec 1 01:52:15 2017 -0500

Initial commit

# Git - Push Operation

User-b modified his last commit by using the amend operation and he is ready to push the changes. The Push operation stores data permanently to the Git repository. After a successful push operation, other developers can see user-b’s changes.

He executes the git log command to view the commit details.

Before push operation, he wants to review his changes, so he uses the git show command to review his changes.

[user-b@gitlab gitlab]$ git show 92d8b72718ff679342c9ed39a07f592fc743d059

commit 92d8b72718ff679342c9ed39a07f592fc743d059

Author: userb <userb@zippyops.com>

Date: Fri Dec 1 02:14:26 2017 -0500

Changed return type of my\_strlen to size\_t

diff --git a/string.c b/string.c

new file mode 100644

index 0000000..ce4665e

--- /dev/null

+++ b/string.c

@@ -0,0 +1,27 @@

+#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",

+ "zippyops"

+ };

+

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

+

+ printf("string lenght of %s = %1u\n", s[i], my\_strlen(s[i]));

+

+ return 0;

+}

User-b is happy with his changes and he is ready to push his changes.

[user-b@gitlab gitlab]$ git push origin master

Counting objects: 4, done.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 508 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To git@github.com:Zippyops/gitlab.git

35ff72a..92d8b72 master -> master

[user-b@gitlab gitlab]$

User-b changes have been successfully pushed to the repository; now other developers can view his changes by performing clone or update operation.

# Git - Update Operation

User-a performs the clone operation and finds a new file string.c. He wants to know who added this file to the repository and for what purpose, so, he executes the git log command.

[user-a@gitlab ~]$ git clone git@github.com:Zippyops/gitlab.git

Cloning into 'gitlab'...

remote: Counting objects: 6, done.

remote: Compressing objects: 100% (4/4), done.

remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0

Receiving objects: 100% (6/6), done.

[user-a@gitlab ~]$ ls -l

total 0

drwxrwxr-x 3 user-a user-a 50 Dec 3 01:44 gitlab

drwxrwxr-x 3 user-a user-a 34 Dec 1 01:51 user-a\_repo

[user-a@gitlab ~]$ cd gitlab/

[user-a@gitlab gitlab]$ ls

string.c test.txt

The Clone operation will create a new directory inside the current working directory. He changes the directory to newly created directory and executes the git log command.

[user-a@gitlab gitlab]$ git log

commit 92d8b72718ff679342c9ed39a07f592fc743d059

Author: userb <userb@zippyops.com>

Date: Fri Dec 1 02:14:26 2017 -0500

Changed return type of my\_strlen to size\_t

commit 35ff72afbbc575e6586789c6737121aa9f5998ee

Author: usera <usera@zippyops.com>

Date: Fri Dec 1 01:52:15 2017 -0500

Initial commit

After observing the log, he realizes that the file string.c was added by user-b to implement basic string operations. He is curious about user-b’s code. So he opens string.c in text editor and immediately finds a bug. In my\_strlen function, user-b is not using a constant pointer. So, he decides to modify user-b’s code. After modification, the code looks as follows:

[user-a@gitlab gitlab]$ git diff

diff --git a/string.c b/string.c

index ce4665e..a8f84f9 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;

[user-a@gitlab gitlab]$

After testing, he commits his change.

[user-a@gitlab gitlab]$ gcc string.c -o string

[user-a@gitlab gitlab]$ git status -s

M string.c

?? string

[user-a@gitlab gitlab]$ git add string.c

[user-a@gitlab gitlab]$ git commit -m 'Changed char pointer to const char pointer'

[master 933971e] Changed char pointer to const char pointer

1 file changed, 2 insertions(+), 2 deletions(-)

[user-a@gitlab gitlab]$ git log

commit 933971e66cf0374c4f16b534f3d02d14086db284

Author: usera <usera@zippyops.com>

Date: Sun Dec 3 01:51:50 2017 -0500

Changed char pointer to const char pointer

commit 92d8b72718ff679342c9ed39a07f592fc743d059

Author: userb <userb@zippyops.com>

Date: Fri Dec 1 02:14:26 2017 -0500

Changed return type of my\_strlen to size\_t

commit 35ff72afbbc575e6586789c6737121aa9f5998ee

Author: usera <usera@zippyops.com>

Date: Fri Dec 1 01:52:15 2017 -0500

Initial commit

User-a uses git push command to push his changes

[user-a@gitlab gitlab]$ git push origin master

Counting objects: 5, done.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 338 bytes | 0 bytes/s, done.

Total 3 (delta 1), reused 0 (delta 0)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To git@github.com:Zippyops/gitlab.git

92d8b72..933971e master -> master

[user-a@gitlab gitlab]$

## Add New Function

Meanwhile, user-b decides to implement string compare functionality. So he modifies string.c. After modification, the file looks as follows:

[user-b@gitlab gitlab]$ cat string.c

#include <stdio.h>

size\_t my\_strlen(char \*s)

{

char \*p = s;

while (\*p)

++p;

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",

"zippyops"

};

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

printf("string lenght of %s = %1u\n", s[i], my\_strlen(s[i]));

printf("%s\n", my\_strcpy(p1, "Hello, World !!!"));

return 0;

}

After testing, he is ready to push his change

[user-b@gitlab gitlab]$ git diff

diff --git a/string.c b/string.c

index ce4665e..7aae7c3 100644

--- a/string.c

+++ b/string.c

@@ -9,10 +9,16 @@ 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",

@@ -22,6 +28,7 @@ int main(void)

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

printf("string lenght of %s = %1u\n", s[i], my\_strlen(s[i]));

+ printf("%s\n", my\_strcpy(p1, "Hello, World !!!"));

return 0;

}

[user-b@gitlab gitlab]$ git status -s

M string.c

?? string

[user-b@gitlab gitlab]$ git add string.c

[user-b@gitlab gitlab]$ git commit -m "Added my\_strcpy function"

[master a73e133] Added my\_strcpy function

1 file changed, 8 insertions(+), 1 deletion(-)

[user-b@gitlab gitlab]$ git log

commit a73e133e5716c6736708124ac646c1a82f501bb9

Author: userb <userb@zippyops.com>

Date: Sun Dec 3 02:00:57 2017 -0500

Added my\_strcpy function

commit 92d8b72718ff679342c9ed39a07f592fc743d059

Author: userb <userb@zippyops.com>

Date: Fri Dec 1 02:14:26 2017 -0500

Changed return type of my\_strlen to size\_t

commit 35ff72afbbc575e6586789c6737121aa9f5998ee

Author: usera <usera@zippyops.com>

Date: Fri Dec 1 01:52:15 2017 -0500

Initial commit

[user-b@gitlab gitlab]$

User-b is happy with the changes and he wants to push his changes.

[user-b@gitlab gitlab]$ git push origin master

To git@github.com:Zippyops/gitlab.git

! [rejected] master -> master (fetch first)

error: failed to push some refs to 'git@github.com:Zippyops/gitlab.git'

hint: Updates were rejected because the remote contains work that you do

hint: not have locally. This is usually caused by another repository pushing

hint: to the same ref. You may want to first merge the remote changes (e.g.,

hint: 'git pull') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

[user-b@gitlab gitlab]$

But Git is not allowing user-b to push his changes. Because Git identified that remote repository and user-b’s local repository are not in sync. Because of this, he can lose the history of the project. To avoid this mess, Git failed this operation. Now, user-b has to first update the local repository and only thereafter, he can push his own changes.

## Fetch Latest Changes

User-b executes the git pull command to synchronize his local repository with the remote one

[user-b@gitlab gitlab]$ git pull

remote: Counting objects: 3, done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0

Unpacking objects: 100% (3/3), done.

From github.com:Zippyops/gitlab

92d8b72..933971e master -> origin/master

Auto-merging string.c

Merge made by the 'recursive' strategy.

string.c | 4 ++--

1 file changed, 2 insertions(+), 2 deletions(-)

After pull operation, user-b checks the log messages and finds the details of user-a’s commit with commit ID 933971e66cf0374c4f16b534f3d02d14086db284

[user-b@gitlab gitlab]$ git log

commit 441f9b52a5b9a13aa695e914fe6fba918b7d6c8c

Merge: a73e133 933971e

Author: userb <userb@zippyops.com>

Date: Sun Dec 3 02:04:10 2017 -0500

Merge branch 'master' of github.com:Zippyops/gitlab

commit a73e133e5716c6736708124ac646c1a82f501bb9

Author: userb <userb@zippyops.com>

Date: Sun Dec 3 02:00:57 2017 -0500

Added my\_strcpy function

commit 933971e66cf0374c4f16b534f3d02d14086db284

Author: usera <usera@zippyops.com>

Date: Sun Dec 3 01:51:50 2017 -0500

Changed char pointer to const char pointer

commit 92d8b72718ff679342c9ed39a07f592fc743d059

Author: userb <userb@zippyops.com>

Date: Fri Dec 1 02:14:26 2017 -0500

Changed return type of my\_strlen to size\_t

commit 35ff72afbbc575e6586789c6737121aa9f5998ee

Author: usera <usera@zippyops.com>

Date: Fri Dec 1 01:52:15 2017 -0500

Initial commit

Now, user-b’s local repository is fully synchronized with the remote repository. So he can safely push his changes.

[user-b@gitlab gitlab]$ git push origin master

Counting objects: 10, done.

Compressing objects: 100% (6/6), done.

Writing objects: 100% (6/6), 765 bytes | 0 bytes/s, done.

Total 6 (delta 2), reused 0 (delta 0)

remote: Resolving deltas: 100% (2/2), completed with 1 local object.

To git@github.com:Zippyops/gitlab.git

933971e..441f9b5 master -> master

# Git - Stash Operation

Suppose you are implementing a new feature for your product. Your code is in progress and suddenly a customer escalation comes. Because of this, you have to keep aside your new feature work for a few hours. You cannot commit your partial code and also cannot throw away your changes. So you need some temporary space, where you can store your partial changes and later on commit it.

In Git, the stash operation takes your modified tracked files, stages changes, and saves them on a stack of unfinished changes that you can reapply at any time.

[user-b@gitlab gitlab]$ git status -s

M string.c

?? string

Now, you want to switch branches for customer escalation, but you don’t want to commit what you’ve been working on yet; so you’ll stash the changes. To push a new stash onto your stack, run the **git stash** command.

[user-b@gitlab gitlab]$ git stash

Saved working directory and index state WIP on master: 441f9b5 Merge branch 'master' of github.com:Zippyops/gitlab

HEAD is now at 441f9b5 Merge branch 'master' of github.com:Zippyops/gitlab

Now, your working directory is clean and all the changes are saved on a stack. Let us verify it with the git status command.

[user-b@gitlab gitlab]$ git status -s

?? string

[user-b@gitlab gitlab]$

Now you can safely switch the branch and work elsewhere. We can view a list of stashed changes by using the **git stash list** command.

[user-b@gitlab gitlab]$ git stash list

stash@{0}: WIP on master: 441f9b5 Merge branch 'master' of github.com:Zippyops/gitlab

Suppose you have resolved the customer escalation and you are back on your new feature looking for your half-done code, just execute the git stash pop command, to remove the changes from the stack and place them in the current working directory.

[user-b@gitlab gitlab]$ git stash pop

# On branch master

# Changes not staged for commit:

# (use "git add <file>..." to update what will be committed)

# (use "git checkout -- <file>..." to discard changes in working directory)

#

# modified: string.c

#

# Untracked files:

# (use "git add <file>..." 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} (944cb22dcf7dc4f8c3669f2718aff2dc22a2e474)

[user-b@gitlab gitlab]$ git status -s

M string.c

?? string

# Git - Move Operation

As the name suggests, the move operation moves a directory or a file from one location to another. User-a decides to move the source code into src directory. The modified directory structure will appear as follows:

[user-a@gitlab gitlab]$ pwd

/home/user-a/gitlab

[user-a@gitlab gitlab]$ ls

string string.c test.txt

[user-a@gitlab gitlab]$ mkdir src

[user-a@gitlab gitlab]$ git mv string.c src/

[user-a@gitlab gitlab]$ git status -s

R string.c -> src/string.c

?? string

To make these changes permanent, we have to push the modified directory structure to the remote repository so that other developers can see this.

[user-a@gitlab gitlab]$ git commit -m "Modified directory structure"

[master 2b46e8c] Modified directory structure

1 file changed, 0 insertions(+), 0 deletions(-)

rename string.c => src/string.c (100%)

[user-a@gitlab gitlab]$ git push origin master

To git@github.com:Zippyops/gitlab.git

! [rejected] master -> master (fetch first)

error: failed to push some refs to 'git@github.com:Zippyops/gitlab.git'

hint: Updates were rejected because the remote contains work that you do

hint: not have locally. This is usually caused by another repository pushing

hint: to the same ref. You may want to first merge the remote changes (e.g.,

hint: 'git pull') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

[user-a@gitlab gitlab]$ git pull

remote: Counting objects: 6, done.

remote: Compressing objects: 100% (4/4), done.

remote: Total 6 (delta 2), reused 6 (delta 2), pack-reused 0

Unpacking objects: 100% (6/6), done.

From github.com:Zippyops/gitlab

933971e..441f9b5 master -> origin/master

Auto-merging src/string.c

Merge made by the 'recursive' strategy.

src/string.c | 9 ++++++++-

1 file changed, 8 insertions(+), 1 deletion(-)

[user-a@gitlab gitlab]$ git push origin master

Counting objects: 8, done.

Compressing objects: 100% (4/4), done.

Writing objects: 100% (6/6), 640 bytes | 0 bytes/s, done.

Total 6 (delta 0), reused 0 (delta 0)

To git@github.com:Zippyops/gitlab.git

441f9b5..cf18c9c master -> master

In user-b’s local repository, before the pull operation, it will show the old directory structure.

[user-b@gitlab gitlab]$ pwd

/home/user-b/user-b\_repo/gitlab

[user-b@gitlab gitlab]$ ls

string string.c test.txt

[user-b@gitlab gitlab]$ git pull

remote: Counting objects: 6, done.

remote: Compressing objects: 100% (4/4), done.

Unpacking objects: 100% (6/6), done.

remote: Total 6 (delta 0), reused 6 (delta 0), pack-reused 0

From github.com:Zippyops/gitlab

441f9b5..cf18c9c master -> origin/master

Updating 441f9b5..cf18c9c

error: Your local changes to the following files would be overwritten by merge:

string.c

Please, commit your changes or stash them before you can merge.

Aborting

[user-b@gitlab gitlab]$ git add string.c

[user-b@gitlab gitlab]$ git commit -m "Added newline"

[master d6212dc] Added newline

1 file changed, 1 insertion(+)

[user-b@gitlab gitlab]$ git pull

Merge made by the 'recursive' strategy.

string.c => src/string.c | 0

1 file changed, 0 insertions(+), 0 deletions(-)

rename string.c => src/string.c (100%)

[user-b@gitlab gitlab]$ ls

src string test.txt

[user-b@gitlab gitlab]$ ls src/

string.c

# Git - Rename Operation

Till now, both user-a and user-b were using manual commands to compile their project. Now, user-b decides to create Makefile for their project and also give a proper name to the file “string.c”.

[user-b@gitlab gitlab]$ pwd

/home/user-b/user-b\_repo/gitlab

[user-b@gitlab gitlab]$ ls

src string test.txt

[user-b@gitlab gitlab]$ cd src/

[user-b@gitlab src]$ touch Makefile

[user-b@gitlab src]$ git add Makefile

[user-b@gitlab src]$ git mv string.c string\_operations.c

[user-b@gitlab src]$ gcc string\_operations.c -o string\_operations

[user-b@gitlab src]$ git status -s

A Makefile

R string.c -> string\_operations.c

?? string\_operations

?? ../string

Git is showing R before file name to indicate that the file has been renamed.

For commit operation, user-b used -a flag that makes git commit automatically detect the modified files.

[user-b@gitlab src]$ git commit -a -m 'Added Makefile and renamed strings.c to string\_operations.c'

[master 0242ac5] Added Makefile and renamed strings.c to string\_operations.c

2 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 src/Makefile

rename src/{string.c => string\_operations.c} (100%)

user-b wants to add string\_operations to repo.

[user-b@gitlab src]$ git add string\_operations

[user-b@gitlab src]$ git commit -m "Added compiled binary"

[master fb1c1fd] Added compiled binary

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100755 src/string\_operations

After commit, he pushes his changes to the repository.

[user-b@gitlab src]$ git push origin master

Counting objects: 13, done.

Compressing objects: 100% (8/8), done.

Writing objects: 100% (10/10), 1.21 KiB | 0 bytes/s, done.

Total 10 (delta 0), reused 0 (delta 0)

To git@github.com:Zippyops/gitlab.git

cf18c9c..0242ac5 master -> master

Now, other developers can view these modifications by updating their local repository.

# Git - Delete Operation

User-a updates his local repository and finds the compiled binary in the src directory. After viewing the commit message, he realizes that the compiled binary was added by user-b.

[user-a@gitlab src]$ pwd

/home/user-a/gitlab/src

[user-a@gitlab src]$ ls -l

total 16

-rw-rw-r-- 1 user-a user-a 0 Dec 3 02:42 Makefile

-rwxrwxr-x 1 user-a user-a 8648 Dec 3 02:42 string\_operations

-rw-rw-r-- 1 user-a user-a 496 Dec 3 02:42 string\_operations.c

[user-a@gitlab src]$ file string\_operations

string\_operations: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=fa919358ba9c23d18a638dfbc11716209123555e, not stripped

[user-a@gitlab src]$ git log

commit fb1c1fd88e068244b96544cf9cb01bce0d97bf51

Author: userb <userb@zippyops.com>

Date: Sun Dec 3 02:48:10 2017 -0500

Added compiled binary

VCS is used to store the source code only and not executable binaries. So, user-a decides to remove this file from the repository. For further operation, he uses the **git rm** command.

[user-a@gitlab src]$ ls

Makefile string\_operations string\_operations.c

[user-a@gitlab src]$ git rm string\_operations

rm 'src/string\_operations'

[user-a@gitlab src]$ git commit -a -m "Removed executable binary"

[master a1b2889] Removed executable binary

1 file changed, 0 insertions(+), 0 deletions(-)

delete mode 100755 src/string\_operations

After commit, he pushes his changes to the repository.

[user-a@gitlab src]$ git push origin master

Counting objects: 5, done.

Compressing objects: 100% (2/2), done.

Writing objects: 100% (3/3), 306 bytes | 0 bytes/s, done.

Total 3 (delta 1), reused 2 (delta 1)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To git@github.com:Zippyops/gitlab.git

fb1c1fd..a1b2889 master -> master

[user-a@gitlab src]$

# Git - Fix Mistakes

To err is human. So every VCS provides a feature to fix mistakes until a certain point. Git provides a feature that we can use to undo the modifications that have been made to the local repository.

Suppose the user accidentally does some changes to his local repository and then wants to undo these changes. In such cases, the revert operation plays an important role.

## Revert Uncommitted Changes

Let us suppose user-b accidentally modifies a file from his local repository. But he wants to undo his modification. To handle this situation, we can use the **git checkout** command. We can use this command to revert the contents of a file.

[user-b@gitlab src]$ pwd

/home/user-b/user-b\_repo/gitlab/src

[user-b@gitlab src]$ ls -l

total 4

-rw-rw-r-- 1 user-b user-b 0 Dec 3 02:26 Makefile

-rw-rw-r-- 1 user-b user-b 496 Dec 3 02:20 string\_operations.c

[user-b@gitlab src]$ vi string\_operations.c

[user-b@gitlab src]$ git checkout string\_operations.c

[user-b@gitlab src]$ git status –s

# On branch master

nothing to commit, working directory clean

[user-b@gitlab src]$

Further, we can use the **git checkout** command to obtain a deleted file from the local repository. Let us suppose user-a deletes a file from the local repository and we want this file back. We can achieve this by using the same command

[user-a@gitlab src]$ pwd

/home/user-a/gitlab/src

[user-a@gitlab src]$ ls -l

total 4

-rw-rw-r-- 1 user-a user-a 0 Dec 3 02:42 Makefile

-rw-rw-r-- 1 user-a user-a 496 Dec 3 02:42 string\_operations.c

[user-a@gitlab src]$ rm string\_operations.c

[user-a@gitlab src]$ ls -l

total 0

-rw-rw-r-- 1 user-a user-a 0 Dec 3 02:42 Makefile

[user-a@gitlab src]$ git status -s

D string\_operations.c

[user-a@gitlab src]$

Git is showing the letter **D** before the filename. This indicates that the file has been deleted from the local repository

[user-a@gitlab src]$ git checkout string\_operations.c

[user-a@gitlab src]$ ls -l

total 4

-rw-rw-r-- 1 user-a user-a 0 Dec 3 02:42 Makefile

-rw-rw-r-- 1 user-a user-a 496 Dec 3 02:58 string\_operations.c

[user-a@gitlab src]$ git status -s

[user-a@gitlab src]$

Note: We can perform all these operations before commit operation.

## Remove Changes from Staging Area

We have seen that when we perform an add operation, the files move from the local repository to the stating area. If a user accidently modifies a file and adds it into the staging area, he can revert his changes, by using the git checkout command.

In Git, there is one HEAD pointer that always points to the latest commit. If you want to undo a change from the staged area, then you can use the git checkout command, but with the checkout command, you have to provide an additional parameter, i.e., the HEAD pointer. The additional commit pointer parameter instructs the git checkout command to reset the working tree and also to remove the staged changes.

Let us suppose user-a modifies a file from his local repository. If we view the status of this file, it will show that the file was modified but not added into the staging area.

[user-a@gitlab src]$ pwd

/home/user-a/gitlab/src

[user-a@gitlab src]$ ls -l

total 4

-rw-rw-r-- 1 user-a user-a 0 Dec 3 02:42 Makefile

-rw-rw-r-- 1 user-a user-a 496 Dec 3 02:58 string\_operations.c

[user-a@gitlab src]$ vi string\_operations.c

[user-a@gitlab src]$ git status -s

M string\_operations.c

[user-a@gitlab src]$ git add string\_operations.c

Git status shows that the file is present in the staging area, now revert it by using the git checkout command and view the status of the reverted file.

[user-a@gitlab src]$ git checkout HEAD -- string\_operations.c

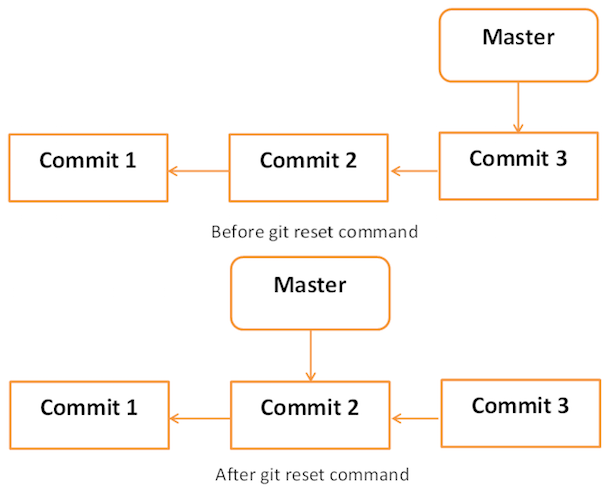
[user-a@gitlab src]$ git status -s

[user-a@gitlab src]$

## Move HEAD Pointer with Git Reset

After doing few changes, you may decide to remove these changes. The Git reset command is used to reset or revert changes. We can perform three different types of reset operations.

Below diagram shows the pictorial representation of Git reset command.



### Soft

Each branch has a HEAD pointer, which points to the latest commit. If we use Git reset command with --soft option followed by commit ID, then it will reset the HEAD pointer only without destroying anything.

.git/refs/heads/master file stores the commit ID of the HEAD pointer. We can verify it by using the git log -1 command.

[user-b@gitlab gitlab]$ pwd

/home/user-b/user-b\_repo/gitlab

[user-b@gitlab gitlab]$ cat .git/refs/heads/master

a1b288970b0d1b29a89f62c0a4f93b61762b2657

Now, view the latest commit ID, which will match with the above commit ID

[user-b@gitlab gitlab]$ git log -2

commit a1b288970b0d1b29a89f62c0a4f93b61762b2657

Author: usera <usera@zippyops.com>

Date: Sun Dec 3 02:51:11 2017 -0500

Removed executable binary

commit fb1c1fd88e068244b96544cf9cb01bce0d97bf51

Author: userb <userb@zippyops.com>

Date: Sun Dec 3 02:48:10 2017 -0500

Added compiled binary

Let us reset the HEAD pointer.

[user-b@gitlab gitlab]$ git reset --soft HEAD~

Now, we just reset the HEAD pointer back by one position. Let us check the contents of .git/refs/heads/master file.

[user-b@gitlab gitlab]$ cat .git/refs/heads/master

fb1c1fd88e068244b96544cf9cb01bce0d97bf51

Commit ID from file is changed, now verify it by viewing commit messages.

[user-b@gitlab gitlab]$ git log -2

commit fb1c1fd88e068244b96544cf9cb01bce0d97bf51

Author: userb <userb@zippyops.com>

Date: Sun Dec 3 02:48:10 2017 -0500

Added compiled binary

commit c3e8a8f3d9b258ab8d739f4b3549998ece04ada6

Author: Zippyops <32255325+Zippyops@users.noreply.github.com>

Date: Sun Dec 3 13:20:04 2017 +0530

Delete string\_operations

### mixed

Git reset with --mixed option reverts those changes from the staging area that have not been committed yet. It reverts the changes from the staging area only. The actual changes made to the working copy of the file are unaffected. The default Git reset is equivalent to the git reset -- mixed.

### hard

If you use --hard option with the Git reset command, it will clear the staging area; it will reset the HEAD pointer to the latest commit of the specific commit ID and delete the local file changes too.

Let us check the commit ID.

[user-b@gitlab gitlab]$ pwd

/home/user-b/user-b\_repo/gitlab

[user-b@gitlab gitlab]$ git log -1

commit fb1c1fd88e068244b96544cf9cb01bce0d97bf51

Author: userb <userb@zippyops.com>

Date: Sun Dec 3 02:48:10 2017 -0500

Added compiled binary

User-b modified a file by adding single-line comment at the start of file.

[user-b@gitlab gitlab]$ vi src/string\_operations.c

[user-b@gitlab gitlab]$ git status -s

D src/string\_operations

M src/string\_operations.c

?? string

User-b adds the modified file to the staging area and verifies it with the git status command.

[user-b@gitlab gitlab]$ git add src/string\_operations.c

[user-b@gitlab gitlab]$ git status

# On branch master

# Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.

# (use "git pull" to update your local branch)

#

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# deleted: src/string\_operations

# modified: src/string\_operations.c

#

# Untracked files:

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

#

# string

[user-b@gitlab gitlab]$

Git status is showing that the file is present in the staging area. Now, reset HEAD with -- hard option.

[user-b@gitlab gitlab]$ git reset --hard fb1c1fd88e068244b96544cf9cb01bce0d97bf51

HEAD is now at fb1c1fd Added compiled binary

Git reset command succeeded, which will revert the file from the staging area as well as remove any local changes made to the file.

[user-b@gitlab gitlab]$ git status -s

?? string

[user-b@gitlab gitlab]$ head -2 src/string\_operations.c

#include <stdio.h>

The head command also shows that the reset operation removed the local changes too.

## Conclusion

When you modify a file in your repository, the change is initially unstaged. In order to commit it, you must stage it—that is, add it to the index—using git add. When you make a commit, the changes that are committed are those that have been added to the index.

git reset changes, at minimum, where the current branch (HEAD) is pointing. The difference between --mixed and --soft is whether or not your index is also modified. So, if we're on branch master with this series of commits:

- A - B - C (master)

HEADpoints to C and the index matches C.

When we run git reset --soft B, master (and thus HEAD) now points to B, but the index still has the changes from C; git status will show them as staged. So if we run git commit at this point, we'll get a new commit with the same changes as C.

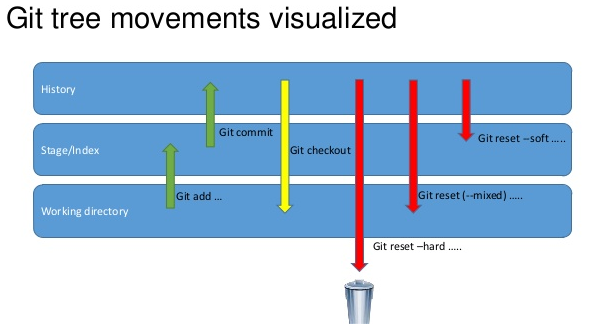
Okay, so starting from here again:

- A - B - C (master)

Now let's do git reset --mixed B. (Note: --mixed is the default option). Once again, master and HEAD point to B, but this time the index is also modified to match B. If we run git commit at this point, nothing will happen since the index matches HEAD. We still have the changes in the working directory, but since they're not in the index, git status shows them as unstaged. To commit them, you would git add and then commit as usual.

And finally, --hard is the same as --mixed (it changes your HEAD and index), except that --hard also modifies your working directory. If we're at C and run git reset --hard B, then the changes added in C, as well as any uncommitted changes you have, will be removed, and the files in your working copy will match commit B. Since you can permanently lose changes this way, you should always run git status before doing a hard reset to make sure your working directory is clean or that you're okay with losing your uncommitted changes.

And finally, a visualization:



# Git - Tag Operation

Tag operation allows giving meaningful names to a specific version in the repository. Suppose user-a and user-b decide to tag their project code so that they can later access it easily.

## Create Tags

Let us tag the current HEAD by using the git tag command. User-a provides a tag name with -a option and provides a tag message with –m option.

[user-a@gitlab gitlab]$ pwd

/home/user-a/gitlab

[user-a@gitlab gitlab]$ git tag -a 'Release\_1\_0' -m 'Tagged basic string operation code' HEAD

If you want to tag a particular commit, then use the appropriate COMMIT ID instead of the HEAD pointer. User-a uses the following command to push the tag into the remote repository.

[user-a@gitlab gitlab]$ git push origin tag Release\_1\_0

Counting objects: 1, done.

Writing objects: 100% (1/1), 178 bytes | 0 bytes/s, done.

Total 1 (delta 0), reused 0 (delta 0)

To git@github.com:Zippyops/gitlab.git

\* [new tag] Release\_1\_0 -> Release\_1\_0

## View Tags

User-a created tags. Now, user-b can view all the available tags by using the Git tag command with –l option.

[user-b@gitlab gitlab]$ git pull

remote: Counting objects: 1, done.

remote: Total 1 (delta 0), reused 1 (delta 0), pack-reused 0

Unpacking objects: 100% (1/1), done.

From github.com:Zippyops/gitlab

\* [new tag] Release\_1\_0 -> Release\_1\_0

Updating fb1c1fd..a1b2889

Fast-forward

src/string\_operations | Bin 8648 -> 0 bytes

1 file changed, 0 insertions(+), 0 deletions(-)

delete mode 100755 src/string\_operations

[user-b@gitlab gitlab]$ cd src/

[user-b@gitlab src]$ pwd

/home/user-b/user-b\_repo/gitlab/src

[user-b@gitlab src]$ git tag -l

Release\_1\_0

User-b uses the Git show command followed by its tag name to view more details about tag.

[user-b@gitlab src]$ git show Release\_1\_0

tag Release\_1\_0

Tagger: usera <usera@zippyops.com>

Date: Sun Dec 3 03:35:18 2017 -0500

Tagged basic string operation code

commit a1b288970b0d1b29a89f62c0a4f93b61762b2657

Author: usera <usera@zippyops.com>

Date: Sun Dec 3 02:51:11 2017 -0500

Removed executable binary

diff --git a/src/string\_operations b/src/string\_operations

deleted file mode 100755

index 2b7a6a9..0000000

Binary files a/src/string\_operations and /dev/null differ

## Delete Tags

User-a uses the following command to delete tags from the local as well as the remote repository.

[user-a@gitlab gitlab]$ git tag

Release\_1\_0

[user-a@gitlab gitlab]$ git tag -d Release\_1\_0

Deleted tag 'Release\_1\_0' (was c2e15a1)

[user-a@gitlab gitlab]$ git push origin :Release\_1\_0

To git@github.com:Zippyops/gitlab.git

- [deleted] Release\_1\_0

# Git - Patch Operation

Patch is a text file, whose contents are similar to Git diff, but along with code, it also has metadata about commits; e.g., commit ID, date, commit message, etc. We can create a patch from commits and other people can apply them to their repository.

User-b implements the strcat function for his project. User-b can create a path of his code and send it to user-a. Then, he can apply the received patch to his code.

char \*my\_strcat(char \*t, char \*s)

{

char \*p = t;

while (\*p)

++p;

while (\*p++ = \*s++)

;

return t;

}

User-b uses the Git format-patch command to create a patch for the latest commit. If you want to create a patch for a specific commit, then use COMMIT\_ID with the format-patch command.

[user-b@gitlab src]$ pwd

/home/user-b/user-b\_repo/gitlab/src

[user-b@gitlab src]$ vi string\_operations.c

[user-b@gitlab src]$ git status -s

M string\_operations.c

?? ../string

[user-b@gitlab src]$ git add string\_operations.c

[user-b@gitlab src]$ vi string\_operations.c

[user-b@gitlab src]$ git commit -m "Added my\_strcat function"

[master 9d59214] Added my\_strcat function

1 file changed, 2 insertions(+)

[user-b@gitlab src]$ git format-patch -1

0001-Added-my\_strcat-function.patch

[user-b@gitlab src]$

The above command creates .patch files inside the current working directory. User-a can use this patch to modify his files. Git provides two commands to apply patches git am and git apply, respectively. Git apply modifies the local files without creating commit, while git am modifies the file and creates commit as well.

[user-b@gitlab src]$ ls -l

total 8

-rw-rw-r-- 1 user-b user-b 476 Dec 3 03:46 0001-Added-my\_strcat-function.patch

-rw-rw-r-- 1 user-b user-b 0 Dec 3 02:26 Makefile

-rw-rw-r-- 1 user-b user-b 614 Dec 3 03:46 string\_operations.c

[user-b@gitlab src]$ exit

exit

[root@gitlab ~]# cp /home/user-b/user-b\_repo/gitlab/src/0001-Added-my\_strcat-function.patch /home/user-a/gitlab/src/

[root@gitlab ~]# chown user-a.user-a /home/user-a/gitlab/src/0001-Added-my\_strcat-function.patch

[root@gitlab ~]# su user-b

[user-b@gitlab root]$ cd ~/user-b\_repo/gitlab/src/

[user-b@gitlab src]$ ls -l

total 8

-rw-rw-r-- 1 user-b user-b 476 Dec 3 03:46 0001-Added-my\_strcat-function.patch

-rw-rw-r-- 1 user-b user-b 0 Dec 3 02:26 Makefile

-rw-rw-r-- 1 user-b user-b 614 Dec 3 03:46 string\_operations.c

[user-b@gitlab src]$

Check the patch fie in user-a

[user-a@gitlab gitlab]$ cd src/

[user-a@gitlab src]$ ls -l

total 8

-rw-r--r-- 1 user-a user-a 476 Dec 3 03:50 0001-Added-my\_strcat-function.patch

-rw-rw-r-- 1 user-a user-a 0 Dec 3 02:42 Makefile

-rw-rw-r-- 1 user-a user-a 496 Dec 3 03:02 string\_operations.c

[user-a@gitlab src]$

To apply patch and create commit, use the following command:

[user-a@gitlab src]$ pwd

/home/user-a/gitlab/src

[user-a@gitlab src]$ git diff

[user-a@gitlab src]$ git status –s

# On branch master

nothing to commit, working directory clean

[user-a@gitlab src]$ git apply 0001-Added-my\_strcat-function.patch

[user-a@gitlab src]$ git status -s

M string\_operations.c

?? 0001-Added-my\_strcat-function.patch

The patch gets applied successfully, now we can view the modifications by using the git diff command.

[user-a@gitlab src]$ git diff

diff --git a/src/string\_operations.c b/src/string\_operations.c

index 85a3579..8383ef5 100644

--- a/src/string\_operations.c

+++ b/src/string\_operations.c

@@ -1,5 +1,13 @@

#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;

@@ -33,3 +41,5 @@ int main(void)

return 0;

}

+

+

# Git - Managing Branches

Branch operation allows creating another line of development. We can use this operation to fork off the development process into two different directions. For example, we released a product for 6.0 version and we might want to create a branch so that the development of 7.0 features can be kept separate from 6.0 bug fixes.

## Create Branch

User-b creates a new branch using the git branch <branch name> command. We can create a new branch from an existing one. We can use a specific commit or tag as the starting point. If any specific commit ID is not provided, then the branch will be created with HEAD as its starting point

[user-b@gitlab gitlab]$ git branch new\_branch

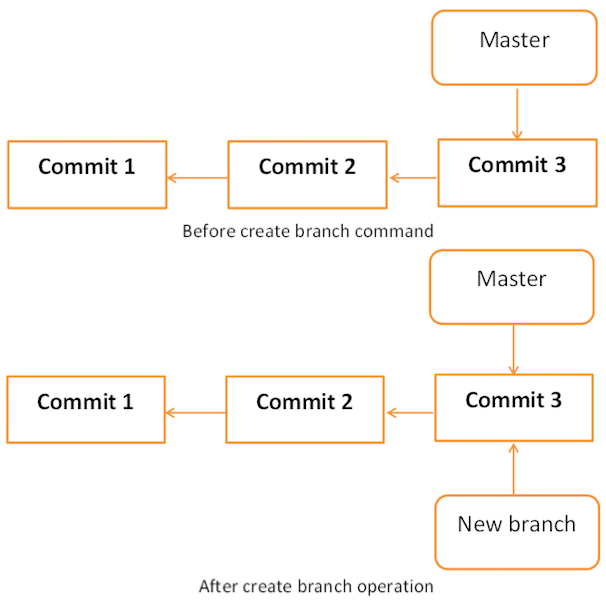
[user-b@gitlab gitlab]$ git branch

\* master

new\_branch

A new branch is created; user-b used the git branch command to list the available branches. Git shows an asterisk mark before currently checked out branch.

The pictorial representation of create branch operation is shown below:



## Switch between Branches

User-b uses the git checkout command to switch between branches.

[user-b@gitlab gitlab]$ git checkout new\_branch

Switched to branch 'new\_branch'

[user-b@gitlab gitlab]$ git branch

master

\* new\_branch

[user-b@gitlab gitlab]$

## Shortcut to Create and Switch Branch

In the above example, we have used two commands to create and switch branches, respectively. Git provides –b option with the checkout command; this operation creates a new branch and immediately switches to the new branch.

[user-b@gitlab gitlab]$ git checkout -b test\_branch

Switched to a new branch 'test\_branch'

[user-b@gitlab gitlab]$ git branch

master

new\_branch

\* test\_branch

## Delete a Branch

A branch can be deleted by providing –D option with git branch command. But before deleting the existing branch, switch to the other branch.

User-b is currently on test\_branch and he wants to remove that branch. So he switches branch and deletes branch as shown below.

[user-b@gitlab gitlab]$ git branch

master

new\_branch

\* test\_branch

[user-b@gitlab gitlab]$ git checkout master

Switched to branch 'master'

Your branch is ahead of 'origin/master' by 2 commits.

(use "git push" to publish your local commits)

[user-b@gitlab gitlab]$ git branch -D test\_branch

Deleted branch test\_branch (was 7521e61).

[user-b@gitlab gitlab]$ git branch

\* master

new\_branch

[user-b@gitlab gitlab]$

## Rename a Branch

User-b decides to add support for wide characters in his string operations project. He has already created a new branch, but the branch name is not appropriate. So he changes the branch name by using –m option followed by the old branch name and the new branch name.

[user-b@gitlab gitlab]$ git branch

\* master

new\_branch

[user-b@gitlab gitlab]$ git branch -m new\_branch wchar\_support

[user-b@gitlab gitlab]$ git branch

\* master

wchar\_support

[user-b@gitlab gitlab]$

## Merge Two Branches

User-b implements a function to return the string length of wide character string. New the code will appear as follows:

[user-b@gitlab gitlab]$ git checkout wchar\_support

Switched to branch 'wchar\_support'

[user-b@gitlab gitlab]$ cd src/

[user-b@gitlab src]$ vi string\_operations.c

[user-b@gitlab src]$ git diff

diff --git a/src/string\_operations.c b/src/string\_operations.c

index 8383ef5..815caed 100644

--- a/src/string\_operations.c

+++ b/src/string\_operations.c

@@ -1,4 +1,13 @@

#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);

+}

+

char \*my\_strcat(char \*t, char \*s)

{

char \*p = t;

[user-b@gitlab src]$

After testing, he commits and pushes his changes to the new branch.

[user-b@gitlab src]$ git status -s

M string\_operations.c

?? 0001-Added-my\_strcat-function.patch

?? ../string

[user-b@gitlab src]$ git add string\_operations.c

[user-b@gitlab src]$ git commit -m 'Added w\_strlen function to return string lenght of wchar\_tstring'

[wchar\_support 3edee63] Added w\_strlen function to return string lenght of wchar\_tstring

1 file changed, 9 insertions(+)

[user-b@gitlab src]$

Note that user-b is pushing these changes to the new branch, which is why he used the branch name wchar\_support instead of master branch.

[user-b@gitlab src]$ git push origin wchar\_support

Counting objects: 15, done.

Compressing objects: 100% (12/12), done.

Writing objects: 100% (12/12), 1.27 KiB | 0 bytes/s, done.

Total 12 (delta 3), reused 0 (delta 0)

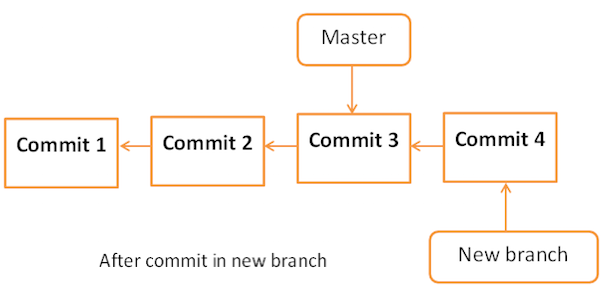
remote: Resolving deltas: 100% (3/3), completed with 1 local object.

To git@github.com:Zippyops/gitlab.git

\* [new branch] wchar\_support -> wchar\_support

[user-b@gitlab src]$

After committing the changes, the new branch will appear as follows:



User-a is curious about what user-b is doing in his private branch and he checks the log from the wchar\_support branch.

[user-a@gitlab src]$ git pull

remote: Counting objects: 12, done.

remote: Compressing objects: 100% (9/9), done.

Unpacking objects: 100% (12/12), done.

remote: Total 12 (delta 3), reused 12 (delta 3), pack-reused 0

From github.com:Zippyops/gitlab

\* [new branch] wchar\_support -> origin/wchar\_support

Already up-to-date.

[user-a@gitlab src]$ git log origin/wchar\_support -2

commit 3edee639dde3c970d06b1945d70eb0e7a4a8b430

Author: userb <userb@zippyops.com>

Date: Sun Dec 3 04:11:14 2017 -0500

Added w\_strlen function to return string lenght of wchar\_tstring

commit 7521e610638924b7aaaa871cb32a1cc7dd6a185e

Author: userb <userb@zippyops.com>

Date: Sun Dec 3 03:54:57 2017 -0500

Added my\_strcat function

[user-a@gitlab src]$

By viewing commit messages, user-a realizes that user-b implemented the strlen function for wide character and he wants the same functionality in the master branch. Instead of re-implementing, he decides to take user-b’s code by merging his branch with the master branch.

[user-a@gitlab gitlab]$ git branch

\* master

[user-a@gitlab gitlab]$ git merge origin/wchar\_support

Updating a1b2889..3edee63

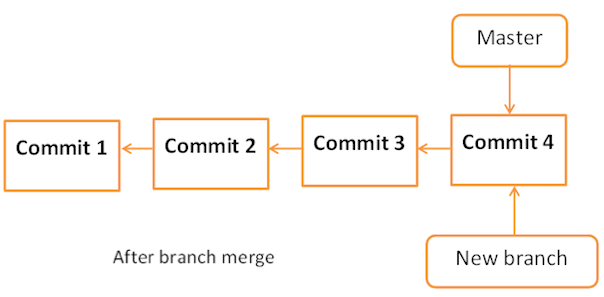
Fast-forward

src/string\_operations.c | 19 +++++++++++++++++++

1 file changed, 19 insertions(+)

[user-a@gitlab gitlab]$

After the merge operation, the master branch will appear as follows:



Now, the branch wchar\_support has been merged with the master branch. We can verify it by viewing the commit message or by viewing the modifications done into the string\_operation.c file.

[user-a@gitlab gitlab]$ git log -1

commit 3edee639dde3c970d06b1945d70eb0e7a4a8b430

Author: userb <userb@zippyops.com>

Date: Sun Dec 3 04:11:14 2017 -0500

Added w\_strlen function to return string lenght of wchar\_tstring

[user-a@gitlab gitlab]$ head -12 src/string\_operations.c

#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);

}

char \*my\_strcat(char \*t, char \*s)

{

[user-a@gitlab gitlab]$

After testing, he pushes his code changes to the master branch.

[user-a@gitlab gitlab]$ git push origin master

Total 0 (delta 0), reused 0 (delta 0)

To git@github.com:Zippyops/gitlab.git

a1b2889..3edee63 master -> master

[user-a@gitlab gitlab]$

## Rebase Branches

The Git rebase command is a branch merge command, but the difference is that it modifies the order of commits.

The Git merge command tries to put the commits from other branches on top of the HEAD of the current local branch. For example, your local branch has commits A−>B−>C−>D and the merge branch has commits A−>B−>X−>Y, then git merge will convert the current local branch to something like A−>B−>C−>D−>X−>Y

The Git rebase command tries to find out the common ancestor between the current local branch and the merge branch. It then pushes the commits to the local branch by modifying the order of commits in the current local branch. For example, if your local branch has commits A−>B−>C−>D and the merge branch has commits A−>B−>X−>Y, then Git rebase will convert the current local branch to something like A−>B−>X−>Y−>C−>D.

When multiple developers work on a single remote repository, you cannot modify the order of the commits in the remote repository. In this situation, you can use rebase operation to put your local commits on top of the remote repository commits and you can push these changes.

# Perform Changes in wchar\_support Branch

User-b is working on the wchar\_support branch. He changes the name of the functions and after testing, he commits his changes.

[user-b@gitlab src]$ git branch

master

\* wchar\_support

[user-b@gitlab src]$ vi string\_operations.c

[user-b@gitlab src]$ git diff

diff --git a/src/string\_operations.c b/src/string\_operations.c

index 815caed..d560632 100644

--- a/src/string\_operations.c

+++ b/src/string\_operations.c

@@ -1,6 +1,6 @@

#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;

while (\*p)

[user-b@gitlab src]$

After verifying the code he commits his changes.

[user-b@gitlab src]$ git status -s

M string\_operations.c

?? 0001-Added-my\_strcat-function.patch

?? ../string

[user-b@gitlab src]$ git add string\_operations.c

[user-b@gitlab src]$ git commit -m 'Changed function name'

[wchar\_support 258a8db] Changed function name

1 file changed, 1 insertion(+), 1 deletion(-)

[user-b@gitlab src]$ git push origin wchar\_support

Counting objects: 7, done.

Compressing objects: 100% (4/4), done.

Writing objects: 100% (4/4), 400 bytes | 0 bytes/s, done.

Total 4 (delta 1), reused 0 (delta 0)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To git@github.com:Zippyops/gitlab.git

3edee63..258a8db wchar\_support -> wchar\_support

[user-b@gitlab src]$

## Perform Changes in Master Branch

Meanwhile in the master branch, user-a also changes the name of the same function and pushes his changes to the master branch.

[user-a@gitlab gitlab]$ git branch

\* master

[user-a@gitlab gitlab]$ vi src/string\_operations.c

[user-a@gitlab gitlab]$ git diff

diff --git a/src/string\_operations.c b/src/string\_operations.c

index 815caed..4fb8046 100644

--- a/src/string\_operations.c

+++ b/src/string\_operations.c

@@ -1,6 +1,7 @@

#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;

while (\*p)

[user-a@gitlab gitlab]$

After verifying diff, he commits his changes.

[user-a@gitlab gitlab]$ git status -s

M src/string\_operations.c

[user-a@gitlab gitlab]$ git add src/string\_operations.c

[user-a@gitlab gitlab]$ git commit -m 'Changed function name from w\_strlen to my\_wc\_strlen'

[master d421678] Changed function name from w\_strlen to my\_wc\_strlen

1 file changed, 2 insertions(+), 1 deletion(-)

[user-a@gitlab gitlab]$ git push origin master

Counting objects: 7, done.

Compressing objects: 100% (4/4), done.

Writing objects: 100% (4/4), 465 bytes | 0 bytes/s, done.

Total 4 (delta 1), reused 0 (delta 0)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To git@github.com:Zippyops/gitlab.git

3edee63..d421678 master -> master

[user-a@gitlab gitlab]$

On the wchar\_support branch, user-b implements strchr function for wide character string.

wchar\_t \*my\_wstrchr(wchar\_t \*ws, wchar\_t wc)

{

while (\*ws)

{

if (\*ws == wc)

return ws;

++ws;

}

return NULL;

}

After testing, he commits and pushes his changes to the wchar\_support branch.

[user-b@gitlab src]$ git branch

master

\* wchar\_support

[user-b@gitlab src]$ vi string\_operations.c

[user-b@gitlab src]$ git diff

diff --git a/src/string\_operations.c b/src/string\_operations.c

index d560632..65dfdd6 100644

--- a/src/string\_operations.c

+++ b/src/string\_operations.c

@@ -1,5 +1,17 @@

#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;

[user-b@gitlab src]$

After verifying, he commits his changes.

[user-b@gitlab src]$ git status -s

M string\_operations.c

?? 0001-Added-my\_strcat-function.patch

?? ../string

[user-b@gitlab src]$ git add string\_operations.c

[user-b@gitlab src]$ git commit -m 'Addded strchr function for wide character string'

[wchar\_support e34be29] Addded strchr function for wide character string

1 file changed, 12 insertions(+)

[user-b@gitlab src]$ git push origin wchar\_support

Counting objects: 7, done.

Compressing objects: 100% (4/4), done.

Writing objects: 100% (4/4), 504 bytes | 0 bytes/s, done.

Total 4 (delta 1), reused 0 (delta 0)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To git@github.com:Zippyops/gitlab.git

258a8db..e34be29 wchar\_support -> wchar\_support

[user-b@gitlab src]$

## Tackle Conflicts

User-a wants to see what user-b is doing on his private branch so, he tries to pull the latest changes from the wchar\_support branch, but Git aborts the operation with the following error message.

[user-a@gitlab src]$ git pull origin wchar\_support

remote: Counting objects: 8, done.

remote: Compressing objects: 100% (6/6), done.

remote: Total 8 (delta 2), reused 8 (delta 2), pack-reused 0

Unpacking objects: 100% (8/8), done.

From github.com:Zippyops/gitlab

\* 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.

[user-a@gitlab src]$

## Resolve Conflicts

From the error message, it is clear that there is a conflict in src/string\_operations.c . He runs the git diff command to view further details.

[user-a@gitlab src]$ git diff

diff --cc src/string\_operations.c

index 4fb8046,65dfdd6..0000000

--- a/src/string\_operations.c

+++ b/src/string\_operations.c

@@@ -1,7 -1,18 +1,23 @@@

#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)

++>>>>>>> e34be296d0062cf833399656425881c53f89021e

{

const wchar\_t \*p = s;

while (\*p)

As both user-a and user-b changed the name of the same function, Git is in a state of confusion and it asks the user to resolve the conflict manually.

User-a decides to keep the function name suggested by user-b, but he keeps the comment added by him, as it is. After removing the conflict markers, git diff will look like this.

[user-a@gitlab src]$ vi string\_operations.c

[user-a@gitlab src]$ git diff

diff --cc src/string\_operations.c

index 4fb8046,65dfdd6..0000000

--- a/src/string\_operations.c

+++ b/src/string\_operations.c

@@@ -1,7 -1,18 +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)

++>>>>>>> e34be296d0062cf833399656425881c53f89021e

{

const wchar\_t \*p = s;

while (\*p)

As user-a has modified the files, he has to commit these changes first and thereafter, he can pull the changes.

[user-a@gitlab src]$ git commit -a -m 'Resolved conflict'

[master 4f8be72] Resolved conflict

[user-a@gitlab src]$ git pull origin wchar\_support.

fatal: Invalid refspec 'wchar\_support.'

[user-a@gitlab src]$ git pull origin wchar\_support

From github.com:Zippyops/gitlab

\* branch wchar\_support -> FETCH\_HEAD

Already up-to-date.

User-a has resolved the conflict, now the pull operation will succeed.