Git - Review Changes

After viewing the commit details, Amarjeet realizes that the string length cannot be negative, that’s why he decides to change the return type of my_strlen function.

Amarjeet uses the **git log** command to view log details.

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

The above command will produce the following result.

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Amarjeet Job <amarjeet@asreetconsulting.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function
```

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

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

The above command will produce the following result −

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Amarjeet Job <amarjeet@asreetconsulting.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 );
   +
}
+
```

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.

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

The above command will produce the following result −

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

Git - Commit Changes

Amarjeet 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.

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

The above command will produce the following result.

```
commit cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Author: Amarjeet Job <amarjeet@asreetconsulting.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Implemented my_strlen function


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Admin Cat <admin@asreetconsulting.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

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

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

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

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

[amarjeet@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
```

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

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

The above command will produce the following result.

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Amarjeet Job <amarjeet@asreetconsulting.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t


commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Admin Cat <admin@asreetconsulting.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Git - Push Operation

Amarjeet 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 Amarjeet’s changes.

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

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

The above command will produce the following result:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Amarjeet Job <amarjeet@asreetconsulting.com>
Date: Wed Sep 11 08:05:26 2013 +0530

Changed return type of my_strlen to size_t
```

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

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

The above command will produce the following result:

```
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Amarjeet Job <amarjeet@asreetconsulting.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 asreetconsultings",
      + "Asreetconsultings Point"
      +
   };
   +
   +
   +
   for (i = 0; i < 2; ++i)
   printf("string lenght of %s = %lu\n", s[i], my_strlen(s[i]));
   +
   +
   return 0;
   +
}
```

Amarjeet is happy with his changes and he is ready to push his changes.

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

The above command will produce the following result:

```
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
```

Amarjeet’s changes have been successfully pushed to the repository; now other developers can view his changes by performing clone or update operation.

Git - Stash Operation

Suppose you are implementing a new feature for your product. Your code is in progress and suddenly a cusadminer 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.

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

Now, you want to switch branches for cusadminer 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.

```
[amarjeet@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
```

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

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

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.

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

Suppose you have resolved the cusadminer 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.

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

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

The above command will produce the following result:

```
# 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)

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

Git - Rename Operation

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

```
[amarjeet@CentOS project]$ pwd
/home/amarjeet/amarjeet_repo/project

[amarjeet@CentOS project]$ ls
README src

[amarjeet@CentOS project]$ cd src/

[amarjeet@CentOS src]$ git add Makefile

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

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

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

For commit operation, Amarjeet used -a flag, that makes git commit auadminatically detect the modified files.

```
[amarjeet@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%)
```

After commit, he pushes his changes to the repository.

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

The above command will produce the following result −

```
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
```

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

Git - Fix Mistakes

o 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 Amarjeet 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.

```
[amarjeet@CentOS src]$ pwd
/home/amarjeet/amarjeet_repo/project/src
```

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

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

[amarjeet@CentOS src]$ git status –s
```

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

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

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

[admin@CentOS src]$ rm string_operations.c

[admin@CentOS src]$ ls -1
Makefile

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

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

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

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

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

Note – We can perform all these operations before commit.

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 Admin 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.

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

[admin@CentOS src]$ git status -s

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

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

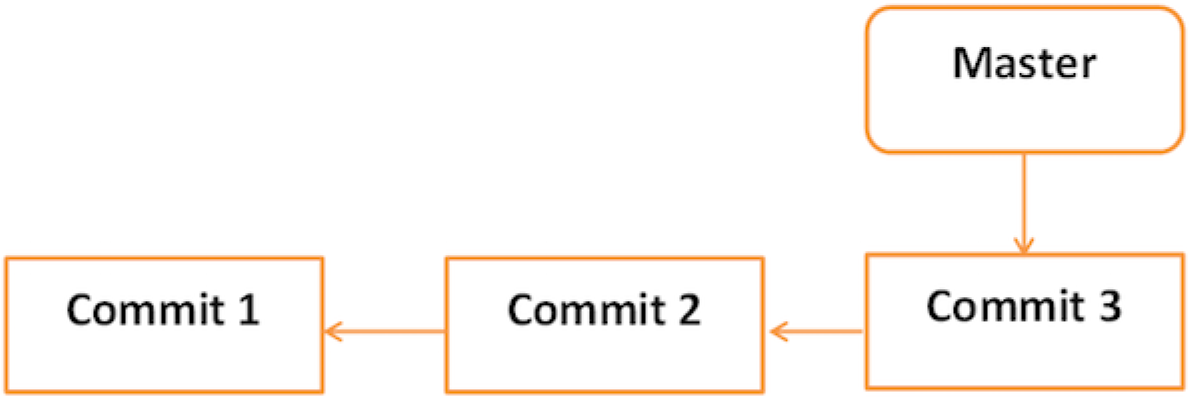
Git status shows that the file is present in the staging area, now revert it by using the git checkout command and view the status of the reverted file.

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

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

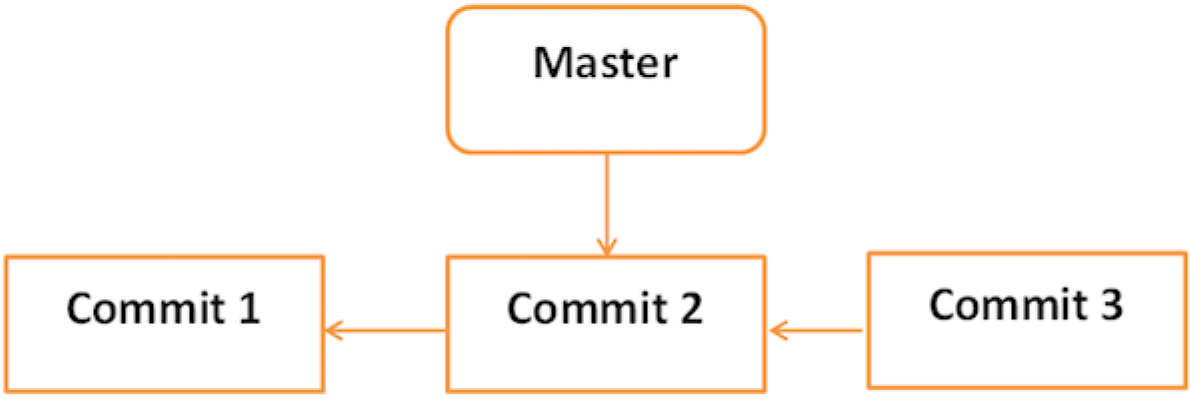
Move HEAD Pointer with Git Reset

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.



Before git reset command



After 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.

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

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

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

The above command will produce the following result.

```
commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Admin Cat <admin@asreetconsulting.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary
```

```
commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Amarjeet Job <amarjeet@asreetconsulting.com>
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary
```

Let us reset the HEAD pointer.

```
[amarjeet@CentOS project]$ 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**.

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

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

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

The above command will produce the following result.

```
commit 29af9d45947dc044e33d69b9141d8d2dad37cc62
Author: Amarjeet Job <amarjeet@asreetconsulting.com>
Date: Wed Sep 11 10:16:25 2013 +0530

Added compiled binary


commit 94f7b26005f856f1a1b733ad438e97a0cd509c1a
Author: Amarjeet Job <amarjeet@asreetconsulting.com>
Date: Wed Sep 11 10:08:01 2013 +0530

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

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.

```
[amarjeet@CentOS src]$ pwd
/home/amarjeet/amarjeet_repo/project/src

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

The above command will produce the following result.

```
commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Admin Cat <admin@asreetconsulting.com>
```

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

Removed executable binary
```

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

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

He verified it by using the git status command.

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

Amarjeet adds the modified file to the staging area and verifies it with the git status command.

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

The above command will produce the following result.

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

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

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

HEAD is now at 5776472 Removed executable 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.

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

Git status is showing that the file has been reverted from the staging area.

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

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

Git - Tag Operation

Tag operation allows giving meaningful names to a specific version in the repository. Suppose Admin and Amarjeet 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. Admin provides a tag name with -a option and provides a tag message with –m option.

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

[admin@CentOS project]$ 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. Admin uses the following command to push the tag into the remote repository.

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

The above command will produce the following result −

```
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
```

View Tags

Admin created tags. Now, Amarjeet can view all the available tags by using the Git tag command with –l option.

```
[amarjeet@CentOS src]$ pwd
/home/amarjeet/amarjeet_repo/project/src

[amarjeet@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.

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

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

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

The above command will produce the following result −

```
tag Release_1_0
Tagger: Admin Cat <admin@asreetconsulting.com>
Date: Wed Sep 11 13:45:54 2013 +0530

Tagged basic string operation code


commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Admin Cat <admin@asreetconsulting.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
```

Delete Tags

Admin uses the following command to delete tags from the local as well as the remote repository.

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

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

[admin@CentOS project]$ git push origin :Release_1_0
To gituser@git.server.com:project.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.

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

Amarjeet 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.

```
[amarjeet@CentOS project]$ pwd
/home/amarjeet/amarjeet_repo/project/src

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

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

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

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

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

The above command creates **.patch** files inside the current working directory. Admin 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.

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

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

[admin@CentOS src]$ git diff

[admin@CentOS src]$ git status –s

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

[admin@CentOS 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.

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

The above command will produce the following result –

```
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 - 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.

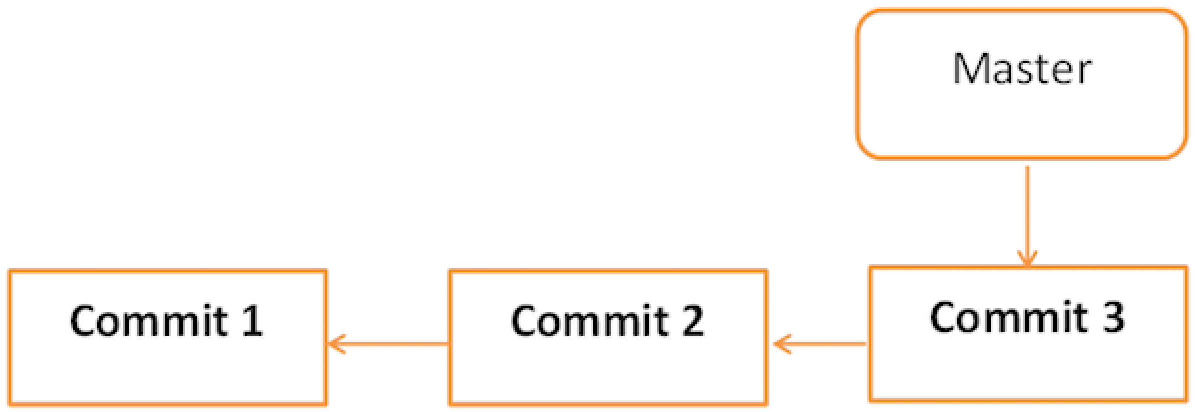
Create Branch

Admin 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.

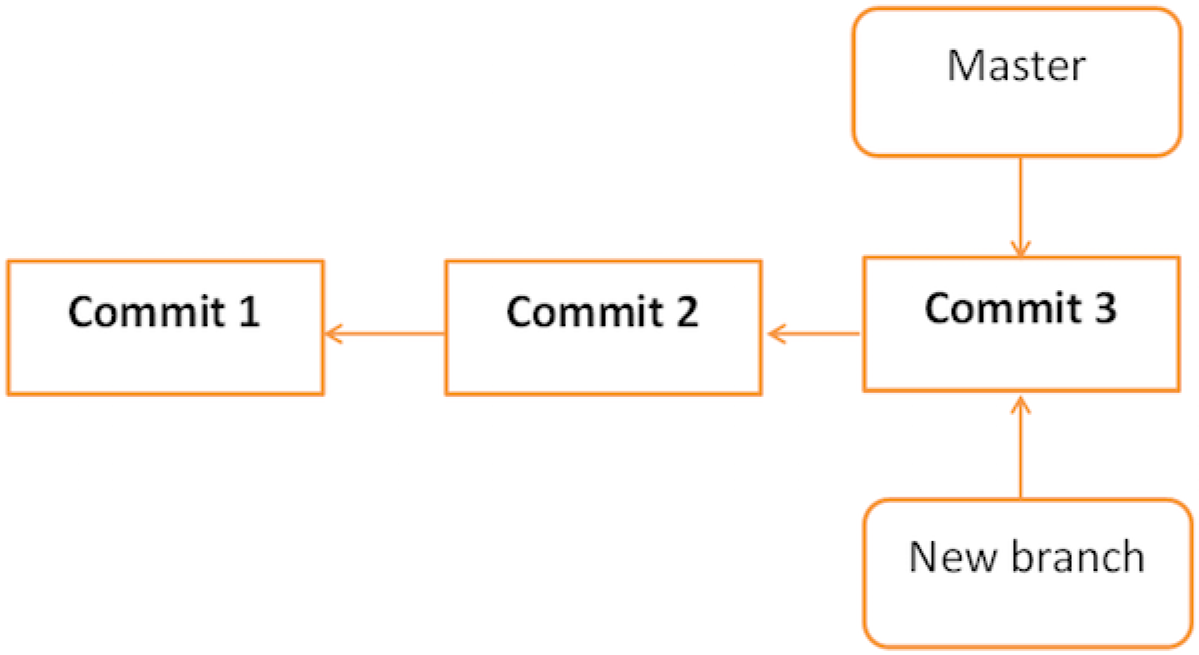
```
[amarjeet@CentOS src]$ git branch new_branch

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

A new branch is created; Admin 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 −



Before create branch command



After create branch operation

Switch between Branches

Amarjeet uses the git checkout command to switch between branches.

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

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.

```
[amarjeet@CentOS src]$ git checkout -b test_branch
```

```
Switched to a new branch 'test_branch'

[amarjeet@CentOS src]$ 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.

Amarjeet is currently on **test_branch** and he wants to remove that branch. So he switches branch and deletes branch as shown below.

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

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

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

Now, Git will show only two branches.

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

Rename a Branch

Amarjeet 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**.

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

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

Now, the git branch command will show the new branch name.

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

Merge Two Branches

Amarjeet implements a function to return the string length of wide character string. New the code will appear as follows −

```
[amarjeet@CentOS src]$ git branch
```

```
master
* wchar_support

[amarjeet@CentOS src]$ pwd
/home/amarjeet/amarjeet_repo/project/src

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

The above command produces the following result –

```
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);
   +
}
```

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

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

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

[amarjeet@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(-)
```

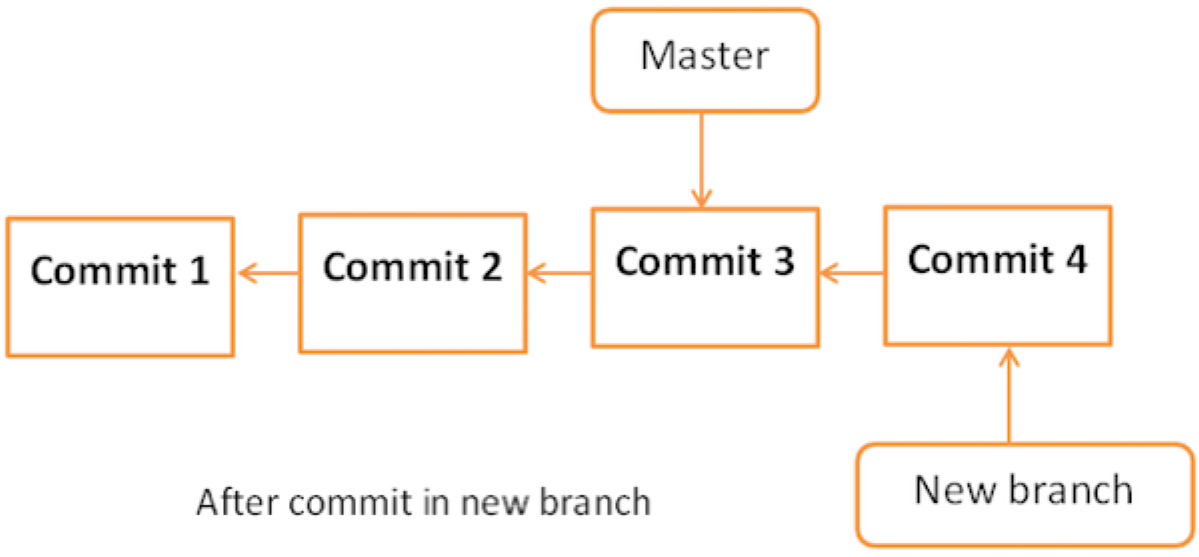
Note that Amarjeet is pushing these changes to the new branch, which is why he used the branch name **wchar_support** instead of **master** branch.

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

The above command will produce the following result.

```
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
```

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



After commit in new branch

Admin is curious about what Amarjeet is doing in his private branch and he checks the log from the **wchar_support** branch.

```
[admin@CentOS src]$ pwd

/home/admin/top_repo/project/src



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

The above command will produce the following result.

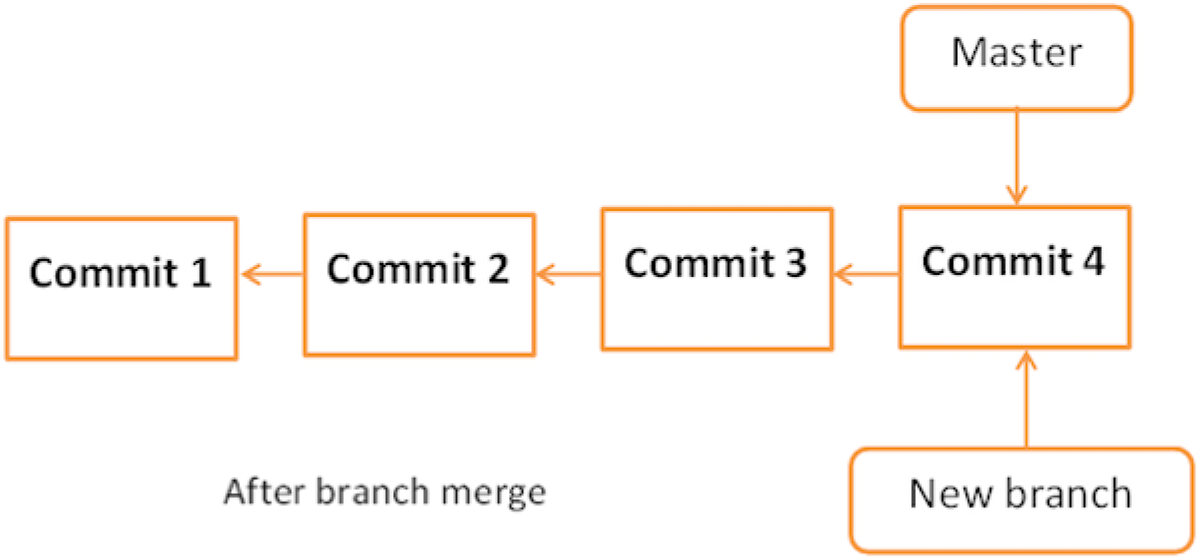
```
commit 64192f91d7cc2bcdf3bf946dd33ece63b74184a3
Author: Amarjeet Job <amarjeet@asreetconsulting.com>
Date: Wed Sep 11 16:10:06 2013 +0530

Added w_strlen function to return string lenght of wchar_t string


commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Admin Cat <admin@asreetconsulting.com>
Date: Wed Sep 11 10:21:20 2013 +0530

Removed executable binary
```

By viewing commit messages, Admin realizes that Amarjeet 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 Amarjeet's code by merging his branch with the master branch.

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

[admin@CentOS project]$ pwd
/home/admin/top_repo/project

[admin@CentOS project]$ git merge origin/wchar_support
Updating 5776472..64192f9
```

```
Fast-forward
src/string_operations.c | 10 ++++++++++
1 files changed, 10 insertions(+), 0 deletions(-)
```

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.

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


[admin@CentOS src]$ git log -1


commit 64192f91d7cc2bcdf3bf946dd33ece63b74184a3

Author: Amarjeet Job

Date: Wed Sep 11 16:10:06 2013 +0530


Added w_strlen function to return string lenght of wchar_t string


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

The above command will produce the following result.

```
#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);
}
```

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

```
[admin@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
```

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.

Git - Handling Conflicts

erform Changes in wchar_support Branch

Amarjeet is working on the **wchar_support** branch. He changes the name of the functions and after testing, he commits his changes.

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

The above command produces the following result −

```
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;
```

After verifying the code he commits his changes.

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

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

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

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

The above command will produce the following result −

```
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
```

Perform Changes in Master Branch

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

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

The above command produces the following result −

```
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;
```

After verifying diff, he commits his changes.

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

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

[admin@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(-)

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

The above command will produce the following result –

```
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
```

On the **wchar_support** branch, Amarjeet implements strchr function for wide character string. After testing, he commits and pushes his changes to the **wchar_support** branch.

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

The above command produces the following result –

```
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;
```

After verifying, he commits his changes.

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

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

[amarjeet@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(-)

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

The above command will produce the following result –

```
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
```

Tackle Conflicts

Admin wants to see what Amarjeet 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.

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

The above command produces the following result –

```
remote: Counting objects: 11, done.
63Git Asreetconsultings
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
Auadminatic merge failed; fix conflicts and then commit the result.
```

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.

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

The above command produces the following result –

```
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;
```

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

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

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

The above command produces the following result.

```
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;
```

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

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

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

Admin has resolved the conflict, now the pull operation will succeed.