Managing branches in Git

Branch operations allow creating different routes of development. We can use this activity to branch the development process into two different directions. For example, we published a version 6 product and we wanted to create a branch to develop 7.0 features that could be kept separate from bug fixes in version 6.0.

Branch operations allow creating different routes of development. We can use this activity to branch the development process into two different directions. For example, we published a version 6 product and we wanted to create a branch to develop 7.0 features that could be kept separate from bug fixes in version 6.0.

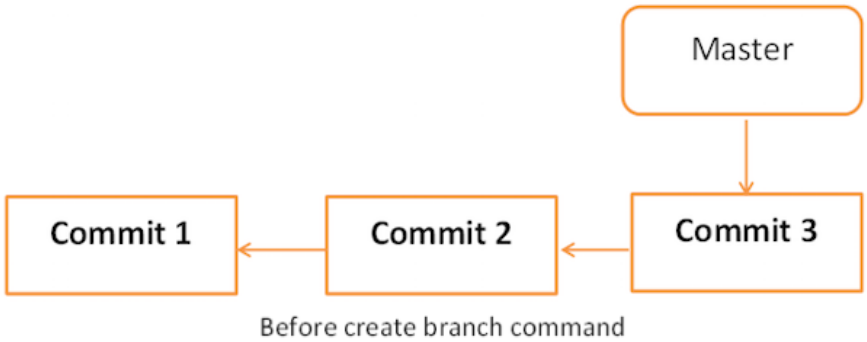
Branching

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

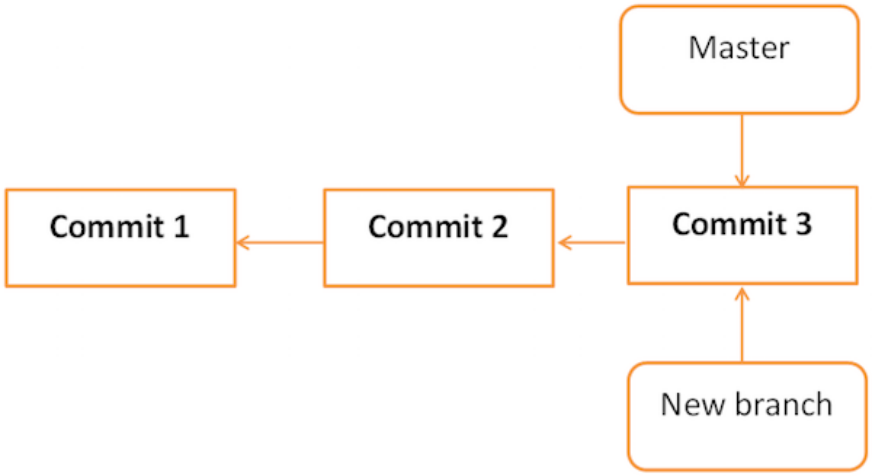
```
[jerry @ CentOS src] $ git branch new_branch

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

A new branch is created; Tom uses the git branch command to list the available branches. Git indicates an asterisk before checking the current branch.

The figure below depicts branching activity:



Before create branch command



After create branch operation

Switch between branches

Jerry uses the git checkout command to switch between branches:

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

How to turn off to branch and switch between branches

In the above example, we used two separate commands to create and switch between branches. Git provides the -b option with the checkout command; This activity creates a new branch and immediately converts to the new branch.

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

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

Delete a branch

A branch can be deleted using the -D option with the git branch command. But before deleting an existing branch, go to another branch.

Jerry is currently on the test_branch branch and he wants to remove that branch. So he switched to another branch and deleted the branch as below:

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

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

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

Now, Git will only have two branches.

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

Rename a branch

Jerry decided to add support for the wide characters in his project sequence of activities. He created a new branch, but the branch name is not provided correctly. So he changed the branch name using the -m option followed by the old branch name and new branch name.

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

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

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

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

Merging two branches

Jerry performs a function to return the string length of a wide character string. A new code will appear as follows:

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

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

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

The above command will produce the following result:

```
ta / src / string_operations.cb / src / string_operations.c
index 8ab7f42.8fb4b00 100644
--- a / src / string_operations.c
+++ b / src / string_operations.c
@@ -1.4 +1,14 @@
#include
+ # include
+
+ size_t w_strlen (const wchar_t * s)
+
{
+
const wchar_t * p = s;
+
+
while (* p)
+ ++ p;
+ return (p - s);
+
}
```

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

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

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

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

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

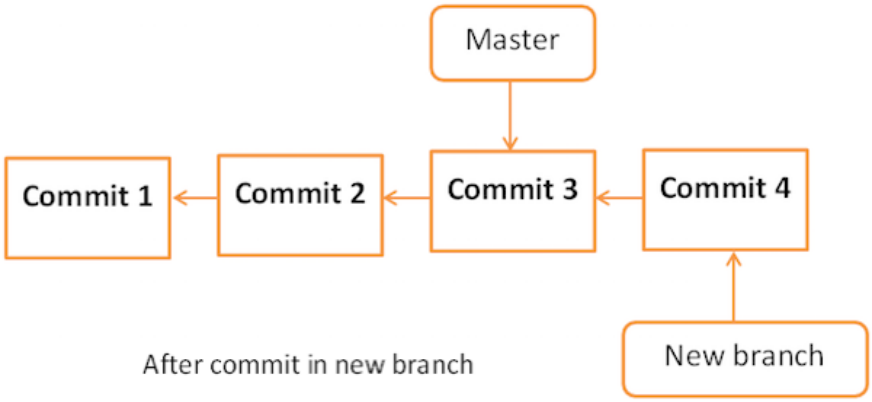
Note that Jerry is pushing these changes to the new branch, which is why he uses the branch named wchar_support instead of the master branch.

```
[jerry@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:



Tom is curious about what Jerry is doing in his private branch and he checks the log from the wchar_support branch.

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

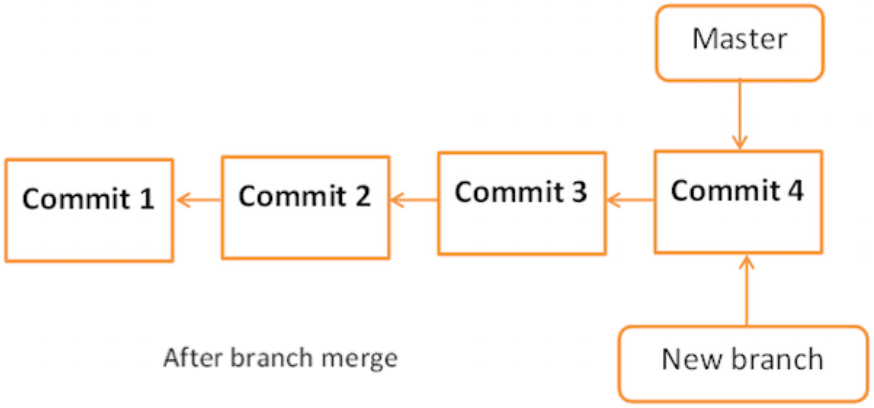
The above command will produce the following result:

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

Added w_strlen function to return string lenght of wchar_t string


commit 577647211ed44fe2ae479427a0668a4f12ed71a1
Author: Tom Cat
Date: Wed Sep 11 10:21:20 2013 +0530

Removed binary executable
```

By observing commit messages, Tom realizes that Jerry performs the strlen function for the extended character and he wants the same functionality in the master branch. Instead of repeating the above steps, he decides to take Jerry's code by merging his branch with the master branch.

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

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

[tom @ CentOS project] $ git merge origin / wchar_support
Updating 5776472.64192f9
Fast-forward
src / string_operations.c |  10 ++++++++++
1 files changed, 10 insertions (+), 0 deletions (-)
```

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



Now, the wchar_support branch has been imported with the master branch. We verify it by observing commit information or observing the modifications made in file string_operation.c.

```
[ tom@CentOS project ] $ cd src / [ tom@CentOS src ] $ git log  - 1  commit  641
Date : Wed Sep 11 16 : 10 : 06 2013 + 0530 Added  w_strlen  function  to  return
```

The above command will produce the following result:

```
#include
#include
size_t w_strlen (const wchar_t * s)
{
const wchar_t * p = s;

while (* p)
++ p;

return (p - s);
}
```

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

```
[ tom@CentOS src ] $ git push origin master  Total 0 ( delta  0 ),  reused  0 (
?>  master
```

Rebase branches

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

The git merge command attempts to place commits from another branch on top of the HEAD of the current internal branch. For example, your local branch has commits A?> B?> C?> D and the merging branch has commits A?> B?> X?> Y, then the git merge command changes branch Current internal into a branch like A?> B?> C?> D?> X?> Y

The git rebase command tries to find the root between the current internal branch and the merged branch. It pushes commits to the local branch by modifying the order of commits in the current local branch. For example, the two branches have the same commits, then the git rebase command will convert the current internal branch into a branch like A> B> X> Y> C> D.

When there are many programmers working on a remote repository, you cannot edit the order of commits in this repository. In this situation, you can use rebase activity to place your internal commits on the top of remote repository commits and you can push changes.

According to Tutorialspoint

Previous post: Patch operation in Git

Next article: Handling Conflict in Git