Git 基本操作 Git 查看提交历史

Git 分支管理

几乎每一种版本控制系统都以某种形式支持分支。使用分支意味着你可以从开发主线上分离开来，然后在不影响主线的同时继续工作。

有人把 Git 的分支模型称为"必杀技特性"，而正是因为它，将 Git 从版本控制系统家族里区分出来。

创建分支命令：

```
git branch (branchname)
```

切换分支命令:

```
git checkout (branchname)
```

当你切换分支的时候，Git 会用该分支的最后提交的快照替换你的工作目录的内容， 所以多个分支不需要多个目录。

合并分支命令:

```
git merge
```

你可以多次合并到统一分支， 也可以选择在合并之后直接删除被并入的分支。

Git 分支管理

列出分支

列出分支基本命令：

```
git branch
```

没有参数时，git branch 会列出你在本地的分支。

```
$ git branch
* master
```

此例的意思就是，我们有一个叫做"master"的分支，并且该分支是当前分支。

当你执行 git init 的时候，缺省情况下 Git 就会为你创建"master"分支。

如果我们要手动创建一个分支。执行 git branch (branchname) 即可。

```
$ git branch testing
$ git branch
```

```
* master
  testing
```

现在我们可以看到，有了一个新分支 testing。

当你以此方式在上次提交更新之后创建了新分支，如果后来又有更新提交， 然后又切换到了"testing"分支，Git 将还原你的工作目录到你创建分支时候的样子

接下来我们将演示如何切换分支，我们用 git checkout (branch) 切换到我们要修改的分支。

```
$ ls
README
$ echo 'runoob.com' > test.txt
$ git add .
$ git commit -m 'add test.txt'
[master 048598f] add test.txt
 2 files changed, 1 insertion(+), 3 deletions(-)
 delete mode 100644 hello.php
 create mode 100644 test.txt
$ ls
README        test.txt
$ git checkout testing
Switched to branch 'testing'
$ ls
README        hello.php
```

当我们切换到"testing"分支的时候，我们添加的新文件test.txt被移除了, 原来被删除的文件hello.php文件又出现了。切换回"master"分支的时候，它们有重新出现了。

```
$ git checkout master
Switched to branch 'master'
$ ls
README        test.txt
```

我们也可以使用 git checkout -b (branchname) 命令来创建新分支并立即切换到该分支下，从而在该分支中操作。

```
$ git checkout -b newtest
Switched to a new branch 'newtest'
$ git rm test2.txt
rm 'test2.txt'
$ ls
README        test.txt
$ git commit -am 'removed test2.txt'
[newtest 556f0a0] removed test2.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 test2.txt
$ git checkout master
```

```
Switched to branch 'master'
$ ls
README        test.txt    test2.txt
```

如你所见，我们创建了一个分支，在该分支的上下文中移除了一些文件，然后切换回我们的主分支，那些文件又回来了。

使用分支将工作切分开来，从而让我们能够在不同上下文中做事，并来回切换。

删除分支

删除分支命令：

```
git branch -d (branchname)
```

例如我们要删除"testing"分支：

```
$ git branch
* master
  testing
$ git branch -d testing
Deleted branch testing (was 85fc7e7).
$ git branch
* master
```

分支合并

一旦某分支有了独立内容，你终究会希望将它合并回到你的主分支。 你可以使用以下命令将任何分支合并到当前分支中去：

```
git merge
```

```
$ git branch
* master
  newtest
$ ls
README        test.txt    test2.txt
$ git merge newtest
Updating 2e082b7..556f0a0
Fast-forward
 test2.txt | 1 -
 1 file changed, 1 deletion(-)
 delete mode 100644 test2.txt
$ ls
README        test.txt
```

以上实例中我们将 newtest 分支合并到主分支去，test2.txt 文件被删除。

合并冲突

合并并不仅仅是简单的文件添加、移除的操作，Git 也会合并修改。

```
$ git branch
* master
$ cat test.txt
runoob.com
```

首先，我们创建一个叫做"change_site"的分支，切换过去，我们将内容改为 www.runoob.com 。

```
$ git checkout -b change_site
Switched to a new branch 'change_site'
$ vim test.txt
$ head -1 test.txt
www.runoob.com
$ git commit -am 'changed the site'
[change_site d7e7346] changed the site
 1 file changed, 1 insertion(+), 1 deletion(-)
```

将修改的内容提交到 "change_site" 分支中。 现在，假如切换回 "master" 分支我们可以看内容恢复到我们修改前的，我们再次修改test.txt文件。

```
$ git checkout master
Switched to branch 'master'
$ head -1 test.txt
runoob.com
$ vim test.txt
$ cat test.txt
runoob.com
新增加一行
$ git diff
diff --git a/test.txt b/test.txt
index 704cce7..f84c2a4 100644
--- a/test.txt
+++ b/test.txt
@@ -1 +1,2 @@
 runoob.com
+新增加一行
$ git commit -am '新增加一行'
[master 14b4dca] 新增加一行
 1 file changed, 1 insertion(+)
```

现在这些改变已经记录到我的 "master" 分支了。接下来我们将 "change_site" 分支合并过来。

```
 $ git merge change_site
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
$ cat test.txt
<<<<<<< HEAD
runoob.com
新增加一行
=======
www.runoob.com
>>>>>>> change_site
```

我们将前一个分支合并到 "master" 分支，一个合并冲突就出现了，接下来我们需要手动去修改它。

```
$ vim test.txt
$ cat test.txt
www.runoob.com
新增加一行
$ git diff
diff --cc test.txt
index f84c2a4,bccb7c2..0000000
--- a/test.txt
+++ b/test.txt
@@@ -1,2 -1,1 +1,2 @@@
- runoob.com
+ www.runoob.com
 +新增加一行
```

在 Git 中，我们可以用 git add 要告诉 Git 文件冲突已经解决

```
$ git status -s
UU test.txt
$ git add test.txt
$ git status -s
M  test.txt
$ git commit
[master 88afe0e] Merge branch 'change_site'
```

现在我们成功解决了合并中的冲突，并提交了结果。

← Git 基本操作 Git 查看提交历史 →