

Git 分支管理详解

大纲:

1. 前言
2. 创建分支
3. 切换分支
4. 合并分支（快速合并）
5. 删除分支
6. 分支合并冲突
7. 合并分支（普通合并）
8. 分支管理策略
9. 团队多人开发协作
10. 总结

注，测试机 CentOS 5.5 x86_64, Git 服务器版本: `git version 1.8.2.1`, 客户端版本: `git version 1.9.2.msysgit.0`。所有软件请到这里下载: <http://msysgit.github.io/>。

1. 前言

在上一篇博客中我们主要讲解了Git 远程仓库，相信大家对远程的Git仓库有一定的了解，嘿嘿。在这一篇博客中我们来在大家讲解一下Git 分支管理，这可以说是Git的又一大特点。下面我们就来学习一下Git分支管理吧。

我们先来说一个简单的案例吧，你们团队中有多个人再开发一下项目，一同事再开发一个新的功能，需要一周时间完成，他写了其中的30%还没有写完，如果他提交了版本，那么团队中的其它人就不能继续开发了。但是等到他全部写完再全部提交，大家又看不到他的开发进度，也不能继续干活，这如何是好呢？

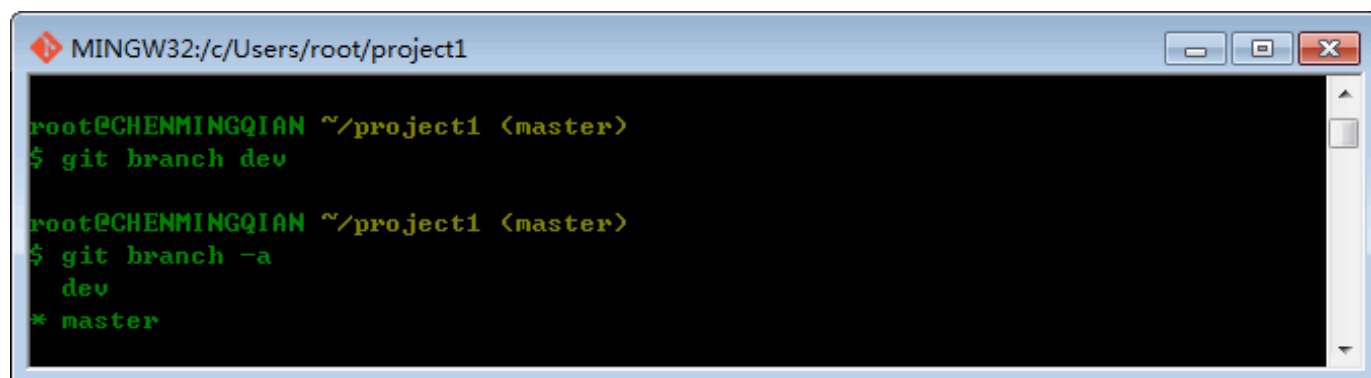
对于上面的这个问题，我们就可以用分支管理的办法来解决，一同事开发新功能他可以创建一个属于他自己的分支，其它同事暂时看不到，继续在开发分支（一般都有多个分支）上干活，他在自己的分支上干活，等他全部开发完成，再一次性的合并到开发分支上，这样我们既可知道他的开发进度，又不影响大家干活，是不是很方便呢？

大家可能会说了，你Git的分支功能人家SVN也有啊，也没什么特殊的嘛。但我想说你那个创建和切换速度怎么样呢？嘿嘿，我想说谁用谁知道啊！但Git呢，无论你创建还是切换或者删除都很快哦！

分支本质上其实就是一个指向某次提交的可变指针。Git 的默认分支名字为 `master` 。而我们是怎么知道当前处于哪个分支当中呢？答案就是在于 `HEAD` 这个十分特殊的指针，它专门用于指向于本地分支中的当前分支。我们可以简单理解为：`commit <- branch <- HEAD` （注，本来我们详细说一下HEAD的，这个东西真不好说，google了一下也没几个大神说这个，嘿嘿。下面我就通过讲解帮助大家理解，简单的说HEAD就 是指向于本地分支中的当前分支，如下图：） 下面我们来创建分支。

2. 创建分支

当我们需要调试某个Bug或者尝试添加或修改程序中的某个模块，而又不能影响主分支的开发时。就可以通过创建分支来满足需求。创建分支相当于是创建一个新的分支指针指向当前所在的提交。我们在 `Commit3` 上创建 `dev` 分支：

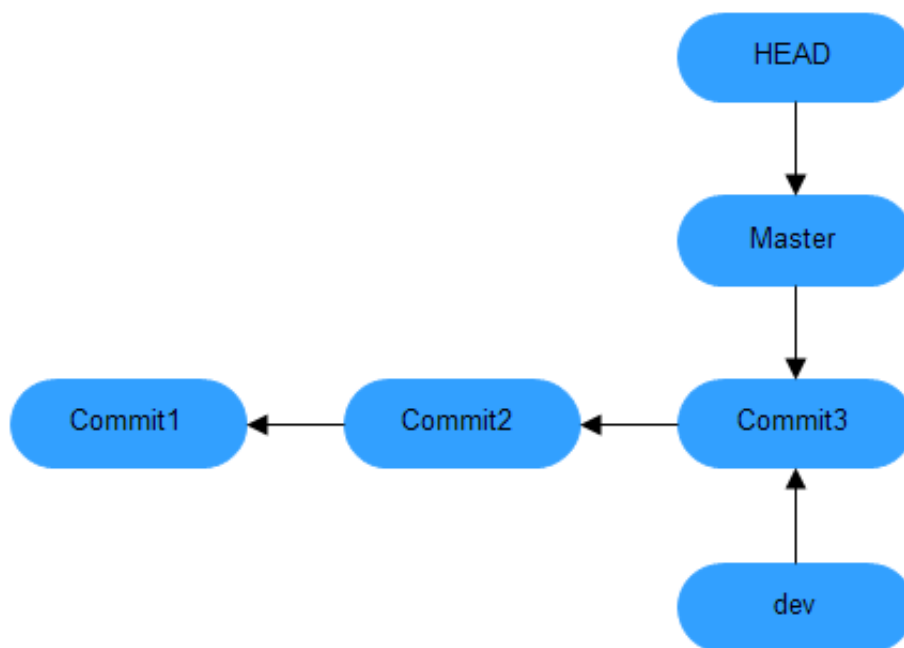


```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <master>
$ git branch dev

root@CHENMINGQIAN ~/project1 <master>
$ git branch -a
  dev
* master
```

如下图所示，`dev` 分支指向 `Commit3`。



从上图可知，虽然我们创建了一个新分支，但是 `HEAD` 仍然指向 `master` 。如果希望在创建分支的同时切换到新分支上，我们可以通过以下命令实现：

```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <dev>
$ git checkout -b dev
```

git checkout命令加上-b参数表示创建并切换分支上。

3. 切换分支

切换当前分支我们可以用以下命令实现：

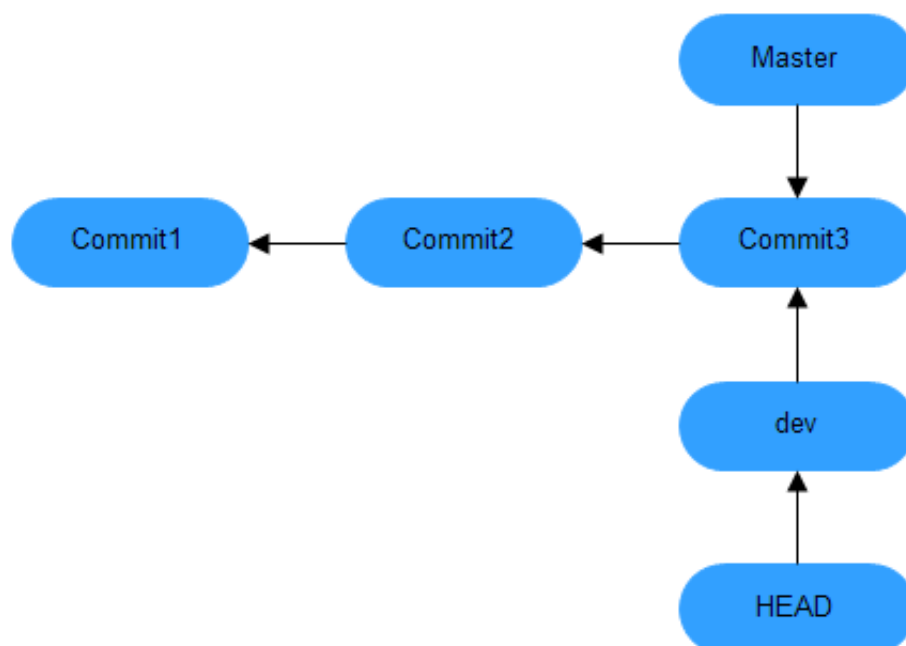
```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <master>
$ git checkout dev
Switched to branch 'dev'

root@CHENMINGQIAN ~/project1 <dev>
$ git branch -a
* dev
  master

root@CHENMINGQIAN ~/project1 <dev>
$
```

git branch -a 命令可以查看所有分支，现在我们HEAD指针便指向dev分支，大家可以在上图中看到dev分支上有个*号。



下面我们修改一下readme.txt中的内容，并在dev分支上提交一下。如下图：

```
MINGW32:/c/Users/root/project1
root@CHENMINGQIAN ~/project1 <dev>
$ git branch -a
* dev
  master

root@CHENMINGQIAN ~/project1 <dev>
$ vim readme.txt

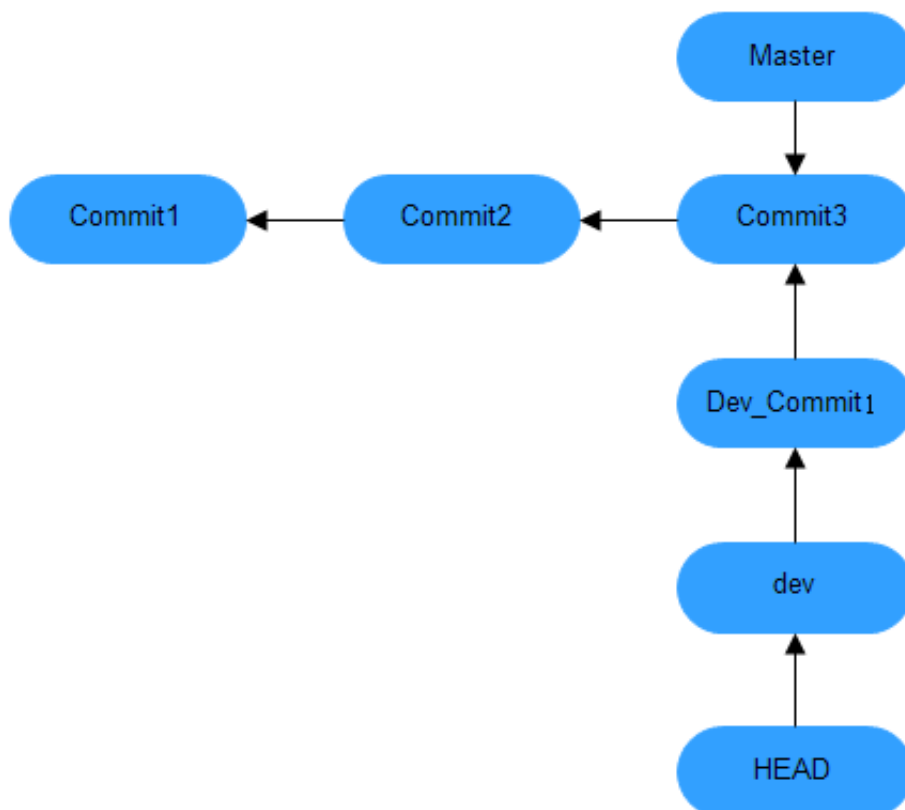
root@CHENMINGQIAN ~/project1 <dev>
$ cat readme.txt
This is git test!

root@CHENMINGQIAN ~/project1 <dev>
$ git add readme.txt

root@CHENMINGQIAN ~/project1 <dev>
$ git commit -m "add a line"
[dev 63ab81c] add a line
1 file changed, 1 insertion(+)

root@CHENMINGQIAN ~/project1 <dev>
$
```

用流程图演示上述过程如下：



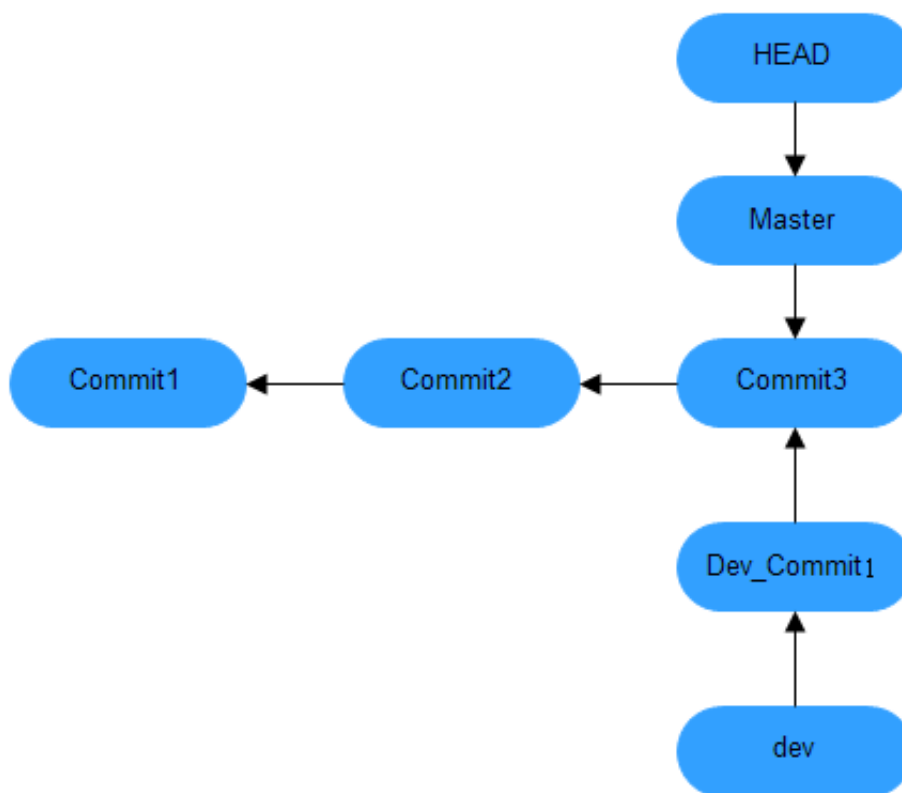
现在我们在dev分支上完成工作，现在到master分支上。如下图：

```
MINGW32/c/Users/root/project1
root@CHENMINGQIAN ~/project1 <dev>
$ git checkout master
Switched to branch 'master'
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

root@CHENMINGQIAN ~/project1 <master>
$ git branch -a
dev
* master

root@CHENMINGQIAN ~/project1 <master>
$
```

流程图表示如下：



切换回master分支后，再查看一个readme.txt文件，如下图：

```
MINGW32/c/Users/root/project1
root@CHENMINGQIAN ~/project1 <master>
$ git branch -a
dev
* master

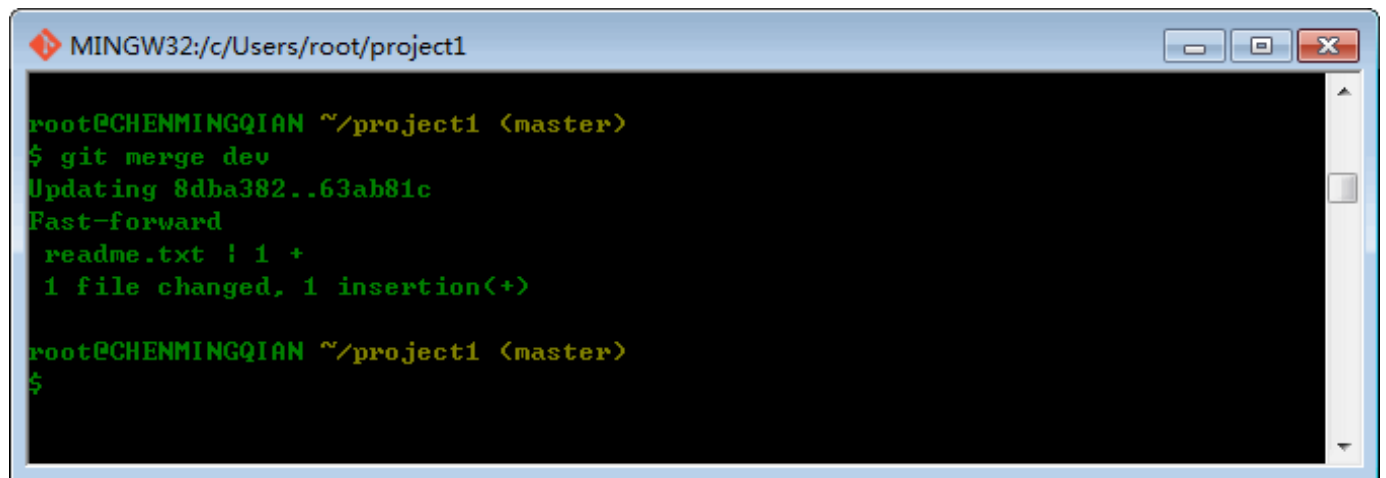
root@CHENMINGQIAN ~/project1 <master>
$ cat readme.txt

root@CHENMINGQIAN ~/project1 <master>
$
```

大家可以看到我们刚才增加的一行内容不见了，嘿嘿。因为那个提交是在dev分支上，而master分支没有变化。好了，下面我们来演示一下合并分支。

4. 合并分支（快速合并）

现在，我们把dev分支的工作成果合并到master分支上，如下图：

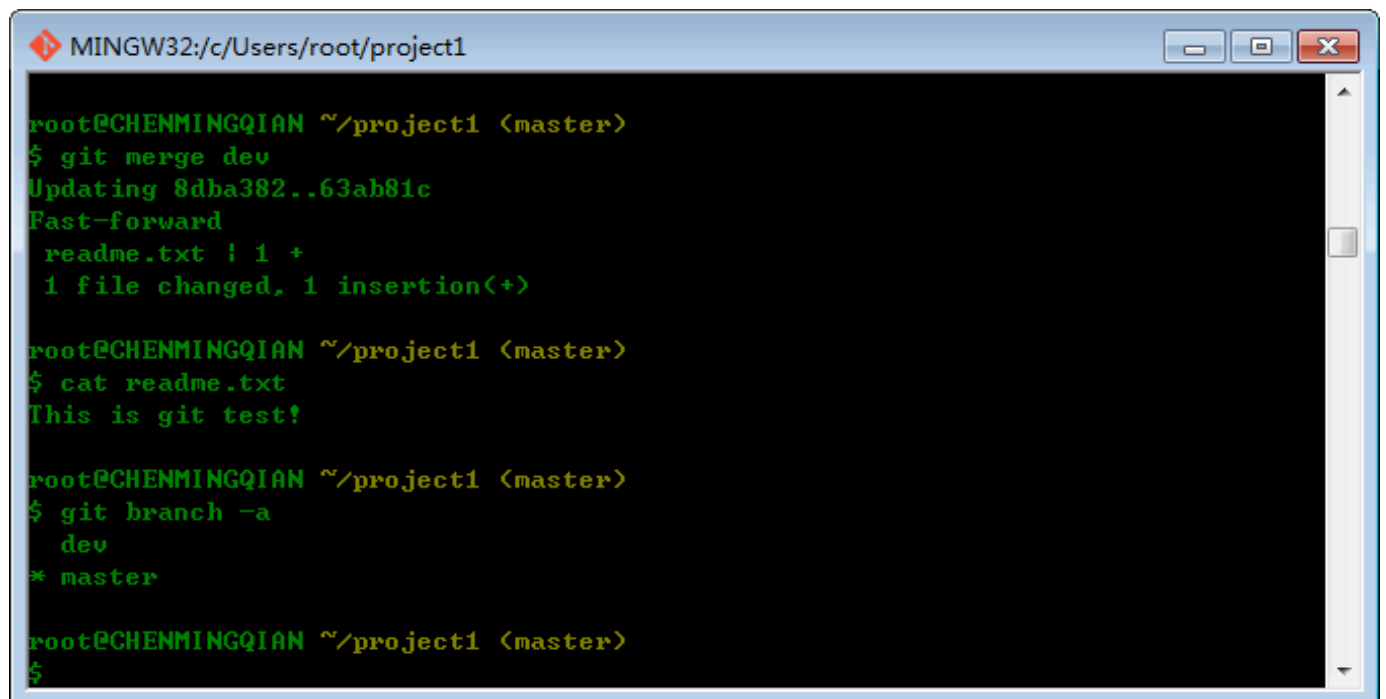


```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <master>
$ git merge dev
Updating 8dba382..63ab81c
Fast-forward
 readme.txt | 1 +
 1 file changed, 1 insertion(+)

root@CHENMINGQIAN ~/project1 <master>
$
```

git merge 命令用于合并指定分支到当前分支。合并后，再查看readme.txt的内容，就可以看到，和dev分支的最新提交是完全一样的。如下图：



```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <master>
$ git merge dev
Updating 8dba382..63ab81c
Fast-forward
 readme.txt | 1 +
 1 file changed, 1 insertion(+)

root@CHENMINGQIAN ~/project1 <master>
$ cat readme.txt
This is git test!

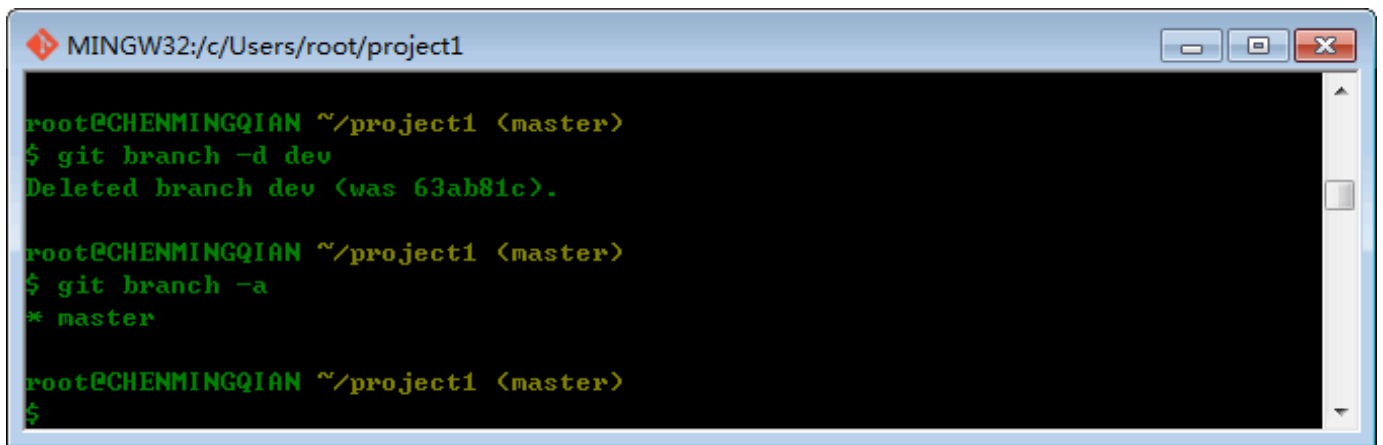
root@CHENMINGQIAN ~/project1 <master>
$ git branch -a
 dev
* master

root@CHENMINGQIAN ~/project1 <master>
$
```

大家注意到上面的Fast-forward信息，Git告诉我们，这次合并是“快进模式”，也就是直接把master指向dev的当前提交，所以合并速度 非常快。当然，也不是每次合并都能Fast-forward，我们后面会将其他方式的合并。合并完成后，就可以放心地删除dev分支了。

5. 删除分支

下面我们来演示一下删除分支，如下图：

A terminal window titled 'MINGW32:/c/Users/root/project1' showing a series of Git commands and their outputs. The user is in the 'master' branch of a project located at '~/project1'. The commands executed are: 'git branch -d dev' which results in 'Deleted branch dev (was 63ab81c).', 'git branch -a' which lists '* master', and a final '\$' prompt.

```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <master>
$ git branch -d dev
Deleted branch dev (was 63ab81c).

root@CHENMINGQIAN ~/project1 <master>
$ git branch -a
* master

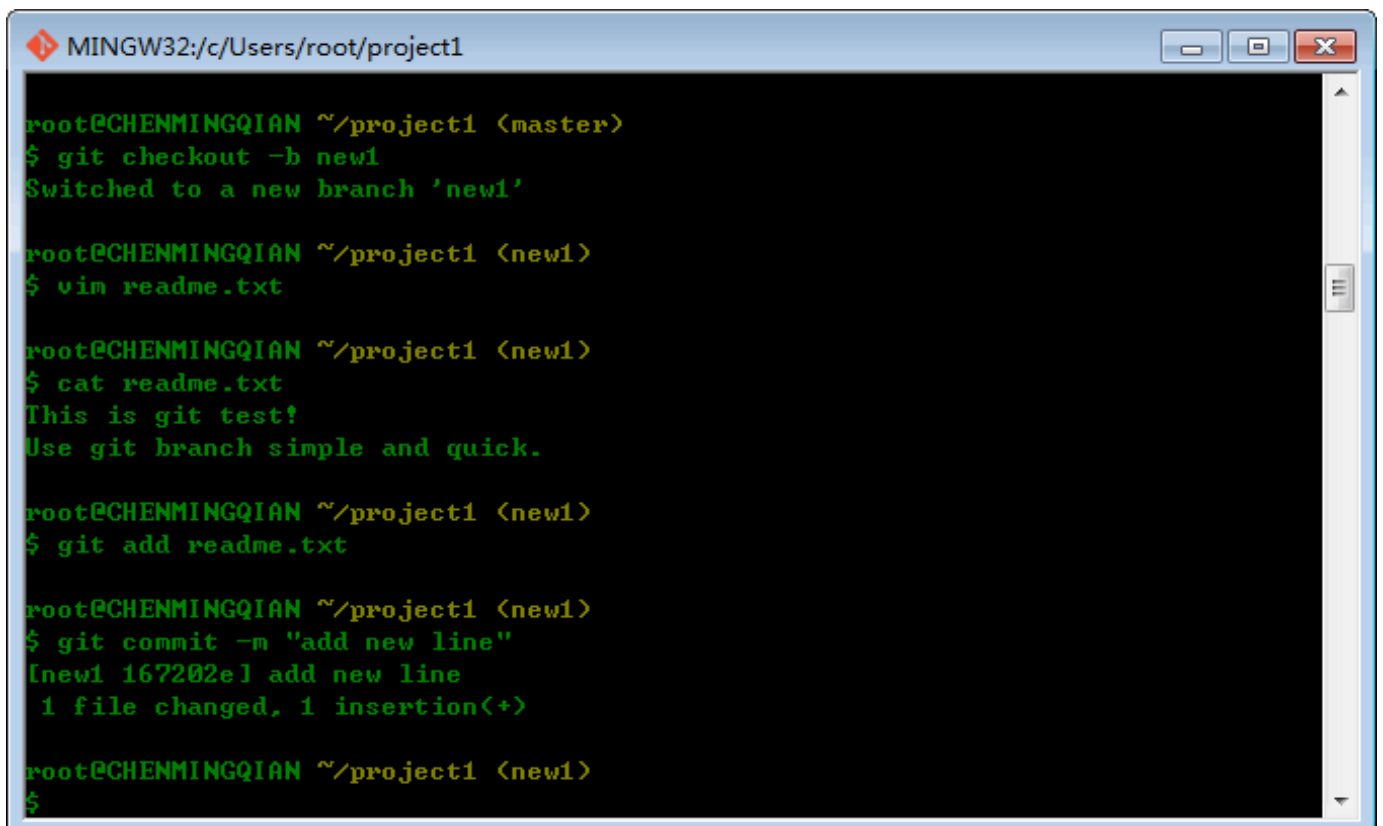
root@CHENMINGQIAN ~/project1 <master>
$
```

大家在实际操作中可以发现在我们创建、合并、删除分支的速度非常快吧，这和直接在master分支上工作效果是一样的，但过程更安全，更可靠。下面我们来简单的总结一下：

- 查看分支 `git branch -a`
- 创建分支 `git branch name`
- 切换分支 `git checkout name`
- 创建并切换 `git checkout -b name`
- 合并某分支到当前分支 `git merge name`
- 删除分支 `git branch -d name`

6. 分支合并冲突

下面我们来演示一下不同分支修改同一个文件产生冲突问题，下面我们来创建一个新的分支，继续开发新功能：

A terminal window titled 'MINGW32:/c/Users/root/project1' showing the process of creating a new branch and making a commit. The user switches to a new branch 'new1' using 'git checkout -b new1'. They then use 'vim readme.txt' to edit a file, view its contents with 'cat readme.txt' (showing 'This is git test!' and 'Use git branch simple and quick.'), and add it to the index with 'git add readme.txt'. Finally, they commit the changes with 'git commit -m "add new line"', which results in a commit hash '167202e1' and a message 'add new line'. The output indicates '1 file changed, 1 insertion(+)'.

```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <master>
$ git checkout -b new1
Switched to a new branch 'new1'

root@CHENMINGQIAN ~/project1 <new1>
$ vim readme.txt

root@CHENMINGQIAN ~/project1 <new1>
$ cat readme.txt
This is git test!
Use git branch simple and quick.

root@CHENMINGQIAN ~/project1 <new1>
$ git add readme.txt

root@CHENMINGQIAN ~/project1 <new1>
$ git commit -m "add new line"
[new1 167202e1] add new line
1 file changed, 1 insertion(+)

root@CHENMINGQIAN ~/project1 <new1>
$
```

现在我们将分支切换到master分支上修改readme.txt内容并提交，如下图：

```
MINGW32:/c/Users/root/project1
root@CHENMINGQIAN ~/project1 <new1>
$ git checkout master
Switched to branch 'master'
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

root@CHENMINGQIAN ~/project1 <master>
$ vim readme.txt

root@CHENMINGQIAN ~/project1 <master>
$ cat readme.txt
This is git test!
Use git branch simple or quick.

root@CHENMINGQIAN ~/project1 <master>
$ git add readme.txt

root@CHENMINGQIAN ~/project1 <master>
$ git commit -m "add new line"
[master 9243e00] add new line
 1 file changed, 1 insertion(+)

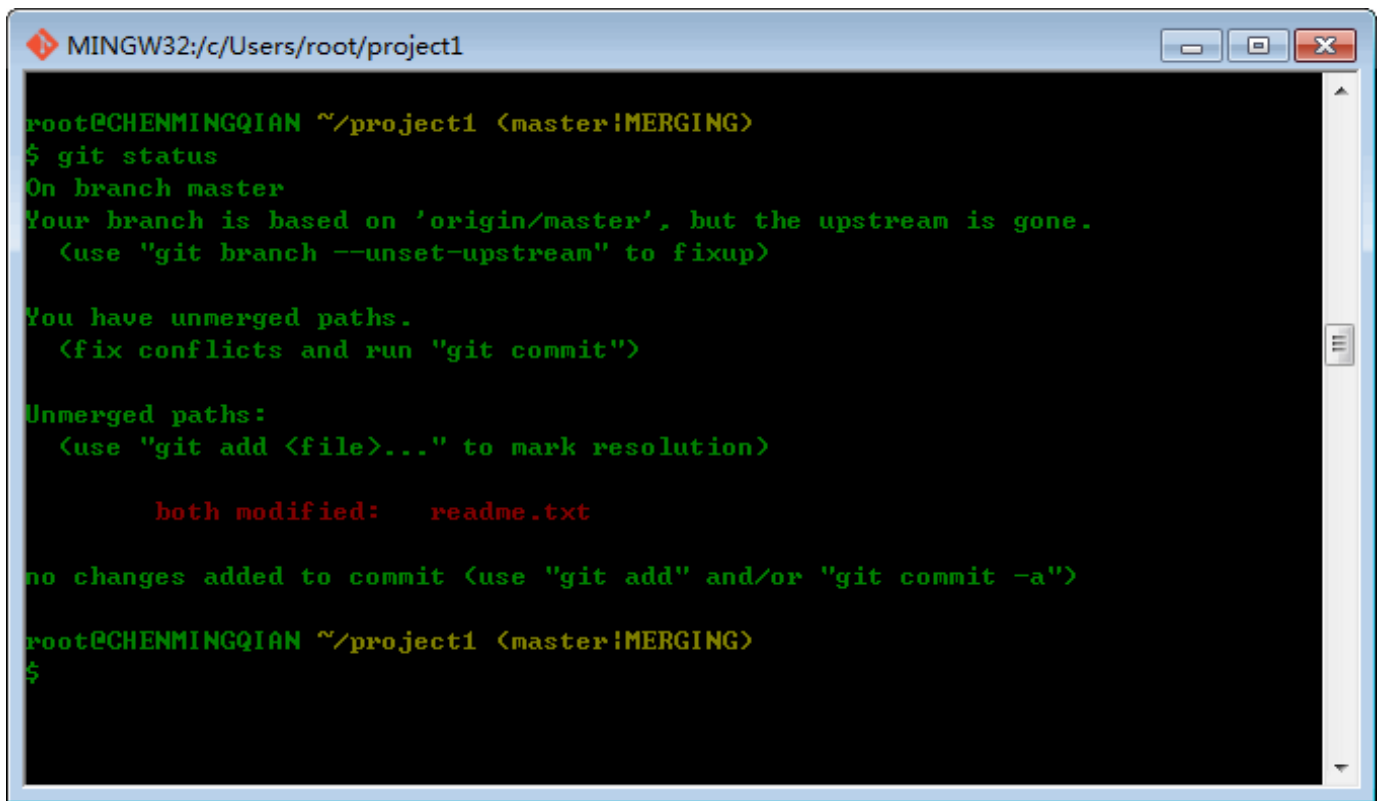
root@CHENMINGQIAN ~/project1 <master>
$
```

这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，如下图：

```
MINGW32:/c/Users/root/project1
root@CHENMINGQIAN ~/project1 <master>
$ git merge new1
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.

root@CHENMINGQIAN ~/project1 <master!MERGING>
$
```

果然冲突了，Git告诉我们readme.txt文件存在冲突，必须手动解决冲突后再提交。下面我们用git status查看一下状态：

A terminal window titled 'MINGW32:/c/Users/root/project1' showing the output of the 'git status' command. The output indicates the user is on the 'master' branch in a merge state, with unmerged paths for 'readme.txt'.

```
root@CHENMINGQIAN ~/project1 <master!MERGING>
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

You have unmerged paths.
  (fix conflicts and run "git commit")

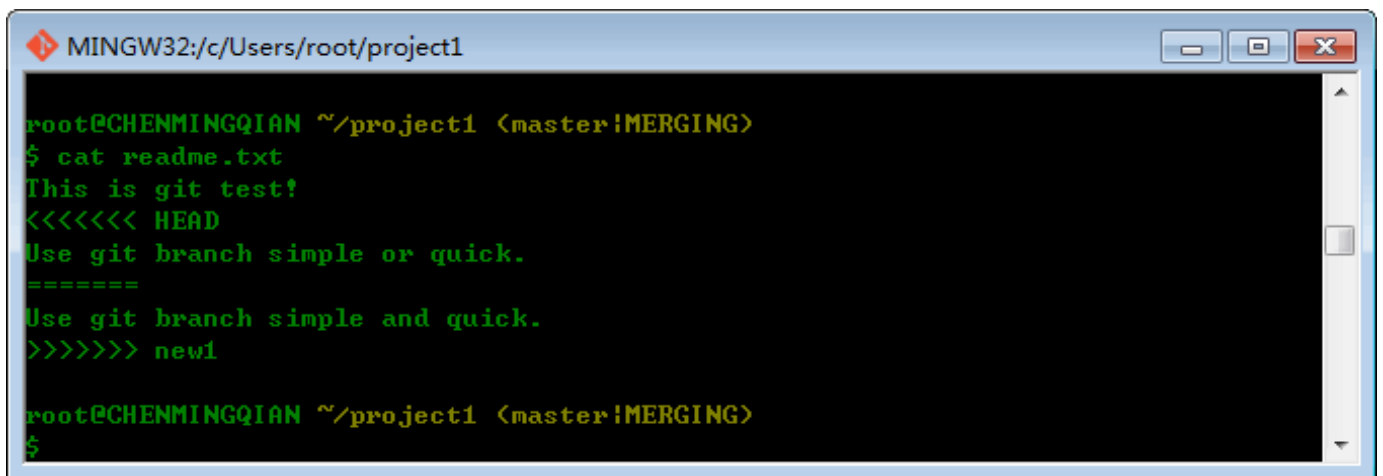
Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")

root@CHENMINGQIAN ~/project1 <master!MERGING>
$
```

下面我们来查看一下readme.txt中的内容，如下图：

A terminal window titled 'MINGW32:/c/Users/root/project1' showing the output of the 'cat readme.txt' command. The output displays text with conflict markers: '<<<<<<< HEAD' and '>>>>>>> new1'.

```
root@CHENMINGQIAN ~/project1 <master!MERGING>
$ cat readme.txt
This is git test!
<<<<<<< HEAD
Use git branch simple or quick.
=====
Use git branch simple and quick.
>>>>>>> new1

root@CHENMINGQIAN ~/project1 <master!MERGING>
$
```

Git用<<<<<<<, =====, >>>>>>>标记出不同分支的内容，让我们选择要保留的内容，下面我们修改一下readme.txt，再次提交。如下图：

```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <master!MERGING>
$ vim readme.txt

root@CHENMINGQIAN ~/project1 <master!MERGING>
$ git add readme.txt

root@CHENMINGQIAN ~/project1 <master!MERGING>
$ git commit -m "fixed"
[master 3cba4ce] fixed

root@CHENMINGQIAN ~/project1 <master>
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

nothing to commit, working directory clean

root@CHENMINGQIAN ~/project1 <master>
$ cat readme.txt
This is git test!
Use git branch simple and quick.

root@CHENMINGQIAN ~/project1 <master>
$
```

好了，到这我们的分支合并冲突就讲解完成了，下面我们来删除分支。如下图：

```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <master>
$ git branch -d new1
Deleted branch new1 (was 167202e).

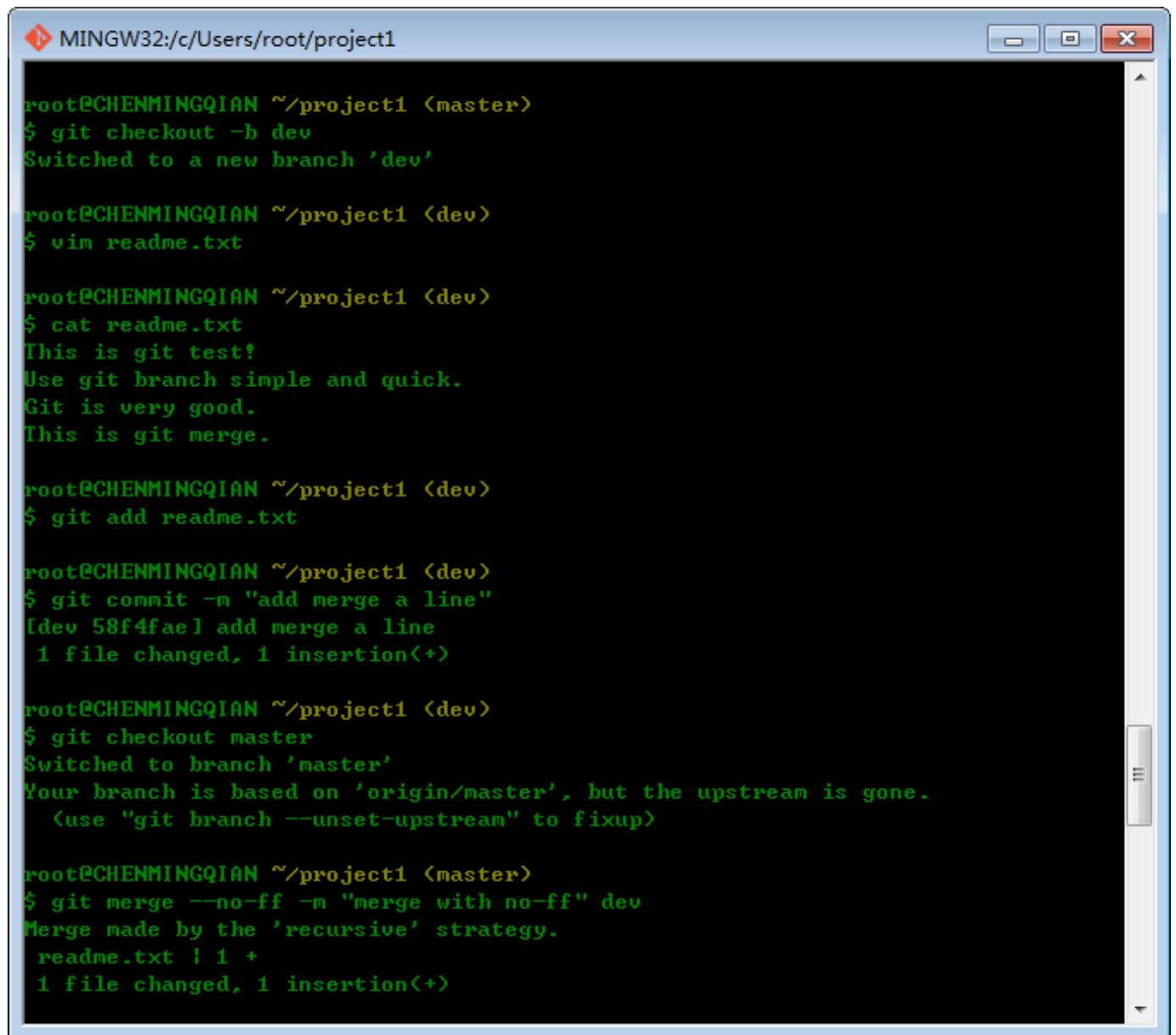
root@CHENMINGQIAN ~/project1 <master>
$ git log --graph --pretty=oneline --abbrev-commit
* 3cba4ce fixed
! \
! * 167202e add new line
* ! 9243e00 add new line
! /
* 63ab81c add a line
* 8dba382 add readme.txt

root@CHENMINGQIAN ~/project1 <master>
$
```

最后，我们可以用 `git log --graph --pretty=oneline --abbrev-commit` 命令，查看一下分支合并。

7. 合并分支（普通合并）

分支合并分为快速合并与普通合并两种模式，普通合并，合并后的历史有分支记录，能看出来曾经做过合并，而快速合并就看不出来曾经做过合并。下面我们来演示一下普通合并，



```
root@CHENMINGQIAN ~/project1 <master>
$ git checkout -b dev
Switched to a new branch 'dev'

root@CHENMINGQIAN ~/project1 <dev>
$ vim readme.txt

root@CHENMINGQIAN ~/project1 <dev>
$ cat readme.txt
This is git test!
Use git branch simple and quick.
Git is very good.
This is git merge.

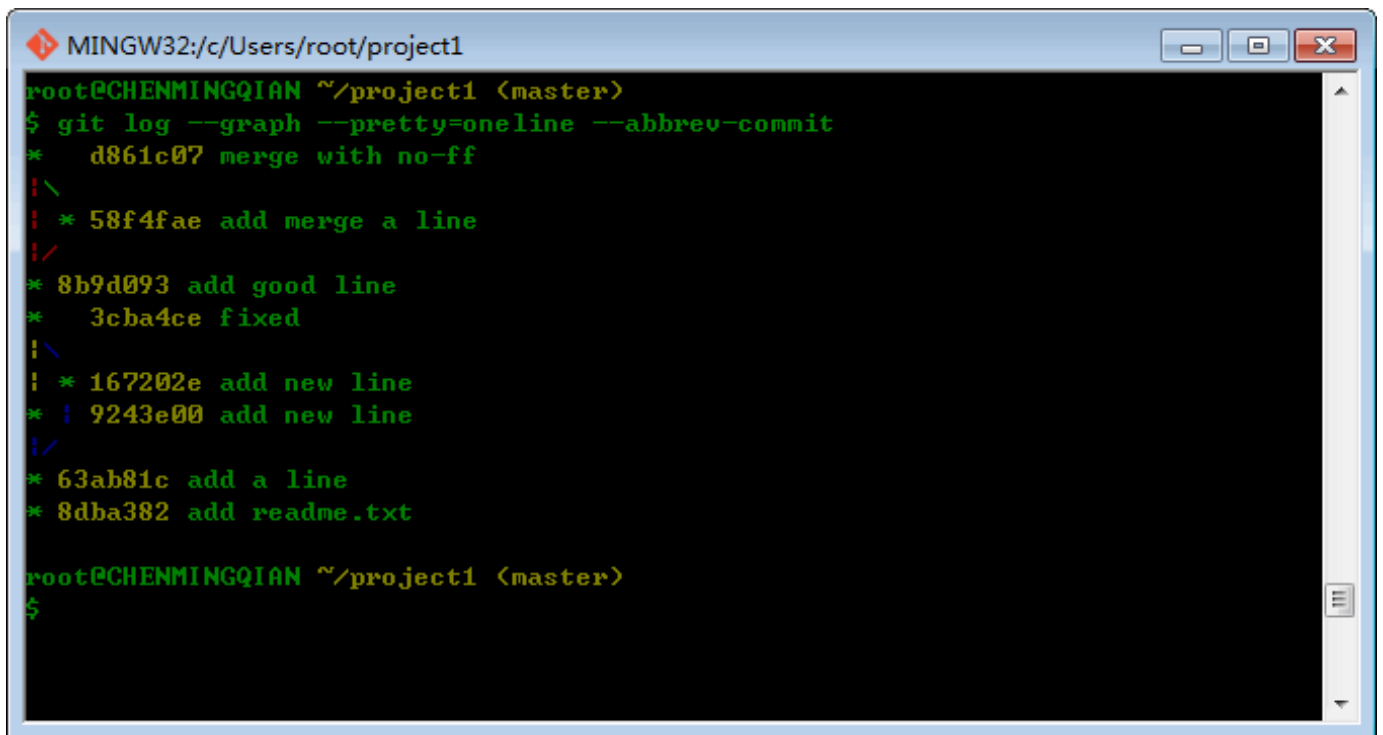
root@CHENMINGQIAN ~/project1 <dev>
$ git add readme.txt

root@CHENMINGQIAN ~/project1 <dev>
$ git commit -m "add merge a line"
[dev 58f4fae] add merge a line
1 file changed, 1 insertion(+)

root@CHENMINGQIAN ~/project1 <dev>
$ git checkout master
Switched to branch 'master'
Your branch is based on 'origin/master', but the upstream is gone.
(use "git branch --unset-upstream" to fixup)

root@CHENMINGQIAN ~/project1 <master>
$ git merge --no-ff -m "merge with no-ff" dev
Merge made by the 'recursive' strategy.
readme.txt | 1 +
1 file changed, 1 insertion(+)
```

大家可以看到我们这次合并用的普通模式合并，--no-ff参数表示禁用快速合并。下面我们用git log命令查看一下合并历史：



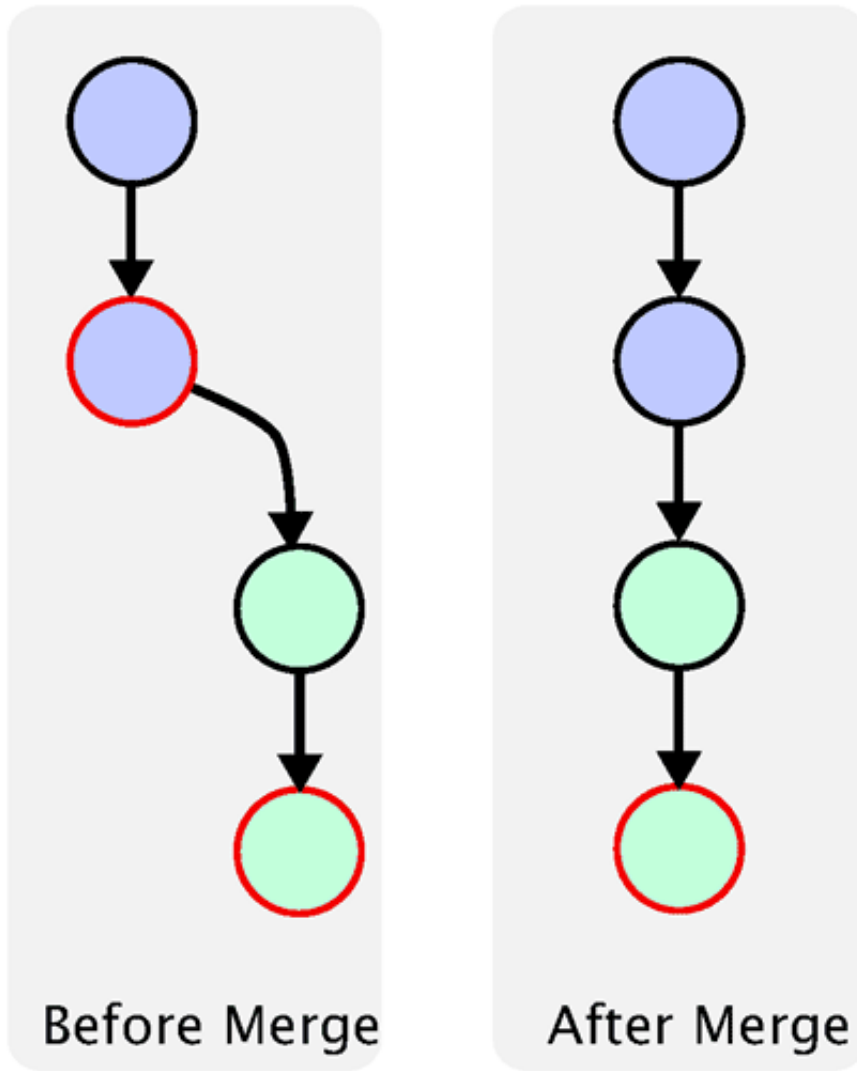
```
MINGW32:/c/Users/root/project1
root@CHENMINGQIAN ~/project1 <master>
$ git log --graph --pretty=oneline --abbrev-commit
* d861c07 merge with no-ff
|\
| * 58f4fae add merge a line
|/
* 8b9d093 add good line
* 3cba4ce fixed
|\
| * 167202e add new line
* | 9243e00 add new line
|/
* 63ab81c add a line
* 8dba382 add readme.txt

root@CHENMINGQIAN ~/project1 <master>
$
```

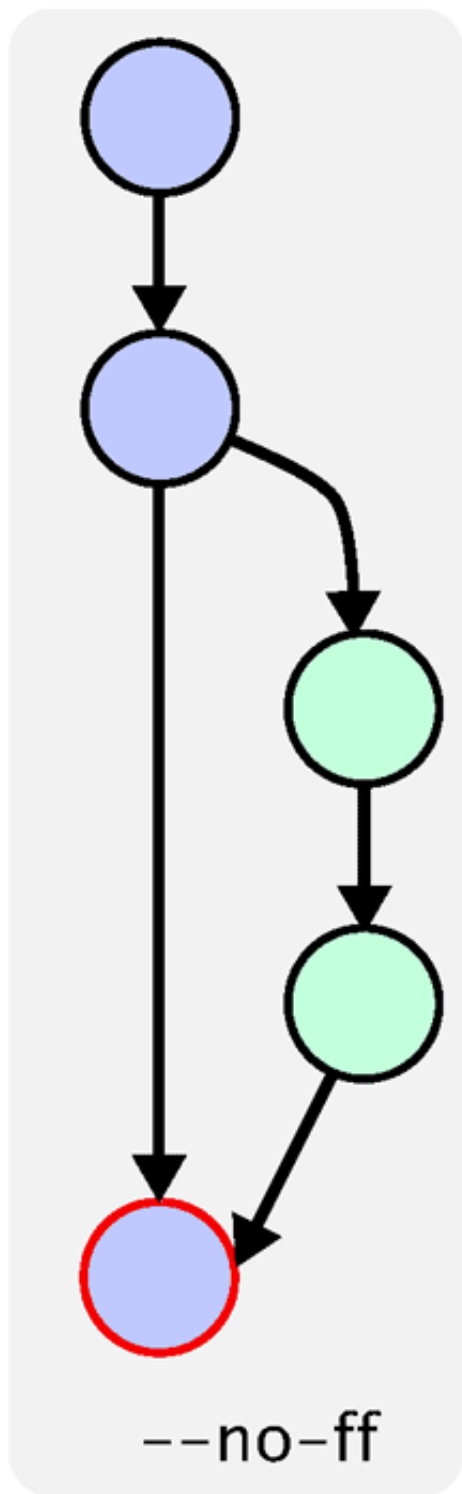
注，合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支记录，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。如下图：

1). 快速合并

Fast Forward Merge



2). 普通合并



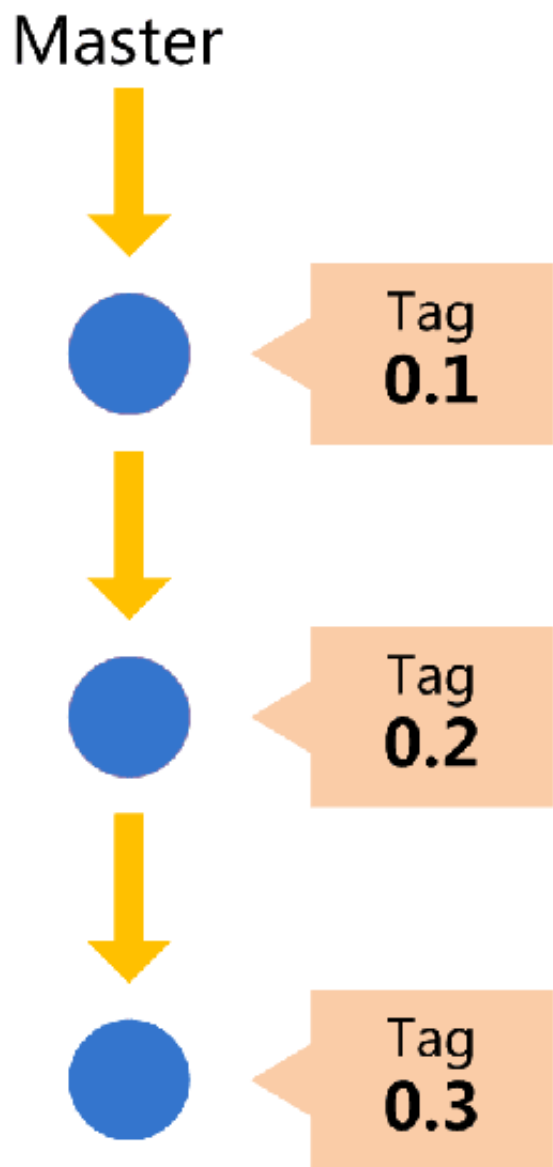
8. 分支管理策略

下面我们来说一下一般企业中开发一个项目的分支策略：

- 主分支 master
- 开发分支 develop
- 功能分支 feature
- 预发布分支 release
- bug 分支 fixbug
- 其它分支 other

1). 主分支 master

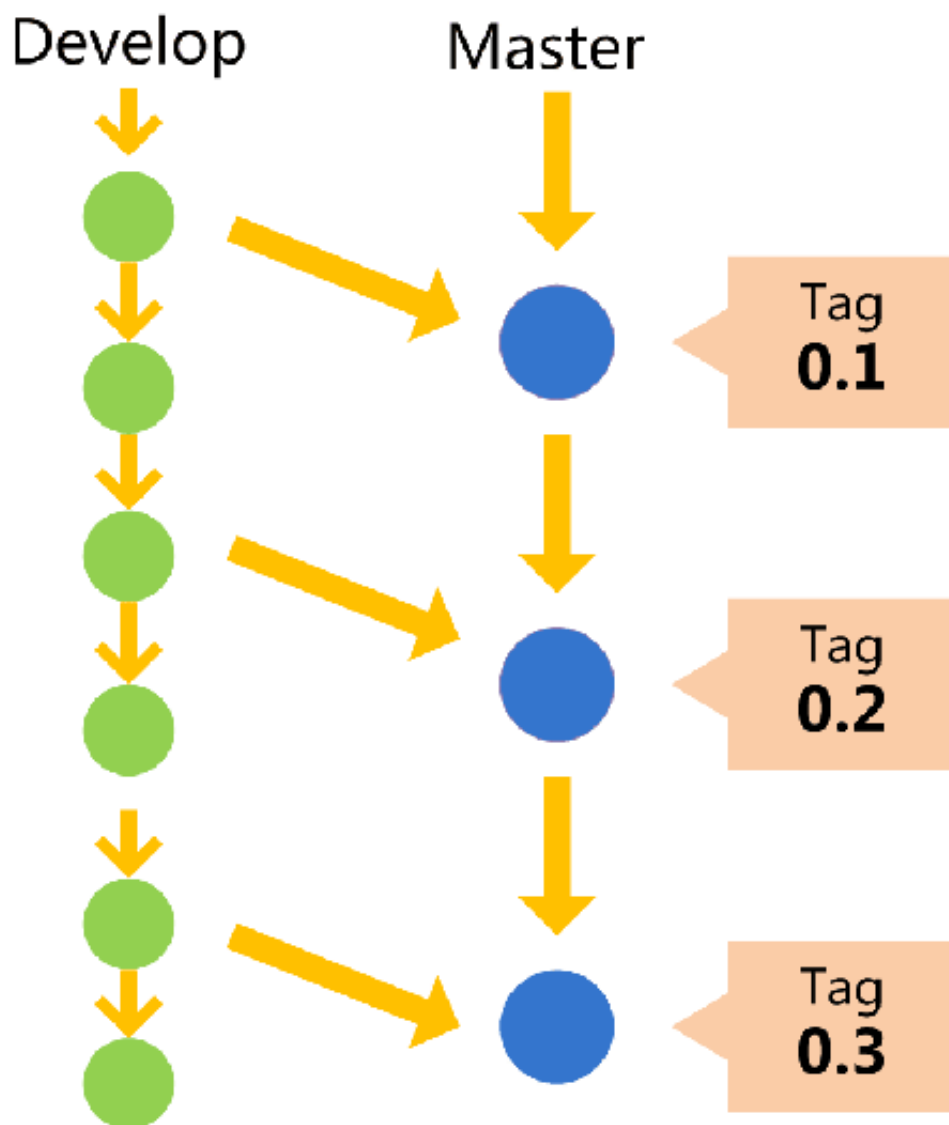
代码库应该有一个、且仅有一个主分支。所有提供给用户使用的正式版本，都在这个主分支上发布。



Git主分支的名字，默认叫做Master。它是自动建立的，版本库初始化以后，默认就是在主分支在进行开发。

2). 开发分支 develop

主分支只用来分布重大版本，日常开发应该在另一条分支上完成。我们把开发用的分支，叫做Develop。

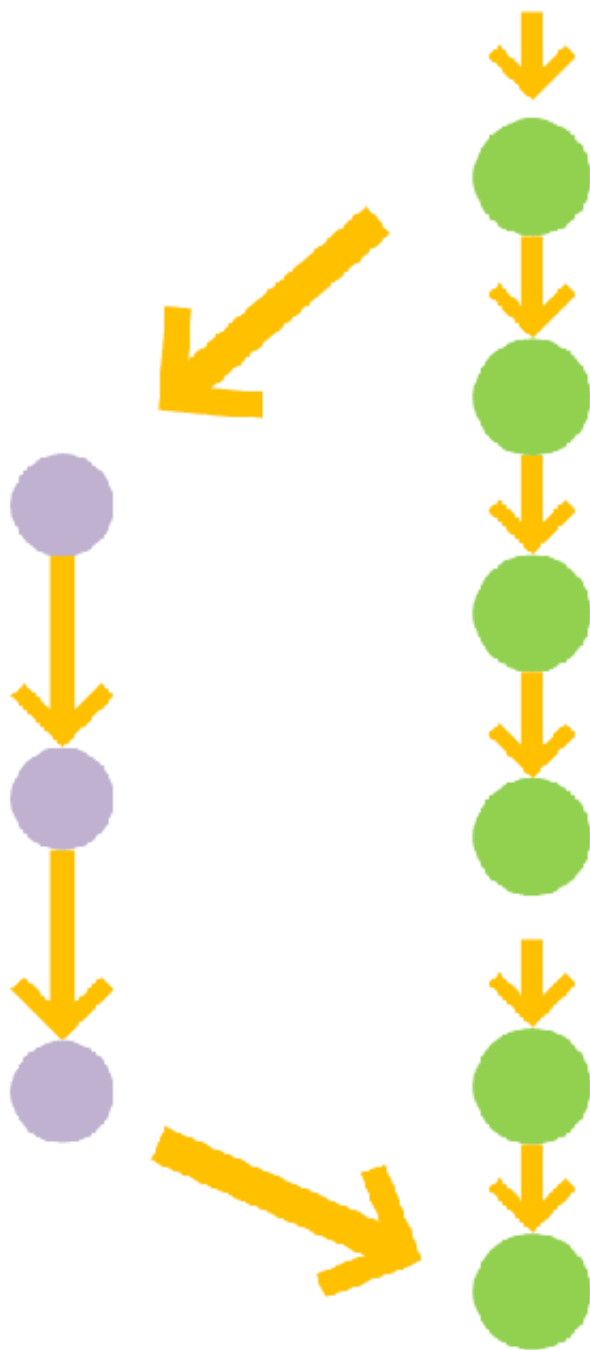


这个分支可以用来生成代码的最新代码版本。如果想正式对外发布，就在Master分支上，对Develop分支进行“合并”（merge）。

3). 功能分支 feature

功能分支，它是为了开发某种特定功能，从Develop分支上面分出来的。开发完成后，要再并入Develop。

Feature Develop



功能分支的名字，可以采用feature-*的形式命名。

4). 预发布分支 release

预发布分支，它是指发布正式版本之前（即合并到Master分支之前），我们可能需要有一个预发布的版本进行测试。预发布分支是从Develop分支上面分出来的，预发布结束以后，必须合并进Develop和Master分支。它的命名，可以采用release-*的形式。

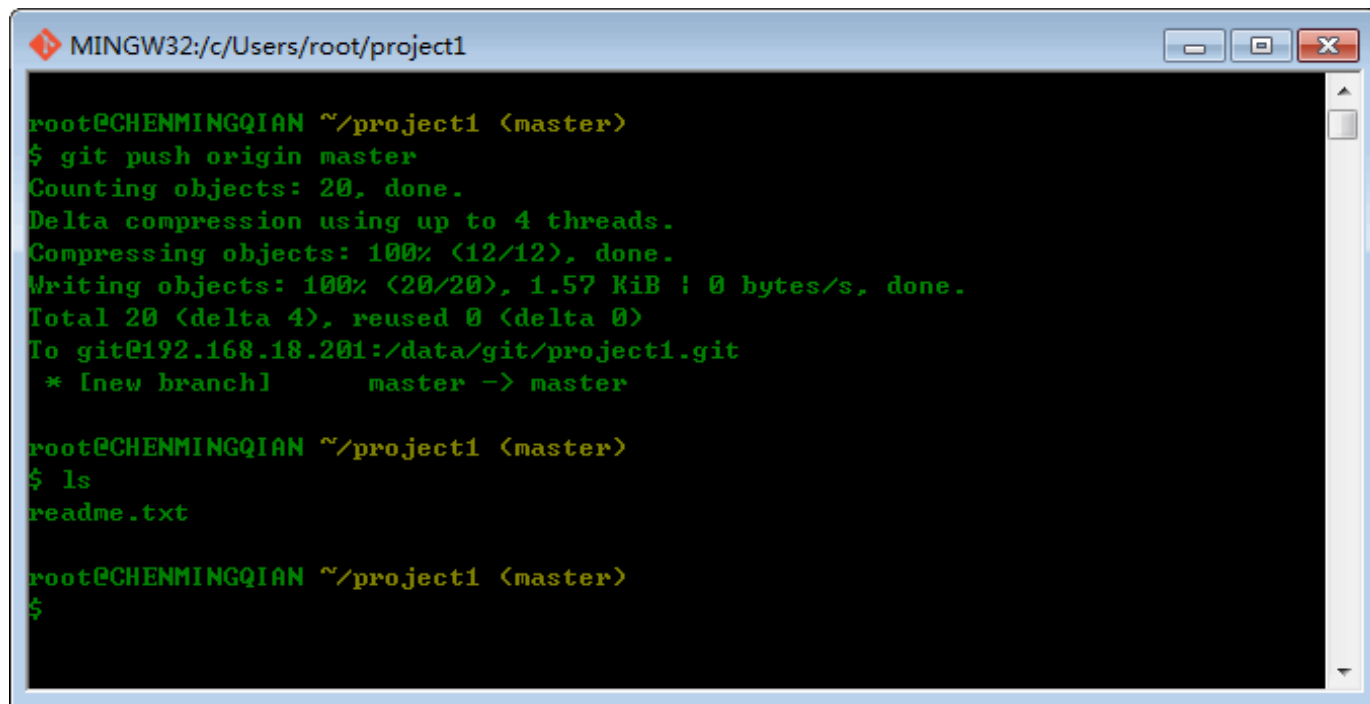
5). bug 分支 fixbug

bug分支。软件正式发布以后，难免会出现bug。这时就需要创建一个分支，进行bug修补。修补bug分支是从Master分支上面分出来的。修补结束以后，再合并进Master和Develop分支。它的命名，可以采用fixbug-*的形式。

大家可以看到git remote命令可以查看远程仓库，加-v选项可以查看详细信息。上面显示了你抓取和推送的origin（源）信息。当你从远程仓库克隆时，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且远程仓库的默认名称是origin。

2). 分支推送

下面我们来演示一下将本地分支推送到远程的仓库中，如下图：



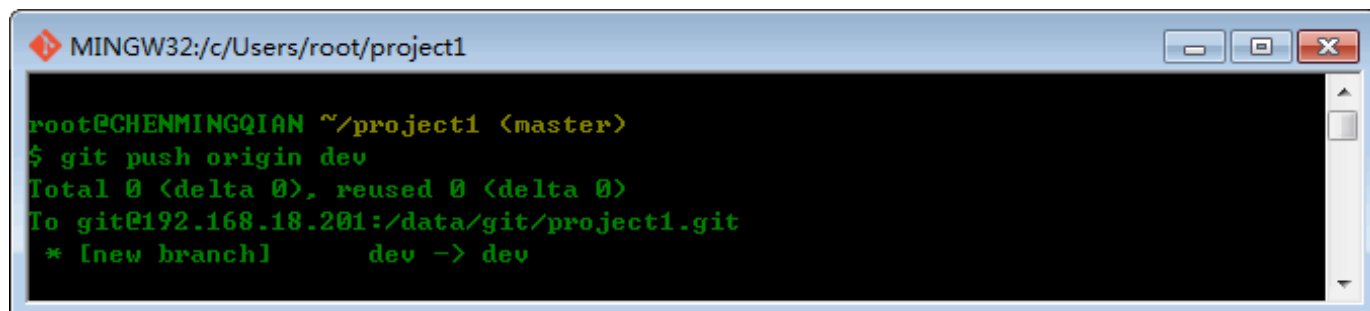
```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <master>
$ git push origin master
Counting objects: 20, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (20/20), 1.57 KiB | 0 bytes/s, done.
Total 20 (delta 4), reused 0 (delta 0)
To git@192.168.18.201:/data/git/project1.git
 * [new branch]      master -> master

root@CHENMINGQIAN ~/project1 <master>
$ ls
readme.txt

root@CHENMINGQIAN ~/project1 <master>
$
```

大家可以看到，我们的本地master分支与远程的master分支已同步。下面我们来演示一下同步dev分支，如下图：



```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <master>
$ git push origin dev
Total 0 (delta 0), reused 0 (delta 0)
To git@192.168.18.201:/data/git/project1.git
 * [new branch]      dev -> dev
```

有博友会问了，我本地有很多分支，有哪些分支需要推送到远程仓库中呢？一般是这样的，

- master 分支是主分支，要时刻与远程同步，一般我们发布最新版本就用master分支
- develop 分支是开发分支，团队中所有人都在这个分支上开发，所以也需要与远程同步
- bug 分支一般只在本地使用来修复bug，一般不需推送远程仓库中
- feature 分支是否需要推送到远程，要看是不是有几个人合作开发新功能，如果你是一个开发，那就留在本地吧
- release 分支一般是系统管理，推送或抓取的分支一般与开发人员无关
- other 分支大家按需求分配

3). 分支抓取

生成公钥:

将生成的公司增加到git服务器上:

克隆远程版本库：

我们现在查看一下分支：


```
1 [root@localhost project1]# git branch
2
3 * master
```

你同事要在dev分支上开发，就得创建与远程origin一样的dev分支到本地的仓库上，下面我们来创建一下：

```
1 [root@localhost project1]# git checkout -b dev origin/dev
2 Branch dev set up to track remote branch dev from origin.
3 Switched to a new branch 'dev'
4 [root@localhost project1]# git branch * dev master
5 [root@localhost project1]#
```

现在你同事就可以在dev分支上开发了，下面我们新建一些文件并提交到远程dev分支：

```
1 [root@localhost project1]# touch index.html
2 [root@localhost project1]# vim index.html
3 This git test index.html!
4 [root@localhost project1]# git add index.html
5 [root@localhost project1]# git commit -m "add index.html"
6 [dev 6e47de0] add index.html
7 Committer: root <root@localhost.localdomain>
8 Your name and email address were configured automatically based on your username and hostname.P
9 You can suppress this message by setting them explicitly:
10 git config --global user.name "Your Name"
11 git config --global user.email you@example.com
12 If the identity used for this commit is wrong, you can fix it with:
13 git commit --amend --author='Your Name <you@example.com>'
14 1 files changed, 1 insertions(+), 0 deletions(-) create mode 100644 index.html
15 [root@localhost project1]# git config --global user.name "leo"
16 [root@localhost project1]# git config --global user.email "leo@jjhh.com"
17 [root@localhost project1]# git commit -m "add index.html"
18 # On branch dev # Your branch is ahead of 'origin/dev' by 1 commit.
19 # nothing to commit (working directory clean)
20 [root@localhost project1]# git status
21 # On branch dev # Your branch is ahead of 'origin/dev' by 1 commit.
22 # nothing to commit (working directory clean)
23 [root@localhost project1]# git push origin dev
24 Counting objects: 4, done. Delta compression using up to 4 threads.
25 Compressing objects: 100% (2/2), done. Writing objects: 100% (3/3), 305 bytes, done.
26 Total 3 (delta 0), reused 0 (delta 0) To git@192.168.18.201:/data/git/project1.git
27 58f4fae..6e47de0; dev -> dev
```

你的同事向origin/dev分支提交了一个index.html页面，现在你也在修改这个文件，并提交：

```
MINGW32:/c/Users/root/project1
root@CHENMINGQIAN ~/project1 <master>
$ git checkout dev
Switched to branch 'dev'

root@CHENMINGQIAN ~/project1 <dev>
$ vim index.html

root@CHENMINGQIAN ~/project1 <dev>
$ cat index.html
This is git test!

root@CHENMINGQIAN ~/project1 <dev>
$ git add index.html

root@CHENMINGQIAN ~/project1 <dev>
$ git commit -m "add index.html"
[dev 5ca5241] add index.html
1 file changed, 1 insertion(+)
create mode 100644 index.html

root@CHENMINGQIAN ~/project1 <dev>
$ git push origin dev
To git@192.168.18.201:/data/git/project1.git
! [rejected]        dev -> dev (fetch first)
error: failed to push some refs to 'git@192.168.18.201:/data/git/project1.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 integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

root@CHENMINGQIAN ~/project1 <dev>
$
```

推送失败，因为你同事的最新提交和你推送的提交有冲突，Git提示我们，先用git pull把最新的提交从origin/dev抓下来，然后在本地合并解决冲突，再推送：

```
MINGW32:/c/Users/root/project1
root@CHENMINGQIAN ~/project1 <dev>
$ git pull
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From 192.168.18.201:/data/git/project1
58f4fae..6e47de0 dev -> origin/dev
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details

    git pull <remote> <branch>

If you wish to set tracking information for this branch you can do so with:

    git branch --set-upstream-to=origin/<branch> dev

root@CHENMINGQIAN ~/project1 <dev>
$
```

git pull 失败了，原因是没有将dev分支与远程origin/dev分支进行链接，Git提示我们设置dev和origin/dev的链接：

```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <dev>
$ git branch --set-upstream dev origin/dev
The --set-upstream flag is deprecated and will be removed. Consider using --track or --set-upstream-to
Branch dev set up to track remote branch dev from origin.

root@CHENMINGQIAN ~/project1 <dev>
$
```

下面我们再来git pull一下试试：

```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <dev>
$ git pull
Auto-merging index.html
CONFLICT (add/add): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.

root@CHENMINGQIAN ~/project1 <dev !MERGING>
$
```

git pull 成功，但是合并有冲突需要手动解决，解决的方法和分支管理中的解决冲突完全一样。解决后提再push：

```
MINGW32:/c/Users/root/project1

root@CHENMINGQIAN ~/project1 <dev !MERGING>
$ vim index.html
```

```
index.html (~\project1) - VIM
<<<<<<< HEAD
This is git test!
=====
This git test index.html!
>>>>>> 6e47de0c42fecc2c94cb1a30f7c4aa8d9b39080b
~
~
~\project1\index.html [dos] <11:06 27/05/2014> 1,12 All
```

```
MINGW32:/c/Users/root/project1
root@CHENMINGQIAN ~/project1 <dev !MERGING>
$ cat index.html
This is git test!

root@CHENMINGQIAN ~/project1 <dev !MERGING>
$ git add index.html

root@CHENMINGQIAN ~/project1 <dev !MERGING>
$ git commit -m "fixed index.html"
[dev 677f8e9] fixed index.html

root@CHENMINGQIAN ~/project1 <dev>
$ git push origin dev
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 420 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To git@192.168.18.201:/data/git/project1.git
6e47de0..677f8e9 dev -> dev

root@CHENMINGQIAN ~/project1 <dev>
$ git status
On branch dev
Your branch is up-to-date with 'origin/dev'.

nothing to commit, working directory clean

root@CHENMINGQIAN ~/project1 <dev>
$
```

好了，这样的我们的远程推送与抓取就讲解完成了，下面我们来总结一下。

10. 总结

一般在团队中多人开发模式是这样的：首先，可以试图用 `git push origin branch-name` 推送自己的修改如果推送失败，则因为远程分支比你的本地更新，需要先用 `git pull` 试图合并如果合并有冲突，则解决冲突，并在本地提交没有冲突或者解决掉冲突后，再用 `git push origin branch-name` 推送就能成功如果 `git pull` 提示 “no tracking information”，则说明本地分支和远程分支的链接关系没有创建，用命令 `git branch --set-upstream branch-name origin/branch-name`。

好了，到这里我们的Git分支管理就讲解到这里了。最后，希望大家有所收获^_^……

本文出自 “Share your knowledge …” 博客，<http://freeloda.blog.51cto.com/2033581/1417525>



[blindcat](#)

发帖于 2年前

[8](#)回/7920阅