

微信：



微博：



全国统一咨询电话：400-0088-518

1、课程名称：分支管理



2、知识点

- 1、 GIT 与传统的开发最大的区别在于分支管理上，而只有处理好了分支才可以说掌握了 GIT 的使用；
- 2、 有了分支管理之后，许多的开发就可以由用户自己去定义模式；
- 3、 如何创建、使用分支、冲突的解决、BUG 管理、补丁操作等；

3、具体内容

3.1、创建与合并分支

只要是进行过项目的开发实际上都会遇见这样一种情况（年轻人）。一方面是客户方代码的催促，另外一方面是作为一个程序开发人员自己的创造力的矛盾。

很多的时候做开发工作中都希望给用户最好的体验，可以使用出更多的优秀功能，但是如果一直所有的精力都关注在优秀功能上，而忘记了代码的整合，那么在进行项目的中间验收时就可能出现交不上代码的情况。

微信：

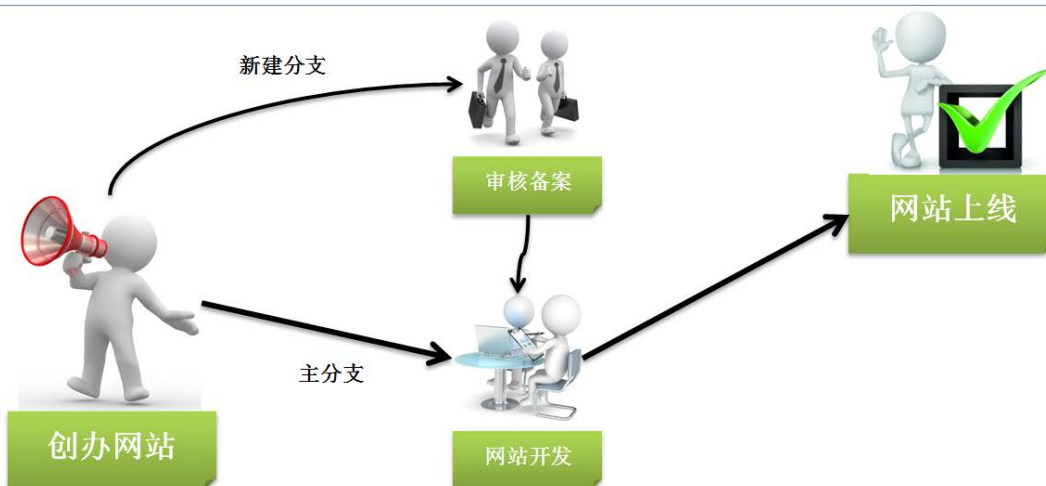


微博：



全国统一咨询电话：400-0088-518

分支

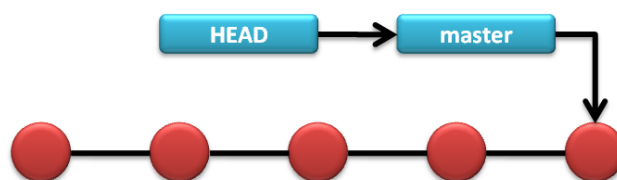


优拓教育 (www.yootk.com) & 魔乐科技 (www.mldnjava.cn) 联合出品

利用分支就可以实现多人开发的伟大模式，从而提高生产效率。在整个 GIT 之中，主分支（master）主要是作为程序的发布使用，一般而言很少会在主分支上进行代码的开发，都会在各自己的子分支上进行。

master分支

- 默认情况下，master是一条线，GIT利用master指向最新的提交，再用“HEAD”指向“master”，就能确定当前分支以及当前分支的提交点。



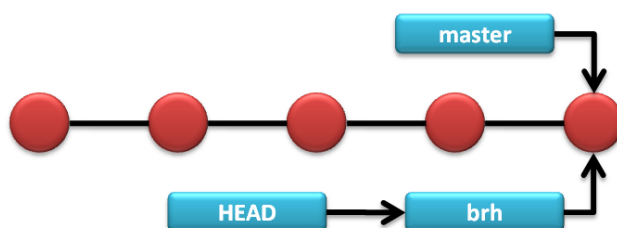
优拓教育 (www.yootk.com) & 魔乐科技 (www.mldnjava.cn) 联合出品

以上的操作属于项目的发布版本的执行顺序，因为最终发布的就是 master 分支。但是对于其他的开发者，不应该在 master 分支上进行。所以应该建立子分支，而子分支最起码建立的时候应该是当前的 master 分支的状态。而子分支一旦创建之后，HEAD 指针就会发生变化。



用分支操作

- 而当用户创建了一个新的分支（假设新的分支为“brh”）后，GIT会建立一个新的brh指针，此指针将指向master相同的提交，再将HEAD指向brh，就表示当前分支在brh上。

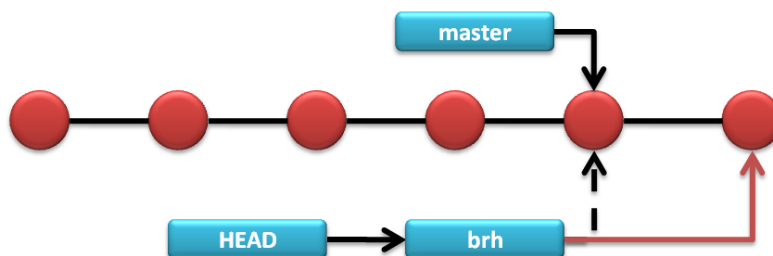


优拓教育 (www.yootk.com) & 魔乐科技 (www.mldnjava.cn) 联合出品

在整个过程之中，发现 HEAD 的指针发生了改变，因为 HEAD 永远都要指向当前的分支（当前工作的分支）。一旦创建了分支，那么一定需要针对于代码进行新的修改（再次进行了提交）。

分支提交

- 如果有新的提交，则master分支不会改变，只有brh分支会发生改变。



优拓教育 (www.yootk.com) & 魔乐科技 (www.mldnjava.cn) 联合出品

那么此时 master 分支的版本号就落后于子分支了。但是不管子分支再怎么开发，也不是最新的发布版本，所有的发布版本都保存在 master 分支上，那么就必须将子分支与 master 的分支进行合并。

微信：



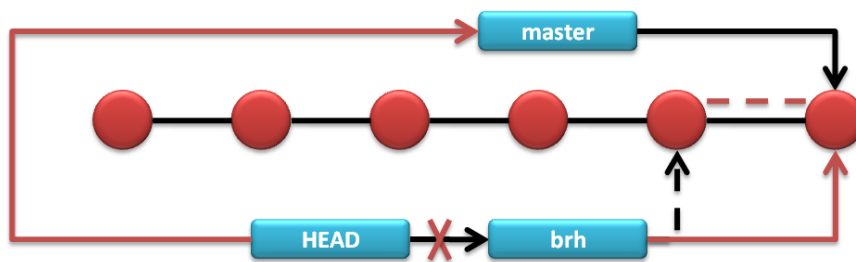
微博：



全国统一咨询电话：400-0088-518

合并分支

- 如果有新的提交，则master分支不会改变，只有brh分支会发生改变。

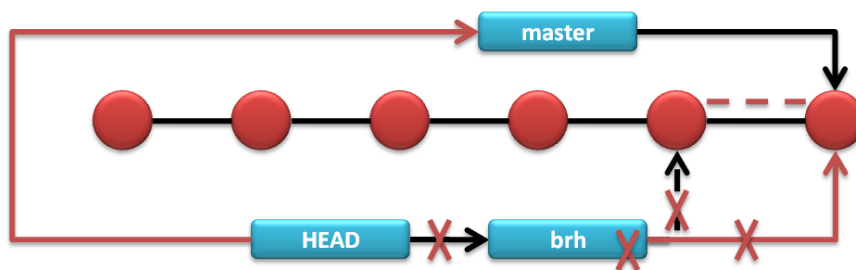


优拓教育 (www.yootk.com) & 魔乐科技 (www.mldnjava.cn) 联合出品

当分支合并之后，实际上就相当于 master 的分支的提交点修改为了子分支的提交点，而后这个合并应该在 master 分支上完成，而后 HEAD 需要修改指针，断开 brh 分支，而指向原本的 master 分支。

删除子分支

- 如果有新的提交，则master分支不会改变，只有brh分支会发生改变。



优拓教育 (www.yootk.com) & 魔乐科技 (www.mldnjava.cn) 联合出品

分支删除掉之后所有的内容也就都取消了。

如果你现在还没有能够理解分支的操作流程，那么请一定要将以上的视频重新学习一次。而后面的 GIT 开发过程与之是完全一样的实现。

1、 创建一个分支

优拓教育 (www.yootk.com) & 魔乐科技软件学院 (www.mldn.cn) 联合出品

第 (4) 页 共 (27) 页

微信：



微博：

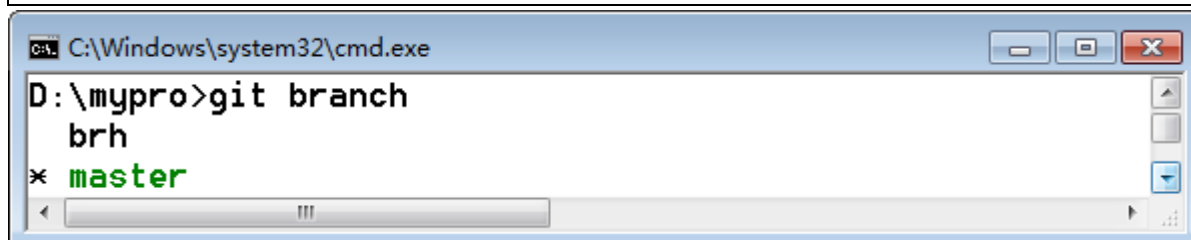


全国统一咨询电话：400-0088-518

```
git branch brh
```

2、 当分支创建完成之后可以通过如下的命令进行察看：

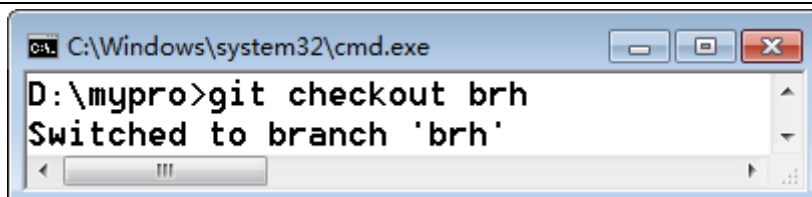
```
git branch
```



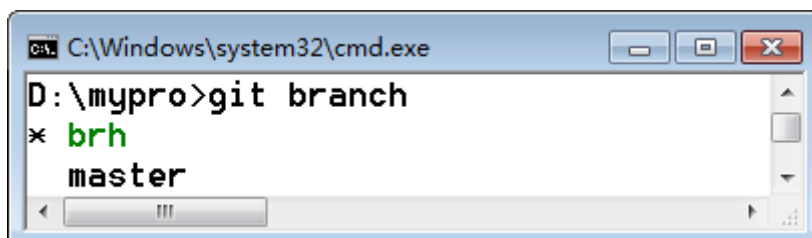
可以发现现在提示当前的工作区之中有两个分支：一个是 brh 分支，另外一个 master 分支，而现在的分支指向的是 master 分支。

3、 切换到 brh 分支：

```
git checkout brh
```



而后再次查询分支的信息：



但是很多时候我们创建分支的最终目的就是为了切换到此分支上进行开发，所以为了方便操作，在 git 之中提供了一个更加简单的功能。

范例：创建并切换分支

如果要想删除子分支，那么不能够在当前分支上，所以切换回了 master 分支	git checkout master
删除子分支（-d 表示删除）	git branch -d brh
建立分支的同时可以自动的切换到子分支	git checkout -b brh

4、 现在已经成功的在 brh 分支上了，那么下面进行代码的修改：

范例：修改 Hello.java 类

```
public class Hello {  
    public static void main(String args[]) {  
        System.out.println("Happy New Year");  
        System.out.println("Happy MySelf");  
    }  
}
```

这个时候的 Hello.java 文件是属于子分支上的，而现在也在子分支上，那么下面查询一下子分支的状态。

微信：



微博：



全国统一咨询电话：400-0088-518

```
C:\Windows\system32\cmd.exe
D:\mypro>git status
On branch brh
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:   Hello.java

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

此时更新的是子分支的内容，但是主分支上的数据呢？

5、 在子分支上将修改进行提交：

```
git commit -a -m "Modified Hello.java File"
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git commit -a -m "Modified Hello.java File"
[brh d42f7c6] Modified Hello.java File
1 file changed, 2 insertions(+), 1 deletion(-)
```

当子分支的数据提交之后实际上并不会去修改 master 分支的内容。这就证明了，两个分支上的内容是彼此独立的。

6、 那么既然分支都已经存在了，那么现在为了更加清楚，将 master 和 brh 两个分支都提交到远程服务器上（GITHUB）。

```
git remote set-url origin https://github.com/yootk/mldn.git
git push origin master
git push origin brh
```

7、 最终发布的版本一定是在 master 分支上，所以下面需要将 brh 分支与 master 分支进行合并（在主分支上）：

```
git merge brh
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git merge brh
Updating 2e1743c..d42f7c6
Fast-forward
 Hello.java | 3 ++-
1 file changed, 2 insertions(+), 1 deletion(-)
```

在之前讲解原理的时候说过实际上是修改了 master 指针为 brh 分支的指针信息。所以此时的合并方式为“Fast-forward”，表示是快速合并方式，快速的合并方式并不会产生任何的 commit id。它只是利用了合并子分支的 commit id 继续操作。

8、 此时的 brh 分支没有任何的用处了，那么就可以执行删除操作

```
git branch -d brh
```

9、 提交 master 分支

```
git push origin master
```

微信：



微博：



全国统一咨询电话：400-0088-518

现在在本地上已经没有了子分支，但是在远程服务器上依然会存在子分支。那么下面要删除远程分支。

10、删除远程分支

```
git push origin --delete brh
```

那么此时远程分支就已经被成功的删除掉了。

3.2、分支的操作管理

在之前已经完整的演示了分支的各个操作，包括使用分支、以及合并分支，同时也清楚了对于分支有两种方式一种是本地分支，另外一种远程分支，但是对于分支在 GIT 使用之中依然会有一些小小的问题，所以下面进行集中式的说明。

1、为了方便还是建立一个新的分支 —— brh

```
git checkout -b brh
```

2、在此分支上建立一些文件；

```
import java.io.*;

public class Emp implements Serializable {

    private Integer empno ;

    private String ename ;

    private String job ;

}
```

以上的代码是在子分支（brh）上建立的。

```
git add .
```

```
git commit -a -m "Add Emp.java File"
```

3、此时并没有进行分支数据的提交，但是有人觉得这个 brh 分支名称不好，应该使用自己的姓名简写完成“lxh”。

```
git branch -m brh lxh
```

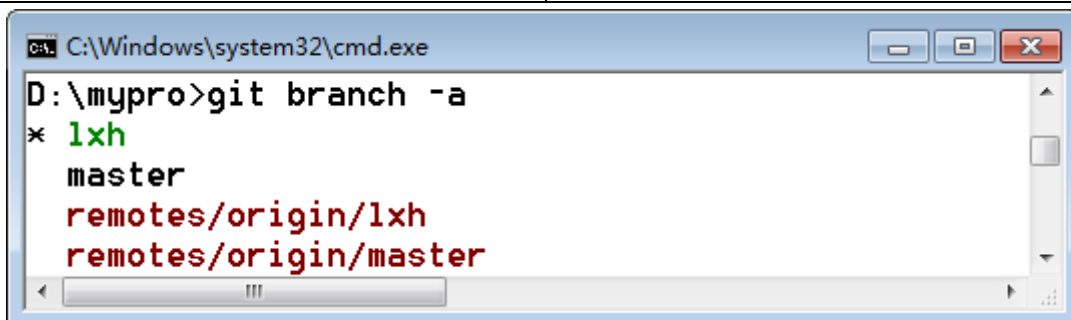
现在相当于分支名称进行了重新的命名；

4、将分支推送到远程服务器端；

```
git push origin lxh
```

5、在本地察看远程的分支；

察看全部的分支，包括远程和本地的分支	git branch -a
只察看远程的分支	git branch -r
只察看本地的分支	git branch -l



6、此时“lxh”分支上已经做出了修改，但是并没有与 master 分支进行合并，因为现在所开发的功能开发到一半发现不再需要了，所以就要废除掉所作出的修改。于是发出了删除 lxh 分支的命令。

微信：



微博：



全国统一咨询电话：400-0088-518

```
git branch -d lxh
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git branch -d lxh
error: The branch 'lxh' is not fully merged.
If you are sure you want to delete it, run 'git branch -D lxh'.
```

此时直接提示，分支并不能够被删除掉，因为这个分支所做出的修改还没有进行合并。如果要想强制删除此分支，则可以使用“-D”的参数完成。

```
git branch -D lxh
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git branch -D lxh
Deleted branch lxh (was db1c0f6).
```

可是现在在远程服务器上依然会存在此分支，那么就必须要一起删除掉，但是对于删除操作，除了之前使用过的方式之外，也可以推送一个空的分支，这样也表示删除。

- 删除方式一：

```
git push origin --delete lxh
```

- 推送一个空的分支过去：

```
git branch lxh
```

```
git push origin :lxh
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git push origin :lxh
Username for 'https://github.com': yootk
Password for 'https://yootk@github.com':
To https://github.com/yootk/mldn.git
- [deleted]          lxh
```

由于推送的是一个新的空分支，所以在服务器端会认为此推送的目的就是为了删除分支。

3.3、冲突自动解决

分支可以很好的实现多人开发的互操作，但是有可能出现这种情况。

- 现在建立了一个新的分支 brh，并且有一位开发者在此分支上修改了 Hello.java 文件；
- 但是这个开发者由于不小心的失误，又将分支切换回了 master 分支上，并且在 master 分支上也对 Emp.java 文件进行了修改。

等于现在有两个分支对同一个文件进行了修改，那么在进行提交的时候一定会出现一个冲突。因为系统不知道到底提交那一个分支的文件。

微信：



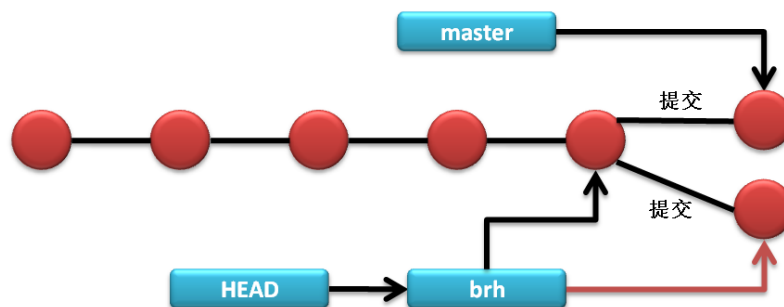
微博：



全国统一咨询电话：400-0088-518

解决冲突

- “master” 和 “brh” 两个分支上都有各自的信息提交，那么此时就形成了冲突。

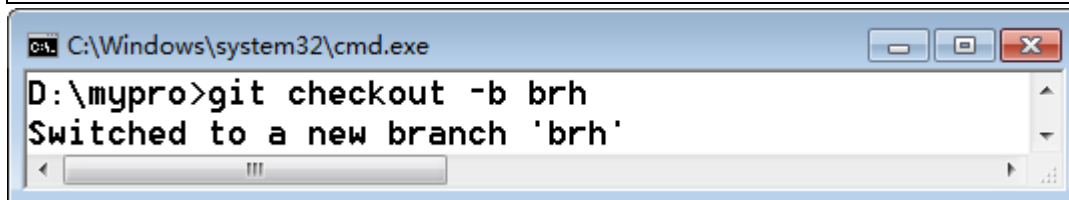


优拓教育 (www.yootk.com) & 魔乐科技 (www.mldnjava.cn) 联合出品

那么很明显，此时有两个提交点，那么会出现怎样的冲突警告呢？为了更好的说明问题，下面通过代码进行验证。

- 1、 建立并切换到 brh 分支上；

```
git checkout -b brh
```



- 2、 在此分支上修改 Hello.java 文件

```
public class Hello {  
    public static final String COMPANY_NAME = "yootk" ;  
    public static void main(String args[]) {  
        System.out.println("Happy New Year");  
        System.out.println("Happy MySelf");  
    }  
}
```

- 3、 在 brh 分支上提交此文件；

```
git commit -a -m "add static attribute"
```

- 4、 切换回 master 分支

```
git checkout master
```

- 5、 在 master 分支上也修改 Hello.java 文件；

```
public class Hello {  
    public static void main(String args[]) {
```

微信：



微博：



全国统一咨询电话：400-0088-518

```
System.out.println("Happy New Year");  
System.out.println("Happy MySelf");  
    print();  
}  
public static void print() {  
    System.out.println("*****");  
    System.out.println("*   Hello World!   *");  
    System.out.println("*****");  
}  
}
```

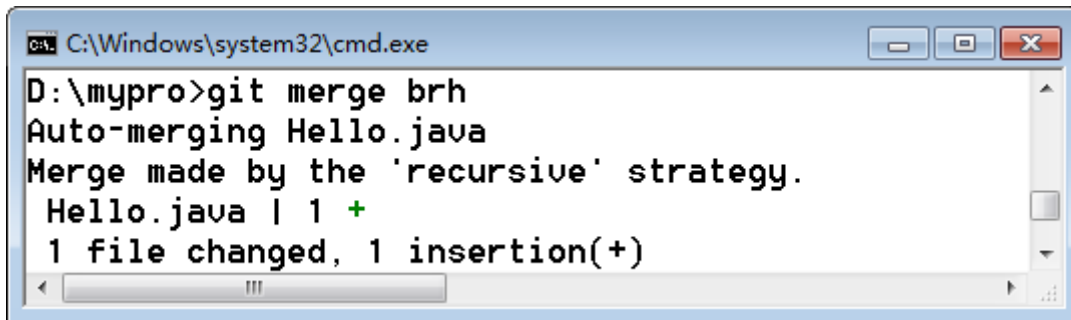
6、 在 master 分支上进行修改的提交

```
git commit -a -m "add static method"
```

现在在两个分支上都存在了代码的修改，而且很明显，修改的是同一个文件，那么自然进行分支合并的时候是无法合并的。

7、 合并分支（此时已经存在于 master 分支上）

```
git merge brh
```



以上的代码直接出现了自动合并，合并之后 Hello.java 程序的代码如下。

```
public class Hello {  
    public static final String COMPANY_NAME = "yootk";  
    public static void main(String args[]) {  
        System.out.println("Happy New Year");  
        System.out.println("Happy MySelf");  
        print();  
    }  
    public static void print() {  
        System.out.println("*****");  
        System.out.println("*   Hello World!   *");  
        System.out.println("*****");  
    }  
}
```

等于是现在的 GIT 工具帮助用户自己解决了冲突问题。

微信：



微博：



全国统一咨询电话：400-0088-518

3.4、冲突手工解决

以上由于代码的修改问题（很有规律），那么程序并没有发现不能够操作的冲突，那么下面进行一个更加有严格冲突产生的代码。

1、 准备过程

- 删除掉 brh 分支

```
git branch -d brh
```

- 修改 Hello.java 文件（还是在 master 分支上）

```
public class Hello {  
    public static void main(String args[]) {  
        System.out.println("*****");  
        System.out.println("#####");  
    }  
}
```

- 提交此修改：

```
git commit -a -m "simple print"
```

那么此时等于是 Hello.java 文件的内容已经变的很简单了。

2、 创建并切换到 brh 分支上；

```
git checkout -b brh
```

3、 修改 Hello.java 文件

```
public class Hello {  
    public static void main(String args[]) {  
        System.out.println("Hello World");  
        System.out.println("Hello Yootk");  
    }  
}
```

4、 在 brh 分支上进行提交；

```
git commit -a -m "hello print"
```

5、 切换到 master 分支上；

```
git checkout master
```

6、 在 master 分支上修改 Hello.java 文件；

```
public class Hello {  
    public static void main(String args[]) {  
        System.out.println("www.yootk.com");  
        System.out.println("www.mldnjava.cn");  
        System.out.println("bbs.mldn.cn");  
    }  
}
```

7、 提交 master 的修改

```
git commit -a -m "url print"
```

微信：



微博：



全国统一咨询电话：400-0088-518

现在两个修改都是针对于主方法中的内容进行的改变，于是在 master 分支中合并数据。

8、 合并分支

```
git merge brh
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git merge brh
Auto-merging Hello.java
CONFLICT (content): Merge conflict in Hello.java
Automatic merge failed; fix conflicts and then commit the result.
```

此时会直接提示出现了冲突。

9、 察看冲突的内容；

```
git status
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git status
On branch master
Your branch is ahead of 'origin/master' by 5 commits.
  (use "git push" to publish your local commits)

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

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

        both modified:   Hello.java

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

直接提示用户，两次修改了 Hello.java 文件。

10、 察看 Hello.java 文件

```
public class Hello {
    public static void main(String args[]) {
<<<<<<< HEAD
        System.out.println("www.yootk.com");
        System.out.println("www.mldnjava.cn");
        System.out.println("bbs.mldn.cn");
=====
        System.out.println("Hello World");
        System.out.println("Hello Yootk");
>>>>>>> brh
    }
```

微信：



微博：



全国统一咨询电话：400-0088-518

```
}
```

它现在把冲突的代码进行了标记，那么现在就必须人为手工修改发生冲突的文件。

11、手工修改 Hello.java 文件

```
public class Hello {  
    public static void main(String args[]) {  
        System.out.println("www.yootk.com");  
        System.out.println("www.mldnjava.cn");  
        System.out.println("bbs.mldn.cn");  
        System.out.println("Hello World");  
        System.out.println("Hello Yootk");  
    }  
}
```

现在是希望这几个输出的内容都同时进行保留。

12、此时已经手工解决了冲突，而后继续进行提交：

```
git commit -a -m "conflict print"
```

那么现在的冲突问题就解决了。

13、向服务器端提交信息

```
git push origin master
```

那么在实际的开发之中，一定会存在有许多的分支合并的情况，那么我怎么知道分支合并的历史呢？

14、察看合并的情况

```
git log --graph --pretty=oneline
```

```
C:\Windows\system32\cmd.exe  
D:\mypro>git log --graph --pretty=oneline  
* 85d834e76cca3f10b90d6b39f2a77c5cd33c3b3b conflict print  
| \  
| * 2c0ea4947de861bc9a603843b1b053077740a74e hello print  
* | 02e228208371904183e8c18944c0336a05f1f5db url print  
| /  
* f64fd97d432aff4092d577834adac369cfcb65cf simple print  
* 934acf1aec27b6baa7e695009dede607ec232f35 Merge branch 'brh'  
| \  
| * 1a2255b0e7dfc7c1e6f9770d73e0f969a39fb5f1 add static attribute  
* | 0de16f8bb3e72b9e8c3ed1a528704575123527c0 add static method  
| /
```

“-graph”指的是采用绘图的方式进行现实。

15、删除掉 brh 分支

```
git branch -d brh
```

那么此时的代码就可以回归正常的开发模式。

微信：



微博：



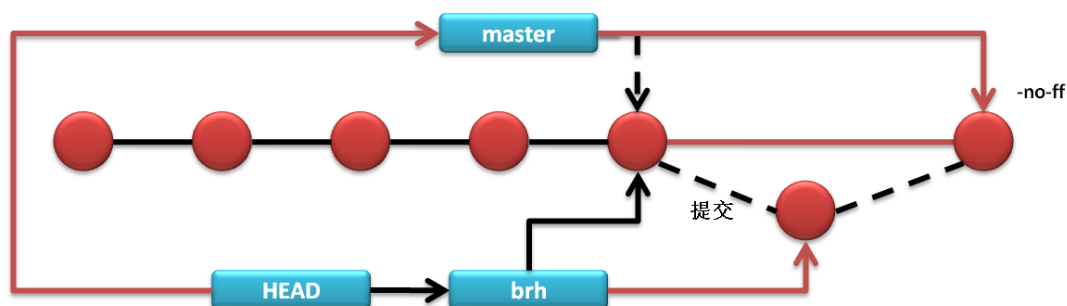
全国统一咨询电话：400-0088-518

3.5、分支管理策略

在之前进行分支合并的时候使用的全部都是“Fast forward”方式完成的，而此种方式只是改变了 master 指针，可是在分支的时候也可以不使用这种快合并，即：增加上一个“--no-ff”参数，这样就表示在合并之后会自动的再生成一个新的 commit id，从而保证合并数据的完整性。

使用--no-ff进行合并

➤ “--no-ff”。其作用是：合并后自动创建一个新的commit。



优拓教育 (www.yootk.com) & 魔乐科技 (www.mldnjava.cn) 联合出品

1、 创建一个新的分支

```
git checkout -b brh
```

2、 建立一个新的 Emp.java 文件

```
import java.io.* ;  
public class Emp implements Serializable {  
    private Integer empno ;  
    private String ename ;  
}
```

3、 提交修改；

```
git add .  
git commit -m "Add Emp.java File"
```

4、 切换回 master 分支

```
git checkout master
```

5、 使用非快速合并的方式进行代码合并

```
git merge --no-ff -m "no ff commit" brh
```

“--no-ff”方式会带有一个新的提交，所以需要为提交设置一个提交的注释。

微信:



微博:



全国统一咨询电话: 400-0088-518

```
C:\Windows\system32\cmd.exe
D:\mypro>git merge --no-ff -m "no ff commit" brh
Merge made by the 'recursive' strategy.
Emp.java | 5 +++++
1 file changed, 5 insertions(+)
create mode 100644 Emp.java
```

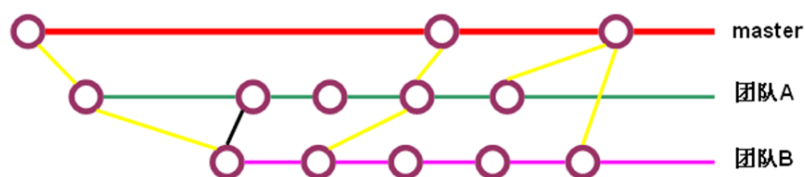
6、 来看一下提交的日志信息

```
git log --graph --pretty=oneline --abbrev-commit
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git log --graph --pretty=oneline --abbrev-commit
* 1b25149 no ff commit
| \
| * dac07a6 Add Emp.java File
| /
* 85d834e conflict print
| \
| * 2c0ea49 hello print
* | 02e2282 url print
| /
```

分支策略

- master分支应该是非常稳定的，也就是仅用来发布新版本，不要在此分支上开发；
- 在各个子分支上进行开发工作；
- 团队中的每个成员都在各个分支上工作。



微信：



微博：



全国统一咨询电话：400-0088-518

3.6、分支暂存

譬如说现在你正在一个分支上进行代码的开发，但是突然你的领导给了你一个新的任务，并且告诉你在半个小时内完成，那么怎么办？

难道那开发一半的分支要提交吗？不可能的，因为对于版本控制的基本的道德方式：你不能把有问题的代码提交上去，你所提交的代码一定都是正确的代码，那么为了这样的问题，在 GIT 中提供了一个分支暂存的机制，可以将开发一半的分支进行保存，而后在适当的时候进行代码的恢复。

那么下面首先创建一个基本的开发场景。

- 1、 创建并切换到一个新的分支：

```
git checkout -b brh
```

- 2、 下面在分支上编写 Emp.java 类的文件：

```
import java.io.* ;
import java.util.* ;

public class Emp implements Serializable {
    private Integer empno ;
    private String ename ;
    private Date hiredate ;
}
```

- 3、 将此文件保存在暂存区之中

```
git add .
```

这个时候由于代码还没有开发完成，所以不能够进行代码的提交。但是你的老板给了你一个新的任务，那么你就不得不去停止当前的开发任务，所以需要当前的开发进度进行“暂存”，等日后有时间了继续进行恢复开发。

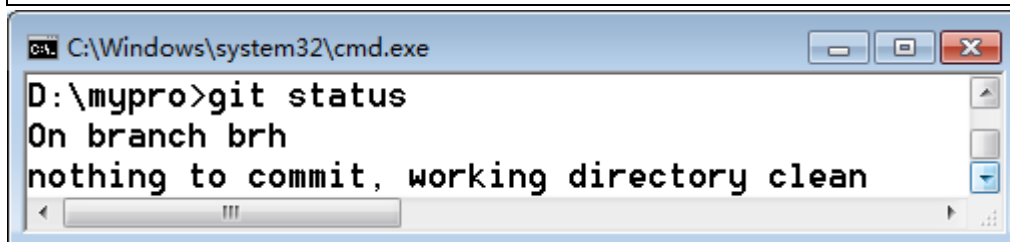
- 4、 将工作暂存

```
git stash
```



- 4、 察看一下当前的工作区中的内容：

```
git status
```



此处会直接告诉用户当前的工作区之中没有任何的修改。

- 5、 而后现在假设要修改的代码还处于 master 分支上，所以下面切换到 master 分支中：

```
git checkout master
```

微信：



微博：



全国统一咨询电话：400-0088-518

```
C:\Windows\system32\cmd.exe
D:\mypro>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)
```

那么现在假设说创建一个新的分支，用于完成老板的需求，假设分支的名称为“dev”（也有可能是一个 bug 调试）。

6、 创建并切换分支

```
git checkout -b dev
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git checkout -b dev
Switched to a new branch 'dev'
```

7、 在新的分支中修改 Hello.java 文件

```
public class Hello {
    public static void main(String args[]) {
        System.out.println("www.yootk.com");
        System.out.println("www.mldnjava.cn");
        System.out.println("bbs.mldn.cn");
        System.out.println("Hello World");
        System.out.println("Hello Yootk");
        System.out.println("HAPPY");
    }
}
```

8、 提交修改的操作：

```
git commit -a -m "dev change"
```

那么到此老板的需求已经完成了，但是这个代码还处于 dev 分支之中，那么现在切换回到 master 分支并且进行合并。

9、 切换回 master 分支

```
git checkout master
```

10、 合并 deve 分支，使用 no fast forward

```
git merge --no-ff -m "merge dev branch" dev
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git merge --no-ff -m "merge dev branch" dev
Merge made by the 'recursive' strategy.
 Hello.java | 1 +
 1 file changed, 1 insertion(+)
```

11、 那么现在突发的问题已经被解决了，被解决之后对于 dev 的分支将没有任何的存在意义，可以直接删除；

微信：



微博：

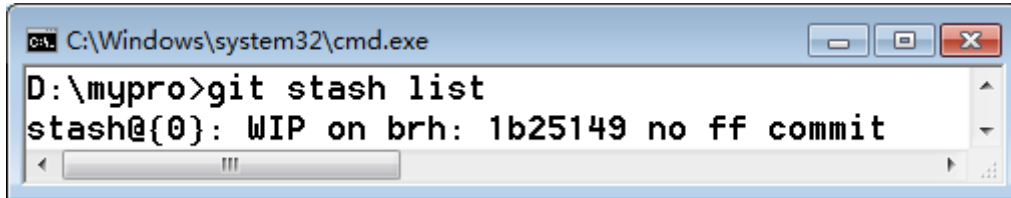


全国统一咨询电话：400-0088-518

```
git branch -d dev
```

12、那么需要回归到已有的工作状态，但是有可能会存在有许多的暂存的状态，可以直接使用如下命令进行列出。

```
git stash list
```



```
C:\Windows\system32\cmd.exe
D:\mypro>git stash list
stash@{0}: WIP on brh: 1b25149 no ff commit
```

13、从暂存区之中进行恢复

暂存区恢复之后那么所暂停的操作将没有存在的意义，但是也有人会认为它有意义，所以对于恢复有两种形式：

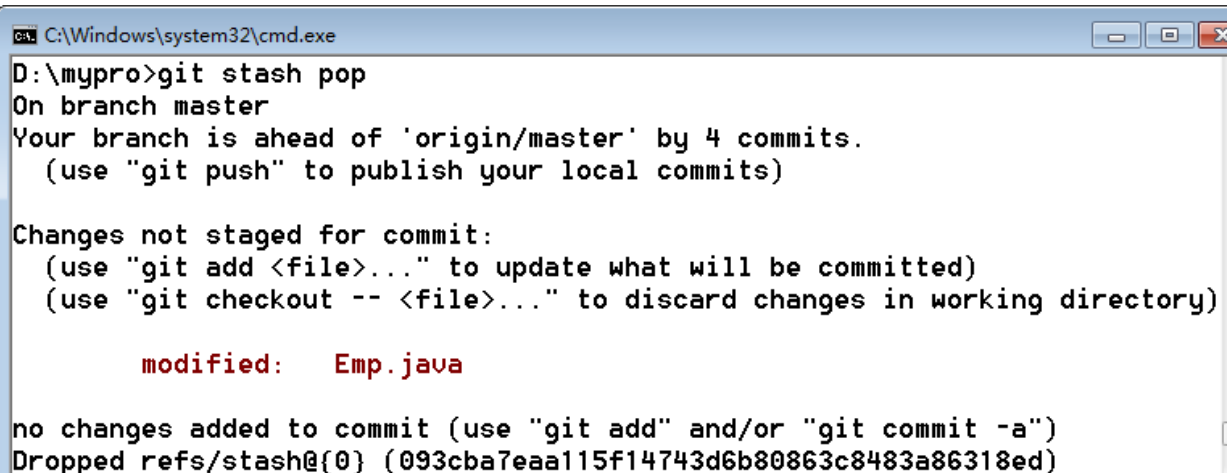
- 形式一：先恢复，而后再手工删除暂存

```
git stash apply
```

```
git stash drop
```

- 形式二：恢复的同时也将 stash 内容删除：

```
git stash pop
```



```
C:\Windows\system32\cmd.exe
D:\mypro>git stash pop
On branch master
Your branch is ahead of 'origin/master' by 4 commits.
(use "git push" to publish your local commits)

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:   Emp.java

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (093cba7eaa115f14743d6b80863c8483a86318ed)
```

那么下面的任务就可以像之前那样进行代码的提交，而后删除掉 brh 分支：

```
git commit -a -m "change Emp.java"
```

```
git branch -d brh
```

使用暂存策略可以很方便的解决代码突然暂停修改的操作，是非常方便。

3.7、补丁：patch

补丁并不是针对于所有代码的修改，只是针对于局部的修改。在很多的代码维护之中，如果按照最早克隆的方式将代码整体克隆下来实际上所花费的资源是非常庞大的，但是修改的时候可能只修改很小的一部分代码，所以在这种情况下就希望可以将一些代码的补丁信息发送给开发者。而发给开发者之后他需要知道那些代码被修改了，这样的话就可以使用一个极低的开销实现代码的修改操作，而在 GIT 之中也提供了两种简单的补丁方案：

- 使用 git diff 生成标准的 patch；
- 使用 git format-patch 声称 git 专用的 patch。

微信：



微博：



全国统一咨询电话：400-0088-518

3.7.1、利用 git diff 生成标准的 patch

补丁一定是针对于文件的修改进行的所以下面是以 Emp.java 文件为例.

1、 当前的 Emp.java 文件

```
import java.io.* ;
import java.util.* ;
public class Emp implements Serializable {
    private Integer empno ;
    private String ename ;
    private Date hiredate ;
}
```

2、 建立一个新的分支 —— cbrh

```
git checkout -b cbrh
```

3、 修改 Emp.java 文件

```
import java.io.* ;
import java.util.* ;
public class Emp implements Serializable {
    private Integer empno ;
    private String ename ;
    private Date hiredate ;
    private Double sal ;
    private Double comm ;
    private Emp mgr ;
}
```

4、 而后察看前后代码的不同

```
git diff Emp.java
```

```
C:\Windows\system32\cmd.exe
D:\mypro>git diff Emp.java
diff --git a/Emp.java b/Emp.java
index 052a225..56e61d3 100644
--- a/Emp.java
+++ b/Emp.java
@@ -4,4 +4,7 @@ public class Emp implements Serializable {
     private Integer empno ;
     private String ename ;
     private Date hiredate ;
+    private Double sal ;
+    private Double comm ;
+    private Emp mgr ;
 }
\ No newline at end of file
```

微信：



微博：

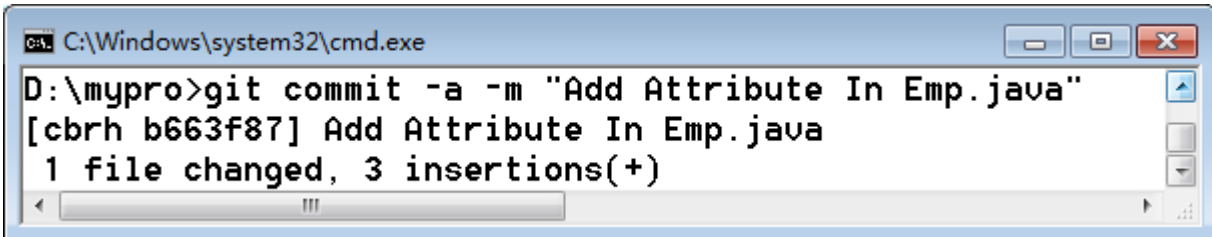


全国统一咨询电话：400-0088-518

此时可以发现 Emp.java 文件修改前后的对比情况。

- 5、 在 cbrh 上进行代码的提交；

```
git commit -a -m "Add Attribute In Emp.java"
```



此时并没有和主分支进行提交，但是代码已经改变了，需要的是将代码的变化提交给开发者。

- 6、 生成补丁文件 —— mypat

```
git diff master > mypat
```

- 7、 切换回 master 分支

```
git checkout master
```

此时会自动在项目目录中生成一个 mypat 的补丁文件信息。这个文件是可以由 git 读懂的信息文件，那么完成之后现在需要模拟另外一个开发者，另外一个开发者假设是专门进行补丁合并的开发者。

- 8、 创建并切换一个新的分支

```
git checkout -b patchbrh
```

- 9、 应用补丁信息

```
git apply mypat
```

此时补丁可以成功的使用了。

- 10、 提交补丁的操作；

```
git commit -a -m "Patch Apple"
```

- 11、 切换回 master 分支之中进行分支合并

```
git checkout master  
git merge --no-ff -m "Merge Patch" patchbrh
```

这样如果只是将补丁数据的文件发送给开发者，那么就没有必要进行大量代码的传输，并且在创建补丁的时候也可以针对于多个文件进行补丁的创建。

3.7.2、利用 git format-patch 生成 GIT 专用补丁

那么下面还是利用分支修改 Emp.java 文件。

- 1、 创建并切换到 cbrh 分支

```
git branch -D cbrh  
git branch -D patchbrh  
git checkout -b cbrh
```

- 2、 修改 Emp.java 文件

```
import java.io.*;  
import java.util.*;  
public class Emp implements Serializable {
```

微信：



微博：



全国统一咨询电话：400-0088-518

```
private Integer empno ;  
private String ename ;  
private Date hiredate ;  
private Double sal ;  
private Double comm ;  
private Emp mgr ;  
public String toString() {  
    return "一位雇员。";  
}  
}
```

现在在文件之中增加了一个 toString() 的方法。

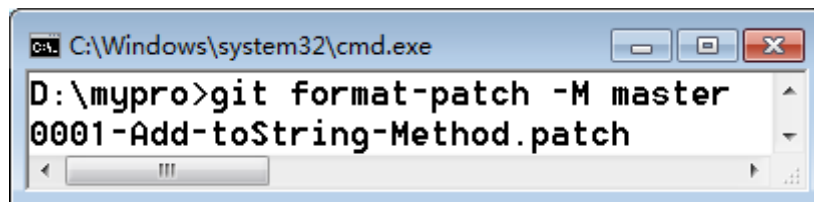
3、 将代码进行提交

```
git commit -a -m "Add toString Method"
```

4、 下面需要与原始代码做一个比较，而且比较后会自动的生成补丁文件

```
git format-patch -M master
```

现在表示要与 master 分支进行比较（而-M 参数就是指定分支）。



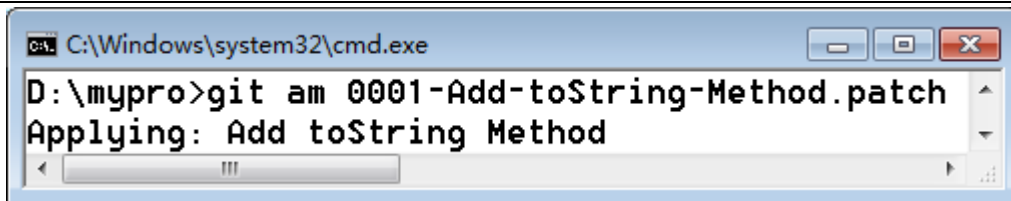
此时已经生成了一个补丁文件，因为只修改了一次的內容。这个补丁文件严格来将就是一个 email 数据，需要将此数据发送给开发者，而后开发者可以进行补丁的应用。

5、 创建并切换到 patchbrh 分支上

```
git checkout master  
git checkout -b patchbrh
```

6、 应用补丁的信息，利用 “git am” 完成

```
git am 0001-Add-toString-Method.patch
```



现在是将发送过来的，带有 email 格式的补丁文件进行了应用。

7、 提交应用的更新

```
git commit -a -m "method patch apply"
```

那么此时就可以成功的应用补丁进行代码的更正。

关于两种补丁方式的说明：

- 使用 git diff 生成补丁兼容性是比较好的，如果你是在不是 git 管理的仓库上，此类方式生成的补丁是非常容易接受的；
- 但是如果你是向公共的开发社区进行代码的补丁更正，那么建议使用 git format-patch，这样不仅标准，而且也可

微信：



微博：



全国统一咨询电话：400-0088-518

以将更正人的信息进行公布。

3.8、多人协作开发

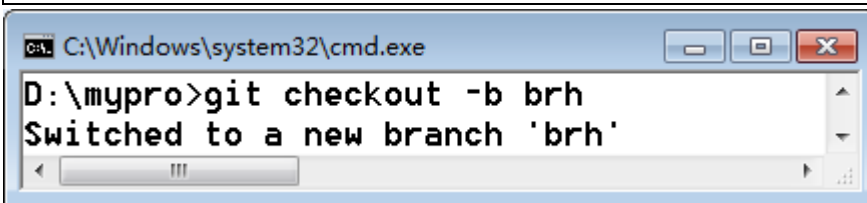
分支的处理实际上是为了更好的多人开发做出的准备，那么下面就将利用两个命令行方式（模拟其他的开发者）进行项目代码的编写。在讲解之前首先说明一下：

- 一般而言，master 分支项目的核心分支，只要进行代码的克隆，那么此分支一定会被保存下来；
- 开发者往往会建立一系列的分支，譬如，我个人建立了一个 brh 的分支进行代码的编写；
- 如果要进行调试可以建立一个 bug 分支；
- 如果要增加某些新的功能则可以建立 feature 分支。

那么下面首先针对于代码进行一些准备：

1、 创建并切换到一个新的分支：brh

```
git checkout -b brh
```



2、 在新的分支上建立一个新的文件 —— Dept.java

```
import java.io.* ;
import java.util.* ;

public class Dept implements Serializable {
    private Integer deptno ;
    private String dname ;
    private String loc ;
}
```

3、 将此代码进行提交：

```
git add .
git commit -a -m "Add Dept.java Files"
```

4、 将两个分支提交到服务器上去

```
git push origin master
git push origin brh
```

5、 [二号]为了模拟第二个开发者，所以建立一个新的命令行窗口，并且将代码复制下来（d:\proclone）

```
git clone https://github.com/yootk/mldn.git
```

微信：



微博：



全国统一咨询电话：400-0088-518

```
C:\Windows\system32\cmd.exe
D:\proclone>git clone https://github.com/yootk/mldn.git
Cloning into 'mldn'...
remote: Counting objects: 63, done.
remote: Compressing objects: 100% (44/44), done.
remote: Total 63 (delta 21), reused 59 (delta 17)
Unpacking objects: 100% (63/63), done.
Checking connectivity... done.
```

6、 [二号]察看分支信息：

```
git branch -a
```

```
C:\Windows\system32\cmd.exe
D:\proclone\mldn>git branch -a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/brh
remotes/origin/master
```

发现现在只是将 master 分支拷贝下来了，但是 brh 分支并没有存在。

7、 [二号]建立并切换到 brh 分支上

```
git checkout -b brh
```

8、 [二号]将远程服务器端上的 brh 分支的内容拷贝到本地的 brh 分支上

```
git merge origin/brh
```

```
C:\Windows\system32\cmd.exe
D:\proclone\mldn>git merge origin/brh
Updating 23ade86..1c7dd07
Fast-forward
 Dept.java | 7 ++++++
 1 file changed, 7 insertions(+)
 create mode 100644 Dept.java
```

9、 [二号]现在开发者增加了一个 Admin.java 文件

```
import java.io.*;
import java.util.*;
public class Admin implements Serializable {
    private String adminid;
    private String password;
}
```


微信：



微博：



全国统一咨询电话：400-0088-518

10、[二号]将新的代码进行提交：

```
git add .  
git commit -m "Add Admin.java File"
```

11、[二号]现在本地的 brh 分支代码发生了变化，那么应该将此变化提交到远程的 brh 分支上：

```
git push origin brh
```

现在代码已经发送到了服务器上了，并且在 brh 分支上增加了新的 Admin.java 文件。

12、[一号]这个时候最原始的开发者的目录下还只是上一次提交的内容。那么需要取得最新的数据才可以。

对于取得最新的分支数据有两种方式：

- git fetch：此操作只是取得最新的分支数据，但是不会发生 merge 合并操作；
- git pull：此操作取出最新分支数据，并且同时发生 merge 合并操作。

```
git pull
```

```
C:\Windows\system32\cmd.exe  
D:\mypro>git pull  
remote: Counting objects: 3, done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 1), reused 3 (delta 1)  
Unpacking objects: 100% (3/3), done.  
From https://github.com/yootk/mldn  
07eb31e..a167e39 brh -> origin/brh  
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> brh
```

实际上错误信息也很简单，指的是，当前的 brh 分支和服务器的分支没有关系，所以如果要想读取代码，必须让两个分支产生关联关系。

```
git branch --set-upstream-to=origin/brh
```

随后再次读取所有的代码。

```
git pull
```

```
C:\Windows\system32\cmd.exe  
D:\mypro>git pull  
Updating 07eb31e..a167e39  
Fast-forward  
Admin.java | 6 ++++++  
1 file changed, 6 insertions(+)  
create mode 100644 Admin.java
```

这个时候就实现了不同开发者之间的代码互相关联。

微信：



微博：



全国统一咨询电话：400-0088-518

13、[二号]修改 Admin.java 类文件；

```
import java.io.* ;
import java.util.* ;
public class Admin implements Serializable {
    private String adminid ;
    private String password ;
    private Date lastLogin ;
}
```

14、[二号]将以上的代码进行提交；

```
git commit -a -m "Update Admin.java File"
```

15、[二号]向服务器端提交代码的修改；

```
git push origin brh
```

16、[一号]开发者也进行 Admin.java 文件的修改；

```
import java.io.* ;
import java.util.* ;
public class Admin implements Serializable {
    private String adminid ;
    private String password ;
    private Integer flag ;
    private String name ;
}
```

17、[一号]将代码提交

```
git commit -a -m "2 Update Admin.java File"
```

但是这个时候很明显，两个用户一起修改了同一个文件。

18、[一号]抓取最新的更新数据

```
git pull
```

现在可以发现，此时的程序，是两位开发者修改了同一个代码，所以产生了冲突。同时一号开发者之中的 Admin.java 文件的内容已经变更为如下情况。

```
import java.io.* ;
```

微信：



微博：



全国统一咨询电话：400-0088-518

```
import java.util.* ;
public class Admin implements Serializable {
    private String adminid ;
    private String password ;
    <<<<<<< HEAD
    private Integer flag ;
    private String name ;
    =====
    private Date lastLogin ;
    >>>>>>> cc2b55381902d971881ec816c2c4d6825e456656
}
```

19、 [一号]手工解决冲突文件内容

```
import java.io.* ;
import java.util.* ;
public class Admin implements Serializable {
    private String adminid ;
    private String password ;
    private Integer flag ;
    private String name ;
    private Date lastLogin ;
}
```

20、 再次执行提交和服务器推送

```
git commit -a -m "3 Update Admin.java File"
git push origin brh
```

现在已经成功的由本地的冲突扩充到了远程的冲突，相信通过一系列的代码大家也可以更好的理解分支的操作问题。

4、总结

在整个 GIT 学习之中，分支的管理操作是最为麻烦的也是最为重要的操作，所以在此部分的程序，希望大家可以反复的练习。分支是进行开发使用的，最终的代码都在 master 分支上。

微信：



微博：



全国统一咨询电话：400-0088-518

标签管理

优拓教育 (www.yootk.com) & 魔乐科技 (www.mldnjava.cn) 联合出品