

# GIT版本创建

新建一个文件夹，然后再命令行中将工作路径切换到新创建的文件夹中，输入命令 `git init` 初始化GIT。



```
C:\WINDOWS\system32\cmd.exe

Microsoft Windows [版本 10.0.10586]
(c) 2015 Microsoft Corporation。保留所有权利。

C:\Users\佳字>d:

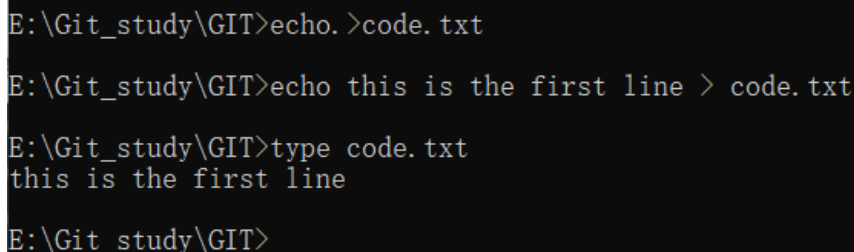
D:\>cd
```

然后在新创建的文件夹下出现一个叫做 `.git` 的隐藏文件（这个文件夹就是 Git 用来跟踪管理版本迭代的）。

## 使用

(1) 在新创建的文件夹（GIT）下，创建一个文件（code.txt），编辑内容：

```
echo.>code.txt
echo this is the first line > code.txt
type code.txt
```



```
E:\Git_study\GIT>echo.>code.txt

E:\Git_study\GIT>echo this is the first line > code.txt

E:\Git_study\GIT>type code.txt
this is the first line

E:\Git_study\GIT>
```

(2) 使用如下命令可以创建一个版本：

```
git add code.txt
git commit -m '版本1'
```

```
E:\Git_study\GIT>git add code.txt

E:\Git_study\GIT>git commit -m '版本1'
[master (root-commit) c3b66d7] '版本1'
1 file changed, 1 insertion(+)
create mode 100644 code.txt
```

(3) 使用如下命令可以查看版本记录:

```
git log
```

```
E:\Git_study\GIT>git log
commit c3b66d74637c683e9a3e6d7d0742db23ed4b883c (HEAD -> master)
Author: hwly <1245470853@qq.com>
Date:   Sun Dec 5 13:44:55 2021 +0800

    '版本1'
```

(4) 继续编辑 code.txt, 在里面增加一行。

```
echo this is the second line >> code.txt
```

```
E:\Git_study\GIT>echo this is the second line >> code.txt

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
```

(5) 使用如下命令再创建一个版本并查看版本记录:

```
E:\Git_study\GIT>git add code.txt

E:\Git_study\GIT>git commit -m '版本2'
[master 3f9b315] '版本2'
1 file changed, 1 insertion(+)

E:\Git_study\GIT>git log
commit 3f9b315f65b5b9b0df16b3a974c073f50a8855de (HEAD -> master)
Author: hwly <1245470853@qq.com>
Date:   Sun Dec 5 13:50:21 2021 +0800

    '版本2'

commit c3b66d74637c683e9a3e6d7d0742db23ed4b883c
Author: hwly <1245470853@qq.com>
Date:   Sun Dec 5 13:44:55 2021 +0800

    '版本1'
```

(6) 现在若想回到某一个版本, 可以使用如下命令:

```
git reset --hard HEAD^
```

其中, HEAD 表示当前最新版本, HEAD^ 表示当前版本的前一个版本, HEAD^^ 表示当前版本的前前个版本, 也可以使用 HEAD~1 表示当前版本的前一个版本, HEAD~100 表示当前版本的前 100 版本。

在 Windows 的命令提示字符 cmd.exe 里却无法执行, 会出现错误:

```
E:\Git_study\GIT>git reset --hard HEAD^
More?
More?
fatal: ambiguous argument 'HEAD'
: unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]
```

^是cmd.exe的escape字符，属于特殊字符，命令里要用到文字 ^ 时必须用双引号把它夹起来，因此只要如下就可以正确执行：

```
git reset --hard HEAD"^"
```

或者：

```
git reset --hard "HEAD^"
```

或者直接使用：

```
git reset --hard HEAD~1
```

```
E:\Git_study\GIT>git reset --hard HEAD~1
HEAD is now at c3b66d7 '版本1'

E:\Git_study\GIT>git log
commit c3b66d74637c683e9a3e6d7d0742db23ed4b883c (HEAD -> master)
Author: hwly <1245470853@qq.com>
Date: Sun Dec 5 13:44:55 2021 +0800

    '版本1'
```

(7) 如果想要回到版本2，可以使用如下命令：

```
git reset --hard 版本号
```

从上面可以看到版本2的版本号为：

```
E:\Git_study\GIT>git log
commit 3f9b315f65b5b9b0df16b3a974c073f50a8855de (HEAD -> master)
Author: hwly <1245470853@qq.com>
Date: Sun Dec 5 13:50:21 2021 +0800

    '版本2'

commit c3b66d74637c683e9a3e6d7d0742db23ed4b883c
Author: hwly <1245470853@qq.com>
Date: Sun Dec 5 13:44:55 2021 +0800

    '版本1'
```

```
git reset --hard 3f9b315f65b5 # 复制前面几个也是可以的
```

```
E:\Git_study\GIT>git reset --hard 3f9b315f65b5
HEAD is now at 3f9b315 '版本2'

E:\Git_study\GIT>git log
commit 3f9b315f65b5b9b0df16b3a974c073f50a8855de (HEAD -> master)
Author: hwly <1245470853@qq.com>
Date: Sun Dec 5 13:50:21 2021 +0800

    '版本2'

commit c3b66d74637c683e9a3e6d7d0742db23ed4b883c
Author: hwly <1245470853@qq.com>
Date: Sun Dec 5 13:44:55 2021 +0800

    '版本1'
```

(9) 假如说上面的终端已经关掉了，怎么回退版本。

执行如下命令将版本回退到版本1。

```
E:\Git_study\GIT>git reset --hard HEAD^^
HEAD is now at c3b66d7 '版本1'

E:\Git_study\GIT>git log
commit c3b66d74637c683e9a3e6d7d0742db23ed4b883c (HEAD -> master)
Author: hwly <1245470853@qq.com>
Date: Sun Dec 5 13:44:55 2021 +0800

    '版本1'
```

下面把终端关了，然后再打开终端，发现之前版本2的版本号看不到了。

使用 `git reflog` 命令可以查看操作的记录。

```
git reflog
```

```
E:\Git_study\GIT>git reflog
c3b66d7 (HEAD -> master) HEAD@{0}: reset: moving to HEAD^
3f9b315 HEAD@{1}: reset: moving to 3f9b315f65b5
c3b66d7 (HEAD -> master) HEAD@{2}: reset: moving to HEAD~1
3f9b315 HEAD@{3}: reset: moving to HEAD
3f9b315 HEAD@{4}: reset: moving to HEAD
3f9b315 HEAD@{5}: commit: '版本2'
c3b66d7 (HEAD -> master) HEAD@{6}: commit (initial): '版本1'
```

```
E:\Git_study\GIT>git reset --hard 3f9b315
HEAD is now at 3f9b315 '版本2'

E:\Git_study\GIT>git log
commit 3f9b315f65b5b9b0df16b3a974c073f50a8855de (HEAD -> master)
Author: hwly <1245470853@qq.com>
Date: Sun Dec 5 13:50:21 2021 +0800

    '版本2'

commit c3b66d74637c683e9a3e6d7d0742db23ed4b883c
Author: hwly <1245470853@qq.com>
Date: Sun Dec 5 13:44:55 2021 +0800

    '版本1'

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
```

## 工作区和暂存区

## 工作区

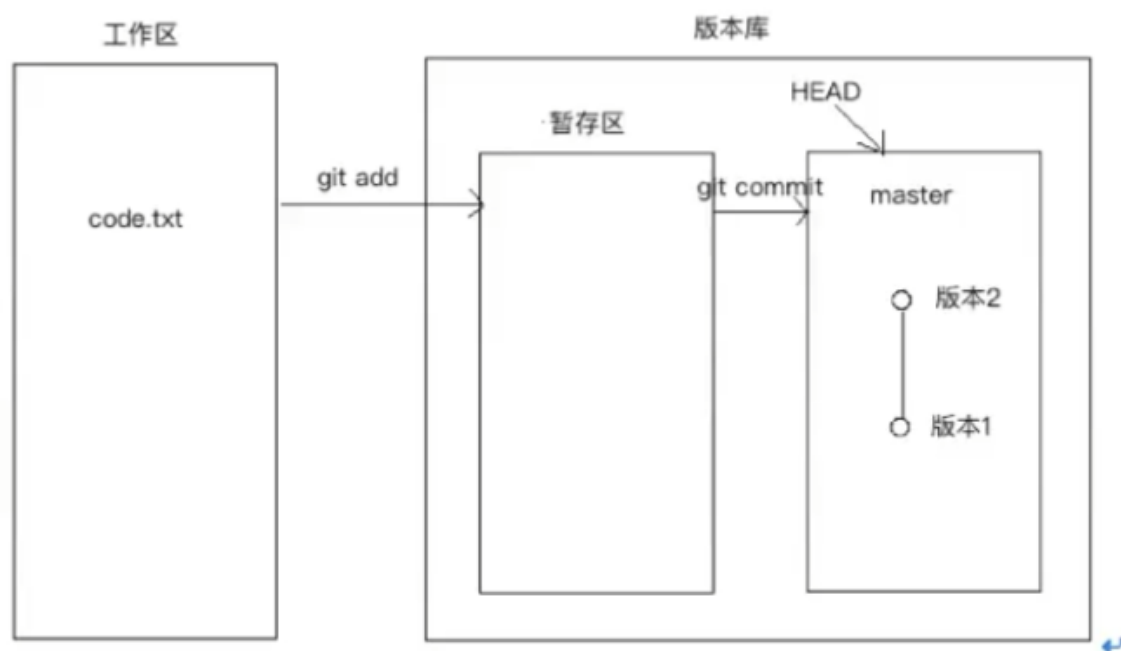
电脑中的目录，比如新创建的文件夹（GIT），就是一个工作区

## 版本库

工作区有个一个隐藏目录 `.git`，这个不是工作区，而是 `git` 的版本库。`git` 的版本库里存了很多东西，其中最重要的就是称为 `stage`（或者叫 `index`）的暂存区，还有 `git` 为我们自动创建的第一个分支 `master`，以及指向 `master` 的一个指针叫 `HEAD`。

因为创建 `git` 版本库时，`git` 自动为我们创建了唯一一个 `master` 分支，所以，现在 `git commit` 就是往 `master` 分支上提交更改。

可以简单理解为，需要提交的文件修改统统放到暂存区，然后，一次性提交暂存区的所有修改。



(1) 下面在 `GIT` 目录下再创建一个文件 `code2.txt`，然后编辑内容如下：

```
echo.>code2.txt
echo the code2 first line > code2.txt
type code2.txt
```

```
E:\Git_study\GIT>echo.>code2.txt
E:\Git_study\GIT>echo the code2 first line > code2.txt
E:\Git_study\GIT>type code2.txt
the code2 first line
```

(2) 然后再次编辑 `code.txt` 内容，在其中加入一行，编辑后内容如下：

```
echo this is the third line >> code.txt
type code.txt
```

```
E:\Git_study\GIT>echo this is the third line >> code.txt

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
```

(3) 使用如下命令查看当前工作树的状态:

```
git status
```

```
E:\Git_study\GIT>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   code.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        code2.txt

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

上面提示我们 `code.txt` 被修改, 而 `code2.txt` 没有被跟踪。

(4) 使用如下命令把 `code.txt` 和 `code2.txt` 加入到暂存区, 然后再执行 `git status` 命令, 结果如下:

```
E:\Git_study\GIT>git add code.txt code2.txt

E:\Git_study\GIT>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   code.txt
        new file:   code2.txt
```

所有 `git add` 命令是把所有提交的修改存放到暂存区。

(5) 然后执行 `git commit` 就可以一次性把暂存区的所有修改提交到分支创建一个版本。

```

E:\Git_study\GIT>git commit -m '版本3'
[master 19b24d8] '版本3'
 2 files changed, 2 insertions(+)
 create mode 100644 code2.txt

E:\Git_study\GIT>git log
commit 19b24d865ba9c7fbb3d06b98aec1b369f5b85763 (HEAD -> master)
Author: hwly <1245470853@qq.com>
Date:   Sun Dec 5 14:44:33 2021 +0800

    '版本3'

commit 3f9b315f65b5b9b0df16b3a974c073f50a8855de
Author: hwly <1245470853@qq.com>
Date:   Sun Dec 5 13:50:21 2021 +0800

    '版本2'

commit c3b66d74637c683e9a3e6d7d0742db23ed4b883c
Author: hwly <1245470853@qq.com>
Date:   Sun Dec 5 13:44:55 2021 +0800

    '版本1'

```

(6) 一旦提交后，如果没有对工作区做任何修改，那么工作区就是“干净”的。执行如下命令可以发现：

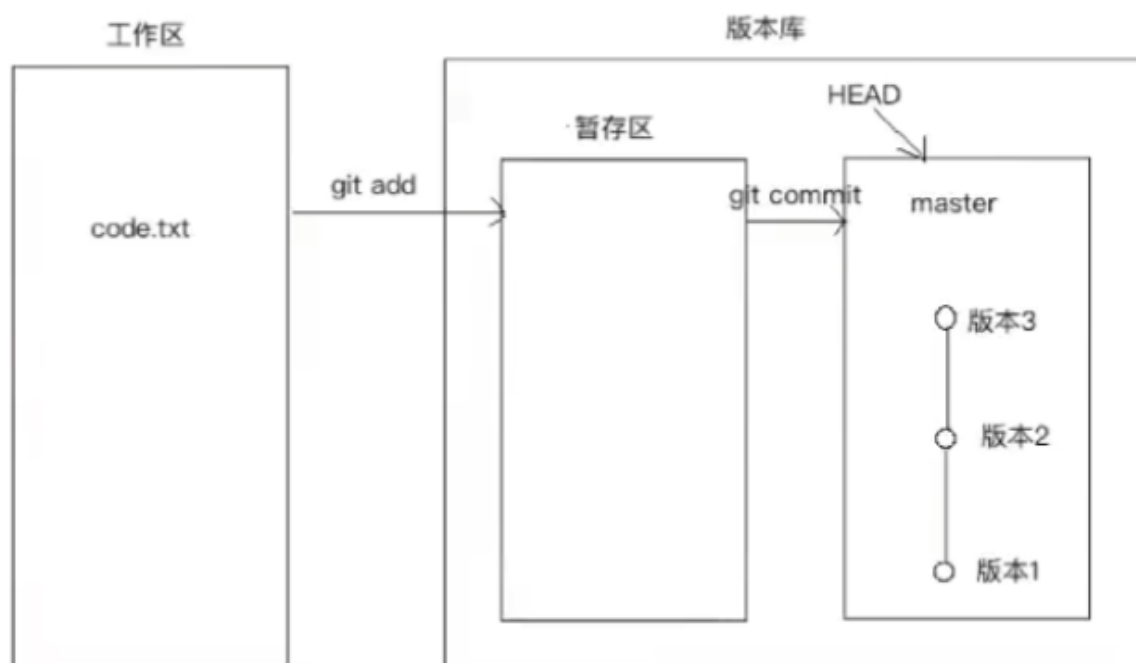
```
git status
```

```

E:\Git_study\GIT>git status
On branch master
nothing to commit, working tree clean

```

现在，版本库变成了这样：



## 管理修改

git 管理的文件的修改，它只会提交暂存区的修改来创建版本。

(1) 编辑 `code.txt`，并使用 `git add` 命令将其添加到暂存区。

```
E:\Git_study\GIT>echo this is the forth line >> code.txt

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line

E:\Git_study\GIT>git add code.txt
```

(2) 继续编辑 `code.txt`，并在其中添加一行：

```
E:\Git_study\GIT>echo this is the new line >> code.txt

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
this is the new line
```

(3) `git commit` 创建一个版本，并使用 `git status` 查看，发现第二次修改 `code.txt` 内容之后，并没有将其添加到工作区，所以创建版本的时候并没有被提交。

```
E:\Git_study\GIT>git commit -m '版本4'
[master 07a7b38] '版本4'
1 file changed, 1 insertion(+)

E:\Git_study\GIT>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   code.txt

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

## 撤销修改

(1) 继续上面的操作，提示我们可以使用 `git restore <文件>` 来丢弃工作区的改动。执行如下命令，发现工作区干净了，第二次的改动内容也没了。



```
E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
this is the new line

E:\Git_study\GIT>git restore code.txt

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line

E:\Git_study\GIT>git status
On branch master
nothing to commit, working tree clean
```

(2) 继续编辑 `code.txt`，并在其中添加如下内容，并将其添加到暂存区。

```
E:\Git_study\GIT>echo the new line >> code.txt

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
the new line

E:\Git_study\GIT>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   code.txt

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

E:\Git_study\GIT>git add code.txt

E:\Git_study\GIT>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   code.txt
```

(3) `git` 同样告诉我们使用命令 `git restore --staged <file>` 可以把暂存区的修改撤销掉，重新放回工作区。

```
E:\Git_study\GIT>git restore --staged code.txt

E:\Git_study\GIT>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   code.txt

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

(4) 现在若想要丢弃 `code.txt` 的修改，执行如下命令即可。

```
E:\Git_study\GIT>git restore code.txt

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line

E:\Git_study\GIT>git status
On branch master
nothing to commit, working tree clean
```

现在，如果不但改错了东西，还从暂存区提交到了版本库，则需要进行版本的回退。

### 小结：

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，使用命令 `git restore file`。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想要丢弃修改，分两步，第一步用命令 `git restore --staged file`，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考回退一节。

## 对比文件的不同

对比工作区和某个版本中的文件的不同：

(1) 继续编辑文件 `code.txt`，在其中添加一行内容。

```
E:\Git_study\GIT>echo the new line >> code.txt

E:\Git_study\GIT>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   code.txt

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

E:\Git_study\GIT>git diff HEAD -- code.txt
diff --git a/code.txt b/code.txt
index 25fcla2..7804c87 100644
--- a/code.txt
+++ b/code.txt
@@ -2,3 +2,4 @@ this is the first line
 this is the second line
 this is the third line
 this is the forth line
+the new line
```

- 代表HEAD版本中code.txt的内容      + 代表工作区code.txt的内容

工作区code.txt比HEAD中code.txt多了一行内容。

(3) 使用如下命令丢弃工作区的改动。

```
git restore code.txt
```

```
E:\Git_study\GIT>git restore code.txt

E:\Git_study\GIT>git status
On branch master
nothing to commit, working tree clean
```

对比两个版本间文件的不同：

(1) 现在要对比 `HEAD` 和 `HEAD^` 版本中 `code.txt` 的不同，使用如下命令：

```
E:\Git_study\GIT>git diff HEAD HEAD~1 -- code.txt
diff --git a/code.txt b/code.txt
index 25fcla2..81f1f87 100644
--- a/code.txt
+++ b/code.txt
@@ -1,4 +1,3 @@
 this is the first line
 this is the second line
 this is the third line
- this is the forth line
```

## 删除文件

(1) 把目录中的 `code2.txt` 删除。

```
del code2.txt
```

这个时候，`git` 知道删除了文件，因此，工作区和版本库就不一致了，`git status` 命令会立刻提示哪些文件被删除了。

```
E:\Git_study\GIT>del code2.txt

E:\Git_study\GIT>git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    code2.txt

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

(2) 现在有两个选择，一是确定要从版本库中删除该文件，那就用命令 `git rm` 删掉，并且 `git commit`：

```
E:\Git_study\GIT>git rm code2.txt
rm 'code2.txt'

E:\Git_study\GIT>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    code2.txt

E:\Git_study\GIT>git commit -m '删除code2.txt'
[master 911f079] '删除code2.txt'
 1 file changed, 1 deletion(-)
 delete mode 100644 code2.txt

E:\Git_study\GIT>git log --pretty=oneline
911f079aa42f59bc1cc95951229ed1e81ff24da7 (HEAD -> master) '删除code2.txt'
07a7b389bb1fe1fece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

另一种情况是删错了，可以直接使用 `git restore code2.txt`，这样文件 `code2.txt` 又回来了。

**小结：**

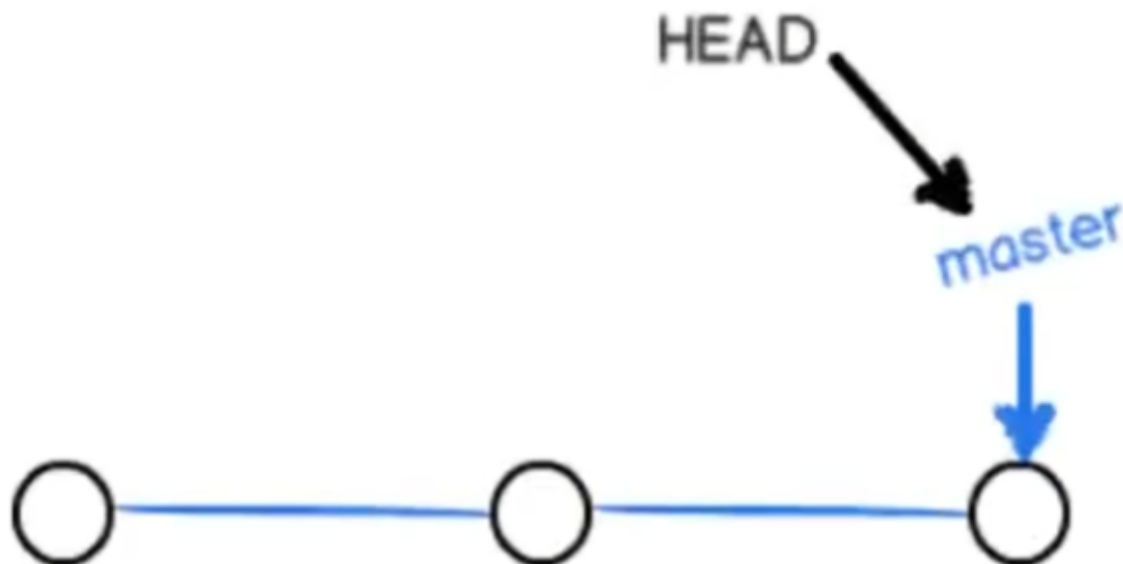
命令 `git rm` 用于删除一个文件。如果一个文件已经被提交到版本库，那么永远不要担心误删，但是要小心，只能回复文件到最新版本，你会丢失**最近一次提交后你修改的内容**。

## 分支管理

## 创建与合并分支

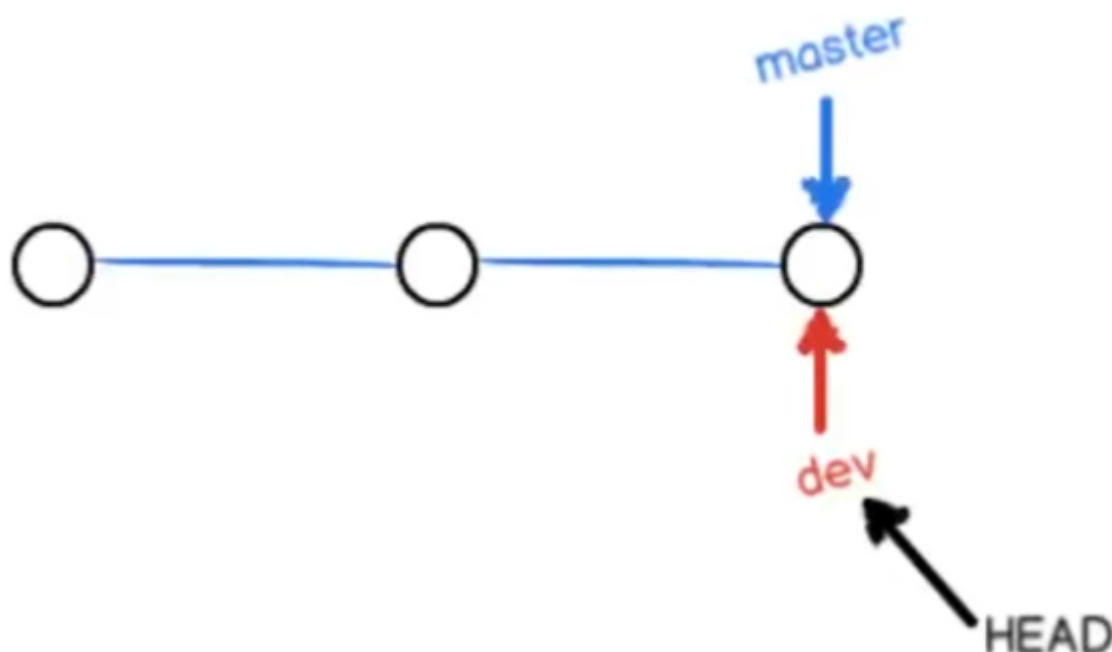
截止到目前只有一条时间线，在 `git` 里，这个分支叫主分支，即 `master` 分支。`HEAD` 严格来说不是指向提交，而是指向 `master`，`master` 才是指向提交的，所以，`HEAD` 指向的就是当前分支。

(1) 一开始的时候，`master` 分支是一条线，`git` 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：



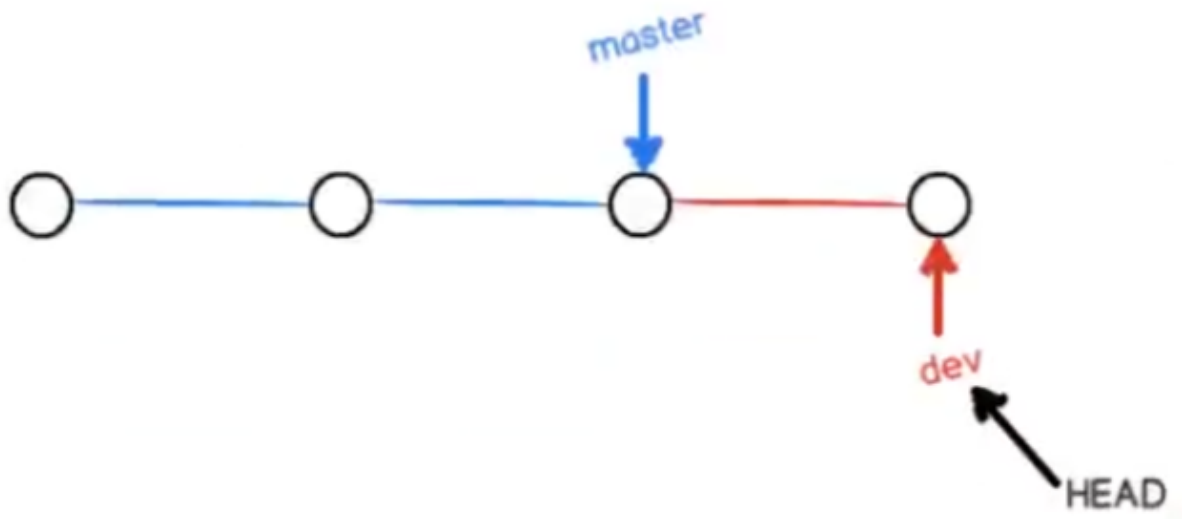
每次提交，`master` 分支都会向前移动一步，随着不断的提交，`master` 分支的线也越来越长。

(2) 当创建新的分支，例如 `dev` 时，`git` 新建了一个指针叫 `dev`，指向 `master` 相同的提交，再把 `HEAD` 指向 `dev`，就表示当前分支在 `dev` 上：

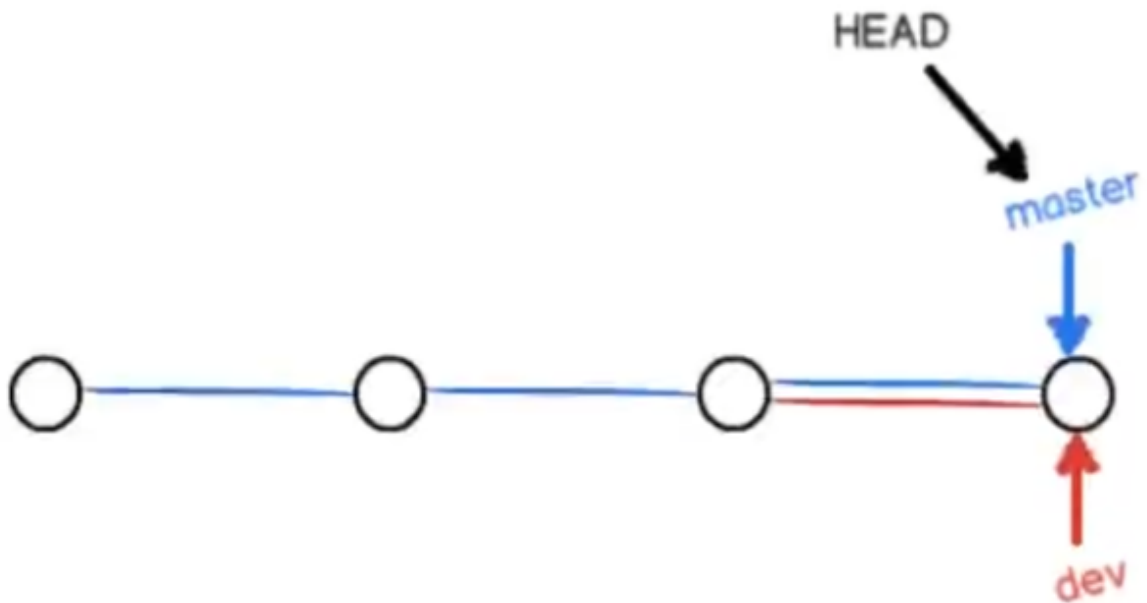


`git` 创建一个分支很快，因为除了增加一个 `dev` 指针，改变 `HEAD` 的指向，工作区的文件都没有任何的变化。

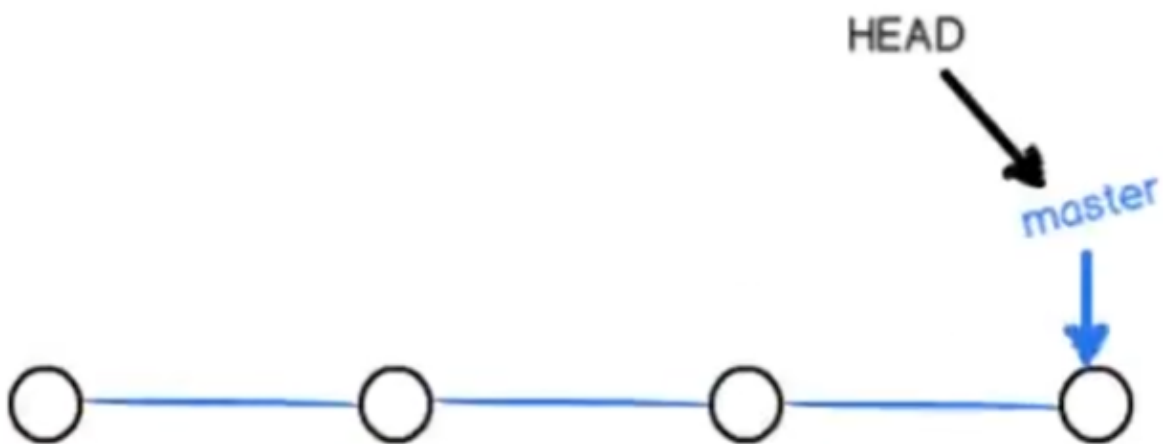
(3) 不过，从现在开始，对工作区的修改和提交就是针对 `dev` 分支了，比如新提交一次后，`dev` 指针往前移动一步，而 `master` 指针不变：



(4) 加入在 `dev` 上的工作完成了，就可以把 `dev` 合并到 `master` 上。`master` 指向 `dev` 的当前的提交，就完成了合并：



(5) 合并完成分支后。甚至可以删除 `dev` 分支。删除 `dev` 分支就是把 `dev` 指针给删掉，删掉后，就剩下一条 `master` 分支：



案例：

(1) 执行如下命令可以查看当前有几个分支并且看到在哪个分支下工作：

```
git branch
```

```
E:\Git_study\GIT>git branch
* master
```

(2) 创建一个分支 `dev` 并切换到其上工作进行工作：

```
git checkout -b dev
```

```
E:\Git_study\GIT>git checkout -b dev
Switched to a new branch 'dev'

E:\Git_study\GIT>git branch
* dev
  master
```

(3) 修改 `code.txt` 内容，在里面添加一行，并进行提交：

```
echo add one line >> code.txt
git add code.txt
git commit -m 'dev分支提交'
git log --pretty=oneline
```

```
E:\Git_study\GIT>echo add one line >> code.txt

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
add one line

E:\Git_study\GIT>git add code.txt

E:\Git_study\GIT>git commit -m 'dev分支提交'
[dev bc01d42] 'dev分支提交'
1 file changed, 1 insertion(+)

E:\Git_study\GIT>git log --pretty=oneline
bc01d42937629150a84e4103733557530def126b (HEAD -> dev) 'dev分支提交'
911f079aa42f59bc1cc95951229ed1e81ff24da7 (master) '删除code2.txt'
07a7b389bb1felfece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

(4) `dev` 分支的工作完成，切换回 `master` 分支：

```
git checkout master
git branch
```

```
E:\Git_study\GIT>git checkout master
Switched to branch 'master'

E:\Git_study\GIT>git branch
  dev
* master
```

查看 `code.txt`，发现添加的内容没有了。因为那个提交是在 `dev` 分支上，而 `master` 分支此刻的提交点并没有变：

(5) 把 `dev` 分支的工作成果合并到 `master` 分支上:

```
git log --pretty=oneline
git merge dev
git log --pretty=oneline
```

```
E:\Git_study\GIT>git log --pretty=oneline
911f079aa42f59bc1cc95951229ed1e81ff24da7 (HEAD -> master) '删除code2.txt'
07a7b389bb1fe1fece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'

E:\Git_study\GIT>git merge dev
Updating 911f079..bc01d42
Fast-forward
 code.txt | 1 +
 1 file changed, 1 insertion(+)

E:\Git_study\GIT>git log --pretty=oneline
bc01d42937629150a84e4103733557530def126b (HEAD -> master, dev) 'dev分支提交'
911f079aa42f59bc1cc95951229ed1e81ff24da7 '删除code2.txt'
07a7b389bb1fe1fece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

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

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。

(6) 合并完成后, 就可以放心的删除 `dev` 分支了, 删除后, 查看 `branch`, 就只剩下 `master` 分支了。

```
git branch -d dev
git branch
```

```
E:\Git_study\GIT>git branch -d dev
Deleted branch dev (was bc01d42).

E:\Git_study\GIT>git branch
* master
```

## 小结

查看分支:

```
git branch
```

创建分支:

```
git branch name
```

切换分支:

```
git checkout name
```

创建并切换分支:

```
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
```

## 解决冲突

(1) 再创建一个新分支 `dev`：

```
E:\Git_study\GIT>git checkout -b dev
Switched to a new branch 'dev'

E:\Git_study\GIT>git branch
* dev
  master
```

(2) 修改 `code.txt` 内容，并进行提交。

```
E:\Git_study\GIT>echo add one more line >> code.txt

E:\Git_study\GIT>git add code.txt

E:\Git_study\GIT>git commit -m 'dev分支提交2'
[dev b045069] 'dev分支提交2'
1 file changed, 1 insertion(+)

E:\Git_study\GIT>git log --pretty=oneline
b045069b4ec36619de600ffb0637f4c62cb016b4 (HEAD -> dev) 'dev分支提交2'
bc01d42937629150a84e4103733557530def126b (master) 'dev分支提交'
911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
07a7b389bb1felfece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

(3) 切换回 `master` 分支。

```
E:\Git_study\GIT>git checkout master
Switched to branch 'master'

E:\Git_study\GIT>git log --pretty=oneline
bc01d42937629150a84e4103733557530def126b (HEAD -> master) 'dev分支提交'
911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
07a7b389bb1felfece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

(4) 在 `master` 的 `code.txt` 添加一行内容并进行提交。



```

E:\Git_study\GIT>echo one more line in master >> code.txt

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
add one line
one more line in master

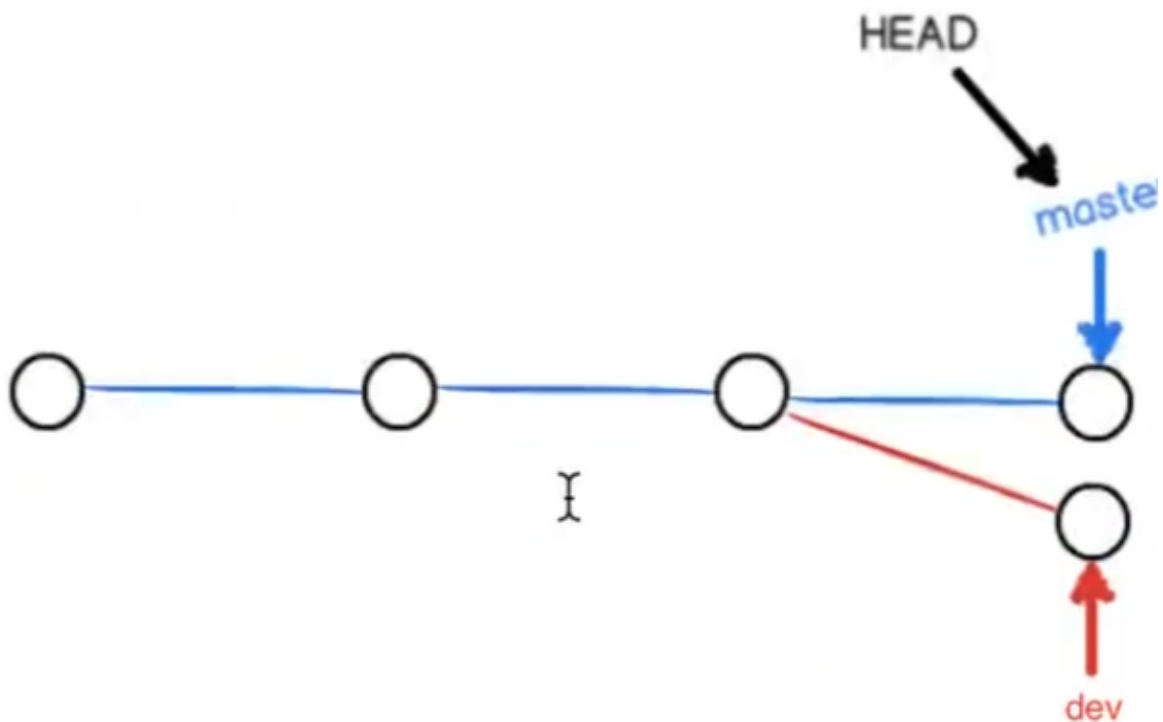
E:\Git_study\GIT>git add code.txt

E:\Git_study\GIT>git commit -m 'master分支提交'
[master 0737e04] 'master分支提交'
1 file changed, 1 insertion(+)

E:\Git_study\GIT>git log --pretty=oneline
0737e04818dc8659e8b01c530d541721e23616f3 (HEAD -> master) 'master分支提交'
bc01d42937629150a84e4103733557530def126b 'dev分支提交'
911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
07a7b389bblfelce31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'

```

现在，`master` 分支和 `dev` 分支各自都分别有新的提交，变成了这样：



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，但是这种合并就可能会有冲突。

(5) 执行如下命令尝试将 `dev` 分支合并到 `master` 分支上来。

```

E:\Git_study\GIT>git merge dev
Auto-merging code.txt
CONFLICT (content): Merge conflict in code.txt
Automatic merge failed; fix conflicts and then commit the result.

```

`git` 告诉我们，`code.txt` 文件存在冲突，必须手动解决冲突后再提交。

(6) `git status` 也可以告诉我们冲突的文件：

```
E:\Git_study\GIT>git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

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

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

(7) 查看 `code.txt` 的内容:

```
E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
add one line
<<<<<<< HEAD
one more line in master
=====
add one more line
>>>>>>> dev
```

(8) `git` 用 `<<<<<<<`, `=====`, `>>>>>>>` 标记出不同分支的内容, 我们修改后如下保存:

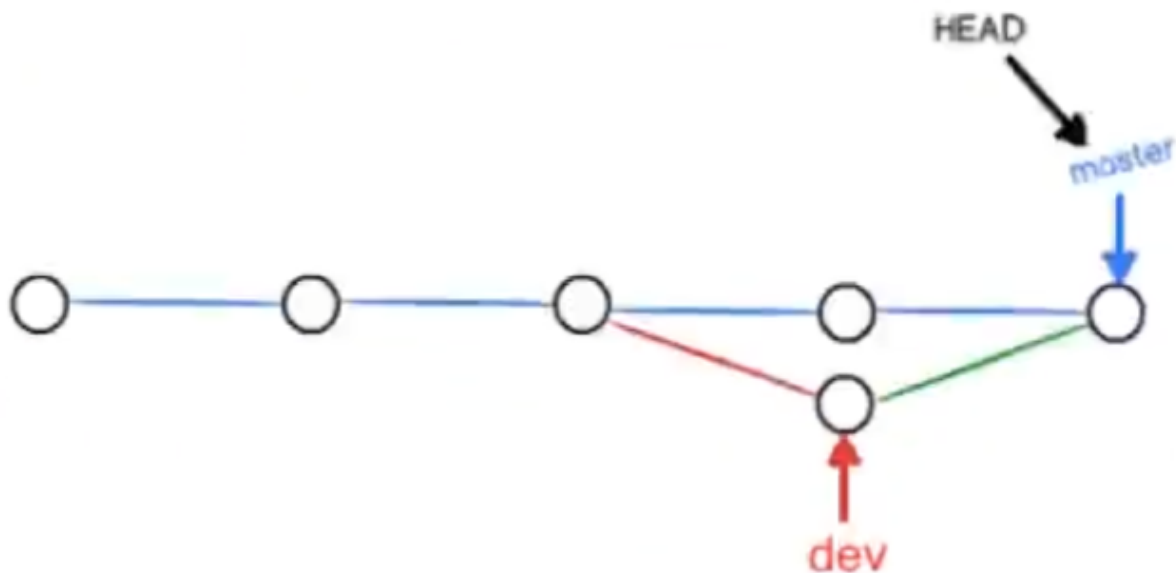
```
E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
add one line
one more line in master
add one more line
```

(9) 再提交。

```
E:\Git_study\GIT>git add code.txt
E:\Git_study\GIT>git commit -m '解决冲突'
[master a8f5696] '解决冲突'

E:\Git_study\GIT>git log --pretty=oneline
a8f569672ecbbc295b86c0b2e1576f65878e9d72 (HEAD -> master) '解决冲突'
0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
b045069b4ec36619de600ffb0637f4c62cb016b4 (dev) 'dev分支提交2'
bc01d42937629150a84e4103733557530def126b 'dev分支提交'
911f079aa42f59bc1cc95951229ed1e81ff24da7 '删除code2.txt'
07a7b389bb1felfece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

(10) 现在, `master` 分支和 `dev` 分支变成了下图所示:



(11) 用带参数的 `git log` 也可以看到分支的合并情况:

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

```
E:\Git_study\GIT>git log --graph --pretty=oneline
* a8f569672ecbbc295b86c0b2e1576f65878e9d72 (HEAD -> master) '解决冲突'
* b045069b4ec36619de600ffb0637f4c62cb016b4 (dev) 'dev分支提交2'
* 0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
* bc01d42937629150a84e4103733557530def126b 'dev分支提交'
* 911f079aa42f59bc1cc95951229ed1e81ff24da7 '删除code2.txt'
* 07a7b389bb1felfece31b8e384a7749035e8550d '版本4'
* 19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
* 3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
* c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

(12) 最后工作完成, 可以删除 `dev` 分支。

```
E:\Git_study\GIT>git branch -d dev
Deleted branch dev (was b045069).
```

## 分支管理策略

通常, 合并分支时, 如果可能, `git` 会用 `fast forward` 模式, 但是有些快速合并不能成而且合并时没有冲突, 这个时候会合并之后做一次新的提交。但这种模式下, 删除分支后, 会丢掉分支信息。

(1) 创建并切换到 `dev` 分支下:

```
E:\Git_study\GIT>git checkout -b dev
Switched to a new branch 'dev'

E:\Git_study\GIT>git branch
* dev
  master

E:\Git_study\GIT>git log --pretty=oneline
a8f569672ecbbc295b86c0b2e1576f65878e9d72 (HEAD -> dev, master) '解决冲突'
0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
bc01d42937629150a84e4103733557530def126b 'dev分支提交'
911f079aa42f59bc1cc95951229ed1e81ff24da7 '删除code2.txt'
07a7b389bb1felfece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

(2) 新建一个文件 `code3.txt`，编辑内容如下，并提交一个 `commit`。

```
E:\Git_study\GIT>echo. >code3.txt
E:\Git_study\GIT>echo one line >> code3.txt
E:\Git_study\GIT>type code3.txt
one line
E:\Git_study\GIT>git add code3.txt
E:\Git_study\GIT>git commit -m '创建文件code3.txt'
[dev 94a376b] '创建文件code3.txt'
1 file changed, 2 insertions(+)
create mode 100644 code3.txt
E:\Git_study\GIT>git log --pretty=oneline
94a376bfaafefa9fe5e0bd26591368315467fd44 (HEAD -> dev) '创建文件code3.txt'
a8f569672ecbbc295b86c0b2e1576f65878e9d72 (master) '解决冲突'
0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
bc01d42937629150a84e4103733557530def126b 'dev分支提交'
911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
07a7b389bb1fe1fece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

(3) 切换回 `master` 分支，编辑 `code.txt` 并进行一个提交。

```
E:\Git_study\GIT>git checkout master
Switched to branch 'master'
E:\Git_study\GIT>git log --pretty=oneline
a8f569672ecbbc295b86c0b2e1576f65878e9d72 (HEAD -> master) '解决冲突'
0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
bc01d42937629150a84e4103733557530def126b 'dev分支提交'
911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
07a7b389bb1fe1fece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
E:\Git_study\GIT>echo one line >> code.txt
E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
add one line
one more line in master
add one more line
one line
E:\Git_study\GIT>git add code.txt
E:\Git_study\GIT>git commit -m '添加新行'
[master fc64255] '添加新行'
1 file changed, 1 insertion(+)
E:\Git_study\GIT>git log --pretty=oneline
fc64255426falea50446f291dd04aeb427dd00c (HEAD -> master) '添加新行'
a8f569672ecbbc295b86c0b2e1576f65878e9d72 '解决冲突'
0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
bc01d42937629150a84e4103733557530def126b 'dev分支提交'
911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
07a7b389bb1fe1fece31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

(4) 合并 `dev` 分支的内容到 `master`。

```

E:\Git_study\GIT>git merge dev
Merge made by the 'ort' strategy.
code3.txt | 2 ++
1 file changed, 2 insertions(+)
create mode 100644 code3.txt

E:\Git_study\GIT>git log --pretty=oneline
b38610ca0ca6eace0acd4897f2e670ef3b924782 (HEAD -> master) Merge branch 'dev'
fc64255426falea50446f291dd04aeb427dd00c '添加新行'
94a376bfaafefa9fe5e0bd26591368315467fd44 (dev) '创建文件code3.txt'
a8f569672ecbbc295b86c0b2e1576f65878e9d72 '解决冲突'
0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
bc01d42937629150a84e4103733557530def126b 'dev分支提交'
911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
07a7b389bb1felfe31b8e384a7749035e8550d '版本4'
19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'

```

(5) 使用分支命令查看分支信息。

```

E:\Git_study\GIT>git log --pretty=oneline --graph
* b38610ca0ca6eace0acd4897f2e670ef3b924782 (HEAD -> master) Merge branch 'dev'
* 94a376bfaafefa9fe5e0bd26591368315467fd44 (dev) '创建文件code3.txt'
* fc64255426falea50446f291dd04aeb427dd00c '添加新行'
* a8f569672ecbbc295b86c0b2e1576f65878e9d72 '解决冲突'
* b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
* 0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
* bc01d42937629150a84e4103733557530def126b 'dev分支提交'
* 911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
* 07a7b389bb1felfe31b8e384a7749035e8550d '版本4'
* 19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
* 3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
* c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'

```

(6) 删除 dev 分支。

```

E:\Git_study\GIT>git branch -d dev
Deleted branch dev (was 94a376b).

```

如果要强制禁用 fast forward 模式，git 就会在 merge 时生成一个新的 commit，这样，从分支历史上就可以看出分支信息。

(1) 创建并切换到分支 dev。

```

E:\Git_study\GIT>git branch -d dev
Deleted branch dev (was 94a376b).

E:\Git_study\GIT>git checkout -b dev
Switched to a new branch 'dev'

E:\Git_study\GIT>git branch
* dev
  master

E:\Git_study\GIT>git log --pretty=oneline --graph
*   b38610ca0ca6eace0acd4897f2e670ef3b924782 (HEAD -> dev, master) Merge branch 'dev'
|
| * 94a376bfaafefa9fe5e0bd26591368315467fd44 '创建文件code3.txt'
| * | fc64255426falea50446f291dd04aeb427dd00c '添加新行'
|
| * a8f569672ecbbc295b86c0b2e1576f65878e9d72 '解决冲突'
|
| * b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
| * | 0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
|
| * bc01d42937629150a84e4103733557530def126b 'dev分支提交'
| * | 911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
| * | 07a7b389bb1felfe31b8e384a7749035e8550d '版本4'
| * | 19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
| * | 3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
| * | c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'

```

(2) 修改 `code.txt` 内容, 并提交一个 `commit`。

```

E:\Git_study\GIT>echo a line >> code.txt

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
add one line
one more line in master
add one more line

one line
a line

E:\Git_study\GIT>git add code.txt

E:\Git_study\GIT>git commit -m '添加新行'
[dev 359a334] '添加新行'
1 file changed, 1 insertion(+)

E:\Git_study\GIT>git log --pretty=oneline --graph
* 359a334b12ec66b053e3f5aafe85acad26e18bbc (HEAD -> dev) '添加新行'
|
| * b38610ca0ca6eace0acd4897f2e670ef3b924782 (master) Merge branch 'dev'
|
| * 94a376bfaafefa9fe5e0bd26591368315467fd44 '创建文件code3.txt'
| * | fc64255426falea50446f291dd04aeb427dd00c '添加新行'
|
| * a8f569672ecbbc295b86c0b2e1576f65878e9d72 '解决冲突'
|
| * b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
| * | 0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
|
| * bc01d42937629150a84e4103733557530def126b 'dev分支提交'
| * | 911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
| * | 07a7b389bb1felfe31b8e384a7749035e8550d '版本4'
| * | 19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
| * | 3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
| * | c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'

```

(3) 切换回 `master` 分支。



```
E:\Git_study\GIT>git checkout master
Switched to branch 'master'

E:\Git_study\GIT>git log --pretty=oneline --graph
* b38610ca0ca6eace0acd4897f2e670ef3b924782 (HEAD -> master) Merge branch 'dev'
* 94a376bfaafefa9fe5e0bd26591368315467fd44 '创建文件code3.txt'
* fc64255426falea50446f291dd04aebc427dd00c '添加新行'
* a8f569672ecbbc295b86c0b2e1576f65878e9d72 '解决冲突'
* b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
* 0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
* bc01d42937629150a84e4103733557530def126b 'dev分支提交'
* 911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
* 07a7b389bb1felfece31b8e384a7749035e8550d '版本4'
* 19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
* 3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
* c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

(4) 准备合并 dev 分支，请注意 `--no-ff` 参数，表示禁用 Fast forward：

```
E:\Git_study\GIT>git merge --no-ff -m '禁用fast-forward' dev
Merge made by the 'ort' strategy.
code.txt | 1 +
1 file changed, 1 insertion(+)

E:\Git_study\GIT>git log --pretty=oneline --graph
* 6485c7d6ed7620fbcfeee266c3650d1e2507b087 (HEAD -> master) '禁用fast-forward'
* 359a334b12ec66b053e3f5aafe85acad26e18bbc (dev) '添加新行'
* b38610ca0ca6eace0acd4897f2e670ef3b924782 Merge branch 'dev'
* 94a376bfaafefa9fe5e0bd26591368315467fd44 '创建文件code3.txt'
* fc64255426falea50446f291dd04aebc427dd00c '添加新行'
* a8f569672ecbbc295b86c0b2e1576f65878e9d72 '解决冲突'
* b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
* 0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
* bc01d42937629150a84e4103733557530def126b 'dev分支提交'
* 911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
* 07a7b389bb1felfece31b8e384a7749035e8550d '版本4'
* 19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
* 3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
* c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'
```

## bug 分支

在 git 中，每个 bug 都可以通过一个新的临时分支来修复，修复后，合并分支，然后将临时分支删除。

(1) 当接到一个修复一个代号 001 的 bug 的任务时，创建一个分支 bug-001 来修复它，但是，当前正在 dev 上进行的工作还没有提交：

```

E:\Git_study\GIT>git branch
* master

E:\Git_study\GIT>git checkout -b dev
Switched to a new branch 'dev'

E:\Git_study\GIT>echo def index(request): >> code3.txt

E:\Git_study\GIT>type code3.txt

one line
def index(request):

E:\Git_study\GIT>git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   code3.txt

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

```

并不是不想提交，而是工作只进行到一半，还没法提交，但是，必须在短时间内修复 bug。

(2) git 还提供了一个 `stash` 功能，可以把当前工作现场“储藏”起来，等以后回复现场后继续工作：

```

E:\Git_study\GIT>git stash
Saved working directory and index state WIP on dev: 6485c7d '禁用fast-forward'

E:\Git_study\GIT>git status
On branch dev
nothing to commit, working tree clean

```

(3) 首先确定要在哪个分支上修复 bug，假定需要在 `master` 分支上修复，就从 `master` 创建临时分支：

```

E:\Git_study\GIT>git checkout master
Switched to branch 'master'

E:\Git_study\GIT>git checkout -b bug-001
Switched to a new branch 'bug-001'

E:\Git_study\GIT>git branch
* bug-001
  dev
  master

```

(4) 现在修复 bug，把 `a line` 删掉，然后提交。



```

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
add one line
one more line in master
add one more line

one line
a line

E:\Git_study\GIT>type code.txt
this is the first line
this is the second line
this is the third line
this is the forth line
add one line
one more line in master
add one more line

one line

E:\Git_study\GIT>git add code.txt

E:\Git_study\GIT>git commit -m '修复bug-001'
[bug-001 b53d701] '修复bug-001'
1 file changed, 1 insertion(+), 1 deletion(-)

```

(5) 修复完成后, 切换到 `master` 分支, 并完成合并, 最后删除 `bug-001` 分支。

```

E:\Git_study\GIT>git log --pretty=oneline --graph
* 6485c7d6ed7620fbcfeee266c3650d1e2507b087 (HEAD -> master, dev) '禁用fast-forward'
* 359a334b12ec66b053e3f5aafe85acad26e18bbc '添加新行'
* b38610ca0ca6eace0acd4897f2e670ef3b924782 Merge branch 'dev'
* 94a376bfaafefa9fe5e0bd26591368315467fd44 '创建文件code3.txt'
* fc64255426falea50446f291dd04aeb427dd00c '添加新行'
* a8f569672ecbbc295b86c0b2e1576f65878e9d72 '解决冲突'
* b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
* 0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
* bc01d42937629150a84e4103733557530def126b 'dev分支提交'
* 911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
* 07a7b389bb1fe1fece31b8e384a7749035e8550d '版本4'
* 19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
* 3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
* c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'

```

```

E:\Git_study\GIT>git merge --no-ff -m '修复bug-001' bug-001
Merge made by the 'ort' strategy.
code.txt | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)

E:\Git_study\GIT>git log --pretty=oneline --graph
* e5ec79c860967f045f4b9ad05017dc0f125cf726 (HEAD -> master) '修复bug-001'
* b53d701de529fbba0771eb9aba440518c12bfbe2 (bug-001) '修复bug-001'
* 6485c7d6ed7620fbcfeee266c3650d1e2507b087 (dev) '禁用fast-forward'
* 359a334b12ec66b053e3f5aafe85acad26e18bbc '添加新行'
* b38610ca0ca6eace0acd4897f2e670ef3b924782 Merge branch 'dev'
* 94a376bfaafefa9fe5e0bd26591368315467fd44 '创建文件code3.txt'
* fc64255426falea50446f291dd04aeb427dd00c '添加新行'
* a8f569672ecbbc295b86c0b2e1576f65878e9d72 '解决冲突'
* b045069b4ec36619de600ffb0637f4c62cb016b4 'dev分支提交2'
* 0737e04818dc8659e8b01c530d541721e23616f3 'master分支提交'
* bc01d42937629150a84e4103733557530def126b 'dev分支提交'
* 911f079aa42f59bclcc95951229ed1e81ff24da7 '删除code2.txt'
* 07a7b389bb1fe1fece31b8e384a7749035e8550d '版本4'
* 19b24d865ba9c7fbb3d06b98aec1b369f5b85763 '版本3'
* 3f9b315f65b5b9b0df16b3a974c073f50a8855de '版本2'
* c3b66d74637c683e9a3e6d7d0742db23ed4b883c '版本1'

```

```
E:\Git_study\GIT>git branch -d bug-001
Deleted branch bug-001 (was b53d701).
```

(6) 现在 `bug-001` 修复完成, 接着回到 `dev` 继续工作。

```
E:\Git_study\GIT>git branch
  dev
* master

E:\Git_study\GIT>git checkout dev
Switched to branch 'dev'

E:\Git_study\GIT>git branch
* dev
  master

E:\Git_study\GIT>git status
On branch dev
nothing to commit, working tree clean
```

(7) 工作区是干净的, 使用 `git stash list` 命令查看:

```
E:\Git_study\GIT>git stash list
stash@{0}: WIP on dev: 6485c7d '禁用fast-forward'
```

工作现场还在, `git` 把 `stash` 内容存在某个地方了, 但是需要恢复一下。

使用 `git stash pop` 命令:

```
E:\Git_study\GIT>git stash pop
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   code3.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (2705054b989c118d3852fbc005a69e5651ce76c8)

E:\Git_study\GIT>git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   code3.txt

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

**小结:**

修复 `bug` 时, 通过创建新的 `bug` 分支进行修复, 然后合并, 最后删除:

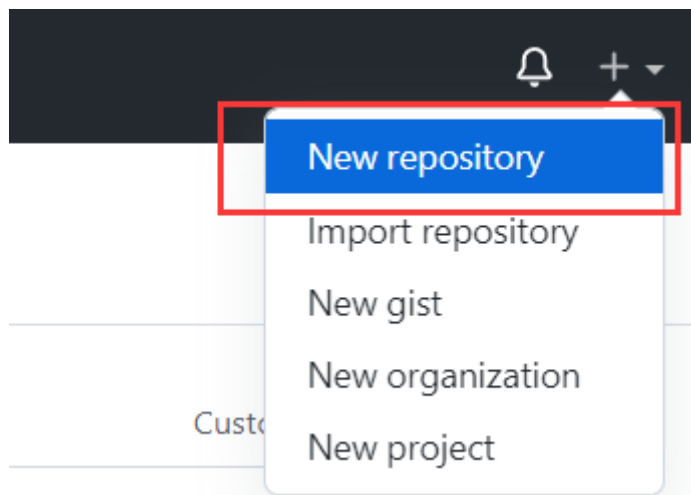
当手头工作没有完成时, 先把工作现场 `git stash` 一下, 然后去修复 `bug`, 修复后, 再 `git stash pop`, 回到工作现场。

## 使用 github

---

## 创建仓库

(1) 注册 `github` 账户，登录后，点击 `New repository`

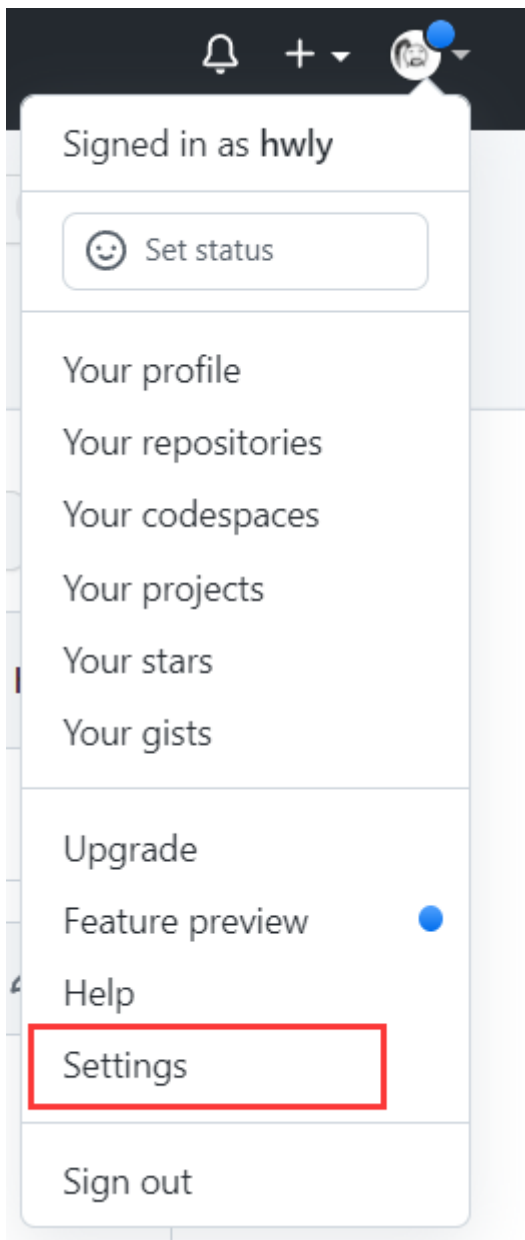


(2) 在新页面中，输入项目名称，勾选 `readme.md`，在 `Add .gitignore` 中选择 `python`，这会忽略掉如 `.pyc` 这种除 `.py` 文件之外的警告。

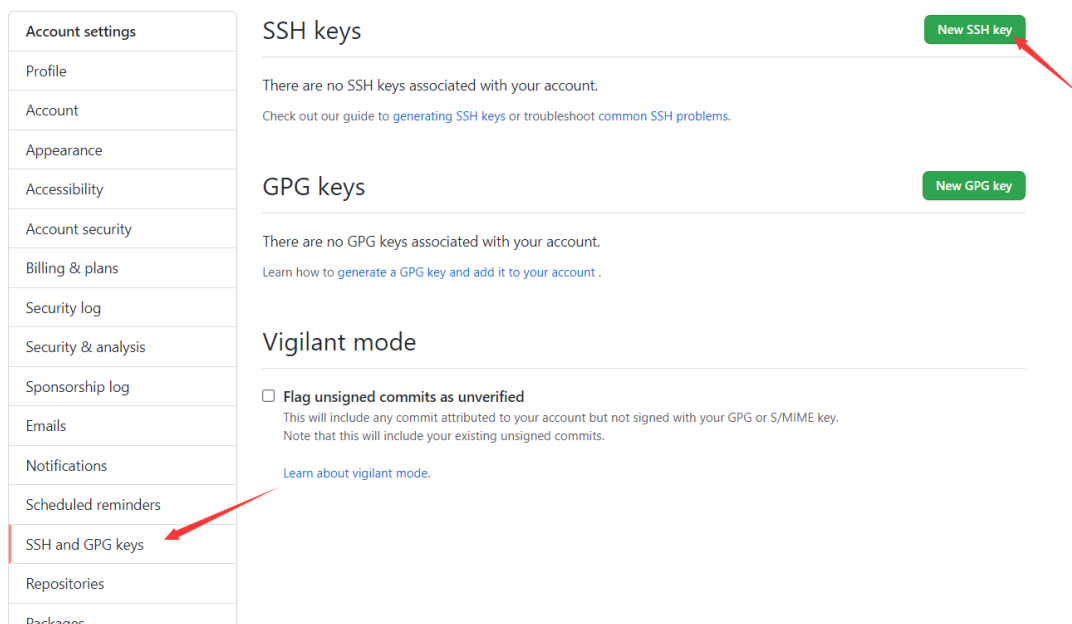
A screenshot of the GitHub 'Create new repository' form. The form is divided into several sections. The first section, 'Owner \*' and 'Repository name \*', is highlighted with a red box; it shows 'hwly' as the owner and 'HuangWly' as the repository name, with a green checkmark next to the name. Below this is a link for repository name inspiration. The second section, 'Description (optional)', is a text input field. The third section, 'Public' and 'Private' options, has 'Public' selected. The fourth section, 'Initialize this repository with:', is also highlighted with a red box; it contains two checked options: 'Add a README file' and 'Add .gitignore'. The '.gitignore' option has a dropdown menu set to 'Python'. Below this is an unchecked option 'Choose a license'. At the bottom, there is a green 'Create repository' button, also highlighted with a red box. The text 'This will set main as the default branch. Change the default name in your settings.' is visible below the license section.

## 添加 ssh 账户

(1) 点击账户头像后的下拉三角，选择 `settings`。如果某台机器需要与 `github` 上的仓库交互，那么就要把这台机器的 `ssh` 公钥添加到这个 `github` 账户上。



点击 `SSH and GPG keys` , 添加 `ssh` 公钥



(2) 回到目录 `C:\Users\H` 下, 编辑 `.gitconfig` , 修改 `git` 配置。

```
1 [user]
2   name = hwly
3   email = 1245470853@qq.com
```

(3) 修改为注册 `github` 时的邮箱，填写用户名。

(4) 使用如下命令生成 `ssh` 密钥。

```
ssh-keygen -t ras -C 注册邮箱
```

(5) 在 `C:\Users\H\.ssh` 目录下有两个文件：

公钥为： `id_rsa.pub`

私钥为： `id_ras`

查看并复制公钥内容到 `github` 的 `key` 中。

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

Add SSH key

## 克隆项目

(1) 在浏览器中点击进入 `github` 首页，再进入项目仓库的页面

Popular repositories

Customize your pins

hello-world

Public

opencv

Forked from opencv/opencv

Open Source Computer Vision Library

C++

openvino

Forked from openvinotoolkit/openvino

OpenVINO™ Toolkit repository

C++

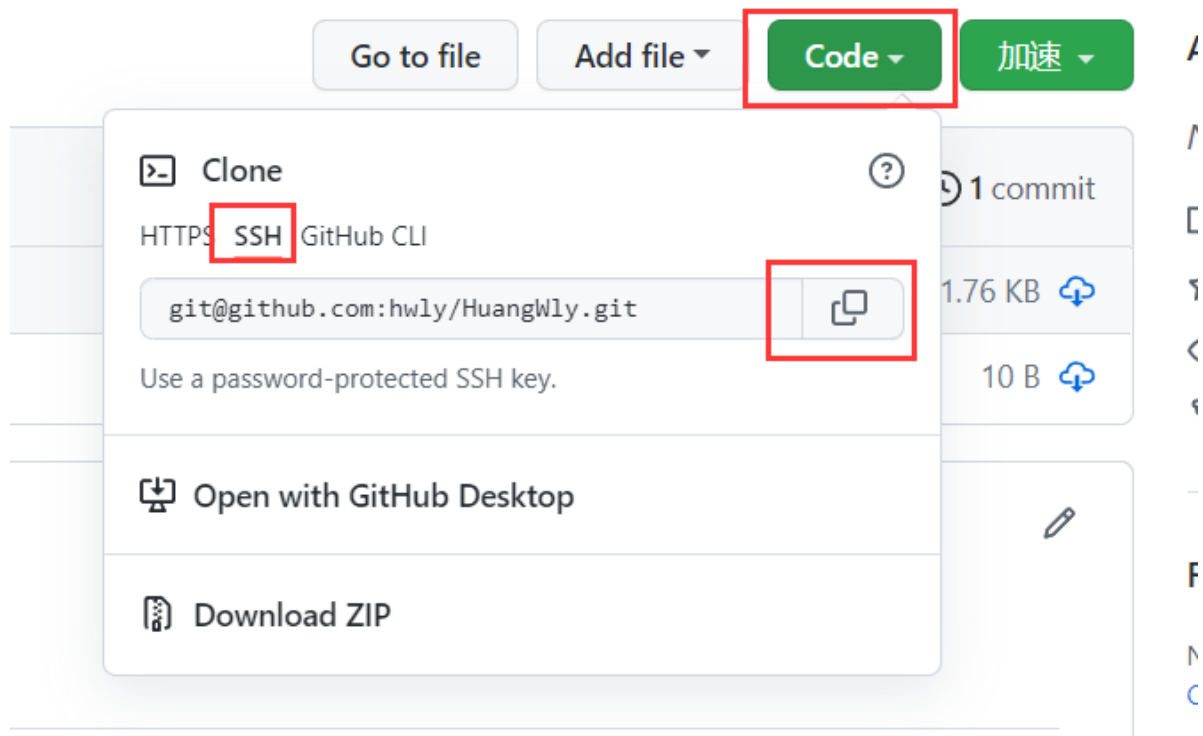
HuangWly

Public

hwly

Config files for my GitHub profile.

(2) 复制 `git` 地址



(3) 克隆出错

输入命令：

```
eval "$(ssh-agent -s)"  
ssh-add
```

(4) 在创建的文件夹下的终端中输入 `git clone ssh地址`，即可克隆：

```
E:\Git_study\CODE>git clone git@github.com:hwly/HuangWly.git  
Cloning into 'HuangWly'...  
remote: Enumerating objects: 4, done.  
remote: Counting objects: 100% (4/4), done.  
remote: Compressing objects: 100% (3/3), done.  
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0  
Receiving objects: 100% (4/4), done.
```

## 创建分支

(1) 项目克隆到本地之后，执行如下命令创建分支 `hwly`。

```
E:\Git_study\CODE>cd HuangWly  
E:\Git_study\CODE\HuangWly>git branch  
* main  
E:\Git_study\CODE\HuangWly>git checkout -b hwly  
Switched to a new branch 'hwly'  
E:\Git_study\CODE\HuangWly>git branch  
* hwly  
  main
```

(2) 创建一个 `view.py`，并提交一个版本。

```

E:\Git_study\CODE\HuangWly>type view.py
from django.http import HttpResponse

def index(request):
    return HttpResponse('index')
E:\Git_study\CODE\HuangWly>git status
On branch hwly
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    view.py

nothing added to commit but untracked files present (use "git add" to track)
E:\Git_study\CODE\HuangWly>git add view.py
E:\Git_study\CODE\HuangWly>git commit -m '创建index视图'
[hwly f5bcfdc] '创建index视图'
1 file changed, 4 insertions(+)
create mode 100644 view.py

```

(3) 推送前 github 上文件列表如图:

The screenshot shows a GitHub repository interface. At the top, there are buttons for 'Go to file', 'Add file', 'Code', and '加速'. Below this, the repository name 'hwly Initial commit' is shown with a commit hash 'ee58c2f' and the time '1 hour ago'. A table lists the files: '.gitignore' (1.76 KB) and 'README.md' (10 B), both marked as 'Initial commit' and '1 hour ago'. Below the file list, the 'README.md' content is displayed, showing the text 'HuangWly'.

(4) 推送前 github 上分支列表如下图:

The screenshot shows the 'Branches' tab in a GitHub repository. It features a search bar 'Search branches...' and tabs for 'Overview', 'Yours', 'Active', 'Stale', and 'All branches'. Under the 'Default branch' section, the 'main' branch is listed as the 'Default' branch, updated '1 hour ago by hwly'.

(5) 推送分支，就是把该分支上的所有本地提交推送到远程库，推送时要指定本地分支，这样，git 就会把该分支推送到远程库对应的远程分支上。

```

git push origin 分支名称
例如:
git push origin hwly

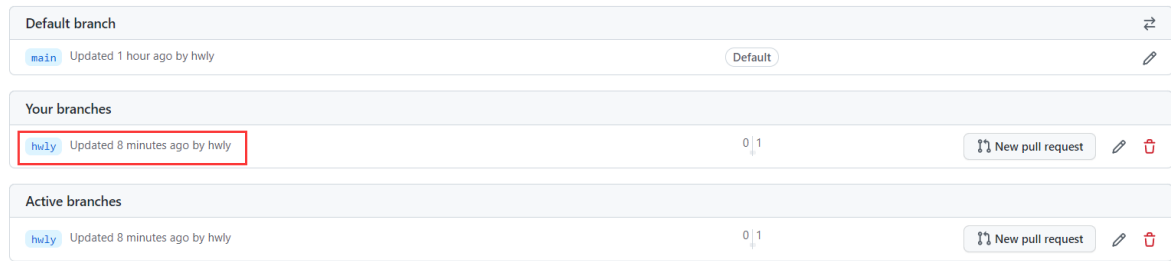
```

```

E:\Git_study\CODE\HuangWly>git push origin hwly
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 398 bytes | 398.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'hwly' on GitHub by visiting:
remote:   https://github.com/hwly/HuangWly/pull/new/hwly
remote:
To github.com:hwly/HuangWly.git
 * [new branch]    hwly -> hwly

```

(6) 再去 `github` 网站上去看分支页面，内容如下：



## 将本地分支跟踪服务器分支

```
git branch --set-upstream-to=origin/远程分支名称 本地分支名称
例如：
git branch --set-upstream-to=origin/hwly hwly
```

```
E:\Git_study\CODE\HuangWly>git branch --set-upstream-to=origin/hwly hwly
Branch 'hwly' set up to track remote branch 'hwly' from 'origin'.
```

(1) 使用 `git status` 查看状态：

```
E:\Git_study\CODE\HuangWly>git status
On branch hwly
Your branch is up to date with 'origin/hwly'.

nothing to commit, working tree clean
```

(2) 编辑文件 `view.py`，并提交：

```
E:\Git_study\CODE\HuangWly>type view.py
from django.http import HttpResponse
from django.shortcuts import redirect

def index(request):
    return HttpResponse('index')

def login(request):
    return redirect('/index')
E:\Git_study\CODE\HuangWly>git add view.py

E:\Git_study\CODE\HuangWly>git commit -m '创建视图login'
[hwly 875e347] '创建视图login'
1 file changed, 5 insertions(+), 1 deletion(-)
```

(3) 通过 `git log` 查看提交：

```
E:\Git_study\CODE\HuangWly>git log
commit 875e34794c0393e377765e536a8634c78950cec5 (HEAD -> hwly)
Author: hwly <1245470853@qq.com>
Date: Sat Dec 11 16:36:35 2021 +0800

    '创建视图login'

commit f5bcfdca2272965f00e5f9da22afbffb0c79b5e3 (origin/hwly)
Author: hwly <1245470853@qq.com>
Date: Sat Dec 11 16:20:05 2021 +0800

    '创建index视图'

commit ee58c2fb5ce3608c02ddf963a535801662da9132 (origin/main, origin/HEAD, main)
Author: hwly <77420345+hwly@users.noreply.github.com>
Date: Sat Dec 11 15:29:45 2021 +0800

    Initial commit
```

(4) 查看状态：

```
E:\Git_study\CODE\HuangWly>git status
On branch hwly
Your branch is ahead of 'origin/hwly' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
```



此时发现本地分支提前远程分支一个版本。

(5) 通过 `git push`，将本地分支内容同步到远程。

```
E:\Git_study\CODE\HuangWly>git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 376 bytes | 376.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:hwly/HuangWly.git
   f5bcfdc..875e347  hwly -> hwly
```

(6) 通过 `git status` 查看状态，版本一致。

```
E:\Git_study\CODE\HuangWly>git status
On branch hwly
Your branch is up to date with 'origin/hwly'.

nothing to commit, working tree clean
```

## 从远程分支上拉取代码

```
git pull origin 分支名称
例如:
git pull origin hwly
```

使用上述命令会把远程分支 `hwly` 上的代码下载并合并到本地所在分支。

```
E:\Git_study\CODE\HuangWly>git pull origin hwly
From github.com:hwly/HuangWly
 * branch          hwly          -> FETCH_HEAD
Already up to date.
```

## 工作使用 git

项目经理：

- (1) 项目经理搭建项目的框架
- (2) 搭建完项目框架之后，项目经理把项目框架代码放到服务器。

普通员工：

- (1) 在自己的电脑上，生成 `ssh` 公钥，然后把公钥给项目经理，项目经理把它添加到服务器上面。
- (2) 项目经理会给每个组员的项目代码的地址，组员把代码下载到自己的电脑上。
- (3) 创建本地的分支 `dev`，在 `dev` 分支中进行每天的开发。
- (4) 每一个员工开发完自己的代码之后，都需要将代码发布到远程的 `dev` 分支上。

**Master：**用于保存发布的项目代码。

**Dev：**保存开发过程中的代码。

