

安装Git

创建版本库

初始化一个Git仓库，使用`git init`命令。

添加文件到Git仓库，分两步：

- 第一步，使用命令`git add <file>`，注意，可反复多次使用，添加多个文件；
- 第二步，使用命令`git commit`，完成。

时光机穿梭

- 要随时掌握工作区的状态，使用`git status`命令。
- 如果`git status`告诉你有文件被修改过，用`git diff`可以查看修改内容。

版本回退

- `HEAD`指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令`git reset --hard commit_id`。
- 穿梭前，用`git log`可以查看提交历史，以便确定要回退到哪个版本。
- 要重返未来，用`git reflog`查看命令历史，以便确定要回到未来的哪个版本。

管理修改

现在，你又理解了Git是如何跟踪修改的，每次修改，如果不`add`到暂存区，那就不会加入到`commit`中

撤销修改

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，用命令`git checkout -- 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，想丢弃修改，分两步，第一步用命令`git reset HEAD 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[版本回退](#)一节，不过前提是没有推送到远程库。

删除文件

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，那么你永远不用担心误删，但是要小心，你只能恢复文件到最新版本，你会丢失最近一次提交后你修改的内容

=====

分支管理

Git鼓励大量使用分支：

查看分支：`git branch`

创建分支：`git branch <name>`

切换分支：`git checkout <name>`

创建+切换分支：`git checkout -b <name>`

合并某分到当前分支：`git merge <name>`

删除分支：`git branch -d <name>`

解决冲突

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

分支管理策略

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，**master**分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在**dev**分支上，也就是说，**dev**分支是不稳定的，到某个时候，比如1.0版本发布时，再把**dev**分支合并到**master**上，在**master**分支发布1.0版本；

你和你的小伙伴们每个人都在**dev**分支上干活，每个人都有自己的分支，时不时地往**dev**分支上合并就可以了。

所以，团队合作的分支看起来就像这样：



例子：

现在，我们切换回**master**：

```
$ git checkout master
Switched to branch 'master'
准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward:

$ git merge --no-ff -m "merge with no-ff" dev
Merge made by the 'recursive' strategy.
 readme.txt | 1 +
 1 file changed, 1 insertion(+)
```

因为本次合并要创建一个新的commit，所以加上`-m`参数，把commit描述写进去。

合并后，我们用`git log`看看分支历史：

```
$ git log --graph --pretty=oneline --abbrev-commit
* 7825a50 merge with no-ff
| \
| * 6224937 add merge
| /
* 59bc1cb conflict fixed
...
```

Bug分支

修复bug时，我们会通过创建新的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，先把手头工作`git stash`一下，然后去修复bug，修复后，再`git stash pop`，回到工作现场

Feature分支

开发一个新feature，最好新建一个分支；

如果要丢弃一个没有被合并过的分支，可以通过`git branch -D <name>`强行删除。

标签管理

创建标签

- 命令`git tag <name>`用于新建一个标签，默认为HEAD，也可以指定一个commit id；
- `git tag -a <tagname> -m "blablabla..."`可以指定标签信息；
- `git tag -s <tagname> -m "blablabla..."`可以用PGP签名标签；
- 命令`git tag`可以查看所有标签。
- 可以用`git show <tagname>`查看标签信息

操作标签

- 命令`git push origin <tagname>`可以推送一个本地标签；
- 命令`git push origin --tags`可以推送全部未推送过的本地标签；
- 命令`git tag -d <tagname>`可以删除一个本地标签；
- 命令`git push origin :refs/tags/<tagname>`可以删除一个远程标签。

远程仓库

添加远程库

从远程库克隆

多人协作

- 查看远程库信息，使用`git remote -v`；
- 本地新建的分支如果不推送到远程，对其他人就是不可见的；
- 从本地推送分支，使用`git push origin branch-name`，如果推送失败，先用`git pull`抓取远程的新提交；

- 在本地创建和远程分支对应的分支，使用`git checkout -b branch-name origin/branch-name`，本地和远程分支的名称最好一致；
- 建立本地分支和远程分支的关联，使用`git branch --set-upstream branch-name origin/branch-name`；
- 从远程抓取分支，使用`git pull`，如果有冲突，要先处理冲突。

