**Git pull 特别慢的解决方案：建议从一开始就解决了，省的以后麻烦：**

<https://blog.csdn.net/kkgbn/article/details/54351787?depth_1->

**遇到Your local changes to the following files would be overwritten by merge:问题的解决方法：**<https://blog.csdn.net/qq_39416311/article/details/88698976>

**如何实现多人同步写代码：**

1. **在建立的库中讲每个人都拉入contributor**
2. **在本地建远程库，并学会push和pull**
3. **开始各自写**
4. **完了一个人push上去了，另一个人再拉就拉不下来，因为他已经修改了本地的文件。**
5. **然后再通过上面说的**

**“遇到Your local changes to the following files would be overwritten by merge:问题的解决方法：**<https://blog.csdn.net/qq_39416311/article/details/88698976>**”里的第一种方法就可以拉下来之后再把我的修改合并上去了**

**创建仓库：**

第一步：

$ mkdir learngit

$ cd learngit

$ pwd//pwd命令用于显示当前目录

第二步，通过git init命令把这个目录变成Git可以管理的仓库：

**添加文件：**

第一步，用命令git add告诉Git，把文件添加到仓库：

第二步，用命令git commit告诉Git，把文件提交到仓库：

简单解释一下git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

**提交文件：**

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。

**回退版本：**

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。

git log --graph命令可以看到分支合并图。

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。

进入log后Q可退出

**文件读取与比较：**

cat filename: 读取文件内容

git diff filename:比较工作区和暂存区

git diff HEAD -- filename:比较工作区和版本库的最新版本

如果git diff输出空白就说明工作区是干净的（干净应该就是指与比较的区相同）

**丢弃当前未add/commit的文件：**

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](https://www.liaoxuefeng.com/wiki/896043488029600/897013573512192)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**。

**关联远程库与push：**

要关联一个远程库，使用命令git remote add origin git@server-name:path/repo-name.git；

关联后，使用命令git push -u origin master第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修改；

**克隆远程库：**

要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆。

Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快。

**使用分支：**

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>或者git switch <name>

创建+切换分支：git checkout -b <name>或者git switch -c <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

**分支策略：**

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；

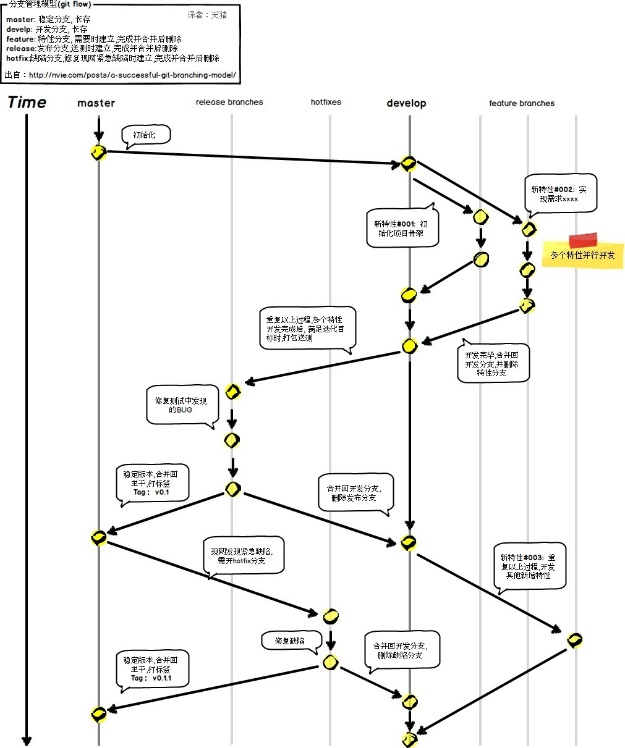
你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

所以，团队合作的分支看起来就像这样：



Git分支十分强大，在团队开发中应该充分应用。

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，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。



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；

在master分支上修复的bug，想要合并到当前dev分支，可以用git cherry-pick <commit>命令，把bug提交的修改“复制”到当前分支，避免重复劳动。

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。

**多人合作：**

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。

**用标签代替字符串：**

命令git tag <tagname>用于新建一个标签，默认为HEAD，也可以指定一个commit id；

命令git tag -a <tagname> -m "blablabla..."可以指定标签信息；

命令git tag可以查看所有标签。

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。