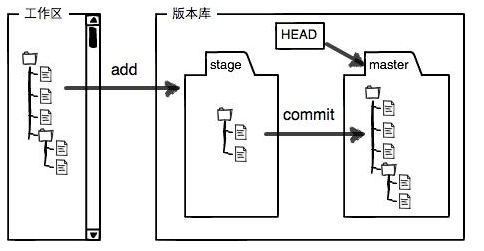
本文档介绍Git使用时常见的场景，以及对应场景下的操作。在接下来的几个场景中体会Git的几组概念：**代码仓和代码分支**；**远程仓和本地仓**；**本地工作区、暂存区和代码仓。**这是使用Git必备的基本概念。

# 概念

## 工作区、暂存区和代码仓

* 工作区就是电脑里能看到的目录。
* 隐藏文件.git叫做版本库，其中包含了暂存区和本地仓。Git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。



git add将修改从工作区添加到暂存区，git commit将修改从暂存区提交到本地仓。

## 本地仓和远程仓

3.1所展示的功能集中式版本控制系统也能做。远程仓才是分布式版本控制系统的精髓。Git是分布式版本控制系统，同一个Git仓库，可以分布到不同的机器上。怎么分布呢？最早，肯定只有一台机器有一个原始版本库，此后，别的机器可以“克隆”这个原始版本库，而且每台机器的版本库其实都是一样的，并没有主次之分。实际情况往往是，找一台电脑充当服务器的角色，每天24小时开机，其他每个人都从这个“服务器”仓库克隆一份到自己的电脑上，并且各自把各自的提交推送到服务器仓库里，也从服务器仓库中拉取别人的提交。分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！当有网络的时候，再把本地提交推送一下就完成了同步。

# 与远程仓交互

## 创建本地仓

### 拉取代码

在当前目录下右键运行git bash，打开命令界面。为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。

* 将代码拉取到当前目录：git clone <远程仓地址>：git clone <https://github.com/Cruise20/git_practice.git>
* 将代码拉取到指定目录：git clone <远程仓地址> <本地路径>。

以上命令将代码克隆下来后，我们只能看到master分支。如果远程仓已经有dev分支了，且我们想修改dev分支的代码，那么就得根据远程dev分支创建一个本地dev分支：git checkout -b dev origin/dev。

### 创建本地仓

进入git\_practice目录，将其作为本地仓：git init。该命令把这个目录变成Git可管理的仓库。当前目录下会增加一个.git的目录，这就是版本库，如果没有看到，那是因为这个目录默认是隐藏的。

### 查看远程仓地址

命令：git remote –v。

### 建立本地仓与远程仓的关联

命令：git remote add <远程仓名字> <远程仓地址>。

示例：git remote add origin <https://github.com/Cruise20/git_practice.git>。其中origin是远程仓的名字，这也是Git默认的叫法，也可以自己任取一个名字。

注意：如果远程仓是Git服务器，可能还需在远程仓下add本地仓所在计算机的ssh key。

若要删除这个关联，把add改成remove。

## 将本地仓的内容推送到远程仓

### 将master分支推送到远程

命令：git push -u origin master。第一次推送时加上-u参数，Git不但会把本地的master分支内容推送到远程的master分支，还会把本地的master分支与远程的master分支关联起来。

### 将dev分支推送到远程

命令：git push origin dev。

### 解决将代码向远程仓推送时发生的冲突

这种情况是由于有人先于自己向远程仓推送了内容。解决办法就是将远程仓内容pull下来，在本地合并、解决冲突再推送。

1. 首先建立远程分支和本地对应分支之间的链接：git branch --set-upstream-to=origin/dev dev。
2. 抓取：git pull <remote-name> <branch-name>。
3. 修改<<<<<<<，=======，>>>>>>>标记着的部分内容，然后git add+git commit，最后git push。

## 多人协作的模式

多人协作的工作模式通常是这样：

1. 首先，可以试图用git push origin <branch-name>推送自己的修改；
2. 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；
3. 如果合并有冲突，则解决冲突，并在本地提交；
4. 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！
5. 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-to <branch-name> origin/<branch-name>。

# 本地开发

## 在某个分支上开发

### 暂存和提交

**所有开发的代码都会放在本地的工作区，开发完成先通过git add放进暂存区，再git commit提交到本地仓。**

命令： git add <代码路径>。将文件修改添加到暂存区。

git commit –m “提交说明” 将暂存区的内容提交到本地仓。

注意：可以一次git add多个文件，然后git commit一次性将它们全部提交。

### 查看提交记录

命令：git log。查看当前分支文件的提交记录。

### 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>。

回退前，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。

注意：回退到上一个版本，<commit-id>可以用HEAD^代替。回退到上上个版本：HEAD^^；回退到上100个版本：HEAD~100。

### 查看本地代码状态

命令： git status。

该命令查看工作区、暂存区和本地仓的状态，可以告诉我们哪些文件是被修改了但还没有提交的。

### 查看代码改动

命令：git diff。

git status只能告诉我们哪些文件被修改了但还没有提交，git diff可以让我们看看具体修改了什么内容。

### 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 -- filename。
* 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 <filename>，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本地仓且还没有推送到远程库，想要撤销本次提交，使用3.1.3“回退版本”。

### 删除文件

如果我们删除了一个文件，工作区和版本库就不一致了，git status命令会立刻告诉你哪些文件被删除了。现在我们有两个选择，一是确实要从版本库中删除该文件，那就用命令git rm filename删掉，并且git commit –m。另一种情况是删错了，好在版本库里还有呢，所以可以很轻松地通过git checkout -- filename把误删的文件恢复到最新版本。

### 保存某个分支上未提交的工作

命令：git stash。使用这个命令之后再用git status查看工作区，发现它是干净的。

## 分支操作

### 创建自己的开发分支并切换到这个新的分支

命令：git checkout –b <branch-name>。效果相当于git branch dev+git checkout dev。注意：-b参数表示根据当前分支的内容创建新的分支并切换分支。如果只是切换分支，去除-b参数。

也可以用这个命令：git switch -c <branch-name>。如果只是切换分支，去除-c参数。

### 删除分支

命令：git branch –d <branch-name>。

### 查看当前分支

命令：git branch。仅查看本地分支。

git branch –a。查看所有分支，当前分支前面会标一个星号。

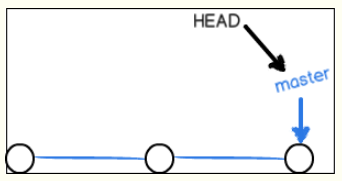
### 在master分支上合并dev分支的修改

命令：git merge <branch-name>。注意：如果添加--no-ff参数就可以用普通模式合并，合并后可以从历史记录中看出来曾经做过合并，而不添加这个参数就是fast forward合并，从历史记录中看不出来曾经做过合并。

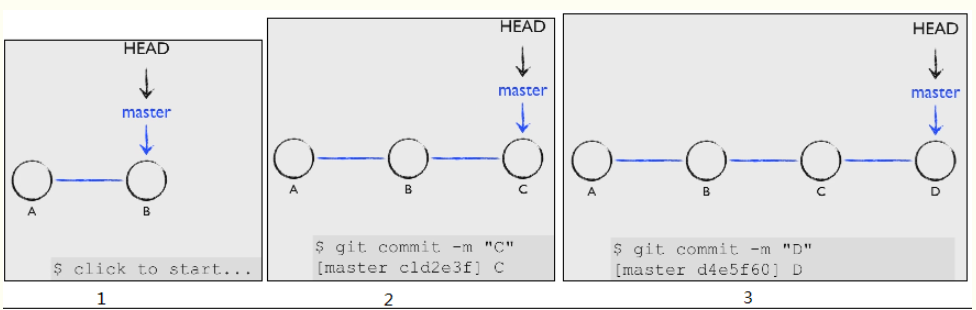
示例：git merge --no-ff -m "merged bug fix 101" bug-101

**以下介绍分支合并的原理！!**

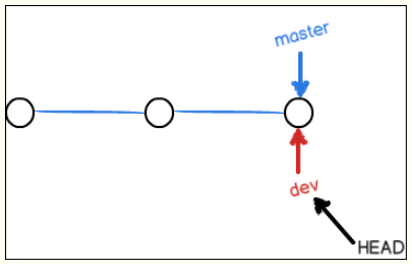
Git将我们的每次提交串成一条时间线，这条时间线就是一个分支，如果只有一条时间线，Git管它叫主分支master。master指针指向最新提交点，HEAD指针指向当前分支的提交点：



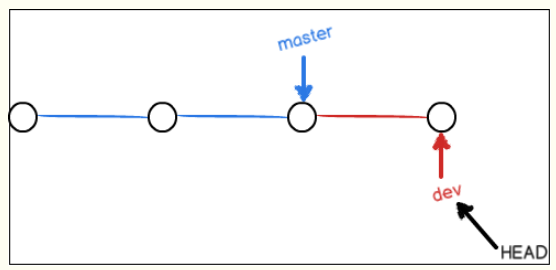
每次提交，master指针都会向前移动一步；随着不断提交，master分支的线也越来越长：



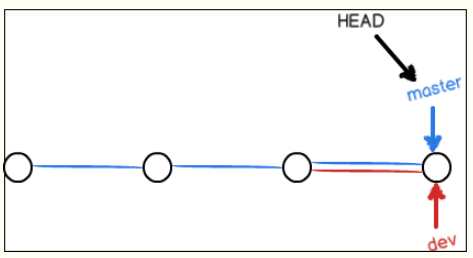
当我们创建新的分支dev时，Git新建了一个分支指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：



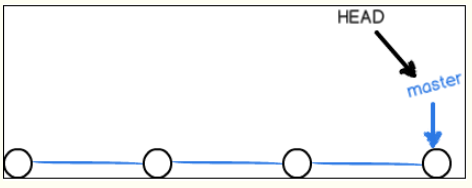
从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变：



假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：



合并完分支后，甚至可以删除dev分支。删除dev分支就是删掉dev指针。之后，我们就剩下了一条master分支：

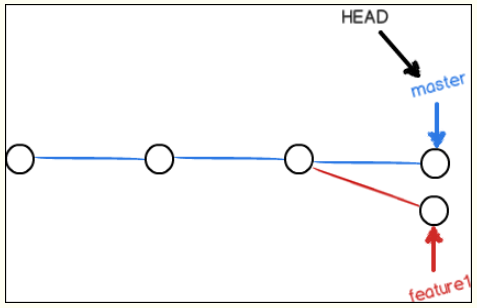


### 解决分支合并时的冲突

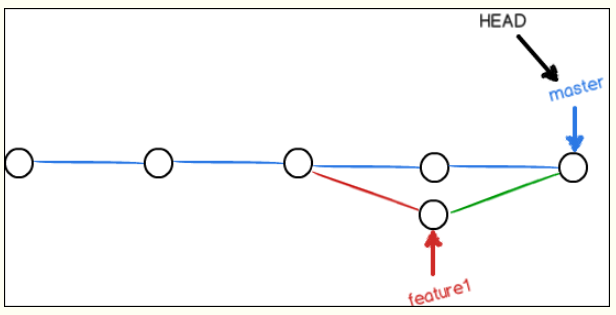
当Git无法自动merge分支时，就必须首先解决冲突。解决冲突后，再git add+git commit，合并就完成了。解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再git add+git commit。

如果master分支和dev分支对同一个文件的同一行代码进行了修改，当两个分支进行合并时，就会出现冲突。但是不要慌，主分支上发生冲突的那个文件用<<<<<<<，=======，>>>>>>>符号将冲突处标记出来，解决冲突的办法就是将标记着的部分内容手动修改为自己想要的样子，再git add+git commit。

冲突：



解决冲突后：



### Bug分支

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。

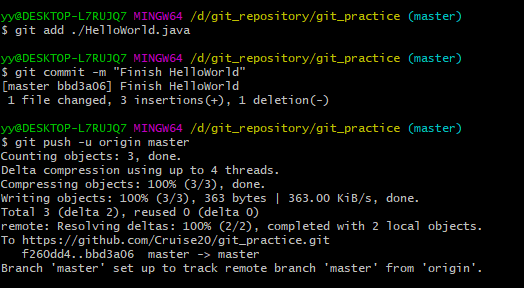
1. 如果在接到修复bug任务时，dev分支上的进行的工作还没完成也没提交，可以用git stash将当前工作现场储藏起来。只有提交工作或储藏工作后，我们才能从dev分支切换到master分支。
2. 然后切换到master分支，并从master分支创建并切换bug分支：git checkout –b bug。
3. 修复完成后，切换到master分支，并完成合并，最后删除bug分支。
4. 接着回dev干活。但是现在工作区是干净的，得把工作现场恢复。首先用git stash list命令查看一下工作现场储藏在哪里了，然后可以用两种办法恢复：git stash apply恢复stash内容但不删除stash内容，可以用git stash drop删除；git stash pop恢复stash内容且删除stash内容。
5. 由于bug分支是从master分支创建出来的，那么这个bug在dev分支上也存在，如何在dev分支上修复同样的bug且避免重复劳动呢？这里我们需要一个将某个提交复制到当前分支的操作：git cherry-pick <commit-id>。

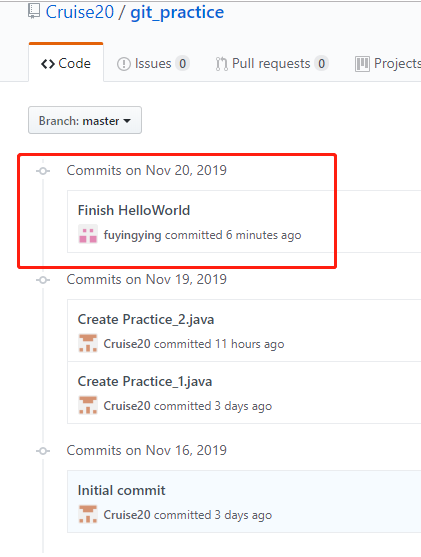
### 强行删除一个分支

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 <branch-name>强行删除。

# 作业1：建立本地与远程的联系

1. 新建D:\git\_repository文件夹，打开git bash，输入git init以建立版本库（.git文件，是否就是本地仓库？）
2. 然后git clone <https://github.com/Cruise20/git_practice.git>
3. 接着修改HelloWorld.java文件，保存。



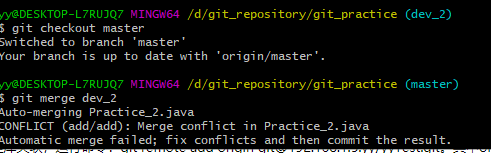


# 作业2：创建分支以及合并分支

1. 创建分支：git checkout –b dev\_1
2. 修改文件并保存，再git add和git commit –m
3. 切换回master分支：git checkout master（注意这里checkout后面不能加-b）
4. 将dev\_1分支合并入master分支：git merge dev\_1
5. 将本地代码推送到远程：git push origin master

# 作业3：回退版本以及解决分支合并的冲突

1. 切换到dev\_2分支：git checkout dev\_2
2. 用git log查看dev\_2分支上的提交记录，发现HEAD指针指向的版本（即当前版本）是dirty，上一个版本是clean，所以回退上一个版本：git reset --hard HEAD^。这时再git log发现HEAD指针指向clean版本。
3. 切换回master分支：git checkout master
4. 将dev\_2分支合并入master分支：git merge dev\_2，但是提示存在冲突：



1. 解决冲突：打开Practice\_2.java文件，将<<<<<<<，=======，>>>>>>>标记出来的区域改成自己想要的样子，保存后再git add+git commit。
2. 最后将本地代码推送到远程：git push origin master