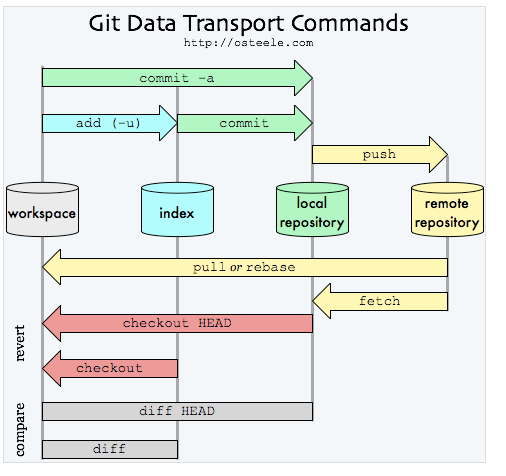
1 原理

（1）原理： 工作空间、本地仓库、远程仓库

（2）常用命令

        a 和本地仓库交互命令：add/commit

        b 和远程仓库交互命令： remote、pull、push、fetch、merg。   除了remote、pull、push、fetch等几个操作，其他大部分工作都是在本地完成的，完成工作后只需要同步远程仓库即可



2 命令说明

（1） 提交

（2） 处理冲突

         已实践，但在解决冲突时的对比工具有待进一步研究

（3） 创建分支（建议只在本地建分支，完成分支合并后可以删除）

         a 分支使用的情景

            个人单独使用本地的分支情况：

            个人新开发一个或者多个功能时，单独拉出一个分支，完成该功能测试后，合并回主分支

            个人处理一些bug时，可单独拉出一个分支，完成修复和验证后，合并回主分支

            团队使用同一个分支情况：

            团队开始一个新的迭代周期（新的迭代周期可以是实现新功能或者修复bug），单独拉出一个分支，迭代完成后合并到主分支，然后删除分支

（4） 分支合并（同一文件存在冲突情况下如何处理）

（5） 还原

          查日志，选择文件的某个版本还原

          查日期，选择还原到某个分支

          待实践如何使用

（6） 标记为已合并

          待实践如何使用

（7） 打标签

3 更新步骤

（1）提交修改的内容到本地仓库

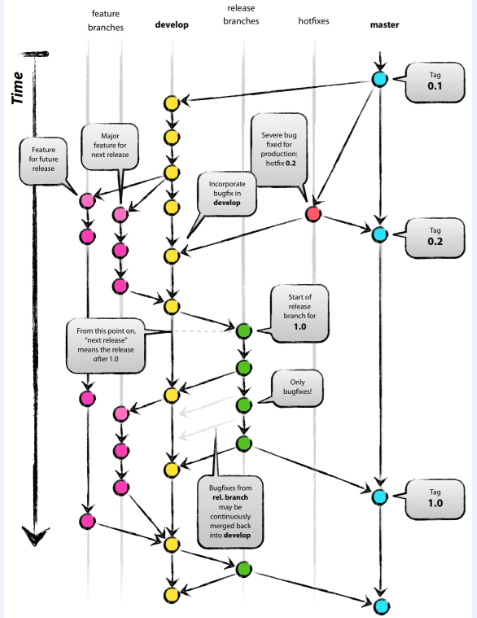
（2）执行拉取pull操作，从远程仓库更新

（3）如果存在冲突，处理冲突后重新提交合并后的文件到本地仓库，然后push到远程仓库

（4）如果没有冲突，直接执行push操作，将修改和新增的内容同步到远程仓库

4 word、excel、原型、图片等不好处理冲突，只能约束统一人员维护

5 分支使用说明



（1） 开始的时候只有master分支和develop分支，两个分支是一样的

（2） 然后开发人员是在develop分支上进行开发，指导觉得可以发布版本时，创建一个release分支，如release\_mmt\_v1.0.0

（3） 然后使用release\_mmt\_v1.0.0分支进行测试并修复bug，期间研发人员不修复任何bug，直至该版本测试完成为止

（4） 开发人员在测试人员测试release\_mmt\_v1.0.0期间可以在develop分支上继续开发新的功能（主要是新功能）

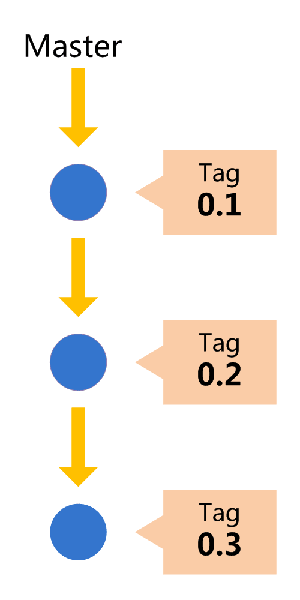
（5） 当release\_mmt\_v1.0.0测试完成时，开始安排开发人员在这个分支上修复bug，直至可以发布为止。

（6） 当release\_mmt\_v1.0.0完成所有bug修复并具备发布条件后，将当release\_mmt\_v1.0.0合并到master分支和develop分支，在master分支上打好tag，然后release\_mmt\_v1.0.0可以删除也可以保留。

（7）当正式环境出现bug时，直接以master分支未基础创建出一个hotfixes分支，然后在hotfixes分支上修复bug，完成修复和测试后，将hotfixes分支合并到master分支和develop分支上，并在master分支上打好tag

**一、主分支Master**

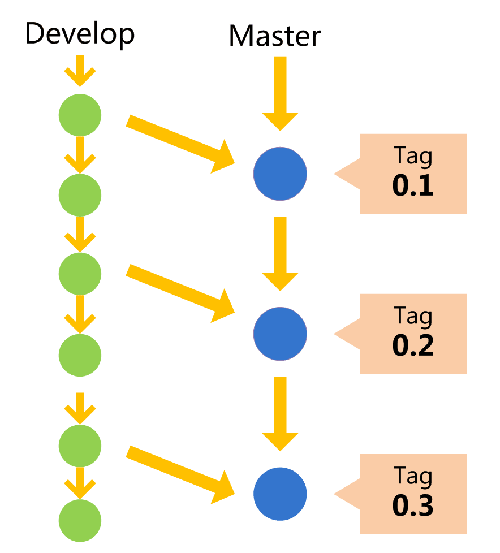
首先，代码库应该有一个、且仅有一个主分支。所有提供给用户使用的正式版本，都在这个主分支上发布。



Git主分支的名字，默认叫做Master。它是自动建立的，版本库初始化以后，默认就是在主分支在进行开发。

**二、开发分支Develop**

主分支只用来分布重大版本，日常开发应该在另一条分支上完成。我们把开发用的分支，叫做Develop。

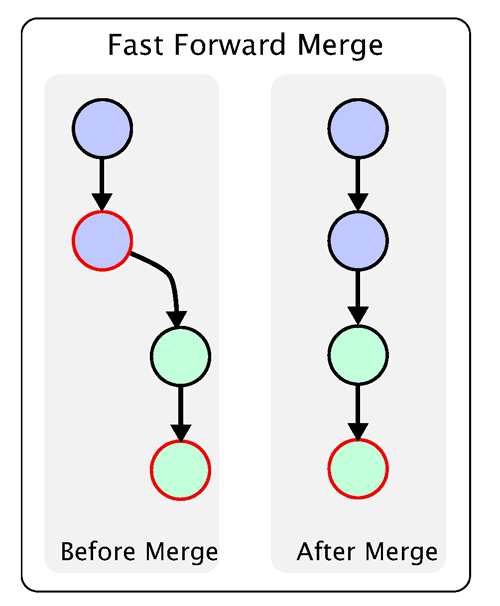


这个分支可以用来生成代码的最新隔夜版本（nightly）。如果想正式对外发布，就在Master分支上，对Develop分支进行"合并"（merge）。

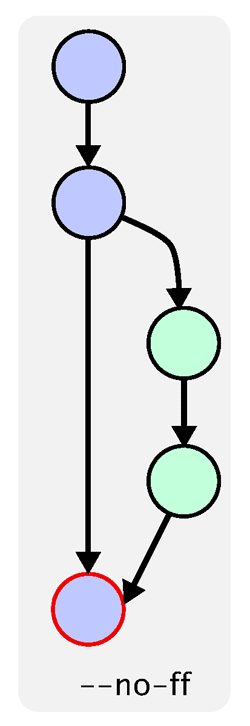
Git创建Develop分支的命令：

将Develop分支发布到Master分支的命令：

这里稍微解释一下，上一条命令的--no-ff参数是什么意思。默认情况下，Git执行"快进式合并"（fast-farward merge），会直接将Master分支指向Develop分支。



使用--no-ff参数后，会执行正常合并，在Master分支上生成一个新节点。为了保证版本演进的清晰，我们希望采用这种做法。关于合并的更多解释，请参考Benjamin Sandofsky的[《Understanding the Git Workflow》](http://sandofsky.com/blog/git-workflow.html)。



**三、临时性分支**

前面讲到版本库的两条主要分支：Master和Develop。前者用于正式发布，后者用于日常开发。其实，常设分支只需要这两条就够了，不需要其他了。

但是，除了常设分支以外，还有一些临时性分支，用于应对一些特定目的的版本开发。临时性分支主要有三种：

　　\* 功能（feature）分支

　　\* 预发布（release）分支

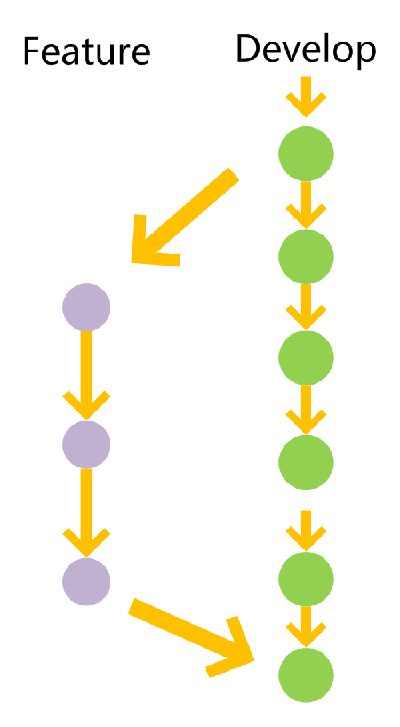
　　\* 修补bug（fixbug）分支

这三种分支都属于临时性需要，使用完以后，应该删除，使得代码库的常设分支始终只有Master和Develop。

**四、 功能分支**

接下来，一个个来看这三种"临时性分支"。

第一种是功能分支，它是为了开发某种特定功能，从Develop分支上面分出来的。开发完成后，要再并入Develop。



功能分支的名字，可以采用feature-\*的形式命名。

创建一个功能分支：

开发完成后，将功能分支合并到develop分支：

删除feature分支：

**五、预发布分支**

第二种是预发布分支，它是指发布正式版本之前（即合并到Master分支之前），我们可能需要有一个预发布的版本进行测试。

预发布分支是从Develop分支上面分出来的，预发布结束以后，必须合并进Develop和Master分支。它的命名，可以采用release-\*的形式。

创建一个预发布分支：

确认没有问题后，合并到master分支：

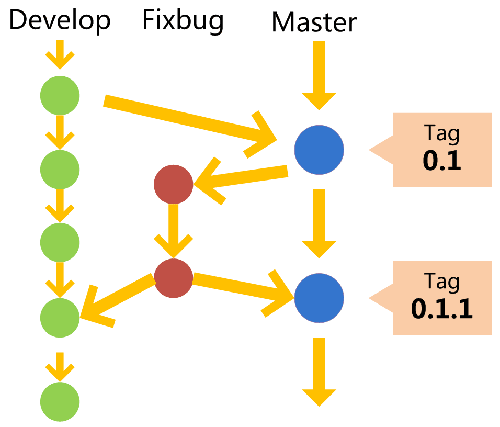
再合并到develop分支：

最后，删除预发布分支：

**六、修补bug分支**

最后一种是修补bug分支。软件正式发布以后，难免会出现bug。这时就需要创建一个分支，进行bug修补。

修补bug分支是从Master分支上面分出来的。修补结束以后，再合并进Master和Develop分支。它的命名，可以采用fixbug-\*的形式。



创建一个修补bug分支：

修补结束后，合并到master分支：

再合并到develop分支：

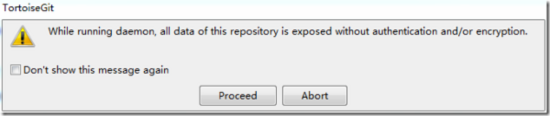
最后，删除"修补bug分支"：

参考资料链接：

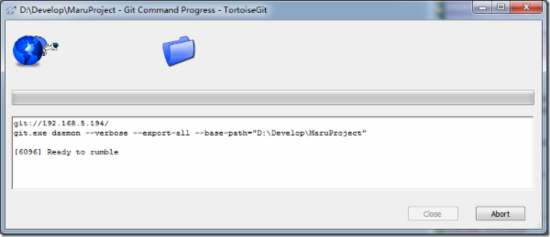
<http://blog.jobbole.com/81196/>

6 TortoiseGit的远程拉取

只需要在一个代码库上右键菜单中选取Daemon即可

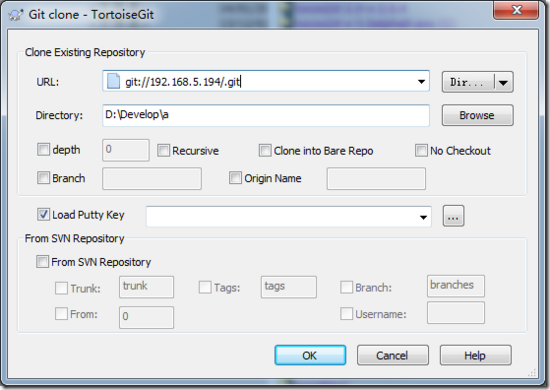


这里的提示告诉你, 这样的拉取无需任何的权限. 其实本地开发, 这不存在, 要的是方便, 所以果断Proceed



本图表示, 你已经对外开放了你指定库的权限

局域网的其他人选择一个空文件夹, 然后选择Clone



表示访问的git的目录!!