# Git基本概念

## 基本概念

### 版本库

### 对象类型

### 索引

## Git工作时的概念

# 文件管理

# 索引

# .gitignore文件

被忽略的文件必须在版本库中明确声明为不可见或被忽略，即使它们可能在你的工作目录中出现。Git维护一个默认忽略文件列表，也可以配置版本库来识别其他文件。

特殊文件.gitignore，只要将想要忽略的文件名添加到同一目录下的.gitignore中即可，此外，可以通过将文件名添加到版本库顶层目录下的.gitignore文件中来忽略它。

一个.gitignore文件可以包含一个文件名模式列表，指定哪些文件要忽略。

**文件格式：**

* 空行要被忽略，而以#开头的行可以用于注释，如果#跟在其他文件后面，它就不表示注释。
* 一个简单的字面置文件名匹配任何目录中的同名文件。
* 目录名由末尾的反斜杠（/）标记。
* 起始的感叹号（！）会对该行其余部分的模式进行取反。此外，被之前模式排除但被取反规则匹配的文件是要包含的。取反模式会覆盖低优先级的规则。

Git允许在版本库中任何目录下有.gitignore文件，每个文件都只影响该目录及其所有子目录。

.gitignore的规则也是级联的：可以覆盖高层目录中的规则，只要在其子目录包含一个取反模式（使用起始的！）

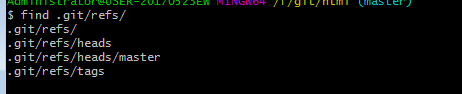
# 引用和符号引用

引用是一个SHA1散列值，指向Git对象库中的对象，虽然一个引用可以指向任何Git对象，但是它通常指向提交对象。

符号引用或称symref，简介指向Git对象，它仍然是一个引用。

本地特性分支名称，远程跟踪分支名称和标签名都是引用。

每一个符号引用都有一个以ref/开头的明确名称，并且都分层存储在版本库的.git/refs/目录中，该目录有三种不同命名空间代表不同的引用：refs/heads/ref代表本地分支，refs/remotes/ref代表远程分支，refs/tags/ref代表标签。



Git自动维护几个用于特定目的的特殊符号引用：

HEAD：始终指向当前分支的最近提交。当切换分支时，HEAD会更新为指向新分支的最近提交。

ORIG\_HEAD：

FETCH\_HEAD：当使用远程库时，git fetch命令将所有抓取分支的头记录到.git/FETCH\_HEAD中。FETCH\_HEAD是最近抓取的分支HEAD的简写，并且仅在刚刚抓取操作之后才有效。使用这个符号引用时，哪怕是一个对没有指定分支名的匿名抓取操作，都可以在

git fetch时找到提交的HEAD。

MERGE\_HEAD：当一个合并操作正在进行时，其他分支的头暂时记录在MERGE\_HEAD中，

MERGE\_HEAD是正在合并进HEAD的提交。

# 分支

分支时在软件项目中启动一条单独的开发线的基本方法，分支从一种统一的，原始的状态分离出来的，是开发能够在多个方向上同时进行，并可能产生项目的不同版本。通常情况下，分支会调节和其他分支合并。

Git允许很多分支，因此一个版本库中可以有许多不同的分支。Git的分支系统是轻量级的，简单的，因此，Git对合并的支持是一流的。

使用分支的原因：（1）一个分支通常代表一个单独的客户发布版

（2）一个分支可以封装一个开发阶段，比如原型，测试，稳定或临近发布。

（3）一个分支可以隔离一个特性的开发或者研究特别复杂的bug。

（4）每一个分支可以代表单个贡献者的工作。

版本库中默认分支命名为master，大多数开发人员在这个分支上保持版本库中最强大和最可靠的开发路线。

分支命名注意事项：（1）可以使用斜杠（/）创建一个分层的命名方案。但是，该分支名不能以斜线结尾。

（2）分支名不能以减号（-）开头。

（3）以斜杠分割的组件不能以点（.）开头。如hao/.sai这样的分支无效

（4）分支名的任何地方不能包含两个连续的点（..）

（5）分支名中不能包含空格或其他空白字符，Git的特殊含义的字符。和ASCLL码控制字符。

在版本库中可能有许多不同的分支，但最多只有一个当前的或活动的分支。活动分支决定工作目录中检出哪些文件。当前分支往往是Git命令中的隐含操作数，默认情况下，master分支作为活动分支，但可以把任何分支设为当前分支。

分支允许版本库中每一个分支的内容向许多不同的方向发散。当一个版本库分出至少一个分支时，把每次提交应用到某个分支，取决于哪个分支时活动的。

每个分支在一个特定的版本库中必须有唯一的名字，这个名字始终指向该分支上最近提交的版本。一个分支的最近提交称为该分支的头部。

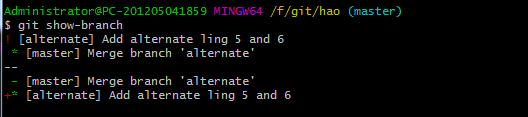
## 创建分支

命令：git branch branch [starting-commit]

如果在创建分支时没有指定starting-commit，则默认认为当前分支的最近提交作为新分支的开始。注意：git branch命令只是把分支名引入版本库中，并没有改变工作目录去使用新的分支，没有工作目录文件发生变化，没有隐式分支环境发生变化，也没有做出新的提交。

命令：git branch列出版本库中的分支名。

git show-branch按时间以递序的形式列出一个或多个分支有贡献的提交，-r显示远程追踪分支，-a显示所有分支。



git show-branch输出被一排破折号分为两部分，**上部分**列出分支名，并用方括号括起来，每行一个，每个分支名跟着一行输出，前面用感叹号或星号标记。上部分的每一个分支都列出该分支最近提交日志消息的一列。**下部分**是一个表示每个分支中提交矩阵。同样，每个提交后面跟着该提交中日志消息的第一行。如果有一个加号（+），星号（\*），或者减号（-）在分支的列中，对应的提交就会在该分支中显示。加号表示提交在一个分支，星号突出显示存在于活动分支的提交，减号表示一个合并分支。如果同一个提交存在于多个分支中，那么每个分支将有一个加号或星号作为标识。

## 检出分支

工作目录一次只能反映一个分支，要在不同的分支上开始工作，要发出git checkout命令，给定一个分支名，git checkout会使该分支变成新的当前分支，它改变了工作树文件和目录结构来匹配给定分支的状态。

选择一个新的当前分支可能会对工作树文件和目录结构产生巨大影响，影响程度的大小取决当前分支和你想检出的目标分支之间的差异。

改变分支的影响有：（1）在要被检出的分支中单不在当前分支中的文件和目录，会从对象库中检出并放置到工作树中。

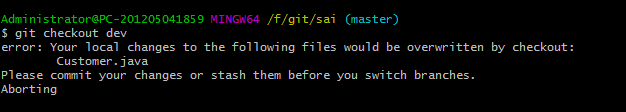
（2）在当前分支中但不在要被检出的分支中的文件和目录，会从工作树中删除。

（3）这两个分支都有的文件会被修改为要被检出的分支的内容。

### 在未提交的更改时进行检出

如果不明确请求，Git会排除本地工作树中数据的删除和修改。工作目录中的未被追踪的文件和目录始终会置之不管，Git不会删除或修改他们，但是，如果一个分支的本地修改不同于新分支上的变更时，Git会报错误信息。

在master分支和dev分支存在相同文件Customer.java，在master分支修改Customer.java后检出dev分支，Git报错：



如果你在dev分支上进行开发时，可能你接到一个紧急任务，对系统出现的bug及时的修复，我们可能会创建一个新的分支，对bug进行修复，但是在dev分支上我们的任务还没有提交，并不是我们不想提交，而是工作才进行一半，还没发进行提交，如果我们检出，则Git会报错，不让我们检出，解决这种问题了有两种方法：第一钟：对当前分支“储藏”。第二种：合并变更到不同的分支。第一种方法稍后详解，第二种方法，我们可以使用 git checkout –m 分支名 命令，Git通过在你的本地修改和对目标分支之间进行一次合并操作，尝试将你的本地修改加入加入到新工作目录中。

# 合并

一次合并会结合两个或多个历史提交，尽管Git还支持同时合并三个，四个或多个分支，但大多数情况下，一次合并只结合两两个分支。在Git中，合并必须在同一个版本库中，当一个分支中的修改与另一个分支中的修改不发生冲突时，Git会计算合并结果，并会创建一个新提交来代表新的统一状态。但是当分支冲突时，Git不解决冲突，这通常出现在对同一个文件的同一行进行修改的时候。相反，git把这种争议性的修改在索引中标记为“未合并”，留给开发人员来处理。

在开始合并之前，最好整理一下工作目录，在正常合并结束的时候，Git会创建新版本的文件并把它们放在工作目录中，并且Git在操作的时候还用索引来存储文件的中间版本。

## 无冲突合并

### 案例一：

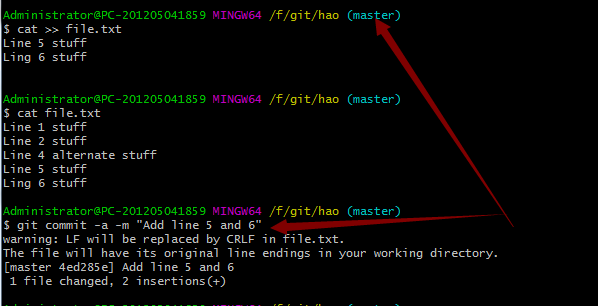
|  |
| --- |
| 第一步：创建版本库    第二步：在master分支上创建文件。        第三步：创建分支    alternate分支是从master^提交初始派生来的，也就是从当前头指针后面的一个提交。  第四步：在alternate分支下修改file.txt    第五步：合并    现在有了两个分支，每个分支都有不同的开发工作，在alternate分支上修改文件，并不影响相同文件的相同部分，所以合并可以顺利进行，不会发生冲突。  使用命令：git log –graph –pretty=oneline –abbrev-commit    两个分支在46417e9处分开，每一个分支显示一个提交（1c80172和a30d130），在849df69处合并。 |

## 有冲突合并

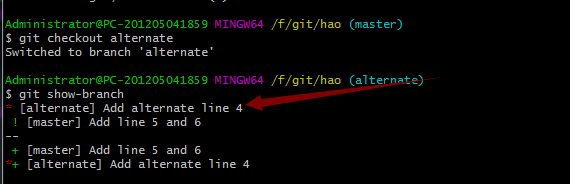
一个分支的修改可能与一个不同分支上的相似或完全不同，修改可能会改变相同文件或无关文件。

继续上面的例子：

在master分支上修改文件file.txt.

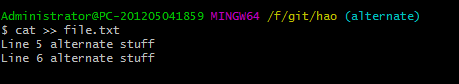


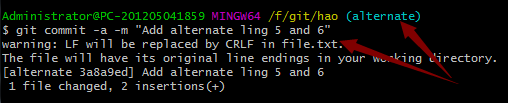
转换到alternate分支上，



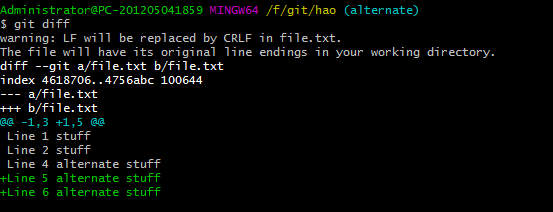
在这个分支上，我们可以看出master分支的提交并没有影响alternate分子，该分支还停在上一次提交时刻。

在alternate分支修改file.txt文件并提交。

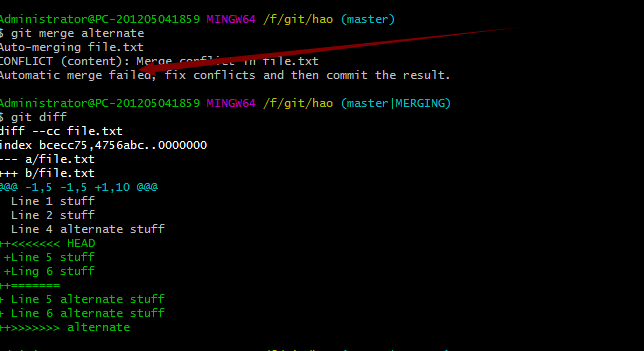




同过命令git diff查看修改前后的差异。



回到master分支，进行合并。

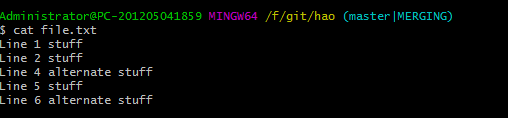


合并失败，发生冲突，使用命令git diff来调查冲突的程度。

git diff命令显示文件在工作目录和索引间的差异，<<<<<<<和===========之间，显示改变内容，==============和>>>>>>>>之间，显示替代的内容。使用额外的加号和减号来表示相对最终版本的来自多个源的变化。

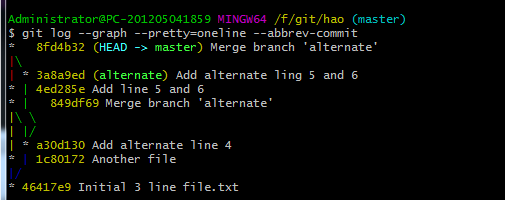
当解决合并冲突时，我们可以自由选择自己喜欢的任何方案，这包括只取一边或另一边的版本，或者两边的混合，甚至全新的内容。

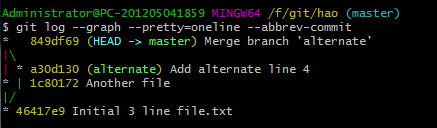
在这里我选择个分支选取一行作为我的解决版本，编译过的文件现如下截图：



如果对自己解决冲突很满意，就可以使用命令git add将文件添加到索引中，并为合并提交而暂存它。

在提交后，使用命令git log –graph –pretty=oneline –abbrev-commit查看提交结果。





# 远程版本库

一个克隆时版本库的副本，一个克隆包含所有原始对象，所以每个克隆都是独立，自治的版本库，与原始版本库时真正对称，地位相同的。

远程版本库是一个引用或句柄，通过文件系统或者网路指向另一个版本库。

Git使用远程追踪分支跟踪其他版本库中的数据。

## 版本库概念

开发版本库：用于常规日常开发。

裸版本库：没有工作目录，并且不应该用于正常开发，没有检出分支的概念。

git clone –bare URL：创建裸版本库。

# 传输协议

Git支持四种不同的传输协议：本地协议，HTTP（S）协议，SSH（Secure Shell）协议和

Git协议。

## 本地协议

本地协议是Git最基本的协议，主要用于本地做一些Git实验。

git clone /opt/soft/public\_html

优点：基于文件仓库的优点在于它的简单，同时保留了现存文件的权限和网络访问权限。

缺点：与基本的网络连接访问相比，难以控制从不同位置来的访问权限。

## SSH协议

安装openssh-server。

SSH协议是一个授权验证的协议

SSH协议是唯一个既支持写操作，有支持读操作的协议。

git clone ssh://GitResosity@qq.com

优点：通过 SSH 进行访问是安全的 — 所有数据传输都是加密和授权的。

缺点：SSH 的限制在于不能通过它实现仓库的匿名访问。

## Git协议

Git协议是一个包含在 Git 软件包中的特殊守护进程； 它会监听一个提供类似于 SSH 服务的特定端口（9418），而无需任何授权。

优点：Git 协议是现存最快的传输协议。

缺点：缺少授权机制。

## HTTP（S）协议