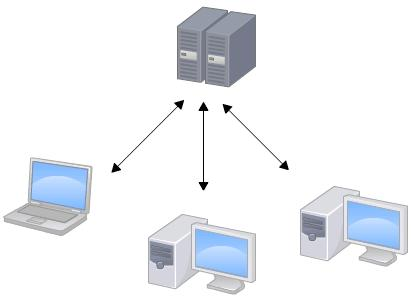
# 版本控制系统

## 集中式版本控制系统（SVN）

1. 集中式版本控制系统，版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了，再放回图书馆。

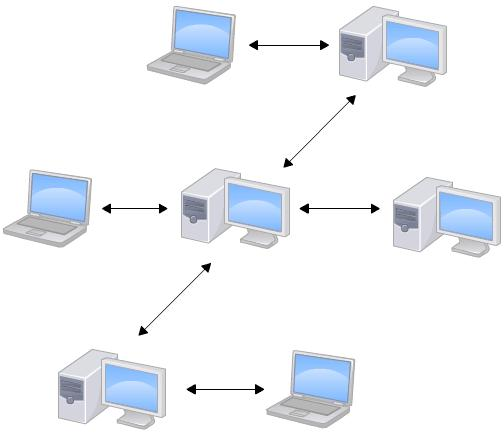


1. 缺点是各台电脑必须联网才能使用，在局域网可以使用，在互联网中速度会很慢。

## 分布式版本控制系统（Git）

1. 基础介绍

**分布式版本控制系统根本没有 “中央服务器”，每个人的电脑上都是一个完整的版本库**，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢？比方说你在自己电脑上改了文件 A，你的同事也在他的电脑上改了文件 A，这时，你们俩之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。



# Git基本流程

## 工作区域



## 基本信息设置

1. 设置用户名

$ git config -- global user.name '林藏宝 '

1. 设置用户邮箱

$ git config -- global user.email 'zangbao.lin@tcl.com'

1. 查看基本信息

$ git config –list

1. git config --global 参数，有了这个参数表示你这台机器上所有的 git 仓库都会使用这个配置，当然你也可以对某个仓库指定不同的用户名和邮箱

## 初始化Git仓库

1. 创建文件夹

$ mkdir test

1. 创建git仓库

$ git init

1. 添加文件

$ touch index.txt （创建新文本文件）

$ git status （查看当前状态）

$ git add ‘index.txt’ （添加文件到暂存区）

$ git commit -m ‘提交文件’ （将文件从暂存区提交到仓库，-m为备注内容）

1. 修改文件

$ vi ‘index.txt’ （进入编辑器，大写ZZ退出）

$ git diff ‘index.txt’ （查看修改内容）

1. 删除文件

$ git rm ‘index.txt’ （删除文件）

$ git commit -m ‘删除文件’ （删除文件完要提交）

## 版本回退

1. 查看提交日志

$ git log --pretty=oneline （获取提交的版本号）

1. 回退版本

$ git reset --hard HEAD^（回退一个版本）

1. 回退到指定版本

$ git reset –hard 版本号

1. 查看命令日志

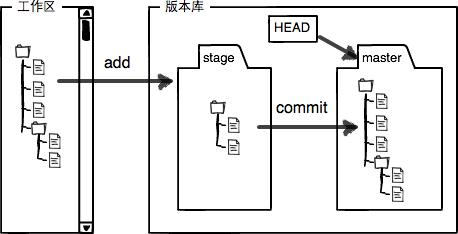
$ git reflog

## 版本库

1. 简介

工作区有一个隐藏目录.git，这个不算工作区，而是 Git 的版本库。

Git 的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



1. git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性把暂存区的所有修改提交到分支。

## 修改以及提交

1. 修改的文件必须 git add添加到暂存区，然后commit到仓库，这样才能完整提交。
2. 查看当前版本和版本库里的最新版本的区别

$ git diff HEAD -- test.txt

1. 撤销工作区的修改

$ git checkout – test.txt （无论是删除还是修改，都可以还原工作区）

一种是在工作区中修改还没提交到暂存区，一种是add到暂存区还没commit到版本库，

撤销修改会回到最近一次add commit的状态。

1. 撤销暂存区的修改

$ git reset HEAD test.txt

## 删除文件

1. 使用rm命令删除文件，那就相当于只删除了工作区的文件，如果想要恢复，直接用

git checkout -- <file> 就可以

1. 用的是 git rm 删除文件，那就相当于不仅删除了文件，而且还添加到了暂存区，

先 git reset HEAD <file>，然后再 git checkout -- <file>

（先还原暂存区，再还原工作区）

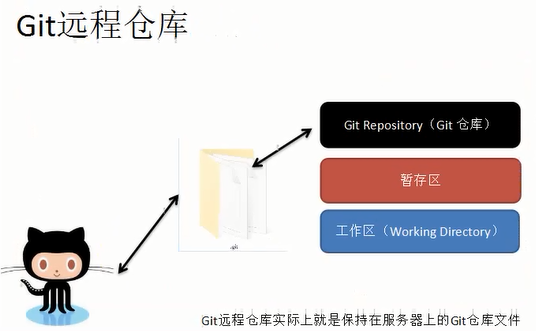
1. 彻底删除版本库内容，先$ git rm 文件 ，再 $ git commit 就可以

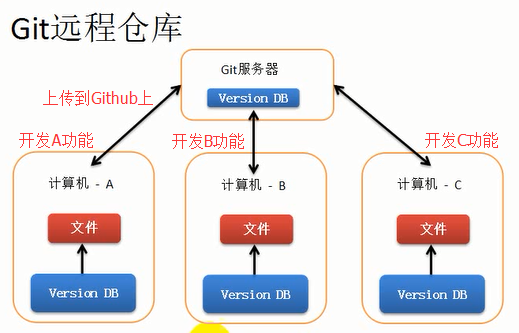
# 远程仓库

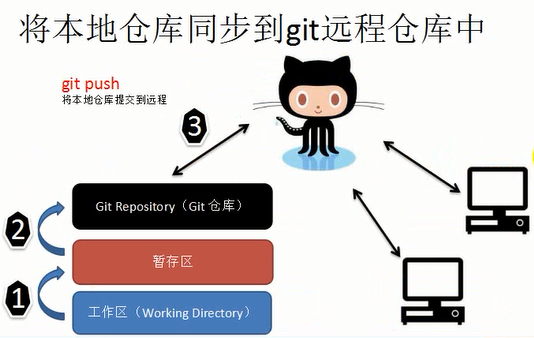
## 基本介绍

使用远程仓库的目的是备份、实现代码共享集中化管理。

Git 远程仓库实际上就是保持在服务器上的 git 仓库文件。







## 添加远程仓库

1. 新建SSH密匙

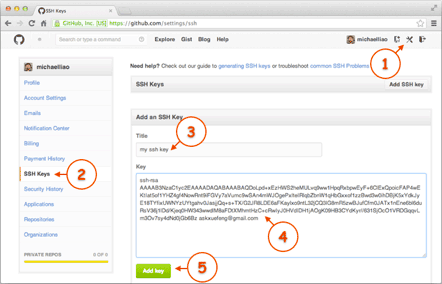
由于你的本地 Git 仓库和 GitHub 仓库之间的传输是通过 SSH 加密的,所以需要预先设置远程仓库。

$ ssh-keygen -t rsa -C [repairsin@163.com](mailto:repairsin@163.com)

如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是 SSH Key 的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

1. 绑定SSH到GitHub（将公匙给GitHub ）

Setting里面，点 “Add SSH Key”，填上任意 Title，在 Key 文本框里粘贴id\_rsa.pub文件的内容



1. 克隆远程仓库

$ git clone 仓库地址

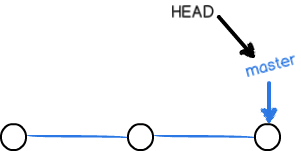
# 分支管理

## 创建和合并分支

1. 简介

在 Git 里，这个分支叫主分支，即master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。

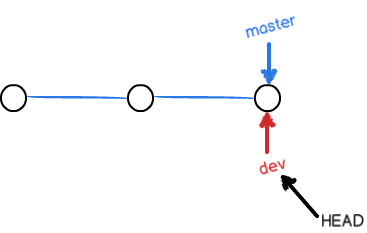
一开始的时候，master分支是一条线，Git 用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点。



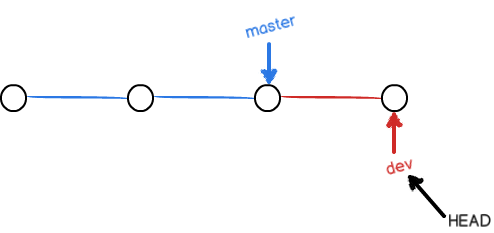
HEAD表示当前分支

1. 创建新的分支

创建新的分支，例如dev时，Git 新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev。

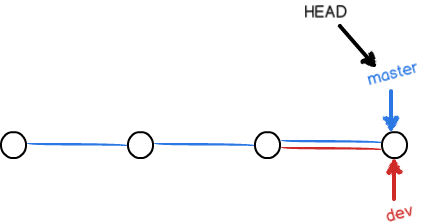


对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变



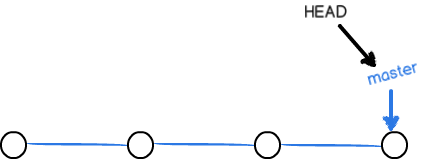
1. 合并分支

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。将master指针指向dev当前位置就可以了。



1. 删除分支

合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支



1. 具体命令

**创建分支**

$ git checkout -b dev

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

$ git branch dev

$ git checkout dev

**查看分支**

$ git branch （git branch命令会列出所有分支，当前分支前面会标一个\*号。）

**合并分支**

$ git merge dev（用于合并指定分支到当前分支）

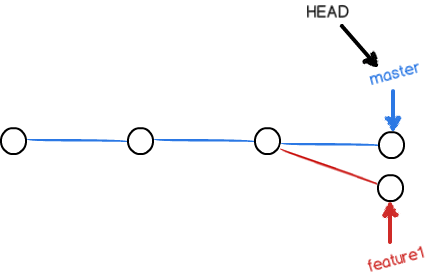
**删除分支**

$ git branch -d dev



## 解决冲突

1. 冲突背景



当两个分支同时修改了一个文件，且修改内容不一样的时候。

如果修改的地方不同，git会尝试将两个文件合并，如果修改的地方一样，git会报错，并将冲突信息反馈。

1. 解决方法

手动修改产生冲突的文件，并且重新提交一份不会冲突的文件。

1. 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。

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，再提交。

用git log --graph命令可以看到分支合并图。

## 分支管理

1. 分支合并策略

通常，合并分支时，如果可能，Git 会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。

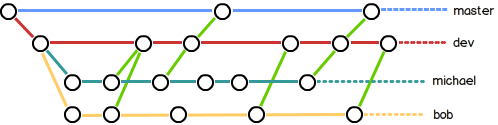
如果要强制禁用Fast forward模式，Git 就会在 merge 时生成一个新的 commit，这样，从分支历史上就可以看出分支信息。

1. 分支工作模式

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。

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



可以理解为：master为稳定测试环境，dev为开发环境

1. 合并分支（默认的合并是fast forward模式，没有日志信息）

-no-ff 合并模式会显示合并的历史记录，可以用git log读取出来

$ git merge --no-ff -m "merge with no-ff" dev

## Bug分支

1. 隐藏当前分支的当前工作区

$ git stash（隐藏工作区到stash）

$ git stash list（查看所有隐藏到stash的工作现场）

$ git stash apply （恢复工作区，但是需要$ git stash drop删除stash内容）

$ git stash pop（恢复同时并且删除）

$ git stash apply stash@{0}（恢复指定的stash）

## 新功能分支

1. 开发一个新 feature，最好新建一个分支；
2. 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。

## 多人协作

当你从远程仓库克隆时，实际上 Git 自动把本地的master分支和远程的master分支对应起来了，并且，远程仓库的默认名称是origin

1. 查看远程库信息

$ git remote

$ git remote -v （显示更详细信息）

origin git@github.com:michaelliao/learngit.git (fetch) （抓取地址）

origin git@github.com:michaelliao/learngit.git (push) （推送地址）

* 1. 推送分支

推送分支，就是把该分支上的所有本地提交推送到远程库。推送时，要指定本地分支，这样，Git 就会把该分支推送到远程库对应的远程分支上：

$ git push origin master

$ git push origin dev

（不一定要全部推送分支）

1. master分支是主分支，因此要时刻与远程同步；
2. dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；
3. bug 分支只用于在本地修复 bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个 bug；
4. feature 分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。
   1. 多人协作的开发模式
5. 首先，可以试图用git push origin <branch-name>推送自己的修改；

（本地新建的分支不推送到远程，则不可见）

1. 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；
2. 如果合并有冲突，则解决冲突，并在本地提交；
3. 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！
4. 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>。