### Git学习

学习网址：<https://www.liaoxuefeng.com/wiki/896043488029600>

Git基础知识

什么是版本控制

版本控制（Revision control）是一种在开发的过程中用于管理我们对文件、目录或工程等内容的修改历史，方便查看更改历史记录，备份以便恢复以前的版本的软件工程技术。

1.实现跨区域多人协同开发

2.追踪和记载一个或者多个文件的历史记录

3.组织和保护你的源代码和文档

4.统计工作量

5.并行开发、提高开发效率

6.跟踪记录整个软件的开发过程

7.减轻开发人员的负担，节省时间，同时降低人为错误

简单说就是用于管理多人协同开发项目的技术。

版本控制分类

1、本地版本控制

记录文件每次的更新，可以对每个版本做一个快照，或是记录补丁文件，适合个人用，如RCS。

2、集中版本控制  SVN

所有的版本数据都保存在服务器上，协同开发者从服务器上同步更新或上传自己的修改

所有的版本数据都存在服务器上，用户的本地只有自己以前所同步的版本，如果不连网的话，用户就看不到历史版本，也无法切换版本验证问题，或在不同分支工作。而且，所有数据都保存在单一的服务器上，有很大的风险这个服务器会损坏，这样就会丢失所有的数据，当然可以定期备份。代表产品：SVN、CVS、VSS

3、分布式版本控制 Git

每个人都拥有全部的代码！安全隐患！

所有版本信息仓库全部同步到本地的每个用户，这样就可以在本地查看所有版本历史，可以离线在本地提交，只需在连网时push到相应的服务器或其他用户那里。由于每个用户那里保存的都是所有的版本数据，只要有一个用户的设备没有问题就可以恢复所有的数据，但这增加了本地存储空间的占用。

不会因为服务器损坏或者网络问题，造成不能工作的情况！

Git与SVN的主要区别

SVN是集中式版本控制系统，版本库是集中放在中央服务器的，而工作的时候，用的都是自己的电脑，所以首先要从中央服务器得到最新的版本，然后工作，完成工作后，需要把自己做完的活推送到中央服务器。集中式版本控制系统是必须联网才能工作，对网络带宽要求较高。

Git是分布式版本控制系统，没有中央服务器，每个人的电脑就是一个完整的版本库，工作的时候不需要联网了，因为版本都在自己电脑上。协同的方法是这样的：比如说自己在电脑上改了文件A，其他人也在电脑上改了文件A，这时，你们两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可以直接看到更新了哪些代码和文件！

Git的安装和配置

打开 [git官网] <https://git-scm.com/，下载git对应操作系统的版本。>

安装完成后会有三个部分：

Git Bash：Unix与Linux风格的命令行，使用最多，推荐最多

Git CMD：Windows风格的命令行

Git GUI：图形界面的Git

安装完成后设置用户名与邮箱（用户标识，必要）

当你安装Git后首先要做的事情是设置你的用户名称和e-mail地址。这是非常重要的，因为每次Git提交都会使用该信息。它被永远的嵌入到了你的提交中：

git config --global user.name "name" #名称

git config --global user.email”email” #邮箱

只需要做一次这个设置，如果你传递了--global 选项，因为Git将总是会使用该信息来处理你在系统中所做的一切操作。如果你希望在一个特定的项目中使用不同的名称或e-mail地址，你可以在该项目中运行该命令而不要--global选项。总之--global为全局配置，不加为某个项目的特定配置。

Git config --global --list #查看用户名和邮箱

Git相关的配置文件：

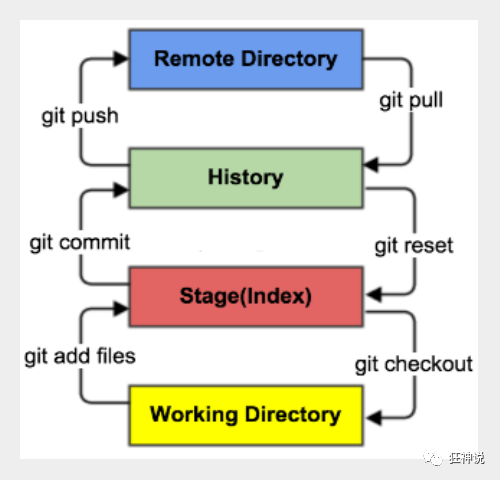
1）、Git\etc\gitconfig  ：Git 安装目录下的 gitconfig     --system 系统级

2）、C:\Users\Administrator\ .gitconfig    只适用于当前登录用户的配置  --global 全局

Git基本理论（重要）

三个区域

Git本地有三个工作区域：工作目录（Working Directory）、暂存区(Stage/Index)、资源库(Repository或Git Directory)。如果在加上远程的git仓库(Remote Directory)就可以分为四个工作区域。文件在这四个区域之间的转换关系如下：



Workspace：工作区，就是你平时存放项目代码的地方

Index / Stage：暂存区，用于临时存放你的改动，事实上它只是一个文件，保存即将提交到文件列表信息

Repository：仓库区（或本地仓库），就是安全存放数据的位置，这里面有你提交到所有版本的数据。其中HEAD指向最新放入仓库的版本

Remote：远程仓库，托管代码的服务器，可以简单的认为是你项目组中的一台电脑用于远程数据交换

本地的三个区域确切的说应该是git仓库中HEAD指向的版本：

Directory：使用Git管理的一个目录，也就是一个仓库，包含我们的工作空间和Git的管理空间。

WorkSpace：需要通过Git进行版本控制的目录和文件，这些目录和文件组成了工作空间。

.git：存放Git管理信息的目录，初始化仓库的时候自动创建。

Index/Stage：暂存区，或者叫待提交更新区，在提交进入repo之前，我们可以把所有的更新放在暂存区。

Local Repo：本地仓库，一个存放在本地的版本库；HEAD会只是当前的开发分支（branch）。

Stash：隐藏，是一个工作状态保存栈，用于保存/恢复WorkSpace中的临时状态。

工作流程

git的工作流程一般是这样的：

１、在工作目录中添加、修改文件；

２、将需要进行版本管理的文件放入暂存区域；

３、将暂存区域的文件提交到git仓库。

因此，git管理的文件有三种状态：已修改（modified）,已暂存（staged）,已提交(committed)

git add files 把文件的修改添加到暂存区

git commit 把暂存区的修改提交到当前分支，提交之后暂存区就被清空了

git reset -- files 使用当前分支上的修改覆盖暂存区，用来撤销最后一次 git add files

git checkout -- files 使用暂存区的修改覆盖工作目录，用来撤销本地修改

可以跳过暂存区域直接从分支中取出修改，或者直接提交修改到分支中。

git commit -a 直接把所有文件的修改添加到暂存区然后执行提交

git checkout HEAD -- files 取出最后一次修改，可以用来进行回滚操作

Git常用命令

创建克隆远程/本地仓库

git init 初始化一个目录为可管理的仓库

git remote add origin git@server-name:path/repo-name.git 关联远程仓库

git push -u origin master 本地master分支推送到远程库 -u 第一次推送关联用

git push origin master 之后可以简化命令，origin 远程主机名 master 本地分支

git clone 链接 克隆远程仓库到本地

基本操作

git add 文件 把文件的修改添加到暂存区

git rm 提交本次删除操作到暂存区

git commit -m 把暂存区的修改提交到当前分支，提交之后暂存区就被清空了

-m 后面输入的是本次提交的说明，可以输入任意内容

git status 查看当前状态

git diff 文件名 查看工作区和暂存区差异

--cached 查看暂存区和仓库差异

HEAD -- 文件名 查看工作区和仓库差异

git log 命令显示从最近到最远的提交日志，大概就是HEAD到根

--pretty=oneline 单行显示

git reflog 查看历史操作，可以查找新版本id用

git reset HEAD file 去除暂缓区file

git reset --hard HEAD^ 工作区和暂缓区版本回退

HEAD 表示当前版本 ^ 表示上一版本，上上版本则是 HEAD^^

上二十个版本HEAD~20

指定版本输入指定版本id

git reset

--hard 回退版本，删除暂存区，回退工作区

--mixed 回退版本，删除暂存区，工作区不变

--soft 回退版本，暂存区不变，工作区不变

git checkout -- 文件名 文件回到最近一次 git commit 或 git add 时的状态

分支管理

git branch 分支名 创建分支

git checkout 分支名 、git switch master 切换分支

git checkout -b 分支名 、git switch -c 分支名 创建分支并切换到该分支

git branch 列出所有分支，当前分支前面会标一个 \* 号

git branch -d 分支名 删除分支

git branch -D feature-vulcan 删除没用合并过的分支

git log --graph --pretty=oneline --abbrev-commit 查看分支情况

git merge 指定分支 命令用于合并指定分支到当前分支

git merge --no-ff -m "描述" 指定分支 合并指定分支到当前分支，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。

Bug分支/stash管理

git stash 存储当前工作区到栈

git stash list 查看栈

git stash apply 回复到现场 stash 内容不变

git stash apply stash@{0} 回复指定现场

git stash drop 删除stash

git stash pop 弹出栈顶，会删除

git cherry-pick 提交编号 复制特顶提交到当前分支

多人协作

git remote 查看远程库信息

-v 更详细

哪些分支需要推送，哪些不需要呢？

maste 分支是主分支，因此要时刻与远程同步；

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；

bug分支只用于在本地修复bug，就没必要推到远程了，除非要看看你每周到底修复了几个bug；

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。

克隆

git clone

git checkout -b branch-name origin/branch-name 在本地创建和远程分支对应的分支

git pull 默认拉取当前分支，当前分支与远程分支存在追踪关系时 git pull 就可以省略远程分支名。

推送

首先，可以试图用 git push origin <branch-name> 推送自己的修改；

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；

如果合并有冲突，则解决冲突，并在本地提交；

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！

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>。

Rebase

git rebase 将分叉的分支重新合并，git push 已经落后版本需要 git pull 后合并提交，这时会产生分支。使用后合并分支

git pull --rebase

标签管理

git tag <name> 默认标签是打在最新提交的commit上

git tag 查看标签

git tag <name> <commit id> 给某个commit id打标签

git show <tagname> 查看标签信息

git tag -a <tagname> -m "blablabla..." <commit id> 指定标签信息

git tag -d <tagname> 删除标签

git push origin <tagname> 可以推送一个本地标签

git push origin --tags 可以推送全部未推送过的本地标签

git push origin :refs/tags/<tagname> 删除远程标签，要先删除本地标签。

Github

在本地的learngit仓库下运行命令：

git remote add origin git@github.com:michaelliao/learngit.git