Git的操作和使用

远程仓库的默认名称是origin,远程主分支的默认名称是masater

第一步,下载安装和配置姓名邮箱

* 下载地址：<https://git-for-windows.github.io>
* 配置个人信息（名字与邮箱）  
  使用Git的第一件事就是设置你的名字和email,这些就是你在提交commit时的签名
* git config --global user.name "Your Name"
* git config --global user.email "email@example.com"

查看是否配置成功，用命名git config -l

第二步,创建本地仓库

git init：把当前目录变成一个git仓库，并自动创建master分支

以上命令会在当前目录下创建了一个.git 隐藏目录，它就是所谓的Git 仓库。生成仓库后的目录就不是普通的文档目录了，我们将其称为工作区，所以工作区中都包含一个git仓库，而一个git仓库中又包含一个暂存区和一个版本库

第三步,添加文件到版本库

工作区与版本库示意图  
Git有三大区：工作区、暂存区、版本库（如上图 ）

* 工作区（Working Directory）：电脑中的目录
* 仓库：工作区有一个隐藏目录.git，这个不算工作区，而是Git仓库。
  + 暂存区（stage/index）：一个临时的存储区域
  + 版本库 （Repository）
* 创建（修改）文件  
  往工作区中添加/修改文件
* 添加到暂存区：git add <file>  
  把修改存放到暂存区
  + git add 文件夹：把文件夹下的所有修改添加到暂存区
  + git add .：添加所有修改到暂存区

如果想过滤部分文件，请查看 [过滤清单](file:///F:\%E7%AC%AC%E4%BA%8C%E9%98%B6%E6%AE%B5\1705\javascript\20git_module\doc\Git.html#git_3)

* 提交到版本库：git commit -m "备注"  
  使用git commit 命令可将暂存区的内容提交至版本库中，这个过程称为提交，每一次提交都意味着版本在进行一次更新（会自动生成一个commit id）

第四步 关联远程仓库

方法1创建SSH Key：ssh-keygen -t rsa -C 'email地址',以上命令会在当前window用户的目录里创建.ssh目录，里面有id\_rsa（私钥）和id\_rsa.pub（公钥）两个文件

添加SSH Key到Git服务器 .添加完成后测试线路是否连通：ssh -T [git@github.com](mailto:git@github.com)

方法2 直接使用http进行连接,无需操作

建立本地仓库与远程仓库的连接

* 方式1：适用于先有本地仓库，后有远程仓库的情况  
  格式：git remote add 远程仓库名 远程仓库地址
* git remote add origin git@github.com:xxx/view.git

PS：删除远程仓库连接：git remote remove 远程仓库名

* 方式2：克隆（适用于先有远程库，后有本地仓库的情况）  
  格式：git clone 远程仓库地址

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

第五步 推送至远程仓库

git push  
格式：git push 远程仓库名 本地分支名:远程分支名  
把本地分支内容推送到远程分支（远程分支名省略表示推送到与本地分支相同的分支）

git push origin master

(自己默认的分支名如果不去创建的话就是master,这拉取的话就是拉取master的内容,如果你有创建本地分支的话,那么省略就推送到你的分支上,如果默认推送到master的话,推送的冲突怎么解决)

第六步 拉取和合并

git pull  
格式：git pull 远程仓库名 远程分支名:本地分支名  
拉取远程分支内容到本地并与本地分支进行合并（本地分支名省略表示合并到与远程分支名相同的分支）

git pull origin master

(自己默认的分支名如果不去创建的话就是master,这拉取的话就是拉取master的内容)

* git fetch  
  拉取远程分支内容
* git merge  
  合并分支内容

git pull origin master

//以上命令相当与以下命令等效

git fetch origin master

git merge origin/master

(拉取到本地,实际是两个步骤,就是拉取和合并,但是拉取后与本地文件的冲突怎么解决????)

以上拉取与合并就到这里了

注释与回退等(针对还没有推到仓库之前)

* 回退命令：git reset
  + 回退到上一个版本  
    git reset --hard HEAD^
  + 回退到指定版本：  
    git reset --hard [commit id] 版本号没必要写全，前几位就可以了，Git会自动去找。
  + 回退指定文件  
    git reset --hard [commit id] <file>
  + 参数说明
    - –hard:工作区、暂存区、版本库的文件同时回退
    - –mixed：暂存区、版本库的文件回退（默认）
    - –soft：仅仅回退版本库中的文件
* 当前版本：HEAD  
  上一个版本：HEAD^  
  上上个版本：HEAD^^  
  … 依此类推  
  前100个版本：HEAD~100
* 显示从最近到最远的提交日志：git log
  + –pretty=oneline（显示简要信息id+备注）
  + –graph（图形显示版本走向）
  + –abbrev-commit（显示简写的id）
  + 一大串类似3628164…882e1e0的是commit id（版本号）
* 查看命令历史：git reflog
* 撤销文件修改
  + git checkout -- <file>：放弃工作区的修改
  + git rm --cache <file>：撤销暂存区的修改
  + git reset HEAD <file>：撤销暂存区的修改
* 对比文件：git diff <file>

重难点(分支操作)

方式1

创建分支：git branch 分支名

切换分支：git checkout 分支名

以上两步合并为：git checkout -b 分支名

方式2

查看分支：git branch  
列出所有分支，当前分支前面会标一个\*号

方式3

合并分支：git merge 分支名

git merge dev：把dev分支合并到当前分支

方式4

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，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；

那在哪干活呢？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。