**Git的简介：**

Git 是一款免费的、开源的、分布式的版本控制系统。旨在快速高效地处理无论规模大小的任何软件工程。

每一个 Git克隆 都是一个完整的文件库，含有全部历史记录和修订追踪能力，不依赖于网络连接或中心服务器。其最大特色就是“分支”及“合并”操作非常快速、简便。

**Git与svn的主要区别:**

Git是分布式SCM，而SVN是基于服务器的，也就是说每个开发者本地都有一套git库，每个人维护自己的版本（或者合并其他人的版本），而SVN是每个人写完代码后都及时的checkin到服务器上，进行合并。

**Git的优势：**

**说到优势，那么自然是相对与SVN而言的**

版本库本地化，支持离线提交，相对独立不影响协同开发。每个开发者都拥有自己的版本控制库，在自己的版本库上可以任意的执行提交代码、创建分支等行为。例如，开发者认为自己提交的代码有问题？没关系，因为版本库是自己的，回滚历史、反复提交、归并分支并不会影响到其他开发者。

更少的“仓库污染”。git对于每个工程只会产生一个.git目录，这个工程所有的版本控制信息都在这个目录中，不会像SVN那样在每个目录下都产生.svn目录。

把内容按元数据方式存储，完整克隆版本库。所有版本信息位于.git目录中，它是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签、分支、版本记录等。

支持快速切换分支方便合并，比较合并性能好。在同一目录下即可切换不同的分支，方便合并，且合并文件速度比SVN快。

分布式版本库，无单点故障，内容完整性好。内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

**本地创建git远程仓库：**

**我这里使用的是：centos7.2、并且在服务器中已经创建git账户；如果没有创建git账户的朋友，需要先创建git，并且在服务器端安装git**

1、在本地建立一个空白的git仓库:

Git –bare init

注：--bare参数相当与只创建一个空白的仓库，只包含记录版本库历史记录的.git目录下面的文件，不会包含实际项目源文件的拷贝;

2、将本地创建的仓库添加到远程服务器，使用的linux上传命令；

Scp –r 文件夹 linux用户名@ip:/文件目录

如：scp –r gittest.git [root@123.207.172.12:/data./git](mailto:root@123.207.172.12:/data./git)

(这里是将本地的gittest.git文件夹拷贝到服务器中 /data/git文件夹下)

3、此时可以直接将本地的gittest文件夹关联远程仓库，也可以在本地再次克隆git仓库；使用命令如下：

Git clone [git@123.207.172.15:/data/git/gittest.git](mailto:git@123.207.172.15:/data/git/gittest.git)

4、可以直接创建一个文件，进行上传测试；如:

cd gittest 进入文件夹

vi app.js 然后随便输入字符，esc+wq!进行保存退出；

然后将操作添加至暂存区：

Git add .

创建本地仓库的版本：

Git commit –m “haha”

提交至远程服务器仓库：

Git push origin master;

注：如果此时出现此种错误：

remote: error: insufficient permission for adding an object to repository database ./objects

**这时注意自己的服务器git账户是否有写的权限；一般是因为git账户对gittest.git文件夹权限不够；此时登陆自己的服务器，给git账户进行权限的赋予；命令为：**

**Chown –R git:git /data/git/gittest.git**

**权限操作完毕后，再次进行本地仓库与远程仓库的同步，一般此时不再会进行报错；**

**注：服务器git仓库的创建也可以直接在linux操作系统中进行；**

**本地git创建仓库与github的远程仓库相关联:**

1、首先登陆github官网；进行注册、登录

2、创建新的仓库

3、在本地创建git仓库

mkdir githubtest

git init

vi app.js

注：https协议和ssh协议的区别就是每次远程操作都需要输入github的用户名和密码；

4、https协议：

Git remote add origin <https://github.com/misterguang/githubtest.git>

Git add .

Git commit –m “haha”

Git push –u origin master

此时需要输入github的用户名和密码

此时会报错: error: failed to push some refs to

**（此种原因一般为在github的远程仓库有文件，在本地仓库没有，所以此时应该先将远程仓库合并到本地仓库，再进行提交）**

git pull --rebase origin master

注意：这里的rebase和merge的区别，简单理解，rebase在log中无分叉，而merge有

再次提交，将会成功；

5、ssh协议：

首先需要创建密钥

Ssh-keygen –t rsa –C [837990335@qq.com](mailto:837990335@qq.com)

可以一路回车；

(Enter file in which to save the key (/c/Users/dream/.ssh/id\_rsa):

这里默认就可以，这是存放ssh密钥的路径)

（Enter passphrase (empty for no passphrase):这里为ssh的密码，可以为空）

然后到C:\Users\用户 \.ssh中找到id\_rsa.pub,然后复制里面的密钥到github

例：C:\Users\dream\.ssh

在github的ssh and gpg keys中new ssh key

Title为此ssh的标记

Key为你本地的密钥

测试下：

ssh [git@github.com](mailto:git@github.com)

如果输出You've successfully authenticated；说明链接成功；

下面的操作跟https的一样：

**Git的分支管理：**

查看本地分支：$ git branch

查看远程分支：$ git branch -r

创建本地分支：$ git branch [name] ----注意新分支创建后不会自动切换为当前分支

切换分支：$ git checkout [name]

创建新分支并立即切换到新分支：$ git checkout -b [name]

删除分支：$ git branch -d [name] ---- -d选项只能删除已经参与了合并的分支，对于未有合并的分支是无法删除的。如果想强制删除一个分支，可以使用-D选项

合并分支：$ git merge [name] ----将名称为[name]的分支与当前分支合并

创建远程分支(本地分支push到远程)：$ git push origin [name]

删除远程分支：$ git push origin :heads/[name] 或 $ gitpush origin :[name]

**分支管理策略：**

**项目开发的分支：**

Master:用于发布版本的分支；（用于大版本号更新时使用）

Dev：开发时的主分支；

Feature：功能分支；（开发某个局部分支，从dev分支上分出来的）

Release：创建一个预发布分支（从dev分支上分出来，合并到master分支上，进行tag标注）

Bug：bug分支（从master分支上分出来，修改完毕后合并到master和dev分支）

在git进行分支的合并时

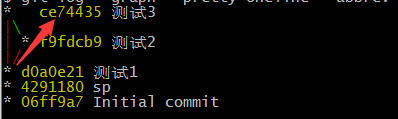
如果使用默认的fast-farward merge

直接修改当前HEAD指针的指向然后再修改当前HEAD指针，说白了就是修改两个指针的指向，而并没有生成新的commit对象。

C:\Users\dream\AppData\Local\Microsoft\Windows\INetCache\Content.Word\QQ截图20170423135221.png

如果使用—no-ff 进行合并

这样会在master分支上创建一个版本；



**git如何解决代码冲突：**

**逻辑冲突**

git自动处理（合并/应用补丁）成功，但是逻辑上是有问题的。

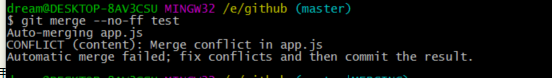
比如另外一个人修改了文件名，但我还使用老的文件名，这种情况下自动处理是能成功的，但实际上是有问题的。

又比如，函数返回值含义变化，但我还使用老的含义，这种情况自动处理成功，但可能隐藏着重大BUG。这种问题，主要通过自动化测试来保障。所以最好是能够写出比较完备的自动化测试用例。

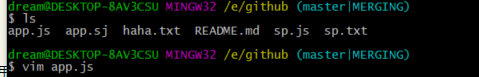
这种冲突的解决，就是做一次BUG修正。不是真正解决git报告的冲突。

**内容冲突**

两个用户修改了同一个文件的同一块区域，git会报告内容冲突。我们常见的都是这种，后面的解决办法也主要针对这种冲突。如图中是：app.js冲突；



然后手动修改冲突的文件，

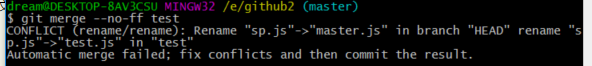


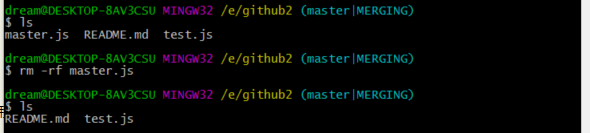
再次进行文件的提交，版本的创建；

**树冲突**

文件名修改造成的冲突，称为树冲突。

比如，a用户把app.js改名为master.js，b用户把app.js文件改名为test.js，那么b将这两个commit合并时，会产生冲突。

如图：

此时打开文件，把需要的删除的文件删除掉即可，如把master.js删除掉；

然后，再次重新创建版本即可；

**git常用操作命令：**

**查看、添加、提交、删除、找回，重置修改文件**

git help <command> # 显示command的help

git show # 显示某次提交的内容 git show $id

git co -- <file> # 抛弃工作区修改

git co . # 抛弃工作区修改

git add <file> # 将工作文件修改提交到本地暂存区

git add . # 将所有修改过的工作文件提交暂存区

git rm <file> # 从版本库中删除文件

git rm <file> --cached # 从版本库中删除文件，但不删除文件

git reset <file> # 从暂存区恢复到工作文件

git reset -- . # 从暂存区恢复到工作文件

git reset --hard # 恢复最近一次提交过的状态，即放弃上次提交后的所有本次修改

git ci <file> git ci . git ci -a # 将git add, git rm和git ci等操作都合并在一起做　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　git ci -am "some comments"

git ci --amend # 修改最后一次提交记录

git revert <$id> # 恢复某次提交的状态，恢复动作本身也创建次提交对象

git revert HEAD # 恢复最后一次提交的状态

**查看文件diff**

git diff <file> # 比较当前文件和暂存区文件差异 git diff

git diff <id1><id1><id2> # 比较两次提交之间的差异

git diff <branch1>..<branch2> # 在两个分支之间比较

git diff --staged # 比较暂存区和版本库差异

git diff --cached # 比较暂存区和版本库差异

git diff --stat # 仅仅比较统计信息

**查看提交记录**

git log git log <file> # 查看该文件每次提交记录

git log -p <file> # 查看每次详细修改内容的diff

git log -p -2 # 查看最近两次详细修改内容的diff

git log --stat #查看提交统计信息

**tig**

Mac上可以使用tig代替diff和log，brew install tig

**Git 本地分支管理**

**查看、切换、创建和删除分支**

git br -r # 查看远程分支

git br <new\_branch> # 创建新的分支

git br -v # 查看各个分支最后提交信息

git br --merged # 查看已经被合并到当前分支的分支

git br --no-merged # 查看尚未被合并到当前分支的分支

git co <branch> # 切换到某个分支

git co -b <new\_branch> # 创建新的分支，并且切换过去

git co -b <new\_branch> <branch> # 基于branch创建新的new\_branch

git co $id # 把某次历史提交记录checkout出来，但无分支信息，切换到其他分支会自动删除

git co $id -b <new\_branch> # 把某次历史提交记录checkout出来，创建成一个分支

git br -d <branch> # 删除某个分支

git br -D <branch> # 强制删除某个分支 (未被合并的分支被删除的时候需要强制)

**分支合并和rebase**

git merge <branch> # 将branch分支合并到当前分支

git merge origin/master --no-ff # 不要Fast-Foward合并，这样可以生成merge提交

git rebase master <branch> # 将master rebase到branch，相当于： git co <branch> && git rebase master && git co master && git merge <branch>

**Git补丁管理(方便在多台机器上开发同步时用)**

git diff > ../sync.patch # 生成补丁

git apply ../sync.patch # 打补丁

git apply --check ../sync.patch #测试补丁能否成功

**Git暂存管理**

git stash # 暂存

git stash list # 列所有stash

git stash apply # 恢复暂存的内容

git stash drop # 删除暂存区

**Git远程分支管理**

git pull # 抓取远程仓库所有分支更新并合并到本地

git pull --no-ff # 抓取远程仓库所有分支更新并合并到本地，不要快进合并

git fetch origin # 抓取远程仓库更新

git merge origin/master # 将远程主分支合并到本地当前分支

git co --track origin/branch # 跟踪某个远程分支创建相应的本地分支

git co -b <local\_branch> origin/<remote\_branch> # 基于远程分支创建本地分支，功能同上

git push # push所有分支

git push origin master # 将本地主分支推到远程主分支

git push -u origin master # 将本地主分支推到远程(如无远程主分支则创建，用于初始化远程仓库)

git push origin <local\_branch> # 创建远程分支， origin是远程仓库名

git push origin <local\_branch>:<remote\_branch> # 创建远程分支

git push origin :<remote\_branch> #先删除本地分支(git br -d <branch>)，然后再push删除远程分支

**Git远程仓库管理**

[GitHub](http://blog.jobbole.com/6492/)

git remote -v # 查看远程服务器地址和仓库名称

git remote show origin # 查看远程服务器仓库状态

git remote add origin git@ github:robbin/robbin\_site.git # 添加远程仓库地址

git remote set-url origin git@ github.com:robbin/robbin\_site.git # 设置远程仓库地址(用于修改远程仓库地址) git remote rm <repository> # 删除远程仓库

**创建远程仓库**

git clone --bare robbin\_site robbin\_site.git # 用带版本的项目创建纯版本仓库

scp -r my\_project.git git@ git.csdn.net:~ # 将纯仓库上传到服务器上

mkdir robbin\_site.git && cd robbin\_site.git && git --bare init # 在服务器创建纯仓库

git remote add origin git@ github.com:robbin/robbin\_site.git # 设置远程仓库地址

git push -u origin master # 客户端首次提交

git push -u origin develop # 首次将本地develop分支提交到远程develop分支，并且track

git remote set-head origin master # 设置远程仓库的HEAD指向master分支

也可以命令设置跟踪远程库和本地库

git branch --set-upstream master origin/master

git branch --set-upstream develop origin/develop