\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Git&Svn\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Git 速度 > SVN

Git是分布式管理，SVN是集中式管理

Git可以轻松拥有无限个分支，SVN使用分支比较笨拙(通过HEAD指针控制)

Git支持本地版本控制工作，SVN必须联网才能正常工作

Git安全性> SVN, 因为每个人电脑里都有完整的版本库,即使服务器崩溃了，也影响不大

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Git \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

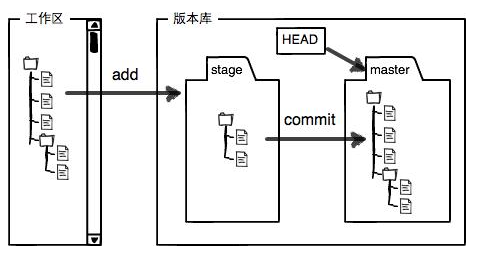
GIT:分布式源代码管理

客户端：SourceTree

多人开发需要一个共享版本库，单人开发初始化一个本地库即可

HEAD指针：用于指向当前分支

工作原理：



在工作区修改内容后，通过add提交到暂存区（stage）,然后commit提交到当前分支，最后push到远程库

命令总结

git init 创建本地版本库

git add . 添加多个文件

git commit -m “message” 提交本地版本库

git diff <file> 查看difference

git diff HEAD -- <file> 查看工作区和版本库里面最新版本的区别

git status 查看仓库当前的状态

git log 查看提交日志，获取对应的commit\_id，以便确定要回退到哪个版本（git log --pretty=oneline 可减少输出信息）

git log --graph命令可以看到分支合并图

git reflog查看命令历史，以便确定要回到未来的哪个版本

撤销—回退

还没有add 的撤销：

git checkout -- <file> 直接丢弃工作区的修改(修改还没提交到暂存区，新增和删除的仍然没有恢复，还需要执行下面的命令)

git clean -df 放弃所有修改、新增、删除文件

git add 后撤销：

撤销所有add文件 git reset HEAD .

撤销单个add文件 git reset HEAD –filename

撤销 commit：

1.只回退commit的信息，保留修改代码：

git reset ID

2彻底回退到上次commit版本，不保留修改代码：

git reset --hard head^/ID

【HEAD ：当前版本，HEAD^ ：上一个版本

--hard 参数会抛弃当前工作区的修改

--soft 参数的话会回退到之前的版本，但是保留当前工作区的修改，可以重新提交】

3.部分文件回退

git checkout -- files

git push后撤销

回滚此次push到服务器的代码：

git log查看commit的信息

git revert 以前commit的id

git push 此时本地回滚的代码到服务器就可以了

git merge 后撤销

git checkout 【行merge操作时所在的分支】

git reset --hard 【merge前的版本号】

git rm <file> 从版本库中删除该文件，并且git commit（另一种情况是删错了，版本库里还有，可以通过git checkout --<file> 把误删的文件恢复到最新版本）

git clone 远程库地址 克隆一个远程仓库

git checkout -b dev(分支名) 创建dev分支，并切换到dev分支，相当于（git branch dev:创建分支 git checkout dev:切换分支)

git branch 查看所有分支，当前分支前面会标一个\*号

git checkout master 切换回master分支

git merge dev 把dev分支合并到master分支上,这是 fast forward模式合并，看不出来曾经做过合并

git merge --no-ff -m "message" dev, 普通模式合并，合并后的历史有分支，能看出来曾经做过合并

git branch -d dev 合并完成后， 删除dev分支

git stash list 查看stash列表

git stash apply stash@{0} 恢复到指定的stash

git stash drop 删除stash内容

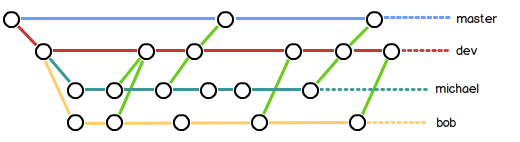
(Git鼓励你使用分支完成某个任务，合并后再删掉分支，这和直接在master分支上工作效果是一样的，但过程更安全。)

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。

分支策略：

1.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；

2. 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



Bug分支：

Git还提供了一个stash功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作

1. git stash 在dev分支先隐藏修改的内容

2. 首先确定要在哪个分支上修复bug，假定需要在master分支上修复，就从master创建临时分支

2.1 git checkout master回到master分支

2.2 git checkout -b bug在master创建并切换到bug分支

2.3 git checkout master修复完成后，切换到master分支

2.4 git merge --no-ff -m "massage" bug 完成合并

3. git checkout dev回到stash前分支

4. git stash pop恢复stash前的修改并把stash内容也删了

Feature分支

本地仓库和远程仓库管理：

1. 关联一个远程库

git remote add origin 远程仓库路径

2.把本地文件添加到本地仓库

git add . 然后 git commit -m "备注"

3. 本地库的所有内容推送到远程库上

git push -u origin master

第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令

注意：当执行3时，终端显示

src refspec master does not match any.

说明本地库为空，先执行2，再重新执行3

4.以后本地推送到远程都不用 -u

git push origin master

5.终端快捷键

pwd 显示当前文件夹所在的路径

ls –a 显示目录下所有文件（包括隐藏文件）

rm file 删除文件

github和本地git仓库关联

1. 创建SSH Key

ssh-keygen -t rsa -C "git账号"

2.关联SSH Key

登陆GitHub，”Add SSH Key”,把1创建的id\_rsa.pub文件内容粘贴到github上，提交

（为什么GitHub需要SSH Key呢？因为GitHub需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送。）

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SVN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SVN:Subversion，集中式版本控制

第三方托管平台：SVNSpot

客户端：CornerStone/Versions

命令总结：

1.上传文件到svn服务器： svn+import+文件本地路径+服务器路径 +（–m)+ “提交的备注信息（方便别人查看）”

2.服务器下载文件到本地： svn+checkout+文件服务器路径+本地保存路径 --username=xxx --password=xxx

3.提交修改的文件：进入文件的路径，svn+commit+（–m)+ “提交的备注信息”

4.新添加文件：进入文件的路径，

svn+add+文件1+文件2+… ,最后要commit

5.删除文件(一定不要手动删除)：进入文件的路径，

svn+remove+文件1+文件2+… ,最后要commit

6.更新 svn+update

7.解决冲突：svn+resolve+冲突的文件名

系统有三种方案：

p:延迟处理，会生成3个文件，冲突代码合并，需要执行 svn resolve filename

mc:使用我的代码

tc:使用他人的代码

8.版本回退：

修改但还没提交：svn+revert+要回退的文件名（返回上一个版本的状态）

修改并已经提交：首先 update到最新版本

然后 svn+merge+文件名+(-r)+版本1：版本2

最后 svn+commit(从版本1会退到版本2)

9.查看本地路径是否修改：进入文件的路径，svn+status

10.查看svn日志 svn+log

11.加锁 svn+lock

12.解锁 svn+unlock

<快捷 ci---commit ; st---status; up---update>

svn目录规范：

trunk:主干，当前开发项目的主目录

branch:分支目录，添加非主线的功能，修改bug，开发新技术

tag:标记目录，通常作为重大版本的备份

在trunk开发完成1.0版本，然后在tag添加1.0版本的备份，接着继续开发2.0版本，这时候测试发现1.0版本有bug，然后在tag开启一个branch分支，当bug修复完成变成1.1版本，同时在tag备份1.1版本，而且把branch的1.1版本合并到trunk中

！：文件丢失（手动删除）

Windows在控制台输入ipconfig获取ip地址

Com+space 快速查找系统软件或文件

输入 “终端” ---- 快速打开终端

com+k --- 清空终端信息

终端快捷键：

touch xxx.xxx --- 新建文件

rm xxx.xxx --- 删除文件

open xxx.xxx --- 打开文件

mkdir xxx --- 新建文件夹

mv xxx.xxx 路径 --- 移动文件

pwd --- 显示文件所在路径

新建 点开头的为隐藏文件，如 .doc

HTTP状态码的含义：

2xx （成功）

200（成功）：服务器已成功处理了请求

202（已接受）：服务器已接受请求，但尚未处理

203（非授权信息）

204（无内容）：服务器成功处理了请求，但没有返回任何内容。

3xx （重定向)

301（永久移动）：请求的网页已永久移动到新位置

302（临时移动）

4XX（请求错误）

400（错误请求）：服务器不理解请求的语法

401（未授权）：请求要求身份验证

403（禁止）：服务器拒绝请求

404（未找到）：服务器找不到请求的网页

408（请求超时）

410（已删除）：请求的资源已永久删除

413（请求实体过大）

414（请求的 URI 过长）

5xx（服务器错误）

500（服务器内部错误）

502（错误网关）

503（服务不可用）：服务器目前无法使用（由于超载或停机维护）。通常，这只是暂时状态。

504（网关超时）

505（HTTP 版本不受支持）