# Git使用相关环境

## 1、git服务器地址:

[http://10.50.80.163:80](http://10.50.80.163/)

## 2、客户端下载地址：

<http://10.50.80.163/liuwei/tools/blob/3fe7c69c0eb7953c8580af5480a94000ef00a52e/msysgit-1.8.3-1374045102000.exe>

## 3、git命令在线中文参考手册

<http://gitref.org/zh/remotes/>

# Git基本工作原理与设置

## 工作原理

### 1、分布式版本控制系统

客户端并不只提取最新版本的文件快照，而是把原始的代码仓库完整地镜像下来。这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜 像出来的本地仓库恢复。因为每一次的提取操作，实际上都是一次对代码仓库的完整备份(见图1-1)。

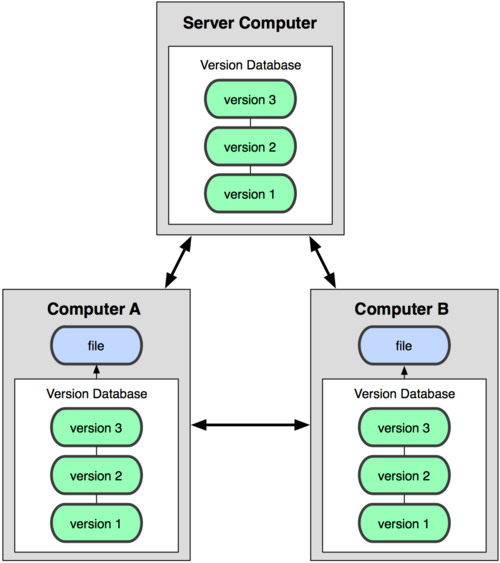


图 1-1. 分布式版本控制系统

### 文件的三种状态

对于任何一个文件，在 Git 内都只有三种状态：已提交（committed），已修改（modified）和已暂存（staged）。已提交表示该文件已经被安全地保存在本地数据库 中了；已修改表示修改了某个文件，但还没有提交保存；已暂存表示把已修改的文件放在下次提交时要保存的清单中。

由此我们看到 Git 管理项目时，文件流转的三个工作区域：Git 的工作目录，暂存区域，以及本地仓库。

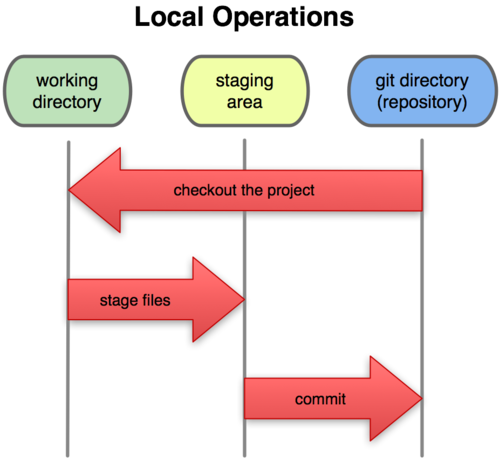


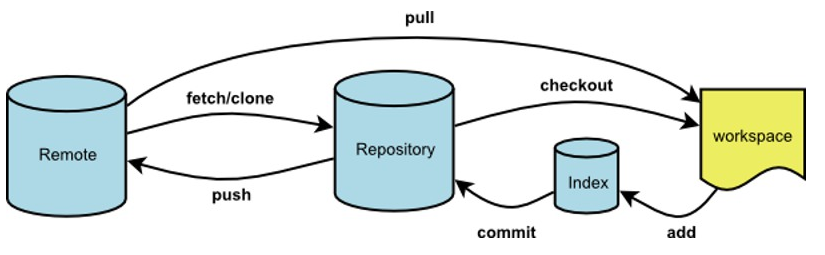
图 1-2. 工作目录，暂存区域，以及本地仓库

每个项目都有一个 Git 目录（译注：如果 git clone 出来的话，就是其中 .git 的目录；如果git clone --bare 的话，新建的目录本身就是 Git 目录。），它是 Git 用来保存元数据和对象数据库的地方。该目录非常重要，每次克隆镜像仓库的时候，实际拷贝的就是这个目录里面的数据。

从项目中取出某个版本的所有文件和目录，用以开始后续工作的叫做工作目录。这些文件实际上都是从 Git 目录中的压缩对象数据库中提取出来的，接下来就可以在工作目录中对这些文件进行编辑。

所谓的暂存区域只不过是个简单的文件，一般都放在 Git 目录中。有时候人们会把这个文件叫做索引文件，不过标准说法还是叫暂存区域。

基本的 Git 工作流程(包括远程操作)如下图：



1. 在工作空间workspace中修改某些文件。
2. 对修改后的文件进行快照，然后执行git add保存到暂存区域。
3. Git commit提交更新，将保存在暂存区域的文件快照永久转储到 Git 目录中。
4. 通过git push origin命令将修改文件推送到远程git服务器.

所以，我们可以从文件所处的位置来判断状态：如果是 Git 目录中保存着的特定版本文件，就属于已提交状态；如果作了修改并已放入暂存区域，就属于已暂存状态；如果自上次取出后，作了修改但还没有放到暂存区域，就 是已修改状态。

## 二、Git与SVN的区别

1、GIT是分布式的，SVN不是，这是GIT和其它非分布式的版本控制系统，最核心的区别；

GIT跟SVN一样有自己的集中式版本库或服务器。但，GIT更倾向于被使用于分布式模式，也就是每个开发人员从中心版本库/服务器上chect out代码后会在自己的机器上克隆一个自己的版本库。  
2、Git直接记录快照，而非差异比较

Git 和其他版本控制系统的主要差别在于，Git 只关心文件数据的整体是否发生变化，而大多数其他系统则只关心文件内容的具体差异。Git 并不保存这些前后变化的差异数据。实际上，Git 更像是把变化的文件作快照后，记录在一个微型的文件系统中。每次提交更新时，它会纵览一遍所有文件的指纹信息并对文件作一快照，然后保存一个指向这次快照 的索引。为提高性能，若文件没有变化，Git 不会再次保存，而只对上次保存的快照作一链接。

3、近乎所有操作都是本地执行

在 Git 中的绝大多数操作都只需要访问本地文件和资源，不用连网。但如果用 CVCS 的话，差不多所有操作都需要连接网络。因为 Git 在本地磁盘上就保存着所有当前项目的历史更新，所以处理起来速度飞快。

## 三、设置SSh

使用SSH密钥在本地电脑和Git服务器之间建立安全连接

（1）检查是否已有SSH key

输入以下命令：

$ cd ~/.ssh

如果提示“No such file or directory“,则跳转到第3步，如果不是则继续第2步。

（2）备份并删除已有的SSH key

$ ls

Lists all the subdirectories in the current directory

config id\_rsa id\_rsa.pub known\_hosts

$ mkdir key\_backup

makes a subdirectory called "key\_backup" in the current directory

$ cp id\_rsa\* key\_backup

Copies the id\_rsa and id\_rsa.pub files into key\_backup

$ rm id\_rsa\*

注意，以上只有美元符号后面才是你要在命令行中输入的命令，其它是对该命令的解释说明，或者输入命令后返回的结果。

（3）.创建一个新的SSH key

$ ssh-keygen -t rsa -C "your\_email@youremail.com"

Creates a new ssh key using the provided email

Generating public/private rsa key pair.

Enter file in which to save the key (/Users/your\_user\_directory/.ssh/id\_rsa):<press enter>

首先注意，[your\_email@youremail.com](mailto:your_email@youremail.com)这里要用你自己的邮箱地址，可别真的复制粘贴这个命令，那就有点2B了。

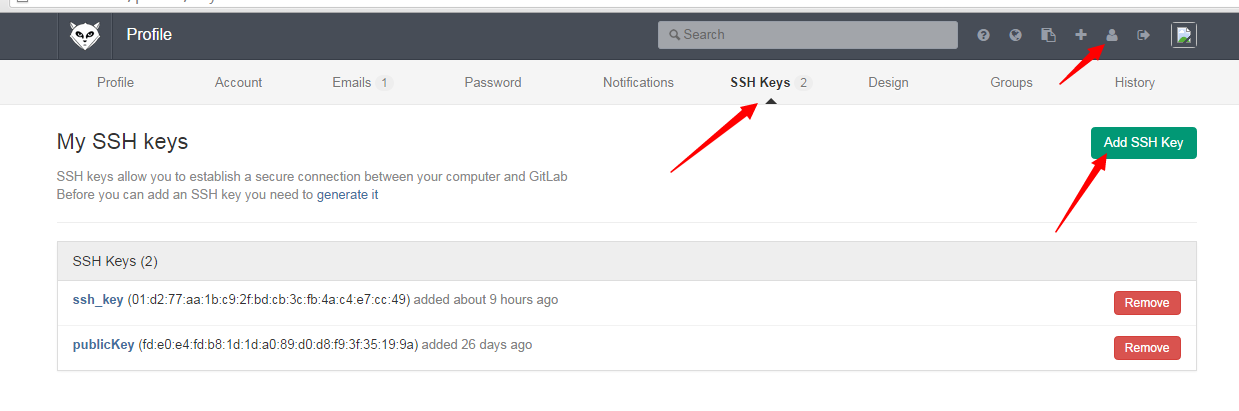
然后会提示输入passphrase（密码）：

Enter passphrase(empty for no passphrase):<输入你的密码>

Enter same passphrase again:<再次输入你的密码>

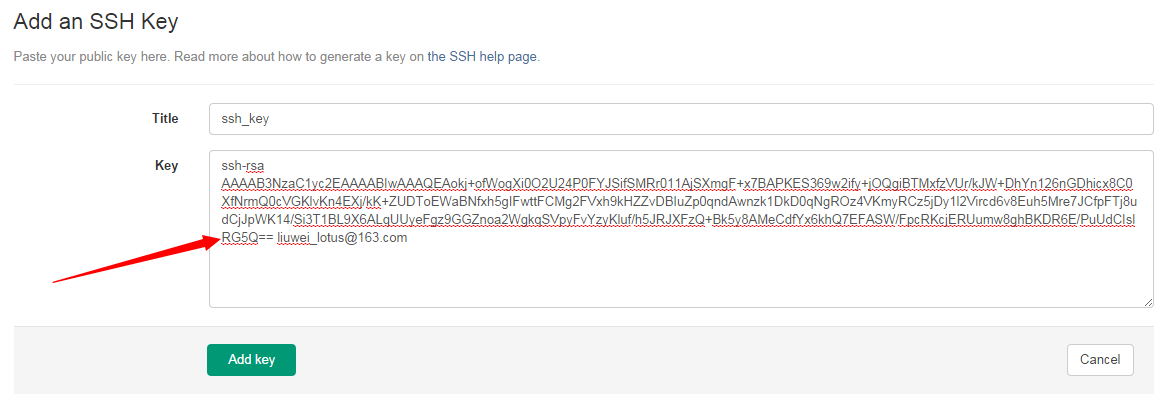
(4)将SSH key添加到Git服务器中

在Git服务器中点击Profile Settings，然后点击SSH Keys<，点击Add SSH Key，参考下图：



使用文本编辑器打开id\_rsa.pub文件。就会显示你的公共SSH密钥。

然后把它复制到”Key”字段中，参考下图：



（5）测试

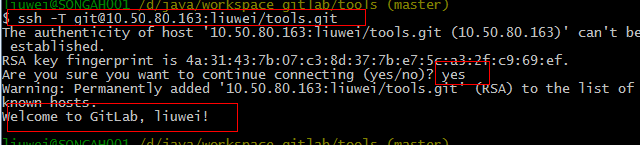
为了确保一切正常，需要进行小小的测试。

 在命令行中输入下面的命令：

$ ssh -T git@10.50.80.163:liuwei/tools.git

 根据提示输入yes，

正常情况下就会看到：



3.设置信息

现在Git已经设置完毕，也输入了SSH Key,就可以配置个人信息了。

（1）设置用户名和email

$ git config --global user.name "Firstname Lastname"

Sets the name of the user for all git instances on the system

$ git config --global user.email [your\_email@youremail.com](mailto:your_email@youremail.com)

需要注意的是，这里的name是你的真名，不是Git的注册账号。

到这里，Git的设置工作就圆满了。

# 基本命令操作流程（以微店weshopweb为例）

git clone #工程克隆

git status #查看文件缓存状态

git add #添加至暂存区

git reset #取消缓存修改(还原)

git diff #文件比较

git checkout #放弃文件修改

git commit #提交本地仓库

git log #查看提交日志

远程操作命令:

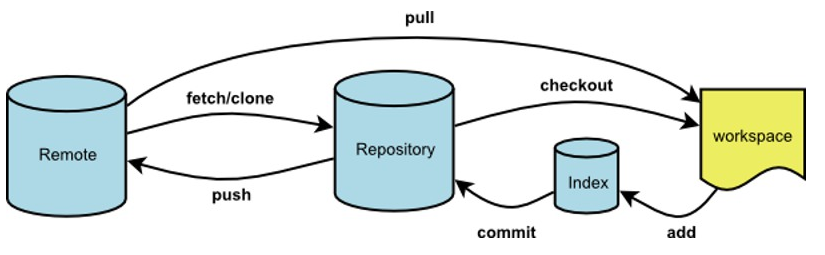
git remote #列出远程别名

git fetch #从远程仓库下载代码

git pull #从远程仓库下载代码并合并 相当于(git fetch + git merge)

git push #向远程仓库推送代码

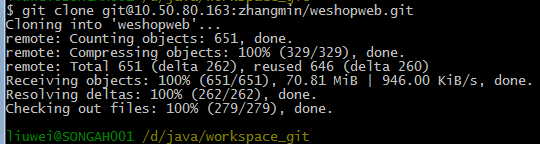
关联关系图如下：



## 一、git clone

第一步，通过Git Bash客户端，进入到本地workspace目录，如：

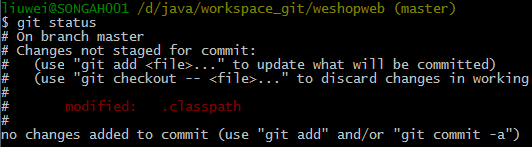
cd /d/java/workspace\_git 然后执行 git clone 命令



## 二、git status

查看你的文件在工作目录与缓存的状态

比如：修改了.classpath文件，执行git status 如下：

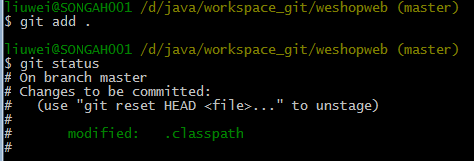


git status –s 以简短的结果输出

## 三、git add

添加文件到缓存

执行git add .后再使用git status命令查看结果如下：

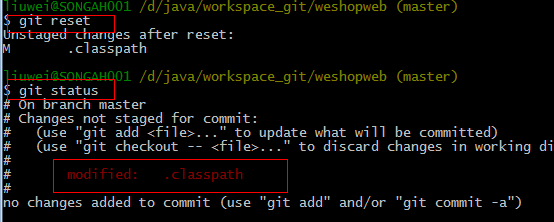


git add . 指添加当前目录下所有修改的文件，添加具体某一个执行git add .classpath

## 四、git reset

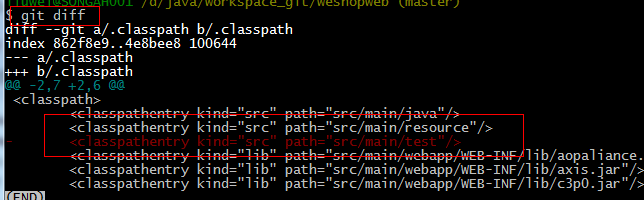
取消已缓存的内容

如果缓存后需要取消，然后重新修改提交，可以执行git reset命令，将文件返回到未缓存的状态



## 五、git diff

查看尚未缓存的改动



显示删除了红色表示的行

#### git diff --cached #查看已缓存的改动

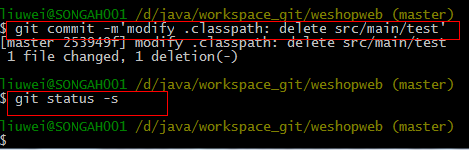
#### git diff HEAD 查看已缓存的与未缓存的所有改动

#### git diff --stat 显示摘要而非整个 diff

## 六、git commit –m’描述’

记录缓存内容的快照

现在你使用 git add 命令将想要快照的内容写入了缓存， 执行 git commit 就将它实际存储快照了。

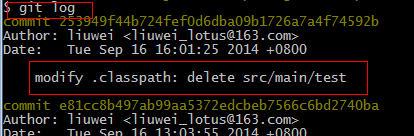


执行完git commit之后用git status查看，发现查询不到修改的文件了

#### git commit –am’描述’  自动将在提交前将已记录、修改的文件放入缓存区

## 七、git log

查看提交日志，按q键退出



git log .classpath #查看某个文件的提交日志

git log --pretty=oneline readme.txt #查看readme.txt文件的历史变动，简易版本

git log --author #只寻找某个特定作者的提交'

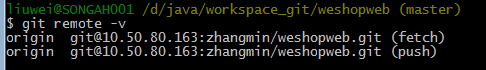
git log --grep 根据提交注释过滤提交记录

git log --oneline --graph 拓扑图形式显示日志

git log --oneline --decorate --graph 可以查看标签

## 七、git remote

列出远程别名



其中origin为默认的远程别名

git remote -v # 列出远程别名

git remote add github git@github.com:schacon/hw.git #添加远程

git remote rm github #删除远程

## 八、git fetch

从远程仓库下载数据及新分支

git fetch origin #更新远程仓库数据到当前分支

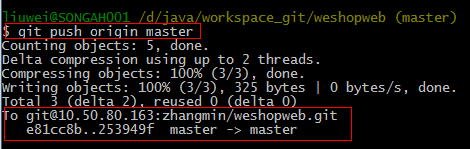
## 九、git pull

从远程仓库提取数据并尝试合并到当前分支

该命令就是在 git fetch 之后紧接着 git merge 远端分支到你所在的任意分支

## 十、git push

推送你的新分支与数据到某个远端仓库



git push -u origin master #推送你的master分支与数据到远端仓库

git push origin master （省略了<dst>，等价于“git push origin master:master”）

git push origin branch1 #推送分支到远程

git push origin :branch1 （再origin repository里面查找branch1，删除它。用一个空的去更新它，就相当于删除了）

# Git分支管理

大概每个程序员都会经常遇到这样的情况：

1. 需要立刻放下手头的工作，去修改曾经一个版本的bug并上线，然后再继续当的工作。

2. 本想向中心库commit一个重要修改，但是由于需要经常备份代码，最终不得不频繁的向中心库commit。从而导致大量无用的commit信息被保留在中心库中。

3. 将一次修改提交同事进行code review， 但是由于同事code review比较慢， 得到反馈时，自己的代码已经发生了变化，从而倒是合并异常困难

这些场景，如果用CVS或者SVN来解决，虽说不一定解决不了，但过程之繁琐，之复杂，肯定另所有人都有生不如死的感觉吧！究其关键，就是CVS或者SNV的branch管理太复杂，基本不具可用性。

在 git 版本库中创建分支的成本几乎为零，所以，不必吝啬多创建几个分支。当第一次执行git-init时，系统就会创建一个名为”master”的分支。而其它分支则通过手工创建。下面列举一些常见的分支策略，这些策略相信会对你的日常开发带来很大的便利。

1.创建一个属于自己的个人工作分支，以避免对主分支 master 造成太多的干扰，也方便与他人交流协作。

2.当进行高风险的工作时，创建一个试验性的分支，扔掉一个烂摊子总比收拾一个烂摊子好得多。

3.合并别人的工作的时候，最好是创建一个临时的分支用来合并，合并完成后在“fatch”到自己的分支（合并和fatch后面有讲述，不明白就继续往下看好了）

2.6.1 查看分支 – git-branch

调用git-branch可以查看程序中已经存在的分支和当前分支

2.6.2 创建分支 – git-branch 分支名

要创建一个分支，可以使用如下方法:

1. git-branch 分支名称

2. git-checout –b 分支名

使用第一种方法，虽然创建了分支，但是不会将当前工作分支切换到新创建的分支上，因此，还需要命令”git-checkout 分支名” 来切换， 而第二种方法不但创建了分支，还将当前工作分支切换到了该分支上。

另外，需要注意，分支名称是有可能出现重名的情况的， 比如说，我在master分支下创建了a和b两个分支，然后切换到b分支，在b分支下又创建了a和c分支。 这种操作是可以进行的。 此时的a分支和master下的a分支实际上是两个不同的分支。因此，在实际使用时，不建议这样的操作，这样会带来命名上的疑惑。

2.6.3 删除分支 – git-branch –D

git-branch –D 分支名可以删除分支，但是需要小心，删除后，发生在该分支的所有变化都无法恢复。

2.6.4 切换分支 – git-checkout 分支名

如果分支已经存在， 可以通过 git-checkout 分支名 来切换工作分支到该分支名

2.6.5 查看分支历史 –git-show-branch

调用该命令可以查看分支历史变化情况。 如：

\* [dev1] d2

! [master] m2

--

\* [dev1] d2

\* [dev1^] d1

\* [dev1~2] d1

\*+ [master] m2

在上述例子中， “--”之上的两行表示有两个分支dev1和master，且dev分支上最后一次提交的日志是“d2”,master分支上最后一次提交的日志是”m2”。 “--”之下的几行表示了分支演化的历史，其中 dev1表示发生在dev分支上的最后一次提交，dev^表示发生在dev分支上的倒数第二次提交。dev1~2表示发生在dev分支上的倒数第三次提交。

2.6.6 合并分支 – git-merge

git-merge的用法为：git-merge “some memo” 合并的目标分支 合并的来源分支。如：

如果合并有冲突，git会由提示，当前，git-merge已经很少用了， 用git-pull来替代了。

用法为：git-pull 合并的目标分支 合并的来源分支。 如git-pull . dev1