**1. git 版本控制系统**

相比CVS\SVN优势：

－ 支持离线开发，离线Repository  
－ 强大的分支功能，适合多个独立开发者协作  
－ 速度块

ps:关于git的更详细的介绍于优点在此就不介绍了，教大家怎么用是关键。:）

==============运行环境========

系统：windows

git : Git-1.7.3.1-preview20101002.rar  下载地址：<http://d.download.csdn.net/down/3169511/z_y_liu89>

===========================

**2. github是一个git项目托管网站**

注册地址：<https://github.com/signup/free>

**3. 安装git程序，执行下面操作**

$ cd ~/.ssh //检查计算机ssh密钥

如果没有提示:No such file or directory 说明你不是第一次使用git,执行下面的操作,清理原有ssh密钥

$ ls   
 config id\_rsa id\_rsa.pub known\_hosts  
 $ mkdir key\_backup  
 $ cp id\_rsa\* key\_backup  
 $ rm id\_rsa\*

获得密钥：

ssh-keygen -t rsa -C "defnngj@gmail.com"//填写email地址，然后一直“回车”ok

打开本地..\.ssh\id\_rsa.pub文件。此文件里面内容为刚才生成人密钥。

4. 登陆github系统。点击右上角的 [Account Settings](https://github.com/account)--->SSH Public keys ---> add another public keys

把你本地生成的密钥复制到里面（key文本框中）， 点击 add key 就ok了

5. 接着打开git ，测试连接是否成功

$ ssh -T git@github.com

如果提示：Hi defnngj You've successfully authenticated, but GitHub does not provide shell access. 说明你连接成功了

6. 设置用户信息：

6.1

$ git config --global user.name "defnngj"//给自己起个用户名  
$ git config --global user.email "defnngj@gmail.com"//填写自己的邮箱

6.2

在github中找到 Account Settings--->Account Admin ,找到一下信息：

Your API token is e97279836f0d415a3954c1193dba522f ---keep it secret! Changing your password will

generate a new token

$ git config --global github.user defnngj //github 上的用户名  
$ git config --global github.token e97279836f0d415a3954c1193dba522f

===================**=创建一个项目**========================

 1. 回到github首页，点击页面右下角“New Repository”

填写项目信息：

project name: hello world

description : my first project

点击“Create Repository” ； 现在完成了一个项目在github上的创建。

2. 我们需要使用git在本地创建一个相同的项目。

[复制代码](javascript:void(0);)

$ makdir ~/hello-world //创建一个项目hello-world  
$ cd ~/hello-world //打开这个项目  
$ git init //初始化   
$ touch README  
$ git add README //更新README文件  
$ git commit -m 'first commit'//提交更新，并注释信息“first commit”   
$ git remote add origin git@github.com:defnngj/hello-world.git //连接远程github项目   
$ git push -u origin master //将本地项目更新到github项目上去

[复制代码](javascript:void(0);)

 现在查看github上面的hello world 项目，是不是发现已经将本地中的README文件更新上来了。 :) 恭喜！

------------------------------------关于可能出现的错误----------------------------------

1.在执行

$ git remote addorigin git@github.com:defnngj/hello-world.git

错误提示：fatal: remote origin already exists.

解决办法：

$ git remote rm origin

然后在执行：$ git remote add origin git@github.com:defnngj/hello-world.git 就不会报错误了

2. 在执行

$ git push origin master

错误提示：error:failed to push som refs to.......

解决办法：

$ git pull origin master //先把远程服务器github上面的文件拉先来，再push 上去。

---------------------------后记-----------------------------------------------------------------------

本文是参考官方帮助进行的：<http://help.github.com/win-set-up-git/> 基本与官方步骤相同，我在此属于翻译了一下！

关于更过的学习：请登陆: <http://progit.org/book/zh/>进行学习。

本来关于此类知识应该属于开发的，本人从事测试工作，因为老大现在在推行git的使用，所以，就花了时间，初步的学习了一下，为了更好的测试嘛。呵呵。

第二个原因，看到有个乐师用版本管理系统（SVN）来更新和管理自己的乐谱，这个很有意思。版本管理系统并不局限于代码的管理。而且版本管理系统的思想也很有意思。

关于理论上的东西，请参考其它文档。

      之前初学过一点git版本控制工具，利用github做仓库，照着github上的文档练习的了一下。不过那只篇只是照虎画猫（我的水平只能照着老虎画个猫模样，嘻嘻！）。

最近在学hibernate，公司与家之间都要调用我练习的小项目，交给git/github来管理，我只想说真的爽歪歪了。

本文在我之前的那篇《git/github学习笔记》的基础上，属于那个的升级加强版，欢迎对照阅读：<http://www.cnblogs.com/fnng/archive/2011/08/25/2153807.html>

如果本文有不太明白的地方请参考上面的文章。

Github: git项目托管网站，请先免费申请一个github帐号：[www.github.com](http://www.github.com/)

Git:分布式版本控制工具，<http://d.download.csdn.net/down/3169511/z_y_liu89>

Github的申请与git的安装我这里就不说了。这个不难。

**----//git/github环境配置**

**一 .  github上创建立一个项目**

用户登录后系统，在github首页，点击页面右下角“New Repository”

填写项目信息：

**project name**: hibernate-demo

**description**: my first project

点击“**Create Repository**” ； 现在完成了一个项目在github上的创建。

说明：我们创建的是一个github仓库，一个仓库里只能存放（或叫对应）一个项目。

当你创建完成一个仓库的之后，github已经给你一个提示：当你看完了我的文章再来看这个提示就非常清楚了。

[复制代码](javascript:void(0);)

**Global setup:**  
 Set up git  
  git config --global user.name "Your Name"  
  git config --global user.email defnngj@gmail.com  
        
**Next steps:**  
  mkdir hibernaet-demo2  
  cd hibernaet-demo2  
  git init  
  touch README  
  git add README  
  git commit -m 'first commit'  
  git remote add origin git@github.com:defnngj/hibernaet-demo2.git  
  git push -u origin master

**Existing Git Repo?**  
  cd existing\_git\_repo  
  git remote add origin git@github.com:defnngj/hibernaet-demo2.git  
  git push -u origin master

**Importing a Subversion Repo?**  
  Click here   
When you're done:  
  Continue

[复制代码](javascript:void(0);)

**二  .  创建密钥**

我们如何让本地git项目与远程的github建立联系呢？之里就用的密钥。通俗点叫口令吧！（天王盖地老，宝塔镇河妖。。）

$ cd ~/. ssh 检查本机的ssh密钥

如果提示：No such file or directory 说明你是第一次使用git。

如果不是第一次使用，请执行下面的操作,清理原有ssh密钥。

$ mkdir key\_backup  
 $ cp id\_rsa\* key\_backup  
 $ rm id\_rsa\*

生成新的密钥：

Ssh-keygen –t rsa –C “defnngj@gmai.com”

**注意:**此处的邮箱地址，你可以输入自己的邮箱地址。在回车中会提示你输入一个密码，这个密码会在你提交项目时使用，如果为空的话提交项目时则不用输入。这个设置是防止别人往你的项目里提交内容。

打开本地C:\Documents and Settings\Administrator\.ssh\id\_rsa.pub文件。此文件里面内容为刚才生成人密钥。

登陆github系统。点击右上角的 [Account Settings](https://github.com/account)--->SSH Public keys ---> add another public keys

把你本地生成的密钥复制到里面（key文本框中）， 点击 add key 就ok了

在git中运行下面命令：

$ git –T git@github.com

如果提示：Hi defnngj You've successfully authenticated, but GitHub does not provide shell access. 说明你连接成功了。

**三 . 设置用户信息**

这一步不是很重要，貌似不设置也行，但github官方步骤中有，所以这里也提一下。

在git中设置用户名，邮箱

$ git config --global user.name "defnngj"//给自己起个用户名  
$ git config --global user.email "defnngj@gmail.com"//填写自己的邮箱

在github中找到 Account Settings--->Account Admin ,找到一下信息：

Your API token is e97279836f0d415a3954c1193dba522f ---keep it secret! Changing your password will

generate a new token

$ git config --global github.user defnngj //github 上的用户名  
$ git config --global github.token e97279836f0d415a3954c1193dba522f

**----//小玩一下git**

上面都是准备工作，一次完成，以后就不用设置了。下面内容才是亮点。

先来说说git下常用的几个基本操作，和linux系统的操作是一样一样的：

[复制代码](javascript:void(0);)

$ ls 查看当前目录的内容  
  
$ cd /d 切换到d盘  
  
$ cd java/ 打开当前目录下的java目录  
  
$ cd j(table键) 如果当你想打开java目录且当前目录下只有一个j开头的目录，输入J 然后按键盘上的table键，会自动帮你补齐。  
  
$ cd .. 返回上一级目录

[复制代码](javascript:void(0);)

**假如你现在新创建了一个项目，想把它提交到github上面？**

假设你创建好了一个项目，并切换到项目的根目录下面：

**$ git status** //查看当前项目下所有文的状态，如果第一次，你会发现都红颜色的，因为它还没有交给git/github管理。

**$ git add .**  //（.）点表示当前目录下的所有内容，交给git管理，也就是提交到了git的本地仓库。

**Ps**:git的强大之处就是有一个本地仓库的概念，在没有网络的情况下可以先将更新的内容提交到本地仓库。

**$ git commit –m”new natter ”**  //对你更新或修改了哪些内容做一个描述。

**$ git remote add origin git@github.com:defnngj/hibernate-demo.git**

//如果你是第一次提交项目，这一句非常重要，这是你本地的当前的项目与远程的哪个仓库建立连接。

**Ps**: **origin**可以改为别人的名字，但是在你下一次push（提交）时，也要用你修改之后的名字。

**$ git remote -v** //查看你当前项目远程连接的是哪个仓库地址。

**$ git push -u origin master** //将本地的项目提交到远程仓库中。

 ------------------------------------------------------------

**假如，你回到了家，想把公司提交的项目克隆到本地？**

如果你是第一次想把github上面的项目克隆到本地或者要克隆别人的项目到地。

**$ git clone git@github.com:defnngj/hibernate-demo.git** //在git下面切换到想存放此项目的文件目录下，运行这条命令就可以将项目克隆下来。

假如本地已经存在了这个项目，而仓库中又有一新的更新，如何把更的合并到本地的项目中？

**$ git fetch origin**   //取得远程更新，这里可以看做是准备要取了

**$ git merge origin/master**  //把更新的内容合并到本地分支/master

-------------------------------------------

**项目中删除了一些文件，如何提交？**

假如远程仓库中已经存了aaa这个文件，我fetch了下来，并删除了aaa这个文件，想再push上到远程仓库中，并使远程仓库中的项目被新的修改覆盖（也是是远程仓库中的aaa也被删除）

**$ git status**   //可以看到我们删除的哪些文件

**$ git add .**   //删除之后的文件提交git管理。

**$ git rm   src/com/hzh/hibernate/dao/aaa.java**    //移除我们删除的那个文件，不然git不允许我们往远程仓库提交。

Ps: 如果你想删除的是某个目录（java包），这里想移除整个目录的内容。

**$ git rm  src/com/hzh/hibernate/bbb/ -r**  // **-r** 会把bbb/目录下的所有内容一次性移动。

------------------------------------------------------------------------

**远程创建了一个新仓库，本地创建了一个新项目，如何使新的项目与仓库对应起来？**

其实，这个也很简单，只是我当时对那些命令不太理解，所以比较模糊，不知如何对应。

**$ git remote add origin git@github.com:defnngj/hibernate-demo.git**

//还是这个命令，在你push项目之前加上这一句就OK了。

**git@github.com:defnngj/hibernate-demo.git** 就是你常见的新仓库的地址啊。git切换到新项目下，在push之前，加上这一句，我们创建的新仓库就与新项目建立了连接。

怎么样？掌握了上面的那些，你是不是就可以在项目中把玩一下了。Git很强大，这些只是冰山一角，不过对新手来说暂时够用了。

**Ps**：前两天聊天，一哥们说又发现个藏片的好地方，可以把岛国的小电影用这个管理，就不怕个人隐私被别人发现了尴尬。哈哈，你要不要试试。

[**Git命令小记**](http://www.cnblogs.com/cassvin/archive/2012/01/29/git_commands.html)

　　由于自己平常git用的不多不熟练，最近写个小东西并把代码托管到github，才发现之前看的《Pro Git》和《看日记学git》完全打水漂。重翻《Pro Git》，把一些重要的常见的命令记下来，备忘，具体的请man。

符号约定：

[]：可选　　<>：必选

Git 配置

git config [--global] user.name <name>　　　　　　　　设置用户名

git config [--global] user.email <email>　　　　　　 　　设置邮箱

git config [--global] core.editor <editor>　　　　　　　　设置编辑器

git config [--global] github.user <user>　　　　　　 　　设置github帐号名

git config [--global] github.token <token>　　　　　　　 设置github的token

--global是对当前系统用户的全局设置，在～/.gitconfig中。对系统所有用户进行配置，/etc/gitconfig。对当前项目，.git/config

Git 创建库

git clone <url>　　　　　　　　　　　　　　　　　　　ssh/http(s)/git三种协议，ssh和https可推送

git init　　　　　　　　　　　　　　　　　　　　　　  初始化Git仓库

Git 日常操作

git add <file>　　　　　　　　　　　　　　　  将文件加入index file

git rm [--cached]　　　　　　　　　　　　　   删除，加--cached表示仅从index file中删除文件，即放弃跟踪

git mv <src> <dest>　　　　　　　　　　　　 移动/更名

git diff --cached/--staged　　　　　　　　　　当前索引与上次提交（有哪些需要commit）

git diff　　　　　　　　　　　　　　　　　     当前索引与工作目录（有哪些需要add）

git diff HEAD[^]　　　　　　　　　　　　　　 工作目录与上次提交（当前目录与上次提交有何改变）

git commit [-a] -m <msg>　　　　　　　　　　提交

git commit --amend [-m <msg>]　　　　　　　修复上次提交

git reset HEAD <file>　　　　　　　　　　　  同--mixed，default option

git reset --mixed HEAD　　　　　　　　　　  撤销 commit 和index file,只保留 working tree 的信息

git reset --hard HEAD[^]　　　　　　　　　　 将 working tree 和 index file 都撤销到以前状态

git reset --soft HEAD[^]　　　　　　　　　　  只撤销 commit,而保留 working tree 和 index file 的信息

　　　　　　　　　　　　　　　　　　　　　回复到某个状态。以git reset --soft HEAD为例，commit回退到

　　　　　　　　　　　　　　　　　　　　　HEAD（相当于无变化），若是HEAD^，则commit回退到HEAD^

git gc　　　　　　　　　　　　　　　　　　   用垃圾回收机制清除由于 reset 而造成的垃圾代码

git status　　　　　　　　　　　　　　　　　 显示当前工作目录状态

git log [-p]　　　　　　　　　　　　　　　　   显示提交历史（many useful options to be learned）

git branch [branch]　　　　　　　　　　　　   显示/新建分支

git branch -d/-D　　　　　　　　　　　　　　 删除分支（d表示“在分支合并后删除分支”，D表示无论如何都删除分支）

git show-branch

git checkout <branch>　　　　　　　　　　　 切换分支（分支未commit无法切换）

git merge <branch>　　　　　　　　　　　　  合并分支

git merge == git pull .

git show <branch | commit | tag | etc>　　　     显示对应对象的信息

git grep <rep> [object]　　　　　　　　　　　  （在指定对象（历史记录）中）搜索

git cat-file 　　　　　　　　　　　　　　　　   查看数据

git cat-file <-t | -s | -e | -p | (type)> <object>        type can be one of: blob, tree, commit, tag

git ls-files [--stage]　　　　　　　　　　　　　 show information about files in the index and the working tree（实际是查看索引文件）

git watchchanged <since>..<until>　　　　　　 显示两个commit（当然也可以是branch）的区别

git remote [-v]　　　　　　　　　　　　　　    　　显示远程仓库，加-v选项可显示仓库地址

git remote add <repo\_name> <url>　　　　　  　　添加远程仓库，repo\_name为shortname，指代仓库地址

git remote rename <old\_name> <new\_name>  　　更名

git remote rm <repo\_name>　　　　　　　　　　  删除远程仓库

git remote show <repo\_name>　　　　　　　　　 查看远程仓库信息

git remote fetch <repo\_name>　　　　　　　　　  从远程仓库抓取数据（并不合并）

git pull <repo\_name> <branch\_name>　　　　　　拉去数据并合并到当前分支

git push <repo\_name> <branch\_name>　　　　　 推送指定分支到指定仓库

git fetch <repo\_name> <branch\_name>[:<local\_branch\_name>]　　　　拉去数据，未合并

Git 标签

git 标签相关……

Git 相关环境变量

GIT\_DIR: 如果指定了那么git init将会在GIT\_DIR指定的目录下创建版本库

GIT\_OBJECT\_DIRECTORY: 用来指示对象存储目录的路径。即原来$GIT\_DIR/objects下的文件会置于该变量指定的路径下

Git 常见变量

HEAD: 表示最近一次的 commit。

MERGE\_HEAD: 如果是 merge 产生的 commit,那么它表示除 HEAD 之外的另一个父母分支。

FETCH\_HEAD: 使用 git-fetch 获得的 object 和 ref 的信息都存储在这里,这些信息是为日后 git-merge 准备的。

HEAD^: 表示 HEAD 父母的信息

HEAD^^: 表示 HEAD 父母的父母的信息

HEAD~4: 表示 HEAD 上溯四代的信息

HEAD^1: 表示 HEAD 的第一个父母的信息

HEAD^2: 表示 HEAD 的第二个父母的信息

COMMIT\_EDITMSG: 最后一次 commit 时的提交信息。

### [git命令入门](http://blog.csdn.net/sfrysh/article/details/5956771)

分类： [版本管理工具](http://blog.csdn.net/sfrysh/article/category/745004)2010-10-21 15:36 1869人阅读 [评论](http://blog.csdn.net/sfrysh/article/details/5956771#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sfrysh/article/details/5956771#report)

**主题一 基本使用**

1：下载 ：git clone \*\*\*

2：本地与服务器端同步 ：git pull

3：本地的状态查询：git status

4：本地修改同步至服务器端 ：git commit -a -m "log\_message" (-a是提交所有改动，-m是加入log信息)

5：本地修改上传至服务器端 ：git push

**主题二 分支**

1:git branch                                            查看所有分支

2:git branch branch\_0.1 master                  从主分支master创建branch\_0.1分支

3:git branch -m branch\_0.1 branch\_1.0       将branch\_0.1重命名为branch\_1.0

4:git checkout branch\_1.0/master              切换到branch\_1.0/master分支

小窍门:3与4的一步实现方式为:git checkout -b branch\_1.0 master

5:git rebase branch\_1.0(前提假设已git checkout master)

在branch\_1.0与master分叉处后master的变更append至branch\_1.0(较难理解)

6:git branch -d branch\_1.0   删除branch\_1.0

**主题三 更补**

1:git commit -C HEAD --amend

使用最近commit增补提交，不产生新的commit log

2:git reset --hard HEAD~1

将版本库回复到HEAD之前的版本

### [git 命令详解（ 一） git 中文使用指南pdf下载](http://hlee.iteye.com/blog/399349)

**博客分类：**

* [RubyOnRails](http://hlee.iteye.com/category/54340)

[Git](http://www.iteye.com/blogs/tag/Git)[Ruby](http://www.iteye.com/blogs/tag/Ruby)[SVN](http://www.iteye.com/blogs/tag/SVN)[Gmail](http://www.iteye.com/blogs/tag/Gmail)[SSH](http://www.iteye.com/blogs/tag/SSH)

[git中文版使用指南pdf版](http://hlee.iteye.com/topics/download/00eaf9df-8746-3a59-ae73-2ec0f6fccbc2)   
  
我用git最开始是在github，每次提交代码以后，右上角的部分就会显示这些信息，类似于下面的：

**引用**

commit  fa6f27b7de063c2f301b0e7148b5bd5e813faa98   
tree       5e7a19c158b89fbc52a078771a833ee839727404   
parent   76f31606376180ca88efa12be341dbb14fb06fdf

咋一看，这40位的乱码挺吓人的，但是你了解它的作用就不会被吓到了。 这是object name，是作为你每次提交的信息标识。这是用SHA1加密hash函数根据你的对象的内容算出来的。Git的一些优点：   
  
Git通过简单比较object name就可以快速的确定这两个对象是否相等。   
因为object name在每个repository里都是以相同的方法来计算的，所以在不同的两个repository的同一个内容都会被存储在相同的object name下。   
Git可以在读取对象的时候检查错误，检查该对象的名称是否还是它的内容的SHA1 hash。   
 **The Objects**   
每个对象都由三个东东组成：type，size，content。size当然是content的size了，而content却是依赖于这个对象是个什么type了。而type又有四类：“blob”，”tree”, “commit”, “tag”。

**Ruby代码  [收藏代码](javascript:void())**

1. blob # 通常被用来存储文件数据，一般是个文件。
2. tree  # 基本上像个目录结构，比如x / yyy
3. commit # 它指向一个单独的tree，它标记了一个时刻的项目的面貌，就像给这个project照了个艳照一样。它包含那个时刻的元信息，例如时间戳，改变本次提交的作者，前一个提交的指向标等等。
4. tag # 专门用来指定某些特殊行为的commits。

和svn的不同是？   
传统的SCM系统，比如svn，CVS, perforce, Mercuial等，它们存储的都是本次和下次提交的differents。而Git不同，这是关键的一点。Git就像一个狗仔队的记者一样，把你项目的所有文件以及每次提交时候的tree 结构都拍了个照，把这个照片存储下来而已。这点对于理解Git来说，相当重要！   
 **Blob Object**   
刚才说了，这是指文件数据。   
git show 命令可以帮助你看type，具体用法可以参照（http://www.kernel.org/pub/software/scm/git/docs/git-show.html）例如： 

**Ruby代码  [收藏代码](javascript:void())**

1. git show
2. commit cf70117f3cc497fba42890171d29ae404061d25f
3. Author: Alex &lt;blackanger.z**@gmail**.com&gt;
4. Date:   Sun Dec 7 05:55:46 2008 +0800
6. modify .gitignore
8. diff --git a/.gitignore b/.gitignore
9. index deb5b7c..77573ab 100644
10. --- a/.gitignore
11. +++ b/.gitignore
12. @@ -1,2 +1,2 @@
13. -log/
14. +/log
15. .DS\_Store

因为blob是由它的数据决定的实体，如果在目录tree里有两个内容相同的blob，那么它们就会共享同一个blob对象。该对象是完全独立于它在目录tree里的位置的，并且你重命名文件的话也不会改变与这个文件相关联的那个blob对象。   
 **Tree Object**   
这是个简单的对象，它就是指向一堆blob对象和其他的tree目录。它一般表现的是目录和子目录的内容。你用git ls-tree命令可以把tree 对象看的更详细。例如：

**引用**

git ls-tree cf701   
100644 blob 77573ab40c4f010c0e050f1bbff2ad3ab78f1428       .gitignore   
100644 blob f8cd794a2d06e6c409737d5bcd39a20abb62db26    README   
100644 blob 3bb0e8592a41ae3185ee32266c860714980dbed7    Rakefile   
040000 tree ee99c7a2e9842acbab05fcd02ec510ea917218b2    app   
040000 tree 0aff84690f37df01a7cac96529657222cbeeebf6               config   
040000 tree 92aaeee021e9c7da661bac141ab55c17a3ae79ee    db   
040000 tree 0269300738b048a5cc34769d1436d9f228499018    doc   
040000 tree 1e384157d8bc3cd7aab5b0b5605e9a78d0612426    features   
040000 tree 9f87e761776449b7d43ea112573f8ae27faaf826    lib   
040000 tree 4875662d9a7e07a7b95b68958e518fdd65235124    public   
040000 tree 11741373949f86364c10af17db764d1af3bc048f               script   
040000 tree 20433690c23ae3a299b5a02147f68fdf94820370    spec   
040000 tree 646d8b1458d23ba0ec396ef5748205977b023f3f    stories   
040000 tree cc48cdef8eba2b5333c6a01d5795bb6efb43182f    test   
040000 tree e85767f4716fd20a53d7d6455972c6ec6571c833    vendor

cf701.。。是上一个tree对象，是整个app的tree对象，熟悉Rails的人就可以看得出来这个tree目录。   
  
请注意这些文件到mode为644或755，以保证git执行时候有足够到权限。   
  
**Commit Object**   
你可以用git show或git  log加–pretty=raw参数来检查你的提交，例如： 

**Ruby代码  [收藏代码](javascript:void())**

1. git show -s --pretty=raw cf701
2. commit cf70117f3cc497fba42890171d29ae404061d25f
3. tree 6ccb7edcdb14205861feb263a7607b8921c75fa3
4. parent 3e794f3e4dcb8d362bfb11464c849070bd04a35f
5. author Alex &lt;blackanger.z**@gmail**.com&gt; 1228600546 +0800
6. committer Alex <blackanger.z**@gmail**.com> 1228600546 +0800
8. modify .gitignore

看见没有？   
有个parent， 这基本上是每个commit都有的，但是当一个commit没有parent的时候，那么就叫root commit，这是初始化工程第一次提交才有的。每个project必然会有一个root commit，有的有多个root commit，不过很少见（project 套project ？）。   
  
author是这次提交的人。   
committer实际创建这次提交的人，它和author有可能不同，例如，我写的代码发email给你，你帮我commit的，但实际author并不是你，对不？   
comment是对这次提交的描述。这里没有我不知道为什么 。。。   
  
**Tag Object**   
到这里可以看到如何创建和验证一个object对象。   
http://www.kernel.org/pub/software/scm/git/docs/git-tag.html   
  
  
**Git Directory and Working Directory**   
  
**1. The Git Directory**   
git directory是存储你工程所有Git历史和元信息的目录。包括所有的对象，所有到不同分支的指向。   
每个project仅有一个git directory。这个目录就是你工程根目录下的那个.git。例如： 

**Ruby代码  [收藏代码](javascript:void())**

1. cd .git
2. ~/work/mars/.git&gt;ls
3. COMMIT\_EDITMSG    branches    description    index        logs        refs
4. HEAD        config        hooks        info        objects

**2.The Working Directory**   
工作目录自然就是你当前检出文件的目录了，大白话就是项目的目录，你要在这里面工作的。。。   
  
**The Git Index**   
  
如果把你当前修改过的这些文件比作是你从你的部队你精选出来的突击队的话，把commit比作深入敌后的话，那么这个git index就是你这支突击队在深入敌后之前的排练场，你需要在这里点兵，看看有没有怕死的逃兵漏掉。因为你要提交的只是你修改的文件，而不是全部文件，所有你只需要有一个修改文件集合就可以了。   
用这个命令来看： 

**Ruby代码  [收藏代码](javascript:void())**

1. git status
2. # On branch master
3. nothing to commit (working directory clean)

我距离上次提交没有改动过任何文件，自然不会出现什么文件列表了。我无耻的抄个例子：

**Ruby代码  [收藏代码](javascript:void())**

1. **$git** status
2. # On branch master
3. # Your branch is behind 'origin/master' by 11 commits, and can be fast-forwarded.
4. #
5. # Changes to be committed:
6. #   (use "git reset HEAD &lt;file&gt;..." to unstage)
7. #
8. #   modified:   daemon.c
9. #
10. # Changed but not updated:
11. #   (use "git add &lt;file&gt;..." to update what will be committed)
12. #
13. #   modified:   grep.c
14. #   modified:   grep.h
15. #
16. # Untracked files:
17. #   (use "git add &lt;file&gt;..." to include in what will be committed)
18. #
19. blametree
20. blametree-init
21. git-gui/git-citool

用过svn的人也很容易懂把。   
  
Setup and Initialization   
  
安装就不说了，用mac或linux的人都会安装，用windows的人我只能无耻的抄一句别人的话来骂你：   
who fucking need to run a separate software just for using git。   
 **Git Config**

**Ruby代码  [收藏代码](javascript:void())**

1. $ git config --global user.name "Alex Zhang"
2. $ git config --global user.email “blackanger.z**@gmail**.com"

然后你   
vi ~/.gitconfig   
就会看到：   
[user]   
name = Alex Zhang   
email = blackanger.z@gmail.com   
  
如果你想为指定的项目用别的git 配置，那么你可以使用git config命令来修改你的信息，不需要加－－global参数。   
  
**Getting a Git Repository**   
那么现在我需要一个Git仓库了。怎么办？   
  
两个办法：   
克隆一个已经存在的。   
初始化一个新的。   
  
如果你要clone的话，你需要知道一个project的git url。git操作跨越多个协议，例如：

**Ruby代码  [收藏代码](javascript:void())**

1. git url  ：  git clone git://git.kernel.org/pub/scm/git/git.git
2. #或者
3. http     ：  git clone http://www.kernel.org/pub/scm/git/git.git

请记住git:// 协议是更快的。但是当你有防火墙或其他原因的时候，还得用http。   
  
**Initializing a New Repository**   
假如你有个工程叫project，那么：

**Ruby代码  [收藏代码](javascript:void())**

1. cd project
2. git init
3. #则会输出：
4. Initialized empty Git repository **in** .git/

你可以去这里看git的视频：   
http://www.gitcasts.com/   
  
**Normal Workflow**   
一般的工作流程就是：   
1.增加一些新的文件   
$ git add file1 file2 file3   把新文件加到git index里。   
2.  git diff –cached   
你可能需要显示你刚add到git index里的那些改变。如果没有－－cached，那么就不会显示刚add到index的改变。 你也可以用git status来看。

**Ruby代码  [收藏代码](javascript:void())**

1. git commit  #你可能需要提交了。
2. git commit -a #把新近的改变加到index里顺便提交。

为每次commit写message是最佳实践。   
svn或其他的版本控制工具也有add方法，它们只是把新加的文件加到index里来跟踪。但是git的add更加强大， git add，不仅仅是增加新文件到index里，还包括最新的修改。   
  
**Basic Branching and Merging**   
一个单一的git仓库就可以提供多个分支给开发者。创建一个新的分支： 

**Ruby代码  [收藏代码](javascript:void())**

1. git  branch experimental
2. <pre lang="bash" line="1">
3. #如果你现在运行：
4. <pre lang="bash" line="1">
5. git branch

你就会得到一个所有分支的列表。

**Ruby代码  [收藏代码](javascript:void())**

1. experimental
2. \* master

experimenta是刚刚创建的那个，那个master是默认的分支。星号标识的是你当前所处的分支。 

**Ruby代码  [收藏代码](javascript:void())**

1. git checkout experimental

来选择那个experimental的分支内容。如果修改了一个文件，commit这个改变，并且选择回到master 分支：   
(edit file) 

**Ruby代码  [收藏代码](javascript:void())**

1. $ git commit -a
2. git branch
4. #输出

7. \* experimental
8. master
10. $ git checkout master
11. git branch
13. #输出：


17. experimental
18. master

然后你这master分支里做了一个不同的改变，然后提交。 那么现在master和experimental这两分支里都有了不同的修改，那么我们来合并这些不同：   
git merge experimental   
注意，你现在是这master里，把其他分支的不同合并到master里来。   
如果没有冲突还好，那么有冲突怎么办？   
git diff 呀   
  
查看不同，然后解决冲突，再提交一次。   
  
命令行里输入：   
gitk   
会出来一个x－windows工具，很挫的，截图给大家看，太难看，github上有个好看的工具介绍，我忘了名字了。   
  
来看看如何解决冲突： 

**Ruby代码  [收藏代码](javascript:void())**

1. $ git merge **next**
2. 100% (4/4) done
3. Auto-merged file.txt
4. CONFLICT (content): Merge conflict **in** file.txt
5. Automatic merge failed; fix conflicts **and** **then** commit the result.

merge的时候有冲突，会有警告信息，这可真墨迹。   
  
git status来看冲突的地方：   
<<<<<<< HEAD:file.txt   
Hello world   
=======   
Goodbye   
>>>>>>> 77976da35a11db4580b80ae27e8d65caf5208086:file.txt   
那么你需要编辑文件来解决冲突，然后： 

**Ruby代码  [收藏代码](javascript:void())**

1. $ git add file.txt
2. $ git commit

具体可以看gitcasts的视频六：GitCast #6: Branching and Merging   
  
查看历史：

**Ruby代码  [收藏代码](javascript:void())**

1. git log #版本号

加各种参数可以让log格式化输入便于阅读。   
  
以上都是简单的情况。 大体命令都去查文档吧。   
  
Distributed Workflows   
麻烦的情况怎么处理呢？   
Alice有个新的工程，存储这一个git仓库：/home/alice/project， 但是Bob，在同一台机器上也有一个home目录。   
  
Bob： 

**Ruby代码  [收藏代码](javascript:void())**

1. $ git clone /home/alice/project myrepo

创建了一个新的project，叫myrepo。   
Bob修改了一些文件，然后提交它们：   
(edit files) 

**Ruby代码  [收藏代码](javascript:void())**

1. $ git commit -a
2. (repeat as necessary)

当他准备好以后，他告诉Alice去更新这些改变从他的项目里：/home/bob/myrepo   
  
那么Alice她是这么做的： 

**Ruby代码  [收藏代码](javascript:void())**

1. $ cd /home/alice/project
2. $ git pull /home/bob/myrepo master #（此命令，master参数不是必须的）

这个操作把bob的master分支合并到了Alice的当前的master分支里。但是如果此时Alice有自己的改变，有可能会出现冲突。因此pull命令是两个操作：   
从远程分支取一些改变。   
merge到本地当前分支。   
  
为了避免这种麻烦，我们可以这么做： 

**Ruby代码  [收藏代码](javascript:void())**

1. $ git remote add bob /home/bob/myrepo

然后先取回来bob的修改，但是并不去merge

**Ruby代码  [收藏代码](javascript:void())**

1. $ git fetch bob

然后看看日志：

**Ruby代码  [收藏代码](javascript:void())**

1. $ git log -p master..bob/master

然后就会显示bob的所有改变，再检查完这些改变以后，Alice就可以合并这些改变到她的分支里了：

**Ruby代码  [收藏代码](javascript:void())**

1. $ git merge bob/master

这一步也可以这么做： 

**Ruby代码  [收藏代码](javascript:void())**

1. $ git pull . remotes/bob/master

之后， Bob更新Alice最新的改变就可以用： 

**Ruby代码  [收藏代码](javascript:void())**

1. $ git pull

没有alice的路径是因为，是bob clone的alice的代码。你可以用这个命令查看：

**Ruby代码  [收藏代码](javascript:void())**

1. $ git config --get remote.origin.url
2. /home/alice/project

Git也可以做一份Alice master分支的原始拷贝在“origin/master”下：

**Ruby代码  [收藏代码](javascript:void())**

1. $ git branch -r
3. #输出
4. origin/master

**Public git repositories**   
使用一个公共的仓库，这是通用的做法，这样可以把一些私有的工作过程清晰的和版本库清晰的分离出来。你只需要在你私人的仓库里完成你的工作，然后把你做的改变提交到你的那个公共仓库里，别的开发者可以从你的公共仓库里更新代码。同样，别的开发者push他的改变到他的公共仓库，你从他的公共仓库获取更新。

**引用**

you push   
your personal repo ——————> your public repo   
^                                     |   
|                                     |   
| you pull                            | they pull   
|                                     |   
|                                     |   
|               they push      V   
their public repo <——————- their repo

**Pushing changes to a public repository** 

**Ruby代码  [收藏代码](javascript:void())**

1. $ git push ssh://yourserver.com/~you/proj.git master:master
2. #或者是
3. $ git push ssh://yourserver.com/~you/proj.git master

当然你可以设置.git/config:

**Ruby代码  [收藏代码](javascript:void())**

1. $ cat >>.git/config <<EOF
2. [remote "public-repo"]
3. url = ssh://yourserver.com/~you/proj.git
4. EOF

这样你就可以使用命令： 

**Ruby代码  [收藏代码](javascript:void())**

1. $ git push public-repo master
2. </pr>
4. #以GitHub为例子， 保存在github上的一个项目， 我查看它的config：
5. [remote "origin"]
6. url = git**@github**.com:blackanger/mars.git
7. fetch = +refs/heads/\*:refs/remotes/origin/\*
9. #这样，我每次往github push代码就是用命令：
10. git push origin master
12. #详细的可以去看 www.gitcasts.com
14. <em><strong>Git Tag</strong></em>
16. 1.Lightweight Tags
17. ~/work/mars&gt;git tag stable-1 cf701
18. ~/work/mars&gt;git show stable-1
19. commit cf70117f3cc497fba42890171d29ae404061d25f
20. Author: Alex &lt;blackanger.z**@gmail**.com&gt;
21. Date:   Sun Dec 7 05:55:46 2008 +0800
23. modify .gitignore
24. 。。。

使用tag和一个object关联起来。这是创建一个轻量级的tag。   
  
**Tag Objects**   
如果加上－a，－s或－u &lt;key-id&gt;参数，则是创建一个tag对象。   
<pre lang="bash" line="1">

**Ruby代码  [收藏代码](javascript:void())**

1. $ git tag -a stable-1 1b2e1d63ff