**Git使用**

Created by：蔡观洋

Time：S-2014.01.21 E-2014 M-2014

Type：工具使用

**目录：**

[1. 基本概念： 2](#_Toc378147460)

[1.1 Git的设计目标 2](#_Toc378147461)

[1.2 Git的特征 2](#_Toc378147462)

[2. 基本配置： 3](#_Toc378147463)

[2.1 常用命令 3](#_Toc378147464)

[2.2 用户信息 3](#_Toc378147465)

[2.3 文本编辑器 3](#_Toc378147466)

[2.4 修改默认主目录 3](#_Toc378147467)

[3. 常用操作： 4](#_Toc378147468)

[3.1 取得项目Git仓库 4](#_Toc378147469)

[3.2 常用命令 4](#_Toc378147470)

[3.3 忽略部分文件 5](#_Toc378147471)

[3.4 命令详解 5](#_Toc378147472)

[3.4.1 git commit 5](#_Toc378147473)

[4.4.2 git diff 5](#_Toc378147474)

[3.4.3 git log 5](#_Toc378147475)

[3.5 分支 5](#_Toc378147476)

[3.6 打标签 5](#_Toc378147477)

[3.7 状态说明 5](#_Toc378147478)

[4. 远程Git仓库交互 6](#_Toc378147479)

[4.1 克隆远程仓库 6](#_Toc378147480)

[4.2 常用命令 6](#_Toc378147481)

[5. 工作流： 6](#_Toc378147482)

[6. 与GitHub交互 6](#_Toc378147483)

[6.1 准备工作 6](#_Toc378147484)

[6.2 Git客户端通过SSH与GitHub连接 6](#_Toc378147485)

[6.3 SSH介绍 7](#_Toc378147486)

[7. 参考文献： 7](#_Toc378147487)

1. 基本概念：

Git是一个开源的分布式版本控制系统，用以有效、高速的处理从很小到非常大的项目版本管理[1]。

* 1. Git的设计目标

速度

简单的设计

对非线性开发模式的强力支持

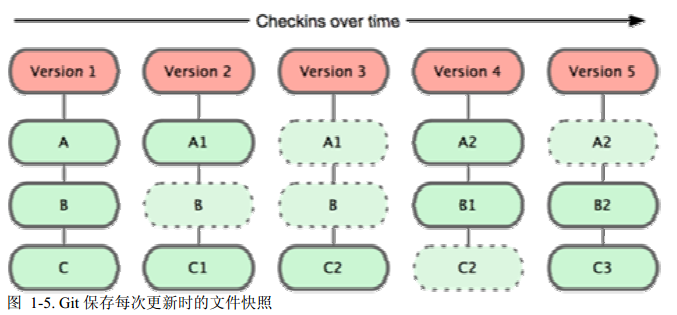
完全分布式

有能力高效的管理类似Linux内核一样的超大规模项目（速度和数据量）

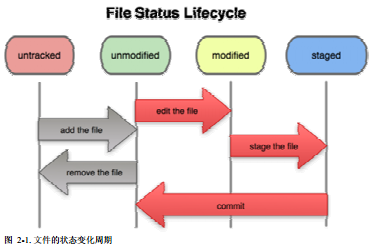
* 1. Git的特征

1. 和其他CVS主要区别是Git只关心文件数据的整体是否发生了变化，如果发生了变化，会创建文件的一份快照保存到GIT中，供后期的版本恢复和对比使用。

Git采用一个160位字符作为文件的指纹（SHA-1算法）来判断文件是否发生了变化。



1. 作为分布式的版本控制软件，每个用户都克隆了所有信息，包括副本和历史信息。因此当一个节点出现问题时，可以采用其他节点来恢复版本控制信息。避免了C-CVS的单点故障问题。
2. 任何一个文件，在Git内部只有三种状态：已提交（commited），已修改（modified）和已暂存（staged）。Git文件提交时状态转换过程如下：



文件在提交之前，是先放到一个暂存列表中的。提交动作则是将暂存列表中的文件保存到Git中。

1. 基本配置：
   1. 常用命令
2. Git config --list 列出所有的配置参数
3. Git help <verb> 查看帮助文档
4. Git config … 配置参数
   1. 用户信息

Git config --global user.name “cai guan yang”

Git config --global user.email [test@126.com](mailto:test@126.com)

当提交时，会将设置的用户信息一块保存到Git中，查看提交历史时可以看到时哪些用户更改了文件。

使用--global 选项配置的是全局的参数，保存在git主目录中的.gitconfig文件中，对所有的项目都使用此信息，如果去掉--global选项， 则仅在当前项目中采用此用户信息，保存在项目.git/config文件中。

* 1. 文本编辑器

Git config --global core.editor emacs

指定一个默认的文本编辑器，当git要求用户输入一些文本信息时会默认调用此程序

* 1. 修改默认主目录

打开Git安装位置\etc\profile文件，找到[2]

# normalize HOME to unix path

HOME="$(cd "$HOME" ; pwd)"

export PATH="$HOME/bin:$PATH"

增加两行，修改后结果如下：

# normalize HOME to unix path

HOME="你想要修改的HOME路径"

HOME="$(cd "$HOME" ; pwd)"

cd

export PATH="$HOME/bin:$PATH"

再次启动Git bash，就会自动进入新修改后的HOME路径了，之后新配置的.SSH文件夹也为在新路径中。

若是Windows系统，为了方便起见可以顺手加上一个变量名用于自定义项目目录，此做法不但便于路径跳转，也可以在启动git时自动进入项目目录。方法极为简单，只需在 profile文件中加入:

# set Project Path

proj="你的项目文件目录"

cd $proj

如代码所「言」，可使用cd $proj命令即可跳转到项目目录，也可以按需定义其他变量。

如果您对bush非常了解，应该能猜到linux的bash的配置方案理应也在这行得通。而事实也的确如此，有profile就有bashrc。

在HOME目录下新建文件.bashrc写入你最惯用的配置，比如alias、bind、export 等，执行source ~/.bashrc即刻生效。

2.5 中文乱码

主要指git bash乱码 git status中文乱码。

直入正题，安装完git后，默认对中文支持是灰常有限的，为了支持中文需要做一系列配置：

1、Git Bash 中输入 ls 命令，可以正常显示中文**文件名**：

修改%Gi%t\etc\git-completion.bash，如果没有则在末尾添加下面一行：

**alias ls='ls --show-control-chars --color=auto'**

2、在Git Bash 中可以正常**输入**中文：

修改%Git%\etc\inputrc，修改配置项如下：

**set output-meta on**

**set convert-meta off**

3、Git Bash中查看日志是能正常显示中文：

修改%Git%\etc\profile，在末尾添加如下一行：

**export LESSCHARSET=utf-8**

4、在git gui 中正常显示代码中的中文：

修改%Git%\etc\gitconfig，添加如下配置项（如果代码文件是gb2312编码，就填gb2312）

**[gui]  
encoding = utf-8**  
5、正常显示推、拉中文修订说明

修改%Git%\etc\gitconfig，添加如下配置项

**[i18n]  
commitencoding = GB2312**

说明：若没有此项存在两个问题，一、我们推到服务器的中文修订说明会变成乱码；二、我们从服务器拉下来的别人推上去的中文修订说明也会是乱码。  
Git Status命令乱码：

在C:Gitetcgitconfig文件中修改或添加如下配置：  
[core]  
quotepath = false  
作用：没有这一条，$git status输出中文会显示为UNICODE编码。

1. 常用操作：
   1. 取得项目Git仓库
2. 从当前目录初始化：

进入一个目录，执行”git init”， 会在当前目录中添加目录.git，包含Git用到的数据和资源；

1. 克隆已知仓库：

执行”git clone <URL> [new name]”，将远程的文件都拷贝到本地。

* 1. 常用命令

1. git add … 添加跟踪/转到暂存（依据当前文件的状态，采用不同的动作）
2. git commit –m 提交，并指定提交说明
3. git rm [-f] file 删除git中和本地工作目录中的file
4. git rm --cached file 删除git中的文件，但是不删除本地目录中的file（用于误上传了部分临时文件，又不想删除本地备份）

（执行删除命令后，需要再执行commit命令）

1. git status 查看工作目录内文件的状态
2. git diff 比较文件的不同
3. gitk 打开图形化界面，可以查看版本信息
4. git log 查看提交历史
5. git mv file\_from file\_to 给文件改名；相当于执行了三条命令（mv file\_from file\_to; git rm file\_from; git add file\_to）
6. git reset HEAD file 取消暂存的某个文件
7. git checkout -- file 撤销对本地文件的修改，由于是用git中的历史版本来覆盖本地的文件，所以要慎重使用；
   1. 忽略部分文件

当使用git status命令查看目录中的部分文件时，用户可以在项目目录中添加文件.gitignore，标示不需要追踪查看的文件名信息，支持glob模式（简单的正则表达式）。

如：



* 1. 命令详解

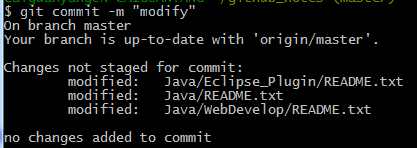
3.4.1 git commit

1. git commit 会弹出默认编辑器，让用户输入提交说明
2. git commit –m “comment” 在提交的同时，指明提交说明
3. git commit –a –m 跳过暂存区域，直接将已跟踪且修改过的文件提交
4. git commit –-amend –m “modify the latest commit” 修改最后一次提交的信息，但是自从上次提交后，中间不能有任何操作；
   * 1. git diff

3.4.3 git log

* 1. 分支
  2. 打标签
  3. 状态说明

1. Changes not staged for commit



1. 远程Git仓库交互
   1. 克隆远程仓库

执行命令：git clone [git@github.com:username/rep.git](mailto:git@github.com:username/rep.git) localFolder

会将远程仓库rep.git中的内容拷贝到本地文件夹localFolder中。

**问题：**

1. 与GitHub连接时需要使用SSH秘钥对，当有多个秘钥对时它怎么识别用哪个
   1. 常用命令

git remote –v 列出此拷贝对应的URL信息

git remote add [shotname] [url] 使用一个简单名字来引用url对应的仓库

git fetch [remote-name] 抓取从你上次克隆到现在别人上传的代码到本地

git pull [remote-name] 获取更新

git push [remote-name] [branch] 将本地的分支推送到远程仓库中

git remote show [remote-name] 查看远程仓库的信息

1. 工作流：
2. 与GitHub交互
   1. 准备工作
3. 安装git客户端：msysgit

地址：<http://msysgit.github.io/>

1. 在github上注册账户
   1. Git客户端通过SSH与GitHub连接
2. 创建SSH秘钥

打开git bash, 输入一些命令：

ssh-keygen –C ‘your@email.address’ –t rsa

注意使用默认目录。参考前面的介绍设置主目录。

1. 提交SSH秘钥到GitHub

回到github的页面上，在右上方工具栏里找到Account Settings。在这个页面上有一个SSH Public Keys标签，选择Add another public key。Title可以任意填写，Key是刚才生成的ssh public key。在刚才创建密匙的那个目录下（默认是C:\Documents and Settings\windows用户名\.ssh）找到id\_rsa.pub文件，把文件内容拷贝并粘贴到github页面key的空白处。然后Apply，就好了。

1. 测试连接：

输入一下命令测试：SSH –v [git@github.com](mailto:git@github.com)

会要求输入刚才设置的密码（空密码除外）

1. 然后可以使用远程仓库命令与GitHub中的仓库交互即可；

（注意：由于是使用SSH连接，因此连接的地址也应是SSH模式的，格式如git@github.com:username/repo）

* 1. SSH介绍

[SSH](http://baike.baidu.com/subview/16184/5909253.htm) 为 [Secure Shell](http://baike.baidu.com/view/2118359.htm) 的缩写，SSH 为建立在应用层和传输层基础上的安全协议。SSH 是目前较可靠，专为[远程登录](http://baike.baidu.com/view/59099.htm" \t "_blank)会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。

**SSH验证的方式：**

从[客户端](http://baike.baidu.com/view/930.htm" \t "_blank)来看，SSH提供两种级别的安全验证。

**第一种级别（基于口令的安全验证）**

只要你知道自己帐号和口令，就可以登录到远程主机。所有传输的数据都会被加密，但是不能保证你正在连接的服务器就是你想连接的服务器。可能会有别的服务器在冒充真正的服务器，也就是受到“中间人”这种方式的攻击。

**第二种级别（基于密匙的安全验证）**

需要依靠[密匙](http://baike.baidu.com/view/1489402.htm)，也就是你必须为自己创建一对密匙，并把公用密匙放在需要访问的服务器上。如果你要连接到SSH服务器上，客户端软件就会向服务器发出请求，请求用你的密匙进行安全验证。服务器收到请求之后，先在该服务器上你的主目录下寻找你的公用密匙，然后把它和你发送过来的公用密匙进行比较。如果两个密匙一致，服务器就用公用密匙加密“质询”（challenge）并把它发送给客户端软件。客户端软件收到“质询”之后就可以用你的私人密匙解密再把它发送给服务器。

用这种方式，你必须知道自己密匙的口令。但是，与第一种级别相比，第二种级别不需要在网络上传送口令。

第二种级别不仅加密所有传送的数据，而且“中间人”这种攻击方式也是不可能的（因为他没有你的私人密匙）。但是整个登录的过程可能需要10秒。

1. 参考文献：

[1] Pro. Git 简体中文版 PDF图书

[2] <http://poetcoder.com/2013/03/21/modify-git-root-direction-on-windows/>

[3]