**Github使用报告**

1. **前言**

此使用报告是基于对《Pro Git》的阅读，网上资料的搜集，再加上自身配置安装使用github的过程后得来。原文电子书地址为<http://git-scm.com/book>。

第二小节将会讲述Git的基本特性-版本控制，然后再介绍它与其他版本控制系统的几点不同之处，突出它的优势与特殊性。

第三小节将会讲述Github的安装配置，首先讲述基于源代码的安装，再着重讲述我们在windows上的安装以及使用。

第四小节将会讲述小组在使用Github时的一些心得体会。

1. **Git特性/原理**
2. **版本控制**

版本控制系统（VCS，即Version Control System）是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。它可以将某个文件回溯到之前的状态，甚至将整个项目都回退到过去某个时间点的状态。你可以比较文件的变化细节，查出最后是谁修改了哪个地方，从而找出导致怪异问题出现的原因，又是谁在何时报告了某个功能缺陷等等。

最初所使用的是本地版本控制系统，而这个系统中最流行的一种叫RCS，它的工作原理基本上就是保存并管理文件补丁。文件补丁是一种特定格式的文本文件，记录着对应文件修订前后的内容变化。所以，根据每次修订后的补丁，rcs 可以通过不断打补丁，计算出各个版本的文件内容。

当然，仅仅进行本地版本控制是不足以令进行共同开发的人们方便起来，于是就有了集中化的版本控制系统（CVCS，即Centralized Version Control Systems）。它有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。集中化版本控制的最大优势就是每个人都可以在一定程度上看到项目中的其他人正在做些什么。而管理员也可以轻松掌控每个开发者的权限，并且管理一个 CVCS 要远比在各个客户端上维护本地数据库来得轻松容易。当然，这样做也有缺点，因为它本质上还是将所有的数据集中在同一个地方。如果中央服务器发生单点故障，那么就会影响到所有的数据，损失数据的风险极大。

为了弥补这个缺点，又有了分布式版本控制系统（DVCS，即Distributed Version Control System）。在这类系统中，客户端并不只提取最新版本的文件快照，而是把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。更进一步，许多这类系统都可以指定和若干不同的远端代码仓库进行交互，这在以前的集中式系统中是无法实现的。另外，在DVCS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。

Git属于分布式版本控制系统，所以它拥有着版本控制的基本功能，又有着上述优势，而没有单点存储数据的缺陷。

1. **数据完整性**

在保存到 Git 之前，所有数据都要进行内容的校验和（checksum）计算，并将此结果作为数据的唯一标识和索引。这项特性作为 Git 的设计哲学，建在整体架构的最底层。所以如果文件在传输时变得不完整，或者磁盘损坏导致文件数据缺失，Git 都能立即察觉。Git 使用 SHA-1 算法计算数据的校验和，通过对文件的内容或目录的结构计算出一个 SHA-1 哈希值，作为指纹字符串。该字串由 40 个十六进制字符（0-9 及 a-f）组成，看起来就像是：

24b9da6552252987aa493b52f8696cd6d3b00373

Git 的工作完全依赖于这类指纹字串，所以你会经常看到这样的哈希值。实际上，所有保存在 Git 数据库中的东西都是用此哈希值来作索引的，而不是靠文件名。

1. **文件快照**

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。这类系统（CVS，Subversion，Perforce，Bazaar 等等）每次记录有哪些文件作了更新，以及都更新了哪些行的什么内容。

Git 会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。当然，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。

这是 Git 同其他系统的重要区别。它完全颠覆了传统版本控制的套路，并对各个环节的实现方式作了新的设计。在接下来的分支特性中，我们可以看到更多的有关这个区别的好处所在。

1. **分支**

在 Git 中提交时，会保存一个提交（commit）对象，该对象包含一个指向暂存内容快照的指针，包含本次提交的作者等相关附属信息，包含零个或多个指向该提交对象的父对象指针：首次提交是没有直接祖先的，普通提交有一个祖先，由两个或多个分支合并产生的提交则有多个祖先。

当使用 git commit 新建一个提交对象前，Git 会先计算每一个子目录（本例中就是项目根目录）的校验和，然后在 Git 仓库中将这些目录保存为树（tree）对象。之后 Git 创建的提交对象，除了包含相关提交信息以外，还包含着指向这个树对象（项目根目录）的指针。

Git 中的分支，其实本质上仅仅是个指向 commit 对象的可变指针。Git 会使用 master 作为分支的默认名字。使用 git branch 命令可以新建一个分支指针，它会即刻指向最后一次提交对象。除此外，Git保存着一个名为 HEAD 的特别指针，它指向你正在工作中的本地分支。

每一次提交，HEAD指针及当前指针（假设为A）都会随之向前移动，而其余指针不动。如果此时将HEAD指向其他指针（假设指向master）的对象，那么工作目录中的文件就会改变为master指向的对象。若此时再行提交，master分支上的文件就会被改动。这些改变分别孤立在不同的分支里，这意味着我们可以在不同分支里反复切换，并在时机成熟时把它们合并到一起。

Git的分支与其他版本控制系统不同的地方在于，由于 Git 中的分支实际上仅是一个包含所指对象校验和（40 个字符长度 SHA-1 字串）的文件，所以创建和销毁一个分支就变得非常廉价。说白了，新建一个分支就是向一个文件写入 41 个字节（外加一个换行符）那么简单了。这和大多数版本控制系统形成了鲜明对比，它们管理分支大多采取备份所有项目文件到特定目录的方式，所以根据项目文件数量和大小不同，可能花费的时间也会有相当大的差别，快则几秒，慢则数分钟。而 Git 的实现与项目复杂度无关，它永远可以在几毫秒的时间内完成分支的创建和切换。同时，因为每次提交时都记录了祖先信息），将来要合并分支时，寻找恰当的共同祖先的工作其实已经自然而然地摆在那里了，所以实现起来非常容易。

1. **Github安装/使用**
2. **从源代码安装**

（注，由于没有使用此方法进行github安装，所以此处摘抄了《Pro Git》原文的安装方法。关于配置信息由于篇幅限制则省略，具体可以参考《Pro Git》原文的配置方法。）

若是条件允许，从源代码安装有很多好处，至少可以安装最新的版本。Git 的每个版本都在不断尝试改进用户体验，所以能通过源代码自己编译安装最新版本就再好不过了。有些 Linux 版本自带的安装包更新起来并不及时，所以除非你在用最新的 distro 或者 backports，那么从源代码安装其实该算是最佳选择。

Git 的工作需要调用 curl，zlib，openssl，expat，libiconv 等库的代码，所以需要先安装这些依赖工具。在有 yum 的系统上（比如 Fedora）或者有 apt-get 的系统上（比如 Debian 体系），可以用下面的命令安装：

$ yum install curl-devel expat-devel gettext-devel \

openssl-devel zlib-devel

$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext \

libz-dev libssl-dev

之后，从下面的 Git 官方站点下载最新版本源代码：

http://git-scm.com/download

然后编译并安装：

$ tar -zxf git-1.7.2.2.tar.gz

$ cd git-1.7.2.2

$ make prefix=/usr/local all

$ sudo make prefix=/usr/local install

现在已经可以用 git 命令了，用 git 把 Git 项目仓库克隆到本地，以便日后随时更新：

$ git clone git://git.kernel.org/pub/scm/git/git.git

1. **在windows上安装/使用**

首先，我们从<https://github.com/>上下载最新的Github for windows的安装包，安装过程中在上述网站中进行账号注册的工作。安装完毕后，启动Github客户端即可使用。

接下来是使用方法。首先我们进入客户端后填写用户名密码登陆，随后点击create创建数据仓库。填写好仓库名后，一定要正确选择工程目录。然后在客户端中打开刚创建的仓库，可以看到工程中的源码，这时候要在左侧的summary栏中填写相关的代码信息，之后点击下面的Commit to master，再点击右上角的publish，就可以完成代码的共享了。

当我们在工作中修改了代码，需要进行共享的时候，我们要先打开该仓库，然后选中需要共享部分的代码，在左侧的Uncommitted changes栏中填写相关修改的信息供他人参考，再点击Commit to master，最后点击右上角的sync进行同步。

在上一小节中我们提到了节点的作用。在Github for windows中，如果要创建/改变节点，可以点击右上角的master按钮，在下拉菜单中切换/创建节点。

当然，最重要的功能还是合作开发，所以我们需要将合作开发者添加进此项目中。添加队友需要使用网页版Github。在网页上登陆后，我们先进入需要共享的库中，之后点击右侧的Settings，然后再点击Collaborators，再在右侧的框中输入队友的github账号，再点击它旁边的Add collaborator就可以完成队友添加工作。

**注：**如果要在 Windows 命令行中使用 Git，就必须要注意到，在参数中间有空格的时候，必须使用双引号将参数括起来（在 Linux 中是单引号）；另外，如果扬抑符（^）作为参数的结尾，并且作为这一行的最后一个字符，则这个参数也需要用双引号括起来。

1. **Github使用心得**

Github作为一个在Linux下开发的分布式版本管理系统，其实在这以前我都没有接触过，原因无非就是两个，一个参与的代码工作量不大，或者是即使代码工作量大，也不太需要这样一个管理系统来线上实时更改和存储进度，另外我平常所使用的操作系统多为windows 7，所以这个Linux下的系统总有很多不便之处，比如在windows下版本更新可能没有那么及时，一些缺陷或者BUG可能修改得也比较缓慢。基于这两个主观与客观上的原因，我便一直没有使用这个号称是全球最大的社交编程及代码托管网站。

当然，以前不使用的原因在现阶段越来越大的代码量和越来越迫切的合作需求之下不复存在，何况Github for windows的图形界面也算是简洁大方，在使用过后自然令人不由自主地喜欢上这个简单高效的管理系统。

虽然使用次数不能算是多，但是有一个很明显的算不上BUG的错误就是，假如路径名里含有中文的话，在Github的快照上便会显示乱码，虽然不影响对文件快照的储存，但是这还是给在图形界面上想要快速浏览数据变化的使用者带来了一定的不便。我自己对此便深有感触，刚开始使用的时候因为有中文路径而看着快照上的一长串数字路径名摸不着头脑，还得直接从本地的库中打开才知道相应的修改是什么，这给了运用中文文档来进行开发的我们很大的不便，希望Github for windows的开发者能够对此进行相应的改善工作。

除去这个小小的瑕疵外，Github的用户体验还是很令人满意的，虽然可能鉴于网速原因，有些改动不能够及时接收或是提交，但总体感觉还是不错的。简洁的图形分块，左侧的更改记录一览无遗，右侧的数据快照又可以让你不必费心去打开文件细看就可以了解到具体的改动，快照上方的修改者、修改标题、备注等能够让我们快速地了解到我们想要得到的信息，每个操作图标也很直观，且符合大多数人的操作习惯，不会说因为是一个新接触的版本管理系统而难以上手。

Github for windows给我的感受还是十分良好的，它的图形界面应该能够给大多数windows用户很好的体验，所以应该是一次成功的发展。见微知著，Github作为现今流行的分布式版本管理系统，在linux下的名声应当是不负其实的，而它强大的功能更不应只局限于作为VCS供人使用，相信我在未来的学习生活中可以体会到它更大的魅力。