NEU-RTES

Git 使用指南（V1）

Litary

2009-7-31

前言

为什么需要版本控制

对于每一个程序员来讲，怎样能够最好的保护你的源代码都是最重要的一件事。一个源码控制软件就相当于一个系统级的 Undo 键，当你的程序可用时，作一个标记，当程序被改的体无完肤不能运行了，你可以方便的回到上一个可用的版本，或者当你发现自己想找回上周被你删除的一个函数的时候，你可以很容易的做到这一点。

当然，好的版本控制并不止于此。对于团队开发来讲，你可以控制每个人的分工和权限。两个人可以同时编辑同一份源代码，由程序来提醒两次修改中存在的冲突。你可以查看某一处改动是谁做的。当你发布了一个正式版，可以建立一个分支，在分支上继续开发下一个版本，而对于后来发现的 Bug，可以在主分支上继续改进，如果分支上同样存在这个 Bug，可以将两者合并。

什么是版本控制

所谓版本控制系统(Version Control System)，从狭义上来说，它是软件项目开发过程中用于储存我们所写的代码所有修订版本的软件，但事实上我们可以将任何对项目有帮助的文档交付版本控制系统进行管理。

什么是 Git

非常简单地说，Git 一个快速、可扩展的分布式版本控制系统，它具有极为丰富的命令集，对内部系统提供了高级操作和完全访问。所谓版本控制系统(Version control System)，从狭义上来说，它是软件项目开发

过程中用于储存我们所写的代码所有修订版本的软件，但事实上我们可以将任何对项目有帮助的文档交付版本控制系统进行管理。

昨天看了一天，现推出我的 Git 使用指南vertion1，时间有限，相关细节不够到位，后续会不断更新，以后大家可以将 Git 的使用心得不断添加到后面，当然，最好使用VCS。

目录

1.1 Git 安装

Git的最新版本可以在http://git-scm.com/下载，它是基于命令行操作的，网上也有第三方开发的相应GUI可供下载，因为我比较喜欢命令行操作，所以没有对GUi下载和安装，有兴趣的同学可以自己试试。网上也有相应的文章和资料可供参考!

安装之前首先确保相应的依赖包已经安装，主要有以下几个:

zlib

libcurl

libcrypto（OpenSSL）

rsync（2.6.0 或更高版本）

这些条件满足之后，就可以对Git进行安装了:

1. tar -xzvf git-1.6.1.tar.gz

2. cd git-1.6.1

3. ./configure --prefix=/usr/local

4. make

5. make install

安装成功可以通过git --vertion 查看版本。

1.2 Git工作流程与项目规范

使用进行版本管理，首先要创建自己的项目仓库，也就是用于Git进行管理的“数据库”，一般在项目文件的根目录下建立，比如我们做linux内核开发，则应该在linux源代码树根目录下建立，详细的建立方法将在1.3节叙述。

下一步就是在仓库里面做我们的工作，这里不推荐直接使用主干(master)分支来进行操作，因此我们要创建一个自己的分支，创建分支将在1.9节讲述。现在你可以在你的新分支(new feature)里面工作，文件的添加和修改都可以进行实时监控，可以通过git-status和git-diff命令查看，你的修改并不会自动提交，1.5节主要说明项目仓库的更新和版本的提交。

在有了多个版本之后，就需要对版本进行维护和控制，1.7节主要是讲述查看历史版本的方法，1.8节则对版本的撤销与恢复进行了说明。在多个分支同步进行的情况下，不可避免的要进行合并，这部分在1.9节稍有涉猎，因为实践中还没有接触，在以后会继续补充道文档中。

项目完成后，可以使用git-format-patch创建项目补丁。这个命令的使用比较简单，而且能针对不同的分支不同的版本进行操作，功能强大。

OK，下面就进入分块学习阶段!

1.3 项目仓库的建立

欲使用Git 对现有文档进行版本控制，首先要基于现有文档建立项目仓库。创建一个 Git 项目仓库是很容易的，只要用命令 git-init-db 就可以了。

$ mkdir project

$ cd project

$ git-init-dbgit 将会作出以下的回应

defaulting to local storage area

或者Initialized empty Git repository in project/.git/

这样，一个空的版本库就创建好了，并在当前目录中创建一个叫 .git 的子目录。你可以用 ls -a 查看一下，并请注意其中的三项内容:

一个叫 HEAD 的文件，我们现在来查看一下它的内容:

$ cat .git/HEAD现在 HEAD 的内容应该是这样:

ref: refs/heads/master

我们可以看到，HEAD 文件中的内容其实只是包含了一个索引信息，并且，这个索引将总是指向你的项目中的当前开发分支。

一个叫 objects 的子目录，它包含了你的项目中的所有对象，我们不必直接地了解到这些对象内容，我们应该关心是存放在这些对象中的项目的数据。

另外project目录也不再是普通的文档目录了，今后我们将其称为工作树。因为我们主要是linux内核的开发，所以下面我举的例子主要是对内核

文件的操作，所以project目录等同于源代码的根目录，亦即linux-2.6-vertex。

下面应当有选择地将工作树中的一些文档存储至Git 仓库中。由于Git在向仓库中添加文档时并非是简单地文档复制过去，势必要将所添加文档进行一番处理，生成Git 仓库所能接受的数据格式，Git 称这个过程为"take a snapshot（" 生成快照）。若将工作树下所有文档（包含子目录）生成快照，可采用以下命令:

```
$ cd project
$ git add .
```

所生成的快照被存放到一个临时的存储区域，Git 称该区域为索引。使用git-commit 命令可将索引提交至仓库中，这个过程称为提交，每一次提交都意味着版本在进行一次更新。

```
$ git commit
```

执行上述git-commit 命令时，Git 会自动调用系统默认的文本编辑器，要求你输入版本更新说明并保存。请记住，输入简约而又意义明确的版本更新说明是非常有必要的，可以帮助你快速回忆起对项目的重大改动。

对于简短的版本更新信息，可以使用git-commit 的“-m”选项，如下:

```
$ git commit -m "你的版本更新信息"
```

git-commit 命令在后面会详细讲解。

上述过程即为建立Git 仓库的一般过程，我将其总结为图1.1所示之流程:

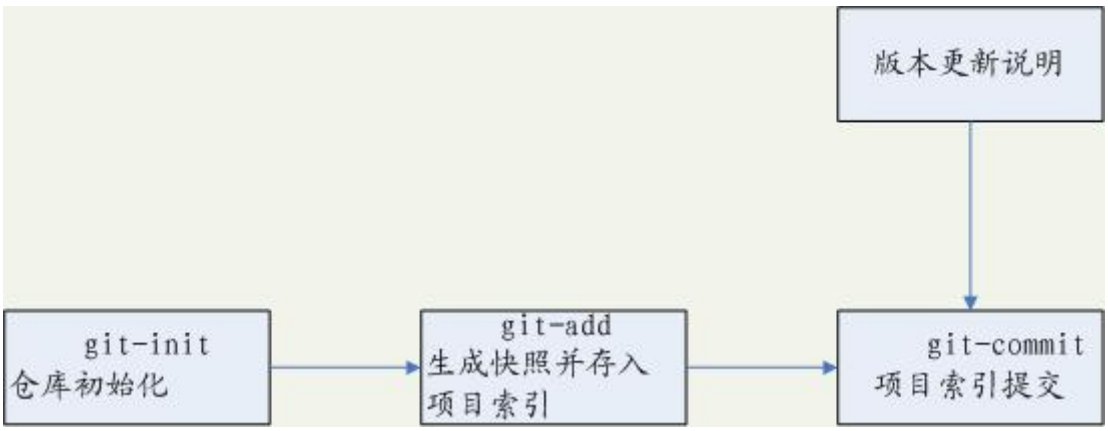

图1.1 Git 仓库建立流程

另外，在git 项目仓库建立中还要注意一下两个问题:

第一个问题是: 在使用Git 之前，你需要面对Git 来一番自我介绍。Git 不喜欢不愿透漏姓名的人，因为它要求每个人在向仓库提交数据时，都应当承担一定的责任。要向Git 进行自我介绍，请使用以下命令:

```
$ git config --global user.name "Your Name Comes Here"
$ git config --global user.email you@yourdomain.example.com
```

第二个问题是: 在生成文档内容快照时，工作树中有一些文档是你不希望接受Git 管理的，譬如程序编译时生成的中间文件，对于这样的文件如何避免为之生成快照?

譬如在工作树中存在以下子目录:

```
doc tmp ipc drivers fs
```

其中的tmp 目录存放着文档编译时生成的中间文件，因此该目录不应该被Git 所管理。为解决此类问题，Git 提供了文档忽略机制，可以将工作树中你不希望接受Git 管理的文档信息写到同一目录下的.gitignore

文件中。对于本例中的tmp 目录，采用如下操作可将其排除仓库之外，然后再对project 生成快照即可。

```
$ cd project
$ echo "tmp" > .gitignore
$ git add .
```

有关gitignore 文件的诸多细节知识可阅读其使用手册:

```
$ man gitignore
```

1.4 项目仓库与工作树

按照前文的说法，Git 仓库就是那个.git 目录，其中存放的是我们所提交的文档索引内容，Git 可基于文档索引内容对其所管理的文档进行内容追踪，从而实现文档的版本控制。工作树是包含.git 的目录，在前文示例中即project 目录。

为了更加明确仓库与工作树的概念，下面做一个实验:

```
$ cp -R project/.git /tmp/test.git
$ cd /tmp
$ git clone test.git test-copy
```

首先，我们将project 目录中的.git 目录复制到/tmp 目录下并进行重命名为test.git ，然后使用git-clone 命令从test.git 中生成test-copy 目录。若进入test-copy 目录观察一下，就会发现该目录所包含的内容是等同于project 目录的。

上述实验意味着，只要我们拥有仓库，即test.git ，那么就可以很容易地生成工作树，而这个工作树又包含着一个仓库，即test-copy/.git 。所以，我们可以这样理解: 在Git 中，仓库与工作树之间无需分的很清楚。

1.5 文件操作

在工作树中，我们日常所进行的工作无非是对Git 仓库所管理的文档进行修改，或者添加／删除一些文件。这些操作与采用Git 管理我们的文档之前没有任何差异，只是在你认为一个工作阶段完成之时，要记得通知Git，命令它记下你所进行更新，这一步骤是通过生成文档快照并将其加入到索引中来实现的。下面举例说明。

譬如我向project 目录添加了一个新文件 fs/binfmt_hwt.c ，我需要通知Git 记住我的这一更新:

```
$ cd project
$ git add fs/binfmt_hwt.c
```

这样，Git 就会将有关fs/binfmt_hwt.c 的更新添加到索引中。然后我又对其它文档进行了一些修改，譬如修改了ipc/msg.c ，继续使用git-add 命令将它们的更新添加到索引中:

```
$ git add ipc/msg.c
```

这里也可以使用以下命令:

```
$ git-update-index
```

晚上，这一天的工作告以段落，我觉得有必要将今天所做的提交到仓库中，于是执行git-commit 操作，将索引内容添加到仓库中。

可能一天下来，你对工作树中的许多文档都进行了更新（文档添加、修改、删除），但是我忘记了它们的名字，此时若将所做的全部更新添加到索引中，比较轻省的做法就是:

```
$ cd project
$ git add .
$ git commit -a
... 输入日志信息...
```

最后这一步-a是通用的方法，我个人比较喜欢使用git-commit –m“版本信息”–a ，这样就不用对版本文件操作了。

git-add 命令通常能够判断出当前目录（包括其子目录）下用户所添加的新文档，并将其信息追加到索引中。git-commit 命令的-a 选项可将所有被修改的文档或者已删除的文档的当前状态提交倒仓库中。记住，如果只是修改或者删除了已被Git 管理的文档，是没必要使用git-add 命令的。

本节并未讲述新的Git 命令，完全是前面所讲过的一些命令的重复介绍，只是它们出现的场景有所区别而已。另外，要注意的问题是，Git 不会主动记录你对文档进行的更新，除非你对它发号施令。

1.6 查看版本历史

在工作树中，使用git-log 命令可以查看当前项目的日志，也就是你在使用git-commit 向仓库提交新版本时所属如的版本更新信息。

$ git log

如果你想看一下每一次版本的大致变动情况，可使用以下命令:

$ git log --stat --summary

下面分析一下git-log 命令的回应信息。如下是我对内核修改后提交的几个版本，版本标记分别为first、second、third ，最下面的那个为原始版本。

```
commit ddea091ee3a5a123d513325bebb93fae592eaec8
Author: litary <litary@XLT.(none)>
Date:   Fri Jul 31 14:27:51 2009 -0800

    third

commit b9fdfbdf8bcab9267cd4bc6defb3c35c6363a143
Author: litary <litary@XLT.(none)>
Date:   Fri Jul 31 08:50:57 2009 -0800

    second

commit 1236a33ab4f668f225512cbdd94f1246fc329d06
Author: litary <litary@XLT.(none)>
Date:   Thu Jul 30 22:21:17 2009 -0800

    first

commit d7ed933b578d9c4dec0e23a5a6f78c464b31c47c
Author: Grant Likely <grant.likely@secretlab.ca>
Date:   Fri Nov 30 16:31:10 2007 -0700
```

每一个版本都对应着一次项目版本更新提交。在项目日志信息中，每条日志的首行（就是那一串莫名奇妙的数字）为版本更新提交所进行的命名，我们可以将该命名理解为项目版本号。项目版本号应该是唯一的，默认由Git 自动生成，用以标示项目的某一次更新。如果我们将项目版本号用作git-show 命令的参数，即可查看该次项目版本的更新细节:

$ git show 版本号（比较长我就不输了）

除了使用完整的版本号查看项目版本更新细节之外，也还可以使用以下方式:

$ git show ddea091 # 一般只使用版本号的前几个字符即可

$ git show HEAD # 显示当前分支的最新版本的更新细节

每一个项目版本号通常都对应存在一个父版本号，也就是项目的前一次版本状态。可使用如下命令查看当前项目版本的父版本更新细节:

$ git show HEAD^ # 查看HEAD 的父版本更新细节

$ git show HEAD^^ # 查看HEAD 的祖父版本更新细节

$ git show HEAD~4 # 查看HEAD 的祖父之祖父的版本更新细节

1.7 撤销与恢复

版本控制系统的一个重要任务就是提供撤销和恢复某一阶段工作的功能。

git-reset 命令就是为这样的任务而准备的，它可以将项目当前版本定位到之前提交的任何版本中。

git-reset 命令有三个选项：--mixed 、--soft 和--hard 。我们在日常使用中仅使用前两个选项；第三个选项由于杀伤力太大，容易损坏项目仓库，需谨慎使用。

--mixed

仅是重置索引的位置，而不改变你的工作树中的任何东西（即，文件中的所有变化都会被保留，也不标记他们为待提交状态），并且提示什么内容还没有被更新了。这个是默认的选项。

--soft

既不触动索引的位置，也不改变工作树中的任何内容，我们只是要求这些内容成为一份好的内容（之后才成为真正的提交内容）。这个选项使你可以将已经提交的东西重新逆转至“已更新但未提交（Updated but not Check in）”的状态。就像已经执行过 git-update-index 命令，但是还没有执行 git-commit 命令一样。

--hard

将工作树中的内容和头索引都切换至指定的版本位置中，也就是说自 <commit-ish> 之后的所有的跟踪内容和工作树中的内容都会全部丢失。因

此，这个选项要慎用，除非你已经非常确定你的确不想再看到那些东西了。关于git-reset 命令的具体如何使用可留作本章的练习题，你可以随便创建一个Git 仓库并向其提交一些版本更新，然后测试--mixed 与--soft 选项的效果。

如果欲查看git-reset 命令对工作树的影响，可使用git-status 命令。这是我们工作中的重点和难点!

1.8 Git命令详解

分支管理: git-branch

直至现在为止，我们的项目版本库一直都是只有一个分支 master。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。下面列举一些常见的分支策略，仅供大家参考:

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。

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。

合并别人的工作的时候，最好是创建一个临时的分支，关于如何用临时分支合并别人的工作的技巧，将会在后面讲述。

创建分支

下面的命令将创建我自己的工作分支，名叫 litary，并且将以后的工作转移到这个分支上开展。

```
$ git-branch litary
$ git-checkout litary
```

删除分支

要删除版本库中的某个分支，使用 git-branch -D 命令就可以了，例如:

```
$ git-branch -D branch-name
```

查看分支

运行下面的命令可以得到你当前工作目录的分支列表:

$ git-branch

输出的分支中前面带*的就是你现在所在的分支，如果你忘记了你现在工作在哪个分支上，可以这样查看，而且运行下面的命令也可以告诉你:

$ cat .git/HEAD

查看项目的发展变化和比较差异

这一节介绍几个查看项目的版本库的发展变化以及比较差异的很有用的命令:

git-show-branch

git-diff

git-whatchanged

git-show-branch 命令可以使我们看到版本库中每个分支的世系发展状态，并且可以看到每次提交的内容是否已进入每个分支。让我们看到版本库的发展记录。

譬如我们要查看世系标号为 master^ 和 litary 的版本的差异情况，我们可以使用这样的命令: $ git-diff master^ litary

合并两个分支: git-merge

既然我们为项目创建了不同的分支，那么我们就要经常地将自己或者是别人在一个分支上的工作合并到其他的分支上去。现在我们看看怎么将 litary 分支上的工作合并到 master 分支中。现在转移我们当前的工作分支到 master，并且将 litary 分支上的工作合并进来。

$ git-checkout master

$ git-merge "Merge work in litary" HEAD litary合并两个分支，还有一个更简便的方式，下面的命令和上面的命令是等价的。

$ git-checkout master

$ git-pull . litary

但是，此时 git 会出现合并冲突提示，就要根据具体的情况和需求对它修改。

1.9 参考文献及推荐学习网站:

git中文教程: http://www.bitsun.com/documents/gittutorcn.htm

linux内核开发中文社区: http://wiki.zh-kernel.org/

git(7) Manual Page: http://wiki.zh-kernel.org/dev/git

linux内核调试分析指南: http://wiki.zh-kernel.org/sniper

Kernel Hackers' Guide to git : http://linux.yyz.us/git-howto.html

还有，永远不要忘记你的身边有个能解决任何问题的人——man!

多多求助man文档!