一步一步学习Git(2)——Git基本操作

前面第一部分写了Git的一些概念，接下来这一篇准备从最基础的git操作入手，看看在实际工作中如何使用git来管理我们的项目。

1、Git基本设置

先看看Git的一些相关设置：

(1) 用户名与Email：由于GIt是分布式版本控制系统，在本地上有一个版本库，我们可以设置自己的用户名与联系方式：

```
代码
ken@Linux:~/project$ git config --global user.name "ken"
ken@Linux:~/project$ git config --global user.email "ken@gmail.com"
ken@Linux:~/project$ git config --global --list
user.name=ken
user.email=ken@gmail.com
```

其中，--global指明user.name, user.email是全局变量。所谓全局变量，就是在你的PC上任何版本库这些变量都是有效的。

user.name, user.email分别表示用户名与邮箱。我们可以通过--list列举出我们已经设置过的内容。需要说明的是，user.name,

user.email是必须设置的，以后才知道是谁修改了项目。实际上git可设置的选项超过130个，只是大部分我们是不常用的。

(2)Git输出颜色：如果你想git反馈的信息中以不同颜色代表不同类型内容，那么可以设置颜色为"always"/'auto':

```
ken@Linux:~/project$ git config --global color.ui "always"
ken@Linux:~/project$ git config --global --list
user.name=ken
user.email=ken@gmail.com
color.ui=always
```

(3)git help：git有着非常丰富的用户手册，只要在命令行上敲入：

```
git help <comand>
```

当然，前提是要安装好git-doc。也可以在线浏览:http://www.kernel.org/pub/software/scm/git/docs/

2、创建版本库

接下来我们可以开始学习如何用git来管理我们的项目了。

第一步，我们必须在本地创建一个版本库，即:.git目录。创建版本库很简单，用Git 提供的git init命令就可以创建了：

```
ken@Linux:~$ mkdir project
ken@Linux:~$ cd project/
ken@Linux:~/project$ git init
Initialized empty Git repository in /home/ken/project/.git/
```

上面的操作我们先创建了project目录，你的项目文件都存放在这个目录下。接着在目录执行：git init。OK。很简单，这时候会在/project/目录下生成一个.git目录，ls -a就可以看到。切换进去该目录，会看到一些文件和目录，主要用来存放版本库的元数据。

3、添加和提交记录

朋友，如果你是一个善良、正直、热爱学习并努力奋斗的人，请加我豆瓣或者QQ，我们一起进步:-)
QQ：397009575
Email: lin.jian1986@gmail.com
豆瓣： http://www.douban.com/people/3721525/

昵称：Linjian
园龄：5年8个月
粉丝：6
关注：0
+加关注

< 2010年7月 >

日	一	二	三	四	五	六
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

搜索

找找看

随笔分类(45)

《代码大全》读书笔记(2)
Assembly(9)
C/C++(2)
CSAPP(2)
English Learning
Git(2)
Linux网络编程(2)
Python(3)
shell(2)
TCPL(4)
That's just life.(2)
经济学
数据结构(6)
职场心得(9)

随笔档案(47)

2011年1月 (4)
2010年11月 (6)
2010年10月 (8)
2010年9月 (15)
2010年8月 (4)
2010年7月 (6)
2010年6月 (4)

假设我们的项目只有一个C文件，这里用经典的hello world作为例子：

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello, world!\n");
    return 0;
}
```

我们先把这个文件添加到版本库的索引当中。这一步并不是提交，但是它是提交的前提。这里要讲到一个git存放代码的问题。在Git中有三个地方可以存放你的代码，第一个就是工作目录树，也就是你之前在编辑上面.c文件的地方；第二个叫做索引，我们刚刚执行add命令后，文件就存储到索引当中。什么是索引？索引也就是暂存区(staging area)，顾名思义，也就是工作目录树与版本库之间的一个缓冲区。执行add就是把文件放到了缓冲区，然后等到我们执行commit(提交)命令后，刚才在暂存区中的文件就被放到版本库上面了。

添加用git add命令：

```
ken@Linux:~/project$ git add helloworld.c
```

好了，已经添加上去了。提交完了之后我们可以进行提交了。提交命令用git commit：

```
ken@Linux:~/project$ git commit -m "add helloworld.c"
[master (root-commit) d1a20e3] add helloworld.c
 1 files changed, 7 insertions(+), 0 deletions(-)
 create mode 100644 helloworld.c
```

其中，-m参数是指定提交的注释，这一点很重要，将来我们要查看之前到底作了什么修改，可以从这些信息获取。所以，好的信息能让以后项目维护起来容易。看看提交后的信息：master是指现在所在的分支是master(主分支)，后面d1a20e3是一个元数据，我们简单理解为是区别该提交文件的唯一编号就行了。再下面的信息，谁都看得懂了：）

好了，添加和提交，操作就这么简单。如果你想查看目前提交的信息，可以执行：git log：

```
commit d1a20e314c527b2bb73538467441eded234778c5
Author: ken <ken@gmail.com>
Date:   Sat Jul 17 01:34:19 2010 +0800

    add helloworld.c
```

因为目前我们只是提交了一次，所以git log显示的信息只有一次提交信息。以后我们再好好学习git log这个执行，可以根据自己需要查看log。还有一点要注意，看到第一行信息了马？commit后面长长的字符串，前7个字符就是我们刚才提交信息中的字符。这怎么回事？git commit只是显示了前7位的SHA－1哈希码作为标识。至于什么叫SHA－1哈希码，我们这里只要简单理解为可以唯一标识每个提交的信息就可以了。

好了，我们已经提交了一个版本。假设我们现在要修改刚才那个文件的内容，把"Hello, world!"存储到一个buffer里面，然后再打印出这个buffer的内容：

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    char str[] = "Hello, world!";
    printf("%s\n", str);
    return 0;
}
```

这个时候，文件已经被修改了，但是版本库里面的那个记录仍然是未修改过的。假如我们要查看文件是否有改动，那么用git status就可以实现这个功能了：

```
代码
ken@Linux:~/project$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#    modified:   helloworld.c
#
no changes added to commit (use "git add" and/or "git commit -a")
```


最新评论

1. Re:一步一步学习Git(2)——Git基本操作

大哥还有没？ 这么好懂，出个中文简单教程～

--Kaiyu Lee

2. Re:一步一步学习Git(1)——Git概述

好文，看的明白，不自量力的去看英文教程，看一会儿就困了。。。

--Kaiyu Lee

阅读排行榜

1. 写了个Linux下简单的FTP客户端程序(3228)
2. 一步一步学习Git(2)——Git基本操作(1596)
3. 关于接口设计的一点看法(1197)
4. 用shell脚本删除相同修改时间的文件(551)
5. 要想着以后总有人来维护你的代码(481)

评论排行榜

1. 一步一步学习Git(2)——Git基本操作(2)
2. 要想着以后总有人来维护你的代码(2)
3. 真正属于你的只有实力(2)
4. 一步一步学习Git(1)——Git概述(1)
5. 你不是一个人在战斗(1)

推荐排行榜

1. 要想着以后总有人来维护你的代码(2)
2. 循环队列数组实现(1)
3. 汇编语言复习摘要四——第一个汇编程序(1)

输入信息表示，文件helloworld.c已被修改，而且"modified: helloworld.c"这句是红色的(之前已设置好颜色)。最后一句是提示你添加并提交文件。这个时候我们直接再次添加跟提交代码了：

代码
```
ken@Linux:~/project$ git add helloworld.c
ken@Linux:~/project$ git commit -m "add a string and print it"
[master c0dcf01] add a string and print it
 1 files changed, 2 insertions(+), 1 deletions(-)
```

值得注意的是，假如你提交的留言太长，可以在每条留言后面加个'\'，另起一行输入留言。

我们已经又提交了一次，为了查看是否提交成功，可以用git log -1。最后面是数字'1'，不是字母'l'。数字1表示我要查看最近的一次提交，如果不加上这个参数，那么以往的提交就会全部打印出来了。而事实上，我们并不需要查看那么多的记录。

代码
```
ken@Linux:~/project$ git add helloworld.c
ken@Linux:~/project$ git commit -m "add a string and print it"
[master c0dcf01] add a string and print it
 1 files changed, 2 insertions(+), 1 deletions(-)
```

4、分支

分支是维护项目中并行历史记录的方法。分支如何理解？我们打两个比方来说明：

第一种情况：有两个人A和B走在同一条大路上(主分支master)要去同一个目的地C，在这条大路上有一条小路可以绕过去到达目标C，但是沿途风景不一样嘛。A还是直接走大路，而B选择了小路，绕了一个弯最后跟A在C相遇了。A跟B这时候交流自从分别后各自沿途看到了什么，于是他们对对方的见闻都有了了解。

第二种情况：A走在一条大路，中途有一条小路，A不知道这条小路能不能成功通向某一个地方，于是A记录了自己现在的位置，走小路一边走一边做记录沿途的风景，最后得到了一个结果：可以通向（或者不可以）。这时候A如果觉得这次实验他只是想知道结果，过程不重要，那么就扔掉刚才的记录（删除分支），如果他觉得有必要留下记录，那就不扔了，暂且保留（不删除分支）。

其实刚才说的两种情况就是最常用到的两种分支：用来支持不同版本的分支（第一个比喻）和特定功能开发的分支（第二个比喻）。接下里我们准备创建一个分支，看看分支有什么特点：

```
ken@Linux:~/project$ git branch sub_1 master
```

这条命令中，第三个参数是新的分支，第四个参数表示父分支，在这里我们的父分支是主分支master。所以这条命令就是在主分支master上创建一条新分支sub_1。我们可以用git branch命令来查看一共有多少个分支已经我们现在所在的分支：

```
ken@Linux:~/project$ git branch
* master
  sub_1
```

前面的星号表示所在分支，所以我们现在有两个分支，所在的分支是master。

这个时候我们想修改刚才的文件。随便条件一个打印语句:printf("Hello, Git!\n");在打印"Hello, world!"的后面，然后提交：

代码
```
ken@Linux:~/project$ git commit -a -m "add a print line after print hello world"
[master 97661f6] add a print line after print hello world
 1 files changed, 1 insertions(+), 0 deletions(-)
ken@Linux:~/project$ git log -1
```

注意到commit 前面有个-a参数，这个参数表示提交全部修改过的文件，连add指令也省了。

那么我们如何切换到新的分支上面去呢？很简单，用checkout命令就ＯＫ了。

```
ken@Linux:~/project$ git checkout sub_1
Switched to branch 'sub_1'
```

切换成功了，如果你现在执行git branch命令，星号应该是在sub_1前面了。这时候你再去看看你的helloworld.c，里面的内容跟刚才修改后的一样吗？

5、发布版本

假如我们要发布项目，定版本号为1.0，那么可以为这个版本打一个标签：

```
ken@Linux:~/project$ git tag 1.0 sub_1
ken@Linux:~/project$ git tag
1.0
```

上面第一条指令中，第三个参数是标签，最后一个则是表示打标签的点；接下来呢，我们要做的最后一件事是为发布的版本做打包成一个tar或者zip包。命令git archive可以实现这个功能：

```
ken@Linux:~/project$ git archive --format=tar \
> --prefix=helloworld-1.0/ 1.0 \
> | gzip > helloworld-1.0.tar.gz
ken@Linux:~/project$ ls
helloworld-1.0.tar.gz  helloworld.c
```

上面的命令相对来说复杂了点。如果你的命令太长，可以用'\'来分行；format参数指定哪种格式输出，也可以是zip；prefix表示发布的包前缀；1.0是指定要对哪个标签打包；最后面就是压缩了，用的是Unix的管道，假如不熟悉的话，google一下吧:)

GIt的一些基本操作就介绍到这里，以后慢慢总结再慢慢添加了。Happy Git :-)

END

分类: Git

« 上一篇：Python小练习——创建简单地址簿
» 下一篇：关于链接的简单总结

posted @ 2010-07-20 01:19 Linjian 阅读(1595) 评论(2) 编辑 收藏

发表评论

#1楼 2012-06-28 11:25 | 东天青帝

写得非常不错，可以作入门用

支持(0) 反对(0)

#2楼 2013-08-13 10:15 | Kaiyu Lee

大哥还有没？ 这么好懂，出个中文简单教程～

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。