Git的基本概念与入门 - CSDN博客

入门

如果你想试着用一下Git的话，那么我们马上就可以开始了。本章将会带领你创建自己的第一个项目。我们会为你演示那些用于提交修改版本、查看历史和与其他开发者交换版本的命令。

1　准备Git环境

首先，我们需要安装好Git。你可以在Git的官网上找到你所需要的一切：

```
http://git-scm.com/download
```

Git是一个高可配置软件。首先，我们可以宣布用**config**命令配置一下用户名和用户邮箱：[1]

```
> git config --global user.email "hans@mustermann.de"
```

2　第一个Git项目

在这里，我们建议你最好能为接下来的Git测试单独开辟一个项目。总之应先从一个简单的小项目开始。在我们这个小小的示例项目中，**first-steps**目录下只有两个文本文件，如图1所示。

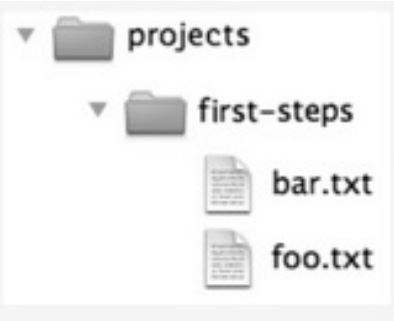

图1　我们的示例项目

在开始摆弄这个玩具项目之前，我们建议你最好先做一个备份！尽管在Git中，想要造成永久性的删除或破坏也不是件容易的事情，而且每当你要做某些“危险”动作的时候，Git通常也会发出相应的警告消息。但是，有备无患总是好的。

1　创建版本库

现在，我们首先需要创建一个版本库，用于存储该项目本身及其历史。为此，我们需要在该项目目录中使用**init**命令。对于一个带版本库的项目目录，我们通常称之为工作区。

```
> cd /projects/first-steps
> git init
Initialized empty Git repository in /projects/first-steps/.git/
```

init命令会在上述目录中创建一个名为.git的隐藏目录，并在其中创建一个版本库。但请注意，该目录在Windows资源管理器或Mac Finder中可能是不可见的。

图2　本地版本库所在的目录

2　首次提交

接下来，我们需要将**foo.txt**和**bar.txt**这两个文件添加到版本库中去。在Git中，我们通常将项目的一个版本称之为一次提交，但这要分两个步骤来实现。第一步，我们要先用**add**命令来确定哪些文件应被包含在下次提交中。第二步，再

用**commit**命令将修改传送到版本库中，并赋予该提交一个散列值以便标识这次新提交。在这里，我们的散列值为2f43cd0，但可能会有所不同，因为该值取决于文件内容。

```
> git add foo.txt bar.txt
> git commit --message "Sample project imported."
master (root-commit) 2f43cd0] Sample project imported.
2 files changed, 2 insertions(+), 0 deletions(-)
create mode 100644 bar.txt
create mode 100644 foo.txt
```

3　检查状态

现在，我们来修改一下**foo.txt**文件的内容，先删除**bar.txt**文件，再添加一个名为**bar.html**的新文件。然后，**status**命令就会显示出该项目自上次提交以来所发生的所有修改。请注意，新文件**bar.html**在这里被标示成了未跟踪状态，这是因为我们还没有用**add**命令将其注册到版本库。

```
> git status
# On branch master
# Changed but not updated:
# (use "git add/rm ..." to update what will be committed)
# (use "git checkout -- ..." to discard changes in
#                               working directory)
#
#       deleted:   bar.txt
#       modified:  foo.txt
#
# Untracked files:
# (use "git add ..." to include in what will be committed)
#
#   bar.html
no changes added to commit (use "git add" and/or "git commit -a")
```

如果我们还想看到更多细节性的内容，也可以通过**diff**命令来显示其每个被修改的行。当然。有很多人可能会觉得**diff**的输出是个非常难读的东西。幸运的是，在这一领域，我们有许多工具和开发环境可用，它们可以将这一切显示得更为清晰（见图3）。

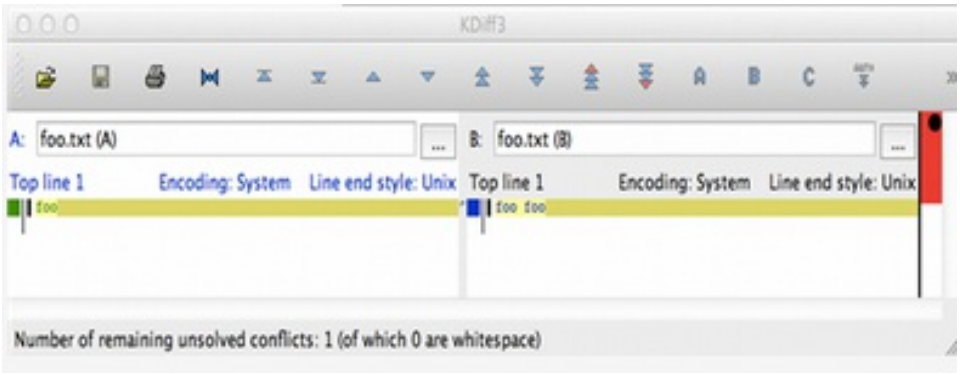

图3　图形工具（kdiff3）中的Diff报告

```
> git diff foo.txt
diff --git a/foo.txt b/foo.txt
index 191028.090387f 100644
--- a/foo.txt
+++ b/foo.txt
@@ -1 +1 @@
-foo
\ No newline at end of file
+foo foo
\ No newline at end of file
```

4　提交修改

接下来，所有的修改都必须要先被归档成一次新的提交。我们要对修改过的文件和新文件执行**add**命令，并对要删除的文件使用**rm**命令。

```
> git add foo.txt bar.html
> git rm bar.txt rm 'bar.txt'
```

现在再次调用**status**命令，我们会看到所有的修改已经被纳入了下一次提交中。

```
> git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD ..." to unstage)
#
#       new file:   bar.html
#       deleted:    bar.txt
#       modified:   foo.txt
#
```

然后用**commit**命令提交这些修改。

```
> git commit --message "Some changes."
[master 7ac0f38] Some changes.

3 files changed, 2 insertions(+), 2 deletions(-)
 create mode 100644 bar.html  delete mode 100644 bar.txt
```

5 显示历史

log命令可用来显示项目的历史，所有提交都会按时间顺序被降序排列出来。

```
> git log
commit 7ac0f38f575a60940ec93c98de11966d784e9e4f
Author: Rene Preissel
Date: Thu Dec 2 09:52:25 2010 +0100

     Some changes.

commit 2f43cd047baadc1b52a8367b7cad2cb63bca05b7
Author: Rene Preissel
Date: Thu Dec 2 09:44:24 2010 +0100
Sample project imported.
```

3 Git的协作功能

现在，我们已经有了一个存放项目文件的工作区，以及一个存放项目历史的版本库。在一个像CVS和Subversion这样传统的集中式版本系统中，尽管每个开发者也都有属于他/她自己的工作区，但所有人都共享了一个通用的版本库。而在Git中，每个开发者拥有的是一个属于他/她自己的、自带独立版本库的工作区，因此这已经是一个不依赖于中央服务器的、完整的版本控制系统了。开发者们可以通过交换各自版本库中的提交来实现项目合作。下面我们就来做个试验，先创建一个新的工作区，以便我们模拟第二位开发者的活动。

3.1 克隆版本库

我们的这位新开发者首先要有一个属于他/她自己的版本库副本（也称为克隆体）。该副本中包含了所有的原始信息与整个项目的历史信息。下面。我们用clone命令来创建一个克隆体。

```
> git clone /projects/first-steps /projects/first-steps-clone
Cloning into first-steps-clone...
done.
```

现在，该项目结构如图4所示。

图4　样例项目与它的克隆体

3.2　从另一版本库中获取修改

下面，我们来修改一下**first-steps/foo.txt**文件，并执行以下操作来创建一次新提交。

```
> cd /projects/first-steps
> git add foo.txt
> git commit --message "A change in the original."
```

现在，新的提交已经被存入了我们原来的**first-steps**版本库中，但其克隆版本库（first-stepsclone）中依然缺失这次提交。为了让你更好地理解这一情况，我们来看一下**first-steps**的日志。

```
> git log --oneline
a662055 A change in the original.
7ac0f38 Some changes.
2f43cd0 Sample project imported.
```

在接下来的步骤中，我们再来修改克隆版本库中的**first-steps-clone/bar.html**文件，并执行以下操作。

```
> cd /projects/first-steps-clone
> git add bar.html
> git commit --message "A change in the clone."
> git log --oneline
1fcc06a A change in the clone.
7ac0f38 Some changes.
2f43cd0 Sample project imported.
```

现在，我们在两个版本库中各做了一次新的提交。接下来，我们要用**pull**命令将原版本库中的新提交传递给它的克隆体。由于之前我们在创建克隆版本库时，原版本库的路径就已经被存储在了它的克隆体中，因此**pul**l命令知道该从哪里去取回新的提交。

```
> cd /projects/first-steps-clone
> git pull

remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /projects/first-steps
   7ac0f38..a662055 master -> origin/master
Merge made by recursive.
foo.txt |      2 +-
1 files changed, 1 insertions(+), 1 deletions(-)
```

如上所示，**pull**命令从原版本库中取回了新的修改，将它们与克隆体中的本地修改进行了对比，并在工作区中合并了两边的修改，创建了一次新的提交。这个过程就是所谓的合并（merge）。

请注意！合并过程在某些情况下可能会带来冲突。一旦遇到了这种情况，Git中就不能进行自动化的版本合并了。在这种情况下，我们就必须要手动清理一些文件，然后再确认要提交哪些修改。

在拉回（pull）、合并(merge)的过程完成之后，我们可以用一个新的log命令来查看结果。这次是日志的图形化版本。

```
> git log --graph
9e7d7b9 Merge branch 'master' of /projects/first-steps
*
|\
| * a662055 A change in the original.
* | 1fcc06a A change in the clone.
|/ * 7ac0f38 Some changes.
* 2f43cd0 Sample project imported.
```

这一次，历史记录不再是一条直线了。在上面的日志中，我们可以很清晰地看到并行开发的过程（即中间的两次提交），以及之后用于合并分支的那次合并提交（即顶部的那次提交）。

3.3 从任意版本库中取回修改

在没有参数的情况下，**pull**命令只在克隆版本库中能发挥作用，因为只有该克隆体中有默认的原版本库的连接。当我们执行**pull**操作时，也可以用参数来指定任意版本库的路径，以便从某一特定开发分支中提取相关修改。

现在，让我们将克隆体中的修改**pull**到原版本库中吧。

```
> cd /projects/first-steps
> git pull /projects/first-steps-clone master
remote: Counting objects: 8, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
From /projects/first-steps-clone
* branch            master → FETCH_HEAD
Updating a662055..9e7d7b9 Fast-forward
bar.html |    2 +-
1 files changed, 1 insertions(+), 1 deletions(-)
```

3.4 创建共享版本库

除了可以用**pull**命令从其他版本库中取回相关提交外，我们也可以用**push**命令将提交传送给其他版本库。只不过，**push**命令只适用于那些没有开发者在上面开展具体工作的版本库。最好的方法就是创建一个不带工作区的版本库，我们称之为裸版本库（bare repository）。你可以使用**clone**命令的**--bare**选项来创建一个裸版本库。裸版本库通常可被用来充当开发者们传递提交（使用**push**命令）的汇聚点，以便其他人可以从中拉回他们所做的修改。下面我们来看一个裸版本库（见图5）。

图5　裸版本库（一个没有工作区的版本库）

```
> git clone --bare /projects/first-steps
/projects/first-steps-bare.git
Cloning into bare repository first-steps-bare.git...
done.
```

3.5　用push命令上载修改

为了演示**push**命令的使用，我们需要再次修改一下**firststeps/foo.txt**文件，并执行以下操作来创建一次新的提交。

```
> cd /projects/first-steps
> git add foo.txt
> git commit --message "More changes in the original."
```

接下来，我们就可以用**push**命令向共享版本库传送该提交了（见图6）。该指令的参数要求与**pull**命令相同，我们需要指定目标版本库的路径及其分支。

```
> git push /projects/first-steps-bare.git master
Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 293 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
To /projects/first-steps-bare.git/
9e7d7b9..7e7e589 master -> master
```

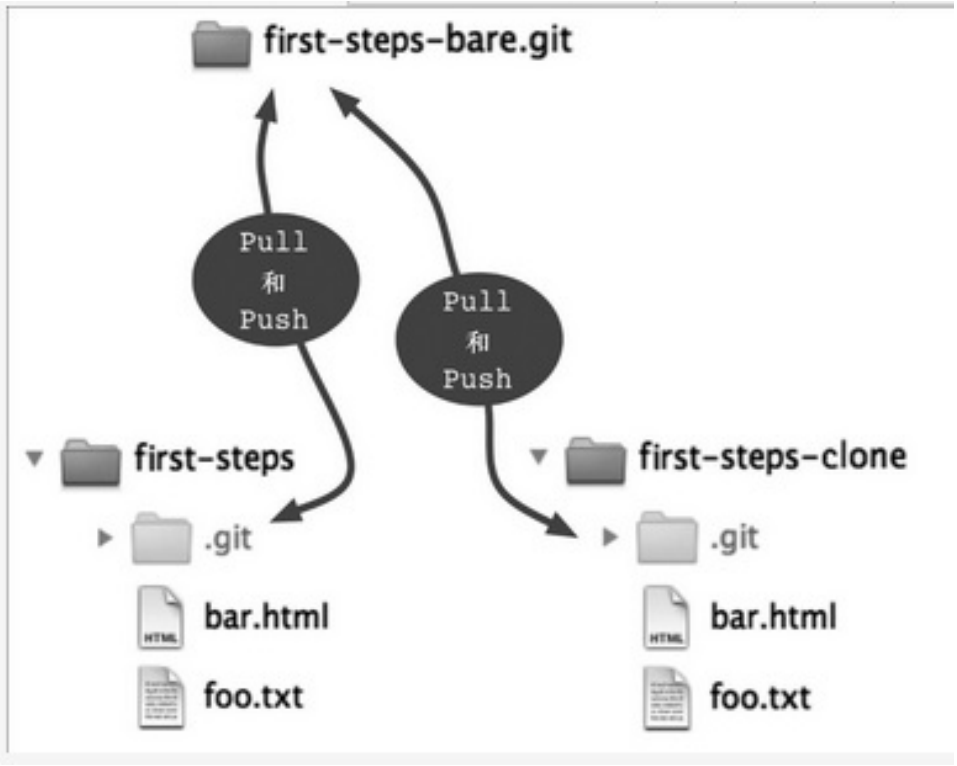

图6　经由共享版本库来进行版本共享

3.6　Pull命令：取回修改

现在，为了让克隆版本库也得到相应的修改，我们需要在执行**pull**命令时配置参数指向共享版本库的路径参数。

```
> cd /projects/first-steps-clone

> git pull /projects/first-steps-bare.git master
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From ../first-steps-bare
* branch      master      -> FETCH_HEAD
Updating 9e7d7b9..7e7e589
Fast-forward
foo.txt |    2 +-
1 files changed, 1 insertions(+), 1 deletions(-)
```

请注意！如果另一个开发者在我们之前已经做过一次**push**操作，此次**push**命令就会被拒绝传送提交。这时候，我们必须要先做一次**pull**操作，将其他人新上载的更新取回，并在本地合并。

4　本章小结

- **工作区与版本库：**工作区是一个包含.git子目录（内含版本库）中的目录。我们可以用**init**命令在当前目录中创建版本库。
- **版本提交：**一次版本提交通常定义了版本库中所有文件的一个版本，它详细说明了该版本是由何人在何时何地创建的。当然，我们需要用**add**命令来确定哪些文件将被纳入下一次提交，然后再用**commit**命令创建新的版本提交。
- **查看信息：**通过**status**命令，我们可以查看哪些文件已被本地修改，以及哪些修改将被纳入下次提交。另外，**log**命令可用来显示提交历史。**diff**命令可用来显示两个版本文件之间的差异。
- **克隆：**对于用**clone**命令创建某一个版本库的副本，我们称之为该版本库的克隆体。在一般情况下，每个开发者都会拥有整个项目版本库的完整克隆体，他/她的工作区中将会包含完整的项目历史。这使他们可以各自独立开展工作，无需连接服务器。
- **推送与拉回：**push与**pull**命令可用于在本地和远程版本库之间共享版本提交。

[1]　译者注：示例中似乎少了用户名的部分：git config --global user.name“Hans”

本文来自《Git学习指南》