Git 学习笔记

高睿昊

2017 年 2 月 10 日

教程蓝本：《廖雪峰的 git 教程》
网址：http://www.liaoxuefeng.com/wiki/001373951630592960 6dd18361248578c67b8067c8c017b000

1 安装 git

一提到 git，大佬们常说：

“Git 是目前世界上最先进的分布式版本控制系统”

所以我们必须理解 Git 实际上是一个软件，同样需要安装。

在 Ubuntu 上安装 Git 的命令：$sudo apt−get install git 。(Windows 和 Mac 上面的安装很简单，问度娘就好了）安装完成以后还需要配置账户信息

```
$ git config −−global user.name ”YourName”
$ git config −−global user.email ”email@example.com”
```

2 创建版本库

在需要 Git 管理的目录下执行命令：

```
$ git init
```

对于需要 Git 管理的目录需要使用 add 命令进行处理

```
$ git add 文件名或者目录名
```

最后使用 commit 命令即可提交到版本库

```
$ git commit −m “用于识别的附加在提交后面的消息”
```

3 Git 时光机

git status 命令可以让我们时刻掌握仓库当前的状态，上面的命令告诉我们，readme.txt 被修改过了，但还没有准备提交的修改。

git diff 顾名思义就是查看 difference，显示的格式正是 Unix 通用的 diff 格式，可以从上面的命令输出看到，我们在第一行添加了一个 “distributed” 单词。

当我们对文件修改后再提交时，也要经过添加新的文件的 add 和 commit 命令。

3.1 版本回退

在 Git 中，可以用 git log 命令查看提交的版本信息，例如在编写个笔记过程中使用的 log 是这样的：

```
commit a0a5fe4e08f77bc5bf609b27575206cb7cb1ccb2
Author: Acinonyx Tungsten <gaoruihao@outlook.com>
Date:   Thu Feb 9 19:54:30 2017 +0800

    1

commit 047711edf118019cd76d9628d3d61b6022f6b008
Author: Acinonyx Tungsten <gaoruihao@outlook.com>
Date:   Thu Feb 9 08:39:34 2017 +0800

    add a line

commit 41320d5047bf318ab067352bfaa5ce658646d188
Author: Acinonyx Tungsten <gaoruihao@outlook.com>
Date:   Thu Feb 9 08:33:59 2017 +0800

    first  time
```

这样看起来比较烦的话可以用 `$ git log --pretty=oneline`，得到的结果是这样的：

```
wolf-tungsten@wolftungsten-ThinkPad-T460p:~/NotePapers$ git log --pretty=oneline
a0a5fe4e08f77bc5bf609b27575206cb7cb1ccb2 1
047711edf118019cd76d9628d3d61b6022f6b008 add a line
41320d5047bf318ab067352bfaa5ce658646d188 first time
```

前面一串串的数字就是十六进制表示的版本号了（官方名称是 commit id）。当我们想回退到上一个版本的时候，就可以用命令`$ git reset --hard HEAD^`。其中 HEAD 代表当前版本，就^ 代表上一个版本——以此类推`$ git reset --hard HEAD^^` 自然就是回退到两个版本啦。

现在我们相当于从 21 世纪回到了 19 世纪，那我们要如何再从 19 实际回到 21 世纪呢？这时候刚才看起来烦人的 commit id 就有大用了。首先我们使用`$ git reflog` 命令来查看各个提交版本的 commit id：

```
wolf-tungsten@wolftungsten-ThinkPad-T460p:~/NotePapers$ git reflog
a0a5fe4 HEAD@{0}: commit: 1
```

```
047711e HEAD@{1}: commit: add a line
41320d5 HEAD@{2}: commit (initial): first time
```

然后`$ git reset --hard 047711e`，就会回到这个版本。这样的话，时光机就可以任意穿梭了。

3.2 撤销修改

首先我们必须了解 git 的**工作区**和**暂存区**：我们实际操作的文件处于工作区，add 命令的执行就是将工作区的内容添加到了暂存区，commit 命令将暂存区的内容正式提交至版本仓库。

git 的存在相当于给文件修改添加了撤销功能，`$ git checkout -- 文件名或者目录名` 可以让文件恢复到最近一次 commit 或者 add 的状态。如果改动已经提交到了暂存区，就要使用`$ git reset HEAD 文件名或者目录名`把文件撤回到工作区进一步修改之后再进行 add 和 commit 的操作。

撤销更改的几种情况

场景 1 当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令 git checkout – file。

场景 2 当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令 git reset HEAD file，就回到了场景 1，第二步按场景 1 操作。

场景 3 已经提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退一节，不过前提是没有推送到远程库。

3.3 删除文件

删除文件是一种常见操作。简单的来说我们可以直接在工作区删除一个文件，然后作为一个新版本提交到版本库，这样我们可以随时利用前面的版本回退，或者撤销更改的方法去“一键还原”。

或者，我们可以采用 `$ git rm 文件或者目录` 命令直接从工作区、版本库删除这个文件。

如果发现删错了，那么还可以用`$ git checkout --文件或目录` 将其恢复为版本库中最新的版本。

4 分支管理

4.1 创建与合并分支

这里以创建 dev 分支并合并到 master 分支并最后将其删除为例进行讲解：

- 创建 dev 分支并切换到 dev 分支：`$ git checkout -b dev`。checkout 命令加上-b 参数表示创建和切换。现在可以随心所欲的对 dev 分支进行修改，而不影响 master 分支的情况。
- 通过`$ git branch` 命令可以查看现有的分支。
- 知道了有哪些分支以后就可以再次使用`$ git checkout 分支名`来切换到需要的分支。
- 合并分支[1] 首先我们切换到 master 分支`$ git checkout master`，然后使用`$ git merge dev` 命令最终把 dev 合并到当前分支上。
- 合并完成后就可以把 dev 分支删除`$ git branch -d dev`。

[1]需要注意一般情况下是把超前的分支（本例中的 dev 分支）合并到滞后的分支上（本例中的 master 分支）。

由于创建 dev 分支后，没有再对 master 分支进行修改，所以合并分支的时候 git 自动为我们使用了“fast-forward”。后面会说明不能 Fast-forward 的情况以及解决办法。

4.2 冲突的产生与解决

在上一节中，创建了 dev 分支，之后所有的更改都是在 dev 分支上进行的。下面就要来处理：“创建 dev 分支后，又对 master 分支修改” 的情况。如果直接进行合并，就会产生冲突。

```
wolf−tungsten@wolftungsten−ThinkPad−T460p:~/test$ git merge dev
自动合并 test
冲突（内容）：合并冲突于 test
```

这时就要使用`$ git status` 命令来查看究竟是哪个文件出了问题

```
wolf−tungsten@wolftungsten−ThinkPad−T460p:~/test$ git status
位于分支 master
您有尚未合并的路径。
  （解决冲突并运行 ”git commit”）

未合并的路径：
  （使用 ”git add <文件>...” 标记解决方案）

        双方修改：    test

修改尚未加入提交（使用 ”git add” 和/或 ”git commit −a”）
```

打开冲突的 test 文件你会发现它变成了这样：

```
first
<<<<<<< HEAD
add on master
=======
add in dev
>>>>>>> dev
```

这其中用`<<<<<<<\=======\>>>>>>>` 标记出来的就是冲突的地方，按照需求修改再保存就可以正确合并了。

再次提交后，用带参数的 git log 可以查看合并的情况：

```
wolf−tungsten@wolftungsten−ThinkPad−T460p:~/test$ git log −−graph −−pretty=oneline
−−abbrev−commit
*   0ee2246 2on master
|\
| * 0cb0c0b 2 on dev
* | 99d0183 commit on master
|/
* c59e421 1
```

4.3 禁用 Fast Forward

合并的时候如果可以使用 fastforward 固然很爽，但是带来的代价是，一旦我们删除分支，那么分支记录就完全丢失了。这里可以在合并的时候加入`--no-ff` 参数来禁用 fastforward。

完整的命令：`git merge --no-ff -m "merge with no-ff" dev`。因为合并后会产生一个新的提交，所以还是需要-m 补充提交的信息。实际开发过程应该尊崇如下的管理规则：

> 在实际开发中，我们应该按照几个基本原则进行分支管理：
>
> 1. master 分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；
> 2. 干活都在 dev 分支上，也就是说，dev 分支是不稳定的，到某个时候，比如 1.0 版本发布时，再把 dev 分支合并到 master 上，在 master 分支发布 1.0 版本；
> 3. 每个人都在 dev 分支上干活，每个人都有自己的分支，时不时地往 dev 分支上合并就可以了

4.4 使用 stash 保护工作现场

适用于：工作进行到一半还不适合提交，但是有需要临时去处理别的版本或者分支上的问题。首先使用`$ git stash` 命令，这时候工作区已经保存成了快照，可以放心的切换版本切换分支干其他的事情。然后，等到其他问题处理完成后，用`$ git stash list` 罗列以保存的工作区。

```
$ git stash list
stash@{0}: WIP on dev: 6224937 add merge
```

接下来有两种选择：

- 切换回工作环境：使用`git stash apply`，这样切换回去以后并不会删除保存的快照，还需要`git stash drop` 来删除保存的快照。
- 使用`git stash pop`，恢复的同时也把保存的快照删掉了。

还有更美好的事情，想我这种不爱收拾桌子的人，可以保存多个快照。需要恢复现场的时候用`$ git stash list` 罗列，再用`$ git stash apply stash@{0}` 恢复到指定状态就可以啦。

4.5 强行删除一个分支

前面说过，用`git branch -d` 分支可以删除一个分支。但是这样操作的前提条件是这个分支已经合并到主分支上去了。但是有的时候可能需要 “甩锅”————还没开发完成就放弃。这样的分支是不能合并到 dev 上去，更不能合并到 master 上去。要删除一个未合并的分支要用`git branch -D` 分支（大写 D）。

5 标签的使用

标签和 commit 是相互对应的，它的用途简单来说就 commit id 不便于记忆和识别，给它起个别名。

5.1 打标签

使用命令`$ git tag <标签名称>` 就直接给当前分支最新的版本打上了标签。
使用命令`$ git tag <标签名称> <commit_id>` 可以给指定的 commit 打标签。
使用命令`$ git tag -a <标签名称> -m <文字说明> <commit id>` 来打一个带文字说明的标签。

需要查看有哪些标签的时候可以git tag，就会列出所有打过的标签[2]

5.2 撕标签

标签打错了就比较尴尬了，不过我们还是有办法缓解尴尬。

$ git tag -d <标签名称> 删除掉指定的标签。

标签默认是储存在本地的,不会推送到远程版本库。如果很任性的话也可以$ git push origin <标签名称>推送到远程库。或者一次性推送所有标签$ git push origin --tags。一旦推送到远程，想要删除就比较麻烦：首先删除本地标签；再用$ git push origin :refs/tags/<标签名称> 删除远程标签。

[2]使用这个命令列出的标签是按照字母顺序排序的，不是按照时间顺序的。