无标题

git学习笔记

---

1. 创建版本库

通过`git init`命令把某个目录变成Git可以管理的仓库

```
$ git init
Initialized empty Git repository in /Users/michael/learngit/.git/
```

1.1 把文件添加到版本库

1.1.1用命令`git add`告诉Git，把文件添加到仓库：

```
$ git add readme.txt
```

1.1.2用命令`git commit`告诉Git，把文件提交到仓库：

```
$ git commit -m "wrote a readme file"
[master (root-commit) eaadf4e] wrote a readme file
 1 file changed, 2 insertions(+)
 create mode 100644 readme.txt

tip:
```

简单解释一下`git commit`命令，`-m`后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录

加上 `-a` ，git会自动把所有已跟踪过的文件暂存起来一并提交
例如: `git commit -a -m "描述"`

2. 时光穿梭机

2.1. 版本回退

2.1.1 查看仓库状态

`git status`命令可以让我们时刻掌握仓库当前的状态

2.1.2 查看difference

`git diff`顾名思义就是查看difference

```
    (
       `git diff #` 就是查看工作区和暂存区的不同
       `git diff--cached` 就是查看暂存区和master的不同
     )
```

2.1.3 查看历史记录

`git log` 告诉我们历史记录
(`git log --pretty=oneline` 显示简单的信息)

```
$ git log --pretty=oneline
1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master) append GPL
e475afc93c209a690c39c13a46716e8fa000c366 add distributed
eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0 wrote a readme file
```

2.1.4 版本回退

`$ git reset`(这个命令也可以把暂存区的修改回退到工作区)
在Git中，用`HEAD`表示当前版本，也就是最新的提交的，上一个版本就是`HEAD^`，上上一个版本就是`HEAD^^`，当然往上100个版本写100个`^`比较容易数不过来，所以写成`HEAD~100`

```
示例:
$ git reset --hard HEAD^
HEAD is now at e475afc add distributed
```

`回退错了,宝宝后悔了咋办??`
要重返未来，用`git reflog`查看命令历史，以便确定要回到未来的哪个版本

2.2 工作区和暂存区

2.3 撤销修改

命令`git checkout -- readme.txt`意思就是，把`readme.txt`文件在工作区的修改全部撤销，这里有`两种情况`：

```
一种是`readme.txt`自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是`readme.txt`已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

总之，就是让这个文件回到最近一次`git commit`或`git add`时的状态。

`git checkout -- file`命令中的--很重要，没有--，就变成了“切换到另一个分支”的命令。
```

2.4 删除版本库文件

假设工作区文件删除了test.txt：

```
1:你想把版本库对应的文件也删调：
用命令`git rm test.txt`删掉，并且`git commit -m "描述"`

2:你删错了，所以你想回退：
   git checkout -- test.txt
   含义:用暂存区的版本替换工作区的版本
```

书籍《pro git》中，补充:

```
要从 Git 中移除某个文件，就必须要`从已跟踪文件清单中移除（确切地说，是从暂存区域移除），然后提交`。
可
以用 `git rm` 命令完成此项工作，并连带从工作目录中删除指定的文件，这样以后就不会出现在未跟踪文件清单
中了。
```

如果希望删除时，在工作区保留，则加上 `--cached`

```
`git rm --cached 文件或目录`
```

3 远程仓库(例如:github)

1.生成ssh密钥对
2. 本地仓库关联远程仓库

```
$ git remote add origin git@github.com:MrZhaoHuan/hadoop.git
```

3.把本地库内容推送到远程仓库

```
$ git push -u origin master
```

小结:
要关联一个远程库，使用命令`git remote add origin git@server-name:path/repo-name.git`；

关联后，使用命令`git push -u origin master`第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令`git push origin master`推送最新修改；

3.1 远程仓库克隆(例如:github)

```
$ git clone git@github.com:MrZhaoHuan/gitskills.git

要克隆一个仓库，首先必须知道仓库的地址，然后使用`git clone`命令克隆。

Git支持多种协议，包括`https`，但通过`ssh`支持的原生`git`协议速度最快。
```

3.2 查看远程仓库的信息

`git remote show [remote-name]`

3.2 修改远程仓库

git remote rename pb paul

4 分支管理(例如:github)

Git鼓励大量使用分支：
查看分支：`git branch`

创建分支：`git branch <name>`

切换分支：`git checkout <name>`

```
git checkout testing
```

创建+切换分支：`git checkout -b <name>`

合并某分支到当前分支：`git merge <name>`

删除分支：`git branch -d <name>`
如果要丢弃一个没有被合并过的分支，可以通过`git branch -D <name>`强行删除。

4.1 分支管理策略

合并分支时，加上`--no-ff`参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，
而`fast forward`合并就看不出来曾经做过合并

4.2 bug分支

修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；
当手头工作没有完成时，先把工作现场`git stash`一下，然后去修复bug，修复后，再`git stash pop`，回到工作现场。

4.3 多人协作

因此，多人协作的工作模式通常是这样：

首先，可以试图用`git push origin <branch-name>`推送自己的修改；
如果推送失败，则因为远程分支比你的本地更新，需要先用`git pull`试图合并；
如果合并有冲突，则解决冲突，并在本地提交；
没有冲突或者解决掉冲突后，再用`git push origin <branch-name>`推送就能成功！
如果`git pull`提示`no tracking information`，则说明本地分支和远程分支的链接关系没有创建，用命令`git branch --set-upstream-to <branch-name> origin/<branch-name>`。
这就是多人协作的工作模式，一旦熟悉了，就非常简单。

`小结:`

查看远程库信息，使用`git remote -v`；
本地新建的分支如果不推送到远程，对其他人就是不可见的；
从本地推送分支，使用`git push origin branch-name`，如果推送失败，先用`git pull`抓取远程的新提交；
在本地创建和远程分支对应的分支，使用`git checkout -b branch-name origin/branch-name`，本地和远程分支的名称最好一致；
建立本地分支和远程分支的关联，使用`git branch --set-upstream branch-name origin/branch-name`；
从远程抓取分支，使用`git pull`，如果有冲突，要先处理冲突。

5 标签管理

- 命令`git tag <tagname>`用于新建一个标签，默认为`HEAD`，也可以指定一个commit id；
- 命令`git tag -a <tagname> -m "blablabla..."`可以指定标签信息；
- 命令`git tag`可以查看所有标签。

`注意：标签总是和某个commit挂钩。如果这个commit既出现在master分支，又出现在dev分支，那么在这两个分支上都可以看到这个标签。`

5.1标签操作

- 命令`git push origin <tagname>`可以推送一个本地标签；
- 命令`git push origin --tags`可以推送全部未推送过的本地标签；
- 命令`git tag -d <tagname>`可以删除一个本地标签；
- 命令`git push origin :refs/tags/<tagname>`可以删除一个远程标签。

6 自定义git

6.1 忽略特殊文件

- 忽略某些文件时，需要编写 `.gitignore` ；
- `.gitignore` 文件本身要放到版本库里，并且可以对 `.gitignore` 做版本管理！

可以用命令 `git check-ignore` 来检查被忽略的文件在文件 `.gitignore` 中第几行定义的

```
$ `git check-ignore -v` App.class
.gitignore:3:*.class    App.class
```

6.2 配置别名

例如:
如果想敲 `git st` 就表示 `git status` ,则如下配置

```
$ git config --global alias.st status
```

加上 `--global` 是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用。

甚至还有人丧心病狂地把 `lg` 配置成了：
git config -global alias.lg
“`log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abb`