Git笔记

> 以下笔记根据Git教程 - 廖雪峰的官方网站 (liaoxuefeng.com) 撰写

建立版本库

创建一个目录：

```
$ mkdir <目录名> //创建目录
$ cd <目录名> //打开目录（每次重新进入git时都要重新打开）
$ pwd //查看该目录路径
```

在该目录下创建一个空的版本库：

```
$ git init
```

新建一个主分支（可选操作）

```
$ git branch -M main
```

在版本库中添加文件

将需要放入的文件添加到创建好的目录后

用`git add`命令告诉Git，把文件添加到仓库：

```
$ git add <文件1> //将文件add进暂存区
$ git add <文件2> <文件3>...//可同时add多个文件
```

用`git commit`命令告诉Git，把文件提交到仓库：

```
$ git commit -m "本次提交的说明"
```

注意！一定要养成说明的习惯！！

用`cat`指令可以查看工作区文件内容：

```
$ cat <文件名>
```

时光机穿梭

可运行`git status`命令查看提交结果：

```
$ git status
```

若有修改，可用`git diff`命令查看修改了什么内容：

```
git diff <文件名>
```

版本回退

运用`git log`命令查看修改历史记录：

```
$ git log //显示的记录带说明
$ git log --pretty=oneline //显示的记录带版本号
```

运用`git reset`命令把当前版本回退到之前的版本：

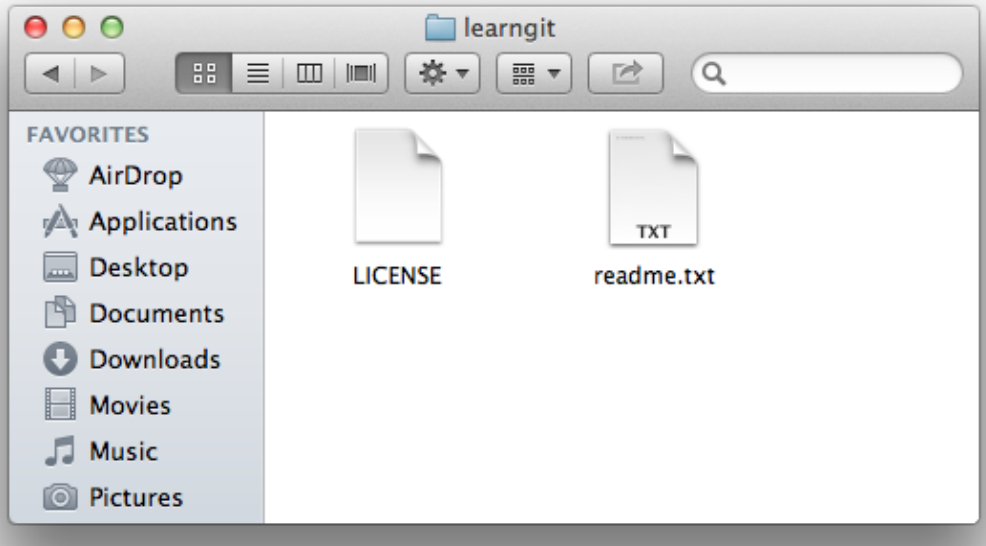
```
$ git reset --hard HEAD^ //回退到上一个版本
$ git reset --hard HEAD^^ //回退到上上一个版本.....
$ git reset --hard HEAD~100 //回退到上100个版本
$ git reset --hard <版本号前几位> //回到该版本号对应版本
```

Git提供了一个命令`git reflog`用来记录每一次命令，即可找回版本号，重返未来！

```
$ git reflog
```

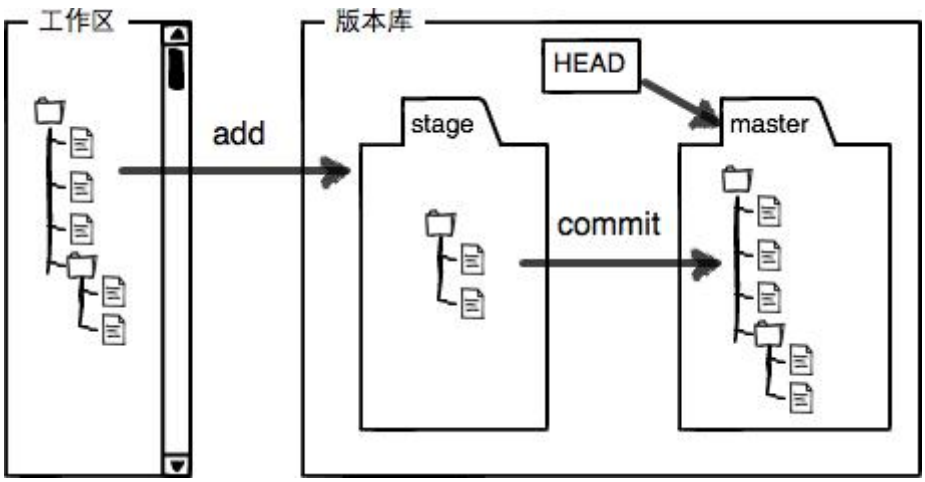
工作区和暂存区

> ***工作区（Working Directory），即电脑中能看到的目录。***



> ***版本库（Repository），工作区中的隐藏目录`.git`，这不算工作区，而是Git的版本库。***

Git的版本库里存了很多东西，其中最重要的就是stage（或者叫index）的一个暂存区，还有Git为我们自动创建的第一个分支`master`（也可以自己创建一个主分支，请参考上述可选操作），以及指向`master`（或自创主分支）的一个指针叫`HEAD`。



因此，上述`git add`命令实际上就是把要提交的所有修改放到暂存区（stage）。然后，执行`git commit`就可以一次性把暂存区所有的修改提交到分支。

可以用命令`git diff HEAD -- <文件名>`命令可以查看工作区和版本库里最新版本的区别

```
$ git diff HEAD -- <文件名>
```

撤销修改

命令`git checkout -- <文件名>`可以丢弃工作区的修改：

```
$ git checkout -- <文件名>
```

这里有两种情况：

1. <文件>自修改以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；
2. <文件>已经添加到暂存区后，又做了修改，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

若<文件>添加到暂存区后，没做修改，也还未提交，可用命令`git reset HEAD <文件名>`把暂存区的修改撤销掉，重新放回工作区：

```
$ git reset HEAD <文件名>
```

删除文件

直接在文件管理器中把文件删了，或者使用`rm`命令删了：

```
$ rm <文件名>
```

此时，你将面临两个选择：

1. 确实要从版本库中删除该文件，那就用命令`git rm`删掉，并且`git commit`

   ```
   $ git rm <文件名>
   $ git commmit -m "remove ..."
   ```

2. 删错了，此时版本库还有该文件，所以可以用命令`git checkout`将版本库里的版本替换工作区的版本以达到恢复的效果。

   ```
   $ git checkout -- <文件名>
   ```

远程仓库

> 将自己的本地版本库与Github服务器仓库进行连接，达到远程管理的效果。

获取SSH Key

第一步：创建SSH Key。在主目录下，看看有没有.ssh目录，再看看该目录下有没有`id_rsa`（私钥）和`id_rsa.pub`（公钥）两个文件。

```
$ ssh-keygen -t rsa -C "youremail@example.com"
```

> 需要将邮箱地址换成自己的邮箱地址，然后一路回车

第二步：登录Github，打开`Setting`，`SSH and GPG keys`界面

第三步：点击`New SSH key`

第四步：填上任意`Title`，在`Key`文本框里粘贴`id_rsa.pub`的内容，可用`cat`命令打开`id_rsa.pub`：

```
$ cat ~/.ssh/id_rsa.pub
```

添加远程库

第一步：登录Github，找到`Create a new repository`按钮，创建一个新的仓库

第二步：在`Repository name`填入该仓库的名称，其他保持默认设置，点击`Create repository`按钮即可创建一个新的Git仓库

第三步：根据Github的提示，在本地对应仓库下运行命令即可将本地库所有内容推送到远程：

```
$ git remote add origin <main branch> https://github.com/<Github name>/<repository name>.git
$ git push -u origin <main branch> //首次推送要加-u
```

> 若出现报错`unable to get local issuer certificate`可用如下命令将git中的sslverify关掉：
>
> ```
> $ git config --global http.sslverify false
> ```

从现在起，只要本地做了提交`commit`，就可以通过命令把

的最新更改推送至Github：

```
$ git push origin <main branch>
```

删除远程库

如果添加的时候地址写错了，或者就是想要删除远程库，可以用`git remote rm <name>`命令删除，建议先用`git remote -v`查看远程库信息：

```
$ git remote -v
$ git remote rm origin
```

从远程库克隆

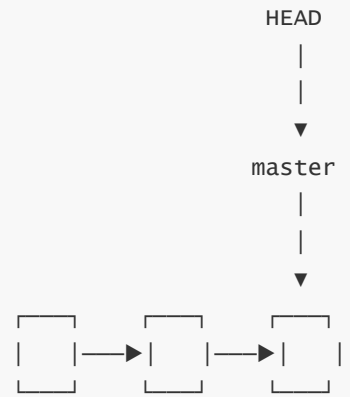
用命令`git clone`可将远程库克隆到本地：

```
$ git clone https://github.com/<Github name>/<repository name>.git
```

分支管理

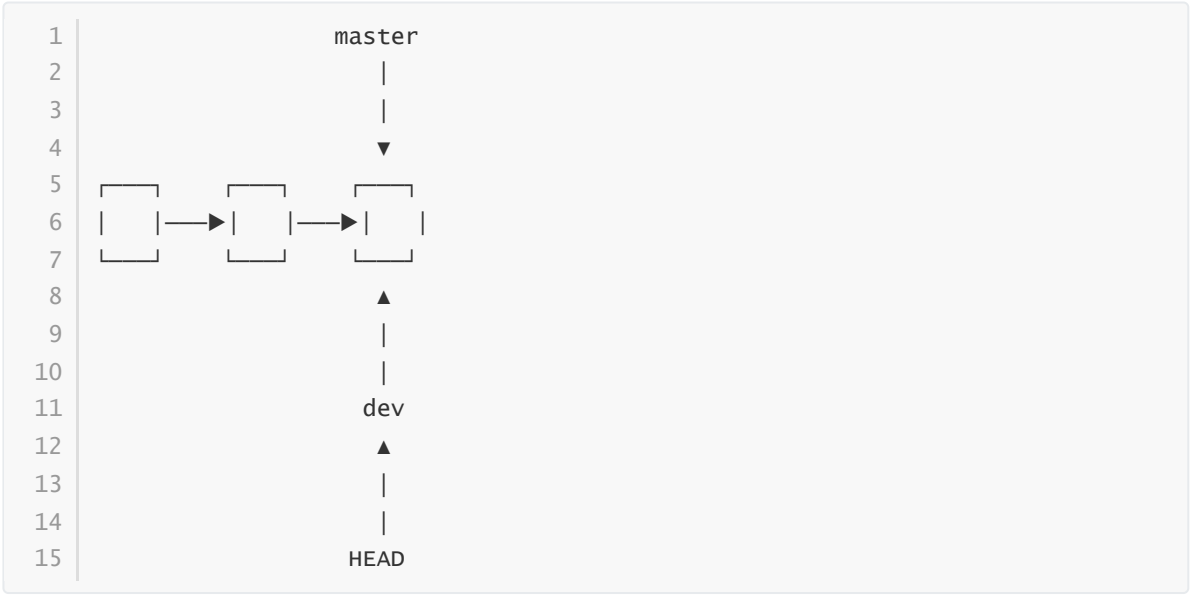
分支的概念

一开始，`master`分支是一条线，Git用`master`指向最新的提交，再用`HEAD`指向`master`，就能确定当前分支以及当前分支的提交点：

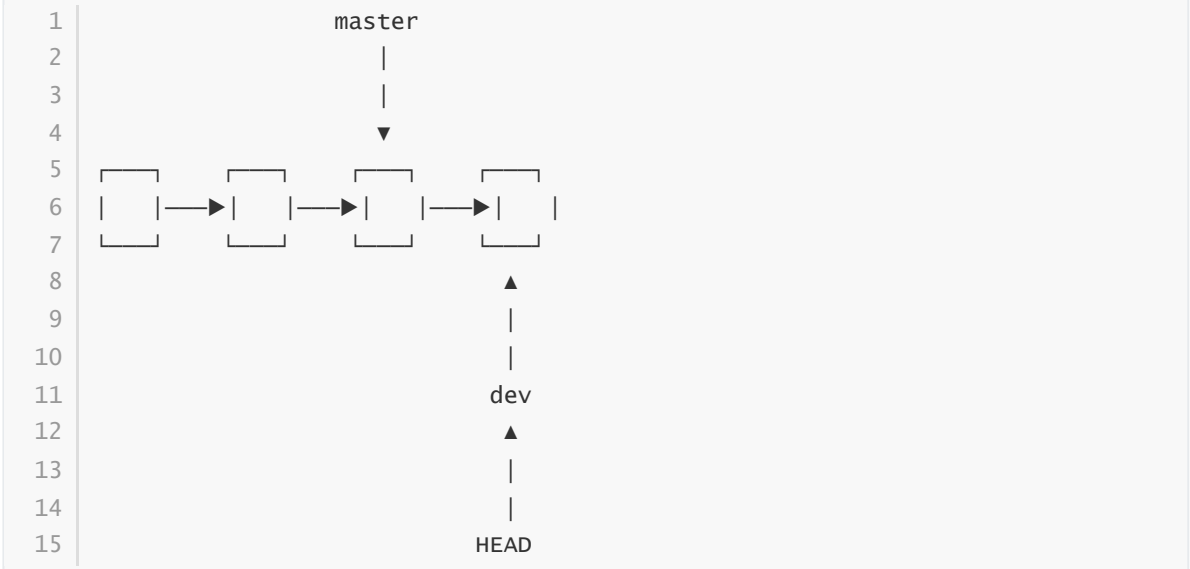


每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也会越来越长。

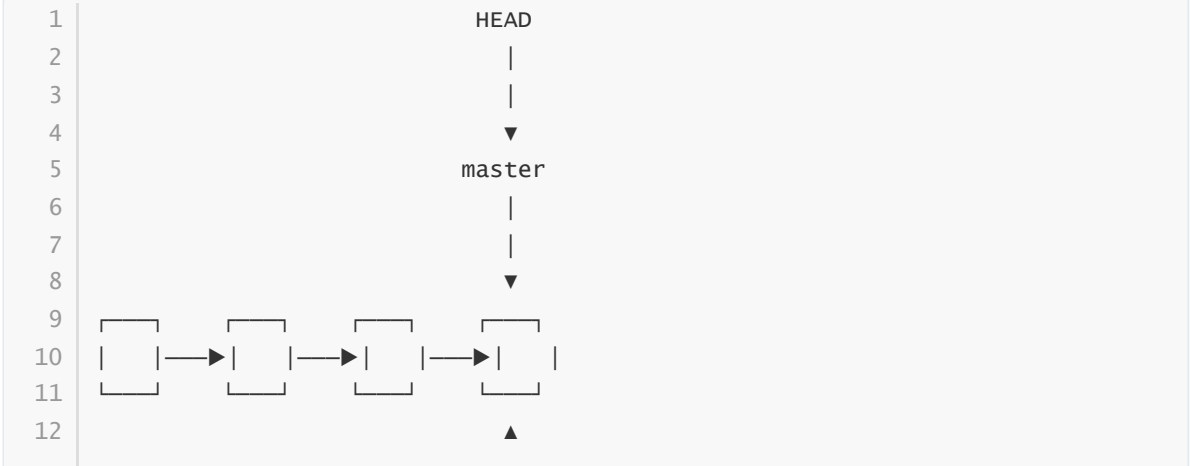
当我们创建新的分支，例如`dev`时，Git新建了一个指针叫`dev`，指向`master`相同的提交，再把`HEAD`指向`dev`，就表示当前分支在`dev`上：



此时，新提交一次后，dev指针往前移动一步，而master指针不变：



假如我们在`dev`上的工作完成了，就可以把`dev`合并到`master`上。直接把`master`指向`dev`的当前提交，就完成了合合并：



```
                                |
                                |
                               dev
```

合并完分支后，甚至可以删除`dev`分支。删除`dev`分支就是把`dev`指针给删掉，删掉后，我们就剩下了一条`master`分支：

```
                              HEAD
                                |
                                |
                                ▼
                             master
                                |
                                |
                                ▼
┌────┐     ┌────┐     ┌────┐     ┌────┐
│    │────▶│    │────▶│    │────▶│    │
└────┘     └────┘     └────┘     └────┘
```

创建与合并分支

首先，我们创建`dev`分支，然后切换到`dev`分支：

```
$ git checkout -b dev
Switched to a new branch 'dev'
```

`git checkout`命令加上`-b`参数表示创建并切换，相当于以下两条命令：

```
$ git branch dev
$ git checkout dev
Switched to branch 'dev'
```

然后，用`git branch`命令查看当前分支：

```
$ git branch
* dev
  master
```

`git branch`命令会列出所有分支，当前分支前面会标一个`*`号。

然后，我们就可以在`dev`分支上正常提交，比如对`readme.txt`做个修改，加上一行：

```
Creating a new branch is quick.
```

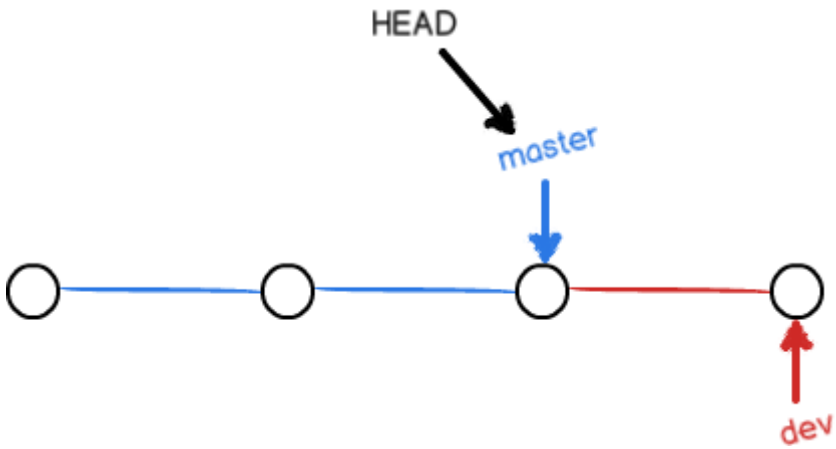
然后提交：

```
$ git add readme.txt
$ git commit -m "branch test"
[dev b17d20e] branch test
 1 file changed, 1 insertion(+)
```

现在，dev分支的工作完成，我们就可以切换回master分支：

```
$ git checkout master
Switched to branch 'master'
```

切换回master分支后，再查看一个readme.txt文件，刚才添加的内容不见了！因为那个提交是在dev分支上，而master分支此刻的提交点并没有变：



现在，我们把dev分支的工作成果合并到master分支上：

```
$ git merge dev
Updating d46f35e..b17d20e
Fast-forward
 readme.txt | 1 +
 1 file changed, 1 insertion(+)
```

git merge命令用于合并指定分支到当前分支。合并后，再查看readme.txt的内容，就可以看到，和dev分支的最新提交是完全一样的。

注意到上面的Fast-forward信息，Git告诉我们，这次合并是“快进模式”，也就是直接把master指向dev的当前提交，所以合并速度非常快。

当然，也不是每次合并都能Fast-forward，我们后面会讲其他方式的合并。

合并完成后，就可以放心地删除dev分支了：

```
$ git branch -d dev
Deleted branch dev (was b17d20e).
```

删除后，查看branch，就只剩下master分支了：

```
$ git branch
* master
```

切换分支

我们注意到切换分支使用`git checkout <branch>`，而前面讲过的撤销修改则是`git checkout -- <file>`，同一个命令，有两种作用，确实有点令人迷惑。

实际上，切换分支这个动作，用`switch`更科学。因此，最新版本的Git提供了新的`git switch`命令来切换分支：

创建并切换到新的`dev`分支，可以使用：

```
$ git switch -c dev
```

直接切换到已有的`master`分支，可以使用：

```
$ git switch master
```

使用新的`git switch`命令，比`git checkout`要更容易理解。

标签管理

创建标签

先切换到需要打标签的分支上，详见切换分支

然后，敲命令`git tag <name>`就可以打一个新标签：

```
$ git tag v1.0
```

可以用命令`git tag`查看所有标签：

```
$ git tag
v1.0
```

> *默认标签是打在最新提交的commit上的。*

也可以找到历史提交的`commit id`，对历史提交打上标签

```
$ git log --pretty=oneline --abbrev-commit //查找历史提交的commit id
$ git tag <tagname> <commit id>
```

可以用可以用`git show <tagname>`查看标签信息以及说明文字：

```
git show v0.9
```

还可以创建带有说明的标签，用`-a`指定标签名，`-m`指定说明文字：

```
$ git tag -a <tagname> -m "说明文字" <commit id>
```

操作标签

如果标签打错了，也可以删除：

```
$ git tag -d v0.1
Deleted tag 'v0.1' (was f15b0dd)
```

因为创建的标签都只存储在本地，不会自动推送到远程。所以，打错的标签可以在本地安全删除。

如果要推送某个标签到远程，使用命令`git push origin <tagname>`：

```
$ git push origin v1.0
Total 0 (delta 0), reused 0 (delta 0)
To github.com:michaelliao/learngit.git
 * [new tag]         v1.0 -> v1.0
```

或者，一次性推送全部尚未推送到远程的本地标签：

```
$ git push origin --tags
Total 0 (delta 0), reused 0 (delta 0)
To github.com:michaelliao/learngit.git
 * [new tag]         v0.9 -> v0.9
```

如果标签已经推送到远程，要删除远程标签就麻烦一点，先从本地删除：

```
$ git tag -d v0.9
Deleted tag 'v0.9' (was f52c633)
```

然后，从远程删除。删除命令也是push，但是格式如下：

```
$ git push origin :refs/tags/v0.9
To github.com:michaelliao/learngit.git
 - [deleted]         v0.9
```

要看看是否真的从远程库删除了标签，可以登陆Github查看。