

A Guide to Git and Github

Internet Service Department

February 15, 2014

Contents

1	安装与配置	2
2	Git 基本概念	2
2.1	仓库 (Repositories)	2
2.2	提交与推送 (commit and push)	3
2.3	更新与恢复	4
3	协作与远端操作	4
3.1	分支 (branches)	4
3.2	Fork 与克隆 (clone)	5
4	检阅代码	6
5	后悔药	6

1 安装与配置

1. 首先，注册一个Github帐号。
在[此处](#)下载并安装 Git for Windows，通常可以直接使用默认配置。安装好之后可以在「开始」菜单中找到「Git Bash」和「Git GUI」两个程序。
2. 打开「Git Bash」，首先设置用户名和邮箱，在命令行中键入如下命令：

```
|| $ git config --global user.name "Your Name"
|| $ git config --global user.email yourmail@server.com
```

3. 然后创建 SSH 密钥

```
|| $ ssh-keygen -C 'yourmail@server.com' -t rsa
```

4. Git Bash 会询问储存密钥的路径，方便起见可以使用其默认路径，在 windows 下即为用户文件夹。生成成功后可以在之前设定的路径中找到 `id_rsa.pub` 文件，用记事本打开，文件中的字符即为所需的 SSH 密钥。
5. 登录 Github 首页，点击 Account Setting → SSH keys → Add SSH key。title 可以随便填写，以可以判断这是自己电脑为原则。在 key 的输入框中粘贴 `id_rsa.pub` 文件中的内容，点击 Apply 即可。
测试与 GitHub 是否连接成功：

```
|| $ SSH -v git@github.com
```

返回

```
|| $ Are you sure you want to continue connecting (yes/no)?
```

时，输入 yes。最后，返回

```
|| You've successfully authenticated, but GitHub does not provide shell access.
```

时，SSH 密钥配置成功。

2 Git 基本概念

2.1 仓库 (Repositories)

仓库即为储存一个项目的代码的目录。仓库里包括了一个项目的全部代码和 README 文件。可以用本地现有的代码新建一个仓库，并上传至 Git，也可以将 Git 上已有的仓库克隆 (clone) 到本地，进行操作。

从本地新建一个仓库的操作是：

```
|| $ cd \c\your\repo\path
|| $ git init name_of_repo
```

接着，用 `git add` 命令来添加改项目中需要追踪的文件（如一个 TeX 项目中，需要追踪的文件即为 .tex 文件，.pdf 文件和其他的素材文件，而一些临时文件则不需要进行追踪）。如果需要永久性的忽略某一类型的文件，可以打开目录中的 .gitignore 文件，并将不需要跟踪的文件添加进去。

```
|| $ git add *.* $ git add *.* $ git add *.* .....
```

也可以直接添加全部文件（在 .gitignore 中添加过的文件会被忽略）

```
|| $ git add .
```

从现有的仓库克隆的方法为：

```
|| $ git clone git://github.com/...../*.*git
```

现有的仓库的克隆地址可以在项目的 Github 页面上找到。这会在当前目录下创建一个名为 “*” 的目录，其中包含一个 .git 的目录文件，用于保存下载下来的所有版本记录，然后从中取出最新版本的文件拷贝。在仓库的目录下键入 `git status` 命令，可以查看已跟踪和未跟踪的文件列表。

有多个项目时，在操作前必须用 `cd` 命令进入需要操作的项目的目录。

2.2 提交与推送 (commit and push)

在 git 的体系中，本地仓库由三部分组成：工作目录 (Working Directory)，即实际存在的文件夹。缓存区 (Index)，用来临时储存项目的改动。和 HEAD，即最后一次正式提交的结果。`git add` 命令的意义就是将有效改动添加至缓存区，准备提交。

当在本地修改项目之后，我们需要提交和推送这些修改。

首先是用 `git add` 命令来添加需要提交的文件，然后键入

```
|| $ git commit -m 'commit message'
```

提交时需要为这次提交写一些简短的说明 (commit message)，说明这次修改了哪些地方。**每次提交相当于为项目创建了一个新的版本。**提交后，所有的改动都提交至了 HEAD。

如果想省去 `git add` 这一步，可以直接用

```
|| git commit -a -m 'commit message'
```

`commit -a` 相当于：

- 自动地 add 所有改动的代码，使得所有的开发代码都列于 Index 中
- 自动地删除那些在 Index 中但在工作目录中已经被删除的文件

- 执行 `commit` 命令来提交

♣ 但是此命令并不会提交新添加的文件，所以如果有增加文件请乖乖使用 `git add`。

如果是本地的新建仓库，从来没有推送到服务器过，则应该使用如下命令

```
|| $ git remote add origin git@github.com:...../*.git
|| $ git push origin master
```

结束工作后，键入以下的命令将本地的改动提交到服务器

```
|| $ git push origin master
```

此处的 `origin` 为服务器的别名，`origin` 是默认值，也可以在 `remote` 的时候自行修改，如果在一个项目 `remote` 了多个服务器（如 Github 和 Bitbucket），该命令可以控制 `push` 到哪一个服务器。`mster` 为 `master` 分支，关于分支会在下面一节中讲到。

2.3 更新与恢复

有时在本地仓库的代码旧于服务器端的代码时（这在多人开发或者在多台设备上开发时比较常见），可以使用如下命令

```
|| $ git pull
```

该命令会抓取服务器端的最新版本的代码并将本地的代码更新。

在 `git` 的体系中，所有提交过的更改都会被保存下来，并且都是可恢复的，即在任何时刻都可以将代码回滚到过去的任何一个版本

3 协作与远端操作

3.1 分支（branches）

使用

```
|| $ git branch branch_name
```

命令创建一个分支，分支可以理解为一些相对独立的开发路线。在 `git` 中，分支可以随时无限量的被创建和被合并。举个例子，假设在新官网开发项目中，A 先生和 B 先生分别从作为开发主线的 `master` 分支中 `fork`（关于 `fork`，会在下一小节中讲到）了一个分支（我们可以称之为 a 分支和 b 分支），分别用来写前台代码和后台代码，此时 a 分支和 b 分支的开发进程是相对独立的，在 A 先生和 B 先生完成了开发之后，他们向服务器 `push` 了自己的分支，这样别人就可以抓取这些代码了。然后，可以使用

```
|| $ git checkout branch_name
```

命令来切换分支。然后，使用

```
|| $ git merge another_branch_name
```

可以合并分支，比如 A 先生键入了

```
|| $ git checkout b
|| $ git merge a
|| $ git branch -d a
```

这些命令的含义为：进入 b 分支，将 a 分支合并入 b 分支，删除 a 分支。

如果要删除的分支没有被合并到其它分支中去，那么就不能用 `git branch -d` 来删除它，需要改用 `git branch -D` 来强制删除。

在分支合并时，有时会出现合并冲突，即两个分支分别修改了同一个文件的同一个地方，此时 git 会要求手动解决冲突。

打开产生冲突的文件，可以看到如下形式的冲突解决标记：

```
|| <<<<<< a:example.html
||
|| AAAAAAA
||
|| =====
||
|| BBBBBBB
||
|| >>>>>> b:example.html
```

分割线的上面和下面为冲突行在 a 分支和欲合并的 b 分支中的内容，手动删除要抛弃的内容之后，运行一次 `git add` 来标记问题已解决。然后可以运行一次 `git status` 来确认一下。解决冲突问题之后，就可以提交 commit 了。

参考阅读：<http://www.open-open.com/lib/view/open1328069889514.html>

3.2 Fork 与克隆 (clone)

git 中最重要的概念之一就是 fork，fork 的意义为，将别人的仓库作为一个分支拷贝到自己的仓库，但是和自己创建的分支不同的是，fork 过来的项目是作为一个独立的项目存在于你自己的账户下的，并不能随意的与原项目合并（但并不是不能合并）。

fork 的方法为，在想要 fork 的项目页面点击 fork 按钮。fork 成功之后，使用

```
|| $ git clone git@github.com:.../*/*git
```

命令将项目克隆到本地，就可以开始进行开发了。申请与原项目合并的方法为，在项目页面使用 pull request 功能。项目所有者通过后即可合并

注：这篇教程的内容也作为一个项目储存在 github 上，项目主页为<https://github.com/lamons/guides>，可以拿来做 fork 和其他一些功能的实验。

4 检阅代码

```
$ git diff --cached  
$ git diff branch_name  
$ git diff HEAD HEAD^
```

`git diff` 是用于查看修改的代码信息，上面的代码中，`git diff --cached` 用来查看缓存区中的文件相对于 `HEAD` 的差异，`git diff branch_name` 用于查看某一分支相对于 `master` 分支的差异，`git diff HEAD HEAD^` 用于查看当前的 `HEAD` 与前一版本的 `HEAD` 之间的差异（一个 `^` 代表了向前推一个版本）。

```
$ git log  
$ git log -p
```

`git log` 命令用于显示开发日志，可以显示出项目从早到晚所有 `commit` 的信息。`git log -p` 则出了 `commit` 信息，还可以调出所有的代码的修改历史。

```
$ git tag v1.0  
$ git tag -l
```

`git tag` 命令可以用来为项目标记版本号。`git tag -l` 可以查看最近的版本号。

5 后悔药

`git` 的一个好处就是，它会保存你所有的完整的历史版本，所以在代码出现问题的时候，随时可以将代码回滚到任意一个历史版本。

```
$ git checkout -- <filename>
```

这条命令可以用 `HEAD` 中的文件替替换掉你的工作目录中的文件，已添加到缓存区的改动，以及新文件，都不受影响。

假如你想要丢弃你所有的本地改动与提交，可以到服务器上获取最新的版本并将你本地主分支指向到它：

```
$ git fetch origin  
$ git reset --hard origin/master
```

可以用以下的代码来回滚到过去的版本

```
$ git revert HEAD  
$ git revert HEAD^  
$ git revert commit <commit code>
```

`git revert commit <commit code>` 命令用来回滚至指定的版本，`commit code` 可以在 `github` 项目页面上找到，类似 `bd22a27dbab52c119c7ff66a1c79a8491f82e5d6` 这样。