Git 版本控制

主讲：尚硅谷 Android 组
谷粉第 46 群：252915839

Git 简介

Git 是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。Git 是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。

Git 的作用

如果你用 word 写过作文可能有这样的经历，想删除一个段落，但觉得这段就算不用在这用在别的地方一样很好？有办法，先把当前文件“另存为……”一个新的 word 文件，再接着改，改到一定程度，再“另存为……”一个新文件，这样一直改下去，最后你的 word 文档变的十分混乱：

过了一年，突然有一天你想找回被删除的文字，但是已经记不清删除前保存在哪个文件里了。

看着一堆乱七八糟的文件，想保留最新的一个，然后把其他的删掉，又怕哪天会用上，还不敢删，真郁闷。

于是你想，如果有一个软件，不但能自动帮我记录每次文件的改动，还可以让同事协作编辑，这样就不用自己管理一堆类似的文件了，也不需要把文件传来传去。如果想查看某次改动，只需要在软件里瞄一眼就可以，岂不是很方便？

如果你有过上述的困惑 那么相信 Git 一定不会让你失望

Git 和 SVN 的区别

SVN 是集中式分布管理系统 Git 是分布式管理系统。OK 这就是他们最大的区别。

看完以后是不是有点想骂街的赶脚。哈哈.....别急别急。看来有些东西用太专业的词的确难以让人理解。

大家用过 svn 的人应该都知道我们合并代码 分支 回滚 一系列的版本控制操作都必须在服务器存在的前提下才能完成，比如现在我刚写了一个支付的模块在没有服务器的前提下我是不能提交的，然后又写了一个分享模块，结果分享模块出现了重大问题，想回到写完支付模块的状态发现根本回滚不了，因为没有服务器。这个时候就觉得太尴尬了。而我们的 Git 就不需要他可以在自己的电脑上去处理，这个时候小伙伴又晕了没有服务器怎么处理，其实是这样的，他会对当前电脑上的代码进行版本控制，连上服务器那根没有问题了。也就是说 Git 主要由本地版本控制和网络版本控制合在一起。这个时候你在本地的任何操作就都没有问题了。

本地版本控制

在 Windows 上安装 Git

从 https://git-for-windows.github.io 下载 然后按默认选项安装即可。安装完成后，在开始菜单里找到“Git”->“Git Bash”，蹦出一个类似命令行窗口的东西，就说明 Git 安装成功！

创建版本库

什么是版本库呢？版本库又名仓库，英文名 repository，你可以简单理解成一个目录或者数据库，这个目录里面的所有文件都可以被 Git 管理起来，每个文件的修改、删除，Git 都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

```
$ cd d:         //进入D盘目录
$ mkdir demo    //创建一个文件夹名叫demo
$ cd demo       //进入这个demo里
$ pwd           //查看这个文件夹的详细路径
$ git init    //初始化我们的仓库
Initialized       empty       Git       repository       in
C:/Users/Administrator/Desktop/demo2/.git/
```

扩展一下 cd .. 是退出当前目录和我们的返回是一样的
我在桌面创建的仓库 大家找到你创建的文件夹进去看一下是不是有了一个.git 的文件恭喜你初始化成功了，如果没有可以在你的 系统的文件夹选项-查看-显示隐藏的文件
或者$ ls -ah //ls 是查看当前目录下有哪些文夹 -ah 是隐藏的也显示

./ ../ .git/
是不是看到了呢。

添加文件

进入我们刚才创建的仓库，在里面创建一个 readme.txt 写上一句话“第一个文件”保存。

```
$ git status   //查看当前的状态
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be
committed)
        readme.txt
nothing added to commit but untracked files present (use
"git add" to track)
//    是不是看到我们的 readme.txt 是一个红色这说明我们的文夹还没有添加到仓库里

$ git add readme.txt   //把我们的文件添加到仓库

$ git status
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   readme.txt
//   是不是变绿了这说明我们已经添加成功了 大家再看这句话
(use "git rm --cached <file>..." to unstage) //执行这条命令
就可以删除

我们继续用命令git commit告诉Git，把文件提交到仓库
$ git commit -m "第一次提交"  //-m  是写注释这个必须得写主要是用
来对你所提交内容的一个描述
[master (root-commit) d34b337] 第一次提交
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt

$ git status   //当我们再看一下状态是不是已经没有显示文件了这就说
明添加成功了
On branch master
nothing to commit, working directory clean
```

版本回退

我们模拟修了好多次 这次在我们的 readme.txt 里再加上一句话 “第二次修改”
然后我们执行

```
$ git status
  ......
        modified:   readme.txt
```

是不是又变红了 没错因为我们修改过了所以要重新提交 和上面一样要add 和commit操作 可以多进行几次修改。

```
$ git log  //查看记录
commit 8267f0e5b335161c4ffb270343b0565769cade15
Author: wangfeilong <flyloong_job@163.com>
Date:   Wed Aug 31 11:53:32 2016 +0800

    第二次修改

commit d34b3379d0766c2c11a856d1b20a4d988af523e3
Author: wangfeilong <flyloong_job@163.com>
Date:   Wed Aug 31 09:28:35 2016 +0800

    第一次提交
```

如果觉得太乱 可以加上以下参数 --pretty=oneline

```
$ git log --pretty=oneline
8267f0e5b335161c4ffb270343b0565769cade15 第二次修改
d34b3379d0766c2c11a856d1b20a4d988af523e3 第一次提交
```

上面的命令大家可以看到我们的提交记录和修改记录
上面一大串的数字其实是Git的commit id 是一个SHA1计算出来的一个非常大的数字，用十六进制表示

假如我们现在的需求要回到第一次提交的时候，可以如下操作

```
$ git reset --hard HEAD^  //回退到上一个版本
HEAD is now at d34b337 第一次提交

Administrator@PC-201608231916 MINGW64 ~/Desktop/demo2
(master)
$ git log
commit d34b3379d0766c2c11a856d1b20a4d988af523e3
Author: wangfeilong <flyloong_job@163.com>
```

```
Date:   Wed Aug 31 09:28:35 2016 +0800
```

    第一次提交

哈哈是不是到我们的第一次提交了呢 再看我们的文件内容是不是也变了

那比如说回退了三次我现在又想回到最开始的怎么办，只要你的id还能找到那就没有问题我们再试一下

```
$ git reset --hard 8267f0e5b33516  //找到原来版本的ID可以只写前几位
HEAD is now at 8267f0e 第二次修改

Administrator@PC-201608231916 MINGW64 ~/Desktop/demo2 (master)
$ git log
commit 8267f0e5b335161c4ffb270343b0565769cade15
Author: wangfeilong <flyloong_job@163.com>
Date:   Wed Aug 31 11:53:32 2016 +0800

    第二次修改

commit d34b3379d0766c2c11a856d1b20a4d988af523e3
Author: wangfeilong <flyloong_job@163.com>
Date:   Wed Aug 31 09:28:35 2016 +0800

    第一次提交
```

是不是又回到了我们指定要回到的版本，就是这么爽。

那比如我的ID找不到了怎么办，请大家看下面

```
$ git reflog   //用来查询我们前面的操作
8267f0e HEAD@{0}: reset: moving to 8267f0e5b33516
d34b337 HEAD@{1}: reset: moving to HEAD^
8267f0e HEAD@{2}: commit: 第二次修改
d34b337 HEAD@{3}: commit (initial): 第一次提交
```

看左边是不是出来了呢

远程仓库

所谓的远程仓库类似我们的 SVN 服务器用来帮我们管理代码，这次我们就选用 github 当然还有很多类似的服务器我们就不一一说明了

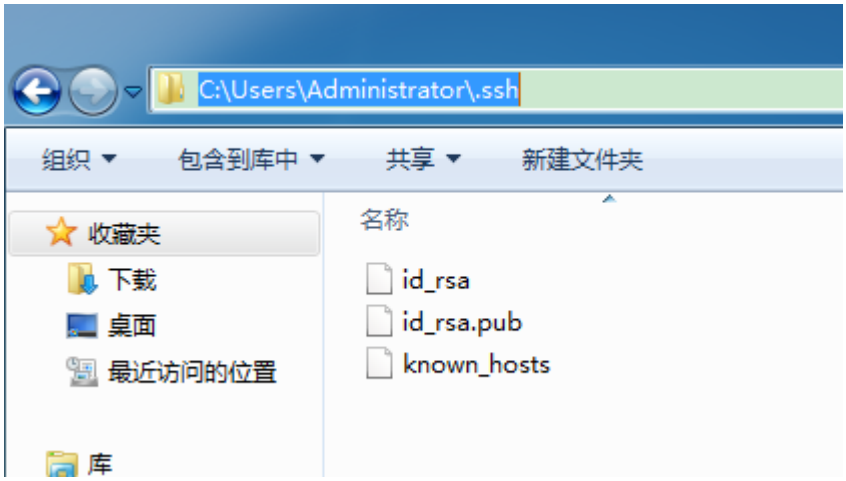
Github 的简单使用

注册 github

Github 网址 ： https://github.com/
首先注册一个账号 首页就是注册的界面 登录的界面在左上角 singin 是登录

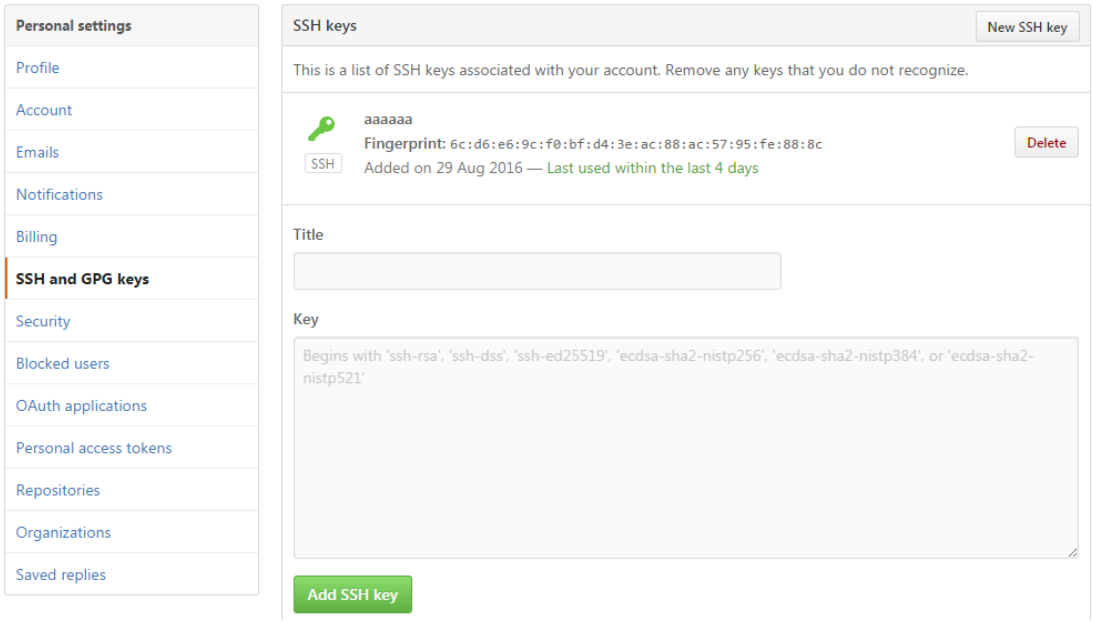
SSH

由于你的本地 Git 仓库和 GitHub 仓库之间的传输是通过 SSH 加密的，所以，需要一点设置：
第一步 ：创建 SSH Key。在用户主目录下，看看有没有.ssh 目录，如果有，再看看这个目录下有没有 id_rsa 和 id_rsa.pub 这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开 Shell（Windows 下打开 Git Bash），创建 SSH Key



重点看路径 当然我这已经创建了 如果没有我们就执行以下操作
$ ssh-keygen -t rsa -C "youremail@example.com"
//后的邮箱是你注册的邮箱然后一路回车，使用默认值即可
这样就可以在我们刚才的路径里找到.ssh 目录，里面有 id_rsa 和 id_rsa.pub 两个文件，这两个就是 SSH Key 的秘钥对，id_rsa 是私钥，不能泄露出去，id_rsa.pub 是公钥，可以放心地告诉任何人。

第 2 步：登陆 GitHub，打开“settings”，“SSH Keys”页面：



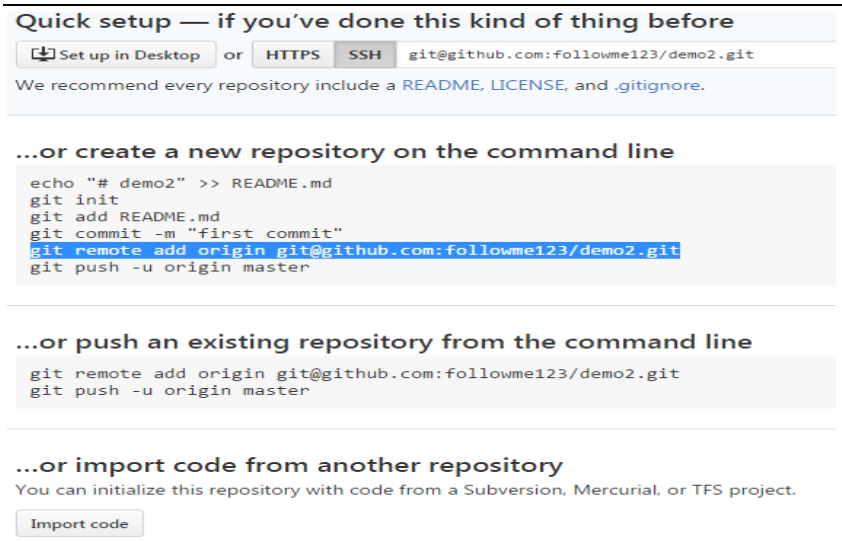
Tittle 随意写
Key　id_rsa.pub 里的内容粘过来就好 然后点击 Add SSH key

解释 ： 为什么 GitHub 需要 SSH Key 呢？因为 GitHub 需要识别出你推送的提交确实是你推送的，而 Git 支持 SSH 协议，所以，GitHub 只要知道了你的公钥，就可以确认只有你自己才能推送。当然，GitHub 允许你添加多个 Key。只要把每台电脑的 Key 都添加到 GitHub,就可以在每台电脑上往 GitHub 推送了。在 GitHub 上免费托管的 Git 仓库，任何人都可以看到（但只有你自己才能改）。如果你不想让别人看到 Git 库，有两个办法，一个是交费，让 GitHub 把公开的仓库变成私有的，这样别人就看不见了（不可读更不可写）。另一个办法是自己动手，搭一个 Git 服务器

本地仓库和远程仓库进行同步

第一步在远程仓库建立一个新的仓库


View 2 new broadcasts
Your repositories 3
New repository
Find a repository...
All
Public
Private
Sources
Forks
demo
aaa
test
Owner
Repository name
followme123
/
填写仓库的名称
Great repository names are short and memorable. Need inspiration? How about furry-parakeet.
Description (optional)
描述可以不写
Public
仓库是公有的免费 但是别人都可以看见和下载
Anyone can see this repository. You choose who can commit.
Private
仓库是私有的但是需要效一定费用
You choose who can see and commit to this repository.
Initialize this repository with a README
生成README文档可以不选
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.
Add .gitignore: None
Add a license: None
Create repository



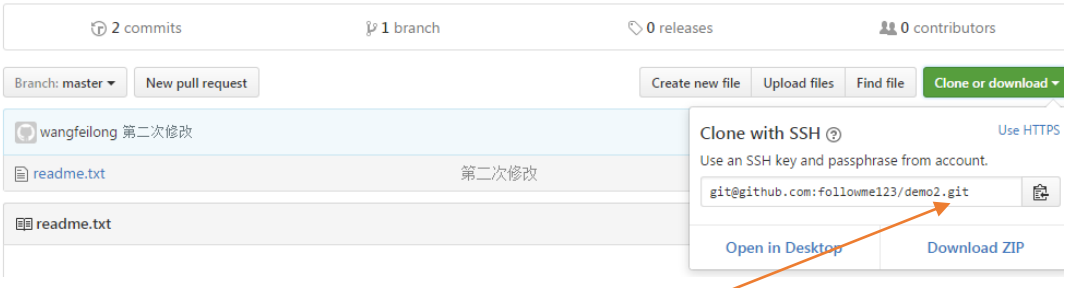
在 GitHub 上的这个仓库还是空的，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到 GitHub 仓库。

```
$ git remote add origin
    git@github.com:followme123/demo2.git //关联仓库
$ git push -u origin master
```

//把本地库的内容推送到远程，用 git push 命令，实际上是把当前分支 maste 推送到远程。由于远程库是空的，我们第一次推送 master 分支时，加上了-u 参数，Git 不但会把本地的 master 分支内容推送的远程新的 master 分支，还会把本地的 master 分支和远程的 master 分支关联起来，在以后的推送或者拉取时就可以简化命令。

我们检测下看有没有成功 可以刷新一下我们建好的仓库看我们本地的仓库数据有没有 Push 到服务器上，如果有的话 OK 那么你成功了

从远程仓库 Clone 代码



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

git@github.com:michaelliao/gitskills.git 的地址就是我们图片上的地址

分支管理

什么是分支

举个例子来说 我们小时候经常在写作业的时候有这种想法，如果我能分身就好了即可以写作业，又可以打游戏。现在 GIT 的分支就满足了我们小时候的愿望嘿嘿。当你需要同时做两件事的时候你就分身一个人写作业，一个人看电影，都搞好以后再合并大功告成全部做完。是不是很爽.....

分支在实际中有什么用？假设你准备开发一个新功能需要两周才能完成，第一周你写了 50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。
现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。
无论创建、切换和删除分支，Git 在 1 秒钟之内就能完成！无论你的版本库是 1 个文件还是 1 万个文件。

创建与合并分支

```
$ git branch  //查看当前分支 绿色就代表现在在当前的分支是master
* master
```

下面我们来创建分支 dev

```
$ git checkout -b dev
Switched to a new branch 'dev'
Administrator@PC-201608231916 MINGW64 ~/Desktop/demo2
(dev   //创建完成后直接就进入到我们的分支了
```

再次查看分支 是不是变了呢

```
$ git branch
* dev
```

```
  Master
```
切换分支
```
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

Administrator@PC-201608231916 MINGW64 ~/Desktop/demo2
(master) //是不是已经切换成功了呢 嘿嘿
```
其实分支相当于两个用户 在每个用户操作的数据 只有在当前分支能看到大家可以自行尝试在切换分支 进行数据修改 肯定看到的也不一样
如果想把两个分支的数据合在一起就要使用
```
$ git merge dev  //合并分支的时候要在master
```
删除分支
```
$ git branch -d dev
```
通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。
如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。
下面我们实战一下--no-ff方式的git merge：
首先，仍然创建并切换dev分支：

```
$ git merge --no-ff -m "merge with no-ff" dev
```

哇!!! 讲了这么多 相信很多朋友都在抱怨这么多的命令行，实在记不住啊!!!

没关系其实 GIT 也有视图化工具比如 sourcetree.

不过我仍然建议大家先用用命令行，再用工具这样大家会更容易理解。就这样了 有问题大家可以加我们的官方群