1. 历史背景

在Git(分布式版本控制系统)面世之前，Linux内核是一直使用的是BitKeeper，但是在这个过程中.有的人太厉害了，就想利用自己的知识来破解这个BitKeeper的协议。但是呢，授权给Linux社区用BitKeeper的公司就不干了，本来就免费给你们使用，怎么能够这样呢，就收回了免费试用权。Linux的大BOSS觉得问题不大，就自己花了两周时间使用C创造一个非常原始的分布式版本控制系统，这也太厉害了吧!

而github是当今世界上最大的代码托管平台，它作为一个开源的项目免费提供Git存储。我 们一 起来了解一下吧。

2. Git初学

2.1 分布式版本控制系统?

其实对于分布式版本控制系统，与其相对应的是集中式版本控制系统（cvs、svn),啊？怎么又多了个概念，没事儿，咱们慢慢来，先来理解什么是集中式版本控制系统。

集中式版本控制系统，从字面上的意思了解到，版本库是集中存在中央服务器，而我们处理文件的时候，需要把文件从中央服务器中调取，处理完了之后，在推送到中央服务器。说的通俗点，一个课题组下面有两个成员：小袁和小振。BOSS有个课题，需要小袁完成，小袁就去BOSS哪里获取，然后辛苦完成之后，再提交给BOSS。小振也想要怎么办呢？也是采取上述方法，从BOSS哪里获取课题，然后提交给BOSS。这样的话，小袁和BOSS都在福州还好说，但是一个在北京一个在福州怎么办，怎么提交啊。是的，集中式版本控制系统必须联网才能工作，在局域网内还行，但是在互联网上，就有点捉急了。特别是像我们实验室网速极慢，使用这样系统就有点难受了。

分布式版本控制系统，咱从字面意思也可理解，那就是分布的啦，就没有“中央服务器”，每个人的电脑都是一个版本库啦，比如小袁和小振需要完成BOSS一个课题，小袁做了那些可以推送给小振或者BOSS，小振和BOSS都是可以这样做，这样三方就可以相互共同促成一个课题的完成。那么，小袁和BOSS一个在北京一个在福州怎么办呢，没事儿，小振这里也有啊，这个课题还可以继续下去啊!

在实际生活中，万一小振生病了怎么办，怎么告诉她我的课题进展情况呢？因此，分布式版本控制系统通常也有一台充当“中央服务器”的电脑，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样干活，只是交换修改方便而已。

其实Git除了不用联网，还有强大的分支管理，好的正文就此正式开始啦

2.2 开始下载Git啦

你如果是Linux系统（Debian或Ubuntu），就在终端命令行输入：sudo apt-get install git

如果是Mac，直接从AppStore安装Xcode，安装后，选择菜单“Xcode”->"Preferences"，在弹出窗口找到“Downloads”，选择“command line tools”，点“install”就可。

如果是Windows，去Git 官网 下载就行，然后按照提示进行安装就行了，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!

安装完之后，还需要设置一下，打开你的命令行输入：

```
$ git config --global user.name "Your Name"

$ git config --global user.email "email@example.com"
```

名字和邮件咱可以随便弄一个

注意git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

2.3 创建库

在Git里面称之为仓库（repository），咱们可以简单的理解为一个目录，Git就充当监控员，可以对这个目录下的每一个文件的修改、追踪、删除进行追踪。这样的话，如果你想知道你曾经对你目录下的文件干了啥，就可以调取Git这个“监控”了。既然如此咱们就开始创建“仓库”了吧。在你的终端输入：

```
$ mkdir syshbmc

####$代表的是你的命令提示符（非root）

$ cd syshbmc

####创建一个syshbmc的文件夹（名字看你啰）

$ pwd

/c/Users/Lucas/syshbmc

####看是否创建目录，并且切换到该目录下
```

然后我们就让Git来监管了

```
$ git init

###Initialized empty Git repository in /YOUR/PATH/syshbmc/.git/
```

哇，Git都提示你这是个空仓库了（empty Git repository），然后它害生成了.git文件夹，当然了，你的“监控器”得有个地方储存你监控的东西吧，就是这个文件夹。tips：不要随意修改这个文件夹哟，要不然别人怎么放监控的东西啦。当然别人也晓得有的同学好奇心太强了，他是一个隐藏的目录，你需要：ls -al命令才能看见的啦。

当然了，Git只能监控文本文件（TXT文件、网页、程序代码等）。但是，像图片、视频等一些二进制文件只晓得它改了啥（如视频从1M改到12M）具体改变了啥，就监控不了啊。

好了，废话少说，我们继续在你/YOUR/PATH/syshbmc这个文件夹下操作：

```
$ vim readme.txt

####进入readme.txt了，此时点击键盘上字母“i”，并且敲打以下两行字。

I love learnning sysbiology

I want to use git to learn more

###然后点击键盘上“Esc”，再按“Shift 和 ： ”，松开，敲入“wq！”即可。
```

之后我们就让Git来监控（添加到缓存区）啦，首先我们得把我们得文件放在仓库里面：

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

然后让Git知道它要监控那个啦（把文件提交到head（指针））

```
$ 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

上面三行是Git反馈给你它晓得了！
```

这个git commit命令，-m后面输入的是本次提交的说明，最好是你晓得你这次改动得意义是啥子，这样下次调取监控就知道自己干了啥。如果不想输入得话，哼！自己百度参数。

git commit命令执行成功后会告诉你，1 file changed：1个文件被改动（我们新添加的readme.txt文件）；2 insertions：插入了两行内容（readme.txt有两行内容）。

让Git晓得监控啥，就很简单的两步

```
$  git add file1.txt

$ git add file2.txt file3.txt

$ git commit -m "add 3 files."

####咦，怎么不一样，其实可以让Git一次监控多个鸭!
```

在这些步骤中容易遇到得troubles，（其实每一步完成，除了完成提示信息，其它没有信息就是最好的信息）：

Q:在windows下，我的PATH怎么一直在报错鸭?

A:是因为你的路径里面有中文啦，改成英文就可

Q：输入git add readme.txt，得到错误：fatal: not a git repository (or any of the parent directories)。

A：Git命令必须在Git仓库目录内执行（git init除外），在仓库目录外执行是没有意义的。

Q：输入git add readme.txt，得到错误fatal: pathspec 'readme.txt' did not match any files。

A：添加某个文件时，该文件必须在当前目录下存在，用ls或者dir命令查看当前目录的文件，看看文件是否存在，或者是否写错了文件名。

3. 版本库的一些操作

3.1 查看当前文件状态

我们用2.3修改readme.txt的方法，再加入两行：

```
I love fafu

I love hbmc
```

我们退出编辑，回到命令行运行git status看看结果呗

```
$ git status

on branch master

Changes not staged for commit:

(use "git add ..." to update what will be committed)

(use "git restore ..." to discard changes in working directory)

modified : readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

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

是的啊，上一章节，我们修改可是有让Git监控。没有的话，我看看状态，它会提示我们，你得提交了，真棒!

但是啊，BOSS突然问你上次bug是什么，你修改了啥。哈！莫慌，监控功能上线了。

你可以在命令行输入：

```
$ git diff readme.txt
diff --git a/readme.txt b/readme.txt

index 7ead6cd..0c2349b 100644

--- a/readme.txt

+++ b/readme.txt

@@ -1,2 +1,4 @@

I love learnning sysbiology

I want to use git to learn more

I love fafu

I love hbmc
```

这样我就可以看到我们修改的具体内容了，然后可以使用git log来看看自己提交的历史记录了。

```
$ git log

commit fc4f7f250f7eaf3150ec6b7831874affaab87196 (HEAD -> master)

Author: Your Name email@example.com

Date:   Sun Apr 25 11:57:56 2021 +0800

change readme file

commit 97d4b51fd892e5ad23c837b0d1bd39daba92ed3e

Author: Your Name email@example.com

Date:   Sun Apr 25 11:38:35 2021 +0800

wrote a readme file
```

哈哈，蛮神奇的吧，所以就要备注好你的信息啦，然后在开始就要备注你的姓名和邮箱啦鸭。

但是，上面给的信息太多了，该怎么办呢?

可以使用一个参数：--pretty=oneline命令

```
$ git log --pretty=oneline

fc4f7f250f7eaf3150ec6b7831874affaab87196(HEAD -> master) change reademe file

97d4b51fd892e5ad23c837b0d1bd39daba92ed3e wrote a readme file
```

3.2 回到从前

是不是感觉学习得越来越有意思了，没错，就像玩游戏打小怪兽。每通过一关就会存档保存，等你哪一天想玩以前某一关，就可以去存档里面找。其实，git也有这样功能鸭。

我们在上一节中可以看到，我们打了两关，存档两次。那么我想回到最开始得第一关，就可以使用git reset命令：

```
$ git reset --hard HEAD^

HEAD is now at 97d4b51 wrote a readme file
```

```
$ cat readme.txt

I love learnning sysbiology

I want to use git to learn more
```

果真，回到了第一个版本！

但是，你的BOSS突然说，你刚刚那一关没有错，不必删掉，天哪，我该怎么回到第二关啊！

哈，我胡汉三还是有办法回来的

Git有版本回退得功能，但是当你窗口没有关闭，就是之前你的git log 的命令展示得有你的第二关得编号fc4f7。

```
git reset --hard fc4f7

HEAD is now at fc4f7f2 change readme file

$ cat readme.txt

I love learnning sysbiology

I want to use git to learn more

I love fafu

I love hbmc
```

咦，咱们又回到第二关了。

3.3 删除文件

为了将怎么删除文件，那我们先来创建一个文件

```
$ touch test.txt

$ git add test.txt

$ git commit -m "add test.txt"
[master 371c5a9] add test.txt

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 test.txt

$ rm test.txt

$ git status

On branch master

Changes not staged for commit:

(use "git add/rm ..." to update what will be committed)

(use "git restore ..." to discard changes in working directory)

deleted:    test.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

是的，Git提醒了，你小子删掉了一个文件啊，听着，你有两个选择：

1、确定要删除该文件，就用命令git rm删掉

```
$ git rm test.txt
```

```
rm 'test.txt'

$ git commit -m "remove test.txt"

[master adc676a] remove test.txt

1 file changed, 1 deletion(-)

delete mode 100644 test.txt
```

2、我不小心删错了，莫慌，也有办法处理

```
$ git checkout --test.txt
```

caution：如果，你创建了test.txt这个文件，但没有git add和git commit这两步，无法恢复的。

4 远程仓库

4.1 基本概念

Git是分布式版本控制系统，同一个Git仓库，可以分布到不同的机器上。怎么分布呢？最早，肯定只有一台机器有一个原始版本库，此后，别的机器可以“克隆”这个原始版本库，而且每台机器的版本库其实都是一样的，并没有主次之分。

你肯定会想，至少需要两台机器才能玩远程库不是？但是我只有一台电脑，怎么玩？

其实一台电脑上也是可以克隆多个版本库的，只要不在同一个目录下。不过，现实生活中是不会有人这么傻的在一台电脑上搞几个远程库玩，因为一台电脑上搞几个远程库完全没有意义，而且硬盘挂了会导致所有库都挂掉，所以我也不告诉你在一台电脑上怎么克隆多个仓库。

实际情况往往是这样，找一台电脑充当服务器的角色，每天24小时开机，其他每个人都从这个“服务器”仓库克隆一份到自己的电脑上，并且各自把各自的提交推送到服务器仓库里，也从服务器仓库中拉取别人的提交。

完全可以自己搭建一台运行Git的服务器，不过现阶段，为了学Git先搭个服务器绝对是小题大作。好在这个世界上有个叫GitHub的神奇的网站，从名字就可以看出，这个网站就是提供Git仓库托管服务的，所以，只要注册一个GitHub账号，就可以免费获得Git远程仓库。

在继续阅读后续内容前，请自行注册GitHub账号。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1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：

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

这个邮件地址，就你在最开始那两行基础设置的邮件地址鸭，然后就可以按键盘上的“Enter”并按照提示，来根据你自己个人喜好来设置的啦

一路“Enter”下来，然后

```
$ ls -al

###你就可以找到.ssh文件夹

$ cd .ssh

ls -al

###就可以看到了id_rsa和id_rsa.pub两个文件
```

现在你的仓库有两把钥匙，一把钥匙是你的私密，另一把是公共的鸭。其中，id_rsa是你的私密钥匙，这个你要好好保管。而id_rsa.pub是你的公共钥匙，这个你可以给别人看。

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

第3步： 然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id_rsa.pub文件的内容,之后点“Add Key”，你就应该看到已经添加的Key：

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，而不是别人冒充的，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。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，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。

如果，你没有设置相关隐私，你上传到Github上的东东，所有人都可以看得到的，请耗子尾汁。

4.2 添加远程库

现在的情景是，你已经在本地创建了一个Git仓库后，又想在GitHub创建一个Git仓库，并且让这两个仓库进行远程同步，这样，GitHub上的仓库既可以作为备份，又可以让其他人通过该仓库来协作，真是一举多得。

首先，登陆GitHub，然后，在右上角找到“Create a new repo”按钮，创建一个新的仓库：

在Repository name填入sysbhmc，其他保持默认设置，点击“Create repository”按钮，就成功地创建了一个新的Git仓库：

目前，在GitHub上的这个learngit仓库还是空的，GitHub告诉我们，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。

现在，我们根据GitHub的提示，在本地的sysbhmc运行命令：

```
$ git remote add origin git@github.com:yuanzhen-lucas/sysbhmc.git
```

可不要傻傻地复制，要把yuanzhen-lucas换成你自己的Github用户名！！

添加后，远程库的名字就是origin，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库。

下一步，就可以把本地库的所有内容推送到远程库上：

```
$ git push -u origin master
Counting objects: 20, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (15/15), done.
Writing objects: 100% (20/20), 1.64 KiB | 560.00 KiB/s, done.
Total 20 (delta 5), reused 0 (delta 0)
remote: Resolving deltas: 100% (5/5), done.
To github.com:michaelliao/learngit.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

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master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。

推送成功后，可以立刻在GitHub页面中看到远程库的内容已经和本地一模一样：

只要本地作了提交，就可以通过命令：

```
$ git push origin master
```

把本地master分支的最新修改推送至GitHub，现在，你就拥有了真正的分布式版本库！

如果添加的时候地址写错了，或者就是想删除远程库，可以用git remote rm 命令。使用前，建议先用git remote -v查看远程库信息：

```
$ git remote -v
origin  git@github.com:michaelliao/learn-git.git (fetch)
origin  git@github.com:michaelliao/learn-git.git (push)
```

然后，根据名字删除，比如删除origin：

```
$ git remote rm origin
```

此处的“删除”其实是解除了本地和远程的绑定关系，并不是物理上删除了远程库。远程库本身并没有任何改动。要真正删除远程库，需要登录到GitHub，在后台页面找到删除按钮再删除。

###4.3 从远程仓库克隆
上次我们讲了先有本地库，后有远程库的时候，如何关联远程库。

现在，假设我们从零开发，那么最好的方式是先创建远程库，然后，从远程库克隆。

首先，登陆GitHub，创建一个新的仓库，名字叫sysbiokills：

然后，我们勾选Initialize this repository with a README，这样GitHub会自动为我们创建一个README.md文件。创建完毕后，可以看到README.md文件：

现在，远程库已经准备好了，下一步是用命令git clone克隆一个本地库：

```
$ git clone git@github.com:michaelliao/sysbioskills.git
Cloning into 'gitskills'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 3
Receiving objects: 100% (3/3), done.
```

注意把Git库的地址换成你自己的，然后进入gitskills目录看看，已经有README.md文件了：

```
$ cd gitskills
$ ls
README.md
```

如果有多个人协作开发，那么每个人各自从远程克隆一份就可以了。

你也许还注意到，GitHub给出的地址不止一个，还可以用https://github.com/yuanzhen-lucas/sysbioskills.git这样的地址。实际上，Git支持多种协议，默认的git://使用ssh，但也可以使用https等其他协议。

使用https除了速度慢以外，还有个最大的麻烦是每次推送都必须输入口令，但是在某些只开放http端口的公司内部就无法使用ssh协议而只能用https。

参考

1、廖雪峰的Git教程

2、Pro Git（中文版）

3、git 官方文档