JHU Data Science Series : The Data Scientist's Toolbox

Author : Yunqiu Xu

`JHU` `Github` `Coursera` `DataAnalysis` `R`

- Some overview of data science;
- Basic R tutorials;
- Git tutorials;

Week 1

- The personal websites of 3 faculties
- Install R Studio
- Set up a Github

1.1 Some important R functions

```
?+the function you dont know #help
help.search("the function you want to know")
args("the function") #get arguments
```

1.2 Stack Overflow tag '[r]'

- R mailing list
- CrossValidated
- R programming content
- Getting and cleaning data
- reproducible research
- regression model
- machine learning

1.3 Installation of R packages from CRAN

```r
a <- available.packages() #get information about Packages
head(rownames(a),3) #give the names of the first 3 packages

install.packages("package_name")
install.packages(c("name1","name2","name3"))

library(package_name) #when the installation is ended, you need to loa
d it
search()  #find all packages that have been loaded
```

Week 2

- Introduction to command line interface
- Git bash
- directory : the so called "folder" , tree shape
  - special one:root- "/"
  - path,redirectory(版本库，该目录内所有文件都可以被git管理)

2.1 Use the git (local repository)

2.1.1 Version control

```
git config --global user.name "Yunqiu Xu"
git config --global user.email "your email"
git config --list
exit
```

2.1.2 Set a directory: `mkdir+cd+pwd+init`

```
mkdir name  #make a new directory
cd name  #switch your directory to this folder
pwd  #show your current situation
git init  #initialization
```

2.1.3 ls command: show files and folders

```
ls -a #list hidden and unhidden ones
ls -al #list details
touch file #create an empty file
```

2.1.5 Add a new file to your directory: add+commit+push

```
#思路：初始化-添加-提交到本地仓库，后面可以再接上push将本地文件推送至github
git init #先将仓库初始化
git add file #添加
    git add -u #update tracking for changed files
    git add -A #do both of these
git commit -m"本次提交的说明，可以为任意内容" #提交
git push #update these changes to your remote repo
```

2.1.6 Make some changes: status/diff

```
git status #检查仓库内文件的状态，若被修改会有显示(如修改文档内容，会显示"modified ***")
git diff #检查修改了什么内容
#如果检查无误，可以将修改后的内容进行add并且commit
```

2.1.7 版本回退：回溯到最近的一个commit（此处可认为是存盘点）

```
git log #由近及远显示提交日志版本号commit ID
git reset --hard HEAD^ #回退至倒数第二个版本
    # HEAD当前
    # HEAD^上一个
    # HEAD^^，以此类推，过多的话HEAD~100
    # 直接给出所需版本的前7位ID，如"--hard aebb52a"
git reflog #命令历史：记录每一条命令
```

- 进行回退后想要返回新版本: `git reflog`获得新版本的commit ID 从而`git reset`返回新版本
- HEAD相当于一个指针，版本替换实际上是该指针指向不同版本的过程

- 回溯旧版本使用提交历史`git log`，新版本则为命令历史`git reflog`
- git 有一个暂存区，每次修改后将文件add到暂存区再commit到仓库
- git管理的是修改而非文件，若add后再进行修改，此次修改不在暂存区内，因此不会被提交

2.1.8. 撤销修改：

```
git checkout --file_name #将版本回退至最近一次git commit/git add的状态(相当于还未add时手动修改文档)
git reset HEAD file_name #将暂存区的修改撤销掉，重新放到工作区
```

- `git reset Head`适用于add到暂存区但还没commit的情况，HEAD表示最新的经过commit的文档
- 若已经commit了但还没有推送到远程repo，则可以使用版本回退到旧版本

2.1.9. 复制，移动与删除：

```
cp file_name directory_name #copy the file to this directory
git cp -r directory_name directory_name #copy the contents of directories
rm file_name #remove the file
git rm -r
git remote rm origin #删除远程仓库关联
    #若确认删除，在该命令后加git commit
    #若想回退，应用前面的git checkout --
mv #move
git mv file_name1 file_name2 #rename file_name1 to file_name2
```

2.1.10 打印与日期

```
echo whatever arguments #print
date #show you current time
```

2.2 github (remote repository)

2.2.1 创建ssh key

- 如果用户主目录里没有ssh目录，需要创建一个

```
ssh-keygen -t rsa -C "yunqiu_xu@163.com"(我的仓库是T450S)
github-> Account settings-> SSH Keys-> Add SSH Key -> Title 任意，文本框中添加id_rsa.pub(需要notepad++才能打开)
```

- create a new repo on the github:将本地仓库同远程仓库关联起来，从而实现推送

```
git remote add origin yoururl.git #link local dir to the remote dir
git push -u(第一次用push的时候加这个) origin master
#如果一开始显示仓库已存在的话，可以先用git remote rm origin删除仓库，然后再新建一个
git pull #将远程的更新拉到本地
```

- 将远程库克隆到本地

```
先在远程新建一个仓库
git clone sshurl
#如果克隆成功，在本地cd进这个仓库可以成功ls到这个仓库的内容
```

- http协议和ssh协议对比，后者更快且不需要口令

2.2.2 git branches: 创建指针

- 创建指针的原理：复习下之前在版本回退时的head指针过程，其中时间线只有一条（master主分支），Head指向master，master指向commit
- 当创建新分支时，git创建了一个新指针，head指向新指针，新指针指向和master相同的提交
- 相当于只是添加指针并改变head指向，而工作区文件不变
- 对工作区的修改和提交均针对新指针，而旧的master指针不变

2.2.3 合并指针: 将master指向新指针的当前提交

```
git checkout -b newpointer #创建并切换到新指针(等价于git branch +git checkout)
```

```
git branch #查看当前分支
#接下来可以在新分支上进行提交上传等动作，而切换回旧分支（git checkout oldpointer)则会发现提交点未变
git merge newpointer #此时在旧分支界面，使用merge将两个分支进行合并
git branch -d newpointer #合并结束后删除旧的分支
```

- 在分支上进行任务，完成后合并并删除，该过程比在主分支上进行更安全

2.2.4 解决冲突

- 创建新分支后，对旧分支和新分支都进行了改动提交，在这种情况下直接合并就会造成冲突
- 解决措施: 手动覆盖冲突（将修改后的文件提交上传），相当于多了一次提交

2.2.5 分支管理策略： 使用`--no-ff`参数可以保存合并历史（通过`git log`查看）

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

- 如果不使用这个参数，代表默认fast forward模式，看不出来分支曾经做过合并

2.2.6 bug分支：创建一个bug分支进行修复，然后与主分支合并，删除bug分支

```
git stash #储存工作现场
创建bug分支并修复bug，合并
git stash stash #查看储存的工作现场
git stash pop #恢复工作现场并删除这条储存记录（相当于git stash apply+git stash drop）
```

2.2.7 feature 分支：添加新功能时，新建分支进行开发调试，最后与主分支合并

```
git branch -D <name> #强行删除未被合并过的分支(一般情况下为 -d)
```

2.2.8 多人协作模式：

```
git remote -v #查看远程库信息
#本地新建分支不推送到remote repo，他人不可见
git push origin branch-name #将本地分支推送到远程
#如果远程分支有修改，需要先解决冲突—使用git pull将远程分支的修改拉到本地，合并后再推送上去
#git checkout -b branch-name origin/branch-name #在本地创建和远程相对应的分支，最好名字一致
git pull #抓取远程分支
```

2.2.9 git 标签管理

- 创建tag：先切换到需要打标签的branch，再添加标签

```
git tag tag_name #add the tag to this branch
git tag #查看标签
#默认标签打在最新的commit上，如果想打在以前的commit，可以先git log得到commit id，然后git tag tag_name commitID
git show <tagname> #显示该标签的分支信息
#标签按字母顺序而非时间顺序列出
#指定标签信息 -a标签名，-m标签说明
```

- 针对tag的操作：

```
git tag -d tag_name #删除标签
git push origin tag_name #将某个标签推送到远程仓库
    git push origin --tags #一次性推送所有标签
```

- 删除远程标签: 先删除本地标签，再运行`git push origin :refs/tags/tag_name`

2.2.10 参与github项目

- 进入他人项目主页-点fork就可以在自己的账号下clone一个repo
- 因为权限问题，无法直接推送修改到对方仓库，只能推送到自己的远程仓库
- 接着可以发起一个pull request请求对方接受你的修改文件