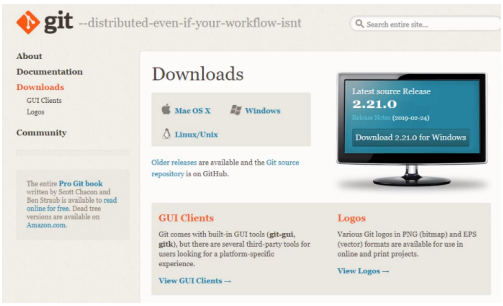
Git与GitHub笔记

- **Git基础**
  - 版本管理
    - Git是什么

      Git是一个版本管理控制系统（缩写VCS），它可以在任何时间点，将文档的状态作为更新记录保存起来，也可以在任何时间点，将更新记录恢复回来。
  - Git安装

    http://git-scm.com/downloads
    安装过程使用默认设置即可
    - git --version

      查看当前安装版本
  - Git基本工作流程

    - git仓库

      用于存放提交记录
    - git暂存区

      临时存放被修改文件（开发者从工作目录提交修改的文件到暂存区）
    - 工作目录

      被git管理的项目目录
  - Git的使用
    - Git使用前配置

      在使用git前，需要告诉git你是谁，在向git仓库中提交时需要用到。
      - 配置提交人姓名

        git config --global user.name 姓名
      - 配置提交人邮箱

        git config --global user.email 邮箱

- 查看git配置信息

  git config --list

- 其他修改途径

  - C盘用户文件夹下当前用户名文件夹中.gitconfig，记事本打开直接修改

- 注意

  * 修改配置信息只需重复上述操作
  * 配置只需执行一次

- 提交步骤

  - git init（初始化git仓库）

    在新建文件夹下会新建一个.git隐藏文件夹

  - git status（查看文件状态）
  - git add 文件列表（追踪文件）

    将想要被git管理的文件添加到暂存区中

  - git commit -m 提交信息（向仓库中提交代码）

    每次只包含一个功能，或者一个功能中包含bug的修改，便于恢复更改

  - git log（查看提交记录）

- 撤销

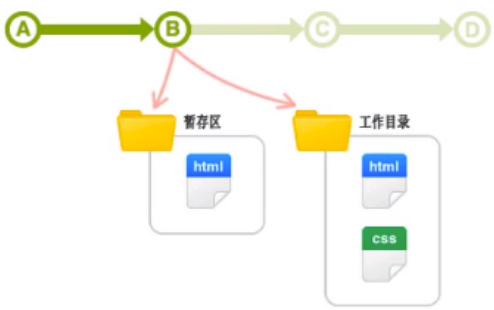
  - 用暂存区中的文件覆盖工作目录中的文件：git checkout 文件名

    暂存区依然存在这个文件并被git管理

  - 将文件从暂存区中删除：git rm --cached 文件名

    不小心添加到暂存区的文件，命令执行完成后暂存区没有这个文件，也不被git所管理，但是工作目录中依然存在这个文件。

  - 将git仓库中指定的更新记录恢复出来，并且覆盖暂存区和工作目录： git reset --hard commitID

    使用场景：工作目录中代码存在问题，git仓库中的一些提交记录也存在问题，希望将更早的提交记录恢复出来，并且删除存在问题的后面所有的提交记录。

- **Git进阶**

  - 分支

    相当于工作目录中代码的一份副本，在不同分支上做不同的事情。

- 分支细节
  - 主分支（master）

    第一次向git仓库中提交更新记录时自动产生一个分支。
    要保持最大程度的稳定性
  - 开发分支（develop）

    作为开发的分支，基于master分支创建。
    开发分支功能累积到一定程度以后再合并到主分支
    也要保持最大程度的稳定性
  - 功能分支（feature）

    作为开发具体功能的分支，基于开发分支创建。
    功能分支完成后将功能分支中的代码合并到开发分支，这时功能分支就可以删除了。
- 分支命令
  - git branch（查看分支）
    - git branch -a（查看本地和远程的所有分支）
    - git branch -r（查看远程分支）
  - git branch 分支名称（创建分支）

    在哪个分支操作就是基于哪个分支创建副本
  - git checkout 分支名称（切换分支）

    在切换分支之前，当前分区工作目录的文件一定要提交到git仓库中，保持当前分支工作区完全干净。
  - git merge 来源分支（合并分支）

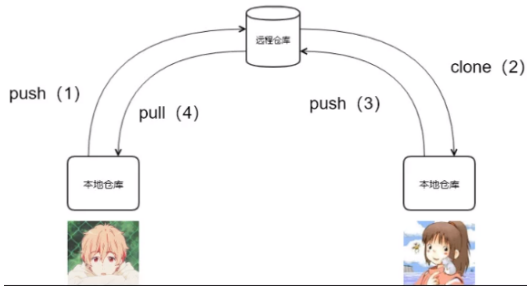
    当前分支的工作已经完成，需要合并到其他分支，原来的分支依然存在，可以回去继续开发。
    如果要把develop合并到master分支，先切换到master分支，然后输入合并命令。
  - git branch -d 分支名称 （删除分支）（分支被合并后才允许删除，-D强制删除）

    某一分支工作哦已经完成，且不需要继续存在。
    先切换到其他分支，才能删除当前分支。
    如果要删除的分支不想合并，用D强制删除
- 暂时保存更改

  在git中可以暂时提取分支上所有的改动并存储，让开发人员得到一个干净的工作副本，临时转向其他分支工作。
  当前工作尚未完成，又不想提交，可以用先暂存，然后转向其他分支。
  - 使用场景：分支临时切换
  - 存储临时改动：**git stash**
  - 恢复改动：**git stash pop**
- Github

在版本控制中，大约90%的操作都是在本地仓库进行的：暂存，提交，查看状态或历史记录等等。除此之外，如果仅仅你一个人在这个项目里面工作，没必要设置一个远程仓库。
只有当你需要和你的开发团队共享数据时，设置一个远程仓库才有意义。类似一个“文件管理服务器”，利用这个服务器可以与开发团队的其他成员进行数据交换。

- 注册
- 多人协作开发流程



  - A在自己计算机中创建本地仓库

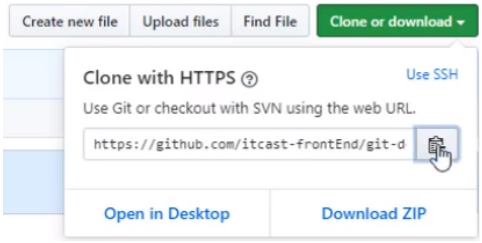
    git init，git status，git add，git commit -m
  - A在github中创建远程仓库

    https地址和ssh地址
  - A将本地仓库推送到远程仓库
    - git push 远程仓库地址 分支名称
    - git push 远程仓库地址别名 分支名称
    - git remote add 远程仓库地址别名 远程仓库地址（为远程仓库创建别名）

      eg. git push origin master
      - 用户名和密码存储在控制面板-凭据管理器-windows凭据中
    - git push -u 远程仓库地址别名 分支名称

      记住远程仓库别名和分支名称，下次提交只需要git push即可。
      - git push -u origin master
  - B克隆远程仓库到本地进行开发

    把本地仓库push到github上面
    - git clone 仓库地址

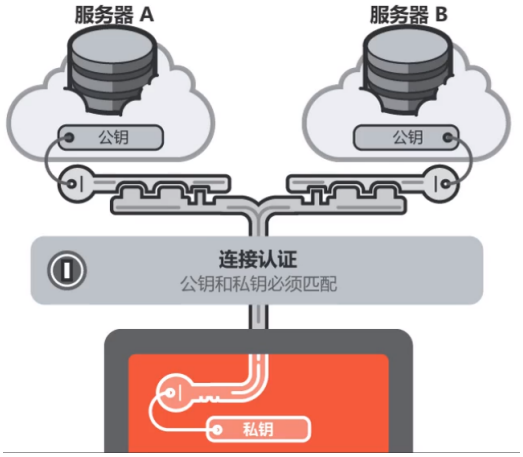
      

      不需要身份验证
  - B将本地仓库中开发的内容推送到远程仓库
    - cd 文件夹名称（切换到文件夹下的子文件夹）

      ```
      mongoose@DESKTOP-RO96G8U MINGW64 /c/course/AliBaiXiu/day01/code/B
      $ cd git-demo
      ```

        - GitHub settings collaborators添加合作者
    - A将远程仓库的**最新**内容拉取到本地
        - git pull 远程仓库地址 分支名称
        - git pull -all（获取所有远程分支）
    - 注意
        - git clone 只在第一次加入开发时使用
        - 如果本地版本低于远程仓库，不能直接向远程仓库推送，要把远程仓库先pull到本地，然后再推送
- 解决冲突
- GitHub跨团队协作
    - Folk仓库

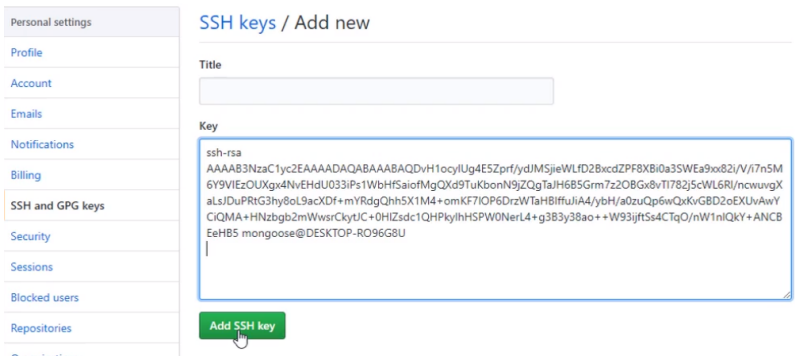
    - git clone本地进行修改
    - 仓库推送到远程
    - 发起pull request
    - 原仓库作者审核
    - 原仓库作者merge代码
- SSH免登录



SSH协议通过验证公钥和私钥是否匹配决定验证是否通过

    - 生成密钥：ssh-keygen

        命令行输入，一路回车
    - 秘钥存储目录：C:\Users\用户\.ssh
    - 公钥名称：id_rsa.pub



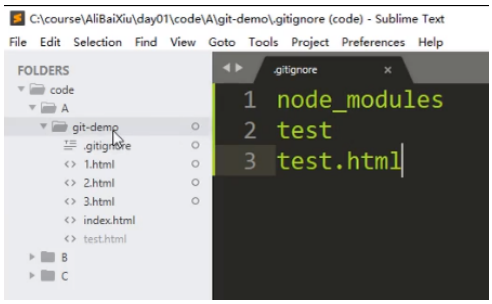
需要放在github服务器中，打开文件全选复制，头像settings-SSH and GPG keys

- 私钥名称：id_rsa

- GIT忽略清单

将不需要被git管理的文件名字添加到此文件中，在执行git命令的时候，git就会忽略这些文件。

- git忽略清单文件名称：.gitignore



- 将工作目录中的文件全部添加到暂存区：git add .

- 为仓库添加详细的说明

- 新建文件README.md
- 提交到本地仓库，然后推送到远程

- 代码阅读软件