# Git基础概念

1. 什么是Git

Git是一个开源的分布式版本控制系统。

1. Git的特性
   1. 直接快照记录，而非差异比较；
   2. 几乎所有操作都是本地执行。
2. Git的三个区域

工作区、暂存区和Git仓库。

1. Git中的三种状态

已修改modified、已暂存staged、已提交committed

1. Git的基本工作流程
   1. 在工作区中修改文件；
   2. 将你想要下次提交的更改进行暂存；
   3. 提交更新，找到暂存区的文件，将快照永久性存储到Git仓库。

# 安装并配置Git

1. 配置用户信息

右键选择git bash here

git config –global user.name “xxx”

git config –global user.email “xxx” //使用–global配置后，即可永久生效

1. Git的全局配置文件

C:\Users\xifeng\.gitconfig文件中可查看已完成的全局性配置

1. 检查配置信息

git config –list –global //查看所有全局配置项

git config user.name //查看指定全局配置项

1. 获取帮助信息

git help config //浏览器中打开git config命令的使用手册

git config -h //终端中打开git config命令的快速参考

# Git的基本操作

1. 获取Git仓库的两种方式
   1. 将尚未进行版本控制的本地目录装换位Git仓库；
   2. 从其他服务器克隆一个已存在的Git仓库。
2. 在现有目录中初始化仓库

实现步骤：①在项目目录中，右键点击Git Bash Here；②执行git init命令将当前目录转换为Git仓库。

git init命令会床架一个名为.git的隐藏目录，即当前项目的Git仓库，里面包含初始必要的文件。

1. 工作区中文件的4种状态

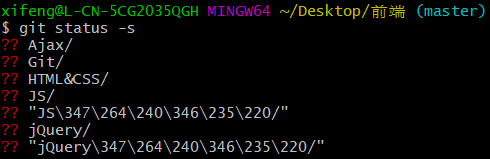
4种状态被分为2类：①未被Git管理；②已被Git管理。

未被Git管理：①未跟踪；

已被Git管理：①未修改，工作区中文件的内容与Git仓库中文件的内容一致；②已修改，工作区中文件的内容与Git仓库中文件的内容不一致；③已暂存，工作区中已修改的文件被放到暂存区。

1. 检查文件的状态

git status/git status -s //查看文件状态，此命令输出的结果叫做状态报告

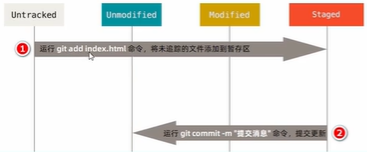


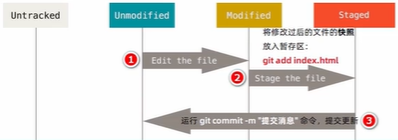
1. 跟踪新文件/将已修改的文件放入暂存区/把有冲突的文件标记为已解决状态

git add 文件名/git add . //将一个或多个新文件加入暂存区

1. 提交更新

git commit [-m “对提交内容的描述”] //将暂存区中的内容提交至Git仓库





1. 撤销对文件的修改

将工作区中已修改的文件撤回至Git仓库中所保存的版本。

git checkout –文件名

1. 将文件移出暂存区

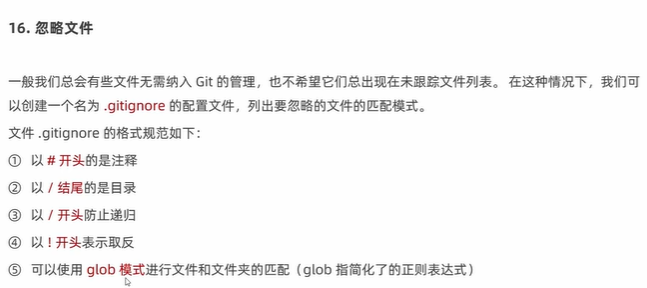
git reset HEAD 文件名/git reset . //将一个或全部文件移出暂存区

1. 跳过使用暂存区

git commit -a -m “对提交内容的描述”

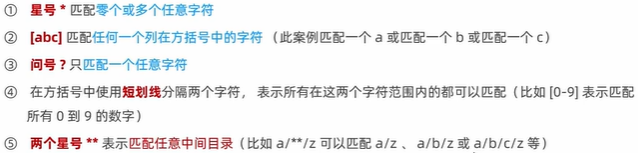
1. 移除文件
   1. git rm -f 文件名 //从Git仓库和工作区中同时移除指定文件
   2. git rm –cached 文件名 //只从Git仓库中移除指定文件，但保留工作区中的此文件
2. 忽略文件

创建名为.gitignore的配置文件，文件中的格式规范



1. glob模式

glob模式指简化的正则表达式



1. 查看提交历史

git log //按时间先后顺序列出所有提交历史

git log -数字 //只展示最新几次的提交历史

git log -数字 –pretty=oneline //在一行上展示最近几次的提交历史，无数字则表示查看所有提交历史

git log -数字 –pretty==format:”%h | %an | %ar | %s” //在一行上展示最近几次的提交历史，并自定义输出格式

1. 回退到指定的版本

git log –pretty=oneline //查看所有提交历史，获得版本唯一标识

git relog –pretty=oneline //在旧版本中使用此命令，查看全部操作历史(回退后只能使用此命令查看全部历史操作)

git reset –hard 版本标识 //回退到指定版本

# Github

1. 什么时Github

Github是全球最大的开源托管平台，只支持Git作为唯一的版本控制工具。

1. 远程仓库的两种访问方式
   1. HTTP：零配置，但每次访问时都需输入Github账号和密码；
   2. SSH：需要进行格外的配置，但配置成功后访问不需要输入Github账号和密码。
2. 基于HTTP将本地仓库上传至Github



* 1. 使用终端命令创建README.md文档，并写入初始内容为#project\_fxr；
  2. 初始化本地Git仓库，并将文件的修改提交到本地Git仓库中；
  3. 将本地仓库与远程仓库进行关联，并把远程仓库命名为origin；
  4. 将本地仓库中的内容推送到远程的origin仓库中。



只有第一次向远程仓库中推送内容才使用git push -u origin main，第二次以后使用git push。

1. 配置SSH

4.1. 生成SSH key

* 1. 打开Git bash；
  2. 输入ssh-keygen -t rsa -b 4096 -C “邮箱地址”；
  3. 连续敲击三次回车，可在C:\Users\用户名文件夹\.ssh目录中生成id\_rsa和id\_rsa.pub文件。

4.2. 配置SSH key

1. 使用记事本打开id\_rsa.pub文件，复制内容；
2. 在浏览器中登录Github，点击头像→Setting→SSH and GPG Keys→New SSH Key；
3. 将id\_rsa.pub文件中的内容粘贴至Key对应的文本框中；
4. 在Title文本框中任意填写一个名称，来表示这个Key从何而来。

4.3. 检测Github的SSH Key是否配置成功

在git bash中输入ssh -T [git@github.com](mailto:git@github.com)命令，再输入yes回车。

1. 基于SSH将本地仓库上传至Github



1. 将远程仓库克隆至本地

在Git Bash中输入” git clone 远程仓库地址”。

# Git 分支

1. 本地分支操作
   1. main/master主分支

初始化本地Git仓库的时候，Git默认创建一个名为main/master的分支，即主分支。主分支的作用是用来保存和记录整个项目已完成的功能代码。

* 1. 功能分支

功能分支是指专门用来开发新功能的分支。

* 1. 查看分支列表

git branch //分支名前的\*表示当前所处分支

* 1. 创建新分支

git branch 新分支名称 //基于当前分支创建新分支，新分支代码与当前分支代码完全相同。创建完成后仍处于当前分支，不会切换至新分支

* 1. 切换分支

git checkout 分支名 //切换至指定分支

* 1. 分支的快速创建和切换

git checkout -b 新分支名称 //基于当前分支创建新分支并切换至新分支

* 1. 合并分支

git merge 分支名称 //将指定分支合并至主分支，执行此命令时应处于主分支

* 1. 删除分支

git branch -d 分支名称 //将指定分支删除，执行此命令时不能处于将要删除的分支上

* 1. 遇到冲突时的分支合并

合并时提示有冲突，需手动解决冲突后再次执行git add .和git commit -m “描述”命令。

1. 远程分支操作
   1. 将本地分支推送至远程仓库

git push -u 远程仓库名称 本地分支名称[:远程分支名称] //将本地分支推送至远程仓库，可给推送的本地分支起别名。只有第一次将本地分支推送至远程仓库需要带-u参数

* 1. 查看远程仓库中所有分支

git remote show 远程仓库名称

* 1. 跟踪分支

跟踪分支是指从远程仓库中把远程分支下载到本地仓库中。

git checkout 远程分支名称 //下载到本地仓库的本地分支名称与远程仓库中的远程分支名称一致

git checkout -b 本地分支名称 远程仓库名称/远程分支名称 //将下载到本地仓库的本地分支重命名

* 1. 拉取远程分支的最新代码

git pull //执行此命令时应处于需要拉取的分支上

* 1. 删除远程分支

git push 远程仓库名称 –delete 远程分支名称 //此命令只删除远程仓库中的远程分支，不影响本地分支