第一章 git基本用法

- git 简介（代码版本管理，多人协作开发）

  > git 是什么？
  >
  > Git是目前世界上最先进的分布式版本控制系统（没有之一）。
  >
  > 特点： 高端大气上档次

  - git 版本管理工具的作用

  > 大型项目需要多人协作
  >
  > git与linux是同一个作者
  >
  > 免费的网络git服务器 coding.net github.com
  >
  > 普通公司代码非开源，都有自己的git服务器

- git 的下载

  > 菜鸟教程 : http://www.runoob.com/git/git-workspace-index-repo.html
  >
  > 简易教程：http://www.bootcss.com/p/git-guide/
  >
  > https://git-scm.com/ // 官网下载

- git的安装

  > windows下安装步骤
  >
  > 1 use git from the windows command prompt // 选择默认项
  >
  > 2 checkout windows-style,commit unix-style line endings // 使用默认项
  >
  > 确认git安装成功： 命令提示符下输入git，提示信息为:
  >
  > 安装成功： 可选参数帮助信息;

- git的配置

  > 因为Git是分布式版本控制系统，所以，每个机器都必须自报家门：你的名字和Email地址
  >
  > 否则版本库不知道是哪位程序员进行的版本更新

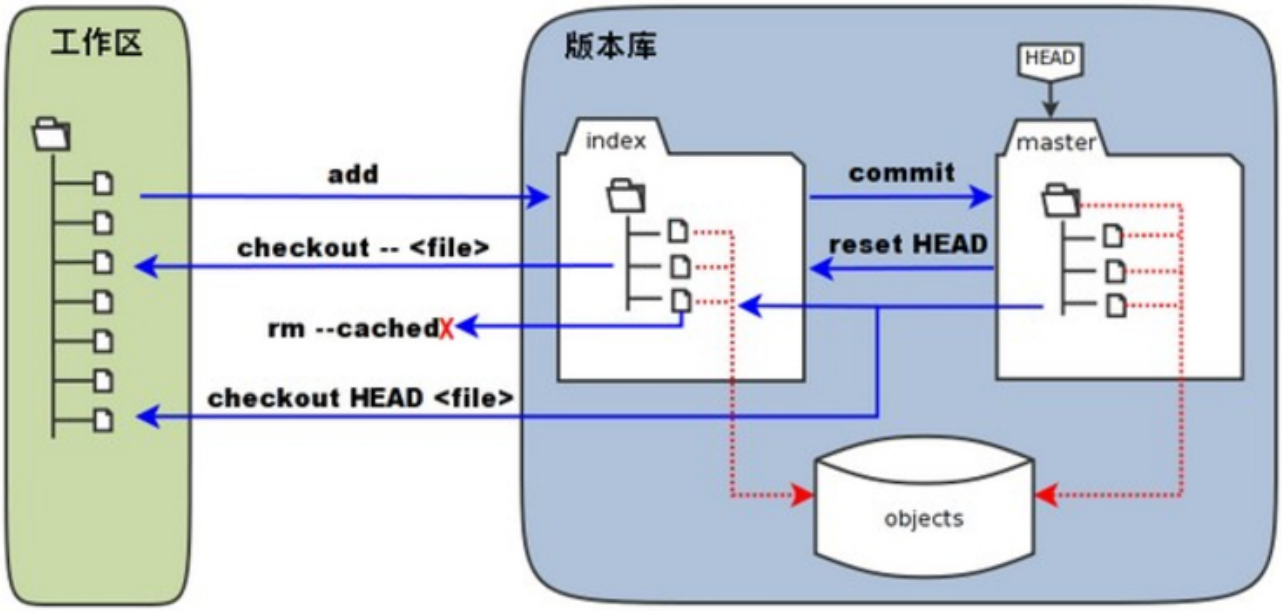
  ```
  $ git config --global user.name "Your Name"           // 配置用户名
  $ git config --global user.email "email@example.com"  // 配置邮箱
  git config --global  --list  // 查看当前git的配置信息
  ```

- 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，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”。
  >
  > d:/git-test> git init ； // 创建仓库;

- 工作区、暂存区、版本库的概念
- 工作区：就是你在电脑里能看到的目录。

- 暂存区：英文叫stage, 或index。一般存放在 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。
- 版本库：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库



- 添加文件到版本库

  - git add　　将工作区内容添加到暂存区

    git add 文件1 文件2 或 git add . 将指定文件或工作区所有文件添加到暂存区
  - git commit -m "版本说明信息" 将暂存区内容提交到版本库
  - git commit -am "commite content" 对已追踪的文件 add+commit合写;

- 查看仓库修改信息

```
git status                    新增了哪些文件，哪些文件被修改过;
git diff                      工作区域缓存区的差别
git diff --cached             缓存区与版本库的差别
git diff HEAD -- filename     工作区与版本库的差别
```

- 撤销文件的修改

```
git checkout -- filename
// 使用缓存区或版本库的文件替换工作区文件

git reset HEAD -- filename
//取消缓存的内容，
```

- 从仓库删除文件

```
git rm -- filename          // 将文件从工作区，缓存区，版本库中同时删除
git rm --cached filename;   // 将文件从缓存区与版本库删除，工作区保留
```

- 查看版本日志

```
git log  // 查看commit 的历史记录

退出 :q
```

- 版本回退与前进

```
回退：

git reset --hard HEAD~1    // 返回上一个版本

git reset --hard HEAD~2    //返回上上版本

git reset --hard HEAD~100  // 返回上100个版本

前进

git reset --hard commit_id  // 前进到 指定的commit_id

git reflog  // 记录我们的每一次命令，可以查找到每次commit的 commit_id
```

- 删除文件

```
删除文件 test.txt;

// 误删除  可以从版本库恢复
git checkout -- test.txt

// 确实想删除 将版本库中对应文件也删除

git add .
git commit -m "deleted test.txt"
```

- 常用的git命令

```
- git init    初始化本地仓库
- git remote add origin git@github.com:gamyys2/mygit.git  将本地仓库和远程仓库关联

- git checkout 文件名  将已缓存文件恢复到工作区

- git clone  git@github.com:gamyys2/mygit.git   // 将服务器仓库拷贝一份到本地文件夹
- git push origin master  将HEAD区内容提交到服务器主分支
- git pull origin master    从服务器主分支拷贝内容到本地

- git branch  dev  创建dev分支
- git checkout -b dev   创建并切换到dev分支
- git checkout  master  切换到主分支
- git pull origin master  确保当前为最新版本
- git merge dev    将dev分支merge到主分支
- git push origin master  将本地仓库push到服务器
```

第二章 远程仓库

> git 服务器实现多人协作：
>
> 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。
>
> 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。

- 注册github账号

```
【方案一】

// 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

//  第1步 创建 SSH key

// 将 C:\Program Files (x86)\Git\usr\bin 加入系统环境变量

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

 // 你需要把邮件地址换成你自己的邮件地址，然后一路回车
```

```
【方案二】
或者找到 C:\Program Files (x86)\Git\usr\bin 里的ssh-kegen.exe 双击执行
```

```
// 生成的文件位置:  C:\Users\gamyys\.ssh

// 主目录里找到.ssh目录，里面有id_rsa和id_rsa.pub两个文件

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

// 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
```

- 添加远程仓库

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。

- 关联远程仓库

```
git remote add origin git@github.com:gamyys9/demo.git   // origin 根源，起源，指远程仓库;
git remote // 查看当前的远程仓库名

git remote -v  // 查看详细远程仓库信息
git remote rm origin2  // 删除远程仓库origin2
```

- 将本地仓库master推送到github服务器

```
$ git push -u origin master  // 第一次推送时，增加 -u 将本地master与远程master关联，以后可以简化代码
$ git clone git@github.com:gamyys9/demo.git  //从远程仓库克隆
```

第三章 git分支管理

- 分支的作用

> 你想开发一个新功能，开发到50%立刻提交，由于功能不完整，会影响其他同事的开发，如果不提交，又担心丢掉已完成的代码，这样的话，可以创建一个新的分支，随时可以提交，不影响主分支，开发完成后，再合并到主分支;

- 分支的管理
  - 创建分支
  - 切换分支
  - 删除分支

```
// 1 创建并切换分支

$ git checkout -b dev

// 相当于

git branch dev;   // 创建分支

git checkout dev // 切换到dev分支

// 2 查看当前分支

git branch;

// 3 切换回主分支

git checkout master;

// 4 将dev分支合并到主分支

git merge dev;

// 5 删除dev分支

git branch -d dev;
```

- 分支的冲突

```
// 创建并切换到分支 friend
git checkout -b friend;
git add .
git commit -m "friend";

// 切换到主分支master
git checkout master
git add .

// 进行分支合并

git merge

// 会发现存在冲突

// 解决冲突并提交，重新合并，发现成功了;

// 如果当前分支没有完成，需要切换到其他分支，可以将当前分支状态保存才可切换

git stash

git stash list 查看储存的分支中间状态

// stash@{0} xxx

// 保存后就可以切换分支了

// 再次切换回当前分支，需要加载保存的状态

git stash apply stash@{0}  // 加载前面保存的状态
```

第四章 多人协作

> 你从远程仓库克隆时，实际上Git自动把本地的`master`分支和远程的`master`分支对应起来了，并且，远程仓库的默认名称是`origin`。

- 查看远程仓库

```
git remote  // origin

git remote -v // 查看远程仓库详细信息
```

- 推送分支

```
git pull origin master // 将本地的master分支推送到远程库;

git pull orgin dev     // 将本地的dev分支推送到远程库
```

第五章 标签管理

> 每个commit版本通过一串数字标识，不好记忆，可以给这串数字起个别名，方便版本管理，这个别名就是标签

- 打标签的方法

```
git tag  // 查看标签列表

git tag v1.0  commit_id  // 对指定的commit版本打标签
```

- 标签的管理

```
// 标签是保存在本地的，可以删除

$ git tag -d v1.0  // 删除标签

// 将本地的某个标签推送到远程

 $ git push origin v1.0

 // 将本地标签全部推送到远程

 $ git push origin --tags

 // 删除远程标签

 $ git tag -d v1.9 // 先从本地删除
 $ git push origin :refs/tags/v1.9  // 从远处删除

// 可以登录github查看是否删除
```

第六章 git其他问题

- 忽略文件

> 并不是所有文件都需要提交进行版本管理，需要忽略的文件可以记录到 .gitignore;
>
> 在 .gitignore 文件中列出需要忽略的文件，则这些文件不会被git忽略不纳入版本管理
>
> 但 .gitignore文件本身需要被提交到git