

第一章 git基础知识

第一节 git 简介

- 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/>

阮一峰教程：http://www.ruanyifeng.com/blog/2014/06/git_remote.html

<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

第二节 git基本操作

- 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 master     // 工作区与版本库的差别
git diff --cached   // 暂存区域版本库的差别
```

- 撤销文件修改

```
git reset -- filename // 将文件从版本库恢复到暂存区；
git checkout -- filename // 将文件从暂存区恢复到工作区；
git checkout HEAD -- filename // 将文件从版本库恢复到暂存区与工作区
```

- 从仓库删除文件

```
git rm -- filename // 将文件从工作区，缓存区中删除,版本库保留
git rm --cached filename; // 将文件从版本库删除，暂存区与工作区保留
```

- 查看版本日志

```
git log      // 查看commit 的历史记录,退出多屏模式(:q)
git reflog   // 简化版的git log ,可以查看commit_id;
```

- 版本切换

切换版本

```
git reset --hard commit_id  // 切换到 指定的commit_id
git reset --hard HEAD~1     // 返回上个版本(~2 返回上上版本)
```

第二章 git企业应用

第一节 github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,并且各自把自己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
- 远程仓库配置(加密问题)

// 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 (开始/所有程序/git/Git Bash)

```
$ ssh-keygen -t rsa -C "youremail@example.com"
// 邮件换成自己邮件,然后一路回车
// 中间需要输入5位短字符,记住后面要用;
```

[第2步] 拷贝文件

生成文件位置: 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3步] 粘贴内容

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 -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  
  
// ssh --> git@github.com:gamyys2/mygit2.git  
  
// 输入短密(生成秘钥时的短字符串)  
  
// https --> https://github.com/gamyys2/mygit2.git  
  
// 输入github的用户名和密码  
  
// 当前仓库与多个远程仓库关联, 当第一次push使用-u 可以设定默认远程仓库, 这样可以使用git push 向默认远程仓库  
推送;  
git push // 第二次会向默认的源推送主本地分支;
```

- 从远程仓库克隆

```
git clone git@github.com:gamyys9/demo.git //从远程仓库克隆(注意需要输入短密)
```

第二节 git企业实战

如果你是新人, 刚加入项目组, 需要先将大家完成的项目部分拷贝至你本地, 使用的就是远程仓库克隆。

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`。

- 克隆远程仓库

```
git clone git@github.com:gamyys2/test.git // 克隆仓库
```

- 查看远程仓库

```
git remote // origin  
  
git remote -v // 查看远程仓库详细信息
```

- 工作区修改

```
// 完成属于你的工作任务, 并提交;  
git add *  
git commit -m "my-work"
```

- 推送分支

```
// 第一次推送
git push -u origin master // 将本地master分支推送到远程;

// 第二次推送
git push
```

- 更新本地仓库

```
git pull // 从远程仓库更新内容到本地;
```

第三章 git分支管理

第一节 分支的基础

- 分支作用

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 branch dropdown; // 创建分支
```

```
git checkout dropdown// 切换到dev分支
```

```
// 创建+切换 git branch -b dev
```

```
// 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; // 注意当前分支为主分支;
```

```
// 5 删除dev分支
```

```
git branch -d dev;
```

```
// 分支切换配图
```

<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/001375840038939c291467cc7c747b1810aab2fb8863508000>

第二节 分支冲突

- 分支的冲突

```
// 创建分支 tom
git checkout -b tom;

// 修改工作区内容

git add .
git commit -m "tom";

// 切换到主分支master
git checkout master

git checkout -b peter;

// 修改工作区内容;
git add *
git commit -m peter

git merge tom

// 产生冲突
// 对于同一文件,同一位置,在不同分支进行不同修改 --- > 冲突

// 手动解决冲突 -- > 保留正确内容的过程;

// 提交
git commit -m "conflict fixed"

// 查看提交历史

git log --graph (显示分支历史图)
```

第三节 远程仓库

```
// 1 设置默认主机+分支;

git checkout master ; // 确保master为当前分支;

git push -u origin master // 将本地主分支推送到远程主分支并合并;

git pull 从远程仓库主分支更新内容到本地主分支

// 2 如果想推送本地的dev分支到远程;

git checkout dev // 设置dev为当前分支;

git push -u origin dev // 将本地dev分支推送到远程;

git pull 从远程的dev分支更新内容到当前dev分支;
```

第四节 git pages

```
//1 创建仓库,仓库名必须为:

gamyys.github.com

// 本地仓库初始化
index.html

git init

// 本地仓库提交
git add .
git commit -m "ok"

// 添加远程源
git remote add origin ***.git

// 推送
git push -u origin master

// 2 访问
https://gamyys2.github.io
```

第四章 git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
```

第五章 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