Linux基本命令

`cd` ： 改变目录
`cd ..` ： 回退到上一个目录，直接cd进入默认目录
`pwd` ： 显示当前目录路径
`ls(ll)` ： 都是列出当前目录中的所有文件，只不过ll列出的内容更为详细
`touch` ： 新建一个文件 如touch index.js就会在当前目录下新建一个index.js文件
`rm` ： 删除一个文件，rm index.js就会把index.js文件删除
`mkdir` ： 新建一个目录，就是新建一个文件夹
`rm -r` ： 删除一个文件夹，rm -r src删除src目录
`mv` ： 移动文件，mv index.html src index.html是我们要移动的文件，src是目标文件夹
`reset` ： 重新初始化终端/清屏
`clear` ： 清屏
`history` ： 查看历史
`help` ： 帮助
`exit` ： 退出
`#` 表示注释

git

查看配置

```
git config -l
git config --system --list
git config --global --list
```

环境配置

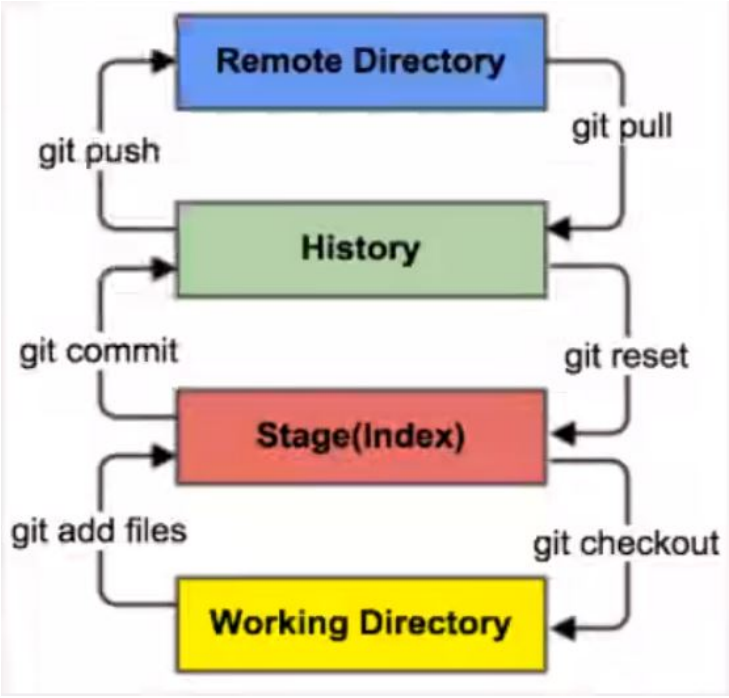
设置用户名 必须设置

```
git config --global user.name ""
git config --global user.email ""
```

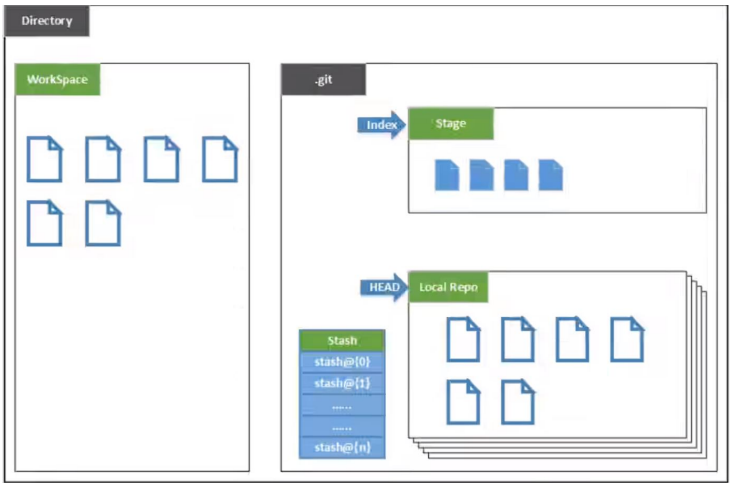
git基本理论

工作区域

git本地有三个工作区域：工作目录(Working Directory)、暂存区(Stage/Index)、资源库(Repository/Git Directory)。
如果在加上远程的git仓库(Remote Directory)就可以分为四个工作区域。



- Workspace： 工作区，就是你平时存放项目代码的地方
- Index/Stage： 暂存区，用于临时放你的改动，事实上它只是一个文件，保存即将提交到文件列表信息
- Repository： 仓库区(或本地仓库)，就是安全存放数据的位置，这里面有你提交到所有版本的数据。
  其中HEAD指向最新放入仓库的版本
- Remote： 远程仓库，托管代码的服务器，可以简单的认为是你项目中的一台电脑用于远程交换数据

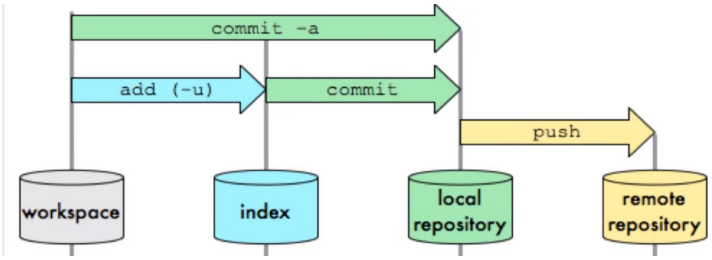


- Directory： 使用Git管理的一个目录，也就是一个仓库，包含我们的工作空间和Git的管理空间
- Workspace： 需要通过Git进行版本控制的目录和文件，这些目录和文件组成了工作空间。
- .git： 存放Git管理信息的目录，初始化仓库的时候自动创建。
- Index/Stage： 暂存区，或者叫待提交跟新区，在提交进入repo之前，我们可以把所有的更新放在暂存区。
- Local Repo ： 本地仓库，一个存放在本地的版本库Head会指向当前的开发分支(branch)
- Stash ： 隐藏，是一个工作状态保存栈，用于保存/恢复Workspace中的临时状态

工作流程

1. 在工作目录中添加，修改文件；
2. 将需要进行版本管理的文件放入暂存区； `git add .`
3. 将暂存区的文件提交到git仓库。 `git commit`

因此，git管理的文件有三种状态： 已修改(modified)、已暂存(staged)、已提交(committed)



Git项目搭建

创建本地仓库的方法有两种： 一种是创建全新的仓库，另一种是克隆远程仓库。

1. 创建全新的仓库，需要用Git管理的项目的根目录执行

   ```
   git init      # 在当前目录新建一个git代码库
   ```

2. 执行后可以看到，仅仅在项目目录多出了一个.git目录，关于版本等的所有信息都在这个目录里面

克隆远程仓库

1. 另一种方式是克隆远程目录，将远程服务器上的仓库完全镜像一份至本地

   ```
   git clone [url]      # 克隆一个项目和它的整个代码历史(版本信息)
   ```

Git文件操作

```
git status [filename]      # 查看指定文件状态

git status      # 查看所有文件状态

git add .      # 添加所有文件到暂存区

git commit -m "消息内容"      # 提交暂存区中的内容到本地仓库 -m 提交信息
```

忽略文件 .gitignore

在主目录下建立".gitignore"文件，此文件有如下规则：

1. 忽略文件中的空行或以井号（#）开始的行将会被忽略。
2. 可以使用Linux通配符。例如：星号（*）代表任意多个字符，问号（？）代表一个字符，方括号（[abc]）代表可选字符范围，大括号（{string1,string2,...}）代表可选的字符串等。
3. 如果名称的最前面有一个感叹号（!），表示例外规则，将不被忽略。
4. 如果名称的最前面是一个路径分隔符（/），表示要忽略的文件在此目录下，而子目录中的文件不忽略。
5. 如果名称的最后面是一个路径分隔符（/），表示要忽略的是此目录下该名称的子目录，而非文件（默认文件或目录都忽略）。

```
#为注释
*.txt        #忽略所有 .txt结尾的文件,这样的话上传就不会被选中!
!lib.txt     #但lib.txt除外
/temp        #仅忽略项目根目录下的TODO文件,不包括其它目录temp
build/       #忽略build/目录下的所有文件
doc/*.txt    #会忽略 doc/notes.txt 但不包括 doc/server/arch.txt
```

使用GitHub

1. 注册GitHub
2. 设置本机绑定SSH公钥，实现免密码登录
3. 将公钥信息public key添加到github账户中即可

   ```
   ssh-keygen -t rsa      # -t rsa加密生成
   ```

4. 在github中创建一个自己的仓库

Git中常用指令

```
git branch      # 列出所有本地分支

git branch -r      # 列出所有远程分支

git branch [branch-name]      # 新建一个分支，但依然停留在当前分支

git chechout -b [branch]      # 新建一个分支，并切换到该分支

git merge [branch]      # 合并指定分支到当前分支

git branch -d [branch-name]      # 删除分支

git push origin --delete [branch]      # 删除远程分支

git branch -dr [remote/branch]      # 删除远程分支
```

git fetch + merge:获取最新代码到本地，然后手动合并分支

```
git remote -v      # 查询当前远程的版本
```

//获取最新代码到本地(本地当前分支为[branch],获取的远端的分支为[origin/branch])

```
git fetch origin master      # 获取远端的[origin/master]分支

git fetch origin dev      # 获取远端的[origin/dev]分支
```

//查看版本差异

```
git log -p master..origin/master      # 查看本地master与远端origin/master的版本差异

git log -p dev..origin/dev      # 查看本地与远端origin/dev的版本差异
```

//合并最新代码到本地分支

```
git merge origin/master      # 合并远程端分支origin/master到当前分支
`git merge origin/dev      # 合并远端分支origin/dev到当前分支
```