1.Linux常用命令

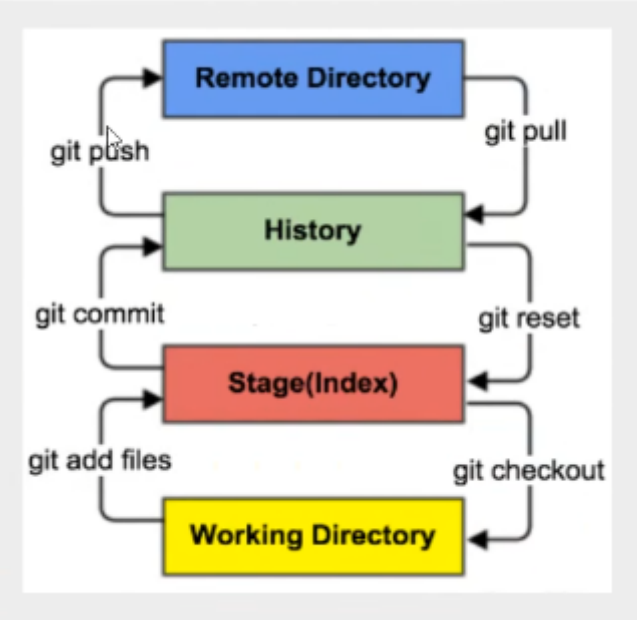
1）、cd：改变目录。
2）、cd . . 回退到上一个目录，直接cd进入默认目录
3）、pwd：显示当前所在的目录路径。
4）、ls(ll)：都是列出当前目录中的所有文件，只不过ll(两个ll)列出的内容更为详细。
5）、touch：新建一个文件 如 touch index.js 就会在当前目录下新建一个index.js文件。
6）、rm：删除一个文件, rm index.js 就会把index.js文件删除。
7）、mkdir：新建一个目录,就是新建一个文件夹。
8）、rm -r：删除一个文件夹, rm -r src 删除src目录
9）、mv 移动文件, mv index.html src index.html 是我们要移动的文件, src 是目标文件夹,当然, 这样写,必
夹在同一目录下。
10）、reset 重新初始化终端/清屏。
11）、clear 清屏。
12）、history 查看命令历史。
13）、help 帮助。
14）、exit 退出。
15）、#表示注释

vim中yy是复制，p是粘贴

2. Git基本理论：(所有的实践基于此)

2.1 工作区域

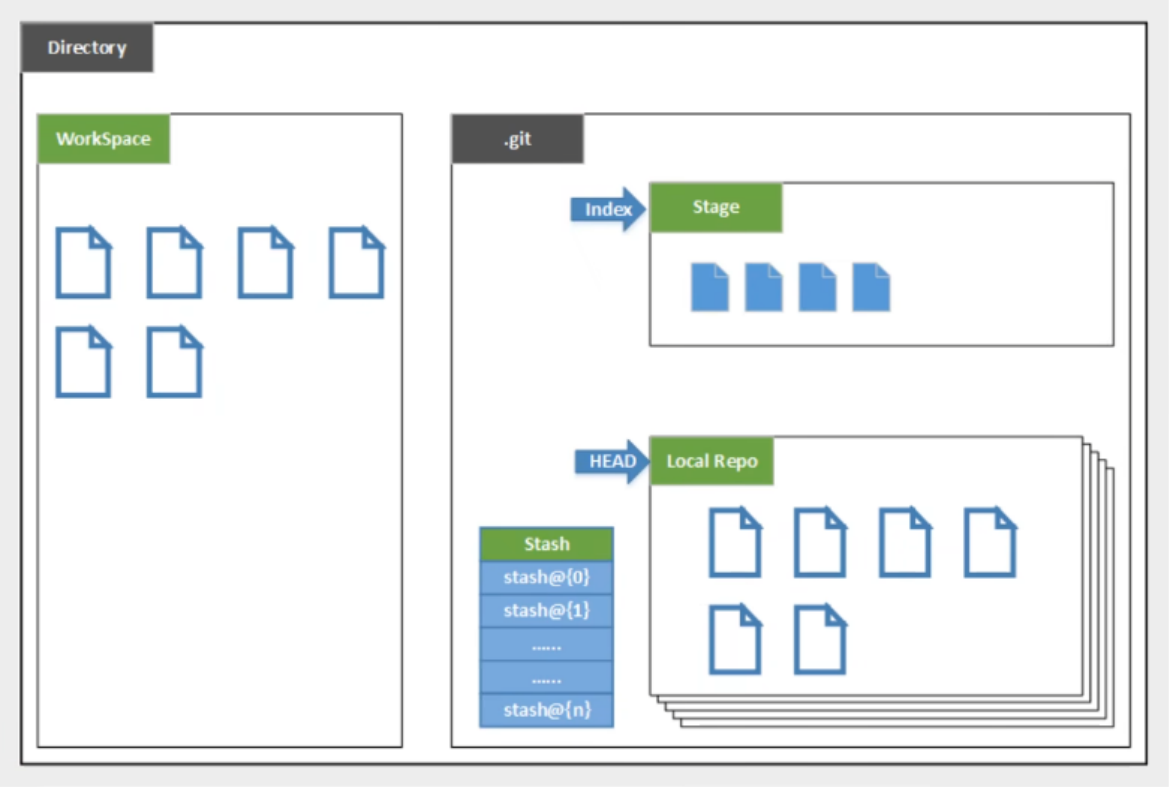
Git有三个工作区域，工作目录(Working Directory)、暂存区（Stage/Index）、资源库（Repository或Git Directory）。如果加在远程的Git仓库（Remove Directory）就可以分为四个工作区域。文件在四个工作区域的



- Workspace：工作区，就是你平时存放项目代码的地方
- Index / Stage：暂存区，用于临时存放你的改动，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组中的一台电脑用于远程数据交换

本地的三个区域确切的说应该是git仓库中HEAD指向的版本：

3. Git基本命令



> 工作流程

git的工作流程一般是这样的：

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域；

3、将暂存区域的文件提交到git仓库。

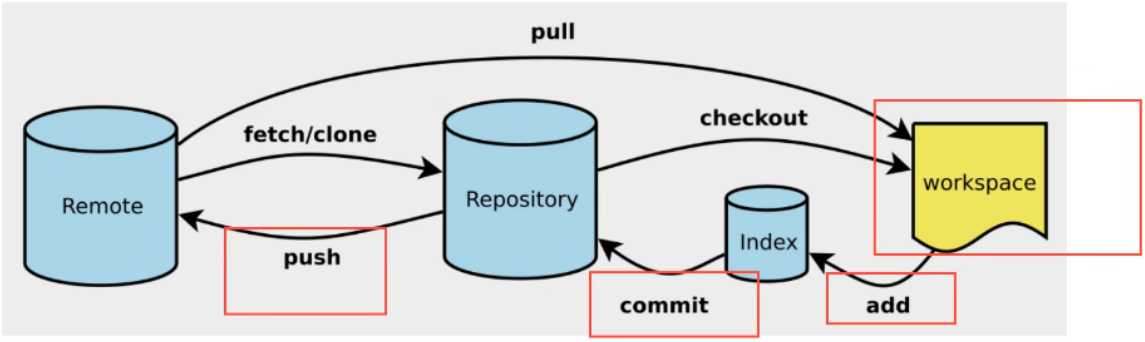
因此，git管理的文件有三种状态：已修改（modified），已暂存（staged），已提交(committed)

3.1 重点记住的命令

创建工作目录与常用指令

工作目录（WorkSpace)一般就是你希望Git帮助你管理的文件夹，可以是你项目的目录，也可以是一个空目录，建议不要有中文。

日常使用只要记住下图6个命令：



git config --global user.name 用户名	设置用户签名
git config --global user.email 邮箱	设置用户签名
git init	初始化本地库
git status	查看本地库状态
git add 文件名	添加到暂存区
git commit -m "日志信息" 文件名	添加到本地库
git reflog	查看历史记录
git reset --hard 版本号	版本穿梭

3.2 创建本地仓库

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：

```
# 在当前目录新建一个Git代码库
$ git init
```

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，由于是将远程服务器上的仓库完全镜像一份至本地！

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

2、去 gitee 或者 github 上克隆一个测试！

4. Git文件操作

> 文件4种状态

版本控制就是对文件的版本控制，要对文件进行修改、提交等操作，首先要知道文件当前在什么状态，不然可能会提交了现在还不想提交的文件，或者要提交的文件没提交上。

- **Untracked**: 未跟踪, 此文件在文件夹中, 但并没有加入到git库, 不参与版本控制. 通过`git add` 状态变为`Staged`.
- **Unmodify**: 文件已经入库, 未修改, 即版本库中的文件快照内容与文件夹中完全一致. 这种类型的文件有两种去处, 如果它被修改, 而变为`Modified`. 如果使用`git rm`移出版本库, 则成为`Untracked`文件
- **Modified**: 文件已修改, 仅仅是修改, 并没有进行其他的操作. 这个文件也有两个去处, 通过`git add`可进入暂存`staged`状态, 使用`git checkout` 则丢弃修改过, 返回到`unmodify`状态, 这个`git checkout`即从库中取出文件, 覆盖当前修改！
- **Staged**: 暂存状态. 执行`git commit`则将修改同步到库中, 这时库中的文件和本地文件又变为一致, 文件为`Unmodify`状态. 执行`git reset HEAD filename`取消暂存, 文件状态为`Modified`

> 上面说的4种状态，通过如下命令可以查看文件状态

4.1 git具体的操作流程

1. `git init`:管理本地的目录应该先进行初始化

```
Initialized empty Git repository in D:/develop/Git/gitcode/gitdemo/.git/
```

`ll -a`:可以查看隐藏文件，`git status [filename]`查看文件状态，`cat`查看文件中的内容

2. `git add test.txt`：添加到暂存区，git追踪文件的过程，暂存区的文件可以删掉，使用命令`git rm --cached`
3. `git commit -m "first commit" test.txt`:提交到本地库,显示如下信息：

```
Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ git commit -m "first commit" test.txt
[master (root-commit) bdc53c3] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

4. 修改版本：（展示版本的迭代）

   再次查看状态，git status,显示如下信息：红色的表示文件还没添加到暂存区，需要进行追踪，然后再添加到本地库

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

`git reflog`可以查看版本日志，如下图所示，有两个版本，指针指向第二个版本

```
$ git reflog
07ec979 (HEAD -> master) HEAD@{0}: commit: second commit
bdc53c3 HEAD@{1}: commit (initial): first commit
```

5. 版本穿梭：可以查看历史版本。命令`git reset --hard 版本号`，可以回到任意版本，调整的是master指针的位置

```
#查看指定文件状态
git status [filename]
#查看所有文件夹状态
git add .   #添加所有文件到暂存区   git add helloworld.txt
git commit -m "版本信息"   #提交暂存区中的内容到本地仓库   -m 是提交信息
git commit -m "third commit" helloworld.txt  #提交不同的版本信息到本地库
vim helloworld.txt   #修改版本，按i进入编辑模式，按ESC退出编辑模式之后，按shift+: 输入wq进行保存
cat helloworld.txt  #查看当前文件的内容
git reflog  #查看提交的版本信息
```

5. Git分支操作

5.1 查看分支

1）基本语法：

git branch -v

2)案例实操

```
Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ git branch -v
* master 07ec979 second commit
```

5.2 创建分支

git branch 分支名

案例实操：

```
$ git branch hot-fix

Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ git branch -v
  hot-fix 07ec979 second commit
* master  07ec979 second commit

Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$
```

5.3 切换分支

基本语法：

git checkout 分支名

案例实操：切换成功后，再进行文件的修改提交

```
Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ git checkout hot-fix
Switched to branch 'hot-fix'

Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (hot-fix)
$ vim test.txt

Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (hot-fix)
$ git status
On branch hot-fix
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test.txt

no changes added to commit (use "git add" and/or "git commit -a")

Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (hot-fix)
$ git add test.txt
```

5.4 合并分支

git merge 分支名：此分支名合并到当前分支中

```
Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ git merge hot-fix
Updating 07ec979..5080201
Fast-forward
 test.txt | 1 +
 1 file changed, 1 insertion(+)
```

```
Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ cat test.txt
hello world! 222222222
hello world! 222222222
hello world! 222222222
hello world! 333333333
```

冲突合并：合并分支时，两个分支在同一个文件的同一个位置有两套完全不同的修改，git无法决定我们使用哪一个，必须人为决定新代码内容。

我们将master和hot-fix 分支的文件都进行了修改之后，再进行合并之后会出现如下冲突：

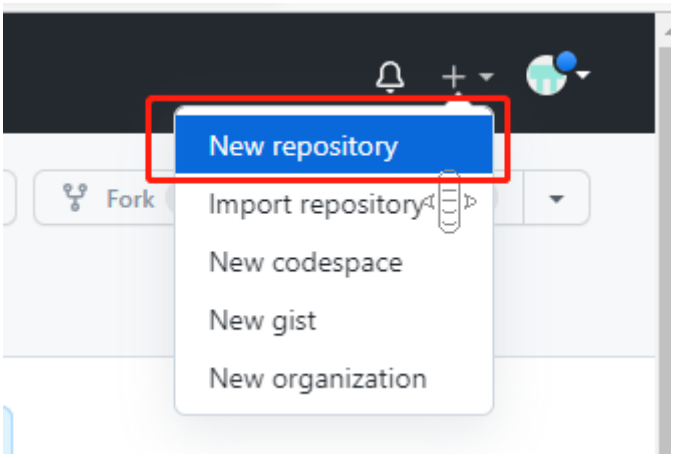
```
Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ git merge hot-fix
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
```

此时需要手动合并，然后再进行

6 Github创操作

6.1 创建远程仓库并创建别名

在自己的GitHub账号中点击如下图示，可以新建一个远程库，远程库的名字和自己本地仓库的名字尽量一致



创建别名：由于这个仓库的名字是一段网址，太长了，需要创建一个别名

`git remote add 别名 网址`

```
Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ git remote add git-demo https://github.com/mengkai0218/gitdemo.git

Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ git remote -v
git-demo        https://github.com/mengkai0218/gitdemo.git (fetch)
git-demo        https://github.com/mengkai0218/gitdemo.git (push)
```

6.2 推送本地分支到远程仓库

1）基本语法

git push 别名 分支

2）案例实操

推送成功之后会显示如下信息：

```
Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ git push git-demo master
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 12 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (18/18), 1.28 KiB | 1.28 MiB/s, done.
Total 18 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), done.
To https://github.com/mengkai0218/gitdemo.git
 * [new branch]      master -> master
```

6.3 GitHub拉取本地库到远程库

1）基本语法

git pull git-demo(别名) master（分支）

2）案例实操

拉取成功，显示如下

```
Administrator@WIN-VR4RNMK6AOP MINGW64 /d/develop/Git/gitcode/gitdemo (master)
$ git pull git-demo master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 646 bytes | 7.00 KiB/s, done.
From https://github.com/mengkai0218/gitdemo
 * branch            master     -> FETCH_HEAD
   95b154e..4b00d07  master     -> git-demo/master
Updating 95b154e..4b00d07
Fast-forward
 test.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

可以在远程库中对代码进行修改，再提交，会自动更新本地库的状态

6.4 github克隆代码到本地库

先搜索凭据管理器，将之前的GitHub管理器删除掉，然后输入指令

`git clone https://github.com/mengkai0218/gitdemo.git`，就可以将代码克隆到本地

这个地址为远程仓库地址，克隆结果为:初始化本地仓库

小结：clone会做如下操作。1.拉取代码。2.初始化本地库。3.创建别名