Linux ubuntu

第一章

1.1 shell

1.2 command

1. 参数 命令执行的目标

```
# 输出目录中文件列表
ls /data
# 可多个参数
ls /data1 /data2
```

2. 选项 用于扩展或修改命令行为方式

```
# 提供文件其他信息(默认为字节大小显式文件大小)
ls -1
# 以kb大小显示文件
ls -lh
# 默认为正序，按照相反字母顺序打印
ls -r
# 结合使用
ls -lr
```

3 历史

向上箭头显式上一个命令，重复按显示历史命令，enter执行

```
history     查看历史记录
！+ 数字     执行对应命令
```

1.3 变量

1.3.1 局部变量

local仅存于当前的shell中，不影响其他变量和应用程序

```
variable1 = 'something'
echo $variable1
#echo + 美元符号+变量名 显示变量的值
```

1.3.2 环境变量（全局）

```
env       显示所有环境变量
env | grep  variable    文本筛选，将输出传递给grep，并由其进行筛选
export    将局部变量转换为全局
export variable2      //variable2被转换为全局变量
unset variable2       //exported 变量可以被删除、普通变量也可
```

环境变量赋值

```
va1='some'
va2='thing'
va3=$va1''$va2          //va3 = some thing
va4=$va1$va2             //va4 = something
```

1.4 命令

```
type env
```

type 查看命令的相关信息（来源等）

1.4.1 内部命令

内部命令在shell本身中

1.4.2 外部命令

which命令来显示相关命令的完整路径:

```
which ls
```

which命令通过搜索PATH变量来搜索命令的位置。

```
type -a echo
输出：// echo is a shell builtin                          // echo is /bin/echo
```

1.4.3 alias 别名

```
alias查看系统给的别名
```

```
也可以自己创建别名
alias name=command
但仅存在于shell打开时，如果shell关闭，则全部消失
```

1.4.4 Functions 函数

```
function_name () {
commands
}
执行一串命令
```

1.5 引号

linux中用一些引号来提醒系统

1.5.1 双引号

双引号"**阻止 shell 解释一些元字符**（特殊字符），包括全局字符。

> 全局字符，也称为通配符，是对 shell 具有特殊含义的符号； 在尝试运行任何命令之前，它们由 shell 本身解释。 全局字符包括星号（*） 字符，问题 （?） 标记字符和方括号 [ ] 等。

但双引号中仍然允许一些**变量替换**和**命令替换**

```
echo "the PATH is $path"
```

如 $path 仍会被解释。

1.5.2 单引号

单引号阻止 shell 对特殊字符进行任何解释，包括 glob、变量、命令替换和其他尚未讨论的元字符。

1.5.3 反斜杠 \

可以用来防止shell解释单个字符

```
echo a is $path //path将被显示为一堆目录
echo b is \$path //$path将不会被解释
```

1.5.4 反引号 `

```
echo today is date
// todat is data
echo today is `data`
//today is Mon Nov 4 03:40:04 UTC 2018
```

1.6 控制语句

1.6.1 分号字符

```
command1; command2; command3
```

连续执行命令，无论上个命令是否执行成功，都将继续运行

1.6.2 双与（&&）号

```
command1 && command2
```

第一个命令成功则执行第二个，否则不执行第二个

1.6.3 双||号

逻辑or

```
command1 || command2
```

第一个命令没有成功则执行第二个

常用的Linux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

```
rm -rf / 切勿在Linux中尝试！删除电脑中全部文件！
```

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,必须保证文件和目标文件夹在同一目录下。

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

关于端口

查看已经连接的服务端口（ESTABLISHED）
netstat -a

查看所有的服务端口（LISTEN，ESTABLISHED）
netstat -ap

查看指定端口，可以结合grep命令：
netstat -ap | grep 8080

也可以使用lsof命令：
lsof -i:8888

若要关闭使用这个端口的程序，使用kill + 对应的pid
kill -9 PID号

> ps：kill就是给某个进程id发送了一个信号。默认发送的信号是SIGTERM，而kill -9发送的信号是SIGKILL，即exit。exit信号不会被系统阻塞，所以kill -9能顺利杀掉进程。

Git命令

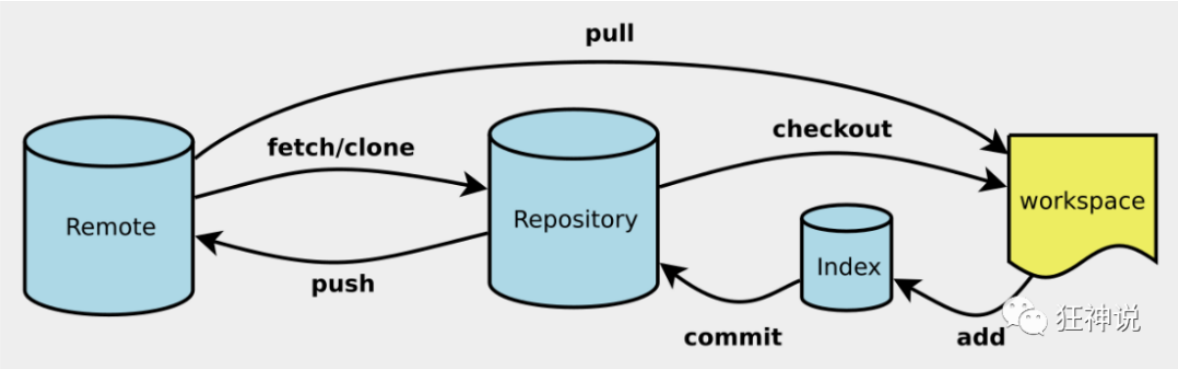
1 git配置

- git config -l　　查看git配置
- git config --system --list 查看系统配置

- git config --global --list　查看本地配置

2 git项目搭建

2.1 创建工作目录与常用指令



2.2 本地仓库搭建

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：

```
# 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。

2.3 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]
# https://gitee.com/kuangstudy/openclass.git
```

3 Git文件操作

文件的四种状态

- 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

查看文件状态

上面说文件有4种状态，通过如下命令可以查看到文件的状态：

```
#查看指定文件状态
git status [filename]

#查看所有文件状态
git status

# 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

有些时候我们不想把某些文件纳入版本控制中，比如数据库文件，临时文件，设计文件等

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
```

Git 入门

1 Git安装以及配置

2 创建Git版本库

Git版本库又名仓库，英文名repository。您可以把版本库简单的理解成一个目录，Git可以管理目录中的所有文件，并跟踪每个文件的修改、删除，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”。

1. 使用**git init**创建本地仓库。
2. 使用git clone克隆远程仓库。

   使用git clone从现有Git仓库中拷贝项目（类似svn checkout）。

   克隆仓库的命令格式如下。

   ```
   git clone <repo> <directory>
   ```

   参数说明：

- repo：Git仓库地址。
- directory：本地目录。

3 创建文件并提交到版本库

Git本地仓库有三大区域：工作区、暂存区、版本区。创建、修改文件等操作是在工作区，git add命令是将工作区的内容添加到暂存区，git commit命令将临时存区内容添加到本地仓库中。

3.1 git add

添加一个或多个文件到暂存区。

```
git add [file1][file2]...
```

添加指定目录到暂存区，包括子目录。

```
git add [dir]
```

添加当前目录下的所有文件到暂存区。

```
git add .
```

3.2 git commit

提交暂存区到本地仓库中。

```
git commit -m 'message'
```

参数说明：

- -m：添加-m参数表示可以直接输入后面的message，如果不加-m参数，表示不直接输入message的，而是会调用一个编辑器来让您输入这个message。
- message：备注信息。

3. 提交暂存区的指定文件到仓库区。

```
git commit [file1] [file2] ... -m 'message'
```

4 Git版本回退

git reset命令用于回退版本，可以指定退回某一次提交的版本。git reset命令语法如下。

```
git reset [--soft | --mixed | --hard] [HEAD]
```

参数说明：

- --mixed：默认，可以不用带该参数。用于重置暂存区的文件，使文件与上一次的提交(commit)保持一致，工作区文件内容保持不变。
- --soft：用于回退到某个版本。
- --hard：参数用于撤销工作区中所有未提交的修改内容，将暂存区与工作区都回到上一次版本，并删除之前的所有信息提交。

- HEAD：

  HEAD和HEAD~0表示当前版本。

  HEAD^和HEAD~1表示上一个版本。

  HEAD^^和HEAD^2表示上上一个版本。

  HEAD^^^和HEAD^3表示上上上一个版本。

5 Git的文件比较

git diff命令用于比较文件的不同，即比较文件在暂存区和工作区的差异。git diff命令显示已写入暂存区和已经被修改但尚未写入暂存区文件的区别。git diff的语法如下。

查看尚未缓存的改动。

```
git diff
```

查看已缓存的改动。

```
git diff --cached
```

查看已缓存的与未缓存的所有改动。

```
git diff HEAD
```

显示摘要，而非整个diff。

```
git diff --stat
```

6 Git的撤销修改

Git的撤销修改有以下两种方式。

1. 文件修改后还没有被放到暂存区，撤销修改就能回到版本库未修改前状态。

   ```
   cat 文件名
   #查看文件内容
   ```

   ```
   git checkout Aliyun.txt
   # 撤销工作区对文件的修改
   ```

2. 文件修改后添加到暂存区，撤销暂存区的修改就回到添加到暂存区后的状态，需要继续执行撤销文件操作，才能完全撤销成功。

   在暂存区取消对Aliyun.txt文件暂存区的更改。

   ```
   git reset HEAD Aliyun.txt
   ```

   执行如下命令，撤销文件。

   ```
   git checkout Aliyun.txt
   ```

Git分支

1. 查看分支。

执行如下命令，查看分支。

```
git branch
```

2. 创建分支。

执行如下命令，创建分支。

```
git branch test
```

3. 切换分支。

a. 执行如下命令，切换分支。

```
git checkout test
```

b. 执行如下命令，查看分支是否已经切换。

```
git branch
```

返回结果如下，您可以看到已经成功切换到test分支。

4. 创建并切换分支。

a. 执行如下命令，您可以实现创建分支并切换分支。

```
git checkout -b test2
```

b. 执行如下命令，查看分支是否已经切换。

```
git branch
```

返回结果如下，您可以看到已经成功切换到test2分支。

5. 删除分支。

a. 执行如下命令，删除分支。

注意:如果您要删除的分支是当前工作分支，必须切换到其他分支后，才能删除。

```
git branch -d test
```

执行如下命令，查看分支是否已经切换。

```
git branch
```

6. 合并分支。

a. 执行如下命令，查看当前所处分支。

```
git branch
```

返回结果如下。

b. 执行如下命令，在当前分支test2下新建test2.txt文件

```
vim test2.txt
```

进入Vim编辑器后，按下i键进入编辑模式，添加以下内容，添加完成后按下Esc键退出编辑模式，最后输入**:wq**后按下Enter键保存并退出Vim编辑器。

```
who are you?
```

添加后的文件内容如下所示。

c. 执行如下命令，将文件提交到暂存区。

```
git add test2.txt
```

d. 执行如下命令，将暂存区内容提交到本地仓库。

```
git commit -m 'test2分支新文件'
```

返结果如下，您可看到test2.txt已经成功提交到本地仓库。

e. 执行如下命令，切换到master分支。

```
git checkout master
```

g. 执行如下命令，将test2分支合并到当前master分支。

```
git merge test2
```

返回结果如下，您可以看到分支合并成功。