

Git

Git概述

版本系统

SVN 是集中式版本控制系统，版本库是集中放在中央服务器的，而开发人员工作的时候，用的都是自己的电脑，所以首先要从中央服务器下载最新的版本，然后开发，开发完后，需要把自己开发的代码提交到中央服务器。

集中式版本控制工具缺点：服务器单点故障、容错性差

Git 是分布式版本控制系统（Distributed Version Control System，简称 DVCS），分为两种类型的仓库：

本地仓库和远程仓库：

- 本地仓库：是在开发人员自己电脑上的 Git 仓库
- 远程仓库：是在远程服务器上的 Git 仓库

工作流程

1. 从远程仓库中克隆代码到本地仓库
2. 从本地仓库中 checkout 代码然后进行代码修改
3. 在提交前先将代码提交到**暂存区**
4. 提交到本地仓库。本地仓库中保存修改的各个历史版本
5. 修改完成后，需要和团队成员共享代码时，将代码 push 到远程仓库

Git安装

下载地址：<https://git-scm.com/download>

代码托管

Git 中存在两种类型的仓库，即本地仓库和远程仓库。那么我们如何搭建Git远程仓库呢？我们可以借助互联网上提供的一些代码托管服务来实现，其中比较常用的有 GitHub、码云、GitLab 等。

GitHub（地址：<https://github.com/>）是一个面向开源及私有软件项目的托管平台，因为只支持 Git 作为唯一的版本库格式进行托管，故名 GitHub

码云（地址：<https://gitee.com/>）是国内的一个代码托管平台，由于服务器在国内，所以相比于 GitHub，码云速度会更快

GitLab (地址: <https://about.gitlab.com/>) 是一个用于仓库管理系统的开源项目, 使用 Git 作为代码管理工具, 并在此基础上搭建起来的 web 服务

环境配置

安装 Git 后首先要设置用户名称和 email 地址, 因为每次 Git 提交都会使用该用户信息, 此信息和注册的代码托管平台的信息无关

设置用户信息:

- `git config --global user.name "Seazean"`
- `git config --global user.email "imseazean@gmail.com"` //用户名和邮箱可以随意填写, 不会校对

查看配置信息:

- `git config --list`
- `git config user.name`

通过上面的命令设置的信息会保存在用户目录下 `/.gitconfig` 文件中

本地仓库

获取仓库

- **本地仓库初始化**

1. 在电脑的任意位置创建一个空目录 (例如 `repo1`) 作为本地 Git 仓库
2. 进入这个目录中, 点击右键打开 Git bash 窗口
3. 执行命令 **`git init`**

如果在当前目录中看到 `.git` 文件夹 (此文件夹为隐藏文件夹) 则说明 Git 仓库创建成功

- **远程仓库克隆**

通过 Git 提供的命令从远程仓库进行克隆, 将远程仓库克隆到本地

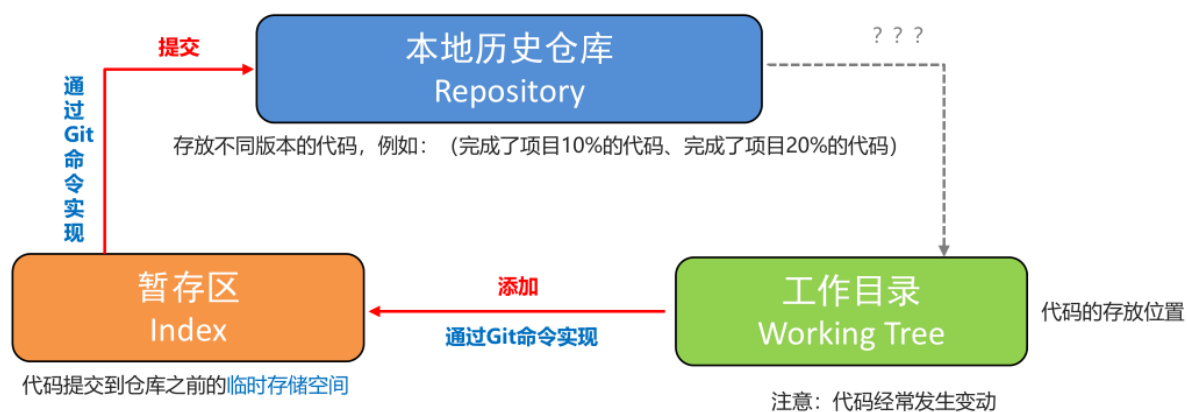
命令: `git clone 远程 Git 仓库地址 (HTTPS 或者 SSH)`

- **生成 SSH 公钥步骤**

- 设置账户
- `cd ~/.ssh` (查看是否生成过 SSH 公钥) `user` 目录下
- 生成 SSH 公钥: `ssh-keygen -t rsa -C "email"`
 - `-t` 指定密钥类型, 默认是 `rsa`, 可以省略
 - `-C` 设置注释文字, 比如邮箱
 - `-f` 指定密钥文件存储文件名
- 查看命令: `cat ~/.ssh/id_rsa.pub`

- 公钥测试命令：ssh -T git@github.com

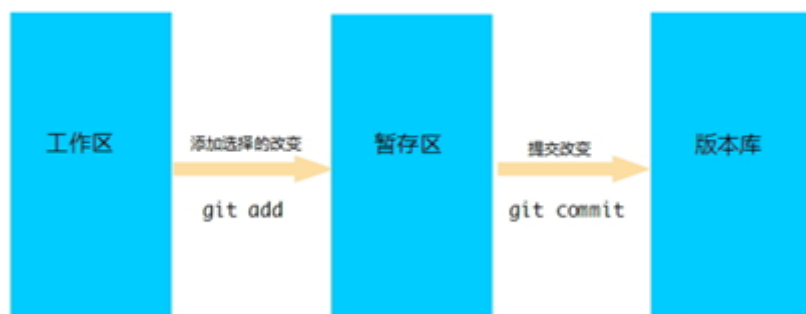
工作过程



版本库：.git 隐藏文件夹就是版本库，版本库中存储了很多配置信息、日志信息和文件版本信息等

工作目录（工作区）：包含 .git 文件夹的目录就是工作目录，主要用于存放开发的代码

暂存区：.git 文件夹中有很多文件，其中有一个 index 文件就是暂存区，也可以叫做 stage，暂存区是一个临时保存修改文件的地方



文件操作

常用命令

命令	作用
git status	查看 git 状态（文件是否进行了添加、提交操作）
git add filename	添加，将指定文件添加到暂存区
git commit -m 'message'	提交，将暂存区文件提交到本地仓库，删除暂存区的该文件
git commit --amend	修改 commit 的 message
git rm filename	删除，删除工作区的文件，不是仓库，需要提交
git mv filename	移动或重命名工作区文件
git reset filename	使用当前分支上的修改覆盖暂存区， 将暂存区的文件取消暂存
git checkout filename	使用暂存区的修改覆盖工作目录，用来撤销本次修改(危险)
git log	查看日志（git 提交的历史日志）
git reflog	可以查看所有分支的所有操作记录（包括已经被删除的 commit 记录的操作）

其他指令：可以跳过暂存区域直接从分支中取出修改，或者直接提交修改到分支中

- git commit -a 直接把所有文件的修改添加到暂存区然后执行提交
- git checkout HEAD -- files 取出最后一次修改，可以用来进行回滚操作

文件状态

- Git 工作目录下的文件存在两种状态：
 - untracked 未跟踪（未被纳入版本控制）
 - tracked 已跟踪（被纳入版本控制）
 - Unmodified 未修改状态
 - Modified 已修改状态
 - Staged 已暂存状态
 - 查看文件状态：文件的状态会随着我们执行 Git 的命令发生变化
 - git status 查看文件状态
 - git status -s 查看更简洁的文件状态
-

文件忽略

一般我们总会有些文件无需纳入Git 的管理，也不希望它们总出现在未跟踪文件列表。通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。在这种情况下，我们可以在工作目录中创建一个名为 .gitignore 的文件（文件名称固定），列出要忽略的文件模式。下面是一个示例：

```
# no .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# only ignore the TODO file in the current directory, not subdir/TODO
/TODO
# ignore all files in the build/ directory
build/
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .pdf files in the doc/ directory
doc/**/*.pdf
```

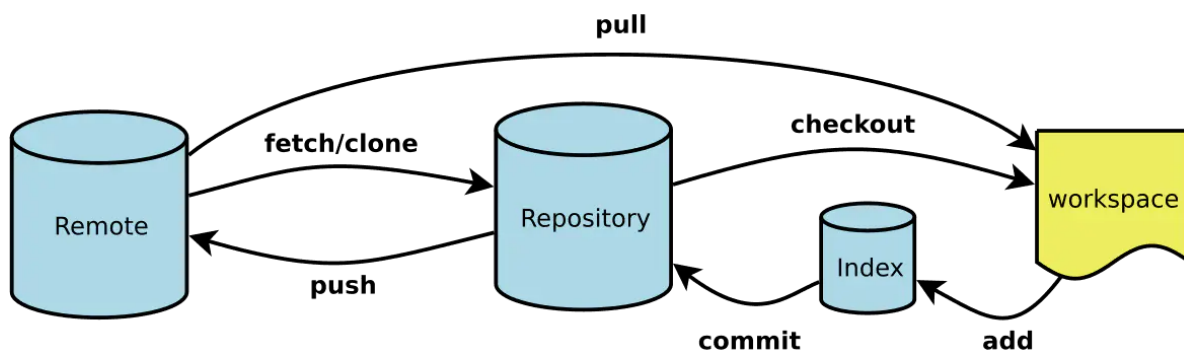
远程仓库

工作流程

Git 有四个工作空间的概念，分别为 工作空间、暂存区、本地仓库、远程仓库。

pull = fetch + merge

fetch 是从远程仓库更新到本地仓库，pull是从远程仓库直接更新到工作空间中



查看仓库

git remote：显示所有远程仓库的简写

git remote -v：显示所有远程仓库

git remote show：显示某个远程仓库的详细信息

添加仓库

git remote add：添加一个新的远程仓库，并指定一个可以引用的简写

克隆仓库

git clone (HTTPS or SSH)：克隆远程仓库

Git 克隆的是该 Git 仓库服务器上的几乎所有数据（包括日志信息、历史记录等），而不仅仅是复制工作所需要的文件，当你执行 git clone 命令的时候，默认配置下远程 Git 仓库中的每一个文件的每一个版本都将被拉取下来。

删除仓库

git remote rm：移除远程仓库，从本地移除远程仓库的记录，并不会影响到远程仓库

拉取仓库

git fetch：从远程仓库获取最新版本到本地仓库，不会自动 merge

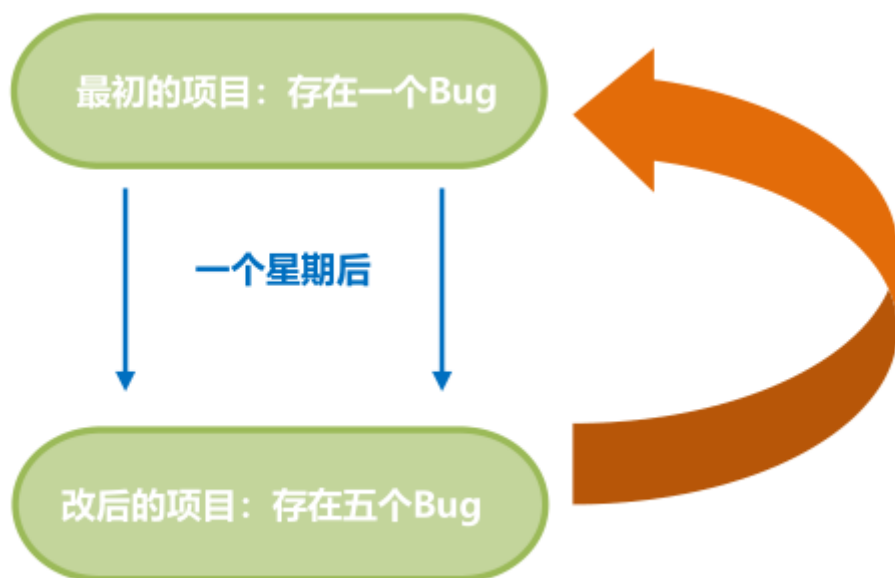
git pull：从远程仓库获取最新版本并 merge 到本地仓库

注意：如果当前本地仓库不是从远程仓库克隆，而是本地创建的仓库，并且**仓库中存在文件**，此时再从远程仓库拉取文件的时候会报错（fatal: refusing to merge unrelated histories），解决此问题可以在 git pull 命令后加入参数 --allow-unrelated-histories

推送仓库

git push：上传本地指定分支到远程仓库

版本管理



命令：git reset --hard 版本唯一索引值

分支管理

查看分支

git branch：列出所有本地分支

git branch -r：列出所有远程分支

git branch -a：列出所有本地分支和远程分支

创建分支

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

git checkout -b branch-name：新建一个分支，并切换到该分支

推送分支

git push origin branch-name：推送到远程仓库，origin 是引用名

切换分支

git checkout branch-name: 切换到 branch-name 分支

合并分支

git merge branch-name: 合并指定分支到当前分支

有时候合并操作不会如此顺利。如果你在两个不同的分支中，对同一个文件的同一个部分进行了不同的修改，Git 就没办法合并它们，同时会提示文件冲突。此时需要我们打开冲突的文件并修复冲突内容，最后执行 git add 命令来标识冲突已解决

```
zhaq@zhaq MINGW64 /d/gitRepos/repo2 (master)
$ git merge b3
Auto-merging UserMapper.xml
CONFLICT (content): Merge conflict in UserMapper.xml
Automatic merge failed; fix conflicts and then commit the result.
```

删除分支

git branch -d branch-name: 删除分支

git push origin -d branch-name: 删除远程仓库中的分支 (origin 是引用名)

如果要删除的分支中进行了开发动作，此时执行删除命令并不会删除分支，如果坚持要删除此分支，可以将命令中的 -d 参数改为 -D: git branch -D branch-name

标签管理

查看标签

git tag: 列出所有 tag

git show tag-name: 查看 tag 详细信息

标签作用: 在开发的一些关键时期，使用标签来记录这些关键时刻，保存快照，例如发布版本、有重大修改、升级的时候、会使用标签记录这些时刻，来永久标记项目中的关键历史时刻

新建标签

git tag tag-name: 新建标签，如 (git tag v1.0.1)

推送标签

git push [remotename] [tagname]: 推送到远程仓库

git push [remotename] --tags: 推送所有的标签

切换标签

git checkout tag-name: 切换标签

删除标签

git tag -d tag-name: 删除本地标签

git push origin :refs/tags/ tag-name: 删除远程标签

IDEA操作

环境配置

File → Settings 打开设置窗口, 找到 Version Control 下的 git 选项

选择 git 的安装目录后可以点击 Test 按钮测试是否正确配置: D:\Program Files\Git\cmd\git.exe

创建仓库

1、VCS → Import into Version Control → Create Git Repository

2、选择工程所在的目录,这样就创建好本地仓库了

3、点击git后边的对勾,将当前项目代码提交到本地仓库

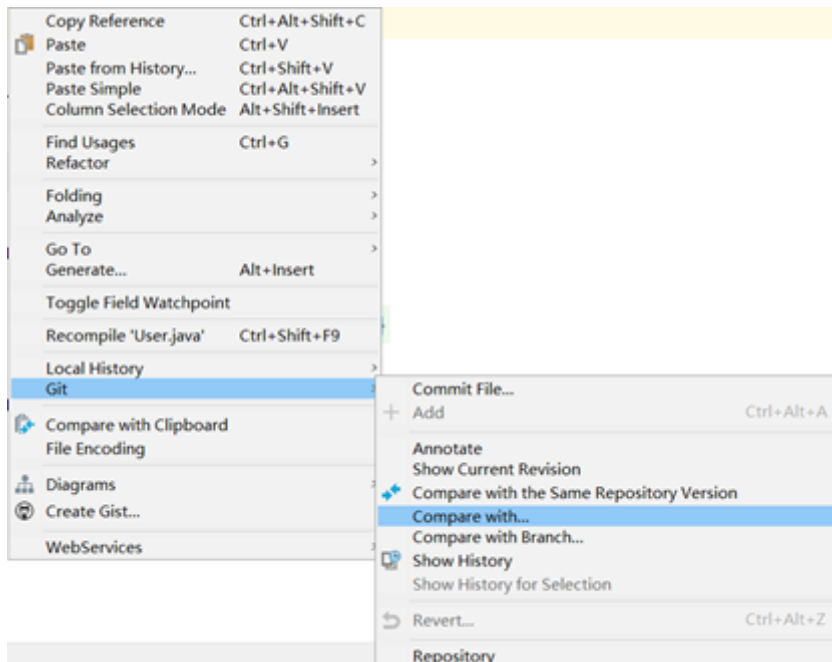
注意: 项目中的配置文件不需要提交到本地仓库中,提交时,忽略掉即可

文件操作

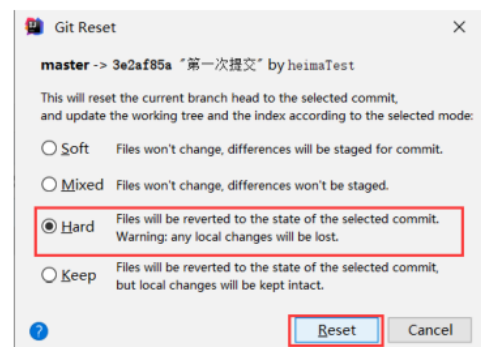
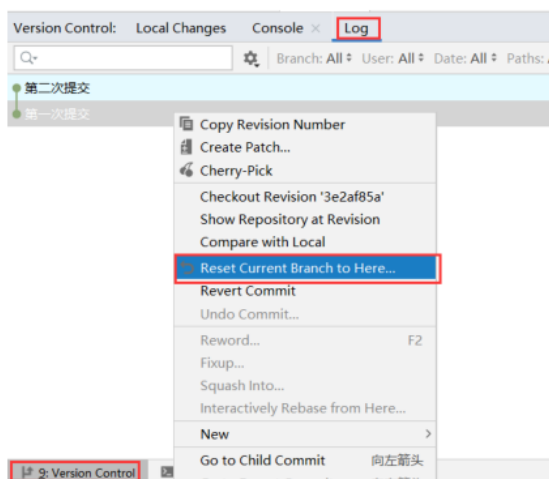
右键项目名打开菜单 Git → Add → commit

版本管理

- 版本对比

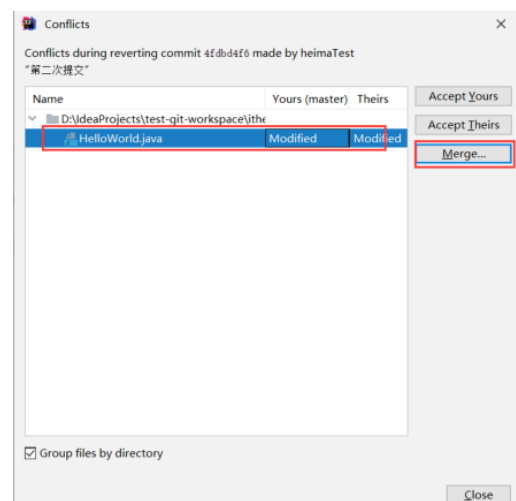
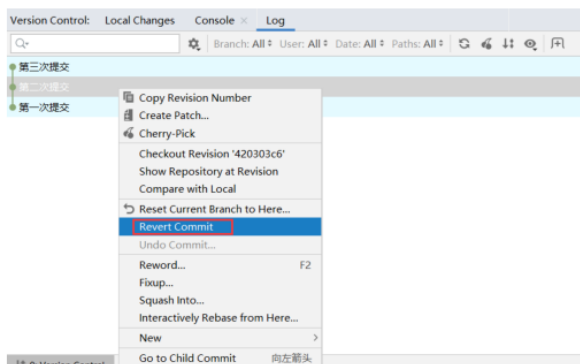


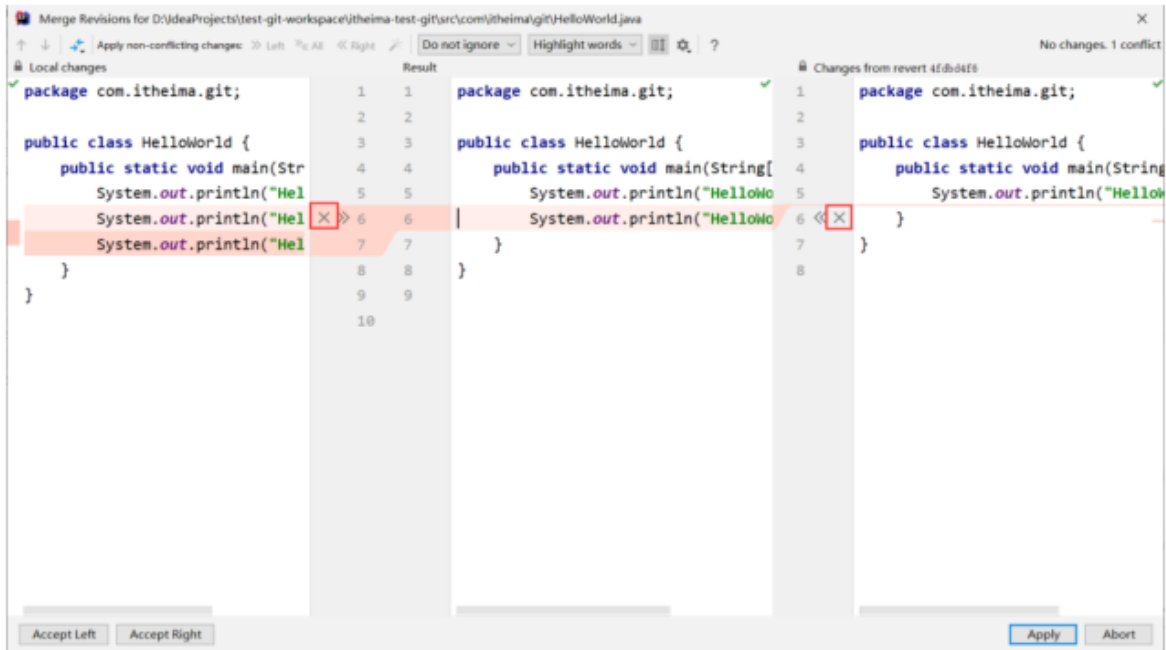
- 版本切换方式一：控制台 Version Control → Log → 右键 Reset Current Branch → Reset, 这种切换会抛弃原来的提交记录



**Reset Head指针, 会抛弃原来的提交记录
使Head指针强制指向指定的版本**

- 版本切换方式二：控制台 Version Control → Log → Revert Commit → Merge → 处理代码 → commit, 这种切换会当成一个新的提交记录, 之前的提交记录也都保留



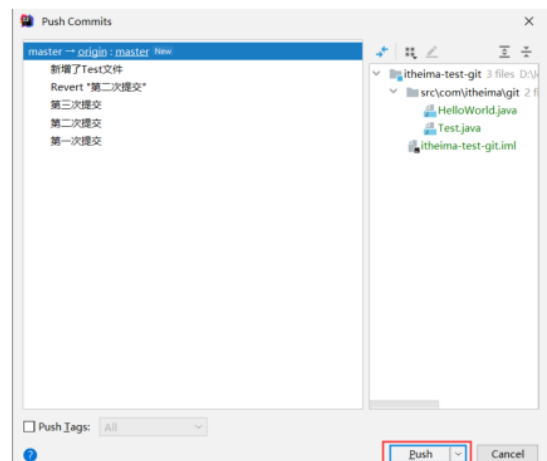
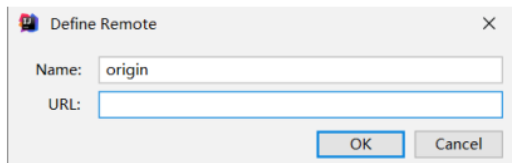


分支管理

- 创建分支: VCS → Git → Branches → New Branch → 给分支起名字 → ok
- 切换分支: idea 右下角 Git → 选择要切换的分支 → checkout
- 合并分支: VCS → Git → Merge changes → 选择要合并的分支 → merge
- 删除分支: idea 右下角 → 选中要删除的分支 → Delete

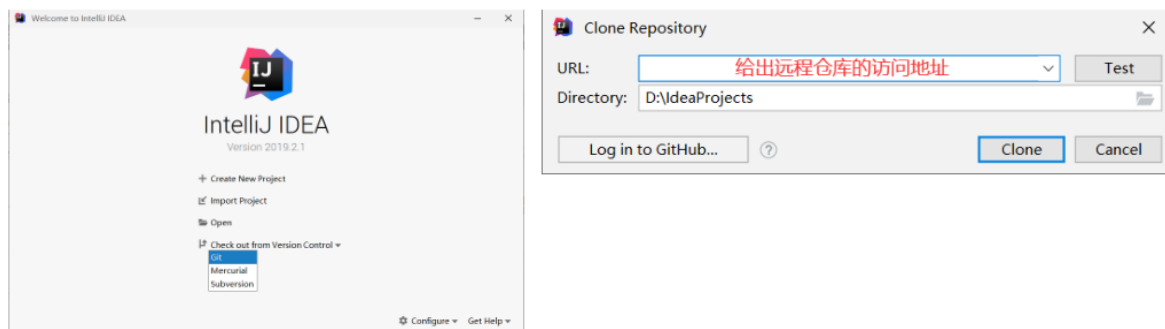
推送仓库

1. VCS → Git → Push → 点击 master Define remote
2. 将远程仓库的 url 路径复制过来 → Push



克隆仓库

File → Close Project → Checkout from Version Control → Git → 指定远程仓库的路径 → 指定本地存放的路径 → clone

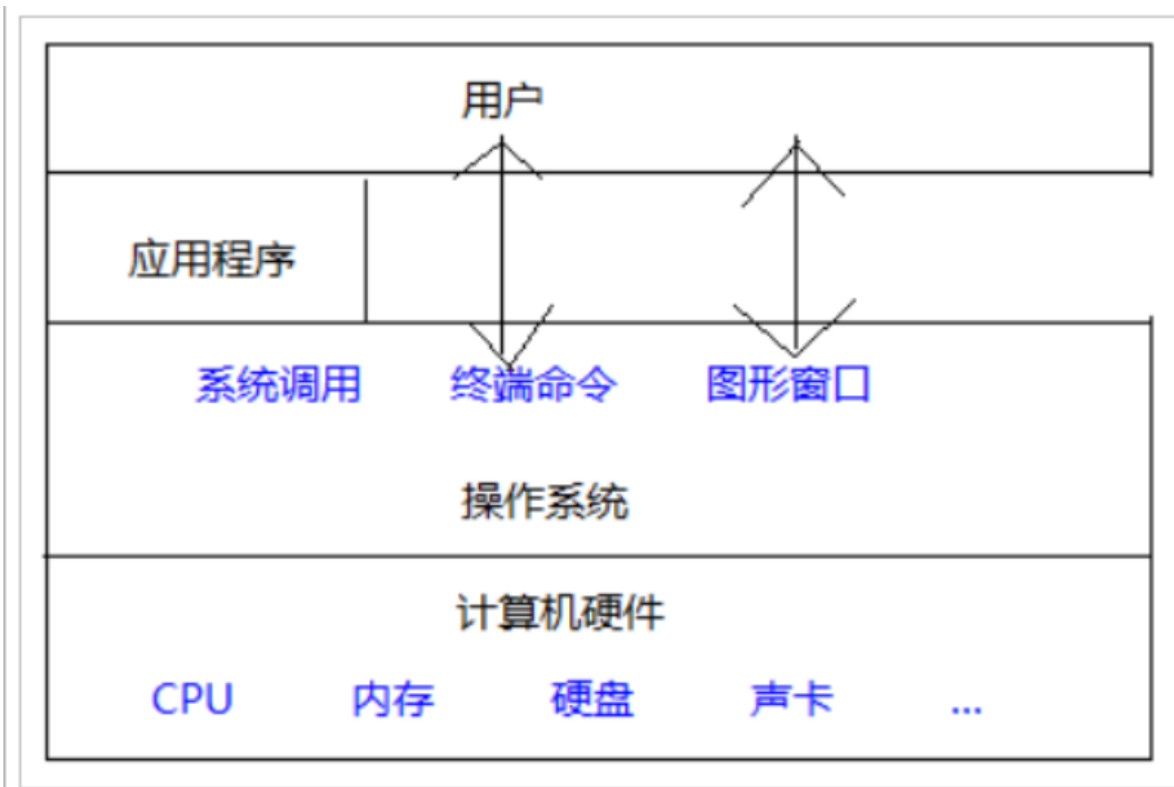


Linux

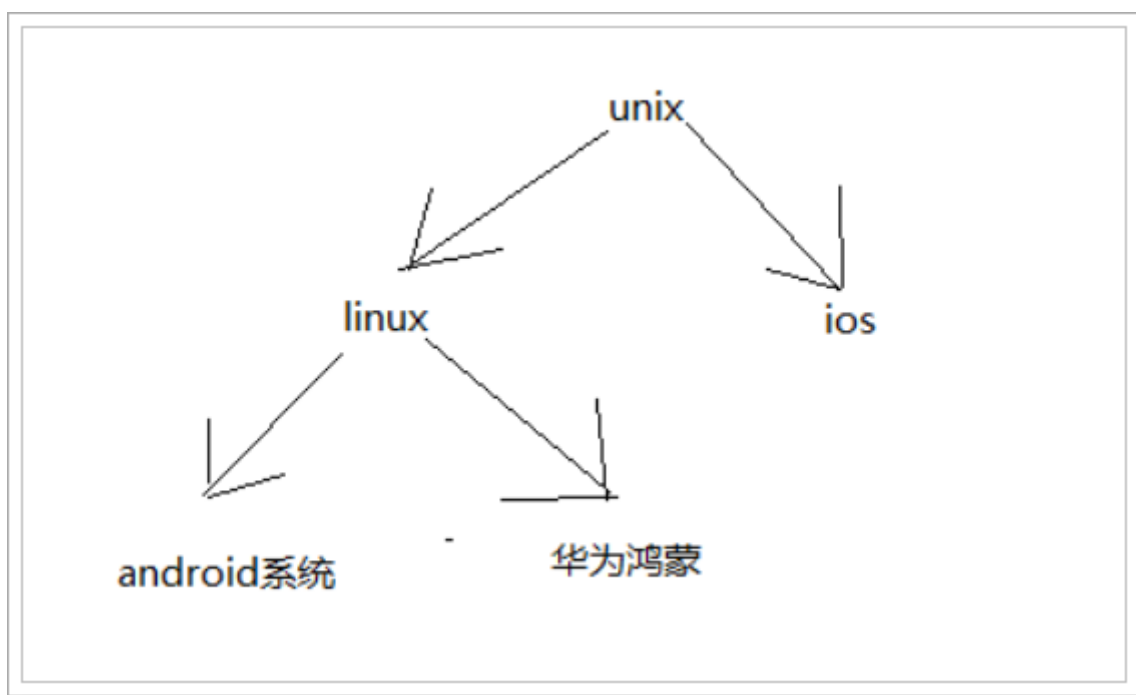
操作系统

操作系统（Operation System），是管理计算机硬件与软件资源的计算机程序，同时也是计算机系统的内核与基石。操作系统需要处理管理与配置内存、决定系统资源供需的优先次序、控制输入设备与输出设备、操作网络与管理文件系统等基本事务，操作系统也提供一个让用户与系统交互的操作界面

操作系统作为接口的示意图：



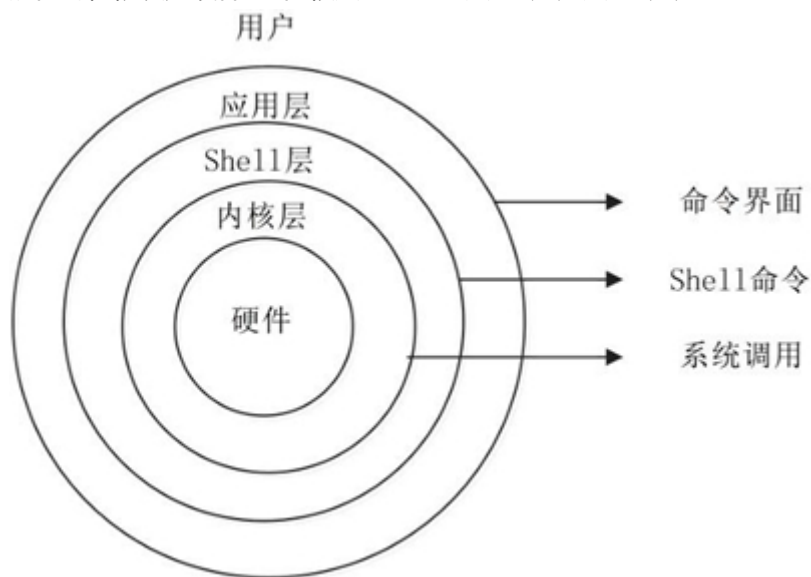
移动设备操作系统：



Linux系统

系统介绍

从内到外依次是硬件 → 内核层 → Shell 层 → 应用层 → 用户

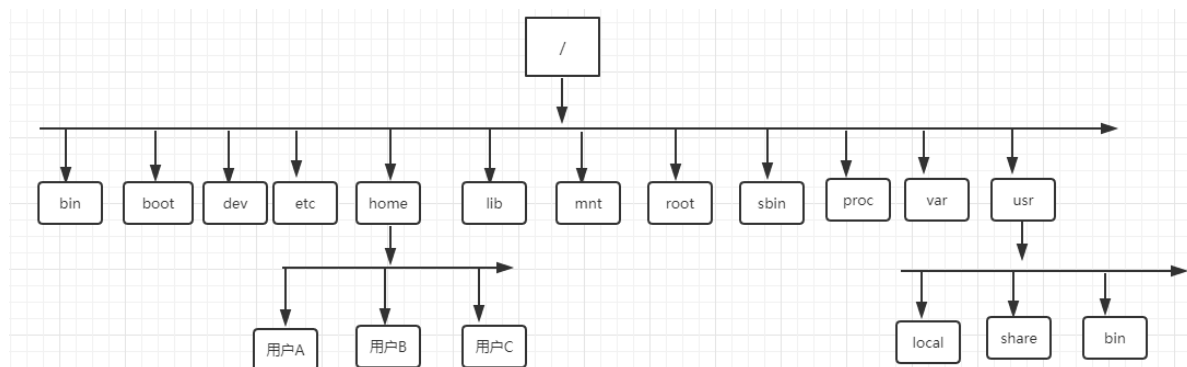


内核层：核心和基础，附着在硬件平台上，控制和管理系统内的各种资源，有效的组织进程的运行，扩展硬件的功能，提高资源利用效率，为用户提供安全可靠的应用环境。

Shell 层：与用户直接交互的界面。用户可以在提示符下输入命令行，由 Shell 解释执行并输出相应结果或者有关信息，所以我们也把 Shell 称作命令解释器，利用系统提供的丰富命令可以快捷而简便地完成许多工作。

文件系统

Linux 文件系统目录结构和熟知的 windows 系统有较大区别，没有各种盘符的概念。根目录只有一个 /，采用层级式的树状目录结构。



远程连接

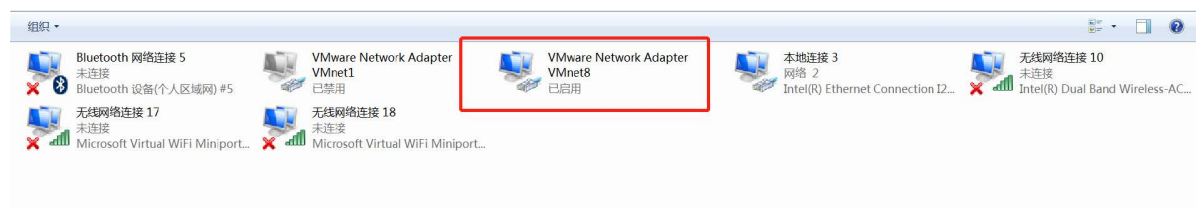
设置IP

NAT

首先设置虚拟机中 NAT 模式的选项，打开 VMware，点击编辑下的虚拟网络编辑器，设置 NAT 参数



注意：VMware Network Adapter VMnet8 保证是启用状态



静态IP

在普通用户下不能修改网卡的配置信息；所以我们要切换到 root 用户进行 ip 配置：su root/su

- 修改网卡配置文件：vim /etc/sysconfig/network-scripts/ifcfg-ens33
- 修改文件内容

```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
IPADDR=10.2.111.62
NETMASK=255.255.252.0
GATEWAY=10.2.111.254
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
```

UUID=2c2371f1-ef29-4514-a568-c4904bd11c82

DEVICE=ens33

ONBOOT=true

#####

BOOTPROTO设置为静态static

IPADDR设置ip地址

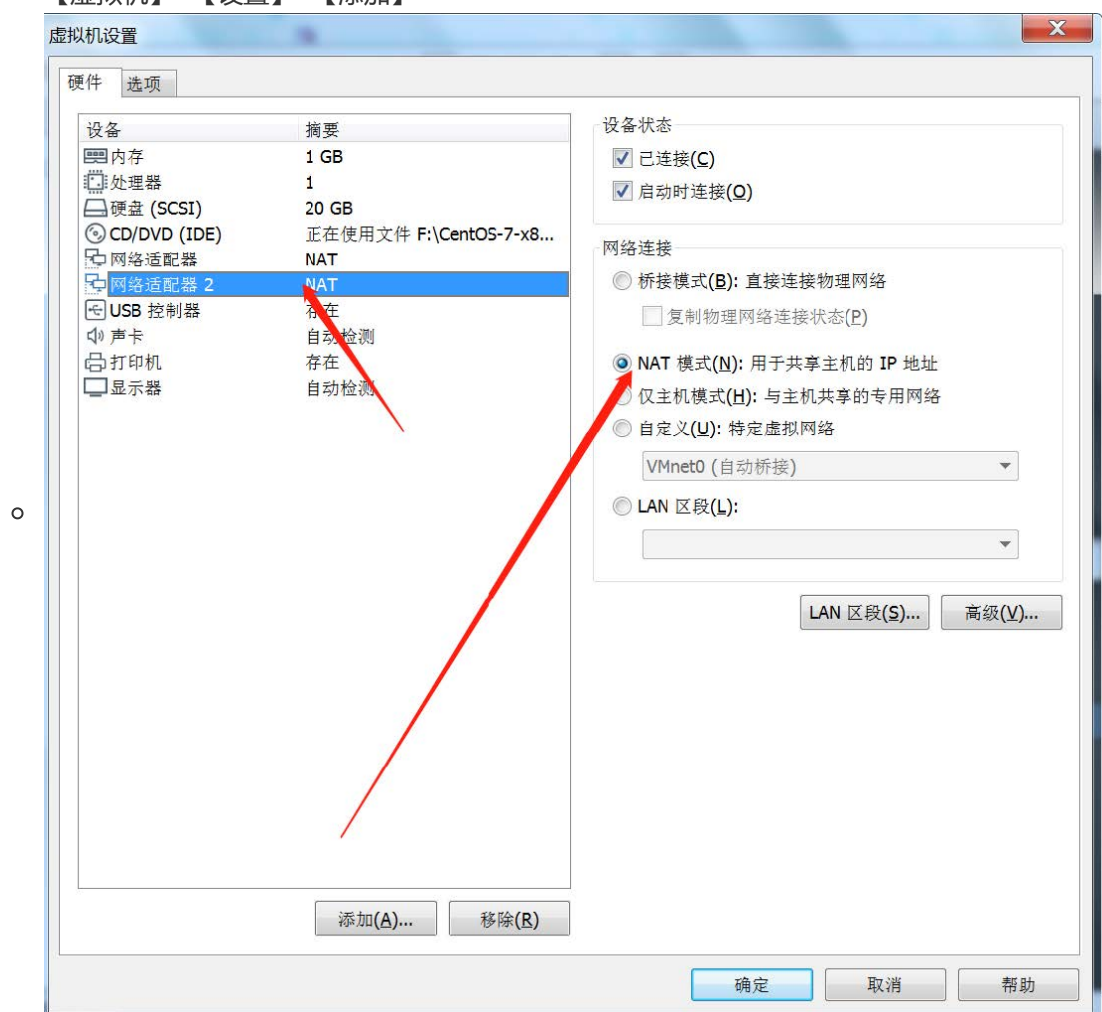
NETMASK设置子网掩码

GATEWAY设置网关

ONBOOT设置为true在系统启动时是否激活网卡

执行保存 :wq!

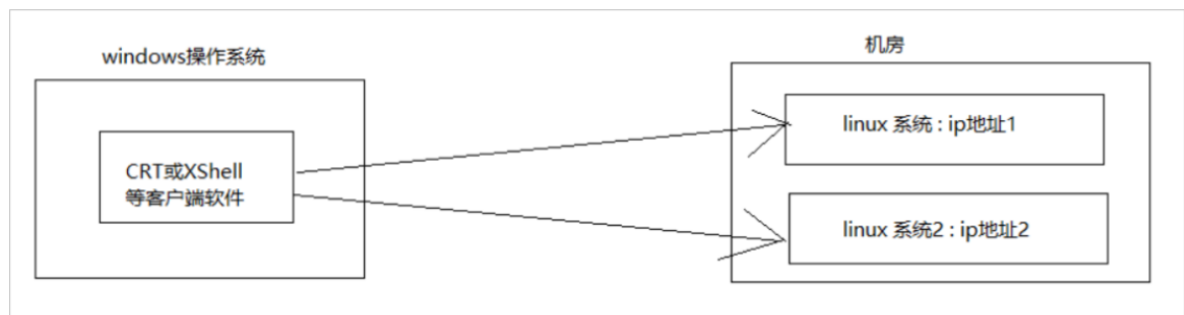
- 重启网络: systemctl restart network
- 查看IP: ifconfig
- 宿主机 ping 虚拟机, 虚拟机 ping 宿主机
- 在虚拟机中访问网络, 需要增加一块 NAT 网卡
 - 【虚拟机】 -- 【设置】 -- 【添加】



远程登陆

服务器维护工作 都是在 远程 通过 SSH 客户端 来完成的，并没有图形界面，所有的维护工作都需要通过命令来完成，Linux 服务器需要安装 SSH 相关服务

首先执行 `sudo apt-get install openssh-server` 指令，接下来用 xshell 连接



先用普通用户登录，然后转成 root

用户管理

Linux 系统是一个多用户、多任务的操作系统。多用户是指在 Linux 操作系统中可以创建多个用户，而这些多用户又可以同时执行各自不同的任务，而互不影响

在 Linux 系统中，会存在着以下几个概念：

- 用户名：用户的名称
- 用户所属的组：当前用户所属的组
- 用户的家目录：当前账号登录成功之后的目录，就叫做该用户的家目录

用户管理

当前用户

logname：用于显示目前用户的名称

- --help：在线帮助
- --vesion：显示版本信息

切换用户

`su UserName`：切换用户

`su -c comman root`：切换用户为 root 并在执行 comman 指令后退出返回原使用者

`su`：切换到 root 用户

用户添加

命令：useradd [options] 用户名

参数说明：

- -c comment 指定一段注释性描述
- -d 指定用户主目录，如果此目录不存在，则同时使用 -m 选项，可以创建主目录
- -m 创建用户的主目录
- -g 用户组，指定用户所属的用户组
- -G 用户组，用户组 指定用户所属的附加组
- -s Shell 文件 指定用户的登录 Shell
- -u 用户号，指定用户的用户号，如果同时有 -o 选项，则可以重复使用其他用户的标识号。

如何知道添加用户成功呢？通过指令 cat /etc/passwd 查看

```
seazean:x: 1000:1000:Seazean:/home/seazean:/bin/bash
用户名 密码 用户ID 组ID 注释 家目录 shell程序
```

useradd -m Username 新建用户成功之后，会建立 home 目录，但是此时有问题没有指定 shell 的版本，不是我们熟知的 bash，功能上有很多限制，进行 **sudo useradd -m -s /bin/bash Username**

用户密码

系统安装好默认的 root 用户是没有密码的，需要给 root 设置一个密码 **sudo passwd root**.

- 普通用户： **sudo passwd UserName**
- 管理员用户： passwd [options] UserName
 - -l: 锁定密码，即禁用账号
 - -u: 密码解锁
 - -d: 使账号无密码
 - -f: 强迫用户下次登录时修改密码

用户权限

usermod 命令通过修改系统帐户文件来修改用户账户信息

修改用户账号就是根据实际情况更改用户的有关属性，如用户号、主目录、用户组、登录 Shell 等

- 普通用户： sudo usermod [options] Username
- 管理员用户： usermod [options] Username
 - usermod -l newName Username
 - -l 新的登录名称

用户删除

删除用户账号就是要将 `/etc/passwd` 等系统文件中的该用户记录删除，必要时还删除用户的主目录

- 普通用户：`sudo userdel [options] Username`
 - 管理员用户：`userdel [options] Username`
 - `-f`：强制删除用户，即使用户当前已登录
 - `-r`：删除用户的同时，删除与用户相关的所有文件
-

用户组管理

组管理

添加组：**`groupadd`** 组名

创建用户的同时加入组：`useradd -m -g 组名 用户名`

添加用户组

新增一个用户组（组名可见名知意，符合规范即可），然后将用户添加到组中，需要使用管理员权限

命令：`groupadd [options] Groupname`

- `-g` GID 指定新用户组的组标识号（GID）
- `-o` 一般与 `-g` 选项同时使用，表示新用户组的 GID 可以与系统已有用户组的 GID 相同

新增用户组 Seazean：`groupadd Seazean`

修改用户组

需要使用管理员权限

命令：`groupmod [options] Groupname`

- `-g` GID 为用户组指定新的组标识号。
- `-o` 与 `-g` 选项同时使用，用户组的新 GID 可以与系统已有用户组的 GID 相同
- `-n` 新用户组 将用户组的名字改为新名字

修改 Seazean 组名为 zhy：`groupmod -n zhy Seazean`

删除用户组

- 普通用户：`sudo groupdel Groupname`
- 管理员用户：`groupdel Groupname`
 - `-f` 用户的主组也继续删除
 - `-h` 显示帮助信息

用户所属组

查询用户所属组：groups Username

查看用户及组信息：id Username

创建用户的同时加入组：useradd -m -g Groupname Username

修改用户所属组：usermod -g Groupname Username

usermod常用选项：

- -d 用户的新主目录
- -l 新的登录名称

gpasswd

gpasswd 是 Linux 工作组文件 /etc/group 和 /etc/gshadow 管理工具，用于将一个用户添加到组或从组中删除

命令：gpasswd 选项 Username Groupname

- -a 向组 GROUP 中添加用户 USER
- -d 从组 GROUP 中添加或删除用户

查看用户组下所有用户（所有用户）： grep 'Groupname' /etc/group

系统管理

man

在控制台输入：命令名 -h/ -help/ --h /空

可以看到命令的帮助文档

man [指令名称]：查看帮助文档，比如 man ls，退出方式 q

date

date 可以用来显示或设定系统的日期与时间

命令：date [options]

- -d<字符串>：显示字符串所指的日期与时间，字符串前后必须加上双引号；
- -s<字符串>：根据字符串来设置日期与时间，字符串前后必须加上双引号

- -u: 显示 GMT
- --version: 显示版本信息

查看时间: date → 2020年 11月 30日 星期一 17:10:54 CST

查看指定格式时间: date "+%Y-%m-%d %H:%M:%S" → 2020-11-30 17:11:44

设置日期指令: date -s "2019-12-23 19:21:00"

id

id 会显示用户以及所属群组的实际与有效 ID, 若两个 ID 相同则仅显示实际 ID; 若仅指定用户名称, 则显示目前用户的 ID

命令: id [-gGnru] [--help] [--version] [用户名称] //参数的顺序

- -g 或--group: 显示用户所属群组的 ID
- -G 或--groups: 显示用户所属附加群组的 ID
- -n 或--name: 显示用户, 所属群组或附加群组的名称。
- -r 或--real: 显示实际 ID
- -u 或--user: 显示用户 ID

id 命令参数虽然很多, 但是常用的是不带参数的 id 命令, 主要看 uid 和组信息

sudo

sudo: 控制用户对系统命令的使用权限, 通过 sudo 可以提高普通用户的操作权限

- -V 显示版本编号
- -h 会显示版本编号及指令的使用方式说明
- -l 显示出自己 (执行 sudo 的使用者) 的权限
- -command 要以系统管理者身份 (或以 -u 更改为其他人) 执行的指令

sudo -u root command -l: 指定 root 用户执行指令 command

top

top: 用于实时显示 process 的动态

- -c: command 属性进行了命令补全
- -p 进程号: 显示指定 pid 的进程信息
- -d 秒数: 表示进程界面更新时间 (每几秒刷新一次)
- -H 表示线程模式

top -Hp 进程 id：分析该进程内各线程的 CPU 使用情况

进程号	USER	PR	NI	VIRT	RES	SHR	%CPU	%MEM	TIME+	COMMAND
1750	seazean	20	0	256764	52508	32216 S	0.7	2.6	0:28.27	Xorg
1885	seazean	20	0	4108868	255204	105228 S	0.3	12.7	0:49.18	gnome-shell
3189	root	20	0	0	0	0 I	0.3	0.0	0:05.38	kworker/1:1-events
3253	seazean	20	0	1050452	70436	54092 S	0.3	3.5	0:05.83	gnome-terminal-
1	root	20	0	168852	12860	8484 S	0.0	0.6	0:03.73	systemd
2	root	20	0	0	0	0 S	0.0	0.0	0:00.01	kthreadd
3	root	0	-20	0	0	0 I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0 I	0.0	0.0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0 I	0.0	0.0	0:00.00	kworker/0:0H-kblockd
9	root	0	-20	0	0	0 I	0.0	0.0	0:00.00	mm_percpu_wq
10	root	20	0	0	0	0 S	0.0	0.0	0:00.20	ksoftirqd/0
11	root	20	0	0	0	0 I	0.0	0.0	0:01.66	rcu_sched
12	root	rt	0	0	0	0 S	0.0	0.0	0:00.10	migration/0
13	root	-51	0	0	0	0 S	0.0	0.0	0:00.00	idle_inject/0
14	root	20	0	0	0	0 S	0.0	0.0	0:00.00	cpuhp/0
15	root	20	0	0	0	0 S	0.0	0.0	0:00.00	cpuhp/1
16	root	-51	0	0	0	0 S	0.0	0.0	0:00.00	idle_inject/1
17	root	rt	0	0	0	0 S	0.0	0.0	0:00.19	migration/1
18	root	20	0	0	0	0 S	0.0	0.0	0:00.27	ksoftirqd/1
20	root	0	-20	0	0	0 I	0.0	0.0	0:00.00	kworker/1:0H-kblockd
21	root	20	0	0	0	0 S	0.0	0.0	0:00.00	kdevtmpfs
22	root	0	-20	0	0	0 I	0.0	0.0	0:00.00	netns
23	root	20	0	0	0	0 S	0.0	0.0	0:00.00	rcu_tasks_kthre
24	root	20	0	0	0	0 S	0.0	0.0	0:00.00	kauditd

各进程（任务）的状态监控属性解释说明：

- PID — 进程 id
- TID — 线程 id
- USER — 进程所有者
- PR — 进程优先级
- NI — nice 值，负值表示高优先级，正值表示低优先级
- VIRT — 进程使用的虚拟内存总量，单位 kb，VIRT=SWAP+RES
- RES — 进程使用的、未被换出的物理内存大小，单位 kb，RES=CODE+DATA
- SHR — 共享内存大小，单位 kb
- S — 进程状态，D=不可中断的睡眠状态 R=运行 S=睡眠 T=跟踪/停止 Z=僵尸进程
- %CPU — 上次更新到现在的 CPU 时间占用百分比
- %MEM — 进程使用的物理内存百分比
- TIME+ — 进程使用的 CPU 时间总计，单位 1/100 秒
- COMMAND — 进程名称（命令名/命令行）

ps

Linux 系统中查看进程使用情况的命令是 ps 指令

命令：ps

- -e: 显示所有进程
- -f: 全格式
- a: 显示终端上的所有进程
- u: 以用户的格式来显示进程信息
- x: 显示后台运行的进程
- -T: 开启线程查看
- -p: 指定线程号

一般常用格式为 ps -ef 或者 ps aux 两种。显示的信息大体一致，略有区别：

- 如果想查看进程的 CPU 占用率和内存占用率，可以使用 `aux`
- 如果想查看进程的父进程 ID 和完整的 COMMAND 命令，可以使用 `ef`

`ps -T -p <pid>`：显示某个进程的线程

ps 和 top 区别：

- `ps` 命令：可以查看进程的瞬间信息，是系统在过去执行的进程的静态快照
- `top` 命令：可以持续的监视进程的动态信息

kill

Linux `kill` 命令用于删除执行中的程序或工作，并不是让进程直接停止，而是给进程发一个信号，可以进入终止逻辑

命令：`kill [-s <信息名称或编号>] [程序]` 或 `kill [-l <信息编号>]`

- `-l <信息编号>`：若不加`<信息编号>`选项，则`-l`参数会列出全部的信息名称
- `-s <信息名称或编号>`：指定要送出的信息
- `-KILL`：强制杀死进程
- **`-9`：彻底杀死进程（常用）**
- `[程序]` 程序的 PID、PGID、工作编号

`kill 15642` . `kill -KILL 15642` . `kill -9 15642`

杀死指定用户所有进程：

1. 过滤出 `user` 用户进程：`kill -9 $(ps -ef | grep user)`
2. 直接杀死：`kill -u user`

shutdown

`shutdown` 命令可以用来进行关闭系统，并且在关机以前传送讯息给所有使用者正在执行的程序，`shutdown` 也可以用来重开机

普通用户：`sudo shutdown [-t seconds] [-rkhncfF] time [message]`

管理员用户：`shutdown [-t seconds] [-rkhncfF] time [message]`

- `-t seconds`：设定在几秒钟之后进行关机程序
- `-k`：并不会真的关机，只是将警告讯息传送给所有使用者
- `-r`：关机后重新开机
- `-h`：关机后停机
- `-n`：不采用正常程序来关机，用强迫的方式杀掉所有执行中的程序后自行关机
- `-c`：取消目前已经进行中的关机动作
- `-f`：关机时，不做 `fck` 动作（检查 Linux 档系统）

- -F: 关机时, 强迫进行 fsck 动作
- time: 设定关机的时间
- message: 传送给所有使用者的警告讯息

立即关机: `shutdown -h now` 或者 `shutdown now`

指定 1 分钟后关机并显示警告信息: `shutdown +1 "System will shutdown after 1 minutes"`

指定 1 分钟后重启并发出警告信息: `shutdown -r +1 "1分钟后关机重启"`

reboot

reboot 命令用于用来重新启动计算机

命令: `reboot [-n] [-w] [-d] [-f] [-i]`

- -n: 在重开机前不做将记忆体资料写回硬盘的动作
- -w: 并不会真的重开机, 只是把记录写到 `/var/log/wtmp` 档案里
- -d: 不把记录写到 `/var/log/wtmp` 档案里 (-n 这个参数包含了 -d)
- -f: 强迫重开机, 不呼叫 shutdown 这个指令
- -i: 在重开机之前先把所有网络相关的装置先停止

who

who 命令用于显示系统中有哪些使用者正在上面, 显示的资料包含了使用者 ID、使用的终端机、上线时间、CPU 使用量、动作等等

命令: `who - [husfv] [user]`

- -H 或 --heading: 显示各栏位的标题信息列 (常用 `who -H`)
 - -i 或 -u 或 --idle: 显示闲置时间, 若该用户在前一分钟之内有进行任何动作, 将标示成 `.` 号, 如果该用户已超过 24 小时没有任何动作, 则标示出 `old` 字符串
 - -m: 此参数的效果和指定 `am i` 字符串相同
 - -q 或 --count: 只显示登入系统的帐号名称和总人数
 - -s: 此参数将忽略不予处理, 仅负责解决 who 指令其他版本的兼容性问题
 - -w 或 -T 或 --mesg 或 --message 或 --writable: 显示用户的信息状态栏
 - --help: 在线帮助
 - --version: 显示版本信息
-

systemctl

命令：systemctl [command] [unit]

- --version 查看版本号
 - start：立刻启动后面接的 unit
 - stop：立刻关闭后面接的 unit
 - restart：立刻关闭后启动后面接的 unit，亦即执行 stop 再 start 的意思
 - reload：不关闭 unit 的情况下，重新载入配置文件，让设置生效
 - status：目前后面接的这个 unit 的状态，会列出有没有正在执行、开机时是否启动等信息
 - enable：设置下次开机时，后面接的 unit 会被启动
 - disable：设置下次开机时，后面接的 unit 不会被启动
 - is-active：目前有没有正在运行中
 - is-enable：开机时有没有默认要启用这个 unit
 - kill：不要被 kill 这个名字吓着了，它其实是向运行 unit 的进程发送信号
 - show：列出 unit 的配置
 - mask：注销 unit，注销后你就无法启动这个 unit 了
 - unmask：取消对 unit 的注销
-

timedatectl

timedatectl 用于控制系统时间和日期。可以查询和更改系统时钟于设定，同时可以设定和修改时区信息。在实际开发过程中，系统时间的显示会和实际出现不同步；我们为了校正服务器时间、时区会使用 timedatectl 命令

timedatectl：显示系统的时间信息

timedatectl status：显示系统的当前时间和日期

timedatectl | grep Time：查看当前时区

timedatectl list-timezones：查看所有可用的时区

timedatectl set-timezone "Asia/Shanghai"：设置本地时区为上海

timedatectl set-ntp true/false：启用/禁用时间同步

timedatectl set-time "2020-12-20 20:45:00"：时间同步关闭后可以设定时间

NTP 即 Network Time Protocol（网络时间协议），是一个互联网协议，用于同步计算机之间的系统时钟，timedatectl 实用程序可以自动同步你的Linux系统时钟到使用NTP的远程服务器

clear

clear 命令用于清除屏幕

通过执行 clear 命令，就可以把缓冲区的命令全部清理干净

exit

exit 命令用于退出目前的 shell

执行 exit 可使 shell 以指定的状态值退出。若不设置状态值参数，则 shell 以预设值退出。状态值 0 代表执行成功，其他值代表执行失败；exit 也可用在 script，离开正在执行的 script，回到 shell

命令：exit [状态值]

- 0 表示成功 (Zero - Success)
 - 非 0 表示失败 (Non-Zero - Failure)
 - 2 表示用法不当 (Incorrect Usage)
 - 127 表示命令没有找到 (Command Not Found)
 - 126 表示不是可执行的 (Not an executable)
 - 大于等于 128 信号产生
-

文件管理

常用命令

ls

ls命令相当于我们在Windows系统中打开磁盘、或者打开文件夹看到的目录以及文件的明细。

命令：ls [options] 目录名称

- -a：全部的文件，连同隐藏档(开头为 . 的文件)一起列出来(常用)
- -d：仅列出目录本身，而不是列出目录内的文件数据(常用)
- -l：显示不隐藏的文件与文件夹的详细信息；(常用)
- **ls -al = ll 命令**：显示所有文件与文件夹的详细信息

pwd

pwd 是 Print Working Directory 的缩写，也就是显示目前所在当前目录的命令

命令：pwd 选项

- -L 打印 \$PWD 变量的值，如果它包含了当前的工作目录
- -P 打印当前的物理路径，不带有任何的符号链接

cd

cd 是 Change Directory 的缩写，这是用来变换工作目录的命令

命令：cd [相对路径或绝对路径]

- cd ~：表示回到根目录
- cd ..：返回上级目录
- **相对路径** 在输入路径时, 最前面不是以 / 开始的, 表示相对**当前目录**所在的目录位置
 - 例如：/usr/share/doc
- **绝对路径** 在输入路径时, 最前面是以 / 开始的, 表示从**根目录**开始的具体目录位置
 - 由 /usr/share/doc 到 /usr/share/man 时, 可以写成：cd ../man
 - 优点：定位准确, 不会因为 工作目录变化 而变化

mkdir

mkdir命令用于建立名称为 dirName 之子目录

命令：mkdir [-p] dirName

- -p 确保目录名称存在，不存在的就建一个，用来创建多级目录。

`mkdir -p aaa/bbb`：在 aaa 目录下，创建一个 bbb 的子目录。若 aaa 目录原本不存在，则建立一个

rmdir

rmdir命令删除空的目录

命令：rmdir [-p] dirName

- -p 是当子目录被删除后使它也成为空目录的话，则顺便一并删除

`rmdir -p aaa/bbb`：在 aaa 目录中，删除名为 bbb 的子目录。若 bbb 删除后，aaa 目录成为空目录，则 aaa 同时也会被删除

cp

cp 命令主要用于复制文件或目录

命令：cp [options] source... directory

- -a: 此选项通常在复制目录时使用，它保留链接、文件属性，并复制目录下的所有内容。其作用等于dpR参数组合
- -d: 复制时保留链接。这里所说的链接相当于Windows系统中的快捷方式
- -f: 覆盖已经存在的目标文件而不给出提示
- -i: 与 -f 选项相反，在覆盖目标文件之前给出提示，要求用户确认是否覆盖，回答 y 时目标文件将被覆盖
- -p: 除复制文件的内容外，还把修改时间和访问权限也复制到新文件中
- -r/R: 若给出的源文件是一个目录文件，此时将复制该目录下所有的子目录和文件
- -l: 不复制文件，只是生成链接文件

`cp -r aaa/* ccc`: 复制 aaa 下的所有文件到 ccc，不加参数 -r 或者 -R，只复制文件，而略过目录

rm

rm命令用于删除一个文件或者目录。

命令: `rm [options] name...`

- -i 删除前逐一询问确认。
- -f 即使原档案属性设为唯读，亦直接删除，无需逐一确认
- -r 将目录及以下之档案亦逐一删除，递归删除

注: 文件一旦通过 rm 命令删除，则无法恢复，所以必须格外小心地使用该命令

mv

mv 命令用来为文件或目录改名、或将文件或目录移入其它位置

```
mv [options] source dest
mv [options] source... directory
```

- -i: 若指定目录已有同名文件，则先询问是否覆盖旧文件
- -f: 在 mv 操作要覆盖某已有的目标文件时不给任何指示

命令格式	运行结果
mv 文件名 文件名	将源文件名改为目标文件名
mv 文件名 目录名	将文件移动到目标目录
mv 目录名 目录名	目标目录已存在，将源目录移动到目标目录。目标目录不存在则改名
mv 目录名 文件名	出错

文件属性

基本属性

Linux 系统是一种典型的多用户系统，不同的用户处于不同的地位，拥有不同的权限。为了保护系统的安全性，Linux系统对不同的用户访问同一文件（包括目录文件）的权限做了不同的规定

```
seazean@Ubuntu1:~$ ls -l
总用量 36
drwxr-xr-x 2 seazean seazean 4096 12月 4 12:41 公共的
drwxr-xr-x 2 seazean seazean 4096 12月 4 12:23 模板
drwxr-xr-x 2 seazean seazean 4096 12月 4 12:23 视频
drwxr-xr-x 2 seazean seazean 4096 12月 4 12:23 图片
drwxr-xr-x 2 seazean seazean 4096 12月 4 12:23 文档
drwxr-xr-x 2 seazean seazean 4096 12月 4 12:23 下载
drwxr-xr-x 2 seazean seazean 4096 12月 4 12:23 音乐
drwxr-xr-x 2 seazean seazean 4096 12月 4 12:23 桌面
drwxrwxr-x 3 seazean seazean 4096 12月 4 12:42 Software
seazean@Ubuntu1:~$
```

在Linux中第一个字符代表这个文件是目录、文件或链接文件等等。

- 当为 d 则是目录
- 当为 - 则是文件
- 若是 l 则表示为链接文档 link file
- 若是 b 则表示为装置文件里面的可供储存的接口设备(可随机存取装置)
- 若是 c 则表示为装置文件里面的串行端口设备，例如键盘、鼠标(一次性读取装置)

接下来的字符，以三个为一组，均为[rwx] 的三个参数组合。其中，[r]代表可读(read)、[w]代表可写(write)、[x]代表可执行(execute)。要注意的是，这三个权限的位置不会改变，如果没有权限，就会出现[-]。

文件 类型	属主 权限			属组 权限			其他用户 权限		
0	1	2	3	4	5	6	7	8	9
d	rwx			r-x			r-x		
目录 文件	读	写	执行	读	写	执行	读	写	执行

从左至右用 0-9 这些数字来表示：

- 第 0 位确定文件类型
- 第 1-3 位确定属主拥有该文件的权限
- 第 4-6 位确定属组拥有该文件的权限
- 第 7-9 位确定其他用户拥有该文件的权限

文件信息

对于一个文件，都有一个特定的所有者，也就是对该文件具有所有权的用户（属主）；还有这个文件是属于哪个组的（属组）

- 文件的【属主】有一套【读写执行权限rwx】
- 文件的【属组】有一套【读写执行权限rwx】

```
itcast@localhost ~]$ ls -l
总用量 4
-rw-r--r--. 1 root root 348 12月 16 19:02 ~
-rwxrwxr-x. 3 itcast itcast 45 12月 17 11:28 aaa
-rwxrwxr-x. 2 itcast itcast 6 12月 17 11:26 jinyanlong
-rwxr-xr-x. 2 itcast itcast 6 12月 6 17:09 公共
-rwxr-xr-x. 2 itcast itcast 6 12月 6 17:09 模板
-rwxr-xr-x. 2 itcast itcast 6 12月 6 17:09 视频
-rwxr-xr-x. 2 itcast itcast 6 12月 6 17:09 图片
-rwxr-xr-x. 2 itcast itcast 6 12月 6 17:09 文档
-rwxr-xr-x. 2 itcast itcast 6 12月 6 17:09 下载
-rwxr-xr-x. 2 itcast itcast 6 12月 6 17:09 音乐
-rwxr-xr-x. 2 itcast itcast 6 12月 6 17:09 桌面
```

A B C D E F G

`ls -l` 可以查看文件夹下文件的详细信息, 从左到右 依次是:

- 权限 (A 区域) : 第一个字符如果是 `d` 表示目录
- 硬链接数 (B 区域) : 通俗的讲就是有多少种方式, 可以访问当前目录和文件
- 属主 (C 区域) : 文件是所有者、或是叫做属主
- 属组 (D 区域) : 文件属于哪个组
- 大小 (E 区域) : 文件大小
- 时间 (F 区域) : 最后一次访问时间
- 名称 (G 区域) : 文件的名称

更改权限

权限概述

Linux 文件属性有两种设置方法, 一种是数字, 一种是符号

Linux 的文件调用权限分为三级: 文件属主、属组、其他, 利用 `chmod` 可以控制文件如何被他人所调用。

```
chmod [-cfvR] [--help] [--version] mode file...
mode : 权限设定字符串, 格式: [ugoa...][+|=][rwxX]...[,...]
```

- `u` 表示档案的拥有者, `g` 表示与该档案拥有者属于同一个 group 者, `o` 表示其他的人, `a` 表示这三者皆是
- `+` 表示增加权限、`-` 表示取消权限、`=` 表示唯一设定权限
- `r` 表示可读取, `w` 表示可写入, `x` 表示可执行, `X` 表示只有该档案是个子目录或者该档案已经被设定过为可执行

数字权限

命令：chmod [-R] xyz 文件或目录

- xyz：就是刚刚提到的数字类型的权限属性，为 rwx 属性数值的相加
- -R：进行递归（recursive）的持续变更，亦即连同次目录下的所有文件都会变更

文件的权限字符为：[-rwxrwxrwx]，这九个权限是三三一组的，我们使用数字来代表各个权限

权限	英文	缩写	数字序号
读	read	r	4
写	write	w	2
执行	execute	x	1
无权限		-	0

各权限的数字对照表：[r]:4、[w]:2、[x]:1、[-]:0

每种身份（owner/group/others）的三个权限（r/w/x）分数是需要累加的，例如权限为：[-rwxrwx---] 分数是

- owner = rwx = 4+2+1 = 7
- group = rwx = 4+2+1 = 7
- others= --- = 0+0+0 = 0

表示为： `chmod -R 770 文件名`

符号权限

chmod	u	+(加入)	r	文件或目录
	g	-(除去)	w	
	o	=(设定)	x	
	a			

- user 属主权限
- group 属组权限
- others 其他权限
- all 全部的身份

我们就可以使用 **u g o a** 来代表身份的权限，读写的权限可以写成 **r w x**

`chmod u=rwx,g=rx,o=r a.txt`：将as.txt的权限设置为 **-rwxr-xr--**

`chmod a-r a.txt`：将文件的所有权限去除 **r**

更改属组

chgrp 命令用于变更文件或目录的所属群组

文件或目录权限的的拥有者由所属群组来管理，可以使用 chgrp 指令去变更文件与目录的所属群组

```
chgrp [-cfhRv] [--help] [--version] [所属群组] [文件或目录...]  
chgrp [-cfhRv] [--help] [--reference=<参考文件或目录>] [--version] [文件或目录...]
```

chgrp -v root aaa: 将文件 aaa 的属组更改成 root (其他也可以)

更改属主

利用 chown 可以将档案的拥有者加以改变。

使用权限：管理员账户

```
chown [-R] 属主名 文件名  
chown [-R] 属主名:属组名 文件名
```

chown root aaa: 将文件aaa的属主更改成root

chown seazean:seazean aaa: 将文件aaa的属主和属组更改为seazean

文件操作

touch

touch 命令用于创建文件、修改文件或者目录的时间属性，包括存取时间和更改时间。若文件不存在，系统会建立一个新的文件

```
touch [-acfm] [-d<日期时间>] [-r<参考文件或目录>] [-t<日期时间>] [--help] [--version] [文件  
或目录...]
```

- -a 改变档案的读取时间记录
- -m 改变档案的修改时间记录
- -c 假如目的档案不存在，不会建立新的档案。与 --no-create 的效果一样
- -f 不使用，是为了与其他 unix 系统的相容性而保留
- -r 使用参考档的时间记录，与 --file 的效果一样
- -d 设定时间与日期，可以使用各种不同的格式
- -t 设定档案的时间记录，格式与 date 指令相同
- --no-create 不会建立新档案

- --help 列出指令格式
- --version 列出版本讯息

`touch t.txt`: 创建 t.txt 文件

`touch t{1..10}.txt`: 创建10个名为 t1.txt 到 t10.txt 的空文件

`touch t.txt`: 更改 t.txt 的访问时间为现在

stat

stat 命令用于显示 inode 内容

命令: stat [文件或目录]

cat

cat 是一个文本文件查看和连接工具, **用于小文件**

命令: cat [-AbeEnstTuv] [--help] [--version] Filename

- -n 显示文件加上行号
- -b 和 -n 相似, 只不过对于空白行不编号

less

less 用于查看文件, 但是 less 在查看之前不会加载整个文件, **用于大文件**

命令: less [options] Filename

- -N 显示每行行号

tail

tail 命令可用于查看文件的内容, 有一个常用的参数 **-f** 常用于查阅正在改变的日志文件

命令: tail [options] Filename

- -f 循环读取,动态显示文档的最后内容
- -n 显示文件的尾部 n 行内容
- -c 显示字节数
- -nf 查看最后几行日志信息

`tail -f filename`: 动态显示最尾部的内容

`tail -n +2 txtfile.txt`: 显示文件 txtfile.txt 的内容, 从第 2 行至文件末尾

`tail -n 2 txtfile.txt`: 显示文件 txtfile.txt 的内容, 最后 2 行

head

head 命令可用于查看文件的开头部分的内容，有一个常用的参数 **-n** 用于显示行数，默认为 10

- -q 隐藏文件名
- -v 显示文件名
- -c 显示的字节数
- -n 显示的行数

`head -n Filename`：查看文件的前一部分

`head -n 20 Filename`：查看文件的前 20 行

grep

grep 指令用于查找内容包含指定的范本样式的文件，若不指定任何文件名称，或是所给予的文件名为 `-`，则 grep 指令会从标准输入设备读取数据

```
grep [-abcEFGhHiLnrsvVwxy] [-A<显示列数>] [-B<显示列数>] [-C<显示列数>] [-d<进行动作>] [-e<范本样式>] [-f<范本文件>] [--help] [范本样式] [文件或目录...]
```

- -c 只输出匹配行的计数
- -i 不区分大小写
- -h 查询多文件时不显示文件名
- -l 查询多文件时只输出包含匹配字符的文件名
- -n 显示匹配行及行号
- -s 不显示不存在或无匹配文本的错误信息
- -v 显示不包含匹配文本的所有行
- --color=auto 可以将找到的关键词部分加上颜色的显示

管道符 |：表示将前一个命令处理的结果传递给后面的命令处理

- `grep aaaa Filename`：显示存在关键字 aaaa 的行
- `grep -n aaaa Filename`：显示存在关键字 aaaa 的行，且显示行号
- `grep -i aaaa Filename`：忽略大小写，显示存在关键字 aaaa 的行
- `grep -v aaaa Filename`：显示存在关键字 aaaa 的所有行
- `ps -ef | grep sshd`：查找包含 sshd 进程的进程信息
- `ps -ef | grep -c sshd`：查找 sshd 相关的进程个数

echo

将字符串输出到控制台，通常和重定向联合使用

命令：`echo string`，如果字符串有空格，为了避免歧义，请增加双引号或者单引号

- 通过 `命令 > 文件` 将命令的成功结果覆盖指定文件内容
- 通过 `命令 >> 文件` 将命令的成功结果追加指定文件的后面
- 通过 `命令 &>> 文件` 将命令的失败结果追加指定文件的后面

`echo "程序员" >> a.txt`：将程序员追加到 a.txt 后面

`cat` 不存在的目录 &>> `error.log`：将错误信息追加到 `error.log` 文件

awk

AWK 是一种处理文本文件的语言，是一个强大的文本分析工具

```
awk [options] 'script' var=value file(s)
awk [options] -f scriptfile var=value file(s)
```

- `-F fs`：指定输入文件拆分分隔符，`fs` 是一个字符串或者是一个正则表达式
- `-v`：`var=value` 赋值一个用户定义变量
- `-f`：从脚本文件中读取 `awk` 命令
- `$n`：获取**第几段**内容
- `$0`：获取**当前行** 内容
- `NF`：表示当前行共有多少个字段
- `$NF`：代表最后一个字段
- `$(NF-1)`：代表倒数第二个字段
- `NR`：代表处理的是第几行

命令：`awk 'BEGIN{初始化操作}{每行都执行} END{结束时操作}'`

文件名 `BEGIN{` 这里面放的是执行前的语句 `}` {这里面放的是处理每一行时要执行的语句}

`END {` 这里面放的是处理完所有的行后要执行的语句 `}`

```
//准备数据
zhangsan 68 99 26
lisi 98 66 96
wangwu 38 33 86
zhaoliu 78 44 36
maq 88 22 66
zhouba 98 44 46
```

- `cat a.txt | awk '/zhang|li/'`：搜索含有 `zhang` 和 `li` 的学生成绩
- `awk "/zhang|li/" a.txt`：同上一个命令，效果一样

```
zhangsan 68 99 26
lisi 98 66 96
zhaoliu 78 44 36
```

- `cat a.txt | awk -F ' ' '{print $1,$2,$3}'`：按照空格分割，打印一二三列内容
- `awk -F ' ' '{OFS="\t"}{print $1,$2,$3}'`：按照制表符 `tab` 进行分割，打印一二三列
`\b`：退格 `\f`：换页 `\n`：换行 `\r`：回车 `\t`：制表符

```
zhangsan    68  99
lisi        98  66
wangwu      38  33
zhaoliu     78  44
maq         88  22
zhouba      98  44
```

- `awk -F ' ' '{print toupper($1)}' a.txt`: 根据逗号分割, 打印内容, 第一段大写

函数名	含义	作用
<code>toupper()</code>	upper	字符 转成 大写
<code>tolower()</code>	lower	字符 转成小写
<code>length()</code>	length	返回 字符长度

- `awk -F ' ' 'BEGIN{{total=total+$4} END{print total}}' a.txt`: 计算的是第4列的总分
- `awk -F ' ' 'BEGIN{{total=total+$4} END{print total, NR}}' a.txt`: 查看总分, 总人数
- `awk -F ' ' 'BEGIN{{total=total+$4} END{print total, NR, (total/NR)}}' a.txt`: 查看总分, 总人数, 平均数
- `cat a.txt | awk -F ' ' 'BEGIN{{total=total+$4} END{print total}}'`: 可以这样写

find

`find` 命令用来在指定目录下查找文件, 如果使用该命令不设置任何参数, 将在当前目录下查找子目录与文件, 并且将查找到的子目录和文件全部进行显示

命令: `find <指定目录> <指定条件> <指定内容>`

- `find . -name "*.gz"`: 将目前目录及其子目录下所有延伸档名是 `gz` 的文件查询出来
- `find . -ctime -1`: 将目前目录及其子目录下所有最近 1 天内更新过的文件查询出来
- `find / -name 'seazean'`: 全局搜索 `seazean`

read

`read` 命令用于从标准输入读取数值

```
read [-ers] [-a aname] [-d delim] [-i text] [-n nchars] [-N nchars] [-p prompt]
[-t timeout] [-u fd] [name ...]
```

sort

Linux `sort` 命令用于将文本文件内容加以排序

```
sort [-bcdfimMnr] [文件]
```

- `-n` 依照数值的大小排序

- -r 以相反的顺序来排序 (sort 默认的排序方式是**升序**, 改成降序, 加 -r)
- -u 去掉重复

面试题: 一系列数字, 输出最大的 4 个不重复的数

```
sort -ur a.txt | head -n 4  
sort -r a.txt | uniq | head -n 4
```

uniq

uniq 用于重复数据处理, 使用前先 sort 排序

```
uniq [OPTION]... [INPUT [OUTPUT]]
```

- -c 在数据行前出现的次数
- -d 只打印重复的行, 重复的行只显示一次
- -D 只打印重复的行, 重复的行出现多少次就显示多少次
- -f 忽略行首的几个字段
- -i 忽略大小写
- -s 忽略行首的几个字母
- -u 只打印唯一的行
- -w 比较不超过 n 个字母

文件压缩

tar

tar 的主要功能是打包、压缩和解压文件, tar 本身不具有压缩功能, 是调用压缩功能实现的。

命令: tar [必要参数] [选择参数] [文件]

- -c 产生 .tar 文件
- -v 显示详细信息
- -z 打包同时压缩
- -f 指定压缩后的文件名
- -x 解压 .tar 文件
- -t 列出 tar 文件中包含的文件的信息
- -r 附加新的文件到tar文件中

`tar -cvf txt.tar txtfile.txt`: 将 txtfile.txt 文件打包 (仅打包, 不压缩)

`tar -zcvf combine.tar.gz 1.txt 2.txt 3.txt`: 将 123.txt 文件打包压缩 (gzip)

`tar -ztvf txt.tar.gz`: 查看 tar 中有哪些文件

```
tar -zxvf Filename -C 目标路径: 解压
```

gzip

gzip命令用于压缩文件。

gzip是个使用广泛的压缩程序，文件经它压缩过后，其名称后面会多出".gz"的扩展名

- gzip *：压缩目录下的所有文件，删除源文件。不支持直接压缩目录
- gzip -rv 目录名：递归压缩目录
- gzip -dv *：解压文件并列出具体的信息

gunzip

gunzip命令用于解压文件。用于解开被gzip压缩过的文件

命令：gunzip [options] [文件或者目录]

gunzip 001.gz：解压001.gz文件

zip

zip 命令用于压缩文件。

zip 是个使用广泛的压缩程序，文件经它压缩后会另外产生具有 `.zip` 扩展名的压缩文件

命令：zip [必要参数] [选择参数] [文件]

- -q 不显示指令执行过程
- -r 递归处理，将指定目录下的所有文件和子目录一并处理

```
zip -q -r z.zip *：将该目录的文件全部压缩
```

unzip

unzip 命令用于解压缩 zip 文件，unzip 为 `.zip` 压缩文件的解压缩程序

命令：unzip [必要参数] [选择参数] [文件]

- -l 查看压缩文件内所包含的文件
- -d<目录> 指定文件解压缩后所要存储的目录。

```
unzip -l z.zip：查看压缩文件中包含的文件
```

```
unzip -d ./unFiles z.zip：把文件解压到指定的目录下
```

bzip2

bzip2 命令是 `.bz2` 文件的压缩程序。

bzip2 采用新的压缩演算法，压缩效果比传统的 LZ77/LZ78 压缩演算法好，若不加任何参数，bzip2 压缩完文件后会产生 `.bz2` 的压缩文件，并删除原始的文件

```
bzip2 [-cdfhkLstvvz][--repetitive-best][--repetitive-fast][- 压缩等级][要压缩的文件]
```

压缩：bzip2 a.txt

bunzip2

bunzip2 命令是 `.bz2` 文件的解压缩程序。

命令：bunzip2 [-fkLsvV] [.bz2压缩文件]

- -v 解压缩文件时，显示详细的信息。

解压：bunzip2 -v a.bz2

文件编辑

Vim

vim：是从 vi 发展出来的一个文本编辑器

- 命令模式：在 Linux 终端中输入 `vim 文件名` 就进入了命令模式，但不能输入文字
- 编辑模式：在命令模式下按 `i` 就会进入编辑模式，此时可以写入程式，按 Esc 可回到命令模式
- 末行模式：在命令模式下按 `:` 进入末行模式，左下角会有一个冒号，可以敲入命令并执行

打开文件

Ubuntu 默认没有安装 vim，需要先安装 vim，安装命令：**sudo apt-get install vim**

Vim 有三种模式：命令模式（Command mode）、插入模式（Insert mode）、末行模式（Last Line mode）

Vim 使用的选项	说明	常用
vim filename	打开或新建一个文件，将光标置于第一行首部	常用
vim -r filename	恢复上次vim打开时崩溃的文件	
vim -R filename	把指定的文件以只读的方式放入Vim编辑器	
vim + filename	打开文件，将光标置于最后一行的首部	常用
vim +n filename	打开文件，将光标置于n行的首部	常用
vim +/pattern filename	打开文件，将光标置于第一个与pattern匹配的位置	
vim -c command filename	对文件编辑前，先执行指定的命令	

插入模式

在命令模式下，通过按下 i、I、a、A、o、O 这 6 个字母进入插入模式

快捷键	功能描述
i	在光标所在位置插入文本，光标后的文本向右移动
I	在光标所在行的行首插入文本，行首是该行的第一个非空白字符
o	在光标所在行的下面插入新的一行，光标停在空行首
O	在光标所在行的上面插入新的一行，光标停在空行首
a	在光标所在位置之后插入文本
A	在光标所在行的行尾插入文本

按下 ESC 键，离开插入模式，进入命令模式

因为我们是一个空文件，所以使用【I】或者【i】都可以

如果里面的文本很多，要使用【A】进入编辑模式，即在行末添加文本

命令模式

Vim 打开一个文件（文件可以存在，也可以不存在），默认进入命令模式。在该模式下，输入的字符会被当做指令，而不会被当做要输入的文字

移动光标

快捷键	功能描述
w	光标移动至下一个单词的单词首
b	光标移动至上一个单词的单词首
e	光标移动至下一个单词的单词尾
0	光标移动至当前行的行首
^	行首, 第一个不是空白字符的位置
\$	光标移动至当前行的行尾
gg	光标移动至文件开头
G	光标移动至文件末尾
ngg	光标移动至第n行
nG	光标移动至第n行
:n	光标移动至第n行

选中文本

在 vi/vim 中要选择文本，需要显示 visual 命令切换到**可视模式**

vi/vim 中提供了三种可视模式，方便程序员的选择**选中文本的方式**

按 ESC 可以放弃选中, 返回到**命令模式**

命令	模式	功能
v	可视模式	从光标位置开始按照正常模式选择文本
V	可视化模式	选中光标经过的完整行
Ctrl + v	可是块模式	垂直方向选中文本

撤销删除

在学习编辑命令之前,先要知道怎样撤销之前一次错误的编辑操作

命令	英文	功能
u	undo	撤销上次的命令(ctrl + z)
Ctrl + r	uredo	恢复撤销的命令

删除的内容此时并没有真正的被删除，在剪切板中，按下 p 键，可以将删除的内容粘贴回来

快捷键	功能描述
x	删除光标所在位置的字符
d	删除移动命令对应的内容
dd	删除光标所在行的内容
D	删除光标位置到行尾的内容
:n1,n2	删除从 a1 到 a2 行的文本内容

删除命令可以和移动命令连用, 以下是常见的组合命令(扩展):

命令	作用
dw	删除从光标位置到单词末尾
d}	删除从光标位置到段落末尾
dG	删除光标所行到文件末尾的所有内容
ndd	删除当前行（包括此行）到后 n 行内容

复制粘贴

vim 中提供有一个 被复制文本的缓冲区

- 复制命令会将选中的文字保存在缓冲区
- 删除命令删除的文字会被保存在缓冲区
- 在需要的位置，使用粘贴命令可以将缓冲对的文字插入到光标所在的位置
- vim 中的文本缓冲区只有一个，如果后续做过复制、剪切操作，之前缓冲区中的内容会被替换

快捷键	功能描述
y	复制已选中的文本到剪切板
yy	将光标所在行复制到剪切板
nyy	复制从光标所在行到向下n行
p	将剪切板中的内容粘贴到光标后
P	将剪切板中的内容粘贴到光标前

注意：vim 中的文本缓冲区和系统的剪切板不是同一个，在其他软件中使用 Ctrl + C 复制的内容，不能在 vim 中通过 p 命令粘贴，可以在编辑模式下使用鼠标右键粘贴

查找替换

查找

快捷键	功能描述
/abc	从光标所在位置向后查找字符串 abc
/^abc	查找以 abc 为行首的行
/abc\$	查找以 abc 为行尾的行
?abc	从光标所在位置向前查找字符串 abc
*	向后查找当前光标所在单词
#	向前查找当前光标所在单词
n	查找下一个，向同一方向重复上次的查找指令
N	查找上一个，向相反方向重复上次的查找指令

替换：

命令	功能	工作模式
r	替换当前字符	命令模式
R	替换当前行光标后的字符	替换模式

- 光标选中要替换的字符
- **R** 命令可以进入替换模式，替换完成后，按下 ESC 可以回到命令模式
- 替换命令的作用就是不用进入编辑模式，对文件进行轻量级的修改

末行模式

在命令模式下，按下 **:** 键进入末行模式

命令	功能描述
:wq	保存并退出 Vim 编辑器
:wq!	保存并强制退出 Vim 编辑器
:q	不保存且退出 Vim 编辑器
:q!	不保存且强制退出 Vim 编辑器
:w	保存但是不退出 Vim 编辑器
:w!	强制保存但是不退出 Vim 编辑器
:w filename	另存到 filename 文件
x!	保存文本，退出保存但是不退出 Vim 编辑器，更通用的命令
ZZ	直接退出保存但是不退出 Vim 编辑器
:n	光标移动至第 n 行行首

异常处理

- 如果 vim 异常退出, 在磁盘上可能会保存有 交换文件
- 下次再使用 vim 编辑文件时, 会看到以下屏幕信息:

```
E325: 注意
发现交换文件 ".a.txt.swp"
    所有者: seazean    日期: 一 12月 21 21:18:30 2020
    文件名: ~/seazean/a.txt
    修改过: 否
    用户名: seazean    主机名: Ubuntu1
    进程 ID: 4724 (STILL RUNNING)
正在打开文件 "a.txt"
    日期: 一 12月 21 21:10:29 2020

(1) Another program may be editing the same file.  If this is the case,
    be careful not to end up with two different instances of the same
    file when making changes.  Quit, or continue with caution.
(2) An edit session for this file crashed.
    如果是这样, 请用 ":recover" 或 "vim -r a.txt"
    恢复修改的内容 (请见 ":help recovery")。
    如果你已经进行了恢复, 请删除交换文件 ".a.txt.swp"
    以避免再看到此消息。

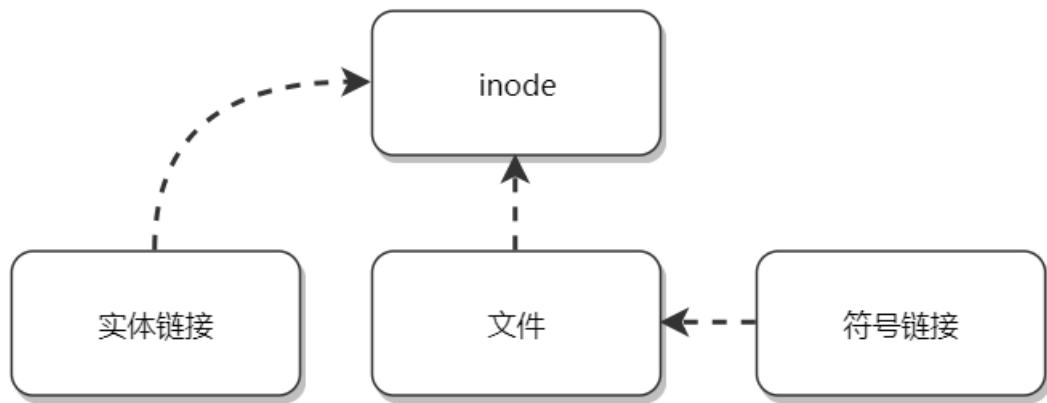
交换文件 ".a.txt.swp" 已存在!
以只读方式打开([O]), 直接编辑((E)), 恢复((R)), 退出((Q)), 中止((A)):
交换文件 ".a.txt.swp" 已存在!
以只读方式打开([O]), 直接编辑((E)), 恢复((R)), 退出((Q)), 中止((A)): 
```

- ls -a 一下, 会看到隐藏的 .swp 文件, 删除了此文件即可

链接

```
ln [-sf] source_filename dest_filename
```

- s: 默认是实体链接, 加 -s 为符号链接
- f: 如果目标文件存在时, 先删除目标文件



实体链接：

- 在目录下创建一个条目，记录着文件名与 inode 编号，这个 inode 就是源文件的 inode
- 删除任意一个条目，文件还是存在，只要引用数量不为 0
- 不能跨越文件系统、不能对目录进行链接

```
ln /etc/crontab .  
11  
34474855 -rw-r--r--. 2 root root 451 Jun 10 2014 crontab  
34474855 -rw-r--r--. 2 root root 451 Jun 10 2014 /etc/crontab
```

符号链接：

- 符号链接文件保存着源文件所在的绝对路径，在读取时会定位到源文件上，可以理解为 Windows 的快捷方式
- 当源文件被删除了，链接文件就打不开了
- 记录的是路径，所以可以为目录建立符号链接

```
34474855 -rw-r--r--. 2 root root 451 Jun 10 2014 /etc/crontab  
53745909 lrwxrwxrwx. 1 root root 12 Jun 23 22:31 /root/crontab2 ->  
/etc/crontab
```

进程管理

查看进程

ps 指令：查看某个时间点的进程信息

top 指令：实时显示进程信息

pstree：查看进程树

进程 ID

进程号：

- 进程号为 0 的进程通常是调度进程，常常被称为交换进程（swapper），该进程是内核的一部分，它并不执行任何磁盘上的程序，因此也被称为系统进程
- 进程号为 1 是 init 进程，是一个守护进程，在自举过程结束时由内核调用，init 进程绝不会终止，是一个普通的用户进程，但是它以超级用户特权运行

父进程 ID 为 0 的进程通常是内核进程，作为系统**自举过程**的一部分而启动，init 进程是个例外，它的父进程是 0，但它是用户进程

- 主存 = RAM + BIOS 部分的 ROM
- DISK：存放 OS 和 Bootloader
- BIOS：基于 I/O 处理系统
- Bootloader：加载 OS，将 OS 放入内存

自举程序存储在内存中 ROM，**用来加载操作系统**，初始化 CPU、寄存器、内存等。CPU 的程序计数器指自举程序第一条指令，当计算机**通电**，CPU 开始读取并执行自举程序，将操作系统（不是全部，只是启动计算机的那部分程序）装入 RAM 中，这个过程是自举过程。装入完成后程序计数器设置为 RAM 中操作系统的**第一条指令**，接下来 CPU 将开始执行（启动）操作系统的指令

存储在 ROM 中保留很小的自举装入程序，完整功能的自举程序保存在磁盘的启动块上，启动块位于磁盘的固定位，拥有启动分区的磁盘称为启动磁盘或系统磁盘（C 盘）

进程状态

状态	说明
R	running or runnable (on run queue) 正在执行或者可执行，此时进程位于执行队列中
D	uninterruptible sleep (usually I/O) 不可中断阻塞，通常为 IO 阻塞
S	interruptible sleep (waiting for an event to complete) 可中断阻塞，此时进程正在等待某个事件完成
Z	zombie (terminated but not reaped by its parent) 僵死，进程已经终止但是尚未被其父进程获取信息
T	stopped (either by a job control signal or because it is being traced) 结束，进程既可以被作业控制信号结束，也可能是正在被追踪

孤儿进程：

- 一个父进程退出，而它的一个或多个子进程还在运行，那么这些子进程将成为孤儿进程
- 孤儿进程将被 init 进程所收养，并由 init 进程对它们完成状态收集工作，所以孤儿进程不会对系统造成危害

僵尸进程：

- 一个子进程的进程描述符在子进程退出时不会释放，只有当父进程通过 wait() 或 waitpid() 获取了子进程信息后才会释放。如果子进程退出，而父进程并没有调用 wait() 或 waitpid()，那么子进程的进程描述符仍然保存在系统中，这种进程称之为僵尸进程
- 僵尸进程通过 ps 命令显示出来的状态为 Z (zombie)
- 系统所能使用的进程号是有限的，产生大量僵尸进程，会导致系统没有可用的进程号而不能产生新的进程
- 要消灭系统中大量的僵尸进程，只需要将其父进程杀死，此时僵尸进程就会变成孤儿进程，从而被 init 进程所收养，这样 init 进程就会释放所有的僵尸进程所占有的资源，从而结束僵尸进程

补充：

- 守护进程(daemon)是一类在后台运行的特殊进程，用于执行特定的系统任务。
- 守护进程是**脱离于终端**并且在后台运行的进程，脱离终端是为了避免在执行的过程中的信息在终端上显示，并且进程也不会被任何终端所产生的终端信息所打断
- 很多守护进程在系统引导的时候启动，并且一直运行直到系统关闭；另一些只在需要的时候才启动，完成任务后就自动结束

状态改变

SIGCHLD

当一个子进程改变了它的状态时（停止运行，继续运行或者退出），有两件事会发生在父进程中：

- 得到 SIGCHLD 信号
- waitpid() 或者 wait() 调用会返回

子进程发送的 SIGCHLD 信号包含了子进程的信息，比如进程 ID、进程状态、进程使用 CPU 的时间等；在子进程退出时进程描述符不会立即释放，父进程通过 wait() 和 waitpid() 来获得一个已经退出的子进程的信息，释放子进程的 PCB

wait

```
pid_t wait(int *status)
```

参数：status 用来保存被收集的子进程退出时的状态，如果不关心子进程**如何**销毁，可以设置这个参数为 NULL

父进程调用 wait() 会阻塞等待，直到收到一个子进程退出的 SIGCHLD 信号，wait() 函数就会销毁子进程并返回

- 成功，返回被收集的子进程的进程 ID
- 失败，返回 -1，同时 errno 被置为 ECHILD（如果调用进程没有子进程，调用就会失败）

waitpid

```
pid_t waitpid(pid_t pid, int *status, int options)
```

作用和 wait() 完全相同，只是多了两个可控制的参数 pid 和 options

- pid：指示一个子进程的 ID，表示只关心这个子进程退出的 SIGCHLD 信号；如果 pid=-1 时，那么和 wait() 作用相同，都是关注所有子进程退出的 SIGCHLD 信号
- options：主要有 WNOHANG 和 WUNTRACED 两个，WNOHANG 可以使 waitpid() 调用变成非阻塞的，就是会立即返回，父进程可以继续执行其它任务

网络管理

network

- 启动：service network start
- 停止：service network stop
- 重启：service network restart

ifconfig

ifconfig 是 Linux 中用于显示或配置网络设备的命令，英文全称是 network interfaces configuring

ifconfig 命令用于显示或设置网络设备。ifconfig 可设置网络设备的状态，或是显示目前的设置

```
ifconfig [网络设备][down up -allmulti -arp -promisc][add<地址>][del<地址>][<hw<网络设备类型>><硬件地址>][io_addr<I/O地址>][irq<IRQ地址>][media<网络媒介类型>][mem_start<内存地址>][metric<数目>][mtu<字节>][netmask<子网掩码>][tunnel<地址>][-broadcast<地址>][-pointopoint<地址>][IP地址]
```


- `ifconfig`: 显示激活的网卡信息 `ens`

```
seazean@Ubuntu1:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.137 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::43d7:2283:38cc:73e4 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:b7:6c:de txqueuelen 1000 (以太网)
    RX packets 4671 bytes 1838430 (1.8 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1987 bytes 166992 (166.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (本地环回)
    RX packets 355 bytes 30206 (30.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 355 bytes 30206 (30.2 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

`ens33` (或 `eth0`) 表示第一块网卡, IP地址是 192.168.0.137, 广播地址 broadcast 192.168.0.255, 掩码地址 netmask 255.255.255.0, `inet6` 对应的是 ipv6

`lo` 是表示主机的回环地址, 用来测试一个网络程序, 但又不想让局域网或外网的用户能够查看, 只能在此台主机上运行和查看所用的网络接口

- `ifconfig ens33 down`: 关闭网卡
- `ifconfig ens33 up`: 启用网卡

ping

`ping` 命令用于检测主机

执行 `ping` 指令会使用 ICMP 传输协议, 发出要求回应的信息, 若远端主机的网络功能没有问题, 就会回应该信息

```
ping [-dfnqrRV] [-c<完成次数>] [-i<间隔秒数>] [-I<网络界面>] [-l<前置载入>] [-p<范本样式>] [-s<数据包大小>] [-t<存活数值>] [主机名称或IP地址]
```

- `-c<完成次数>`: 设置完成要求回应的次数;

- `ping -c 2 www.baidu.com`

```
seazean@Ubuntu1:~$ ping -c 4 www.baidu.com
PING www.a.shifen.com (14.215.177.38) 56(84) bytes of data.
64 bytes from 14.215.177.38 (14.215.177.38): icmp_seq=1 ttl=55 time=22.6 ms
64 bytes from 14.215.177.38 (14.215.177.38): icmp_seq=2 ttl=55 time=23.9 ms
64 bytes from 14.215.177.38 (14.215.177.38): icmp_seq=3 ttl=55 time=24.0 ms
64 bytes from 14.215.177.38 (14.215.177.38): icmp_seq=4 ttl=55 time=24.1 ms

--- www.a.shifen.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 22.587/23.629/24.114/0.608 ms
seazean@Ubuntu1:~$
```

`icmp_seq`: ping 序列, 从1开始

`ttl`: IP 生存时间值

`time`: 响应时间, 数值越小, 联通速度越快

netstat

netstat 命令用于显示网络状态

```
netstat [-acCeFghilMnNoprstuvVwx] [-A<网络类型>] [--ip]
```

- -a 显示所有连线中的 Socket，显示详细的连接状况
- -i 显示网络界面信息表单，显示网卡列表
- -p 显示正在使用 Socket 的程序识别码和程序名称
- -n 显示使用 IP 地址，而不通过域名服务器
- -t 显示 TCP 传输协议的连线状况。
- -u 显示 UDP 传输协议的连线状况
- -aptn: 查看所有 TCP 开启端口
- -apun: 查看所有 UDP 开启端口

补充:

- netstat -apn | grep port: 查看指定端口号
- lsof -i:port : 查看指定端口号

磁盘管理

挂载概念

在安装 Linux 系统时设立各个分区，如根分区、/boot 分区等都是自动挂载的，也就是说不需要人为操作，开机就会自动挂载。但是光盘、U 盘等存储设备如果需要使用，就必须人为的进行挂载

在 Windows 下插入 U 盘也是需要挂载（分配盘符）的，只不过 Windows 下分配盘符是自动的。其实挂载可以理解为 Windows 当中的分配盘符，只不过 Windows 当中是以英文字母 ABCD 等作为盘符，而 Linux 是拿系统目录作为盘符，当然 Linux 当中也不叫盘符，而是称为挂载点，而把为分区或者光盘等存储设备分配一个挂载点的过程称为挂载

Linux 中的根目录以外的文件要想被访问，需要将其关联到根目录下的某个目录来实现，这种关联操作就是挂载，这个目录就是挂载点，解除次关联关系的过程称之为卸载

挂载点的目录需要以下几个要求：

- 目录要先存在，可以用 mkdir 命令新建目录
- 挂载点目录不可被其他进程使用到
- 挂载点下原有文件将被隐藏

lsblk

lsblk 命令的英文是 list block，即用于列出所有可用块设备的信息，而且还能显示他们之间的依赖关系，但是不会列出 RAM 盘的信息

命令：lsblk [参数]

- `lsblk`：以树状列出所有块设备

```
seazean@Ubuntu1:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0       7:0      0   55M  1 loop /snap/core18/1880
loop1       7:1      0 255.6M  1 loop /snap/gnome-3-34-1804/36
loop2       7:2      0  55.4M  1 loop /snap/core18/1944
loop3       7:3      0 217.9M  1 loop /snap/gnome-3-34-1804/60
loop4       7:4      0  62.1M  1 loop /snap/gtk-common-themes/1506
loop5       7:5      0  64.8M  1 loop /snap/gtk-common-themes/1514
loop6       7:6      0  49.8M  1 loop /snap/snap-store/467
loop7       7:7      0   51M   1 loop /snap/snap-store/518
loop8       7:8      0  31.1M  1 loop /snap/snapd/10492
loop9       7:9      0  29.9M  1 loop /snap/snapd/8542
sda         8:0      0   20G   0 disk
├─sda1      8:1      0   512M   0 part /boot/efi
├─sda2      8:2      0    1K   0 part
└─sda5      8:5      0  19.5G   0 part /
sr0        11:0     1   2.6G   0 rom  /media/seazean/Ubuntu 20.04.1 LTS amd64
```

NAME：这是块设备名

MAJ: MIN：本栏显示主要和次要设备号

RM：本栏显示设备是否可移动设备，在上面设备 sr0 的 RM 值等于 1，这说明他们是可移动设备

SIZE：本栏列出设备的容量大小信息

RO：该项表明设备是否为只读，在本案例中，所有设备的 RO 值为 0，表明他们不是只读的

TYPE：本栏显示块设备是否是磁盘或磁盘上的一个分区。在本例中，sda 和 sdb 是磁盘，而 sr0 是只读存储（rom）。

MOUNTPOINT：本栏指出设备挂载的挂载点。

- `lsblk -f`：不会列出所有空设备

```
seazean@Ubuntu1:~$ lsblk -f
NAME        FSTYPE LABEL          UUID                                FSAVAIL FSUSE% MOUNTPOINT
loop0       squashfs                                0      100% /snap/core18/1880
loop1       squashfs                                0      100% /snap/gnome-3-34-1804/36
loop2       squashfs                                0      100% /snap/core18/1944
loop3       squashfs                                0      100% /snap/gnome-3-34-1804/60
loop4       squashfs                                0      100% /snap/gtk-common-themes/1506
loop5       squashfs                                0      100% /snap/gtk-common-themes/1514
loop6       squashfs                                0      100% /snap/snap-store/467
loop7       squashfs                                0      100% /snap/snap-store/518
loop8       squashfs                                0      100% /snap/snapd/10492
loop9       squashfs                                0      100% /snap/snapd/8542
sda
├─sda1      vfat              739E-BCCD              511M     0% /boot/efi
├─sda2
└─sda5      ext4              7bccf826-c4ef-42d7-b23a-d5c5c4243e21 10.2G    41% /
sr0         iso9660 Ubuntu 20.04.1 LTS amd64 2020-07-31-16-51-12-00 0      100% /media/seazean/Ubuntu 20.04.1 LTS amd64
seazean@Ubuntu1:~$
```

NAME表示设备名称

FSTYPE表示文件类型

LABEL表示设备标签

UUID设备编号

MOUNTPOINT表示设备的挂载点

df

df 命令用于显示目前在 Linux 系统上的文件系统的磁盘使用情况统计。

命令：df [options]... [FILE]...

- -h 使用人类可读的格式(预设值是不加这个选项的...)
- --total 计算所有的数据之和

```
seazean@Ubuntu1:~$ df -h --total
文件系统 容量 已用 可用 已用% 挂载点
udev      953M  0  953M   0% /dev
tmpfs     196M  1.5M  195M   1% /run
/dev/sda5  20G  7.9G  11G  44% /
tmpfs     980M  0  980M   0% /dev/shm
tmpfs     5.0M  4.0K  5.0M   1% /run/lock
tmpfs     980M  0  980M   0% /sys/fs/cgroup
/dev/loop1 256M  256M   0 100% /snap/gnome-3-34-1804/36
/dev/loop0  55M   55M   0 100% /snap/core18/1880
/dev/loop2  56M   56M   0 100% /snap/core18/1944
/dev/loop3 218M  218M   0 100% /snap/gnome-3-34-1804/60
/dev/loop4  63M   63M   0 100% /snap/gtk-common-themes/1506
/dev/loop6  50M   50M   0 100% /snap/snap-store/467
/dev/loop5  65M   65M   0 100% /snap/gtk-common-themes/1514
/dev/loop9  30M   30M   0 100% /snap/snapd/8542
/dev/loop8  32M   32M   0 100% /snap/snapd/10492
/dev/loop7  52M   52M   0 100% /snap/snap-store/518
/dev/sda1  511M  4.0K  511M   1% /boot/efi
tmpfs     196M  32K  196M   1% /run/user/1000
/dev/sr0   2.6G  2.6G   0 100% /media/seazean/Ubuntu 20.04.1 LTS amd64
total     27G  12G  14G  45% -

seazean@Ubuntu1:~$ df -h
文件系统 容量 已用 可用 已用% 挂载点
udev      953M  0  953M   0% /dev
tmpfs     196M  1.5M  195M   1% /run
/dev/sda5  20G  7.9G  11G  44% /
tmpfs     980M  0  980M   0% /dev/shm
tmpfs     5.0M  4.0K  5.0M   1% /run/lock
tmpfs     980M  0  980M   0% /sys/fs/cgroup
/dev/loop1 256M  256M   0 100% /snap/gnome-3-34-1804/36
/dev/loop0  55M   55M   0 100% /snap/core18/1880
/dev/loop2  56M   56M   0 100% /snap/core18/1944
/dev/loop3 218M  218M   0 100% /snap/gnome-3-34-1804/60
/dev/loop4  63M   63M   0 100% /snap/gtk-common-themes/1506
/dev/loop6  50M   50M   0 100% /snap/snap-store/467
/dev/loop5  65M   65M   0 100% /snap/gtk-common-themes/1514
/dev/loop9  30M   30M   0 100% /snap/snapd/8542
/dev/loop8  32M   32M   0 100% /snap/snapd/10492
/dev/loop7  52M   52M   0 100% /snap/snap-store/518
/dev/sda1  511M  4.0K  511M   1% /boot/efi
tmpfs     196M  32K  196M   1% /run/user/1000
/dev/sr0   2.6G  2.6G   0 100% /media/seazean/Ubuntu 20.04.1 LTS amd64
seazean@Ubuntu1:~$
```

第一列指定文件系统的名称；第二列指定一个特定的文件系统，1K 是 1024 字节为单位的总容量；已用和可用列分别指定的容量；最后一个已用列指定使用的容量的百分比；最后一栏指定的文件系统的挂载点

mount

mount 命令是经常会使用到的命令，它用于挂载 Linux 系统外的文件

使用者权限：所有用户，设置级别的需要管理员

```
mount [-hv]
mount -a [-fFnrsvw] [-t vfstype]
mount [-fFnrsvw] [-o options [,...]] device | dir
mount [-fFnrsvw] [-t vfstype] [-o options] device dir
```

- -t: 指定档案系统的型态, 通常不必指定。mount 会自动选择正确的型态。

通过挂载的方式查看 Linux CD/DVD 光驱, 查看 ubuntu-20.04.1-desktop-amd64.iso 的文件

- 进入【虚拟机】--【设置】, 设置 CD/DVD 的内容, ubuntu-20.04.1-desktop-amd64.iso
- 创建挂载点 (注意: 一般用户无法挂载 cdrom, 只有 root 用户才可以操作)

`mkdir -p /mnt/cdrom`: 切换到 root 下创建一个挂载点 (其实就是创建一个目录)

- 开始挂载

`mount -t auto /dev/cdrom /mnt/cdrom`: 通过挂载点的方式查看上面的【ISO文件内容】

```
root@Ubuntu1:~/mnt# mount -t auto /dev/cdrom /mnt/cdrom
mount: /mnt/cdrom: WARNING: device write-protected, mounted read-only.
root@Ubuntu1:~/mnt#
```

- 查看挂载内容: `ls -l -a ./mnt/cdrom/`
- 卸载 cdrom: `umount /mnt/cdrom/`

防火墙

概述

防火墙技术是通过有机结合各类用于安全管理与筛选的软件和硬件设备, 帮助计算机网络于其内、外网之间构建一道相对隔绝的保护屏障, 以保护用户资料与信息安全性的一种技术。在默认情况下, Linux 系统的防火墙状态是打开的

状态

启动语法: `service name status`

- 查看防火墙状态: `service iptables status`
- 临时开启: `service iptables start`
- 临时关闭: `service iptables stop`
- 开机启动: `chkconfig iptables on`
- 开机关闭: `chkconfig iptables off`

放行

设置端口防火墙放行

- 修改配置文件: `vim /etc/sysconfig/iptables`
- 添加放行端口: `-A INPUT -m state --state NEW -m tcp -p tcp --dport 端口号 -j ACCEPT`
- 重新加载防火墙规则: `service iptables reload`

备注: 默认情况下 22 端口号是放行的

Shell

入门

概念

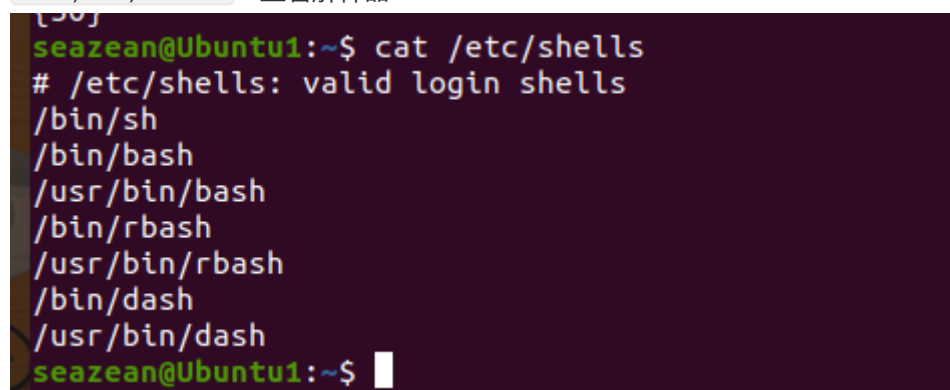
Shell 脚本 (shell script) , 是一种为 shell 编写的脚本程序, 又称 Shell 命令稿、程序化脚本, 是一种计算机程序使用的文本文件, 内容由一连串的 shell 命令组成, 经由 Unix Shell 直译其内容后运作

Shell 被当成是一种脚本语言来设计, 其运作方式与解释型语言相当, 由 Unix shell 扮演命令行解释器的角色, 在读取 shell 脚本之后, 依序运行其中的 shell 命令, 之后输出结果

环境

Shell 编程跟 JavaScript、php 编程一样, 只要有一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以了。

cat /etc/shells: 查看解释器



```
[30]
seazean@Ubuntu1:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
seazean@Ubuntu1:~$
```

Linux 的 Shell 种类众多, 常见的有:

- Bourne Shell (/usr/bin/sh或/bin/sh)
- Bourne Again Shell (/bin/bash) : Bash 是大多数Linux 系统默认的 Shell
- C Shell (/usr/bin/csh)
- K Shell (/usr/bin/ksh)
- Shell for Root (/sbin/sh)
- 等等.....

第一个shell

- 新建 s.sh 文件: touch s.sh
- 编辑 s.sh 文件: vim s.sh

```
#!/bin/bash --- 指定脚本解释器
echo "你好, shell !" ----向窗口输入文本

:<<!
写shell的习惯 第一行指定解释器
文件是sh为后缀名
括号成对书写
注释的时候尽量不用中文注释。不友好。
[] 括号两端要要有空格。 [ neirong ]
习惯代码索引, 增加阅读性
写语句的时候, 尽量写全了, 比如if。。。
!
```

- 查看 s.sh文件: `ls -l s.sh`文件权限是【-rw-rw-r--】
- `chmod a+x s.sh` s.sh文件权限是【-rwxrwxr-x】
- 执行文件: `./s.sh`
- 或者直接 `bash s.sh`

注意:

`#!` 是一个约定的标记, 告诉系统这个脚本需要什么解释器来执行, 即使用哪一种 Shell

`echo` 命令用于向窗口输出文本

注释

- 单行注释: 以 `#` 开头的行就是注释, 会被解释器忽略
- 多行注释:

```
:<<EOF
注释内容...
注释内容...
EOF
```

```
:<<! ----这里的符号要和结尾处的一样
注释内容...
注释内容...
注释内容...
!
```

变量

定义变量

变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命名须遵循如下规则：

- 命名只能使用英文字母，数字和下划线，首个字符不能以数字开头。
- 中间不能有空格，可以使用下划线（_）。
- 不能使用标点符号。
- 不能使用bash里的关键字（可用help命令查看保留关键字）。

使用变量

使用一个定义过的变量，只要在变量名前面加美元符号\$即可

```
name="seazean"
echo $name
echo ${name}
name="zhy"
```

- 已定义的变量，可以被重新定义变量名
- 外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界。推荐加！！

比如：echo "I am good at \${shell-t}Script"

通过上面的脚本我们发现，如果不给shell-t变量加花括号，写成echo "I am good at \$shell-tScript"，解释器shell就会把\$shell-tScript当成一个变量，由于我们前面没有定义shell-t变量，那么解释器执行执行的结果自然就为空了。

只读变量

使用 readonly 命令可以将变量定义为只读变量，只读变量的值不能被改变。(类似于final)

```
#!/bin/bash
myUrl="https://www.baidu.com"
readonly myUrl
myUrl="https://cn.bing.com/"
#报错 myUrl readonly
```


删除变量

使用 unset 命令可以删除变量，变量被删除后不能再次使用。

语法：unset variable_name

```
#!/bin/sh
myUrl="https://www.baidu.com"
unset myUrl
echo $myUrl
```

定义myUrl变量，通过unset删除变量，然后通过echo进行输出，**结果是空**，没有任何的结果输出。

字符变量

字符串是shell编程中最常用也是最有用的数据类型，字符串可以用单引号，也可以用双引号，也可以不用引号，在Java SE中我们定义一个字符串通过String s="abc" 双引号的形式进行定义，而在shell中也是可以的。

引号

- 单引号

```
str='this is a string variable'
```

单引号字符串的限制：

- 单引号里的任何字符都会原样输出，单引号字符串中的**变量是无效的**；
- 单引号字符串中不能出现单独一个的单引号（对单引号使用转义符后也不行），但可成对出现，作为字符串拼接使用。

- 双引号

```
your_name='frank'
str="Hello,\"$your_name\"! \n"
echo -e $str      #Hello, "frank"!
```

双引号的优点：

- 双引号里可以有变量
- 双引号里可以出现转义字符

拼接字符串

```
your_name="frank"
# 使用双引号拼接
greeting="hello, "$your_name" !"
greeting_1="hello, ${your_name} !"
echo $greeting $greeting_1
#hello,frank! hello,frank
```

获取字符串长度

命令: `${#variable_name}`

```
string="seazean"  
echo ${#string} #7
```

提取字符串

```
string="abcdefghijklmn"  
echo ${string:1:4}
```

输出为【bcde】，通过截取我们发现，它的下标和我们在java中的读取方式是一样的，下标也是从0开始。

数组

bash支持一维数组（不支持多维数组），并且没有限定数组的大小。

定义数组

在 Shell 中，用括号来表示数组，数组元素用"空格"符号分割开

```
数组名=(值1 值2 ... 值n)  
array_name=(value0 value1 value2 value3)  
array_name=(  
value0  
value1  
value2  
value3  
)
```

通过下标定义数组中的其中一个元素：

```
array_name[0]=value0  
array_name[1]=value1  
array_name[n]=valuen
```

可以不使用连续的下标，而且下标的范围没有限制

读取数组

读取数组元素值的一般格式是：

```
${数组名[下标]}

value=${array_name[n]}
echo ${value}
```

使用 @ 符号可以获取数组中的所有元素，例如：`echo ${array_name[@]}`

获取长度

获取数组长度的方法与获取字符串长度的方法相同，数组前加#

```
# 取得数组元素的个数
length=${#array_name[@]}
# 或者
length=${#array_name[*]}
```

```
#!/bin/bash
g=(a b c d e f)
echo "数组下标为2的数据为:" ${g[2]} #c
echo "数组所有数据为:" ${#g[@]} #6
echo "数组所有数据为:" ${#g[*]} #6
```

运算符

Shell 和其他编程一样，**支持**包括：算术、关系、布尔、字符串等运算符。原生 bash **不支持** 简单的数学运算，但是可以通过其他命令来实现，例如expr。expr 是一款表达式计算工具，使用它能完成表达式的求值操作。

规则

- **表达式和运算符之间要有空格**，例如 2+2 是不对的，必须写成 2 + 2
- **完整的表达式要被 `` 包含，注意不是单引号**
- **条件表达式要放在方括号之间，并且要有空格**，例如：`[$a==$b]` 是错误的，必须写成 `[$a == $b]`
- **(()) 双括号里可以跟表达式**，例如 `((i++))`，`((a+b))`

算术运算符

运算符	说明	举例
+	加法	<code>expr \$a + \$b</code> 结果为 30。
-	减法	<code>expr \$a - \$b</code> 结果为 -10。
*	乘法	<code>expr \$a * \$b</code> 结果为 200。
/	除法	<code>expr \$b / \$a</code> 结果为 2。
%	取余	<code>expr \$b % \$a</code> 结果为 0。
=	赋值	<code>a=\$b</code> 将把变量 b 的值赋给 a。
==	相等。用于比较两个数字，相同则返回 true。	<code>[\$a == \$b]</code> 返回 false。
!=	不相等。用于比较两个数字，不相同则返回 true。	<code>[\$a != \$b]</code> 返回 true。

```
#!/bin/bash
a=4
b=20
echo "加法运算" `expr $a + $b`
echo "乘法运算，注意*号前面需要反斜杠" `expr $a \* $b`
echo "加法运算" `expr $b / $a`
((a++))
echo "a = $a"
c=$((a + b))
d=$((a + b))
echo "c = $c"
echo "d = $d"
```

```
//结果
加法运算 24
减法运算 -16
乘法运算，注意*号前面需要反斜杠 80
加法运算 5
a = 5
c = 25
d = 25
```

字符运算符

假定变量 a 为 "abc", 变量 b 为 "efg", true=0, false=1。

运算符	说明	举例
=	检测两个字符串是否相等，相等返回 true。	[\$a = \$b] 返回 false。
!=	检测两个字符串是否相等，不相等返回 true。	[\$a != \$b] 返回 true。
-z	检测字符串长度是否为0，为0返回 true。	[-z \$a] 返回 false。
-n	检测字符串长度是否为0，不为0返回 true。	[-n "\$a"] 返回 true。
\$	检测字符串是否为空，不为空返回 true。	[\$a] 返回 true。

```
a="abc"
b="efg"

if [ $a = $b ]
then
    echo "$a = $b : a 等于 b"
else
    echo "$a = $b: a 不等于 b"
fi
if [ $a != $b ]
then
    echo "$a != $b : a 不等于 b"
else
    echo "$a != $b: a 等于 b"
fi
```

关系运算符

关系运算符只支持数字，不支持字符串，除非字符串的值是数字。

下表列出了常用的关系运算符，假定变量 a 为 10，变量 b 为 20：

运算符	说明	举例
-eq	检测两个数是否相等，相等返回 true。	<code>[\$a -eq \$b]</code> 返回 false。
-ne	检测两个数是否不相等，不相等返回 true。	<code>[\$a -ne \$b]</code> 返回 true。
-gt	检测左边的数是否大于右边的，如果是，则返回 true。	<code>[\$a -gt \$b]</code> 返回 false。
-lt	检测左边的数是否小于右边的，如果是，则返回 true。	<code>[\$a -lt \$b]</code> 返回 true。
-ge	检测左边的数是否大于等于右边的，如果是，则返回 true。	<code>[\$a -ge \$b]</code> 返回 false。
-le	检测左边的数是否小于等于右边的，如果是，则返回 true。	<code>[\$a -le \$b]</code> 返回 true。

```
a=10
b=20

if [ $a -eq $b ]
then
    echo "$a -eq $b : a 等于 b"
else
    echo "$a -eq $b: a 不等于 b"
fi
```

布尔运算符

下表列出了常用的布尔运算符，假定变量 a 为 10，变量 b 为 20：

运算符	说明	举例
!	非运算，表达式为 true 则返回 false，否则返回 true。	<code>[! false]</code> 返回 true。
-o	或运算，有一个表达式为 true 则返回 true。	<code>[\$a -lt 20 -o \$b -gt 100]</code> true
-a	与运算，两个表达式都为 true 才返回 true。	<code>[\$a -lt 20 -a \$b -gt 100]</code> false

逻辑运算符

假定变量 a 为 10, 变量 b 为 20:

运算符	说明	举例
&&	逻辑的 AND	<code>[[\$a -lt 100 && \$b -gt 100]]</code> 返回 false
	逻辑的 OR	<code>[[\$a -lt 100 \$b -gt 100]]</code> 返回 true

流程控制

if

```
if condition
then
    command1
    command2
    ...
    commandN
fi
#末尾的fi就是if倒过来拼写
```

```
if condition
then
    command1
    command2
    ...
    commandN
else
    command
fi
```

```
if condition1
then
    command1
elif condition2
then
    command2
else
    commandN
fi
```

- 查找一个进程，如果进程存在就打印true

```
if [ $(ps -ef | grep -c "ssh") -gt 1 ]
then
    echo "true"
fi
```

- 判断两个变量是否相等

```
a=10
b=20
if [ $a == $b ]
then
    echo "a 等于 b"
elif [ $a -gt $b ]
then
    echo "a 大于 b"
elif [ $a -lt $b ]
then
    echo "a 小于 b"
else
    echo "没有符合条件的"
fi
```

for

for循环格式为：

```
for var in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done
```

顺序输出当前列表中的字母：

```
for loop in A B C D E F G
do
    echo "顺序输出字母为：$loop"
done
```

```
顺序输出字母为:A
顺序输出字母为:B
....
顺序输出字母为:G
```


while

while循环用于不断执行一系列命令，也用于从输入文件中读取数据

```
while condition
do
    command
done
```

需求：如果int小于等于10，那么条件返回真。int从0开始，每次循环处理时，int加1。

```
#!/bin/bash
a=1
while [ "${a}" -le 10 ]
do
    echo "输出的值为: " $a
    ((a++))
done
输出的值为: 1
输出的值为: 2
...
输出的值为: 10
```

case...esac

与 switch ... case 语句类似，是一种多分支选择结构，每个 case 分支用右圆括号开始，用两个分号 ;; 表示 break，即执行结束，跳出整个 case ... esac 语句，esac（就是 case 反过来）作为结束标记。

```
case 值 in
模式1)
    command1
    command2
    command3
    ;;
模式2)
    command1
    command2
    command3
    ;;
*)
    command1
    command2
    command3
    ;;
esac #case反过来
```

- case 后为取值，值可以为变量或常数。
- 值后为关键字 in，接下来是匹配的各种模式，每一模式最后必须以右括号结束，模式支持正则表达式。

```
v="czbk"

case "$v" in
"czbk")
    echo "传智播客"
    ;;
"baidu")
    echo "baidu 搜索"
    ;;
"google")
    echo "google 搜索"
    ;;
esac
```

函数

输入

函数语法如下：

```
[ function ] funname [()]
{
    action;
    [return int;]
}
```

- 1、可以使用function fun() 定义函数，也可以直接fun() 定义,不带任何参数。
- 2、函数参数返回，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。return后跟数值n(0-255)

```
#无参无返回值的方法
method(){
    echo "函数执行了!"
}

#方法的调用
#method

#有参无返回值的方法
method2(){
    echo "接收到的第一个参数$1"
    echo "接收到的第二个参数$2"
}
```

```
#方法的调用
#method2 1 2

#有参有返回值方法的定义
method3(){
    echo "接收到的第一个参数$1"
    echo "接收到的第二个参数$2"
    return $(( $1 + $2 ))
}

#方法的调用
method3 10 20
echo $?
```

读取

`read` 变量名 --- 表示把键盘录入的数据复制给这个变量

需求：在方法中键盘录入两个整数,返回这两个整数的和

```
method(){
    echo "请录入第一个数"
    read number1
    echo "请录入第二个数"
    read number2
    echo "两个数字分别为${number1},${number2}"
    return $((number1+number2))
}

method
echo $?
```

Docker

基本概述

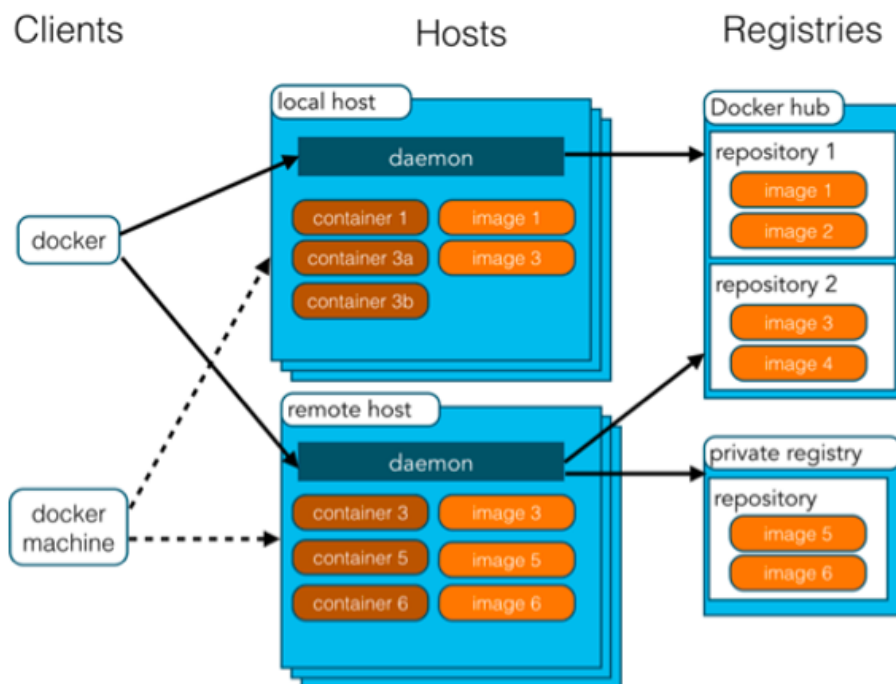
Docker 是一个开源的应用容器引擎，诞生于 2013 年初，基于 Go 语言实现，dotCloud 公司出品

Docker 让开发者打包开发应用以及依赖包到一个轻量级、可移植的容器中，可以发布到任何Linux机器上

- 容器是完全使用沙箱机制，相互隔离
- 容器性能开销极低。

Docker 架构：

- **镜像 (Image)**：Docker 镜像，就相当于一个 root 文件系统。比如官方镜像 ubuntu:16.04 就包含了完整的一套 Ubuntu16.04 最小系统的 root 文件系统
- **容器 (Container)**：镜像 (Image) 和容器 (Container) 的关系，就像是面向对象程序设计中的类和对象一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等
- **仓库 (Repository)**：仓库可看成一个代码控制中心，用来保存镜像



安装步骤：

```
# step 1: 安装必要的一些系统工具
sudo apt-get update
sudo apt-get -y install apt-transport-https ca-certificates curl software-properties-common
# step 2: 安装GPG证书
curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
# step 3: 写入软件源信息
sudo add-apt-repository "deb [arch=amd64] https://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb_release -cs) stable"
# step 4: 更新并安装Docker-CE
sudo apt-get -y update
sudo apt-get -y install docker-ce
```

配置镜像加速器：

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<- 'EOF'
{
  "registry-mirrors": ["https://hicqe4pi.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

操作命令

进程相关

- 启动docker服务:

```
systemctl start docker
```

- 停止docker服务:

```
systemctl stop docker
```

- 重启docker服务:

```
systemctl restart docker
```

- 查看docker服务状态:

```
systemctl status docker
```

- 设置开机启动docker服务:

```
systemctl enable docker
```

镜像相关

- 查看镜像: 查看本地所有的镜像

```
docker images
docker images -q # 查看所用镜像的id
```

- 搜索镜像: 从网络中查找需要的镜像

```
docker search 镜像名称
```

- 拉取镜像：从Docker仓库下载镜像到本地，镜像名称格式为 名称:版本号，如果版本号不指定则是最新的版本。如果不知道镜像版本，可以去docker hub 搜索对应镜像查看

```
docker pull 镜像名称
```

- 删除镜像：删除本地镜像

```
docker rmi 镜像id # 删除指定本地镜像  
docker rmi `docker images -q` # 删除所有本地镜像 tab上面的键
```

容器相关

- 查看容器：

```
docker ps # 查看正在运行的容器  
docker ps -a # 查看所有容器
```

- 创建并启动容器：

```
docker run 参数 --name=... /bin/bash
```

参数说明：

- -i：保持容器运行，通常与 -t 同时使用，加入it这两个参数后，容器创建后自动进入容器中，退出容器后，容器自动关闭
 - -t：为容器重新分配一个伪输入终端，通常与 -i 同时使用
 - -d：以守护（后台）模式运行容器。创建一个容器在后台运行，需要使用docker exec 进入容器。退出后，容器不会关闭
 - -it 创建的容器一般称为交互式容器，-id 创建的容器一般称为守护式容器
 - --name：为创建的容器命名
- 进入容器：

```
docker exec 参数 # 退出容器，容器不会关闭
```

- 停止容器：

```
docker stop 容器名称
```

- 启动容器：

```
docker start 容器名称
```

- 删除容器：如果容器是运行状态则删除失败，需要停止容器才能删除

```
docker rm 容器名称
```

- 查看容器信息:

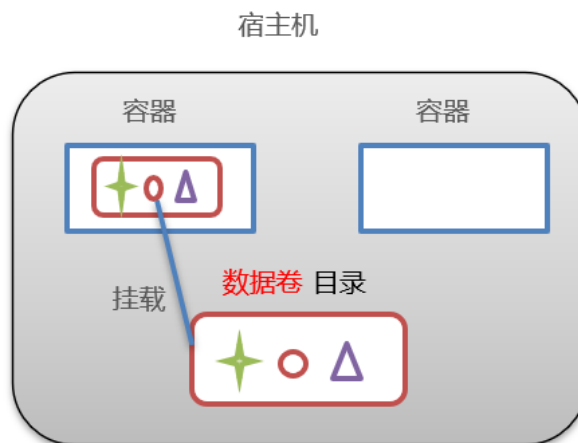
```
docker inspect 容器名称
```

数据卷

Docker 容器删除后，在容器中产生的数据也会随之销毁

Docker 容器和外部机器可以直接交换文件吗？

容器之间想要进行数据交互？



数据卷：数据卷是宿主机中的一个目录或文件，当容器目录和数据卷目录绑定后，对方的修改会立即同步

- 一个数据卷可以被多个容器同时挂载
- 一个容器也可以被挂载多个数据卷

数据卷的作用：

- 容器数据持久化
- 外部机器和容器间接通信
- 容器之间数据交换

配置数据卷

- 创建启动容器时，使用-v参数设置数据卷

```
docker run ... -v 宿主机目录(文件):容器内目录(文件) ...  
docker run -it --name=c1 -v /root(or~/)/data:/root/data_container centos:7
```

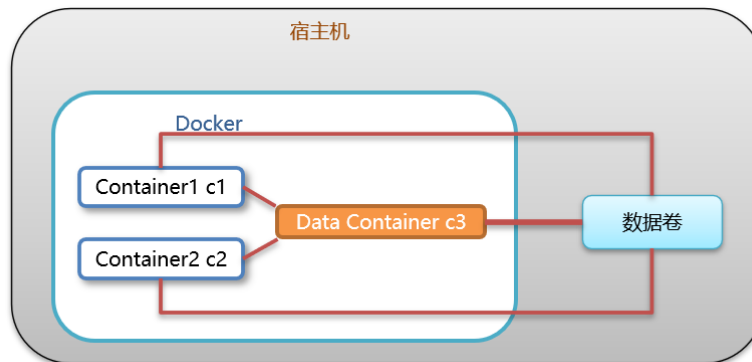
注意事项：

1. 目录必须是绝对路径
2. 如果目录不存在，会自动创建

3. 可以挂载多个数据卷

多容器进行数据交换：

- 多个容器挂载同一个数据卷
- 数据卷容器



- 创建启动c3数据卷容器，使用 -v 参数设置数据卷

```
docker run -it --name=c3 -v /volume centos:7 /bin/bash
```

- 创建启动 c1 c2 容器，使用 --volumes-from 参数设置数据卷

```
docker run -it --name=c1 --volumes-from c3 centos:7 /bin/bash
docker run -it --name=c2 --volumes-from c3 centos:7 /bin/bash
```

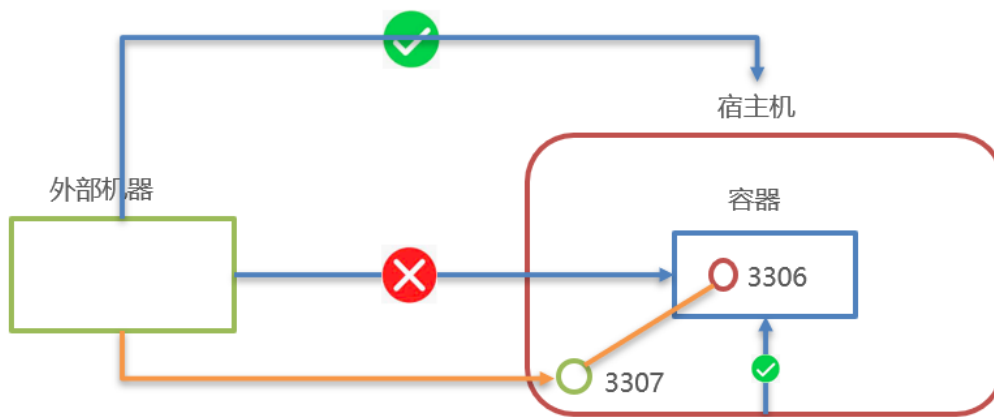
应用部署

MySQL

在Docker容器中部署MySQL，通过外部mysql客户端操作MySQL Server

端口映射：

- 容器内的网络服务和外部机器不能直接通信，外部机器和宿主机可以直接通信，宿主机和容器可以直接通信
- 当容器中的网络服务需要被外部机器访问时，可以将容器中提供服务的端口映射到宿主机的端口上。外部机器访问宿主机的该端口，从而间接访问容器的服务。这种操作称为：**端口映射**



MySQL部署步骤：搜索mysql镜像，拉取mysql镜像，创建容器，操作容器中的mysql

1. 搜索mysql镜像

```
docker search mysql
```

2. 拉取mysql镜像

```
docker pull mysql:5.6
```

3. 创建容器，设置端口映射、目录映射

```
# 在/root目录下创建mysql目录用于存储mysql数据信息
mkdir ~/mysql
cd ~/mysql
```

```
docker run -id \
-p 3307:3306 \
--name=c_mysql \
-v $PWD/conf:/etc/mysql/conf.d \
-v $PWD/logs:/logs \
-v $PWD/data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456 \
mysql:5.6
```

参数说明：

- `-p 3307:3306`：将容器的 3306 端口映射到宿主机的 3307 端口
- `-v $PWD/conf:/etc/mysql/conf.d`：将主机当前目录下的 conf/my.cnf 挂载到容器的 /etc/mysql/my.cnf，配置目录
- `-v $PWD/logs:/logs`：将主机当前目录下的 logs目录挂载到容器的 /logs，日志目录
- `-v $PWD/data:/var/lib/mysql`：将主机当前目录下的 data目录挂载到容器的 /var/lib/mysql。数据目录
- `-e MYSQL_ROOT_PASSWORD=123456`：初始化 root 用户的密码。

4. 进入容器，操作mysql

```
docker exec -it c_mysql /bin/bash
```

5. 使用外部机器连接容器中的mysql

Tomcat

1. 搜索tomcat镜像

```
docker search tomcat
```

2. 拉取tomcat镜像

```
docker pull tomcat
```

3. 创建容器，设置端口映射、目录映射

```
# 在/root目录下创建tomcat目录用于存储tomcat数据信息  
mkdir ~/tomcat  
cd ~/tomcat
```

```
docker run -id --name=c_tomcat \  
-p 8080:8080 \  
-v $PWD:/usr/local/tomcat/webapps \  
tomcat
```

参数说明：

- `-p 8080:8080`：将容器的8080端口映射到主机的8080端口
- `-v $PWD:/usr/local/tomcat/webapps`：将主机中当前目录挂载到容器的webapps

4. 使用外部机器访问tomcat

Nginx

1. 搜索nginx镜像

```
docker search nginx
```

2. 拉取nginx镜像

```
docker pull nginx
```

3. 创建容器，设置端口映射、目录映射

```
# 在/root目录下创建nginx目录用于存储nginx数据信息
mkdir ~/nginx
cd ~/nginx
mkdir conf
cd conf
# 在~/nginx/conf/下创建nginx.conf文件,粘贴下面内容
vim nginx.conf
```

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include        /etc/nginx/mime.types;
    default_type   application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

```
docker run -id --name=c_nginx \
-p 80:80 \
-v $PWD/conf/nginx.conf:/etc/nginx/nginx.conf \
-v $PWD/logs:/var/log/nginx \
-v $PWD/html:/usr/share/nginx/html \
nginx
```

参数说明:

- `-p 80:80`: 将容器的 80 端口映射到宿主机的 80 端口
- `-v $PWD/conf/nginx.conf:/etc/nginx/nginx.conf`: 将主机当前目录下的 /conf/nginx.conf 挂载到容器的 :/etc/nginx/nginx.conf, 配置目录

- `-v $PWD/logs:/var/log/nginx`：将主机当前目录下的 logs 目录挂载到容器的 /var/log/nginx，日志目录
4. 使用外部机器访问nginx
-

Redis

1. 搜索redis镜像

```
docker search redis
```

2. 拉取redis镜像

```
docker pull redis:5.0
```

3. 创建容器，设置端口映射

```
docker run -id --name=c_redis -p 6379:6379 redis:5.0
```

4. 使用外部机器连接redis

```
./redis-cli.exe -h 192.168.149.135 -p 6379
```

镜像原理

底层原理

Docker 镜像本质是什么？

Docker 中一个centos镜像为什么只有200MB，而一个centos操作系统的iso文件要几个G？

Docker 中一个tomcat镜像为什么有500MB，而一个tomcat安装包只有70多MB？

操作系统的组成部分：进程调度子系统、进程通信子系统、内存管理子系统、设备管理子系统、文件管理子系统、网络通信子系统、作业控制子系统

Linux文件系统由bootfs和rootfs两部分组成：

- bootfs：包含bootloader（引导加载程序）和 kernel（内核）
- rootfs：root文件系统，包含的就是典型 Linux 系统中的 /dev, /proc, /bin, /etc等标准目录和文件
- 不同的linux发行版，bootfs基本一样，而rootfs不同，如ubuntu, centos

Docker镜像原理：

- Docker镜像是一个**分层文件系统**，是由特殊的文件系统叠加而成，最底端是 bootfs，并复用宿主机的bootfs，第二层是 root文件系统rootfs称为base image，然后再往上可以叠加其他的镜像文件
- 统一文件系统（Union File System）技术能够将不同的层整合成一个文件系统，为这些层提供了一个统一的视角，这样就隐藏了多层的存在，在用户的角度来看，只存在一个文件系统
- 一个镜像可以放在另一个镜像的上面。位于下面的镜像称为父镜像，最底部的镜像成为基础镜像。
- 当从一个镜像启动容器时，Docker会在最顶层加载一个读写文件系统作为容器

问题：

- Docker 中一个Ubuntu镜像为什么只有200MB，而一个Ubuntu操作系统的iso文件要几个G？
Ubuntu的iso镜像文件包含bootfs和rootfs，而docker的Ubuntu镜像复用操作系统的bootfs，只有rootfs和其他镜像层
- Docker 中一个tomcat镜像为什么有500MB，而一个tomcat安装包只有70多MB？
由于docker中镜像是分层的，tomcat虽然只有70多MB，但他需要依赖于父镜像和基础镜像，整个对外暴露的tomcat镜像大小500多MB

镜像制作

镜像制作

Docker 镜像如何制作？

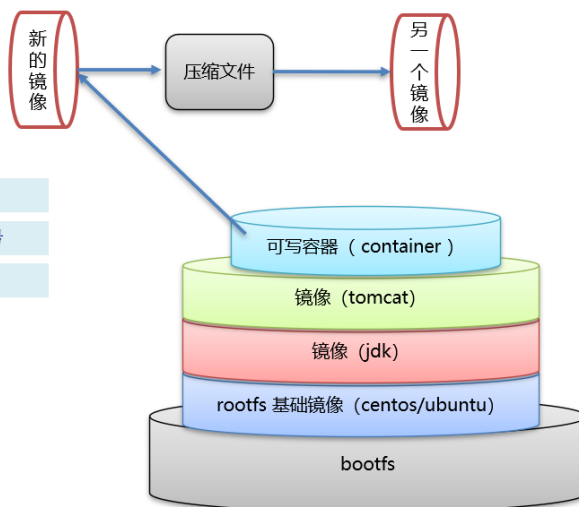
1. 容器转为镜像

```
docker commit 容器id 镜像名称:版本号
```

```
docker save -o 压缩文件名称 镜像名称:版本号
```

```
docker load -i 压缩文件名称
```

2. dockerfile



Dockerfile

基本概述

Dockerfile是一个文本文件，包含一条条的指令，每一条指令构建一层，基于基础镜像最终构建出新的镜像

- 对于开发人员：可以为开发团队提供一个完全一致的开发环境
- 对于测试人员：可以直接拿开发时所构建的镜像或者通过Dockerfile文件构建一个新的镜像开始工作了
- 对于运维人员：在部署时，可以实现应用的无缝移植

关键字	作用	备注
FROM	指定父镜像	指定dockerfile基于那个image构建
MAINTAINER	作者信息	用来标明这个dockerfile谁写的
LABEL	标签	用来标明dockerfile的标签 可以使用Label代替Maintainer 最终都是在docker image基本信息中可以查看
RUN	执行命令	执行一段命令 默认是/bin/sh 格式: RUN command 或者 RUN ["command" , "param1","param2"]
CMD	容器启动命令	提供启动容器时候的默认命令 和ENTRYPOINT配合使用.格式 CMD command param1 param2 或者 CMD ["command" , "param1","param2"]
ENTRYPOINT	入口	一般在制作一些执行就关闭的容器中会使用
COPY	复制文件	build的时候复制文件到image中
ADD	添加文件	build的时候添加文件到image中 不仅仅局限于当前build上下文可以来源于远程服务
ENV	环境变量	指定build时候的环境变量 可以在启动的容器的时候 通过-e覆盖 格式ENV name=value
ARG	构建参数	构建参数 只在构建的时候使用的参数 如果有ENV 那么ENV的相同名字的值始终覆盖arg的参数
VOLUME	定义外部可以挂载的数据卷	指定build的image那些目录可以启动的时候挂载到文件系统中 启动容器的时候使用 -v 绑定 格式 VOLUME ["目录"]
EXPOSE	暴露端口	定义容器运行的时候监听的端口 启动容器的使用-p来绑定暴露端口 格式: EXPOSE 8080 或者 EXPOSE 8080/udp
WORKDIR	工作目录	指定容器内部的工作目录 如果没有创建则自动创建 如果指定/使用的是绝对地址 如果不是/开头那么是在上一条workdir的路径的相对路径
USER	指定执行用户	指定build或者启动的时候 用户 在RUN CMD ENTRYPOINT执行的时候的用户
HEALTHCHECK	健康检查	指定监测当前容器的健康监测的命令 基本上没用 因为很多时候应用本身有健康监测机制
ONBUILD	触发器	当存在ONBUILD关键字的镜像作为基础镜像的时候 当执行FROM完成之后 会执行 ONBUILD的命令 但是不影响当前镜像用处也不怎么大
STOPSIGNAL	发送信号量到宿主机	该STOPSIGNAL指令设置将发送到容器的系统调用信号以退出。

关键字	作用	备注
SHELL	指定执行脚本的shell	指定RUN CMD ENTRYPOINT 执行命令的时候 使用的shell

Centos

自定义centos7镜像：

1. 默认登录路径为 /usr
2. 可以使用vim

实现步骤：

1. 定义父镜像：FROM centos:7
2. 定义作者信息：MAINTAINER seazean <zhyzhyang@sina.com>
3. 执行安装vim命令：RUN yum install -y vim
4. 定义默认的工作目录：WORKDIR /usr
5. 定义容器启动执行的命令：CMD /bin/bash
6. 通过dockerfile构建镜像：docker bulid -f dockerfile文件路径 -t 镜像名称:版本

Boot

定义dockerfile，发布springboot项目：

实现步骤：

1. 定义父镜像：FROM java:8
2. 定义作者信息：MAINTAINER seazean <zhyzhyang@sina.com>
3. 将jar包添加到容器：ADD springboot.jar app.jar
4. 定义容器启动执行的命令：CMD java-jar app.jar
5. 通过dockerfile构建镜像：docker bulid -f dockerfile文件路径 -t 镜像名称:版本

服务编排

基本介绍

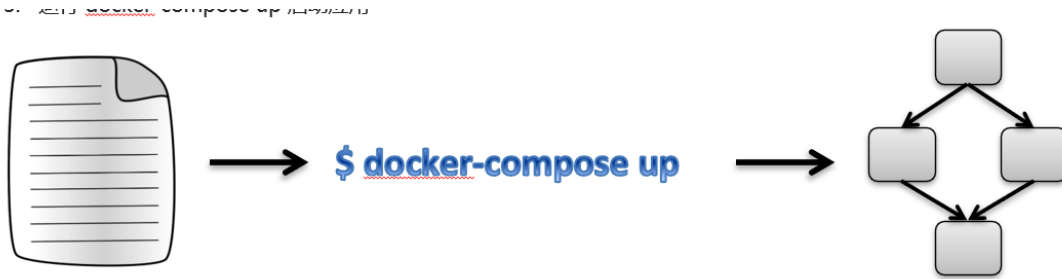
微服务架构的应用系统中一般包含若干个微服务，每个微服务一般都会部署多个实例，如果每个微服务都要手动启停，维护的工作量会很大。

- 从Dockerfile build image 或者去dockerhub拉取image；
- 创建多个container，管理这些container（启动停止删除）

服务编排：按照一定的业务规则批量管理容器

Docker Compose是一个编排多容器分布式部署的工具，提供命令集管理容器化应用的完整开发周期，包括服务构建，启动和停止。使用步骤：

1. 利用 Dockerfile 定义运行环境镜像
2. 使用 docker-compose.yml 定义组成应用的各服务
3. 运行 docker-compose up 启动应用



功能实现

使用docker compose编排nginx+springboot项目

1. 安装Docker Compose
2. 创建docker-compose目录

```
mkdir ~/docker-compose
cd ~/docker-compose
```

3. 编写 docker-compose.yml 文件

```
version: '3'
services:
  nginx:
    image: nginx
    ports:
      - 80:80
    links:
      - app
    volumes:
      - ./nginx/conf.d:/etc/nginx/conf.d
  app:
    image: app
    expose:
      - "8080"
```

4. 创建./nginx/conf.d目录

```
mkdir -p ./nginx/conf.d
```

5. 在./nginx/conf.d目录下编写***.conf文件


```
server {
    listen 80;
    access_log off;

    location / {
        proxy_pass http://app:8080;
    }
}
```

6. 在~/docker-compose 目录下使用docker-compose启动容器

```
docker-compose up
```

7. 测试访问

```
http://192.168.0.137/hello
```

私有仓库

Docker官方的Docker hub (<https://hub.docker.com>) 是一个用于管理公共镜像的仓库，我们可以从上面拉取镜像 到本地，也可以把我们自己的镜像推送上去。但是当服务器无法访问互联网，或者不希望将自己的镜像放到公网当中，那么我们就需要搭建自己的私有仓库来存储和管理自己的镜像

- 私有仓库搭建

```
# 1、拉取私有仓库镜像
docker pull registry
# 2、启动私有仓库容器
docker run -id --name=registry -p 5000:5000 registry
# 3、输入地址http://私有仓库服务器ip:5000/v2/_catalog, 显示{"repositories": []}
# 4、修改daemon.json
vim /etc/docker/daemon.json
# 在上述文件中添加一个key，保存退出。此步用于让 docker 信任私有仓库地址；注意将私有仓库服务器ip修改为自己私有仓库服务器真实ip
{"insecure-registries":["192.168.0.137:5000"]}
# 5、重启docker 服务
systemctl restart docker
docker start registry
```

- 将镜像上传至私有仓库

```
# 1、标记镜像为私有仓库的镜像
docker tag centos:7 私有仓库服务器IP:5000/centos:7

# 2、上传标记的镜像
docker push 私有仓库服务器IP:5000/centos:7
```

- 从私有仓库拉取镜像

```
#拉取镜像
docker pull 私有仓库服务器ip:5000/centos:7
```

虚拟机

容器：

- 容器是将软件打包成标准化单元，以用于开发、交付和部署
- 容器镜像是轻量的、可执行的独立软件包，包含软件运行所需的所有内容：代码、运行时环境、系统工具、系统库和设置
- 容器化软件在任何环境中都能够始终如一地运行。
- 容器赋予了软件独立性，使其免受外在环境差异的影响，从而有助于减少团队间在相同基础设施上运行不同软件时的冲突

容器和虚拟机对比：

- 相同：容器和虚拟机具有相似的资源隔离和分配优势
- 不同：
 - 容器虚拟化的是操作系统，虚拟机虚拟化的是硬件。
 - 传统虚拟机可以运行不同的操作系统，容器只能运行同一类型操作系统



特性	容器	虚拟机
启动	秒级	分钟
硬盘使用	一般为MB	一般为GB
性能	接近原生	弱于原生
系统支持量	单机支持上千个容器	一般几十个

