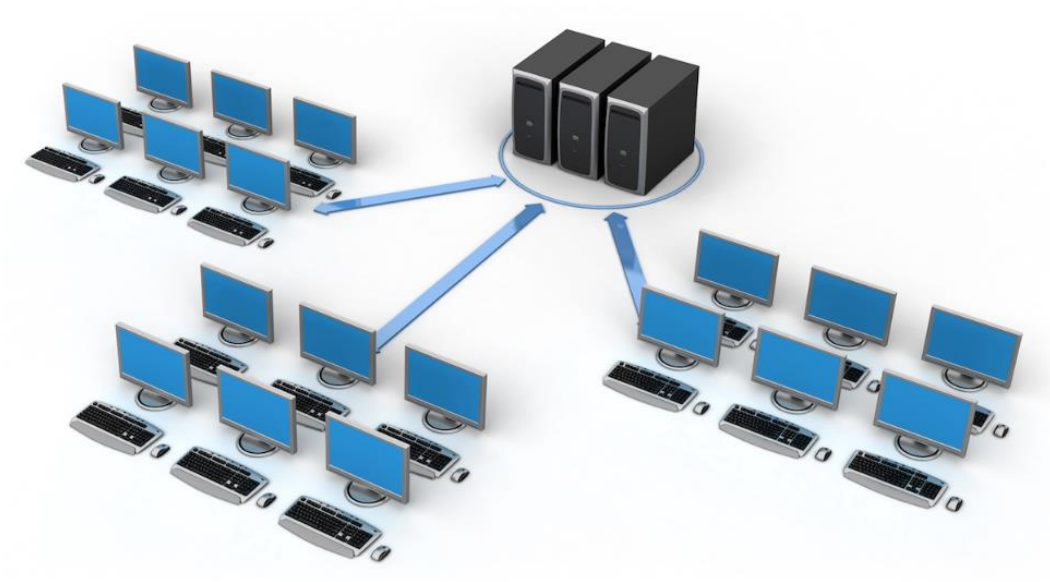
大数据技术之 linux

第一章：linux 系统简介与安装部署

1.1 发展背景

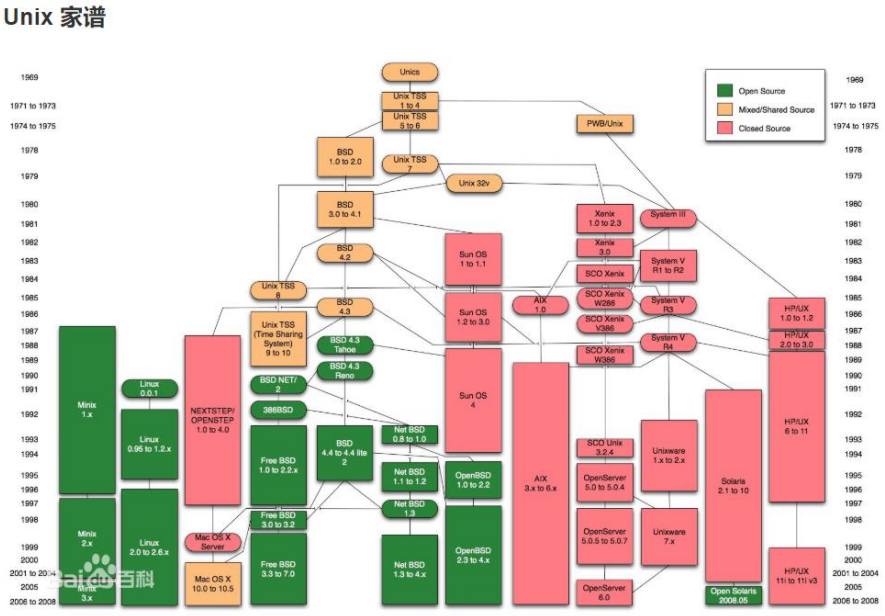


Linux 内核最初只是由芬兰人林纳斯·托瓦兹（Linus Torvalds）在赫尔辛基大学上学时出于个人爱好而编写的。Linux 是一套免费使用和自由传播的类 Unix 操作系统，是一个基于 POSIX 和 UNIX 的多用户、多任务、支持多线程和多 CPU 的操作系统。Linux 能运行主要的 UNIX 工具软件、应用程序和网络协议。它支持 32 位和 64 位硬件。Linux 继承 Unix 以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。

目前市面上较知名的发行版有：Ubuntu、RedHat、CentOS、Debian、Fedora、SuSE、OpenSUSE、Arch Linux、SolusOS 等，其中以 centos

在企业中使用最广。

现在企业使用的主流版本是 centos7 系列，随着国家国产化的的要求，一些国产操作系统也开始在一些国企央企以及军队项目中使用，比如中标麒麟，银河麒麟等国产化操作系统，在选择这些操作系统的时候要注意软件与 linux 内核版本的适用性，本质来说国产化操作系统还是基于 redhat 的内核。



1.2linux 与 windows 的区别

1.免费开源

2.模块化程度高

3.广泛的硬件支持

4.安全稳定

5.多用户，多任务

6.良好的可移植性

比较	Windows	Linux
界面	界面统一，外壳程序固定所有 Windows 程序菜单几乎一致，快捷键也几乎相同	图形界面风格依发布版不同而不同，可能互不兼容。GNU/Linux 的终端机是从 UNIX 传承下来，基本命令和操作方法也几乎一致。
驱动程序	驱动程序丰富，版本更新频繁。默认安装程序里面一般包含有该版本发布时流行的硬件驱动程序，之后所出的新硬件驱动依赖于硬件厂商提供。对于一些老硬件，如果没有了原配的驱动有时很难支持。另外，有时硬件厂商未提供所需版本的 Windows 下的驱动，也会比较头痛。	由志愿者开发，由 Linux 核心开发小组发布，很多硬件厂商基于版权考虑并未提供驱动程序，尽管多数无需手动安装，但是涉及安装则相对复杂，使得新用户面对驱动程序问题（是否存在和安装方法）会一筹莫展。但是在开源开发模式下，许多老硬件尽管在Windows下很难支持的也容易找到驱动。HP、Intel、AMD 等硬件厂商逐步不同程度支持开源驱动，问题正在得到缓解。
使用	使用比较简单，容易入门。图形化界面对没有计算机背景知识的用户使用十分有利。	图形界面使用简单，容易入门。文字界面，需要学习才能掌握。
学习	系统构造复杂、变化频繁，且知识、技能淘汰快，深入学习困难。	系统构造简单、稳定，且知识、技能传承性好，深入学习相对容易。
软件	每一种特定功能可能都需要商业软件的支持，需要购买相应的授权。	大部分软件都可以自由获取，同样功能的软件选择较少。

1.3Centos 下载地址

官网下载链接：http://isoredirect.centos.org/centos/7/isos/x86_64/

阿里云站点：http://mirrors.aliyun.com/centos/7/isos/x86_64/

1.4VM 与 linux 的安装

详情安装步骤见下面安装文档，双击可打开

vmware与centos7安装.docx

第二章：linux 文件与与目录结构

2.1 Linux 文件

Linux 系统中一切皆文件。

2.2 Linux 目录结构

/bin　二进制可执行命令
/dev　设备特殊文件
/etc　系统管理和配置文件
/etc/rc.d　启动的配置文件和脚本
/home　用户主目录的基点，比如用户 user 的主目录就是/home/user，可以用~user 表示
/lib　标准程序设计库，又叫动态链接共享库，作用类似 windows 里的.dll 文件
/sbin　超级管理命令，这里存放的是系统管理员使用的管理程序
/tmp　公共的临时文件存储点
/root　系统管理员的主目录
/mnt　系统提供这个目录是让用户临时挂载其他的文件系统
/lost+found 这个目录平时是空的，系统非正常关机而留下“无家可归”的文件（windows 下叫什么.chk）就在这里
/proc　虚拟的目录，是系统内存的映射。可直接访问这个目录来获取系统信息。
/var　某些大文件的溢出区，比方说各种服务的日志文件
/usr　最庞大的目录，要用到的应用程序和文件几乎都在这个目录

第三章：VI 与 VIM 编辑器

3.1vi 简介

vi 是“Visual interface”的简称，它在 Linux 上的地位就仿佛 Edit 程序在 DOS 上一样。它可以执行输出、删除、查找、替换、块操作等众多文本操作，而且用户可以根据自己的需要对其进行定制。Vi 不是一个排版程序，它不象 Word 或 WPS 那样可以对字体、格式、段落等其他属性进行编排，它只是一个文本编辑程序。 vi 没有菜单，只有命令，且命令繁多。

Vi 有三种基本工作模式：

+ 命令模式
+ 文本输入模式
+ 末行模式

命令行模式

任何时候，不管用户处于何种模式，只要按一下 ESC 键，即可使 Vi 进入命令模式；我们在 shell 环境(提示符为$)下输入启动 Vi 命令，进入编辑器时，也是处于该模式下。在该模式下，用户可以输入各种合法的 Vi 命令，用于管理自己的文档。此时从键盘上输入的任何字符都被当做编辑命令来解释，若输入的字符是合法的 Vi 命令，则 Vi 在

接受用户命令之后完成相应的动作。但需注意的是，所输入的命令并不在屏幕上显示出来。若输入的字符不是 Vi 的合法命令，Vi 会响铃报警。

文本输入模式

在命令模式下输入插入命令 i、附加命令 a 、打开命令 o、修改命令 c、取代命令 r 或替换命令 s 都可以进入文本输入模式。在该模式下，用户输入的任何字符都被 Vi 当做文件内容保存起来，并将其显示在屏幕上。在文本输入过程中，若想回到命令模式下，按键 ESC 即可。

末行模式

末行模式也称 ex 转义模式。在命令模式下，用户按“:”键即可进入末行模式下，此时 Vi 会在显示窗口的最后一行(通常也是屏幕的最后一行)显示一个“:”作为末行模式的提示符，等待用户输入命令。多数文件管理命令都是在此模式下执行的(如把编辑缓冲区的内容写到文件中等)。末行命令执行完后，Vi 自动回到命令模式。

3.2vim 基础操作

vim 是从 vi 发展出来的一个文本编辑器 。代码补完、编译及错误跳转等方便编程的功能特别丰富

进入插入模式:

i: 插入光标前一个字符

I: 插入行首

a: 插入光标后一个字符

A: 插入行未

o: 向下新开一行,插入行首

O: 向上新开一行,插入行首

进入末行模式

在命令模式下，用户按“:”键即可进入末行模式

退出

:q 退出

:q! 退出并不保存

:w 保存

:wq 保存并退出

:x 保存并退出

查找命令:

/: str 查找

n: 下一个

N：上一个

替换命令：

把 abc 全部替换成 123

末行模式下，将当前文件中的所有 abc 替换成 123。如果不使用 g，则只会替换每一行的第一个 abc

:%s/abc/123/g

末行模式下，将第一行至第 10 行之间的 abc 替换成 123

:1, 10s/abc/123/g

vim 里执行 shell 下命令:

末行模式里输入!,后面跟命令

进入命令模式:

ESC:从插入模式或末行模式进入命令模式

移动光标:

h: 左移

j: 下移

k: 上移

l: 右移

M: 光标移动到当前屏幕的中间行

L: 光标移动到当前屏幕最后一行行首

gg: 光标移动文件开头

G: 光标移动到文件末尾

G: 移动到指定行,行号 G (扩展：打开文件时跳转指定行,vi 文件名 + 行数)

w: 向后一次移动一个字

b: 向前一次移动一个字

{: 按段移动,上移

}: 按段移动,下移

Ctr-d: 向下翻半屏

Ctr-u: 向上翻半屏

Ctr-f: 向下翻一屏

Ctr-b: 向上翻一屏

撤销命令:

u: 一步一步撤销

Ctr-r: 反撤销

删除命令:

x: 删除光标后一个字符,相当于 Del

X: 删除光标前一个字符,相当于 Backspace

dd: 删除光标所在行,n dd 删除指定的行数

D: 删除光标后本行所有内容,包含光标所在字符

d0: 删除光标前本行所有内容,不包含光标所在字符

dw: 删除光标开始位置的字,包含光标所在字符

复制粘贴:

yy: 复制当前行,n yy 复制 n 行

p: 在光标所在位置向下新开辟一行,粘贴

替换操作:

r: 替换当前字符

R: 替换当前行光标后的字符

文本行移动:

>>: 文本行右移

<<: 文本行左移

重复命令:

.: 重复上一次操作的命令

可视模式:

v: 按字符移动,选中文本

V: 按行移动,选中文本可视模式可以配合 d, y, >>, << 实现对文本块的删除,复制,左右移动

3.3 终端实用技巧

vim 文件名

按 i 进入编辑，需要修改哪块按上下左右键可移动

修改完毕后按【ESC】键退出，需要保存退出按 :wq 不保存退出:q!

第四章 ：linux 常用基本命令

4.1 基础快捷键

ctrl + c	停止进程
ctrl+l	清屏；彻底清屏是：reset
ctrl + q	退出
善于用 tab 键补全	提示(更重要的是可以防止敲错)
上下键	查找执行过的命令
ctrl +alt	linux 和 Windows 之间切换

4.2 文件命令

#pwd 显示当前工作目录的绝对路径

#ls,ll,ll -a

#cd 进入到某一个目录下

#mkdir 创建文件夹

#rmdir 删除文件夹

#touch 创建文件

#cp 复制文件/文件夹

#cp -r 递归复制，多级目录

#mv 移动文件夹/重命名

#rm 删除文件

#rm -rf 强制删除文件不需要确认

#cat 查看文件

#more 查看文件

#less 查看文件

#echo 输出

#head 查看文件头部

#tail 查看文件末尾

#tail -f 实时查看文件末尾

#nl 文件带行号标准输出

> 覆盖

>> 追加

#ln -s 目标目录 软链接地址 创建软链接

rm -rf 软链接地址 删除软连接

ln -snf 新目标目录 软链接地址 修改软连接

history 查看已经执行过历史命令

文本处理类的命令：

wc

wc [option] [file]...

-l: 统计行数

-c: 统计字节数

-w；统计单词数

tr

tr: 转换字符或删除字符

tr '集合 1' '集合 2'

tr -d '字符集合'

cut

This is a test line.

-d 字符：指定分隔符

-f#: 指定要显示字段

单个数字：一个字段

逗号分隔的多个数字：指定多个离散字段

-：连续字段，如 3-5；

例子：

准备一个测试文件，内容如下：

```
> cat test.txt

Hello World! I am maqian!

I am now in guangzhou!

Today is so hot, but i still have a fever!

这里是一行中文。
```

cut -c

显示第 1 个和第 2 个字符

```
cut test.txt -c "1,2"
```

-d 和-f 选项

以空格分开每一行并输出第 1 个和第 3 个字段

```
cut test.txt -f "1,3" -d " "
```

sort

按字符进行比较

sort [option] file...

-f: 忽略字符大小写；

-n: 比较数值大小；

-t: 指定分隔符

-k: 指定分隔后进行比较字段

-u: 重复的行，只显示一次；

准备数据：

>cat sort.txt

Apple

Cut

Blue

Cut

Apple

排序：sort sort.txt

去重：sort -u sort.txt

uniq

移除重复的行

-c：显示每行重复的次数

-d：仅显示重复过的行

-u: 仅显示不曾重复的行

4.3 系统信息命令

#date	查看当前系统时间
#data -s	修改时间
#w	显示登陆用户
#uname -a	查看系统内核
#cat /proc/cpuinfo	查看 cpu 信息
#cat /proc/meminfo	查看内存信息

4.4 压缩/解压命令

#tar -xvf file.tar	解压.tar 结尾的
#tar -zxvf file.tar.gz	解压.tar.gz 文件
#tar -cf file.tar file	创建包含 files 的文件 file.tar
#gzip -d file.gz	将 file.gz 解压缩为 file

4.5 网络命令

#ping host(主机名)	网络是否连通
#ifconfig	查看本机 ip 等信息
#telnet ip 端口	查看端口是否占用
(没有这个命令执行 yum -y install telnet)	
#wget file	下载文件
#tcpdump tcp port 端口	抓包 tcp

#hostname　　查看主机名

4.6 权限命令

1.文件或目录权限包括：

序号	权限	英文	缩写	数字代号
01	读	read	r	4
02	写	write	w	2
03	执行	excute	x	1

2.文件权限命令：

#chmod　777　file　为所有用户添加读，写，执行权限

#chmod　755　file　为所有者添加 rwx 权限，为组和其他用户添加 rx 权限

3.文件所属用户和用户组权限命令：

#chown hadoop:hadoop　file　将 file 的用户和用户组都改为 hadoop

4.7 用户管理命令

useradd 用户名　　添加新用户

useradd -g 组名 用户名　给某个组创建用户

passwd 用户名　　设置用户密码

cat　/etc/passwd　　查看创建了那些用户

Su　用户名　　切换用户

userdel　用户名　　删除用户但保存用户主目录

userdel -r 用户名	用户和用户主目录，都删除
whoami	显示自身用户名称
who am i	显示登录用户的用户名
usermod -g	更改用户组 用户名

设置普通用户具有 root 权限，可以使用 sudo

1．添加 hadoop 用户，并对其设置密码。

[root@hadoop101 ~]#useradd hadoop

[root@hadoop101 ~]#passwd hadoop

2．修改配置文件

[root@hadoop101 ~]#vi /etc/sudoers

修改 /etc/sudoers 文件，找到下面一行(91 行)，在 root 下面添加一行，如下所示：

Allow root to run any commands anywhere

root ALL=(ALL) ALL

hadoopALL=(ALL) ALL

或者配置成采用 sudo 命令时，不需要输入密码

Allow root to run any commands anywhere

root ALL=(ALL) ALL

hadoopALL=(ALL) NOPASSWD:ALL

修改完毕，现在可以用 hadoop 帐号登录，然后用命令 sudo ，即可获得 root 权限进行操作。

4.8 用户组管理命令

groupadd 组名	添加组
groupdel 组名	删除组
groupmod -n 新组名 老组名	指定工作组的新组名
cat /etc/group	查看创建了哪些组

4.9 搜索查找命令

1. find 查找文件或者目录

常用：find / -name file	查找 /目录下 file 文件

2. grep 过滤查找及“|”管道符（详细使用见下一章中的 grep）

1.管道符，“|”，表示将前一个命令的处理结果输出传递给后面的命令处理

2.grep 常常跟在|的后面做过滤查找

3.反转 grep -v

4.示例

#查找某文件在第几行 ls | grep -n test

#查找某进程 ps -ef | grep PID

#查看日志中含有 errer cat file |grep error

3. which 查找命令

查找命令在那个目录下

1. 基本语法

which 命令

2．案例实操

which ll

4. find 详解

由于 find 具有强大的功能，所以它的选项也很多，其中大部分选项都值得我们花时间来了解一下。即使系统中含有网络文件系统(NFS)，find 命令在该文件系统中同样有效，只要你具有相应的权限。

在运行一个非常消耗资源的 find 命令时，很多人都倾向于把它放在后台执行，因为遍历一个大的文件系统可能会花费很长的时间(这里是指 30G 字节以上的文件系统)。

一、find 命令格式

1、find 命令的一般形式为；

find pathname -options [-print -exec -ok ...]

2、find 命令的参数；

pathname: find 命令所查找的目录路径。例如用.来表示当前目录，用/来表示系统根目录，递归查找。

-print： find 命令将匹配的文件输出到标准输出。

-exec： find 命令对匹配的文件执行该参数所给出的 shell 命令。相应命令的形式为'command' { } \;，注意{ }和\；之间的空格。

-ok： 和-exec 的作用相同，只不过以一种更为安全的模式来执行该参数所给出的 shell 命令，在执行每一个命令之前，都会给出提示，

让用户来确定是否执行。

find 命令选项

-name 按照文件名查找文件。

-perm 按照文件权限来查找文件。

-prune 使用这一选项可以使 find 命令不在当前指定的目录中查找，如果同时使用-depth 选项，那么-prune 将被 find 命令忽略。

-user 按照文件属主来查找文件。

-group 按照文件所属的组来查找文件。

-mtime -n +n 按照文件的更改时间来查找文件，-n 表示文件更改时间距现在 n 天以内，+n 表示文件更改时间距现在 n 天以前。find 命令还有-atime 和-ctime 选项，但它们都和-m time 选项。

-nogroup 查找无有效所属组的文件，即该文件所属的组在/etc/groups 中不存在。

-nouser 查找无有效属主的文件，即该文件的属主在/etc/passwd 中不存在。

-newer file1 ! file2 查找更改时间比文件 file1 新但比文件 file2 旧的文件。

-type 查找某一类型的文件，诸如：

b - 块设备文件。

d - 目录。

c - 字符设备文件。

p - 管道文件。

l- 符号链接文件。

f- 普通文件。

-size n：[c] 查找文件长度为 n 块的文件，带有 c 时表示文件长度以字节计。

-depth 在查找文件时，首先查找当前目录中的文件，然后再在其子目录中查找。

-fstype 查找位于某一类型文件系统中的文件，这些文件系统类型通常可以在配置文件/etc/fstab 中找到，该配置文件中包含了本系统中有关文件系统的信息。

-mount 在查找文件时不跨越文件系统 mount 点。

-follow 如果 find 命令遇到符号链接文件，就跟踪至链接所指向的文件。

另外,下面三个的区别:

-amin n 查找系统中最后 N 分钟访问的文件

-atime n 查找系统中最后 n*24 小时访问的文件

-cmin n 查找系统中最后 N 分钟被改变文件状态的文件

-ctime n 查找系统中最后 n*24 小时被改变文件状态的文件

-mmin n 查找系统中最后 N 分钟被改变文件数据的文件

-mtime n 查找系统中最后 n*24 小时被改变文件数据的文件

4、使用 exec 或 ok 来执行 shell 命令

使用 find 时，只要把想要的操作写在一个文件里，就可以用 exec 来配合 find 查找，很方便 在有些操作系统中只允许-exec 选项执行诸

如 ls 或 ls -l 这样的命令。大多数用户使用这一选项是为了查找旧文件并删除它们。建议在真正执行 rm 命令删除文件之前，最好先用 ls 命令看一下，确认它们是所要删除的文件。

exec 选项后面跟随着所要执行的命令或脚本，然后是一对儿{}，一个空格和一个\，最后是一个分号。为了使用 exec 选项，必须要同时使用 print 选项。如果验证一下 find 命令，会发现该命令只输出从当前路径起的相对路径及文件名。

例如：为了用 ls -l 命令列出所匹配到的文件，可以把 ls -l 命令放在 find 命令的-exec 选项中

```
# find . -type f -exec ls -l {} \;
```

上面的例子中，find 命令匹配到了当前目录下的所有普通文件，并在-exec 选项中使用 ls -l 命令将它们列出。

在/logs 目录中查找更改时间在 5 日以前的文件并删除它们：

```
$ find logs -type f -mtime +5 -exec rm {} \;
```

记住：在 shell 中用任何方式删除文件之前，应当先查看相应的文件，一定要小心！当使用诸如 mv 或 rm 命令时，可以使用-exec 选项的安全模式。它将在对每个匹配到的文件进行操作之前提示你。

在下面的例子中，find 命令在当前目录中查找所有文件名以.LOG 结尾、更改时间在 5 日以上的文件，并删除它们，只不过在删除之前先给出提示。

```
$ find . -name "*.conf"    -mtime +5 -ok rm {    } \;
< rm ... ./conf/httpd.conf > ? n
```

按 y 键删除文件，按 n 键不删除。

任何形式的命令都可以在-exec 选项中使用。

在下面的例子中我们使用 grep 命令。find 命令首先匹配所有文件名为“ passwd*”的文件，例如 passwd、passwd.old、passwd.bak，然后执行 grep 命令看看在这些文件中是否存在一个 itcast 用户。

```
# find /etc -name "passwd*" -exec grep "itcast" {    } \;


itcast:x:1000:1000::/home/itcast:/bin/bash
```

选项详解

1.使用 name 选项

文件名选项是 find 命令最常用的选项，要么单独使用该选项，要么和其他选项一起使用。

可以使用某种文件名模式来匹配文件，记住要用引号将文件名模式引起来。

不管当前路径是什么，如果想要在自己的根目录$HOME 中查找文件名符合*.txt 的文件，使用~作为 'pathname'参数，波浪号~代表了你的$HOME 目录。

```
$ find ~ -name "*.txt" -print
```

想要在当前目录及子目录中查找所有的‘ *.txt’文件，可以用：

```
$ find . -name "*.txt" -print
```

想要的当前目录及子目录中查找文件名以一个大写字母开头的文件，可以用：

```
$ find . -name "[A-Z]*" -print
```

想要在/etc 目录中查找文件名以 host 开头的文件，可以用：

```
$ find /etc -name "host*" -print
```

想要查找$HOME 目录中的文件，可以用：

```
$ find ~ -name "*" -print  或 find . -print
```

要想让系统高负荷运行，就从根目录开始查找所有的文件：

```
$ find / -name "*" -print
```

如果想在当前目录查找文件名以两个小写字母开头，跟着是两个数字，最后是.txt 的文件，下面的命令就能够返回例如名为 ax37.txt 的文件：

```
$find . -name "[a-z][a-z][0-9][0-9].txt" -print
```

2、用 perm 选项

按照文件权限模式用-perm 选项,按文件权限模式来查找文件的话。最好使用八进制的权限表示法。

如在当前目录下查找文件权限位为 755 的文件，即文件属主可以读、写、执行，其他用户可以读、执行的文件，可以用：

```
$ find . -perm 755 -print
```

还有一种表达方法：在八进制数字前面要加一个横杠-，表示都匹配，如-007 就相当于 777，-006 相当于 666

```
# ls -l
# find . -perm 006
# find . -perm -006


-perm mode:文件许可正好符合 mode
-perm +mode:文件许可部分符合 mode
-perm -mode:  文件许可完全符合 mode
```

3、忽略某个目录

如果在查找文件时希望忽略某个目录，因为你知道那个目录中没有你所要查找的文件，那么可以使用-prune 选项来指出需要忽略的目录。在使用-prune 选项时要当心，因为如果你同时使用了-depth 选项，那么-prune 选项就会被 find 命令忽略。

如果希望在/apps 目录下查找文件，但不希望在/apps/bin 目录下查找，可以用：

```
$ find /apps -path "/apps/bin" -prune -o -print
```

4、使用 find 查找文件的时候怎么避开某个文件目录

比如要在/home/itcast 目录下查找不在 dir1 子目录之内的所有文件

```
find /home/itcast -path "/home/itcast/dir1" -prune -o -print
```

避开多个文件夹

```
find /home \( -path /home/itcast/f1 -o -path /home/itcast/f2 \) -prune -o -print
```

注意(前的\,注意(后的空格。

5、使用 user 和 nouser 选项

按文件属主查找文件，如在$HOME 目录中查找文件属主为 itcast 的文件，可以用：

```
$ find ~ -user itcast -print
```

在/etc 目录下查找文件属主为 uucp 的文件：

```
$ find /etc -user uucp -print
```

为了查找属主帐户已经被删除的文件，可以使用-nouser 选项。这样就能够找到那些属主在/etc/passwd 文件中没有有效帐户的文件。在使用-nouser 选项时，不必给出用户名； find 命令能够为你完成相应的工作。

例如，希望在/home 目录下查找所有的这类文件，可以用：

```
$ find /home -nouser -print
```

6、使用 group 和 nogroup 选项

就像 user 和 nouser 选项一样，针对文件所属于的用户组， find 命令也具有同样的选项，为了在/apps 目录下查找属于 itcast 用户组的文件，可以用：

```
$ find /apps -group itcast -print
```

要查找没有有效所属用户组的所有文件，可以使用 nogroup 选项。下面的 find 命令从文件系统的根目录处查找这样的文件

```
$ find / -nogroup -print
```

7、按照更改时间或访问时间等查找文件

如果希望按照更改时间来查找文件，可以使用 mtime,atime 或 ctime 选项。如果系统突然没有可用空间了，很有可能某一个文件的长度在此期间增长迅速，这时就可以用 mtime 选项来查找这样的文件。

用减号-来限定更改时间在距今 n 日以内的文件，而用加号+来限定更改时间在距今 n 日以前的文件。

希望在系统根目录下查找更改时间在 5 日以内的文件，可以用：

```
$ find / -mtime -5 -print
```

为了在/var/adm 目录下查找更改时间在 3 日以前的文件，可以用：

```
$ find /var/adm -mtime +3 -print
```

8、查找比某个文件新或旧的文件

如果希望查找更改时间比某个文件新但比另一个文件旧的所有文件，可以使用-newer 选项。它的一般形式为：

```
newest_file_name ! oldest_file_name
```

其中，！是逻辑非符号。

9、使用 type 选项

在/etc 目录下查找所有的目录，可以用：

```
$ find /etc -type d -print
```

在当前目录下查找除目录以外的所有类型的文件，可以用：

```
$ find . ! -type d -print
```

在/etc 目录下查找所有的符号链接文件，可以用

```
$ find /etc -type l -print
```

10、使用 size 选项

可以按照文件长度来查找文件，这里所指的文件长度既可以用块（block）来计量，也可以用字节来计量。以字节计量文件长度的表达形式为 N c；以块计量文件长度只用数字表示即可。

在按照文件长度查找文件时，一般使用这种以字节表示的文件长度，在查看文件系统的大小，因为这时使用块来计量更容易转换。 在当前目录下查找文件长度大于 1 M 字节的文件：

```
$ find . -size +1000000c -print
```

在/home/apache 目录下查找文件长度恰好为 100 字节的文件：

```
$ find /home/apache -size 100c -print
```

在当前目录下查找长度超过 10 块的文件（一块等于 512 字节）：

```
$ find . -size +10 -print
```

11、使用 depth 选项

在使用 find 命令时，可能希望先匹配所有的文件，再在子目录中查找。使用 depth 选项就可以使 find 命令这样做。这样做的一个原因就是，当在使用 find 命令向磁带上备份文件系统时，希望首先备份所有的文件，其次再备份子目录中的文件。

在下面的例子中， find 命令从文件系统的根目录开始，查找一个名为 CON.FILE 的文件。

它将首先匹配所有的文件然后再进入子目录中查找。

```
$ find / -name "CON.FILE" -depth -print
```

12、使用 mount 选项

在当前的文件系统中查找文件（不进入其他文件系统），可以使用 find 命令的 mount 选项。

从当前目录开始查找位于本文件系统中文件名以 XC 结尾的文件：

```
$ find . -name "*.XC" -mount -print
```

练习：请找出你 10 天内所访问或修改过的.c 和.cpp 文件。

find 命令的例子；

1、查找当前用户主目录下的所有文件：

下面两种方法都可以使用

```
$ find $HOME -print
$ find ~ -print
```

2、让当前目录中文件属主具有读、写权限，并且文件所属组的用户和其他用户具有读权限的文件；

```
$ find . -type f -perm 644 -exec ls -l {    } \;
```

3、为了查找系统中所有文件长度为 0 的普通文件，并列出它们的完整路径；

```
$ find / -type f -size 0 -exec ls -l {    } \;
```

4、查找/var/logs 目录中更改时间在 7 日以前的普通文件，并在删除之前询问它们；

```
$ find /var/logs -type f -mtime +7 -ok rm {    } \;
```

5、为了查找系统中所有属于 root 组的文件；

```
$find . -group root -exec ls -l {    } \;
```

6、find 命令将删除当目录中访问时间在 7 日以来、含有数字后缀的 admin.log 文件。

该命令只检查三位数字，所以相应文件的后缀不要超过 999。先建几个 admin.log*的文件 ，才能使用下面这个命令

```
$ find . -name "admin.log[0-9][0-9][0-9]" -atime -7    -ok rm {    } \;
```

7、为了查找当前文件系统中的所有目录并排序；

```
$ find . -type d | sort
```

三、xargs

xargs - build and execute command lines from standard input

在使用 find 命令的-exec 选项处理匹配到的文件时， find 命令将所有匹配到的文件一起传递给 exec 执行。但有些系统对能够传递给 exec 的命令长度有限制，这样在 find 命令运行几分钟之后，就会出现 溢出错误。错误信息通常是“参数列太长”或“参数列溢出”。这就是 xargs 命令的用处所在，特别是与 find 命令一起使用。

find 命令把匹配到的文件传递给 xargs 命令，而 xargs 命令每次只获取一部分文件而不是全部，不像-exec 选项那样。这样它可以先处理最先获取的一部分文件，然后是下一批，并如此继续下去。

在有些系统中，使用-exec 选项会为处理每一个匹配到的文件而发起一个相应的进程，并非将匹配到的文件全部作为参数一次执行；这样在有些情况下就会出现进程过多，系统性能下降的问题，因而效率不高；

而使用 xargs 命令则只有一个进程。另外，在使用 xargs 命令时，究竟是一次获取所有的参数，还是分批取得参数，以及每一次获取参数的数目都会根据该命令的选项及系统内核中相应的可调参数来确定。

来看看 xargs 命令是如何同 find 命令一起使用的，并给出一些例子。

下面的例子查找系统中的每一个普通文件，然后使用 xargs 命令来测试它们分别属于哪类文 件

```
#find . -type f -print | xargs file
```

在当前目录下查找所有用户具有读、写和执行权限的文件，并收回相应的写权限：

```
# ls -l
# find . -perm -7 -print | xargs chmod o-w
# ls -l
```

用 grep 命令在所有的普通文件中搜索 hello 这个词：

```
# find . -type f -print | xargs grep "hello"
```

用 grep 命令在当前目录下的所有普通文件中搜索 hello 这个词：

```
# find . -name \* -type f -print | xargs grep "hello"
```

注意，在上面的例子中， \用来取消 find 命令中的*在 shell 中的特殊含义。

find 命令配合使用 exec 和 xargs 可以使用户对所匹配到的文件执行几乎所有的命令。

4.10 磁盘分区挂载命令

df -h	**查看磁盘使用/剩余空间**
fdisk -l	**磁盘分区**
mount	**挂载**

umount	**卸载**

4.11 进程管理命令

UID	**用户 ID**
PID	**进程 ID**
ps aux | grep xxx	**查看系统中所有进程**
ps -ef | grep xxx	**可以查看子父进程之间的关系**
kill -9 PID	**强制杀死进程**
top	**查看所有进程/cpu/内存/负载**
netstat -anp |grep 进程号	**查看该进程网络信息**
netstat -nlp | grep 端口号	**查看网络端口号占用情况**

4.12 crond 系统定时任务

Crontab -e 编辑定时文件

详情见网页文档：

https://blog.csdn.net/qq_22172133/article/details/81263736

实例操作：

4.13 rpm 包管理与 yum 源

1.rpm 相关命令：

rpm -qa|grep 包名 查找已经安装的 rpm 某包

rpm -ivh 包名 安装 rpm 包

rpm -e 包名 删除 rpm 包

rpm -e --nodeps 软件包 删除 rpm 包不检查依赖

2.yum 源管理：详情见链接文档

https://blog.csdn.net/qingfenggege/article/details/80394564

3.yum 在线安装 lrzsz 上传下载工具

yum -y install lrzsz

4.14 SSH 免密

1.ssh 是什么？

SSH（SecureShell），是建立在应用层基础上的安全协议，其 SSH 客户端适用于多种平台，可以有效防止远程管理过程中的信息泄露问题。

2.配置 hadoop 用户 ssh 免密：

ssh-keygen -t rsa

三台机器在 hadoop 用户下，执行以下命令将公钥拷贝到 node01 服务器上面去

ssh-copy-id node01

node01 在 hadoop 用户下，执行以下命令，将 authorized_keys 拷贝到 node02 与 node03 服务器

cd /home/hadoop/.ssh/

scp authorized_keys node02:$PWD

scp authorized_keys node03:$PWD

4.15 nohup

nohup 是 no hang up 的缩写，就是不挂断的意思。

nohup 命令：如果你正在运行一个进程，而且你觉得在退出帐户时该进程还不会结束，那么可以使用 nohup 命令。该命令可以在你退出帐户/关闭终端之后继续运行相应的进程。

在缺省情况下该作业的所有输出都被重定向到一个名为 nohup.out 的文件中。

常规用法：

nohup command > myout.file 2>&1 &

说明：2>&1 是将标准错误（2）重定向到标准输出（&1），标准输出（&1）再被重定向输入到 myout.file 文件中。

4.16 关闭防火墙

systemctl stop firewalld.service	**关闭防火墙**
systemctl status firewalld.service	**查看防火墙状态**
systemctl disable firewalld.service	**禁止开启启动防火墙**

第五章: linux 三剑客 grep,awk,sed

#1.grep 更适合单纯的查找或匹配文本

#2. sed 更适合编辑文本

#3. awk 更适合格式化文本，对文本进行较复杂格式处理

5.1 grep

1.作用

Linux 系统中 grep 命令是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹 配的行打印出来。grep 全称是 Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户。

grep 家族包括 grep、egrep 和 fgrep。egrep 和 fgrep 的命令只跟 grep 有很小不同。egrep 是 grep 的扩展，支持更多的 re 元字符， fgrep 就是 fixed grep 或 fast grep，它们把所有的字母都看作单词，也就是说，正则表达式中的元字符表示回其自身的字面意义，不再特殊。linux 使用 GNU 版本的 grep。它功能更强，可以通过-G、-E、-F 命令行选项来使用 egrep 和 fgrep 的功能。

2.格式

```
grep [options]
```

3.主要参数

grep --help

[options]主要参数：

-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。

pattern 正则表达式主要参数：

\： 忽略正则表达式中特殊字符的原有含义。

^：匹配正则表达式的开始行。

$: 匹配正则表达式的结束行。

\<：从匹配正则表达 式的行开始。

\>：到匹配正则表达式的行结束。

[ ]：单个字符，如[A]即 A 符合要求 。

[-]：范围，如[A-Z]，即 A、B、C 一直到 Z 都符合要求 。

.：所有的单个字符。

***：有字符，长度可以为 0。**

4.grep 命令使用简单实例

$ grep ‘test’ d*

显示所有以 d 开头的文件中包含 test 的行。

$ grep ‘test’ aa bb cc

显示在 aa，bb，cc 文件中匹配 test 的行。

$ grep ‘[a-z]\{5\}’ aa

显示所有包含每个字符串至少有 5 个连续小写字符的字符串的行。

$ grep ‘w\(es\)t.*\1′ aa

如果 west 被匹配，则 es 就被存储到内存中，并标记为 1，然后搜索任意个字符(.*)，这些字符后面紧跟着 另外一个 es(\1)，找到就显示该行。如果用 egrep 或 grep -E，就不用”\”号进行转义，直接写成’w(es)t.*\1′就可以了。

5.grep 命令使用复杂实例

明确要求搜索子目录：

grep -r

或忽略子目录：

grep -d skip

如果有很多输出时，您可以通过管道将其转到’less’上阅读：

$ grep magic /usr/src/Linux/Documentation/* | less

这样，您就可以更方便地阅读。

有一点要注意，您必需提供一个文件过滤方式(搜索全部文件的话用*)。如果您忘了，’grep’会一直等着，直到该程序被中断。如果您遇到了这样的情况，按 ，然后再试。

下面还有一些有意思的命令行参数：

grep -i pattern files ：不区分大小写地搜索。默认情况区分大小写，

grep -l pattern files ：只列出匹配的文件名，

grep -L pattern files ：列出不匹配的文件名，

grep -w pattern files ：只匹配整个单词，而不是字符串的一部分(如匹配’magic’，而不是’magical’)，

grep -C number pattern files ：匹配的上下文分别显示[number]行，

grep pattern1 | pattern2 files ：显示匹配 pattern1 或 pattern2 的行，

例如：grep "abc\|xyz" testfile 表示过滤包含 abc 或 xyz 的行

grep pattern1 files | grep pattern2 ：显示既匹配 pattern1 又匹配 pattern2 的行。

grep -n pattern files 即可显示行号信息

grep -c pattern files 即可查找总行数

这里还有些用于搜索的特殊符号：

\< 和 \> 分别标注单词的开始与结尾。

例如：

grep man * 会匹配 ‘Batman’、’manic’、’man’等，

grep ‘\<man’ * 匹配’manic’和’man’，但不是’Batman’，

grep ‘\<man\>’ 只匹配’man’，而不是’Batman’或’manic’等其他的字符串。

‘^’：指匹配的字符串在行首，

‘$’：指匹配的字符串在行尾，

5.2 sed

Linux sed 命令是利用脚本来处理文本文件。

sed 可依照脚本的指令来处理、编辑文本文件。

Sed 主要用来自动编辑一个或多个文件、简化对文件的反复操作、编写转换程序等。

参数说明：

-e<script>或--expression=<script> 以选项中指定的script来处理输入的文本文件。

-f<script文件>或--file=<script文件> 以选项中指定的script文件来处理输入的文本文件。

-h 或--help 显示帮助。

-n 或--quiet 或--silent 仅显示 script 处理后的结果。

-V 或--version 显示版本信息。

动作说明：

a ：新增， a 的后面可以接字串，而这些字串会在新的一行出现(目前的下一行)～

c ：取代， c 的后面可以接字串，这些字串可以取代 n1,n2 之间的行！

d ：删除，因为是删除啊，所以 d 后面通常不接任何咚咚；

i ：插入， i 的后面可以接字串，而这些字串会在新的一行出现(目前的上一行)；

p ：打印，亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运行～

s ：取代，可以直接进行取代的工作哩！通常这个 s 的动作可以搭配正规表示法！例如 1,20s/old/new/g 就是啦！

实例

注意：sed -e 是修改输出终端，sed -i 才是在源文件修改！！！

实操案例：

#1.sed -e 与 sed -i

在 testfile 文件的第四行后添加一行，并将结果输出到标准输出，在命令行提示符下输入如下命令：

```
sed -e 4a\newLine testfile
```

首先查看 testfile 中的内容如下： cat testfile #查看 testfile 中的内容

自己 vim testfile 文件将下列内容输入做测试用。

```
HELLO LINUX!    Linux is a free unix-type opterating system.
```

```
This is a linux testfile!

Linux test

Are you ok



You is pig
```

使用 sed 命令后，输出结果如下：

```
sed -e 4a\newline testfile #使用 sed  在第四行后添加新字符串
```

那如果是要在第二行前:

```
nl /etc/passwd | sed '2i drink tea'
```

将第 2-5 行的内容取代成为 No 2-5 number:

```
nl /etc/passwd | sed '2,5c No 2-5 number'
```

#2. sed -s　替换

```
把 is  换成 are：

sed 's/is/are/g' testfile
```

#3.sed -d　删除

```
$删除头三行:

sed '1,3d' testfile

$删除包含 pig 的行:

sed '\%pig%d' testfile

$删除所有空白行

sed '/^$/d' testfile
```

#sed -n

显示 5-10 行:

sed -n '5,10p' testfile

5.3 awk

这里我们介绍一下 awk 实用用法，更加详细参考下面链接文档

https://www.cnblogs.com/ginvip/p/6352157.html

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。之所以叫 AWK 是因为其取了三位创始人 Alfred Aho，Peter Weinberger, 和 Brian Kernighan 的 Family Name 的首字符。

语法

选项参数说明：

-F fs or --field-separator fs

指定输入文件折分隔符，fs 是一个字符串或者是一个正则表达式，如 -F:。

-v var=value or --asign var=value

赋值一个用户定义变量。

-f scripfile or --file scriptfile

从脚本文件中读取 awk 命令。

-mf nnn and -mr nnn

对 nnn 值设置内在限制，-mf 选项限制分配给 nnn 的最大块数目；-mr

选项限制记录的最大数目。这两个功能是 Bell 实验室版 awk 的扩展功能，在标准 awk 中不适用。

-W compact or --compat, -W traditional or --traditional

在兼容模式下运行 awk。所以 gawk 的行为和标准的 awk 完全一样，所有的 awk 扩展都被忽略。

-W copyleft or --copyleft, -W copyright or --copyright

打印简短的版权信息。

-W help or --help, -W usage or --usage

打印全部 awk 选项和每个选项的简短说明。

-W lint or --lint

打印不能向传统 unix 平台移植的结构的警告。

-W lint-old or --lint-old

打印关于不能向传统 unix 平台移植的结构的警告。

-W posix

打开兼容模式。但有以下限制，不识别：/x、函数关键字、func、换码序列以及当 fs 是一个空格时，将新行作为一个域分隔符；操作符**和**=不能代替^和^=；fflush 无效。

-W re-interval or --re-inerval

允许间隔正则表达式的使用，参考(grep 中的 Posix 字符类)，如括号表达式[[:alpha:]]。

-W source program-text or --source program-text

使用 program-text 作为源代码，可与-f 命令混用。

```
-W version or --version

打印 bug 报告信息的版本。
```

基本用法

log.txt 文本内容如下：

自己 vim log.txt,将下面文件输入做测试。

```
2 this is a test3 Are you like awkThis's a test

10 There are orange,apple,mongo
```

用法一：

```
awk '{[pattern] action}' {filenames}     # 行匹配语句 awk '' 只能用单引号
```

实例：

```
# 每行按空格或 TAB 分割，输出文本中的 1、4 项

 $ awk '{print $1,$4}' log.txt

#从文件中找出长度大于 8 的行

awk 'length>8' log.txt
```

用法二：

```
awk -F   #-F 相当于内置变量 FS, 指定分割字符
```

实例：

```
# 使用","分割

 $   awk -F, '{print $1,$2}'     log.txt

# 使用多个分隔符.先使用空格分割，然后对分割结果再使用","分割
```

```
$ awk -F '[ ,]'  '{print $1,$2,$5}'     log.txt
```

用法三：

```
awk -v  # 设置变量
```

实例：

```
#设置变量 a,并赋值=1，打印第一列和第一列加变量 a 的值
$ awk -va=1 '{print $1,$1+a}' log.txt
#设置两个变量 a=1 和 b=2，并赋值打印第一列，第一列+a,第一列+b 的值：
$ awk -va=1 -vb=2 '{print $1,$1+a,$1b}' log.txt
```

用法四：

```
awk -f {awk 脚本} {文件名}
```

实例：

```
$ awk -f cal.awk log.txt
```

运算符

运算符	描述
= += -= *= /= %= ^= **=	赋值
?:	C 条件表达式
||	逻辑或
&&	逻辑与
~ 和 !~	匹配正则表达式和不匹配正则表达式
< <= > >= != ==	关系运算符
空格	连接
+ -	加，减

运算符	描述
* / %	乘，除与求余
+ - !	一元加，减和逻辑非
^ ***	求幂
++ --	增加或减少，作为前缀或后缀
$	字段引用
in	数组成员

```
#过滤第一列大于 2 的行
$ awk '$1>2' log.txt
#过滤第一列等于 2 的行
$ awk '$1==2 {print $1,$3}' log.txt
```

内建变量

变量	描述
$n	当前记录的第 n 个字段，字段间由 FS 分隔
$0	完整的输入记录
ARGC	命令行参数的数目
ARGIND	命令行中当前文件的位置(从 0 开始算)
ARGV	包含命令行参数的数组
CONVFMT	数字转换格式(默认值为%.6g)ENVIRON 环境变量关联数组
ERRNO	最后一个系统错误的描述
FIELDWIDTHS	字段宽度列表(用空格键分隔)
FILENAME	当前文件名
FNR	各文件分别计数的行号

变量	描述
FS	字段分隔符(默认是任何空格)
IGNORECASE	如果为真，则进行忽略大小写的匹配
NF	一条记录的字段的数目
NR	已经读出的记录数，就是行号，从 1 开始
OFMT	数字的输出格式(默认值是%.6g)
OFS	输出记录分隔符（输出换行符），输出时用指定的符号代替换行符
ORS	输出记录分隔符(默认值是一个换行符)
RLENGTH	由 match 函数所匹配的字符串的长度
RS	记录分隔符(默认是一个换行符)
RSTART	由 match 函数所匹配的字符串的第一个位置
SUBSEP	数组下标分隔符(默认值是/034)

```
$ awk 'BEGIN{printf "%4s %4s %4s %4s %4s %4s %4s %4s %4s\n","FILENAME","ARGC","FNR","FS","NF","NR","OFS","O
RS","RS";printf "---------------------------------------------\n"} {printf "%4s %4s %4s %4s %4s %4s %4s %4s
 %4s\n",FILENAME,ARGC,FNR,FS,NF,NR,OFS,ORS,RS}'  log.txt

FILENAME ARGC  FNR   FS   NF   NR  OFS  ORS   RS---------------------------------------------

log.txt    2    1         5    1

log.txt    2    2         5    2

log.txt    2    3         3    3

log.txt    2    4         4    4

$ awk -F\' 'BEGIN{printf "%4s %4s %4s %4s %4s %4s %4s %4s %4s\n","FILENAME","ARGC","FNR","FS","NF","NR","OF
S","ORS","RS";printf "---------------------------------------------\n"} {printf "%4s %4s %4s %4s %4s %4s %4
s %4s %4s\n",FILENAME,ARGC,FNR,FS,NF,NR,OFS,ORS,RS}'  log.txt

FILENAME ARGC  FNR   FS   NF   NR  OFS  ORS   RS---------------------------------------------

log.txt    2    1    '    1    1

log.txt    2    2    '    1    2
```

```
log.txt    2    3    '    2    3

log.txt    2    4    '    1    4# 输出顺序号 NR，匹配文本行号
```

使用正则，字符串匹配：

```
# 输出第二列包含 "th"，并打印第二列与第四列

$ awk '$2 ~ /th/ {print $2,$4}' log.txt
```

第六章 shell 快速入门

Shell 更多用法见下面 shell 文档（双击可打开）

Shell编程快速入门.pdf

6.1 shell 历史

Shell 的作用是解释执行用户的命令，用户输入一条命令，Shell 就解释执行一条，这种方式称为交互式（Interactive），Shell 还有一种执行命令的方式称为批处理（Batch），用户事先写一个 Shell 脚本（Script），其中有很多条命令，让 Shell 一次把这些命令执行完，而不必一条一条地敲命令。Shell 脚本和编程语言很相似，也有变量和流程控制语句，但 Shell 脚本是解释执行的，不需要编译，Shell 程序从脚本中一行一行读取并执行这些命令，相当于一个用户把脚本中的命令一行一行敲到 Shell 提示符下执行。

由于历史原因，UNIX 系统上有很多种 Shell：

1.sh（Bourne Shell）：由 Steve Bourne 开发，各种 UNIX 系统都配有 sh。

2.csh（C Shell）：由 Bill Joy 开发，随 BSD UNIX 发布，它的流程控制语句很像 C 语言，支持很多 Bourne Shell 所不支持的功能：作业控制，命令历史，命令行编辑。

3.ksh（Korn Shell）：由 David Korn 开发，向后兼容 sh 的功能，并且添加了 csh 引入的新功能，是目前很多 UNIX 系统标准配置的 Shell，在这些系统上/bin/sh 往往是指向/bin/ksh 的符号链接。

4.tcsh（TENEX C Shell）：是 csh 的增强版本，引入了命令补全等功能，在 FreeBSD、Mac OS X 等系统上替代了 csh。

5.bash（Bourne Again Shell）：由 GNU 开发的 Shell，主要目标是与 POSIX 标准保持一致，同时兼顾对 sh 的兼容，bash 从 csh 和 ksh 借鉴了很多功能，是各种 Linux 发行版标准配置的 Shell，在 Linux 系统上/bin/sh 往往是指向/bin/bash 的符号链接。虽然如此，bash 和 sh 还是有很多不同的，一方面，bash 扩展了一些命令和参数，另一方面，bash 并不完全和 sh 兼容，有些行为并不一致，所以 bash 需要模拟 sh 的行为：当我们通过 sh 这个程序名启动 bash 时，bash 可以假装自己是 sh，不认扩展的命令，并且行为与 sh 保持一致。

6.zsh 的命令补全功能非常强大，可以补齐路径，补齐命令，补齐参数等。

```
vim /etc/passwd
```

```
其中最后一列显示了用户对应的 shell 类型

root:x:0:0:root:/root:/bin/bash

nobody:x:65534:65534:nobody:/nonexistent:/bin/sh

syslog:x:101:103::/home/syslog:/bin/false

itcast:x:1000:1000:itcast,,,:/home/itcast:/bin/bash

ftp:x:115:125:ftp daemon,,,:/srv/ftp:/bin/false
```

用户在命令行输入命令后，一般情况下 Shell 会 fork 并 exec 该命令，但是 Shell 的内建命令例外，执行内建命令相当于调用 Shell 进程中的一个函数，并不创建新的进程。以前学过的 cd、alias、umask、exit 等命令即是内建命令，凡是用 which 命令查不到程序文件所在位置的命令都是内建命令，内建命令没有单独的 man 手册，要在 man 手册中查看内建命令，应该

```
$ man bash-builtins
```

如 export、shift、if、eval、[、for、while 等等。内建命令虽然不创建新的进程，但也会有 Exit Status，通常也用 0 表示成功非零表示失败，虽然内建命令不创建新的进程，但执行结束后也会有一个状态码，也可以用特殊变量$?读出。

6.2 shell 执行脚本

编写一个简单的脚本 test.sh：

```
#! /bin/bash

cd ..

ls
```

Shell 脚本中用#表示注释，相当于 C 语言的//注释。但如果#位于第一行开头，并且是#!（称为 Shebang）则例外，它表示该脚本使用后面指定的解释器/bin/sh 解释执行。如果把这个脚本文件加上可执行权限然后执行：

```
chmod a+x test.sh

sh test.sh
```

注：

u 表示用户

g 表示用户组

o 表示其它

a 表示所有

chmod a+x a.txt 等价于 chmod +x a.txt

给所有用户给予 a.txt 文件可执行权限

chmod u+x a.txt

a.txt 文件的所有用户可执行权限

chmod g+x a.txt

a.txt 用户组可执行权限

chmod o+x a.txt

a.txt 其他用户可执行权限

Shell 会 fork 一个子进程并调用 exec 执行./test.sh 这个程序，exec 系统调用应该把子进程的代码段替换成./test.sh 程序的代码段，并从它

的_start 开始执行。然而 test.sh 是个文本文件，根本没有代码段和_start 函数，怎么办呢？其实 exec 还有另外一种机制，如果要执行的是一个文本文件，并且第一行用 Shebang 指定了解释器，则用解释器程序的代码段替换当前进程，并且从解释器的_start 开始执行，而这个文本文件被当作命令行参数传给解释器。因此，执行上述脚本相当于执行程序

```
$ /bin/sh ./test.sh
```

以这种方式执行不需要 test.sh 文件具有可执行权限。

如果将命令行下输入的命令用()括号括起来，那么也会 fork 出一个子 Shell 执行小括号中的命令，一行中可以输入由分号;隔开的多个命令，比如：

```
$ (cd ..;ls -l)
```

和上面两种方法执行 Shell 脚本的效果是相同的，cd ..命令改变的是子 Shell 的 PWD，而不会影响到交互式 Shell。然而命令

```
$ cd ..;ls -l
```

则有不同的效果，cd ..命令是直接在交互式 Shell 下执行的，改变交互式 Shell 的 PWD，然而这种方式相当于这样执行 Shell 脚本：

```
$ source ./test.sh
```

或者

```
$ . ./test.sh
```

source 或者.命令是 Shell 的内建命令，这种方式也不会创建子 Shell，而是直接在交互式 Shell 下逐行执行脚本中的命令。

6.3 shell 变量

按照惯例，Shell 变量由全大写字母加下划线组成，有两种类型的 Shell 变量：

1.环境变量

环境变量可以从父进程传给子进程，因此 Shell 进程的环境变量可以从当前 Shell 进程传给 fork 出来的子进程。用 printenv 命令可以显示当前 Shell 进程的环境变量。

2.本地变量

只存在于当前 Shell 进程，用 set 命令可以显示当前 Shell 进程中定义的所有变量（包括本地变量和环境变量）和函数。

环境变量是任何进程都有的概念，而本地变量是 Shell 特有的概念。在 Shell 中，环境变量和本地变量的定义和用法相似。在 Shell 中定义或赋值一个变量：

```
itcast$ VARNAME=value
```

注意等号两边都不能有空格，否则会被 Shell 解释成命令和命令行参数。

一个变量定义后仅存在于当前 Shell 进程，它是本地变量，用 export 命令可以把本地变量导出为环境变量，定义和导出环境变量通常可以一步完成：

```
itcast$ export VARNAME=value
```

也可以分两步完成：

```
itcast$ VARNAME=value

itcast$ export VARNAME
```

用 unset 命令可以删除已定义的环境变量或本地变量。

```
itcast$ unset VARNAME
```

如果一个变量叫做 VARNAME，用${VARNAME}可以表示它的值，在不引起歧义的情况下也可以用$VARNAME 表示它的值。通过以下例子比较这两种表示法的不同：

```
itcast$ echo $SHELL
```

注意，在定义变量时不用$，取变量值时要用$。和 C 语言不同的是，Shell 变量不需要明确定义类型，事实上 Shell 变量的值都是字符串，比如我们定义 VAR=45，其实 VAR 的值是字符串 45 而非整数。Shell 变量不需要先定义后使用，如果对一个没有定义的变量取值，则值为空字符串。

6.4 shell 基本运算符

Shell 和其他编程语言一样，支持多种运算符，包括：

算数运算符

关系运算符

布尔运算符

字符串运算符

文件测试运算符

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 awk 和 expr，expr 最常用。

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。

例如，两个数相加(注意使用的是反引号 ` 而不是单引号 '):

实例

```
#!/bin/bash

val=`expr 2 + 2`
echo "两数之和为 : $val"
```

运行实例 »

执行脚本，输出结果如下所示：

```
两数之和为 : 4
```

两点注意：

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，这与我们熟悉的大多数编程语言不一样。

完整的表达式要被 ` ` 包含，注意这个字符不是常用的单引号，在 Esc 键下边。

算术运算符

下表列出了常用的算术运算符，假定变量 a 为 10，变量 b 为 20：

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

注意：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]。

实例

算术运算符实例如下：

实例

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

a=10
b=20

val=`expr $a + $b`
echo "a + b : $val"

val=`expr $a - $b`
echo "a - b : $val"

val=`expr $a \* $b`
echo "a * b : $val"

val=`expr $b / $a`
```

```
echo "b / a : $val"

val=`expr $b % $a`
echo "b % a : $val"

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

执行脚本，输出结果如下所示：

```
a + b : 30

a - b : -10

a * b : 200

b / a : 2

b % a : 0

a 不等于 b
```

注意：

乘号(*)前边必须加反斜杠(\)才能实现乘法运算；

if...then...fi 是条件语句，后续将会讲解。

在 MAC 中 shell 的 expr 语法是：$((表达式))，此处表达式中的 "*" 不需要转义符号 "\" 。

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

实例

关系运算符实例如下：

实例

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

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
if [ $a -ne $b ]
then
   echo "$a -ne $b: a 不等于 b"
else
   echo "$a -ne $b : a 等于 b"
fi
if [ $a -gt $b ]
then
   echo "$a -gt $b: a 大于 b"
else
   echo "$a -gt $b: a 不大于 b"
```

```
fi
if [ $a -lt $b ]
then
   echo "$a -lt $b: a 小于 b"
else
   echo "$a -lt $b: a 不小于 b"
fi
if [ $a -ge $b ]
then
   echo "$a -ge $b: a 大于或等于 b"
else
   echo "$a -ge $b: a 小于 b"
fi
if [ $a -le $b ]
then
   echo "$a -le $b: a 小于或等于 b"
else
   echo "$a -le $b: a 大于 b"
fi
```

执行脚本，输出结果如下所示：

```
10 -eq 20: a 不等于 b10 -ne 20: a 不等于 b10 -gt 20: a 不大于 b10 -lt 20: a 小于 b10 -ge 20: a 小于 b10 -le 2
0: a 小于或等于 b
```

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。	[! false] 返回 true。
-o	或运算，有一个表达式为 true 则返回 true。	[$a -lt 20 -o $b -gt 100] 返回 true。
-a	与运算，两个表达式都为 true 才返回 true。	[$a -lt 20 -a $b -gt 100] 返回 false。

实例

布尔运算符实例如下：

实例

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

a=10
b=20

if [ $a != $b ]
then
   echo "$a != $b : a 不等于 b"
else
   echo "$a == $b: a 等于 b"
fi
if [ $a -lt 100 -a $b -gt 15 ]
then
   echo "$a 小于 100 且 $b 大于 15 : 返回 true"
else
   echo "$a 小于 100 且 $b 大于 15 : 返回 false"
fi
if [ $a -lt 100 -o $b -gt 100 ]
then
   echo "$a 小于 100 或 $b 大于 100 : 返回 true"
else
   echo "$a 小于 100 或 $b 大于 100 : 返回 false"
fi
if [ $a -lt 5 -o $b -gt 100 ]
then
   echo "$a 小于 5 或 $b 大于 100 : 返回 true"
else
   echo "$a 小于 5 或 $b 大于 100 : 返回 false"
fi
```

执行脚本，输出结果如下所示：

```
10 != 20 : a 不等于 b10 小于 100 且 20 大于 15 : 返回 true10 小于 100 或 20 大于 100 : 返回 true10 小于 5 或 20
 大于 100 : 返回 false
```

逻辑运算符

以下介绍 Shell 的逻辑运算符，假定变量 a 为 10，变量 b 为 20:

运算符	说明	举例
&&	逻辑的 AND	[[$a -lt 100 && $b -gt 100]] 返回 false

||	逻辑的 OR	[[$a -lt 100 || $b -gt 100]] 返回 true

实例

逻辑运算符实例如下：

实例

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

a=10
b=20

if [[ $a -lt 100 && $b -gt 100 ]]
then
   echo "返回 true"
else
   echo "返回 false"
fi

if [[ $a -lt 100 || $b -gt 100 ]]
then
   echo "返回 true"
else
   echo "返回 false"
fi
```

执行脚本，输出结果如下所示：

```
返回 false返回 true
```

字符串运算符

下表列出了常用的字符串运算符，假定变量 a 为 "abc"，变量 b 为 "efg"：

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 $a] 返回 false。
-n	检测字符串长度是否不为 0，不为 0 返回 true。	[-n "$a"] 返回 true。
$	检测字符串是否为空，不为空返回 true。	[$a] 返回 true。

实例

字符串运算符实例如下：

实例

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

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
if [ -z $a ]
then
   echo "-z $a : 字符串长度为 0"
else
   echo "-z $a : 字符串长度不为 0"
fi
if [ -n "$a" ]
then
   echo "-n $a : 字符串长度不为 0"
else
```

```
    echo "-n $a : 字符串长度为 0"
fi
if [ $a ]
then
    echo "$a : 字符串不为空"
else
    echo "$a : 字符串为空"
fi
```

执行脚本，输出结果如下所示：

```
abc = efg: a 不等于 b

abc != efg : a 不等于 b-z abc : 字符串长度不为 0-n abc : 字符串长度不为 0

abc : 字符串不为空
```

文件测试运算符

文件测试运算符用于检测 Unix 文件的各种属性。

属性检测描述如下：

操作符	说明	举例
-b file	检测文件是否是块设备文件，如果是，则返回 true。	[-b $file] 返回 false。
-c file	检测文件是否是字符设备文件，如果是，则返回 true。	[-c $file] 返回 false。
-d file	检测文件是否是目录，如果是，则返回 true。	[-d $file] 返回 false。
-f file	检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。	[-f $file] 返回 true。
-g file	检测文件是否设置了 SGID 位，如果是，则返回 true。	[-g $file] 返回 false。
-k file	检测文件是否设置了粘着位(Sticky Bit)，如果是，则返回 true。	[-k $file] 返回 false。
-p file	检测文件是否是有名管道，如果是，则返回 true。	[-p $file] 返回 false。
-u file	检测文件是否设置了 SUID 位，如果是，则返回 true。	[-u $file] 返回 false。
-r file	检测文件是否可读，如果是，则返回 true。	[-r $file] 返回 true。
-w file	检测文件是否可写，如果是，则返回 true。	[-w $file] 返回 true。

-x file	检测文件是否可执行，如果是，则返回 true。	[-x $file] 返回 true。
-s file	检测文件是否为空（文件大小是否大于 0），不为空返回 true。	[-s $file] 返回 true。
-e file	检测文件（包括目录）是否存在，如果是，则返回 true。	[-e $file] 返回 true。

其他检查符：

-S: 判断某文件是否 socket。

-L: 检测文件是否存在并且是一个符号链接。

实例

变量 file 表示文件 /var/www/runoob/test.sh，它的大小为 100 字节，具有 rwx 权限。下面的代码，将检测该文件的各种属性：

实例

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

file="/var/www/runoob/test.sh"
if [ -r $file ]
then
   echo "文件可读"
else
   echo "文件不可读"
fi
if [ -w $file ]
then
   echo "文件可写"
else
   echo "文件不可写"
fi
if [ -x $file ]
then
   echo "文件可执行"
else
   echo "文件不可执行"
fi
if [ -f $file ]
```

```
then
   echo "文件为普通文件"
else
   echo "文件为特殊文件"
fi
if [ -d $file ]
then
   echo "文件是个目录"
else
   echo "文件不是个目录"
fi
if [ -s $file ]
then
   echo "文件不为空"
else
   echo "文件为空"
fi
if [ -e $file ]
then
   echo "文件存在"
else
   echo "文件不存在"
fi
```

执行脚本，输出结果如下所示：

```
文件可读文件可写文件可执行文件为普通文件文件不是个目录文件不为空文件存在
```

6.5 shell 常用符号：

1.通配符：* ? [ ]

这些用于匹配的字符称为通配符（Wildcard），具体如下：

```
通配符：

*   匹配 0 个或多个任意字符

?   匹配一个任意字符

[若干字符]  匹配方括号中任意一个字符的一次出现
```

```
$ ls /dev/ttyS*

$ ls ch0?.doc

$ ls ch0[0-2].doc

$ ls ch[012]   [0-9].doc
```

注意，Globbing 所匹配的文件名是由 Shell 展开的，也就是说在参数还没传给程序之前已经展开了，比如上述 ls ch0[012].doc 命令，如果当前目录下有 ch00.doc 和 ch02.doc，则传给 ls 命令的参数实际上是这两个文件名，而不是一个匹配字符串。

2. 命令代换：`或 $()

由''反引号括起来的也是一条命令，Shell 先执行该命令，然后将输出结果立刻代换到当前命令行中。例如定义一个变量存放 date 命令的输出：

```
itcast$ DATE=`date`

itcast$ echo $DATE
```

命令代换也可以用$()表示：

```
itcast$ DATE=$(date)
```

3.算术代换：$(())

用于算术计算，$(())中的 Shell 变量取值将转换成整数，同样含义的$[]等价例如：

```
itcast$ VAR=45

itcast$ echo $(($VAR+3))

$(())中只能用+-*/和()运算符，并且只能做整数运算。



$[base#n],其中 base 表示进制,n 按照 base 进制解释，后面再有运算数，按十进制解释。
```

```
echo $[2#10+11]

echo $[8#10+11]

echo $[10#10+11]
```

4.转义字符

和 C 语言类似，\在 Shell 中被用作转义字符，用于去除紧跟其后的单个字符的特殊意义（回车除外），换句话说，紧跟其后的字符取字面值。例如：

```
itcast$ echo $SHELL

/bin/bash

itcast$ echo \$SHELL

$SHELL

itcast$ echo \\

\
```

比如创建一个文件名为“$ $”的文件可以这样：

```
itcast$ touch \$\ \$
```

还有一个字符虽然不具有特殊含义，但是要用它做文件名也很麻烦，就是-号。如果要创建一个文件名以-号开头的文件，这样是不行的：

```
itcast$ touch -hello

touch: invalid option -- h

Try `touch --help' for more information.
```

即使加上\转义也还是报错：

```
itcast$ touch \-hello
```

```
touch: invalid option -- h

Try `touch --help' for more information.
```

因为各种 UNIX 命令都把-号开头的命令行参数当作命令的选项，而不会当作文件名。如果非要处理以-号开头的文件名，可以有两种办法：

```
itcast$ touch ./-hello
```

或者

```
itcast$ touch -- -hello
```

\还有一种用法，在\后敲回车表示续行，Shell 并不会立刻执行命令，而是把光标移到下一行，给出一个续行提示符>，等待用户继续输入，最后把所有的续行接到一起当作一个命令执行。例如：

```
itcast$ ls \

> -l

（ls -l 命令的输出）
```

5.单引号

和 C 语言不一样，Shell 脚本中的单引号和双引号一样都是字符串的界定符（双引号下一节介绍），而不是字符的界定符。单引号用于保持引号内所有字符的字面值，即使引号内的\和回车也不例外，但是字符串中不能出现单引号。如果引号没有配对就输入回车，Shell 会给出续行提示符，要求用户把引号配上对。例如：

```
itcast$ echo '$SHELL'

$SHELL

itcast$ echo 'ABC\（回车）
```

```
> DE'（再按一次回车结束命令）

ABC\

DE
```

6.双引号

被双引号用括住的内容，将被视为单一字串。它防止通配符扩展，但允许变量扩展。这点与单引号的处理方式不同

```
itcast$ DATE=$(date)

itcast$ echo "$DATE"

itcast$ echo '$DATE'
```

6.6 shell 条件测试 test

命令 test 或[可以测试一个条件是否成立，如果测试结果为真，则该命令的 Exit Status 为 0，如果测试结果为假，则命令的 Exit Status 为 1（注意与 C 语言的逻辑表示正好相反）。例如测试两个数的大小关系：

```
itcast@ubuntu:~$ var=2

itcast@ubuntu:~$ test $var -gt 1

itcast@ubuntu:~$ echo $?

0

itcast@ubuntu:~$ test $var -gt 3

itcast@ubuntu:~$ echo $?

1

itcast@ubuntu:~$ [ $var -gt 3 ]

itcast@ubuntu:~$ echo $?

1
```

```
itcast@ubuntu:~$
```

虽然看起来很奇怪，但左方括号[确实是一个命令的名字，传给命令的各参数之间应该用空格隔开，比如，$VAR、-gt、3、]是[命令的四个参数，它们之间必须用空格隔开。命令 test 或[的参数形式是相同的，只不过 test 命令不需要]参数。以[命令为例，常见的测试命令如下表所示：

```
[ -d DIR ]              如果 DIR 存在并且是一个目录则为真

[ -f FILE ]             如果 FILE 存在且是一个普通文件则为真

[ -z STRING ]           如果 STRING 的长度为零则为真

[ -n STRING ]           如果 STRING 的长度非零则为真

[ STRING1 = STRING2 ]   如果两个字符串相同则为真

[ STRING1 != STRING2 ]  如果字符串不相同则为真

[ ARG1 OP ARG2 ]        ARG1 和 ARG2 应该是整数或者取值为整数的变量，OP 是-eq（等于）-ne（不等于）-lt（小于）-le（小于等于）-gt（大于）-ge（大于等于）之中的一个
```

和 C 语言类似，测试条件之间还可以做与、或、非逻辑运算：

```
带与、或、非的测试命令



[ ! EXPR ]          EXPR 可以是上表中的任意一种测试条件，!表示逻辑反

[ EXPR1 -a EXPR2 ]  EXPR1 和 EXPR2 可以是上表中的任意一种测试条件，-a 表示逻辑与

[ EXPR1 -o EXPR2 ]  EXPR1 和 EXPR2 可以是上表中的任意一种测试条件，-o 表示逻辑或
```

例如：

```
$ VAR=abc

$ [ -d Desktop -a $VAR = 'abc' ]

$ echo $?

0
```

注意，如果上例中的$VAR 变量事先没有定义，则被 Shell 展开为空字符串，会造成测试条件的语法错误（展开为[-d Desktop -a = 'abc']），作为一种好的 Shell 编程习惯，应该总是把变量取值放在双引号之中（展开为[-d Desktop -a "" = 'abc']）：

```
$ unset VAR

$ [ -d Desktop -a $VAR = 'abc' ]

bash: [: too many arguments

$ [ -d Desktop -a "$VAR" = 'abc' ]

$ echo $?

1
```

6.7 shell 流程控制：

1. if/then/elif/else/fi

和 C 语言类似，在 Shell 中用 if、then、elif、else、fi 这几条命令实现分支控制。这种流程控制语句本质上也是由若干条 Shell 命令组成的，例如先前讲过的

```
if [ -f ~/.bashrc ]; then

    . ~/.bashrc

fi
```

其实是三条命令，if [-f ~/.bashrc]是第一条，then . ~/.bashrc 是第二条，fi 是第三条。如果两条命令写在同一行则需要用;号隔开，一行只写一条命令就不需要写;号了，另外，then 后面有换行，但这条命令没写完，Shell 会自动续行，把下一行接在 then 后面当作一条命令处理。和[命令一样，要注意命令和各参数之间必须用空格隔开。if 命令的

参数组成一条子命令，如果该子命令的 Exit Status 为 0（表示真），则执行 then 后面的子命令，如果 Exit Status 非 0（表示假），则执行 elif、else 或者 fi 后面的子命令。if 后面的子命令通常是测试命令，但也可以是其它命令。Shell 脚本没有{}括号，所以用 fi 表示 if 语句块的结束。见下例：

```
#! /bin/sh

if [ -f /bin/bash ]then echo "/bin/bash is a file"else echo "/bin/bash is NOT a file"fiif :; then
echo "always true"; fi
```

:是一个特殊的命令，称为空命令，该命令不做任何事，但 Exit Status 总是真。此外，也可以执行/bin/true 或/bin/false 得到真或假的 Exit Status。再看一个例子：

```
#! /bin/sh


  echo "Is it morning? Please answer yes or no."
  read YES_OR_NO
  if [ "$YES_OR_NO" = "yes" ]; then
    echo "Good morning!"
  elif [ "$YES_OR_NO" = "no" ]; then
    echo "Good afternoon!"
  else
    echo "Sorry, $YES_OR_NO not recognized. Enter yes or no."
    exit 1
  fi
  exit 0
```

exit 0

上例中的 read 命令的作用是等待用户输入一行字符串，将该字符串存到一个 Shell 变量中。

此外，Shell 还提供了&&和||语法，和 C 语言类似，具有 Short-circuit 特性，很多 Shell 脚本喜欢写成这样：

```
test "$(whoami)" != 'root' && (echo you are using a non-privileged account; exit 1)
```

&&相当于“if...then...”，而||相当于“if not...then...”。&&和||用于连接两个命令，而上面讲的-a 和-o 仅用于在测试表达式中连接两个测试条件，要注意它们的区别，例如，

```
test "$VAR" -gt 1 -a "$VAR" -lt 3
```

和以下写法是等价的

```
test "$VAR" -gt 1 && test "$VAR" -lt 3
```

2.case/esac

case 命令可类比 C 语言的 switch/case 语句，esac 表示 case 语句块的结束。C 语言的 case 只能匹配整型或字符型常量表达式，而 Shell 脚本的 case 可以匹配字符串和 Wildcard，每个匹配分支可以有若干条命令，末尾必须以;;结束，执行时找到第一个匹配的分支并执行相应的命令，然后直接跳到 esac 之后，不需要像 C 语言一样用 break 跳出。

```
#! /bin/sh


echo "Is it morning? Please answer yes or no."
read YES_OR_NO
case "$YES_OR_NO" in
yes|y|Yes|YES)
```

```
    echo "Good Morning!";;
  [nN]*)
    echo "Good Afternoon!";;
  *)
    echo "Sorry, $YES_OR_NO not recognized. Enter yes or no."
    exit 1;;
  esac
  exit 0
```

使用 case 语句的例子可以在系统服务的脚本目录/etc/init.d 中找到。这个目录下的脚本大多具有这种形式（以/etc/init.d/nfs-kernel-server 为例）：

```
case "$1" in
      start)
          ...
      ;;
      stop)
          ...
      ;;
      reload | force-reload)
          ...
      ;;
      restart)
      ...
      *)
          log_success_msg "Usage: nfs-kernel-server
{start|stop|status|reload|force-reload|restart}"
          exit 1
```

```
    ;;
esac
```

启动 nfs-kernel-server 服务的命令是

```
$ sudo /etc/init.d/nfs-kernel-server start
```

$1 是一个特殊变量，在执行脚本时自动取值为第一个命令行参数，也就是 start，所以进入 start)分支执行相关的命令。同理，命令行参数指定为 stop、reload 或 restart 可以进入其它分支执行停止服务、重新加载配置文件或重新启动服务的相关命令。

3.for/do/done

Shell 脚本的 for 循环结构和 C 语言很不一样，它类似于某些编程语言的 foreach 循环。例如：

```
#! /bin/sh

for FRUIT in apple banana pear; do
  echo "I like $FRUIT"
done
```

FRUIT 是一个循环变量，第一次循环$FRUIT 的取值是 apple，第二次取值是 banana，第三次取值是 pear。再比如，要将当前目录下的 chap0、chap1、chap2 等文件名改为 chap0~、chap1~、chap2~等（按惯例，末尾有~字符的文件名表示临时文件），这个命令可以这样写：

```
$ for FILENAME in chap?; do mv $FILENAME $FILENAME~; done
```

也可以这样写：

```
$ for FILENAME in `ls chap?`; do mv $FILENAME $FILENAME~; done
```

4.while/do/done

while 的用法和 C 语言类似。比如一个验证密码的脚本：

```
#! /bin/sh



echo "Enter password:"

read TRY

while [ "$TRY" != "secret" ]; do

  echo "Sorry, try again"

  read TRY

done
```

下面的例子通过算术运算控制循环的次数：

```
#! /bin/sh

COUNTER=1

while [ "$COUNTER" -lt 10 ]; do

  echo "Here we go again"

  COUNTER=$(($COUNTER+1))

done
```

Shell 还有 until 循环，类似 C 语言的 do...while 循环。本章从略。

5.break 和 continue

break[n]可以指定跳出几层循环，continue 跳过本次循环步，没跳出整个循环。

break 跳出，continue 跳过。

6.8 shell 数组

数组中可以存放多个值。Bash Shell 只支持一维数组（不支持多维数组），初始化时不需要定义数组大小（与 PHP 类似）。

与大部分编程语言类似，数组元素的下标由 0 开始。

Shell 数组用括号来表示，元素用"空格"符号分割开，语法格式如下：

```
array_name=(value1 ... valuen)
```

实例

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com



my_array=(A B "C" D)
```

我们也可以使用下标来定义数组:

```
array_name[0]=value0

array_name[1]=value1

array_name[2]=value2
```

读取数组

读取数组元素值的一般格式是：

```
${array_name[index]}
```

实例

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com



my_array=(A B "C" D)
```

```
echo "第一个元素为: ${my_array[0]}"

echo "第二个元素为: ${my_array[1]}"

echo "第三个元素为: ${my_array[2]}"

echo "第四个元素为: ${my_array[3]}"
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh

$ ./test.sh 第一个元素为: A 第二个元素为: B 第三个元素为: C 第四个元素为: D
```

获取数组中的所有元素

使用@ 或 * 可以获取数组中的所有元素，例如：

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com

my_array[0]=A

my_array[1]=B

my_array[2]=C

my_array[3]=D

echo "数组的元素为: ${my_array[*]}"

echo "数组的元素为: ${my_array[@]}"
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh

$ ./test.sh 数组的元素为: A B C D 数组的元素为: A B C D
```

获取数组的长度

获取数组长度的方法与获取字符串长度的方法相同，例如：

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com



my_array[0]=A

my_array[1]=B

my_array[2]=C

my_array[3]=D



echo "数组元素个数为: ${#my_array[*]}"

echo "数组元素个数为: ${#my_array[@]}"
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh

$ ./test.sh 数组元素个数为: 4 数组元素个数为: 4
```

6.9 shell 参数和特殊变量

有很多特殊变量是被 Shell 自动赋值的，我们已经遇到了$?和$1，现在总结一下：

在 shell 中我们会见到 $0、$1、$2 这样的符号，这是什么意思呢?

简单来说 $0 就是你写的 shell 脚本本身的名字，$1 是你给你写的 shell 脚本传的第一个参数，$2 是你给你写的 shell 脚本传的第二个参数

比如你新建了一个 shell 脚本 Test.sh, 内容如下：

```
#!/bin/bash

echo "shell 脚本本身的名字: $0"
```

```
echo "传给 shell 的第一个参数: $1"
echo "传给 shell 的第二个参数: $2"
```

保存退出后，你在 Test.sh 所在的目录下输入 bash Test.sh 1 2

结果为:

```
shell 脚本本身的名字: Test.sh
传给 shell 的第一个参数: 1
传给 shell 的第二个参数: 2
```

常用的位置参数和特殊变量

```
$0  相当于 C 语言 main 函数的 argv[0]
$1、$2...   这些称为位置参数（Positional Parameter），相当于 C 语言 main 函数的 argv[1]、argv[2]...
$#  相当于 C 语言 main 函数的 argc - 1，注意这里的#后面不表示注释
$@  表示参数列表"$1" "$2" ...，例如可以用在 for 循环中的 in 后面。
$*  表示参数列表"$1" "$2" ...，同上
$?  上一条命令的 Exit Status
$$  当前进程号
```

位置参数可以用 shift 命令左移。比如 shift 3 表示原来的$4 现在变成$1，原来的$5 现在变成$2 等等，原来的$1、$2、$3 丢弃，$0 不移动。不带参数的 shift 命令相当于 shift 1。例如：

```
#! /bin/sh
echo "The program $0 is now running"
```

```
    echo "The first parameter is $1"

    echo "The second parameter is $2"

    echo "The parameter list is $@"

    shift    echo "The first parameter is $1"

    echo "The second parameter is $2"

    echo "The parameter list is $@"
```

6.10 shell 输入输出

1.echo

echo 显示文本行或变量，或者把字符串输入到文件。

```
echo [option] string

-e 解析转义字符

-n 不回车换行。默认情况 echo 回显的内容后面跟一个回车换行。

echo "hello\n\n"

echo -e "hello\n\n"

echo  "hello"

echo -n "hello"
```

2.Prinft

上一章节我们学习了 Shell 的 echo 命令，本章节我们来学习 Shell 的另一个输出命令 printf。

printf 命令模仿 C 程序库（library）里的 printf() 程序。

printf 由 POSIX 标准所定义，因此使用 printf 的脚本比使用 echo 移植性好。

printf 使用引用文本或空格分隔的参数，外面可以在 printf 中使用格式化字符串，还可以制定字符串的宽度、左右对齐方式等。默认 printf 不会像 echo 自动添加换行符，我们可以手动添加 \n。

printf 命令的语法：

```
printf  format-string  [arguments...]
```

参数说明：

format-string: 为格式控制字符串

arguments: 为参数列表。

实例如下：

```
$ echo "Hello, Shell"Hello, Shell

$ printf "Hello, Shell\n"Hello, Shell

$
```

接下来,我来用一个脚本来体现 printf 的强大功能：

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com



printf "%-10s %-8s %-4s\n" 姓名 性别 体重kg

printf "%-10s %-8s %-4.2f\n" 郭靖 男 66.1234

printf "%-10s %-8s %-4.2f\n" 杨过 男 48.6543

printf "%-10s %-8s %-4.2f\n" 郭芙 女 47.9876
```

执行脚本，输出结果如下所示：

```
姓名     性别   体重kg 郭靖     男      66.12 杨过     男      48.65 郭芙     女      47.99
```

%s %c %d %f 都是格式替代符

%-10s 指一个宽度为 10 个字符（-表示左对齐，没有则表示右对齐），任何字符都会被显示在 10 个字符宽的字符内，如果不足则自动以空格填充，超过也会将内容全部显示出来。

%-4.2f 指格式化为小数，其中.2 指保留 2 位小数。

更多实例：

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com

 # format-string 为双引号

printf "%d %s\n" 1 "abc"

# 单引号与双引号效果一样

printf '%d %s\n' 1 "abc"

# 没有引号也可以输出

printf %s abcdef

# 格式只指定了一个参数，但多出的参数仍然会按照该格式输出，format-string 被重用

printf %s abc def



printf "%s\n" abc def



printf "%s %s %s\n" a b c d e f g h i j

# 如果没有 arguments，那么 %s 用 NULL 代替，%d 用 0 代替

printf "%s and %d \n"
```

执行脚本，输出结果如下所示：

```
1 abc1 abc

abcdefabcdefabcdef

a b c
```

```
d e f

g h i

j

 and 0
```

printf 的转义序列

序列	说明
\a	警告字符，通常为 ASCII 的 BEL 字符
\b	后退
\c	抑制（不显示）输出结果中任何结尾的换行字符（只在%b 格式指示符控制下的参数字符串中有效），而且，任何留在参数里的字符、任何接下来的参数以及任何留在格式字符串中的字符，都被忽略
\f	换页（formfeed）
\n	换行
\r	回车（Carriage return）
\t	水平制表符
\v	垂直制表符
\\	一个字面上的反斜杠字符
\ddd	表示 1 到 3 位数八进制值的字符。仅在格式字符串中有效
\0ddd	表示 1 到 3 位的八进制值字符

实例

```
$ printf "a string, no processing:<%s>\n" "A\nB"

a string, no processing:<A\nB>



$ printf "a string, no processing:<%b>\n" "A\nB"

a string, no processing:<A

B>
```

```
$ printf "www.runoob.com \a"

www.runoob.com $                    #不换行
```

3.管道

可以通过管道把一个命令的输出传递给另一个命令做输入。管道用竖线 | 表示。

```
cat myfile | more

ls -l | grep "myfile"

df -k | awk '{print $1}' | grep -v "文件系统"

df -k 查看磁盘空间，找到第一列，去除“文件系统”，并输出
```

4.tee

tee 命令把结果输出到标准输出，另一个副本输出到相应文件。

```
df -k | awk '{print $1}' | grep -v "文件系统" | tee a.txt



tee -a a.txt 表示追加操作。

df -k | awk '{print $1}' | grep -v "文件系统" | tee -a a.txt
```

5.文件重定向

```
cmd > file             把标准输出重定向到新文件中

cmd >> file            追加

cmd > file 2>&1         标准出错也重定向到 1 所指向的 file 里

cmd >> file 2>&1

cmd < file1 > file2     输入输出都定向到文件里

cmd < &fd              把文件描述符 fd 作为标准输入

cmd > &fd              把文件描述符 fd 作为标准输出
```

```
cmd < &-                    关闭标准输入
```

6.11 shell 函数

linux shell 可以用户定义函数，然后在 shell 脚本中可以随便调用。

shell 中函数的定义格式如下：

```
[ function ] funname [()]

{

    action;

    [return int;]

}
```

说明：

1、可以带 function fun() 定义，也可以直接 fun() 定义,不带任何参数。

2、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

下面的例子定义了一个函数并进行调用：

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com



demoFun(){

    echo "这是我的第一个 shell 函数!"}

echo "-----函数开始执行-----"

demoFun

echo "-----函数执行完毕-----"
```

输出结果：

```
-----函数开始执行-----这是我的第一个 shell 函数!-----函数执行完毕-----
```

下面定义一个带有 return 语句的函数：

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com



funWithReturn(){

    echo "这个函数会对输入的两个数字进行相加运算..."

    echo "输入第一个数字: "

    read aNum

    echo "输入第二个数字: "

    read anotherNum

    echo "两个数字分别为 $aNum 和 $anotherNum !"

    return $(($aNum+$anotherNum))}

funWithReturn

echo "输入的两个数字之和为 $? !"
```

输出类似下面：

```
这个函数会对输入的两个数字进行相加运算...输入第一个数字: 1 输入第二个数字: 2 两个数字分别为 1 和 2 !输入的两个数字之和为 3 !
```

函数返回值在调用该函数后通过 $? 来获得。

注意：所有函数在使用前必须定义。这意味着必须将函数放在脚本开始部分，直至 shell 解释器首次发现它时，才可以使用。调用函数仅使用其函数名即可。

函数参数

在 Shell 中，调用函数时可以向其传递参数。在函数体内部，通过 $n 的形式来获取参数的值，例如，$1 表示第一个参数，$2 表示第二个

参数...

带参数的函数示例：

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com



funWithParam(){

    echo "第一个参数为 $1 !"

    echo "第二个参数为 $2 !"

    echo "第十个参数为 $10 !"

    echo "第十个参数为 ${10} !"

    echo "第十一个参数为 ${11} !"

    echo "参数总数有 $# 个!"

    echo "作为一个字符串输出所有参数 $* !"}

funWithParam 1 2 3 4 5 6 7 8 9 34 73
```

输出结果：

```
第一个参数为 1 !第二个参数为 2 !第十个参数为 10 !第十个参数为 34 !第十一个参数为 73 !参数总数有 11 个!作为一个字符串输出所有参数 1 2 3 4 5 6 7 8 9 34 73 !
```

注意，$10 不能获取第十个参数，获取第十个参数需要${10}。当n>=10 时，需要使用${n}来获取参数。

另外，还有几个特殊字符用来处理参数：

参数处理	说明
$#	传递到脚本或函数的参数个数
$*	以一个单字符串显示所有向脚本传递的参数
$$	脚本运行的当前进程 ID 号
$!	后台运行的最后一个进程的 ID 号

$@	与$*相同，但是使用时加引号，并在引号中返回每个参数。
$-	显示 Shell 使用的当前选项，与 set 命令功能相同。
$?	显示最后命令的退出状态。0 表示没有错误，其他任何值表明有错误。

6.12 shell 脚本的调试

Shell 提供了一些用于调试脚本的选项，如下所示：

-n

读一遍脚本中的命令但不执行，用于检查脚本中的语法错误

-v

一边执行脚本，一边将执行过的脚本命令打印到标准错误输出

-x

提供跟踪执行信息，将执行的每一条命令和结果依次打印出来

使用这些选项有三种方法，一是在命令行提供参数

```
$ sh -x ./script.sh
```

二是在脚本开头提供参数

```
#! /bin/sh -x
```

第三种方法是在脚本中用 set 命令启用或禁用参数

```
#! /bin/sh

if [ -z "$1" ]; then
```

```
    set -x
    echo "ERROR: Insufficient Args."
    exit 1
    set +x
  fi
```

set -x 和 set +x 分别表示启用和禁用-x 参数，这样可以只对脚本中的某一段进行跟踪调试。

6.13 shell 正则表达式

以前我们用 grep 在一个文件中找出包含某些字符串的行，比如在头文件中找出一个宏定义。其实 grep 还可以找出符合某个模式（Pattern）的一类字符串。例如找出所有符合 xxxxx@xxxx.xxx 模式的字符串（也就是 email 地址），要求 x 字符可以是字母、数字、下划线、小数点或减号，email 地址的每一部分可以有一个或多个 x 字符，例如 abc.d@ef.com、1_2@987-6.54，当然符合这个模式的不全是合法的 email 地址，但至少可以做一次初步筛选，筛掉 a.b、c@d 等肯定不是 email 地址的字符串。再比如，找出所有符合 yyy.yyy.yyy.yyy 模式的字符串（也就是 IP 地址），要求 y 是 0-9 的数字，IP 地址的每一部分可以有 1-3 个 y 字符。

如果要用 grep 查找一个模式，如何表示这个模式，这一类字符串，而不是一个特定的字符串呢？从这两个简单的例子可以看出，要表示一个模式至少应该包含以下信息：

字符类（Character Class）：如上例的 x 和 y，它们在模式中表示一个字符，但是取值范围是一类字符中的任意一个。

数量限定符（Quantifier）： 邮件地址的每一部分可以有一个或多个 x 字符，IP 地址的每一部分可以有 1-3 个 y 字符

各种字符类以及普通字符之间的位置关系：例如邮件地址分三部分，用普通字符@和.隔开，IP 地址分四部分，用.隔开，每一部分都可以用字符类和数量限定符描述。为了表示位置关系，还有位置限定符（Anchor）的概念，将在下面介绍。

规定一些特殊语法表示字符类、数量限定符和位置关系，然后用这些特殊语法和普通字符一起表示一个模式，这就是正则表达式（Regular Expression）。例如 email 地址的正则表达式可以写成[a-zA-Z0-9_.-]+@[a-zA-Z0-9_.-]+.[a-zA-Z0-9_.-]+，IP 地址的正则表达式可以写成[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}。下一节介绍正则表达式的语法，我们先看看正则表达式在 grep 中怎么用。例如有这样一个文本文件 testfile：

```
192.168.1.1

1234.234.04.5678

123.4234.045.678

abcde
```

查找其中包含 IP 地址的行：

```
$ egrep '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' testfile

192.168.1.1
```

```
1234.234.04.5678
```

egrep 相当于 grep -E，表示采用 Extended 正则表达式语法。grep 的正则表达式有 Basic 和 Extended 两种规范，它们之间的区别下一节再解释。另外还有 fgrep 命令，相当于 grep -F，表示只搜索固定字符串而不搜索正则表达式模式，不会按正则表达式的语法解释后面的参数。

注意正则表达式参数用单引号括起来了，因为正则表达式中用到的很多特殊字符在 Shell 中也有特殊含义（例如\），只有用单引号括起来才能保证这些字符原封不动地传给 grep 命令，而不会被 Shell 解释掉。

192.168.1.1 符合上述模式，由三个.隔开的四段组成，每段都是 1 到 3 个数字，所以这一行被找出来了，可为什么 1234.234.04.5678 也被找出来了呢？因为 grep 找的是包含某一模式的行，这一行包含一个符合模式的字符串 234.234.04.567。相反，123.4234.045.678 这一行不包含符合模式的字符串，所以不会被找出来。

grep 是一种查找过滤工具，正则表达式在 grep 中用来查找符合模式的字符串。其实正则表达式还有一个重要的应用是验证用户输入是否合法，例如用户通过网页表单提交自己的 email 地址，就需要用程序验证一下是不是合法的 email 地址，这个工作可以在网页的 Javascript 中做，也可以在网站后台的程序中做，例如 PHP、Perl、Python、Ruby、Java 或 C，所有这些语言都支持正则表达式，可以说，目前不支持正则表达式的编程语言实在很少见。除了编程语言之外，很多 UNIX 命令和工具也都支持正则表达式，例如 grep、vi、sed、awk、emacs 等

等。“正则表达式”就像“变量”一样，它是一个广泛的概念，而不是某一种工具或编程语言的特性。

6.14 shell 文件包含

和其他语言一样，Shell 也可以包含外部脚本。这样可以很方便的封装一些公用的代码作为一个独立的文件。

Shell 文件包含的语法格式如下：

```
. filename   # 注意点号(.)和文件名中间有一空格

或

source filename
```

实例

创建两个 shell 脚本文件。

test1.sh 代码如下：

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com

url="http://www.runoob.com"
```

test2.sh 代码如下：

```
#!/bin/bash# author:菜鸟教程# url:www.runoob.com

#使用 . 号来引用 test1.sh 文件. ./test1.sh

# 或者使用以下包含文件代码# source ./test1.sh

echo "菜鸟教程官网地址：$url"
```

接下来，我们为 test2.sh 添加可执行权限并执行：

```
$ chmod +x test2.sh
```

```
$ ./test2.sh 菜鸟教程官网地址：http://www.runoob.com
```

第七章：linux 性能优化与内核参数优化

7.1 性能问题理论综述

系统的性能是指操作系统完成任务的有效性、稳定性和响应速度。Linux 系统管理员可能经常会遇到系统不稳定、响应速度慢等问题，例如在 linux 上搭建了一个 web 服务，经常出现网页无法打开、打开速度慢等现象，而遇到这些问题，就有人会抱怨 linux 系统不好，其实这些都是表面现象。操作系统完成一个任务时，与系统自身设置、网络拓朴结构、路由设备、路由策略、接入设备、物理线路等多个方面都密切相关，任何一个环节出现问题，都会影响整个系统的性能。因此当 linux 应用出现问题时，应当从应用程序、操作系统、服务器硬件、网络环境等方面综合排查，定位问题出现在哪个部分，然后集中解决。

在应用程序、操作系统、服务器硬件、网络环境等方面，影响性能最大的是应用程序和操作系统两个方面，因为这两个方面出现的问题不易察觉，隐蔽性很强。而硬件、网络方面只要出现问题，一般都能马上定位。下面主要讲解操作系统方面的性能调优思路，应用程序方面需要具体问题具体对待。

7.2 系统硬件资源

7.2.1 CPU

CPU 是操作系统稳定运行的根本，CPU 的速度与性能在很大程度上决定了系统整体的性能，因此，CPU 数量越多、主频越高，服务器性能也就相对越好。但事实并非完全如此。

目前大部分 CPU 在同一时间内只能运行一个线程，超线程的处理器可以在同一时间运行多个线程，因此，可以利用处理器的超线程特性提高系统性能。在 Linux 系统下，只有运行 SMP 内核才能支持超线程，但是，安装的 CPU 数量越多，从超线程获得的性能方面的提高就越少。另外，Linux 内核会把多核的处理器当作多个单独的 CPU 来识别，例如两个 4 核的 CPU，在 Lnux 系统下会被当作 8 个单核 CPU。但是从性能角度来讲，两个 4 核的 CPU 和 8 个单核的 CPU 并不完全等价，根据权威部门得出的测试结论，前者的整体性能要比后者低 25％~30％。

可能出现 CPU 瓶颈的应用有 db 服务器、动态 Web 服务器等，对于这类应用，要把 CPU 的配置和性能放在主要位置。

7.2.2 内存

内存的大小也是影响 Linux 性能的一个重要的因素，内存太小，系统进程将被阻塞，应用也将变得缓慢，甚至失去响应；内存太大，导致资源浪费。Linux 系统采用了物理内存和虚拟内存两种方式，虚拟内存虽然可以缓解物理内存的不足，但是占用过多的虚拟内存，应用程

序的性能将明显下降，要保证应用程序的高性能运行，物理内存一定要足够大；但是过大的物理内存，会造成内存资源浪费，例如，在一个 32 位处理器的 Linux 操作系统上，超过 8GB 的物理内存都将被浪费。因此，要使用更大的内存，建议安装 64 位的操作系统，同时开启 Linux 的大内存内核支持。

由于处理器寻址范围的限制，在 32 位 Linux 操作系统上，应用程序单个进程最大只能使用 4GB 的内存，这样以来，即使系统有更大的内存，应用程序也无法“享”用，解决的办法就是使用 64 位处理器，安装 64 位操作系统。在 64 位操作系统下，可以满足所有应用程序对内存的使用需求 ，几乎没有限制。

可能出现内存性能瓶颈的应用有 NOSQL 服务器、数据库服务器、缓存服务器等，对于这类应用要把内存大小放在主要位置。

7.2.3 磁盘 I/O 性能

磁盘的 I/O 性能直接影响应用程序的性能，在一个有频繁读写的应用中，如果磁盘 I/O 性能得不到满足，就会导致应用停滞。好在现今的磁盘都采用了很多方法来提高 I/O 性能，比如常见的磁盘 RAID 技术。

通过 RAID 技术组成的磁盘组，就相当于一个大硬盘，用户可以对它进行分区格式化、建立文件系统等操作，跟单个物理硬盘一模一样，唯一不同的是 RAID 磁盘组的 I/O 性能比单个硬盘要高很多，同时在数据的安全性也有很大提升。

根据磁盘组合方式的不同，RAID 可以分为 RAID0，RAID1、RAID2、RAID3、RAID4、RAID5、RAID6、RAID7、RAID0+1、RAID10 等级别，常用的 RAID 级别有 RAID0、RAID1、RAID5、RAID0+1，这里进行简单介绍。

RAID 0：通过把多块硬盘粘合成一个容量更大的硬盘组，提高了磁盘的性能和吞吐量。这种方式成本低，要求至少两个磁盘，但是没有容错和数据修复功能，因而只能用在对数据安全性要求不高的环境中。

RAID 1：也就是磁盘镜像，通过把一个磁盘的数据镜像到另一个磁盘上，最大限度地保证磁盘数据的可靠性和可修复性，具有很高的数据冗余能力，但磁盘利用率只有 50%，因而，成本最高，多用在保存重要数据的场合。

RAID5：采用了磁盘分段加奇偶校验技术，从而提高了系统可靠性，RAID5 读出效率很高，写入效率一般，至少需要 3 块盘。允许一块磁盘故障，而不影响数据的可用性。

RAID0+1：把 RAID0 和 RAID1 技术结合起来就成了 RAID0+1，至少需要 4 个硬盘。此种方式的数据除分布在多个盘上外，每个盘都有其镜像盘，提供全冗余能力，同时允许一个磁盘故障，而不影响数据可用性，并具有快速读/写能力。

通过了解各个 RAID 级别的性能，可以根据应用的不同特性，选择适合自身的 RAID 级别，从而保证应用程序在磁盘方面达到最优性能。

4．网络宽带

Linux 下的各种应用，一般都是基于网络的，因此网络带宽也是影响性能的一个重要因素，低速的、不稳定的网络将导致网络应用程序的访问阻塞，而稳定、高速的网络带宽，可以保证应用程序在网络上畅通无阻地运行。幸运的是，现在的网络一般都是千兆带宽或光纤网络，带宽问题对应用程序性能造成的影响也在逐步降低。

7.3 操作系统相关资源

基于操作系统的性能优化也是多方面的，可以从系统安装、系统内核参数、网络参数、文件系统等几个方面进行衡量，下面依次进行简单介绍。

7.3.1 系统安装优化

系统优化可以从安装操作系统开始，当安装 Linux 系统时，磁盘的划分，SWAP 内存的分配都直接影响以后系统的运行性能，例如，磁盘分配可以遵循应用的需求：对于对写操作频繁而对数据安全性要求不高的应用，可以把磁盘做成 RAID 0；而对于对数据安全性较高，对读写没有特别要求的应用，可以把磁盘做成 RAID 1；对于对读操作要求较高，而对写操作无特殊要求，并要保证数据安全性的应用，可以选择 RAID 5；对于对读写要求都很高，并且对数据安全性要求也很高的应用，可以选择 RAID10/01。这样通过不同的应用需求设置不同的 RAID 级别，在磁盘底层对系统进行优化操作。

随着内存价格的降低和内存容量的日益增大，对虚拟内存 SWAP 的设定，现在已经没有了所谓虚拟内存是物理内存两倍的要求，但是 SWAP 的设定还是不能忽略，根据经验，如果内存较小（物理内存小于 4GB），一般设置 SWAP 交换分区大小为内存的 2 倍；如果物理内存大于 8GB 小于 16GB，可以设置 SWAP 大小等于或略小于物理内存即可；如果内存大小在 16GB 以上，原则上可以设置 SWAP 为 0，但并不建议这么做，因为设置一定大小的 SWAP 还是有一定作用的。在我们大数据环境中，一般搭建大数据集群的时候都会禁用 swap 分区

7.3.2 内核参数优化

系统安装完成后，优化工作并没有结束，接下来还可以对系统内核参数进行优化，不过内核参数的优化要和系统中部署的应用结合起来整体考虑。例如，如果系统部署的是 Oracle 数据库应用，那么就需要对系统共享内存段（kernel.shmmax、kernel.shmmni、kernel.shmall）、系统信号量（kernel.sem）、文件句柄（fs.file-max）等参数进行优化设置；如果部署的是 Web 应用，那么就需要根据 Web 应用特性进行网络参数的优化，例如修改 net.ipv4.ip_local_port_range、net.ipv4.tcp_tw_reuse、net.core.somaxconn 等网络内核参数。

7.3.3 文件系统优化

文件系统的优化也是系统资源优化的一个重点，在 Linux 下可选的文件系统有 ext2、ext3、ReiserFS、ext4、xfs，根据不同的应用，选择不同的文件系统。

Linux 标准文件系统是从 VFS 开始的，然后是 ext，接着就是 ext2，应该说，ext2 是 Linux 上标准的文件系统，ext3 是在 ext2 基础上增加日志形成的，从 VFS 到 ext4，其设计思想没有太大变化，都是早期 UNIX 家族基于超级块和 inode 的设计理念。

XFS 文件系统是一个高级日志文件系统，XFS 通过分布处理磁盘请求、定位数据、保持 Cache 的一致性来提供对文件系统数据的低延迟、高带宽的访问，因此，XFS 极具伸缩性，非常健壮，具有优秀的日志记录功能、可扩展性强、快速写入性能等优点。

目前服务器端 ext4 和 xfs 是主流文件系统，如何选择合适的文件系统，需要根据文件系统的特点加上业务的需求综合来定。

7.4 centos7 内核参数优化

众所周知在默认参数情况下 Linux 对高并发支持并不好，主要受限于单进程最大打开文件数限制、内核 TCP 参数方面和 IO 事件分配机制等。下面就从几方面来调整使 Linux 系统能够支持高并发环境。

7.4.1 Iptables 相关

如非必须，关掉或卸载 iptables 防火墙，并阻止 kernel 加载 iptables 模块。这些模块会影响并发性能。

7.4.2. 单进程最大打开文件数限制

一般的发行版，限制单进程最大可以打开 1024 个文件，这是远远不能满足高并发需求的，调整过程如下：

在#号提示符下敲入：

ulimit －n 65535

将 root 启动的单一进程的最大可以打开的文件数设置为 65535 个。如果系统回显类似于“Operationnotpermitted”之类的话，说明上述限制修改失败，实际上是因为在中指定的数值超过了 Linux 系统对该用户打开文件数的软限制或硬限制。因此，就需要修改 Linux 系统对用户的关于打开文件数的软限制和硬限制。

第一步，修改 limits.conf 文件，并添加：

```
# vim /etc/security/limits.conf

* softnofile 65536
* hard nofile65536
```

其中’*'号表示修改所有用户的限制；soft 或 hard 指定要修改软限制还是硬限制；65536 则指定了想要修改的新的限制值，即最大打开文件数(请注意软限制值要小于或等于硬限制)。修改完后保存文件。

第二步，修改/etc/pam.d/login 文件，在文件中添加如下行：

```
vim /etc/pam.d/login

sessionrequired /lib/security/pam_limits.so
```

这是告诉 Linux 在用户完成系统登录后，应该调用 pam_limits.so 模块来设置系统对该用户可使用的各种资源数量的最大限制(包括用户可打开的最大文件数限制)，而 pam_limits.so 模块就会从 /etc/security/limits.conf 文件中读取配置来设置这些限制值。修改完后保存此文件。

第三步，查看 Linux 系统级的最大打开文件数限制，使用如下命令：

```
cat/proc/sys/fs/file-max

32568
```

这表明这台 Linux 系统最多允许同时打开(即包含所有用户打开文件数总和)32568 个文件，是 Linux 系统级硬限制，所有用户级的打开文件数限制都不应超过这个数值。通常这个系统级硬限制是 Linux 系统在启动时根据系统硬件资源状况计算出来的最佳的最大同时打开文件数限制，如果没有特殊需要，不应该修改此限制，除非想为用户级打开文件数限制设置超过此限制的值。修改此硬限制的方法是修改 /etc/sysctl.conf 文件内 fs.file-max= 131072

这是让 Linux 在启动完成后强行将系统级打开文件数硬限制设置为 131072。修改完后保存此文件。

完成上述步骤后重启系统，一般情况下就可以将 Linux 系统对指定用户的单一进程允许同时打开的最大文件数限制设为指定的数值。如果重启后用 ulimit-n 命令查看用户可打开文件数限制仍然低于上述步骤中设置的最大值，这可能是因为在用户登录脚本/etc/profile 中使用 ulimit-n 命令已经将用户可同时打开的文件数做了限制。由于通过 ulimit-n 修改系统对用户可同时打开文件的最大数限制时，新修改的值只能小于或等于上次 ulimit-n 设置的值，因此想用此命令增大这个限制值是不可能的。所以，如果有上述问题存在，就只能去打开 /etc/profile 脚本文件，在文件中查找是否使用了 ulimit-n 限制了用户可同时打开的最大文件数量，如果找到，则删除这行命令，或者将其

设置的值改为合适的值，然后保存文件，用户退出并重新登录系统即可。

通过上述步骤，就为支持高并发 TCP 连接处理的通讯处理程序解除关于打开文件数量方面的系统限制。

7.4.3 内核 TCP 参数方面

Linux 系统下，TCP 连接断开后，会以 TIME_WAIT 状态保留一定的时间，然后才会释放端口。当并发请求过多的时候，就会产生大量的 TIME_WAIT 状态的连接，无法及时断开的话，会占用大量的端口资源和服务器资源。这个时候我们可以优化 TCP 的内核参数，来及时将 TIME_WAIT 状态的端口清理掉。

下面介绍的方法只对拥有大量 TIME_WAIT 状态的连接导致系统资源消耗有效，如果不是这种情况下，效果可能不明显。可以使用 netstat 命令去查 TIME_WAIT 状态的连接状态，输入下面的组合命令，

查看当前 TCP 连接的状态和对应的连接数量：

```
# netstat-n | awk   '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'

#这个命令会输出类似下面的结果：
LAST_ACK16
SYN_RECV348
ESTABLISHED70
FIN_WAIT1229
FIN_WAIT230
CLOSING33
TIME_WAIT18098
```

我们只用关心 TIME_WAIT 的个数，在这里可以看到，有 18000 多个 TIME_WAIT，这样就占用了 18000 多个端口。要知道端口的数量只有 65535 个，占用一个少一个，会严重的影响到后继的新连接。这种情况下，我们就有必要调整下 Linux 的 TCP 内核参数，让系统更快的释放 TIME_WAIT 连接。

编辑配置文件:/etc/sysctl.conf，在这个文件中，加入下面的几行内容：

```
vim /etc/sysctl.conf

net.ipv4.tcp_syncookies= 1    #表示开启 SYNCookies。当出现 SYN 等待队列溢出时，启用 cookies 来处理，可防范少量 SYN 攻击，默认为 0，表示关闭；
net.ipv4.tcp_tw_reuse= 1      #表示开启重用。允许将 TIME-WAITsockets 重新用于新的 TCP 连接，默认为 0，表示关闭；
net.ipv4.tcp_tw_recycle= 1    #表示开启 TCP 连接中 TIME-WAITsockets 的快速回收，默认为 0，表示关闭；
net.ipv4.tcp_fin_timeout= 30 #修改系统默认的 TIMEOUT 时间。
```

#输入下面的命令，让内核参数生效：

```
sysctl-p
```

在经过这样的调整之后，除了会进一步提升服务器的负载能力之外，还能够防御小流量程度的 DoS、CC 和 SYN 攻击。

此外，如果你的连接数本身就很多，我们可以再优化一下 TCP 的可使用端口范围，进一步提升服务器的并发能力。依然是往上面的参数文件中，加入下面这些配置

```
net.ipv4.tcp_keepalive_time= 1200      #表示当 keepalive 起用的时候，TCP 发送 keepalive 消息的频度。缺省是 2 小时，改为 20 分钟。
net.ipv4.ip_local_port_range= 1024 65535   #表示用于向外连接的端口范围。缺省情况下很小，改为 1024 到 65535。
net.ipv4.tcp_max_syn_backlog= 8192   #表示 SYN 队列的长度，默认为 1024，加大队列长度为 8192，可以容纳更多等待连接的网络连接数。
net.ipv4.tcp_max_tw_buckets= 5000    #表示系统同时保持 TIME_WAIT 的最大数量，如果超过这个数字，TIME_WAIT 将立刻被清除并打印警告信息。默认为 180000，改为 5000。此项参
```

数可以控制 TIME_WAIT 的最大数量，只要超出了。

7.4.4 内核其他 TCP 参数说明

net.ipv4.tcp_max_syn_backlog= 65536　　#记录的那些尚未收到客户端确认信息的连接请求的最大值。对于有 128M 内存的系统而言，缺省值是 1024，小内存的系统则是 128。
net.core.netdev_max_backlog= 32768　　#每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数目。
net.core.somaxconn= 32768　　#例如 web 应用中 listen 函数的 backlog 默认会给我们内核参数的 net.core.somaxconn 限制到 128，而 nginx 定义的 NGX_LISTEN_BACKLOG 默认为 511，所以有必要调整这个值。
net.core.wmem_default= 8388608
net.core.rmem_default= 8388608
net.core.rmem_max= 16777216 #最大 socket 读 buffer,可参考的优化值:873200
net.core.wmem_max= 16777216 #最大 socket 写 buffer,可参考的优化值:873200
net.ipv4.tcp_timestsmps= 0　#时间戳可以避免序列号的卷绕。一个 1Gbps 的链路肯定会遇到以前用过的序列号。时间戳能够让内核接受这种“异常”的数据包。这里需要将其关掉。
net.ipv4.tcp_synack_retries= 2　　#为了打开对端的连接，内核需要发送一个 SYN 并附带一个回应前面一个 SYN 的 ACK。也就是所谓三次握手中的第二次握手。这个设置决定了内核放弃连接之前发送 SYN+ACK 包的数量。
net.ipv4.tcp_syn_retries= 2　#在内核放弃建立连接之前发送 SYN 包的数量。
#net.ipv4.tcp_tw_len= 1
net.ipv4.tcp_tw_reuse= 1　　#开启重用。允许将 TIME-WAITsockets 重新用于新的 TCP 连接。
net.ipv4.tcp_wmem= 8192 436600 873200　#TCP 写 buffer,可参考的优化值:8192 436600 873200

net.ipv4.tcp_rmem = 32768 436600 873200 #TCP 读 buffer,可参考的优化值:32768 436600 873200

net.ipv4.tcp_mem= 94500000 91500000 92700000

#同样有 3 个值,意思是:
#net.ipv4.tcp_mem[0]:低于此值，TCP 没有内存压力。
#net.ipv4.tcp_mem[1]:在此值下，进入内存压力阶段。
#net.ipv4.tcp_mem[2]:高于此值，TCP 拒绝分配 socket。
#上述内存单位是页，而不是字节。可参考的优化值是:7864321048576 1572864

net.ipv4.tcp_max_orphans= 3276800
#系统中最多有多少个 TCP 套接字不被关联到任何一个用户文件句柄上,如果超过这个数字，连接将即刻被复位并打印出警告信息,这个限制仅仅是为了防止简单的 DoS 攻击，不能过分依靠它或者人为地减小这个值，更应该增加这个值(如果增加了内存之后)。

net.ipv4.tcp_fin_timeout= 30
#如果套接字由本端要求关闭，这个参数决定了它保持在 FIN-WAIT-2 状态的时间。对端可以

出错并永远不关闭连接，甚至意外当机。缺省值是 60 秒。2.2 内核的通常值是 180 秒，你可以按这个设置，但要记住的是，即使你的机器是一个轻载的 WEB 服务器，也有因为大量的死套接字而内存溢出的风险，FIN-WAIT-2 的危险性比 FIN-WAIT-1 要小，因为它最多只能吃掉 1.5K 内存，但是它们的生存期长些。

同时还涉及到一个 TCP 拥塞算法的问题，你可以用下面的命令查看本机提供的拥塞算法控制模块：

```
$ sysctl net.ipv4.tcp_available_congestion_control

#对于几种算法的分析，详情可以参考下：TCP 拥塞控制算法的优缺点、适用环境、性能分析，比如

高延时可以试用 hybla，中等延时可以试用 htcp 算法等。

net.ipv4.tcp_congestion_control=hybla     #如果想设置 TCP 拥塞算法为 hybla

net.ipv4.tcp_fastopen= 3     #额外的，对于内核版高于于 3.7.1 的，我们可以开启 tcp_fastopen
```

7.4.5 IO 事件分配机制

在 Linux 启用高并发 TCP 连接，必须确认应用程序是否使用了合适的网络 I/O 技术和 I/O 事件分派机制。可用的 I/O 技术有同步 I/O，非阻塞式同步 I/O，以及异步 I/O。在高 TCP 并发的情形下，如果使用同步 I/O，这会严重阻塞程序的运转，除非为每个 TCP 连接的 I/O 创建一个线程。但是，过多的线程又会因系统对线程的调度造成巨大开销。

因此，在高 TCP 并发的情形下使用同步 I/O 是不可取的，这时可以考虑使用非阻塞式同步 I/O 或异步 I/O。非阻塞式同步 I/O 的技术包括使用 select()，poll()，epoll 等机制。异步 I/O 的技术就是使用 AIO。

从 I/O 事件分派机制来看，使用 select()是不合适的，因为它所支持的并发连接数有限(通常在 1024 个以内)。如果考虑性能，poll()也是不

合适的，尽管它可以支持的较高的 TCP 并发数，但是由于其采用“轮询”机制，当并发数较高时，其运行效率相当低，并可能存在 I/O 事件分派不均，导致部分 TCP 连接上的 I/O 出现“饥饿”现象。

而如果使用 epoll 或 AIO，则没有上述问题(早期 Linux 内核的 AIO 技术实现是通过在内核中为每个 I/O 请求创建一个线程来实现的，这种实现机制在高并发 TCP 连接的情形下使用其实也有严重的性能问题。但在最新的 Linux 内核中，AIO 的实现已经得到改进)。

综上所述，在开发支持高并发 TCP 连接的 Linux 应用程序时，应尽量使用 epoll 或 AIO 技术来实现并发的 TCP 连接上的 I/O 控制，这将为提升程序对高并发 TCP 连接的支持提供有效的 I/O 保证。

经过这样的优化配置之后，服务器的 TCP 并发处理能力会显著提高。以上配置仅供参考，用于生产环境请根据自己的实际情况调整观察再调整。

7.5 安装大数据组件环境基础优化

1. 系统盘 raid1，存储盘 raid0，hdfs 三备份不用 raid 造成冗余，若对元数据安全系数特别高，namenode 元数据所在磁盘可做 raid1 双备份。
2. 关闭硬件节能模式（bios）
3. 双网卡模式选择 bound6(平衡负载)
4. 存储盘多块，减少磁盘 IO

5. 禁用 swap

6. 时间同步

7. 关闭 selinux

8. 关闭防火墙 iptables

9. 将 root 启动的单一进程的最大可以打开的文件数设置为 65535 个（ulimit－n 65535）

第八章: linux 实用故障排查

8.1 排查思路

大数据集群运行在 linux 系统上总会遇见各种各样的问题，我们要定位问题，基本从这几个方面入手排查：cpu, 内存，磁盘 IO，网络，GC 等。

8.2 cpu

一些概念：多核，超线程 ，CPU 频率（2.2GHZ）

（节能模式，普通模式，超能模式，bios 里设置，搭集群要注意下这个参数尽量关闭节能模式）bios 里可以关闭。

#1 查看物理 CPU 个数

cat /proc/cpuinfo |grep "physical id"|sort |uniq|wc -l

#2 查看逻辑 cpu 个数

cat /proc/cpuinfo |grep "processor"|wc -l

#3 查看 CPU 多少核

```
processor       : 23
vendor_id       : GenuineIntel
cpu family      : 6
model           : 85
model name      : Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz
stepping        : 4
microcode       : 0x1
cpu MHz         : 2099.998
cache size      : 16384 KB
physical id     : 0
siblings        : 24
core id         : 23
cpu cores       : 24
apicid          : 23
initial apicid  : 23
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_
tsc arch_perfmon rep_good nopl xtopology eagerfpu pni pclmulqdq ssse3 fma cx16 p
```

#4 在生产集群中我们通常通过 top 来查看 cpu 的使用率来判断系统的负载情况，Top 然后按 1，可以看到 cpu 的使用率

```
top - 15:47:44 up 20:10,  3 users,  load average: 16.02, 15.84, 14.15
Tasks: 281 total,   1 running, 279 sleeping,   1 stopped,   0 zombie
%Cpu0  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 99.7 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.3 st
%Cpu5  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 99.7 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.3 st
%Cpu7  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  6.3 us, 12.3 sy,  0.0 ni, 77.5 id,  0.0 wa,  0.0 hi,  0.0 si,  3.9 st
%Cpu9  :  5.7 us, 10.3 sy,  0.0 ni, 79.1 id,  0.0 wa,  0.0 hi,  0.0 si,  5.0 st
%Cpu10 :  6.1 us, 11.1 sy,  0.0 ni, 78.9 id,  0.0 wa,  0.0 hi,  0.0 si,  3.9 st
%Cpu11 :  5.7 us, 11.4 sy,  0.0 ni, 78.6 id,  0.0 wa,  0.0 hi,  0.4 si,  3.9 st
%Cpu12 :  6.4 us, 10.4 sy,  0.0 ni, 78.9 id,  0.0 wa,  0.0 hi,  0.4 si,  3.9 st
%Cpu13 :  4.3 us, 11.1 sy,  0.0 ni, 80.4 id,  0.0 wa,  0.0 hi,  0.0 si,  4.3 st
%Cpu14 :  6.0 us, 11.0 sy,  0.0 ni, 79.0 id,  0.0 wa,  0.0 hi,  0.0 si,  3.9 st
%Cpu15 :  4.7 us, 11.2 sy,  0.0 ni, 80.1 id,  0.0 wa,  0.0 hi,  0.0 si,  4.0 st
%Cpu16 :  5.7 us, 11.0 sy,  0.0 ni, 79.4 id,  0.0 wa,  0.0 hi,  0.0 si,  3.9 st
%Cpu17 :  6.1 us, 11.5 sy,  0.0 ni, 79.2 id,  0.0 wa,  0.0 hi,  0.0 si,  3.2 st
%Cpu18 :  6.1 us, 10.5 sy,  0.0 ni, 79.4 id,  0.0 wa,  0.0 hi,  0.0 si,  4.0 st
%Cpu19 :  5.5 us, 10.5 sy,  0.0 ni, 81.1 id,  0.0 wa,  0.0 hi,  0.0 si,  2.9 st
%Cpu20 :  5.1 us, 11.2 sy,  0.0 ni, 80.1 id,  0.0 wa,  0.0 hi,  0.0 si,  3.6 st

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
```

8.3 内存

#1，常见内存大小 64 G---128G--256G---512G，通过 free -g 来查看系统内存是否不足

```
[root@node02 ~]# free -g
              total        used        free      shared  buff/cache   available
Mem:             94          25          43           0          25          68
Swap:             0           0           0
[root@node02 ~]#
```

#2 查看内存详细内容：

```
[root@node02 ~]# cat /proc/meminfo
MemTotal:       98833404 kB
MemFree:        45170012 kB
MemAvailable:   71603220 kB
Buffers:            2088 kB
Cached:         26761980 kB
SwapCached:            0 kB
Active:         27143772 kB
Inactive:       25151524 kB
Active(anon):   25536076 kB
Inactive(anon):    42076 kB
Active(file):    1607696 kB
Inactive(file): 25109448 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:               428 kB
Writeback:             0 kB
AnonPages:      25524004 kB
Mapped:           175560 kB
```

8.4 磁盘 IO

#1 磁盘种类：

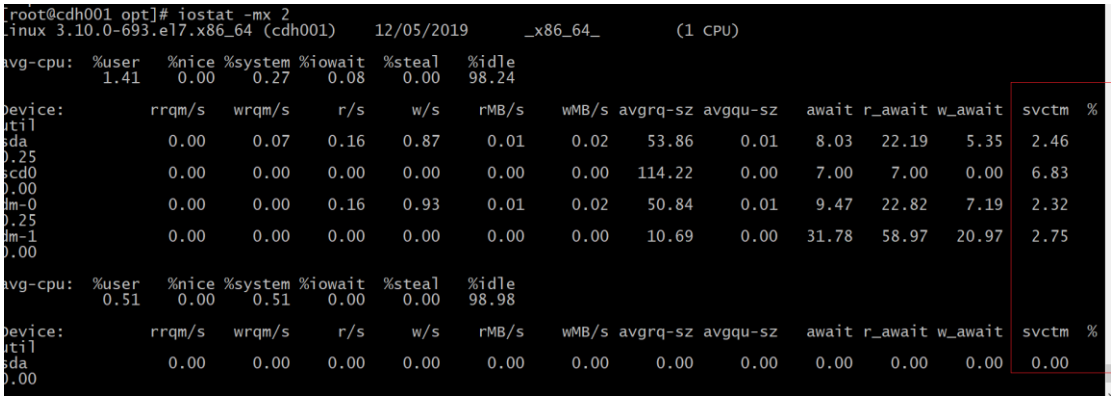
sata(150M/s 左右)　sas（300 M/S 左右）　　ssd（最快也最贵）

一般磁盘 2T-4T,服务器支持的最大存储也不同，比较常见的 48T，一般服务器可以支持 12-24 块盘。

#2 磁盘 io 的查看：可以定位是否是因为 io 过大导致性能下降

iostat -mx 2

如果没有这个命令：yum　install -y sysstat



```
root@cdh001 opt]# iostat -mx 2
inux 3.10.0-693.el7.x86_64 (cdh001)     12/05/2019      _x86_64_        (1 CPU)

vg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1.41    0.00    0.27    0.08    0.00   98.24

evice:         rrqm/s   wqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %
til
da               0.00    0.07    0.16    0.87     0.01     0.02    53.86     0.01    8.03   22.19    5.35   2.46
.25
cd0              0.00    0.00    0.00    0.00     0.00     0.00   114.22     0.00    7.00    7.00    0.00   6.83
.00
m-0              0.00    0.00    0.16    0.93     0.01     0.02    50.84     0.01    9.47   22.82    7.19   2.32
.25
m-1              0.00    0.00    0.00    0.00     0.00     0.00    10.69     0.00   31.78   58.97   20.97   2.75
.00

vg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.51    0.00    0.51    0.00    0.00   98.98

evice:         rrqm/s   wqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %
til
da               0.00    0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00
.00
```

#3 df -h 查看磁盘空间

```
[root@tdh003 ~]# df -h
Filesystem                      Size  Used Avail Use% Mounted on
/dev/mapper/centos-root          50G  6.2G   44G  13% /
devtmpfs                        1.9G     0  1.9G   0% /dev
tmpfs                           1.9G     0  1.9G   0% /dev/shm
tmpfs                           1.9G  9.1M  1.9G   1% /run
tmpfs                           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/sda1                      1014M  179M  836M  18% /boot
/dev/mapper/centos-mnt_disk1    5.0G   33M  5.0G   1% /mnt/disk1
/dev/mapper/centos-mnt_disk2    5.0G   33M  5.0G   1% /mnt/disk2
/dev/mapper/centos-var_log       10G  339M  9.7G   4% /var/log
/dev/mapper/centos-var_lib_docker  29G  7.4G   22G  26% /var/lib/docker
tmpfs                           378M  8.0K  378M   1% /run/user/42
tmpfs                           378M     0  378M   0% /run/user/0
[root@tdh003 ~]#
```

#4 df -T 查看磁盘的格式化的格式，主流一般是 xfs 和 ext4，系统盘默认是 xfs，后续添加数据盘尽量也是 xfs 格式这样统一比较好，也方便写脚本批量挂盘

```
root@tdh003 ~]# df -T
ilesystem                     Type     1K-blocks    Used Available Use% Mounted on
dev/mapper/centos-root        xfs       52403200 6385540  46017660  13% /
evtmpfs                       devtmpfs   1916740       0   1916740   0% /dev
mpfs                          tmpfs      1932652       0   1932652   0% /dev/shm
mpfs                          tmpfs      1932652    9660   1922992   1% /run
mpfs                          tmpfs      1932652       0   1932652   0% /sys/fs/cgroup
dev/sda1                      xfs        1038336  182280    856056  18% /boot
dev/mapper/centos-mnt_disk1   xfs        5232640   33056   5199584   1% /mnt/disk1
dev/mapper/centos-mnt_disk2   xfs        5232640   33056   5199584   1% /mnt/disk2
dev/mapper/centos-var_log     xfs       10475520  346912  10128608   4% /var/log
dev/mapper/centos-var_lib_docker xfs    30385668 7582148  22803520  25% /var/lib/docker
mpfs                          tmpfs       386532       8    386524   1% /run/user/42
mpfs                          tmpfs       386532       0    386532   0% /run/user/0
verlay                        overlay   30385668 7582148  22803520  25% /var/lib/docker/overl
3383358419694a4611a01be16c71604be252dde29486e1f/merged
```

#5 lsblk 能够查看盘与分区以及 ssd 盘，用 fdisk 查看盘可能识别不到

ssd

```
[root@tdh003 ~]#
[root@tdh003 ~]# lsblk
NAME                  MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                     8:0    0  100G  0 disk
├─sda1                  8:1    0    1G  0 part /boot
└─sda2                  8:2    0   99G  0 part
  ├─centos-root       253:0    0   50G  0 lvm  /
  ├─centos-var_lib_docker 253:1  0   29G  0 lvm  /var/lib/docker
  ├─centos-var_log    253:2    0   10G  0 lvm  /var/log
  ├─centos-mnt_disk1  253:3    0    5G  0 lvm  /mnt/disk1
  └─centos-mnt_disk2  253:4    0    5G  0 lvm  /mnt/disk2
sr0                    11:0    1  4.2G  0 rom
[root@tdh003 ~]#
```

#6 fdisk -l 查看磁盘与未挂载的磁盘和分区信息

```
[root@tdh003 ~]# fdisk -l

Disk /dev/sda: 107.4 GB, 107374182400 bytes, 209715200 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000d2f30

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048     2099199     1048576   83  Linux
/dev/sda2         2099200   209715199   103808000   8e  Linux LVM

Disk /dev/mapper/centos-root: 53.7 GB, 53687091200 bytes, 104857600 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/mapper/centos-var_lib_docker: 31.1 GB, 31130124288 bytes, 60801024 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
```

#7 cat /etc/fstab 查看磁盘挂载信息，新加磁盘要手动永久挂在需要在这个配置文件里添加，6 列。如果这个文件写错了，重启服务器正常模式下是启动不了的！！！

```
[root@tdh003 ~]# cat /etc/fstab

#
# /etc/fstab
# Created by anaconda on Tue Dec  3 18:14:12 2019
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/centos-root /                       xfs     defaults        0 0
UUID=13e43664-ce1f-48e4-8d9d-a8faaf36295f /boot                   xfs     defaults        0 0
/dev/mapper/centos-mnt_disk1 /mnt/disk1             xfs     defaults        0 0
/dev/mapper/centos-mnt_disk2 /mnt/disk2             xfs     defaults        0 0
/dev/mapper/centos-var_lib_docker /var/lib/docker       xfs     defaults        0 0
/dev/mapper/centos-var_log /var/log                 xfs     defaults        0 0
ramfs                   /mnt/ramdisk/ngmr       ramfs   mode=777,size=4g        0 0
/mnt/disk1/hadoop/data    /transwarp/mounts/hdfs1/mnt/disk1/hadoop/data    none    bind    0    0
/mnt/disk2/hadoop/data    /transwarp/mounts/hdfs1/mnt/disk2/hadoop/data    none    bind    0    0
/guardian/txsql/data/    /transwarp/mounts/guardian/guardian/txsql/data/    none    bind    0    0
[root@tdh003 ~]#
```

#8 Lvm：逻辑卷管理（了解）

PV: 物理卷:硬盘和分区

VG：卷组，多个 PV

LV： 逻辑卷，多个 VG

实际需求应用：

1.集群磁盘扩容加盘，要永久挂载（信息写入/etc/fstab）

2.根目录空间不够要给根目录扩容(lvm)

https://www.cnblogs.com/himismad/p/7851548.html
http://heylinux.com/archives/3392.html

#9 磁盘阵列 （bios 里设置）常见：

raid0（无冗余），raid1（双备份），raid5（n-1/n,损失其中一块盘的空间）等等

8.5 网络

#1 查看是否 ping 通:ping IP

#2 查看某端口是否被监听

示例：比如你起启动 datanode 没有启动，查看日志报错：Address in used ，翻译过来就是端口被占用，那你得查看下这个端口被哪个程序占用了，就用 netatat -anp|grep 端口号，

就可以知道它的进程，比如下图进程 pid 就是 107484，就可以用 kill -9 PID 强制杀死

```
[root@cdh001 network-scripts]# netstat -anp |grep 50070
tcp        0      0 192.168.52.110:50070    0.0.0.0:*               LISTEN      107484/java
[root@cdh001 network-scripts]#
```

#3 网卡模式配置（bound6，负载均衡）

BONDING_OPTS="miimon=100 mode=1"（配置Bond的核心语句，mod1 为主备模式）

ifcfg-bond0文件的配置

网卡Bond是通过把多张网卡绑定为一个逻辑网卡，实现本地网卡的冗余，带宽扩容和负载均衡。通过Bond技术让多块网卡看起来是一个单独的以太网接口设备并具备相同的ip地址

- mod0 平衡轮循环策略
- mod1 主备份策略
- mod2 平衡策略
- mod3 广播策略
- mod4 IEEE 802.3ad动态链接聚合
- mod5 适配器传输负载均衡
- mod6 适配器适应性负载均衡

8.6 系统负载

Linux 命令：

Top

Uptime

W

cat /proc/loadavg

8.7 GC 问题

#1 是否 full GC（GC 是 java 里面 JVM 的东西，不了解的可以去看下）

Jps 查出要查看的 pid，用 jstat -gcutil pid,看第四列 O 如果 100%就是 fullGC

```
[root@cdh001 network-scripts]# jps
25841 ResourceManager
130789 Jps
107815 JournalNode
4663 QuorumPeerMain
107484 NameNode
107996 DFSZKFailoverController
107599 DataNode
[root@cdh001 network-scripts]# jstat -gcutil 107484
  S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     GCT
  0.00  87.64  64.35  60.00  98.11  94.73     75    0.487     3    0.295    0.783
[root@cdh001 network-scripts]# jstat -gcutil 107484 2000
  S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     GCT
  0.00  87.64  65.03  60.00  98.11  94.73     75    0.487     3    0.295    0.783
  0.00  87.64  65.38  60.00  98.11  94.73     75    0.487     3    0.295    0.783
^C[root@cdh001 network-scripts]# A
```

8.8 日志的查看

#1 首先要知道日志的位置,一种是去配置文件查看有没有配置相关位置，一种是去通过进程查看如下 用 ps -ef |grep pid 通过显示出来的信息可以找到日志的位置。

查看某个进程日志的位置：

```
[root@cdh001 network-scripts]# jps
25841 ResourceManager
107815 JournalNode
4663 QuorumPeerMain
131017 Jps
107484 NameNode
107996 DFSZKFailoverController
107599 DataNode
[root@cdh001 network-scripts]# ps -ef |grep 107484
hadoop   107484      1  0 Dec04 ?        00:01:06 /kkb/install/jdk1.8.0_141/bin/java -Dproc_namenode -Xmx1000m -Djava
.net.preferIPv4Stack=true -Dhadoop.log.dir=/kkb/install/hadoop-2.6.0-cdh5.14.2/logs -Dhadoop.log.file=hadoop.log -Dha
doop.home.dir=/kkb/install/hadoop-2.6.0-cdh5.14.2 -Dhadoop.id.str=hadoop -Dhadoop.root.logger=INFO,console -Djava.lib
rary.path=/kkb/install/hadoop-2.6.0-cdh5.14.2/lib/native -Dhadoop.policy.file=hadoop-policy.xml -Djava.net.preferIPv4
Stack=true -Djava.net.preferIPv4Stack=true -Djava.net.preferIPv4Stack=true -Dhadoop.log.dir=/kkb/install/hadoop-2.6.0
-cdh5.14.2/logs -Dhadoop.log.file=hadoop-hadoop-namenode-cdh001.log -Dhadoop.home.dir=/kkb/install/hadoop-2.6.0-cdh5.
14.2 -Dhadoop.id.str=hadoop -Dhadoop.root.logger=INFO,RFA -Djava.library.path=/kkb/install/hadoop-2.6.0-cdh5.14.2/lib
/native -Dhadoop.policy.file=hadoop-policy.xml -Djava.net.preferIPv4Stack=true -Dhadoop.security.logger=INFO,RFAS -Dh
dfs.audit.logger=INFO,NullAppender -Dhadoop.security.logger=INFO,RFAS -Dhdfs.audit.logger=INFO,NullAppender -Dhadoop.
security.logger=INFO,RFAS -Dhdfs.audit.logger=INFO,NullAppender -Dhadoop.security.logger=INFO,RFAS org.apache.hadoop.
hdfs.server.namenode.NameNode
root     131028 128548  0 00:49 pts/0    00:00:00 grep --color=auto 107484
[root@cdh001 network-scripts]#
```

2 查看日志

实时查看日志文件后 100 行：tail -f 100 file

静态查看：vim filelog，然后可以通过关键字 error/warn 等查找错误

第九章：linux 生产问题与解决方案

9.1 生产环境

很多开发人员基本没进过机房，简单描述一下，一般机房中有多个机架，一般每个机架上可以存放 5-10 几台的服务器，像大型的系统为了安全性可能选择异地灾备的方式采用双机房，不同地区各建一个机房。服务器一般带有光驱位

机柜/机架

单台服务器

9.2 生产问题方案

1.生产环境中机房要安装百台机器如何批量装机?

解决方案：PXE 批量装机

PXE，就是预启动执行环境，是一种引导启动的方式。这种协议一般由两部分构成，一部分是服务器端，一个是客户端。简单来说，我们通过这种方式可以自己创建一个“安装源”，在安装系统的时候只要能找到这个“源”便可以实现系统的安装。在实现无人值守的安装前，我们必须要搭建一些服务，来实现“安装源”的建立，例如 ftp、http、tftp、dhcp 等。当一台主机启动时，标准输入输出会将 PXE 客户端调入我们的内存中进行相关的操作，并提示相关的选项，在这里我们可以进行选择。PXE 的客户端通过网络下载(download)启动文件到本地运行。具体过程是，PXE 客户端通过网卡向局域网内发送 ip

请求，然后 DHCP 服务器会提供给给它一个 ip 地址和系统安装所需要的文件，接下使用接收到的文件进行系统安装。而安装的过程又需要其他服务器提供的资源，例如：yum 源，内核文件等，当主机拿到这些资源，便可以顺利的安装了。最终结果是：任意一台主机在选着网络启动时会获取 DHCP 服务器分发的 ip，通过通过获取到的 ip 地址与局域网内的 TFTP 服务器通信并获取启动文件，与 FTP 或者 HTTP 通信并获取 yum 源文件及内核文件等。之后开始自动安装，而这个过程不需要人在做任何操作。

PXE 安装优点，这种安装系统的方式可以不受光驱，光盘以及一些外部设备的限制，还可以做到无人值守，大大减轻了运维人员的工作负荷，像在一些主机数量庞大的机房进行批量安装，PXE 将是你不二的选择。

详细参考：https://www.cnblogs.com/x_wukong/p/8880606.html

2.有很多机器需要采用 yum 方式安装软件，但不想每台去配置 yum 如何解决？

解决方案：基于 http 方式搭建本地 yum 源，实现共享

详细参考：https://blog.csdn.net/networken/article/details/80729234

第十章：linux 运维之生产监控工具

系统的监控对我们后续大数据平台出现问题找原因很关键，但是这是运维的工作，作为开发人员我们简单了解下生产环境中常用的监控工

具。具体的安装使用见链接文档

10.1 zabbix

zabbix 是一个基于 web 界面的企业级开源监控软件，Zabbix 服务器需要 LAMP 环境或 LNMP 环境，提供分布式系统监控与网络监视功能。具备主机的性能监控，网络设备性能监控，数据库性能监控，多种告警方式，详细报表、图表的绘制等功能。监测对象可以是 Linux 或 Windows 服务器，也可以是路由器、交换机等网络设备，通过 SNMP、zabbix Agent、PING、端口监视等方法提供对远程网络服务器等监控、数据收集等功能。

https://blog.51cto.com/14154700/2419934

10.2 nagios（网络监控）

Nagios 是一款开源的电脑系统和网络监视工具，能有效监控 Windows、Linux 和 Unix 的主机状态，交换机路由器等网络设备，打印机等。在系统或服务状态异常发出邮件或短信报警第一时间通知网站运维人员，在状态恢复后发出正常的邮件或者短信通知。

Nagios 原名为：NetSaint，由 Ethan Galstad 开发并维护至今。NAGIOS 是一个缩写形式。Nagios 被开发在 Linux 下使用。但是在 Unix 下也工作的非常好。

https://www.cnblogs.com/52-qq/p/9773880.html

10.3 Prometheus(普罗米修斯)

DevOps 工程师或 SRE 工程师，可能都知道 Prometheus 普罗米修斯。Prometheus 于 2012 年由 SoundCloud 创建，目前已经已发展为最热门的分布式监控系统。Prometheus 完全开源的，被很多云厂商（架构）内置，在这些厂商（架构）中，可以简单部署 Prometheus，用来监控整个云基础架构设施。比如 DigitalOcean 或 Docker 都是普罗米修斯作为基础监控。现在比较火的 k8s 平台基本都是 Prometheus 监控

https://blog.csdn.net/csolo/article/details/82460539

10.4 ganglia （hadoop 集群监控常用工具）

ganglia 相比于 zabbix 主要在于集群的状态集中显示，可以很便捷的对比各主机的性能状态。

https://www.cnblogs.com/marility/p/9444357.html

第十一章：自动化运维工具 ansible 与 pdsh

详细使用：

https://www.cnblogs.com/keerya/p/7987886.html#_label2_0_1
https://blog.csdn.net/weixin_42193400/article/details/82148974

11.1 自动化运维工具 ansible

1.什么是 ansible？

ansible 是一个用来批量部署远程主机上服务的工具。这里“远程主机（Remote Host）”是指任何可以通过 SSH 登录的主机，所以它既可

以是远程虚拟机或物理机，也可以是本地主机。简要总结就是：ansible 是一个用来远程管理服务器的工具软件。

ansible 通过 SSH 协议实现管理节点与远程节点之间的通信。理论上来说，只要能通过 SSH 登录到远程主机来完成的操作，都可以通过 ansible 实现批量自动化操作。

涉及管理操作：复制文件、安装服务、服务启动停止管理、配置管理等。

目前，ansible 已经成为专业运维人员必备的批量化工具,在我们大规模的大数据集群的部署和运维中经常会使用 ansible.

2.为什么要用批量管理工具运维？

提高效率，比如百度和阿里云服务器很多几万台或者几十万台。使用 ansible 简单好管理。

3. ansible 架构介绍

（ansible 是模块化的 它所有的操作都依赖于模块）

1）连接插件 connectior plugins 用于连接主机 用来连接被管理端

2）核心模块 core modules 连接主机实现操作， 它依赖于具体的模块来做具体的事情

3）自定义模块 custom modules，根据自己的需求编写具体的模块

4）插件 plugins，完成模块功能的补充

5）剧本 playbooks，ansible 的配置文件,将多个任务定义在剧本中，由 ansible 自动执行

6）主机清单 inventor，定义 ansible 需要操作主机的范围

4.ansible 的安装与配置

#1 环境准备：

root 用户

配置好免密（基于 ssh）

#2 安装（使用 yum 安装）

yum install epel-release -y

yum install ansible –y

#3 ansible 配置文件/etc/ansible/ansible.cfg

这里无需更改，我们大致看下有哪些常用参数：

inventory = /etc/ansible/hosts #这个参数表示资源清单 inventory 文件的位置

library = /usr/share/ansible #指向存放 Ansible 模块的目录，支持多个目录方式，只要用冒号（：）隔开就可以 forks = 5 #并发连接数，默认为 5

sudo_user = root #设置默认执行命令的用户

remote_port = 22 #指定连接被管节点的管理端口，默认为 22 端口，建议修改，能够更加安全

host_key_checking = False #设置是否检查 SSH 主机的密钥，值为 True/False。关闭后第一次连接不会提示配置实例

timeout = 60 #设置 SSH 连接的超时时间，单位为秒

log_path = /var/log/ansible.log #指定一个存储 ansible 日志的文件(默认不记录日志)

#4 添加主机列表--关键

vim /etc/ansible/hosts

其中[cdh]为这一组主机的模块的名称，自己定义

```
# Here's another example of host ra
# leading 0s:

## db-[99:101]-node.example.com


[cdh]
192.168.52.110
192.168.52.120
192.168.52.130
```

上面 ip 根据实际的 ip 配置！！！

5.ansible 基础使用

ansible 是基于模块化的，常用的模块有 Command 模块，shell 模块 file 模块，yum 模块 等等批量操作的模块，下面我们简单介绍几种常用用法，其他模块详细使用见上面链接。

Command 模块：

查看主机是否连通

```
[root@cdh001 opt]# ansible cdh -m ping
192.168.52.120 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
192.168.52.130 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
192.168.52.110 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

查看节点的时间是否同步

ansible cdh -m command -a “date”

```
[root@cdh001 opt]# ansible cdh -a "date"
192.168.52.120 | CHANGED | rc=0 >>
Wed Dec  4 23:56:31 CST 2019

192.168.52.130 | CHANGED | rc=0 >>
Wed Dec  4 23:56:31 CST 2019

192.168.52.110 | CHANGED | rc=0 >>
Wed Dec  4 23:56:31 CST 2019

[root@cdh001 opt]#
```

Shell 模块：

例：查看节点磁盘空间，可以实现一个界面显示所有节点的信息

ansible cdh -m shell -a "df -h"

```
[root@cdh001 opt]# ansible cdh -m shell -a "df -h"
192.168.52.120 | CHANGED | rc=0 >>
Filesystem               Size  Used Avail Use% Mounted on
/dev/mapper/centos-root   17G  5.9G   12G  35% /
devtmpfs                 977M     0  977M   0% /dev
tmpfs                    993M     0  993M   0% /dev/shm
tmpfs                    993M   18M  975M   2% /run
tmpfs                    993M     0  993M   0% /sys/fs/cgroup
/dev/sda1               1014M  161M  854M  16% /boot
tmpfs                    199M   12K  199M   1% /run/user/42
tmpfs                    199M     0  199M   0% /run/user/1001
tmpfs                    199M     0  199M   0% /run/user/0

192.168.52.130 | CHANGED | rc=0 >>
Filesystem               Size  Used Avail Use% Mounted on
/dev/mapper/centos-root   17G  9.8G  7.3G  58% /
devtmpfs                 977M     0  977M   0% /dev
tmpfs                    993M     0  993M   0% /dev/shm
```

11.2 并行管理工具 pdsh 与 pssh

相关文档：https://www.cnblogs.com/goldenblade/p/9604642.html

1.并行管理工具分类

并行管理的方式有很多种：

命令行：一般是 for 循环

脚本：　一般是 expect+ssh 等自编辑脚本

工具：　pssh，pdsh，mussh，cexec 等

平台：　ansible，pupet 等

其中，pdsh 和 pssh 比较轻量级，在我们规模比较大的集群中经常在监控脚本中使用 pdsh 来进行并行操作将结果返回到一个文件中批量查看，而 pssh 可以实现文件的批量传送到 n 台机器上，而不需要额外写脚本比较麻烦，方便我们集群的运维管理。

2. pdsh 的安装与使用

安装：

方式一：使用我提供的安装包（pdsh-2.26.tar.bz2）

```
将脚本传到/opt 下，执行下面命令编译安装：
tar -xvf pdsh-2.26.tar.bz2
cd /opt/pdsh-2.26/
sudo yum install    -y openssh-clients
sudo yum install gcc -y
./configure --with-ssh --without-rsh
make&&make install
```

方式二：

在 http://code.google.com/p/pdsh/ 下载最新的源码包进行编译安装，目前最新版本为 pdsh-2.29，这里下载的源码包为 pdsh-2.29.tar.bz2。

编译安装过程如下：

```
[root@server ~]# tar jxvf pdsh-2.29.tar.bz2
[root@server ~]# cd pdsh-2.29
[root@server pdsh-2.29]#./configure --with-ssh --with-rsh --with-mrsh
--with-mqshell --with-qshell --with-dshgroups
--with-machines=/etc/pdsh/machines
[root@server pdsh-2.29]# make
[root@server pdsh-2.29]# make install
```

在执行 configure 阶段，“--with-ssh”参数表示启用 ssh 模块，其他参数都有类似的含义，而“--with-dshgroups”表示启用主机组支持，启用此参数后，就可以将一组主机列表写入一个文件并放到~/.dsh/group 或/etc/dsh/group 目录下，然后通过 pdsh 的“-g”参数进行调用。最后的参数“--with-machines”是“--with-dshgroups”参数的扩展，通过将所有要管理的主机列表都写入指定的/etc/pdsh/machines 文件中，接着通过 pdsh 的“-a”参数调用，最终完成所有主机的便捷管理。

```
[opsuser@server ~]# pdsh -Vpdsh-2.29rcmd modules: ssh,rsh,exec (default: rsh)misc modules: machines,dshgroup
```

pdsh 语法介绍：

安装 pdsh 完成后，通过执行 pdsh –h 和

pdcp –h 即可得到两个命令的完整用法，由于两个命令的参数大同小异，因此这里以 pdsh 命令为主介绍一些常用的参数及含义。

-w host,host 指定远程主机，可以指定多个，每个主机用逗号隔开，host 可以是主机名也可以是 IP 地址。此参数非常灵活，常用的形式有：

```
[root@server ~]# pdsh -w ssh:node01,ssh:node02,ssh:node03 "date"
```

此命令用来查看 user001、user002、user003 主机上的时间，其中 ssh 表示在远程主机上执行命令的形式，默认是 rsh。

```
[root@server ~]# pdsh -w ssh:node0[1-3] "date"
```

此命令用于在 user001 到 user0010 上执行 date 命令。

```
[root@server ~]#pdsh -w ssh:node0[1-3],/1$/ "uptime"
```

此命令在选择远程主机时使用了正则表达式，表示在 node01 到 node03 中选择以 1 结尾的主机名，即在 node01 上执行 uptime 命令

-R 指定使用 rcmd 的模块名，默认是 rsh。如果要选择 ssh，可以通过如下方式指定：

```
[root@server ~]#pdsh -R ssh -w node0[1-10] "date"
```

-l 指定在远程主机上使用的用户名称。例如：

```
[root@server ~]#pdsh -R ssh -l opsuser -w node0[1-9] "date"
```

-x 此参数用来排除某些或某个主机，例如：

```
[root@server ~]#pdsh -R ssh -l opsuser -w node0[1-9] -x node05,node07 "date"
```

-t 指定连接远程主机的超时时间，以秒为单位，默认是 10 秒，可以通过此参数修改默认值，例如：

```
pdsh -R ssh -w node0[1-9] -t 15 "date"
```

-u 设置远程命令执行的超时时间，以秒为单位，以 ssh 方式连接时，默认时间为无限

-f 设置同时连接到远程主机的个数

-N 此参数用来关闭远程主机所返回结果中的主机名显示

-a 通过此参数可以指定所有的远程主机，设置此参数后，pdsh 默认会查看/etc/machines 文件中的主机列表，要改变此路径，在编译 pdsh 时通过“--with-machines”参数指定即可

-g 此参数用来指定一组远程主机，在编译 pdsh 时可以通过“--with-dshgroups”参数来激活此选项，默认可以将一组主机列表写入一个文件中并放到本地主机的~/.dsh/group 或/etc/dsh/group 目录下，这样就可以通过“-g”参数调用了。例如：pdsh -R ssh -g userhosts "date"，其中“userhosts”是一个主机列表文件，可以将此文件放在~/.dsh/group 或/etc/dsh/group 目录下

-X 此参数用来排除指定组内的所有主机，经常与“-a”参数一起使用。例如：

```
[root@server ~]#pdsh -R ssh -a -X userhosts "date"
```

-q 此参数可以列出 pdsh 执行时的一些配置信息

-V 此参数可以查看软件的版本信息以及可用的模块信息

生产最常用：pdsh -w

pdsh -w 后面加上批量主机名可以实现所有机器返回结果到执行命令的这台机器，常常用在写监控大数据集群的脚本中

例：查看所有节点根目录占用超过 30%的：参数 W 后面跟 host 名

```
pdsh -w node0[1-3] df -h | egrep /$ | sed 's/%//g' | awk '$5>30' | sort    -k 6 -nr | awk '{print $1,$2,$3,$5,$6"%"}'
```

3. pssh 的安装与使用

1. pssh 安装(依次执行)

```
wget http://peak.telecommunity.com/dist/ez_setup.py

python ez_setup.py

wget http://parallel-ssh.googlecode.com/files/pssh-2.2.2.tar.gz

tar zxvf pssh-2.2.2.tar.gz

cd pssh-2.2.2

python setup.py install
```

2. pssh 使用

假设 ssh 已做好 SSH 信任,ssh 信任请参看:关于 ssh 命令研究以及 SSH 信任详解

pssh 工具包主要有 5 个程序:

#1 pssh 多主机并行运行命令

```
[root@server pssh-2.2.2]# pssh -P -h test.txt uptime
192.168.9.102: 14:04:58 up 26 days, 17:05, 0 users, load average: 0.07, 0.02, 0.00
192.168.9.102: [1] 14:04:58 [SUCCESS] 192.168.9.102 9922
192.168.8.171: 14:04:59 up 35 days, 2:01, 6 users, load average: 0.00, 0.00, 0.00
192.168.8.171: [2] 14:04:59 [SUCCESS] 192.168.8.171 22
192.168.9.104: 14:04:59 up 7 days, 20:59, 0 users, load average: 0.10, 0.04, 0.01
192.168.9.104: [3] 14:04:59 [SUCCESS] 192.168.9.104 9922
[root@server pssh-2.2.2]# cat test.txt
192.168.9.102:9922
192.168.9.104:9922
192.168.8.171:22
//注意我的端口号不仅是默认的22, 假如想将输出重定向到一个文件 加-o file 选项
```

#2 pscp 把文件并行地复制到多个主机上

[root@server pssh-2.2.2]# pscp -h test.txt /etc/sysconfig/network /tmp/network

//标示将本地的/etc/sysconfig/network 传到目标服务器的/tmp/network

#3 prsync 使用 rsync 协议从本地计算机同步到远程主机

[root@server ~]# pssh -h test.txt -P mkdir /tmp/etc

[root@server ~]# prsync -h test.txt -l dongwm -a -r /etc/sysconfig /tmp/etc

//标示将本地的/etc/sysconfig 目录递归同步到目标服务器的/tmp/etc 目录下,并保持原来的时间戳

#4 pslurp 将文件从远程主机复制到本地,和 pscp 方向相反:

```
[root@server ~]# pslurp -h test.txt -L /tmp/test -l root /tmp/network test
//标示将目标服务器的/tmp/network文件复制到本地的/tmp/test目录下,并更名为test
[1] 14:53:54 [SUCCESS] 192.168.9.102 9922
[2] 14:53:54 [SUCCESS] 192.168.9.104 9922
[root@server ~]# ll /tmp/test/192.168.9.10
192.168.9.102/ 192.168.9.104/
[root@server ~]# ll /tmp/test/192.168.9.102/
总计 4.0K
-rw-r--r-- 1 root root 60 2011-04-22 14:53 test
[root@server ~]# ll /tmp/test/192.168.9.104/
总计 4.0K
-rw-r--r-- 1 root root 60 2011-04-22 14:53 test
```

#5 pnuke 并行在远程主机杀进程:

[root@server ~]# pnuke -h test.txt syslog

//杀死目标服务器的 syslog 进程,只要 ps 进程中出现相关词语 都能杀死

[1] 15:05:14 [SUCCESS] 192.168.9.102 9922

[2] [2] 15:05:14 [SUCCESS] 192.168.9.104 9922

第十二章：linux 知识点扩展

12.1 Linux 系统运行级别

Linux 系统有 7 个运行级别(runlevel)

运行级别 0：系统停机状态，系统默认运行级别不能设为 0，否则不能正常启动

运行级别 1：单用户工作状态，root 权限，用于系统维护，禁止远程登陆

运行级别 2：多用户状态(没有 NFS)

运行级别 3：完全的多用户状态(有 NFS)，登陆后进入控制台命令行模式

运行级别 4：系统未使用，保留

运行级别 5：X11 控制台，登陆后进入图形 GUI 模式

运行级别 6：系统正常关闭并重启，默认运行级别不能设为 6，否则不能正常启动

运行级别的原理：

1.在目录/etc/rc.d/init.d 下有许多服务器脚本程序，一般称为服务(service)

2.在/etc/rc.d 下有 7 个名为 rcN.d 的目录，对应系统的 7 个运行级别

3.rcN.d 目录下都是一些符号链接文件，这些链接文件都指向 init.d 目录下的 service 脚本文件，命名规则为 K+nn+服务名或 S+nn+服务名，其中 nn 为两位数字。

4。系统会根据指定的运行级别进入对应的 rcN.d 目录，并按照文件名顺序检索目录下的链接文件

对于以 K 开头的文件，系统将终止对应的服务

对于以 S 开头的文件，系统将启动对应的服务

5。查看运行级别用：runlevel

6。进入其它运行级别用：init N

7。另外 init0 为关机，init 6 为重启系统

由于现在的 Linux 系统安装完后就运行在第 5 个级别，即系统启动后直接进入图形界面，而不用在字符模式下登录后用 startx 或者 xinit 来起动图形界面。建议在系统安装完成后把系统的默认运行等级设置在第 3 级，在字符终端登录后，再手工输入 startx 命令起动图形界面。可以用如下的方法修改：

用文本编辑器修改 /etc/inittab 文件，把

代码:

id:5:initdefault:这一行，修改成

代码:

id:3:initdefault:保存后就 reboot 重起，系统就默认起动到字符界面。不同运行级别之间的 差别的在于系统默认起动的服务的不同，如运行级别 3 默认不启动 X 图形界面服务，而运行级别 5 却默认起动。

本质上是没有区别的，更无所谓不同级别间功能强弱的问题。用户完全可自给定义不同级别的默认服务。在任何运行级别，用户都可用 init 命令来切换到其他运行级别。

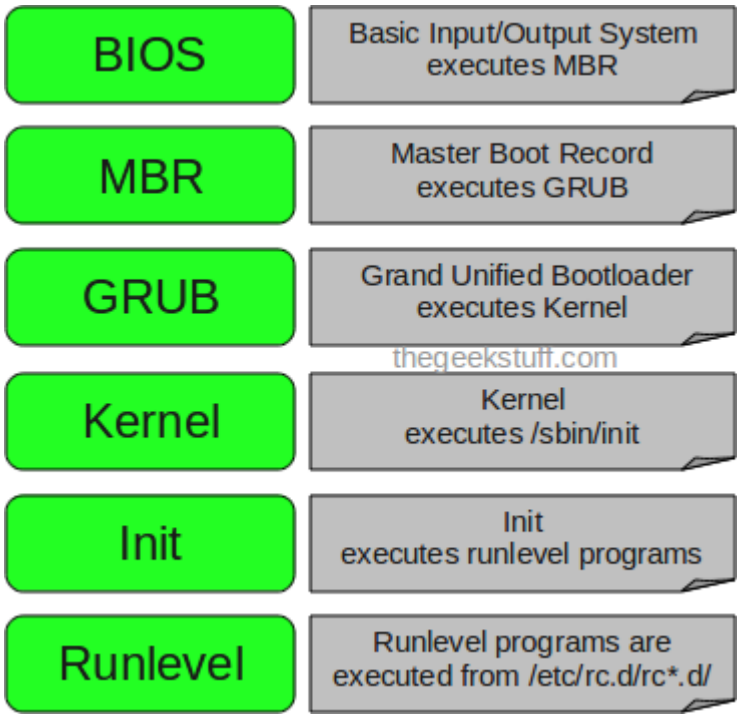
12.2linux 开机过程发生了什么

相信去面试过大厂 linux 的对这个问题并不陌生，知道启动过程才能够解决启动中遇到的问题。

当你按下主机电源几分钟后看到 Linux 的登陆界面出现，这中间你知道发生了什么事吗？

通电->BIOS->MBR->GRUB->Kernel->/sbin/init->Runlevel

下图显示了典型 Linux 系统启动的 6 个主要阶段：



1. BIOS

BIOS: Basic Input/Output System

执行系统完整性检查

从软盘，光盘，硬盘中查找 boot loader（取决与你的启动顺序）

BIOS 的主要功能就是搜索并加载执行 MBR boot loader 程序

2. MBR

MBR: Master Boot Record.

MBR 位于主盘的第一快扇区，如:/dev/hda 或者/dev/sda.MBR 共 512bytes. 1-446bytes 存放主要的 boot loader 信息，447~510bytes 存放分区表信息，最后 2 个 bytes 存放 mbr 验证信息。

MBR 包含 GRUB 信息。（或者 LILO 信息）

MBR 的主要功能就是加载并执行 GRUB boot loader.

3. GRUB

GRUB：Grand Unified Bootloader.

如果你有多个 kernel p_w_picpaths 安装在你的系统中，你可以通过 GRUB 选择那个被执行。

GRUB 启动的时候会有个选择界面，几秒钟没有任何输入的话，将会加载你 GRUB 配置文件中指定的默认 kernel p_w_picpath 。

GRUB 的配置文件：/boot/grub/grub.conf (/etc/grub.conf is a link to this).

下面是 CentOS 的 GRUB 的配置文件 grub.conf .你可以看到里面包含 kernel and initrd p_w_picpath。

```
#boot=/dev/sda
default=0
timeout=5
splashp_w_picpath=(hd0,0)/boot/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.18-194.el5PAE)
        root (hd0,0)
        kernel /boot/vmlinuz-2.6.18-194.el5PAE ro root=LABEL=/
        initrd /boot/initrd-2.6.18-194.el5PAE.img
```

GRUB 的主要功能就是加载并执行 Kernel and initrd p_w_picpaths.

4. Kernel

挂载 grub.conf 中指定“root=” 的根目录

执行/sbin/init 程序

因为/sbin/init 是 LINUX kernel 执行的第一程序，理所当然/sbin/init 的 PID 为 1. ‘ps -ef | grep init’ 命令可以验证.

initrd: Initial RAM Disk.

在 kernel 完全起来，root 文件系统被挂载之前，initrd 被 kernel 当做临时 root 文件系统。当然 initrd 还包含了一些编译好的驱动，这些驱动用来在启动的时候访问硬件。

5. Init

查看/etc/inittab 配置文件来决定 Linux 的运行（run level）.

LINUX 的运行级别：

0 – halt （关机）

1 – Single user mode （单用户模式）

2 – Multiuser, without NFS （多用户模式，无网络）

3 – Full multiuser mode （多用户模式，有网络，无图形界面）

4 – unused （没用，用户自定义）

5 – X11 （多用户模式，有网络，有图形界面）

6 – reboot （重启）

Init 从/etc/inittab 配置文件中得到默认启动级别，然后加载并执行相应级别的程序.

执行 ‘grep initdefault /etc/inittab’ 命令可得到你系统的默认启动级别。

6. Runlevel programs

当 LINUX 系统启动完成后，你会发现许多的服务进程也启动了，例如：“starting sendmail …. OK”. 这些服务程序都放在相应 LINUX 系统启动级别的文件夹下面。

根据你 LINUX 默认启动级别，系统将会执行以下其中一个文件架下面的服务程序：

Run level 0 – /etc/rc.d/rc0.d/

Run level 1 – /etc/rc.d/rc1.d/

Run level 2 – /etc/rc.d/rc2.d/

Run level 3 – /etc/rc.d/rc3.d/

Run level 4 – /etc/rc.d/rc4.d/

Run level 5 – /etc/rc.d/rc5.d/

Run level 6 – /etc/rc.d/rc6.d/

请注意，在/etc 下面的那些文件有些是连接文件。如 etc/rc0.d 连接到 /etc/rc.d/rc0.d。

在/etc/rc.d/rc*.d/ 文件架下面, 你会发现服务程序名是以 S 或 K 开头。.

以 S 开头的表示用于服务启动。S 表示 startup。

以 K 开头的表示用于服务关闭。 K 表示 kill。

在 S 或 K 的后面的数字，表示启动/关闭程序的顺序。例如：S12syslog is to start the syslog deamon, which has the sequence number of 12. S80sendmail is to start the sendmail daemon, which has the sequence number of 80. So, syslog program will be started before sendmail.
S12syslog 启动 syslog 守护进程，12 是启动顺序；S80sendmail 启动 sendmail 守护进程,80 是启动顺序，12 在 80 的前面，所以 syslog 服务程序将在 sendmail 之前启动。

这就是 LINUX 系统在开启电源到完全启动所发生的一切。

12.3 linux 的零拷贝

详细参考链接：

https://www.cnblogs.com/skying555/p/11122072.html

总结：（kafka 面试中常问）

“零拷贝” 描述的是 CPU 不执行拷贝数据从一块内存区域到另一块区域的任务的计算机操作。它通常用于在网络上传输文件时节省 CPU 周期和内存带宽。简单来说，零拷贝就是一种避免 CPU 将数据从一块存储拷贝到另外一块存储的技术。

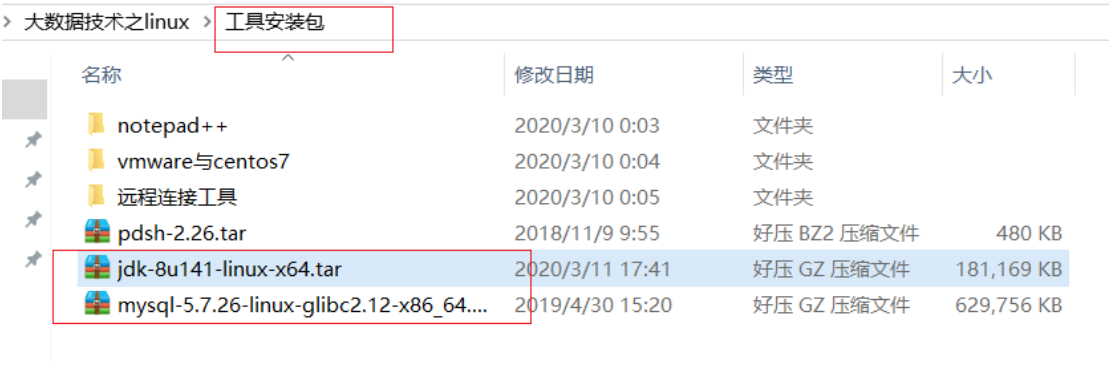
“零拷贝”正是通过消除这些多余的拷贝来提升性能的。在数据传输的过程中，**避免数据在内核空间缓冲区和用户空间缓冲区之间进行拷贝，以及数据在内核空间缓冲区内的 CPU 拷贝**。

第十三章：linux 常用软件离线安装

13.1 离线安装包版本

1. jdk1.8 版本（linux）

2.mysql5.7.26 版本 tar 包



13.2 离线安装 jdk

步骤一：在/opt 下创建 soft 和 install 文件夹分别存放未解压和已解压文件

```
# cd /opt

# mkdir soft

# mkdir install

# chmod 777 soft ;chmod 777 install
```

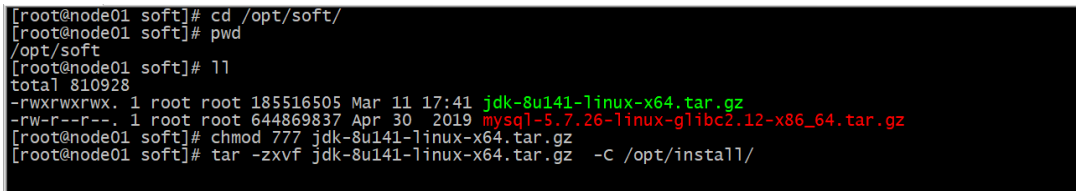
```
[root@node01 opt]# mkdir soft
[root@node01 opt]# mkdir install
[root@node01 opt]# ll
total 0
drwxr-xr-x. 2 root root 6 Mar 17 11:35 install
drwxr-xr-x. 2 root root 6 Mar 17 11:35 soft
[root@node01 opt]#
```

步骤二：将 jdk 的 tar 包通过 rz 命令传到/opt/soft 下，解压缩到 /opt/install 下。

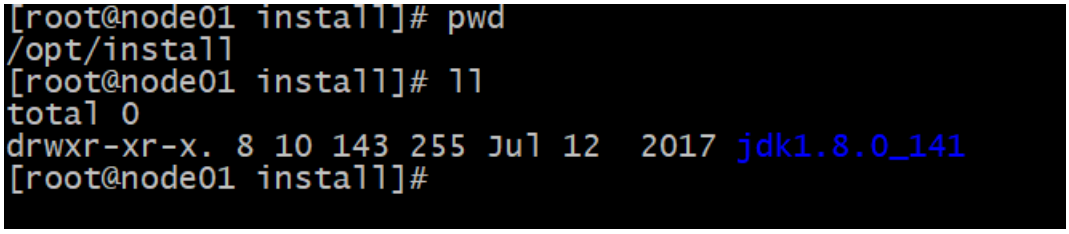
```
# cd /opt/soft

# chmod 777 jdk-8u141-linux-x64.tar.gz

# tar -zxvf jdk-8u141-linux-x64.tar.gz    -C /opt/install/
```



到/opt/install 下查看已经存在解压后的 jdk 版本



步骤三：配置/etc/profile,并 source 环境变量生效(最后一行开始加)

```
#vim /etc/profile
```

```
JAVA_HOME=/opt/install/jdk1.8.0_141

PATH=$PATH:$HOME/bin:$JAVA_HOME/bin:

export JAVA_HOME
```

```
# source /etc/profile
```

```
else
    umask 022
fi

for i in /etc/profile.d/*.sh ; do
    if [ -r "$i" ]; then
        if [ "${-#*i}" != "$-" ]; then
            . "$i"
        else
            . "$i" >/dev/null
        fi
    fi
done

unset i
unset -f pathmunge

JAVA_HOME=/opt/install/jdk1.8.0_141
PATH=$PATH:$HOME/bin:$JAVA_HOME/bin:
export JAVA_HOME
"/etc/profile" 86L, 1892C written
[root@node01 install]# source /etc/profile
```

步骤四：查看验证 jdk 版本

```
# java -version
```

```
[root@node01 install]# java -version
java version "1.8.0_141"
Java(TM) SE Runtime Environment (build 1.8.0_141-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.141-b15, mixed mode)
[root@node01 install]#
```

13.3 离线安装 mysql

参考：https://www.cnblogs.com/yy3b2007com/p/10497787.html

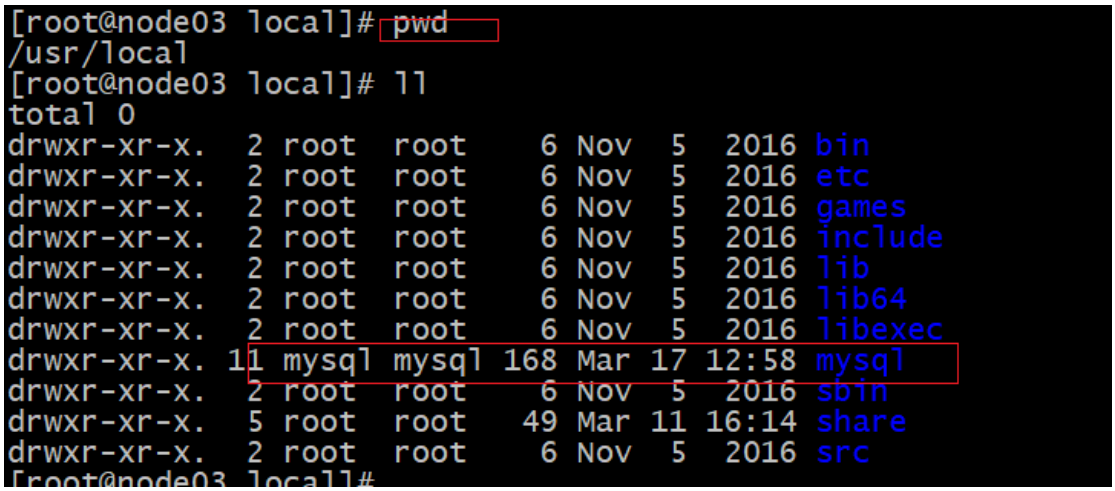
步骤一：将 mysql 安装包上传到/usr/local/下并解压

```
# cd /usr/local

# tar -zxvf mysql-5.7.26-linux-glibc2.12-x86_64.tar.gz -C /usr/local
```

将解压后的 mysql 的 tar 包更改名称为 mysql

```
# mv mysql-5.7.26-linux-glibc2.12-x86_64/ mysql
```



步骤二：**查看系统自带的 Mariadb 并卸载**

```
# rpm -qa|grep mariadb

# rpm -e --nodeps mariadb-libs-5.5.56-2.el7.x86_64(根据实际情况复制
rpm 包)
```

删除/etc 目录下的 my.cnf（有就删除）

```
# rm /etc/my.cnf
```

检查 mysql 是否存在

```
# rpm -qa |grep mysql
```

步骤三：检查 mysql 的组和用户是否存在，如无则创建

检查用户和组：

```
# cat /etc/group | grep mysql

# cat /etc/passwd | grep mysql
```

创建用户和组：

```
# groupadd mysql

# useradd -g mysql mysql

# passwd mysql          （输入密码密码设置成 123456）
```

步骤四：将解压后的 mysql 的安装包更改用户和组为 mysql

```
# cd /usr/local

# chown -R mysql:mysql mysql/
```

```
[root@node03 install]# chown -R mysql:mysql mysql/
[root@node03 install]# ll
total 0
drwxr-xr-x. 9 mysql mysql 129 Mar 17 12:35 mysql
[root@node03 install]#
```

在 mysql 文件夹下创建 data 目录，并修改用户和组的权限都为 mysql

```
# cd /usr/local/mysql

# mkdir data

# chown -R mysql:mysql data
```

步骤五：在/etc 下创建 my.cnf 文件

```
# cd /usr/local/mysql

# vim my.conf
```

```
[mysql]

socket=/var/lib/mysql/mysql.sock# set mysql client default

chararterdefault-character-set=utf8
```

```
[mysqld]
socket=/var/lib/mysql/mysql.sock# set mysql server port
port = 3323 #默认是 3306，这里发现 3306 已经被占用，因此防止这种情况发生，可以避免使用 3306mysql 默认端口
# set mysql install base dir
basedir=/usr/local/mysql# set the data store dir
datadir=/usr/local/mysql/data# set the number of allow max connnection
max_connections=200# set server charactre default encoding
character-set-server=utf8# the storage
enginedefault-storage-engine=INNODB
lower_case_table_names=1
max_allowed_packet=16M
explicit_defaults_for_timestamp=true

[mysql.server]
user=mysql
basedir=/usr/local/mysql
```

```
# chown -R mysql:mysqsl my.cnf

#chmod 777 my.conf
```

步骤六进入 mysql 文件夹，安装 mysql

```
# cd /usr/local/mysql

# bin/mysql_install_db --user=mysql --basedir=/usr/local/mysql/
--datadir=/usr/local/mysql/data/

# cp ./support-files/mysql.server /etc/init.d/mysqld

# chmod a+x /etc/init.d/mysqld
```

步骤七：配置环境变量

```
# vim /etc/profile

export PATH=$PATH:/usr/local/mysql/bin

#source /etc/profile
```

步骤八：启动 mysql

```
# service mysqld start
```

查看状态：

```
# service mysqld status
```

步骤九：登录 mysql 并设置基础环境

获取 mysql 初始密码：

```
#cat /root/.mysql_secret
```

登录：

```
# mysql -u root -p        (输入刚才查出的密码)
```

更改密码：

```
mysql>    set PASSWORD = PASSWORD('123456');
```

刷新保存设置：

```
mysql>    flush privileges;
```

设置 root 用户的远程登录：

```
mysql>grant all on *.* to root@'%' identified by '123456';

mysql> flush privileges;

mysql> select user,host from user;

mysql> delete from user where user='root' and host='localhost';

mysql> select user,host from user;

mysql> flush privileges;
```

```
#添加root用户远程访问数据库
mysql>grant all on *.* to root@'%' identified by '!Qaz123456';
mysql> flush privileges;
mysql> select user,host from user;
+---------------+-----------+
| user          | host      |
+---------------+-----------+
| root          | %         |
| scm           | %         |
| mysql.session | localhost |
| mysql.sys     | localhost |
| root          | localhost |
+---------------+-----------+
5 rows in set (0.00 sec)

mysql> delete from user where user='root' and host='localhost';
Query OK, 1 row affected (0.02 sec)

mysql> select user,host from user;
+---------------+-----------+
| user          | host      |
+---------------+-----------+
| root          | %         |
| scm           | %         |
| mysql.session | localhost |
| mysql.sys     | localhost |
+---------------+-----------+
4 rows in set (0.00 sec)

mysql> flush privileges;
```

步骤 10：重启 mysql

```
# service mysqld restart
```