# 第一章 Base

开启端口

添加端口：firewall-cmd --zone=public --add-port=80/tcp --permanent

firewall-cmd --reload

查看端口状态：firewall-cmd --query-port=9200/tcp

## 服务

服务：开机不用登录就可以执行的命令

开机启动

1.在路径/usr/lib/systemd/system添加服务文件updateDNS.service

2.开机启动systemctl enable updateDNS.service

3.修改后重启 systemctl daemon-reload

**原理**

systemctl enable/ disable updateDNS

原理：systemctl enable updateDNS.service时，updateDNS.service的链接文件/etc/systemd/system/ multi-user.target.wants/路径下。

**启动模式**

修改文件xx.service

[Install]

WantedBy=服务组 Target

multi-user.target 这个组开机都启动。缺省模式。

graphical.target：图形用户状态。

服务配置

路径1：/usr/lib/systemd/system 开机不用登录

路径2：/usr/lib/systemd/user/

路径3：/etc/systemd/system

路径下有四种类型文件

xxx.mount 挂载点。

xxx.service 定义服务

xxx.target 基础组件，供.service调用

xxx.wants 服务组

xx.service

三部分： Unit、Service、Install

[Unit]

Description=update dns server at home machine

[Service]

ExecStart=/usr/local/myservice/updateDNS/start.sh

[Install]

运行模式。

### Service

执行/etc/init.d/目录下脚本

增加服务方法

1.脚本，普通shell，但要实现service的start等， 参考/etc/init.d/updateDNS

2.updateDNS脚本放/etc/init.d/目录

开机启动

chkconfig [--level 3] httpd on/off

或者chkconfig --add/del updateDNS。(原理：执行后/etc/rc2.d,/etc/rc3.d等添加K10/S10链接)

chkconfig --list

service updateDNS start 相对于 updateDNS start，不同在于脚本在init.d目录。

很多服务以d结尾，d代表 deamon 守护进程

init.d

/etc/init.d 是 /etc/rc.d/init.d 的软链接(soft link)。放init.d保证不同linux版本兼容性。

chkconfig

配置某服务开关机自动启动。默认启动2345四个级别。

执行级别

等级0表示：表示关机

等级1表示：单用户模式

等级2表示：无网络连接的多用户命令行模式

等级3表示：有网络连接的多用户命令行模式

等级4表示：不可用

等级5表示：带图形界面的多用户模式

等级6表示：重新启动

修改 chkconfig --level 35 updateDNS on

## Centos

### 环境变量

方法一 暂时生效,只会对当前回话生

export PATH=$PATH:/usr/local/apache-flume-1.6.0-bin

方法二对当前登陆用户生效，永久生效

vi ~/.bash\_profile

source ~/.bash\_profile

方法三 对所有系统用户生效，永久生效

vi /etc/profile

PATH=$PATH:/usr/local/apache-flume-1.6.0-bin

（export PATH）

最后执行命令 source /etc/profile

### 防火墙端口

添加端口：firewall-cmd --zone=public --add-port=80/tcp --permanent

firewall-cmd --reload

查看端口状态：firewall-cmd --query-port=9200/tcp

## Log

系统日志一般都存在/var/log下

常用的系统日志如下:

核心启动日志:/var/log/dmesg

系统报错日志:/var/log/messages

邮件系统日志:/var/log/maillog

FTP系统日志:/var/log/xferlog

/var/log/messages

## Shell

解释器，看成一个线程

登录shell：是需要用户名、密码登录后才能进入的shell

非登录shell： 不需要输入用户名和密码即可打开的Shell，例如：直接命令“bash”就是打开一个新的非登录shell，在Gnome或KDE中打开一个“终端”（terminal）窗口程序也是一个非登录shell。

/bin/nologin l，

交互式：在终端上执行，shell等待你的输入

非交互式：以shell script方式执行。

交互Shell:

Shell是用户与Linux系统之间的接口。常用的有

Bourne Shell（/usr/bin/sh或/bin/sh）

Bourne Again Shell（/bin/bash）

C Shell（/usr/bin/csh）

K Shell（/usr/bin/ksh）

Shell for Root（/sbin/sh）

系统管理员可以根据系统情况和用户习惯为用户指定某个Shell。

用户的登录Shell也可以指定为某个特定的程序（此程序不是一个命令解释器）。

利用这一特点，我们可以限制用户只能运行指定的应用程序，在该应用程序运行结束后，用户就自动退出了系统。

登陆shell：户登陆时，输入用户名和密码后启动的shell；

### Shell Script

扩展名为 sh（sh代表shell）

1. #!/bin/bash
2. echo "Hello World !"

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

#### 运行

1./test.sh 脚本加#!/bin/bash

2.sh test.sh

#### 变量

$# 是传给脚本的参数个数

$0 是脚本本身的名字

$1 是传递给该shell脚本的第一个参数

$2 是传递给该shell脚本的第二个参数

$@ 是传给脚本的所有参数的列表

$\* 是以一个单字符串显示所有向脚本传递的参数，与位置变量不同，参数可超过9个

$$ 是脚本运行的当前进程ID号

$? 是显示最后命令的退出状态，0 = success, 其他 = error

${var}等价于$var（简写）

if/else

[Reference](http://www.cnblogs.com/kangyoung/p/3556173.html)

#### 基本运算符

[Reference](https://www.runoob.com/linux/linux-shell-basic-operators.html)

算术运算符

* -n 检测字符串长度是否为0，不为0返回 true。
* $ 检测字符串是否为空，不为空返回 true。
* -z 检测字符串长度是否为0，为0返回 true。
* = 检测两个字符串是否相等，相等返回 true。

#### 比较

if ([ condition ] || [condition])

then

cmd

else

cmd

fi

if method()

method return 0=true ,1=false

#### 文件表达式

-e filename 如果 filename存在，则为真  
-d filename 如果 filename为目录，则为真   
-f filename 如果 filename为常规文件，则为真  
-L filename 如果 filename为符号链接，则为真  
-r filename 如果 filename可读，则为真   
-w filename 如果 filename可写，则为真   
-x filename 如果 filename可执行，则为真  
-s filename 如果文件长度不为0，则为真  
-h filename 如果文件是软链接，则为真

整数变量

eq:=

ne:!=

gt:>

ge:>=

lt: <

le <=

函数

LaunchJDeveloper(){

DisplayBanner

}

#### expr

手工命令行计数器



模式匹配

# 第二章 文件管理

## 系统目录

/bin 二进制可执行命令。一般用户可用。ls,cp

/sbin 系统管理命令。shutdown,reboot。

/etc 系统管理和配置文件

|--/etc/rc.d 启动的配置文件和脚本

|--/etc/init.d

/home 用户主目录的基点，比如我的主目录就是/home/nadong，可用~表示

/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。其中包含：

|--/usr/local 本地安装的程序，配置和其他。系统升级无需重新安装。

|--/usr/local/bin 本地增加的命令

|--/usr/local/lib 本地增加的库 比如java

|--/usr/local/etc配置文件

|--/usr/local/share其它的资源文件放在。

|--/usr/bin 预装，随系统升级改变。 和/usr/local/bin优先级看path配置。

|--/usr/sbin 非必须的系统管理命令。

|--/usr/doc linux文档

|--/usr/include linux下开发和编译应用程序所需要的头文件

|--/usr/lib 常用的动态链接库和软件包的配置文件

|--/usr/man 帮助文档

|--/usr/src 源代码，linux内核的源代码就放在/usr/src/linux里

|--/usr/x11r6 存放x window的目录

/root 系统管理员的主目录（特权阶级）

/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

/dev 设备特殊文件

/tmp 公用的临时文件存储点

/var 某些大文件的溢出区，比方说各种服务的日志文件

/mnt 系统提供这个目录是让用户临时挂载其他的文件系统。

/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。

/lost+found 平时空的，系统非正常关机而留下“无家可归”的文件。

Warning

usr 指 Unix System Resource，而不是User

path配置，前面覆盖后面的。

## 文件属性

文件颜色

绿色文件---------- 可执行文件，可执行的程序

红色文件-----------压缩文件或者包文件

蓝色文件----------目录

白色文件----------一般性文件，如文本文件，配置文件，[源码](https://www.2cto.com/ym/)文件等

浅蓝色文件----------链接文件，主要是使用ln命令建立的文件



链接文件

分配不同权限

缩写路径

软链接 VS 硬链接

软链接和Windows的快捷方式类似

文件权限

chmod

chmod u+w test

chmod ugo+r test;

chown

修改文件拥有者。一般只有系统管理员(root)拥有此操作权限。

## 文件搜索

which       查看可执行文件的位置   
whereis    查看文件的位置   
locate       配合数据库查看文件位置   
find          查询文件名称

find

find [dir] -name filename

1.在当前及子文件夹查找

find –name common.jar

2.在etc及子文件夹查找

find /etc -name httpd.conf

3.模糊查找

find –name ‘httpd\*'

Grep

-n ：输出行号

-r ：在某一目录下递归[所有子目录]查找某一字串:

grep -r ‘str\*' . (注意此处有点)

grep -r a /scratch

Whereis

whereis useradd

## 文件查看及编辑

more

Enter:一行

Space:翻页

less

less 的用法比起 more 更加的有弹性。在 more 的时候，我们并没有办法向前面翻， 只能往后面看

Space:翻页

G:倒序查看

vim

**全部删除：**按esc后，然后dG

lsof

list open files

查看端口连接情况

lsof -i:80



# 第三章 系统资源分析

## CUP线程

### top

–p pid 监控某个进程

-H show thread, top –Hp pid

4 users：当前登录user数

Tasks:进程数。-

Cpus(s)

us/sy — 用户/内核 占用CPU的百分比。

ni — 改变过优先级的进程占用CPU的百分比

id — 空闲CPU百分比

wa — IO等待占用CPU的百分比

hi — 硬中断（Hardware IRQ）占用CPU的百分比

si — 软中断（Software Interrupts）占用CPU的百分比

Title

PR 优先级

VIRT 进程使用的虚拟内存总量，单位kb。VIRT=SWAP+RES

RES 进程使用的、未被换出的物理内存大小，单位kb。RES=CODE+DATA

SHR 共享内存大小，单位kb

%CPU  CPU时间占用百分比

%MEM  进程使用的物理内存百分比

TIME+  进程使用的CPU时间总计，单位1/100秒  
CMD 进程名称（命令名/命令行）

**快捷键**

P:CPU排序。默认按cpu占用量排序。

M:内存排序

H: show thread

1:show cups 列出单个cpu的负荷情况

### ps

查看线程。Process Status

a 显示现行终端机下的所有程序，包括其他用户的程序

f 用ASCII字符显示树状结构，表达程序间的相互关系

e 列出程序时，显示程序所使用的环境变量

-A 显示所有进程

-u 以用户为主的格式来显示程序状况

-x 显示所有程序，不以终端机来区分

常用

ef



USER 用户名

UID User ID

PID Process ID（进程），当然jps命令也可以查到进行ID

PPID parent pid

TIME 进程使用的总cpu时间

COMMAND 正在执行的命令行命令

aux



SID 会话ID（Session id）

%CPU **进程的cpu占用率，单个CPU，双核的最大值为200%**

%MEM 进程的内存占用率

VSZ 进程所使用的虚存的大小（Virtual Size）

RSS 进程使用的驻留集大小或者是实际内存的大小，Kbytes字节。

TTY 与进程关联的终端（tty）

STAT 进程的状态：进程状态使用字符表示的（STAT的状态码）

START 进程启动时间和日期

## 内存

### free

[Reference](http://www.cnblogs.com/coldplayerest/archive/2010/02/20/1669949.html)



-/+ buffers/cache: 应用程序认为系统被用掉多少内存

设置缓存大小：

## 硬盘

### df

Disk File

-i 查看inodes

df –hl

df -h /home/nadong

### du

Disk usage of each File

du –sh

分区

## 网络

### netstat

系统所有的网络套接字连接情况，包括 tcp, udp 以及 unix 套接字。

Active Internet connections，称为有源TCP连接

Active UNIX domain sockets，称为有源Unix域套接口(和网络套接字一样，但是只能用于本机通信，性能可以提高一倍)

Param

-a (all)显示所有选项，默认不显示LISTEN相关

-t (tcp)仅显示tcp相关选项

-n 拒绝显示别名(域名)，能显示数字的全部转化成数字(ip)。

-u (udp)仅显示udp相关选项  
-p 显示建立相关链接的程序名

-l 仅列出有在 Listen (监听) 的服務状态

-x**只列出所有监听 UNIX 端口**

提示：LISTEN和LISTENING的状态只有用-a或者-l才能看到

-p 显示建立相关链接的程序名  
-r 显示路由信息，路由表  
-e 显示扩展信息，例如uid等  
-s 按各个协议进行统计  
-c 每隔一个固定时间，执行该netstat命令

**常用**

1)netstat -at|grep slc01boa

2)统计80端口连接数

netstat -nat|grep -i "8080"|wc –l

3)Linux下查看tomcat连接数

netstat -na | grep ESTABLISHED | grep 8080 | wc -l

1. 统计httpd协议连接数  
   ps -ef|grep httpd|wc –l
2. 统计已连接上的，状态为“established

netstat -na|grep ESTABLISHED|wc -l

3)找出程序运行的端口

netstat -ap | grep java

**4)**[根据端口查找进程](http://www.cnblogs.com/paul8339/p/6638370.html)

netstat -apn | grep 8989

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

windows

netstat -ano -p tcp -n 1|find "8080"

-n 1间隔1s自动刷新



PID是进程

统计

netstat -an|find "ESTABLISHED" /c

title

local address本机的地址

foreign address 外部电脑电脑有联系的ip地址

### ping

[Reference](https://www.runoob.com/linux/linux-comm-ping.html)

-c 要求回应的次数

-I 指定网卡送出数据包

通过网卡enp3s0发送数据

ping 192.168.0.1 -c 1 -I enp3s0



ARP 将IP转换成MAC地址的协议

### ifconfig

[Reference](https://zhuanlan.zhihu.com/p/27551846)



第一行

* UP 代表这个网卡是开启状态
* RUNNING 代表这个网卡是处于网络连接状态
* MULTICAST 代表这个网卡是支持组播的

Ether 网卡mac地址

最后四行

* RX packets 网卡接收数据包信息
* RX errors 网卡接收数据时错误的信息
* TX packets 网卡发送数据包的信息
* TX errors 网卡发送数据是错误的信息

### ip

ip命令和ifconfig类似，但前者功能更强大，并旨在取代后者

[Reference](https://my.oschina.net/happyBKs/blog/1940675)

### Bridge

创建网桥

添加：ip link add name br0 type bridge

启动：ip link set br0 up

分配ip：ip addr add/del 192.168.0.100/24 dev br0

查看link：bridge link

31: veth0 state UP @veth1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 master br0 state forwarding priority 32 cost 2

删除

若要删除应先移除关联的所有端口，同时关闭端口的混杂模式并关闭端口以恢复至原始状态。

ip link set eth0 promisc off

ip link set veth0 down

ip link set dev eth0 nomaster

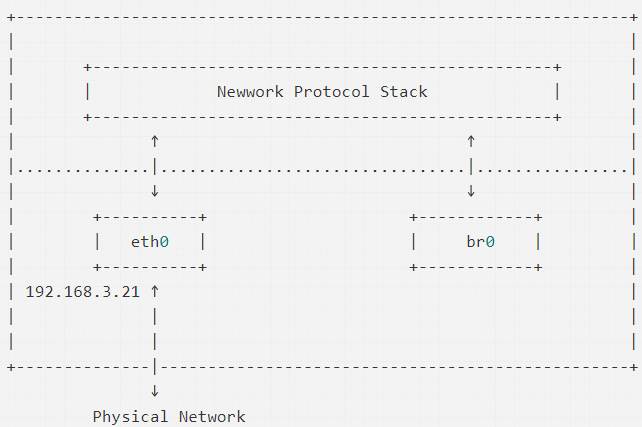
当网桥的配置清空后就可以将其删除：

ip link delete br0 type bridge

p1：rtnetlink answers file exists

ip addr flush dev veth0

当刚创建一个bridge时，它是一个独立的网络设备，只有一个端口连着协议栈，其它的端口啥都没连，这样的bridge没有任何实际功能



网桥原理

网桥在功能上和交换机几乎是一样的，只不过它是由软件实现这些功能。



veth虚拟网卡

Linux container 中用到一个叫做veth的东西，这是一种新的设备，专门为 container 所建。veth 从名字上来看是 Virtual ETHernet 的缩写，它的作用很简单，就是要把从一个 network namespace 发出的数据包转发到另一个 namespace。

veth设备是成对的，一个是container之中，另一个在 container 之外，在真实机器上能看到的。

仅有veth-pair设备，容器是无法访问网络的。因为容器发出的数据包，实质上直接进入了veth1设备的协议栈里。如果容器需要访问网络，需要使用bridge等技术，将veth1接收到的数据包通过某种方式转发出去。

创建：ip link add veth0 type veth peer name veth1

添加ip：ip addr add 192.168.3.101/24 dev veth

连网桥：ip link set dev veth0 master br0

ip link list

### route

-n 不解析名字

-v 显示详细的处理信息



Gateway为0.0.0.0的网络，通常是直连到网络设备上的，不需要网关的。

Flags标志说明：

U Up表示此路由当前为启动状态

H Host，表示此网关为一主机

G Gateway，表示此网关为一路由器

### arp

-n addrss使用数字方式显示

-s设置指定主机的mac地址映射

-i显示指定设备的arp缓冲区

添加静态映射

arp -i eth0 -s 192.168.1.6 ff:ee:ee:ee:ee:ee //将目标ip地址映射固定mac

### tcpdump

-n 表示不要解析域名，直接显示 ip。

-i 监听所有的网卡

tcpdump -n -i veth0



# 第四章 权限

[Reference](https://www.cnblogs.com/cisum/p/8005641.html)

权限管理实际上都是对有关的系统文件进行修改，与用户和用户组相关的信息都存放在一些系统文件中，这些文件包括/etc/passwd, /etc/shadow, /etc/group等。

UP 代表这个网卡是开启状态

RUNNING 代表这个网卡是处于网络连接状态

MULTICAST 代表这个网卡是支持组播的

## 3.3.1 Summary

## 3.3.3 User

### 查看用户

cat /etc/passwd



用户名:口令:UID:GID:注释性描述:主目录:Shell

* 口令：密码标识。真正的密码存在/etc/shadow，只有root有权限读写。
* UID：用户标识号。0管理员。1~499系统用户，用于系统用户，不能登陆。
* GID：组标识号。
* 主目录：登录后所处的目录。
* Shell：登陆后的Shell。

### adduser

useradd userName



默认创建用户组nan，用户目录为/home/nan1。

* -d 指定用户主目录；
* -g 指定用户组，删除初始组；-G 指定附加组，不删除原来组)。
* -c comment
* -s 指定用户的登录Shell。

nologin shell

使用nologin账号来运行程序，提升安全性

* 创建：useradd -s /sbin/nologin nan
* 运行程序：su -s /bin/bash -c "ls" nginx或者sudo -u nginx command

### userdel



默认删除用户组love，

* -r 删除用户主目录，默认不删

usermod

### usermod

修改用户

更改登陆目录usermod -d /home/hnlinux root

## 3.3.2 group

每个用户都属于某个用户组，一个组中可以有多个用户，一个用户也可以属于不同的组。

**用户组在adduser同时创建。userdel时同时删除。**

### 查看用户组

cat /etc/group



格式：groupName:口令:GID:UserList

### groupadd

groupadd group1

**groups** 查看当前登录用户的组内成员

**groupmod:** groupmod -n newName groupName (-n改组名 –g 改gid)

### groupdel

groupdel groupName先删除group的用户，才能删除群组

删除组成员

vi /etc/group

## 3.3.4 passwd

-l禁用账号。

-u 口令解锁。

-d 使账号无口令。

-f 强迫下次登录时修改口令。

超级用户修改/su不需要输入用户密码



普通用户直接输passwd

2./etc/shadow

moon:$6$007qnwIe$kS:17795:0:99999:7:::

用户名:口令:last\_update:最小时间间隔:最大时间间隔:警告时间:不活动时间:失效时间:标志

passwd文件所有用户可读，如果密码放passwd文件，即便加密也很容易破解。真正的加密后的用户口令字存放到shadow文件中，只有超级用户可读。

其他

**chown**将指定文件的拥有者改为指定的用户或组

chown -R mysql: mysql ./

su:Su switch use /sudo输入当前用户的密码。

sudo1.需要root赋予sudo权限

## 5.5 安全

添加端口

iptables -I INPUT -p tcp --dport 7001 -j ACCEPT

iptables VS Firewalld

从Cent7以后，iptables服务的启动脚本已被忽略。firewalld自身并不具备防火墙的功能，而是和iptables一样需要通过内核的netfilter来实现。

firewalld使用XML进行配置。除非非常特殊的配置，而应该使用firewalld-cmd。

配置文件：

/usr/lib/firewalld    #保存默认配置，避免修改他们；

/etc/firewalld    #保存系统配置文件，这些文佳你将覆盖默认配置

Pass: 6d%SW5e)[.xvzt@Z](mailto:.xvzt@Z)?

### SSL证书

[Reference](https://blog.csdn.net/liuchunming033/article/details/48470575)

* .key：私钥
* .csr：Certificate Signing Request，即证书签名请求，含公钥信息，生成证书时要把这个提交给权威的证书颁发机构。
* .crt： 即 certificate的缩写，即证书文件。
* .pem：用于导出，导入证书时候的证书的格式。

证书文件

一般以.crt结尾，根据文件内容编码格式，分为二种格式：

* pem：Privacy Enhanced Mail。
* der-Distinguished Encoding Rules,打开看是二进制格式,不可读.

X.509

一种证书格式，认证者总是CA或由CA指定的人。一份X.509证书是一些标准字段的集合，这些字段包含有关用户或设备及其相应公钥的信息。

CA根证书生成步骤

自签名得到根证书ca.crt，自签指是CA给自已颁发的证书。

1.Generate CA private key

openssl genrsa 2048 -out **ca.key**

2.Generate CSR

openssl req -new -key ca.key -out **ca.csr**

3.Generate Self Signed certificate（CA 根证书）

openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out **ca.crt**

颁发证书

服务器端用户证书：

1.private key

$openssl genrsa -des3 1024 -out server.key

2.generate csr

$openssl req -new -key server.key -out server.csr

3.generate certificate

$openssl ca **-cert ca.crt** -in server.csr -keyfile ca.key -out **server.crt**

server.crt 就是我们需要的证书

客户端用户证书：

$openssl genrsa -des3 1024 -out client.key

$openssl req -new -key client.key -out client.csr

$openssl ca **-cert ca.crt** -in client.csr -keyfile ca.key -out client.crt

生成pem格式证书

合并证书（crt）和私钥（key）来生成, 可以用以下方式

$cat client.crt client.key> client.pem

$cat server.crt server.key > server.pem

结果：

服务端证书：ca.crt, server.key, server.crt, server.pem

客户端证书：ca.crt, client.key, client.crt, client.pem

# 第五章 Install&Config

## 软件安装

### yum

yum list 所有已安装和可安裝的软件清单

yum search java 查找软件包

yum list installed

yum install xxx

设置代理

/etc/yum.conf

proxy=http://cn-proxy.jp.oracle.com:80/

### make

configure

生成Makefile:GUN规范

./configure –prefix=/usr 将该软件安装在 /usr 下面，执行文件就会安装在 /usr/bin 资源文件就会安装在 /usr/share

make 编译

默认安装

使用make编译安装软件时，若./configure不指定prefix，则：

可执行文件默认放在/usr/local/bin（/usr/local/libexec中也可能有一些），

库文件默认放在/usr/local/lib，

配置文件默认放在/usr/local/etc，

其它的资源文件放在/usr/local/share。

卸载

1.在原来的make目录下用一次make uninstall（前提是make文件指定过uninstall）

2.去上述目录里面把相关的文件一个个手工删掉。

-

配置文件

通常情况下，用户软件的配置文件放在自已的 home 目录下，比如 /home/riku/ 下，一般以 . 作为目录开始字符，比如 .gnome 。

而系统软件的配置文件一般放在 /etc/ 目录下。

### Systemd

#### Service配置

Unit

Systemd 可以管理所有系统资源。不同的资源统称为 Unit（单位）。共分12种。

* Service unit：系统服务
* Target unit：多个 Unit 构成的一个组
* Device Unit：硬件设备
* Mount Unit：文件系统的挂载点
* Automount Unit：自动挂载点
* Path Unit：文件或路径
* Scope Unit：不是由 Systemd 启动的外部进程
* Slice Unit：进程组
* Snapshot Unit：Systemd 快照，可以切回某个快照
* Socket Unit：进程间通信的 socket
* Swap Unit：swap 文件
* Timer Unit：定时器

Unit配置文件

路径：/usr/lib/systemd/system/xxx.service

[Unit]定义 Unit 的元数据，以及配置与其他 Unit 的关系

* Description 简短描述
* After
* Wants与当前 Unit 配合的其他 Unit，如果它们没有运行，当前 Unit 不会启动。

[Service]区块用来 Service 的配置，只有 Service 类型的 Unit 才有这个区块。

* Type 定义启动时的进程行为。simple/notify/idle
* WorkingDirectory
* EnvironmentFile环境参数文件
* User
* ExecStart启动当前服务的命令
* Restart定义何种情况 Systemd 会自动重启当前服务，可能的值包括always（总是重启）、on-success、on-failure、on-abnormal、on-abort、on-watchdog
* LimitNOFILE

[Install]用来定义开机启动行为

* WantedBy字段：表示该服务所在的 Target。

比如Target 是multi-user.target。在这个组里的所有服务，都将开机启动。

修改配置文件后重启

修改配置文件以后，需要重新加载配置文件，然后重新启动相关服务。

# 重新加载配置文件

$ systemctl daemon-reload

# 重启相关服务

$ sudo systemctl restart etcd

开机启动

systemctl enable etcd.service

开机时，Systemd执行/etc/systemd/system目录里面的配置文件， enable用于建立符号链接关系。

# enable等同于

$ ln -s '/usr/lib/systemd/system/etcd.service' '/etc/systemd/system/multi-user.target.wants/etcd.service'

如果配置文件里面设置了开机启动，systemctl enable命令相当于激活开机启动。

#### systemctl

systemctl是 Systemd 的主命令，用于管理系统。

#重载所有修改过的配置文件

systemctl daemon-reload

#### journalctl

-k 内核日志

-u 按服务过滤

-f 同tail -f

journalctl -f -u kubelet

## 5.1 服务器

### Nginx

查看配置文件位置 nginx –t

配置

1. upstream cluster1 {
2. server localhost:8080;
3. server localhost:8081;
4. }
6. server {
7. listen       81;
8. server\_name  localhost;
9. client\_max\_body\_size 1024M;
11. location /home {
12. proxy\_pass http://cluster1;
13. proxy\_set\_header Host $host:$server\_port;
14. }
15. }

上面upstream默认负载均衡策略为RR

http服务器

1. # 所有静态请求都由nginx处理，存放目录为html
2. location ~ \.(gif|jpg|jpeg|png|bmp|swf|css|js)${
3. root    e:\wwwroot;
4. }
6. # 所有动态请求都转发给tomcat处理
7. location ~\.(jsp|do)$ {
8. proxy\_pass  http://test;
9. }
10. error\_page  500 502 503 504 /50x.html;
11. location =/50x.html {
12. root   e:\wwwroot;
13. }

Configuration summary

+ using system PCRE library

+ OpenSSL library is not used

+ using system zlib library

nginx path prefix: "/usr/local/nginx"

nginx binary file: "/usr/local/nginx/sbin/nginx"

nginx modules path: "/usr/local/nginx/modules"

nginx configuration prefix: "/usr/local/nginx/conf"

nginx configuration file: "/usr/local/nginx/conf/nginx.conf"

nginx pid file: "/usr/local/nginx/logs/nginx.pid"

nginx error log file: "/usr/local/nginx/logs/error.log"

nginx http access log file: "/usr/local/nginx/logs/access.log"

nginx http client request body temporary files: "client\_body\_temp"

nginx http proxy temporary files: "proxy\_temp"

nginx http fastcgi temporary files: "fastcgi\_temp"

nginx http uwsgi temporary files: "uwsgi\_temp"

nginx http scgi temporary files: "scgi\_temp"

1)启动操作

nginx -c /usr/local/nginx/conf/nginx.conf

2)停止操作

从容停止Nginx

快速停止Nginx：

kill -TERM 主进程号

强制停止Nginx：

kill -9 主进程号

3)

nginx -s reload

reload

./nginx -s reload

reload会重新加载配置文件，服务不中断，出错会rollback用上一次正确配置文件。

Request\_body

[Reference](http://www.madblog.cn/posts/e4dd5bbae50fa621.html)

### Mysql

Mgr

mysql [-h slc11fsp.us.oracle.com] -uroot -p

启动服务

service mysqld start

版本：mysql –V；或者select version();

Tables：show databases/tables;

查看表结构

desc table\_name;

Show DDl: show create table ld\_cart;

indexs

Show: show index from fund\_history\G;

Create: create index idx\_fund\_name on fund(name);

GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;

### 安装centos

[Reference](https://www.linuxidc.com/Linux/2014-10/108014.htm)

1.修改bios启动顺序

2.进入安装界面，按e，修改参数

linuxefi /images/pxeboot/vmlinuz inst.stage2=hd:/dev/sdb4 nomodeset quiet

initrdefi /images/pxeboot/initrd.img

然后使用Ctrl+X启动即可进入正常安装流程

配置网卡

查看地址ip addr

1.vi /etc/sysconfig/network-scripts/ifcfg-enp3s0修改为onboot=yes

2.reboot

静态ip

[Reference](https://blog.csdn.net/johnnycode/article/details/40624403)

盒盖息屏

[Reference](https://www.jianshu.com/p/cfa90c8a900f)

## 5.2 大数据处理

### Flume

启动

bin/flume-ng agent -c ./conf -f ./conf/2.conf -n agent1 -Dflume.root.logger=INFO,console

* -c conf 指定flume自身的配置文件所在目录
* -f conf/netcat-logger.con 指定我们所描述的采集方案
* -n a1 指定agent的名字

组合

[Reference](http://www.voidcn.com/article/p-wgqahebc-ev.html)

### Es

安装 [Reference](https://www.cnblogs.com/yufeng218/p/8331438.html)

启动es

su elastic

/usr/local/elasticsearch-5.6.0/bin/elasticsearch –d

chown elastic /usr/local/elasticsearch-6.8.0/ -R

外网访问

vi elasticsearch.yml

network.bind\_host: 0.0.0.0 (network.host: 0.0.0.0 5.6?)

安装问题汇总

<https://blog.51cto.com/10950710/2124131>

vm

sysctl -w vm.max\_map\_count=262144

修改jvm ops

vi config/jvm.options

3.修改es文件拥有着 chown elastic:elastic -R ./\*

#### kibana

1.kibana.yml server.host: "0.0.0.0"

Test url

http://localhost:5601/

### Kafka

安装

[Reference](https://juejin.im/post/5c1f9b3bf265da61590ba9b1)

ver: kafka\_2.11-2.1.0

启动/停止

* 启动 ./bin/kafka-server-start.sh ./config/server.properties &
* 停止 ./bin/kafka-server-stop.sh ./config/server.properties &

配置

vi ./config/server.properties

1.Zookeeper

* zookeeper.connect=localhost:2181
* zookeeper.connection.timeout.ms=6000

2. socket settings

* listeners：the socket server listens on
* advertised.listeners：broker的地址。配置外网访问.(0.9.x后新增，废弃了 advertised.host.name 和 advertised.host.port)

3.broker

broker.id：broker id

#### Topic

* Topic列表： bin/kafka-topics.sh -list -zookeeper localhost:2181
* 创建topic：bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test
* 话题描述： bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test
* 删除topic： bin/kafka-topics.sh --zookeeper 127.0.0.1:2181 --delete --topic test

启动Producer

./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

启动Consumer

0.10.0版本：/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test [--from-beginning]

1.1.0版本：./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning

## 5.3 分布式

### 3.1 Docker

#### 安装

[Reference](https://www.runoob.com/docker/centos-docker-install.html)

yum install docker-ce docker-ce-cli containerd.io

启动：systemctl start docker

vi /lib/systemd/system/docker.service

仓库地址配置/etc/yum.repos.d/docker-ce.repo

Uninstall

yum remove -y docker docker-latest docker-ce docker-common \

docker-selinux container-selinux \

docker-engine \

docker-ce-cli \

docker-client docker-client-latest \

docker-logrotate docker-latest-logrotate

containerd.io

delete conf

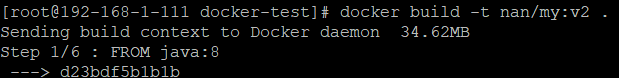
/var/lib/docker

/etc/docker

#### 3.1.1 镜像

##### 创建镜像

docker build -t nan/my:v2 .



* --t/tag: 镜像的名字及标签
* 最后的 . 代表在当前目录下寻找 dockerfile,也可-f指定位置。

Dockerfile

FROM java:8

VOLUME /tmp

COPY ./my-1.0-SNAPSHOT.jar my.jar

RUN bash -c "touch /my.jar"

EXPOSE 8080

ENTRYPOINT ["java","-jar","my.jar"]

~

##### 查看镜像

docker images [name]



#### docker hub

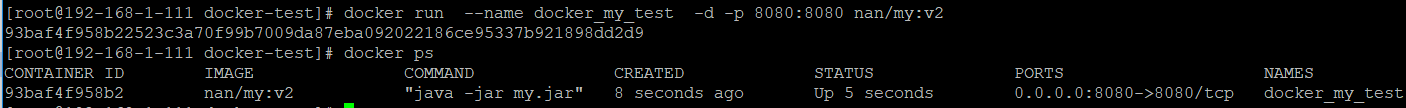
docker pull/push + name

docker search +name

#### 3.1.2 容器

##### 创建容器

docker run --name names -d -p 80:80 repository:tag



* -d 后台运行
* -p 端口映射 -P容器端口随机

Docker run 用镜像创建容器 + docker start。

##### 启动容器

启动：docker start/restart container\_id

停止：docker stop container\_id或container\_name

删除：docker rm container\_id

##### 管理容器

logs

查看内部标准输出 docker logs -f bf08b7f2cd89

管理容器 docker exec -it docker\_nginx\_v1 bash

ps

docker ps [-a]

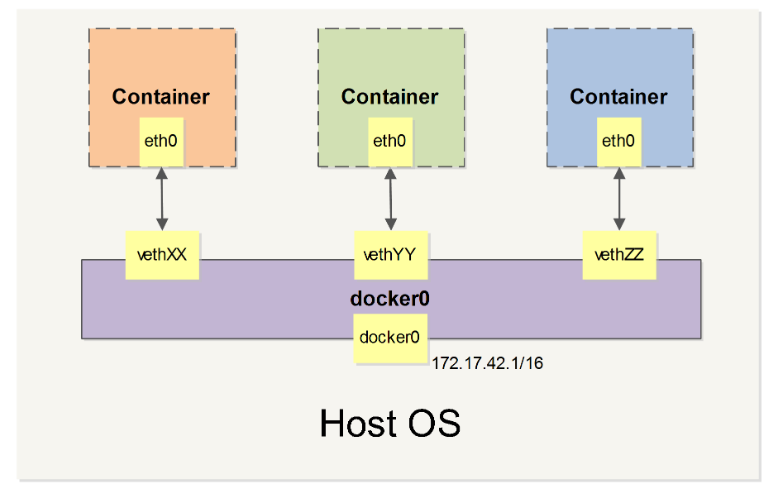
container id image command port names

d11a0358e403 nginx:v1 "nginx -g 'daemon ..." 0.0.0.0:88->80/tcp docker\_nginx\_v2

查看容器信息

docker inspect container\_id

#### 3.1.3 网络



### 6.3.2 k8s

#### 安装

[Reference](https://kubernetes.io/docs/tasks/tools/install-kubectl/)

配置对多集群的访问:[Reference](https://kubernetes.io/zh/docs/tasks/access-application-cluster/configure-access-multiple-clusters/)

kubectl version

##### flannel

master和node上都要安装Flannel网络

配置

vi /usr/lib/systemd/system/flanneld.service

vi /etc/sysconfig/flanneld

etcdctl --endpoints=${ETCD\_ENDPOINTS} put /kubernetes/network/config '{"Network":"'${CLUSTER\_CIDR}'", "SubnetLen": 24, "Backend": {"Type": "vxlan"}}'

etcdctl --endpoints=${ETCD\_ENDPOINTS} get /kubernetes/network/config

--cacert=/etc/kubernetes/ssl/ca.pem --key=/etc/etcd/ssl/etcd-key.pem --cert=/etc/etcd/ssl/etcd.pem

通过http向etcd增加key

curl -X PUT http://192.168.0.110:2379/v2/keys/kubernetes/network/config -d value='{"Network":"172.30.0.0/16", "SubnetLen": 24, "Backend": {"Type": "vxlan"}}'

#### master节点

安装kubernetes： yum -y install kubernetes

##### kube-apiserver

vi /usr/lib/systemd/system/kube-apiserver.service

vi /etc/kubernetes/apiserver

systemctl daemon-reload

systemctl restart kube-apiserver.service

##### kube-controller-manager

vi /usr/lib/systemd/system/kube-controller-manager.service

systemctl daemon-reload

systemctl restart kube-controller-manager.service

查看节点

curl http://192.168.0.110:8080/api/v1/nodes

##### kube-scheduler

vi /usr/lib/systemd/system/kube-scheduler.service

systemctl enable kube-scheduler

#### Node节点

1.安装kubernetes

yum -y install kubernetes&vi /etc/kubernetes/config

* KUBE\_LOGTOSTDERR="--logtostderr=true"
* KUBE\_LOG\_LEVEL="--v=0"
* KUBE\_ALLOW\_PRIV="--allow-privileged=false"
* KUBE\_MASTER="--master=http://k8s-master:8080"

##### kubelet

vi /usr/lib/systemd/system/kubelet.service

vi /etc/kubernetes/kubelet

* KUBELET\_ADDRESS="--address=0.0.0.0"
* KUBELET\_HOSTNAME="--hostname-override=k8s-node-1"
* KUBELET\_API\_SERVER="--api-servers=http://k8s-master:8080"
* KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=registry.access.redhat.com/rhel7/pod-infrastructure:latest" #确认此镜像存在，否则pod无法启动
* KUBELET\_ARGS=""

/etc/kubernetes/kubelet.conf

–client-ca-file

3.启动服务

systemctl daemon-reload

systemctl restart kubelet

##### kube-proxy

/usr/lib/systemd/system/kube-proxy.service

Vi

##### 集群安全设置

[Reference](https://o-my-chenjian.com/2017/04/25/Security-Settings-Of-K8s/)

#### 3.2.2 kubectl

kubectl version：Print the client and server version information

Client Version: version.Info{Major:"1", Minor:"6", GitVersion:"v1.6.0", GoVersion:"go1.7.5", Platform:"linux/amd64"}

Server Version: version.Info{Major:"1", Minor:"5", GitVersion:"v1.5.2", GitTreeState:"clean", GoVersion:"go1.7.4", Compiler:"gc", }

##### kubeconfig

kubeconfig文件记录k8s集群的各种信息，对集群构建非常重要。

* kubectl从~/.kube/config，即kubectl的kubeconfig文件中获取访问kube-apiserver的地址，证书和用户名等信息
* kubelet/kube-proxy等在Node上的程序进程同样通过bootstrap.kubeconfig和kube-proxy.kubeconfig上提供的认证与授权信息与Master进行通讯

查询集群地址和凭证

kubectl config view

1. apiVersion: v1
2. clusters:
3. - cluster:
4. certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSU....
5. server: https://192.168.0.110:6443
6. name: kubernetes
7. contexts:
8. - context:
9. cluster: kubernetes
10. user: kubelet-bootstrap
11. name: default
12. current-context: default
13. kind: Config
14. preferences: {}
15. users:
16. - name: kubelet-bootstrap
17. user:
18. token: 5123398c0116eb8617398e4c38c3be

certificate-authority-data 是 ca.pem 的base64加密的内容

client-key-data:

kubectl config get-contexts

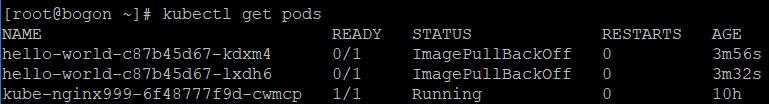
get

-o/output=json/yaml/wide|

##### 管理pods

查看pod

kubectl get pods [--all-namespaces]



删除：kubectl delete pods [pod-name]

kubectl describe pod <pod-name>

创建pod

kubectl run <pod-nane> --image=IMAGE [--port=port] [--replicas=NUM]

--replicas 副本数量

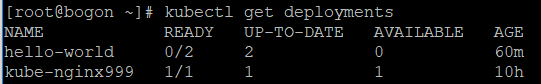
示例：

kubectl run nan-my1 --image=nan/my:v1 --port=8080

创建过程：kubectl http--> kube-apiserver --> deployment 持久化到etcd🡪 创建ReplicaSet和Pod，每个 Pod 都被调度到合适的节点 🡪 Kubelet按PodSpec定义，创建并启动容器

##### 管理Deployment

kubectl get deploy/deployments [name]



kubectl delete deployments name

显示 ReplicaSet 的信息

kubectl get replicasets/rs

kubectl describe replicasets

创建

kubectl create deployment NAME --image=image [--dry-run]

扩容

kubectl scale deployment nan-my1 --replicas 2

更新镜像

kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1

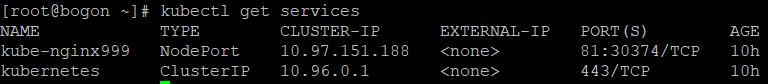
回滚

kubectl rollout undo deployment/nginx-deployment

##### 管理Service

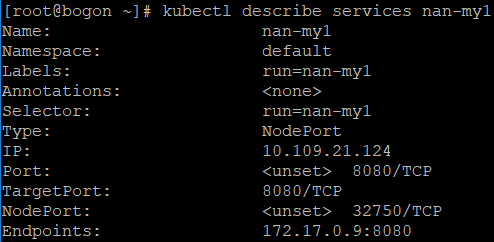
查看service

kubectl get svc/services



kubectl delete services <name>

kubectl describe services [label]



发布service

expose <-f fileName | type name> [--port=port] [--target-port=number-or-name] [--type=type]

target-port 容器端口

将资源暴露为新的Kubernetes Service，资源包括：

pod（po），service（svc），replication controller（rc），deployment（deploy），replica set（rs）

示例：

1. 将pod暴露给外网

kubectl expose deployment nan-my1 --type=NodePort

2.为rc创建service，并通过Service的80端口转发至容器的8000端口上。

kubectl expose rc nginx --port=80 --target-port=8000

3.由“nginx-controller.yaml”中指定的type和name标识的RC创建Service

kubectl expose -f nginx-controller.yaml --port=80 --target-port=8000

minikube查看service

minikube service kube-nginx999 [–url] [-n NAMESPACE]



##### 查看logs

查看指定pod的日志

kubectl logs <pod\_name>

kubectl logs -f <pod\_name> #类似tail -f

查看指定pod中指定容器的日志

kubectl logs <pod\_name> -c <container\_name>

PS：看Docker容器日志

docker logs <container\_id>

日志

minikube logs

##### kubectl create

create -f fileName

通过配置文件名或stdin创建一个集群资源对象。支持json和yaml格式的文件。

示例：

kubectl create -f ./pod.json

##### 查看集群

kubectl cluster-info

/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

查看组件状态

kubectl get cs

kubectl get ep/endpoints

查看未授权的 CSR 请求

kubectl get csr (Certificate Signing Request)

### Volume存储数据卷

exception

Orphaned pod "cdb27b68-9e8d-4708-8179-b5579dab861f" found, but volume paths are still present on disk.

rm -rf /var/lib/kubelet/pods/\*

### 6.3.3 etcd

etcdctl version

ETCDCTL\_API=3

#### 配置

vi /usr/lib/systemd/system/etcd.service

vi /etc/etcd/etcd.conf

#[Member]

ETCD\_DATA\_DIR：etcd数据保存目录

ETCD\_LISTEN\_CLIENT\_URLS：供外部**客户端**使用的url

ETCD\_ADVERTISE\_CLIENT\_URLS：暴露给外部**客户端**使用的url

ETCD\_NAME：etcd实例名称

#[Clustering]

ETCD\_LISTEN\_PEER\_URLS：集群内部通信使用的URL

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS：广播给集群内其他成员访问的URL

ETCD\_INITIAL\_CLUSTER：初始集群成员列表

ETCD\_INITIAL\_CLUSTER\_TOKEN：集群的名称

ETCD\_INITIAL\_CLUSTER\_STATE：初始集群状态，new为新建集群

rm -rf /var/lib/etcd/default.etcd

systemctl daemon-reload

systemctl restart etcd

etcdctl cluster-health

配置TSL

etcdctl member list --endpoints=https://192.168.0.111:2379 --cacert=/etc/kubernetes/ssl/ca.pem --key=/etc/etcd/ssl/etcd-key.pem --cert=/etc/etcd/ssl/etcd.pem

#### CMD

* -w, --write-out="simple" 输出格式(fields, json, protobuf, simple, table)
* --endpoints=[127.0.0.1:2379] ：gRPC endpoints

Restful api

curl http://192.168.0.110:2379/v2/keys/

#### 查看集群

etcdctl endpoint status

etcdctl [--write-out=table] endpoint health

etcdctl member list

etcdctl endpoint health --endpoints=https://127.0.0.1:2379 --cacert=/etc/kubernetes/ssl/ca.pem --key=/etc/kubernetes/ssl/etcd-key.pem --cert=/etc/kubernetes/ssl/etcd.pem

示例：

etcdctl --endpoints=http://192.168.0.110:2379 member list -w table

集群成员

跟集群成员相关的命令如下：

member add Adds a member into the cluster

member remove Removes a member from the cluster

member update Updates a member in the cluster

member list Lists all members in the cluster

示例

etcdctl member list -w table



CRUD

* 增：etcdctl --endpoints=$ENDPOINTS put foo "Hello World!"
* 查：etcdctl [--endpoints=$ENDPOINTS] [--write-out="json"] get foo
* 删：etcdctl --endpoints=$ENDPOINTS del key

Failed to connect to apiserver: the server has asked for the client to provide credentials

Failed to connect to apiserver: Forbidden: "/healthz?timeout=1s

### kubeadm

### Minikube

[Reference](https://yq.aliyun.com/articles/221687)

安装

**1.下载minikube**

curl -Lo minikube http://kubernetes.oss-cn-hangzhou.aliyuncs.com/minikube/releases/v1.4.0/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

启动

minikube start --vm-driver=none

查看状态：minikube status

minikube dashboard

kubectl proxy --port=8001 --address='192.168.0.5' --accept-hosts='^.\*' &

更新

如需更新minikube，需要更新 minikube 安装包

minikube delete 删除现有虚机，删除 ~/.minikube 目录缓存的文件

vm driver

~/.minikube/machines/minikube/config.json

## 5.4 开发工具

### Git

Clone git clone https://github.com/lovemooner/moon.git

Branch

本地分支：git branch。【-a】 切换分支：git checkout branchName

Commit

git commit -a

git push origin master

git push origin HEAD:refs/for/reserved (or other branch you wish to push, start with refs/for/)（push内容在其他库并非立即可见）

撤销本地修改 git checkout filename

删除文件 git rm filename (-r 文件夹)

git log

--stat 加stat，仅显示修改的文件)

查看单个文件

git log <filename> 查看所有commit历史(包括本地未push到库里的)

--pretty=oneline pretty

-n 显示最后的几次提交日志

-p 详细包括diff内容

Config

查看当前配置 git config –list

配置：git config [--global] user.email "[email address]"

设置proxy：git config --global https.proxy http://127.0.0.1:1080

system级别 /global（用户级别）/local（当前仓库）三个 设置先从system-》global-》local

删除 git config --global --unset https.proxy

remote

git remote set-url origin [url]

-- INSERT --

#### Base

[Reference](https://git-scm.com/book/zh/v2)



Workspace：工作区

Index / Stage：暂存区

Repository：仓库区（或本地仓库）

Remote：远程仓库

入门教程：[Reference](https://mp.weixin.qq.com/s?__biz=MzI2OTQ4OTQ1NQ==&mid=2247485703&idx=1&sn=69d22b19f48f6bb3cecec163f4e946eb&chksm=eadecd47dda944517f9060c59952759ad65f605adba53a2ea2b34bce1058252f799de59d9ce7&mpshare=1&scene=1&srcid=#rd])

[Reference](https://blog.csdn.net/mine_song/article/details/70770467)



全红：删除 全绿：增加

#### 分支管理

#### 分支查看

git branch -a 查看所有分支，包括local/remote

git branch 查看本地分支

#### 分支创建

git branch mybranch 创建分支

git checkout -b branchName：相当于git branch+git checkout

分支切换





分支合并

Git merge dev：合并dev到当前分支

git rebase 和 merge 有什么区别？M(Ali)

#### log

**-p 详细包括diff内容**

git log -p <filename>查看某个文件的修改历史

**-n**

显示最后的几次提交日志

git log -2

git log –p -2 查看最近2次更新内容

**show**

git show <commit-hash-id>查看某次commit的修改内容  
git show [--stat]

查看本地修改了还未提交

git diff filename

权限

查看key

cd ~/.ssh

Git VS Svn

svn更像一个文件服务器+版本控制，集中式管理文件。Git是一个分布式版本控制系统

1. 分支管理

Git擅长的是程序代码的版本化管理。

场景一:在分支开发新功能，需要新建分支修复紧急的bug

svn:创建branch--> 将代码下载到本地--> eclipse打开项目

git :创建branch-->同一工作目录下切换分支

svn下载的是完整代码库，速度慢，特别对于大型的项目，文件数达上万个以上，将非常的耗时，而git下载的是文件的镜像，大文件无需下载，而且当打开项目时需要编译，配置开发环境。像大型项目整个库的代码本地硬盘都放不下。git只下载镜像文件，需要时才下载。

2.离线提交

为了不让开发的过程代码影响其他人，svn一般task完成时一次性提交。而git在开发过程可以自由的提交，并且只有自己可见，实现本地的版本管理，非常方便。

### maven

#### jar包管理策略

**1.公共依赖在的父项目中声明**

统一版本号,方便维护，尽量避免出现版本冲突。

1. **<properties>**
2. **<spring.version>**spring4.2.4**</spring.version>**
3. **<properties>**
5. **<dependencyManagement>**
6. **<dependencies>**
7. **<dependency>**
8. **<groupId>**org.springframework**</groupId>**
9. **<artifactId>**spring-beans**</artifactId>**
10. **<version>**${spring.versio}**</version>**
11. **</dependency>**
12. **</dependencies>**
13. **</dependencyManagement>**

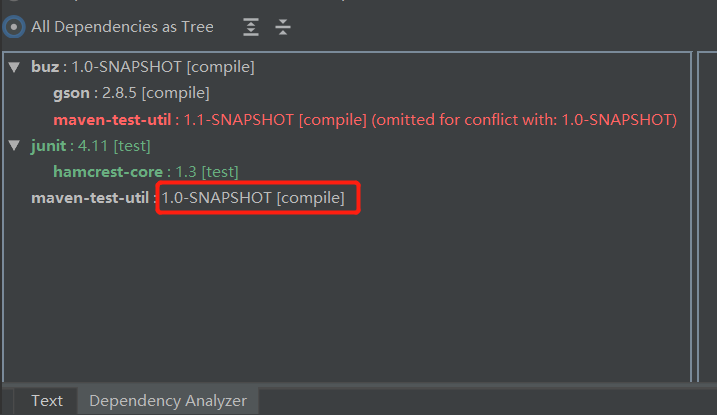
子模块使用spring-bean的jar包(不用再定义版本)：

1. **<dependencies>**
2. **<dependency>**
3. **<groupId>**org.springframework**</groupId>**
4. **<artifactId>**spring-beans**</artifactId>**
5. **</dependency>**
6. **</dependencies>**

#### dependency

mvn dependency:tree [-Dverbose] >tree.txt

-Dverbose 查看jar冲突,或者用Maven helper



出现冲突原因

[Reference](https://www.cnblogs.com/ptqueen/p/7985198.html)

1.高版本兼容低版本,所以不会出现问题

2.高版本不兼容低版本。

# 附件

## 命令速查

Base

uname –a

locate cmd

whereis

### tail

-f 等同于--follow=descriptor，根据文件描述符进行追踪，当文件改名或被删除，追踪停止

-F 等同于--follow=name --retry，根据文件名进行追踪，并保持重试，即该文件被删除或改名后，如果再次创建相同的文件名，会继续追踪

### cat

cat > /root/test1.txt <<EOF

>

EOF

### redis

redis

1)启动： redis-server.exe redis.windows.conf

开启memcache，并连接测试：

以守护进程模式启动memcache

memcached-1.5.2/bin/memcached -d -l slc11fsp.us.oracle.com -p 11211 -m 2048 -u root

### hostnamectl

# 显示当前主机的信息

$ hostnamectl

# 设置主机名。

$ sudo hostnamectl set-hostname k8s-master

### ssh

[Reference](https://blog.csdn.net/u011054333/article/details/52443061)

-v debug

-p port

ssh -v -p 8080 nadong@slc11fsp.us.oracle.com

TODO1:如何防止中间人攻击？

## 开发工具

### Jar

java-jar

-c 创建一个jar包

-t 显示jar中的内容列表

-x 解压jar包

-u 添加文件到jar包中

-f 指定jar包的文件名

-v生成详细的报造，并输出至标准设备

1）解压

Jar -xvf test.ear

2）创建

Jar -cvf test.jar ./\*

3）更新

jar -uvf RESTWebService.ear APP-INF/lib/AdfHcmUsersRestModel.jar

4）查看

jar tvf test.jar

5）搜索：

jar tvf ZhcLib.jar | grep 'pattern'

搜索整个目录下的所有jar文件：

find . -name "\*.jar"|awk '{print "jar -tvf "$1}' | sh | grep ".\*.xml"

## other

### Zookeeper

启动

./bin/zkServer.sh start &

Kafka自带启动

./bin/zookeeper-server-start.sh ./config/zookeeper.properties

### svn

svn ls svn://slc02vlf.us.oracle.com:3690/repos

svn checkout svn://slc02vlf.us.oracle.com:3690/repos/parking

# log

svn log url

### vnc

• Kill VNC: vncserver -kill :1

• Create VNC: vncserver -geometry 1910x1050

• VNC set copy: vncconfig -nowin &

• List: vncserver -