1. 服务器分区

**1.根目录**

根的数据确实在磁盘上，但根不在磁盘上，而是在内核自身中。内核在装载之前，是没有根的概念的。分区机制是软件提供的，文件系统也是软件提供的。调用bootloader从某个分区中将内核加载到内存完成初始化后，自己创造了1个根，然后挂载分区，从而使根作为访问其他分区的入口。

**2.和windows的区别**

Linux和windows都采用的是树型结构。最上层是根目录，其他的所有目录都是从根目录出发而生成的。在linux中，无论操作系统管理几个磁盘分区，目录树只有一个。而windows树型结构的根是磁盘分区的盘符，有几个分区就有几个树型结构，他们之间的关系是并列的。

**3.分区规划**

（1）默认安装下的分区



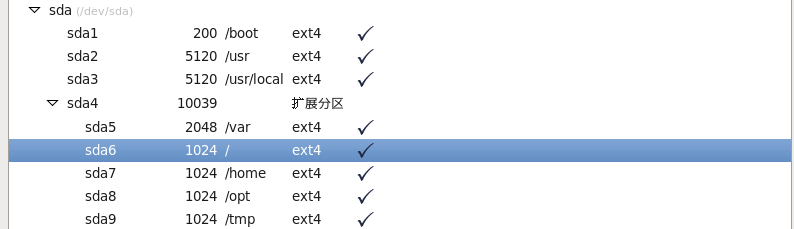
如果选择系统默认安装，会分为/和/boot两个分区。

（2）简单分区

只要分区/及swap即可，并预留一个备用的剩余磁盘容量作为将来练习。

（3）目录和分区的关系

在系统安装时自定义分区，需要指定分区的挂载目录。然后系统安装会将对应目录的文件安装到该分区下。



（4）服务器分区方案（安装时的下拉列表中就这些项）

将需要较大容量的目录、读写较为频繁的目录单独出来，这样如果出错不会影响到/分区。

1）/：为除了保存到其他分区下的目录之外的目录所需要的空间，FHS建议/所在分区应越小越好，通常2G以上，LVM格式。

2）/boot：100M即可。

3）/usr：默认安装程序目录，通常10G。

4）/usr/local：用户安装程序目录，通常10G以上。

5）/opt：附加应用程序，1G最多。

6）/home：通常每个用户100M即可，用户不多2G即可。

7）/tmp：最多1G即可。

8）/var：视服务器用途而定，剩下全部空间，一般2G左右。

8）/swap：通常分配和物理内存一样的空间，如果需要大的内存数据库，至少4G。

1. 编译安装

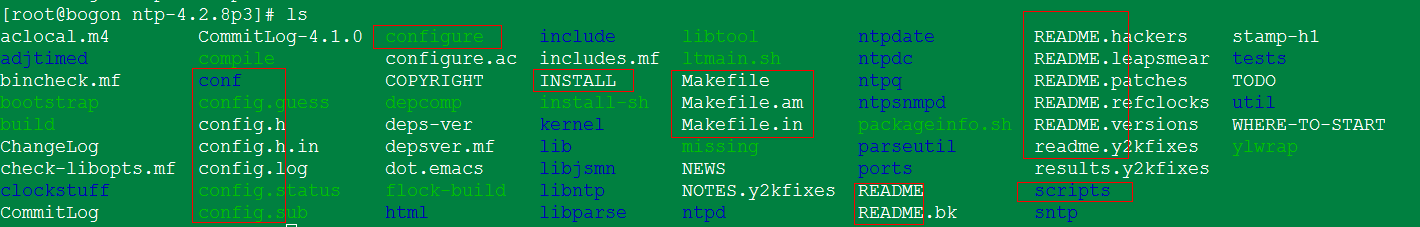
以安装npt为例。

**1.下载、解包**

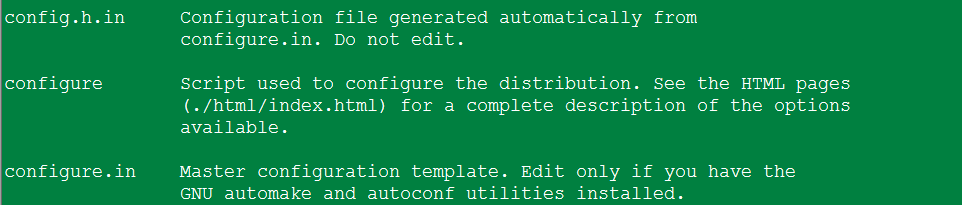
wget <http://linux.vbird.org/linux_basic/0520source/ntp-4.2.8p3.tar.gz>

tar -zxvf ntp-4.2.8p3.tar.gz

**2.文件列表**

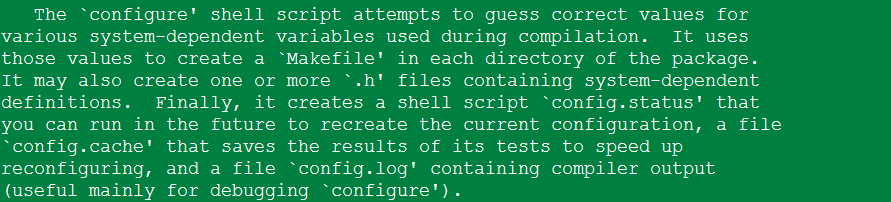


conf为源码配置，include为引用lib，scripts为一些脚本（也有源码）。README文件为介绍文件/目录内容：

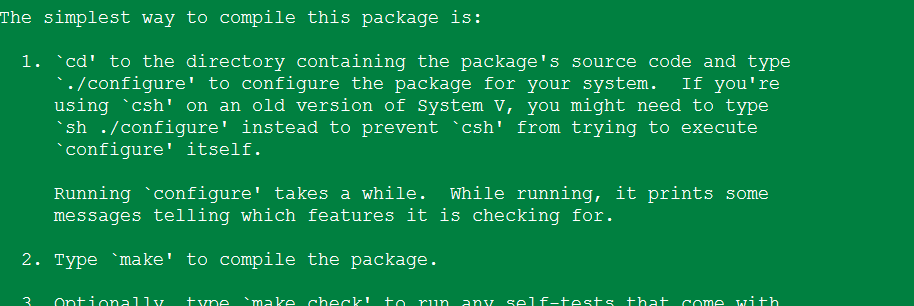


INSTALL为安装说明，里面有：

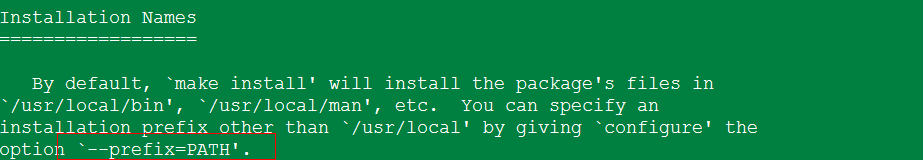
（1）configure作用说明

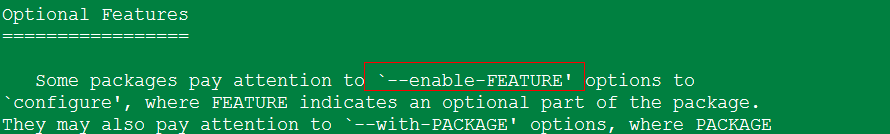


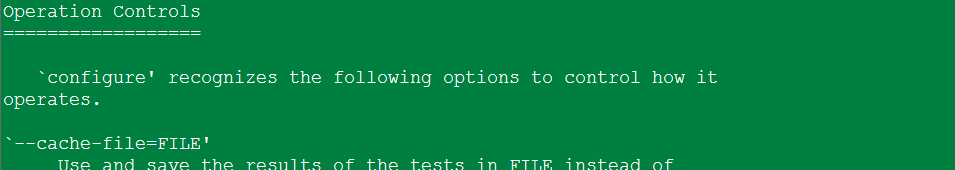
（2）编译说明



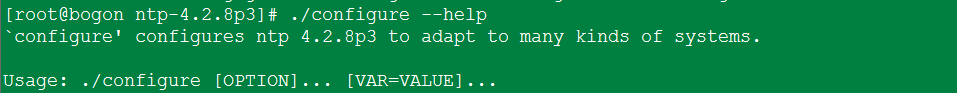
（3）Installation Names/ Optional Features/ Operation Controls等





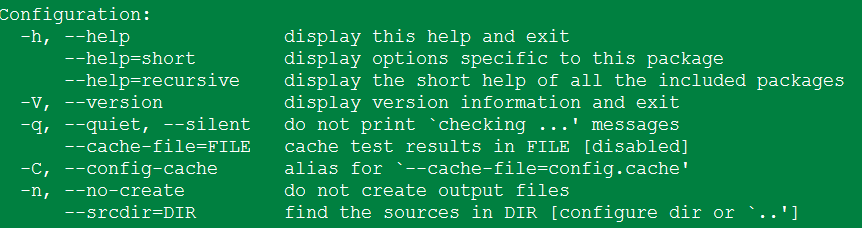


**3.查看configure帮助**

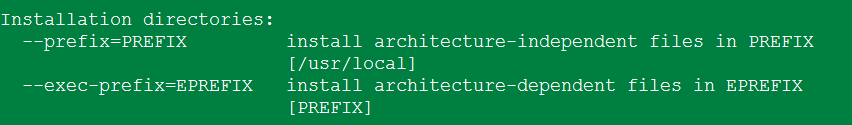


里面有README中的全部说明，包括：

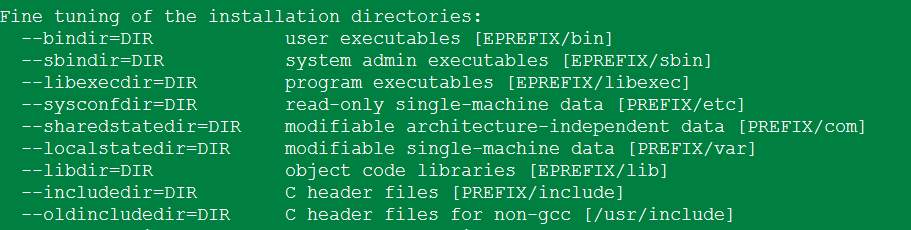
（1）配置



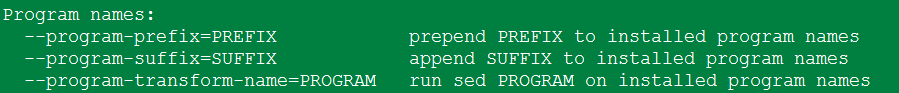
（2）安装目录



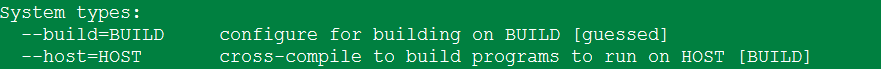
还有更细致目录控制



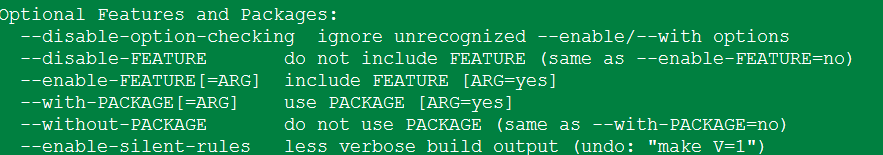
（3）程序名称设置



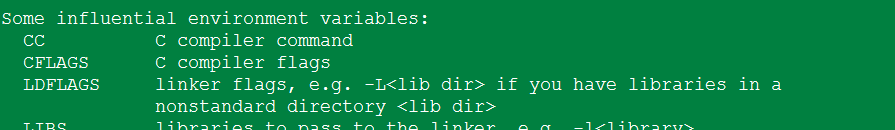
（4）系统类型



（5）功能控制



（6）环境变量



**4.正式安装**





1. PAM模块

PAM是一套应用程序接口，提供了一连串的验证机制，只要使用者将验证阶段的需求告知PAM后，PAM就能够回报使用者验证的结果（成功或失败）。PAM可以提供给其他程序调用。PAM用来进行验证的数据称为模块（Modules），每个PAM模块的功能都不太相同。

**1.PAM模块设置语法**

以passwd指令调用PAM为例，执行流程如下：

（1）使用者执行/usr/bin/passwd程序，输入密码。

（2）passwd调用PAM模块进行验证。

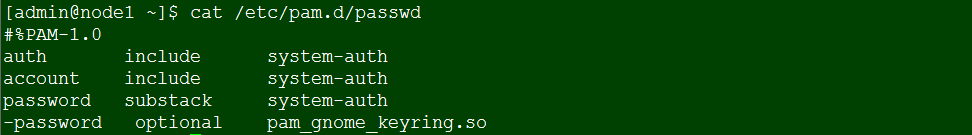
（3）PAM模块会到/etc/pam.d/找寻与程序（passwd）同名的配置文件。

（4）依据/etc/pam.d/passwd内的设置，引用相关的PAM模块逐步进行验证分析。

（5）将验证结果（成功、失败以及其他讯息）回传给passwd程序。

（6）passwd程序根据PAM回传结果决定下一个动作。

因此，/etc/pam.d/passwd中的配置最重要。



第1行#后为说明PAM版本，其余以#开头的均为注释。

每1行为1个独立的验证流程，分为3个字段，分别是验证类别（type）、控制标准（flag）、PAM的模块与该模块的参数。

1）验证类别

有如下4类：

auth：认证，主要用来检验使用者的身份验证，通常需要密码，后接模块是用来检验使用者的身份。

account：帐号，大部分是在进行authorization（授权），主要是在检验使用者是否具有正确的使用权限。

session：期间，session管理的就是使用者在这次登陆（或使用这个指令）期间，PAM所给予的环境设置。通常在记录使用者登陆与登出时的信息。如使用su或sudo指令时， /var/log/secure里会有相应pam的说明，而且记载的数据是session open、session close信息。

password：密码，主要在提供验证的修订工作，如修改/变更密码。

4种验证类型也通常按上述顺序进行。

2）验证的控制旗标

即验证通过的标准，管控该验证的放行方式，也有如下4类：

required：如果成功则带有success标志，失败则带有failure标志，但不论成功或失败都会继续后续的验证流程。由于后续的验证流程可以继续进行，通常用于日志记录。

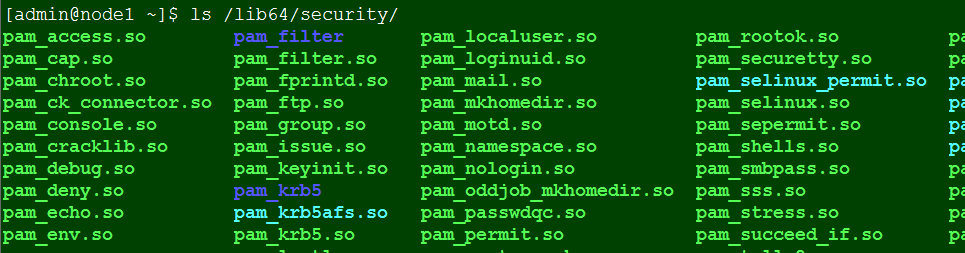
requisite：若验证失败则立刻回报原程序failure标志，并终止后续的验证流程。若验证成功则带有success标志并继续后续的验证流程。与required最大的区别在于失败时是否继续验证下去。

sufficient：若验证成功则立刻回传success给原程序，并终止后续的验证流程；若验证失败则带有failure标志并继续后续的验证流程。

optional：通常显示一些通知，并不是用于验证。

（2）常用模块简介

PAM模块放置在/lib64/security/下：



常用模块说明如下：

pam\_securetty.so：限制系统管理员只能够从安全的终端机登陆（tty）。

pam\_nologin.so：这个模块可以限制一般使用者是否能够登陆主机。

pam\_selinux.so：SELinux是个针对程序设置管理权限的功能。

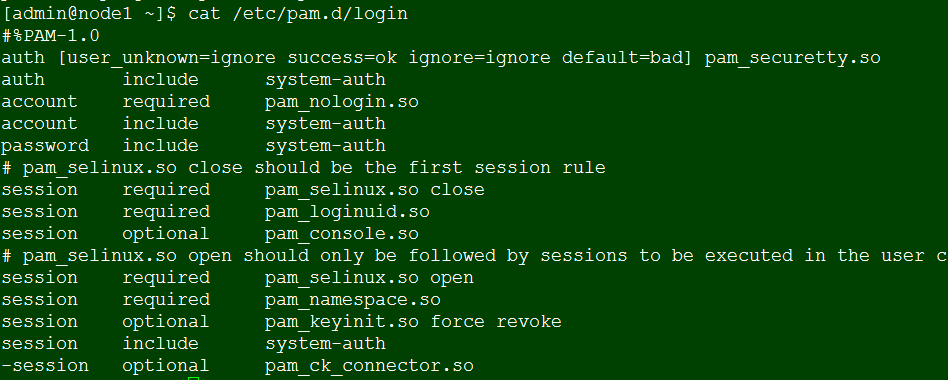
pam\_console.so：当系统出现某些问题，让使用者可以通过特殊终端接口顺利的登陆系统。

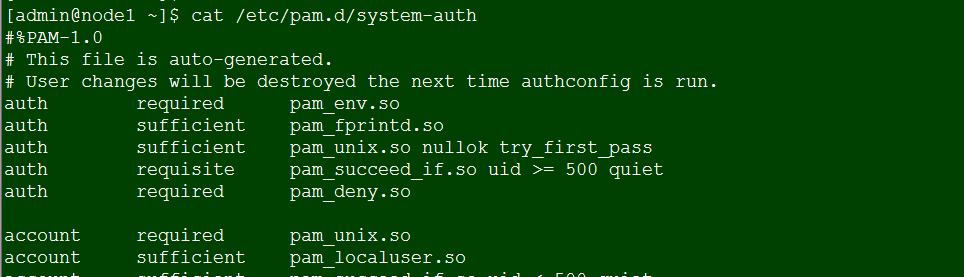
pam\_env.so：用来设置环境变量的1个模块。

pam\_unix.so：很复杂且重要的1个模块，可以用在验证阶段的认证功能，可以用在授权阶段的帐号授权管理，可以用在会议阶段的登录文件记录等。

pam\_pwquality.so：用来检验密码的强度。

以login为例，其PAM验证机制流程如下：





1）验证阶段

先通过pam\_securetty.so判断是否root。再通过pam\_env.so设置额外的环境变量。再通过pam\_unix.so检验密码。继续往下以pam\_succeed\_if.so判断UID是否大于1000。

2）授权阶段

先以pam\_nologin.so判断/etc/nologin是否存在，若存在则不许一般使用者登陆。再以pam\_unix.so及pam\_localuser.so进行帐号管理。再以pam\_succeed\_if.so判断UID是否小于1000，若小于1000则不记录登录信息。最后以pam\_permit.so允许该帐号登陆。

3）密码阶段

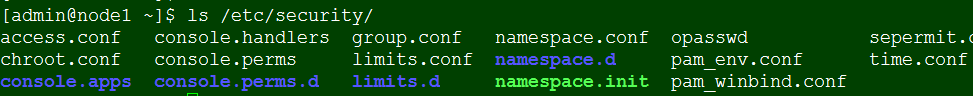
先以pam\_pwquality.so设置密码仅能尝试错误3次。再以pam\_unix.so通过sha512、shadow等功能进行密码检验。

4）会议阶段

先以pam\_selinux.so暂时关闭SELinux。再使用pam\_limits.so设置好使用者能够操作的系统资源。再以pam\_loginuid.so规范不同的UID权限等。

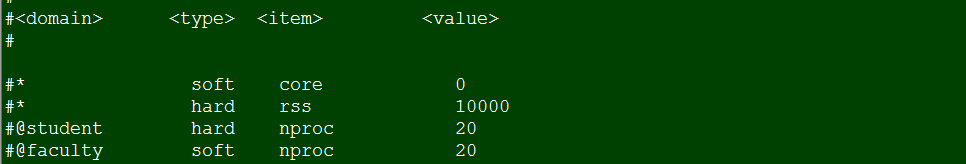
6.其他相关文件

其他PAM相关的文件主要在/etc/security目录下。



（1）limits.conf

对各个用户进行限制，包括一些文件大小、登录次数等。



（2）/var/log/secure、/var/log/messages

登录日志。

1. docker
   1. 基本概念

Docker系统包括服务端和客户端。其中，服务端是一个服务进程，管理着所有的容器。客户端则扮演着docker服务端的远程控制器，用来控制docker的服务端进程。通常服务端和客户端运行在一台机器上。

Docker包括三个基本概念：镜像、容器、仓库。

**1.镜像**

（1）操作系统可分为内核和用户空间。如Linux内核启动后，会挂载root文件系统为其提供用户空间支持。Docker镜像就相当于是一个root文件系统。如ubuntu官方镜像就包含了一套完整的ubuntu最小系统的root文件系统。

（2）Docker镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变（那是容器的事）。

（3）Docker镜像的文件系统使用分层存储架构设计。构建镜像时，会一层层构建。前一层是后一层的基础。每一层构建完就不会再发生改变。后一层上的任何改变只发生在自己这一层。如删除前一层文件的操作，实际不是真的删除前一层的文件，而是仅在当前层标记为该文件已删除。在最终容器运行的时候，虽然不会看到这个文件，但是实际上该文件会一直跟随镜像。

所以严格来说，镜像并非是像一个ISO那样的打包文件，而只是一个虚拟的概念。它并非由一个文件组成，而是由多层文件系统联合组成。

（4）分层存储的特征使得镜像的复用、定制变的更为容易。可以用之前构建好的镜像作为基础层，然后进一步添加新的层，以定制自己所需的内容，构建新的镜像。

（5）官网index.docker.io页面展示所有可用的镜像。镜像通常按“用户名/镜像名”存储。而对于如ubuntu这类基础镜像，经过官方验证值得信任，可以直接用镜像名来检索。

**2.容器**

（1）镜像和容器的关系，就像面向对象程序设计中的类和实例一样。镜像是静态的定义，容器是镜像运行时的实体。

（2）容器的实质是进程，可以被创建、启动、停止、删除、暂停等。与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的命名空间，是在一个隔离的环境里。容器可以拥有自己的root文件系统、网络配置、进程空间，用户ID空间等。

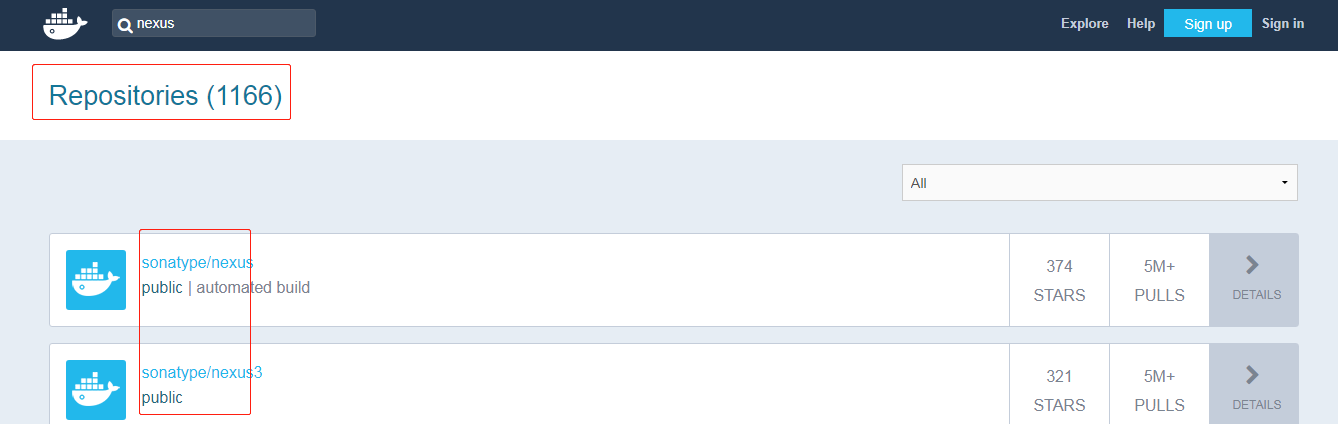
（3）容器也是采用分层存储设计。容器以镜像为基础层，在其上创建一个当前容器的存储层。容器存储层用于容器运行时的读写操作而准备。容器消亡时，容器存储层也随之消亡。因此，任何保存于容器存储层的信息都会随容器删除而丢失。

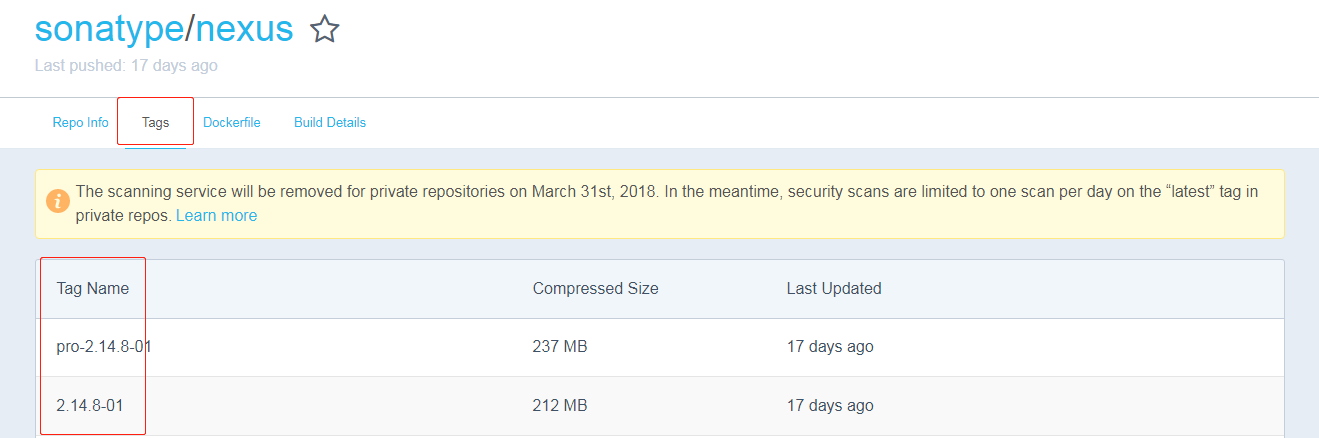
（4）按照Docker最佳实践的要求，容器不应该向其存储层内写入任何数据，容器存储层要保持无状态化。所有的文件写入操作，都应该使用数据卷或绑定宿主目录。

**3.Docker Registry**

（1）Docker Registry用于集中存储、分发镜像。

（2）一个Docker Registry中可以包含多个仓库。每个仓库可以包含多个标签。每个标签对应一个镜像。可以通过<仓库名>:<标签>的格式来指定具体是这个软件哪个版本的镜像。如果不给出标签，将以latest作为默认标签。





* 1. 基本使用

**1.仓库**

（1）配置阿里云国内加速器

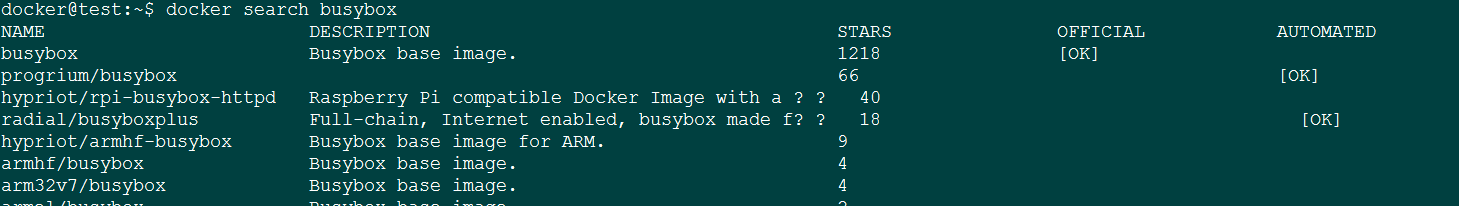
点击https://cr.console.aliyun.com/#/accelerator，会生成专属加速器地址。对于Docker Toolbox的用户，如下配置：



执行docker info，查看RegistryMirrors的值是否是自己配置的。

（2）可以直接在阿里云网站上搜索，也可以通过docker search搜索。





（3）使用login和logout命令登录、退出DockerHub。

**2.镜像**

（1）下载镜像



本地存储位置是/var/lib/docker（注意是虚拟机下的位置）。



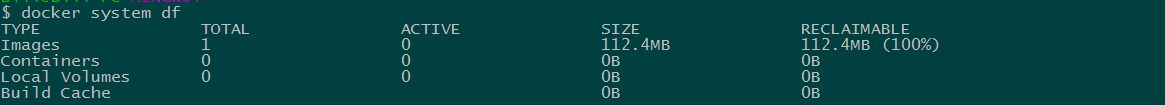
因镜像的多层存储结构，下载也是一层层的去下载。下载过程会给出每一层的ID的前12位。下载结束后，会给出该镜像完整的sha256的摘要，以确保下载一致性。



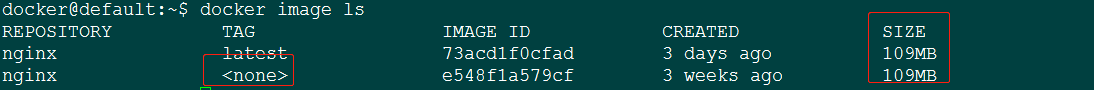
（2）查看本地镜像



因显示的本地镜像是解压后的大小，会比DockerHub上显示的体积小。因镜像的多层存储结构，镜像可能拥有共同的层，因此，列表中的镜像体积总和并非是所有镜像实际硬盘消耗。可以使用system df命令查看镜像、容器、数据卷所占用的空间。

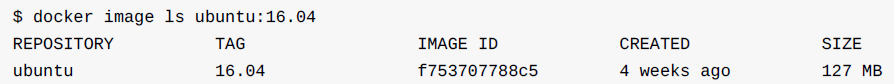


有的镜像的仓库名或标签可能为<none>。出现这种情况的原因是由于新旧镜像同名，旧镜像名称被取消导致，可以直接删除。

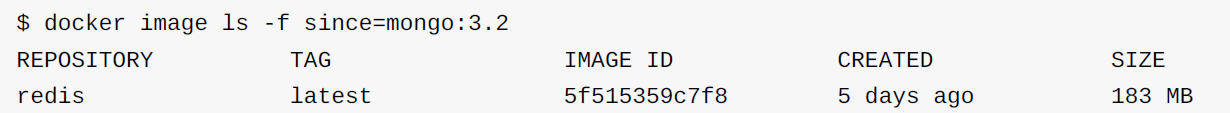


ls默认只显示顶层镜像，不显示中间层镜像（可重复利用的中间层，目的是加速镜像构建）。可以使用-a选项显示包括中间层镜像在内的全部镜像。

ls后加镜像名和标签，可列出特定镜像信息。



ls可结合--filter或-f过滤镜像显示，如使用since/before表示之后/之前。



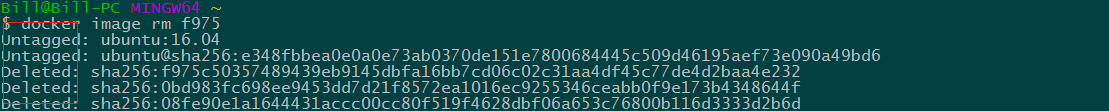
ls结合-q，可只显示镜像的ID列表。



（3）删除本地镜像



其中，<镜像>可以是镜像短/长ID、镜像名、镜像摘要。



删除行为分为两类：Untagged和Deleted。因为一个镜像可能对应多个标签（镜像的标签），如果还有别的标签指向了这个镜像，镜像并不会删除，只是取消该标签对镜像的指定。当该镜像所有的标签都被取消了，才会触发删除Deleted行为。

**3.容器**

（1）新建并启动



如果本地没有对应的镜像，会自动从远程下载。参数比较多，其中容器管理相关的如下：

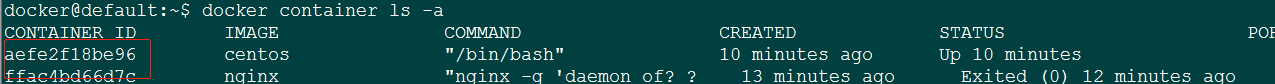
-d=true表示容器在后台运行，此时container不再监听执行docker run的这个终端命令行窗口，所有I/O只能通过网络或者共享卷组进行交互。

如果没有设置-d选项，则为前台运行，通常需要i和t选项结合使用。其中i让容器打开标准输入，t让docker给容器分配1个伪终端（有linux命令提示符）。

无论是前台还是后台运行，docker中必须要保持一个进程的运行，否则整个容器就会退出。因此通常后面让容器打开1个/bin/bash。



如果不指定名称，容器会被自动分配1个UUID名称，也可以使用--name参数指定。





（2）启动已停止容器

使用docker container start命令可将一个已经终止的容器启动运行。使用docker container restart重启容器。

（3）查看容器

使用container ls查看正在运行中的容器，加-a参数可以查看包括已终止的容器。

（4）终止容器

使用docker container stop终止一个运行中的容器。另外，如果容器中的应用终结时，容器也自动终止。如在容器绑定的终端输入exit，容器也会终止。

（5）删除容器

使用docker container rm删除一个处于终止状态的容器。使用docker container prune删除所有终止的容器。

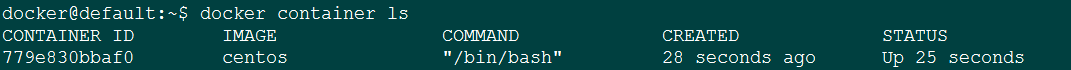
（6）导出和导入容器快照

使用docker export命令导出本地某个容器快照到本地。使用docker import从容器快照文件中再导入为镜像。

（7）进入容器

对于使用-d参数启动的在后台运行的容器，可以使用docker exec进入容器进行操作，之后即使exit容器，容器仍然还会继续运行。







* 1. 定制镜像
     1. 利用commit理解镜像构成

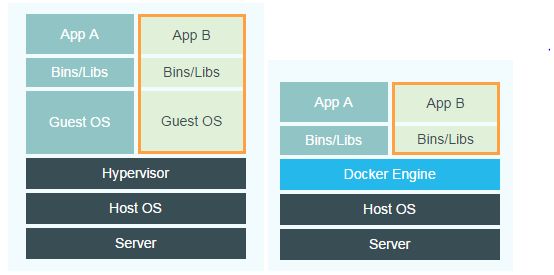
不要使用docker commit定制镜像，应该使用Dockerfile。但可以通过commit理解镜像构成，如定制1个Web服务器。



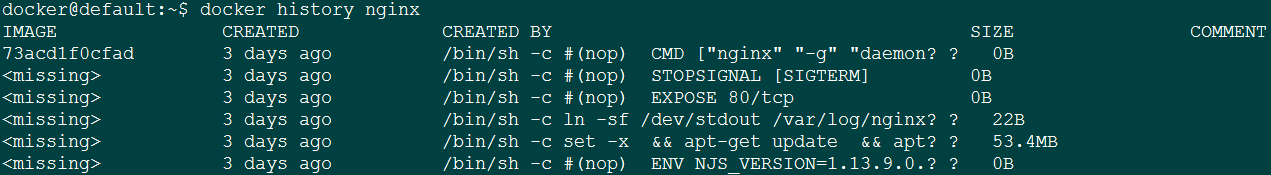
然后就可以进入容器。



之所以能进入容器，是因为nginx容器内也模拟出了1个操作系统，nginx服务就在这个模拟出的操作系统内。每个容器都是1个模拟出的操作系统。



可以通过docker history nginx查看容器构建历史。



此时，将如下内容写入nginx欢迎页。



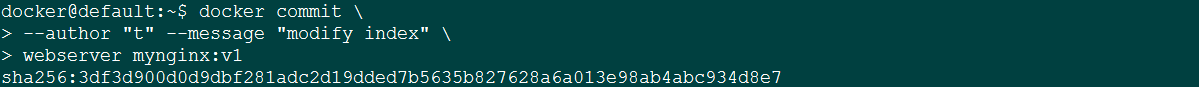


可以使用diff命令查看具体改动。

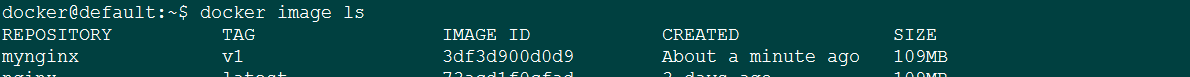


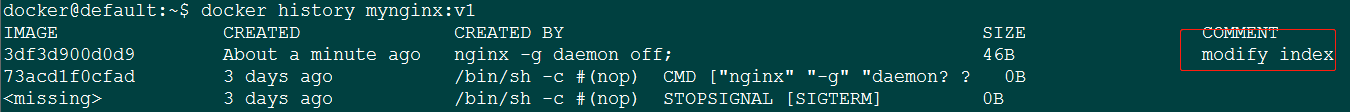
目前所做的任何文件修改都记录于容器存储层里，可使用commit命令，将容器的存储层保存下来成为镜像。





可再次使用docker history具体查看镜像的历史记录。





使用commit定制镜像存在如下严重问题：1）所有镜像的操作都是黑箱操作，生成的镜像也被称为黑箱镜像，很难了解执行过什么命令。2）没制作1次镜像，就更加臃肿，即使后面删除一些东西，因为分层，实际并没有删除。

* + 1. 使用Dockerfile定制镜像

**1.概述**

（1）文件结构

Dockerfile由一行行命令语句组成，分为四部分：基础镜像信息、维护者信息、镜像操作指令和容器启动时执行指令。Dockerfile支持以#开头的注释行。

（2）shell表示法和exec表示法

指令RUN、ENTRYPOINT、CMD都支持shell表示法和exec表示法。

使用shell表示法，命令行程序将作为sh程序的子程序运行，即，命令由/bin/sh -c调用，可以使用docker ps命令查看。使用shell表示法存在问题有：1）/bin/sh命令不会转发消息给实际运行的ping命令，不能安全得关闭docker容器。如，运行ping命令，即使按下trl-c也不能停止ping命令，因为ctrl-c的信号没有被转发给ping命令。2）如果容器没有安装shell，则无法运行。

exec表示法可以指定执行的命令和参数。

（3）最佳实践

通常每一个镜像只跑一个process，以便于方便应用横向扩张和重复利用容器。

**2.指令**

（1）FROM



指定基础镜像。如果为scratch，则表示一个空白的镜像。第一条指令必须是FROM，但要创建多个镜像时，FROM指令可以有多条。

（2）MAINTAINER

指定维护者信息。

（3）RUN



每条RUN指令将在当前镜像基础上执行指定命令，并提交为新的镜像。

RUN和后面的CMD和ENTRYPOINT不同，前者在Dockerfile构建镜像的过程中执行的命令，最终会被提交到镜像中。后者是启动容器（容器run或者start）时执行的命令。

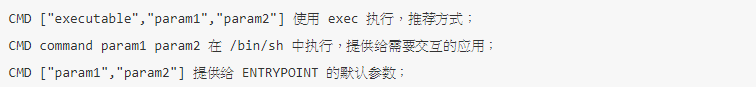
每一个RUN代表新建立一层，应该尽量减小层级，当命令较长时可以使用\来换行。

（4）ADD和COPY

均为拷贝命令。COPY用于复制本地主机的（为Dockerfile所在目录的相对路径）到容器中的。ADD还可以是一个URL，且拷贝tar文件时会自动解压。

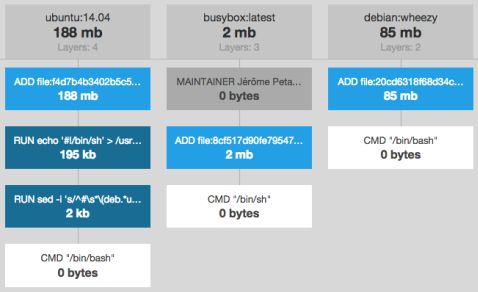
（5）CMD和ENTRYPOINT

CMD为启动容器时执行的命令。ENTRYPOINT为容器启动后执行的命令。如果有多条指令，均只执行最后1条。





从根本上说，ENTRYPOINT和CMD都是让用户指定一个可执行程序。这个可执行程序在container启动后自动启动。如果想让镜像自动运行程序（即，不需要在docker run后面添加指定运行的命令），则必须在Dockerfile里使用ENTRYPOINT或CMD命令。如下面3个Linux镜像都在Dockerfile的最后调用/bin/bash。



CMD和ENTRYPOINT的区别为：CMD命令可以被启动Docker容器时指定的参数覆盖，ENTRYPOINT命令可以将Docker容器启动时执行的参数当做自己的参数。如：





将会输出xianhu。





将输出echo xianhu。

可以组合使用ENTRYPOINT和CMD。ENTRYPOINT指定默认的运行命令，CMD指定默认的运行参数。

（6）EXPOSE



容器暴露的端口号，供互联系统使用。在启动容器时需要通过-P，Docker主机会自动分配一个端口转发到指定的端口。

（7）ENV



指定一个环境变量，会被后续 RUN 指令使用，并在容器运行时保持。

（8）USER



指定运行容器时的用户名或UID，后续的RUN也会使用指定用户。当服务不需要管理员权限时，可以通过该命令指定运行用户。并且可以在之前创建所需要的用户，例如：RUN groupadd -r postgres && useradd -r -g postgres postgres。要临时获取管理员权限可以使用gosu，而不推荐sudo。

（9）WORKDIR



为后续的 RUN、CMD、ENTRYPOINT 指令配置工作目录。

（10）VOLUME



创建一个可以从本地主机或其他容器挂载的挂载点，一般用来存放数据库和需要保持的数据等。

（11）ONBUILD

配置当所创建的镜像作为其它新创建镜像的基础镜像时，所执行的操作指令。

**3.创建镜像**



读取指定路径下（包括子目录）的Dockerfile，并将该路径下所有内容发送给Docker服务端，由服务端来创建镜像。可以通过.dockerignore文件忽略路径下的目录和文件。

docker分为服务端守护进程和客户端工具两部分。为了让服务端获得本地文件，就需要设置镜像构建上下文。如下：



相对的的是上下文的路径，都是相对路径，不能是绝对路径。因此，还需要.dockerignore来剔除不需要作为上下文传递给Docker引擎的文件。如果不额外指定Dockerfile的话，会将上下文目录下的名为Dockerfile的文件作为Dockerfile。

* 1. 数据管理

（1）数据卷是可供多个容器使用的特殊目录，可以在容器之间共享和重用。数据卷修改会立马生效，并默认会一直存在，即使容器被删除。

（2）创建数据卷



（3）查看数据卷的具体信息



web为容器名称（不是数据卷名称）。



（4）查看所有的数据卷



（4）删除数据卷



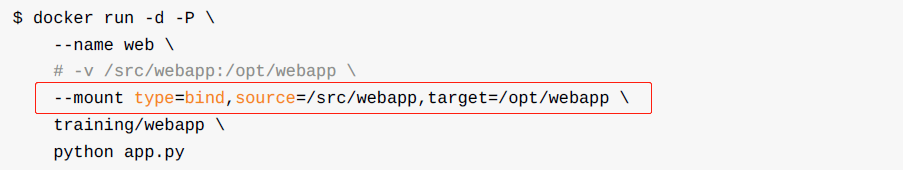
如果需要在删除容器的同时移除数据卷，可以使用docker rm -v命令。

清理无主的数据卷：



（5）挂载一个主机目录作为数据卷

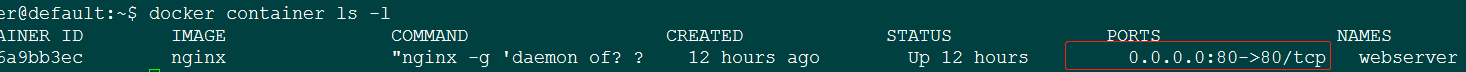
使用--mount标记可以指定挂载一个本地主机的目录到容器中去。本地目录的路径必须是绝对路径。挂载主机目录的默认权限是读写，用户也可以通过增加readonly指定为只读。



* 1. 网络管理

**1.端口映射**

（1）本地主机和容器开放的端口使用端口映射实现信息转发。如下，可以看到将本机80端口映射到容器80端口。



（2）使用-P选项，Docker会随机映射一个本机49000~49900的端口到内部容器开放的网络端口。使用-p可以指定要映射的端口，支持格式有：ip:hostPort:containerPort、ip::containerPort、hostPort:containerPort。即，还可以指定IP地址。可以用来绑定多个端口。

**2.容器互联**

（1）使用自定义网络实现容器互联。首先需要创建网络。



-d指定Docker网络类型，如bridge（其他不用了解）。

（2）在运行容器时，使用--network选项指定网络连接容器。

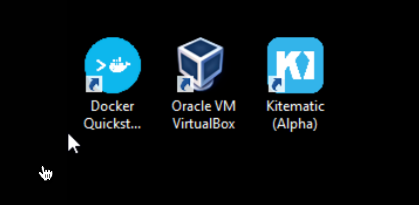


现在连入这个网络的容器已实现互联。

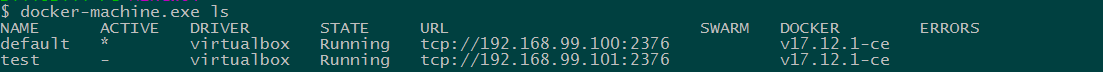
* 1. Docker Machine

**1.win7下安装**

因Docker引擎基于Linux内核（kernel），因此需要在Windows安装linux虚拟机来运行Docker。官方提供Docker Toolbox在win7下安装Docker。Toolbox主要集成了VirtualBox、git、Docker Client for Windows、Docker Machine，应该还有linux内核等。安装完成之后，有如下3个图标：



使用Docker Quickstart Termina可以快速启动1个Docker终端。也可以直接通过Docker Machine创建1个Docker主机。



**2.Docker Machine使用**

（1）创建、启动、关闭相关

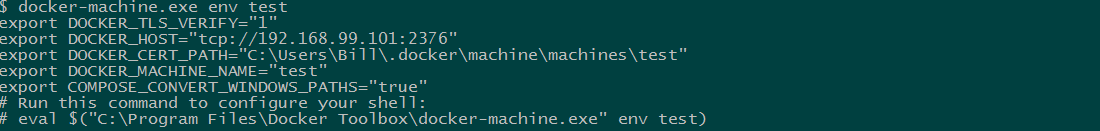
create创建一个Docker主机，创建时-d选项指定驱动，如virtualbox、vmwarefusion等；rm删除某台主机；upgrade更新主机Docker版本为最新。

start启动一个主机；restart：重启主机；stop停止一个主机；kill停止某个主机。

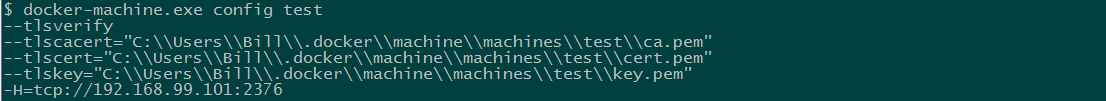
ls列出所有管理的主机；active查看活跃的Docker主机；status查看主机状态。

（2）操作相关

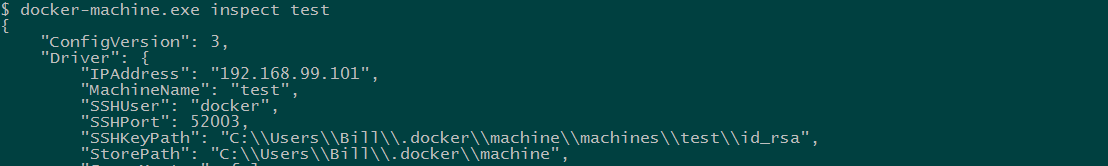
env显示连接到某个主机需要的环境变量。



config输出连接的配置信息。



inspect输出主机更多信息；provision重新设置一个已存在的主机。



url获取主机的URL；ip获取主机地址；ssh到主机上执行命令；scp在主机之间复制文件；regenerate-certs为某个主机重新生成TLS认证信息。



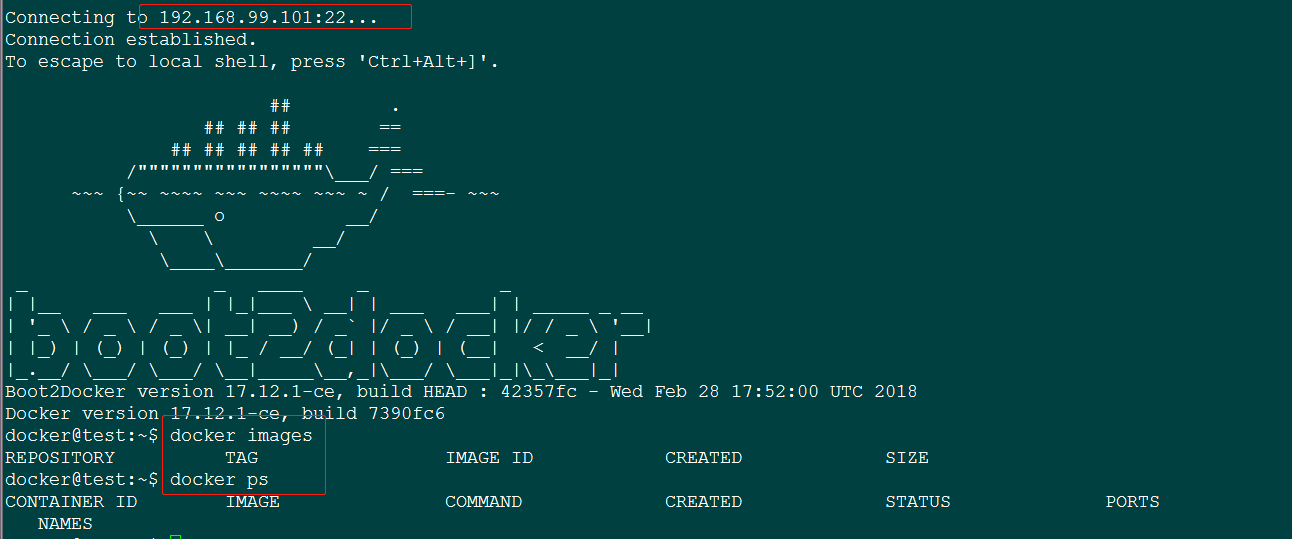
mount挂载主机目录到本地。

（3）其他

version输出docker-machine版本信息；help输出帮助信息。

**3.远程连接Docker主机**

可以使用ssh直接连接Docker主机，默认用户名和密码为docker/tcuser。



由此可见，win7系统的docker命令和docker主机内的命令一样。