# **[docker常用命令详解](https://www.cnblogs.com/duanxz/p/7905233.html)**

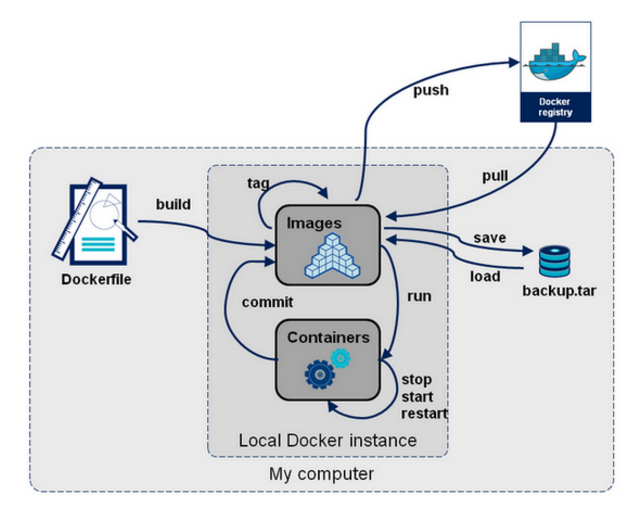
 docker logs   -t --since="2018-12-26 06:39:22" --until "2018-12-26 06:39:55" c-front

本文只记录docker命令在大部分情境下的使用，如果想了解每一个选项的细节，请参考官方文档，这里只作为自己以后的备忘记录下来。

根据自己的理解，总的来说分为以下几种：

* Docker环境信息 — docker [info|version]
* 容器生命周期管理 — docker [create|exec|run|start|stop|restart|kill|rm|pause|unpause]
* 容器操作运维 — docker [ps|inspect|top|attach|wait|export|port|rename|stat]
* 容器rootfs命令 — docker [commit|cp|diff]
* 镜像仓库 — docker [login|pull|push|search]
* 本地镜像管理 — docker [build|images|rmi|tag|save|import|load]
* 容器资源管理 — docker [volume|network]
* 系统日志信息 — docker [events|history|logs]

从docker命令使用出发，梳理出如下命令结构图：



### **-Docker环境信息**

### **--info命令**

用于检测Docker是否正确安装，一般结合docker version命令使用。

[IMG_257](https://www.cnblogs.com/duanxz/p/javascript:void(0);)

[devadmin@swarm1 ~]$ docker info

Containers: 43

Running: 9

Paused: 0

Stopped: 34

Images: 140

Server Version: 17.09.0-ce

Storage Driver: overlay

Backing Filesystem: xfs

Supports d\_type: false

Logging Driver: json-file

Cgroup Driver: cgroupfs

...

[IMG_258](https://www.cnblogs.com/duanxz/p/javascript:void(0);)

### **--version命令**

[IMG_259](https://www.cnblogs.com/duanxz/p/javascript:void(0);)

[devadmin@swarm1 ~]$ docker version

Client:

Version: 17.09.0-ce

API version: 1.32

Go version: go1.8.3

Git commit: afdb6d4

Built: Tue Sep 26 22:41:23 2017

OS/Arch: linux/amd64

Server:

Version: 17.09.0-ce

API version: 1.32 (minimum version 1.12)

Go version: go1.8.3

Git commit: afdb6d4

Built: Tue Sep 26 22:42:49 2017

OS/Arch: linux/amd64

Experimental: false

[devadmin@swarm1 ~]$

[IMG_260](https://www.cnblogs.com/duanxz/p/javascript:void(0);)

### **容器运维操作**

#### **--attach命令**

docker attach命令对应开发者很有用，可以连接到正在运行的容器，观察容器的运行状况，或与容器的主进程进行交互。

#### **--inspect命令**

用于查看镜像和容器的详细信息，默认会列出全部信息，可以通过--format参数来指定输出的模板格式，以便输出特定信息。

#### **查看容器的信息container（ps）**

docker ps命令可以查看容器的CONTAINER ID、NAME、IMAGE NAME、端口开启及绑定、容器启动后执行的COMMNAD。最常用的功能是通过ps来找到CONTAINER\_ID，以便对特定容器进行操作。  
docker ps 默认显示当前正在运行中的container  
docker ps -a 查看包括已经停止的所有容器  
docker ps -l 显示最新启动的一个容器（包括已停止的）

示例

[devadmin@swarm1 ~]$ docker ps -l

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1cfb8ad8642e regloc.com/test/settlequery:201711271708 "sh entrypoint.sh" 22 hours ago Up 22 hours 0.0.0.0:18182->18182/tcp settlequery.z4taavbos34np74za1bu674az.ncj2uocqbh15ljxhui0yccd9t

[devadmin@swarm1 ~]$

### **-- 列出机器上的镜像（images）**

# docker images REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

ubuntu 14.10 2185fd50e2ca 13 days ago 236.9 MB

…

其中我们可以根据REPOSITORY来判断这个镜像是来自哪个服务器，如果没有 / 则表示官方镜像，类似于username/repos\_name表示Github的个人公共库，类似于regsistory.example.com:5000/repos\_name则表示的是私服。  
IMAGE ID列其实是缩写，要显示完整则带上--no-trunc选项

### **2. 在docker index中搜索image（search）**

Usage: docker search TERM

# docker search seanloNAME DESCRIPTION STARS OFFICIAL AUTOMATED

seanloook/centos6 sean's docker repos 0

搜索的范围是官方镜像和所有个人公共镜像。NAME列的 / 后面是仓库的名字。

### **3. 从docker registry server 中下拉image或repository（pull）**

Usage: docker pull [OPTIONS] NAME[:TAG]

# docker pull centos

上面的命令需要注意，在docker v1.2版本以前，会下载官方镜像的centos仓库里的所有镜像，而从v.13开始官方文档里的说明变了：will pull the centos:latest image, its intermediate layers and any aliases of the same id，也就是只会下载tag为latest的镜像（以及同一images id的其他tag）。  
也可以明确指定具体的镜像：

# docker pull centos:centos6

当然也可以从某个人的公共仓库（包括自己是私人仓库）拉取，形如docker pull username/repository<:tag\_name> ：

# docker pull seanlook/centos:centos6

如果你没有网络，或者从其他私服获取镜像，形如docker pull registry.domain.com:5000/repos:<tag\_name>

# docker pull dl.dockerpool.com:5000/mongo:latest

### **4. 推送一个image或repository到registry（push）**

与上面的pull对应，可以推送到Docker Hub的Public、Private以及私服，但不能推送到Top Level Repository。

# docker push seanlook/mongo# docker push registry.tp-link.net:5000/mongo:2014-10-27

registry.tp-link.net也可以写成IP，172.29.88.222。  
在repository不存在的情况下，命令行下push上去的会为我们创建为私有库，然而通过浏览器创建的默认为公共库。

### **5. 从image启动一个container（run）**

docker run命令首先会从特定的image创之上create一层可写的container，然后通过start命令来启动它。停止的container可以重新启动并保留原来的修改。run命令启动参数有很多，以下是一些常规使用说明，更多部分请参考[http://www.cnphp6.com/archives/24899](http://www.cnphp6.com/archives/24899" \t "https://www.cnblogs.com/duanxz/p/_blank)  
当利用 docker run 来创建容器时，Docker 在后台运行的标准操作包括：

* 检查本地是否存在指定的镜像，不存在就从公有仓库下载
* 利用镜像创建并启动一个容器
* 分配一个文件系统，并在只读的镜像层外面挂载一层可读写层
* 从宿主主机配置的网桥接口中桥接一个虚拟接口到容器中去
* 从地址池配置一个 ip 地址给容器
* 执行用户指定的应用程序
* 执行完毕后容器被终止

Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

#### **5.1 使用image创建container并执行相应命令，然后停止**

# docker run ubuntu echo "hello world"hello word

这是最简单的方式，跟在本地直接执行echo 'hello world' 几乎感觉不出任何区别，而实际上它会从本地ubuntu:latest镜像启动到一个容器，并执行打印命令后退出（docker ps -l可查看）。需要注意的是，默认有一个--rm=true参数，即完成操作后停止容器并从文件系统移除。因为Docker的容器实在太轻量级了，很多时候用户都是随时删除和新创建容器。  
容器启动后会自动随机生成一个CONTAINER ID，这个ID在后面commit命令后可以变为IMAGE ID

#### **使用image创建container并进入交互模式, login shell是/bin/bash**

# docker run -i -t --name mytest centos:centos6 /bin/bash

bash-4.1#

上面的--name参数可以指定启动后的容器名字，如果不指定则docker会帮我们取一个名字。镜像centos:centos6也可以用IMAGE ID (68edf809afe7) 代替），并且会启动一个伪终端，但通过ps或top命令我们却只能看到一两个进程，因为容器的核心是所执行的应用程序，所需要的资源都是应用程序运行所必需的，除此之外，并没有其它的资源，可见Docker对资源的利用率极高。此时使用exit或Ctrl+D退出后，这个容器也就消失了（消失后的容器并没有完全删除？）  
（那么多个TAG不同而IMAGE ID相同的的镜像究竟会运行以哪一个TAG启动呢

#### **5.2 运行出一个container放到后台运行**

# docker run -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 2; done"

ae60c4b642058fefcc61ada85a610914bed9f5df0e2aa147100eab85cea785dc

它将直接把启动的container挂起放在后台运行（这才叫saas），并且会输出一个CONTAINER ID，通过docker ps可以看到这个容器的信息，可在container外面查看它的输出docker logs ae60c4b64205，也可以通过docker attach ae60c4b64205连接到这个正在运行的终端，此时在Ctrl+C退出container就消失了，按ctrl-p ctrl-q可以退出到宿主机，而保持container仍然在运行  
另外，如果-d启动但后面的命令执行完就结束了，如/bin/bash、echo test，则container做完该做的时候依然会终止。而且-d不能与--rm同时使用  
可以通过这种方式来运行memcached、apache等。

#### **5.3 映射host到container的端口和目录**

映射主机到容器的端口是很有用的，比如在container中运行memcached，端口为11211，运行容器的host可以连接container的 internel\_ip:11211 访问，如果有从其他主机访问memcached需求那就可以通过-p选项，形如-p <host\_port:contain\_port>，存在以下几种写法：

-p 11211:11211 这个即是默认情况下，绑定主机所有网卡（0.0.0.0）的11211端口到容器的11211端口上-p 127.0.0.1:11211:11211 只绑定localhost这个接口的11211端口-p 127.0.0.1::5000-p 127.0.0.1:80:8080

目录映射其实是“绑定挂载”host的路径到container的目录，这对于内外传送文件比较方便，在搭建私服那一节，为了避免私服container停止以后保存的images不被删除，就要把提交的images保存到挂载的主机目录下。使用比较简单，-v <host\_path:container\_path>，绑定多个目录时再加-v。

-v /tmp/docker:/tmp/docker

另外在两个container之间建立联系可用--link，详见高级部分或[官方文档](http://docs.docker.com/v1.1/reference/commandline/cli/" \l "run" \t "https://www.cnblogs.com/duanxz/p/_blank)。  
下面是一个例子：

# docker run --name nginx*\_test \> -v /tmp/docker:/usr/share/nginx/html:ro \> -p 80:80 -d \> nginx:1.7.6*

在主机的/tmp/docker下建立index.html，就可以通过http://localhost:80/或http://host-ip:80访问了。

### **6. 将一个container固化为一个新的image（commit）**

当我们在制作自己的镜像的时候，会在container中安装一些工具、修改配置，如果不做commit保存起来，那么container停止以后再启动，这些更改就消失了。  
docker commit <container> [repo:tag]  
后面的repo:tag可选  
只能提交正在运行的container，即通过docker ps可以看见的容器，

查看刚运行过的容器# docker ps -l

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

c9fdf26326c9 nginx:1 nginx -g.. 3 hours ago Exited (0).. nginx\_test

启动一个已存在的容器（run是从image新建容器后再启动），以下也可以使用docker start nginx\_test代替

[root@hostname docker]# docker start c9fdf26326c9

c9fdf26326c9

docker run -i -t --sig-proxy=false 21ffe545748baf /bin/bash

nginx服务没有启动

# docker commit -m "some tools installed" fcbd0a5348ca seanlook/ubuntu:14.10\_tutorial

fe022762070b09866eaab47bc943ccb796e53f3f416abf3f2327481b446a9503

-a "seanlook7@gmail.com"  
请注意，当你反复去commit一个容器的时候，每次都会得到一个新的IMAGE ID，假如后面的repository:tag没有变，通过docker images可以看到，之前提交的那份镜像的repository:tag就会变成<none>:<none>，所以尽量避免反复提交。  
另外，观察以下几点:

* commit container只会pause住容器，这是为了保证容器文件系统的一致性，但不会stop。如果你要对这个容器继续做其他修改：
  + 你可以重新提交得到新image2，删除次新的image1
  + 也可以关闭容器用新image1启动，继续修改，提交image2后删除image1
  + 当然这样会很痛苦，所以一般是采用Dockerfile来build得到最终image，参考[]
* 虽然产生了一个新的image，并且你可以看到大小有100MB，但从commit过程很快就可以知道实际上它并没有独立占用100MB的硬盘空间，而只是在旧镜像的基础上修改，它们共享大部分公共的“片”。

下

### **1. 开启/停止/重启container（start/stop/restart）**

容器可以通过run新建一个来运行，也可以重新start已经停止的container，但start不能够再指定容器启动时运行的指令，因为docker只能有一个前台进程。  
容器stop（或Ctrl+D）时，会在保存当前容器的状态之后退出，下次start时保有上次关闭时更改。而且每次进入attach进去的界面是一样的，与第一次run启动或commit提交的时刻相同。

CONTAINER\_ID=$(docker start <containner\_id>)

docker stop $CONTAINER\_ID

docker restart $CONTAINER\_ID

关于这几个命令可以通过一个完整的实例使用：[docker如何创建一个运行后台进程的容器并同时提供shell终端](http://segmentfault.com/blog/seanlook/1190000000755980" \t "https://www.cnblogs.com/duanxz/p/_blank)。

### **2. 连接到正在运行中的container（attach）**

要attach上去的容器必须正在运行，可以同时连接上同一个container来共享屏幕（与screen命令的attach类似）。  
官方文档中说attach后可以通过CTRL-C来detach，但实际上经过我的测试，如果container当前在运行bash，CTRL-C自然是当前行的输入，没有退出；如果container当前正在前台运行进程，如输出nginx的access.log日志，CTRL-C不仅会导致退出容器，而且还stop了。这不是我们想要的，detach的意思按理应该是脱离容器终端，但容器依然运行。好在attach是可以带上--sig-proxy=false来确保CTRL-D或CTRL-C不会关闭容器。

# docker attach --sig-proxy=false $CONTAINER\_ID

### **3. 查看image或container的底层信息（inspect）**

inspect的对象可以是image、运行中的container和停止的container。

查看容器的内部IP# docker inspect --format='{{.NetworkSettings.IPAddress}}' $CONTAINER\_ID172.17.42.35

### **4. 删除一个或多个container、image（rm、rmi）**

你可能在使用过程中会build或commit许多镜像，无用的镜像需要删除。但删除这些镜像是有一些条件的：

* 同一个IMAGE ID可能会有多个TAG（可能还在不同的仓库），首先你要根据这些 image names 来删除标签，当删除最后一个tag的时候就会自动删除镜像；
* 承上，如果要删除的多个IMAGE NAME在同一个REPOSITORY，可以通过docker rmi <image\_id>来同时删除剩下的TAG；若在不同Repo则还是需要手动逐个删除TAG；
* 还存在由这个镜像启动的container时（即便已经停止），也无法删除镜像；

TO-DO  
如何查看镜像与容器的依存关系

删除容器  
docker rm <container\_id/contaner\_name>

删除所有停止的容器

docker rm $(docker ps -a -q)

删除镜像  
docker rmi <image\_id/image\_name ...>  
下面是一个完整的示例：

# docker images <==

ubuntu 13.10 195eb90b5349 4 months ago 184.6 MB

ubuntu saucy 195eb90b5349 4 months ago 184.6 MB

seanlook/ubuntu rm\_test 195eb90b5349 4 months ago 184.6 MB

使用195eb90b5349启动、停止一个容器后，删除这个镜像# docker rmi 195eb90b5349Error response from daemon: Conflict, cannot delete image 195eb90b5349 because it is

tagged in multiple repositories, use -f to force2014/11/04 14:19:00 Error: failed to remove one or more images

删除seanlook仓库中的tag <==# docker rmi seanlook/ubuntu:rm\_testUntagged: seanlook/ubuntu:rm\_test

现在删除镜像，还会由于container的存在不能rmi# docker rmi 195eb90b5349Error response from daemon: Conflict, cannot delete 195eb90b5349 because the

container eef3648a6e77 is using it, use -f to force2014/11/04 14:24:15 Error: failed to remove one or more images

先删除由这个镜像启动的容器 <==# docker rm eef3648a6e77

删除镜像 <==# docker rmi 195eb90b5349Deleted: 195eb90b534950d334188c3627f860fbdf898e224d8a0a11ec54ff453175e081Deleted: 209ea56fda6dc2fb013e4d1e40cb678b2af91d1b54a71529f7df0bd867adc961Deleted: 0f4aac48388f5d65a725ccf8e7caada42f136026c566528a5ee9b02467dac90aDeleted: fae16849ebe23b48f2bedcc08aaabd45408c62b531ffd8d3088592043d5e7364Deleted: f127542f0b6191e99bb015b672f5cf48fa79d974784ac8090b11aeac184eaaff

注意，上面的删除过程我所举的例子比较特殊——镜像被tag在多个仓库，也有启动过的容器。按照<==指示的顺序进行即可。

### **5. docker build 使用此配置生成新的image**

build命令可以从Dockerfile和上下文来创建镜像：  
docker build [OPTIONS] PATH | URL | -  
上面的PATH或URL中的文件被称作上下文，build image的过程会先把这些文件传送到docker的服务端来进行的。  
如果PATH直接就是一个单独的Dockerfile文件则可以不需要上下文；如果URL是一个Git仓库地址，那么创建image的过程中会自动git clone一份到本机的临时目录，它就成为了本次build的上下文。无论指定的PATH是什么，Dockerfile是至关重要的，请参考[Dockerfile Reference](http://docs.docker.com/reference/builder/" \t "https://www.cnblogs.com/duanxz/p/_blank)。  
请看下面的例子：

# cat Dockerfile

FROM seanlook/nginx:bash\_vim

EXPOSE 80

ENTRYPOINT /usr/sbin/nginx -c /etc/nginx/nginx.conf && /bin/bash

# docker build -t seanlook/nginx:bash\_vim\_Df .

Sending build context to Docker daemon 73.45 MB

Sending build context to Docker daemon

Step 0 : FROM seanlook/nginx:bash\_vim

---> aa8516fa0bb7

Step 1 : EXPOSE 80

---> Using cache

---> fece07e2b515

Step 2 : ENTRYPOINT /usr/sbin/nginx -c /etc/nginx/nginx.conf && /bin/bash

---> Running in e08963fd5afb

---> d9bbd13f5066

Removing intermediate container e08963fd5afb

Successfully built d9bbd13f5066

上面的PATH为.，所以在当前目录下的所有文件（不包括.dockerignore中的）将会被tar打包并传送到docker daemon（一般在本机），从输出我们可以到Sending build context...，最后有个Removing intermediate container的过程，可以通过--rm=false来保留容器。  
TO-DO  
docker build github.com/creack/docker-firefox失败。

### **6. 给镜像打上标签（tag）**

tag的作用主要有两点：一是为镜像起一个容易理解的名字，二是可以通过docker tag来重新指定镜像的仓库，这样在push时自动提交到仓库。

将同一IMAGE\_ID的所有tag，合并为一个新的# docker tag 195eb90b5349 seanlook/ubuntu:rm\_test

新建一个tag，保留旧的那条记录# docker tag Registry/Repos:Tag New\_Registry/New\_Repos:New\_Tag

### **7. 查看容器的信息container（ps）**

docker ps命令可以查看容器的CONTAINER ID、NAME、IMAGE NAME、端口开启及绑定、容器启动后执行的COMMNAD。经常通过ps来找到CONTAINER\_ID。  
docker ps 默认显示当前正在运行中的container  
docker ps -a 查看包括已经停止的所有容器  
docker ps -l 显示最新启动的一个容器（包括已停止的）

### **8. 查看容器中正在运行的进程（top）**

容器运行时不一定有/bin/bash终端来交互执行top命令，查看container中正在运行的进程，况且还不一定有top命令，这是docker top <container\_id/container\_name>就很有用了。实际上在host上使用ps -ef|grep docker也可以看到一组类似的进程信息，把container里的进程看成是host上启动docker的子进程就对了。

### **9. 其他命令**

docker还有一些如login、cp、logs、export、import、load、kill等不是很常用的命令，比较简单，请参考官网。

### **events、history和logs命令**

这3个命令用于查看Docker的系统日志信息。events命令会打印出实时的系统事件；history命令会打印出指定镜像的历史版本信息，即构建该镜像的每一层镜像的命令记录；logs命令会打印出容器中进程的运行日志。

docker events [options] ：从服务器获取实时事件。

OPTIONS说明：

**-f ：**根据条件过滤事件；

**--since ：**从指定的时间戳后显示所有事件;

**--until ：**流水时间显示到指定的时间为止；

docker history [options] image：查看指定镜像的创建历史。

OPTIONS说明：

**-H :**以可读的格式打印镜像大小和日期，默认为true；

**--no-trunc :**显示完整的提交记录；

**-q :**仅列出提交记录ID。

docker logs [options] container

Options:

--details 显示更多的信息

-f, --follow 跟踪日志输出，最后一行为当前时间戳的日志

--since string 显示自具体某个时间或时间段的日志

--tail string 从日志末尾显示多少行日志， 默认是all

-t, --timestamps 显示时间戳

分类: [operation](https://www.cnblogs.com/duanxz/category/762508.html" \t "https://www.cnblogs.com/duanxz/p/_blank), [docker](https://www.cnblogs.com/duanxz/category/1346307.html" \t "https://www.cnblogs.com/duanxz/p/_blank)

标签: [docker](https://www.cnblogs.com/duanxz/tag/docker/), [operation](https://www.cnblogs.com/duanxz/tag/operation/)