🡪--help显示帮助信息。例如docker image –help

🡪Docker CLI命令分为模块化命令以及一般快速命令。随着docker的发展，Docker将命令放入模块中例如docker image（对应image模块）, docker swarm（对应swarm模块）, docker network（对应network模块）等。另外还保留一些快速命令，例如docker start, docker ps等。

🡪docker version # 显示docker版本信息

docker info # 显示有多少container在运行，本地有多少image, 是否在docker swam下运行。

docker port [container名字] [port号码] # 显示指定container的port口信息

docker ps # 显示正在运行的container, -a 显示所有container信息。包括container id, 根据image名字，状态, container名字

-a 显示所有docker

-q 只显示数字ID

docker container # docker container命令模块

docker container list # 展示所有container与docker ps 等效

docker container prune # 删除所有停止的容器

docker top [container名字] # 展示container进程信息，例如PID (Process ID), PPID （ Parent Process ID）

docker image # docker的image模块

docker image list # 展示所有本地image模块

docker stop [container名] # 停止一个container的运行（给container进程发送结束信号，让container在停止工作前进行退出准备工作再退出）

docker stop $(docker ps -aq) #停止所有container运行

docker kill [container名] # 杀死一个container进程 （直接杀死container进程，不进行优雅退出）

docker rm [container名] # 删除一个指定的container

docker rm $(docker ps -aq) # 删除所有的容器

docker rmi [image名] # 删除本地的image

docker rmi $(docker images -q) # 删除本地所有image

docker login # 登录到registry

docker pull [image名字]:[tag名字] # 从image registry中取下image,并且保存到本地

docker push [image名字] # 将image推送到远端registry

docker commit # 将改动的image提交到本地的image库

docker create [OPTIONS] IMAGE [COMMAND] [ARG...] # 创建一个新的容器但不启动它

参数用法同docker run

Docker start # 启动一个container

docker run [OPTIONS] IMAGE [COMMAND] [ARG...] # 从image中创建一个新的容器使其运行并可选的运行一个命令

--name 命名该container

-d 后台运行, 默认在前台运行

-i 交互模式，保持打开container的标准输入流

-t 分配一个伪终端到container的标准输入流

-it 以交互前台模式运行container. 例如我们通过docker run -it ubuntu bash启动一个ubuntu,那么可以直接使用ubuntu 镜像中的bash.

--rm退出容器以后，这个容器就被删除. 常用于—it启动的container.

-p <host-port>:<container-port> 映射主机端口到container端口中。 -p 6379:6379映射主机6379到container的6379中。如果只有一个参数-p 6379则表示动态随机映射主机某端口到container 6379端口中，具体主机哪个端口可以使用docker ps查看。

-v <host-dir>:<container-dir> 将主机中目录与container中目录挂载。主机目录必须为绝对路径。

<container-dir> 在container中创建一个数据挂载卷

-e 设置将要运行的container的环境变量

--volumes-from [container名] 将指定container的数据挂载卷放入本container中

--network 将启动的container连接到某个network上

--link [需要连接的目的容器名: 目的容器在本容器的别名]

-h: 指定容器的hostname

docker exec [container名字] [命令] # 在指定名称的container中运行指定命令

-t 为container分配一个伪终端

-i 交互模式

例如：

docker exec -it mynginx /bin/sh /root/runoob.sh # 对mynginx container以bash形式运行一个bash脚本

docker exec -it mynginx /bin/bash # 对mynginx container开启一个bash终端

docker exec -i r1 redis-cli # 对名为r1的container执行redis-cli命令

docker stats [container名字] # 展示container资源使用情况

docker save [image名字] > [tar文件名字] # 将指定image存为某tar file.

docker search [image名字] # 在docker hub中找寻image

docker inspect [container名字] # 返回container的底层信息

docker log [container名字] # 返回docker的日志

docker build [Dockerfile 所在路径] # 从dockerfile中创建一个本地image

-t 为新创建的image加上标签, [name:tag]格式

docker cp: 用于容器与主机之间的数据拷贝。可以从宿主主机拷贝到container, 也可以从container拷贝到宿主主机中。

例如：docker cp /www/runoob 96f7f14e99ab:/www/ 将主机/www/runoob目录拷贝到容器96f7f14e99ab的/www目录下。

docker export: 将container中文件系统作为一个tar归档文件导出到STDOUT。

例如：docker export -o mysql-`date +%Y%m%d`.tar a404c6c174a2

docker import: 从归档文件中创建镜像。

例如： docker import my\_ubuntu\_v3.tar runoob/ubuntu: v4

docker network: docker的network模块

create # 创建一个network

connect # 将一个container加入某个network

disconnect # 将一个container移出某个network

ls # 展示所有network

inspect # 检视某个network, 显示attach上这个network的container

docker logs [container名字] # 获取指定container的标准输出和标准错误输出

docker service # 使用docker swarm集群来启动服务

docker system: docker的system模块

df # 查询镜像（Images）、容器（Containers）和本地卷（Local Volumes）等空间使用大户的空间占用情况。

prune # 清除已停止的容器(container)，未被任何容器所使用的卷(volume), 未被任何容器所关联的网络(network), 所有悬空镜像(image)

🡪 docker run -v /docker/redis-data:/data --name r1 -d redis redis-server --appendonly yes

从image redis中启动一个名为r1的container,并且将宿主主机的/docker/redis-data和container的/data挂载。后面的redis-server –appendonly yes是启动container后执行的命令。redis-server不是image名，只是redis数据库服务器的命令。

🡪 docker run -v /docker/redis-data:/data:ro -it ubuntu

从image ubuntu中启动一个container，挂载宿主主机/docker/redis-data到container中/data中，并且container对该目录只有只读权限。