1. **docker container run alpine sh**
   This command will pull an alpine image, run a container out of it and then run *sh* command to run shell inside the container.

2. **docker container run -it alpine sh**
   This command will do the same job as the above command, but will run the container in an interactive mode.

3. **docker container ls**
   Get a list of running containers.

4. **docker container ls -a**
   Get a list of all containers whether running or not.

5. **docker container run -it alpine sh; hostname; Ctrl + P & Ctrl + Q; docker container attach <hostname>**
   It runs the container in an interactive mode. Then note down the hostname/Container ID of the container. Instead of terminating the container, now we will detach from the interactive mode. After detaching, we can attach to the container later by using hostname.

6. **docker container start <hostname>**
   Use this command to start a container.

7. **docker container stop <hostname>**
   Use this command to stop a container.

8. **docker container run -itd alpine sh**
   Use this command to run a container in detached mode.

9. **docker container rm <Container ID>**
   Use this command to remove a container.

10. **docker container ls -aq**
    List container ID(s) of all the containers.

11. **docker container rm $(docker container ls -aq)**
    Remove all the containers.

12. **docker image ls**
    List all the images.

13. **docker image rm <image name>**
    Use this command to remove an image.
14. **docker container run --rm hello-world**
    The *--rm* flag will automatically delete the container upon its exit.

15. docker container run --name my_container_1 hello-world

Docker automatically assigns names to containers. To override this behaviour, you can pass your own name with *--name* flag.

16. docker container run -p 80:80 nginx

-p flag is used to create a port mapping. The first port is the local port to which we want to forward in the local machine, and the second port is the port running inside the container.

17. docker container run -d alpine

-d flag is used to run a container in detached mode.

Containers are isolated from the host operating system, and from one another by the help of kernel namespaces. Docker uses *net*, *pid*, and *mnt (mount)* namespaces to isolate containers.

18. docker container run -p 8088:80 -v /home/dhruv/html:/usr/share/nginx/html nginx

We can use *-v/--volume* flag to mount a local volume to the volume inside the container. The first volume is the volume present in the local machine, and the second volume is the volume inside the container to which we want to mount the local volume.

19. docker container run -it --name c2 --link c1 alpine sh

This will link the container *c2* to an already running container *c1*, and will allow *c2* to communicate to *c1* by using just the container name i.e. *c1*. For example, now you can use command *ping c1* to ping container c1 from container *c2*. Alternately, you can also use the IP address of the container *c1*.

Links are deprecated. There are now modern ways that also enable communication between containers.

20. docker network ls

List all the networks available. By default, all the containers are linked to the *bridge* network.

21. docker network create test

Use this command to create a new network. *Test* here refers to the new network.

22. docker container run -it --rm --name c1 --network test alpine sh

Use *--network* flag to attach a container to a user-defined network. User-defined networks offer name resolution for free and, thus we need not to use *--link* flag.

Docker client talks to the docker daemon via REST API.
docker run is an alias for *docker container run*.

23. docker -help

List all the management commands and legacy commands. Each of the command in management commands represent a resource such as image, container, network, etc.

24. export DOCKER_HIDE_LEGACY_COMMANDS=false
Use this to hide the legacy commands displayed when we type *docker --help*.

Docker images not only contain the application itself but also, all the dependencies required to run the application.
Tag in an image, specify the version of the image. If no tag is provided, then docker automatically takes the tag as *latest*.

25. docker image rm alpine:latest
Removes the image *alpine* with the tag *latest*.

26. docker image pull alpine:latest
Use this command to download an image *alpine* with tag *latest*.

27. docker login --username [username]
Use this command to login docker CLI. It will ask for an access token. Go here, create an access token and use this in place of password.

FROM instruction inside *Dockerfile* is used to base our image on an already existing image.
CMD instruction inside *Dockerfile* is used to set the default command on startup for our container.

28. docker image build -t myalpine:latest .
Use this command to build an image from a Dockerfile. Here *myalpine* is the name of the image that you want to give and *latest* is the tag that you want to attach with this image. And then use *sudo docker run -it myalpine:latest* to run the container.

Whenever you want to use a comment, use # in the beginning of the line.

29. docker image tag myalpine:latest dhruvawasthi/myalpine:latest; docker image push dhruvawasthi/myalpine:latest
To push an image to the docker hub, you first need to tag (rename) the image as alpine and nginx are official images and they live in the root namespace. Our image is not an official image, so it cannot live in root namespace. But we get our own namespace, and we can use it to push the image.

30. docker plugin install store/weaveworks/net-plugin:2.5.2
Use this command to install a plugin that you can find on docker hub here.

31. docker plugin ls
Use this command to list all the installed plugins.

You can use the *docker plugin* command to manage plugins.

A registry is a storage and content delivery system, holding named Docker images, available in different tagged versions. Till now we were using *docker hub* registry. But we can also use a third party registry to pull and push images. For demonstration here, we will use quay. Quay is a container image registry that enables you to build, organize, distribute, and deploy containers.

32. docker login quay.io; docker image tag dhruvawasthi/myalpine:latest quay.io/dhruvawasthi/myalpine:latest; docker image push quay.io/dhruvawasthi/myalpine:latest

Use this command to first login to quay.io. Then you can push the image, after tagging it properly. This time, you also need to pass the domain of the repository.

33. docker image pull quay.io/dhruvawasthi/myalpine:latest

Use this command to pull an image from the registry.

34. docker container kill <Container-Name/Container-ID>

Use this command to kill a docker container.

AUTHOR:
Dhruv Awasthi
Email: dhruvawasthicc@gmail.com
Github: https://github.com/DhruvAwasthi
Linkedin: https://www.linkedin.com/in/dhruv-awasthi/