Command Line

> ***Note:*** *if you are using a remote Docker daemon, such as Boot2Docker, then do not type the* `sudo` *before the* `docker` *commands shown in the documentation's examples.*

To list available commands, either run `docker` with no parameters or execute `docker help`:

```
$ sudo docker
  Usage: docker [OPTIONS] COMMAND [arg...]
    -H, --host=[]: The socket(s) to bind to in daemon mode, specified using one or more tcp://host:port, unix:///path/to/so
cket, fd://* or fd://socketfd.

  A self-sufficient runtime for Linux containers.

  ...
```

Option types

Single character commandline options can be combined, so rather than typing `docker run -t -i --name test busybox sh`, you can write `docker run -ti --name test busybox sh`.

Boolean

Boolean options look like `-d=false`. The value you see is the default value which gets set if you do **not** use the boolean flag. If you do call `run -d`, that sets the opposite boolean value, so in this case, `true`, and so `docker run -d` **will** run in "detached" mode, in the background. Other boolean options are similar – specifying them will set the value to the opposite of the default value.

Multi

Options like `-a=[]` indicate they can be specified multiple times:

```
$ sudo docker run -a stdin -a stdout -a stderr -i -t ubuntu /bin/bash
```

Sometimes this can use a more complex value string, as for `-v`:

```
$ sudo docker run -v /host:/container example/mysql
```

Strings and Integers

Options like `--name=""` expect a string, and they can only be specified once. Options like `-c=0` expect an integer, and they can only be specified once.

daemon

```
Usage: docker [OPTIONS] COMMAND [arg...]

A self-sufficient runtime for linux containers.

Options:
  --api-enable-cors=false                    Enable CORS headers in the remote API
  -b, --bridge=""                            Attach containers to a pre-existing network bridge
                                               use 'none' to disable container networking
  --bip=""                                   Use this CIDR notation address for the network bridge's IP, not compatible wit
h -b
  -D, --debug=false                          Enable debug mode
  -d, --daemon=false                         Enable daemon mode
  --dns=[]                                   Force Docker to use specific DNS servers
  --dns-search=[]                            Force Docker to use specific DNS search domains
  -e, --exec-driver="native"                 Force the Docker runtime to use a specific exec driver
  --fixed-cidr=""                            IPv4 subnet for fixed IPs (ex: 10.20.0.0/16)
                                               this subnet must be nested in the bridge subnet (which is defined by -b or -
-bip)
  -G, --group="docker"                       Group to assign the unix socket specified by -H when running in daemon mode
                                               use '' (the empty string) to disable setting of a group
  -g, --graph="/var/lib/docker"              Path to use as the root of the Docker runtime
  -H, --host=[]                              The socket(s) to bind to in daemon mode or connect to in client mode, specifie
d using one or more tcp://host:port, unix:///path/to/socket, fd://* or fd://socketfd.
  --icc=true                                 Allow unrestricted inter-container and Docker daemon host communication
  --insecure-registry=[]                     Enable insecure communication with specified registries (disables certificate
verification for HTTPS and enables HTTP fallback) (e.g., localhost:5000 or 10.20.0.0/16)
  --ip=0.0.0.0                               Default IP address to use when binding container ports
  --ip-forward=true                          Enable net.ipv4.ip_forward
  --ip-masq=true                             Enable IP masquerading for bridge's IP range
  --iptables=true                            Enable Docker's addition of iptables rules
   -l, --log-level="info"                    Set the logging level
  --label=[]                                 Set key=value labels to the daemon (displayed in `docker info`)
  --mtu=0                                    Set the containers network MTU
                                               if no value is provided: default to the default route MTU or 1500 if no defa
ult route is available
  -p, --pidfile="/var/run/docker.pid"        Path to use for daemon PID file
  --registry-mirror=[]                       Specify a preferred Docker registry mirror
  -s, --storage-driver=""                    Force the Docker runtime to use a specific storage driver
  --selinux-enabled=false                    Enable selinux support. SELinux does not presently support the BTRFS storage d
river
  --storage-opt=[]                           Set storage driver options
  --tls=false                                Use TLS; implied by --tlsverify flag
  --tlscacert="/home/sven/.docker/ca.pem"    Trust only remotes providing a certificate signed by the CA given here
  --tlscert="/home/sven/.docker/cert.pem"    Path to TLS certificate file
  --tlskey="/home/sven/.docker/key.pem"      Path to TLS key file
  --tlsverify=false                          Use TLS and verify the remote (daemon: verify client, client: verify daemon)
  -v, --version=false                        Print version information and quit
```

Options with [] may be specified multiple times.

The Docker daemon is the persistent process that manages containers. Docker uses the same binary for both the daemon and client. To run the daemon you provide the `-d` flag.

To run the daemon with debug output, use `docker -d -D`.

Daemon socket option

The Docker daemon can listen for Docker Remote API (/reference/api/docker_remote_api/) requests via three different types of Socket: `unix`, `tcp`, and `fd`.

By default, a `unix` domain socket (or IPC socket) is created at `/var/run/docker.sock`, requiring either `root` permission, or `docker` group membership.

If you need to access the Docker daemon remotely, you need to enable the `tcp` Socket. Beware that the default setup provides un-encrypted and un-authenticated direct access to the Docker daemon - and should be secured either using the built in HTTPS encrypted socket (/articles/https/), or by putting a secure web proxy in front of it. You can listen on port `2375` on all network interfaces with `-H tcp://0.0.0.0:2375`, or on a particular network interface using its IP address: `-H tcp://192.168.59.103:2375`. It is conventional to use port `2375` for un-encrypted, and port `2376` for encrypted communication with the daemon.

> ***Note*** *If you're using an HTTPS encrypted socket, keep in mind that only TLS1.0 and greater are supported. Protocols SSLv3 and under are not supported anymore for security reasons.*

On Systemd based systems, you can communicate with the daemon via systemd socket activation (http://0pointer.de/blog/projects/socket-activation.html), use `docker -d -H fd://`. Using `fd://` will work perfectly for most setups but you can also specify individual sockets: `docker -d -H fd://3`. If the specified socket activated files aren't found, then Docker will exit. You can find examples of using Systemd socket activation with Docker and Systemd in the Docker source tree (https://github.com/docker/docker/tree/master/contrib/init/systemd/).

You can configure the Docker daemon to listen to multiple sockets at the same time using multiple `-H` options:

```
# listen using the default unix socket, and on 2 specific IP addresses on this host.
docker -d -H unix:///var/run/docker.sock -H tcp://192.168.59.106 -H tcp://10.10.10.2
```

The Docker client will honor the `DOCKER_HOST` environment variable to set the `-H` flag for the client.

```
$ sudo docker -H tcp://0.0.0.0:2375 ps
# or
$ export DOCKER_HOST="tcp://0.0.0.0:2375"
$ sudo docker ps
# both are equal
```

Setting the `DOCKER_TLS_VERIFY` environment variable to any value other than the empty string is equivalent to setting the `--tlsverify` flag. The following are equivalent:

```
$ sudo docker --tlsverify ps
# or
$ export DOCKER_TLS_VERIFY=1
$ sudo docker ps
```

Daemon storage-driver option

The Docker daemon has support for several different image layer storage drivers: `aufs`, `devicemapper`, `btrfs` and `overlay`.

The `aufs` driver is the oldest, but is based on a Linux kernel patch-set that is unlikely to be merged into the main kernel. These are also known to cause some serious kernel crashes. However, `aufs` is also the only storage driver that allows containers to share executable and shared library memory, so is a useful choice when running thousands of containers with the same program or libraries.

The `devicemapper` driver uses thin provisioning and Copy on Write (CoW) snapshots. For each devicemapper graph location – typically `/var/lib/docker/devicemapper` – a thin pool is created based on two block devices, one for data and one for metadata. By default, these block devices are created automatically by using loopback mounts of automatically created sparse files. Refer to Storage driver options below for a way how to customize this setup. ~jpetazzo/Resizing Docker containers with the Device Mapper plugin (http://jpetazzo.github.io/2014/01/29/docker-device-mapper-resize/) article explains how to tune your existing setup without the use of options.

The `btrfs` driver is very fast for `docker build` - but like `devicemapper` does not share executable memory between devices. Use `docker -d -s btrfs -g /mnt/btrfs_partition`.

The `overlay` is a very fast union filesystem. It is now merged in the main Linux kernel as of 3.18.0 (https://lkml.org/lkml/2014/10/26/137). Call `docker -d -s overlay` to use it.

> ***Note:*** *It is currently unsupported on `btrfs` or any Copy on Write filesystem and should only be used over `ext4` partitions.*

Storage driver options

Particular storage-driver can be configured with options specified with `--storage-opt` flags. The only driver accepting options is `devicemapper` as of now. All its options are prefixed with `dm`.

Currently supported options are:

- `dm.basesize`

  Specifies the size to use when creating the base device, which limits the size of images and containers. The default value is 10G. Note, thin devices are inherently "sparse", so a 10G device which is mostly empty doesn't use 10 GB of space on the pool. However, the filesystem will use more space for the empty case the larger the device is.

  Warning: This value affects the system-wide "base" empty filesystem that may already be initialized and inherited by pulled images. Typically, a change to this value will require additional steps to take effect:

  ```
  $ sudo service docker stop
  $ sudo rm -rf /var/lib/docker
  $ sudo service docker start
  ```

Example use:

```
$ sudo docker -d --storage-opt dm.basesize=20G
```

- `dm.loopdatasize`

  Specifies the size to use when creating the loopback file for the "data" device which is used for the thin pool. The default size is 100G. Note that the file is sparse, so it will not initially take up this much space.

  Example use:

  ```
  $ sudo docker -d --storage-opt dm.loopdatasize=200G
  ```

- `dm.loopmetadatasize`

  Specifies the size to use when creating the loopback file for the "metadata" device which is used for the thin pool. The default size is 2G. Note that the file is sparse, so it will not initially take up this much space.

  Example use:

  ```
  $ sudo docker -d --storage-opt dm.loopmetadatasize=4G
  ```

- `dm.fs`

  Specifies the filesystem type to use for the base device. The supported options are "ext4" and "xfs". The default is "ext4"

  Example use:

  ```
  $ sudo docker -d --storage-opt dm.fs=xfs
  ```

- `dm.mkfsarg`

  Specifies extra mkfs arguments to be used when creating the base device.

  Example use:

  ```
  $ sudo docker -d --storage-opt "dm.mkfsarg=-O ^has_journal"
  ```

- `dm.mountopt`

  Specifies extra mount options used when mounting the thin devices.

  Example use:

  ```
  $ sudo docker -d --storage-opt dm.mountopt=nodiscard
  ```

- `dm.datadev`

  Specifies a custom blockdevice to use for data for the thin pool.

  If using a block device for device mapper storage, ideally both datadev and metadatadev should be specified to completely avoid using the loopback device.

  Example use:

  ```
  $ sudo docker -d \
      --storage-opt dm.datadev=/dev/sdb1 \
      --storage-opt dm.metadatadev=/dev/sdc1
  ```

- `dm.metadatadev`

  Specifies a custom blockdevice to use for metadata for the thin pool.

  For best performance the metadata should be on a different spindle than the data, or even better on an SSD.

  If setting up a new metadata pool it is required to be valid. This can be achieved by zeroing the first 4k to indicate empty metadata, like this:

```
$ dd if=/dev/zero of=$metadata_dev bs=4096 count=1
```

Example use:

```
$ sudo docker -d \
    --storage-opt dm.datadev=/dev/sdb1 \
    --storage-opt dm.metadatadev=/dev/sdc1
```

- `dm.blocksize`

  Specifies a custom blocksize to use for the thin pool. The default blocksize is 64K.

  Example use:

  ```
  $ sudo docker -d --storage-opt dm.blocksize=512K
  ```

- `dm.blkdiscard`

  Enables or disables the use of blkdiscard when removing devicemapper devices. This is enabled by default (only) if using loopback devices and is required to res-parsify the loopback file on image/container removal.

  Disabling this on loopback can lead to *much* faster container removal times, but will make the space used in `/var/lib/docker` directory not be returned to the system for other use when containers are removed.

  Example use:

  ```
  $ sudo docker -d --storage-opt dm.blkdiscard=false
  ```

Docker exec-driver option

The Docker daemon uses a specifically built `libcontainer` execution driver as its interface to the Linux kernel `namespaces`, `cgroups`, and `SELinux`.

There is still legacy support for the original LXC userspace tools (https://linuxcontainers.org/) via the `lxc` execution driver, however, this is not where the primary development of new functionality is taking place. Add `-e lxc` to the daemon flags to use the `lxc` execution driver.

Daemon DNS options

To set the DNS server for all Docker containers, use `docker -d --dns 8.8.8.8`.

To set the DNS search domain for all Docker containers, use `docker -d --dns-search example.com`.

Insecure registries

Docker considers a private registry either secure or insecure. In the rest of this section, *registry* is used for *private registry*, and `myregistry:5000` is a placeholder example for a private registry.

A secure registry uses TLS and a copy of its CA certificate is placed on the Docker host at `/etc/docker/certs.d/myregistry:5000/ca.crt`. An insecure registry is either not using TLS (i.e., listening on plain text HTTP), or is using TLS with a CA certificate not known by the Docker daemon. The latter can happen when the certificate was not found under `/etc/docker/certs.d/myregistry:5000/`, or if the certificate verification failed (i.e., wrong CA).

By default, Docker assumes all, but local (see local registries below), registries are secure. Communicating with an insecure registry is not possible if Docker assumes that registry is secure. In order to communicate with an insecure registry, the Docker daemon requires `--insecure-registry` in one of the following two forms:

- `--insecure-registry myregistry:5000` tells the Docker daemon that myregistry:5000 should be considered insecure.
- `--insecure-registry 10.1.0.0/16` tells the Docker daemon that all registries whose domain resolve to an IP address is part of the subnet described by the CIDR syntax, should be considered insecure.

The flag can be used multiple times to allow multiple registries to be marked as insecure.

If an insecure registry is not marked as insecure, `docker pull`, `docker push`, and `docker search` will result in an error message prompting the user to either secure or pass the `--insecure-registry` flag to the Docker daemon as described above.

Local registries, whose IP address falls in the 127.0.0.0/8 range, are automatically marked as insecure as of Docker 1.3.2. It is not recommended to rely on this, as it may change in the future.

Running a Docker daemon behind a HTTPS_PROXY

When running inside a LAN that uses a `HTTPS` proxy, the Docker Hub certificates will be replaced by the proxy's certificates. These certificates need to be added to your Docker host's configuration:

1. Install the `ca-certificates` package for your distribution
2. Ask your network admin for the proxy's CA certificate and append them to `/etc/pki/tls/certs/ca-bundle.crt`
3. Then start your Docker daemon with `HTTPS_PROXY=http://username:password@proxy:port/ docker -d`. The `username:` and `password@` are optional - and are only needed if your proxy is set up to require authentication.

This will only add the proxy and authentication to the Docker daemon's requests - your `docker build`s and running containers will need extra configuration to use the proxy

Miscellaneous options

IP masquerading uses address translation to allow containers without a public IP to talk to other machines on the Internet. This may interfere with some network topologies and can be disabled with --ip-masq=false.

Docker supports softlinks for the Docker data directory (`/var/lib/docker`) and for `/var/lib/docker/tmp`. The `DOCKER_TMPDIR` and the data directory can be set like this:

```
DOCKER_TMPDIR=/mnt/disk2/tmp /usr/local/bin/docker -d -D -g /var/lib/docker -H unix:// > /var/lib/boot2docker/docker.log 2>
&1
# or
export DOCKER_TMPDIR=/mnt/disk2/tmp
/usr/local/bin/docker -d -D -g /var/lib/docker -H unix:// > /var/lib/boot2docker/docker.log 2>&1
```

attach

```
Usage: docker attach [OPTIONS] CONTAINER

Attach to a running container

  --no-stdin=false    Do not attach STDIN
  --sig-proxy=true    Proxy all received signals to the process (non-TTY mode only). SIGCHLD, SIGKILL, and SIGSTOP are not
proxied.
```

The `attach` command lets you view or interact with any running container's primary process (`pid 1`).

You can attach to the same contained process multiple times simultaneously, screen sharing style, or quickly view the progress of your daemonized process.

> ***Note:*** *This command is not for running a new process in a container. See:* `docker exec`*.*

You can detach from the container again (and leave it running) with `CTRL-p CTRL-q` (for a quiet exit), or `CTRL-c` which will send a SIGKILL to the container, or `CTRL-\` to get a stacktrace of the Docker client when it quits. When you detach from the container's process the exit code will be returned to the client.

To stop a container, use `docker stop`.

To kill the container, use `docker kill`.

Examples

```
$ ID=$(sudo docker run -d ubuntu /usr/bin/top -b)
$ sudo docker attach $ID
top - 02:05:52 up  3:05,  0 users,  load average: 0.01, 0.02, 0.05
Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.1%us,  0.2%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    373572k total,   355560k used,    18012k free,    27872k buffers
Swap:   786428k total,        0k used,   786428k free,   221740k cached

PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1 root      20   0 17200 1116  912 R    0  0.3   0:00.03 top

 top - 02:05:55 up  3:05,  0 users,  load average: 0.01, 0.02, 0.05
 Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
 Cpu(s):  0.0%us,  0.2%sy,  0.0%ni, 99.8%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
 Mem:    373572k total,   355244k used,    18328k free,    27872k buffers
 Swap:   786428k total,        0k used,   786428k free,   221776k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
       1 root      20   0 17208 1144  932 R    0  0.3   0:00.03 top


 top - 02:05:58 up  3:06,  0 users,  load average: 0.01, 0.02, 0.05
 Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
 Cpu(s):  0.2%us,  0.3%sy,  0.0%ni, 99.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
 Mem:    373572k total,   355780k used,    17792k free,    27880k buffers
 Swap:   786428k total,        0k used,   786428k free,   221776k cached

 PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
      1 root      20   0 17208 1144  932 R    0  0.3   0:00.03 top
^C$
$ sudo docker stop $ID
```

build

```
Usage: docker build [OPTIONS] PATH | URL | -

Build a new image from the source code at PATH

  --force-rm=false         Always remove intermediate containers, even after unsuccessful builds
  --no-cache=false         Do not use cache when building the image
  --pull=false             Always attempt to pull a newer version of the image
  -q, --quiet=false        Suppress the verbose output generated by the containers
  --rm=true                Remove intermediate containers after a successful build
  -t, --tag=""             Repository name (and optionally a tag) to be applied to the resulting image in case of success
```

Use this command to build Docker images from a Dockerfile and a "context".

The files at `PATH` or `URL` are called the "context" of the build. The build process may refer to any of the files in the context, for example when using an *ADD* (/reference/builder/#add) instruction. When a single Dockerfile is given as `URL` or is piped through `STDIN` (`docker build - < Dockerfile`), then no context is set.

When a Git repository is set as `URL`, then the repository is used as the context. The Git repository is cloned with its submodules (`git clone -recursive`). A fresh `git clone` occurs in a temporary directory on your local host, and then this is sent to the Docker daemon as the context. This way, your local user credentials and VPN's etc can be used to access private repositories.

If a file named `.dockerignore` exists in the root of `PATH` then it is interpreted as a newline-separated list of exclusion patterns. Exclusion patterns match files or directories relative to `PATH` that will be excluded from the context. Globbing is done using Go's filepath.Match (http://golang.org/pkg/path/filepath#Match) rules.

Please note that `.dockerignore` files in other subdirectories are considered as normal files. Filepaths in .dockerignore are absolute with the current directory as the root. Wildcards are allowed but the search is not recursive.

Example .dockerignore file

```
*/temp*
*/*/temp*
temp?
```

The first line above `*/temp*`, would ignore all files with names starting with `temp` from any subdirectory below the root directory. For example, a file named `/somedir/temporary.txt` would be ignored. The second line `*/*/temp*`, will ignore files starting with name `temp` from any subdirectory that is two levels below the root directory. For example, the file `/somedir/subdir/temporary.txt` would get ignored

in this case. The last line in the above example `temp?` will ignore the files that match the pattern from the root directory. For example, the files `tempa`, `tempb` are ignored from the root directory. Currently there is no support for regular expressions. Formats like `[^temp*]` are ignored.

See also:

Dockerfile Reference (/reference/builder).

Examples

```
$ sudo docker build .
Uploading context 10240 bytes
Step 1 : FROM busybox
Pulling repository busybox
 ---> e9aa60c60128MB/2.284 MB (100%) endpoint: https://cdn-registry-1.docker.io/v1/
Step 2 : RUN ls -lh /
 ---> Running in 9c9e81692ae9
total 24
drwxr-xr-x    2 root     root        4.0K Mar 12  2013 bin
drwxr-xr-x    5 root     root        4.0K Oct 19 00:19 dev
drwxr-xr-x    2 root     root        4.0K Oct 19 00:19 etc
drwxr-xr-x    2 root     root        4.0K Nov 15 23:34 lib
lrwxrwxrwx    1 root     root           3 Mar 12  2013 lib64 -> lib
dr-xr-xr-x  116 root     root           0 Nov 15 23:34 proc
lrwxrwxrwx    1 root     root           3 Mar 12  2013 sbin -> bin
dr-xr-xr-x   13 root     root           0 Nov 15 23:34 sys
drwxr-xr-x    2 root     root        4.0K Mar 12  2013 tmp
drwxr-xr-x    2 root     root        4.0K Nov 15 23:34 usr
 ---> b35f4035db3f
Step 3 : CMD echo Hello world
 ---> Running in 02071fceb21b
 ---> f52f38b7823e
Successfully built f52f38b7823e
Removing intermediate container 9c9e81692ae9
Removing intermediate container 02071fceb21b
```

This example specifies that the `PATH` is `.`, and so all the files in the local directory get `tar`d and sent to the Docker daemon. The `PATH` specifies where to find the files for the "context" of the build on the Docker daemon. Remember that the daemon could be running on a remote machine and that no parsing of the Dockerfile happens at the client side (where you're running `docker build`). That means that *all* the files at `PATH` get sent, not just the ones listed to *ADD* (/reference/builder/#add) in the Dockerfile.

The transfer of context from the local machine to the Docker daemon is what the `docker` client means when you see the "Sending build context" message.

If you wish to keep the intermediate containers after the build is complete, you must use `--rm=false`. This does not affect the build cache.

```
$ sudo docker build .
Uploading context 18.829 MB
Uploading context
Step 0 : FROM busybox
 ---> 769b9341d937
Step 1 : CMD echo Hello world
 ---> Using cache
 ---> 99cc1ad10469
Successfully built 99cc1ad10469
$ echo ".git" > .dockerignore
$ sudo docker build .
Uploading context  6.76 MB
Uploading context
Step 0 : FROM busybox
 ---> 769b9341d937
Step 1 : CMD echo Hello world
 ---> Using cache
 ---> 99cc1ad10469
Successfully built 99cc1ad10469
```

This example shows the use of the `.dockerignore` file to exclude the `.git` directory from the context. Its effect can be seen in the changed size of the uploaded context.

```
$ sudo docker build -t vieux/apache:2.0 .
```

This will build like the previous example, but it will then tag the resulting image. The repository name will be `vieux/apache` and the tag will be `2.0`

```
$ sudo docker build - < Dockerfile
```

This will read a Dockerfile from `STDIN` without context. Due to the lack of a context, no contents of any local directory will be sent to the Docker daemon. Since there is no context, a Dockerfile `ADD` only works if it refers to a remote URL.

```
$ sudo docker build - < context.tar.gz
```

This will build an image for a compressed context read from `STDIN`. Supported formats are: bzip2, gzip and xz.

```
$ sudo docker build github.com/creack/docker-firefox
```

This will clone the GitHub repository and use the cloned repository as context. The Dockerfile at the root of the repository is used as Dockerfile. Note that you can specify an arbitrary Git repository by using the `git://` or `git@` schema.

> ***Note:*** *`docker build` will return a `no such file or directory` error if the file or directory does not exist in the uploaded context. This may happen if there is no context, or if you specify a file that is elsewhere on the Host system. The context is limited to the current directory (and its children) for security reasons, and to ensure repeatable builds on remote Docker hosts. This is also the reason why `ADD ../file` will not work.*

commit

```
Usage: docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

Create a new image from a container's changes

  -a, --author=""     Author (e.g., "John Hannibal Smith <hannibal@a-team.com>")
  -m, --message=""    Commit message
  -p, --pause=true    Pause container during commit
```

It can be useful to commit a container's file changes or settings into a new image. This allows you debug a container by running an interactive shell, or to export a working dataset to another server. Generally, it is better to use Dockerfiles to manage your images in a documented and maintainable way.

By default, the container being committed and its processes will be paused while the image is committed. This reduces the likelihood of encountering data corruption during the process of creating the commit. If this behavior is undesired, set the 'p' option to false.

Commit an existing container

```
$ sudo docker ps
ID                  IMAGE               COMMAND             CREATED             STATUS              PORTS
c3f279d17e0a        ubuntu:12.04        /bin/bash           7 days ago          Up 25 hours
197387f1b436        ubuntu:12.04        /bin/bash           7 days ago          Up 25 hours
$ sudo docker commit c3f279d17e0a  SvenDowideit/testimage:version3
f5283438590d
$ sudo docker images | head
REPOSITORY                        TAG                 ID                  CREATED             VIRTUAL SIZE
SvenDowideit/testimage            version3            f5283438590d        16 seconds ago      335.7 MB
```

cp

Copy files/folders from a container's filesystem to the host path. Paths are relative to the root of the filesystem.

```
Usage: docker cp CONTAINER:PATH HOSTPATH

Copy files/folders from the PATH to the HOSTPATH
```

create

Creates a new container.

```
Usage: docker create [OPTIONS] IMAGE [COMMAND] [ARG...]

Create a new container

  -a, --attach=[]            Attach to STDIN, STDOUT or STDERR.
  --add-host=[]              Add a custom host-to-IP mapping (host:ip)
  -c, --cpu-shares=0         CPU shares (relative weight)
  --cap-add=[]               Add Linux capabilities
  --cap-drop=[]              Drop Linux capabilities
  --cidfile=""               Write the container ID to the file
  --cpuset=""                CPUs in which to allow execution (0-3, 0,1)
  --device=[]                Add a host device to the container (e.g. --device=/dev/sdc:/dev/xvdc:rwm)
  --dns=[]                   Set custom DNS servers
  --dns-search=[]            Set custom DNS search domains (Use --dns-search=. if you don't wish to set the search domain)
  -e, --env=[]               Set environment variables
  --entrypoint=""            Overwrite the default ENTRYPOINT of the image
  --env-file=[]              Read in a line delimited file of environment variables
  --expose=[]                Expose a port or a range of ports (e.g. --expose=3300-3310) from the container without publish
ing it to your host
  -h, --hostname=""          Container host name
  -i, --interactive=false    Keep STDIN open even if not attached
  --ipc=""                   Default is to create a private IPC namespace (POSIX SysV IPC) for the container
                               'container:<name|id>': reuses another container shared memory, semaphores and message queues
                               'host': use the host shared memory,semaphores and message queues inside the container.  Note
: the host mode gives the container full access to local shared memory and is therefore considered insecure.
  --link=[]                  Add link to another container in the form of name:alias
  --lxc-conf=[]              (lxc exec-driver only) Add custom lxc options --lxc-conf="lxc.cgroup.cpuset.cpus = 0,1"
  -m, --memory=""            Memory limit (format: <number><optional unit>, where unit = b, k, m or g)
  --mac-address=""           Container MAC address (e.g. 92:d0:c6:0a:29:33)
  --name=""                  Assign a name to the container
  --net="bridge"             Set the Network mode for the container
                               'bridge': creates a new network stack for the container on the docker bridge
                               'none': no networking for this container
                               'container:<name|id>': reuses another container network stack
                               'host': use the host network stack inside the container.  Note: the host mode gives the cont
ainer full access to local system services such as D-bus and is therefore considered insecure.
  -P, --publish-all=false    Publish all exposed ports to the host interfaces
  -p, --publish=[]           Publish a container's port to the host
                               format: ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort | containerPo
rt
                               (use 'docker port' to see the actual mapping)
  --privileged=false         Give extended privileges to this container
  --restart=""               Restart policy to apply when a container exits (no, on-failure[:max-retry], always)
  --security-opt=[]          Security Options
  -t, --tty=false            Allocate a pseudo-TTY
  -u, --user=""              Username or UID
  -v, --volume=[]            Bind mount a volume (e.g., from the host: -v /host:/container, from Docker: -v /container)
  --volumes-from=[]          Mount volumes from the specified container(s)
  -w, --workdir=""           Working directory inside the container
```

The `docker create` command creates a writeable container layer over the specified image and prepares it for running the specified command. The container ID is then printed to `STDOUT`. This is similar to `docker run -d` except the container is never started. You can then use the `docker start <container_id>` command to start the container at any point.

This is useful when you want to set up a container configuration ahead of time so that it is ready to start when you need it.

Note that volumes set by `create` may be over-ridden by options set with `start`.

Please see the run command section for more details.

Examples

```
$ sudo docker create -t -i fedora bash
6d8af538ec541dd581ebc2a24153a28329acb5268abe5ef868c1f1a261221752
$ sudo docker start -a -i 6d8af538ec5
bash-4.2#
```

As of v1.4.0 container volumes are initialized during the `docker create` phase (i.e., `docker run` too). For example, this allows you to `create` the `data` volume container, and then use it from another container:

```
$ docker create -v /data --name data ubuntu
240633dfbb98128fa77473d3d9018f6123b99c454b3251427ae190a7d951ad57
$ docker run --rm --volumes-from data ubuntu ls -la /data
total 8
drwxr-xr-x  2 root root 4096 Dec  5 04:10 .
drwxr-xr-x 48 root root 4096 Dec  5 04:11 ..
```

Similarly, `create` a host directory bind mounted volume container, which can then be used from the subsequent container:

```
$ docker create -v /home/docker:/docker --name docker ubuntu
9aa88c08f319cd1e4515c3c46b0de7cc9aa75e878357b1e96f91e2c773029f03
$ docker run --rm --volumes-from docker ubuntu ls -la /docker
total 20
drwxr-sr-x  5 1000 staff  180 Dec  5 04:00 .
drwxr-xr-x 48 root root  4096 Dec  5 04:13 ..
-rw-rw-r--  1 1000 staff 3833 Dec  5 04:01 .ash_history
-rw-r--r--  1 1000 staff  446 Nov 28 11:51 .ashrc
-rw-r--r--  1 1000 staff   25 Dec  5 04:00 .gitconfig
drwxr-sr-x  3 1000 staff   60 Dec  1 03:28 .local
-rw-r--r--  1 1000 staff  920 Nov 28 11:51 .profile
drwx--S---  2 1000 staff  460 Dec  5 00:51 .ssh
drwxr-xr-x 32 1000 staff 1140 Dec  5 04:01 docker
```

diff

List the changed files and directories in a container's filesystem

```
Usage: docker diff CONTAINER

Inspect changes on a container's filesystem
```

There are 3 events that are listed in the `diff`:

1. `A` - Add
2. `D` - Delete
3. `C` - Change

For example:

```
$ sudo docker diff 7bb0e258aefe

C /dev
A /dev/kmsg
C /etc
A /etc/mtab
A /go
A /go/src
A /go/src/github.com
A /go/src/github.com/docker
A /go/src/github.com/docker/docker
A /go/src/github.com/docker/docker/.git
....
```

events

```
Usage: docker events [OPTIONS]

Get real time events from the server

  -f, --filter=[]    Provide filter values (i.e., 'event=stop')
  --since=""         Show all events created since timestamp
  --until=""         Stream events until this timestamp
```

Docker containers will report the following events:

```
create, destroy, die, export, kill, pause, restart, start, stop, unpause
```

and Docker images will report:

```
untag, delete
```

Filtering

The filtering flag (`-f` or `--filter`) format is of "key=value". If you would like to use multiple filters, pass multiple flags (e.g., `--filter "foo=bar" --filter "bif=baz"`)

Using the same filter multiple times will be handled as a *OR*; for example `--filter container=588a23dac085 --filter container=a8f7720b8c22` will display events for container 588a23dac085 *OR* container a8f7720b8c22

Using multiple filters will be handled as a *AND*; for example `--filter container=588a23dac085 --filter event=start` will display events for container container 588a23dac085 *AND* the event type is *start*

Current filters: * event * image * container

Examples

You'll need two shells for this example.

Shell 1: Listening for events:

```
$ sudo docker events
```

Shell 2: Start and Stop containers:

```
$ sudo docker start 4386fb97867d
$ sudo docker stop 4386fb97867d
$ sudo docker stop 7805c1d35632
```

Shell 1: (Again .. now showing events):

```
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) start
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) die
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) stop
2014-05-10T17:42:14.999999999Z07:00 7805c1d35632: (from redis:2.8) die
2014-05-10T17:42:14.999999999Z07:00 7805c1d35632: (from redis:2.8) stop
```

Show events in the past from a specified time:

```
$ sudo docker events --since 1378216169
2014-03-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) die
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) stop
2014-05-10T17:42:14.999999999Z07:00 7805c1d35632: (from redis:2.8) die
2014-03-10T17:42:14.999999999Z07:00 7805c1d35632: (from redis:2.8) stop

$ sudo docker events --since '2013-09-03'
2014-09-03T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) start
2014-09-03T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) die
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) stop
2014-05-10T17:42:14.999999999Z07:00 7805c1d35632: (from redis:2.8) die
2014-09-03T17:42:14.999999999Z07:00 7805c1d35632: (from redis:2.8) stop

$ sudo docker events --since '2013-09-03 15:49:29 +0200 CEST'
2014-09-03T15:49:29.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) die
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) stop
2014-05-10T17:42:14.999999999Z07:00 7805c1d35632: (from redis:2.8) die
2014-09-03T15:49:29.999999999Z07:00 7805c1d35632: (from redis:2.8) stop
```

Filter events:

```
$ sudo docker events --filter 'event=stop'
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) stop
2014-09-03T17:42:14.999999999Z07:00 7805c1d35632: (from redis:2.8) stop

$ sudo docker events --filter 'image=ubuntu-1:14.04'
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) start
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) die
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) stop

$ sudo docker events --filter 'container=7805c1d35632'
2014-05-10T17:42:14.999999999Z07:00 7805c1d35632: (from redis:2.8) die
2014-09-03T15:49:29.999999999Z07:00 7805c1d35632: (from redis:2.8) stop

$ sudo docker events --filter 'container=7805c1d35632' --filter 'container=4386fb97867d'
2014-09-03T15:49:29.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) die
2014-05-10T17:42:14.999999999Z07:00 4386fb97867d: (from ubuntu-1:14.04) stop
2014-05-10T17:42:14.999999999Z07:00 7805c1d35632: (from redis:2.8) die
2014-09-03T15:49:29.999999999Z07:00 7805c1d35632: (from redis:2.8) stop

$ sudo docker events --filter 'container=7805c1d35632' --filter 'event=stop'
2014-09-03T15:49:29.999999999Z07:00 7805c1d35632: (from redis:2.8) stop
```

exec

```
Usage: docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container

  -d, --detach=false         Detached mode: run command in the background
  -i, --interactive=false    Keep STDIN open even if not attached
  -t, --tty=false            Allocate a pseudo-TTY
```

The `docker exec` command runs a new command in a running container.

The command started using `docker exec` will only run while the container's primary process (`PID 1`) is running, and will not be restarted if the container is restarted.

If the container is paused, then the `docker exec` command will fail with an error:

```
$ docker pause test
test
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS
  NAMES
1ae3b36715d2        ubuntu:latest       "bash"              17 seconds ago      Up 16 seconds (Paused)
  test
$ docker exec test ls
FATA[0000] Error response from daemon: Container test is paused, unpause the container before exec
$ echo $?
1
```

Examples

```
$ sudo docker run --name ubuntu_bash --rm -i -t ubuntu bash
```

This will create a container named `ubuntu_bash` and start a Bash session.

```
$ sudo docker exec -d ubuntu_bash touch /tmp/execWorks
```

This will create a new file `/tmp/execWorks` inside the running container `ubuntu_bash`, in the background.

```
$ sudo docker exec -it ubuntu_bash bash
```

This will create a new Bash session in the container `ubuntu_bash`.

export

```
Usage: docker export CONTAINER

Export the contents of a filesystem as a tar archive to STDOUT
```

For example:

```
$ sudo docker export red_panda > latest.tar
```

> ***Note:*** *`docker export` does not export the contents of volumes associated with the container. If a volume is mounted on top of an existing directory in the container, `docker export` will export the contents of the underlying directory, not the contents of the volume. Refer to Backup, restore, or migrate data volumes (/userguide/dockervolumes/#backup-restore-or-migrate-data-volumes) in the user guide for examples on exporting data in a volume.*

history

```
Usage: docker history [OPTIONS] IMAGE

Show the history of an image

  --no-trunc=false     Don't truncate output
  -q, --quiet=false    Only show numeric IDs
```

To see how the `docker:latest` image was built:

```
$ sudo docker history docker
IMAGE                                                         CREATED             CREATED BY
                                                                                                                  SIZE
3e23a5875458790b7a806f95f7ec0d0b2a5c1659bfc899c89f939f6d5b8f7094   8 days ago          /bin/sh -c #(nop) ENV LC_ALL=C.UTF-8
                                                                                                                  0 B
8578938dd17054dce7993d21de79e96a037400e8d28e15e7290fea4f65128a36   8 days ago          /bin/sh -c dpkg-reconfigure locales
&&    locale-gen C.UTF-8 &&    /usr/sbin/update-locale LANG=C.UTF-8                                                 1.24
5 MB
be51b77efb42f67a5e96437b3e102f81e0a1399038f77bf28cea0ed23a65cf60   8 days ago          /bin/sh -c apt-get update && apt-get
 install -y    git    libxml2-dev    python    build-essential    make    gcc    python-dev    locales    python-pip   338.
3 MB
4b137612be55ca69776c7f30c2d2dd0aa2e7d72059820abf3e25b629f887a084   6 weeks ago         /bin/sh -c #(nop) ADD jessie.tar.xz
in /                                                                                                              121
MB
750d58736b4b6cc0f9a9abe8f258cef269e3e9dceced1146503522be9f985ada   6 weeks ago         /bin/sh -c #(nop) MAINTAINER Tianon
Gravi <admwiggin@gmail.com> - mkimage-debootstrap.sh -t jessie.tar.xz jessie http://http.debian.net/debian            0 B
511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158   9 months ago
                                                                                                                  0 B
```

images

```
Usage: docker images [OPTIONS] [REPOSITORY]

List images

  -a, --all=false      Show all images (by default filter out the intermediate image layers)
  -f, --filter=[]      Provide filter values (i.e., 'dangling=true')
  --no-trunc=false     Don't truncate output
  -q, --quiet=false    Only show numeric IDs
```

The default `docker images` will show all top level images, their repository and tags, and their virtual size.

Docker images have intermediate layers that increase reusability, decrease disk usage, and speed up `docker build` by allowing each step to be cached. These intermediate layers are not shown by default.

The `VIRTUAL SIZE` is the cumulative space taken up by the image and all its parent images. This is also the disk space used by the contents of the Tar file created when you `docker save` an image.

An image will be listed more than once if it has multiple repository names or tags. This single image (identifiable by its matching `IMAGE ID`) uses up the `VIRTUAL SIZE` listed only once.

Listing the most recently created images

```
$ sudo docker images | head
REPOSITORY                TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
<none>                    <none>              77af4d6b9913        19 hours ago        1.089 GB
committ                   latest              b6fa739cedf5        19 hours ago        1.089 GB
<none>                    <none>              78a85c484f71        19 hours ago        1.089 GB
docker                    latest              30557a29d5ab        20 hours ago        1.089 GB
<none>                    <none>              5ed6274db6ce        24 hours ago        1.089 GB
postgres                  9                   746b819f315e        4 days ago          213.4 MB
postgres                  9.3                 746b819f315e        4 days ago          213.4 MB
postgres                  9.3.5               746b819f315e        4 days ago          213.4 MB
postgres                  latest              746b819f315e        4 days ago          213.4 MB
```

Listing the full length image IDs

```
$ sudo docker images --no-trunc | head
REPOSITORY                    TAG                 IMAGE ID                                                           CREATE
D             VIRTUAL SIZE
<none>                        <none>              77af4d6b9913e693e8d0b4b294fa62ade6054e6b2f1ffb617ac955dd63fb0182   19 hou
rs ago        1.089 GB
committest                    latest              b6fa739cedf5ea12a620a439402b6004d057da800f91c7524b5086a5e4749c9f   19 hou
rs ago        1.089 GB
<none>                        <none>              78a85c484f71509adeaace20e72e941f6bdd2b25b4c75da8693efd9f61a37921   19 hou
rs ago        1.089 GB
docker                        latest              30557a29d5abc51e5f1d5b472e79b7e296f595abcf19fe6b9199dbbc809c6ff4   20 hou
rs ago        1.089 GB
<none>                        <none>              0124422dd9f9cf7ef15c0617cda3931ee68346455441d66ab8bdc5b05e9fdce5   20 hou
rs ago        1.089 GB
<none>                        <none>              18ad6fad340262ac2a636efd98a6d1f0ea775ae3d45240d3418466495a19a81b   22 hou
rs ago        1.082 GB
<none>                        <none>              f9f1e26352f0a3ba6a0ff68167559f64f3e21ff7ada60366e2d44a04befd1d3a   23 hou
rs ago        1.089 GB
tryout                        latest              2629d1fa0b81b222fca63371ca16cbf6a0772d07759ff80e8d1369b926940074   23 hou
rs ago        131.5 MB
<none>                        <none>              5ed6274db6ceb2397844896966ea239290555e74ef307030ebb01ff91b1914df   24 hou
rs ago        1.089 GB
```

Filtering

The filtering flag (`-f` or `--filter`) format is of "key=value". If there is more than one filter, then pass multiple flags (e.g., `--filter "foo=bar" --filter "bif=baz"`)

Current filters: * dangling (boolean - true or false)

Untagged images

```
$ sudo docker images --filter "dangling=true"

REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
<none>              <none>              8abc22fbb042        4 weeks ago         0 B
<none>              <none>              48e5f45168b9        4 weeks ago         2.489 MB
<none>              <none>              bf747efa0e2f        4 weeks ago         0 B
<none>              <none>              980fe10e5736        12 weeks ago        101.4 MB
<none>              <none>              dea752e4e117        12 weeks ago        101.4 MB
<none>              <none>              511136ea3c5a        8 months ago        0 B
```

This will display untagged images, that are the leaves of the images tree (not intermediary layers). These images occur when a new build of an image takes the `repo:tag` away from the image ID, leaving it untagged. A warning will be issued if trying to remove an image when a container is presently using it. By having this flag it allows for batch cleanup.

Ready for use by `docker rmi ...`, like:

```
$ sudo docker rmi $(sudo docker images -f "dangling=true" -q)

8abc22fbb042
48e5f45168b9
bf747efa0e2f
980fe10e5736
dea752e4e117
511136ea3c5a
```

NOTE: Docker will warn you if any containers exist that are using these untagged images.

import

```
Usage: docker import URL|- [REPOSITORY[:TAG]]

Create an empty filesystem image and import the contents of the tarball (.tar, .tar.gz, .tgz, .bzip, .tar.xz, .txz) into it
, then optionally tag it.
```

URLs must start with `http` and point to a single file archive (.tar, .tar.gz, .tgz, .bzip, .tar.xz, or .txz) containing a root filesystem. If you would like to import from a local directory or archive, you can use the `-` parameter to take the data from `STDIN`.

Examples

Import from a remote location:

This will create a new untagged image.

```
$ sudo docker import http://example.com/exampleimage.tgz
```

Import from a local file:

Import to docker via pipe and `STDIN`.

```
$ cat exampleimage.tgz | sudo docker import - exampleimagelocal:new
```

Import from a local directory:

```
$ sudo tar -c . | sudo docker import - exampleimagedir
```

Note the `sudo` in this example – you must preserve the ownership of the files (especially root ownership) during the archiving with tar. If you are not root (or the sudo command) when you tar, then the ownerships might not get preserved.

info

```
Usage: docker info

Display system-wide information
```

For example:

```
$ sudo docker -D info
Containers: 14
Images: 52
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Dirs: 545
Execution Driver: native-0.2
Kernel Version: 3.13.0-24-generic
Operating System: Ubuntu 14.04 LTS
CPUs: 1
Name: prod-server-42
ID: 7TRN:IPZB:QYBB:VPBQ:UMPP:KARE:6ZNR:XE6T:7EWV:PKF4:ZOJD:TPYS
Total Memory: 2 GiB
Debug mode (server): false
Debug mode (client): true
Fds: 10
Goroutines: 9
EventsListeners: 0
Init Path: /usr/bin/docker
Docker Root Dir: /var/lib/docker
Username: svendowideit
Registry: [https://index.docker.io/v1/]
Labels:
 storage=ssd
```

The global `-D` option tells all `docker` commands to output debug information.

When sending issue reports, please use `docker version` and `docker -D info` to ensure we know how your setup is configured.

inspect

```
Usage: docker inspect [OPTIONS] CONTAINER|IMAGE [CONTAINER|IMAGE...]

Return low-level information on a container or image

  -f, --format=""    Format the output using the given go template.
```

By default, this will render all results in a JSON array. If a format is specified, the given template will be executed for each result.

Go's text/template (http://golang.org/pkg/text/template/) package describes all the details of the format.

Examples

Get an instance's IP address:

For the most part, you can pick out any field from the JSON in a fairly straightforward manner.

```
$ sudo docker inspect --format='{{.NetworkSettings.IPAddress}}' $INSTANCE_ID
```

Get an instance's MAC Address:

For the most part, you can pick out any field from the JSON in a fairly straightforward manner.

```
$ sudo docker inspect --format='{{.NetworkSettings.MacAddress}}' $INSTANCE_ID
```

List All Port Bindings:

One can loop over arrays and maps in the results to produce simple text output:

```
$ sudo docker inspect --format='{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} -> {{(index $conf 0).HostPort}} {{end}
}' $INSTANCE_ID
```

Find a Specific Port Mapping:

The `.Field` syntax doesn't work when the field name begins with a number, but the template language's `index` function does. The `.NetworkSettings.Ports` section contains a map of the internal port mappings to a list of external address/port objects, so to grab just the numeric public port, you use `index` to find the specific port map, and then `index` 0 contains the first object inside of that. Then we ask for the `HostPort` field to get the public address.

```
$ sudo docker inspect --format='{{(index (index .NetworkSettings.Ports "8787/tcp") 0).HostPort}}' $INSTANCE_ID
```

Get config:

The `.Field` syntax doesn't work when the field contains JSON data, but the template language's custom `json` function does. The `.config` section contains complex JSON object, so to grab it as JSON, you use `json` to convert the configuration object into JSON.

```
$ sudo docker inspect --format='{{json .config}}' $INSTANCE_ID
```

kill

```
Usage: docker kill [OPTIONS] CONTAINER [CONTAINER...]

Kill a running container using SIGKILL or a specified signal

  -s, --signal="KILL"    Signal to send to the container
```

The main process inside the container will be sent `SIGKILL`, or any signal specified with option `--signal`.

load

```
Usage: docker load [OPTIONS]

Load an image from a tar archive on STDIN

  -i, --input=""     Read from a tar archive file, instead of STDIN
```

Loads a tarred repository from a file or the standard input stream. Restores both images and tags.

```
$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
$ sudo docker load < busybox.tar
$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
busybox             latest              769b9341d937        7 weeks ago         2.489 MB
$ sudo docker load --input fedora.tar
$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
busybox             latest              769b9341d937        7 weeks ago         2.489 MB
fedora              rawhide             0d20aec6529d        7 weeks ago         387 MB
fedora              20                  58394af37342        7 weeks ago         385.5 MB
fedora              heisenbug           58394af37342        7 weeks ago         385.5 MB
fedora              latest              58394af37342        7 weeks ago         385.5 MB
```

login

```
Usage: docker login [OPTIONS] [SERVER]

Register or log in to a Docker registry server, if no server is specified "https://index.docker.io/v1/" is the default.

  -e, --email=""       Email
  -p, --password=""    Password
  -u, --username=""    Username
```

If you want to login to a self-hosted registry you can specify this by adding the server name.

```
example:
$ sudo docker login localhost:8080
```

logout

```
Usage: docker logout [SERVER]

Log out from a Docker registry, if no server is specified "https://index.docker.io/v1/" is the default.
```

For example:

```
$ sudo docker logout localhost:8080
```

logs

```
Usage: docker logs [OPTIONS] CONTAINER

Fetch the logs of a container

  -f, --follow=false        Follow log output
  -t, --timestamps=false    Show timestamps
  --tail="all"              Output the specified number of lines at the end of logs (defaults to all logs)
```

The `docker logs` command batch-retrieves logs present at the time of execution.

The `docker logs --follow` command will continue streaming the new output from the container's `STDOUT` and `STDERR`.

Passing a negative number or a non-integer to `--tail` is invalid and the value is set to `all` in that case. This behavior may change in the future.

The `docker logs --timestamp` commands will add an RFC3339Nano timestamp, for example `2014-09-16T06:17:46.000000000Z`, to each log entry. To ensure that the timestamps for are aligned the nano-second part of the timestamp will be padded with zero when necessary.

pause

```
Usage: docker pause CONTAINER

Pause all processes within a container
```

The `docker pause` command uses the cgroups freezer to suspend all processes in a container. Traditionally, when suspending a process the `SIGSTOP` signal is used, which is observable by the process being suspended. With the cgroups freezer the process is unaware, and unable to capture, that it is being suspended, and subsequently resumed.

See the cgroups freezer documentation (https://www.kernel.org/doc/Documentation/cgroups/freezer-subsystem.txt) for further details.

port

```
Usage: docker port CONTAINER [PRIVATE_PORT[/PROTO]]

List port mappings for the CONTAINER, or lookup the public-facing port that is NAT-ed to the PRIVATE_PORT
```

You can find out all the ports mapped by not specifying a `PRIVATE_PORT`, or just a specific mapping:

```
$ sudo docker ps test
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
                       NAMES
b650456536c7        busybox:latest      top                 54 minutes ago      Up 54 minutes       0.0.0.0:1234->9876/tcp,
 0.0.0.0:4321->7890/tcp   test
$ sudo docker port test
7890/tcp -> 0.0.0.0:4321
9876/tcp -> 0.0.0.0:1234
$ sudo docker port test 7890/tcp
0.0.0.0:4321
$ sudo docker port test 7890/udp
2014/06/24 11:53:36 Error: No public port '7890/udp' published for test
$ sudo docker port test 7890
0.0.0.0:4321
```

ps

```
Usage: docker ps [OPTIONS]

List containers

  -a, --all=false       Show all containers. Only running containers are shown by default.
  --before=""           Show only container created before Id or Name, include non-running ones.
  -f, --filter=[]       Provide filter values. Valid filters:
                          exited=<int> - containers with exit code of <int>
                          status=(restarting|running|paused|exited)
  -l, --latest=false    Show only the latest created container, include non-running ones.
  -n=-1                 Show n last created containers, include non-running ones.
  --no-trunc=false      Don't truncate output
  -q, --quiet=false     Only display numeric IDs
  -s, --size=false      Display total file sizes
  --since=""            Show only containers created since Id or Name, include non-running ones.
```

Running `docker ps` showing 2 linked containers.

```
$ sudo docker ps
CONTAINER ID        IMAGE                          COMMAND               CREATED             STATUS              PORTS
          NAMES
4c01db0b339c        ubuntu:12.04                   bash                  17 seconds ago      Up 16 seconds
          webapp
d7886598dbe2        crosbymichael/redis:latest     /redis-server --dir   33 minutes ago      Up 33 minutes       6379/tcp
          redis,webapp/db
```

`docker ps` will show only running containers by default. To see all containers: `docker ps -a`

Filtering

The filtering flag (`-f` or `--filter)` format is a `key=value` pair. If there is more than one filter, then pass multiple flags (e.g. `--filter "foo=bar" --filter "bif=baz"`)

Current filters: * exited (int - the code of exited containers. Only useful with '--all') * status (restarting|running|paused|exited)

Successfully exited containers

```
$ sudo docker ps -a --filter 'exited=0'
CONTAINER ID        IMAGE             COMMAND                CREATED             STATUS                     PORTS
          NAMES
ea09c3c82f6e        registry:latest   /srv/run.sh            2 weeks ago         Exited (0) 2 weeks ago   127.0.0.1:5000->5
000/tcp   desperate_leakey
106ea823fe4e        fedora:latest     /bin/sh -c 'bash -l'   2 weeks ago         Exited (0) 2 weeks ago
          determined_albattani
48ee228c9464        fedora:20         bash                   2 weeks ago         Exited (0) 2 weeks ago
          tender_torvalds
```

This shows all the containers that have exited with status of '0'

pull

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

Pull an image or a repository from the registry

  -a, --all-tags=false    Download all tagged images in the repository
```

Most of your images will be created on top of a base image from the Docker Hub (https://hub.docker.com) registry.

Docker Hub (https://hub.docker.com) contains many pre-built images that you can `pull` and try without needing to define and configure your own.

It is also possible to manually specify the path of a registry to pull from. For example, if you have set up a local registry, you can specify its path to pull from it. A repository path is similar to a URL, but does not contain a protocol specifier (`https://`, for example).

To download a particular image, or set of images (i.e., a repository), use `docker pull`:

```
$ sudo docker pull debian
# will pull the debian:latest image, its intermediate layers
# and any aliases of the same id
$ sudo docker pull debian:testing
# will pull the image named debian:testing and any intermediate
# layers it is based on.
# (Typically the empty `scratch` image, a MAINTAINER layer,
# and the un-tarred base).
$ sudo docker pull --all-tags centos
# will pull all the images from the centos repository
$ sudo docker pull registry.hub.docker.com/debian
# manually specifies the path to the default Docker registry. This could
# be replaced with the path to a local registry to pull from another source.
```

push

```
Usage: docker push NAME[:TAG]

Push an image or a repository to the registry
```

Use `docker push` to share your images to the Docker Hub (https://hub.docker.com) registry or to a self-hosted one.

restart

```
Usage: docker restart [OPTIONS] CONTAINER [CONTAINER...]

Restart a running container

  -t, --time=10      Number of seconds to try to stop for before killing the container. Once killed it will then be restart
ed. Default is 10 seconds.
```

rm

```
Usage: docker rm [OPTIONS] CONTAINER [CONTAINER...]

Remove one or more containers

  -f, --force=false      Force the removal of a running container (uses SIGKILL)
  -l, --link=false       Remove the specified link and not the underlying container
  -v, --volumes=false    Remove the volumes associated with the container
```

Examples

```
$ sudo docker rm /redis
/redis
```

This will remove the container referenced under the link `/redis`.

```
$ sudo docker rm --link /webapp/redis
/webapp/redis
```

This will remove the underlying link between `/webapp` and the `/redis` containers removing all network communication.

```
$ sudo docker rm --force redis
redis
```

The main process inside the container referenced under the link `/redis` will receive `SIGKILL`, then the container will be removed.

This command will delete all stopped containers. The command `docker ps -a -q` will return all existing container IDs and pass them to the `rm` command which will delete them. Any running containers will not be deleted.

rmi

```
Usage: docker rmi [OPTIONS] IMAGE [IMAGE...]

Remove one or more images

  -f, --force=false    Force removal of the image
  --no-prune=false     Do not delete untagged parents
```

Removing tagged images

Images can be removed either by their short or long IDs, or their image names. If an image has more than one name, each of them needs to be removed before the image is removed.

```
$ sudo docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
test1                     latest              fd484f19954f        23 seconds ago      7 B (virtual 4.964 MB)
test                      latest              fd484f19954f        23 seconds ago      7 B (virtual 4.964 MB)
test2                     latest              fd484f19954f        23 seconds ago      7 B (virtual 4.964 MB)

$ sudo docker rmi fd484f19954f
Error: Conflict, cannot delete image fd484f19954f because it is tagged in multiple repositories
2013/12/11 05:47:16 Error: failed to remove one or more images

$ sudo docker rmi test1
Untagged: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8
$ sudo docker rmi test2
Untagged: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8

$ sudo docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
test                      latest              fd484f19954f        23 seconds ago      7 B (virtual 4.964 MB)
$ sudo docker rmi test
Untagged: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8
Deleted: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8
```

run

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

Run a command in a new container

  -a, --attach=[]            Attach to STDIN, STDOUT or STDERR.
  --add-host=[]              Add a custom host-to-IP mapping (host:ip)
  -c, --cpu-shares=0         CPU shares (relative weight)
  --cap-add=[]               Add Linux capabilities
  --cap-drop=[]              Drop Linux capabilities
  --cidfile=""               Write the container ID to the file
  --cpuset=""                CPUs in which to allow execution (0-3, 0,1)
  -d, --detach=false         Detached mode: run the container in the background and print the new container ID
  --device=[]                Add a host device to the container (e.g. --device=/dev/sdc:/dev/xvdc:rwm)
  --dns=[]                   Set custom DNS servers
  --dns-search=[]            Set custom DNS search domains (Use --dns-search=. if you don't wish to set the search domain)
  -e, --env=[]               Set environment variables
  --entrypoint=""            Overwrite the default ENTRYPOINT of the image
  --env-file=[]              Read in a line delimited file of environment variables
  --expose=[]                Expose a port or a range of ports (e.g. --expose=3300-3310) from the container without publish
ing it to your host
  -h, --hostname=""          Container host name
  -i, --interactive=false    Keep STDIN open even if not attached
  --ipc=""                   Default is to create a private IPC namespace (POSIX SysV IPC) for the container
                               'container:<name|id>': reuses another container shared memory, semaphores and message queues
                               'host': use the host shared memory,semaphores and message queues inside the container.  Note
: the host mode gives the container full access to local shared memory and is therefore considered insecure.
  --link=[]                  Add link to another container in the form of name:alias
  --lxc-conf=[]              (lxc exec-driver only) Add custom lxc options --lxc-conf="lxc.cgroup.cpuset.cpus = 0,1"
  -m, --memory=""            Memory limit (format: <number><optional unit>, where unit = b, k, m or g)
  --mac-address=""           Container MAC address (e.g. 92:d0:c6:0a:29:33)
  --name=""                  Assign a name to the container
  --net="bridge"             Set the Network mode for the container
                               'bridge': creates a new network stack for the container on the docker bridge
                               'none': no networking for this container
                               'container:<name|id>': reuses another container network stack
                               'host': use the host network stack inside the container.  Note: the host mode gives the cont
ainer full access to local system services such as D-bus and is therefore considered insecure.
  -P, --publish-all=false    Publish all exposed ports to the host interfaces
  -p, --publish=[]           Publish a container's port to the host
                               format: ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort | containerPo
rt
                               (use 'docker port' to see the actual mapping)
  --privileged=false         Give extended privileges to this container
  --restart=""               Restart policy to apply when a container exits (no, on-failure[:max-retry], always)
  --rm=false                 Automatically remove the container when it exits (incompatible with -d)
  --security-opt=[]          Security Options
  --sig-proxy=true           Proxy received signals to the process (non-TTY mode only). SIGCHLD, SIGSTOP, and SIGKILL are n
ot proxied.
  -t, --tty=false            Allocate a pseudo-TTY
  -u, --user=""              Username or UID
  -v, --volume=[]            Bind mount a volume (e.g., from the host: -v /host:/container, from Docker: -v /container)
  --volumes-from=[]          Mount volumes from the specified container(s)
  -w, --workdir=""           Working directory inside the container
```

The `docker run` command first `creates` a writeable container layer over the specified image, and then `starts` it using the specified command. That is, `docker run` is equivalent to the API `/containers/create` then `/containers/(id)/start`. A stopped container can be restarted with all its previous changes intact using `docker start`. See `docker ps -a` to view a list of all containers.

There is detailed information about `docker run` in the Docker run reference (/reference/run/).

The `docker run` command can be used in combination with `docker commit` to *change the command that a container runs*.

See the Docker User Guide (/userguide/dockerlinks/) for more detailed information about the `--expose`, `-p`, `-P` and `--link` parameters, and linking containers.

Examples

```
$ sudo docker run --cidfile /tmp/docker_test.cid ubuntu echo "test"
```

This will create a container and print `test` to the console. The `cidfile` flag makes Docker attempt to create a new file and write the container ID to it. If the file exists already, Docker will return an error. Docker will close this file when `docker run` exits.

```
$ sudo docker run -t -i --rm ubuntu bash
root@bc338942ef20:/# mount -t tmpfs none /mnt
mount: permission denied
```

This will *not* work, because by default, most potentially dangerous kernel capabilities are dropped; including `cap_sys_admin` (which is required to mount filesystems). However, the `--privileged` flag will allow it to run:

```
$ sudo docker run --privileged ubuntu bash
root@50e3f57e16e6:/# mount -t tmpfs none /mnt
root@50e3f57e16e6:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
none            1.9G     0  1.9G   0% /mnt
```

The `--privileged` flag gives *all* capabilities to the container, and it also lifts all the limitations enforced by the `device` cgroup controller. In other words, the container can then do almost everything that the host can do. This flag exists to allow special use-cases, like running Docker within Docker.

```
$ sudo docker  run -w /path/to/dir/ -i -t  ubuntu pwd
```

The `-w` lets the command being executed inside directory given, here `/path/to/dir/`. If the path does not exists it is created inside the container.

```
$ sudo docker  run  -v `pwd`:`pwd` -w `pwd` -i -t  ubuntu pwd
```

The `-v` flag mounts the current working directory into the container. The `-w` lets the command being executed inside the current working directory, by changing into the directory to the value returned by `pwd`. So this combination executes the command using the container, but inside the current working directory.

```
$ sudo docker run -v /doesnt/exist:/foo -w /foo -i -t ubuntu bash
```

When the host directory of a bind-mounted volume doesn't exist, Docker will automatically create this directory on the host for you. In the example above, Docker will create the `/doesnt/exist` folder before starting your container.

```
$ sudo docker run -t -i -v /var/run/docker.sock:/var/run/docker.sock -v ./static-docker:/usr/bin/docker busybox sh
```

By bind-mounting the docker unix socket and statically linked docker binary (such as that provided by https://get.docker.com (https://get.docker.com)), you give the container the full access to create and manipulate the host's Docker daemon.

```
$ sudo docker run -p 127.0.0.1:80:8080 ubuntu bash
```

This binds port `8080` of the container to port `80` on `127.0.0.1` of the host machine. The Docker User Guide (/userguide/dockerlinks/) explains in detail how to manipulate ports in Docker.

```
$ sudo docker run --expose 80 ubuntu bash
```

This exposes port `80` of the container for use within a link without publishing the port to the host system's interfaces. The Docker User Guide (/userguide/dockerlinks) explains in detail how to manipulate ports in Docker.

```
$ sudo docker run -e MYVAR1 --env MYVAR2=foo --env-file ./env.list ubuntu bash
```

This sets environmental variables in the container. For illustration all three flags are shown here. Where `-e`, `--env` take an environment variable and value, or if no "=" is provided, then that variable's current value is passed through (i.e. `$MYVAR1` from the host is set to `$MYVAR1` in the container). All three flags, `-e`, `--env` and `--env-file` can be repeated.

Regardless of the order of these three flags, the `--env-file` are processed first, and then `-e`, `--env` flags. This way, the `-e` or `--env` will override variables as needed.

```
$ cat ./env.list
TEST_FOO=BAR
$ sudo docker run --env TEST_FOO="This is a test" --env-file ./env.list busybox env | grep TEST_FOO
TEST_FOO=This is a test
```

The `--env-file` flag takes a filename as an argument and expects each line to be in the `VAR=VAL` format, mimicking the argument passed to `--env`. Comment lines need only be prefixed with `#`

An example of a file passed with `--env-file`

```
$ cat ./env.list
TEST_FOO=BAR

# this is a comment
TEST_APP_DEST_HOST=10.10.0.127
TEST_APP_DEST_PORT=8888

# pass through this variable from the caller
TEST_PASSTHROUGH
$ sudo TEST_PASSTHROUGH=howdy docker run --env-file ./env.list busybox env
HOME=/
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=5198e0745561
TEST_FOO=BAR
TEST_APP_DEST_HOST=10.10.0.127
TEST_APP_DEST_PORT=8888
TEST_PASSTHROUGH=howdy

$ sudo docker run --name console -t -i ubuntu bash
```

This will create and run a new container with the container name being `console`.

```
$ sudo docker run --link /redis:redis --name console ubuntu bash
```

The `--link` flag will link the container named `/redis` into the newly created container with the alias `redis`. The new container can access the network and environment of the `redis` container via environment variables. The `--name` flag will assign the name `console` to the newly created container.

```
$ sudo docker run --volumes-from 777f7dc92da7 --volumes-from ba8c0c54f0f2:ro -i -t ubuntu pwd
```

The `--volumes-from` flag mounts all the defined volumes from the referenced containers. Containers can be specified by repetitions of the `--volumes-from` argument. The container ID may be optionally suffixed with `:ro` or `:rw` to mount the volumes in read-only or read-write mode, respectively. By default, the volumes are mounted in the same mode (read write or read only) as the reference container.

The `-a` flag tells `docker run` to bind to the container's `STDIN`, `STDOUT` or `STDERR`. This makes it possible to manipulate the output and input as needed.

```
$ echo "test" | sudo docker run -i -a stdin ubuntu cat -
```

This pipes data into a container and prints the container's ID by attaching only to the container's `STDIN`.

```
$ sudo docker run -a stderr ubuntu echo test
```

This isn't going to print anything unless there's an error because we've only attached to the `STDERR` of the container. The container's logs still store what's been written to `STDERR` and `STDOUT`.

```
$ cat somefile | sudo docker run -i -a stdin mybuilder dobuild
```

This is how piping a file into a container could be done for a build. The container's ID will be printed after the build is done and the build logs could be retrieved using `docker logs`. This is useful if you need to pipe a file or something else into a container and retrieve the container's ID once the container has finished running.

$ sudo docker run --device=/dev/sdc:/dev/xvdc --device=/dev/sdd --device=/dev/zero:/dev/nulo -i -t ubuntu ls -l /dev/{xvdc,sdd,nulo} brw-rw---- 1 root disk 8, 2 Feb 9 16:05 /dev/xvdc brw-rw---- 1 root disk 8, 3 Feb 9 16:05 /dev/sdd crw-rw-rw- 1 root root 1, 5 Feb 9 16:05 /dev/nulo

It is often necessary to directly expose devices to a container. The `--device` option enables that. For example, a specific block storage device or loop device or audio device can be added to an otherwise unprivileged container (without the `--privileged` flag) and have the application directly access it.

By default, the container will be able to `read`, `write` and `mknod` these devices. This can be overridden using a third `:rwm` set of options to each `--device` flag:

```
$ sudo docker run --device=/dev/sda:/dev/xvdc --rm -it ubuntu fdisk  /dev/xvdc

Command (m for help): q
$ sudo docker run --device=/dev/sda:/dev/xvdc:r --rm -it ubuntu fdisk  /dev/xvdc
You will not be able to write the partition table.

Command (m for help): q

$ sudo docker run --device=/dev/sda:/dev/xvdc --rm -it ubuntu fdisk  /dev/xvdc

Command (m for help): q

$ sudo docker run --device=/dev/sda:/dev/xvdc:m --rm -it ubuntu fdisk  /dev/xvdc
fdisk: unable to open /dev/xvdc: Operation not permitted
```

Note:

> `--device` *cannot be safely used with ephemeral devices. Block devices that may be removed should not be added to untrusted containers with* `--device`*.*

A complete example:

```
$ sudo docker run -d --name static static-web-files sh
$ sudo docker run -d --expose=8098 --name riak riakserver
$ sudo docker run -d -m 100m -e DEVELOPMENT=1 -e BRANCH=example-code -v $(pwd):/app/bin:ro --name app appserver
$ sudo docker run -d -p 1443:443 --dns=10.0.0.1 --dns-search=dev.org -v /var/log/httpd --volumes-from static --link riak --
link app -h www.sven.dev.org --name web webserver
$ sudo docker run -t -i --rm --volumes-from web -w /var/log/httpd busybox tail -f access.log
```

This example shows five containers that might be set up to test a web application change:

1. Start a pre-prepared volume image `static-web-files` (in the background) that has CSS, image and static HTML in it, (with a `VOLUME` instruction in the Dockerfile to allow the web server to use those files);
2. Start a pre-prepared `riakserver` image, give the container name `riak` and expose port `8098` to any containers that link to it;
3. Start the `appserver` image, restricting its memory usage to 100MB, setting two environment variables `DEVELOPMENT` and `BRANCH` and bind-mounting the current directory (`$(pwd)`) in the container in read-only mode as `/app/bin`;
4. Start the `webserver`, mapping port `443` in the container to port `1443` on the Docker server, setting the DNS server to `10.0.0.1` and DNS search domain to `dev.org`, creating a volume to put the log files into (so we can access it from another container), then importing the files from the volume exposed by the `static` container, and linking to all exposed ports from `riak` and `app`. Lastly, we set the hostname to `web.sven.dev.org` so its consistent with the pre-generated SSL certificate;
5. Finally, we create a container that runs `tail -f access.log` using the logs volume from the `web` container, setting the workdir to `/var/log/httpd`. The `--rm` option means that when the container exits, the container's layer is removed.

Restart Policies

Using the `--restart` flag on Docker run you can specify a restart policy for how a container should or should not be restarted on exit.

An ever increasing delay (double the previous delay, starting at 100 milliseconds) is added before each restart to prevent flooding the server. This means the daemaon will wait for 100 mS, then 200 mS, 400, 800, 1600, and so on until either the `on-failure` limit is hit, or when you `docker stop` or even `docker rm -f` the container.

When a restart policy is active on a container, it will be shown in `docker ps` as either `Up` or `Restarting` in `docker ps`. It can also be useful to use `docker events` to see the restart policy in effect.

no - Do not restart the container when it exits.

on-failure - Restart the container only if it exits with a non zero exit status.

always - Always restart the container regardless of the exit status.

You can also specify the maximum amount of times Docker will try to restart the container when using the **on-failure** policy. The default is that Docker will try forever to restart the container.

```
$ sudo docker run --restart=always redis
```

This will run the `redis` container with a restart policy of **always** so that if the container exits, Docker will restart it.

```
$ sudo docker run --restart=on-failure:10 redis
```

This will run the `redis` container with a restart policy of **on-failure** and a maximum restart count of 10. If the `redis` container exits with a non-zero exit status more than 10 times in a row Docker will abort trying to restart the container. Providing a maximum restart limit is only valid for the **on-failure** policy.

Adding entries to a container hosts file

You can add other hosts into a container's `/etc/hosts` file by using one or more `--add-host` flags. This example adds a static address for a host named `docker`:

```
    $ docker run --add-host=docker:10.180.0.1 --rm -it debian
    $$ ping docker
    PING docker (10.180.0.1): 48 data bytes
    56 bytes from 10.180.0.1: icmp_seq=0 ttl=254 time=7.600 ms
    56 bytes from 10.180.0.1: icmp_seq=1 ttl=254 time=30.705 ms
    ^C--- docker ping statistics ---
    2 packets transmitted, 2 packets received, 0% packet loss
    round-trip min/avg/max/stddev = 7.600/19.152/30.705/11.553 ms
```

> ***Note:*** *Sometimes you need to connect to the Docker host, which means getting the IP address of the host. You can use the following shell commands to simplify this process:*
>
> ```
> $ alias hostip="ip route show 0.0.0.0/0 | grep -Eo 'via \S+' | awk '{ print \$2 }'"
> $ docker run  --add-host=docker:$(hostip) --rm -it debian
> ```

save

```
Usage: docker save [OPTIONS] IMAGE [IMAGE...]

Save an image(s) to a tar archive (streamed to STDOUT by default)

  -o, --output=""    Write to a file, instead of STDOUT
```

Produces a tarred repository to the standard output stream. Contains all parent layers, and all tags + versions, or specified `repo:tag`, for each argument provided.

It is used to create a backup that can then be used with `docker load`

```
$ sudo docker save busybox > busybox.tar
$ ls -sh busybox.tar
2.7M busybox.tar
$ sudo docker save --output busybox.tar busybox
$ ls -sh busybox.tar
2.7M busybox.tar
$ sudo docker save -o fedora-all.tar fedora
$ sudo docker save -o fedora-latest.tar fedora:latest
```

It is even useful to cherry-pick particular tags of an image repository

$ sudo docker save -o ubuntu.tar ubuntu:lucid ubuntu:saucy

search

Search Docker Hub (https://hub.docker.com) for images

```
Usage: docker search [OPTIONS] TERM

Search the Docker Hub for images

  --automated=false    Only show automated builds
  --no-trunc=false     Don't truncate output
  -s, --stars=0        Only displays with at least x stars
```

See *Find Public Images on Docker Hub* (/userguide/dockerrepos/#searching-for-images) for more details on finding shared images from the command line.

> ***Note:*** *Search queries will only return up to 25 results*

start

```
Usage: docker start [OPTIONS] CONTAINER [CONTAINER...]

Restart a stopped container

  -a, --attach=false         Attach container's STDOUT and STDERR and forward all signals to the process
  -i, --interactive=false    Attach container's STDIN
```

When run on a container that has already been started, takes no action and succeeds unconditionally.

stop

```
Usage: docker stop [OPTIONS] CONTAINER [CONTAINER...]

Stop a running container by sending SIGTERM and then SIGKILL after a grace period

  -t, --time=10      Number of seconds to wait for the container to stop before killing it. Default is 10 seconds.
```

The main process inside the container will receive `SIGTERM`, and after a grace period, `SIGKILL`.

tag

```
Usage: docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG]

Tag an image into a repository

  -f, --force=false    Force
```

You can group your images together using names and tags, and then upload them to *Share Images via Repositories* (/userguide/dockerrepos/#contributing-to-docker-hub).

top

```
Usage: docker top CONTAINER [ps OPTIONS]

Display the running processes of a container
```

unpause

```
Usage: docker unpause CONTAINER

Unpause all processes within a container
```

The `docker unpause` command uses the cgroups freezer to un-suspend all processes in a container.

See the cgroups freezer documentation (https://www.kernel.org/doc/Documentation/cgroups/freezer-subsystem.txt) for further details.

version

```
Usage: docker version

Show the Docker version information.
```

Show the Docker version, API version, Git commit, and Go version of both Docker client and daemon.

wait

```
Usage: docker wait CONTAINER [CONTAINER...]

Block until a container stops, then print its exit code.
```