

Where are Docker images stored?

WRITTEN BY TROY HOWARD

If you're just starting out with [Docker](#), it's super easy to follow the examples, get started and run a few things. However, moving to the next step, making your own Dockerfiles, can be a bit confusing. One of the more common points of confusion seems to be:

Where are my Docker images stored?

- Everyone

I know this certainly left me scratching my head a bit. Even worse, as a n00b, the last thing you want to do is publish your tinkering on the [public Docker Index](#).

Checkout my awesome new Docker image

`thoward/i_have_no_idea_what_im_doing`.

Yeah. Not really what I want to do.

Even worse, for quite a while there was no way to delete something that you had published, so your shamefully awkward learning process was up there for good. Luckily, deleting published Docker repositories is now quite easy.

So let me start with this small assurance; *Nothing you do will become public* especially if:

1. You haven't made an account on the public index.
2. You haven't run `docker login` to authenticate via the command-line client.

3. You don't run `docker push`, to push an image up to the index.

Vocabulary

One of the things that contributes to much of the confusion around Docker is the language that's used. There's a lot of terminology which seem to overlap, or is a bit ambiguous, used somewhat incorrectly, or has a well-established meaning that is different from how Docker uses it.

I'll try to clear those up here, in a quick vocabulary lesson.

Image vs Dockerfile

This one is the least confusing, but it's an important distinction. Docker uses *images* to run your code, not the *Dockerfile*. The *Dockerfile* is used to build the image when you run `docker build`.

If you go browsing around on the Docker Index, you'll see lots of images listed there, but weirdly, you can't see the *Dockerfile* that built them. The image is an *opaque* asset that is compiled from the *Dockerfile*.

When you run `docker push` to publish an image, it's not publishing your source code, it's publishing the image that was built from your source code.

Registry vs Index

The next weird thing is the idea of a *Registry* and an *Index*, and how these are separate things.

An *index* manages user accounts, permissions, search, tagging, and all that nice stuff that's in the public web interface.

A *registry* stores and serves up the actual image assets, and it delegates authentication to the *index*.

When you run `docker search`, it's searching the *index*, not the *registry*. In fact, it might be searching *multiple registries* that the index is aware of.

When you run `docker push` or `docker pull`, the *index* determines if you are allowed to access or modify the image, but the *registry* is the piece that stores it or sends it down the wire to you after the *index* approves the operation. Also, the *index* figures out which *registry* that particular image lives in and forwards the request appropriately.

Beyond that, when you're working locally and running commands like `docker images`, you're interacting with something that is neither an index or a registry, but a little of both.

Repository

Docker's use of this word is similar to its use at [Github](#), and other source control systems, but also, kind of not.

Three common head-scratching questions are:

- What's the difference between a repository and a registry?
- What's the difference between a repository and an image?
- What's the difference between a repository and an index username?

In fact, this is a problem, because a *repository* is all of those things and not really any of them either. Further, when you run `docker images` you get output like this:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
ubuntu              12.04              8dbd9e392a96       8 months ago
ubuntu              latest             8dbd9e392a96       8 months ago
ubuntu              precise            8dbd9e392a96       8 months ago
ubuntu              12.10              b750fe79269d       8 months ago
ubuntu              quantal            b750fe79269d       8 months ago
```

So, the list of *images* seems to be a list of repositories? Huh? Actually the *images* are the *CHIDs*, but that's not how you interact with them

images are the GUIDs, but that's not how you interact with them.

Let's start over with this.

When you run `docker build` or `docker commit`, you can specify a name for the *image*. The name is usually in the format of `username/image_name`, but it doesn't have to be. It could be anything, and it could even be the same as something well known and published.

However, when the time comes to `docker push`, the *index* will look at the name, and will check to see if it has a matching *repository*. If it does, it will check to see if you have access to that *repository*, and if so, allow you to push the new version of the *image* to it. So, a *registry* holds a collection of named *repositories*, which themselves are a collection of *images* tracked by GUIDs. This is also where *tags* come in. You can tag an image, and store multiple versions of that image with different GUIDs in a single named *repository*, access different tagged versions of an image with a special syntax like `username/image_name:tag`.

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
ubuntu	12.04	8dbd9e392a96	8 months ago
ubuntu	latest	8dbd9e392a96	8 months ago
ubuntu	precise	8dbd9e392a96	8 months ago
ubuntu	12.10	b750fe79269d	8 months ago
ubuntu	quantal	b750fe79269d	8 months ago

If we look at the output from `docker images` again, now it makes a little more sense. We have five different versions of the image named `ubuntu`, each one tagged slightly differently. The repository holds all of those under that name `ubuntu`. So, while it may seem like `ubuntu` is an *image name*, it's actually a *repository name*, indicating where it came from, or where it should go during a push.

Further, the repository name has a specific schema to it. An *index* can parse out the username from first part, and figure out where it is.

So, this the confusing part: Suppose there's a Docker image called `thoward/scooby_snacks`.

The official "repository name" is `thoward/scooby_snacks`, even though we

would normally think of the repository as just being `scooby_snacks` (eg, in GitHub, or elsewhere).

In fact, when the Docker documentation refers to a *repository*, it sometimes means the whole thing, username included, and sometimes only means the part after the username.

That's because some repositories don't have usernames (like `ubuntu`). The username is very important to handle separately, because it's used for authentication by the *index*, so that part of the repository name has its own semantics separate from the name, when it's there.

Local Storage on the Docker Host

So far I've been explaining the intricacies of remote storage, and how that relates to the confusing vocabulary, but running `docker images` shows you only what is local to your machine.

Where is this stuff? The first place to look is in `/var/lib/docker/`.

Open up the file `repositories` to find a JSON list of the repositories on your host:

```
$ sudo cat /var/lib/docker/repositories | python -mjson.tool
{
  "Repositories": {
    "ubuntu": {
      "12.04": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bb",
      "12.10": "b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589f",
      "latest": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1b",
      "precise": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1",
      "quantal": "b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d58"
    }
  }
}
```

Hey, that matches the output from `docker images` !

REPOSITORY	TAG	IMAGE ID	CREATED
ubuntu	12.04	8dbd9e392a96	8 months ago
ubuntu	latest	8dbd9e392a96	8 months ago
ubuntu	precise	8dbd9e392a96	8 months ago
ubuntu	12.10	b750fe79269d	8 months ago
ubuntu	quantal	b750fe79269d	8 months ago

Checkout what's in `/var/lib/docker/graph/`:

```
$ sudo ls -al /var/lib/docker/graph
total 24
drwx----- 6 root root 4096 Nov 22 06:52 .
drwx----- 5 root root 4096 Dec 13 04:25 ..
drwxr-xr-x 3 root root 4096 Dec 13 04:26 27cf784147099545
drwxr-xr-x 3 root root 4096 Nov 22 06:52 8dbd9e392a964056420e5d58ca5cc376ef18e
drwxr-xr-x 3 root root 4096 Nov 22 06:52 b750fe79269d2ec9a3c593ef05b4332b1d1a0
drwx----- 3 root root 4096 Nov 22 06:52 _tmp
```

Not terribly friendly, but we can see how Docker is keeping track of these, based on the `repositories` JSON file which holds a mapping of repository names and tags, to the underlying image GUIDs.

We have two images from the `ubuntu` repository, with the tags **12.04**, **precise**, and **latest** all corresponding to the image with id `8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c` (or `8dbd9e392a96` for short).

So what's actually stored there?

```
$ sudo ls -al /var/lib/docker/graph/8dbd9e392a964056420e5d58ca5cc376ef18e2de93
total 20
drwxr-xr-x 3 root root 4096 Nov 22 06:52 .
drwx----- 6 root root 4096 Nov 22 06:52 ..
-rw----- 1 root root 437 Nov 22 06:51 json
drwxr-xr-x 22 root root 4096 Apr 11 2013 layer
-rw----- 1 root root 9 Nov 22 06:52 layersize
```

The entries here are:

- `json` - holds metadata about the image
- `layersize` - just a number, indicating the size of the layer
- `layer/` - sub-directory that holds the rootfs for the container image

```
$ sudo cat /var/lib/docker/graph/8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5c
{
  "comment": "Imported from -",
  "container_config": {
    "AttachStderr": false,
    "AttachStdin": false,
```

```

    "AttachStdout": false,
    "Cmd": null,
    "Env": null,
    "Hostname": "",
    "Image": "",
    "Memory": 0,
    "MemorySwap": 0,
    "OpenStdin": false,
    "PortSpecs": null,
    "StdinOnce": false,
    "Tty": false,
    "User": ""
  },
  "created": "2013-04-11T14:13:15.57812-07:00",
  "docker_version": "0.1.4",
  "id": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c"
}

```

```

$ sudo cat /var/lib/docker/graph/8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c
131301903

```

```

$ sudo ls -al /var/lib/docker/graph/8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c
total 88
drwxr-xr-x 22 root root 4096 Apr 11 2013 .
drwxr-xr-x 3 root root 4096 Nov 22 06:52 ..
drwxr-xr-x 2 root root 4096 Apr 11 2013 bin
drwxr-xr-x 2 root root 4096 Apr 19 2012 boot
drwxr-xr-x 4 root root 4096 Nov 22 06:51 dev
drwxr-xr-x 41 root root 4096 Nov 22 06:51 etc
drwxr-xr-x 2 root root 4096 Apr 19 2012 home
drwxr-xr-x 11 root root 4096 Nov 22 06:51 lib
drwxr-xr-x 2 root root 4096 Nov 22 06:51 lib64
drwxr-xr-x 2 root root 4096 Apr 11 2013 media
drwxr-xr-x 2 root root 4096 Apr 19 2012 mnt
drwxr-xr-x 2 root root 4096 Apr 11 2013 opt
drwxr-xr-x 2 root root 4096 Apr 19 2012 proc
drwx----- 2 root root 4096 Nov 22 06:51 root
drwxr-xr-x 4 root root 4096 Nov 22 06:51 run
drwxr-xr-x 2 root root 4096 Nov 22 06:51/sbin
drwxr-xr-x 2 root root 4096 Mar 5 2012 selinux
drwxr-xr-x 2 root root 4096 Apr 11 2013 srv
drwxr-xr-x 2 root root 4096 Apr 14 2012 sys
drwxrwxrwt 2 root root 4096 Apr 11 2013 tmp
drwxr-xr-x 10 root root 4096 Nov 22 06:51 usr
drwxr-xr-x 11 root root 4096 Nov 22 06:51 var

```

Pretty easy. This is the magic behind being able to refer to an image by its repository name, even if you're not interacting with the remote Docker Index or Docker Registry. Once you've pulled it down to your workstation, Docker can work with it by name using these files. This is also where things go when you're developing a new Dockerfile.

DIY Dockerfiles

Let's try an example. Make a `Dockerfile` with the following contents:

```
FROM ubuntu
```

This basically doesn't do anything except say that we're including the `ubuntu` image as our base layer, but that's enough to get started.

Next, run `docker build -t scooby_snacks .`. What that will do is look in the directory we specified (`.`) for a file called `Dockerfile` and then build it, and use the name `scooby_snacks` for the repository.

```
$ docker build -t scooby_snacks .
Uploading context 64184320 bytes
Step 1 : FROM ubuntu
----> 8dbd9e392a96
Successfully built 8dbd9e392a96
```

Oh no! It said "Uploading context"... Did we just upload it to the public registry?

Let's check:

```
$ docker search scooby_snacks
```

NAME	DESCRIPTION	STARS	OFFICIAL	TRUSTED
------	-------------	-------	----------	---------

Whew! Not there. So why did it say that?

I have no idea, but you can ignore it. Where did it really end up?

```
$ sudo cat /var/lib/docker/repositories | python -mjson.tool
{
  "Repositories": {
    "scooby_snacks": {
      "latest": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1b"
    },
    "ubuntu": {
      "12.04": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bb"
      "12.10": "b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589f"
      "latest": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1b"
      "precise": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1"
      "quantal": "b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d58"
    }
  }
}
```


Well, looks like Docker just "uploaded" it to `/var/lib/docker` . :)

It should also show up in `docker images` .

```
$ docker images
REPOSITORY          TAG             IMAGE ID        CREATED
ubuntu              12.04          8dbd9e392a96   8 months ago
ubuntu              latest         8dbd9e392a96   8 months ago
ubuntu              precise        8dbd9e392a96   8 months ago
scooby_snacks       latest         8dbd9e392a96   8 months ago
ubuntu              12.10         b750fe79269d   8 months ago
ubuntu              quantal        b750fe79269d   8 months ago
```

There it is! Docker was smart enough to realize that we didn't change anything, so it kept the same image id, and didn't bother copying the `ubuntu` image. Pretty sweet.

Next, we'll make a small change so that Docker will have to build a new layer.

Edit `Dockerfile` to have these contents:

```
FROM ubuntu

RUN touch scooby_snacks.txt
```

Then run `docker build -t scooby_snacks .` to rebuild.

```
$ docker build -t scooby_snacks .
Uploading context 64184320 bytes
Step 1 : FROM ubuntu
--> 8dbd9e392a96
Step 2 : RUN touch scooby_snacks.txt
--> Running in 86664242766c
--> 91acef3a5936
Successfully built 91acef3a5936
```

There should be a new directory under `/var/lib/docker/graph`

```
$ sudo ls -al /var/lib/docker/graph
total 28
drwx----- 7 root root 4096 Dec 13 06:27 .
drwx----- 5 root root 4096 Dec 13 06:27 ..
drwxr-xr-x 3 root root 4096 Dec 13 04:26 27cf784147099545
```

```
drwxr-xr-x 3 root root 4096 Nov 22 06:52 8dbd9e392a964056420e5d58ca5cc376ef18e
drwxr-xr-x 3 root root 4096 Dec 13 06:27 91acef3a5936769f763729529e736681e5079
drwxr-xr-x 3 root root 4096 Nov 22 06:52 b750fe79269d2ec9a3c593ef05b4332b1d1a0
drwx----- 3 root root 4096 Dec 13 06:27 _tmp
```

Docker gave it a new image ID:

- 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9

It has also been updated in `/var/lib/docker/repositories` and `docker images`.

```
$ sudo cat /var/lib/docker/repositories | python -mjson.tool
{
  "Repositories": {
    "scooby_snacks": {
      "latest": "91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a
    },
    "ubuntu": {
      "12.04": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bb
      "12.10": "b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589f
      "latest": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1b
      "precise": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1
      "quantal": "b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d58
    }
  }
}
$ docker images
REPOSITORY          TAG                IMAGE ID           CREATED
scooby_snacks       latest            91acef3a5936      5 minutes ago
ubuntu              12.04            8dbd9e392a96      8 months ago
ubuntu              latest           8dbd9e392a96      8 months ago
ubuntu              precise          8dbd9e392a96      8 months ago
ubuntu              12.10            b750fe79269d      8 months ago
ubuntu              quantal           b750fe79269d      8 months ago
```

Let's see what

`/var/lib/docker/graph/91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9` looks like now:

```
$ sudo cat /var/lib/docker/graph/91acef3a5936769f763729529e736681e5079dc6ddf6a
{
  "Size": 0,
  "architecture": "x86_64",
  "config": {
    "AttachStderr": false,
    "AttachStdin": false,
    "AttachStdout": false,
    "Cmd": null,
    "CpuShares": 0,
    "Dns": null,
    "Domainname": "",
    "Entrypoint": [],
    "Env": [
      "HOME=/"
    ]
  }
}
```

```

    "PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
  ],
  "ExposedPorts": {},
  "Hostname": "86664242766c",
  "Image": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc831",
  "Memory": 0,
  "MemorySwap": 0,
  "NetworkDisabled": false,
  "OpenStdin": false,
  "PortSpecs": null,
  "StdinOnce": false,
  "Tty": false,
  "User": "",
  "Volumes": {},
  "VolumesFrom": "",
  "WorkingDir": ""
},
"container": "86664242766c5548f8118716e873835c171811176a710e425c1fcf1fa36",
"container_config": {
  "AttachStderr": false,
  "AttachStdin": false,
  "AttachStdout": false,
  "Cmd": [
    "/bin/sh",
    "-c",
    "touch scooby_snacks.txt"
  ],
  "CpuShares": 0,
  "Dns": null,
  "Domainname": "",
  "Entrypoint": [],
  "Env": [
    "HOME=/",
    "PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
  ],
  "ExposedPorts": {},
  "Hostname": "86664242766c",
  "Image": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc831",
  "Memory": 0,
  "MemorySwap": 0,
  "NetworkDisabled": false,
  "OpenStdin": false,
  "PortSpecs": null,
  "StdinOnce": false,
  "Tty": false,
  "User": "",
  "Volumes": {},
  "VolumesFrom": "",
  "WorkingDir": ""
},
"created": "2013-12-13T06:27:03.234029255Z",
"docker_version": "0.6.7",
"id": "91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9",
"parent": "8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1"
}

```

```

$ sudo cat /var/lib/docker/graph/91acef3a5936769f763729529e736681e5079dc6ddf6a
12288

```

```

$ sudo ls -al /var/lib/docker/graph/91acef3a5936769f763729529e736681e5079dc6dd
total 16
drwxr-xr-x 4 root root 4096 Dec 13 06:27 .
drwxr-xr-x 3 root root 4096 Dec 13 06:27 ..

```

```
-rw-r--r-- 1 root root    0 Dec 13 06:27 scooby_snacks.txt
-r--r--r-- 1 root root    0 Dec 13 06:27 .wh..wh.aufs
drwx----- 2 root root 4096 Dec 13 06:27 .wh..wh.orph
drwx----- 2 root root 4096 Dec 13 06:27 .wh..wh.plnk
```

Our tiny change had a big impact! Notice that Docker only kept the *differences* from the base image. This is the key to the *layer* concept.

Run it!

We can now run our new image and try it out. We'll just run an interactive `bash` prompt for now.

```
$ docker run -i -t scooby_snacks /bin/bash
root@1f8602a7d589:/# ls -al
total 12308
drwxr-xr-x 30 root root    4096 Dec 13 06:43 .
drwxr-xr-x 30 root root    4096 Dec 13 06:43 ..
-rw----- 1 root root     208 Dec 13 06:43 .dockerenv
-rwxr-xr-x 1 root root 12516574 Nov 22 02:34 .dockerinit
drwxr-xr-x 2 root root    4096 Apr 11 2013 bin
drwxr-xr-x 2 root root    4096 Apr 19 2012 boot
drwxr-xr-x 6 root root    4096 Nov 22 06:52 dev
drwxr-xr-x 41 root root    4096 Nov 22 06:52 etc
drwxr-xr-x 2 root root    4096 Apr 19 2012 home
drwxr-xr-x 11 root root    4096 Nov 22 06:51 lib
drwxr-xr-x 2 root root    4096 Nov 22 06:51 lib64
drwxr-xr-x 2 root root    4096 Apr 11 2013 media
drwxr-xr-x 2 root root    4096 Apr 19 2012 mnt
drwxr-xr-x 2 root root    4096 Apr 11 2013 opt
dr-xr-xr-x 102 root root      0 Dec 13 06:43 proc
drwx----- 2 root root    4096 Nov 22 06:51 root
drwxr-xr-x 4 root root    4096 Nov 22 06:51 run
drwxr-xr-x 2 root root    4096 Nov 22 06:51 sbin
-rw-r--r-- 1 root root      0 Dec 13 06:27 scooby_snacks.txt
drwxr-xr-x 2 root root    4096 Mar  5 2012 selinux
drwxr-xr-x 2 root root    4096 Apr 11 2013 srv
dr-xr-xr-x 13 root root      0 Dec 13 06:43 sys
drwxrwxrwt 2 root root    4096 Apr 11 2013 tmp
drwxr-xr-x 10 root root    4096 Nov 22 06:51 usr
drwxr-xr-x 11 root root    4096 Nov 22 06:51 var
root@1f8602a7d589:/#
```

The effects of our `RUN touch scooby_snacks.txt` command in the `Dockerfile` are exactly as expected.

Publish it!

Until now, we've been doing everything locally and not interacting with the outside world at all. This is great, we can work up a perfect

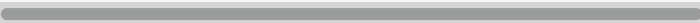
`Dockerfile` before we go live. That said, I'm pretty happy with this one now, and I'm ready to publish it.

If you haven't already, make sure you [make an account](#), and then login with `docker login`.

```
$ docker login
Username: thoward
Password:
Email: thoward37@gmail.com
Login Succeeded
```

Publish the image with `docker push scooby_snacks`

```
$ docker push scooby_snacks
2013/12/13 06:49:36 Impossible to push a "root" repository. Please rename your
```



Oops. Docker Index won't let us publish without our username in the repository name. No big deal.

Rebuild this with the correct username using

`docker build -t thoward/scooby_snacks .`

```
$ docker build -t thoward/scooby_snacks .
Uploading context 64184320 bytes
Step 1 : FROM ubuntu
----> 8dbd9e392a96
Step 2 : RUN touch scooby_snacks.txt
----> Using cache
----> 91acef3a5936
Successfully built 91acef3a5936
```

Nice! The message "*Using cache*" means Docker was smart enough to know that we didn't really change the image, so it didn't bother rebuilding it.

Let's try publishing again, but this time with the correct repository name:

```
$ docker push thoward/scooby_snacks
The push refers to a repository [thoward/scooby_snacks] (len: 1)
Sending image list
Pushing repository thoward/scooby_snacks (1 tags)
```

```
Pushing 8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c
Image 8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c already
Pushing tags for rev [8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bb
Pushing 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9
```

```
Pushing tags for rev [91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a9
```

Now that it's published, it should show up with **docker search**.

```
$ docker search scooby_snacks
NAME                DESCRIPTION          STARS   OFFICIAL   TRUSTED
thoward/scooby_snacks          0
```

There it is. Next, let's cleanup a bit and delete the old root level one:

```
$ docker images
REPOSITORY          TAG                IMAGE ID           CREATED
scooby_snacks       latest            91acef3a5936       30 minutes ago
thoward/scooby_snacks latest            91acef3a5936       30 minutes ago
ubuntu              12.04            8dbd9e392a96       8 months ago
ubuntu              latest           8dbd9e392a96       8 months ago
ubuntu              precise          8dbd9e392a96       8 months ago
ubuntu              12.10            b750fe79269d       8 months ago
ubuntu              quantal          b750fe79269d       8 months ago
```

```
$ docker rmi scooby_snacks
Untagged: 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9
```

```
$ docker images
REPOSITORY          TAG                IMAGE ID           CREATED
thoward/scooby_snacks latest            91acef3a5936       29 minutes ago
ubuntu              12.04            8dbd9e392a96       8 months ago
ubuntu              latest           8dbd9e392a96       8 months ago
ubuntu              precise          8dbd9e392a96       8 months ago
ubuntu              12.10            b750fe79269d       8 months ago
ubuntu              quantal          b750fe79269d       8 months ago
```

Ok, that one is gone.

Also, to be honest, this is not a very interesting image to share publicly, and we don't want to look like n00bs, so let's delete it as well.

```
$ docker rmi thoward/scooby_snacks
Untagged: 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9
Deleted: 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9
```

```
$ docker images
REPOSITORY          TAG                IMAGE ID           CREATED
ubuntu              12.04            8dbd9e392a96       8 months ago
ubuntu              latest           8dbd9e392a96       8 months ago
ubuntu              precise          8dbd9e392a96       8 months ago
ubuntu              12.10            b750fe79269d       8 months ago
```

This time, since Docker realized it was the last reference to that image ID, `docker rmi` has an additional message indicating that it deleted it instead of just 'untagging' it.

But wait! It is still public at the Docker Index, isn't it? Let's check:

```
$ docker search scooby_snacks
NAME                DESCRIPTION          STARS     OFFICIAL   TRUSTED
thoward/scooby_snacks          0
```

Hmm.. Well this is handy, before we delete it, we can try `docker pull` and fetch it down like a "real" image and run it.

```
$ docker pull thoward/scooby_snacks
Pulling repository thoward/scooby_snacks
91acef3a5936: Download complete
8dbd9e392a96: Download complete

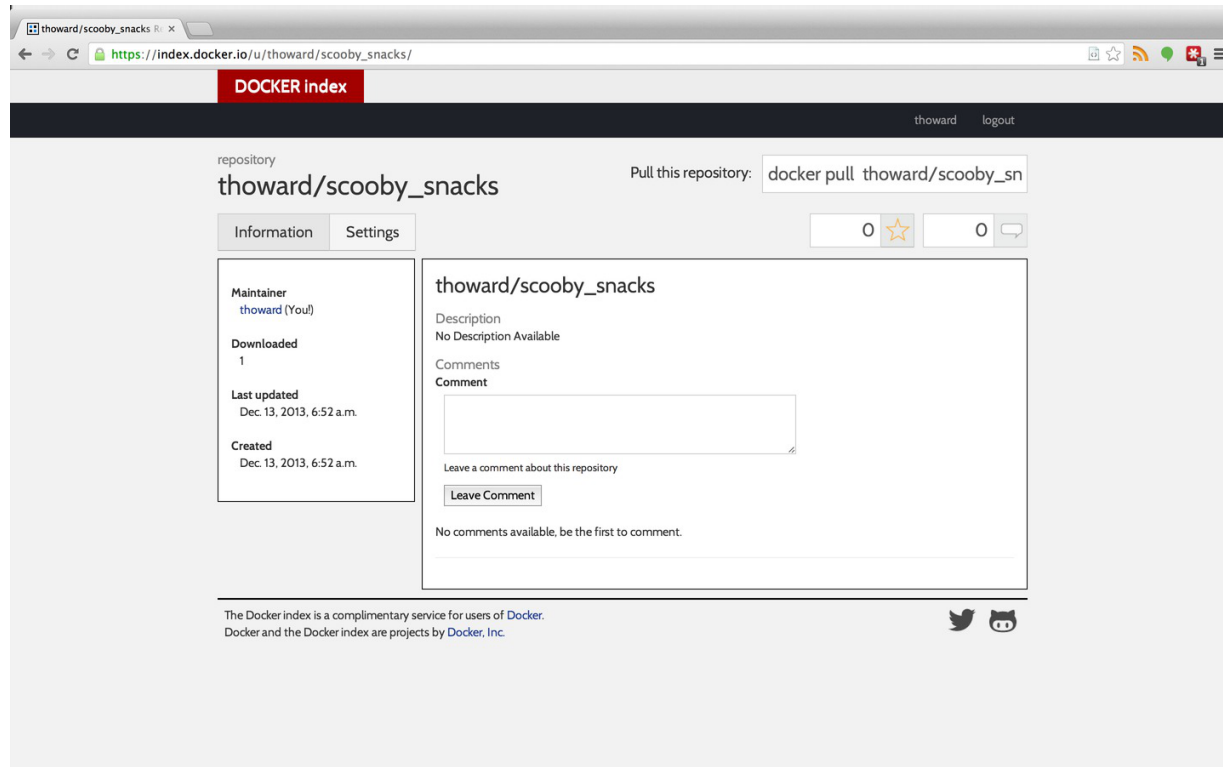
$ docker run -i -t thoward/scooby_snacks /bin/bash
root@90f6546bf3b7:/# ls -al
total 12308
drwxr-xr-x 30 root root      4096 Dec 13 07:03 .
drwxr-xr-x 30 root root      4096 Dec 13 07:03 ..
-rw----- 1 root root        208 Dec 13 07:03 .dockerenv
-rwxr-xr-x 1 root root 12516574 Nov 22 02:34 .dockerinit
drwxr-xr-x 2 root root      4096 Apr 11 2013 bin
drwxr-xr-x 2 root root      4096 Apr 19 2012 boot
drwxr-xr-x 6 root root      4096 Nov 22 06:52 dev
drwxr-xr-x 41 root root      4096 Nov 22 06:52 etc
drwxr-xr-x 2 root root      4096 Apr 19 2012 home
drwxr-xr-x 11 root root      4096 Nov 22 06:51 lib
drwxr-xr-x 2 root root      4096 Nov 22 06:51 lib64
drwxr-xr-x 2 root root      4096 Apr 11 2013 media
drwxr-xr-x 2 root root      4096 Apr 19 2012 mnt
drwxr-xr-x 2 root root      4096 Apr 11 2013 opt
dr-xr-xr-x 105 root root        0 Dec 13 07:03 proc
drwx----- 2 root root      4096 Nov 22 06:51 root
drwxr-xr-x 4 root root      4096 Nov 22 06:51 run
drwxr-xr-x 2 root root      4096 Nov 22 06:51 sbin
-rw-r--r-- 1 root root        0 Dec 13 06:27 scooby_snacks.txt
drwxr-xr-x 2 root root      4096 Mar  5 2012 selinux
drwxr-xr-x 2 root root      4096 Apr 11 2013 srv
dr-xr-xr-x 13 root root        0 Dec 13 07:03 sys
drwxrwxrwt 2 root root      4096 Apr 11 2013 tmp
drwxr-xr-x 10 root root      4096 Nov 22 06:51 usr
drwxr-xr-x 11 root root      4096 Nov 22 06:51 var
root@90f6546bf3b7:/#
```

It works!

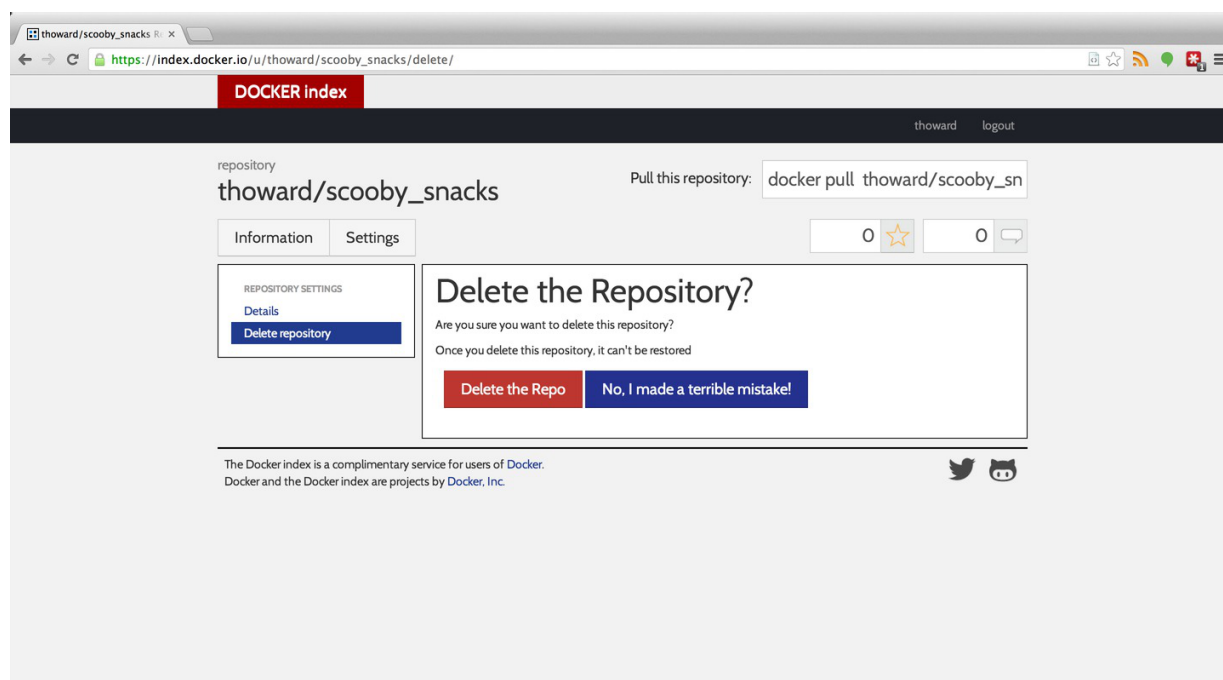
Deleting a Published Repository

Unfortunately, to delete it from the public index/registry, we have to use the web interface, not the command-line.

First, [login via the web](#) then navigate to the repository at https://index.docker.io/u/thoward/scooby_snacks/.



Click on 'Settings' tab, then 'Delete Repository' tab, then the 'Delete Repo' button.



Back on the command-line we can verify it's gone with

```
docker search scooby_snacks
```

```
$ docker search scooby_snacks
NAME                DESCRIPTION    STARS     OFFICIAL   TRUSTED
```

But of course, since we never deleted the local version of it after we pulled it back down, it's still going to show up in `docker images`, since we have a local copy:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
thoward/scooby_snacks latest             91acef3a5936       46 minutes ago
ubuntu              12.04              8dbd9e392a96       8 months ago
ubuntu              latest             8dbd9e392a96       8 months ago
ubuntu              precise            8dbd9e392a96       8 months ago
ubuntu              12.10              b750fe79269d       8 months ago
ubuntu              quantal            b750fe79269d       8 months ago
```

So to completely remove it we need to run `docker rmi` again.

```
$ docker rmi thoward/scooby_snacks
Untagged: 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9
Deleted: 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9
```

Not to worry, we can always rebuild it with our `Dockerfile`. :)

Important Security Lesson

It's really important to consider the security implications of what we just saw though.

Even if a Docker image is deleted from the Docker Index *it may still be out there on someones machine*. There's no way to change that.

Also, as we saw when looking at the files we have locally, it's not quite an "opaque binary" image. All the information from the Dockerfile was in

the JSON file for the image, and the artifacts of those commands are in the layer, as accessible as a filesystem. If you accidentally published a password or key, or some other critical secret, there's no getting it back, and people can find as easily as they can find anything else in a published open source code base.

Be very careful about what you're publishing. If you do accidentally publish a secret, take it down right away and update credentials on whatever systems it might have compromised.

Conclusion

Docker can be a bit confusing with its terminology, but once you wrap your head around the basic workflow described here, it should be very easy to be in-control of what you're building, knowing exactly when and how you share that with the world.

« [Full blog](#)



Hi, I'm [Troy Howard](#) and this is my blog.

Want to chat? Come find me on [Twitter](#), or [GitHub](#).