Docker

Docker

- In this section we will spend time looking at more Docker functionality

Docker Commands

- You can see a list of Docker commands via:

  docker --help

Docker Commands

- Docker's help has:
  - Options
  - Management Commands
  - Commands

- The following slides summarize some of Docker's options, management commands, and commands

Docker help (partial)

```
JeffsMacBookPro:py01 jeffm$ docker --help

Usage:  docker COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/Users/jeffm/.docker")
  -D, --debug              Enable debug mode
      --help               Print usage
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/Users/jeffm/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/Users/jeffm/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/Users/jeffm/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit
```

Docker Options

- -- config string

  by default Docker looks for a client configuration file named .docker in your home directory . This setting can specify a different location. NOTE: you may not find a .docker file.

- -D or −debug

  enable Debug mode

Docker Options

- -l or −log-level

  debug, info, warn, error, fatal

- -v or --version

Docker Management Commands

- The following slides will present a few of the Docker management commands

```
Management Commands:
  checkpoint  Manage checkpoints
  config      Manage Docker configs
  container   Manage containers
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  volume      Manage volumes
```

Docker Management Commands

```
JeffsMacBookPro:docker jeffm$ docker checkpoint --help

Usage:  docker checkpoint COMMAND

Manage checkpoints

Options:
      --help   Print usage

Commands:
  create      Create a checkpoint from a running container
  ls          List checkpoints for a container
  rm          Remove a checkpoint
```

A checkpoint tales a "snapshot" of a running process, our running Docker image in this case, and saves it.

You can start a docker container using a checkpoint (see docker start −help)

Docker Management Commands

```
JeffsMacBookPro:docker jeffm$ docker container

Usage:  docker container COMMAND

Manage containers

Options:
```

The following slides will demonstrate example usage of the following docker container commands:

- ls – list containers
- attach – attach local streams into a running container
- exec – execute a command in a running container
- run – run a command in a new container
- export/import – export/import a container's file system as a tar archive

Docker Management Commands

```
JeffsMacBookPro:docker jeffm$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
2a719a0cc18a        8ae6f52035b5        "/bin/sh"           12 hours ago
```

Your instructor has done a lot of work with Docker - why is there only 1 container? Let's look at our Docker images before answering this.

```
JeffsMacBookPro:docker jeffm$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
buildpack-deps-jessie   py01                b24d23f1dfa7        13 hours ago        673MB
cassandra               latest              c82d9de5d478        3 days ago          386MB
mongo                   latest              71c101e16e61        10 days ago         358MB
ubuntu                  latest              d355ed3537e9        12 days ago         119MB
awslinux-saved01        latest              228ca7fc9dd1        8 weeks ago         901MB
awslinux                latest              228ca7fc9dd1        8 weeks ago         901MB
mongo                   ok                  ad974e767ec4        4 months ago        402MB
ubuntu                  <none>              f49eec89601e        5 months ago        129MB
amazonlinux             issetup             8ae6f52035b5        6 months ago        292MB
hello-world             latest              c54a2cc56cbb        12 months ago       1.85kB
alpine                  latest              13e1761bf172        14 months ago       4.8MB
```

Let's review Docker containers and images

Docker Management Commands

- A docker image is the specification of what you will run.
- A docker container runs a Docker image and contains runtime changes made within the running image.
- Different Docker containers can run the same Docker Image

Docker Commands Examples

- To run the examples on the following slides (run, attach, exec, and export/ import,cp,commit) – lets run our py01 Docker image from the previous section.

Docker Commands: cmd01

- The following script:
  - gets the ID of Docker image py01
  - runs image py01
  - uses the run command to run df

Docker Commands: cmd01

```
#!/bin/sh
dockerimage=$(docker images | grep py01 | awk '$1=$1')
echo image: $dockerimage
dockerimageid=$(echo $dockerimage | awk '{print $3}')
echo imageid: $dockerimageid
docker run $dockerimageid df
```

```
JeffsMacBookPro:py01 jeffm$ cmd01.sh
image: buildpack-deps-jessie py01 b24d23f1dfa7 14 hours ago 673MB
imageid: b24d23f1dfa7
Filesystem     1K-blocks    Used Available Use% Mounted on
none            61889524 7113716  51608936  13% /
tmpfs            1023384       0   1023384   0% /dev
tmpfs            1023384       0   1023384   0% /sys/fs/cgroup
/dev/vda2       61889524 7113716  51608936  13% /etc/hosts
shm                65536       0     65536   0% /dev/shm
tmpfs            1023384       0   1023384   0% /sys/firmware
```

Docker Commands: cmd02

- In this example:
  - we run Docker image py01 in one terminal
  - in a second terminal we use the exec command to execute a command in a running Docker container

Docker Commands: cmd02

```
~/DevOps-Tech/docker/py01 — docker ◂ py01.sh
JeffsMacBookPro:py01 jeffm$ cat py01.sh
#!/bin/sh

docker run -it buildpack-deps-jessie:py01 /bin/sh
JeffsMacBookPro:py01 jeffm$ py01.sh
#
```

```
~/DevOps-Tech/docker/py01 — -bash
JeffsMacBookPro:py01 jeffm$ cmd02.sh
cid: b648b39c7379
total 72
drwxr-xr-x  56 root root 4096 Jul  8 07:33 .
drwxr-xr-x  56 root root 4096 Jul  8 07:33 ..
-rwxr-xr-x   1 root root    0 Jul  8 07:33 .dockerenv
drwxr-xr-x   2 root root 4096 Jun 20 21:09 bin
drwxr-xr-x   2 root root 4096 Apr 20 21:43 boot
drwxr-xr-x   5 root root  360 Jul  8 07:33 dev
drwxr-xr-x  62 root root 4096 Jul  8 07:33 etc
drwxr-xr-x   2 root root 4096 Apr 20 21:43 home
drwxr-xr-x  12 root root 4096 Jun 20 21:09 lib
drwxr-xr-x   2 root root 4096 Jun 20 00:00 lib64
drwxr-xr-x   2 root root 4096 Jun 20 00:00 media
drwxr-xr-x   2 root root 4096 Jun 20 00:00 mnt
drwxr-xr-x   2 root root 4096 Jun 20 00:00 opt
dr-xr-xr-x 134 root root    0 Jul  8 07:33 proc
drwx------   3 root root 4096 Jul  3 07:03 root
drwxr-xr-x   3 root root 4096 Jun 20 00:00 run
drwxr-xr-x   2 root root 4096 Jun 20 00:00 sbin
drwxr-xr-x   2 root root 4096 Jun 20 00:00 srv
dr-xr-xr-x  13 root root    0 Jul  8 07:28 sys
drwxrwxrwt   2 root root 4096 Jul  3 07:03 tmp
drwxr-xr-x  37 root root 4096 Jul  3 07:03 usr
drwxr-xr-x  26 root root 4096 Jul  3 07:01 var
```

```
#!/bin/sh
# cmd02.sh

containerid=$(docker ps | grep py01 | awk '{print $1}')
echo cid: $containerid

docker exec $containerid ls -la
```

Docker Commands: cmd02

```
JeffsMacBookPro:devops-docker jeffm$ docker ps
CONTAINER ID        IMAGE                        COMMAND             CREATED             STATUS              PORTS               NAMES
46d05539f6f2        buildpack-deps-jessie:py01   "/bin/sh"           6 minutes ago       Up 6 minutes                            suspicious_ritchie
```

Running "docker ps" while the py01 docker is running

Docker Commands: cmd03

```
JeffsMacBookPro:devops-docker jeffm$ docker attach --help

Usage:  docker attach [OPTIONS] CONTAINER

Attach local standard input, output, and error streams to a running container

Options:
      --detach-keys string   Override the key sequence for detaching a container
      --help                 Print usage
      --no-stdin             Do not attach STDIN
      --sig-proxy            Proxy all received signals to the process (default true)
```

Docker Commands: cmd03

```
JeffsMacBookPro:py01 jeffm$ cat py01.sh
#!/bin/sh

docker run -it buildpack-deps-jessie:py01 /bin/sh
JeffsMacBookPro:py01 jeffm$ py01.sh
#
```

```
JeffsMacBookPro:py01 jeffm$ cat cmd03.sh
#!/bin/sh

containerid=$(docker ps | grep py01 | awk '{print $1}')
echo cid: $containerid

docker attach --no-stdin $containerid
JeffsMacBookPro:py01 jeffm$ cmd03.sh
cid: 8739513dcee4
```

In cmd03.sh, we attached a terminal to a running Docker image, without attaching stdin.

The output of commands entered into the top terminal (bin/sh from running py01.sh) will be displayed in the bottom terminal (created via cmd03.sh). However, no input from the bottom terminal will be redirected to the top terminal since we did not attach stdin.

```
docker run -it buildpack-deps-jessie:py01 /bin/sh
JeffsMacBookPro:py01 jeffm$ py01.sh
# pwd
/
# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
#
```

```
echo cid: $containerid

docker attach --no-stdin $containerid
JeffsMacBookPro:py01 jeffm$ cmd03.sh
cid: 8739513dcee4
pwd
/
# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
#
```

Docker Commands: cmd04

- In this example we will use Docker export to create a tar file of a Docker
  - We will create a file in the running Docker container
  - We will export an image of the Docker container into a tar file
  - We will terminate the Docker container
  - We will import the tar file to create a Docker image

Docker Commands: cmd04

```
py01 — docker ◂ py01.sh — 80×29
~/DevOps-Tech/docker/py01 — docker ◂ py01.sh
JeffsMacBookPro:py01 jeffm$ py01.sh
# mkdir mytmp
# ls -la > mytmp/cats.txt
# cat mytmp/cats.txt
total 76
drwxr-xr-x  57 root root 4096 Jul  9 23:26 .
drwxr-xr-x  57 root root 4096 Jul  9 23:26 ..
-rwxr-xr-x   1 root root    0 Jul  9 23:26 .dockerenv
drwxr-xr-x   2 root root 4096 Jun 20 21:09 bin
drwxr-xr-x   2 root root 4096 Apr 20 21:43 boot
drwxr-xr-x   5 root root  360 Jul  9 23:26 dev
drwxr-xr-x  62 root root 4096 Jul  9 23:26 etc
drwxr-xr-x   2 root root 4096 Apr 20 21:43 home
drwxr-xr-x  12 root root 4096 Jun 20 21:09 lib
drwxr-xr-x   2 root root 4096 Jun 20 00:00 lib64
drwxr-xr-x   2 root root 4096 Jun 20 00:00 media
drwxr-xr-x   2 root root 4096 Jun 20 00:00 mnt
drwxr-xr-x   2 root root 4096 Jul  9 23:26 mytmp
drwxr-xr-x   2 root root 4096 Jun 20 00:00 opt
dr-xr-xr-x 136 root root    0 Jul  9 23:26 proc
drwx------   3 root root 4096 Jul  3 07:03 root
drwxr-xr-x   3 root root 4096 Jun 20 00:00 run
drwxr-xr-x   2 root root 4096 Jun 20 00:00 sbin
drwxr-xr-x   2 root root 4096 Jun 20 00:00 srv
dr-xr-xr-x  13 root root    0 Jul  9 22:15 sys
drwxrwxrwt   2 root root 4096 Jul  3 07:03 tmp
drwxr-xr-x  37 root root 4096 Jul  3 07:03 usr
drwxr-xr-x  26 root root 4096 Jul  3 07:01 var
#
```

start Docker image py01

when in the shell in py01:

- create a folder, mytmp
- put some dummy data in file cats.txt in folder mytmp

Docker Commands: cmd04

```
JeffsMacBookPro:py01 jeffm$ docker export --help

Usage:  docker export [OPTIONS] CONTAINER

Export a container's filesystem as a tar archive

Options:
      --help            Print usage
  -o, --output string   Write to a file, instead of STDOUT
JeffsMacBookPro:py01 jeffm$
```

Docker Commands: cmd04

```
JeffsMacBookPro:py01 jeffm$ cat cmd04.sh
#!/bin/sh

containerid=$(docker ps | grep py01 | awk '{print $1}')
echo cid: $containerid

docker export --output ./py01.tar $containerid
JeffsMacBookPro:py01 jeffm$ cmd04.sh
cid: facb1159e2a9
JeffsMacBookPro:py01 jeffm$ ls -la
total 1338464
drwxr-xr-x@ 9 jeffm  staff        306 Jul  9 16:31 .
drwxr-xr-x@ 8 jeffm  staff        272 Jul  9 15:27 ..
-rw-r--r--@ 1 jeffm  staff       2431 Jun 30 15:04 Dockerfile
-rwxr-xr-x@ 1 jeffm  staff        287 Jul  3 14:12 cmd01.sh
-rwxr-xr-x@ 1 jeffm  staff        122 Jul  6 18:45 cmd02.sh
-rwxr-xr-x@ 1 jeffm  staff        129 Jul  9 15:46 cmd03.sh
-rwxr-xr-x@ 1 jeffm  staff        138 Jul  9 16:30 cmd04.sh
-rwxr-xr-x@ 1 jeffm  staff         61 Jul  8 00:28 py01.sh
-rw-------@ 1 jeffm  staff  685265920 Jul  9 16:31 py01.tar
```

Docker Commands: cmd04

```
py01 — -bash — 145×21
~/DevOps-Tech/docker/py01 — -bash
JeffsMacBookPro:py01 jeffm$ docker ps
CONTAINER ID        IMAGE                          COMMAND             CREATED
facb1159e2a9        buildpack-deps-jessie:py01     "/bin/sh"           9 minutes ag
JeffsMacBookPro:py01 jeffm$
JeffsMacBookPro:py01 jeffm$ docker stop --help

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

Stop one or more running containers

Options:
      --help       Print usage
  -t, --time int   Seconds to wait for stop before killing it (default 10)
JeffsMacBookPro:py01 jeffm$
JeffsMacBookPro:py01 jeffm$ docker stop --time 2 facb1159e2a9
facb1159e2a9
JeffsMacBookPro:py01 jeffm$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             S
JeffsMacBookPro:py01 jeffm$
```

Run "docker ps" to get the container-id
Run "docker stop" to stop the container

Docker Commands: cmd04

```
~/DevOps-Tech/docker/py01 — docker ◂ py01.sh
JeffsMacBookPro:py01 jeffm$ py01.sh
# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
#
```

Run image py01 again, notice the mytmp directory is not there

Docker Commands: cmd04

```
~/DevOps-Tech/docker/py01 — docker ◂ py01.sh        ~/DevOps-Tech/devops-dock
JeffsMacBookPro:py01 jeffm$ docker export --help

Usage:  docker export [OPTIONS] CONTAINER

Export a container's filesystem as a tar archive

Options:
      --help             Print usage
  -o, --output string    Write to a file, instead of STDOUT
```

```
~/DevOps-Tech/docker/py01 — docker ◂ py01.sh        ~/DevOps-Tech/devops-d
JeffsMacBookPro:py01 jeffm$ ls  -la py01.tar
-rw-------@ 1 jeffm  staff  685265920 Jul  9 16:31 py01.tar
JeffsMacBookPro:py01 jeffm$
```

Docker Commands: cmd04,py02.sh

```
JeffsMacBookPro:py01 jeffm$ docker import ./py01.tar buildpack-deps-jessie:py02
sha256:558ca76dabc7f4682783638286233e4df488aa2fe5a5fa34eee0c29cf432e310
```

```
JeffsMacBookPro:py01 jeffm$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
buildpack-deps-jessie   py02                558ca76dabc7        3 minutes ago       665MB
```

```
JeffsMacBookPro:py01 jeffm$ cat py02.sh
#!/bin/sh

docker run -it buildpack-deps-jessie:py02 /bin/sh
```

Docker Commands: cmd04,py02.sh

```
JeffsMacBookPro:py01 jeffm$ py02.sh
# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  mytmp  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
# cat mytmp/cats.txt | more
total 76
drwxr-xr-x  57 root root 4096 Jul  9 23:26 .
drwxr-xr-x  57 root root 4096 Jul  9 23:26 ..
-rwxr-xr-x   1 root root    0 Jul  9 23:26 .dockerenv
drwxr-xr-x   2 root root 4096 Jun 20 21:09 bin
drwxr-xr-x   2 root root 4096 Apr 20 21:43 boot
drwxr-xr-x   5 root root  360 Jul  9 23:26 dev
drwxr-xr-x  62 root root 4096 Jul  9 23:26 etc
drwxr-xr-x   2 root root 4096 Apr 20 21:43 home
drwxr-xr-x  12 root root 4096 Jun 20 21:09 lib
drwxr-xr-x   2 root root 4096 Jun 20 00:00 lib64
drwxr-xr-x   2 root root 4096 Jun 20 00:00 media
drwxr-xr-x   2 root root 4096 Jun 20 00:00 mnt
drwxr-xr-x   2 root root 4096 Jul  9 23:26 mytmp
drwxr-xr-x   2 root root 4096 Jun 20 00:00 opt
dr-xr-xr-x 136 root root    0 Jul  9 23:26 proc
drwx------   3 root root 4096 Jul  3 07:03 root
drwxr-xr-x   3 root root 4096 Jun 20 00:00 run
drwxr-xr-x   2 root root 4096 Jun 20 00:00 sbin
drwxr-xr-x   2 root root 4096 Jun 20 00:00 srv
dr-xr-xr-x  13 root root    0 Jul  9 22:15 sys
```

Notice when we create a Docker image using "docker import" to define a new Docker image using a tar from a "docker export" , we see the "mytmp" folder and the cats.txt file.

Docker Commands cmd04 Summary

- In the this example we:
  - Used **docker export** to export the file system of a container into a tar file
  - Next, use used **docker import** to create a new Docker image from the exported tar file

Docker Commands cmd04 Summary

- An important point to take note of is:

  changes made within a Docker container (a running Docker image) are not saved.

- To save changes made within a Docker container, we need to create a new Docker images that contains:
  - The original base image we started the Docker container with
  - And include the run-time changes made in the Docker container into the new Docker image

Docker Command cmd05

- In this example we will:
  - Use **docker cp** to copy files from the host into a Docker container
  - Use **docker exec** to run a Python script in the container
  - Use **docker commit** to create a new image from the Docker container

Docker Command cmd05

- First, let's run a Docker container with image py02 in a slightly different way - detached:

```
#! / bin / sh – py02d.sh

docker run -td buildpack-deps-jessie:py02 /bin/sh
```

The −d flag runs the docker container − detached − not attached to a terminal

-t flag has docker allocate a **pseudo terminal**

Docker Command cmd05

- Using "docker ps" – we can see Docker containers – in the case we see the container that was created to run image py02 via script py02d.sh on the previous slide:

docker run -td buildpack-deps-jessie:py02 /bin/sh

```
JeffMacBookPro13:py01 jeff$ docker ps
CONTAINER ID        IMAGE                       COMMAND
f5b6f1727835        buildpack-deps-jessie:py02  "/bin/sh"
```

Docker Command cmd05

A **pseudo-terminal (ptty)** is a "virtual terminal". Linux and Unix provide provide pseudo-terminals a way to simulate a terminal, allow applications to read and write from/to the pseudo-terminals as if they are physical terminals (i.e. screen and keyboard).

Pttys use the concept of master/slave.

The slave is connected to the pseudo-terminal (possibly not knowing it is a pseudo-terminal).

The master program puts input into the pseudo-terminal and receives output from the ptty.

Using a ptty, allows programs in our Docker containers to read/write from terminals without actually being connected to a physical terminal.

Docker Command cmd05

- Next, lets create a simple Python file, test.py, that we will:
  - Copy into the Docker container running image py02 using **docker cp**
  - Invoke Python to run test.py using **docker exec** from the host
  - Use **docker commit** to create a new image py03-…
  - Run our new image, py03d-…, in a Docker container to execute test.py using **docker exec**

cmd05.sh

```
#! /bin/sh
containerid=$(docker ps | grep py02 | awk '{print $1}')
echo cid: $containerid

#this may fail since test.py is not initialliy inside the container
docker exec -ti $containerid sh -c "rm /test.py" > /dev/null

docker cp ./test.py "$containerid":/test.py

docker exec -ti "$containerid" sh -c "python /test.py date-time='$(date)'"

docker commit -m "commit-date='$(date)'" "$containerid" buildpack-deps-jessie:py03d-$RANDOM

rnd=$RANDOM
echo 'new images is:'
docker images | grep py03d-$rnd

docker commit -m "commit-date='$(date)'" "$containerid" buildpack-deps-jessie:py03d-$rnd
```

We will examine each of these commands in the following slides, first lets look at the output

Docker Command cmd05

```
JeffMacBookPro13:py01 jeff$ cmd05.sh
cid: f5b6f1727835
platform=linux2,date-time=Sat Jul 15 14:51:27 PDT 2017
sha256:cf2f48502a0babe09c2a402c6e9af8bcdc0e41b1665fdf3d3ed369ddd3d3db92
new images is:
buildpack-deps-jessie   py03d-24668        cf2f48502a0b        Less than a second ago   665MB
JeffMacBookPro13:py01 jeff$
JeffMacBookPro13:py01 jeff$
```

cmd05.sh

```
JeffMacBookPro13:py01 jeff$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
buildpack-deps-jessie   py03d-24668         cf2f48502a0b        3 minutes ago       665MB
buildpack-deps-jessie   py02                9d4f48cfe97d        3 days ago          665MB
buildpack-deps-jessie   py01                2adaf6d4ed4d        8 days ago          673MB
cassandra               latest              c82d9de5d478        2 weeks ago         386MB
mono                    latest              34ff573d43b0        3 weeks ago         508MB
mongo                   latest              71c101e16e61        3 weeks ago         358MB
ubuntu                  latest              d355ed3537e9        3 weeks ago         119MB
buildpack-deps          jessie              a5a7d7ba45bb        3 weeks ago         610MB
amazonlinux             latest              766ebb052d4f        3 months ago        162MB
amazonlinux             <none>              8ae6f52035b5        6 months ago        292MB
```

docker images

Docker Command cmd05

```
containerid=$(docker ps | grep py02 | awk '{print $1}')
echo cid: $containerid
```

- Run **docker ps**
  - pipe the output into
- **grep py02** , to filter the output of docker ps to see containers with py02 in the name
  - Pipe the output into
- **awk** to just print out the containerid column into the output, notice in the "docker ps" output container-id is the first column of output

- Bash variable **containerid** will contain the container id

Docker Command cmd05

```
#this may fail since test.py is not initialliy inside the container
docker exec -ti $containerid sh -c "rm /test.py" > /dev/null
```

- Run **docker exec**
- **-ti** – as discussed on the previous slide
  - **-i** perform the command interactive
  - **-t** allocate a pseudo-terminal
- **$containerid**
  - place the container-id into the command
- **sh -c "rm /test.py"** > /dev/null
  - **sh** - run a bash shell in the container
  - **-c** – use the following string as a command to run in the shell
  - "**rm /test.py**" – delete file /test.py which may not be present
  - **> /dev/null** – run the output into the null device so we do not see output

Docker Command cmd05

```
docker cp ./test.py "$containerid":/test.py
```

- **docker cp** – run the docker cp command
- **./test.py** – specify test.py in the local directory as the file to copy
- **"$containerid":/test.py** – specify the container-id and path to copy into

Docker Command cmd05

```
docker exec -ti "$containerid" sh -c "python /test.py date-time='$(date)'"
```

- **docker exec −ti**
  - run docker exec using an interactive terminal
- **"$containerid"**
  - put the container-id into the command line
- **sh -c "python /test.py date-time='$(date)'"**
  - **sh −c** - within the Docker container run the shell using the commands in the string that follows −c
  - **/test.py date-time='$(date)'"** − run test.py in python passing in date-time=the current return from running the date command

Docker Command cmd05

```
rnd=$RANDOM
echo 'new image is:'
docker images | grep py03d-$rnd
```

- **rnd-$RANDOM**
  - set variable rnd to be a random number via environment variab;e $RANDOM
- **echo 'new images is:'å**
  - display 'new image is' in terminal
- **docker images | grep py03d-$rnd**
  - run docker images, filter output through grep filtering on (selecting) and line that contains string

Docker Command cmd05

```
docker commit -m "commit-date='$(date)'" "$containerid" buildpack-deps-jessie:py03d-$rnd
```

- **docker commit**
  - run docker commit
- **-m "commit-date='$(date)'"**
  - a commit message
- **"$containerid"**
  - id of running container
- **buildpack-deps-jessie:py03d-$rnd**
  - repository-name:tag

Docker Command cmd05

```
JeffsMacBookPro:py01 jeffm$ docker commit --help

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

Create a new image from a container's changes

Options:
  -a, --author string    Author (e.g., "John Hannibal Smith <hannibal@a-team.com>")
  -c, --change list      Apply Dockerfile instruction to the created image
      --help             Print usage
  -m, --message string   Commit message
  -p, --pause            Pause container during commit (default true)
```