Docker CLI Cheat Sheet

Docker CLI Cheat Sheet

A practical quick-reference guide for Docker command-line operations. Commands are organized by category with common flags and realistic examples.

Images

Docker images are read-only templates used to create containers.

Build an Image

```
# Build from Dockerfile in current directory
docker build -t myapp:v1.0 .

# Build with custom Dockerfile name
docker build -t myapp:v1.0 -f Dockerfile.prod .

# Build with build arguments
docker build -t myapp:v1.0 --build-arg NODE_ENV=production .

# Build with no cache (fresh build)
docker build -t myapp:v1.0 --no-cache .

# Build for multiple platforms
docker buildx build --platform linux/amd64,linux/arm64 -t myapp:v1.0 .
```

Pull Images

```
# Pull from Docker Hub
docker pull nginx:alpine

# Pull specific version
docker pull postgres:15.2-alpine

# Pull all tags for an image
docker pull -a myregistry.io/myimage
```

```
# Pull from private registry
docker pull myregistry.io/myapp:v1.0
```

Push Images

```
# Tag image for registry
docker tag myapp:v1.0 myregistry.io/myapp:v1.0

# Push to registry
docker push myregistry.io/myapp:v1.0

# Push all tags
docker push -a myregistry.io/myapp
```

List Images

```
# List all images
docker images

# List with dangling (untagged) images
docker images -a

# List with digest information
docker images --digests

# Filter by reference
docker images --filter reference="nginx*"

# Format output
docker images --format "table {{.Repository}}\t{{.Tag}}\t{{.Size}}"
```

Remove Images

```
# Remove specific image
docker rmi nginx:alpine

# Remove by image ID
docker rmi abc123def456

# Force removal (even if used by stopped containers)
docker rmi -f nginx:alpine

# Remove multiple images
docker rmi nginx:alpine redis:alpine postgres:15
```

Inspect Images

```
# View image details
docker inspect nginx:alpine

# Get specific field with format
docker inspect --format='{{.Config.Cmd}}' nginx:alpine
```

```
# View image layers
docker history nginx:alpine

# View history with human-readable sizes
docker history --human nginx:alpine
```

Save and Load Images

```
# Save image to tar file
docker save -o myapp.tar myapp:v1.0

# Save multiple images
docker save -o images.tar nginx:alpine redis:alpine

# Load image from tar file
docker load -i myapp.tar

# Load from stdin
docker load < myapp.tar
```

Prune Images

```
# Remove dangling (untagged) images
docker image prune

# Remove all unused images (not just dangling)
docker image prune -a

# Force without confirmation
docker image prune -f
```

Containers

Containers are running instances of images.

Run Containers

```
# Run container in foreground
docker run nginx:alpine

# Run in detached mode (background)
docker run -d nginx:alpine

# Run with custom name
docker run -d --name web-server nginx:alpine

# Run with port mapping
docker run -d -p 8080:80 nginx:alpine

# Run with multiple ports
docker run -d -p 8080:80 -p 8443:443 nginx:alpine

# Run with environment variables
```

```
docker run -d -e POSTGRES_PASSWORD=secret postgres:15

# Run with env file
docker run -d --env-file .env myapp:v1.0

# Run with volume mount
docker run -d -v /host/path:/container/path nginx:alpine

# Run with bind mount (read-only)
docker run -d -v /host/path:/container/path:ro nginx:alpine

# Run with named volume
docker run -d -v mydata:/var/lib/postgresql/data postgres:15

# Run with resource limits
docker run -d --memory=512m --cpus=1.5 myapp:v1.0

# Run interactively with shell
docker run -it alpine:latest /bin/sh

# Run with auto-restart policy
docker run -d --restart=always nginx:alpine

# Run with restart on failure (max 3 attempts)
docker run -d --restart=on-failure:3 myapp:v1.0

# Run with network
docker run -d --network mynetwork myapp:v1.0

# Run with hostname
docker run -d --hostname web01 nginx:alpine

# Run with workdir
docker run -d -w /app myapp:v1.0

# Run with user
docker run -d -u 1000:1000 myapp:v1.0

# Remove container automatically when it exits
docker run --rm -it alpine:latest /bin/sh
```

Start and Stop Containers

```
# Start stopped container
docker start web-server

# Start multiple containers
docker start web-server db-server cache-server

# Stop running container (graceful, SIGTERM)
docker stop web-server

# Stop with timeout (seconds)
docker stop -t 30 web-server

# Stop multiple containers
docker stop web-server db-server
```

```
# Stop all running containers
docker stop $(docker ps -q)
```

Restart Containers

```
# Restart container
docker restart web-server

# Restart with timeout
docker restart -t 10 web-server
```

Kill Containers

```
# Kill container (SIGKILL, no grace period)
docker kill web-server

# Kill with custom signal
docker kill -s SIGINT web-server

# Kill all running containers
docker kill $(docker ps -q)
```

Remove Containers

```
# Remove stopped container
docker rm web-server

# Force remove running container
docker rm -f web-server

# Remove multiple containers
docker rm web-server db-server cache-server

# Remove all stopped containers
docker container prune

# Remove with volumes
docker rm -v web-server
```

Pause and Unpause Containers

```
# Pause container (freeze processes)
docker pause web-server

# Unpause container
docker unpause web-server
```

Rename Containers

```
# Rename container
docker rename web-server nginx-prod
```

Container Inspection

Commands to inspect and monitor running containers.

View Logs

```
# View container logs
docker logs web-server

# Follow logs in real-time
docker logs -f web-server

# Show last N lines
docker logs --tail 100 web-server

# Show logs since timestamp
docker logs --since 2024-01-15T10:00:00 web-server

# Show logs from last 30 minutes
docker logs --since 30m web-server

# Show logs with timestamps
docker logs -t web-server

# Combine options
docker logs -f --tail 50 --since 10m web-server
```

Inspect Containers

```
# View container details
docker inspect web-server

# Get specific field
docker inspect --format='{{.State.Status}}' web-server

# Get IP address
docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' web-server

# Get port mappings
docker inspect --format='{{json .NetworkSettings.Ports}}' web-server

# Get environment variables
docker inspect --format='{{range .Config.Env}}{{println .}}{{end}}' web-server

# Get volumes
docker inspect --format='{{json .Mounts}}' web-server | jq
```

Resource Statistics

```
# Live stream of resource usage
docker stats

# Stats for specific container
docker stats web-server
```

```
# Single snapshot (no stream)
docker stats --no-stream

# Custom format
docker stats --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}"
```

Running Processes

```
# View processes in container
docker top web-server

# View with custom ps options
docker top web-server aux
```

Port Information

```
# Show port mappings
docker port web-server

# Show specific port mapping
docker port web-server 80
```

Filesystem Changes

```
# Show filesystem changes since container started
docker diff web-server

# A = added, C = changed, D = deleted
```

Stream Events

```
# Stream Docker events
docker events

# Filter by container
docker events --filter container=web-server

# Filter by event type
docker events --filter type=container

# Filter by event (start, stop, die, etc.)
docker events --filter event=start

# Events since timestamp
docker events --since 60m
```

Execution

Commands to interact with running containers.

Execute Commands

```
# Run command in running container
docker exec web-server ls /etc/nginx

# Run interactive shell
docker exec -it web-server /bin/sh

# Run as specific user
docker exec -u root web-server cat /etc/passwd

# Run with environment variable
docker exec -e DEBUG=true web-server npm run debug

# Run with workdir
docker exec -w /app web-server npm test

# Run in detached mode
docker exec -d web-server nginx -s reload
```

Attach to Container

```
# Attach to running container
docker attach web-server

# Detach with Ctrl+P, Ctrl+Q (not Ctrl+C)
```

Copy Files

```
# Copy from host to container
docker cp ./config.json web-server:/app/config/

# Copy from container to host
docker cp web-server:/app/logs/ ./local-logs/

# Copy with container ID
docker cp abc123def456:/data/backup.tar ./backup.tar
```

Export Container

```
# Export container filesystem as tar
docker export web-server > backup.tar

# Export and compress
docker export web-server | gzip > backup.tar.gz

# Pipe to import
docker export web-server | docker import - myapp:backup
```

Commit Container

```
# Create image from container
docker commit web-server myapp:modified

# Commit with message
docker commit -m "Added custom config" web-server myapp:modified

# Commit with author
docker commit -a "DevOps Team" web-server myapp:modified

# Commit with new CMD
docker commit --change 'CMD ["nginx", "-g", "daemon off;"]' web-server myapp:modified
```

Volumes

Volumes persist data independent of container lifecycle.

Create Volumes

```
# Create named volume
docker volume create mydata

# Create with specific driver
docker volume create --driver local mydata

# Create with options
docker volume create --opt type=tmpfs --opt device=tmpfs mydata
```

List Volumes

```
# List all volumes
docker volume ls

# Filter by name
docker volume ls --filter name=mydata

# Filter by driver
docker volume ls --filter driver=local
```

Inspect Volumes

```
# View volume details
docker volume inspect mydata

# Get mountpoint
docker volume inspect --format='{{.Mountpoint}}' mydata
```

Remove Volumes

```
# Remove volume
docker volume rm mydata
```

```
# Remove multiple volumes
docker volume rm mydata cache sessions

# Force removal
docker volume rm -f mydata
```

Prune Volumes

```
# Remove all unused volumes
docker volume prune

# Force without confirmation
docker volume prune -f
```

Networks

Networks enable communication between containers.

Create Networks

```
# Create bridge network
docker network create mynetwork

# Create with specific driver
docker network create --driver bridge mynetwork

# Create overlay network (Swarm)
docker network create --driver overlay mynetwork

# Create with subnet
docker network create --subnet=192.168.1.0/24 mynetwork

# Create with gateway
docker network create --subnet=192.168.1.0/24 --gateway=192.168.1.1 mynetwork

# Create with IP range
docker network create --subnet=192.168.1.0/24 --ip-range=192.168.1.128/25 mynetwork

# Create attachable overlay network
docker network create --driver overlay --attachable mynetwork

# Create with labels
docker network create --label env=prod mynetwork
```

List Networks

```
# List all networks
docker network ls

# Filter by driver
docker network ls --filter driver=bridge
```

```
# Filter by name
docker network ls --filter name=my*
```

Inspect Networks

```
# View network details
docker network inspect mynetwork

# Get container IPs
docker network inspect --format='{{range .Containers}}{{.Name}}: {{.IPv4Address}}{{println}}{
```

Connect Containers to Networks

```
# Connect running container to network
docker network connect mynetwork web-server

# Connect with static IP
docker network connect --ip 192.168.1.100 mynetwork web-server

# Connect with alias
docker network connect --alias db mynetwork postgres-server
```

Disconnect Containers from Networks

```
# Disconnect container from network
docker network disconnect mynetwork web-server

# Force disconnect
docker network disconnect -f mynetwork web-server
```

Remove Networks

```
# Remove network
docker network rm mynetwork

# Remove multiple networks
docker network rm net1 net2 net3
```

Prune Networks

```
# Remove all unused networks
docker network prune

# Force without confirmation
docker network prune -f
```

Docker Compose

Docker Compose manages multi-container applications.

Start Services

```
# Start all services (detached)
docker compose up -d

# Start with build
docker compose up -d --build

# Start specific services
docker compose up -d web db

# Start with force recreate
docker compose up -d --force-recreate

# Start with scale
docker compose up -d --scale web=3

# Start with environment file
docker compose --env-file .env.prod up -d

# Start with project name
docker compose -p myproject up -d

# Start with custom compose file
docker compose -f docker-compose.prod.yml up -d
```

Stop Services

```
# Stop all services
docker compose down

# Stop and remove volumes
docker compose down -v

# Stop and remove images
docker compose down --rmi all

# Stop and remove orphans
docker compose down --remove-orphans

# Stop without removing
docker compose stop

# Stop specific service
docker compose stop web
```

Build Services

```
# Build all services
docker compose build

# Build specific service
docker compose build web

# Build with no cache
```

```
docker compose build --no-cache

# Build with parallel
docker compose build --parallel

# Pull images before build
docker compose build --pull
```

View Logs

```
# View all logs
docker compose logs

# Follow logs
docker compose logs -f

# Show last N lines
docker compose logs --tail 100

# Logs for specific service
docker compose logs web

# Logs with timestamps
docker compose logs -t
```

List Services

```
# List running services
docker compose ps

# List all services (including stopped)
docker compose ps -a

# Show service IDs only
docker compose ps -q
```

Execute in Services

```
# Run command in service container
docker compose exec web ls /app

# Run interactive shell
docker compose exec web /bin/sh

# Run as specific user
docker compose exec -u root web cat /etc/passwd

# Run one-off command
docker compose run --rm web npm test

# Run with service ports
docker compose run --service-ports web
```

Scale Services

```
# Scale service to 3 instances
docker compose up -d --scale web=3

# Scale multiple services
docker compose up -d --scale web=3 --scale worker=2
```

Other Compose Commands

```
# Validate compose file
docker compose config

# Show service images
docker compose images

# Show top processes
docker compose top

# Pause services
docker compose pause

# Unpause services
docker compose unpause

# Restart services
docker compose restart

# Restart specific service
docker compose restart web

# Pull all images
docker compose pull

# Push all images
docker compose push

# Copy files from service
docker compose cp web:/app/logs ./logs

# View port mappings
docker compose port web 80
```

System

System-level Docker commands.

System Information

```
# Display system-wide information
docker info

# Show Docker version
docker version
```

```
# Show version (short)
docker version --format='{{.Server.Version}}'
```

Disk Usage

```
# Show disk usage
docker system df

# Show detailed disk usage
docker system df -v
```

System Events

```
# Stream real-time events
docker system events

# Filter by type
docker system events --filter type=container

# Filter by image
docker system events --filter image=nginx:alpine
```

Login to Registry

```
# Login to Docker Hub
docker login

# Login to private registry
docker login myregistry.io

# Login with credentials
docker login -u username -p password myregistry.io

# Logout
docker logout myregistry.io
```

Cleanup

Commands to reclaim disk space and remove unused resources.

System-Wide Prune

```
# Remove unused data (images, containers, networks)
docker system prune

# Remove all unused data including volumes
docker system prune -a --volumes

# Force without confirmation
docker system prune -f
```

```
# Filter by time (remove older than)
docker system prune --filter "until=24h"
```

Prune by Resource Type

```
# Prune containers (stopped)
docker container prune

# Prune containers older than 1 hour
docker container prune --filter "until=1h"

# Prune images (dangling)
docker image prune

# Prune all unused images
docker image prune -a

# Prune volumes
docker volume prune

# Prune volumes with filter
docker volume prune --filter "label!=keep"

# Prune networks
docker network prune

# Prune build cache
docker builder prune

# Prune all build cache
docker builder prune -a
```

Remove Everything

```
# Nuclear option: remove all containers, images, volumes, networks
docker stop $(docker ps -aq) 2>/dev/null
docker rm $(docker ps -aq) 2>/dev/null
docker rmi $(docker images -q) 2>/dev/null
docker volume rm $(docker volume ls -q) 2>/dev/null
docker network rm $(docker network ls -q) 2>/dev/null

# Or use prune with all flags
docker system prune -a --volumes -f
```

Clean Up Specific Resources

```
# Remove exited containers
docker rm $(docker ps -q -f status=exited)

# Remove containers created in last hour
docker rm $(docker ps -q -f created=1h)

# Remove dangling images
```

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

# Remove images with specific label
docker rmi $(docker images -q -f label=stage=build)

# Remove volumes not used by any container
docker volume rm $(docker volume ls -qf dangling=true)
```

Common Flags Reference

Flag	Description
`-d, --detach`	Run in background
`-i, --interactive`	Keep STDIN open
`-t, --tty`	Allocate pseudo-TTY
`-p, --publish`	Publish port (host:container)
`-P, --publish-all`	Publish all exposed ports
`-v, --volume`	Bind mount volume
`-e, --env`	Set environment variable
`--env-file`	Read environment from file
`--name`	Assign container name
`--restart`	Restart policy (no, always, on-failure)
`--rm`	Auto-remove container on exit
`-w, --workdir`	Working directory
`-u, --user`	Username or UID
`--network`	Connect to network
`--hostname`	Container hostname
`--memory, -m`	Memory limit
`--cpus`	CPU limit
`-a, --all`	Show all (including stopped)
`-q, --quiet`	Only show IDs
`-f, --filter`	Filter output
`--format`	Format output
`--no-cache`	Disable cache
`--force`	Force operation
`-f, --file`	Specify Dockerfile or compose file

Exit Codes

Code	Meaning
0	Success

Code	Meaning
1	Application error
137	SIGKILL (OOM killed or manual kill)
139	Segmentation fault
143	SIGTERM (graceful stop)
255	Exit code out of range

Quick Reference

```
# See what's running
docker ps

# See all containers (including stopped)
docker ps -a

# See images
docker images

# See disk usage
docker system df

# Follow logs
docker logs -f <container>

# Shell into container
docker exec -it <container> /bin/sh

# Quick cleanup
docker system prune -f

# Full cleanup (destructive)
docker system prune -a --volumes -f
```

Next Steps

- Kubernetes kubectl Cheat Sheet — Orchestrate containers at scale
- Helm CLI Cheat Sheet — Package and deploy applications
- Docker Multi-Stage Builds — Optimize production images
- Container Building and Hardening — Security best practices