Docker File Syntax

1. **FROM:** This instruction initializes a new build stage and sets the base image for subsequent instructions. As such, a valid Dockerfile must have FROM as its first instruction. This is typically the first line in your Dockerfile.
   ```
   FROM ubuntu:18.04
   ```

2. **RUN**: This instruction allows you to execute a command-line within the Docker image. It has two forms: the shell form RUN <command> and the exec form RUN ["executable", "param1", "param2"].
   This could be used to install necessary packages.
   ```
   RUN apt-get update && apt-get install -y python3
   ```

3. **CMD:** This provides defaults for an executing container. These can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction. There can only be one CMD instruction in a Dockerfile. If you list more than one CMD, then the last CMD will take effect.
   This sets the default command to execute when a container starts.
   ```
   CMD ["python3", "app.py"]
   ```

4. **LABEL**: This instruction adds metadata to an image. It accepts a dictionary of <key>=<value> pairs.
   This could be used to add metadata to the image.
   ```
   LABEL version="1.0" description="My Python App"
   ```

5. **EXPOSE:** This informs Docker that the container listens on the specified network ports at runtime. This does not actually publish the port.
   This could be used to expose a certain port of the container.
   ```
   EXPOSE 8080
   ```

6. **ENV:** This instruction sets environment variables. These variables consist of “key=value” pairs which can be accessed within the container by scripts and applications alike.
   This could be used to set environment variables.
   ENV API_KEY="123456"

7. **ADD:** This instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.
   This could be used to copy a local file to the container.
   ADD ./app /app

8. **COPY:** This is similar to ADD but without the extra features of tar extraction and remote URL support.
   This is used similarly to ADD, but without handling URLs or uncompressed tar files.
   COPY ./app /app

9. **ENTRYPOINT:** This allows you to configure a container that will run as an executable. It has two forms: the exec form and the shell form.
   This could be used to run a specific executable.
   ENTRYPOINT ["python3", "/app/main.py"]

10. **VOLUME:** This instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.
    This could be used to create a directory mount point.
    VOLUME /data

11. **USER:** This instruction sets the user name (or UID) and optionally the user group (or GID) to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.
    This could be used to set the user to run the container.
    USER root

12. **WORKDIR:** This instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.
    This could be used to set the working directory.
    WORKDIR /app

13. **ARG:** This instruction defines a variable that users can pass at build-time to the builder with the docker build command.
    This could be used to define a build-time variable.
    ARG VERSION=latest

14. **ONBUILD:** This instruction adds a trigger instruction when the image is used as the base for another build.
    This could be used to execute commands when the image is being used as a base image.
    ONBUILD ADD . /app

15. **STOPSIGNAL**: This instruction sets the system call signal that will be sent to the container to exit.
    This could be used to specify a stop signal.
    STOPSIGNAL SIGTERM

16. **HEALTHCHECK:** This instruction tells Docker how to test a container to check that it is still working.
    This could be used to determine the health of a container.
    HEALTHCHECK CMD curl --fail http://localhost:8080/ || exit 1

17. **SHELL:** This instruction allows the default shell used for the RUN, CMD, ENTRYPOINT, COPY and ADD instructions to be overridden.
    This could be used to override the default shell.
    SHELL ["/bin/bash", "-c"]

The .dockerignore file is also an important part of a Dockerfile, which allows you to specify a pattern for files and directories that should be ignored by Docker's build process. It works much like a .gitignore file.

Understanding and properly using these instructions allows for efficient, accurate, and secure Docker image creation. It's important to note that Docker instructions are case-sensitive and should be written in uppercase.
.dockerignore file example might look something like this:
*.log
tmp/
This would ignore all .log files and everything in the tmp directory.

Here is a list of some commonly used Docker commands:

docker run: Runs a command in a new container.

docker start: Starts one or more stopped containers.

docker stop: Stops one or more running containers.

docker build: Builds an image from a Dockerfile.

docker pull: Pulls an image or a repository from a registry.

docker push: Pushes an image or a repository to a registry.

docker exec: Runs a command in a running container.

docker ps: Lists all running containers.

docker logs: Shows logs from a container.

docker rm: Removes one or more containers.

docker rmi: Removes one or more images.

docker images: Lists all images on the host.

docker inspect: inspects one or more containers or images.

docker network : Create, inspect and manage networks

docker volume: Create and manage volumes

docker system prune: clean up system

docker commit: Creates a new image from a container's changes. It allows you to save the container current state as a new image.

Note: This is not an exhaustive list and there are many more commands available.

```
# The `FROM` instruction initializes a new build stage and sets the base image.
FROM ubuntu:20.04

# The `LABEL` instruction adds metadata to an image.
LABEL maintainer="example@example.com" version="1.0"

# The `ARG` instruction defines a variable that users can pass at build-time.
ARG DEBIAN_FRONTEND=noninteractive

# The `ENV` instruction sets environment variables.
ENV TZ=America/New_York

# The `RUN` instruction allows you to execute a command-line within the Docker image.
RUN apt-get update && apt-get install -y curl

# The `WORKDIR` instruction sets the working directory for any instructions that follow it in the
Dockerfile.
WORKDIR /app

# The `ADD` instruction copies new files, directories or remote file URLs and adds them to the
filesystem of the image.
ADD https://example.com/big-file.tar.gz /app/

# The `COPY` instruction copies new files or directories from source and adds them to the
filesystem of the image.
COPY . .

# The `USER` instruction sets the user name to use when running the image.
USER root

# The `EXPOSE` instruction informs Docker that the container listens on the specified network
ports at runtime.
EXPOSE 8080

# The `VOLUME` instruction creates a mount point with the specified name.
VOLUME /var/data

# The `ENTRYPOINT` instruction allows you to configure a container that will run as an
executable.
ENTRYPOINT ["./app"]

# The `CMD` provides defaults for an executing container.
CMD ["./app", "param1", "param2"]

# The `ONBUILD` instruction adds a trigger instruction when the image is used as the base for
another build.
ONBUILD RUN echo 'We are running some #coolapp'

# The `STOPSIGNAL` instruction sets the system call signal that will be sent to the container to
exit.
STOPSIGNAL SIGTERM

# The `HEALTHCHECK` instruction tells Docker how to test a container to check that it is still
working.
HEALTHCHECK --interval=5s --timeout=3s CMD curl -f http://localhost || exit 1

# The `SHELL` instruction allows the default shell used for the instructions to be overridden.
SHELL ["/bin/bash", "-c"]
```