How I Reduced the Size of a Docker Image by 40%

Key words: Docker, scripts, Git, Dockerfile, bash

Authors: Pham Manh Tuan

The problem defined:

Every single time I deal with Dockerfiles. I've written several myself, created containers, and all that. But I've never published them to the Docker Hub registry. I wanted to create ugit—a tool for undoing git commands (written as a shell script) that people who don't like installing random shell scripts from the Internet could use.

Yes, yes, I know. IT NEEDS TO BE REWRITTEN IN GO/RUST/ANOTHER MAGICAL LANGUAGE. Currently, the script consists of over five hundred lines of Bash, so I will only rewrite it in another language under the threat of death. Besides, ugit already has almost all the functions (there are only a few infrequently used commands left to implement).

In this report, I will talk about how I wrote the official Dockerfile for ugit (a shell script) and reduced the image size by almost 40% (from 31.4 MB to 17.6 MB) by following step-by-step attempts according to the instructions. I hope this will motivate other shell enthusiasts to publish their scripts as Docker images too!

1 THE VERY FIRST ATTEMPT AT CREATING A DOCKERFILE

```
# Use the official Alpine as the parent image
FROM alpine:3.18

# Set the working directory in the container to /app
WORKDIR /app

# Copy the contents of the current directory to /app in the container
COPY . /app

# Install dependencies
RUN apk add --no-cache \
    bash \
    gawk \
    findutils \
    coreutils \
    git \
    ncurses \
    fzf

# Set permissions and move the script to the appropriate location
RUN chmod +x ugit && mv ugit /usr/local/bin/

# Run ugit when the container launches
CMD ["ugit"]
```

It looks pretty simple, doesn't it? It is. You can copy-paste this Dockerfile and build the image yourself, provided you've cloned *ugit* into the current folder beforehand.

```
docker build -t ugit .
docker run --rm -it -v $(pwd):/app ugit
```

ugit should run inside the container.

ugit requires binaries like bash (version 4.0 and above), *awk*, *xargs*, *git*, *fzf*, *tput*, *cut*, *tr*, and *nl*. *ugit* should run inside the container.

- We need to install findutils because it comes with *xargs*.
- We also need to install coreutils because it comes with *tr*, *cut*, and *nl*.
- *ncurses* is required for tput (which is used to get terminal information).

That's all you need to run *ugit* on a Unix-like operating system or in a container. At this stage, the image size is 31.4 MB. Not bad for a first attempt. Let's see if we can reduce it even further.

The path to optimization is reducing the image size by 40%

During our (desperate) attempts at micro-optimizations below, we'll aim to achieve the following high-level goals:

- Use multi-stage builds to reduce the image size.

- Eliminate binaries like *sleep*, *watch*, *du*, and so on. None of these are required to run *ugit*.
- Remove unnecessary dependencies introduced by these binaries.
- Get the minimal version of all dependencies required to run *ugit*.
- Load only the necessary binaries and their dependencies into the final image

2 THE SECOND ATTEMPT - ALPINE ON ALPINE

Next, I decided to create a multi-stage build. The second stage will be used to copy only the necessary binary files and their dependencies. Once again, I chose Alpine as the base image for this stage.

```
# First stage: Install packages
FROM alpine:3.18 as builder

RUN apk add --no-cache \
    bash \
    gawk \
    findutils \
    coreutils \
    git \
    ncurses \
    fzf

# Copy only the ugit script to /app in the container
WORKDIR /app
COPY ugit .

# Set permissions and move the script to the appropriate location
RUN chmod +x ugit && mv ugit /usr/local/bin/

# Second stage: Copy only necessary binary files and their dependencies
FROM alpine

COPY --from=builder /usr/local/bin/ugit /usr/bin/
COPY --from=builder /usr/bin/git /usr/bin/
COPY --from=builder /usr/bin/fzf /usr/bin/
COPY --from=builder /usr/bin/tput /usr/bin/
COPY --from=builder /usr/bin/cut /usr/bin/
COPY --from=builder /usr/bin/tr /usr/bin/
COPY --from=builder /usr/bin/nl /usr/bin/
COPY --from=builder /usr/bin/gawk /usr/bin/
COPY --from=builder /usr/bin/xargs /usr/bin/
COPY --from=builder /usr/bin/env /bin/
COPY --from=builder /bin/bash /bin/

WORKDIR /app

# Run ugit when the container starts
CMD ["ugit"]
```

With just a simple multi-stage build, we've managed to reduce the image size to an impressive 20.6 MB. The image builds successfully, but ugit isn't running yet.

```
Error loading shared library libreadline.so.8: No such file or directory
(needed by /bin/bash)
```

It looks like we're getting *xargs* and *awk* for free?

It turns out both xargs and awk are included by default in the Alpine image. You can confirm this by running the following commands:

```
docker run -it alpine /bin/sh -c "awk --help
docker run -it alpine /bin/sh -c "xargs --help
```

Rookie mistake. Let's remove *gawk* and *findutils* from our Dockerfile.

```
# First stage: Install packages
FROM alpine:3.18 as builder
RUN apk add --no-cache \
    bash \
    coreutils \
    git \
    ncurses \
    fzf

# Copy only the ugit script to /app in the container
WORKDIR /app
COPY ugit .

# Set permissions and move the script to the appropriate location
RUN chmod +x ugit && mv ugit /usr/local/bin/

# Second stage: Copy only necessary binary files and their dependencies
FROM alpine:3.18
COPY --from=builder /usr/local/bin/ugit /usr/bin/
COPY --from=builder /usr/bin/git /usr/bin/
COPY --from=builder /usr/bin/fzf /usr/bin/
COPY --from=builder /usr/bin/tput /usr/bin/
COPY --from=builder /usr/bin/cut /usr/bin/
COPY --from=builder /usr/bin/tr /usr/bin/
COPY --from=builder /usr/bin/nl /usr/bin/
COPY --from=builder /usr/bin/env /bin/
COPY --from=builder /bin/bash /bin/

WORKDIR /app

# Run ugit when the container starts
CMD ["ugit"]
```

The image size has been reduced to 20 MB. We're getting closer, but *ugit* still isn't running.

3 THE THIRD ATTEMPT - USING SCRATCH ON THE SECOND STAGE

Most people don't attempt this. It's quite daunting and requires a lot of courage. I was a bit scared too, going down this path.

The Docker SCRATCH image is simply an empty file system. There's nothing in it at all.

The only thing you need to know is that we'll have to put everything together manually. Let's cross our fingers and replace alpine with scratch.

```
# First stage: Install packages
FROM alpine:3.18 as builder
RUN apk add --no-cache \
    bash \
    coreutils \
    git \
    ncurses \
    fzf

# Copy only the ugit script to /app in the container
COPY ugit .

# Set permissions and move the script to the appropriate location
RUN chmod +x ugit && mv ugit /usr/local/bin/

# Second stage: Copy only necessary binary files and their dependencies
FROM scratch
COPY --from=builder /usr/local/bin/ugit /usr/bin/
COPY --from=builder /usr/bin/git /usr/bin/
COPY --from=builder /usr/bin/fzf /usr/bin/
COPY --from=builder /usr/bin/tput /usr/bin/
COPY --from=builder /usr/bin/cut /usr/bin/
COPY --from=builder /usr/bin/tr /usr/bin/
COPY --from=builder /usr/bin/nl /usr/bin/
COPY --from=builder /usr/bin/env /bin/
COPY --from=builder /bin/bash /bin/

WORKDIR /app

# Run ugit when the container starts
CMD ["ugit"]
```

This reduces our image size to 12.4 MB. A 60% reduction? Did we just outsmart ourselves? Let's try running *ugit*.

```
$ docker run --rm -it -v $(pwd):/app ugit-a3
exec /usr/bin/ugit: no such file or directory
$ docker run --rm -it --entrypoint /bin/bash ugit-a4
exec /bin/bash: no such file or directory
```

It turns out we broke the bash binary by not including its dependencies.

Here's the completed Dockerfile

```
# First stage: Install packages
FROM alpine:3.18 as ugit-ops

# Install necessary packages
RUN apk add --no-cache \
```

```
    bash \
    coreutils \
    git \
    ncurses \
    curl

# Download the fzf binary from GitHub, version 0.46.0 as ugit requires at
least 0.21.0
RUN curl -L -o fzf.tar.gz
https://github.com/junegunn/fzf/releases/download/0.46.0/fzf-0.46.0-
linux_amd64.tar.gz && \
    tar -xzf fzf.tar.gz && \
    mv fzf /usr/bin/

# Copy the ugit script to the container
COPY ugit .

# Set permissions and move the script to the appropriate location
RUN chmod +x ugit && mv ugit /usr/bin/

# Second stage: Use scratch and copy only necessary files and dependencies
FROM scratch

# Copy necessary binaries
COPY --from=ugit-ops /usr/bin/ugit /bin/
COPY --from=ugit-ops /usr/bin/git /usr/bin/
COPY --from=ugit-ops /usr/bin/fzf /usr/bin/
COPY --from=ugit-ops /usr/bin/tput /usr/bin/
COPY --from=ugit-ops /usr/bin/nl /usr/bin/
COPY --from=ugit-ops /usr/bin/awk /usr/bin/
COPY --from=ugit-ops /usr/bin/xargs /usr/bin/
COPY --from=ugit-ops /usr/bin/cut /usr/bin/
COPY --from=ugit-ops /usr/bin/tr /usr/bin/
COPY --from=ugit-ops /bin/bash /bin/
COPY --from=ugit-ops /bin/sh /bin/

# Copy library files
COPY --from=ugit-ops /usr/lib/libncursesw.so.6 /usr/lib/
COPY --from=ugit-ops /usr/lib/libncursesw.so.6.4 /usr/lib/
COPY --from=ugit-ops /usr/lib/libpcre* /usr/lib/
COPY --from=ugit-ops /usr/lib/libreadline* /usr/lib/
COPY --from=ugit-ops /lib/libacl.so.1 /lib/
COPY --from=ugit-ops /lib/libattr.so.1 /lib/
COPY --from=ugit-ops /lib/libc.musl-* /lib/
COPY --from=ugit-ops /lib/ld-musl-* /lib/
COPY --from=ugit-ops /lib/libutmps.so.0.1 /lib/
COPY --from=ugit-ops /lib/libskarnet.so.2.13 /lib/
COPY --from=ugit-ops /lib/libz.so.1 /lib/

# Copy terminfo database
COPY --from=ugit-ops /etc/terminfo/x/xterm-256color /usr/share/terminfo/x/

# Set TERM environment variable
ENV TERM=xterm-256color

# Set working directory
WORKDIR /app

# Run ugit when the container starts
CMD ["/bin/bash", "/bin/ugit"]
```

The final Docker image has a size of 17.6 MB and contains no security vulnerabilities. We successfully reduced the image size by 40%.