

docker/scout-cli

Docker Scout CLI

ref: 01e0b04d228e9ff32307848ee06594c3676f0cf1

License: Other

Stars: 309

Forks: 74

This PDF was generated by gitprint.me

Top Contributors



cdupuis (93)



eunomie (41)



docker-scout-
ci[bot] (19)



felipecruz91
(5)



mcapell (4)



xcirel (2)



Dhanush-Reddy
(1)



GTRekter (1)



lamtrinhdev
(1)



black-snow
(1)



rraszewski
(1)

Chapter 0.0.0

root

README.md

- Docker Scout
- Usage
- CLI Plugin Installation
- Run as container
- CI integration
- License

Docker Scout

Docker Scout is a set of software supply chain features integrated into Docker's user interfaces and command line interface (CLI). These features offer comprehensive visibility into the structure and security of container images. This repository contains installable binaries of the docker scout CLI plugin.

Usage

The CLI documentation is available in this repository.

See the reference documentation to learn about Docker Scout including Docker Desktop and Docker Hub integrations.

Environment Variables

The following environment variables are available to configure the Scout CLI:

Name	Description
DOCKER_SCOUT_CACHE_FORMAT	Format of the local image cache; can be oci or tar
DOCKER_SCOUT_CACHE_DIR	Directory where the local SBOM cache is stored
DOCKER_SCOUT_NO_CACHE	Disable the local SBOM cache
DOCKER_SCOUT_REGISTRY_TOKEN	Registry Access token to authenticate when pulling images
DOCKER_SCOUT_REGISTRY_USER	Registry user name to authenticate when pulling images
DOCKER_SCOUT_REGISTRY_PASSWORD	Registry password/PAT to authenticate when pulling images
DOCKER_SCOUT_HUB_USER	Docker Hub user name to authenticate against the Docker Scout backend
DOCKER_SCOUT_HUB_PASSWORD	Docker Hub password/PAT to authenticate against the Docker Scout backend
DOCKER_SCOUT_OFFLINE	Offline mode during SBOM indexing
DOCKER_SCOUT_NEW_VERSION_WARN	Warn about new versions of the Docker Scout CLI
DOCKER_SCOUT_EXPERIMENTAL_WARN	Warn about experimental features
DOCKER_SCOUT_EXPERIMENTAL_POLICY_OUTPUT	Disable experimental policy output

CLI Plugin Installation

Docker Desktop

docker scout CLI plugin is available by default on Docker Desktop starting with version 4.17.

Manual Installation

To install it manually:

- Download the docker-scout binary corresponding to your platform from the latest or other releases.
- Uncompress it as
- docker-scout on *Linux* and *macOS*

- `docker-scout.exe` on *Windows*
- Copy the binary to the scout directory
- `$HOME/.docker/scout` on *Linux* and *macOS*
- `%USERPROFILE%\docker\scout` on *Windows*
- Make it executable on *Linux* and *macOS*
- `chmod +x $HOME/.docker/scout/docker-scout`
- Authorize the binary to be executable on *macOS*
- `xattr -d com.apple.quarantine $HOME/.docker/scout/docker-scout`
- Add the scout directory to your `.docker/config.json` as a plugin directory
- `$HOME/.docker/config.json` on *Linux* and *macOS*
- `%USERPROFILE%\docker\config.json` on *Windows*
- Add the `cliPluginsExtraDirs` property to the `config.json` file

```
{
  ...
  "cliPluginsExtraDirs": [
    "<full path to the .docker/scout folder>"
  ],
  ...
}
```

Script Installation (macOS and Linux)

To install, run the following command in your terminal:

```
curl -sSfL https://raw.githubusercontent.com/docker/scout-cli/main/install.sh | sh -s --
```

Run as container

A container image to run the Docker Scout CLI in containerized environments is available at `docker/scout-cli`.

CI Integration

Docker Scout CLI can be used in CI environments. See below for the various ways to integrate the CLI into your CI pipelines.

GitHub Action

An early prototype of running the Docker Scout CLI as part of a GitHub Action workflow is available at `docker/scout-action`.

The following GitHub Action workflow can be used as a template to integrate Docker Scout:

```
name: Docker

on:
  push:
    tags: [ "*" ]
    branches:
      - 'main'
  pull_request:
    branches: [ "*" ]

env:
  # Use docker.io for Docker Hub if empty
  REGISTRY: docker.io
  IMAGE_NAME: ${ github.repository }
  SHA: ${ github.event.pull_request.head.sha || github.event.after }

jobs:
  build:

    runs-on: ubuntu-latest
    permissions:
      contents: read
```

```
packages: write
```

```
steps:
```

```
- name: Checkout repository
  uses: actions/checkout@v3
  with:
    ref: ${{ env.SHA }}

- name: Setup Docker buildx
  uses: docker/setup-buildx-action@v2.5.0

# Login against a Docker registry except on PR
# https://github.com/docker/login-action
- name: Log into registry ${{ env.REGISTRY }}
  uses: docker/login-action@v2.1.0
  with:
    registry: ${{ env.REGISTRY }}
    username: ${{ secrets.DOCKER_USER }}
    password: ${{ secrets.DOCKER_PAT }}

# Extract metadata (tags, labels) for Docker
# https://github.com/docker/metadata-action
- name: Extract Docker metadata
  id: meta
  uses: docker/metadata-action@v4.4.0
  with:
    images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
    labels: |
      org.opencontainers.image.revision=${{ env.SHA }}
    tags: |
      type=edge,branch=$repo.default_branch
      type=semver,pattern=v{{version}}
      type=sha,prefix=,suffix=,format=short

# Build and push Docker image with Buildx (don't push on PR)
# https://github.com/docker/build-push-action
- name: Build and push Docker image
  id: build-and-push
  uses: docker/build-push-action@v4.0.0
  with:
    context: .
    push: true
    tags: ${{ steps.meta.outputs.tags }}
    labels: ${{ steps.meta.outputs.labels }}
    cache-from: type=gha
    cache-to: type=gha,mode=max

- name: Docker Scout
  id: docker-scout
  if: ${{ github.event_name == 'pull_request' }}
  uses: docker/scout-action@dd36f5b0295baffa006aa6623371f226cc03e506
  with:
    command: cves
    image: ${{ steps.meta.outputs.tags }}
    only-severities: critical,high
    exit-code: true
```

GitLab

Use the following pipeline definition as a template to get Docker Scout integrated in GitLab CI:

```
docker-build:
  image: docker:latest
  stage: build
```

```

services:
  - docker:dind
before_script:
  - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY

# Install curl and the Docker Scout CLI
- |
  apk add --update curl
  curl -sSfL https://raw.githubusercontent.com/docker/scout-cli/main/install.sh | sh
  apk del curl
  rm -rf /var/cache/apk/*
# Login to Docker Hub required for Docker Scout CLI
- echo "$DOCKER_HUB_PAT" | docker login --username "$DOCKER_HUB_USER" --password-stc
script:
- |
  if [[ "$CI_COMMIT_BRANCH" == "$CI_DEFAULT_BRANCH" ]]; then
    tag=""
    echo "Running on default branch '$CI_DEFAULT_BRANCH': tag = 'latest'"
  else
    tag=":$CI_COMMIT_REF_SLUG"
    echo "Running on branch '$CI_COMMIT_BRANCH': tag = $tag"
  fi
- docker build --pull -t "$CI_REGISTRY_IMAGE${tag}" .

- |
  if [[ "$CI_COMMIT_BRANCH" == "$CI_DEFAULT_BRANCH" ]]; then
    # Get a CVE report for the built image and fail the pipeline when critical or hi
    docker scout cves "$CI_REGISTRY_IMAGE${tag}" --exit-code --only-severity critica
  else
    # Compare image from branch with latest image from the default branch and fail i
    docker scout compare "$CI_REGISTRY_IMAGE${tag}" --to "$CI_REGISTRY_IMAGE:latest"
  fi

- docker push "$CI_REGISTRY_IMAGE${tag}"
rules:
  - if: $CI_COMMIT_BRANCH
    exists:
      - Dockerfile

```

CircleCI

Use the following pipeline definition as a template to get Docker Scout integrated in CircleCI project:

```
version: 2.1
```

```
jobs:
```

```
  build:
```

```
    docker:
```

```
      - image: cimg/base:stable
```

```
    environment:
```

```
      IMAGE_TAG: docker/scout-demo-service:latest
```

```
    steps:
```

```
      # Checkout the repository files
      - checkout
```

```
      # Set up a separate Docker environment to run `docker` commands in
```

```
      - setup_remote_docker:
          version: 20.10.24
```

```
      # Install Docker Scout and login to Docker Hub
```

```

- run:
  name: Install Docker Scout
  command: |
    env
    curl -sSfL https://raw.githubusercontent.com/docker/scout-cli/main/install.s
    echo $DOCKER_HUB_PAT | docker login -u $DOCKER_HUB_USER --password-stdin

# Build the Docker image
- run:
  name: Build Docker image
  command: docker build -t $IMAGE_TAG .

# Run Docker Scout
- run:
  name: Scan image for CVEs
  command: |
    docker-scout cves $IMAGE_TAG --exit-code --only-severity critical,high

```

workflows:

```

  build-docker-image:
    jobs:
      - build

```

Microsoft Azure DevOps Pipelines

Use the following pipeline definition as a template to get Docker Scout integrated in Azure DevOps Pipelines:

```

trigger:
- main

```

```

resources:
- repo: self

```

```

variables:
  tag: '$(Build.BuildId)'
  image: 'vonwig/nodejs-service'

```

```

stages:
- stage: Build
  displayName: Build image
  jobs:
  - job: Build
    displayName: Build
    pool:
      vmImage: ubuntu-latest
    steps:
  - task: Docker@2
    displayName: Build an image
    inputs:
      command: build
      dockerfile: '$(Build.SourcesDirectory)/Dockerfile'
      repository: $(image)
      tags: |
        $(tag)
  - task: CmdLine@2
    displayName: Find CVEs on image
    inputs:
      script: |
        # Install the Docker Scout CLI
        curl -sSfL https://raw.githubusercontent.com/docker/scout-cli/main/install.sh
        # Login to Docker Hub required for Docker Scout CLI
        docker login -u $(DOCKER_HUB_USER) -p $(DOCKER_HUB_PAT)
        # Get a CVE report for the built image and fail the pipeline when critical or
        docker scout cves $(image):$(tag) --exit-code --only-severity critical,high

```

Jenkins

The following snippet can be added to a Jenkinsfile to install and analyze images:

```
stage('Analyze image') {
  steps {
    // Install Docker Scout
    sh 'curl -sSfL https://raw.githubusercontent.com/docker/scout-cli/main/installs

    // Log into Docker Hub
    sh 'echo $DOCKER_HUB_PAT | docker login -u $DOCKER_HUB_USER --password-s

    // Analyze and fail on critical or high vulnerabilities
    sh 'docker-scout cves $IMAGE_TAG --exit-code --only-severity critical,hi
  }
}
```

This example assume two secrets to be available to authenticate against Docker Hub, called DOCKER_HUB_USER and DOCKER_HUB_PAT.

Bitbucket

Use the following pipeline definition as a template to get Docker Scout integrated in Bitbucket Pipelines:

```
image: docker
```

```
pipelines:
```

```
  default:
```

```
    - step:
```

```
      name: Build
```

```
      services:
```

```
        - docker
```

```
      caches:
```

```
        - docker
```

```
      script:
```

```
        - echo "$DOCKER_HUB_PAT" | docker login --username "$DOCKER_HUB_USER" --passw
```

```
        # Install curl and the Docker Scout CLI
```

```
        - |
```

```
          export DOCKER_BUILDKIT=0
```

```
          apk add --update curl
```

```
          curl -sSfL https://raw.githubusercontent.com/docker/scout-cli/main/install.s
```

```
          apk del curl
```

```
          rm -rf /var/cache/apk/*
```

```
        # Login to Docker Hub required for Docker Scout CLI
```

```
        - echo "$DOCKER_HUB_PAT" | docker login --username "$DOCKER_HUB_USER" --passw
```

```
        - |
```

```
          export DEVELOPMENT_BRANCH="main"
```

```
          if [[ "$BITBUCKET_BRANCH" == "$DEVELOPMENT_BRANCH" ]]; then # Bitbucket uses
```

```
            tag=":latest"
```

```
            echo "Running on default branch '$DEVELOPMENT_BRANCH': tag = 'latest'"
```

```
          else
```

```
            tag=":$BITBUCKET_COMMIT"
```

```
            echo "Running on branch '$BITBUCKET_BRANCH': tag = $tag"
```

```
          fi
```

```
        - docker build --pull -t "$CI_REGISTRY_IMAGE${tag}" .
```

```
        - |
```

```
          if [[ "$BITBUCKET_BRANCH" == "$DEVELOPMENT_BRANCH" ]]; then
```

```
            # Get a CVE report for the built image and fail the pipeline when critical
```

```
            docker scout cves "$CI_REGISTRY_IMAGE${tag}" --exit-code --only-severity c
```

```
          else
```

```
            # Compare image from branch with latest image from the default branch and
```

```

        docker scout compare "$CI_REGISTRY_IMAGE${tag}" --to "$CI_REGISTRY_IMAGE:1
    fi
- docker push "$CI_REGISTRY_IMAGE${tag}"

definitions:
  services:
    docker:
      memory: 2048 # Optional: Increase if needed

```

This example assumes two secrets to be available to authenticate against Docker Hub, called DOCKER_HUB_USER and DOCKER_HUB_PAT, also is necessary more two secrets called CI_REGISTRY, CI_REGISTRY_IMAGE about registry info.

License

The Docker Scout CLI is licensed under the Terms and Conditions of the Docker Subscription Service Agreement.

LICENSE

The Docker Scout CLI is released under the Terms and Conditions of the Docker Subscription Service Agreement. Please review the agreement at <https://www.docker.com/legal/docker-subscription-service-agreement/>

install.sh

```

#!/bin/sh

# note: we require errors to propagate (don't set -e)

# Copyright © 2023 Docker, Inc.

set -u

PROJECT_NAME="docker-scout"
OWNER=docker
REPO="scout-cli"
GITHUB_DOWNLOAD_PREFIX=https://github.com/${OWNER}/${REPO}/releases/download
INSTALL_SH_BASE_URL=https://raw.githubusercontent.com/${OWNER}/${REPO}
BINARY="docker-scout"
DOCKER_HOME=${DOCKER_HOME:~/.docker}
DEFAULT_INSTALL_DIR=${DOCKER_HOME}/cli-plugins
PROGRAM_ARGS=$@

# do not change the name of this parameter (this must always be backwards compatible)
DOWNLOAD_TAG_INSTALL_SCRIPT=${DOWNLOAD_TAG_INSTALL_SCRIPT:-true}

#
# usage [script-name]
#
usage() (
  this="$1"
  cat <<EOF
$this:download go binaries for ${OWNER}/${REPO}

```



```

Usage: $this [-b] dir [-d] [tag]
  -b the installation directory (defaults to ${DEFAULT_INSTALL_DIR})
  -d turns on debug logging
  -dd turns on trace logging
  [tag] the specific release to use (if missing, then the latest will be used)
EOF
  exit 2
)

# -----
# https://github.com/client9/shlib - portable posix shell functions
# Public domain - http://unlicense.org
# https://github.com/client9/shlib/blob/master/LICENSE.md
# but credit (and pull requests) appreciated.
# -----

is_command() (
  command -v "$1" >/dev/null
)

echo_stderr() (
  echo "$@" 1>&2
)

_logp=2
log_set_priority() {
  _logp="$1"
}

log_priority() (
  if test -z "$1"; then
    echo "$_logp"
    return
  fi
  [ "$1" -le "$_logp" ]
)

init_colors() {
  RED=''
  BLUE=''
  PURPLE=''
  BOLD=''
  RESET=''
  # check if stdout is a terminal
  if test -t 1 && is_command tput; then
    # see if it supports colors
    ncolors=$(tput colors)
    if test -n "$ncolors" && test "$ncolors" -ge 8; then
      RED='\033[0;31m'
      BLUE='\033[0;34m'
      PURPLE='\033[0;35m'
      BOLD='\033[1m'
      RESET='\033[0m'
    fi
  fi
}

init_colors

log_tag() (
  case "$1" in
    0) echo "${RED}${BOLD}[error]${RESET}" ;;

```

```

    1) echo "${RED}[warn]${RESET}" ;;
    2) echo "[info]${RESET}" ;;
    3) echo "${BLUE}[debug]${RESET}" ;;
    4) echo "${PURPLE}[trace]${RESET}" ;;
    *) echo "[$1]" ;;
esac
)

log_trace_priority=4
log_trace() (
    priority=$log_trace_priority
    log_priority "$priority" || return 0
    echo_stderr "$(log_tag $priority)" "${@}" "$RESET"
)

log_debug_priority=3
log_debug() (
    priority=$log_debug_priority
    log_priority "$priority" || return 0
    echo_stderr "$(log_tag $priority)" "${@}" "$RESET"
)

log_info_priority=2
log_info() (
    priority=$log_info_priority
    log_priority "$priority" || return 0
    echo_stderr "$(log_tag $priority)" "${@}" "$RESET"
)

log_warn_priority=1
log_warn() (
    priority=$log_warn_priority
    log_priority "$priority" || return 0
    echo_stderr "$(log_tag $priority)" "${@}" "$RESET"
)

log_err_priority=0
log_err() (
    priority=$log_err_priority
    log_priority "$priority" || return 0
    echo_stderr "$(log_tag $priority)" "${@}" "$RESET"
)

uname_os_check() (
    os="$1"
    case "$os" in
        darwin) return 0 ;;
        dragonfly) return 0 ;;
        freebsd) return 0 ;;
        linux) return 0 ;;
        android) return 0 ;;
        nacl) return 0 ;;
        netbsd) return 0 ;;
        openbsd) return 0 ;;
        plan9) return 0 ;;
        solaris) return 0 ;;
        windows) return 0 ;;
    esac
    log_err "uname_os_check '$(uname -s)' got converted to '$os' which is not a GOOS
value. Please file bug at https://github.com/client9/shlib"
    return 1
)

```

```

uname_arch_check() (
  arch="$1"
  case "$arch" in
    386) return 0 ;;
    amd64) return 0 ;;
    arm64) return 0 ;;
    armv5) return 0 ;;
    armv6) return 0 ;;
    armv7) return 0 ;;
    ppc64) return 0 ;;
    ppc64le) return 0 ;;
    mips) return 0 ;;
    mipsle) return 0 ;;
    mips64) return 0 ;;
    mips64le) return 0 ;;
    s390x) return 0 ;;
    amd64p32) return 0 ;;
  esac
  log_err "uname_arch_check '$(uname -m)' got converted to '$arch' which is not a
GOARCH value. Please file bug report at https://github.com/client9/shlib"
  return 1
)

```

```

unpack() (
  archive="$1"

  log_trace "unpack(archive=${archive})"

  case "$archive" in
    *.tar.gz | *.tgz) tar --no-same-owner -xzf "$archive" ;;
    *.tar) tar --no-same-owner -xf "$archive" ;;
    *.zip) unzip -q "$archive" ;;
    *.dmg) extract_from_dmg "$archive" ;;
    *)
      log_err "unpack unknown archive format for ${archive}"
      return 1
      ;;
  esac
)

```

```

extract_from_dmg() (
  dmg_file="$1"

  mount_point="/Volumes/tmp-dmg"
  hdiutil attach -quiet -nobrowse -mountpoint "$mount_point" "$dmg_file"
  cp -fR "${mount_point}/." ./
  hdiutil detach -quiet -force "$mount_point"
)

```

```

http_download_curl() (
  local_file="$1"
  source_url="$2"
  header="$3"

  log_trace "http_download_curl(local_file=$local_file, source_url=$source_url,
header=$header)"

  if [ -z "$header" ]; then
    code=$(curl -w '%[http_code]' -sL -o "$local_file" "$source_url")
  else
    code=$(curl -w '%[http_code]' -sL -H "$header" -o "$local_file" "$source_url")
  fi

  if ["$code" != "200" ]; then

```

```

        log_err "received HTTP status=$code for url='$source_url'"
        return 1
    fi
    return 0
)

http_download_wget() (
    local_file="$1"
    source_url="$2"
    header="$3"

    log_trace "http_download_wget(local_file=$local_file, source_url=$source_url,
header=$header)"

    if [ -z "$header" ]; then
        wget -q -O "$local_file" "$source_url"
    else
        wget -q --header "$header" -O "$local_file" "$source_url"
    fi
)

http_download() (
    log_debug "http_download(url=$2)"
    if is_command curl; then
        http_download_curl "$@"
        return
    elif is_command wget; then
        http_download_wget "$@"
        return
    fi
    log_err "http_download unable to find wget or curl"
    return 1
)

http_copy() (
    tmp=$(mktemp)
    http_download "$tmp" "$1" "$2" || return 1
    body=$(cat "$tmp")
    rm -f "$tmp"
    echo "$body"
)

hash_sha256() (
    TARGET=${1:-/dev/stdin}
    if is_command gsha256sum; then
        hash=$(gsha256sum "$TARGET") || return 1
        echo "$hash" | cut -d ' ' -f 1
    elif is_command sha256sum; then
        hash=$(sha256sum "$TARGET") || return 1
        echo "$hash" | cut -d ' ' -f 1
    elif is_command shasum; then
        hash=$(shasum -a 256 "$TARGET" 2>/dev/null) || return 1
        echo "$hash" | cut -d ' ' -f 1
    elif is_command openssl; then
        hash=$(openssl -dst openssl dgst -sha256 "$TARGET") || return 1
        echo "$hash" | cut -d ' ' -f a
    else
        log_err "hash_sha256 unable to find command to compute sha-256 hash"
        return 1
    fi
)

hash_sha256_verify() (
    TARGET="$1"

```

```

checksums="$2"
if [ -z "$checksums" ]; then
    log_err "hash_sha256_verify checksum file not specified in arg2"
    return 1
fi
BASENAME=${TARGET##*/}
want=$(grep "$BASENAME" "$checksums" 2>/dev/null | tr '\t' ' ' | cut -d ' ' -f 1)
if [ -z "$want" ]; then
    log_err "hash_sha256_verify unable to find checksum for '${TARGET}'
in '${checksums}'"
    return 1
fi
got=$(hash_sha256 "$TARGET")
if [ "$want" != "$got" ]; then
    log_err "hash_sha256_verify checksum for '$TARGET' did not verify ${want} vs $got"
    return 1
fi
)

# -----
# End of functions from https://github.com/client9/shlib
# -----

# asset_file_exists [path]
#
# returns 1 if the given file does not exist
#
asset_file_exists() (
    path="$1"
    if [ ! -f "$path" ]; then
        return 1
    fi
)

# github_release_json [owner] [repo] [version]
#
# outputs release json string
#
github_release_json() (
    owner="$1"
    repo="$2"
    version="$3"
    test -z "$version" && version="latest"
    giturl="https://github.com/${owner}/${repo}/releases/${version}"
    json=$(http_copy "$giturl" "Accept:application/json")

    log_trace "github_release_json(owner=${owner}, repo=${repo}, version=${version})
returned '${json}'"

    test -z "$json" && return 1
    echo "$json"
)

# extract_value [key-value-pair]
#
# outputs value from a colon delimited key-value pair
#
extract_value() (
    key_value="$1"
    IFS=':' read -r _ value << EOF
    ${key_value}
    EOF
    echo "$value"
)

```

```

)

# extract_json_value [json] [key]
#
# outputs value of the key from the given json string
#
extract_json_value() (
    json="$1"
    key="$2"
    key_value=$(echo "$json" | grep -o "\"$key\":[^,]*[,}\"" | tr -d ',}')

    extract_value "$key_value"
)

# github_release_tag [release-json]
#
# outputs release tag string
#
github_release_tag() (
    json="$1"
    tag=$(extract_json_value "$json" "tag_name")
    test -z "$tag" && return 1
    echo "$tag"
)

# download_github_release_checksums [release-url-prefix] [name] [version] [output-dir]
#
# outputs path to the downloaded checksums file
#
download_github_release_checksums() (
    download_url="$1"
    name="$2"
    version="$3"
    output_dir="$4"

    log_trace "download_github_release_checksums(url=${download_url}, name=${name},
version=${version}, output_dir=${output_dir})"

    checksum_filename=${name}_${version}_checksums.txt
    checksum_url=${download_url}/${checksum_filename}
    output_path="${output_dir}/${checksum_filename}"

    http_download "$output_path" "$checksum_url" ""
    asset_file_exists "$output_path"

    log_trace "download_github_release_checksums() returned '${output_path}'"

    echo "$output_path"
)

# search_for_asset [checksums-file-path] [name] [os] [arch] [format]
#
# outputs name of the asset to download
#
search_for_asset() (
    checksum_path="$1"
    name="$2"
    os="$3"
    arch="$4"
    format="$5"

    log_trace "search_for_asset(checksum-path=${checksum_path}, name=${name}, os=${os},
arch=${arch}, format=${format})"

```

```

asset_glob="${name}.*_${os}_${arch}.${format}"
output_path=$(grep -o "$asset_glob" "$checksum_path" || true)

log_trace "search_for_asset() returned '${output_path}'"

echo "$output_path"
)

# uname_os
#
# outputs an adjusted os value
#
uname_os() (
  os=$(uname -s | tr '[:upper:]' '[:lower:]')
  case "$os" in
    cygwin_nt*) os="windows" ;;
    mingw*) os="windows" ;;
    msys_nt*) os="windows" ;;
  esac

  uname_os_check "$os"

  log_trace "uname_os() returned '${os}'"

  echo "$os"
)

# uname_arch
#
# outputs an adjusted architecture value
#
uname_arch() (
  arch=$(uname -m)
  case "$arch" in
    x86_64) arch="amd64" ;;
    x86) arch="386" ;;
    i686) arch="386" ;;
    i386) arch="386" ;;
    aarch64) arch="arm64" ;;
    armv5*) arch="armv5" ;;
    armv6*) arch="armv6" ;;
    armv7*) arch="armv7" ;;
  esac

  uname_arch_check "$arch"

  log_trace "uname_arch() returned '${arch}'"

  echo "$arch"
)

# get_release_tag [owner] [repo] [tag]
#
# outputs tag string
#
get_release_tag() (
  owner="$1"
  repo="$2"
  tag="$3"

  log_trace "get_release_tag(owner=${owner}, repo=${repo}, tag=${tag})"

  json=$(github_release_json "$owner" "$repo" "$tag")
  real_tag=$(github_release_tag "$json")

```

```
    if test -z "$real_tag"; then
        return 1
    fi

    log_trace "get_release_tag() returned '${real_tag}'"

    echo "$real_tag"
)

# tag_to_version [tag]
#
# outputs version string
#
tag_to_version() (
    tag="$1"
    value="${tag#v}"

    log_trace "tag_to_version(tag=${tag}) returned '${value}'"

    echo "$value"
)

# get_binary_name [os] [arch] [default-name]
#
# outputs a the binary string name
#
get_binary_name() (
    os="$1"
    arch="$2"
    binary="$3"
    original_binary="$binary"

    case "$os" in
        windows) binary="${binary}.exe" ;;
    esac

    log_trace "get_binary_name(os=${os}, arch=${arch}, binary=${original_binary})
returned '${binary}'"

    echo "$binary"
)

# get_format_name [os] [arch] [default-format]
#
# outputs an adjusted file format
#
get_format_name() (
    os="$1"
    arch="$2"
    format="$3"
    original_format="$format"

    case "$os" in
        windows) format=zip ;;
    esac

    log_trace "get_format_name(os=${os}, arch=${arch}, format=${original_format})
returned '${format}'"

    echo "$format"
)

# download_and_install_asset [release-url-prefix] [download-path] [install-path] [name]
```



```

[os] [arch] [version] [format] [binary]
#
# attempts to download the archive and install it to the given path.
#
download_and_install_asset() (
    download_url="$1"
    download_path="$2"
    install_path="$3"
    name="$4"
    os="$5"
    arch="$6"
    version="$7"
    format="$8"
    binary="$9"

    asset_filepath=$(download_asset "$download_url" "$download_path" "$name" "$os"
"$arch" "$version" "$format")

    # don't continue if we couldn't download an asset
    if [ -z "$asset_filepath" ]; then
        log_err "could not find release asset for os='${os}' arch='${arch}'
format='${format}' "
        return 1
    fi

    install_asset "$asset_filepath" "$install_path" "$binary"
)

# download_asset [release-url-prefix] [download-path] [name] [os] [arch] [version]
[format] [binary]
#
# outputs the path to the downloaded asset asset_filepath
#
download_asset() (
    download_url="$1"
    destination="$2"
    name="$3"
    os="$4"
    arch="$5"
    version="$6"
    format="$7"

    log_trace "download_asset(url=${download_url}, destination=${destination},
name=${name}, os=${os}, arch=${arch}, version=${version}, format=${format})"

    checksums_filepath=$(download_github_release_checksums "$download_url" "$name"
"$version" "$destination")

    log_trace "checksums content:\n$(cat ${checksums_filepath})"

    asset_filename=$(search_for_asset "$checksums_filepath" "$name" "$os"
"$arch" "$format")

    # don't continue if we couldn't find a matching asset from the checksums file
    if [ -z "$asset_filename" ]; then
        return 1
    fi

    asset_url="${download_url}/${asset_filename}"
    asset_filepath="${destination}/${asset_filename}"
    http_download "$asset_filepath" "$asset_url" ""

    hash_sha256_verify "$asset_filepath" "$checksums_filepath"

```

```

log_trace "download_asset_by_checksums_file() returned '${asset_filepath}'"

echo "$asset_filepath"
)

# install_asset [asset-path] [destination-path] [binary]
#
install_asset() (
    asset_filepath="$1"
    destination="$2"
    binary="$3"

    log_trace "install_asset(asset=${asset_filepath}, destination=${destination},
binary=${binary})"

    # don't continue if we don't have anything to install
    if [ -z "$asset_filepath" ]; then
        return
    fi

    archive_dir=$(dirname "$asset_filepath")

    # unarchive the downloaded archive to the temp dir
    (cd "$archive_dir" && unpack "$asset_filepath")

    # create the destination dir
    test ! -d "$destination" && install -d "$destination"

    # install the binary to the destination dir
    install "${archive_dir}/${binary}" "${destination}/"
)

main() (
    # parse arguments

    # note: never change default install directory (this must always be
backwards compatible)
    install_dir=${install_dir:-${DEFAULT_INSTALL_DIR}}

    # note: never change the program flags or arguments (this must always be
backwards compatible)
    while getopts "b:dh?x" arg; do
        case "$arg" in
            b) install_dir="$OPTARG" ;;
            d)
                if [ "$logp" = "$log_info_priority" ]; then
                    # -d == debug
                    log_set_priority $log_debug_priority
                else
                    # -dd (or -ddd...) == trace
                    log_set_priority $log_trace_priority
                fi
            ;;
            h | \?) usage "$0" ;;
            x) set -x ;;
        esac
    done
    shift $((OPTIND - 1))
    set +u
    tag="$1"

    if [ "$install_dir" = "$DEFAULT_INSTALL_DIR" ]; then
        if [ ! -d "$DOCKER_HOME" ]; then
            log_err "docker is not installed; refusing to install to '${install_dir}'"

```

```

        exit 1
    fi
fi

if [ -z "$tag" ]; then
    log_debug "checking github for the current release tag"
    tag=""
else
    log_debug "checking github for release tag='${tag}'"
fi
set -u

tag=$(get_release_tag "$OWNER" "$REPO" "$tag")

if [ "$?" != "0" ]; then
    log_err "unable to find tag='${tag}'"
    log_err "do not specify a version or select a valid version from
https://github.com/${OWNER}/${REPO}/releases"
    return 1
fi

# run the application

version=$(tag_to_version "$tag")
os=$(uname_os)
arch=$(uname_arch)
format=$(get_format_name "$os" "$arch" "tar.gz")
binary=$(get_binary_name "$os" "$arch" "$BINARY")
download_url="${GITHUB_DOWNLOAD_PREFIX}/${tag}"

# we always use the install.sh script that is associated with the tagged release.
Why? the latest install.sh is not
# guaranteed to be able to install every version of the application. We use the
DOWNLOAD_TAG_INSTALL_SCRIPT env var
# to indicate if we should continue processing with the existing script or to
download the script from the given tag.
if [ "$DOWNLOAD_TAG_INSTALL_SCRIPT" = "true" ]; then
    export DOWNLOAD_TAG_INSTALL_SCRIPT=false
    log_info "fetching release script for tag='${tag}'"
    http_copy "${INSTALL_SH_BASE_URL}/${tag}/install.sh" "" | sh -s -
- ${PROGRAM_ARGS}
    exit "$?"
fi

log_info "using release tag='${tag}' version='${version}' os='${os}' arch='${arch}'"

download_dir=$(mktemp -d)
trap 'rm -rf -- "$download_dir"' EXIT

log_debug "downloading files into ${download_dir}"

download_and_install_asset "$download_url" "$download_dir" "$install_dir"
"$PROJECT_NAME" "$os" "$arch" "$version" "$format" "$binary"

# don't continue if we couldn't install the asset
if [ "$?" != "0" ]; then
    log_err "failed to install ${BINARY}"
    return 1
fi

log_info "installed ${install_dir}/${binary}"
)

# entrypoint

```

```
set +u
if [ -z "$TEST_INSTALL_SH" ]; then
  set -u
  main "$@"
fi
set -u
}
```

The End

This PDF was generated by gitprint.me