```bash
#!/usr/bin/env bash

# This is the bootstrap Unix installer served by
`https://get.volta.sh`.
# Its responsibility is to query the system to
determine what OS the system
# has, fetch and install the appropriate build of
Volta, and modify the user's
# profile.

# NOTE: to use an internal company repo, change how
this determines the latest version
get_latest_release() {
  curl --silent "https://volta.sh/latest-version"
}

release_url() {
  echo "https://github.com/volta-
cli/volta/releases"
}

download_release_from_repo() {
  local version="$1"
  local os_info="$2"
  local tmpdir="$3"

  local filename="volta-$version-$os_info.tar.gz"
  local download_file="$tmpdir/$filename"
  local
archive_url="$(release_url)/download/v$version/$fil
ename"

  curl --progress-bar --show-error --location --
fail "$archive_url" --output "$download_file" --
write-out "$download_file"
}

usage() {
    cat >&2 <<END_USAGE
volta-install: The installer for Volta

USAGE:
    volta-install [FLAGS] [OPTIONS]

FLAGS:
    -h, --help                  Prints help
information

OPTIONS:
        --dev                   Compile and install
Volta locally, using the dev target
        --release               Compile and install
Volta locally, using the release target
        --skip-setup            Do not run 'volta
setup' to modify startup scripts
        --version <version>     Install a specific
release version of Volta
END_USAGE
}

info() {
  local action="$1"
  local details="$2"
  command printf '\033[1;32m%12s\033[0m %s\n'
"$action" "$details" 1>&2
}

error() {
  command printf '\033[1;31mError\033[0m: %s\n\n'
"$1" 1>&2
}

warning() {
  command printf '\033[1;33mWarning\033[0m: %s\n\n'
"$1" 1>&2
}

request() {
  command printf '\033[1m%s\033[0m\n' "$1" 1>&2
}

eprintf() {
  command printf '%s\n' "$1" 1>&2
}

bold() {
  command printf '\033[1m%s\033[0m' "$1"
}

# check for issue with VOLTA_HOME
# if it is set, and exists, but is not a directory,
the install will fail
volta_home_is_ok() {
  if [ -n "${VOLTA_HOME-}" ] && [ -e "$VOLTA_HOME"
] && ! [ -d "$VOLTA_HOME" ]; then
    error "\$VOLTA_HOME is set but is not a
directory ($VOLTA_HOME)."
    eprintf "Please check your profile scripts and
environment."
    return 1
  fi
  return 0
}

# Check if it is OK to upgrade to the new version
upgrade_is_ok() {
  local will_install_version="$1"
  local install_dir="$2"
  local is_dev_install="$3"

  # check for Volta in both the old location and
the new 0.7.0 location
  local volta_bin="$install_dir/volta"
  if [ ! -x "$volta_bin" ]; then
    volta_bin="$install_dir/bin/volta"
  fi

  # this is not able to install Volta prior to
0.5.0 (when it was renamed)
  if [[ "$will_install_version" =~ ^([0-9]+\.[0-
9]+) ]]; then
    local major_minor="${BASH_REMATCH[1]}"
    case "$major_minor" in
      0.1|0.2|0.3|0.4|0.5)
        eprintf ""
        error "Cannot install Volta prior to
version 0.6.0"
        request "    To install Volta version
$will_install_version, please check out the source
and build manually."
        eprintf ""
        return 1
        ;;
    esac
  fi

  if [[ -n "$install_dir" && -x "$volta_bin" ]];
then
    local prev_version="$( ($volta_bin --version
2>/dev/null || echo 0.1) | sed -E 's/^.*([0-9]+\.
[0-9]+\.[0-9]+).*$/\1/')"
    # if this is a local dev install, skip the
equality check
    # if installing the same version, this is a no-
op
    if [ "$is_dev_install" != "true" ] && [
"$prev_version" == "$will_install_version" ]; then
      eprintf "Version $will_install_version
already installed"
      return 1
    fi
    # in the future, check $prev_version for
incompatible upgrades
  fi
  return 0
}

# returns the os name to be used in the packaged
release
parse_os_info() {
  local uname_str="$1"
  local arch="$(uname -m)"

  case "$uname_str" in
    Linux)
      if [ "$arch" == "x86_64" ]; then
        echo "linux"
      else
        error "Releases for non x64 architectures
are not currently supported."
        return 1
      fi
      ;;
    Darwin)
      if [ "$(uname -m)" == "arm64" ]; then
        echo "macos-aarch64"
      else
        echo "macos"
      fi
      ;;
    *)
      return 1
  esac
  return 0
}

parse_os_pretty() {
  local uname_str="$1"

  case "$uname_str" in
    Linux)
      echo "Linux"
      ;;
    Darwin)
      echo "macOS"
      ;;
    *)
      echo "$uname_str"
  esac
}

# return true(0) if the element is contained in the
input arguments
# called like:
#  if element_in "foo" "${array[@]}"; then ...
element_in() {
  local match="$1";
  shift

  local element;
  # loop over the input arguments and return when a
match is found
  for element in "$@"; do
    [ "$element" == "$match" ] && return 0
  done
  return 1
}

create_tree() {
  local install_dir="$1"

  info 'Creating' "directory layout"

  # .volta/
  #     bin/

  mkdir -p "$install_dir" && mkdir -p
"$install_dir"/bin
  if [ "$?" != 0 ]
  then
    error "Could not create directory layout.
Please make sure the target directory is writeable:
$install_dir"
    exit 1
  fi
}

install_version() {
  local version_to_install="$1"
  local install_dir="$2"
  local should_run_setup="$3"

  if ! volta_home_is_ok; then
    exit 1
  fi

  case "$version_to_install" in
    latest)
      local latest_version="$(get_latest_release)"
      info 'Installing' "latest version of Volta
($latest_version)"
      install_release "$latest_version"
"$install_dir"
      ;;
    local-dev)
      info 'Installing' "Volta locally after
compiling"
      install_local "dev" "$install_dir"
      ;;
    local-release)
      info 'Installing' "Volta locally after
compiling with '--release'"
      install_local "release" "$install_dir"
      ;;
    *)
      # assume anything else is a specific version
      info 'Installing' "Volta version
$version_to_install"
      install_release "$version_to_install"
"$install_dir"
      ;;
  esac

  if [ "$?" == 0 ]
  then
      if [ "$should_run_setup" == "true" ]; then
        info 'Finished' "installation. Updating
user profile settings."
        "$install_dir"/bin/volta setup
      else
        "$install_dir"/bin/volta --version
&>/dev/null # creates the default shims
        info 'Finished' "installation. No changes
were made to user profile settings."
      fi
  fi
}

# parse the 'version = "X.Y.Z"' line from the input
Cargo.toml contents
# and return the version string
parse_cargo_version() {
  local contents="$1"

  while read -r line
  do
    if [[ "$line" =~ ^version\ =\ \"(.*)\" ]]
    then
      echo "${BASH_REMATCH[1]}"
      return 0
    fi
  done <<< "$contents"

  error "Could not determine the current version
from Cargo.toml"
  return 1
}

install_release() {
  local version="$1"
  local install_dir="$2"
  local is_dev_install="false"

  info 'Checking' "for existing Volta installation"
  if upgrade_is_ok "$version" "$install_dir"
"$is_dev_install"
  then
    download_archive="$(download_release
"$version"; exit "$?")"
    exit_status="$?"
    if [ "$exit_status" != 0 ]
    then
      error "Could not download Volta version
'$version'. See $(release_url) for a list of
available releases"
      return "$exit_status"
    fi

    install_from_file "$download_archive"
"$install_dir"
  else
    # existing legacy install, or upgrade problem
    return 1
  fi
}

install_local() {
  local dev_or_release="$1"
  local install_dir="$2"
  # this is a local install, so skip the version
equality check
  local is_dev_install="true"

  info 'Checking' "for existing Volta installation"
  install_version="$(parse_cargo_version
"$(<Cargo.toml)" )" || return 1
  if upgrade_is_ok "$install_version"
"$install_dir" "$is_dev_install"
  then
    # compile and package the binaries, then
install from that local archive
    compiled_archive="$(compile_and_package
"$dev_or_release")" &&
      install_from_file "$compiled_archive"
"$install_dir"
  else
    # existing legacy install, or upgrade problem
    return 1
  fi
}

compile_and_package() {
  local dev_or_release="$1"

  local release_output

  # get the directory of this script
  # (from https://stackoverflow.com/a/246128)
  DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )"
>/dev/null 2>&1 && pwd )"

  # call the release script to create the packaged
archive file
  # '2> >(tee /dev/stderr)' copies stderr to
stdout, to collect it and parse the filename
  release_output="$( "$DIR/release.sh" "--
$dev_or_release" 2> >(tee /dev/stderr) )"
  [ "$?" != 0 ] && return 1

  # parse the release filename and return that
  if [[ "$release_output" =~ release\ in\ file\
(target[^\ ]+) ]]; then
    echo "${BASH_REMATCH[1]}"
  else
    error "Could not determine output filename"
    return 1
  fi
}

download_release() {
  local version="$1"

  local uname_str="$(uname -s)"
  local os_info
  os_info="$(parse_os_info "$uname_str")"
  if [ "$?" != 0 ]; then
    error "The current operating system
($uname_str) does not appear to be supported by
Volta."
    return 1
  fi
  local pretty_os_name="$(parse_os_pretty
"$uname_str")"

  info 'Fetching' "archive for $pretty_os_name,
version $version"
  # store the downloaded archive in a temporary
directory
  local download_dir="$(mktemp -d)"
  download_release_from_repo "$version" "$os_info"
"$download_dir"
}

install_from_file() {
  local archive="$1"
  local install_dir="$2"

  create_tree "$install_dir"

  info 'Extracting' "Volta binaries and launchers"
  # extract the files to the specified directory
  tar -xf "$archive" -C "$install_dir"/bin
}

check_architecture() {
  local version="$1"
  local arch="$2"

  if [[ "$version" != "local"* ]]; then
    case "$arch" in
      x86_64)
        return 0
        ;;
      arm64)
        if [ "$(uname -s)" = "Darwin" ]; then
          return 0
        fi
        ;;
    esac

    error "Sorry! Volta currently only provides
pre-built binaries for x86_64 architectures."
    return 1
  fi
}


# return if sourced (for testing the functions
above)
return 0 2>/dev/null

# default to installing the latest available
version
version_to_install="latest"

# default to running setup after installing
should_run_setup="true"

# install to VOLTA_HOME, defaulting to ~/.volta
install_dir="${VOLTA_HOME:-"$HOME/.volta"}"

# parse command line options
while [ $# -gt 0 ]
do
  arg="$1"

  case "$arg" in
    -h|--help)
      usage
      exit 0
      ;;
    --dev)
      shift # shift off the argument
      version_to_install="local-dev"
      ;;
    --release)
      shift # shift off the argument
      version_to_install="local-release"
      ;;
    --version)
      shift # shift off the argument
      version_to_install="$1"
      shift # shift off the value
      ;;
    --skip-setup)
      shift # shift off the argument
      should_run_setup="false"
      ;;
    *)
      error "unknown option: '$arg'"
      usage
      exit 1
      ;;
  esac
done

check_architecture "$version_to_install" "$(uname -
m)" || exit 1

install_version "$version_to_install"
"$install_dir" "$should_run_setup"
```