This document proposes a initial reference implementation of `archupgrade`, a software package that aims to provide reliable, predictable versioning upgrades for Arch Linux-based distributions

1. Overview

`archupgrade` system requirements

For system builders:

- A build system using `archupgrade` to perform image-based, read-only system image preparation for use in deployment **should not** include `archupgrade` in the final image. Upgrades **should** be handled by the builder's choice of version control/imaging solution.

For client systems:

- A system that will be using `archupgrade` to mandate system updates **shall not** have any user-accessible means of installing packages into system repositories by default. This is, by design, so that the system states remain deterministic before and after an update is executed.
- Any such utilities that the user expects **shall** be symlinked towards the `archupgrade` binary. This is so `archupgrade` can inform the user appropriately when there is an attempt to execute such an utility.

`archupgrade` specification requirements

- A specification for `archupgrade` **shall** includes every single packages that will be installed on a given system. Do not assume any set of base packages will be available or will be installed as dependancies
- A specification for `archupgrade` **should** contain multiple phases, when applicable. After installation of any given phase, the system being services **should** remain fully-functional, minimizing the possiblity of an unexpected error during the following phases causing complete system failures.

`archupgrade` implementations specfics

- An implementation of `archupgrade` **should** be able to be executed without using any system-dependant libraries, ie. the implementation's binary should have all of its dependencies statically linked.
- An implementation of `archupgrade` **shall** be able to parse every part of a given specification, unless said part is marked as "optional"

2. Specification file format

We propose using a machine-parsable format like `json` or `yaml` for the specification file format, which shall specifies as follows:

Versioning

A specification **shall** declare a version of the specification file format. This is required in order to enforce a minimum version for any given implementation that shall be used in order to apply said specification successfully.

The versioning **should** follow Semantic Versioning. For example:

```
version: '0.0.1-alpha'
```

Nesting specifications

A specification may choose to include other specifications as part of itself. This mechanism allows more modularity and reusability between different shipping configurations.

This is implemented as an array

```
includes:
    - path/to/specification.yml
```

Specifications specifics

- A specification **may** nests any arbitrary amount of other specifications. Each nested specifications may also themselves nests more specifications.
- A nested specification **shall** always be executed before its host.
- If multiple sub-specifications nests the same specification file, ie.

```
# main.yml
includes:
    - specification_a.yml
    - specification_b.yml
# specification_a.yml
includes:
    - specification_c.yml
# specification_b.yml
includes:
    - specification_c.yml
```

said nested specifications **shall** only get executed once at the point where they are first imported (in this example, it gets executed as part of `specification_a`)

Implementation specifics

- An implementation **shall** execute every nested specifications in the order that they are imported in

- An implementation **shall** attempt to detect and abort if nesting loops are detected in nested specifications.

Package installation and configurations

An upgrade as specified by `archupgrade` shall run in multiple phases to account for any inconsistencies that may arise from upgrading from older package revisions. For example, it should handle anything on the Arch Linux News that would normally requires manual intervention (for example, JDK 21).

While not required, it is recommended that that each phase

For this purpose, we propose the following format:

```
upgrade:
    phases:
        first:
            backend: libalpm
            message: "Upgrading system libraries..."
            preinstall:
                - some_bash_commands
                - do_some_patching
            packages:
                - package_a:
                    url:
'https://archive.archlinux.org/p/package_a.tar.zst'
                    hash: 'dQw4w9WgXcQ'
                    hash-algorithm: 'sha256'
                - package_b:
                    url:
'https://archive.archlinux.org/p/package_b.tar.zst'
                    hash: '_xc7tNbjnHM'
                    hash-algorithm: 'sha256'
            postinstall:
                - some_more_bash_commands
                - do_some_configurations
            reboot: false
        second:
            backend: libalpm
            message: "Upgrading desktop environment..."
            preinstall:
                - some_bash_commands
                - do_some_patching
            packages:
                - package_c:
                    url:
'https://archive.archlinux.org/p/package_c.tar.zst'
                    hash: 'dQw4w9WgXcQ'
                    hash-algorithm: 'sha256'
                - package_d:
                    url:
'https://archive.archlinux.org/p/package_d.tar.zst'
                    hash: '_xc7tNbjnHM'
```

```
                hash-algorithm: 'sha256'
        postinstall:
            - some_more_bash_commands
            - do_some_configurations
        reboot: true
```

Directives

- `required-space`: Specifies an amount of space that would be necessary on the `root` directory of the target system for the upgrade to be performed successfully
- `phases`: Specifies an array of phases that the update shall be performed in, each of which includes:
    - `backend`: Specifies the backend that would be used to perform this phase of the upgrade`
    - `message`: A message that shall be displayed to the user during the phase of the upgrade
    - `preinstall`: An array of shell command that should be run before packages in this phase are installed
    - `packages`: An array of packages that shall be installed on the system. Each `package` is an object that specifies
        - `url`: A URI to the package archive itself. This may be `https://` or `file://` for remote and local archives, respectively.
        - `hash`: A hash of the package archive.
        - `hash-algorithm`: The algorithm used to generate the package's hash
    - `postinstall`: An array of shell command that should be run after packages in this phase are installed
    - `reboot`: A boolean specifying whether or not a system reboot is required before continuing with the next phase.

Specifications specifics

- A single specification **shall not** have phases with completely matching names.
- Packages declared in any given phase **shall** provide a hash in order to protect the integrity of the downloaded package.

Implementation specifics

- An implementation **shall** perform installation phases in order that they are specified. Packages within the same phases **may** be installed together.
- An implementation **shall** execute the `preinstall` and `postinstall` commands in their specific order before and after package installation, respectively.

- An implementation **shall** display the message as specified by `message` to the user to indicate the upgrade progress.
- An implementation **shall** perform hash calculation of the packages after obtaining them from remote or local sources.
- An implementation **may** attempt to re-download a given package archive which failed verification for a fixed number of retries. If the amount of retries exceed said fixed amounts, the implementation **shall** perform deletion of the archive to prevent accidental reuse.
- An implementation **shall** save its progress in persistent storage, in case of a temporary failure causing the upgrade process to be restarted.
- An implementation **should**, when possible, use the installed/total package counts along with the finished/total upgrade phases to provide the user with an approximate progress of installation
- An implementation **may**, when possible, attempt to perform estimations of the user system's state (eg. required disk space) that may affect the ability to complete an upgrade.

Finalization

After an upgrade is performed by an `archupgrade` implementation, an optional finalization procedure may be performed to facilitates any post-upgrade configurations, as well as housekeeping items like cleaning up package caches or deprecated package removal.

```
finalize:
    shell:
        - some_shell_command
    file_write:
        - path: /etc/os-release
          content: >
            VERSION="10.0 (Firefly)"
            VERSION_ID=10.0
            VERSION_CODENAME=firefly
            BUILD_ID=20240618
            IMAGE_ID=firefly-shipping-final
    file_remove:
        - /etc/pacman.d/mirrorlist
    clean-caches: true
    reboot: true
```

Directives

- `shell`: Shell commands that shall be executed after installation has completed
- `file_write`: Files that should be written to the target filesystem after installation
- `file_remove`: Files that should be removed from the target filesystem after installation

- `clean-caches`: Whether or not the implementation should remove the package cache after installation
- `reboot`: Whether a system reboot shall be issued after installation.

Implementation specifics

- An implementation **shall** run the `shell` in the order that they appears in.
- An implementation **shall** perform writing and deletion of files in the order that they appears in. If a file already exist, the implementation **shall** overwrite said file.
- An implementation **shall** clean the package cache of every backend if they exist and has been in use when `clean-caches` is specified.