Release Strategy

Unified versioning and release management

Current Release Strategy

Uses release-please for automated releases

Triggered on every push to main

Creates release PRs automatically

Current Versioning

Version managed in package.json

Controls, firmware, and explorer have independent versions

Main branch releases use `-beta` suffix

Example: `1.25.0-beta`

Current Release Process

1. Developers merge PRs to main
2. release-please analyzes commits
3. Opens/updates a release PR
4. Merge release PR to create release
5. Release creation triggers branch workflow
6. Creates `release/X.Y.x` branch

Current Branch Creation

When a prerelease is published:

- Extracts major.minor from version
- Creates branch `release/X.Y.x`
- Commits with `Release-As: X.Y.0`
- Bumps to stable version on branch

Current Changelog

Automatically generated from conventional commits

Groups by commit type (feat, fix, etc.)

Links to commits and Pull Requests

Updated with each release PR

Current Hotfix Workflow

Automated via release-please

Runs on push to release/** branches

Always bumps patch version on release branches

Creates release PRs on release branches

Current Release Branch Behavior

Separate release-please configuration

Uses always-bump-patch versioning

No manual version control needed

Relies on conventional commits

Current Challenges

Independent versioning per workspace creates confusion

Package.json: 1.25.0-beta

Controls: workspace shared version

Firmware: 0.1.0

Explorer: 1.0.0

Difficult to track which components work together

Benefits Worth Preserving

From current approach:

Automated release process

Conventional commit based

Changelog generation included

Separate workflows for main and release branches

New Release Strategy

Unified versioning across the entire suite

All workspaces share the same version number

Version synced across all Cargo.toml files and package.json

Key Principles

Tags mark specific commits as releases on main

Release branches created from tagged commits

Clear traceability: tag → commit → release

Explicit control over version bumps

The makeline-release Tool

Custom tool that handles releases reliably

Just commands wrap each step for convenience

Release Process

1. Bump all version strings uniformly
2. Update the changelog
3. Commit changes
4. Tag that commit (e.g., `v1.2.0`)
5. Create release branch (e.g., `release/hyphenx-v1.2.x`)
6. Publish release to GitHub

Hotfix Process

To apply hotfixes to release branches:

1. Cherry-pick the SHA of the commit
2. Bump the patch version
3. Publish release to GitHub

Bump Version

```
# Update version in all 5 places
# And update CHANGELOG.md

just bump-minor-version  # 1.25.0 → 1.26.0
just bump-major-version  # 1.25.0 → 2.0.0
```

Create Release Branch

```
# Tag current commit on main
# Create/switch to release branch

just create-release-branch       # release/1.26.x

# Or with a specific suffix
just create-release-branch beta  # release/1.26.x-beta
```

Publish Release

```
# Publish the release to GitHub

just publish-release  # Creates hyphenx-v1.26.x
```

Apply Hotfixes

```
# Find commits to backport
git log main --oneline

# Apply hotfix (bumps patch version)
just hotfix <sha>

# Publish updated release
just publish-release
```

Dry Run Mode

All commands support dry run:

```
just bump-minor-version-dry
just bump-major-version-dry
just create-release-branch-dry
just publish-release-dry
```

Changelog Management

CHANGELOG.md is automatically updated during version bump

Uses git-cliff for changelog generation

No manual changelog editing required

Executable Validation

Generate executables.json with SHA256 hashes

Includes suite version from workspace

Hash all release binaries automatically

Command: `just generate-hashes`

Validation Widget

Explorer has a Validation widget:

- File picker for executables.json
- Directory picker for binaries
- Validates each executable against manifest
- Shows PASS/FAIL/MISSING/ERROR status
- Table view with expected vs actual hashes
- Color-coded results for quick scanning

Validation Workflow

Build binaries from tagged commit

Generate executables.json with hashes

Use explorer widget to validate

Confirms binaries match the release

Firmware Artifacts

Embedded firmware for lift/cabinet screens

Built for RP2040 microcontroller

Multiple artifacts generated:

- firmware.hex, firmware.uf2
- stage3.hex, stage3.uf2 (bootloader)
- stage4.hex, stage4.uf2 (bootloader)
- merged.hex (combined)

Firmware Upload to Release

On release published:

Build embedded firmware artifacts

Upload to GitHub Actions artifacts

Attach all .hex and .uf2 files to release

Available for download with release

Greengrass Components

Rust binaries for IoT edge devices

Cross-compiled for ARM architecture

Components deployed:

- controls-bridge
- batch-telemetry-uploader
- makeline-ui
- orderitem-uploader
- All Rust control system binaries

Greengrass Deployment

On release published or push to release/** branches:

Compile Rust binaries for ARM

Deploy to AWS IoT Greengrass

Targets dev, staging, and prod environments

Version stamped from package.json

CI Integration

Publishing with `release/**` branch triggers:

- Firmware artifact build and upload
- Greengrass component compilation
- Greengrass component deployment
- Artifact uploading to GitHub release

Everything continues to work!

Key Differences

Old: Independent versions per workspace

New: Single unified version

Old: release-please manages everything

New: Explicit control via makeline-release tool

Old: release-please for changelog

New: git-cliff for changelog

Both: Conventional commits required

New Approach Benefits

Preserves current benefits:

- Automated changelog generation
- Conventional commit based
- Separate main/release workflows

Adds new capabilities:

- Unified semantic versioning
- Explicit version control
- Improved hotfix workflow
- Tags on main branch commits

Questions?