Documentation Versioning

This document outlines the versioning strategy for UtilityFog-Fractal-TreeOpen documentation.

Version Structure

Documentation follows the same semantic versioning as the codebase:

- **Latest**: Always points to the most recent stable release (`/latest` → `/v0.1.0`)
- **Versioned**: Specific version documentation (`/v0.1.0`, `/v0.2.0`, etc.)
- **Next**: Development version from main branch (`/next`)

URL Structure

```
https://goldislops.github.io/UtilityFog-Fractal-TreeOpen/
├── latest/          # Redirects to current stable (v0.1.0)
├── v0.1.0/         # Version 0.1.0 documentation
├── v0.2.0/         # Version 0.2.0 documentation (future)
└── next/           # Development documentation (main branch)
```

Version Management

Current Stable (v0.1.0)

- **Path**: `/latest` and `/v0.1.0`
- **Source**: `release/0.1.x` branch
- **Content**: Stable, tested documentation
- **Updates**: Bug fixes and clarifications only

Development (next)

- **Path**: `/next`
- **Source**: `main` branch
- **Content**: Latest features and changes
- **Updates**: Continuous integration from main

Future Versions

- **Path**: `/v0.2.0`, `/v0.3.0`, etc.
- **Source**: Respective release branches
- **Content**: Version-specific documentation
- **Updates**: Maintained during active support period

Documentation Deployment

Automated Deployment

Documentation is automatically deployed via GitHub Actions:

```yaml
# .github/workflows/docs.yml
on:
  push:
    branches: [main, 'release/*']
    tags: ['v*']
```

Manual Deployment

For manual deployment or testing:

```
# Build documentation
mkdocs build

# Deploy to gh-pages
mkdocs gh-deploy --config-file mkdocs.yml
```

Version Configuration

MkDocs Configuration

Each version has its own `mkdocs.yml` configuration:

```yaml
# mkdocs.yml
site_name: UtilityFog-Fractal-TreeOpen
site_url: https://goldislops.github.io/UtilityFog-Fractal-TreeOpen/

# Version-specific settings
extra:
  version:
    provider: mike
    default: latest
```

Version Switching

Users can switch between versions using:

1. **Navigation menu**: Version selector in header
2. **Direct URLs**: Bookmark specific version URLs
3. **Redirects**: `/latest` always points to current stable

Content Guidelines

Version-Specific Content

- **API Changes**: Document breaking changes clearly
- **Feature Additions**: Mark new features with version badges
- **Deprecations**: Include deprecation warnings with timeline
- **Migration Guides**: Provide upgrade instructions between versions

Cross-Version Consistency

- **Navigation**: Maintain consistent menu structure
- **Styling**: Use same theme across versions
- **Search**: Enable search within each version

- **Links**: Use relative links within same version

Maintenance Policy

Active Versions

- **Latest Release**: Full maintenance and updates
- **Previous Major**: Security fixes and critical bugs
- **Development**: Continuous updates from main branch

Archived Versions

- **Read-only**: No further updates
- **Available**: Remain accessible for reference
- **Marked**: Clearly labeled as archived

End-of-Life

Versions reach end-of-life when:
- Two major versions behind current
- No longer supported by maintainers
- Security vulnerabilities cannot be patched

Implementation Checklist

For New Releases

- [ ] Create release branch documentation
- [ ] Update version references
- [ ] Test all links and examples
- [ ] Deploy versioned documentation
- [ ] Update `/latest` redirect
- [ ] Announce new documentation version

For Development

- [ ] Keep `/next` updated with main branch
- [ ] Document new features as they're added
- [ ] Maintain compatibility with existing versions
- [ ] Test documentation builds in CI

Tools and Scripts

Version Management Scripts

```
# scripts/docs-version.sh
#!/bin/bash
VERSION=$1
mike deploy --push --update-aliases $VERSION latest
mike set-default --push latest
```

Build Verification

```
# scripts/docs-check.sh
#!/bin/bash
mkdocs build --strict
linkchecker site/
```

Migration Between Versions

User Migration

When upgrading between versions, users should:

1. Check the changelog for breaking changes
2. Review migration guides in documentation
3. Test changes in development environment
4. Update bookmarks to new version URLs

Content Migration

When creating new version documentation:

1. Copy from previous version
2. Update version-specific content
3. Add new features and changes
4. Remove deprecated content
5. Update all internal links

This versioning strategy ensures users always have access to accurate, version-appropriate documentation while maintaining a clear upgrade path.