Gen-changelog

Gen-changelog is a release tool that generates changelogs in the Keep a Changelog format. It analyses a repository's commit history and uses conventional commit types to categorise and filter commits for inclusion in the changelog.

- Gen-changelog
  - Main Features
  - Gen-changelog CLI
    - Installation
    - Overview
    - Commands
      - `generate` - Generate Changelog
        - Options
        - Examples
      - `config` - Configuration Management
        - Options
        - Examples
    - Configuration File
    - How It Works
    - Conventional Commit Support
    - Logging
    - Getting Help
  - Gen-changelog Library Documentation
    - Installation
    - Default Configuration
      - Default Groups
    - Configuration File
    - Usage Examples
      - Basic Usage
      - Custom Configuration
      - Release Preparation
    - Requirements

Main Features

- **Commit Categorization**: Uses Conventional Commits specification to automatically categorise commits and filter them for changelog inclusion
- **Summary Counts**: Displays summary counts for each commit category in releases, including uncategorised (non-conventional) commits
- **Detailed Commit Summaries**: Shows commit details for Added, Fixed, Changed, and Security categories
- **Security Classification**: Automatically classifies commits made to the dependency scope as Security commits, regardless of their conventional commit type

- **Flexible Configuration**: Configurable mapping of commit types to headings, customizable heading display options, and optional commit summary counts

Gen-changelog CLI

A command-line tool that generates changelogs from git commits using conventional commit messages and keep-a-changelog formatting.

Installation

Install gen-changelog using Cargo:

```
cargo install gen-changelog
```

Overview

Gen-changelog CLI automatically generates changelogs by analysing your git commit history. It uses conventional commit patterns to categorize changes and outputs them in a format compatible with Keep a Changelog.

```
gen-changelog --help
```

```
Generate a change log based on the git commits compatible
with keep-a-changelog and using conventional commits to categorise commits.

Usage: gen-changelog [OPTIONS] [COMMAND]

Commands:
  generate  Generate changelog from git commits
  config    Manage configuration settings
  help      Print this message or the help of the given subcommand(s)

Options:
  -v, --verbose...  Increase logging verbosity
  -q, --quiet...    Decrease logging verbosity
  -h, --help        Print help
  -V, --version     Print version
```

Commands

`generate` - Generate Changelog

Creates a changelog file based on your repository's commit history.

```
gen-changelog generate [OPTIONS]
```

Options

Option	Description	Default
`-n, --next-version <VERSION>`	Version number for unreleased changes	-
`-s, --sections <NUMBER>`	Number of version sections to include in changelog	All
`-c, --config-file <FILE>`	Path to configuration file	-
`-r, --repo-dir <PATH>`	Path to git repository	`.` (current directory)
`-d, --display-summaries`	Show commit summaries in output	-
`--add-groups <GROUPS>`	Include additional commit type groups	-
`--remove-groups <GROUPS>`	Exclude specific commit type groups	-

Examples

Generate a changelog for the current repository:

```
gen-changelog generate
```

Generate with a specific next version:

```
gen-changelog generate --next-version "2.1.0"
```

Limit to the last 3 releases and show commit summaries:

```
gen-changelog generate --sections 3 --display-summaries
```

`config` - Configuration Management

Manage configuration settings for gen-changelog.

```
gen-changelog config [OPTIONS]
```

Options

Option	Description	Default
`-s, --save`	Save current configuration to file	-
`-f, --file <FILE>`	Configuration file name	`gen-changelog.toml`
`-p, --show`	Display current configuration	-

Examples

Show the current configuration:

```
gen-changelog config --show
```

Save configuration to the default file:

```
gen-changelog config --save
```

Save configuration to a custom file:

```
gen-changelog config --save --file my-config.toml
```

Configuration File

Gen-changelog CLI uses a TOML configuration file to customize its behaviour. The default configuration file is `gen-changelog.toml` in your project root.

To generate a configuration file with default settings and helpful comments:

```
gen-changelog config --save
```

How It Works

1. **Analyses Git History**: Scans your repository's commit messages
2. **Applies Conventional Commits**: Categorizes commits based on conventional commit patterns (feat, fix, chore, etc.)
3. **Groups Changes**: Organizes commits by type and version
4. **Generates Changelog**: Outputs formatted changelog following Keep a Changelog standard

Conventional Commit Support

gen-changelog recognizes standard conventional commit types:

- **feat**: New features

- **fix**: Bug fixes
- **docs**: Documentation changes
- **style**: Code style changes
- **refactor**: Code refactoring
- **test**: Test additions or changes
- **chore**: Maintenance tasks

Logging

Control output verbosity with logging options:

- `-v, --verbose`: Increase verbosity (can be used multiple times: `-vv`, `-vvv`)
- `-q, --quiet`: Decrease verbosity (can be used multiple times: `-qq`, `-qqq`)

Getting Help

For command-specific help, use:

```
gen-changelog <command> --help
```

For general help and available commands:

```
gen-changelog --help
```

Gen-changelog Library Documentation

The `gen-changelog` library provides comprehensive changelog generation from Git repositories using conventional commit messages. The library centres around the `ChangeLogConfig` and `ChangeLog` structs for configuring and constructing changelog documents.

Installation

Add the library to your program's `Cargo.toml` using `cargo add`:

```
$ cargo add gen-changelog
```

Or by configuring the dependencies manually in `Cargo.toml`:

```
[dependencies]
gen-changelog = "0.0.8"
```

Default Configuration

The library provides sensible defaults for conventional commit types:

Default Groups

Group	Commit Types	Published
Added	feat	✓
Fixed	fix	✓
Changed	refactor	✓
Security	security, dependency	✗
Build	build	✗
Documentation	doc, docs	✗
Chore	chore	✗
Continuous Integration	ci	✗
Testing	test	✗
Deprecated	deprecated	✗
Removed	removed	✗
Miscellaneous	misc	✗

By default, only `Added`, `Fixed`, `Changed`, and `Security` groups are published in the changelog.

Configuration File

The library looks for a `gen-changelog.toml` configuration file. Example structure:

```toml
## Controls the number of changelog sections to display.
display-sections = "all"

## Defines the display order of groups in the changelog.
[headings]
1 = "Added"
2 = "Fixed"
3 = "Changed"
4 = "Security"

## Group tables define the third-level headings used to organize commits.
[groups.Added]
name = "Added"
publish = true
cc-types = ["feat"]

[groups.Fixed]
name = "Fixed"
publish = true
```

```
cc-types = ["fix"]

## ... additional groups
```

Usage Examples

Basic Usage

```
use gen_changelog::{ChangeLog, ChangeLogConfig};
use git2::Repository;

fn generate_changelog() -> Result<(), Box<dyn std::error::Error>> {
    let repo = Repository::open(".")?;
    let config = ChangeLogConfig::from_file_or_default()?;

    let changelog = ChangeLog::builder()
        .with_config(config)
        .with_header("Changelog", &[
            "All notable changes to this project will be documented in this
file.",
            "The format is based on Keep a Changelog."
        ])
        .with_repository(&repo)?
        .build();

    changelog.save()?;
    Ok(())
}
```

Custom Configuration

```
use gen_changelog::{ChangeLog, ChangeLogConfig};
use git2::Repository;

fn generate_custom_changelog() -> Result<(), Box<dyn std::error::Error>> {
    let repo = Repository::open(".")?;
    let mut config = ChangeLogConfig::from_file_or_default()?;

    // Add custom groups to be published
    config.add_commit_groups(&["Documentation".to_string(),
"Testing".to_string()]);

    // Limit to last 5 releases
    config.set_display_sections(Some(5));

    let changelog = ChangeLog::builder()
        .with_config(config)
        .with_summary_flag(true)
```

```
        .with_repository(&repo)?
        .build();

    changelog.save()?;
    Ok(())
}
```

Release Preparation

```
use gen_changelog::{ChangeLog, ChangeLogConfig};
use git2::Repository;

fn prepare_release(version: &str) -> Result<(), Box<dyn std::error::Error>>
{
    let repo = Repository::open(".")?;
    let config = ChangeLogConfig::from_file_or_default()?;

    let changelog = ChangeLog::builder()
        .with_config(config)
        .with_repository(&repo)?
        .update_unreleased_to_next_version(Some(&version.to_string()))
        .build();

    changelog.save()?;
    Ok(())
}
```

Requirements

- Git repository with conventional commit messages
- GitHub repository for generating comparison links

The library automatically detects GitHub repositories and generates appropriate comparison and release links in the changelog output.