# Introduction

Thank you for considering contributing to Kedro! It's people like you that make Kedro such a great tool. We welcome contributions in the form of pull requests (PRs), issues or code reviews. You can add to code, [documentation](https://kedro.readthedocs.io), or simply send us spelling and grammar fixes or extra tests. Contribute anything that you think improves the community for us all!

The following sections describe our vision and contribution process.

## Vision

There is some work ahead, but Kedro aims to become the standard for developing production-ready data pipelines. To be production-ready, a data pipeline needs to be monitored, scheduled, scalable, versioned, testable and reproducible. Currently, Kedro helps you develop data pipelines that are testable, versioned, reproducible and we'll be extending our capability to cover the full set of characteristics for data pipelines over time.

## Code of conduct

The Kedro team pledges to foster and maintain a welcoming and friendly community in all of our spaces. All members of our community are expected to follow our [Code of Conduct](/CODE\_OF\_CONDUCT.md) and we will do our best to enforce those principles and build a happy environment where everyone is treated with respect and dignity.

# Get started

We use [GitHub Issues](https://github.com/quantumblacklabs/kedro/issues) to keep track of known bugs. We keep a close eye on them and try to make it clear when we have an internal fix in progress. Before reporting a new issue, please do your best to ensure your problem hasn't already been reported. If so, it's often better to just leave a comment on an existing issue, rather than create a new one. Old issues also can often include helpful tips and solutions to common problems.

If you are looking for help with your code, and the [FAQs](docs/source/11\_faq/01\_faq.md) in our documentation haven't helped you, please consider posting a question on [Stack Overflow](https://stackoverflow.com/questions/tagged/kedro). If you tag it `kedro` and `python`, more people will see it and may be able to help. We are unable to provide individual support via email. In the interest of community engagement we also believe that help is much more valuable if it's shared publicly, so that more people can benefit from it.

If you're over on Stack Overflow and want to boost your points, take a look at the `kedro` tag and see if you can help others out by sharing your knowledge. It's another great way to contribute.

If you have already checked the existing issues in [GitHub issues](https://github.com/quantumblacklabs/kedro/issues) and are still convinced that you have found odd or erroneous behaviour then please file an [issue](https://github.com/quantumblacklabs/kedro). We have a template that helps you provide the necessary information we'll need in order to address your query.

## Feature requests

### Suggest a new feature

If you have new ideas for Kedro functionality then please open a [GitHub issue](https://github.com/quantumblacklabs/kedro/issues) with the label `Type: Enhancement`. You can submit an issue [here](https://github.com/quantumblacklabs/kedro/issues) which describes the feature you would like to see, why you need it, and how it should work.

### Contribute a new feature

If you're unsure where to begin contributing to Kedro, please start by looking through the `good first issues` and `help wanted issues` on [GitHub](https://github.com/quantumblacklabs/kedro/issues).

We focus on three areas for contribution: `core`, [`extras`](/kedro/extras/) or `plugin`:

- `core` refers to the primary Kedro library

- [`extras`](/kedro/extras/) refers to features that could be added to `core` that do not introduce too many depencies or require new Kedro CLI commands to be created e.g. adding a new dataset to the `kedro.extras.dataset` data management module. All the datasets are placed under `kedro.extras.datasets` to separate heavy depencies (e.g Pandas) from Kedro `core` components.

- [`plugin`](https://kedro.readthedocs.io/en/stable/04\_user\_guide/10\_developing\_plugins.html) refers to new functionality that requires a Kedro CLI command e.g. adding in Airflow functionality

Typically, we only accept small contributions for the `core` Kedro library but accept new features as `plugin`s or additions to the [`extras`](/kedro/extras/) module. We regularly review [`extras`](/kedro/extras/) and may migrate modules to `core` if they prove to be essential for the functioning of the framework.

## Your first contribution

Working on your first pull request? You can learn how from these resources:

\* [First timers only](https://www.firsttimersonly.com/)

\* [How to contribute to an open source project on GitHub](https://egghead.io/courses/how-to-contribute-to-an-open-source-project-on-github)

### Developer Workflow

#### First time (one-off)

Ensure you have done these:

- [ ] Read through the pre-requisites in the [ReadTheDocs documentation](https://kedro.readthedocs.io/en/stable/02\_get\_started/01\_prerequisites.html).

- [ ] Run the below commands:

```

make install-test-requirements

make install-pre-commit

```

- [ ] Once the above commands have executed successfully, do a sanity check to ensure that `kedro` works in your environment:

```

make test

make build-docs

```

#### Then onwards (code or documentation changes)

If you picked up a code or documentation related issue, before pushing the branch to the repo and creating a Pull Request please do the below:

- [ ] for code changes:

```

make test

```

- [ ] for documentation related changes:

```

make build-docs

```

### Guidelines

\* Aim for cross-platform compatibility on Windows, macOS and Linux

\* Aim for [backwards compatible](#Backwards-compatibility) changes where possible, and break compatibility only in exceptional cases

\* We use [Anaconda](https://www.anaconda.com/distribution/) as a preferred virtual environment

\* We use [SemVer](https://semver.org/) for versioning

Our code is designed to be compatible with Python 3.6 onwards and our style guidelines are (in cascading order):

\* [PEP 8 conventions](https://www.python.org/dev/peps/pep-0008/) for all Python code

\* [Google docstrings](https://google.github.io/styleguide/pyguide.html#38-comments-and-docstrings) for code comments

\* [PEP 484 type hints](https://www.python.org/dev/peps/pep-0484/) for all user-facing functions / class methods e.g.

```

def count\_truthy(elements: List[Any]) -> int:

return sum(1 for elem in elements if element)

```

> \*Note:\* We only accept contributions under the Apache 2.0 license and you should have permission to share the submitted code.

Please note that each code file should have a licence header, include the content of [`legal\_header.txt`](https://github.com/quantumblacklabs/kedro/blob/master/legal\_header.txt).

There is an automated check to verify that it exists. The check will highlight any issues and suggest a solution.

### Branching conventions

We use a branching model that helps us keep track of branches in a logical, consistent way. All branches should have the hyphen-separated convention of: `<type-of-change>/<short-description-of-change>` e.g. `feature/io-dataset`

| Types of changes | Description |

| ---------------- | ----------------------------------------------------------------------- |

| `docs` | Changes to the documentation under `docs/source/` |

| `feature` | Change which adds or removes functionality |

| `fix` | Non-breaking change which fixes an issue |

| `tests` | Changes to project unit `tests/` and / or integration `features/` tests |

### Backwards compatibility

#### What is a breaking change?

A breaking change is any change that modifies Kedro's public APIs. Examples include making a change to the signature of public functions or removing a module. Your change is \*\*not\*\* considered a breaking change if a user can upgrade their Kedro version and include your change without anything breaking in their project.

A backwards-compatible change is any change that is not a breaking change.

#### When should I make a breaking change?

We aim to minimise the number of breaking changes to help keep the Kedro software stable and reduce the overhead for users as they migrate their projects. However, there are cases where a breaking change brings considerable value or increases the maintainability of the codebase. In these cases, breaking backwards compatibility can make sense.

Before contributing a breaking change, you should create an [issue](https://github.com/quantumblacklabs/kedro/issues) describing the change and justify the value gained by breaking backwards compatibility.

# Our release model

All non-breaking changes go into `master`, from which a minor release can be deployed at any time. Any non-breaking change should branch off from `master` and be merged into `master`, as explained in the [contribution process](/CONTRIBUTING.md#core-contribution-process) below.

All breaking changes go into `develop`, from which a major release can be deployed at any time. Any breaking change should branch off from `develop` and be merged into `develop`, as explained in the [contribution process](/CONTRIBUTING.md#core-contribution-process) below. The `develop` branch contains all commits from the `master` branch, but the `master` branch does not contain all the commits from `develop` until the next major release.

![Kedro Gitflow Diagram](https://raw.githubusercontent.com/quantumblacklabs/kedro/master/static/img/kedro\_gitflow.svg)

## `core` contribution process

Small contributions are accepted for the `core` library:

1. Fork the project by clicking \*\*Fork\*\* in the top-right corner of the [Kedro GitHub repository](https://github.com/quantumblacklabs/kedro)and then choosing the target account the repository will be forked to.

2. Create a feature branch on your forked repository and push all your local changes to that feature branch. Your feature branch should branch off from:

a. `master` if you intend for it to be a non-breaking, backwards-compatible change.

b. `develop` if you intend for it to be a breaking change

3. Before submitting a pull request (PR), please ensure that unit, end-to-end tests and linting are passing for your changes by running `make test`, `make e2e-tests` and `make lint` locally, have a look at the section [Running checks locally](/CONTRIBUTING.md#running-checks-locally)below.

4. Determine if your change is [backwards compatible](#Backwards\_compatibility):

a. For backwards compatible changes, open a PR against the `quantumblacklabs:master` branch from your feature branch.

b. For changes that are NOT backwards compatible, open a PR against the `quantumblacklabs:develop` branch from your feature branch.

5. Await reviewer comments.

6. Update the PR according to the reviewer's comments.

7. Your PR will be merged by the Kedro team once all the comments are addressed.

> \_Note:\_ We will work with you to complete your contribution but we reserve the right to take over abandoned PRs.

## `extras` contribution process

You can add new work to `extras` if you do not need to create a new Kedro CLI command:

1. Create an [issue](https://github.com/quantumblacklabs/kedro/issues) describing your contribution.

2. Fork the project by clicking \*\*Fork\*\* in the top-right corner of the [Kedro GitHub repository](https://github.com/quantumblacklabs/kedro)and then choosing the target account the repository will be forked to.

3. Work in [`extras`](/kedro/extras/) and create a feature branch on your forked repository and push all your local changes to that featurebranch.

4. Before submitting a pull request, please ensure that unit, e2e tests and linting are passing for your changes by running `make test`,`make e2e-tests` and `make lint` locally, have a look at the section [Running checks locally](/CONTRIBUTING.md#running-checks-locally) below.

5. Include a `README.md` with instructions on how to use your contribution.

6. Determine if your change is [backwards compatible](#Backwards\_compatibility):

a. For backwards compatible changes, open a PR against the `quantumblacklabs:master` branch from your feature branch.

b. For changes that are NOT backwards compatible, open a PR against the `quantumblacklabs:develop` branch from your feature branch.

7. Reference your issue in the PR description (e.g., `Resolves #<issue-number>`).

8. Await review comments.

9. Update the PR according to the reviewer's comments.

10. Your PR will be merged by the Kedro team once all the comments are addressed.

> \_Note:\_ We will work with you to complete your contribution but we reserve the right to take over abandoned PRs.

## `plugin` contribution process

See the [`plugin` development documentation](https://kedro.readthedocs.io/en/stable/04\_user\_guide/10\_developing\_plugins.html) for guidance on how to design and develop a Kedro `plugin`.

## CI / CD and running checks locally

To run E2E tests you need to install the test requirements which includes `behave`.

Also we use [pre-commit](https://pre-commit.com) hooks for the repository to run the checks automatically.

It can all be installed using the following command:

```bash

make install-test-requirements

make install-pre-commit

```

> \_Note:\_ If Spark/PySpark/Hive tests for datasets are failing it might be due to the lack of Java>8 support from Spark. You can try using `export JAVA\_HOME=$(/usr/libexec/java\_home -v 1.8)` which works under MacOS or other workarounds. [Reference](https://stackoverflow.com/questions/53583199/pyspark-error-unsupported-class-file-major-version-55)

### Running checks locally

All checks run by our CI / CD servers can be run locally on your computer.

#### PEP-8 Standards (`pylint` and `flake8`)

```bash

make lint

```

#### Unit tests, 100% coverage (`pytest`, `pytest-cov`)

Note that you will need the dependencies installed in `test\_requirements.txt`, which includes `memory-profiler`. If you are on a Unix-like system, you may need to install the necessary build tools. See the `README.md` [file](/kedro/contrib/decorators/README.md) in `kedro.contrib.decorators` for more information.

```bash

make test

```

> Note: We place [conftest.py](https://docs.pytest.org/en/latest/fixture.html#conftest-py-sharing-fixture-functions) files in some test directories to make fixtures reusable by any tests in that directory. If you need to see which test fixtures are available and where they come from, you can issue:

```bash

pytest --fixtures path/to/the/test/location.py

```

#### End-to-end tests (`behave`)

```bash

behave

```

#### Others

Our CI / CD also checks that `kedro` installs cleanly on a fresh Python virtual environment, a task which depends on successfully building the docs:

```bash

make build-docs

```

This command will only work on Unix-like systems and requires `pandoc` to be installed.

## Hints on pre-commit usage

The checks will automatically run on all the changed files on each commit.

Even more extensive set of checks (including the heavy set of `pylint` checks)

will run before the push.

The pre-commit/pre-push checks can be omitted by running with `--no-verify` flag, as per below:

```bash

git commit --no-verify <...>

git push --no-verify <...>

```

(`-n` alias works for `git commit`, but not for `git push`)

All checks will run during CI build, so skipping checks on push will

not allow you to merge your code with failing checks.

You can uninstall the pre-commit hooks by running:

```bash

make uninstall-pre-commit

```

`pre-commit` will still be used by `make lint`, but will not install the git hooks.