

NetworKit Development Guide

NetworKit is an open-source toolkit for high-performance network analysis. This text is meant to provide some guidelines for the ongoing development of the project. It is meant for core developers, occasional contributors, and students working on the code.

The following text assumes some basic familiarity with the Mercurial version control software. It is not a Mercurial tutorial, because you will find a good one at hginit.com. Rather, it explains concepts and workflows for the development of this project.

Repositories

The NetworKit main development repository is at <http://alghub.iti.kit.edu/parco/NetworKit/NetworKit>. Access to this repository is provided on request.

alghub.iti.kit.edu (an installation of [RhodeCode](#)) makes it easy to create and manage forks. Forking is distinct from branching and creates a new repository with a new address, its own access control etc. A fork contains all branches of its parent.

Branches

Currently, the two most important branches of NetworKit are **Dev** and **default**.

```
----- Dev
----/----- default
```

As the name says, **default** is the branch which you are on if you do not switch. It is therefore the release branch, containing code which is ready for use. Unless you are a core developer preparing a release or fixing an urgent bug, you do not make changes to **default**.

Dev is the development branch and most of the development of new features happens in this branch. This is also where new releases are being prepared. When pushing into this branch, think about whether your code is ready for the core development team to work with and will be suitable for a release in the foreseeable future.

It can be appropriate to create additional branches for projects, features, developer teams etc. Creation of branches should be coordinated with the core development team. For this purpose, post to the [developers e-mail list](#).

Tags

A tag is nothing more than a “symbolic name” for a revision. In NetworKit tags are used to mark release versions in the **default** branch, with a MAJOR.MINOR version name scheme.

Workflows

This section describes how to work with branches and forks in different scenarios.

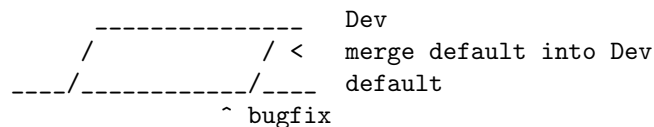
Using NetworKit

If you want to build and use NetworKit and do not plan to contribute changes, simply clone the repository. By default, you will be on the **default** branch, which represents the current release. Follow the setup instructions in the **Readme**.

Core Development

This section describes workflows for the core development team.

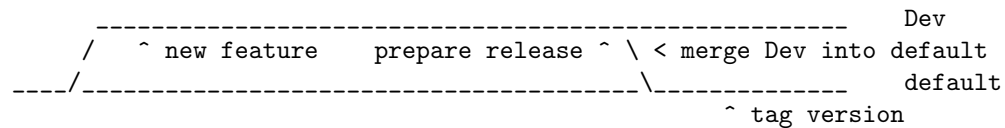
Bugfixes Bugfixes are changes that should be immediately visible to users of NetworKit, such as solutions for urgent errors or improvements of the **Readme** document. In this case, make the changes in the **default** branch and commit. Then switch to the **Dev** branch and merge the **default** branch back into **Dev**.



Example:

```
hg up default
...
hg com -m "fixed bug xyz"
hg up Dev
hg merge default
hg com -m "backmerge bugfix xyz"
```

Releasing New Features When new features should be released, the `Dev` branch is merged into the `default` branch. Additional testing and cleanup is performed before that happens. The new major or minor release is then tagged with a version number.



Example:

```

hg up Dev
hg com -m "ready for release X.Y"
hg up default
hg merge Dev
hg com -m "release X.Y"

```

Multiple heads in multiple branches If remote changes have happened in multiple branches and you pull them, these branch will have multiple heads. Merging now needs to happen for each of the affected branches before you can push. Switch to each branch and perform a merge as usual.

Contributions

Users of NetworkKit are welcome to contribute their modifications. New features must be added to the `Dev` branch, not the `default` branch. We recommend the following workflow:

1. create a fork of the main repository
2. switch to the `Dev` branch
3. make and commit your changes while being on the `Dev` branch
4. send a pull request to the main repository

Student Exercises

NetworkKit is currently also used as a teaching tool. This section describes the workflow for student teams. Suppose the course is named, “Networks 101”, then there will be a dedicated branch `Networks101` for student exercises.

1. Fork the main repository via algothub.iti.kit.edu and name the fork according to your team. (On the repository page, click 'Options -> Fork')
2. Make sure that the correct access rights for your team are set. (On the repository page: **Options** -> **Settings**)
3. Switch to the appropriate branch for the course (e.g. **hg up Networks101**) and **ONLY** work on this branch.
4. Work with the forked repository as you please. Coordinate with your team.
5. On completion of the exercise, send a pull request from your fork to the main repository. (On the repository page, click **Options** -> **Create Pull Request**)
6. The pull request is now under review. Watch for and react to comments from the reviewer.

We also ask student teams to adhere to the following conventions:

- With multiple teams working on the same exercise, append your team name to the class and file names as well as the names of unit tests to avoid naming clashes.
- If you plan to make modifications to existing parts of NetworKit, discuss them with the core developers first, e.g. by posting to the [developers e-mail list](#).
- Delete forked repositories when they are no longer needed.

Reviewing Student Exercises

Incoming pull requests appear as notifications on algothub.iti.kit.edu. It is also possible to receive notifications via e-mail.

1. Before the course starts, create an appropriate branch for the course (e.g. **Networks101**). Derive the branch from the **Dev** branch.
2. Receive pull requests from student teams via algothub.iti.kit.edu.
3. To review a pull request, switch to the course branch and pull from the forked repository of the student team. Make sure to pull the revision associated with the pull request (e.g. **'hg pull -r '**)
4. If everything is okay, change the status of the pull request to **Accepted** (click **Change Status** above the comment field). The comment field can be used to send feedback, creators of the request will be notified via email.

Good contributions from the student exercises should be merged back into the **Dev** branch.

Student Projects

Students with long-term projects like Bachelor's or Master's theses should familiarize themselves with the guidelines and select a forking/branching model with their advisor.

Branching Cheat Sheet

- list all available branches: `hg branches`
- check on which branch you are: `hg branch`
- see heads (most recent commits) of all branches: `hg head`
- see tip (most recent commits) of the branch you are currently working on: `hg tip`
- switch to a specific branch: `hg update <branchname>`
- start a new branch: `hg branch <branchname>`
- merge `branchY` into `branchX`: `hg update branchX`, then `hg merge branchY`

Conventions

The following general conventions apply to all NetworKit developers.

Versioning

- Before you commit, make sure your code compiles and run the unit tests. Never push code which breaks the build for others.
- Commit regularly and often to your local repository.
- Use meaningful commit messages.
- Get the newest changes from the repository regularly and merge them into your local repository.
- Make sure that you merged correctly and did not break other people's work.
- Push correct code early if possible. Merging is easier if all developers are up to date.
- Never `push --force` to the main repository.

Unit Tests

Every new feature must be covered by a unit test. Omitting unit tests makes it very likely that your feature will break silently as the project develops, leading to unnecessary work in tracing back the source of the error.

Unit tests for NetworKit are based on the `googletest` library. For more information read the [googletest primer](#).

- Each source folder contains a `test` folder with `googletest` classes. Create the unit tests for each feature in the appropriate `test/*GTest` class by adding a `TEST_F` function.
- Prefix standard unit tests with `test` and experimental feature tests with `try`. A `test*` must pass when pushed to the main repository, a `try*` is allowed to fail.
- Keep the running time of test functions to the minimum needed for testing functionality. Testing should be fast, long-running unit tests look like infinite loops.
- If the unit test requires a data set, add the file to the `input/` folder. Only small data sets (a few kilobytes maximum) are acceptable in the repository.
- Any output files produced by unit tests must be written to the `output/` folder.

Code Style

- Compiler warnings are likely to turn into future errors. Try to fix them as soon as they appear.
- Read some code to get used to the code style and try to adopt it.
- Document classes, methods and attributes in Doxygen style.
- Use the `count` and `index` integer types for non-negative integer quantities and indices.
- In most cases, objects are passed by reference. New objects are stack-allocated and returned by value. Avoid pointers and `new` where possible.
- Use the `override` keyword to indicate that a method overrides a virtual method in the superclass.
- In Python, indent using tabs, not spaces.

Exposing C++ Code to Python

TODO - documentation coming soon

Contact

To discuss important changes to NetworKit, use the [developers e-mail list](#).

To suggest improvements to workflows and conventions, write to `christian.staudt@kit.edu`.

Further Reading

- [hginit.com](#)
- [Working with named branches](#)
- [Managing releases and branchy development](#)