1. The central repo holds two main branches with an infinite lifetime:
    I. master: It should be the main branch where the source code of `HEAD` always reflects a *production-ready* state.
    II. develop: It should be the main branch where the source code of `HEAD` always reflects a state with the latest delivered development changes for the next release.
2. Each time when changes are merged back into `master`, this is a new production release *by definition*. We should be very strict at this, so that, we could use a Git hook script to automatically build and roll-out our software to our production servers everytime there was a commit on `master`.
3. Supporting branches: feature branch, release branch and hotfix branch,
    I. feature branch:
        A. May branch off from **`develop`**
        B. Must merge back into **`develop`**
        C. It exists as long as the feature is in development, but will eventually be merged back into `develop` (to definitely add the new feature to the upcoming release) or discarded (in case of a disappointing experiment).
        D. When starting work on a new feature, branch off from the **`develop`** branch as:
            - `$ git checkout -b myfeature develop`
        E. Finished features may be merged into the `develop` branch to definitely add them to the upcoming release:
            - `$ git checkout develop`
            - `$ git merge --no-ff myfeature`
            - `$ git branch -d myfeature`
            - `$ git push origin develop`
        F. The `--no-ff` flag causes the merge to always create a new commit object, even if the merge could be performed with a fast-forward. This avoids losing information about the historical existence of a feature branch and groups together all commits that together added the feature.

            In the latter case, it is impossible to see from the Git history which of the commit objects together have implemented a feature—you would have to manually read all the log messages. Reverting a whole feature (i.e. a group of commits), is a true headache in the latter situation, whereas it is easily done if the `--no-ff` flag was used.
    II. release branch:
        A. May branch off from **`develop`**
        B. Must merge back into **`develop`** and **`master`**
        C. Branch naming convention: **`release-*`**
        D. Release branches support preparation of a new production release. Furthermore, they allow for minor bug fixes and preparing meta-data for a release (version number, build dates, etc.).

E. By doing all of this work on a release branch, the `develop` branch is cleared to receive features for the next big release.
F. The key moment to branch off a new release branch from `develop` is when develop (almost) reflects the desired state of the new release.
G. All features targeted at future releases should wait until after the release branch is branched off.
H. It is exactly at the start of a release branch that the upcoming release gets assigned a version number—not any earlier. Up until that moment, the `develop` branch reflected changes for the "next release", but it is unclear whether that "next release" will eventually become 0.3 or 1.0, until the release branch is started. That decision is made on the start of the release branch and is carried out by the project's rules on version number bumping.
I. Release branch is branched off from the **`develop`** branch as:
    - `$ git checkout -b release-1.2 develop`
    - `./bump-version.sh 1.2`
    - `git commit -a -m "Bumped version number to 1.2"`
J. This new branch may exist there for a while, until the release may be rolled out definitely. During that time, bug fixes may be applied in this branch (rather than on the `develop` branch).
K. Adding large new features in the release branch is strictly prohibited. They must be merged into `develop`, and therefore, wait for the next big release.
L. When the state of the release branch is ready to become a real release, some actions need to be carried out:
    - `$ git checkout master`
    - `$ git merge --no-ff release-1.2`
    - `$ git tag -a 1.2`
M. A commit on `master` must be tagged for easy future reference to this historical version.
N. To keep the changes made in the release branch, we need to merge those back into `develop`, though.
    - `$ git checkout develop`
    - `$ git merge --no-ff release-1.2`
O. This step may well lead to a merge conflict (probably even, since we have changed the version number). If so, fix it and commit.
P. Now we are really done and the release branch may be removed, since we don't need it anymore:
    - `$ git branch -d release-1.2`

III. hotfix branch:
    A. May branch off from **`master`**

B. Must merge back into **`develop`** and **`master`**
C. Branch naming convention: **`hotfix-*`**
D. Hotfix branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned.
E. The essence is that work of team members (on the `develop` branch) can continue, while another person is preparing a quick production fix.
F. This is how we can work with hotfix branch:
    - `$ git checkout -b hotfix-1.2.1 master`
    - `$ ./bump-version.sh 1.2.1`
    - `$ git commit -a -m "Bumped version number to 1.2.1"`
    - `$ git commit -m "Fixed severe production problem"`
G. Then, fix the bug and commit the fix in one or more separate commits.
    - `$ git commit -m "Fixed severe production problem"`
H. Then update `master` and tag the release:
    - `$ git checkout master`
    - `$ git merge --no-ff hotfix-1.2.1`
    - `$ git tag -a 1.2.1`
I. Next, include the bugfix in `develop`, too:
    - `$ git checkout develop`
    - `$ git merge --no-ff hotfix-1.2.1`
J. The one exception to the rule here is that, **when a release branch currently exists, the hotfix changes need to be merged into that release branch, instead of** `develop`. Back-merging the bugfix into the release branch will eventually result in the bugfix being merged into `develop` too, when the release branch is finished. (If work in `develop` immediately requires this bugfix and cannot wait for the release branch to be finished, you may safely merge the bugfix into `develop` now already as well.)
K. Finally, remove the temporary branch:
    - `$ git branch -d hotfix-1.2.1`
L. Abc

4. Abc

Pull request in github:
https://yangsu.github.io/pull-request-tutorial/