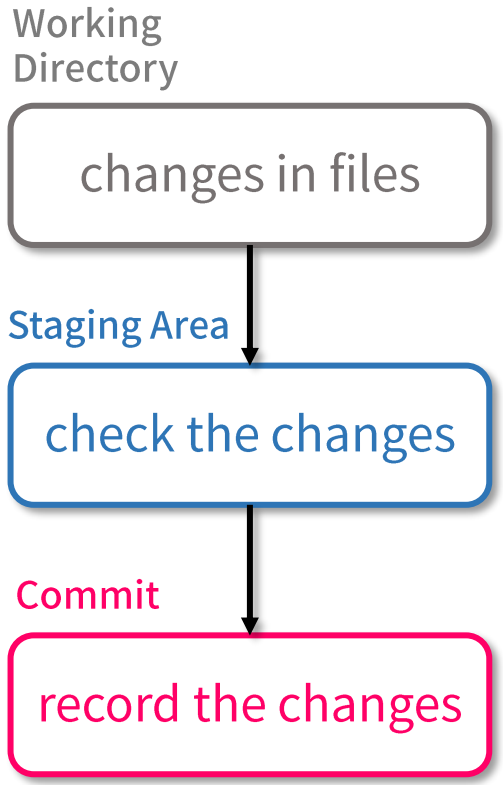
# Introduction to Git and Development Cycle

2022-12-12

Ping-Han Hsieh, Tatiana Belova, Katalin Ferenc
Centre for Molecular Medicine Norway
RSG Norway

# What is Git (1)

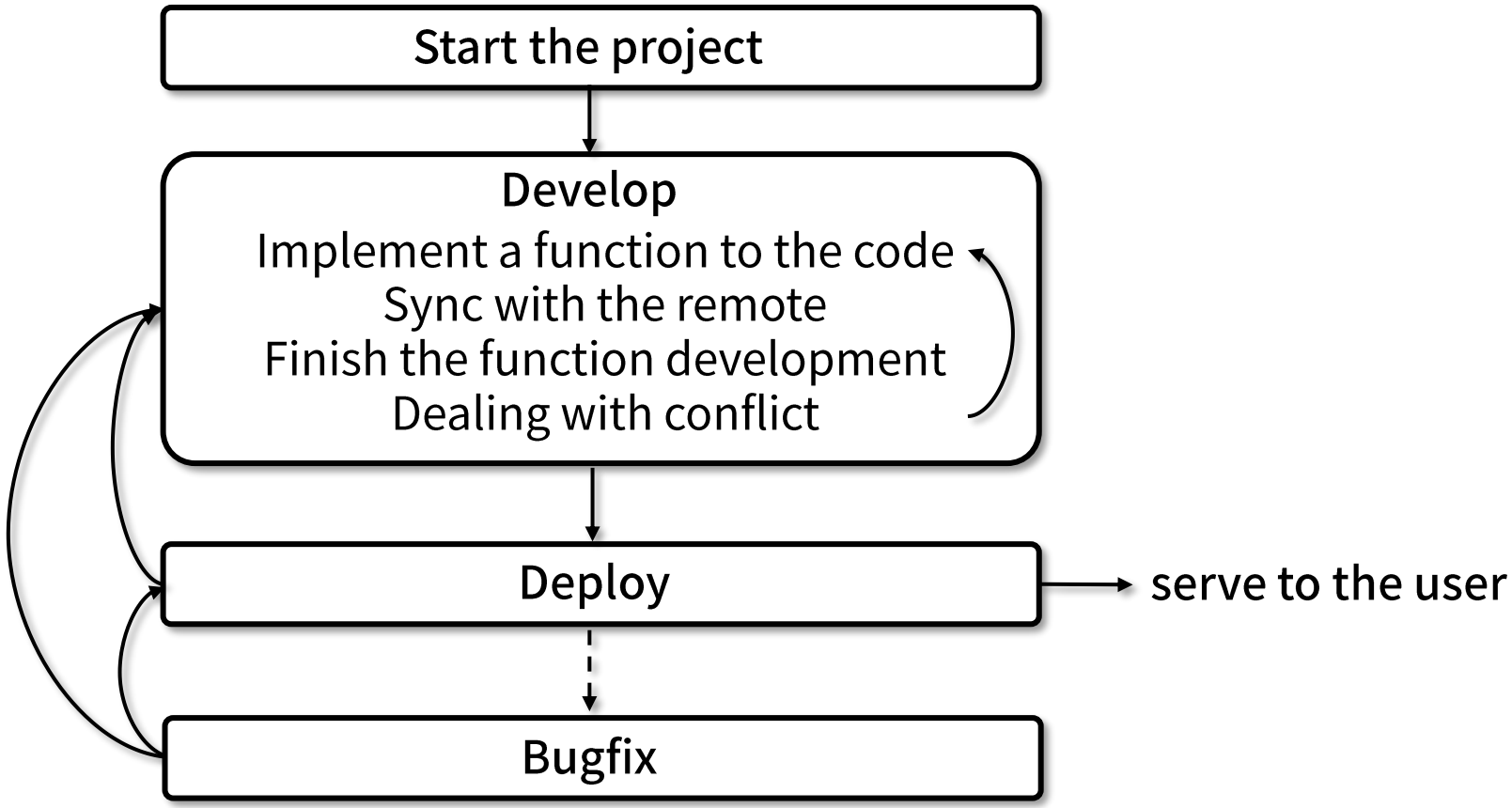
- Git
  - An open-source distributed version control system.
  - Lightweight and efficient.
  - Disposable experimentation.
  - Suitable for collaboration.
- Gitflow
  - Strategy for managing Git branches.
- Important Terminologies
  - working directory
  - repository
  - staging area
  - commit
  - branch
  - remote



# What is Git (2)

| Git | Google Drive / OneDrive | Description |
|---|---|---|
| repository | Google Drive / OneDrive directory | root directory where the files inside could be tracked |
| working directory | directory on the computer | where we modify files |
| staging area | not supported, changes are recorded automatically | where we temporarily put the changes of files |
| commit | version index | a permanent tag to the changes of files |
| branch | not supported | where we put a series of changes (commit) |
| remote | the Google/OneDrive Cloud | cloud storage of the changes (e.g., GitHub, GitLab, Bitbucket) |

# Development Cycle



# Start the Project

- Now we want to start a project
  - We will need to set up the environment
- If we are using Google Drive/One Drive,
  - We create a directory in the Cloud (or in the local environment).
  - Then we share the directory with our collaborator
- If we are using Google Doc
  - We create a document/form/presentation in the Cloud.
  - Then we share the files with our collaborator
- How is it done with Git?
  - Create a project (repository)
  - Share with other collaborators

# Create a Project (1)

- Make current directory a Git local repository.

```
git init
```

everything inside this directory belongs to this project

- Allow the Git local repository to use Gitflow (optional).

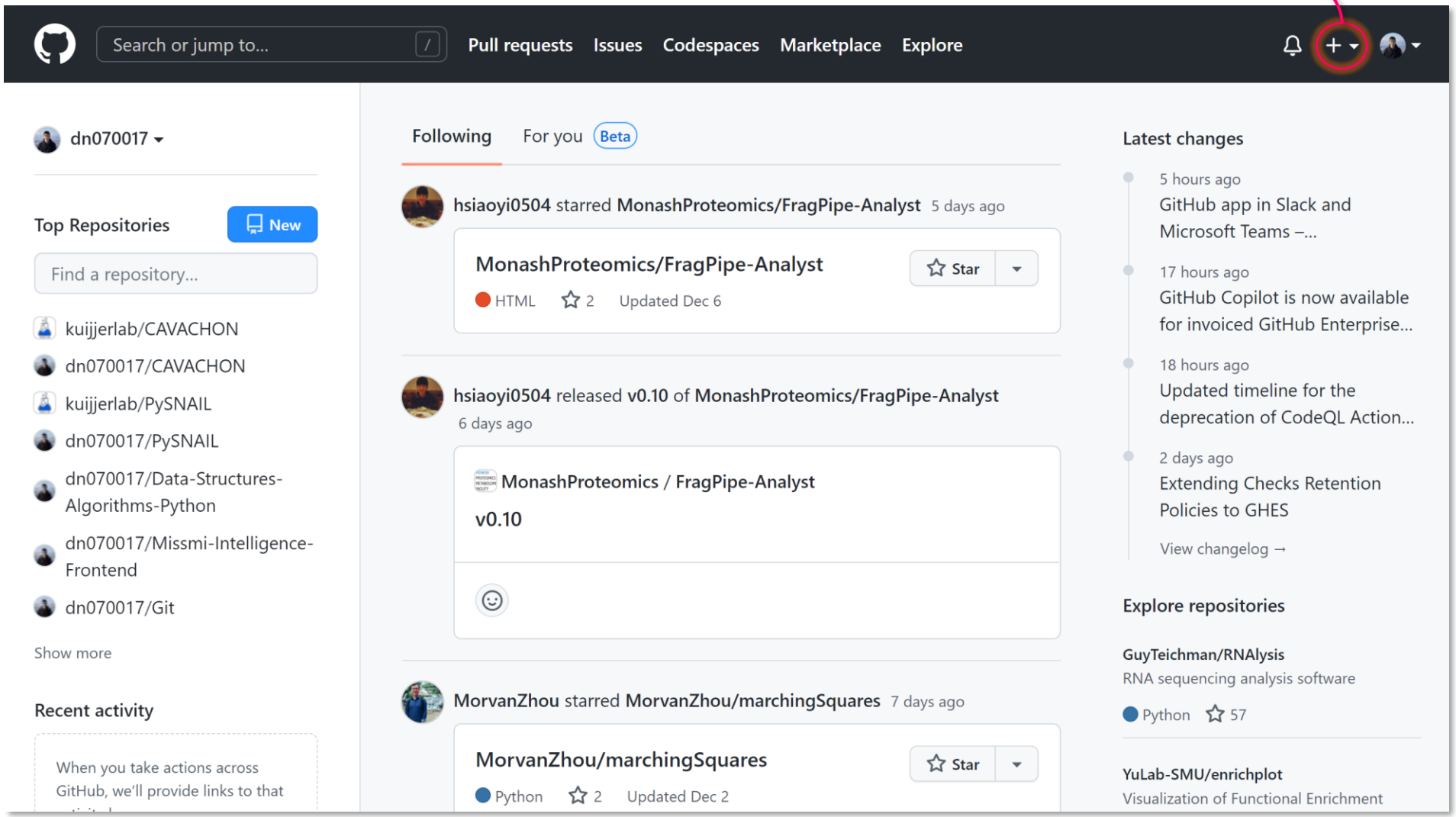
```
git flow init
```

- Configure the Git local repository.

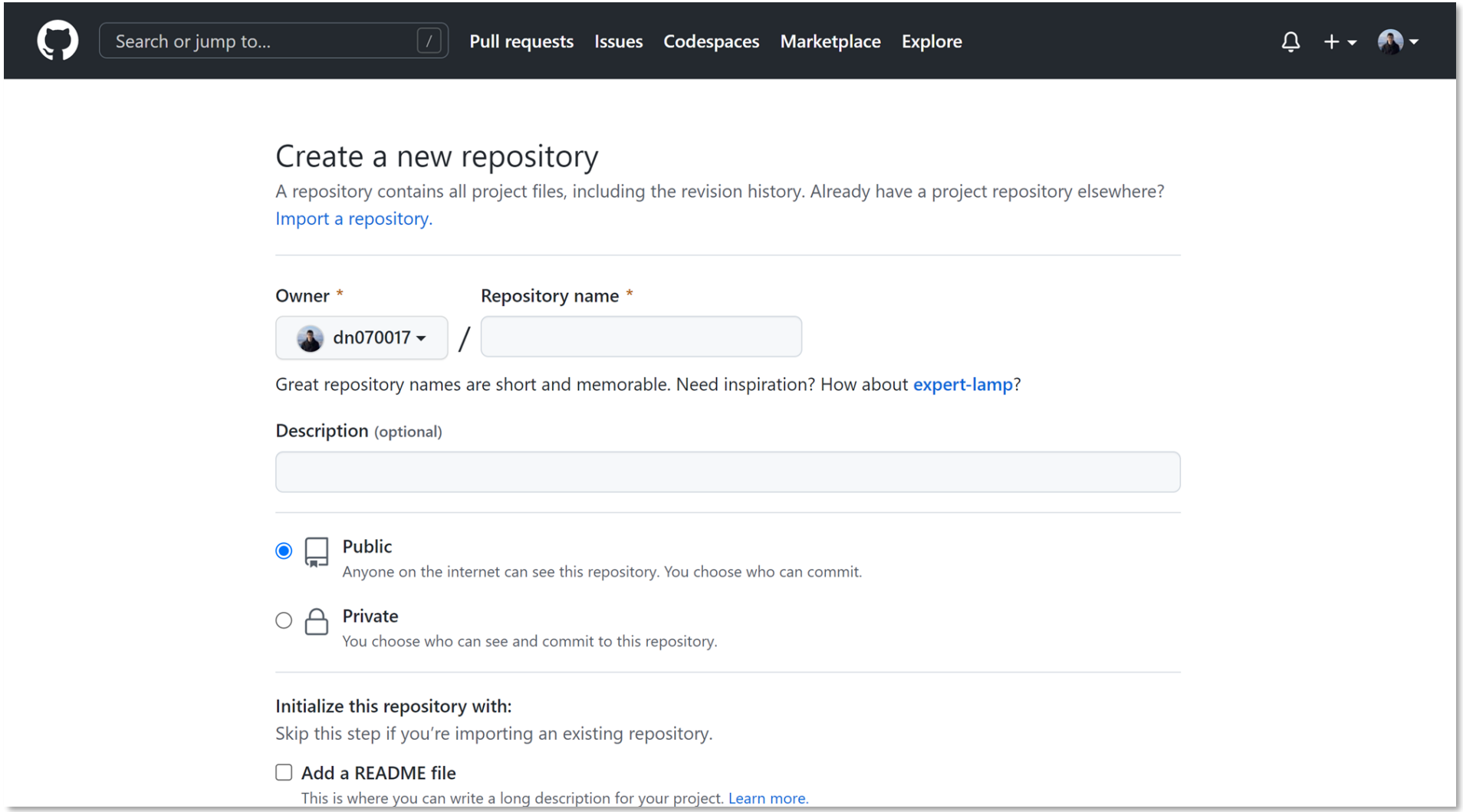
```
git config --local user.name  (name)
git config --local user.email (email)
```

setup your identity as the developer of this project

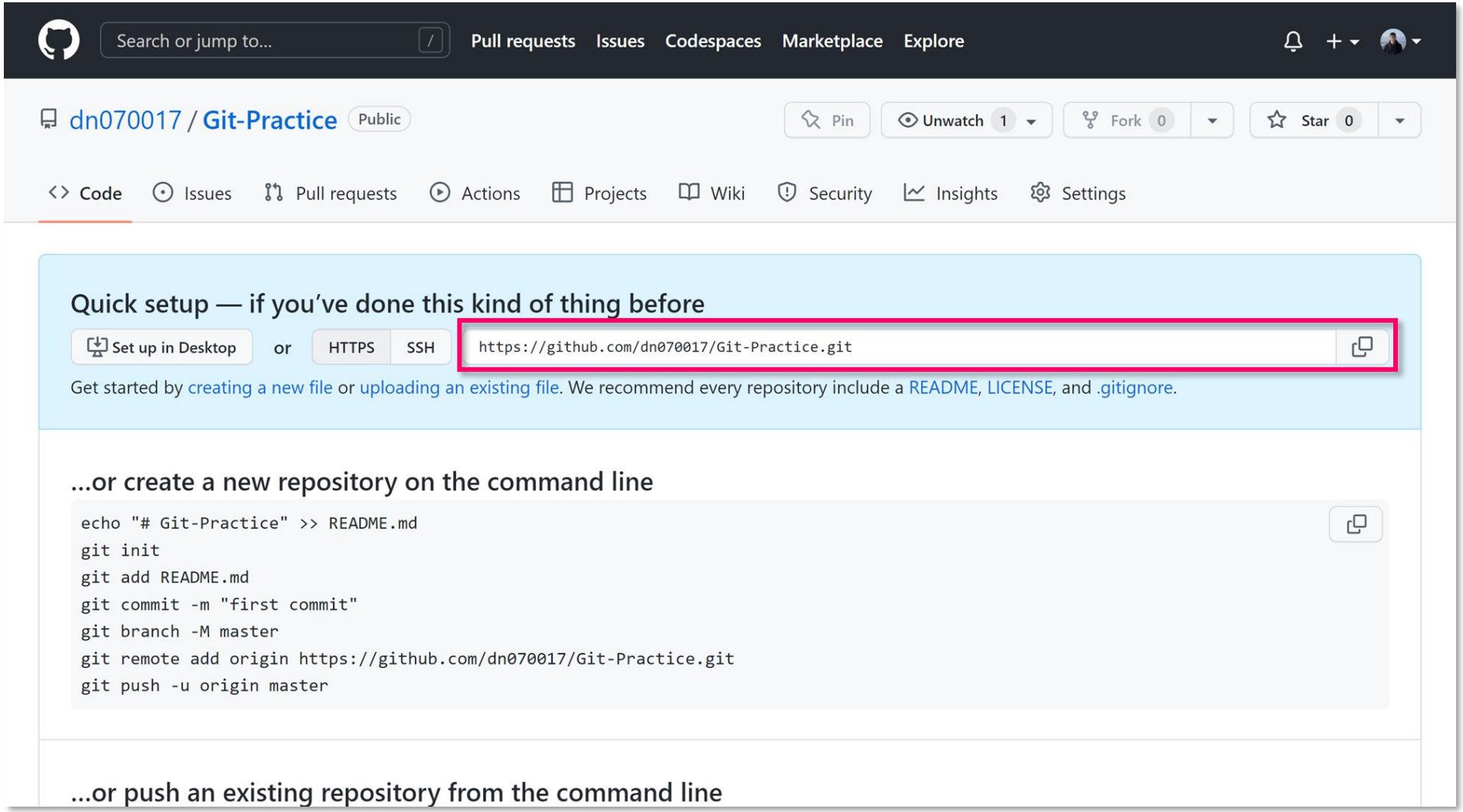
# Create a Project (2)



# Create a Project (3)



# Create a Project (4)




Project URL

# Create a Project (5)

- Set the URL to the remote repository for the local one.

```
git remote add (name) (remote.url)
```

```
git remote add origin git@github.com:dn070017/Git-Workshop.git
```

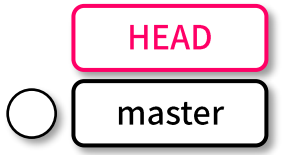
change it to your group URL

- One can make multiple aliases to the URL.
- One local repository can sync with multiple remote repositories.

# Create a Project (6)

- Make an empty commit.

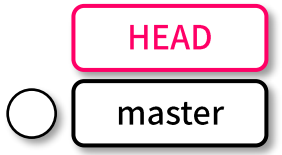
```
git commit --allow-empty -m "Initial Commit"
```



# Create a Project (7)

- Create a develop branch

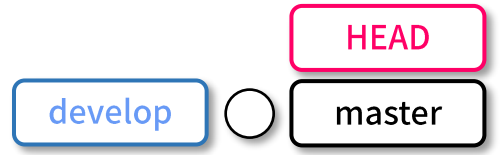
```
git branch develop
```



# Create a Project (8)

- Create a develop branch

```
git branch develop
```



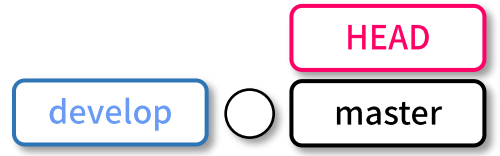
# Create a Project (9)

- Create a develop branch

```
git branch develop
```

- Change to develop branch

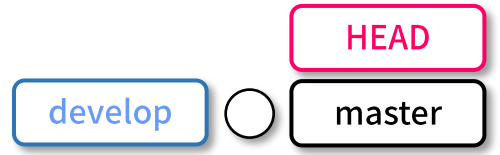
```
git checkout develop
```



# Create a Project (10)

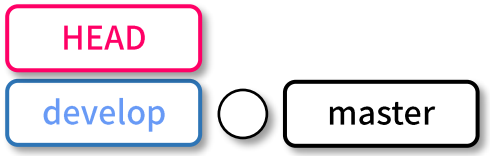
- Create a develop branch

```
git branch develop
```




- Change to develop branch

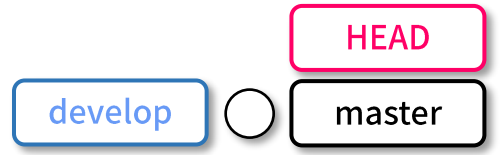
```
git checkout develop
```



# Create a Project (11)

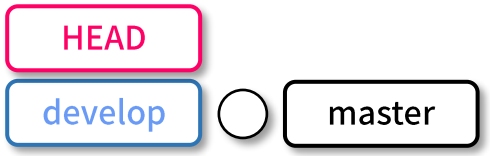
- Create a develop branch

```
git branch develop
```




- Change to develop branch

```
git checkout develop
```




- Check branch info

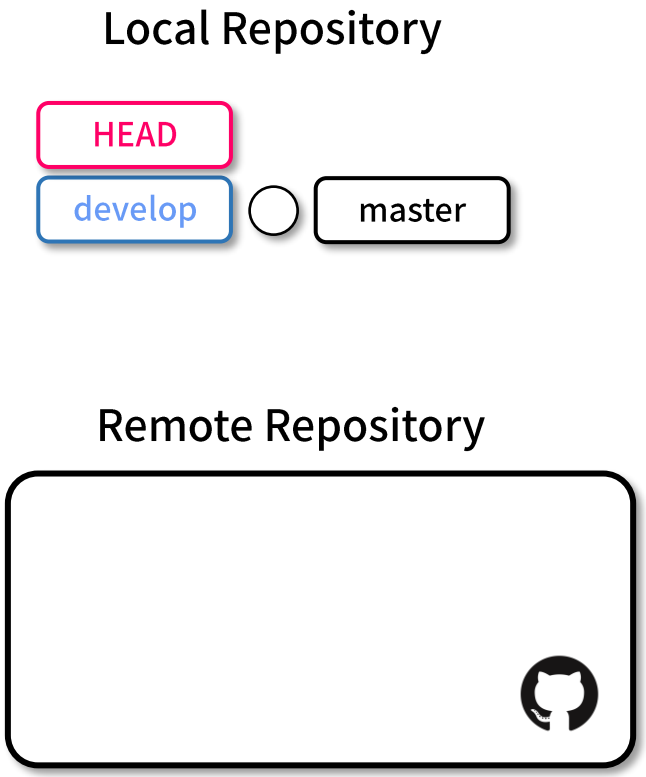
```
git branch
```

# Sync with the Remote (1)

- Update the remote repository to sync with the local repository.

```
git push (remote.name) (local.branch):(remote.branch)
```

```
git push origin master:master
git push origin develop:develop
```

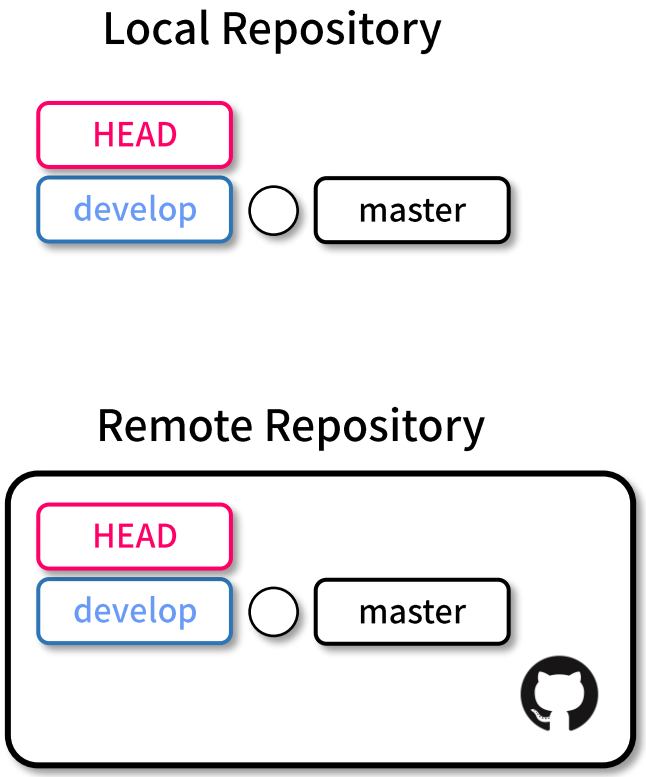


# Sync with the Remote (2)

- Update the remote repository to sync with the local repository.

```
git push (remote.name) (local.branch):(remote.branch)
```

```
git push origin master:master
git push origin develop:develop
```



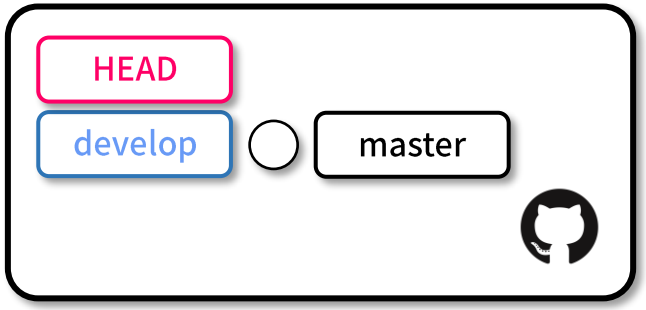
# View Project as Collaborator (1)

- Clone the repository.

```
git clone (URL)
```

Local Repository (Collaborator)

Remote Repository



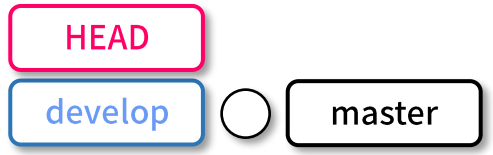
# View Project as Collaborator (2)

- Clone the repository.

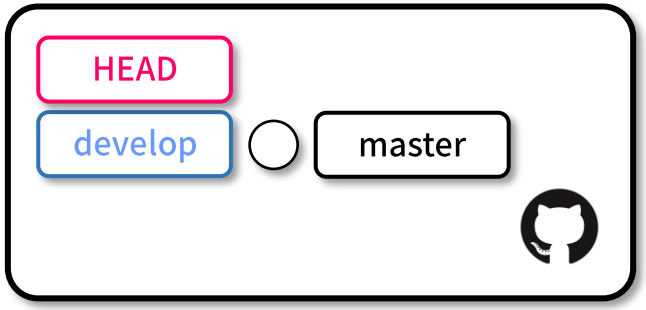
```
git clone (URL)
```

**By default, everyone can view any public repository, but only the collaborator can edit the repository.**

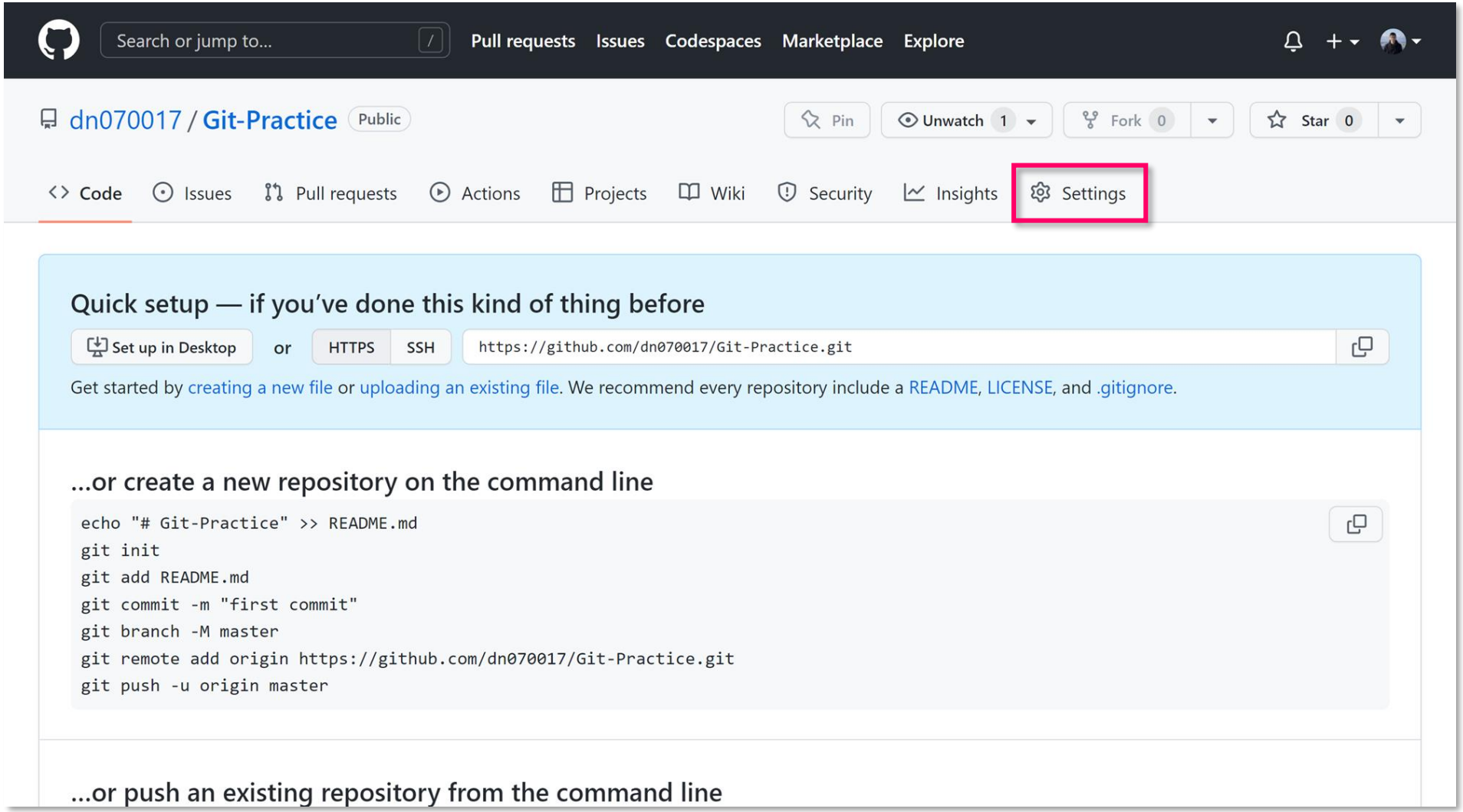
Local Repository (Collaborator)



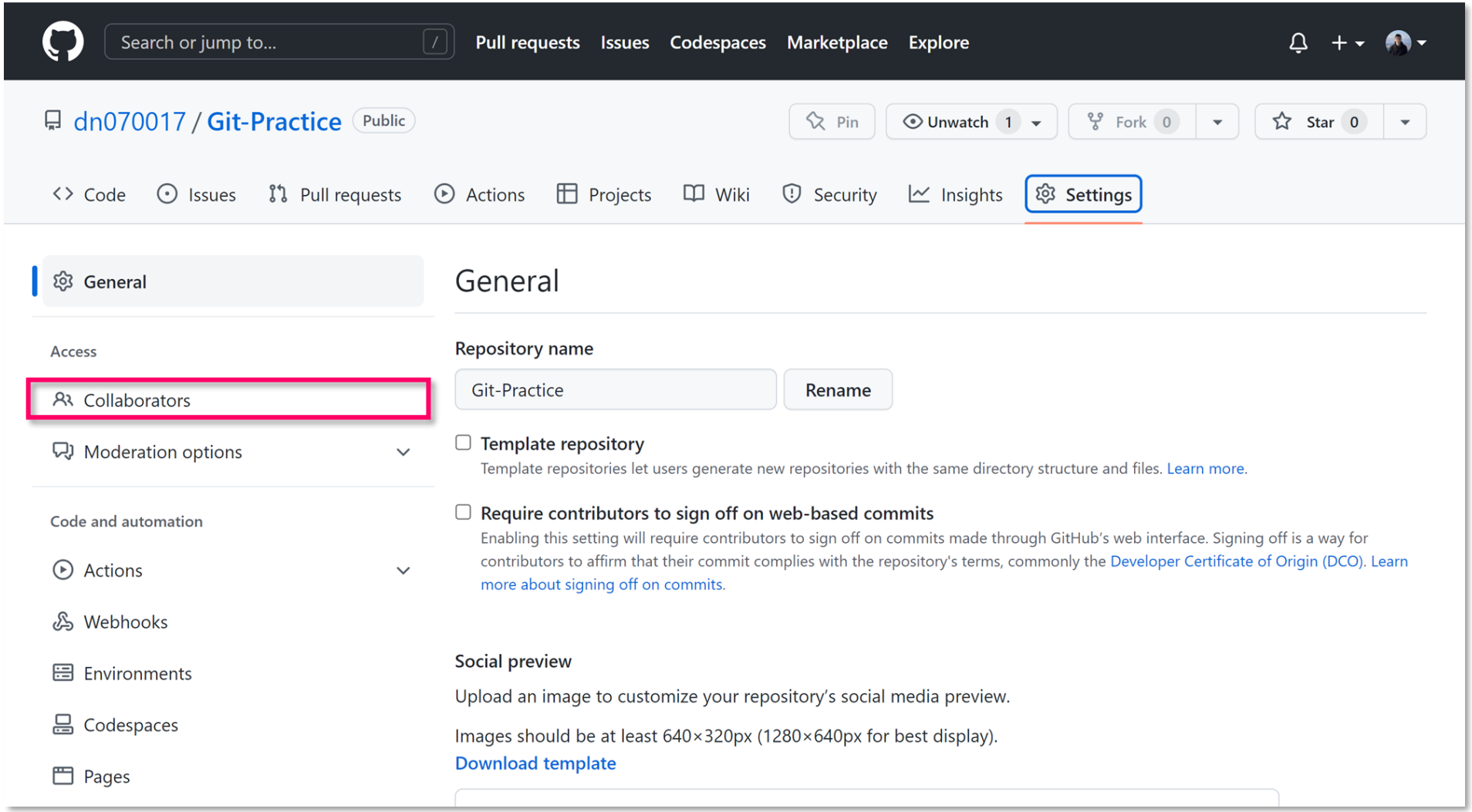

Remote Repository



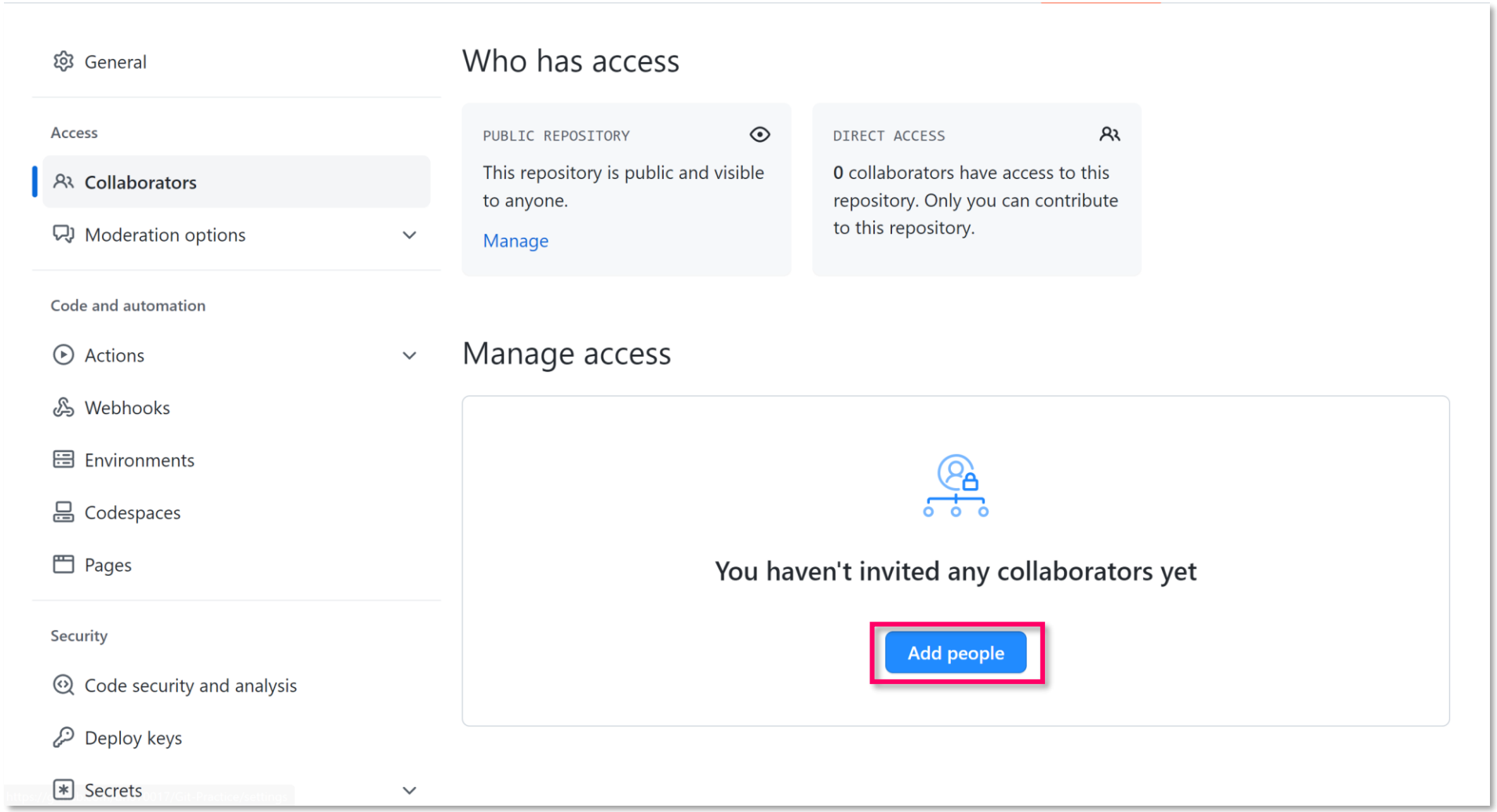
# Add Collaborators (1)



# Add Collaborators (2)



# Add Collaborators (3)



# Development

- Now we want to develop the project

- If we are using Google Drive/One Drive/Google Doc,
  - We simply change the files
  - We save the files.
  - All changes are automatically saved.
  - It will sync with the cloud automatically

- How is it done with Git?
  - Edit files.
  - Save files.
  - Record changes.
  - Sync with the cloud.

# Implement Function to the Code (1)

- Before making the changes, create a feature branch

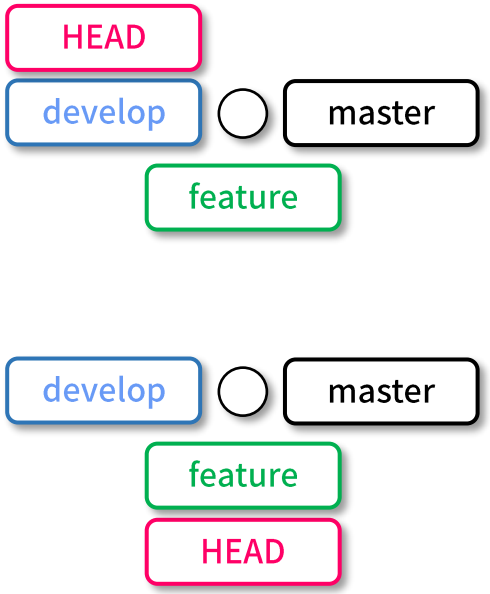
```
git branch feature/(name)
```

- Change to feature branch

```
git checkout feature/(name)
```

- Or use Gitflow command

```
git flow feature start (name)
```



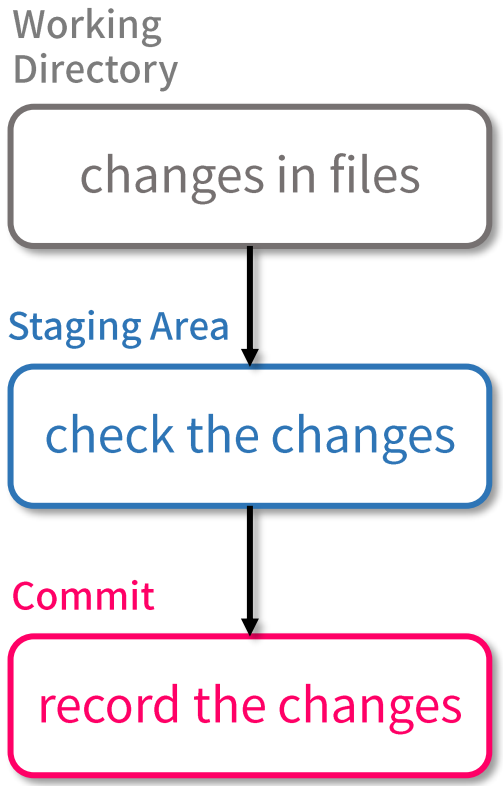
# Implement Function to the Code (2)

- Make changes to the code, save the file.
- Put the changes to the staging area

```
git add (files with changes to be recorded)
```

- Check which files are in the staging area (and other information)

```
git status
```



# Git Status

```
On branch develop
```

which branch are we

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   cavachon/utils/DataFrameUtils.py
```

changes being staged

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test/utils/AnnDataUtilsTestCase.py
```

files previously recorded, but the changes are not staged

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        cavachon/distributions/
        cavachon/environment/
```

files never been recorded

files and changes in the working directory

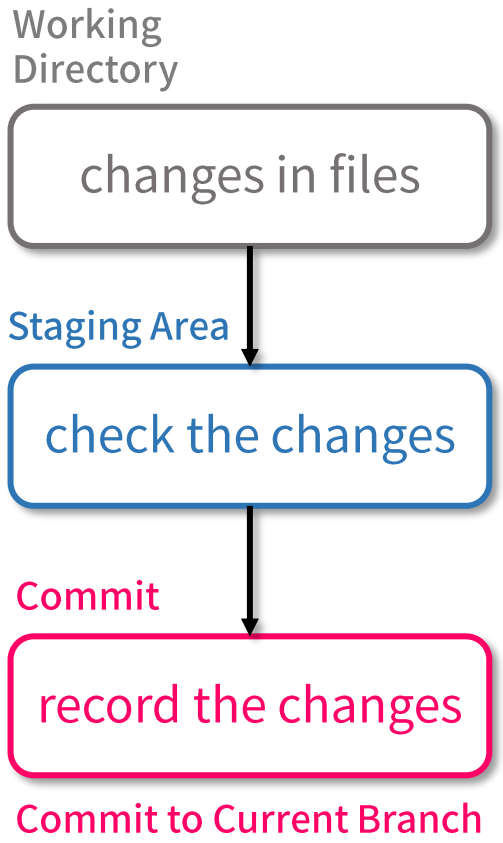
# Implement Function to the Code (3)

- Make changes to the code
- Put the changes to the staging area

```
git add (files with changes to be recorded)
```

- Record the changes into the current branch

```
git commit -m "(message)"
```



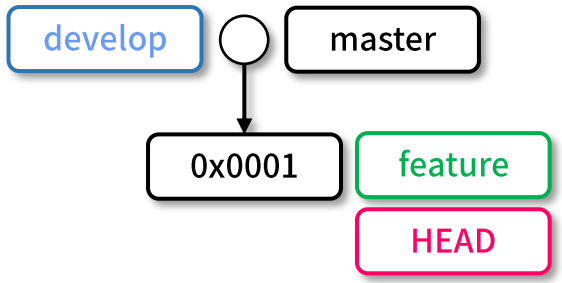
# Implement Function to the Code (4)

- Make changes to the code
- Put the changes to the staging area

```
git add (files with changes to be recorded)
```

- Record the changes into the current branch

```
git commit -m "(message)"
```




HEAD will move along with branch tag

# Implement Function to the Code (5)

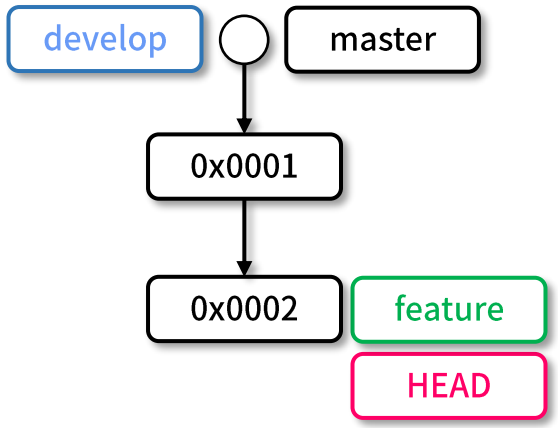
- Make changes to the code
- Put the changes to the staging area

```
git add (files with changes to be recorded)
```

- Record the changes into the current branch

```
git commit -m "(message)"
```

- Make more changes, and repeat the process



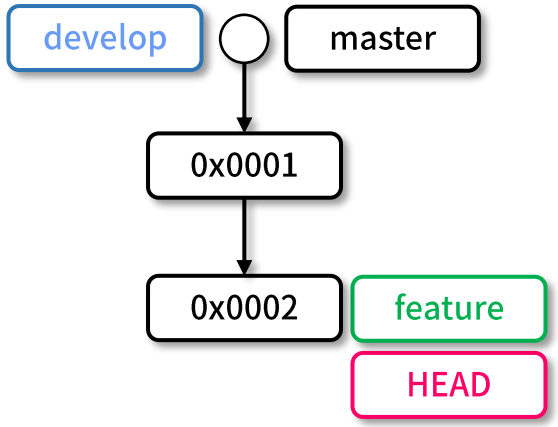
# Implement Function to the Code (6)

- Check the branch history, commits and ID.

```
git log
```

- And with other parameters

```
git log --oneline
git log --graph
```



# Git Log

```
commit 740dc023d6904c35d22ad9072d217c2078de8b45 (HEAD -> develop, origin/develop)
Author: dn070017 <dn070017@gmail.com>
Date:   Thu Apr 21 14:23:25 2022 +0200

    change wordwrap of docstrings to 72 characters

commit 8c244ad4c3b87bcf6c27639f76366feee597918b
Merge: 24ce325 116e2b9
Author: dn070017 <dn070017@gmail.com>
Date:   Thu Apr 21 09:48:56 2022 +0200

    Merge branch 'feature/implement_dataloader' into develop
```

# Sync with the Remote (1)

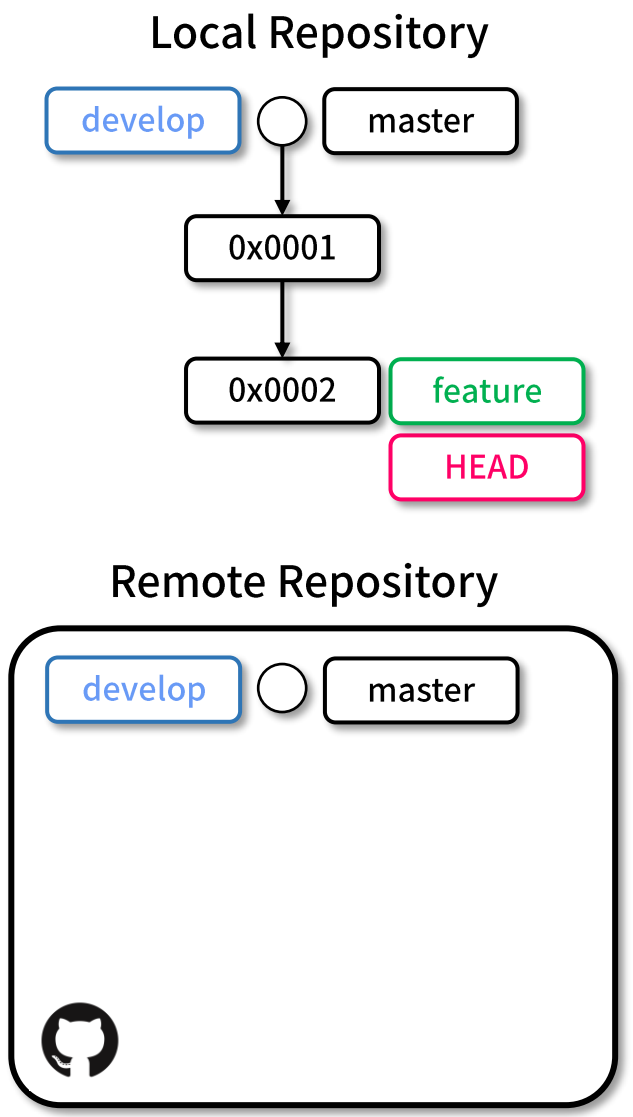
- Update the remote repository to sync with the local repository.

```
git push (remote.name) (local.branch):(remote.branch)
```

```
git push origin feature/func:feature/func
```

- Or use Gitflow command

```
git flow feature publish (name)
```



# Sync with the Remote (2)

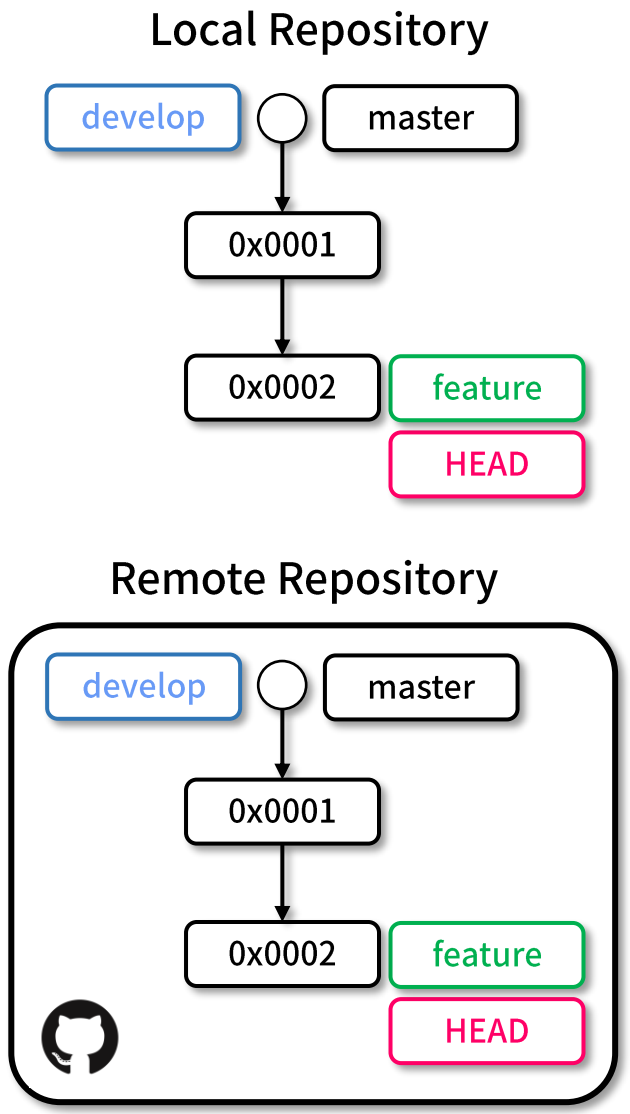
- Update the remote repository to sync with the local repository.

```
git push (remote.name) (local.branch):(remote.branch)
```

```
git push origin feature/func:feature/func
```

- Or use Gitflow command

```
git flow feature publish (name)
```



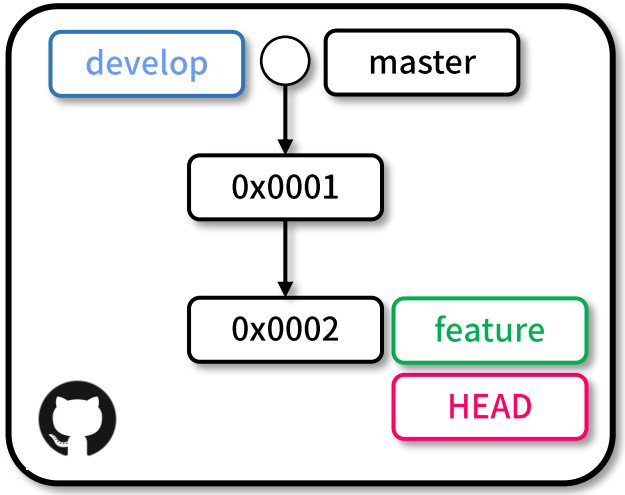
# Sync with the Remote (3)

- Update the current branch in local repository to sync with the remote repository.

```
git branch feature/func
git checkout feature/func
git pull origin feature/func
```

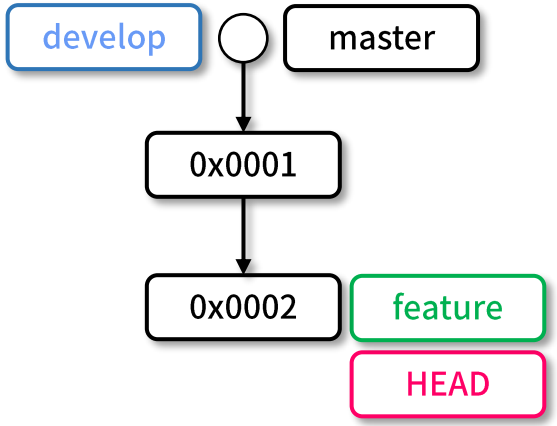
* If we do this in the second local repository that just being initialized

Remote Repository




Local Repository (Laptop 2)


Local Repository (Laptop 1)



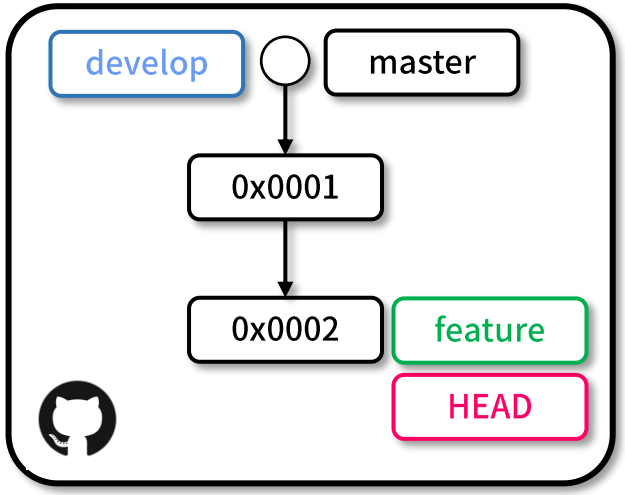
# Sync with the Remote (4)

- Update the current branch in local repository to sync with the remote repository.

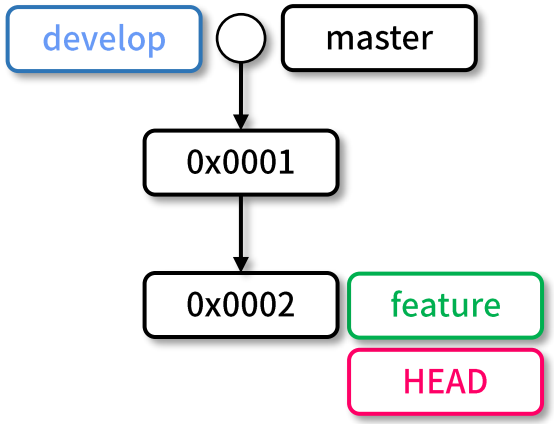
```
git branch feature/func
git checkout feature/func
git pull origin feature/func
```

* If we do this in the second local repository that just being initialized

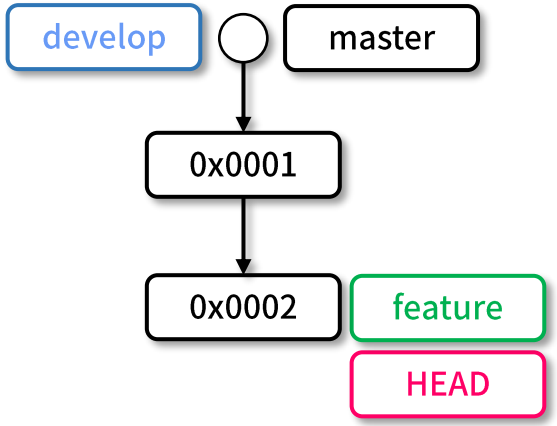
Remote Repository




Local Repository (Laptop 2)




Local Repository (Laptop 1)



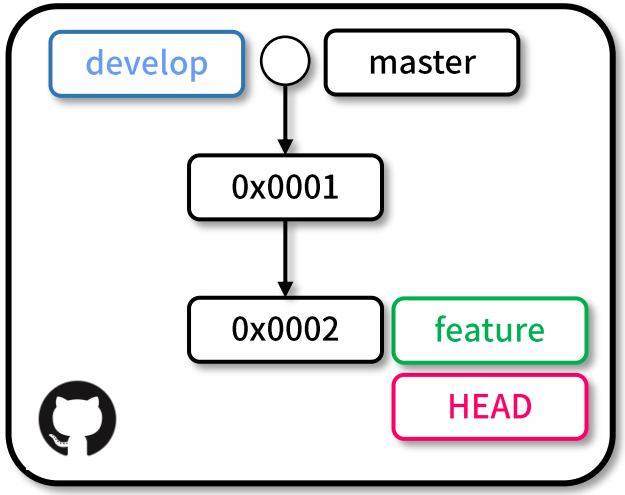
# Sync with the Remote (5)

- Or use Gitflow command

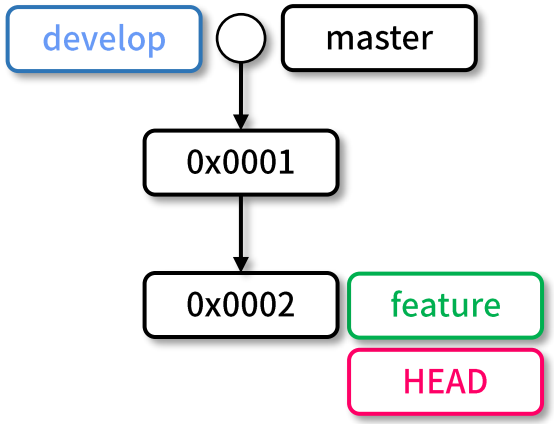
```
git branch feature/func
git flow feature pull origin (name)
```

* If we do this in the second local repository that just being initialized

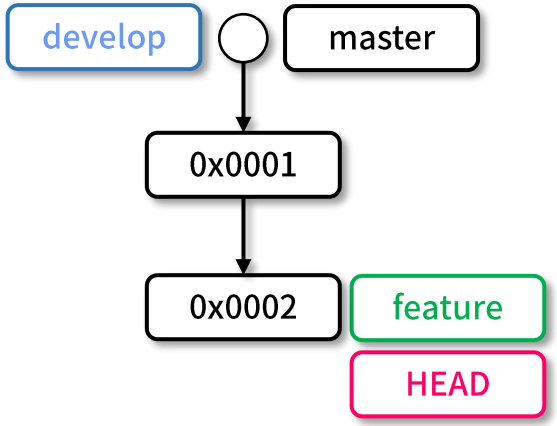
Remote Repository




Local Repository (Laptop 2)




Local Repository (Laptop 1)



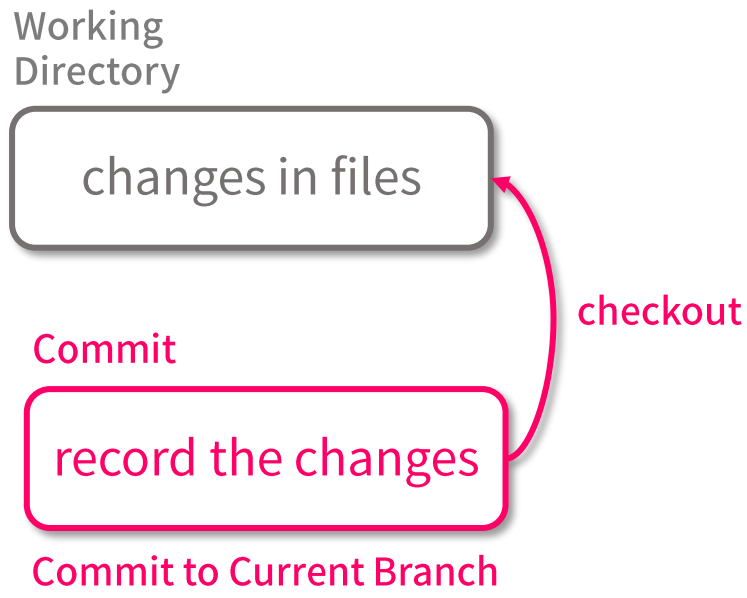
# Implement Function to the Code (7)

- If we did something wrong and not commit yet, we can revert to the latest commit of the branch

```
git checkout HEAD (files)
```

- If we did something wrong and accidentally made the commit, we can revert to the previous commit

```
git reset (commit.id)
```



# Finish the Function Development (1)

- Sync the code in feature branch to the remote repository.
- Create a pull request to the develop branch.
- Review the code.
- Merge the code
  - Merge it locally with develop branch.
  - Sync with the remote repository.

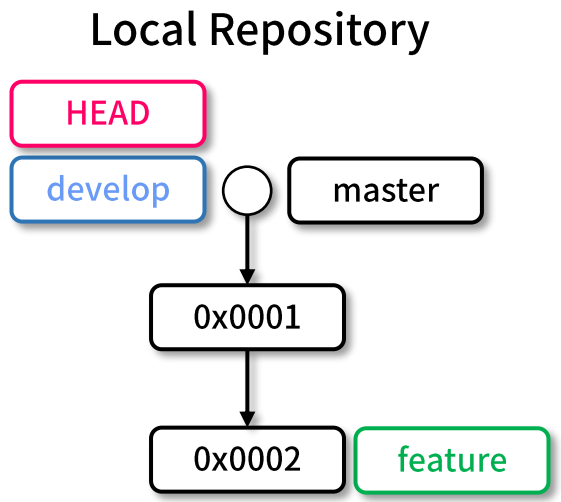
# Finish the Function Development (2)

- Merge feature branch with develop branch

```
git checkout develop
git merge --no-ff feature/func
git branch -d feature/func
```

- Or use Gitflow command

```
git flow feature finish (name)
```



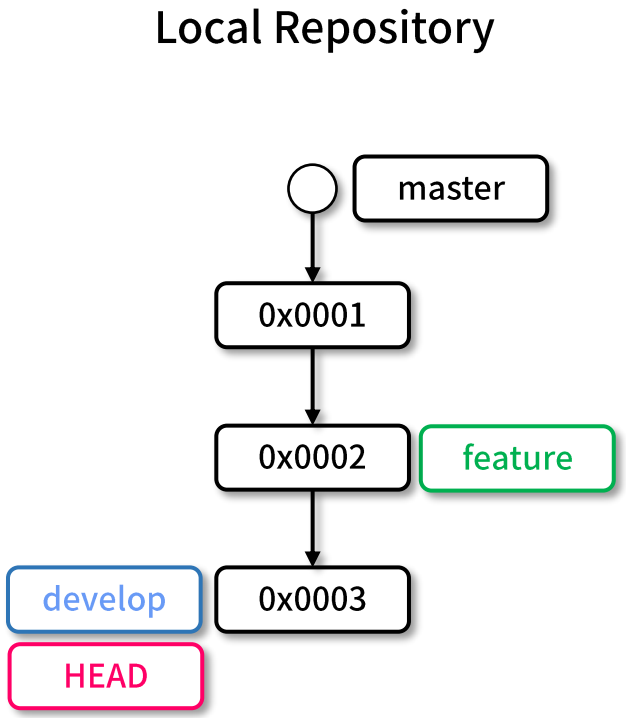
# Finish the Function Development (3)

- Merge feature branch with develop branch

```
git checkout develop
git merge --no-ff feature/func
git branch -d feature/func
```

- Or use Gitflow command

```
git flow feature finish (name)
```



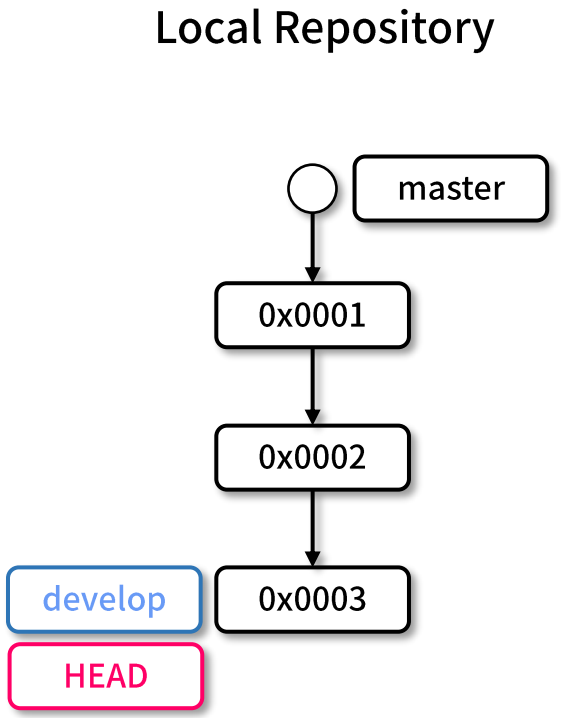
# Finish the Function Development (4)

- Merge feature branch with develop branch

```
git checkout develop
git merge --no-ff feature/func
git branch -d feature/func
```

- Or use Gitflow command

```
git flow feature finish (name)
```



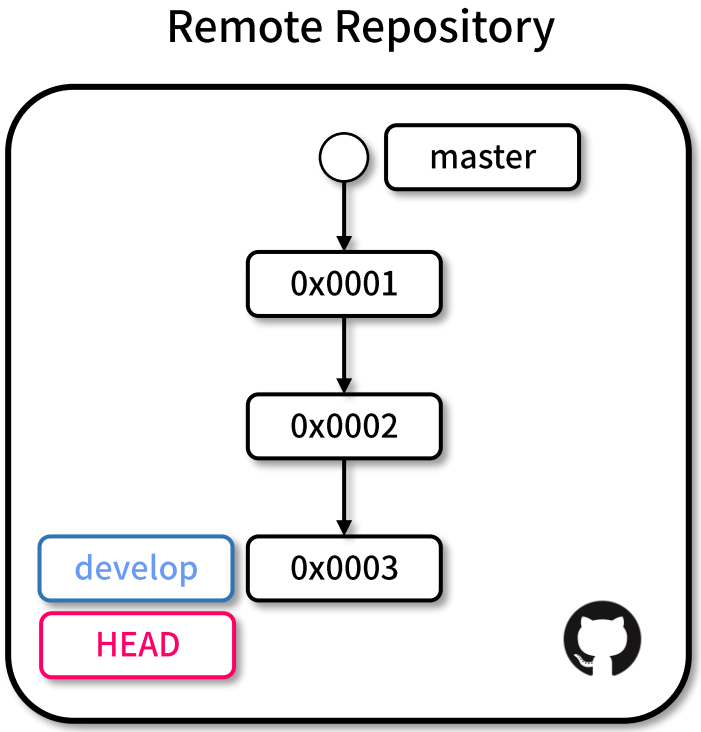
# Finish the Function Development (5)

- Update the remote repository to sync with the local repository.

```
git push origin develop:develop
```

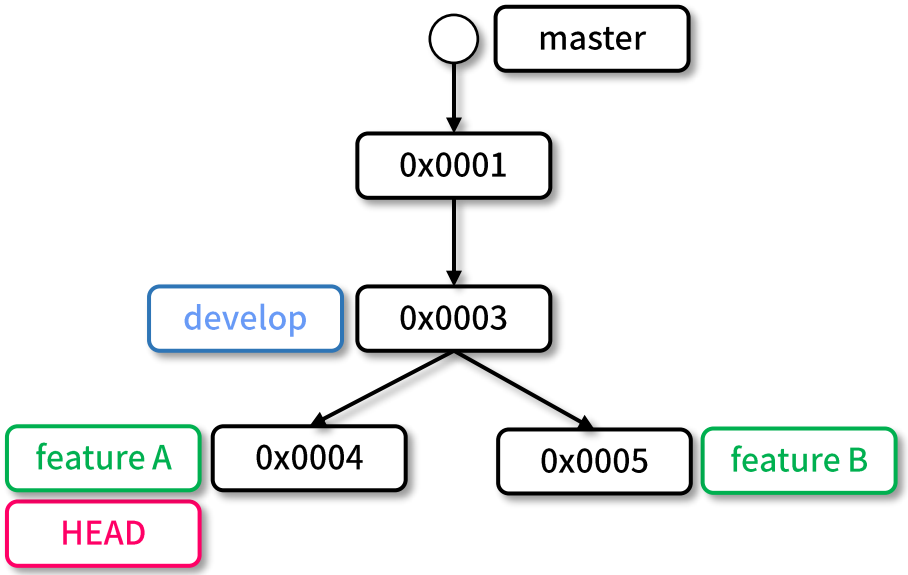
- Update the local repository to sync with the remote repository (other developers).

```
git checkout develop
git pull origin develop
```



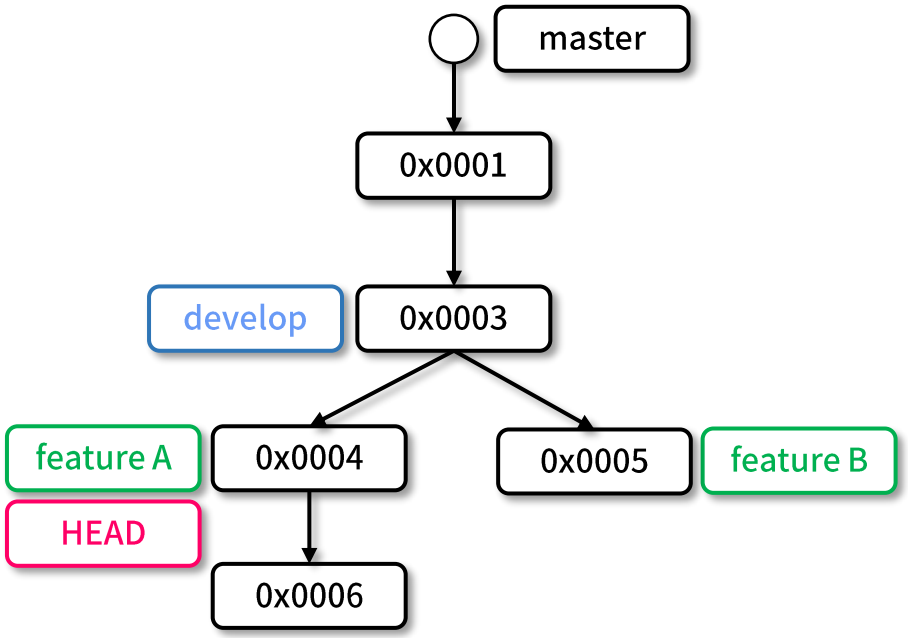
# Continue the Development (1)

Developers can work on different function based on the develop branch



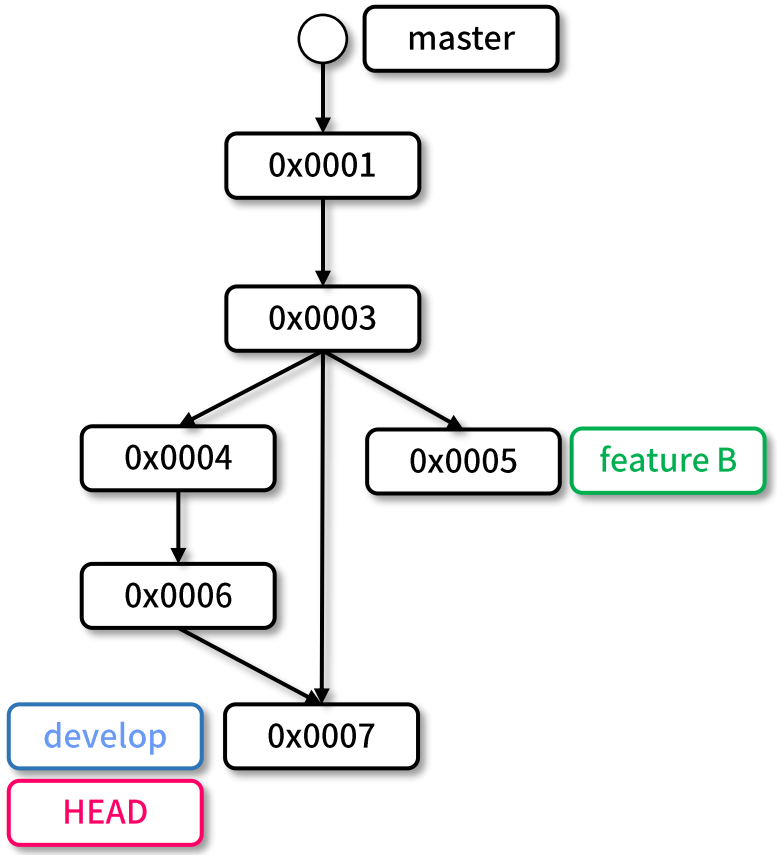
# Continue the Development (2)

Developers can work on different function based on the develop branch



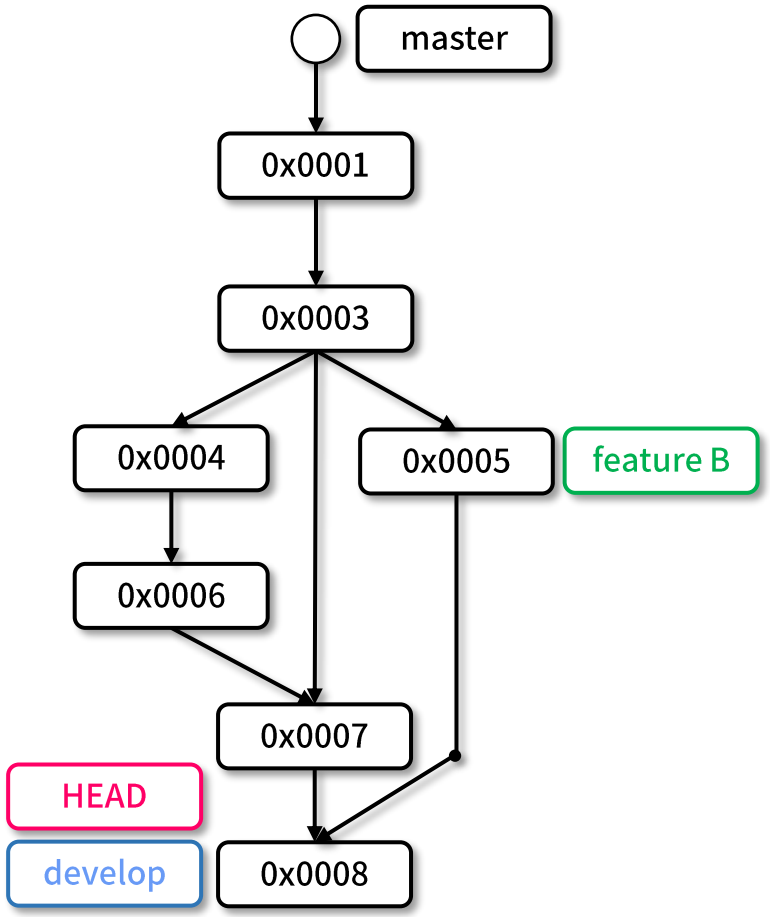
# Continue the Development (3)

Developers can work on different function based on the develop branch



# Continue the Development (4)

Developers can work on different function based on the develop branch



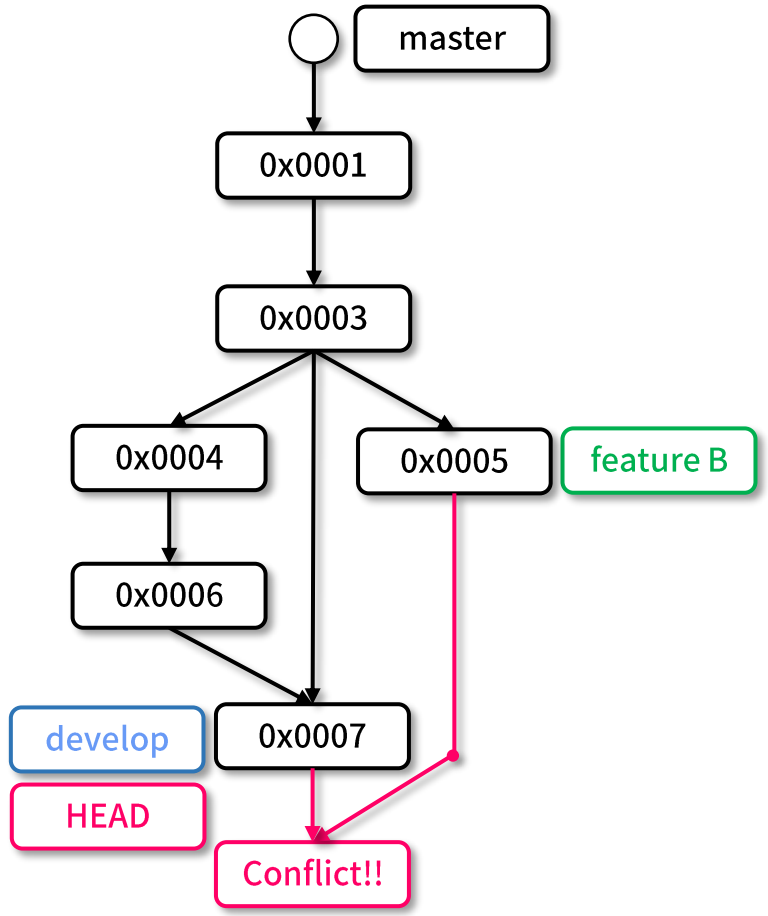
# Dealing with Conflict

- Modify the file with conflict, make a new commit

```
# Edit the conflict files
git add (conflict.files)
git commit -m "merge message"
```

- Or decide which branch should be used

```
git checkout --ours (conflict.files)
# or git checkout --theirs (confict.files)
git add (conflict.files)
git commit -m "merge message"
```



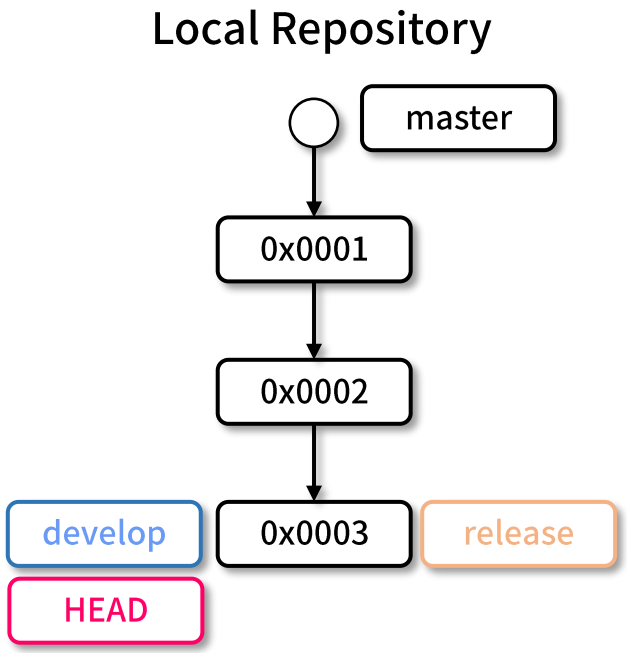
# Deployment (1)

- Create a release branch

```
git checkout develop
git branch release/(version)
git checkout release/(version)
```

- Or use Gitflow command

```
git flow release start (version)
```



# Deployment (2)

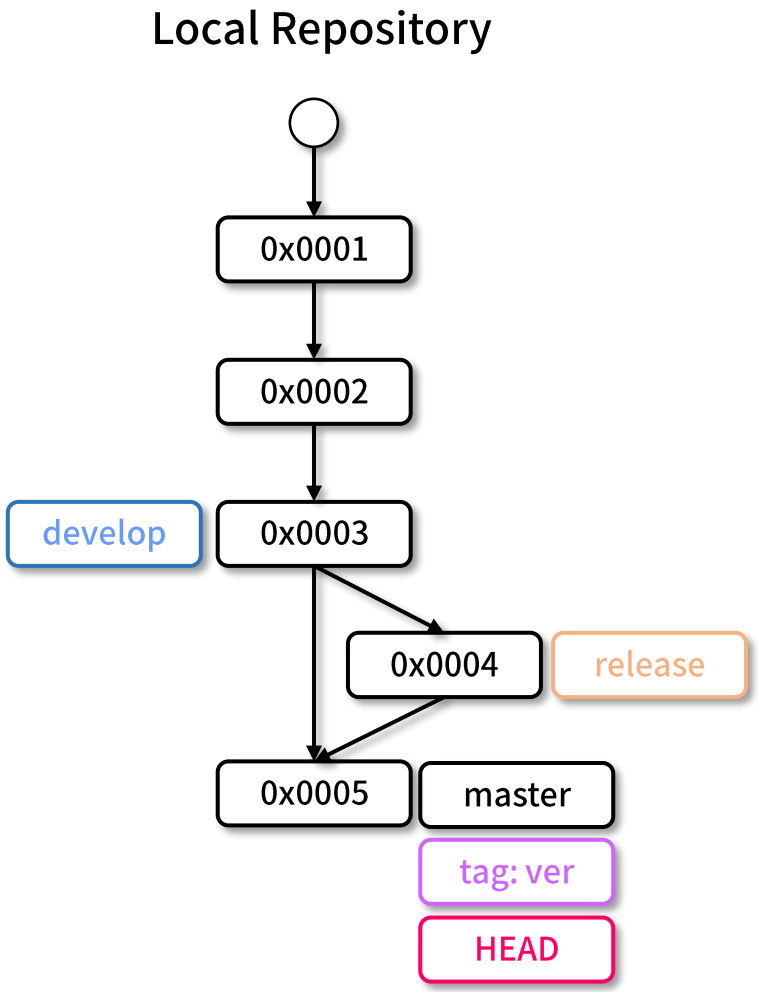
- Merge the master branch with release branch

```
git checkout master
git merge --no-ff release/(version)
git tag -a (version)
```

* tag will not move with new commit

- Then merge the develop branch with master branch, and delete release branch

```
git checkout develop
git merge --no-ff release/(version)
git branch -d release/(version)
```




* after commit in release branch

# Deployment (3)

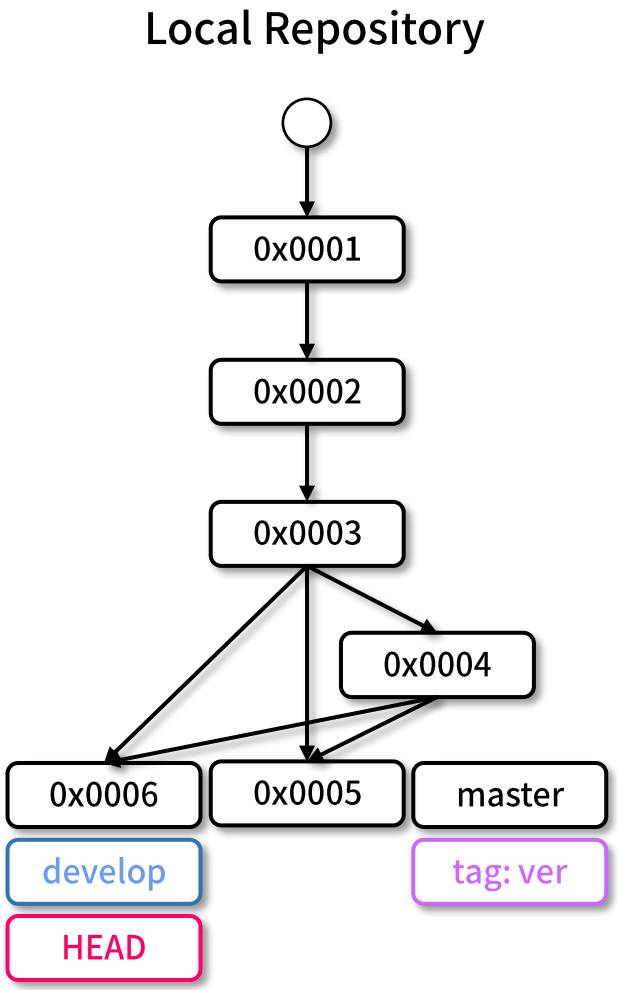
- Merge the master branch with release branch

```
git checkout master
git merge --no-ff release/(version)
git tag -a (version)
```

* tag will not move with new commit

- Then merge the develop branch with master branch, and delete release branch

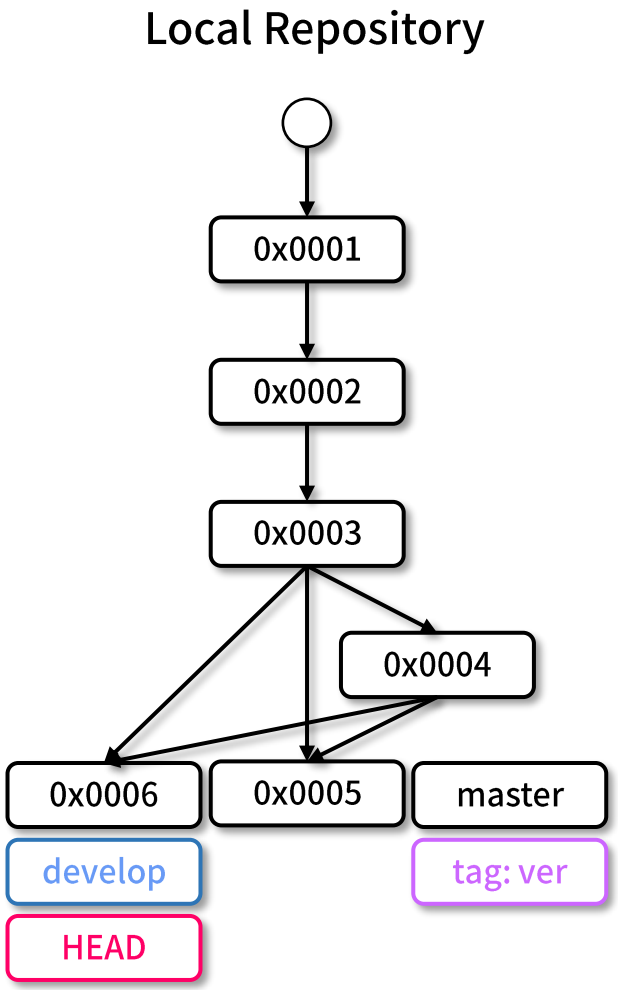
```
git checkout develop
git merge --no-ff release/(version)
git branch -d release/(version)
```



# Deployment (4)

- Or use Gitflow command

```
git flow release finish (version)
```



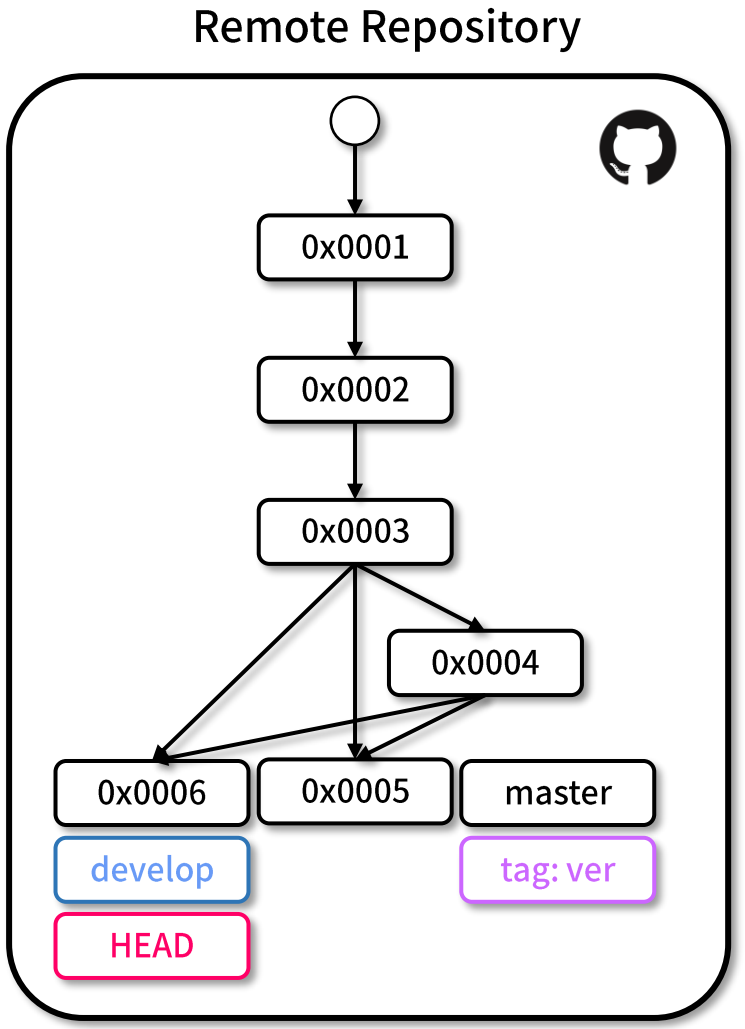
# Deployment (5)

- Update the remote repository to sync with the local repository.

```
git push origin develop:develop
git push origin master:master --tags
```

- Update the local repository to sync with the remote repository (other developers).

```
git checkout master
git pull origin master
git checkout develop
git pull origin develop
```



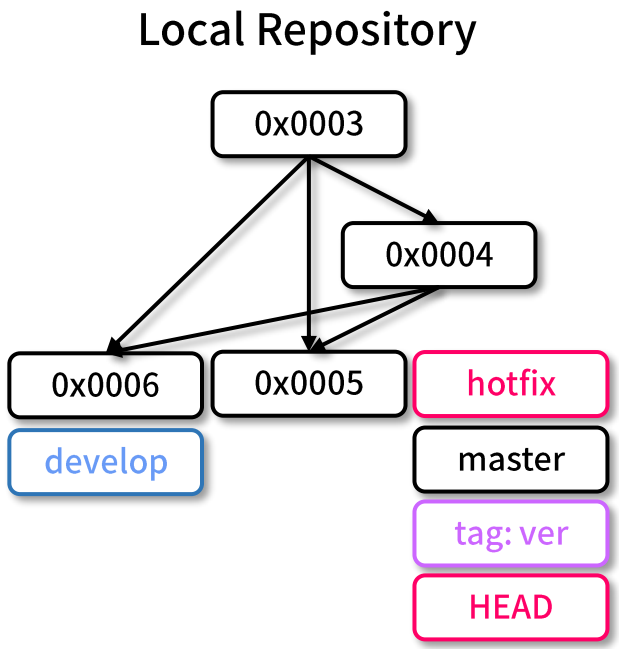
# Bug Fix on the Releases (1)

- For serious bugs that need to be fixed immediately, create hotfix branch

```
git checkout master
git branch hotfix/(version)
git checkout hotfix/(version)
```

- Or use Gitflow command

```
git flow hotfix start (version)
```



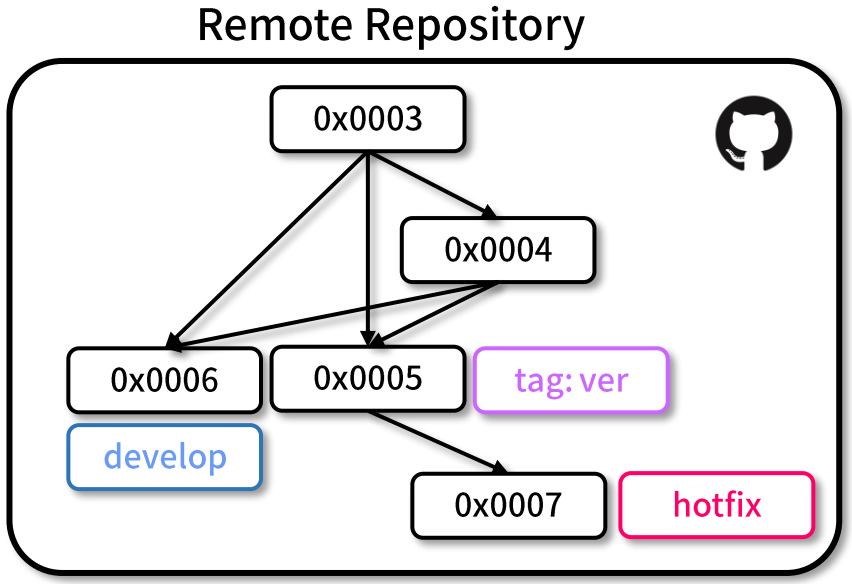
# Bug Fix on the Releases (2)

- Make changes and commit to hotfix branch
- Update the remote repository to sync with the local repository.

```
git push origin hotfix/(version):hotfix/(version)
```

- Update the local repository to sync with the remote repository (other developers).

```
git branch hotfix/(version)
git checkout hotfix/(version)
git pull origin hotfix/(version)
```



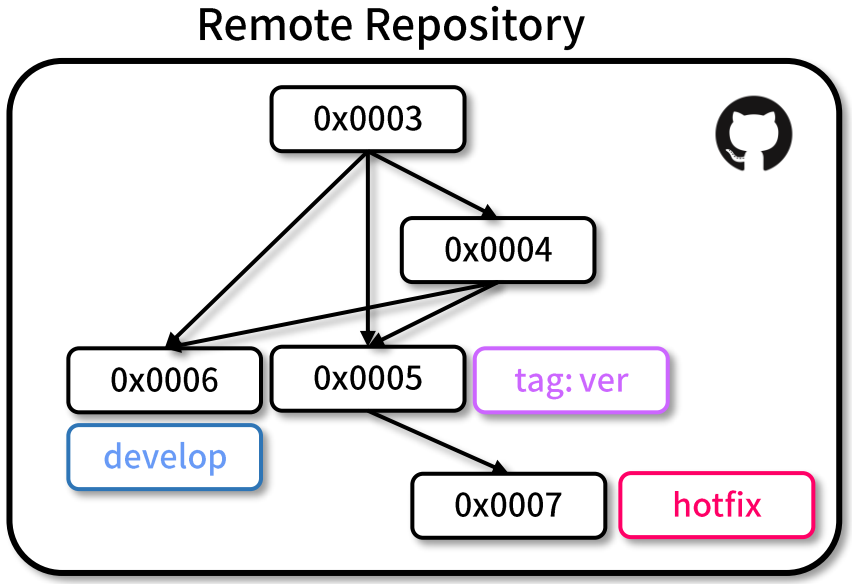
# Bug Fix on the Releases (3)

- Or use Gitflow command
- Update the remote repository to sync with the local repository.

```
git flow hotfix publish (version)
```

- Update the local repository to sync with the remote repository (other developers).

```
git flow hotfix pull origin (version)
```



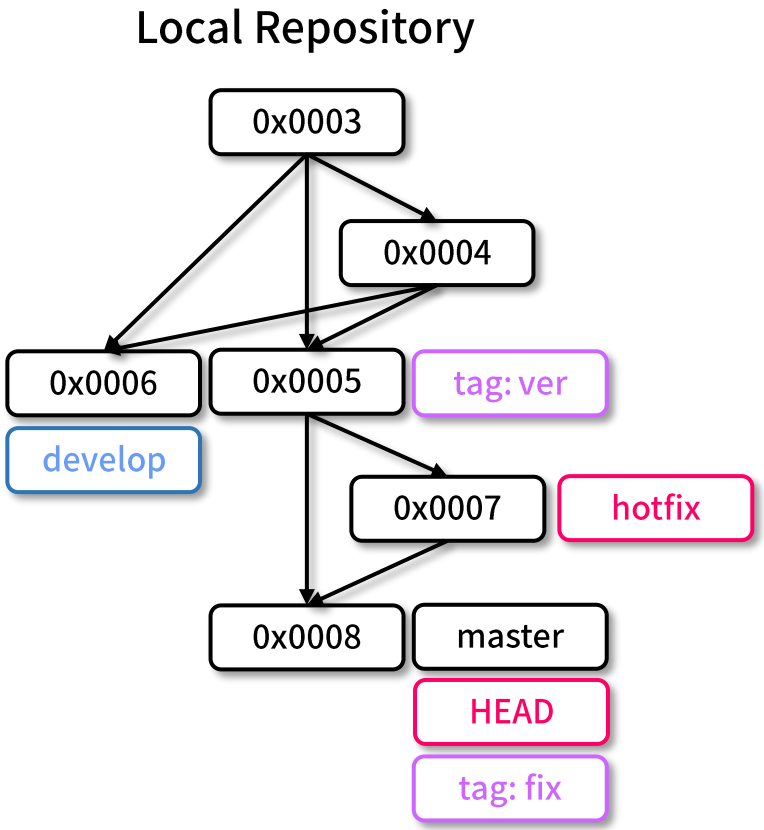
# Bug Fix on the Releases (4)

- Merge the master branch with hotfix branch

```
git checkout master
git merge --no-ff hotfix/(version)
git tag -a (version)
```

- Merge the develop branch with hotfix branch

```
git checkout develop
git merge --no-ff hotfix/(version)
git branch -d hotfix/(version)
```



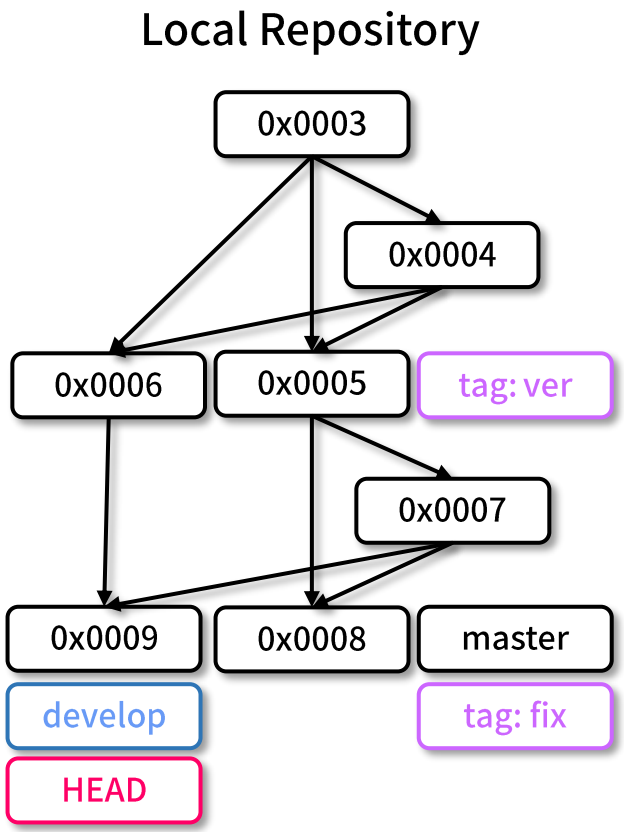
# Bug Fix on the Releases (5)

- Merge the master branch with hotfix branch

```
git checkout master
git merge --no-ff hotfix/(version)
git tag -a (version)
```

- Merge the develop branch with hotfix branch

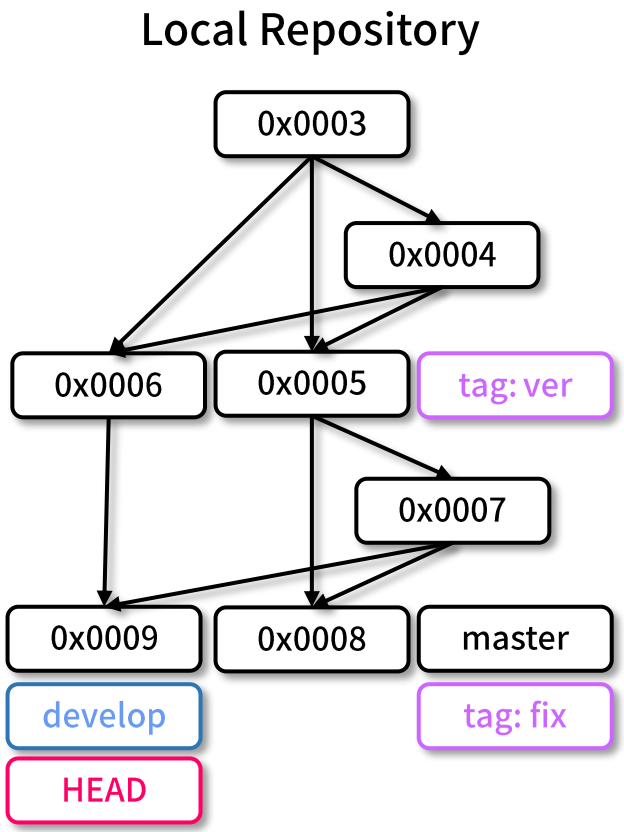
```
git checkout develop
git merge --no-ff hotfix/(version)
git branch -d hotfix/(version)
```



# Bug Fix on the Releases (6)

- Or use Gitflow command

```
git flow hotfix finish (version)
```



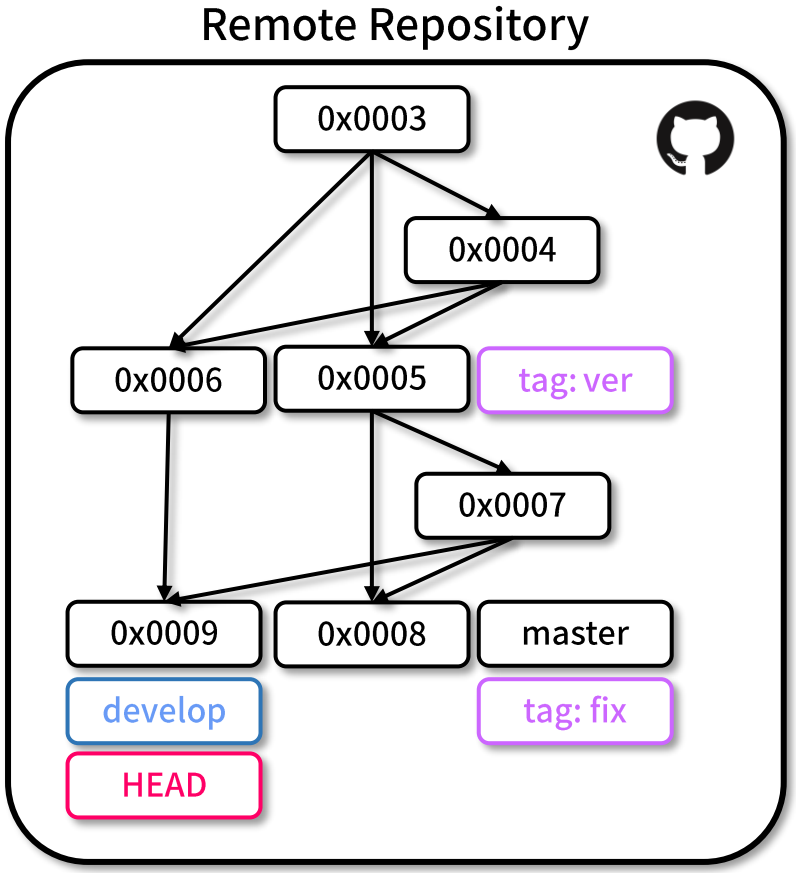
# Bug Fix on the Releases (7)

- Update the remote repository to sync with the local repository.

```
git push origin develop:develop
git push origin master:master --tags
```

- Update the local repository to sync with the remote repository (other developers).

```
git checkout master
git pull origin master
git checkout develop
git pull origin develop
```



# Summary

- Start the project
- Development (develop → feature → develop)
  - Implement a function to the code
  - Sync with the remote
  - Finish the function development
  - Dealing with conflict
- Deployment (develop → release → master, develop)
- Bug fix on the releases (master → hotfix → master, develop)

nordic-compbio.iscbsc.org

rsg-norway.iscbsc.org

Feedback

Join Slack

@RSGNorway

rsg-norway@iscbsc.org

# Thanks

# Appendix - Useful Commands

- Code editing

```
git add (files with changes to be recorded)

git commit -m "(message)"
```

- Branch

```
git branch (create branch name)

git checkout (branch name)
```

```
git merge --no-ff (branch to merge)
```

- Remote

```
git push (remote.name) (local.branch):(remote.branch)

git pull (remote.name) (branch.name)
```

# Appendix - Reset

```
git reset [--mixed] [commit.id]
```

- all the commited changes will be kept in working directory
- all the staged changes will be kept in working directory
- all the local changes will be kept in working directory

```
git reset --hard [commit.id]
```

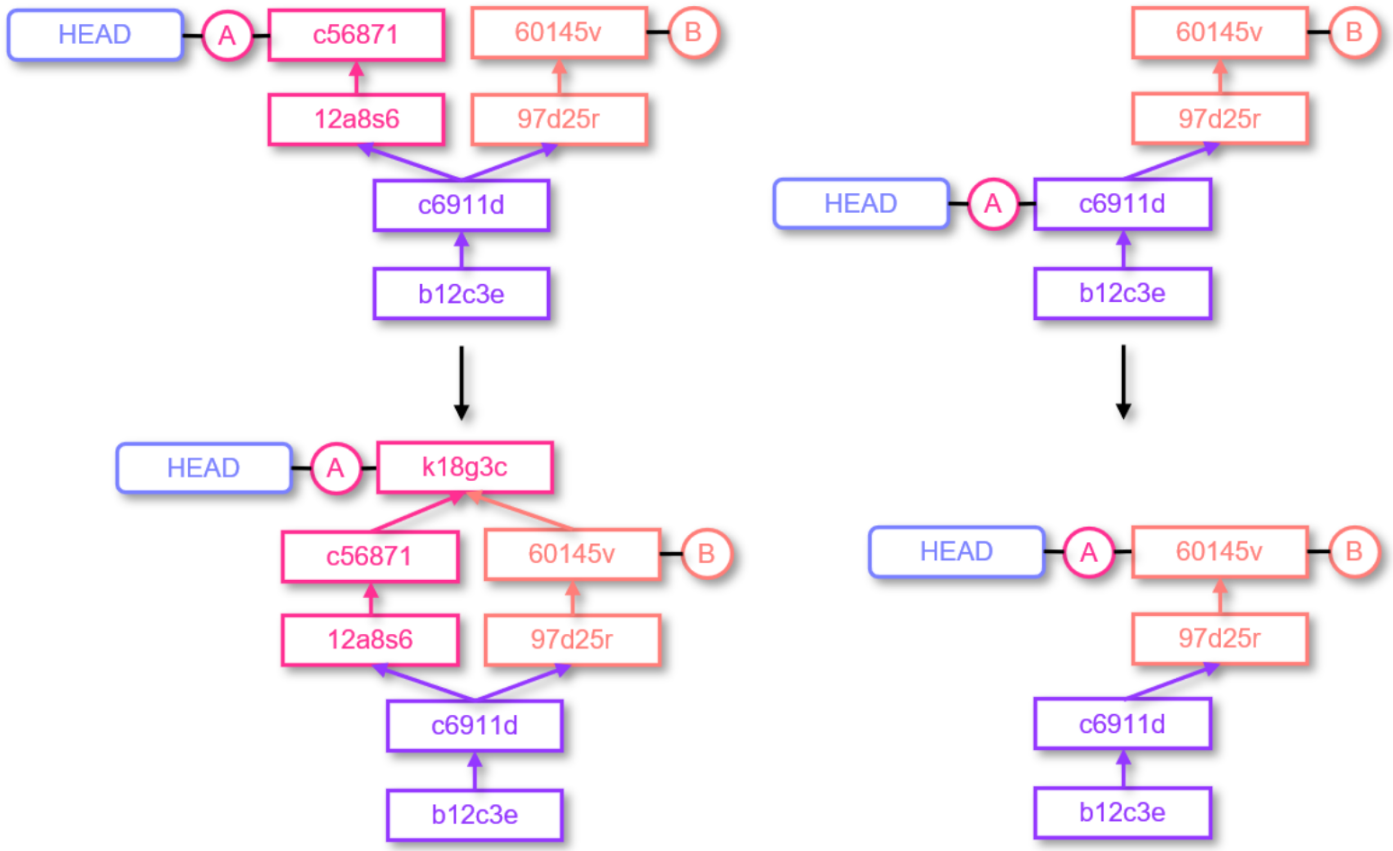
- all the commited changes will be removed
- all the staged changes will be removed
- all the local changes will be removed

```
git reset --soft [commit.id]
```

- all the commited changes will be kept in staging area
- all the staged changes will be kept in working directory
- all the local changes will be kept in working directory

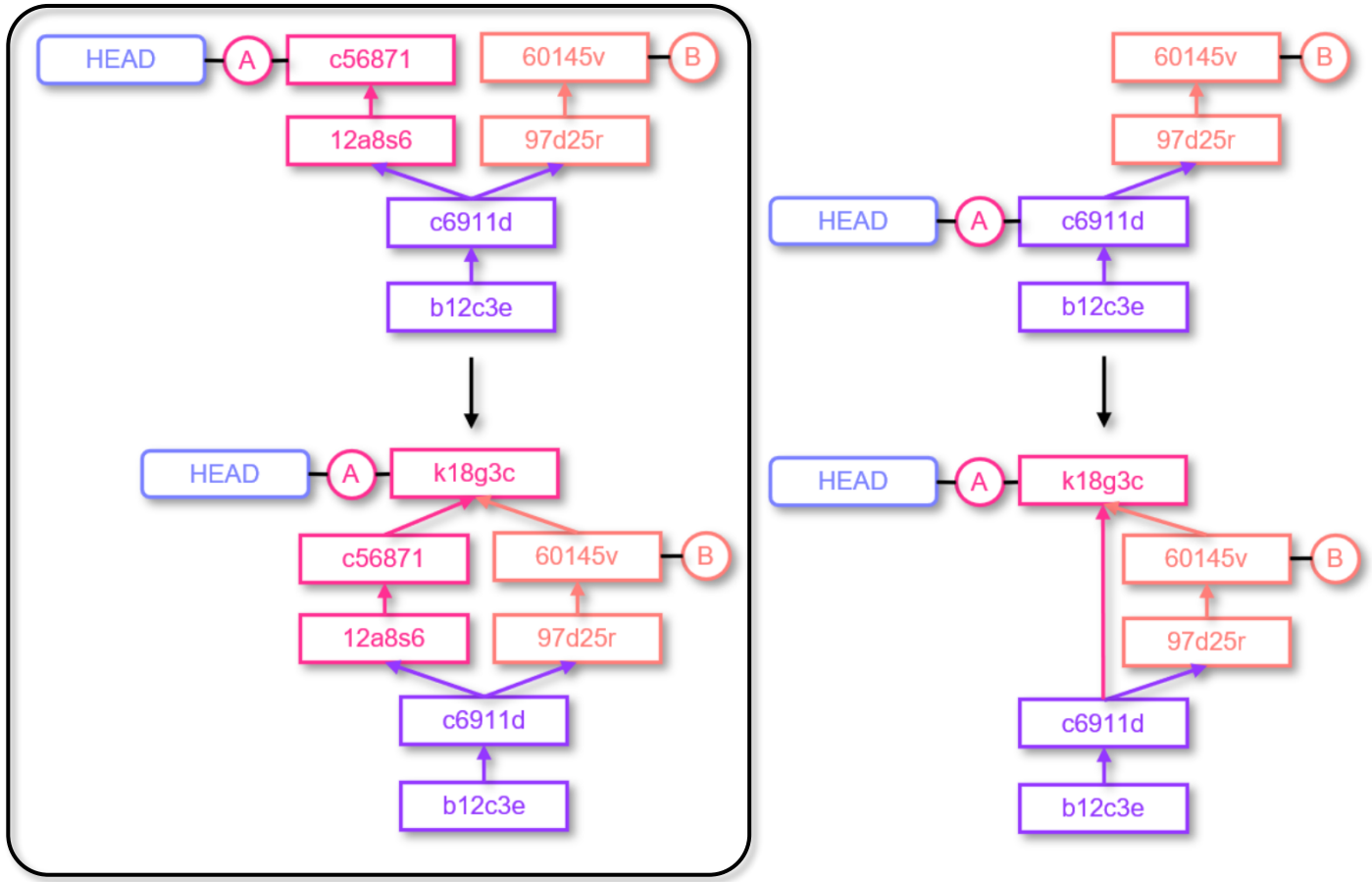
# Appendix – Merge (1)

with fast-forward



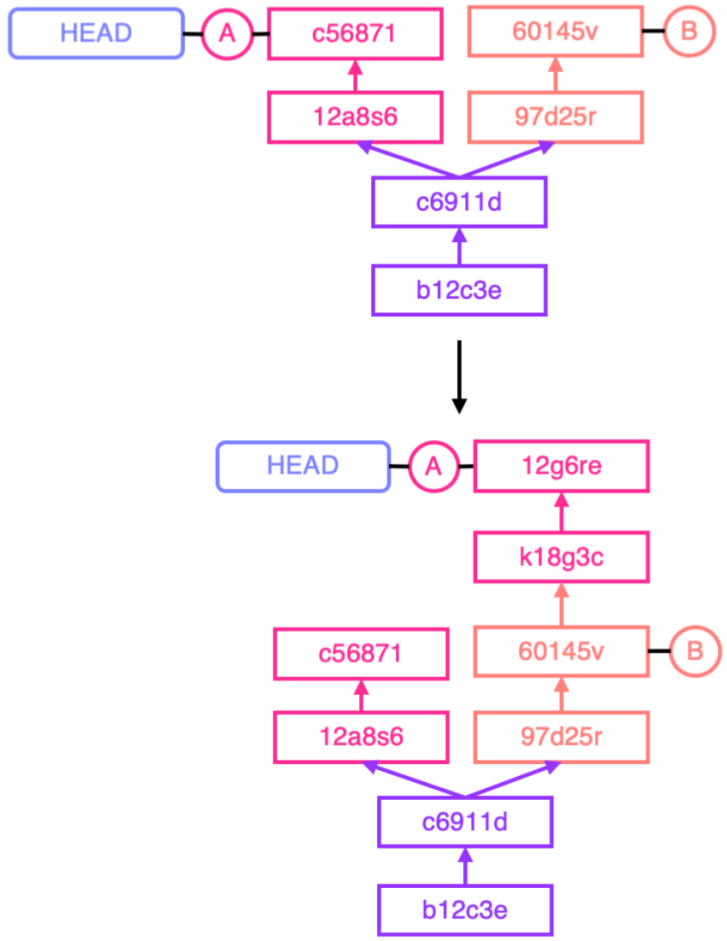
# Appendix – Merge (2)

without fast-forward




same as fast-forward

# Appendix – Rebase



# Appendix – Useful References

- Atlassian Bitbucket Git Tutorial
- Git Cheatsheet
- Gitflow Cheatsheet