

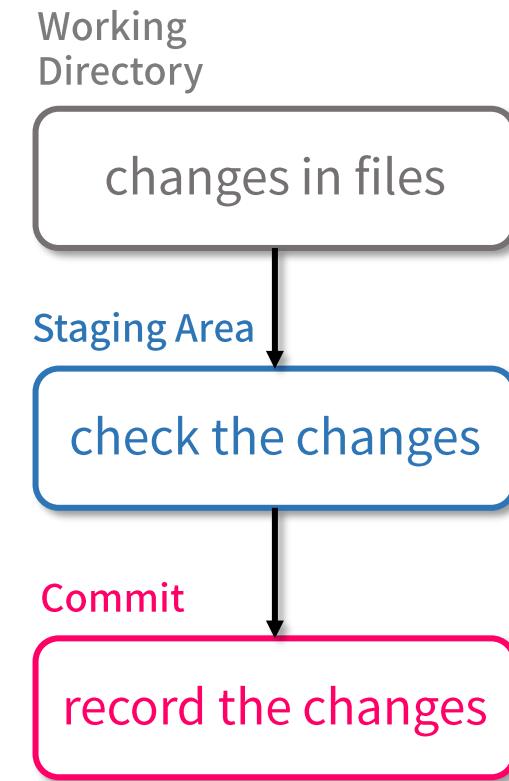
Introduction to Git and Development Cycle

2022-12-12

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

What is Git (1)

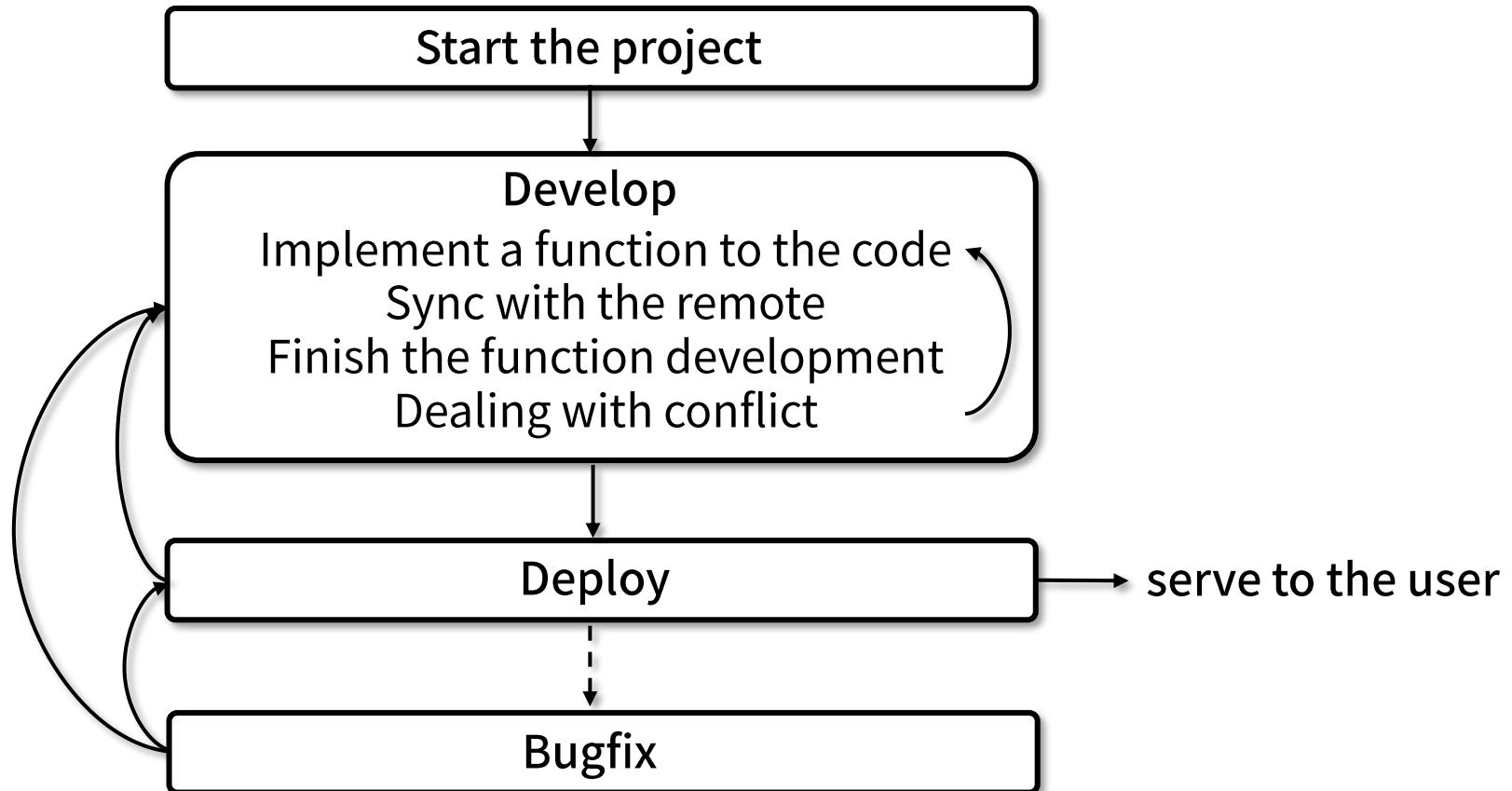
- **Git**
 - An open-source distributed version control system.
 - Lightweight and efficient.
 - Disposable experimentation.
 - Suitable for collaboration.
- **Gitflow**
 - Strategy for managing Git branches.
- **Important Terminologies**
 - working directory
 - repository
 - staging area
 - commit
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

The screenshot shows the GitHub homepage. At the top right, there is a pink callout bubble with the text "Create New Repository" and a red arrow pointing to the "+" button in the top right corner of the header bar. The header also includes links for Pull requests, Issues, Codespaces, Marketplace, and Explore.

Left Sidebar:

- User profile: dn070017
- Top Repositories:**
 - New
 - Find a repository...
- Repository list:
 - kuijjerlab/CAVACHON
 - dn070017/CAVACHON
 - kuijjerlab/PySNAIL
 - dn070017/PySNAIL
 - dn070017/Data-Structures-Algorithms-Python
 - dn070017/Missmi-Intelligence-Frontend
 - dn070017/Git
- Show more
- Recent activity:**

When you take actions across GitHub, we'll provide links to that

Middle Content:

- Following** (selected) and **For you** (Beta)
- Activity feed:
 - hsiaoyi0504 starred MonashProteomics/FragPipe-Analyst 5 days ago
 - MonashProteomics/FragPipe-Analyst
 - HTML
 - 2 stars
 - Updated Dec 6
 - hsiaoyi0504 released v0.10 of MonashProteomics/FragPipe-Analyst 6 days ago
 - MonashProteomics / FragPipe-Analyst
v0.10
 - MorvanZhou starred MorvanZhou/marchingSquares 7 days ago
 - MorvanZhou/marchingSquares
 - Python
 - 2 stars
 - Updated Dec 2

Right Sidebar:

- Latest changes:**
 - 5 hours ago: GitHub app in Slack and Microsoft Teams —...
 - 17 hours ago: GitHub Copilot is now available for invoiced GitHub Enterprise...
 - 18 hours ago: Updated timeline for the deprecation of CodeQL Action...
 - 2 days ago: Extending Checks Retention Policies to GHES
- [View changelog →](#)
- Explore repositories:**
 - GuyTeichman/RNAlysis: RNA sequencing analysis software
 - Python ★ 57
 - YuLab-SMU/enrichplot: Visualization of Functional Enrichment

Create a Project (3)

Screenshot of the GitHub 'Create a new repository' form.

The GitHub header bar is visible, showing the GitHub logo, a search bar with placeholder text 'Search or jump to...', and navigation links for Pull requests, Issues, Codespaces, Marketplace, and Explore. On the right, there are icons for notifications, a plus sign, and a user profile.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *  dn070017 **Repository name ***

Great repository names are short and memorable. Need inspiration? How about [expert-lamp](#)?

Description (optional)

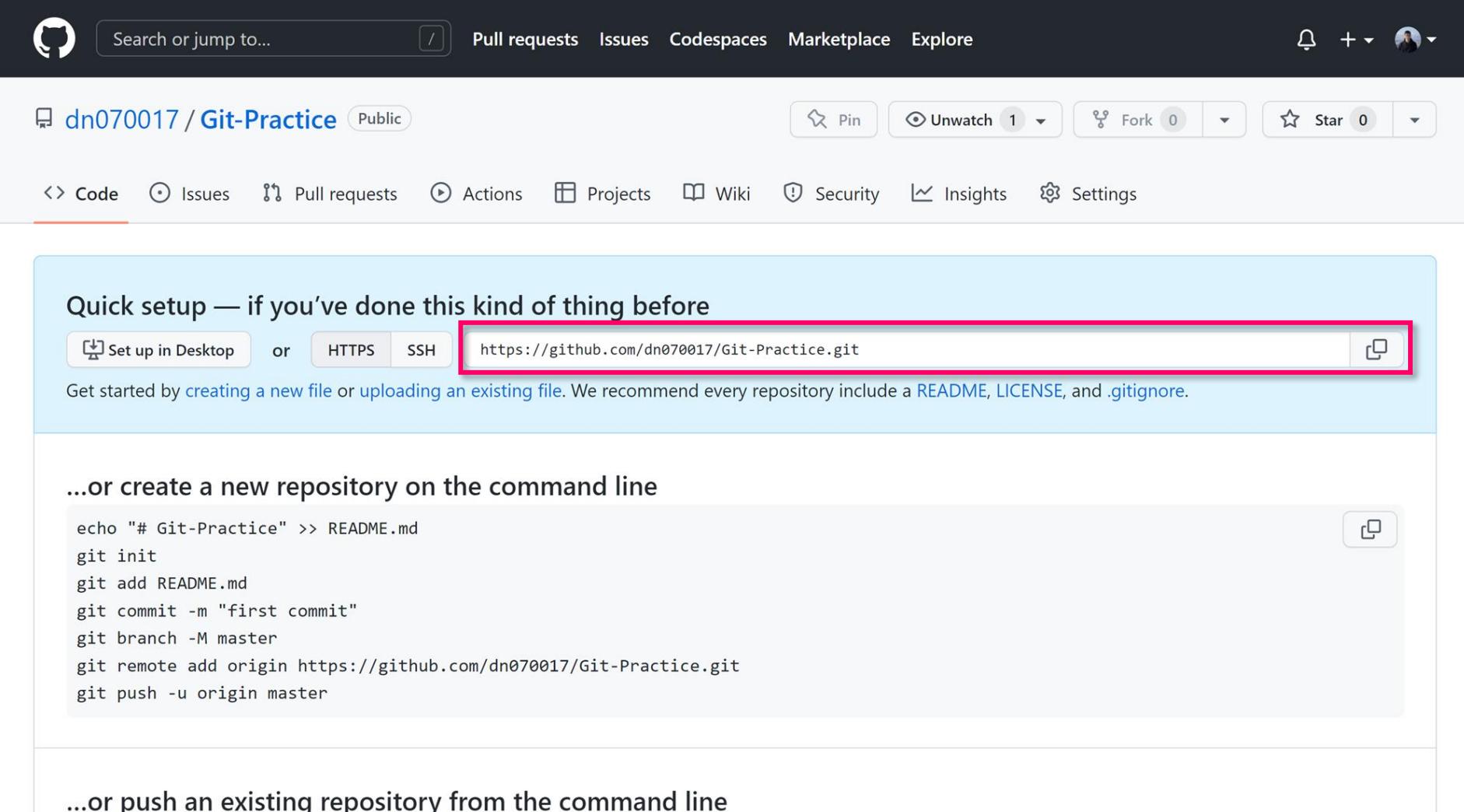
 **Public**
Anyone on the internet can see this repository. You choose who can commit.

 **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. [Learn more](#).

Create a Project (4)



The screenshot shows a GitHub repository page for "dn070017/Git-Practice". The top navigation bar includes links for Pull requests, Issues, Codespaces, Marketplace, and Explore. Below the repository name, there are buttons for Pin, Unwatch (with 1 follower), Fork (with 0 forks), and Star (with 0 stars). The main menu below the repository name includes Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A red box highlights the project URL "https://github.com/dn070017/Git-Practice.git".

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#) <https://github.com/dn070017/Git-Practice.git> [Copy](#)

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# Git-Practice" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M master
git remote add origin https://github.com/dn070017/Git-Practice.git
git push -u origin master
```

...or push an existing repository from the command line

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

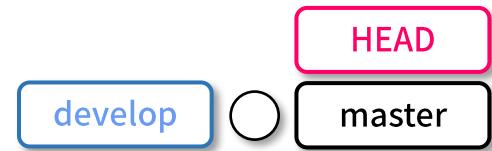
```
git branch develop
```



Create a Project (8)

- Create a develop branch

```
git branch develop
```



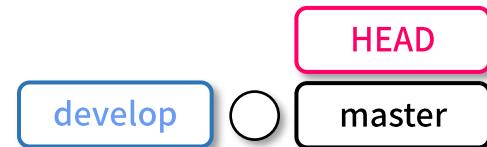
Create a Project (9)

- Create a develop branch

```
git branch develop
```

- Change to develop branch

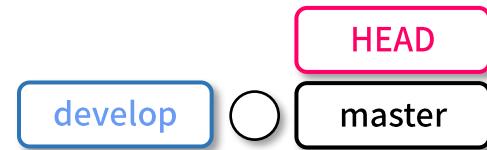
```
git checkout develop
```



Create a Project (10)

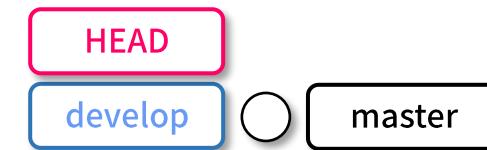
- Create a develop branch

```
git branch develop
```



- Change to develop branch

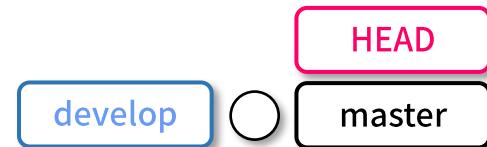
```
git checkout develop
```



Create a Project (11)

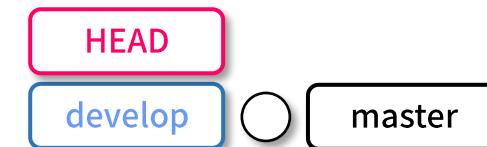
- Create a develop branch

```
git branch develop
```



- Change to develop branch

```
git checkout develop
```



- Check branch info

```
git branch
```

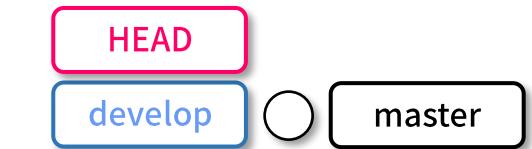
Sync with the Remote (1)

- 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
```

Local Repository



Remote Repository



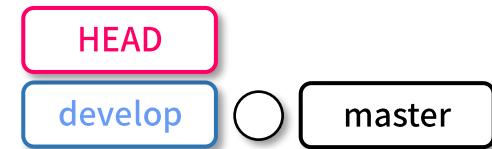
Sync with the Remote (2)

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

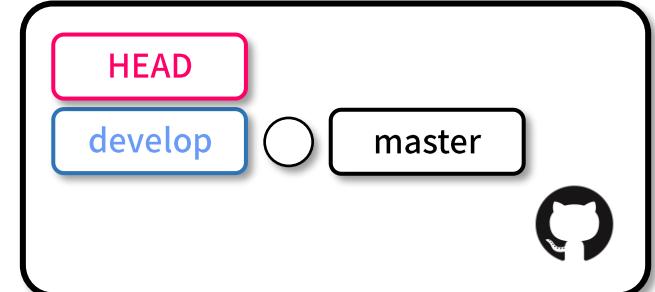
```
git push (remote.name) (local.branch):(remote.branch)
```

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

Local Repository



Remote Repository



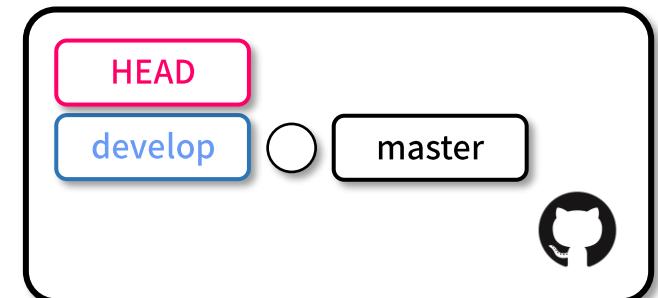
View Project as Collaborator (1)

- Clone the repository.

Local Repository (Collaborator)

```
git clone (URL)
```

Remote Repository

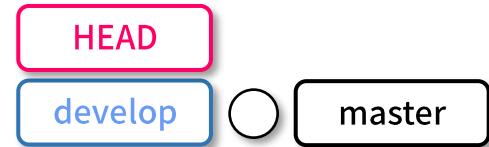


View Project as Collaborator (2)

- Clone the repository.

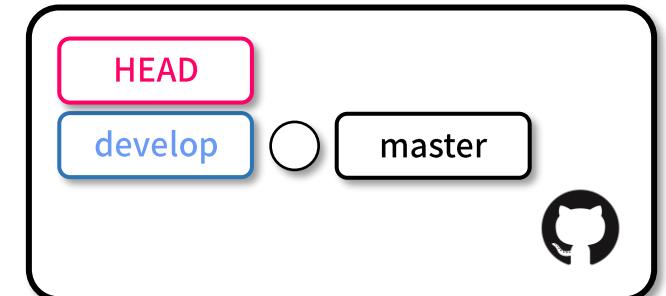
```
git clone (URL)
```

Local Repository (Collaborator)

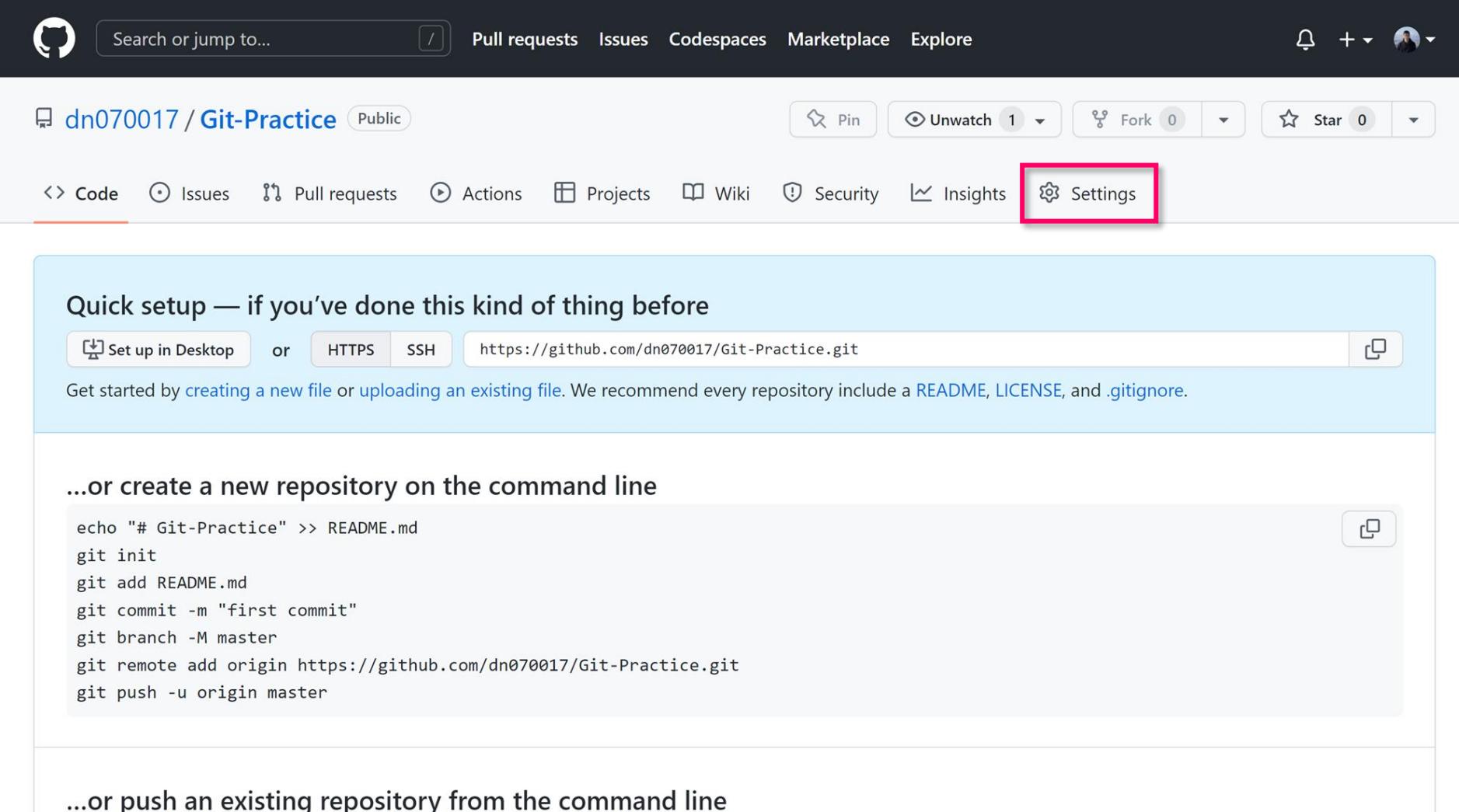


By default, everyone can view any public repository, but only the collaborator can edit the repository.

Remote Repository



Add Collaborators (1)



The screenshot shows a GitHub repository page for "dn070017/Git-Practice". The "Settings" tab is highlighted with a red box. The page includes sections for quick setup, command-line repository creation, and pushing existing repositories.

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#) <https://github.com/dn070017/Git-Practice.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# Git-Practice" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M master
git remote add origin https://github.com/dn070017/Git-Practice.git
git push -u origin master
```

...or push an existing repository from the command line

Add Collaborators (2)

The screenshot shows the GitHub repository settings page for 'dn070017/Git-Practice'. The 'Settings' tab is selected. On the left, a sidebar lists repository settings categories: General, Access, Code and automation, and Pages. The 'Access' category is expanded, showing the 'Collaborators' section, which is highlighted with a red box. Other sections in the sidebar include Moderation options, Actions, Webhooks, Environments, Codespaces, and Pages.

General

Repository name
Git-Practice [Rename](#)

Template repository
Template repositories let users generate new repositories with the same directory structure and files. [Learn more](#).

Require contributors to sign off on web-based commits
Enabling this setting will require contributors to sign off on commits made through GitHub's web interface. Signing off is a way for contributors to affirm that their commit complies with the repository's terms, commonly the [Developer Certificate of Origin \(DCO\)](#). [Learn more about signing off on commits](#).

Social preview
Upload an image to customize your repository's social media preview.
Images should be at least 640×320px (1280×640px for best display). [Download template](#)

Add Collaborators (3)

General

Access

- [Collaborators](#) (selected)
- [Moderation options](#)

Code and automation

- [Actions](#)
- [Webhooks](#)
- [Environments](#)
- [Codespaces](#)
- [Pages](#)

Security

- [Code security and analysis](#)
- [Deploy keys](#)
- [Secrets](#)

Who has access

PUBLIC REPOSITORY

This repository is public and visible to anyone.

[Manage](#)

DIRECT ACCESS

0 collaborators have access to this repository. Only you can contribute to this repository.

Manage access

You haven't invited any collaborators yet

[Add people](#)

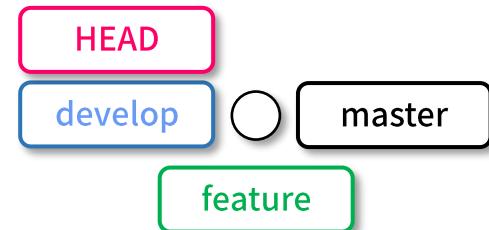
Development

- 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)

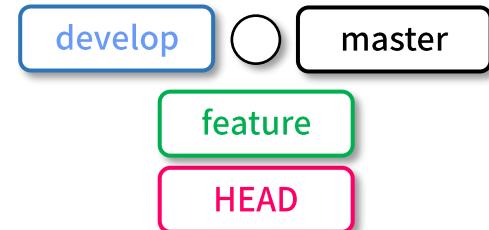
- Before making the changes, create a feature branch

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



- Change to feature branch

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



- Or use Gitflow command

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

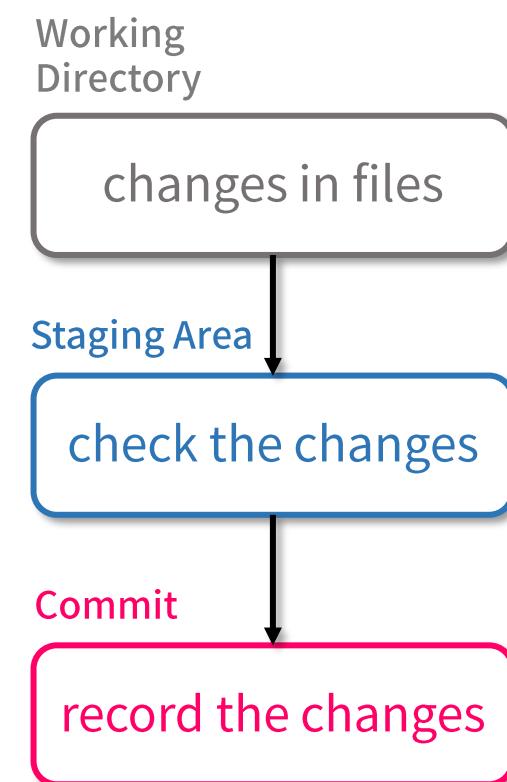
Implement Function to the Code (2)

- 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)"
```

Working
Directory

changes in files

Staging Area

check the changes

Commit

record the changes

Commit to Current Branch

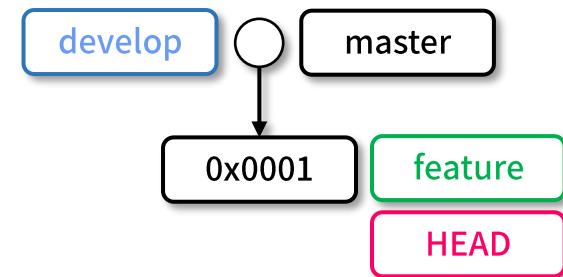
Implement Function to the Code (4)

- 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)

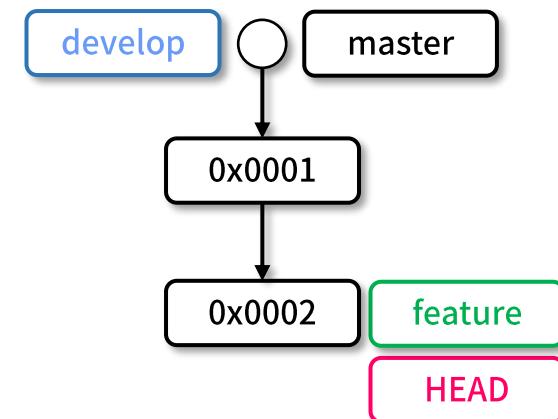
- Make changes to the code
- Put the changes to the staging area

```
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



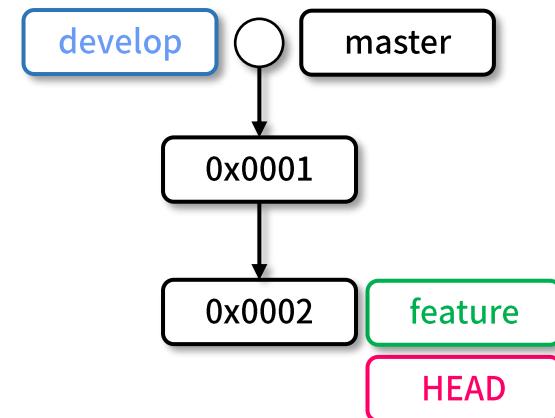
Implement Function to the Code (6)

- 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)

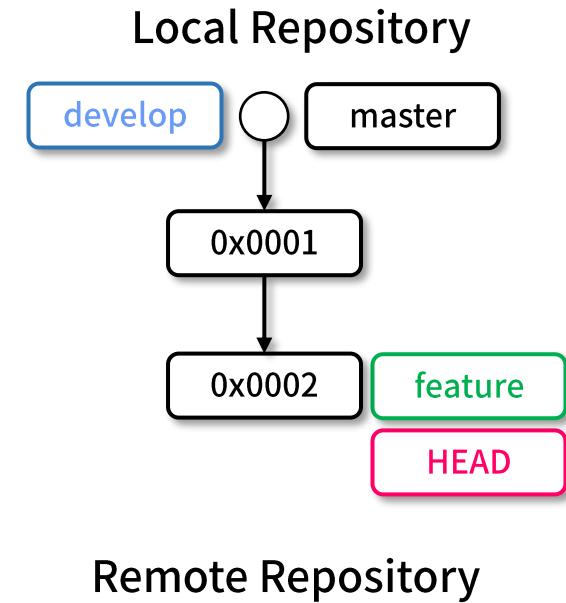
- Update the remote repository to sync with the local repository.

```
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)

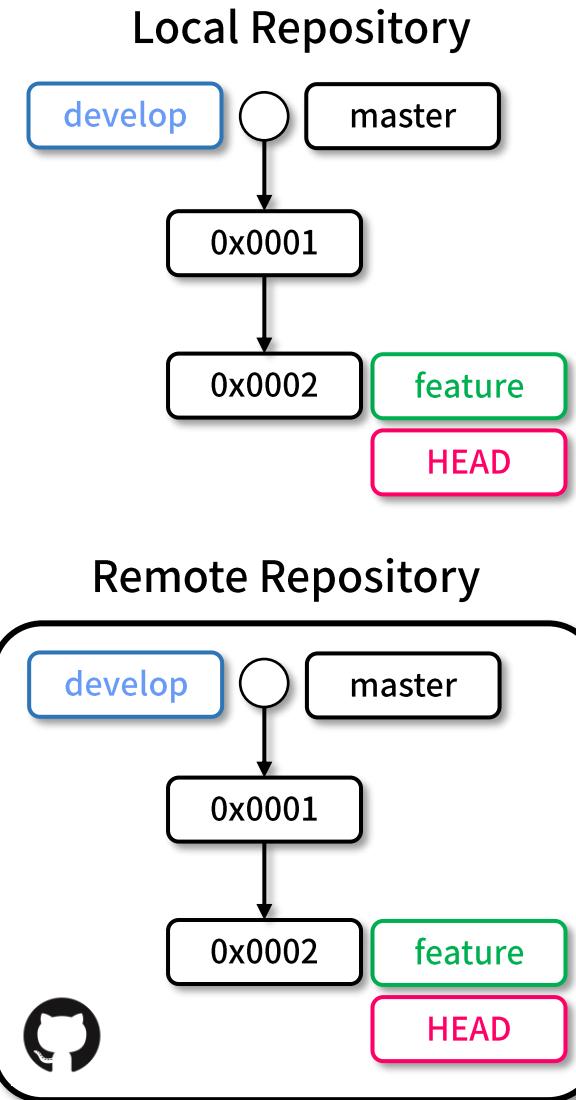
- Update the remote repository to sync with the local repository.

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

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

- Or use Gitflow command

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

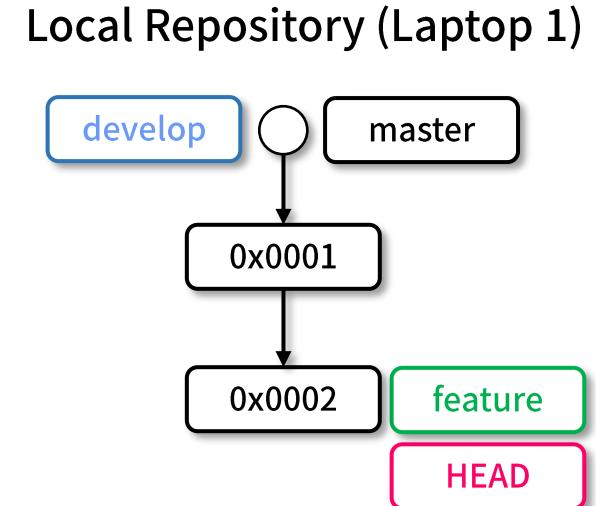
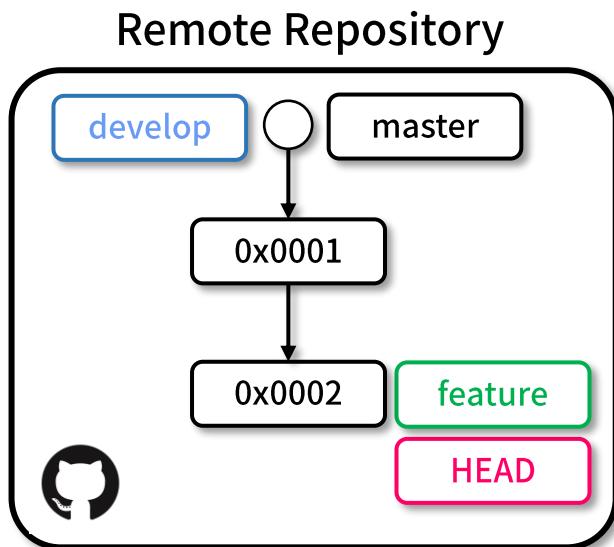


Sync with the Remote (3)

- 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

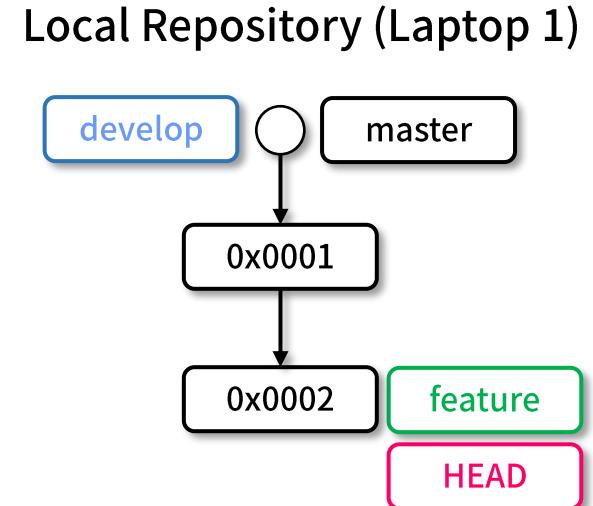
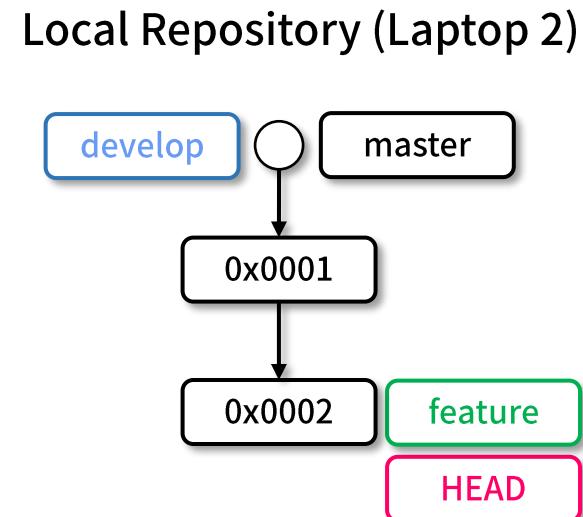
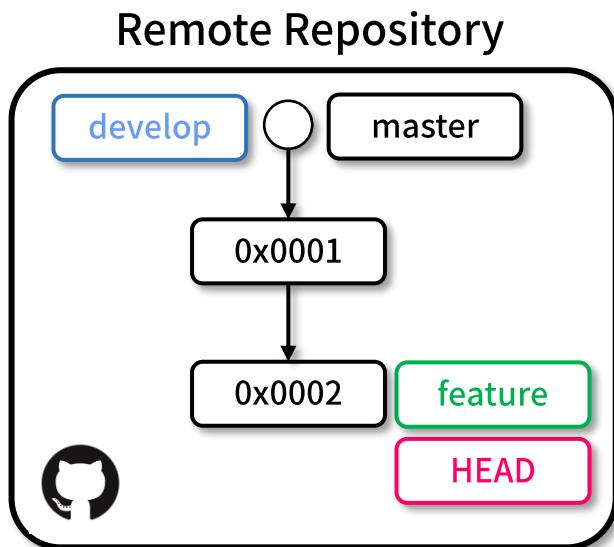


Sync with the Remote (4)

- 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

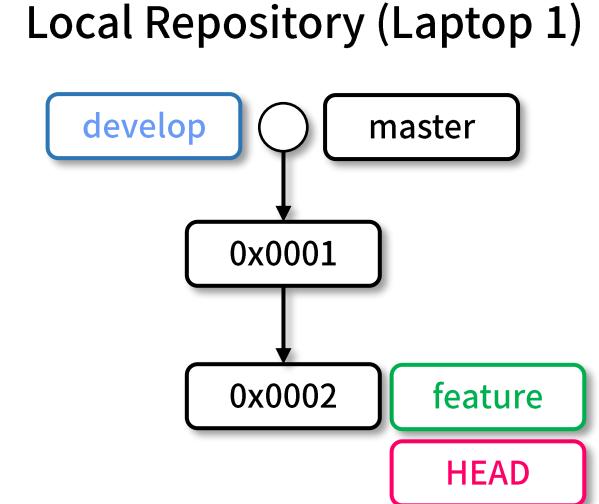
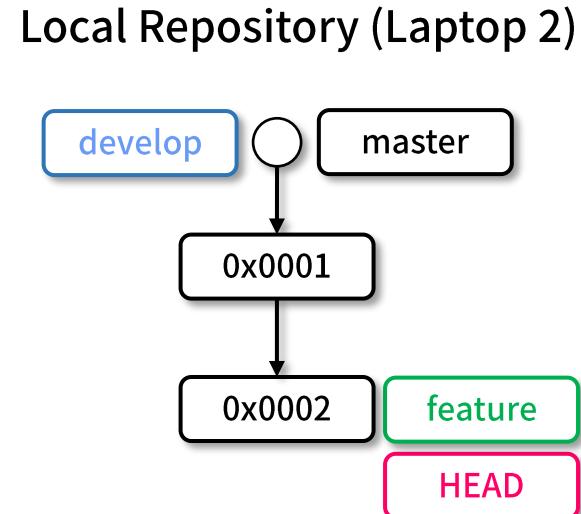
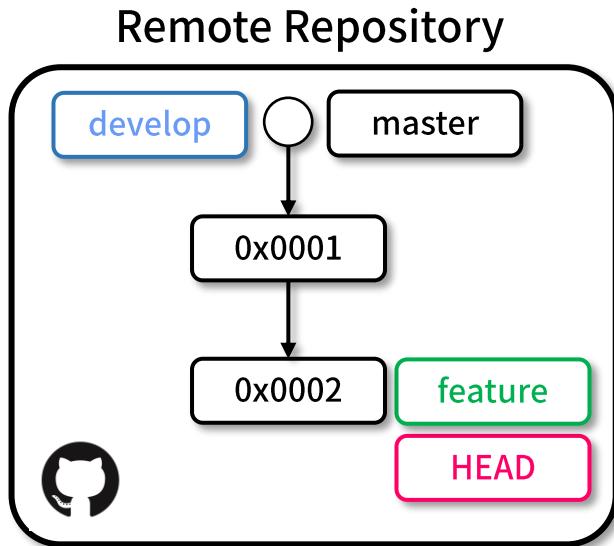


Sync with the Remote (5)

- 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



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)
```

Working
Directory

changes in files

Commit

record the changes

Commit to Current Branch

checkout

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)

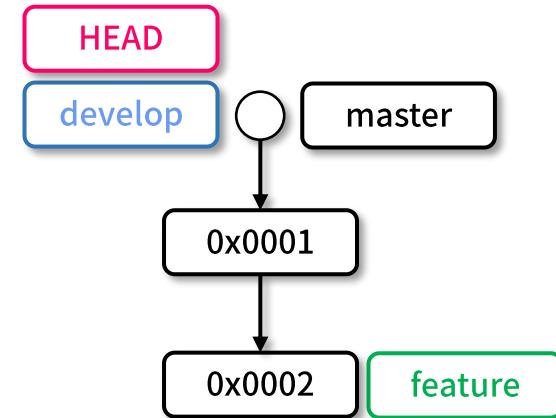
- Merge feature branch with develop branch

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

- Or use Gitflow command

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

Local Repository



Finish the Function Development (3)

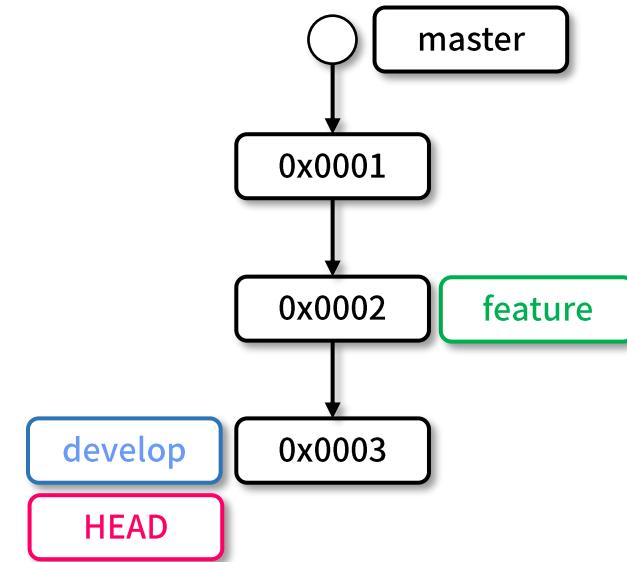
- Merge feature branch with develop branch

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

- Or use Gitflow command

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

Local Repository

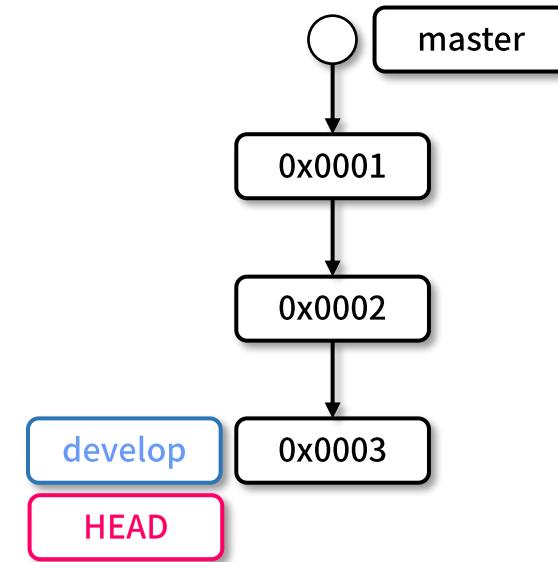


Finish the Function Development (4)

- Merge feature branch with develop branch

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

Local Repository



- Or use Gitflow command

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

Finish the Function Development (5)

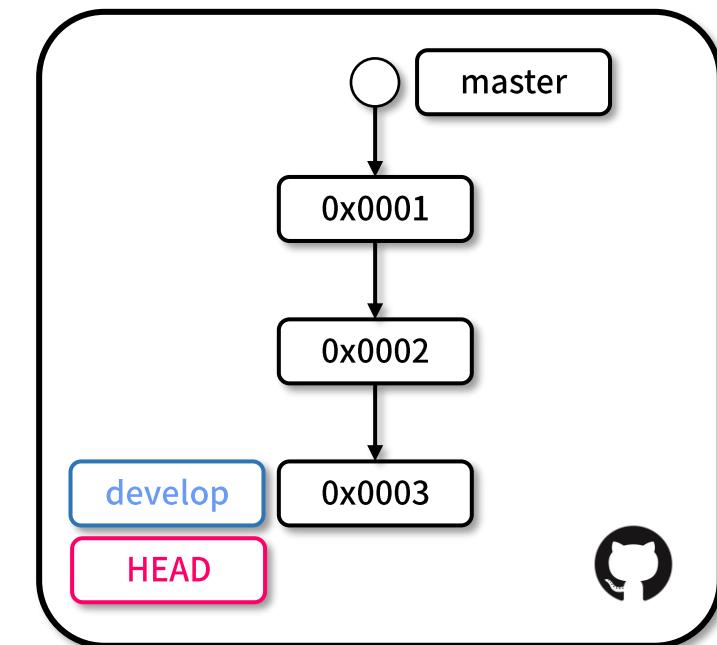
- Update the remote repository to sync with the local repository.

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

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

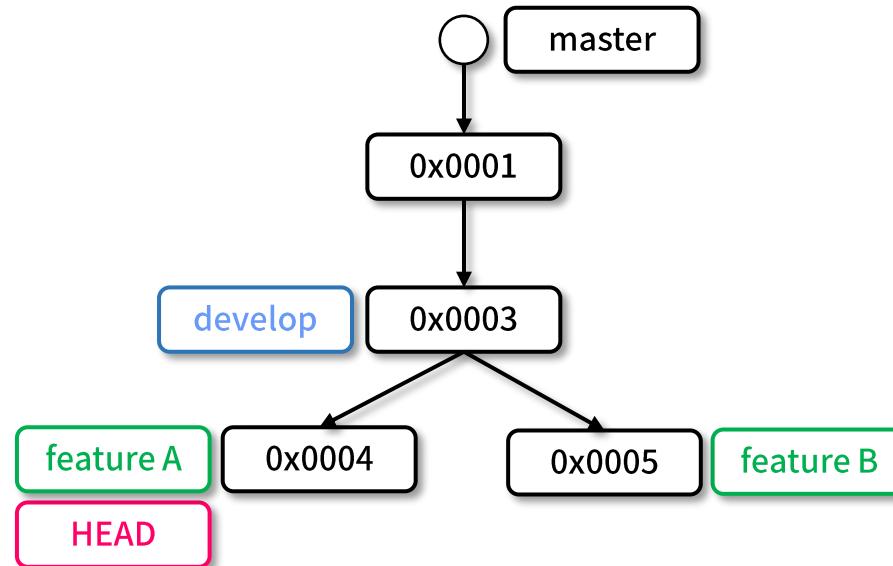
```
git checkout develop  
git pull origin develop
```

Remote Repository



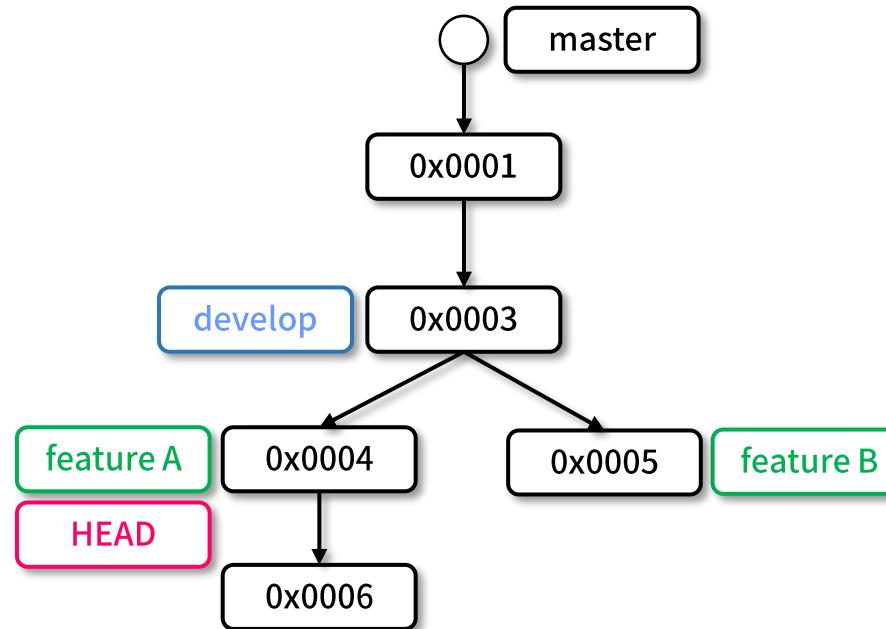
Continue the Development (1)

Developers can work on different function based on the develop branch



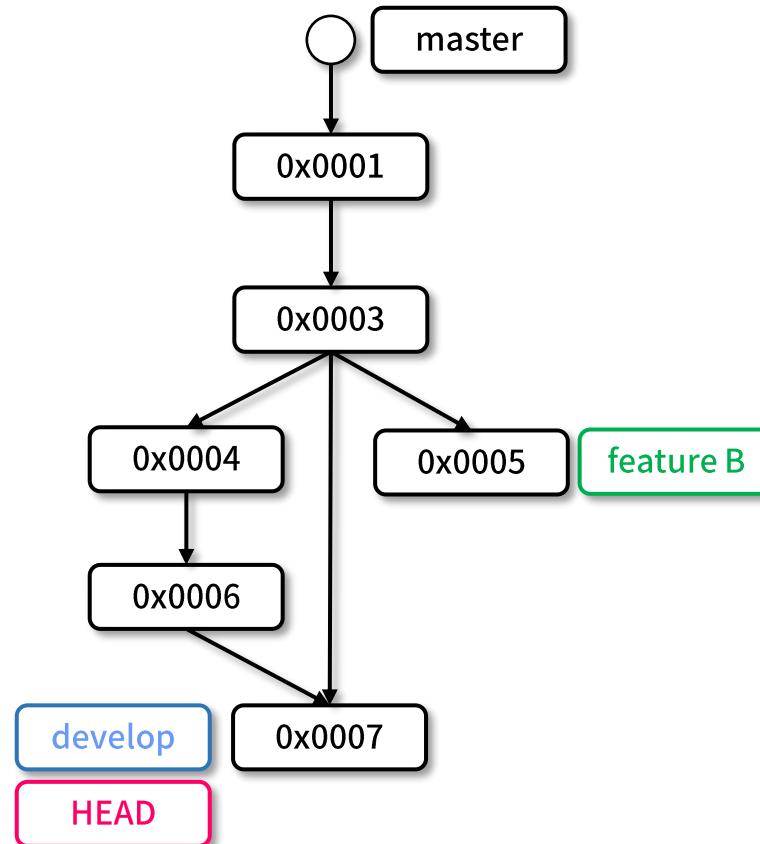
Continue the Development (2)

Developers can work on different function based on the develop branch



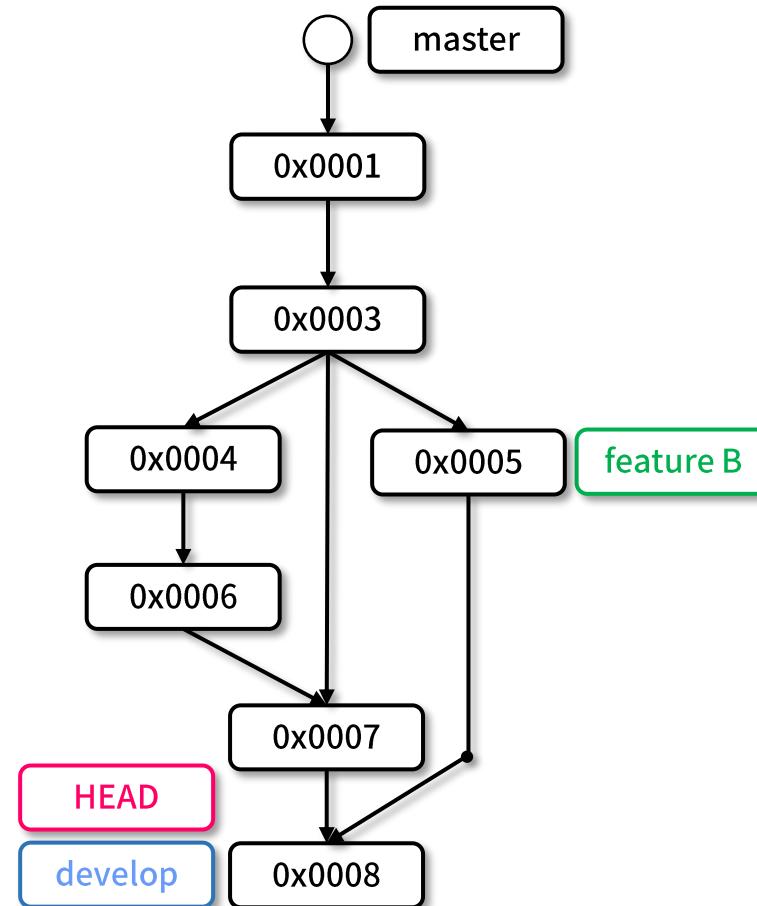
Continue the Development (3)

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



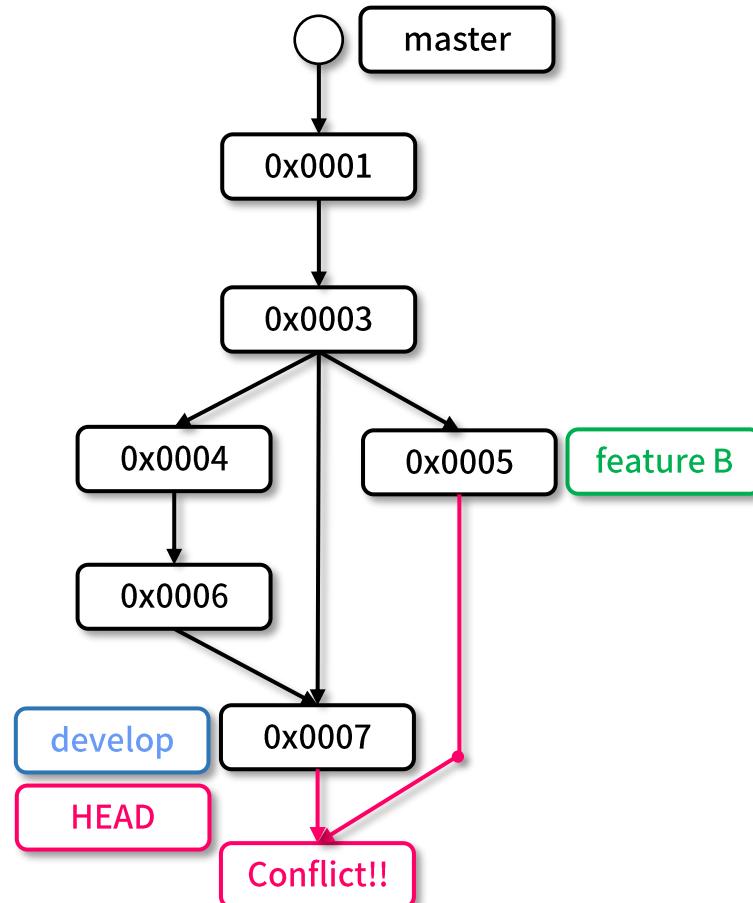
Dealing with Conflict

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



Deployment (1)

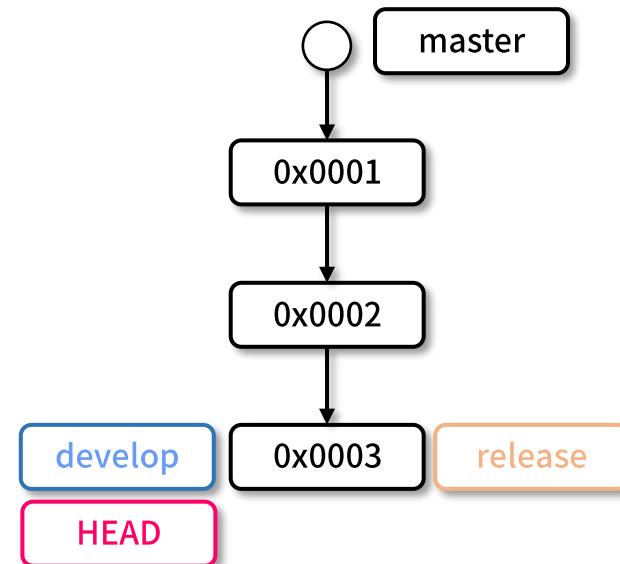
- Create a release branch

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

- Or use Gitflow command

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

Local Repository



Deployment (2)

- Merge the master branch with release branch

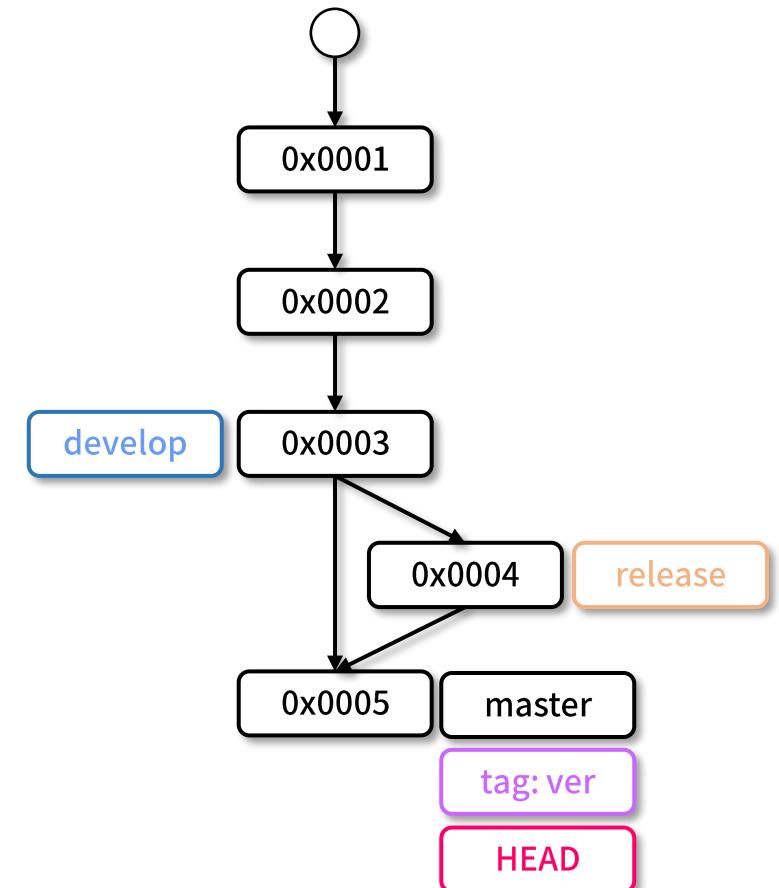
```
git checkout master  
git merge --no-ff release/(version)  
git tag -a (version)
```

* 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)
```

Local Repository



* after commit in release branch

Deployment (3)

- Merge the master branch with release branch

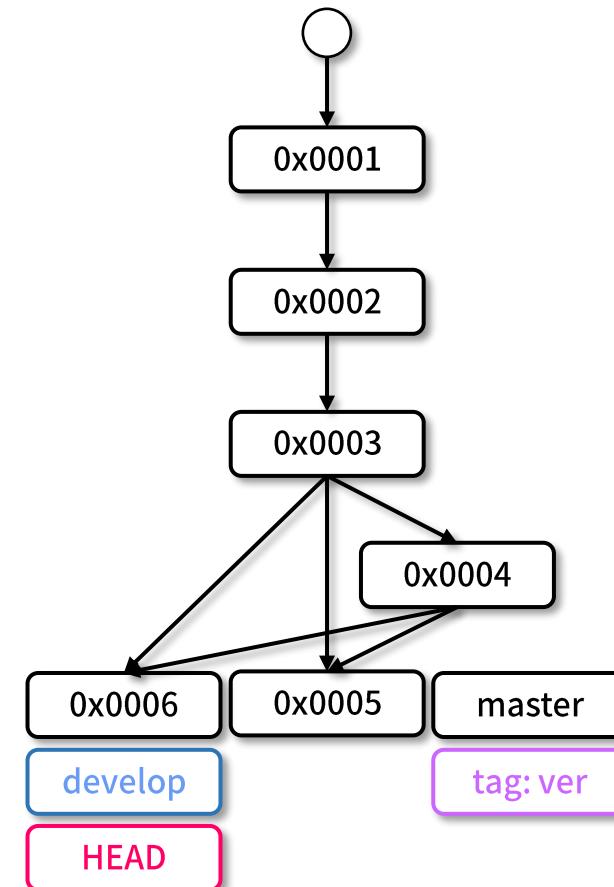
```
git checkout master  
git merge --no-ff release/(version)  
git tag -a (version)
```

* 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)
```

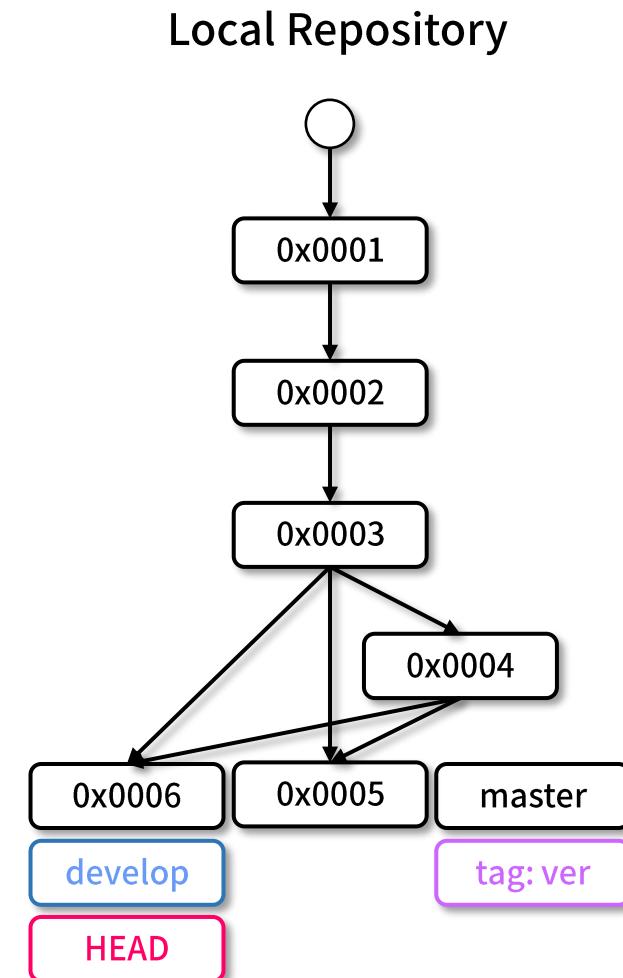
Local Repository



Deployment (4)

- Or use Gitflow command

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



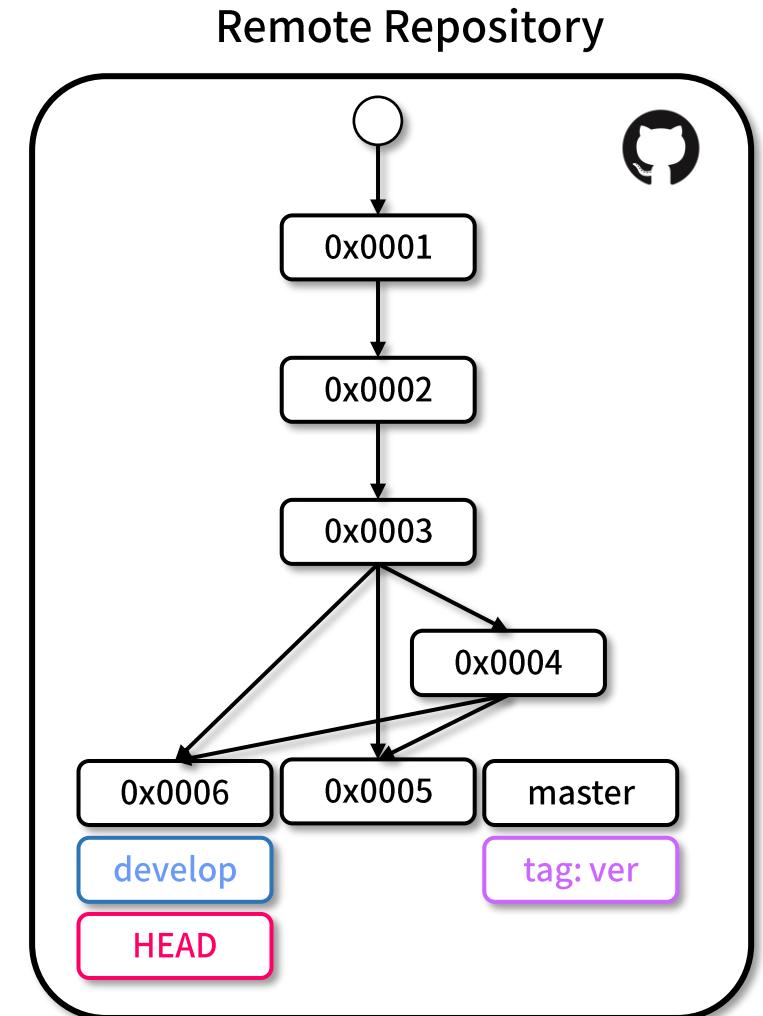
Deployment (5)

- 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
```



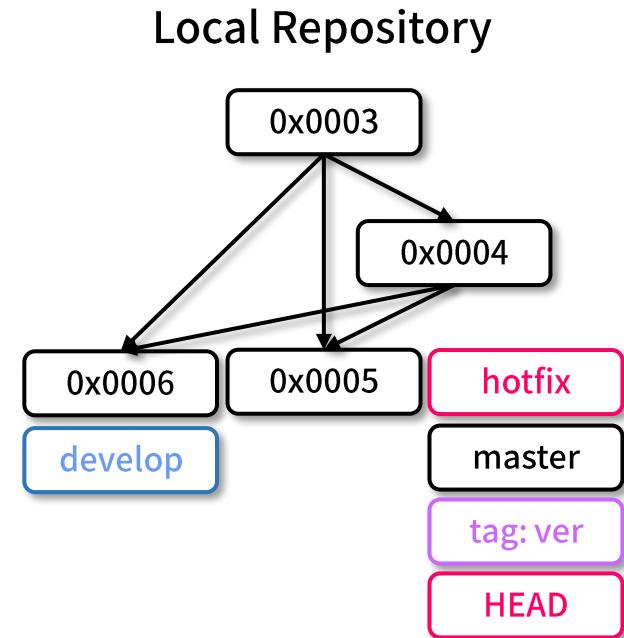
Bug Fix on the Releases (1)

- 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)
```



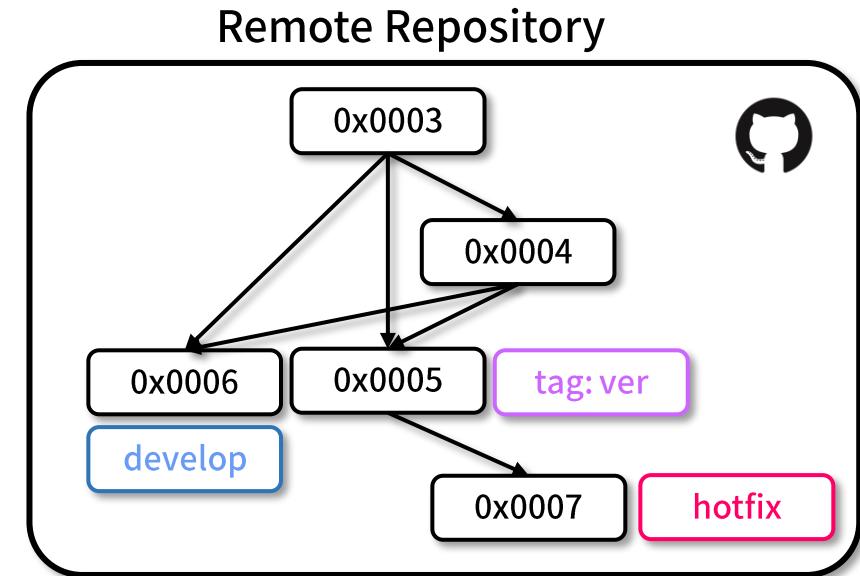
Bug Fix on the Releases (2)

- 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)
```



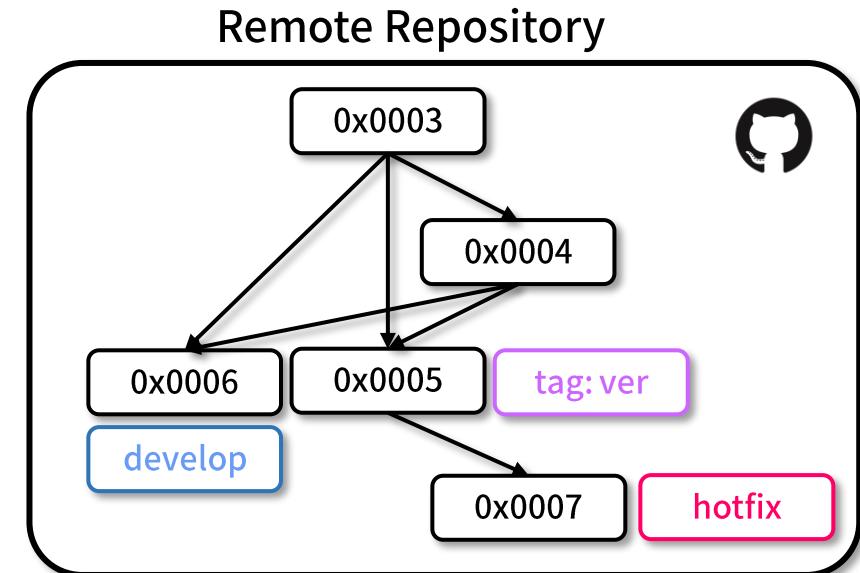
Bug Fix on the Releases (3)

- 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)
```



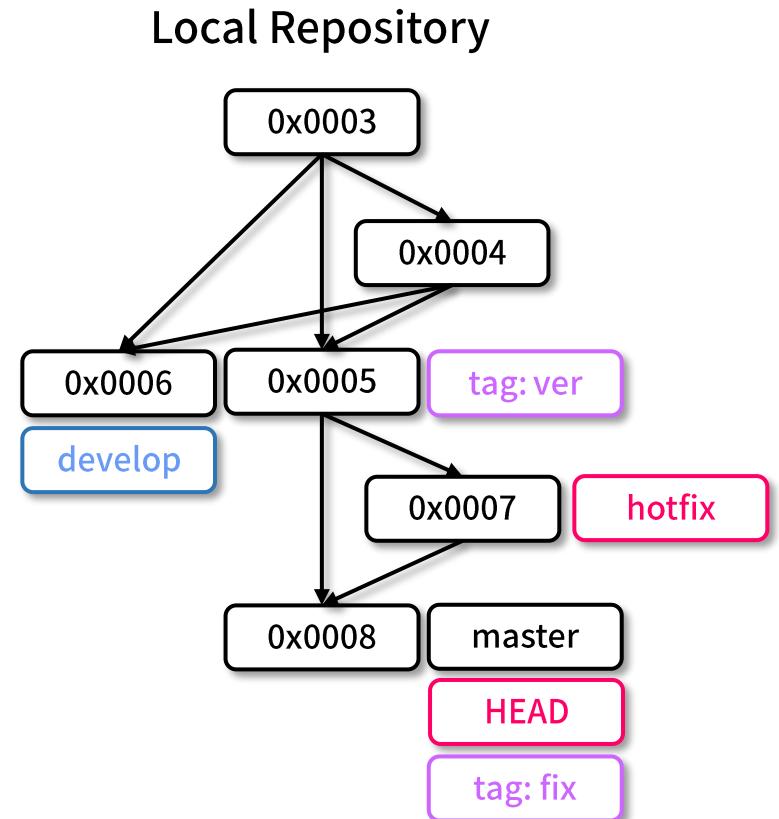
Bug Fix on the Releases (4)

- 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)
```



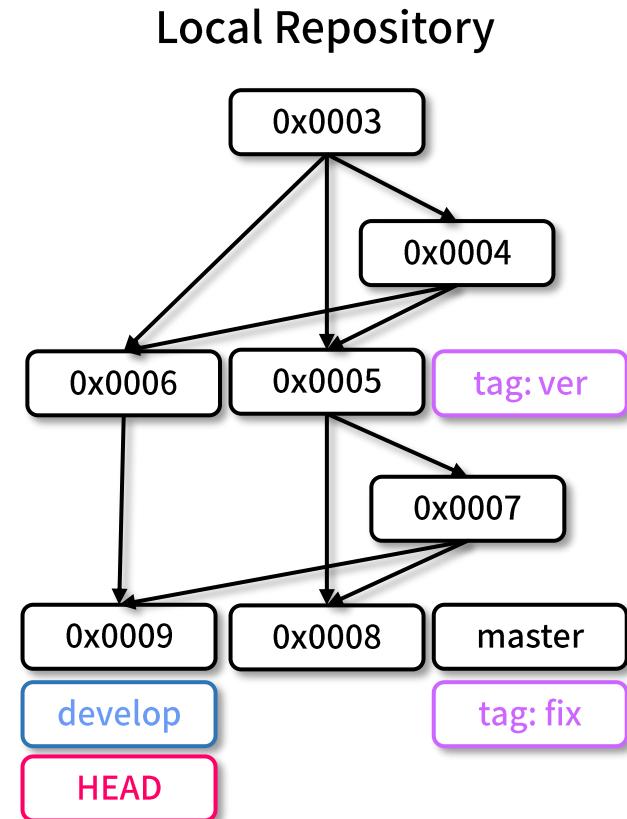
Bug Fix on the Releases (5)

- 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

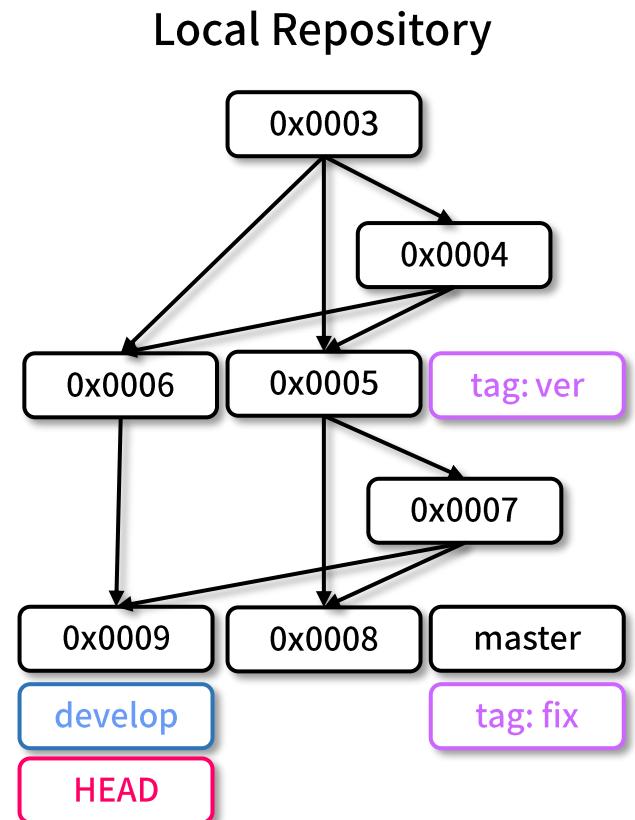
```
git checkout develop  
git merge --no-ff hotfix/(version)  
git branch -d hotfix/(version)
```



Bug Fix on the Releases (6)

- Or use Gitflow command

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



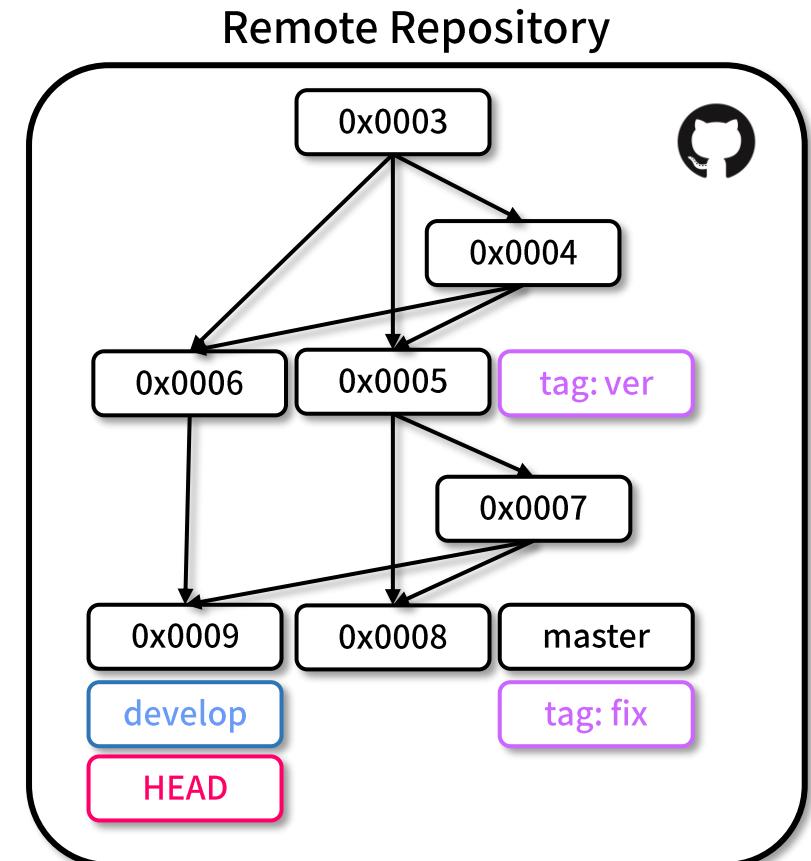
Bug Fix on the Releases (7)

- 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.iscb.org



rsg-norway.iscb.org



Feedback



Join Slack



@RSGNorway



rsg-norway@iscb.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 committed 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]
```

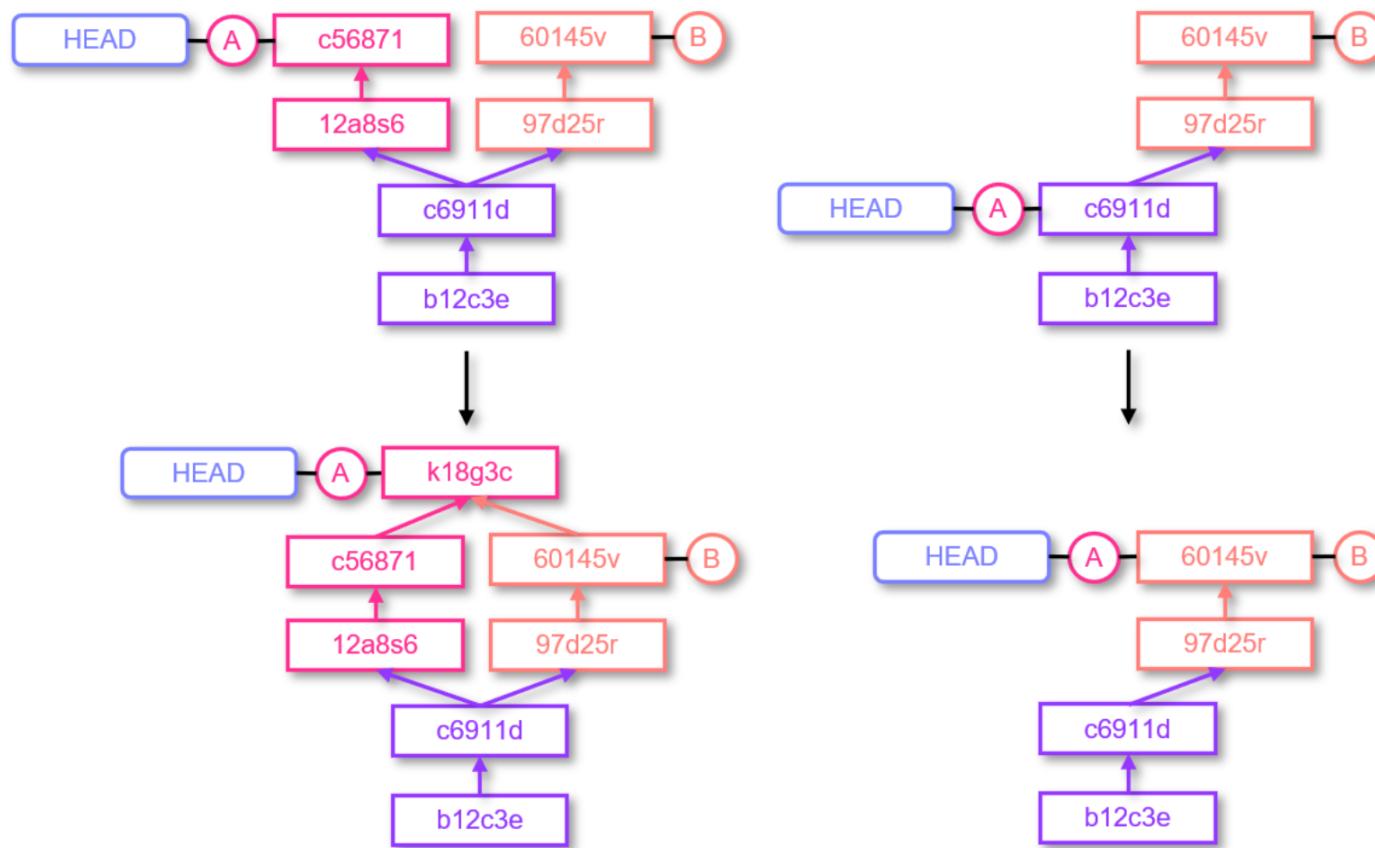
- all the committed changes will be **removed**
- all the staged changes will be **removed**
- all the local changes will be **removed**

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

- all the committed 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**

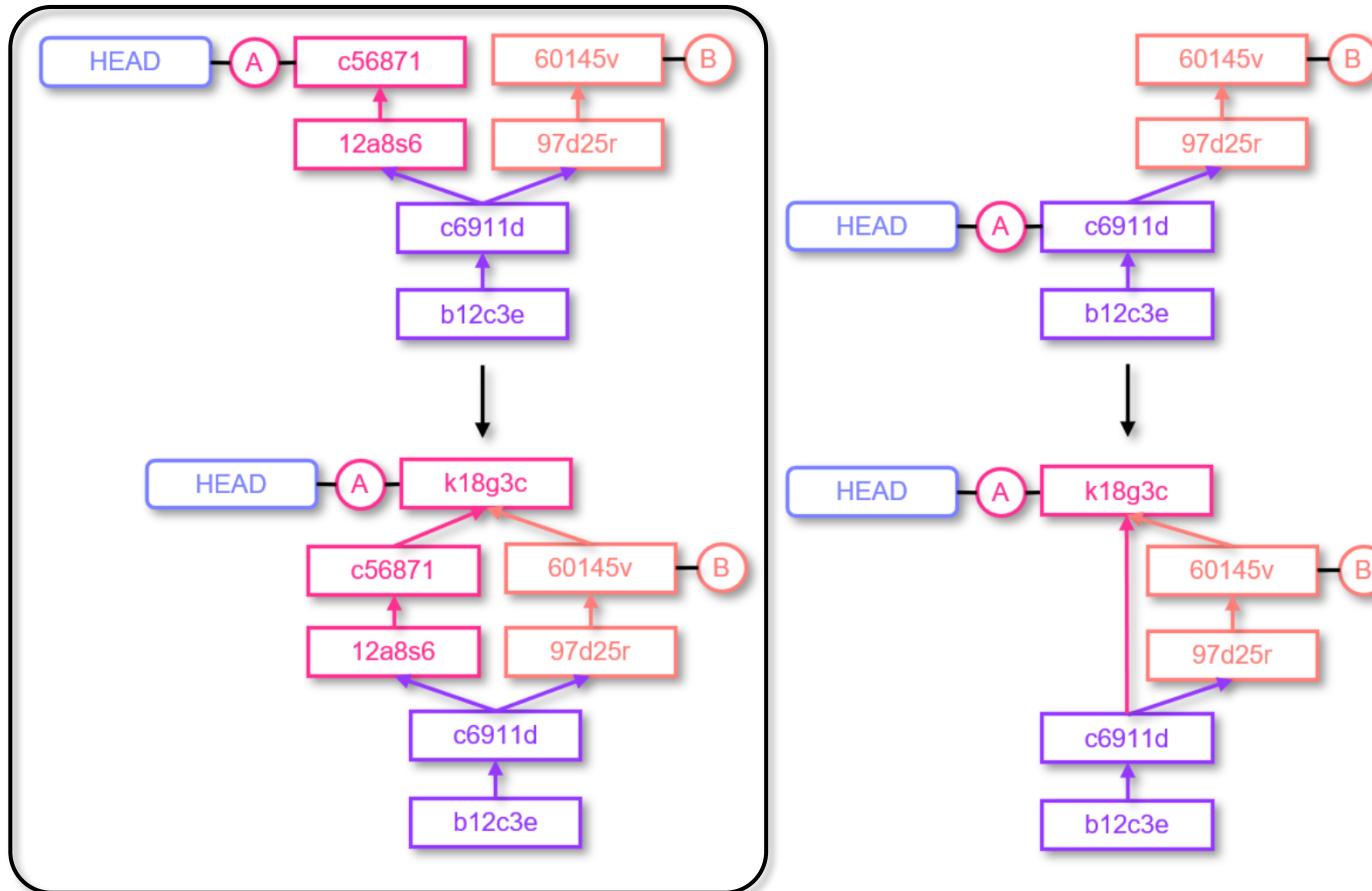
Appendix – Merge (1)

with fast-forward



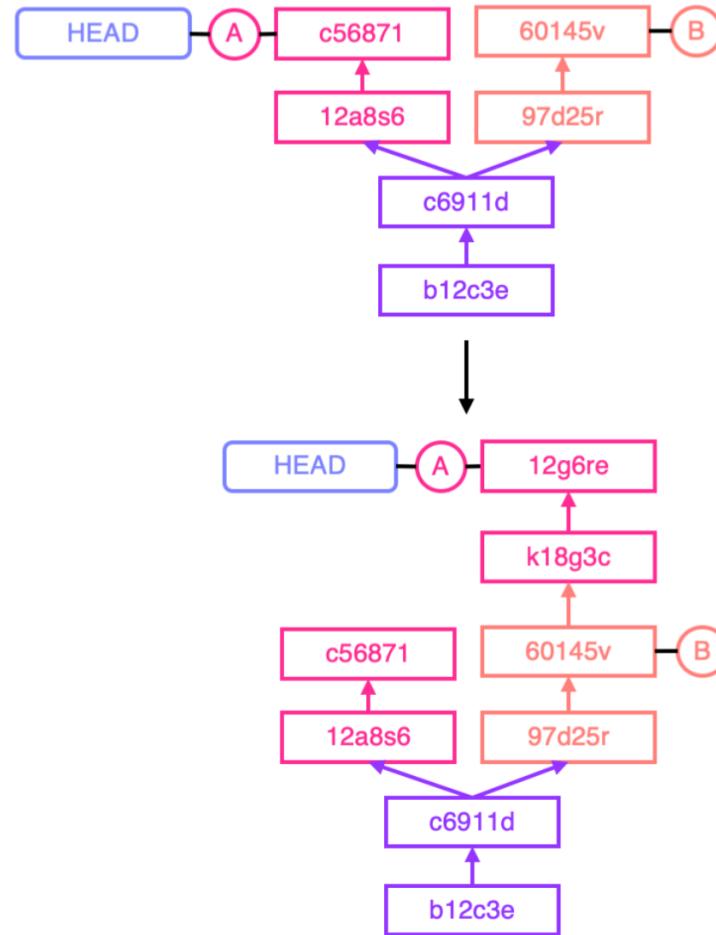
Appendix – Merge (2)

without fast-forward



same as fast-forward

Appendix – Rebase



Appendix – Useful References

- [Atlassian Bitbucket Git Tutorial](#)
- [Git Cheatsheet](#)
- [Gitflow Cheatsheet](#)