Git Overview

What is Git?

> Git is a distributed version control system which means each user has a local copy of the remote repository and can easily be synchronized

What is a Git Repository?

- A series of **snapshots** or **commits**

Getting Started

Configuring User Information

- Use the `git config [--local | --global | --system] <key> [<value>]` command to specify or read configuration information if you don't use the `[value]` option

```
$ git config --global user.email "yuxiangwang0829@gmail.com" # set
$ git config --global user.email # get
yuxiangwang0829@gmail.com
```

Configuring the Default Editor

- Use the `git config --global core.editor [<editorname>]` command to specify or read your preferred Git editor if you don't use the `[editorname]` option

```
$ git config --global core.editor vim # set
$ git config --global core.editor # get
vim
```

Git Locations

Project Directory

Working Tree

- A single commit's directories and files
- You can view and edit the files of project, preparing for the next commit

Staging Area/Index

- Files that are planned for the next commit

Local Repository

- Contains all of the commits that have been made for the project

.git Directory

- A hidden .git directory which contains the staging area and local repository

Remote Repository

- Contains the commits of the project

Create a Local Repository

- `git init` -- initialize (create) an empty repository

Commit to a Local Repository

- Use `git status` to view the status of files in the working tree and staging area
- Use `git add` to add content to the staging area

- Use the `git commit [-m]` command to add staged content to the local repository as a commit, you can use the `-m` flag to specify a short commit message, if you don't use the `-m` option, and then Git will open the default editor to let you write the commit message
- Use `git log [--oneline]` to view the local repository's commit history

Push to a Remote Repository

`git clone` vs. `git remote add`

If you want to begin working with the remote repository, you have two main options:

- If have a local repository, you want to **push** to a remote repository, than you will **add** the remote repository to your local repository

- If not have a local repository, you will **clone** the remote repository which will create a local repository that is associated with the remote repository

Clone a Remote Repository

> A clone is a local copy of a remote repository
>
> A reference to the remote repository is included in the local repository, so you can synchronize the repository
>
> **origin** is the default alias for the remote repository URL

How to Clone

- `git clone <url/to/projectname.git> [localprojectname]` is used to create a local copy of a remote repository, the last argument (which defines the name of the directory name where the local repository is located) is optional, cloning will create a project directory in your local computer

What Does Clone Do

- After cloning, all the commits on the remote repository will also in your local repository

Note

- `git remote [--verbose]` displays information about remote repositories associated with the local repository

Add a Remote Repository to a Local Repository

- `git remote add <name> <url>` command will add information about the remote repository to the local repository, the two repositories can then be synchronized using aliases instead of URL

Push Commits to a Remote Repository

> All commits belong to a branch
>
> A branch can be thought of as an independent line of development of the project

Default Branch

- By default, there is a single branch and it is called **master**

How to Push

- `git push [-u] [<repository>] [<branch>]` writes commits for a branch from the local repository to a remote repository
  - `<repository>` can be a name(shortcut which is often origin) or URL
  - `-u` flag is set to set up a tracking relationship between the local and remote branch
  - `[<branch>]` is the branch in the remote repository that you want to push to
  - The arguments after `git push` are all optional, because git will assume default information or use previous arguments after you have executed the first push

What Does Push Do

- A successful push synchronizes the branches on the local and remote repositories so that they they contain exactly the same commits

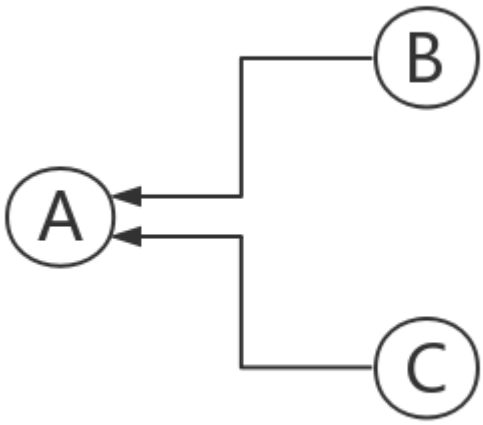
Note

- Before you run `git push` command you can use `git remote --verbose` command to ensure that your local repository has an association with the remote repository

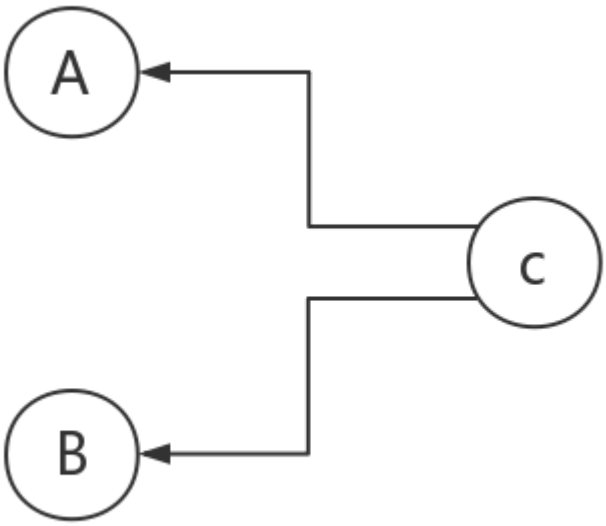
---

Git's Graph Model

- **Git models the relationship of commits with directed acyclic graph**
- **The arrows point at a commit's parent(s)**
- **The commits and the relationship between them is what forms the project's history**
- **A branch occurs if a commit has more than one child**



- **A merge occurs when a commit has more than one parent**



- `git log [--graph]` will show you a diagram

Git IDs

Git Objects

1. **Commit object** - A small text file
2. **Annotated tag** - A reference to a specific commit
3. Tree - Directories and filenames in the project
4. Blob - The content of a file in the project

Git ID

- The name of a Git Object
- 40 - character hexadecimal string also known as object ID, SHA-1, hash, and checksum

Shortening Git IDs

- The first portion of the Git ID

References

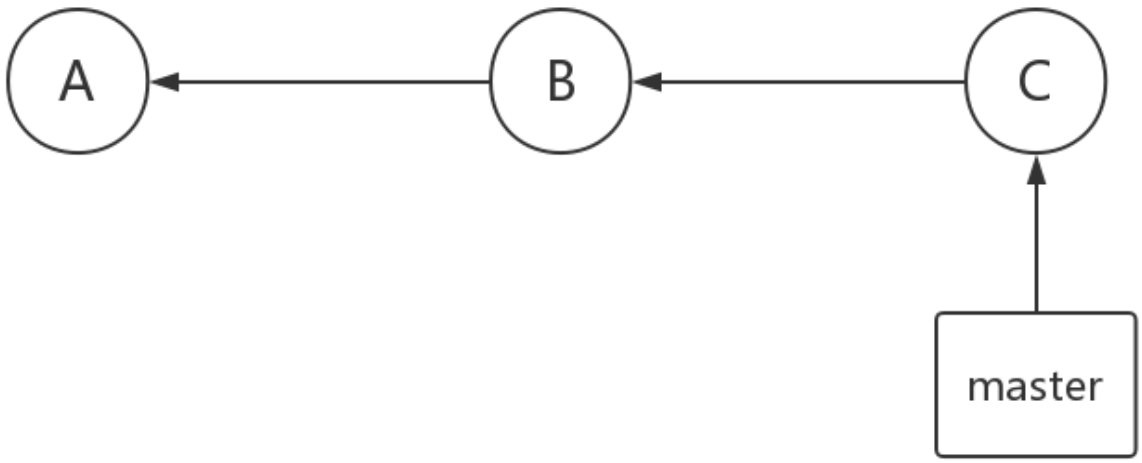
Overview of References

- Commits objects can be associated with references
- A reference is a user-friendly name that points to:
  - a commit SHA-1 hash
  - another reference(known as a symbolic reference) (such as HEAD)
- Use references instead of SHA-1 hashes, `git show` command can show the specified commit message, such as use `git show HEAD`
- References are stored in the .git directory

Branch Label and HEAD Reference

Branch Label References

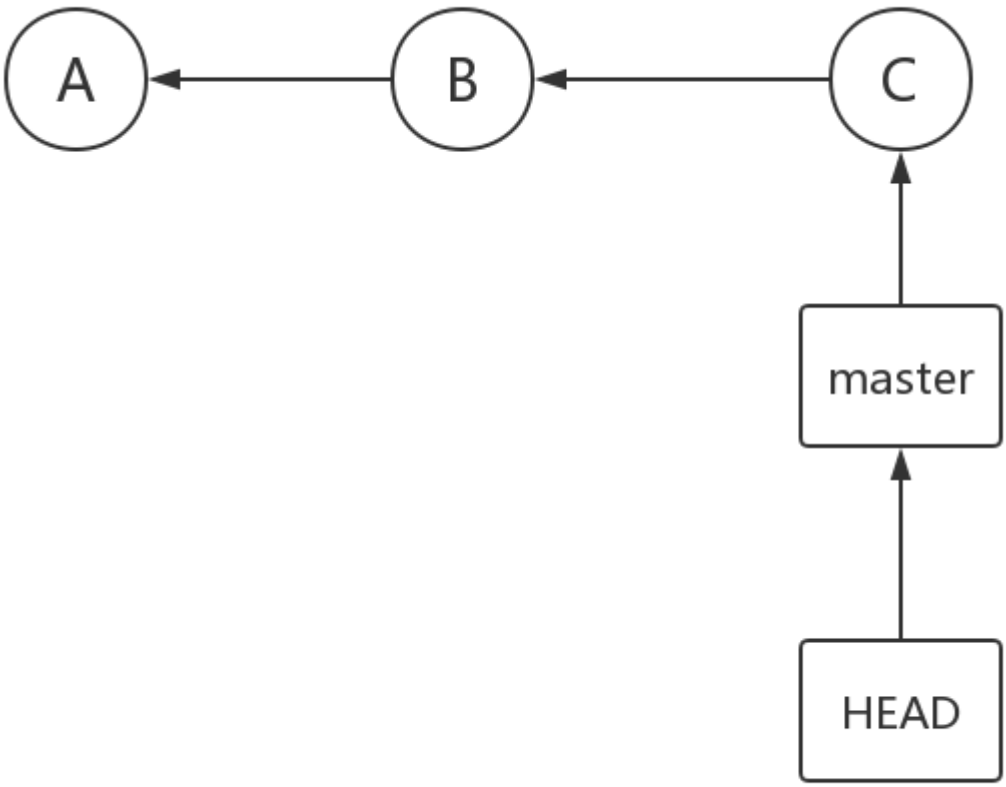
- **master** is the default name of the main branch in the repository
- **A branch label** is a reference points to the most recent commit object in the branch



- **Branch labels** are implemented as references
- The difference between a branch and a branch label:
  - All commits belong to the branch
  - The branch label is only at the latest commit of the branch

HEAD Reference

- **HEAD** is a reference to the current commit object
- **HEAD** usually points to the branch label reference of the current branch

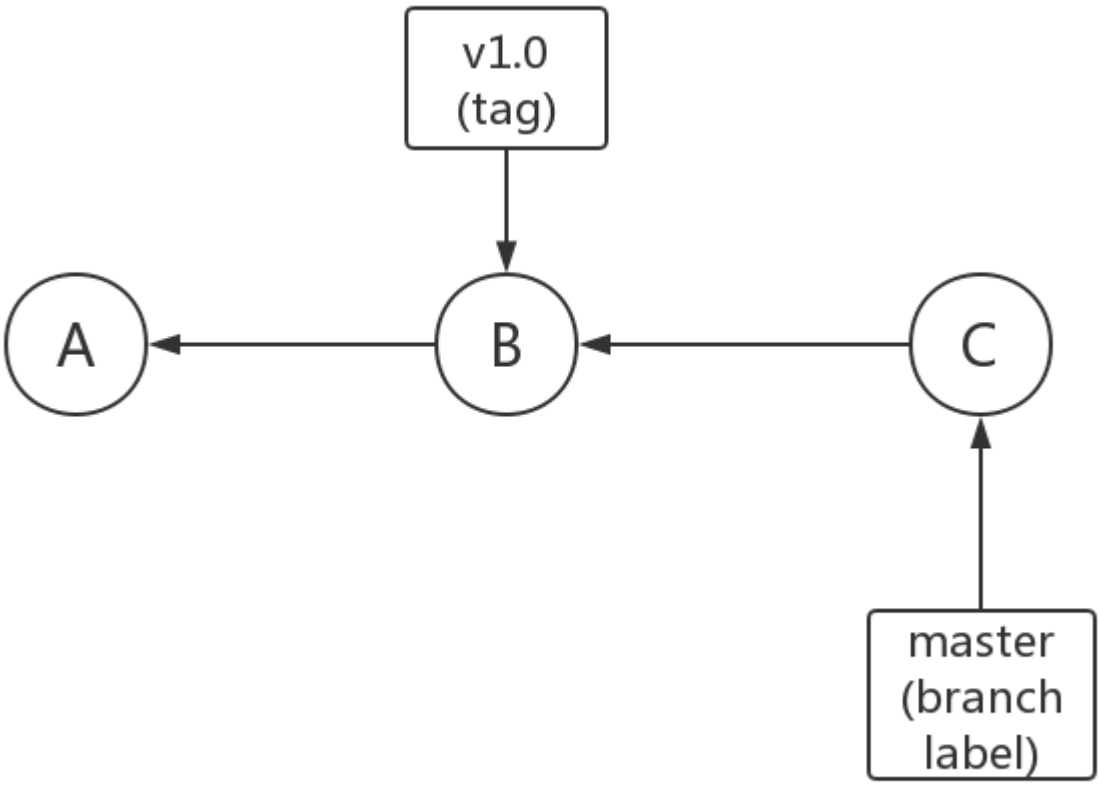


- Only one **HEAD** per repository

Tag Reference

Tag Overview

- A **tag** is a reference points to a specified commit object, it acts a user-friendly label for the commit



- There are two types of tags:
  - Lightweight(A simple reference points to a commit object much like branch label or HEAD)
  - Annotated(recommended)
    - A full Git object that references a commit
    - Includes tag author information, tag date, tag message, the commit ID

Create and View tags

Commands

- `git tag` - View all tags in the repository
- Tags can be used instead of branch labels or Git IDs in Git commands such as `git show v1.0`
- To tag a commit with a lightweight tag:
  - `git tag <tagname> [<commit>]`
  - `<commit>` defaults to HEAD
- To tag a commit with an annotated tag:
  - `git tag -a [-m <msg> | -F <file>] <tagname> [<commit>]`
  - `commit` default to HEAD

Examples

```
$ git init
Initialized empty Git repository in /.../.../.git/
$ git log
fatal: your current branch 'master' does not have any commits yet
$ touch file1
$ git status
On branch master

No commits yet

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

        file1

nothing added to commit but untracked files present (use "git add" to track)
$ git add .
$ git commit -m "file1"
[master (root-commit) e37cbd7] file1
 1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file1
$ git log --oneline
e37cbd7 (HEAD -> master) file1
$ git tag v1.0
$ git log --oneline
e37cbd7 (HEAD -> master, tag: v1.0) file1
$ touch file2
$ git add .
$ git commit -m "file2"
$ git log --oneline
18e1299 (HEAD -> master) file2
e37cbd7 (tag: v1.0) file1
$ git tag -a -m "this is an annotated tag" v2.0
$ git log --oneline
18e1299 (HEAD -> master, tag: v2.0) file2
e37cbd7 (tag: v1.0) file1
$ git tag
v1.0
v2.0
```

Notice

- `git push` does not automatically transfer tags to the remote repository
- To transfer a single tag: `git push [<remote>] <tagname>`
- To transfer all of your tags: `git push [<remote>] --tags`

Branches

Branch Overview

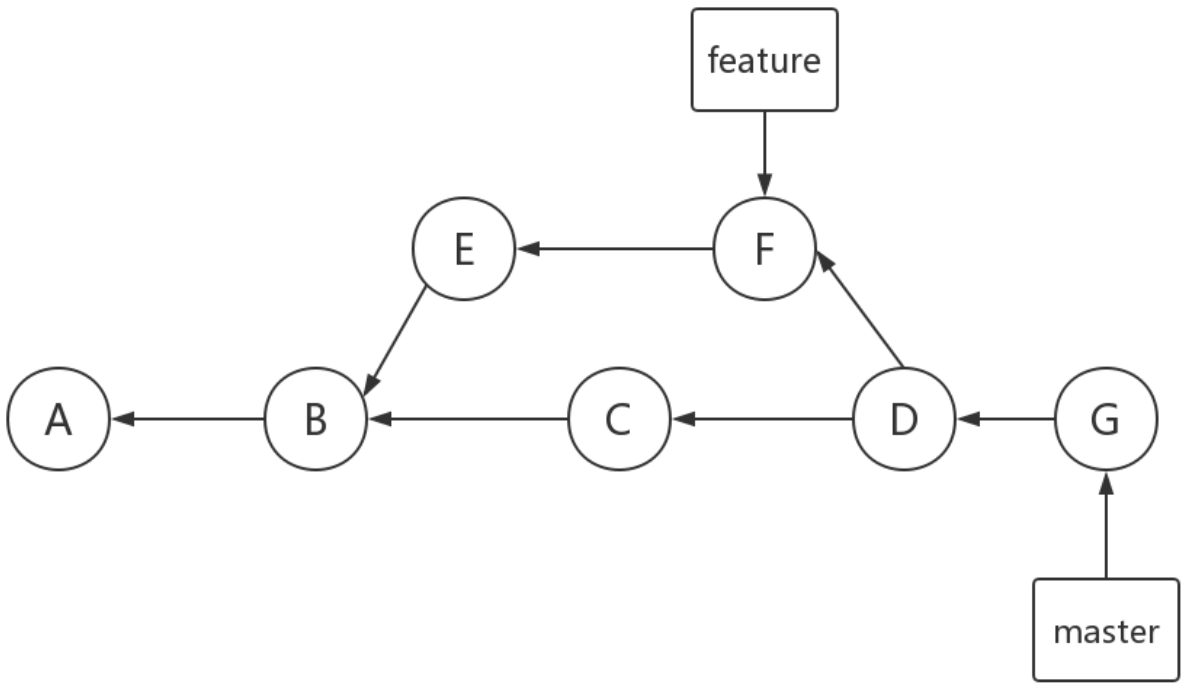
> A branch is a set of commits starting with the most recent commit in the branch and tracing back to the project's first commit -- root commit

Features

1. Fast and easy to create
2. Enable experimentation
3. Enable team development
4. Support multiple project versions

Type

1. **Short-lived** (commonly called topic or feature branches) usually contain one small change to the project



2. **Long-lived** such as master, develop, release

Look Over

- Use `git branch` to see a list of branches in a repository

Creating a branch

Commands

- Use `git branch <name>` to create a branch

Note

- Creating a branch simply creates a new branch label reference but you remain still on the original branch

Checkout

What Checkout Will Do

- Update the **HEAD** reference
- Update the **working tree** with the commit's files

How To Checkout

- Use `git checkout <branch_or_commit>` to checkout a **branch** or **commit**
- Use `git checkout -b <branchname>` to combine creating and checking out a branch into a single command

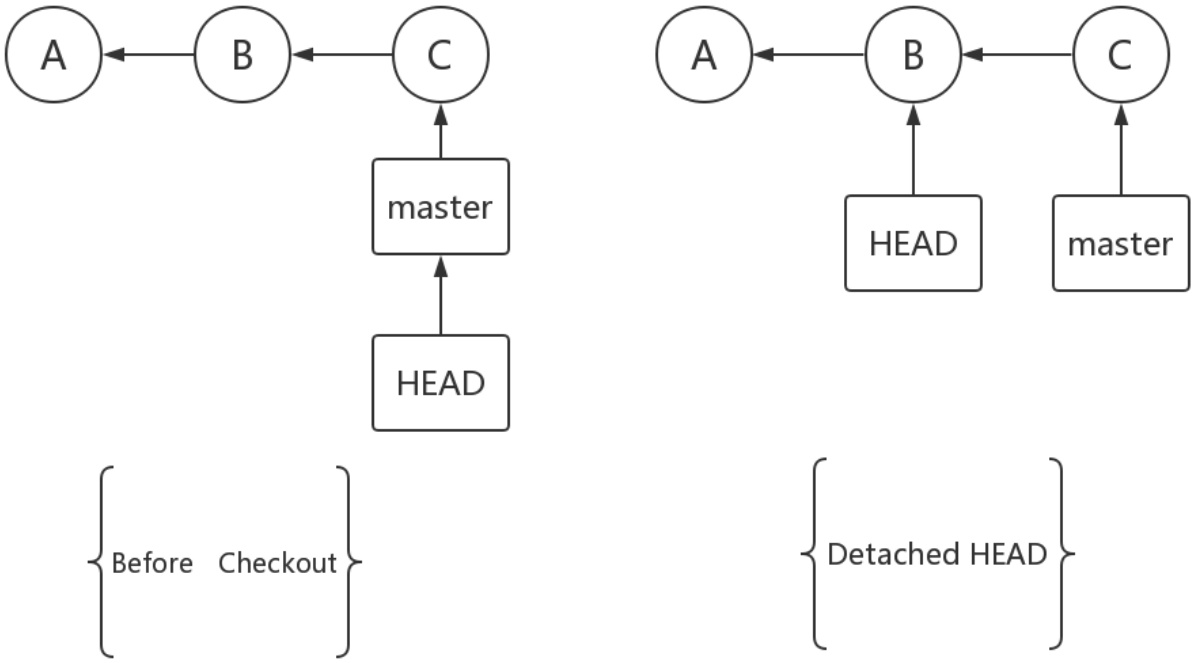
Detached HEAD

- Checking out a **commit** rather than a **branch label** leads to a detached HEAD state

```
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>
```



Deleting a Branch Label

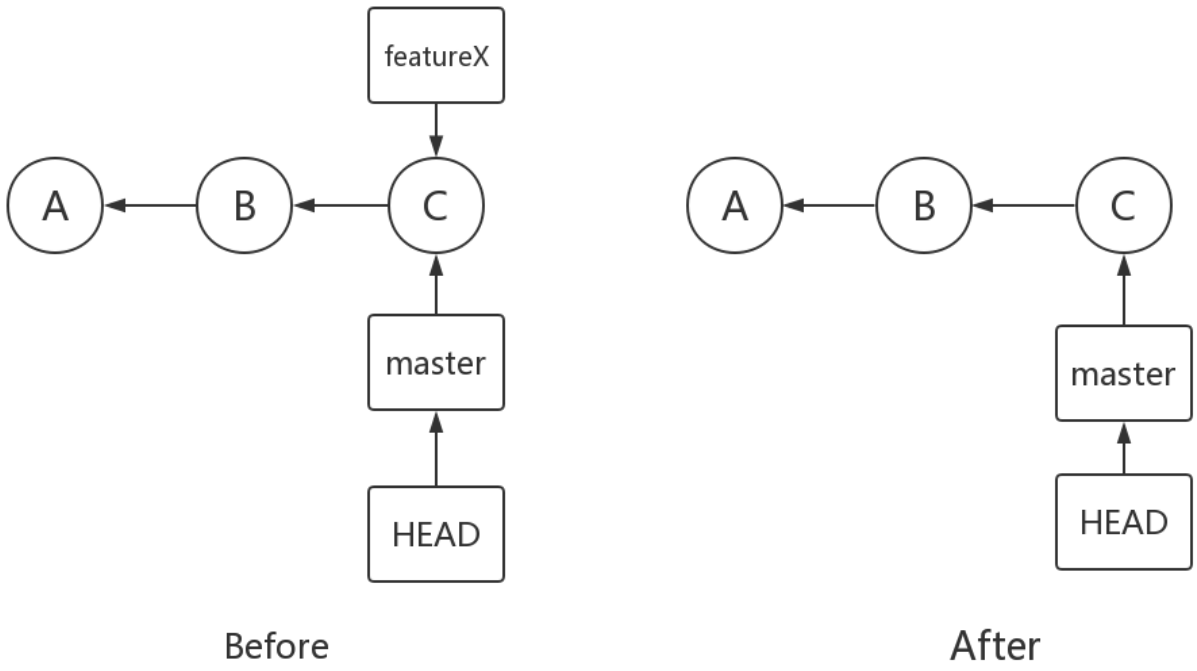
How to Delete

- Use `git branch -d <branchname>` to delete a branch

```
$ git log --oneline
3388bbb (HEAD -> master, featureX) C
47efad3 B
7811736 A
$ git checkout featureX
Switched to branch 'featureX'
$ git log --oneline
3388bbb (HEAD -> featureX, master) C
47efad3 B
7811736 A
$ git branch -d featureX
error: Cannot delete branch 'featureX' checked out at '...'
$ git checkout master
Switched to branch 'master'
$ git branch -d featureX
Deleted branch featureX (was 3388bbb).
$ git log --oneline
3388bbb (HEAD -> master) C
47efad3 B
```

```
7811736 A
```

> Deleting a branch really just means that you're deleting a branch label

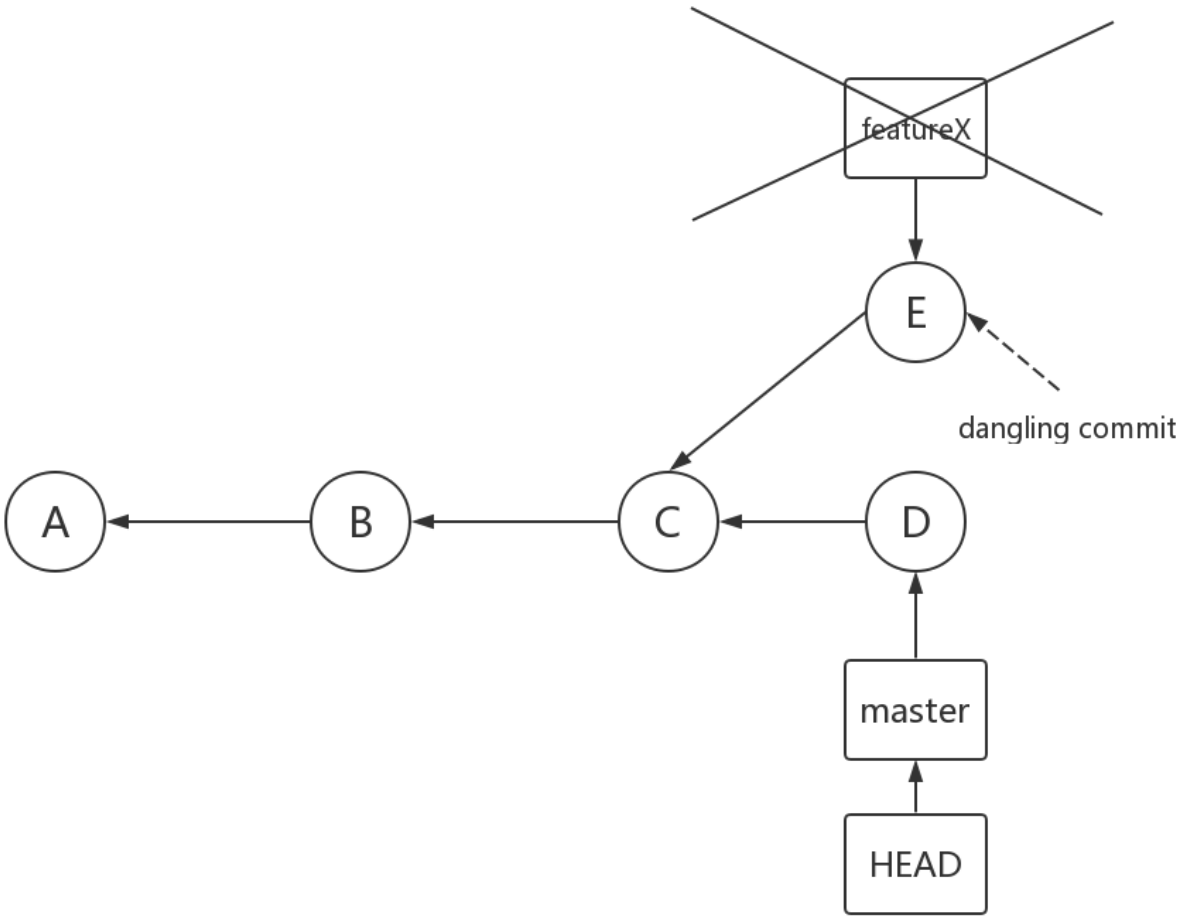


Deleting a not Merged Branch is Forbidden

- **If you are trying to delete a not fully merged branch, Git will not let you to do that**

```
$ git log --oneline
3db9474 (HEAD -> master) E
3388bbb C
47efad3 B
7811736 A
$ git checkout featureX
Switched to branch 'featureX'
$ git log --oneline
4a4439d (HEAD -> featureX) D
3388bbb C
47efad3 B
7811736 A
$ git checkout master
Switched to branch 'master'
$ git branch -d featureX
error: The branch 'featureX' is not fully merged.
If you are sure you want to delete it, run 'git branch -D featureX'.
```

- If you are sure you want to delete the branch, use the command `git branch -D <branchname>` but that will lead to the **dangling commits**




```
$ git branch -D featureX
Deleted branch featureX (was 4a4439d).
```

> Git will periodically garbage collect looking for and deleting older dangling commits, So be careful if you use the D option

- You can use `git reflog` to return a local list of recent HEAD commits

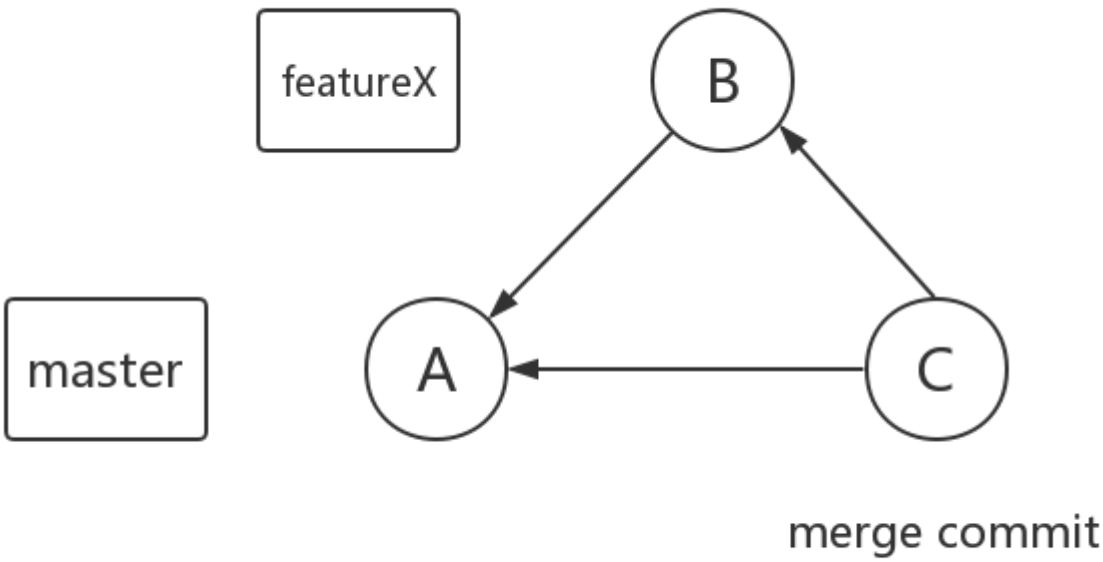
```
$ git reflog
3db9474 (HEAD -> master) HEAD@{0}: checkout: moving from featureX to master
4a4439d HEAD@{1}: commit: D
3388bbb HEAD@{2}: checkout: moving from master to featureX
...
$ git checkout -b featureX 4a4439d
# Creating a new branch label pointing to our previously dangling commit and now are back in bussiness
Switched to a new branch 'featureX'
$ git log --oneline
4a4439d (HEAD -> featureX) D
3388bbb C
47efad3 B
7811736 A
```

Merge

Merge Overview

What Does Merge Do

> Merging combines the work of independent branches



Type of Merge

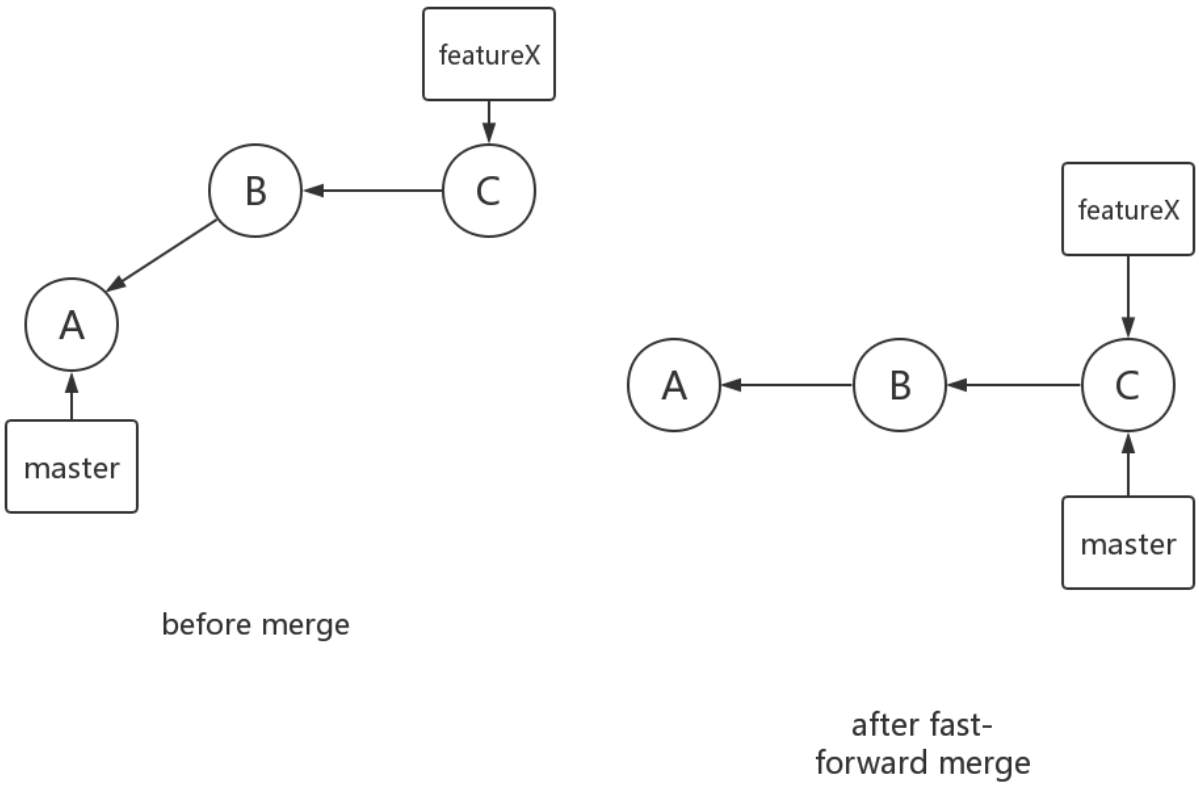
- Two main types of merges:
    1. Fast-forward merge
    2. Merge commit

Fast-forward Merge

What Does Fast-forward Merge Do

> Just moving the base branch label to the tip of the topic branch

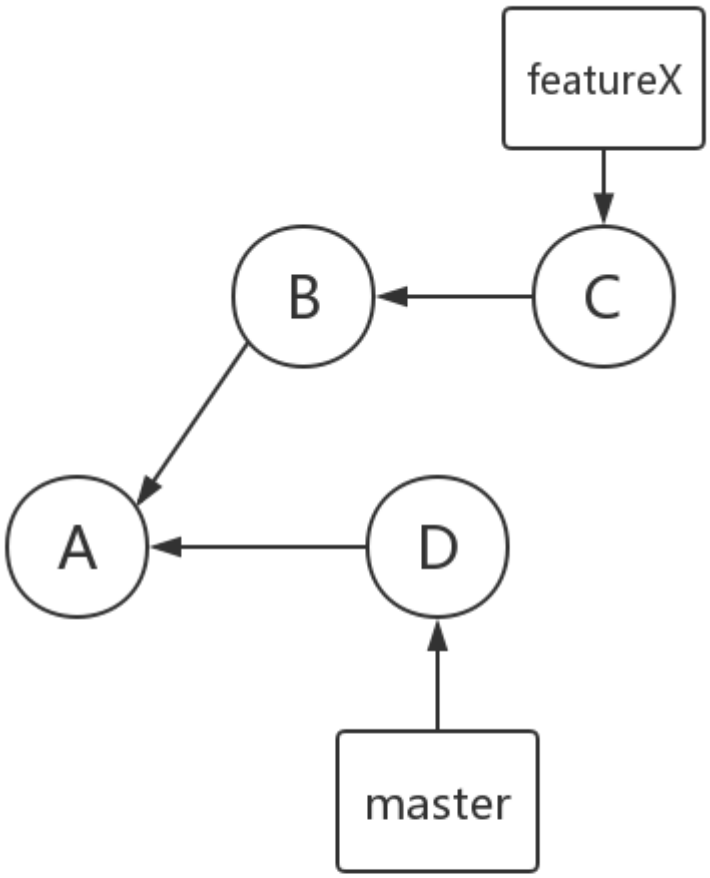
Graph



Note

- A Fast-forward merge is possible only when no other commits have been made to the base branch since the topic branch was created

In this example, a Fast-forward merge is not possible:



fast-forward
merge not
possible

Example

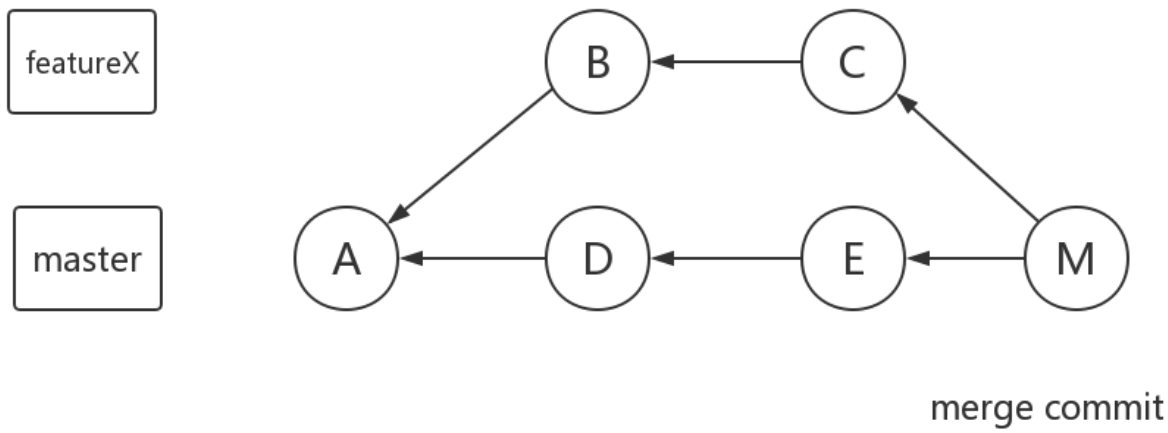
```
# The basic steps to performing a fast-forward-merge:
$ git checkout master
$ git merge featureX # Attemping a fast forward merge is the default
$ git branch -d featureX
```

Merge Commit

What Does Merge Commit Do

1. Combines the commits at the tips of the merged branches
2. Places the result in the merge commit

Graph



Example

Not Fast Forwardable

```
# The basic steps to performing a merge commit(if the merge is not fast forwardable):
$ git checkout master
$ git merge featureX
#1. automatically attempt to create a merge commit if the merge is not fast forwardable
#2. accept or modify the default merge message created by git
$ git branch -d featureX
```

Is Fast Forwardable

```
# The basic steps to performing a merge commit(if the merge is fast forwardable):
$ git checkout master
$ git merge --no-ff featureX
#1. [--no-ff] means that a merge commit will always be created even the merge is fast forwardable
#2. accept or modify the merge default message created by git
$ git branch -d featureX
```

Resolving Merge Conflicts

Merge Conflict Overview

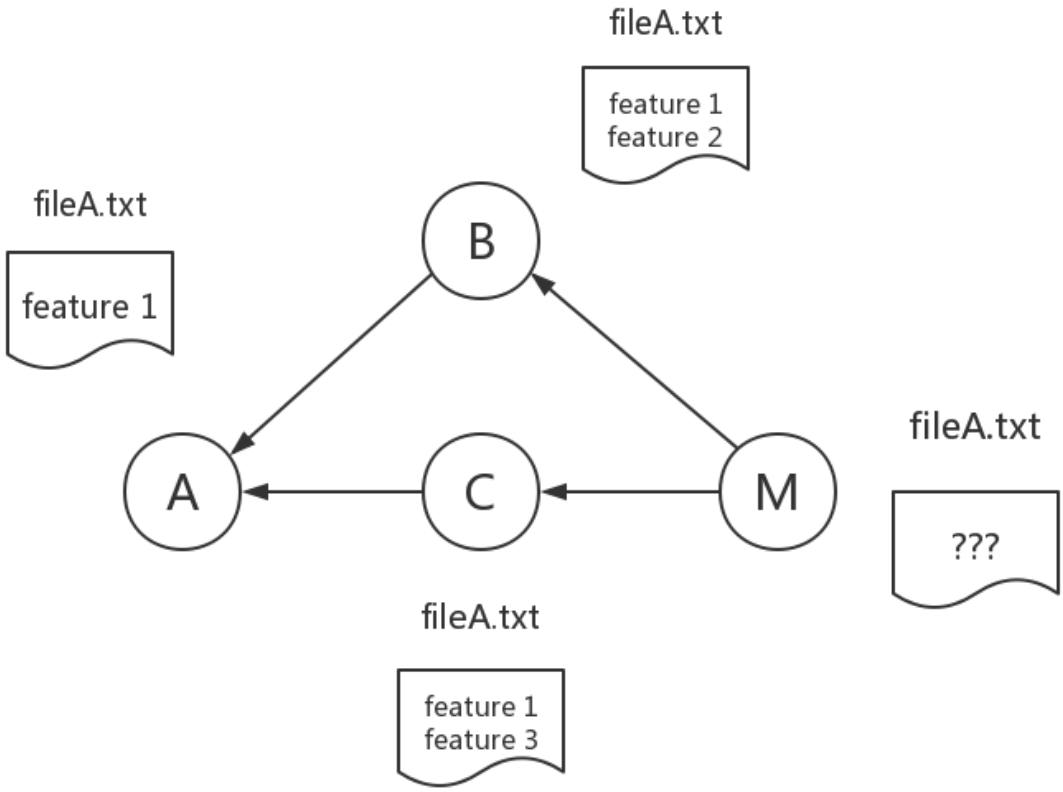
Hunk

> In Git, a part of file is called a **hunk**

When Does Merge Conflict occur

> The merge conflicts occur when two branches modify the same hunk of the same file

Graph

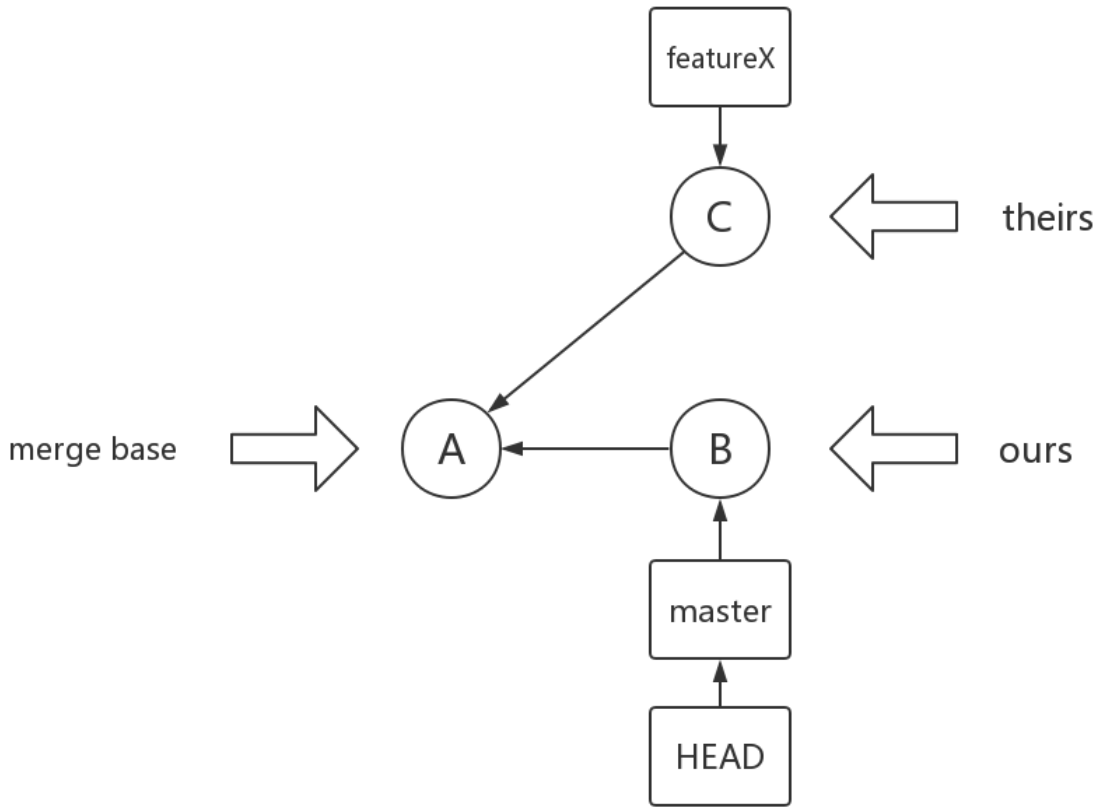


Note

> 1. A person needs to make a decision on how to resolve the merge conflict
> 2. Avoid making a lot of changes over a long period of time without merging

Resolving a Merge Conflict

Graph



1. The tip of the current branch (B) - "ours" or "mine"
2. The tip of the branch to merged (C) - "theirs"
3. A common ancestor (A) - "merge base"

Example

```
# Basic steps to resolve a merge conflict:
$ git checkout master
$ git merge featureX
#1. Assume both branches modified the same hunk in a file in different ways
#2. At this point, git will modifed the file which will show you exactly where the conflicts are and placed the file in your working tree
---
#3. At this point, if you don't want to continue to merge, you can use the fllowing command to abort the merge:
$ git merge --abort
---
#3. Modifiy the file and resolve the merge conflict which requires human judgement and then make a commit
$ git add fileA.txt
$ git commit # This will open the default editor since you don't use '-m'
$ git branch -d featureX
```

Note

The conflicted hunks will be displaying just like this:

```
feature 1 # unconflicted
<<<<<<< HEAD
feature 3 # ours
=======
feature 2 # theirs
>>>>>>> featureX
```

Tracking Branches

Tracking Branch Overview

What Is a Tracking Branch

1. A tracking branch is a local branch that represents a remote branch
2. A tracking branch name is like `remotename/branchname`

Note

> Tracking branches are only updated with network commands like clone, fetch, pull, and push

Viewing Tracking Branch

How to Watch the Tracking Branch

Command

`git branch --all` to display all local and tracking branch names

Example

```
$ git branch --all
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
```

Special Tracking Branch

HEAD Reference

> A reference named `remotes/origin/HEAD` is a symbolic reference meaning that it is a reference that points to another reference, and this **specifies the default remote tracking branch**

Example

```
$ git branch --all
* master
  remotes/origin/HEAD -> origin/master # the default remote tracking branch
  remotes/origin/master
```

Note

- You can use only `<remotename>` name instead of the whole tracking branch name which is known as `remotename/branchname` in git command and in this case you will use the default remote tracking branch which is specified by `remotes/origin/HEAD`

```
$ git log origin/master --oneline # see the commits of tracking branch origin/master
$ git log origin --oneline # the same as before by using the default tracking branch
```

- Change the default remote tracking branch with `git remote set-head <remotename> <branch>`

```
$ git remote set-head origin master
$ git branch --all
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
```

Viewing Tracking Branch Status

- `git status` includes tracking branch status

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.
# Means that since of the last time that we use a network command like fetch, we have
the latest commit in our local repository
```

- `git status` will inform you if the cached tracking branch information is out of synchronous with your local branch

```
$ git status
On branch master
Your branch is ahead of `origin/master` by 1 commit
```

Fetch, Pull and Push

Network Command Overview

- **Clone** - Copies a remote repository
- **Fetch** - Retrieves new objects and references from the remote repository
- **Pull** - Fetches and merges commits locally
- **Push** - Adds new objects and references to the remote repository

Fetch

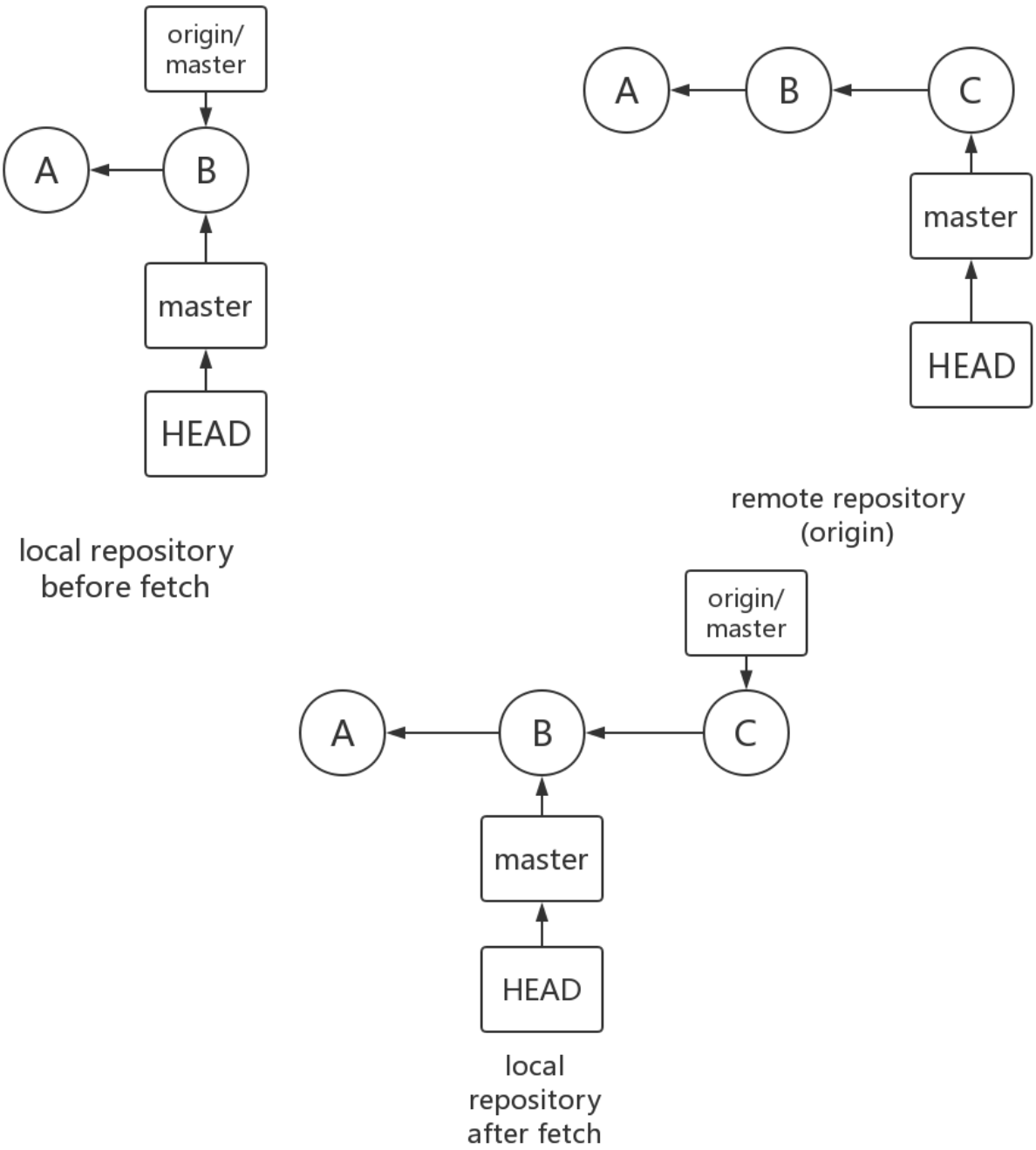
What Does Fetch Do

1. Retrieves new objects and references from another repository
2. Tracking branches are updated
3. Does not impact the local branch labels

Example

```
$ git log origin/master --oneline
* 92dd144 (HEAD -> master, origin/master, origin/HEAD) ...
$ git fetch
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From ../url/to/projectname.git
   92dd144..90fcaa3  master     -> origin/master
$ git log origin/master --oneline
90fcaa3 (origin/master, origin/HEAD) ...
92dd144 (HEAD -> master) ...
```

Graph



Note

- `git status` will inform you that your current branch is behind the tracking branch

```
$ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
```

Pull

What Does Pull Do

Combines `git fetch` and `git merge FETCH_HEAD` (FETCH_HEAD is an alias for the tip of the tracking branch)

- If objects are fetched, the tracking branch is merged into the current local branch
- This is similar to a topic branch merging into a base branch

Example

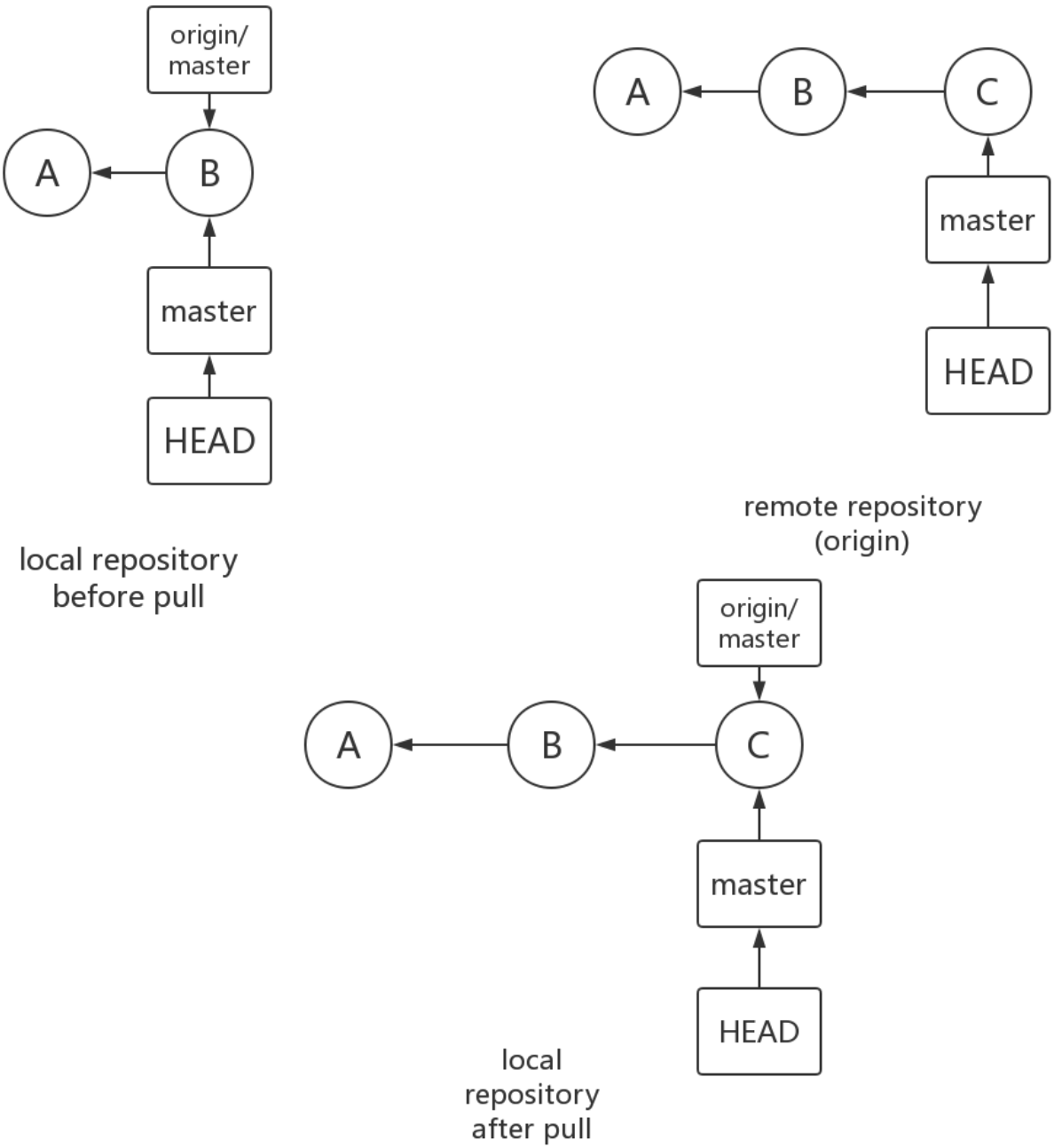
```
$ git pull
Updating 92dd144..90fcaa3
Fast-forward
...
```

`git pull` Command Options

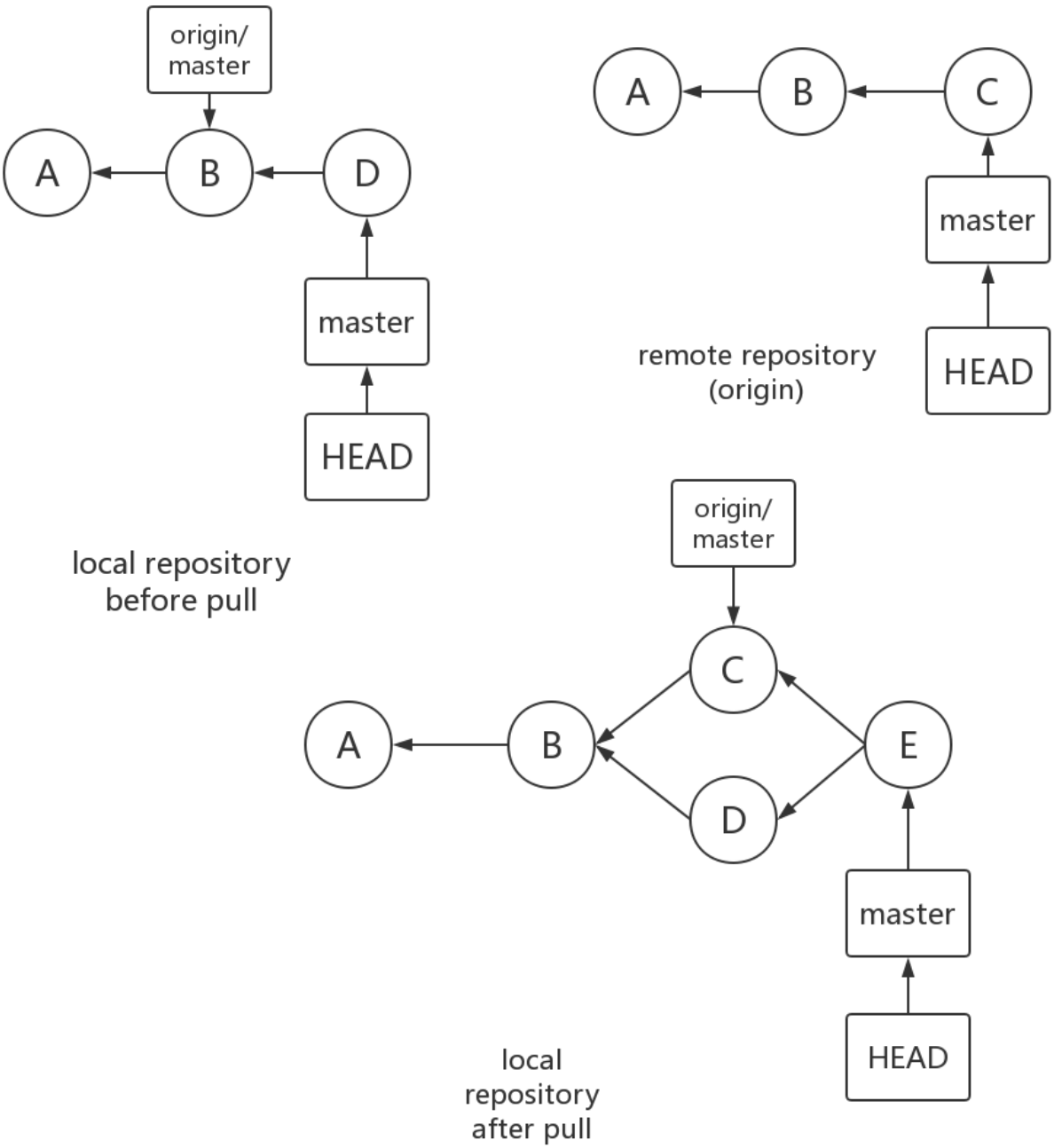
- `--ff` (default) fast-forward if possible, otherwise perform merge commit
- `--no-ff` -always include a merge commit
- `--ff-only` only accepts fast-forward merges and Git will cancel the merge instead of doing a merge commit

Graph

Pull With a Fast-Forward Merge



Pull With A Merge Commit



Push

How to Push

`git push [-u] [repository] [branch]`

- `-u` Track this branch (`--set-upstream`)

Example

```
$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.04 KiB | 354.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To ../url/to/projectname.git
   8b42715..7c42389  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Note

> Fetching or pulling before you push is suggested