Git Deep Dive & Best Practice

What is version control system?

- The system can be used to track the history of changes, can find out:
  - Which changes were made?
  - Who made the changes?
  - When were the changes made?
  - Why were changes needed?

Hash 算法

- 是把任意长度的输入通过散列算法变换成固定长度的输出
  - MD5 128bit
  - SHA1 160bit
  - SHA256 256bit
  - SHA512 512bit
- https://passwordsgenerator.net/md5-hash-generator/
- https://passwordsgenerator.net/sha1-hash-generator/

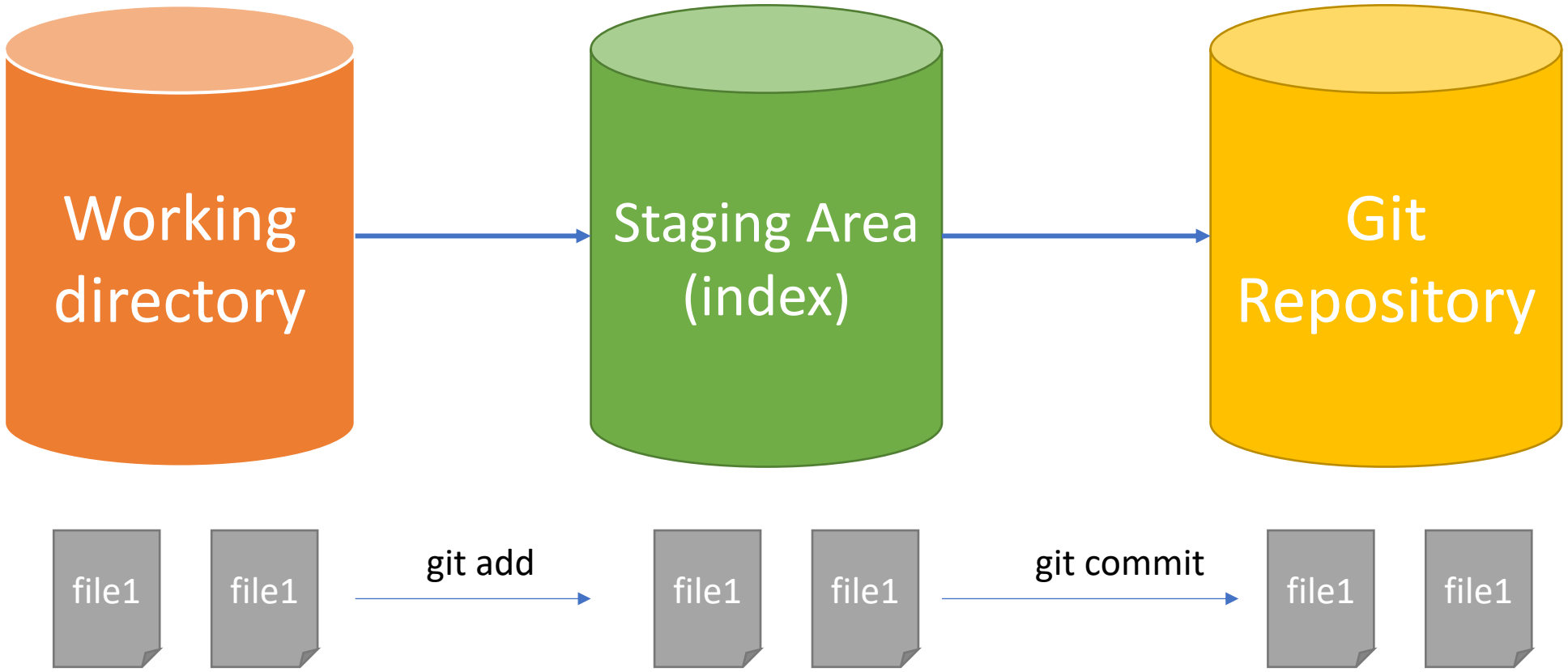
hello git

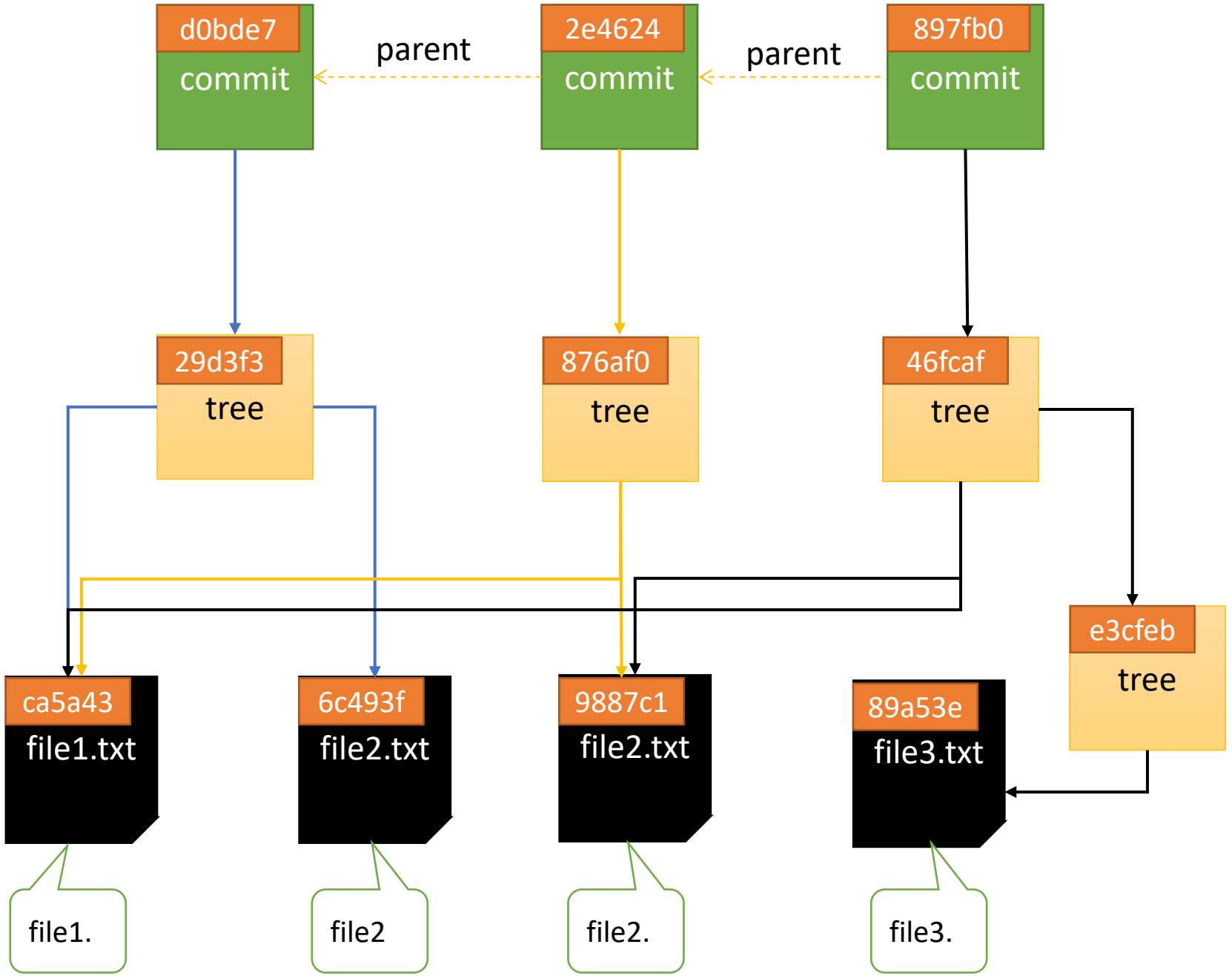
SHA1

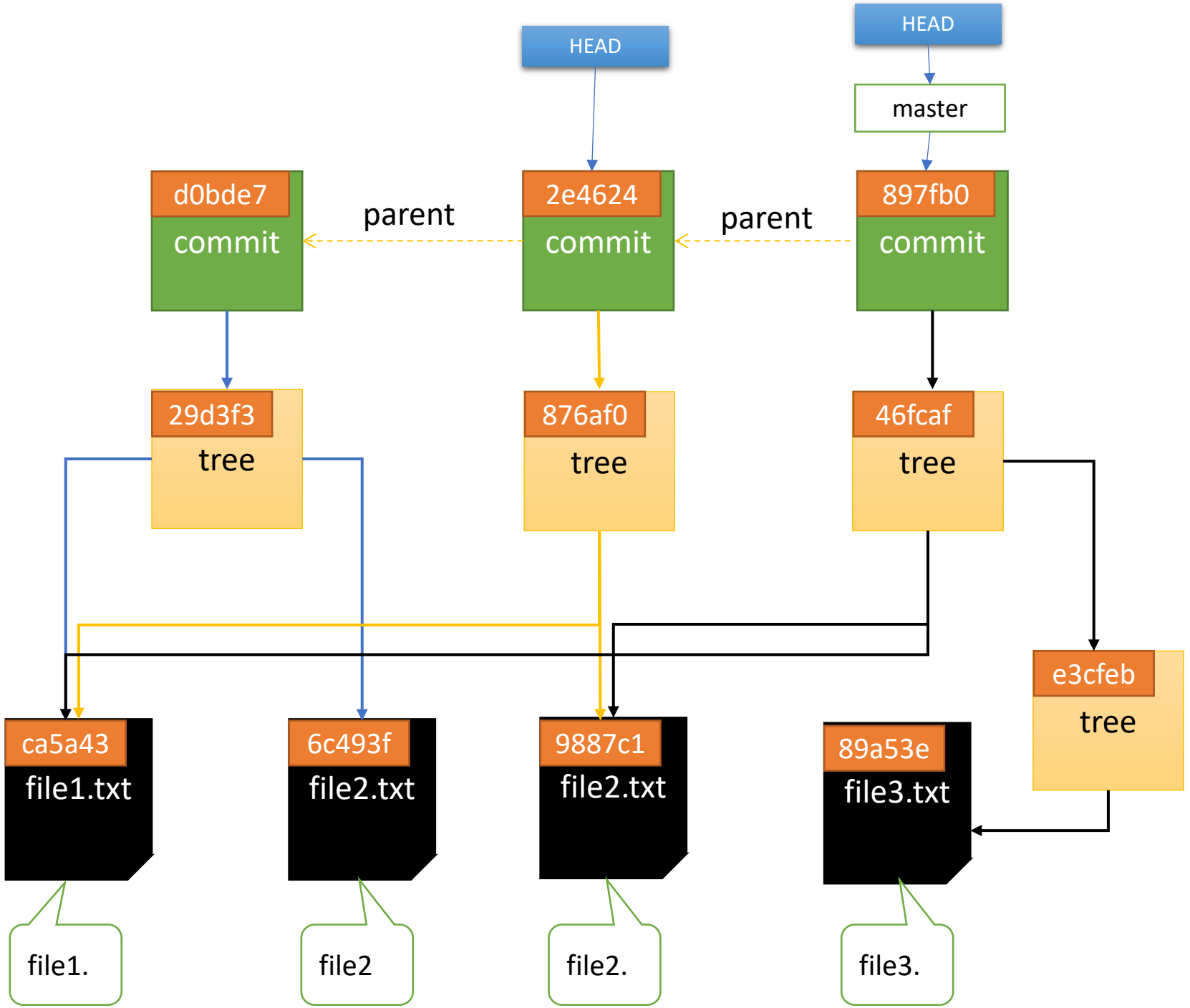
021d153ca2ec1771bb2b0ca33031f9ef5e6946bd　　长度40, 16进制

00000010000111010001010100111100101000101110110000010111011100011011110110010101100001100101000110000110000111111001111011110101111001101001010001101011 1101

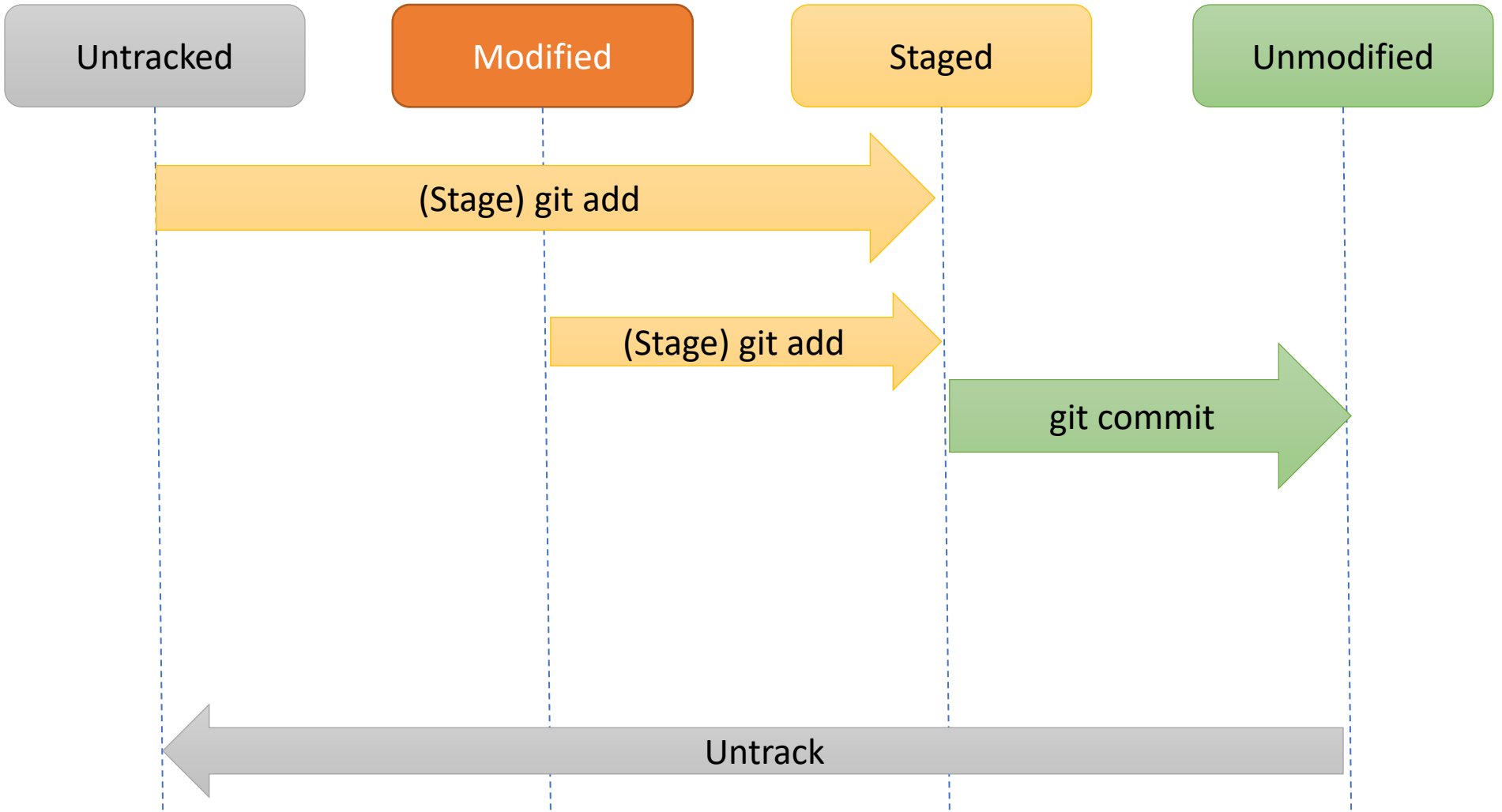
长度160，　二进制


Working directory
Staging Area (index)
Git Repository
file1
file1
git add
file1
file1
git commit
file1
file1


d0bde7
commit
parent
2e4624
commit
parent
897fb0
commit
29d3f3
tree
876af0
tree
46fcaf
tree
e3cfeb
tree
ca5a43
file1.txt
6c493f
file2.txt
9887c1
file2.txt
89a53e
file3.txt
file1.
file2
file2.
file3.


HEAD
HEAD
master
d0bde7
commit
parent
2e4624
commit
parent
897fb0
commit
29d3f3
tree
876af0
tree
46fcaf
tree
e3cfeb
tree
ca5a43
file1.txt
6c493f
file2.txt
9887c1
file2.txt
89a53e
file3.txt
file1.
file2
file2.
file3.

GIT的文件状态


Untracked
Modified
Staged
Unmodified
(Stage) git add
(Stage) git add
git commit
Untrack

Git Basic Commands

- git init (initialize an empty git repository)
- git status (show the working tree status)
- git add (add file contents to the index)
- git rm (Remove files from the working tree and from the index)
- git restore (Restore working tree files)
- git commit (Records the changes to repository)
- git log (show commit logs)

Git Low Level Commands

- git cat-file <SHA1>
  - -t (git object type)
  - -s (git object size)
  - -p (git object content)
- git ls-files
  - -s (Show staged contents' mode bits, object name and stage number in the output)

Git Objects

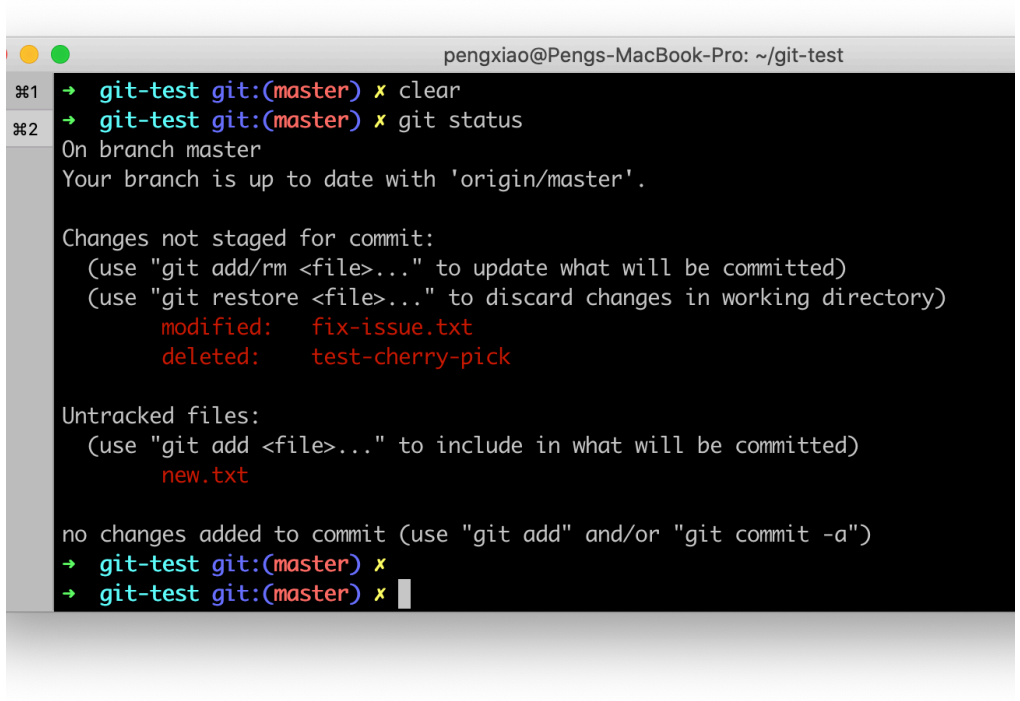
- Git is a content-addressable filesystem. Great. What does that mean? It means that at the core of Git is a simple key-value data store. What this means is that you can insert any kind of content into a Git repository, for which Git will hand you back a unique key you can use later to retrieve that content.
- Object types:
  - Blob - data (source code file)
  - Tree – Pointers to file name, content, other trees
  - Commit – Author, message, pointer to a tree

Git Commands

- git init (initialize an empty git repository)
- git status (show the working tree status)
- git add (add file contents to the index)

File State

- Modified/deleted/new
- Staged
- Committed

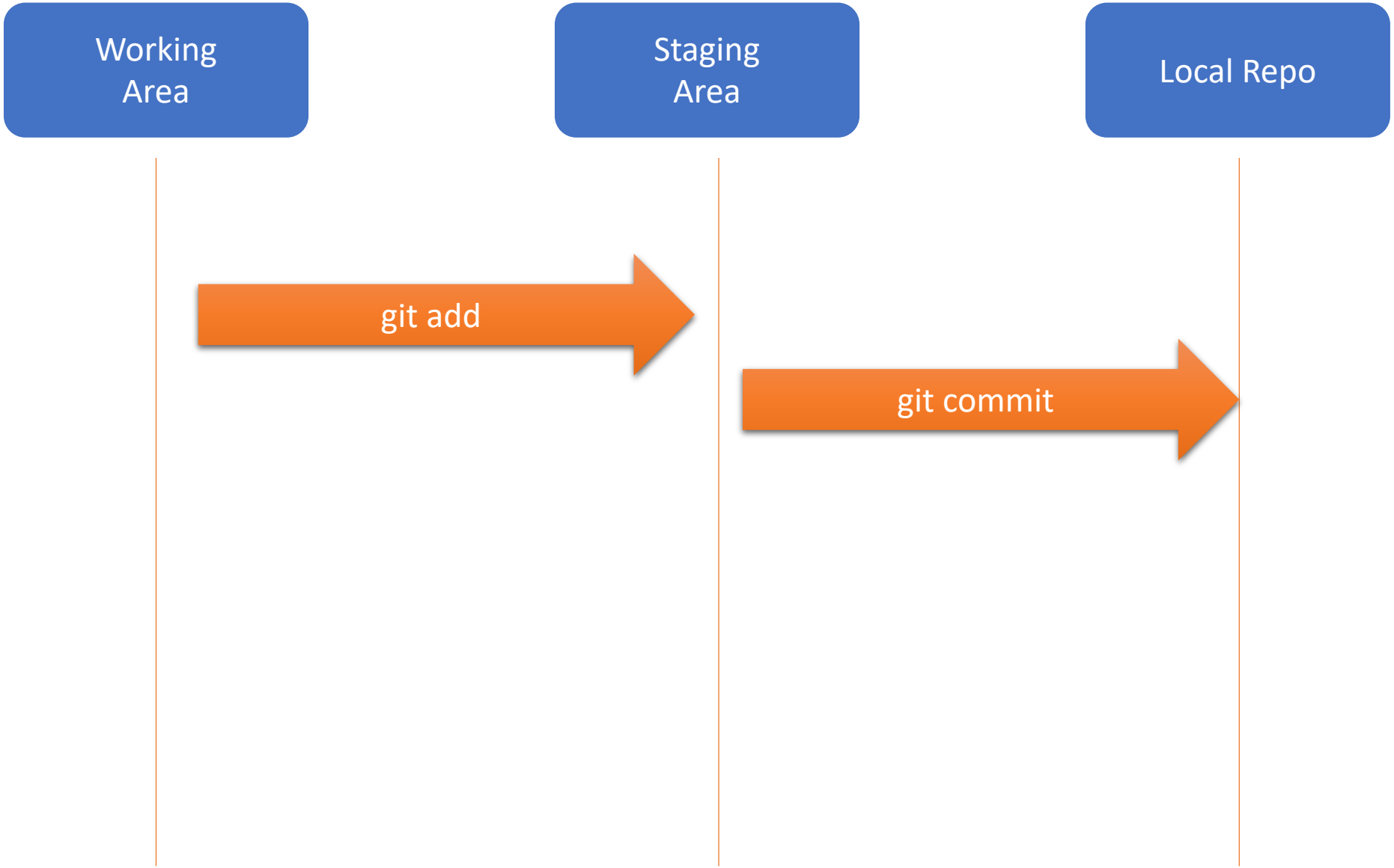


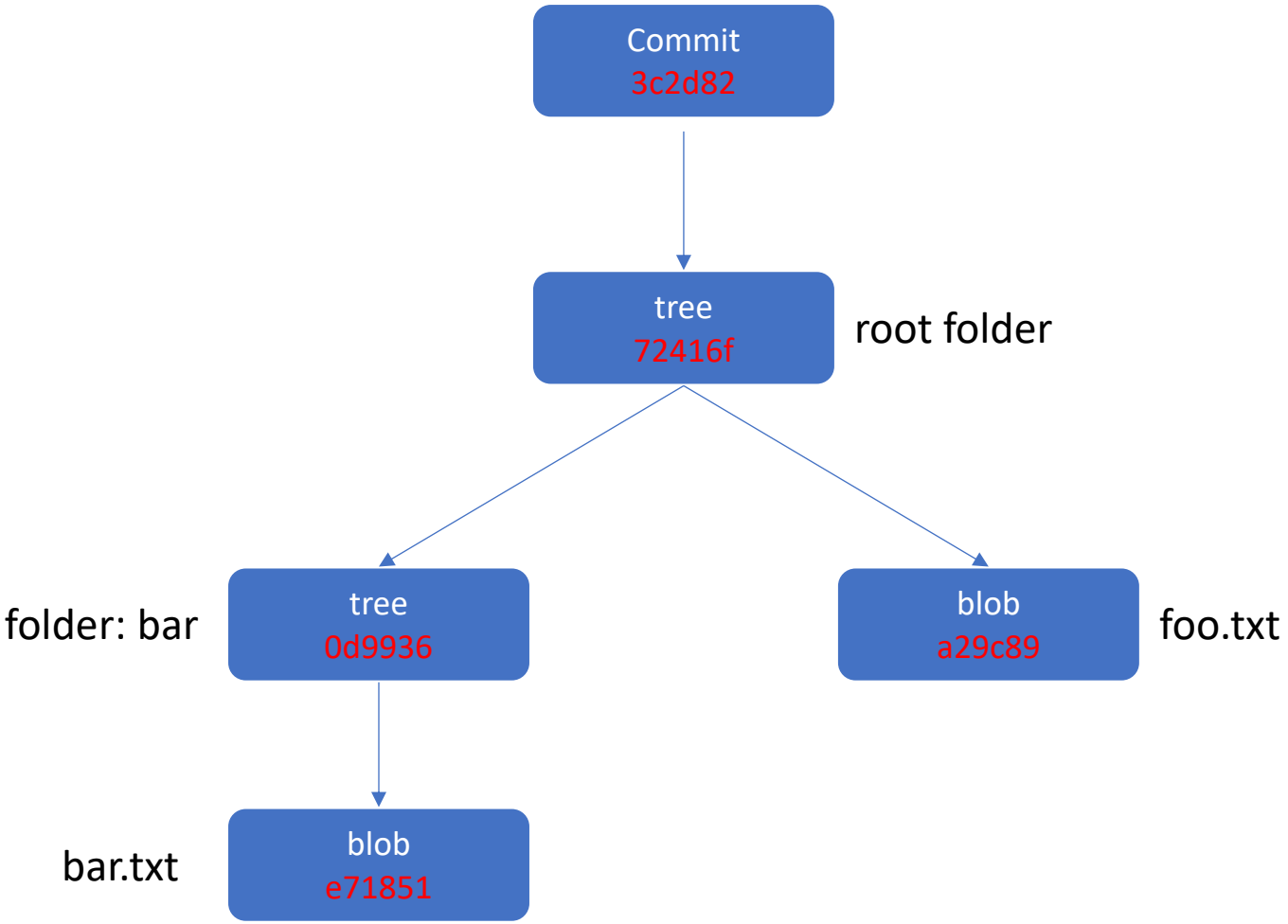
Homework

- Initialize one empty git repository
- create one file called foo.txt, add text "this is foo" into this file
- create one folder called bar, create a file called bar.txt inside of bar folder, and add text "this is bar" into this file
- Run command git add * to add all files and folder into git database
- use command git cat-file to check object files in .git/objects

Homework

- Initialize one empty git repository
- create one file called foo.txt, add text "this is foo" into this file
- create one folder called bar, create a file called bar.txt inside of bar folder, and add text "this is bar" into this file
- Run command git add * to add all files and folder into git database
- use command git cat-file to check object files in .git/objects


Working Area
Staging Area
Local Repo
git add
git commit


Commit
3c2d82
tree
72416f
root folder
folder: bar
tree
0d9936
blob
a29c89
foo.txt
bar.txt
blob
e71851

Git Pack file

- The **.pack file** contains the actual **Git** objects and the . **idx file** contains the **index** used to quickly locate objects within the . **pack file**. Objects in a **packfile** can either be deltified or non-deltified.

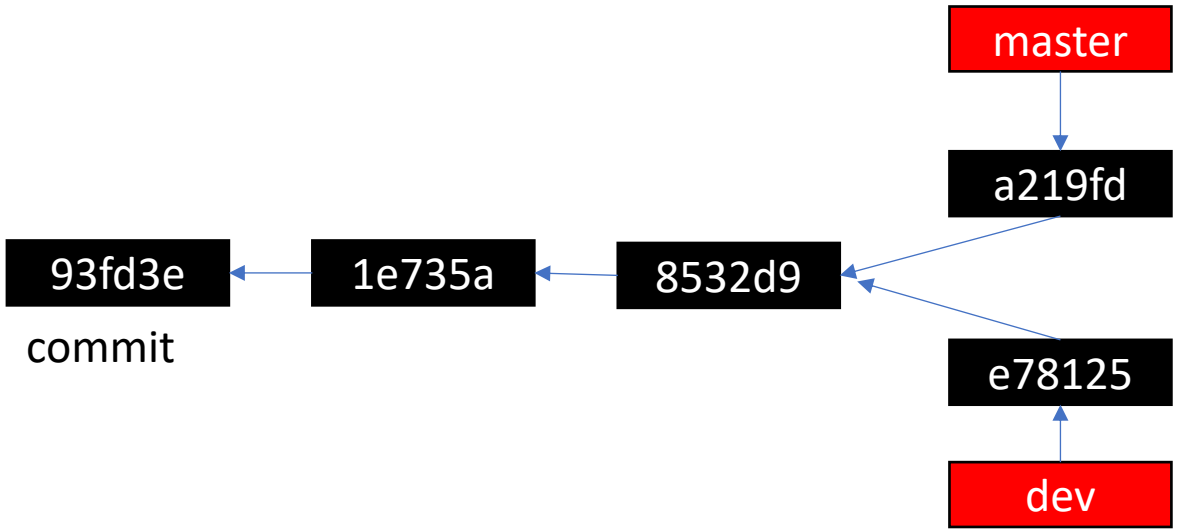
```
./.git/objects/pack
./.git/objects/pack/pack-3c65967d7eb8782749ba3b66be726b2941fc099a.idx
./.git/objects/pack/pack-3c65967d7eb8782749ba3b66be726b2941fc099a.pack
```

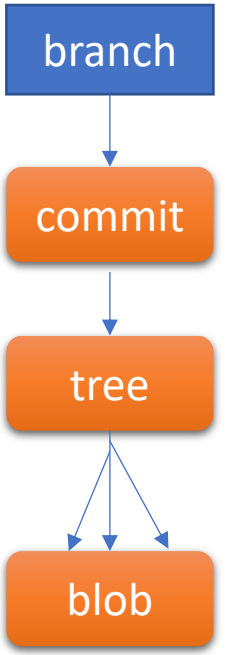
Git Commands

- git init (initialize an empty git repository)
- git status (show the working tree status)
- git add (add file contents to the index)
- git commit (Records the changes to repository)
- git log (show commit logs)
- git rm (Remove files from the working tree and from the index)

Branches

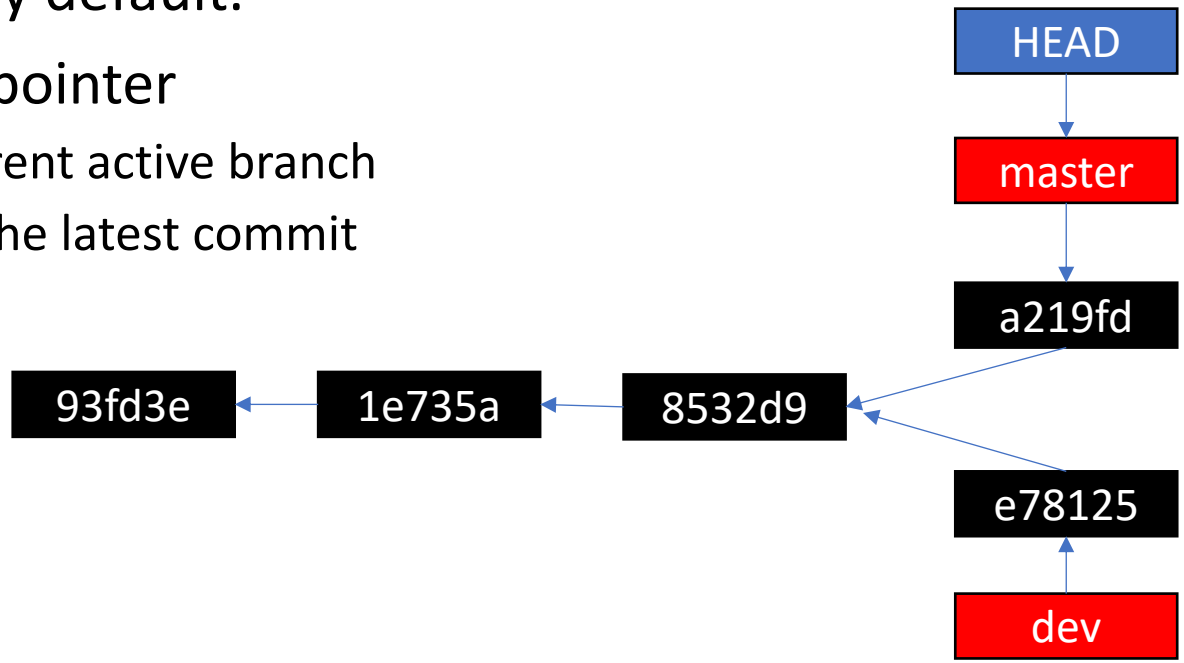
Branches are named pointers to commits





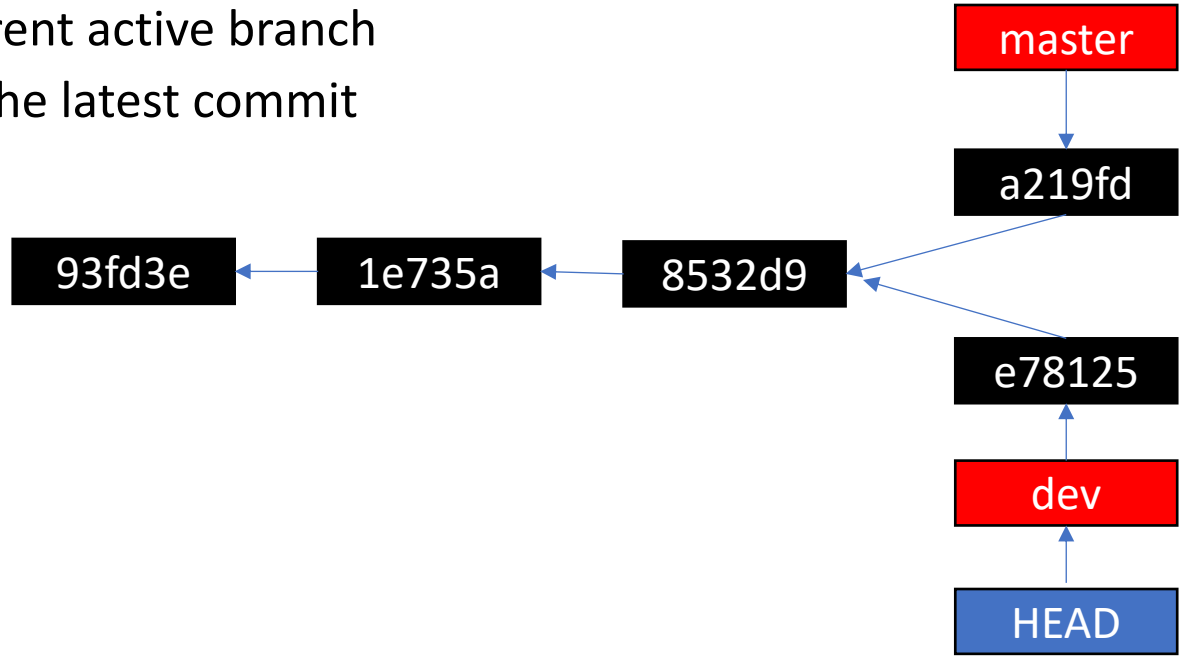
Master branch and HEAD

- Master is a branch, the canonical mainline branch by default.
- HEAD is a special pointer
  - Related with current active branch
  - Always point to the latest commit



Master branch and HEAD

- Master is a branch, the canonical mainline branch by default.
- HEAD is a special pointer
  - Related with current active branch
  - Always point to the latest commit



Git Commands

- git init (initialize an empty git repository)
- git status (show the working tree status)
- git add (add file contents to the index)
- git commit (Records the changes to repository)
- git log (show commit logs)
- git rm (Remove files from the working tree and from the index)
- git branch (list, create or delete branches)
- git checkout (change the current active branch)
- git merge (Join two or more development histories together)

Git Branch

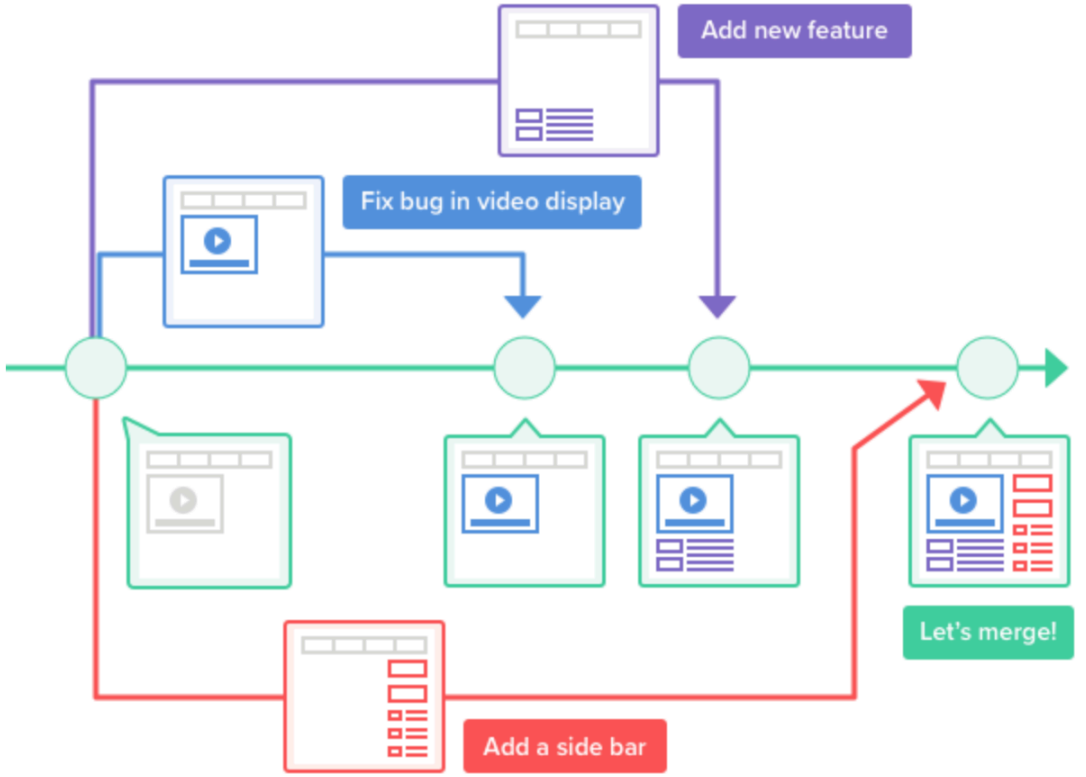
- git branch (list and branches we have)
- git branch <branch_name> (create a branch with branch_name, if there already have branch with the name you want to create, it will return error)
- git branch –D <branch_name> (delete branch, can't delete current active branch or branch not existing)
- git checkout (change the current active branch)
- git checkout –b <branch_name> (checkout a branch, will create that branch if it doesn't exsit)
- git branch –m <old_name> <new_name> (rename branch with new name)

Git Commands

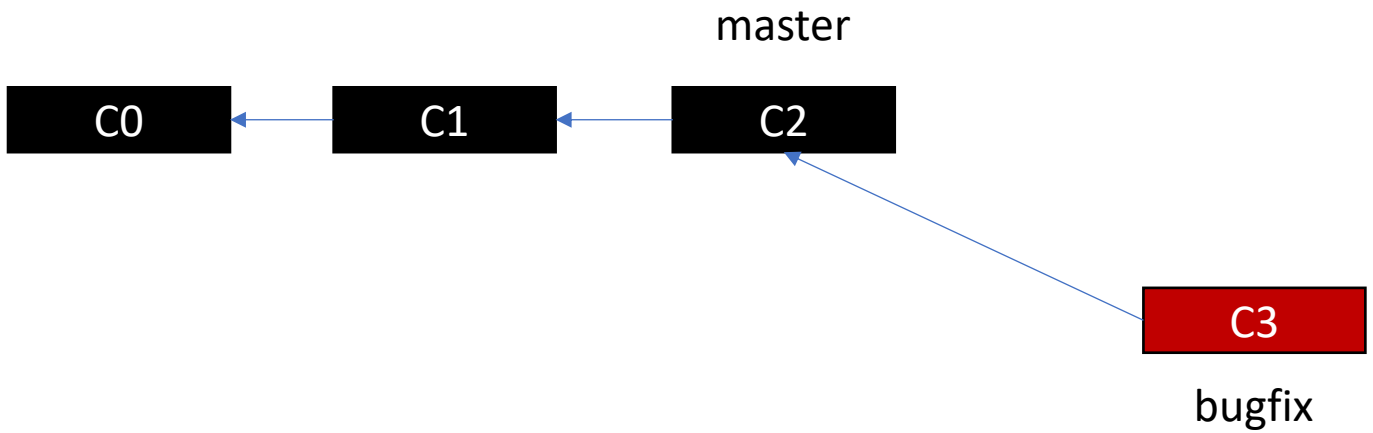
- git init (initialize an empty git repository)
- git status (show the working tree status)
- git add (add file contents to the index)
- git commit (Records the changes to repository)
- git log (show commit logs)
- git rm (Remove files from the working tree and from the index)
- git branch (list, create or delete branches)
- git checkout (change the current active branch)
- git merge (Join two or more development histories together)

Git Branch and Git Merge

- Nobody should work on Master branch directly.
- We work on our own branch, that branch could be a feature branch or a bugfix branch
- Always try to merge your own branch to master branch when you think your job is done

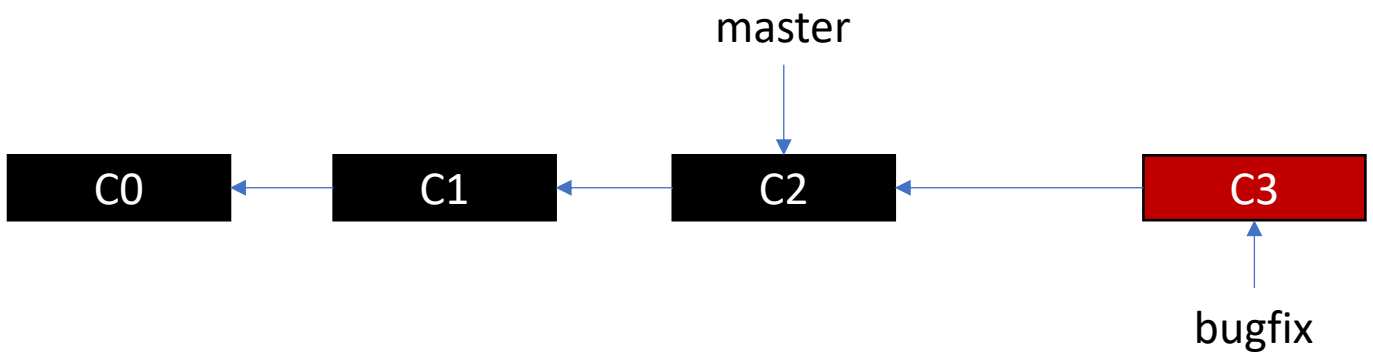


Git Merge – Fast Forward

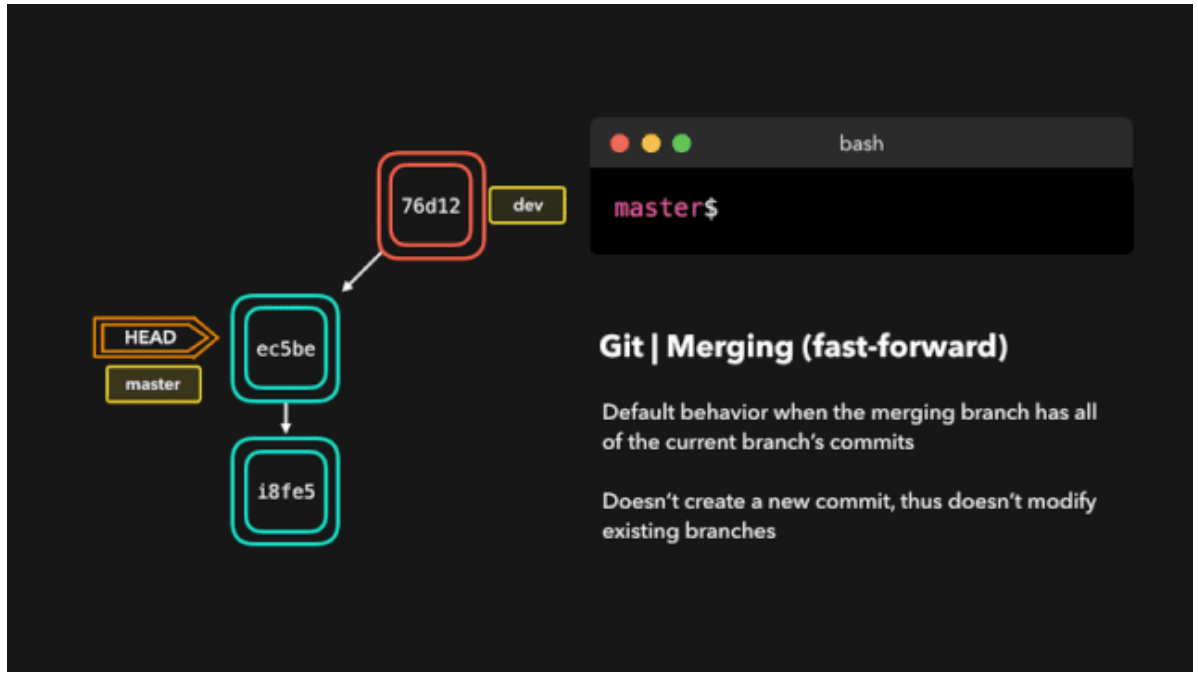


Git Merge – Fast Forward

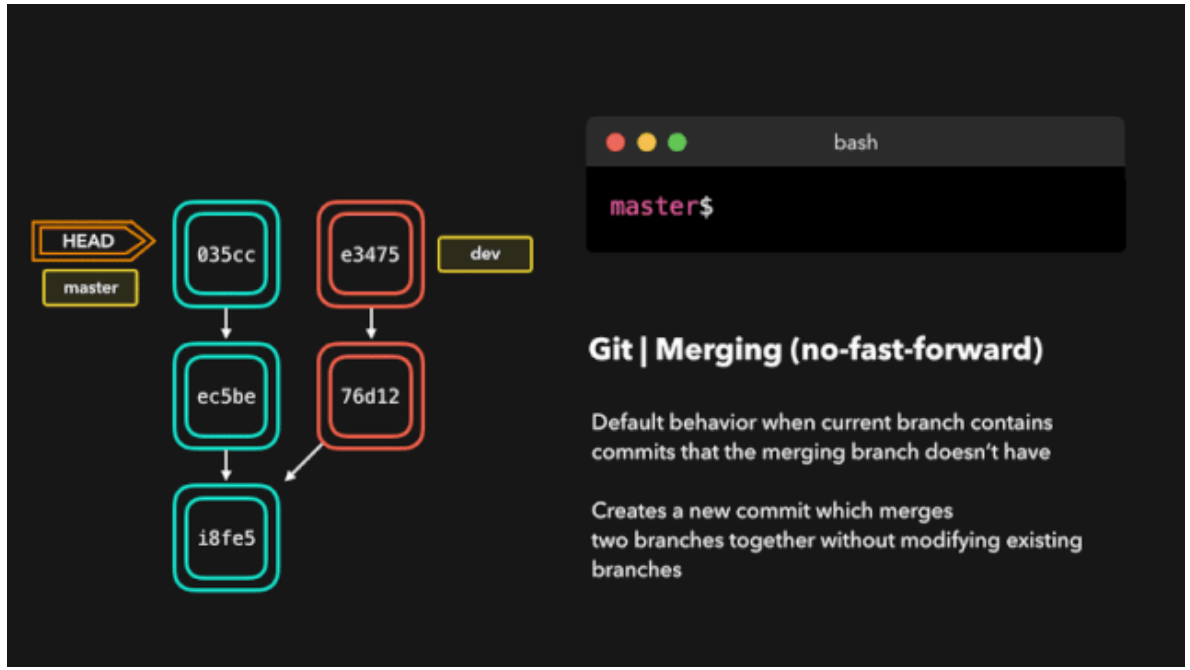
$ git merge bugfix

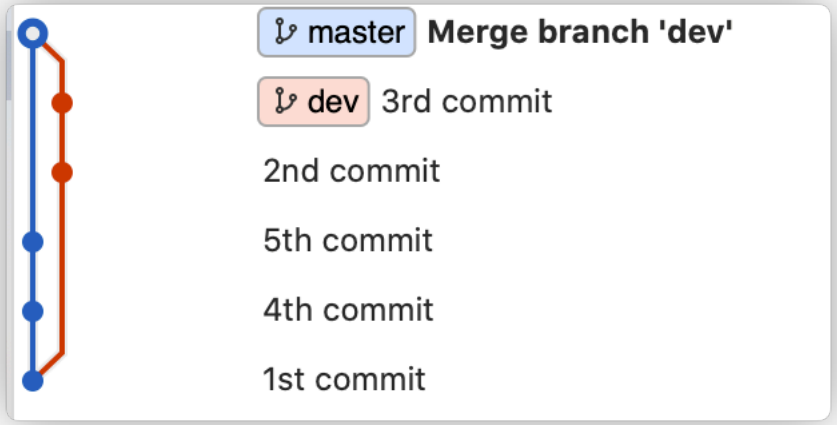


Git Merge – Fast Forward

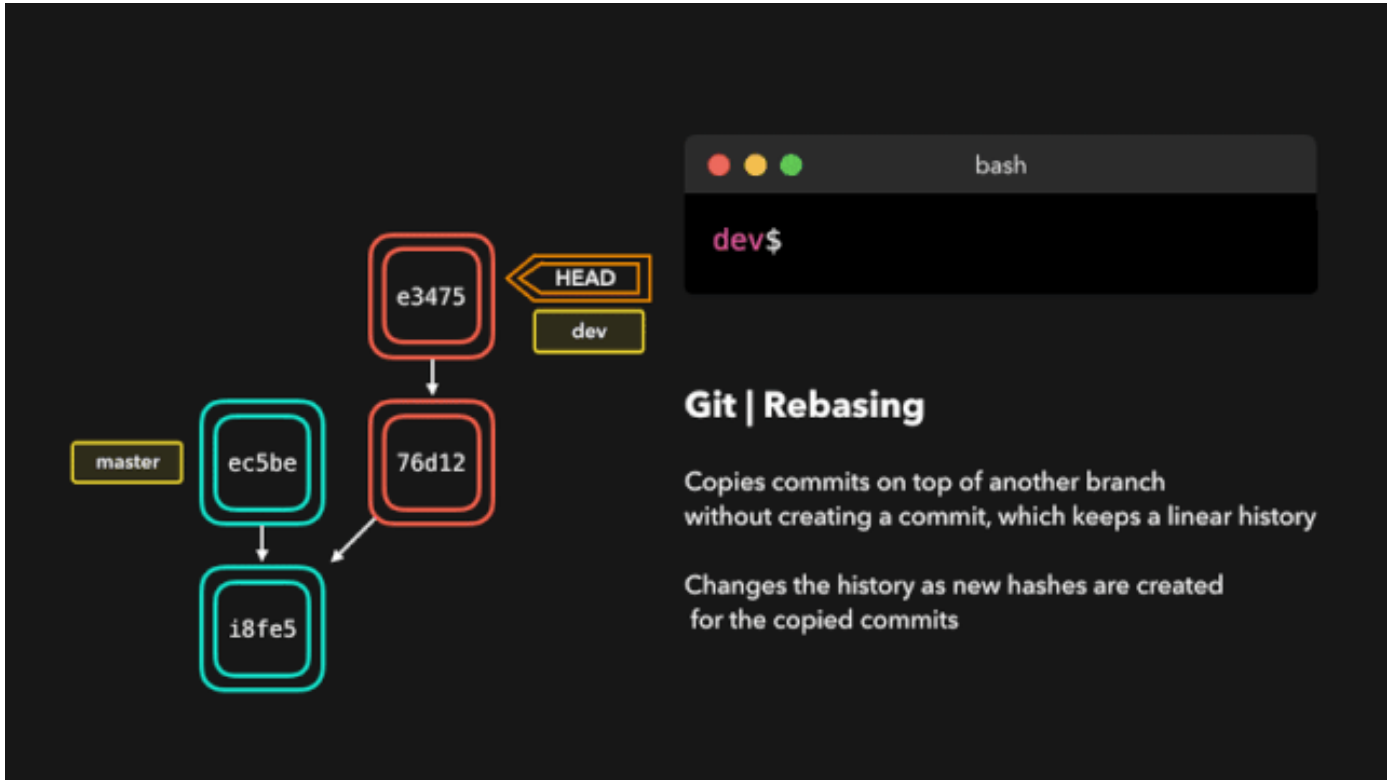


Git Merge – 3 way merge



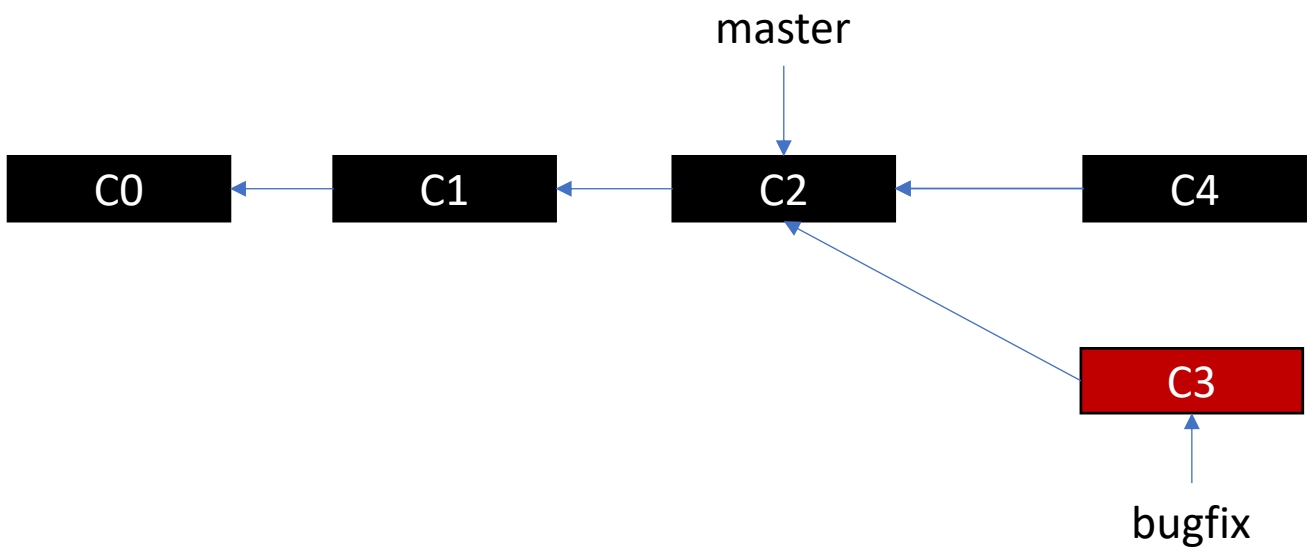


Git rebase


bash
dev$
e3475
HEAD
dev
master
ec5be
76d12
i8fe5
Git | Rebasing
Copies commits on top of another branch
without creating a commit, which keeps a linear history
Changes the history as new hashes are created
for the copied commits

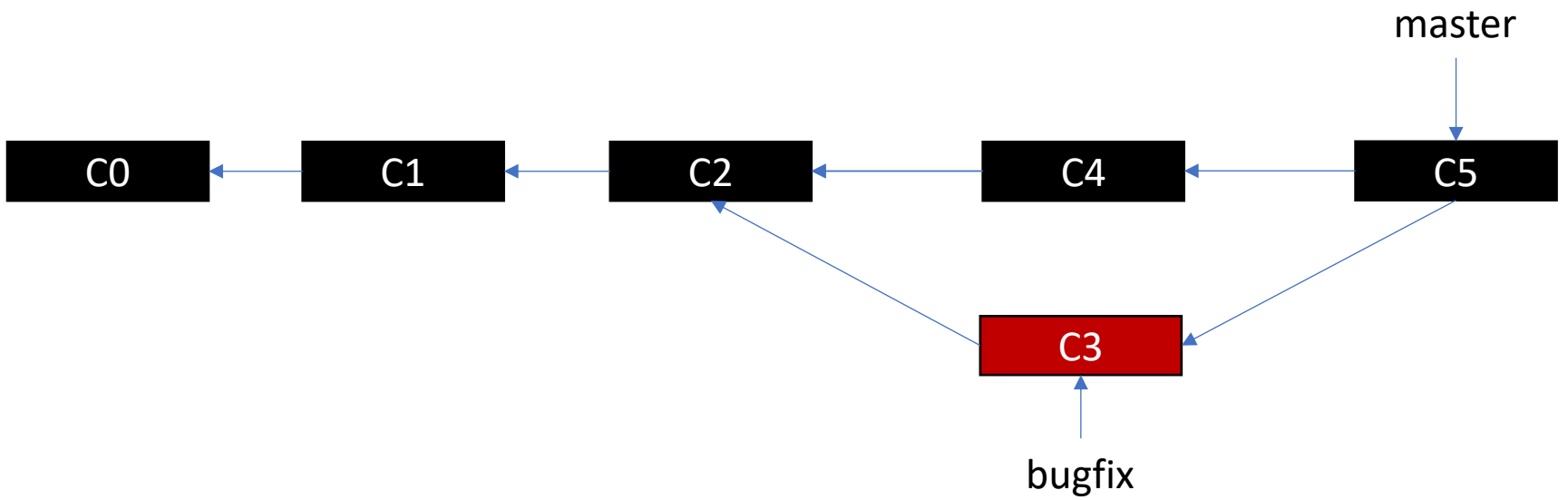
Git Merge – 3 way merge

$ git merge bugfix

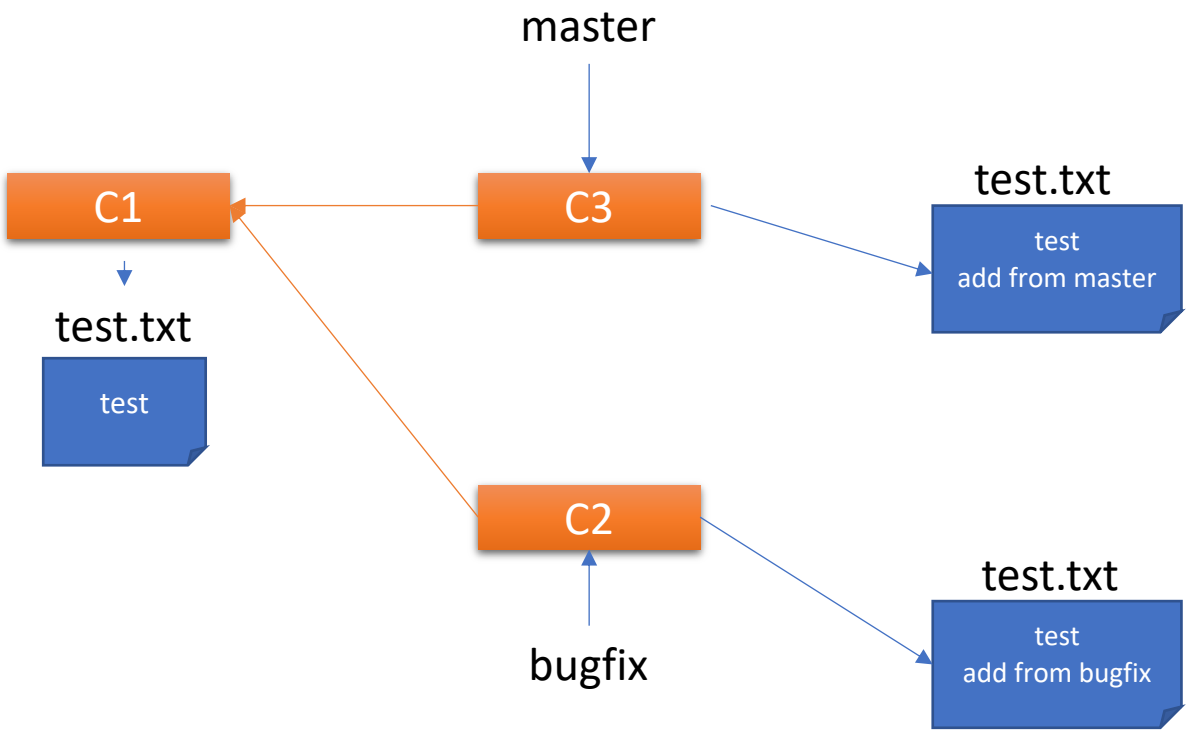


Git Merge – 3 way merge

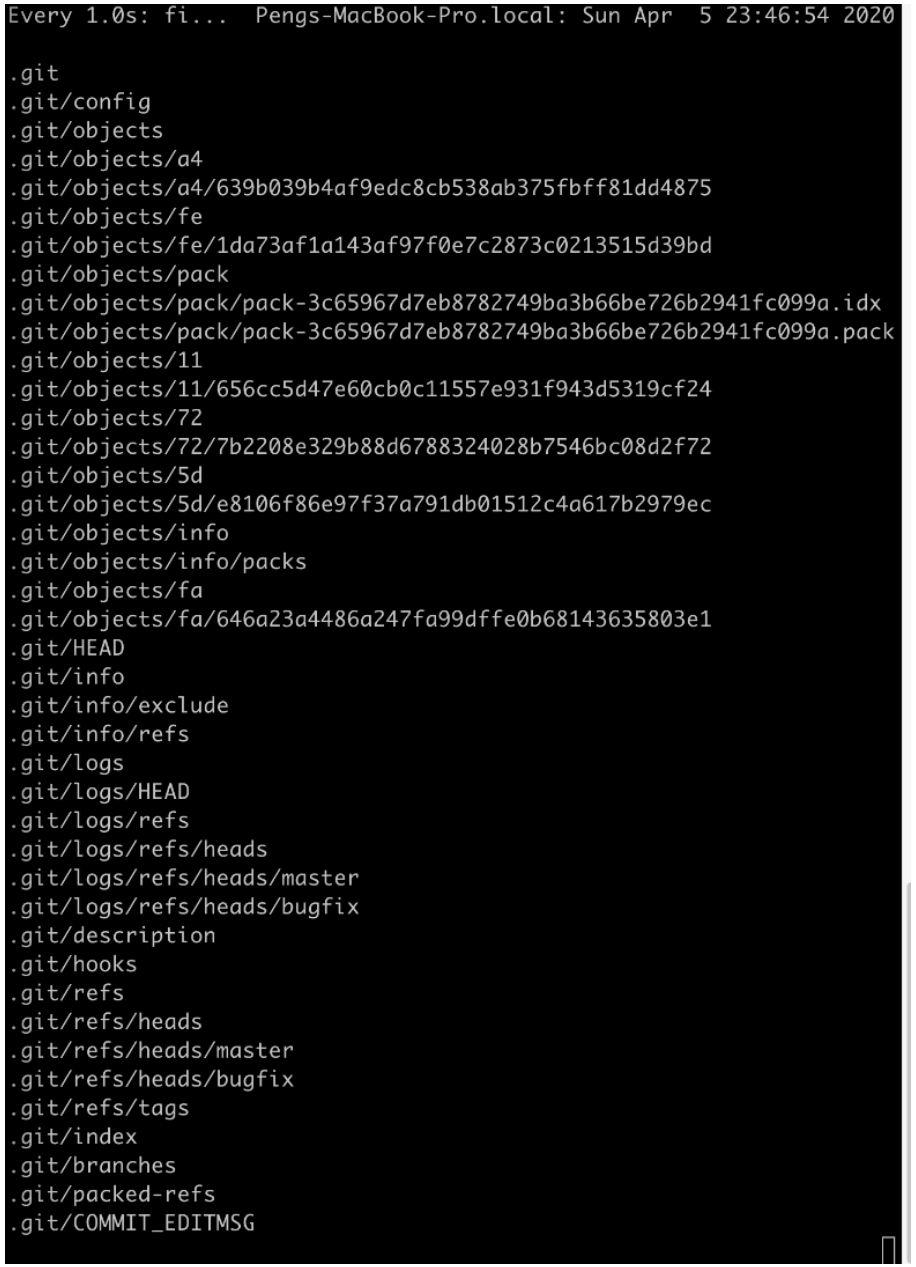
$ git merge bugfix



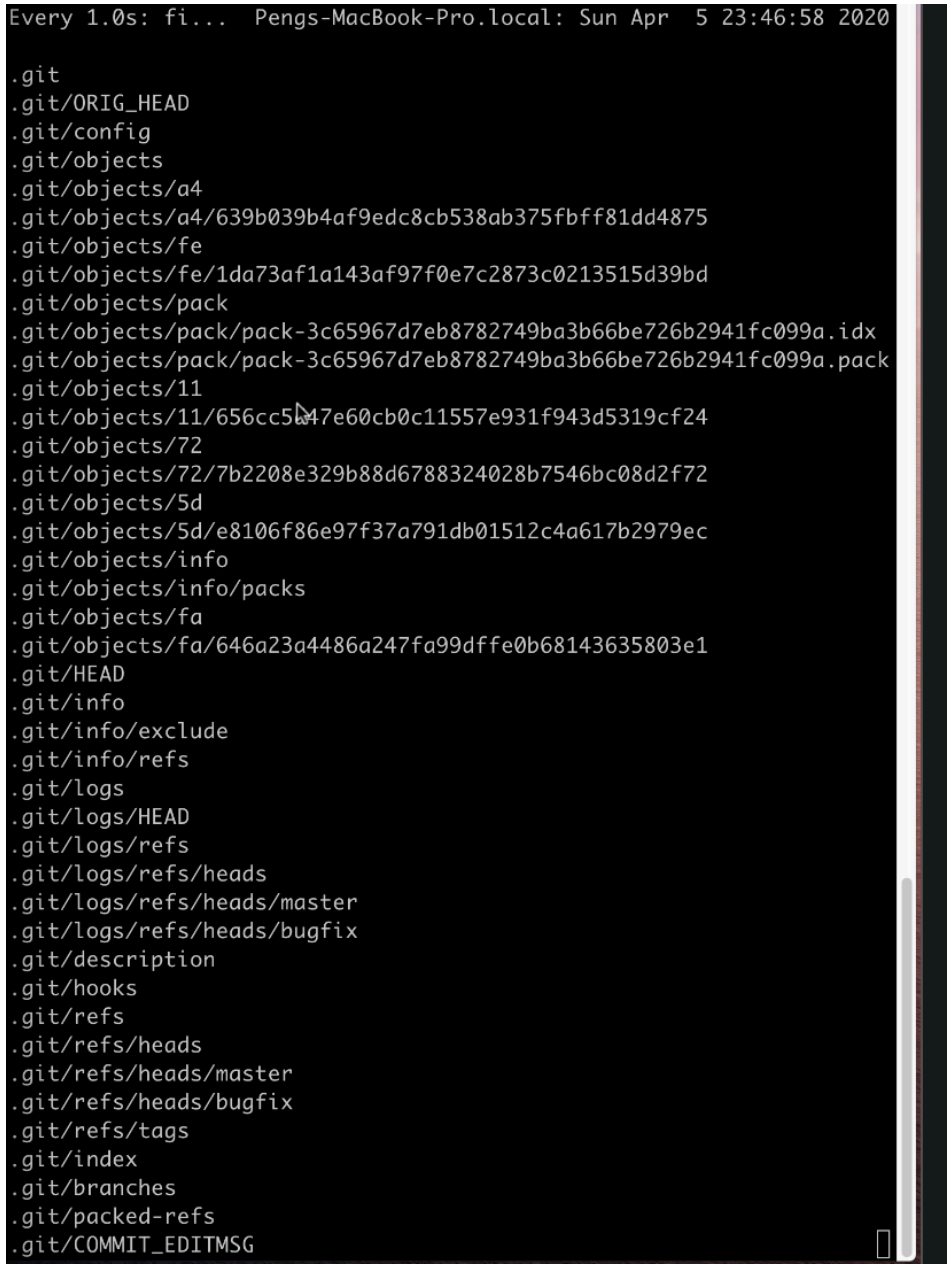
Git Merge – 3 way merge with conflict




master
dev
3rd commit
2nd commit
1st commit



before



after

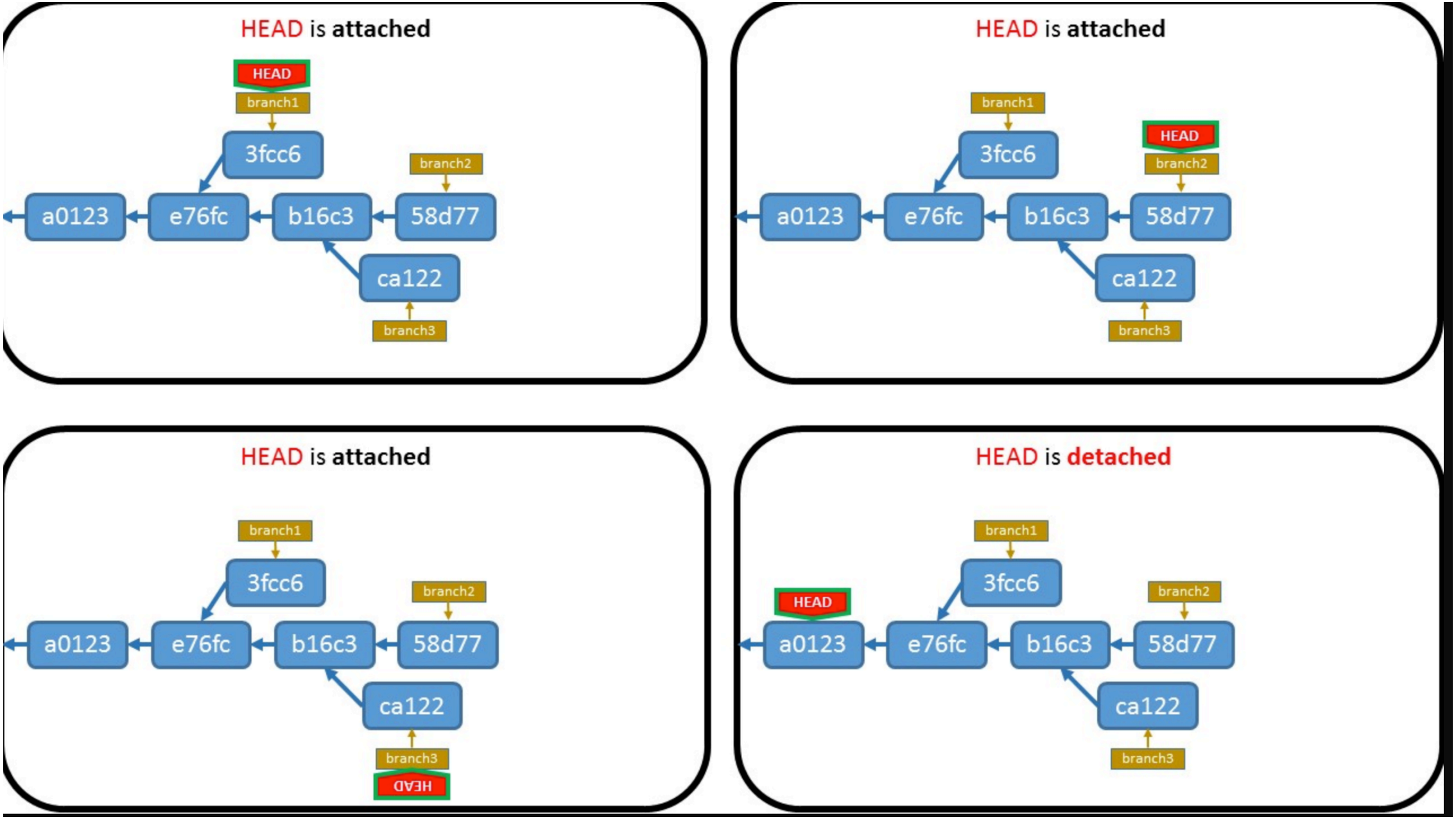
Git Commands

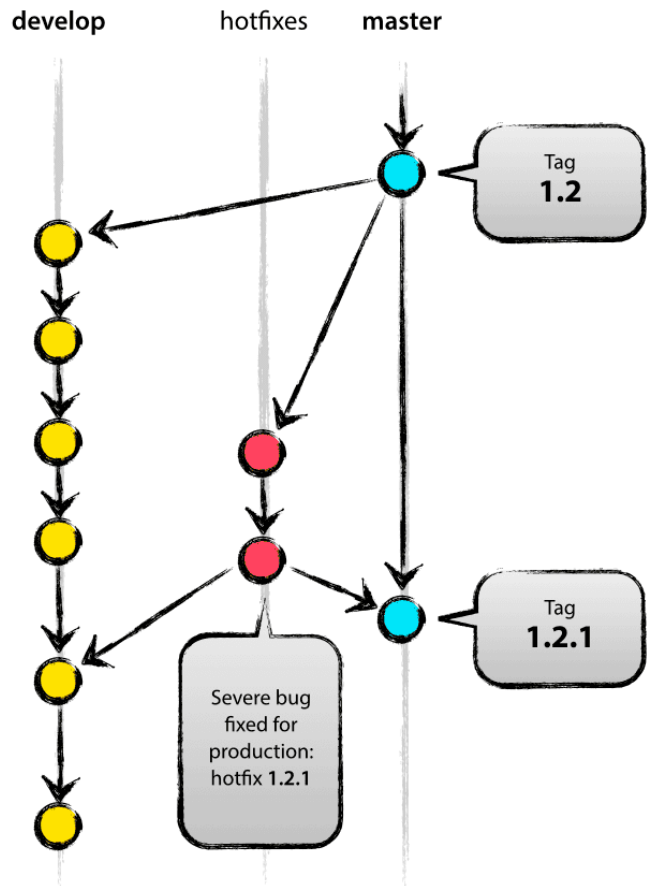
- git init (initialize an empty git repository)
- git status (show the working tree status)
- git add (add file contents to the index)
- git commit (Records the changes to repository)
- git log (show commit logs)
- git rm (Remove files from the working tree and from the index)
- git branch (list, create or delete branches)
- git checkout (change the current active branch)
- git merge (Join two or more development histories together)

Delete Branch

- git branch -D <branch_name> **will delete a branch regardless of its merge status**
- git branch -d <branch_name> = git branch --delete <branch_name> **will delete the branch only if the branch was fully merged**
- Git branch -D <branch_name> = git branch --delete --force = git branch –d -f

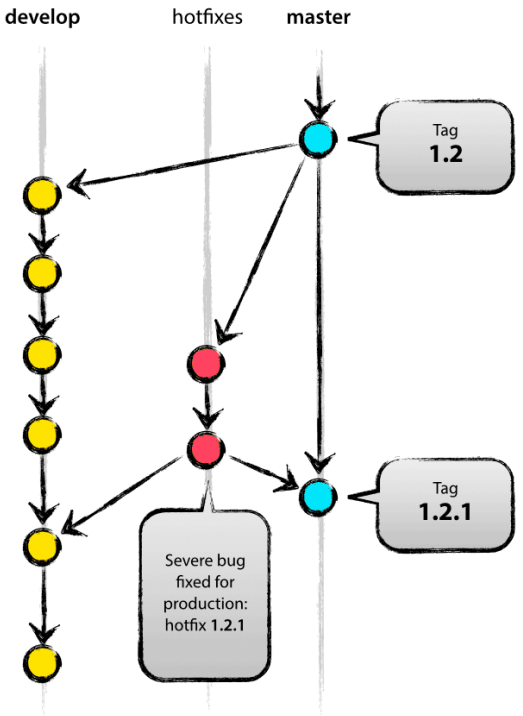
-d, --delete	delete fully merged branch
-D	delete branch (even if not merged)
-f, --force	force creation, move/rename, deletion


HEAD is attached
HEAD
branch1
3fcc6
branch2
a0123
e76fc
b16c3
58d77
ca122
branch3
HEAD is attached
branch1
3fcc6
HEAD
branch2
a0123
e76fc
b16c3
58d77
ca122
branch3
HEAD is attached
branch1
3fcc6
branch2
a0123
e76fc
b16c3
58d77
ca122
branch3
HEAD
HEAD is detached
branch1
3fcc6
HEAD
branch2
a0123
e76fc
b16c3
58d77
ca122
branch3


develop
hotfixes
master
Tag
1.2
Tag
1.2.1
Severe bug
fixed for
production:
hotfix 1.2.1

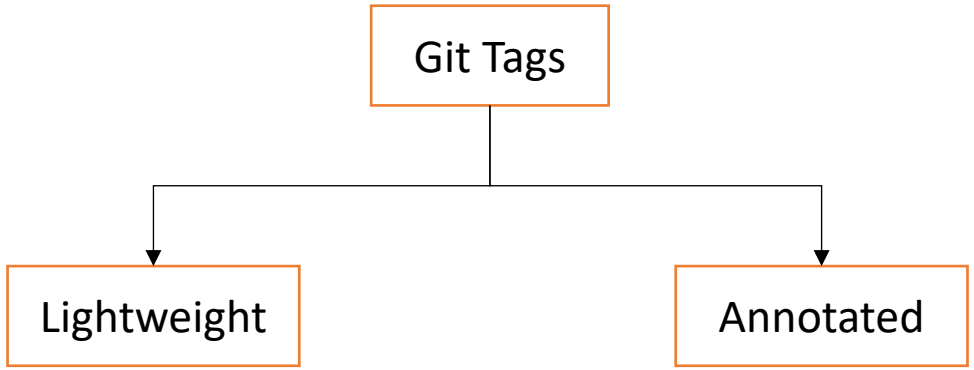
Git Tags

- **Branches** are named pointers to commits (refs)
- **HEAD** is a special pointer
  - Related with current active branch
  - Always point to the latest commit
- **Tagging** is generally used to capture a point in history that is used for a marked version release (i.e. v1.0.1). A tag is like a branch that doesn't change. Unlike branches, tags, after being created, have no further history of commits



Tags Operation

- Create tag
  - git tag <tag name> create a Lightweight tag (i.e. git tag v1.0)
  - git tag -a <tag name> -m <tag message> create annotated tag that have all the associated meta-data(like email, date, etc.). (i.e. git tag -a v1.0 -m “version 1.0”)
  - git tag -a <tag name> <commit SHA1 value> create annotated tag for an older commit (i.e. git tag -a v1.0 236e1ef5d755f6ea6894f8f46f36186190be7b92)
- List tags
  - git tag
- Delete tags: git tag -d <tag name> (i.e. git tag -d v1.0)




```
$ git tag v1.0.0
```

- ✓ Is stored in the .git/refs/tags

```
$ git tag –a v1.0.0 –m “version 1.0.0”
```

- ✓ Is stored in the .git/refs/tags
- ✓ Is also stored in the .git/objects
- ✓ Tag message
- ✓ Tag author and date

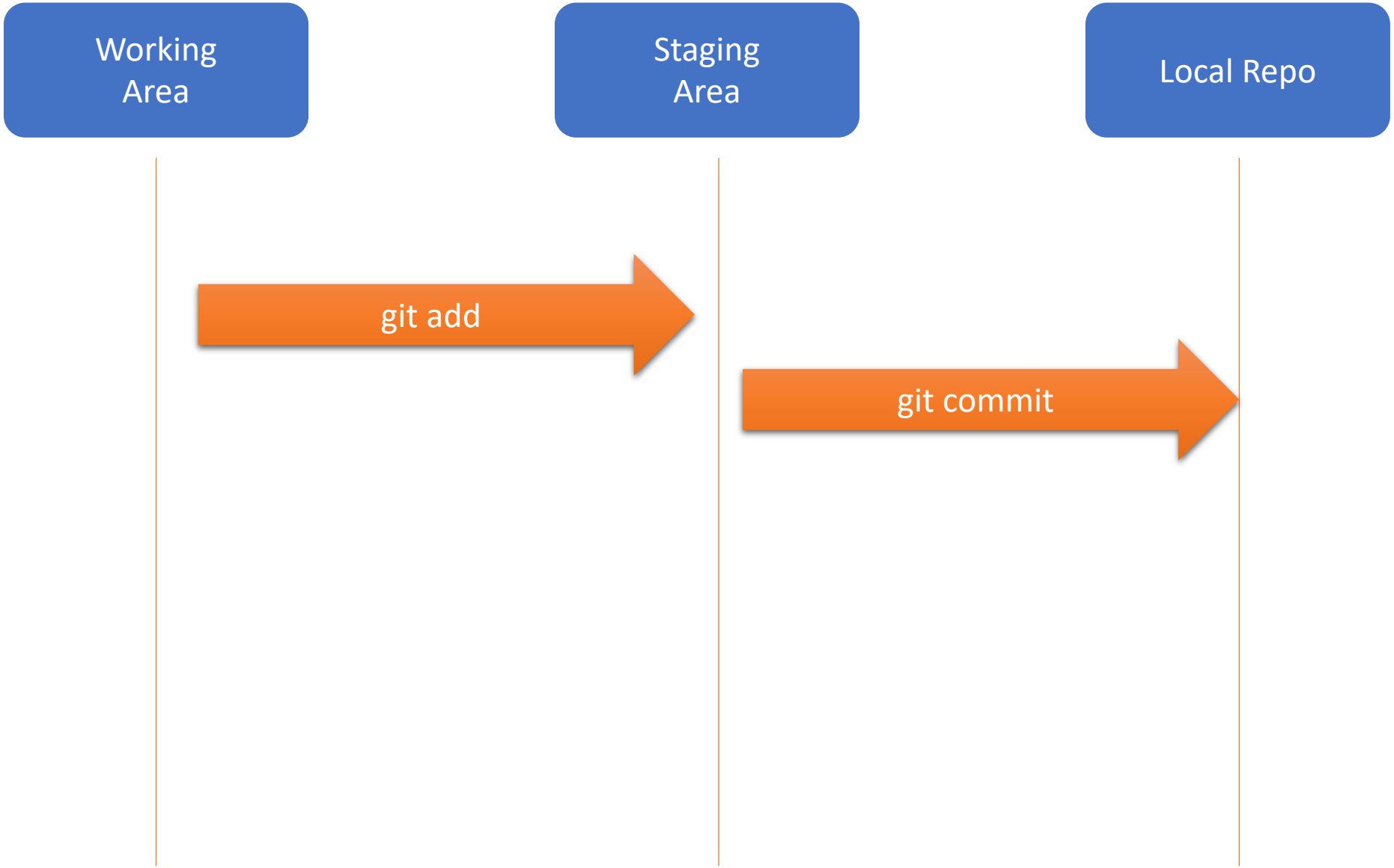
Tags Operation

- push tag
  - git push origin <tag name> push tag to remoteg (i.e. git push origin v1.0)
  - git push --tags push multiple tags simultaneously
- sync tags from remote
  - git fetch
- Delete tags:
  - git tag -d <tag name> (i.e. git tag -d v1.0)
  - git push –delete origin <tag name> delete remote git tag

Quick Summary

Git Commands

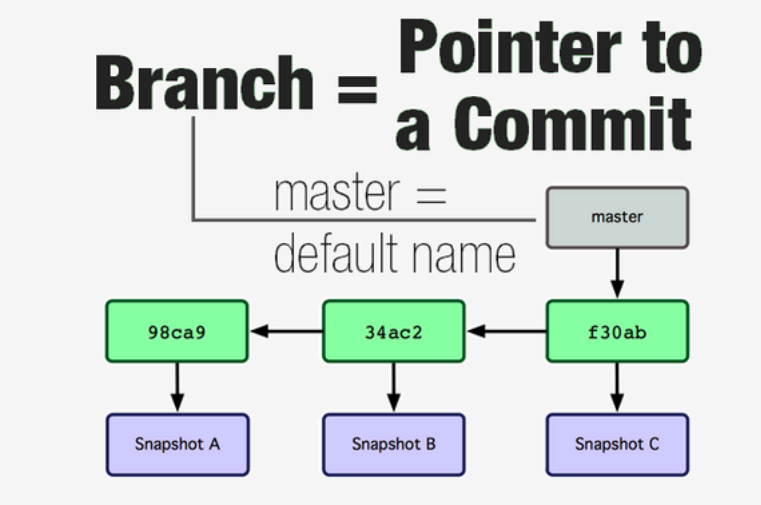
- git init (initialize an empty git repository)
- git status (show the working tree status)
- git add (add file contents to the index)
- git commit (Records the changes to repository)
- git log (show commit logs)
- git rm (Remove files from the working tree and from the index)
- git branch (list, create or delete branches)
- git checkout (change the current active branch)
- git merge (Join two or more development histories together)
- git tag (list, create or delete tags)

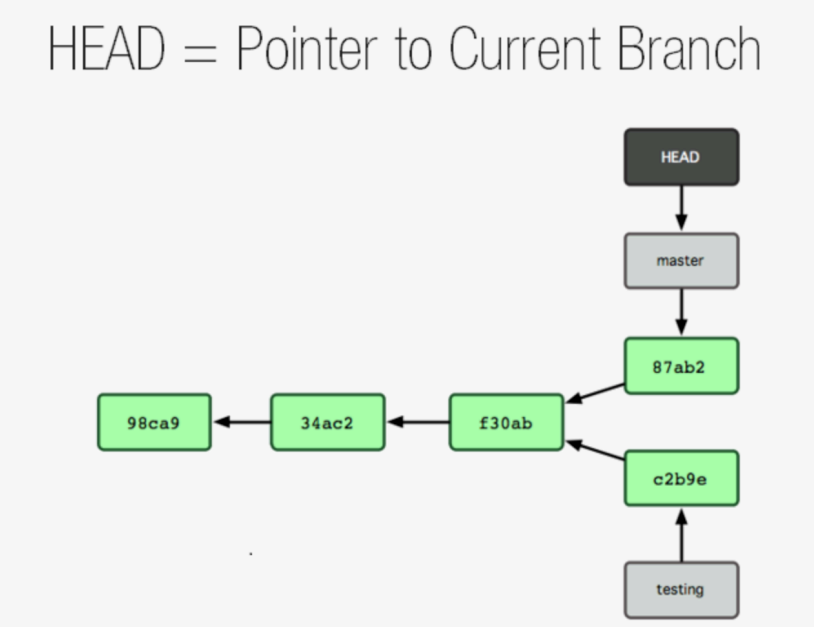

Working Area
Staging Area
Local Repo
git add
git commit

Git Objects

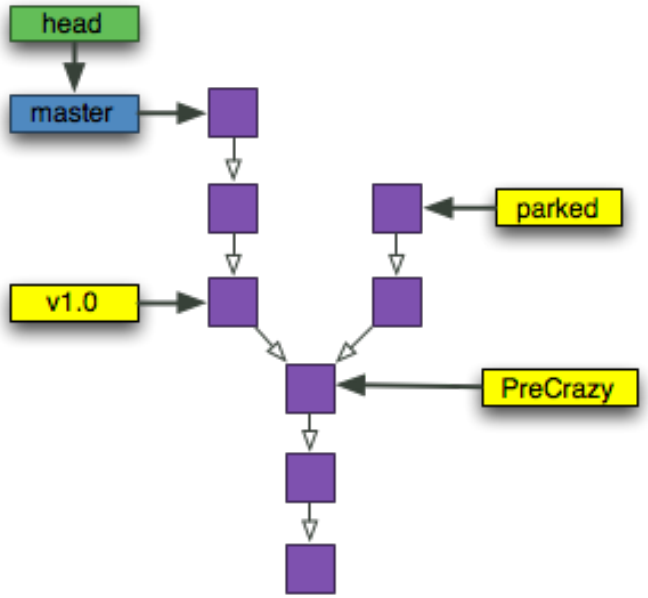
- Git is a content-addressable filesystem. Great. What does that mean? It means that at the core of Git is a simple key-value data store. What this means is that you can insert any kind of content into a Git repository, for which Git will hand you back a unique key you can use later to retrieve that content.
- Object types:
  - Blob - data (source code file)
  - Tree – Pointers to file name, content, other trees
  - Commit – Author, message, pointer to a tree
  - Tag: tag information(commit, tag name, email, user, date, ect.)

Git Branches

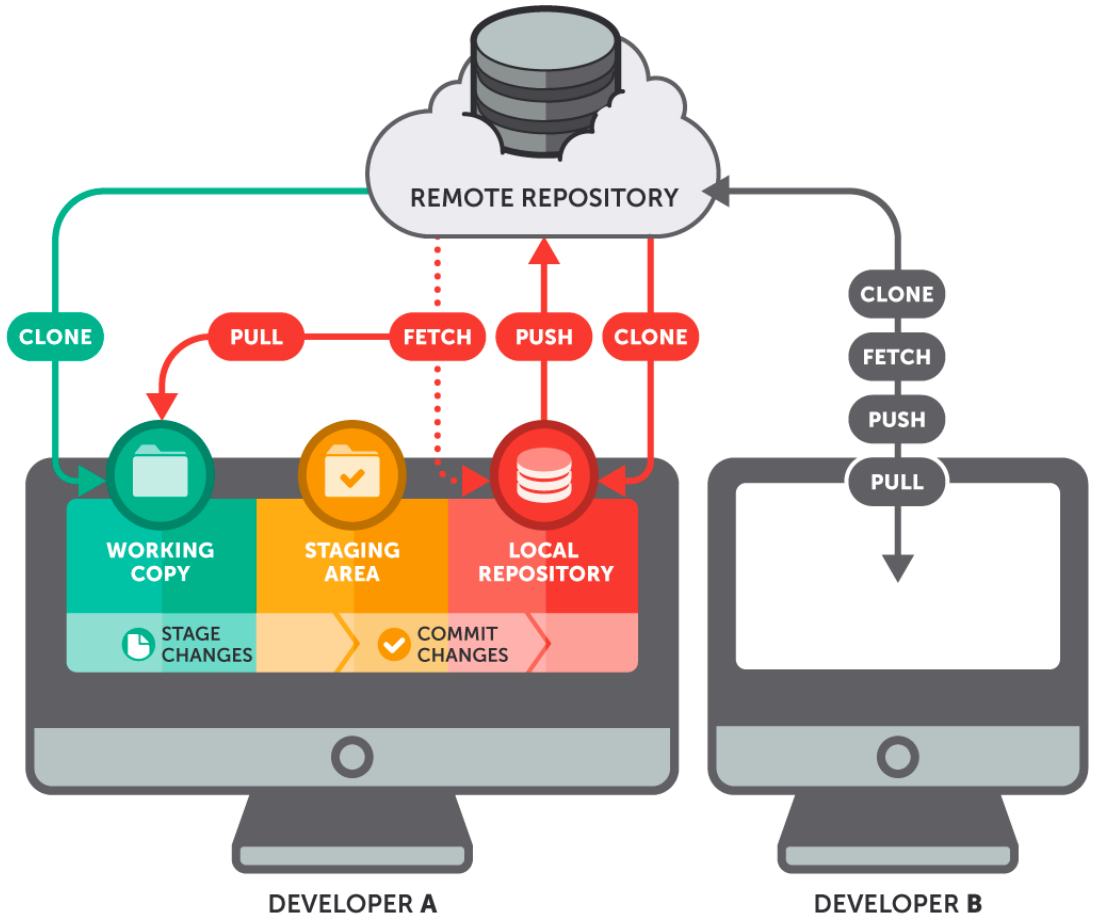




Git Tags



Git Remote

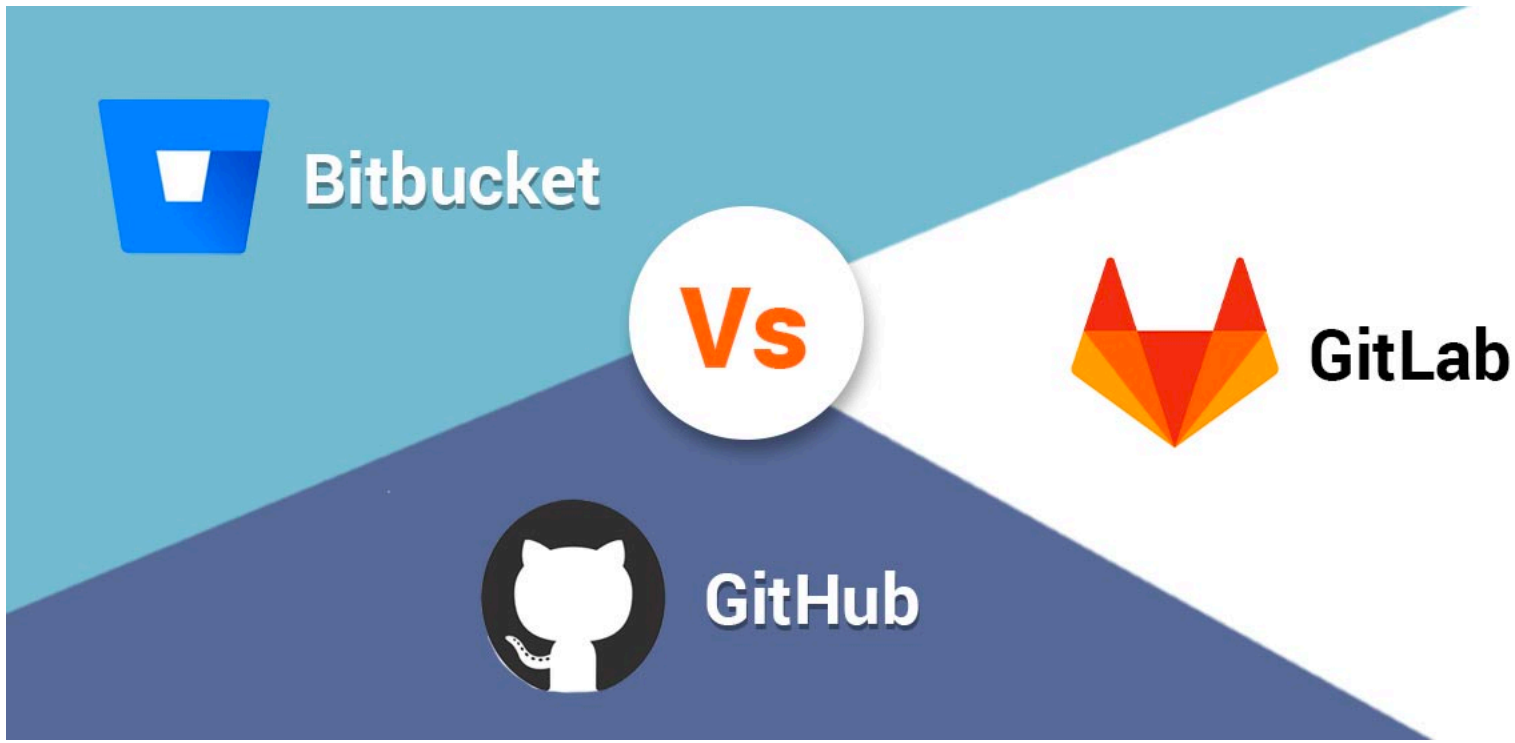

REMOTE REPOSITORY
CLONE
PULL
FETCH
PUSH
CLONE
CLONE
FETCH
PUSH
PULL
WORKING COPY
STAGING AREA
LOCAL REPOSITORY
STAGE CHANGES
COMMIT CHANGES
DEVELOPER A
DEVELOPER B

What is remote repository

- A remote in Git is a common repository that all team members use to exchange their changes. In most cases, such a remote repository is stored on a code hosting service like **GitHub** or on an internal server.
- In contrast to a *local* repository, a remote typically **does not provide a file tree** of the project's current state. Instead, it only consists of the **.git** versioning data.

What is GitHub

- While Git is a command line tool, GitHub provides a web-based graphical interface that works on top of Git. It can also be treated as a social platform to share knowledge and work.
- It also provides access control and several collaboration features, such as issues, wikis and basic task management tools.


Bitbucket
Vs
GitLab
GitHub

Git remote

- While Git is a command line tool, GitHub provides a web-based graphical interface that works on top of Git. It can also be treated as a social platform to share knowledge and work.
- It also provides access control and several collaboration features, such as issues, wikis and basic task management tools.

Git remote command

- git clone <repository URL> clone a remote repository to local disk

git clone https://github.com/git2022/test-git

git clone https://github.com/git2022/test-git my-test-git

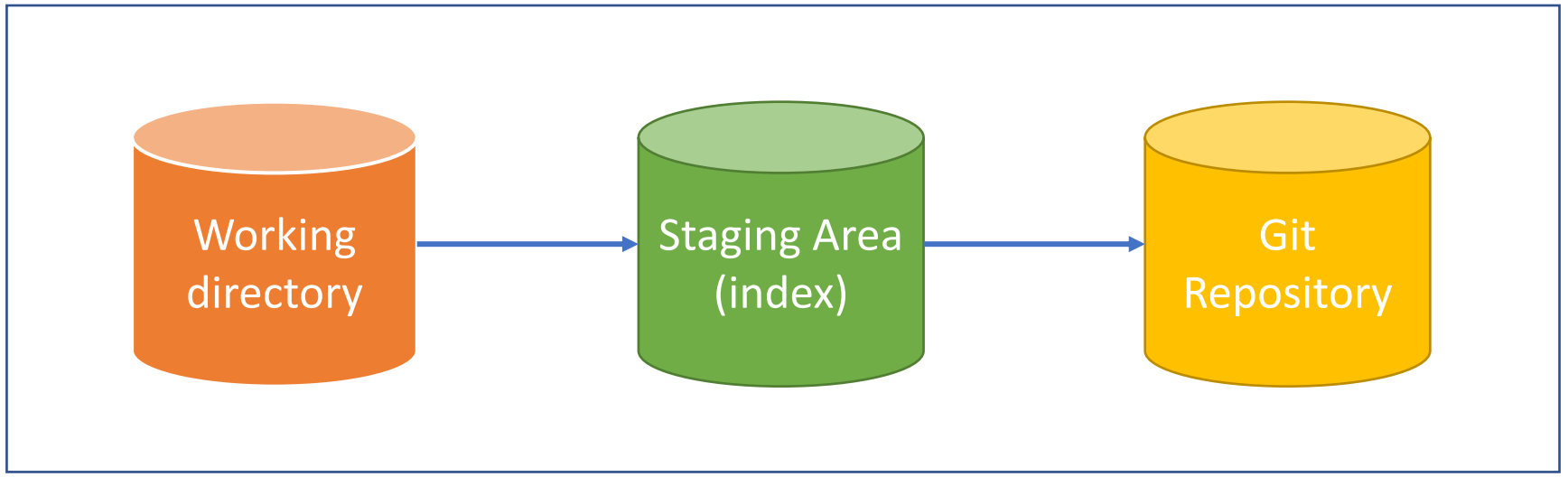
Git remote command

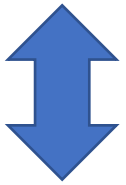
- git clone <repository URL> clone a remote repository to local disk
- git push origin <branch_name> push local branch <branch_name> to remote branch dev, if dev branch does not exist remotely, it would be created. (--delete to delete a remote branch)
- git branch -a list all branches both local and remote
- git fetch download objects and refs from remote repository
- git pull - Fetch from remote and integrate with local branch
- git push origin <tag_name> push local tag to github
- git push origin –delete <tag_name> to delete github tags

Git Local & Git Remote

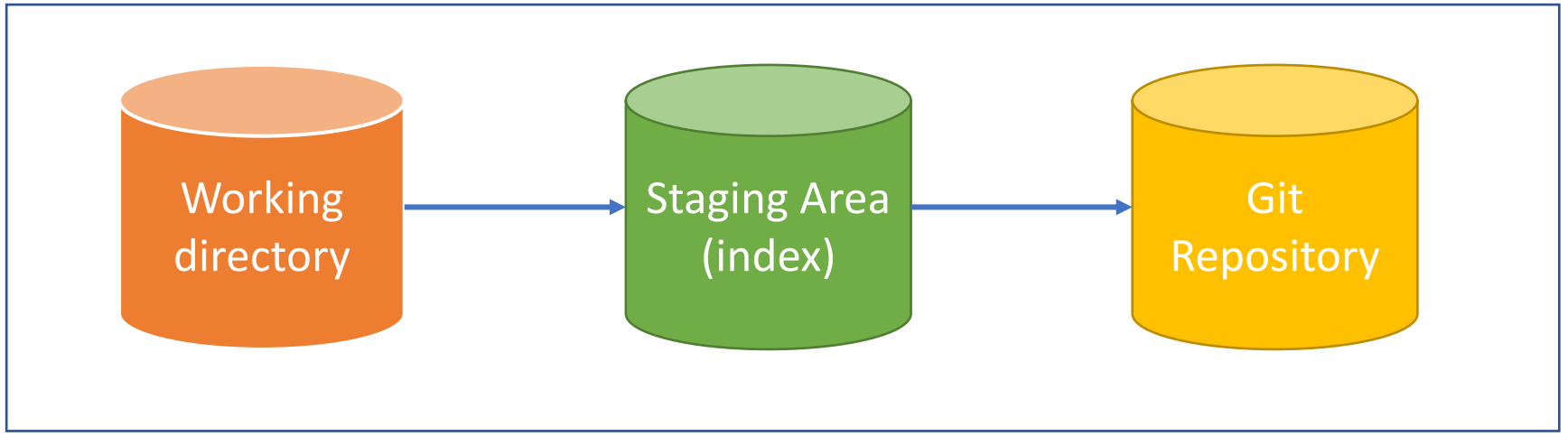
Origin

Remote Repository



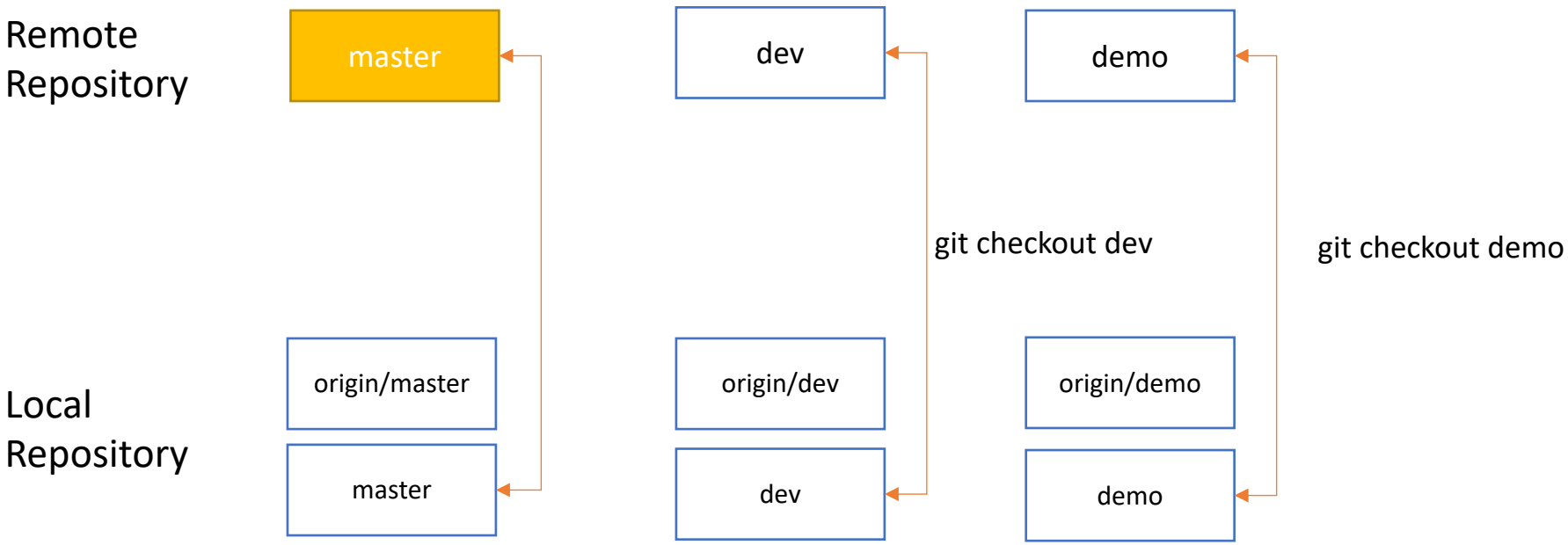


Local Repository



1. git clone https://github.com/git2022/git-demo
2. git remote add origin https://github.com/git2022/git-demo.git

Tracking Branches



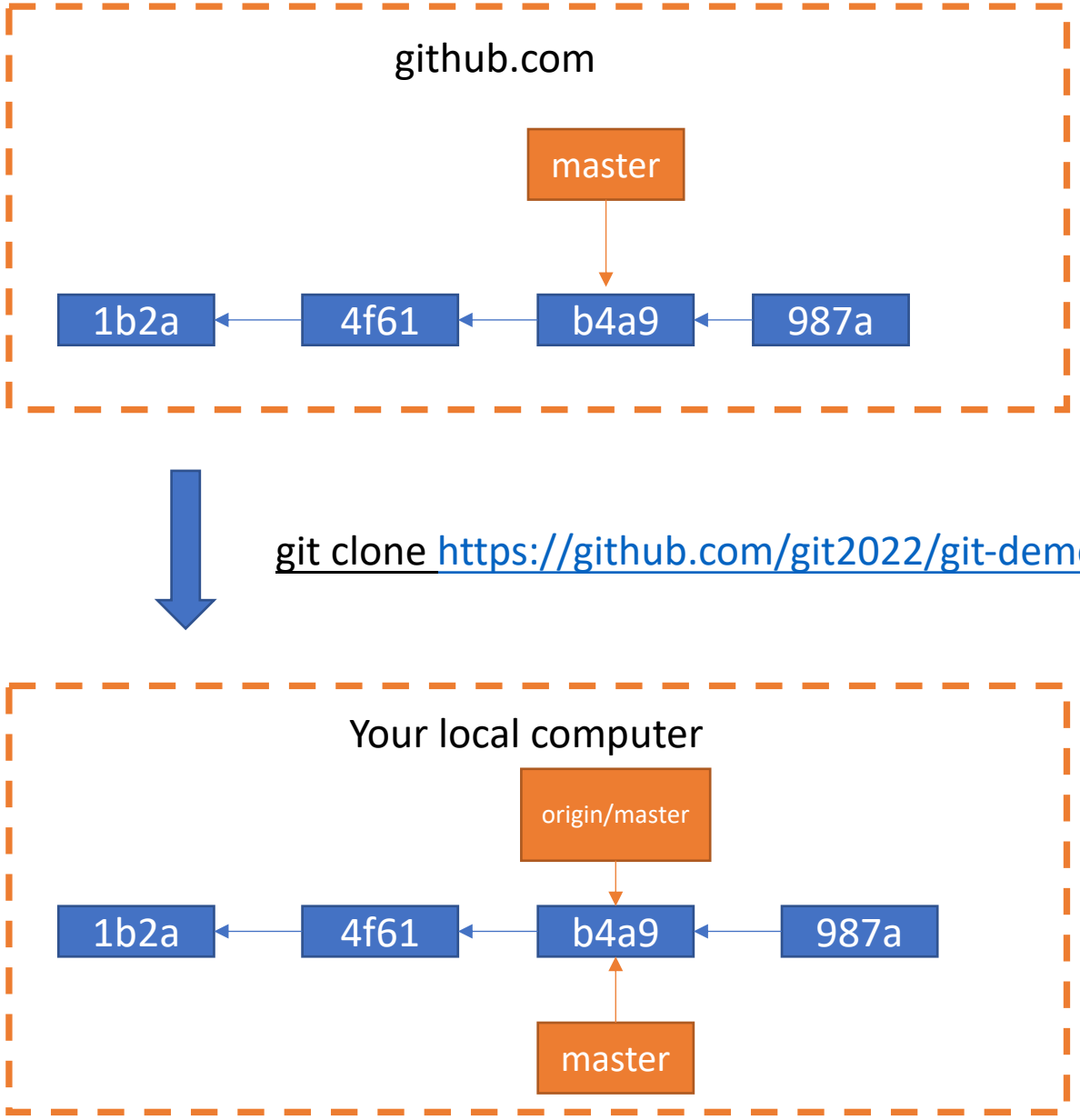
Git remote

- git remote add/remove <name> <url> add a remote
- git remote get remote name, origin be default, -v (--verbose) display more information like remote URL
- git remote show origin show remote origin tracking information, need connect to internet.
- git remote prune origin delete local origin branch which does not existing in remote

Git remote branch

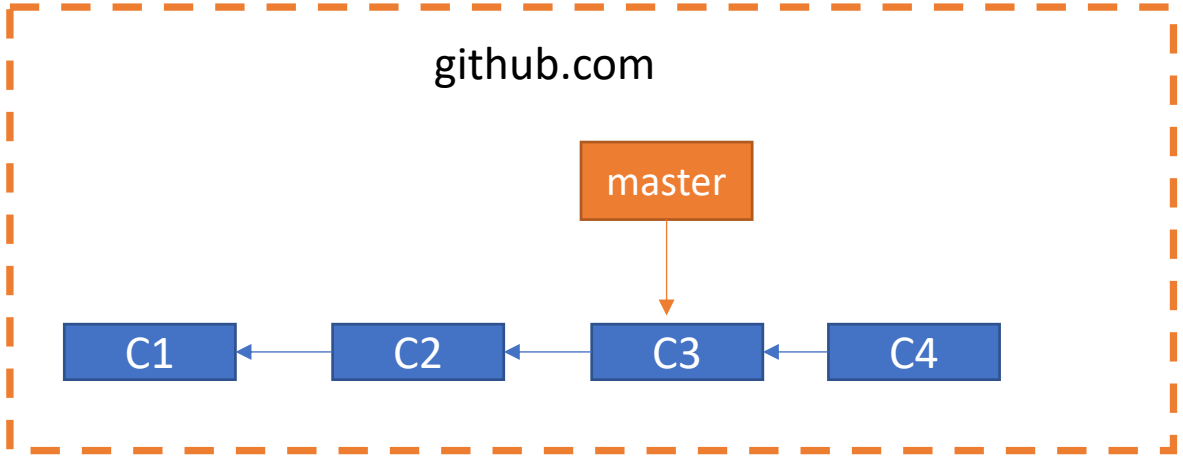
- Branch Creation:
  - Github web page
  - git push –u origin <branch_name> push local branch <branch_name> to remote branch dev, if <branch_name> branch does not exist remotely, it would be created.
- Branch Deletion
  - GitHub web page
  - git branch –d <branch_name> delete local branch
  - git push origin -d <branch_name> delete remote branch
- List all Local and Remote Branch
  - git branch -a list all branches both local and remote
  - git branch –r list all remote origin branches
  - git branch –vv list all local branches tracking information
- Sync remote branches
  - git fetch download objects and refs from remote repository --prune Before fetching, remove any remote-tracking references that no longer exist on the remote
  - git pull fetch and merge origin to local tracking branch

git fetch & git pull



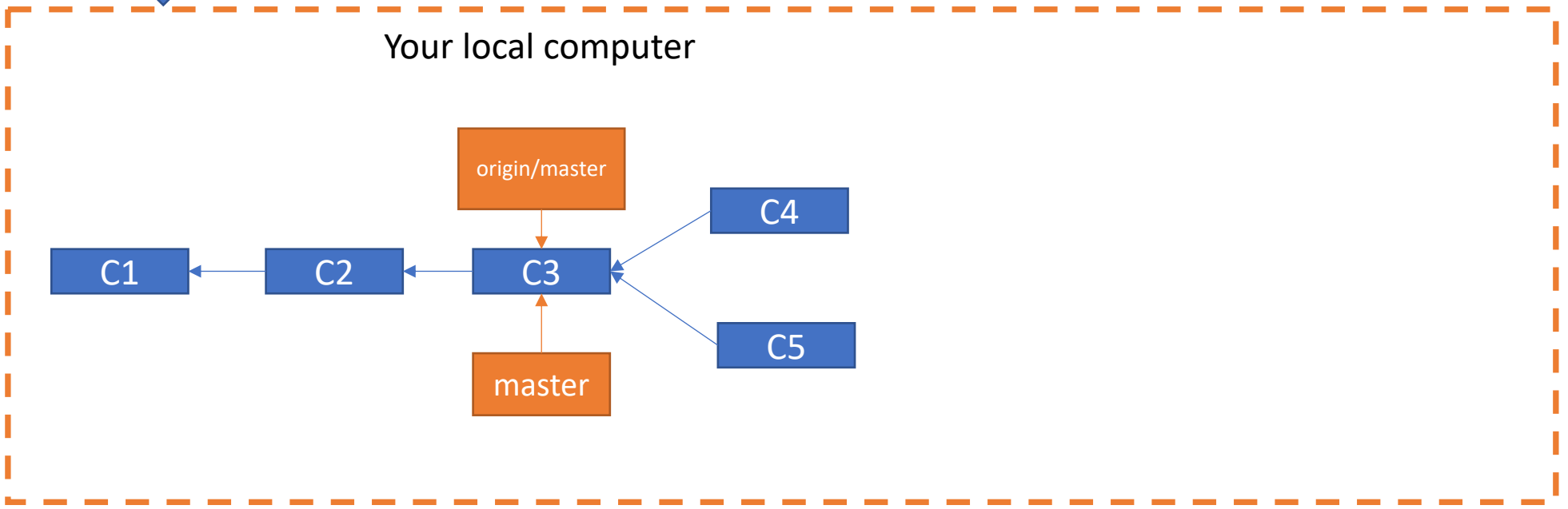
1. Someone pushed a new commit to github
2. Use git fetch to sync remote master
3. git merge origin/master

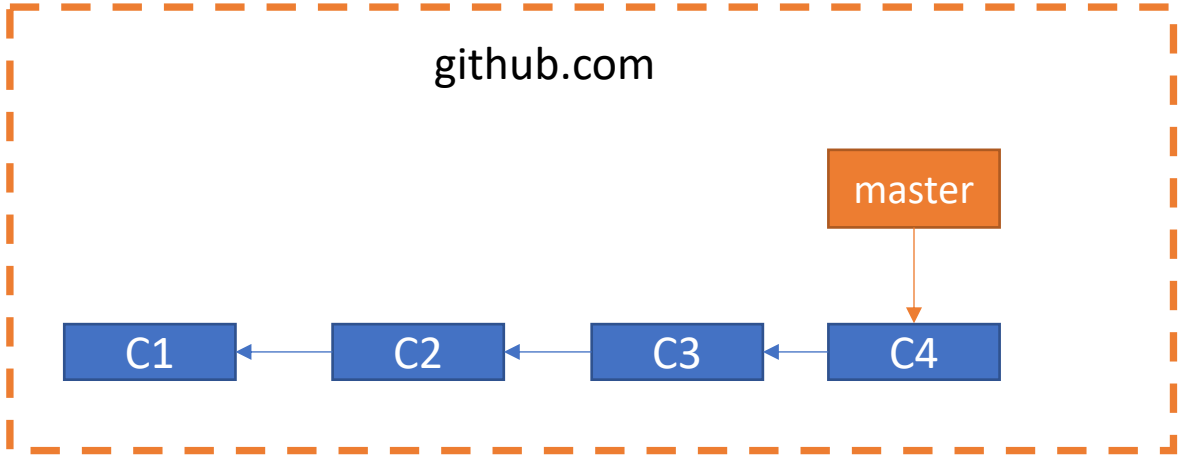
Step 2 and 3 can be done through Just one command: git pull



1. Someone pushed a new commit to github
2. Create a new commit on local master
3. git fetch
4. git merge origin/master

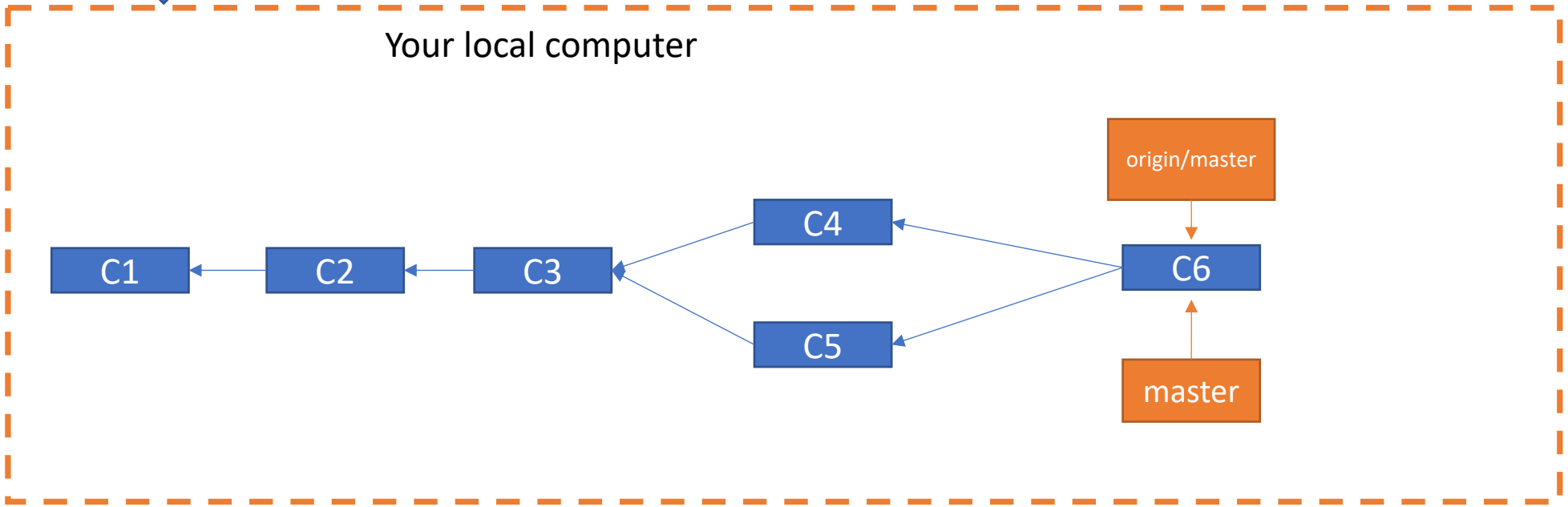
git clone git@github.com:git2022/test-git.git

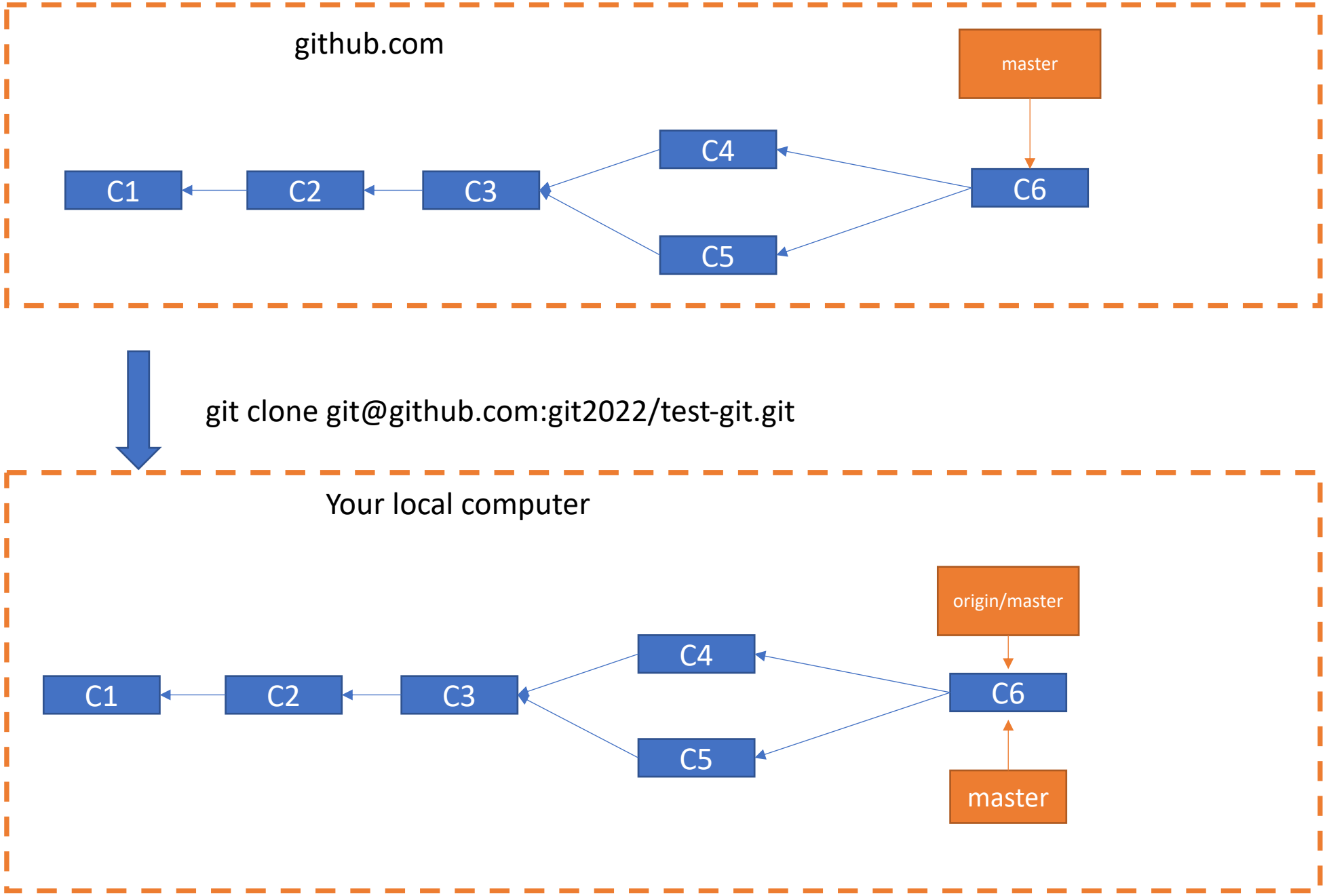




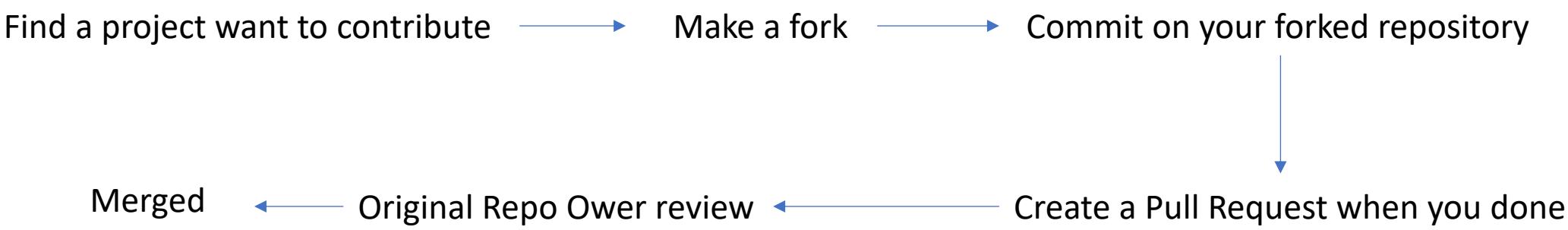
1. Someone pushed a new commit to github
2. Create a new commit on local master
3. git fetch
4. git merge origin/master

git clone git@github.com:git2022/test-git.git




github.com
master
C4
C1
C2
C3
C6
C5
git clone git@github.com:git2022/test-git.git
Your local computer
origin/master
C4
C1
C2
C3
C6
C5
master

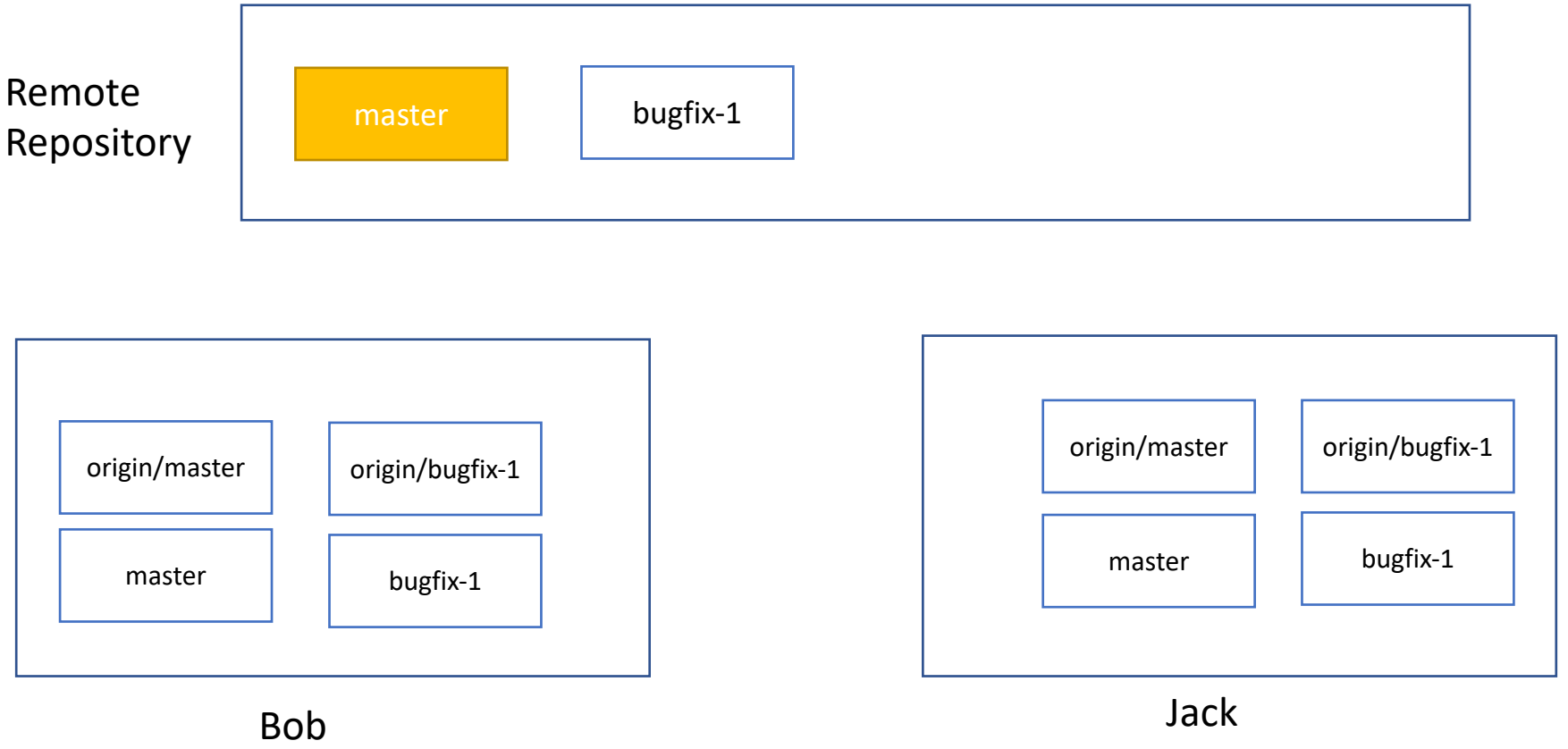
GitHub Pull Reqeust

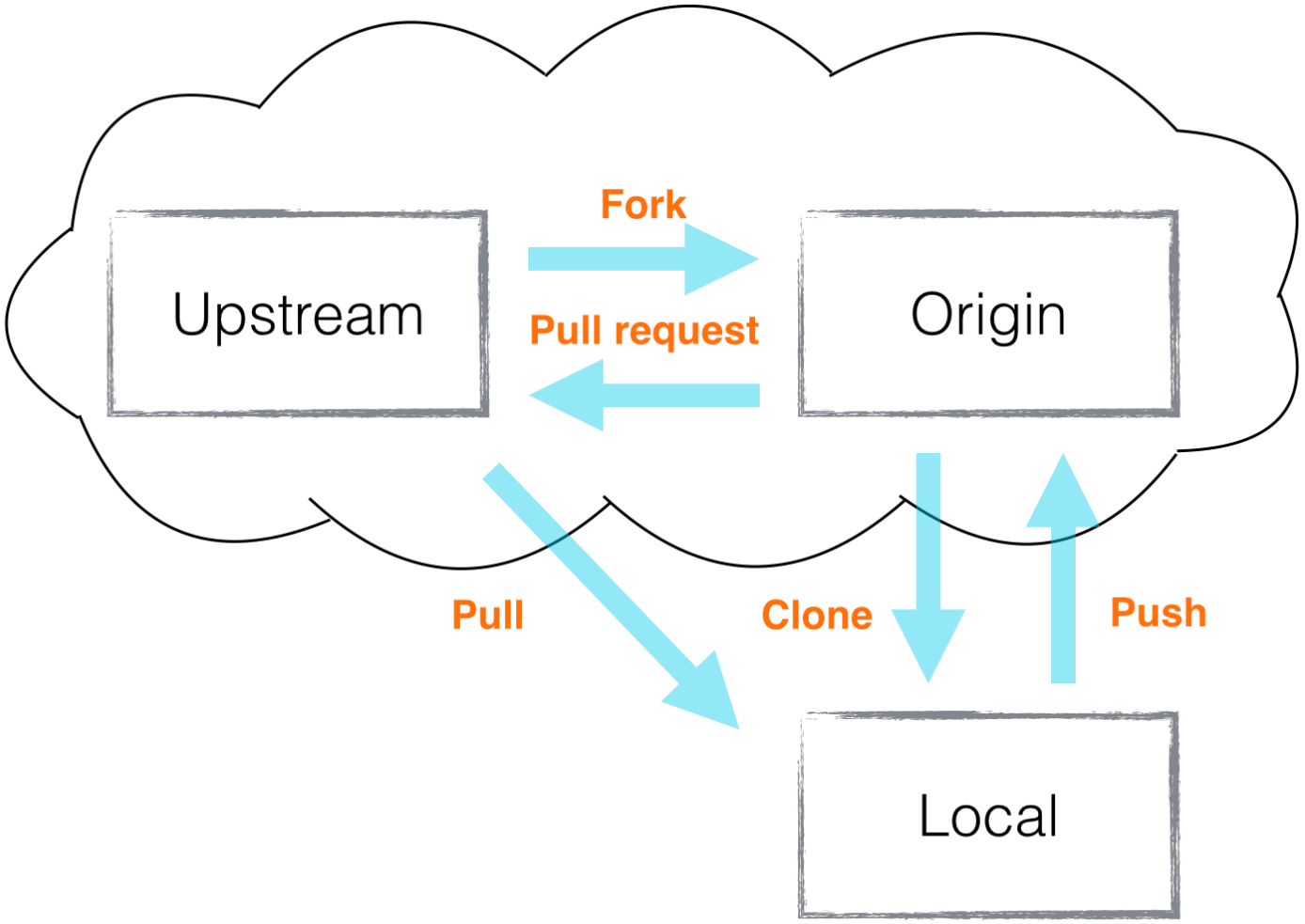


Pull Request

Can you push code to a repository which does not belong to you on GitHub?

Pull Request




Upstream
Fork
Pull request
Origin
Pull
Clone
Push
Local

Sync with upstream

- Configure a remote for a fork
  - git remote add upstream https://github.com/*ORIGINAL_OWNER*/*ORIGINAL_REPOSITORY*.git
  - git remote –v to list current remote
- Syncing a fork
  - git fetch upstream
  - git merge upstream/master
  - git push origin master

git pull upstream master

Git Best Practice

Best Practice最佳实践第一条

拥抱命令行，远离IDE图形界面

Best Practice最佳实践第二条

写好每一条commit message

Best Practice最佳实践第三条

用好.gitignore文件

Best Practice最佳实践第四条

基于分支或者fork的开发模式

Best Practice最佳实践第五条

用好release分支和tag

比较常用的一种tag版本规划

- A.B.C
  - A: 大版本, 大的feature更新
  - B: 小版本，小的feature更新
  - C: bug fix 版本，只修复bug，无任何新feature加入
- 发布了2.9，那么一般我们要去下载2.9版本的最新版本，比如2.9.7

Best Practice最佳实践第六条

如何修改最后一次commit

修改最后一次Commit

- 提交的修改有**问题**（有**错误**，或者是工作没有完成）
- Commit message写**错**了

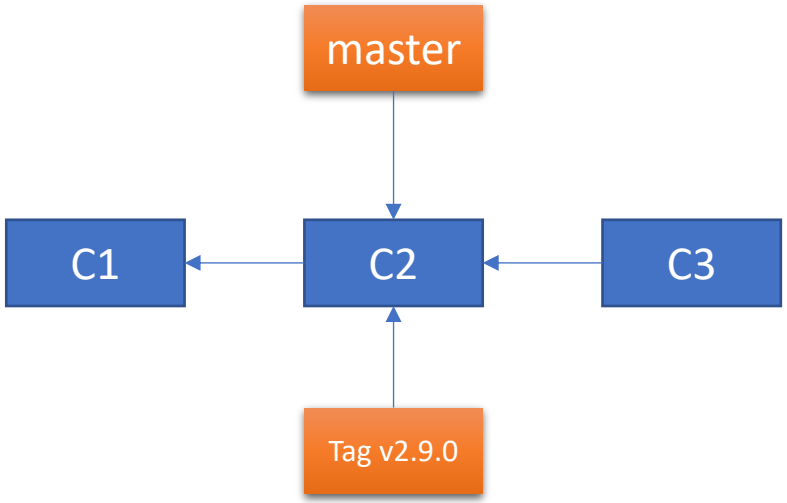
Best Practice最佳实践第七条

如何修改最后N次commit

Git reset的三种模式

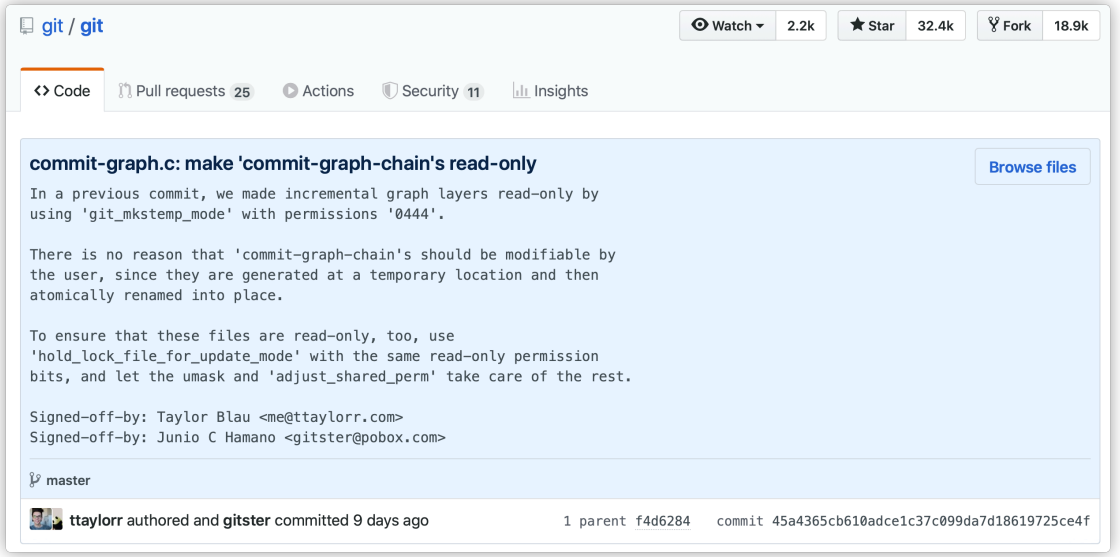
- git reset 是恢复到某次 commit 记录的命令， 有三种常用模式：
  - mixed(默认模式): git reset <commit-id>
  - Soft: git reset --soft <commit-id>
  - Hard: git reset --hard <commit-id>
- Git reset 和git commit --amend 一样都会修改git的commit历史

- watch -n 1 -d find .


master
C1
C2
C3
Tag v2.9.0

写好commit message的几条原则

1. 区分subject和body，用一个空行隔开
2. Subject一般不超过50个字符
3. Subject首字母大写
4. Subject结尾不需要用句号句点结尾
5. Body每一行的长度控制在72个字符
6. 用body来详细解释这个commit做了什么



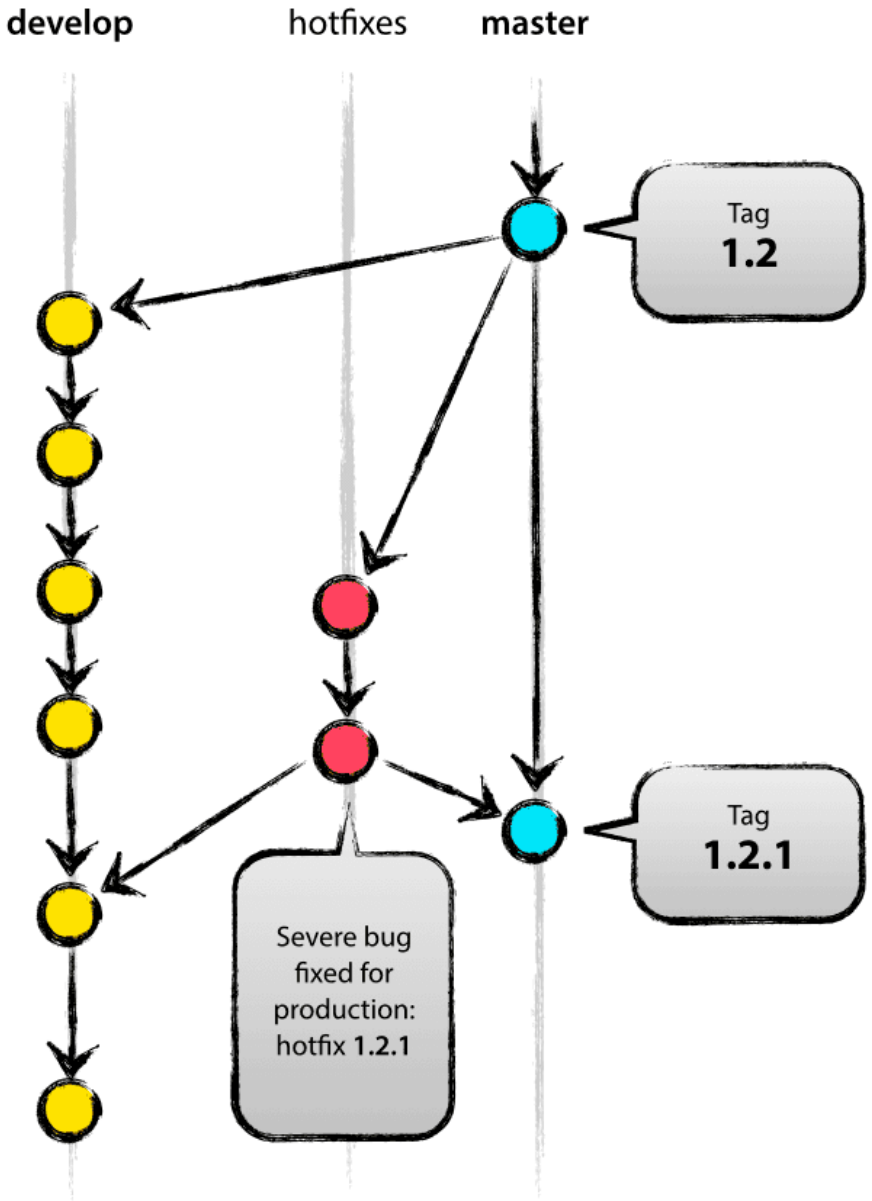
Imperative mood just means “spoken or written as if giving a command or instruction”. A few examples:

- Clean your room
- Close the door
- Take out the trash

Subject Like

- Fixed bug with Y
- Changing behavior of X

Branch Based Development



Good

```
92fb72f (HEAD -> bugfix) fix bug
```

Bad

```
d40325d (HEAD -> bugfix) finally fixed
da9e088 sorry, made a typo
54defc2 sorry, made a mistake
92fb72f fix bug
```

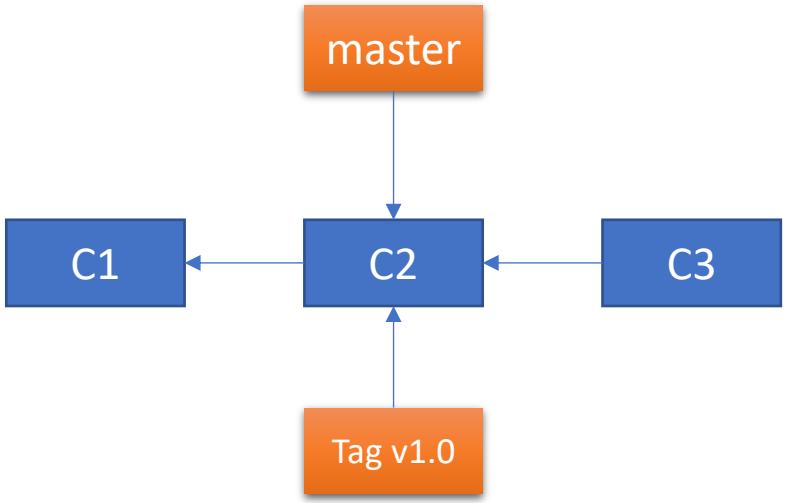
git commit --amend

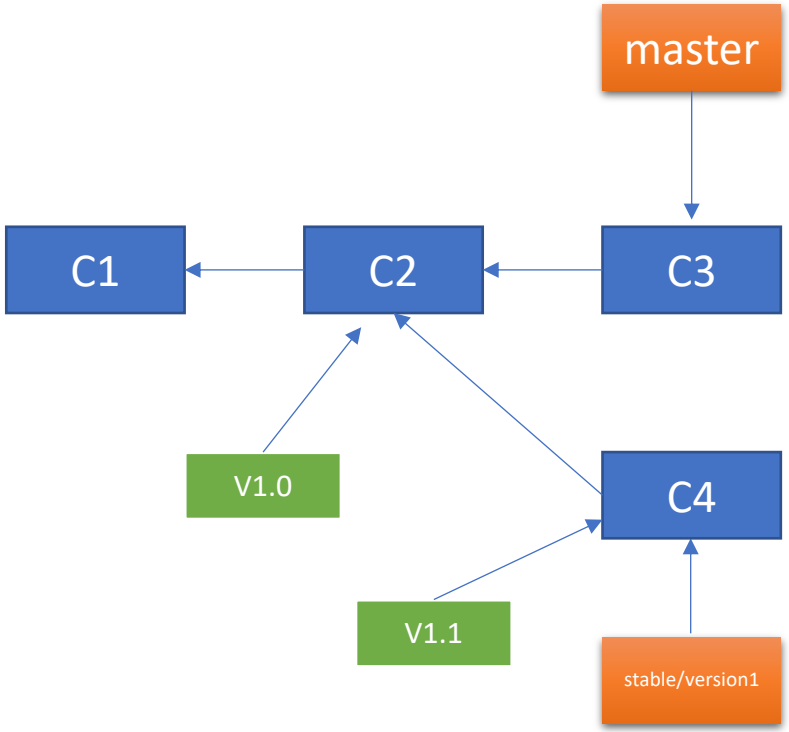
If you use “git commit --amend” to change the last commit which already pushed to GitHub
You need to use **git push origin master --force** to push the current change to GitHub, it will fail
If you only use **git push origin master**

If you use “git commit --amend” to change the last commit which already pushed to GitHub
You need to use git push origin master --force to push the current change to GitHub, it will fail
If you only use git push origin master

Main release

V1.x
V2.x
V3.x




master
C1
C2
C3
V1.0
C4
V1.1
stable/version1