GIT & GITHUB

Trainer **Nguyễn Văn Thọ**

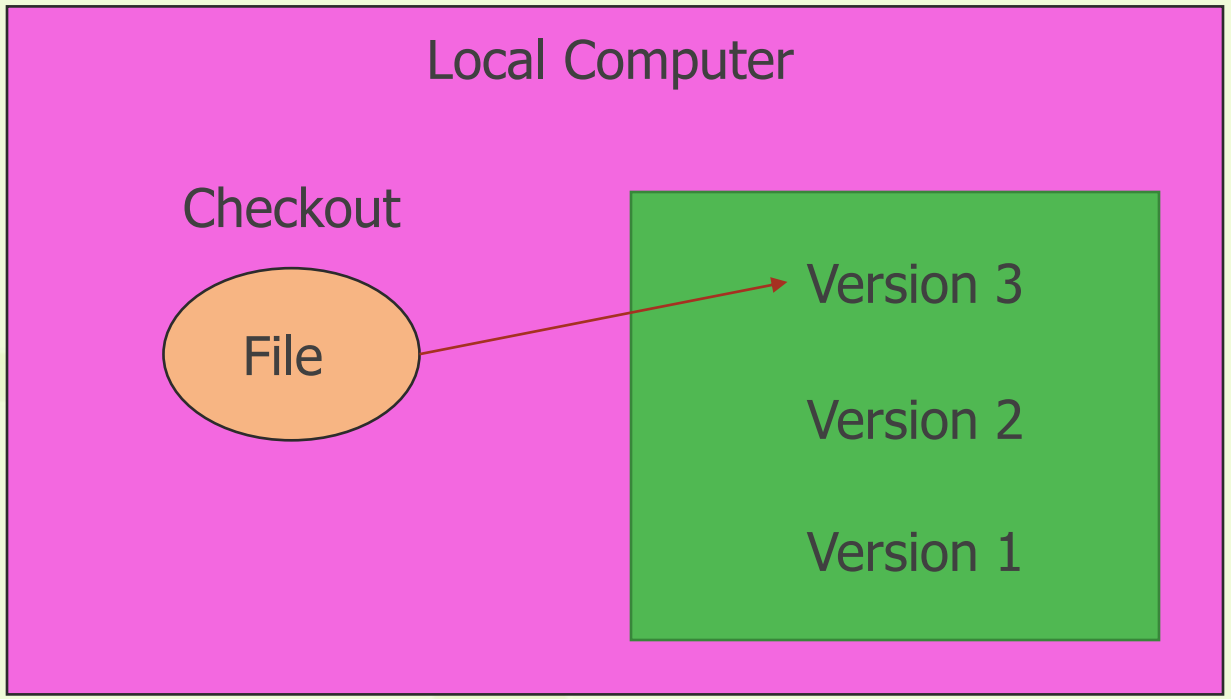
18CDT1 Khoa Cơ Khí - BKDN

Introduce about Version Control System (VCS)

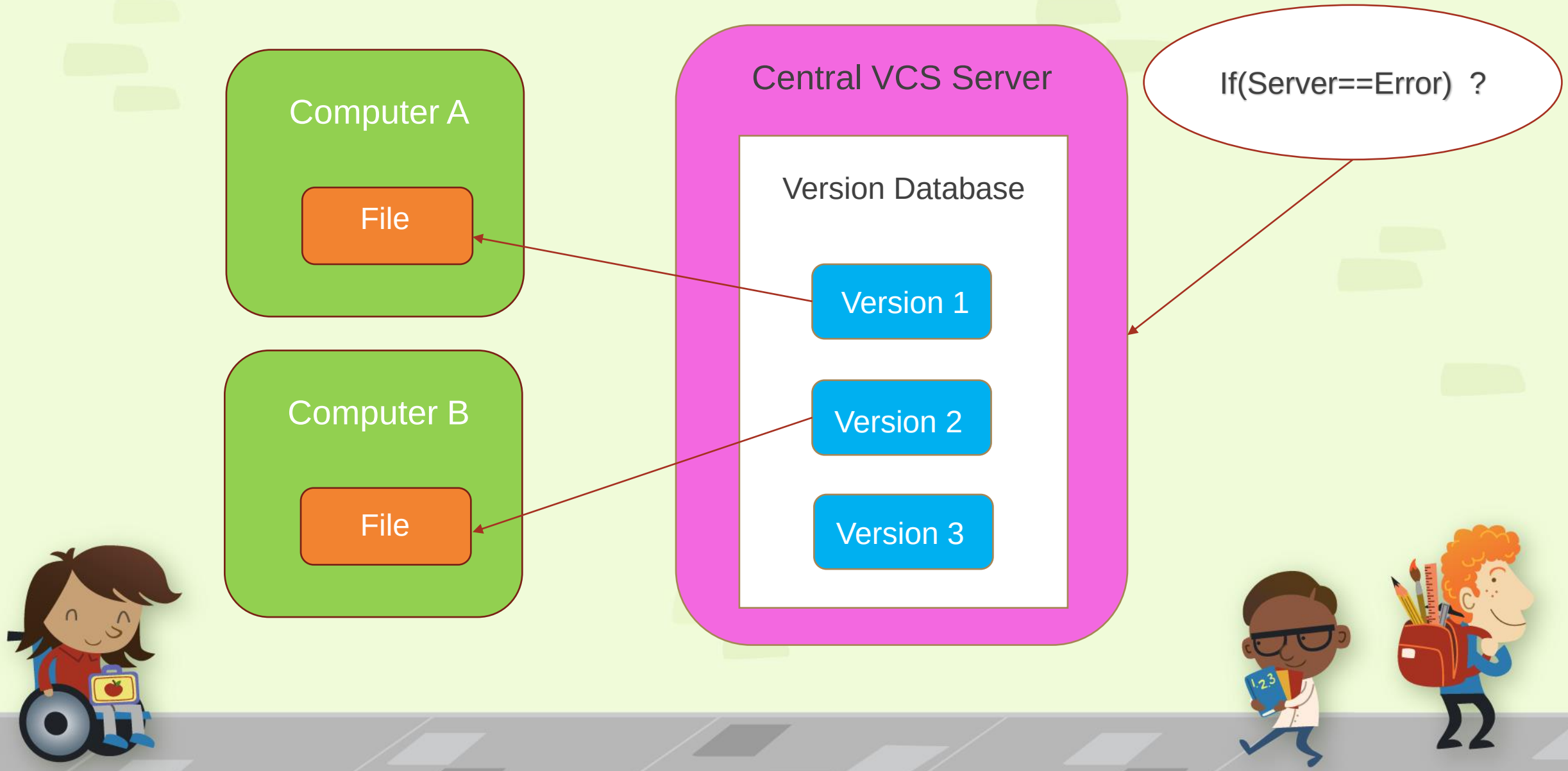
- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when , and more ☺)))
- Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.

Local Version Control Systems

- VCS that had a simple database that kept all the changes to files under reversion control

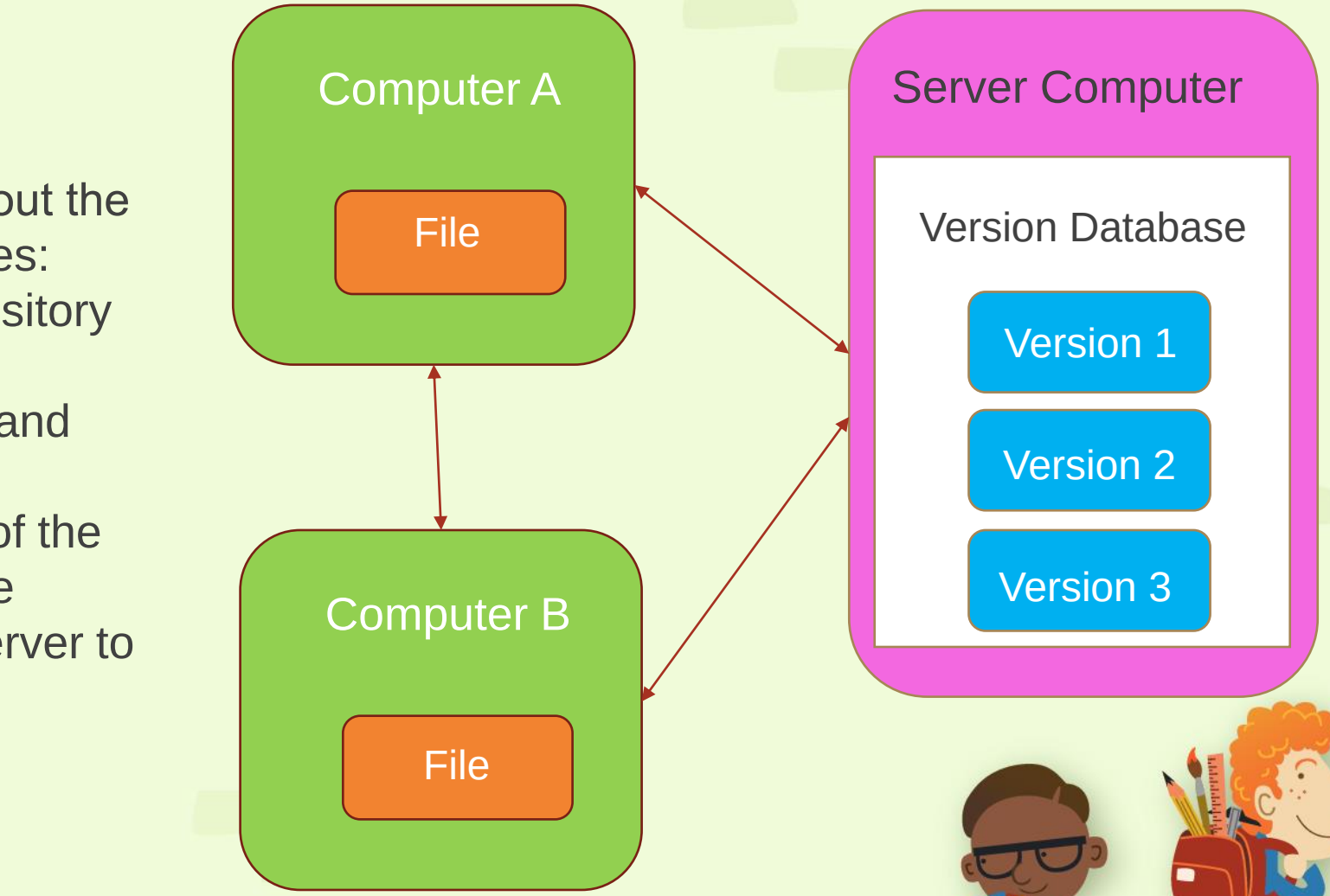


Centralized Version Control Systems



Distributed Version Control Systems

- Such Git
- Clients don’t just check out the latest snapshot of the files: they fully mirror the repository
- Thus if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it



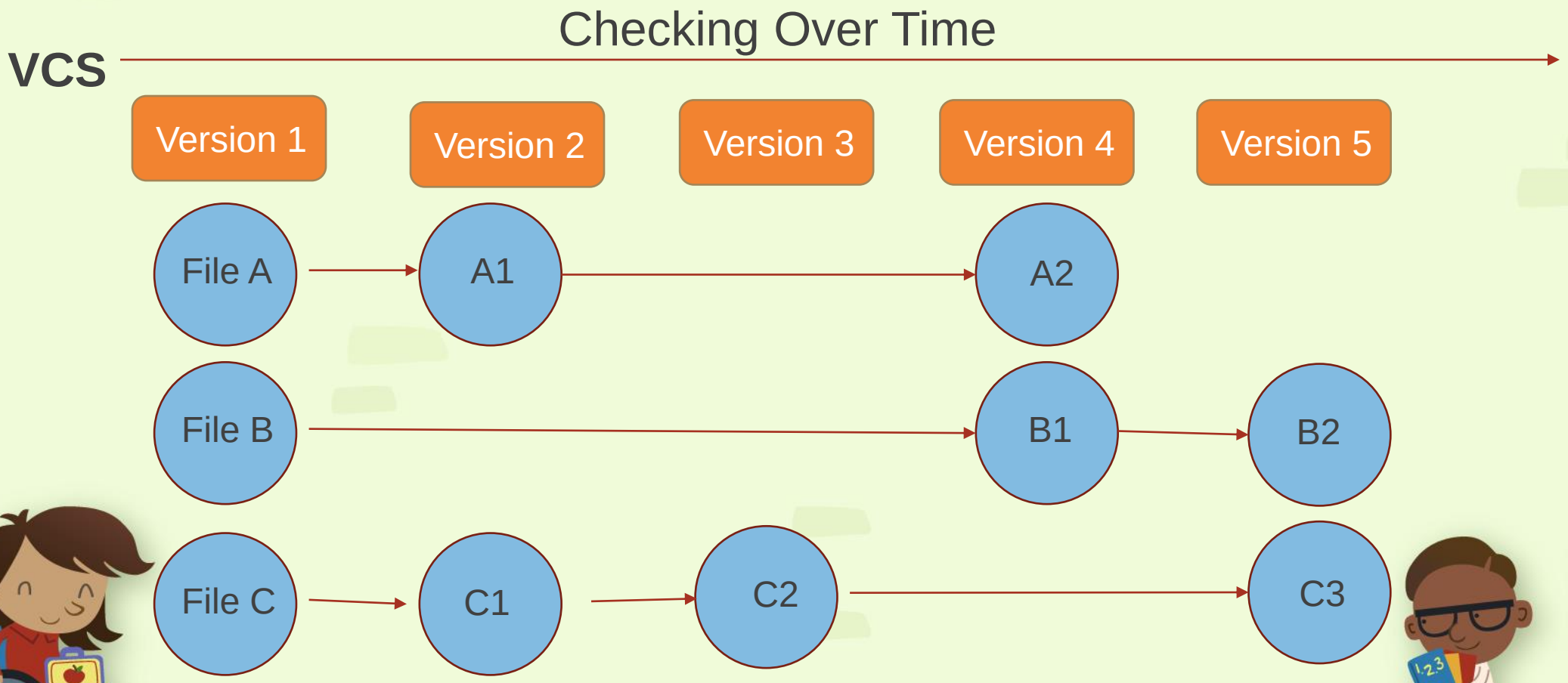
Introduce about GIT

- In 2002, Linux project began using a proprietary called BitKeeper.
- In 2005, the relationship go down.
- => Git

1. Snapshots
2. Nearly Every Operation Is Local
3. Git Has Integrity
4. Git Generally Only Adds Data
5. The Three States

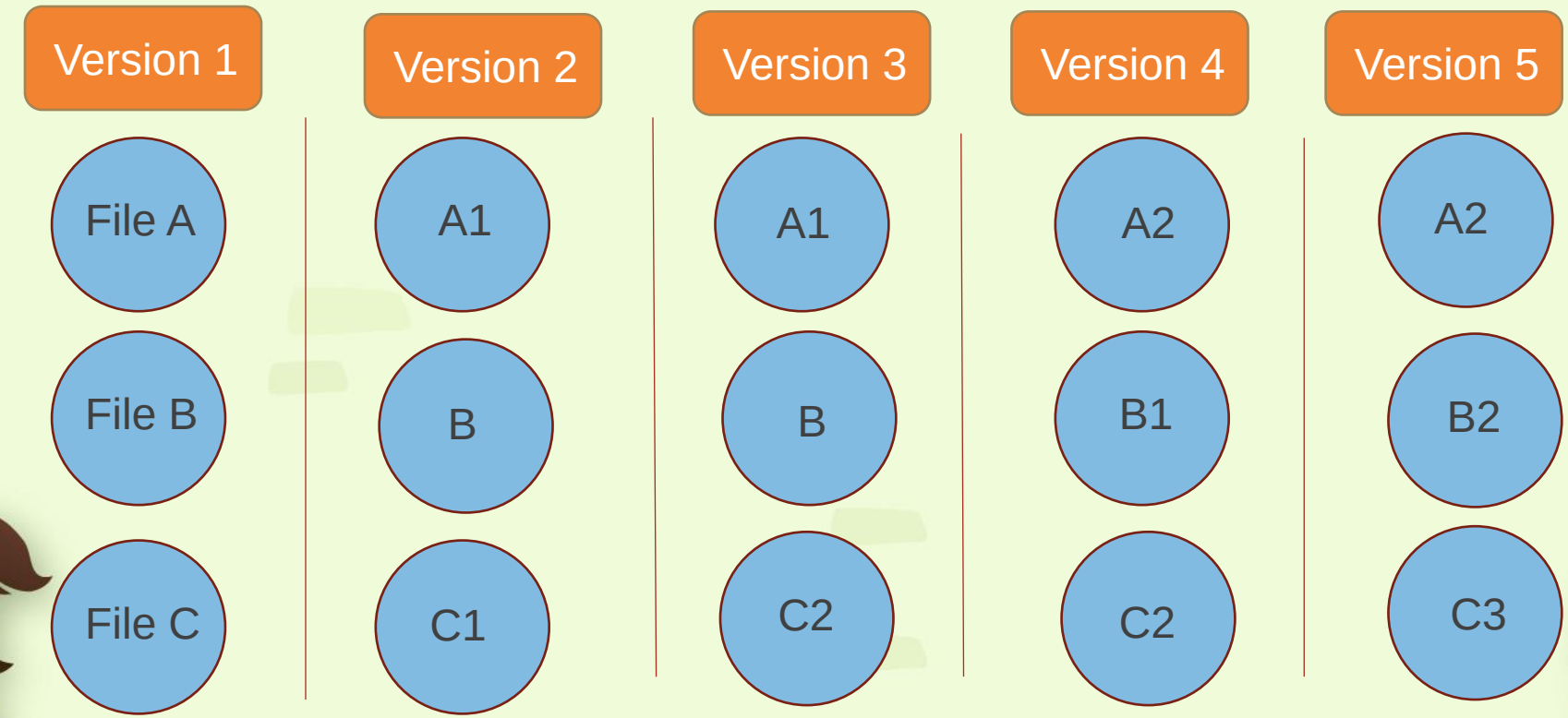
Snapshots

+ The major difference between Git and any VCS



Snapshots

+ Git think of its data like a set of snapshots of a miniature filesystem



Nearly Every Operation Is Local

1. Most operations on Git only need local files and resources to operate – generally no information is needed from another computer on your network.

Git Has Integrity

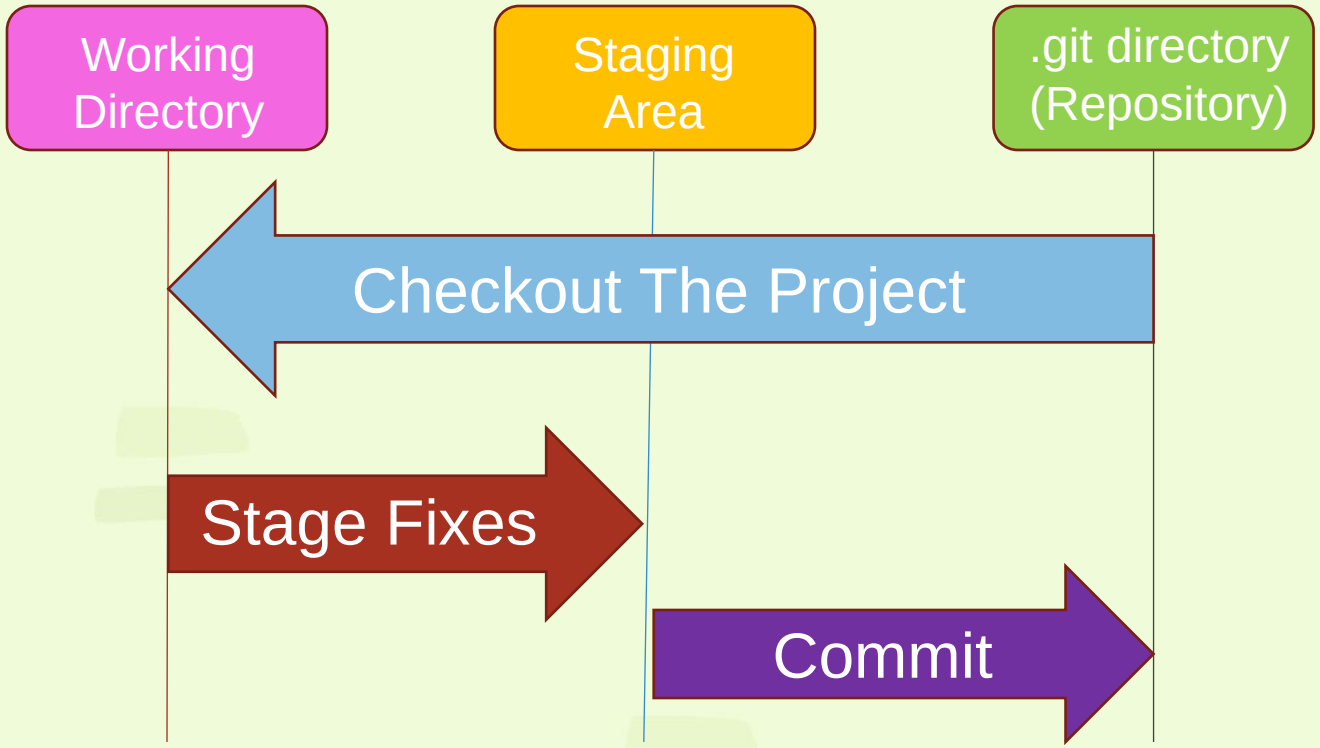
1. Everything in Git is check-summed before it is stored and is then referred to by that check-sum
2. You can't lose information in transit or get file corruption without Git being able to detect it.
3. The mechanism that Git uses for this check-summing is called a SHA-1 hash. This is a 40-character string composted of hexadecimal characters (0-9 and a-f) and calculated based on the contents of a file or directory structure in Git

Git Generally Only Adds Data

1. When you do actions in Git, nearly all of them only add data to the Git database
2. After you commit a snapshot into Git, it is very different to lost , especially if you regularly push your database to another repository

The Three States

1. Git has three main states that your files can reside in : committed, modified, staged.



Git Setting

1. Your Identity
    1. Git config –global user.name “VanTho15”
    2. Git config –global user.email nguyenvanthochutsinivesoss15@gmail.com
    3. git config --list
2. Help
    1. Git “Lenh” help

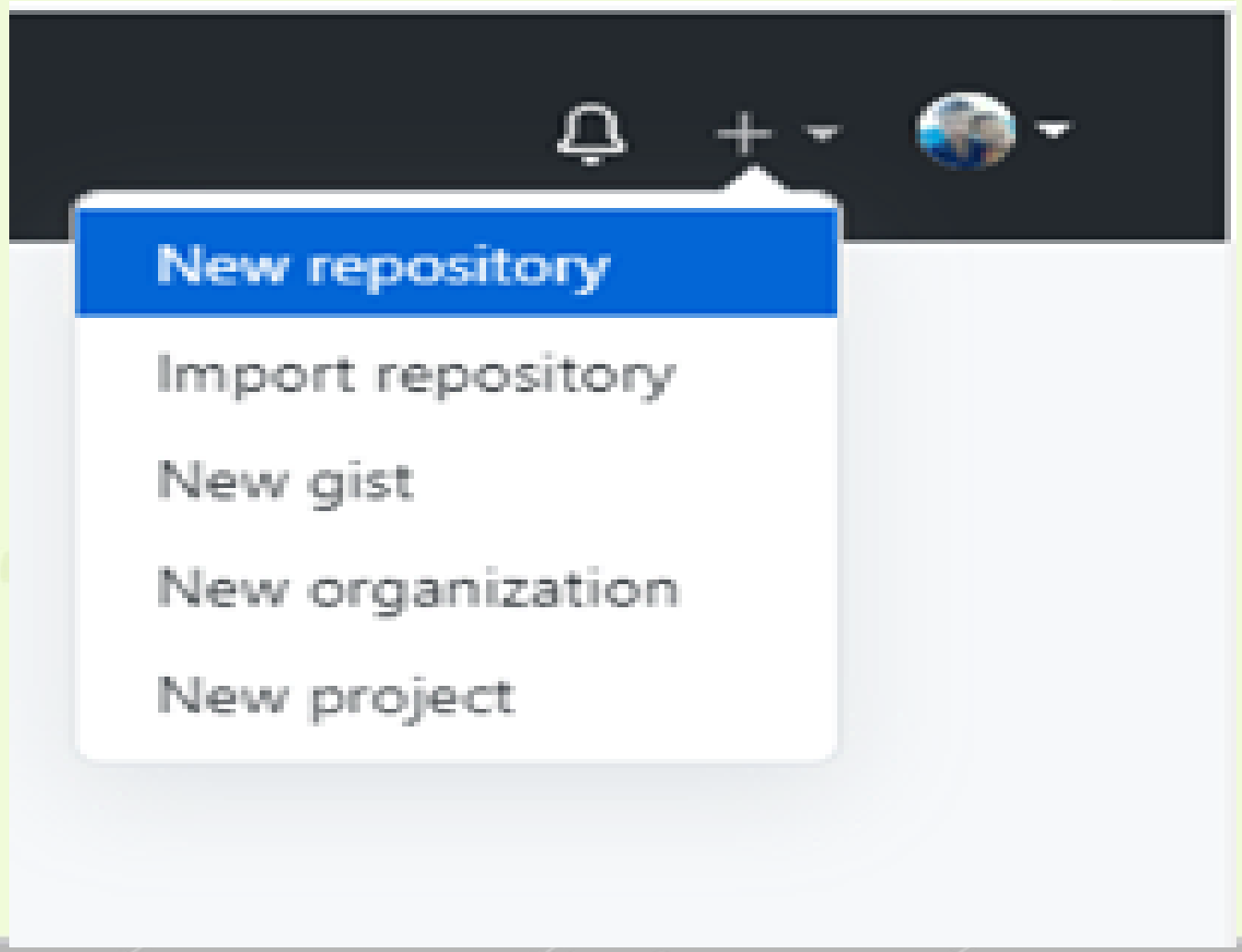
GITHUB

- Là 1 website cho ta đăng kí Free
- Mỗi tài khoản đăng kí có thể lưu rất nhiều các file
- Cho phép hosting website

- Dùng GIT&GITHUB để lưu trữ file làm việc
- GIT&GITHUB hoàn toàn miễn phí
- Không giới hạn dung lượng lưu trữ
- Có khả năng lấy lại file mà ta lỡ xóa, quay lại bước trước đó mà ta mới làm
- Chúng ta làm việc thì trên local
- Git quản lý toàn bộ nội dung của file, đảm bảo cho file nguyên vẹn thông
  qua checksum 40 kí tự mã 16

Used

B1. Create New Repository



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
Import a repository.

Owner * **Repository name** *

VANTHO15 / LoveYou

Great repository names are short and memorable. Need inspiration? How about verbose-octo-chainsaw?

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.

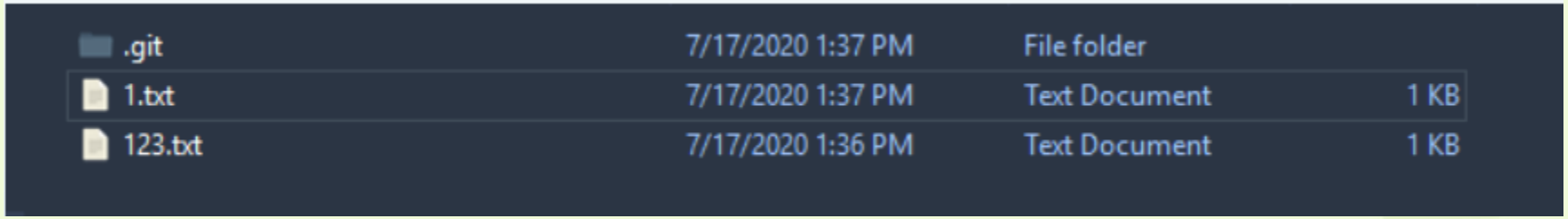
Initialize this repository with:
Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.

B2. Create 1 forder
B3. Choose “Git Bash Here”
B4. “Git init” : Create 1 Store

Create some files



B5. Click “git status”

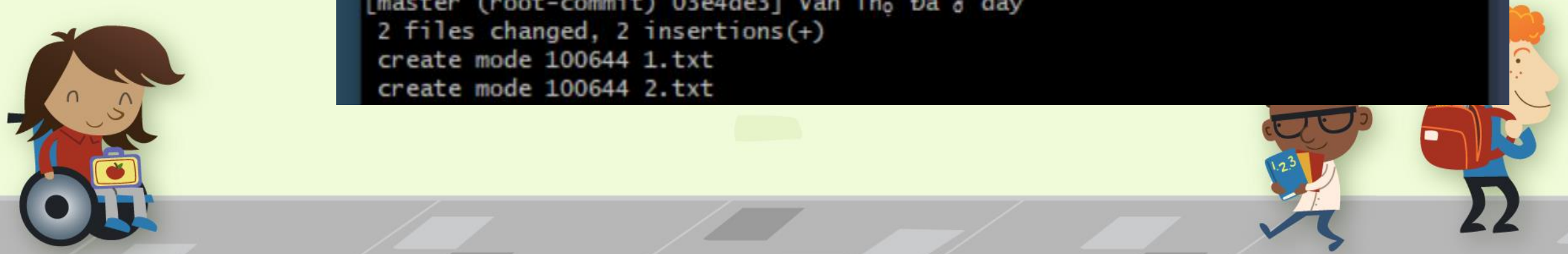
```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        1.txt
        2.txt
```

B6: - Click “ git add .” or “ git add *” : Add all file
- Cick “ git add file_name” : Add one file

```
Admin@DESKTOP-NGS38CK MINGW64 ~/Desktop/Git/Demo_Git (master)
$ git add *

Admin@DESKTOP-NGS38CK MINGW64 ~/Desktop/Git/Demo_Git (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   1.txt
        new file:   2.txt
```

B7: Click “ git commit –m “note”” : commit up, “m” is message

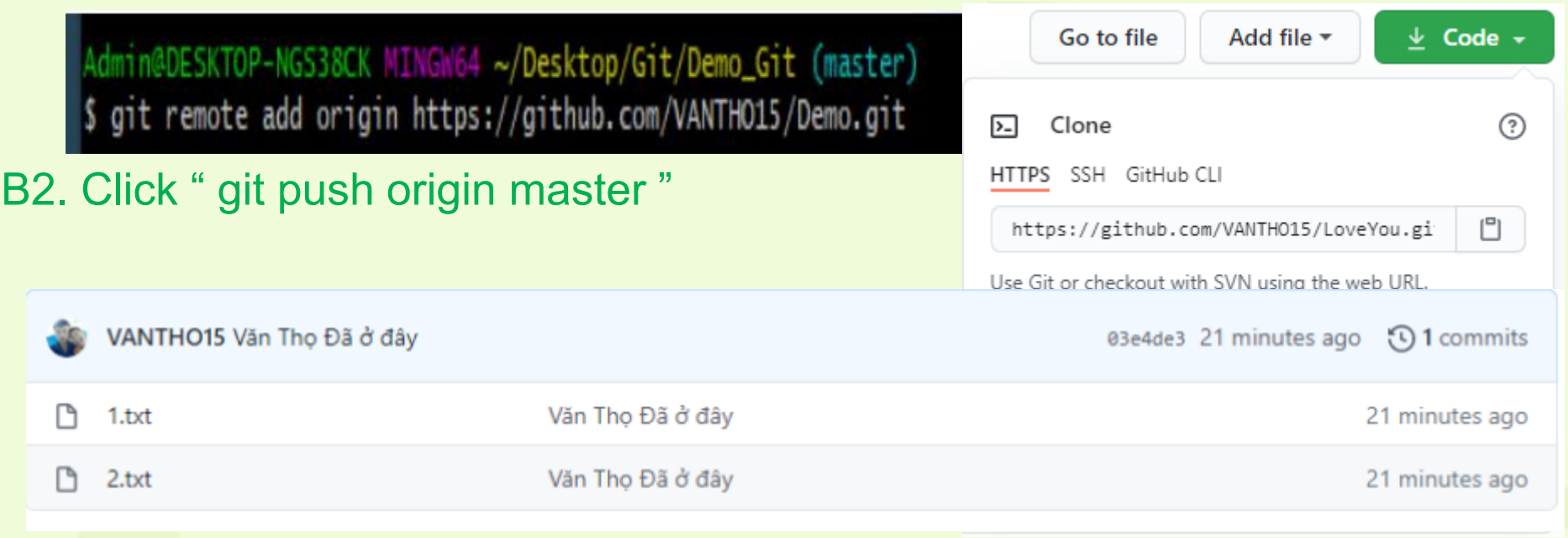
```
$ git commit -m "Văn Thọ Đã ở đây"
[master (root-commit) 03e4de3] Văn Thọ Đã ở đây
 2 files changed, 2 insertions(+)
 create mode 100644 1.txt
 create mode 100644 2.txt
```



B8: Click “ git remove add origin http://.....”

```
Admin@DESKTOP-NGS38CK MINGW64 ~/Desktop/Git/Demo_Git (master)
$ git remote add origin https://github.com/VANTHO15/Demo.git
```

B2. Click “ git push origin master ”



OK

Pull Data

- ❑Create foder
- ❑Click “Git bash here”
- ❑Git init
- ❑Git remove add origin http://.....
- ❑Git pull origin master

Clone Data

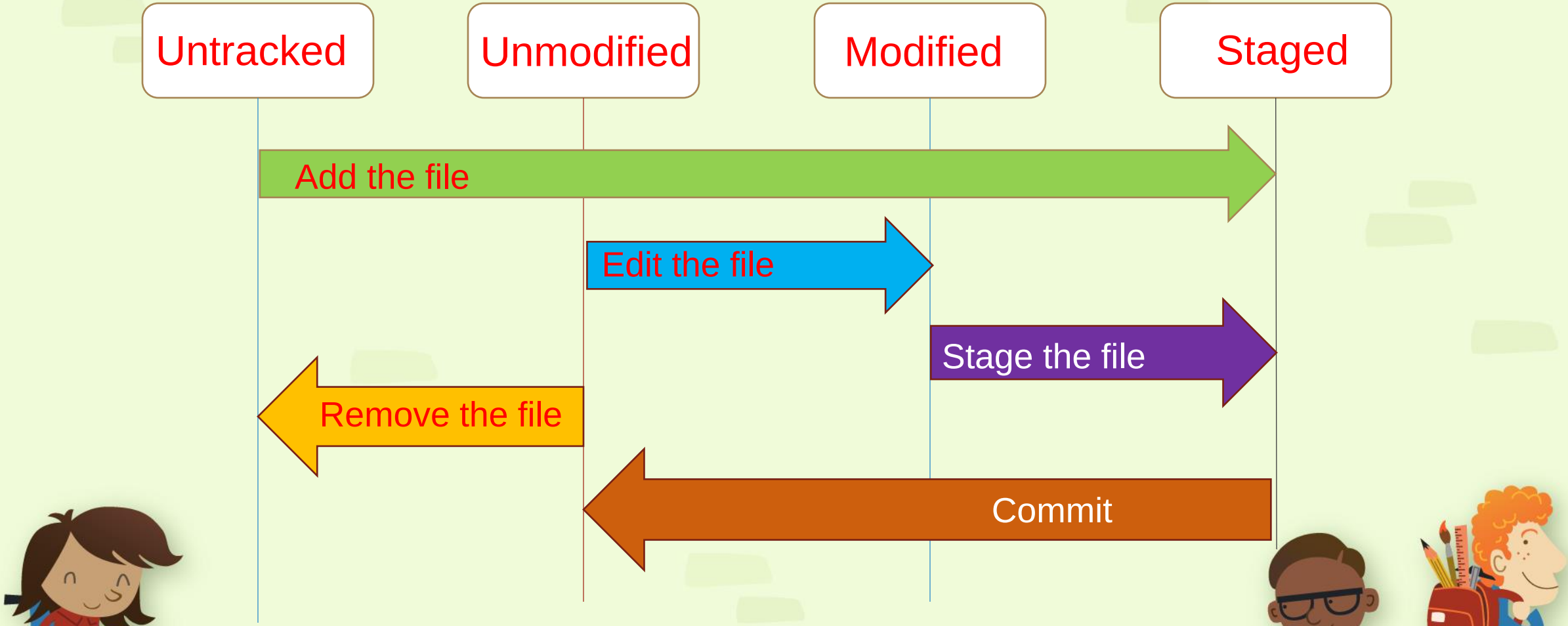
- Git clone link
- Cd Ten_Repository
- ……
- Git pull origin master

Recoding changes to the Repository

Content

1. Checking the status of the file
2. Tracking new file
3. Ignoring files
4. Viewing your stager and unstaged changes
5. Committing your change
6. Skipping the Staging aria
7. Remove files
8. Moving files

The lifecycle of the status of your files



1. Git status :Checking the status of the file
2. Git add “Tenfile”: Tracking new file
3. Ignoring files
4. Viewing your stager and unstaged changes
5. Git commit -a –m “ahihi” :Committing your change
6. Git add * :Skipping the Staging aria
7. Git rm –cached 12.txt: Remove files
8. Moving files

Viewing The Commit History

1. You can also use -2, which limits the output to only the last two entries
2. Git log –p -2
3. Git log
4. If you want to see some abbreviated start for each commit you can use the –start option
5. The –start option prints below each commit entry a list of modified files, how many files were changed, and how many lines in those files added and removed
6. Git log --start

TAG

- Tagging
- Listing your Tag
- Creating tags
- Lightweight Tags
- Tagging Later
- Sharing Tag
- Checking Out Tags
- Dalete Tag

TAGGING

- Git has the ability to tag specific points in history as being important
- Ex V1.0, V1.1 …
- “git tag” :Listing the available tags in Git

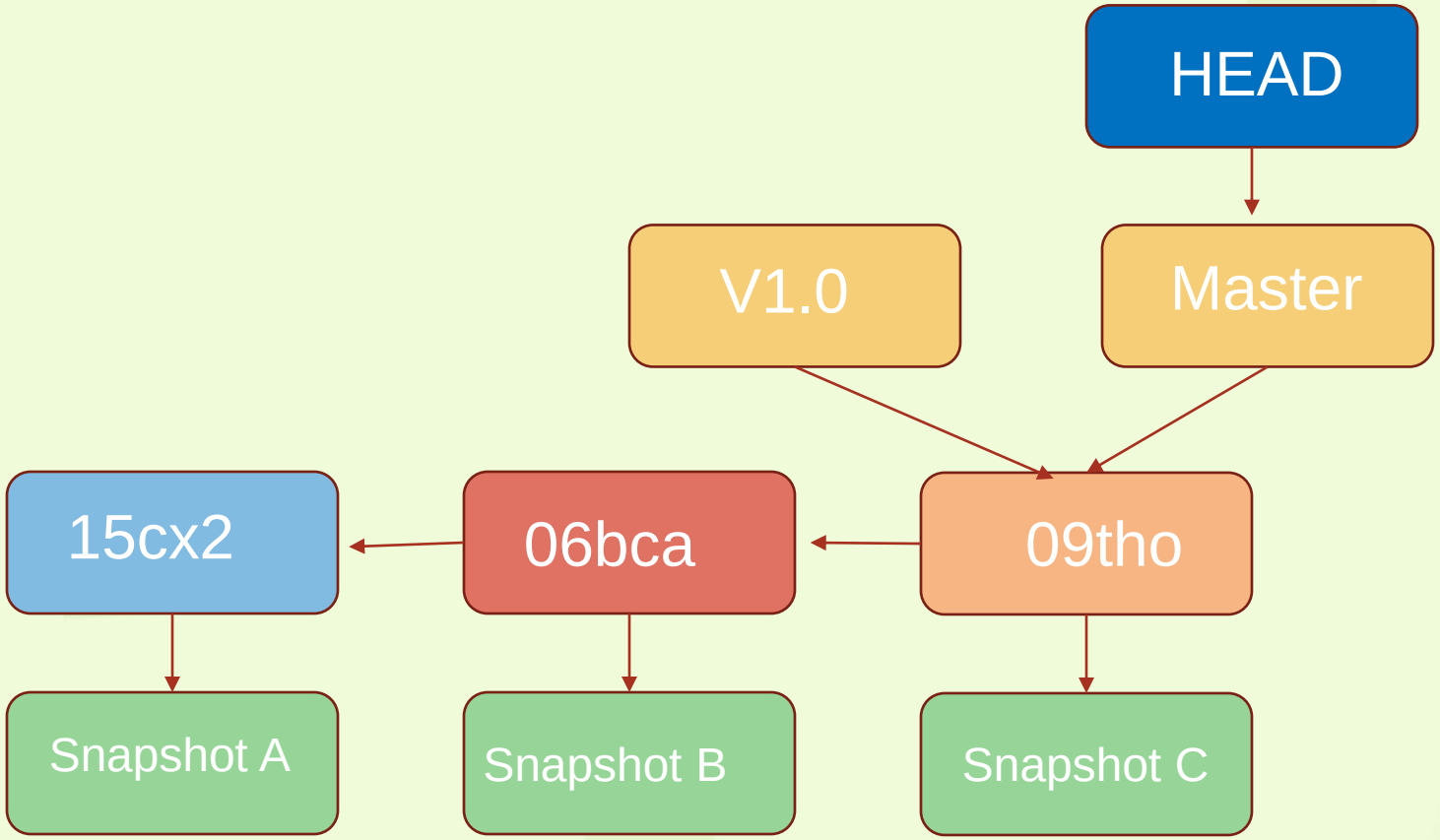
Create Tag

+ Create tag for commit if we just commit
    - Git tag –a V1.0 –m “Version 1”
    - Git show V1.0 : show info Tag
+ Create for commit random
    - Git tag –a V1.1 key (key is commit random)

Used Tag

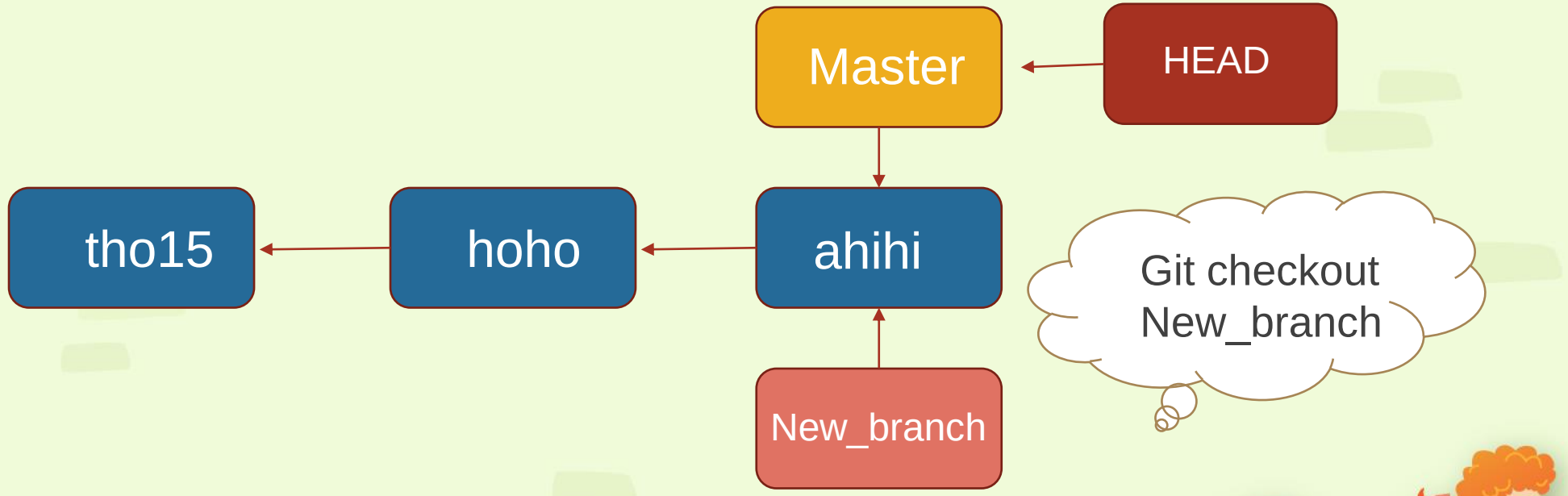
- Git push origin V1.0 : Push Tag
- Git push origin --tags : Push all Tag
- Git checkout -b nhanh2 v1.0 : Detach 1 tag to 1 branch
- Git tag -d v1.1 : delete tag
- Git push origin –delete v1.0 : delete Tag on Server

Git Branching



Creating a New Branch

+ Git branch Name_branch.
+ This creates a new pointer to the same commit you're currently on.



Branch Management

1. We will create, merged, delete some branches

- Git branch : view list branch
- Git branch –v : view commit finish of branch
- Git branch –merged : what is branch was merged to current branch
- Git branch –no-merged : what is branch was not merger to current branch
- Git branch –d name_branch : delete branch
- Git push origin –delete name_branch

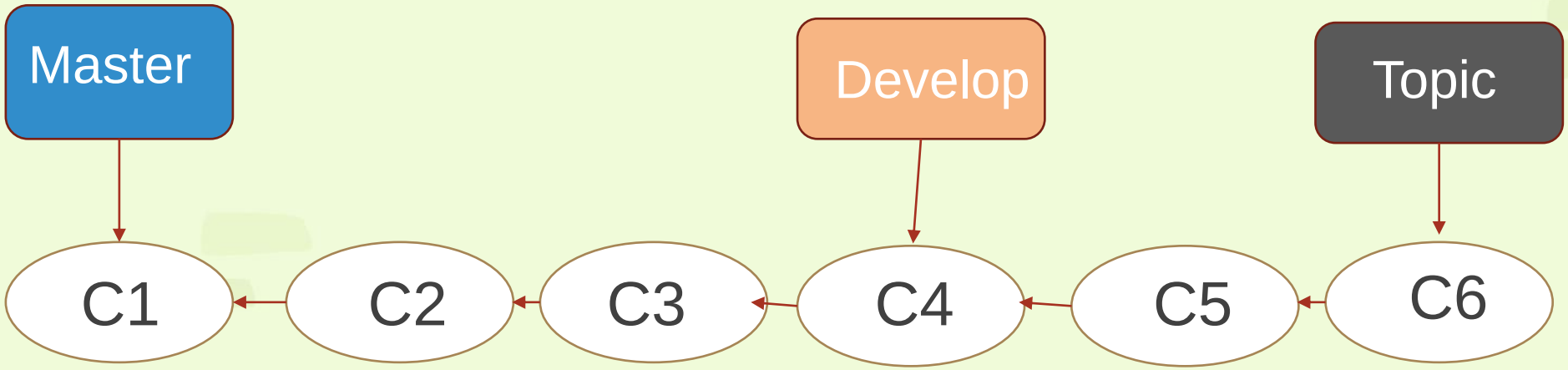
Note :

+ the * character => the branch that HEAD points to
+ “-d” delete branch merged
+ “-D” delete branch merge

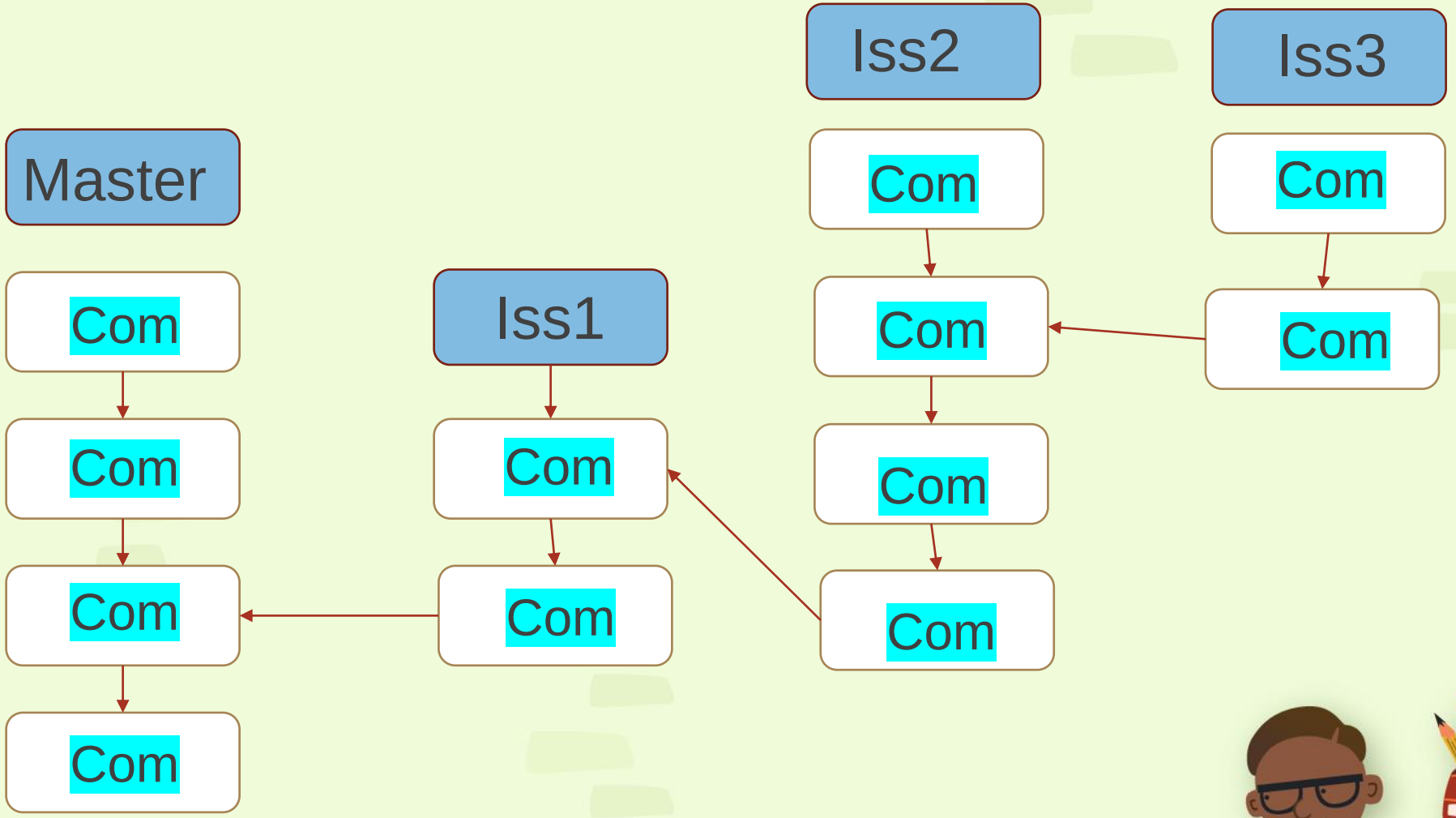
Branching workflows

(Luồng làm việc của nhánh)

Long Running Branches



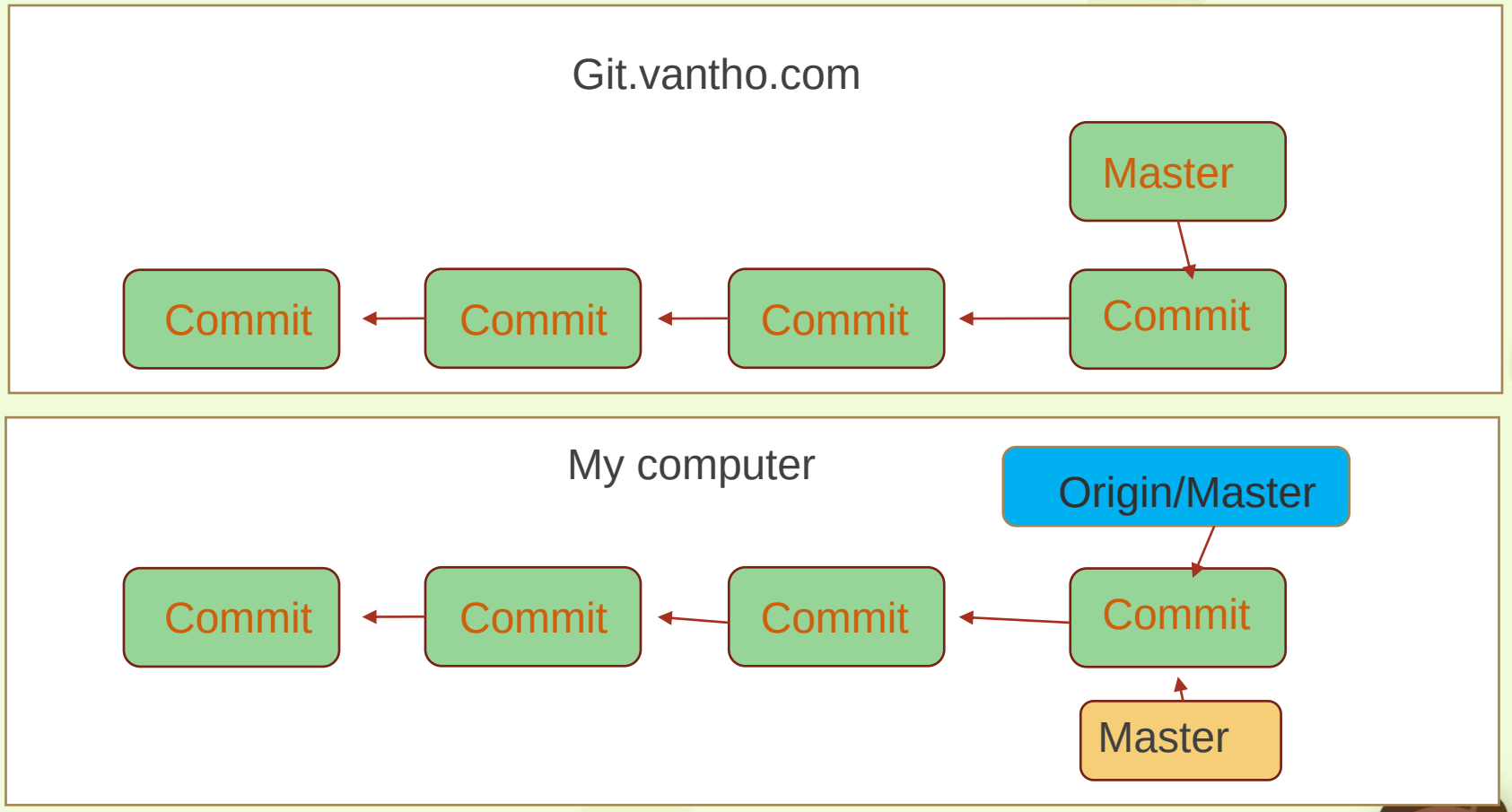
Topic Branches

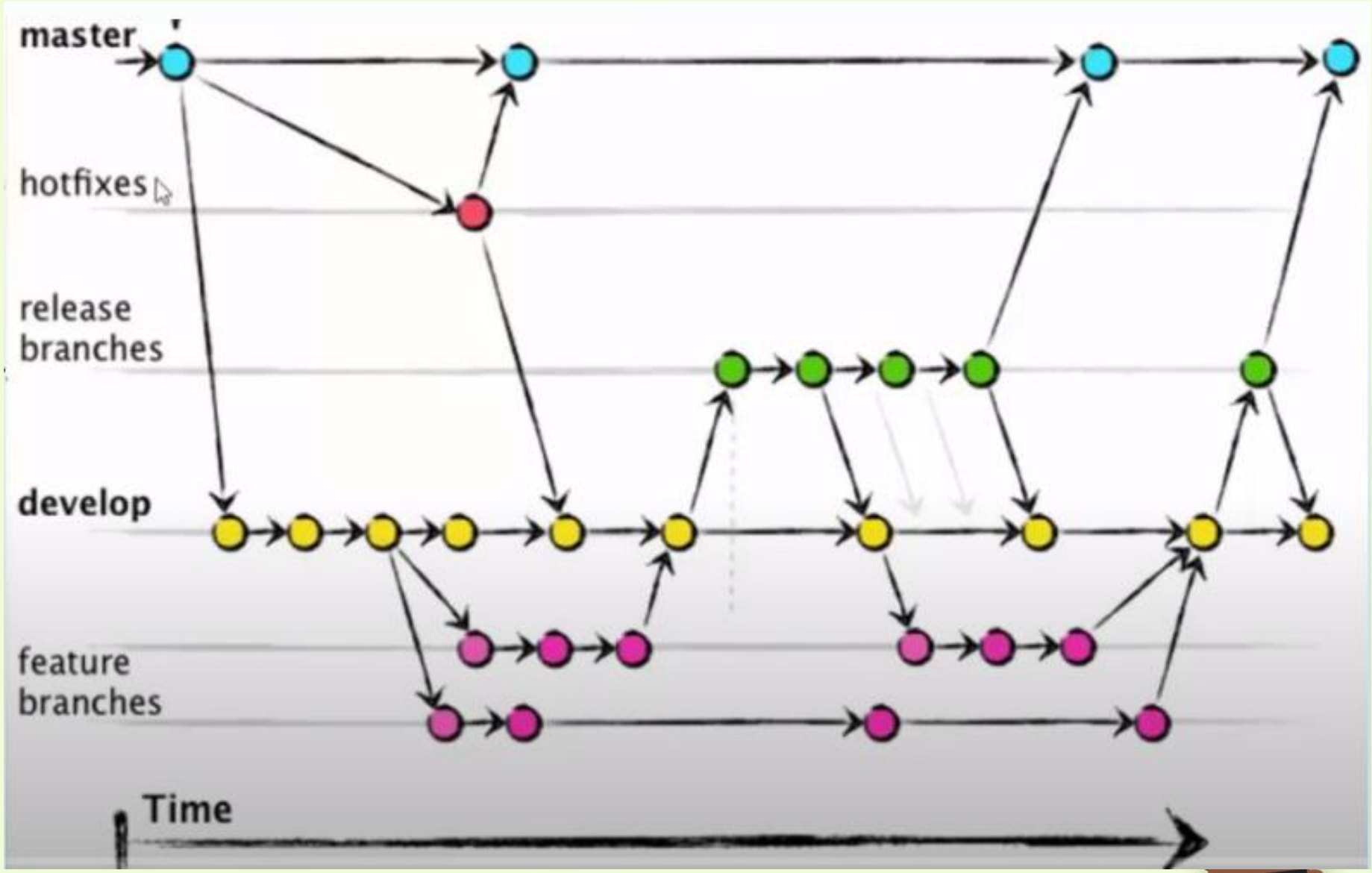


Branches far

- Remove references are references (pointers) in your Repo
- You can get a full list of remote reference with
- Git ls-remote “origin” or git remote show “origin”

Server and Local Repo After Cloning




master
hotfixes
release
branches
develop
feature
branches
Time

Command

- Git branch : view list branch
- Git checkout name_branch : return branch name_branch
- Git log –online –decore: view HEAD

Create Branch and Merge

- Git branch name_branch : Create branch
- Khi này nhánh name_branch đang trỏ tới commit mà nhánh trước đó ta đang trỏ tới
- Vd ta đang ở nhánh master mà trỏ tới commit A thì nhánh mới cũng trỏ tới commit A
- Merge thì đưa HEAD về master
- Khi marge thì sẽ tạo ra 1 commit mới

Example

Ví dụ ra đang ở nhánh master muốn marge nhánh nhanh1

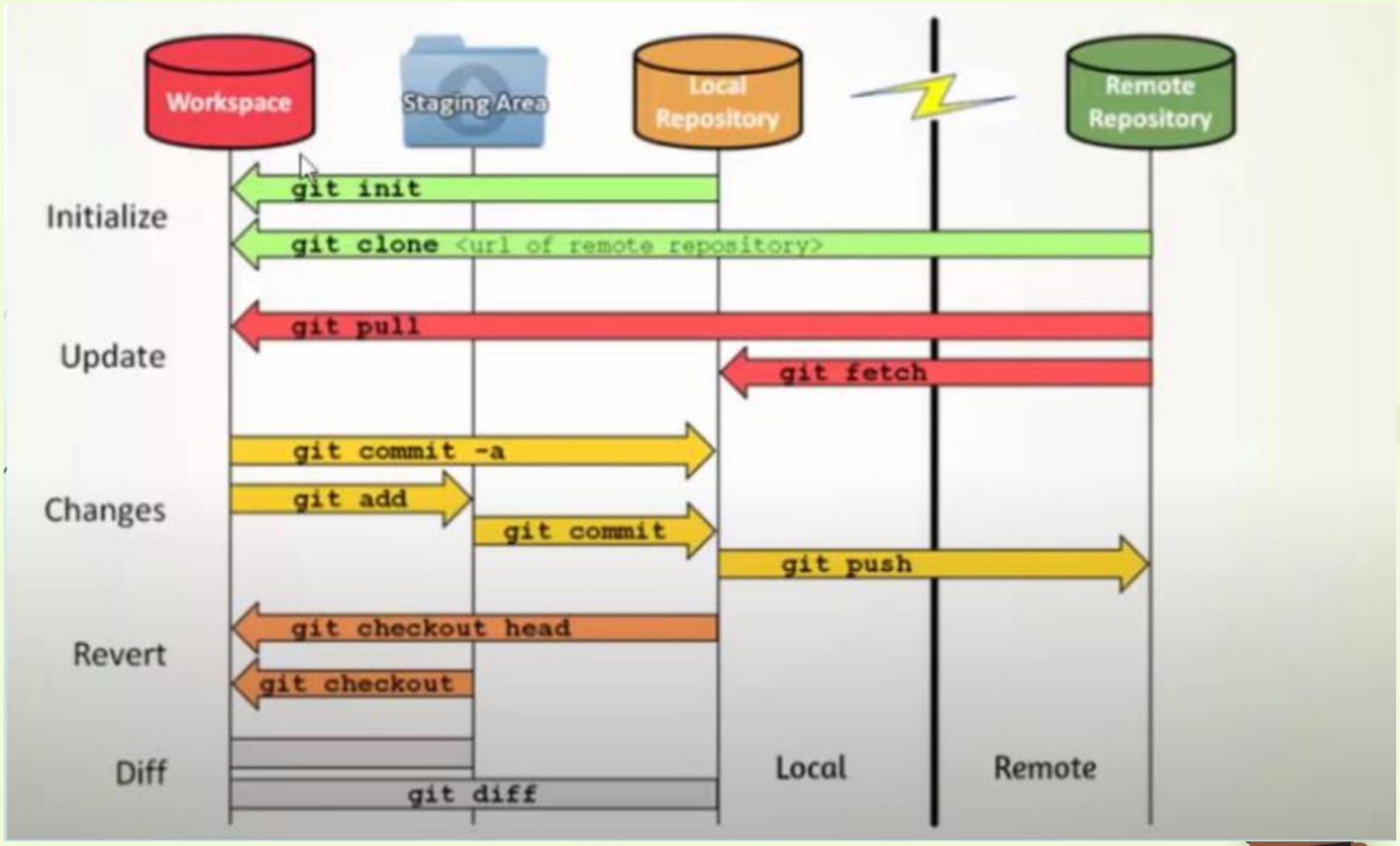
- Git marge nhanh1
- Git push –set-upstream origin master : kết nối master local với master server
- Git checkout nhanh1
- Git push –set-upstream origin nhanh1

ERROR

Vấn đề : 1 người sửa 1 file trên master ,1 người sửa trên nhanh1 có tên file trùng nhau, khi merge thì nó không biết nên lấy cái nào.

Sửa:

+ Tải diffmerge về : https://sourcegear.com/diffmerge/downloads.php
+ git config --global merge.tool : xem tool
+ git config --global merge.tool diffmerege
+ git merge nhanh1
+ git mergetool
+ diffmerge


Workspace
Staging Area
Local Repository
Remote Repository
Initialize
git init
git clone <url of remote repository>
Update
git pull
git fetch
Changes
git commit -a
git add
git commit
git push
Revert
git checkout head
git checkout
Diff
git diff
Local
Remote

Thank you