1. Giới thiệu về Git và GitHub:

    - Hệ thống quản lý phiên bản phân tán (Distributed Version Control System - DVCS) và tại sao Git được sử dụng phổ biến:

Hệ thống quản lý phiên bản phân tán (DVCS) là một phương pháp để quản lý và theo dõi các phiên bản của mã nguồn và tài liệu trong một dự án phần mềm. DVCS cho phép nhiều người cùng làm việc trên một dự án, mỗi người có bản sao đầy đủ của toàn bộ lịch sử phiên bản trên máy tính cá nhân của mình.

Git là một hệ thống quản lý phiên bản phân tán (DVCS) phổ biến nhất hiện nay. Nó được phát triển bởi Linus Torvalds vào năm 2005 để quản lý mã nguồn của dự án hệ điều hành Linux. Git đã trở thành một tiêu chuẩn trong cộng đồng phát triển phần mềm nhờ tính linh hoạt, tốc độ nhanh, khả năng xử lý dự án lớn, và khả năng làm việc offline.

Một số lý do tại sao Git được sử dụng phổ biến:

- Hệ thống phân tán: Git cho phép mỗi người dùng có bản sao đầy đủ của toàn bộ lịch sử phiên bản, không phụ thuộc vào một máy chủ trung tâm. Điều này cho phép làm việc offline, làm việc song song và tích hợp các thay đổi dễ dàng.
- Tính linh hoạt: Git không đặt ra nhiều ràng buộc về cách làm việc của bạn. Bạn có thể tạo nhánh, ghép nhánh, thay đổi lịch sử, và hợp nhất các nhánh một cách linh hoạt.
- Tốc độ nhanh: Git được thiết kế để hoạt động nhanh chóng với các dự án lớn. Thao tác như tạo nhánh, chuyển đổi giữa các nhánh, và ghi lại thay đổi đều được thực hiện một cách nhanh chóng.
- Hỗ trợ đa nền tảng: Git có sẵn trên nhiều nền tảng, bao gồm Windows, macOS và Linux. Bạn có thể sử dụng Git trên môi trường phát triển của bạn mà không cần thay đổi công cụ.

b) Giới thiệu về GitHub và vai trò của nó trong việc lưu trữ mã nguồn và hợp tác trong dự án phần mềm:

GitHub là một dịch vụ lưu trữ mã nguồn trực tuyến dựa trên Git. Nó cung cấp một giao diện web cho phép các nhà phát triển tải lên, lưu trữ và quản lý mã nguồn của họ. GitHub cung cấp nền tảng để quản lý dự án phần mềm và hợp tác giữa các thành viên trong một nhóm làm việc.

Vai trò của GitHub trong việc lưu trữ mã nguồn và hợp tác trong dự án phần mềm bao gồm:

- Lưu trữ mã nguồn: GitHub cho phép lưu trữ các repository (kho chứa) Git công khai hoặc riêng tư. Bạn có thể tải lên mã nguồn của dự án lên GitHub và duy trì lịch sử phiên bản, nhánh và thực hiện các thao tác quản lý phiên bản trên đó.
- Hợp tác: GitHub cung cấp các tính năng giúp đồng đội làm việc cùng nhau trên cùng một dự án phần mềm. Bạn có thể mời thành viên vào dự án của mình, kiểm soát quyền truy cập, và cho phép họ đóng góp vào mã nguồn. GitHub cũng cung cấp công cụ để quản lý vấn đề, theo dõi yêu cầu kéo (pull requests) và đánh giá mã nguồn.

- Tích hợp và liên kết: GitHub cung cấp tích hợp với nhiều dịch vụ và công cụ phát triển phổ biến khác. Bạn có thể kết hợp GitHub với các dịch vụ kiểm tra tự động, CI/CD (Continuous Integration/Continuous Deployment), và hệ thống quản lý công việc để tạo một quy trình làm việc phát triển phần mềm toàn diện.

Cài đặt và cấu hình Git:
a) Hướng dẫn cài đặt Git trên hệ điều hành của bạn:

- Windows:

1. Truy cập trang web chính thức của Git tại địa chỉ: https://git-scm.com/
2. Tải xuống bản cài đặt Git phù hợp với hệ điều hành Windows của bạn.
3. Chạy tệp cài đặt đã tải xuống và tuân theo các hướng dẫn trên màn hình để hoàn thành quá trình cài đặt.

- macOS:

1. Có thể cài đặt Git thông qua Homebrew (một trình quản lý gói phổ biến trên macOS) bằng cách mở Terminal và chạy lệnh sau:

```
brew install git
```

Linux (Ubuntu):

1. Mở Terminal và chạy các lệnh sau:

```
sudo apt update
sudo apt install git
```

b) Giới thiệu các cấu hình cơ bản:
Sau khi cài đặt Git, bạn có thể cấu hình một số thông tin cơ bản, bao gồm:

- Thiết lập tên người dùng:

```
git config --global user.name "Your Name"

Thiết lập địa chỉ email:

git config --global user.email "your.email@example.com"
```

Cấu hình khóa SSH (nếu cần):

- Nếu bạn muốn sử dụng khóa SSH để xác thực khi làm việc với GitHub hoặc các dịch vụ tương tự, bạn cần tạo một cặp khóa SSH trên máy tính của mình. Hướng dẫn chi tiết về việc tạo khóa SSH có thể được tìm thấy trên trang web chính thức của Git hoặc GitHub.
- Sau khi tạo khóa SSH, bạn cần thêm khóa công khai vào tài khoản GitHub của bạn (nếu sử dụng GitHub).
- Sau đó, bạn có thể cấu hình Git để sử dụng khóa SSH bằng cách thêm khóa vào Agent SSH trên máy tính của bạn.

> Lưu ý: Các bước chi tiết để cấu hình khóa SSH có thể khác nhau tùy theo hệ điều hành và môi trường sử dụng. Hãy kiểm tra tài liệu chính thức của Git hoặc GitHub để có hướng dẫn chi tiết hơn.

Đây là các cấu hình cơ bản để bắt đầu sử dụng Git. Bạn có thể tùy chỉnh các cấu hình khác theo nhu cầu của bạn bằng cách sử dụng lệnh `git config`.

c) Cấu hình Git mặc định:
Sau khi cài đặt Git, bạn có thể cấu hình một số tùy chọn mặc định cho Git. Dưới đây là một số cấu hình phổ biến:

- Xác nhận tên nhánh hiện tại trong output của Git:

  ```
  git config --global branch.autosetuprebase always
  ```

- Tự động điền vào thông tin xác nhận (commit):

  ```
  git config --global commit.template ~/.gitmessage.txt
  ```

- Hiển thị thông tin commit dạng graph:

  ```
  git config --global alias.graph "log --graph --oneline --decorate --all"
  ```

- Tắt kiểm tra sự thay đổi không được theo dõi (untracked files):

  ```
  git config --global status.showUntrackedFiles no
  ```

- Cấu hình một trình soạn thảo mặc định cho Git (ví dụ: Notepad++):

  ```
  git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"
  ```

Đây chỉ là một số ví dụ về cấu hình. Bạn có thể tùy chỉnh các tùy chọn khác theo nhu cầu của mình bằng cách sử dụng lệnh `git config`. Để xem danh sách các tùy chọn cấu hình hiện tại, bạn có thể sử dụng lệnh:

```
git config --list
```

Cấu hình được thêm với tùy chọn `--global` sẽ áp dụng cho tất cả các repository trên máy tính của bạn. Nếu bạn chỉ muốn áp dụng cấu hình cho một repository cụ thể, hãy loại bỏ tùy chọn `--global` và thực hiện lệnh trong thư mục của repository đó.

Để thêm khóa SSH vào Git, bạn có thể làm theo các bước sau:

1. Kiểm tra sự tồn tại của khóa SSH: Mở Terminal và chạy lệnh sau để kiểm tra xem bạn đã có khóa SSH hay chưa:

   ```
   ls -al ~/.ssh
   ```

2. Tạo khóa SSH mới (nếu chưa có):

   - Nếu kết quả từ bước trên không chứa các tệp khóa SSH (`id_rsa` và `id_rsa.pub`), bạn có thể tạo khóa SSH mới bằng cách chạy lệnh sau:

     ```
     ssh-keygen -t rsa -b 4096 -C "your.email@example.com"
     ```

   - Lưu ý: Thay thế `"your.email@example.com"` bằng địa chỉ email mà bạn muốn liên kết với khóa SSH.

3. Sao chép khóa công khai:

   - Chạy lệnh sau để sao chép nội dung của tệp khóa công khai (`id_rsa.pub`):

     ```
     pbcopy < ~/.ssh/id_rsa.pub
     ```

   - Nếu bạn đang sử dụng Windows Subsystem for Linux (WSL), thay lệnh `pbcopy` bằng `clip`:

     ```
     cat ~/.ssh/id_rsa.pub | clip
     ```

4. Thêm khóa SSH vào tài khoản GitHub (hoặc dịch vụ Git tương tự):

   - Sao chép khóa công khai SSH:
   - Mở Terminal và chạy lệnh sau để sao chép nội dung của tệp khóa công khai (`id_rsa.pub`):

     ```
     bashCopy code
     cat ~/.ssh/id_rsa.pub
     ```

   - Truy cập vào tài khoản Git:
   - Đăng nhập vào tài khoản của bạn trên trang web Git (ví dụ: GitHub).
   - Truy cập vào phần cài đặt (Settings) hoặc tùy chọn tương tự:
   - Tìm đến phần cài đặt tài khoản của bạn.
   - Tìm đến phần SSH and GPG keys:

- Trong cài đặt tài khoản, tìm và chọn phần "SSH and GPG keys" hoặc tương tự.
- Thêm khóa SSH:
- Chọn "Add SSH key" hoặc tương tự để thêm khóa SSH mới.
- Đặt tên cho khóa SSH (ví dụ: "My SSH Key").
- Dán nội dung khóa công khai (`id_rsa.pub`) đã sao chép từ bước 1 vào ô "Key" hoặc tương tự.
- Lưu và xác nhận:
- Nhấn "Add SSH key" hoặc tương tự để lưu khóa SSH vào tài khoản của bạn.
- Có thể yêu cầu xác nhận mật khẩu của bạn hoặc hình thức xác thực khác.
-

Sau khi hoàn thành các bước trên, khóa SSH của bạn đã được liên kết với tài khoản GitHub hoặc dịch vụ Git tương tự. Bạn có thể sử dụng khóa SSH này để xác thực khi làm việc với các repository từ xa mà không cần cung cấp thông tin đăng nhập mỗi lần.

Các khái niệm cơ bản trong Git:

a) Repository: Repository (kho chứa) trong Git là nơi lưu trữ và quản lý mã nguồn của dự án. Nó chứa tất cả các tệp tin, thư mục, lịch sử thay đổi và thông tin về phiên bản của mã nguồn. Repository có thể là một thư mục trên máy tính cục bộ hoặc trên một dịch vụ như GitHub, GitLab hoặc Bitbucket.

Để tạo một repository trên GitHub, bạn có thể làm theo các bước sau:

1. Đăng nhập vào tài khoản GitHub của bạn.
2. Trên trang chính của GitHub, nhấp vào nút "New" (hoặc "Tạo repository" nếu bạn đang sử dụng giao diện bằng tiếng Việt).
3. Đặt tên cho repository của bạn.
4. Tùy chọn chọn công khai (public) hoặc riêng tư (private).
5. Chọn "Initialize this repository with a README" nếu bạn muốn tạo một tệp README.md trong repository.
6. Nhấp vào nút "Create repository" để tạo repository mới trên GitHub.

b) Commit: Commit là một hành động quan trọng trong Git, đại diện cho việc lưu trạng thái hiện tại của repository. Khi bạn thực hiện commit, bạn lưu lại các thay đổi trong repository cùng với một thông điệp mô tả về những gì đã thay đổi. Mỗi commit có một mã định danh duy nhất (hash) để xác định nó.

Để tạo một commit trong Git, bạn cần thực hiện các bước sau:

1. Sử dụng lệnh `git add` để thêm các tệp hoặc thay đổi vào stage (vùng chờ commit).

   ```
   git add <file1> <file2> ...
   ```

2. Sử dụng lệnh `git commit` để tạo commit với thông điệp mô tả.

```
git commit -m "Commit message"
```

Lưu ý: Thay thế "Commit message" bằng thông điệp mô tả ngắn gọn về những gì đã thay đổi trong commit.

3. Commit sẽ được tạo và lưu trữ trong repository của bạn.

Bạn có thể sử dụng các lệnh như `git log` để xem lịch sử commit và `git diff` để xem các thay đổi trong mỗi commit.

c) Branch: Branch (nhánh) là một bản sao của repository, cho phép bạn phát triển và thử nghiệm các tính năng mới mà không ảnh hưởng đến phiên bản chính của mã nguồn. Mỗi nhánh có lịch sử commit riêng và có thể được tạo, chuyển đổi và xóa.

Để tạo một branch mới trong Git, bạn có thể sử dụng lệnh:

```
git branch <branch-name>
```

Để chuyển đổi sang một branch khác, bạn có thể sử dụng lệnh:

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

Sau khi hoàn thành các thay đổi trong branch, bạn có thể merge (hợp nhất) branch vào nhánh chính bằng cách sử dụng lệnh:

```
git merge <branch-name>
```

d) Conflict: Conflict (xung đột) trong Git xảy ra khi hai hoặc nhiều phiên bản của cùng một tệp hoặc cùng một dòng trong tệp được chỉnh sửa trong các nhánh khác nhau, và Git không thể tự động giải quyết xung đột này. Khi xảy ra xung đột, bạn cần giải quyết nó bằng cách xác định phiên bản nào sẽ được giữ lại hoặc kết hợp các phiên bản lại với nhau.

Để giải quyết xung đột, bạn cần thực hiện các bước sau:

1. Xem các tệp có xung đột bằng lệnh `git status` hoặc các công cụ hỗ trợ Git.
2. Mở và chỉnh sửa các tệp có xung đột để giải quyết xung đột theo ý muốn.
3. Thực hiện commit sau khi giải quyết xung đột.
4. Tiếp tục quá trình merge hoặc rebase nếu còn các xung đột khác.

Để giải quyết xung đột là một quá trình thủ công và đòi hỏi sự cẩn thận để đảm bảo không mất mát dữ liệu quan trọng trong quá trình hợp nhất mã nguồn từ các nhánh khác nhau.

1. Các hoạt động hàng ngày với Git và GitHub:

a) Clone: Để clone (sao chép) một repository từ GitHub về máy tính của bạn, bạn có thể làm theo các bước sau:

1. Truy cập vào trang repository trên GitHub mà bạn muốn clone.
2. Nhấp vào nút "Code" (hoặc "Clone" nếu bạn sử dụng giao diện bằng tiếng Việt) và sao chép URL của repository.
3. Mở Terminal và di chuyển đến thư mục mà bạn muốn clone repository vào.
4. Chạy lệnh `git clone` và dán URL repository sau đó nhấn Enter.

   ```
   git clone <repository-url>
   ```

   Repository sẽ được sao chép từ GitHub về máy tính của bạn.

b) Pull và Push:

- Pull: Khi bạn muốn cập nhật mã nguồn trong repository cục bộ của mình với phiên bản mới nhất từ repository trên GitHub, bạn sử dụng lệnh `git pull`. Lệnh này sẽ tự động kéo các thay đổi mới nhất từ repository từ xa và áp dụng chúng vào repository cục bộ của bạn.

  ```
  Copy code
  git pull
  ```

- Push: Khi bạn muốn chia sẻ các thay đổi của mình lên repository trên GitHub, bạn sử dụng lệnh `git push`. Lệnh này sẽ gửi các commit của bạn lên repository từ xa.

  ```
  git push
  ```

Lưu ý: Trước khi push, hãy đảm bảo bạn đã thực hiện commit các thay đổi của mình trong repository cục bộ.

c) Pull Request: Pull Request (PR) là quy trình trong GitHub cho phép bạn đề xuất các thay đổi của mình vào repository chính. Người khác có thể xem xét, thảo luận và đánh giá thay đổi của bạn trước khi hợp nhất chúng vào repository chính.

Để tạo và xem xét pull request trên GitHub, bạn có thể làm theo các bước sau:

1. Truy cập vào trang repository trên GitHub mà bạn muốn tạo pull request.
2. Nhấp vào tab "Pull requests".
3. Nhấp vào nút "New pull request".
4. Chọn branch bạn muốn hợp nhất thay đổi từ (branch nguồn) và branch bạn muốn hợp nhất vào (branch đích).
5. Xem và kiểm tra các thay đổi, thêm các comment và mô tả chi tiết về pull request của bạn.
6. Nhấp vào nút "Create pull request" để tạo pull request và chờ xem xét và phản hồi từ người khác.

Trong quá trình xem xét pull request, bạn và các thành viên khác trong dự án có thể thảo luận, yêu cầu thay đổi hoặc đưa ra ý kiến. Khi pull request được chấp nhận, các thay đổi sẽ được hợp nhất vào repository chính.

Các tính năng nâng cao:

a) Tag: Tag trong Git được sử dụng để đánh dấu một điểm cụ thể trong lịch sử của repository, thường được sử dụng để đánh dấu phiên bản phần mềm hoặc các bản phát hành quan trọng. Tag giúp dễ dàng truy cập và phân biệt các phiên bản của mã nguồn.

Để tạo tag trong Git, bạn có thể sử dụng lệnh:

- Tag gắn kết với commit hiện tại:

```
git tag <tag-name>
```

- Tag gắn kết với một commit cụ thể:

```
git tag <tag-name> <commit-hash>
```

Sau khi tạo tag, bạn có thể chia sẻ tag với người khác bằng cách sử dụng lệnh `git push` và tên tag:

```
git push origin <tag-name>
```

b) Submodule: Submodule trong Git cho phép bạn quản lý các phần tử con (sub-repositories) trong một repository lớn hơn. Điều này hữu ích khi bạn muốn sử dụng mã nguồn từ các dự án khác hoặc muốn chia nhỏ và quản lý các thành phần độc lập trong dự án.

Để thêm một submodule vào repository hiện tại, bạn có thể sử dụng lệnh:

```
git submodule add <repository-url> <path>
```

Trong đó, `<repository-url>` là URL của submodule và `<path>` là đường dẫn đến thư mục nơi submodule sẽ được lưu trữ.

Khi clone repository chứa submodule, bạn cần thực hiện lệnh sau để lấy mã nguồn của submodule:

```
git submodule init
git submodule update
```

c) Gitignore: File `.gitignore` được sử dụng để chỉ định cho Git những tệp và thư mục mà bạn không muốn theo dõi hoặc đồng bộ hóa trong quá trình commit và push. Bạn có thể sử dụng file

`.gitignore` để loại bỏ các tệp tin tạm thời, dữ liệu cấu hình cục bộ, tệp nhị phân đã biên dịch và các thành phần không cần thiết khác.

Để sử dụng file `.gitignore`, bạn cần tạo một file có tên `.gitignore` trong thư mục gốc của repository và liệt kê các mẫu tệp tin và thư mục mà bạn muốn loại bỏ. Một số ví dụ mẫu trong file `.gitignore`:

```
cssCopy code
# Bỏ qua tất cả các tệp tin .txt
*.txt

# Bỏ qua thư mục "build" và nội dung bên trong
build/

# Bỏ qua tệp tin có tên "config.ini"
config.ini
```

Các tệp và thư mục được liệt kê trong `.gitignore` sẽ không được theo dõi bởi Git và không thể thực hiện commit và push lên repository.

Các công cụ hỗ trợ:

a) Giới thiệu và so sánh với các hệ thống quản lý phiên bản khác:

- Subversion (SVN): Subversion là một hệ thống quản lý phiên bản tập trung (centralized version control system). Trong Subversion, toàn bộ lịch sử và phiên bản của mã nguồn được lưu trữ trên một máy chủ tập trung. Người dùng cần kết nối mạng để thực hiện các thao tác quản lý phiên bản. Trong khi đó, Git là một hệ thống quản lý phiên bản phân tán (distributed version control system - DVCS), cho phép người dùng sao chép toàn bộ repository và lịch sử lên máy tính cục bộ và làm việc ngoại tuyến. Git cung cấp tính linh hoạt cao hơn và khả năng làm việc hiệu quả trong các dự án phức tạp.
- Mercurial: Mercurial cũng là một hệ thống quản lý phiên bản phân tán, tương tự như Git. Mercurial và Git có nhiều điểm tương đồng về khái niệm và tính năng, nhưng có một số khác biệt về cú pháp và cách thực hiện một số thao tác. Tuy nhiên, cả hai đều mạnh mẽ và được sử dụng rộng rãi trong cộng đồng phát triển phần mềm.

b) Nêu lợi ích của việc sử dụng các công cụ hỗ trợ:

- GitKraken: GitKraken là một công cụ giao diện đồ họa cho Git, giúp người dùng thao tác với Git một cách trực quan và dễ dàng. Nó cung cấp các tính năng như xem và quản lý commit, branch, merge, và hỗ trợ xử lý xung đột. GitKraken có giao diện người dùng thân thiện, đồ họa mạnh mẽ và tích hợp với nhiều dịch vụ như GitHub và Bitbucket.

- SourceTree: SourceTree là một công cụ quản lý phiên bản đa nền tảng cho Git và Mercurial. Nó cung cấp một giao diện đồ họa dễ sử dụng để thực hiện các thao tác Git và Mercurial như commit, push, pull, và xem lịch sử. SourceTree tích hợp với các dịch vụ như GitHub, Bitbucket và GitLab.
- TortoiseGit: TortoiseGit là một ứng dụng tích hợp vào Windows Explorer, cho phép người dùng thực hiện các thao tác Git thông qua giao diện đồ họa. Bằng cách sử dụng các menu ngữ cảnh, người dùng có thể commit, push, pull và xem lịch sử của repository Git. TortoiseGit làm cho việc sử dụng Git trở nên thuận tiện và dễ dàng trong môi trường Windows.

Các công cụ hỗ trợ như GitKraken, SourceTree và TortoiseGit giúp đơn giản hóa quy trình làm việc với Git, đồng thời cung cấp giao diện đồ họa, xem lịch sử, quản lý commit và branch một cách trực quan. Điều này giúp người dùng tiết kiệm thời gian và giảm thiểu các lỗi do nhập lệnh sai hoặc không rõ ràng. Tùy thuộc vào sở thích và nhu cầu của bạn, bạn có thể chọn công cụ hỗ trợ phù hợp để làm việc với Git một cách hiệu quả.

Chúng tôi đã giới thiệu về Git và GitHub. Git là một hệ thống quản lý phiên bản phân tán (DVCS) được sử dụng rộng rãi, có khả năng theo dõi các thay đổi trong mã nguồn và hỗ trợ quản lý phiên bản phần mềm. GitHub là một dịch vụ lưu trữ mã nguồn và hợp tác trong dự án phần mềm dựa trên Git.

Chúng tôi cũng đã hướng dẫn cài đặt và cấu hình Git, bao gồm thiết lập tên người dùng, địa chỉ email và khóa SSH. Bạn cũng đã được hướng dẫn về cách thêm khóa SSH của mình vào tài khoản GitHub.

Chúng tôi đã giới thiệu các khái niệm cơ bản trong Git như repository, commit, branch và conflict. Bạn đã học cách tạo repository trên GitHub, tạo và quản lý commit, tạo branch mới và merge branch. Chúng tôi cũng đã đề cập đến xử lý xung đột khi merge code trong Git.

Bên cạnh đó, chúng tôi đã giới thiệu các hoạt động hàng ngày với Git và GitHub như clone repository, pull và push để cập nhật và chia sẻ code, cũng như tạo và xem xét pull request để hợp nhất các thay đổi vào repository chính.

Cuối cùng, chúng tôi đã trình bày các tính năng nâng cao trong Git như sử dụng tag để đánh dấu phiên bản phần mềm, sử dụng submodule để quản lý các phần tử con và sử dụng file .gitignore để loại bỏ các file và thư mục không cần thiết khỏi việc theo dõi của Git.

Chúng tôi cũng đã đề cập đến một số công cụ hỗ trợ như GitKraken, SourceTree và TortoiseGit, giúp đơn giản hóa và tăng cường trải nghiệm làm việc với Git.

Hi vọng rằng thông tin trong bài viết này đã giúp bạn hiểu rõ hơn về Git và GitHub và cung cấp cho bạn những kiến thức cơ bản để bắt đầu làm việc với Git và tham gia vào các dự án phần mềm hợp tác.