1. **Back end** 
   1. **Golang**
      1. **Gioi thieu chung**

Go hay còn gọi là Golang là ngôn ngữ lập trình mã nguồn mở, được thiết kế tại Google bởi Robert Griesemer, Rob Pike, and Ken Thompson.

Cú pháp của ngôn ngữ lập trình Golang (syntax) khá tinh gọn, mặc dù thật sự có những cú pháp mà người mới nhìn vào thật sự hơi khó chịu.

Golang giống như C++ hay Java, nó cũng là một ngôn ngữ dùng để lập trình. Tuy nhiên điểm khiến ngôn ngữ Go trở nên khác biệt nằm ở sự đơn giản của nó. Nó sở hữu những cú pháp khá tinh gọn. Điều này có thể khiến những người mới học thấy khó khăn. Dù vậy cú pháp của Go có độ tương đồng rất lớn với C++. Vì thế nếu bạn đã quen thuộc với C++ thì việc học Golang không còn là điều khó khăn.

* + 1. **Cac khai niem quan trong**
       1. **Goroutine**

Goroutine là một hàm có thể chạy đồng thời với các hàm khác.

Goroutines là những luồng gọn nhẹ, được khởi tạo với chỉ 2KB trong stack size có thể tăng hoặc giảm vùng nhớ tùy yêu cầu sử dụng.

Những ứng dụng Go có thể có rất nhiều Goroutines chạy đồng thời với nhau.

* Cơ chế của goroutine khá là đơn giản: 1 function tồn tại một cách đa luồng với các goroutine khác trên cùng một không gian bộ nhớ, Go có bộ điều khiển quản lý các goroutine rồi phân phối chúng vào các bộ xử lý logic và gắn mỗi bộ xử lý logic này với một thread hệ thống được tạo ra trước đó để thực thi các goroutine này. Nói cách khác, mỗi thread hệ thống sẽ xử lý một nhóm goroutine được điều phối thông qua bộ xử lý logic. Với bộ điều khiển quản lý tác vụ đồng thời và cơ chế bộ xử lý logic, những cái khó khăn, phức tạp khi khai báo thread Go đã xử lý hết giúp chúng ta rồi.
* Để khởi tạo một goroutine ta chỉ cần thêm phía trước một function call hay method call từ khoá go
  + - 1. **Channel**

**Channel** là để các tiến trình bất đồng bộ (trong **Go** hay gọi là **Goroutines**) có thể giao tiếp với nhau mà không cần phải dùng biến từ bên ngoài. Nó không được thiết kế để giải quyết mọi trường hợp của concurrent nhưng nó giúp việc này dễ dàng, an toàn và vui hơn. Thậm chí tới những trường hợp không cần dùng channel nhưng các gopher vẫn dùng channel chỉ vì họ thực sự thích nó.

* + - 1. **Mutex**

Mutex là hình thức ngắn gọn để loại trừ lẫn nhau. Mutex được sử dụng khi bạn không muốn một tài nguyên bị truy cập bởi nhiều [Routine](https://vngeeks.com/go-goroutine/) con trong cùng một thời điểm. Mutex có 2 phương thức Lock() và Unlock(). Mutex được đặt trong package sync. Nên chúng ta phải import package sync. Các câu lệnh phải được thực hiên riêng rẽ lẫn nhau có thể được đặt trong mutex.Lock() hoặc mutex.Unlock(), do đó tránh được điều kiện Race.

* + 1. **Uu nhuoc diem**

**Uu điểm**

**Tiện ích**

Không phải tự nhiên là Golang lại được so sánh với nhiều loại ngôn ngữ lập trình đến thế. Nó thậm chí còn không thua kém những loại ngôn ngữ kịch bản như Python nhờ khả năng đáp ứng vô vàn những nhu cầu lập trình thường thấy.

Ngôn ngữ Golang sở hữu một tính năng nổi bật và độc quyền mang tên goroutines. Goroutines tồn tại như một công cụ tích cực giúp giải quyết rất nhiều vấn đề còn tồn tại. Nó có thời gian khởi động nhanh hơn threads thông thường. Công cụ này cũng sở hữu đa kênh và có khả năng cho phép sự giao tiếp giữa các kênh này. Ngoài ra, goroutines còn có mutex locking, một tính năng cho phép khóa lại các cấu trúc dữ liệu để việc đọc và ghi nhớ không xảy ra xung đột.

#### Tốc độ

Tốc độ nhị phân của ngôn ngữ Golang đương nhiên sẽ có thể chậm hơn C++. Tuy nhiên hiệu suất làm việc giữa chúng lại không có chênh lệch nhiều. Và Golang cũng nhanh hơn các ngôn ngữ phổ biến khác như Java, Python hay Ruby… Vì thế đây được coi như một điểm cộng của Golang.

#### Linh hoạt

Không cần có sự xuất hiện bắt buộc của external dependencies, các tệp dữ liệu executable được tạo bằng toolchain của Golang vẫn có thể tự hoạt động một cách trơn tru. Toolchain của Go cho phép hỗ trợ rất nhiều hệ điều hành cũng như các loại phần cứng khác nhau và cũng được dùng như một công cụ biên dịch hệ nhị phân qua các nền tảng.

#### Tương thích

Một trong những điều kiện cần thiết ở một ngôn ngữ lập trình là sự tương thích của nó đối với những hệ thống khác nhau. Ngôn ngữ Golang có thể liên kết với thư viện C bên ngoài lẫn hệ thống native bên trong. Các liên kết được thực hiện có khả năng bổ trợ cho hiệu suất của nhau. Từ đó rút gọn thời gian phát triển phần mềm.

#### Hỗ trợ

Golang được liên kết với rất nhiều bên thứ ba khác nhau. Từ đó mang đến cho người dùng những trải nghiệm mới lạ và độc đáo. Ba nền tảng lớn nhất mà Go có cơ hội góp mặt là Linux, MacOS và Windows. Việc Toolchain của Go luôn có sẵn trong thư viện của những hệ điều hành này giúp cho những newbie triển khai chương trình Golang lần đầu tiên không còn gặp khó khăn nữa. Go cũng tích cực hỗ trợ cho các môi trường phát triển được sử dụng nhiều như Microsoft Visual Studio Code và Komodo IDE của ActiveState.

**Nhuoc điểm**

Nhược điểm đầu tiên phải kể đến là ngôn ngữ Go không hỗ trợ công cụ Generics. Điều này làm người dùng đánh mất đi cơ hội kiến tạo nên những code rõ ràng. Thay vào đó, các mã được tạo ra rất khó hiểu và trừu tượng. Go vẫn đang trong quá trình xem xét trang bị thêm Generics. Điều này có thể không cần thiết với những nhà lập trình lão làng, những người vốn không ngại những đoạn mã khó.

Tiếp theo phải kể đến quy tắc vô cùng nghiêm ngặt của Go. Đối với nhiều người đây là lợi thế bởi nó giúp hệ thống hóa các quy trình bạn đang thực hiện. Tuy nhiên nó cũng sẽ giới hạn khả năng sáng tạo của người dùng. Bạn có thể khắc phục bằng cách ghi đè lên những quy tắc này bằng những plugin sáng tạo.

Nhược điểm cuối cùng của ngôn ngữ Golang đến từ sự tối giản của nó. Golang rất nhẹ và nhỏ gọn. Tuy nhiên chính điều đó đôi khi gây ra sự quá tải trong chu trình vận hành. Thư viện của Golang cũng không rộng rãi như các ngôn ngữ khác. Điều đó làm hạn chế khả năng lưu trữ của người dùng.

1. **heroku** 
   1. **gioi thiệu chung**

Heroku là một nền tảng đám mây cho phép các cá nhân, doanh nghiệp xây dựng, triển khai, quản lý và mở rộng ứng dụng. Lợi thế chính của nền tảng này nằm ở tính linh hoạt và dễ sử dụng, đây là giải pháp giúp các nhà phát triển đưa ứng dụng của họ ra thị trường đơn giản và nhanh chóng nhất

Được quản lý bởi một công ty cùng tên, Heroku mang lại một trải nghiệm tuyệt vời cho nhà phát triển ứng dụng, khi họ chỉ cần tập trung vào phát triển, nâng cấp sản phẩm cốt lõi của mình mà không bị phân tâm trong việc duy trì hoạt động máy chủ, phần cứng hoặc các cơ sở hạ tầng.



Heroku còn được biết đến như một nơi để triển khai các ứng dụng trong dyno. Khi bạn bắt đầu triển khai một ứng dụng Heroku, bạn phải thiết lập một máy ảo được gọi là slug, sử dụng một hoặc nhiều buildpacks. Lúc này, khi một máy ảo được khởi chạy từ slug sẽ được gọi là dyno.

Đối với trường hợp phải xử lý nhiều công việc phức tạp, bạn sẽ phải thêm một số block (scale chiều ngang) hoặc tăng thêm kích thước cho các block (scale chiều dọc). Lúc này, phí sử dụng Heroku sẽ được tính dựa trên số lượng dyno cũng như kích thước của mỗi dyno mà bạn đang dùng.

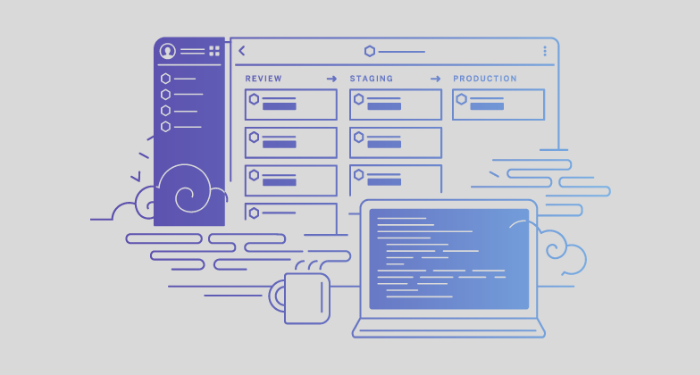
Có thể bạn sẽ phải trả phí nếu muốn sử dụng đầy đủ tính năng của Heroku, nhưng thực tế Heroku không hề lưu trữ ứng dụng của bạn mà cả nền tảng Heroku và tất cả các ứng dụng được phát triển trên đó đều triển khai trên Amazon Web Services (AWS).

Một số ngôn ngữ lập trình mà Heroku cơ khả năng hỗ trợ gồm: NodeJS, Ruby, Python,[PHP](https://wiki.tino.org/chon-phien-ban-php-cho-website-tren-directadmin/),[Java](https://wiki.tino.org/html-css-javascript-la-gi/), Scala, Clojure, Go, Kotlin,…Bên cạnh đó, Heroku còn hỗ trợ cả Postgresql database.

* 1. **cach hoạt động**

Heroku có cách thức hoạt động khá giống với các dịch vụ git (github, gitlab). Để sử dụng nền tảng này, bạn cần phải đăng ký một tài khoản, sau đó tạo các git repository (heroku gọi là app) và dùng các git command để đẩy code lên. Điểm khác biệt của Heroku so với các dịch vụ git ở chỗ với mỗi git repository trên Heroku bạn sẽ có một domain dạng https://ten-repository.herokuapp.com – đây là nơi show demo cho repository của bạn.

Ngoài ra, bạn cũng có thể sử dụng một git repository trên github để deploy lên Heroku. Tuy nhiên, bạn không nên sử dụng Heroku để thay thế cho các dịch vụ git như github hay gitlab.



* 1. **cac khai niệm co bản**

### **Heroku Runtime**

Hỗ trợ cung cấp các smart container mà ứng dụng của bạn sẽ chạy trong đó. Bên cạnh đó, tính năng này còn giúp xử lý mọi thứ quan trọng trong ứng dụng bao gồm cấu hình, điều phối, cân bằng tải, backup, log, bảo mật…

### **Heroku Postgres (SQL)**

Đây là một dịch vụ cơ sở dữ liệu[SQL](https://wiki.tino.org/mysql-la-gi/) được quản lý trực tiếp bởi Heroku nên rất đáng tin cậy và rất an toàn cho người dùng. Chúng được thiết lập dễ dàng với mã hóa nhanh chóng, mở rộng quy mô đơn giản,…

### **Scale**

Tính năng Scale mang lại cho bạn khả năng mở rộng quy mô của ứng dụng ngay lập tức, theo cả chiều dọc lẫn chiều ngang. Hơn nữa, bạn có thể điều hành mọi thứ một cách đơn giản từ các dự án sở thích nhỏ lẻ cho đến thương mại điện tử cấp doanh nghiệp.

### **Heroku Redis**

Hỗ trợ dịch vụ Redis phục vụ cho lập trình viên sử dụng. Dịch vụ này cho phép lưu trữ trên bộ nhớ với tốc độ nhanh phổ biến

### **Add-ons**

Hỗ trợ bạn quản lý các ứng dụng với dịch vụ được tích hợp sẵn như New Relic, Mongo DB, SendGrid, Searchify, Fasty, ClearDB,…

### **Code/data rollback**

Giúp người dùng có thể dễ dàng khôi phục mã nguồn hoặc cơ sở dữ liệu của mình về trạng thái trước đó.

### **App Metrics**

Đây là tính năng giúp giám sát lưu lượng, thời gian phải hồi, bộ nhớ, tải CPU và lỗi. Giúp bạn quản lý ứng dụng chặt chẽ

### **GitHub Integration**

Giúp bạn có thể thực hiện các tính năng như pull request, push, commit…hoạt động với mọi branch mà bạn muốn.

### **Continuous Delivery**

Heroku Flow sử dụng Heroku Pipeline, review Apps và Githus hỗ trợ xây dựng quy trình pipeline CI/CD gồm xây dựng, test, deploy

* 1. **uu nhuoc điểm**

**uu điểm**

* [Database](https://wiki.tino.org/database-la-gi/) hoàn toàn miễn phí
* Cung cấp nhiều gói xây dựng đa dạng
* Chứa nhiều ngôn ngữ lập trình phổ biến
* SSL sử dụng miễn phí
* Có khả năng hỗ trợ làm việc nhóm
* Cho phép liên kết với các loại Github đơn giản nhất.

**nhuoc điểm**

Heroku chỉ cho bạn sử dụng miễn phí 550 giờ/tháng. Tuy nhiên, bạn có thể gia tăng số giờ sử dụng lên đến 1000 giờ/tháng qua việc bạn trả phí. Với 1000 giờ này sẽ đủ để cho một blog cá nhân hoạt động mượt mà.

Chỉ sau từ 2 đến 3 giờ ,nếu server không có người truy cập thì nó sẽ tự động chuyển sang trạng thái ngủ.

1. **goland** 
   1. **gioi thiệu chung**

Phần mềm GoLand JetBrains là một IDE đa nền tảng được xây dựng đặc biệt cho các nhà phát triển Go. Việc phân tích mã tĩnh kết hợp với thiết kế ergonomic giúp phát triển không chỉ có hiệu quả mà còn là trải nghiệm thú vị. Các công cụ thiết yếu như rest-runner, coverage tool, full-featured debugger và tích hợp kiểm soát phiên bản – không bao gồm plugin.

* 1. **Chuc nang chinh**

**Code Insight mạnh mẽ, GoLand giúp bạn dễ dàng đọc, viết và thay đổi mã Go**

Phát hiện lỗi nhanh chóng và đề xuất các bản sửa lỗi, tái cấu trúc nhanh chóng và an toàn với hoàn tác một bước, hoàn thành mã thông minh, phát hiện mã chết và gợi ý tài liệu giúp tất cả các nhà phát triển Go, từ người mới đến các chuyên gia có kinh nghiệm, tạo ra nhanh chóng, hiệu quả, và mã đáng tin cậy.

**Hệ thống plugin IntelliJ**

Trong trường hợp bạn cần bất cứ thứ gì ngoài các bộ công cụ đã được xây dựng sẵn, IntelliJ plugin ecosystem sẽ giúp bạn.

Điều hướng & Tìm kiếm – Chuyển giữa các loại, tệp và các ký hiệu khác

Việc khám phá và tìm hiểu nhóm, kế thừa hoặc các dự án nước ngoài cần rất nhiều thời gian và công sức. Điều hướng mã GoLand giúp bạn chuyển đổi nhanh chóng sang các phương pháp ẩn, triển khai, sử dụng, khai báo hoặc giao diện được triển khai theo loại. Chuyển giữa các loại, tệp hoặc bất kỳ ký hiệu nào khác, hoặc tìm cách sử dụng của chúng và kiểm tra chúng bằng cách phân nhóm thuận tiện theo loại sử dụng.

**Run & Debug – Các công cụ tích hợp mạnh mẽ giúp chạy và gỡ lỗi các ứng dụng của bạn**

Các công cụ tích hợp mạnh mẽ giúp chạy và gỡ lỗi các ứng dụng của bạn. Bạn có thể viết và gỡ lỗi các bài kiểm tra mà không cần bất kỳ plugin bổ sung hoặc nỗ lực cấu hình nào và kiểm tra các ứng dụng của bạn ngay trong IDE. Công cụ Code Coverage được tích hợp sẵn sẽ đảm bảo rằng các thử nghiệm của bạn không bỏ lỡ bất kỳ điều gì quan trọng. IDE phân tích mã của bạn, tìm kiếm các kết nối giữa các symbol, để cung cấp code completion, điều hướng nhanh, phân tích lỗi thông minh, định dạng và tái cấu trúc.

**Tích hợp với VCSs – Hỗ trợ ngoài cho Git, GitHub và Mercurial**

GoLand cung cấp hỗ trợ ngoại vi cho Git, GitHub và Mercurial. Perforce, ClearCase và những người khác được hỗ trợ bởi các plugin người dùng có thể cài đặt.  
Không chỉ là một Go IDE.

GoLand sẽ không thể là một IDE thực sự nếu không có một bộ công cụ phong phú, ngoài việc phát triển cốt lõi của Go, hỗ trợ JavaScript, TypeScript, NodeJS, SQL, Databases, Docker, Kubernetes và Terraform. Tất cả cùng nhau, những khả năng này làm cho nó được trang bị hoàn hảo để làm việc trên bất kỳ tác vụ nào, có thể là một ứng dụng web hiện đại hoặc các công cụ DevOps.

* 1. **Khả năng mở rộng**

Hệ sinh thái phong phú của GoLand bao gồm hơn 1000 plugin có thể được sử dụng để điều chỉnh IDE cho phù hợp với nhu cầu cụ thể của bạn. Tùy chỉnh, mở rộng và thay đổi mọi thứ theo sở thích của bạn: từ việc thêm hỗ trợ cho các dự án Angular và Vue.js, hoặc nhận thanh tiến trình Nyan Cat, đến thiết lập một chủ đề IDE hoàn toàn khác.

1. **git** 
   1. **giới thiệu chung**

Git là tên gọi của một Hệ thống quản lý phiên bản phân tán (Distributed Version Control System – DVCS) là một trong những hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay. DVCS nghĩa là hệ thống giúp mỗi máy tính có thể lưu trữ nhiều phiên bản khác nhau của một mã nguồn được nhân bản (clone) từ một kho chứa mã nguồn (repository), mỗi thay đổi vào mã nguồn trên máy tính sẽ có thể ủy thác (commit) rồi đưa lên máy chủ nơi đặt kho chứa chính. Và một máy tính khác (nếu họ có quyền truy cập) cũng có thể clone lại mã nguồn từ kho chứa hoặc clone lại một tập hợp các thay đổi mới nhất trên máy tính kia. Trong Git, thư mục làm việc trên máy tính gọi là Working Tree.

* 1. **các lệnh cơ bản**

**1. git config**

Tác dụng : Để set user name và email của bạn trong main configuration file. Để kiểm tra tên và kiểu email trong cấu hình dùng:

$ git config -- global user.name

$ git config -- global user.email.

Để set email hoặc tên mới:

$ git config -- global user.name = “Anh Nguyen”

$ git config -- global user.email = “anh98052@gmail.com”

**2. git init**

$ git init

Nếu như bạn muốn theo dõi một dự án cũ trong Git, bạn cần ở trong thư mục của dự án đó. Lệnh này sẽ tạo một thư mục mới có tên .git, thư mục này chứa tất cả các tập tin cần thiết cho kho chứa.

**3. git clone**

Copy 1 git repository từ remote source.

$ git clone https://github.com/user/repository.git

Câu lệnh trên sẽ tạo một thư mục mới có tên giống trên của repo.

**4. git status**

$ git status Để check trạng thái của những file bạn đã thay đổi trong thư mục làm việc. VD: Tất cả các thay đổi cuối cùng từ lần commit cuối cùng.

Cách dùng: git status trong thư mục làm việc.

**5. git add .**

Sau khi bạn thay đổi source code: thêm mới, sửa, xoá files,… Bạn cần phải cập nhật lên Staging Area. Để cập nhật hết các files:

$ git add .

**6. git commit** Sau lệnh add, bạn cần sử dụng câu lệnh Commit để đây thông tin thay đổi lên Local Respository:

$ git commit -m ”Đây là message, bạn dùng để note những thay đổi để sau này dễ dò lại”

Commit nghĩa là một action để Git lưu lại một snapshot của các sự thay đổi trong thư mục làm việc. Và các tập tin, thư mục được thay đổi đã phải nằm trong Staging Area. Mỗi lần commit nó sẽ được lưu lại lịch sử chỉnh sửa của code kèm theo tên và địa chỉ email của người commit. Ngoài ra trong Git bạn cũng có thể khôi phục lại tập tin trong lịch sử commit của nó để chia cho một branch khác, vì vậy bạn sẽ dễ dàng khôi phục lại các thay đổi trước đó.

**7. git push**

Sau câu lệnh Commit, thông tin mới chỉ được cập nhật lên Local Repository. Nếu muốn cập nhật lên server thì bạn phải sử dụng câu lệnh push:

$ git push origin <name\_branch>

Ngoài ra, nếu chưa tồn tại remote trên server thì bạn cần phải add mới một remote trước rồi mới push:

$ git remote add origin <remote\_url>

$ git push origin <name\_branch>

**8. git pull**

Pull từ remote repository

$ git pull origin master

Lệnh trên sẽ gộp những thay đổi mới kéo về từ máy chủ từ xa với nhánh hiện tại trên máy local.

**9. git branch**

* Khi sử dụng Git, bạn có thể tạo ra nhiều nhánh (branch) khác nhau. Câu lệnh Git này dùng để kiểm tra branch hiện tại:

$ git branch

-Để tạo mới một branch:

$ git branch <name\_branch>

* Để chuyển và tạo mới:

$ git branch -b <name\_branch>

**10. git checkout** Chuyển sang branch khác:

$ git checkout <name\_branch>

**11. git merge**

Sau một thời gian cập nhật các file và push lên git trên branch mới, bây giờ mình cần ghép (merge) code lại vào nhánh gốc (master). Trước tiên, cần phải checkout ra khỏi branch hiện tại cần gộp để vào branch master, sau đó thì dùng lệnh merge để ghép branch mới vào master:

$ git checkout master

$ git merge <new\_branch>

**12. git log**

$ git log

Lệnh git log sẽ cho bạn xem lại lịch sử commit: biết về người commit, ngày giờ, message của những lần commit đó.

**13. git diff** Xem thay đổi trước khi push $ git diff Lệnh này giúp bạn biết những gì đã được thay đổi giữa nhánh hiện tại và nhánh trước nó.

**14. git rebase**

$ git rebase -i HEAD~

LỆnh này giúp bạn gộp commit. Sau dấu ~ là số commit bạn muốn gộp. Sau khi gõ lệnh này một cửa sổ trình soạn thảo hiện ra. Thay đổi ký tự pick của dòng các dòng sau dòng đầu thành s rồi lưu lại/kết thúc. Khi đó, trình soạn thảo để chỉnh sửa giải thích commit thiết lập cho commit sau khi đã tổng hợp sẽ được hiển thị, nên hãy chỉnh sửa lưu lại/kết thúc.

* 1. **ưu nhược điểm**

**ưu điểm**

* + - 1. **Sắp xếp công việc tốt hơn**
      2. **Linh hoạt hơn khi phải làm cùng lúc nhiều task**
      3. **Tự tin hơn khi thử nghiệm những ý tưởng mới**
      4. **Git cho phép chúng ta làm việc offline**
      5. **Git miễn phí**

**Nhược điểm**

## **Điểm Yếu SHA1**

Thời gian trôi đi, những nhà mật mã đã phát hiện ra ngày càng nhiều điểm yếu của thuật toán SHA1. Thực tế người ta đã đã phát hiện thấy sự va chạm giá trị băm. Trong khoảng vài năm, có lẽ những chiếc PC thông thường cũng đủ sức để âm thầm làm hư hỏng một kho Git.

Hy vọng là Git sẽ chuyển sang sử dụng hàm băm tốt hơn trước khi có người tìm ra cách phá mã SHA1.

## **Microsoft Windows**

Sử dụng Git trên hệ điều hành Microsoft Windows có vẻ hơi cồng kềnh một chút:

* [Cygwin](http://cygwin.com/), mô phỏng Linux dành cho Windows, có chứa [Git đã chuyển đổi để chạy trên Windows](http://cygwin.com/packages/git/).
* [Git cho Windows](https://gitforwindows.org/) là một thay thế với các hỗ trợ tối thiểu nhất, bởi vì chỉ cần một ít lệnh để thực hiện một số việc mà thôi.

## **Các Tập tin Không liên quan**

Nếu dự án của bạn rất lớn và chứa rất nhiều tập tin không có liên quan mà luôn luôn bị thay đổi, Git có thể chịu thiệt thòi hơn các hệ thống khác bởi vì các tập tin không được giữ dấu viết từng cái riêng lẻ. Git giữ các dấu vết thay đổi cho toàn bộ dự án, điều này thường là có lợi.

Giải pháp là chia nhỏ dự án của bạn ra, mỗi một phần bao gồm các tập tin liên quan đến nhau. Hãy sử dụng **git submodule** nếu bạn vẫn muốn giữ mọi thứ trong một kho chung.

## **Ai Sửa và Sửa gì?**

Một số hệ thống quản lý mã nguồn bắt buộc bạn đánh dấu rõ ràng vào tập tin theo một cách nào đó trước khi biên soạn. Trong khi mà điều này đặc biệt phiền toái vì nó lại dính líu đến việc phải liên lạc với máy chủ trung tâm, việc làm này có hai lợi ích:

1. Thực hiện lệnh diff diễn ra nhanh bởi vì nó chỉ kiểm tra các tập tin đã đánh dấu.
2. Một người có thể biết được khác đang làm việc trên một tập tin bằng cách hỏi máy chủ trung tâm ai đã đánh dấu là đang sửa.

Với một đoạn kịch bản thích hợp, bạn có thể lưu giữ theo cách này với. Điều này yêu cầu sự hợp tác từ người lập trình, người có thể chạy các kịch bản chuyên biệt khi biên soạn một tập tin.

## **Lịch Sử Tập Tin**

Sau khi Git ghi lại các thay đổi cho các dự án lớn, việc cấu trúc lại lịch sử của một tập tin đơn lẻ yêu cầu phải làm việc nhiều hơn các chương trình quản lý mã nguồn giữ dấu vết theo các tập tin riêng lẻ.

Thiệt hại thường là không đáng kể, và thứ đáng giá mà nó nhận được là các tác vụ khác hoạt động hiệu quả đến không ngờ. Ví dụ, git checkout nhanh hơn cp -a, và dữ liệu trong dự án lớn nén tốt hơn việc gom lại từ tập tin cơ bản.

## **Khởi tạo Bản Sao**

Việc tạo một bản sao có vẻ hơi xa xỉ hơn là việc checkout trong các hệ thống quản lý mã nguồn khác khi phần mềm có lịch sử phát triển lâu dài.

Cái giá phải trả ban đầu là cần nhiều thời gian để lấy về, nhưng nếu đã làm như thế, các tác vụ cần làm sau này sẽ nhanh chóng và không cần có mạng. Tuy nhiên, trong một số hoàn cảnh, cách làm phù hợp hơn là tạo một bản sao không đầy đủ bằng tùy chọn --depth. Điều này giúp ta tạo bản sao nhanh hơn, nhưng bản sao nhận được sẽ thiếu đi một số chức năng do đó bạn sẽ không thể thực thi được một số lệnh.

## **Các Dự Án Hay Thay Đổi**

Git được viết ra với mục đích chú tâm đến kích thước tạo ra bởi các thay đổi. Con người chỉ tạo ra sự thay đổi rất nhỏ giữa các phiên bản. Như là bổ xung lời nhận xét là có sửa lỗi ở đây, có đặc tính mới ở đây, sửa lỗi chú thích, v.v.. Nhưng nếu các tập tin của bạn căn bản khác nhau, thì trong mỗi lần commit, nó sẽ ghi lại toàn bộ các thay đổi vào lịch sử và làm cho dự án của bạn tất yếu sẽ tăng kích cỡ.

Không có bất kỳ một hệ thống quản lý mã nguồn nào có thể làm được điều này, nhưng những người sử dụng Git theo tiêu chuẩn sẽ còn phải chịu tổn thất hơn khi lịch sử của nó được nhân bản.

Đây là lý do tại sao các thay đổi quá lớn cần được xem xét. Định dạng các tập tin có thể bị thay đổi. Các thay đổi nhỏ chỉ xảy ra phần lớn tại một số ít tập tin.

Việc xét đến việc sử dụng cơ sở dữ liệu hay các giải pháp sao-lưu/lưu-trữ có lẽ là thứ có vẻ thực tế hơn, không nên dùng hệ thống quản lý mã nguồn. Ví dụ, quản lý mã nguồn không thích hợp cho việc quản lý các ảnh được chụp một cách định kỳ từ webcam.

Nếu các tập tin thực sự thay đổi thường xuyên và chúng cần phải quản lý, việc xem xét khả năng sử dụng Git hoạt động như một hệ thống quản lý tập trung là có thể chấp nhận được. Một người có thể tải về một bản sao không đầy đủ, nó chỉ lấy về một ít hay không lấy về lịch sử của dự án. Dĩ nhiên, nhiều công cụ dành cho Git sẽ không thể hoạt động được, và sự sửa chữa phải được chuyển lên như là các miếng vá. Điều này chắc chắn là tốt và nó giống như là ta không thể hiểu nổi tại sao một số người lại muốn có được lịch sử của rất nhiều các tập tin chẳng hoạt động ổn định.

Một ví dụ khác là dự án phụ thuộc vào firmware, cái này có dạng thức là tập tin nhị phân có kích thước rất lớn. Người sử dụng không quan tâm tới lịch sử của firmware, vả lại khả năng nén của nó lại cũng rất ít, vì vậy quản lý firmware có lẽ là không cần thiết vì nó làm phình to kích thước kho chứa.

Trong trường hợp này, mã nguồn có thể lưu giữ trong kho Git, và tập tin nhị phân được giữ ở nơi khác. Để cho công việc trở nên dễ dàng hơn, một người có thể tạo ra một đoạn kịch bản mà nó sử dụng Git để nhân bản mã nguồn, và dùng lệnh rsync hay Git lấy về firmware.

## **Bộ Đếm**

Một số hệ quản trị mã nguồn tập trung duy trì một số nguyên dương tự động tăng lên khi có lần commit mới được chấp nhận. Git quy các thay đổi này bởi giá trị băm của chúng, điều này là tốt trong phần lớn hoàn cảnh.

Nhưng một số người thích có nó ở dạng số nguyên. May mắn thay, rất dễ dàng để viết các đoạn kịch bản làm được như thế với mỗi lần cập nhật, kho Git trung tâm Git gia một số nguyên, có thể là trong một thẻ (tag), và kết hợp nó với giá trị băm của lần commit cuối.

Mỗi bản sao có thể có một bộ đếm riêng, nhưng điều này chẳng ích lợi gì, chỉ có kho chứa trung tâm và bộ đếm của nó là có ý nghĩa với mọi người.

## **Với Thư Mục Rỗng**

Các thư mục rỗng không được theo dõi. Tạo ra các thư mục giả để thử trục trặc này.

Xét về mặt thi hành của Git, thay vì thiết kế của nó, điều hạn chế này này là đáng trách. Với một chút may mắn, một khi Git thấy có thêm lợi ích từ việc này, thêm nhiều người đòi hỏi tính năng này và nó sẽ được thực hiện.

## **Lần Commit Khởi tạo**

Hệ thống số đếm khoa học của máy tính đếm từ 0, thay vì 1. Thật không may, có liên quan đến các lần commit, Git không tôn trọng quy ước này. Rất nhiều lệnh bất lợi trước lần commit khởi tạo. Thêm nữa, các trường hợp ngoại lệ phải được xử lý theo một cách đặc biệt, như là việc rebasing một nhánh với lần commit khởi tạo khác.

Git có thể có được lợi ích từ việc định nghĩa lần commit zero: ngay khi kho chứa được tạo ra, HEAD được đặt cho một chuỗi ký tự bao gồm 20 byte rỗng. Lần commit đặc biệt này tương ứng với một cây (tree) rỗng, không có gốc, tại một thời điểm được đề lùi về trước.

Sau đó chạy lệnh git log, ví dụ thế, thì Git nên báo cho người dùng biết chưa có lần commit nào, thay vì phát ra một lỗi nghiêm trọng. Điều tương tự xảy ra với các công cụ khác.

Tất cả các bản commit đầu tiên hoàn toàn là con cháu của bản 0 (zero).

Tuy nhiên, ở đây có một số vấn đề xảy ra trong một số trường hợp đặc biệt. Nếu nhiều nhánh với các lần khởi tạo commit khác nhau được trộn với nhau, sau đó rebase kết quả đòi hỏi thực chất có sự can thiệp bằng tay.

## **Giao diện chưa rõ ràng**

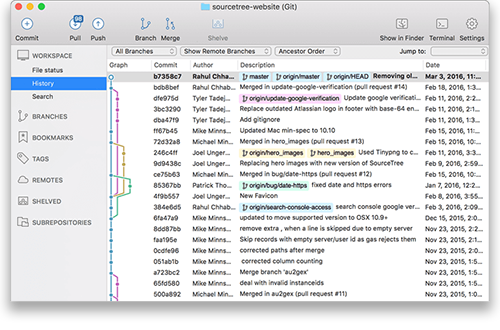
Để commit A và B, nghĩa của biểu thức "A..B" và "A…B" tùy thuộc vào việc lệnh mong đó là hai đầu mút hay là một vùng. Xem **git help diff** và **git help rev-parse**.

1. **source tree**
   1. **khái niệm chung**

**Sourcetree là một ứng dụng máy tính để bàn giao diện người dùng đồ họa (GUI) miễn phí giúp đơn giản hóa cách bạn tương tác với các kho lưu trữ Git để bạn có thể hoàn toàn tập trung vào việc viết mã**

* 1. **các đặc điểm nổi bật**
     + 1. **Tự động theo dõi hoạt động của mã của bạn.**

**Dấu trang cung cấp cho bạn cái nhìn tóm tắt về tất cả các dự án và kho lưu trữ trong thời gian thực.**



* + - 1. **Dễ dàng cho người mới bắt đầu**

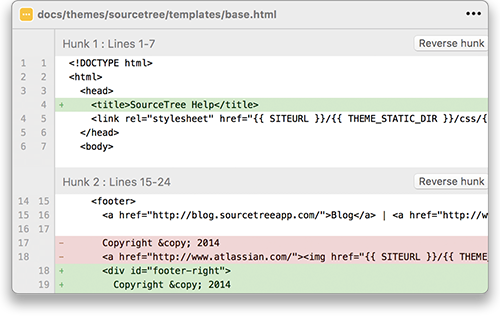
**Sourcetree là một GUI đầy đủ chức năng hoạt động với Git và Mercurial.**

**Nó tương thích với cả hệ điều hành Windows hoặc Mac.**

**Hướng dẫn chi tiết giúp bạn bắt đầu nhanh chóng.**

**Tất cả các lệnh đều có thể truy cập được chỉ với một cú nhấp chuột.**

**Nó cung cấp kiểm soát phiên bản phân tán được đơn giản hóa.**



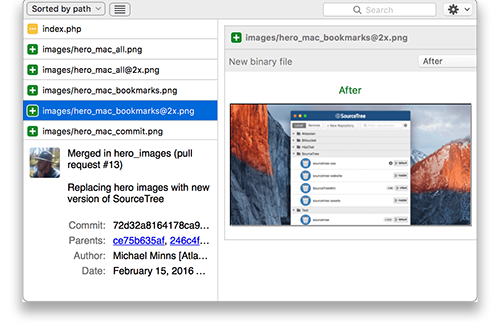
* + - 1. **Mạnh mẽ cho các chuyên gia**

**Thực hiện đánh giá và lưu các thay đổi.**

**Kiểm tra các tập thay đổi đi và đến của bạn.**

**Tùy chỉnh các bộ thay đổi và kho lưu trữ.**

**Lựa chọn tự do giữa các nhánh.**

**Áp dụng các bản vá lỗi, làm lại cơ sở hoặc đặt các bộ thay đổi của bạn lên kệ.** 

1. **tài liệu tham khảo**

[Golang là gì và tại sao bạn nên học Go? | TopDev](https://topdev.vn/blog/golang-la-gi-va-tai-sao-ban-nen-hoc-go/)

[Ngôn ngữ Golang là gì? Tại sao bạn nên sử dụng ngôn ngữ này? (teky.edu.vn)](https://teky.edu.vn/blog/ngon-ngu-golang/)

[[Go Lang] Lập trình golang căn bản - Goroutine (viblo.asia)](https://viblo.asia/p/go-lang-lap-trinh-golang-can-ban-goroutine-djeZ1pvQKWz)

[Channel trong Golang là gì? So sánh Callback function và mutex lock với channel | TopDev](https://topdev.vn/blog/golang-channel-la-gi/)

[Go - Mutex - VN GEEKS](https://vngeeks.com/go-mutex/)

[Heroku là gì? Cách đăng ký Heroku và đưa ứng dụng lên (tino.org)](https://wiki.tino.org/heroku-la-gi/)

[Phần mềm GoLand - IDE mới cho phát triển Go - Mở rộng nền tảng IntelliJ (vihoth.com)](https://vihoth.com/phan-mem-van-phong/jetbrains/phan-mem-goland-ide-moi-cho-phat-trien-go)

[Sourcetree | GlobalLogic](https://www.globallogic.com/offerings/atlassian/products/sourcetree/)