Sử dụng Git cho làm việc lập trình nhóm

15/04/2012 - 18:23

**Chia sẻ:**[Facebook](http://facebook.com/sharer.php?u=http://www.giaidieu.com/blog-cong-ty/su-dung-git-cho-lam-viec-lap-trinh-nhom&t=S%E1%BB%AD+d%E1%BB%A5ng+Git+cho+l%C3%A0m+vi%E1%BB%87c+l%E1%BA%ADp+tr%C3%ACnh+nh%C3%B3m) [Twitter](http://twitter.com/intent/tweet?url=http://www.giaidieu.com/blog-cong-ty/su-dung-git-cho-lam-viec-lap-trinh-nhom&text=S%E1%BB%AD+d%E1%BB%A5ng+Git+cho+l%C3%A0m+vi%E1%BB%87c+l%E1%BA%ADp+tr%C3%ACnh+nh%C3%B3m) [Yahoo](http://bookmarks.yahoo.com/toolbar/savebm?opener=tb&u=http://www.giaidieu.com/blog-cong-ty/su-dung-git-cho-lam-viec-lap-trinh-nhom&t=S%E1%BB%AD+d%E1%BB%A5ng+Git+cho+l%C3%A0m+vi%E1%BB%87c+l%E1%BA%ADp+tr%C3%ACnh+nh%C3%B3m&d=Git+l%C3%A0+m%E1%BB%99t+ph%E1%BA%A7n+m%E1%BB%81m+cho+ph%C3%A9p+ki%E1%BB%83m+so%C3...) [LinkedIn](http://www.linkedin.com/shareArticle?url=http://www.giaidieu.com/blog-cong-ty/su-dung-git-cho-lam-viec-lap-trinh-nhom&mini=true&title=S%E1%BB%AD+d%E1%BB%A5ng+Git+cho+l%C3%A0m+vi%E1%BB%87c+l%E1%BA%ADp+tr%C3%ACnh+nh%C3%B3m&ro=false&summary=Git+l%C3%A0+m%E1%BB%99t+ph%E1%BA%A7n+m%E1%BB%81m+cho+ph%C3%A9p+ki%E1%BB%83m+so%C3...&source=)

Git là một phần mềm cho phép kiểm soát các phiên bản (version control) được đưa lên bởi những thành viên cùng làm việc trong một nhóm tránh khả năng dữ liệu bị ghi đè, xung đột hoặc bị mất do lỗi con người.

Github (www.github.com) hiện là nơi cung cấp dịch vụ cho Git dùng rất phổ biển bởi các công ty chuyên phát triển phần mềm, làm việc theo nhóm, trên toàn thế giới. Tại Giaiđiệu, hiện tại chúng tôi cũng đang sử dụng Git để kiểm soát các phiên bản lập trình trong quá trình phát triển sản phẩm của mình.

Git chia ra làm 2 bản, một bản gọi là *Git server* được cài đặt trên máy chủ web là nơi tiếp nhận phiên bản tốt nhất cuối cùng để hoạt động chính thức. Bản còn lại gọi là *Git client* được cài trên máy cá nhân của lập trình viên để kiểm soát và cập nhật dữ liệu giữa máy của lập trình viên với máy chủ của Github. Trong bài viết này, chúng tôi xin giới thiệu một số bước căn bản để các nhà phát triển, lập trình viên có thể dễ dàng tiếp cận, cài đặt và thực hiện các thao tác với Git client, nhanh chóng hòa nhập vào một trong những công nghệ phổ biến nhất thế giới hiện nay.

**1. Tạo Github account**

Bạn có thể tạo Github account rất dễ dàng, miễn phí và nhanh chóng tại đây https://github.com/signup/free

**2. Cài đặt Github client vào máy cá nhân**

Git client cũng được cung cấp miễn phí tại đây http://help.github.com/win-set-up-git/

**3. Cấu hình Github client để làm việc với máy chủ Github**

Cấu hình Github client được hướng dẫn khá chi tiết trên trang của Github, ở đây, chúng tôi xin tóm lược lại một số các bước chính dịch sang tiếng Việt.

***3.1 Tạo SSH key***

* Khởi động Gitbash
* cd ~/.ssh
* ssh-keygen –t rsa –C “youraddress@email.com”  (Bỏ qua phần đặt tên)
* Mở file “id\_rsa.pub” trong thư mục “C:\Users\<Tên>\.ssh”, copy nội dung vào trong Github account của bạn trên máy chủ: https://github.com/settings/ssh  (Vào Add SSH key)
* Test kiểm tra: ssh –T git@github.com (Nếu nhận được dòng thông báo “Hi username! You’ve successfully authenticated, but Github does not provide shell access.” nghĩa là cài đặt đã thành công.

Xem thêm hướng dẫn cho Windows tại đây: http://help.github.com/win-set-up-git/  
  
***3.2 Cài đặt thông tin cá nhân***

* git config --global user.name “Tên của bạn”
* git config --global user.email “Địa chỉ email của bạn”

**4. Đưa file từ client lên Github**

Sau khi Git client được cài đặt và cấu hình đúng, bước tiếp theo là khởi tạo Git trên máy cá nhân của bạn và tạo ra các tương tác thực sự tới Github.

***4.1 Khởi tạo thư mục repo trên máy client (máy của lập trình viên)***

* mkdir C:\<Tên repo>
* cd <Tên repo>
* git init

***4.2 Kết nối tới Github repo***

git remote add origin git@github.com:(Tên repo)  
*Ví dụ tên repo: giaidieu-com:giaidieu.com.git*  
  
***4.3 Luôn cập nhật nội dung từ server trước khi tải nội dung mới từ client lên***

git pull git@github.com:(Tên repo) master  
*Ví dụ tên repo: giaidieu-com/giaidieu.com.git*  
  
***4.4 Kiểm tra trạng thái thay đổi***

git status  
  
***4.5 Đưa những file vào danh sách trước khi commit (bỏ qua nếu không có file mới được tạo)***

git add -A  (-A Tất cả files)  
  
***4.6 Commit những thay đổi trước khi push lên server***

git commit -a -m “Thông tin ngắn gọn về thay đổi này” (-a Tất cả thay đổi)  
  
***4.7 Tải lên server nhánh master***

git push -u origin master   
  
**5. Tạo nhánh (branch) riêng**

***5.1 Xem toàn bộ các nhánh đang có***  
git branch -a  
  
***5.2 Tạo nhánh mới***  
git branch <tên mới>  
  
***5.3 Chuyển nhánh***  
git checkout <tên nhánh>  
  
***5.4 Xóa nhánh***

git branch -d <Tên branch>  
  
**6. Nhập nhánh con vào nhánh hiện tại**

git merge <Tên nhánh>  
  
**7. Liệt kê một số lệnh hay dùng**

* touch “tên file”  (Tạo 1 file)
* echo “nội dung” > “tên file” (viết nội dung vào trong file)
* git remote add origin git@github.com:<tên repo>  (Tạo kết nối tới Github server)
* git pull git@github.com:<tên repo> <tên branch>  (Cập nhật dữ liệu từ server)
* git status (Xem trạng thái hiện tại)
* git add -A (Cộng tất cả nhưng file đã thay đổi vào danh sách để đưa lên server)
* git commit -a -m “Thông tin note”  (Đưa tất cả những file vào danh sách để chuẩn bị push)
* git push origin master  (Gửi toàn bộ file đã commit lên server vào nhánh master)

Bạn có thể tham khảo thêm thông tin về các lệnh hay dùng khác của Github tại đây <http://www.vogella.com/articles/Git/article.html>

# Git (phần mềm)

Bách khoa toàn thư mở Wikipedia

**Git** là phần mềm quản lý mã nguồn phân tán được phát triển bởi Linus Torvalds dành cho việc phát triển Linux kernel. Git là phần mềm mã mở được phân phối theo giấy phép công GPL2.

## Mục lục

  [[ẩn](http://vi.wikipedia.org/wiki/Git_(ph%E1%BA%A7n_m%E1%BB%81m))]

* [1 Hỗ trợ nhiều hệ điều hành khác nhau](http://vi.wikipedia.org/wiki/Git_(ph%E1%BA%A7n_m%E1%BB%81m)#H.E1.BB.97_tr.E1.BB.A3_nhi.E1.BB.81u_h.E1.BB.87_.C4.91i.E1.BB.81u_h.C3.A0nh_kh.C3.A1c_nhau)
* [2 Khái niệm cơ bản trong Git](http://vi.wikipedia.org/wiki/Git_(ph%E1%BA%A7n_m%E1%BB%81m)#Kh.C3.A1i_ni.E1.BB.87m_c.C6.A1_b.E1.BA.A3n_trong_Git)
* [3 Nơi lưu giữ mã nguồn](http://vi.wikipedia.org/wiki/Git_(ph%E1%BA%A7n_m%E1%BB%81m)" \l "N.C6.A1i_l.C6.B0u_gi.E1.BB.AF_m.C3.A3_ngu.E1.BB.93n)
* [4 Liên kết ngoài](http://vi.wikipedia.org/wiki/Git_(ph%E1%BA%A7n_m%E1%BB%81m)#Li.C3.AAn_k.E1.BA.BFt_ngo.C3.A0i)

## Hỗ trợ nhiều hệ điều hành khác nhau[[sửa](http://vi.wikipedia.org/w/index.php?title=Git_(ph%E1%BA%A7n_m%E1%BB%81m)&veaction=edit&section=1) | [sửa mã nguồn](http://vi.wikipedia.org/w/index.php?title=Git_(ph%E1%BA%A7n_m%E1%BB%81m)&action=edit&section=1)]

Git có khả năng chạy trên nhiều hệ điều hành khác nhau như Linux, Windows, Mac OSX v.v..

## Khái niệm cơ bản trong Git[[sửa](http://vi.wikipedia.org/w/index.php?title=Git_(ph%E1%BA%A7n_m%E1%BB%81m)&veaction=edit&section=2) | [sửa mã nguồn](http://vi.wikipedia.org/w/index.php?title=Git_(ph%E1%BA%A7n_m%E1%BB%81m)&action=edit&section=2)]

1.Repository Repository (nhà kho) hay được gọi tắt là Repo đơn giản là nơi chứa/cơ sở dữ liệu (database) tất cả những thông tin cần thiết để duy trì và quản lý các sửa đổi và lịch sử của dự án. Trong Repo có 2 cấu trúc dữ liệu chính là Object Store và Index. Tất cả dữ liệu của Repo đèu được chứa trong thư mục bạn đang làm việc dưới dạng folder ẩn có tên là.git (không có phần tên trước dấu chấm).

1.1.Object store  
Object store là trái tim của Git, nó chứa dữ liệu nguyên gốc (original data files), các file log ghi chép quá trình sửa đổi, tên người tạo file, ngày tháng và các thông tin khác.  
Git có 4 loại object là:Blobs, Trees, Commits, Tags  
+ Blobs: là file nhị phân có thể chứa được mọi loại dữ liệu bất kể là dữ liệu của chương trình gì.  
+ Trees: lớp đại diện cho thông tin thư mục như thông tin định danh của blob, đường dẫn, chứa một ít metadata chứa thông tin cấu trúc và các thư mục nhỏ có trong thư mục đó.  
+ Commits: Chứa metadata có thông tin về mọi thứ như tên tác giả, người tải lên (commiter), ngày tải lên, thông tin log...  
+ Tags: đánh dấu cho dễ đọc. thấy vì 1 cái tên dài như là 9da581d910c9c4ac93557ca4859e767f5caf5169, chúng ta có thể tên tag là Ver-1.0- Alpha. Dễ nhớ và dễ sử dụng hơn.  
  
1.2.Index  
Index là file nhị phân động và tạm thời miêu tả cấu trúc thư mục của toàn bộ Repo và trạng thái của dự án được thể hiện thông qua commit và tree tại một thời điểm nào đó trong lịch sử phát triển. Git là một hệ thống truy tìm nội dung (content tracking system). Index không hứa nội dung file mà chỉ dùng để truy tìm (track) những thứ mà bạn muốn commit.

## Nơi lưu giữ mã nguồn[[sửa](http://vi.wikipedia.org/w/index.php?title=Git_(ph%E1%BA%A7n_m%E1%BB%81m)&veaction=edit&section=3) | [sửa mã nguồn](http://vi.wikipedia.org/w/index.php?title=Git_(ph%E1%BA%A7n_m%E1%BB%81m)&action=edit&section=3)]

Có nhiều địa chỉ cho phép lưu trữ miễn phí cũng như có phí trên mạng Internet.

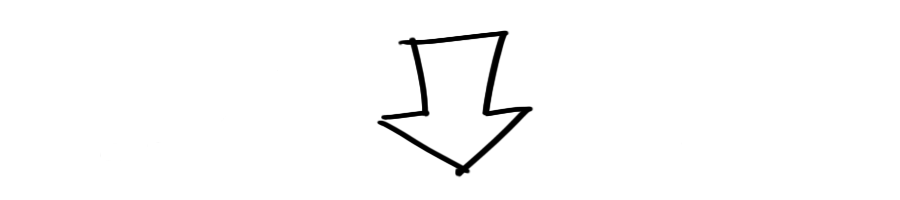
* [BerliOS](http://vi.wikipedia.org/w/index.php?title=BerliOS&action=edit&redlink=1)
* [GitHub](http://vi.wikipedia.org/w/index.php?title=GitHub&action=edit&redlink=1)
* [Gitorious](http://vi.wikipedia.org/w/index.php?title=Gitorious&action=edit&redlink=1)
* [Sourceforge](http://vi.wikipedia.org/wiki/SourceForge)
* [GNU Savannah](http://vi.wikipedia.org/w/index.php?title=GNU_Savannah&action=edit&redlink=1)
* [GitEnterprise](http://vi.wikipedia.org/w/index.php?title=GitEnterprise&action=edit&redlink=1)

# Sổ tay Git

Đây là một hướng dẫn đơn giản cho người mới học ;)

by [Roger Dudler](http://www.twitter.com/rogerdudler)   
credits to [@tfnico](http://www.twitter.com/tfnico), [@fhd](http://www.twitter.com/fhd) and [Namics](http://www.namics.com/)  
this guide in [deutsch](http://rogerdudler.github.io/git-guide/index.de.html), [español](http://rogerdudler.github.io/git-guide/index.es.html), [français](http://rogerdudler.github.io/git-guide/index.fr.html), [italiano](http://rogerdudler.github.io/git-guide/index.it.html), [nederlands](http://rogerdudler.github.io/git-guide/index.nl.html), [português](http://rogerdudler.github.io/git-guide/index.pt_BR.html), [русский](http://rogerdudler.github.io/git-guide/index.ru.html), [türkçe](http://rogerdudler.github.io/git-guide/index.tr.html),   
[မြန်မာ](http://rogerdudler.github.io/git-guide/index.my.html), [日本語](http://rogerdudler.github.io/git-guide/index.ja.html), [中文](http://rogerdudler.github.io/git-guide/index.zh.html), [한국어](http://rogerdudler.github.io/git-guide/index.ko.html) [Vietnamese](http://rogerdudler.github.io/git-guide/index.vi.html)   
vui lòng báo cáo vấn đề trên [github](https://github.com/hnq90/git-guide/issues)

[](http://frontify.com/?utm_source=gitguide&utm_campaign=founder_frontify)



## cài đặt

[Tải git về cho OSX](http://code.google.com/p/git-osx-installer/downloads/list?can=3)

[Tải git về cho Windows](http://code.google.com/p/msysgit/downloads/list?can=3)

[Tải git về cho Linux](http://book.git-scm.com/2_installing_git.html)

## tạo một repository mới

Để tạo 1 repository mới, bạn hãy mở cửa sổ lệnh và gõ dòng lệnh sau   
git init

## sao chép (clone) một repository

để clone 1 repository có sẵn ở trên máy cục bộ, bạn hãy sử dụng dòng lệnh sau:  
git clone /đường-dẫn-đến/repository/  
Nếu repository đó ở máy chủ khác thì bạn hãy gõ dòng lệnh sau:  
git clone tênusername@địachỉmáychủ:/đường-dẫn-đến/repository

## quy trình làm việc

thư mục cục bộ của bạn bao gồm ba "trees" được duy trì bởi git. đầu tiên là Thư Mục Đang Làm Việc (Working Directory) có chứa các tập tin hiện tại. cái thứ hai là Chỉ Mục (Index) đóng vai trò như staging area và cuối cùng là HEAD trỏ đến commit gần đây nhất của bạn.



## thêm (add) & commit

Bạn có thể đề xuất thay đổi (thêm nó vào chỉ mục **Index**) bằng cách  
git add <tên-tập-tin>  
git add \*  
Đây là bước đầu tiên trong quy trình git cơ bản. Để thật sự commit những thay đổi, bạn sử dụng  
git commit -m "Ghi chú Commit"  
Bây giờ thì tập tin đã được commit đến **HEAD**, nhưng chưa phải trên thư mục remote.

## đẩy (push) các thay đổi

Thay đổi của bạn hiện đang nằm tại **HEAD** của bản sao cục bộ đang làm việc. Để gửi những thay đổi đó đến repository remote, bạn thực thi  
git push origin master  
Thay đổi *master* bằng bất cứ nhánh nào mà bạn muốn đầy những thay đổi đến.   
  
Nếu bạn chưa clone một repository hiện có và muốn kết nối repository của bạn đến máy chủ remote, bạn phải thêm nó với  
git remote add origin <máy-chủ>  
Bây giờ bạn đã có thể đẩy các thay đổi của mình vào máy chủ đã chọn

## nhánh

Các nhánh (branches) được dùng để phát triển tính năng tách riêng ra từ những nhánh khác. Nhánh *master* là nhánh "mặc định" khi bạn tạo một repository. Sử dụng các nhánh khác tri đang trong giai đoạn phát triển và merge trở lại nhánh master một khi đã hoàn tất.



tạo một nhánh mới và đặt tên là "feature\_x" và chuyển qua nhánh đó (từ master) bằng cách  
git checkout -b feature\_x  
trở lại nhánh master  
git checkout master  
và xóa nhánh feature\_x đó lần nửa  
git branch -d feature\_x  
một nhánh *không có giá trị với các nhánh khác* trừ khi bạn đẩy nhánh đó đến remote repository  
git push origin <nhánh>

## cập nhật & trộn (update & merge)

để cập nhật repository cục bộ của bạn và commit mới nhất, thực thi  
git pull  
trong thự mục đang làm việc để *lấy về (fetch)* và *trộn (merge)* các thay đổi ở remote.  
để trộn một nhánh khác vào nhánh đang hoạt động (vd: master), sử dụng  
git merge <nhánh>  
trong cả hai trường hợp, git cố gắng trộn tự động (auto-merge) các thay đổi. Không may, điều này không phải lúc nào cũng làm được và thường dẫn đến *xung đột*. Trách nhiệm của bạn là trộn *các xung đột* đó thủ công bằng cách chỉnh sửa các tập tin được hiển thị bởi git. Sau khi thay đổi, bạn phải đánh dấu chúng là đã được trộn (merged) với lệnh  
git add <tên-tập-tin>  
trước khi trộn các thay đổi, bạn có thể xem trước chúng bằng các  
git diff <nhánh\_nguồn> <nhánh\_mục\_tiêu>

## gắn nhãn (tagging)

người ta khuyên nên tạo nhãn (tags) khi phát hành phần mềm. đây là khái niệm được biết đến, đã từng có trên SVN. Bạn tạo tag mới tên là*1.0.0* bằng cách  
git tag 1.0.0 1b2e1d63ff  
chuỗi *1b2e1d63ff* là 10 ký tự đầu tiên của mã commit (commit id) mà bạn muốn tham chiếu đến bằng nhãn của bạn. Bạn có thể lấy mã commit với lệnh   
git log  
bạn cũng có thể sử dụng ít ký tự hơn từ mã commit, nó chỉ cần phải là duy nhất.

## thay thế các thay đổi cục bộ

Trong trường hợp bạn làm sai điều gì đó, bạn có thể thay thế các thay đổi cục bộ bằng lệnh  
git checkout -- <tên-tập-tin>  
lệnh này thay thế những thay đổi trong "tree" đang làm việc với nội dung mới nhất của HEAD. Các thay đổi đã được thêm vào chỉ mục, kể cả các tập tin mới, điều này sẽ được giữ lại.

Nếu bạn muốn hủy tất cả thay đổi và commit cục bộ, lấy về (fetch) lịch sử gần đây nhất từ máy chủ và trỏ nhánh master cục bộ vào nó như sau  
git fetch origin  
git reset --hard origin/master

## các gợi ý hữu ích

git GUI tích hợp sẵn  
gitk  
sử dụng kết quả git với nhiều màu  
git config color.ui true  
hiện log trên chỉ một dùng mỗi commit  
git config format.pretty oneline  
sử dụng thêm tập tin tương tác  
git add -i

## các liên kết và tài nguyên

### trình sử dụng có đồ họa

* [GitX (L) (OSX, open source)](http://gitx.laullon.com/)
* [Tower (OSX)](http://www.git-tower.com/)
* [Source Tree (OSX, free)](http://www.sourcetreeapp.com/)
* [GitHub for Mac (OSX, free)](http://mac.github.com/)
* [GitBox (OSX, App Store)](https://itunes.apple.com/gb/app/gitbox/id403388357?mt=12)

### hướng dẫn

* [Git Community Book](http://book.git-scm.com/)
* [Pro Git](http://progit.org/book/)
* [Think like a git](http://think-like-a-git.net/)
* [GitHub Help](http://help.github.com/)
* [A Visual Git Guide](http://marklodato.github.com/visual-git-guide/index-en.html)
* [Learn Git Branching](http://pcottle.github.io/learnGitBranching/)
* [More Git and GitHub Secrets](http://zachholman.com/talk/more-git-and-github-secrets/)
* [Understanding Git Conceptually](http://www.sbf5.com/~cduan/technical/git/)

### misc

* [ungit - The easiest way to use git. On any platform. Anywhere.](https://github.com/FredrikNoren/ungit)

## bình luận

## Chương 9. Phụ lục A: Hạn chế của Git

Git bộc lộ một số nhược điểm mà tôi đã gặp qua. Một số có thể xử lý thủ công một cách dễ dàng bằng các đoạn kịch bản và hook, một số yêu cầu phải tổ chức lại hay xác lập lại dự án, một số ít rắc rối còn lại, chỉ còn cách là ngồi đợi. Hay tốt hơn cả là bắt tay vào và giúp đỡ họ viết!

## 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 chạy trên MSys](http://code.google.com/p/msysgit/) 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 Tệp tin Không liên quan

Nếu dự án của bạn rất lớn và chứa rất nhiều tệp 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 tệp 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 tệp 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 tệp 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 tệp tin đã đánh dấu.
2. Một người có thể biết được khác đang làm việc trên một tệp 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 tệp tin.

## Lịch Sử Tệp Tin

Sau khi Git ghi lại các thay đổi cho các dự án lớn, việc cấu trúc lại lịch sử của một tệp tin đơn lẻ yêu cầu phải làm việc nhiều hơn các chương trình quản lý mã nguồn giữ dấu vết theo các tệp 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 tệp 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 tệp 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 tệp tin có thể bị thay đổi. Các thay đổi nhỏ chỉ xảy ra phần lớn tại một số ít tệp 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 tệp 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 tệp tin chẳng hoạt động ổn định.

Một ví dụ khác là dự án phụ thuộc vào firmware, cái này có dạng thức là tệp 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 tệp 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 Lập lờ

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**.