**GIT**

1. Cấu hình git lần đầu

Việc đầu tiên bạn nên làm khi cấu hình Git là chỉ định tên tài khoản và địa chỉ e-mail. Điều này rất quan trọng vì mỗi Git sẽ sử dụng chúng cho mỗi lần commit, những thông tin này được gắn bất di bất dịch vào các commit:

$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com

Liệt kê tất cả các cài đặt của Git:

$ git config --list

1. Sao chép kho chứa có sẵn

Để sao chép một kho chứa:

$ git clone git://github.com/schacon/grit.git

 Sao chép kho chứa này vào một thư mục có tên khác:

$ git clone git://github.com/schacon/grit.git mygrit

1. Thay đổi kho chứa

Ban đầu, khi bạn tạo bản sao của một kho chứa, tất cả tập tin ở trạng thái "đã được theo dõi" (tracked) và "chưa thay đổi" (unmodified) vì bạn vừa mới tải chúng về và chưa thực hiện bất kỳ thay đổi nào.

Khi bạn chỉnh sửa các tập tin, Git coi là chúng đã bị thay đổi so với lần commit trước đó. Bạn stage các tập tin bị thay đổi này và sau đó commit tất cả các thay đổi đã được staged (tổ chức) đó, và quá trình này cứ thế lặp đi lặp lại.



**Để xem trạng thái của tập tin :**

$ git status

# On branch master

nothing to commit, working directory clean

sau khi bạn thêm mới 1 tập tin README

$ git status

# On branch master

# Untracked files:

# (use "git add <file>..." to include in what will be committed)

#

# README

nothing added to commit but untracked files present (use "git add" to track)

**Stage tập tin thành staged :**

$ git add README

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# new file: README

đôi khi sửa 1 tập tin

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# new file: README

#

# Changes not staged for commit:

# (use "git add <file>..." to update what will be committed)

#

# modified: benchmarks.rb

Stage chúng

$ git add benchmarks.rb

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# new file: README

# modified: benchmarks.rb

**Xóa tập tin :**

$ rm grit.gemspec

$ git status

# On branch master

#

# Changes not staged for commit:

# (use "git add/rm <file>..." to update what will be committed)

#

# deleted: grit.gemspec

Stage tập tin muốn xóa:

$ git rm grit.gemspec

rm 'grit.gemspec'

$ git status

# On branch master

#

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# deleted: grit.gemspec

Xóa tập tin trong thư mục log có đuôi .log:

$ git rm log/\\*.log

xoá toàn bộ tập tin kết thúc bằng ~

$ git rm \\*~

1. Xem lịch sử commit

Xem lịch sử commit bằng giao diện:

$ gitk

1. Phục hồi

**Unstage các tập tin staged:**

$ git add .

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# modified: README.txt

# modified: benchmarks.rb

$ git reset HEAD benchmarks.rb

benchmarks.rb: locally modified

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# modified: README.txt

#

# Changes not staged for commit:

# (use "git add <file>..." to update what will be committed)

# (use "git checkout -- <file>..." to discard changes in working directory)

#

# modified: benchmarks.rb

#

**Phục hồi tập tin đã thay đổi:**

# Changes not staged for commit:

# (use "git add <file>..." to update what will be committed)

# (use "git checkout -- <file>..." to discard changes in working directory)

#

# modified: benchmarks.rb

#

$ git checkout -- benchmarks.rb

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# modified: README.txt

1. Làm việc từ xa

Đến một thời điểm nào đó bạn muốn chia sẻ dự án của bạn, bạn phải đẩy ngược nó lên. Câu lệnh để thực hiện rất đơn giản:

git push [tên-máy-chủ] [tên-nhánh]

 Nếu bạn muốn đẩy nhánh master vào nhánh orgin trên máy chủ (nhắc lại, khi sao chép Git thường cài đặt/cấu hình mặc định các tên đó cho bạn), bạn có thể chạy lệnh sau để đẩy các công việc đã hoàn thành ngược lại máy chủ:

$ git push origin master

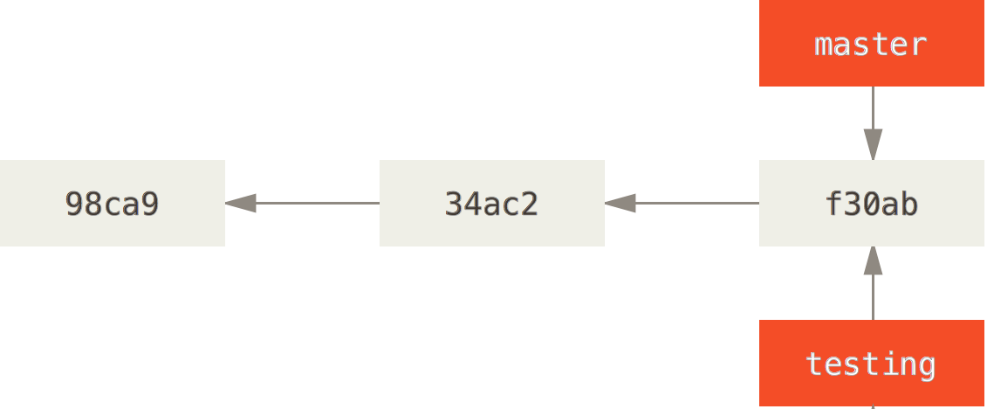
Nếu bạn và ai khác cùng sao chép tại cùng một thời điểm; người kia đẩy ngược lên, sau đó bạn cũng muốn đẩy lên, thì hành động của bạn sẽ bị từ chối ngay tức khắc. Trước hết bạn phải thực hiện kéo các thay đổi mà người đó đã thực hiện và tích hợp/gộp nó vào của bạn, sau đó bạn mới được phép đẩy lên.

1. Nhánh (branch)

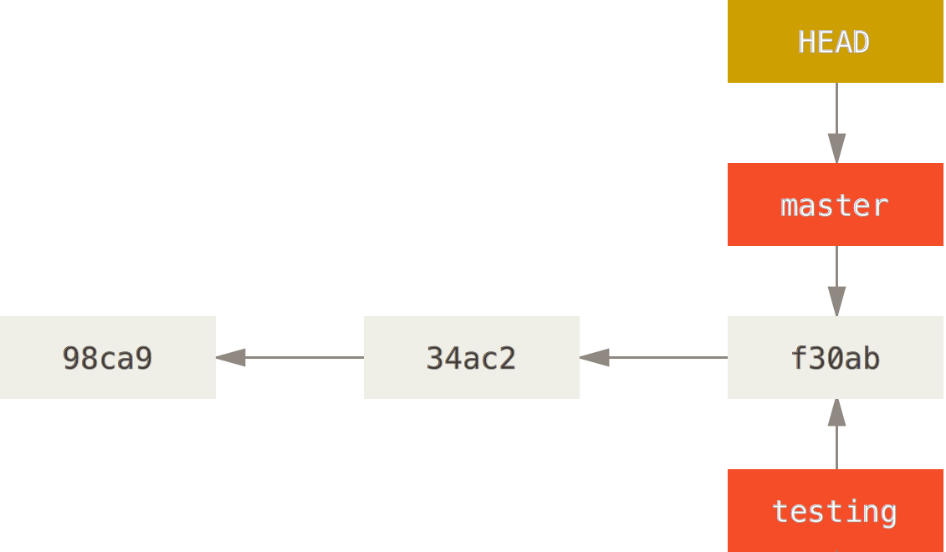
### [Creating a New Branch](http://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell#Creating-a-New-Branch)

**$** git branch testing

This creates a new pointer at the same commit you’re currently on.



How does Git know what branch you’re currently on? It keeps a special pointer called HEAD. In Git, this is a pointer to the local branch you’re currently on. In this case, you’re still on master. The git branch command only created a new branch – it didn’t switch to that branch.



### [Switching Branches](http://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell#Switching-Branches)

**$** git checkout testing

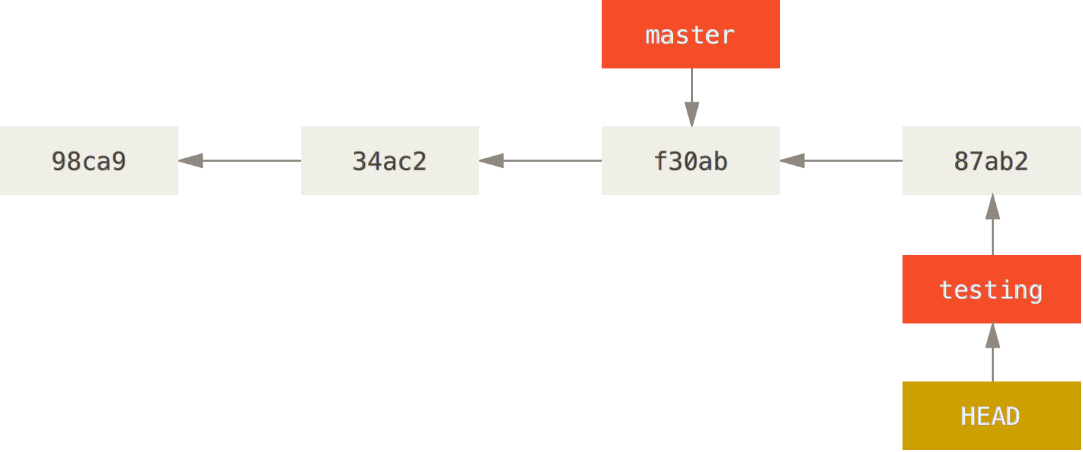
This moves HEAD to point to the testing branch.

### HEAD points to the current branch.

Well, let’s do another commit:

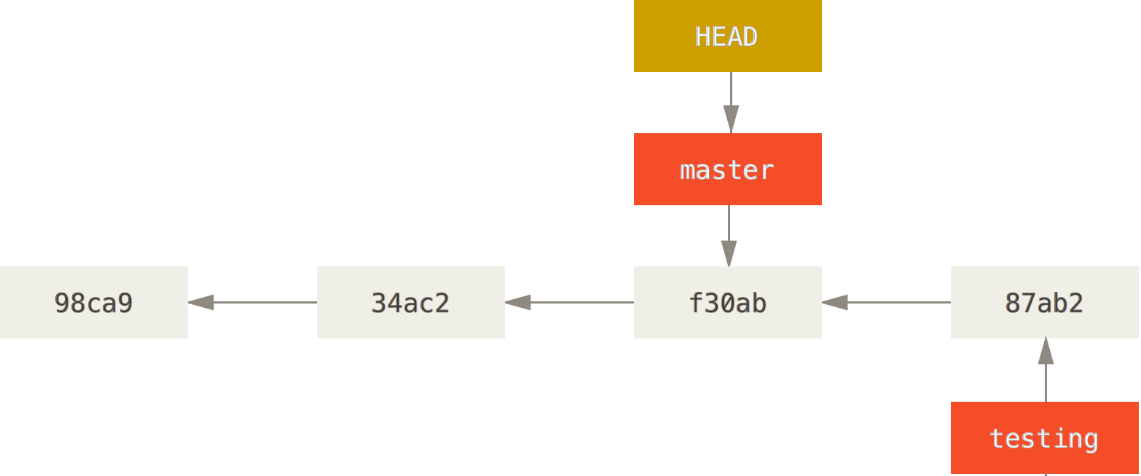
**$** vim test.rb

**$** git commit -a -m 'made a change'



This is interesting, because now your testing branch has moved forward, but your master branch still points to the commit you were on when you ran git checkout to switch branches. Let’s switch back to the master branch:

**$** git checkout master



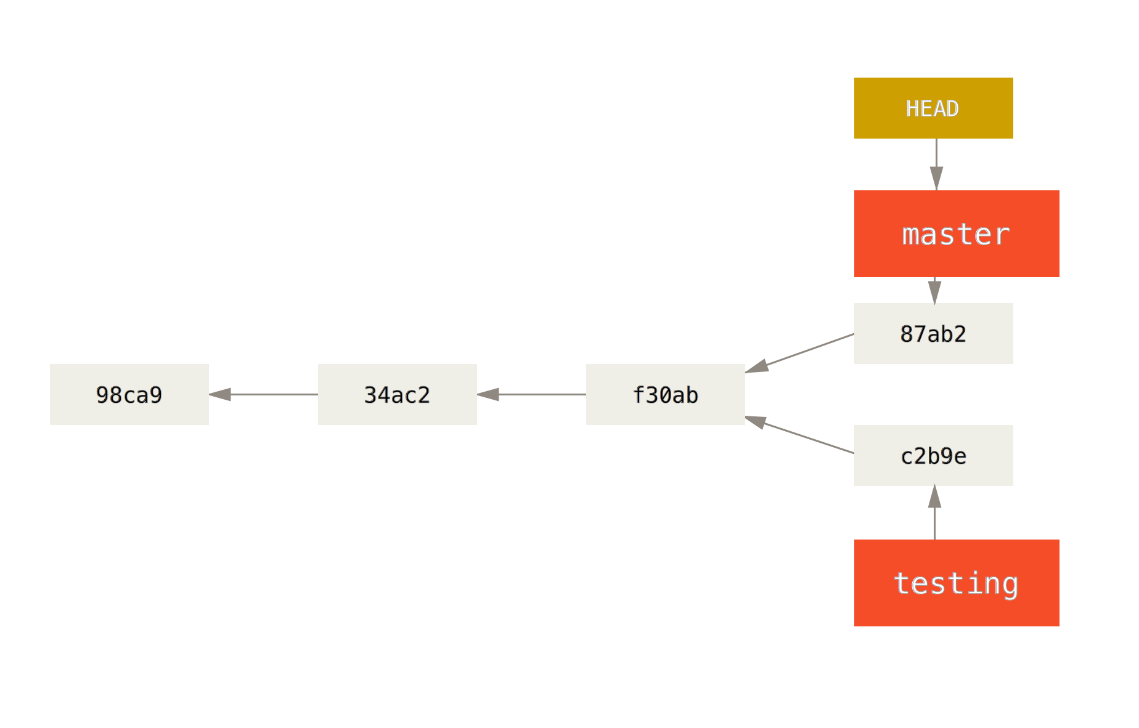
That command did two things. It moved the HEAD pointer back to point to the master branch, and it reverted the files in your working directory back to the snapshot that master points to.

Let’s make a few changes and commit again:

**$** vim test.rb

**$** git commit -a -m 'made other changes'

You created and switched to a branch, did some work on it, and then switched back to your main branch and did other work. Both of those changes are isolated in separate branches: you can switch back and forth between the branches and merge them together when you’re ready. And you did all that with simple branch, checkout, andcommit commands.



## Basic Branching and Merging

Let’s go through a simple example of branching and merging with a workflow that you might use in the real world. You’ll follow these steps:

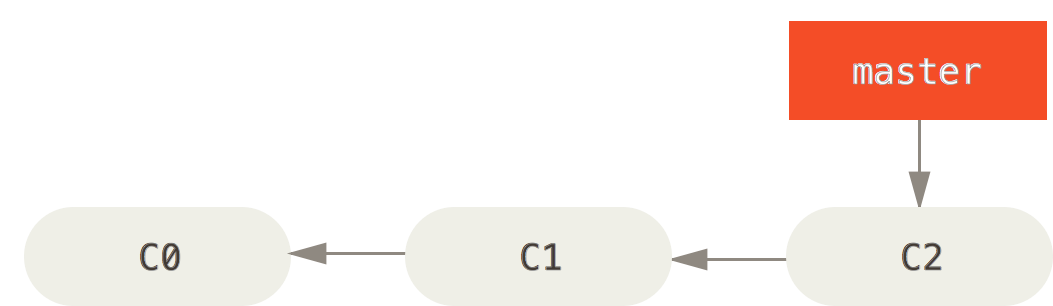
1. Do work on a web site.
2. Create a branch for a new story you’re working on.
3. Do some work in that branch.

At this stage, you’ll receive a call that another issue is critical and you need a hotfix. You’ll do the following:

1. Switch to your production branch.
2. Create a branch to add the hotfix.
3. After it’s tested, merge the hotfix branch, and push to production.
4. Switch back to your original story and continue working.

[Basic Branching](http://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging#Basic-Branching)

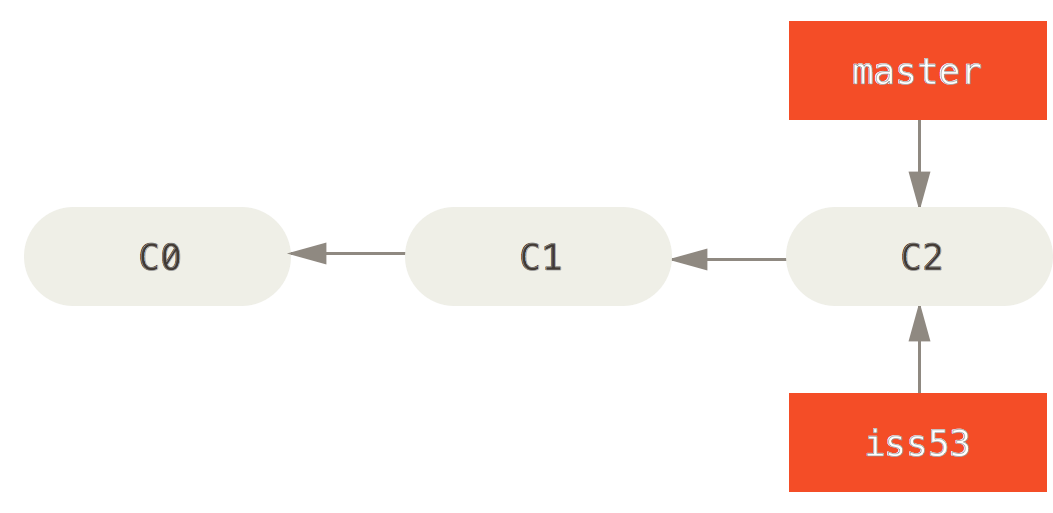
First, let’s say you’re working on your project and have a couple of commits already.



You’ve decided that you’re going to work on issue #53 in whatever issue-tracking system your company uses. To create a branch and switch to it at the same time, you can run the git checkout command with the -b switch:

**$** git checkout -b iss53

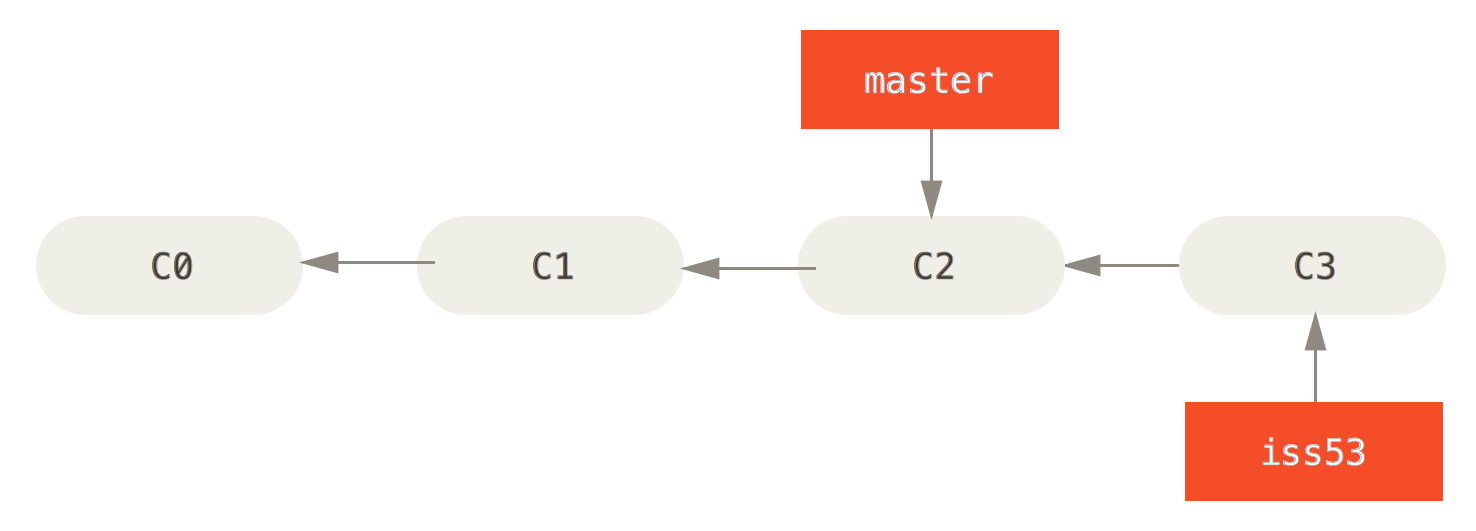
Switched to a new branch "iss53"



You work on your web site and do some commits. Doing so moves the iss53 branch forward, because you have it checked out (that is, your HEAD is pointing to it):

**$** vim index.html

**$** git commit -a -m 'added a new footer [issue 53]'



Now you get the call that there is an issue with the web site, and you need to fix it immediately. All you have to do is switch back to your master branch.

Before you do that, note that if your working directory or staging area has uncommitted changes that conflict with the branch you’re checking out, Git won’t let you switch branches. It’s best to have a clean working state when you switch branches.

**$** git checkout master

Switched to branch 'master'

 when you switch branches, Git resets your working directory to look like it did the last time you committed on that branch.

Next, you have a hotfix to make. Let’s create a hotfix branch on which to work until it’s completed:

**$** git checkout -b hotfix

Switched to a new branch 'hotfix'

**$** vim index.html

**$** git commit -a -m 'fixed the broken email address'

[hotfix 1fb7853] fixed the broken email address

1 file changed, 2 insertions(+)

1. Ok
2. Hoi chua