**Git & GitLab**

**Phạm Vũ Duy Nam**

**Skype: nampham9192**

**Phone: 01688370099**

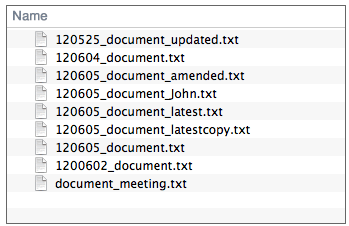
**Mail: nampvd@runsystem.net**

1. ***Giới thiệu :***

* **Vấn đề quản lý source code :**

Khi muốn đưa về trạng thái trước khi chỉnh sửa của file ta phải làm thế nào?

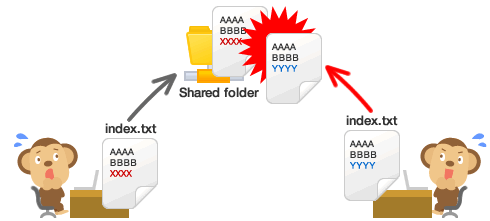
Một cách thủ công nhất là copy 1 file mới trước khi chỉnh sửa, dùng phương pháp này thì phải thường xuyên thêm ngày thay đổi vào folder hoặc file



→ Khó phân biệt đâu là file mới nhất?

→ Trong cùng 1 nhóm cần thêm tên người chỉnh sửa, nhưng lại không thể biết đã thay đổi nội dung gì

→ Nếu trong nhóm có 2 người cùng sửa 1 file, dẫn đến mất nội dung thay đổi của người sửa trước đó



→ Cần đảm bảo phiên bản source code trên máy cá nhân là mới nhất

→ Phục hồi lại 1 thời điểm nhanh chóng khi project có thay đổi hoặc sự cố …

* **Thế mạnh mô hình quản lý source phân tán :**
* CVS (Concurrent Version System) và SVN (SubVersion) với mô hình quản lý source code tập trung (CENTRALIZED), SVN thực chất là phiên bản cải tiến so với CVS, về cơ bản cả 2 đều hoạt động như nhau (tất cả source code sẽ được đặt trên 1 server trung tâm, mọi thành viên đều làm việc trên source code đó)
* Git lại có hướng tiếp cận hoàn toàn mới so với SVN, đó là hướng **DISTRIBUTED** (phân tán)
* Với 1 hệ thống như Git, SVN, CVS :

+ Repository gốc sẽ được lưu trữ trên server

+ Developer sẽ checkout 1 bản copy mới nhất về local để làm việc

+ Khi muốn apply thay đổi ở local sẽ send yêu cầu lên server

→ Nguyên tắc chung của hệ thống **Version Control System** (nhiều developer có thể làm việc với nhau trên cùng 1 project)

* Nguyên lý SVN & CVS là việc quản lý tập trung yêu cầu quyền truy cập đến server khi developer commit hoặc update từ thành viên khác & chính mỗi cá nhân cũng apply thay đổi lên server
* Git hoàn toàn đối lập :

+ Mỗi thành viên có 1 repository ở local

+ Commit & update trực tiếp từ local thành viện khác (không thông qua repository gốc trên server)

→ Tính phân tán an toàn hơn tập trung, mỗi bản copy đều là code mới nhất từ repository gốc, khi server down các thành viên vẫn làm việc offline (thao tác qua lại lẫn nhau), sau đó sẽ cập nhật lên server lại

* **Git là gì?**

Một hệ thống quản lý phiên bản mã nguồn phân tán (Distributed Version Control System – DVCS), được phát triển bởi Linus Torvalds dành cho việc phát triển Linux Kernel

* **Tại sao nên sử dụng Git?**

Git là lựa chọn hoàn hảo & vượt trội so với các phần mềm quản lý dự án khác :

* Dễ sử dụng, an toàn & nhanh chóng
* Mô hình phân chia nhánh giúp cho công việc dễ dàng được chia nhỏ và hoàn thành độc lập
* Có khả năng làm việc ngoại tuyến. Người dùng sao lưu các thay đổi, commit trên Local Repository mà không cần Internet, các thay đổi này sẽ được “push” thủ công lên Remote Repository khi có kết nối

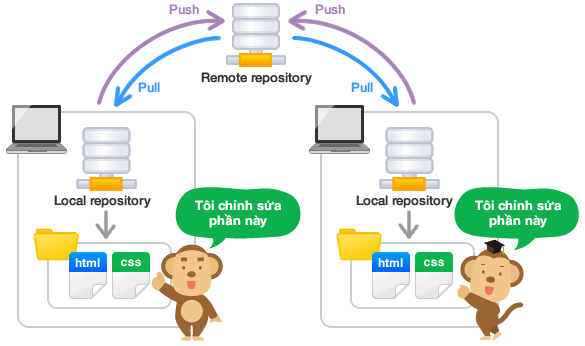
1. ***Một số khái niệm căn bản :***

* **Repository :**

(Kho chứa) nghĩa là nơi mà bạn sẽ lưu trữ trạng thái mã nguồn và một người khác có thể sao chép (clone) lại mã nguồn đó nhằm làm việc khác

Repository có hai loại là **Local Repository** (Kho chứa trên máy cá nhân) và **Remote Repository** (Kho chứa trên một máy chủ từ xa)

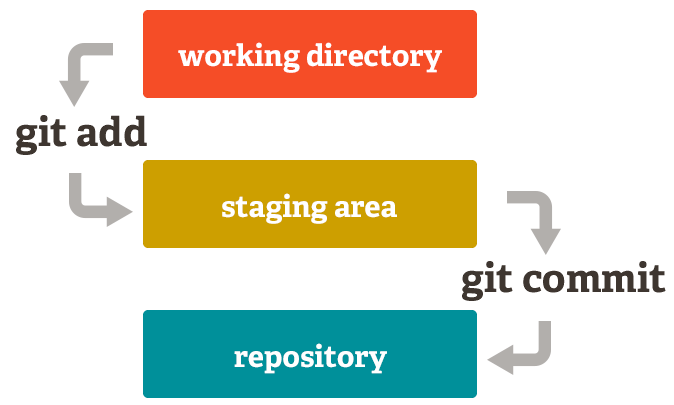


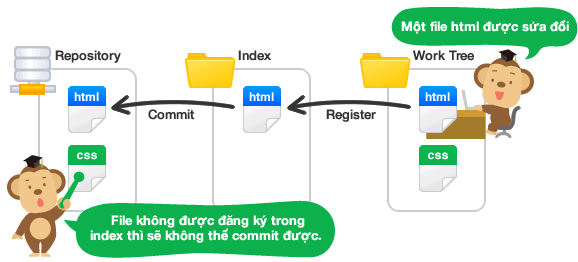


* **The working tree :**

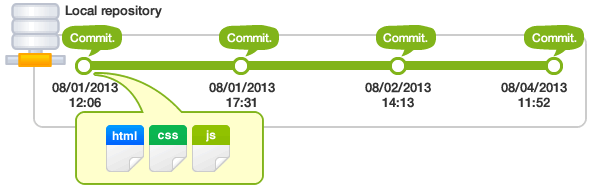
Các file được lưu trong Local Repository, nơi người dùng thao tác trực tiếp

* **The staging index :**
* Git lưu trữ nội bộ một thứ gọi là **Index** (ảnh chụp các file trong project), sau khi tạo 1 repository, nó là 1 repository rỗng và index của git rỗng
* Index không phải là danh sách các file đã bị thay đổi
* **Staging Area** là lợi thế của Git, một khu vực lưu trữ những thay đổi của người dùng trên tập tin để nó có thể được commit, trạng thái của tập tin trong Staging Area là **statged**

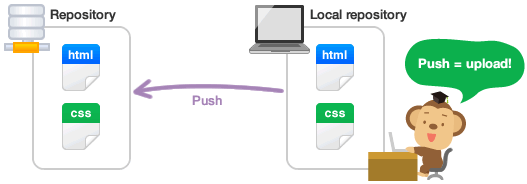




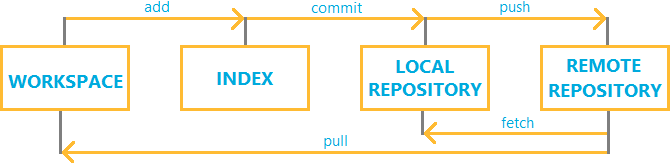
* **Commit :**
* Hoạt động dưới local, không liên quan server, lấy nội dung của Index và lưu lại ảnh chụp của project
* Khi thực hiện repository sẽ tạo ra 1 commit (hoặc revision) ghi lại sự khác biệt từ trạng thái đã commit lần trước đến hiện tại
* Mỗi commit gắn 1 nhãn xác định duy nhất (thường dùng 7 chữ số đầu từ kết quả SHA-1 hash của nội dung được chụp lại)



* **Push :**
* Sử dụng để đưa toàn bộ thay đổi được lưu trữ ở local repository lên remote repository
* Lịch sử thay đổi tại remote repository sẽ giống với local repository



* **Sự khác biệt giữa Pull, Fetch, Clone :**
* **git clone** : sao chép toàn bộ dữ liệu và thiết lập của repository (tạo 1 master branch trên local), sử dụng khi cần sao chép 1 remote repository về local
* **git pull** : tự động lấy toàn bộ dữ liệu từ remote repository và merge vào branch hiện tại
* **git fetch** : lấy tòan bộ dữ liệu từ remote repository về local repository & cho phép bạn merge thủ công vào một branch nào đó trên local



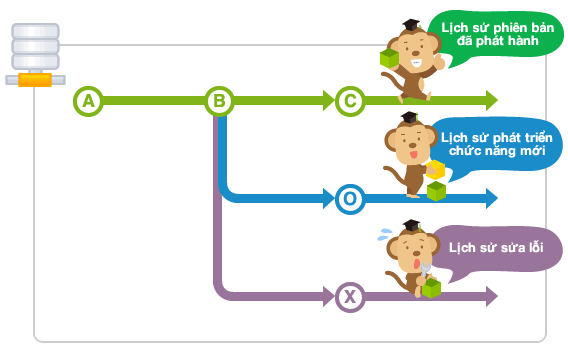
* **Branch :**

Muốn tạo 1 phiên bản thử nghiệm với mã nguồn đang làm việc tại working tree mà không gây ảnh hưởng đến code hiện tại

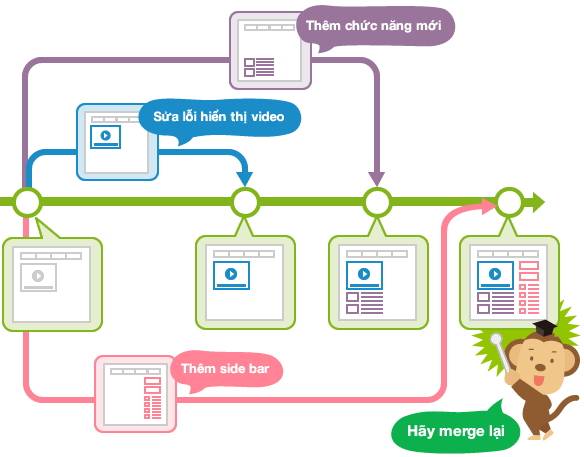
→ Clone 1 repository từ chính working tree hiện tại rồi sửa … rất mất công và gây khó khăn khi đồng bộ

→ Sử dụng phân nhánh (branching)

* Branch dùng để phân nhánh và ghi lại luồng lịch sử, các branch khác nhau sẽ không ảnh hưởng nên có thể tiến hành nhiều thay đổi đồng thời trong cùng 1 repository

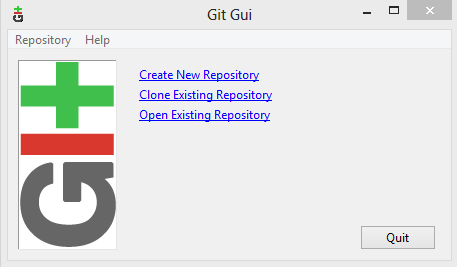
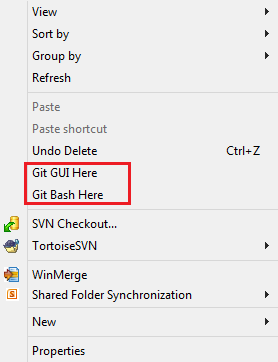


* Các branch có thể được chỉnh sửa tổng hợp lại thành 1 branch bằng cách merge



1. ***Cài đặt Git trên Windows :***

* **Download :** https://git-scm.com/download/win
* Cài đặt thành công khi Right Click sẽ thấy :



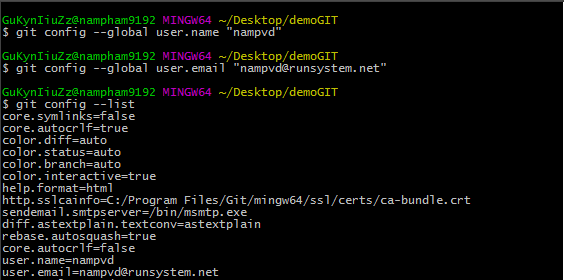


1. ***Thiết lập chứng thực cá nhân :***

**git config --global user.name "nampvd"**

**git config --global user.email "nampvd@runsystem.net"**

Sử dụng lệnh **git config --list** để kiểm tra setting

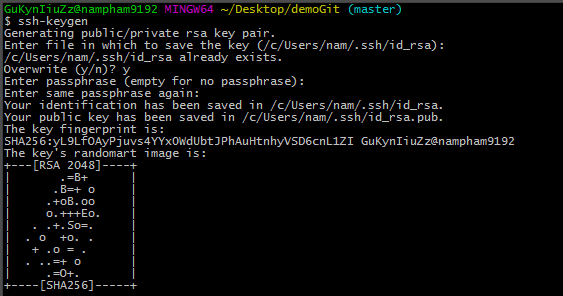


1. ***Get source code :***

* Tạo folder chứa source (Ví dụ : **demoGit**)
* Lấy code bằng giao thức **SSH** (mỗi lần Pull, Push code sẽ không yêu cầu nhập username / password)
* **Bước 1:**

Vào folder demoGit → right click → chọn **Git Bash Here** để gõ lệnh **ssh-keygen** → enter cho đến hết → truy cập file chứa key → copy toàn bộ nội dung trong file

Ví dụ: C:\Users\nam\.ssh\id\_rsa.pub



* **Bước 2:**

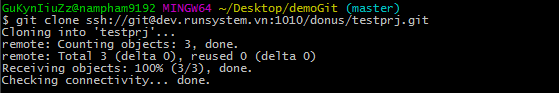
Vào server git → chọn **Profile settings** → chọn **tab ssh keys** → **Add SSH Key** → nhập title (nhập tự do) → dán key được sinh từ bước 1 → Add key



* **Bước 3:**

Vào folder demoGit → right click → chọn **Git Bash Here** để gõ lệnh

**git clone ssh://git@dev.runsystem.vn:1010/donus/testprj.git**



**Lưu ý:** Port gitlab GMO-Runsystem là **1010** nên cần truyền thêm port

* Clone dữ liệu 1 branch :

**git clone -b <branch name> --single-branch git://github/repository.git**

* Clone dữ liệu 1 folder :

**git clone git://github/repository.git <folder name>**

1. ***Thao tác với nhánh (branch) :***

* **Tạo branch :**
* **Cách 1 :** Vừa tạo vừa checkout code từ nơi tạo nhánh

**git checkout -b <tên nhánh mới> <nhánh hiện tại>**

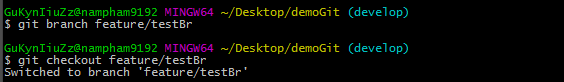
Ví dụ:



→ Tạo nhánh develop từ nhánh master, đồng thời lấy code từ master về develop

Đơn giản hơn có thể viết như sau: **git checkout –b develop** (hiện tại đang ở nhánh master)

* **Cách 2 :** Tạo branch trước, vào checkout code sau



* **Quy tắc đặt tên nhánh :** **feature/xxx**
* **Đưa file lên từ nhánh bất kỳ :**
* **Add file :**

**git add <tên file>** : add 1 file

**git add -A** : add tất cả file

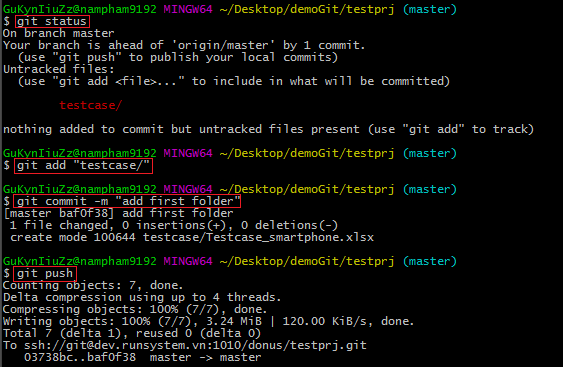
**git add <tên folder>/\*** : add folder và tất cả file bên trong

* **Commit file :** **git commit -a -m '<nội dung comment>'**

+ Lệnh trên sẽ add tất cả các thay đổi, ngoại trừ các file mới tạo trước khi commit

+ Nội dung comment cần ghi rõ thay đổi những gì, lý do tại sao cần thay đổi

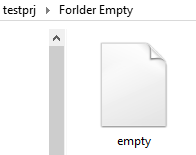
* **Push file :** **git push origin <tên branch đang thao tác>**



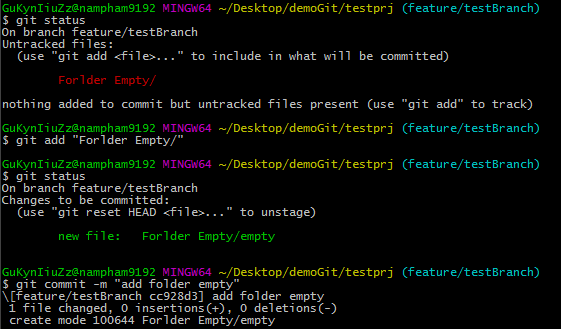
*Đối với nhánh master & develop, khi dùng lệnh push không cần chỉ rõ remote (origin) hay tên nhánh*

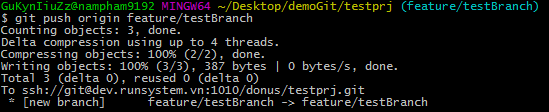
* **Thủ thuật đưa 1 folder trống lên GitLab :**

Trong folder tạo 1 file tạm (chẳng hạn file “empty”) như sau:

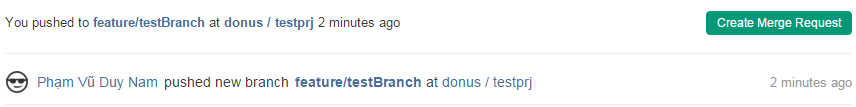


Sau đó add & commit file này tương tự như những file bình thường, sau khi folder có dữ liệu sẽ xóa đi

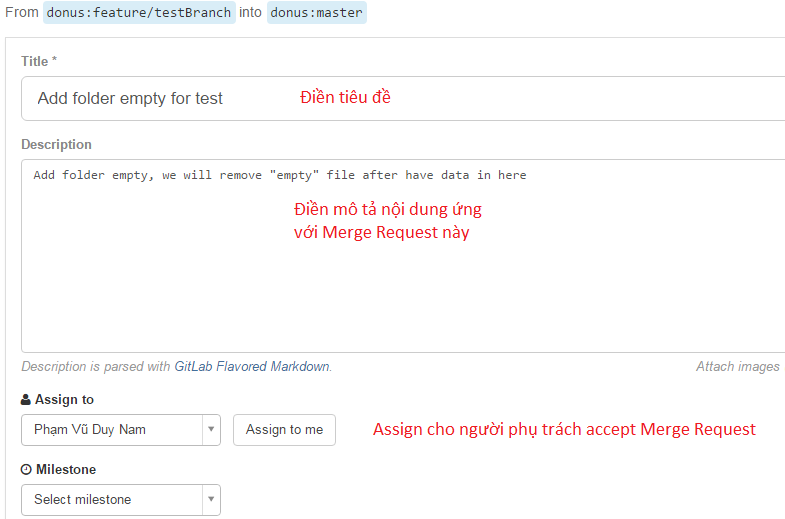




* **Gửi request đến gốc cần merge code (Create Merge Request) :**
* Vào server git ở trang index sẽ có xuất hiện button yêu cầu gửi request



* Điền thông tin như sau → **click** **Submit merge request**



* **Từ branch lấy source mới nhất của gốc :**

Vào folder demoGit → right click → chọn **Git Bash Here** để gõ lệnh

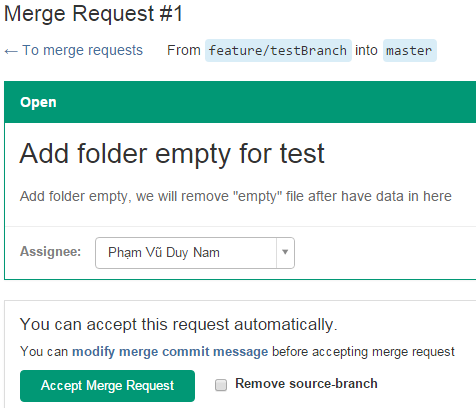
→ **git checkout <tên nhánh>** → **git merge <tên respository>/<tên gốc>**

1. ***Thao tác với gốc (master, develop hoặc gốc sinh ra branch) :***

* **Merge code bằng giao diện :**
* Sau khi tạo Merge Request, tại server git xuất hiện thông báo



* **Click Accept Merge Request** (apply thay đổi của nhánh con vào gốc)



* Sau khi merge sẽ xuất hiện thông báo



* **Merge code bằng command :**
* **Bước 1:**

Vào folder demoGit → click chuột phải → chọn **Git Bash Here** để gõ lệnh

* **Bước 2:**

Vào branch muốn merge, Pull code mới nhất về → **git checkout feature/testBranch** → **git pull**

* **Bước 3:**

Trở về nhánh cần merge (ở đây là develop)

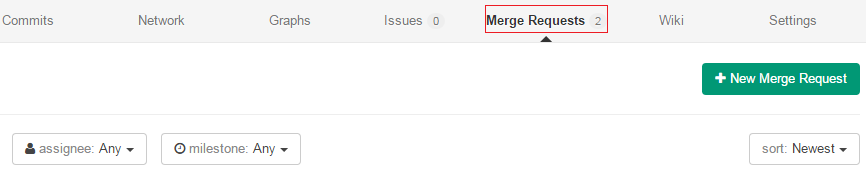
**git checkout master** → trở về develop

**git pull** → lấy source mới nhất

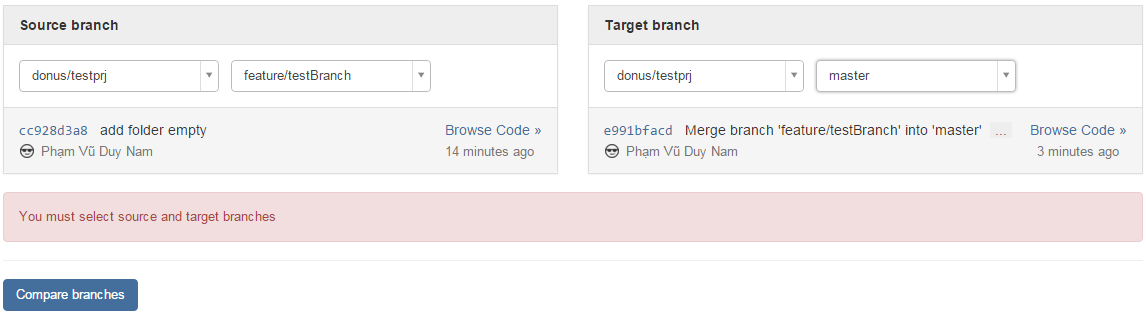
**git merge feature/testBranch** → merge source

1. ***Tạo Merge Request :***

* **Bước 1:** Vào server git → chọn **tab Merge Requests** → click **New Merge Request**

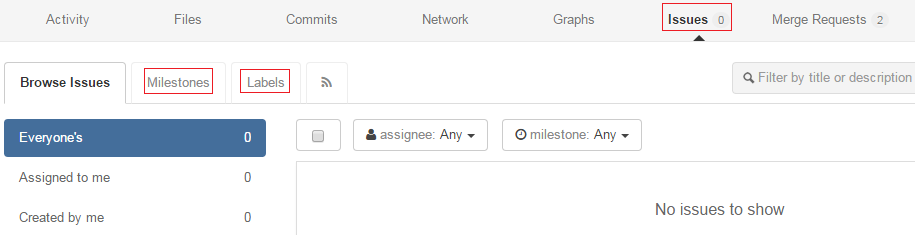


* **Bước 2:** Chọn 2 branch cần tạo Merge Request → **click Compare Branches**



* **Bước 3:** Check các thay đổi giữa 2 branch → Điền thông tin → **click** **Submit merge request**
* **Cách tạo milestones, labels trong màn hình tạo mới Merge Request :**

Vào server git → chọn project → chọn **tab Issues** → click **New Merge Request**



1. ***Một số lệnh thường dùng :***

* **Tạo respository mới :** **git init**

Thay vì vào folder demoGit → right click→ chọn **Git Innit Here**, ta có thể sử dụng lệnh trên.

* **Xóa nhánh :**

**git branch –D <tên nhánh>** → xóa nhánh tại local

**git push origin :<tên nhánh>** → đưa thay đổi lên server

* **Đổi tên nhánh :**

# Đổi tên nhánh tại local

**git branch -m old\_branch new\_branch**

# Xóa nhánh cũ trên server

**git push origin :old\_branch**

# Áp dụng thay đổi lên server

**git push --set-upstream origin new\_branch**

* **Lấy code mới nhất :**

**git pull**  → cập nhật local respository và commit mới nhất (master, develop)

**git pull origin <tên nhánh>** → sử dụng cho branch checkout từ gốc

* **Xem những gì sẽ được commit :**

**git status**

#change to be committed → Những gì thay đổi đã được add vào Index, nằm trong lần commit kế tiếp

#changed but not updated → Những thay đổi được thực hiện tại Working Tree, chưa add vào Index

* **Xóa file (khi đã add vào index)**
* Xóa file trong Local Repository nhưng không xóa tại Local (file thực tế)

**git rm --cached <tên file>**

**git commit –m “delete <tên file>”**

**git push <tên repo> <tên nhánh>**

* Xóa file cả trong Local & Local Repository

**git rm <tên file>**

**git commit –m “delete <tên file>”**

**git push <tên repo> <tên nhánh>**

* **Theo dõi log :**

**git log** → Xem lịch sử commit

**git log --pretty=oneline –abbrev-commit** → Xem lịch sử commit dạng 1 dòng ngắn gọn

**git log --author=<name>** → Xem những thay đổi của 1 thành viên trong project

**git log --name-status** → Xem lịch sử những file đã thay đổi

* **Config user settings :**
* Config từng project

**git config user.name “<user name>”**

**git config user.email “<email>”**

* Config một lần cho tất cả các git project

**git config --global** **user.name “<user name>”**

**git config --global user.email “<email>”**

* **Hủy bỏ các thay đổi hiện tại :**
* Nếu chưa add vào Index, có thể khôi phục tại Working Tree : git checkout <tên file>
* Nếu đã add vào Index, có thể khôi phục về lần commit gần nhất :

**git reset HEAD <tên file>**

**git reset HEAD** → khôi phục toàn bộ Index

**git reset --hard origin/develop** → Xóa bỏ những thay đổi và undo lại phiên bản mới nhất từ server, trỏ về develop

Ví dụ :

Thực hiện 3 lần commit : commit1 – commit2 – commit3

**git reset --hard HEAD** → khôi phục về commit 3

**git reset --hard HEAD^** → khôi phục về commit 2

**git reset --hard HEAD^^** → khôi phục về commit 1

(HEAD^ có ý nghĩa giống với HEAD~ hay @^, quay về trước 1 commit)

**git reset --soft HEAD** → bỏ đi commit nhưng vẫn giữ nguyên những thay đổi chưa được commit trong Working Tree, hữu ích khi muốn giữ lại những thay đổi chưa commit cho lần commit tiếp theo

* **Hủy commit cũ :**
* Nếu đã commit những thay đổi, có thể undo lại commit đó : **git revert <commit>**

Ví dụ :

git revert 068b9b9 → tạo ra commit thứ 2, undo thay đổi của commit chỉ định

* Có thể dùng để undo một commit cũ hơn, tuy nhiên có thể dẫn đến conflict nội dung code nếu các commit mới hơn nó cũng sửa trùng nội dung đó
* **Diff giữa các commit :**
* So sánh một commit với tiền thân của nó **git show <commit>**
* So sánh hai commit cụ thể với nhau **git diff <commit1>..<commit2>**
* Xem diff của toàn bộ lịch sử **git log -p**
* **Hiển thị branch :**
* Các branch local : **git branch**
* Các branch local + server : **git branch –a**
* **Fetch :**

**git fetch <tên nhánh>** → lấy code thay đổi mới nhất từ 1 nhánh, lưu vào database của Git local, không được gộp vào repository

**git fetch --all**  → lấy code thay đổi từ tất cả các nhánh server

* Muốn gộp code vào branch hiện tại, sau khi fetch phải dùng lệnh **merge**
* Muốn lấy code trực tiếp mà không cần merge, ta sử dụng : **git pull**

(khuyến khích nên merge code thủ công cẩn thận để tránh bị mất code hiện tại)

* **Merge (gộp code) :**

**git merge <tên nhánh>** → Gộp code với nhánh local được chỉ định

**git merge origin/<tên nhánh>** → Gộp code với nhánh server được chỉ định

(**origin** là một original remote repository)

* **Sửa commit message :**

Ghi đè commit mới nhất bằng option : **git commit –amend**

Add thêm file mà không muốn thay đổi commit message :

**git add <tên file>** → **git commit –amend –no-edit**

* **Lưu tạm những thay đổi chưa commit :**

**Stash** là chức năng dùng bộ nhớ tạm lưu lại các thay đổi đang làm mà chưa commit được, sau đó phục hồi code về nguyên trạng ban đầu trước khi có những thay đổi đó.

**git stash**

**git stash list** → Xem danh sách các stash đã lưu

**git stash apply** → Phục hồi stash mới nhất vừa lưu, không xóa nó

**git stash pop** → Phục hồi stash mới nhất vừa lưu, xóa nó khỏi danh sách

**git stash apply <tên stash>** → Phục hồi stash bất kỳ (VD: git stash apply stash@{1})

**git stash drop <tên stash>** → Hủy bỏ một stash

**git stash show –p <tên stash>** → Xem chi tiết một stash

* **Gộp commit với nhánh :**

**git checkout <tên nhánh>** → chuyển sang nhánh cần merge

**git cherry-pick <commit>** → Sao chép commit và tạo 1 commit mới trên nhánh hiện tại

Ví dụ :

git cherry-pick 5b97361

* **Gắn tag cho commit :**
* Lệnh git **tag <name> <commit>** gắn một tag cho một commit. Nếu bỏ qua phần <commit>, tag sẽ được gắn cho commit gần nhất:

**git tag working 3720b35**

**git tag broken**

* Tag chỉ là một cái nhãn mà có thể dùng để gọi tên commit được gắn nhãn
* Có thể dùng tag ở bất cứ ngữ cảnh nào mà ta có thể dùng nhãn hash, chẳng hạn trong lệnh git diff:

**git diff working..broken**

* Tag là công cụ hoàn hảo để đặt tên cho các mốc cụ thể trong lịch sử project. Người ta thường dùng tag dạng version number khi commit các bản release của project – bằng cách này ta có thể dễ dàng tìm thấy các commit quan trọng: **git tag v1.0.3**

1. ***Xử lý Conflict :***

* Khái niệm : Conflict là trường hợp có 2 sự thay đổi trên cùng 1 dòng code, Git không thể xác định đâu là đúng ý coder
* Dấu hiệu nhận biết :

Khi Pull, Merge code sẽ xuất hiện thông báo dạng **CONFLICT (content): Merge conflict in …**

Nội dung file khi bị conflict sẽ xuất hiện thêm vài dòng để phân biệt các thay đổi, dạng như:

*< <<<<<< HEAD*

*your changes*

*=======*

*friend changes*

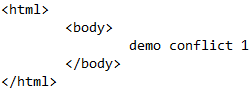
*>>>>>>> master*

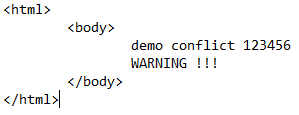
→ Những dòng code nằm trong HEAD … ==== là những thay đổi của ta, dựa vào đây ta sẽ thay đổi, bổ sung, merge code sau đó lưu lại.

→ **git commit –am “<nội dung>”**

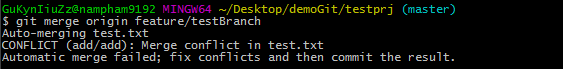
→ **git push origin <tên nhánh>**

* Ví dụ :
* 2 branch: master, feature/testBranch
* 1 file test.txt nằm chung path ở cả 2 branch, tuy nhiên nội dung dòng thứ nhất khác nhau

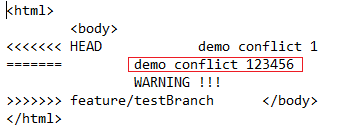
 → Nội dung tại nhánh master

 → Nội dung tại nhánh feature/testBranch

* Khi merge code sẽ bị conflict như sau :



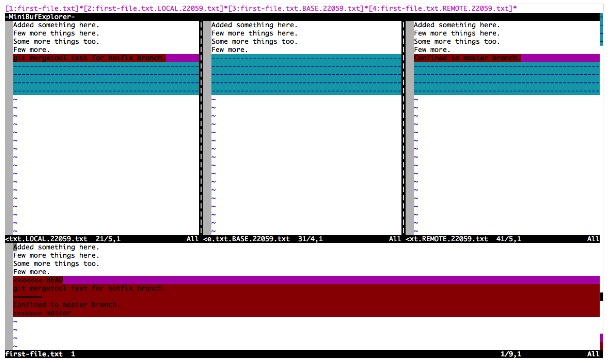
* **Xử lý :**



Nội dung file test.txt như trên, tự tay giải quyết conflict + xóa bỏ các marker đánh dấu



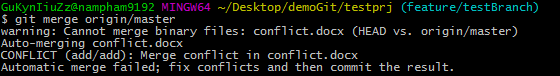
\*\*\* Có thể sử dụng **git mergetool** để view và chỉnh sửa (cần config tool sẽ sử dụng khi resolve conflict bằng lệnh : **git config --global merge.tool vimdiff**)



1. ***Xử lý Conflict & merge 2 file office***

Tại nhánh feature/testBranch tạo file “conflict.docx” với nội dung: **Test conflict with file DOC**

Tại nhánh master tạo file “conflict.docx” với nội dung: **Conflict !!!**

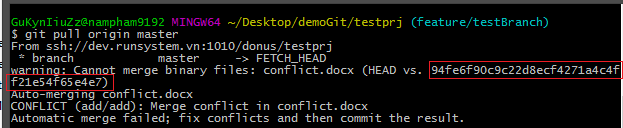


Sử dụng lệnh git reset HEAD để quay về commit gần nhất trước đó tại nhánh feature/testBranch:



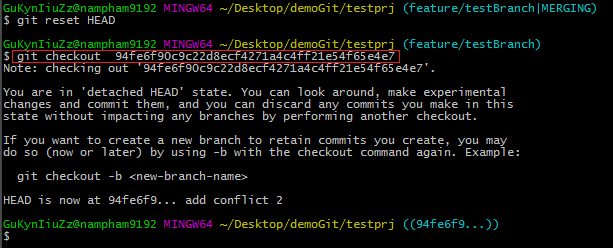
Copy file “conflict.docx” cũ ra folder khác

Chạy lệnh git pull origin feature/master:



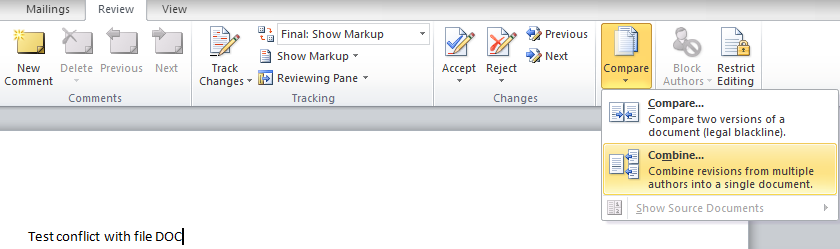
→ Lệnh này sẽ cho biết hiện tại đang bị conflict với commit nào

Nhảy đến commit đó để lấy file “conflict.docx”:

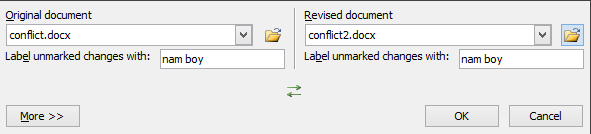


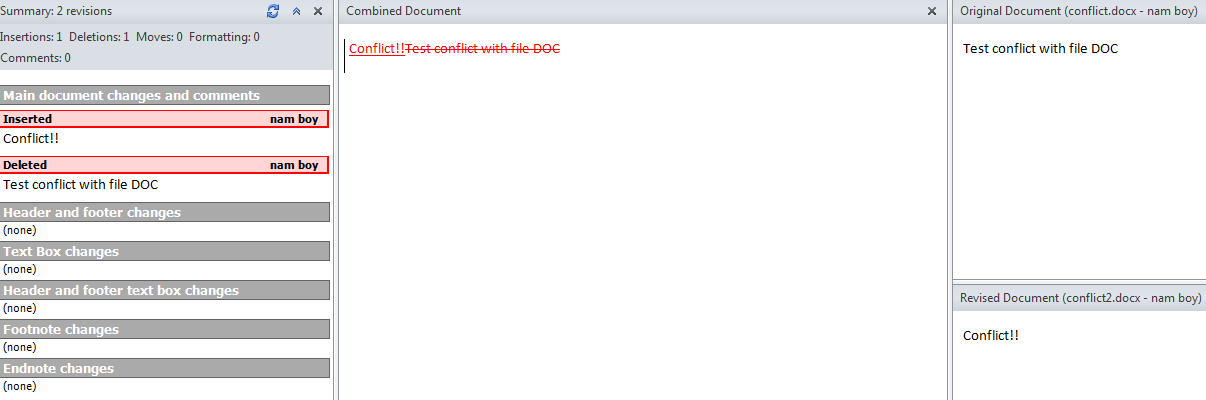
Copy file “conflict.docx” hiện tại cũ ra folder khác (đặt tên khác để k bị trùng với file trên)

Sử dụng bất kỳ công cụ nào để DIFF nội dung giữa 2 file office, hoặc dùng tính năng có sẵn của Word:



Chọn 2 file để compare:



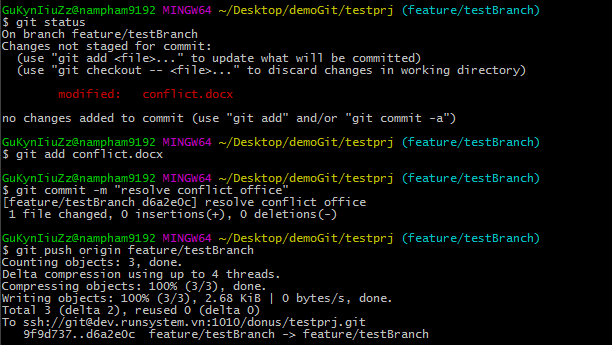


Lưu thành 1 file mới sau khi đã xử lý merge nội dung 2 file (cẩn thận không để mất dữ liệu)

Quay lại nhánh cũ bằng lệnh git checkout <tên nhánh>



Tiến hành đưa file đã xử lý lên server



Bên nhánh master khi pull code cũng tương tự, sẽ lấy file đã xử lý conflict đè lên file hiện tại & commit lên

1. ***Tài liệu tham khảo :***

<http://wiki.nukeviet.vn/programming:vcs:git>

<http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/vi/>

<https://git-scm.com/doc>

<http://backlogtool.com/git-guide/vn/>