Git Guideline

Dùng cho dự án Citus-Digital-Workplace

1. Tóm tắt nội dung tài liệu

Nội dung chính là hướng dẫn developer dùng Git Bash để làm việc với https://git.fsoft.com.vn/fsoft/Citus-Digital-Workplace. Tuy nhiên một số task có thể làm trên giao diện web nên hãy xem mục 2.2 trước, có thể bạn không cần đến Git Bash cho công việc của mình.

Các bạn đã quen thuộc với Git vẫn nên đọc qua tài liệu này để biết một số qui ước tổ chức branch không giống thông lệ của Git.

Các bạn chỉ làm việc trên các nhánh có sẵn có thể bỏ qua mục 5 - Tạo project mới.

Nếu bạn chỉ muốn bắt đầu thật nhanh thì hãy đọc mục 2.3, mục 3, mục 6.

Tài liệu bạn đang đọc còn đang thiếu hướng dẫn Merge và Pick commit giữa các nhánh. Bạn có thể kiểm tra bản mới nhất tại URL trên.

2. Giới thiệu về Git và git.fsoft.com.vn

2.1 Git và git.fsoft.com.vn

Git là phần mềm open source được tạo ra bởi Linus Torvalds.

GitHub, tức https://github.com và GitLab tức https://gitlab.com là hai dịch vụ well-known (nhưng không phải duy nhất) được xây dựng dựa trên Git. GitLab cho phép free user được có private repo.

Fsoft Git, tức https://git.fsoft.com.vn là phiên bản tương tự GitLab. Một số đặc điểm:

- Dùng account Fsoft để truy cập.

- Cho đến 12/2017 chỉ truy cập được từ mạng Fsoft.

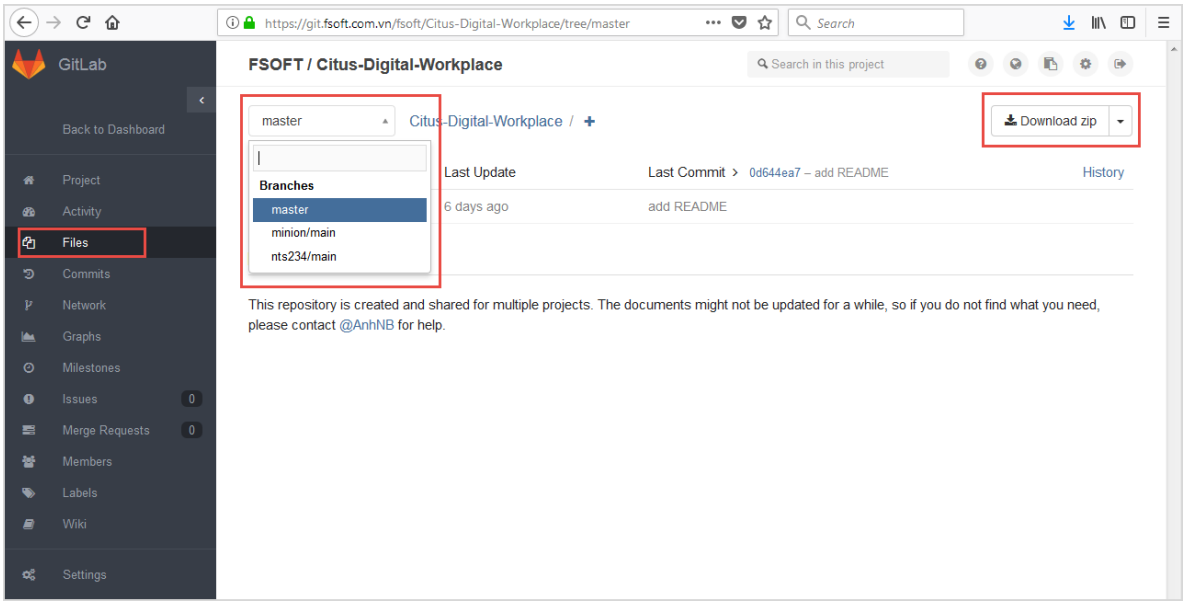
- So với GitLab thì có thể có một số feature không được enable.

2.2 Một số task có thể thực hiện thông qua giao diện web

Bạn có thể làm một số việc trên web mà không cần cài đặt Git client. Các gợi ý sau đây có ích khi bạn chỉ cần lấy package về dùng, hoặc bạn đang truy cập từ một máy không có Git client.

Download: bạn có thể download một branch.

Sau khi truy cập trang https://git.fsoft.com.vn/fsoft/Citus-Digital-Workplace , chọn *Files*, chọn branch và bấm *Download zip.*



Dự án có thể dùng một số branch làm nơi chứa binary để bạn có thể download về. Thông tin sẽ được cập nhật trên các file README hoặc wiki.

Review: Bạn có thể review file hoặc commit. Có thể add comment (comment trên web - không phải là nội dung file), có thể send link trỏ đến một dòng code cụ thể.

Protected branches: Bạn có thể set/unset một số branch hạn chế commit.

Add member.

2.3 Một số hiểu biết giúp bạn dùng Git đúng

Bạn có thể dùng Git đúng ngay từ đầu mà không cần có kinh nghiệm với Git. Việc dùng Git đúng giúp cho repository được quản lý hiệu quả, history sạch sẽ, source code dễ review, developer được tập trung vào code.

2.3.1 Thoải mái tạo branch của riêng bạn

Điểm khác biệt lớn nhất của Git so với svn là branch. Branch trong Git là các nhánh code được tạo ra để phục vụ bất kỳ mục đích nào: để thử nghiệm một ý tưởng của bạn, để test, để bắt đầu code một feature mới, để baseline, release hoặc deliver.

Đừng tạo ra một folder để tách biệt code của bạn như vẫn làm trong svn. Thay vì vậy hãy tạo một branch. Nếu như trong svn bạn phải làm bằng tay việc giữ cho nhánh (folder) của bạn có được code mới của các folder khác, cũng như phải làm bằng tay nhiều lần để code trên folder của bạn

được áp dụng vào các folder khác, thì branches của Git được quản lý để track nhau qua lịch sử commit. Bạn có thể quan sát các branch rẽ nhánh như thế nào, có thể pick commit từ nhánh này vào nhánh khác, có thể merge hoặc rebase các nhánh. Việc duy nhất bạn phải làm bằng tay là merge code khi có conflict.

Bạn có thể ngay lập tức thử nghiệm tạo một branch của riêng mình, commit thử và rồi xóa branch tùy thích. Việc thử nghiệm của bạn sẽ không gây ra ảnh hưởng gì đến các branch khác.

2.3.2 Quản lý working folder trên máy bạn

Việc switch giữa các branch trong cùng một git folder được thực hiện bằng lệnh **git checkout**. Khi bạn đang làm việc trên branch A và muốn switch sang branch B, thông thường bạn phải commit tất cả các thay đổi của branch A xong xuôi rồi mới switch được. Mặc dù bạn có thể dùng lệnh **git stash** để lưu giữ các thay đổi dở dang mà không commit, nhưng đây là task phức tạp và dễ nhầm lẫn.

Khi bạn switch từ branch A sang một branch khác, về bản chất là file của branch A biến mất trên file system và đương nhiên sẽ có vấn đề với các IDE đang làm việc với các file đó. Mặc dù các IDE như Eclipse hay Visual Studio có các add on để support Git branching, nhưng vẫn là không an toàn khi switch giữa các branch đã import vào IDE.

Vì vậy, với người dùng Git làm việc trên các IDE phức tạp, bạn nên clone nhiều git repo để làm việc trên các branch khác nhau. Đây là một ví dụ:

Tôi muốn làm việc trên 2 project tách biệt là Citus Detector và CitusS. Ba branch tương ứng là CitusDetector/main, CitusDetector/stable/1.0 và CitusS/main. Tôi sẽ tạo 3 working folder và đặt tên chúng tương ứng với tên branch:

/d/CDW/git/CitusDetector/main ← tại đây tôi clone project và checkout CitusDetector/main

/d/CDW/git/CitusDetector/stable/1.0 ← tại đây tôi clone lần nữa và checkout CitusDetector/stable/1.0

/d/CDW/git/CitusS/main ← tại đây tôi clone lần nữa và checkout CitusS/main

Liệu có vấn đề gì khi bạn import 2 branch của cùng một project (CitusDetector) vào IDE? Bạn hãy kiểm tra nhé nhưng có vẻ Visual Studio không có vấn đề gì với hai solution giống hệt nhau nhưng có location khác nhau. Còn Eclipse thì bạn sẽ gặp vấn đề với 2 project có file pom.xml giống hệt nhau chẳng hạn, khi đó bạn phải giải quyết bằng cách dùng 2 workspace (khái niệm của Eclipse) khác nhau.

2.3.3 Hiểu file Gitignore

File Gitignore chứa các name pattern để Git dựa trên đó "ignore" các file không nên được versioning, nói cách khác là các file chỉ nên tồn tại trong working folder của một developer mà không nên được push lên repo chung.

Ví dụ đơn giản là các file .exe do Visual Studio tạo ra khi build.

Nếu bắt đầu một branch mới, empty và không có sẵn file gitignore, việc đầu tiên bạn nên làm là tạo file này. File gitignore cho từng loại dự án và IDE có thể tìm thấy dễ dàng trên internet. Chỉ cần download về, đặt tên là .gitignore và đặt tại root folder của repo.

Cơ chế ignore của Git linh động, visual và tự động nhiều hơn so với svn. Bạn có thể dùng wildcard, regular expression và reverse (!) để viết file này.

2.3.4 Giữ cho branch của bạn fast-forward

Phần lớn thời gian, branch local của bạn được keep track với một remote branch (hay gọi là upstream branch). So sánh lịch sử commit của 2 branch, bạn có thể có những commit mà remote không có (ahead), có thể không có những commit mà remote có (behind), và có thể vừa ahead vừa behind (divert).

Khi divert xảy ra, có thể có conflict hoặc không. Nếu không có conflict, bạn có thể pull / push dễ dàng mà không phải merge code bằng tay. Trạng thái đó là fast-forward.

Để tiết kiệm thời gian merge, bạn nên cố gắng giữ được trạng thái fast-forward bằng cách update code thường xuyên. Việc này cũng giống như trong svn, nhưng thủ tục tất nhiên có khác:

git fetch origin

git status

git pull

Đến đây bạn sẽ được báo trạng thái và xem xét có phải merge hay chưa. Làm việc này thường xuyên bạn có thể cũng phải merge nhiều hơn nhưng mỗi lần chỉ merge ít và an toàn.

2.3.5 Dùng câu lệnh git để xóa hoặc rename (move) file

Với những file bạn mới tạo ra và chưa commit thì sao cũng được. Với những file đã được commit, bạn không nên dùng tiện ích nào ngoài git (như câu lệnh của OS) để xóa hoặc move.

Ví dụ bạn dùng Windows Explorer để đổi vị trí của file, git sẽ hiểu là một file mất đi và một file mới tạo ra, chưa có history.

Thay vì vậy, bạn dùng câu lệnh git để move file sẽ giữ được history của file.

Trong trường hợp bạn dùng IDE (VS, Eclipse, …) để refactor cấu trúc code, về cơ bản bạn chấp nhận mất history. Vì lí do này, hãy cố gắng design namespaces tốt ngay từ đầu để những thay đổi sau đó là nhỏ và có thể làm bằng câu lệnh Git.

2.3.6 Tại sao history của file lại quan trọng

SVN cũng có thể show cho bạn history của một file từ lúc tạo ra đến thời điểm hiện tại, tuy nhiên với Git việc này dễ dàng hơn nhiều dẫn đến người dùng Git sử dụng history nhiều hơn trong việc review code và phân tích bug.

2.3.7 Tại sao bạn cần viết message rõ ràng, dễ hiểu và bằng English khi commit

Message của commit là phần duy nhất dễ hiểu và ngắn gọn trong một commit và một lịch sử commit. Do đó bạn đừng ngại mất thời gian suy nghĩ viết một message cho rõ ràng, dễ hiểu. Cũng đừng ngại viết message dài.

Một commit để fix bug thì không nên viết là “Fix bug ABC-123”. Bug ABC-123 chỉ có ý nghĩa với bạn, người khác thì sẽ phải truy cập Jira để biết bug đó là bug gì. Bạn nên viết “Fix bug ABC-123: misuse absolute link” chẳng hạn.

Message của commit là hướng tới technical review. Bạn nên thể hiện nội dung technical của commit và đừng ngại gây khó hiểu cho những người không cần làm công việc technical review.

Tại sao lại nên dùng English và nên tuân thủ văn phạm trong message? Rất nhiều tool đọc commit message của bạn để generate ra document như test report, release note. Viết bằng English giúp đảm bảo hiển thị đúng trong các format và tool khác nhau.

Tại sao nhiều công ty yêu cầu viết Jira ID ở đầu message? Đó là do các công ty này tích hợp các tool như Fisheye vào source management process của họ. Khi bạn commit hoặc commit của bạn được merge lên một nhánh qui ước, một message tự động được ghi vào phần comment của issue trên Jira.

2.3.8 Code formatter và code convention

Các công ty lớn như Google thường qui định code convention với những chi tiết nhỏ như dùng tab hay space khi save tab, tab dài 2 hay 4 space, … Việc này giúp cho các commit chỉ chứa nội dung thay đổi thực sự mà không có những thay đổi chỉ về mặt format.

Hiện nay chúng ta không có convention tuy nhiên bạn có thể go ahead và đề xuất convention, nhưng hãy cố gắng để convention của bạn được apply một cách tự động, ví dụ như Eclipse có tính năng code formatter dựa trên file có thể import/export nên có thể dùng chung cho các member trong dự án.

Còn khi không có convention, hãy thận trọng khi dùng các tính năng auto format hàng loạt của IDE để tránh tạo ra các commit lặp đi lặp lại và reverse nhau.

2.3.9 Document trên Git

Document là một phần công việc của developer, nhưng phần lớn chúng ta lại cảm thấy đây là phần công việc buồn chán và không hiệu quả.

Để tránh việc được giao những task tốn nhiều thời gian để fix những comment "lặt vặt" về format, bạn nên chủ động document theo cách của một developer.

Tại sao lại như vậy? Bạn được giao viết một file Word hướng dẫn cài đặt tool thường là do ngoài bạn không có ai làm được việc này, nói cách khác bạn đã không chia sẻ thông tin thông suốt tới các bộ phận khác. Nếu bạn chủ động document trong quá trình development, bạn được làm theo ý mình còn người khác có thể dùng thông tin của bạn để viết theo format mà họ muốn.

Một vài lựa chọn có sẵn trên Git: file README.md, một file .txt sẽ phù hợp với những guideline chỉ cần plain text. Những hướng dẫn cần hình ảnh thì có thể dùng Wiki.

Nếu bạn cần một dạng mark up đơn giản để viết code snippet, câu lệnh hoặc configuration, bạn có thể dùng Markdown (.md).

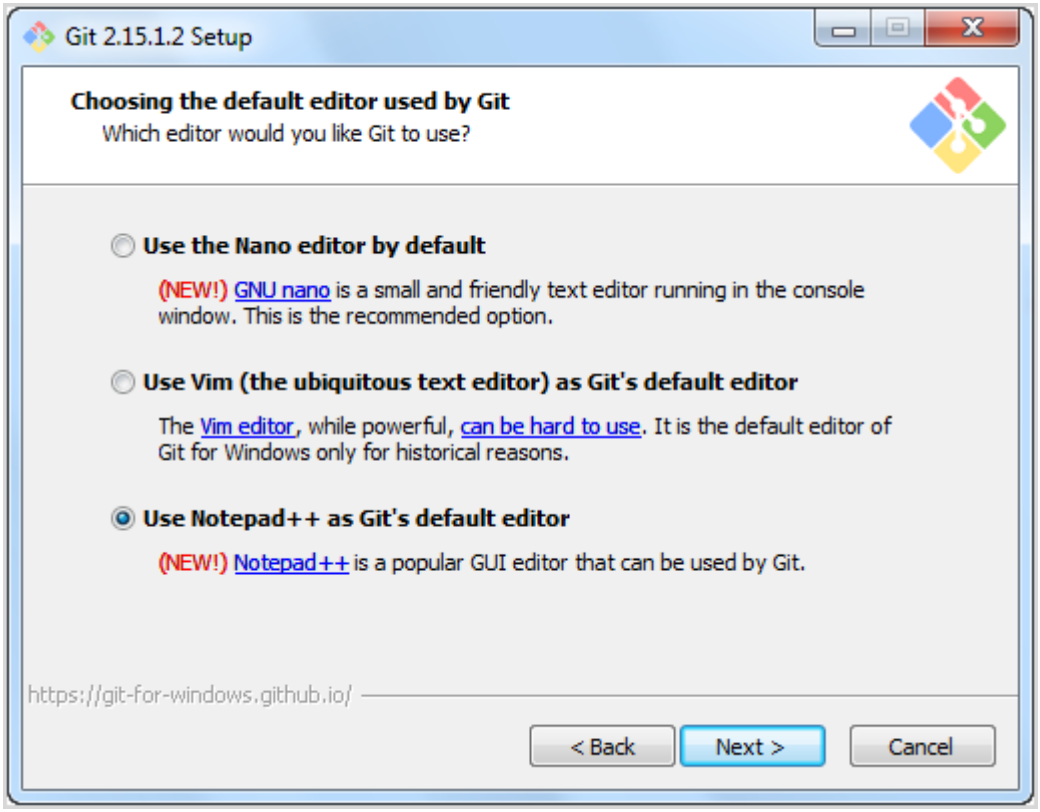
Việc viết User guide ở Fsoft vẫn phải dùng Word do chúng ta không phổ biến các tool viết doc chuyên nghiệp (AsciiDoc, Docbook XSL, ...), tuy nhiên Installation / Configuration guide thì rất nên được giải quyết bằng những công cụ trên.

3. Cài Git Bash

3.1 Cài đặt

Download từ: https://git-scm.com/downloads

Cách cài: Next Next. Chú ý màn hình sau đây. Vim và Nano khó dùng trừ phi bạn đã quen với Linux, vì vậy nếu trên máy có Notepad++ thì hãy chọn nó. Các màn hình khác nên để default.



Sau khi cài xong, bạn có thể thấy "Git GUI" từ menu Start. Tuy nhiên nên dùng Git Bash (command line) và guideline này cũng chỉ hướng dẫn dùng Git Bash. Có vài lí do nhưng để ngắn gọn xin không trình bày.

Từ menu Start, gõ "git bash" và … chạy nó thôi.

3.2 Configure username và email

Lưu ý: bạn nên tự gõ các câu lệnh để tránh một số lỗi do paste từ Word.

git config --global user.name ChauTN

git config --global user.email 'chautn@fsoft.com.vn'

Không cần phải đắn đo vì các thông tin này không được dùng để authenticate, mà được dùng để display author trên các commit. Có thể sửa bất cứ lúc nào. Chỉ cần để người khác nhận ra đó là bạn.

Để kiểm tra gõ câu lệnh sau:

git config -l

3.3 Authentication

Bạn có thể dùng SSH hoặc HTTPS để connect với git.fsoft.com.vn.

SSH giúp bạn tránh được một số lỗi có thể gặp do restrict của Fsoft network. HTTPS thì nhanh khỏi mất công setup.

Bạn có thể dùng cả hai và tùy ý (tức không phải bạn chọn cái gì là cứ phải dùng cái đó mãi) nên bây giờ, để cho nhanh thì dùng HTTPS nhé. Nhưng sau này có thời gian thì nên dùng SSH.

Trường hợp bạn set up git trên một máy robot, máy này dùng account của bạn để authenticate, nhưng yêu cầu là mỗi khi bạn thay password Fsoft thì máy này không bị ảnh hưởng, thì SSH là lựa chọn bắt buộc.

Nếu dùng HTTPS:

git clone https://git.fsoft.com.vn/fsoft/Citus-Digital-Workplace.git

Nếu dùng SSH:

git clone git@git.fsoft.com.vn:fsoft/Citus-Digital-Workplace.git

4. Tạo branch dev

Trong phần này, bạn tạo một branch trên local và push nó lên server. Hãy chú ý tới start point của branch và upstream branch.

Giả sử bạn muốn phát triển tool NTS234, bạn có thể bắt đầu từ nhánh NTS234/main hoặc các nhánh có tên “stable”. Nhánh main được maintain thường xuyên để có các feature và bug fix mới nhất từ tất cả các nhánh khác. Nhánh stable được đảm bảo well-tested.

Các nhánh trên được protected và do git master quản lý. Các developer sẽ không làm việc (push) trực tiếp trên nhánh này, thay vì vậy họ tạo nhánh dev.

4.1 Clone remote repository

Các câu lệnh sau đây tạo một folder dưới ổ D và clone repo về đó:

cd /d

mkdir -p NTS234/dev/shi

cd NTS234/dev/shi

git clone https://git.fsoft.com.vn/fsoft/Citus-Digital-Workplace.git

Bạn đã có một git folder. Các câu lệnh sau view content của folder:

cd Citus-Digital-Workplace

ls -a

Tip: Bạn thấy một folder có tên “chấm git”. Trong SVN bạn thấy folder “chấm svn” có mặt khắp nơi, trong git thì database của git chỉ chứa ở top level thôi, vì vậy nếu muốn extract code bạn chỉ cần xóa .git là xong.

4.2 Update thông tin remote repository

Một đặc điểm chúng ta chưa đề cập đến đó là một local repo có thể track nhiều remote repo. Ở bước 4.1 chúng ta tạo ra một local repo bằng cách clone một remote repo đại diện bởi URL trong câu lệnh. Remote này có tên default là origin. Bạn có thể add thêm các remote khác với URL khác và tên khác.

Điều này có nghĩa là code trên local của bạn có thể được compare/pull và push lên nhiều git repo khác nhau (cùng hay khác server, cùng hay khác organization, cùng server và organization nhưng khác repo). Tất nhiên điều này không được công ty cho phép và guideline này cũng không hướng dẫn bạn làm việc với nhiều remote repo.

Bạn chỉ cần nhớ câu lệnh sau đây update thông tin của remote repo:

git fetch origin

Ngay khi vừa clone xong, thông tin của origin đã được fetch rồi nhưng sau này bạn hãy dùng câu lệnh này thường xuyên.

Tranh thủ một chút khái niệm:

Organization: trong URL của remote repo chúng ta đang dùng, fsoft là organization, fsoft/Citus-Digital-Workplace là repo.

Về mặt kĩ thuật, “project” không phải là một khái niệm trong cấu trúc dữ liệu của git. Một project có thể có nhiều repo, hoặc như trong trường hợp Citus-Digital-Workplace, thì repo gồm nhiều project.

Thông thường các module code không có liên quan với nhau sẽ được đặt dưới các repo khác nhau. Các branch sẽ tương ứng với các version khác nhau của cùng một software hay library.

Với fsoft/CDW, chúng ta khiêm tốn chỉ xin cấp một repo và nhét chung rất nhiều project vào. Điều này dẫn đến chúng ta buộc phải tạo ra rất nhiều nhóm branch và branch master thì sẽ không được

keep track với một source code nào cả. Vì vậy, các developer hãy chú ý name của các branch sao cho phản ánh đúng project/software của nó nhé.

4.3 Xem danh sách remote branch, cách đặt tên branch

git branch -r ←- Câu lệnh git branch với option -r cho bạn xem danh sách các remote branch.

Như trên đã nói, tên branch theo thông lệ phản ánh version của software. Các từ thường dùng trong tên branch:

master thường là main stream (snapshot mới nhất nhưng stable và chính thức).

stable/1.0 là version 1.0 đã release.

features/tên_feature là nhánh POC (thử nghiệm feature mới).

Chúng ta dùng một repo cho nhiều software không liên quan, lịch sử commit của chúng bị trộn lẫn với nhau. Để khắc phục phần nào nhược điểm này chúng ta bổ sung tên của software/project vào đầu tên branch. Sau đây là một số gợi ý:

CitusS/main coi là master của CitusS.

CitusS/stable/1.0 là một bản baseline, well-tested.

CitusS/stable/projectA là baseline cho tool đã dùng trong dự án A, dự án đã kết thúc, đang hoặc có thể phải maintain trong tương lai, chúng ta cần baseline.

CitusS/projectB/dev là nhánh customize cho dự án B đang phát triển.

CitusS/feature/console là nhánh thử nghiệm tool CitusS bản console.

MinionJ/test/jdk6 : Giả sử MinionJ là một tool Java đã được viết, build và dùng với JDK 7. Bây giờ chúng ta cần biết nó có hoạt động tốt với JDK 6 hay JDK 8 hay không, chúng ta tạo nhánh test cho mục đích này và nếu cần sẽ tạo nhánh feature để fix bug. Tương tự với version của OS, version của .Net hay SharePoint, hay Nintex, …

Một lưu ý quan trọng là bạn không thể "mở rộng" tên branch đã có, cụ thể bạn không thể tạo branch "A/A1" nếu như đã có một branch tên "A". Bạn cũng không thể tùy tiện rename branch nếu không biết chắc là không có một tool nào đang tham chiếu branch đó. Vì vậy cho những branch thử nghiệm cá nhân, bạn nên đặt tên mình vào branch nhé.

4.4 Tạo branch dev

Giả sử branch NTS234/dev/shi chưa tồn tại trên remote. Bạn sẽ tạo nó trên local và push nó lên remote.

git checkout -b NTS234/dev/shi origin/NTS234/main

Tại điểm này, một local branch được tạo ra và upstream branch của nó là origin/NTS234/main. Kiểm tra nó bằng một trong 2 câu lệnh:

git status

git branch -vv ←- Option -vv dùng để show upstream branch. Dấu * đánh dấu branch hiện tại.

Kiểm tra danh sách 5 commit mới nhất với câu lệnh git log -5 :

git log -5

Option -5 có thể thay thế bằng số commit bạn muốn xem. Nếu không có option này, Git Bash hiển thị các commit hết màn hình và chờ bạn nhấn phím Down Arrow để tiếp tục xem, nhấn phím q để thôi.

Tới đây bạn có thể code, commit rồi mới push. Nhưng push ngay thôi nhỉ:

git push -u origin NTS/dev/shi

Tại điểm này, một remote branch được tạo ra **cùng tên** với local branch của bạn, đồng thời local branch của bạn sẽ được tự động chuyển upstream sang branch mới. Kiểm tra lại bằng lệnh git branch -r, git branch -vv và bằng web.

Tip: Option -u thực hiện tự động chuyển upstream. Trong những lần commit và push sau bạn không cần dùng option -u nữa nhé.

4.5 Pull, commit và push

Bạn và các developer khác đã có thể checkout branch dev và bắt đầu code. Xem mục 6 để biết các câu lệnh cần thiết nhé.

Import project vào IDE của bạn, thực hiện các bước cần thiết để đảm bảo project có thể build success (nếu không, hãy blame cái branch starting point nhé).

5. Tạo project mới (branch main)

Như đã nói, “project” không phải là khái niệm kĩ thuật của Git. Trong mục này chúng ta cũng tạo ra một branch mới, nhưng khác với mục 4. Trong mục 4, starting point là một branch đã có và branch mới kế thừa toàn bộ history của starting point.

Tình huống của mục 5 là bạn tạo một project mới, ví dụ như lần đầu đưa source code của một tool nào đó lên Git, do vậy starting point của nó là empty tức là chưa có gì trong lịch sử commit.

Công việc này nên được làm bởi Git master.

Trong commit đầu tiên nên có file README.md, .gitignore và source code structure của dự án. File README.md nên giới thiệu tên và usage của software. File .gitignore có thể tìm kiếm trên mạng, nếu là cho Visual Studio thì bạn có thể copy từ các nhánh khác.

Trước hết, tạo working folder và clone repo tương tự như ở mục 4. Giả sử tên software của bạn là ToolA (tên software chứ không phải tên project mà tool phục vụ nhé).

mkdir -p ToolA/main

cd ToolA/main

git clone https://git.fsoft.com.vn/fsoft/Citus-Digital-Workplace.git

cd Citus-Digital-Workplace

5.1 Tạo orphan branch

git checkout --orphan ToolA/main

git rm --cached -r .

Lưu ý câu lệnh kết thúc bằng “.”

Tại điểm này bạn có empty history, nhưng không có nghĩa là empty folder. Vì trước khi checkout, bạn đang ở branch master, nên sau 2 câu lệnh trên bạn có toàn bộ file của nhánh master. Câu lệnh git status sẽ giúp bạn nhìn thấy là các file này đang ở trạng thái unstaged.

<<Sửa file README.md và những file bạn muốn, xóa bỏ các file còn lại.>>

<<Tạo và sửa file .gitignore>>

<<Copy source code nếu bạn đã có>>

Bạn có thể tạo một folder <<src>> để chứa source code. Việc này giúp cấu trúc tốt hơn nếu bạn dự định là sau này sẽ có thêm các folder như <<doc>> chẳng hạn.

Kiểm tra file ignore của bạn bằng lệnh git status với option --ignored :

git status –ignored

File ignore là file được dùng rộng rãi bởi community và nếu như source của bạn bị ignored thì source nên được sửa chứ không phải là file ignore. Tuy nhiên với những source code cũ do chưa aware vấn đề này hoặc do tình huống đặc biệt, chúng ta đang có những namespace bị ignored như “Log” (lẽ ra nên đặt là Logging hay Logger) hoặc Packages. Để không phải sửa source code, bạn có thể workaround bằng cách đặt các chỉ định inverse ở cuối file. Ví dụ:

!Fsoft.Citus.Suite/Citus.Core.Common/Log/

!Citus4Word/NotesDesignExtractor/NoteDesignUltilities/Log/

!QAReviewer/QAReviewer/Core/Packages/*

Nên làm như vậy chứ không nên tìm và sửa phần “chuẩn” của file ignore nhé, vì dùng file ignore chuẩn sẽ giúp tiết kiệm thời gian của developer.

5.2 First commit và push

Sau khi đã kiểm tra file ignore, bạn đã sẵn sàng để commit:

git add -A

git commit -a -m “Create project ToolA”

git push -u origin ToolA/main

5.3 Set protected branch

Git giúp cho các developer có thể chia sẻ code của mình nhanh chóng và thuận tiện, vì vậy một số branch sẽ cần được bảo vệ bằng cách hạn chế quyền push. Chúng ta qui ước các nhánh sau đây nên được bảo vệ:

main: là code base cho tất cả các nhánh khác nên các commit đều phải được review và test cẩn thận.

stable: là baseline nên sau khi tạo ra thì sẽ không commit nữa, nếu có fix bug thì phải tạo revision mới.

binary: tương tự stable, nếu có code change thì phải nâng version/revision.

Việc set một branch là protected được thực hiện qua giao diện web. Sau khi push nhánh main lên remote repo, bạn vào web, Settings, Protected.

6. Các câu lệnh hay dùng

6.1. Update thông tin remote

git fetch origin

6.2. Checkout

Ở mục 4, lệnh checkout -b tạo một branch mới thừa kế history của một branch đã có.

Ở mục 5, lệnh checkout --orphan tạo một empty branch.

Nếu branch remote đã được tạo rồi, bạn chỉ tạo local thôi thì dùng câu lệnh checkout -b giống mục 4 nhưng lưu ý tên local giống tên remote:

git checkout -b **ABC** origin/**ABC**

Còn câu lệnh checkout thông thường dùng để chuyển làm việc từ branch này sang branch khác:

git checkout master

git checkout NTS234/main

Lệnh checkout với tên file thì lại dùng để undo (discard) những thay đổi bạn làm với file (mà chưa commit):

git checkout README.md

git checkout file1 file2

6.3. Pull, commit, push

Đây là thủ tục commit hàng ngày:

git checkout branch_name

git fetch origin

git pull

<<code>>

git status

git add -A

git commit -a -m”message tóm tắt nội dung commit”

git push origin branch_name

6.4. Reset, Undo, Clean

Để undo toàn bộ những thay đổi:

git reset --hard

Lệnh reset --hard không giúp bạn xóa bỏ những file mới tạo ra. Dùng lệnh clean để làm việc đó:

git clean -f

Nếu muốn xem trước khi xóa:

git clean -n

6.5. Xem nội dung thay đổi của một file

git diff file_name

6.6. Xem nội dung một commit

Trước hết tìm commit id bằng lệnh git log:

git log (hoặc git log -10 để giới hạn 10 commit cuối cùng)

git show commit_id

6.7 Rename, move, delete file

Như đã giải thích ở mục 2.3.5, với file mà git đã quản lý (đã có history), muốn xóa, đổi tên hay đổi đường dẫn thì bạn hãy dùng git command để giữ được history của file.

git rm file1 ← xóa file

git mv old_path new_path ← đổi tên hoặc đường dẫn

Cố gắng design namespace tốt từ đầu và hạn chế việc refactor trong IDE nhé.