5.US\_states = {"Alabama" => "AL",

"Alaska" => "AK",

"Arizona" => "AZ",

"Arkansas" => "AR",

"California" => "CA",

"Colorado" => "CO",

"Connecticut" => "CT",

"Delaware" => "DE",

"District of Columbia" => "DC",

"Florida" => "FL",

"Georgia" => "GA",

"Hawaii" => "HI",

"Idaho" => "ID",

"Illinois" => "IL",

"Indiana" => "IN",

"Iowa" => "IA",

"Kansas" => "KS",

"Kentucky" => "KY",

"Louisiana" => "LA",

"Maine" => "ME",

"Maryland" => "MD",

"Massachusetts" => "MA",

"Michigan" => "MI",

"Minnesota" => "MN",

"Mississippi" => "MS",

"Missouri" => "MO",

"Montana" => "MT",

"Nebraska" => "NE",

"Nevada" => "NV",

"New Hampshire" => "NH",

"New Jersey" => "NJ",

"New Mexico" => "NM",

"New York" => "NY",

"North Carolina" => "NC",

"North Dakota" => "ND",

"Ohio" => "OH",

"Oklahoma" => "OK",

"Oregon" => "OR",

"Pennsylvania" => "PA",

"Rhode Island" => "RI",

"South Carolina" => "SC",

"South Dakota" => "SD",

"Tennessee" => "TN",

"Texas" => "TX",

"Utah" => "UT",

"Vermont" => "VT",

"Virginia" => "VA",

"Washington" => "WA",

"West Virginia" => "WV",

"Wisconsin" => "WI",

"Wyoming" => "WY"}

puts " Process hash M , N , S"

puts "-----------------------------------"

US\_states.each {|key,value| puts value if value[1] == "A" || value[1] == "E" || value[1] == "I" || value[1] == "O" || value[1] == "U"}

puts "-----------------------------------"

ปล.ได้แต่ทำให้ตัวหลังออกเป็นตัวสระ

11 โปรแกรม version control มีประโยชน์อย่างไร

ระบบที่จัดเก็บการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์หนึ่งหรือหลายไฟล์ และยังช่วยให้ย้อนไฟล์บางไฟล์หรือแม้กระทั่งทั้งโปรเจคกลับไปเป็นเวอร์ชั่นเก่าได้

12.ข้อได้เปรียบของ distributed version control เมื่อเทียบกับ centralized version control คืออะไร

-distributed version control นั้นเมื่อระบบหรือเซิฟเวอร์ล่ม ก็จะสามารถที่จะทำงานได้ต่อ และยังก๊อปปี้ที่เซิร์ฟเวอร์เพื่อกูข้อมูลกลับคืนก็ได้ แต่ Centralized version control นั้นเมื่อเซิฟเวอร์ล่มไม่สามารถทำงานได้ต่อและถ้าฮาร์ดดิสเสียก็ไม่มีการสำรองข้อมูลอีกด้วย

13.ข้อได้เปรียบของ Centralized control เมื่อเทียบกับ distributed version control คืออะไร

-Centralized control จะได้เปรียบตรงที่ มีเซิฟเวอร์เป็นตัวกลางเก็บไฟล์ทั้งหมดไว้ที่เซิฟเวอร์ตัวเดียวและ User คนอื่นสามารถเรียกใช้ไฟล์ได้จาก Server ตัวนั้นได้อย่างง่าย

14.วิธีแก้ก็แค่ลบพวกโค๊ดส่วนเกินออก แล้วแก้ไขใหม่ให้เรียบร้อย จากนั้นลองเช็คสถานะ จะขึ้นประมาณนี้

git status

Your branch and 'origin/master' have diverged,

and have 1 and 1 different commit each, respectively.

(use "git pull" to merge the remote branch into yours)

You have unmerged paths.

(fix conflicts and run "git commit")

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: README.md

no changes added to commit (use "git add" and/or "git commit -a")

ก็ commit และ push ได้ปกติแล้วครับ

git add README.md

git commit -m "fixed conflict on README.md"

git push

เป็นอันเรียบร้อย

15.

1. ทำการ Merge บ่อย ๆ สิ

ปัญหาใหญ่ ๆ ของ Merge conflict เกิดจากจำนวน source code ที่ชนหรือขัดแย้งกันมากเหลือเกิน คำถาม คุณทำการ merge source code กันบ่อยเพียงใด ? 2-3 วัน ? หนึ่งสัปดาห์ ? สองสัปดาห์ ? ครึ่งเดือน ? หนึ่งเดือน ?

สิ่งที่หนึ่งที่บอกได้เลยก็คือ ยิ่งใช้เวลานานเพียงใด ความเสี่ยงก็ยิ่งมากขึ้นเท่านั้น ยิ่งก็ให้เกิดข้อขัดแย้งขนาดใหญ่ และ มากมาย

ดังนั้น ถ้าคุณต้องการหลีกเลี่ยงปัญหาเหล่านี้ ให้ทำการ merge บ่อย ๆ ไปเลย นั่นคือ ทุกครั้งเมื่อคุณทำการเปลี่ยนแปลง หรือ commit source code นั่นเอง จะช่วยลดข้อขัดแย้งต่าง ๆ ลงไปอย่างมาก ถึงจะเกิดข้อขัดแย้ง ก็เป็นเพียงปัญหาเล็ก ๆ ซึ่งสามารถแก้ไขได้อย่างง่ายดาย

หัวใจคือ Fast feedback ครับ

2. เล็ก ๆ สิดี

ว่าด้วยเรื่องของ Single Responsibility Principle (SRP) เป็นหนึ่งในแนวทางการออกแบบระบบงานที่ดี คือ SOLID นั่นคือ ในหนึ่ง class ใน หนึ่ง method นั้น ควรจะมีหน้าที่การทำงานเพียงอย่างเดียวเท่านั้น หรือในแต่ละ class แต่ละ method ควรมีเหตุผลเดียวในการเปลี่ยนแปลงเท่านั้น ผลที่ได้ก็คือ นักพัฒนาจะไม่ทำงาน หรือ เปลี่ยนแปลง source code ที่เดียวกันอย่างแน่นอน ยกเว้นจะทำงานเดียวกัน หรือ ทำงานด้วยกัน ดังนั้น การออกแบบที่ดีมันช่วยให้ทีมทำงานร่วมกันได้อย่างดี ยิ่งแต่ละส่วนการทำงานเล็ก ๆ แล้ว ก็ยิ่งทำให้คุณภาพของการออกแบบระบบดี รวมทั้ง ลดความเสี่ยงจาก Merge conflict อีกด้วย

ลองกลับไปดู source code ของคุณดูสิว่า มีการแก้ไข source code อย่างไรกันบ้าง เช่น มี class ไหนที่ต้องแก้ไขบ่อยมาก ๆ จากทุก ๆ คน ทุก ๆ feature ?

3. การพูดคุย การสื่อสาร มันสำคัญอย่างมาก

Communication is a Key !!

แต่ละคนในทีมพูดคุยกันหรือไม่ ? แต่ละคนในทีมรู้หรือไม่ว่า เพื่อน ๆ แต่ละคนทำงานอะไร ? รู้หรือไม่ว่า แต่ละคนแก้ไข class อะไรกันอยู่ ? รู้หรือไม่ว่า สิ่งที่แก้ไขไปนั้นกระทบใครบ้าง ? ดังนั้น ถ้าคุณไม่รู้ ก็ควรพูดคุยกันซะเพื่อให้รู้ หรือบางครั้งต้องแก้ไข class เดียวกันอยู่ตลอดเวลา ทำไมไม่มานั่งทำงานด้วยกันไปเลยล่ะ !!

แก้ไขการ Merge conflict ได้ง่าย ๆ มากเลยนะ เพียงแค่เราคุยกันในเรื่องที่ควรจะคุยกันมากขึ้น

4. Mob programming

เป็นวิธีการที่ทรงประสิทธิภาพอย่างมาก และเชื่อได้เลยว่า แก้ไขปัญหา Merge conflict ได้ 100% เนื่องจากทุกคนมานั่งทำงานด้วยกัน ใช้เครื่องทำงานเดียวกัน ดังนั้น ไม่มีทางที่ source code จะขัดแย้งกัน แต่มันมีค่าใช้จ่ายที่สูงมาก ๆ ดังนั้นจึงเป็นวิธีการสุดท้ายที่จะทำนั่นเอง

16.Git คืออะไร แตกต่างจาก Github อย่างไร

-git จะทำหน้าที่ในการจัดการเกี่ยวกับเวอร์ชันของซอร์สโค้ดของเรา ไม่ว่าจะเป็นเรื่องการตรวจสอบความแตกต่างระหว่างเวอร์ชัน (diff) การย้อนกลับมาเวอร์ชันก่อนหน้าเมื่อมีปัญหา (revert) หรือตรวจสอบเกี่ยวกับรายละเอียดของแต่ละเวอร์ชัน

-แต่ Github นั้นคือการฝากไฟล์ไว้บนเซิฟเวอร์

17.จุดประสงค์หลักในการ branch คืออะไร

- เพื่อแยกการพัฒนานั้น จะต้องทำการ merge การเปลี่ยนแปลงเข้ามายัง master branch หลังจากที่พัฒนาเสร็จแล้วเสมอ

สิ้นสุดการสนทนาผ่านแชท

18.$ git checkout master

$ git merge hotfix

Updating f42c576..3a0874c

Fast forward

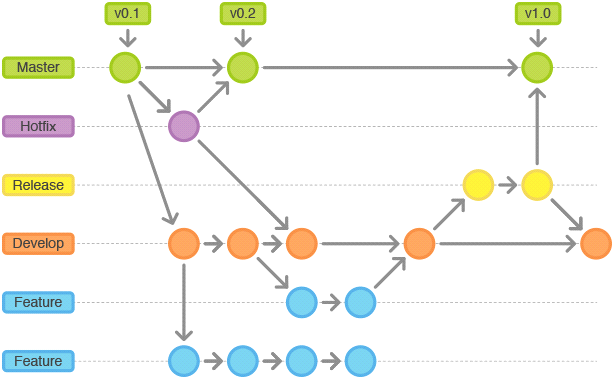
README | 1 -

1 files changed, 0 insertions(+), 1 deletions(-)

คุณจะเห็นคำว่า "Fast forward" ใน merge นั้น เพราะ commit ที่ถูกชี้โดย branch ที่คุณ merge มันเป็น upstream ของ commit ที่คุณอยู่โดยตรง Git ก็เลยขยับ pointer ไปข้างหน้า พูดอีกนัยหนึ่งก็คือ เวลาที่คุณพยายามจะ merge commit ซักอันเข้ากับ commit ที่สามารถไปถึงได้โดยการตาม history ของ commit อันแรก Git จะทำให้ทุกอย่างง่ายขึ้นโดยการขยับ pointer ไปข้างหน้าเพราะมันไม่มีงานที่ถูกแยกออกไปให้ merge สิ่งนี้เรียกว่า "fast forward".

19.git pull ก็คือรวมโค๊ดจาก remote มายัง local โดยที่เราไม่สามารถรู้ได้เลยว่าจะรวมโค๊ดอะไรบ้าง รู้แค่หลังจาก pull เสร็จแล้วนั่นเอง ซึ่งจริงๆแล้ว git pull มันก็คือการทำ git fetch และต่อด้วย git merge อัตโนมัตินั่นเอง

20.workflow มันคือแนวทางที่จะทำให้การทำงานมันราบรื่นนั่นแหละ แต่มันมีแนวทางแนวทางหนึ่งที่เป็นที่นิยม เขาตั้งชื่อมันว่า gitflow



1. เมื่อจะพัฒนาฟีเจอร์ใหม่ ให้แตก branch Feature มาจาก Develop ให้ทีมพัฒนาโค้ดกันใน branch นั้น เมื่อพัฒนาเสร็จให้ merge โค้ดเข้า Develop แล้วลบ branch Feature ทิ้ง

2. เมื่อจะเอาโค้ดขึ้นโปรดักชั่น ให้แตก branch Release ออกมาจาก Develop ให้ Tester ตรวจสอบว่าโปรแกรมทำงานถูกต้องหรือเปล่า มีบักหรือเปล่า หากมีบักก็แก้ใน Branch Release เลย จนเมื่อโปรแกรมถูกต้องสมบูรณ์จึง merge เข้า Master เพื่อเอาขึ้นโปรดักชั่นต่อไป อย่าลืม Tag เวอร์ชั่น และ merge เข้า Develop ด้วยจากนั้นจึงลบ branch Release ทิ้ง

3. ทีนี้หลังจากที่เอาโค้ดขึ้นโปรดักษ์ชั่นแล้ว มันอาจจะพบบักที่ไม่คาดฝัน อาจจะเพราะ environment เครื่องหรืออะไรก็แล้วแต่ ซึ่งจำเป็นต้องรีบแก้ ให้แตก branch Hotfix ออกมาจาก Master แล้วแก้ไขบักซะ หลังจากแก้ไขเสร็จแล้ว ให้ merge โค้ดเข้าไปยัง Master แล้วลบ branch Hotfix ทิ้ง

จากภาพ เราจะเห็นวิธีการตั้งเวอร์ชั่นของโปรแกรมที่เราเขียนด้วย ถ้าหากเป็นการเอาโค้ดจาก Release ขึ้นโปรดักชั่น เขาจะใช้หมายเลขเวอร์ชั่นใหญ่ เช่น 1.0, 2.0, 3.0 แต่ถ้าเป็นการ Hotfix เขาจะใช้เลขเวอร์ชั่นย่อย เช่น 1.1, 1.2, 2.1, 3.1 เป็นต้น

จากที่กล่าวมา 3 ข้อด้านบน จะเห็นว่ามีการ สร้าง branch , merge branch, ลบ branch วนๆซ้ำไปซ้ำมา เวลาคุยกันในทีม มันก็กลายเป็นคำภาษาเทคนิคไป คนที่เขาเสนอแนวทาง gitflow ก็เลยสร้างชุดคำสั่งเสริมเข้าไปใน git เพิ่มเติม เพื่อให้ฟิลลิ่งมันระรื่น feature start, feature finish, release start, release finish, hotfix start, hotfix finish