11

  ช่วยในการจัดการระบบที่คอยจัดการ backup source code ของเรา โดยเก็บเป็นลักษณะ version ต่างๆ เช่นในกรณีเกิดปัญหาขึ้น ก็สามารถหยิบเอา source code ตัวเก่าที่เคยใช้งานได้มาแทน พูดง่ายๆ ก็คล้ายๆ กับ Undo ที่เราทำกันนั่นแหละ แต่ว่า Version Control ยังเป็นตัวกลางที่ทำให้ source code ของแต่ละเครื่อง (แต่ละ programmer) มี source code ที่ตรงกันด้วย ซึ่งโดยปกติทั่วไปแล้ว programmer มักจะทำโปรแกรมให้เสร็จสมบูรณ์ และใช้งานได้ก่อน จึงค่อยโยนขึ้นไปที่ repository เพื่อให้คนอื่นดึงไปใช้ต่อไป

12.

**distributed version control**

* central code repository model คือหมายความง่ายๆ ว่า changes ของ file ทั้งหมดถูกเก็บอยู่ที่ central repo หรือศูนย์กลางที่เดียว ถ้าสมมติว่า central repo เกิดพังขึ้นมาหรือผู้ใช้ไม่สามารถติดต่อกับ central repo ได้ก็จะทำให้ไม่สามารถดึงไฟล์ที่พึ่งเปลี่ยนแปลงมาทีตัวเองได้
* ส่วน distributed version control ที่ใช้ใน BitKeeper และ git นั้น แต่ละคนจะมี copy ของไฟล์ตั้งแต่เริ่มแรกจนท้ายสุด ดังนั้นแต่ละทีมหรือแต่ละคนสามารถ maintain code ได้โดยที่ทุกคนมี copy ของไฟล์ทั้งหมดอยู่ที่ local computer และ change set ของแต่ละ version ที่เกิดขึ้นจากหลายๆ ทีมนั้นสามารถรวมกันได้ ไม่จำเป็นต้องรอ changes copy จาก central repo อีกต่อไป
* ดังนั้น distributed version control จึงเร็วกว่า และผู้ใช้งานก็สามารถแก้ได้ทุกที่ โดยไม่จำเป็นจะต้องติดต่อกับ central repo ก็ได้
* นอกจากนี้เรายังสามารถ fork projects ได้ด้วย แปลง่ายๆ คือดึงโปรเจกต์มาอยู่ในชื่อเราเพื่อแก้ไขในแบบที่เราต้องการหรือใช้สำหรับแก้โค้ดและส่งกลับไปให้กับเจ้าของที่พัฒนาโค้ดก็ได้ การ fork ในที่นี้คือการยอมให้ outside developers นำโค้ดของผู้พัฒนาไปแก้และ pull request กลับไปให้ผู้พัฒนานั่นเอง

13

Centralized version control

* เช่น subversion (svn)
* หลักการง่าย ๆ คือมี repository คอย track กับเก็บประวัติการแก้ไขอยู่ตรงกลาง ผู้ใช้งานจะทำการ pull มาจาก repository แก้ไขจนเสร็จและแน่ใจว่าทำงานได้ จึงค่อย commit กับมาที่ repository
* คนอื่น ๆ ก็จะเห็นสิ่งที่เรา commit ไปแล้ว pull มาแก้ได้ต่อ

14

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

15

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

16.

Git จะช่วยผู้ใช้ทราบถึงการเปลี่ยนแปลงระหว่าง code ดั่งเดิม กับ code ที่พัฒนาขึ้นมา Git ไม่ใช่เจ้าหรือระบบเดียวที่เป็น version control ครับแต่ยังมี เช่น [SVN](https://subversion.apache.org/) แต่ Git เป็นที่นิยมมาก ถ้าไม่มี Github การจะทำงานร่วมกันกับ Git จะต้องพิมพ์ผ่าน command หรือ Terminal เท่านั้น ดังนั้น Github จึงถูกพัฒนาขึ้นมาเพื่อทำให้การทำงานกับ Git สะดวกและง่ายดายมากขึ้น Github คือ application ทำงานร่วมกันกับ Git โดยมี interface ที่ทำใช้งานง่าย

17.

branch เป็น feature ที่ช่วยให้นักพัฒนาสามารถที่จะทำงานได้สะดวกขึ้น ยกตัวอย่างเช่น เรามีโค๊ดที่ดีอยู่แล้ว แต่อยากจะทดลองอะไรนิดๆหน่อย หรือแก้ไขอะไรก็ตาม ไม่ให้กระทบกับตัวงานหลัก ก็เพียงแค่สร้าง branch ใหม่ขึ้นมา เมื่อแก้ไขหรือทำอะไรเสร็จแล้ว ก็ค่อยเซฟกลับมาที่ master เหมือนเดิม

18

Fast forward เวลาสั่ง git pull มานานแล้ว, มาอ่าน git-doc ก็เลยเข้าใจขึ้น  
เวลาเราสั่ง git merge, การทำงานของมันสามารถแบ่งได้เป็น 3 ประเภท

* ถ้า merged commit ที่เราดึงมา อยู่ใน Head(current tree ของเรา) ของเราแล้ว, ก็จะแสดงผลลัพท์ "Already up-to-date." แล้วก็จบการทำงาน
* ถ้า Head ของเราอยู่ใน commits ที่ดึงมา, case นี้มักเกิดจากคำสั่ง "git pull" เพื่อดึง code จากต้นน้ำมา update code(ที่ไม่มีการเปลี่ยนแปลง) ของเรา, สิ่งที่เกิดขึ้นก็คือ git จะ update HEAD ของเราให้ตรงตาม HEAD ของ merged commit (โดยไม่มีการสร้าง commit object ใหม่ขึ้นมา) มีศัพท์เฉพาะสำหรับกรณีนี้ว่า "Fast-forward"
* สุดท้ายเป็นกรณีที่เกิดการ merge จริงๆ นั่นคือ ตัว HEAD ของเรา independent กับ merged commit, ดังนั้นกรณนี้จะเกิดการ merge จริง และมีการสร้าง commmit object ใหม่ขึ้นมา

19

git pull ใช้ดึงความเปลี่ยนแปลงจาก remote มายัง local และรวมเข้าด้วยกัน (มีค่าเท่ากับfetch+merge

20

คือการทำ Git Workflow นั้นให้มองภาพการพัฒนาซอฟต์แวร์ที่มีนักพัฒนาหลายคนร่วมในโครงการ การพัฒนาฟีเจอร์ การ release version ใหม่ การแก้ bug ซึ่ง git workflow ก็คือการนำสิ่งเหล่านั้นมาพัฒนาในรูปแบบของ workflow โดยใช้ git ช่วยในการพัฒนา