ข้อ 11

Version control คือ ระบบที่คอยจัดการ 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

- source code มีระเบียบ

- ย้อนกลับ เมื่อเกิดข้อผิดพลาดได้

- เรียกงานได้จากทุกที่ของมุมโลก ขอมีเพียง internet

- ลดพื้นที่ความจำในสมอง แบ่งมาใช้กับส่วนคิดมากขึ้น

ข้อ 12

ระบบ Version Control ที่ไม่มี Server หลักเป็นศูนย์กลาง แต่ละเครื่องของแต่ละคนก็จะเก็บข้อมูลของโปรเจคทั้งหมดไว้ หมายความว่าข้อมูล History ของโปรเจคก็จะอยู่ในทุกเครื่อง และก็มีการ Share Changes กัน คล้ายๆ ระบบ Peer-to-Peer ครับ แนวคิดคือเวลาข้อมูลของเครื่องหนึ่งมีการเปลี่ยนแปลงก็จะมีการติดต่อไปยังเครื่องๆ อื่นๆ เพื่ออัพเดทให้ทุกคนเหมือนกัน ข้อดีคือมันกระจายความเสี่ยงของ Project History จะหายได้ครับ เพราะพังไปเครื่องหนึ่งเครื่องอื่นๆ ก็ยังมีอยู่ไม่หายไปไหน

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

ข้อ 13

ระบบ Version Control ที่มี Server หลักเป็นศูนย์กลางในการเก็บข้อมูลของโปรเจคทั้งหมด Client แต่ละเครื่องจะเป็นแค่ Working Copy เท่านั้น วิธีทำงานคร่าวๆ เริ่มต้นเราก็สร้างโปรเจคขึ้นมาบน Server หลังจากนั้นแต่ละคนก็มาดาวน์โหลดลงเครื่องของตัวเอง เวลามีการเปลี่ยนแปลงอะไรก็อัพโหลดกลับขึ้นไปบน Server แล้วคนอื่นๆ ก็จะสามารถดาวน์โหลด Update นั้นมายังเครื่องของตัวเองได้ ข้อเสียคือถ้า Server เดี้ยงไป Project History ทั้งหมดก็จะหายไปหมด ยี่ห้อที่เป็นที่นิยมกันในประเภทนี้ได้แก่

ทั้ง SCCS, RCS, CVS, SVN ที่กล่าวในช่วงแรกทั้งหมดนั้นใช้ central code repository model คือหมายความง่ายๆ ว่า changes ของ file ทั้งหมดถูกเก็บอยู่ที่ central repo หรือศูนย์กลางที่เดียว ถ้าสมมติว่า central repo เกิดพังขึ้นมาหรือผู้ใช้ไม่สามารถติดต่อกับ central repo ได้ก็จะทำให้ไม่สามารถดึงไฟล์ที่พึ่งเปลี่ยนแปลงมาทีตัวเองได้

ข้อ 14

1. Code ที่คาดว่าสมบูรณ์ ถูก Merge ลง Master Branch ลงไป ก่อนที่จะค้นพบว่า เป็นโค้ดที่มีปัญหา

2.โค้ดถูก Revert

3. โค้ดที่ Reverted เสร็จแล้ว ถูก Merge กลับเข้าไป เพื่อให้ Master Branch Stable

4. แตก Branch ใหม่ออกมาเอาไว้ซ่อมโค้ด ซึ่ง Reverted โค้ดที่เพิ่ง Reverted ไป ให้หน้าตาเหมือน Feature Branch ตอนยังไม่ได้ Merge ลง Master ที่ (1)

5.แก้ส่วนที่ต้องการแก้เพิ่มเข้าไป

6.Merge กลับเข้าไปอีกครั้งเมื่อโค้ดสมบูรณ์

ปัญหาก็คือ เมื่อเรา Merge กลับเข้าไปในขั้นตอนที่ (6) โค้ดส่วนใหญ่ที่ถูกแก้ กลับมี Comment Message ไว้ว่า Revert ...... ซึ่งไม่ตรงกับเจตนา ที่เรา Comment ไว้แต่แรก ก่อนที่เราจะ Merge ลง Master Branch ที่ (1)

ข้อ 15

อย่าคอมมิตตู้มเดียว เช่น ทำฟีเจอร์ X ตอนทำจะหั่นยิบย่อยให้ได้มากที่สุด สมมุติมันต้องสร้างคลาส A ขึ้นมา ก็จะทำ Skeleton มันขึ้นมาก่อน ยังไม่ต้อง Implement อะไร วิธีนี้ต่อให้มัน Conflict ก็จะ Solve ได้ไม่ยาก

ข้อ 16

Git คือ Version Control ตัวหนึ่ง ซึ่งเป็นระบบที่มีหน้าที่ในการจัดเก็บการเปลี่ยนแปลงของไฟล์ในโปรเจ็คเรา มีการ backup code ให้เรา สามารถที่จะเรียกดูหรือย้อนกลับไปดูเวอร์ชั่นต่างๆของโปรเจ็คที่ใด เวลาใดก็ได้ หรือแม้แต่ดูว่าไฟล์นั้นๆใครเป็นคนเพิ่มหรือแก้ไข หรือว่าจะดูว่าไฟล์นั้นๆถูกเขียนโดยใครบ้างก็สามารถทำได้ ฉะนั้น Version Control ก็เหมาะอย่างยิ่งสำหรับนักพัฒนาไม่ว่าจะเป็นคนเดียวโดยเฉพาะอย่างยิ่งจะมีประสิทธิภาพมากหากเป็นการพัฒนาเป็นทีม

GitHub คือ เว็บที่ให้บริการพื้นที่จัดเก็บโครงการโอเพ่นซอร์สด้วยระบบควบคุมเวอร์ชันแบบ Git โดยมีจุดประสงค์หลักคือ ทำให้การแบ่งปันและพัฒนาโครงการต่างๆด้วยกันเป็นไปได้ง่ายๆ ในบทแทรกนี้ เราจะมาดูวิธีการหลายๆอย่างที่คุณสามารถใช้ GitHub เพื่อทำตามที่หนังสือ Discover Meteor บอกได้

Github เป็นเว็บเซิฟเวอร์ที่ให้บริการในการฝากไฟล์ Git (ทั่วโลกมักนิยมใช้ในการเก็บโปรเจ็ค Open Source ต่างๆ ที่ดังๆ ไม่ว่าจะเป็น Bootstrap, Rails, Node.js, Angular เป็นต้น)

ข้อ 17

branch : เหมาะสำหรับการทำโปรเจ็ค ที่มีหลายๆ features ก็ทำการแยก branch ออกมา เช่น branch-dev, branch-release เป็นต้น แปลตรง ๆ คือการแตกกิ่งก้านสาขาของต้นไม้ การแตก branch จะเป็นการก๊อปปี้กลุ่มของไฟล์ที่มีอยู่ปัจจุบันไปแตกเป็นกิ่งก้านสาขาอันใหม่ ช่วยให้นักพัฒนาสามารถพัฒนาบน branch ใหม่ได้โดยไม่มีการเข้าไปยุ่งกับส่วนของระบบที่ทำงานได้ใน branch หลัก

ข้อ 18

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

ข้อ 19

git pull ใช้ดึงความเปลี่ยนแปลงจาก remote มายัง local และรวมเข้าด้วยกัน (มีค่าเท่ากับ fetch+merge) คือรวมโค๊ดจาก remote มายัง local โดยที่เราไม่สามารถรู้ได้เลยว่าจะรวมโค๊ดอะไรบ้าง รู้แค่หลังจาก pull เสร็จแล้วนั่นเอง ซึ่งจริงๆแล้ว git pull มันก็คือการทำ git fetch และต่อด้วย git merge อัตโนมัตินั่นเอง

ข้อ 20

develop branch เป็น branch ที่มักถูกสร้างขึ้นมาจากทีม เป็น branch หลักสำหรับการพัฒนา ดังนั้นถ้าใช้รูปแบบการทำงานแบบ feature branch แล้วนั้น ทุกๆ feature branch จะต้องถูกสร้างมาจาก develop branch เสมอ เมื่อทำการพัฒนาบน feature branch เสร็จแล้ว ก็ต้องทำการลบ feature branch และ merge code กลับไปยัง develop branch เสมอ