11. ตรวจสอบว่าในแต่ละครั้งซอสร์สโค้ดมันมีอะไรเปลี่ยนแปลงไปบ้าง แล้วเก็บความเปลี่ยนแปลงนั้นไว้ ตั้งเป็น version ตามด้วยตัวเลขที่เพิ่มขึ้นไปเรื่อยๆ มันยังบันทึกไว้ด้วยใครเป็นคนเขียนโค้ดบรรทัดไหน เวลาเมื่อไร เป็นต้น ทั้งหมดจะง่ายต่อการหา การเปลี่ยนแปลงข้อโค้ด

12. มีเซิร์ฟเวอร์กลางที่เก็บไฟล์ทั้งหมดไว้ในที่เดียวและผู้ใช้หลาย ๆ คนสามารถต่อเข้ามาเพื่อดึงไฟล์จากศูนย์กลางนี้ไปแก้ไขได้ ระบบการทำงานแบบรวมศูนย์นี้ได้ถูกนำมาใช้เป็นเวลานานหลายปี

13. ไม่เพียงได้ก๊อปปี้ล่าสุดของไฟล์เท่านั้น แต่ได้ทั้งก๊อปปี้ของ repository เลย หมายความว่าถึงแม้ว่าเซิร์ฟเวอร์จะเสีย client ก็ยังสามารถทำงานร่วมกันได้ต่อไป และ repository เหล่านี้ของ client ยังสามารถถูกก๊อปปี้กลับไปที่เซิร์ฟเวอร์เพื่อกูข้อมูลกลับคืนก็ได้ การ checkout แต่ละครั้งคือการทำสำรองข้อมูลทั้งหมดแบบเต็ม ๆ

14. การแก้ Conflict

ก่อนอื่น ลองมาดูกันว่า Conflict จะเกิดขึ้นได้อย่างไรครับ เริ่มจาก ผมมี Repository ที่มีไฟล์อยู๋ไฟล์หนึ่ง



และผมมีนักพัฒนาอยู่สองคน ชื่อ Nant และ John นะครับ ซึ่งทั้งคู่ ต่างก็ Checkout Repo นี่ไว้ในโฟลเดอร์ของตัวเอง



จากนั้น Nant ก็ทำการแก้ไขไฟล์ โดยการเพิ่มฟังก์ชั่น ไปที่บรรทัดที่ 50



ส่วน John ก็ไปเพิ่มฟังก์ชั่น Test ไว้ที่ด้านล่างของไฟล์



จากนั้น John ก็ทำการ Commit ก่อน



แล้วถ้า Nant จะ Commit บ้าง ก็จะ Commit ไม่ได้ และ SVN ก็จะแนะนำว่า ให้ผมสั่ง Update ก่อน เพราะว่าไฟล์มัน Out-of-Date ไปแล้ว



จากนั้น เมื่อผมสั่ง Update แล้ว ผมก็จะได้ไฟล์ AutoCompleteBox.cs ที่ Merge สำเร็จ แต่มีฟังก์ชั่น Test สองครั้ง ทำให้คอมไพล์ไม่ผ่าน ก็จะเป็นหน้าที่ของผม ที่จะต้องไปสื่อสารกับคุณ John เขาว่า เราจะจัดการยังไง และก็สั่ง Lock ไฟล์นี้ไว้ก่อน เพื่อป้องกันใครมาแก้ไขเพิ่ม ระหว่างเราตกลงกันอยู่ เมื่อเรียบร้อย ผมจึงจะสั่ง Commit ครับ แล้วค่อยปลดล็อค แล้วคุณ John ก็สั่ง Update อีกครั้ง เพื่อให้ไฟล์ของเขา เป็นตัวล่าสุด ที่แก้ไขให้คอมไพล์ได้แล้ว



ทีนี้ ผมก็แก้ไขฟังก์ชั่น Test ต่อไป แล้วสั่ง Commit



แต่คุณ John ก็แก้ไขเหมือนกัน แล้วพอจะ Commit คุณ John ก็จะต้อง Update ก่อน



แต่เนื่องจากเป็นการแก้ไข ที่บริเวณเดียวกัน (บางครั้ง แก้ไขในบรรทัดเดียวกัน ก็ยังสามารถ Merge ได้นะครับ) คือ true กับ false ก็จะเกิดการ Conflict ครับ



สังเกตว่า จะมีไอคอนแสดงเลยว่า ไฟล์นี้ กำลัง Conflict พร้อมกับมีไฟล์ต้นฉบับ เพื่อใช้ในการแก้ Conflict มาให้เห็นด้วย



เนื่องจากว่าในการแก้ไข หรือ Commit แต่ละครั้ง SVN จะไม่ได้จำ Revision เฉพาะไฟล์ใด ไฟล์หนึ่ง แต่จะเป็นการจำการเปลี่ยนแปลงของทั้ง Repository ดังนั้น ถ้าเกิดเรา Commit หลายๆ ไฟล์พร้อมกัน ในการ Update ก็เป็นไปได้ที่จะเกิดการ Conflict หลายๆ ไฟล์พร้อมกัน ซึ่งเราก็ควรจะแก้ให้หมดก่อน แล้วค่อย Commit ครับ

ทีนี้ บางท่านอาจจะบอกว่า ถ้าอย่างนั้น ไล่ Commit ทีละไฟล์จะดีกว่าหรือเปล่า ผมขอแนะนำว่า อย่าครับ เพราะบางครั้ง การเปลี่ยนแปลงที่เราทำ อาจจะกระทบไปหลายๆ ไฟล์ และถ้าเราพบว่า การเปลี่ยนแปลงที่เราทำ มันไปทำให้เกิดปัญหา แล้วต้องการ Revert (ย้อนเวลา) กลับไป Revision ก่อน เราก็จะต้องจำให้ได้ ว่า Commit ไฟล์ไหนไปบ้าง แล้วไล่ Revert ให้ครบ แทนที่จะเป็น Revert การเปลี่ยนแปลงที่เกิดขึ้นทั้งหมด ซึ่งนี่คือจุดดีของ SVN ด้วยครับ

ทีนี้ การแก้ Conflict ก็ทำได้โดยการคลิ๊กขวาที่ไฟล์ แล้วเลือก Edit Conflict



จากนั้น เราก็จะพบหน้าต่างแบบนี้ครับ เรียกว่า TortoiseMerge



สิ่งที่เราต้องทำก็คือ กด Ctrl+ลูกศรขึ้น หรือ ลง เพื่อไล่ดูจุดที่มัน Conflict กันทั้งหมด ซึ่งบางครั้ง อาจจะไม่ได้เป็นแค่บรรทัดเดียว แต่เป็นทั้งบล็อค หลายๆ บรรทัด แล้วจัดการแก้ไขครับ โดยการดูจากหน้าต่างซ้ายมือ (ไฟล์นี้ ใน SVN) กับทางขวามือ (ไฟล์ในเครื่องเรา) แล้วเลือกว่า จะให้ผลลัพธ์การ Merge มาจากด้านไหน โดยการกด Use “Theirs” หรือ Use “Mine” บนทูลบาร์ ตรงนี้ เราก็คงต้องเอาคุณ John มานั่งคุยกับผมครับ ว่าจะ Merge กันอย่างไรดี



แล้วตรงที่ Conflict ในหน้าต่าง Merged ก็จะกลายเป็นสีเขียวครับ



หรือถ้าไม่สามารถเลือกข้างได้ จะพิมพ์ลงไปเลยก็ได้เหมือนกันครับ เมื่อเรียบร้อยแล้ว กด Save แล้วทำการ Flag บอก SVN ว่า ไฟล์นี้แก้ไขเรียบร้อยแล้ว ขั้นตอนนี้ขึ้นอยู่กับดุลยพินิจของเรานะครับ ว่าจะให้มันผ่านหรือเปล่า เพราะยังไง SVN ก็บังคับเราไมได้ จะให้มันผ่านเลย โดยทียังไม่แก้ Conflict ก็ยังได้ครับ



เมื่อถึงจุดนี้แล้ว ก็สามารถ Commit ได้ตามปกติครับ

15.

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

17. เป็นเรื่องปกติที่ stable app จะต้องมีการแก้บั๊กหลัง roll out และในเวลาเดียวกัน เราก็อาจจะต้องเพิ่ม features ใหม่ๆที่อาจจะทำให้ app ไม่ stable สิ่งที่นิยมทำใน revisioning control คือการแยก branch ใหม่เพื่อให้เราเริ่มโค้ด features ใหม่ได้ ส่วน app ที่ stable ก็จะอยู่ใน master branch เพื่อให้งานดำเนินต่อไปได้โดยไม่ต้องรอแก้บั๊กใน master ให้เสร็จก่อน ใน Git เราใช้คำสั่ง branch เพื่อแตก branch ใหม่ออกมา

Branch ใหม่ที่แยกออกมาก็จะมี history เป็นของมันเอง เราสามารถ checkout และ commit สิ่งใหม่ๆใน branch นี้ได้ นั่นคือ ก็เหมือนใน local repository หนี่งๆ เราสามารถมี sub repository ย่อยๆนั่นเอง เพียงแต่เราเรียก sub repository พวกนี้ว่า branch

พอถีงเวลาที่เราเสร็จการแก้ master branch แล้ว เราสามารถใช้คำสั่ง merge เพื่อรวมงานของ master และ new features branch เข้าด้วยกัน ไม่ต่างจากการ merge remote และ local repository ข้างต้น

18. If Master has not diverged, instead of creating a new commit, git will just point master to the latest commit of the feature branch.

19. จะดึงสิ่งใหม่ๆจาก origin ลงมา merge ทั้งบน clone repository และ working directory โดยทันที หากเป็นมี conflict จากการ merge ใน working directory เราต้อง resolve conflict นั้นๆก่อนจะ commit ได้ต่อไป