**Version controlและgit**

11.โปรแกรม version control มีประโยชน์อย่างไร

ตอบ  ตัวควบคุมเวอชั่น คือตัวที่จะคอยตรวจสอบว่าในแต่ละครั้งซอสร์สโค้ดมันมีอะไรเปลี่ยนแปลงไปบ้าง แล้วเก็บความเปลี่ยนแปลงนั้นไว้ ตั้งเป็น version ตามด้วยตัวเลขที่เพิ่มขึ้นไปเรื่อยๆ เหมือนกับการที่เราเซฟโค้ดไว้เป็นก้อปปี้ๆนั่นแหละ แต่ version control มันมีระบบการทำงานภายในที่เจ๋งกว่าแค่การก้อปปี้ข้อมูลไว้เป็นช่วงๆ มันยังบันทึกไว้ด้วยใครเป็นคนเขียนโค้ดบรรทัดไหน เวลาเมื่อไร เป็นต้น

มันทำแบ้กอัพให้เราเอง ถ้าเราทำอะไรผิดขึ้นมา เราสามารถจะย้อนมันกลับไปตอนที่ยังทำงานได้ดีๆ อย่างสบายๆ

12.ข้อได้เปรียบของ distributed version control เมื่อเทียบกับ centralized version control คืออะไร

* 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
* 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.ข้อได้เปรียบของ centralized version control เมื่อเทียบกับ ditstributed version controlคืออะไร

ตอบ centralized version control คือระบบ 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

* [CVS](http://savannah.nongnu.org/projects/cvs/)
* [SVN](https://subversion.apache.org/)
* [Perforce](http://www.perforce.com/)
* [Team Foundation Server](http://www.google.co.th/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CCMQFjAA&url=http%3A%2F%2Fmsdn.microsoft.com%2Fen-us%2Fvstudio%2Fff637362.aspx&ei=lvhiVP_SJsfiuQTE4ILAAw&usg=AFQjCNHYGbQJJK6uruHJixZ7sZH4yU-jpw&bvm=bv.79189006,d.c2E)

distributed version control คือระบบ 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

* [Git](http://git-scm.com/)
* [Mercurial](http://mercurial.selenic.com/)
* [Aegis](http://aegis.sourceforge.net/)
* [BitKeeper](http://www.bitkeeper.com/)

14.บอกแนวทางในการแก้ไข conflict ที่จะเกิดขึ้นเมื่อมีการ merge โปรแกรมของผู้พัฒนาหลายๆคนเข้าด้วยกัน

ตอบ โดยปกติแล้ว git merge จะรวมโค๊ดให้เราเองอัตโนมัติ แต่ก็จะมีข้อยกเว้นเมื่อ แก้ไขไฟล์เดียวๆกัน ลองนึกถึงกรณีที่เราและเพื่อนร่วมทีม แก้ไขไฟล์เดียวกัน Git จะเกิดการ conflict เมื่อเราจะ merge โค๊ด โดยไม่รู้ว่าจะใช้โค๊ดของเราหรือของเพื่อน วิธีแก้ก็คือ ทำการ edit แล้ว commit ไปใหม่นั่นเอง

15. บอกแนวทางในการลด conflict ที่จะเกิดขึ้นจากการ merge โปรแกรมของผูพัฒนาหลายๆคนเข้าด้วยกัน

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

16.จุดประสงคืหลักในการ branchคืออะไร

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

17.git คืออะไร แตกต่างจาก githup อย่างไร

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

Git ก็คือการ *คอมมิท* คุณสามารถมองได้ว่า คอมมิทก็คือภาพถ่ายโค้ดของโปรแกรมคุณในขณะหนึ่ง

แทนที่เราจะให้คุณเห็นโค้ดสำเร็จของแอพ Microscope ในครั้งเดียว เราก็ถ่ายภาพโค้ดของโปรแกรมไว้ทีละขั้นจนครบทุกขั้น ซึ่งคุณสามารถเข้าไปดูออนไลน์ได้ที่ GitHub

18. fast forward merge คืออะไรและทำไมการ push ไปที่ remote repo จึงควรจะต้อง merge แบบนี้

ตอบ เห็นคำว่า 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.หน้าที่หลักของคำสั่ง git pull คืออะไร

ตอบจริงๆแล้ว git pull ก็คือรวมโค๊ดจาก remote มายัง local โดยที่เราไม่สามารถรู้ได้เลยว่าจะรวมโค๊ดอะไรบ้าง รู้แค่หลังจาก pull เสร็จแล้วนั่นเอง ซึ่งจริงๆแล้ว git pull มันก็คือการทำ git fetch และต่อด้วยgit merge อัตโนมัตินั่นเอง

20.แผนภาพด้านล่างนี้ต้องการสื่อความหมายอย่างไร

[](http://www.select2web.com/wp-content/uploads/git-workflow-release-cycle-4maintenance.png)ตอบ คือ 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  อ้างอิง<https://github.com/nvie/gitflow/wiki/Command-Line-Arguments>

เบื้องหลังการทำงานของคำสั่งก็คือ แตก branch, merge โค้ด, ลบ branch ให้นั่นเอง

การจะใช้ gitflow นั้น ให้พยายามทำความเข้าใจจากภาพที่เขาวาดอธิบายแหละว่า แนวความคิดมันเป็นยังไง  เราอาจจะไม่ใช้คำสั่งตระกูล gitflow ( feature start, feature finish, release start, release finish, hotfix start, hotfix finish ) ก็ได้ อาศัยการแตก branch, merge โค้ด, ลบ branch เอง มันก็ยังเป็น gitflow อยู่ดี