**11.โปรแกรม Version Control มีประโยชน์อย่างไร**

**ตอบ เป็นระบบที่จัดเก็บการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์หนึ่งหรือหลายไฟล์ เพื่อเรียกเวอร์ชั่นใดเวอร์ชั่นหนึ่งกลับมาดูเมื่อไรก็ได้ สามารถเปรียบเทียบการแก้ไขที่เกิดขึ้นในอดีต ดูว่าใครเป็นคนแก้ไขคนสุดท้ายที่อาจทำให้เกิดปัญหา แก้ไขเมื่อไร ฯลฯ และยังสามารถกู้คืนไฟล์ที่ถูกลบหรือทำเสียโดยไม่ตั้งใจได้**

**12.ข้อได้เปรียบของ Distributed Version Control เมื่อเทียบกับ Centralized Version Control คืออะไร**

**ตอบ 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**

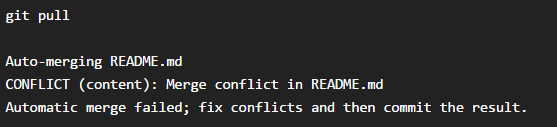
**ดังนั้น distributed version control จึงเร็วกว่า และผู้ใช้งานก็สามารถแก้ได้ทุกที่ โดยไม่จำเป็นจะต้องติดต่อกับ central repo ก็ได้**

**13.ข้อได้เปรียบของ Centralized Version Control เมื่อเทียบกับ Distributed Version Control คืออะไร**

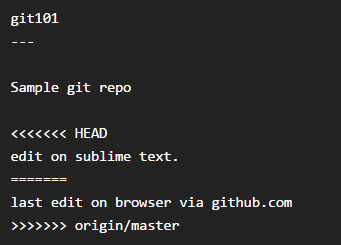
**ตอบ Centralized Version Control คือ changes ของ file ทั้งหมดถูกเก็บอยู่ที่ central repo หรือศูนย์กลางที่เดียวที่เก็บไฟล์ทั้งหมดไว้ในที่เดียวและผู้ใช้หลาย ๆ คนสามารถต่อเข้ามาเพื่อดึงไฟล์จากศูนย์กลางนี้ไปแก้ไขได้**

**14.บอกแนวทางในการแก้ไข conflict ที่จะเกิดขึ้นเมื่อมีการ merge โปรแกรมของผู้พัฒนาหลายๆคนเข้าด้วยกัน**

**ตอบ วิธีแก้ก็คือ ทำการ edit แล้ว commit ไปใหม่นั่นเอง**

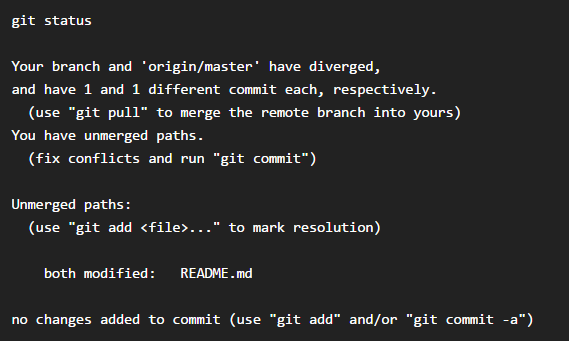
****

**และตัวอย่างไฟล์**

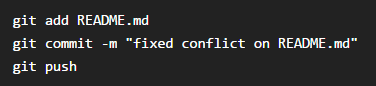
****

**format ของไฟล์ conflict จะถูกขั้นด้วย <<<<<<< HEAD จนถึง ======= สำหรับโค๊ดส่วนที่เราแก้ไข และ =======ถึง >>>>>>>> branch\_name ส่วนที่เป็นโค๊ดของคนอื่นๆ/branch อื่น**

**วิธีแก้ก็แค่ลบพวกโค๊ดส่วนเกินออก แล้วแก้ไขใหม่ให้เรียบร้อย จากนั้นลองเช็คสถานะ จะขึ้นประมาณนี้**

****

**ก็ commit และ push ได้ปกติแล้วครับ**

****

**เป็นอันเรียบร้อย**

**15. บอกแนวทางในการลด conflict ที่จะเกิดขึ้นเมื่อมีการ merge โปรแกรมของผู้พัฒนาหลายๆคนเข้าด้วยกัน**

**ตอบ ลดการเกิด conflict ที่จะเกิดขึ้นเมื่อมีการ merge คือ การ merge บ่อยๆ จะทำให้เล็ดลงเมื่อเกิดปัญหาสามารถแก้ไขได้ง่ายกว่า**

**16.Git คืออะไร แตกต่างจาก Github อย่างไร**

**ตอบ git เป็นเทคโนโลยีที่จะให้นักพัฒนาอย่างเราๆ สะดวกสบายในการพัฒนาอะไรสักอย่างหนึ่งขึ้นมา ซึ่ง git จะทำหน้าที่ในการจัดการเกี่ยวกับเวอร์ชันของซอร์สโค้ดของเรา ไม่ว่าจะเป็นเรื่องการตรวจสอบความแตกต่างระหว่างเวอร์ชัน (diff) การย้อนกลับมาเวอร์ชันก่อนหน้าเมื่อมีปัญหา (revert) หรือตรวจสอบเกี่ยวกับรายละเอียดของแต่ละเวอร์ชัน (log) แตกต่างจาก Github ตรงที่ Github เป็นเว็บไซต์ที่ให้บริการเสมือน git server เป็นบริการฟรีแบบมีเงื่อนไขสามารถใช้งานได้ฟรี แต่โปรเจ็กที่สร้างขึ้นจะต้องเป็นแบบ public เท่านั้น**

**17.จุดประสงค์หลักในการ branch คืออะไร**

**ตอบ ใช้เพื่อแยกการพัฒนาความสามารถใหม่ๆ ออกมาจากส่วนหลัก โดยที่ git มี master branch มาให้อยู่แล้ว หลังจากทำการสร้าง git repository ขึ้นมา ในการใช้ branch เพื่อแยกการพัฒนานั้น จะต้องทำการ merge การเปลี่ยนแปลงเข้ามายัง master branch หลังจากที่พัฒนาเสร็จแล้วเสมอ**

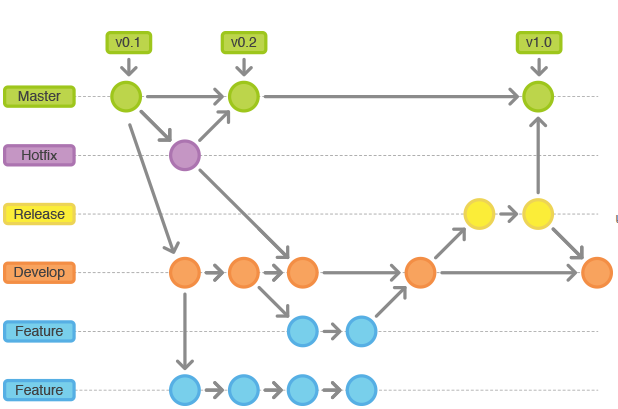
**18. Fast forward merge คืออะไรและทำไมการ push ไปที่ remote repo จึงควรต้อง merge แบบนี้**

**ตอบ Fast forward merge คือ การแยกออกมาแทนการสร้างใหม่ของ Git จะ update HEAD ของเราให้ตรงตาม HEAD ของ merged commit (โดยไม่มีการสร้าง commit object ใหม่ขึ้นมา) และต้อง push ไปที่ remote repo เพื่อดึง code จากต้นน้ำมา update code โดยที่ไม่มีการเปลี่ยนแปลงของเรา**

**19. หน้าที่หลักของคำสั่ง git pull คืออะไร**

**ตอบ คำสั่งที่รวมร่างกันระหว่าง git fetch กับ git merge นักพัฒนาบางท่านแนะนำให้ใช้ fetch แล้วตามด้วย git merge เพราะสามารถตรวจสอบก่อนการ Merge รวมได้ แต่เพื่อความง่ายเราก็จะใช้ git pull ดึงข้อมูลกันเป็นหลัก**

**20.แผนภาพด้านล่างนี้ต้องการสื่อความหมายอะไร**

****

**ตอบ 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 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**