11.โปรแกรม version control มีประโยชน์อย่างไร

จัดเก็บการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์หนึ่งหรือหลายไฟล์เพื่อที่จะสามารถเรียกเวอร์ชั่นใดเวอร์ชั่นหนึ่งกลับมาดูเมื่อไรก็ได้ ตัวอย่าง เช่น

นักออกแบบกราฟฟิคหรือเว็บดีไซเนอร์และต้องการเก็บทุกเวอร์ชั่นของรูปภาพหรือเลย์เอาต์ การใช้ Version Control System (VCS) เป็นสิ่งที่ชาญฉลาดมาก เพราะมันช่วยให้สามารถย้อนไฟล์บางไฟล์หรือแม้กระทั่งทั้งโปรเจคกลับไปเป็นเวอร์ชั่นเก่าได้ นอกจากนั้นระบบ VCS ยังจะช่วยให้เปรียบเทียบการแก้ไขที่เกิดขึ้นในอดีต ดูว่าใครเป็นคนแก้ไขคนสุดท้ายที่อาจทำให้เกิดปัญหา แก้ไขเมื่อไร และยังช่วยให้สามารถกู้คืนไฟล์ที่ลบหรือทำเสียโดยไม่ตั้งใจได้อย่างง่ายดาย

12.ข้อได้เปรียบของ distributed version control เมื่อเทียบกับ centralized version control คืออะไร

Distributed คือ VCS ที่ผู้ใช้แต่ละคนจะเก็บเวอร์ชั่นของไฟล์เอาไว้ในเครื่องของตัวเอง และสามารถที่จะ synchronize ความเปลี่ยนแปลงของไฟล์ในแต่ละเครื่องให้เหมือนกันได้

13.ข้อได้เปรียบของ centralized version control เมื่อเทียบกับ distributed version control คืออะไร

Centralized คือ VCS ที่มี server กลางคอยเก็บเวอร์ชั่นของไฟล์ทุกเวอร์ชั่น ผู้ใช้จะใช้ client ต่อเข้ามาเพื่อที่จะดึงโค๊ดไปใช้งานในเครื่องของตัวเอง

14.บอกแนวทางในการแก้ไข conflict ที่เกิดขึ้นเมื่อมีการ merge โปรแกรมของผู้พัฒนาหลาย ๆ คนเข้าด้วยกัน

ถ้าใช้ workflow แบบ feature branch ดังนั้น ในทุกๆ feature จำเป็นจะต้องสร้าง feature branch จาก develop branch อยู่เสมอ เมื่อทำการพัฒนา feature เรียบร้อยแล้ว ก็จะต้อง merge code กลับไปยัง develop branch แต่ก่อนที่จะทำการ merge code นั้น ควรที่ทำให้มั่นใจก่อนว่า codeใน feature branch

มันจะไม่เกิด conflict กับ code ใน develop branch

ดังนั้น ถ้าเกิด conflict ขึ้นมา ควรทำการแก้ไขใน feature branch ก่อนด้วยการ rebase จากนั้นจึงทำการ merge กลับไปยัง develop branch หลังจากนั้นลบ feature branch ทิ้ง

15.บอกแนวทางในการลด conflict ที่เกิดขึ้นจากการ merge โปรแกรมของผู้พัฒนาหลาย ๆ คนเข้าด้วยกัน

ปกติแล้ว git merge จะรวม code ให้เราเองอัตโนมัติ แต่ก็จะมีข้อยกเว้นเมื่อ แก้ไขไฟล์เดียวๆกัน กรณีที่เราและเพื่อนร่วมทีม แก้ไขไฟล์เดียวกัน Git จะเกิดการ conflict เมื่อเราจะ merge code โดยไม่รู้ว่าจะใช้ code ของเราหรือของเพื่อน วิธีแก้ก็คือ ทำการ edit แล้ว commit ไปใหม่

16.Git คืออะไร แตกต่างจาก Github อย่างไร

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

จัดเก็บข้อมูลของเป็นเสมือนภาพถ่าย(snapshot)ของระบบไฟล์ขนาดเล็กๆ ทุกครั้งที่มีการ commit หรือบันทึกสถานะของโปรเจคลงใน Git มันจะทำการถ่ายภาพของไฟล์ทั้งหมดในตอนนั้นและบันทึกการอ้างอิงไปยัง snapshot นั้น เพื่อให้การจัดเก็บนั้นมีประสิทธิภาพ ถ้าไฟล์ใดที่ไม่ได้มีการเปลี่ยนแปลง Git ก็จะไม่บันทึกไฟล์นั้นอีกครั้ง เพียงแต่จะทำการเชื่อมโยงไปยังไฟล์เดิมที่เคยถูกบันทึกเอาไว้อยู่แล้ว

แต่ Github คือ เว็บเซิฟเวอร์ที่ให้บริการในการฝากไฟล์ เมื่อเป็นสมัครสมาชิกแล้ว เราสามารถที่จะสร้าง Repository ของเราแล้วเอาไปฝากไว้บน Github (การทำงานทุกอย่างก็เหมือนทำ local) จุดเด่นของ Github คือใช้ฟรี และสร้าง repository ได้ไม่จำกัด แต่ต้องเป็น public repository เท่านั้น หากอยาก private ต้องเสียตัง

17.จุดประสงค์หลักในการ branch คืออะไร

การสร้างใหม่และย้ายไปทำงานบนนั้น ยกตัวอย่าง เช่น เรามี code ที่ดีอยู่แล้ว แต่อยากจะทดลองอะไรนิด ๆหน่อย หรือแก้ไขอะไรก็ตาม ไม่ให้กระทบกับตัวงานหลัก ก็เพียงแค่สร้าง branch ใหม่ขึ้นมา เมื่อแก้ไขหรือทำอะไรเสร็จแล้ว ก็ค่อยเซฟกลับมาที่ master เหมือนเดิม

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

"Fast forward" ใน merge นั้น เพราะ commit ที่ถูกชี้โดย branch ที่ merge มันเป็น upstream ของ commit ที่อยู่โดยตรง Git ก็เลยขยับ pointer ไปข้างหน้า พูดอีกนัยหนึ่งก็คือ เวลาที่พยายามจะ merge commit lydอันเข้ากับ commit ที่สามารถไปถึงได้โดยการตาม history ของ commit อันแรก Git จะทำให้ทุกอย่างง่ายขึ้นโดยการขยับ pointer ไปข้างหน้าเพราะมันไม่มีงานที่ถูกแยกออกไปให้ merge สิ่งนี้เรียกว่า "fast forward"

19.หน้าที่หลักของคำสั่ง git pull คือออะไร

git pull ก็คือรวมcodeจาก remote มายัง local โดยที่เราไม่สามารถรู้ได้เลยว่าจะรวม code อะไรบ้าง รู้แค่หลังจาก pull เสร็จแล้ว ซึ่งจริงๆแล้ว git pull มันก็คือการทำ git fetch และต่อด้วย git merge อัตโนมัติ

20.แผนภาพต้องการสื่อความหมายอะไร

ความสามารถในการแตก branch การแตก branch ก็เพื่อจะให้การเขียนโค้ดเป็นทีม บริหารจัดการง่าย สมมติ ในบริษัทมี 2 ทีม ประชุมกันแล้วว่าจะให้ 2 ทีมนี้ทำกันคนละฟีเจอร์ แตก branch ไปให้ทีมละ branch โค้ดก็จะได้ไปปะปนกัน ในขั้นตอนการพัฒนา (พัฒนาเสร็จค่อยเอามันมารวมกัน) git นั้นมันไม่ได้บอกในตัวมันเองหรอกว่า ต้องแตก branch

ในเมื่อมันไม่มีข้อกำหนดยังไง เลยมีคนเสนอแนวทางในการแตก branch ไว้หลายแนวทาง แนวทางเหล่านี้เขาเรียกกันว่า workflow มันคือแนวทางที่จะทำให้การทำงานมันราบรื่น แต่มันมีแนวทางแนวทางหนึ่งที่เป็นที่นิยม เรียกว่า 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. ทีนี้หลังจากที่เอาโค้ดขึ้นโปรดักชันแล้ว 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

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