11. โปรแกรม version control มีประโยชน์อย่างไร

-เป็นโปรแกรมที่สามารถจัดเก็บการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์หนึ่งหรือหลายไฟล์เพื่อที่จะสามารถเรียกเวอร์ชั่นใดเวอร์ชั่นหนึ่งกลับมาดูเมื่อไรก็ได้ เช่น ระบบ git

12. ข้อได้เปรียบของ distributed version control เมื่อเทียบกับ centralized version control คืออะไร

ข้อได้เปรียบของ distributed version control คือ ระบบ centralized version control มันมีจุดอ่อน ตรงที่การรวมศูนย์ทำให้มันเป็นจุดอ่อนจุดเดียวที่จะล่มได้เหมือนกันเพราะทุกอย่างรวมกันอยู่ที่เซิร์ฟเวอร์ที่เดียว ถ้าเซิร์ฟเวอร์นั้นล่มซักชั่วโมงนึง หมายความว่าในชั่วโมงนั้นไม่มีใครสามารถทำงานร่วมกันหรือบันทึกการเปลี่ยนแปลงงานที่กำลังทำอยู่ไปที่เซิร์ฟเวอร์ได้เลย หรือถ้าฮาร์ดดิสก์ของเซิร์ฟเวอร์เกิดเสียขึ้นมาและไม่มีการสำรองข้อมูลเอาไว้ก็จะสูญเสียข้อมูลประวัติและทุกอย่างที่มี จะเหลือก็แค่ก๊อปปี้ของงานบนเครื่องแต่ละเครื่องเท่านั้นเอง แต่ถ้าเป็นข้อได้เปรียบของ distributed version control คือ แต่ละคนไม่เพียงได้ก๊อปปี้ล่าสุดของไฟล์เท่านั้น แต่ได้ทั้งก๊อปปี้ของ repository เลย หมายความว่าถึงแม้ว่าเซิร์ฟเวอร์จะเสีย client ก็ยังสามารถทำงานร่วมกันได้ต่อไป และ repository เหล่านี้ของ client ยังสามารถถูกก๊อปปี้กลับไปที่เซิร์ฟเวอร์เพื่อกู้ข้อมูลกลับคืนก็ได้ การ checkout แต่ละครั้งคือการทำสำรองข้อมูลทั้งหมดแบบเต็ม ๆ นั่นเอง

13 .ข้อได้เปรียบของ centralized version control เมื่อเทียบกับ distributed version control คืออะไร

การทำงานแบบนี้มีประโยชน์เหนือ Distributed Version Control Systems (DVCSs) หรือระบบ VCS แบบกระจายศูนย์ ในหลายด้าน เช่น ทุกคนสามารถรู้ได้ว่าคนอื่นในโปรเจคกำลังทำอะไร ผู้ควบคุมระบบสามารถควบคุมได้อย่างละเอียดว่าใครสามารถแก้ไขอะไรได้บ้าง การจัดการแบบรวมศูนย์ในที่เดียวทำได้ง่ายกว่าการจัดการฐานข้อมูลใน client แต่ละเครื่องเยอะ

14. บอกแนวทางในการแก้ไข conflict ที่เกิดขึ้นเมื่อมีการ merge โปรแกรมของผู้พัฒนาหลายๆคนเข้าด้วยกัน

- ทำการ edit แล้ว commit ไปใหม่

15. บอกแนวทางในการลด conflict ที่เกิดขึ้นเมื่อมีการ merge โปรแกรมของผู้พัฒนาหลายๆคนเข้าด้วยกัน

โดยปกติแล้ว git merge จะรวมโค๊ดให้เราเองอัตโนมัติ การที่เราจะลดการ conflict นั้น ทั้งเราและเพื่อน ควรที่จะ ตรวจสอบโค้ด ในโปรเจค ตามหน้าที่ของแต่ละคนที่ได้รับมอบหมายงานมาทำว่า มีข้อผิดพลาดอะไรบ้าง ควรตรวจสอบให้แน่ชัดก่อนที่จะ commit ขึ้นไป จะเป็นการช่วย ลด การ conflict ไม่มากก็น้อย

16. Git คืออะไร แตกต่างจาก Github อย่างไร

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

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

สรุปโดยย่อ คือ Git เป็นระบบที่จัดเก็บการเปลี่ยนแปลงของไฟล์ในโปรเจ็คเราเราสามารถ เรียกดูแก้ไข หรือทำอะไรก็ตามในโปรเจ็คได้ ส่วน Github มันเป็นตัวที่ทำงานบนเว็ป มันก็คือ เว็ปฝากไฟล์ Git นั่นแหละ

17. จุดประสงค์หลักในการ branch คืออะไร

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

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

ในการ Merge Branch บน Git นั้น หาก Commit สุดท้ายของ Branch ปลายทาง เป็น Commit เดียวกับจุดที่แยก Branch ออกมา การ Merge จะได้ผลเป็นแบบ Fast-forward

การ Merge แบบ Fast-forward สายของ Commit เป็นเส้นตรง สวยงาม

19. หน้าที่หลักของคำสั่ง git pull คืออะไร

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

20. แผนภาพต้องการสื่อความหมายอะไร

ทำการจัดการ code แต่ละ branch แยกตาม feature ไป

เมื่อ developer ทำการพัฒนา และ ทดสอบเสร็จแล้ว

จะทำการ merge code จาก feature branch

ไปยัง integration branch

จากนั้นทำการทดสอบอีกครั้งบน integration branch

เมื่อทุกอย่างเรียบร้อย ก็ทำการ merge กลับไปยัง branch หลักต่อไป

แสดงดังรูป

ดังนั้นสถานะของ code บน branch หลัก

คือพร้อมที่จะ dpeloy/release อยู่ตลอดเวลา

อีกอย่างหนึ่งที่เห็นได้ชัดเจนก็คือ มีการทำงานแบบ manual เยอะมาก ๆ

ดังนั้น เราสามารถลดด้วยการทำระบบทำงานแบบอัตโนมัติเข้ามาช่วย

ทั้งการ merge การทดสอบ และ การ deploy ระบบงาน

ตัวอย่างที่นำวิธีการนี้ไปประยุกต์ใช้งาน เช่น Github flow

กล่าวคือ เมื่อ developer แต่ละคนทำการพัฒนา feature เสร็จแล้ว

จะไม่สามารถ merge code กลับไปยัง integration branch และ branch หลักได้

แต่จะต้องส่ง Pull Request ไปยัง integration branch

ซึ่งการจัดการ Pull Request ส่วนใหญ่จะเป็นแบบ manual

เนื่องจากต้องทำการ review และ ทดสอบ code

เมื่อผ่านทั้งหมดแล้ว จะทำการ merge ไปยัง branch หลักต่อไป

ข้อควรระวัง

ยิ่ง feature branch มีอายุ หรือ การพัฒนาที่ยาวนานเพียงใด

การดูแลรักษาก็ยากมากขึ้นไปเท่านั้น

ชื่อของ feature branch ควรดูดี มีสาระ !!

นั่นคือ ต้องมีชื่อตรงตามความต้องการของระบบ

ไม่ใช่อยากจะตั้งชื่ออะไรก็ตั้ง

แถมยังต้องมีการจัดการ branch ต่าง ๆ อยู่เป็นประจำ

เช่นการลบ branch ต่าง ๆ ที่ทำการ merge ไปยัง branch หลักต่อไปซะ

ไม่เช่นนั้น จะมี branch อยู่มากมายก่ายกอง

ซึ่งยากต่อการจัดการเป็นอย่างมาก