11.โปรแกรม version control มีประโยชน์อย่างไร

- Backup and Restore = ต้องการย้อนกลับไปแก้ไขงานเดิม

- Synchronization = ช่วยให้ทำงานพร้อมกันได้

- Short-term undo = ย้อนกลับไปแก้ไขงานเล็กๆได้

- Long-term undo = ย้อนกลับไปแก้ไขงานใหญ่ๆได้

- Track Changes = รู้ว่างานเปลี่ยนแปลงตอนไหน

- Track Ownership = บอกว่าใครเป็นคนทำส่วนไหน

- Sandboxing

- Branching and Merging = การแยกงานและการรวมงานกลับมา

12.ข้อได้เปรียบของ distributed version control

-มันทำงานแบบออฟไลน์ได้

- ทุกคนสามารถเปลี่ยนแปลงงานในเครื่องคอมของตนเองก่อนที่จะย้ายลง repo กลางได้

- มันทำงานได้เร็ว

- การทำ Branching (แยกงานไปทำคนละสาขา) หรือ Merging (การเอางานมารวมกัน) สามารถทำได้ง่าย

13.ข้อได้เปรียบของ centralized version control

- ทุกคนสามารถรู้ได้ว่าคนอื่นในโปรเจคกำลังทำอะไร

- ผู้ควบคุมระบบสามารถควบคุมได้อย่างละเอียดว่าใครสามารถแก้ไขอะไรได้บ้าง

- การจัดการแบบรวมศูนย์ในที่เดียวทำได้ง่ายกว่าการจัดการฐานข้อมูลใน client แต่ละเครื่องเยอะ

- มีลำดับในการเปลี่ยนแปลงงานที่ชัดเจน

14.

- เปิดไฟล์ที่ทำให้เกิด conflict

- ส่วนที่เกิด conflict จะถูกคั่นด้วยบรรทัดที่ขึ้นต้นด้วยเครื่องหมาย <<<<<<< , ======= , >>>>>>>

- ลบส่วนที่เกิด conflict ทิ้งไป

- จัดการเปิดอ่านโปรแกรมแล้วจัดการเรียบเรียง code ใหม่ให้ทำงานได้ถูกต้อง

15.วิธีการลด conflict

- พยายามทำการ Merge บ่อยๆ เนื่องจากว่ายิ่งใช้เวลานานเพียงใด งานก็จะถูกแก้ไขจากงานเดิมมากเท่านั้น ความเสี่ยงก็ยิ่งมากขึ้นทำให้เกิดข้อขัดแย้งขนาดใหญ่ และ มากมาย ดังนั้นควรจะทำการ Merge ทันทีหลังจาก commit source code

- การทำงานเล็กๆ คือ ในหนึ่ง class หรือ หนึ่ง mathod ควรมีเหตุผลเดียวในการเปลี่ยนแปลงเท่านั้น จะช่วยทำให้นักพัฒนาไม่ทำงาน หรือ เปลี่ยนแปลง source code ที่เดียวกันอย่างแน่นอน ยกเว้นจะทำงานเดียวกัน หรือ ทำงานด้วยกัน

- การพูดคุยสื่อสารกันเป็นสิ่งสำคัญมาก ถ้าหากเราพูดคุยกันในทีมว่าจะแก้ไขไฟล์ไหน แก้ส่วนใด เพื่อนในทีมไม่แก้งานส่วนนั้น หรือพูดคุยกันว่าจะแก้งานออกมาในรูปแบบไหน ก็จะทำให้ไม่เกิด conflict

- Mob programming เป็นวิธีการที่ทรงประสิทธิภาพอย่างมาก และเชื่อได้เลยว่า แก้ไขปัญหา Merge conflict ได้ 100% เนื่องจากทุกคนมานั่งทำงานด้วยกัน ใช้เครื่องทำงานเดียวกัน ดังนั้น ไม่มีทางที่ source code จะขัดแย้งกัน แต่มันมีค่าใช้จ่ายที่สูงมาก ๆ ดังนั้นจึงเป็นวิธีการสุดท้ายที่จะทำนั่นเอง

16. Git แตกต่างจาก Github อย่างไร

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

GitHub คือ เว็บที่ให้บริการพื้นที่จัดเก็บโครงการโอเพ่นซอร์สด้วยระบบควบคุมเวอร์ชันแบบ Git โดยมีจุดประสงค์หลักคือ ทำให้การแบ่งปันและพัฒนาโครงการต่างๆด้วยกันเป็นไปได้ง่ายๆ

สรุปก็คือ Git คือ ระบบที่ช่วยควบคุมลำดับและขั้นตอนของการพัฒนา code เพื่อช่วยให้ผู้ใช้ทราบถึงการเปลี่ยนแปลงระหว่าง code ดั่งเดิม กับ code ที่พัฒนาขึ้นมา ถ้าไม่มี Github การจะทำงานร่วมกันกับ Git จะต้องพิมพ์ผ่าน command หรือ Terminal เท่านั้น ดังนั้น Github จึงถูกพัฒนาขึ้นมาเพื่อทำให้การทำงานกับ Git สะดวกและง่ายดายมากขึ้น Github คือ application ทำงานร่วมกันกับ Git โดยมี interface ที่ทำใช้งานง่าย

17. Branch มีจุดประสงค์เพื่อแยกการพัฒนาความสามารถใหม่ๆ ออกมาจากส่วนหลัก โดยที่ git มี master branch มาให้อยู่แล้ว หลังจากทำการสร้าง git repository ขึ้นมา การแยกงานกันออกมาทำจะช่วยทำให้ใช้เวลาในการพัฒนาน้อยลง และงานมีประสิทธิภาพเพราะจะแยกงานกันทำตามความถนัดของผู้ใช้



18. Fast Forword Merges คือ Merge แบบเร็วๆ ใช้ในกรณีที่เรา branch ไปแล้ว เมื่อจะ Merge กับมาใน Main พบว่า main ยังคงเหมือนเดิม ไม่มีการเปลี่ยนแปลง source code จากเพื่อนร่วมทีม จึงนำ branch มาเป็น head แทน โดยไม่ต้องทำการ diff เพื่อดูความแตกต่างของ source code

เหตุผลที่การ push ไปที่ remote repo จึงควรต้อง merge แบบนี้เพราะ เป็นการ merge ที่จะไม่มีวันทำให้เกิด conflict หรือไม่มีความขัดแย้งกันของ source code ขึ้น

19. หน้าที่หลักของ git pull คือดึง source code ที่มีการเปลี่ยนแปลงล่าสุดจาก remote repository มายัง local repository โดย git จะทำการ ดึงข้อมูล และ รวมข้อมูล การเปลี่ยนแปลงจาก remote repository ในเครื่องของคุณ

20.แผนภาพนี้ต้องการสื่อความหมายถึงการแยกงานกันไปพัฒนาความสามารถใหม่ๆ ออกมาจากส่วนหลัก หรือที่เรียกว่าการ "Branch" โดยมีการแยกงานไปพัฒนาในช่วง v0.1 โดยพัฒนาในส่วนของ Develop และ Feature ใหม่ๆ ซึ่งในขณะนั้น Master ก็มีการพัฒนาจนออก v0.2 เมื่อพัฒนาไปช่วงหนึ่งก็มีการ merge งานกลับมาที่ mester หรือ repo กลางในช่วง v1.0 ถือเป็นจุดเด่นของการใช้งาน git ในรูปแบบ distributed version control