**11. ตอบ** มีเซิร์ฟเวอร์กลางที่เก็บไฟล์ทั้งหมดไว้ในที่เดียวและผู้ใช้หลาย ๆ คนสามารถต่อเข้ามาเพื่อดึงไฟล์จากศูนย์กลางนี้ไปแก้ไขได้ ทุกคนสามารถรู้ได้ว่าคนอื่นในโปรเจคกำลังทำอะไร ผู้ควบคุมระบบสามารถควบคุมได้อย่างละเอียดว่าใครสามารถแก้ไขอะไรได้บ้าง การจัดการแบบรวมศูนย์ในที่เดียวทำได้ง่ายกว่าการจัดการฐานข้อมูลใน client แต่ละเครื่อง นอกจากนั้นระบบเหล่านี้ยังทำงานกับหลาย ๆ repository ได้อย่างดี ทำให้คุณสามารถทำงานกับคนหลายกลุ่มซึ่งทำงานในรูปแบบต่างกันในโปรเจคเดียวกันได้อย่างง่ายดาย เนื่องจากระบบเหล่านี้สนับสนุนการทำงานได้หลากหลายรูปแบบ ซึ่งอาจทำได้ยากในระบบแบบรวมศูนย์

----------------------------------------------------------------------------------------------------------------------

**12. ตอบ** distributed version control เร็วกว่า และผู้ใช้งานก็สามารถแก้ได้ทุกที่ โดยไม่จำเป็นจะต้องติดต่อกับ central repo ก็ได้

----------------------------------------------------------------------------------------------------------------------

**13. ตอบ** มี Server หลักเป็นศูนย์กลางในการเก็บข้อมูลของโปรเจคทั้งหมด Client แต่ละเครื่องจะเป็นแค่ Working Copy เท่านั้น วิธีทำงานคร่าวๆ เริ่มต้นเราก็สร้างโปรเจคขึ้นมาบน Server หลังจากนั้นแต่ละคนก็มาดาวน์โหลดลงเครื่องของตัวเอง เวลามีการเปลี่ยนแปลงอะไรก็อัพโหลดกลับขึ้นไปบน Server แล้วคนอื่นๆ ก็จะสามารถดาวน์โหลด Update นั้นมายังเครื่องของตัวเองได้

----------------------------------------------------------------------------------------------------------------------

**14. ตอบ** ให้ทำการ merge บ่อย ๆ นั่นคือ ทุกครั้งเมื่อมีการเปลี่ยนแปลง หรือ commit source code จะช่วยลดข้อขัดแย้งต่าง ๆ ลงไปอย่างมากถึงจะเกิดข้อขัดแย้ง ก็เป็นเพียงปัญหาเล็ก ๆซึ่งสามารถแก้ไขได้อย่างง่ายดาย

----------------------------------------------------------------------------------------------------------------------

**15. ตอบ** ให้สมาชิกในทีมมีการพูดคุยกันเกิดขึ้น ปรึกษาหารือกัน ให้มีการทำงานต่างๆไปในแนวทางเดียวกัน ให้สมาชิกเข้าใจในสิ่งต่างๆให้ตรงกัน

----------------------------------------------------------------------------------------------------------------------

**16. ตอบ** Git คือ Version Control ตัวหนึ่ง ซึ่งเป็นระบบที่มีหน้าที่ในการจัดเก็บการเปลี่ยนแปลงของไฟล์ในโปรเจ็คเรา มีการ backup code ให้เรา สามารถที่จะเรียกดูหรือย้อนกลับไปดูเวอร์ชั่นต่างๆของโปรเจ็คที่ใด เวลาใดก็ได้ หรือแม้แต่ดูว่าไฟล์นั้นๆใครเป็นคนเพิ่มหรือแก้ไข หรือว่าจะดูว่าไฟล์นั้นๆถูกเขียนโดยใครบ้างก็สามารถทำได้ แตกต่างจาก Github คือ Github เป็นเพียงเว็บเซิฟเวอร์ที่ให้บริการในการฝากไฟล์ Git เท่านั้น

----------------------------------------------------------------------------------------------------------------------

**17. ตอบ** เพื่อแยกการพัฒนาความสามารถใหม่ๆ ออกมาจากส่วนหลัก และ จะต้องทำการ merge การเปลี่ยนแปลงเข้ามายัง master branch หลังจากที่พัฒนาเสร็จแล้วเสมอ

----------------------------------------------------------------------------------------------------------------------

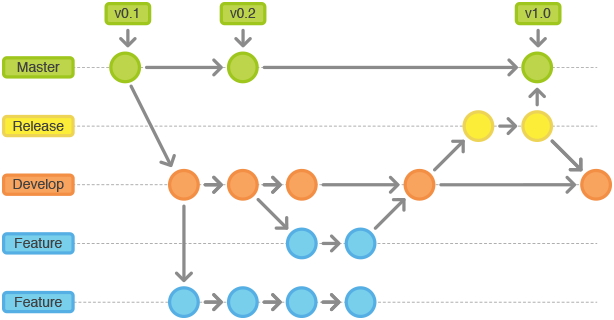
**18. ตอบ** "Fast forward" ใน merge นั้น เพราะ commit ที่ถูกชี้โดย branch ที่คุณ merge มันเป็น upstream ของ commit ที่คุณอยู่โดยตรง Git ก็เลยขยับ pointer ไปข้างหน้า พูดอีกนัยหนึ่งก็คือ เวลาที่คุณพยายามจะ merge commit ซักอันเข้ากับ commit ที่สามารถไปถึงได้โดยการตาม history ของ commit อันแรก Git จะทำให้ทุกอย่างง่ายขึ้นโดยการขยับ pointer ไปข้างหน้าเพราะมันไม่มีงานที่ถูกแยกออกไปให้ merge สิ่งนี้เรียกว่า "fast forward".

----------------------------------------------------------------------------------------------------------------------

**19. ตอบ** ดึงข้อมูล และ รวมข้อมูล การเปลี่ยนแปลงจาก remote repository ในเครื่องของคุณ

----------------------------------------------------------------------------------------------------------------------

### 20. ตอบ Release Branches



Once develop has acquired enough features for a release (or a predetermined release date is approaching), you fork a release branch off of develop.

Creating this branch starts the next release cycle, so no new features can be added after this point-only bug fixes, documentation generation, and other release-oriented tasks should go in this branch.

Once it’s ready to ship, the release gets merged into master and tagged with a version number. In addition, it should be merged back intodevelop, which may have progressed since the release was initiated.

Using a dedicated branch to prepare releases makes it possible for one team to polish the current release while another team continues working on features for the next release. It also creates well-defined phases of development (e.g., it’s easy to say, “this week we’re preparing for version 4.0” and to actually see it in the structure of the repository).

Best Practices:

* May branch off: develop
* Must merge back into: develop and master
* Tag: increment major or minor number
* Branch naming convention: release-\* or release/\*