Version control และ git

11.

ช่วยให้คุณสามารถย้อนไฟล์บางไฟล์หรือแม้กระทั่งทั้งโปรเจคกลับไปเป็นเวอร์ชั่นเก่าได้ นอกจากนั้นระบบ VCS ยังจะช่วยให้คุณเปรียบเทียบการแก้ไขที่เกิดขึ้นในอดีต ดูว่าใครเป็นคนแก้ไขคนสุดท้ายที่อาจทำให้เกิดปัญหา แก้ไขเมื่อไร ฯลฯ และยังช่วยให้คุณสามารถกู้คืนไฟล์ที่คุณลบหรือทำเสียโดยไม่ตั้งใจได้อย่างง่ายดาย

12.

The act of cloning an entire repository gives distributed version control tools several advantages over centralized systems:

* Performing actions other than pushing and pulling changesets is *extremely* fast because the tool only needs to access the hard drive, not a remote server.
* Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once.
* Everything but pushing and pulling can be done without an internet connection. So you can work on a plane, and you won't be forced to commit several bugfixes as one big changeset.
* Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone.

13.

To be quite honest, there are almost no disadvantages to using a distributed version control system over a centralized one. Distributed systems do *not* prevent you from having a single “central” repository, they just provide more options on top of that.

There are only two major inherent disadvantages to using a distributed system:

* If your project contains many large, binary files that cannot be easily compressed, the space needed to store all versions of these files can accumulate quickly.
* If your project has a very long history (50,000 changesets or more), downloading the entire history can take an impractical amount of time and disk space.

14.

1.ให้ทำการ merge บ่อย ๆ

2.**การออกแบบที่ดีมันช่วยให้ทีมทำงานร่วมกันได้อย่างดี** ยิ่งแต่ละส่วนการทำงานเล็ก ๆ แล้ว ก็ยิ่งทำให้คุณภาพของการออกแบบระบบดี

3.การพูดคุย สื่อสาร นั้นสำคัญ เราควรมีการสื่อสารที่ชัดเจนหากไม่เข้าใจจริงควรทำงานที่เดียวกันเลยเพื่อลดปัญหาที่เกิดจากการสื่อสาร

4. [Mob programming](http://mobprogramming.org/" \t "_blank)

15.

รูปแบบของ workflow ในการใช้ git นั้นมีหลากหลายเนื่องจากตัวมันยืดหยุ่นมากๆ แต่ทีมพัฒนานั้น จำเป็นต้องตัดสินใจเลือก workflow ว่าจะไปในทิศทางไหน เพื่อลด conflict ต่างๆ เพื่อลดความสับสนต่างๆ เพื่อทำให้ทีมใช้งาน git ได้อย่างเต็มความสามารถโดย workflow ที่ดี จะทำให้ทีมมี productivity ที่สูงขึ้น

รูปแบบของ workflow ที่ได้รับความนิยมสูง คือ **[Git-Flow](http://nvie.com/posts/a-successful-git-branching-model/" \t "_blank)**

16.

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

แตกต่างจาก Github คือ Github เป็นเว็บไซต์ที่ให้บริการเสมือน git server เป็นบริการฟรีแบบมีเงื่อนไข คุณสามารถใช้งานได้ฟรี แต่โปรเจ็กที่สร้างขึ้นจะต้องเป็นแบบ public เท่านั้น หากต้องการให้เป็น private จำเป็นต้องเสียค่าบริการ

17.

Branch นั้นถูกใช้เพื่อแยกการพัฒนาความสามารถใหม่ๆ ออกมาจากส่วนหลัก โดยที่ git มี master branch มาให้อยู่แล้ว หลังจากทำการสร้าง git repository ขึ้นมา

18.

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

ทำไมจึงควร merge แบบนี้ เพราะ จะเกิดเป็น Commit ใหม่แล้ว ทำให้เราสามารถเห็น ทางเข้า-ทางออก ของ Branch test ได้อย่างชัดเจน

19.

ใช้ดึงความเปลี่ยนแปลงจาก remote มายัง local และรวมเข้าด้วยกัน (มีค่าเท่ากับfetch+merge)

20.

Gitflow คือ การแตก branch ไว้หลายแนวทาง  แนวทางเหล่านี้เขาเรียกกันว่า workflow มันคือแนวทางที่จะทำให้การทำงานมันราบรื่น

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