**การทดลองที่ 2**

**Introduction to team work with repository**

|  |
| --- |
| **กล่าวนำ** |

1.1 ทบทวนความรู้จากครั้งที่แล้ว

ในระบบการทำงานของ git ที่เครื่องของนักพัฒนานั้น จะมีพื้นที่เก็บไฟล์ 3 ส่วนด้วยกันคือ Working directory, staging area และ Local repository

ไฟล์ต่างๆ ที่เป็นโปรเจคของเรา จะอยู่ใน working directory สามารถเรียกดูได้โดยการใช้คำสั่ง ls, dir หรือเรียกดูได้โดยการใช้ file browser

ส่วนของ Local repository จะอยู่ใน directory ที่ชื่อว่า .git และถูกซ่อนไว้จากการเรียกดูไฟล์โดย วิธีการปกติ แต่อาจเรียกดูได้จากคำสั่ง ls -al หรือการแก้ไข properties ให้ file browser สามารถมองเห็นไฟล์ที่ซ่อนไว้ เนื้อหาในส่วนนี้ git จะมีวิธีการบริหารจัดการในลักษณะฐานข้อมูล และไม่สามารถใช้วิธีการปกติในการเปิดอ่านไฟล์ได้

ส่วนที่เหลืออีกอย่างหนึ่งคือ staging เป็นส่วนที่ระบบ git ใช้เปรียบเทียบความแตกต่าง ระหว่างไฟล์ใน working dorectory กับข้อมูลที่เก็บใน database เพื่อติดตามการเปลี่ยนแปลงของไฟล์ทั้งหมดที่ถูก track เราสามารถตรวจสอบสถานะได้จากการใช้คำสั่ง git status



**รูปที่ 1** การทำงานในส่วนต่างๆ ของ git บนเครื่อง local computer

1.2 สิ่งที่ควรรู้จักก่อนทำงานเป็นทีมบน git

|  |
| --- |
| **git branch** |

Git branch เป็นคำสั่งที่ทำหน้าที่ แสดง สร้าง หรือลบ branch ออกจาก repository โดยมีคำสั่งย่อยๆ ดังต่อไปนี้

|  |  |
| --- | --- |
| รูปแบบคำสั่ง | การทำงาน |
| git branch | แสดง branch ทั้งหมด |
| git branch <branch> | สร้าง branch ใหม่ที่มีชื่อว่า <branch>, โดยอ้างอิงจุดในอดีตตาม branch ปัจจุบัน |
| git branch <branch> <start-point> | สร้าง branch ใหม่ที่มีชื่อว่า <branch> โดยอ้างอิงจุดเริ่มต้นจาก <start-point> ที่ระบุในคำสั่ง ซึ่งอาจจะเป็น branch name หรือ tag name. |
| git branch -d <branch> | ลบ branch ที่ชื่อว่า <branch> |
| git branch -D <branch> | ลบ branch ที่ชื่อว่า <branch> โดยไม่สนใจสถานะการ merged |
| git checkout <branch> | ย้าย branch มาอยู่ที่ <branch> พร้อมทั้งเปลี่ยนแปลงทุกอย่างที่อยู่ใน working directory ให้เป็นไปตาม <branch> ใน repository |
| git checkout -b <new> <start-point> | สร้าง branch ใหม่ที่ชื่อว่า <new> โดยอ้างอิงจุดเริ่มต้นเป็น <start-point> จากนั้นก็ทำการ checkout |

|  |
| --- |
| git merge |

Git merge เป็นคำสั่งที่ใช้รวมประวัติการพัฒนาตั้งแต่สองจุดขึ้นไปเข้าด้วยกัน เนื่องจากโดยปกติแล้ว การพัฒนาโครงการต่างๆ บน git มักจะถูก commit ตามเวลา ดังนั้น ในการที่เราจะพัฒนาโครงการ ไม่ว่าจะเป็นการเพิ่ม feature ต่างๆ หรือการแก้ไขบัก (bug fix) ก็ไม่ควรที่จะไปรบกวน source code บนเส้นทางหลักของการพัฒนาโปรแกรม เนื่องจากในขณะที่เรายังพัฒนา feature หรือแก้ไขบักไม่เสร็จ อาจจะมีใครมา clone โครงการที่วางบนเส้นทางหลัก ดังนั้น เราควรสร้าง branch สำหรับพัฒนาหรือเพิ่ม fetures ต่างๆ ขึ้นมา จากนั้นเมื่อพัฒนาหรือแก้ไขบักสำเร็จแล้ว จึงค่อยนำ code ใน branch กลับไปรวม (merge) เข้ากับโปรแกรมในเส้นทางหลัก

**แนวคิดในการทำ branch และ merge**

การสร้าง branch ขึ้นมาเพื่อพัฒนา feature ใหม่ๆ (เช่น feature-1 ) จะช่วยให้เราสามารถพัฒนาได้อย่างอิสระ โดยที่ไม่ต้องกังวลว่าการ commit code จะไปรบกวนการทำงานของ code ใน master จากรูปที่ 2 จะพบว่าเราสามารถพัฒนา master ควบคู่กับการพัฒนา feature-1 ได้



**รูปที่** 2 การพัฒนาโดยการแยก branch ออกมา

เมื่อใดที่ทำการพัฒนา feature-1 เสร็จเรียบร้อยแล้ว เราสามารถนำกลับไปรวมกับ master ได้ทุกเวลา ดังรูปที่ 3



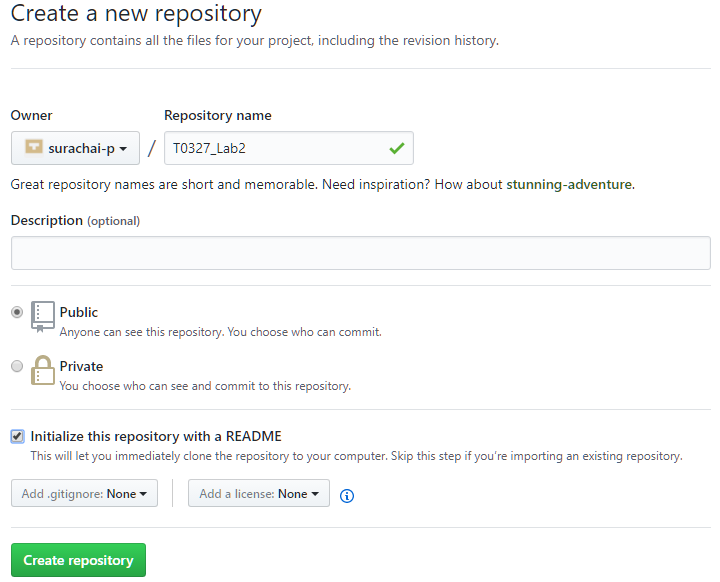
**รูปที่** 3 การรวม feature เข้ากับ master

จากรูปที่ 3 พบว่าการ merge เกิดขึ้นที่ commit D โดยที่มี commit C เป็น parrent แรก และ G เป็น parrent ที่สอง และต้องมั่นใจว่า branch ปัจจุบันเป็น branch ที่เราต้องการ merge เอา branch อื่นเข้ามา

ในการทำงานร่วมกันบน repository มีสถานการณ์ (scenario) ที่สามารถเกิดขึ้นได้มากมาย มีวิธีการแก้ไข สถานการณ์ต่างๆ เหล่านั้น ซึ่งสามารถศึกษาเพิ่มเติมได้จากเอกสารอ้างอิงต่างๆ

|  |
| --- |
| **2. ขั้นตอนการทดลอง** |

1. จับคู่การทดลอง ให้นักศึกษาคนแรก คือ S1 คนที่สองคือ S2
2. นักศึกษา **S1** สร้าง Repository ชื่อ Sxxxxxxxx\_Lab2 (xxxxxxxx คือรหัสนักศึกษาของตนเอง) ตัวอย่างดังรูปที่ 4



**รูปที่** 4 การสร้าง repo ใหม่

1. **S1** เพิ่ม Member จำนวน 3 คนดังนี้
   1. นักศึกษาที่เป็นคู่ปฏิบัติการทดลอง (S2)
   2. ผศ.โกศล ตราชู (koson trachu)
   3. ผศ.สุระชัย พิมพ์สาลี (surachai-p)

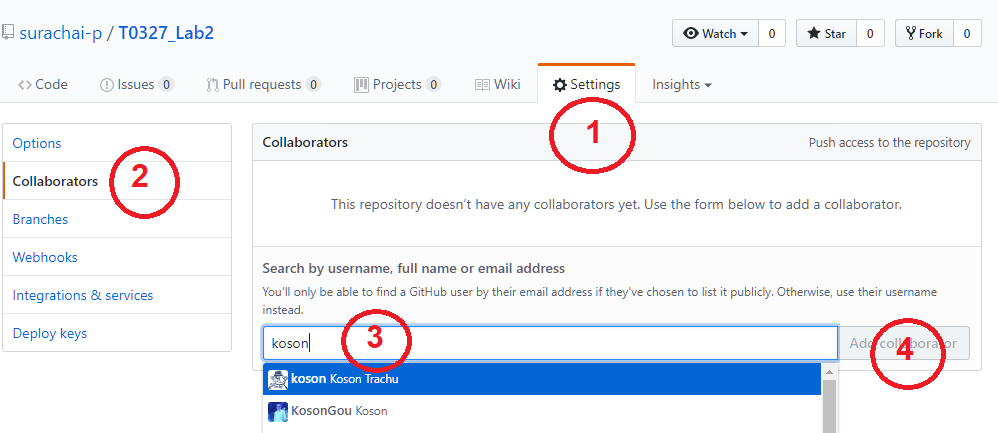
โดยมีขั้นตอนการ invite สมาชิกเข้าร่วมการพัฒนา ดังนี้

3.1 เลือกเมนู Settings

3.2 เลือกเมนู Collaborators

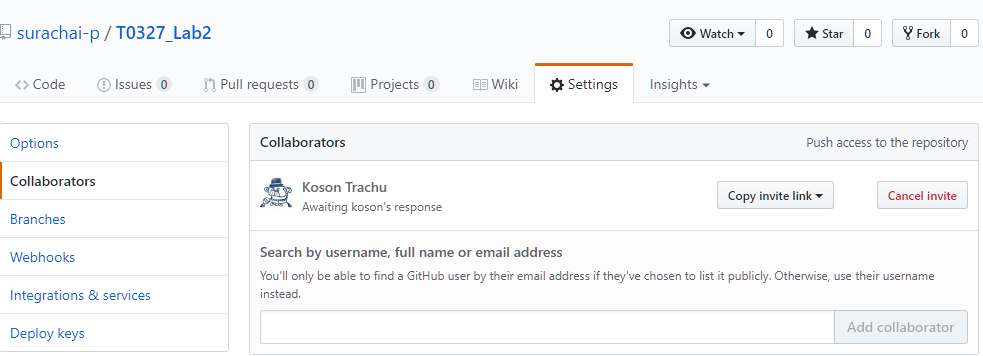
3.3 พิมพ์ username, full name หรืออีเมล์ของสมาชิกที่ต้องการเชิญเข้าร่วมการพัฒนา

3.4 กดปุ่ม Add collaborator



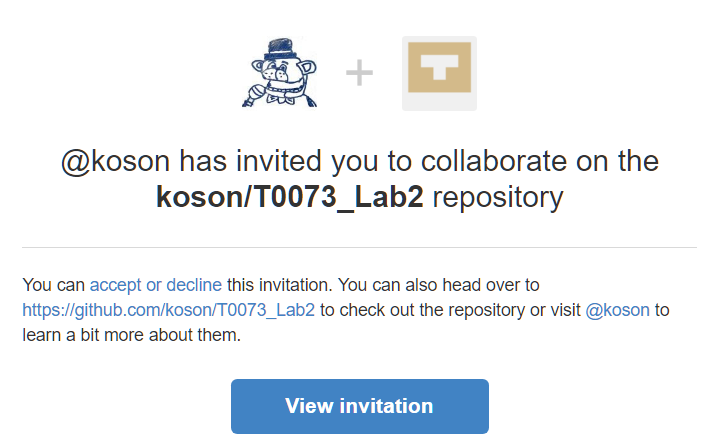
**รูปที่** 5 การเพิ่มผู้พัฒนาร่วมกันเข้าใช้ repo

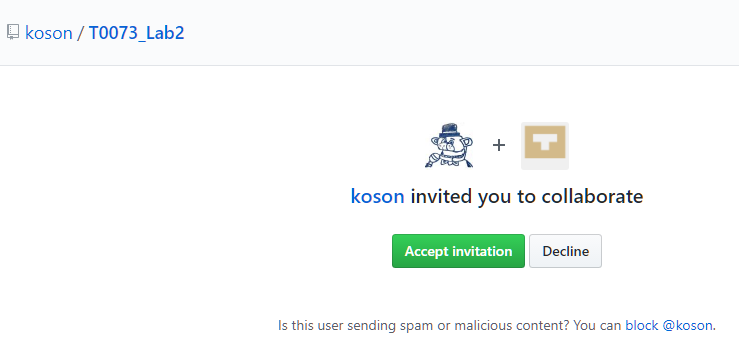
หลังจากนั้นจะปรากฎผลลัพธ์ดังรูปที่ 6



**รูปที่ 6** หน้าต่างแสดงรายชื่อผู้พัฒนาร่วมที่ถูกเชิญ

4. หลังจากนั้นจะมีอีเมล์ส่งไปยังผู้ที่ถูกเชิญเข้าร่วมพัฒนา ให้ผู้ได้รับการเชิญ กดรับการเชิญที่ Accept Invitation ดังรูปที่ 7

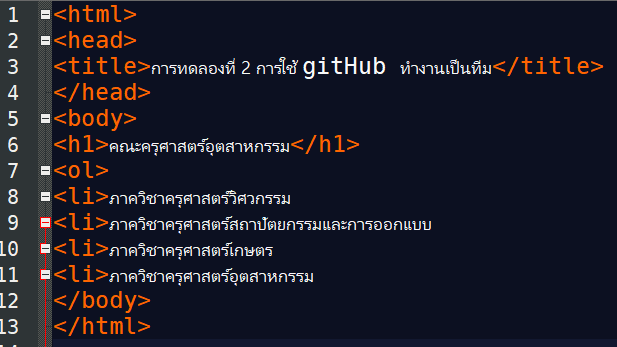




**รูปที่ 7** การตอบรับการเชิญเข้าร่วมใช้งาน repo

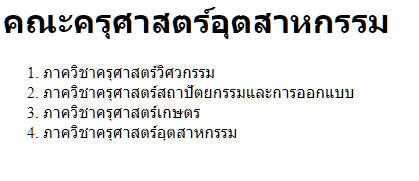
5. **S1** ทำการ clone repository ของตนเอง (ศึกษาคำสั่ง clone จากการทดลองที่ 1)

6. **S1** เปิด Notepad เพื่อสร้างไฟล์ชื่อ ietech.html ซึ่งมีข้อมูลดังนี้ (รูปที่ี 8 ใช้ Notepad++ เป็น editor)



**รูปที่ 8** โค้ดไฟล์ ietech.html

6. **S1** บันทึกไฟล์ไว้ที่ไดเรคทอรี่ ที่เก็บ repository ในเครื่อง (ใช้คำสั่ง pwd ใน git bash เพื่อดูตำแหน่ง folder) หลังจากนั้นทดสอบทำการทำงาน โดยเปิดไฟล์ ietech.html ด้วยเว็บบราวเซอร์ (chrome,firefox,ie) จะได้ผลลัพธ์ดังรูปที่ 9

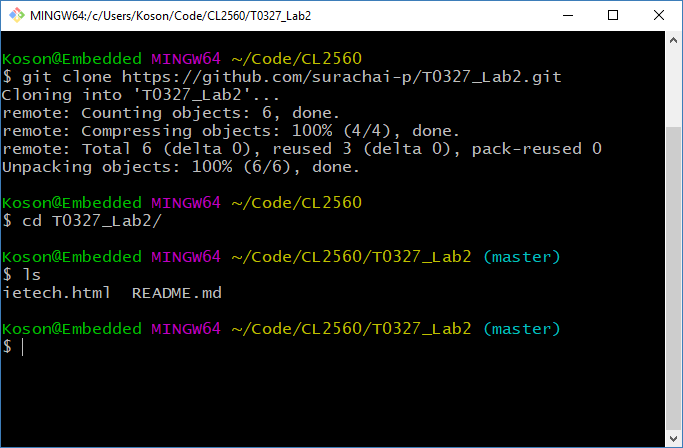


**รูปที่ 9** ผลลัพธ์การรันไฟล์ ietech.html ด้วยเว็บบราวเซอร์

7. **S1** ทำการ Add ไฟล์เข้าไปยัง repository ทำการ commit และ push ข้อมูลไว้บน gitHub (ดูวิธีการจากการทดลองที่ 1)

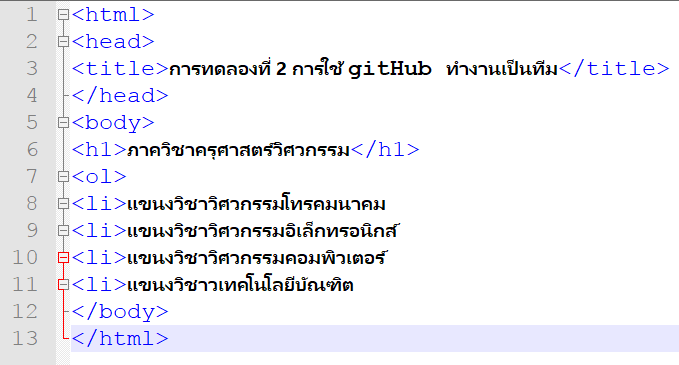
8. **S2** เปิดโปรแกรม git bash แล้วทำการ clone repository ที่ต้องการร่วมพัฒนากับคู่ปฏิบัติการทดลอง (repository ของ S1) ด้วยคำสั่ง git clone (ดูการใช้งานคำสั่ง git clone จากกการทดลองที่ 1)

9. เปลี่ยนไดเรคทอรี่ไปยัง directory ที่ทำการ clone ข้อมูลมา หลังจากนั้นให้ทำการตรวจสอบ repository จะเห็นไฟล์ต่างๆ ดังรูปที่ 10

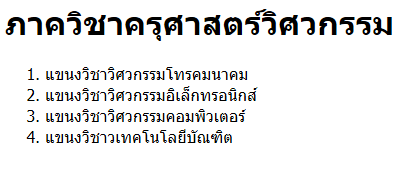


**รูปที่ 10** ผลลัพธ์การ Clone repo

10. **S2** สร้างหน้าเพจที่ต้องการเพิ่มเติมใน web site ในที่นี้ จะทำการเพิ่มหน้าเพจภาควิชาครุศาสตร์วิศวกรรม ตั้งชื่อไฟล์ engedu.html มี code และหน้าเพจดังรูปที่ 11 และ 12 ตามลำดับ **(มีจุดที่พิมพ์ผิด ให้พิมพ์ผิด ตามรูปที่กำหนด)**



**รูปที่ 11** โค้ดไฟล์ engedu.html



**รูปที่ 12** ผลลัพธ์การรันไฟล์ engedu.html ด้วยเว็บบราวเซอร์

11. **S2** ทำการแก้ไขไฟล์ ietech.html โดยการสร้าง Link ในตัวเลือกภาควิชาครุศาสตร์วิศวกรรม ดังรูปที่ 13



**รูปที่ 13** การแก้ไขโค้ดในไฟล์ ietech.html

12. **S2** ทดสอบเรียกหน้าเพจโดยการเรียกใน working directory จะพบว่ามีลิงค์ไปยังหน้าเพจเพิ่มขึ้นมาดังรูป 14



**รูปที่ 14** ผลลัพธ์การรันไฟล์ engedu.html ด้วยเว็บบราวเซอร์

13. **S2** เรียก git status พบว่า มีไฟล์ที่ถูกเปลี่ยนแปลงคือ ietech.html ดังรูปที่ 15



**รูปที่ 15** การตรวจสอบสถานะการเปลี่ยนแปลงของไฟล์ใน working directory

14. **S2** เพิ่มไฟล์ที่ถูกเปลี่ยนแปลง เข้าสู่พื้นที่ staging ของ git แล้ว commit ไปยัง Local repository

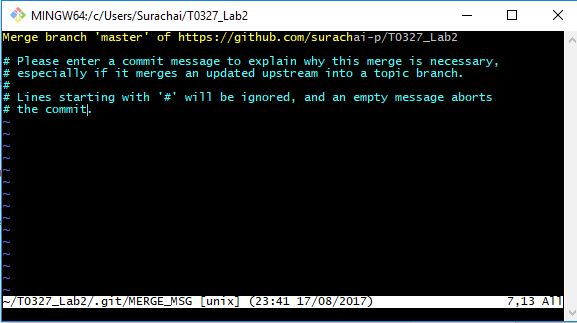


**รูปที่ 16** สถานะการเปลี่ยนแปลงของไฟล์ใน working directory หลังใช้คำสั่ง commit

15. **S2** ทำการ push local repository ขึ้นไปบน server

16. **S1** แก้ไขไฟล์ ietech.html บน Local repository โดยแก้ไขชื่อคณะให้ถูกต้อง และบันทึกไฟล์ดังกล่าว

17. **S1** ทำการ pull repository เพื่อให้ Local repository ข้อมูลเป็นปัจจุบันเนื่องจาก S2 ได้ทำการ push ข้อมูลจาก Local repository ไปที่ gitHub แล้ว ทำการตรวจสอบข้อมูลที่อยู่บน gitHub เปลี่ยนแปลงเป็นข้อมูลปัจจุบันหรือไม่ ซึ่งขั้นตอนการ pull อาจจะมีผลลัพธ์แสดง ดังรูปที่ 17



**รูปที่ 17** ผลการทำงานคำสั่ง git pull

18. **S1** ทำการ push local repository ขึ้นไปบน server

19. **S2** ทำการแก้ไขไฟล์ ietech.html ใน Local repository โดยสร้าง Link ในชื่อคณะฯ ให้ Link ไปยังหน้าเว็บไซต์ของคณะ ดังรูปที่ 18



**รูปที่ 18** โค้ดการแก้ไขเพื่อสร้าง Linkg ไปยังเว็บไซต์คณะฯ

20. **S2** ทำการ commit และ push ไฟล์ขึ้นไปยัง gitHub

21. ตรวจสอบผลการทำงานที่ได้ว่าถูกต้องหรือไม่

|  |
| --- |
| **การทำงานเป็นทีมโดยการ fork และ pull request** |

22. เปิด web browser ไปที่ repo ที่ต้องการ fork จากนั้นกดที่ fork ดังรูปที่ 19



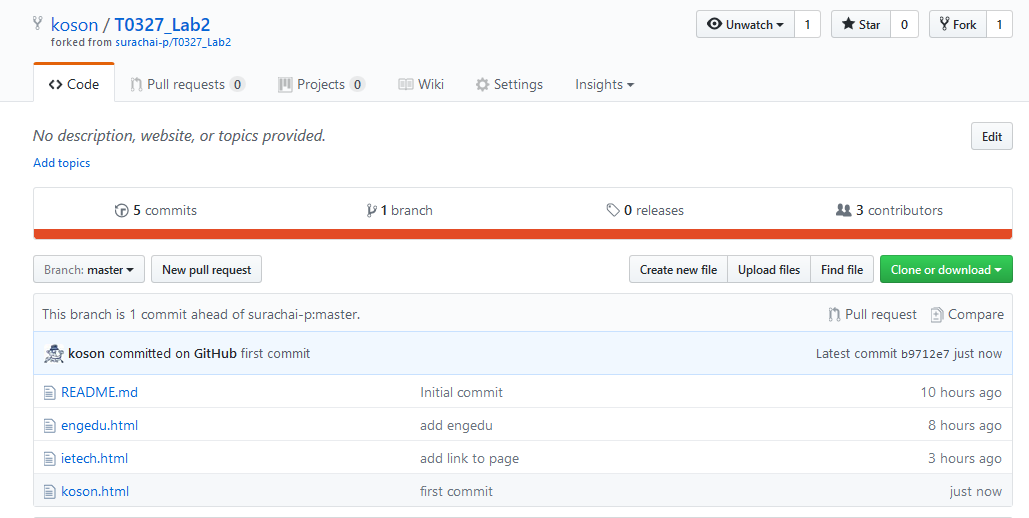
**รูปที่ 19** การ fork repository

23. เมื่อ fork มาได้แล้ว จะพบว่าเราได้ repository ใหม่ในบัญชีของเราเองโดยที่ github จะบอกว่า fork มาจาก repository ใด ดังรูปที่ 20



**รูปที่ 20** ผลลัพธ์การ fork repository

24. clone repo มาไว้ที่เครื่อง แก้ไข แล้ว push ตามขั้นตอนปกติ ในที่นี้ ได้เพิ่มไฟล์ชื่อ koson.html ดังรูปที่ 21



**รูปที่ 21** ผลการเพิ่มไฟล์ koson.html

25. หากเราต้องการส่งการเปลี่ยนแปลงใดๆ กลับเข้าไปแก้ยัง repo ที่เรา fork มา (นิยมเรียกว่า repo ต้นน้ำ) เราต้องขออนุญาตส่งไฟล์เข้าร่วมใน repo นั้น โดยการคลิกที่ “New pull request” ดังรูปที่ 22



**รูปที่ 22** การขอ Pull request

26. Git จะแสดงหน้าจอเพื่อที่จะให้เราตรวจสอบความถูกต้อง และขอทำ pull request ไปยังเจ้าของ repo ต้นน้ำ ดังรูปที่ 23



**รูปที่ 23** การตรวสอบความถูกต้องของการ pull request

27. ให้กดปุ่ม Create pull request สีเขียว ดังรูปด้านบน git จะให้กรอกข้อความบางอย่าง ดังรูปด้านล่าง ให้กรอกข้อความหรือเหตุผลที่ขอทำ pull request ไปยังเจ้าของ repo ดังรูปที่ 24



**รูปที่ 24** การระบุเหตุผลในการทำ pull request

28. เมื่อเราทำการคลิกที่ปุ่ม pull request แล้ว จะปรากฏหน้าจอสรุปผลการทำ pull request ดังรูปที่ 25



**รูปที่ 25** ผลการทำ pull request

29. การทำ pull request เป็นการร้องขอให้เจ้าของ repo ดึงการเปลี่ยนแปลงของเราเข้าไปร่วมใน master ซึ่งระบบ github จะมีการส่งข่าวสาร (เช่นอีเมล์) แจ้งเตือนไปยังเจ้าของ repo ดังรูปที่ 26

****

**รูปที่ 26** การแจ้งเตือนไปยังเจ้าของ repo

30. เจ้าของ repo ต้นน้ำ พิจารณาความเข้ากันได้ของโค้ด พิจารณาว่า code สามารถรันได้หรือไม่ ส่วนกรณี conflict ที่อาจเกิดขึ้นใน code นั้น git ก็จะช่วยตรวจสอบเบื้องต้น หากไม่มีสิ่งใดผิดปกติ ก็สามารถตอบรับคำร้องขอโดยการคลิกที่ “Merge pull request” (และกด Confirm merge ในหน้าจอถัดไป)

|  |
| --- |
| **แบบฝึกหัด** |

**การสร้างเว็บไซต์ประวัตินักศึกษา ชั้นปีที่ ๒ แขนงวิศวกรรมคอมพิวเตอร์**

1. ให้หัวหน้าห้อง สร้าง repository สำหรับการพัฒนาเว็บไซต์ของชั้นเรียน โดยเริ่มต้น ให้มีเพียงชื่อสถาบันฯ ชื่อคณะ ชื่อภาควิชา ชื่อสาขาวิชา

2. ให้หัวหน้าห้องเพิ่มทีมดูแล repo เป็นจำนวน 3-4 คน เพื่อช่วยงานด้านต่างๆ

3. ให้นักศึกษาในชั้นเรียนทุกคน ทำการ Fork repository เข้ามายัง account ของตนเอง

4. ให้นักศึกษาในชั้นเรียนทุกคน ทำการ แก้ไขโฮมเพจ ให้มีลิ้งค์ไปยัง เว็บเพจของตนเอง

5. ให้นักศึกษาในชั้นเรียนทุกคน ทำการ สร้างเวบเพจของตนเอง ซึ่งอาจจะมีกราฟฟิก รุปภาพ ต่างๆ ได้ตามต้องการ

6. ให้นักศึกษาในชั้นเรียนทุกคน ทำการ pull request เพื่ิอขอเพิ่มโครงการของตนเองเข้าไปยัง repo สำหรับการพัฒนาเว็บไซต์ของชั้นเรียน

7. ให้ทีมดูแล repo พิจารณารับการร้องขอ

8. ให้หัวหน้าชั้นนำส่ง repo สำหรับการพัฒนาเว็บไซต์ของชั้นเรียนบน google classroom

**เอกสารอ้างอิง / ศึกษาเพิ่มเติม** [[1]](#footnote-0) [[2]](#footnote-1) [[3]](#footnote-2) [[4]](#footnote-3)

1. "Git Best Practises - The Pencil Code." <http://pencil-code.nordita.org/doc/git-best-practises.pdf>. Accessed 17 Aug. 2017. [↑](#footnote-ref-0)
2. "Using Git for Centralized and Distributed Version Control ... - CISL." <https://www2.cisl.ucar.edu/sites/default/files/2016%20March%2011%20-%20Git%20Training.pdf>. Accessed 17 Aug. 2017. [↑](#footnote-ref-1)
3. "git merge - CISL." <https://www2.cisl.ucar.edu/sites/default/files/2016%20March%2025%20-%20Git%20Training.pdf>. Accessed 17 Aug. 2017. [↑](#footnote-ref-2)
4. "Using Git for Centralized and Distributed Version Control ... - CISL." 1 Apr. 2016, <https://www2.cisl.ucar.edu/sites/default/files/2016%20April%201%20-%20Git%20Training.pdf>. Accessed 17 Aug. 2017. [↑](#footnote-ref-3)