

เรียนรู้การใช้งาน Git & GitHub

สำหรับผู้เริ่มต้น



ຮູ້ຈັກກຳບາຣຈັດເກີບເວອຣ໌ຊັ້ນ (Version Control)



Version Control คืออะไร

Version Control หรือ Source Control หมายถึง เครื่องมือช่วยติดตาม การเปลี่ยนแปลงของ Source Code โดยการเก็บประวัติการเปลี่ยนแปลง ลงฐานข้อมูลชนิดพิเศษ ซึ่งจุดประสงค์ของการเก็บบันทึกทุกการเปลี่ยนแปลง ก็คือ หากมีความผิดพลาดเกิดขึ้น ไม่ว่าจะผิดพลาดเล็กน้อย ไปจนถึงขั้น ร้ายแรงที่จะส่งผลให้ซอฟแวร์ที่พัฒนาขึ้นมาล้มเหลว หรือเสียหาย



Version Control คืออะไร

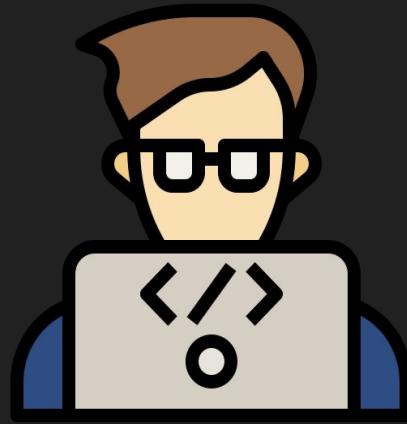
การบันทึกประวัติเก็บไว้อย่างต่อเนื่อง จะช่วยทำให้นักพัฒนาซอฟต์แวร์สามารถย้อนกลับไปในช่วงเวลาต่างๆ โดยเปรียบเทียบโค้ดใหม่กับโค้ดเดิมซึ่งก่อนหน้าได้ และตรวจสอบความผิดพลาดเพื่อแก้ไขให้มีผลกระทบต่อมุ่งหมายที่สุด คล้ายๆ กับการ Undo / Redo ในโปรแกรม (แต่ถ้าหากปิดโปรแกรมไปก็จะไม่สามารถ Undo / Redo อีกรอบได้)

BREAK!



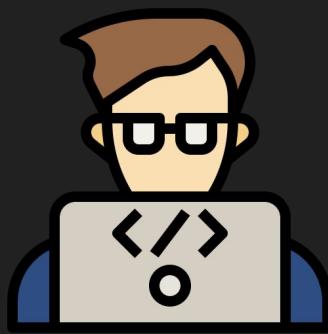
สาเหตุที่ต้องใช้ Version Control

สาเหตุที่ต้องใช้ Version Control



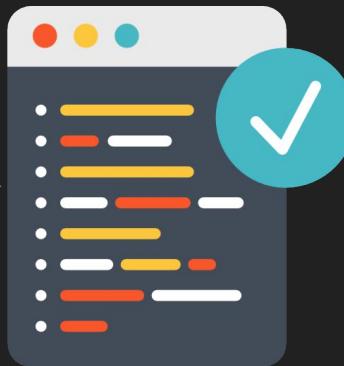
เด็กชายก้อง

สาเหตุที่ต้องใช้ Version Control

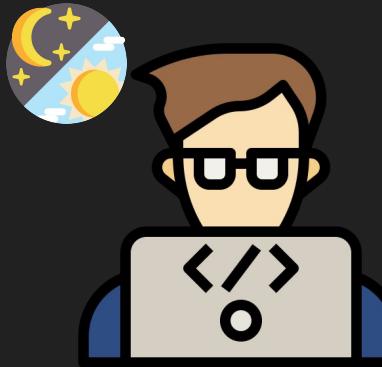


เด็กชายก้อง

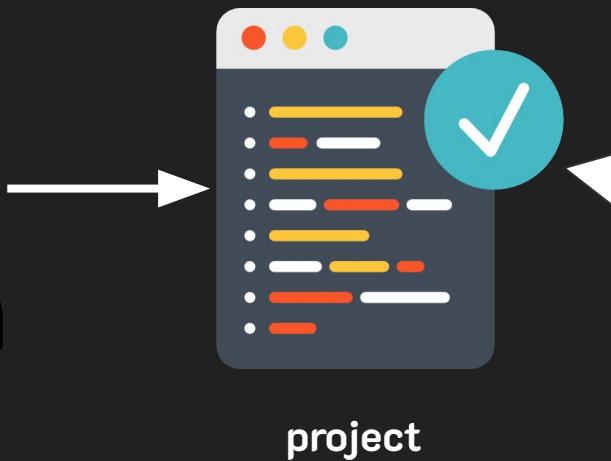
เขียนโค๊ด



สาเหตุที่ต้องใช้ Version Control



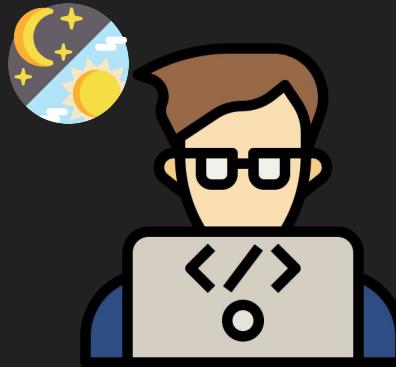
เด็กขายก้อน



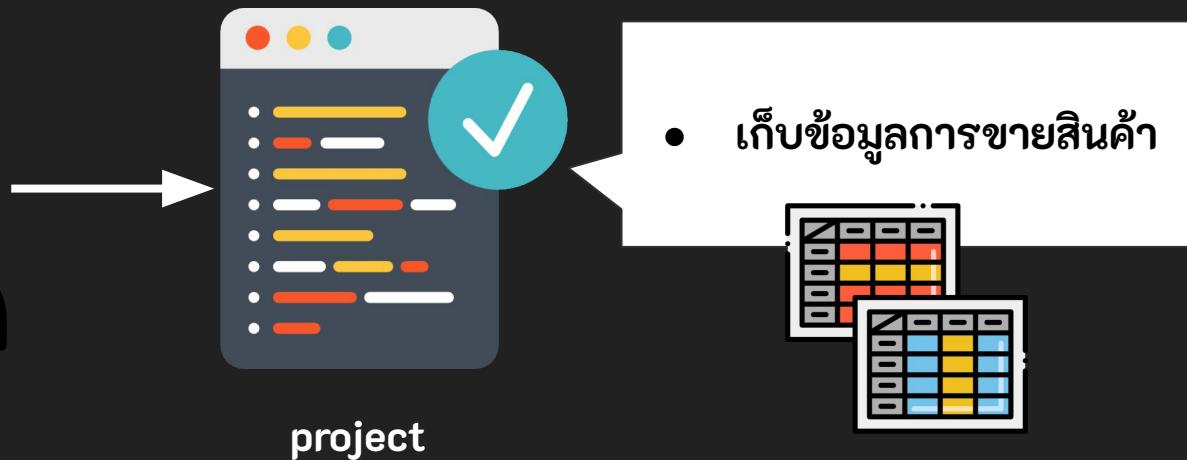
project

- เก็บข้อมูลการขายสินค้า

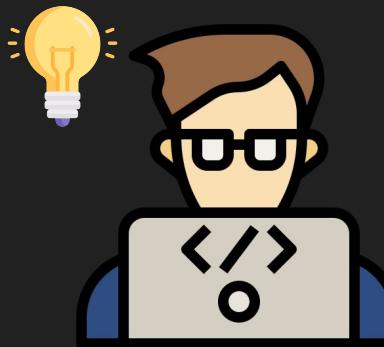
สาเหตุที่ต้องใช้ Version Control



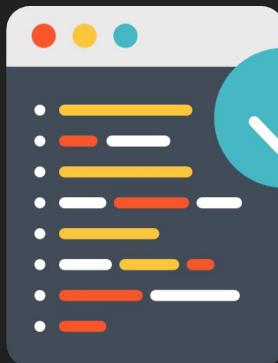
เด็กขายก้อน



สาเหตุที่ต้องใช้ Version Control



เด็กขายก้อน



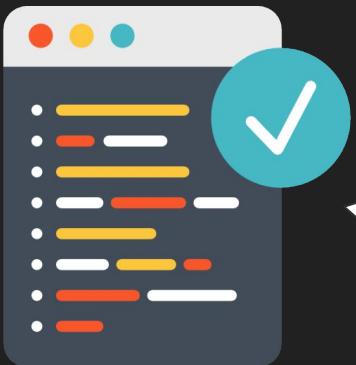
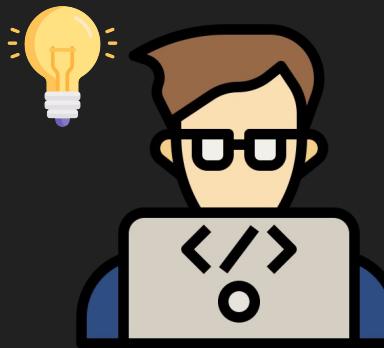
project

- เก็บข้อมูลการขายสินค้า
- **รายงานยอดขายแต่ละเดือน





สาเหตุที่ต้องใช้ Version Control

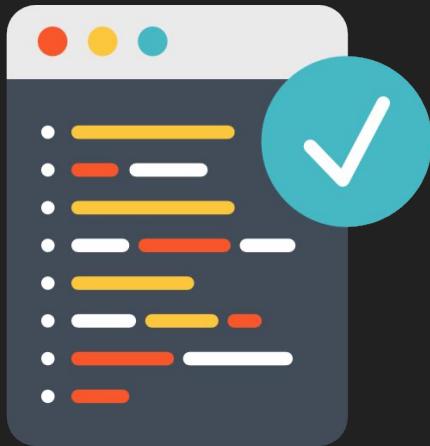


project

เด็กชายก่อ

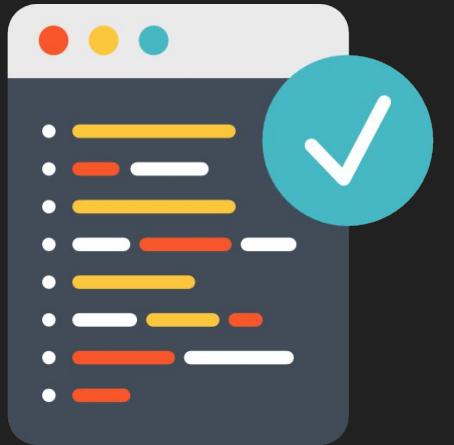
ทำยังไงจึงจะสามารถพัฒนา
โปรเจกต์ใหม่ / เพิ่มฟีเจอร์เข้าไป
ในโปรเจกต์เก่าและใช้งานระบบใน
เวอร์ชันเก่าได้ด้วย ?

สาเหตุที่ต้องใช้ Version Control



1. Backup (สำรอง) โค้ดที่เขียนเพื่อให้สามารถลับมาแก้ไขได้ในภายหลังได้

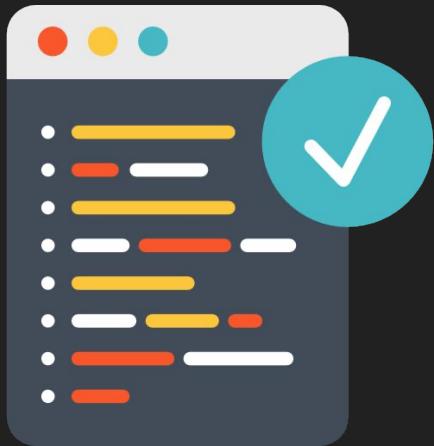
สาเหตุที่ต้องใช้ Version Control



2. คัดลอกไฟล์โค๊ดเก่าและสร้าง
ไฟล์ใหม่ทุกครั้งที่มีการแก้ไขโค๊ด



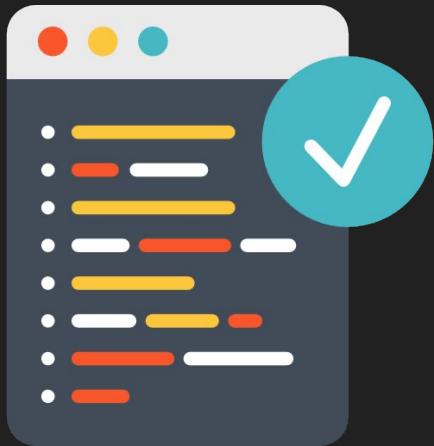
สาเหตุที่ต้องใช้ Version Control



- project

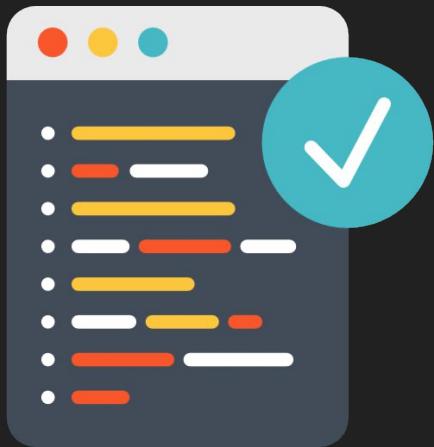


สาเหตุที่ต้องใช้ Version Control



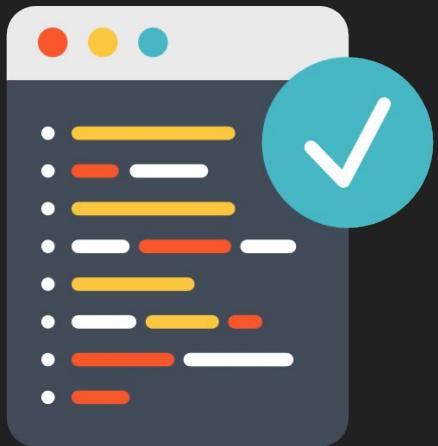
- project
- projectv2

สาเหตุที่ต้องใช้ Version Control



- project
- projectv2
- projectv3 fixbug

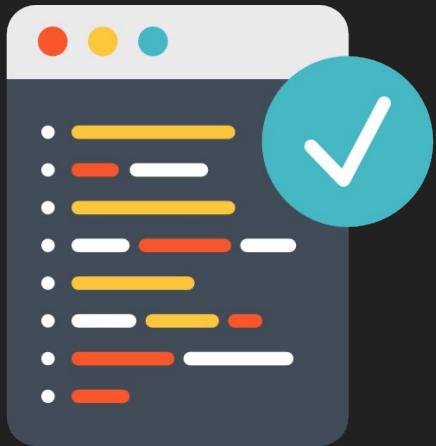
สาเหตุที่ต้องใช้ Version Control



** เก็บโค๊ดเวอร์ชั่นเก่าไว้เพื่อเวอร์ชั่นใหม่
ที่กำลังพัฒนานั้นเกิดพังขึ้นมา จะได้กลับ
ไปใช้ โค๊ดเวอร์ชั่นเก่าได้นั่นเอง



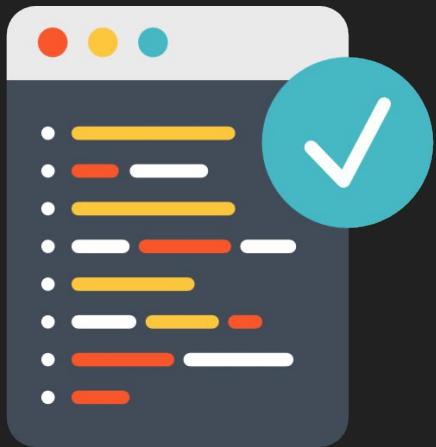
สาเหตุที่ต้องใช้ Version Control



- project
- projectv2
- projectv3 fixbug
- projectv4 add features



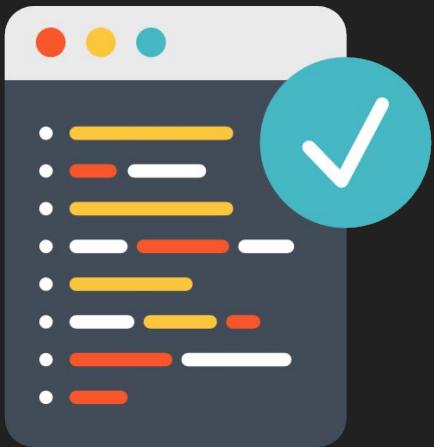
สาเหตุที่ต้องใช้ Version Control



- project
- projectv2
- projectv3 fixbug
- projectv4 add feature
- projectv5 final

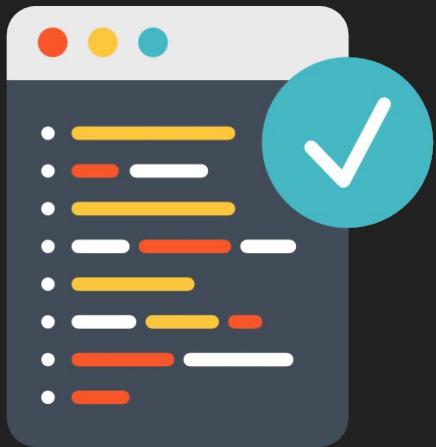


สาเหตุที่ต้องใช้ Version Control



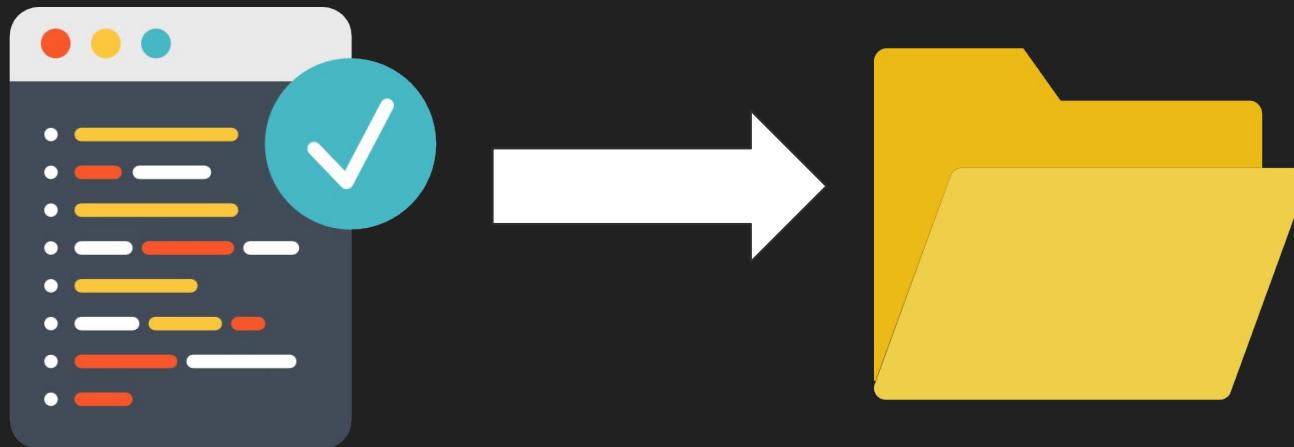
- project
- projectv2
- projectv3 fixbug
- projectv4 add feature
- projectv5 final
- projectv5 final fixbug

สาเหตุที่ต้องใช้ Version Control

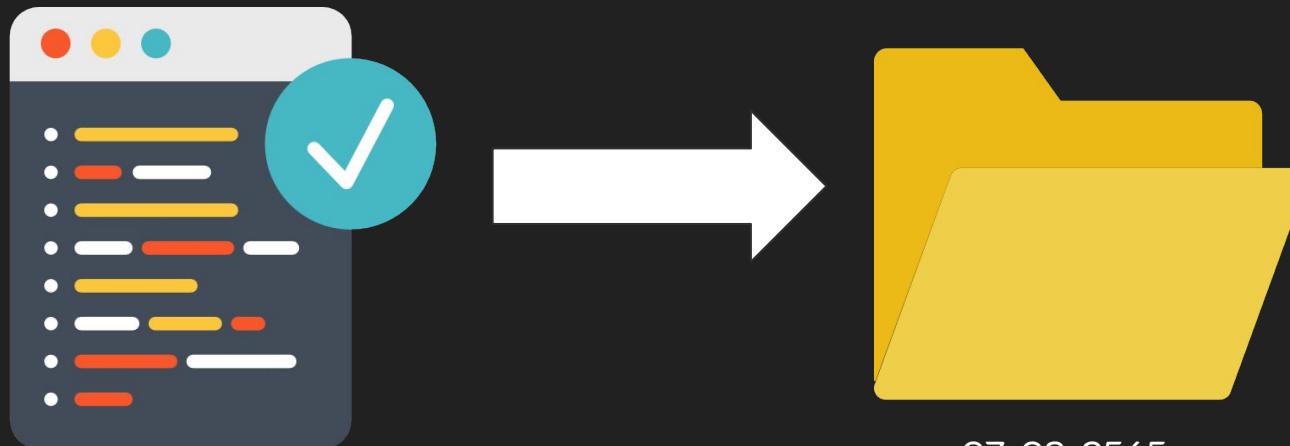


ถ้าเป็นโครงการใหญ่
มีความซับซ้อน มีไฟล์เยอะๆ ล่ะ ?

สาเหตุที่ต้องใช้ Version Control



สาเหตุที่ต้องใช้ Version Control



07-03-2565



สาเหตุที่ต้องใช้ Version Control



07-03-2565



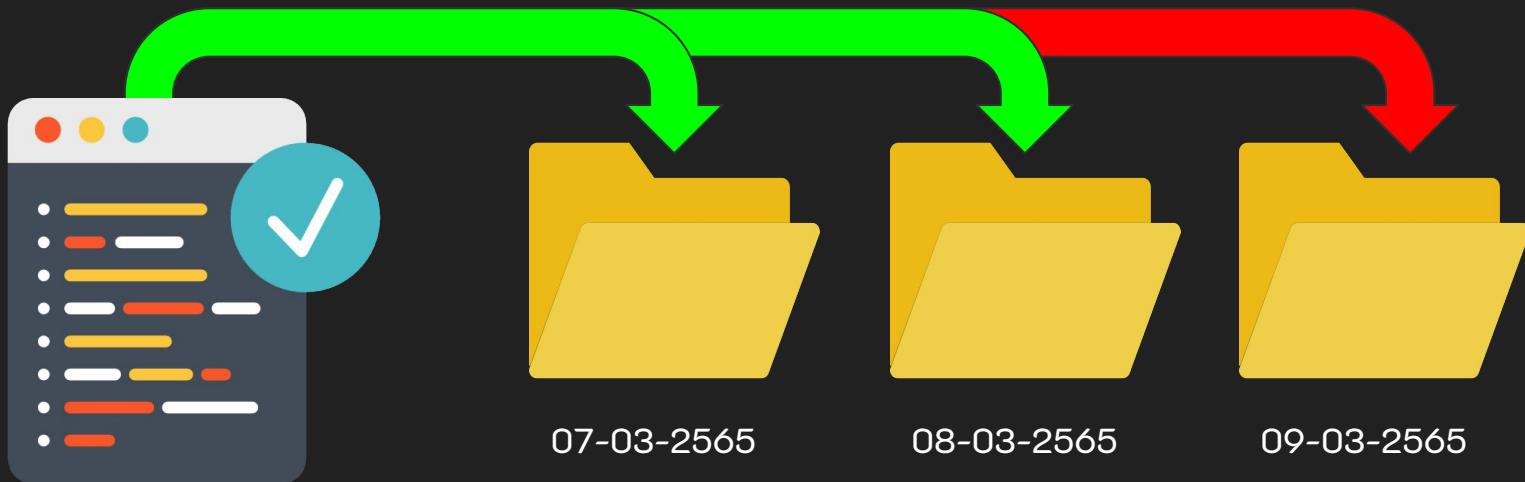
08-03-2565



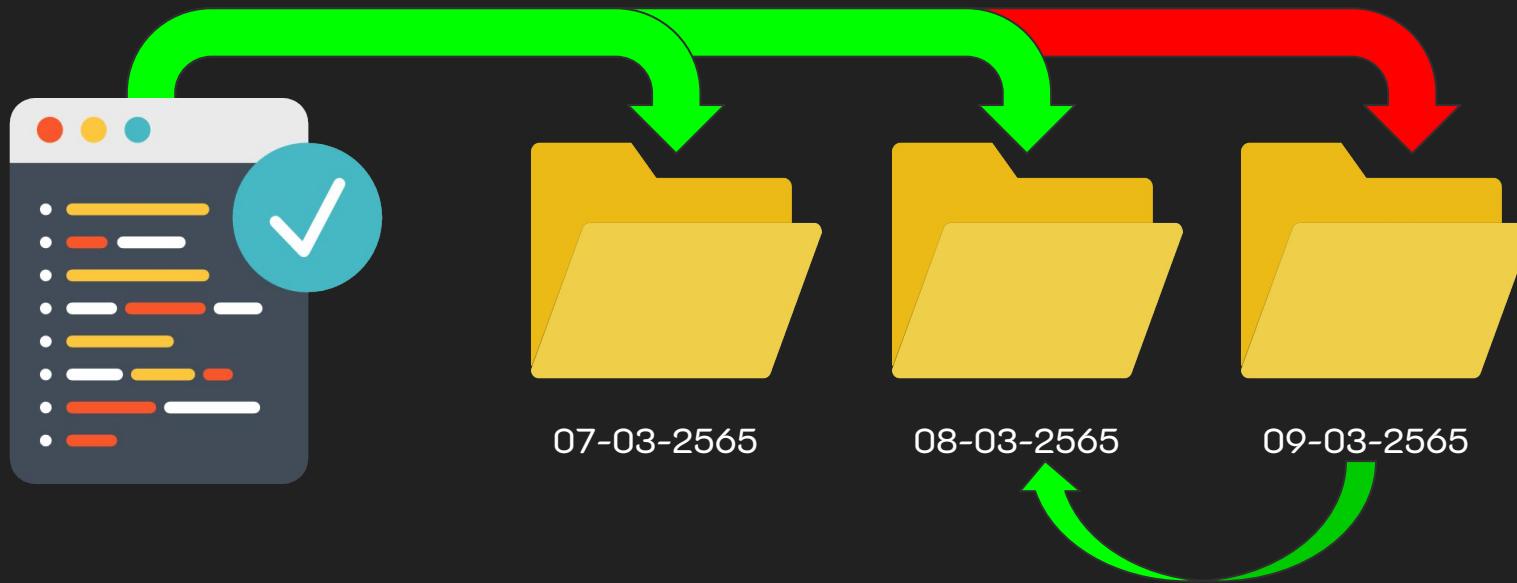
09-03-2565



สาเหตุที่ต้องใช้ Version Control



สาเหตุที่ต้องใช้ Version Control

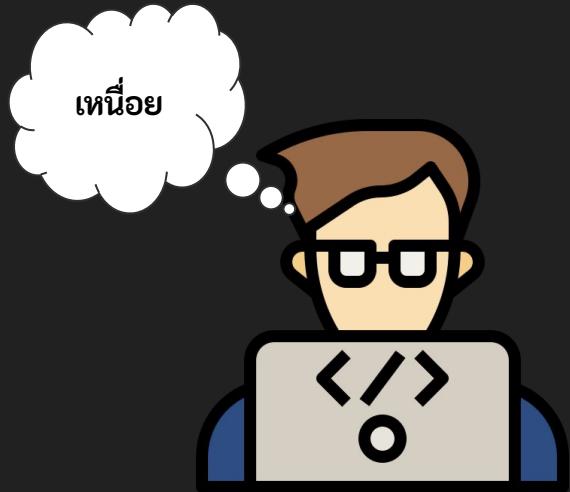


สาเหตุที่ต้องใช้ Version Control





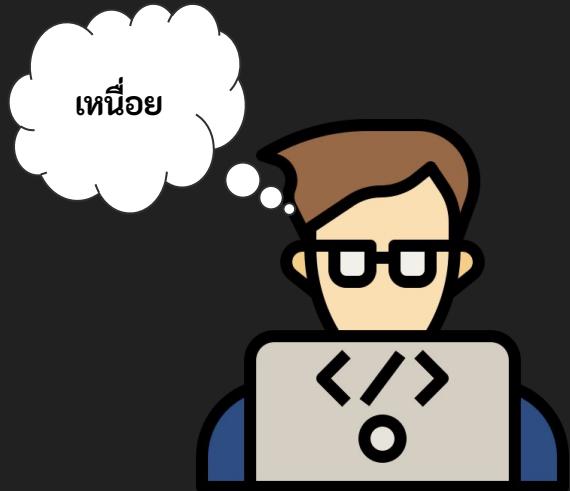
สาเหตุที่ต้องใช้ Version Control



อยากกลับไปแก้งานเก่าที่ทำไว้ จะต้องไปหาไฟล์ Backup และจำได้ไหมว่าไฟล์ Backup นั้น ถูกแก้ไขอะไรไปบ้าง
แก้ไขล่าสุดวันไหน แก้ไขเวลาใด ? / ไม่รู้



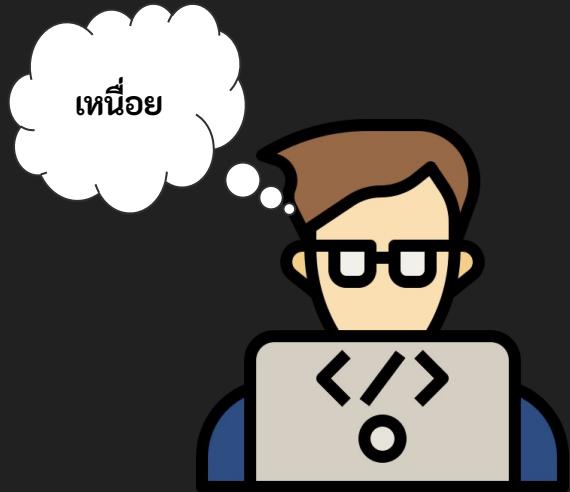
สาเหตุที่ต้องใช้ Version Control



อยากรอดลองเพิ่มฟีเจอร์ใหม่เข้าไป
ในโปรเจกต์เก่า จะทำอย่างไร
โดยที่ไม่ต้อง Copy โค๊ดทั้งโปรเจ็ค^ๆ
แล้วมาเปลี่ยนชื่อในภายหลัง / **ไม่รู้**

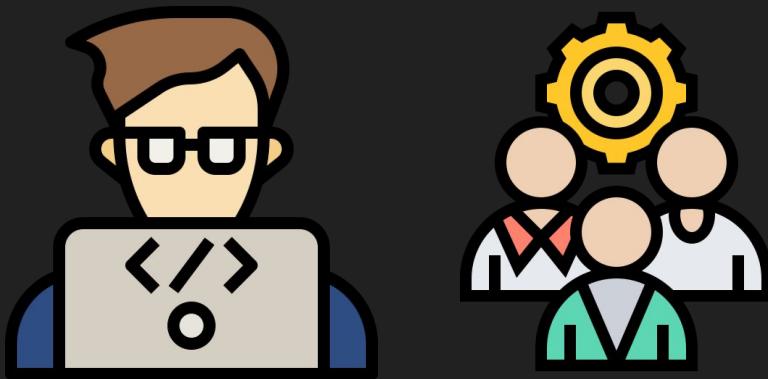


สาเหตุที่ต้องใช้ Version Control



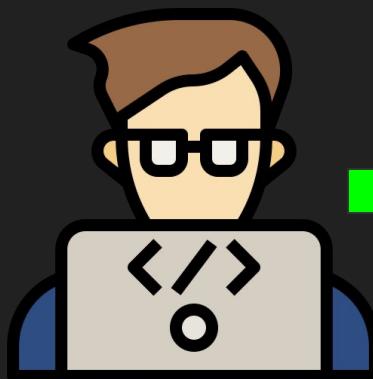
หลังเขียนโค้ดเสร็จต้องทำการ Backup
โค้ดเก่าเก็บไว้ทุกวัน ถ้าวันหนึ่งพื้นที่เก็บ
ข้อมูลเต็มจะทำอย่างไร ? / **ไม่รู้**

สาเหตุที่ต้องใช้ Version Control





สาเหตุที่ต้องใช้ Version Control

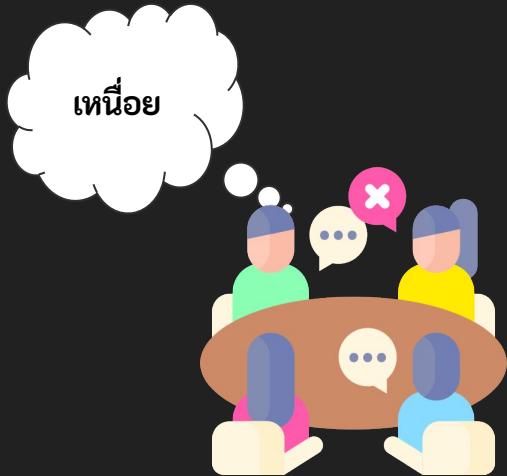


- project
- projectv2
- projectv3 fixbug
- projectv4 add feature
- projectv5 final
- projectv5 final fixbug





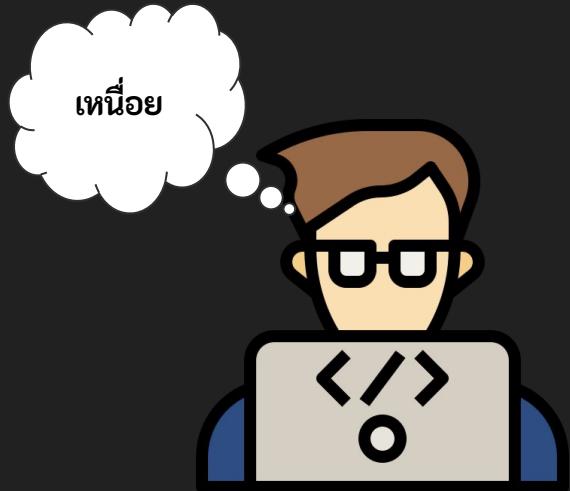
สาเหตุที่ต้องใช้ Version Control



- ต้องส่งงานที่ทำล่าสุดให้กันอย่างไร ?
- คนในทีมทำการปรับปรุงแก้ไขโค้ดอะไรมากบ้าง ?
- ถ้าแก้ไขโค้ดจุดเดียวกันจะเกิดปัญหาหรือไม่ ?
- อยากพัฒนาระบบที่มีอยู่ไม่ให้กระทบกับ Production จะทำอย่างไร ?

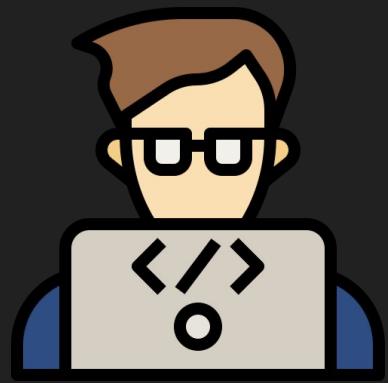


สาเหตุที่ต้องใช้ Version Control



- หากเครื่องมือมาช่วยในการเก็บการเปลี่ยนแปลงของไฟล์ได้ โดยการเก็บประวัติไฟล์ว่าถูกสร้าง/ลบ/แก้ไข โดยใคร เมื่อไหร่
- สามารถติดตามการเปลี่ยนแปลงของโค้ดในไฟล์ได้หรือย้อนเวลาโค้ดกลับไปก่อนตอนที่จะพังได้

สาเหตุที่ต้องใช้ Version Control



“ Version Control ”

BREAK!



วิัฒนาการของ Version Control



วิถีตามน้ำการของ Version Control

1. Copy File & Folder
2. Patch
3. Local Version Control System
4. Centralized Version Control System (CVCS)
5. Distributed Version Control System (DVCS)



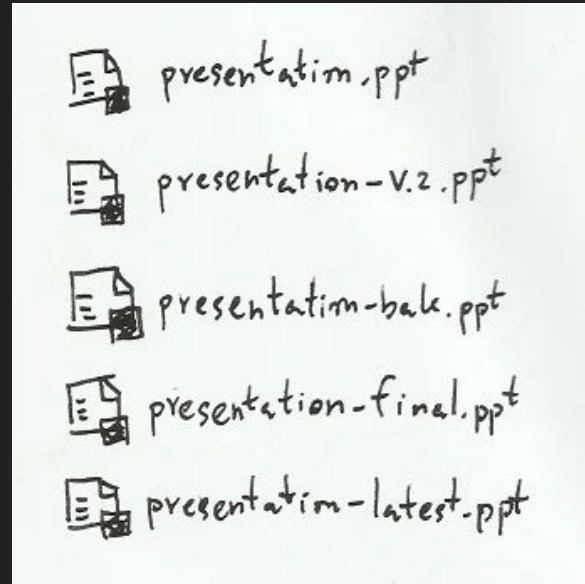
วิถีตามน้ำการของ Version Control

- 1. Copy File & Folder**
2. Patch
3. Local Version Control System
4. Centralized Version Control System (CVCS)
5. Distributed Version Control System (DVCS)



Copy File & Folder

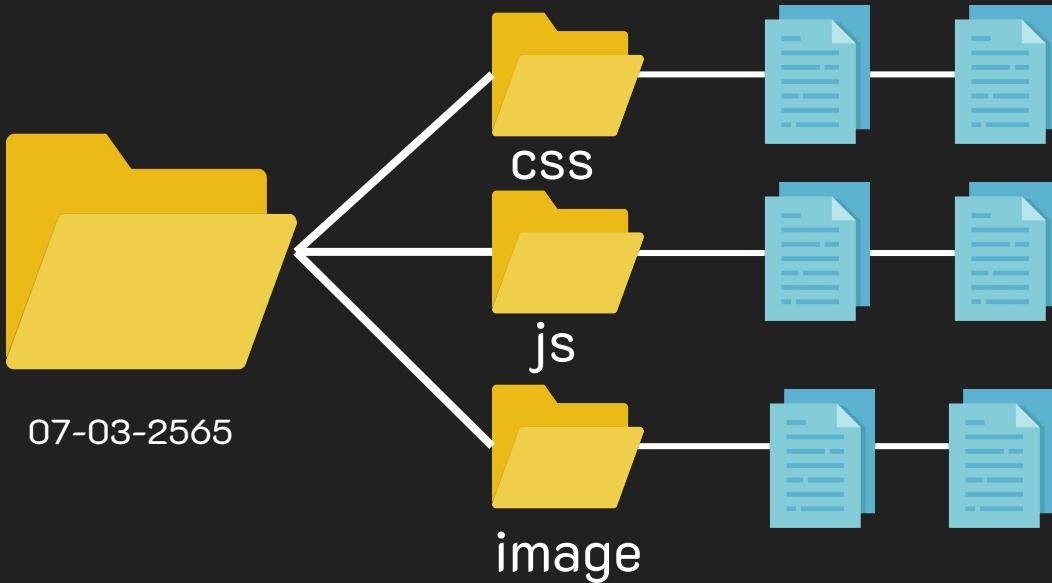
คือ ลักษณะการบันทึก
เอกสารและแยกออกเป็นหลาย
ไฟล์ แล้วตั้งชื่อไฟล์พร้อมระบุ,
เวอร์ชันตามลำดับ



<https://www.blognone.com/sites/default/files/externals/06af9600e2d49c14619e38cec7ed4905.jpg>

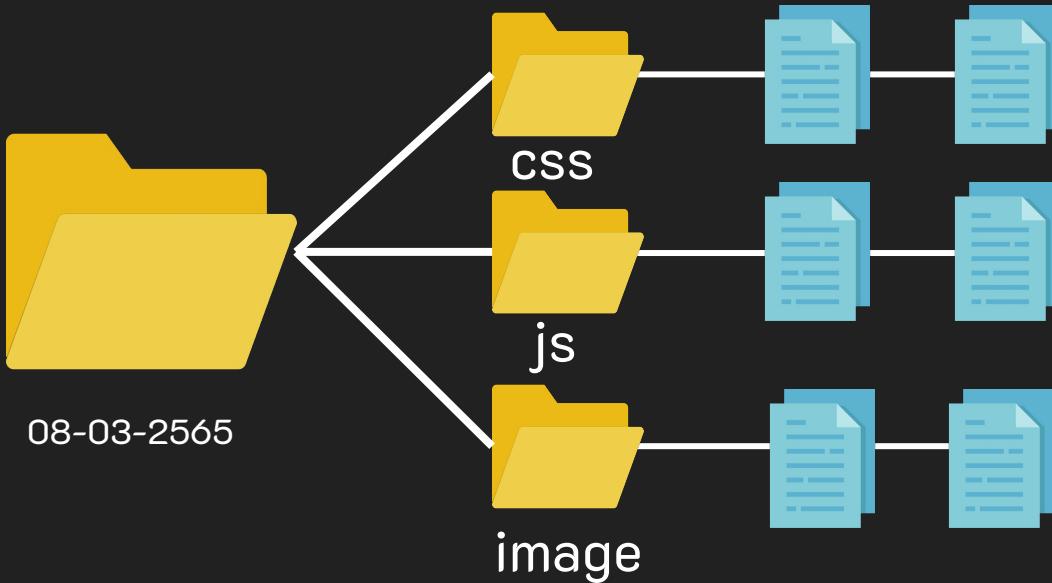


ตัวอย่าง



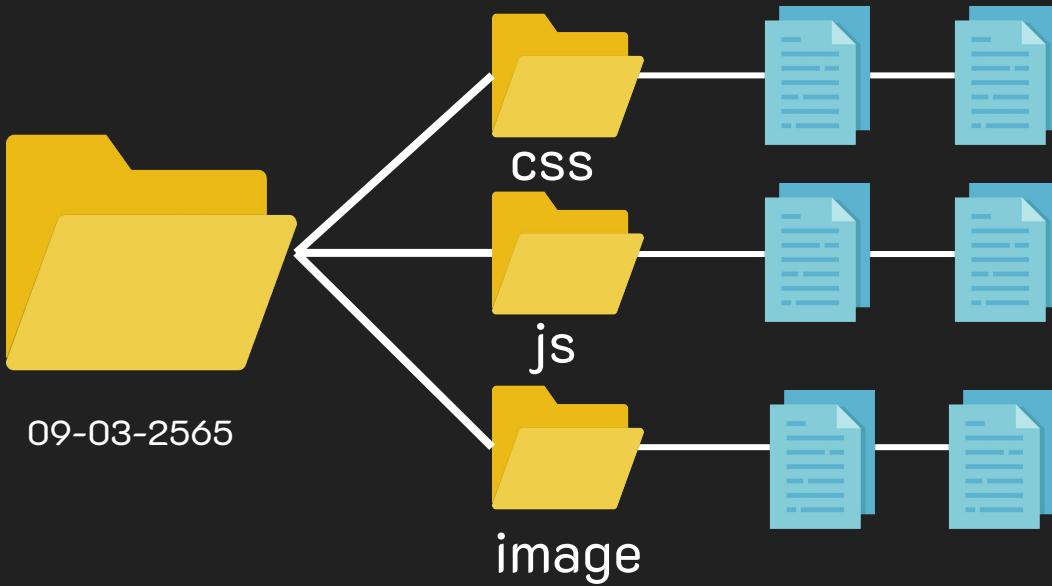


ตัวอย่าง





ตัวอย่าง





ตัวอย่าง



07-03-2565



08-03-2565



09-03-2565



ตัวอย่าง



project.py

07-03-2565



projectv1.py

10-03-2565



projectv1.2.py

15-03-2565



projectfinal.py

30-03-2565



Copy File & Folder

**ข้อเสีย

เปลี่ยนพื้นที่จัดเก็บข้อมูลตามจำนวนเวอร์ชันของเอกสาร เพราะต้องคัดลอกไฟล์ทั้งโฟลเดอร์เพื่อสร้างเอกสารเวอร์ชันใหม่ขึ้นมา



วิถีตามน้ำการของ Version Control

1. Copy File & Folder
2. Patch
3. Local Version Control System
4. Centralized Version Control System (CVCS)
5. Distributed Version Control System (DVCS)



Patch (แพตช์)

Patch (แพตช์) เพื่อเลี้ยงความสิ้นเปลืองพื้นที่ในการจัดเก็บแบบวิธี Copy File & Folder เกิดจากการเปรียบเทียบไฟล์เก่า กับไฟล์ใหม่ที่มีการเปลี่ยนแปลง



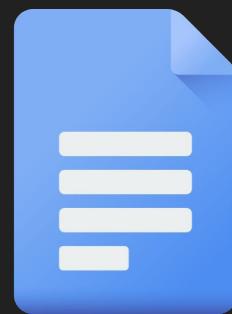
Original

Diff



Change

=



Patch



Patch (แพตช์)

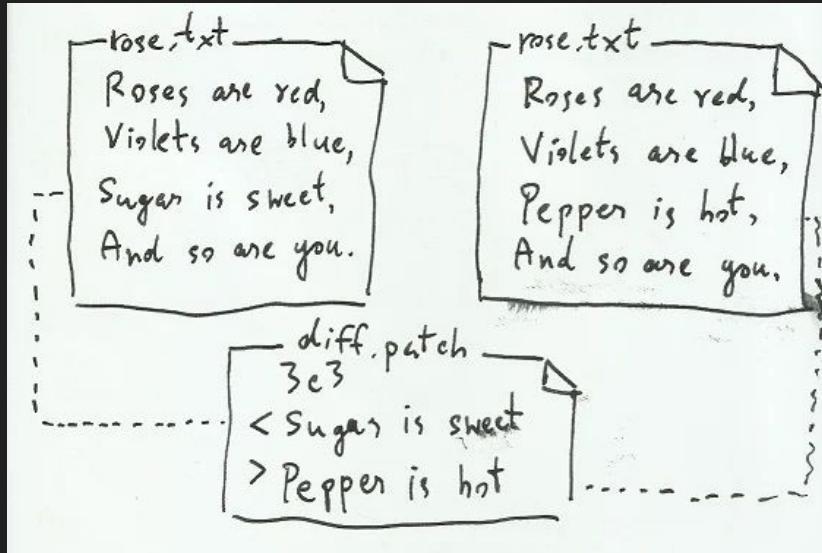
**ข้อเสีย

อาจมีความเสี่ยงต่อการทำ

Patch บาง Patch หาย จนไม่

สามารถประกอบร่าง Source Code

กลับไปยังเวอร์ชันต่างๆได้



<https://www.blognone.com/node/78730>



วิถีตามน้ำการของ Version Control

1. Copy File & Folder
2. Patch
- 3. Local Version Control System**
4. Centralized Version Control System (CVCS)
5. Distributed Version Control System (DVCS)



Local Version Control System

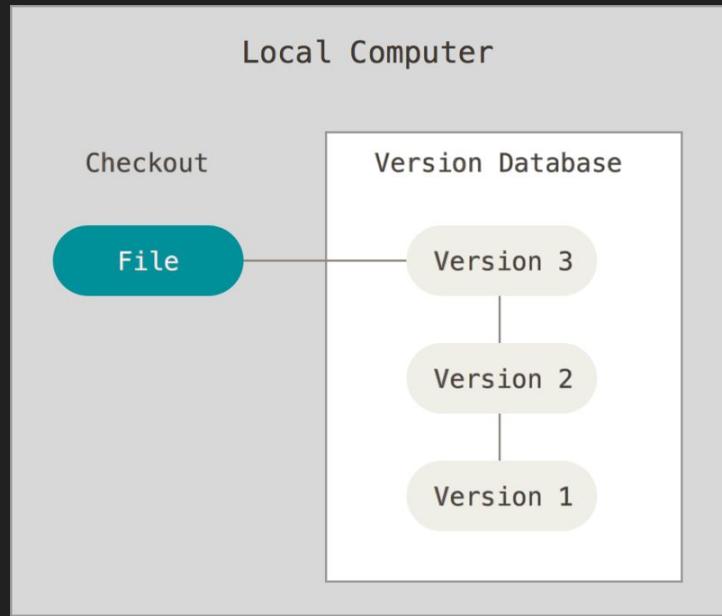


เพื่อแก้ปัญหา Patch หาย จนไม่สามารถประกอบร่าง Source Code กลับไปยัง เกอร์ชั่นต่างๆได้ จึงได้มีการพัฒนา Version Control System (VCS) ที่มีฐานข้อมูล เช่น คุณจัดเก็บทุกการเปลี่ยนแปลงของ Source Code โดยจะเรียกการจัดเก็บ เกอร์ชั่นของ Source Code ลงฐานข้อมูลเรียกว่า “Check-In” และเรียกคืน Source Code จากฐานข้อมูลเพื่อทำงานต่อว่า “Check-Out”



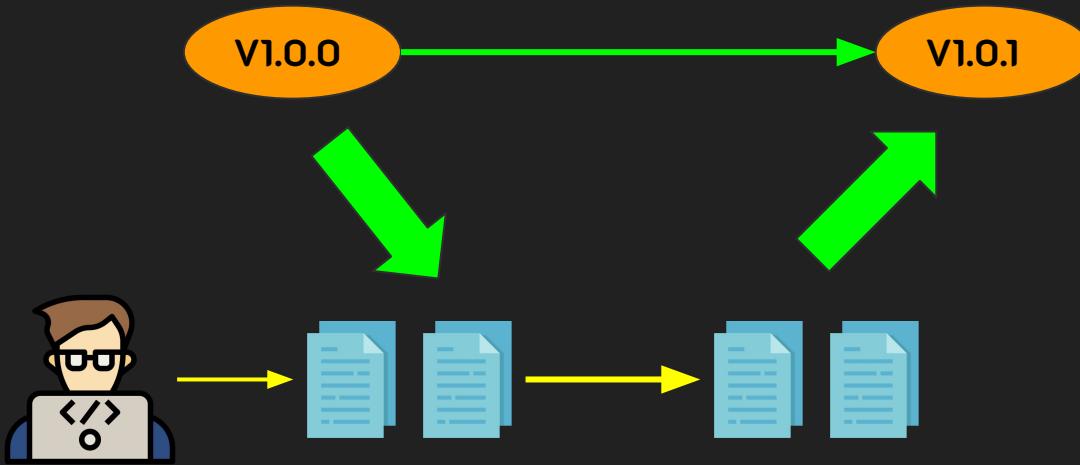
Local Version Control System

Local Version Control System มีความสามารถในการจัดเก็บเวอร์ชัน (Check-in) พร้อมทั้งข้อความช่วยจำ (Log Message) ลงในฐานข้อมูลและเรียกคืนเวอร์ชันจากฐานข้อมูล (Check-Out) กลับมายังพื้นที่ทำงาน เพื่อให้นักพัฒนาซอร์ฟแวร์แก้ไขซอฟต์แวร์ได้



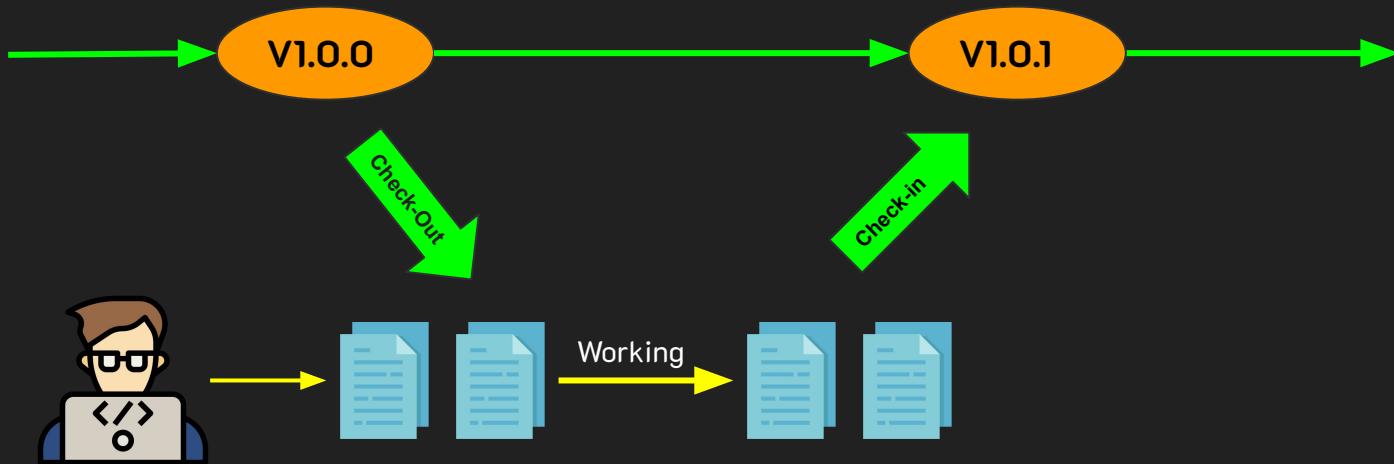


Local Version Control System



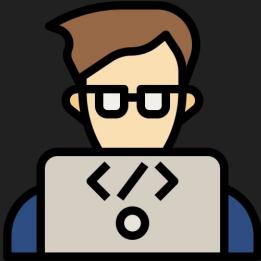


Local Version Control System





Local Version Control System



ข้อเสีย

การ Check-Out จากผู้ใช้หลายคน อาจทำให้เกิดปัญหาขึ้น



วิถีตามน้ำการของ Version Control

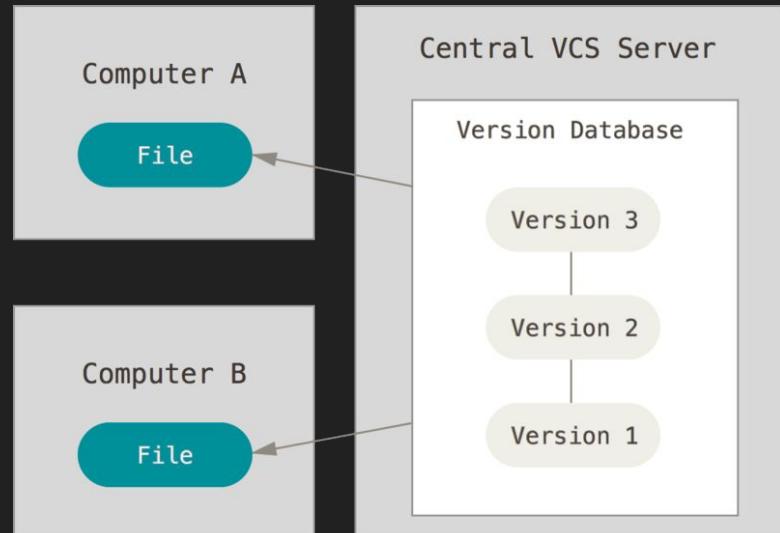
1. Copy File & Folder
2. Patch
3. Local Version Control System
- 4. Centralized Version Control System (CVCS)**
5. Distributed Version Control System (DVCS)



Centralized Version Control System

เป็น Version Control System แบบรวมศูนย์
เพื่อแก้ปัญหากรณีที่มีผู้ใช้งานหลายคนโดยการเก็บ
ข้อมูลไว้บน Server ซึ่งเมื่อผู้ใช้ Check-Out งาน
เวอร์ชันเดียวกันแล้วกลับมา Check-in ระบบจะ
พยายามรวมเนื้อหาเข้าด้วยกัน (Merge)

- ถ้ารูปแบบการรวมเนื้อหาไม่มีความเรียบง่าย
ระบบจะรวมเนื้อหาให้วัตโน้มัติ
- ถ้าการรวมมีความซับซ้อน ระบบจะแจ้งให้
ผู้ใช้งานตัดสินใจแทน



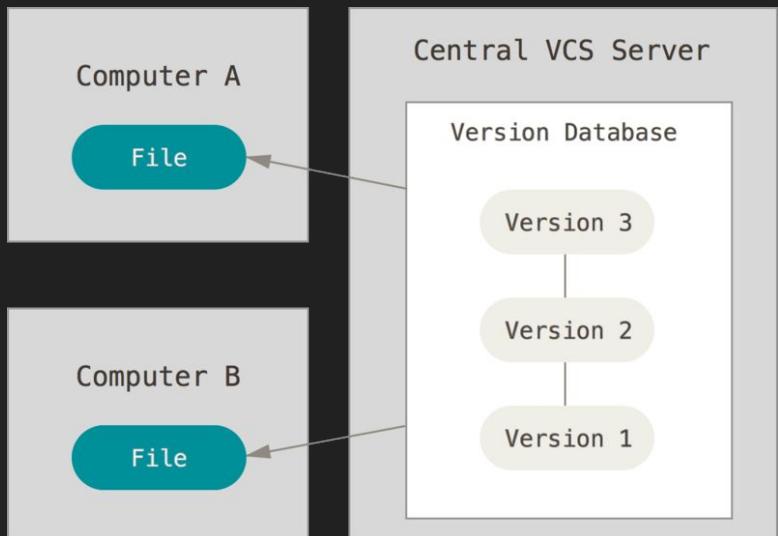
<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>



Centralized Version Control System

ข้อเสีย

- การทำงานแบบรวมศูนย์ หาก Server ล่ม ชั่วคราว ผู้ใช้จะไม่สามารถ Check-in หรือ Check-Out ได้
- ถ้า Server ล่มถาวร เวอร์ชันของ Source Code ทั้งหมดจะได้รับผลกระทบไปด้วย
- ทำงานแบบ Online เท่านั้น



<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>



วิถีตามน้ำการของ Version Control

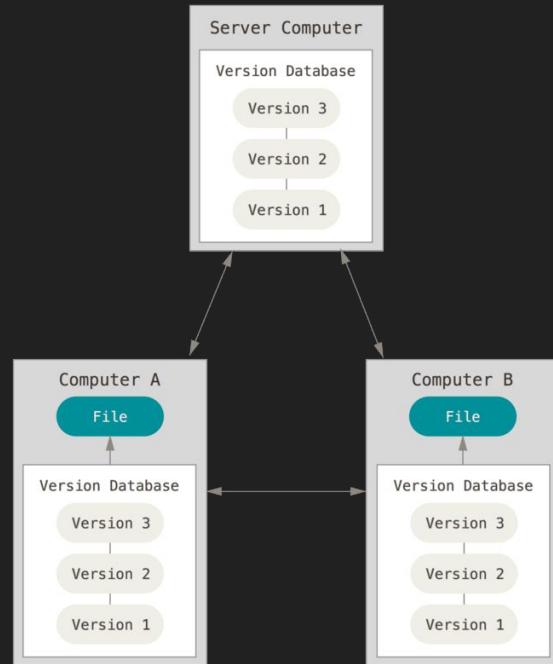
1. Copy File & Folder
2. Patch
3. Local Version Control System
4. Centralized Version Control System (CVCS)
5. Distributed Version Control System (DVCS)



Distributed Version Control System

Distributed Version Control System (DVCS)

จะแก้ปัญหาการทำงานแบบรวมศูนย์โดยการ
клонฐานข้อมูลมาทั้งหมด (**เรียกว Database**
ที่เก็บเวอร์ชันของโค้ดว่า Repository) ซึ่งเมื่อ
Clone มาแล้ว สามารถ Check-In และ
Check-Out บน Local Host แบบ Offline
ก่อนจะ Push ขึ้น Server ในภายหลัง





สรุป

จุดประสงค์ที่สำคัญของการใช้งาน Version Control System คือ เพื่อให้สามารถย้อนกลับไปยังเวอร์ชันก่อนหน้าได้ เมื่อพบปัญหาระหว่างพัฒนาโปรแกรม การ Check-in เพื่อเปลี่ยนแปลง Source Code ไปยังระบบจัดเก็บเวอร์ชัน (Version Control System) จึงเป็นเรื่องที่ต้องทำเป็นประจำ

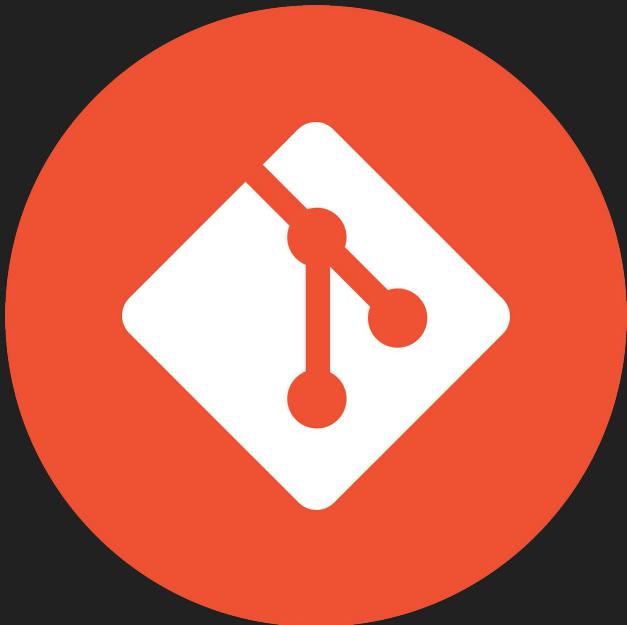
เพราะถ้าคนในทีมพัฒนาโปรแกรมไม่ทำการ Check-In เป็นประจำ แล้วพบปัญหาขึ้นกับซอฟแวร์ที่อยู่ระหว่างพัฒนา ก็อาจก่อให้เกิดปัญหาใหญ่ตามมาคือ ไม่สามารถย้อนกลับไปยังเวอร์ชันต่างๆก่อนหน้าได้ หรืออาจจะต้องยุ่งยากหรือใช้เวลานานกว่าจะย้อนกลับไปได้

BREAK!



ຮຽນຈັກກຳນົດ Git & GitHub

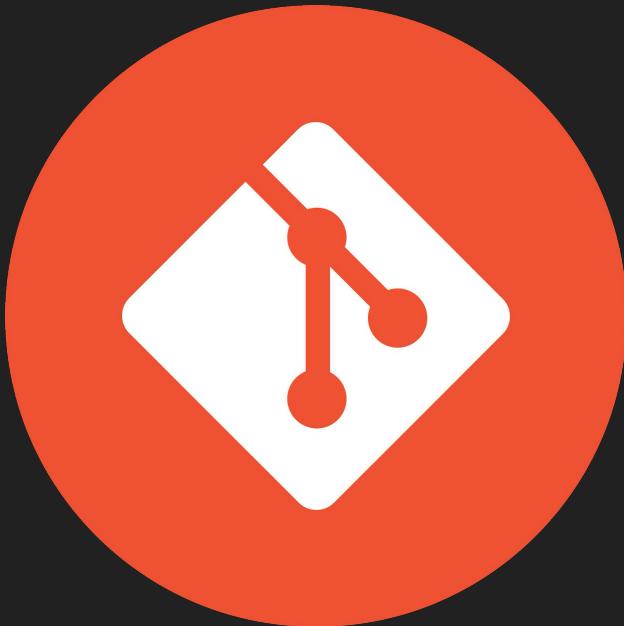
ทำความรู้จักกับ Git



Git เป็น Version Control รูปแบบ Distributed Version Control System (DVCS) ใช้สำหรับติดตาม ตรวจสอบการแก้ไขไฟล์ หรือ Source Code ซึ่งสามารถตรวจสอบได้ทุกตัวอักษร ทุกบรรทัด ทุกไฟล์ พร้อมทั้งระบุว่าไฟล์ดังกล่าวถูกปรับปรุงแก้ไข ณ ช่วงเวลาใด และใครเป็นผู้แก้ไข



ทำความรู้จักกับ Git



ทำให้ผู้พัฒนาสามารถติดตาม
การเปลี่ยนแปลงของโค๊ดได้ตลอด
หรือแม้กระทั่งย้อนเวลาโค๊ดกลับ
ไปก่อนตอนที่จะพังก็ยังทำได้



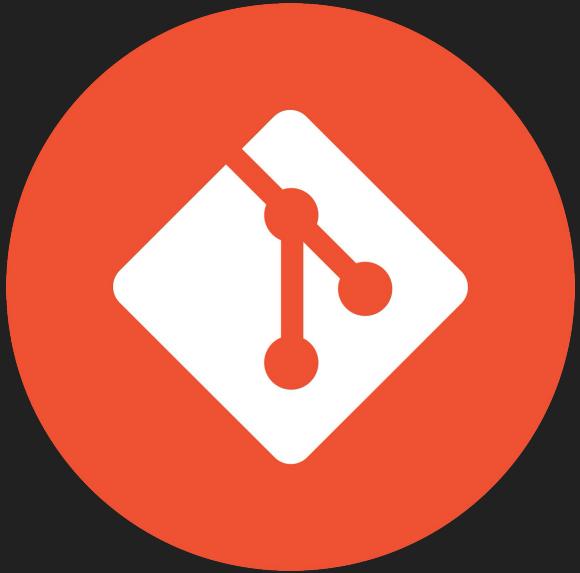


ทำความรู้จักกับ Git



ดังนั้น Git จึงเหมาะสมสำหรับนักพัฒนาไม่
ว่าจะเป็นงานแบบเดียวหรือแบบทีม เพราะ
สามารถเรียกดูได้ว่าคนในทีมใครเป็นคน
เขียนหรือใครเป็นคนแก้ไขโปรเจกต์ใน
แต่ละส่วน

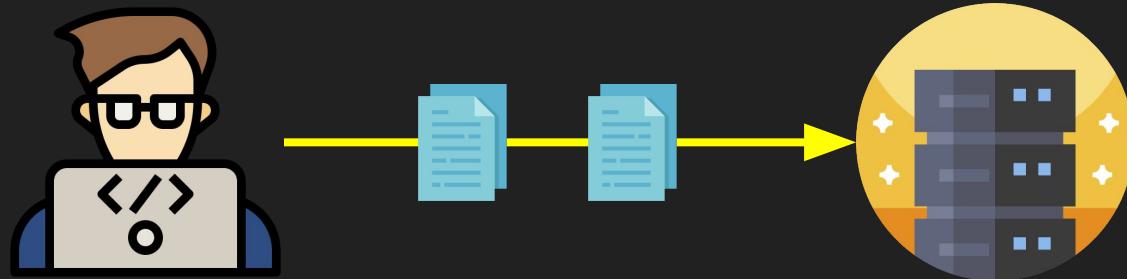
หลักการพื้นฐานของ Git



สามารถดัดลอกเวอร์ชันของ Source Code มาเก็บไว้ในเครื่องก่อนได้ (Local Host) ส่งผลให้การทำงานมีความรวดเร็วมากยิ่งขึ้น



หลักการพื้นฐานของ Git



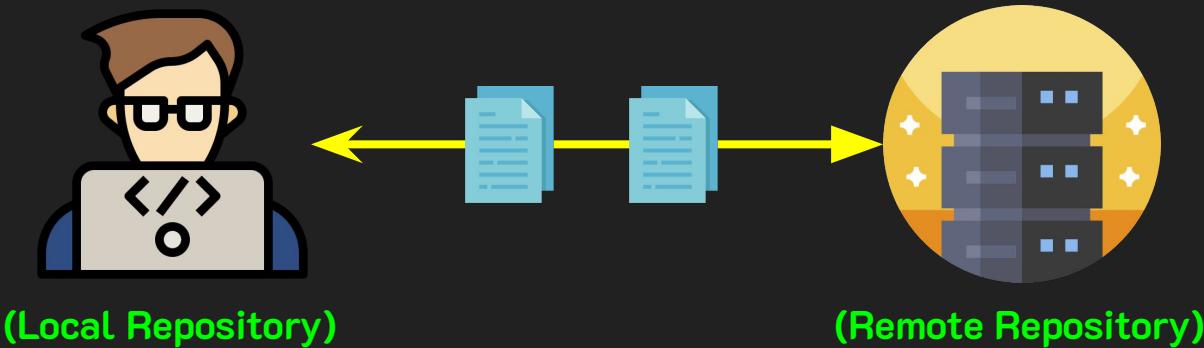
(Local Repository)

(Remote Repository)

ผู้พัฒนาสามารถแก้ไขโค๊ดโปรเจกต์ได้ทุกที่ในรูปแบบ Offline และ Check-In เพื่อจัดเก็บความเปลี่ยนแปลงของ Source Code ลงในฐานข้อมูลภายในเครื่องของตนเอง (**Local Repository**) โดยไม่จำเป็นต้องติดต่อกับ Git Repository บน Server (**Remote Repository**)



หลักการพื้นฐานของ Git



เมื่อแก้ไขเสร็จแล้วจึงค่อยสั่ง Sync เพื่อให้ Version ฝั่ง Local และ Server อัปเดตเหมือนกันในภายหลังด้วยวิธีการ Pull / Merge / Push



ทำไมต้องใช้ Git ?

1. ติดตามเวอร์ชันของโค๊ดได้ (Track Version)



เมื่อจัดเก็บไฟล์เข้าไปในระบบของ Git จะเรียกว่า Git Repository ซึ่งเก็บสำรองข้อมูลและการเปลี่ยนแปลงของ Source Code ทำให้สามารถย้อนกลับไปดูรายละเอียดการเปลี่ยนแปลงของแต่ละเวอร์ชัน ก่อนหน้าได้



ทำไมต้องใช้ Git ?

2. เครื่องมืออำนวยความสะดวกสำหรับการทำงานเป็นทีม

Git สามารถเก็บบันทึกการเปลี่ยนแปลงของ Source Code เวอร์ชันล่าสุด ไว้ที่ **Local Repository** ซึ่งไม่จำเป็นต้องต่ออินเทอร์เน็ตเวลาทำงาน ถ้าต้องการ อัพเดตการเปลี่ยนแปลงของ Source Code เวอร์ชันล่าสุดให้กับเพื่อนร่วมทีม ก็ สามารถที่จะ Push ขึ้นไปเก็บที่ **Remote Repository (Git Hosting : GitHub)** และเพื่อนร่วมทีมก็สามารถดึง (Pull) เวอร์ชันล่าสุดนั่นมารวมที่เครื่องของตนเอง ได้ ทำให้ Source Code ที่พัฒนาร่วมกันในทีมเป็นเวอร์ชันเดียวกัน



ทำความรู้จักกับ GitHub



เว็บไซฟ์เวอร์ที่ให้บริการในการ
ฝากไฟล์ Git หรือ Git ที่ทำงานบน
เว็บไซต์

ทำให้สามารถใช้ Git ร่วมกับคน
อื่นได้ผ่านเว็บไซต์ซึ่งจะมักนิยมใช้
เก็บ Project Open Source ต่างๆ



ทำความรู้จักกับ GitHub



BREAK!



ดาวน์โหลดและติดตั้ง Git



สมัครใช้งาน GitHub



วงจรการทำงานของ Git (Git WorkFlow)



วงจรการทำงานของ Git

Working
Directory
(Untracked)

Working
Directory
(Tracked)

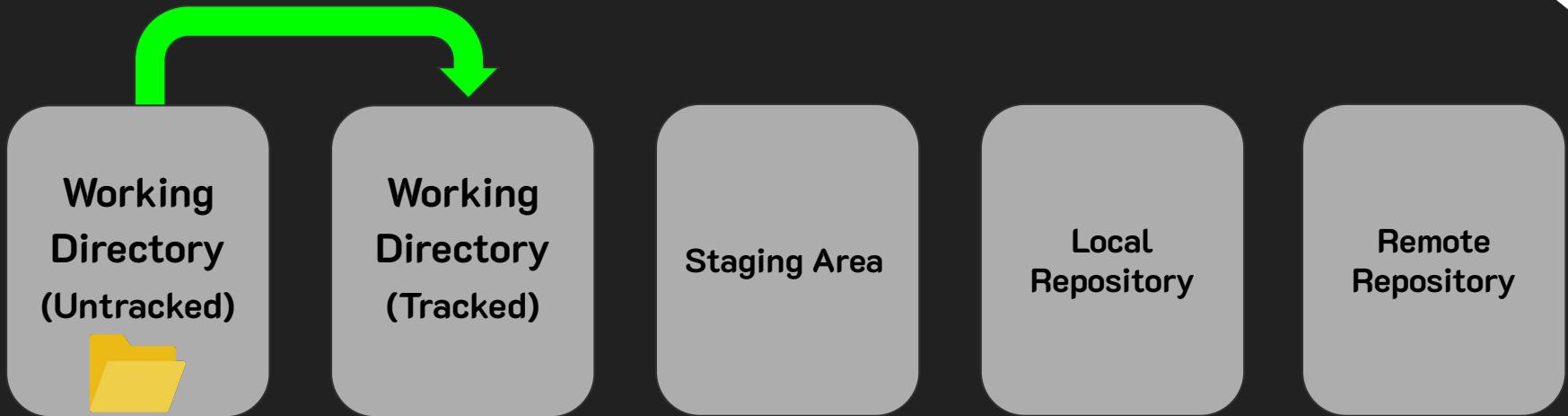
Staging Area

Local
Repository

Remote
Repository

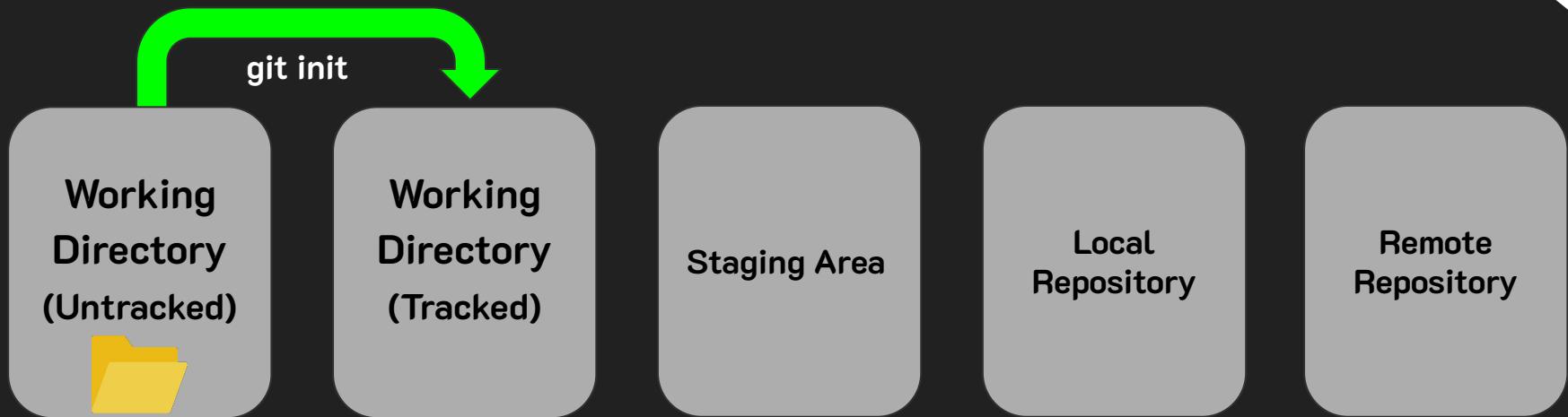


วงจรการทำงานของ Git



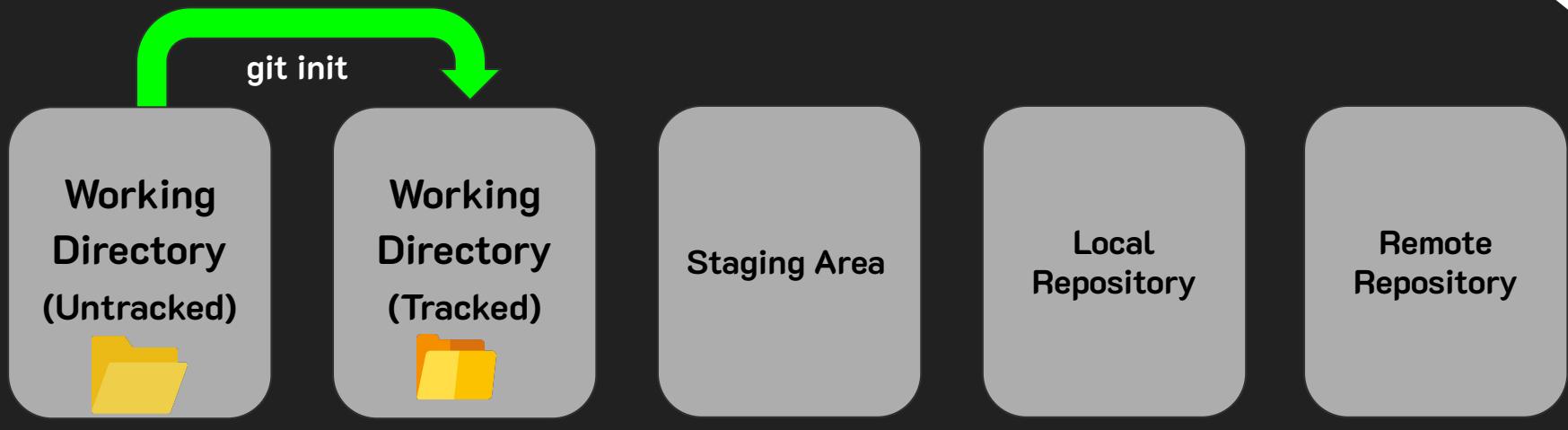


วุฒิการทำงานของ Git





วงจรการทำงานของ Git



ใช้ Git มาตรวจสอบ
ไฟล์ในโฟลเดอร์



วงจรการทำงานของ Git



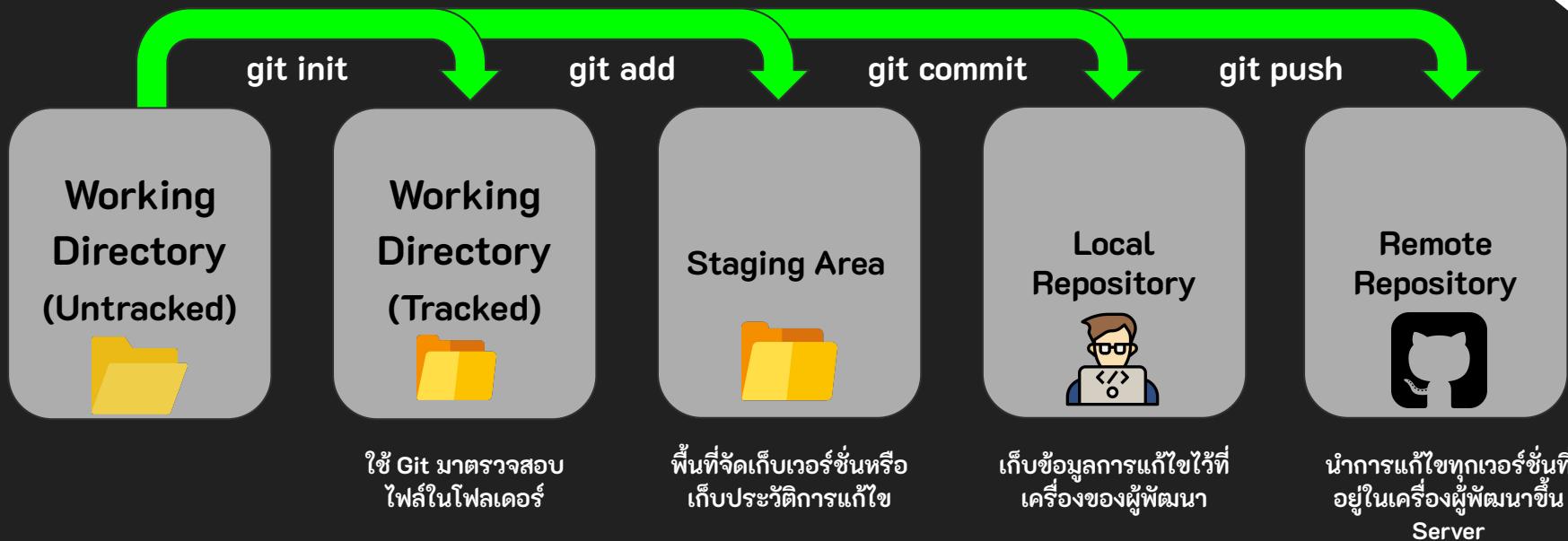


วงจรการทำงานของ Git





วงจรการทำงานของ Git





เริ่มต้นใช้งาน Git เป็นง่ายๆ



วงจรการทำงานของ Git





สร้าง Local Git Repository

- `git init`

การเพิ่มไฟล์เข้าไปใน Staging Area (Check-in)

- `git add <file_name>` => ระบุไฟล์ เช่น `git add index.html`
- `git add *.html` => เพิ่มหลายไฟล์พร้อมระบุนามสกุล
- `git add .` => เพิ่มทุกไฟล์ที่อยู่ภายใต้ Directory ปัจจุบัน



ลบไฟล์หรือโฟลเดอร์ออกจาก Git Repository (Remove)

- `git rm -r --cached .` => ลบทั้งหมด
- `git rm --cached <file_name>` => ระบุไฟล์

ตรวจสอบสถานะ (status) ดูสถานะการเปลี่ยนแปลงของ Repository

บนเครื่องของเรา (Local) เช่น การเพิ่ม ,แก้ไข, ลบ ไฟล์ต่างๆ

- `git status`



สถานะการติดตาม (Tracked Status)

- **Modified** หมายถึง มีการแก้ไขไฟล์แล้วแต่ยังไม่เริ่มต้นจัดเก็บลงบน Repository
- **Staged** หมายถึง ได้ทำเครื่องหมาย File ที่ได้ถูกแก้ไขเพื่อบันทึกในเวอร์ชั่นหน้า
- **Committed** หมายถึง ข้อมูลถูกบันทึกลงใน Repository เรียบร้อยแล้ว



Git Commit , Git Log



วงจรการทำงานของ Git





วงจรการทำงานของ Git





เก็บประวัติการแก้ไขดาวร (Commit)

- `git commit -m "Log Message"`

Option

- `m` เป็นพารามิเตอร์สำหรับใส่ข้อความช่วยจำ (Log Message) เพื่ออธิบายการ `commit` แต่ละเวอร์ชัน
- เมื่อ `Commit` ไปแล้ว จะได้ `SHA-1 Hash` เป็น `Commit ID` (รหัสประจำเวอร์ชัน) (`SHA-1 Hash = 40 ตัวอักษร แต่ตอนอ้างอิงใช้แค่ 7 ตัวอักษรแรก`)



ดูประวัติการ Commit (log)

โดยจะแสดง Commit ID, Message, ชื่อผู้เขียน , อีเมล , และเวลาที่ Commit

- `git log`
- `git log --oneline` -> แสดงแต่ละ log เหลือบรรทัดเดียว
- `git log --graph` -> แสดงเป็นเส้น Branch ให้ดูง่ายขึ้น

BREAK!



เปรียบเทียบเวอร์ชัน (diff)

ใช้สำหรับตรวจสอบและเปรียบเทียบไฟล์โดยดูว่ามีอะไรเปลี่ยนแปลง และแตกต่างไปจากเดิมบ้างเมื่อเทียบกับ Commit ที่ผ่านมา

- `git diff <commit_id>` // แบบระบุ Commit ID
- `git diff <commit_id> <commit_id>` // เปรียบเทียบระหว่างสอง Commit



เปรียบเทียบเวอร์ชั่น (diff)

สถานะ

- แสดงเครื่องหมาย - และตัวอักษรสีแดงในบรรทัดเดิมก่อนถูกแก้ไขและถูกลบ
- แสดงเครื่องหมาย + และตัวอักษรสีเขียวและโค๊ดใหม่ที่ถูกแก้ไขและเพิ่มใหม่



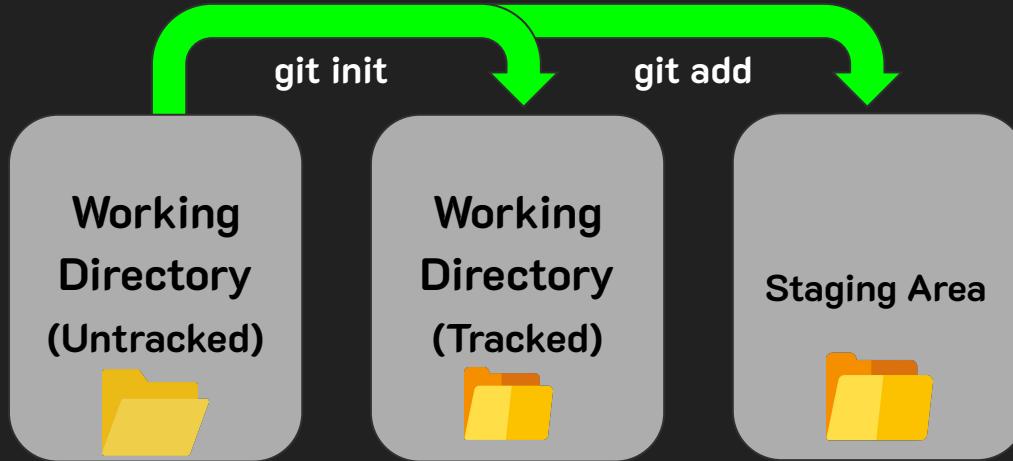
ยกเลิกการแก้ไขไฟล์ (Check-Out)

คำสั่งย้อนกลับไปยัง Commit ล่าสุด หรือยกเลิกการแก้ไขไฟล์

- `git checkout <file-name>`



Git Reset



เป็นการย้อนเวอร์ชันให้กลับไปอยู่ในสภาพก่อนที่จะ `add` ไฟล์เข้าสู่ Staging Area ซึ่งบางครั้ง
มีการเพิ่มไฟล์ลงใน Staging Area โดยไม่ตั้งใจ สามารถเอาออกได้โดยใช้ `git reset`

BREAK!



Git Reset (สำหรับย้อนคืนເວົ້ອຮ່ານ)

Commit #1

Commit #2

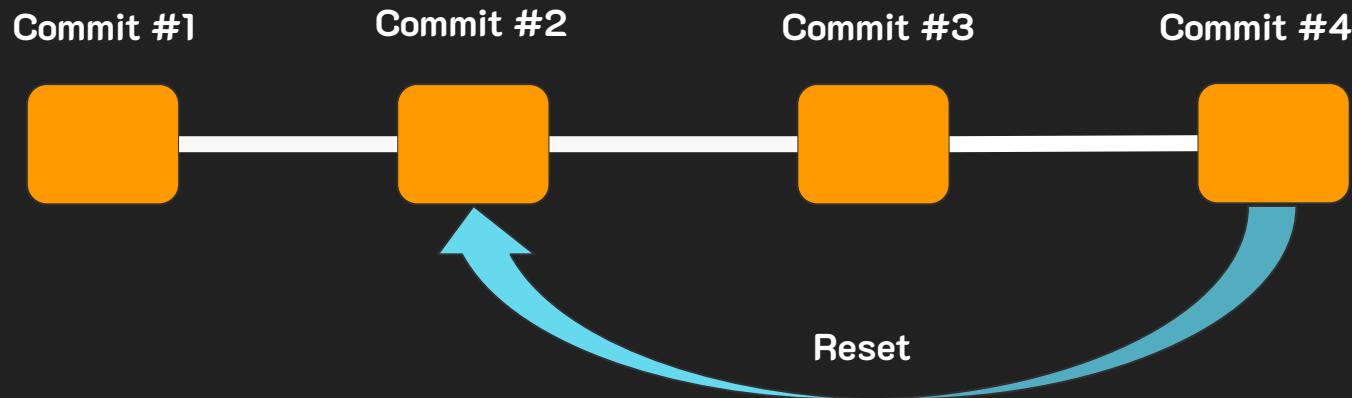
Commit #3

Commit #4





Git Reset (สำหรับย้อนคืนເວົ້ອຮ່ານ)





Git Reset (สำหรับย้อนคืนเวอร์ชัน)

`git reset -- option <commit_id>`

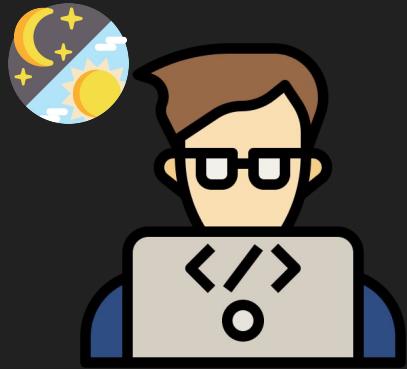
- **soft** ใช้เพื่อลบ Commit ทั้งหมดที่อยู่หลัง Commit ID และนำไฟล์ที่เคยอยู่ใน Commit นั้นกลับมายัง Staging Area
- **mixed** ใช้เพื่อลบ Commit ทั้งหมดที่อยู่หลัง Commit ID และนำไฟล์ที่เคยอยู่ใน Commit นั้นกลับมายัง Working Directory
- **hard** ใช้เพื่อลบ Commit ทั้งหมดที่อยู่หลัง Commit ID และจะลบไฟล์ที่เคยอยู่ใน Commit เหล่านั้น



Git Branching

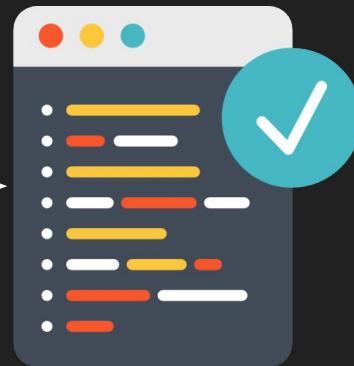


Git Branching



เด็กชายก่อจง

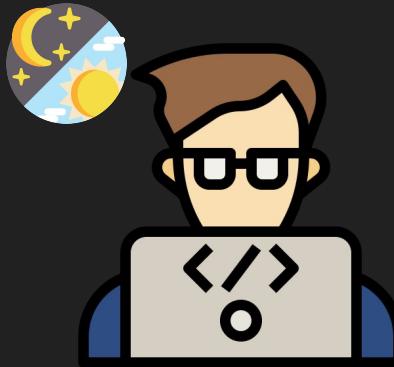
เขียนโค้ด



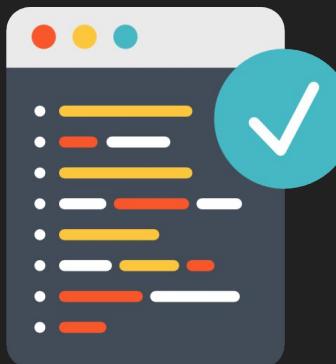
project



Git Branching



เด็กขายก้อง

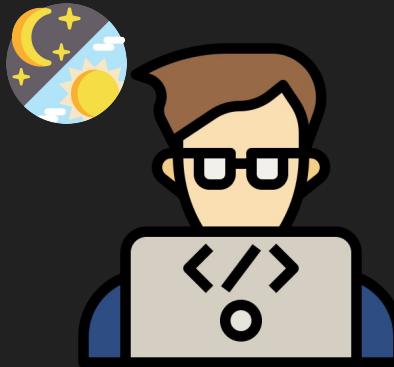


project

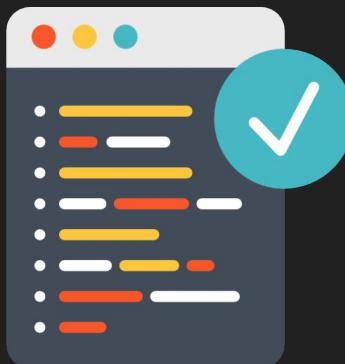
- เก็บข้อมูลการขายสินค้า



Git Branching

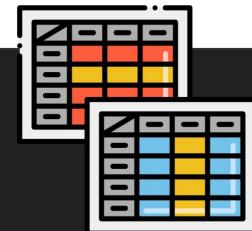


เด็กขายก้อง



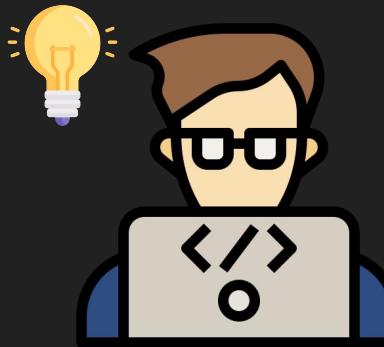
project

- เก็บข้อมูลการขายสินค้า

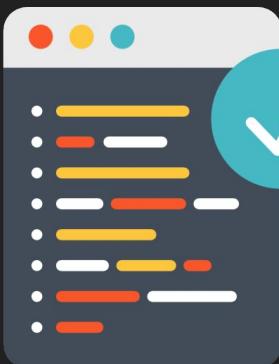




Git Branching



เด็กขายก้อง



project

- เก็บข้อมูลการขายสินค้า
- **รายงานยอดขายแต่ละเดือน





Git Branching

Commit #1

Commit #2

Commit #3





Git Branching

Commit #1

Commit #2

Commit #3



เก็บข้อมูลการขายสินค้า



Git Branching

Commit #1



Commit #2



Commit #3



Commit #4



Commit #5



เก็บข้อมูลการขายสินค้า

รายงานยอดขายแต่ละเดือน



Git Branching

Commit #1



Commit #2



Commit #3



Commit #4



Commit #5



เก็บข้อมูลการขายสินค้า

รายงานยอดขายแต่ละเดือน



Git Branching

Commit #1



Commit #2



Commit #3



Commit #4



Commit #5



เก็บข้อมูลการขายสินค้า

รายงานยอดขายแต่ละเดือน



Git Branching

Commit #1



Commit #2



Commit #3



Commit #4



Commit #5



เก็บข้อมูลการขายสินค้า

Git Reset



Git Branching

Commit #1

Commit #2

Commit #3



เก็บข้อมูลการขายสินค้า



Git Branching

Commit #1

Commit #2

Commit #3



เก็บข้อมูลการขายสินค้า



Git Branching

Commit #1



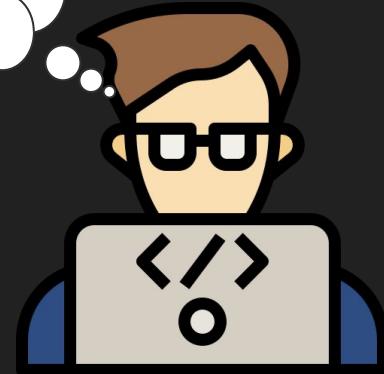
Commit #2



Commit #3



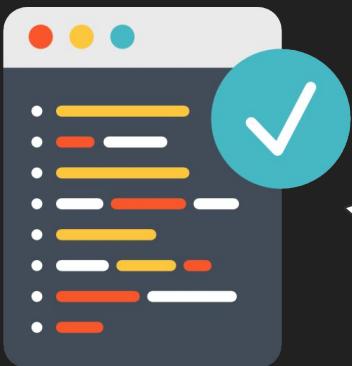
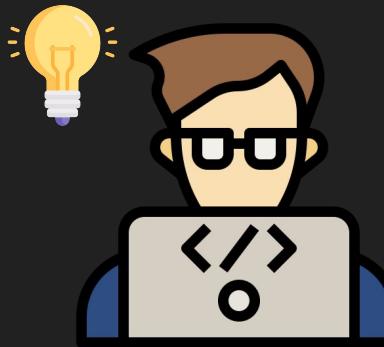
เห็นอยู่



เก็บข้อมูลการขายสินค้า



Git Branching



project

เด็กชายก้อง

ทำยังไงจึงจะสามารถพัฒนา
โปรเจกต์ใหม่ / เพิ่มฟีเจอร์เข้าไป
ในโปรเจกต์เก่าและใช้งานระบบใน
เวอร์ชันเก่าได้ด้วย ?



Git Branching

Commit #1

Commit #2

Commit #3





Git Branching

Commit #1

Commit #2

Commit #3

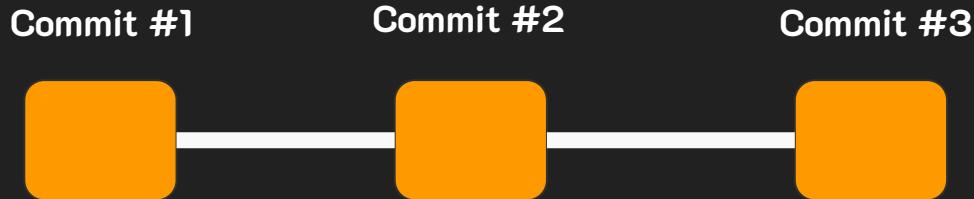


Master เป็น Branch Default (กึ่งหลัก)

ตอนใช้คำสั่ง `git init`



Git Branching



Master เป็น Branch Default (กึ่งหลัก)

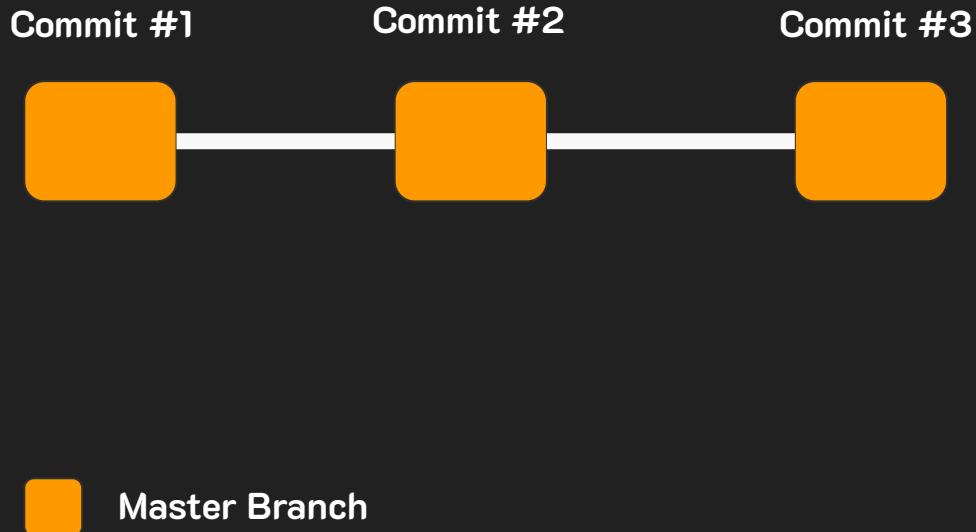
ตอนใช้คำสั่ง `git init`



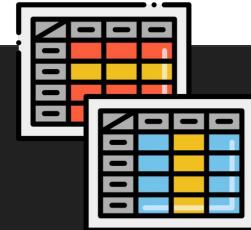
Master Branch



Git Branching

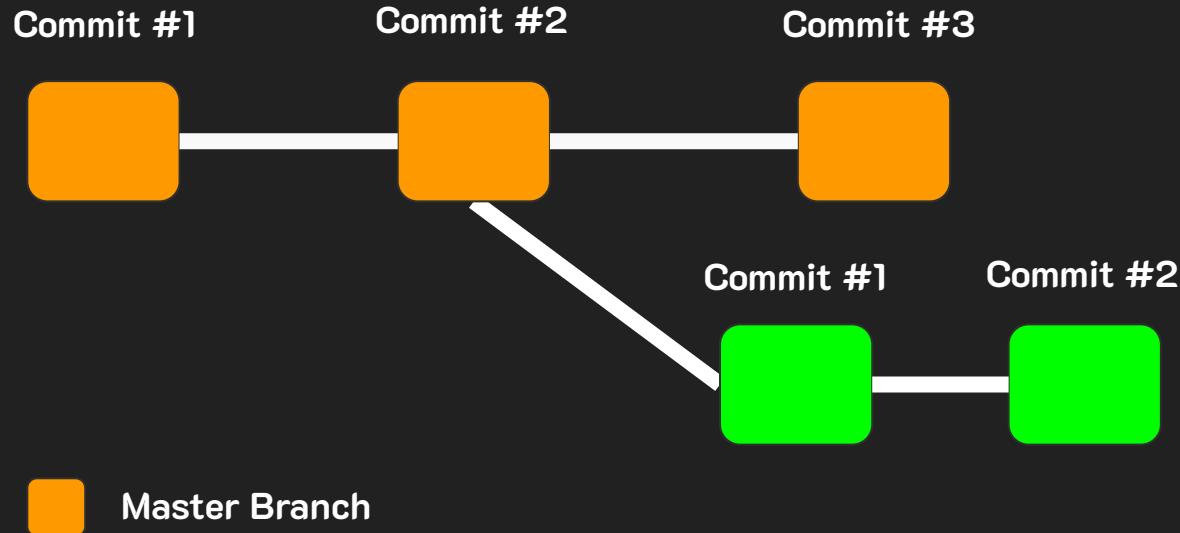


- เก็บข้อมูลการขายสินค้า



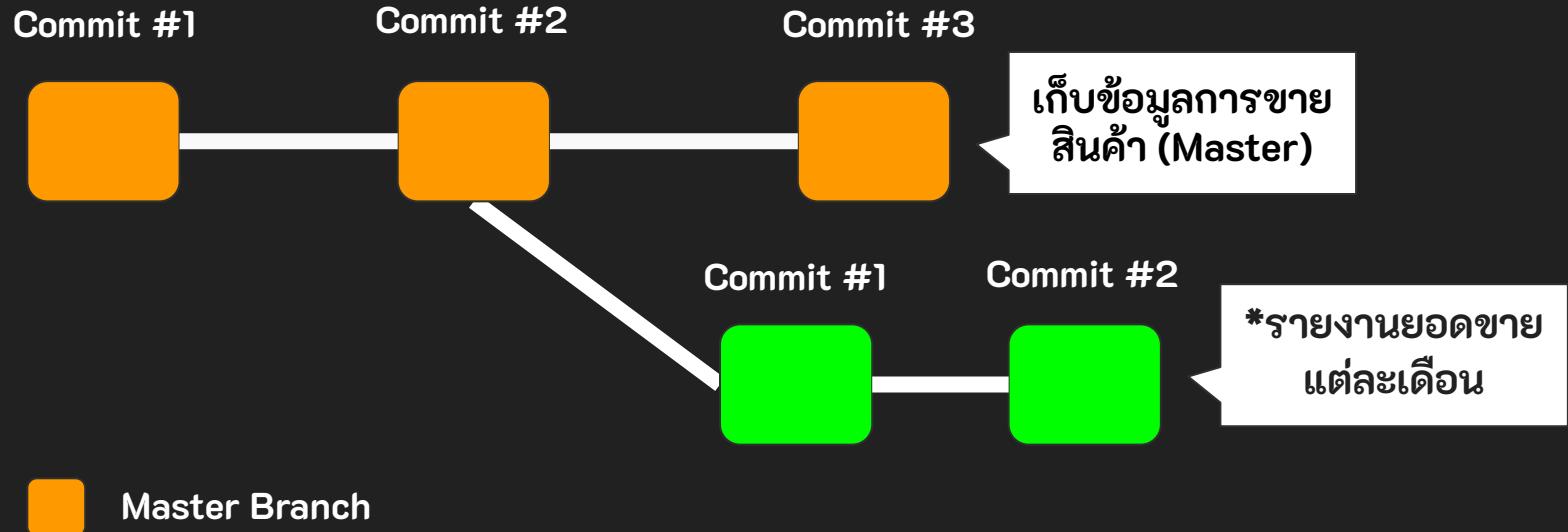


Git Branching



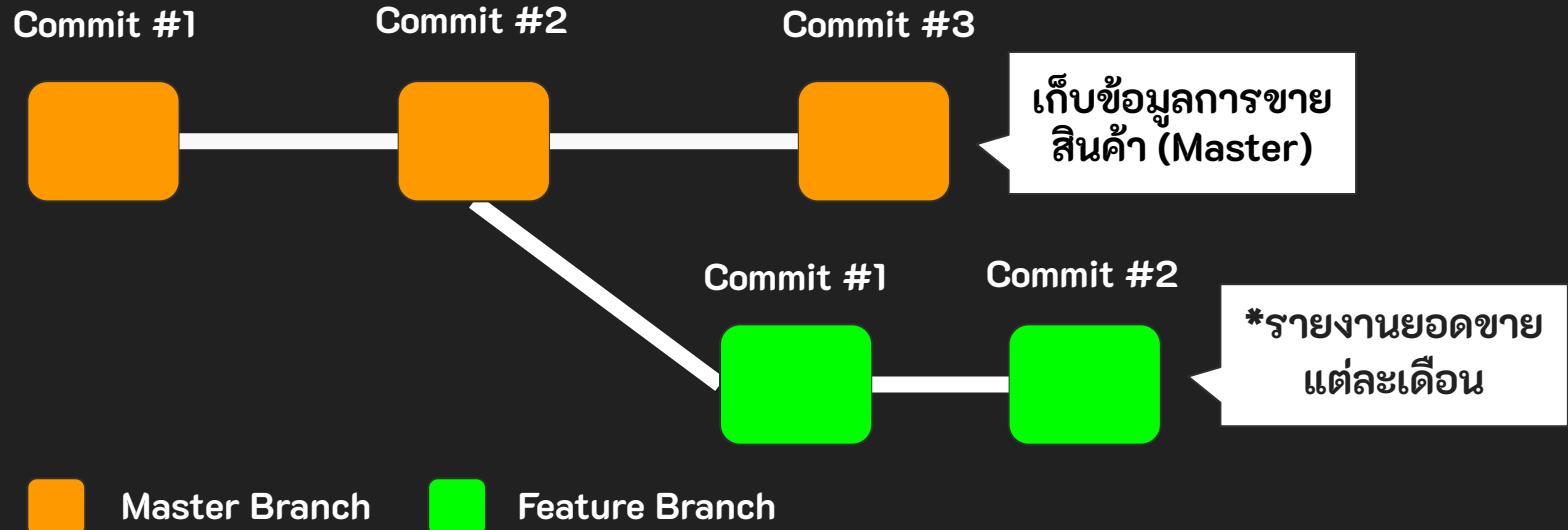


Git Branching



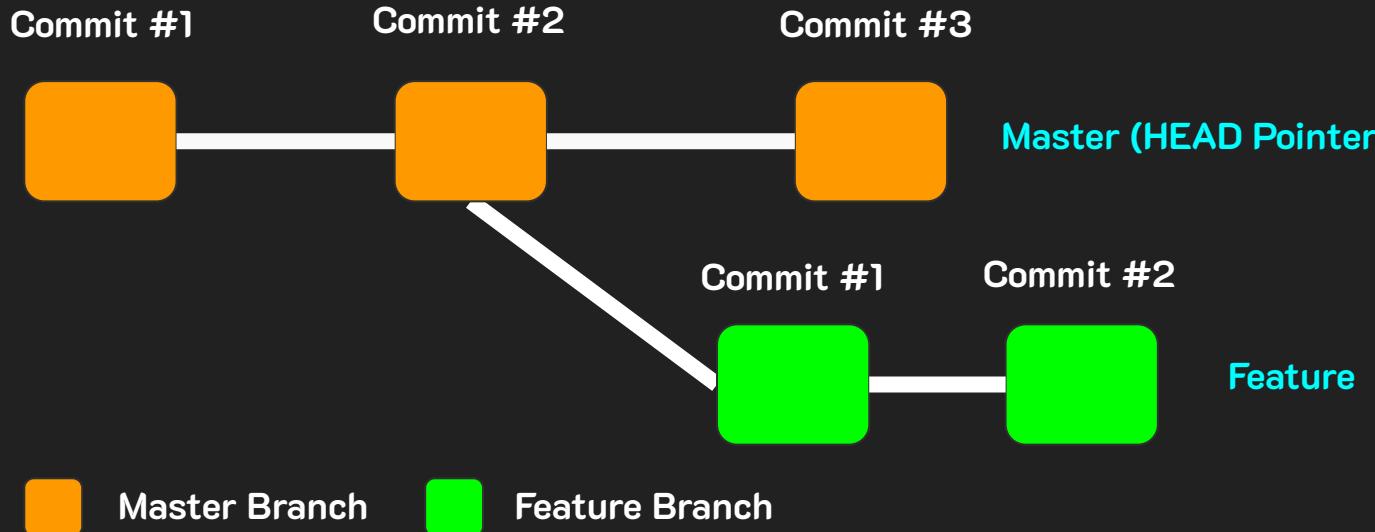


Git Branching



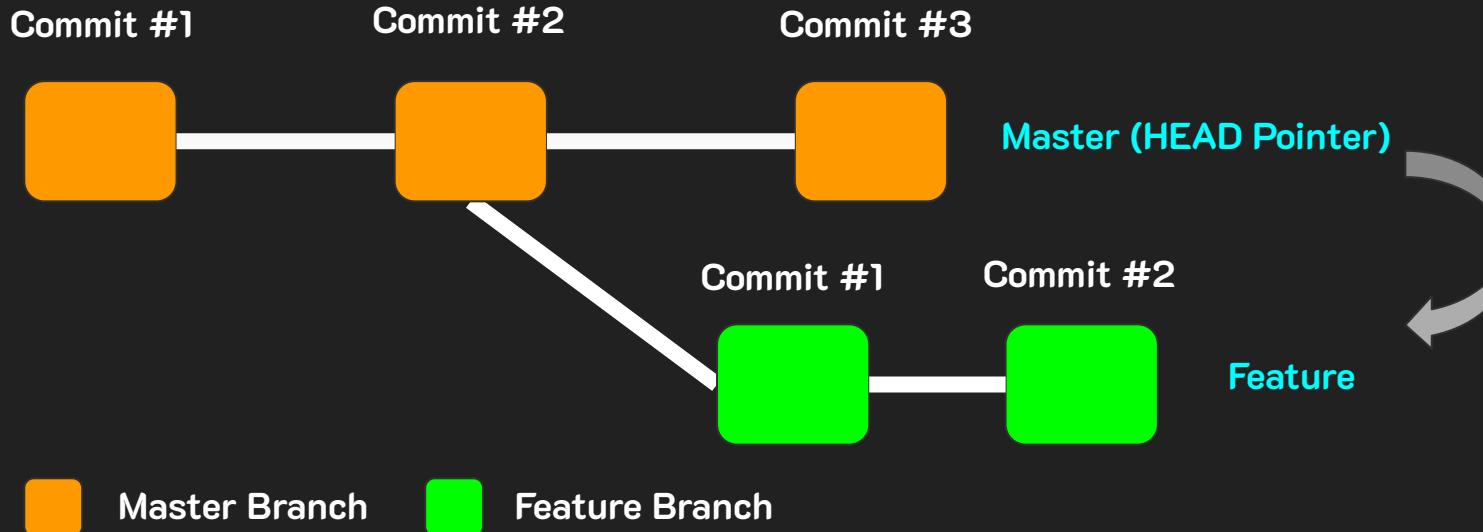


Git Branching

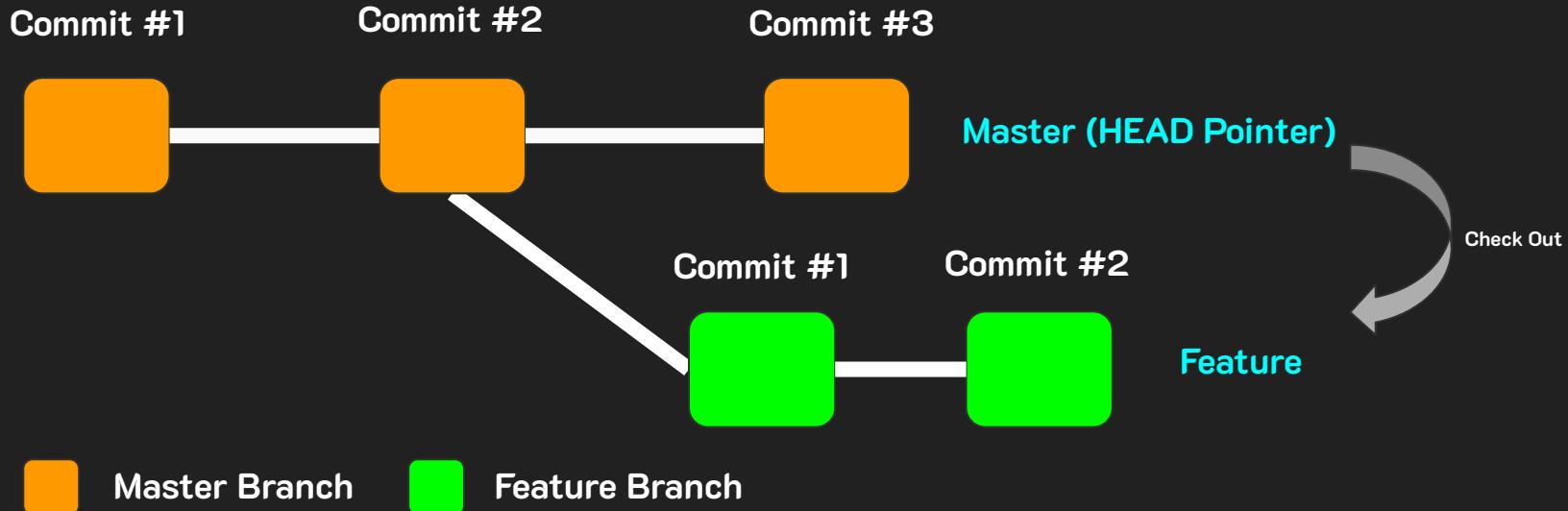




Git Branching

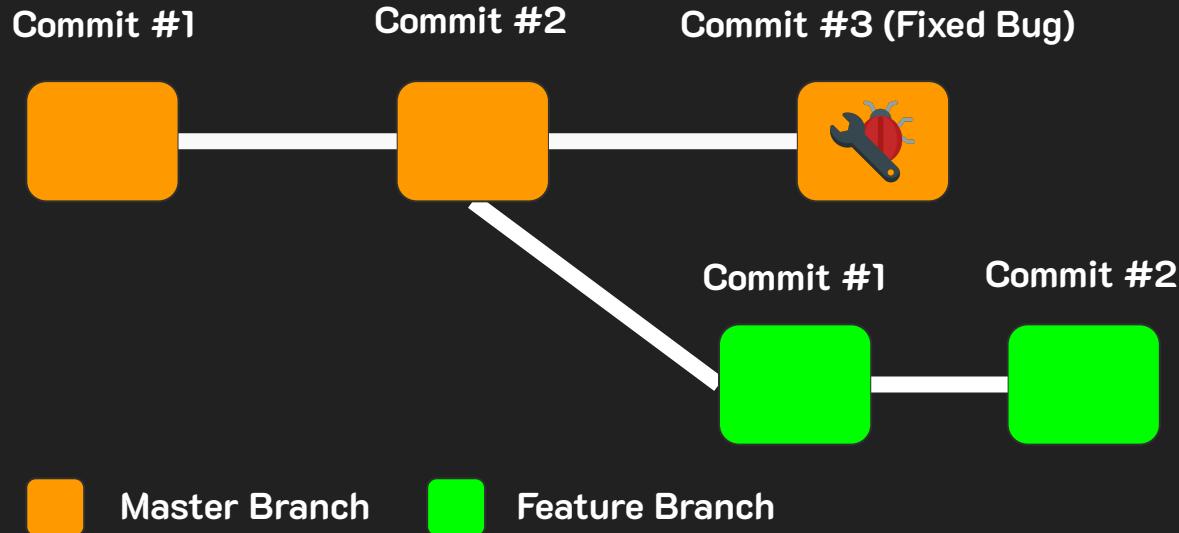


Git Branching



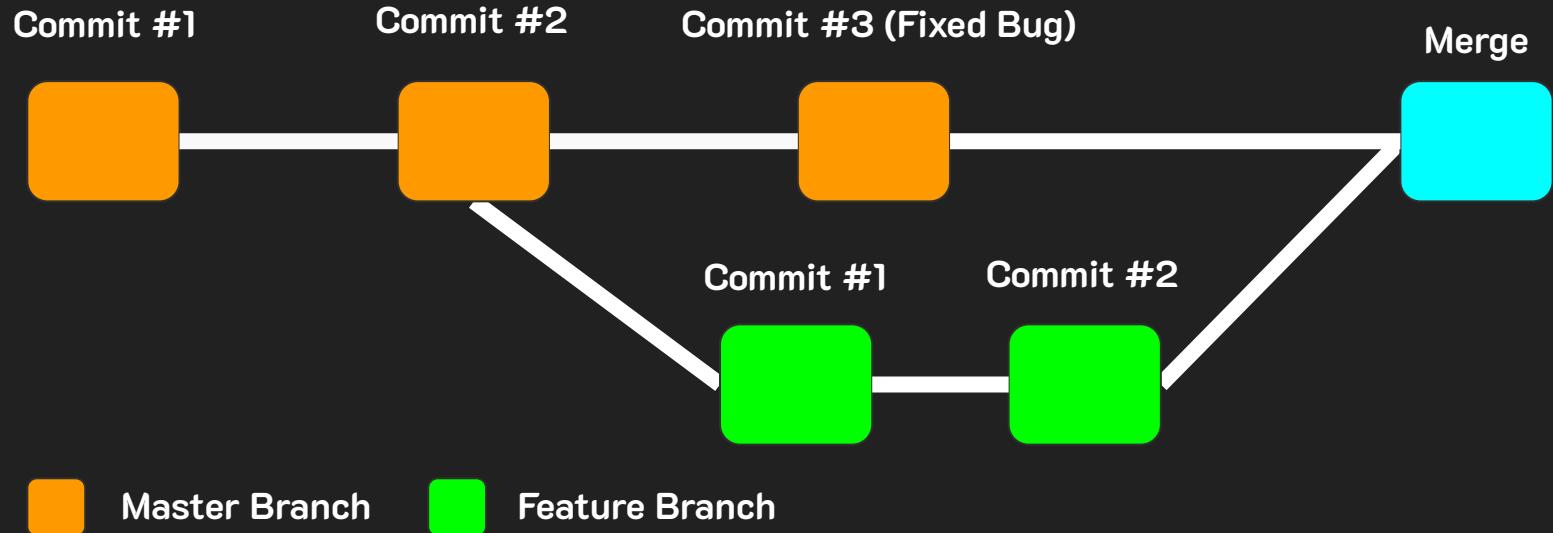


Git Branching





Git Branching



BREAK!



การจัดการ Git Branch เป็นต้น



คำสั่งที่เกี่ยวกับ Git Branching

- `git branch`
- `git checkout`
- `git merge`



คำสั่งที่เกี่ยวกับ Git Branching

- การแสดงชื่อ Branch

`git branch`

- การลับและสร้าง Branch

`git checkout -b <ชื่อ branch>` (ห้ามตั้งชื่อเว้นวรรค)



คำสั่งที่เกี่ยวกับ Git Branching

- การลบ Branch

```
git branch -d <ชื่อ branch>
```

- สลับไป Branch ที่ต้องการ

```
git checkout <ชื่อ branch>
```



คำสั่งที่เกี่ยวกับ Git Branching

- สลับไป Branch หลัก

`git checkout master`

- การรวม Branch

`git merge <ชื่อ branch>`

BREAK!

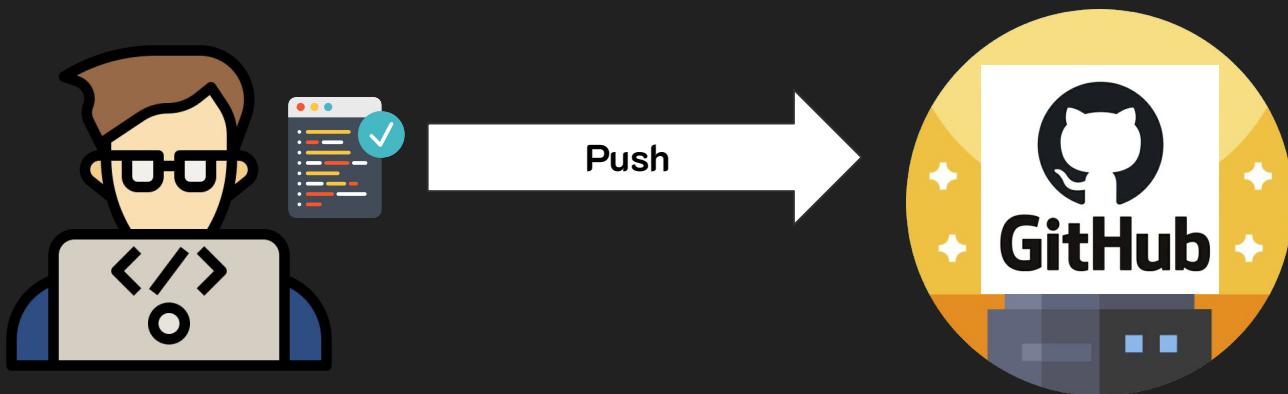


Remote Repository (Push / Pull)



Git Push (ผลัก)

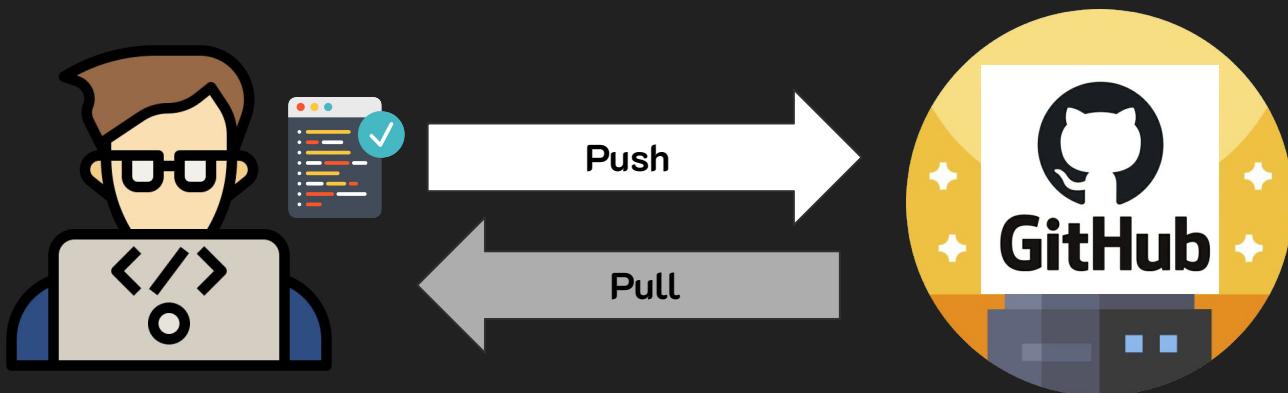
เป็นคำสั่งที่ใช้สำหรับนำสิ่งที่อยู่ในเครื่องของเรา (Local Repository) ไปอัพเดตให้กับ Remote Repository (Server)





Git Pull (ดึง + รวม)

เป็นคำสั่งที่ใช้สำหรับนำสิ่งที่อยู่บน Remote Repository (Server) มาอัพเดตในเครื่องของเรา (Local Repository)





Git Clone

เป็นคำสั่งที่ใช้สำหรับนำเอาโปรเจกต์ที่อยู่ใน Remote

Repository (Server) มาไว้ในเครื่องของเราหรือคนในทีม (Local Repository)

