นี่คือตารางเปรียบเทียบ Git กับ GitHub:

| **ลักษณะ** | **Git** | **GitHub** |
| --- | --- | --- |
| การใช้งาน | Git เป็นระบบควบคุมรุ่น (version control system) ที่ใช้สำหรับการจัดการรหัสทั้งในเครื่องคอมพิวเตอร์ของคุณและในพื้นที่ที่เป็นร่วม (remote repository) | GitHub เป็นเว็บไซต์และบริการโฮสติ้ง (hosting service) ที่ใช้ Git ในการจัดการรหัสและโปรเจคของคุณ |
| การใช้งานที่รอบด้าน | คุณสามารถใช้ Git โดยไม่ต้องเชื่อมต่ออินเทอร์เน็ต | GitHub เป็นเว็บไซต์และต้องใช้อินเทอร์เน็ตเพื่อเข้าถึง |
| การเก็บข้อมูล | Git เก็บข้อมูลโดยใช้ระบบที่เรียกว่า "เรปอิทอรี" (repository) ในเครื่องคอมพิวเตอร์ | GitHub เก็บข้อมูลในรูปแบบเรปอิทอรี บนเซิร์ฟเวอร์ออนไลน์ |
| การทำงานกลุ่ม | Git ช่วยให้ทีมสามารถทำงานร่วมกันได้โดยการแชร์และจัดการรหัสผ่านระหว่างสมาชิก | GitHub มีความสามารถในการทำงานร่วมกันของทีมที่ยอดเยี่ยม ที่ช่วยให้สมาชิกของทีมสามารถทำงานร่วมกันได้โดยอยู่ในระยะทางที่ห่างกัน |
| ความน่าเชื่อถือ | Git เป็นเครื่องมือที่น่าเชื่อถือในการจัดการรหัส มีประสิทธิภาพและเป็นที่ยอมรับอย่างกว้างขวาง | GitHub เป็นแหล่งเก็บข้อมูลที่น่าเชื่อถือสำหรับโปรเจ็คโอเพนซอร์ส (open-source projects) และโปรเจ็คภาคเอกชน (private projects) มีชุมชนที่ใหญ่ขนาดใหญ่และเครื่องมือที่มีประสิทธิภาพสำหรับการทำงานร่วมกัน |

ฉะนั้น, การเลือกใช้ Git หรือ GitHub ขึ้นอยู่กับความต้องการและวัตถุประสงค์ของโปรเจ็คของคุณว่าคุณต้องการเครื่องมือในการจัดการรหัสเพียงอย่างเดียว (Git) หรือคุณต้องการเครื่องมือที่ช่วยในการทำงานร่วมกันของทีม (GitHub) หรือไม่

A diagram of a diagram

Description automatically generated

**อธิบายภาพ Git Flow บน GitHub แบบเข้าใจง่าย**

**ภาพนี้แสดงกราฟการพัฒนาโครงการบน GitHub** แสดงความสัมพันธ์ระหว่างแขนงต่างๆ ของโครงการ และวิธีการไหลของการเปลี่ยนแปลง

**มีแขนงหลัก 4 แขนง**

1. **master:** แขนงที่เสถียรที่สุด เป็นเวอร์ชันล่าสุดของโครงการที่เผยแพร่สู่สาธารณะ
2. **develop:** แขนงสำหรับพัฒนาฟีเจอร์ใหม่!
3. **feature:** แขนงย่อยสำหรับพัฒนาฟีเจอร์เฉพาะ
4. **release:** แขนงสำหรับเตรียมการเปิดตัวใหม่

**ลูกศร** แสดงทิศทางการไหลของการเปลี่ยนแปลง

* **ลูกศรจาก develop ไป master:** แสดงว่าฟีเจอร์ใหม่จาก develop ถูกผสานเข้ากับ master
* **ลูกศรจาก feature ไป develop:** แสดงว่าฟีเจอร์จาก feature ถูกผสานเข้ากับ develop
* **ลูกศรจาก master ไป release:** แสดงว่าเวอร์ชันล่าสุดจาก master ถูกนำไปเตรียมการเปิดตัวบน release
* **ลูกศรจาก release ไป master:** แสดงว่าเวอร์ชันที่เปิดตัวแล้วจาก release ถูกผสานเข้ากับ master

**หลักการสำคัญ**

* **master** ควรมีโค้ดที่เสถียรและพร้อมใช้งานเสมอ
* **develop** ควรมีโค้ดล่าสุดที่กำลังพัฒนาฟีเจอร์ใหม่
* **feature** ใช้สำหรับพัฒนาฟีเจอร์เฉพาะ เมื่อเสร็จแล้วจะถูกผสานเข้ากับ develop
* **release** ใช้สำหรับเตรียมการเปิดตัวใหม่ เมื่อเสร็จแล้วจะถูกผสานเข้ากับ master

**กราฟนี้แสดงภาพรวมของกระบวนการพัฒนา GitHub ทั่วไป** ช่วยให้เข้าใจวิธีการทำงานของแขนงต่างๆ และวิธีการไหลของการเปลี่ยนแปลง

**เพิ่มเติม**

* โครงการ GitHub อาจมีแขนงเพิ่มเติมขึ้นอยู่กับความซับซ้อน
* กระบวนการพัฒนา GitHub อาจแตกต่างกันไปขึ้นอยู่กับทีมและองค์กร

A diagram of a diagram

Description automatically generated

**อธิบายภาพนี้ที่บน Git**

**ภาพนี้แสดงถึงกระบวนการคอมมิท (commit) ใน Git** โดยใช้วงกลมที่เชื่อมต่อกันด้วยเส้นสาย แต่ละวงกลมแสดงถึงขั้นตอนที่แตกต่างกัน ดังนี้

* **Working area:** วงกลมสีขาว แสดงถึงพื้นที่ทำงานของผู้พัฒนาซอฟต์แวร์ (developer) ซึ่งเป็นที่ที่ผู้พัฒนาสร้างและแก้ไขไฟล์
* **Staging area:** วงกลมสีฟ้า แสดงถึงพื้นที่ Staging ซึ่งเป็นที่ที่ผู้พัฒนาเลือกไฟล์ที่จะคอมมิท
* **History area:** วงกลมสีเขียว แสดงถึงพื้นที่ History ซึ่งเป็นที่เก็บ Commit ทั้งหมดของโครงการ
* **Generation:** ข้อความด้านบนวงกลมสีขาว แสดงถึงการเปลี่ยนแปลงที่ผู้พัฒนาทำกับไฟล์ใน Working area
* **GIT:** ข้อความด้านบนวงกลมสีฟ้า แสดงถึงคำสั่ง Git ที่ใช้ในการคอมมิทไฟล์
* **git add:** ข้อความด้านล่างวงกลมสีฟ้า แสดงถึงคำสั่ง Git add ซึ่งใช้ในการเพิ่มไฟล์จาก Working area ไปยัง Staging area
* **git commit:** ข้อความด้านล่างวงกลมสีเขียว แสดงถึงคำสั่ง Git commit ซึ่งใช้ในการคอมมิทไฟล์จาก Staging area ไปยัง History area

**ขั้นตอนการคอมมิทใน Git มีดังนี้**

1. ผู้พัฒนาสร้างหรือแก้ไขไฟล์ใน Working area
2. ผู้พัฒนาใช้คำสั่ง git add เพื่อเลือกไฟล์ที่จะคอมมิท
3. ผู้พัฒนาใช้คำสั่ง git commit เพื่อคอมมิทไฟล์

**คำอธิบายเพิ่มเติม**

* การคอมมิทเป็นวิธีเก็บประวัติการเปลี่ยนแปลงไฟล์ในโครงการ Git
* ผู้พัฒนาสามารถคอมมิทไฟล์ได้บ่อยเท่าที่ต้องการ
* แต่ละ Commit จะมีข้อความอธิบายที่ผู้พัฒนาสามารถเขียนได้
* ผู้พัฒนาสามารถย้อนกลับไปยัง Commit ก่อนหน้าได้โดยใช้คำสั่ง git checkout

**ตัวอย่าง**

สมมติว่าผู้พัฒนาซอฟต์แวร์กำลังพัฒนาโปรแกรมและได้ทำการเปลี่ยนแปลงไฟล์สองไฟล์ใน Working area ผู้พัฒนาต้องการคอมมิทไฟล์ทั้งสองไฟล์ไปยัง Git repository

ผู้พัฒนาสามารถทำดังนี้:

git add file1.txt file2.txt

git commit -m "เพิ่มฟีเจอร์ใหม่."

คำสั่ง git add จะเลือกไฟล์ file1.txt และ file2.txt ไปยัง Staging area

คำสั่ง git commit จะคอมมิทไฟล์จาก Staging area ไปยัง History area และข้อความ "เพิ่มฟีเจอร์ใหม่" จะถูกใช้เป็นข้อความอธิบาย Commit

**สรุป**

ภาพนี้แสดงถึงกระบวนการคอมมิท (commit) ใน Git ซึ่งเป็นวิธีเก็บประวัติการเปลี่ยนแปลงไฟล์ในโครงการ Git ผู้พัฒนาซอฟต์แวร์สามารถคอมมิทไฟล์ได้บ่อยเท่าที่ต้องการ และแต่ละ Commit จะมีข้อความอธิบายที่ผู้พัฒนาซอฟต์แวร์สามารถเขียนได้

**การปรับปรุง**

* เปลี่ยน "ผู้ใช้" เป็น "ผู้พัฒนาซอฟต์แวร์"
* เปลี่ยน "โปรแกรม" เป็น "โครงการ"
* เปลี่ยน "ข้อความ" เป็น "ข้อความอธิบาย"
* เปลี่ยน "คำสั่ง" เป็น "คำสั่ง Git"
* ปรับสำนวนให้เป็นทางการมากขึ้น
* ลบรายละเอียดที่ไม่จำเป็นออก

**หมายเหตุ**

* คำศัพท์บางคำอาจถูกปรับให้เหมาะสมกับบริบทของผู้ใช้งาน
* ตัวอย่างการใช้งานคำสั่ง Git อาจถูกปรับให้เหมาะสมกับสถานการณ์
* ผู้ใช้ควรศึกษาข้อมูลเพิ่มเติมเกี่ยวกับ Git ก่อนใช้งานจริง

A screenshot of a computer

Description automatically generated

ได้เลยครับ ผมจะอธิบายภาคนี้ให้ฟังแบบกันเองมากขึ้น

จากภาพที่ส่งมา เป็นโปรแกรมเทอร์มินัลที่มีข้อความเกี่ยวกับการอัปเดตโค้ด คิดง่ายๆ ว่าเหมือนสมุดบันทึกการเปลี่ยนแปลงโค้ด

**ลองจินตนาการว่า:**

* **Neetibut** นักเขียนโค้ดคนเก่ง กำลังพัฒนาโค้ดโปรแกรมเจ๋งๆ อยู่
* Neetibut อัปเดตโค้ดไปเรื่อยๆ ทุกครั้งที่มีการเปลี่ยนแปลง เขาก็จะบันทึกไว้ในสมุดบันทึกนี้
* ข้อความในสมุดบันทึกนี้ บอกเราว่า Neetibut เปลี่ยนแปลงอะไรในโค้ดบ้าง เปลี่ยนแปลงเมื่อไหร่ ใช้เครื่องมืออะไรช่วย
* ข้อความแรก บอกชื่อ Neetibut และวันที่เวลาที่อัปเดตโค้ดล่าสุด
* ข้อความที่สอง บอกว่า Neetibut เปลี่ยนแปลงชื่อฟุตเตอร์ของโค้ด คิดง่ายๆ ว่าเหมือนเปลี่ยนชื่อท้ายกระดาษ
* ข้อความที่สาม บอกชื่อการอัปเดตโค้ดนี้ว่า "commit 966f55fd6ce6d7f23fecdc274c061755ba148ae5" และชื่อโค้ดที่อัปเดตคือ "master"
* ข้อความที่สี่ บอกว่า Neetibut เปลี่ยนแปลงโค้ดไปทั้งหมด 40 จุด
* ข้อความที่ห้า บอกว่าการเปลี่ยนแปลงโค้ดนี้ยังไม่ได้ตรวจสอบ
* ข้อความที่หก บอกว่า Neetibut ใช้ปลั๊กอินช่วยเขียนโค้ดชื่อ "tabnine starter" คิดง่ายๆ ว่าเหมือนโปรแกรมช่วยพิมพ์
* ข้อความที่เจ็ด บอกว่า Neetibut ใช้เครื่องมือจัดรูปแบบโค้ดชื่อ "Prettier" คิดง่ายๆ ว่าเหมือนโปรแกรมจัดหน้ากระดาษ

**สรุป:**

* Neetibut อัปเดตโค้ดโปรแกรม "master" ล่าสุดเมื่อวันที่ 24 เมษายน 2567 เวลา 10:41 น.
* Neetibut เปลี่ยนแปลงโค้ดไปทั้งหมด 40 จุด
* Neetibut ใช้ปลั๊กอิน "tabnine starter" ช่วยเขียนโค้ด และใช้เครื่องมือ "Prettier" จัดรูปแบบโค้ด

**หมายเหตุ:**

* ข้อมูลในภาพไม่เพียงพอที่จะบอกได้ว่า Neetibut เปลี่ยนแปลงโค้ดอะไรบ้าง จำเป็นต้องดูรายละเอียดเพิ่มเติมจากระบบควบคุมเวอร์ชันโค้ด
* ข้อมูลในภาพอาจไม่เข้าใจง่ายสำหรับคนทั่วไป ผมพยายามอธิบายให้เข้าใจง่ายขึ้นแล้ว แต่หากยังมีคำถามเพิ่มเติม ถามได้เลยครับ