

git ฉบับอนุบาล ๒



@AorJoa

สารบัญ

เรื่อง	หน้า
Chapter 1 : Introduction	
1.1 เครื่องมือความคุ้มต้นรหัสข้อมูล (Version Control System : VCS)	1
- Local Version Control System : LVCS	1
- Centralized Version Control System : CVCS	2
- Distributed Version Control System : DVCS	3
1.2 ประวัติของ Git	4
1.3 การทำงานของ Git	6
Chapter 2 : Git basic	
2.1 ศัพท์แสง	16
2.2 คำสั่งพื้นฐานของ Git	18
Chapter 3 : Installation	
3.1 Windows	29
3.2 Mac OS X	36
3.3 Others operating system	38
Chapter 4 : Git scenario	
สถานการณ์ที่ 1 : มือใหม่ มีนิติบัญญัติ !@#%!@	39
สถานการณ์ที่ 2 : ฝากไว้ใน Remote repo เธอ	44
สถานการณ์ที่ 3 : กลับมาเหมือนเดิมนะ,,ข้อมูล	50
สถานการณ์ที่ 4 : เจอ Conflict	53
สถานการณ์ที่ 5 : เพิ่ม Alias ย่อคำสั่งให้ใช้ง่ายๆ	55
สถานการณ์ที่ 6 : Fast Forward กับ 3-way Merge	55
สถานการณ์ที่ 7 : จัดการ Branch	56
สถานการณ์ที่ 8 : แก้ไข Commit หรือ History	58
สถานการณ์ที่ 9 : ดู Difference	60
สถานการณ์ที่ 10 : งานหาย แก้ได้ด้วย Reflog	62
สถานการณ์ที่ 11 : Optimize	63
สถานการณ์ที่ 12 : Stash ซ่อนการแก้ไข	63

สถานการณ์ที่ 13 : ติด Tag ให้ Commit	65
สถานการณ์ที่ 14 : ignore ไฟล์	65
Workflow	66
Fork และ Pull Request	69
Chapter 5 : Git GUI	
ตัวอย่างการใช้	72
Chapter 6 : บรรณาธุกรรม	
บรรณาธุกรรม	76
คนเขียน	77

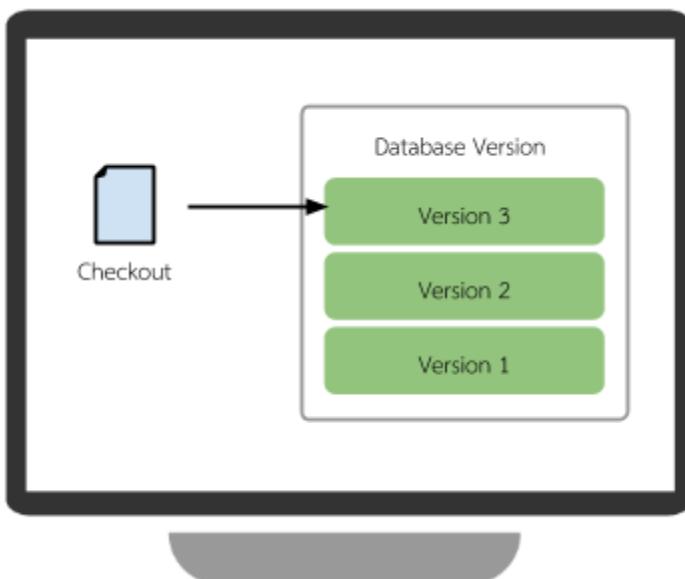
Chapter 1 : Introduction

Git เป็นหนึ่งในหลายเครื่องมือควบคุมต้นรหัสข้อมูล (Version control) ซึ่งทำหน้าที่จัดเก็บการเปลี่ยนแปลงข้อมูลในไฟล์ข้อมูล ทำให้ผู้ใช้งานสามารถเรียกข้อมูลก่อนที่มันจะถูกแก้ไขและหาทางไปต่อไม่ถูก กลับมาได้ในระดับไฟล์หรือรูปแบบแม่ไฟล์เดอร์ ยิ่งกว่านั้นมันยังสามารถระบุได้ว่าใครทำมันเงี้ยและทำไปตอนไหน ดังนั้น เราจะมาทำความรู้จักกับ Version control กันจะก่อน จะได้รู้ว่ามันคืออะไรบ้าง?

1.1 เครื่องมือควบคุมต้นรหัสข้อมูล (Version Control System : VCS)

อย่างที่บอกไปแล้วข้างต้น ว่า Version control (ซึ่อื่นเช่น Revision control, Source control) คือตัวควบคุมต้นรหัสข้อมูล ซึ่งข้อมูลในที่นี่หมายความรวมไปถึงทุกอย่างที่อยู่ในรูปแบบของรหัสคอมพิวเตอร์ เช่น ข้อความ รูปภาพ โปรแกรม เพลง (& Something like that) โดย Version control จะทำให้ที่เก็บข้อมูล เป็นการสำรองข้อมูลให้กับไฟล์หรือไฟล์เดอร์ที่เราต้องการ รวมทั้งบอกได้ว่าใครเป็นคนแก้ไข แก้ไขเวลาไหน รวมถึงความแตกต่างระหว่างก่อนแก้ไขและหลังแก้ไขไฟล์ ซึ่งคุณสมบัติเหล่านี้ทำให้เราไม่ต้องปวดหัวกับการต้องกอบปี้ไปไว้ในไฟล์เดอร์ Backup และเปลี่ยนชื่อเรียงตามวันที่ซึ่งเป็นวิธีที่เช่ากราวด์มากๆ (ข้อเสียคือเจ้านายจะรู้ว่าเราแอบอ้างงาน) ซึ่ง Version control ก็จะแบ่งเป็น 3 ประเภทให้เลือกใช้ คือ Local, Centralized และ Distributed

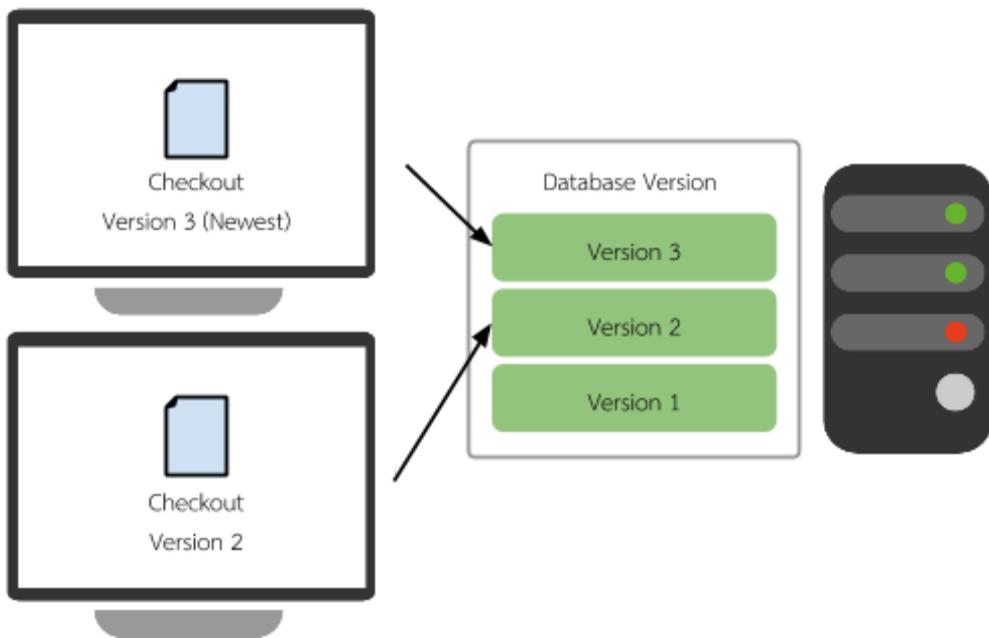
- Local Version Control System : LVCS



จากโปรแกรมเมอร์ยุคดึกดำบรรพ์ที่ต้องกอบปี้โค้ดไปไว้ในไฟล์เดอร์ Backup ก่อนที่จะแก้ไขไฟล์ ทำให้โปรแกรมเมอร์คิดว่ามันไม่น่าจะเวิร์คแล้ว จึงมีการพัฒนาฐานข้อมูลง่ายๆ เพื่อใช้เก็บข้อมูลในเครื่องของโปรแกรมเมอร์เอง ระบบที่ดังมากคือ Revision Control System : RCS ที่ริเริ่มพัฒนาโดย Walter F. Tichy โปรแกรมเมอร์ชาวเยอรมัน ระบบ RCS จะติดมากับ Developer Tools ของ Mac OS X เพื่อสำรองข้อมูลก่อนการอัพเดท Patch ซึ่งหากไม่สำเร็jmันก็จะเรียกข้อมูลก่อนหน้าซึ่งทำงานได้ปกติมาใช้

- Centralized Version Control System : CVCS

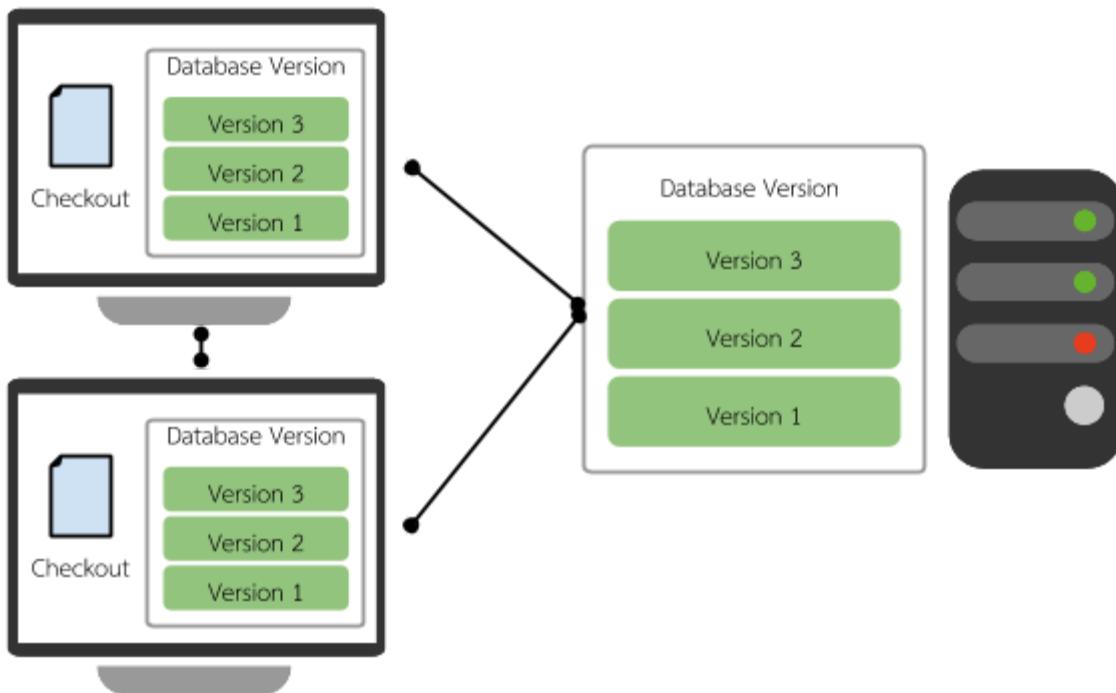
จากโปรแกรมเมอร์ในยุค Local Version Control ที่สามารถสำรองได้เฉพาะบนเครื่องตัวเองจึงหมายกับการทำงานแบบฉายเดี่ยวซึ่งมากกว่า แล้วถ้าเรามีทีมงานหลายคนที่ต้องช่วยกันรุ่มปูริ่งปูร์มาโปรเจกนี้ก็จะเกิดปัญหาเรื่องการใช้งานรวมไฟล์ร่วมกันขึ้น เพื่อแก้ปัญหาว่าถ้าเพื่อนเราแก้ไฟล์ เราไม่ต้องเอา USB Drive ไปเสียบก็อปข้อมูลอกมา เพราะกว่าจะครบถ้วนกันทุกคนก็แล้วกันหมดพอตี แต่เราจะใช้รีวิวให้คนที่ทำservice เลี้ยวส่งข้อมูลไปเก็บไว้ที่ Server กลาง ซึ่ง Server กลางตัวนี้ก็ทำหน้าที่สำรองข้อมูลไว้ด้วย ถ้าใครอยากรู้ด้วยกันก็มาดึงมาจาก Server ตัวนี้ได้ ซึ่งข้อมูลที่อยู่ใน Server กลางโดยทั่วไปแล้วจะเป็นรุ่นที่ใหม่ล่าสุดและทดสอบว่าใช้งานได้แล้ว (ควรทดสอบว่ามันรันได้จริงๆ ก่อนที่จะส่งข้อมูลขึ้นไป ไม่งั้นอาจจะโดนค่าผู้ปกครองเอาได้)



ระบบแบบนี้มีข้อดีคือการทำงานรวมศูนย์ไว้ที่เดียวกัน คนที่เป็น Project manager ก็ดูได้่ายว่าทำงานกันไปถึงไหนแล้ว ไฟล์ข้อมูลก็จะใหม่อยู่เสมอ และทุกคนสามารถมีไฟล์ข้อมูลที่เหมือนกันได้่าย เพียงแค่ให้คนที่ต้องการใช้ดึงไฟล์ข้อมูลจาก Server ไปไว้ในเครื่องของตัวเอง แต่ข้อเสียก็คือหาก Server มีปัญหาหรือเราไม่ได้เชื่อมต่อกับเครือข่าย จะไม่สามารถส่งข้อมูลขึ้นไปหรือดึงข้อมูลลงจาก Server ได้ และถ้า Harddisk พังก็จะมีไฟล์สำรองแค่ที่อยู่ในเครื่องแต่ละคนเท่านั้น (Server ข้อมูลหายแต่ Client ข้อมูลยังอยู่) ข้อมูลที่หายไปจาก Server นั้นรวมทั้ง Log ประวัติการแก้ไขไฟล์ต่างๆ ก็จะหายไปด้วย (มีเก็บแค่ใน Server เท่านั้นจ้า ถ้าข้อมูลที่ Server หายก็ตัวมัน) โดย Version control ที่ใช้การเก็บข้อมูลแบบ Centralized หรือรวมศูนย์นี้ก็เช่น CVS, SourceSafe, Subversion, Team Foundation Server

- Distributed Version Control System : DVCS

จากข้อเสียหลักๆของ Centralized Version Control System ที่ถ้าเกิดเครื่อง Server พังหรือหากเราเดินทางอยู่ไม่สามารถเข้ามายังต่อ กับเครือข่ายได้ ก็จะไม่สามารถทำงานได้ ดังนั้นจึงมีคนคิดค้นระบบ Distributed หรือแบบกระจายศูนย์ขึ้นมาแก้ปัญหา เพราะระบบนี้จะไม่ต้องหรือส่งเฉพาะไฟล์ข้อมูลจาก Server ไปเท่านั้น แต่จะก่อปื้นฐานข้อมูลรุ่นแบบเดียวกับที่มีอยู่ใน Server ไปด้วยเพื่อว่าถ้า Server มีอันเป็นไปหรือเรามิ่งสามารถเข้าต่อเครือข่ายกับ Server ได้เราถึงจะสามารถทำงานได้และยังสามารถเก็บรุ่นของไฟล์ พร้อมทั้งประวัติการแก้ไขและเวลาไว้เสร็จสรรพ โดยที่เราไม่จำเป็นต้องมี Server ด้วยซ้ำไป จึงทำให้เราไม่มีข้อหัวง่าว่าไฟล์หายก่อนส่งงานได้อีกต่อไป (เย้ T__T)



และระบบนี้สนับสนุนให้มีการทำงานแบบหลายทีมในโปรเจกเดียวกัน เช่น Team A เที่นงานฟีเจอร์ลับ ส่วน Team B ไม่เห็น หรือโปรแกรมเมอร์ก็ไม่ต้องรอให้กราฟิกดีไซน์วาดภาพเสร็จ แค่ร่างมาก่อนก็จะมีไฟล์ในระบบ โปรแกรมเมอร์ก็สามารถเรียกใช้ได้และหากกราฟิกดีไซน์แก้ไขภาพแล้วส่งขึ้น Server กลาง และเมื่อเราดึงข้อมูลใหม่ลงมาก็จะได้รูปที่แก้ไขแล้วได้เลยง่ายๆ และความเร็วเข้าถึงข้อมูลยังสูงเว่อร์ เพราะในขั้นตอนการทำงานปกติมักจะทำงานที่ฐานข้อมูลรุ่นภายนอกเครื่องเราเอง ต่างกับระบบ Centralized ที่ต้องวิ่งไปตามที่ Server ก่อน ตัวอย่างระบบที่ใช้งานแบบกระจายศูนย์ก็เช่น Git (พระเอกของเรา), BitKeeper(ตัวร้าย), Mercurial, Bazaar หรือ Darcs

1.2 ประวัติของ Git



Linus Benedict Torvalds

ss : wikipedia.org

กล่าวครั้งหนึ่งนานมาแล้วตอนพัฒนา Linux kernel ใหม่ๆ ประมาณปี ค.ศ. 1991-2002 Linus Benedict Torvalds คนต้นคิดสร้าง Linux kernel ก็ใช้วิธีส่ง Patch ไฟล์ ไม่จับก์ส่งไฟล์ซิป Source code กันไปมาระหว่างโปรแกรมเมอร์ (เราเรียกวิธีนี้ว่า Tarball) ใครแก้อะไรได้ก็ส่งมาให้ Linus เพิ่มใน Linux kernel รุ่นดังเดิมที่เขาสร้างขึ้นมาเพื่อให้คนหลังๆ ที่ต้องการใช้จะได้มามาโหลดรุ่นของ Linus ซึ่งได้รับการเพิ่มฟีเจอร์ หรือ Patch แก้บักไปแล้วเนี่ยแค่ศูนย์กลางเพียงที่เดียว แต่พอโปรเจกขยายตัวขึ้น Linus ก็ได้รับ Patch และ Source code เป็นร้อยเป็นพันชุด ซึ่งทำให้จัดการได้ยาก แฉมยังไม่รู้ว่า Patch ที่ส่งมาพอกaoไปรวมกับของที่มีอยู่แล้วมันจะยังทำงานได้ถูกต้องหรือเปล่า ดังนั้นในปี 2002 จึงได้เริ่มอา DVCS ที่ชื่อว่า BitKeeper ซึ่งเป็นทรัพย์สินทางปัญญาของบริษัท BitMover มาใช้ โดยตัว BitKeeper เองเป็น Commercial software แต่ Larry McVoy ซึ่งเป็น CEO ของ BitMover อนุญาตให้นักพัฒนา Open source สามารถใช้ได้ฟรี

แต่ใช้ไปได้ช่วงหนึ่งจนถึงปี 2005 ปัญหาเกิดขึ้นจาก Andrew Tridgell (มักจะถูกเรียกว่า "Tridge") คนสร้างระบบจัดการทรัพยากรในเครือข่าย Samba นั้นได้สร้างระบบติดต่อกับ BitKeeper ในส่วนที่เก็บข้อมูล ซึ่งเป็น Open source ขึ้นมาใหม่ชื่อว่า SourcePuller แทนที่จะใช้ BitKeeper client ซึ่งเป็นของเสียตังค์ (แต่ได้ใช้ฟรี เพราะ Larry McVoy อนุญาต) Tridge ถูกกล่าวหาว่าเขาใช้วิธี Reverse-engineered เพื่อดูการทำงานของ BitKeeper ดังนั้น Larry จึงโมโหมากว่า Tridge ไม่ได้จ่ายตังค์ซื้อซอฟแวร์ และครอที่ไม่ซื้อซอฟแวร์ก็ไม่มีสิทธิ์ที่จะได้ดู Metadata ของซื้อขาย จบป่ะ? Tridge ก็สวนกลับว่ามันไม่ได้ผิดกฎหมายที่จะดู Metadata เพราะ BitKeeper และ SourcePuller ต่างก็ใช้ฟรีเหมือนกันแต่สร้างขึ้นมาเป็นทางเลือกให้ผู้ใช้ แฉมไอ้ Metadata เนี่ยชาวบ้านชาวช่องหากไม่ได้ปิดกันหรอก บางที่สนับสนุนกันด้วยซ้ำและวิเคราะห์แค่ Telnet เข้าไป BitKeeper server แล้วสั่งคำสั่ง HELP เท่านั้นและหากไม่เคยใช้ BitKeeper client ด้วย ยังไงก็ยืนยัน นั่นยัน นอนยัน ตีลังกา yawn ว่าทำถูกกฎหมายและอยู่ในศีลธรรมอันดี ส่วนป่า Linux ก็ไม่เห็นด้วยกับการกระทำของ Tridge ถึงกับบอกว่าเป็นวิธีการที่ Tridge ทำเป็นวิธีที่สกปรก และก็ขอให้ Tridge หยุด แต่ตัว Tridge ตั้งเหตุก็ยืนกรานว่าสิ่งที่เขาทำถูกต้องแล้ว Larry จึงบอกว่าถ้า真ก็ไม่ต้องซั่งต้องใช้มันแหล่ ให้เวลา 3 เดือนในการเอาไปประกอบก่อไปจาก



Git logo

ss : git-scm.com

BitKeeper server จะถึงตอนนี้ใครที่ด่า Tridge อยู่ก็ต้องคิดถึงในแง่อื่นด้วยนะครับ Linus เป็นคนเริ่มก่อจิริแรกที่ต้องฟังเสียงของ Community ด้วยเพราะการจะพัฒนาของที่เป็น Open source จากของที่ติด License อื่นเนี่ย Community ไม่ค่อยเห็นด้วย (ถึงมันจะเป็น Tools ที่ดีมากๆ อย่าง BitKeeper ก็ตาม)

Linus บอกว่าเขามักตั้งชื่อ Project ตามสิ่งที่เขาเป็นเหมือนกับชื่อระบบปฏิบัติการ Linux และ Git ก็เป็นศักดิ์และลงที่หมายความว่า “คนใจกว้างแจ้ง”, “คนที่ไม่เป็นที่น่าพอใจ”, “คนน่ารังเกียจ” หรือ “พวกรุนทดีกว่าตัวเองถูกอยู่ตลอดเวลา” ส่วนคำว่า Git ย่อมาจากอะไรนั้นขึ้นอยู่กับอารมณ์มันจะย่อมาจาก “Global information tracker” ตอนที่อารมณ์ดีๆ หรือ "Goddamn idiotic truckload of sh*t" ตอนที่มันเจ็บ (แนวประชดกล้ายๆ)

Linus จึงมองหา Version control ที่จะใช้งานที่มีให้ใช้ได้ฟรีๆ แต่ก็ไม่เห็นที่เหมาะสมกับ Linux kernel เลย

จำเป็นต้องสร้าง Version control ขึ้นมาเองโดยใช้ประสบการณ์จาก BitKeeper มาปรับปรุงใน Version control ตัวใหม่ซึ่งต้องมีคุณสมบัติดังนี้

- ความเร็วสูง
- สนับสนุนการแยกพัฒนาหลายทาง (แต่ branch ได้หลักพันเป็นอย่างน้อย)
- ทำงานแบบ Distributed Version Control System
- ทำงานกับโปรเจกใหญ่ๆได้ดีเช่น Linux kernel (ห้องความเร็วและปริมาณขนาดข้อมูล)

โปรเจกเริ่มสร้าง Git เริ่มเมื่อ 3 เมษายน 2005 พัฒนาโดยภาษา C, Shell script และ Perl, ประจำวันที่ 6 เมษายนให้ Git เป็นโปรเจก Open source โดยใช้สัญญาอนุญาตแบบ GNU GPL version 2 หลังจากนั้นเริ่ม Self-hosting (ใช้ตัว Git เองดูแล Source code ของ Git เอง) ในวันที่ 7 เมษายน, สามารถรวม Branch ที่แตกออกมากได้ในวันที่ 18 เมษายน และได้ประสิทธิภาพตามที่คาดหวังในวันที่ 29 เมษายน (ทำ Patch Linux kernel Tree ได้ในเวลา 6.7 วินาที) หลังจากนั้นในวันที่ 26 กรกฎาคม 2005 Linus ให้ Junio Hamano เป็นคนดูแล Git ต่อ

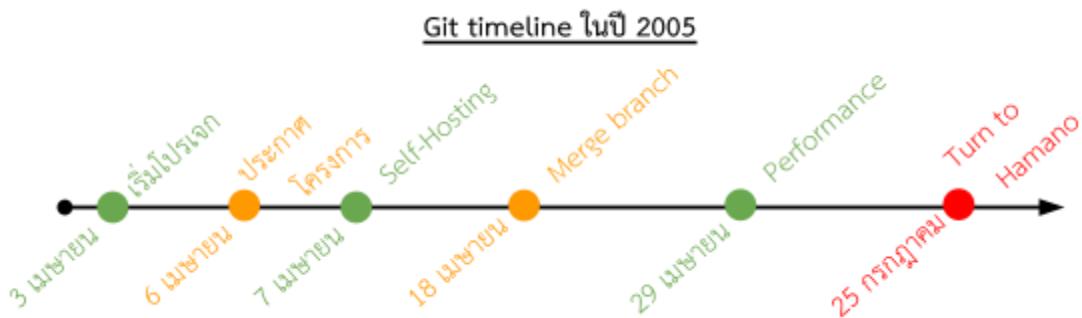
```

--- hello.c      2014-04-29 17:59:49.000000000 +0530
+++ hello-new.c 2014-04-29 18:00:43.000000000 +0530
@@ -1,5 +1,6 @@
 #include<stdio.h>

-main(){
+int main(void){
        printf("Hello, world!\n");
+
        return 0;
}

```

ตัวอย่าง Patch ไฟล์



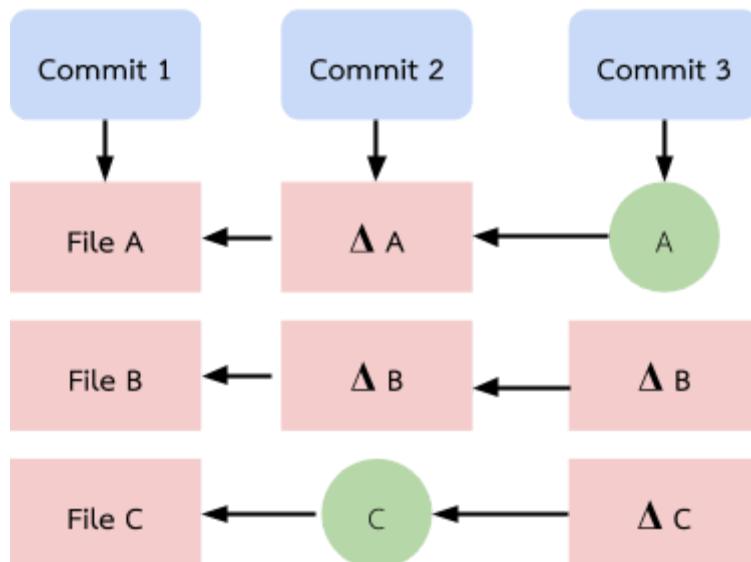
1.3 การทำงานของ Git

ถึงแม้ว่า Version control จะมีหน้าที่เหมือนกันแต่ยังไม่ได้ทำงานเหมือนกันทั้งหมดทุกด้าน (ไม่จับเข้าจะมีหลายตัวทำอะไรไว้) และตัว Version control มักจะเก็บข้อมูลแบบเพิ่มพูน (ไม่ลบของที่เก็บไว้แล้วแต่เพิ่มของใหม่เข้าไปเรื่อยๆแทน) ดังนั้น Git ก็จะโตขึ้นเรื่อยๆ

- การเก็บข้อมูล

ใน Version control อื่นๆที่ทำงานคล้ายกันเช่น CVS, Subversion, Perforce, Bazaar จะมีการเก็บข้อมูลเป็นแบบความแตกต่างกันของข้อมูลในไฟล์ ดังนั้นการเก็บข้อมูลแบบนี้ต้องเก็บข้อมูลก่อนหน้านั้นเอาไว้ตามลำดับด้วยถึงจะทำงานได้ถูกต้อง เรียกว่า Delta Storage (Δ คือค่าผลต่าง หรือ Differences ของไฟล์นั้นแหล่ง)

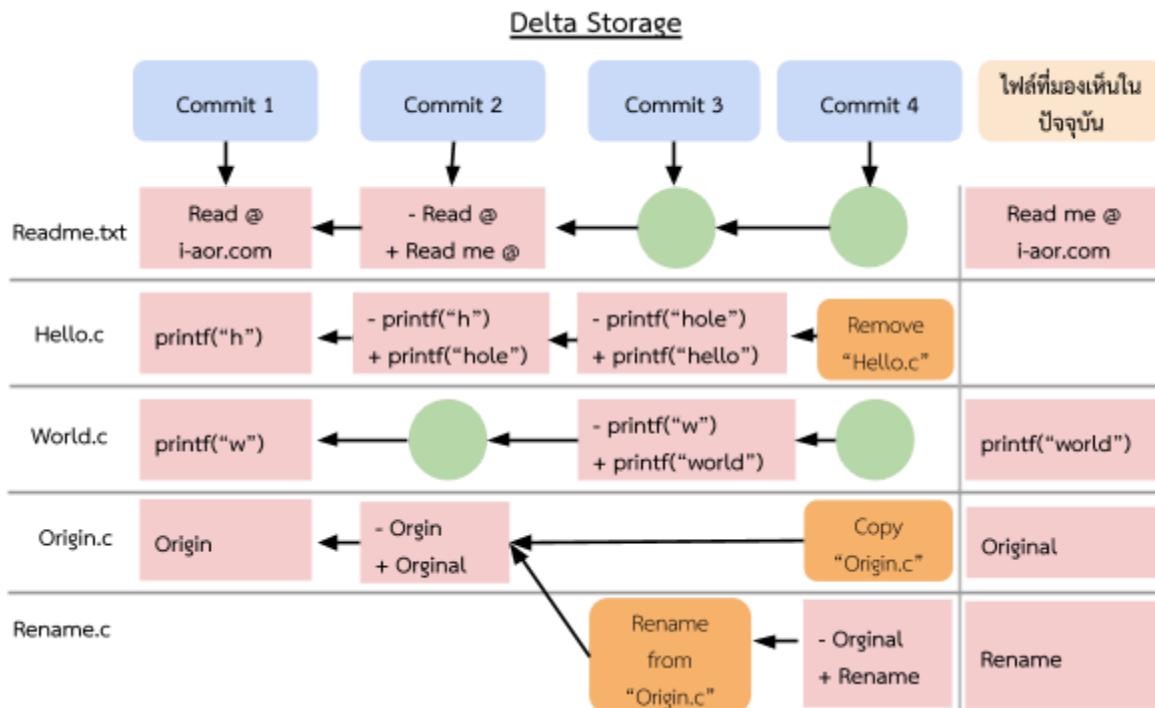
Delta Storage



การเก็บข้อมูลแบบนี้ถ้ามีไฟล์อยู่ 3 ไฟล์ตามรูปด้านบน เมื่อเรา Commit (สร้าง Version นั้นเอง) ใน

- Commit 1 จะเก็บข้อมูลทั้งหมดในไฟล์ A, ไฟล์ B และไฟล์ C ไว้
- Commit 2 จะเก็บข้อมูลที่แตกต่างกันเท่านั้น (เพราะมีไฟล์เดิมอยู่แล้ว) หากเรามีการแก้ไฟล์ข้อมูลที่แตกต่างก็คือ ΔA และ ΔB ส่วนไฟล์ C นั้นไม่มีการแก้ไขจึงไม่มี Δ หรือข้อมูลที่แตกต่างกัน ดังนั้น ไฟล์ C (วงกลมเขียว C) จึงจะเป็น Pointer ซึ่งลับไปที่ไฟล์ C ตัวหลัก ใน Commit 1
- Commit 3 ไฟล์ A ไม่มีการแก้ไข ข้อมูลก็ยังเหมือนเดิม ดังนั้นจึงมี Pointer ซึ่งลับไปที่ Commit ก่อนหน้า ซึ่งก็คือ Commit 2 ส่วนไฟล์ B มีการเปลี่ยนแปลงก็ทำงานตามปกติ และสุดท้ายไฟล์ C นั้นจากเดิม Commit 2 ไม่มีการแก้ไข แต่ใน Commit 3 มีการแก้ไขเป็น ΔC จึงมี Pointer ซึ่งลับไปที่ Commit ก่อนหน้า (ก็ Commit 2 นั่นแหล่ง) แต่ที่ Commit 2 บอกว่าที่มันไม่มีอะไร ไม่ได้แกะอะไร จึงก็ไปตามที่ Pointer ต่อไปเลย ดังนั้น ΔC จะมีข้อมูลก่อนหน้าคือไฟล์ C ใน Commit 1 เพราะเป็นไฟล์ที่ C วงกลมเขียวซึ่ง Pointer ไป

หากมีการดำเนินการแก้ไขไฟล์แบบพิเศษ เช่นแก้ไขชื่อไฟล์ ลบไฟล์หรือก็อปปี้ใน Commit ไหน ตัว Commit นั้นก็จะมีอ้างอิงไปไฟล์ก่อนหน้าด้วยค่าคำสั่งพิเศษที่บอกความแตกต่างกับระหว่างตัวที่เรากำลัง Commit และ Commit ก่อนหน้านั้น



(Note : ใน Commit 3 Origin.c ถูก Rename ไปแล้ว แต่ใน Commit 4 ก็อปปี้มาเป็นอีกไฟล์เลยมองเห็นในตำแหน่งปัจจุบัน)

นอกจากการเก็บไฟล์แบบ Delta storage แล้วก็ยังมีการเก็บไฟล์อีกแบบหนึ่งที่เรียกว่า Snapshot

storage การเก็บไฟล์ของระบบ Snapshot storage มองไฟล์ที่ถูกแก้ไขเป็น Snapshot ใหม่ (แบบ Delta มองว่า เก็บไว้แค่ความเปลี่ยนแปลง) ดังนั้นใน Snapshot storage ข้อมูลก็คล้ายๆเป็นไฟล์อีกไฟล์หนึ่งเลย เวลาต้องการ Revision กลับไปรุ่นก่อนระบบก็จะไปหา Snapshot หรือ Snapshot ที่ถูก References ไป เพราะไม่ได้ถูกแก้ใน Commit นั้น แต่ในส่วนของ Delta storage แบ่งการย้อนเวอร์ชันเป็นสองส่วนคือแบบ Reverse delta และ Forward delta ตัวอย่างนี้สมมุติว่ามีไฟล์อยู่ 4 เวอร์ชัน และกำลังจะทำเวอร์ชันที่ 5 (ยังไม่ได้ Commit) แต่อยากกลับไปเวอร์ชัน 2

Reverse

$$(\text{File } \text{ปัจจุบัน} - 4) - \Delta_4 - \Delta_3$$

$$\Delta_n = n - (n-1), \quad \Delta_n \text{ คือ } n \text{ ที่ต่างกัน } (n-1) \text{ ดังนั้น } \Delta_4 \text{ คือ ส่วนต่างระหว่างรุ่น } 4 \text{ กับ } 3$$

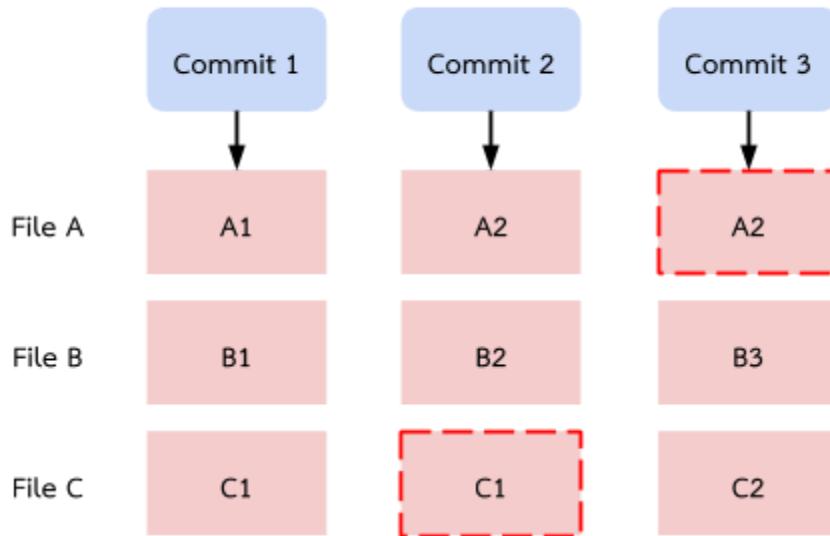
Forward

$$(\text{File } \text{ตั้งต้น} + \Delta_1) + \Delta_2$$

Reverse เป็นการถอยจากตำแหน่ง File ปัจจุบันที่เราทำงานอยู่ว่าต่างจากเวอร์ชันก่อนหน้า

Forward เป็นการเริ่มวิ่งจาก File ตั้งต้น เพิ่ม Delta ในแต่ละรุ่นไปจนถึงรุ่นที่ต้องการ

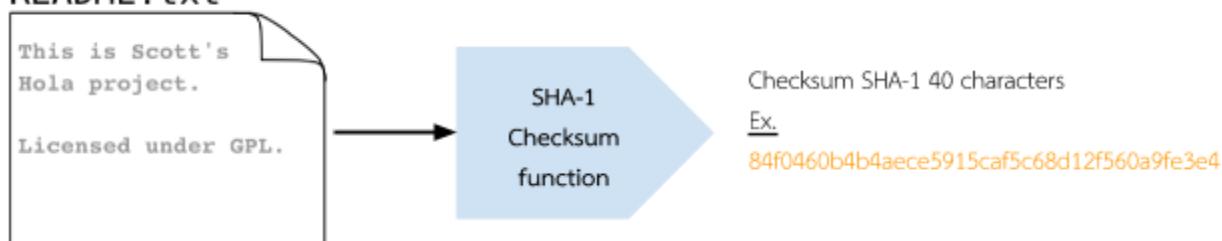
Snapshot Storage



จากรูปจะเห็นว่า

- ในรูปมี 3 ไฟล์ คือ ไฟล์ A , ไฟล์ B และไฟล์ C
- Commit 1 จะเป็น Commit เริ่มต้นเก็บข้อมูลในไฟล์เป็นรุ่นแรก ซึ่งข้อมูลที่เก็บไว้เราจะเรียกว่า Snapshot ซึ่ง เมื่อนำมาพ่อๆ ที่มีข้อมูลในไฟล์ที่แก้ไขแล้วจะถูกถอดโดยที่ไม่ต้องอ้างอิง Snapshot หรือไฟล์ก่อนหน้านั้น ก็คือ A1,B1 และ C1
- Commit 2 จะเห็นว่าถ้ามีการแก้ไขไฟล์ A และ ไฟล์ B ระบบจะเก็บ Snapshot ไว้ทั้งไฟล์ไว้ จึงได้ A2 และ B2 ส่วนไฟล์ C ที่ไม่มีการแก้ไขก็จะมี Reference ชี้ไปที่ Snapshot ของ C1
- Commit 3 ไฟล์ A ไม่มีการแก้ไขดังนั้นจึงมี Reference ชี้ไปที่ Snapshot ของ A2 ซึ่งเป็นตัว Snapshot ล่าสุด โดยไม่ต้องวิ่งไปถึง A1 ส่วนไฟล์ B และไฟล์ C ถูกแก้ไข เลยได้ Snapshot ตัวใหม่คือ B3 และ C2 ขึ้นมา
- ถึงตอนนี้เราเก็บรายละเอียดของการเก็บข้อมูลแบ่งออกเป็น 2 แบบใหญ่ๆ คือ Delta storage และ Snapshot storage ระบบ Git จะใช้งาน Storage แบบ Snapshot เป็นหลัก (แต่ Git ก็มีบางคำสั่งเช่น git pack เพื่อลดขนาดไฟล์หน้างานกับการเก็บไว้ในระบบสำรอง ซึ่งเป็นการเก็บไฟล์แบบ Delta ได้)

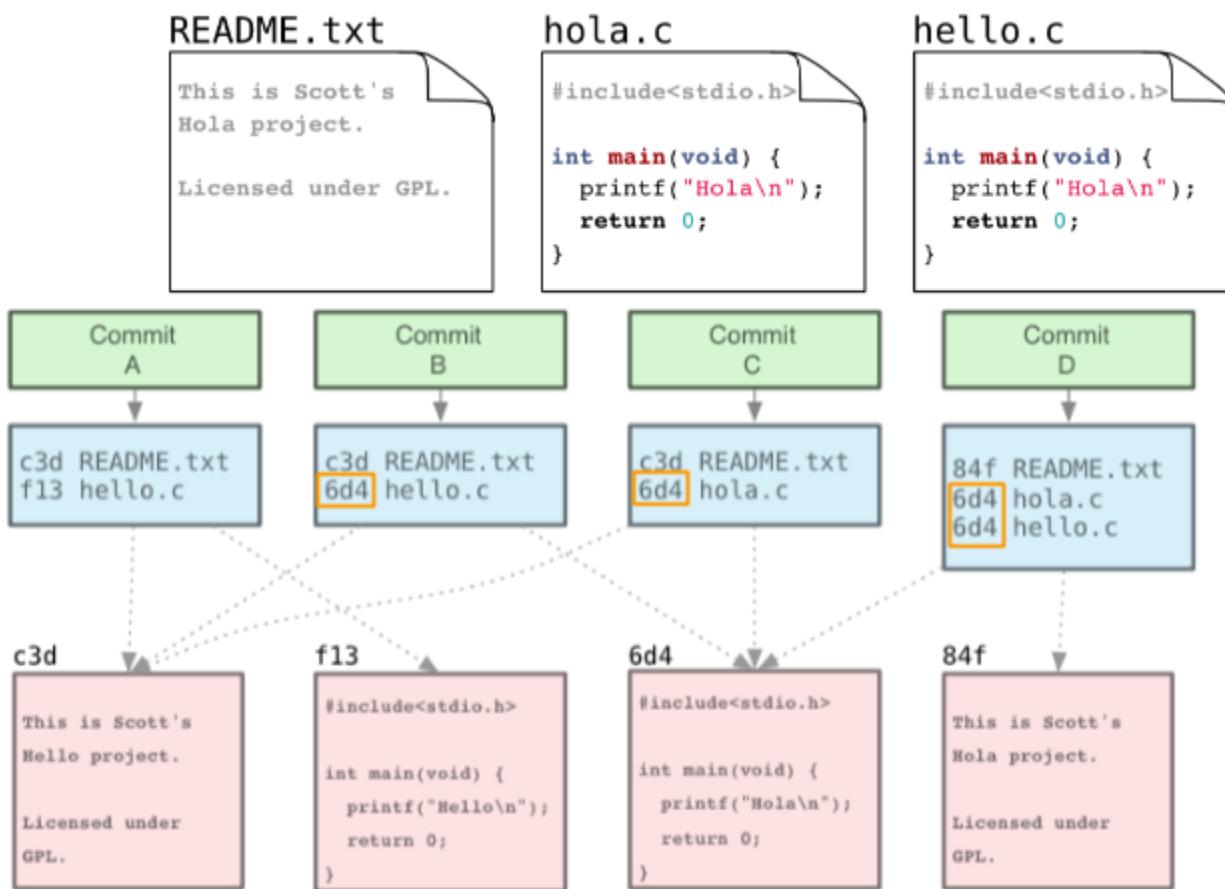
README.txt



ระบบ Git Snapshot จะนำไฟล์ไปเข้าฟังก์ชัน Hash ที่ชื่อว่า SHA-1 (ชื่อเล่นว่าชาลวัม) ขนาด 160 Bits ซึ่งตัว SHA-1 จะนำข้อมูลไปบี้ยปูมีแล้วให้ค่าอ กมาค่าหนึ่งความยาว 40 ตัวอักษร (0-9 และ a-f) เรียกว่า Checksum ซึ่งเมื่อมีข้อมูลเปลี่ยนไป ค่า Checksum ของมันก็จะเปลี่ยนตามไปด้วย ถึงแม้ข้อมูลจะต่างกันแค่บิทเดียว Checksum ก็จะสามารถระบุความแตกต่างได้ ดังนั้นมันจึงใช้อาไวมากกว่าไฟล์มีการเปลี่ยนแปลงหรือไม่และยืนยันว่าข้อมูลไม่เกิดความผิดพลาด แต่เมื่อนานมานั่นก็ทำให้เราทราบว่า SHA-1 ไม่ปลอดภัยแล้วครับ

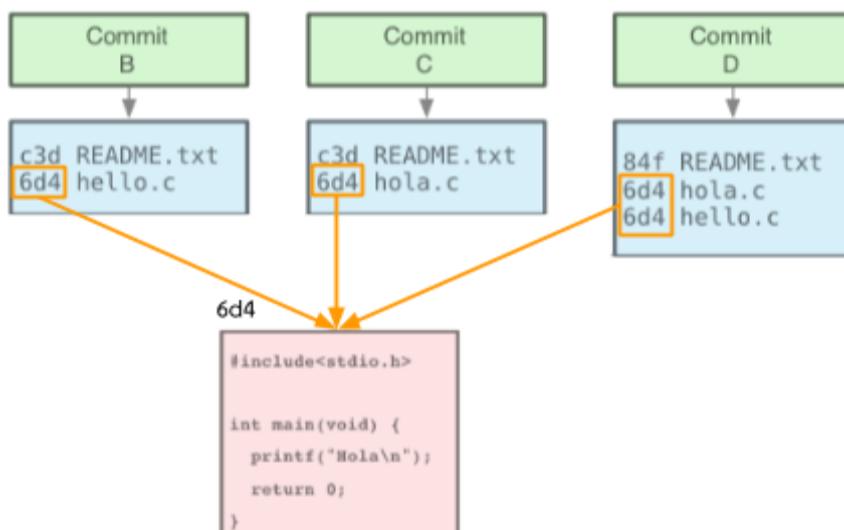
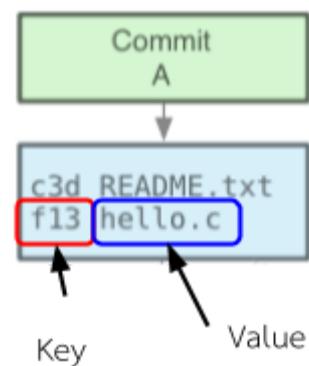


ล้อเล่นครับ มันเกิดจากการที่นักวิจัยพบว่าแม้มันจะมี 160 Bits แต่ความปลอดภัยที่แข็งแกร่งมีเพียง 60 Bits



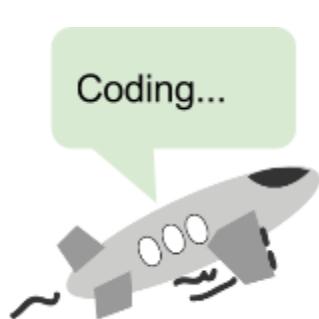
Git snapshot
cc : Introduction to Git with Scott Chacon of GitHub

จะเห็นว่าในแต่ละ Git Snapshot จะมีเลขกำกับไว้ 40 ตัวอักษรซึ่งเป็นค่าของ Checksum แต่ละ Snapshot ที่เห็นมีแค่ 3 ตัวเช่น c3d, f13, 6d4 และ 84f ก็อย่าเพิ่งgalssing เขาย่อเหลือ 3 ตัวแรกเฉยๆ ให้ 40 ตัวมามันยะ燥ไป (แค่ 3 ตัวพอแล้วไม่ซ้ำกันแล้ว) และสิ่งที่ Git Commit บอกในตอน Commit ก็คือคู่ Key-Value ของแต่ละไฟล์โดย Key ก็จะชี้ไปที่ Snapshot ที่เราบันทึกไว้ ส่วน Value ก็จะเป็นชื่อไฟล์ และในกรณีที่มีการเปลี่ยนชื่อไฟล์ หรือทำก็อปปี้ไฟล์แล้วเปลี่ยนชื่อ Git Commit จะมองว่ามันมีหลายไฟล์ก็จริง แต่ตัวอ้างอิง Snapshot จะชี้ไปที่ Snapshot เดียวกัน



ไม่ว่าจะ Delta หรือ Snapshot มองการเก็บข้อมูลแบบ Conceptual หรือการมองคอนเซปเท่านั้น การเก็บข้อมูลของ Git จริงๆแล้วก็ใช้งานแบบ Delta เมื่อนอกัน

- การทำงานเก็บทั้งหมดอยู่ในเครื่องเราเอง



เป็นข้อได้เปรียบของ Distributed VCS เพราะเครื่องเราเปรียบเสมือน Server ตัวหนึ่ง เวลาเราต้องการดู Log หรือ Commit ก็สามารถดูได้เลยไม่จำเป็นต้องใช้งานเครือข่ายเพื่อลิงก์ไปที่ Server ซึ่งพาก Subversion หรือ Version control ที่ใช้ระบบ Centralized VCS สามารถแก้ไขไฟล์ในเครื่องได้ก็จริง แต่ไม่สามารถดู Log หรือ Commit ไปที่ Server ได้หากขาดการเชื่อมต่อเครือข่าย ทำให้ไม่มีข้อมูลสำรองเอาไว้ในเครื่อง แต่ Git สามารถ Coding ระหว่างการเดินทางได้สบายๆ แต่ก็ต้องการใช้เครือข่ายหากเราต้องการส่งหรือดึงข้อมูลจาก Server

- เพิ่มข้อมูลเสมอ

การทำงานของ Git เมื่อไฟล์ถูกแก้แล้วเราทำการ Add และ Commit แล้วมันจะเป็นการเพิ่มข้อมูลไปเรื่อยๆ การลบข้อมูลยากมากเกือบจะเป็นไปไม่ได้เลย ซึ่ง Git มองการแก้ไขไฟล์เป็น Snapshot ใหม่ๆ ไปเรื่อยๆ ไม่ลบของเดิมทิ้งเว้นแต่ว่ามันจะเป็นของที่เรามีได้ใช้

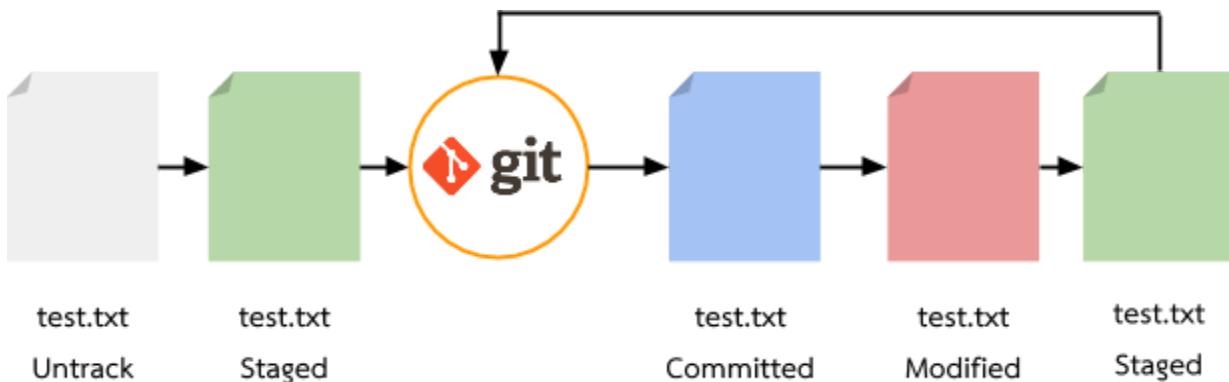


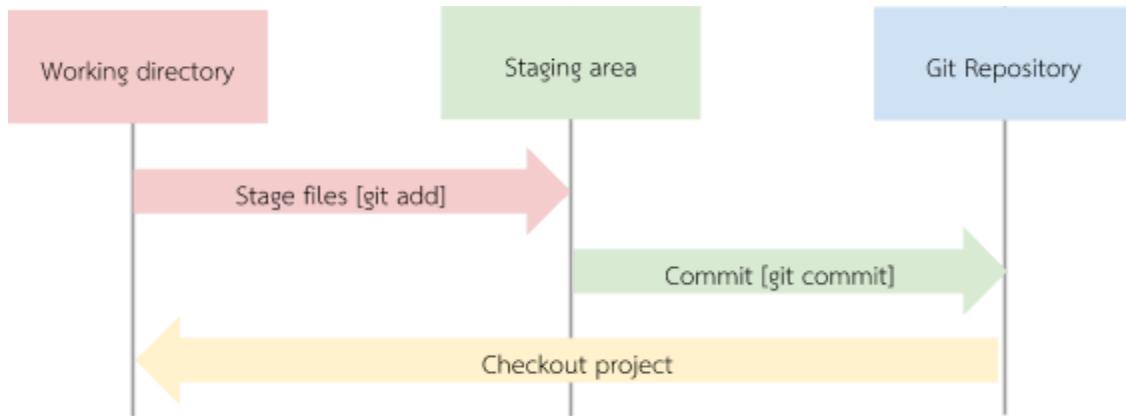
- สถานะของไฟล์ข้อมูล



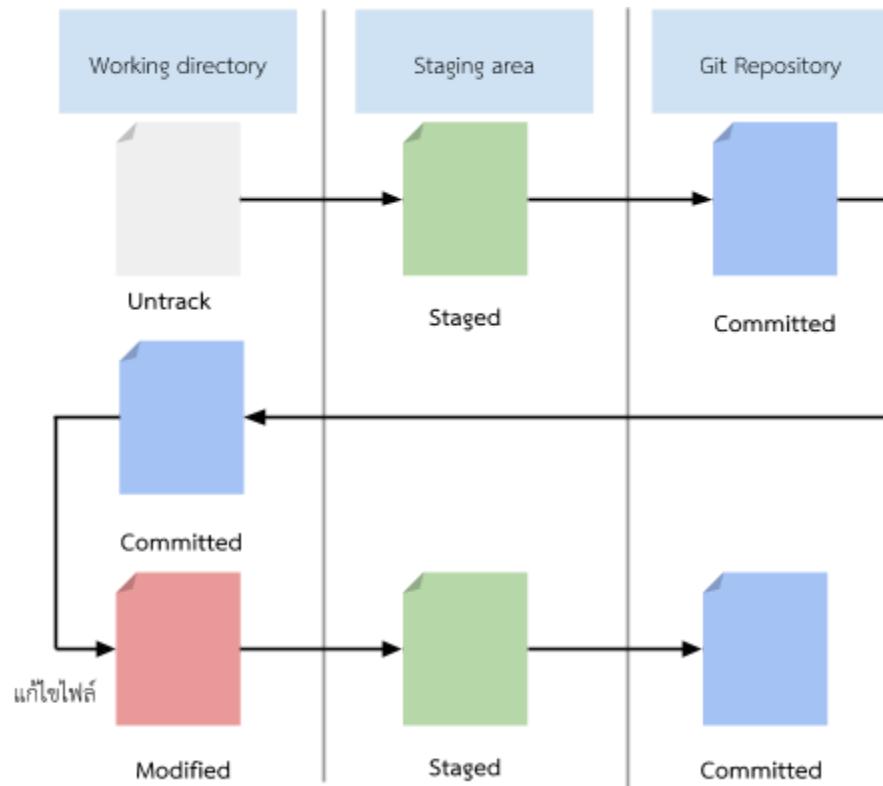
สถานะของไฟล์ในนั้นมี 4 สถานะคือ

- Untrack เป็นข้อมูลที่ถูกสร้างขึ้นใหม่ยังไม่ได้ส่งเข้าเป็นไฟล์ของ Git หรือไฟล์ที่เราไม่ต้องการให้ Git นำไปเก็บ สำรองไว้ เช่นไฟล์ที่เป็น Stack trace, ไฟล์ที่เอาไว้ติดตามการทำงาน หรือ Error เราอาจจะไม่ Track มัน
- Committed คือไฟล์ข้อมูลที่ถูก Commit เข้าในฐานข้อมูลไปแล้วจากนั้น Git จะดึงสถานะไฟล์กลับมาเพื่อรอแก้ไขอีกรอบ แต่ระหว่างที่มันยังไม่ได้แก้ไขในรอบใหม่นี้มันจะค้างอยู่ในสถานะ Committed
- Modified คือไฟล์ข้อมูลที่มีอยู่ในฐานข้อมูลของ Git และถูกแก้ไขแล้ว แต่ยังไม่ได้ Staged ไฟล์ และ Commit ไปไว้ในฐานข้อมูลของ Git อีกรอบหนึ่ง
- Staged คือไฟล์ข้อมูลที่อยู่ในสถานะเตรียม Commit โดยไฟล์ไหนที่อยู่ในสถานะ Untrack หรือ Modified แล้วต้องการเอาไปเก็บไว้ในฐานข้อมูลของ Git ต้องเปลี่ยนให้อยู่ในสถานะ Staged ก่อน ทำให้เลือกได้ว่าไฟล์ไหนจะถูก Commit บ้าง เพราะบางทีอาจจะไม่ต้องการเก็บไฟล์บางไฟล์ไป (แม้ว่าจะแก้ไฟล์ไปแล้ว เช่นไฟล์ Config ใช้เฉพาะเครื่องเรา ก็ไป Commit ไป)





จากขั้นตอนการทำงานกับไฟล์ข้อมูลของ Git จะเห็นว่าหากมีไฟล์เพิ่มเข้ามา ใน Working directory ที่ (ไฟล์เดอร์ที่เรากำลังทำงานอยู่เนี่ยแหละ) จะต้องเพิ่มเข้าในพื้นที่ Staging area (คำสั่ง git add) เพื่อจะบอกว่าไฟล์ใดบ้างที่เราต้องการเอาเข้าใน Git Repository จากนั้น Git จะสร้าง Snapshot ไว้และตอนที่เรา Commit (คำสั่ง git commit) เพื่อให้เอา Snapshot ไฟล์จาก Staging area ไปเก็บในฐานข้อมูลของ Git และเปลี่ยนสถานะไฟล์เป็น Committed ซึ่งการทำงานของ Git จะต่างกับการทำงานของ Version control อื่น ซึ่งหมายความว่าจะมีแค่การ Commit ไฟล์เท่านั้น (ไม่มีสถานะ Staging แต่ Commit จาก Working directory เลย) ข้อดีของการ Commit สองต่อแบบนี้ก็คือถ้าเรากำลังแก้บางไฟล์แต่ไม่เสร็จก็ยังสามารถ สร้าง Snapshot ไฟล์เก็บไว้ใน Staging area ก่อนได้



- Git object model

อย่างที่เคยบอกไว้ว่าในการเก็บประวัติไฟล์ไว้ Git จะนำไฟล์ไปเข้าฟังก์ชัน Hash ด้วย SHA-1 จำนวน 40 ตัวอักษร เพื่อใช้เป็น Object แต่ละตัว เช่น

6ff87c4664981e4397625791c8ea3bbb5f2279a3

ในทุกๆ Object จะประกอบไปด้วย 3 ส่วนคือ Type ซึ่งใช้บอกว่า Object นั้นเป็นประเภทไหน, Size ใช้บอกขนาดของ Object และ Content ที่ใช้บอกข้อมูลต่าง โดย Type หรือประเภทของ Object ก็มี 4 ประเภท คือ Blob,Tree,Commit และ Tag ซึ่งการเก็บข้อมูลแบบนี้จะเป็นแค่การย้าย Pointer ที่ชี้ไป Object ต่างๆทำให้ Git ทำงานได้เร็วสูงมาก

- Blob มาจาก Binary Large Object เป็นไฟล์ที่ใช้ในการเก็บข้อมูลไฟล์ทั่วไปในลักษณะของbinary file ใน Object แบบนี้จะไม่มีเวลา ชื่อไฟล์ และ Metadata อื่น

5b1d3..

blob	size
#ifndef REVISION_H #define REVISION_H #include "parse-options.h" #define SEEN (1u<<0) #define UNINTERESTING (1u<<1) #define TREESAME (1u<<2)	

```
$ git show 6ff87c4664
Note that the only valid version of the GPL as far as this project
is concerned is _this_ particular version of the license (ie v2, not
v2.2 or v3.x or whatever), unless explicitly otherwise stated.
...

```

- Tree ลักษณะคล้ายกราฟที่เอาไว้เก็บ Directory ย่อยๆและอ้างอิงถึง Blob ที่จะใช้กว่า Snapshot นั้นที่มีค่า SHA-1 ค่าใด มีชื่อไฟล์ว่าอะไร (คล้ายกับไฟล์เดอร์ที่เอาไว้เก็บไฟล์หรือไฟล์เดอร์ย่อยๆได้อีก) Tree จะบอกเราว่า ต้องเดินไปหา Snapshot ไหนบ้างถึงจะทำให้เราถูกข้อมูลคืนมาได้ถูกต้อง

c36d4..

tree		
blob	5b1d3	README
tree	03e78	lib
tree	cdc8b	test
blob	cba0a	test.rb
blob	911e7	xdiff

\$ git ls-tree fb3a8bdd0ce		
100644 blob 63c918c667fa005ff12ad89437f2fdc80926e21c	.gitignore	
100644 blob 5529b198e8d14decbe4ad99db3f7fb632de8439d	.mailmap	
100644 blob 6ff87c4664981e4397625791c8ea3bbb5f2279a3	COPYING	
040000 tree 2fb783e477100ce076f6bf57e4a6f026013dc745	Documentation	
100755 blob 3c0032cec592a765692234f1cba47dfdcc3a9200	GIT-VERSION-GEN	
100644 blob 289b046a443c0647624607d471289b2c7dc470b	INSTALL	
100644 blob 4eb463797adc693dc168b926b6932ff53f17d0b1	Makefile	
100644 blob 548142c327a6790ff8821d67c2ee1eff7a656b52	README	
...		

Mode Type Block reference

ไฟล์ Makefile มี SHA-1 Hash คือ 4eb463797...

- Commit เก็บข้อมูลที่ใช้ในการอ้างอิง Tree และข้อมูลประวัติของ Commit นั้นๆ

```
$ git show -s --pretty=raw 2be7fcb476
commit 2be7fcb4764f2dbcee52635b91fedb1b3dcf7ab4
tree fb3a8bdd0cedd019615af4d57a53f43d8cee2bf
parent 257a84d9d02e90447b149af58b271c19405edb6a
author Dave Watson <dwatson@mimvista.com> 1187576872 -0400
committer Junio C Hamano <gitster@pobox.com> 1187591163 -0700

Fix misspelling of 'suppress' in docs

Signed-off-by: Junio C Hamano <gitster@pobox.com>
```

ae668..

commit	size
tree	c4ec5
parent	a149e
author	Scott
committer	Scott
my commit message goes here and it is really, really cool	

- + Tree บอกว่ามันอ้างอิงไปที่ Tree ไหน จะได้ตามไฟล์ไปได้ถูก
- + Parent บอก Commit ก่อนหน้ามัน
- + Author บอกคนที่ทำหน้าที่รับผิดชอบการเปลี่ยนแปลง
- + Committer คือชื่อคนที่ Commit เข้ามาในระบบ

Author กับ Committer ต่างกันคือ
บางโปรเจกไม่ใช่ทุกคนที่จะ Commit
เข้ามาได้ ต้องเป็นหัวหน้าทีมซึ่งเป็นต้น
ซึ่งหัวหน้าทีมก็จะริวิวโค้ดก่อน

Object Model

98ca9..

commit	size
tree	0de24
parent	nil
author	Scott
committer	Scott
my commit message goes here and it is really, really cool	

0de24..

tree	size
blob	e8455
tree	10af9
README	lib

e8455..

blob	size
— LICENSE:	
(The MIT License)	
Copyright (c) 2007 Tom Preston-Werner	
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, under the terms of the License.	

10af9..

tree	size
blob	bc52a
mylib.rb	
tree	b70f8
inc	

bc52a..

blob	size
require 'grit/index'	
require 'grit/status'	
module Grit	
class << self	
attr_accessor :debug	

b70f8..

tree	size
blob	0ad1a
tricks.rb	

0ad1a..

blob	size
require 'grit/git-ruby/repo'	
require 'grit/git-ruby/file_i'	
module Grit	
module Tricks	

Flow การทำงานของ Commit ใน Git (cc: schacon.github.io) Git commit จะอ้างถึง Tree จากนั้น Tree จะอ้างถึง Tree ขั้นย่อยๆ ลงไป หรือไฟล์ Blob ที่เก็บ Snapshot ไว้ ทำให้บอกได้ว่าในลำดับการเรียก Commit นั้นไฟล์ไหนเปลี่ยนบ้าง

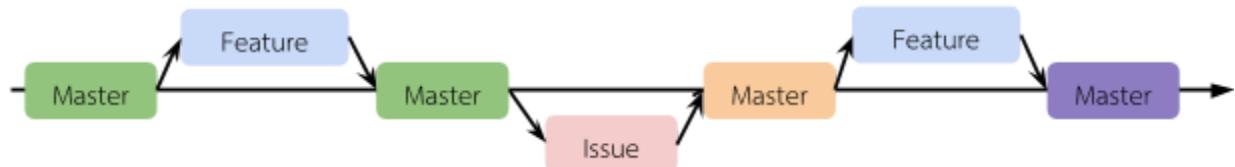
- Tag เป็นตัว Commit แบบพิเศษ ใช้ชี้แหล่งเก็บข้อมูล Digital signature หรือใช้ในการเป็น Tag ที่ช่วยในการย้อนกลับไปที่ Object ก็ได้ เช่น เรากำลังพัฒนา Software และมี Commit เยอะๆ เพราะพัฒนามานานแล้ว เราจะสามารถ Tag ได้ว่า Commit นี้เป็น Release รุ่น V1.0 ได้ เวลาจะหา ก็ไม่ต้องรื้อหา Commit แค่ใช้ Tag ไปเรียก ก็ได้

49e11..	
tag	size
object	ae668
type	commit
tagger	Scott
my tag message that explains this tag	

- สนับสนุนการทำงานแบบคู่ขนาน

Git สนับสนุนการแยก Branch ทำให้เราพัฒนาไฟล์หรือใหม่ๆ พร้อมๆ กับแก้ไขปัญหานี้ พร้อมกัน หรือระหว่างที่เรากำลังแก้ไขลูกค้าหรือหัวหน้าที่ต้องการดูข้อมูลก็สามารถ Checkout ออกมาให้ได้เลย

Branch timeline



จากรูปจะเห็นว่าปกติเราอยู่ที่ Branch ชื่อ **Master** ถ้ามี **Feature** เราจะแตก Branch แล้วไปพัฒนาที่ **Feature** ระหว่างนั้นพอดีเราเจอปัญหาอะไรบางอย่างที่ส่งผลกับ Software ที่เรากำลังพัฒนา เราจะเลยต้องการแก้ไขมันก่อน จึงแตก Branch **Issue** ออกมาจาก **Master** แก้แล้วก็ Merge รวมกลับมายัง **Master** ตั้งนั้น **Master** ตัวนี้จะลูกแกะบักไปแล้ว จากนั้นก็พัฒนา **Feature** ต่อไปจากของเดิมที่ทำค้างไว้ หลังจากเสร็จแล้วเลย Merge กลับ **Master**

Master ควรจะเป็น Branch ที่ได้ที่ทำงานได้เสมอ

เพราะถ้าเจ้านายหรือลูกค้ามาดูเรา ก็จะสามารถเปิดให้ดูได้ทันทีว่าระบบทำงานได้ดังนั้นเรามีเครื่องแก้ไขได้ใน Branch นี้ แต่ควรใช้การ Merge รวมกับ Branch ใดดีที่แก้ไขชุดใหม่แทน เพราะจะอำนวยความสะดวกเรื่องเช่น Log การแก้ไขหรือ Conflict ในไฟล์ที่ Pull ลงมาจาก Remote repository ได้ด้วย

พึงระวังไว้เสมอว่าการแก้ไฟล์ที่ Branch Master ตรงๆ เป็นบาปหันต์ ไม่ควรทำอย่างยิ่ง

Chapter 2 : Git basic

2.1 ศัพท์แสง

- Init

Init หรือ Initial คือการเริ่มต้นสร้างระบบของ Git ในโฟลเดอร์นั้นเพื่อใช้เป็น Version control

- Clone

ในการนี้ที่เราไม่ได้เริ่มสร้างระบบของ Git เอง เราต้องสามารถที่จะ Clone ข้อมูลจาก Repository ที่มีอยู่แล้วได้ ซึ่งหลังจากการ Clone เราจะได้ระบบของ Git ประวัติการแก้ไขไฟล์ และข้อมูลต่างๆ ครบถ้วนเหมือนกับที่อยู่ใน Repository ต้นทาง



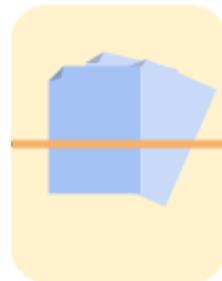
- Snapshot

Snapshot คือการเก็บสถานะของไฟล์ต่างๆ เอาไว้อ้างอิง คล้ายๆ กับการที่เราถ่ายรูปในรูปนั้นเราจะสามารถบอกได้ว่าวัตถุไหนรูปร่างแบบไหน อยู่ตำแหน่งใด เช่นถ้าแม่ให้เราจัดกระถางต้นไม้ใหม่ และบอกว่าไม่สวยงามให้ยกกลับมา เราจะใช้ภาพอ้างอิงก่อนเราจะย้ายเพื่อจัดกระถางต้นไม้มีกลับได้ ในทำนองเดียวกัน Git snapshot ก็เอาไว้บอกว่าไฟล์เรามีรูปร่างเป็นยังไงบ้างนั้นเอง และถ้าตอนนั้นไม่มีการเปลี่ยนแปลงมันก็จะเก็บแค่ตัวชี้ ไม่ถ่ายภาพใหม่



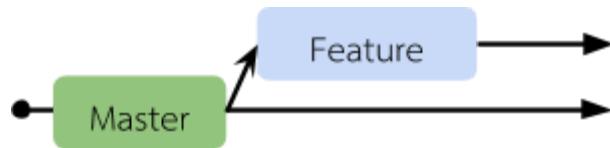
- Repository

Repository มักเรียกว่า Repo เป็นฐานข้อมูลที่จะเก็บ Git object model ไว้เพื่อสำรองข้อมูล ข้อมูลที่ถูกแก้ไขและ Commit มาเก็บไว้ที่ Repo แทนจะเรียกได้ว่าเป็นข้อมูลที่ถูกเก็บไว้แล้วทั่วๆ ไป ไม่ต้องกล่าวถึง Repo หรือ Repository มันก็คือตัวเดียวกันครับ มีทั้งแบบ Local repo ที่อยู่บนเครื่องเรา และ Remote repo ที่อยู่บนเครือข่าย



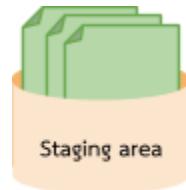
- Branch

Branch เป็นลักษณะของการแตกกิ่งออกไปพัฒนาโดยมีพื้นฐานจาก Branch ตั้งต้นหรือแม่ของมัน หากเราริ่มการใช้งานระบบ Git ก็จะสร้าง Branch หลักให้เรา เป็น Branch เริ่มต้นชื่อว่า Branch Master จากนั้นถ้าเราแยกไปพัฒนาใน Branch Feature ข้อมูลตั้งต้นใน Branch Feature ก็จะมีค่าเท่ากับ Master ซึ่งเป็นแม่ของมันในขณะนั้น และหลังจากที่แยก Branch ไปแล้วข้อมูลจะไม่เกี่ยวกัน ต่อให้เราทำพัฒนาใน Branch Feature ตัว Master ก็ยังจะเป็นข้อมูลชุดเดิมก่อนที่จะแตก Branch ไป เราจะเรียก Branch ปัจจุบันที่เรากำลังทำงานอยู่ว่า HEAD



- Add

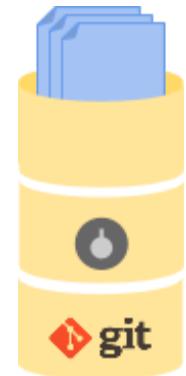
เมื่อเรา Add ระบบจะสร้าง Snapshot ของไฟล์และโฟลเดอร์ย่อๆแล้ว Add เข้าไปใน Staging area เป็นการบอกระบบทว่า Commit ต่อไปมี Snapshot อะไรบ้าง เพื่อจะเอาไปเก็บไว้ใน Git Repository แต่อย่าลืมว่าแค่เราต้องการเก็บยังไม่พอ เราต้อง Commit ด้วย เพราะสถานะการเก็บข้อมูลไว้ในฐานข้อมูลที่แท้จริงเกิดขึ้นตอน Commit



- Commit

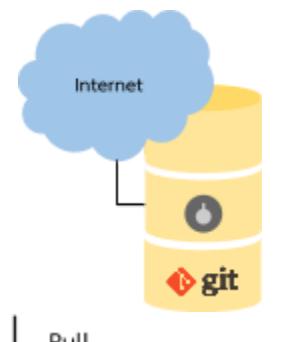
เรามักจะกล่าวถึง Commit ในความหมายสถานะหนึ่งของการทำงาน เช่น Commit ไฟล์นี้ หรือยัง? การ Commit ไฟล์คือการเอา Snapshot ของไฟล์ที่อยู่ใน Staging area ไว้ใน Git Repository นั้นเอง

เพื่อเป็นการยืนยันว่าเราต้องการแก้ไขข้อมูลที่ทำແเน้นนั้น เราต้องทำการ Commit มันจะก่อน เช่น เวลาใช้งานหลายๆ Branch เราจะแก้ไขไฟล์ที่ทำແเน้นเดียวกัน ไฟล์ซึ่งเหมือนกัน หากเราไม่ Commit ไว้จะก่อนที่จะเปลี่ยน Branch ระบบของ Git ก็จะมองเห็นเป็นการแก้ไฟล์ใน Branch ที่เราเพิ่ง Switch ไปซึ่งพิດไปจากที่เราต้องการ



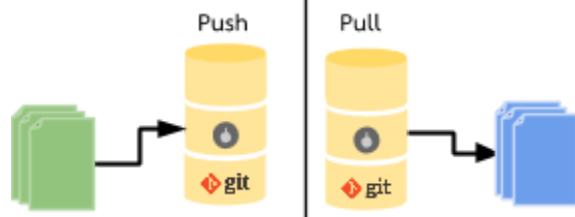
- Remote

Remote เป็นคำที่ใช้เรียก Repository ของ Git ที่อยู่บนเครื่องอื่นๆในระบบเครือข่าย ลักษณะเป็น Repo กลาง ปกติจะใช้ทำเป็น Official version เพราะถึงแม้ว่า Git จะไม่ต้องการ Server เพราะเป็น DVCS ก็จริง แต่คงจะงกันไม่น้อยถ้าไม่มี Repo ตัวกลาง เพื่อให้เข้าใจตรงกันว่า Code ที่อยู่ Repo ไหนเป็นหลัก โดยปกติใน Repo แบบนี้จะพัฒนาไฟล์บน Repo ไม่ได้ (ทำได้แค่ Merge เข้าไปรวมตัวกัน กับตัวใหม่)



- Push

ส่งข้อมูลขึ้นไป Update ให้กับ Remote repository

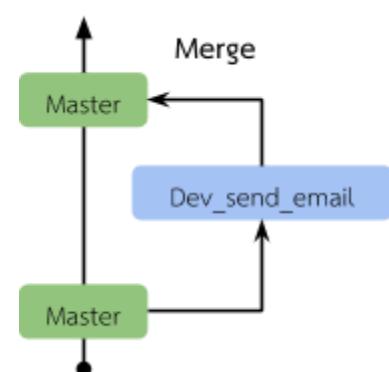


- Pull

ดึงข้อมูลลงมาจาก Remote repository

- Merge

รวมไฟล์ที่เกิดจากการแก้ไขแล้วใน Branch อื่นเข้ามา เช่นเรามี Master อยู่แล้วแต่ก็ต้องอุปเดตไฟล์อีเมล์ใช้ชื่อ Branch ว่า Dev_send_email เมื่อพัฒนาเสร็จแล้วเราจะ Merge ข้อมูลกลับมาที่ Master อีกครั้ง ดังนั้นเราไม่ได้



แก้ข้อมูลที่ Master ตรงๆ แต่เราใช้ชีรีพัฒนาที่อื่นแล้วรวมกลับทำให้ Master ทำงานได้เสมอ ดังนั้นเราจึงไม่มีบางติดตัว

2.2 คำสั่งพื้นฐานของ Git

\$ git config

เป็นคำสั่งที่ใช้กำหนดข้อมูลส่วนตัวของผู้พัฒนาแต่ละคน ทำให้รู้ว่าใครทำอะไรได้ใน Log ของ Git ซึ่งตัวที่เราจะตั้งค่ากันบ่อยๆ ก็ เช่น

\$ git config --global user.name "AorJoa"

ใช้ตั้งค่าข้อผู้ใช้

\$ git config --global user.email aorjоа@i-aor.com

หากเราต้องการให้ Git ตั้งค่าเป็นค่าตั้งต้นจะใส่ --global เข้าไปด้วย จากนั้นตั้งค่าเสร็จแล้วใช้ \$ git config --list ดู

```
AorJoa:repository dekcom$ git config --global user.name "AorJoa"
AorJoa:repository dekcom$ git config --global user.email aorjоа@i-aor.com
AorJoa:repository dekcom$ git config --list
credential.helper=osxkeychain
user.name=AorJoa
user.email=aorjоа@i-aor.com
user.ui=true
color.ui=true
push.default=simple
AorJoa:repository dekcom$
```

\$ git config --global color.ui true

ตั้งค่าให้ UI เวลาแสดงผลให้เป็นสีเวลาดูจะเด่นง่ายๆ

\$ git init

เป็นคำสั่งสร้างระบบ Git ขึ้นมา โดยที่ Git จะสร้างไฟล์เดอร์ที่เก็บข้อมูลการทำงาน Commit หรือ Repo ต่างๆ เก็บไว้ที่นี่ และจะถูกชื่อนเอาไว้ภายในไฟล์เดอร์ชื่อว่า .git (ไฟล์เดอร์ก็ใช้ . ได้นะ เพราะระบบ OS ถูกองไฟล์เดอร์เป็นไฟล์แบบหนึ่ง) ดังนั้นอย่าเพลอลบไฟล์เดอร์นี้ทิ้ง หละ เพราะมันเก็บเกือบทุกอย่างที่เป็น Repo ภายในเครื่องเราเลยที่เดียว

\$ git init สร้างระบบของ Git

```
AorJoa:git_basic dekcom$ git init
Initialized empty Git repository in /Users/dekcom/Dropbox/repository/git_basic/.git/
```

\$ git config --global user.name "AorJoa"

↑ ↑ ↑ ↑
คำสั่ง ตัวเลือก พารามิเตอร์

บอกว่าเป็นคำสั่งใน Command line

ขอเรียกตามนี้เพื่อความง่ายนะครับ

ใช้ตั้งค่า email ผู้ใช้

AorJoa:git_basic dekcom\$ cd .git
AorJoa:.git dekcom\$ tree .

```
.
├── HEAD
├── branches
├── config
├── description
└── hooks
    ├── applypatch-msg.sample
    ├── commit-msg.sample
    ├── post-update.sample
    ├── pre-applypatch.sample
    ├── pre-commit.sample
    ├── pre-push.sample
    ├── pre-rebase.sample
    └── prepare-commit-msg.sample
        └── update.sample
├── info
│   └── exclude
└── objects
    ├── info
    │   └── pack
    └── refs
        ├── heads
        └── tags
```

\$ git clone

กรณีที่เรามีโปรเจกใน Remote Repo ออยู่แล้ว และเราต้องการที่จะดึงข้อมูลอອกมาใช้ ลักษณะการทำงานก็คล้ายกับ Git init ที่เป็นการเริ่มต้นโปรเจกจากการที่ไม่มีอะไรเลย แต่ Git clone นั้นไม่ได้เริ่มต้นใหม่ค่ะที่เดียว เพราะมันคือการนำเอาข้อมูลใน Remote Repo ที่เจ้าของโปรเจกเขาร่างไว้แล้วมาเก็บต่อ ซึ่งจะมีข้อมูลเก็บทั้งหมดในโฟลเดอร์ .git ของคนอื่นติดมาด้วย ตัวอย่างนี้ลอง Clone มาจาก Repo

<https://github.com/chanwit/spock-report-html.git>

\$ `git clone https://github.com/chanwit/spock-report-html.git` ใช้ Clone ข้อมูลจาก Remote Repo

```
AorJoa:git_basic dekcom$ git clone https://github.com/chanwit/spock-report-html.git
Cloning into 'spock-report-html'...
remote: Counting objects: 125, done.
remote: Total 125 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (125/125), 14.56 KiB | 13.00 KiB/s, done.
Resolving deltas: 100% (25/25), done.
Checking connectivity... done.
```

\$ git status

ขณะที่เรากำลังทำงานอยู่ เราสามารถตรวจสอบไฟล์ทำถูกแก้ไข และสถานะของไฟล์ได้โดยการใช้คำสั่ง

\$ `git status` ก็จะเห็นสถานะของไฟล์

```
AorJoa:git_basic dekcom$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    index.txt

nothing added to commit but untracked files present (use "git add" to track)
```

\$ git add

ใช้ในการเพิ่มไฟล์ไปที่ Staging area เพื่อเตรียม Commit เข้า Git Repo ด้วยคำสั่ง

\$ `git add .` เอาทุกไฟล์ที่อยู่ภายในโฟลเดอร์นี้

\$ `git add sub_folder/*.txt` เอาทุกไฟล์ที่อยู่ภายในโฟลเดอร์ sub_folder ที่มีนามสกุลของไฟล์เป็น .txt

\$ `git add index.txt` เอาเฉพาะไฟล์ index.txt

จากนั้นลอง สั่ง \$ `git status` ก็จะเห็นสถานะของไฟล์

```

AorJoa:git_basic dekcom$ tree .
.
├── index.txt
└── sub_folder
    ├── about.html
    └── test.txt

1 directory, 3 files
AorJoa:git_basic dekcom$ git add sub_folder/*.txt
AorJoa:git_basic dekcom$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   index.txt
    new file:   sub_folder/test.txt

Untracked files:
  (use "git add <file>..." to include in what will
be committed)

    sub_folder/about.html

```

ในตัวอย่างเรา Add ไฟล์ไปที่ Staging area แค่ 2 ไฟล์คือ index.txt และไฟล์ test.txt ที่อยู่ในโฟลเดอร์ sub_folder และถ้าเรา Add ไฟล์เข้าไปผิดแล้วอยากเอาออกในขั้นตอนนี้ก็ใช้คำสั่ง
`$ git rm --cached sub_folder/test.txt` เพื่อลบออกจาก Cached เป็นต้น

ส่วนหากเป็นการแก้ไขไฟล์ที่อยู่ในระบบอยู่แล้ว แต่เราไม่ต้องการส่ง version นั้นออกไป ก็ใช้คำสั่ง `$ git checkout -- index.txt`

```

(use "git checkout -- <file>..." to discard changes in working directory)

modified:   index.txt

```

`$ git mv`
 ใช้เปลี่ยนชื่อไฟล์หรือโฟลเดอร์ วิธีการใช้ก็ไม่ยาก `$ git mv` ตั้งทาง เว้น Space และปลายทาง
`$ git mv sub_folder/test.txt sub_folder/bio.txt`

```

AorJoa:git_basic dekcom$ git mv sub_folder/test.txt
sub_folder/bio.txt
AorJoa:git_basic dekcom$ tree .
.
└── index.txt
└── sub_folder
    ├── about.html
    └── bio.txt

1 directory, 3 files
AorJoa:git_basic dekcom$ git status
On branch master

Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

    new file:   index.txt
    new file:   sub_folder/bio.txt

Untracked files:
(use "git add <file>..." to include in what will
be committed)

    sub_folder/about.html

```

\$ git commit

ใช้ยืนยันว่าเราต้องการเก็บไฟล์ที่อยู่ใน Staging area ลงไปไว้ใน Git Repository จะง่าย สั่งได้โดยใช้ \$ **git commit** ตามด้วยตัวเลือก และพารามิเตอร์ของตัวเลือก โดยปกติแล้วตัวเลือกที่ใช้จะใช้แค่ **-m** และมีพารามิเตอร์คือข้อความคอมเมนต์สั้นๆลงไว้ใน Commit เวลากลับมาดูจะได้ดูง่ายขึ้นว่าที่คอมมิทนั้นเราทำอะไรกับมันไป
\$ **git commit -m "Initial Git basic"** แต่ถ้าพารามิเตอร์ Commit ไม่ได้ต่อต้องการแก้ไขไฟล์

```

AorJoa:git_basic dekcom$ git commit -m "Initial Git
basic"
[master (root-commit) 07663fd] Initial Git basic
2 files changed, 3 insertions(+)
create mode 100644 index.txt
create mode 100644 sub_folder/bio.txt

```

\$ git rm

ใช้ลบไฟล์หรือโฟลเดอร์ วิธีการใช้ก็ไม่ยาก \$ `git rm` หรือ \$ `git remove` ตามด้วยไฟล์ที่ต้องการลบ แต่ถ้าเป็นโฟลเดอร์ต้องใส่ตัวเลือก `-r` เช้าไปได้วย

\$ `git rm -r sub_folder`

จากภาพด้านจะเห็นว่า Git ลบแค่ไฟล์เดียว ทั้งที่ในโฟลเดอร์มี 2 ไฟล์ นั่นก็เพราะเรา Commit ไฟล์เข้า Git Repo ไปแค่ไฟล์เดียว

และเมื่อสั่งลบไปแล้วถ้าจะให้เกิดผลต้องสั่ง Commit เพื่อยืนยันด้วย แต่จะเก็บไว้ Commit ยืนยันพร้อมกับการแก้ไขไฟล์ครั้งใหม่ก็ไม่ว่ากันครับ

```
AorJoa:git_basic dekcom$ git rm -r sub_folder
rm 'sub_folder/bio.txt'
AorJoa:git_basic dekcom$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:   sub_folder/bio.txt

Untracked files:
  (use "git add <file>..." to include in what will
be committed)

  sub_folder/
```

\$ git branch

ใช้ตรวจสอบ Branch ปัจจุบันที่เรากำลังทำงานอยู่ จะสร้าง Branch ใหม่ หรือจะลบ Branch ออกໄไปก็ได้

\$ <code>git branch</code>	ดู Branch ปัจจุบัน
\$ <code>git branch dev</code>	แตก Branch ใหม่ชื่อ dev
\$ <code>git branch -d dev</code>	ลบ Branch ชื่อ dev ซะ

```
AorJoa:git_basic dekcom$ git branch
* master
AorJoa:git_basic dekcom$ git branch dev
AorJoa:git_basic dekcom$ git branch
  dev
* master
AorJoa:git_basic dekcom$ git branch -d dev
Deleted branch dev (was 07663fd).
AorJoa:git_basic dekcom$ git branch
* master
AorJoa:git_basic dekcom$
```

\$ git checkout

ใช้สลับไป Branch หรือ Commit ที่เราต้องการ หรือจะสร้าง Branch จากคำสั่งนี้ก็ได้

\$ <code>git checkout dev</code>	สลับไป Branch ชื่อ dev ในกรณีที่มี Branch dev อยู่แล้ว
\$ <code>git checkout -b dev</code>	ใช้สร้าง Branch dev และ Checkout ไปด้วย

เหมือนไข่ \$`git branch dev` และตามด้วย \$`git checkout dev`

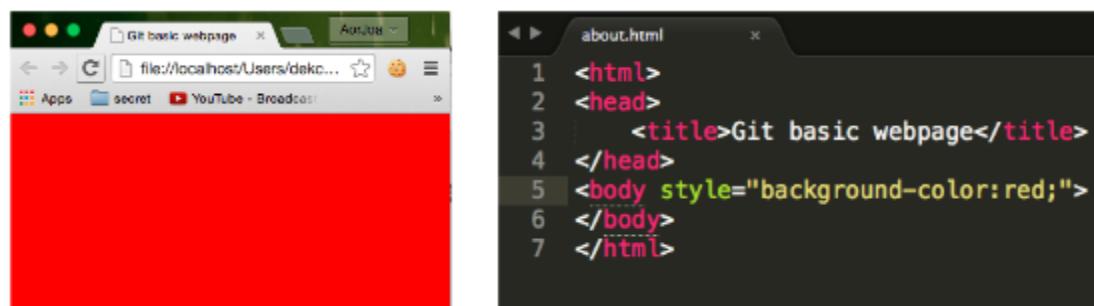
```
AorJoa:git_basic dekcom$ git checkout -b dev
Switched to a new branch 'dev'
AorJoa:git_basic dekcom$ git branch
* dev
  master
```

\$ git merge

เป็นคำสั่งที่ใช้ในการรวมอัพเดทไฟล์ที่ถูกแก้ไขใน Branch สอง Branch เข้าด้วยกัน เช่นถ้าเราจะแก้พื้นหลังไฟล์ about.html ให้เป็นสีแดง ก็แตกต่าง Branch background_red ออกไปจาก master จากนั้นถ้าแก้ไขโค้ดให้พื้นหลังเป็นสีแดงพอเสร็จแล้วเราอาจจะ Checkout ไปที่ master และ Merge กิ่ง background_red กลับเข้ามาให้ master ทำให้ตอนนี้เราไม่ได้แก้ไขโค้ดที่ master แต่เราได้พื้นหลังของไฟล์ about.html

\$ git checkout -b background_red แตกต่าง background_red จากนั้นก็แก้ไขโค้ด

```
$ git checkout -b background_red
Switched to a new branch 'background_red'
```

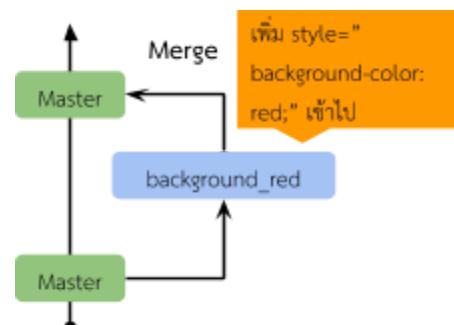


เสร็จแล้วสั่ง \$ git add . ตามด้วย Commit ที่ Branch background_red สั่ง \$ git commit -m "change about.html background to red"

```
AorJoa:git_basic dekcom$ git add .
AorJoa:git_basic dekcom$ git commit -m "change about.html background to red"
[background_red 33bd098] change about.html background to red
 1 file changed, 1 insertion(+), 1 deletion(-)
```

จากนั้น \$ git checkout master เพื่อกลับไปที่ master จากนั้นก็ merge \$ git merge background_red

```
AorJoa:git_basic dekcom$ git checkout master
Switched to branch 'master'
AorJoa:git_basic dekcom$ git merge background_red
Updating 67ca318..33bd098
Fast-forward
  sub_folder/about.html | 2 +-
  1 file changed, 1 insertion(+), 1 deletion(-)
```



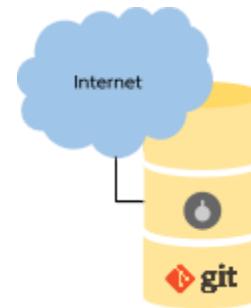
\$ git remote

เป็นคำสั่งที่ใช้จัดการ Remote repository กลางที่ใช้ในการเก็บข้อมูลซึ่งมักเป็น Repository ที่อยู่ในระบบเครือข่าย ถ้าเราต้องการเพิ่ม Remote repository ชื่อ origin ซึ่งเป็น Server ของ Github.com ที่迅猛ทะเบียนไว้แล้ว และ Repository ตาม Path นี้ https://github.com/Aorjoa/basic_git.git

```
$ git remote add origin https://github.com/Aorjoa/basic_git.git
```

```
AorJoa:git_basic dekcom$ git remote add origin
https://github.com/Aorjoa/basic_git.git
```

ถ้าจะลบก็ใช้คำสั่ง \$git remote rm origin



\$ git push

โดยปกติแล้วตัว Repo ที่อยู่บนระบบเครือข่ายหรือที่เรียกว่า Remote repo จะถูกตั้งค่าให้ไม่สามารถทำบางอย่างได้เช่น แก้ไขไฟล์ หรือ Commit โค้ดที่ Repo แบบนี้ได้ เพราะข้อมูลที่อยู่ข้างในจะเป็นการเก็บข้อมูลที่อยู่ในแบบที่ระบบของ Git จัดการแล้ว ไม่มี Working directory ที่เก็บไฟล์ปกติที่เอาไว้แก้ไข (คล้ายๆส่งไฟล์เดอร์ .git ที่อยู่ในโปรเจก เราขึ้นไปยังไงก็จัดได้) เราเรียก Repo ที่อยู่บน Server แบบนี้ว่า Bare repository (เพิ่ม --bare ใน \$ git init --bare หรือ \$ git clone --bare <https://github.com/chanwit/spock-report-html.git>) เวลาเราจะปรับมันให้เป็นรุ่นใหม่เราจะ push โดยดูดใหม่ขึ้นไปให้มันแทน โดยใช้คำสั่ง git push แล้วตามด้วย Remote repo และ Branch (ปกติแล้วชื่อ Branch จะตรงกันทั้งที่ Local และ Remote และใส่ Branch เพราะเราส่งไปแค่ master ตัวเดียว)

```
$ git push origin master
```

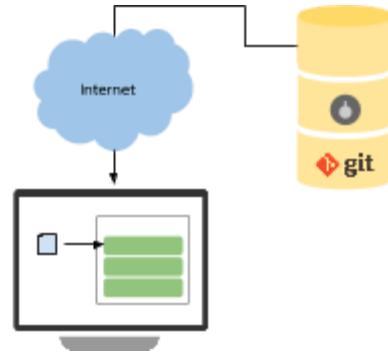
```
AorJoa:git_basic dekcom$ git push origin master
Counting objects: 13, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (13/13), 1.11 KiB | 0 bytes/s, done.
Total 13 (delta 0), reused 0 (delta 0)
To https://github.com/Aorjoa/basic_git.git
 * [new branch]      master -> master
```

\$ git fetch

เป็นคำสั่งที่ใช้ดึงข้อมูลจาก Remote Repo มาคงไว้ที่เครื่องเราก่อน (จริงๆแล้วมันไปดึงข้อมูลใน Remote Repo มา Update ตัว Local Repo ที่เครื่องเราในส่วนที่เป็นตัวติดตาม Remote Repo ให้เป็นข้อมูลที่ล่าสุด แต่จะไม่แท็คไฟล์ที่อยู่ใน Working directory ที่เรากำลังทำงานอยู่) คำสั่งนี้มากใช้คู่กับคำสั่ง Merge ครับ เพราะ Merge จะเป็นการเอาข้อมูลที่ดึงมาคงไว้ไปอัปเดตใน Working directory ที่เรากำลังทำงานอยู่จริงๆ

\$ `git fetch origin master` ดึงข้อมูลจาก Remote Repo origin มาคงไว้ (ดึงมาแค่ Branch master)

```
AorJoa:git_basic dekcom$ git fetch origin master
From https://github.com/Aorjоа/basic_git
 * branch           master      -> FETCH_HEAD
```



\$ git pull

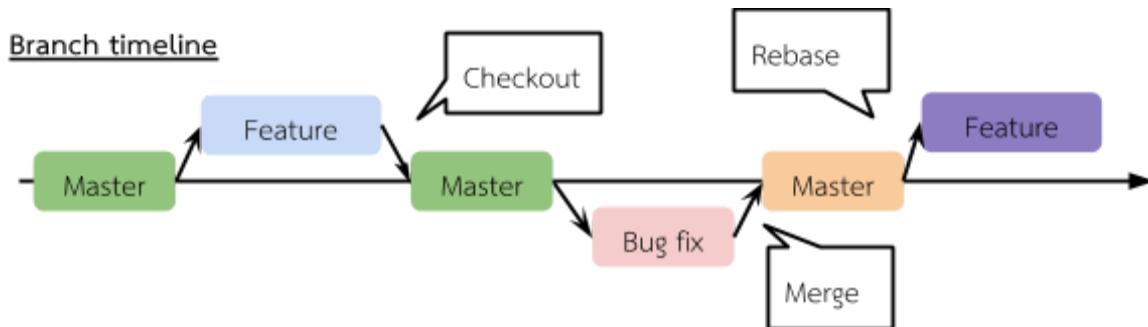
ถ้าจะว่ากันง่ายๆ คำสั่ง git pull ก็คือคำสั่งที่รวมร่างกันระหว่าง git fetch กับ git merge นักพัฒนาบางท่านแนะนำให้ใช้ git fetch และตามด้วย git merge เพราะสามารถตรวจสอบก่อนการ Merge รวมได้ แต่เพื่อความง่ายเราก็จะใช้ git pull ดึงข้อมูลกันเป็นหลัก

\$ `git pull origin master` ดึงข้อมูลจาก Remote Repo origin มาอัปเดตใน Local repo ในส่วนที่ติดตาม Remote repo และ Merge มันเข้ากับ Branch ปัจจุบัน)

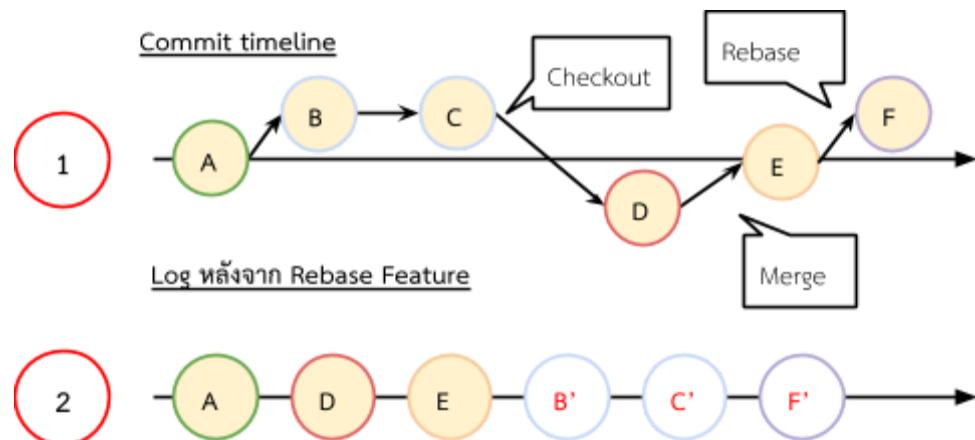
```
AorJoa:git_basic dekcom$ git pull origin master
From https://github.com/Aorjоа/basic_git
 * branch           master      -> FETCH_HEAD
Updating 33bd098..86d25bf
Fast-forward
 index.txt | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
```

\$ git rebase

เป็นการรวม Branch คล้ายกับ Merge แต่ตัว Rebase นั้นสามารถแก้ไขประวัติการเปลี่ยนแปลงไฟล์ได้ทำให้มันซับซ้อนกว่า และเพรากการที่มันแก้ประวัติการเปลี่ยนแปลงไฟล์ได้นั้นแหล่งอาจทำให้ข้อมูลเราหายได้ สิ่งที่ควรจำคือ **ห้าม Rebase commit ที่ถูก Push ขึ้น Remote repo ไปแล้ว** ลักษณะของการ Rebase คือการที่เปลี่ยนฐานของโค๊ดชุดใหม่ ลองนึกถึงตอนที่เราแตก Branch จาก master ออกมารสสร้าง Feature สักตัวขณะที่กำลังทำอยู่ เรา ก็จะ Bug เลยกับไปที่ Branch master และแก้ Bug จนเสร็จ จากนั้นก็กลับมาทำงานที่ Branch ของ Feature อีกรอบ แต่อย่าลืมว่าในขณะนี้ master เป็นไปแล้ว เรา ก็เลยทำการเปลี่ยนฐานหรือ Rebase ให้ฐานมันกลายเป็นรุ่นใหม่นั่นเอง ที่นี่ที่ Branch ที่พัฒนา Feature ก็ไม่มี Bug ตัวนั้นแล้ว



หลังจากสั่ง `$ git rebase master` หากดู Log แบบ Graph หลังจากการ Rebase ใน Branch ที่พัฒนา Feature ก็จะเห็น Commit เรียงกันเป็นเส้นตรง



จากภาพจะเห็นตัว Commit **B'**, **C'** และ **F'** ซึ่งไอ้ 3 Commit นี้จะเป็นคนละตัวกับ B, C และ F ใน Commit timeline แต่ข้อมูลข้างในเหมือนกันทุกประการ นั่นเป็นเหตุผลที่ว่าทำไมห้าม Rebase Commit ที่ถูก Push ขึ้น Remote repo ไปแล้ว เพราะถ้ามี Commit ที่ถูก Push ไป มันจะสร้าง Object ของ Commit ใหม่ และเมื่อไหร่ที่เรา Push ขึ้น Remote rep อีกครั้ง มันจะได้ Object Commit สองตัวที่มีข้อมูลเหมือนกัน เวลามาดูก็จะงง ปล. ถ้าอ่านไม่รู้เรื่องก็ลืมๆมันไปก่อน ฝึกปรือเดินลง平原เพียงพอแล้วค่อยกลับมาแก้แค้นก็ยังมีราย

\$ git log

ใช้ดู Log หรือประวัติการทำงาน ไฟล์จะมี Hash SHA1 มีหลากหลายให้เล่น เช่น Commit มี Hash SHA1 เป็นอะไร ใช้ดูแบบกราฟได้ ดูได้ว่าใคร Commit ส่วนวิธีใช้หลักๆคือ \$ git log แค่ไหนก็จะได้ประวัติของ Repository ออกมาก โดยส่วนนี้ก็จะสามารถดูรายละเอียดการ Commit ผ่านจีน Commit การเส้นทางของการพัฒนา แต่ Branch ไหนอย่างไร ก็สามารถเห็นได้

```
AorJoa:test dekcom$ git log --graph
* commit a32406dbcc479b3533a348ba86a390b5e867ae5b
| Merge: 9bb7f17 ed49208
| Author: AorJoa <aorjoa@i-aor.com>
| Date: Mon Oct 13 00:47:43 2014 +0700
|
|     Merge branch 'bug'
|
* commit ed49208c80a7739db12cd3d216edbc47bf1c08
| Author: AorJoa <aorjoa@i-aor.com>
| Date: Mon Oct 13 00:47:04 2014 +0700
|
|     bug fix
|
* commit 9bb7f1714c649f78e7f717c395e0b575a680358e
Author: AorJoa <aorjoa@i-aor.com>
Date: Mon Oct 13 00:44:49 2014 +0700
```

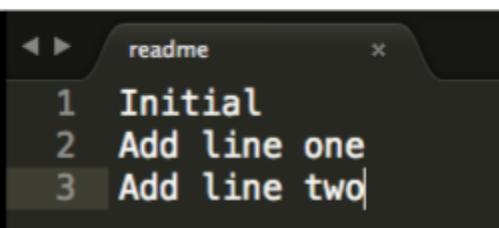
\$ git reset

เป็นคำสั่งที่ใช้ย้อนประวัติการทำงานกลับไปรุ่นก่อนๆหน้า คำสั่งนี้มีที่ใช้บ่อยๆ 2 แบบคือ soft reset โดยใช้พารามิเตอร์ --soft ในกรณีย้อน Commit แต่ไม่ลบสิ่งที่เรากำลังทำ กับแบบ hard reset โดยใช้พารามิเตอร์ --hard ย้อน Commit แบบทั้งสิ่งที่เรากำลังทำด้วย (ไฟล์ต่างๆจะถูกทับด้วยข้อมูลใน Snapshot ที่เราสั่งให้ Reset ไป) คำสั่ง git reset --hard ค่อนข้างอันตราย เพราะมันจะทำให้ประวัติหายไป จึงไม่แนะนำให้ใช้สำหรับเมื่อใหม่ครับ

\$ git reset --hard ed51e23

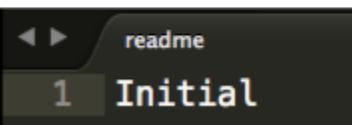
ก่อน Reset

```
AorJoa:test dekcom$ git log --oneline
a497ebe Add line two
951cfb6 Add line one
ed51e23 Initial repository
AorJoa:test dekcom$
```



หลัง Reset

```
AorJoa:test dekcom$ git reset --hard ed51e23
HEAD is now at ed51e23 Initial repository
AorJoa:test dekcom$
```



\$ git revert

เป็นคำสั่งที่ใช้ย้อนประวัติการทำงานกลับไปรุ่นก่อนๆหน้า คำสั่งนี้ปลดภัยกว่า \$ git reset เพราะโดยปกติแล้วมันจะสร้าง Commit ใหม่โดยไม่ไปยุ่งกับของเดิม จากตัวอย่างผมใช้คำสั่ง

\$ git revert --no-edit ed51e23..HEAD
บอกว่าให้ย้อนจาก Commit ed51e23 ไปจนถึง HEAD ซึ่งก็คือตำแหน่งที่เราอยู่ปัจจุบัน (ที่ทำอย่างนี้เพราะป้องการ Conflict หรือการแก้ไขไฟล์ชนกันในกรณีที่เราแก้ไฟล์หลายครั้ง) เลยจะเห็นว่า มี Commit ใหม่เพิ่งเข้ามา 2 ตัว คือ ca764a3 กับ 4e8849b และในคำสั่ง ผมใส่ --no-edit เพราะต้องการใช้ข้อความ Comment ใน Commit เป็นค่าตั้งต้นของระบบ คือ Revert แล้วตามด้วย Comment ที่อยู่ใน Commit ตัวต้นแบบ ถ้าสังเกตจะเห็นว่าเวลา Revert มันย้อนกลับจาก Commit ใหม่กว่าไปทางก้ากว่า (Add line two ใหม่กว่า Add line one)

```
a497ebe Add line two
951cfb6 Add line one
ed51e23 Initial repository
AorJoa:test dekcom$ git revert --no-edit ed51e23
..HEAD
[master 4e8849b] Revert "Add line two"
 1 file changed, 1 insertion(+), 2 deletions(-)
[master ca764a3] Revert "Add line one"
 1 file changed, 1 insertion(+), 2 deletions(-)
AorJoa:test dekcom$ git log --oneline
ca764a3 Revert "Add line one" Commit
4e8849b Revert "Add line two" ที่เกิดจากการ
a497ebe Add line two
951cfb6 Add line one Revert
ed51e23 Initial repository
```

จะเห็นว่าคำสั่งบางคำสั่งที่ผมยกตัวอย่างมาๆ ตัวเราสามารถใช้แทนกันได้ และบางตัวมีการใช้พารามิเตอร์เพื่อความง่ายในการใช้งาน หากอ่านการใช้งานคำสั่งข้างบนแล้วยังรู้สึกงงก็ไม่แปลก เพราะบางคำสั่งเกี่ยวเนื่องกันเป็น Flow ต่อกัน หรือใช้งานร่วมกัน การลองใช้งาน Git และการอ่านเนื้อหาเพิ่มเติมจะทำให้ท่านมีการมีแก่กล้าขึ้น จากนั้นค่อยกลับมาอ่านอีก ก็จะเข้าใจได้ไม่ยาก



Chapter 3 : Installation

หลังจากที่เรารู้ข้อมูลเบื้องต้นเกี่ยวกับ Version control, ประวัติของ Git และคำสั่งพื้นฐานของ Git ไปแล้ว ก็ถึงขั้นตอนที่ยกที่สุดในการที่จะใช้งาน Git ก็คือขั้นตอนการติดตั้งเนี่ยแหละจะทำผ่านผู้ชุม =====

3.1 Windows

1) ไปที่ <http://git-scm.com/downloads>

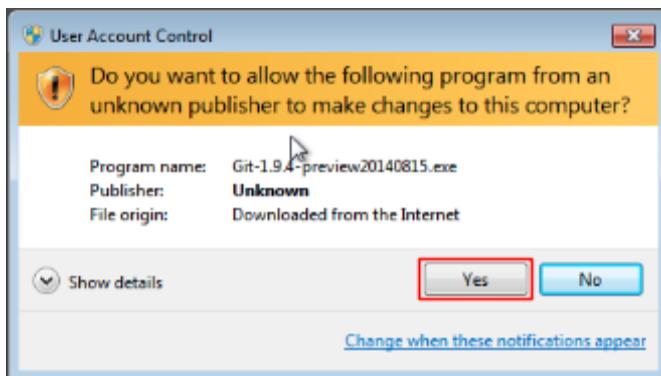
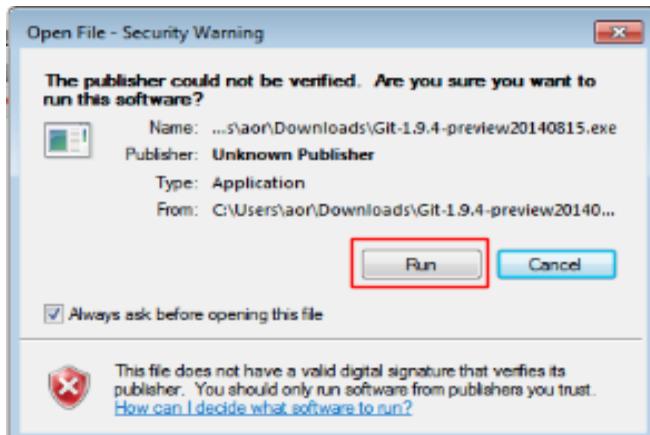


กด Download ของ Windows และรอโหลด

2) ติดตั้ง Git ลงในเครื่อง

Git-1.9.4-preview20140815	9/1/2014 3:54 AM	Application	17,390 KB
---------------------------	------------------	-------------	-----------

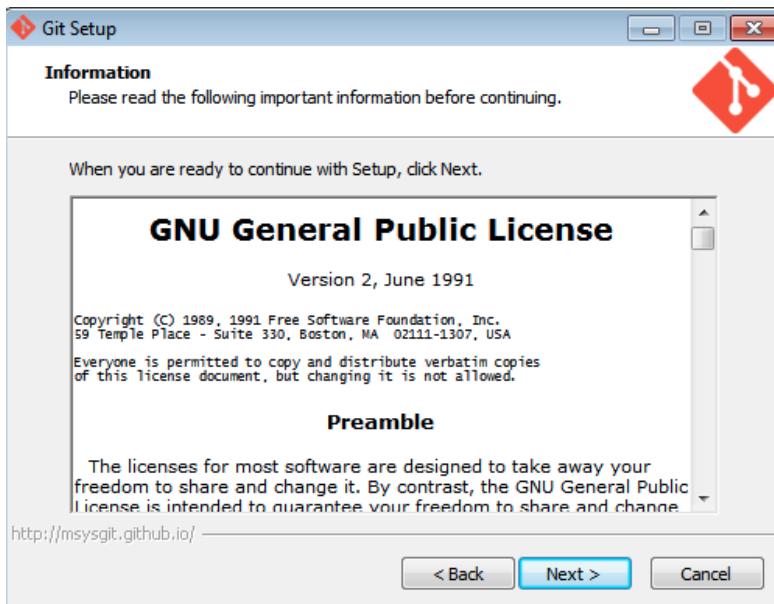
ถ้ามีการเตือนก็กดตามภาพด้านล่างเลย ถ้าไม่มีก็ผ่านไปได้เลย



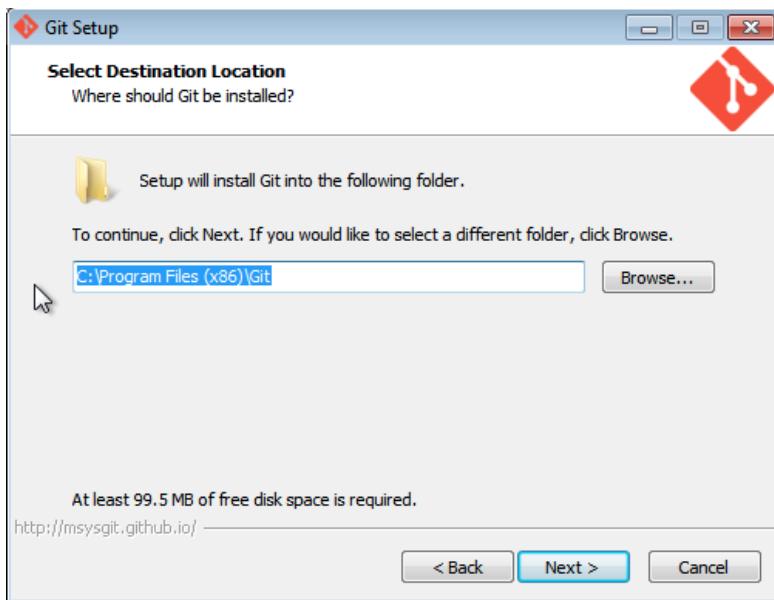
ตัว Installer จะเด้งขึ้นมา กด Next ไปเลย



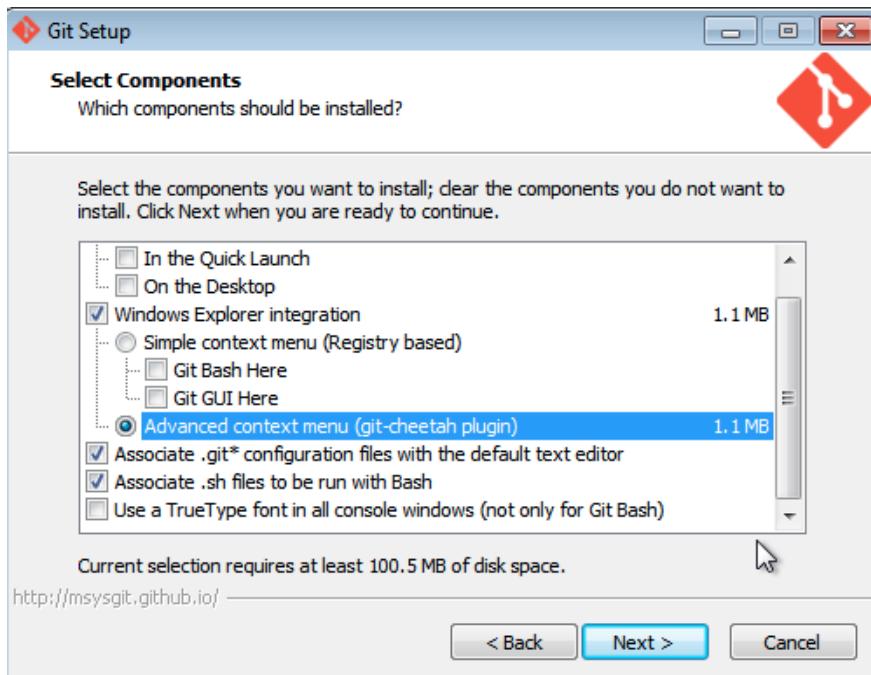
ตัวนี้เป็นคำอธิบายลิขสิทธิ์การใช้งาน กด Next ไป



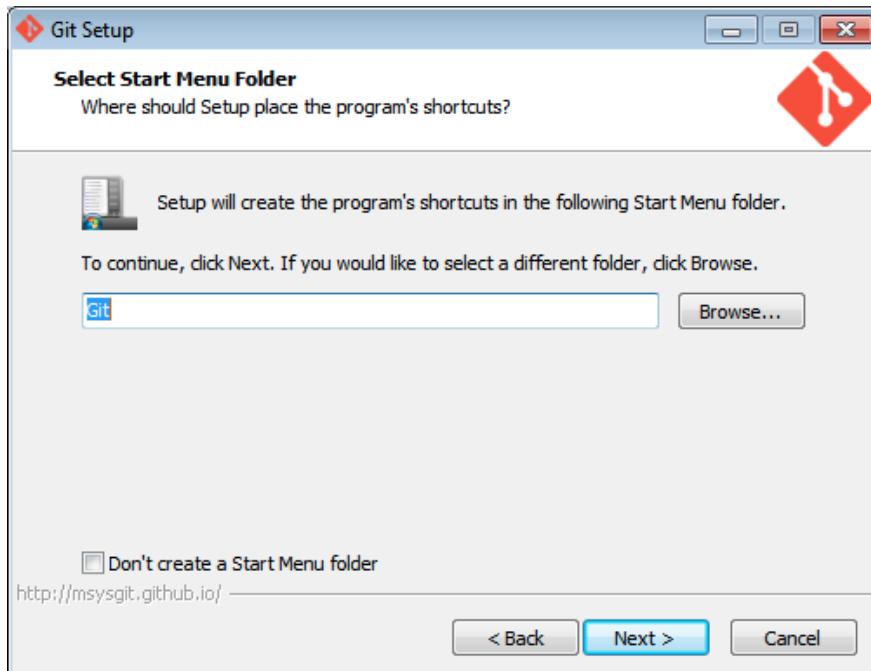
เลือก Directory ปลายทางที่จะเอาไฟล์ของ Git ไปเก็บไว้ จากนั้นกด Next



เลือก Component ว่าจะเอาอะไรไว้ เอาอย่างอื่นออก กด Next ได้เลย



กำหนดชื่อของ Shortcut กด Next



ตัวเลือกแรก : ใช้คำสั่ง Git ได้เฉพาะใน Git Bash

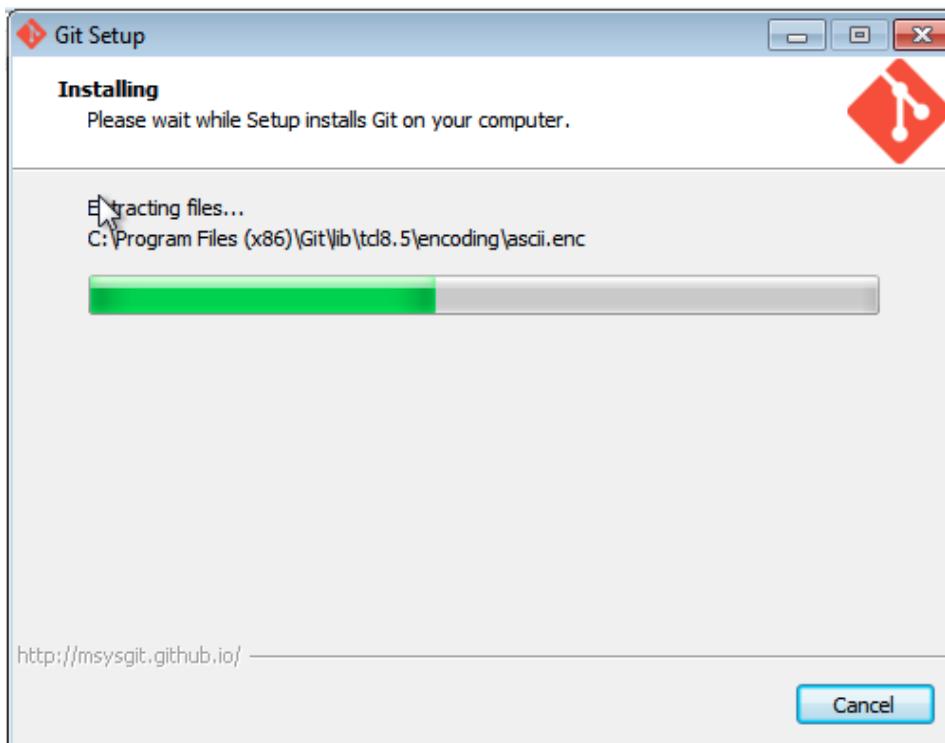
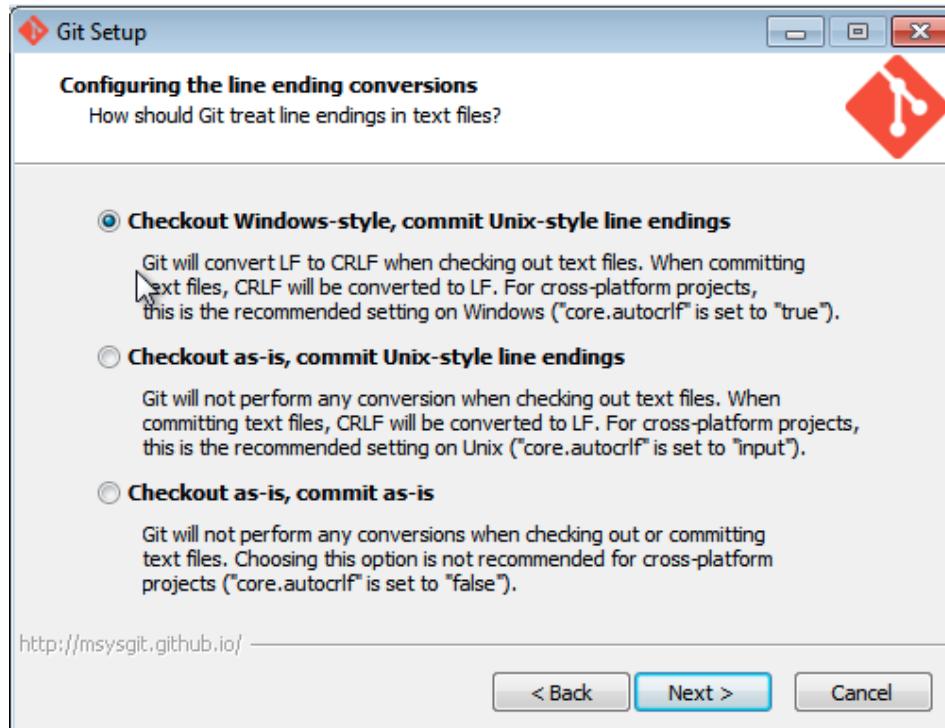
ตัวเลือกที่สอง : แก้ไข Windows Command Prompt ให้สามารถใช้คำสั่ง Git ได้ด้วย แนะนำให้เลือกตัวนี้

ตัวเลือกสุดท้าย : แก้ไข Windows Command Prompt ให้สามารถใช้คำสั่ง Git และติดตั้งคำสั่งที่คล้ายของ UNIX ลงไปด้วย (มีคำเตือนว่าบางคำสั่งอาจจะทับกัน ซึ่งอาจจะทำให้การใช้งาน Command ใน CMD เปลี่ยนไป)

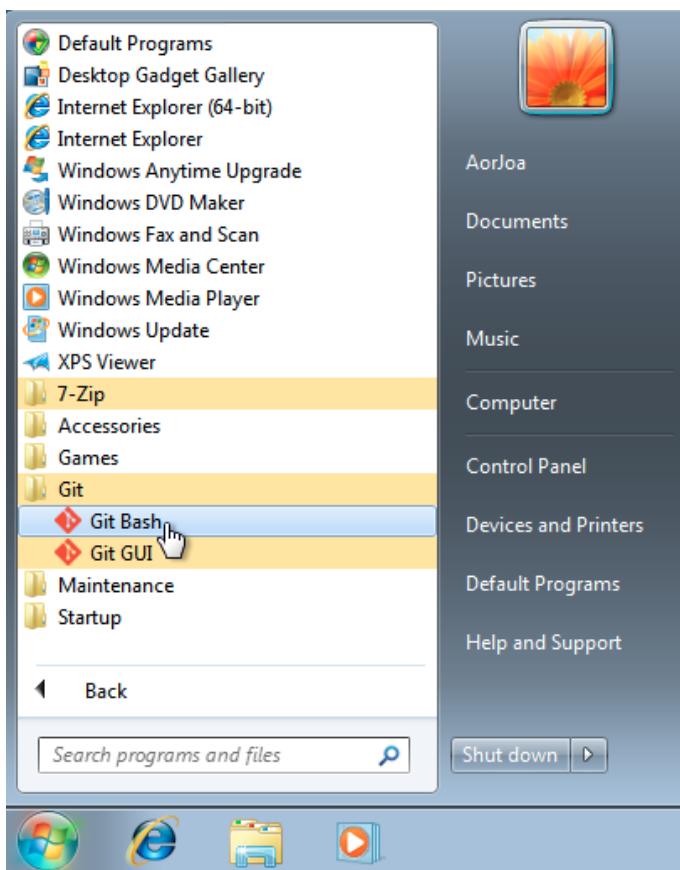
จากนั้น กด Next



เป็นตัวเลือกในการจับบรรทัด ให้เลือกตัวเลือกแรก เพราะการจับบรรทัดของ Windows กับของตระกูล UNIX เนี่ยจะไม่เหมือนกัน การเลือกตัวเลือกนี้จะทำให้สามารถทำงานข้าม Platform ได้โดยไม่ติดปัญหานี้ จากนั้นกด Next



เมื่อติดตั้งเสร็จแล้วกด View ReleaseNotes.rtf ถ้าต้องการอ่านไฟล์ข้อมูลประวัติการแก้ไข จากนั้น Finish ไป



ลองเข้า Git bash ดูโดยไปที่ Start menu > All Programs > Git > Git bash จากนั้นลองใช้คำสั่ง `git version` ลองว่าทำงานได้จริงไหม

```
aor@AOR-PC ~
$ git version
git version 1.9.4.msysgit.1
```

ถ้าขึ้นเลข Version ของ Git ก็เป็นอันใช้ได้

3.2 Mac OS X

1) ไปที่ <http://git-scm.com/downloads> และดาวน์โหลดตัวติดตั้งใน Mac OS X มา



Downloading Git

⬇️

Your download is starting...

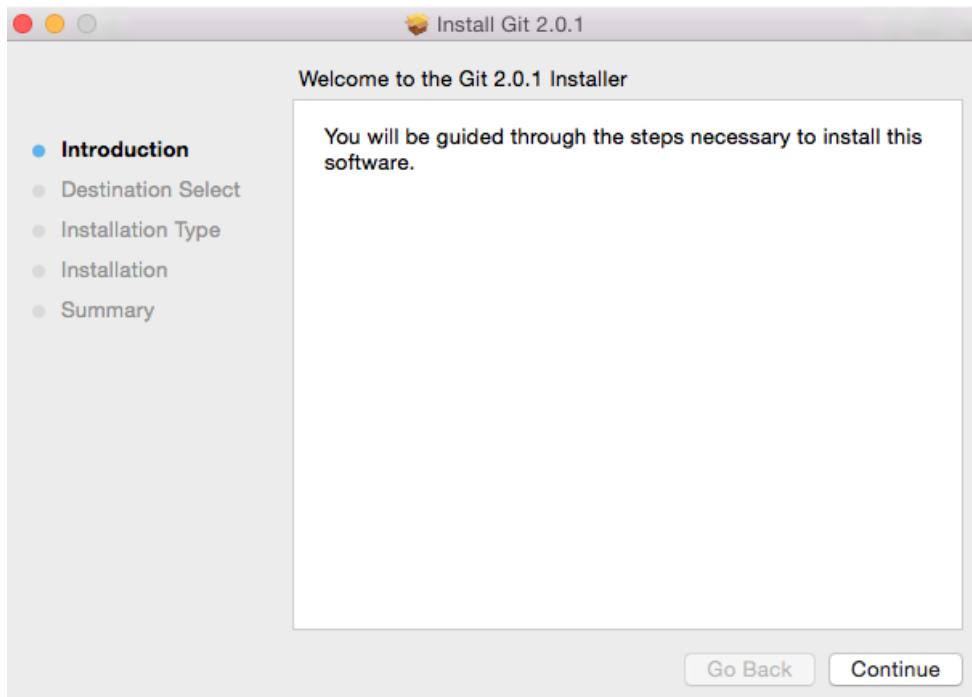
You are downloading version **2.0.1** of Git for the **Mac** platform. This is the most recent [maintained build](#) for this platform. It was released **2 months ago**, on 2014-06-29.

If your download hasn't started, [click here to download manually](#).

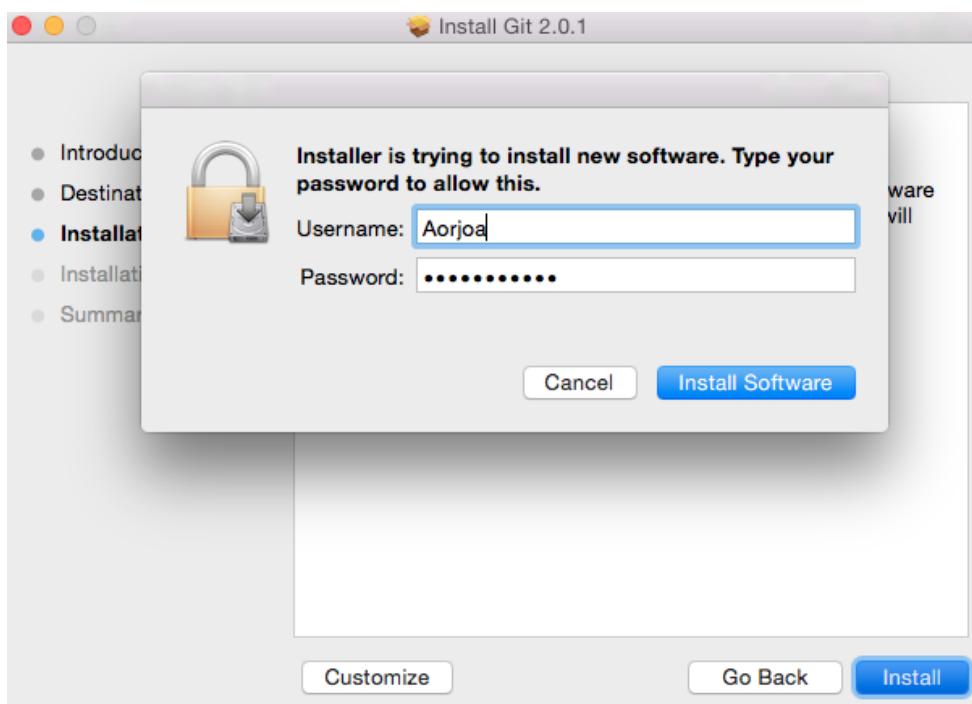
The current source code release is version **2.1.0**. If you want the newer version, you can build it from [the source code](#).

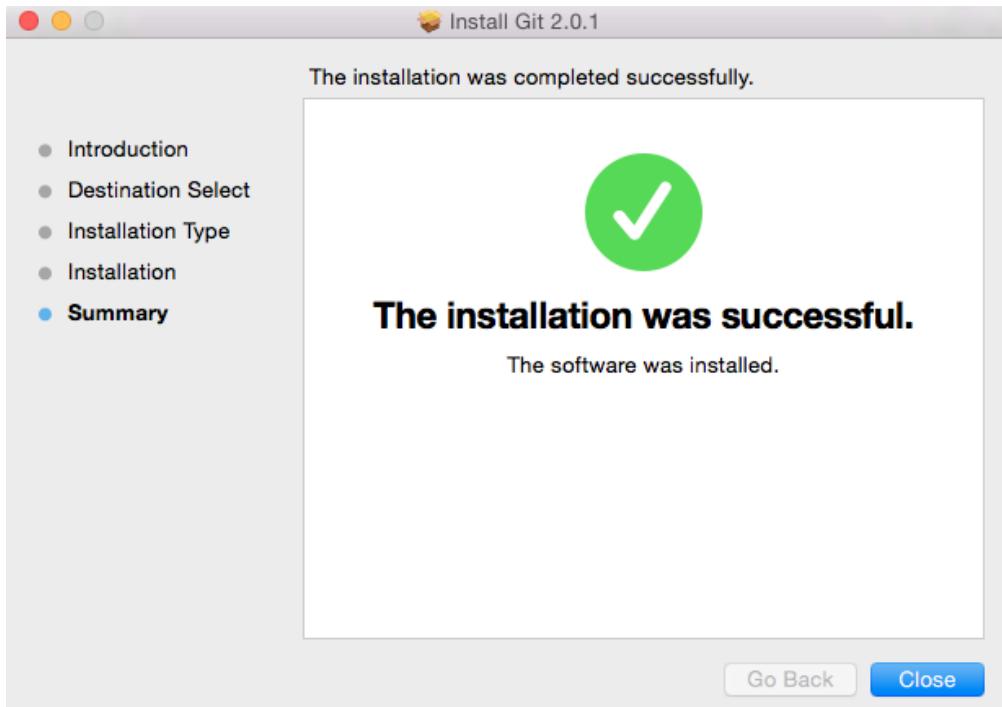
2) ติดตั้ง Git ลงในเครื่อง



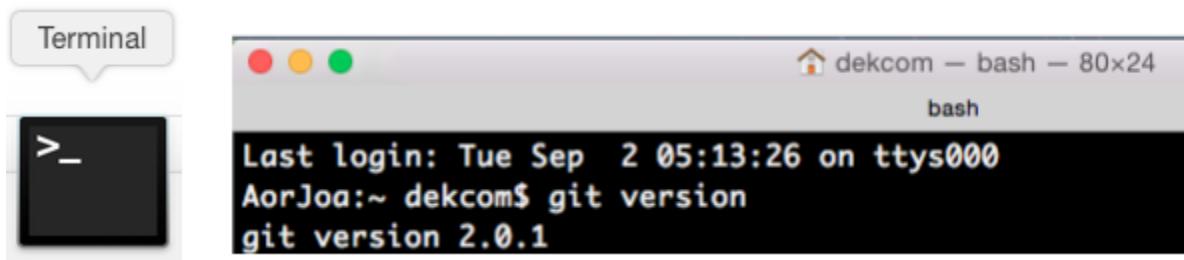


กด Continue รัวๆได้เลย





หลังจากติดตั้งเสร็จแล้วให้เปิด Terminal และลองพิมพ์ git version และแสดงผลออกมาแบบนี้เป็นใช่ได้



3.3 Others operating system

ในระบบปฏิบัติการอื่นก็สามารถ Download และติดตั้งได้ตามลิงก์ <http://git-scm.com/downloads> วิธีการติดตั้งก็คล้ายๆกัน

Chapter 4 : Git scenario

การใช้งาน Git หลังจากที่เราติดตั้งและรู้จักกับคำสั่งแล้ว สิ่งที่ควรทราบอีกนิดคือ Git นั้นมีคำสั่งที่คล้ายๆ กันในทุกๆ ระบบปฏิบัติการ และไฟล์ต่างๆ ก็จะคล้ายกัน ส่วนตัวผมจะใช้งานใน Mac OS X และใช้ไฟล์ Source code ของ Software เป็นหลักในการแสดงตัวอย่าง แต่ก็สามารถใช้คำสั่งเดียวกันในระบบปฏิบัติการอื่นได้ และ ผมมีคำแนะนำการใช้งานสำหรับ Windows คือให้เปิดใช้งานด้วย Git bash เพราะคำสั่งบางคำสั่งนั้นทำได้เฉพาะใน UNIX ซึ่ง Git bash จะจำลองให้เราใช้ได้ (ข้อจำกัดของคำสั่งใน Command Prompt)

สถานการณ์ที่ 1 : มือใหม่ มีนติ๊บ !@#%!@

อันดับแรกให้ตั้งค่าข้อมูลทั่วไปของนักพัฒนา ก่อนครับ

```
$ git config --global user.name "AorJoa"
$ git config --global user.email aorjoa@i-aor.com
$ git config --global color.ui true
$ git config --global user.ui true
```

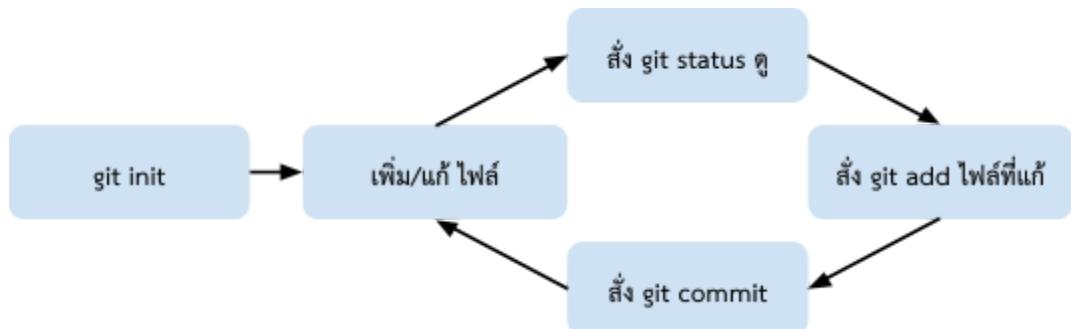
ใช้ตั้งค่าชื่อผู้ใช้ (ไม่ใส่ “ ก็ได้)

ใช้ตั้งค่า Email

ข้อมูลข้างบนเป็น 2 คำสั่งแรกเป็นข้อมูลของผู้ใช้ อย่าลืมเปลี่ยนเป็นของตัวเอง ไม่งั้นผิดมหันต์ยังกะลอกการบ้านแล้วลอกซื้อเพื่อนมาด้วยยังไงยังเงิน

```
AorJoa:scenario_git dekcom$ git config --global user.name "AorJoa"
AorJoa:scenario_git dekcom$ git config --global user.email aorjoa@i-aor.com
AorJoa:scenario_git dekcom$ git config --global color.ui true
AorJoa:scenario_git dekcom$ git config --global user.ui true
```

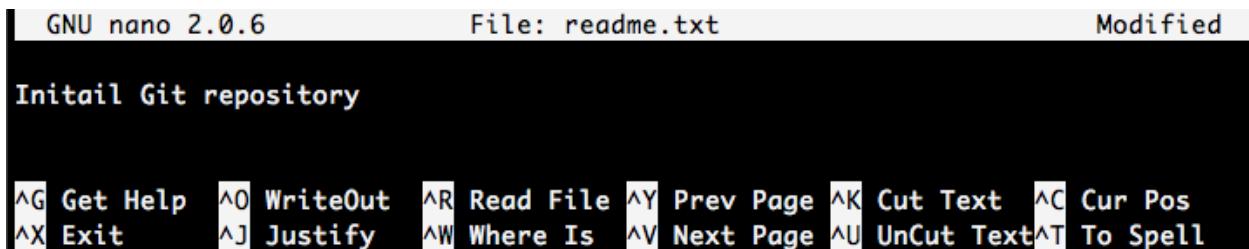
จากนั้นมาดูตัว Git ซึ่งมี flow แบบที่ง่ายที่สุดในการที่เราจะเก็บโค้ดดังนี้ครับ



สำหรับมือใหม่มีอะไรที่จะดีไปกว่าการพิมพ์ตามตัวอย่างแล้วหละครับ ดังนั้นก็จัดการเลยครับสร้างไฟล์เดอร์ขึ้นมา สักที่หนึ่งแล้ว สั่งไปเลย \$ **git init** เริ่มสร้างระบบ Repository ของ Git ขึ้นมาใช้งาน

```
AorJoa:scenario_git dekcom$ git init
Initialized empty Git repository in /Users/dekcom/Downloads/scenario_git/.git/
```

จากตอนนี้ไฟล์ในระบบจะมีเพรำพเพิ่งสร้างไฟล์เดอร์ตະกີເອງ ດັ່ງນັ້ນເຮົາຈະລອງສ້າງໄຟລ໌ສັກໄຟລ໌ນີ້ຊື່ອ
readme.txt ມີໃຫຍ່ຄຳສັ່ງ \$ nano readme.txt ຄຳສັ່ງ nano ອື່ນ Text editor ຕົວໜຶ່ງໃນ Unix ຊຶ່ງພມເຂົາມາໃຫຍ່ສ້າງ
ໄຟລ໌ແລະແກ້ໄຂຂໍ້ຄວາມ ຄວາມຈິງຈະໃໝ່ Notepad ກີ່ໄດ້ ເຂົາເປັນວ່າໃໝ່ນສ້າງແລະແກ້ໄຟລ໌ໄດ້ເປັນພວ
ຈາກນັ້ນກົດ Ctrl + X (ກົດປຸ່ມຄອນໂທຣລພ້ອມກັບປຸ່ມ X ບນຄີຢືນບ່ອຮົດ)



ມັນຈະຄາມວ່າຕ້ອງການ Save ທີ່ໄປ

```
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
Y Yes
N No
^C Cancel
```

ຈາກນັ້ນຈະຄາມຊື່ໄຟລ໌ ກົດປຸ່ມ Enter ໄປໄດ້ເລີຍ ເພົ່າເຮົາຕັ້ງຊື່ໃຫ້ ຕັ້ງແກ້ຕອນເຮົາກຳສັ່ງ nano ແລ້ວ

```
File Name to Write: readme.txt
^G Get Help      ^T To Files      M-M Mac Format      M-P Prepend
^C Cancel        M-D DOS Format    M-A Append       M-B Backup File
```

ຈາກນັ້ນມັນຈະເລີນອາກມາທີ່ທີ່ນ້າຈອ Command ປົກຕິເຮົາ ໃຫ້ລອງສັ່ງ ls ເພື່ອດູໄຟລ໌ວ່າມາທີ່ໄປ ຄ້າເຫັນຊື່ໄຟລ໌ແສດງ
ວ່າສ້າງໄດ້ແລ້ວ

```
AorJoa:scenario_git dekcom$ nano readme.txt
AorJoa:scenario_git dekcom$ ls
readme.txt
```

ຄ້າໃໝ່ Text editor ຕົວໜຶ່ນກີ່ໃໝ່ງານຕາມປົກຕິຂອງມັນນະຄົບ Notepad ກົດ Save ໄຟລ໌ໄດ້ເລີຍ ໂີ່ຕ້ອງທໍາຫລາຍຂັ້ນຕອນ
ແບບຜົມກີ່ໄດ້ ຂອແຄ້ໄຕໄຟລ໌ອາກມາ ຊັດມາກີ່ສັ່ງ \$ git status ປຶ້ງໃຫ້ດູສະນະ Working directory ຂອງເຮົາ

> ຂອດ້າວາ Add/Delete/Rename

ການເພີ່ມໄຟລ໌ທີ່ຍັງໄໝເຄຍເພີ່ມ ປົກຕິໄຟລ໌ທີ່ຍັງໄໝເຄຍເພີ່ມໄໝຈະເປັນຕ້ອງດໍາເນີນການ Add ກີ່ໄດ້ ແຕ່ຕົວນີ້ມັນເປັນ Commit
ແຮກເລີຍຈະເປັນຕ້ອງເພີ່ມຂໍ້ມູນເຂົ້າໄປໄໝຈັ້ນມັນ Commit ໄນໄດ້ (ພົມພາຍານຈະບອກວ່າປົກຕິໄຟລ໌ທີ່ສ້າງໃໝ່ຈະຍັງໄໝ
Track ແລະ Commit ເກັບກີ່ໄດ້ ແຕ່ນີ້ມັນເປັນ Commit ແຮກເລີຍຈະເປັນຕ້ອງ Add) ຄຳສັ່ງທີ່ໃຫຍ່ໂດຍ \$ git add ຊື່ໄຟລ໌

```
AorJoa:scenario_git dekcom$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README.txt

nothing added to commit but untracked files present (use "git add" to track)
```

แต่ถ้าเราแก้ไขไฟล์จะมีคำสั่ง 2 แบบที่ทำได้ คือการเพิ่มไฟล์เข้าใน Commit แบบปกติ (\$ git add ชื่อไฟล์) และไม่เก็บไฟล์เวอร์ชันนั้นไว้ใน Commit นั้น แต่เอาไว้ Commit หลังตอนหลังก็ได้ (\$ git checkout -- ชื่อไฟล์)

```
AorJoa:scenario_git dekcom$ git status
On branch master
Changes not staged for commit: เพิ่มไฟล์
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   README.txt ไม่ทำการแก้ไขเก็บใน Commit นั้น

no changes added to commit (use "git add" and/or "git commit -a")
```

การลบไฟล์ สั่ง \$ git rm ชื่อไฟล์ ได้เลย แต่ถ้าเป็น โฟลเดอร์ต้องใส่พารามิเตอร์ -r ให้ recursive ไฟล์ในโฟลเดอร์ สั่ง \$ git rm -r ชื่อโฟลเดอร์ มันต่างจากการลบปกติตรงที่ออกจากไฟล์จะหายไปจากโฟลเดอร์แบบปกติแล้ว Git ยังรู้ด้วยว่าเราลบไฟล์ออกไปจาก Working directory ของเรา

```
AorJoa:scenario_git dekcom$ git rm README.txt
rm 'README.txt'
AorJoa:scenario_git dekcom$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:   README.txt
```

การเปลี่ยนชื่อ ก็ใช้ \$ git mv ชื่อไฟล์ปัจจุบัน ชื่อไฟล์ที่จะเปลี่ยน

```
AorJoa:scenario_git dekcom$ git mv README.txt readyou.txt
AorJoa:scenario_git dekcom$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:   README.txt -> readyou.txt
```

จะเห็นว่าไฟล์ readme.txt เพิ่งขึ้นมา ในสถานะ Untracked files คือยังไม่ได้ติดตามการเปลี่ยนแปลง และระบบแนะนำให้ใช้ git add ตามด้วยชื่อไฟล์ ตามมันเลยครับสั่ง \$ `git add readme.txt` แล้วสั่ง \$ `git status` ดูว่าเข้าจริงปะ

```
AorJoa:scenario_git dekcom$ git add readme.txt
AorJoa:scenario_git dekcom$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:  readme.txt
```

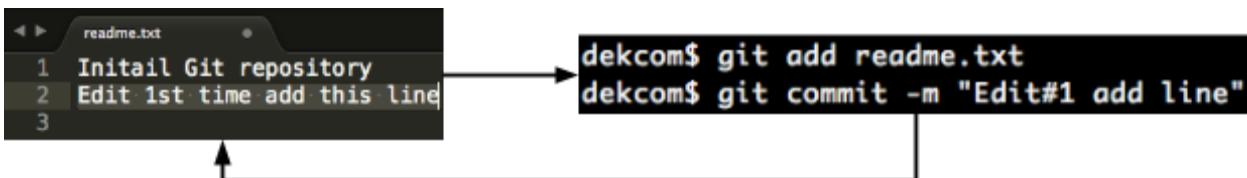
ถ้าเข้าไปแล้วจะขึ้นสีเขียวๆในกรอบที่บอกว่า Change to be committed คือสถานะที่มีการเปลี่ยนแปลง และไปอยู่ที่ Staging area แล้ว (สร้าง Snapshot) ถัดมาเราก็สั่ง \$ `git commit -m "Initial Git repository"` เพื่อเอา Snapshot ใน Staging area ไปเก็บในฐานข้อมูลของ Git

```
AorJoa:scenario_git dekcom$ git commit -m "Initial Git repository"
[master (root-commit) 8ba8bb9] Initial Git repository
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt
```

ถ้าเราสั่ง \$ `git status` จะเห็นว่า working directory ว่างแล้ว

```
AorJoa:scenario_git dekcom$ git status
On branch master
nothing to commit, working directory clean
```

ถ้ามีการแก้ไขไฟล์ readme.txt นี้อีก เราก็แค่ทำตามลำดับคือ git add readme.txt แล้วก็ git commit -m แล้วใส่ comment ไป เช่น “Edit#1 add line” วนอยู่แบบนี้ (ตัวอย่างข้างล่างผมชอบใช้ Sublime แก้ไฟล์แทน nano)



ถ้าจะดูประวัติการทำงานก็ได้จากคำสั่ง \$ `git log` ก็จะมีประวัติการ Commit โผล่ขึ้นมา

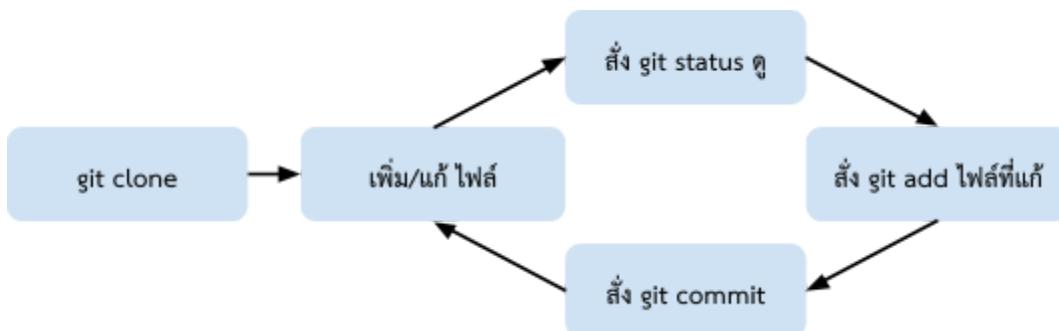
```
AorJoa:scenario_git dekcom$ git log
commit 2e7885772c1054ac54e84ce3be0eac30d90b7f7e
Author: "AorJoa" <aorjoa@i-aor.com>
Date: Thu Oct 23 10:32:06 2014 +0700

    Edit#1 add line

commit 8ba8bb97d319ca5f21d1ca4693573f6703bb2b73
Author: "AorJoa" <aorjoa@i-aor.com>
Date: Thu Oct 23 01:18:35 2014 +0700

Initial Git repository
```

และนั้นคือ Flow ปกติที่เราใช้ Git เริ่มเก็บประวัติการแก้ไขไฟล์ครับ ถ้าหากเรามี Repo อยู่แล้วหรือต้องการจะ Clone มาจากที่อื่นเช่น Github ก็แค่เปลี่ยนคำสั่ง \$ `git init` เป็น \$ `git clone` แล้วตามด้วยที่อยู่ Repo



\$ `git clone` <https://github.com/chanwit/spock-report-html.git> ใช้ Clone ข้อมูลจาก Remote Repo

```
AorJoa:git_basic dekcom$ git clone https://github.com/chanwit/spock-report-html.git
Cloning into 'spock-report-html'...
remote: Counting objects: 125, done.
remote: Total 125 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (125/125), 14.56 KiB | 13.00 KiB/s, done.
Resolving deltas: 100% (25/25), done.
Checking connectivity... done.
```

โค้ดที่ได้มาจะเป็นโค้ดที่ต้นทางมีการสร้างไว้แล้ว ส่วนการทำงานก็เหมือนกับการเริ่มโปรเจกใหม่ด้วยตัวเองครับ ต่างกันที่จะมีโค้ดและ Remote repo ติดมาแล้ว

สถานการณ์ที่ 2 : ฝากไว้ใน Remote repo เหรอ

โปรแกรมเมอร์รุ่นเด็กๆ ชอบใช้วิธีการบีบอัดไฟล์ เช่น .zip หรือ .tar วิธีการนี้เรียกว่าทำ Tarball และส่งกันไปทาง Email พอมันหลายๆครั้งเข้ากับพบร่วมกันไม่เวิร์ก เพราะตู้ไม่ออกรอแล้วว่ารุ่นไหนเป็นรุ่นไหน แ昏เนื้อที่เก็บ Email ก็จะเต็มอึด เลยมีการคิดค้น Version control ขึ้นมาใช้ซึ่ง Git ก็เป็นเวอร์ชั่นคอนโตรลแบบที่มี Remote repository ไว้เป็นตัวกลางให้คนที่ทำงานเดียวกันมาอัพเดทที่เดียวกันจะได้ไม่ว่าตัว

มีหลายเจ้าที่รับเป็นตัวแทนการเป็น Git remote repository เช่น Github.com, Gitlab.org หรือ Bitbucket.org ตัวอย่างนี้จะลองใช้ Remote repo ของ Github.com เพราะใช้งานง่าย มี Tutorial สอนใช้ที่ try.github.io เหมาะกับทั้งมือใหม่และมือเก่า และประโยชน์ในการใช้เป็นเหมือน Portfolio ในการสมัครงานได้ด้วยนะเออ (ประเดิมหนึ่งที่สำคัญคือผู้เป็นท้าทายมี nick Octocat ที่เป็นมาสคอตของ Github เขาหละ >_<) เอาเป็นว่าเริ่มต้นแบบ Step-by-Step กันเลยนะ

(1) ไปที่ <https://www.github.com> และก็สมัครสมาชิก

กรอกข้อมูล

กด Sign up for GitHub

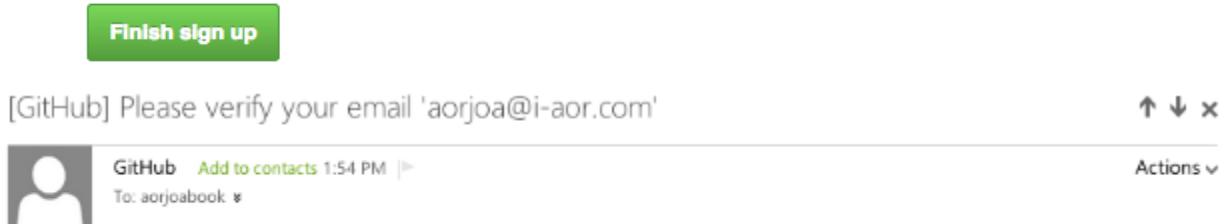
Sign up for GitHub

(2) เลือกแผนการใช้และแน่นอนผมเลือกตัวสุดท้ายเพราใช้ฟรี (แต่ถ้าใครยังเรียนอยู่ก็ขอเขาใช้ Micro Plan ได้นะครับฟรีหมดเลยขอมาแล้ว) ข้อดีของ Private repo คือมันสามารถให้เฉพาะคนที่สิทธิ์ถึงจะดูโค้ดได้

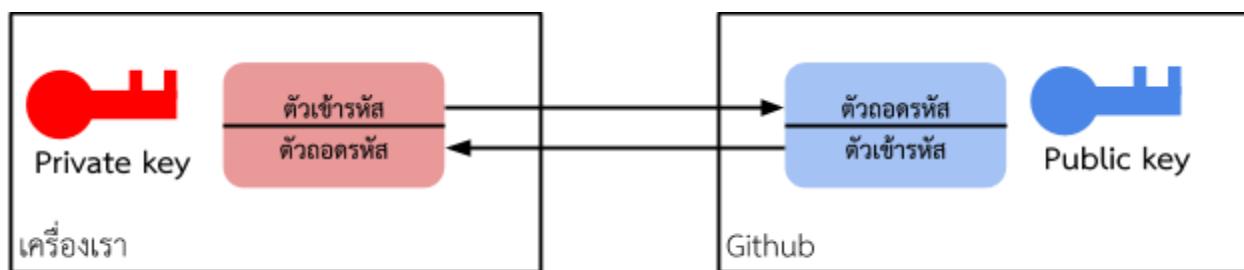
Choose your personal plan

Plan	Cost (view in THB)	Private repos	
Large	\$50/month	50	Choose
Medium	\$22/month	20	Choose
Small	\$12/month	10	Choose
Micro	\$7/month	5	Choose
Free	\$0/month	0	Chosen

- (3) กด Finish sign up ไปได้เลยจากนั้นก็ไปเช็ค Email ที่เรากรอกไปจะเห็นเมล์ที่ส่งมาจาก Github เพื่อขอยืนยันตัวตน ก็ให้เรากด link มา แล้วกด Confirm เป็นอันจบพิธีการสมัคร แต่ยังเหลือการเพิ่มไบร์บอร์องผู้ใช้กับสร้าง Repo



- (4) เพิ่มไบร์บอร์องผู้ใช้ ตอนที่เรา Connect เข้ามาเพื่อ Pull หรือ Push ข้อมูลเข้ามาที่ Github และ Github จะรู้ได้ว่าคุณเป็นตัวจริง ลำพังข้อกับ Email มันก็ปลอมกันได้ง่ายๆ ดังนั้น Github จึงใช้การรับรอง SSH key (Secure Shell key) ยืนยันตัวผู้ใช้ซึ่งเป็นแบบ [Public-key cryptography](#) เวลาสร้างจะได้ไฟล์สองไฟล์คือ Public key (นามสกุล .pub) กับ Private Key (ไม่มีนามสกุล) คีย์ทั้งสองสามารถใช้เข้ารหัสข้อมูลได้ แต่ถ้าจะเปิดอ่านต้องใช้ออกใบที่เป็นคู่กัน ถ้าใช้ Public key ในการเข้ารหัสต้องถอดรหัสด้วย Private Key เท่านั้น และในทางกลับกันถ้าเข้ารหัสด้วย Private Key ต้องถอดรหัสด้วย Public key เท่านั้น จะใช้ตัวที่เข้ารหัสตัวมันเองถอดรหัสตัวมันเองໄมได้ ดังนั้นวิธีการนี้ทำให้บอร์งสิทธิ์ว่าผู้รับและผู้ส่งเป็นคนที่เราต้องการจริงๆ เพราะจะมีแค่ผู้รับและผู้ส่งเท่านั้นที่มีคีย์เปิดอ่านของกันและกันได้



เราจะส่ง Public key ไปเก็บไว้ที่ Github และเก็บ Private key ไว้ที่เครื่องเรา วิธีการสร้าง SSH key ให้สั่ง
\$ `ssh-keygen -t rsa -C "aorjoa@i-aor.com"`

Enter file in which to save the key (/Users/dekcom/.ssh/id_rsa): กดปุ่ม Enter ผ่านไปได้เลย

/Users/dekcom/.ssh/id_rsa already exists.

Overwrite (y/n)? ไฟล์เบร์บรองมีอยู่แล้ว ตอบ y ให้มันเขียนทับเป็นตัวใหม่ (ของใครไม่มีก็จะไม่ถูกถาม)

Enter passphrase (empty for no passphrase): ใส่ Passphrase (คำลับที่เอาไว้เปิดใช้เบร์บรองป้องกันเพื่อมีคนได้เบร์บรองเร่อไป) คำนี้จะไม่แสดงออกทางหน้าจอ ถ้าไม่อยากใส่ก็กด Enter ผ่านไปเลย

Enter same passphrase again: ใส่ Passphrase อีกรอบ

--- ข้างล่างบอกว่าอยู่ของไฟล์ทั้งสองคือ Private key (identification) และ Public key ---

Your identification has been saved in /Users/dekcom/.ssh/id_rsa.

Your public key has been saved in /Users/dekcom/.ssh/id_rsa.pub.

```
AorJoa:~ ssh dekcom$ ssh-keygen -t rsa -C "aorjaa@i-aor.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/dekcom/.ssh/id_rsa):
/Users/dekcom/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/dekcom/.ssh/id_rsa.
Your public key has been saved in /Users/dekcom/.ssh/id_rsa.pub.
The key fingerprint is:
66:e4:1d:4c:1e:49:76:0e:03:3e:54:9a:c3:4e:26:57 aorjaa@i-aor.com
The key's randomart image is:
+--[ RSA 2048]----+
|   o+E..   |
|   + 0.*   |
|   . % + .  |
|   0 + .   |
|   S .     |
|   o        |
|           |
|           |
+-----+
```

จากนั้นก็เปิดดู Public key สำ

§ cat ~/ssh/id_rsa.pub

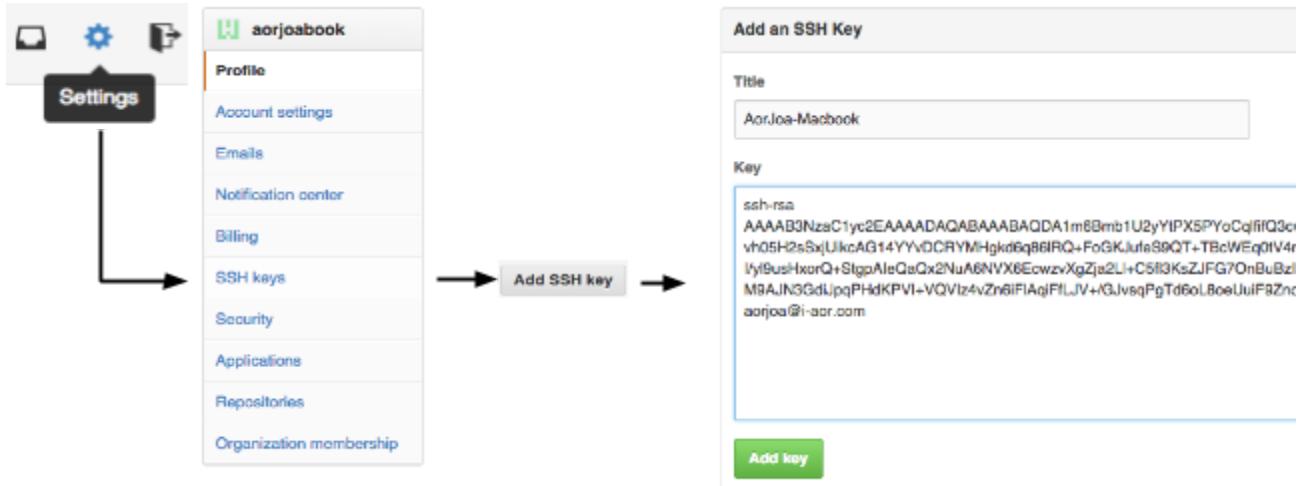
เพื่อดูข้อมูลข้างในไฟล์ แล้วก็

ก็อปปี้ข้อความข้างในออกมาไว้

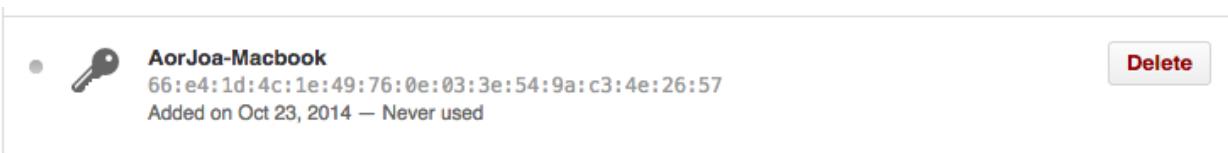
รอเอาไปเพิ่มใน Github

```
AorJoa:~ ssh dekcom$ cat ~/.ssh/
Backup/      id_rsa      id_rsa.pub      known_hosts
AorJoa:~ ssh dekcom$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQDA1m6Bmb1U2yYtPX5PYoCqlfifQ3c
Qe1ViHYfIX4YHUrjh05H2sSxjUikcAG14YYvDCRYMHgkd6q861RQ+FoGKJuFeS9QT+T
ogr6GFA/CFFP8kfX4D6ipoq0bI/y19usHxorQ+StgpAIeQaQx2NuA6NVX6EcwzvXgZj
70nBuBzIuZX9UHTrjLSkWHRGPQjrDUdeFd/zM9AJN3GdiJpqPHdKPVI+VQVIz4vZn6i
qPgTd6oL8oeUiF9ZndyPa4PF92IuqclByqfE0Fvcks2pYnh aorjaa@i-aor.com
```

จากนั้นก็ไปที่หน้าเว็บของ github.com เพื่อเพิ่ม Public key ของเราที่สร้างเมือกี้ให้ github รู้จัก กดปุ่ม Setting > SSH Keys > กรอก Title (ปกติจะใส่ชื่อเครื่องคอม) > วางคีย์ที่ก้อบปี้มาจากในช่อง Key > กด Add key

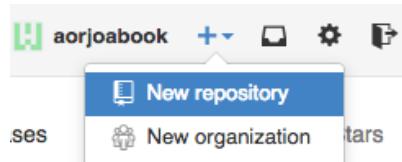


เป็นอันเรียบร้อย



(5) สร้าง Repository

> กด New repository



> ตั้งชื่อ Repo



> คำอธิบาย Repo ไม่สำคัญ

Description (optional)

Basic git tutorial

> เลือกว่า Repository จะเป็นแบบ Public หรือ Private



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

> Git เตรียมไฟล์ README ,ไฟล์ .gitignore (ไฟล์ที่เอาไว้บอกว่าไฟล์ประเภทไหนที่จะไม่ถูกเก็บใน Git repo) และไฟล์คำอธิบาย License (ถ้าไม่เอาไว้ไม่ต้องเลือกนะ)

Initialize this repository with a README

This will allow you to git clone the repository immediately. Skip this step if you have already run git init locally.

Add .gitignore: **None**

Add a license: **None**



จากนั้นกด Create repository

Create repository

ในหน้าคัดมา กดที่ SSH แล้วก็ปี้ URL ของ Repo ไว้

HTTP

SSH

git@github.com:aorjoabook/basic_git.git



(6) คลับไปที่หน้าจอ Command line เพิ่ม Remote repository เข้าใน Git โดยสั่ง

\$ git remote add origin git@github.com:aorjoabook/basic_git.git

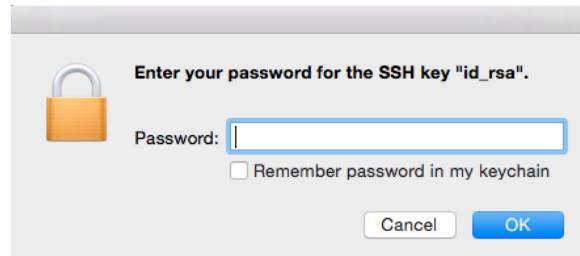
AorJoa:scenario_git dekcom\$ git remote add origin git@github.com:aorjoabook/basic_git.git

(7) ลอง Push ข้อมูลไปเก็บที่ Repo ที่เราสร้าง

มา สั่ง \$ git push origin master

dekcom\$ git push origin master

ถ้าใส่ Passphrase ไว้จะมี Dialog ขึ้นมาตาม



หลังจากที่เราใส่ Passphrase เรียบร้อยก็กด OK ไป (บาง OS อาจจะสามารถใน Command line เลยก็ได้)

```
AorJoa:scenario_git dekcom$ git push origin master
Saving password to keychain failed
Identity added: /Users/dekcom/.ssh/id_rsa (/Users/dekcom/.ssh/id_rsa)
Counting objects: 6, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 525 bytes | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To git@github.com:aorjoabook/basic_git.git
 * [new branch]      master -> master
```

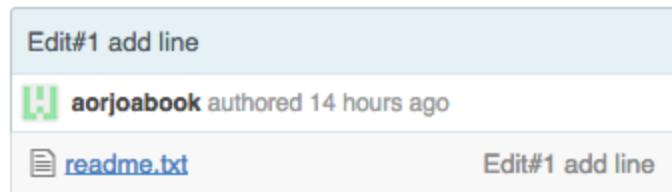
จากนั้น Git ก็จะส่งข้อมูลขึ้นไปที่ github.com ตาม Repo ที่เราบอก ถ้าเสร็จแล้วก็เปิดดูได้ที่หน้าแรก เลือก Repositories จากนั้นก็เลือกเข้าไปที่ Repo ที่ Push ขึ้นไป ก็จะเห็นความเปลี่ยนแปลงที่เกิดขึ้น เลือก basic_git

[basic_git](#)

Basic git tutorial

Updated an hour ago

เปิดไฟล์ readme.txt



ด้านในไฟล์ readme.txt จะแสดงเวอร์ชันล่าสุด



แค่นี้เราก็จะมี Remote repo ของ Github ไว้ใช้แล้ว ส่วนใครที่ต้องการใช้ของเจ้าอื่น ก็มีวิธีทำคล้ายๆกันครับ ไม่ยากແน່องครับ เพราะว่ายังไนในระยะได้แล้ว จะให้ไปว่ายในคลองก็ไม่มีปัญหา ใช้ [Github](https://github.com) เป็นแล้ว ไปใช้ [Bitbucket](https://bitbucket.org) หรือ [GitLab](https://gitlab.com) ก็สบายมาก (คือมันคล้ายกันมากเลยครับ อย่างการเซ็ต SSH keys เหมือนยังกะก็อปกันมา)

คำแนะนำ : ลองเล่นใหม่ๆ บารมียังไม่แก่กล้า แนะนำให้ Push ขึ้นไปแค่ Branch ซึ่ง master ครับ
(เพราะอาจจะเกิดสับสน) \$ `git push origin master` คือส่ง master บนเครื่องเราไปเป็น master ของ Remote repo ที่ชื่อว่า origin (Remote repo มีได้หลายที่ เช่นใช้ github ด้วย ใช้ gitlab ด้วย)

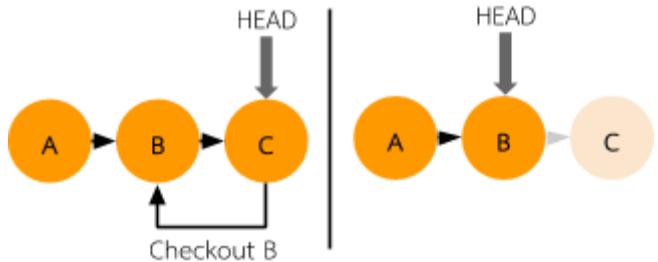
สถานการณ์ที่ 3 : กลับมาเหมือนเดิมนะ,,ข้อมูล

การย้อนเวอร์ชันข้อมูลกลับ มีวิธีอยู่ 3 อย่างที่ใช้กันให้เห็นบ่อยๆ ก็คือ Checkout, Revert และ Reset ครับ คำแนะนำคือก่อนเราจะทำอะไรให้สั่ง \$ git status ดูก่อน ถ้ามีการแก้ไขข้อมูลก็ให้ Commit บันเก็บไว้ก่อน เรามักจะทำงานกันใน Working directory ที่อยู่สถานะ Clean เพื่อลดการ Conflict หรือสับสนเวลา�้อนกลับไปกลับม

```
AorJoa:scenario_git dekcom$ git status
On branch master
nothing to commit, working directory clean
```

> Checkout

Checkout เป็นการมองย้อนกลับไปจาก ตำแหน่งปัจจุบัน ลักษณะคล้ายๆเป็น Branch หนึ่ง ที่มี ข้อมูลเป็นไฟล์รุ่นที่อยู่ใน Commit นั้นๆ และถ้าเรา Checkout กลับมาที่ Branch master ก็จะเป็นข้อมูลปกติ



(1) ลองสั่ง \$ git log ดูประวัติ

```
AorJoa:scenario_git dekcom$ git log
commit 2e7885772c1054ac54e84ce3be0eac30d90b7f7e
Author: "AorJoa" <aorjoa@i-aor.com>
Date: Thu Oct 23 10:32:06 2014 +0700

Edit#1 add line

commit 8ba8bb97d319ca5f21d1ca4693573f6703bb2b73
Author: "AorJoa" <aorjoa@i-aor.com>
Date: Thu Oct 23 01:18:35 2014 +0700

Initial Git repository
```

(2) ลองกลับไปที่คอมมิท 8ba8bb9... สั่ง \$ git checkout 8ba8bb9

```
AorJoa:scenario_git dekcom$ git checkout 8ba8bb9
Note: checking out '8ba8bb9'.
```

(3) ใช้ \$ git log ดูประวัติอีกที

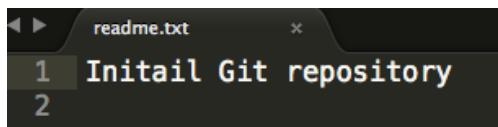
```
AorJoa:scenario_git dekcom$ git log
commit 8ba8bb97d319ca5f21d1ca4693573f6703bb2b73
Author: "AorJoa" <aorjoa@i-aor.com>
Date: Thu Oct 23 01:18:35 2014 +0700

Initial Git repository
```

- (4) ถ้าลองใช้ `$ git branch` ดูจะเห็นว่ามี Detached branch ซึ่งเป็น branch ลอยๆขึ้นมา ถ้า checkout ไป branch จริงตัวนี้จะหายไป

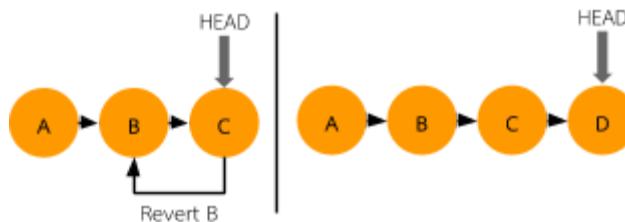
```
AorJoa:scenario_git dekcom$ git branch
* (detached from 8ba8bb9)
  master
```

ถ้าดูข้อมูลก็จะเห็นว่ามันกลับไปเป็นรุ่นที่เราต้องการแล้ว



> Revert

Revert เป็นการคืนค่า Commit โดยจะสร้าง Commit ใหม่ ไม่ทับของเดิม เช่นถ้าเรารอ已久 Revert Commit B มันก็จะสร้าง Commit D ขึ้นมาใหม่โดยต่อจาก Commit C ซึ่งเป็น Commit ล่าสุด ข้อนำมาการใช้คือควรจะ Revert เป็นลำดับ



ถ้าอยากรีvert Commit A เราอาจจะสั่งให้ Revert ตั้งแต่ Commit A ไปจนถึง HEAD ป้องกันการ Conflict (หรือจะแก้ Conflict เองก็ได้นะ ถ้ามีการ Conflict)

- (1) ลองดู Log ก่อน สั่ง `$ git log`

```
AorJoa:scenario_git dekcom$ git log
commit 2e7885772c1054ac54e84ce3be0eac30d90b7f7e
Author: "AorJoa" <aorjua@i-aor.com>
Date: Thu Oct 23 10:32:06 2014 +0700

Edit#1 add line

commit 8ba8bb97d319ca5f21d1ca4693573f6703bb2b73
Author: "AorJoa" <aorjua@i-aor.com>
Date: Thu Oct 23 01:18:35 2014 +0700

Initial Git repository
```

- (2) สั่ง `$ git revert 8ba8bb9..HEAD` (ที่ ..HEAD ไว้กันการ Conflict อย่างที่บอก)

```
AorJoa:scenario_git dekcom$ git revert 8ba8bb9..HEAD
```

พอเราสั่งอาจจะมีหน้าจอ Editor ขึ้นมาตามว่าจะแก้ประวัติ Commit ใหม่ ให้กดปุ่ม `:q!` และ Enter ไปเลย หรือจะไม่เปิด Editor สั่ง `$ git revert --no-edit 8ba8bb9..HEAD`

```
Revert "Edit#1 add line"
This reverts commit 2e7885772c1054ac54e84ce3be0eac30d90b7f7e.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#       modified:   readme.txt
```

- (3) ถ้าลองสั่ง \$ `git log` ดูอีกทีจะเห็นว่ามีการ Revert ข้าม commit 2e7885 ไป

```
AorJoa:scenario_git dekcom$ git log
commit 1d1a5499d8dc04117e9ad6cdc8c097308891d1b2
Author: "AorJoa" <aorjoadi-aor.com>
Date: Sun Oct 26 13:01:24 2014 +0700

    Revert "Edit#1 add line"

    This reverts commit 2e7885772c1054ac54e84ce3be0eac30d90b7f7e.

commit 2e7885772c1054ac54e84ce3be0eac30d90b7f7e
Author: "AorJoa" <aorjoadi-aor.com>
Date: Thu Oct 23 10:32:06 2014 +0700

    Edit#1 add line

commit 8ba8bb97d319ca5f21d1ca4693573f6703bb2b73
Author: "AorJoa" <aorjoadi-aor.com>
Date: Thu Oct 23 01:18:35 2014 +0700

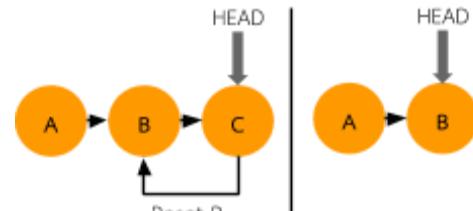
Initial Git repository
```

- (4) ลองดูในไฟล์ `readme.txt` จะเห็นว่าข้อมูลเปลี่ยนกลับเป็นรุ่นเรารอยากย้อนไปแล้ว

```
readme.txt
1 Initail Git repository
2
```

> Reset

การใช้งานคำสั่ง `Reset` เป็นการย้อนกลับและเขียนทับประวัติการทำงานด้วย ยกตัวอย่างเช่นถ้าผมพัฒนาไฟล์ `readme.txt` แล้วเข้าใจ Spec ของ Software ผิดพลาดไปแบบคนละทิศทางเลย ผมย้อนดูพบว่ามี 2 Commit ล่าสุดที่ผมทำผิด และผมไม่ต้องการมันเลย ลังบันทึกซึ่งจะย้อนกลับและลบประวัติการทำงานที่ผมทำมา 2 Commit ด้วย ให้นำมาใช้อีกแล้ว เพื่อเพื่อนมาดูจะได้ไม่งงว่าผม Commit อะไรเข้าไป ซึ่งก็มีสองแบบหลักๆ คือ `--soft` จะไม่ย้อนกลับไฟล์ที่เรากำลังแก้อยู่ กับ `--hard` ย้อนกลับทุกอย่างแม้แต่ไฟล์ที่เรากำลังแก้ไขอยู่ ไปเป็นรุ่นปลายทางที่เรารอยากได้



- (1) ดู `log` สั่ง \$ `git log --oneline` (ดู `log` แบบบรรทัดเดียว)

```
AorJoa:scenario_git dekcom$ git log --oneline
5880ae7 Revert "Edit#1 add line"
2e78857 Edit#1 add line
8ba8bb9 Initial Git repository
```

- (2) สั่ง \$ `git reset --hard`

```
AorJoa:scenario_git dekcom$ git reset --hard 8ba8bb9
HEAD is now at 8ba8bb9 Initial Git repository
```

- (3) ข้อมูลเปลี่ยนไปแล้ว

```
◀ ▶ readme.txt *
1 Initail Git repository
2
```

(4) ดู Log อีกครั้งสั้ง \$ `git log --oneline` จะเห็นว่า Log ก็เปลี่ยนไปด้วย

```
AorJoa:scenario_git dekcom$ git log --oneline
8ba8bb9 Initial Git repository
```

สถานการณ์ที่ 4 : เจอ Conflict

ถ้าใช้ Version control ตั้งแต่เริ่มจนจบโปรเจกแล้วไม่เกิด Conflict นี่นับได้ว่าชาติก่อนทำบุญใหญ่ระดับสร้างศาลาหรือมหาวิหารมาเลยที่เดียว เพราะปกติจะเห็น Conflit กันอยู่บ่อยๆ โดย Conflict ก็เกิดจากที่เราแก้ไขข้อมูลตำแหน่งเดียวกันทำให้ git สับสนว่าเราต้องการเอาโค้ดชุดไหนกันแน่ ผูกตัวอย่างว่าถ้าเพื่อนแก้โค้ดใน Branch master (Remote repo) และ Push ขึ้น Github มา ขณะเดียวกันผู้แก้ไขไฟล์ที่ Branch master บนเครื่องของ (Local repo) บังเอิญว่าแก้ไฟล์ที่ตำแหน่งเดียวกันซะด้วยทำให้ Git สับสนว่าอื้จะเอาเวอร์ชันไหนกันแน่ ว่าแล้วก็มาลองกันเลย

(1) อันดับแรกผมจะขอบเข้าไปแก้ข้อมูลใน Github ทำนองว่าเพื่อนแก้แล้ว Push ขึ้นมา และจะแก้ไฟล์บนเครื่องที่ตำแหน่งเดียวกัน จากนั้นก็ Commit เก็บไว้ (สังเกตุบรรทัดที่ 3) และ ทำที่ Branch เดียวกันคือ Branch master

The screenshot shows two side-by-side terminal windows. The left window is titled 'Github' and the right is 'My Computer'. Both show a log of commits:

- Github:**

```
1 Initail Git repository
2 Edit 1st time add this line
3 ===> Github Edit (Remote repo) <===
4
```
- My Computer:**

```
1 Initail Git repository
2 Edit 1st time add this line
3 ===> My Computer Edit (Local repo) <===
4
```

Below the terminals are two buttons:

- Commit changes** (in green)
- \$ git commit -am 'Update from my computers (Local repo)'**

At the bottom, it shows the current branch: **branch: master**.

คำสั่ง \$ `git commit -am` มันคือการรวม `git add` (เฉพาะไฟล์ที่เคย `add` แล้ว) กับ `git commit` ไว้ในคำสั่งเดียว

(2) ลองสั่ง \$ `git pull origin master` ดู (คงจำได้นะ ว่า `git pull` คือ `git fetch + git merge` นั่นเอง)

```
AorJoa:scenario_git dekcom$ git pull origin master
From github.com:aorjoabook/basic_git
 * branch           master    -> FETCH_HEAD
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.
```

จะเห็นว่ามี Conflict เกิดขึ้น เพราะแก้ไฟล์ที่ตำแหน่งเดียวกัน

(3) ลองเปิดดูที่ Editor

```

1 Initail Git repository
2 Edit 1st time add this line
3 <<<<< HEAD
4 ==> My Computer Edit (Local repo) <==> จาก HEAD
5 =====
6 ==> Github Edit (Remote repo) <==> จาก Commit
7 >>>>> 9a570f68608e09ad7a8d5410658c1db8adc0e23a 9a570f...
8

```

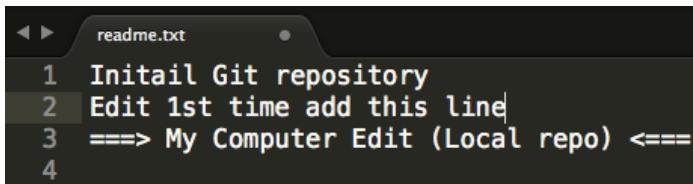


```

1 Initail Git repository
2 Edit 1st time add this line ส่วนที่ผิดชอบ
3 <<<<< HEAD <-->
4 ==> My Computer Edit (Local repo) <==> ตัวอย่างนี้ผิดจะเก็บ HEAD ไว้
5 =====
6 ==> Github Edit (Remote repo) <==>
7 >>>>> 9a570f68608e09ad7a8d5410658c1db8adc0e23a
8

```

จะเห็นว่ามี 2 Commit ที่ Conflict กันครับ คือ จาก HEAD ในเครื่องเรา และ Commit ที่มีเลข 9a570f... ในตัวอย่างนี้ผิดจะเก็บ HEAD ไว้



```

readme.txt
1 Initail Git repository
2 Edit 1st time add this line
3 ==> My Computer Edit (Local repo) <==>
4

```

จากนั้นก็สั่ง Commit เก็บไว้

```

AorJoa:scenario_git dekcom$ git commit -am 'After solv conflict'
[master 8456e87] After solv conflict

```

(4) ลองดู Log

```

AorJoa:scenario_git dekcom$ git log --oneline
8456e87 After solv conflict
af13edd Update from my computers (Local repo)
9a570f6 Update from github (Remote repo)
2e78857 Edit#1 add line
8ba8bb9 Initial Git repository

```

อันที่จริงแล้ว Git มี Tools ที่เอาไว้ช่วย Merge conflict ได้ในกรณีที่มี Conflict เกิดขึ้นสามารถเลือกได้ว่าจะเอาอันไหน และถ้ามี Conflict หลายๆ อันมันก็จะเลือกมาให้เราดูได้ด้วย แต่ปกติแล้วใช้ Editor ปกติก็เอาอยู่

สถานการณ์ที่ 5 : เพิ่ม Alias ย่อคำสั่งให้ใช้ง่ายๆ

Alias หรือนามแฝง เป็นการย่อคำสั่งที่ใช้บ่อยๆ ให้ใช้ง่ายๆ ดูตัวอย่างเลยแล้วกัน

ถ้ามี \$ git log --graph --decorate --pretty=oneline --abbrev-commit --all สร้าง alias ชื่อ lol โดยสั่ง \$ git config --global --add alias.lol "log --graph --decorate --pretty=oneline --abbrev-commit --all"

```
AorJoa:scenario_git dekcom$ git config --global --add alias.lol
"log --graph --decorate --pretty=oneline --abbrev-commit --all"
```

```
AorJoa:scenario_git dekcom$ git lol
*   8456e87 (HEAD, master) After solv conflict
|\ 
| * 9a570f6 (origin/master) Update from github (Remote repo)
* | af13edd Update from my computers (Local repo)
|/
* 2e78857 Edit#1 add line
* 8ba8bb9 Initial Git repository
```

คำสั่ง \$ git log --graph ช่วยให้เห็นการเข้าออกของ Commit

(cc : <http://uberblo.gs/2010/12/git-lol-the-other-git-log>)

สถานการณ์ที่ 6 : Fast Forward กับ 3-way Merge

> Fast forward เกิดขึ้นเวลาเราใช้คำ

สั่ง Merge แล้ว Git พบว่ามันมีประวัติ

มาจากการเดียวกัน มันก็จะพยายาม

ทำให้เป็นเส้นตรงแนวเดียวกัน

> 3-way Merge เป็นการ Merge ที่

ประวัติรวมกัน และสร้าง Commit

ใหม่ขึ้นมา จากรูปจะเห็นว่าสีแดง 3

จุด ได้วางสีฟ้าเป็นที่มาของ 3-way

Merge

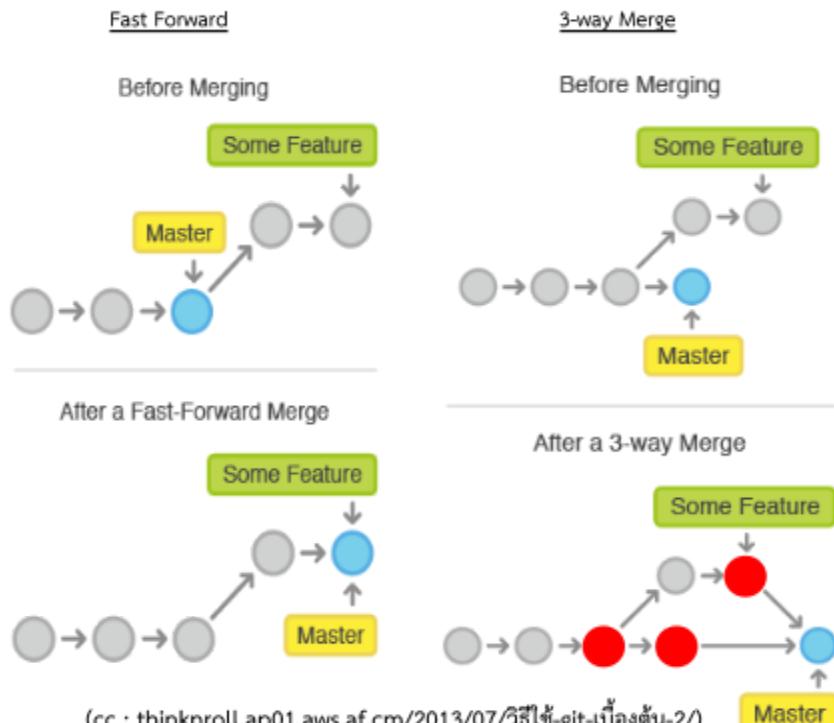
เนื่องจาก 3-way Merge แบบนี้เห็น

ภาพการเข้าออกได้ดี จึงแนะนำให้ใช้

โดยความสามารถบังคับให้เป็น 3-way

Merge โดยสั่ง \$ git merge --no-ff

ตามด้วยชื่อ branch



สถานการณ์ที่ 7 : จัดการ Branch

เรามักจะสร้าง Branch เพื่อพัฒนาไฟล์เอกสารหรือแก้ไข และเมื่อเราพัฒนาไฟล์เอกสารเสร็จแล้ว เราอาจจะไม่อยากได้ Branch แล้ว ตามจำนวนเสร็จนาฬิกาโคทึก เสร็จไฟล์เอกสารเราก็ลบ Branch ที่ไม่ต้องการทิ้งไปเพื่อไม่ให้มันเบอจะเกินไปจนสับสนว่า Branch ไหนเป็น Branch ไหน ดังนั้นมาดูวิธีสร้างและลบ Branch กัน

> เพิ่ม Local branch

การเพิ่ม Branch บน Local ทำได้ง่ายๆ ด้วย \$ git checkout -b 'bugfix' และ \$ git branch ดู

```
AorJoa:scenario_git dekcom$ git checkout -b 'bugfix'
Switched to a new branch 'bugfix'
AorJoa:scenario_git dekcom$ git branch
* bugfix
  master
```

> เพิ่ม Remote branch

หลังจากที่เพิ่ม Local branch แล้วผมอยากรสั่ง Branch นี้ไปไว้ที่ Remote repo ด้วย วิธีการคือผมแก้ไฟล์
จากนั้น Commit ไฟล์ไว้ให้เป็นรุ่นปัจจุบัน \$ git commit -am 'bugfix branch'

```
readme.txt
1 Initial Git repository
2 Edit 1st time add this line
3 ===> My Computer Edit (Local repo) <==
4 -Commit form bugfix branch-
5 |
```

```
AorJoa:scenario_git dekcom$ git commit -am 'bugfix branch'
[bugfix 63beb1a] bugfix branch
 1 file changed, 1 insertion(+), 4 deletions(-)
```

ถัดมาแก้สั่ง \$ git push origin bugfix:issue ซึ่งคำสั่งนี้บอกว่าเอา Branch bugfix บนเครื่องเราไปเป็น Branch issue บน Remote repo ที่ชื่อว่า origin ปกติเราสั่ง \$ git push origin master มันก็ \$ git push origin master:master ผังข้ายคือชื่อ Branch บนเครื่องเราส่วนผังขวาคือ Branch บน Remote repo แต่ถ้าชื่อเหมือนกันรวมมาเขียนตัวเดียวได้

```
AorJoa:scenario_git dekcom$ git push origin bugfix:issue
Counting objects: 9, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 936 bytes | 0 bytes/s, done.
Total 9 (delta 2), reused 0 (delta 0)
To git@github.com:aorjoabook/basic_git.git
 * [new branch]      bugfix -> issue
```

ถ้าดูใน Github ซึ่งเป็น Remote repo (ตัว origin) จะเห็นว่ามี Branch ชื่อ issue เพิ่มขึ้นมา และถ้ากดมา Branch นี้ก็จะเห็นข้อมูลเป็นข้อมูลใน Branch issue (ซึ่งก็คือ Branch bugfix ใน Local repo นั่นแหละ) แต่ถ้ากดมา master ก็จะเห็นข้อมูลใน Branch master

```
Branches
✓ Issue
master
readme.txt
Initail Git repository
Edit 1st time add this line
====> My Computer Edit (Local repo) <===
-Commit form bugfix branch-
```

> ลบ Local branch

การลบ Local branch ก็ทำได้ง่ายๆ แต่ต้องไปที่ Branch อื่นก่อน เพราะตัวมันเองจะลบมันเองไม่ได้

```
AorJoa:scenario_git dekcom$ git branch -D bugfix
error: Cannot delete the branch 'bugfix' which you are currently on.
```

จากนั้น Checkout ไป master

```
AorJoa:scenario_git dekcom$ git checkout master
Switched to branch 'master'
```

```
AorJoa:scenario_git dekcom$ git branch
  bugfix
* master
```

จากนั้นจึงสั่งลบด้วย \$ `git branch -D bugfix`

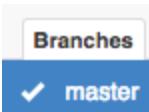
```
AorJoa:scenario_git dekcom$ git branch -D bugfix
Deleted branch bugfix (was 63beb1a).
```

> ลบ Remote branch

สั่ง \$ `git push origin --delete issue` เพื่อลบ Branch issue ใน Remote repo

```
AorJoa:scenario_git dekcom$ git push origin --delete issue
To git@github.com:aorjoabook/basic_git.git
  - [deleted]           issue
```

ใน Github ก็จะเหลือแค่ Branch master เพียงตัวเดียว



มือิคำสั่งหนึ่งที่เอาไว้ลบคือ \$ `git push origin :issue` ได้ผลเหมือนกัน (เหมือนไม่ส่ง Branch อะไร ไปไว้ที่ Remote repo ก็คือการลบันทึกของ)

สถานการณ์ที่ 8 : แก้ไข Commit หรือ History

กรณีที่เราทำ Commit ล้วน แล้วต้องการแก้ไขโดยรวมเอาตัวใหม่กับตัวเก่าเข้าไว้ด้วยกัน เช่น ผู้มีcommit แก้ไขไฟล์ README.txt ผู้มาเข้าใจว่าผู้มีcommit ทำถูกหมดแล้วก็เลย Commit เก็บไว้ พ่อเพื่อนมาเห็น เหี้ยย! ทำพิดนี่หว่า แต่ผู้มีcommit ไปแล้ว เพื่อไม่ให้เสียฟอร์มจนเกินไปและลดความวุ่นวายหลาย Commit ผู้มีcommit ร่วมมั่นไว้ด้วยกันเลย (ลองนึกๆ สมมติว่าผู้มีcommit ครั้งแรกพร้อมกับ Comment ว่า “Done” แต่เพื่อนมาเห็นว่ามันผิด หลังจากที่ผู้มีcommit แก้แล้วจะ Commit ว่า “Done#2” มันก็คงแปลกดีอยู่ที่งานจะเสร็จสองครั้ง)

> กรณีที่ Commit ต่อ กัน

กรณีที่เราต้องการจะแก้ไข Commit ก่อนหน้านี้ (เป็น Commit ล่าสุดนะครับ) \$ `git commit --amend -m 'Edit comment'` ตัวอย่างนี้ผู้มีcommit จะแก้ Comment ใน Commit ล่าสุด

```
AorJoa:scenario_git dekcom$ git log --oneline
8456e87 After solv conflict
af13edd Update from my computers (Local repo)
9a570f6 Update from github (Remote repo)
2e78857 Edit#1 add line
8ba8bb9 Initial Git repository
AorJoa:scenario_git dekcom$ git commit --amend -m 'Edit comment'
[detached HEAD 9730421] Edit comment
AorJoa:scenario_git dekcom$ git log --oneline
9730421 Edit comment
af13edd Update from my computers (Local repo)
9a570f6 Update from github (Remote repo)
2e78857 Edit#1 add line
8ba8bb9 Initial Git repository
```

> แก้ History โดยเลือก Commit เพื่อ Rebase

Git เปิดโอกาสให้เราได้แก้ไขประวัติการเก็บได้ด้วย โดยใช้ \$ `git rebase -i` ซึ่งเป็นการ Rebase แบบมีปฏิสัมพันธ์กับผู้ใช้ ผู้ใช้สามารถเลือกได้ว่าจะเอา Commit ไหน ไม่เอา Commit ไหน หรือจะแก้ลำดับการ Commit ก็ได้ (มีเงื่อนไขคือมันต้องอยู่บน base เดียวกัน)

(1) ลองดู Log สั้น \$ `git lol`

```
AorJoa:scenario_git dekcom$ git lol
* b94a068 (HEAD, master) before rebase interactive
* 8456e87 After solv conflict
| \
| * 9a570f6 (origin/master) Update from github (Remote repo)
* | af13edd Update from my computers (Local repo)
| /
* 2e78857 Edit#1 add line
* 8ba8bb9 Initial Git repository
```

(2) สั่ง \$ git rebase -i 8ba8bb9

```
AorJoa:scenario_git dekcom$ git rebase -i 8ba8bb9
```

จะเด้งหน้าจอให้เราเลือก Commit จาก base ที่ต้องการคือ 8ba8bb9 (สังเกตว่าไม่มี Commit 8456e87 เพราะมันคนละ base)

```
pick 2e78857 Edit#1 add line
pick af13edd Update from my computers (Local repo)
pick 9a570f6 Update from github (Remote repo)
pick b94a068 before rebase interactive
```

โดยผมจะลบ Commit af13edd ออก ก็เอา Cursor ไปชี้ที่มัน กดปุ่ม dd บนคีย์บอร์ด (กด d สองครั้ง)

```
pick 2e78857 Edit#1 add line
pick 9a570f6 Update from github (Remote repo)
pick b94a068 before rebase interactive
```

จากนั้นก็กดปุ่ม :wq เพื่อบันทึกและออก

```
:wq
```

Git ก็จะพยายาม Rebase ถ้าผ่านก็แล้วไป ก็กรณีมีข้อความนี้อยู่ มี Conflict ติดมา

```
AorJoa:scenario_git dekcom$ git rebase -i 8ba8bb9
error: could not apply b94a068... before rebase interactive

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".
Could not apply b94a068b84df7e27766fd7e7ba53831d88f52011... before rebase interactive
```

ผมเลยสั่ง \$ git status ดูว่าไฟล์ไหนที่มีนติด Conflict

```
AorJoa:scenario_git dekcom$ git status
rebase in progress; onto 8ba8bb9 | กำลังติดขั้นตอนการ Rebase อよ
You are currently rebasing branch 'master' on '8ba8bb9'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git reset HEAD <file>..." to unstage)
  (use "git add <file>..." to mark resolution)

    both modified:  readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

จะเห็นไฟล์ readme.txt แดงแล้ว พร้อมกับบอกว่า both modified คือมันสับสนว่าจะเอาอันไหนแน่ จากนั้นผมก็ไปดูไฟล์ readme.txt ที่ติด Conflict แล้วก็แก้ให้ดงตามตามต้องการ

```
readme.txt
1 Initail Git repository
2 Edit 1st time add this line
3 ===> Github Edit (Remote repo <==
4 --- Commit before rebase interactive --
5
```

จากนั้นก็ Commit เก็บไว้

```
AorJoa:scenario_git dekcom$ git commit -am 'Conflict fixed'
[detached HEAD a1417f1] Conflict fixed
 1 file changed, 1 insertion(+)
```

แล้วก็สั่ง rebase ต่อ จากคำสั่ง \$ [git rebase --continue](#)

```
AorJoa:scenario_git dekcom$ git rebase --continue
Successfully rebased and updated refs/heads/master.
```

ถ้าสั่ง \$ [git lol](#) หรือ \$ [git log](#) ดูก็จะเห็นประวัติมันเปลี่ยนไปแล้ว

```
AorJoa:scenario_git dekcom$ git lol
* a1417f1 (HEAD, master) Conflict fixed
* 9a570f6 (origin/master) Update from github (Remote repo)
* 2e78857 Edit#1 add line
* 8ba8bb9 Initial Git repository
```

สถานการณ์ที่ 9 : ดู Difference

โดยทั่วไป Version control จะมี Option ที่จะสามารถใช้ดูความแตกต่างของไฟล์ในแต่ละเวอร์ชันได้ถือว่า เป็นฟังก์ชันพื้นฐานเลยทีเดียว และเช่นกัน Git ก็มีความสามารถนี้ การใช้งานก็ไม่ยากแค่สั่ง \$ [git diff](#) ต้นทาง ปลายทาง เช่น

เทียบระหว่าง Commit ล่าสุด กับปัจจุบัน \$ [git diff](#)

```
AorJoa:scenario_git dekcom$ git diff
diff --git a/readyou.txt b/readyou.txt
index f0cf66f..88abc6c 100644
--- a/readyou.txt
+++ b/readyou.txt
@@ -2,3 +2,5 @@ Initail Git repository
 Edit 1st time add this line
 ===> Github Edit (Remote repo <==
 --- Commit before rebase interactive --
+
+Add more line of code >_<'
```

เมื่อยกระหว่าง Branch \$ [git diff master dev](#)

```
AorJoa:scenario_git dekcom$ git diff dev master
diff --git a/readyou.txt b/readyou.txt
index f70be54..f0cf66f 100644
--- a/readyou.txt
+++ b/readyou.txt
@@ -2,4 +2,3 @@ Initail Git repository
Edit 1st time add this line
====> Github Edit (Remote repo) <===
--- Commit before rebase interactive --
-Add line in branch dev
AorJoa:scenario_git dekcom$ git diff master dev
diff --git a/readyou.txt b/readyou.txt
index f0cf66f..f70be54 100644
--- a/readyou.txt
+++ b/readyou.txt
@@ -2,3 +2,4 @@ Initail Git repository
Edit 1st time add this line
====> Github Edit (Remote repo) <===
--- Commit before rebase interactive --
+Add line in branch dev
```

จากด้านบนจะเห็นว่าถ้ามองจาก dev ไป master จะเห็นบรรทัดหายไป (สีแดง) แต่ในมองของ master ไป dev จะเห็นบรรทัดเพิ่มขึ้นมาในไฟล์

เมื่อยกระหว่าง Commit \$ [git diff 2e78857 9a570f6](#)

```
AorJoa:scenario_git dekcom$ git lol
* e6d4582 (dev) Add line in branch dev
* 40adb0e (HEAD, master) rename file
* dc19c17 Conflict fixed
* 9a570f6 (origin/master) Update from github (Remote repo)
* 2e78857 Edit#1 add line
* 8ba8bb9 Initial Git repository
```

```
AorJoa:scenario_git dekcom$ git diff 2e78857 9a570f6
diff --git a/readme.txt b/readme.txt
index 3983f72..579e042 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,2 +1,3 @@
Initail Git repository
Edit 1st time add this line
+====> Github Edit (Remote repo) <==
```

ที่เล่นกว่านั้นคือ \$ `git whatchanged --since="2 day ago" --oneline` หรือ \$ `git whatchanged --since="2 weeks ago" --oneline` ตามกันตรงๆแบบนี้เลย

```
AorJoa:scenario_git dekcom$ git whatchanged --since="2 day ago" --oneline
b4af941 test signing commit
:100644 100644 579e042... 90e965c... M readme.txt
AorJoa:scenario_git dekcom$ git whatchanged --since="2 weeks ago" --oneline
b4af941 test signing commit
:100644 100644 579e042... 90e965c... M readme.txt
9a570f6 Update from github (Remote repo)
:100644 100644 3983f72... 579e042... M readme.txt
2e78857 Edit#1 add line
:100644 100644 1c7cc9b... 3983f72... M readme.txt
8ba8bb9 Initial Git repository
:000000 100644 0000000... 1c7cc9b... A readme.txt
```

สถานการณ์ที่ 10 : งานหาย แก้ได้ด้วย Reflog

ขณะที่ผมกำลังทำงานอยู่ สั่ง Git 2-3 คำสั่ง ทันใดข้อมูลทั้งหมดหายไปต่อหน้าต่อตา ไม่ต้องจับมือแม่เวตัวให้หนดม ผมเองเหละที่ทำมันเจ็บ!! แต่ยังชิวๆอยู่ เพราะรู้ว่าสั่ง \$ `git reset` ข้อมูลก็กลับมาได้ แต่มือกีดันเลือก Commit ผิดชะนั้น Holy Sh**t นี่มันวันอะไรของผม นอกจากข้อมูลหายเกลี้ยงแล้ว ประวัติยังหายไปด้วย!

โชคยังดีที่ผมใช้ Git ซึ่งระบบของ Git ยังคงเก็บประวัติทุกอย่างไว้ย้อนหลังไปเป็นเดือนๆ แม้ยังบอกกว่า เป็นการกระทำอะไร Commit, Checkout บลาๆ บอกได้หมด ฟีเจอร์นี้เรียกว่า Reflog เราจะสั่ง \$ `git reflog`

```
AorJoa:scenario_git dekcom$ git reflog
40adb0e HEAD@{0}: checkout: moving from dev to master
e6d4582 HEAD@{1}: commit: Add line in branch dev
40adb0e HEAD@{2}: checkout: moving from master to dev
40adb0e HEAD@{3}: reset: moving to 40adb0e
```

....

```
2e78857 HEAD@{97}: commit: Edit#1 add line
8ba8bb9 HEAD@{98}: reset: moving to 8ba8
34c23df HEAD@{99}: commit: Edit#1 add lin
8ba8bb9 HEAD@{100}: commit (initial): Initial Git repository
```

จากนั้นผมก็จะสั่ง \$ `git reset --hard 2e78857`

```
AorJoa:scenario_git dekcom$ git log --oneline
2e78857 Edit#1 add line
8ba8bb9 Initial Git repository
```

จากนั้นผมจะสั่ง \$ `git cherry-pick af13edd` เพื่อรวม Commit af13edd เข้ามา (ดูเลข Commit จาก Reflog)

```
AorJoa:scenario_git dekcom$ git cherry-pick af13edd
[master b9d8c3b] Update from my computers (Local repo)
 1 file changed, 1 insertion(+)
```

```
AorJoa:scenario_git dekcom$ git log --oneline
b9d8c3b Update from my computers (Local repo)
2e78857 Edit#1 add line
8ba8bb9 Initial Git repository
```

```
readme.txt
1 Initail Git repository
2 Edit 1st time add this line
3 ===> My Computer Edit (Local repo) <===
4
```

สถานการณ์ที่ 11 : Optimize

Git มีความสามารถในการปรับแต่ง และเคลียร์ข้อมูลที่ไม่จำเป็นใน Local repo ทำให้ Repo เร็วขึ้นและใช้เนื้อที่น้อยลง แต่ก็มีข้อแนะนำคือมันอาจจะทำให้ Reflog ที่หมดอายุแล้วถูกเคลียร์หายไปด้วยคำสั่งที่ใช้คือ \$ `git gc`

```
AorJoa:scenario_git dekcom$ git gc
Counting objects: 46, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (32/32), done.
Writing objects: 100% (46/46), done.
Total 46 (delta 20), reused 0 (delta 0)
```

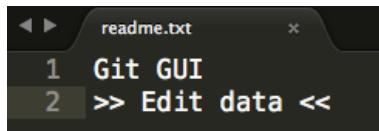
จะเห็นว่ามีการใช้ Delta compression ด้วย

สถานการณ์ที่ 12 : Stash ซ่อนการแก้ไข

เมื่อเราทำงานอยู่ เราอาจจะมีการ Pull, Merge, Rebase หรือมีการเปลี่ยนแปลงของมูลใน Working directory แต่จะทำยังไงถ้าเรายังทำงานค้างอยู่ เพราะยังไม่เสร็จ และยังไม่อยาก Commit เราอาจจะซ่อนการแก้ไขข้อมูลด้วย

\$ `git stash` จากนั้นก็ Pull, Merge, Rebase หรือมีการเปลี่ยนแปลงของมูลใน Working directory ตามต้องการ พอกเสร็จแล้วก็สั่ง \$ `git stash pop` เพื่อดึงข้อมูลที่เก็บไว้ออกมา

ทดลองแก้ข้อมูล เพิ่มบรรทัดที่ 2 เข้าไป



ถ้าสั่ง \$ git status ดูจะเห็นการเปลี่ยนแปลง

```
AorJoa:scenario_git dekcom$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

ใช้ \$ git stash ซ่อนข้อมูลไว้

```
AorJoa:scenario_git dekcom$ git stash
Saved working directory and index state WIP on master: 68a21f7 Test form git gui
HEAD is now at 68a21f7 Test form git gui
```

ถ้าสั่ง Git status ดูจะเห็นว่า Working directory มันว่างแล้ว (เห็นการเปลี่ยนแปลงข้อมูล เพราะซ่อนไว้แล้ว)

```
AorJoa:scenario_git dekcom$ git status
On branch master
nothing to commit, working directory clean
```

ไฟล์จะกลับไปเหมือนเดิม



ตัดจากนั้นจะ Pull , Merge หรือ Rebase ก็ตามสบาย และเมื่อเสร็จแล้วก็ใช้ \$ git stash pop ดึงข้อมูลกลับคืนมา

```
AorJoa:scenario_git dekcom$ git stash pop
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (a65787d803f3f0ea61df2bc913162483a36dc543)
```

สถานการณ์ที่ 13 : ติด Tag ให้ Commit

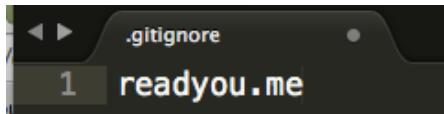
อย่างที่เคยอธิบายเกี่ยวกับ Object ของ Git แล้วจะเห็นว่ามันมี Tag object อญุ่ด้วย ยกตัวอย่าง เราสามารถใช้มันได้จากคำสั่ง `git tag -a v1.0 d256a7f -m 'Tag release V.1.0'` (จะมองเห็น Tag เฉพาะเครื่องเรา ถ้าอยากให้คนอื่นเห็นด้วยต้องใส่พารามิเตอร์ `--tags`) ไปตอนที่ Push ด้วย `$ git push origin --tags`

```
AorJoa:testGit dekcom$ git lol
* d256a7f (HEAD, master) readme edit
| * d490d2e (dev) feature
| * 313dce9 dev#2 conflict
| * 7489af1 dev#1 conflict
|
* af735ab (issue) issue fixed
* fd4ea8e initail commit
```

```
AorJoa:testGit dekcom$ git tag -a v1.0 d256a7f -m 'Tag release V.1.0'
AorJoa:testGit dekcom$ git tag
v1.0
```

สถานการณ์ที่ 14 : ignore ไฟล์

ไฟล์บางไฟล์ หรือไฟล์เดอร์บางไฟล์เดอร์ที่เราไม่ต้องการให้ Git ไปเก็บข้อมูลหรืออยู่ก็เกี่ยวได้ เช่นไฟล์เดอร์ที่ใช้เก็บไฟล์ .war เพื่อเตรียม Deploy เราสามารถสั่งให้ Git ไม่สนใจไฟล์เหล่านั้นได้ ด้วยการสร้างไฟล์ชื่อ .gitignore ขึ้นมาแล้วเอาชื่อไฟล์หรือไฟล์เดอร์ที่ไม่ต้องการไปเก็บไว้ในไฟล์ .gitignore จากนั้นก็สั่ง `$ git add .gitignore` เข้าไปในระบบ



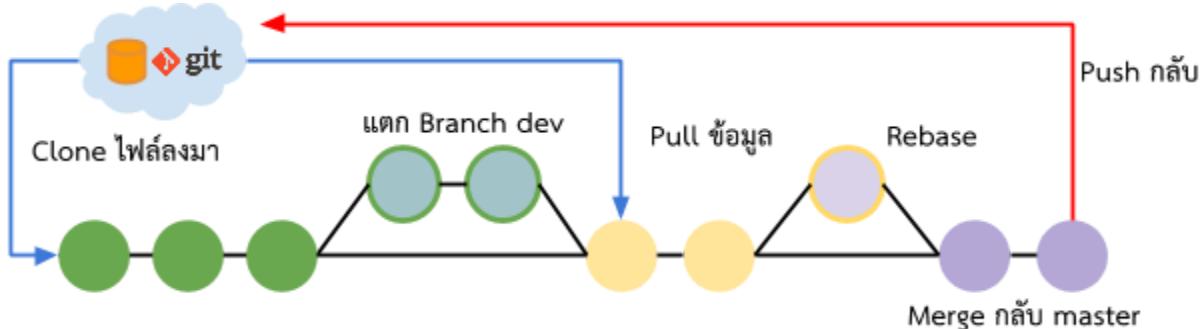
```
drwxr-xr-x  14 dekcom  staff   476 Nov  2 18:58 .git
-rw-rw-r--    1 dekcom  staff    10 Nov  2 18:56 .gitignore
-rw-r--r--    1 dekcom  staff    31 Nov  2 18:18 readme.txt
-rw-r--r--    1 dekcom  staff      5 Nov  2 18:18 readyou.me
```

```
AorJoa:testGit dekcom$ git add .gitignore
```

เพียงนี้ Git ก็มองไม่เห็นไฟล์ readyou.me แล้ว

Workflow

ถึงตอนนี้ยังงงหละจะว่าจะแตก Branch อะไรกันเวลาไหน ตอนเรียนนี่ผมก็เรียกได้ว่าผ่านมาแบบแรกเลือด กันเลยที่เดียว แต่ก็มี Flow ที่เรานับถือเป็นที่พึ่งที่อาศัยอยู่คือ Clone โปรเจกเพื่อน > แตก Branch dev ไฟเจอร์ ของเรา > แก้โค้ดบน Branch dev > เมื่อแก้เสร็จแล้วก่อนจะ Merge กลับก็ Checkout master > Pull โค้ด เวอร์ชั่นล่าสุดลงมาอัปเดทให้ Branch master > Checkout branch dev > Rebase branch dev กับโค้ดใหม่ > Checkout ไปที่ master > Merge branch dev > Push โค้ดกลับไปที่ Remote repo เป็นอันเสร็จพิธี



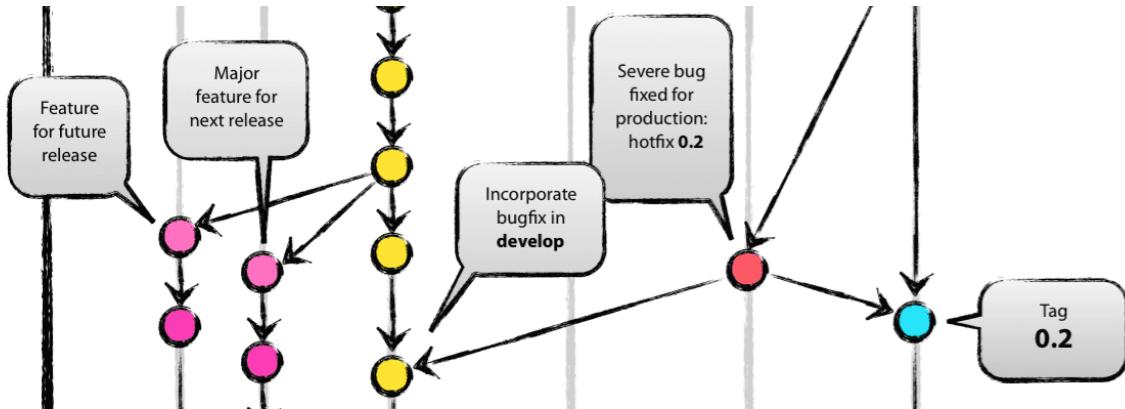
ถึงแต่มาตรฐาน Flow ที่เทพๆ แน่นักกัน (ผมขออธิบายตามที่ผมเข้าใจนะครับ)



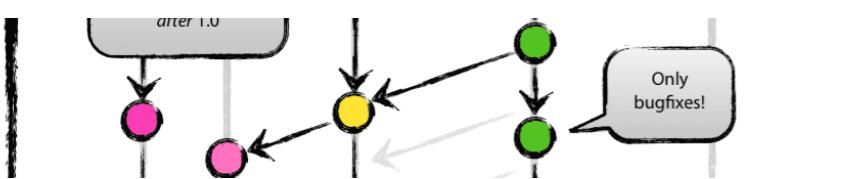
แตก Branch จาก master (master ติด Tag เวอร์ชั่น 0.1 อย่างที่เคยบอกไป Tag ทำให้ Commit ให้เรียกได้ง่าย)



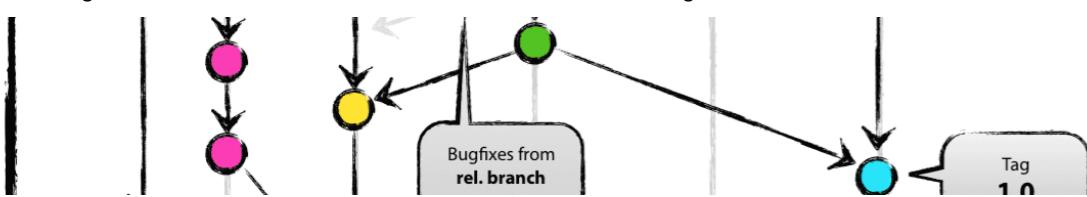
Dev แตก อกมาพัฒนาไฟเจอร์ (**สีชมพู**) ระหว่างนั้นก็มีบี้กเกิดขึ้น เราจึงต้องแก้บี้กก่อนใน hotfix (**สีแดง**) จากนั้นก็ Merge กลับ Master ได้เป็น master Tag 0.2 และไหนๆ ก็แก้บี้กไปแล้วจำให้ Dev เป็นตัวเก่าก็มีความเสี่ยงที่โค้ด เราจะทำงานไม่ถูกต้อง เลย Merge hotfix มาที่ Dev



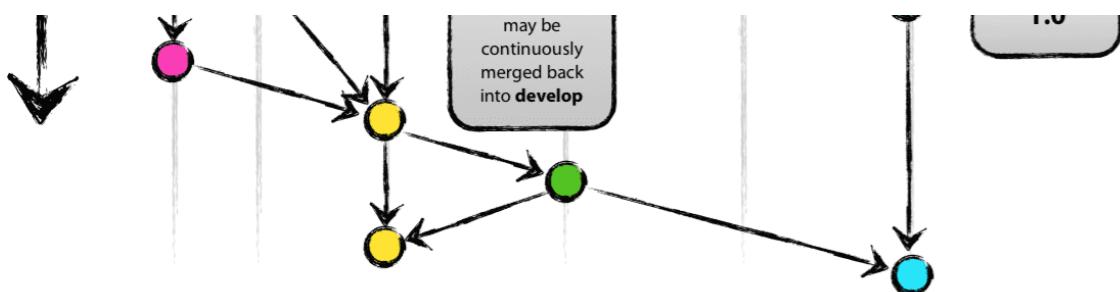
มี Checkout dev และไปพัฒนาฟีเจอร์ต่อ ส่วนใน Release มีการแก้บิ๊ก

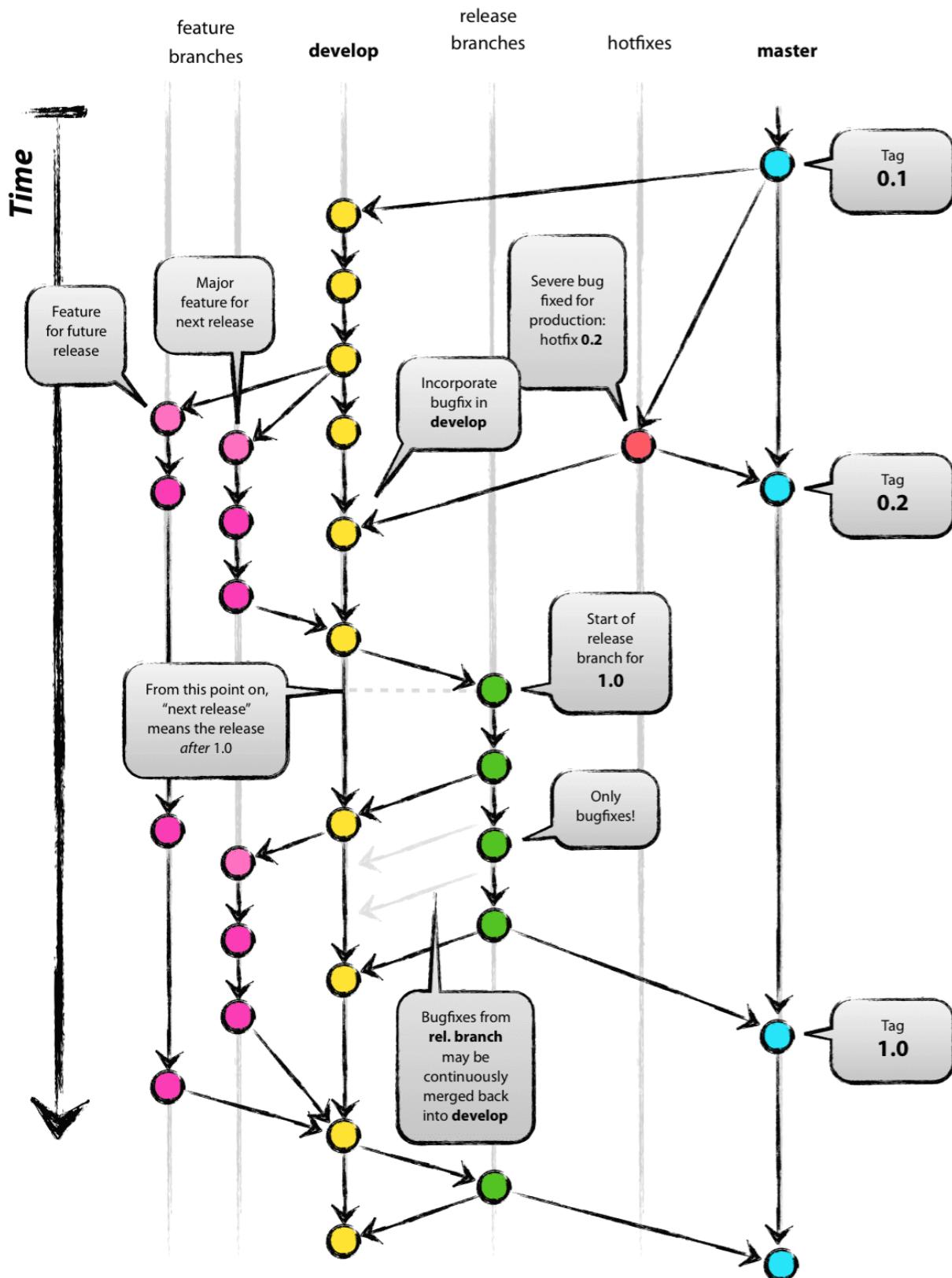


จากนั้นก็ Merge Branch Dev ด้วย Release (สีเขียว) จากนั้นก็ Merge เข้า Master



มีการ Merge ฟีเจอร์กลับ มาที่ Dev และ Dev ก็ Merge กลับมาที่ Release และ Release ก็ Merge เข้า Master





(cc : <http://nvie.com/posts/a-successful-git-branching-model/>)

Fork และ Pull Request

Fork

ถ้าเราอยากรีดโปรเจกของคนอื่นในโดยปกติเราก็จะ Clone จาก Remote repo ของคนอื่นมาแล้วก่อสร้าง Remote repo ของเราไว้จากนั้นก็ Push ข้อมูลขึ้นไปไว้ ก็จะได้ก็อปปี้หนึ่งของ Repo ที่เราเป็นเจ้าของ สามารถ Push โค้ดหรือจัดการ Repo ได้ แต่ Github สร้างฟังก์ชันการใช้งานที่ทำให้เราทำงานแบบที่เหมือนกันนี้ได้ภายใน การคลิกเดียวเพียงครั้งเดียว เรียกว่าการ Fork

(1) Login Github จากนั้นไปที่ Repository ที่ต้องการ Fork



(2) กด Fork



ตัวเลขแสดงจำนวนคนที่ Fork



(3) ลองไปดูที่ Repo ของเราจะมีบวกว่า Fork มาจากไหน

[basic_git](#)

forked from [aorjoabook/basic_git](#)

Basic git tutorial

Updated 22 hours ago

Pull Request

ถ้าเรา Fork repo มาจาก Repo ของคนอื่น Github จะรู้ว่า Repo เรา มีความเชื่อมโยงกัน ทำให้สามารถ Pull Request หรือขอให้ต้นฉบับเอาโค้ดของคุณไปรวมกับของเขายกตัวอย่างถ้าผม Fork ข้อมูลของ Library graph มาแล้วทำการแก้ไข ผมได้พัฒนาไฟล์กราฟแท่งมาตัวหนึ่งเห็นว่าโค้ดสวยได้ที่แล้ว เลยขอให้เจ้าของ Library graph ทำการ Pull Request รวมโค้ดกราฟแท่งที่ผมสร้างไปใน Repo ของเขารับ

ผู้ของคนขอ Pull Request

(1) ไปที่ Repo ที่ Fork มา

[basic_git](#)

forked from [aorjoabook/basic_git](#)

Basic git tutorial

Updated 22 hours ago

Repo ที่ Fork มาแล้ว ผู้แก้โค้ดเพิ่ม

1 lines (1 sloc) 0.007 kb	3 lines (2 sloc) 0.022 kb
1 Git GUI	1 Git GUI
	2 Add this line

(2) เมื่อเข้าไปใน Repo แล้วเลือก Pull Requests



(3) สร้าง New pull request



ลองดูประวัติ

Commits on Oct 30, 2014
Aorjaa Update readme.txt

Showing 1 changed file with 2 additions and 1 deletion.

3 README.md		
...	...	@@ -1 +1,2 @@
1		-Git GUI ↵
	1	+Git GUI
	2	+Add this line

ถ้าโควต้าแล้วก็กด Create pull request



เขียน Comment แล้วก็กด Create pull request อีกรอบ

Update readme.txt	
Write Preview Parsed as Markdown Edit in fullscreen	
Leave a comment	✓ Able to merge. These branches can be automatically merged.
Attach images by dragging & dropping, selecting them, or pasting from the clipboard.	Create pull request

ตอนนี้ Pull Request ก็ถูกเปิดแล้ว

Update readme.txt #1

 Open

Aorjoa wants to merge 1 commit into `aorjoabook:master` from `Aorjoa:master`

ฝั่งของเจ้าของ Repo ต้นฉบับ

(1) เช็ค inbox message



เมื่อคดเข้าไปจะเห็นว่ามีคนขอ Update readme.txt

aorjoabook/basic_git		
	Update readme.txt	a minute ago

(2) กด Merge pull request ข้อมูล

This pull request can be automatically merged.
You can also merge branches on the [command line](#).

(3) Confirm merge

Merge pull request #1 from Aorjoa/master

Update readme.txt

Cancel

(4) สถานะของ Request ก็จะเปลี่ยนไป

Update readme.txt #1

 Merged

aorjoabook merged 1 commit into `aorjoabook:master` from `Aorjoa:master` 24 seconds ago

จะเห็นว่าข้อมูลมีการเปลี่ยนไปแล้ว

 readme.txt

Git GUI
Add this line

Chapter 5 : Git GUI

เห็นใช้ Command line มาตลอดใช่ว่า Git จะไม่มี GUI หรือ Graphical User Interface สวยงามแบบที่ Version control ดังๆ ทั้งนี้มีนั้น แต่ที่อธิบายแบบ Plain Text ก่อน เพราะคิดว่าดูขั้นตอนได้ดีกว่า ชื่องาน Git GUI เราจะมาลองใช้ Software ที่ชื่อว่า SourceTree โดยปุ่มนี้ในแต่ละ OS ก็จะคล้ายๆ กัน

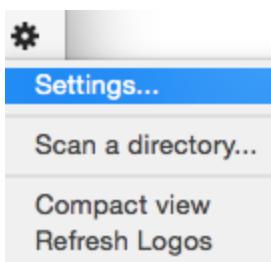
ตัวอย่างการใช้

ตัวอย่างการ Git GUI ผมจะลองใช้ SourceTree ในการ Commit ไฟล์ที่เคยมีอยู่ในระบบของ Git และ ผมแก้ไขไฟล์ไป จากนั้นจะ Commit และ Push ไป

- (1) เข้าไปที่เว็บ <http://www.sourcetreeapp.com> จากนั้นก็โหลด SourceTree



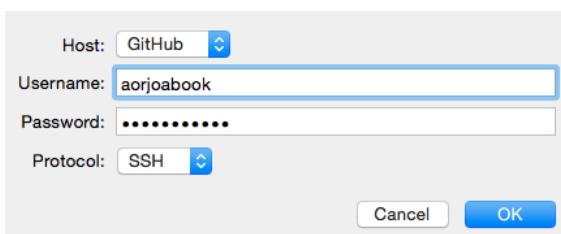
- (2) ติดตั้ง ส่วนนี้ผมไม่ขออธิบายครับ การติดตั้งง่ายมาก ถ้าใครติดตั้งไม่ได้ยกเครื่องมาหาผมเดี๋ยวลงให้ (กล้า ยกมาถูกกล้าลงให้อะ)
- (3) ไปที่ Settings...



- (4) กด Add Account เพื่อเพิ่ม Account Git ใน SourceTree



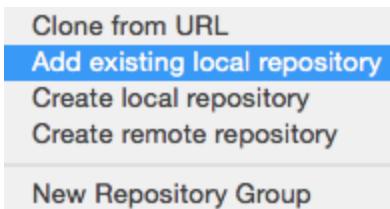
- (5) เพิ่ม Username ของ GitHub เข้าไป



(6) จะเห็นข้อมูลของ Remote repo ที่ Add ไป

Host	Username	Protocol	Default?
GitHub	aorjoabook	ssh	Yes

(7) ไปที่ + New Repository > Add existing local repository



(8) Browse หานานเจอไฟบเดอร์ที่เก็บ Local repo ของเรา



(9) ดับเบิลคลิก เข้า scenario_git



(10) เมื่อเข้ามาจะเห็นหน้าจอและข้อ



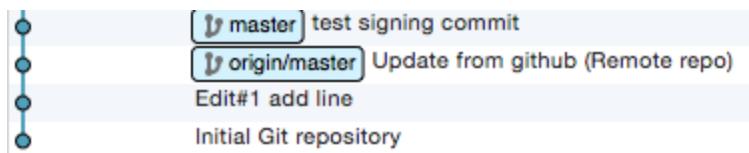
(11) ไปที่ Setting แล้วเลือก Advanced



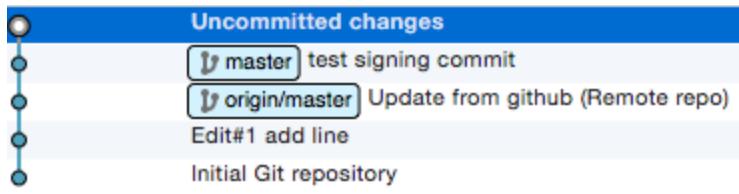
(12) เลือก Use global user settings สำหรับใช้การตั้งค่าจาก global หรือ เอาตัวเลือกออกแล้วตั้งค่าเอง

<input checked="" type="checkbox"/> Use global user settings
Full Name: <input type="text"/>
Email address: <input type="text"/>
<input type="checkbox"/> Use global user settings
Full Name: <input type="text" value="AorJoa"/>
Email address: <input type="text" value="aorjoa.gui@i-aor.com"/>

(13) ประวัติการแก้ไข



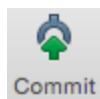
(14) ลองแก้ไขไฟล์ที่อยู่ใน Folder ของ Repo จะเห็น Uncommitted changes



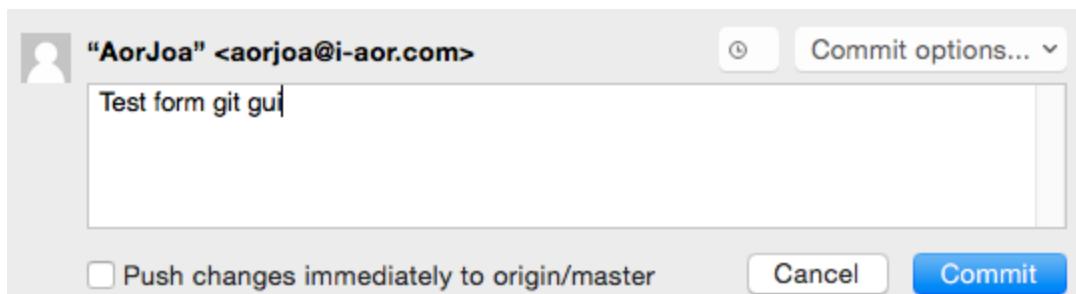
(15) ตรง Unstaged files ติ๊กเลือก แล้วมันจะเด้งไป Staged files ให้



(16) Commit



(17) เพิ่ม Comment จากนั้นกด Commit



(18) หลังจาก Commit ก็จะเห็นการอัพเดท Master ก็จะบัญญัปแน้ว และ Origin ยังอยู่ที่เดิม



(19) Push ไปที่ Origin (Github) เลือก Branch master



(20) สังเกตว่า master และ origin/master อัพเดทแล้ว

Graph	Description	Commit
○	origin/master	68a21f7
○	master	b4af941
○	test signing commit	
○	Update from github (Remote r...)	9a570f6
○	Edit#1 add line	2e78857
○	Initial Git repository	8ba8bb9

(21) ลองเข้าไปดูใน Github

Test form git gui

aorjoabook authored a minute ago

readme.txt Test form git gui

readme.txt

Git GUI

การใช้งาน Git GUI ก็ช่วยอำนวยความสะดวกให้ดี แต่ฟังก์ชันการทำงานก็คล้ายๆกับในแบบ Command line ตัวอย่างด้านบนผมแสดงเครื่องมือการ Commit ไฟล์ที่แก้ไข แต่ส่วนฟังก์ชันการทำงานอย่างอื่นก็ไม่ยากเกินไปที่จะเรียนรู้ด้วยตัวเอง

Chapter 6 : บรรณานุกรม

- (1) Pro Git, [Scott Chacon, Ben Straub], ISBN-13: 978-1484200773
- (2) Pro Git Grean Edition, <https://github.com/opendream/progit>
- (3) อยากรีบ commit รวมกับ commit ก่อนหน้านั้น,
<http://pphetra.blogspot.com/2011/02/git-commit-commit.html>
- (4) git(1) Manual Page, <https://www.kernel.org/pub/software/scm/git/docs/>
- (5) Getting Git Right, <https://www.atlassian.com/git/>
- (6) Software Engineering (บทที่ 5), <https://sites.google.com/site/chanwit/courses/se-56-2>
- (7) มาเริ่มใช้งาน Feature Branch กับ Pull Request กัน,
<http://www.narisa.com/forums/index.php?app=blog&blogid=9&showentry=3070>
- (8) git lol - the other git log, <http://uberblo.gs/2010/12/git-lol-the-other-git-log>
- (9) Git workflow, <http://scottchacon.com>
- (10) Git introduction, <http://en.oreilly.com/rails2008/public/asset/attachment/2816>
- (11) Git Magic by Ben Lynn - Students of Stanford,
<http://www-cs-students.stanford.edu/~blynn/gitmagic/book.pdf>
- (12) สร้าง Git Alias สำหรับคำสั่งที่ใช้งานบ่อย, <http://armno.in.th/2012/10/25/creating-git-alias>
- (13) version control Delta Storage,
<http://version-control.net/version-control-howto/version-control-delta-storage>

Thanks Wikipedia, Google, StackOverflow, Github and another source. :)

คนเขียน



ปกติผมเขียน Blog เพื่อทบทวนและเผยแพร่ข้อมูลพากนี้แต่ผมพบว่าปัญหาของการเขียน Blog คือข้อมูลไม่ค่อยจัดเรียงเป็นระเบียบเวลาจะอ่านหาน่าอ่านยาก(ถึงจะมีติด Tag ก็เถอะ) เลยกะจะเขียนเรื่อง Git ให้เป็นเรื่องเป็นราวเลyen่าจะดีกว่า พอดีเริ่มเขียนไปก็พบว่ามีหลายคำสั่งและข้อมูลหลายส่วนที่ผมก็ไม่เคยรู้ แต่จำเป็นต้องรู้ หรือรู้ไว้ จะมีประโยชน์มากก็ไปหาข้อมูลแล้วก็นำมาเพิ่มเป็นเอกสาร เพิ่มไปเพิ่มมามันก็เยอะไปเรื่อยเลยทำเป็นหนังสือมันจะเลย ดังนั้นหากผิดพลาดส่วนไหนก็ขอให้แจ้งผู้มาด้วยจะดีมากครับ

ขอสารภาพมา ณ ที่นี่ว่าตอนเรียนนี้แอบลักไก่กันเป็นประจำ พอดีใช้ๆไปแล้ว มีปัญหา มีนิติบัตร์ กับ Git ทั้งไฟล์เดอร์ออก และ Clone ลงมาใหม่ประจำ (>.<)” แต่ ก็ยังดีที่มี Community ที่พอจะเป็นที่พึ่งที่อาศัยได้ถ้าใครมีคำถามข้อสงสัยก็ลอง เข้าไปคุยกันได้ที่กลุ่ม “Git อี๊ป” :

<https://www.facebook.com/groups/440497309296387>

สุดท้ายหนังสือเล่มนี้ใช้สัญญาอนุญาต Creative Common 4.0 ซึ่งสามารถทำ ขึ้น ทำสำเนา เผยแพร่ต่อ แก้ไข และใช้ได้สำหรับงานทุกวัตถุประสงค์แม้แต่ทำการค้าก็ ใช้ได้ แต่จำเป็นต้องแสดงแหล่งที่มาของเอกสารนี้

สามารถติดต่อผมได้ทาง

E-mail : bhuridech@gmail.com

Website : www.i-aor.com

WebBlog : aorjua.blogspot.com

GitHub : <https://github.com/Aorjua>



“ขอ Source code จงสถิตอยู่ได้ท่าน”