Feb 16, 2017 · 9 min read

Git คืออะไร ... Git is your friend

0. ยาวไปไม่อ่าน

1. Git คืออะไร ทำไมถึงต้องใช้ Git
2. ติดตั้ง Git
3. Git Command Line
4. Git-GUI Client (SourceTree, Gitk)
5. Git Hosting (Fork, Pull Request)

1. Git คืออะไร

Git คือ Version Control แบบ Distributed ตัวหนึ่ง เป็นระบบที่ใช้จัดเก็บและควบคุมการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์ชนิดใดก็ได้ ไม่ว่าจะเป็น Text File หรือ Binary File (จากนี้จะขอเรียก Text File หรือ Binary File รวมกันว่า Source Code)

ทำไมถึงต้องใช้ Git

Track version ของ Source Code ย้อนกลับได้

- เมื่อจัดเก็บไฟล์เข้าไปในระบบของ Git จะเรียกว่า Git Repository ซึ่งเก็บสำรองข้อมูลและการเปลี่ยนแปลงของ Source Code ทำให้สามารถย้อนกลับไปที่เวอร์ชั่นใดๆ ก่อนหน้า และดูรายละเอียดการเปลี่ยนแปลงของแต่ละเวอร์ชั่นได้ นอกจากนั้นยังสามารถดูได้ว่าใครเป็นคนแก้ไข!!

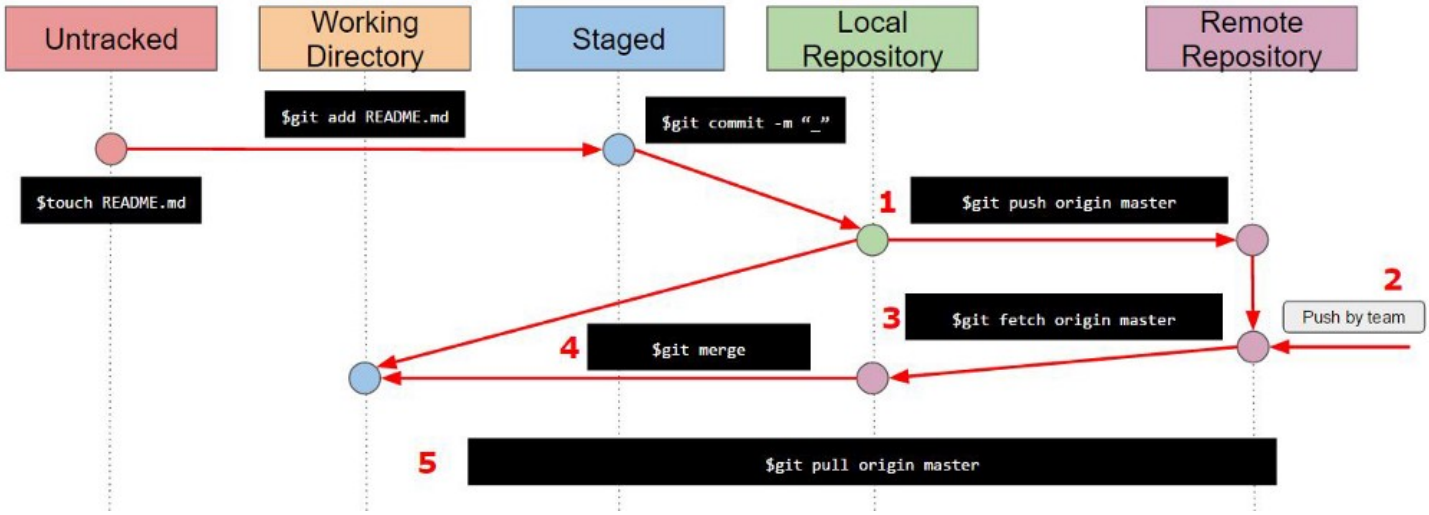
ช่วยในการพัฒนาซอฟต์แวร์เป็นทีม

- Git สามารถเก็บบันทึกการเปลี่ยนแปลงของ Source Code เวอร์ชั่นล่าสุดไว้ที่ Local Repository ซึ่งสามารถทำงานได้โดยที่ไม่ต้องต่อกับอินเตอร์เน็ต และเมื่อต้อง Update การเปลี่ยนแปลงของ Source Code เวอร์ชั่นล่าสุดให้กับเพื่อนร่วมทีมก็สามารถที่จะ Push ขึ้นไปเก็บที่ Remote Repository(Git Hosting) และเพื่อนร่วมทีมก็สามารถ Pull เวอร์ชั้นล่าสุดนั้นมารวม(Auto Merge) ที่เครื่องของเขาเอง ทำให้ Source Code ที่พัฒนาร่วมกันกับคนภายในทีมเป็นเวอร์ชั่นล่าสุดเสมอ

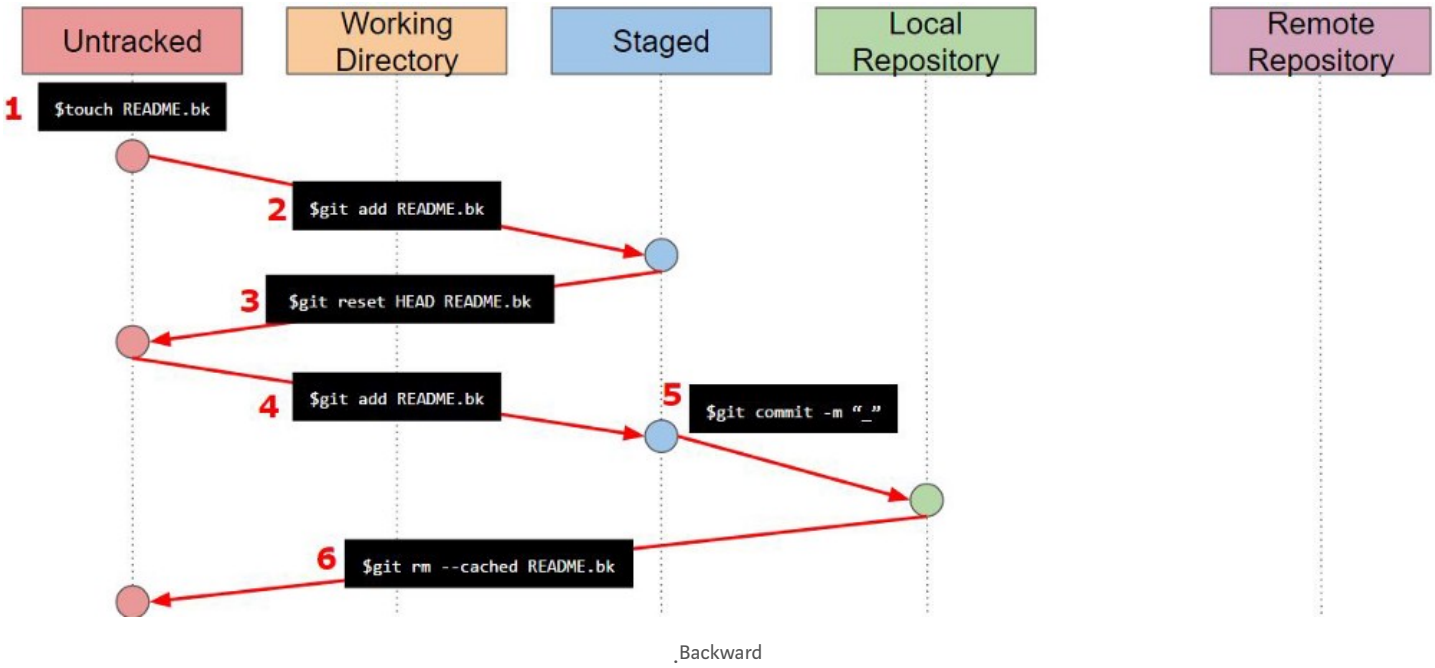
Git Status

สถานะของ Source Code ที่เก็บอยู่ในระบบของ Git นั้นมีดั่งนี้

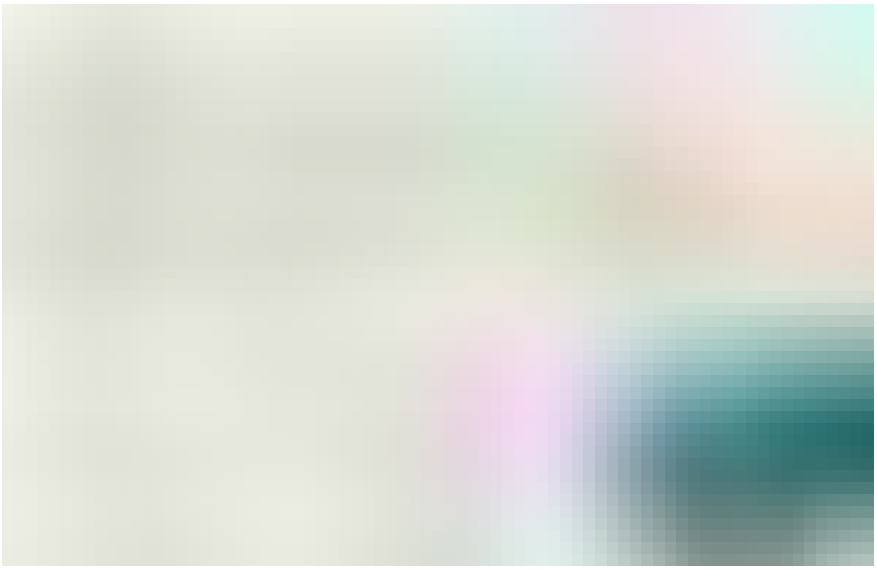
- **Untracked** เป็นสถานะที่ Source Code ถูกเพิ่มเข้ามาใหม่และยังไม่ได้ถูกเก็บไว้ในระบบของ Git
- **Working Directory** เป็นสถานะที่กำลังมีการเปลี่ยนแปลงหรือแก้ไข Source Code หรืออาจจะเรียกสถานะนี้ว่า Modified
- **Staged** เป็นสถานะที่ Source Code กำลังเตรียมที่จะ Commit เพื่อยืนยันการเปลี่ยนแปลงก่อนที่จะเก็บลงในสถานะ Local Repository
- **Local Repository** เป็นสถานะที่มีการเก็บบันทึกข้อมูลการเปลี่ยนแปลงของ Source Code ลงไปที่ Git Repository ที่เป็น Local (ที่เครื่องตัวเอง)
- **Remote Repository** เป็นสถานะที่มีการเก็บบันทึกข้อมูลการเปลี่ยนแปลงของ Source Code ลงไปที่ Git Repository ที่เป็น Hosting (ที่เครื่องเซิร์ฟเวอร์)



Forward



2. ติดตั้ง Git

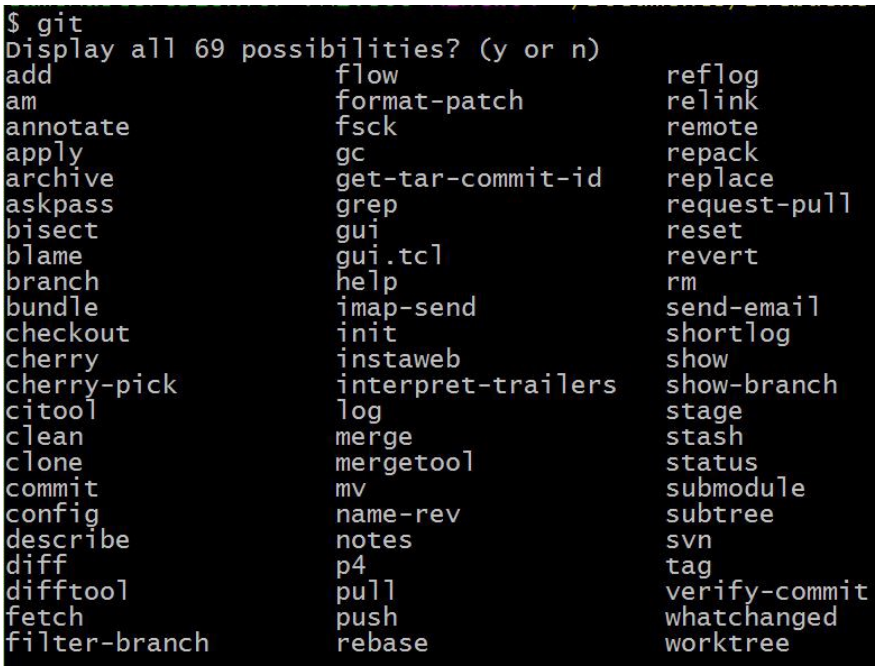


Git สามารถที่จะติดตั้งได้ทั้ง Windows, Mac OS X, Linux ซึ่ง Download ได้จาก https://git-scm.com สำหรับการติดตั้งก็คงไม่ต้องอธิบายแล้วนะครับ แต่มีคำแนะนำว่าควรใช้เวอร์ชั่นล่าสุด(2017/01/12 v2.11.1) เหตุผลเพราะมีการแก้บั๊กและเพิ่มคำอธิบายให้เข้าใจยิ่งขึ้นในเวอร์ชั่นใหม่ครับ [2], [3]

เมื่อติดตั้งเสร็จแล้วสามารถตรวจสอบ Git Version ได้จากคำสั่งนี้ (Max OS X, Linux ใช้ Terminal ส่วนใน Windows ใช้ Git Bash ติดตั้งมาพร้อมกับ Git)

```
$git --version
```

3. Git Command Line



ถามว่าทำไมถึงใช้ Git แบบ Command Line ส่วนตัวคิดว่ามันจะทำให้เข้าใจพื้นฐานของ Git ได้ดีกว่าการใช้ Git-GUI Client และมีส่วนที่มีแค่ Command Line เท่านั้นที่ทำได้ \(-__-)/ เมื่อเข้าใจพื้นฐานแล้วสามารถที่จะใช้ Git-GUI Client ตัวไหนก็ได้ หรือใช้แค่ Command Line อย่างเดียวก็พอครับ ;P

แนะนำให้ลองเล่นตามทีละคำสั่งเลยนะครับ จะได้เข้าใจมากขึ้น Go Go Go..

เริ่มที่คำสั่งที่ใช้งานบ่อยๆ

Git Config

เป็นคำสั่งที่ใช้แสดงและกำหนดข้อมูลของผู้ใช้เพื่อระบุตัวตน และคุณสมบัติอื่นๆ ของ Git

```
$git config --global --list #แสดงคุณสมบัติของ Git ทั้งหมด
$git config --list          #แสดงคุณสมบัติของ Git เฉพาะ
Repository นั้น
```

การกำหนดชื่อและอีเมล์ของผู้ใช้

```
$git config --global user.name "Your Name"           #กำหนดชื่อผู้ใช้
$git config --global user.email "example@email.com"  #กำหนดอีเมล์ของผู้ใช้
$git config --global --list #ตรวจสองอีกครั้งหลังจากกำหนดค่าเสร็จแล้ว
```

Git Init

เป็นคำสั่งที่ใช้สร้างระบบของ Git ขึ้นมาภายใต้โฟลเดอร์หรือ Path นั้น โดยจะสร้างโฟลเดอร์ .git ขึ้นมาเพื่อใช้เก็บ สำรองข้อมูล การเปลี่ยนแปลงและคุณสมบัติอื่นๆ ของ Git

```
$git init
```

Git Status

เป็นคำสั่งที่ใช้ตรวจสองสถานะของ Source Code ในระบบของ Git ซึ่งจะแสดงสถานะดั่งที่ได้อธิบายข้างต้นไปแล้ว

```
$git status
```

Git Add

เป็นคำสั่งที่ใช้เพิ่มการเปลี่ยนแปลงของ Source Code เข้าไปที่สถานะ Staged

```
$git add <file_name>
$git add README.md    #เพิ่มไฟล์ชื่อ README.md เข้าไปที่สถานะ Staged
$git add .            #ใช้ในกรณีที่มีหลายๆ ไฟล์และต้องการเพิ่มเข้าไปทั้งหมด
```

Git Commit

เป็นคำสั่งที่ใช้ยืนยัน Source Code ที่อยู่ในสถานะ Staged เข้าไปเก็บไว้ที่ Local Repository

```
$git commit -m "message"          #ยืนยันการเปลี่ยนแปลงพร้อม
ข้อความ
$git commit -am "message"         #เพิ่มการเปลี่ยนแปลงและยืนยัน
พร้อมข้อความ
$git commit                       #เพิ่มข้อความในโปรแกรม vi
```

```
#ยืนยันการเปลี่ยนแปลงพร้อมข้อความและ merge ลงใน commit ล่าสุด
$git commit --amend -m "message"
```

ถ้าต้องการเขียน Commit Message ยาวๆ สามารถใช้คำสั่ง Git commit ระบบจะเปิดโปรแกรม vi ให้เขียน Message [วิธีใช้ vi, vim]

Git Log

เป็นคำสั่งที่ใช้แสดงประวัติของ Commit ที่เก็บไว้ใน Repository

```
$git log
$git log --oneline
$git log --oneline --decorate
$git log --oneline --decorate --graph
```

```
$git log --stat                #Diff from log
$git log --grep="Message"      #Log by message
$git log --after="2017-2-14"   #Log in date
$git log --before="2017-2-14"  #Log in date
$git log --author=pakin        #Log by user
```

Git Branch

เป็นคำสั่งที่ใช้ในแสดงและแตงกิ่งสาขาในการพัฒนา ซึ่งทำให้การพัฒนาซอฟต์แวร์มีความยืดยุ่นมากขึ้น

```
$git branch
$git branch --all
$git branch develop  #สร้าง branch ชื่อ develop
```

```
$git branch --delete develop #ลบ branch ชื่อ develop
```

```
#ส่งการเปลี่ยนแปลง branch develop ไปยัง Remote ที่ชื่อ origin
$git push origin develop
```

```
#ส่งการเปลี่ยนแปลงลบ branch develop ไปยัง Remote ที่ชื่อ origin
$git push --delete origin develop
```

เรื่องของ Branch และ Tag นั้นมีความเกี่ยวข้องกับเรื่องของ Release Process ของการพัฒนาซอฟต์แวร์ ขึ้นอยู่กับการตกลงกันภายในทีมและรูปแบบที่เหมาะสมกับซอฟต์แวร์ที่กำลังพัฒนา ซึ่งเรียกเทคนิคนี้ว่า Branch Strategy(Git Workflow, Branching Models, Branching Workflow, Git Flow)

Git Checkout

เป็นคำสั่งที่ใช้ในการสลับ Working Directory ไปยัง Branch หรือ Commit ที่เราระบุ คำสั่งนี้ยังสามารถให้งานได้ในอีกหลายๆ แบบ

```
#ย้ายการทำงานไปที่ Branch หรือ commit_id ที่ระบุ
$git checkout <branch name, commit id>

#สร้าง branch ชื่อ test และทำการสลับการทำงานมาที่ Branch นี้
$git checkout -b test

#ยกเลิกการเปลี่ยนแปลงของไฟล์ใน Working Directory
$git checkout -- <file name>

#เลือกแค่บางไฟล์จาก Branch อื่น เข้ามา Merge กับ Working Directory ที่กำลังทำงาน
$git checkout <branch name> <file name>

#คำสั่งนี้จะเหมือนคำสั่งด้านบนแต่จะมีโหมดตอบโต้กับผู้ใช้ในการเลือกสถานะของไฟล์ที่ระบุ
$git checkout --patch <branch name> <file name>
```

Git Reset

เป็นคำสั่งที่ใช้ย้อนกลับไปที่เวอร์ชั่นใดๆ ก่อนหน้า โดยระบุ Branch หรือ Commit Id (SH-1 แบบย่อของ Commit 7 ตัว เช่น 4bcb295) ซึ่งมี Option ที่สำคัญ 3 ตัวดังนี้

- **soft** ย้อนการเปลี่ยนแปลง และคงสถานะการเปลี่ยนแปลงของ Source Code ไว้ที่สถานะ Staged
- **mixed** ย้อนการเปลี่ยนแปลง และคงสถานะการเปลี่ยนแปลงของ Source Code ไว้ที่สถานะ Working Directory หรือ Modified
- **hard** ย้อนการเปลี่ยนแปลงแบบลบทับการเปลี่ยนแปลงก่อนหน้าทั้งหมด คำสั่งนี้อันตรายเพราะมันจะทำให้ประวัติของ Commit ที่เก็บไว้ใน Repository หายไป จึงยังไม่เหมาะกับมือใหม่

```
$git reset --soft 4bcb295   #ย้อนกลับไปที่ Commit id 4bcb295
$git reset --mixed develop  #ย้อนกลับไปที่ Branch develop
```

Git Merge

เป็นคำสั่งที่ใช้ในการรวม Branch หรือ Commit ทั้งสองเข้าด้วยกัน

ตัวอย่างเราจะอยู่ที่ Branch Master และต้องการ Merge Branch Feature เข้ามาทำงานร่วมด้วย การ Merge แบบ No Fast Forward จะเรียกอีกอย่างหนึ่งว่า 3-Way Merge

```
#รวม branch master กับ branch feature แบบ no fast forward
$git merge --no-ff feature
```

```
#รวม branch master กับ branch feature แบบ fast forward
$git merge feature
```

Git Remote [เริ่มต้นทำงานกับ Git Hosting (Remote Repository)]

เพิ่ม URL ของ Remote Repository เข้าไปยังคุณสมบัติของ Git โดยชื่อว่า origin ส่วนใหญ่จะเป็นชื่อ Default ที่หลายๆ คนเข้าใจตรงกัน แต่เราก็สามารถตั้งชื่ออื่นๆ ได้

```
$git remote add origin <URL> #เพิ่ม Remote Repository ชื่อ
origin


$git remote add origin
https://github.com/NewGame0/Android_HelloWorld.git
```

```
#เพิ่ม Remote Repository ใหม่ชื่อ origin
$git remote set-url origin <New URL>
```

```
$git remote -v      #แสดง Remote Repository
$git config --list  #แสดงคุณสมบัติต่างๆของ Git ซึ่งจะมี Remote
Repository แสดงออกมาด้วย
```

Git Push

เป็นคำสั่งที่ใช้ส่งการเปลี่ยนแปลงของ Source Code ที่เก็บอยู่บน Local Repository ขึ้นไปยัง Remote Repository

```
#ส่งการเปลี่ยนแปลง Branch master ไปยัง Remote ที่ชื่อ origin
$git push origin master
```

Git Fetch

เป็นคำสั่งที่ใช้รับการเปลี่ยนแปลงของ Source Code ล่าสุดที่อยู่บน Remote Repository ลงมายัง Local Repository แต่ยังไม่ได้ทำการรวม Source Code (Merge)

```
#รับการเปลี่ยนแปลงทุก Branch จาก Remote Repository
$git fetch --all

#รับการเปลี่ยนแปลง Branch master จาก Remote Repository ที่ชื่อ
origin
$git fetch origin master
```

Git Pull [fetch + merge]

เป็นคำสั่งที่ใช้รับการเปลี่ยนแปลงของ Source Code ล่าสุดที่อยู่บน Remote Repository ลงมายัง Local Repository และทำการ Auto Merge

```
$git pull <remote> <branch>
$git pull origin master
```

Git Clone

เป็นคำสั่งที่ใช้ดึงประวัติทั้งหมดบน Remote Repository ของเพื่อนร่วมทีมของคนอื่นหรือของเราเองที่มีอยู่แล้วบน Git Hosting มาที่เครื่องของเรา คำสั่งนี้จะคล้ายๆ Git Init ที่ใช้สร้างระบบ Git ขึ้นมาตอนเริ่มต้น แต่เราจะได้ประวัติเดิมของ Repository มาด้วย ทำให้เราเริ่มพัฒนาต่อจากตรงจุดนี้ได้เลย

```
$git clone
https://github.com/NewGame0/Android_HelloWorld.git
```

คำสั่ง Git Clone นั้นจะ Checkout Branch หลักมาเป็น Master และดึง Tag ลงมาทั้งหมด

คำสั่งอื่นๆ

Git Ignore [.gitignore]

Git Ignore ไม่ได้เป็นคำสั่งแต่เป็นคุณสมบัติของ Git โดยการเพิ่มไฟล์ที่ชื่อ .gitignore เข้าไปในระบบของ Git เพื่อทำการบอกให้ Git ไม่ต้องสนใจไฟล์หรือโฟลเดอร์นั้นๆ เช่นไฟล์หรือโฟลเดอร์ที่เป็น Output ของการ Build ใน Java (.class) ไฟล์ที่เป็นคุณสมบัติเฉพาะของ IDE หรือ Working Space ก็ไม่ควรแชร์ไปให้คนอื่นๆ ในทีม

```
$touch .gitignore #สร้างไฟล์ .gitignore
```

```
#เพิ่ม String เข้าไปในไฟล์ .gitignore เพื่อ ignore ไฟล์ .class ทั้งหมด และโฟลเดอร์ Debug, Build
$echo >> .gitignore "*.class"
$echo >> .gitignore "/Debug"
$echo >> .gitignore "/Build"
```

```
$git add .gitignore #เพิ่มไฟล์ชื่อ .gitignore เข้าไปที่สถานะ Staged
$git commit -m "Add .gitignore file"
```

ในกรณีที่มีการเพิ่มไฟล์ที่ไม่ต้องการเข้าไปยังสถานะ Staged แล้ว และเพิ่มไฟล์ .gitignore เข้าไปทีหลัง สามารถใช้คำสั่งนี้เพื่อลบไฟล์หรือโฟล์เดอร์ที่ไม่ต้องการออกจากสถานะ Staged และ Commit Apply .gitignore เข้าไปอีกครั้ง

```
$git rm --cached <file name>
$git rm --cached <path to file>
$git rm --cached .class
$git rm --cached Debug/*
$git rm --cached Build/*
$git rm -r --cashed *
```

```
$git commit -am "apply .gitignore file"
```

```
#แสดงไฟล์ที่ Track ไว้ในระบบของ Git ไฟล์ที่ ignore จะหายไปแม้จะยังอยู่ในโฟล์เดอร์
$git ls-files
```

Git Ignore ที่มีการรวบรวมไว้บน GitHub
https://github.com/github/gitignore

Git Tag

เป็นคำสั่งที่ใช้แสดงและสร้าง Tag ขึ้นที่จุด commit นั้น

```
$git tag         #แสดงแท็กทั้งหมด
$git tag -n99    #แสดงแท็กทั้งหมดพร้อมข้อความ

$git tag v1.0.0                    #สร้างแท็กชื่อ v1.0.0
$git tag v1.0.1 -m "Tag Message"  #สร้างแท็กชื่อ v1.0.0 พร้อมระบุ
ข้อความ

$git tag --delete v1.0.0          #ลบแท็กชื่อ v1.0.0

$git push origin <tag name> #ส่งแท็กขึ้นไปที่ Remote Repository
$git push origin --tags     #ส่งแท็กทั้งหมดขึ้นไปที่ Remote
Repository

$git push --delete origin <tag name> #ลบแท็กที่ Remote
Repository
```

เรื่องของ Branch และ Tag นั้นมีความเกี่ยวข้องกับเรื่องของ Release Process ของการพัฒนาซอฟต์แวร์ ดั่งที่ได้กล่าวไปแล้ว ส่วนใหญ่แล้วการตั้งชื่อ Tag จะตรงกับเลขเวอร์ชั้นของซอฟต์แวร์ที่ Release, Deploy, หรือที่ส่งให้กับลูกค้า เช่น v1.12.4 (v.x.y.z) [4]

- v คือบอกว่าเป็นเวอร์ชั่นอะไร
- x คือ Major เวอร์ชั่น
- y คือ Minor เวอร์ชั่น
- z คือ Patch เวอร์ชั้น

ส่วนข้อความภายใน Tag จะนิยมระบุ Date, Release to, New Feature?, Fix Bug? ...

Git Clean

เป็นคำสั่งที่ใช้แสดงและลบ Source Code ที่อยู่ในสถานะ Untracked ออกจาก Working Directory

```
$git clean -n   #แสดง Source Code ที่อยู่ในสถานะ Untracked
$git clean -df  #ลบ Source Code ที่อยู่ในสถานะ Untracked
```

Git Diff

เป็นคำสั่งที่ใช้แสดงความเปลี่ยนแปลงระหว่าง Working Directory ที่กำลังทำงานอยู่กับ Branch หรือ Commit Id ที่ระบุ

```
$git diff 82de188
$git diff develop
```

Git Stash

เป็นคำสั่งที่ใช้ซ่อนการเปลี่ยนแปลงใน Working Directory ทำให้ Working Directory Clean นิยมใช้ก่อนคำสั่ง Git Pull

```
$git stash       #ซ่อนการเปลี่ยนแปลงลงใน stash
$git stash list  #แสดงรายการการเปลี่ยนแปลงที่ซ่อนไว้
$git stash show  #แสดงการเปลี่ยนแปลงล่าสุดที่ซ่อนไว้

$git stash pop   #ดึงการเปลี่ยนแปลงล่าสุดมาออกมา merge กับ working
directory
```

Git Reflog

เป็นคำสั่งที่ใช้แสดงและจัดการกับ Reference Log ของ Git Repository

ขอยกตัวอย่างการใช้งาน ผม(พลาด)ใช้คำสั่ง Git Reset -hard ย้อนกลับไป 3 commit ก่อนหน้า ทำให้ประวัติของ Commit ทั้ง 3 ก่อนหน้าที่จะย้อนมาหายไป ใช้คำสั่งตามด้านล่านนี้

```
#แสดง Reference Log ของ Git Repository
$git reflog show

#ย้อนกลับไปยัง head log ก่อนหน้า เท่านี้ก็จะได้ 3 commit กลับมาแล้ว
$git reset HEAD@{1}
```

Git Help

เป็นคำสั่งที่ใช้ในการแสดงและอธิบายคำสั่งของ Git

```
$git help <command name>
$git help merge
```

คำสั่งอื่นๆ ที่เคยได้ยิน (ยังไม่เคยใช้จริง แค่ลองเล่น -__-)

- git revert
- git rebase
- git cherry-pick
- git hooks

Git Command + Option + Parameter เยอะแบบนี้จะจำกันได้ยังไง ?

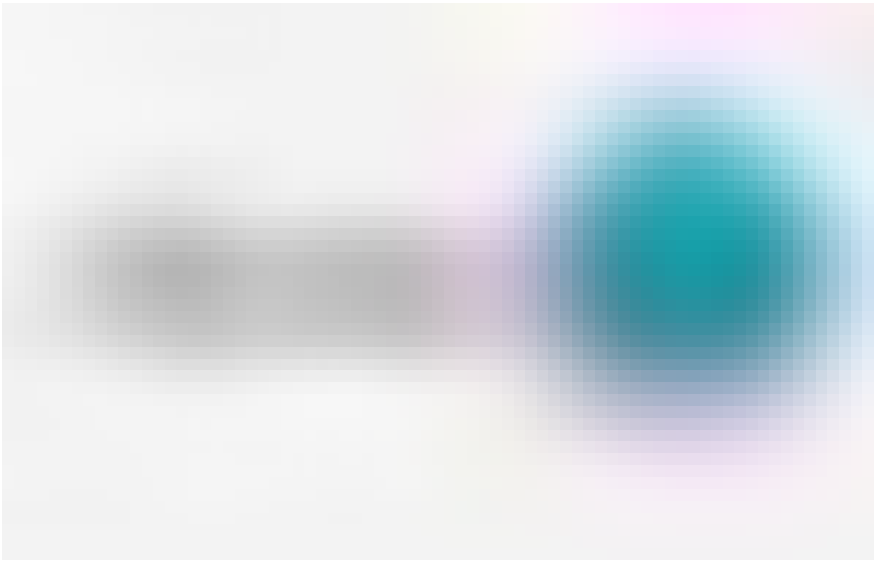
จำไม่ได้ครับ… ถ้าแค่อ่าน Command มาจากด้านบนจนถึงตรงนี้แล้ว งง ก็คงไม่แปลก คำแนะนำคือต้องฝึกใช้แต่ละ Command และใช้งานบ่อยๆ บางคำสั่งมี Option ช่วยทำให้ใช้งานง่ายขึ้น และ Command ส่วนใหญ่จะใช้ต่อกันเป็นชุดๆ ครับ เช่น Git Status, Git Add , Git Commit, Git Push ถ้าใช้บ่อยๆ แล้วจะจำกันได้เองครับ :P

ทดลองเล่นคำสั่ง Git บนเว็บไซต์ Try Git: Git Tutorial

4. Git-GUI Client

สำหรับคนที่ยัง งง และไม่ชอบใช้ Command Line ลองเปลี่ยนมาใช้งานแบบ GUI อาจจะช่วยให้ใช้งานง่ายขึ้น :)

Source Tree



เป็นซอฟต์แวร์ของบริษัท Atlassian สามารถติดตั้งได้ทั้งบน Windoew และ Mac
OS X ซึ่งใช้งานได้แบบฟรีเพียงแค่สมัคร Account ของ Atlassian

แนะนำสำหรับมือใหม่หน้าตาใช้งานง่าย Graphic สวยเข้าใจได้ง่าย ส่วนวิธีใช้ก็คงไม่ยากเกินไป สามารถ Download ได้จาก
https://www.sourcetreeapp.com/

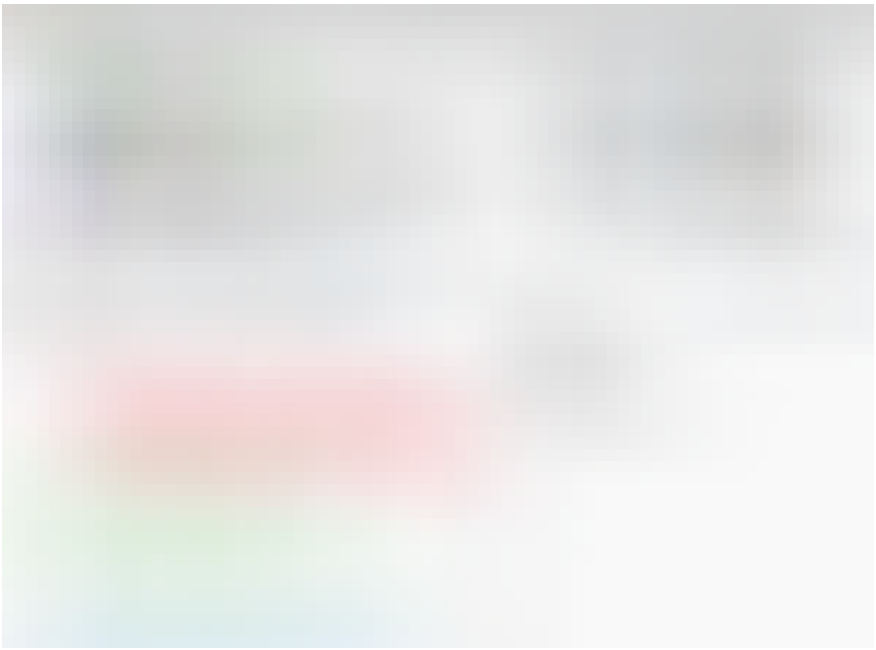
Gitk

แนะนำเป็นการส่วนตัว เพราะเมื่อใช้ Command Line บ่อยๆ มีปัญหากับคำสั่ง Git Diff, Git, Log ที่แสดงผลแล้วดูยากไปหน่อยใน Command Line

เลยต้องใช้ GUI ช่วย ถ้าต้องไปเปิด SourceTree ก็คงดูยุ่งยากไปหน่อย(รู้สึกหนวงๆ ช้าๆ ยังไงไม่รู้)
ก็มาเจอกับ Gitk ที่เบาและสามารถตอบโจทย์ได้ ติดตั้งมาพร้อมกับ Git บน Windows ส่วนใน Linux ต้องติดตั้งเพิ่ม สามารถเรียกใช้งานได้ที่ Terminal เลย

```
$gitk       #แสดง Working Directory ที่กำลังทำงานอยู่
$gitk --all #แสดง Working Directory ที่กำลังทำงานอยู่และ Branch ทั้งหมด
```

หน้าตาอาจจะดูใช้งานยากไปหน่อยแต่แค่ดู Diff, Log แล้วถือว่าเพียงพอ



gitk

Git-GUI Client ตัวอื่นๆ

https://git-scm.com/downloads/guis

5. Git Hosting (Remote Repository)

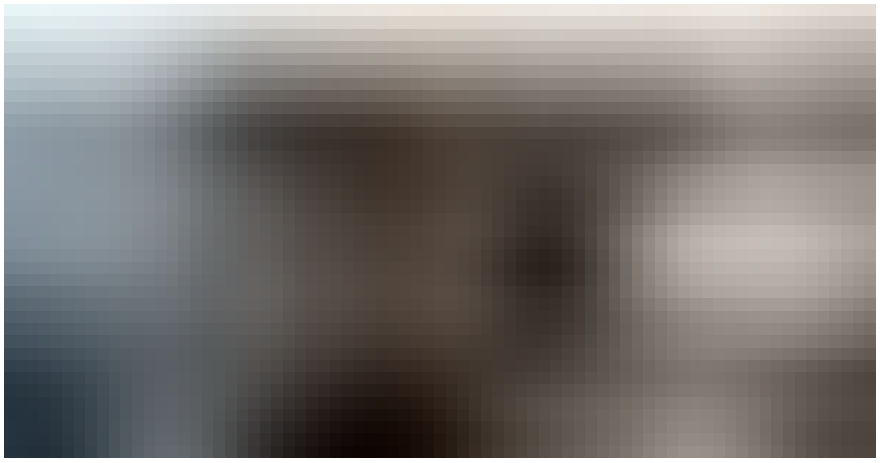
บริการฝาก Repository ที่ไว้ Git Hosting ซึ่งมีเจ้าหลักๆ ดั่งนี้

Bitbucket

https://bitbucket.org

ส่วนตัวแล้วใช้ของ Bitbucket เป็นหลัก สามารถสร้าง Repository ทั้งแบบ Private และ Public ได้ฟรีไม่จำกัด Repository (แต่จำกัดขนาดแต่ละ Repo ไม่เกิน 1 GB) แบบ Private จำกัดจำนวนผู้ใช้ที่เข้าถึงได้แค่ 5 คน(แบบฟรี) ถ้าเยอะกว่านี้ต้องจ่ายตังเพิ่ม ทดลองใช้เลยที่ https://bitbucket.org

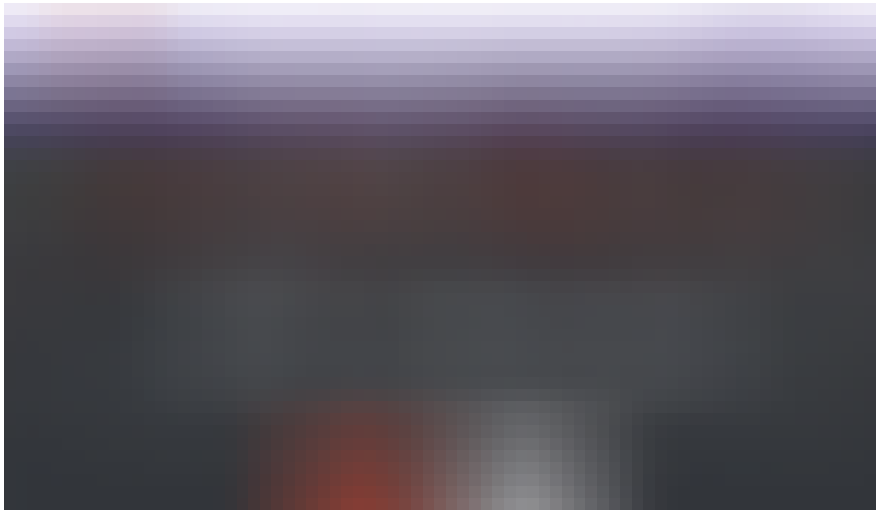
GitHub



https://github.com

เชื่อว่าหลายคนคงรู้ต้องรู้จักเพราะเป็นบริการฝาก Repository ที่มีคนใช้งานเยอะมาก สามารถสร้าง Repository แบบ Public ได้ฟรีไม่จำกัด Repository แต่ถ้าต้องการให้เป็นแบบ Private ต้องจ่ายตังเพิ่ม ทดลองใช้เลยที่ https://github.com

GitLab



https://gitlab.com

GitLab สามารถสร้าง Repository ฝากไว้ที่ Hosting แบบ Private ได้ฟรีไม่จำกัด Repo สำหรับคนที่กลัวและไม่อยากที่จะนำ Source Code ไปฝากไว้ที่ Git Hosting สามารถที่จะตั้ง Server Git เองได้ที่บริษัท โดยติดตั้ง GitLab ได้ฟรี (สามารถติดตั้งบน Linux Server ได้โดยง่าย รายละเอียดเพิ่มเติมที่ https://gitlab.com และควรมีคนที่คอย Management Server Git)

https://about.gitlab.com/downloads

GitLab เพิ่งมีข่าวว่าเผลอลบข้อมูลผู้ใช้บน Server ของตัวเองและมีปัญการในการกู้ข้อมูลกลับคืนก ลองพิจารณาดูครับก่อนตัดสินใจใช้งาน รายละเอียดตามนี้ครับ https://www.blognone.com/node/89724

Fork & Pull Request

Fork และ Pull Request ไม่ใช่เรื่องของ Git โดยตรงแต่เป็นคุณสมบัติที่เพิ่มมากับ Git Hosting เพื่อให้สามารถใช้งานได้สะดวกขึ้น

โดยปกติแล้วเมื่อเราสนใจ Project บน Remote Repository ของคนอื่นและต้องการนำมาพัฒนาต่อยอด เราก็เริ่มด้วยคำสั่ง Git Clone หลังจากนั้นก็สร้าง Remote Repository บน Git Hosting ของเราเองและทำการ Push ขึ้นไปเก็บไว้ การทำงานแบบนี้เกิดขึ้นได้บ่อยในการพัฒนาซอฟต์แวร์ร่วมกัน ซึ่ง Git Hosting(Bitbucket, GitHub, GitLab) ก็ช่วยอำนวยความสะดวกด้วยคำสั่งเดียวคือ Fork

การ Fork Project ของคนอื่นๆ มาพัฒนายังมีความเกี่ยวข้องกับ Repository ของเจ้าของเดิม เมื่อเรามีการเพิ่มการเปลี่ยนแปลงของ Source Code ใน Repository ที่ Fork มา และเราเห็นว่ามันมีประโยชน์กับ Project หลักของคนที่เรา Fork Project มา เราก็สามารถส่งการเปลี่ยนแปลงนี้เข้าไปที่ Repository หลักของผู้ที่เป็นเจ้าของ Project ได้ ซึ่งเรียกว่า Pull Request เจ้าของ Project จะเป็นคนตัดสินใจเองว่าจะรวมการเปลี่ยนแปลงของเราเข้าไปยัง Repository หลักของเขาหรือเปล่า

Working with Git Hosting

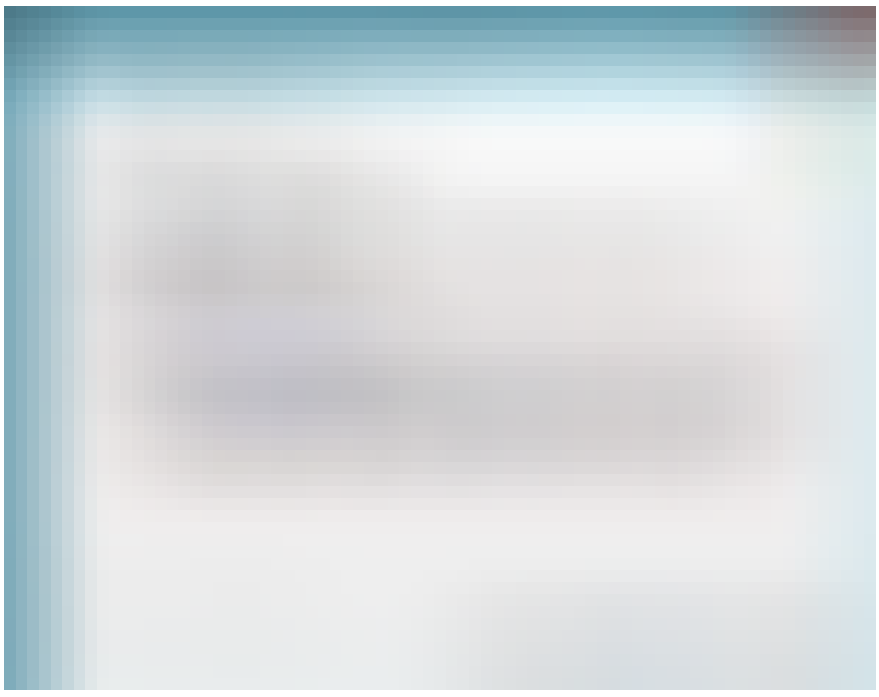
การติดต่อกับ Git Hosting จะมี Protocol ที่ใช้ติดต่อ 2 แบบคือ HTTPs และ SSH Key (Secure Shell key) เพื่อยืนยันตัวตนของผู้ใช้

HTTPs จะเหมือนกับการ Log in เข้าใช้งานที่ Git Hosting โดยเมื่อเราใช้คำสั่ง Git Clone, Git Push, Git Pull ในครั้งแรกจะมีหน้าต่างขึ้นมาหรือ Command Line ให้เราใส่ Account ของ Git Hosting ที่เราใช้งาน

Git Credential Manager for Windows

ในเวอร์ชั่นของ Git (< 1.7.10) มีการเก็บ Account(User, Password) ไว้ในไฟล์ โดยใครๆ ก็สามารถเข้าไปดูได้ ซึ่งไม่ปลอดภัย หลังจาก Git เวอร์ชั่นที่ 1.7.10 เป็นต้นมา จะมีคุณสมบัตินี้เรียกว่า Credential Helpers ซึ่งเข้ามาช่วยในการจดจำ User และ Password ทุกครั้งที่มีการติดต่อกับ Git Hosting โดยจะนำไปเก็บไว้ที่ Secure Disk บนแต่ละระบบปฏิบัติการ ทำให้เราไม่ต้องใส่ User และ Password ทุกๆ ครั้ง และ User และ Password ของเราจะถูกเก็บไว้ในที่พื้นที่ปลอดภัย (คนทั่วไปไม่สามรถอ่านไม่ได้)

Credential Helper ที่ใช้งานในแต่ละระบบปฏิบัติการไม่เหมือนกัน ใน Windows นั้น ตั้งแต่ Git เวอร์ชั่นที่ 2.7.3 จะมีการติดตั้ง Git Credential Manager เข้ามาด้วยตั้งแต่ขั้นตอนในการติดตั้ง Git



ส่วนใน Mac OS X นั้นใช้ osxkeychain และใน Linux ใช้ gnome-keyring

```
#Mac OS X
#git config --global credential.helper osxkeychain

#Linux
sudo apt-get install libgnome-keyring-dev

cd /usr/share/doc/git/contrib/credential/gnome-keyring

sudo make

git config --global credential.helper
/usr/share/doc/git/contrib/credential/gnome-keyring/git-
credential-gnome-keyring
```

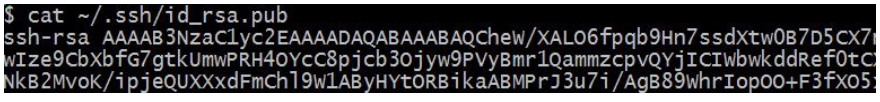
Reference [5], [6]

SSH Key จะเป็นสร้าง Key ที่ใช้ในการเข้ารหัสข้อมูลขึ้นคือไฟล์ Public Key(id_rsa.pub) กับ Private Key(id_rsa) ที่เป็นคู่กันเพื่อยืนยันตัวตนของผู้ใช้
โดยใช้คำสั่งดั่งนี้

```
#คำสั่งสร้าง SSH key โดยที่ถูกเก็บไว้ที่ Path ~/.ssh/ หรือ
C:\Users\Pakin\.ssh
$ssh-keygen -t rsa -C "example@email.com"

#ถ้าเครื่องมีการสร้าง SSH Key ไว้แล้วระบบจะถามว่าต้องการเขียนทับข้อมูล SSH
Key เดิมเลยไหม และจะมีการถาม Password ในการป้องกันคนอื่นๆ มาอ่านไฟล์
SSH Key ของเรา ขั้นตอนนี้ถ้าไม่ต้องการใส่รหัสก็สามารถ Enter ข้ามไปได้เลย

cat ~/.ssh/id_rsa.pub
```



Public Key

Public Key (id_rsa.pub) กับ Private Key (id_rsa) สามารถที่จะใช้เข้าหรัสข้อมูลได้ทั้งคู่ และถ้าต้องการเปิดอ่านข้อมูลที่เข้ารหัสต้องใช้ Key อีกอันที่เป็นคู่กัน เช่นถ้าใช้ Private Key (id_rsa) ในการเข้ารหัสข้อมูลก็ต้องถอดรหัสด้วย Public key (id_rsa.pub) ที่คู่กัน(สร้างมาพร้อมกัน) วิธีนี้จะ

สามารถยืนยันตัวตนของผู้ใช้กับ Git Hosting ได้และวิธีนี้น่าจะมีความปลอดภัยมากกว่า HTTPs ถ้าเราไม่ปล่อยให้ Key หลุดออกไป

เมื่อทำการสร้าง SSH Key แล้ว เราจะนำ Public Key(id_rsa.pub) ไปเพิ่มเข้าใน Git Hosting ซึ่งขั้นตอนในการเพิ่มของทั้ง Bitbucket, GitHub, GitLab ก็คล้ายๆ กัน

Add SSH Key on GitHub

Add SSH Key on Bitbucket

สรุป

Git กลายมาเป็นสิ่งที่จำเป็นและต้องใช้สำหรับผมไปแล้ว ในการทำงานกับ Source Code เหตุผลก็ได้อธิบายไปข้างต้นแล้ว และแทบไม่มีข้อเสียอะไรเลย ช่วยเก็บ Source Code ให้ไม่หาย สามารถย้อนไปเวร์ชั่นเก่าๆ ได้เมื่องานมีปัญหา ชีวิตการในการเขียนโปรแกรมดีขึ้นเยอะ

คนที่กำลังสนใจ(สับสน)อยู่ว่า Git คืออะไร ควรใช้ดีไหม และทำไมถึงต้องใช้ แนะนำให้ลองใช้เลยครับ และเราจะขอบคุณตัวเราเองเมื่อเวลาผ่านไปที่เราได้รู้จักกับ Git

> *"Git is your friend"*

ผมเขียนเรื่องของ Git เป็นบล็อกแรก หากผิดพลาดประการใดต้องขออภัยและคำแนะนำด้วยครับ

เรื่องที่ยังไม่ได้กล่าวถึง ขอติดไว้ก่อน จะเขียนเป็น Blog แยกครับ -__-)"

- Commit Message
- Merge Conflict (Merge tools)
- Git Flow
- Git Submodule, Git Subtree

References

[1] รู้จักกับ Git ประวัติศาสตร์และแนวคิดของระบบจัดการซอร์ส
[2] พบช่องโหว่ Remote Code Execution บน Git รุ่นเก่ากว่า 2.7.1 ผู้ใช้ควรอัพเดทโดยเร็ว
[3] Git 2.10 ออกแล้ว แสดงสถานะความคืบหน้าของ git push อย่างละเอียด
[4] Semantic Versioning
[5] Credential Helper
[6] How to use git with gnome-keyring integration