

Git คืออะไร ? + พร้อมสอนใช้งาน Git และ Github

Posted on August 23, 2015 by



[Chai Phonbopit](#)



ถ้าพูดถึง Git ณ ชั่วโมงนี้หากใครที่ไม่เคยได้ยินหรือไม่เคยใช้งานเลย ก็ต้องบอกว่าท่านเขยมากครับ สำหรับนักพัฒนาแล้ว Git เป็นสิ่งที่จำเป็นมากๆ แม้ว่าเราจะทำงานคนเดียวหรือทำงานเป็นทีมก็ตาม ฉะนั้นบทความนี้ผมจึงเขียนขึ้นเพื่อสำหรับมือใหม่ที่กำลังศึกษา หรือยังไม่รู้จัก Git ให้สามารถใช้ Git เบื้องต้นได้

สำหรับบทความก่อนหน้าที่ผมเขียนก็เป็นสรุปๆ จาก Try Git และ Cheat Sheet ตามนี้ครับ

- [Git Cheat Sheet](#)
- [สรุปจากการเรียน Try Git](#)

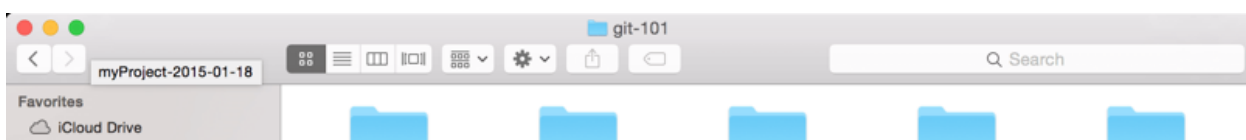
ส่วนเนื้อหาของบทความนี้ก็ประกอบไปด้วย

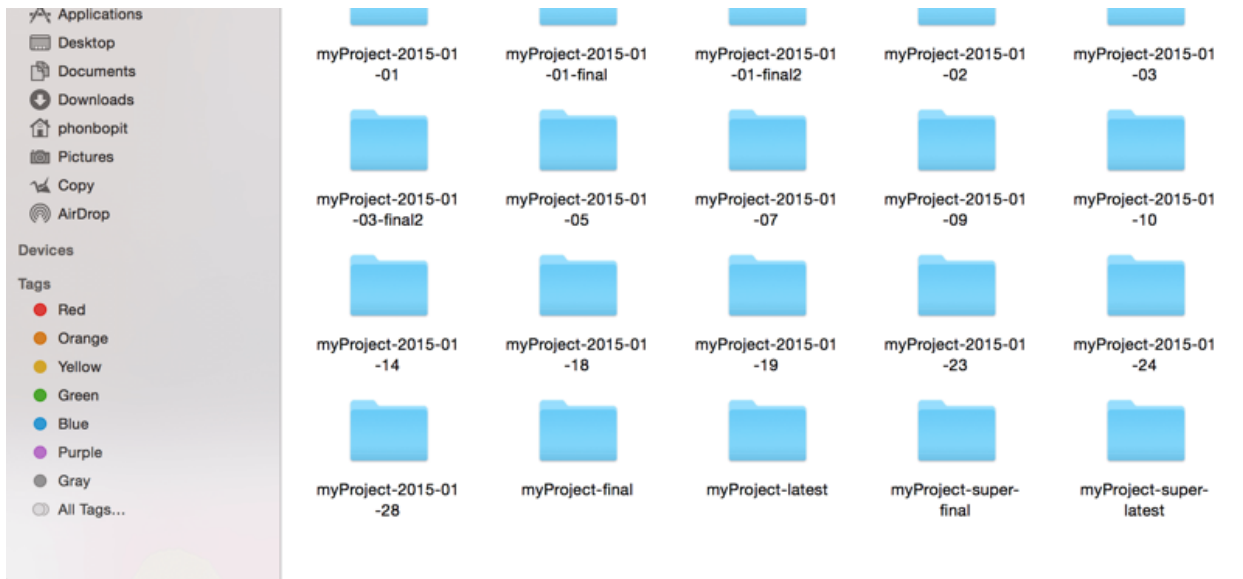
Table of Contents

- Step 0 : ปัญหาการไม่ใช้ Git
- Step 1 : Git คืออะไร ?
- Step 2 : เริ่มต้นติดตั้ง Git
- Step 3 : สร้างโปรเจ็คแรกด้วย Git
- Step 4 : รู้จักกับ Github
- Step 5 : Git GUI / Client
- Step 6 : Workshop

ปัญหาการไม่ใช้ Git

สำหรับคนที่ยังไม่เคยได้ใช้ Git หรือกำลังคิดว่า Git มันมีประโยชน์อย่างไร ให้ท่านลองคิดว่าเคยเจอปัญหาเหล่านี้หรือไม่ ?





- Backup Project ทุกๆวัน โดยการ copy folder แล้วเปลี่ยนชื่อ เช่น ตั้งเป็นวันเดือนปี เป็นต้น
- หากวันใดอยากกลับไปทำงานที่เคยทำไว้ จะหา project ที่เรา backup ไว้อย่างไร? จำได้ไหมว่าที่แก้ไปนั้น แก้ไว้วันไหน ?
- หากเผลอลบโค้ด หรือทำโค้ดหาย อยากจะกู้คืนทำยังไง ? ใช้ Backup เก่า? หากต้องการแก้แค่ก่อนหน้า ไม่ได้แก้ทั้งหมดของวันละ ?
- อยากทดลองทำ feature ใหม่ๆจากโปรเจ็คเก่า จะทำยังไง โดยไม่ต้องก๊อปปี้ทั้งโปรเจ็ค แล้วมาเปลี่ยนชื่อโฟลเดอร์ ?
- ปัญหา Hard Disk เต็ม เนื่องจากนั่ง backup ทุกๆวัน จนเนื้อที่ไม่พอ
- ทำงานกันเป็นทีม ยังต้องเซฟใส่ Flash Drive ส่งให้กันอยู่หรือเปล่า ?
- ทำงานกันเป็นทีม แล้วจะแบ่งแยกงานกันยังไง แล้วถ้าเกิดว่าแก้ไขที่เดียวกัน ถ้าใช้การเซฟลง Flash Drive จะแก้ปัญหายังไง ? ช่วยกันนั่งไล่โค้ด ? เสียเวลาไหม :)
- หากงานที่เราทำเป็น Production ใช้งานไปแล้ว แต่เราอยากพัฒนาโดยไม่กระทบกับ Production จะทำยังไง ?

หากด้านบน เป็นปัญหาที่ท่านพบเจออยู่ทุกวัน ได้เวลาแล้วที่ท่านจะมาเรียนรู้การใช้งาน Git กันครับ

Step 1 : Git คืออะไร ?

Git คือ Version Control ตัวหนึ่ง ซึ่งเป็นระบบที่มีหน้าที่ในการจัดเก็บการเปลี่ยนแปลงของไฟล์ในโปรเจ็คเรา มีการ backup code ให้เรา สามารถที่จะเรียกดูหรือย้อนกลับไปดูเวอร์ชันต่างๆของโปรเจ็คที่ใด เวลาใดก็ได้ หรือแม้แต่ดูว่าไฟล์นั้นๆ ใครเป็นคนเพิ่มหรือแก้ไข หรือว่าจะดูว่าไฟล์นั้นๆถูกเขียนโดยใครบ้างก็

สามารถหาได้ ฉะนั้น Version Control ก็เหมาะอย่างยิ่งสำหรับนักพัฒนาไม่ว่าจะเป็นคนเดียว โดยเฉพาะอย่างยิ่งจะมีประสิทธิภาพมากหากเป็นการพัฒนาเป็นทีม

อ่านรายละเอียดเพิ่มเติมได้ที่ [Pro Git : แปลไทย](#)

Step 2 : เริ่มต้นติดตั้ง Git

แน่นอนเว็บสำหรับดาวน์โหลดและติดตั้ง Git ก็ที่เว็บนี้เลย เลือกลงตาม OS ที่ใช้งาน [Git SCM](#)

อีกทั้งภายในเว็บยังมีบทความทั้ง Tutorials หรือ Documents ต่างๆให้อ่านอีกด้วย

หากใช้ Linux ก็ติดตั้งง่ายๆ ผ่าน Terminal ด้วย:

```
sudo apt-get install git
```

หรือบน Mac OS X ถ้าไม่ใช่ตัว Git Installer ด้านบน ก็ติดตั้งผ่าน Homebrew ได้เช่นกัน

เมื่อดาวน์โหลดและติดตั้ง Git เรียบร้อยแล้ว สิ่งที่ต้องทำต่อมาก็คือ Setup ชื่อและอีเมลสำหรับใช้งาน Git ครั้น ตั้งค่าผ่าน Terminal ด้วย:

```
git config --global user.name "YOURNAME"  
git config --global user.email "your@email.com"
```

เช็คสถานะ ว่า config อะไรไปแล้วบ้างด้วย

Step 3 : สร้างโปรเจกต์แรกด้วย Git

เริ่มต้นสร้างโปรเจกต์ขึ้นมาเปล่าๆ ผมทำการตั้งชื่อว่า git101 ภายในประกอบด้วยไฟล์ index.html 1 ไฟล์ ง่ายๆดังนี้

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <title>Introduction to Git and Github by DevAhoy</title>
```

```
</head>
<body>
  <h1>Hello World</h1>
</body>
</html>
```

git init

เปิด Terminal แล้วไปที่ Folder ของโปรเจ็คเรา จากนั้นสั่ง:

```
git init
```

```
Initialized empty Git repository in /path/to/your/project/.git/
```

คำสั่ง `git init` เอาไว้เพื่อสร้าง git repository เปล่าๆขึ้นมา โดย Git จะทำการสร้างโฟลเดอร์ `.git` ขึ้นมาภายในโปรเจ็คของเรา (Hidden ไว้อยู่)

บางครั้งเราจะเรียกโปรเจ็คที่ใช้ Git ว่า repository ก็ได้

git status

ต่อมาทดสอบสั่งรัน:

เพื่อตรวจสอบสถานะ repository ของเรา จะเห็นข้อความประมาณนี้

```
On branch master
```

```
Initial commit
```

```
Untracked files:
```

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

```
index.html
```

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

ข้อความด้านบน จะทำให้เรารู้ว่า Git บอกว่าไฟล์ของเรา `index.html` ยังไม่ได้ถูก track

git add

วิธีที่เราจะทำให้ไฟล์เราโดน track โดย Git ก็คือการใช้ `git add FILENAME` เช่น

ตอนนี้ไฟล์ของเราได้ถูก track โดย Git แล้ว ต่อมาลองเช็คสถานะด้วย `git status` อีกครั้ง จะเห็นว่าสถานะของ repository เราได้เปลี่ยนไปแล้ว

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: index.html

เราสามารถ ignore ได้ว่าจะให้ git ไม่ต้องเก็บไฟล์ไหน โดยสร้างไฟล์ชื่อ `.gitignore` ไว้ที่ root folder ภายในก็กำหนดค่าเช่น `*.log` ไม่ต้องเก็บไฟล์ `.log` ทั้งหมดไว้บน git เป็นต้น

ตอนนี้ไฟล์ `index.html` ของเราอยู่ใน staged ซึ่งเป็นสถานะที่พร้อมจะทำการ commit แล้ว

เกี่ยวกับ Git Life Cycle

- **Staged** : คือสถานะที่ไฟล์พร้อมจะ commit
- **unstaged** : คือไฟล์ที่ถูกแก้ไขแต่ยังไม่ได้เตรียม commit
- **untracked** : คือไฟล์ที่ยังไม่ถูก track โดย Git (ส่วนมากจะเป็นไฟล์ที่เพิ่งสร้างใหม่)
- **deleted** : ไฟล์ที่ถูกลบไปแล้ว

git commit

ต่อมา เราจะทำ commit ด้วยคำสั่ง `git commit -m "YOUR MESSAGE"` ฉะนั้น commit แรกผมจะใส่ข้อความบอกไว้ว่า ได้ทำการเพิ่มไฟล์ index แล้วนะ ดังนี้

```
git commit -m "add sample index page"
```

```
[master (root-commit) 2a33074] add sample index page
1 file changed, 10 insertions(+)
```

```
create mode 100644 index.html
```

git commit คือการสร้าง snapshot ให้กับ repository ทำให้เราสามารถย้อนกลับมาดูว่าเราเปลี่ยนหรือแก้ไข โค้ดอะไรไปบ้างได้

เกี่ยวกับการใช้งาน `git add` เราสามารถ add ที่หลายๆไฟล์ด้วยการใช้ `*` ก็ได้ เช่น

- `git add .` : add ไฟล์ทั้งหมด
- `git add '*.txt'` : add ไฟล์ทั้งหมดที่นามสกุล `.txt` เป็นต้น

git log

เราสามารถตรวจสอบดูได้ว่าเราทำอะไร `git` ไปแล้วบ้าง ด้วยการใช้คำสั่ง

ต่อมาผมทำการสร้างไฟล์เปล่าๆ ขึ้นมาอีก 2 ไฟล์คือ `about.html` และ `contact.html` จากนั้นลองเช็คสถานะด้วย `git status` อีกครั้ง จะได้ดังภาพ

```
On branch master
```

```
Untracked files:
```

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

```
about.html
```

```
contact.html
```

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

สิ่งที่เราต้องทำก็คือ ทำการ `git add` เพื่อให้ไปอยู่ใน `staged` และพร้อมที่จะ `commit` ก็เริ่มเลย

```
git add about.html contact.html
```

```
git commit -m "add empty about and contact page"
```

ลองนึกถึง หากตอนนี้ถ้าเป็นโปรเจกต์จริงๆ แล้วเราเผลอลบไฟล์ไปจะทำยังไง ? `Git` ช่วยได้ครับ ตอนนี้สมมติ ลองลบไฟล์ `index.html` ที่ดูครับ แล้วลอง `git status` ดูว่าเป็นยังไง

On branch master

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

deleted: index.html

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

Git ก็รู้ทันทีว่าไฟล์ของเราถูกลบออกไปแล้ว แต่ Git มัน track ไว้ให้ และลองใช้อีกคำสั่งครับ นั่นก็คือ

git diff

เอาไว้เช็คว่ามีอะไรเปลี่ยนแปลงบ้าง จากข้อความด้านล่าง โค้ดสีแดงคือโค้ดที่ลบ
ถ้ามีสีเขียวแสดงว่าเป็นโค้ดที่ถูกเพิ่ม

```
diff --git a/index.html b/index.html
deleted file mode 100644
index 144f699..0000000
--- a/index.html
+++ /dev/null
@@ -1,10 +0,0 @@
-<!DOCTYPE html>
-<html lang="en">
-<head>
-  <meta charset="UTF-8">
-  <title>Introduction to Git and Github by DevAhoy</title>
-</head>
-<body>
-  <h1>Hello World</h1>
-</body>
-</html>
\ No newline at end of file
```

git reset

ทีนี้อยากจะ reset กู้คืนไฟล์ที่เผลอลบไป เช่น เผลอลบ index.html ก็เพียงแค่ใช้คำสั่ง

แต่ถ้าเราดันเผลอลบ แล้วก็ยังไม่ add เข้าสู่ staged แล้ว ก็ต้องใช้

แล้วถ้าเราดันไป commit มันซะแล้ว วิธีที่จะย้อนกลับไป commit ล่าสุดก็คือ

ส่วนถ้าเราใช้คำสั่งนี้ จะเป็นการลบไฟล์ index.html ออกจากการ track ของ Git แต่ไม่ได้ลบไฟล์ในโปรเจ็คของเรา

```
git remove --cached index.html
```

Git branch

ต่อมาเป็นการพูดถึง git branch เป็น feature ที่ช่วยให้นักพัฒนาสามารถที่จะทำงานได้สะดวกขึ้น ยกตัวอย่างเช่น เรามีโค้ดที่ติดอยู่แล้ว แต่อยากจะทดลองอะไรนิดๆหน่อยๆ หรือแก้ไขอะไรก็ตาม ไม่ให้กระทบกับตัวงานหลัก ก็เพียงแค่สร้าง branch ใหม่ขึ้นมา เมื่อแก้ไขหรือทำอะไรเสร็จแล้ว ก็ค่อยเซฟกลับมาที่ master เหมือนเดิม

วิธีการสร้าง branch ใหม่ จะใช้คำสั่ง

```
git branch create_new_page
```

ลองดูรายชื่อ branch ทั้งหมดที่เรามี ดังนี้

```
git branch
```

```
* master  
  create_new_page
```

ตอนนี้เรามี 2 branch อยู่ในเครื่อง วิธีการสลับ branch เราจะใช้ git checkout

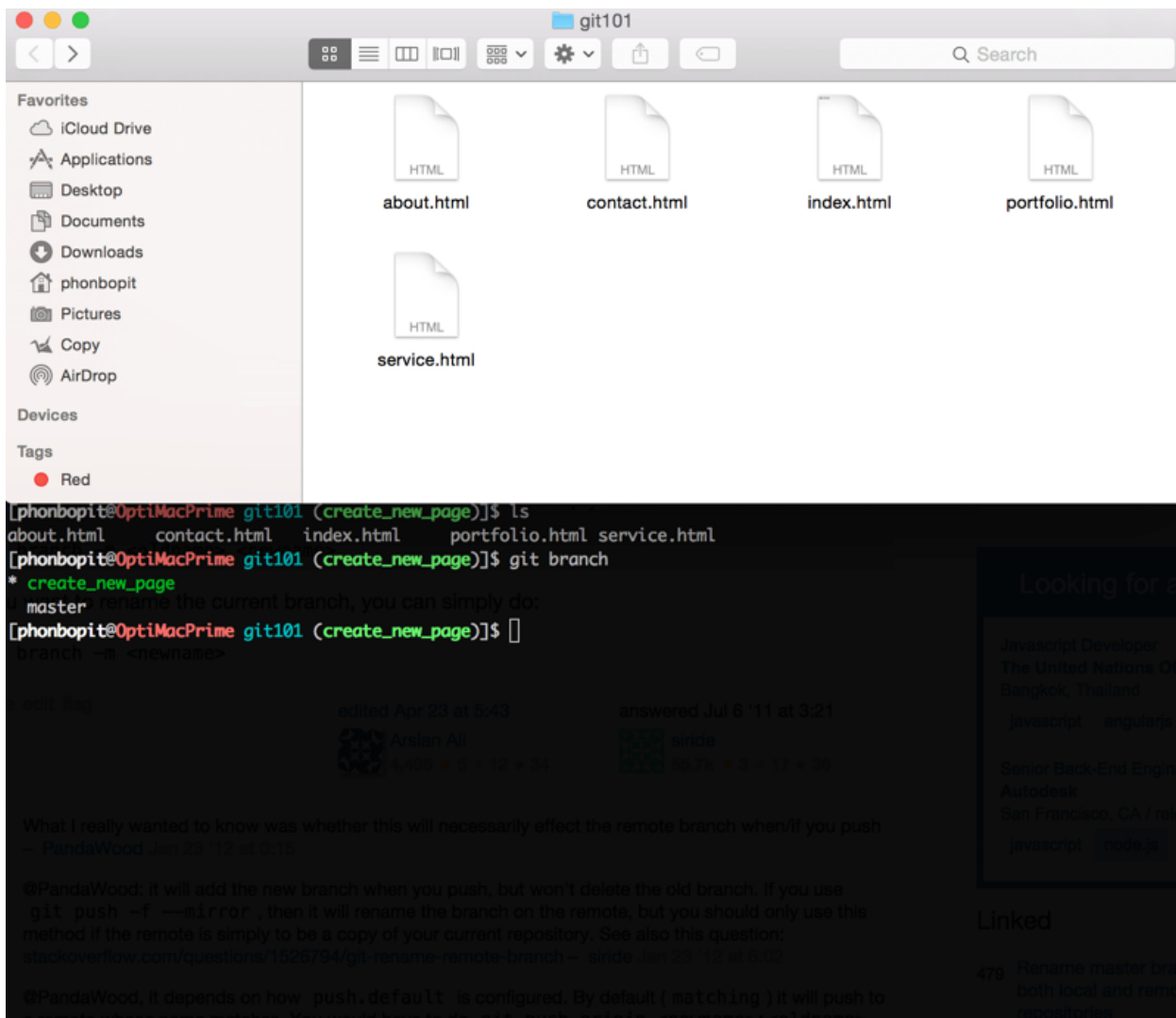
git checkout

git checkout branch_name เอาไว้สำหรับเปลี่ยน branch ตัวอย่างเช่น git branch create_new_page ย้าย branch ไปยัง create_new_page

เราสามารถสั่ง `git checkout -b create_new_page` แบบสั้นๆได้ เป็นการสร้าง branch ใหม่และ checkout ให้เลย เป็นการรวมสองคำสั่งเป็นคำสั่งเดียว

ตนเอง

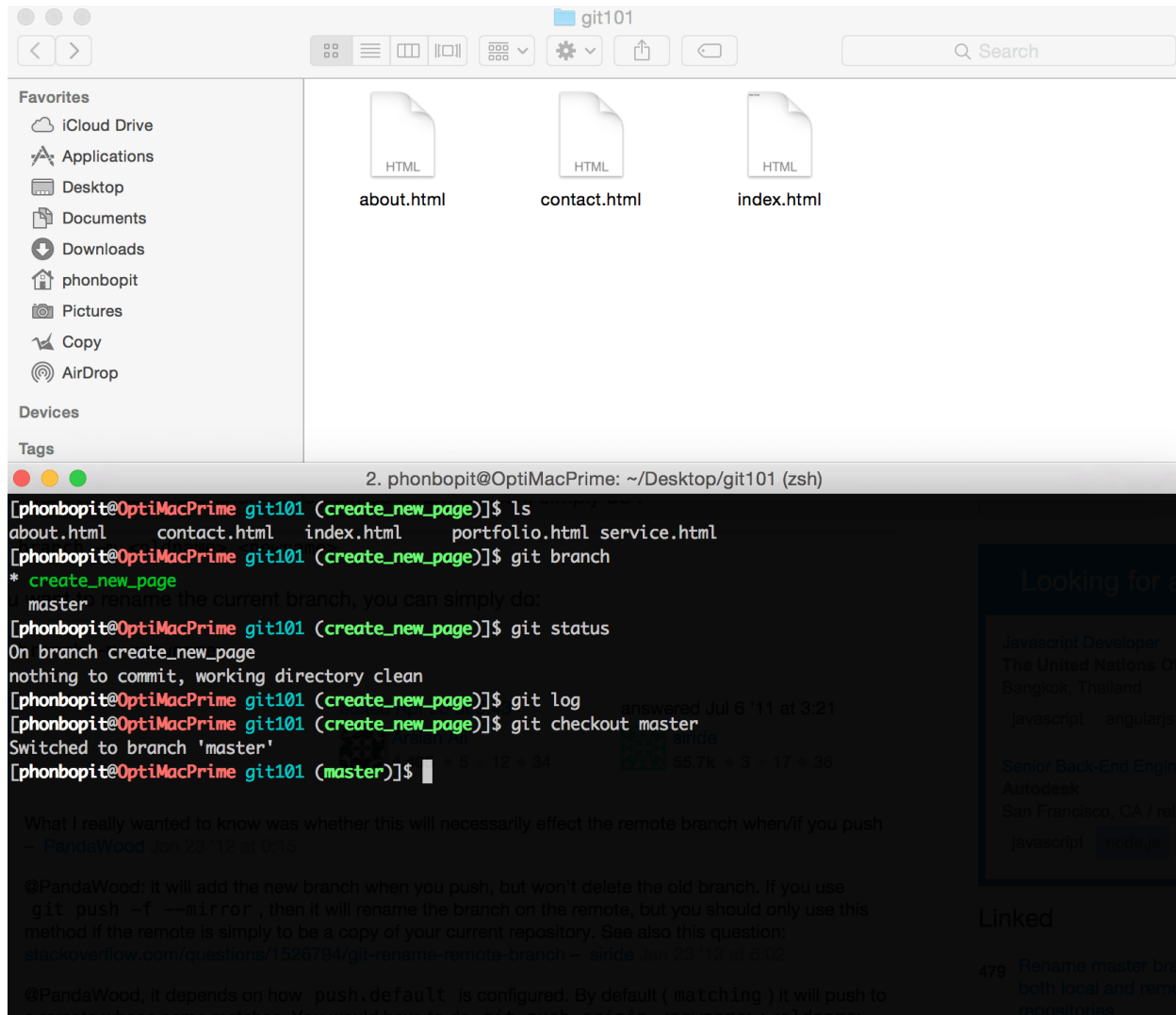
ใน branch `create_new_page` ผมเพิ่มไฟล์มาสองไฟล์ `service.html` และ `portfolio.html` จะเห็นว่าไฟล์ในโฟลเดอร์จะเป็นแบบนี้



จากนั้นทำการ commit ไฟล์ที่เพิ่มลงไปใหม่

```
git add '*.txt'
git commit -m "I just added new two pages on create_new_page branch"
```

แล้วลองย้อนกลับไป brach master ด้วย `git checkout master` ไฟล์ในโฟลเดอร์จะไม่มี `service.html` และ `portfolio.html` เนื่องจากเราไม่ได้สร้างที่ brach master นั้นเอง



git merge

ที่นี้สมมติว่าเรา ทดลองทำ feature อะไรใหม่ๆ แล้วคิดว่ามันดีแล้ว อยากเอามา รวมกับ master เราก็ใช้คำสั่ง `git merge` โดยตัวอย่าง ผมสร้างไฟล์ `service.html` และ `portfolio.html` ที่ branch `create_new_page` และจะนำมารวมกับ master ก็เพียงแค่ใช้:

```

git checkout master
git merge create_new_page

```

```
git merge create_new_page
```

1. เริ่มต้นด้วยการ ไปที่ branch หลัก (master)
2. จากนั้นก็สั่ง `git merge branch_name` จาก branch ที่ต้องการ มาที่

master

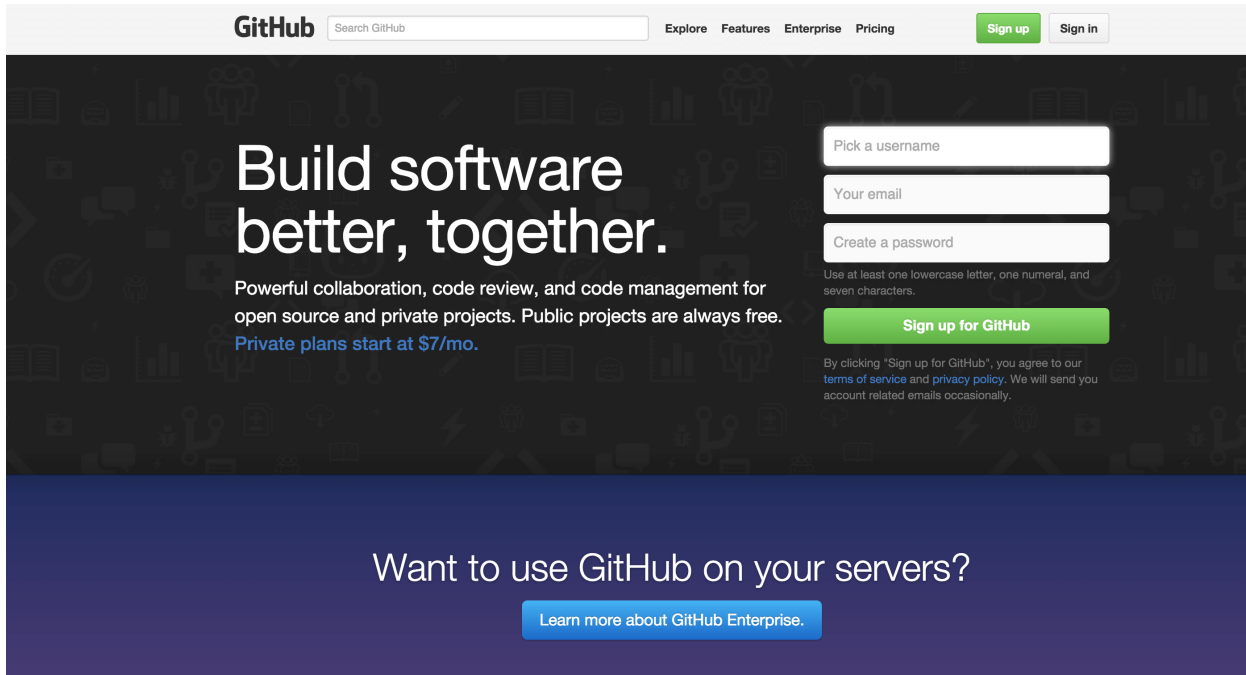
```
Updating 9d0d9f8..4f176d9
Fast-forward
 portfolio.html | 0
 service.html   | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 portfolio.html
 create mode 100644 service.html
```

จะเห็นว่าตอนนี้ brach master มีไฟล์ 2 ไฟล์ที่เพิ่มเข้ามาใหม่ พร้อมกับ commit message ที่เคย commit ไว้ก็จะถูกรวมมาอยู่ใน master หหมด ทีนี้เมื่อเราทำ feature อะไรเสร็จแล้ว brach create_new_page ก็ไม่ต้องการละ สามารถลบ branch ได้ด้วยคำสั่ง

```
git branch -d branch_name
git branch -d create_new_page
```

สำหรับ Git เบื้องต้นในส่วนของ local ก็หมดเพียงเท่านี้ครับ ต่อไปจะเป็น Git บน Server ห

Step 4 : รู้จักกับ Github



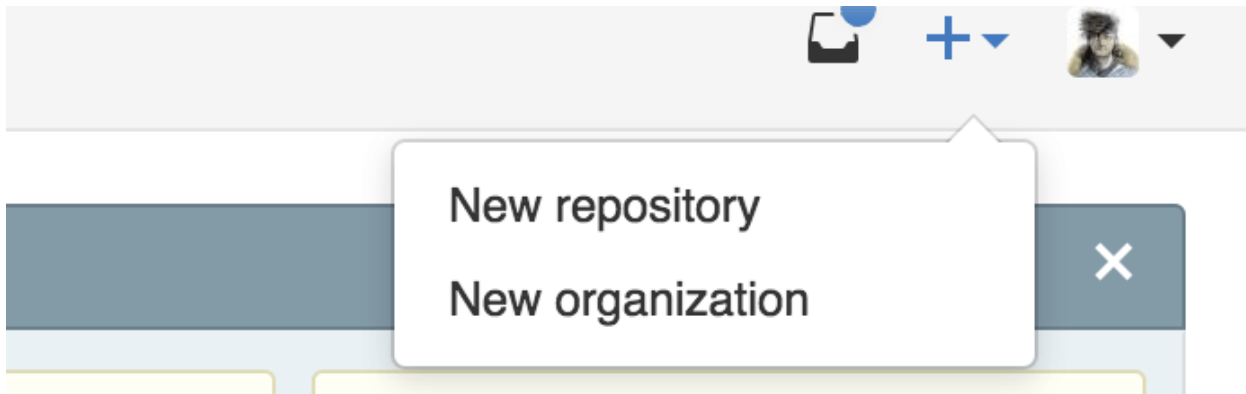
Github คืออะไร ? Github เป็นเว็บเซิร์ฟเวอร์ที่ให้บริการในการฝากไฟล์ Git (ทั่วโลกมักนิยมใช้ในการเก็บโปรเจกต์ Open Source ต่างๆ ที่ตั้งๆ ไม่ว่าจะเป็น Bootstrap, Rails, Node.js, Angular เป็นต้น)

ขั้นตอนแรก ให้ทำการสมัครสมาชิกกับ Github จากนั้นเมื่อสมัครสมาชิกแล้ว เราสามารถที่จะสร้าง Repository ของเราแล้วเอาไปฝากไว้บน Github ได้ครับ (การทำงานทุกอย่างก็เหมือนตอนทำ local)

จุดเด่นของ Github คือใช้ฟรี และสร้าง repository ได้ไม่จำกัด แต่ต้องเป็น public repository เท่านั้น หากอยาก private ก็ต้องเสียตังครับ

Create Repo

ตอนนี้ผมจะทำการสร้าง repository โดยจะเอาโค้ดที่ได้เขียนไว้ ก็โปรเจกต์ git101 จาก Step 3 นั้นแหละครับ คราวนี้นั้นจะถูกอัปโหลดขึ้นเซิร์ฟเวอร์แล้ว วิธีการก็ง่ายๆ แค่กด + มุมบนขวา แล้วเลือก **New repository**



จากนั้นก็ทำการตั้งชื่อ repo รวมถึงคำอธิบาย

Owner: Phonbopit / Repository name: git101 ✓

Great repository names are short and memorable. Need inspiration? How about **verdant-octo-turtle**.

Description (optional): Sample git repo

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository


ไม่ต้องติ๊กเลือก *initialize this repository with a README* เนื่องจากเรามี repository อยู่แล้ว

ซึ่งจริงๆแล้ว Github ก็มีคำอธิบายบอกไว้แล้วว่าเราจะต้องทำยังไงต่อ ดังรูป

Phonbopit / git101

Un

Quick setup — if you've done this kind of thing before

 Set up in Desktop

 or

HTTPS

SSH

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# git101" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:Phonbopit/git101.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:Phonbopit/git101.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

สิ่งที่เราต้องทำต่อมาก็คือ เพิ่ม url ของ remote (Server) เพื่อให้รู้ว่าเราจะฝากโค้ดไว้ที่ใด

กลับมาที่เครื่องเรา เข้าไปที่โฟลเดอร์ปัจจุบัน จากนั้นพิมพ์คำสั่ง

```
git remote add origin git@github.com:Phonbopit/git101.git
```

ที่อยู่ url ของ repository แต่ละคนก็ต่างกันไปในะครับ อย่าเผลอก็อปปี้ url ของผมไปละ ซึ่งบางคนอาจจะเป็น https ก็ไม่ต้องแปลกใจครับ

ต่อมาเช็คให้แน่ใจว่าเราทำการเพิ่ม remote ได้ถูกต้องด้วย `git remote -v`

```
origin  git@github.com:Phonbopit/git101.git (fetch)
```

```
origin  git@github.com:Phonbopit/git101.git (push)
```

Git Push

ต่อมาเราจะทำการส่งโค้ดจากเครื่อง local ไปที่ Github ด้วยคำสั่ง

```
git push -u origin master
```

- `-u` : เอาไว้จำ parameter origin master ต่อไปก็แค่พิมพ์ `git push`
- `origin` : คือชื่อ alias ของ remote (github)
- `master` : คือชื่อ branch ที่เราต้องการ push ขึ้นไป

จากนั้นก็ใส่ชื่อ account github และ password (โดย password เราจะไม่เห็นว่าพิมพ์อะไรลงไป เมื่อพิมพ์ password เสร็จให้กด Enter)





สำหรับคนที่เลือกเป็น `https` จะต้องใส่ Username และ Password ทุกครั้งที่มีการ connect github แต่ถ้าเลือกแบบ `ssh` จะไม่ต้องใส่ Password แต่จะต้องไปเช็คค่า ตามนี้ [Generating SSH Keys](#)

เพื่อ push โค้ดเรียบร้อย จะได้ข้อความประมาณนี้

```
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 829 bytes | 0 bytes/s, done.
Total 8 (delta 1), reused 0 (delta 0)
To git@github.com:Phonbopit/git101.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

เมื่อเราทำการ Push Code ขึ้นไปบน Github แล้วเมื่อลองเข้าเว็บไซต์ <https://github.com/Phonbopit/git101> ก็จะเห็นไฟล์ของเราขึ้นไปแล้ว(url แต่ละคนจะไม่เหมือนกันนะครับ ต่างกันที่ username และชื่อ repo)

The screenshot shows the GitHub interface for a repository named 'git101' by user 'Phonbopit'. At the top, it says 'Sample git repo — Edit'. Below this, there's a summary bar showing '3 commits', '1 branch', '0 releases', and '1 contributor'. A green button with a plus icon is next to 'Branch: master', followed by 'git101 / +'. Below the summary bar, there's a section titled 'add service and portfolio' with a light blue background. It shows a commit by 'Phonbopit' from an hour ago with the latest commit hash '4f176d99e6'. At the bottom, there's a link to 'about.html' and a note to 'add empty about and contact page' from 2 hours ago.

 contact.html	add empty about and contact page	2 hours ago
 index.html	add sample index page	2 hours ago
 portfolio.html	add service and portfolio	an hour ago
 service.html	add service and portfolio	an hour ago

Help people interested in this repository understand your project by adding a README.

Add a README

Git Fetch

Git Fetch เป็นการเช็คโค้ดของเราระหว่าง local กับ remote(server) ว่าโค้ดตรงกันล่าสุดหรือไม่ ซึ่งตอนนี้เราเพิ่ง Push ลงไป ทำให้โค้ดมันเหมือนกัน ถ้าเรา fetch ก็ไม่มีไรเกิดขึ้น ฉะนั้นลองเข้าหน้า project ใน Github แล้วกด **Add a README** ปุ่มเขียวๆ เพื่อเพิ่มไฟล์ README.md ผ่านบราวเซอร์เลยครับ

จากนั้นก็พิมพ์ข้อความ เสร็จแล้วกด **Commit new file**

เราก็จะได้ไฟล์ README.md เพิ่มเข้ามาในโปรเจกต์แล้ว ใน Github มี แต่ว่าลองย้อนกลับมาที่โปรเจกต์ local ของเรา ยังคงไม่มีไฟล์ README.md ที่เพิ่งเพิ่มลงไป วิธีการก็คือ ทำการ

จะเห็นข้อความประมาณนี้

```
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:Phonbopit/git101
   4f176d9..a7f9fd4  master    -> origin/master
```

เมื่อลอง `git status` ก็จะได้ดังนี้

On branch master

Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded
(use "git pull" to update your local branch)
nothing to commit, working directory clean

นั่นหมายความว่า โค้ด local ของเรานั้นล้าสมัย หรือว่าตามหลัง master บน

remote อยู่ | commit ของเราจะรวมเคตจาก remote มาที่ local ของเรา
เดียวกันกับการรวมโค้ดคนละ branch นั่นก็คือ git merge ครับ

โดยทำการระบุ branch ที่ต้องการ ในที่นี้คือ origin/master

```
git merge origin/master
```

```
Updating 4f176d9..a7f9fd4
Fast-forward
 README.md | 4 ++++
 1 file changed, 4 insertions(+)
 create mode 100644 README.md
```

ตอนนี้ที่เครื่อง local ก็อัปเดตตรงกับ remote แล้ว และก็มีไฟล์ README.md ที่
สร้างไว้ด้วย คราวนี้ลองนึกถึงกรณีที่ทำงานร่วมกันหลายๆคน แต่ละคนก็ Push
โค้ดขึ้นไปเก็บไว้ และวิธีที่จะอัปเดตโค้ดให้ล่าสุดเสมอก็คือการใช้ git fetch และ
git merge นั่นเอง

Git Pull

จริงๆแล้ว git pull ก็คือรวมโค้ดจาก remote มาที่ local โดยที่เราไม่สามารถรู้
ได้เลยว่าจะรวมโค้ดอะไรบ้าง รู้แค่หลังจาก pull เสร็จแล้วนั่นเอง ซึ่งจริงๆแล้ว git
pull มันก็คือการทำ git fetch และต่อด้วย git merge อัตโนมัตินั่นเอง

Git Conflict

โดยปกติแล้ว git merge จะรวมโค้ดให้เราเองอัตโนมัติ แต่ก็จะมีข้อยกเว้นเมื่อ
แก้ไขไฟล์เดียวกัน ลองนึกถึงกรณีที่เราและเพื่อนร่วมทีม แก้ไขไฟล์เดียวกัน Git จะ
เกิดการ conflict เมื่อเราจะ merge โค้ด โดยไม่รู้ว่าจะใช้โค้ดของเราหรือของ
เพื่อน วิธีแก้ก็คือ ทำการ edit แล้ว commit ไปใหม่นั่นเอง

ตัวอย่างคร่าวๆ ของไฟล์ที่เกิด conflict

```
git pull
```

```
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

และตัวอย่างไฟล์

```
git101
```

```
---
```

```
Sample git repo
```

```
<<<<<<< HEAD
```

```
edit on sublime text.
```

```
=====
```

```
last edit on browser via github.com
```

```
>>>>>>> origin/master
```

format ของไฟล์ conflict จะถูกขึ้นด้วย <<<<<<< HEAD จนถึง ===== สำหรับ
โค้ดส่วนที่เราแก้ไข และ =====ถึง >>>>>>> branch_name ส่วนที่เป็นโค้ดของ
คนอื่นๆ/branch อื่น

วิธีแก้ก็แค่ลบพวกโค้ดส่วนเกินออก แล้วแก้ไขใหม่ให้เรียบร้อย จากนั้นลองเช็ค
สถานะ จะขึ้นประมาณนี้

```
git status
```

```
Your branch and 'origin/master' have diverged,  
and have 1 and 1 different commit each, respectively.
```

```
(use "git pull" to merge the remote branch into yours)
```

```
You have unmerged paths.
```

```
(fix conflicts and run "git commit")
```

```
Unmerged paths:
```

```
(use "git add <file>..." to mark resolution)
```

```
both modified:   README.md
```

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

ก็ commit และ push ได้ปกติแล้วครับ

```
git add README.md
```

```
git commit -m "fixed conflict on README.md"
```

```
git push
```

เป็นอันเรียบร้อย

Git Clone

ในกรณีที่เราไปเจอโปรเจ็ค cool cool น่าสนใจ เราสามารถที่จะเซฟมาลงเครื่องของเราได้เลย เราเรียกการทำแบบนี้ว่าการ clone เหมือนกับการกดดาวน์โหลดโปรเจ็คมานั้นแหละครับ วิธีง่ายๆ ก็คือ ใช้คำสั่ง

วิธีการดูโปรเจ็คที่เราต้องการ ก็อย่างเช่น [gitignore](#) ด้านขวามือจะมี url แสดง ก็จะได้เป็น

```
git clone https://github.com/github/gitignore.git
```

```
Cloning into 'gitignore'...
remote: Counting objects: 4776, done.
remote: Total 4776 (delta 0), reused 0 (delta 0), pack-reused 4776
Receiving objects: 100% (4776/4776), 1.04 MiB | 227.00 KiB/s, done.
Resolving deltas: 100% (2323/2323), done.

Checking connectivity... done.
```

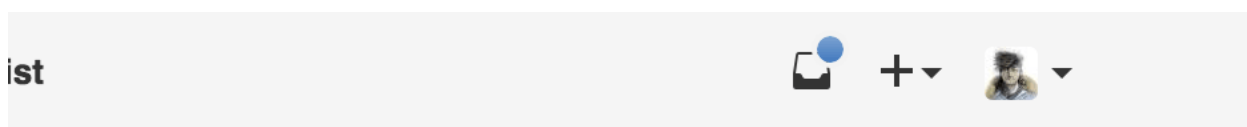
แค่นี้โปรเจ็คบน Github เราก็สามารถจะ clone มาลงเครื่องเราได้แล้ว :)

Fork

Fork เป็น feature ของทาง Github เอาไว้สำหรับ copy โปรเจ็คหรือว่า clone โปรเจ็คอื่นมาเป็นของเรา เพียงแต่ว่าการ clone จะเกิดขึ้นบน remote(server) เท่านั้น ข้อดีของการ Fork ก็คือ กรณีที่เราเจอโปรเจ็คที่น่าสนใจ แล้วเราอยากจะช่วยพัฒนา เราสามารถที่จะ Fork มาเป็นของเรา จากนั้น clone มาที่ local เมื่อแก้ไขโค้ดเสร็จ ก็ทำการ Push โค้ดขึ้นไป ทีนี้ก็จะทำการ Pull Request เพื่อ merge โค้ด (Pull Request ดูหัวข้อถัดไป)

ซึ่งวิธีการ Fork นั้นทำได้ง่ายๆ เช่น ตัวอย่าง จะทำการ Fork [Bootstrap](#) ครับ

กดปุ่ม Fork ด้านมุมบนขวามือ



Watch ▾

5,298

★ Star

85,128

Fork

34,855

Fork your own copy of twbs/bo

เมื่อทำการ Fork แล้ว ก็รอมัน Forking เປปปนง

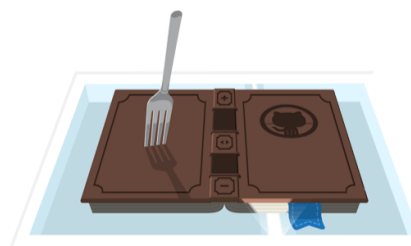
Phonbopit / **bootstrap**

forked from twbs/bootstrap

Forking twbs/bootstrap

It should only take a few seconds.

Refresh



จากนั้น Github จะทำการ copy โปรเจ็คมาอยู่ที่ account เรา (url จะเปลี่ยน แต่ก็จะบอกว่าเรา fork มาจากที่ใด)

Phonbopit / **bootstrap**

forked from twbs/bootstrap

หากเราลอง clone โปรเจ็คที่เรา fork มา ในกรณีที่เรายากช่วยพัฒนา และตัวต้นฉบับมีการ commit เพิ่มตลอด เราจะทำยังไงที่จะให้รู้ว่าโค้ดของเรานั้นเหมือนกับต้นฉบับแล้ว วิธีการก็คือ ทำการเพิ่ม remote ของต้นฉบับ จะเรียกว่า upstream ดังนี้

ตัวอย่างผม Fork Bootstrap และทำการ add upstream ไปที่ original ตอนนี้เมื่อลอง `git remote -v` จะได้ดังนี้

```
origin  git@github.com:Phonbopit/bootstrap.git (fetch)
origin  git@github.com:Phonbopit/bootstrap.git (push)

upstream  git@github.com:Phonbopit/bootstrap.git (fetch)
upstream  git@github.com:Phonbopit/bootstrap.git (push)
```

ถ้าอยากเช็คการอัปเดตจาก original ก็เพียงแค่พิมพ์

```
git fetch upstream

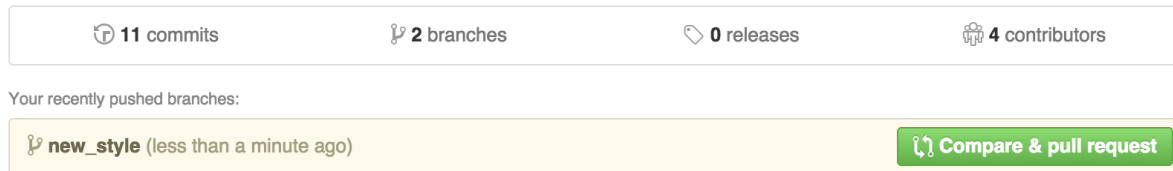
git checkout master
git merge upstream/master
```

ส่วนเวลาเราแก้ไขโค้ดแล้ว Push ขึ้นไป ก็ไม่ต้องกลัวว่าจะไปกระทบกับโค้ดต้นฉบับที่เรา Fork มานะครับ การ Push มันจะไปที่ repo ของเรา ไม่มีทางที่จะไปรวมกับ Original นอกจากเราจะ Pull Request และเจ้าของ repo นั้นกด accept

Pull Request

Pull Request เป็น feature ที่ช่วยให้นักพัฒนาคนอื่นๆ ที่ไม่ใช่เจ้าของ repo สามารถที่จะร่วมกันส่งโค้ดเข้ามา merge รวมกับโค้ดต้นฉบับได้ โดยการใช้งาน Pull Request ต้องทำการ Fork โปรเจ็คมาก่อน

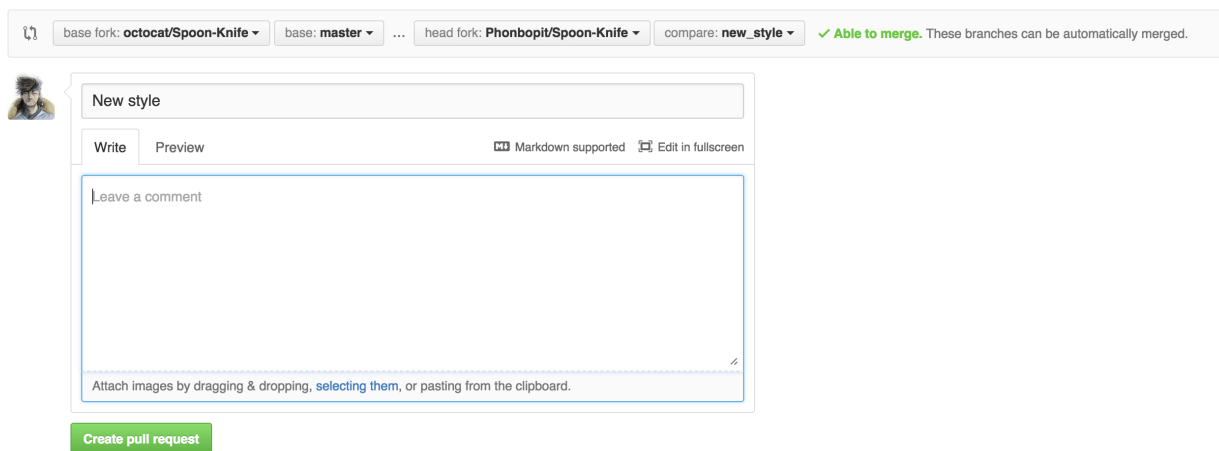
ตัวอย่างการ Pull Request ในกรณีที่เรทำการแก้ไขโปรเจ็คที่ Fork แล้วทำการ commit และ push โค้ดขึ้นมา จะเห็นปุ่ม **Pull Request** ขึ้นมาแสดงว่าเราจะต้องการ Pull Request หรือไม่ อย่างในตัวอย่างด้านล่าง ผมเพิ่ง push โค้ดที่ `branch new_style` ขึ้นมา



เมื่อกดเข้าไปจะเจอหน้า Pull Request เพื่อให้ใส่รายละเอียด มีการเปรียบเทียบระหว่าง Original Repo และตัว Fork ของเรา รวมถึงเปรียบเทียบและให้เลือก branch ที่ต้องการ Pull Request ก็ได้

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



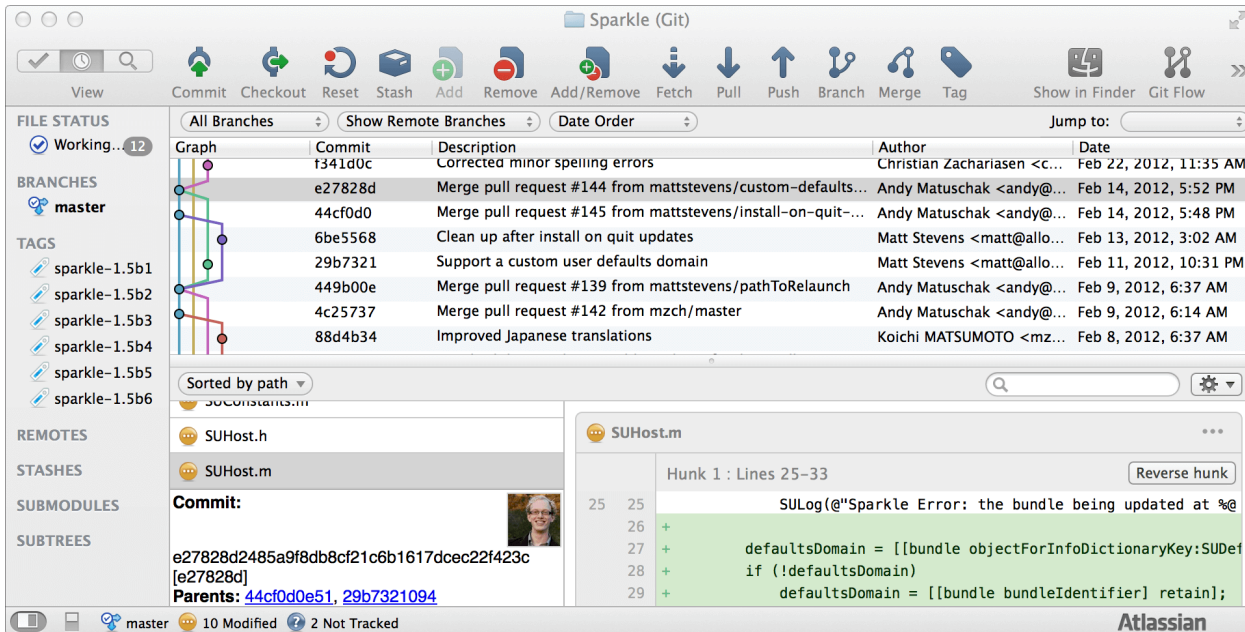
เพื่อ Pull Request ไปแล้ว ก็แค่รอให้ทางเจ้าของ Repository รออนุมัติว่าจะ merge โค้ดของเราหรือไม่

ในกรณีที่ต้องการส่ง Pull Request(PR) เราควรจะแตก branch ออกมา จากนั้นถึงส่ง PR เป็น featureๆ แต่ละ branch ไป ไม่ควรทำใน master

Step 5 : Git GUI / Git Client

สำหรับบางคน อาจจะบ่นว่าใช้งาน Git ยุ่งยากเหลือเกิน ต้องพิมพ์ command line ตลอดเลยหรือเปล่า ? จริงๆแล้วก็มี Program GUI หลายๆตัวให้ใช้งานอยู่เหมือนกันนะครับ ในบทความนี้ก็จะยกตัวอย่างมาคร่าวๆ 2 ตัวครับ ก็คือ SourceTree และ Github Desktop

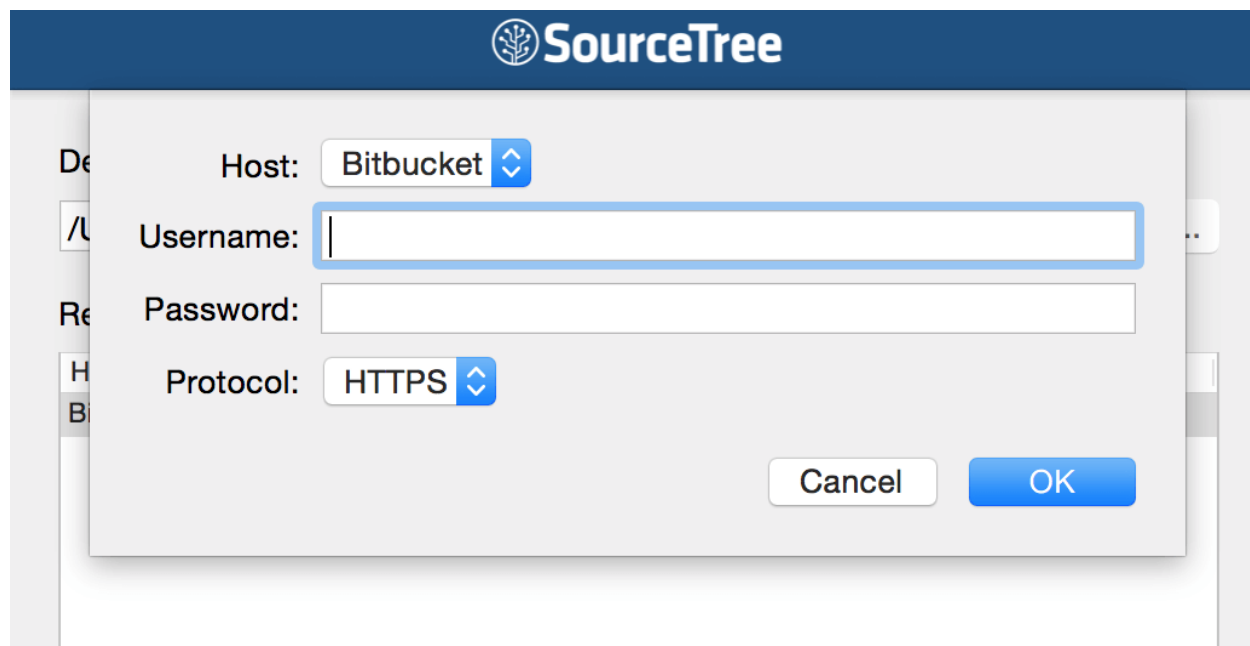
Source Tree

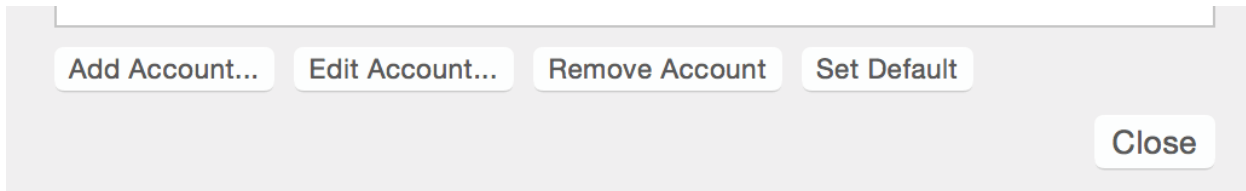


การใช้งาน Source Tree นั้นก็ง่ายเลยครับ

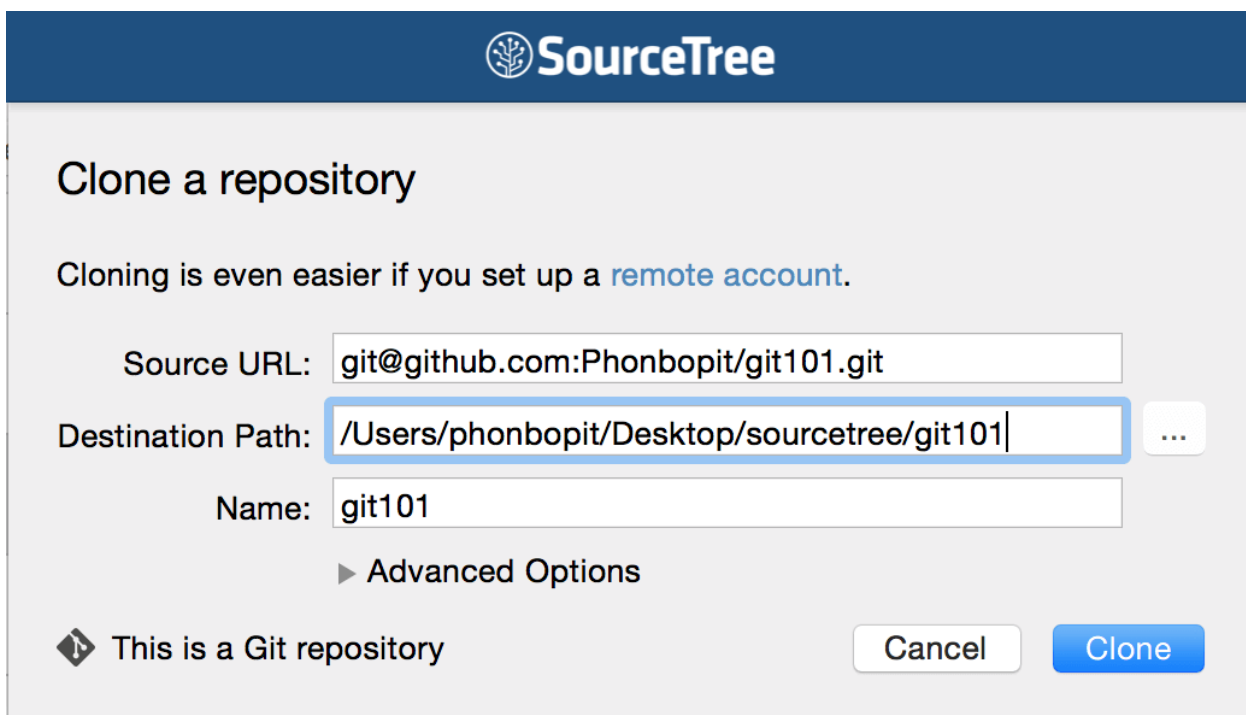
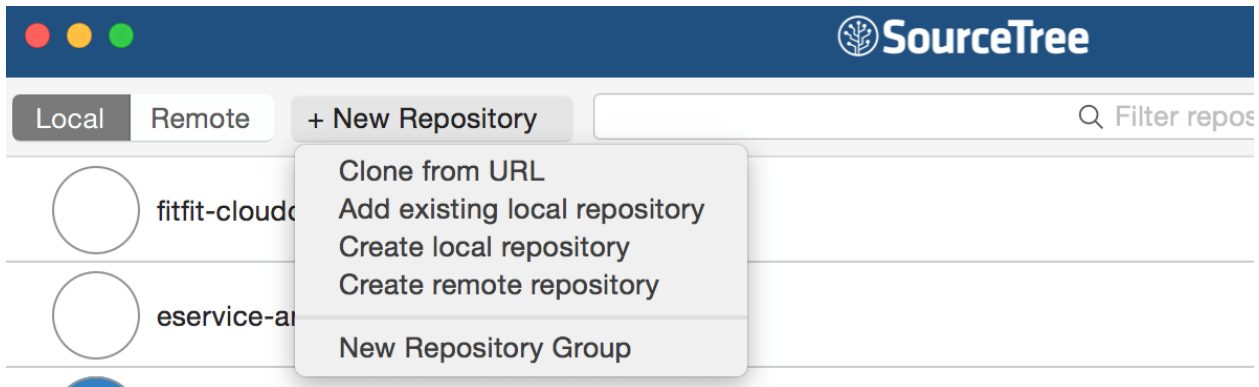
Setting Account สำหรับ BitBucket [สมัครได้ที่นี้](#)

เพื่อเอาไว้สร้าง repository แบบ private ฟรี คล้ายๆกับ Github เลย

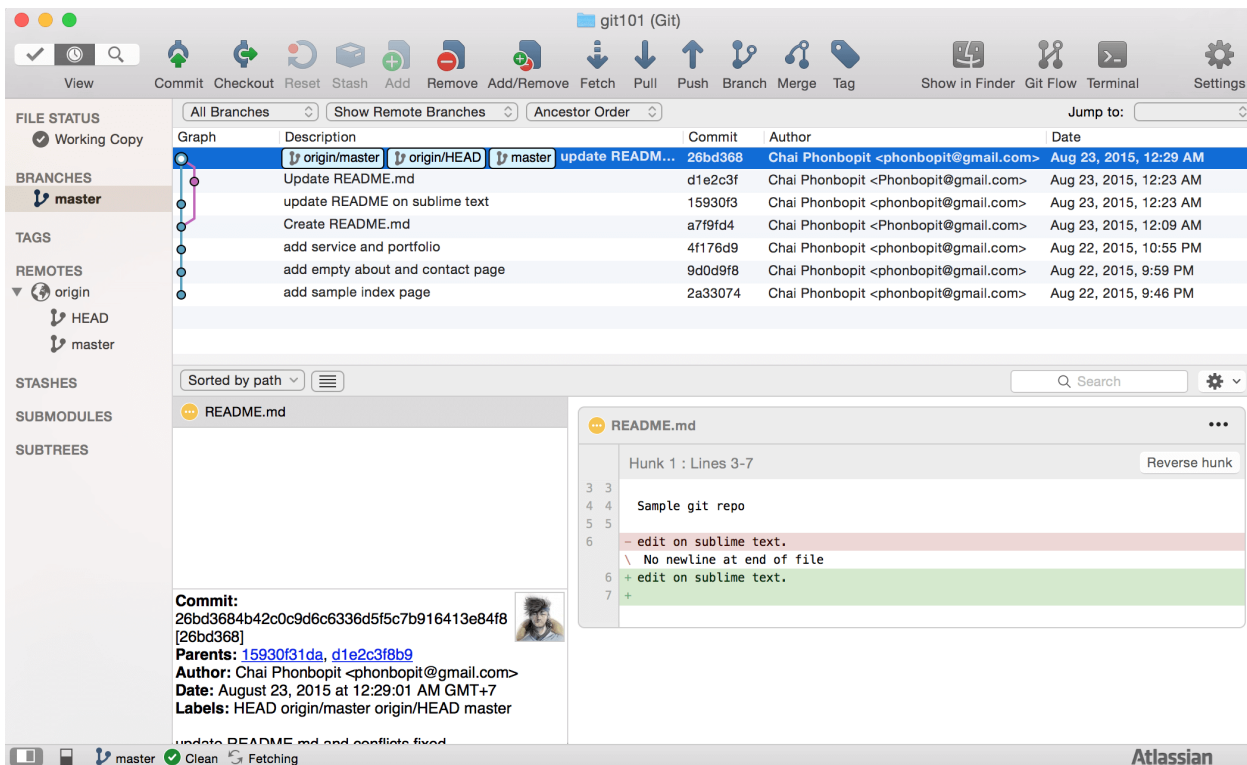




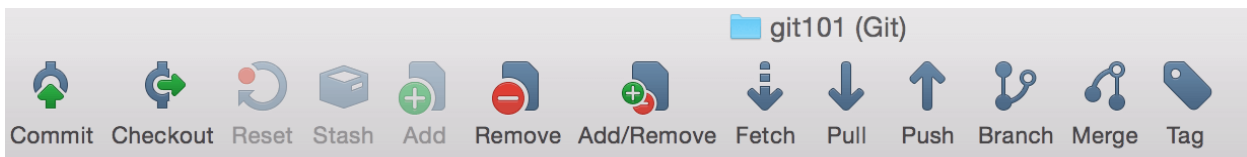
หน้าตาการ clone url ก็จะประมาณนี้ ใส่ url ของ repository ที่เราต้องการจะ clone



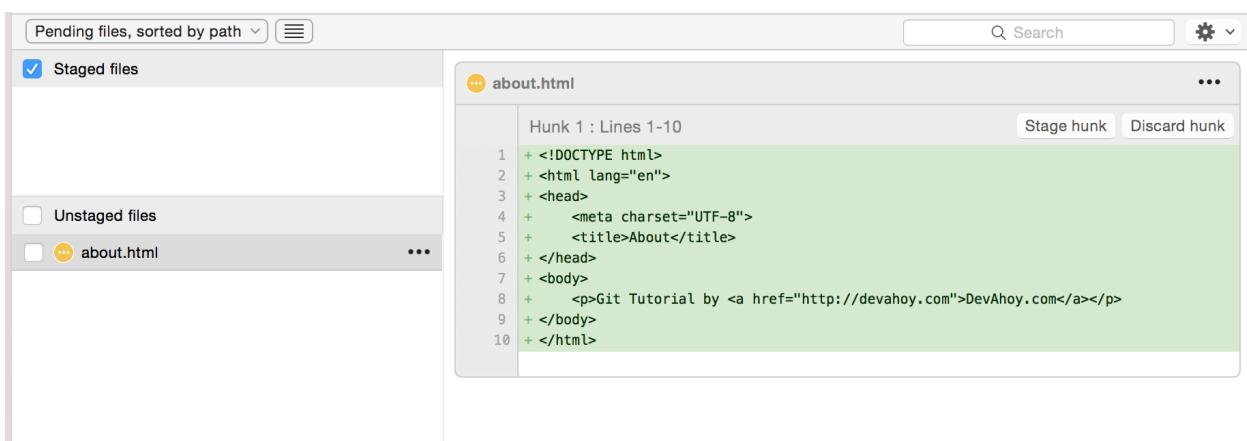
หน้าตาเมื่อเปิดโปรแกรม SourceTree ขึ้นมา จะเห็นว่าผมทำการ clone repo ตัว git101 ก็ให้เห็น history ต่างๆทั้งหมด ว่าแต่ละ commit ทำอะไรไปบ้าง ซึ่งจริงๆ มันก็คือการเรียก git log นั่นแหละ



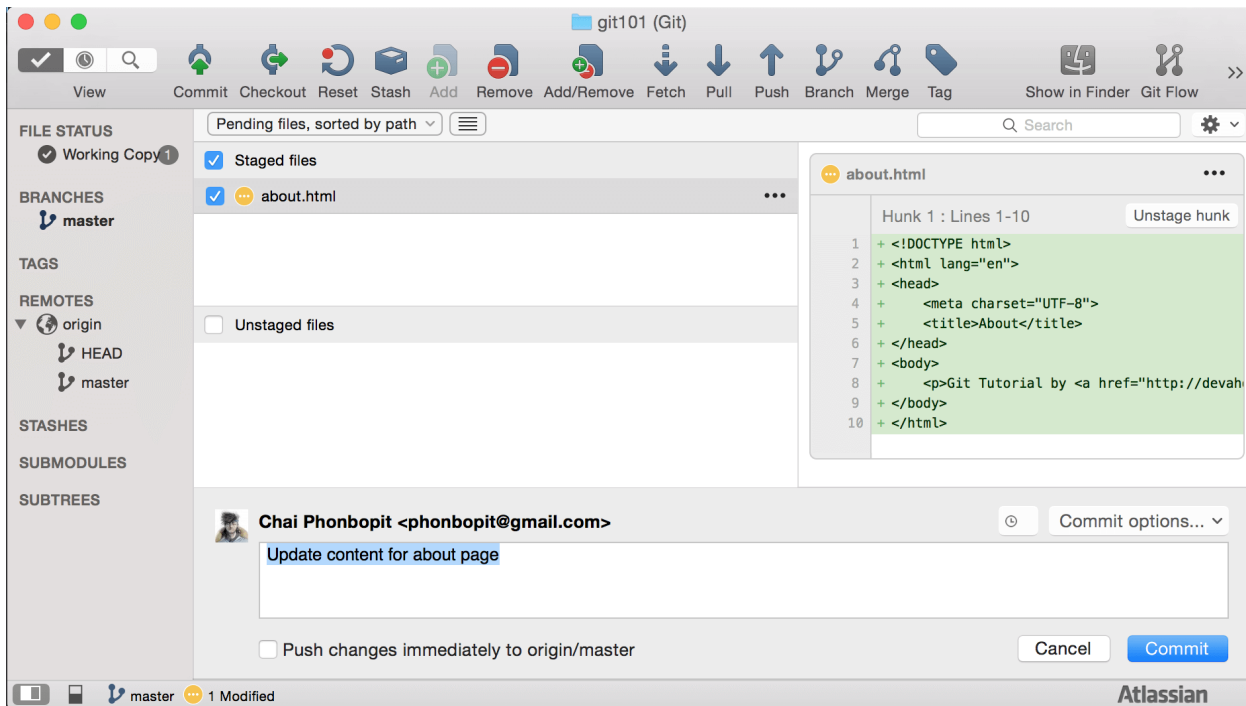
เมนูด้านบนของ SourceTree ก็มีฟีเจอร์ครบครันครับ ไม่ว่าจะเป็น commit, checkout, reset, fetch, pull, push เป็นต้น



และเมื่อเราลองแก้ไขไฟล์ อย่างเช่นผมเพิ่มเนื้อหาที่ไฟล์ about.html แล้วกลับมาดูที่ SourceTree จะเห็นว่า มีไฟล์อะไรที่ถูกแก้ไขและแก้ไขบรรทัดไหนไป ได้ทันที โดยปกติถ้าเป็น Command Line ก็เช็คด้วย `git diff HEAD`



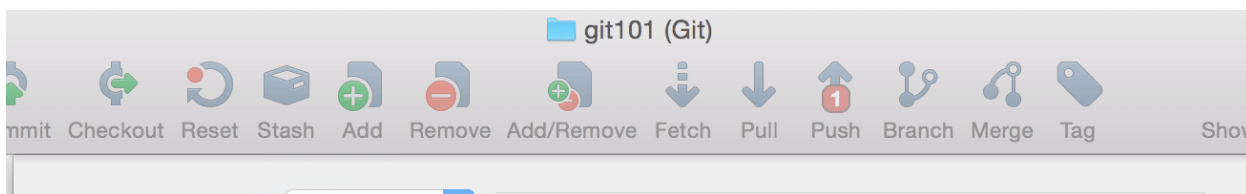
ทำการเลือกไฟล์จาก unstaged ไปที่ staged ไฟล์ จากนั้นกด Commit

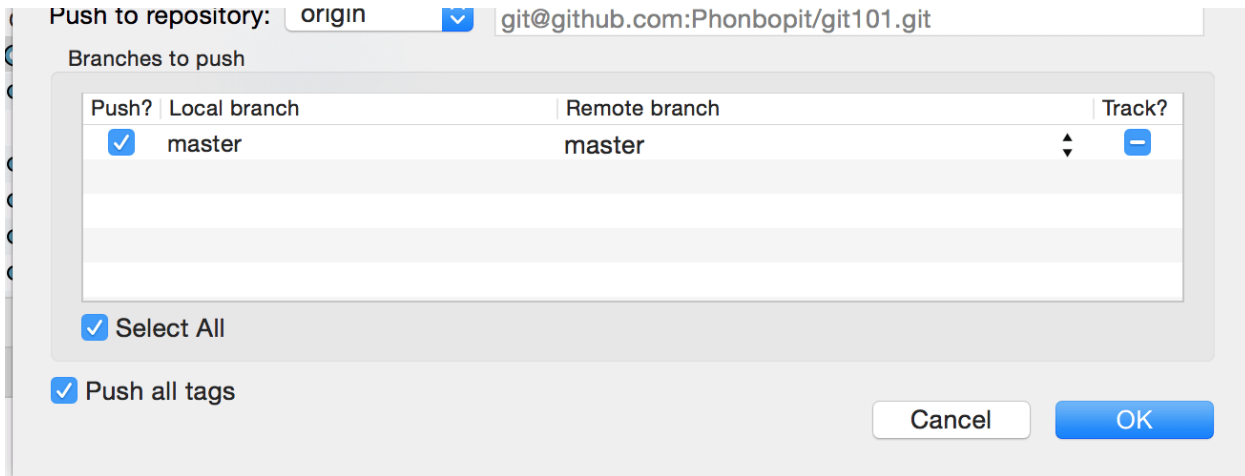


เทียบเท่ากับ

```
git add about.html  
git commit -m "Update content for about page"
```

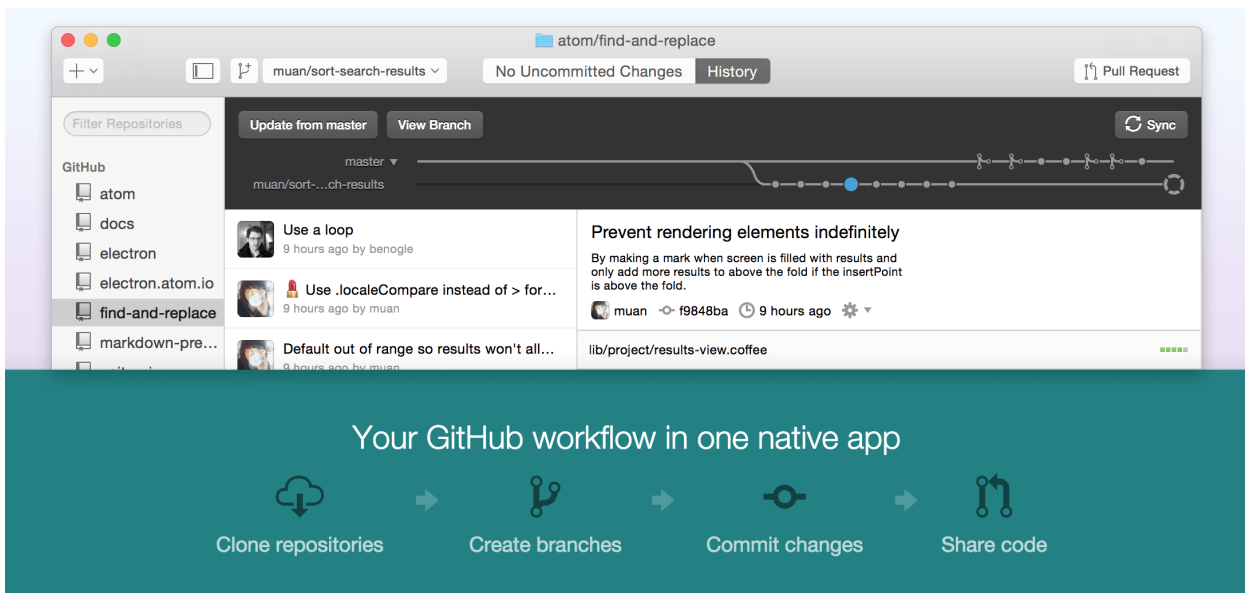
สุดท้ายก็สั่ง `git push` โดยกดที่ปุ่ม **Push** ด้านบน เป็นอันเรียบร้อย



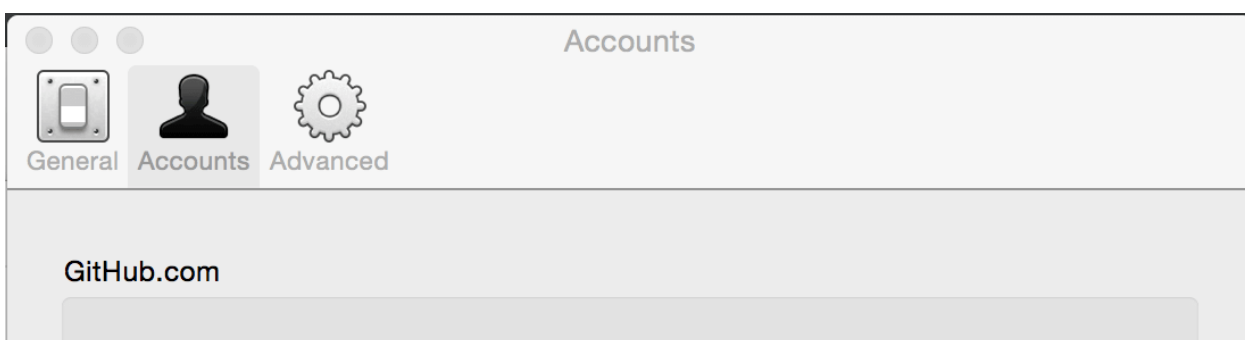


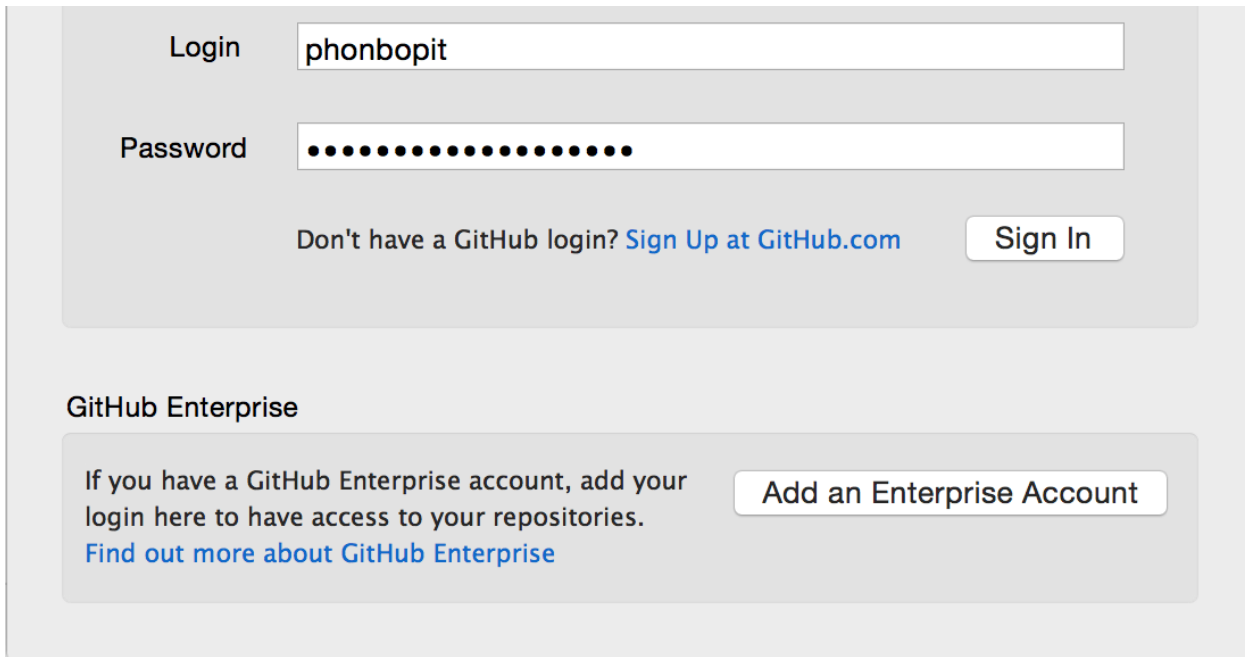
นี่ก็เป็นการใช้ SourceTree เบื้องต้นเท่านั้น รายละเอียดปลีกย่อยอื่นๆ ก็ต้องไปศึกษา ไปลองเรียนรู้กันเอาเองนะครับ

Github Desktop



เมื่อติดตั้ง Github Desktop แล้ว สิ่งแรกที่ต้องทำคือทำการ Login ด้วยบัญชี Github ของเรานะครับ





Login

Password

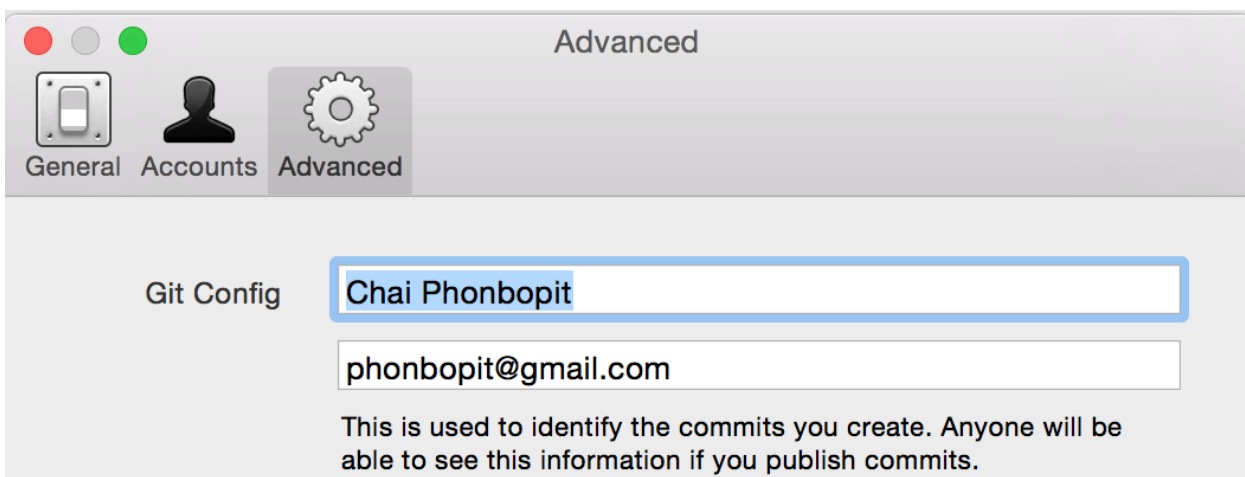
Don't have a GitHub login? [Sign Up at GitHub.com](#)

GitHub Enterprise

If you have a GitHub Enterprise account, add your login here to have access to your repositories.
[Find out more about GitHub Enterprise](#)

จะเห็นว่าในส่วนแท็บ **Advanced** เราสามารถที่จะ config ชื่อและอีเมลได้ด้วย เหมือนกันกับทำ command line ด้วย

```
git config --global user.name "YOURNAME"  
git config --global user.email "your@email.com"
```



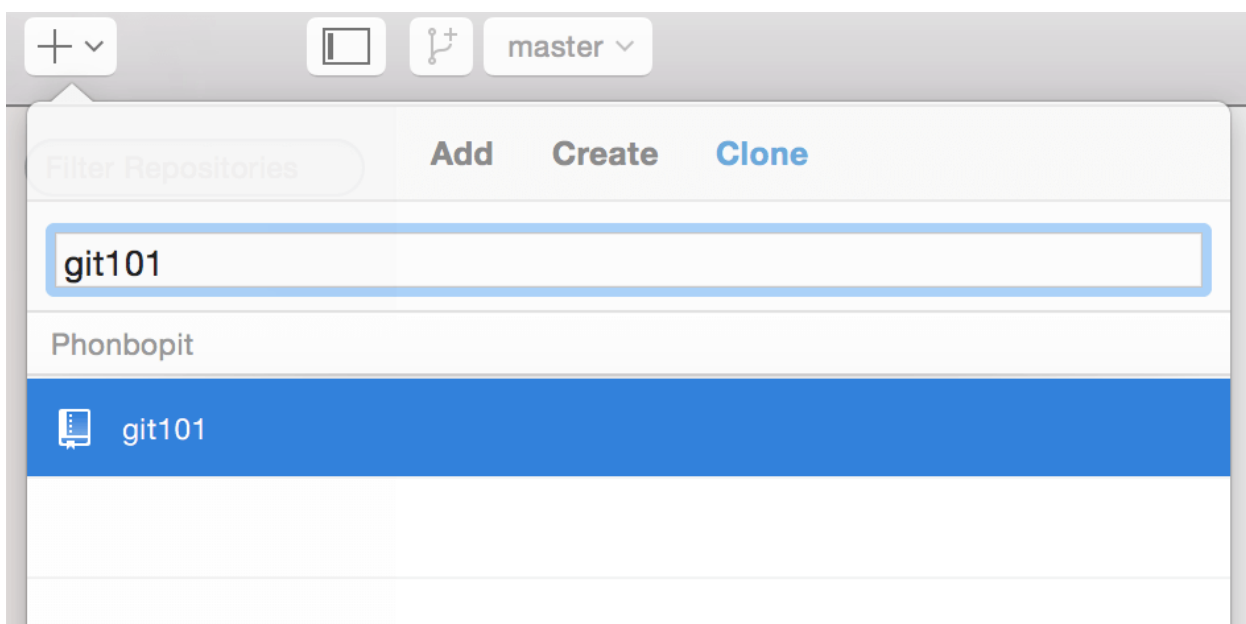
Advanced

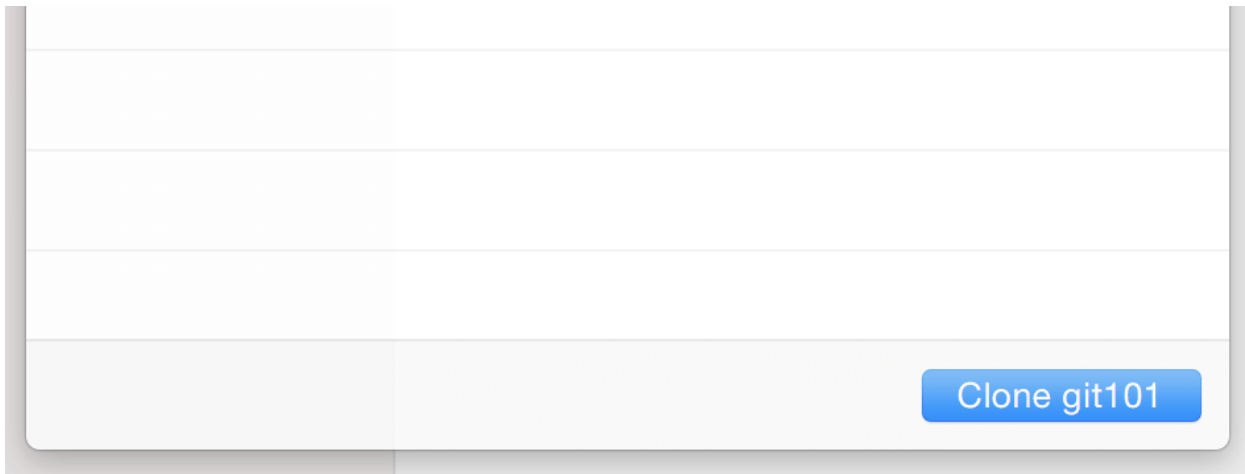
General Accounts **Advanced**

Git Config

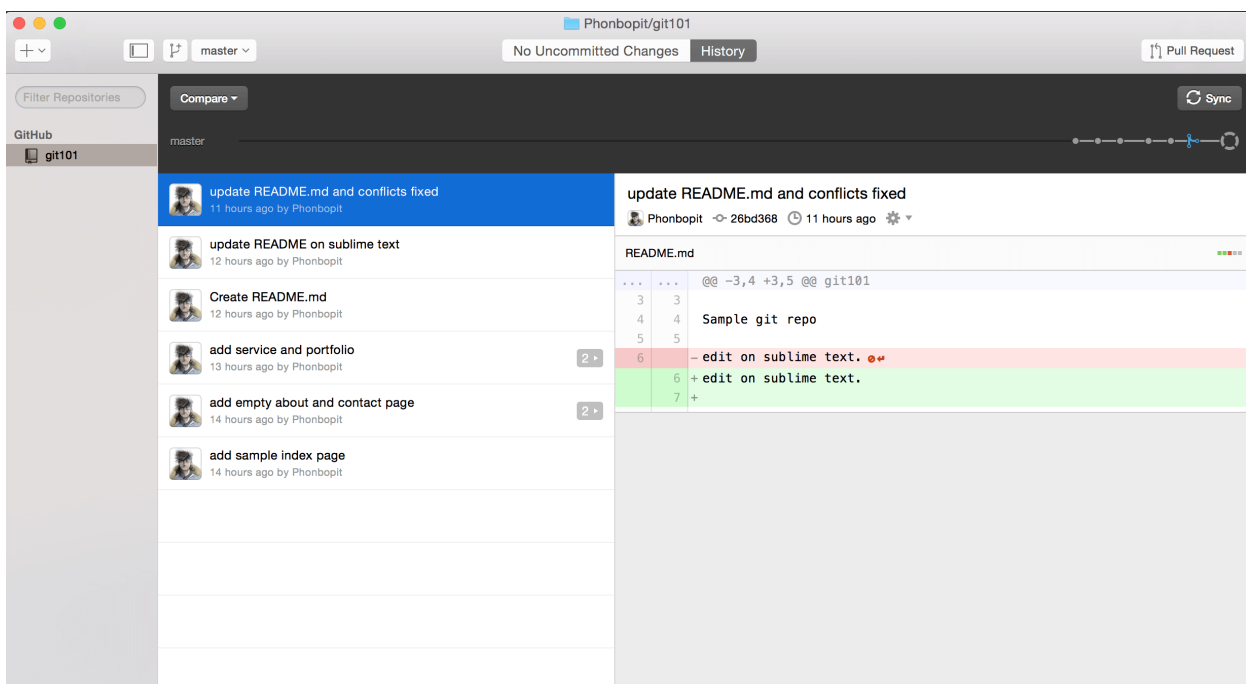
This is used to identify the commits you create. Anyone will be able to see this information if you publish commits.

ต่อมาทำการ clone repo จาก Github สำหรับ Github Desktop เราจะ clone ได้เฉพาะ repo ที่เราเป็นเจ้าของอยู่ครับ วิธีการจะ clone repo คนอื่น ก็ต้องเข้าหน้าเว็บไซต์ของ Github แล้วเลือก **Clone in Desktop** เอาครับ





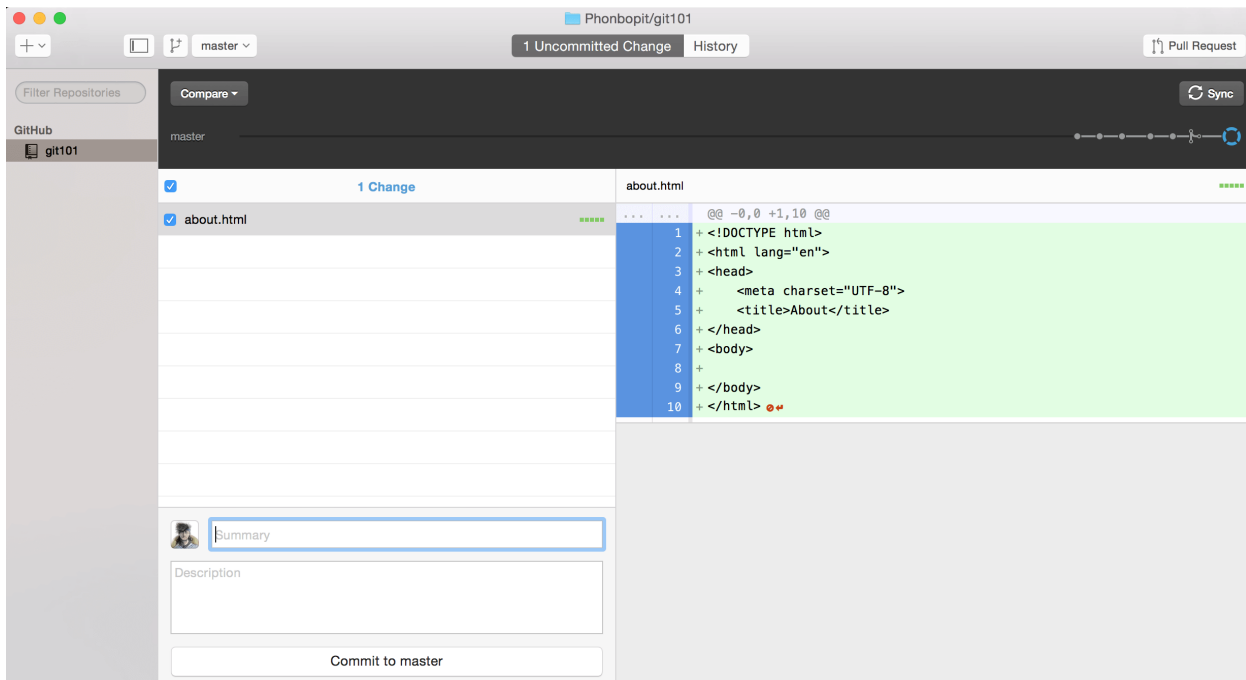
เหมือนกับ SourceTree เลย เราสามารถที่จะดู History ต่างๆที่เกิดขึ้นกับโปรเจ็คของเราได้ ไม่ว่าจะเป็นไฟล์ไหนถูกแก้ไขบ้าง บรรทัดไหนบ้าง



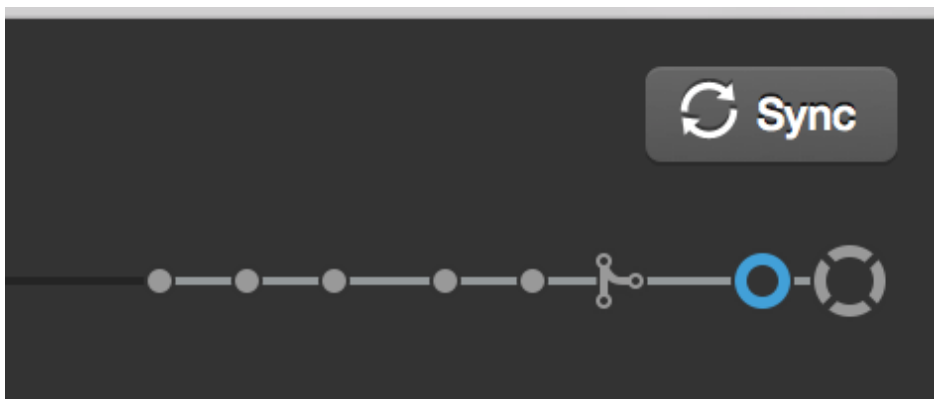
เมื่อแก้ไขงานไฟล์ใดๆไป Github Desktop ก็จะมีบอกว่า uncommit มีอะไรบ้าง

และต้องการ add และ commit ไฟล์ใดบ้าง

การ commit ก็ทำได้ง่ายๆ เพียงแค่เลือกมาที่ **Uncommitted Changes** แล้วก็เลือกไฟล์ที่จะ add จากนั้นใส่ commit message ที่ต้องการ



เมื่อ commit แล้วก็กดปุ่ม Sync ขวามือ มันก็จะทำการ Push ให้เอง



สำหรับ Github Desktop ปุ่ม Sync จะทำการ *fetch, pull, push* ให้เราเอง

การใช้งาน Github Desktop เบื้องต้นก็มีเท่านี้ครับ อยากูรู้รายละเอียดให้มากขึ้นก็ต้องไปหัดเล่นกันเอาเองละครับ

Step 6 : Workshop with Github

สำหรับส่วนสุดท้าย จะเป็น Workshop ให้ลองใช้งานครับ โดยผมจะทำการสร้าง repository ขึ้นมา 1 ตัว โดยผมทำการสร้างไว้ที่

Source Code

หลังจากนั้นก็ให้ผู้ที่เข้ามาอ่าน ได้ลองทำจากสิ่งต่างๆที่ได้เรียนรู้จากบทความนี้ครับ โดยมีเงื่อนไขดังนี้

1. ให้ทำการ clone repository นี้ไปไว้ที่เครื่อง local ของตัวเอง (ต้องทำการ Fork โปรเจ็คของผม ไปเป็นของตัวเองก่อน)
2. จากนั้นก่อนทำการแก้ไขไฟล์ใดๆ ให้แตก branch ออกมาก่อน ห้ามทำใน master เด็ดขาด
3. การสร้างโฟลเดอร์ (จะเรียกว่า subfolder) จะมีเงื่อนไขโดยตั้งชื่อตาม username ของ Github
4. ภายในโฟลเดอร์ของตัวเอง จะเป็นสิ่งที่แต่ละคนจะแก้ไขครับ สามารถใส่อะไรก็ได้ อยากแนะนำตัว สไลด์ snippets มาหรืออะไรก็ได้ แต่ขอแค่ ต้องมีไฟล์ README.md มาด้วย
5. เมื่อแก้ไขเรียบร้อยแล้ว ให้ทำการกด Pull Request เพื่อทำการส่ง Pull Request มาที่ตัว original repository ของผม

ใครทำไม่ตรงตามเงื่อนไขจะไม่ได้รับการกด Accept Pull Request (PR) นะครับ สำหรับตัวอย่าง [Example โฟลเดอร์ของผม สามารถดูได้ที่นี้](#)

Hint :

- [Fork A Repo](#)
- [Using Pull Requests](#)

เป็นอันเรียบร้อย ง่ายมั๊ยครับ? หากใครมีอะไรสงสัย สอบถามได้ที่นี้ หรือว่าที่ [Issue on hello_github](#) ก็ได้ครับ

สรุป

หวังว่าบทความนี้จะเปิดมุมมองของคนที่ยังไม่ได้ใช้งาน Git หรือว่ายังช่างใจว่าจะใช้ดีหรือไม่ ให้มาสนใจและใช้งาน Git มากขึ้นนะครับ โดยบทความนี้พูดถึงการใช้งาน Git ผ่าน Command Line เพื่อให้เราได้ที่มาที่ไป และใช้คำสั่งได้ถูกต้อง เมื่อไปใช้พวก GUI จะทำให้เราเข้าใจว่ามันทำงานยังไง ซึ่งทั้ง SourceTree และ Github Desktop ก็ใช้งานเบื้องต้นได้เหมือนกัน แต่ SourceTree จะมีฟังก์ชันที่ครอบคลุมดีกว่า ทีเหลือก็ให้ผู้อ่านไปลองดูกันเองครับว่าชอบแบบไหน จะ command line. GUI ก็แล้วแต่สะดวก ขอแค่ให้เข้าใจว่าจะทำอะไรกับมันเท่านั้น

พอ

หากส่วนไหนผิดพลาด หรือมีข้อเสนอแนะ แนะนำติชมมาได้เลยครับ ขอบคุณครับ

Additional Resoures

- [Official Git](#)
- [Ry's Git Tutorial](#)
- [GIT IMMERSION](#)
- [GitTower : Learn Version Control with Git](#)
- [Atlassian Git Tutorials](#)
- [Github Guides](#)
- [Git Simple Guide](#)
- [Pro Git](#)

- [Git Tutorial: A Comprehensive Guide](#)

TAGS :

- [#Git](#)
- [#Github](#)
- [#Git คืออะไร](#)
- [#Github คือ](#)
- [#Version Control](#)

[Chai Phonbopit](#) : Developer แห่งหนึ่ง • ผู้ชายธรรมดาๆ ที่ชื่นชอบ Node.js, JavaScript และ Open Source มีงานอดิเรกเป็น Acoustic Guitar และ Football