

الفصل الثالث

بعد أن أكملنا التعامل مع المراحل الثلاثة في Git , الان لنفترض واجهنا واحد من السيناريوهات التالية ماذا سوف نفعل :

السيناريو الأول – لدي مشروع سواء كتابة شفرة أو وثيقة أو تصميم صورة , وعملت بعض التعديلات عليها ولم أنقلها من Working Directory الى Staging Area , وفي اليوم التالي أردت التراجع عن التعديلات التي عملتها بالأمس !

السيناريو الثاني – نفس السيناريو الأول لكنني نقلتها الى Staging Area و اردت التراجع عن التعديلات .

السيناريو الثالث – نفس السيناريو الأول لكنني نقلتها الى Staging Area ثم عملت لها . Commit

هذه السيناريوهات سوف تواجهها كثيراً عند العمل على Git وربما حدثت معك حتى عند عدم استخدامك ل Git وربما حللت المشكلة بجهد ووقت . لنأخذ كل سيناريو ونضع الحلول له .

السيناريو الأول – واحد من اعمال Freelancer اليوم هي الطباعة , بمعنى تعطى لك مجموعة من الوثائق ويطلب منك طباعتها بملف وورد مقابل مبلغ من المال وحسب عدد الكلمات التي سوف تطبعها .

لنفترض طلب منك طباعة مئة وثيقة , وبعد يومين طبعت ما يقارب خمسين وثيقة حسب التسلسل , لكن صاحب العمل طلب منك حذف عشر وثائق (من صفحة 40 الى صفحة 50) واستبدالها بوثائق جديدة , وعملت ذلك لكن في اليوم التالي رجع عليك صاحب العمل وطلب إعادة طباعة الوثائق التي مسحتها ! في هذه الحالة وبدون Git فأنت سوف ترجع تطبعها من جديد اليس كذلك ؟

في مثل هذه الحالات من الوارد ان تحدث معك وليس بالطباعة فقط , وانما في تطوير مشروع برمجي أو تصميم صوري . ولكي تكون لديك مرونة بالعمل وتخطط لكل السيناريوهات التي ممكن ان تواجهها فعليك استخدام Git .

هذا السيناريو يفترض انك تعمل على Git ولكنك أبقيت التعديلات التي أجريتها في مرحلة Working بدون ما تنقلها الى Staging .

الان نطبقه بصورة عملية , اذهب الى مجلد المستودع على مشروعك وأفتح الملف work.txt , وأمسخ ما بداخله , ثم أفتح Git Bash وأدخل على مسار المجلد كما في الصورة أدناه :

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 ~
$ cd d:

lenovo@Mohammed MINGW64 /d
$ cd Projects/

lenovo@Mohammed MINGW64 /d/Projects
$ ls
First_Repo/

lenovo@Mohammed MINGW64 /d/Projects
$ cd First_Repo/

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ ls
README.md work.txt

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ |
```

الان لنرى حالة المشروع من خلال الأمر git status

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ pwd
/d/Projects/First_Repo

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   work.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   work.txt

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ |
```

في مثل هذا فإن Git يعطيك مرونة في أسترجاع (Rrecove) الملف قبل اخر تعديل عليه , من خلال الامر **git checkout filename**

```
MINGW64:/d/Projects/First_Repo
tenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git checkout work.txt
Updated 1 path from the index

tenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   work.txt

tenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$
```

الان لو فتحت الملف work.txt سوف تجده أستعاد المعلومات التي حذفها .

السيناريو الثاني: هذا السيناريو نفس الأول ولكن ماذا لو أننا نقلنا التعديلات الى مرحلة Staging ؟

لنمسح ما بداخل الملف work.txt ونحفظه ثم نذهب الى ال Git Bash نستعرض حالة مشروعنا :

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   work.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   work.txt

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ |
```

ثم ننقله الى مرحلة Staging من خلال الامر **git add**

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git add work.txt

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   work.txt

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ |
```

كما تلاحظ التعديلات صارت في مرحلة Staging , في مثل هذا الوضع لو أردت استعادة محتويات الملف work.txt وهو في هذه المرحلة ! هنا Git يعطيك الأمر **git reset HEAD filename** سوف يرجعه الى مرحلة Working Directory ثم نطبق الامر السابق عليه لاسترجاعه قبل التعديلات الأخيرة (مسح المحتويات)

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git reset HEAD work.txt
Unstaged changes after reset:
M   work.txt

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   work.txt

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

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git checkout work.txt
Updated 1 path from the index

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$
```

الان لو فتحت الملف work.txt ستجده استعاد المعلومات التي حذفنا فيها سابقاً.

السيناريو الثالث : ماذا لو كان التعديلات في مرحلة Git Directory ؟

لنجرب نحذف ما بداخل الملف Work.txt وننقله الى مرحلة Staging ثم نعمل له
. Commit

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git add work.txt

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git commit -m "I had removed all text in work file!"
[master 0328ccb] I had removed all text in work file!
1 file changed, 1 deletion(-)

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$
```

في مثل هذه الحالة سوف نستخدم الأمر git log لنشاهد التعديلات التي صارت :

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git log --oneline --decorate
0328ccb (HEAD -> master) I had removed all text in work file!
65b67c5 (origin/master, origin/HEAD) I had added a new text to the working file!
ad9581d This first change in my project!
a56e190 Initial commit

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$
```

لاحظ هذا ال Commit الذي حذفنا فيه جميع محتويات الملف work.txt , نأخذ المعرف الخاص به (أعلاه هو 0328ccb) ثم نستخدم الامر git revert كما في الصورة أدناه حتى نرجعه الى مرحلة Staging :

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git log --oneline --decorate
0328ccb (HEAD -> master) I had removed all text in work file!
65b67c5 (origin/master, origin/HEAD) I had added a new text to the working file!
ad9581d This first change in my project!
a56e190 Initial commit

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git revert 0328ccb --no-edit
[master 4c06dc9] Revert "I had removed all text in work file!"
Date: Mon Feb 3 01:00:48 2020 +0300
1 file changed, 1 insertion(+)

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)

nothing to commit, working tree clean

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$
```

طيب الان نستخدم الامر git log لنرى ماذا حصل :

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git log --oneline --decorate
4c06dc9 (HEAD -> master) Revert "I had removed all text in work file!"
0328ccb I had removed all text in work file!
65b67c5 (origin/master, origin/HEAD) I had added a new text to the working file!
ad9581d This first change in my project!
a56e190 Initial commit

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ |
```

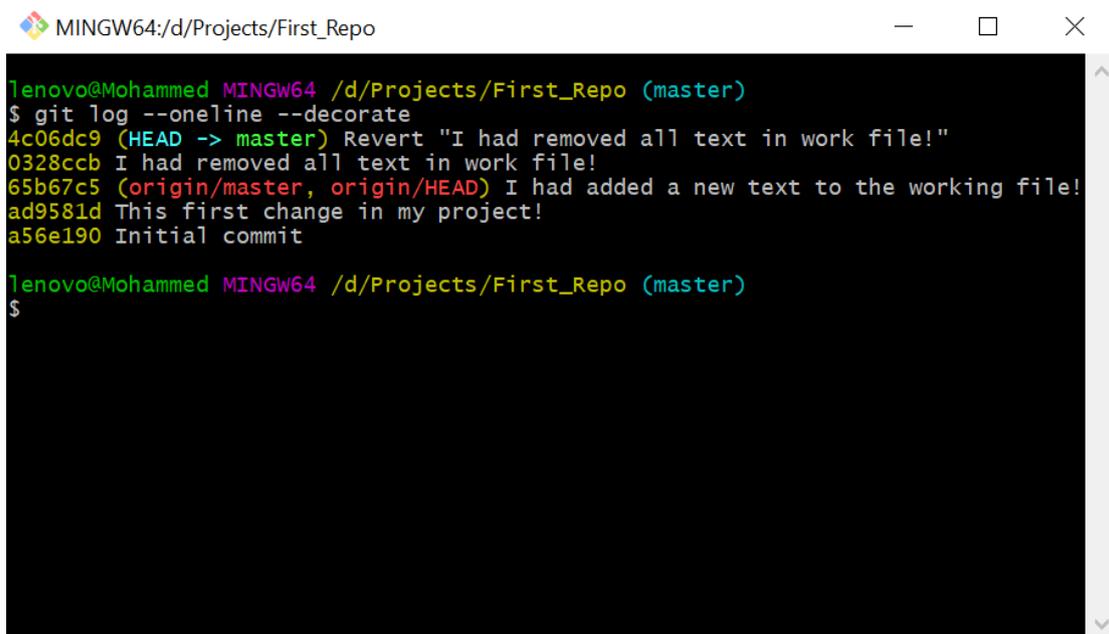
الان افتح الملف وسترى !

عمل إشارة (Tag) للتعديلات :

على بسيل الافتراض انك عملت على مشروع برمجي , وتريد ان تضع إشارة (Tag) لكل تعديل تجريه حتى تستفاد منه في إصدارات التطبيق التي سوف تكملها .

ال Git يساعدك على إعطاء أشاره (Tag) لكل تعديلات تجريها على المشروع .

من خلال الامر git tag نستطيع عمل ذلك , لكن ! نحتاج الى المعرف (Commit id) حتى نسند تاغ (اصدار) للتعديلات, وفي هذه الحالة سوف نستخدم الأمر git log لاستخراج المعرف .



```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git log --oneline --decorate
4c06dc9 (HEAD -> master) Revert "I had removed all text in work file!"
0328ccb I had removed all text in work file!
65b67c5 (origin/master, origin/HEAD) I had added a new text to the working file!
ad9581d This first change in my project!
a56e190 Initial commit

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$
```

مثلا ترى في الشاشة أعلاه فانه لكل (commit) أعطى معرف (Id) باللون الأصفر , سوف أختار مثلا المعرف (ad9581d) ونص رسالته (!This first change in my project) سوف أعتبره الإصدار رقم (v1.0) . ولتنفيذ أمر الإشارة سوف يكون بهذا الشكل :

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git log --oneline --decorate
4c06dc9 (HEAD -> master) Revert "I had removed all text in work file!"
0328ccb I had removed all text in work file!
65b67c5 (origin/master, origin/HEAD) I had added a new text to the working file!
ad9581d This first change in my project!
a56e190 Initial commit

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git tag -a v1.0 ad9581d -m "v1.0"

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$
```

وحتى نقدر نشاهد الأشارة فأننا سوف نستخدم الامر `git log` :

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git log --oneline --decorate
4c06dc9 (HEAD -> master) Revert "I had removed all text in work file!"
0328ccb I had removed all text in work file!
65b67c5 (origin/master, origin/HEAD) I had added a new text to the working file!
ad9581d (tag: v1.0) This first change in my project!
a56e190 Initial commit

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$
```

الان بإمكانك رفع التعديلات على ال Github والتي من ضمنها الاشارة من خلال الامر `git push` :

```
MINGW64:/d/Projects/First_Repo
lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ git push origin --tags
Enter passphrase for key '/c/Users/lenovo/.ssh/id_rsa':
Everything up-to-date

lenovo@Mohammed MINGW64 /d/Projects/First_Repo (master)
$ |
```

الآن أذهب الى المستودع (First_Repo) على Github وشاهد .

mohammed-alsaady / First_Repo

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

This for testing Edit

Manage topics

5 commits 1 branch 0 packages **1 release** 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

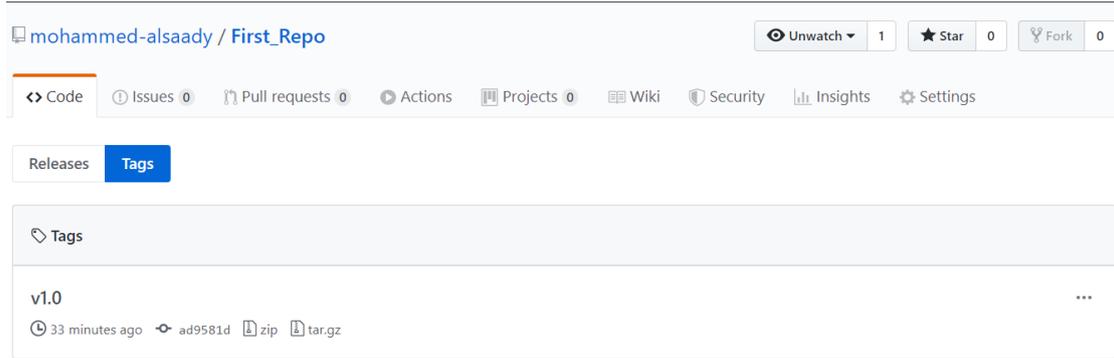
Commit	Message	Time
4c06dc9	Revert "I had removed all text in work file!"	15 hours ago
Initial commit	Initial commit	5 days ago

README.md

First_Repo

This for testing

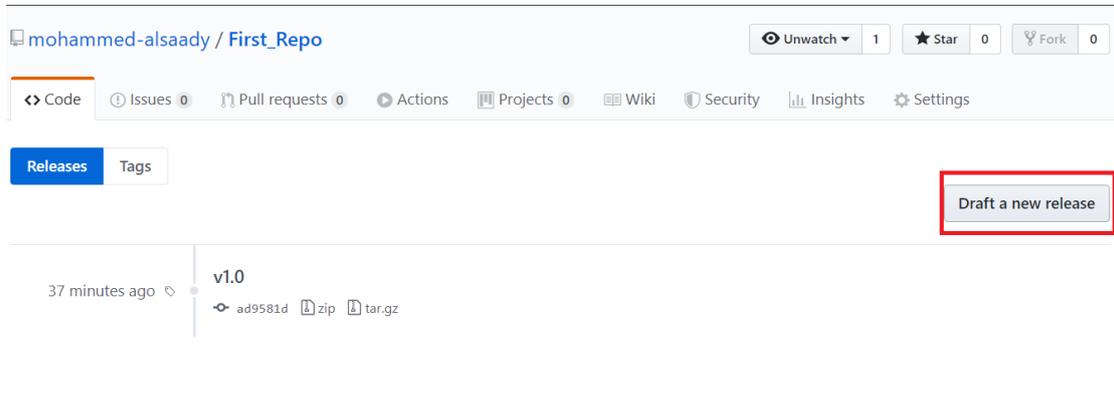
ومن ثم تنقر على الرابط release , بعدها نختار tag حتى يعرض لنا الأشارات التي عملناها لكل Commit .



هنا السؤال المطروح , ما الفرق بين Tag و Release ؟ الفرق بسيط جدا , لنفترض أردت تطوير تطبيق موبايل , وهذا التطبيق أكملت التصميم الأول له فهنا تقدر تعطيه (Tag) ثم أكملت قاعدة بياناته وهنا أيضا تقدر تعطيه كذلك (Tag) وبعدها أكملت الشفرات وكذلك تسند له (Tag) , بعد التاغ الثالث صار التطبيق جاهز وبإمكانك تحزيمه (Package) تحويله من شفرات الى ملف تنصيبي (وبالتالي فأنتك سوف تعطيه إصدار مثلا (الإصدار رقم 1.0) وفي ال Github الأصدار يعتمد على tag أي بمعنى انك يجب ان تعطي تاغ ثم اصدار .

كيف نعمل إصدار (Release) :

نفتح ال Github ونذهب الى المستودع ثم نختار Release وبعدها ننقر على `Draft a new release` .



سوف تظهر لنا الصفحة التالية ومنها نختار رقم tag و (Branch) الرئيسي , وعنوان الإصدار ووصف للمشروع . كذلك تقدر تضيف ملفات Binary اذا موجودة , وان كان مشروعك ليس جاهز تماما فتقدر تأسر على (pre-release) ثم ننقر على Publish Release :

v1.0 @ Target: master

Excellent! This tag will be created from the target when you publish this release.

version 1.0.0

Write Preview

this test release .

Attach files by dragging & dropping, selecting or pasting them.

Attach binaries by dropping them here or selecting them.

This is a pre-release
We'll point out that this release is identified as non-production ready.

Publish release Save draft

Tagging suggestions

It's common practice to prefix your version names with the letter v. Some good tag names might be v1.0 or v2.3.4.

If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be v0.2-alpha or v5.9-beta.3.

Semantic versioning

If you're new to releasing software, we highly recommend reading about [semantic versioning](#).

بعدها سيكون الأصدار جاهز:

Edit release Delete

Pre-release

v1.0

ad9581d

Compare

version 1.0.0

mohammed-alsaady released this now · 3 commits to master since this release

this test release .

Assets 2

Source code (zip)

Source code (tar.gz)