Git && GitHub

المحاضرة الثانية

م. محمد السلوم

التفريعات Branches في Git

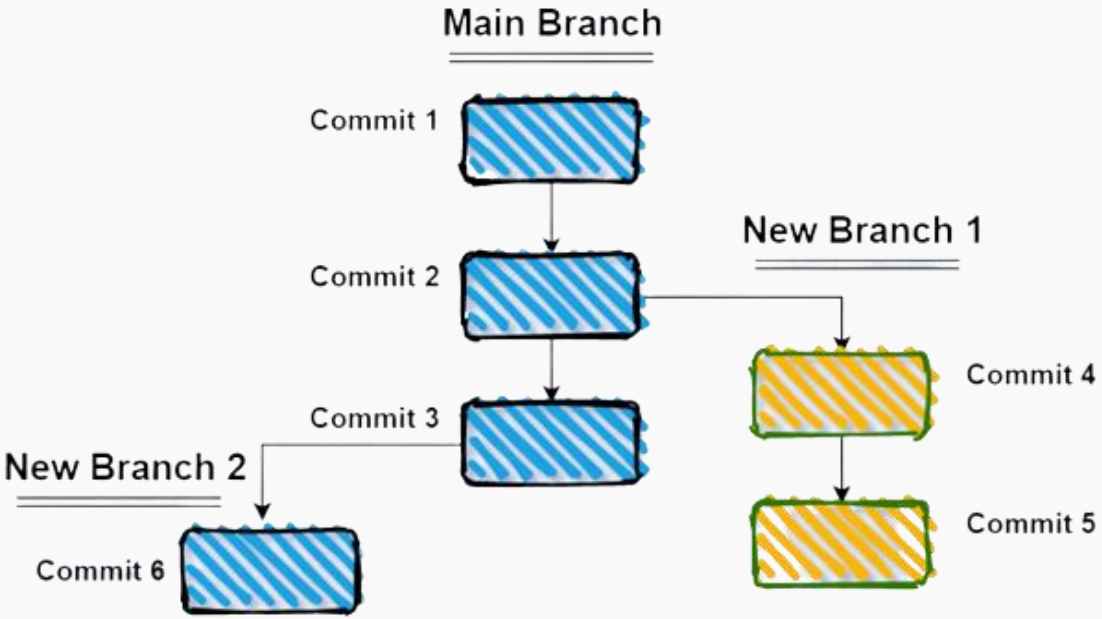
ما هي التفريعات Branches في Git	دمج الفروع في Git
لماذا نستخدم التفريعات ؟	حذف الفروع في Git
إنشاء فرع جديد في Git	Merge conflicts
الانتقال من فرع الى آخر في Git	Conflict Resolve
الفروع الطارئة	تثبيت أداة gitk

01

ما هي التفريعات Branches في Git

في العمل ضمن الفرق التقنية، يستخدم نظام Git لإدارة التفريعات (Branches) بحيث يمكن لكل عضو أو مجموعة من الأعضاء تطوير جزء معين من المشروع بشكل مستقل دون التأثير على الآخرين يتيح هذا لكل مطور العمل على نسخة منفصلة تقريبًا من المشروع، مما يمنحه المرونة والوقت الكافي للتطوير دون التأثير على الفرع الرئيسي. عند الانتهاء من العمل في الفرع الجديد، يتم دمجه مع المشروع الرئيسي. يتيح Git إمكانية التبديل بين الفروع بسهولة والعمل على مهام مختلفة دون تداخل، مما يجعل التفريعات وسيلة فعالة وسريعة لإجراء التعديلات في المشروع.

01 ما هي التفريعات Branches في Git



02 لماذا نستخدم التفريعات ؟

العمل المتوازي: يمكنك العمل على ميزات جديدة أو إصلاح الأخطاء في فرع خاص بك دون التأثير على الشفرة الأساسية في الفرع الرئيسي (عادة يسمى master أو main).

التجريب: يمكن إنشاء فرع لتجربة فكرة أو تنفيذ مهمة محددة، وإذا كانت التغييرات ناجحة، يمكن دمج الفرع مع الفرع الرئيسي وإذا لم تنجح الفكرة، يمكن حذف الفرع بسهولة.

إدارة الإصدارات: عند العمل في فرق، كل مطور يمكن أن يعمل على فرعه الخاص به عند الانتهاء، يمكن دمج جميع الفروع في الفرع الرئيسي.

03 إنشاء فرع جديد في Git

لنضف الآن بعض المزايا الجديدة في الصفحة التي أنشأناها في الفصل السابق، و كوننا نرغب أن نقوم بهذا في مستودعنا المحلي دون التأثير على المشروع الرئيسي أو تعطيله سنقوم **بإنشاء فرع جديد** باستخدام الأمر :

```
git branch BranchName
```

الان أنشأنا فرعاً جديداً باسم **images** ويمكننا التحقق من وجوده من خلال تنفيذ الامر التالي دون اسناد اسم الفرع :

```
git branch
```

03 إنشاء فرع جديد في Git

```
MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/te
ster)
$ git branch
  images
* master
```

وكما نرى فيوجد الآن فرع جديد باسم **images** و فرع آخر اسمه **master** بجانبه علامة نجمة *

تعني أننا الآن في هذا الفرع.

في الفرع **master** ربما تتساءل الآن من أين جاء الفرع **master** و أنا أتعلم التفريعات للتو ؟

والجواب أن **git** تنشئ الفرع **master** بشكل آلي في كل مستودع جديد.

ملاحظة: يجب الانتباه إلى الفرع الذي تتواجد فيه لحظة إنشاء فرع جديد، لأن الفرع الجديد

سيكون في لحظة إنشائه نسخة مطابقة تماماً للفرع الذي تم إنشاؤه منه.

04 الانتقال من فرع الى آخر في Git

يستخدم الأمر **checkout** للانتقال من الفرع الحالي إلى فرع آخر نمرّر اسمه مع الأمر، و لتجربة ذلك يمكننا الانتقال إلى الفرع الجديد الذي أنشأناه توا كما يلي:

```
git checkout branchName
```

```
MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/tes
ster)
$ git checkout images
Switched to branch 'images'
M       index.html
```

و هكذا نكون قد نقلنا مساحة العمل الحالية من الفرع الرئيسي إلى الفرع الجديد، و الآن لنفتح الملف لنجري بعض التعديلات عليه.

04 الانتقال من فرع الى آخر في Git

في هذا المثال سنقوم بإضافة صورة الى مجلد العمل ثمّ نضمنها ضمن صفحة index.html بإضافة السطر التالي:

```
<body>
    <h1>Welcome in our Session </h1>

    <img src="iusr.png" alt="iusr">
</body>
```

ملاحظة: يمكننا استخدام الخيار b- مع الأمر checkout لنطلب من git أن ينتقل إلى الفرع الجديد في حال كان موجوداً، وإلّا أن ينشئه ثم ينتقل إليه مباشرة.

04 الانتقال من فرع الى آخر في Git

وبهذا نكون قد أجرينا تغييرات على ملف index.html و أضفنا الملف الجديد إلى مجلد العمل، و إذا نفذنا **git status** سنجد ما يلي:

```
MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/test4 (im
ages)
$ git status
On branch images
Changes not staged for commit:
  (use "git add <file>..." to update what will be committ
ed)
  (use "git restore <file>..." to discard changes in work
ing directory)
        modified:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be com
mitted)
        iusr.png

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

وكما نلاحظ فالملف index.html يحوي بعض التعديلات (modified) غير أنّه لم يُدرج بعد Not staged، و الصورة فحالته Untracked كونه يضاف للمرة الأولى و لم يودع من قبل.

04 الانتقال من فرع الى آخر في Git

و للقيام بإيداع كلا التغييرين سنحتاج إلى إضافة كلا الملفين إلى بيئة الإدراج ننفذ الامر التالي:

```
git add --all
```

```
MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/test4 (im
ages)
$ git add --all

MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/test4 (images)
$ git status
On branch images
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html
        new file:   iusr.png
```

04 الانتقال من فرع الى آخر في Git

والان يجب إيداع هذه التعديلات يمكننا استخدام الامر **git commit** الذي تعلمناه سابقا مع ترك رسالة مناسبة توضح الهدف من عملية الإيداع:

```
git commit -m  " commit text "
```

```
MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/test4 (images)
$ git commit -m "Added Image to Index.html"
[images 3bb1890] Added Image to Index.html
 2 files changed, 2 insertions(+)
 create mode 100644 iusr.png
```

04 الانتقال من فرع الى آخر في Git

لإعادة تسمية الفرع باسم جديد نستخدم الامر التالي:

```
git branch -m newBranchName
```

```
MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/test4 (imag
$ git branch -m images-file

MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/test4 (imag
$ git branch
* images-file
  master
```

04 الانتقال من فرع الى آخر في Git

ما يهمنا الآن و نريد التركيز عليه هو وجود فرع branch جديد يختلف عن الفرع الرئيسي master و الاختلاف يكمن في عملية الإيداع الأخيرة، و لنرى الآن مدى سرعة وسهولة العمل مع الفروع المختلفة ومدى كفاءتها في العمل نحن الآن في الفرع **images-file**، وأضفنا صورة إلى ذلك الفرع، و إذا استعرضنا الملفات التي في المجلد الحالي سنجد الملفين **index.html** و **iusr.png**، بمعنى أنّنا نستطيع الآن رؤية الملف الجديـد **iusr.png**، وكذلك نستطيع رؤية التغييرات الحادثة في الشيفرة إذا فتحنا ملف **index.html**، وهو ما نتوقعه تماماً

04 الانتقال من فرع الى آخر في Git

والآن، لنرى ما يحدث عند تغيير الفرع إلى master :

```
git checkout master
```

لم تعد الصورة الجديدة جزءاً من هذا الفرع، و لو استعرضنا الملفات الموجودة في المجلد الحالي مرة أخرى لرأينا الآن أن الصورة الجديدة غير موجودة و كذلك الشيفرة التي أضفناها إلى **index.html** و هذا يرينا سهولة العمل مع الفروع وكيف أنها تسمح بالعمل على أشياء مختلفة في نفس الوقت.

05 الفروع الطارئة

لنتخيل الآن أننا لم ننه العمل على الفرع **images-file**، لكننا و لسبب هام نريد إصلاح خطأ طارئ في الفرع الرئيسي **master** دون المساس بذلك الفرع الرئيسي ولا فرع **images-file** وبما أننا لم ننه العمل عليه بعد، في تلك الحالة يمكننا ببساطة إنشاء فرع جديد للتعامل مع هذا الإصلاح الطارئ و يمكننا تسميته **Urgent-fix** اصطلاحاً

```
git checkout -b branchName
```

و نكون الآن قد أنشأنا فرعاً جديداً من الفرع الرئيسي و أجرينا تعديلات عليه، و نستطيع الآن إصلاح الخطأ دون التأثير على الفروع الأخرى ثم إيداع هذا الإصلاح في الفرع الجديد، و لكن كيف نجلب هذا الإصلاح إلى الفرع الرئيسي؟ الجواب بدمج الفرعين كما سوف نرى في الفقرات التالية.

06 دمج الفروع في Git

الان على فرض قمنا بإصلاح الخطأ في الفقرة السابقة واصبح جاهز لعملية الدمج. نقوم الان بعملية دمج الفرع **Urgent-fix** مع الفرع **master** باتباع الخطوتين التاليتين:

1. الانتقال للفرع الهدف **master** الذي نرغب في دمج الفرع الآخر معه وذلك من خلال التعليمة :

```
git checkout master
```

2. دمج الفرع **Urgent-fix** مع الفرع الهدف **master** من خلال التعليمة:

```
git merge Urgent-fix
```

في حال كون كل شيء على ما يرام سيتم الدمج وستظهر المساهمات التي قمنا بها في فرع الإصلاحات العاجلة ضمن الفرع الرئيسي.

07 حذف الفروع في Git

في حال لم نعد بحاجة إلى فرع ما لسبب معين فيمكننا حذفه من خلال التعليمة:

```
git branch -D branchName
```

حيث ان branchName يتم استبداله باسم الفرع الهدف والمطلوب حذفه ويجب مراعاة ان يكون حرف الوسيط D- بحالة الحرف الكبير كما هو موضح بالصورة:

```
MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/test4 (master)
$ git branch -D urgent-fix
Deleted branch urgent-fix (was 3bb1890).

MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/test4 (master)
$ git branch
  images-file
* master
```

07 حذف الفروع في Git

ولحذف الفرع بشكل آمن وذلك اذا تم دمج جميع التغييرات فيه بشكل سليم نستخدم التعليمة:

```
git branch -d branchName
```

ولحذف الفرع بشكل قوي وذلك بدون التحقق من دمج جميع التغييرات فيه بشكل سليم نستخدم التعليمة:

```
git branch -D branchName
```

Merge conflicts

08

في الفقرة قبل السابقة، تمت عملية دمج الفرعين بشكل سلس و دون مشاكل و ذلك نظراً لعدم وجود أكثر من شخص يعملون على نفس الملفات في الوقت نفسه، لكن أحياناً و عند العمل ضمن فريق، لا تكون الأمور بهذه البساطة، و لا يستطيع git أحياناً معرفة الطريقة المثلى لدمج مساهمات المساهمين عندما يتصادف وجود تعديلين أو أكثر في نفس الملف و في نفس الأسطر، عندها يحدث ما يسمى بتعارض الدمج Merge conflict و الذي يتطلب التدخل اليدوي من أحد المساهمين للقيام بعملية حلّ التعارض Conflict resolvent بحيث يمكن إتمام الدمج بسلاسة و لشرح هذه الحالة لا بدّ من اصطناع حالة تستوجب التعارض أولاً، ثم سنرى معاً كيف يحدث التعارض، و الأهم كيف يمكن حلّه!

Merge conflicts

08

حسناً، لأنشاء حالة التعارض قم بعمل فرع جديد ثم عدّل السطر رقم 5 مثلاً من ملف index.html في نفس الفرع الحالي، و قم بحفظ المساهمة، ثمّ انتقل إلى الفرع الجديد و قم بتعديل مختلف في نفس السطر و في نفس الملف السطر من الملف (index.html) ثم قم بحفظ المساهمة. الآن عندما تعود إلى الفرع الرئيسي و تحاول دمج الفرع الجديد سيظهر التعارض و سترى كلمة Conflict أمام الملفات المتعارضة كما تبيّنه الصورة التالية:

```
MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/test4 (master)
$ git merge tow
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the resu
```

Conflict Resolve

09

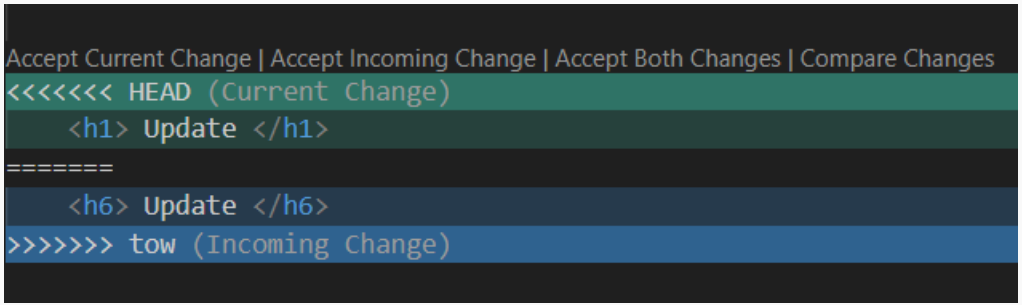
حسناً كما رأينا في الفقرة السابقة فقد حصل تعارض عندما حاولنا دمج الفرعين معاً نتيجة لحقيقة أننا قمنا بإجراء تعديلات مختلفة في نفس الأماكن من نفس الملفات و بالتالي لن يستطيع git تحديد طريقة لدمج المساهمات معاً بشكل آلي في الملفات المتأثرة بالتعارض، يمكننا من خلال الأمر **git status** معرفة الملفات التي تحوي تعارضات حيث سيظهرها git و بجانبها وصف **both modified** بمعنى أنّ الملف خضع لتعديل في كلا الفرعين، كما يظهر في الصورة أدناه.

```
MOHAMD-SALOUM@DESKTOP-K2UL453 MINGW64 ~/Desktop/test4 (master|MERGI
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   index.html

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

Conflict Resolve

09

لو قمنا الآن بفتح الملف المتأثر باستخدام المحرر النصي، سنجد أنّ git قام بوضع علامات تحدد مناطق التعارض ضمن الملف متوقعاً منا القيام بتعديلات يدوية على الملف لحل التعارضات، و من الجدير بالذكر أن بعض المحررات النصية المتقدمة تسهل عملية حل التعارضات البسيطة من خلال بعض الأدوات كما في حالة VS Code مثلاً كما يظهر في الصورة أدناه:



Conflict Resolve

09

يمكننا بوضوح رؤية أن git ترك التعديلين المتعارضين مع إشارة واضحة لاسم الفرع الذي جاء منه كل تعديل (head يعني الفرع الذي نحاول الدمج إليه master في مثالنا). والآن يمكننا إبقاء التعديل المطلوب و حذف التعديل غير المرغوب به ليعود الملف إلى حالة طبيعية، ثم نقوم بعد ذلك بعمل التعليمات **git add --all** للملفات التي تم تعديلها و حل التعارضات فيها، ثم التعليمة **git commit**

10 تثبيت أداة gitk

بهذا نكون قد غطّينا الأوامر الأساسية في git للتعامل مع المستودع محليّاً على الحاسب الخاص بنا)، و سنقوم في الفصل القادم بشرح كيفية التعامل مع مستودع بعيد مستعرضين منصة GitHub كمثال على منصات git البعيدة، مع الإشارة إلى أنّ غالبيّة ما سنغطّيه عند الحديث عن GitHub ينطبق على جميع المنصات الشهيرة مثل BitBucket أو Gitlab .

والحمد لله رب العالمين