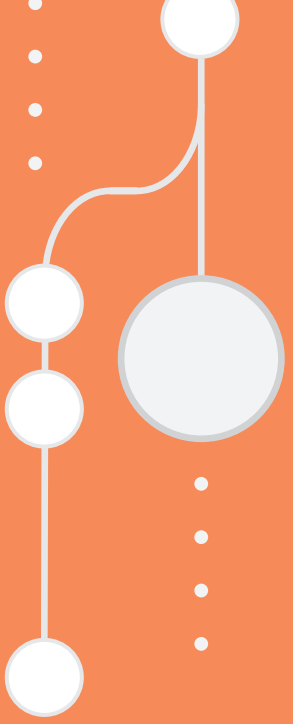
Git



01

مقدمة في مفهوم Git

تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل عديدة، من تحديث وحذف وإضافة.

فماذا لو أننا أردنا التوقف في مرحلة من تلك المراحل عن تحديث ملفات معينة؟

في هذا الفصل سنتعلم:

- تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل
- عديدة تمر ملفات المشروع أثناء العمل عليه وبنائه
- بمراحل عديدة تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل عديدة

١. نظرة على مفهوم Git

نظرة على مفهوم أنظمة التحكم بالنسخ Version Control

تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل عديدة، من تحديث وحذف وإضافة. فماذا لو أننا أردنا التراجع في مرحلة من تلك المراحل عن تحديث ملفات معينة؟ ووجدنا أن التغييرات التي أجريت على الملفات كانت كثيرة مما جعل عملية الرجوع إلى الخطوة المطلوبة صعب أو غير ممكن، ماذا لو تم حذف ملف معين ومن ثم اردنا التراجع عن ذلك؟ ماذا لو أراد أكثر من شخص العمل على ملفات المشروع في نفس الوقت، كيف سيتم ذلك؟ هل ستتم مشاركة الملفات عبر البريد الالكتروني؟ وكيف سيعلم أعضاء الفريق إن قام أحد الأعضاء بالتعديل على أحد الملفات؟. إن العمل على المشروع يصبح أكثر تعقيدا عندما يكون هناك أكثر من شخص يعمل عليه، وهنا يأتي دور Version Control والذي يسهل عملية إدارة ومتابعة المشاريع ومعرفة التغييرات التي أجريت عليها بحيث يمكنك التراجع عن أي تعديل والعودة لأي نسخة من المشروع بكل سهولة، بالإضافة إلى معرفة متى ومن قام بالتحديث والتغيير على الملف، وما سبب هذا التحديث ومعرفة شكل الملف قبل وبعد التحديث. ومن أهم وأشهر المنصات والبرامج التي تقدم للمستخدم هذه الخدمات هو git فيما يلي سنتعرف على كيفية التعامل مع git والاستفادة من خصائصه.

ما هو git

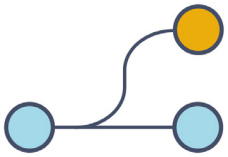
هو عبارة عن نظام يقوم بتتبع وتسجيل التغييرات التي أجريت على الملفات عبر أخذ نسخ "snapshot" يحددها المستخدم، فيقوم هذا النظام بحفظ التغييرات خطوة بخطوة مع التسلسل الزمني ويرفق وصفًا لكل خطوة من هذه التغييرات حتى يتمكن المستخدم من تتبعها والرجوع إليها بسهولة.

كما يسهل نظام git عمل الفرق، فهو يتيح لكل عضو من أعضاء الفريق أخذ نسخة من المشروع والعمل على تحسينها. حيث يقوم كل عضو بالعمل بشكل مستقل ومن ثم يقوم برفع النسخة المحسنة إلى الخادم والذي يقوم بدوره بدمج فروع المشروع branches، أي النسخ التي عمل عليها أعضاء الفريق

أهم العناصر التى سنتعامل معها في Git

Commit اللقطة

هي عبارة عن نقطة التخزين التي يتم فيها حفظ التغييرات و يمكن العودة لها في وقت لاحق



Branch الفرع

وهو عبارة عن فرع أو نسخة من المشروع يقوم فيها المبرمج بالعمل على جزء معين من ذلك المشروع ثم يتم دمج عمله مع الفرع الرئيسي في المشروع

Repository المستودع

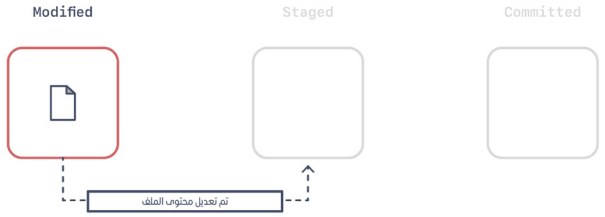
هو عبارة عن المجلد الذي يحتوي على المشروع وبداخله سجل بالتغييرات والتحديثات التي تمت على هذا المشروع

مراحل الملفات في git

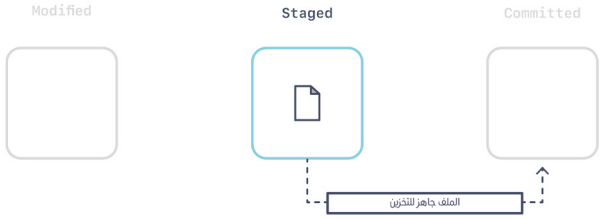
هناك ثلاث مراحل للملفات في git وهي

- modified : تعني أنك قمت بتعديل الملف ولم تقم بمراقبة هذا التعديل
- staged : تعني أنك قمت بمراقبة التغير ولكن لم تقم بحفظه في repository
- committed : تعني أنك قمت بحفظ التغير في المستودع repository

توضح الصور التالية المراحل الثلاث التي تمر بها الملفات، و يمكننا القول أنها دورة حياة الملفات في Git



عند القيام بأي تغيير في الملفات سواء كان حذف أو اضافة أو حتى تغيير بسيط في اسم ملف، فإن الملفات لن تتوافق مع نسخة الملفات المخزنة في Repository المشروع وتسمى هذه المرحلة Modified



في هذه المرحلة يتم تحديد الملفات المحدثة والمراد تخزينها وحفظها في Repository المشروع، وتعتبر هذه المرحلة المرحلة الأولى من مراحل تخزين الملفات في Git

في هذه المرحلة يتم تخزين وحفظ التعديلات فعليا في Repository المشروع. وهي المرحلة الثانية من مراحل تخزين الملفات في Git. وتصبح ملفات المشروع في هذه المرحلة في حالة توافق تام مع تلك المخزنة في Repository المشروع

02

مقدمة في أوامر Git

تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل عديدة، من تحديث وحذف وإضافة.
فماذا لو أننا أردنا التراجع في مرحلة من تلك المراحل عن تحديث ملفات معينة؟

في هذا الفصل ستتعلم :

- تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل
- عديدة تمر ملفات المشروع أثناء العمل عليه وبنائه
- بمراحل عديدة تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل عديدة

١. مقدمة في أوامر Git

هناك عدد من الاوامر الأساسية التي هناك عدد من الاوامر الأساسية التي هناك عدد من الاوامر الأساسية التي هناك عدد من الاوامر الأساسية التي

■ إنشاء مُستودع git init

كلمة init هي اختصار initialize ويقصد بها ربط وتجهيز Git و من خلالها يتم انشاء repository جديد عند استخدام الأمر git init سيصبح المجلد في حالة Working Directory أي أنه جاهز لاستقبال التعديلات

```
01.      git init
```

عند استدعاء هذا الأمر فإننا نحصل على ملف مخفي باسم git. يقوم هذا الملف بحفظ سجلات بتواريخ التحديثات والتغييرات التي أجريت على المشروع.

• مثال على إنشاءRepository

في المثال التالي سنقوم بإنشاء مجلد جديد لمشروعنا على سطح المكتب، ثم سنقوم بالانتقال إلى ذلك المجلد، ثم سنقوم بتعريف وإنشاء مستودع Repository لهذا المشروع. توضح الأسطر التالية كيفية القيام بذلك:

```
02.     mkdir first-app
03.     cd first-app
04.     git init
```

بعد تنفيذ الأسطر السابقة، ستكون المخرجات على النحو التالي:

```
05.     Initialized empty Git repository in
  /Users/username/Desktop/first-app/.git/
```

لاحظ في المخرجات أعلاه أنها تمت إضافة المجلد .git داخل المشروع. وهذا الملف هو عبارة عن مجلد مخفي خاص ببرنامج git وفيه يتم حفظ جميع معلومات repository المشروع.

أصبح لدينا مستودع خاص بمشروعنا وصار بإمكاننا حفظ وتتبع عملنا وحفظ النسخ والتنقل بينها. وسنقوم بالتعرف على كيفية القيام بذلك في الدروس القادمة بإذن الله.

استخدام الأمر git add

يستخدم الأمر git add لإ

stag

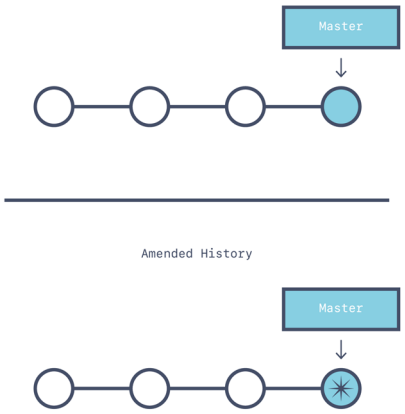
```
06.     git add -A
```

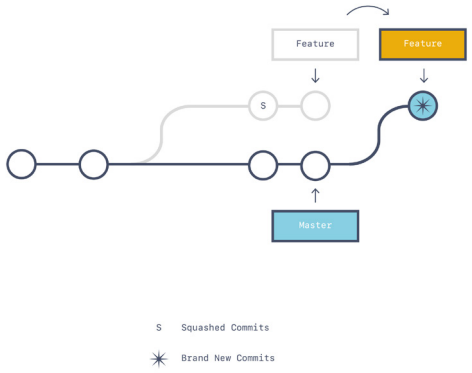
استخدام الأمر git commit

يستخدم الأمر git commit لحفظ التغييرات في المستودع Repository

```
07.     git commit -m «message»
```

بين علامتي التنصيص الموضحة في المثال أعلاه نقوم بكتابة رسالة قصيرة من اختيارنا توضح لنا التغييرات التي تمت على هذه النسخة





• المثال على استخدام الأمر Git commit

في المثال السابق قمنا بإنشاء `Repository` لمشروعنا `first-app`، لنفترض أننا في مرحلة ما قمنا بإجراء بعض التعديلات على محتوى المشروع. علينا حينها أن نقوم بتتبع وحفظ وتخزين تلك التحديثات في `Repository` . لتوضيح الفكرة لاحظ معي الخطوات التالية:

أولا علينا التأكد من وجودنا في ملف المشروع المطلوب `first-app`

```
08.     pwd
```

بعد تنفيذ السطر السابق، ستكون المخرجات على النحو التالي:

```
09.     /Users/username/Desktop/first-app/
```

في حال كانت المخرجات تدل على وجود المستخدم في مجلد آخر فيمكن استخدام `cd` للإنتقال للملف المطلوب.

سنقوم أولا باضافة ملف `README.md` للمشروع

```
10.     touch README.md
```

في هذه الخطوة قمنا بإنشاء ملف من نوع `md` في مجلد المشروع `first-app` ويمكننا بنفس الطريقة إنشاء أي نوع من أنواع الملفات فعلي سبيل المثال لو أننا نقوم بكتابة مشروع بلغة `JavaScript` فسنقوم بإنشاء الملفات باستخدام الأمر `touch` ثم نختار اسم الملف ونحدد نوعه، مثلا: `touch app.js`، وهكذا

حفظ وتخزين التحديث الذي تم على المشروع في `Repository`

```
11.     git add .
12.     git commit -m «initial commit»
```

في السطر الأول قام الامر `add . ` بتحديد وتجهيز كامل الملفات لعملية التخزين بإضافتها إلى `staging area`، ثم قام الأمر `commit` بعملية الحفظ و التخزين الفعلية للملفات في `Repository` المشروع. أرفقت مع التحديث الرسالة التالية "`initial commit`"

تكتب بين علامتي التنصيص الموجودة مع الأمر `git commit` رسالة اختيارية ولكن ينبغي أن تكون ذات معنى ودلالة واضحة على التحديث الذي حصل للمشروع في هذه المرحلة.

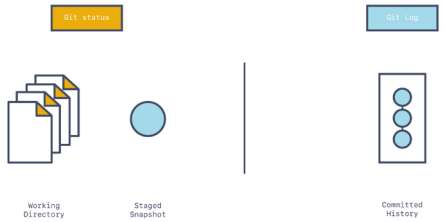
بعد تنفيذ الأسطر السابقة، ستكون المخرجات على النحو التالي:

```
13.     [master (root-commit)
  544376a] initial commit
14.     1 file changed, 0
  insertions(+), 0 deletions(-)
15.     create mode 100644 README.md
```

يوضح السطر الأول بعض المعلومات الخاصة بهذا التحديث كالمعرف `544376a` والذي أنشئ تلقائيا ورسالة التحديث التي قمنا بإرفاقها `initial commit`. ويوضح السطر الثاني نوع وعدد التعديلات التي أجريت على الملف وهو تعديل على ملف واحد فقط، ثم ذكر التعديل في السطر الثالث وهو إنشاء ملف جديد باسم `README.md`. والآن أصبح `Repository` المشروع محدثًا.

استخدام الأمر git status

يستخدم هذا الأمر للاستعلام والتحقق من حالة الملفات والتعديلات التي أجريت عليها




```
16.     git status
```

مثال على استخدام حالةRepository

في بعض الأحيان بعد إجراء بعض التحديثات قد نرغب من التأكد من حالة الملف وأن جميع التحديثات قد تم رفعها وحفظها في المستودع بشكل صحيح، للقيام بذلك يمكننا استخدام الأمر `git status` والذي سيقوم بطباعة تقرير مصغر عن حالة المشروع. لتوضيح الفكرة لاحظ معي السطور التالية:

```
git status
```

يجب أن تكون داخل مجلد المستودع عند تنفيذ الأمر السابقبعد تنفيذ السطر السابق، ستكون المخرجات على النحو التالي:

```
On branch master
nothing to commit, working tree clean
```

يوضح السطر الأول اسم الفرع `branch` الحالي، وهو `master` أي الفرع الرئيسي والذي أنشئ تلقائيا عند إنشاء المستودع. وسنقوم لاحقا بإذن الله بشرح الفروع وعملها وكيف تستخدم وما إلى ذلك بتفصيل أكبر. ويوضح السطر الثاني عدم وجود أي ملفات لم يتم حفظها أو تخزينها.

دعنا الآن نقوم بتحديث المشروع ثم نتحقق من حالته مجددا لنرى شكل المخرجات في حال وجود ملفات لم يتم حفظها وتخزينها. من الممكن أن تكون التحديثات على محتوى ملف معين أو عبارة عن اضافة ملف جديد او حذف ملف وما إلى ذلك، وفي هذا المثال سيكون التحديث عبارة عن إنشاء ملف من نوع `txt` و سنسميه `test-file` ثم سنقوم بتنفيذ الأمر `git status` مرة أخرى، تابع معي الأسطر التالية:

```
touch test-file.txt
  git status
```

بعد تنفيذ الأسطر السابقة، ستكون النتائج على النحو التالي:

```
22.      w(use «git add <file>...» to include in what will be committed)
23.      test-file.txt
24.      nothing added to commit but untracked files present (use «git add» to track)
```

لا يختلف السطر الأول عن السطر الأول في مخرجات المثال السابق، ولكن السطور المتبقية مختلفة، فالسطر الثاني يوضح وجود ملف محدث لم يتم تحديده وإضافته إلى ` staging area `، وهو بالطبع الملف الذي تم إنشاؤه للتو. وفي السطر الثالث يوضح لنا git طريقة إضافة الملف إلى ` staging area ` حتى يصبح جاهزا للحفظ والتخزين، وفي السطر الرابع اسم الملف المحدث. وأخيرا في السطر السادس يوضح ` git ` مجددا عدم وجود ملف جاهز للتخزين وأن هناك ملف جاهز للإضافة إلى ` staging area `

دعنا الآن نقوم باتباع تعليمات ` git ` وننفذ الأمر ` git add ` على ملفنا الجديد، ثم نقوم بالتحقق من حالة المشروع مجددا:

```
25.      git add test-file.txt
26.      git status
```

بعد تنفيذ الأسطر السابقة، ستكون النتائج على النحو التالي:

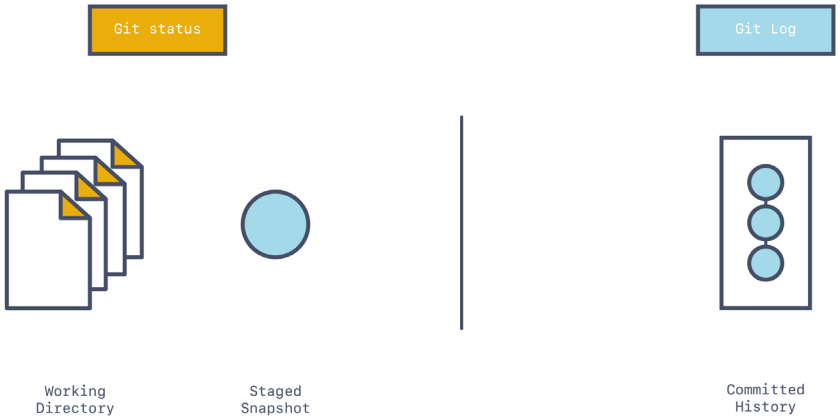
```
27.      On branch master
28.      Changes to be committed:
29.        (use «git restore --staged <file>...» to unstage)
30.      new file:   test-file.txt
```

والآن أصبحت المخرجات في السطر الثاني تنص على أن الملف جاهز للتخزين، بالإضافة إلى إقتراح بتنفيذ الأمر ` git restore ` ليتم حذف الملف من ` staging area `، سنقوم لاحقا بشرح هذا الامر ولكن في الوقت الحالي سنقوم بحفظ وتخزين الملف في Repository المشروع باستخدام ` git commit `، على النحو التالي:

```
31.     git commit -m «add new txt file»
```

والآن أصبحت جميع ملفاتنا محفوظة ومخزنة على المستودع

استخدام الأمر git log



يستخدم هذا الأمر لعرض قائمة باللقطات Commit التي تمت على الفرع الحالي

```
32.     git log
```

- مثال على عرض سجل تحديثاتRepository

يمكننا عرض سجل بجميع التحديثات التي تمت وحفظت في ` Repository ` المشروع باستخدام الأمر ` git log `، كماهو موضح في السطر التالي:

```
33.     git log
```

بعد تنفيذ السطر السابق، ستكون النتائج على النحو التالي:

```
34.     commit bda90bdc19b69f88684deb154c642a16148e5103 (HEAD -> master)
35.     Author: Mohammad <mohammad@hotmail.com>
36.     Date:   Thu May 14 18:03:02 2020 +0300
37.     rename: ReadMe and Delete test-file
38.     commit 521ff1aa68c505ab003b092a78ab9fb50d9afb7e
39.     Author: Mohammad <mohammad@hotmail.com>
40.     Date:   Thu May 14 17:57:03 2020 +0300
41.     add new txt file
42.     commit 544376a25a7302f2fc11b0cf99c8b8e5d84c25c4
43.     Author: Mohammad <mohammad@hotmail.com>
44.     Date:   Thu May 14 17:30:36 2020 +0300
45.     initial commit
```

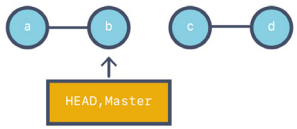
لاحظ أن git قام بعرض سجل التحديثات بشكل تنازلي فالتحديث الأقدم يكون في أسفل القائمة والأجدث في أعلاها. وفي السجل الحالي يوجد ثلاث تحديثات فقط وهي:

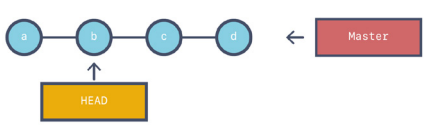
```
46.     commit bda90bdc19b69f88684deb154c642a16148e5103
47.     commit 521ff1aa68c505ab003b092a78ab9fb50d9afb7e
48.     commit 544376a25a7302f2fc11b0cf99c8b8e5d84c25c4
```

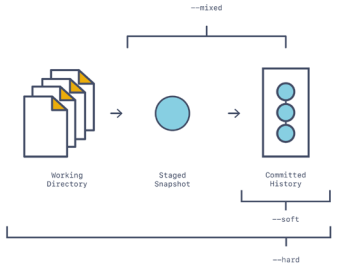
زودتنا المخرجات السابقة بعدد من المعلومات المفيدة عن كل ` commit ` فمثلا لاحظ أنه تم في السطر الأول والسابع والثالث عشر تزويد كل ` commit ` بمعرف خاص به يمكننا من الوصول إليه فيما بعد في حال رغبنا بالانتقال أو العودة إليه. وفي داخل كل ` commit ` نجد معلومات إضافية عن التحديث وهي اسم الشخص الذي قام بالتعديل وبريده الإلكتروني ثم تاريخ التعديل ووقته وأخيرا الرسالة التي كتبناها سابقا عندما قمنا بحفظ التحديثات.

استخدام الأمر git diff

استخدام الأمر git reset







استخدام الأمر git show

03

مقدمة في Command Line

تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل عديدة، من تحديث وحذف وإضافة. فماذا

في هذا الفصل ستتعلم :

- تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل
- عديدة تمر ملفات المشروع أثناء العمل عليه وبنائه
- بمراحل عديدة تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل عديدة

١. مقدمة في Command Line

قبل أن نبدأ في استخدام git والتعامل مع المستودعات والمشاريع ورفع التحديثات فإن علينا أولًا التعرف على طريقة التعامل Command Line، والتي نقوم عن طريقها بتنفيذ العديد من المهام من إنشاء الملفات وحذفها والتنقل بين المجلدات وعرض محتوياتها وما إلى ذلك، بالإضافة إلى كتابة أوامر git. سنتعرف فيما يلي على بعض من أهم تلك الأوامر.

■ موقع المستخدم pwd

يستخدم الأمر pwd لمعرفة موقعك الحالي، ويستخدم على النحو التالي:

```
01.     pwd
```

يقوم الأمر pwd بطباعة مسار المستخدم أي اسم المجلد الذي يوجد فيه المستخدم، على سبيل المثال إن كان المستخدم موجودًا في مجلد سطح المكتب المسمى Desktop فعندها ستتم طباعة:

```
02.     /Users/username/Desktop
```

■ عرض محتويات مجلد ls

يستخدم هذا الأمر لاستعراض محتوى المجلد الذي يتم استدعاء الأمر فيه، لاحظ معي المثال التالي:

```
01.     ls
```

في المثال السابق تم استدعاء أمر ls من داخل مجلد Desktop و بالتالي سوف يقوم بعرض محتوى هذا الملف كما هو موضح بالشكل التالي :

```
02.     HomeWork        image.png
03.     myProject       test.txt
```

التنقل بين المجلدات باستخدام cd

يستخدم الأمر cd للتنقل بين المجلدات، لاحظ معي المثال التالي:

```
01.     cd Desktop
```

المثال أعلاه يوضح انتقال المستخدم لسطح المكتب عن طريق كتابة الأمر cd متبوعاً بإسم المجلد المراد الإنتقال له.

و الان عند استخدام أمر pwd سوف يتم طباعة المسار التالي :

```
02.     pwd
03.     /Users/username/Desktop
```

في حال كان اسم المجلد يحتوي على مسافة، نضع اسم المجلد بين علامات التنصيص ""، لاحظ مع المثال التالي:

```
04.     cd "folder name"
```

يوضح المثال أعلاه الانتقال لمجلد يحتوي اسمه على مسافة folder name

يُستخدم هذا الأمر للانتقال بين المجلدات ولا يمكن استخدامه مع الملفات

للعودة إلى الخلف خطوة واحدة نستخدم امر cd متبوعا بنقطتين .. كما هو موضح بالمثال التالي :

```
05.     pwd
06.     /Users/username/Desktop
07.     cd ..
08.     pwd
09.     /Users/username
```

و للعودة إلى الملف الرئيسي نستخدم الأمر cd

```
10.     cd
```

فتح ملف open

يستخدم الأمر open لفتح ملف:

```
11.     open image.png
```

إنشاء مجلد mkdir

يستخدم الأمر `mkdir` لإنشاء مجلد جديد، لاحظ معي المثال التالي :

```
01.          mkdir my-folder
```

في المثال أعلاه قمنا بإنشاء مجلد جديد باسم. `my-folder`

يستخدم هذا الأمر لإنشاء المجلدات فقط،
ولا يمكن استخدامه لإنشاء ملف.

إنشاء ملف touch

يستخدم هذا الأمر لإنشاء ملف، لاحظ معي المثال التالي:

```
02.          touch fileName.txt
```

المثال أعلاه يوضح إنشاء ملف جديد باسم `fileName` من نوع. `txt`

نقل ملف mv

يستخدم الأمر `mv` لنقل ملف من مجلد إلى مجلد آخر، ويستخدم على النحو التالي:

```
01.          mv fileName.txt my-folder/
```

في المثال أعلاه قمنا بنقل الملف المسمى `fileName.txt` إلى المجلد ``my-folder``

لكي تتمكن من نقل الملف يجب أن تكون داخل
المجلد الذي يحتوي على هذا الملف

حذف ملف باستخدام rm

يستخدم هذا الأمر لحذف الملفات، ويستخدم على النحو التالي:

```
01.          rm fileName.txt
```

سيقوم الأمر في المثال السابق بحذف الملف المسمى `fileName.txt`

لا يمكنك استعادة الملفات المحذوفة باستخدام هذا الأمر، مما يعني اننا يجب ان نكون حذرين عند التعامل مع هذا الأمر لكي لا نقوم بحذف احد ملفاتنا المهمة.

حذف مجلد rm -r

يستخدم هذا الأمر لحذف المجلدات، ويستخدم على النحو التالي:

```
02.          rm -r my-folder
```

سيقوم الأمر في المثال أعلاه بحذف المجلد المسمى `my-folder` وجميع محتوياته.

03

نظرة على مفهوم الفروع Branches

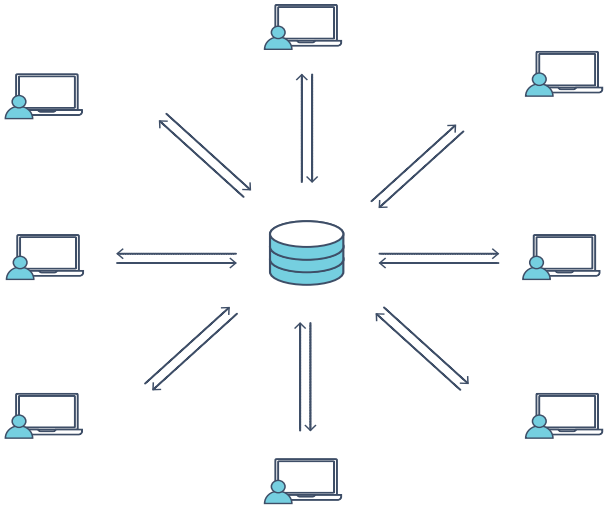
تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل عديدة، من تحديث وحذف وإضافة. فماذا

في هذا الفصل ستتعلم :

- تعرفنا في هذا الدرس على مفهوم الفروع وطريقة إنشاءهاو استخدامها

١. نظرة على مفهوم الفروع Branches

ما هي الفروع



ذكرنا سابقا أن git عملية جدا ومفيدة لعمل الفرق، فعلى سبيل المثال لو كان الفريق البرمجي يتكون من ثلاث أشخاص وكل منهم مسؤول عن خاصية معينة يقوم باضافتها وتحسينها فكيف سيتم الأمر وكيف سيدار المشروع، هل سيقوم كل شخص بانتظار دوره للبرمجة حتى ينتهي المبرمج الآخر من البرمجة ثم يرسل له المشروع؟ من الممكن بالطبع القيام بذلك لكن هذا فيه هدر للوقت، إذا مالحل وما الطريقة التي تستخدم في git للتعامل مع هذا الأمر ؟

توفر لنا git ما يسمى بالفروع Branches وهي عبارة عن تفرعات أو نسخ فرعية من المشروع، أي أن العضو الذي سيعمل على أمر أو خاصية ما يقوم بإنشاء فرع ويعمل عليه وعندما ينهي عمله يقوم بارساله ودمجه مع الفرع الرئيسي للمشروع والذي يعرف باسم master.

استخدام الأمر git branch لعرض الفروع

هنالك عدد من الأوامر التي يمكن تنفيذها على الفرع branch في المستودع المحلي أي على المستودع الموجود على جهازك الشخصي، حيث يمكننا عرض الفروع الموجودة مسبقاً و إنشاء فرع جديد وكذلك حذف أي فرع من تلك الفرع، ويمكن القيام بذلك على النحو التالي:

العرض : يستخدم الأمر التالي لعرض كل فروع المستودع المحلي

```
git branch
```

خلونا الآن نأخذ مثال عملي على طريقة التعامل مع الفروع:

أولا خلونا ننشيء مجلد باسم `my-app` ثم ننتقل له باستخدام `cd` وبعدها نقوم بتعريفه وتهيئته باستخدام `git init` ، وبعد ذلك نضيف بعض التعديلات على المشروع مثلا ننشئ ملف `txt` ونسيمه `students` والان خلونا نخزن هذا المشروع ونحفظه وبنحط ملاحظة تدل على التعديل الخاص في هذي النسخة زي ما تعودنا وفي حالتنا هنا بنكتب `add txt file`

```
mkdir my-app
cd my-app
git init
touch students.txt
git add -A
git commit -m «add txt file»
```

لان بما انه صار عندنا مشروع مخزن فخلونا نتحقق من عدد فروع هذا المشروع، ونقدر نتحقق من ذلك باستخدام الأمر git branch ويكتب بهذا الشكل:

```
08.     git branch
```

المخرجات:

```
09.     * master
```

زي ما هو موضح في المخرجات، المشروع لا يحتوي إلا على فرع واحد وهو الفرع الرئيسي master

استخدام الأمر git branch <name> (انشاء فرع)

يستخدم هذا الأمر لإنشاء فرع في المستودع المحلي

```
10.     git branch <branch name>
```

يمثل <branch name> اسم الفرع المراد إنشاؤه

تعرفنا في الفيديو السابق على طريقة عرض الفروع الخاصة بالمشروع وزي ما هو موضح على الشاشة فالمشروع لا يحتوي إلا على فرع واحد وهو الفرع الرئيسي

الان خلونا نقوم بانشاء فرع ثاني وللقيام بذلك نقوم بكتابة الأمر ` git branch ` متبوع باسم الفرع وهذا الاسم من اختيارنا،

ويكتب الأمر بالشكل التالي:

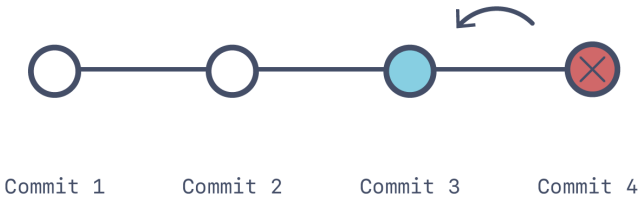
```
11.     //git branch <name>
12.     git branch developer-1
```

والان عشان نتأكد أن الفرع أنشئ فعلا بنستخدم الأمر ` git branch ` لعرض جميع الفروع المنشأة لهذا المشروع، وبتكون المخرجات على النحو التالي:

```
13.     developer-1
14.     * master
```

عند تنفيذ هذا الأمر ما ظهر لنا أي تغيير في المشروع، وهذا لأن إنشاء الفرع لا يحدث أي تغيير على المشروع نفسه بل يأخذ نسخة منه فقط ونشتغل ونعدل على هذي النسخة ولما ننجز مهمتنا بنجاح ونتأكد أن الخاصية التي نبي نضيفها جاهزة فوقتها نقوم بدمج هذا الفرع مع الفرع الرئيسي

■ استخدام الأمر git checkout <name> (الانتقال لفرع)



يستخدم هذا الأمر للإنتقال من branch إلى آخر.

```
15.     git checkout <branch name>
```

يمثل `<branch name>` اسم الفرع المرجو الانتقال إليه

قمنا في الفيديو السابق بإنشاء فرع جديد وسميناه `developer-1` عشان نبدأ العمل على هذا الفرع نحتاج بالأول ننتقل لهذا الفرع، ونقوم بذلك باستخدام الأمر

```
16.     //git checkout <name>
17.     git checkout developer-1
```

عند تنفيذ هذا الأمر بنحصل على رسالة التأكيد التالية:

```
18.     Switched to branch ‹developer-1›
```

تنص الرسالة على أننا انتقلنا للفرع `developer-1`

الان لو قمنا بعرض الفروع باستخدام `git branch` فبتظهر لنا المخرجات التالية:

```
19.     * developer-1
20.     master
```

لاحظ أن النجمة أصبحت بجانب الفرع `developer-1` بمعنى أنه الفرع الذي نحن فيه الآن

وببساطة هذي هي طريقة اختيار الفرع اللي بي نشتغل عليه والانتقال له

استخدام الأمر git checkout -b (انشاء فرع والانتقال له)

يقوم هذا الأمر بإنشاء branch **جديد و يتنقل إليه أيضًا**

```
21.      git checkout -b <name_of_your_new_branch>
```

يمثل ` <name_of_your_new_branch> ` اسم الفرع المرجو انشاؤه و الانتقال إليه

تعلمنا في الدروس السابقة طريقة انشاء فرع جديد باستخدام ` git branch <name> ` متبوع باسم الفرع ثم الانتقال إليه باستخدام ` git checkout <name> ` متبوع باسم ذلك الفرع نقدر نستخدم أمر آخر يقوم بالخطوتين هذي اللي هو git checkout -b متبوع باسم الفرع، واللي باستخدامه راح ينشئ الفرع وينتقل له تلقائياً، عشان توضح الفكرة خلونا نطبق مثال عليه

عندنا الان فرعين وهم master و ` developer-1` -واللي استعرضناهم بالأمر git branch-، نلاحظ اننا حالياً متواجدين في الفرع developer-1، الحين خلونا ننشئ branch وننتقل له بنفس اللحظة باستخدام git branch -b، وخلونا نسمي هذا الفرع developer-2

```
22.      git checkout -b developer-2`
```

الحين لو نعرض الفروع باستخدام الامر ` git branch ` مره ثانية ، بنلاحظ ان الفرع ` developer-2 ` تم انشاءه بالفعل واننا انتقلنا له ومتواجدين فيه حالياً

وهذه ببساطة هي طريقة استخدام الامر ` git checkout -b `

استخدام الامر git branch -d <name> (حذف فرع)

يستخدم الأمر التالي لحذف فرع من فروع المستودع المحلي

```
23.      git branch -d <branch name>
```

يمثل <branch name> اسم الفرع المراد حذفه

فلفترض اننا نريد حذف فرع ما ولأي سبب من الأسباب مثلا خلصنا شغلنا على هالفرع ومعد نحتاجه، نقدر نحذف ي فرع باستخدام الأمر ` git branch -d <name>، في المثال السابق أنشأنا branch وسميناه ` developer-2 ` فخلونا الان نحذفه، ولكن قبل حذف الـ branch يجب أن نتأكد انا مو علىه وان كنا عليه فعلينا ان ننتقل لـ ` branch ` اخر والا ستظهر لنا رسالة خطأ. وطبعا نستخدم الامر ` git branch ` لمعرفة وتحديد الفرع الحالي

الان خلونا ننتقل للفرع master وبعدها نحذف الـ branch اللي اسمه ` developer-2 ` وللقيام بذلك نكتب الامر بهذا الشكل:

```
24.      git branch -d developer-2
```

عند تنفيذ الأمر ستظهر لنا الرسالة التالية:

```
25.      Deleted branch developer-2 (was 7f314b6).
```

وتعني ان الفرع المحدد واللي هو ` developer-2 ` تم حذفه بالفعل ، وهذه هي طريقة حذف الفروع في ` git `

04

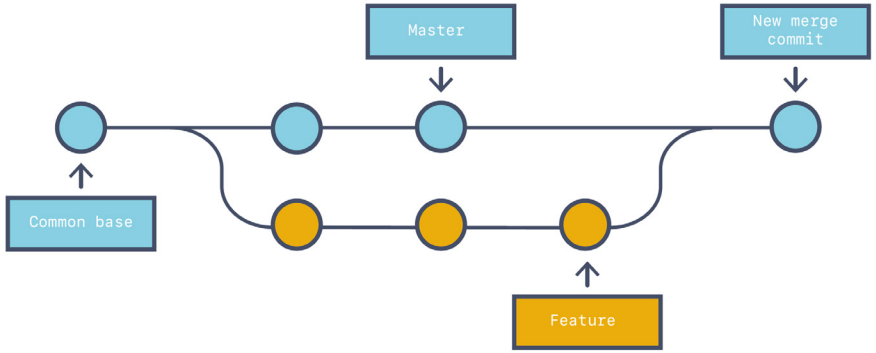
مقدمة في دمج الفروع

تمر ملفات المشروع أثناء العمل عليه وبنائه بمراحل عديدة، من تحديث وحذف وإضافة. فماذا

في هذا الفصل ستتعلم :

- تعرفنا في هذا الدرس على طريقة دمج الفروع وعلى طريقة التعامل مع التعارض

١. مقدمة في دمج الفروع



استخدام الأمر git merge لدمج الفروع يقوم هذا الأمر بدمج الفرع المحدد بالفرع الحالي

```
01.     git merge <branch name>
```

يمثل `<branch name>` اسم الفرع المراد دمجه بالفرع الحالي

فلنفترض اننا الان قمنا باضافة أمر ما للمشروع في الفرع `developer-1` ، على سبيل المثال،واتممنا المهمة أو العمل المطلوب منا اضافته، الحين كيف راح نقوم بدمج عملنا اللي في الفرع developer-1 مع الفرع الرئيسي `master` ؟

توفر لنا git أمر الدمج `git merge` وهذا أمر يستخدم لدمج العمل الموجود الفروع اللي انشأناها مع الفرع الرئيسي ماستر، خلونا نجرب الأمر مع بعض أول شي

خلونا نعرض محتوى المشروع باستخدام الأمر `ls` ولاحظوا انا لازلنا على الفرع `master` ، وزي ماهو موضح ما عندنا الا ملف واحد في المشروع اللي هو `students.txt` ، الان خلونا ننتقل للفرع اللي بنضيف شغلنا عليه وهو الفرع `developer-1` باستخدام الأمر git checkou ثم نضيف ملف txt جديد ونسميه `teachers` ، طيب لو عرضنا محتوى المشروع الان بنلاقي ان صار عندنا ملفين وهي students.txt و teachers.txt، خلونا الان نحفظ ونخون هذي الملفات في مشروعنا، ونكتب الأمرين التالية:

```
02.      git add -A
03.      git commit -m ‹add teachers.txt›
```

الخطوة التالية هي أن ننتقل للفرع الرئيسي master واللي بيتم الدمج له

```
04.      git checkout master
```

لاحظ معي الان انه عند عرض محتوى المشروع المحفوظ في الفرع master ما بيطلع لنا الا الملف students.txt وهذا لأن الاضافة والتغيير اللي حصل على المشروع حصل في branch ثاني واللي هو `developer-1`

الان وصلنا لاخر مرحلة وهي دمج الاضافة مع المشروع المحفوظ في الفرع `master`،وعشان ننفذ هذا الأمر بنستخدم الأمر merge ونحدد اسم الفرع الذي نريد دمجه، وفي حالتنا هذه الفرع هو `developer-1` فيكتب الأمر بهذا الشكل:

```
05.      git merge developer-1
```

لاحظ الرسالة اللي ظهرت لنا واللي تدل على أن المهمة تمت بنجاح

```
06.     Updating 7f314b6..6486bb6
07.     Fast-forward
08.       style.css | 0
09.       1 file changed, 0
  insertions(+), 0 deletions(-)
10.       create mode 100644 style.css
```

استخدام الأمر abort للتراجع عن الدمج

في بعض الأحيان لا يكون الدمج بهذه البساطة، فمثلا قد يقوم شخصين من الفريق بالتعديل على نفس الجزء من الملف دون أن يعلما، وعندها كيف ستتم عملية الدمج أي من الملفين سيتم الاحتفاظ به، لتوضيح الأمر سنستخدم مثال بسيط، فلنفترض ان كلا الشخصين والذين يعملان على فرعين مختلفين قاما باضافة تنسيقات على نفس العنصر أحدهما جعله أزرق والآخر أخضر ثم حفظا عملهما والان يريدنا دمجه، عند تنفيذ الامر git merge sj ستظهر لنا الرسالة التالية:

```
11.     Auto-merging style.css
12.     CONFLICT (content): Merge conflict in style.css
13.     Automatic merge failed; fix conflicts and then commit the result
```

وتنص هذه الرسالة على وجود خطأ وأن عملية الدمج لم تتم بنجاح، وتذكر تحديديا الملف الذي وقعت فيه المشكلة وهو ` style.css، هناك عدد من الحلول لهذه المشكلة الأول وهو التراجع عن عملية الدمج كاملة، حيث أنه في هذه المرحلة تم دمج جميع الملفات ماعدا ملف style.css وهذا قد يتسبب في بعض المشكال والاعطال، للتراجع والعودة للنقطة التي كنت فيها قبل الدمج يمكننا استخدام الأمر التالي:

```
14.     git merge --abort
```

قد يكون هذا خيار مناسب لك. لكن بالطبع يمكنك اتمام عملية الدمج لو أردت وهذا ما ستتعرف عليه في الجزء الثاني من هذا الدرس

إتمام عملية الدمج عند التعارض

لمعرفة الملفات المتسببة في المشكلة يمكننا استخدام الأمر git status ، للتحقق من حالة الملفات، وعندها ستظهر لنا الرسالة التالي:

```
15.       On branch master
16.      You have unmerged paths.
17.      (fix conflicts and run ågit commitò)
18.      (use ågit merge --abortò to abort the merge)
19.      Unmerged paths:
20.      (use ågit add <file>...ò to mark resolution)
21.      both modified:   style.css
22.       no changes added to commit (use ågit addò and/or ågit commit -aò)
```

لاستكمال الدمج، يمكننا الذهاب للملف وحل المشاكل الحاصلة فيه يدويا ثم نقوم بحفظها ثم نستخدم الأمر git add لاضافتها إلى منطقة الإدراج وبعدها ننفذ الأمر git commit، وعندها ستظهر لنا الرسالة التالية:

```
    merge branch ‹developer-1›
        Conflicts:
        style.css
        #
        It looks like you may be committing a merge.
        If this is not correct, please remove the file
        .git/MERGE_HEAD
        and try again.
        Please enter the commit message for your changes. Lines starting
        with ‹#› will be ignored, and an empty message aborts the commit.
        #
        On branch master
        All conflicts fixed but you are still merging.
        #
        Changes to be committed:
        modified: style.css
```

والان ستتم عملية الدمج بنجاح