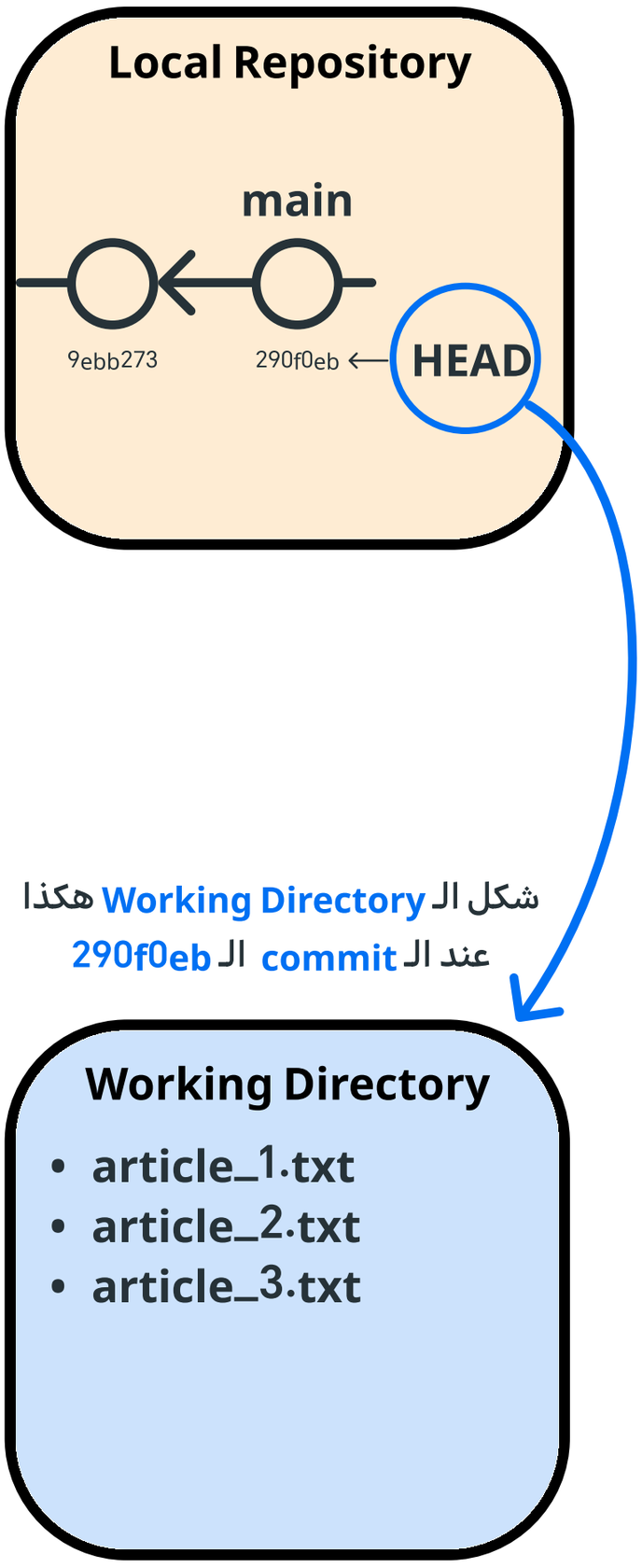
تعلم استخدم Git
لكي تسافر عبر الزمن والأبعاد

ما هو الـ HEAD ؟ وما مفهوم الـ Branch في Git ؟
هل فعلًا الـ Git جهاز للسفر عبر الزمن والأبعاد؟

Ahmed El-Tabarani
Back-End Developer

أولًا لنتعرف على الـ HEAD الكبير

هو مؤشر يشاور على commit، وغالبا يكون على آخر commit في الـ branch، ويمكننا جعله يؤشر على أي commit آخر



شكل الـ Working Directory هكذا عند الـ commit الـ 290f0eb

ملحوظة:

مكان الـ HEAD يعكس الملفات التي في الـ Working Directory

فإذا غيرنا مكان الـ HEAD وجعلناه يشاور على commit قديم

فإن محتوى الـ Working Directory ستكون نفس حالة المشروع

التي كان عليها في هذا الـ commit القديم

كما لو أن الـ HEAD أداة السفر عبر الزمن الخاصة بالـ Git

كيف نحرك مكان الـ HEAD ؟

قبل أن نتعمق في عالم الـ Branch

دعونا نستكشف كيف نسافر عبر الزمن عن طريق

تحريك مكان الـ HEAD

ونجعله يشاور على أي commit حتى وإن كان قديم

يوجد العديد من الأوامر التي نستطيع من خلالها تحريك الـ HEAD،

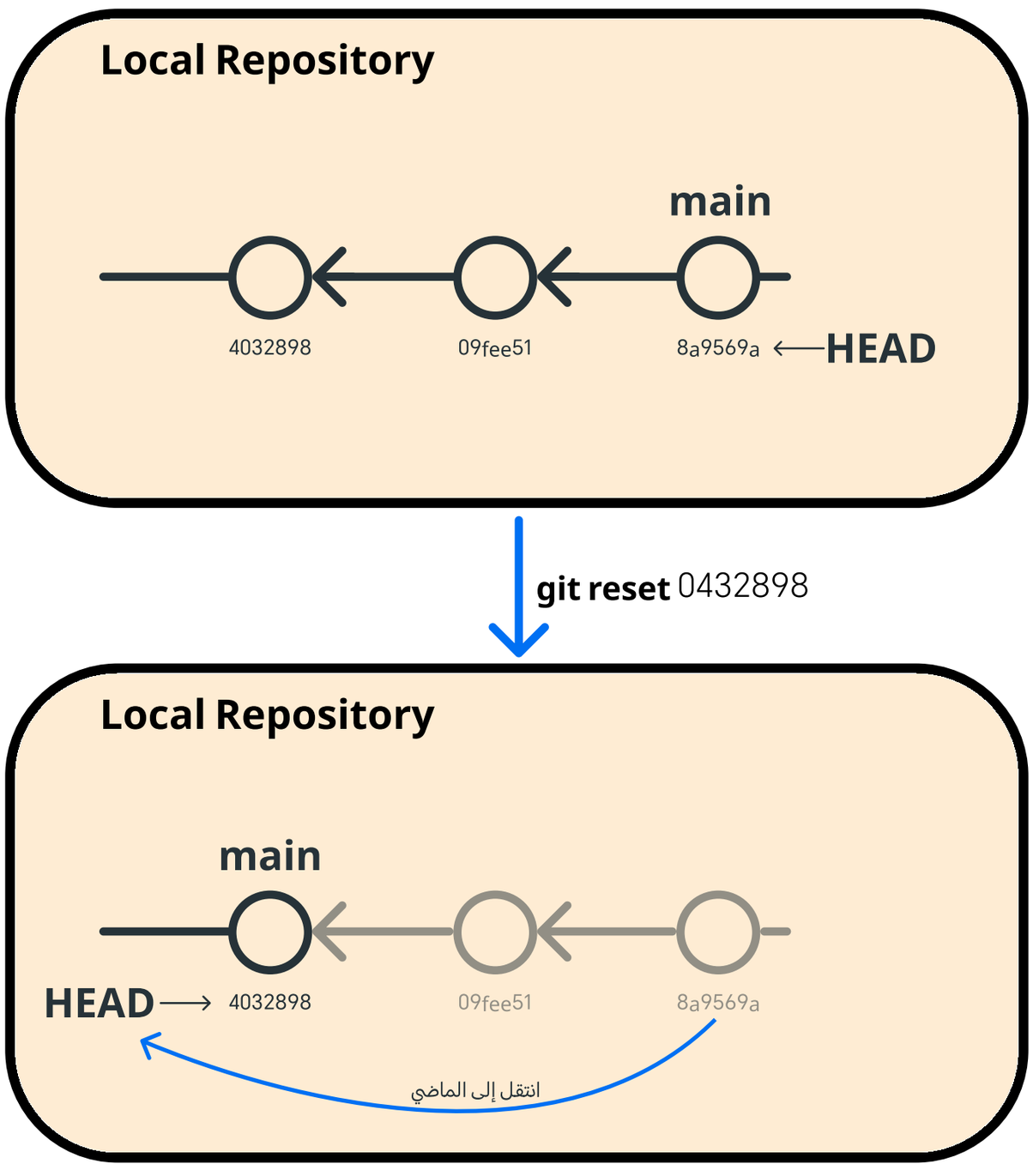
ومن ضمنها:

git reset, git checkout, git switch

Ahmed El-Tabarani
Back-End Developer

git reset

نستخدمه لتحريك كلا الـ HEAD والـ branch
في آن واحد إلى الـ commit الذي يتم تحديده



عليك أن تكون حذرًا في استخدام
الـ git reset لكي لا تفقد كل شي

يمكنك استخدام get reset HEAD~n
لتنتقل للماضي بعدد n من الـ commit

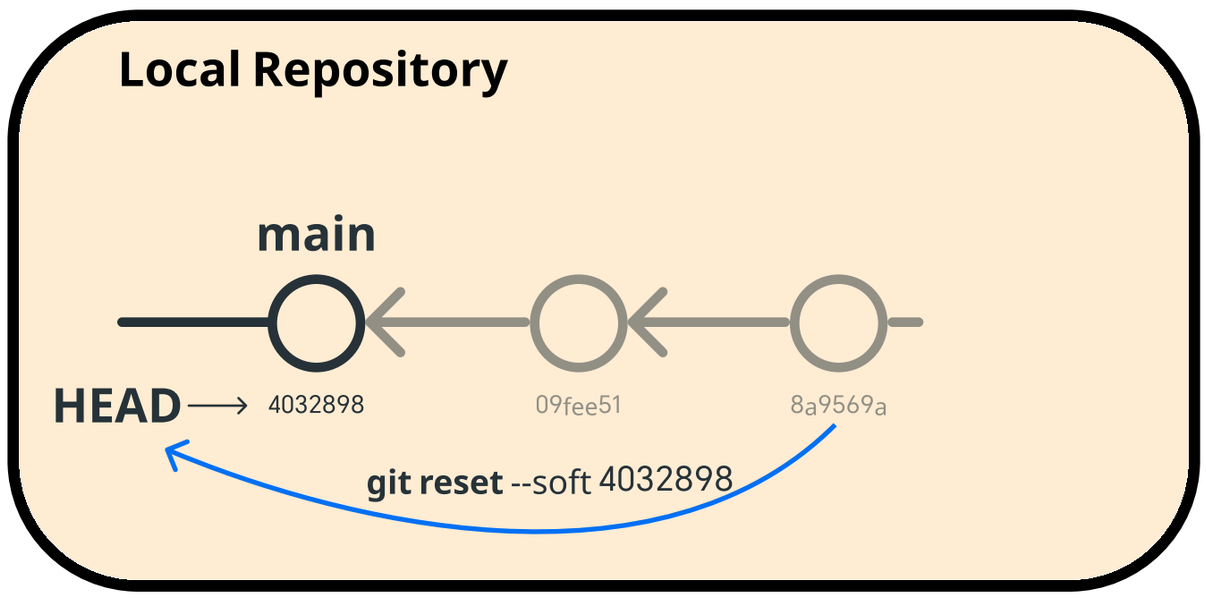
وللعودة للـ commit الذي كنت فيه
يمكنك استخدام الأمر git reflog
لعرض كل الـ commit ثم معرفة
الـ commit الذي كنت فيه
ثم استخدام git reset مجددًا للإنتقال إليه

الم أقل لك أن الـ Git يمتلك جهاز لسفر عبر الزمن

هناك ثلاث خيارات مهمة في الـ git reset وهم --soft و--mixed و--hard
وعليك ان تفرق بينهم وخصوصًا --hard لكي لا تفقد ما كل شيء

Ahmed El-Tabarani
Back-End Developer

git reset --soft



Working Directory

الملفات التي كانت في الـ commit
الذي كنا فيه 8a9569a تظل كما هي
ولا يتم حذفها

والملفات التي يتم التعديل عليها هنا
والتي لم تنقل إلى مرحلة الـ Staging بعد
سيتم نقلها تلقائيًا

git add .

يتم تنفيذ هذا الأمر تلقائيًا بعد تنفيذ git reset

Staging Area

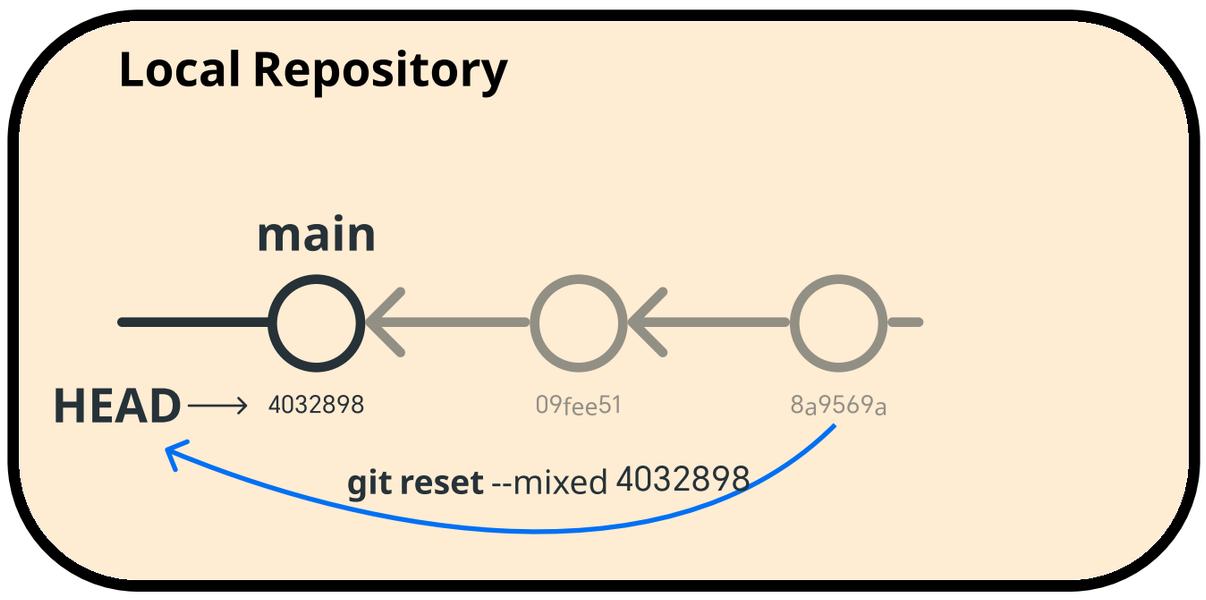
الملفات المعدلة
التي تم نقلها هنا تظل كما هي
لا يتم حذفها

أهم النقاط:

- لا يقوم بتعديل الـ Working Directory على الإطلاق بل يظل كما هو قبل الـ git reset
- لا يفعل أي شيء مع الملفات التي في مرحلة الـ Staging بل تظل كما هي
- التعديلات التي حدثت ما بين الـ HEAD والـ commit المحدد يتم نقلها إلى الـ Staging
- إذا كان هناك تعديلات كنا نعمل عليها في الـ Working Directory
ولم تنقل إلى الـ Staging فسيتم نقلها تلقائيًا كما لو أنه قام بتنفيذ git add

Ahmed El-Tabarani
Back-End Developer

git reset --mixed (القيمة الافتراضية)



Working Directory

الملفات التي كانت في الـ commit الذي كنا فيه 8a9569a تظل كما هي ولا يتم حذفها

والملفات التي يتم التعديل عليها هنا والتي لم تنقل إلى مرحلة الـ Staging بعد ستظل كما هي ولن يتم نقلها

git restore --staged .

يتم تنفيذ هذا الأمر تلقائيًا بعد تنفيذ git reset

Staging Area

الملفات المعدلة التي تم نقلها هنا تنقل إلى الـ Working Directory

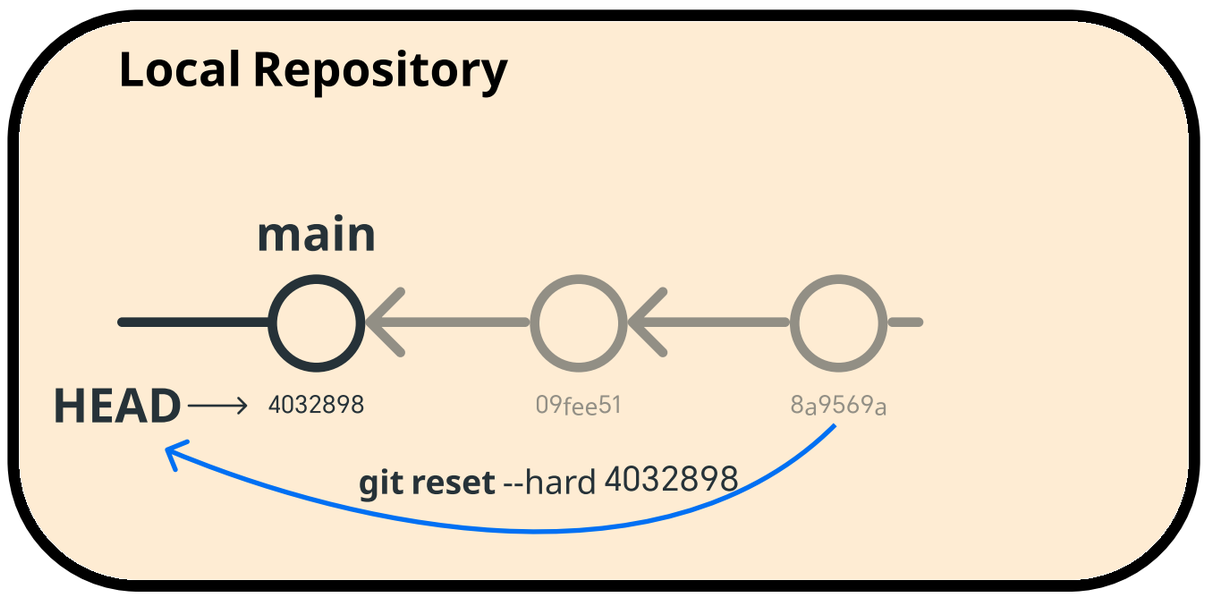
---

أهم النقاط:

- لا يقوم بتعديل الـ Working Directory على الإطلاق بل يظل كما هو قبل الـ git reset
- كل الملفات التي في مرحلة الـ Staging تخرج ويتم نقلها إلى الـ Working Directory كم لو أنه قا بتنفيذ git restore --staged
- التعديلات التي حدثت ما بين الـ HEAD و commit لا يتم نقلها إلى الـ Staging بل تظل كما هي في الـ Working Directory
- إذا قمنا بتنفيذ --mixed ثم قمنا بعمل git add فسنحصل على نفس النتيجة إذا نفذنا --soft من البداية

Ahmed El-Tabarani
Back-End Developer

git reset --hard (احذر أثناء استخدامه)



Working Directory

الملفات التي كانت في الـ commit
الذي كنا فيه 8a9569a
يتم حذفها

والملفات التي يتم التعديل عليها هنا
والتي لم تنقل إلى مرحلة الـ Staging بعد
سيتم التخلص منها **دون رجعة**

Staging Area

الملفات المعدلة التي تم نقلها هنا
يتم التخلص منها **دون رجعة**

احذر من استخدام hard-- لأنه:

يتخلص من أي تعديلات تقوم بالعمل عليها سواء
التعديلات التي في الـ Working Directory
أو التي نقلت إلى الـ Staging
ستختفي دون رجعة

أهم النقاط:

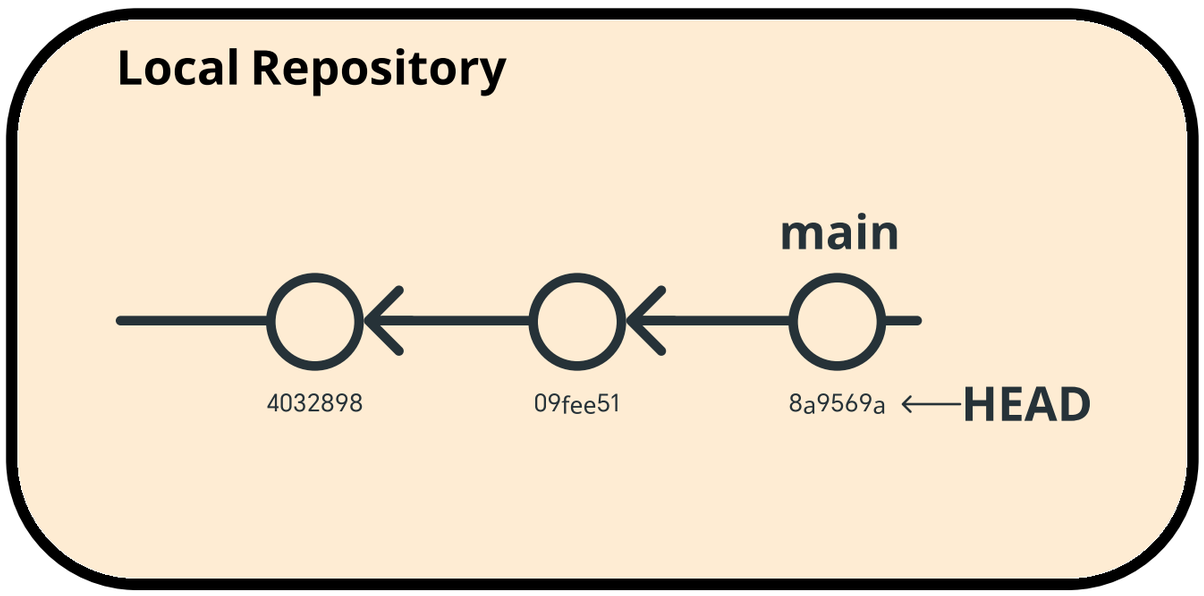
- يحرك الـ HEAD إلى الـ commit وينقل التعديلات إلى الـ Working Directory مباشرةً
- الملفات التي في الـ Working Directory يتم استبدالها اجباريًا
  لجعل الـ Working Directory يطابق النسخة التي كان عليها في الـ commit المحدد
- يتخلص من أي تعديلات كانت في مرحلة الـ Staging دون رجعة
- يتخلص من أي تعديلات كانت في الـ Working Directory دون رجعة
  أي أنه يتخلص من الـ Modified Files

Ahmed El-Tabarani
Back-End Developer

git checkout

له استخدامات عديدة منها

تغير مكان الـ HEAD فقط إلى commit محدد



لاحظ التالي:

- الـ HEAD هو من تحرك فقط
  والـ main بقى كما هو
- يمكننا جعل الـ HEAD يرجع لأصله
  عن طريق git checkout main
- عندما يصبح الـ HEAD يشاور
  على commit مجهول
  أي ليس هناك اي branch عليه
  ويسمى هنا detached HEAD

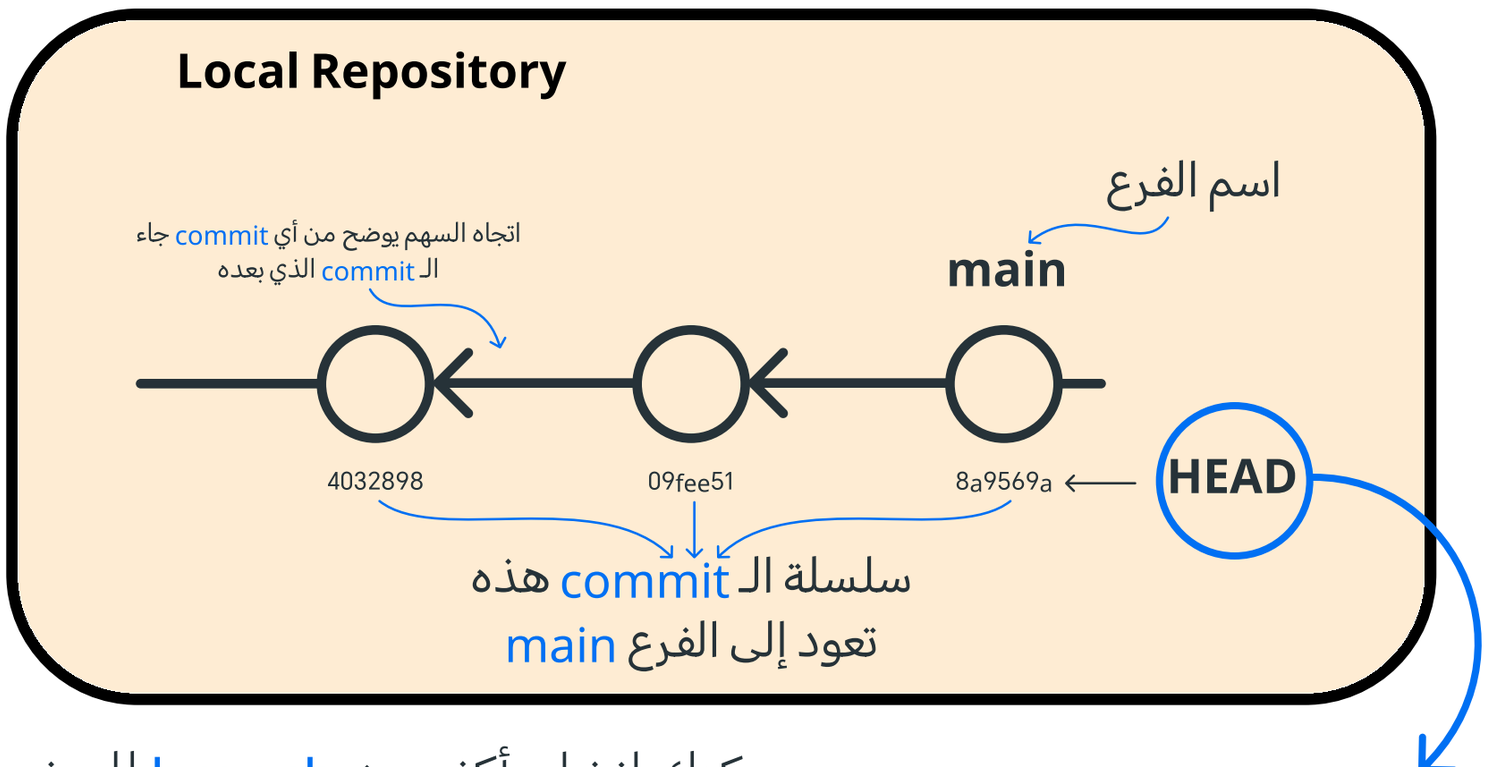
في حالة الـ detached HEAD اذا كنت تريد عمل تعديلات
فيجب أن تنشيء branch في مكان الـ HEAD ثم يمكنك عمل
commit جديد يضم هذه التعديلات
ويكون مرتبط بهذا الـ branch

إذا كنت تملك ملفات تقوم بالتعديل عليها في الـ Working Directory
أو في الـ Staging، فلن تستطيع تنفيذ الـ git checkout
والـ Git سينبهك بأنك لا يمكنك تحريك الـ HEAD وهناك ملفات يتم التعديل عليها

Ahmed El-Tabarani
Back-End Developer

ثانيًا ما هو الـ Branch؟

أهم المزايا التي يقدمها الـ Git هي فكرة الـ branch وهو تجميع أكثر من commit تحت مظلة واحدة ويستخدم الـ HEAD ليتنقل ما بينهم



ويمكنك إنشاء أكثر من branch للمشروع فعلى سبيل المثال فقط

فرع خاص بالنسخة المستقرة للمشروع الخاصة بالـ Production

فرع خاص فقط بالتطوير واختبار الأشياء فقط

ويمكن لكل فريق أو شخص أن يكون ينشيء فرع خاص للشيء الذي يعمل عليه سواء إضافة شيء او تعديل او إصلاح مشكلة

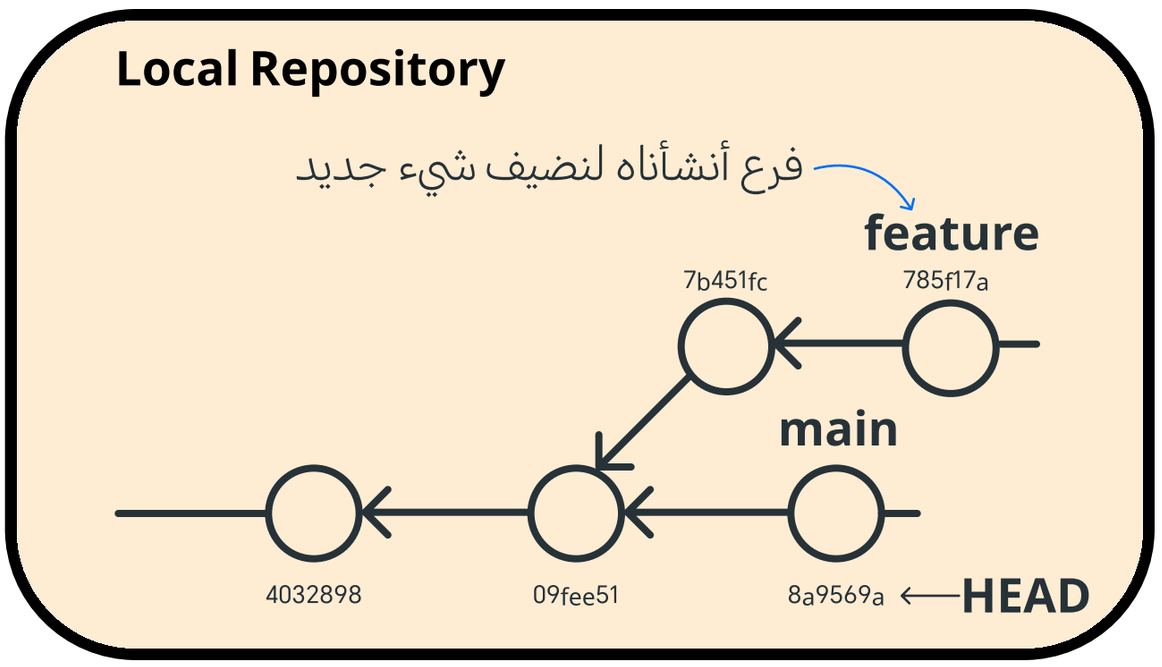
والسبب أن الـ branch الواحد يكون مستقل تماما عن باقي الفروع الأخرى، فيمكنك إنشاء فرع وتلعب فيه وتختبر ثم تحذفه وكأنه لم يكن

Ahmed El-Tabarani
Back-End Developer

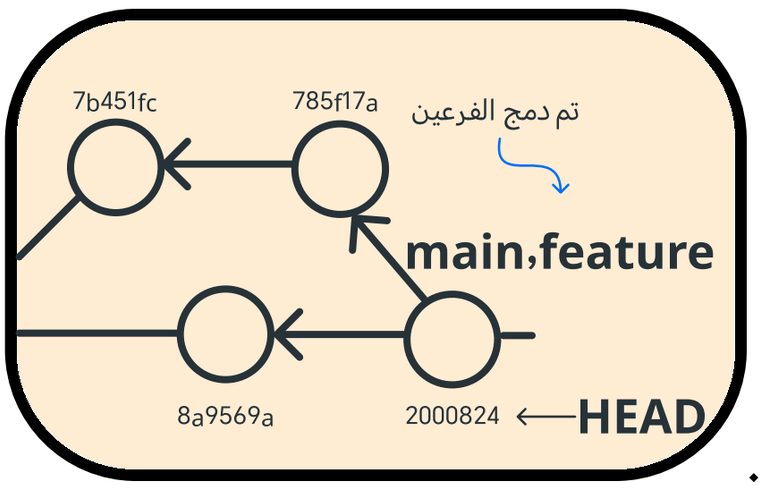
ط

ما مميزات الأخرى في إنشاء أكثر من فرع ؟

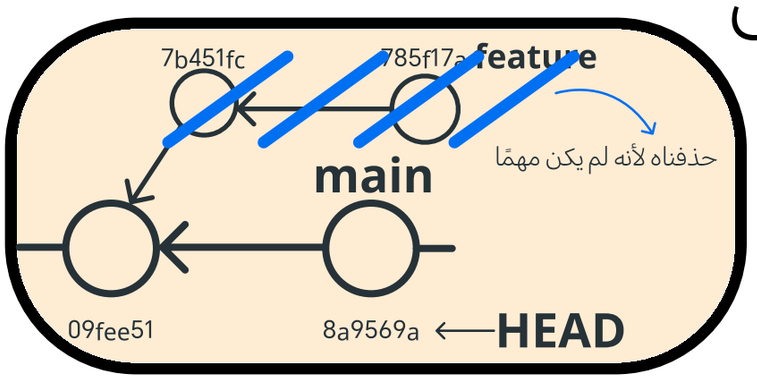
أولًا تقسيم العمل ضمن الفرق حيث يمكن لكل عضو أو كل فريق العمل على نسخته من المشروع في فرع مختلف ومستقل عن الآخرين



وعندما ينتهي أحد من عمله داخل نسخته من المشروع ويختبرها وتكون جاهزة



يمكننا أن ندمجها مع الفرع الرئيسي وتصبح جزءًا من المشروع

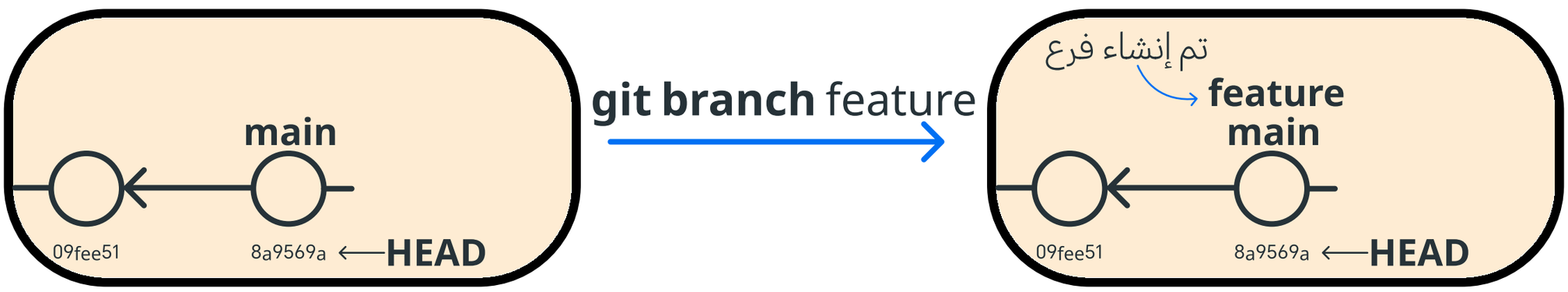


أو يمكنه حذف الـ branch وكأن شيء لم يكن في حالة أنه كان فقط يختبر شيء ما أو أنه لم يعد مهمًا للمشروع ولا يريده أحد

Ahmed El-Tabarani
Back-End Developer

git branch

الآن لنفترض أنني أريد أن أقوم بإضافة شيء ما في المشروع ولا أريد أن أعبث بالفرع الرئيسي هنا يمكننا أن ننشيء branch جديد، ونقوم بعمل ما نريده فيه



ونستطيع معرفة كل الـ branch التي لدينا عن طريق git branch -v

```
> git branch -v
  feature  8a9569a Revert "add xyz article"
* main     8a9569a Revert "add xyz article"
```

لاحظ أنه الآن اصبح لدينا فرع جديد وهو feature

ووضعنا -v لكي نعرض آخر commit يقف عليه كل فرع

وعلامة * تدل على مكان الـ HEAD وتدل على الفرع الذي نقف فيه حاليًا

ونحن حاليًا على main

وستلاحظ ان كلا من الـ main و feature لديهم نفس الـ commit

وهذا يدل أن feature أشتق من الـ main بالفعل

Ahmed El-Tabarani
Back-End Developer

git switch

الآن نريد أن نجعل الـ HEAD يترك الـ main ويذهب إلى الفرع الـ feature، ماذا نفعل ؟

ببساطة لدينا أمر متخصص لهذا وهو git switch

```
> git switch feature
Switched to branch 'feature'
```

الآن نقوم بتنفيذ git branch لنتأكد فقط بأن الـ HEAD أم لا

```
> git branch
* feature
  main
```

سترى أن * على feature لذا فأن الـ HEAD فعلًا تحرك إلى الـ feature

---

الآن ونحن في الـ feature لنقم بإضافة ملف جديد article_2.txt ثم نقوم بعمل commit لها لنرى ما الذي سيحدث

Ahmed El-Tabarani
Back-End Developer

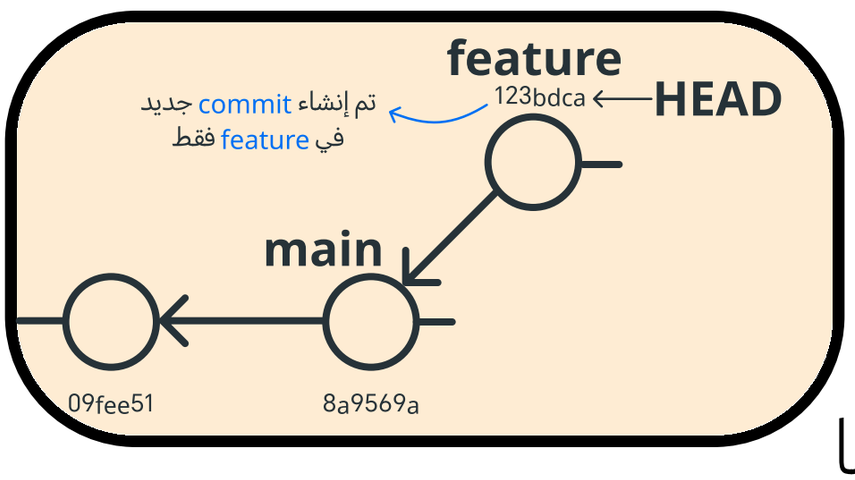
إنشاء commit جديد في feature

نُضيف ملف article_2.txt ثم ننفذ git add

```
> git add article_2.txt
```

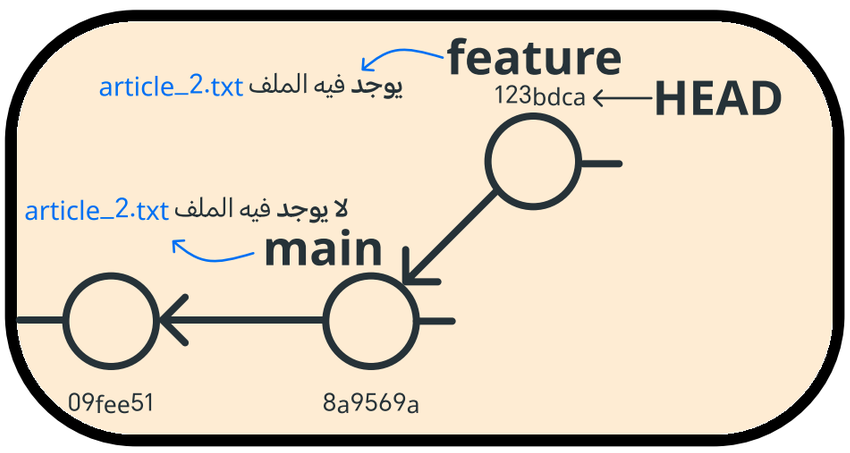
ثم نقوم بتنفيذ git commit

```
> git commit -m "add article 2"
[feature 123bdca] add article 2
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 article_2.txt
```



الـ feature يقف على الـ commit الجديد والـ main مازال على القديم، لم يتأثر بأي شيء

لأن ما نقوم به في feature يكون منفصل تمامًا عن الـ main، ولاحظ أن الـ HEAD أصبح في الـ feature ويشير على الـ commit الذي فيه وليس الذي في الـ main

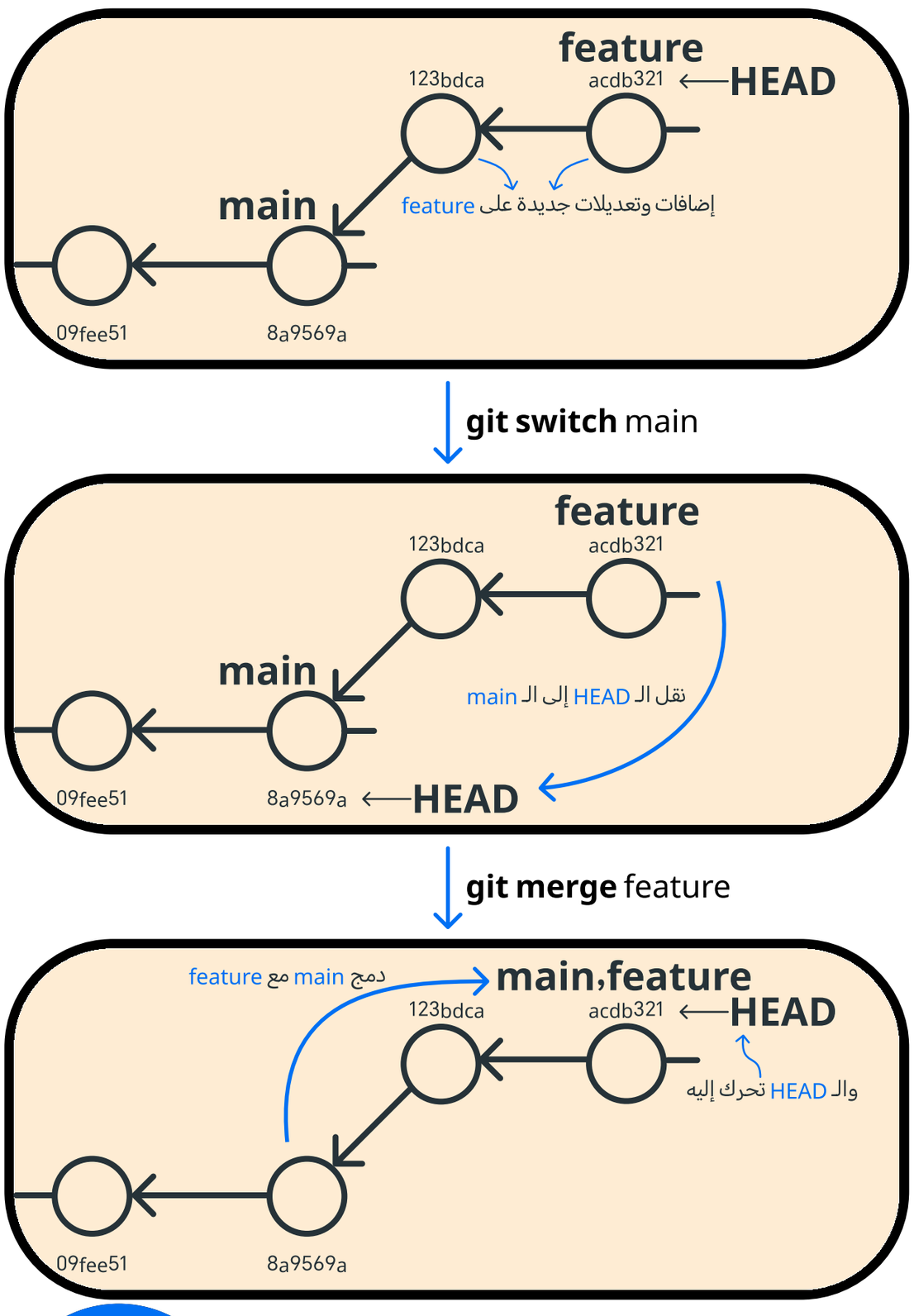


الآن الملف الجديد article_2.txt موجود فقط في feature وليس موجودًا في الـ main، وإن نقلنا الـ HEAD إلى الـ main فلن نجد article_2.txt

Ahmed El-Tabarani
Back-End Developer

git merge

لنفترض أننا قمنا بعمل تعديلات كثيرة في
الـ feature وتم اختبارها جيدًا وأصبح جاهزًا
لوضع التعديلات التي فيه وندمجها في الـ main



سنستخدم git merge، لكن أولًا
علينا أن نذهب إلى الـ main
عن طريق تنفيذ git switch main

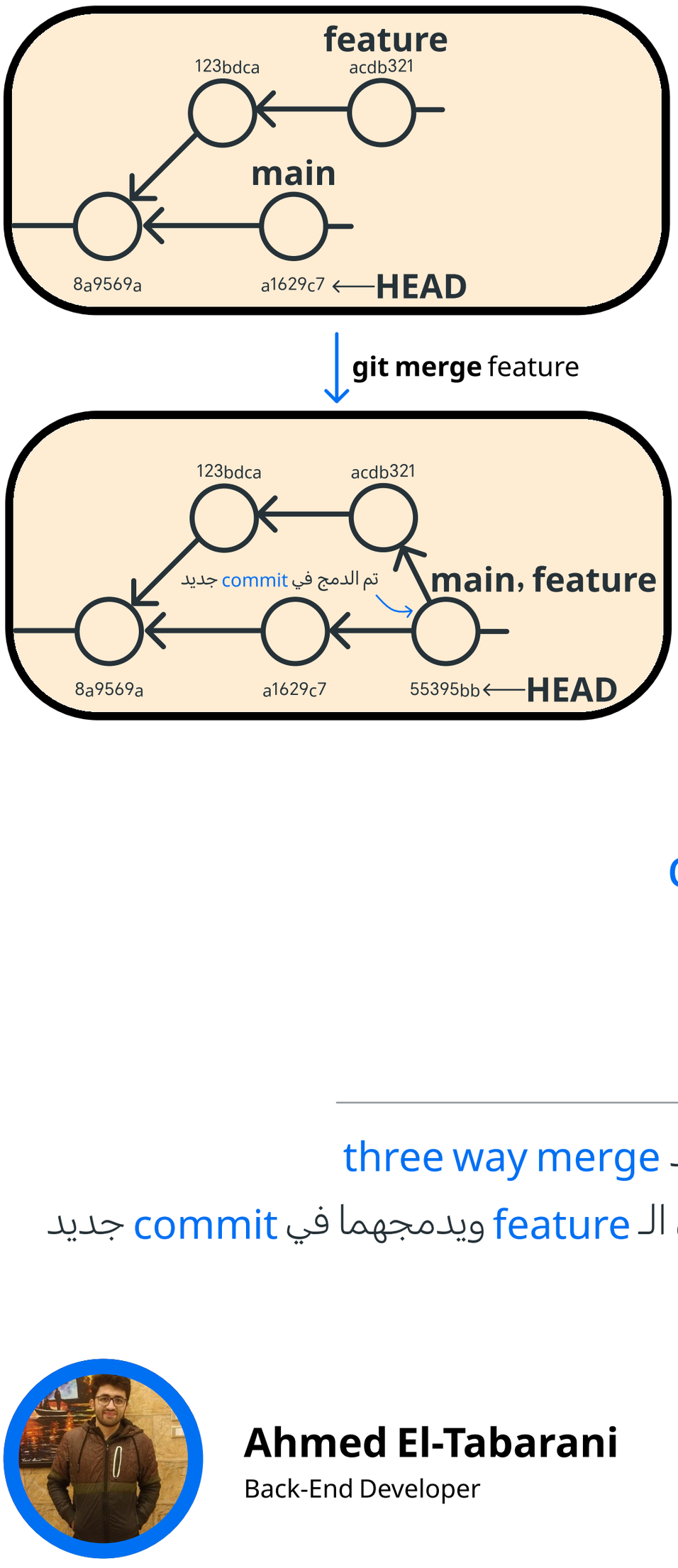
بعد ما نقلنا الـ HEAD عند الـ main
نقوم بتنفيذ أمر الدمج الجميل
git merge feture

هكذا سيتم دمج ما في feature
إلى الـ main

هذا النوع يسمى fast-forward merge
لأنه فقط قام بتحريك الـ main إلى الـ feature

Ahmed El-Tabarani
Back-End Developer

ما فعلناه أشبه بأن الـ main ذهب إلى المستقبل
حيث يوجد الـ feature لا أكثر، وهذا أبسط أنواع
الدمج، ولكن الأمور بالطبع لا تكون دائمًا بسيطة



لاحظ أن هناك فرعين مختلفين يخرجان
من نفس الـ commit الـ 8a9569a

ولاحظ أن feature عنده commit
ليس عند الـ main
وأيضًا الـ main عنده commit
ليس عند الـ feature

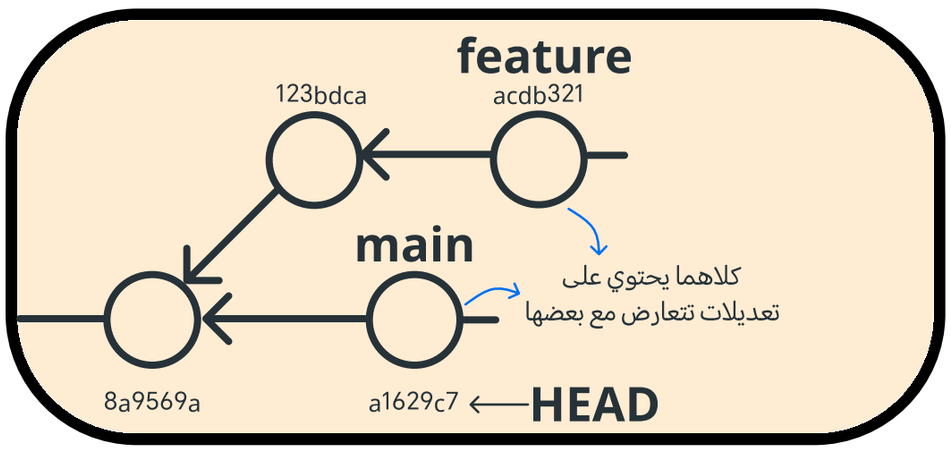
في هذه الحالة عندما نحاول دمج
feature مع main سيتم إنشاء commit
جديد يضم التغيرات الموجودة
في كلا الفرعين

هذا النوع يسمى عملية دمج الأبعاد الميتوفيزاقية .. أقصد three way merge
لأنه يقوم بأخذ الـ commit من الـ main والـ commit من الـ feature ويدمجهما في commit جديد

Ahmed El-Tabarani
Back-End Developer

مشكلة الـ Merge Conflict

تخيل أنك في الـ feature قمت بتعديل
الملف article_1.txt ثم عملت commit
ثم في الـ main قمت بتعديل نفس الملف وعملت commit



هذه التعديلات قد تكون تتعارض مع بعضها
البعض وهذا وارد وشائع جدًا

وعندما تحاول دمج التعديلات الموجودة
في الـ feature والـ main
ستظهر مشكلة التعارض

```
> git merge feature
Auto-merging article_1.txt
CONFLICT (content): Merge conflict in article_1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

ستلاحظ أنه يخبرك بأن الـ Git يحتار ما بين التعديلات التي تتعارض مع بعضها
ولا يعرف هل يختار التي على الـ feature أم التي على الـ main

لذا يقول لك أن تحل المشكلة بنفسك بشكل يدوي ثم بعد انتهائك تقوم
بعمل git add للملفات التي أصلحت فيها التعارض ثم git commit

Ahmed El-Tabarani
Back-End Developer

عندما تفتح الملف التي به التعارض ستجد أن الـ Git يقوم
بتقسيم الملف ويعرض لك التعديلات المتعارضة بهذا الشكل

```
<<<<<<< HEAD
update article 1 in main
=======
update article 1 in feature
>>>>>>> feature
```

ويقول لك Git أن القسم الذي في الأعلى هي
التعديلات التي في الـ HEAD أي المكان الذي
أنت فيه الآن وهو الـ main
والقسم الذي في الأسفل هي التعديلات التي كانت في الـ feature

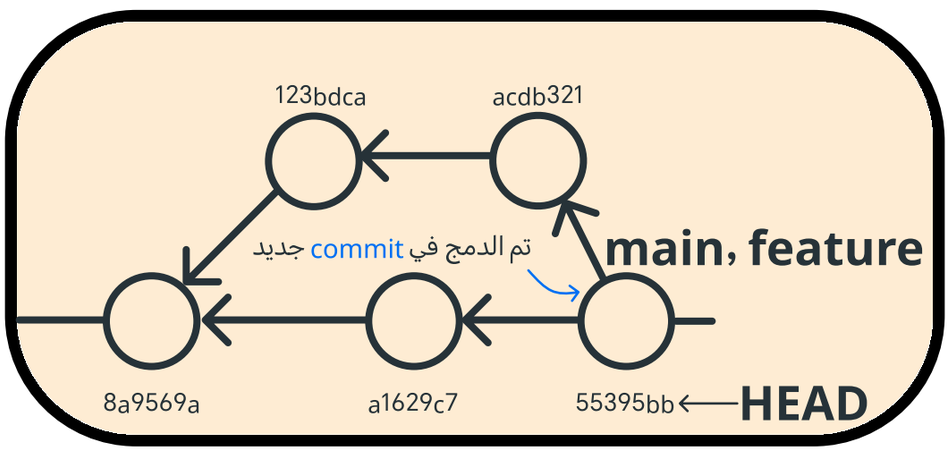
بعد ما تقوم بحل كل التعارضات وتختار تقوم بعمل git add ثم git commit

```
> git add article_.txt

> git commit -m "merge feature into main, and fix the merge conflict"
[main 55395bb] merge feature into main, and fix the merge conflict
```

الآن تم حل التعارض بنجاح
وتم دمج الفرعين في commit جديد

تم الدمج تلقائيًا بعد الـ commit



مشكلة الـ merge conflict تحدث بكثرة
خصوصًا عندما تكون تتعامل مع Remote Repository ضمن فريق
وحلها يتضمن تدخل الشخصين المسؤولين عن التعارض
ليتم اختيار ومناقشة التعديلات المتعارضة بشكل يدوي

Ahmed El-Tabarani
Back-End Developer

خاتمة!

الآن أصبح لديك معرفة جيدة عن مفهوم
الـ Branch في الـ Git

حيث أنها تساعدنا عن إنشاء عدة نسخ من المشروع
تقسم العمل ضمن الفريق بشكل مستقل

بالطبع هذا مجرد ملخص بسيط لا أكثر
ويوجد مقالة على مدونتي استفيض قليلًا في الشرح
بشكل عملي، يمكنكم تفقد المقالة من هنا:

tabarani.tk/articles/git-with-branches

Ahmed El-Tabarani
Back-End Developer

أنا أحمد الطبراني، مهندس برمجيات SWE 😁

مبرمج متخصص في عالم الـ Backend ⚙️💻

أحب دائمًا أن أُشارك معرفتي المتواضعة مع الآخرين لعله عسى أن يستفيد شخص ما بما أكتبه 🤗
أكتب بعض المقالات عن أشياء مختلفة في عالم البرمجة، أرجوا أن تستفيدوا وتستمتعوا 😉

تابعني علي لينكدإن او علي مدونتي الشخصية