https://simplifyCpp.org

Git

للمبتدئين

إعداد : أيمن الحراكي

Git

تعليمي للمبتدئين

إعداد : أيمن الحراكي

29 ديسمبر 2025

المحتويات

مقدمة المؤلف 7

1 المفاهيم الأساسية 9

- ما هو Git ولماذا نستخدمه ؟ 9

1 لماذا نستخدم Git ؟ 9

2 خلاصة: 10

- ما الفرق بين Git و GitHub ؟ 11

1 Git: 11

2 GitHub: 11

3 العلاقة بينهما: 11

- ما الفرق بين Local و Remote ؟ 12

1 Local (محلي): 12

2 Remote (بعيد): 12

3 مثال عملي: 12

- ما معنى Version Control ؟ 13

1 ماذا يفعل هذا النظام؟ 13

2 لماذا هو مهم؟ 13

2 البداية العملية 15

- تثبيت Git على النظام (Windows / Linux / macOS) 15

1 على Windows: 15

2 على Linux (Debian/Ubuntu): 15

3 على macOS: 16

- إعداد Git لأول مرة (الاسم والبريد الإلكتروني) 17
  1 الخطوتان الأساسيتان: 17
  2 ماذا تعني `--global`؟ 17
  3 التحقق من الإعدادات: 17
- إنشاء مشروع جديد محلي باستخدام Git 18
  1 الخطوات: 18
  2 ملاحظة: 18
- أوامر Git الأساسية: `init`, `add`, `commit`, `status` 19
  1 `git init` 19
  2 `git add` 19
  3 `git commit` 19
  4 `git status` 20
  5 مثال بسيط متكامل: 20

3 الربط مع GitHub 21
- إنشاء مستودع GitHub جديد 21
  1 خطوات إنشاء المستودع: 21
  2 بعد الإنشاء: 22
- ربط المشروع المحلي بالمستودع البعيد 23
  1 لماذا نقوم بذلك؟ 23
  2 الأمر المستخدم: 23
  3 التحقق من الربط: 23
- رفع المشروع لأول مرة (`push`) 24
  1 الخطوة الأولى: التأكد من وجود commit 24
  2 الخطوة الثانية: رفع المشروع باستخدام `push` 24
  3 بعد الرفع: 24
- تحديث المستودع بالتعديلات الجديدة (`commit` ثم `push`) 26
  1 الخطوة 1: حفظ التعديلات محليًا (`commit`) 26
  2 الخطوة 2: رفع التعديلات إلى GitHub (`push`) 26

3 ملخص العملية بعد أي تعديل: 27
- جلب التعديلات من GitHub (pull) 28
1 الأمر المستخدم: 28
2 ماذا يحدث عند تنفيذ الأمر؟ 28
3 متى تستخدم pull ؟ 28

4 تنظيم العمل المتقدم 31
- ما هو .gitignore ولماذا نستخدمه ؟ 31
1 ما هو .gitignore ؟ 31
2 أمثلة على محتوى .gitignore: 32
3 كيف يعمل؟ 32
- إنشاء فروع والعمل عليها (branch, checkout) 33
- رفع الفرع إلى GitHub والعمل الجماعي 35
1 رفع الفرع إلى GitHub: 35
2 ما الذي يحدث بعد الرفع؟ 35
3 العمل الجماعي على الفرع: 36
- عرض سجل التعديلات (log) ومقارنة الفروقات (diff) 37
1 أولاً: عرض سجل التعديلات باستخدام log 37
2 تنسيقات إضافية مفيدة: 37
3 ثانيًا: مقارنة الفروقات باستخدام diff 38
4 استخدام عملي: 38

5 التعامل مع المشاكل 39
- أخطاء شائعة عند استخدام Git 39
- كيفية استخدام Git Token و SSH 41
1 أولاً: استخدام Git Token (HTTPS) 41
2 ثانيًا: استخدام مفتاح SSH 42
3 أيهما أفضل للمبتدئ؟ 43

6 مراجعة عامة 45

- أوامر Git المهمة للمراجعة السريعة 45

1 تهيئة المشروع: 45

2 تتبع الملفات وحفظ التعديلات: 45

3 رفع التعديلات إلى :GitHub 46

4 جلب التعديلات من :GitHub 46

5 الفروع: 46

6 سجل التعديلات: 47

7 مقارنة التعديلات: 47

8 حالة المشروع: 47

- نموذج جاهز لملف `.gitignore` (C++, Python, Node.js) 48
- نصائح للمبتدئين 50
- مصادر مجانية لتعلم Git (كتب، فيديوهات، مواقع) 51

7 معلومات إضافية مهمة 53

- مقدمة سريعة عن GitHub Desktop أو VS Code Git GUI 53
- مقدمة عن Git Flow للمشاريع الجماعية 54
- كيفية استعادة ملفات أو حذف commit 56

الملاحق 59

الملحق 1: جدول مصطلحات Git للمبتدئين 59

الملحق 2: أوامر Git السريعة حسب الاستخدام 60

الملحق 3: أمثلة على `.gitignore` لمشاريع شائعة 62

الملحق 4: خطوات العمل المعتادة في Git 63

الملحق 5: مصادر إضافية لتعلم Git عمليًا (دون روابط) 64

فهرس المراجع 65

كتب ودلائل رسمية 65
الوثائق الرسمية 65
دورات ومصادر تعليمية مرئية 65
مواقع ومقالات تعليمية 66
أدوات مساعدة وتمثيل بصري 66

مقدمة المؤلف

في بدايات تعلمي للبرمجة قبل أكثر من اربعين عامًا، لم تكن أدوات إدارة الإصدارات منتشرة كما هي اليوم. كنا نحتفظ بنسخ الملفات يدويًا ونُضيف تواريخ في أسماء المجلدات، ونتمنى ألاّ نضيع شيئًا مهمًا في الطريق. ثم جاء Git، وغير هذا المفهوم تمامًا.

اليوم، لا يمكن لأي مبرمج جاد أن يتجاهل Git.

سواء كنت تعمل وحدك أو ضمن فريق، فإن Git هو الأداة التي تمنحك الأمان، التحكم، والمرونة في إدارة مشروعك. فهو يسجّل كل خطوة، يتيح لك الرجوع في أي لحظة، ويساعدك على العمل بثقة تامة.

قد يبدو Git في البداية مخيفًا أو معقدًا، خاصة مع كثرة الأوامر والمفاهيم. لكن الحقيقة هي أن Git سهل جدًا إذا بدأته بالشكل الصحيح. ومع القليل من التمرين، ستكتشف أنه رفيقك اليومي في كل مشروع، مهما كان حجمه.

هذا الكتيب كتبته خصيصًا للمبرمج العربي المبتدئ، بأسلوب مباشر وسهل، دون مصطلحات مربكة أو شروحات نظرية معقدة. ستبدأ من الصفر، وتتعلم كيف تنشئ مشروعًا، تحفظ التعديلات، تتعامل مع GitHub، وتتجاوز المشاكل الشائعة بثقة.

لا تحتاج إلى معرفة سابقة، فقط افتح الكتيب، وابدأ خطوة خطوة، وستجد أن Git أبسط مما كنت تتخيل.

أتمنى أن يكون هذا الدليل فاتحة خير لك في مشوارك مع أدوات البرمجة الحديثة، وأن يكون دافعًا للاستمرار والتطور.

أيمن الحراكي

مطور نظم وخبير في البرمجة بلغة C++ منذ 1991

المملكة العربية السعودية -- يوليو 2025

فصل 1

المفاهيم الأساسية

1ما هو **Git** ولماذا نستخدمه ؟

Git هو برنامج لإدارة التعديلات التي تجريها على ملفات مشروعك البرمجي بمرور الوقت، ويُعرف بنظام "إدارة نسخ" أو **Version Control System**.

بمعنى آخر، Git يقوم بتسجيل كل تغيير تقوم به على الملفات (مثل الأكواد البرمجية)، ويتيح لك العودة إلى أي نقطة سابقة في تاريخ المشروع، أو التفرع إلى نسخة جديدة من المشروع والعمل عليها بشكل مستقل.

1لماذا نستخدم **Git** ؟

1. تتبع التعديلات: يمكنك معرفة من غيّر ماذا ومتى.
2. الرجوع للنسخ السابقة: إن حصل خطأ، يمكنك العودة إلى نسخة سابقة من الملفات.
3. العمل الجماعي: يتيح Git للمبرمجين العمل معًا على نفس المشروع دون أن يتعارض عملهم.
4. تجربة أفكار جديدة: يمكنك إنشاء نسخة فرعية من المشروع (فرع) وتجربة فكرة جديدة دون التأثير على النسخة الأساسية.
5. التكامل مع خدمات مثل :GitHub يمكنك تخزين مشروعك على الإنترنت ومشاركته أو نسخه والعمل عليه من أي مكان.

2 خلاصة:

Git أداة قوية وضرورية لكل مبرمج، سواء كنت تعمل بمفردك أو ضمن فريق، لأنها تمنحك تحكمًا كاملاً في تطور مشروعك وتاريخه البرمجي.

2 ما الفرق بين Git و GitHub ؟

رغم أن الاسمين متشابهان، إلا أن Git و GitHub هما شيئان مختلفان:

1: Git

- هو برنامج يُثبّت على جهازك.
- يعمل محليًا (على جهازك الشخصي).
- يُستخدم لتتبع التعديلات والاحتفاظ بنسخ من ملفات المشروع.
- لا يحتاج إلى إنترنت لكي يعمل.

2: GitHub

- هو منصة على الإنترنت تستضيف مستودعات .Git
- تُمكّنك من حفظ نسخة من مشروعك في السحابة.
- يُستخدم لمشاركة المشاريع مع الآخرين والعمل الجماعي.
- يتطلب الاتصال بالإنترنت.

3 العلاقة بينهما:

تستخدم Git لتتبع التعديلات، ثم تستخدم GitHub لحفظ نسخة من مشروعك على الإنترنت، مما يسمح لك بالوصول إليه من أي مكان ومشاركته مع زملائك.

3 ما الفرق بين Local و Remote ؟

عند استخدام Git، ستتعامل غالبًا مع مصطلحين أساسيين:

1 Local (محلي):

- يعني العمل على مشروعك من جهازك الشخصي فقط.
- كل الملفات وتعديلات Git محفوظة داخل جهازك.
- يمكنك إنشاء المشروع، إضافة الملفات، حفظ التعديلات، ومراجعة السجل بدون الحاجة إلى الإنترنت.

2 Remote (بعيد):

- يعني وجود نسخة من مشروعك على خادم أو منصة عبر الإنترنت مثل GitHub.
- يُستخدم لمشاركة المشروع، التعاون مع فريق، أو الاحتفاظ بنسخة احتياطية.
- يتطلب الاتصال بالإنترنت للتحديث أو المزامنة.

3 مثال عملي:

- تبدأ مشروعك باستخدام Git على جهازك (local).
- ثم ترفع نسخة منه إلى GitHub (remote).
- بعد ذلك، يمكنك إرسال التعديلات الجديدة من جهازك إلى GitHub أو استلام التعديلات من هناك.

4 ما معنى **Version Control** ؟

Version Control أو "إدارة الإصدارات" هو نظام يُستخدم لتتبع التعديلات التي تتم على الملفات، وخاصة في المشاريع البرمجية.

1 ماذا يفعل هذا النظام؟

- يسجّل كل تغيير تقوم به في ملفات المشروع.
- يسمح لك بالعودة إلى أي نسخة سابقة من الملفات.
- يوضح من أجرى التغيير ومتى ولماذا.
- يسهل مقارنة التعديلات بين الإصدارات المختلفة.
- يتيح العمل على أكثر من نسخة أو فرع من المشروع في وقت واحد.

2 لماذا هو مهم؟

عند العمل على مشروع يتطور باستمرار، ستحتاج إلى وسيلة تحفظ كل مراحل العمل، وتساعدك على التراجع عند حدوث أخطاء، أو دمج التعديلات من عدة أشخاص.

Git هو واحد من أشهر أنظمة إدارة الإصدارات، ويُستخدم على نطاق واسع بسبب قوته وسرعته ومرونته.

فصل 2

البداية العملية

1تثبيت Git على النظام (Windows / Linux / macOS)

قبل أن تبدأ باستخدام ،Git يجب أولاً تثبيته على جهازك. تختلف طريقة التثبيت حسب نظام التشغيل.

1 على :Windows

- قم بتنزيل مثبت Git من الموقع الرسمي.
- بعد تشغيل الملف، اتبع الخطوات الافتراضية في معالج التثبيت.
- بعد انتهاء التثبيت، يمكنك فتح "Git Bash" أو "Command Prompt" وكتابة الأمر:

```
git --version
```

إذا ظهرت لك نسخة ،Git فهذا يعني أن التثبيت تم بنجاح.

2 على **Linux (Debian/Ubuntu)**:

- افتح الطرفية، واكتب الأوامر التالية:

```
sudo apt update
sudo apt install git
```

- للتحقق من التثبيت:

```
git --version
```

3 على macOS:

- إذا كان لديك Homebrew، فاستخدم الأمر:

```
brew install git
```

- أو تأكد من وجود Git مسبقًا بكتابة:

```
git --version
```

بعد التثبيت، تكون جاهزًا لبدء استخدام Git محليًا على جهازك.

2 إعداد Git لأول مرة (الاسم والبريد الإلكتروني)

بعد تثبيت Git، يجب إعداد بياناتك الأساسية حتى يتم تسجيل كل تعديل تقوم به باسمك في سجل المشروع.

1 الخطوتان الأساسيتان:

1. تحديد الاسم:

```
git config --global user.name "Full Name"
```

2. تحديد البريد الإلكتروني:

```
git config --global user.email
 ↪  "example@email.com"
```

2 ماذا تعني `--global`؟

تعني أن هذا الإعداد سيُستخدم لجميع المشاريع التي تنشئها على هذا الجهاز.
يمكنك لاحقًا تغييره أو تخصيص إعداد مختلف لمشروع معين.

3 التحقق من الإعدادات:

لمراجعة الإعدادات التي قمت بإدخالها:

```
git config --list
```

سيعرض هذا الأمر جميع الإعدادات العامة في Git، بما في ذلك اسمك وبريدك الإلكتروني.

3 إنشاء مشروع جديد محلي باستخدام Git

بعد تثبيت Git وإعداد اسمك وبريدك الإلكتروني، يمكنك الآن بدء مشروع جديد على جهازك باستخدام Git لتتبع التعديلات.

1 الخطوات:

1. إنشاء مجلد للمشروع (إذا لم يكن موجوداً):

```
mkdir myProject
cd myProject
```

2. تهيئة Git داخل المجلد:

```
git init
```

هذا الأمر ينشئ مجلداً مخفيًا باسم `git` . داخل مجلد المشروع. هذا المجلد يحتوي على جميع ملفات التتبع الخاصة بـ Git، وهو ما يجعل هذا المجلد مشروعًا Git.

2 ملاحظة:

في هذه المرحلة، لم يتم تتبع أي ملف بعد. أنت فقط أخبرت Git أن هذا المجلد سيكون مشروعًا يديره Git.

يمكنك الآن البدء بإضافة ملفات المشروع ثم تتبعها بالأوامر المناسبة مثل `add` و `commit` في الأقسام القادمة.

4 أوامر Git الأساسية: **`status` ,`commit` ,`add` ,`init`**

في بداية استخدامك لـ Git، هناك مجموعة من الأوامر الأساسية التي تحتاج إلى معرفتها لإدارة التعديلات في مشروعك.

1 **`git init`**

يستخدم لتهيئة مجلد كمشروع Git:

```
git init
```

يُنشئ مجلدًا مخفيًا `.git` داخل مجلد المشروع لتخزين بيانات التتبع.

2 **`git add`**

يستخدم لإعلام Git بأن هناك ملفات جديدة أو معدّلة يجب تتبعها.
لإضافة ملف معين:

```
git add FileName
```

لإضافة جميع الملفات:

```
git add .
```

3 **`git commit`**

يُستخدم لحفظ التغييرات التي تم إضافتها بالأمر `add` في سجل المشروع، ويجب إرفاق رسالة توضح محتوى التعديل:

```
git commit -m "Describe Update message"
```

4 `git status`

يعرض حالة المشروع في اللحظة الحالية:

- ما الملفات الجديدة؟
- ما الملفات التي تم تعديلها؟
- ما الذي أُضيف للتتبع؟ وما الذي لم يُضف بعد؟

```
git status
```

5 مثال بسيط متكامل:

```
git init
echo "Hello World" < hello.txt
git add hello.txt
git commit -m "Welcome add file"
```

هذه الأوامر تمثّل الأساس اليومي للعمل مع Git، وهي بداية التعامل الحقيقي مع المشروع كمشروع قابل للتتبع.

فصل 3

الربط مع GitHub

1إنشاء مستودع GitHub جديد

بعد إنشاء مشروعك المحلي باستخدام ،Git ستحتاج إلى رفعه على الإنترنت لتخزينه، مشاركته، أو العمل عليه من أجهزة أخرى.
لذلك، يجب أولاً إنشاء مستودع (Repository) على .GitHub

1 خطوات إنشاء المستودع:

1. سجّل الدخول إلى حسابك على .GitHub
2. من الصفحة الرئيسية، اضغط على زر "New" "repository أو من القائمة "Repositories" اختر ."New"
3. أدخل البيانات الأساسية للمستودع:
   - **Repository name**: اسم المشروع.
   - Description: وصف اختياري.
   - **Public / Private**: اختر ما إذا كنت تريد المشروع متاحًا للجميع أو خاصًا.
4. لا تقم بتفعيل أي خيار لإنشاء ملفات تلقائية (مثل README أو ،(gitignore. لأنك ستربط هذا المستودع لاحقًا بمشروع Git الموجود على جهازك.
5. اضغط على زر **Create repository** في الأسفل.

2 بعد الإنشاء:

سيعرض لك GitHub رابط المستودع (HTTPS or SSH)، ستستخدم هذا الرابط لربط مشروعك المحلي بالمستودع البعيد.

2 ربط المشروع المحلي بالمستودع البعيد

بعد إنشاء مشروع محلي على جهازك باستخدام Git، وإنشاء مستودع جديد على GitHub، تأتي الخطوة التالية: ربط المشروع المحلي بالمستودع البعيد.

1 لماذا نقوم بذلك؟

حتى يتمكن Git من معرفة العنوان الذي يجب أن يرسل إليه التعديلات عند استخدام الأمر `push` أو يستقبل منه التحديثات عند استخدام `pull`.

2 الأمر المستخدم:

```
git remote add origin
 ↪  https://github.com/RepoName/UserName.git
```

- `remote add`: تعني إضافة مستودع بعيد.
- `origin`: هو الاسم الافتراضي للمستودع البعيد (يمكن تغييره لكن يفضل تركه كما هو).
- الرابط: هو عنوان المستودع على GitHub باستخدام HTTPS أو SSH.

3 التحقق من الربط:

للتأكد من أن الربط تم بنجاح:

```
git remote -v
```

سيعرض هذا الأمر عنوان المستودع البعيد المرتبط بمشروعك.
بعد هذه الخطوة، يصبح مشروعك المحلي جاهزًا لرفع التعديلات إلى GitHub.

3 رفع المشروع لأول مرة (**push**)

بعد أن قمت بربط مشروع Git المحلي بالمستودع البعيد على GitHub، يمكنك الآن رفع الملفات إلى GitHub لأول مرة.

1 الخطوة الأولى: التأكد من وجود commit

قبل رفع المشروع، تأكد أنك نفّذت على الأقل عملية `commit` واحدة:

```
git add .
git commit -m "First Copy of project"
```

2 الخطوة الثانية: رفع المشروع باستخدام **push**

```
git push -u origin main
```

- `push`: تعني إرسال التعديلات إلى المستودع البعيد.
- `-u`: تعني تعيين `main origin` كمسار افتراضي للرفع لاحقًا.
- `origin`: هو اسم المستودع البعيد.
- `main`: هو اسم الفرع (قد يكون `master` إذا أنشأته بهذا الاسم).

3 بعد الرفع:

- ستظهر ملفاتك الآن على GitHub في المستودع الذي أنشأته.

- يمكنك في المرات القادمة استخدام الأمر `push git` فقط دون الحاجة لتكرار `-u main origin`.

4تحديث المستودع بالتعديلات الجديدة (**commit** ثم **push**)

عندما تُجري تغييرات على ملفات مشروعك (مثل تعديل كود أو إنشاء ملف جديد)، فإن Git لا يرفع هذه التعديلات تلقائيًا إلى GitHub.
بل عليك اتباع خطوتين أساسيتين في كل مرة:

1 الخطوة 1: حفظ التعديلات محليًا (**commit**)

أولاً، أضف الملفات المعدّلة إلى منطقة التتبع:

```
git add .
```

ثم احفظ التغييرات في سجل Git المحلي:

```
git commit -m "Describe update you made it"
```

- `add .`: يضيف جميع الملفات المعدّلة والجديدة.
- `commit -m`: يحفظ التغييرات مع كتابة رسالة توضح محتوى التعديل.

2 الخطوة 2: رفع التعديلات إلى GitHub (**push**)

بعد تنفيذ `commit`، استخدم الأمر التالي لرفع التغييرات إلى المستودع البعيد:

```
git push
```

طالما كنت قد استخدمت `main origin -u` سابقًا، فلن تحتاج لتحديد المسار مرة أخرى.

3 ملخص العملية بعد أي تعديل:

1. عدّل ملفات المشروع.
2. `git add .`
3. `git commit -m "update message"`
4. `git push`

5 جلب التعديلات من GitHub (`pull`)

عندما تعمل على مشروع مخزَّن على GitHub، قد تحتاج أحيانًا إلى جلب آخر التعديلات التي أُجريت على المستودع البعيد، سواء كانت من زملائك في الفريق أو من جهاز آخر تعمل عليه. للقيام بذلك، تستخدم الأمر `pull`.

1 الأمر المستخدم:

```
git pull origin main
```

- `pull`: يجلب التحديثات من المستودع البعيد.
- `origin`: اسم المستودع البعيد.
- `main`: اسم الفرع الذي تريد تحديثه (قد يكون `master` في بعض المشاريع).

2 ماذا يحدث عند تنفيذ الأمر؟

- Git يتصل بالمستودع البعيد (GitHub).
- يتحقق من وجود تغييرات غير موجودة لديك.
- إذا وُجدت تغييرات، يتم دمجها في ملفات مشروعك الحالي.

3 متى تستخدم `pull`؟

- قبل أن تبدأ العمل، لتتأكد أن لديك آخر نسخة من المشروع.
- بعد أن يقوم شخص آخر برفع تعديلات جديدة.
- قبل رفع تغييراتك، لتفادي تعارض النسخ (conflicts).

إذا حدث تعارض بين تعديلاتك وتعديلات المستودع البعيد، سيُطلب منك حل هذا التعارض يدويًا، وهي مهارة تُكتسب بالممارسة وستأتي لاحقًا في الكتيب.

فصل 4

تنظيم العمل المتقدم

1ما هو **`.gitignore`** ولماذا نستخدمه ؟

عند العمل على مشروع باستخدام Git، قد توجد ملفات لا ترغب في تتبعها أو رفعها إلى GitHub، مثل:

- ملفات مؤقتة.
- مجلدات النظام.
- ملفات الإعدادات الشخصية.
- ملفات البناء .(build)
- ملفات البيئة (مثل ملفات كلمات المرور أو المفاتيح السرية).

هنا تأتي فائدة الملف `.gitignore`.

1 ما هو **`.gitignore`** ؟

هو ملف تضعه داخل مجلد مشروعك، وتكتب فيه أسماء الملفات أو المجلدات التي لا تريد أن يتتبعها Git أو يرفعها إلى المستودع.

2 أمثلة على محتوى **.gitignore**:

```
*.log
.env
__pycache__/
node_modules/
build/
*.exe
```

- `*.log`: يتجاهل جميع الملفات التي تنتهي بامتداد log.
- `.env`: يتجاهل الملف الذي يحتوي على متغيرات البيئة
- `node_modules/`: يتجاهل مجلد مكتبات Node.js

3 كيف يعمل؟

بمجرد إضافة ملف إلى `.gitignore`، سيتوقف Git عن تتبعه، بشرط ألا يكون قد تم تتبعه مسبقًا.
إذا كان الملف مضافًا من قبل، يجب حذفه من التتبع باستخدام:

```
git rm --cached FileName
```

باستخدام `.gitignore`، تحافظ على نظافة مستودعك، وتتجنب رفع ملفات لا تهم بقية الفريق أو لا يجب مشاركتها.

2 إنشاء فروع والعمل عليها (**checkout ,branch**)

في Git، يُعتبر الفرع (Branch) وسيلة لتجربة أفكار جديدة أو تنفيذ ميزات جديدة دون التأثير على النسخة الأساسية من المشروع.
الفرع الرئيسي في أغلب المشاريع يُسمى `main` أو `master`.

1. لماذا نستخدم الفروع؟

- لتطوير ميزة جديدة بشكل مستقل.
- لإصلاح مشكلة دون التأثير على النسخة الثابتة.
- لتجربة أكواد أو تغييرات قبل دمجها في المشروع الأساسي.

2. إنشاء فرع جديد:

```
git branch BranchName
```

مثال:

```
git branch login-feature
```

3. التبديل إلى فرع آخر:

```
git checkout BranchName
```

مثال:

```
git checkout login-feature
```

4. اختصار لإنشاء فرع والتبديل إليه مباشرة:

```
git checkout -b BranchName
```

مثال:

```
git checkout -b login-feature
```

5. عرض جميع الفروع:

```
git branch
```

الفرع الذي تعمل عليه حاليًا يظهر بعلامة النجمة *.

عند الانتهاء من العمل على الفرع، يمكنك رفعه إلى GitHub أو دمجه في الفرع الرئيسي، وسنشرح ذلك في الأقسام التالية.

3رفع الفرع إلى GitHub والعمل الجماعي

بعد إنشاء فرع جديد والعمل عليه محليًا، قد تحتاج إلى رفع هذا الفرع إلى GitHub لمشاركته مع زملائك أو لمراجعته لاحقًا.

1رفع الفرع إلى :GitHub

```
git push -u origin BranchName
```

مثال:

```
git push -u origin login-feature
```

- `push`: رفع التعديلات إلى .GitHub
- `-u`: تعني ربط هذا الفرع بالمستودع البعيد ليتم التعرف عليه تلقائيًا لاحقًا.
- `origin`: هو اسم المستودع البعيد.
- `login-feature`: هو اسم الفرع المحلي الذي تريد رفعه.

2 ما الذي يحدث بعد الرفع؟

- يتم إنشاء فرع بنفس الاسم على .GitHub
- يستطيع أي عضو في الفريق سحب هذا الفرع والعمل عليه.
- يمكن فتح "طلب دمج" (Pull Request) لاحقًا لدمج التعديلات مع الفرع الرئيسي.

3 العمل الجماعي على الفرع:

أي شخص لديه صلاحية يمكنه:

1. سحب الفرع:

```
git checkout -b login-feature
 ↪  origin/login-feature
```

2. إجراء تعديلاته.

3. رفعها باستخدام:

```
git push
```

رفع الفروع والعمل عليها بشكل منفصل يُعد من أفضل ممارسات Git، لأنه يسمح بتنظيم العمل وتجنّب التعديلات العشوائية على النسخة الأساسية من المشروع.

4 عرض سجل التعديلات (`log`) ومقارنة الفروقات (`diff`)

عند العمل على مشروع باستخدام Git، من المهم أن تتمكن من مراجعة تاريخ التعديلات ومعرفة الفروقات بين الإصدارات المختلفة.

1 أولاً: عرض سجل التعديلات باستخدام `log`

```
git log
```

يعرض هذا الأمر قائمة بكل العمليات التي تم تنفيذها باستخدام commit، وتشمل:

- رقم المعرّف (commit ID).
- اسم الكاتب.
- تاريخ التنفيذ.
- رسالة التعديل.

2 تنسيقات إضافية مفيدة:

عرض سجل مختصر:

```
git log --oneline
```

عرض سجل فرع محدد:

```
git log BranchName
```

3 ثانيًا: مقارنة الفروقات باستخدام **diff**

1. لرؤية التعديلات غير المحفوظة (بين الملفات الحالية وآخر **commit**):

```
git diff
```

2. لرؤية الفروقات بين ملفين محددين:

```
git diff file1 file2
```

3. لرؤية الفروقات بين نسختين سابقتين:

```
git diff commitID1 commitID2
```

4 استخدام عملي:

قبل تنفيذ `commit`، يمكنك استخدام `git diff` لمراجعة التعديلات بالتفصيل.
بعد تنفيذ عدة `commit`، يمكنك استخدام `git log` لفهم كيف تطوّر المشروع خطوة بخطوة.

فصل 5

التعامل مع المشاكل

1أخطاء شائعة عند استخدام Git

خلال استخدام Git، قد تواجه بعض الأخطاء التي تبدو مقلقة في البداية، لكنها في الغالب سهلة الفهم والحل. إليك بعض الأخطاء الشائعة وكيفية التعامل معها.

1. خطأ: **`fatal: not a git repository`**

   السبب:
   هذا الخطأ يعني أنك تحاول تنفيذ أمر Git في مجلد لا يحتوي على مشروع Git (أي لا يحتوي على مجلد `.git`).

   الحل:
   تأكد أنك داخل مجلد مشروع تم تهيئته بـ `git init`.
   استخدم:

```
cd / Project Path
```

2. خطأ:**`fatal: Authentication failed`**

   السبب:
   يحدث هذا الخطأ عادةً عند محاولة الاتصال بـ GitHub باستخدام كلمة مرور عادية، وهو أمر لم يعد مدعومًا.

الحل:

استخدم "Token" بدلاً من كلمة المرور عند الدفع (`push`) أو السحب (`pull`).

يمكنك إنشاء Token من إعدادات حسابك في GitHub، ثم استخدامه عند طلب بيانات الدخول.

3. خطأ: **`error: failed to push some refs to ...`**

مع رسالة: `non-fast-forward`

السبب:

تحدث هذه المشكلة عندما يكون هناك تحديثات على GitHub لم تقم بجلبها قبل تنفيذ `push`.

الحل:

أولاً، اجلب التحديثات:

```
git pull --rebase origin main
```

ثم أعد تنفيذ:

```
git push
```

4. خطأ: **`Permission denied (publickey)`**

السبب:

يعني هذا الخطأ أن GitHub لا يستطيع التحقق من هويتك باستخدام مفتاح SSH.

الحل:

إما أن تستخدم HTTPS بدل SSH عند الربط، أو:

- أنشئ مفتاح SSH جديد:

```
ssh-keygen -t ed25519 -C
 ↪  "your_email@example.com"
```

- ثم أضف المفتاح إلى حسابك في GitHub.

التعامل مع هذه الأخطاء هو جزء طبيعي من استخدام Git، ومع الممارسة ستصبح حلها سهلاً وسريعًا.

2 كيفية استخدام Git Token و SSH

عند التعامل مع GitHub، تحتاج إلى طريقة موثوقة لتوثيق هويتك عند تنفيذ أوامر مثل `push` و `pull`.
ولأن GitHub لم يعد يسمح باستخدام كلمة المرور العادية، هناك طريقتان شائعتان:

1 أولاً: استخدام Git Token (HTTPS)

- ما هو الـ Token ؟

هو سلسلة من الأحرف تُستخدم بدلاً من كلمة المرور عند الاتصال بـ GitHub عبر HTTPS.

- خطوات الاستخدام:

1. أنشئ Token من إعدادات حسابك في GitHub
(قسم Developer Settings > Personal Access Tokens).
2. اختر الأذونات المناسبة مثل: repo و workflow.
3. احفظ الـ Token في مكان آمن.

- استخدامه:

عند تنفيذ `git push` لأول مرة:

- سيطلب Git اسم المستخدم: اكتب اسم حسابك في GitHub.
- سيطلب كلمة المرور: الصق الـ Token بدلاً من كلمة المرور.

2 ثانيًا: استخدام مفتاح SSH

- ما هو SSH ؟

هو أسلوب آمن للاتصال بين جهازك و GitHub دون الحاجة لإدخال اسم مستخدم وكلمة مرور في كل مرة.

- خطوات إعداد SSH:

1. إنشاء مفتاح SSH على جهازك:

```
ssh-keygen -t ed25519 -C
 ↪  "your_email@example.com"
```

2. نسخ المفتاح العام:

```
cat ~/.ssh/id_ed25519.pub
```

3. إضافة المفتاح إلى حساب GitHub (من قسم SSH Keys).
4. اختبار الاتصال:

```
ssh -T git@github.com
```

- ربط المشروع بـ SSH:

عند ربط مشروعك بـ GitHub، استخدم الرابط من نوع SSH بدل HTTPS:

```
git remote add origin
 ↪  git@github.com:username/repository.git
```

3 أيهما أفضل للمبتدئ؟

- **Git Token**: أسهل في البداية، خاصة عند استخدام HTTPS.
- SSH: أكثر أمانًا وراحة عند العمل اليومي، لأنه لا يطلب إدخال كلمة مرور.

فصل 6

مراجعة عامة

1أوامر Git المهمة للمراجعة السريعة

فيما يلي مجموعة من الأوامر الأساسية التي يحتاجها أي مبتدئ للتعامل مع Git بشكل يومي. هذه الأوامر تمثل أساس الاستخدام العملي لـ Git وتكفي لمعظم المهام في المشاريع الفردية أو الجماعية.

1 تهيئة المشروع:

```
git init
```

تهيئة مجلد كمشروع Git محلي.

2 تتبع الملفات وحفظ التعديلات:

```
git add .
```

إضافة جميع الملفات الجديدة أو المعدّلة إلى قائمة التتبع.

```
git commit -m "Message describe the updates"
```

حفظ التعديلات في سجل Git المحلي.

3 رفع التعديلات إلى GitHub:

```
git push
```

رفع التعديلات من جهازك إلى المستودع البعيد.

```
git push -u origin main
```

رفع أول مرة مع تحديد اسم المستودع والفرع.

4 جلب التعديلات من GitHub:

```
git pull
```

جلب آخر التعديلات من المستودع البعيد.

5 الفروع:

```
git branch
```

عرض الفروع الموجودة.

```
git checkout -b BranchName
```

إنشاء فرع جديد والتبديل إليه.

6 سجل التعديلات:

```
git log
```

عرض سجلات commit.

```
git log --oneline
```

عرض سجل مختصر لكل تعديل.

7 مقارنة التعديلات:

```
git diff
```

عرض الفروقات بين التعديلات الحالية وآخر commit.

8 حالة المشروع:

```
git status
```

عرض حالة الملفات: ما الجديد، وما تم تتبعه، وما لم يُتتبع بعد.

هذه الأوامر هي الأساس اليومي لكل مستخدم لـ Git، ويُفضّل التدرّب عليها عمليًا حتى تصبح مألوفة.

2 نموذج جاهز لملف **.gitignore** (C++, Python, Node.js)

- ما هو **.gitignore**؟

هو ملف نصي تضع فيه أسماء الملفات أو المجلدات التي لا تريد أن يُدرجها Git في التتبع أو يتم رفعها إلى المستودع البعيد.

يختلف محتوى هذا الملف من مشروع لآخر حسب اللغة المستخدمة أو الأدوات التي يتم العمل بها.

- نموذج **.gitignore** لمشاريع C++:

```
# Compiled and build files
*.o
*.obj
*.exe
*.out
*.log
*.a
*.lib
*.dll
*.so

# build files
```

```
build/
bin/
```

- نموذج **.gitignore** لمشاريع Python:

```
# Bytecodes files
*.pyc
*.pyo
__pycache__/

# Virtual data
env/
venv/

# setting files
*.env
.settings/
```

- نموذج **.gitignore** لمشاريع Node.js:

```
# Packages Folders
node_modules/

# Registers files
*.log
```

```
# Environment files
.env
```

- ملاحظات:

  - يمكنك تخصيص الملف حسب مشروعك.
  - يُفضّل إنشاء الملف في جذر مجلد المشروع.
  - Git لن يتجاهل الملفات التي أُضيفت بالفعل للتتبع، حتى لو أدرجتها لاحقًا في `.gitignore`.

استخدام `.gitignore` يُساعد في الحفاظ على نظافة المستودع، ويمنع رفع ملفات غير ضرورية أو حساسة.

3 نصائح للمبتدئين

تعلم Git في البداية قد يبدو معقداً، لكنه يصبح سهلاً مع الممارسة. إليك مجموعة من النصائح التي ستساعدك على استخدام Git بثقة وفعالية:

- لا تحفظ الأوامر، افهمها

  بدلاً من حفظ الأوامر عن ظهر قلب، حاول أن تفهم ما يقوم به كل أمر ولماذا تحتاجه. الفهم سيجعل التعلّم أسرع وأثبت.

- استخدم Git في مشاريعك الحقيقية

  أفضل طريقة للتعلم هي التطبيق العملي. استخدم Git في كل مشروع صغير أو كبير تعمل عليه، ولو كنت تعمل بمفردك.

- نفّذ **`git status`** باستمرار

  هذا الأمر يُطلعك على حالة المشروع في كل لحظة. استخدامه بشكل متكرر يقلل الأخطاء ويوضح لك ما يحدث.

- لا تتردد في استخدام الفروع

استخدم الفروع لتجربة أفكار جديدة دون التأثير على النسخة الأساسية من المشروع. لا تعمل دائمًا على الفرع الرئيسي.

- اكتب رسائل `commit` واضحة

رسالة التعديل يجب أن تصف ما قمت به. تجنب الرسائل الغامضة مثل: "تعديل بسيط" أو "تجربة". الرسائل الواضحة تساعدك مستقبلاً في فهم تاريخ المشروع.

- احفظ نسخة احتياطية على GitHub

حتى لو كنت تعمل وحدك، رفع المشروع إلى GitHub يحميك من فقدان الملفات، ويساعدك في العمل من أجهزة متعددة.

- لا تخف من الأخطاء

ستواجه أخطاء في البداية، وهذا طبيعي. اقرأ الرسالة جيدًا، وجرّب فهم السبب، وغالبًا سيكون الحل بسيطًا.

باختصار: الممارسة المنتظمة، والصبر في الفهم، واستخدام Git في مشاريعك اليومية هي مفاتيح إتقان هذه الأداة المهمة.

4 مصادر مجانية لتعلم Git (كتب، فيديوهات، مواقع)

بعد أن تتعرف على الأساسيات وتبدأ بتطبيقها، قد تحتاج إلى التوسع أكثر، أو مراجعة بعض التفاصيل التي لم تتضح لك. هذه قائمة بمصادر تعليمية مجانية يمكنك الرجوع إليها لتطوير مهاراتك في :Git

- كتب إلكترونية:

- كتاب Git الرسمي (**Pro Git**):

كتاب شامل يغطي Git من الأساسيات إلى المتقدم، متوفر بلغات متعددة منها العربية. مفيد للمبتدئين والمحترفين.

- كتيبات مختصرة للمبتدئين:

توجد كتيبات خفيفة تشرح Git بشكل مصور ومبسط، مناسبة للقراءة السريعة والمراجعة.

- فيديوهات تعليمية:

- دورات على منصات الفيديو:

ابحث عن دورة تعليم Git من الصفر إلى الاحتراف. اختر الدورة التي تستخدم اللغة والأسلوب المناسب لك، سواء كانت عربية أو إنجليزية.

- شروحات عملية قصيرة:

توجد قنوات تعليمية تقدم شروحات قصيرة مخصصة لأوامر Git الأكثر استخداماً، خطوة بخطوة.

- مواقع تدريب تفاعلي:

- منصات تقدم محاكيات :Git

بعض المواقع تتيح لك تجربة Git مباشرة من المتصفح، عبر تمارين تطبيقية وأوامر حقيقية، بدون الحاجة لتثبيت شيء.

- مدونات تقنية ومجتمعات برمجية:

تحتوي على دروس ومقالات تشرح مشاكل وحلول عملية، وتجارب مستخدمين حقيقية.

- نصيحة للاستفادة من المصادر:

ابدأ بالمصدر المناسب لمستواك، ورافق التعلم بالتطبيق العملي على مشروع شخصي. لا تكثر من المصادر دفعة واحدة، بل استوعب كل جزء تدريجياً مع التمرين.

فصل 7

معلومات إضافية مهمة

1مقدمة سريعة عن **GitHub Desktop** أو **VS Code Git GUI**

بعض المبتدئين يواجهون صعوبة في حفظ أوامر Git في الطرفية، لذلك تم توفير أدوات رسومية تساعد على استخدام Git بطريقة مرئية وسهلة. من أبرز هذه الأدوات:

- **GitHub Desktop**

هي أداة مجانية مقدمة من GitHub، وتُستخدم لتسهيل التعامل مع Git دون الحاجة لكتابة أوامر.
يمكنك من خلالها:

- إنشاء مستودع جديد محليًا أو على GitHub.
- تنفيذ `commit`, `push`, `pull` بنقرة واحدة.
- رؤية الفروقات بين الملفات قبل الحفظ.
- إدارة الفروع ودمجها بسهولة.

واجهة GitHub Desktop مناسبة جدًا للمبتدئين لأنها توضح كل خطوة بصريًا وتعرض حالة المشروع بوضوح.

- Git مدمج في **VS Code (Git GUI)**

محرر Visual Studio Code يحتوي على واجهة رسومية مدمجة للتعامل مع Git.
من خلاله يمكنك:

- تتبع حالة الملفات الجديدة والمعدلة.
- تنفيذ `commit` عبر الواجهة.
- إجراء `push` و `pull` دون مغادرة المحرر.
- عرض سجل التعديلات، مقارنة الفروقات، والعمل على الفروع.

مميزاته:

- لا حاجة لأداة خارجية.
- يوفر تكاملاً مباشراً مع GitHub.
- مناسب لمن يحبون الكتابة والبرمجة داخل بيئة واحدة.

- لمن تناسب هذه الأدوات؟

- لمن يفضلون الواجهات الرسومية على الطرفية.
- للمبتدئين الذين لا يزالون يتعلمون أوامر Git الأساسية.
- لمن يعملون على مشاريع صغيرة أو متوسطة الحجم.

استخدام هذه الأدوات لا يغني عن فهم أوامر Git، لكنه يساعدك في تعلمها عمليًا بطريقة مريحة ومرئية.

2 مقدمة عن **Git Flow** للمشاريع الجماعية

عند العمل ضمن فريق على مشروع برمجي، يصبح من الضروري وجود نظام واضح لإدارة الفروع وتنظيم سير العمل باستخدام Git. هنا يأتي دور **Git Flow**.

- ما هو **Git Flow**؟

Git Flow هو نموذج تنظيمي لإدارة الفروع في مشروع Git، يحدد كيف يتم:

- تطوير الميزات الجديدة.

- إصلاح الأخطاء.
- تجهيز النسخ الجاهزة للإصدار.
- إدارة التعديلات الطارئة.

- الفروع الأساسية في **Git Flow:**

1. main

يحتوي على النسخة الجاهزة للإصدار والتي تُنشر للمستخدمين.

2. develop

هو الفرع الرئيسي لتطوير المشروع، ويحتوي على آخر التعديلات المجربة.

- الفروع المؤقتة:

1. feature/

لإنشاء ميزات جديدة، تنشأ من `develop` وتُدمج فيه بعد الانتهاء.

2. release/

لتحضير نسخة جديدة للإصدار، تنشأ من `develop` وتُدمج في `main` بعد اكتمالها.

3. hotfix/

لإصلاح مشكلات عاجلة في `main`. تُنشأ من `main` وتُدمج فيه وفي `develop`.

- مثال عملي مبسّط:

- تبدأ العمل على ميزة جديدة:

```
git checkout -b feature/login develop
```

- بعد الانتهاء، تدمجها في `develop`.

- لتحضير الإصدار:

```
git checkout -b release/v1.0 develop
```

- بعد الاختبار، تدمج `release/v1.0` في `main`.

- لماذا نستخدم ؟ **Git Flow**

- لتنظيم العمل بين أكثر من شخص.
- لتجنّب التعديلات العشوائية على النسخة المستقرة.
- لفصل مراحل التطوير والإصلاح والإصدار بشكل واضح.

Flow Git ليس إلزاميًا، لكنه يُعد من أكثر الطرق شيوعًا في المشاريع المتوسطة والكبيرة، وهو يسهل التنسيق بين أعضاء الفريق وتفادي التعارضات.

3 كيفية استعادة ملفات أو حذف commit

أثناء استخدام ،Git قد تقوم بخطأ مثل حذف ملف، تنفيذ commit غير مرغوب فيه، أو تعديل ملف وتريد التراجع. Git يوفر طرقًا آمنة ومرنة للتراجع عن معظم الحالات.

- أولاً: استعادة ملف تم تعديله قبل الحفظ

إذا عدّلت ملفًا ولم تنفذ `commit` بعد، يمكنك إرجاعه إلى حالته الأصلية (من آخر commit):

```
git restore filename
```

- ثانيًا: استعادة ملف تم حذفه قبل الحفظ

إذا حذفت ملفًا عن طريق الخطأ ولم تحفظ التعديل بعد:

```
git restore filename
```

- ثالثًا: التراجع عن ملف أُضيف لكنه لم يُحفظ (بعد **add** وقبل **commit**)

```
git reset filename
```

سيُخرج الملف من منطقة التتبع، لكن لا يحذفه من القرص.

- رابعًا: حذف آخر commit محليًا (مع إبقاء التعديلات)

```
git reset --soft HEAD~1
```

- يرجع خطوة واحدة في السجل.
- يبقي الملفات كما هي في منطقة التتبع.

- خامسًا: حذف آخر commit بالكامل (ومحو التعديلات)

```
git reset --hard HEAD~1
```

- يحذف آخر .commit
- يحذف التعديلات المصاحبة له من الملفات.

ملاحظة: استخدم هذا الأمر بحذر، لأنه لا يمكن التراجع عنه بسهولة.

- سادسًا: إلغاء commit دُفع إلى GitHub (للمتقدمين)

إذا قمت بـ `push` وتريد التراجع:

1. أعد تعيين الحالة محليًا باستخدام `reset`.
2. ثم ادفع التغيير بالقوة:

```
git push --force
```

لكن هذا الخيار غير مستحب عند العمل الجماعي لأنه يغير التاريخ العام للمشروع.

- متى تستخدم أي طريقة؟

  - restore: لتراجع بسيط على الملفات.
  - reset: لتراجع عن commit محلي.
  - --force push: فقط في حالات خاصة بعد التنسيق مع الفريق.

Git يمنحك تحكمًا كبيرًا في تاريخ مشروعك، ومع الوقت ستعرف متى تستخدم كل أداة بأمان.

الملاحق

الملحق 1: جدول مصطلحات Git للمبتدئين

المصطلح	المعنى ببساطة
Repository	مستودع: هو المشروع الكامل الذي يحتوي على الملفات وتاريخ التعديلات
Commit	حفظ التعديلات في سجل المشروع مع كتابة رسالة توضيحية
Branch	فرع: نسخة مستقلة من المشروع لتجربة أو تعديل معين
Merge	دمج: جمع التعديلات من فرع معين إلى فرع آخر
Remote	مستودع بعيد على الإنترنت مثل GitHub
Push	رفع التعديلات إلى المستودع البعيد
Pull	جلب التعديلات من المستودع البعيد إلى جهازك
Clone	نسخ مستودع موجود على GitHub إلى جهازك

الملحق 2: أوامر Git السريعة حسب الاستخدام

- تهيئة المشروع:

```
git init
```

- إضافة التعديلات:

```
git add .
```

- حفظ التعديلات:

```
git commit -m "Update describe"
```

- رفع التعديلات:

```
git push
```

- جلب التحديثات:

```
git pull
```

- إنشاء فرع:

```
git checkout -b BranchName
```

- تغيير الفرع:

```
git checkout BranchName
```

الملحق 3: أمثلة على **`.gitignore`** لمشاريع شائعة

- لمشاريع C++:

```
*.o
*.exe
build/
```

- لمشاريع Python:

```
__pycache__/
*.pyc
env/
```

- لمشاريع Node.js:

```
node_modules/
.env
```

الملحق 4: خطوات العمل المعتادة في Git

1. افتح المجلد.
2. عدّل الملفات.
3. نفذ:

```
git add .
git commit -m "Message describe the update"
git push
```

4. قبل تعديل جديد، يفضل:

```
git pull
```

الملحق 5: مصادر إضافية لتعلم Git عمليًا (دون روابط)

- اقرأ كتابًا مبسطًا مخصصًا للمبتدئين.
- شاهد دروس فيديو باللغة التي تفضلها.
- طبّق الأوامر على مشروع تجريبي بسيط.
- راجع الأسئلة الشائعة والمشاكل في مجتمعات البرمجة.
- استخدم محررات مثل Code VS لتتبع التعديلات بصريًا.

فهرس المراجع

كتب ودلائل رسمية

- *Pro Git Book*
  تأليف Scott Chacon و Ben Straub.
  الكتاب الرسمي لتعلم Git من البداية حتى المستوى المتقدم، متاح مجانًا، ويمثل مرجعًا أساسًا لفهم المفاهيم العميقة.

- *Git Pocket Guide*
  تأليف Richard E. Silverman، من سلسلة O'Reilly.
  دليل صغير الحجم، سهل الحمل، مناسب للمراجعة السريعة لأوامر Git.

الوثائق الرسمية

- دليل Git الرسمي (git-scm)
  يشمل وصفًا مفصلاً لكل أوامر Git، مع أمثلة عملية وشروحات تقنية.

- *GitHub Docs*
  وثائق GitHub الرسمية التي تغطي استخدام المنصة، الواجهات الرسومية، إعدادات SSH وTokens، وأفضل الممارسات للمستودعات.

دورات ومصادر تعليمية مرئية

- سلسلة Git بالعربي -- قناة الزيرو على YouTube
  دورة مجانية باللغة العربية، تشرح Git خطوة بخطوة مع تطبيقات عملية.

- *Git & GitHub for Beginners -- FreeCodeCamp*

دورة مجانية باللغة الإنجليزية، موجهة للمبتدئين وتتناول Git و GitHub معًا.

مواقع ومقالات تعليمية

- موسوعة أكاديمية حسوب

مقالات تعليمية باللغة العربية تغطي المفاهيم الأساسية لـ Git وطرق استخدامه.

- *Dev.to -- Git Tag*

مجتمع تقني يضم مقالات وتجارب مستخدمين حول Git، نصائح، حلول للمشاكل الشائعة، ودروس ميدانية.

- *Stack Overflow*

قاعدة بيانات هائلة للأسئلة والأجوبة التقنية المتعلقة بأخطاء Git وحلولها.

أدوات مساعدة وتمثيل بصري

- *GitHub Desktop*

أداة رسومية رسمية من GitHub تساعد على تنفيذ الأوامر الأساسية بطريقة بصرية.

- *Visual Studio Code Git Integration*

دمج مباشر بين Git ومحرر الكود VS Code، مناسب للمبتدئين لتعلم التعديلات ومتابعة الحالة من داخل المحرر.