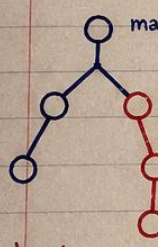
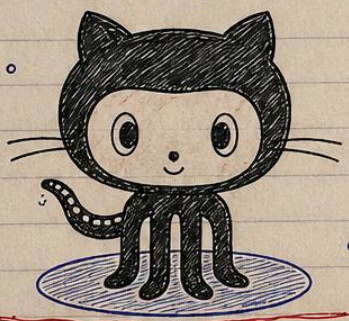
```
$ git status
$ git add .
$ git commit -m "..."
$ git push
>_
```

Git & GitHub

الدليل الشامل للمبتدئين

15 صفحة تشرح كل حاجة محتاج تعرفها
عن GitHub و GitHub





مستوى
المبتدئين

في هذا الكتيب ستتعلم: ←

- من التثبيت لـ push
- الـ Merge والـ Branches
- حل الـ Conflicts
- كتابة README احترافي
- Cheat Sheet كامل



اسم الكاتب :
Mostafa

عدد الصفحات :
15 صفحة

Introduction to Git & GitHub

1. ما هو Git ؟

Git هو نظام للتحكم في الإصدارات (Version Control System).
يساعد المطّورين على تتبع التغييرات في الملفات والمشروع مع الوقت،
والرجوع لأي نسخة سابقة بسهولة.

فكرة بسيطة:
Git بيحفظ تاريخ
كل التغييرات
اللي بتعملها على
مشروعك

2. ما هو GitHub ؟



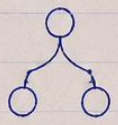
GitHub هو منصة على الإنترنت (موقع) بتستخدم Git لحفظ
المشاريع ومشاركتها والتعاون عليها.
يعني GitHub مكان على السحابة (Cloud) لحفظ مشاريعك.

3. الفرق بين Git و GitHub

Git هو المحرك
GitHub
هو الطريق اللي
بيوصل بيه
مشروعك للناس

GitHub	Git
موقع على الإنترنت (خدمة سحابية).	برنامج (أداة) ثبّته على جهازك.
يعمل عبر الإنترنت ويمكن الوصول له من أي مكان.	يعمل محليًا على جهازك ففط.
لمشاركة المشروع والتعاون مع الأخرين.	لإدارة الإصدارات والتغييرات.
يحتاج حساب (Sign up).	لا يحتاج تسجيل دخول.

4. ما هو Version Control ؟



هو نظام بيحفظ كل نسخ الملف مع كل تغيير، علشان تقدر ترجع
لأي نسخة سابقة وتعرف مين عمل ايه ومتى.

5. لماذا يُستخدمه المبرمجون ؟



- يتتبّع جميع التغييرات بسهولة.
- الرجوع لأي نسخة سابقة في أي وقت.
- التعاون مع فريق بدون تعارض.
- حفظ نسخة احتياطية آمنة للمشروع.

6. فوائد GitHub للمشاريع الجماعية



- مشاركة الكود مع الفريق.
- إدارة الصلاحيات (من يقرأ، من يكتب).
- متابعة المشاكل (Issues) والمهام (Tasks).
- مراجعة الكود قبل دمجه (Pull Request).
- عرض المشروع للعالم (Portfolio).



نصيحة: اتعلم الأساسيات كويس، وتمرن على مشروع صغير لوحدك
قبل ما تشتغل مع فريق.



تعلم اليوم ... تبني مستقبلك ♡

[Mostafa]

Installing Git & First Setup

24/5/2024

① كيف نحمل Git ؟

- نذهب إلى الموقع الرسمي:

https://git-scm.com/downloads

- نختار نظام التشغيل (Windows) ثم نبدأ التحميل.

Git هو نظام تحكم في الإصدارات يساعدنا نتابع التغييرات في ملفات المشروع.

② خطوات تثبيت Git على Windows

- بعد تحميل الملف، شغّل الـ installer.
- اضغط Next في كل خطوة.
- في خطوة اختيار المكونات، اترك الإعدادات الافتراضية.
- اختر محرر النصوص (يفضل VS Code أو Nano).
- اختر الافتراضي في خطوة اختيار الـ PATH.
- اضغط Install ثم Finish.

ملاحظة:
اترك الخيارات الافتراضية غالباً تكون مناسبة للبداية.

③ كيف تتأكد أن Git تم تثبيته ؟

افتح الطرفية (Terminal) واكتب:

```
git --version
```

إذا ظهر رقم الإصدار، يعني Git مثبت بنجاح.

⚠ تنبيه!
تأكد من اتصالك بالإنترنت عند التحميل لأول مرة.

④ إعداد الاسم والإيميل باستخدام git config

نستخدم الأمر التالي لتحديد اسمك وإيميلك (يظهر كل commit):

```
git config --global user.name "Mostafa"
git config --global user.email "example@gmail.com"
```

غير المعلومات وفقا لك

⑤ global و local config

global (عام)	(خاص بالمستودع)
• يتم تطبيقه على جميع المستودعات.	• يتم تطبيقه على مستودع واحد فقط.
• يُخزن في ملف الإعدادات الخاص بالمستخدم.	• يُخزن داخل مجلد المشروع في .git/config.
• مناسب للإعدادات الشخصية (الاسم، الإيميل).	• مناسب لإعدادات خاصة بالمشروع.

تقدر تشوف الإعدادات الحالية بالأمر:

```
git config --list
```

⑥ أول خطوة بعد تثبيت Git

ابدأ بمجلد مشروع جديد ثم حوّله إلى مستودع Git:

MyProject →

```
git init
```

ستظهر مجلد مخفي باسم .git داخل مشروعك.

نصيحة:
ابدأ بتجربة الأوامر البسيطة وتعلم خطوة بخطوة.

[Mostafa] ♡

Creating a Repository – إنشاء المستودع

25/5/2024

① ما هو الـ Repository ؟

- الـ Repository (المستودع) هو مكان بيتخزن فيه المشروع وكل التغييرات اللي بتحصل عليه مع تاريخ كامل لكل التعديلات.

Repository = مجلد بيتتبع فيه كل التغييرات ☆

② Local Repository و Remote Repository

Local Repository (محلي)	Remote Repository (بعيد)
• موجود على جهازك فقط. • بتتحكم فيه أنت وحدك. • مثال: مجلد فيه git.	• موجود على سيرفر (زي GitHub). • تقدر تشاركه مع أي حد. • مثال: GitHub Repository

ملاحظة:
بنربط الـ Local بالـ Remote علشان نرفع التغييرات ونشاركها.

③ إزاي تعمل Repository جديد على جهازك

في أي مجلد تكتب الأمر:

```
git init
```

هيعمل مجلد مخفي اسمه .git. وده اللي بيخلي المجلد Repository.

! Tip
مجلد .git. هو قلب الـ Repository. لا تفتحه أو تحذفه يدوياً. ♡

تقدر تعمل Repository وفي نفس الوقت تعطية اسم:

```
git init my-project
```

هيعمل مجلد باسم my-project وبداخله .git تلقائياً.

تأكد من وجود المجلد المخفي:

```
ls -a    (to see .git folder)
```

على Windows استخدم: dir /a

④ إزاي تعمل Repository على GitHub

1. ادخل على GitHub.com وسجل دخول.
2. اضغط على (+) ثم New repository.
3. اكتب اسم المستودع (مثلاً my-project).
4. اختار Public أو Private.
5. (اختياري) أضف وصف.
6. اضغط Create repository.

مهم:
لا تعمل README الأن لو هتربط من الطرفية. (نخليه فاضي) ☆

⑤ ربط الـ Local بالـ Remote

بعد إنشاء الـ Repository على GitHub، انسخ رابط الـ HTTPS.
ثم في الطرفية داخل مجلد المشروع:

```
git remote add origin https://github.com/username/repo.git
```

تأكد هو الاسم هو origin للـ Remote.

تأكد من الربط:

```
git remote -v
```

هيظهر لك الرابط (fetch- و push).

☆ أوامر مهمة

الأمر	الشرح
git init	إنشاء Repository في المجلد الحالي.
git init my-project	إنشاء Repository جديد باسم my-project.
ls -a	عرض الملفات المخفية (مثل .git).
git remote add origin <url>	ربط الـ Local بالـ Remote.
git remote -v	عرض روابط الـ Remote.

⚠ تنبيه !
تأكد أنك داخل مجلد مشروعك عند تنفيذ الأوامر.

[Mostafa] ♡

Git Workflow – Working Directory / Staging / Repository

26/5/2024

① ما هو الـ Git Workflow ؟

هو الطريقة اللي بيتبعها Git علشان يتابع التغييرات ويحولها من ملفات على جهازك لحد ما تتخزن بشكل نهائي في الـ Repository .

Tip !
فكر فيها زي إنك بتجهز حاجات في شنطة قبل السفر

Staging Area
منطقة مؤقتة بتحط فيها التغييرات اللي عايز تضمنها في الـ Commit الجاي.

② Working Directory و Staging Area و Repository ؟

Working Directory	Staging Area	Repository (.git)
• دي ملفات مشروعك على جهازك.	• منطقة وسيطة .	• دي قاعدة البيانات بتاعة Git.
• هنا بتعمل وتعدل الملفات .	• بتحط فيها التغييرات اللي عايز تضمنها في الـ Commit .	• هنا بيتخزن كل الـ Commits بشكل دائم .
• التغييرات دي لسه غير جاهزة للحفظ النهائي .	• مش محفوظة بشكل نهائي بعد.	• ده تاريخ المشروع .

③ إزاي الملف بيتحرك من مرحلة لمرحلة ؟



④ ليه الـ Staging Area مهمة ؟

- بتديك تحكم في إيه اللي هيتسجل في الـ Commit .
- تقدر تختار تغييرات معينة وتسيب الباقي .
- بتساعدك تكتب رسائل Commit واضحة ومنظمة .
- بتخلي تاريخ المشروع نظيف .

أوامر مهمة

- عرض حالة الملفات : `git status`
- إضافة ملف معين : `git add filename.txt`
- إضافة كل التغييرات : `git add .`
- حفظ التغييرات (Commit) : `git commit -m "my message"`

⑤ الحالات المختلفة للملف في Git

الحالة	الوصف	أين يوجد ؟	كيف تنتقل لهذه الحالة ؟
Untracked ○	ملف جديد لم يتم تتبعه بواسطة Git بعد.	Working Directory	إنشاء ملف جديد .
Modified ●	ملف تم تغييره بعد آخر Commit .	Working Directory	تعديل ملف متتبع.
Staged ●	تمت إضافة التغييرات إلى Staging Area .	Staging Area	`git add filename.txt`
Committed ●	تم حفظ التغييرات في Repository .	Repository (.git)	`git commit -m "msg"`

ملاحظات سريعة :

- إنشاء ملف هو أفضل صديق ليك دائمًا.
- كل Commit = لقطة (Snapshot) لحالة مشروعك .
- تقدر ترجع لأي Commit سابق بسهولة .

ملخص سريع

تعدل ملفاتك (Working Directory) → `git add` (Staging Area) → `git commit` (Repository)

[Mostafa] ♡

git add / commit / status

27/5/2024

① ما هو git status ؟

هو أمر يعرض حالة الملفات في الـ Working Directory ويخبرك إذا كان في ملفات جديدة أو متعدلة أو جاهزة للـ Commit

إزاي تقراه ؟

- ملفات جديدة لم يدرج Git بتابعها بعد .
- Changes not staged for commit : ملفات متعدلة ولم تضاف للـ Staging .
- Changes to be committed : ملفات جاهزة ومضافة للـ Staging .

```
$ git status
On branch main
Untracked files:
    index.html
Changes not staged for commit:
    style.css
Changes to be committed:
    .script.js
```

بتقول git add لـ Git تراقب الملف، git commit بتحفظه فعلاً.

② git add . و git add filename

git add filename	git add .
• تضيف ملف محدد فقط للـ Staging Area .	• تضيف كل التغييرات والملفات الجديدة في المجلد الحالي للـ Staging .
• تستخدم عندما تريد إضافة ملف واحد فقط .	• تستخدم عندما تريد إضافة كل شيء مرة واحدة.

أمثلة :

```
$ git add index.html
```
(إضافة ملف واحد فقط)

```
$ git add .
```
(إضافة كل الملفات)

! Tip

commit message لازم تكون واضحة زي: add login page

③ ما هو الـ commit ؟ وليه مهم ؟

- الـ commit هو حفظ للتغييرات التي أضفتها في الـ Staging Area داخل الـ Repository بشكل دائم .
- كل commit يمثل نقطة حفظ في تاريخ مشروعك .
- يمكنك الرجوع لأي commit سابق وقت ما تحتاج .

أمثلة :

```
$ git commit -m "add homepage layout"
$ git commit -m "fix navbar bug"
```

④ إزاي تكتب commit message كويسة ؟

- اكتب رسالة قصيرة وواضحة توضح التغيير .
- استخدم فعل في صيغة الأمر (يفضل) .
- اكتب باللغة الإنجليزية .
- لا تكتب Message عامة مثل: fix أو fix .
- ركز على لماذا تم هذا التغيير .

Don't's ✗	Do's ✓
✗ update	✓ add login page UI
✗ fix bug	✓ fix navbar bug on mobile
✗ changes	✓ update README file
✗ test	✓ refactor auth logic

⑤ الفرق بين git add و git status

git commit	git add	git status
• يحفظ التغييرات من Staging Area إلى Repository .	• ينقل التغييرات من Working Directory إلى Staging Area .	• يعرض حالة الملفات في المشروع .
• يسجل التغييرات بشكل دائم .	• يخبر Git أي التغييرات تريد حفظها.	• يخبرك ما الذي تم تعديله أو إضافته .
• يضيف رسالة توضح التغيير .	• لا يحفظ التغييرات بشكل دائم .	• لا يغير أي شيء في الملفات .
`$ git commit -m "message"`	`$ git add .` `$ git add filename`	`$ git status`

تذكر: سير عمل Git يعتمد على 3 خطوات : status → add → commit

[Mostafa] ♡

Viewing Changes & History – مشاهدة التغييرات والتاريخ

28/5/2024

① إزاي تشوف التغييرات اللي عملتها قبل الـ commit ؟

تقدر تشوف التغييرات اللي عملتها في الملفات ومضيقهاش الـ Staging Area أو الـ Repository باستخدام الأمر:

```
git status
```

بيعرضلك:
- الملفات الجديدة (Untracked)
- الملفات المعدلة (Modified)
- الملفات المضافة للتجهيز (Staged).

كل commit ليه ID فريد اسمه hash تقدر ترجعله في أي وقت ☆

② إزاي تشوف تاريخ الـ commits ؟

تاريخ كل التغييرات اللي اتعملت في المشروع.

```
git log
```

! Tip

git log --oneline أسهل وأسرع لما يكون عندك commits كتير ♡

③ git log و git log --oneline

git log	git log --oneline
• يعرض كل المعلومات.	• يعرض كل commit في سطر واحد.
• مفضل وطويل.	• مختصر وسريع.
• مناسب لو عايز تفاصيل.	• مناسب لمراجعة سريعة.

④ إزاي تشوف الفرق بين نسختين من ملف ؟

لو عايز تشوف التغييرات اللي عملتها في ملف قبل ما تعمل commit:

```
git diff
```
يعرض الفرق في كل الملفات.

أو لملف معين:

```
git diff filename.txt
```
يعرض الفرق في ملف واحد فقط.

⑤ إزاي ترجع تشوف commit قديم ؟

تقدر تشوف تفاصيل أي commit قديم باستخدام الـ ID بتاعه (hash).

مثال:

```
git show abc1234
```
يعرض كل التغييرات اللي حصلت في commit ده.

ملاحظة
- ممكن تستخدم أول 6 أو 7 أحرف فقط من الـ hash لو مفيش تكرار. ☺

☆ أوامر مهمه للمراجعة

الأمر	الوظيفه	مثال
git log	يعرض تاريخ كل الـ commits بالتفاصيل.	git log
git log --oneline	يعرض كل commit في سطر واحد.	git log --oneline
git status	يعرض حالة الملفات والتغييرات.	git status
git diff	يعرض الفرق بين النسخة الحالية و أخر نسخة تم تجهيزها (Staging Area).	git diff
git diff filename.txt	يعرض الفرق في ملف معين.	git diff filenne.html
git show <hash>	يعرض تفاصيل commit معين.	git show abc1234

مثال على git log

```
commit a1b2c3d (HEAD -> main)
Author: Mostafa <mostafa@gmail.com>
Date:   Tue May 28 10:20:00 2024 +0200

    fix: update navbar style

commit d4e5f6a
Author: Mostafa <mostafa@gmail.com>
Date:   Tue May 28 09:15:00 2024 +0200

    add: homepage layout

commit 7g8h9i0
Author: Mostafa <mostafa@gmail.com>
Date:   Tue May 28 08:30:00 2024 +0200

    initial commit
```

☆ نصيحة ذهبية
- اعمل commit صغيرة ومفيدة مع كل تغيير منطقي.

ده بيسهل عليك المراجعة والرجوع لأي نقطة في المشروع. ☺

تذكر:

افهم تاريخ مشروعك كويس... علشان تتحكم فيه بثقة! ♡

[Mostafa] ♡

Connecting Project to GitHub - ربط الشروع بـ المشروع

إزاي تعمل Remote Repository على GitHub ؟

1. ادخل على GitHub وسجل دخولك.
2. اضغط على (+) ثم New repository .
3. اختر اسم للمشروع .
4. (اختياري) اكتب وصف .
5. اختـر Private أو Public .
6. لا تضع علامة على Add a README .
7. اضغط Create repository .

إزاي تربط مشروعك المحلي بـ GitHub ☆

ربط مشروعك المحلي بـ GitHub يعني إنك ترفع كود مشروعك على السحابة علشان تحافظ عليه، وتشارك فيه، وتشتغل عليه من أي مكان .

الفرق بين و HTTPS

الجانب	SSH	HTTPS
الرابط يبدأ بـ	git@github.com:	https://
طريقة التوثيق	مفتاح SSH	اسم مستخدم + Token
الأمان	أعلى	جيد
الاستخدام	محترفين	مبتدفين
الإعداد	سهل	يتطلب إعداد مفتاح

نصيحة مهمة
استخدم HTTPS لو مبتدئ ، SSH للمحترفين ☆

هـــو origin الافتراضي للـ Remote Repository ☆

خطوات الربط من الأول للآخر

1. انتقل إلى مجلد المشروع ← `cd path/to/your/project`
2. تأكد إن المشروع بـ Git ← `git status`
3. أضف رابط الريبوت (HTTPS أو SSH) ← `git remote add origin https://github.com/username/repo.git`
4. تأكد من إضافة الريموت ← `git remote -v`
5. تأكد من اسم الفرع الرئيسي (غالبًا main) ← `git branch -M main`
6. ارفع المشروع لأول مرة ← `git push -u origin main`

المشروع المحلى (Local Repo)

ارتباط عبر الإنترنت

GitHub (Remote Repo)

إزاي تتأكد إن الربط اشتغل صح ؟

- `git remote -v` ← لو ظهر رابط origin يبقى تمام.
- ادخل على صفحة الريبو على GitHub وتأكد إن الملفات ظهرت.
- كل push جديد هيظهر على GitHub.

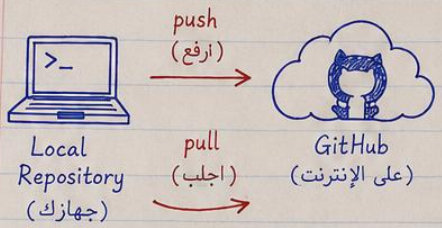
ملاحظات
- لو غيرت اسم الريبو على GitHub. هيجتاج تحدث الرابط .
- تقدر تضيف أكثر من remote لو بتشتغل على أكثر من مكان.

استمر في التعلم والتجربة، كل يوم بتقرب من الاحتراف ! ♡

[Mostafa]

git push / pull / fetch

① ما هو git push و إزاي تستخدمه ؟



git يستخدم لرفع التغييرات من الـ Local Repository إلى الـ Remote Repository على GitHub .

```
git push origin main
```

أو لأول مرة فقط :

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

② ما هو git pull و إمتى تستخدمه ؟

- git pull يجلب التغييرات من الـ Remote Repository ويدمجها مع الـ Local Repository .
- تستخدمه لما عايز تجيب آخر التحديثات وتدمجها في شغلك .

```
git pull origin main
```

> git fetch بتجيب التغييرات
> بس مش بتدمجها ،
> git pull بتجيب وبتدمج

③ الفرق بين git pull و git fetch

الأمر	git pull	git fetch	git push
الإتجاه	من Remote إلى Local	من Remote إلى Local	من Local إلى Remote
البيانات	يجلب التغييرات	يجلب التغييرات	يرفع التغييرات
الدمج	يدمج تلقائياً	لا يدمج	—
التأثير على المشروع	يعدل ملفاتك	لا يغير ملفاتك	يُحدث الريموت

④ إزاي تجيب التغييرات من غير ما تدمجها ؟

استخدم git fetch لجلب التغييرات فقط ، وبعدها تقدر تشوفها ونقرر إمتى تدمجها .

```
git fetch origin
```

```
$ git fetch origin
From github.com:username/repo
 * [new branch]      main -> origin/main
```

← التغييرات اتجلبت بس مش متدمجة !

⑤ أكثر الأخطاء الشائعة عند الـ push و pull

⚠ لو في conflict مش هتقدر تعمل push لازم تحل التعارض الأول .

- error: failed to push some refs
  سبب : الريموت أحدث من المحلي . (احل : اعمل git pull الأول)
- error: failed to push (rejected)
  سبب : في حد رافع تغييرات قبلك . (احل : اعمل git pull ثم حاول تاني)
- merge conflict عند git pull
  سبب : فيه تعارض بين التغييرات . (احل : افتح الملفات ، حل التعارض ، ثم commit)
- نسيت تحدد الفرع
  احرص على استخدام الفرع الصحيح ، مثال : git pull origin main

★ نصيحة مهمة
دايماً اعمل git pull الأول قبل ما تبدأ شغل جديد

ملخص سريع للأوامر

git pull	git fetch origin	git pull origin main	git push -u origin main	git push origin main
جلب ودمج (الفرع الحالي)	جلب التغييرات فقط	جلب ودمج التغييرات	رفع لأول مرة وتحديد الـ upstream	رفع التغييرات

★ [Mostafa] ★

Branches in Git - الفروع

① ما هو الـ Branch وليه بنستخدمه ؟

- الـ Branch هو خط تطوير مستقل عن الخط الرئيسي.
- بنستخدمه عشان نشتغل على مميزات جديدة أو نصلح أخطاء من غير ما نأثر على الكود الأساسي في المشروع.
- بعد ما نخلص شغلنا، ندمجه مع الـ main.

الـ branch هو زي نسخة موازية من مشروعك ☺

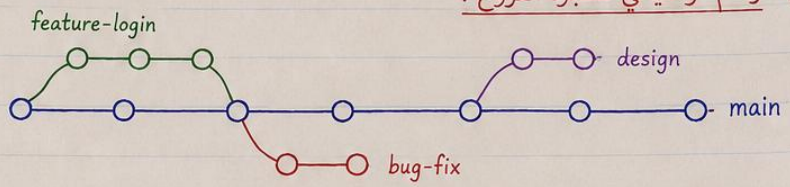
② ما هو الـ main branch ؟

- الـ main (أو master) هو الفرع الرئيسي للمشروع.
- يحتوي على الكود المستقر والجاهز للإنتاج.
- بندمج عليه فقط بعد ما نتأكد من أن التغييرات جاهزة.

مثال من الحياة الواقعية :

- branch لـ login feature
- branch لـ design
- branch لإصلاح bug

③ رسم توضيحي لشجرة الفروع :



④ مقارنة سريعة بين بعض الـ branches :

اسم الـ Branch	الغرض	أمثلة	ملاحظات
main	الخط الرئيسي المستقر	الكود الجاهز للإنتاج	لا تشتغل مباشرة عليه
feature/*	إضافة مميزات جديدة	feature-login , feature-payment	تُدمج في الـ main بعد الانتهاء
bugfix/*	إصلاح الأخطاء	bugfix/login-error	تُدمج في الـ main بعد الاختبار
hotfix/*	إصلاح سريع في الإنتاج	hotfix/critical-bug	غالباً من الـ main مباشرة

⑤ إزاي تعمل branch جديد ؟

- `git branch feature-login` ← إنشاء branch فقط
- `git checkout -b feature-login` ← إنشاء + الانتقال له مباشرة

⑥ إزاي تتنقل بين الـ branches ؟

- `git checkout feature-login` ← (الطريقة القديمة)
- `git switch feature-login` ← (الطريقة الأحدث)

⑦ إزاي تشوف كل الـ branches الموجودة ؟

- `git branch` ← يعرض الـ branches المحلية
- `git branch -a` ← يعرض المحلية والـ remote

⑧ إزاي تحذف branch ؟

- `git branch -d feature-login` ← يحذف لو مدموج
- `git branch -D feature-login` ← يحذف بالقوة (حتى لو مش مدموج)

★ نصيحة مهمة

اعمل branch جديد لكل feature أو bug fix عشان الـ main يفضل نظيف ♥

مثال عملي سريع

`git push origin feature-login` ← `(شغلك) ...` ← `git checkout -b feature-login` ← `git pull` ← `git switch main`

- `git push origin feature-login` : ارفع الفرع على GitHub
- `(شغلك) ...` : اعمل التعديلات
- `git checkout -b feature-login` : اعمل branch جديد لـ login
- `git pull` : حدث الـ main
- `git switch main` : ارجع لـ main

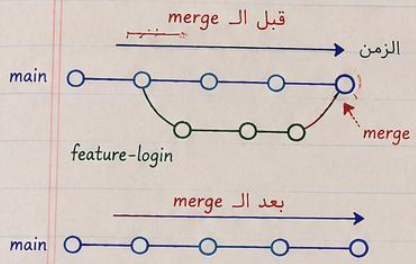
★ [Mostafa] ★

Merging Branches - دمج الفروع

① ما هو الـ Merge وليه بنستخدمه ؟

- الـ Merge هو عملية دمج التغييرات من فرع (branch) في فرع آخر.
- بنستخدمه عشان نجيب كل الشغل اللي اتحط في الفرع وندمجه في الفرع الرئيسي (main).

الـ merge بيجمع التغييرات من برانشين في برانش واحد



② إزاي تدمج branch في الـ main ؟

اتأكد إنك على الـ main ← `git checkout main`

ادمج الفرع في الـ main ← `git merge feature-login`

احذف الفرع بعد ما تدمج (اختياري) ← `git branch -d feature-login`

③ الفرق بين Fast-forward Merge و 3-way Merge

وجه المقارنة	Fast-forward Merge	3-way Merge
تعريف	بيتحرك الـ main لقدام مباشرة لآخر commit في الفرع.	ينشئ commit جديد يجمع بين تاريخ الـ main والفرع.
متى يحدث ؟	لو مفيش commits جديدة في الـ main من بعد ما اتعمل الفرع.	لو فيه commits جديدة في الـ main من بعد ما اتعمل الفرع.
شكل التاريخ	التاريخ يكون خط مستقيم (نظيف).	هيظهر commits دمج في التاريخ.
الأمر المستخدم	`git merge feature-branch`	`git merge --no-ff feature-branch`

④ إزاي تتأكد إن الـ Merge اتعمل صح ؟

- راجع تاريخ الـ commits.
- شوف الفرق في الملفات.
- شغل المشروع وتأكد إن كل حاجة شغالة.
- استخدم الأوامر دي للمراجعة :

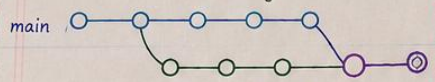
يعرض شكل التاريخ ← `git log --oneline --graph`

بتأكد من عدم وجود تغييرات ← `git status`

يعرض الفروع الحالية ← `git branch`

مثال واقعي :

فرع جديد لتطوير صفحة تسجيل الدخول
feature-login



بعد الانتهاء من الشغل على feature-login نقوم بدمجه في main ليصل التغيير للمشروع.

⑤ إمتى تعمل merge وإمتى لأ ؟

تعمل merge عندما :

- ✓ الفرع جاهز ومختبر.
- ✓ التغييرات منتهية ومراجعة.
- ✓ تريد إضافة الشغل إلى المشروع الرئيسي.

لا تعمل merge عندما :

- ✗ الفرع لسه في مرحلة تطوير.
- ✗ في تغييرات غير مكتملة أو بها أخطاء.
- ✗ فريقك لسه بيشتغل على نفس الفرع.

★ نصيحة مهمة

دايماً تأكد إنك على الـ main branch قبل ما تعمل merge

♥

ملخص عملي سريع

`git log --oneline --graph` ← `git branch -d feature-login` ← `git merge feature-login` ← `git checkout main`

مراجعة التاريخ — حذف الفرع (اختياري) — دمج الفرع — الانتقال للـ main

★ [Mostafa] ★

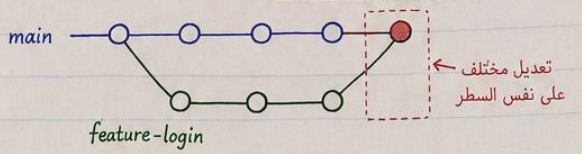
Merge Conflicts - تعارض الدمج

① ما هو Merge Conflict وإمتي بيحصل ؟

- الـ Merge Conflict هو تعارض بين تغييرات اتعملت في نفس الجزء من الملف في برانشين مختلفين.
- بيحصل لما تحاول تدمج برانش في برانش تاني وفيه اختلافات على نفس السطر أو نفس الجزء من الكود.

الـ conflict يحصل لما نفس السطر اتغير في برانشين مختلفين

رسم توضيحي للمشكلة :



② ليه بيحصل conflict ؟

- شخصين عدلوا على نفس السطر.
- حد عدل وحذف سطر، والتاني عدله.
- تعديلين متعارضين على نفس الكود.

③ إزاي تعرف إن في conflict ؟

- أثناء الـ merge ، Git هيوقف الدمج ويظهر رسالة.
- هيكون في ملفات عليها (unmerged).
- تلاقي كلمة "both modified" في git status.

```
git status
```

مثال علي شكل الـ conflict داخل الملف :

```
// file: app.js
function login() {
<<<<<<< HEAD (main)
  console.log("Login from main");
=======
  console.log("Login from feature-login");
>>>>>>> feature-login
}
```

main ← في تعديل
الفاصل ←
تعديل في feature-login ←

علامات الـ conflict ومعناها :

العلامة	الشرح
<<<<<<< HEAD	بداية التغيير في البرانش الحالي (اللي أنت عليه)
=======	فاصل بين التعديلين المتعارضين
>>>>>>> branch-name	نهاية التغيير في البرانش الأخر

★ نصيحة مهمة
دايماً اعمل git pull قبل ما تبدأ شغل عشان تقلل الـ conflicts
♥

⚠ تحذير
متعملش merge وانت مش فاهم الـ conflict
راجع التغييرات كويس قبل ما تحلها.

④ إزاي تحل الـ conflict خطوة خطوة ؟

① ابدأ عملية الدمج ← `git merge feature-login`
② اعرّف الـ merge فيها تلقائياً :
③ افتح الملف وشوف أماكن التعارض : `git status`
④ عدّل الملف يدوياً واختار التعديل الصحيح.
⑤ بعد ما تحل التعارض، احفظ الملف.
⑥ أضف الملف المحل : `git add filename.txt`
⑦ اعمل commit لإنهاء الـ merge :

```
git commit -m "resolve merge conflict"
```

أوامر مفيدة أثناء التعارض :

`git merge --abort` ← إلغاء عملية الدمج والرجوع الحالة قبل الـ merge
`git status` ← لمعرفة حالة الملفات

⚠ تحذير
متعملش merge وانت مش فاهم الـ conflict
راجع التغييرات كويس قبل ما تحلها.

⑤ إزاي تتجنب الـ conflicts من الأساس ؟

- ✓ دايماً اعمل git pull قبل ما تبدأ شغل جديد.
- ✓ خلي التعديلات صغيرة ومتكررة واعمل commit باستمرار.
- ✓ تواصل مع فريقك ووزّع الشغل على ملفات مختلفة.
- ✓ اتجنب تعديل نفس السطر مع حد تاني.
- ✓ اعرف آخر تغييرات على main قبل ما تعمل merge.

[Mostafa]

Undo Changes – (التراجع عن التغييرات / reset / restore / revert)

ما الهدف من التراجع ؟

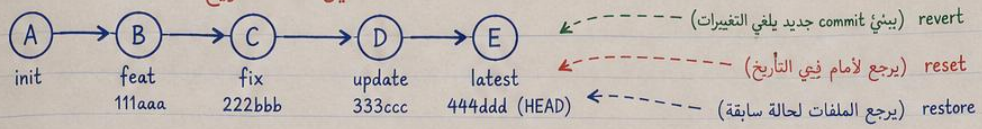
أثناء العمل على أي مشروع، ممكن تعمل تغييرات أو commits وتكتشف بعدين إنك عايز ترجع لحالة سابقة.

Git بيقدم لنا أكثر من طريقة للتراجع، وكل واحدة ليها استخدام مناسب ليها تأثير مختلف.

الفرق بين restore و revert

- restore : يرجع التغييرات في (Working / Staging)
- revert : يرجع لأمام الـ commits لمستوى سابق (ويسحب تغييرات من التاريخ).
- revert : يعمل commit جديد يلغي التغييرات السابقة. بدون ما يمسح أي حاجة من التاريخ.

تخيل عندك التاريخ ده :



مقارنة reset / restore / revert

الأمر	الوظيفة	يؤثر على التاريخ؟	الاستخدام المناسب
restore	يرجع الملفات لحالة سابقة (Working أو Staging)	لا	للتراجع عن تغييرات محلية في الملفات
reset	يرجع الـ commits لمستوى سابق	نعم	للتراجع عن commits محلياً قبل النشر
revert	ينشئ commit جديد يعكس التغييرات	لا	للتراجع عن تغييرات تم نشرها بالفعل

git reset --hard خطير جدًا لأنه يمسح التغييرات نهائيًا

فكر كويس قبل ما تستخدم reset --hard

إزاي تتراجع عن commit من غير ما تمسحه ؟

استخدم revert لإنشاء commit جديد يلغي التغييرات:

```
git revert abc1234
```

- هيفتح editor تكتب رسالة، تحفظ وتخرج.
- هيعمل commit جديد يعكس التغييرات.
- مناسب جدًا لو الـ commit اتنشر على GitHub.

إزاي ترجع ملف لحالته قبل التعديل ؟

لو عدّلت في ملف وندمت قبل ما تضيفه (add):

```
git restore filename.txt
```

يرجع الملف لحالته الأخيرة المحفوظة.

لو أضفته (staged) وعايز ترجع الـ staging بس:

```
git restore --staged filename.txt
```

يلغي الـ staged ويخليه untracked (متغير).

لو شايل تغييرات كتير ونِدِم ترجع كل الملفات:

```
git restore .
```

☆ يرجع كل الملفات لحالتها الأخيرة.

إزاي تمسح commits بالكامل ؟

لو عايز ترجع لمستوى سابق وتمسح اللي بعده:

- تمسح آخر commit فقط:

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

يرجع الـ HEAD والـ staging بس، والتغييرات بتفضل موجودة.

- تمسح آخر commit وتتخلص من التغييرات:

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

يرجع كل حاجة ويحذف التغييرات نهائياً.

⚠ (خطير جدًا، لا يمكن التراجع بعد كدة).

- ترجع لعدة commits سابقة:

```
git reset --hard HEAD~3
```

☆ يرجع لثلاثة commits ويمسح اللي بعدهم.

نصيحة ذهبية

استخدم revert لو شغلك على GitHub عشان تأمن من reset وتحافظ على التاريخ.

تذكر دائمًا.

- لا تستخدم reset --hard إلا إذا كنت متأكد 100٪.
- قبل أي عملية تراجع، تأكد من أنك على الفرع الصحيح!
- لو فقدت حاجة بالخطأ، جرب git reflog يمكن ينقذك.

حالات ستخدام سريعة

- غيرت حاجة في ملف وندمت → restore
- أضفت بالخطأ للـ staging → restore --staged
- عملت commit محلي وندمت → reset --soft
- عايز تمسح تغييرات بدون رجعة → reset --hard
- عايز تلغي commit على GitHub → revert

أوامر مهمة تساعدك تفهم أنت فين

- git status : حالة الملفات.
- git log --oneline : → يعرض التاريخ بشكل مختصر.
- git reflog : مِش تحركات الـ HEAD.
- git show <commit> : → يعرض تفاصيل commit معين.

قوي ومفيد جدّاً، لكن مع القوة تأتي المسؤولية!

☆ افهم اللي بتعمله — قبل أي تراجع، واحتفظ بنسخ احتياطية من شغلك.

[Mostafa]

ملف التجاهل - .gitignore File

① ماهو ملف الـ gitignore وليه مهم ؟

ملف gitignore. بيحدد الملفات والمجلدات اللي عايز Git يتجاهلها.
يعني مبتابعهاش ومبيرفعهاش على الـ repository.
ده بيساعد في الحفاظ على نظافة المشروع وأمان البيانات.

الـ gitignore
بيقول لـ Git تجاهل
الملفات دي
ومتتبعهاش

② إزاي تعمل ملف gitignore ؟

(إنشاء ملف gitignore في الجُذر) `touch .gitignore`

③ إيه الملفات اللي المفروض تتجاهلها ؟

نوع الملف / المجلد	ليه نتجاهله ؟
node_modules/	حجم كبير ويتسبب تاني بـ npm install
.env	يحتوي على بيانات حساسة (مفاتيح وأسرار)
*.log	ملفات لوج مؤقتة
build/ أو dist/	ملفات البناء (Build) تتولد تلقائياً
.DS_Store	ملف خاص بنظام macOS
__pycache__/	ملفات كاش بايثون

④ إزاي تكتب rules جوه الـ gitignore ؟

- كل سطر قاعدة واحدة.
- تقدر تكتب اسم ملف أو مجلد.
- تقدر تستخدم wildcards زي *.
- # للبداية تعليق (مش قاعدة).

مثال على ملف .gitignore

```
# dependencies
node_modules/

# environment variables
.env

# logs
*.log

# macOS files
.DS_Store

# build
dist/
build/

# python cache
__pycache__/
*.pyc
```

مثال من الواقع

```
my-project/
├── node_modules/   ✗ متجاهل
├── .env            ✗ متجاهل
├── index.js        ✓ متتبع
└── .gitignore      ✓ متتبع
```

⑤ إزاي تتأكد إن الملف اشتغل صح ؟

1. شغّل الأمر ده وشوف الملفات اللي مش متتبعة:

`git status`

2. الملفات اللي متجاهلة مش هتظهر في Untracked files.
3. تقدر تتأكد كمان بجرِبْ تضيف ملف متجاهل وتشوف إنه مش بيتضاف.

لما ملف يكون متتبع قبل ما تضيفه في الـ gitignore، Git هيكمل بتابعه.
الخل:

`git rm --cached filename`

ومش هيحذف الملف من جهازك.

TIP
دايماً أعمل gitignore من أول المشروع مش في الأخر

☆ ملاحظات سريعة

☆ تقدر تُستخدم منضات زي https://github.com/github/gitignore علشان تولد ملف جاهز لمشروعك.
☆ كل مشروع ليه احتياجات مختلفة، استخدم اللي يناسبك.

[Mostafa]

Writing a Professional README.md - كتابة README احترافي

① ما هو ملف الـ README وليه مهم ؟

- ملف README.md هو الواجهة الأولى لمشروعك على GitHub .
- يشرح فكرة المشروع، طريقة استخدامه، وكيف يساهم الناس فيه .
- كلما كان واضح ومرتب، زادت فرصه في جذب مستخدمين ومساهمين .

استخدم Markdown عشان تنسق الـ README بشكل احترافي

② إيه الأجزاء الأساسية في أي README احترافي ؟

القسم	ماذا تكتب فيه ؟
# Project Title	اسم المشروع بشكل واضح وجذاب
## Description	شرح مختصر عن المشروع ولماذا تم إنشاؤه
## Installation	خطوات تثبيت المشروع والمتطلبات
## Usage	طريقة استخدام المشروع مع أمثلة
## Features	أهم المميزات والخصائص
## Contributing	كيفية المساهمة في تطوير المشروع
## License	نوع الرخصة المستخدمة في المشروع

TIP

الـ README هو أول حاجة الناس بتشوفها في مشروعك خليه يبهرهم

③ إزاي تكتب README يجذب الناس لمشروعك ؟

- اكتب عنوان واضح وجذاب .
- أشرح المشكلة والحل اللي بيقدمه مشروعك .
- استخدم أمثلة وصور (screenshots) لو أمكن .
- اجعل التعليمات سهلة وخطوة بخطوة .
- أضف build, license, (مثل: Badges) ملصقات .
- اكتب بلغة بسيطة ومباشرة .
- حدث الـ README باستمرار . ♡

④ الفرق بين README عادي و README احترافي

✓ README احترافي (مميز)	✗ README عادي (ضعيف)
• منظم وواضح بجميع الأقسام الأساسية .	• عنوان فقط أو وصف قصير .
• يحتوي على أمثلة وصور توضيحية .	• لا توجد تعليمات تثبيت أو استخدام .
• يسهل على المستخدم البدء بسرعة .	• غير مرتب وصعب القراءة .
• يشجع على المساهمة ويدعم المشروع .	• لا يوجد أمثلة أو صور .
• يظهر احترافية المشروع ويزيد من ثقته .	• لا يوضح طريقة المساهمة أو الرخصة .

⑤ نصايح لكتابة README مميز

- ✓ استخدم عناوين (Headings) واضحة .
- ✓ استخدم القوائم النقطية والجداول عند الحاجة .
- ✓ أضف روابط مهمة (Website - Demo - Docs)
- ✓ استخدم الصور أو الـ GIF . لشرح الفكرة .
- ✓ اجعل الـ README مختصر لكن شامل .
- ✓ راجع الإملاء واللغة قبل النشر .

أوامر مفيدة

`touch README.md`	إنشاء ملف Reapme
`code README.md`	فتح الملف في VS Code .
`git add README.md`	إضافة للـ commit .
`git commit -m "add README"`	عمل commit .
`git push`	رفع التغييرات

قالب README سريع

```
# Project Title
## Description
## Installation
## Usage
## Features
## Contributing
## License
```

مثال : README فاضي ☹

```
# my-project
...
```

مثال : README احترافي ☺

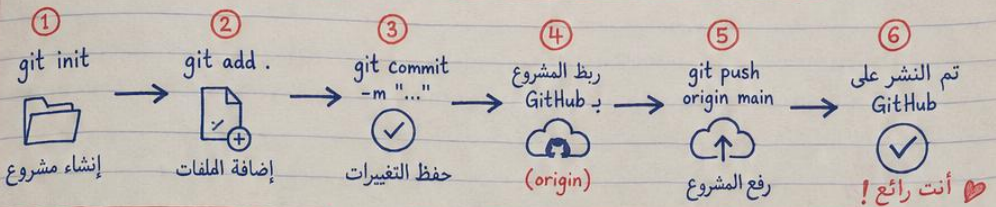
```
# my-project
(وصف واضح + خطوات + أمثلة + صور + و...)
build passing  license MIT  stars 1.2k
```

♡ إجعل مشروعك يتكلم عن نفسه

[Mostafa]

Full GitHub Workflow + Cheat Sheet - الملخص النهائي

الـ Workflow الكامل من أول مشروع لحد ما يتنزل على GitHub



☆ نصايح ذهبية للمبتدئين

- إعمل commit صغير ومركز .
- إكتب رسائل commit واضحة .
- إعمل pull قبل أي push .
- جرب وماتخافش من الأخطاء .
- إقرأ التوثيق الرسمي لـ Git .

! أكثر الأخطاء الشائعة وإزاي تتجنبها

- نسيـن git add . → الملفات مش هتترفع .
- رسائل commit غامضة → مستقبلك هيشكرك .
- pull بدون push → تعارض في التغييرات .
- التلاعب في التاريخ (reset) → خلي بالك في المشاريع المشتركة .
- رفع ملفات كبيرة أو غير مهمة → إستخدم gitignore .

نصيحة مهمة

Git مش هتتعلمه بالحفظ ،
هتتعلمه بالتطبيق يومياً ❤

☆ أهم أوامر Git (Cheat Sheet)

الأمر	المعنى
git init	إنشاء مستودع Git جديد في المجلد الحالي
git clone	نسخ (Clone) مستودع موجود من GitHub
git add .	إضافة كل الملفات للتجهيز للحفظ
git commit -m ""	حفظ التغييرات مع رسالة توضيحية
git push origin main	رفع التغييرات إلى GitHub (فرع main)
git pull origin main	جلب التغييرات من GitHub إلى جهازك
git branch	عرض الفروع في المشروع
git checkout -b	إنشاء فرع جديد والتنقل إليه
git merge	دمج فرع مع الفرع الحالي
git log --oneline	عرض سجل الـ commits بشكل مختصر
git status	عرض حالة الملفات والتغييرات
git reset --hard	الرجوع لحالة سابقة وحذف التغييرات
git revert	التراجع عن commit معين بشكل آمن
git stash	حفظ التغييرات مؤقتاً بدون commit

كل مشروع بتعمله على GitHub بيزود فرصك في الشغل ☆

الخطوات القادمة بعد ما تتعلم Git

- تعلم CI/CD (GitHub Actions) → لأتمتة المهام
- شارك في Open Source → طور مهاراتك وتعاون مع الأخرين
- ابني Portfolio قوي على GitHub → اعرض مشاريعك وأفكارك

Next Steps

- ☑ GitHub Actions
- ☑ Open Source
- ☑ Portfolio



استمر في التعلم ، كل سطر كود بتكتبه النهاردة
هو خطوة أقرب لحلمك بكرة ☆
أنت قدها !

[Mostafa]