KHATWA TECH • STUDY NOTES • 09_GITHUB_BRANCHES_PULL_REQUESTS

GitHub Collaboration

الشغل على GitHub: Branches وPull Requests وCode Review

مذكرة مراجعة تطبيقية: افهم المفاهيم، راجع الأمثلة، حل التدريبات، وبعدها نفّذ المشروع بنفسك.

الإصدار / المرجع: GitHub Docs • نوفمبر 2026 • Git CLI concepts

إعداد: محمود منصور (Mahmoud Mansour)

التاريخ: 24 نوفمبر 2026

المنصة: khatwa tech

---

youtube.com/@khatwatech

02 / 35

خريطة المذكرة | المحتويات

راجع الموضوعات حسب ترتيبها، ثم اختبر نفسك قبل قراءة الحلول.

القسم	الصفحات
GitHub Basics & Permissions	4–3
Repositories, Clone, Fork, Sync	8–5
Feature Branch Workflow	11–9
Pull Requests, Draft, Diff	15–12
Code Review	18–16
Merge & Conflicts	21–19
Rulesets, Checks, Workflow	24–22
مشروع الفريق	27–25
الأخطاء + Cheat Sheet	29–28
6 أسئلة + 6 تمارين	31–30
الإجابات + Edge Cases	34–32
المراجع	35

طريقة استخدام المذكرة في 3 خطوات

- اقرأ فكرة كل قسم وركّز على المثال، ثم نفّذه أو ارسمه بنفسك.
- غطّي خانة الناتج أو الحالة، وتوقّعها قبل ما تشوف الحل.
- حل 12 سؤالًا وتمرينًا دون الرجوع للإجابات، ثم اختبر الحالات الحدّية.

نسخة المراجعة

المصطلحات التقنية بالإنجليزي، والشرح بالعربي. أمثلة GitHub حالات محاكاة ما لم تُنفّذ على خدمة حقيقية.

03 / 35

01. GitHub مش هو Git

Git محلي للتحكم في التاريخ؛ GitHub منصة إدارة وتعاون.

- GitHub Repository يحتوي كود وIssues وPull Requests حسب صلاحيات المستخدم.
- Push يرفع Commits إلى Remote؛ لا يعني تلقائيًا فتح Pull Request.
- Code Review يساعد الفريق يراجع التغييرات قبل دمجها في main.

Git	GitHub
Commit محلي	Pull Request للتعاون
Branch محلي	Branch منشور على Remote
Merge محلي	Merge via PR مع Checks/Reviews

انتبه

PR ليس Commit ولا مجرد رابط؛ هو اقتراح تغييرات بين base وhead.

04 / 35

02. Repository وصلاحيات العمل

حدد مين يقدر يقرأ ويكتب ويراجع ويدمج.

- Public Repo يمكن لأي شخص قراءة محتواه العام؛ Private وصوله محدود حسب الصلاحيات.
- صلاحية Read لا تعني حق Push مباشر إلى مستودع الفريق.
- الـFork يسمح لك تقترح تعديلات لو لا تملك حق Write على الأصل.

الدور المفاهيمي	مثال عملي
Contributor	يعمل Branch أو Fork
Reviewer	يراجع Diff ويعلّق
Maintainer	يتأكد من شروط الدمج

انتبه

صلاحيات Merge وRulesets بتتغير حسب خطة المؤسسة وإعدادات المستودع.

05 / 35

03. README وملفات المشروع

مستودع قابل للتعاون محتاج وصفًا واضحًا.

- README يشرح فكرة المشروع، التشغيل، المتطلبات، وطرق المساهمة.
- .gitignore يمنع الملفات المولدة أو المحلية من الدخول في Commit.
- CONTRIBUTING.md يمكن تحديد Naming وPR workflow فيه.

مقتطف README.md TEXT

```
# Grades Tracker

Track student scores.

## Run
Open grades.csv.
```

الناتج / الحالة

```
(no output)
```

جرّب بنفسك

اكتب README فيه Setup وRun وTesting قبل ما تفتح أول PR.

06 / 35

Clone vs Fork .04

اختيار الطريقة يعتمد على صلاحياتك وهدفك.

- Clone: نسخة محلية من Repo، وغالبًا origin يشير إلى المصدر المستنسخ.
- Fork: نسخة على حساب/منظمة أخرى على GitHub.
- لمساهمة خارجية: Fork → Clone Fork → Branch → Push → PR إلى الأصل.

الحالة	الاختيار
لك صلاحية كتابة في Repo الفريق	Clone + Branch
مالكش Write على المشروع	Fork + PR
محتاج تشغيل محلي فقط	Clone دون PR

07 / 35

05. origin وupstream في Fork

ميز اسم الـRemote عن مصدر الحقيقة.

- origin غالبًا يشير إلى Fork الخاص بك بعد Clone.
- upstream اسم شائع للمستودع الأصلي للفريق.
- تأكد من git remote -v قبل أي Push عشان ما ترفعش للمكان الغلط.

أسماء عناوين توضيحية فقط BASH

```
git remote add upstream https://github.com/TEAM/grades.git
git remote -v
```

الناتج / الحالة

```
origin   <your-fork> (fetch/push)
upstream <team-repo> (fetch/push)
```

انتبه

عناوين TEAM/grades وyour-fork أمثلة تعليمية؛ مش روابط Repo مؤكدة.

08 / 35

Sync Fork .06

خلي Fork متوافق مع أحدث تغييرات الأصل.

- git fetch upstream يجلب التغييرات من المستودع الأصلي.
- git switch main ثم git merge --ff-only upstream/main عند نجاح Fast-forward.
- لو عندك Commits مختلفة على main، افهم سبب Divergence قبل الدمج.

تسلسل عام بعد إعداد الصلاحيات BASH

```
git fetch upstream
git switch main
git merge --ff-only upstream/main
git push origin main
```

الناتج / الحالة

نجاح الأوامر يعتمد على حالة تاريخ Git والوصول للـRemotes

جرّب بنفسك

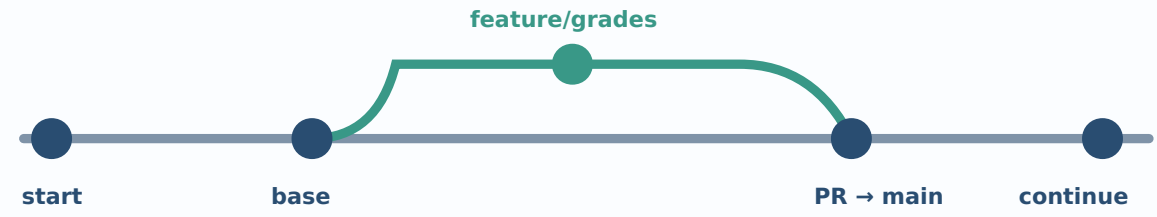
قبل Sync نفذ git log --oneline --graph --all لمعرفة الفرق بين فروعك والأصل.

09 / 35

Feature Branch Naming .07

كل مهمة واضحة في Branch مستقل.

- اسم مناسب: feature/grade-report أو fix/empty-list.
- فرع صغير بموضوع واحد أسهل في Code Review من PR ضخمة.
- ابدأ من main حديثة قدر الإمكان عشان تقل Merge Conflicts.



مسار تغييرات معزولة ثم مراجعتها ودمجها. الرسم مفاهيمي وليس سجل Commits حقيقيًا.

على Repo محلي BASH

```
git switch main
git switch -c feature/grade-report
git branch --show-current
```

الناتج / الحالة

```
feature/grade-report
```

الفرق اللي يهمك

Branch Naming Convention اختياري حسب الفريق، مش قاعدة مفروضة من GitHub.

Push Feature Branch .08

بعد Commit ادفع الفرع عشان GitHub يقدر يشوفه.

- Push لا ينشئ PR تلقائيًا؛ يرفع Commits فقط.
- git push -u origin feature/grade-report يحدد Upstream Tracking.
- لا تستخدم force push على فرع مشترك من غير اتفاق.

إذا كان origin يشير إلى GitHub Repo بصلاحية مناسبة

BASH

```
git add grades.csv
git commit -m "Add grade report"
git push -u origin feature/grade-report
```

الناتج / الحالة

Remote : مخرجات النقل تختلف حسب Commits تُرفع الـ

خطأ مشهور

git push origin main من Feature Branch مش نفس git push origin feature/grade-report.

09. Compare Branches

اختلاف Base وHead يحدد محتوى الـPR.

- Base: الفرع الذي تريد إدخال التغييرات إليه — غالبًا main.
- Compare/Head: الفرع الذي يحتوي تعديلاتك.
- لو بدلت الاتجاه بالغلط، محتوى الـDiff يتغيّر وقد يظهر عكس المطلوب.

خاصية	القيمة في مشروعنا
base	main
head	feature/grade-report
الملفات	grades.csv وREADME.md
الهدف	تقرير درجات جديد

انتبه

في PR من Fork، تأكد أيضًا من اسم المستودع لكل من base وhead.

10. إنشاء Pull Request

حوّل Branch منشور إلى طلب مراجعة واضح.

- من GitHub افتح مقارنة الفرعين ثم New Pull Request أو Compare & Pull Request.
- حدد base:main وhead:feature/grade-report، واكتب Title وDescription.
- اربط Issue إن وجد، وحدد Reviewer مناسبًا، ثم راجع قائمة الملفات.

اختياري باستخدام GitHub CLI بعد تسجيل الدخول BASH

```
gh pr create --base main --head feature/grade-report --title "Add grade report" --body "Adds grade summary output"
```

الناتج / الحالة

```
بعد الإنشاء؛ لم يُنفذ هنا GitHub يتحدد بواسطة PR URL الـ
```

انتبه

الغرض هنا توضيح أمر gh؛ يحتاج Repo حقيقي وAuthentication وصلاحيات.

11. PR Description واضح

الوصف هو عقد المراجعة: التغيير، السبب، والاختبارات.

- وضح What / Why / How Tested في 3 أقسام قصيرة.
- اذكر إن كان هناك Breaking Change أو قيود محتاجة تنبيه.
- ارفق لقطة أو Output لما التغيير في واجهة أو Console.

مثال Body للـPR — TEXT

```
## Summary
- Add grade report

## Testing
- Checked empty input
- Checked grades 0 and 100
```

الناتج / الحالة

Console ولـيس Description يظهر في Markdown نص

جرّب بنفسك

اكتب Body يتيح للمراجع تشغيل التعديل بدون سؤال إضافي.

12. Draft Pull Request

لما الكود لسه غير جاهز لمراجعة نهائية.

- Draft يوضح أن PR عمل تحت التنفيذ.
- يمكن مشاركته لطلب Feedback مبكر، لكن لا يُدمج وهو Draft.
- وقت اكتمال التنفيذ اختر Ready for review إذا كانت لديك الصلاحية.

الحالة	المعنى
Draft	قيد التنفيذ
Open / Ready	جاهز للمراجعة بحسب سياسة الفريق
Merged	التغييرات اتدمجت
Closed unmerged	اتقفل بدون دمج

> **انتبه**
> تغيير الحالة وحده لا يضمن قبول الـChecks أو Approvals.

13. Files Changed & Review Scope

راجع Diff قبل ما تطلب من الفريق يراجعه.

- Files changed تكشف كل الملفات التي تغيّرت بين Head وBase.
- تأكد مفيش Secrets أو ملفات Generated بالخطأ.
- لو PR تشمل أكتر من مشكلة منفصلة، الأفضل تقسيمها إن أمكن.

الملف	التغيير المتوقع
grades.csv	إضافة درجات
README.md	تعليمات تشغيل
env.	لا يُنشر ضمن PR

خطأ مشهور

صورة Diff جميلة لا تعني إن الاختبارات اشتغلت؛ راجع Checks بشكل مستقل.

14. Request Review

المراجعة عملية فنية، مش مجرد زر Approve.

- المراجع يتحقق من Correctness وReadability وTesting وSecurity.
- يمكن طلب Changes أو طرح Comments أو Approve.
- الفريق قد يطلب Approvals محددة قبل الدمج حسب Rulesets.

نوع المراجعة	الرسالة
Comment	ملاحظة عامة
Approve	المراجع موافق على التغيير الحالي
Request changes	فيه تعديل لازم يتصلح

15. Review Comments

اسأل أو اقترح تعديلًا محددًا ومهذبًا.

- Comment جيد يحدد File/Line، المشكلة، واقتراحًا قابلًا للتطبيق.
- ردّ صاحب PR بتوضيح أو Commit جديد ثم علّم المناقشة Resolved لو انتهت.
- فرق بين Nitpick أسلوبي وBug يؤثر على النتائج.

مثال Review Comment TEXT

```
Potential issue: empty grades list causes division by zero.
Suggestion: return 0.0 when list is empty.
```

الناتج / الحالة

ملاحظة في واجهة GitHub، لا Console Output

جرّب بنفسك

اكتب Comment على Function بتقسم مجموع الدرجات على len بدون فحص القائمة.

18 / 35

16. Updating an Open PR

نفس Branch المرفوع يمكن تحديث PR تلقائيًا بـCommits جديدة.

- عدل الملف محليًا على Head Branch، ثم git commit ثم git push.
- GitHub يحدّث Diff وChecks المرتبطة بالـPR المفتوح.
- بعض قواعد الحماية قد تبطل Approvals القديمة بعد Push جديد.

بعد ربط Upstream BASH

```
git switch feature/grade-report
printf "Sara,100\n" >> grades.csv
git add grades.csv
git commit -m "Cover upper boundary"
git push
```

الناتج / الحالة

إذا تم Push بنجاح PR جديد ضمن Commit سيظهر

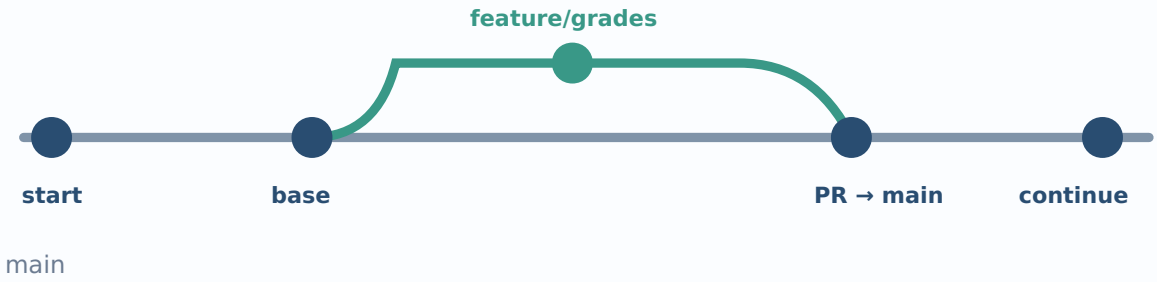
انتبه

لا تحتاج تفتح PR جديدة لكل Commit في نفس Feature Branch.

Merge Commit vs Squash vs Rebase .17

اختيار طريقة الدمج يغيّر شكل التاريخ، لا الهدف الوظيفي.

- Merge Commit يحافظ عادة على تاريخ الفرع والدمج.
- Squash ينتج Commit واحدة على base تجمع تغييرات PR.
- Rebase Merge يعيد تطبيق Commits على base بشكل خطي.
- الخيارات المتاحة تتحدد من إعدادات Repo وسياسة الفريق.



مسار تغييرات معزولة ثم مراجعتها ودمجها. الرسم مفاهيمي وليس سجل Commits حقيقيًا.

الطريقة	الفكرة
Merge commit	تاريخ الفرع + Commit دمج
Squash	Commit مجمعة واحدة
Rebase	تاريخ خطي مع إعادة تطبيق Commits

18. أسباب Merge Conflict

نفس الجزء في ملف واحد اتعدل بطريقتين متنافستين.

- Conflict مش معناه إن GitHub ضيع الكود؛ هو يطلب قرارًا من المطور.
- افحص الـDiff على الـBase والـFeature قبل حل التعارض.
- في بعض الحالات يمكن حل التعارض في GitHub، والحالات المعقدة محليًا.

شكل علامات التعارض للتوضيح TEXT

```
<<<<<<< HEAD
Passed: >= 50
=======
Passed: >= 60
>>>>>>> feature/rules
```

الناتج / الحالة

```
ده نص Conflict داخل الملف، وليس ناتج برنامج
```

تحذير

بعد حل التعارض اختبر السلوك الصحيح؛ حذف العلامات وحده مش كفاية.

19. حل Conflict محليًا

حل المشكلة بعد تحديث فرعك من base بالأسلوب المعتمد في الفريق.

- نفذ git fetch origin ثم Git merge/rebase بحسب سياسة الفريق.
- افتح الملفات المتعارضة واحذف markers بعد اختيار المنطق النهائي.
- بعد تعديل الملفات: git add؛ ثم أكمل merge أو rebase، واختبر وPush.

تسلسل عام عند وجود Conflict — BASH

```
git fetch origin
git switch feature/grade-report
git merge origin/main
# resolve conflict, edit files, then:
git add grades.csv
git commit
```

الناتج / الحالة

نجاح الدمج ومحتوى الرسالة يعتمد على حالة الـBranches

انتبه

لو الـRepo تفرض Rebase بدل Merge اتبع سياستها؛ لا تنفذ الأمرين معًا بلا داعٍ.

20. Protected Branches & Rulesets

سياسات قابلة للتطبيق لفرض معايير جودة قبل الدمج.

- يمكن طلب PR قبل الدمج إلى main، ومنع Force Push حسب الإعداد.
- Rulesets أو Branch Protection يمكن أن تطلب Review وChecks.
- تختلف الإمكانيات المتاحة حسب Repo Plan وOwner وAdmin Permissions.

القاعدة	الأثر
Require pull request	منع دمج مباشر وفق الإعداد
Required approvals	طلب موافقات قبل الدمج
Require status checks	الاختبارات المطلوبة لازم تنجح
Block force pushes	حماية التاريخ على فروع محددة

انتبه

Approval واحدة لا تتخطى Check فاشل لو مطلوب.

21. Checks وCI Status

Check Run إشارة قابلة للفحص، مش إثباتًا وحيدًا للجودة.

- Pull Request قد ترتبط بـGitHub Actions أو CI خارجي.
- Passed يعني Check معينة نجحت؛ Failed يعني محتاج إصلاح.
- لو Required Check Pending، غالبًا Merge يظل متوقفًا حتى يكتمل.

الحالة	القرار المعتاد
Success	يمكن الانتقال إن باقي الشروط متحققة
Failure	أصلح المشكلة ثم Push
Pending	انتظر اكتمال الفحص
Skipped	راجع هل القاعدة تتطلب هذه Check

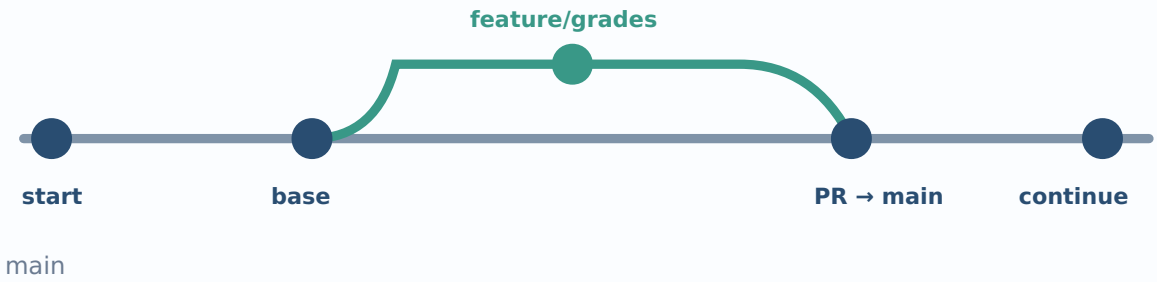
الفرق اللي يهمك

Checks الفعلية وسياسة الدمج من GitHub؛ الأمثلة في المذكرة محاكاة.

22. Workflow متكامل من البداية للنهاية

اربط كل الخطوات في سلسلة قابلة للتطبيق.

- Issue → Branch → Code → Commit → Push → PR → Review → Checks → Merge.
- اختبر محليًا قبل Push ثم راجع Checks بعد Push.
- بعد Merge حدث main عندك واحذف Branch القديمة لو لم تعد مطلوبة.



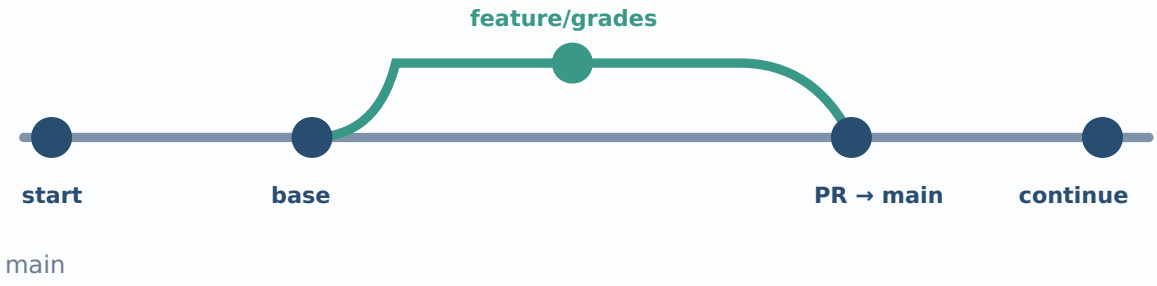
مسار تغييرات معزولة ثم مراجعتها ودمجها. الرسم مفاهيمي وليس سجل Commits حقيقيًا.

المرحلة	المسؤول
Code & Push	Author
Review	Reviewer
Checks	CI / GitHub
Merge	Person with permission

المشروع | 1 — فريق Grades Tracker

محاكاة عملية لثلاثة مطورين على مستودع واحد.

- Owner يحمي main بسياسة Review + Checks حسب الإعدادات المتاحة.
- Developer A يضيف Student Grade Report على feature/report.
- Developer B يصلح Empty Grades على fix/empty-input.
- Developer C يراجع PRs ويحدد ما يستحق الموافقة.



مسار تغييرات معزولة ثم مراجعتها ودمجها. الرسم مفاهيمي وليس سجل Commits حقيقيًا.

PR	الهدف	المراجعة المتوقعة
feature/report #1	إضافة تقرير	اختبار 0 و100
fix/empty-input #2	منع القسمة على صفر	اختبار قائمة فاضية

تنبيه

الأرقام #1 و#2 افتراضية داخل سيناريو تعليمي، لم تُنشأ PR حقيقية.

المشروع | 2 — أفرع محلية ونص الـPR

نفّذ جزء Git المحلي واكتب وصف PR قابلًا للمراجعة.

محاكاة محلية في مجلد جديد BASH

```bash
git init -b main
git config user.name 'Student'
git config user.email 'student@example.test'
printf 'name,grade\nAli,90\n' > grades.csv
git add grades.csv && git commit -m 'Start project'
git switch -c feature/report
printf 'Mona,75\n' >> grades.csv
git add grades.csv && git commit -m 'Add report data'
git branch --show-current
```

الناتج / الحالة

```
feature/report
```

نموذج PR Description TEXT

```
## Summary
- Add grade report data
## Testing
- Check score = 0
- Check score = 100
- Check empty list
```

الناتج / الحالة

```
نص Markdown وليس Console Output
```

المشروع | 3 — مراجعة افتراضية ونتيجة الدمج

انتقل من العمل الفردي إلى قرار فريق واضح.

المرحلة	الحالة في السيناريو	معنى الحالة
PR #1	Checks passed, approved	جاهز إذا لم توجد شروط أخرى
PR #2	Checks failed	ممنوع دمجه مع Required Checks
PR #2 بعد الإصلاح	Checks passed, approved	يمكن دمجه إذا باقي الشروط اتحققت

3 أفكار للتطوير

1) أضف PR Template. 2) جهز GitHub Actions لتشغيل اختبار. 3) حاكي Conflict بين تقريرين على نفس السطر.

التوضيح المهم

هذه حالات افتراضية للتعلم وليست Status Checks أو Reviews مضافة فعليًا على GitHub.

28 / 35

الأخطاء الشائعة على GitHub

عالج السبب أولًا قبل Force Push أو تجاهل Checks.

الرسالة / الحالة	السبب المحتمل	تعمل إيه
No changes between branches	Base/Head متماثلان أو اختيارهما خطأ	راجع اتجاه الـPR والـCommits
Merge blocked	Required Review أو Check ناقص	افحص Rulesets والـChecks
Conflicts must be resolved	تعارض مع Base	حدث فرعك وحل التعارض
Permission denied	مافيش Write أو Auth	Fork أو اطلب صلاحية
Draft PR	الـPR غير Ready	Mark ready when done
Review requested changes	المراجع يطلب تعديل	عدّل وPush وأعد الطلب

ورقة GitHub Collaboration Cheat Sheet

سلسلة العمل اليومية في صفحة.

عايز أعمل كذا	الخطوة / الأمر
تنشئ فرعًا	git switch -c feature/name
ترفع فرعًا	git push -u origin feature/name
تفتح PR	Base:main ← Head:feature/name
تطلب Review	Reviewers في PR
تراجع التغييرات	Files changed
تحدث PR	Commit + Push لنفس Head Branch
توقف الدمج مؤقتًا	Draft PR
تطبق Policy	Ruleset / Branch Protection
تعالج Conflict	Fetch, merge/rebase, resolve, push
تختار طريقة الدمج	Merge / Squash / Rebase حسب إعداد Repo

30 / 35

اختبر نفسك | 6 توقّع الحالة

جاوب في شكل قرار + السبب.

س01 هل git push تلقائيًا ينشئ Pull Request؟

س02 إذا PR Draft وChecks ناجحة، هل يمكن دمجها وهي Draft؟

س03 base=main وhead=feature/x؛ أيهما الهدف؟

س04 Required Check = Failed وReview = Approved؛ هل الدمج متاح؟

س05 Push Commit جديد إلى Head Branch؛ ما الذي يحدث للـPR المفتوح؟

س06 لا تملك Write في Repo عام؛ كيف تقترح تغييرًا؟

31 / 35

اختبر نفسك | 6 تمارين عملية

كل تمرين له Input أو حالة وOutput متوقع.

ت01 رتّب خطوات المساهمة من Branch إلى Merge

Input → Output: Branch → Commit → Push → PR → Review → Checks → Merge

ت02 اكتب PR Description لميزة Grade Report

Input → Output: Summary + Testing + Notes

ت03 حدد base وhead لتغيير في feature/report

Input → Output: base=main, head=feature/report

ت04 عالج Review Comment عن Empty List

Input → Output: أضف شرط len==0 واختبر ثم Push

ت05 اذكر سببين لـMerge Blocked رغم وجود Approval

Input → Output: Required Check failed; Conflict

ت06 اختر طريقة دمج تجعل PR كلها Commit واحدة

Input → Output: Squash Merge

32 / 35

الإجابات | توقّع 1–3

حلول من منطق GitHub، لا من التخمين.

س01 لا، ينشر Branch/Commits فقط.

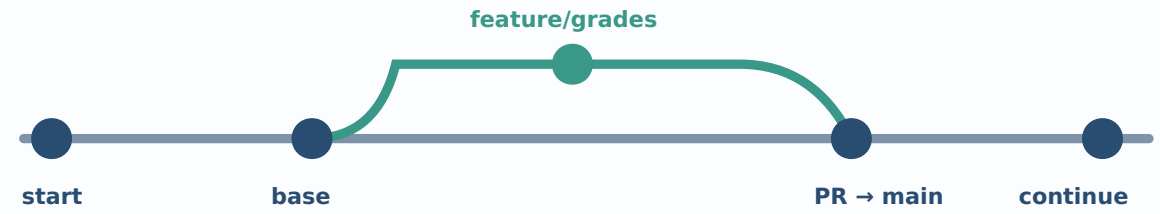
هل git push تلقائيًا ينشئ Pull Request؟

س02 لا، لازم تحولها إلى Ready قبل الدمج.

إذا PR Draft وChecks ناجحة، هل يمكن دمجها وهي Draft؟

س03 main هو الهدف (Base).

base=main وhead=feature/x؛ أيهما الهدف؟



مسار تغييرات معزولة ثم مراجعتها ودمجها. الرسم مفاهيمي وليس سجل Commits حقيقيًا.

قاعدة

Base هو المكان الذي ستدخل إليه التعديلات. Head هو مصدرها.

الإجابات | توقّع 4-6 + ت1 وت2

حلول عملية للـWorkflow والوصف.

س04 لا، Check مطلوب فاشل.

س05 يتحدث Diff وChecks وفق إعداد Repo.

س06 Fork ثم Branch ثم PR إلى الأصل.

حل ت2 — PR Description كامل

```text
## Summary
Add a student grade report.

## Testing
- Score 0: Passed? No
- Score 100: Passed? Yes
- Empty list: Report "No grades"
```

الناتج / الحالة

Output ليس PR، وصف Markdown

حل ت1

Branch → Commit → Push → PR → Review → Checks → Merge. قد تعمل Checks تلقائيًا قبل اكتمال Review؛ الترتيب يعبر عن مراحل قرار الدمج.

الإجابات | ت3–ت6 وEdge Cases

قرارات الدمج لا تعتمد على Approval وحدها.

تمرين	الحل
ت3	base=main / head=feature/report
ت4	Fix empty list, add tests, commit and push
ت5	Merge Conflict أو Required Checks failed
ت6	Squash and merge إذا كانت مفعلة في Repo

Edge Case	النتيجة الصحيحة
Draft رغم Success	لا دمج حتى Ready
Approval + Failed Check	يُحجب الدمج لو Check Required
فرع بدون اختلاف	لا Changes للـPR
Fork بدون Write على الأصل	PR من Fork بدل Push للأصل
Push بعد Approval	قد تحتاج Review جديدة حسب Ruleset
Merge Conflict	يحتاج Resolution واختبار

الخطوة العملية

جهز Repo حقيقي لاحقًا لتنفيذ الأوامر وملاحظة الحالات الحقيقية بدل الاعتماد على المحاكاة.

المراجع والخطوة التالية

تعلم Workflow مباشرةً من توثيق GitHub الرسمي.

المراجع الرسمية

- About Pull Requests
- Creating a Pull Request
- Reviewing Pull Requests
- About Protected Branches
- Available Rulesets
- Merging a Pull Request

ملاحظة جودة

تجارب Git المحلية تمت دون الوصول إلى حساب GitHub. PR وApprovals وRequired Checks هنا سيناريوهات موثقة غير منفذة على الخدمة.

بعد المذكرة

اعمل Repo تدريبي بـREADME وPR Template، وافتح Branch ثم PR صغيرة مع مراجعة فعلية.