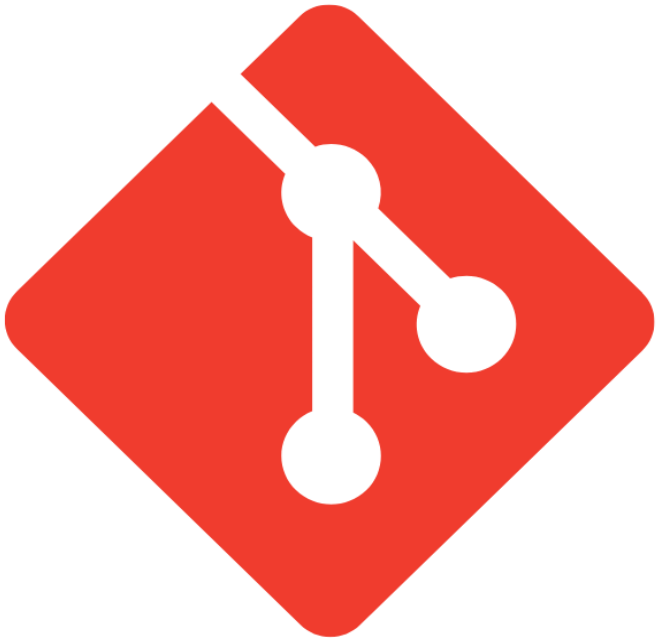
# Git & GitHub



# Git

## ما هو Version control ؟

- أداة تُدير التغيرات المتعلقة بالأكواد والملفات داخل المشروع.
- أداة تجعلك تتبع جميع التعديلات بالتوقيتات الزمنية لكل تعديل.
- أداة تُتيح لك تخزين نسخة من الملفات على جهازك الشخصي أو على الاداة المستخدمة لتطبيق الـ version control مثل Git.

## ما هو Git؟

هو Software يُستخدم لتطبيق الـ version control وهو من أشهر أنواع الـ version control.

يوجد العديد من البرامج المختلفة التي تُستخدم لتطبيق الـ version control مثل:

- CVS

- SVN

- Perforce

- Bazzare

## أبرز تعليمات ومصطلحات ال Git:

### Git init

- ال git init هي أمر ُيستخدم في Git لإنشاء مستودع (Repository) جديد. يتم استخدام هذا الأمر عندما تحتاج إلى إنشاء مستودع (Repository) جديد لمشروعك الحالي أو لبدء مشروع جديد.
- عندما تقوم بتشغيل git init يقوم Git بإنشاء مجلد جديد في المسار الحالي يسمى ".git". يتم استخدام هذا المجلد لتخزين جميع بيانات Git المتعلقة بمستودعك (Repository)، مثل تاريخ المشروع وسجلات الإصدارات وغيرها من المعلومات.

### git add

- يستخدم هذا الأمر لإضافة الملفات المحددة إلى منطقة الانتظار (area staging)، والتي تسمح لك بتحديد الملفات التي سيتم تضمينها في ال commit القادم.
- لإضافة ملفات جديدة يمكنك استخدام git add متبوعاً بأسماء الملفات.

على سبيل المثال:

```
git add file1.txt file2.txt.
```

## git commit

- يستخدم هذا الأمر لحفظ التغييرات التي تمت إضافتها إلى منطقة الانتظار (staging area) وتطبيقها على المستودع (Repository) الخاص بك.
- لإنشاء commit جديد يمكنك استخدام commit git متبوعاً برسالة commit وصفاً للتغييرات التي قمت بها. على سبيل المثال:

  m- commit git "أضفت ملفات جديدة"
- يتم تخزين التغييرات التي تم إجراؤها في commit بشكل دائم في المستودع (Repository) الخاص بك، مما يسمح لك بالعودة إليها في أي وقت لتحديث المشروع أو استعادة نسخة سابقة منه.

## git log

- هي أمر في Git يستخدم لعرض سجل التزامات (Commits) المستودع (Repository).
- يعرض هذا الأمر قائمة بكل الالتزامات (Commits) الموجودة في الفرع الحالي للمستودع (Repository)، ويشمل كل التزام (Commit) معلومات مثل الشفرة التعريفية للالتزام ( hash commit) والمؤلف والتاريخ والرسالة التوضيحية للالتزام (Commit).
- بشكل عام، Git log يمكن استخدامه لفهم تاريخ المستودع وتحليل تطوراته بشكل أفضل، كما يمكن استخدامه كأداة لتحديد مكان الأخطاء ومراجعتها.

## .gitignore

- هو ملف يتم إنشاؤه داخل مجلد العمل الخاص بمشروع Git ويحتوي على قائمة بالملفات والمجلدات التي يجب تجاهلها عند تتبع تغييرات المشروع باستخدام Git.
- عندما تنشئ مستودع Git جديداً، ستضع ملفاتك ومجلداتك داخل مجلد العمل، وستقوم بتتبع تغييرات هذه الملفات باستخدام Git. ومع ذلك، قد يكون هناك ملفات أو مجلدات غير ضرورية أو غير مرغوب فيها، مثل ملفات التكوين أو ملفات تضم معلومات حساسة أو ملفات مؤقتة. يمكن استخدام ملف gitignore. لتعليم Git تجاهل هذه الملفات، حتى لا يتم تتبعها ضمن سجلات التاريخ والإصدارات.

## Branches

- في ال Git، تعرف الـ" Branches" بأنها تسمح لك بتطوير مسارات مختلفة للتغييرات على مستودع Git الخاص بك.
- عند إنشاء فرع (Branch) جديد يمكنك تطوير أجزاء أو إجراء التغييرات دون التأثير على الأجزاء الأخرى التي تعمل عليها فرق العمل الأخرى في نفس المشروع. أي أنك سوف تأخذ نسخة من المشروع وتقوم بالتعديل عليه ولن يتأثر المشروع الرئيسي بأي تعديل تقوم به، وبعد أن تتأكد أن ما قمت بإضافته أو تعديله مضبوط تستطيع أن تقوم بإضافته للمشروع الرئيسي.
- هو أمر في Git ُيستخدم لإنشاء فروع (Branches) جديدة في مستودع Git الخاص بك. عندما تنشئ فرعاً (Branch) جديداً يتم إنشاء نسخة من كل شيء موجود في الفرع الحالي، بما في ذلك جميع الملفات وسجل التاريخ الخاص بالمستودع، وتستخدم هذه الفروع لتجربة، وتطوير ميزات جديدة بدون التأثير على الفرع الرئيسي.
  - لإنشاء فرع جديد، يمكنك استخدام الأمر git branch متبوعاً بـ اسم الفرع الجديد، على سبيل المثال:

    git branch new-feature

## Git Checkout

- هو أمر في Git يستخدم للتبديل بين الفروع المختلفة في مستودع Git الخاص بك. عند تشغيل هذا الأمر مع اسم الفرع، يتم التحويل إلى هذا الفرع ويصبح الفرع النشط الذي يتم العمل عليه.
- للتبديل إلى فرع معين، يمكن استخدام الأمر git checkout متبوعاً بـ اسم الفرع، على سبيل المثال:

```
git checkout new-feature
```

- بمجرد التبديل إلى الفرع الجديد، يمكنك العمل على الملفات وإجراء التغييرات كما تشاء، دون التأثير على الفرع الرئيسي أو أي فروع أخرى.

## Git Revert

- ال Git Revert لا يحذف أي تعديلات سابقة بل ُينشئ الالتزام (commit) الجديد الذي يعكس حالة المستودع قبل إجراء التعديلات التي تم إجراؤها في الالتزام (commit) المحدد.
- بمعنى آخر يتم إنشاء الالتزام الجديد الذي يحتوي على تعديلات تعكس الحالة السابقة للمستودع.
- لمعرفة المزيد عن كيفية استخدام Git Revert، دعنا نلقي نظرة على المثال التالي:

```
git revert abc123
```

## Git Switch

- هو أمر في Git يستخدم للتبديل بين الفروع أو الالتزامات (commits) المختلفة في مستودع Git الخاص بك. وهذا الأمر يقوم بوظيفة كال من git branch وgit checkout معا.
- عند تشغيل هذا الأمر مع اسم الفرع أو معرف الالتزام (commit) يتم التحويل إلى هذا الفرع أو الالتزام (commit) ويصبح الفرع أو الالتزام (commit) النشط الذي يتم العمل عليه.
- للتبديل إلى فرع أو التزام (commit) معين، يمكن استخدام الأمر git switch متبوعاً بـ اسم الفرع، على سبيل المثال:

```
git switch -c new-feature
```

## git branch -d branchname

- يستخدم أمر "git branch –d branchname" لحذف فرع Git من مستودع Git .
- عند استخدام الأمر " git branch –d branchname "، ستقوم Git بحذف الفرع المحدد (branchname) من مستودع Git الحالي. وتحتاج إلى التأكد من أنك تستخدم هذا الأمر بحذر، لأنه إذا قمت بحذف الفرع الذي تعمل عليه دون قصد، فقد تفقد العمل الذي تم إنجازه في ذلك الفرع.

## git merge

- يتم استخدام الأمر"git merge" لدمج فرع Git محدد بفرع آخر في مستودعGit .
- يسمح لك الدمج بإضافة التغييرات من فرع إلى آخر .عندما تقوم بدمج فرعين في Git باستخدام الأمر"git merge" ، يتم إنشاء "commit" جديد يحتوي على جميع التغييرات من الفرع الأول والفرع الثاني.
- يتم استخدام الأمر"git merge" بالشكل التالي:

```
git merge <branchname>
```

- حيث أن" branchname" هو اسم الفرع الذي تريد دمجه بالفرع الحالي. ومن المهم التأكد من التحديث (الانتقال) إلى الفرع الرئيسي قبل استخدام هذا الأمر، وذلك باستخدام الأمر.

```
git checkout <main-branch>
```

## fast-forward merge

- يشير المصطلح" merge forward-fast" إلى نوع من الدمج في Git يتم فيه دمج فرع (branch) إلى الفرع الرئيسي في حالة عدم وجود تغييرات في الmain .
- وبمعنى آخر يتم دمج التغييرات في الفرع الفرعي مباشرةً مع الفرع الرئيسي دون إنشاء نقطة تفرع جديدة.
- يتم استخدام هذا النوع من الدمج عندما يتم العمل على فرع فرعي بشكل منفرد وبدون أي تداخلات مع الفرع الرئيسي.

## 3-way merge

- هو نوع آخر من دمج الفروع في Git، ويتم استخدامه في حالة وجود تغييرات في الmain .
- في هذه الحالة يقوم Git بإنشاء نقطة تفرع جديدة (commit merge) ويحاول دمج التغييرات من الفرعين.

## Squash Merge

- يعد Squash Merge أحد الطرق التي يتم بها دمج الفروع في Git.
- يستخدم هذا الأسلوب عندما تريد دمج التعديلات التي تم إجراؤها في فرع معين إلى فرع آخر.
- يعد Squash Merge طريقة جيدة لتقليل عدد الالتزامات (commits) الزائدة في تاريخ (History) مشروع Git، وللمحافظة على التاريخ (History) الخاص بالمشروع أنيقاً ومنظماً.
- من المهم ما لحظة أنه إذا كنت تستخدم Squash Merge في Git فيجب عليك التأكد من الحفاظ على تاريخ المشروع مرتباً ومنظماً، وذلك باستخدام رسائل الالتزامات الصحيحة ووصف دقيق للتعديلات المدمجة.

## Merge conflicts

- يعد Merge conflicts أو دمج التعارضات جزءاً من عملية الدمج (merge) في Git.
- تحدث عملية conflicts Merge عندما يتم تعديل نفس الملف أو الجزء في فرعين مختلفين بطريقة مختلفة من قبل أكثر من مستخدم في نفس الوقت، وهذا يؤدي إلى وجود تعارضات بين التعديلات المختلفة التي يجب حلها.
- عند حدوث التعارضات يقوم Git بإبلاغ المستخدم بالتعارضات ويطلب منه إجراء اللازم لحل هذه التعارضات.
- يمكن حل التعارضات بطرق مختلفة مثل دمج التعديلات المختلفة بشكل يدوي، أو اختيار إحدى التعديلات على حساب الأخرى، أو استخدام أدوات تلقائية لدمج التعديلات، ويعتمد الخيار المناسب على طبيعة التعارض وحجم الملف وتفضيلات المستخدم. وعند الانتهاء من حل التعارضات، يتم حفظ النسخة الجديدة.

# مقارنة بين مميزات وعيوب Git merge وGit rebase

## Git merge

| المميزات | العيوب |
|---|---|
| يحافظ على تاريخ المشروع الخطي (Linear)، مما يسهل تتبع التغييرات وفهم عملية التطوير. | يمكن أن يؤدي إلى تاريخ التزامات (Commits) فوضوي ومربك عند دمج فروع متعددة. |
| لا يقوم بتغيير الفرع (Branch) الأصلي أو تاريخ الالتزامات (Commits) الأصلي. | يمكن أن يؤدي إلى إنشاء علامات دمج غير ضرورية ولا تساهم في عملية التطوير. |
| هو طريقة بسيطة ومباشرة لدمج التغييرات من فرع واحد إلى آخر. | قد يتطلب وقتاً وجهداً إضافياً لحل تضاربات الدمج. |

## Git rebase

| المميزات | العيوب |
|---|---|
| يوفر تاريخ التزامات (Commits) أكثر نظافة وتنظيماً عن طريق دمج التغييرات من فرع واحد في آخر بدون إنشاء علامات دمج غير ضرورية. | يقوم بتغيير تاريخ الالتزام (Commit) الأصلي، مما يمكن أن يجعل من الصعب تتبع التغييرات وفهم عملية التطوير. |
| يتيح عملية تطوير أكثر تيسيراً وخطية | يمكن أن يؤدي إلى تاريخ أكثر تعقيداً، خاصة عند إعادة ترتيب فروع كبيرة أو طويلة الأمد. |
| يقلل من عدد تضاربات الدمج التي يتعين حلها، حيث يتم دمج التغييرات في تاريخ خطي. | يمكن أن يؤدي إلى مشكلات في الفروع المشتركة، حيث يتم إعادة كتابة التاريخ ويمكن أن يسبب تضاربات للمطورين الآخرين. |

بشكل عام، git merge هو خيار جيد عندما تريد الحفاظ على تاريخ المشروع بشكل واضح ومباشر.

# GitHub

- يقوم بتوفير بيئة عمل مرنة يمكن للمطورين استخدامها للتحكم في إدارة مشاريعهم، ومشاركة الأعمال والتعاون مع الآخرين في المشاريع المشتركة، وإدارة الإصدارات والتعديلات على الكود المصدري، وإدارة الأخطاء والعيوب وتتبع الأعمال.
- الكود الذي تقوم بكتابته يكون محلي (Local) لا أحد يمكنه رؤيته غيرك ولكن أحيانا تريد مشاركته مع الآخرين وهنا يأتي دور Github الذي يجعلك تقوم بإنشاء مستودع (repository) عليه وتحميل الكود المحلي إليه، ويمكنك مشاركة المستودع (repository) مع الآخرين عن طريق إرسال رابط المستودع (repository) لهم.
- وبمجرد أن يتمتع أي شخص بصلاحيات الوصول إلى المستودع، يمكنه القيام بعمليات التحرير والتعديل والتحديثات على الكود المصدري وإرسالها مرة أخرى للمستودع (repository).

**اتبع الخطوات التالية لرفع مشروعك علي Github.**

git init git

add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/userName/repoName.git

git push -u origin main

## git init

هذا الأمر يبدأ مشروع Git جديد. ينشئ Git مجلداً جديداً باسم ".git" والذي يحتوي على جميع الأدوات التي يحتاجها Git لتتبع التغييرات في المشروع.

## git add README.md

هذا الأمر يقوم بإضافة ملف README.md.

## git commit -m "first commit"

يستخدم هذا الأمر لتأكيد التغييرات في مشروع Git . يتم تضمين الملفات التي تم إضافتها إلى منطقة الانتظار في عملية التأكيد (commit)، ويتم وصف التغييرات في رسالة التأكيد.

## git branch -M main

يقوم هذا الأمر بإعادة تسمية الفرع الافتراضي الخاص بالمستودع من "master" الى "main".

## git remote add origin https://github.com/userName/repoName.git

يستخدم هذا الأمر لإضافة اسم الخادم البعيد (remote repository) الذي تم ربط المشروع به إلى Git . يتم استخدام "origin" كاسم الخادم البعيد (remote repository) في هذا المثال.

## git push -u origin main

يقوم هذا الأمر بإرسال التغييرات المؤكدة إلى الخادم البعيد (remote repository).

## Git Fetch

يستخدم لجلب التحديثات الجديدة من المستودع البعيد (remote repository) إلى المستودع المحلي ( local repository)، لكنه لا يقوم بدمج هذه التحديثات مع النسخة المحلية، ولذلك يجب أن تقوم بتنفيذ الأمرgit merge .

## Git Pull

يستخدم لجلب التحديثات الجديدة من المستودع البعيد (remote repository) و دمجها مع النسخة المحلية.

يقوم Git Pull بالقيام بنفس العمل الذي يقوم به Git Fetch، ولكن يضيف خطوة إضافية وهي تنفيذ الأمر

git merge لدمج التحديثات الجديدة مع النسخة المحلية بشكل تلقائي.

يمكن استخدام Git Fetch عندما تريد فقط الحصول على تحديثات جديدة دون دمجها مع النسخة المحلية، بينما يمكن استخدامGit Pull عندما تريد جلب التحديثات الجديدة ودمجها مع النسخة المحلية.

## Fork

يعني إنشاء نسخة من مستودع Git الحالي وتخزينه في حساب Git آخر.

عادةً ما يتم استخدام هذا الأمر عند الرغبة في إضافة تحسينات أو تعديلات لمشروع Git موجود على الإنترنت، حيث يتم إنشاء نسخة من المستودع الأصلي وتعديلها دون التأثير على المستودع الأصلي.

يمكن لأي شخص إنشاء نسخة من المستودع وإجراء التعديلات التي يريدها وإرسالها إلى المستودع الأصلي للمشروع من خلال إرسال طلب دمج Pull Request.

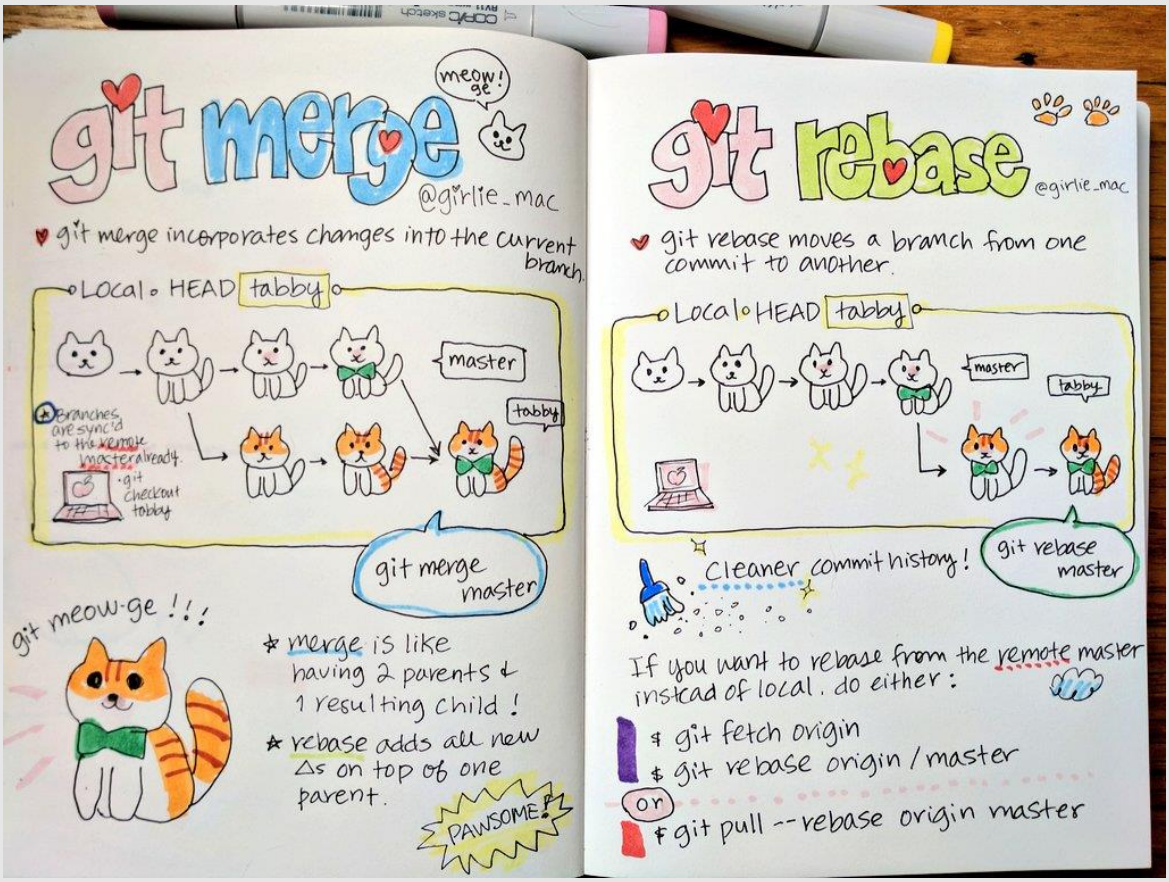
## Clone

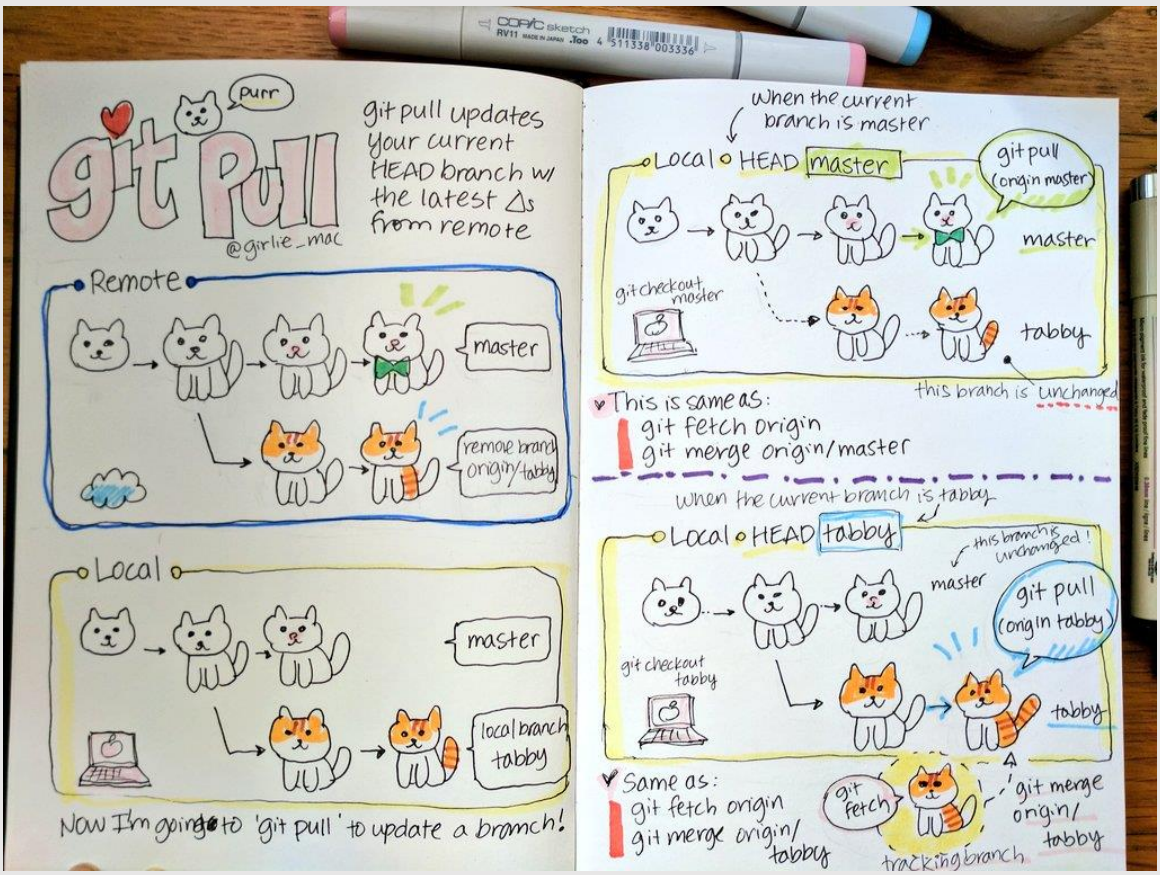
يعني إنشاء نسخة من مستودع Git الحالي وتخزينها على جهاز الكمبيوتر الخاص بك.

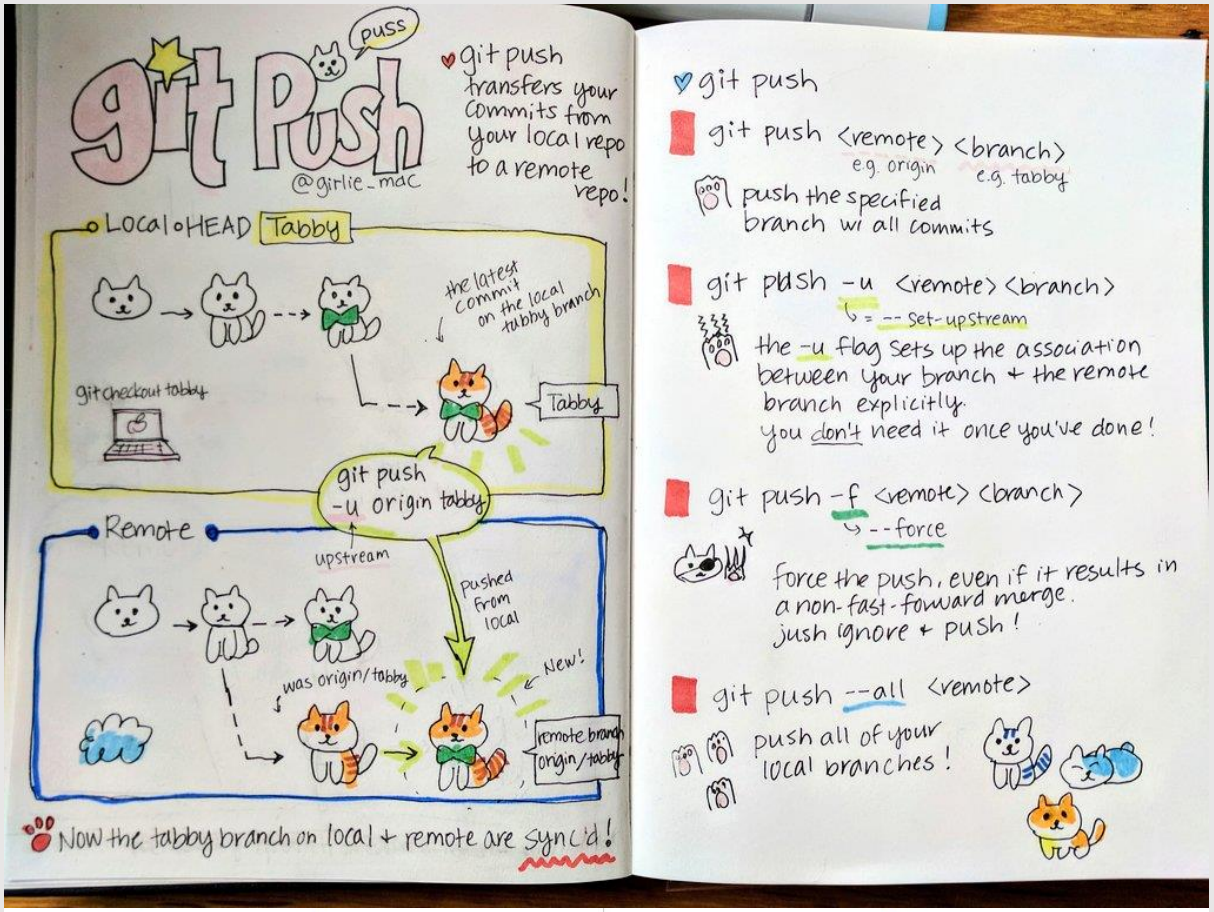
يتم استخدام هذا الأمر عادةً عند الرغبة في الحصول على نسخة من مستودع Git موجود على الإنترنت على جهاز الكمبيوتر الخاص بك لتتمكن من العمل عليها بشكل محلي.

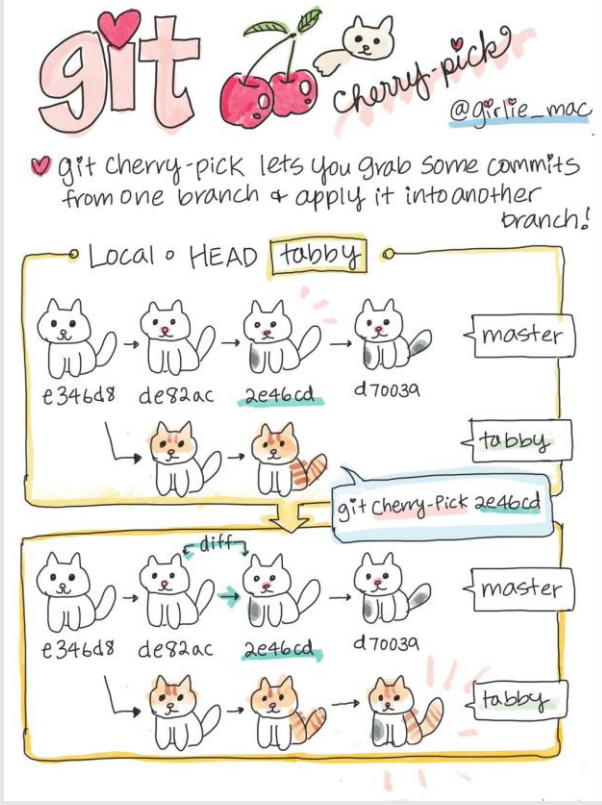
يمكن استخدام الأمر clone عندما ترغب في الحصول على إصدار محدث من المستودع بعد إجراء تغييرات من قبل فريق التطوير.

# رسوم توضيحية لبعض التعليمات




git Push
puss
@girlie_mac
git push transfers your commits from your local repo to a remote repo!
Local HEAD Tabby
the latest commit on the local tabby branch
git checkout tabby
Tabby
git push -u origin tabby
upstream
Remote
Pushed from local
was origin/tabby
New!
remote branch origin/tabby
Now the tabby branch on local + remote are sync'd!
git push
git push <remote> <branch>
e.g. origin
e.g. tabby
push the specified branch w/ all commits
git push -u <remote> <branch>
= --set-upstream
the -u flag sets up the association between your branch + the remote branch explicitly.
you don't need it once you've done!
git push -f <remote> <branch>
--force
force the push, even if it results in a non-fast-forward merge.
just ignore + push!
git push --all <remote>
push all of your local branches!


git cherry-pick
@girlie_mac
git cherry-pick lets you grab some commits from one branch & apply it into another branch!
Local HEAD tabby
e346d8
de82ac
2e46cd
d70039
master
tabby
git cherry-pick 2e46cd
diff
master
tabby


git log
git log lets you view the commit history
@girlie_mac
e346d8
de82ac
2e46cd
d70039
SHA-1 Checksum
git log prints out
commit e346d8
Author: Little-princess Leia <leia@kitty.cat>
Date: Sun Sept 23 17:30:42 2018 -0700
Add body
commit de82ac
Author: Jamie <jam@kitty.cat>
Date :
Simpler log with:
git log --oneline
e346d8 Add body
de82ac Edit nose
2e46cd Add gray dot on leg
git cherry pick
this SHA into the tabby branch