

Git&Github

واجهات المستخدم (User Interfaces)

من أشهر أنواع واجهات المستخدم :(User Interfaces)

- **واجهات GUI (Graphical User Interface)**

هي واجهة تستخدم العناصر الرسومية مثل: windows, menus, icons) لتشغيل البرامج و إدارة ملفات الكمبيوتر و التفاعل مع الكمبيوتر.

- **واجهات CLI (Command Line Interface)**

هي واجهة تستخدم الأوامر النصية (commands) لتشغيل البرامج و إدارة ملفات الكمبيوتر و التفاعل مع الكمبيوتر.

<https://satr.codes/courses/5f87136c-c85f-40fd-98b3-e31ef2961c16/session/7926d9e6-db72-446b-a75e-118aa3d98b8b/view>



• منصة سطر التعليمية

أوامر CLI

أهم الأوامر المستخدمة في Command Line Interface

وظيفة الأمر	Windows(command prompt) الأمر في نظام	الأمر في نظام (terminal)
عرض المسار الحالي	cd	pwd
الخروج من المجلدات	cd ..	cd ..
إنشاء مجلد ملاحظة: يمكنك استخدام علامة	mkdir <i>folderName</i>	mkdir <i>folderName</i>

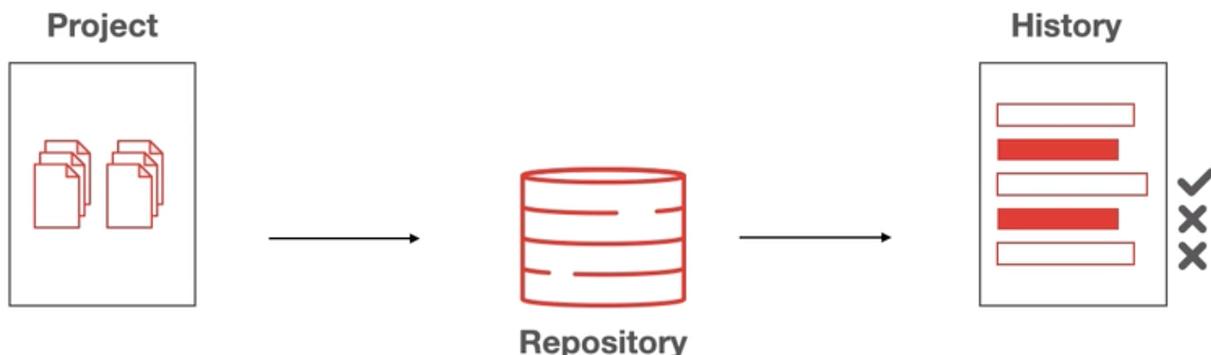
الاستفهام إذا كان ؟ اسم الملف أو المجلد يحتوي مسافة.		
الدخول إلى المجلدات	cd <i>directoryName</i>	cd <i>directoryName</i>
إنشاء ملف نصي فارغ	cd > <i>fileName.txt</i> OR type nul > <i>fileName.txt</i>	touch <i>file.txt</i>
عرض محتويات المجلد	dir	ls
عرض جميع محتويات المجلد بما فيها المحتويات المخفية	dir /a:h	ls -a
الكتابة في ملف نصي	echo My First File >> <i>fileName.txt</i>	echo "My First File" >> <i>fileName.txt</i>
عرض محتويات الملف النصي	type <i>fileName.txt</i>	cat <i>fileName.txt</i>
إزالة أي محتوى على نافذة الأوامر	cls	clear

الدخول إلى مجلد المستخدم الحالي	<code>cd %USERPROFILE%</code>	<code>cd ~</code>
فتح مجلد أو ملف	<code>start fileName</code>	Open <i>folderName/directoryName</i>
حذف ملف	<code>del fileName</code>	<code>rm fileName</code>
حذف مجلد	<code>rmdir folder or rd folder</code>	<code>rm -r directoryName</code>

مفهوم Version Control Systems

أنظمة التحكم بالنسخ

أنظمة التحكم بالنسخ أو ماتسمى **Version Control System** هي أنظمة تقوم بإدارة وتتبع مراحل تطور المشروع، بحيث يتم تسجيل أي تعديل سواء كان إضافة ملف جديد أو حذف أو تحديث ملف موجود مسبقاً في تاريخ المشروع منذ البداية. بعض أنواع أنظمة التحكم بالنسخ يساعدنا على تطوير المشاريع بشكل أسرع من خلال توفير الخدمات التي يحتاجها الفريق للعمل معاً.



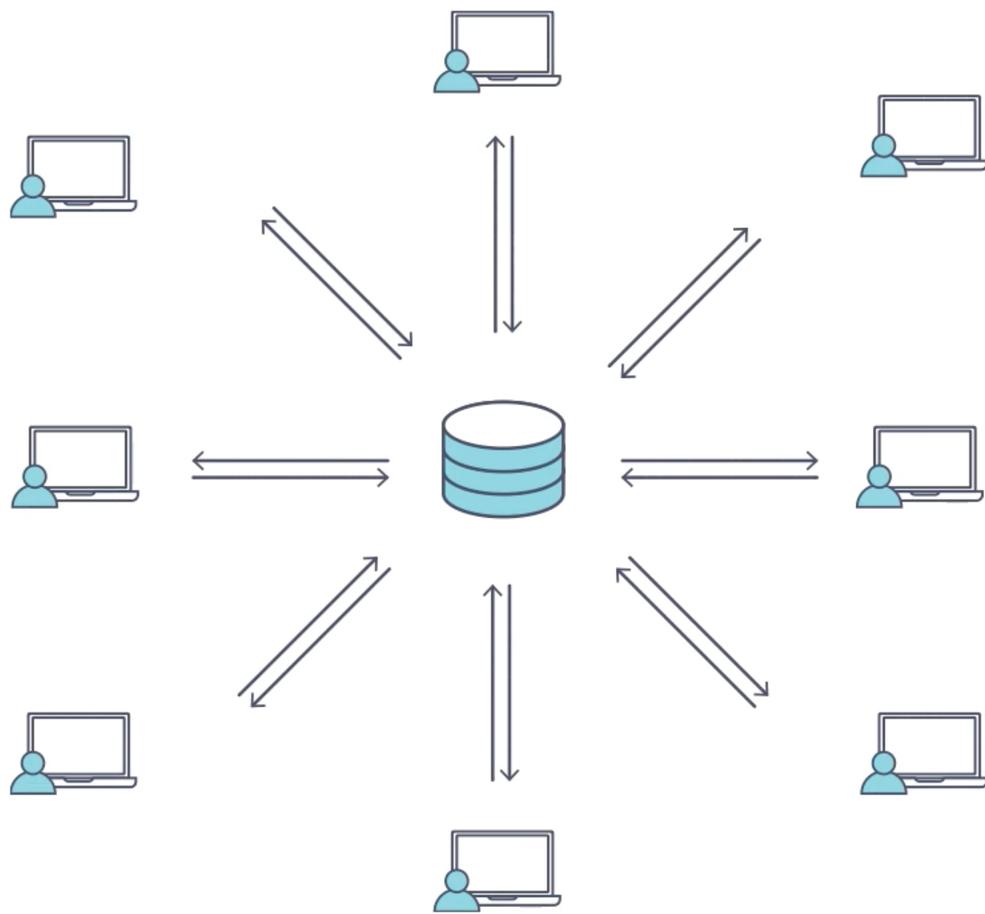
أنظمة التحكم بالنسخ لها أنواع:

- **Centralized**

أنظمة التحكم بالنسخ المركزية وتعمل بالطريقة التالية:

توجد نسخة واحدة من المشروع مشتركة بين جميع أعضاء الفريق، لكن من سلبياتها أنها تحتوي على نقطة واحدة عند حصول الخطأ، أي أن نتيجة انقطاع الاتصال من الخادم الذي تم رفع المشروع عليه تعني انقطاع الاتصال لدى جميع أعضاء الفريق، وبالتالي لا يمكن لأي عضو الوصول للمشروع وتطويره خلال فترة الانقطاع.

Centralized

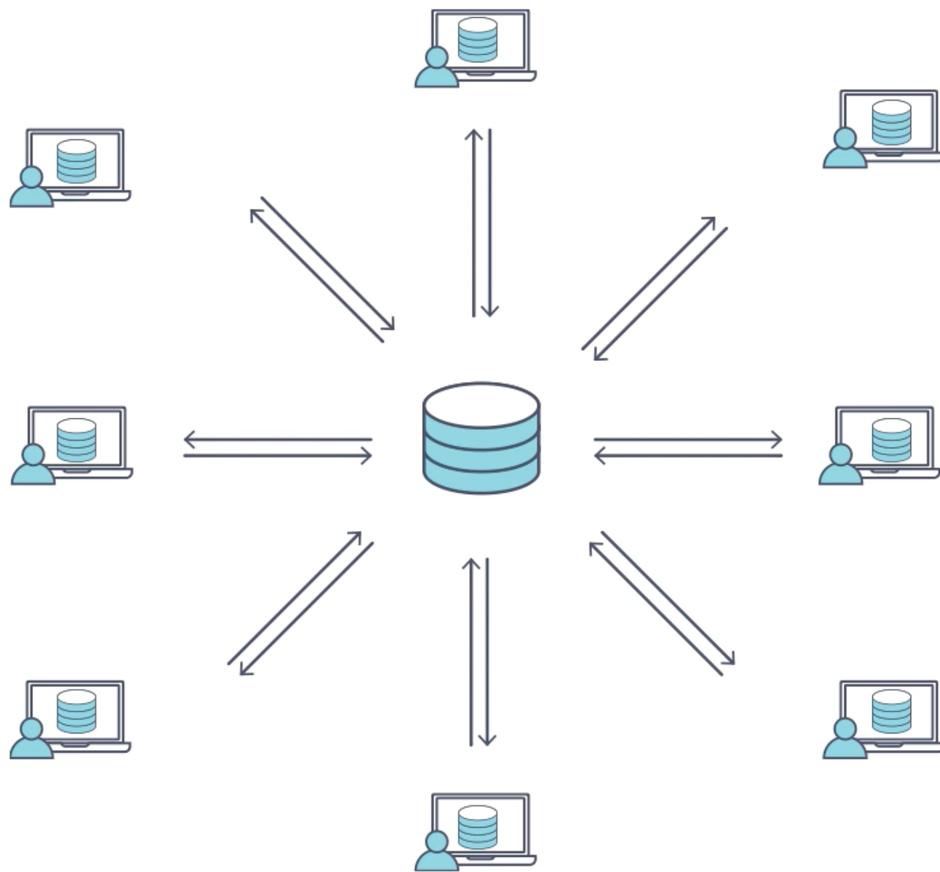


• الموزّعة :Distributed

أنظمة التحكم بالنسخ الموزّعة وتعمل بالطريقة التالية:

توجد نسخة مشتركة بين جميع أعضاء الفريق مع وجود نسخة خاصة لكل عضو، بحيث يقوم كل عضو بإضافة تعديلاته على النسخة المحلية ومن ثم رفعها إلى النسخة المشتركة بحيث تصل التحديثات إلى جميع أعضاء الفريق.

Distributed



التعرف على نظام Git

ما هو Git؟

هو عبارة عن نظام ينتمي إلى أنظمة التحكم بالنسخ الموزعة، أي أنه يقوم بتتبع وتسجيل التغييرات التي أجريت على الملفات، وطريقته في تتبع الملفات تكون عبر أخذ نسخ "snapshot" (تسمى أيضًا commits) من المشروع يحددها المستخدم فيقوم هذا النظام بحفظ التغييرات خطوة بخطوة مع التسلسل الزمني ويرفق وصفاً لكل خطوة من هذه التغييرات حتى يتمكن المستخدم من تتبعها والرجوع إليها بسهولة، وجميع التغييرات التي تحصل على المشروع يتم تخزينها في Repository أي مخزن أو مستودع.

تثبيت Windows - أجهزة Git

قبل أن نستطيع التعامل مع Git يجب علينا أولاً أن نقوم بتنصيبه على أجهزتنا، لذلك قم بالدخول على موقع Git الرسمي من خلال هذا الرابط (<https://git-scm.com>) والضغط على Download كما هو موضح في الصورة المرفقة:

The screenshot shows the official Git website at <https://git-scm.com>. At the top, there's a search bar and a navigation menu. Below the header, there's a brief introduction to Git as a free and open source distributed version control system. To the right is a diagram illustrating the distributed nature of Git with multiple repositories connected by various colored lines (red, green, blue, yellow) representing branches and pull requests. The main content area features several sections: 'About' (with a gear icon), 'Documentation' (with a book icon), 'Downloads' (with a download arrow icon), and 'Community' (with a speech bubble icon). A prominent 'Download' section on the right shows the latest release '2.32.0' (Release Notes from 2021-06-06) with a 'Download for Mac' button. Below this are links for 'Mac GUIs', 'Tarballs', 'Windows Build', and 'Source Code'. At the bottom, there's a section titled 'Companies & Projects Using Git' featuring logos for Google, Facebook, Microsoft, Twitter, LinkedIn, Netflix, PostgreSQL, MySQL, Android, Linux, Rails, Qt, GNOME, Eclipse, Kubuntu, and Xfce.

تثبيت MacOS - أجهزة Git

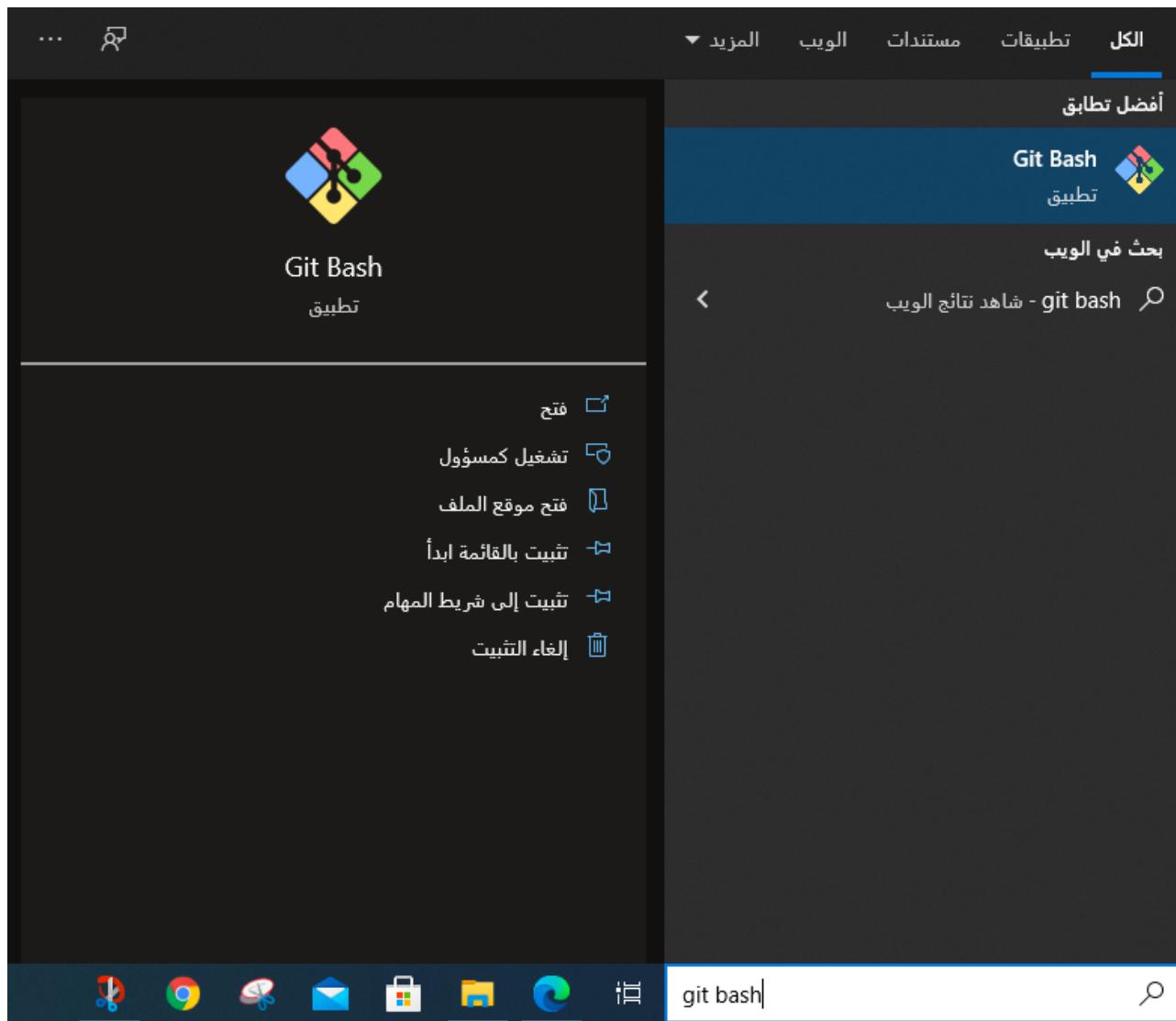
يعتبر Git موجود مسبقاً على أجهزة MacOS وبالتالي كل ما عليك عمله هو التحقق من وجوده من خلال استخدام الأمر git --version في Terminal كالتالي:

```
dev: git --version  
git version 2.30.1 (Apple Git-130)
```

في حالة ظهور النسخة، هذا يعني أن Git موجود لديك، وإن لم يكن موجود سيقوم Terminal بارشادك إلى ما يجب عمله لتنزيله.

فتح Windows على نظام Git bash

عن طريق البحث عن git بعد إتمام عملية تتنزيل git في جهازك (نظام التشغيل Windows) كما يلي:



مفاهيم و أوامر Git

يوجد في Git العديد من المفاهيم والأوامر سنقوم في هذا الدرس بتغطية أكثر هذه المفاهيم شيوعاً.

الأمر git init

في Git جميع التعديلات يتم تخزينها في Repository لذلك إن أردنا تتبع مشروع ما باستخدام Git يجب علينا أو لا إنشاء هذا المخزن لتسجيل جميع التعديلاتداخله، وبمعنى آخر أي فريق يريد استخدام Git لتتبع المشروع الذي يتم العمل على تطويره يحتاج المخزن لتسجيل جميع التعديلات التي حدثت خلال رحلة تطوير المشروع.

خطوات إنشاء Repository

- فتح command prompt أو terminal.
- الوصول إلى المسار الذي يحتوي على المشروع الذي تريد تتبعه وإدارته.
- الدخول على المشروع.
- استخدام الأمر git init الذي يقوم بإنشاء Repository جديد.

الأمر المستخدم لإنشاء Repository هو:

```
mkdir git-test  
git init
```

هذا الأمر سيقوم بإنشاء مجلد git. والذي يحتوي على جميع الملفات والمجلدات التي يحتاجها Git لتتبع المشروع.
استعراض الملفات عن طريق

```
ls  
ls -a
```

طريقة مختصرة لإنشاء Repository

```
git init git-test
```

مفهوم Git Stages

خلال العمل على المشروع والتطوير عليه، لابد من إنشاء ملفات، أو حذفها، أو التعديل عليها ولحفظ هذه التعديلات لابد من إعلام Git بأنه تم التعديل عليها وأنك تريد حفظها لأن Git لن يقوم بشكل تلقائي بحفظ هذه التعديلات على تاريخ المشروع، وقد تحتاج للرجوع عن التعديل لذلك وجد مفهوم Git Stages والذي ينص على أن الملفات في Git تمر بمراحل أو Stages كالتالي:

المرحلة الأولى Untracked

- تعني أن Git لا يقوم بتتبع الملفات إلى الآن ولم يخزنها في Repository.

المرحلة الثانية tracked وتنسمى أيضا Staged

- تعني أن Git يعلم بالتغييرات الحاصلة على الملف ويقوم بتنبيتها لكن لم يتم حفظها إلى الآن في Repository.

المرحلة الثالثة Committed

- وتعني أن الملفات تم التعديل عليها وحفظها في تاريخ المشروع Repository.

يمكّن التفكير بالمراحل كمتجر الكعك، فعندما تريد أن تطلب كعكة يجب عليك أولاً اختيارها وتحديد ماتريد فتشابه هنا مرحلة untracked، بمعنى أنك لم تقم إلى الآن باختيار طلبك بالفعل، وبعد اختيارك يقوم الموظف بتأليفها مثل مرحلة staged لكن لن يقوم بتسليمها لك إلا بعد تسجيل الفاتورة والدفع، وهذه تشابه مرحلة committed بحيث أنه تم تسجيل شرائك في قاعدة البيانات لديهم ويستطيعون العودة لتاريخ الشراء ومالمذي قمت بشرائه في أي وقت.

أمر git status

يساعدنا الأمر git status على معرفة حالة المشروع، فيزودنا بعدد من المعلومات مثل أسماء الملفات المتواجدة في مرحلة staged ومرحلة untracked.

مثال:

قبل تطبيق أي أمر من أوامر Git على أحد المشاريع يجب أن يكون لدينا مجلد git. والذي يتم إنشاؤه من خلال الأمر init، دعونا أو لا ننشئ مجلد جديد في Desktop كالتالي:

```
> pwd
/Users/user/Desktop
```

بعد التأكد من تواجدنا في Desktop سنقوم بإنشاء المجلد الجديد ولتكن بالاسم git-test:
> git-test

لنقم الآن بالدخول إليه واستخدام الأمر ls و -a لعرض الملفات والمجلدات سواء كانت مخفية أم لا:
> cd git-test
> ls
> ls -a

نلاحظ أن المجلد فارغ تماماً، بمعنى أننا لم نقم بإنشاء Repository باستخدام الأمر git init بعد، لنقم باستخدام أمر git status للتحقق من أنه سيعمل دون تواجد Repository أم لا:

```
> git status
fatal: not a git repository (or any of the parent directories): .git
```

نتج عن الأمر رسالة خطأ كرد، والسبب هو عدم استخدامنا للأمر `git init` قبل استخدامنا للأمر `git status` لذلك سنقوم باستخدامه الآن:

```
> git init  
Initialized empty Git repository in /Users/user/Desktop/git-test/.git/
```

بعد إنشاء Repository نستطيع استخدام أوامر Git، لنقم باستخدام أمر `git status` مرة أخرى:

```
> git status  
On branch master  
  
No commits yet  
  
nothing to commit (create/copy files and use "git add" to track)
```

بهذا الشكل يقوم الأمر `git status` بعرض حالة المشروع لنا.

الأوامر المستخدمة:

```
|dev: pwd  
/Users/user/Desktop  
|dev: mkdir git-test  
|dev: cd git-test  
|dev: ls  
|dev: ls -a  
..  
|dev: git status  
fatal: not a git repository (or any of the parent directories): .git  
|dev: git init
```

Initialized empty Git repository in /Users/user/Desktop/git-test/.git/

```
dev: git status  
On branch master  
  
No commits yet  
  
nothing to commit (create/copy files and use "git add" to track)  
dev: █
```

أمر git add

كما ذكرنا سابقاً، يوجد مراحل تمر فيها الملفات في Git، أحد هذه المراحل هي Untracked وتعني أن الملف لا يتم تتبعه، مثل بمحرّد إنشائه لملف جديد يعتبر الملف Untracked، لذلك إذا أردت أن يقوم Git بتتبعه يجب عليك نقله إلى المرحلة التالية وهي Staged، ويتم ذلك من خلال استخدام الأمر `git add` مع إرسال اسم الملف الذي تريد نقله، وبالتالي نستخلص مما ذكر بأن الأمر `git add` يقوم بنقل الملفات من مرحلة Untracked إلى Staged.

مثال:

ملاحظة: الأوامر في هذا المثال تابعة للأوامر المتواجدة أعلاه.

لرؤية المراحل التي يمر بها الملف لنقم بإنشاء ملف جديد باستخدام الأمر `touch` كالتالي:

```
> pwd  
/Users/user/Desktop/git-test  
> touch file.txt  
> ls  
file.txt
```

الآن بعدما تأكدنا من أنه تم إنشاء الملف بنجاح، لنقم بالاستعلام عن حالة المشروع من خلال استخدام الأمر `git status`:

```
> git status  
On branch master
```

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

```
file.txt
```

```
nothing added to commit but untracked files present (use "git  
add" to track)
```

نلاحظ أنه قام بوضع الملف الجديد في مرحلة **Untracked**، وكما ذكرنا لتبقيه نستخدم الأمر `git add` كالتالي:

```
> git add file.txt
```

```
> git status
```

```
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   file.txt
```

بهذا الشكل قمنا بنقل الملف إلى المرحلة الأخرى وهي **.Staged/Tracked**

الأوامر المستخدمة:

```

dev: pwd
|/Users/user/Desktop/git-test
dev: touch file.txt
|dev: ls
|file.txt
dev: git status
|On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    file.txt

nothing added to commit but untracked files present (use "git add" to track)
dev: git add file.txt
|dev: git status
|On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   file.txt

```

نلاحظ الرسالة أعلاه، بأنه في حال أردت التراجع عن تتبع الملف يمكنك استخدام الأمر `git rm`، سنقوم باستخدامه ونرى ما سيحصل:

```

> git rm --cached file.txt
rm 'file.txt'
> git status
On branch master

No commits yet

```

```

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    file.txt

```

```
nothing added to commit but untracked files present (use "git add" to track)
```

بهذا الشكل قمنا بإعادة الملف إلى مرحلة **Untracked**.

الأوامر المستخدمة:

```
|dev: git status  
|On branch master  
  
No commits yet  
  
Changes to be committed:  
  (use "git rm --cached <file>..." to unstage)  
    new file:   file.txt  
  
|dev: git rm --cached file.txt  
rm 'file.txt'  
|dev: git status  
On branch master  
  
No commits yet  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    file.txt  
  
nothing added to commit but untracked files present (use "git add" to track)
```

تهيئة Git

```
git config --global --get user.email  
git config --global --get user.name
```

عند عدم ظهر أي اسم، هذا يدل على أننا لم نعين البريد و الإسم، ويمكننا القيام بذلك عن طريق التالي:

```
git config --global user.email dev@example.com  
git config --global user.name dev
```

عند وجود مسافات في الإسم نقوم بكتابته كالتالي:

```
git config --global user.name "dev desck"
```

الأمر git commit

يقوم هذا الأمر بنقل الملفات من مرحلة **Staged** إلى **Committed**.

مثال:

سنقوم أولاً بنقل الملف من مرحلة untracked إلى staged مرة أخرى كالتالي:

```
> git add file.txt  
> git status  
On branch master  
  
No commits yet  
  
Changes to be committed:  
(use "git rm --cached <file>..." to unstage)  
new file:   file.txt
```

والآن سنقوم باستخدام الأمر `git commit` لنقل الملفات إلى مرحلة Committed، هذه العملية تسمى commit لكن يجب الأخذ بعين الاعتبار أننا نحتاج إلى رسالة commit، بحيث نصف فيها ما قمنا بالتعديل عليه كالتالي:

```
> git commit -m "new file added"  
[master (root-commit) e4f436b] new file added  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 file.txt
```

الآن تم تخزين التعديلات التي قمنا بها (إنشاء ملف جديد) في Repository، وللحتحقق من أنه لا يوجد تعديل آخر جديد سنقوم باستخدام الأمر `git status`.

```
> git status  
On branch master  
nothing to commit, working tree clean
```

الأوامر المستخدمة:

```
[dev: git commit -m "new file added"
[master (root-commit) e4f436b] new file added
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 file.txt
[dev: git status
On branch master
nothing to commit, working tree clean
```

الأمر git log

يقوم هذا الأمر بعرض تاريخ التعديلات commit التي قمنا بها سابقاً.

مثال:

```
> git log
commit e4f436b662be5e7183351e1636a4d15faf6c274f (HEAD -> master)
Author: dev <dev@test.com>
Date:   Wed Jul 28 17:10:58 2021 +0300

    new file added
```

نلاحظ وجود عدد من المعلومات مع commit الذي قمنا به سابقاً، منها التاريخ والوقت، الرسالة الخاصة في commit، معلومات الشخص الذي قام بحفظ التعديلات مثل اسمه dev والإيميل الخاص به، كما أنه أيضاً قام بإنشاء رقم hash لتحديد commit بحيث يدل كل commit على hash معين ولا مجال لداخلهم.

ملاحظة: يستفيد Git من رقم hash لتحديد ما إذا تم التعديل على الملف أم لا، فإذا تغير على رقم hash هو دليل على تغيير محتوى الملف.

الأوامر المستخدمة:

```
[dev: git log  
commit e4f436b662be5e7183351e1636a4d15faf6c274f (HEAD -> master)  
Author: dev <dev@test.com>  
Date:   Wed Jul 28 17:10:58 2021 +0300  
  
new file added
```

الأمر `git log --oneline`

يقوم هذا الأمر بعرض تاريخ التعديلات commit التي قمنا بها سابقاً بشكل مختصر (في سطر واحد).

مثال:

```
> git log --oneline  
e4f436b (HEAD -> master) new file added
```

الأوامر المستخدمة:

```
[dev: git log --oneline  
e4f436b (HEAD -> master) new file added
```

لتغيير محتوى الرسالة في الأمر `git commit` ، يمكننا كتابة:

```
> git commit --amend -m "last update"
```

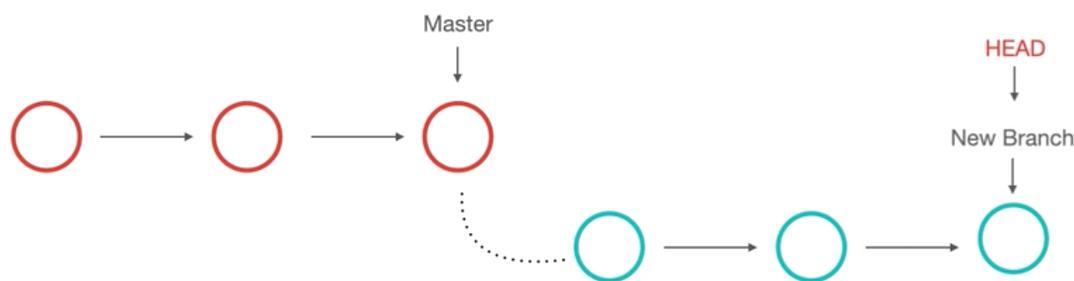
إخفاء الملفات باستخدام `gitignore`

لفترض أننا لا نريد تتبع ملف `x.txt`

```
touch x.txt  
git status  
touch .gitignore  
ls  
ls -a  
echo x.txt > .gitignore
```

branches مفهوم

تعتبر الفروع أو **Branches** مسارات لتطوير المشروع، لنفترض أننا نريد التعديل على المشروع من خلال إضافة خاصية جديدة لكن لسنا متأكدين من فعاليتها أو إمكانية تطبيقها، وبالتالي لا نريد أن نخاطر بالتعديل على المشروع الفعلي، من هنا أنت تخصية الفروع، بحيث نستطيعأخذ نسخة من المشروع للتجربة وإضافة الخصائص الجديدة من خلال إنشاء فرع جديد (مسار جديد للمشروع).



الأمر git branch

يستخدم هذا الأمر لعرض الفروع المتواجدة في المشروع.

مثال:

```
> git branch
* master
```

من المخرجات نلاحظ وجود فرع واحد يسمى **master** وهو الفرع الثقائي/ الرئيسي الخاص بالمشروع، كما نلاحظ وجود **.master** وتعني أننا حالياً متواجدين في **master** وأي تعديل أو إضافة حاصلة الآن ستكون في الفرع **.master**.

الأمر git branch <branchName>

هذا الأمر يقوم بإنشاء فرع جديد.

مثال:

```
> git branch test
> git branch
* master
test
```

نلاحظ أعلاه بأنه فعلاً تم إنشاء فرع جديد بالاسم `.test`.

الأمر `git checkout`

يقوم هذا الأمر بالانتقال إلى فرع آخر.

مثال:

```
> git branch
* master
  test
```

نلاحظ هنا أننا مازلنا على نفس الفرع وأي تعديل سيكون في `master`، وبالتالي إن أردنا التعديل على الفرع الجديد يجب الانتقال له باستخدام الأمر `git checkout` كالتالي:

```
> git checkout test
Switched to branch 'test'
```

بهذا الشكل أي تعديل يحصل الآن سيكون في فرع `test`، ولتأكد أكثر سنستخدم الأمر `git branch` مرة أخرى.

```
> git branch
  master
* test
```

الأوامر المستخدمة:

```
[dev: git branch
* master
[dev: git branch test
[dev: git branch
* master
    test
[dev: git checkout test
Switched to branch 'test'
[dev: git branch
    master
* test
```

طريقة مختصرة لإنشاء فرع جديد والانتقال له مباشرة.

```
git checkout -b test2
```

لإعادة تسمية أحد فروع المشروع.

```
git branch -m test2 test3
git branch
```

حذف أو إزالة أحد فروع المشروع.

ملاحظة: لا يمكن حذف الفرع إلا بعد الانتقال منه.

```
git branch -d test3
git checkout test
git branch -d test3
```

Merge مفهوم

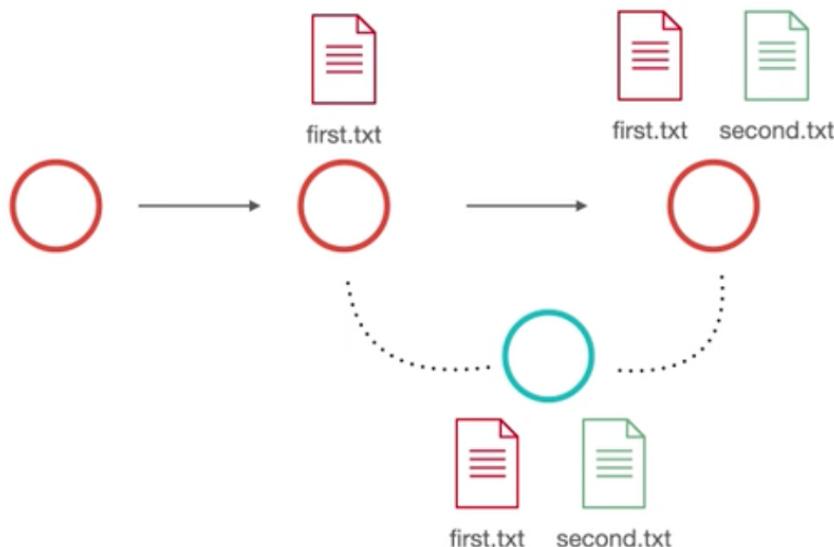
طرقنا سابقاً إلى مفهوم الفروع، وذكرنا أنه يمكننا أخذ فرع من المسار الخاص بالمشروع والتعديل عليه كما نريد، لكن ماذا سنفعل عند نجاح الخاصية الجديدة؟ عند نجاح الخاصية سنقوم بدمجها مع المسار/الفرع الرئيسي للمشروع.

الأمر `git merge`

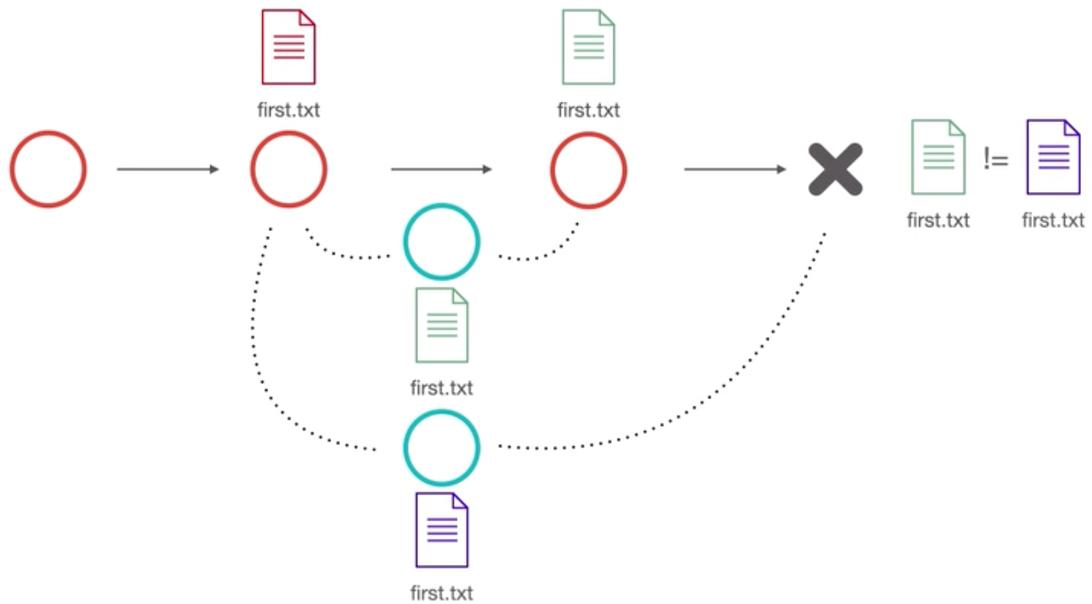
يستخدم الأمر `git merge` لدمج الفروع مع بعضها.

عند دمج مسار بأخر سينقسم الدمج إلى حالتين:

- دمج `fast forward`: وهو الدمج الحالى عندما تتم إضافة الخصائص في الفرع الجديد مع الفرع الرئيسي فقط.



- دمج مع `true merge` أو ما يسمى `conflict`، وهو ما يحدث عند ظهور تعارض `conflict` بسبب اختلاف نسخ المشروع، فعندهما يقوم شخص بالتطوير على المسار الرئيسي وأنت لازلت تعمل على إضافة الخاصية في الفرع الخاص بك ومن ثم حاولت الدمج، سيظهر اختلاف بين النسختين وسيطألك `Git` بحل هذا الاختلاف/التعارض.



مثال على الدمج :fast forward

لقم بالتعديل على الفرع test من خلال إنشاء ملف جديد فيه كالتالي:

```
> git branch
  master
* test
> touch new-file.txt
```

الآن بعد إنشاء الملف يجب أن يكون هذا الملف في مرحلة untracked ولنتحقق من ذلك سنقوم باستخدام الأمر

كالتالي: status

```
> git status
On branch test
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  new-file.txt
```

```
nothing added to commit but untracked files present (use "git  
add" to track)
```

لنقم الآن بنقل الملف إلى مرحلة `staged` من ثم `committed` بحيث يزيد بعد `commit` واحد عن الفرع `:(master)`

```
> git add new-file.txt  
> git commit -m "new-file.txt added to test branch"  
[test 52fa698] new-file.txt added to test branch  
 1 file changed, 0 insertions(+), 0 deletions(-)  
 create mode 100644 new-file.txt  
> ls  
file.txt new-file.txt
```

نلاحظ الآن أنه أصبح لدينا ملفان في فرع `test` وبعد أن قمنا بحفظ التعديلات سننتقل لفرع `master` ونرى عدد الملفات هناك كالتالي:

```
> git branch  
  master  
* test  
> git checkout master  
Switched to branch 'master'  
> ls  
file.txt
```

نلاحظ أنه في الفرع `master` يوجد ملف واحد فقط، والسبب أن الملف الثاني تم إنشاؤه في الفرع `test`، وبالتالي سنقوم بدمج الفرع مع `master` لجلب التحديثات الخاصة بالفرع `test` في المسار الرئيسي باستخدام `: git merge test`

```
> git merge test  
Updating e4f436b..52fa698  
Fast-forward  
  new-file.txt | 0  
  1 file changed, 0 insertions(+), 0 deletions(-)  
 create mode 100644 new-file.txt
```

الآن لتحقق من عدد الملفات باستخدام الأمر ls:

```
> ls  
file.txt new-file.txt
```

بالنالي نلاحظ أنه بالفعل تم دمج الفرع test مع master.

الأوامر المستخدمة:

```
|dev: git status  
On branch test  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    new-file.txt  
  
nothing added to commit but untracked files present (use "git add" to track)  
|dev: git add new-file.txt  
|dev: git commit -m "new-file.txt added to test branch"  
[test 52fa698] new-file.txt added to test branch  
 1 file changed, 0 insertions(+), 0 deletions(-)  
  create mode 100644 new-file.txt  
|dev: ls  
file.txt      new-file.txt  
|dev: git branch  
  master  
* test  
|dev: git checkout master  
Switched to branch 'master'  
|dev: ls  
file.txt  
|dev: git merge test  
Updating e4f436b..52fa698  
Fast-forward  
  new-file.txt | 0  
  1 file changed, 0 insertions(+), 0 deletions(-)  
  create mode 100644 new-file.txt  
|dev: ls  
file.txt      new-file.txt
```

مثال على الدمج :true merge

```
mkdir y  
cd y  
git init  
echo i am in the master >> 1.txt  
git add .
```

```
git commit -m "master changes"
git branch
git branch first
git branch second
git checkout first
echo i am in the first branch >> 1.txt
git add .
git commit -m "first merge"
git checkout main
git merge first
git checkout second
echo i am in the second branch >> 1.txt
git add .
git commit -m "second merge"
git checkout main
git merge second
Auto-merging 1.txt
CONFLICT (content): Merge conflict in 1.txt
Automatic merge failed; fix conflicts and then commit the result.
open 1.txt
git add .
git commit -m "final version"
```

تعرفنا سابقاً على برنامج git وعلى أهميته في حفظ نسخ المشاريع وتنبيعها والوصول إليها بكل سهولة، لكن ماذا لو أننا فقدنا جهازنا الشخصي أو احتجنا للوصول إلى Repository المشروع في وقت لم يكن جهازنا برفقتنا، كيف سنقوم بذلك؟ هنا يأتي دور منصة GitHub وغيرها من المنصات المشابهة والتي تقوم بحفظ نسخة من Repository المشروع على الإنترن特 حتى تستطيع الوصول لها من أي جهاز أو مكان متصل بالإنترنت.

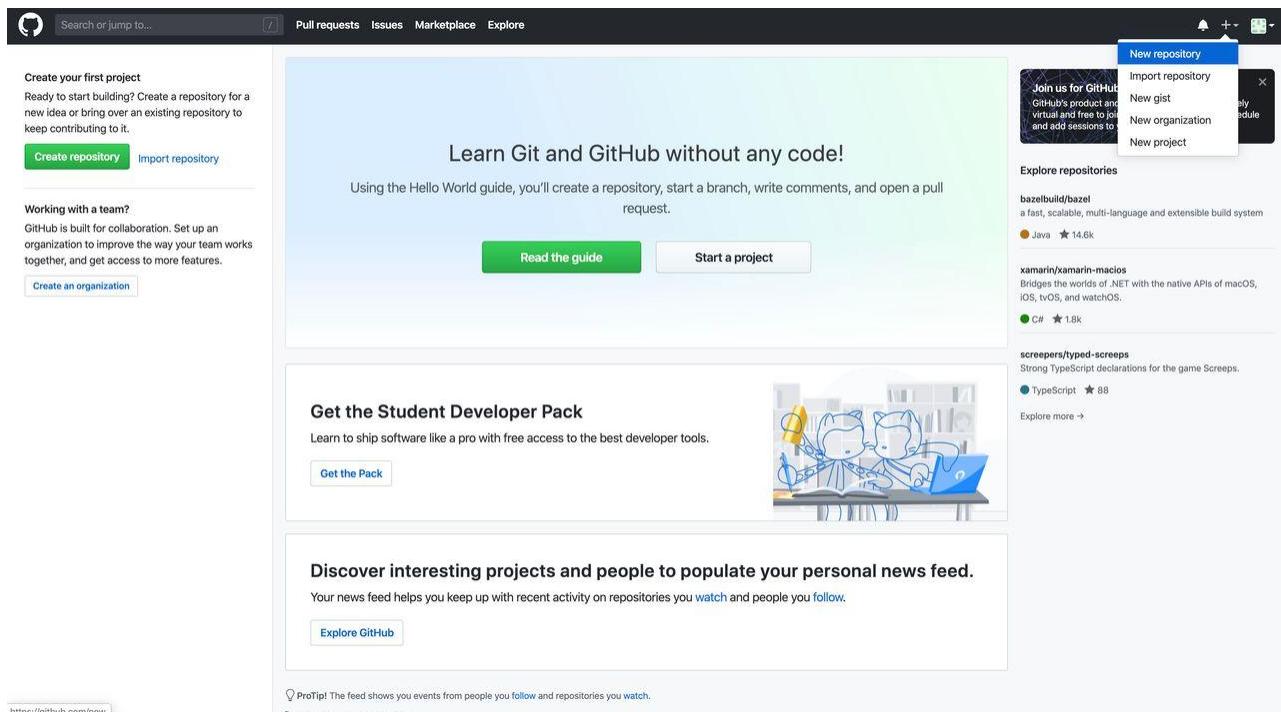
تسجيل حساب في GitHub

لإنشاء حساب على منصة GitHub من الرابط التالي: <https://github.com/join>

ملاحظة: GitHub هي المنصة الأشهر لحفظ ومشاركة المشاريع أما بالنسبة git فهو النظام الذي يقوم بالتحكم بالنسخ.

إنشاء GitHub على Repository

بعد أن أتممنا عملية التسجيل في موقع GitHub وقمنا بتسجيل الدخول، سنقوم بإنشاء مستودع repository جديد لإنشاء مشروع جديد نقوم بالنقر على علامة الزائد الموجودة على في الجزء العلوي من الصفحة، ثم نقوم باختيار New repository.

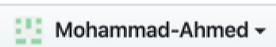


ثم سننتقل للصفحة التالية وهي الصفحة الخاصة بإنشاء المستودع :repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner



Repository name *

Great repository names are short and descriptive. [my-app is available.](#) Inspiration? How about [friendly-computing-machine?](#)

Description (optional)

Public

Anyone can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Initialize this repository with a README

This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

والآن يمكننا إدخال اسم مشروع من اختيارنا في الخانة **Repository name**، فمثلاً اختيار الاسم **my-app** لهذا المشروع وقمنا باختيار **Public** لجعل مشروعنا عام، وأخيراً سنقوم بالنقر على زر **Create repository** لإتمام عملية الإنشاء، ستظهر لنا ثلاثة خيارات أو مسارات يمكننا اتباعها لربط المستودع **remote repository** بالملف **:local file**.

Mohammad-Ahmed / my-app

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security 0 Insights Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or [HTTPS](#) [SSH](#) <https://github.com/Mohammad-Ahmed/my-app.git>

Get started by creating a new file or uploading an existing file. We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# my-app" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git remote add origin https://github.com/Mohammad-Ahmed/my-app.git  
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/Mohammad-Ahmed/my-app.git  
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

الأمر `git clone`

يُمكننا هذا الأمر من نسخ مشروع موجود مثلاً على Github Repository

مثال:

nodejs / node

About

Node.js JavaScript runtime 🚀

Clone

Releases 679

2021-07-14, Version 16.5.... (Latest)

+ 678 releases

لدينا أعلاه مشروع متواجد على Github، لنفرض أننا نريد استنساخ المشروع والعمل عليه، لذلك سنستخدم الأمر git من خلال نسخ الرابط كالتالي:

nodejs / node

About

Node.js JavaScript runtime 🚀

Clone

Releases 679

2021-07-14, Version 16.5.... (Latest)

+ 678 releases

بعد نسخ الرابط سنقوم بالذهاب إلى terminal والذهاب إلى المسار الذي نريد وضع المشروع به، من ثم استخدام الأمر git والذهب إلى المسار الذي نريد وضع المشروع به، من ثم استخدام الأمر clone كالتالي:

```
git clone https://github.com/nodejs/node.git
```

بهذا الشكل أصبح لدينا مشروع.

الأمر git push

يقوم هذا الأمر بإرسال التغييرات التي أجريت ورفعها إلى Repository محدد.

```
git push <remote name> <branch name>
```

حيث تمثل <branch name> الاسم أو المؤشر المعطى للمستودع الموجود على الإنترن特 ويمثل اسم الفرع الخاص به.

الأمر git pull

يجلب هذا الأمر ويدمج التغييرات من Repository المطلوب.

```
git pull <variable name> <branch name>
```

حيث تمثل <branch name> الاسم أو المؤشر المعطى للمستودع الموجود على الإنترنط ويمثل اسم الفرع الخاص به.

تنبيه: عملية pull تقوم بدمج ملفات المستودع الموجودة على الإنترنط بالمستودع المحلي فيجب أن نقوم بعملية commit للتغييرات التي قمنا بها أولاً، ثم بعد ذلك نقوم بعملية السحب pull حتى لا تتأثر التغييرات التي قمنا بها على النسخة المحلية.

الأمر git remote

يسمح لك هذا الأمر باستعراض remote المتواجد في المشروع الخاص بك، ويستخدم لتحديد موقع Repository الخاص بك في Github أو أي منصة أخرى.

مثال:

```
git remote
```

ملاحظة: من الممكن أن يشير `remote` إلى موقع مشروع آخر في الجهاز ولا يتشرط أن يؤشر على موقع `Repository` في أحد المنصات.

الأمر `git remote add`

يسمح لك هذا الأمر بإنشاء `remote` بحيث يقوم برفع المشروع المحلي الخاص بك إلى المشروع الموجود على `internet`.

مثال:

```
: git remote add Github باستخدام الأمر repository لإنشاء Githubي ومن ثم رفعه على
```

```
git remote add origin https://github.com/Mohammad-Ahmed/my-ap  
p.git
```

مصادر إضافية:

<https://satr.codes/courses/ZIKLfufzmW/view>



• منصة سطر التعليمية satr.codes