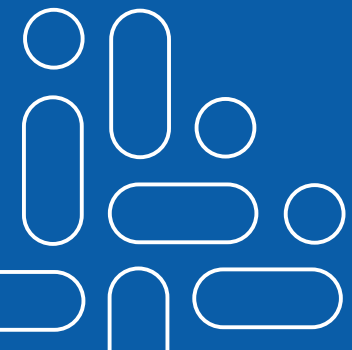
مفهوم Version Control Systems

سنتعرف في هذه الوحدة على مفهوم التحكم بالاصدارات وطريقة استخدامه لادارة ومتابعة المشاريع.



مفهوم Version Control Systems

أنظمة التحكم بالنسخ

أنظمة التحكم بالنسخ أو ماتسمى `Version Control System` هي أنظمة تقوم بإدارة وتتبّع مراحل تطوّر المشروع، بحيث أي تعديل سواء كان إضافة ملف جديد أو حذف أو تحديث ملف موجود مسبقًا يتم تسجيله في تاريخ المشروع منذ البداية.

أيضًا البعض من أنواع أنظمة التحكم بالنسخ يُساعدنا على تطوير المشاريع بشكل أسرع من خلال توفير الخدمات التي يحتاجها الفريق للعمل معاً.

أنواع أنظمة التحكم بالنسخ

- المركزيّة `Centralized`:

أنظمة التحكم بالنسخ المركزيّة 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، بالتالي لايُمكن لأي عضو الوصول للمشروع وتطويره خلال فترة الانقطاع.**

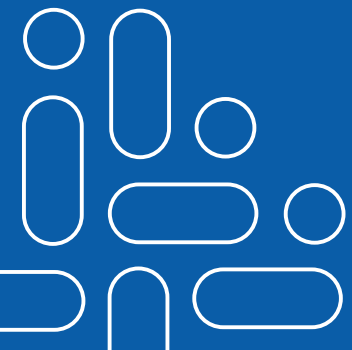
- الموزّعة `Distributed`:

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 المُشتركة بحيث تصل التحديثات إلى جميع أعضاء الفريق.

نظام Git

سنتعرف في هذه الوحدة على نظام Git وعلى طريقة استخدامه للتحكم بالاصدارات وطريقة استخدامه لادارة ومتابعة المشاريع.



التعرف على نظام Git

ما هو Git

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

صورة 01 هنا

لكن خذ بعين الاعتبار أن Git موجود مسبقاً في أجهزة macOS بالتالي 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 Git130-)
```

في حالة ظهور النسخة، هذا يعني أن Git موجود لديك. وان لم يكن موجود سيقوم terminal بارشادك الى ما يجب

عمله لتنزيله.

مثال للعرض

مفاهيم وأوامر Git

يوجد العديد من المفاهيم والأوامر في Git، لذلك سنقوم بمحاولة تغطية اغلب المفاهيم والأوامر شيوعاً في Git خلال هذا الدرس.

الأمر git init

في Git جميع التعديلات يتم تخزينها في Repository لذلك إن أردنا تتبع مشروع ما باستخدام Git يجب علينا أولًا إنشاء هذا المخزن لتسجيل جميع التعديلات بداخله. بمعنى 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 terminal أو command prompt.
- الوصول إلى المسار الذي يحوي المشروع الذي تريد تتبعه وادارته.
- الدخول على المشروع.
- استخدام الأمر git init الذي يقوم بإنشاء Repository جديد.

الأمر المستخدم لإنشاء Repository هو :

```
01        git init
```

مثال للعرض

هذا الأمر سيقوم بإنشاء مجلد `.git` الذي يحتوي على جميع الملفات والمجلدات التي يحتاجها `Git` لتتبع المشروع.

مفهوم Git Stages

خلال العمل على المشروع والتطوير عليه، لابد من انشاء ملفات، حذفها، او التعديل عليها. ولحفظ هذه التعديلات لابد من إعلام `Git` بأنه تمّ التعديل عليها وانك تريد حفظها لأن `Git` لن يقوم بشكل تلقائي بحفظ هذه التعديلات على تاريخ المشروع، فربما تريد التراجع عن التعديل؟ لذلك وجد لدينا مفهوم `Git Stages` و ينص على أن الملفات في `Git` تمر

بمراحل أو Stages كالتالي.

المرحلة الأولى Untracked.

تعني ان Git لا يقوم بتتبع الملفات الى الآن ولم يخزنها في Repository.

المرحلة الثانية Staged و تسمى ايضاً tracked.

تعني ان Git يعلم بالتغييرات الحاصلة على الملف ويقوم بتتبعها لكن لم يقم بحفظها الى الآن في Repository.

المرحلة الثالثة Committed

وتعني أن الملفات تمّ التعديل عليها وحفظها في تاريخ المشروع Repository.

يمكنك التفكير بالمراحل كمتجر الكعك، فعندما تريد ان تطلب كعكة يجب عليك أولا اختيارها وتحديد ماتريد فتشابه هنا مرحلة untracked بمعنى انك لم تقم الى الآن باختيار طلبك بالفعل. بعد اختيارك يقوم الموظّف بتغليفها لك مثل مرحلة staged لكن لن يقوم باعطائك اياها بعد. من ثمّ يقوم بتسجيل الفاتورة وعند دفعك يسلّمها لك وتشابه مرحلة committed بحيث أنه تم تسجيل شرائك في قاعدة البيانات لديهم ويستطيعون العودة لتاريخ الشراء وماذا قمت بشرائه

في أي وقت.

أمر git status

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

مثال

قبل تطبيق أي أمر من أوامر `Git` على أحد المشاريع يجب أن يكون لدينا مجلّد `git`. والذي يتم انشاؤه من خلال الأمر `git init` . دعونا أولا ننشئ مجلّد جديد في `Desktop` كالتالي.

```
01    > pwd
02    /Users/user/Desktop
```

بعد التأكد من تواجدنا في `Desktop` سنقوم بانشاء المجلّد الجديد وليكن بالاسم `git-test` .

```
03    > git-test
```

لنقم الآن بالدخول اليه واستخدام الأمر `ls` و `ls -a` لعرض الملفات والمجلدات سواء كانت مخفيّة ام لا.

```
04    > cd git-test
05    > ls
06    > ls -a
```

نلاحظ أن المجلّد فارغ تمام، بمعنى اننا لم نقم بانشاء `Repository` باستخدام الأمر `git init` بعد. لنقم باستخدام أمر `git status` للتحقق ان كان سيعمل بدون تواجد `Repository` ام لا.

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

نتجت عن الأمر رسالة خطأ كرد، والسبب هو عدم استخدامنا للأمر `` `git init` `` قبل استخدام الأمر، لذلك دعونا نقوم باستخدامه الآن.

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

الآن، بعد انشاء `Repository` نستطيع استخدام أوامر `Git`. لنقم باستخدام أمر ` `git status` ` مره 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.

أمر git add

كما ذكرنا سابقاً، يوجد مراحل تمر فيها الملفات في Git، أحد هذه المراحل هي Untracked وتعني ان الملف لا يتم تتبعه. كمثال، بمجرّد انشائك لملف جديد يعتبر الملف Untracked، لذلك ان كنت تريد ان يقوم Git بتتبعه يجب عليك نقله الى المرحلة التالية وهي Staged. ويتم ذلك من خلال استخدام الأمر `git add` مع ارسال اسم الملف الذي تريد نقله. بالتالي نستخلص من ما ذكر بأن الأمر git add يقوم بنقل الملفات من مرحلة Untracked الى Staged.

لرؤية المراحل التي يمر بها الملف لنقم بانشاء ملف جديد باستخدام الأمر touch كالتالي.

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

الآن بعد أن تأكدنا بانه تمّ انشاء الملف بنجاح، لنقم بالاستعلام عن حالة المشروع من خلال استخدام الأمر `git status`.

```
 > git status
On branch master
No commits yet
```

الأوامر في هذا المثال تابعة للأوامر المتواجدة أعلاه.

```
  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 انه قام بوضع الملف الجديد في مرحلة `Untracked`، وكما ذكرنا لتتبعه دعونا 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
```

مثال للعرض

بهذا الشكل قمنا بنقل الملف الى المرحلة الأخرى وهي `Staged/Tracked`.

نلاحظ الرسالة أعلاه، بأنه في حالة اردت التراجع عن تتبع الملف يمكنك استخدام الأمر `git rm`، دعونا نقوم باستخدامه لنرى 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 باعادة الملف الى مرحلة `Untracked`.

مثال للعرض

الأمر git commit

يقوم هذا الأمر بنقل الملفات من مرحلة `Staged` الى `Committed`.

دعونا أولاً ننقل الملف من مرحلة `untracked` الى `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
```

الآن، سنقوم باستخدام الأمر `git commit` لنقل الملفات الى مرحلة `Committed`، هذه العمليّة تسمى `commit` لكن يجب الأخذ بعين الاعتبار اننا نحتاج الى 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 (انشاء ملف جديد) في `Repository`. وللتحقق من انه لا يوجد تعديل آخر جديد سنقوم باستخدام الأمر `git status`.

```
> 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 0300+ 2021 17:10:58 28
    new file added
```

يستفيد Git من رقم hash لتحديد ما اذا تمّ التعديل على الملف ام لا. فأي تغيير على رقم hash هو دليل على تغيير محتوى الملف.

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 والايميل الخاص به. كما انه ايضاً قام بانشاء رقم hash لتحديد commit بحيث يدل كل hash على commit معيّن ولا مجال لتداخلهم.

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
```

مثال للعرض

مفهوم branches

تعتبر الفروع او `Branches` مسارات لتطوير المشروع، دعونا نفترض اننا نريد التعديل على المشروع من خلال اضافة خاصيّة جديده. لكن لسنا متأكدين من فعاليتها او امكانيّة تطبيقها، بالتالي لا نريد ان نخاطر بالتعديل على المشروع الفعلي. من هنا أتت خاصيّة الفروع، بحيث نستطيع أخذ نسخة من المشروع للتجربة واضافة الخصائص الجديده من خلال انشاء فرع جديد (مسار جديد للمشروع).

الأمر git branch

يستخدم هذا الأمر لعرض الفروع المتواجدة في المشروع.

مثال

```
01  > git branch
02  * master
```

من المخرجات نلاحظ وجود فرع واحد يسمّى `master` وهو الفرع التلقائي/الرئيسي الخاص بالمشروع. كما نلاحظ وجود علامة النجمة * وتعني اننا حالياً متواجدين في `master` وأي تعديل او اضافة حاصلة الآن ستكون في الفرع `master`.

مثال للعرض

الأمر git branch <branchName>

هذا الأمر يقوم بانشاء فرع جديد.

مثال

```
01  > git branch test
02  > git branch
03  * master
04    test
```

نلاحظ أعلاه بأنه فعلاً تمّ انشاء فرع جديد بالاسم `test`.

الأمر git checkout

يقوم هذا الأمر بالانتقال الى فرع آخر.

مثال

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

نلاحظ هنا أننا ما زلنا على نفس الفرع وأي تعديل سيكون في `master`، بالتالي ان 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`. وللتأكد أكثر دعونا نستخدم الأمر `git branch` مره أخرى.

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

مثال للعرض

مفهوم Merge

تطرقنا سابقاً الى مفهوم الفروع، وذكرنا انه يمكننا أخذ فرع من المسار الخاص بالمشروع والتعديل عليه كما نريد، لكن ماذا سنفعل عند نجاح الخاصيّة الجديده؟ عند نجاح الخاصيّة سنقوم بدمجها مع المسار/الفرع الرئيسي للمشروع.

الأمر git merge

يستخدم الأمر `git merge` لدمج الفروع مع بعضها.

عند دمج مسار بآخر سينقسم الدمج الى حالتين:

- دمج `fast forward`، وهو الدمج الحاصل عندما فقط تتم اضافة الخصائص في الفرع الجديد مع الفرع الرئيسي.
- دمج مع `conflict` او ما يسمى `merge true`، وهو ما يحدث عند ظهور تعارض `conflict` بسبب اختلاف نسخ المشروع، فعندما يقوم شخص بالتطوير على المسار الرئيسي وانت ما زلت تعمل باضافة الخاصيّة في الفرع الخاص بك من ثمّ حاولت الدمج. سيظهر اختلاف في النسختين وسيطالبك `Git` بحلّ هذا الاختلاف/التعارض.

مثال للعرض

مثال

لنقم بالتعديل على الفرع `test` من خلال انشاء ملف جديد فيه كالتالي.

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

الآن بعد انشاء الملف يجب أن يكون هذا الملف في مرحلة `untracked` ولنتحقق من ذلك سنقوم باستخدام الأمر `` `git status` `` كالتالي.

```
> 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 الى مرحلة staged من ثمّ committed بحيث يزيد بعدد commit واحد عن الفرع الرئيسي(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 انه اصبح 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 ان الملف الثاني تم انشاؤه في الفرع test بالتالي سنقوم

بدمج الفرع `test` مع `master` لجلب التحديثات الخاصّة بالفرع `test` في المسار الرئيسي باستخدام `git merge` .

```
> 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 لنتحقق من عدد الملفات باستخدام الأمر `ls`.

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

بالتالي نلاحظ انه بالفعل تمّ دمج الفرع `test` مع `master`.

مثال للعرض

التعريف بمنصة GitHub

مثال للعرض

التعامل مع فريق في Git and Github

قائد الفريق

- إنشاء `repo` فارغ على `Github`.
- على جهازك `mkdir project_folder_name` ثم `cd` و `git init` .
- على جهازك `git url` <`git remote add` > .
- تأكد من الربط باستخدام `git remote -v` .
- إنشاء بعض الملفات ثم `git add -A` ثم `git commit -m relevant message` .
- قم برفع الملف على `GitHub` باستخدام `git push origin master`.
- إنشاء فرع جديد باستخدام `git checkout -b newName`.
- إنشاء ملف جديد ثم `git add -A` ثم `git commit -m relevant message` .
- قم برفع الملف على `GitHub` باستخدام `git push origin newName`.
- إذا كان هناك تحديث على `GitHub` من أحد أعضاء الفريق اتبع الآتي: `git checkout master` ثم `git merge newName`.

مثال للعرض

أعضاء الفريق

- اذهب إلى رابط المشروع لقائد الفريق ثم Fork و clone مثل git clone <your forked git repo url> .
- ثم git remote add upstream <team leader git url>.
- ثم git checkout -b newName لإنشاء فرع جديد.
- ثم git remote add upstream <team leader git url>.
- إنشاء ملف جديد.
- ثم git remote add upstream <team leader git url>.
- ثم قم بتشغيل git checkout -b yourname-dev لإنشاء branch.
- أنشئ ملف جديد (marc.html مثلًا) و git add -A ثم git commit -m <relevant message>
- ثم `git push origin yourname-dev`
- ارسل جميع الpull request لقائد الفريق.
- يقوم قائد الفريق بدمج merge التغييرات للmaster.
- ثم قم بتشغيل `git pull upstream master` للحصول على الإصدار الحالي من قائد الفريق (locally). يجب أن يكون الmaster branch بعد هذه الخطوة مُحدّث لآخر تعديل مع upstream master.
- ثم git checkout yourname-dev
- ثم git merge master سيقوم بدمج آخر نسخة للdev branch

مثال للعرض

مثال للعرض