نظام Git

هو نظام يساعد على الاحتفاظ بتسلسل التغييرات التي تطرأ على مشروع ما خلال فترة زمنية معينة. أفضل طريقة للتعرف على Git هي بإستخدام Git نفسه. دعنا نتخيل أنك بيتر كريستين اسبيورنسن في العام ١٨٤١ بالنرويج تجلس على طاولة مكتبك لتكتب قصة للأطفال.

تقوم بفتح جهاز الحاسب الخاص بك وتقوم بإنشاء مجلد جديد، ثم تقوم بإنشاء ملف لتكتب فيه قصتك. هل كان بيتر يملك حاسوبا في ١٨٤١؟ بالطبع فهذا من وحي خيالنا فما المانع في ذلك؟ المهم انك تضع عنوانا للقصة: الجديان الثلاثة.

ولأنك كاتب متمرس، فإن أول ما تود أن تفعله هو ان تفتح برنامج Terminal وتكتب هذا الأمر حتى تسمح ل Git من متابعة التغييرات التي ستقوم بها في هذا الملف:

|  |
| --- |
| git init |

يعتمد Git على ثلاثة بيئات للعمل:

1. Working Directory: المكان الذي تقوم فيه بإنشاء وحذف وتعديل ملفات المشروع.
2. Staging Area: المكان الذي نقوم فيه بإدراج فيه الملفات التي نرغب ل Git أن يحفظها.
3. Repository: المكان الذي يستخدم للتخزين النهائي للملفات.

الآن وبعد أن قمت بتهيئة المشروع للعمل مع Git، فقد تود الاطلاع على الوضع الحالي للملفات لتتمكن من رؤية جميع الملفات تحت إدارة Git وهل تمت إضافتها لل Staging area أم لا تزال في الـ working area:

|  |
| --- |
| git status |

ثم تقوم بإضافة الملفات لل Staging area تمهيدا لحفظها نهائيا:

|  |
| --- |
| git add fileName |

ولكن قبل الحفظ النهائي قد تود في رؤية جميع التغييرات في الملف:

|  |
| --- |
| git diff |

اخيرا انت راض عن عنوان القصة فتقوم بحفظ هذه التغييرات مع إضافة رسالة ليسهل عليك التعرف على كل تغيير قمت به من دون الدخول للملف:

|  |
| --- |
| git commit -m “Add story title” |

بيتر (انت بيتر، الكاتب تذكر!)، لقد كتبت قصتك الآن على عدة مراحل متبعا نفس النظام السابق:

* تعديل الملف
* إضافته لـ staging area
* ثم حفظه في ال repo

يمكنك الاطلاع على كل التعديلات التي قمت بها الآن (ومن على [Github](https://github.com/bashmoanas/TheThreeGoats/commits/main/TheThreeGoats.md)):

|  |
| --- |
| git log |

لقد كتبت قصة جميلة حقا ولكنك لست متأكدا من النهاية التي كتبتها بشكل كاف، تفكر انه يجب ان تكون طريقة ما لكتابة نهاية مختلفة دون المساس بالنهاية التي كتبتها مسبقا. بالطبع أنت محق وإلا فلم نحن هنا. يمكنك من خلال Git ان تقوم بعمل branch (فرع) جديد من قصتك تكتب فيه النهاية الجديدة. (ومن على [Github](https://github.com/bashmoanas/TheThreeGoats/commits/anotherVersion)):

|  |
| --- |
| git branch branchName // Create new branch  git checkout branch // Go to the new branch |

لقد كنت محقا فالنهاية الثانية تعجبك اكثر وتقرر الاحتفاظ بها. حينها يمكنك التخلص من ال branch الذي قمت بإنشائه سابقا فقد أتم مهمته على أكمل وجه.

|  |
| --- |
| git checkout main // Go to the main branch  git merge branchName // merge the new End to main  git branch -d branchName // Delete the new branch |

بيتر، لقد كتبت كتابا رائعا ولكن لا يمكن عمل كل شيء بمفردك. تقوم باستدعاء المراجع اللغوي ودار النشر لمساعدتك على إنهاء الكتاب. بإمكانهم عمل نسخة من ال repo على جهازهم الخاص للعمل عليها دون المساس بنسختك.

|  |
| --- |
| git clone repoLocation cloneNameOfChoice //clone repo |

بهذه الطريقة يمكن لكل شخص العمل على نسخته الخاصة بشكل منفصل عن النسخة المشتركة بينكم جميعا والتي يتم تسميتها عادة origin. عادة تكون طريقة العمل كالتالي:

* يقوم الشخص بعمل fetch لاية تعديلات من ال origin
* ثم يقوم بعمل دمج merge لهذه التعديلات في نسخته الخاصة على الحاسب
* ثم يقوم بعمل branch جديد لعمل التعديلات التي يريدها
* تجنبا أن يكون قد تم تحديثات على النسخة المشتركة، يقوم بعمل fetch مرة اخرى وmerge.
* في النهاية يقوم بإضافة تعديلاته على النسخة المشتركة من خلال push.

|  |
| --- |
| git fetch  git merge origin/branchName  git push branchName origin |

حسنا يمكن لبيتر أن يرتاح الآن. إذا كنت ترغب في معرفة المزيد عن قصة الجديان الثلاثة برجاء قراءة هذه السلسلة من [التغريدات](https://mobile.twitter.com/labanshy/status/1424099093488586761?lang=ar-x-fm). يمكنك الإطلاع على المشروع على [Github](https://github.com/bashmoanas/TheThreeGoats).

قد تتسائل الآن، ولكنني لست كاتبا نرويجيا يكتب قصص أطفال منذ قرون مضت، انا مبرمج، كيف يمكن ل Git أن يساعدني.

إذا كنت تعمل بمفردك، فبإمكانك استخدام ال branching في Git لتطوير خاصية جديدة دون المساس بالكود المكتوب سلفا والذي يعمل بالفعل. في هذه الحالة سوف تتمكن من التركيز على الخاصية التي تود إضافتها وتستطيع ان تقوم بتجربتها قبل إضافتها للكود الذي كتبته بالفعل.

اما اذا كنت تعمل مع فريق عمل، ف Git لا غنى عنه. فبإستخدام ال branching يمكن لكل فرد في الفريق أن يساهم في المشروع دون حدوث مشاكل في الكود الذي تم كتابته مسبقا.

كما يتم إستخدام Github\* بحيث يحتوي على النسخة المشتركة من ال repository ويكون لدى كل فرد نسخته الخاصة ليعمل عليها دون الحاجة حتى لوجود الانترنت.

يسمح Github بعمل Pull request وذلك حتى تتمكنوا كفريق من مناقشة التعديلات التي تمت قبل إضافتها للنسخة المشتركة بينكم.

اذا لم تستعمل Git من قبل فقد يبدو لك معقدا. وهذه صحيح لذا ابدأ بإستخدامه فورا حتى تتأقلم عليه سريعا 🤷🏻‍♂️.

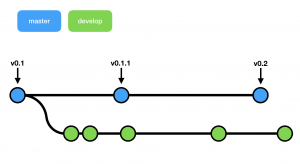
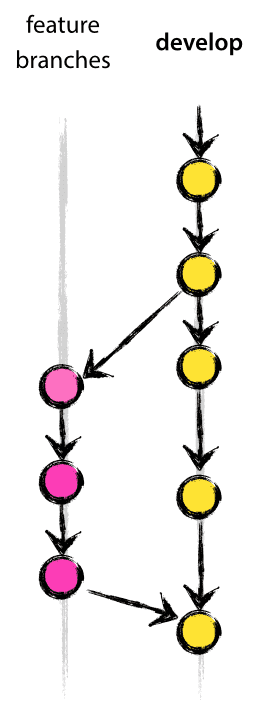
* مواقع Github او GitBucket او GitLab كلها أدوات ل Git ذات واجهة استخدام.

يعني ايه Git-flow؟

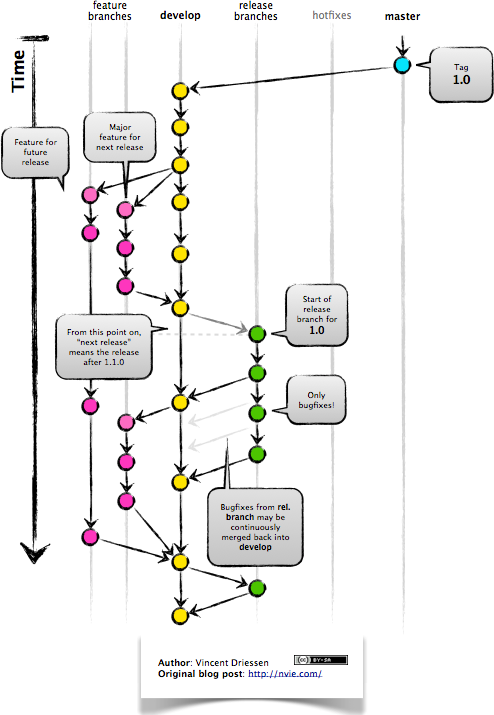
هو git workflow بيساعد انه يخصص أدوار معينة  للفروع (Branches) المختلفة، وبتحدد امتى وازاي بيتعاملوا مع بعض.  باختصار هو بيحدد ايه الفروع(Branches) اللي نبنيها وازاي نجمعهم سوا.

طب بيشتغل ازاي؟

في البداية لازم يكون عندنا فرعين (2 Branches) ال **Main Branch** ده عبارة عن نسخة ال production المستقرة من المشروع، وال **Develop Branch** بنستخدمه لدمج ال Features في كل مرة بنعمل Feature جديدة أو بنصلح Bug.  والفرع ده بيحتوي على التاريخ الكامل للمشروع.

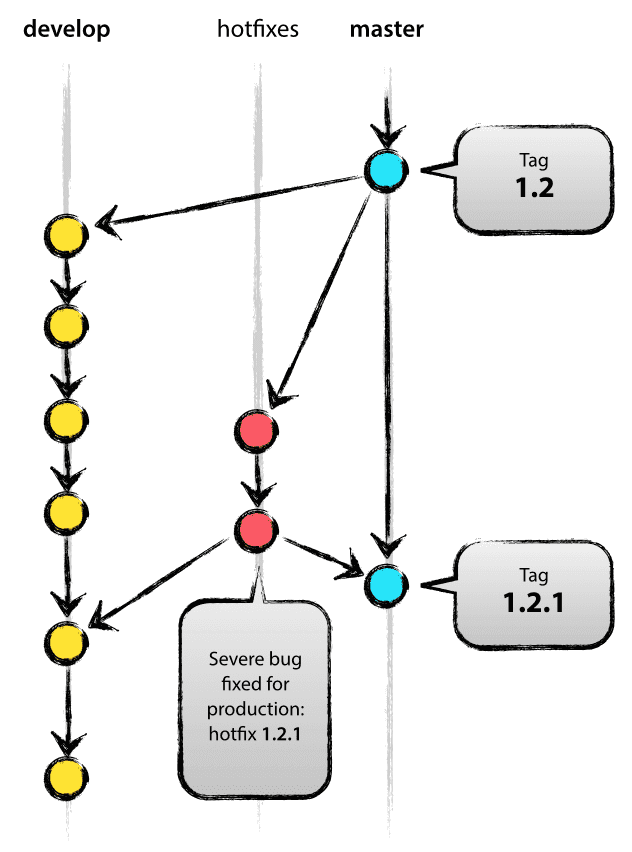


ومع التعديلات بنوصل إلى نسخة جديدة فبنعملها **Release-Branch**، وهنا بنختبرها ( بس مش بنضيف ليها اي Features جديدة بنحل ال Bugs اللي تظهر فقط).  الميزة هنا او الفائدة من ال workflow اننا نقدر نكمل تطوير او نعمل Features جديده بس بتكون في ال **Features-Branches** وكله بيكون متزامن مع بعض.



بعد ما بنتأكد من أن النسخة الجديدة مستقرة وجاهزة للانطلاق بندمجها مع ال **Main Branch** وبتكون دي ال Version الجديدة اللي شغالة.

احياناً بعد ما نrelease ممكن نكتشف Bug، وبتكون دي لازم تتصلح في اسرع وقت و بالتزامن معاها بيكون المبرمجين شغالين على ال **Develop Branch** ف بنعمل واحد جديد **HotFix Branch** من ال **Main Branch.** بنصلح المشكلة وندمج تاني مع ال **Main Branch**.



**استخدام Git بطريقة أفضل في مشروعك**

كما عرفنا في الجزء السابق عن Gitflow فإنها تعتبر من أفضل الممارسات حيث أنها تساعد في Parallel Development العمل علي اكتر من خاصية في نفس الوقت و يساعد في التعاون بين المطورين في نفس الخاصية و يساعد علي رؤية و متابعة ما يفعله كل مطور و الاستفادة من

Release Staging Area و Support for Emergency Fixes

ومن أفضل ممارسات استخدام Git أيضا:

* إضافة .gitignore في بداية مسار مشروعك حيث يعلم Git بالملفات و المسارات التي لا نريد تتبعها مثل الملفات التي تم إنشاؤها من بناء و تشغيل المشروع
* استخدم الأسماء المستعارة لأوامر git التي تستخدمها كثيرا مثل

**git config --global alias.co checkout**

**>> git co main**

* اجعل commits الخاصة بك واضحة و ذات هدف واحد فمثلا عندما تحاول تصليح خطأ و اكتشفت خطا أخر اجعل لكل واحد commit الخاص به حتي لا ينتهي commit بتغييرات كتير ليس لها صلة ببعض.
* اجعل الرسائل الخاصة ب commit معبرة عن التغييرات التي حدثت حتى يسهل الرجوع لها فيما بعد.

**Right: Add feature to alert admin for new user registration**

**Wrong: Added feature**

* اجعل Pull Request الخاص بك يحتوى علي كل المعلومات المطلوبة ليتم مراجعته بسهولة و دمجه مع الفرع الرئيسى و يفضل استخدام Pull Requests Templates لكي يتم انشاء شكل الوصف تلقائيا و عليك ادخال فقط التغييرات التي قمت بها
* استخدم الامر **git log --oneline** لعرض السجل في صورة مختصرة ويمكن الانتقال لأى نسخة باستخدام الرقم المختصر الموجود بدلا من استخدام Commit ID كما يمكنك استخدام اكتر من امر لعرض السجل بشكل اخر مثل لو اردت عرض commits الخاصة بمطور معين

**git log --author = ali**

* احذف Local Branches التي تم حذفها من Remote Repository
* يفضل استخدام العلامات tags لتكون المرجع لنقطة معينة في سجل git مثل استخدامها لتحديد الإصدارات من المشروع **git tag 1.0.0 dc7a3cf**
* استخدم **git stash**  اذا كنت تريد حفظ التغييرات لاستخدامها لاحقا مثل اردت العمل علي خاصية جديدة و ترك تعديلاتك علي الخاصية الحالية للرجوع لاستكمالها فيما بعد
* استخدم **git bisect** لمعرفة أول commit يحتوى علي خطأ لكن يجب أن تخبر bisect علي commit ID يحتوي علي الخطأ و commit ID أخر لا يحتوى علي خطأ و سيبدأ bisect في البحث عنه

**git bisect start**

**git bisect bad HEAD**

**git bisect good dc7a3cf**

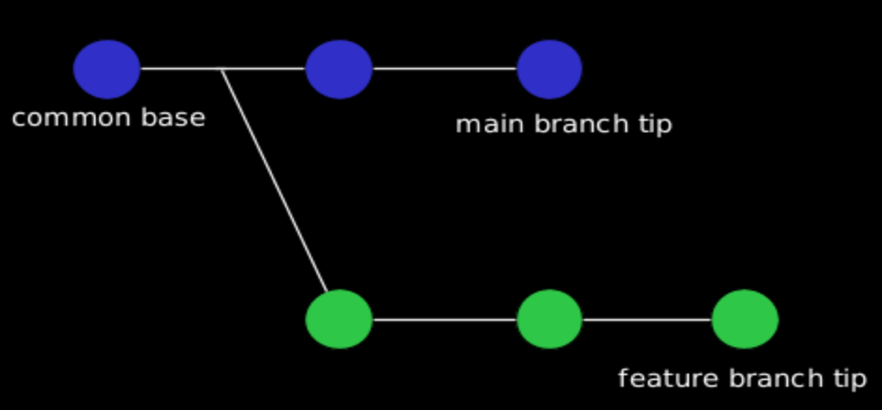
**git bisect reset**

Start git practices for teams

// to do

: git merge و git rebase الفرق بين

في البداية محتاجين نعرف انهم مجرد طريقتين لدمج الفروع بمعنى

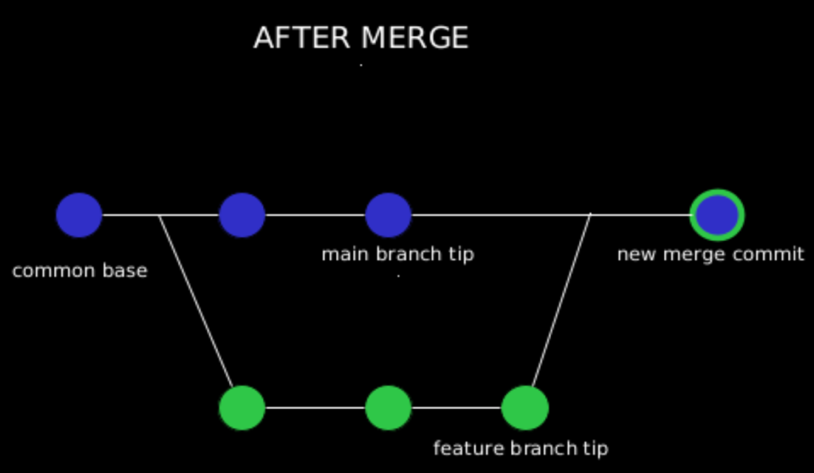


commit يحتوي على اكتر من التزام feature branch لو عندك فرع جديد -  
main branch في الفرع الرئيسي feature branch محتاج ادمج فرع -

يمكن ان يقوموا بعملية الدمج بين الفرعين لكن مع اختلاف طريقة الدمج git rebase او git merge سواء

***:git merge كيف تعمل***

في سجل واحد موحد commits بدمج سلسلة متعددة من الالتزامات git merge تقوم

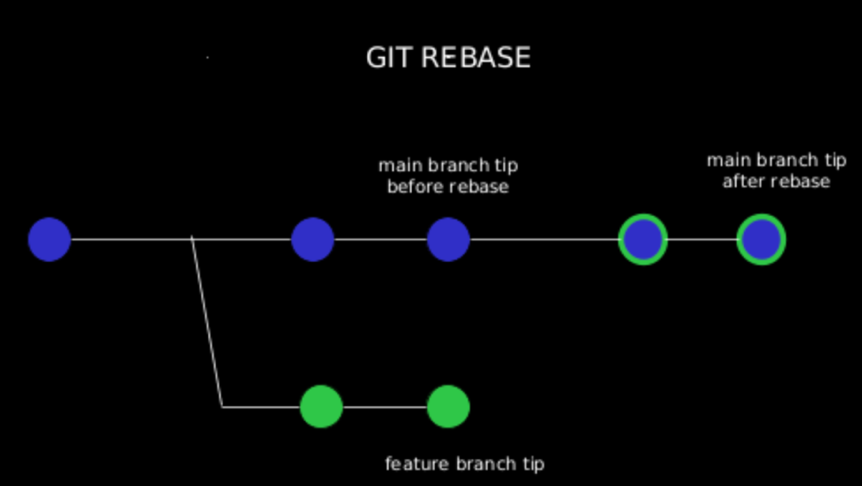


main branch بعد الدمج يصبح لدينا التزام جديد واحد في الفرع الذي ندمج فيه

feature branch ويحتوي هذا الالتزام على التغييرات من فرع

***:git merge كيف تعمل***

بترحيل أو جمع بين سلسلة من الالتزامات الي التزام أساسي جديد git rebase تقوم



**؟ rebase او merge الان يجب ان تقرر ماذا تستخدم عند دمج فرعين**



***:git merge متى تستخدم***

مرة أخرى الي الفرع الرئيسي feature branch إذا اردنا اضافة تغييرات فرع الميزة -

بدلا من اعادة كتابته History إذا كنت تريد الاحتفاظ بنفس السجل -

إذا كنت تريد التراجع عن التغييرات بسرعة -

***:git rebase متى تستخدم***

feature branch إذا اردنا اضافة تغييرات من الفرع الرئيسي الى فرع الميزة -

نظيفا git الاحتفاظ بسجل -

التراجع عن التغييرات سيكون صعب جدا -