# Worflow d'utilisation de GIT pour DAISY

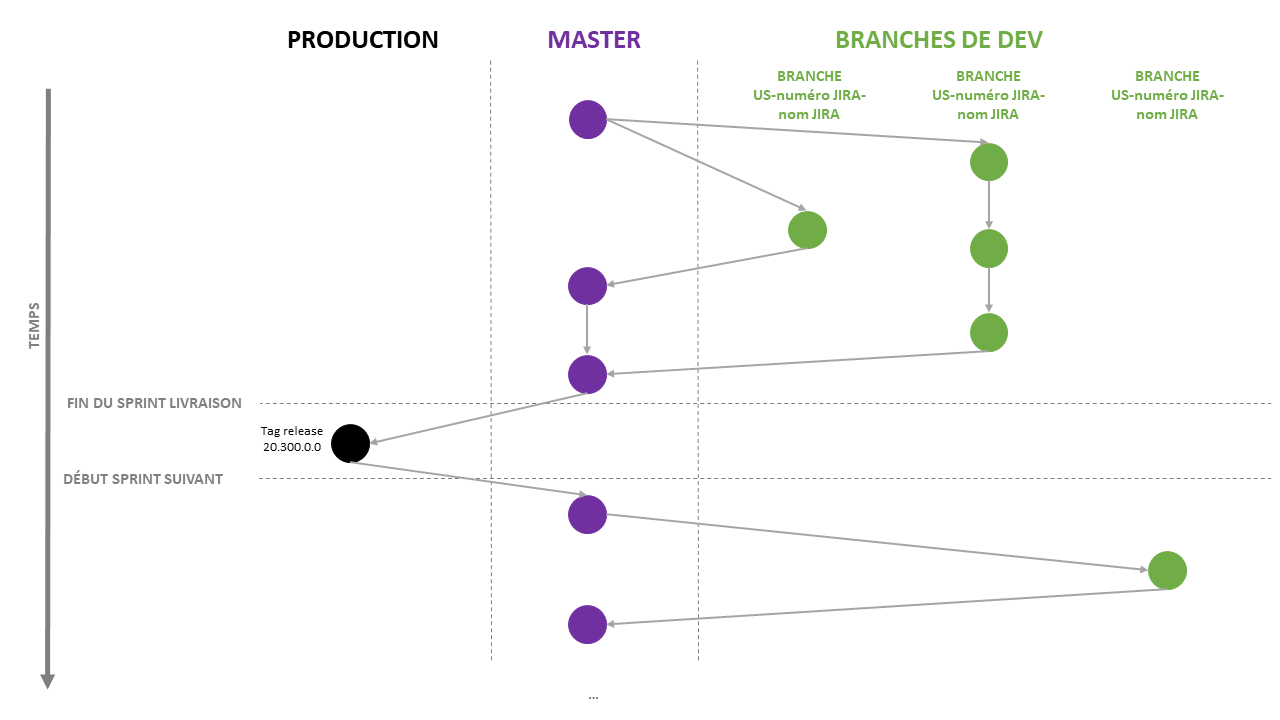
Sources d’inspiration :

<https://www.nicoespeon.com/fr/2013/08/quel-git-workflow-pour-mon-projet/>

<https://docs.gitlab.com/ee/topics/gitlab_flow.html>

### Worflow GIT pour daisy

* ***master*** : branche principale de développement. Lors du démarrage d’une US ou correction d’anomalie, il faut créer une nouvelle branche partant de la dernière version de la branche master. La branche master ne doit donc contenir que des versions stables, pas de développements en cours (à effectuer sur leurs branches spécifiques).
* ***US / QC*** : branches spécifiques à chaque US / QC, à partir de la branche master. Ces branches sont mergées sur la branche master une fois que l’US / QC est terminée.
* ***production*** : branche de production contenant uniquement les releases. Lors d’une release, on merge la branche master sur la branche production.



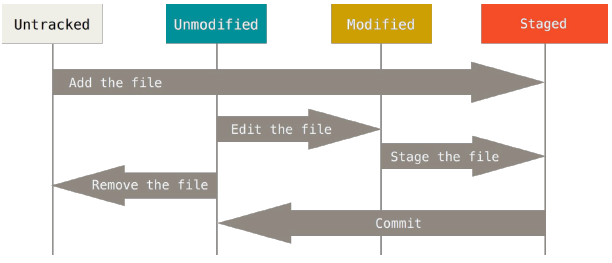
### Règles de nommage des branches :

* Au démarrage d’une US, création d’une branche à partir de la branche master qui reprend le formalisme suivant : US-NUMERO-nom de l’US tel que défini dans JIRA.
* Au démarrage d’une correction d’anomalie, création d’une branche à partir de la branche master qui reprend le formalisme suivant : QC-PALIER-NUMERO-titre de l’anomalie tel que défini dans HPQC.

### Bonnes pratiques :

1. Comme indiqué précédemment, pour toute nouvelle US ou toute nouvelle QC, créer une branche avec le nom approprié.
2. Chaque soir, ajouter, commiter et pousser sur sa branche toutes les modifications effectuées durant la journée. Le message du commit doit être de la forme des messages de Sirhen, sauf les merges. Ainsi, nous aurons par exemple :
   1. “[AUTRES][Message]”
   2. “[QC][111][Intitulé QC - Message éventuel]”
   3. [US][111][Intitulé US - Message éventuel]”.
3. Chaque matin, récupérer le contenu ajouté sur le Master en effectuant un merge du Master sur sa branche. Veiller à corriger tout conflit.
4. Dès que l’US est terminée, commit/push l’ensemble du travail sur sa branche distante.
5. Prévenir la personne qui testera (cf. 6) tout ce qui peut manquer dans la finition complète de l’US.
6. Laisser une autre personne tester sa branche :
   1. En cas de problème, le développement sera à reprendre et le processus reprendra au point 2.
   2. Si les tests sont concluants et l’application stable, la personne qui vient de tester procède au merge de la branche vers le Master et en avertit le développeur.
7. Le développeur initial prévient la personne qui doit qualifier des éventuels manques pour finaliser à 100% une US. La personne qui qualifie avertit le développeur initial du résultat. En cas de rejet ou de QC, le développeur initial reprend en 1.

### Comprendre GIT :



Cycle de vie des états de fichiers

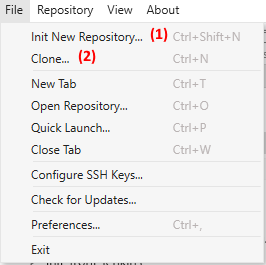
Un développeur procède, en général, à trois actions :

1. Il modifie les fichiers de son dépôt à l’aide d’une commande add
2. Il indexe ces fichiers à l’aide d’une commande de commit
3. Il valide ces modifications à l’aide d’une commande de push vers le serveur distant

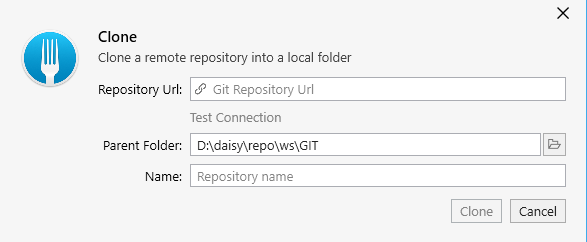
### Utiliser GIT :

#### Avec Fork :

##### Au démarrage d’un projet :

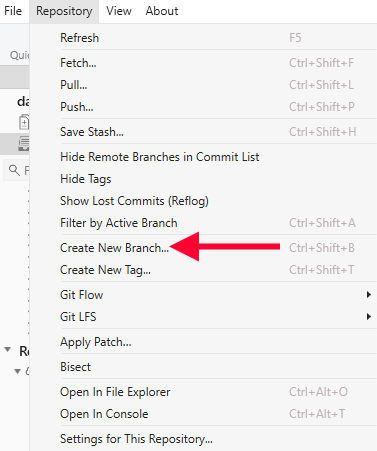


**Dans “File”**, en (1), on initialise un nouveau projet Git en lui spécifiant un répertoire  
**Dans “File”**, en (2), on clone un projet Git à partir d’une adresse et d’un nom qu’il faudra renseigner (cf. ci-dessous).

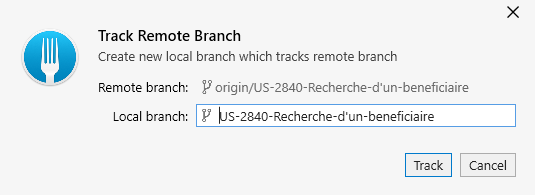
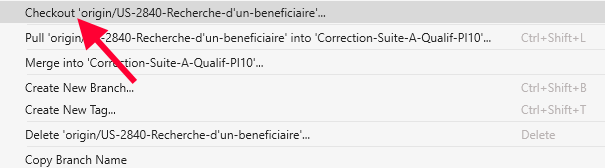


##### Actions traditionnelles en cours de projet :

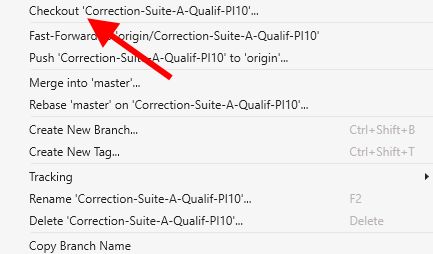
Pour créer une nouvelle branche, **cliquer sur “Repository”** puis sur **Create new branch…** :



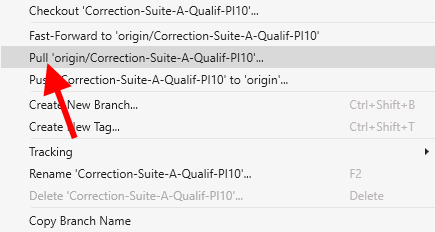
Ou, tout simplement, pour récupérer une branche distante, **clic droit sur la branche** souhaitée puis **cliquer sur checkout ‘<remote>/<nom-de-branche>’** :



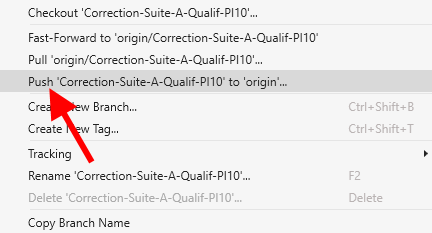
Ensuite, pour changer de branche (sans la créer), faire un **clic droit sur la branche** souhaitée puis **cliquer sur** **checkout ‘<nom-de-branche>’** :



Pour récupérer les données de la branche distante, faire un **clic droit sur la branche** souhaitée puis **cliquer sur pull ‘<remote>\<nom-de-la-branche>’** :



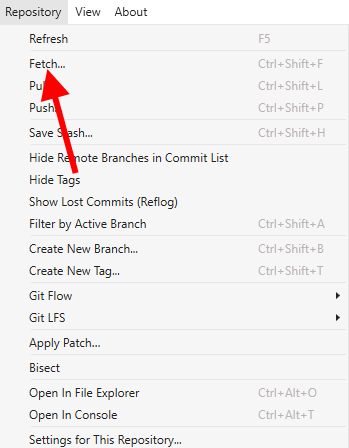
Pour add, commit et push en même temps (donc ajouter ses données locales sur une branche distante), faire un **clic droit sur la branche** souhaitée puis **cliquer sur push ‘<nom-de-la-branche>' to '<remote>’** :



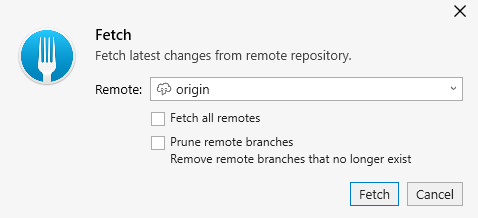
Sélectionner ensuite la branche locale et la branche distante sur laquelle push le contenu ajouté.

##### Actions de “finition” de projet :

Pour importer les informations du dépôt (<remote>, souvent origin) distant, voire d’une branche en particulier, **cliquer sur “Repository”** puis sur **Fetch…** :

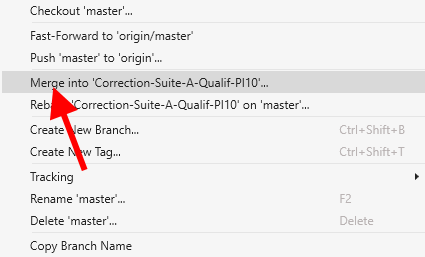


Sélectionner ensuite le <remote> souhaité, comme ici, “origin” :



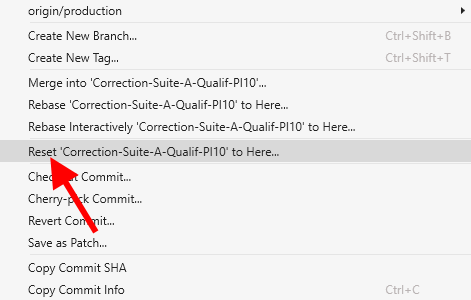
Note : l'import se fait à travers des branches spéciales pour donner la possibilité de comparer et si besoin fusionner manuellement.

Pour réaliser un merge (fusion de branches en l'état), **se placer sur la branche qui recevra** (de préférence, d’abord une branche accueillant le Master puis l’inverse, permettant de corriger les conflits) et faire un **clic droit sur l’autre branche qui doit “transmettre” ses données** puis **choisir “Merge into <branche>** :

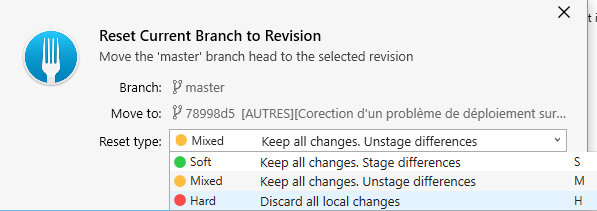


##### Actions bonus :

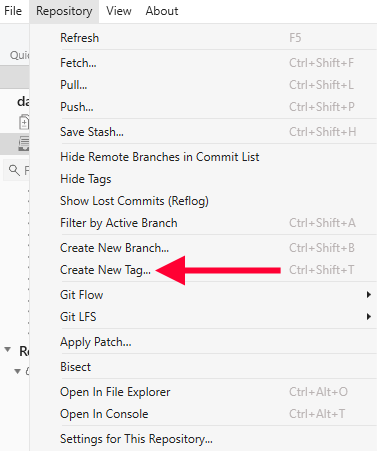
Pour revenir en arrière dans un commit/push et, éventuellement, effacer toutes ses modifications, faire un **clic droit sur un descriptif de commit/push** (au centre de l'écran, sur Fork) et **sélectionner Reset '<nom-de-branche>” to Here…** :



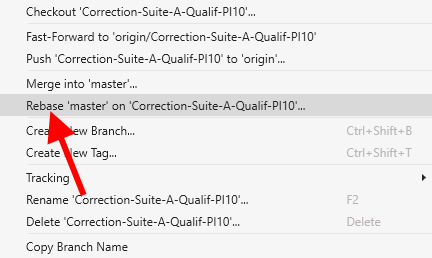
Ensuite, **sélectionner le type de reset**, souvent en “Hard” pour bien repartir à zéro :



Pour créer un nouveau tag, **cliquer sur “Repository”** et **choisir “Create New Tag”**.



Pour procéder à un rebase (mise bout à bout d’une “branche2” qui “suit” niveau commit/push une “branche1”), à l’inverse du merge, **se positionner sur la “branche2”** et faire un **clic droit sur la “branche1”** puis **sélectionner Rebase <branche2> on <branche1>** :



**Note :** il est préférable dans notre cas d’utiliser un git merge

#### En ligne de commandes (sous cmd ou git bash) :

##### Au démarrage d’un projet :

|  |  |
| --- | --- |
| **Commandes** | **Description** |
| git init --bare | Initialise un dépôt Git dans le répertoire courant |
| git clone <adresse> | Récupère le dépôt Git et en crée le répertoire adéquat |

##### Commandes traditionnelles en cours de projet :

|  |  |
| --- | --- |
| **Commandes** | **Description** |
| git checkout [-b] <nom-de-branche> | Change de branche. L’ajout de -b permet de créer **en local** une nouvelle branche puis de changer de branche pour s’y positionner dessus.  Note : annule les modifications de fichier (git add) avant commit ! |
| git pull [--all] | Récupère le contenu distant, à condition de ne pas avoir de conflit. Le paramètre --all est optionnel |
| git status | Permet de voir ce qu’il reste à ajouter, commiter et push |
| git add **.** | Permet de “modifier”, c’est-à-dire ajouter au futur commit, les fichiers du répertoire courant. Remplacer le point par un nom de fichier si on ne veut en ajouter qu’un par exemple |
| git commit -m “message ici“ | Permet d’indexer les changements avant de les envoyer au serveur distant. Veiller à bien respecter les recommandations ci-dessus en matière de nommage des messages. |
| git push [--set-upstream <remote> <branche>] | Permet de “valider” son contenu en l’envoyant sur le serveur distant. En cas de nouvelle branche n’existant pas encore sur le serveur distant, il faudra ajouter l’option --set-upstream <remote> <branche>, <remote> valant le plus souvent “origin” et <branche> le nom de sa branche ou “master” |
| git revert [HEAD] | Annule le commit/push et revient au précédent. Remplacer HEAD (précédent) par ce qui est souhaité :   * HEAD^ (avant-dernier commit) * HEAD^^ (ou HEAD~2, pour l’avant-avant-dernier commit) * <hash> (pour le commit de ce hash. Se trouve par exemple avec git log) |

##### Commandes de “finition” de projet :

|  |  |
| --- | --- |
| **Commandes** | **Description** |
| git branch | Permet de voir toutes ses branches en local |
| git merge <branche-cible> | Fusionne la <branche-cible> dans la branche sur laquelle on se trouve. Corriger les conflits via tout éditeur de texte ou Visual Studio Code. Pour le merge final, se placer sur le Master et répéter la même opération avec la branche de l’US ou de la QC testée et à jour comme <branche-cible> |
| git remote | Liste les dépôts distants |
| git remote show <remote> | Montre les branches et leur mise à jour ou non sur le dépôt <remote> (le plus souvent, ce sera origin) |
| git fetch <remote> [<branche>] | Importe les informations du dépôt (<remote>, souvent origin) distant, voire d’une branche en particulier. L'import se fait à travers des branches spéciales pour donner la possibilité de comparer et si besoin fusionner manuellement. |

##### Commandes bonus :

|  |  |
| --- | --- |
| **Commandes** | **Description** |
| git log | Permet de voir les cinq derniers commit/push |
| git reset [HEAD] | Annule le dernier commit. Le HEAD signifie dernier commit et n’est pas obligatoire. Toutefois, il est possible de changer HEAD par ce qui suit :   * HEAD^ (avant-dernier commit) * HEAD^^ (ou HEAD~2, pour l’avant-avant-dernier commit) * <hash> (pour le commit de ce hash. Se trouve par exemple avec git log) |
| git reset --hard [HEAD] | Revient au dernier commit/push. Tout le contenu supplémentaire en local sera perdu ! HEAD peut être remplacé par ce qui se trouve à la ligne ci-dessus |
| git reset -- fichier-à-supprimer | Permet de retirer un fichier du commit avant de faire un push |
| git commit --amend | Permet de modifier le message du dernier commit |
| git mv fichier-origine fichier | Renomme le fichier |
| git rm fichier | Supprime le fichier |
| git tag -a <tag> -m “Message…” | Crée un tag (nom <tag>) en y ajoutant un message explicatif. |
| git show <tag> | Permet de montrer les informations de commit/push du tag |
| git push <remote> <tag> | Envoie le tag créé vers le dépôt distant (le plus souvent, origin) |
| git rebase <branche1> <branche2> | Met les commit/push historiquement à la suite, avec branche1 en premier (plus vieux) et branche2 à la suite. Il est préférable dans notre cas d’utiliser un git merge |