Processus de développement

On y est ! Tu vas enfin développer sur une tâche cliente !
Pour ce faire, une méthodologie précise est à prendre en main, qui permet notamment de retrouver simplement un développement d'après la tâche, de déclencher plus simplement le processus de facturation … etc !
Pour cela, voici comment ça se passe !

1. Tâche Odoo

> Tip
>
> a résolution des tickets suit globalement le même processus notamment parce que les tickets doivent être associées à des tâches. Dans la tâche associée au ticket, il faudra choisir le bon "article du bon de commande" créé par le chef de projet. Ceux-ci existent par client et pour chaque grand domaine d'intervention : stocks, ventes, etc.

1.1 Description

La réalisation d'un développement dans le cadre d'une tâche cliente repose en premier lieu sur la création ou le remplissage d'une **tâche** cliente présente dans le module de **Gestion de projet**.
La description de la tâche doit se situer (idéalement) dans un niveau d'abstraction inférieur à la demande du client. **C'est à dire que l'on commence à traduire le langage du client dans le langage d'un développeur**. Elle doit contenir **toutes les informations** nécessaires au développement afin d'éviter des écarts coûteux entre le travail réalisé et la demande du client.
La description peut être saisie directement dans l'onglet description ou être dans un fichier joint à la tâche. Dans ce cas, on écrit « voir PJ » dans la description.

> Tip
>
> Si le rédacteur a des astuces ou des conseils qui peuvent aider le développeur, il faut les mettre. **Tout ce qui peut empêcher des écarts et faciliter la compréhension est utile.**

1.2 Titre de la tâche

Le titre de la tâche doit pouvoir permettre de savoir en un coup d'œil en quoi consiste la tâche (grosso modo).
Exemple de mauvais titre : « POS ». Bon titre : « Mise en place d'un double contrôle de la caisse ».
Il faut éviter des abréviations qui ont des sens multiples (par exemple : MEP - "Mise en place" ou "Mise en production" ?)

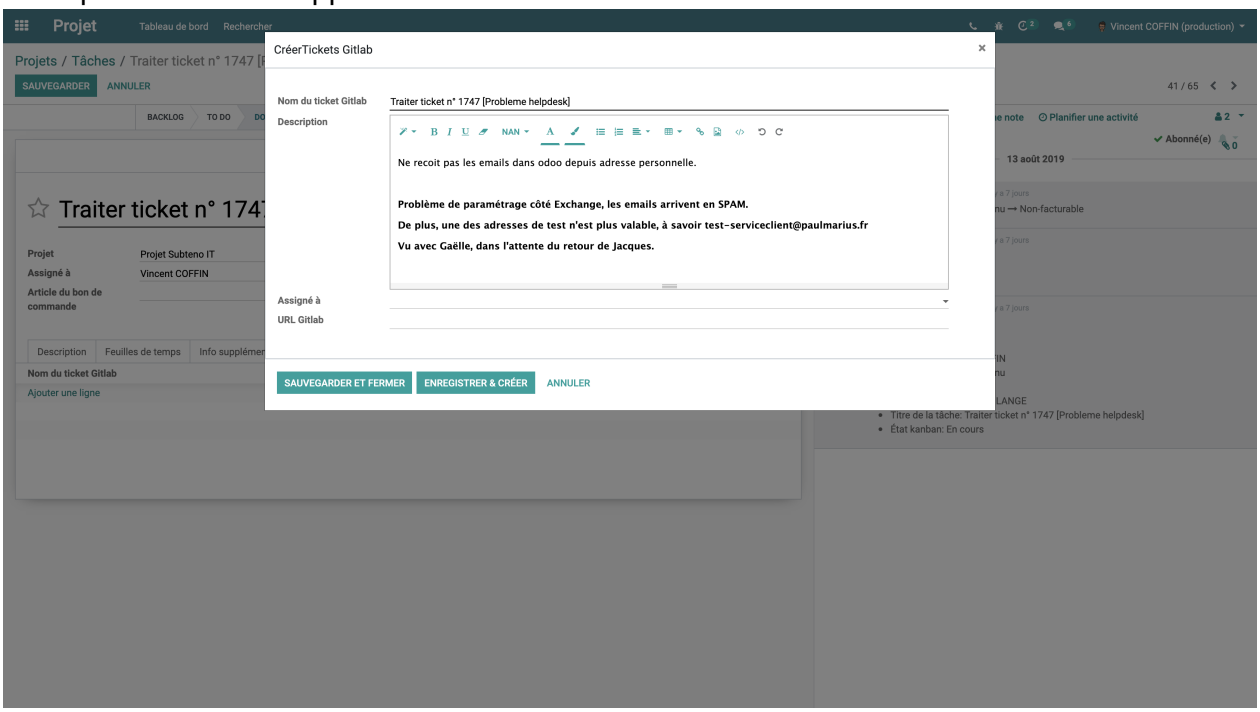
1.3 Client

Il ne faut pas oublier de renseigner le client sur la tâche. Le client, s'il est une personne, sera automatiquement ajouté à la liste des abonnés, dans le cas d'un échange d'email.

> Warning
>
> Les échanges d'e-mails, aussi bien dans le cadre du développement d'une tâche que lors du traitement d'un ticket de support, doivent se faire **EXCLUSIVEMENT DEPUIS ODOO**. Cela permet à tous, lorsque vous partez en congés, de reprendre la suite de votre travail avec tous les échanges entre vous et le client !

1.4 Gitlab : Création de ticket et de branche

Vous avez créé votre tâche ou vous êtes assigné une tâche client. Il est désormais possible de créer une **issue Gitlab** depuis Odoo, qui sera par la suite liée à la branche que vous allez créer pour votre développement.



Dans la tâche, il y a un volet ticket Gitlab, qui vous permet de renseigner / créer un nouveau ticket Gitlab depuis Odoo.
Il faut simplement ajouter la personne assignée au ticket, puis cliquer sur **sauvegarder et fermer**. Enfin, **sauvegardez** la tâche, et un lien sera automatiquement attribué au ticket dans Odoo via l'API Gitlab.
Dans l'URL, le numéro d'issue sera précisé. Ce numéro **doit être repris lors de la création de la branche de développement en local. De cette façon, Gitlab fera automatiquement le lien entre votre **Merge-Request (MR)** et l'issue.

2. Création de notre nouvelle branche

Avant de commencer à coder, il est important d'avoir son dépôt local à jour, pour ce faire:

```
git checkout master
git pull
```

Maintenant vous pouvez créer votre nouvelle branche en respectant la convention de nommage.

```
git checkout -b 2-ma-tache
```

Il est important de pousser sa branche et de créer un merge request avant de commencer la tâche

```
git push --set-upstream origin 2-ma-tache
```

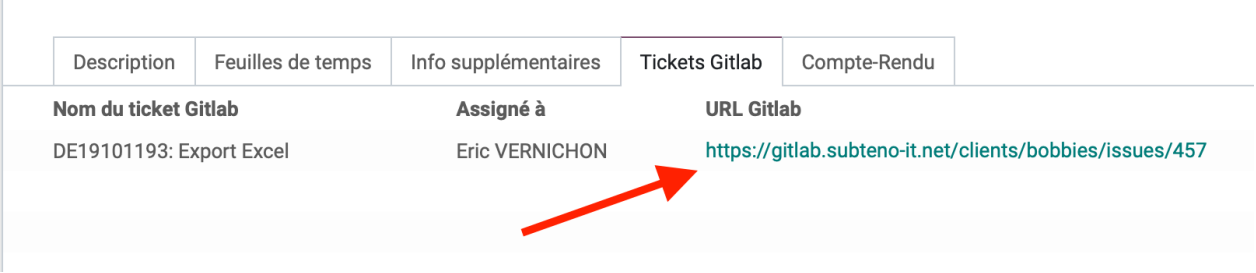
Git vous retourne le lien pour créer la MR. Il est important de créer dès à present votre MR. Gitlab va automatiquement mettre votre MR en WIP (Work In Progress) car il a détecté aucune modification avec la branche master du dépôt client.

> Tip
>
> Si le titre de la MR ne commence pas par WIP:, il y a de forte chancge que vous n'avez pas mis à jour votre branche master à jour.

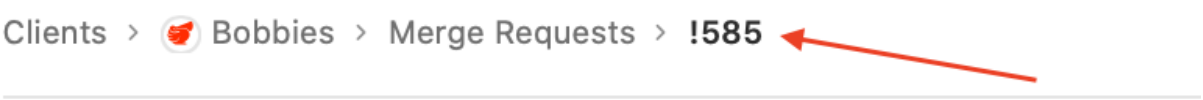
Votre dépot n'est accessible que par vous d'où l'importance de créer la MR dès le départ afin qu'il soit possible de reprendre votre travail si besoin.

3. Reprendre la suite d'un développement du collègue

Vous venez de reprendre une tâche d'un(e) collègue, pas de panique, votre collègue a fait une MR :) À partir de de la tâche Odoo, onglet Gitlab, cliquez sur l'issue pour retrouver la MR dans Gitlab.



Récupérez l'ID de la dernière MR :



On se remet sur la branche master

```
git checkout master
```

On en profite pour mettre à jour notre branche master

```
git pull
```

On va récupérer la MR de notre collègue

```
git fetch subteno refs/merge-requests/585/head
```

Git nous indique que l'on peut récuperer le code en création une branche à partir du FETCH_HEAD

```
git checkout FETCH_HEAD
```

Git nous invite à créer une branche en local, il est préférable de la nommer à l'identique de celle de notre collègue

Une fois copié, nous créons notre branche

```
git checkout -b 457-export-excel-reassort
```

Et on n'oublie pas de pousser notre branche et de créer notre MR de suite.

4. Commits

Les commits sont importants pour permettre de retrouver rapidement quel développement a été réalisé à un moment donné.

4.1 Nommage des commits

Les commit devraient avoir des noms concis mais clairs. Il est hors de question de devoir aller dans le code pour savoir ce que le commit change. Parce qu'on ne fait pas le même travail plusieurs fois, on évitera donc les commit avec le même nom.
Enfin, les commit ne doivent pas être nommés « f » ou « test » ou encore « makefile ».
Pour gagner de l'espace dans le nom du commit, on peut éviter de conjuguer les verbes ou encore de mettre les prépositions. Il faut garder à l'esprit que c'est le message qui compte et qu'on doit savoir de quoi il s'agit avant tout.
Plus simplement, on nomme ses commit de la même manière que ses variables. On veut juste pouvoir s'y retrouver.
Les noms de commit doivent être en anglais.

4.2 Préfixes

[WIP] Cela indique que le travail stipulé dans le nom du commit n'est pas encore terminé. Ce préfixe ne doit pas se retrouver dans les commit de la branche master. Il n'y a pas de work in progress dans master. Pour cela on squash les commit lors du merge.
[FIX] indique un commit qui résout un bogue.
[IMP] indique que le commit améliore une feature au projet.
[UNKNOWN NODE problematic][ADD]* indique que le commit ajoute une nouvelle feature au projet
[REM] indique que le commit enlève une partie du code projet
Il ne faut pas oublier que pousser son code est aussi le moyen de sauvegarder son travail et de pouvoir revenir dans un état précédent. Il ne faut pas hésiter à faire plusieurs commit dans la même journée sur la même branche si vous pensez que les développements à venir pourraient briser ce que vous avez déjà fait. Cela facilitera également la saisie des descriptions dans les feuilles de temps.
Avant de résoudre l'état WIP de votre merge request, avez-vous fait les vérifications suivantes :

- Rebasé votre branche par rapport à la branche **master** du dépôt client
- Tests fonctionnels (Avec le bon utilisateur, vérifier les règles de visibilité …)
- Lint Check (via la commande `make lint_check`)
- Traductions
- Droits d'accès, obligatoires lors de la création d'un nouvel objet

- Tests unitaires

> Warning
>
> IL NE FAUT JAMAIS EFFECTUER VOS TESTS AVEC L'UTILISATEUR **ADMIN** !
> CET UTILISATEUR BYPASSE LES REGLES DE VISIBILITE, ET INTRODUIT GÉNÉRALEMENT DE GROS BUGS EN PRODUCTION !

5 Merges-Requests

Lorsque vous avez terminé votre développement, il faut rendre tout beau votre MR pour la présenter à la personne qui va faire le Code Review. C'est comme faire un super bon repas et ne pas faire une belle présentation dans l'assiette.

5.1 On se met à jour

Première chose, nous allons nous mettre à jour par rapport à la branche master du dépôt client

```
git fetch subteno master
git rebase -p subteno/master
```

Zut, j'ai des conflits, il faut les corriger et ensuite

```
git add fichier-corrigé
git rebase --continue
```

Une fois le rebase terminé, nous poussons sur Gitlab

```
git push -f
```

> Tip
>
> L'option -f permet de forcer car nous avons rebasé, Git va rejeter car il y un incoréhence avec les commits du serveur distant.

5.2 On fait du ménage dans nos commits

On s'est fait plaisir dans les commits car nous avons des tests, annuler des commits et j'en passe. C'est tout à fait normal lorsque nous sommes en développement. Mais là, il faut que l'on livre avec un seul commit bien détaillé de ce que l'on a fait.

```
git fetch subteno master
git rebase -i subteno/master
```

> Tip
>
> L'option "-i" permet de lancer git rebase en mode interactive

Une fois arrivé dans le mode interactive de Git, remplacer pick par squash sauf le premier commit
Git va vous demander de rédiger la description du commit, une fois fait, enregistrez et quittez
Il faut reste à pousser sur le dépôt

```
git push -f
```

5.3 On retire le WIP

Maintenant que vous êtes à jour par rapport à master et que vous avez refait vos commits, vous pouvez retirer le WIP et créer une activité Code Review dans Odoo pour le Code Reviewer.