Le protocole Git comme outil d'édition et d'écriture : entretien avec Antoine Fauchié

Lescouet Emmanuelle

LE.CARNET.DE¬
LA FABRIQUE DU NUMÉRIQUE
Édition, innovation et actualités du livre

Publié initialement sur le Carnet de la Fabrique du numérique le **28 mars 2022**

https://carnet.fabriquedunumerique.org/git-outil-antoine-fauchie/

Le protocole Git comme outil d'édition et d'écriture : entretien avec Antoine Fauchié

par **Emmanuelle Lescouet**

Antoine Fauchié décrit son utilisation de Git. Git, ce n'est pas un outil, c'est plutôt un protocole de travail qui permet de gérer des documents dans l'espace et dans le temps, d'en gérer les versions.

Antoine Fauchié est doctorant à l'Université de Montréal et travaille sur la question des processus d'édition. Son projet de recherche s'intitule provisoirement « Reconfiguration des processus de publication et évolution des pratiques d'écriture et d'édition dans le champ littéraire ».

Antoine Fauchié a accompagné les professionnels du livre de la région Rhône-Alpes autour des questions numériques pendant près de 10 ans, il a enseigné à l'IUT2 de Grenoble et il a développé une activité d'indépendant pour le développement d'éditions numériques.

Qu'est-ce que c'est que GIT ?

Git, ce n'est pas un outil, c'est plutôt un protocole de travail. Git, c'est une façon de travailler, de gérer des documents dans l'espace et dans le temps. C'est un peu pompeux comme définition.

Git répond au problème : « je travaille sur un projet avec plusieurs fichiers informatiques (n'importe lesquels) et je cherche à éviter d'avoir des titres à rallonge comme fichier_reluparXX_v1, v2, etc. »

On va vouloir ajouter des choses, donc on sauvegarde de nouvelles versions. Au bout d'un moment, c'est ingérable ! Tu te retrouves avec plein de fichiers différents sur ton ordinateur, avec plein d'informations, mais à différents états, à différents moments de travail. Git est arrivé assez tardivement dans l'histoire des systèmes de gestion de versions (SVM). Il y en a plusieurs, ce sont des logiciels qui facilitent la tâche pour naviguer dans le temps et tracer les modifications faites seul.e ou à plusieurs sur un fichier.

Il y a une question politique liée à Git, dans le fait qu'au début des années 2000, il y a eu plusieurs systèmes de gestion de versions qui cohabitaient et maintenant tout le monde utilise Git. Les autres systèmes ont été éclipsés. Git s'est fait connaître par la plateforme GitHub qui était d'abord un projet indépendant, façon *start-up*, qui a été rachetée il y a quelques années par Microsoft. Ça a eu une influence !

Microsoft est un des plus gros contributeurs au logiciel libre aujourd'hui, même si ça paraît contradictoire. GitHub reste la plateforme la plus utilisée pour le partage de code et Git est au cœur de cette plateforme. À citer, on a également GitLab, qui est une plateforme que l'on peut utiliser en ligne simplement en se créant un compte, comme GitHub. Mais si l'on veut des fonctions plus avancées, il faut payer (comme avec GitHub). On peut installer le logiciel GitLab sur son propre serveur et on peut l'utiliser, c'est aussi en partie un logiciel libre contrairement à GitHub qui est propriétaire et qu'on ne peut pas installer sur son propre serveur.

Quand on fait le choix d'utiliser Git, il reste à choisir la plateforme qu'on va utiliser et c'est un choix politique. Est-ce qu'on utilise du logiciel libre ou propriétaire ? Est-ce qu'on utilise une plateforme portée par un acteur précis ?

Ça pose la question de la gestion des données, aussi. Pour donner un exemple, GitHub propose un outil qui s'appelle Copilote qui permet d'aller chercher dans tout le code hébergé sur GitHub et de proposer automatiquement la suite du code en fonction de ce qui est souvent utilisé. Ce n'est pas directement lié à Git, mais ça s'intègre dans la pratique commune.

Cet outil peut aider à écrire du code plus vite et à corriger des problèmes, voire à écrire mieux du code (ou moins bien aux vues des premiers tests). Ça pose plein de questions : ça utilise du code libre, mais qui n'est pas forcément réutilisé dans une optique libre, le tout sur une plateforme commerciale. C'est bien de garder à l'esprit que si Git est un protocole libre, il y a son implémentation et là, il y a plein de choses différentes qui existent.

Cette séparation, on l'a vue dans des entretiens précédents entre SPIP et Drupal par exemple, avec d'un côté une proposition libre parfois plus instable et une proposition commerciale qui pose des questions politiques… Est-ce aussi le cas pour Git ?

Il y a clairement ça : aujourd'hui, si on gère du code avec GitHub, on a des fonctions extrêmement avancées qui simplifient plein de processus. Soit on utilise le logiciel, soit on est directement sur l'interface en ligne et ça peut devenir très fluide; on peut faire des choses très puissantes. On le peut avec GitLab, mais dans une moindre mesure, puisque le travail sur l'interface est moins abouti. C'est une façon différente de travailler. Sur d'autres plateformes libres, comme Gitea, on a vraiment moins de fonctionnalités, on va être dans un usage plus basique de GIT. Il y a beaucoup de choses autour de Git, l'avantage c'est qu'on a le même cœur. La différence par rapport aux CMS que tu citais, c'est qu'eux sont des systèmes vraiment différents, avec des avantages et des inconvénients particuliers et divers. Ici, on a toujours Git au centre. Si je travaille sur un projet sur GitHub, je sais que je peux avoir un usage basique de GIT ou les fonctions avancées de GitHub, mais j'ai le choix.

Gitea va être beaucoup plus léger, donc pour des besoins basiques, ce sera le plus simple à installer sur un serveur. Ça pose la question de la propriété, mais aussi de la gestion des serveurs et des pannes possibles.

Tu l'utilises principalement pour des tâches d'édition et de travail collaboratif. Comment l'inclus-tu dans un processus de travail ?

Ce qui est compliqué, avec Git, c'est que c'est très puissant et donc parfois assez complexe. Je ne comprends pas tout son fonctionnement, par exemple. C'est plutôt bien fait pour gérer du texte, en général, et du code, particulièrement. Mais ça peut être un peu ardu pour d'autres utilisations. Mais tu n'es pas obligé de l'utiliser directement, comme on le fait traditionnellement. C'est-à-dire soit avec un petit logiciel qui permet de dire « c'est une nouvelle version », je lui donne un nom puis je mets un message associé à cette version; ou avec un terminal, il y a des commandes assez complexes à utiliser. Mais tu peux utiliser des interfaces qui vont venir se connecter à Git. Il peut devenir une façon de travailler à laquelle tu connectes des outils qui existent déjà. C'est assez pratique ! Notamment pour des interfaces web à la façon de WordPress, sans que tu aies tous les détails dont tu n'as pas besoin en tant qu'éditeur.ice.

Que conseillerais-tu comme outil pour manipuler Git, alors ?

Il n'y a pas d'outils que je vais vraiment conseiller, il faudrait beaucoup de recul !
Ce n'est pas une solution pratique, mais une méthode. Il y a de nombreux outils en fonction de ce que tu veux faire.

Si tu veux gérer un site web, par exemple, il y a plein d'interfaces, qu'on appelle des Headless CMS. J'utilise souvent Forestry.io qui est très basique et très simple. C'est une interface pour éditer du texte qui s'occupe des enregistrements, *commits*, du texte, sans que tu aies besoin de taper des commandes ni rien. Il y a des logiciels qui te permettent de gérer des logiciels sur ton PC, GitHub en propose un qui a une belle interface et qui fonctionne très bien; pour d'autres plateformes, GitKraken fonctionne très bien. Il permet de bien comprendre où l'on se situe dans l'arbre des versions de Git, où tu peux avoir plein de branches parallèles. Il permet de naviguer entre ces branches et de visualiser facilement où l'on se situe.

Pour revenir aux gestes éditoriaux, qui sont au cœur de ce que tu en fais, comment articules-tu Git dans une équipe ? Intervient-il dès le départ ? Comment ça marche ?

L'idéal, si c'est utilisé au sein d'une équipe, c'est de le mettre en place dès que les fichiers arrivent et que le travail commence. Ça évite les problèmes de nomenclature dont on parlait, et le risque de ne pas tous.tes travailler sur la bonne version. Git va venir structurer la façon de travailler, ça va la contraindre aussi, forcément. Ce n'est pas une méthode totalement ouverte, elle va avoir des impacts sur la façon de travailler.

Ce qui est intéressant, c'est que tu vas pouvoir tracer toutes les modifications en fonction des rôles des personnes qui composent l'équipe; tout le monde peut travailler de façon parallèle, plusieurs personnes peuvent travailler sur le même fichier en même temps et ensuite fusionner les différentes contributions. Chacun.e travaille sur un fichier sur son ordinateur et

versionne son travail, sans forcément être connecté.e à Internet. Une connexion est seulement nécessaire pour envoyer les modifications sur un dépôt commun. Il peut y avoir des conflits, s'il y a eu des modifications aux mêmes endroits.

Tu peux avoir des systèmes de branches, c'est ce qui est intéressant ! Par exemple, tu peux travailler sur la relecture du texte sur une branche consacrée, quelqu'un d'autre va commencer la mise en forme sur une autre branche. On voit assez vite que Git permet de gérer des fichiers textes bruts, du balisage comme du XML, du HTML ou des langages plus légers comme le Markdown. Quand on travaille avec Git, on sépare la structure et la mise en forme, et dès qu'on l'utilise, on est contraint de séparer tous les fichiers selon un fonctionnement à peu près logique où toute la structure va être gérée dans un fichier balisé et la mise en forme sera gérée ailleurs. Ce n'est pas aussi vrai en pratique : on va intervenir sur le balisage pour la mise en forme, il va peut-être nous manquer des éléments et peut-être que quelqu'un qui va relire le texte va intervenir sur la mise en forme à la marge.

Ce qui est génial, c'est que tu peux travailler en même temps sur le même texte sans avoir à séparer les étapes. Elles n'ont pas à être imperméables, surtout entre travail sur le texte et sa mise en forme. C'est souvent compliqué de faire les deux en même temps. Si tu sépares ces étapes, il n'y a pas de discussion qui s'opère entre les différentes faces. L'équipe peut travailler en synergie dès le début du projet. Git te force à avoir ce fonctionnement-là.

Pourrais-tu nous parler de ta rencontre avec cet outil ? Comment as-tu commencé à le mettre en place ? As-tu essayé des méthodologies avant de le prendre en main ?

J'utilise Git depuis 4 ou 5 ans. D'abord, c'était pour des projets personnels. Ça me permettait de structurer mon travail, d'avoir une certaine discipline, de ne pas avoir plein de fichiers sur mon ordinateur dont je ne sais plus à quoi ils correspondent. Je l'ai utilisé seul, par exemple pour fabriquer des objets numériques, principalement des sites web.

En équipe interdisciplinaire, je l'utilise pour du travail d'édition ou de design, cette fois encore, sur des objets numériques en majorité. Je me suis d'abord inséré dans ces projets, je n'y imposais pas Git : c'était déjà mis en place. Maintenant, la plupart des projets textuels sur lesquels je travaille, j'essaie de faire utiliser Git avec la contrainte que si les personnes n'ont pas l'habitude de l'utiliser, il y a une marche, une organisation à prendre en main. Chacun.e a ses habitudes…

Ce n'est pas juste le protocole à utiliser sur un ordinateur pour versionner, il y a des plateformes qui permettent d'héberger des projets gérés avec Git et qui proposent d'autres fonctionnalités intéressantes, mais pas propres à GIT, notamment la gestion de tickets (*issues*), qui permettent de signaler des éléments et d'assigner des choses à faire à des membres de

l'équipe. On peut avoir du déploiement continu sur des plateformes qui hébergent des projets GIT. Chaque fois qu'on versionne le projet, on peut prévoir un certain nombre de tâches et les déclencher automatiquement. Si une équipe travaille sur un projet d'édition et veut travailler sur une nouvelle mise en forme, on peut travailler sur une branche spécifique. Chaque fois qu'on va y enregistrer des modifications, ça va mettre à jour un site web qui montre le projet, par exemple.

Tu parles de projets numériques, mais je sais que tu utilises Git pour des projets d'édition imprimée ou possiblement imprimée. Comment l'insères-tu dans cette chaîne de travail ?

OUI ! Il y a des choses qu'on ne peut pas faire avec Git. Notamment, on peut très difficilement gérer des fichiers binaires. À partir du moment où l'on travaille dans Git, on ne va pas travailler avec des fichiers .doc, .docx, .odt, etc. C'est possible, mais c'est très compliqué et ça n'a pas beaucoup de sens. On versionnera, mais on ne verra pas les différences entre les versions. Ce n'est plus que de la sauvegarde.

On ne peut pas non plus gérer des projets utilisant des logiciels de publication assistée par ordinateur (PAO) comme InDesign. C'est limité à des fichiers en texte brut.

L'idée du *Single Source Publishing* est de pouvoir produire plein de formes différentes à partir d'une même source. Git permet de le faire, parce qu'on versionne la source, et derrière, on peut déclencher des actions pour produire des versions différentes de cette source. On n'a pas un seul fichier, mais plusieurs. On peut avoir une seule et même source, mais plusieurs fichiers de configuration qui vont définir des détails propres à chaque artefact produit à la fin.

Pour la version imprimée, on peut avoir des informations comme le papier, les marques de coupes, le profil colorimétrique utilisé, des choses comme ça. Alors que pour la version site web, on va juste préciser l'URL, la feuille de style associée et ainsi de suite. C'est surtout ça, l'avantage. Git versionne tout au même endroit, donc on va avoir la source et les fichiers de configurations au même endroit, on peut donc automatiser un certain nombre d'actions pour concentrer le travail sur le contenu et le laisser se déployer sur les fichiers de mise en forme autour.

Aurais-tu des exemples de projets que tu as réalisés avec Git ? Des exemples concrets de ce que ça a pu permettre de faire ?

Je peux parler d'un catalogue de musée où l'on a suivi ce protocole-là du début à la fin. C'est le catalogue de la Villa Chiragan, un catalogue édité par le Musée Saint-Raymond à Toulouse. C'était un projet d'édition porté par une bibliothécaire et un conservateur. Ils ont conduit le

projet. Julie Blanc et moi avons proposé un certain nombre d'outils pour mettre en place du *Single Source Publishing*. Ce qui a été très pratique, c'est qu'eux utilisaient une plateforme en ligne, Forestry, où ils ont édité les contenus au fur et à mesure, associé à Zotero pour la gestion des références bibliographiques.

C'est un catalogue de musée plutôt érudit, il y a un gros appareil critique. Eux n'ont pas directement interagi avec Git, ce qui leur a évité de devoir apprendre à l'utiliser. On avait tout placé sur GitLab. Comme exemple d'utilisation, ils ont pu gérer des tickets pour signaler des problèmes, ce qui est assez pratique. Ils ont interagi avec des fichiers Markdown, sans les voir directement, via l'interface de Forestry.

Avec Julie, on a fait tout le travail de structuration des informations et de design pour les versions numériques et imprimées. On a quelque peu contourné le *Single Source Publishing* pour ce projet : on a retouché un peu au contenu pour la mise en forme imprimée, spécifiquement parce qu'on n'avait pas les bons outils à l'époque et parce que ça impliquait de rajouter beaucoup de balisage dans le texte que les éditeur.ices allaient voir et l'on ne voulait pas les surcharger…

Aujourd'hui, on pourrait faire du vrai *Single Source Publishing* ! Mais on a fait au plus simple, ce qui veut dire qu'à chaque fois, il fallait une petite passe supplémentaire pour vérifier que les pages étaient bien composées, qu'on n'ait pas de blanc, des crevasses dans le texte. L'intérêt est aussi qu'on a pu faire un peu de déploiement continu pour tester l'application sur plein d'adresses différentes pour voir des versions de démonstrations et les valider. C'était vraiment pratique !

Tu parles de difficultés potentielles de prise en main. Comment le vis-tu dans ton travail au quotidien ?

Plus récemment, j'ai travaillé avec Louis-Olivier Brassard sur la partie design et code, moi sur la structuration de la chaîne. On travaillait avec deux éditeur.ices, Jean-François Vallée et Chantale Ringuet, sur le Novendécaméron. Sur ce projet-là, on avait deux profils différents. Deux profils non techniques, à la base, mais un profil qui avait déjà expérimenté des plateformes, du XML, etc., qu'on aurait pu former à Git, même si ça aurait demandé du temps, et quelqu'un qui était plus habitué à des interfaces de traitement de texte classique comme Word. Là, le saut était trop important, trop pénible pour la personne et une trop grande perte de temps pour le projet. Donc, on a mis en place Forestry, encore une fois parce que c'est vraiment une interface simple. Cette fois, c'est vraiment du *Single Source Publishing* : eux vont éditer les contenus. On leur a expliqué les principes, mais on n'aurait pas pris le temps d'expliquer comment fonctionne Git.

Je me rends compte que dans 95 % des cas, ça fonctionne bien sur un usage simple, c'est-à-

dire enregistrer les versions, envoyer des modifications et éventuellement basculer d'une branche à l'autre. Ce sont les choses les plus simples à gérer. Souvent, ces personnes-là vont changer un peu leurs habitudes de travail en se rendant compte que ça leur apporte quelque chose qu'elles n'avaient pas forcément avant.

Travailler avec des gens qui connaissent déjà Git, ce n'est pas si automatique que ça : on n'a pas tous et toutes les mêmes fonctionnements. C'est rigolo d'arriver sur un projet où il y a des gens qui ont les mains dans la technique, qui utilisent Git, mais pas de la même façon. Par exemple, quand on travaille sur une plateforme comme GitHub ou GitLab, si l'on veut développer une nouvelle fonctionnalité, on va créer un ticket (des *issues* dans les interfaces en question) et on décrit ce qu'on veut faire. On va ensuite créer une branche qui va avoir le nom de ce ticket. Ça nous permet de retrouver assez vite quelle branche correspond à quelle problématique : on sait que les deux sont liées. Par contre, c'est n'est pas une façon de travailler que tout le monde a. Si l'on arrive sur un projet, on va découvrir de nouvelles façons de faire.

Est-ce propre à Git ou est-ce un reflet des façons de travailler de chacun.e ?

Le seul problème, c'est qu'on est dans le monde du développement informatique et il y a souvent une tendance : des personnes qui travaillent depuis longtemps vont penser que leur façon de faire est la bonne et, souvent, ça crée des frictions humaines. Ce n'est pas toujours très fluide au début, il faut réussir à jongler. L'avantage, c'est que je ne suis pas développeur, je peux donc soit facilement me plier à des contraintes, si je vois qu'elles sont pertinentes, soit aussi parce que je vais apporter un regard un peu extérieur. Je peux donc apporter d'autres pratiques, venues plutôt de la gestion du texte. C'est toujours intéressant.

On connaît Git comme un lieu de développement, que ce soit de modules ou de programmes entiers. L'usage en est-il très différent dans cette optique-là et dans une optique d'édition ?

Pour la petite histoire, Git a été créé par Linus Torvalds, le développeur créateur du noyau Linux. Il l'a créé parce que le logiciel qu'il utilisait jusqu'ici pour versionner Linux était propriétaire, et ça ne lui allait plus. Donc il a décidé de développer lui-même son système de gestion de versions, qui est aujourd'hui le plus utilisé. Il a été pensé pour gérer du code, ce qui fait qu'on en voit les limites quand on manipule du texte.

Quand on code, on obtient des fichiers avec beaucoup de lignes, mais peu de caractères par lignes, une soixantaine en général, avec des systèmes d'indentation. Il y a assez peu d'informations sur une ligne. Git gère des lignes, il voit la différence entre une ligne et une autre. Si l'on gère du texte, sur une même ligne, on peut avoir 500 caractères. Ça devient

compliqué de suivre les différences entre deux lignes.

Il faut adapter les pratiques. Soit on fait des sauts de lignes à la fin de chaque phrase. Ça fonctionne très bien avec du HTML, du Markdown ou du XML, parce que le saut de ligne ne sera pas interprété dans la suite, c'est uniquement visuel dans l'éditeur de texte. L'autre solution, et c'est le rêve de certaines personnes comme Marcello Vitali-Rosati, ce serait de développer un système de gestion de versions pour le texte. Plutôt que de tracer des lignes, on tracerait des chaînes de caractères, voire des caractères. Ce qui implique derrière un fonctionnement plus complexe et assez différent, mais ça pourrait être une solution.

Je trouve intéressant de détourner un outil qui a été pensé pour gérer du code et de l'utiliser pour gérer une autre forme de texte, du texte littéraire, par exemple. De ne pas créer quelque chose de zéro pour ça, et de bénéficier de toutes les pratiques et de tout l'environnement pensé pour le développement, mais qui peut être déplacé dans la gestion du texte littéraire.

En parlant de texte littéraire, est-ce que c'est un outil qui peut servir à la création, à l'écriture plus qu'à l'édition ? Louis-Olivier Brassard, dont on parlait tout à l'heure, a fait une pièce de théâtre sur Git. De ton côté, tu écris ta thèse avec, je crois ?

Au même moment que Louis-Olivier, j'avais expérimenté la création sur Git, même si c'était moins poussé. J'essayais d'écrire un texte composé des *commits* (l'action d'enregistrer un fichier), tout en produisant autre chose. C'est un peu ce qu'a fait aussi Louis-Olivier.

Ce qui est intéressant dans Git, c'est qu'on écrit en dehors de tous les fichiers qui sont gérés. On passe notre temps à écrire. Chaque fois qu'on enregistre une version, on lui associe un message, le message du *commit*. Chaque fois qu'on en fait un, on va modifier seulement quelques fichiers, mais c'est tout le projet qui va être versionné. À chaque *commit*, on a tout le projet qui est là, dans cette version-là. Quand on va en faire un nouveau, on aura tout le projet à nouveau.

Quand on se balade de *commit* en *commit*, on retrouve non pas le fichier qu'on a modifié, mais tous les fichiers qui composent le projet. Ces messages, ce sont de l'écriture, typiquement. On va indiquer ce qu'on a fait pour pouvoir retracer facilement et retrouver les différentes actions : on a développé telle chose, corrigé telle autre, etc. Quand on gère un projet éditorial, on va faire à peu près la même chose. Par exemple, j'ai corrigé les accords, remplacé les « on » par des « vous », etc. Et puis, on peut mettre aussi des messages qui vont être autre chose, c'est ça qu'avait fait Louis-Olivier.

J'écris ma thèse en écrivant chaque message de *commit*, comme une phrase qui a un sens. Ça

ne signale pas du tout ce que j'ai fait sur les fichiers… Ça peut être en rapport, mais pas forcément. C'est une phrase en lien avec mon sujet de thèse. Je le partage grâce à un *bot* sur Twitter. J'ai la contrainte de faire des messages assez courts, moins de 80 caractères espaces comprises en général, mais je peux aller jusqu'à 280 caractères.

Cette longueur fait partie des bonnes pratiques, aussi. Normalement, un message de *commit* ne fait pas plus de 80 caractères espaces comprises, et ce, pour plein de raison. Notamment, parce qu'en le faisant circuler sur les plateformes, on perd en lisibilité avec des messages longs. Il y a d'autres initiatives, comme la gitérature, Abrüpt en fait partie, dont je suis assez curieux : le but est de faire de la littérature avec Git. Abrüpt (NDRL: qui a fait l'objet d'un article sur le Carnet) sont ceux qui ont été le plus loin. Ils ont créé un système assez complexe avec des branches, des *commits*, etc. On peut naviguer dans le projet littéraire, il faut une compréhension fine du fonctionnement de Git et de la façon dont on peut articuler à la fois des branches, des *commits*, des tags, etc.

Pourrais-tu nous présenter un peu plus Abrüpt ?

C'est une petite maison d'édition basée en Suisse qui fait de la littérature contemporaine, de la réédition de textes classiques et de pamphlets, des essais… Elle publie en numérique et en imprimé. Sa particularité, c'est que tous les projets sont gérés avec Git. On peut les retrouver en ligne, ils sont versionnés. Mais ça, ce n'est peut-être pas le plus intéressant. C'est plutôt le fait qu'ils vont essayer de repenser la façon dont ils utilisent la technologie pour produire des publications très diverses. Ils vont essayer d'avoir une démarche *low-tech* — même s'ils ne s'en revendiquent pas du tout. Ils vont utiliser des programmes assez basiques, assez simples à utiliser plutôt que des logiciels qui créent beaucoup de dépendances. Ils utilisent beaucoup Pandoc, LaTex, Bash, Make, etc. Cela leur permet de produire assez facilement, avec un souci typographique assez avancé.

Ils produisent leurs livres imprimés à la demande, c'est-à-dire que les livres peuvent être distribués en librairie, mais ils sont juste disponibles : si l'on veut les avoir, on les commande et ils seront livrés rapidement.

C'est un collectif qui travaille et édite à visage caché : on ne sait rien des personnes qui composent le collectif. Abrüpt est une structure composée d'humain.es, mais sans qu'on sache qui. Ils ne prennent pas la parole. Ils participent à des événements, mais sans être en présence.

Ça a l'air d'être assez marginal comme usage de Git, dans le sens où dans le paysage littéraire, même des littératures numériques, ça paraît assez expérimental, assez restreint. Est-ce que c'est la vocation de l'outil ?

C’est sûr que c’est très expérimental ! Même dans des projets éditoriaux, en général, en édition savante ou généraliste, c’est très peu utilisé. C’est assez dur de trouver des projets qui sont gérés avec Git, et puis tout mouvement d’utiliser Git pour la création est encore plus marginal.

Le problème, c’est vraiment le saut technique. Si l’on veut vraiment expérimenter avec Git, on est obligé de passer par un terminal. On peut utiliser une interface, mais à un moment, elle va nous contraindre trop fortement, donc il faudra passer par un terminal.

Ce saut est énorme pour quelqu’un qui n’a pas l’habitude. Je pense donc que l’usage de Git va rester expérimental. Après, on peut noter que ça peut avoir des influences, de la même façon que Kenneth Goldsmith et son uncreative writing. En fait, quelque part, les pratiques que lui peut avoir avec des étudiant.es ne vont pas se répandre énormément, mais elles vont avoir une influence sur la façon dont on va écrire et dont on va penser les projets littéraires et éditoriaux. Le mouvement va plus dans ce sens, à mon avis : montrer comment on peut faire les choses différemment, plutôt que de réellement dire qu’on abandonne les anciennes pratiques pour passer sur Git.

C’est vraiment intéressant et c’est quelque chose que j’observe beaucoup du côté du développement web, il y a vraiment des pratiques émergentes pour créer des interfaces intermédiaires pour utiliser Git sans l’utiliser. Ça, ça va vraiment arriver.

C’est possible que Git soit utilisé dans toutes les chaînes éditoriales existantes, même celles qu’on connaît avec InDesign et Word, sans qu’on s’en rende compte directement, mais avec la puissance du versionnement qui sera derrière. Un détail là-dessus : tout ce qui est création d’illustrations vectorielles, avec Illustrator ou d’autres, c’est en train de basculer sur l’utilisation de Git. Un dessin vectorisé, ce n’est que des lignes de codes qui peuvent être versionnées facilement, contrairement à un fichier binaire comme un fichier image. Ça pourra arriver jusqu’à l’utilisation de Word ou d’InDesign.

Au-delà de cet usage presque invisible, qu’est-ce que tu espèrerais ? Comment aimerais-tu utiliser Git ?

Pour un projet de création, ça m’intéresse d’observer comment ça fonctionne, mais ça ne m’intéresserait pas de créer directement avec Git comme Abrüpt peut le faire. Ça m’intéresse vraiment de voir comment d’autres personnes ou structures peuvent s’en emparer.

Si j’avais du temps et de l’argent, ce serait justement de créer des interfaces intermédiaires permettant de gérer du contenu tout en pouvant utiliser Git, mais de façon approfondie. De pouvoir gérer des systèmes de branches et d’étiquettes (de tags), de gestions de conflits, plein de choses comme ça. Ça demande beaucoup de temps, et même si des briques technologiques existent, il faudrait des mois de travail pour les assembler avec des

développeur.euses chevronné.es. Ça m'intéresserait d'avancer là-dessus pour pouvoir gérer des projets littéraires de création directement sur Git. Ou pour des projets plus compliqués de publications scientifiques, où il y a plus de choses à gérer avec l'appareil critique, les notes, les bibliographies, etc.

Pourrais-tu nous parler de la gestion de conflits et des tags que tu évoques ?

Les tags ou étiquettes, c'est pour permettre de naviguer et de repérer plus facilement les *commits*. Ça peut être n'importe quoi, plus souvent un mot. On les utilise pour gérer des versions (*releases*), des mises à jour d'un programme. Ça va être v1, v2, etc., ou ça peut être *brouillon*, *rédaction*, etc. On pourra retrouver les *commits* plus facilement qu'avec les codes chiffrés des *commits* par défaut qui comportent trop de caractères pour être retenus par des humain.es.

La branche, c'est un peu le même fonctionnement : c'est une sorte de tag qu'on déplace à chaque fois qu'on versionne.

La gestion de conflits, c'est juste que comme on peut modifier en même temps, si deux personnes ne modifient pas les mêmes lignes, il n'y a pas de conflits, les deux lignes sont modifiées dans la version fusionnée. Mais si les deux personnes ont modifié la même ligne, il y a un conflit : Git va nous demander de choisir entre les deux versions. Ça peut être compliqué, parce que si l'on n'a pas une interface adaptée, c'est pénible à faire avec un terminal. Il y a des logiciels qui le font très bien, comme GitHub ou GitKraken : ils vont montrer la différence entre les deux fichiers et l'on n'aura qu'à cliquer sur un bouton pour choisir la bonne version, on enregistre et tout va bien. D'autres logiciels, comme des éditeurs de texte, vont permettre d'afficher ça, et nous mettre des couleurs différentes pour montrer les différentes versions.

En sachant qu'au moment du conflit, on n'est pas obligé de choisir l'une ou l'autre des versions, on peut en modifier une avec les informations des deux, ce qui crée une troisième version. C'est souvent ce que je fais : c'est juste qu'on a modifié deux choses différentes au même endroit. Comme Git ne permet de tracer que les lignes et pas les caractères, on a des conflits.

Cela semble être une des fonctions les plus compliquées et à la fois les plus pratiques pour l'avancée éditoriale ? Ça a l'air de pouvoir éviter les allers-retours incessants entre les différents membres de l'équipe.

On peut vraiment aller loin. Je ne suis pas le plus innovant dans les usages de révisions, etc. Je reste très axé sur les commentaires et les modifications, chacun son tour. Tout se passe alors

directement dans le fichier.

Ce n'est pas la façon la plus efficiente de travailler. Par contre, ce qu'on peut faire, c'est utiliser des outils associés à GIT disponibles sur des plateformes comme GitLab ou GitHub, ou d'autres, et commenter des lignes. On utilise d'autres options, notamment la proposition de fusion (*merge request*). On va faire une modification et plutôt que de fusionner avec les autres modifications, on va la proposer. On va dire : « voilà ce que je vous propose », et quelqu'un va passer dessus pour valider ou invalider. C'est à ce niveau-là qu'on peut commenter, on aura une première phase de relecture avec des modifications, on crée une branche parallèle au projet, et quand on a fini, on fait une proposition de fusion : on ne fusionne pas directement. À ce niveau-là, on va pouvoir commenter chaque modification.
La personne pourra modifier le fichier en fonction des commentaires pour faire une nouvelle proposition.

Ces systèmes mettent en place beaucoup de rapport de pouvoir, ça se passe bien en général ou c'est un peu délicat ?

Il n'y a pas nécessairement de tensions. L'avantage qu'on a, c'est que quand on utilise Git, il n'y a pas de différenciation des rôles : tout le monde peut faire un *commit*, créer une branche, etc. Après, sur la plateforme, on peut distinguer des rôles : certain.es vont avoir le droit de proposer, mais pas de fusionner, tandis que d'autres vont avoir ce privilège. Je ne pense pas qu'à ce niveau-là, Git crée de tensions. C'est vraiment la façon dont on va l'utiliser qui va jouer, ça implique de bien formuler les différents rôles.

Par exemple, à la Chaire de recherche du Canada sur les Écritures Numériques on utilise beaucoup Git pour gérer les projets, on gère les fichiers textes comme ça… Et là, par contre, on ne gère pas très bien les rôles, tout le monde fait un peu tout, donc forcément, ça frictionne. Par contre, on peut assez vite définir ça.

Par exemple, sur le Novendécaméron, les éditeur.ices pouvaient éditer les contenus, mais pas fusionner sur la version finale publique du projet, parce qu'il y avait un certain nombre de détails techniques qu'on préférait valider. On avait donc le dernier mot pour appuyer sur le bouton de mise en ligne. On vérifiait que l'interface fonctionnait bien. Si l'on voyait que tout s'affichait bien, on validait, mais si un truc nous semblait bizarre, on leur renvoyait en leur indiquant ce qui pouvait coincer.

C'est intéressant sur la répartition des rôles, pas que ça crée de la friction, mais au contraire, c'est sécurisant pour des personnes qui ne se sentent pas à l'aise, qui ont peur de casser le projet. Ou au contraire, de dire « j'ai fait ce que j'avais à faire, c'est à une autre personne de valider ».

Une organisation pyramidale n'est pas nécessaire, ça peut être plus horizontal. C'est souvent le cas sur des projets techniques où tout le monde a la main sur le projet, mais il y a une bonne circulation d'information et l'on sait quand valider ou pas.

Sur les formes littéraires que ça engendre potentiellement, beaucoup d'outils vont pousser une forme particulière : par exemple Twitter, pour des choses très fragmentaires plutôt que des romans (même s'il y en a, mais qui négocient leur forme). Y a-t-il un équivalent pour Git ?

L'écriture par *commit* va amener des systèmes fragmentaires, parce qu'il y a la contrainte du nombre de caractères et après, on aura une composition de fragments.

Pour d'autres formes d'écriture, ça n'aura pas forcément trop d'impacts, mais puisqu'on utilise Git, on utilise des fichiers en texte brut balisés. L'influence va plutôt être de ce côté-là. Si l'on utilise du balisage pour structurer le contenu, derrière la mise en forme, ça va surtout être de la mise en forme web, CSS, avec des contraintes particulières. On peut avoir des choses très diverses, comme le web en général.

Quelques œuvres qui utilisent Git :

- *qtrnmnet* d'Antoine Fauchié relaie ses *commits* de thèse : https://twitter.com/qtrnmnet
- *Les petites cosmogonies* de Christine Jeanney, publié par Abrüpt et disponible en ligne est une oeuvre fragmentaire où la vie et la pensée sont découpées pour être interrogées.

Une liste plus complète et mise à jour sur le Répertoire des Écritures Numériques.