GIT SUR LE SERVEUR

Les différents outils pour le serveur

À présent, vous devriez être capable de réaliser la plupart des tâches quotidiennes impliquant Git. Néanmoins, pour pouvoir collaborer avec d'autres personnes au moyen de Git, vous allez devoir disposer d'un dépôt distant Git.

Bien que vous puissiez techniquement **tirer** et **pousser** des modifications depuis et vers des dépôts personnels, cette pratique est déconseillée parce qu'elle introduit très facilement une confusion avec votre travail actuel.

De plus, vous souhaitez que vos collaborateurs puissent accéder à votre dépôt de sources, y compris si vous n'êtes pas connecté— disposer d'un dépôt accessible en permanence peut s'avérer utile. De ce fait, la **méthode canonique** pour collaborer consiste à instancier un dépôt intermédiaire auquel tout le monde a accès, que ce soit pour pousser ou tirer.

Un serveur Git est simple à lancer. Premièrement, vous devez choisir quels protocoles seront supportés.

La première partie de ce chapitre traite des protocoles disponibles et de leurs avantages et inconvénients.

La partie suivante explique certaines configurations typiques de ces protocoles et comment les mettre en œuvre. Enfin, nous traiterons de quelques types d'hébergement, si vous souhaitez héberger votre code sur un serveur tiers, sans avoir à installer et maintenir un serveur par vous-même.

Si vous ne voyez pas d'intérêt à gérer votre propre serveur, vous pouvez sauter directement à la dernière partie de ce chapitre pour détailler les options pour mettre en place un compte hébergé, avant de continuer au chapitre suivant dans lequel les problématiques de développement distribué sont abordées.

Un dépôt distant est généralement un **dépôt nu** (**bare repository**) : un dépôt Git qui n'a pas de copie de travail. Comme ce dépôt n'est utilisé que comme centralisateur de collaboration, il n'y a aucune raison d'extraire un instantané sur le disque; seules les données Git sont nécessaires. Pour simplifier, un dépôt nu est le contenu du répertoire **.git** sans fioriture

Protocoles

Git peut utiliser quatre protocoles réseau majeurs pour transporter des données : local, HTTP, **Secure Shell** (SSH) et Git. Nous allons voir leur nature et dans quelles circonstances ils peuvent (ou ne peuvent pas) être utilisés.

Protocole local

Le protocole de base est le protocole **local** pour lequel le dépôt distant est un autre répertoire dans le système de fichiers. Il est souvent utilisé si tous les membres de l'équipe ont accès à un répertoire partagé via NFS par exemple ou dans le cas moins probable où tous les développeurs travaillent sur le même ordinateur. Ce dernier cas n'est pas optimum car tous les dépôts seraient hébergés de fait sur le même ordinateur, rendant ainsi toute défaillance catastrophique.

Si vous disposez d'un système de fichiers partagé, vous pouvez cloner, pousser et tirer avec un dépôt local. Pour cloner un dépôt ou pour l'utiliser comme dépôt distant d'un projet existant,

utilisez le chemin vers le dépôt comme URL. Par exemple, pour cloner un dépôt local, vous pouvez lancer ceci :

```
$ git clone /opt/git/project.git
```

Ou bien cela :

```
$ git clone file:///opt/git/project.git
```

Git opère légèrement différemment si vous spécifiez explicitement le protocole **file://** au début de l'URL. Si vous spécifiez simplement le chemin et si la destination se trouve sur le même système de fichiers, Git tente d'utiliser des liens physiques pour les fichiers communs. Si vous spécifiez le protocole **file://**, Git lance un processus d'accès à travers le réseau, ce qui est généralement moins efficace. La raison d'utiliser spécifiquement le préfixe **file://** est la volonté d'obtenir une copie propre du dépôt, sans aucune référence ou aucun objet supplémentaire qui pourraient résulter d'un import depuis un autre système de gestion de version ou d'une action similaire. Nous utiliserons les chemins normaux par la suite car c'est la méthode la plus efficace.
Pour ajouter un dépôt local à un projet Git existant, lancez ceci :

```
$ git remote add local_proj /opt/git/project.git
```

Ensuite, vous pouvez pousser vers et tirer depuis ce dépôt distant de la même manière que vous le feriez pour un dépôt accessible sur le réseau.

Avantages
Les avantages des dépôts accessibles sur le système de fichiers sont qu'ils sont simples et qu'ils utilisent les permissions du système de fichiers. Si vous avez déjà un montage partagé auquel toute votre équipe a accès, déployer un dépôt est extrêmement facile. Vous placez la copie du dépôt nu à un endroit accessible de tous et positionnez correctement les droits de lecture/écriture de la même manière que pour tout autre partage. Nous aborderons la méthode pour exporter une copie de dépôt nu à cette fin dans la section suivante Installation de Git sur un serveur.
C'est un choix satisfaisant pour partager rapidement le travail. Si vous et votre coéquipier travaillez sur le même projet et qu'il souhaite partager son travail, lancer une commande telle que **git pull /home/john/project** est certainement plus simple que de passer par un serveur intermédiaire.

Inconvénients
Les inconvénients de cette méthode sont qu'il est généralement plus difficile de rendre disponible un partage réseau depuis de nombreux endroits que de simplement gérer des accès réseau. Si vous souhaitez pousser depuis votre portable à la maison, vous devez monter le partage distant, ce qui peut s'avérer plus difficile et plus lent que d'y accéder directement via un protocole réseau.

Il faut aussi mentionner que ce n'est pas nécessairement l'option la plus rapide à l'utilisation si un partage réseau est utilisé. Un dépôt local n'est rapide que si l'accès aux fichiers est rapide. Un dépôt accessible sur un montage NFS est souvent plus lent qu'un dépôt accessible via SSH sur le même serveur qui ferait tourner Git avec un accès aux disques locaux.
Enfin, ce protocole ne protège pas le dépôt contre un dommage accidentel. Chaque utilisateur à un accès total au répertoire «distant» et il n'y a rien pour les empêcher de modifier ou supprimer des fichiers internes à Git et de corrompre le dépôt.

Protocoles sur HTTP
Git peut communiquer sur HTTP de deux manières. Avant Git 1.6.6, il n'existait qu'une seule manière qui était très simple et généralement en lecture seule. Depuis la version 1.6.6, il existe un nouveau protocole plus intelligent qui nécessite que Git puisse négocier les transferts de données de manière similaire à ce qu'il fait pour SSH. Ces dernières années, le nouveau protocole HTTP a gagné en popularité du fait qu'il est plus simple à utiliser et plus efficace dans ses communications. La nouvelle version est souvent appelée protocole HTTP «intelligent» et l'ancienne version protocole HTTP «idiot». Nous allons voir tout d'abord le protocole HTTP «intelligent».

HTTP Intelligent
Le protocole HTTP «intelligent» se comporte de manière très similaire aux protocoles SSH ou Git mais fonctionne par-dessus les ports HTTP/S et peut utiliser différents mécanismes d'authentification, ce qui le rend souvent plus facile pour l'utilisateur que SSH, puisque l'on peut utiliser des méthodes telles que l'authentification par utilisateur/mot de passe plutôt que de devoir gérer des clés SSH.
C'est devenu probablement le moyen le plus populaire d'utiliser Git, car il peut être utilisé pour du service anonyme, comme le protocole **git://** aussi bien que pour pousser avec authentification et chiffrement, comme le protocole SSH. Au lieu de devoir gérer différentes URL pour ces usages, vous pouvez maintenant utiliser une URL unique pour les deux. Si vous essayez de pousser et que le dépôt requiert une authentification (ce qui est normal), le serveur peut demander un nom d'utilisateur et un mot de passe. De même pour les accès en lecture.
En fait, pour les services tels que GitHub, l'URL que vous utilisez pour visualiser le dépôt sur le web (par exemple https://github.com/schacon/simplegit) est la même URL utilisable pour le cloner et, si vous en avez les droits, y pousser.

HTTP idiot
Si le serveur ne répond pas avec un service Git HTTP intelligent, le client Git essayera de se rabattre sur le protocole HTTP «idiot». Le protocole idiot consiste à servir le dépôt Git nu comme des fichiers normaux sur un serveur web. La beauté du protocole idiot réside dans sa simplicité de mise en place. Tout ce que vous avez à faire, c'est de copier les fichiers de votre dépôt nu sous la racine de documents HTTP et de positionner un crochet (**hook**) **post-update** spécifique, et c'est tout (voir Crochets Git). Dès ce moment, tous ceux qui

peuvent accéder au serveur web sur lequel vous avez déposé votre dépôt peuvent le cloner. Pour permettre un accès en lecture seule à votre dépôt via HTTP, faites quelque chose comme:

```
$ cd /var/www/htdocs/
$ git clone --bare /chemin/vers/projet_git projetgit.git
$ cd projetgit.git
$ mv hooks/post-update.sample hooks/post-update
$ chmod a+x hooks/post-update
```

Et voilà! Le crochet **post-update** livré par défaut avec Git lance la commande appropriée (**git update-server-info**) pour faire fonctionner correctement le clonage et la récupération HTTP. Cette commande est lancée quand vous poussez sur ce dépôt (peut-être sur SSH). Ensuite, les autres personnes peuvent cloner via quelque chose comme :

```
$ git clone https://exemple.com/projetgit.git
```

Dans ce cas particulier, nous utilisons le chemin **/var/www/htdocs** qui est le plus commun pour une configuration Apache, mais vous pouvez utiliser n'importe quel serveur web statique – placez juste les dépôts nus dans son chemin. Les données Git sont servies comme de simples fichiers statiques (voir Les tripes de Git pour la manière exacte dont elles sont servies).
Généralement, vous choisirez soit de lancer un serveur HTTP intelligent avec des droits en lecture/écriture ou de fournir simplement les fichiers en lecture seule par le protocole idiot. Il est rare de mélanger les deux types de protocoles.

Avantages
Nous nous concentrerons sur les avantages de la version intelligente du protocole sur HTTP.
La simplicité vient de l'utilisation d'une seule URL pour tous les types d'accès et de la demande d'authentification seulement en cas de besoin. Ces deux caractéristiques rendent les choses très faciles pour l'utilisateur final. La possibilité de s'authentifier avec un nom d'utilisateur et un mot de passe apporte un gros avantage par rapport à SSH puisque les utilisateurs n'ont plus à générer localement les clés SSH et à télécharger leur clé publique sur le serveur avant de pouvoir interagir avec lui. Pour les utilisateurs débutants ou pour des utilisateurs utilisant des systèmes où SSH est moins commun, c'est un avantage d'utilisabilité majeur. C'est aussi un protocole très rapide et efficace, similaire à SSH.
Vous pouvez aussi servir vos dépôts en lecture seule sur HTTPS, ce qui signifie que vous pouvez chiffrer les communications; ou vous pouvez pousser jusqu'à faire utiliser des certificats SSL à vos clients.
Un autre avantage est que HTTP/S sont des protocoles si souvent utilisés que les pare-feux d'entreprise sont souvent paramétrés pour les laisser passer.

Inconvénients
Configurer Git sur HTTP/S peut être un peu plus difficile que sur SSH sur certains serveurs. Mis à part cela, les autres protocoles ont peu d'avantages sur le protocole HTTP intelligent pour servir Git.

Si vous utilisez HTTP pour pousser de manière authentifiée, fournir vos information d'authentification est parfois plus compliqué qu'utiliser des clés sur SSH. Il existe cependant des outils de mise en cache d'informations d'authentification, comme Keychain sur OSX et Credential Manager sur Windows pour rendre cela indolore. Reportez-vous à Stockage des identifiants pour voir comment configurer la mise en cache des mots de passe HTTP sur votre système.

Protocole SSH

SSH est un protocole répandu de transport pour Git en auto-hébergement. Cela est dû au fait que l'accès SSH est déjà en place à de nombreux endroits et que si ce n'est pas le cas, cela reste très facile à faire. Cela est aussi dû au fait que SSH est un protocole authentifié; et comme il est très répandu, il est généralement facile à mettre en œuvre et à utiliser.
Pour cloner un dépôt Git à travers SSH, spécifiez le préfixe **ssh://** dans l'URL comme ceci:

```
$ git clone ssh://utilisateur@serveur/projet.git
```

Vous pouvez utiliser aussi la syntaxe scp habituelle avec le protocole SSH:

```
$ git clone utilisateur@serveur:projet.git
```

Vous pouvez aussi ne pas spécifier de nom d'utilisateur et Git utilisera par défaut le nom de login.

Avantages

Les avantages liés à l'utilisation de SSH sont nombreux. Premièrement, SSH est relativement simple à mettre en place, les **daemons** SSH sont facilement disponibles, les administrateurs réseau sont habitués à les gérer et de nombreuses distributions de systèmes d'exploitation en disposent ou proposent des outils pour les gérer. Ensuite, l'accès distant à travers SSH est sécurisé, toutes les données sont chiffrées et authentifiées. Enfin, comme les protocoles HTTP/S, Git et local, SSH est efficace et permet de comprimer autant que possible les données avant de les transférer.

Inconvénients

Le point négatif avec SSH est qu'il est impossible de proposer un accès anonyme au dépôt. Les accès sont régis par les permissions SSH, même pour un accès en lecture seule, ce qui s'oppose à une optique open source. Si vous souhaitez utiliser Git dans un environnement d'entreprise, SSH peut bien être le seul protocole nécessaire. Si vous souhaitez proposer de l'accès anonyme en lecture seule à vos projets, vous aurez besoin de SSH pour vous permettre de pousser mais un autre protocole sera nécessaire pour permettre à d'autres de tirer.

Protocole Git

Vient ensuite le protocole Git. Celui-ci est géré par un **daemon** spécial livré avec Git. Ce **daemon** (démon, processus en arrière-plan) écoute sur un port dédié (9418) et propose un service similaire au protocole SSH, mais sans aucune sécurisation. Pour qu'un dépôt soit publié

via le protocole Git, le fichier **git-daemon-export-ok** doit exister mais mise à part cette condition sans laquelle le **daemon** refuse de publier un projet, il n'y a aucune sécurité. Soit le dépôt Git est disponible sans restriction en lecture, soit il n'est pas publié. Cela signifie qu'il ne permet pas de pousser des modifications. Vous pouvez activer la capacité à pousser mais étant donné l'absence d'authentification, n'importe qui sur Internet ayant trouvé l'URL du projet peut pousser sur le dépôt. Autant dire que ce mode est rarement recherché.

Avantages
Le protocole Git est souvent le protocole avec la vitesse de transfert la plus rapide. Si vous devez servir un gros trafic pour un projet public ou un très gros projet qui ne nécessite pas d'authentification en lecture, il est très probable que vous devriez installer un **daemon** Git. Il utilise le même mécanisme de transfert de données que SSH, la surcharge du chiffrement et de l'authentification en moins.

Inconvénients
Le défaut du protocole Git est le manque d'authentification. N'utiliser que le protocole Git pour accéder à un projet n'est généralement pas suffisant. Il faut le coupler avec un accès SSH ou HTTPS pour quelques développeurs qui auront le droit de pousser (écrire) et le garder en accès **git://** pour la lecture seule. C'est aussi le protocole le plus difficile à mettre en place. Il doit être géré par son propre **daemon** qui est spécifique. Il nécessite la configuration d'un **daemon xinetd** ou apparenté, ce qui est loin d'être simple. Il nécessite aussi un accès à travers le pare-feu au port 9418 qui n'est pas un port ouvert en standard dans les pare-feux professionnels. Derrière les gros pare-feux professionnels, ce port obscur est tout simplement bloqué.

Pour l'installation et la mise en place de Git sur un serveur, veuillez consulter les liens suivants :

https://git-scm.com/book/fr/v2/Git-sur-le-serveur-Installation-de-Git-sur-un-serveur#s_git_on_the_server

https://git.goffinet.org/04-git-sur-le-serveur.html#installation

CLOUD ET NAS

Utilisation de la sauvegarde et de la synchronisation sur Internet

Le **NAS** (Network Attached Storage) est un serveur de fichiers indépendant connecté au réseau de l'entreprise. Il prend la forme d'un boîtier contenant le disques durs, la carte mère et le système logiciel. Il sert au stockage et à la sauvegarde des données, qui restent alors en interne, dans les locaux de l'entreprise.
Le **Cloud**, quant à lui, permet à l'entreprise d'externaliser le stockage de ses données, via un prestataire. C'est alors lui qui est en charge de la maintenance des serveurs de stockage. Si l'entreprise opte pour une solution Cloud en mode Saas, le prestataire gère également l'application. L'entreprise n'a plus qu'à souscrire à un abonnement pour accéder au service, et ne gère plus ni l'installation ni la maintenance de la solution de stockage de données.

NAS vs Cloud : le coût

Opter pour un **NAS** signifie des **coûts initiaux élevés** pour l'achat du matériel. A ceux-ci s'ajoutent les frais liés à l'installation, au paramétrage, et à **la maintenance sur le long terme**. En effet, un NAS suppose l'intervention d'un informaticien pour l'administration et la mise à jour du système.

Les solutions Cloud en mode SaaS, quant à elles, vous permettent d'optimiser le coût du stockage de vos données, puisque vous pouvez bénéficier de prestations sur-mesure pour ne payer que ce que vous consommez :

vous n'achetez pas par exemple un NAS de 2 To alors que vous avez 500 Go de données à stocker. Vous pouvez ajuster l'espace de stockage lorsque vous le souhaitez, vous jouissez ainsi d'une plus grande flexibilité. De plus, vous n'avez qu'à souscrire à un abonnement pour profiter d'une solution clé en main, prête à l'emploi, qui ne demande aucune installation, et dont les mises à jours sont déployées automatiquement par votre prestataire. Oubliez toutefois les solutions grands publics, bien que gratuites elles n'offrent pas assez de garanties de sécurité pour une utilisation professionnelle.

NAS ou Cloud : l'accessibilité aux données

Si le NAS convient très bien au stockage de données, qu'en est-il de l'accès distant ?
Les solutions Saas de Cloud proposent en général des applications mobiles gratuites pour accéder à vos fichiers sur smartphones et tablettes, et si certains fabricants de NAS en disposent aussi, il est néanmoins beaucoup plus compliqué d'accéder à ses documents en mobilité. De la même manière, si le fabricant de NAS ne propose pas de synchronisation, vous serez entièrement dépendant d'une bonne connexion Internet pour accéder à vos documents lors de déplacements.

Comparaison entre NAS et Cloud : gestion des droits, traçabilité et fonctions collaboratives

Gérer la gestion des droits d'accès sur les documents stockés dans un NAS suppose de passer par un prestataire informatique, ou de monopoliser un informaticien en interne, ce qui est coûteux. Toutefois, ne négligez pas cette étape si vous optez pour un stockage de données dans un NAS, car la confidentialité de vos données pourrait être remise en cause.
Avec une solution Cloud en mode Saas, vous administrez vous même votre plateforme et la gestion des droits, aussi finement que le permet le fournisseur que vous aurez choisi.

Au niveau de la traçabilité des actions réalisées sur les documents, là encore vous aurez besoin de l'intervention d'un spécialiste en informatique pour avoir un retour fiable. Les solutions cloud vous proposent par contre pour la plupart d'accéder à un journal d'évènement répertoriant l'ensemble des actions réalisées sur les documents.

Pour le partage collaboratif de documents, il est impossible avec un NAS. Des solutions cloud vous permettent par contre de partager des documents et des dossiers par simple génération de lien, en quelques clics seulement.

LES DIFFÉRENTES TECHNIQUES DE SAUVEGARDE

Cryptage de données

Sous sa forme la plus simple, le cryptage des données consiste à convertir les données d'une information lisible (texte en clair ou texte ordinaire) en des chaînes inintelligibles (texte chiffré) au moyen d'une valeur dite clé de cryptage.

Nous pouvons ainsi traiter et stocker des données en toute sécurité en préservant leur confidentialité et intégrité. Un second aspect important du **cryptage des données** réside dans la suite des étapes suivies pour échanger sans risque des clés et du texte crypté dans des messages : **le protocole de sécurité.**

On crypte les données en les soumettant à un algorithme de cryptage itératif (chiffre), pour combiner le texte en clair avec la clé de cryptage. Au terme de ce processus, les données en clair sont devenues illisibles sous forme de texte chiffré. Bien entendu, la clé de cryptage doit être strictement protégée, et c'est l'objectif le plus important de tous les protocoles de cryptage.

Vous pensez peut-être que l'algorithme de cryptage lui-même doit être tenu secret, mais ce genre de secret va plutôt à l'encontre de la sécurité du cryptage. En effet, les algorithmes de cryptage, bien que souvent privés (propriétaires), sont à la disposition du public, parce que ce n'est que quand les algorithmes sont ouverts pour l'inspection et le test que les utilisateurs leur font le plus confiance. Par conséquent, la clé de cryptage est l'élément le plus précieux dans ce processus.

Une **clé de cryptage** est une suite de caractères aléatoires, souvent divisée en plusieurs parties ou sections. D'une manière générale, plus la clé de cryptage est longue plus il est difficile de décoder un message par «force brutale», c'est-à-dire en essayant toutes les valeurs de clé possibles. Cette longueur fait aussi que le texte chiffré résultant est encore plus éloigné du texte en clair original. Pour renforcer l'intégrité de cryptage, un procédé courant consiste à demander à plusieurs utilisateurs autorisés de créer chacun leur propre partie de la clé de cryptage complète. On l'aura compris, ce morcellement garantit qu'aucun d'eux ne pourra décrypter l'information à lui tout seul.

Différents algorithmes de cryptage peuvent être utilisés selon les données concernées. Les questions sont nombreuses :

- **l'information est-elle stockée dans un seul champ, ou faudra-t-il crypter tout le fichier ?**
- **S'il n'y a qu'un champ, quelle est sa taille ?**
- **Les données doivent-elles être transmises d'un système à un autre ?**

Le **cryptage est symétrique** quand une même clé sert au **cryptage** et au **décryptage**. C'est généralement le cas pour un usage interne. Le c**ryptage asymétrique** fait intervenir une paire de clés, **une pour le cryptage** et l**'autre pour le décryptage**. Dans ce cas, l'une des clés est publique et l'autre privée. Ce procédé est utile pour échanger des informations avec d'autres entités. L'envoyeur crypte ses données avec la clé publique du destinataire, lequel décrypte le texte chiffré résultant au moyen de sa clé privée, toujours gardée secrète. Cette façon de faire résoud le problème de devoir partager une clé secrète entre des parties : les clés publiques peuvent être publiées dans un répertoire ouvert à tous, de sorte que chacun peut envoyer des messages sécurisés aux membres du répertoire.

Utilisation de EFS

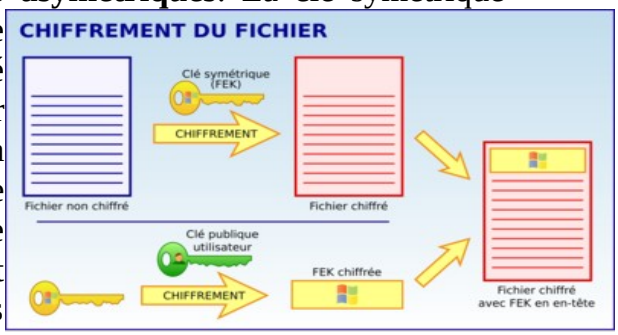
L'**Encrypting File System** (abrégé **EFS**) est une fonctionnalité apparue avec la troisième version des systèmes de fichiers **NTFS** disponible sur **Microsoft Windows 2000** et ultérieurs. Cette technologie permet d'enregistrer des fichiers **chiffrés** sur ce système de fichiers, ce qui protège les informations personnelles des attaques de personnes ayant un accès direct à l'ordinateur.

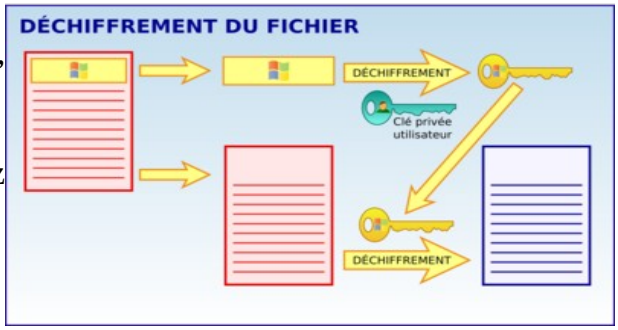
L'**authentification** de l'utilisateur et les listes de contrôle d'accès peuvent protéger les fichiers d'un accès non autorisé pendant l'exécution du système d'exploitation. Mais elles sont facilement contournables si un attaquant obtient un accès physique à l'ordinateur. Une solution est de **chiffrer les fichiers** sur les disques de cet ordinateur. EFS réalise cette opération en utilisant la cryptographie symétrique, et assure que le déchiffrement des fichiers est pratiquement impossible sans posséder la bonne clé. Cependant, EFS ne prévient pas les attaques par force brute contre les mots de passe des utilisateurs. Autrement dit, le chiffrement de fichier ne procure pas une bonne protection si le mot de passe utilisateur est facilement trouvable.

Fonctionnement

Les fichiers et dossiers qui doivent être chiffrés par le système de fichiers doivent être marqués par un attribut de chiffrement. Tout comme les permissions de fichiers dans NTFS, si un dossier est marqué pour être chiffré, alors **par défaut** tous les fichiers et sous-dossiers seront aussi chiffrés. Quand les fichiers sont copiés vers un autre volume formaté avec un autre système de fichiers (par exemple **FAT32**), ils sont automatiquement déchiffrés avant d'être déplacés. La seule exception se produit lorsque ces fichiers sont archivés; dans ce cas, ils restent chiffrés.

EFS fonctionne en chiffrant un fichier avec une **clé symétrique** générale, appelée ***File Encryption Key*** ou ***FEK.*** Cette clé permet un temps de chiffrement/déchiffrement relativement court pour les fichiers volumineux, par rapport aux **clés asymétriques**. La clé symétrique utilisée est elle-même chiffrée avec une clé publique associée à l'utilisateur qui a chiffré le fichier; la clé symétrique chiffrée est stockée dans l'en-tête du fichier chiffré. Pour le déchiffrer, le système de fichiers utilise la clé privée de l'utilisateur pour déchiffrer la clé symétrique située dans l'en-tête. Enfin, il utilise cette clé symétrique pour le déchiffrement du fichier. Étant donné que tout est réalisé au niveau du système de fichiers, ces opérations sont invisibles pour l'utilisateur.





Pour comparaison, on trouve sur les systèmes GNU/Linux, le système LUKS.

Pour plus d'information concernant l'EFS, veuillez consulter le lien suivant :

https://docs.microsoft.com/fr-fr/security-updates/security/20200345#:~:text=Pour%20utiliser%20EFS%2C%20il%20est,Certificats%20EFS.

Autres outils (TrueCrypt)

TrueCrypt est à la fois un format de système de fichier chiffré, notamment géré par Linux dans son module **dm-crypt** depuis la version 3.131,2, et un logiciel de **chiffrement à la volée** fonctionnant sur **Microsoft Windows** XP/2000/2003/Vista (32-bit et 64-bit)/7, Mac OS X et GNU/Linux, ce dernier étant à l'origine de ce système de fichier.
TrueCrypt est gratuit et son code source est disponible bien qu'il n'ait pas le statut de logiciel libre. Le logiciel **tc-play**, pour Linux, est par contre un logiciel libre sous **Licence BSD**, dont la version 1.0 est sortie en et qui est compatible avec TrueCrypt3.
TrueCrypt permet de créer un disque virtuel chiffré (*volume TrueCrypt*) contenu dans un fichier et de le monter comme un disque physique réel. TrueCrypt peut aussi chiffrer une partition entière ou un périphérique, par exemple une disquette ou une clé USB. Le chiffrement est automatique, en temps réel et transparent.
Tout ce qui sera stocké dans un volume TrueCrypt sera entièrement chiffré, y compris noms de fichiers et répertoires. Les volumes TrueCrypt se comportent, une fois montés, comme des disques durs physiques. Il est ainsi possible, par exemple, d'en réparer le système de fichiers avec ***Check Disk***, ou de **défragmenter** les volumes créés par TrueCrypt.
TrueCrypt n'est plus maintenu par ses auteurs originaux depuis le 28 mai 2014, mais plusieurs alternatives reprenant le code du logiciel (***forks***), don't **VeraCrypt**, ont vu le jour depuis, ainsi que différents projets d'hébergement des anciennes versions de TrueCrypt.

TrueCrypt se destine principalement au stockage chiffré des données, non à leur transmission de façon sécurisée comme peut le faire PGP. TrueCrypt ne gère pas le problème de la transmission de la clef (le mot de passe) qui doit être transmis sur un autre canal par **Diffie-Hellman** ou similaire en cas d'envoi d'un volume contenant des données à un destinataire. La taille des volumes (de quelques Mio à plusieurs Gio/Tio) peut dans certains cas ne pas se prêter commodément à un transport autre que physique.
En tant que logiciel de chiffrement à la volée, TrueCrypt ne protège pas d'éventuelles failles de sécurité. Il convient donc une fois un volume ouvert (on dit aussi «monté») de vérifier les droits d'accès à ce disque (chevaux de Troie, droits utilisateurs, partage réseau...) tout comme de l'accès physique à la machine.
Il ne s'agit pas là de failles de sécurité : Truecrypt n'a pas pour objet de protéger contre des accès une fois un volume ouvert.
Une analogie serait un coffre-fort dans des bureaux. Le coffre fermé, seuls ceux qui connaissent la clef peuvent l'ouvrir, mais une fois ouvert tout le monde peut accéder à son contenu. Il convient donc de protéger l'accès au coffre quand il est ouvert et de veiller avec qui partager la clef.
Une fois ce principe acquis, Truecrypt fait preuve d'une solidité intéressante. La version 6.0a a d'ailleurs été qualifiée au niveau élémentaire le par l'ANSSI (durée de validité d'un an).

Pour plus d'information concernant l'EFS, veuillez consulter le lien suivant :
https://fr.wikipedia.org/wiki/TrueCrypt

https://docs.microsoft.com/fr-fr/security-updates/security/20200345#:~:text=Le%20syst%C3%A8me%20EFS%20est%20une,de%20stockage%20de%20l'ordinateur.

……...Conclusion……

Un serveur informatique est un dispositif informatique (matériel et logiciel) qui offre des services à un ou plusieurs clients (parfois des milliers). Cependant Git dispose de par son système de versioning un moyen qui nous facilite l'accès à nos données, on peut remarquer qu'un ensemble de protocoles est mis en jeu afin de faciliter le transport des données.

Le **NAS** est un serveur de fichiers indépendant connecté au réseau de l'entreprise et le **Cloud** est un système de Stockage en ligne via internet.

le cryptage des données consiste à convertir les données d'une information lisible (texte en clair ou texte ordinaire) en des chaînes inintelligibles (texte chiffré) au moyen d'une valeur dite clé de cryptage.

L'**Encrypting File System** (abrégé **EFS**) est une fonctionnalité apparue avec la troisième version des systèmes de fichiers **NTFS** disponible sur **Microsoft Windows 2000** et ultérieurs. Cette technologie permet d'enregistrer des fichiers **chiffrés** sur ce système de fichiers, ce qui protège les informations personnelles des attaques de personnes ayant un accès direct à l'ordinateur.

TrueCrypt est à la fois un format de système de fichier chiffré, notamment géré par Linux dans son module **dm-crypt** depuis la version 3.131,2, et un logiciel de **chiffrement à la volée** fonctionnant sur **Microsoft Windows** XP/2000/2003/Vista (32-bit et 64-bit)/7, Mac OS X et GNU/Linux, ce dernier étant à l'origine de ce système de fichier.