

Bastien Gorissen & Thomas Stassin

PROLOGUE

Où nos héroïnes découvrent une contrée
inconnue...

THIS IS GIT. IT TRACKS COLLABORATIVE WORK
ON PROJECTS THROUGH A BEAUTIFUL
DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL
COMMANDS AND TYPE THEM TO SYNC UP.
IF YOU GET ERRORS, SAVE YOUR WORK
ELSEWHERE, DELETE THE PROJECT,
AND DOWNLOAD A FRESH COPY.



PETIT SONDAGE...

Avant de plonger dans le vif du sujet, voyons quelle est la situation :D

Levez la main si...

- ...vous avez entendu le nom Git.
- ...vous avez une idée de ce que Git fait.
- ...vous pensez en avoir besoin.
- ...vous avez déjà perdu des changements dans votre code (dans un crash disk ça compte :D).
- ...vous n'avez pas peur de la ligne de commande.

PRÉSENTATION

Git, les élèves.
Les élèves, Git.

GIT, ÇA SERT À QUOI ?

Git est un système de contrôle de version distribué.

Son rôle est de vous aider à garder une trace des versions successives de votre code, et de vous assister quand vous devez collaborer sur des projets.

Idéalement, **TOUS vos projets devraient utiliser Git.**

Non seulement pour vous entraîner, mais pour éviter toute catastrophe du genre: "Oh mon chat a marché sur le clavier et tout effacé !", "Mon petit frère a versé son verre d'eau sur mon laptop !", "Mon disque dur est tombé dans un volcan !").

ET CONCRÈTEMENT ?

Git garde en mémoire tous les changements successifs que vous effectuez sur votre code, en faisant une "photo" de votre code quand vous lui demandez de le faire.

Vous pouvez donc à tout moment annuler un changement problématique, ou retrouver un morceau de code que vous auriez supprimé.

Git construit un historique de vos changements:



OK. ET COMMENT ON UTILISE ÇA ?

Nous allons utiliser 2 choses :

1) <https://github.com/>

C'est un site qui vous permet de sauvegarder sur "le cloud" vos projets Git, pour que vous y ayez toujours accès.

2) <https://cmder.net/> (full version pour avoir les commandes Git)

Une ligne de commande (miam) qui permet d'interagir avec Git.

Oui, il existe des GUI, mais non, on ne va pas les utiliser tout de suite :p

NAVIGUER AVEC GIT-BASH

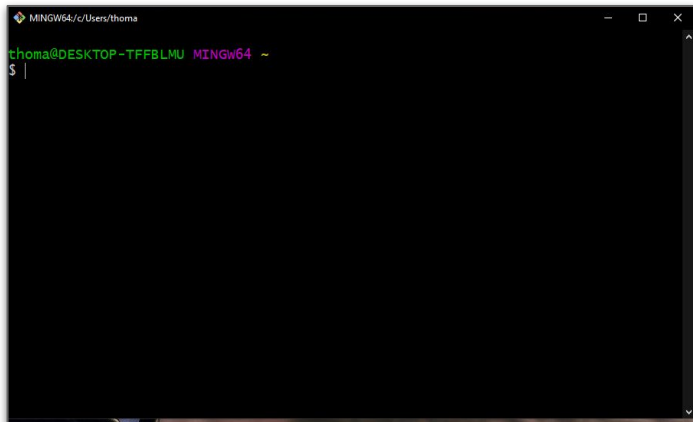
cd? ls??

JOUER À LA CONSOLE

La console Git bash peut-être démarrée de deux manières différentes:

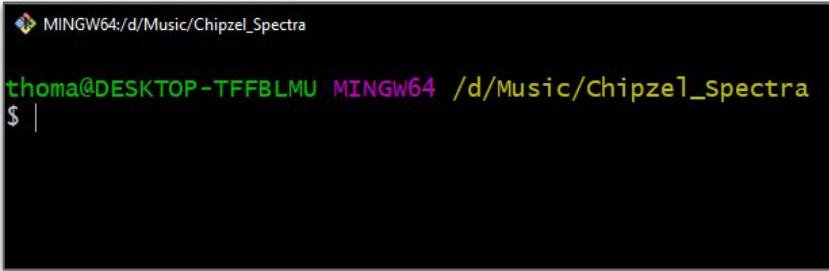
- Pressez la touche Windows (ou cliquez sur l'icône), tapez "**git bash**", suivi de [Enter]
- Dans l'explorateur Windows, cliquez droit dans un dossier, puis "Git Bash Here"

Cette seconde option va vous positionner directement dans le bon dossier et nous allons la privilégier. Mais il est malgré tout important d'apprendre à se déplacer dans l'arborescence de vos dossiers via quelques commandes simples.



LE CHEMIN

Le chemin d'un dossier (ou d'un fichier) est l'endroit où celui-ci se trouve sur votre ordinateur.

A screenshot of a Windows command prompt window. The title bar at the top reads 'MINGW64: /d/Music/Chipzel_Spectra'. The command prompt shows the user 'thoma@DESKTOP-TFFBLMU' in green, the shell 'MINGW64' in pink, and the current directory '/d/Music/Chipzel_Spectra' in yellow. Below this, a prompt character '\$' is followed by a vertical bar '|', indicating the cursor is ready for input.

```
MINGW64: /d/Music/Chipzel_Spectra  
thoma@DESKTOP-TFFBLMU MINGW64 /d/Music/Chipzel_Spectra  
$ |
```

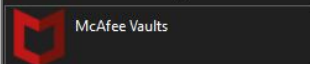
Ici on peut constater que le chemin (en jaune) est

`/d/Music/Chipzel_Spectra`

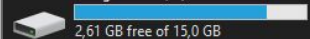
Ce qui indique que le dossier se trouve sur le disque **d** dans un dossier `Music/Chipzel_Spectra`

EN CLAIR: /d/Music/Chipzel_Spectra

Devices and drives (4)



Google Drive (G:)



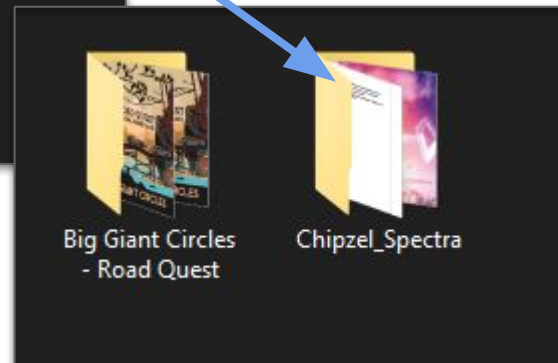
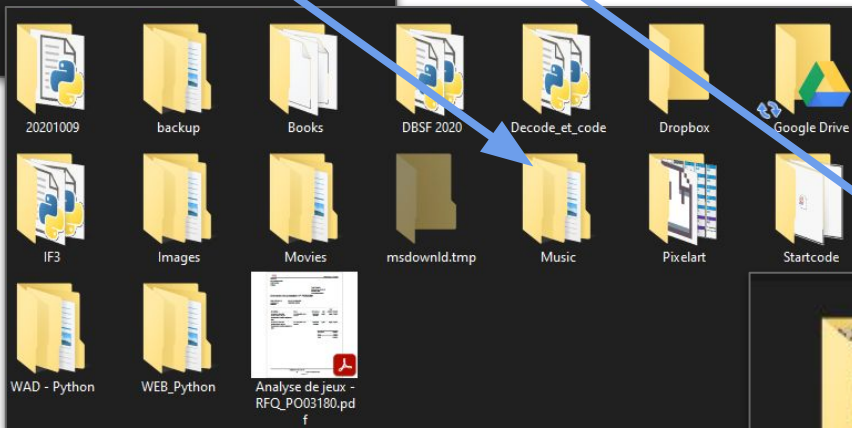
OS (C:)

20,2 GB free of 457 GB



DATA (D:)

864 GB free of 931 GB



QUELQUES INSTRUCTIONS UTILES

Pour naviguer dans la console Git-bash, vous avez besoin de 2 commandes, principalement :

1. `cd`
2. `ls`

Il y a évidemment tout un panel de commandes qui existent, mais ces deux là sont les seules qu'il vous sera nécessaire de mémoriser pour naviguer à travers les dossiers.

CD COMME...

cd est l'abréviation de "*change directory*" (ou "changer de dossier"), et permet de changer de dossier.

Par exemple, si je suis dans le dossier `\c\MonDossier`, la commande suivante :

cd superJeu

m'amènera dans le dossier

`\C\MonDossier\superJeu`

CD, DEUXIÈME SERVICE

Deux cas particuliers...

Si je veux "remonter" dans le dossier parent (c'est à dire le dossier contenant le dossier en cours) :

```
cd ..
```

Et si je veux changer de disque (par exemple passer de C: à D:) :

```
cd d:
```

Attention cette façon de faire est propre à git-bash qui est basé sur Linux. Mais comme sous Linux la notion de lettre pour un disque n'existe pas, c'est une adaptation pour Windows. En DOS la commande pour changer de disque est la lettre suivie de ":" et serait donc d:

LS COMME "LISTE" ? (OUI OK, C'EST PAS SI ÉVIDENT...)

Exact, la commande "ls" permet de lister le contenu d'un dossier, utile quand on veut vérifier que notre fichier est bien dans le dossier actuel.

Voyons un exemple...

thoma@DESKTOP-TFFBLMU MINGW64 /d/Music/Chipzel_Spectra

\$ ls

'Chipzel - Spectra - 01 Spectra.mp3'
'Chipzel - Spectra - 02 Tokyo Skies.mp3'
'Chipzel - Spectra - 03 Forged in Stars.mp3'
'Chipzel - Spectra - 04 Formed in the Clouds (Interlude).mp3'
'Chipzel - Spectra - 05 Only Human.mp3'
'Chipzel - Spectra - 06 Aurora Borealis.mp3'
'Chipzel - Spectra - 07 Evolution.mp3'
'Chipzel - Spectra - 08 Beyond the Cosmos.mp3'
'Chipzel - Spectra - 09 Veteran.mp3'
'Chipzel - Spectra - 10 Sonnet.mp3'
'Chipzel - Spectra - 11 Sunday.mp3'
'Chipzel - Spectra - 12 The Art of War.mp3'
'Chipzel - Spectra - 13 Final Credits.mp3'
'Chipzel - Spectra - Chipzel Spectra wallpaper (1) 1680x1050.png'
'Chipzel - Spectra - Chipzel Spectra wallpaper (2) 1680x1050.png'
'Chipzel - Spectra - Chipzel Spectra wallpaper (2) 1920x1080.png'
'Chipzel - Spectra - Chipzel Spectra wallpaper(1) 1920x1080.png'
'Chipzel - Spectra - Chipzel Umbrella wallpaper 1680x1050.png'
'Chipzel - Spectra - Chipzel Umbrella wallpaper 1920x1080.png'
'Chipzel - Spectra - Spectra Artwork.png'
'Chipzel - Spectra - Spectra Inlay.png'
'Chipzel - Spectra - Tokyo Skies save file.pdf'
cover.png

thoma@DESKTOP-TFFBLMU MINGW64 /d/Music/Chipzel_Spectra

\$ |

REPO 101

Back to basics

LA GENÈSE D'UN "REPO"

Avant d'utiliser github, nous allons voir comment utiliser Git en local, sur votre PC.

- Ouvrez git-bash (touche windows, tapez "**git bash**", [Enter], ou bien clic droit dans l'explorateur, "Git Bash Here")
- Faites en sorte d'être sur le dossier que vous voulez transformer en repo.
- ensuite tapez la commande **git init**

Et voilà ! Votre répertoire courant est connu par Git, et Git a créé un "repository" pour traquer les changements. Il s'agit du sous-dossier .git que Git-bash mentionne.

SE PRÉSENTER PAR POLITESSE

La première chose à faire (et à ne faire qu'une seule fois), est de nous présenter à Git. Pour ce faire, on peut utiliser les commandes suivantes:

- **git config --global user.name "My Name"**
- **git config --global user.email "email@provider"**

Git va maintenant attacher notre identité à tous nos changements, ce qui pourra être pratique en cas de collaboration avec d'autres.

Notez que si vous ne mettez pas **--global**, vous changez la config pour le repo dans lequel vous êtes.

LES COMMANDES DE BASE

Nous allons maintenant voir les commandes principales de Git, celles que vous allez utiliser tous les jours, plein de fois par jour.

- `git status`
- `git add`
- `git commit`

Une première chose à savoir : Git peut faire plein de choses, mais au moins, il vous donne en général pas mal d'info sur ce qu'il se passe. Il faut donc prendre l'habitude de bien lire ce qu'il vous dit :)

STATUS QUO

Testons directement la première commande :

- `git status`

On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)

Résumé : il n'y a rien, nada, et Git vous donne comme conseil d'utiliser **git add** pour changer ça.

STATUS QUO

`git status` permet donc de voir l'état actuel de votre repo.

Par état, on veut dire quels changements ont été opérés depuis le dernier "snapshot" de votre code, quels fichiers ont été supprimés, etc.

Prenez l'habitude de l'utiliser tout le temps pour voir où vous en êtes.

C'est la commande la plus utile de votre arsenal.

Mais donc, ce fameux `git add...`

ADDICTED TO GIT

git add permet, comme son nom l'indique, d'ajouter un fichier, ou les changements d'un fichier, dans le prochain paquet de changements traqué par Git.

Comme il vaut toujours mieux essayer par soi-même...

- Créez un nouveau fichier ("test.txt" par exemple) dans le répertoire.
- **git status**
- **git add test.txt**
- **git status**

MAIS C'EST QU'IL EST GENTIL...

Remarquez que Git continue de vous donner des conseils sur les probables prochaines étapes que vous pourriez vouloir suivre.

Mais donc, qu'avons-nous fait ?

- Nous avons dit à Git "test.txt fait partie des fichiers que tu dois tenir à l'oeil"
- Git en a pris note, et fera donc à l'avenir mention des changements qui sont effectués sur le fichier.
- Git est prêt à "commiter" le fichier.

COMMIT ?

Depuis le départ, nous parlons de faire des "snapshots" ou des "photos" de notre code. En réalité, on nomme ça des "commits".

Chaque commit est donc une étape dans votre développement, et représente un ensemble de changements sur vos fichiers sources.

Git vous laisse ajouter (**git add**) tout ce que vous voulez avant de finalement fermer le commit et le considérer comme un paquet isolé (**git commit**).

COMMIT D'OFFICE

Dans notre exemple, nous avons donc ajouté notre fichier "test.txt" au commit en cours.

Git appelle cette "zone" intermédiaire la "staging area". Vous pouvez voir ça comme une boîte en carton ouverte. Chaque **git add** rajoute quelque chose dans la boîte, et quand vous voulez la fermer :

- **git commit -m "Added our first file to the repo."**

Conseil : utilisez toujours -m "message", sinon...

REMOTE?

On va *pousser* un cran plus loin

C'EST PAS FOLICHON

Bon, utiliser Git sur son PC, c'est bien, mais ça ne nous sauve pas d'un crash de disque dur.

Pour ça, on peut envoyer son code ailleurs, par exemple sur github(ou sur un autre serveur).

Avant de voir ça, voyons les commandes dont nous allons avoir besoin pour synchroniser notre travail en local avec un serveur distant.

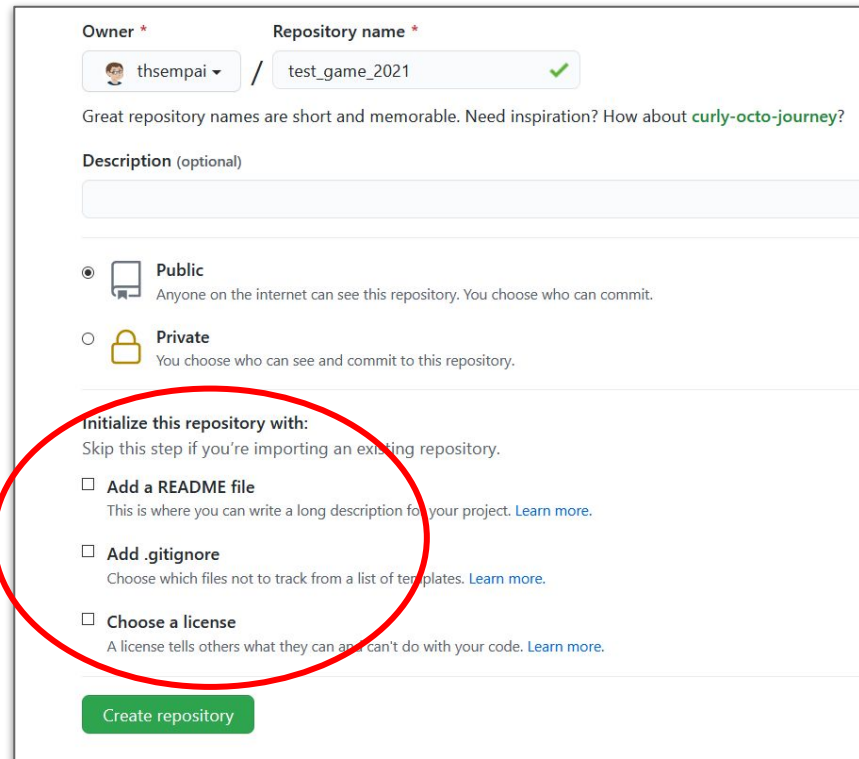
LA PREMIÈRE POUSSÉE

Pour utiliser Git avec un remote server, il faut les lier entre eux.

Pour ce faire, la première chose à faire est de créer un repo sur un server. Pour ce cours, nous allons utiliser Github.

Créez donc un nouveau repo sur votre server.

Dans ce cas-ci veuillez bien à ce que toutes les cases à cocher restent non-cochées.





Owner * / Repository name *

thsempai / test_game_2021 ✓

Great repository names are short and memorable. Need inspiration? How about [curly-octo-journey?](#)

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

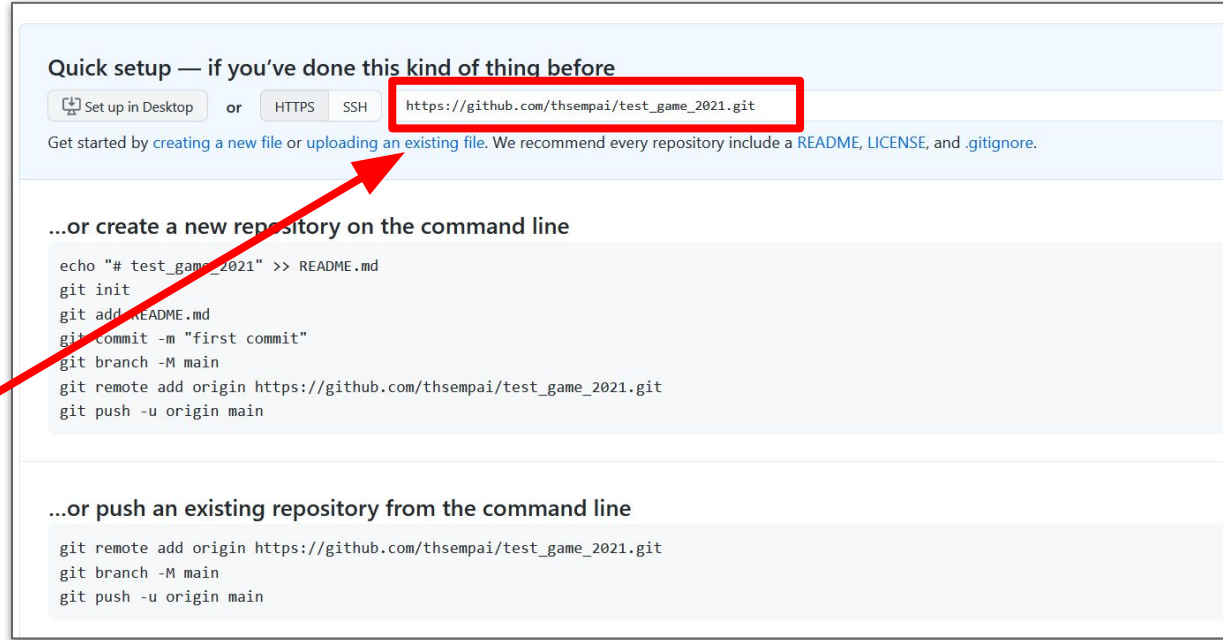
☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

[Create repository](#)

LA PREMIÈRE POUSSÉE

Une fois que vous avez créé le repo, vous aurez un écran similaire à l'image de droite.

Il vous reste pour le moment à copier l'url du repo qui se trouve ici.



LA PREMIÈRE POUSSÉE

Ensuite dans git-bash vous aller taper la ligne de commande

```
git remote add origin <url>
```

qui, je vous le donne en mille, ajoute un remote à votre git "local"
(origin est le nom que l'on donne habituellement au serveur remote)

D'ailleurs, si on fait `git remote -v`

On verra qu'il a bien été ajouté, une ligne pour le push une autre pour le fetch (ces deux notions seront vues plus tard).

```
thoma@DESKTOP-TFFBLMU MINGW64 /d/test (master)
$ git remote -v
origin https://github.com/thsempai/test_game.git (fetch)
origin https://github.com/thsempai/test_game.git (push)
```


UN DERNIER EFFORT

Une fois que c'est fait, il vous reste à pousser les commits.

La commande pour effectuer cela est **git push**, mais si c'est votre première fois, cela ne fonctionnera pas, mais comme souvent git vous expliquera ce qu'il faut faire.

```
$ git push
fatal: The current branch master has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin master

thoma@DESKTOP-TFFBLMU MINGW64 /d/test (master)
```

Donc la première fois, vous devez faire **git push --set-upstream origin master** ou **git push -u origin master**

ON Y EST :D

Notez que le mot `master` est pour nommer la branche (ce qui est indiqué entre parenthèses en cyan), mais cette notion sera vue plus tard.

```
thoma@DESKTOP-TFFBLMU MINGW64 /d/test (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 221 bytes | 221.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/thsempai/test_game.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Retenez juste que cette manoeuvre doit être réalisée à chaque premier *push* dans une branche. Le mot `master` sera remplacé par le nom de la branche dans ce cas. Mais, nous y reviendrons.

A DAY IN THE LIFE...

Au jour le jour, la séquence classique quand vous travaillez avec un "remote" sera :

- `git add`
- `git commit -m "Super commentaire"`
- `git pull`
- `git push`

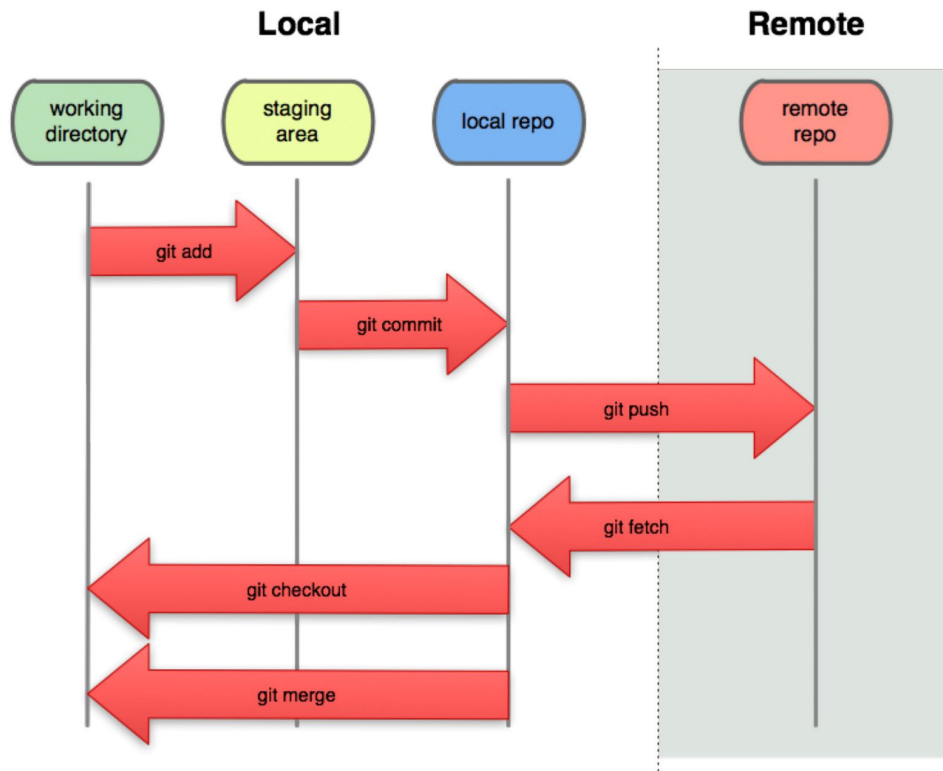
Après, évidemment, tout n'est pas toujours si simple :)

PLUSIEURS PC

Jardins & Loisirs

REMOTE CONTROL

- **git push** permet d'envoyer son travail (après l'avoir commité) sur le serveur.
- **git fetch** ramène les changements depuis le serveur (mais ne change pas vos fichiers locaux)
- Il existe **git pull** qui fait **fetch** + **merge**.



ON PEUT FAIRE UNE MICRO-RÉVISION ?

Nous avons vu les commandes de base :

`git status` pour savoir où on en est

`git add <mon_fichier>` pour ajouter mon_fichier au commit

`git commit -m "Mon message"` pour créer un commit

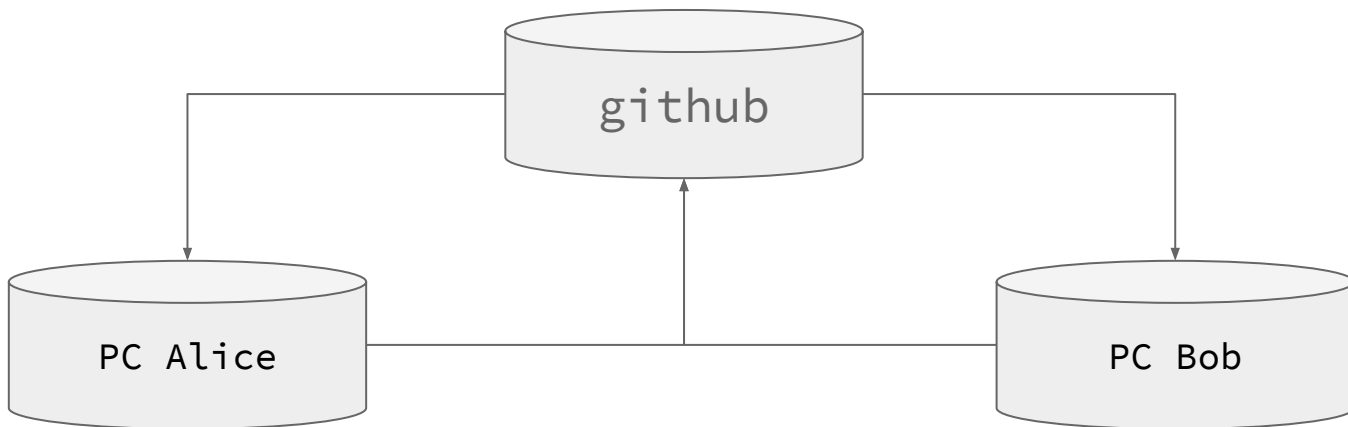
`git push` pour envoyer tout ça vers Github

Rappel : un commit est un "snapshot" de votre code à un moment donné.

A QUOI D'AUTRE SERT GIT ?

Git est un très bon outil de collaboration et d'organisation du travail.

Imaginez la situation suivante :



GIT S'Y RETROUVE ?

Git est capable de travailler avec plusieurs "historiques" qui varient entre eux et est capable de les "merger" (rassembler).

On peut voir ça comme des espèces d'univers parallèles.

Le nom technique est "branche".

Et Git vous donne le pouvoir de créer et manipuler ces univers parallèles comme bon vous semble !

Voyons comment ça se passe quand vous travaillez seules.

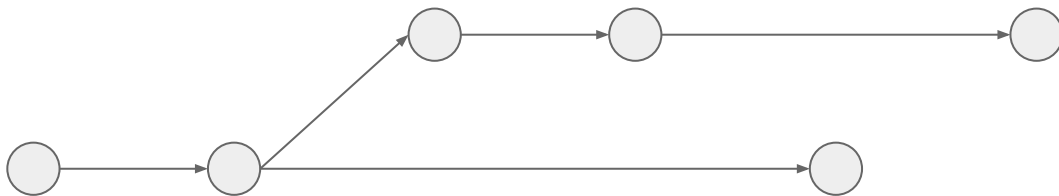
BRANCHEMENTS

Nous avons vu le cas "linéaire" simple :



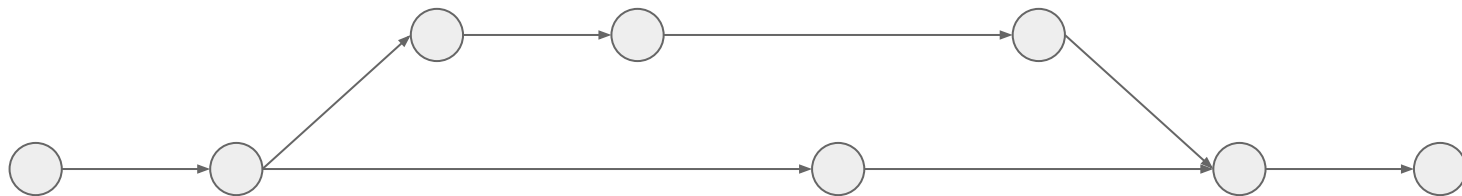
Mais imaginez que vous ayez deux fonctionnalités à implémenter.

Vous pouvez partir d'un point donné et aller dans deux directions différentes.



CROISEMENTS

Et bien entendu, les branches peuvent se rejoindre (on appelle ceci un "merge").



Git va donc créer un nouveau commit au moment du merge, et va ensuite vous laisser continuer à travailler à partir de là.

Voyons rapidement comment créer et merger des branches !

COMMANDES !

Voici les commandes principales pour les branches

git branch <ma_branche> pour créer une nouvelle branche

git checkout <ma_branche> pour passer sur la branche ma_branche

git checkout -b <ma_branche> pour créer la branche et passer dessus directement

git merge <ma_branche> pour merger la branche ma_branche dans la branche actuellement active

WAAAAIT A MINUTE...

Que se passe-t-il si on a modifié le même code sur 2 branches ?

Il se peut que vous ayez des conflits entre plusieurs branches au moment d'un merge.

C'est l'une des situations les plus "stressantes" en travaillant avec Git.

Nous allons donc tenter d'en créer un pour voir comment nous en dépêtrer...

CONFLICT CHEAT SHEET

Donc en cas de conflits :

- 1) Regarder quels fichiers posent problème
- 2) Choisir quoi garder
- 3) Sauvegarder les fichiers
- 4) Les ajouter à un nouveau commit
- 5) Commiter
- 6) Et voilà !

Maintenant que nous savons comment gérer les branches, voyons comment trouver notre "flow".

GIT FLOW

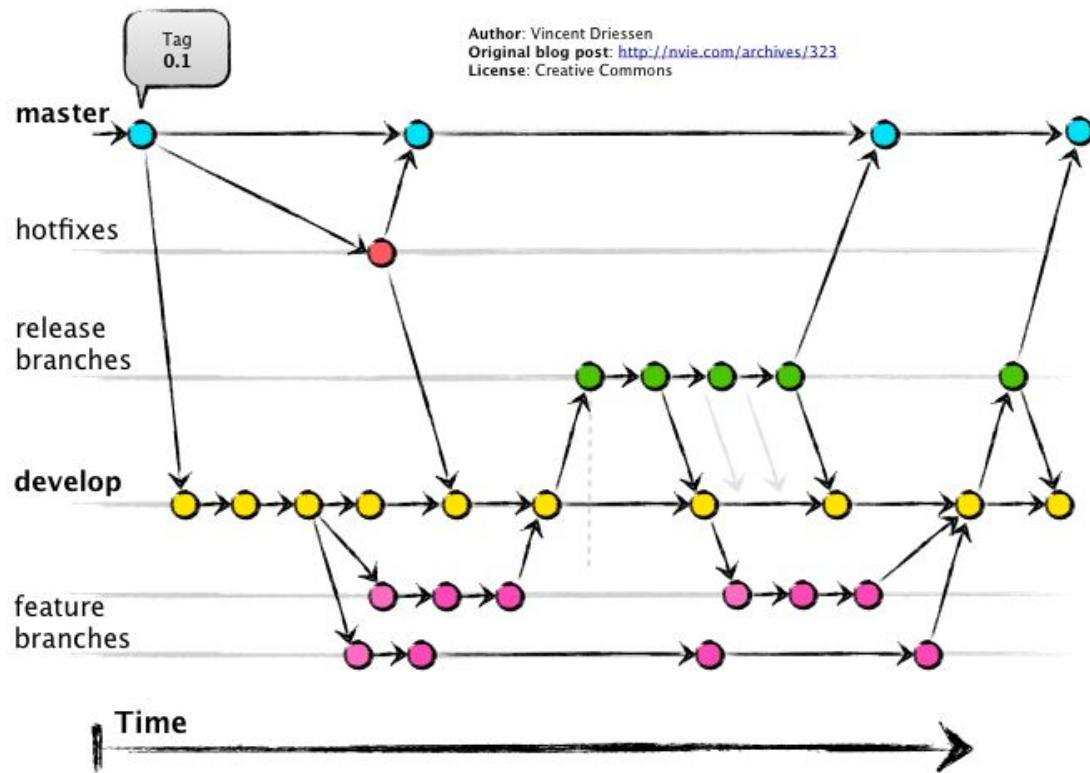
Go with the flow...

QU'EST-CE QUE C'EST ENCORE QUE CE TRUC ?

Git-flow est simplement une proposition de façon de travailler qui permet de rester organisé et clair. Le tout basé sur un certain nombres de branches:

- **main**: Contient les versions "officielles", considérées comme finies.
- **develop** : Branche principale, qui contient cependant toujours du code qui tourne.
- **feature/*** : Branches de travail. Une par fonctionnalité à implémenter.
- **release/*, hotfix/*** : Branches temporaires.

EN UNE IMAGE...



ET DONC ON FAIT PLEIN DE "GIT BRANCH" ?

La beauté de Git-flow est de vous mâcher le travail en proposant des commandes "toutes faites". Essayons !

- **git flow init**: Initialise le repo en tant que repo utilisant Git-flow.
- **git flow feature start <my_feature>** : Pour commencer une nouvelle feature.
- **git flow feature finish** : Pour clôturer une feature.
- **git release start <my_release>** : Commencer une release.
- **git flow release finish**
- Bonus: **git flow feature publish**