

Université de Toulouse

IUT de Blagnac

Département Réseaux et Télécoms

Mettre en place une solution informatique pour l’entreprise

Travaux Pratiques SAé23



**Nom: BOURAS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*Réseaux & Télécoms 1ière Année*

Table des matières

[1 - Principe de la gestion de version 2](#_Toc166434093)

[2 - Gestion de version d’un site web existant 4](#_Toc166434094)

[2.1 - Utilisation d’un site web existant 4](#_Toc166434095)

[2.2 - Création d’un compte GitHub 4](#_Toc166434096)

[2.3 - Installation et configuration de Git 6](#_Toc166434097)

[2.4 - Création d’un dépôt Git pour le site web 8](#_Toc166434098)

[2.5 - Création d’un dépôt distant sur GitHub pour le site web 12](#_Toc166434099)

[2.6 - Bilan 15](#_Toc166434100)

[3 - Récupération d’une version et gestion avancée des versions 16](#_Toc166434101)

[3.1 - Clonage d’un dépôt 16](#_Toc166434102)

[3.2 - Gestion de version 17](#_Toc166434103)

[3.3 - Gestion de l’historique 21](#_Toc166434104)

[3.4 - Exploitation des branches de l’historique 27](#_Toc166434105)

[3.5 - Gestion et visualisation avancées des branches 33](#_Toc166434106)

[4 - Formation complémentaire 37](#_Toc166434107)

[5 - Aide-mémoire Git 37](#_Toc166434108)

[5.1 - Initialisation d’un dépôt Git 37](#_Toc166434109)

[5.2 - Gestion des branches Git 38](#_Toc166434110)

[5.3 - Gestion des commits Git 38](#_Toc166434111)

[5.4 - Gestion des conflits de fusion 38](#_Toc166434112)

[5.5 - Gestion des tags Git 39](#_Toc166434113)

# Principe de la gestion de version

En entreprise, seule une petite minorité de solutions informatiques (logiciels, applications, sites Web…) sont conçues par un seul développeur. La grande majorité des projets est réalisée et/ou maintenue par une équipe de plusieurs personnes travaillant sur la même base de fichiers. Ce travail en parallèle est source de complexité :

* Comment récupérer le travail d'un autre membre de l'équipe ?
* Comment publier/partager ses propres modifications ?
* Comment faire en cas de modifications conflictuelles (travail sur le même fichier d'un ou plusieurs collègues) ?
* Comment accéder/revenir à une version précédente ?

Pour les raisons précédentes, toute solution informatique d'entreprise (même mono-développeur) doit faire l'objet d'une gestion des versions (**V**ersion **C**ontrol **S**ystem ou versioning en anglais).

La gestion des versions vise les objectifs suivants :

* assurer la pérennité des projets/solutions informatiques.
* permettre le travail collaboratif.
* fournir une gestion de l'historique des projets/solutions informatiques.
* fournir une information sur la nature des modifications réalisées.

La gestion des versions la plus basique consiste à déposer les fichiers sur un **dossier partagé** par l'équipe de développement et à créer des copies dans un autre dossier, pour assurer une sauvegarde en cas d’incidents. Si elle permet à tous de récupérer les fichiers, elle n'offre aucune solution aux autres complexités du développement en équipe et n'autorise pas la gestion simple des versions. Enfin elle n’est absolument pas à l’abri d’une erreur humaine (suppression de fichiers, suppression de répertoire, modification accidentelle de fichiers…).

Dans ce TP, vous allez apprendre à utiliser le gestionnaire de versions **Git** et le service **GitHub**, afin de pouvoir gérer au mieux votre travail de groupe lors de cette SAé. Ils vous seront également utiles pour les futures SAé des futurs semestres.

Git (qui signifie **idiot** en argot américain) est le système de contrôle de version le plus largement utilisé aujourd'hui. A l’origine, il a été créé, par Linus Torvalds en 2005, pour être un outil simple afin de faciliter le développement du noyau Linux.

C’est un outil que vous utiliserez **pour créer un dépôt local (**c’est-à-dire un espace de stockage, "repository" ou même "repo" en anglais) et **pour gérer les versions des fichiers**.Cet outil possède les caractéristiques suivantes :

* utilisation de la ligne de commande, qui est le seul moyen d’accès à toutes les commandes Git.
* disponible sur toutes les plates-formes (GNU/Linux, Windows, Solaris et Mac).
* structure décentralisée : l’historique complet des fichiers n’est pas conservé dans un unique emplacement. Chaque copie d’un fichier correspond à un nouveau dépôt dans lequel est conservé l’historique des modifications.
* logiciel libre / open source.

**GitHub est un service en ligne** qui héberge votre dépôt initialement local dans le cloud. Dans ce cas, on parle **de dépôt distant** puisqu’il n’est pas stocké sur votre machine. C’est le plus grand hébergeur de dépôts Git du monde.

Une grande partie des dépôts hébergés sur GitHub sont publics. Ce qui signifie que n’importe qui peut télécharger les fichiers de ces dépôts et contribuer éventuellement à leur développement en proposant de nouvelles fonctionnalités (sauf si le propriétaire du projet ne le permet pas, bien entendu).

Ainsi à partir de GitHub, vous allez pouvoir cloner des projets (dépôts) publics, dupliquer ("fork") des projets ou encore contribuer à des projets en proposant des modifications ("pull request").

Pour information :

* GitHub a été racheté par Microsoft en 2018
* Différentes offres GitHub : <https://fr.github.com/pricing.html>
* GitLab et Bitbucket sont des alternatives à GitHub



**En résumé** :

* Un **dépôt** ("**repository**" ou même "repo") est comme un dossier qui conserve un historique des versions et des modifications d’un projet. Il est essentiel pour travailler en équipe ou collaborer à un projet open source déjà existant.
* Un dépôt local est l’endroit où l’on stocke, sur sa machine, une copie d’un projet, ses différentes versions et l’historique des modifications.
* Un dépôt distant est une version dématérialisée du dépôt local. Il permet de centraliser le travail des développeurs dans un projet collectif.
* Il existe plusieurs services en ligne pour héberger un dépôt distant, GitHub étant l’un des plus populaires.

# Gestion de version d’un site web existant

## Utilisation d’un site web existant

Démarrez votre poste de travail sous Windows (à l’aide de vos identifiants personnels si possible).

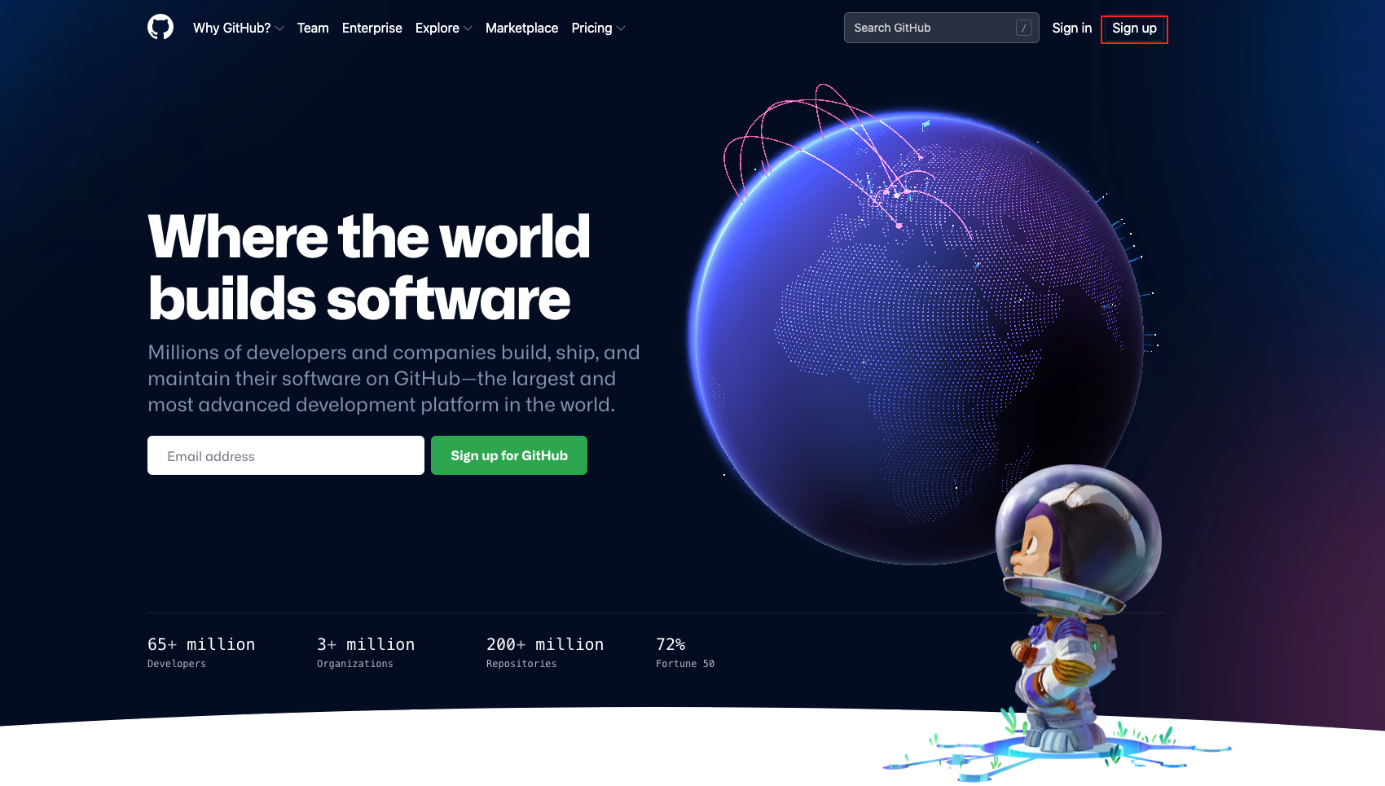
Copiez sur votre Bureau Windows le dossier **36th\_americas\_cup** à partir du serveur de fichiers **S2022** (\\192.168.100.1\Public\BUT1\SAé23). Ouvrez ensuite la page **index.html** dans un navigateur et validez le bon accès aux différentes pages du site web.

Vous devez obtenir une page d’accueil qui ressemble à celle-ci :

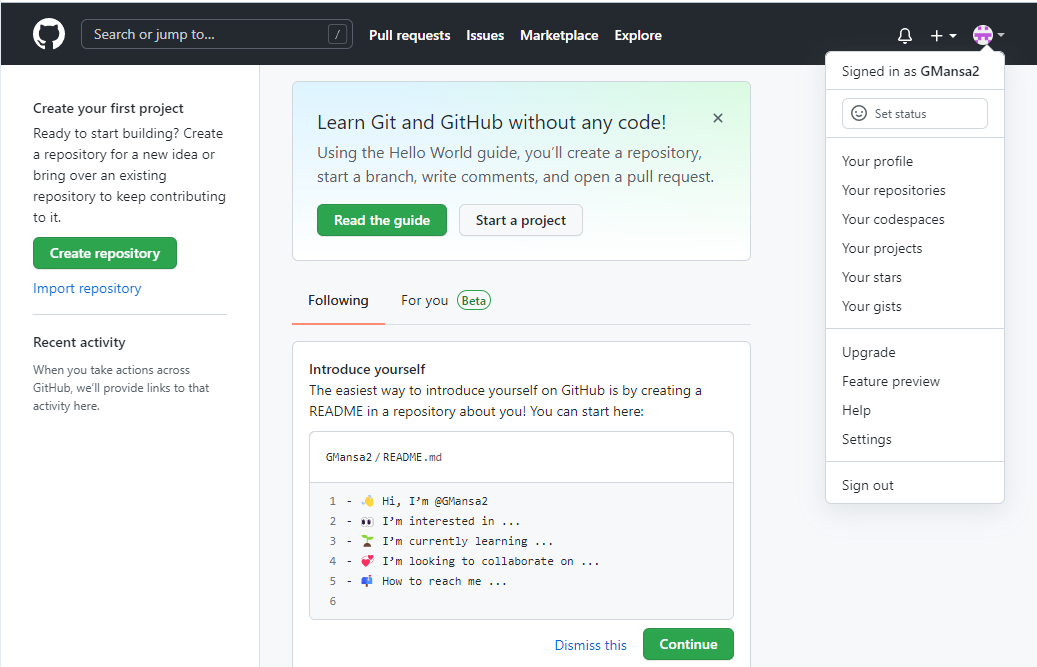


## Création d’un compte GitHub

Accédez à la page d’accueil de GitHub ([https://github.com](https://github.com/)) et cliquez sur **Sign up** en haut à droite de l’écran :



Complétez le formulaire proposé par un e-mail, un mot de passe et un nom d’utilisateur pour vous inscrire. Un code de vérification vous sera envoyé sur votre adresse e-mail afin de confirmer votre identité. Vous pouvez ensuite sauter la personnalisation en cliquant sur **Skip personalization**.

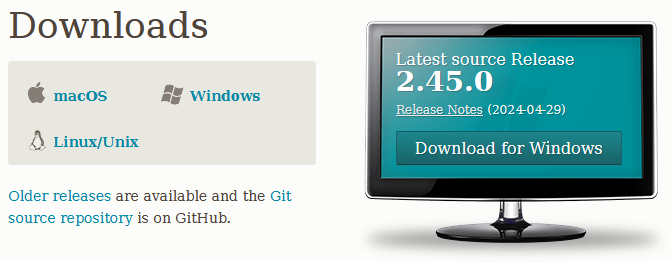
Vous pouvez découvrir cette interface assez facile à prendre en main et simple d’utilisation :

Vous reviendrez par la suite sur votre compte GitHub.

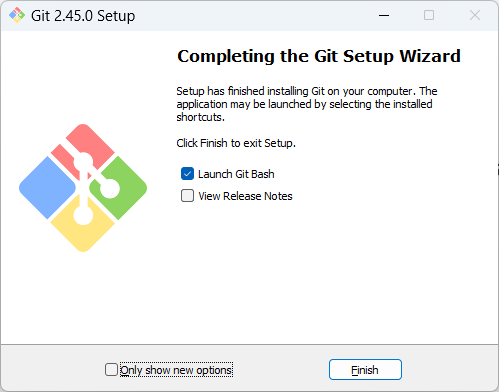
## Installation et configuration de Git

Pour assurer la gestion de version du projet en cours   
(et des futurs projets par la suite), il faut installer Git sur   
votre machine.

A partir de la page de téléchargement de Git (<https://git-scm.com/downloads>), téléchargez la dernière version de Git (actuellement version : 2.45.0).



Exécutez le fichier que vous venez de télécharger. Lors de l’installation, laissez toutes **les options par défaut**, elles conviennent bien, en cliquant sur **Next** à chaque fenêtre puis sur **Install**.



Cochez ensuite **Launch Git Bash**,  
et cliquez sur **Finish**.

Git Bash est l’interface permettant d’utiliser Git en ligne de commande. Comme son nom l’indique, vous devrez utiliser les commandes usuelles Bash.

Lancez la commande **git --version**

Résultat :



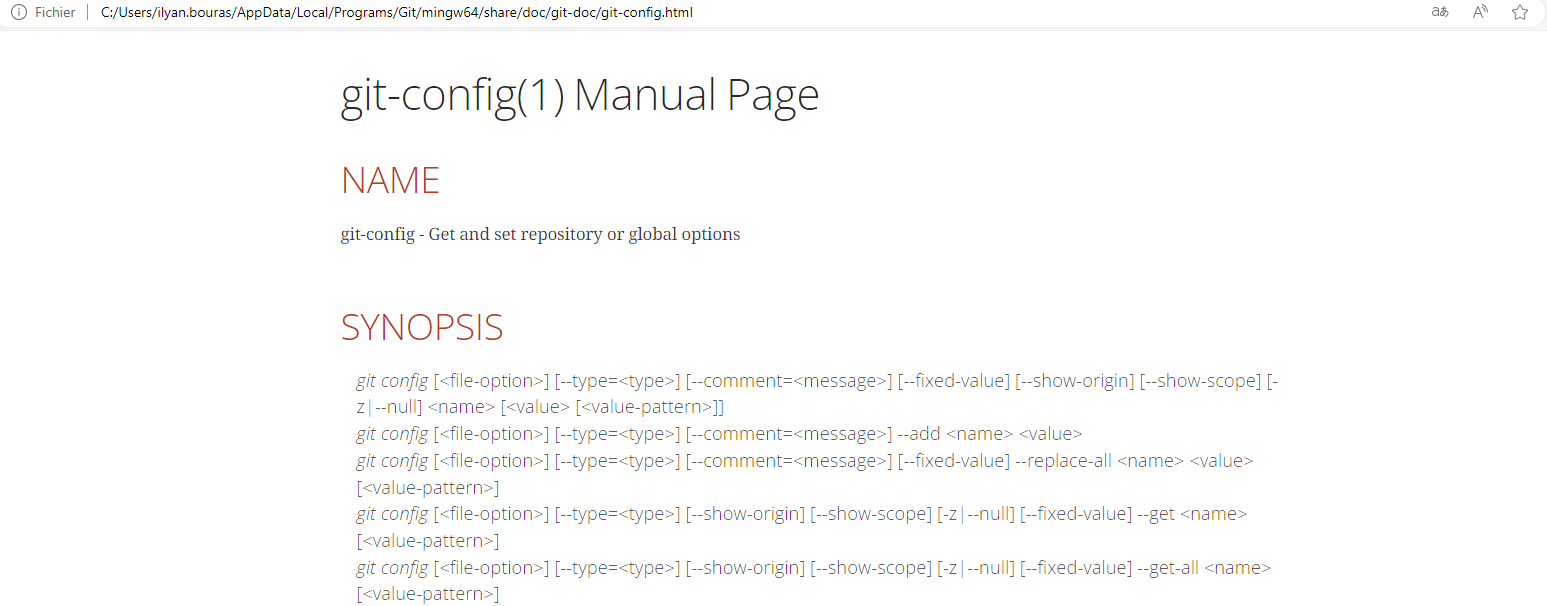
La version du git est : 2.45.1

Vous disposez d’une aide distante complète (https://git-scm.com/book/fr/v2), mais également d’une aide locale détaillée accessible par les commandes **git help <commande>** et **git <commande> --help**

Lancez par exemple la commande **git config --help**

Résultat :

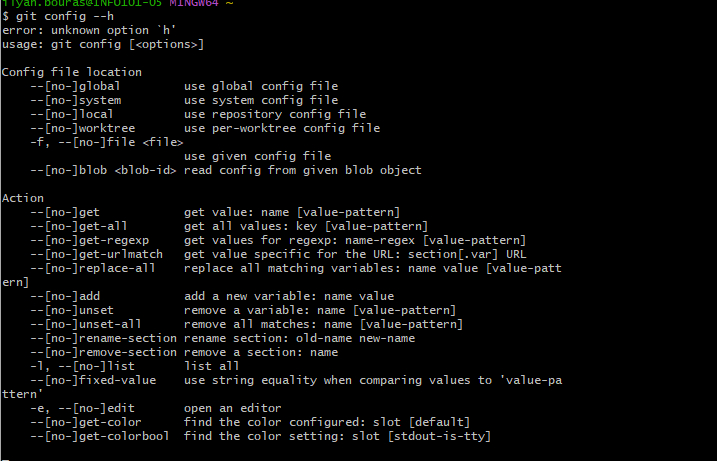
En exécutant la commande git config –help nous ouvrons une page concernant cela



Lancez ensuite la commande **git config -h**

Résultat et différence par rapport aux cas précédents :

Nous pouvons gra^che à l’option ---h avpoir les option de la commande config



Maintenant que Git est installé, vous pouvez personnaliser votre environnement Git. Vous ne devriez avoir à réaliser ces réglages qu’une seule fois ; ils persisteront lors des mises à jour. Cependant, si nécessaire, vous pourrez les changer à tout instant en relançant les mêmes commandes.

La première chose à faire après l’installation de Git est de renseigner votre nom et votre adresse de courriel. Il est important (surtout en équipe) de savoir qui a fait quoi, tapez alors les commandes suivantes :

* **git config --global user.name "Votre Nom"**
* **git config --global user.email votre.nom@email.com**

Puis, finissez la première configuration de Git par les commandes suivantes  :

* **git config --global color.diff auto**
* **git config --global color.status auto**
* **git config --global color.branch auto**
* **git config --global init.defaultBranch main**
* **git config --global core.editor nano**
* **git config --global merge.tool vimdiff**
* **git config --global push.default simple**

En vous aidant de la documentation, précisez le rôle de ces commandes :

1. cette commande sert à différencier par la couleur les parties des lignes dans le terminal.
2. Le fichier modifier sera de couleur différentete
3. La color.branch sert à afficher la branch sur la quel on travaille dans une couleur différenteen fonction de la branch
4. La commande git braLors de la création de la branch, l’itialition de la branche main
5. Définie nano comme l’éditeur de texte par défault.
6. Logiciel utilisait pour montrer des différente
7. GIT Push met à jour des fichiers distance stcoker par example sur un dépôt

Enfin, vérifiez que ces commandes ont bien été prises en compte, par :

* **git config --list**

## Création d’un dépôt Git pour le site web

Depuis la console Git Bash, identifiez le dossier actuellement en cours, par la commande Bash **pwd**

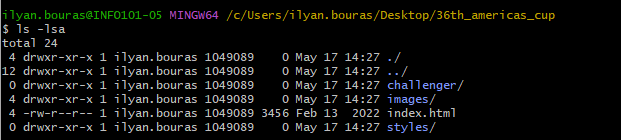
Résultat :



Nous sommes dans le répertoire /z/

Déplacez-vous dans le dossier **36th\_americas\_cup** et lister de manière détaillée son contenu.

Reportez les commandes utilisées :



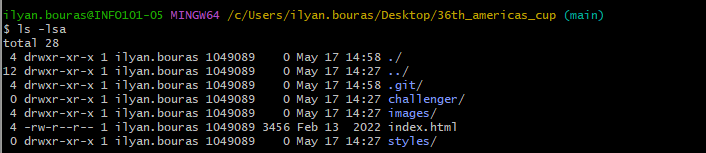
Dans le dossier **36th\_americas\_cup**, initialisez le dépôt Git par la commande **git init**

Que signifie la réponse de cette commande ? Et quel dossier caché a été créé ?

La commande git init sert créer un répertoire servant à un un dépôt GIT vide.



Reportez la commande Bash qui permet de vérifier la présence de ce dossier caché :



En entrant la commande ls –lsa nous permet de connaitre le répertoire caché git

Vous remarquerez aussi le terme **main** entre parenthèse et en bleu. Il s’agit de la branche de dépôt qui est actuellement utilisée. Nous y reviendrons ultérieurement.

Regardez l'état du dépôt par la commande **git status**

Vous devriez obtenir ceci :

$ git status

On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

challenger/

images/

index.html

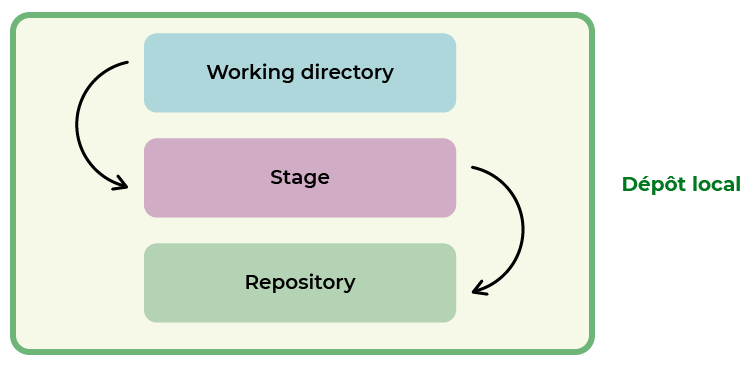
styles/

nothing added to commit but untracked files present (use "git add" to track)

Ici, **git status** nous informe que le projet possède des fichiers et dossiers qui ne sont pas **suivis** ("**untracked**"), qu’aucun fichier/dossier n’a été **validé** [on dit également **consigné**] ("**commit**") pour le moment ni ajouté ("**nothing added**") pour validation. La commande git statuts nous informe également sur la branche sur laquelle on se trouve ("**main**", dans le cas présent).

Vous pouvez à tout moment utiliser cette commande **pour voir où vous en êtes.** Connaissant l'état courant de vos manipulations, cette commande donne des indications contextualisées sur ce que vous pouvez (ou devez) faire. Ici par exemple, elle vous invite à ajouter ces fichiers/dossiers dans l'index (cf. ci-dessous) avant leur validation.

Les états des fichiers sont liés à des zones de travail dans Git. En fonction de son état, un fichier va pouvoir apparaitre dans telle ou telle zone de travail. Tout projet Git est composé de trois sections : le répertoire de travail (**Working directory**), la zone d’index (**staging area** ou simplement **Stage**) et le répertoire Git (**Repository**). L’ensemble constituant le dépôt local.



**add**

**commit**

L’étape suivante consiste donc à indexer nos fichiers (dans le jargon informatique, on dit parfois que l’on **stage les fichiers**) afin qu’ils puissent ensuite être validés, c’est-à-dire ajoutés au premier historique de version du projet.

warningPour indexer l’ensemble du projet courant, lancez la commande **git add .**

Il faut bien écrire le point en fin de commande !

Regardez de nouveau l'état du répertoire de travail et de la zone d’index par la commande **git status**

Vous devriez obtenir ceci :

$ git status

On branch main

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: challenger/luna\_rossa.html

new file: images/36th.ico

new file: images/AC75\_Defiant1.jpg

new file: images/AC75\_Defiant2.jpg

. . .

Cela indique que les fichiers sont prêts à être ajoutés au premier historique de version du projet et que l’on peut revenir en arrière si nécessaire par la commande **git rm**

Maintenant que les fichiers sont indexés, vous pouvez créer une version, c'est-à-dire archiver le projet web en l’état. Il suffit de lancer la commande :

**git commit -m "Premiere version du projet"**

**-m** (comme message) est ce qu'on appelle un argument, qui est ajouté à la commande principale. Il permet de définir un message particulier rattaché au commit effectué. Si vous n’utilisez pas cet argument, la commande **git commit** ouvrira un éditeur de texte (en l’occurrence nano – cf. Configuration) dans lequel vous pourrez saisir le message de commit.

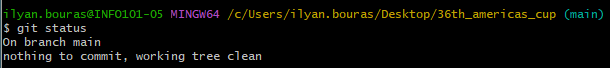
Le message de description est très important pour retrouver le fil des **commits**, et revenir sur un **commit** en particulier.

**Ne le négligez pas !**

Il ne serait pas pratique de nommer des **commits** d’un fichier du type "version 1", "version 2", "version 2.1", car vous seriez alors obligé de lire le contenu du fichier pour connaître les modifications réalisées. Avec un message clair, vous pouvez tout de suite identifier les modifications effectuées sur le fichier.

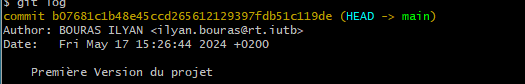
Regardez de nouveau l'état du répertoire de travail et de la zone d’index.

Qu’obtenez-vous ?



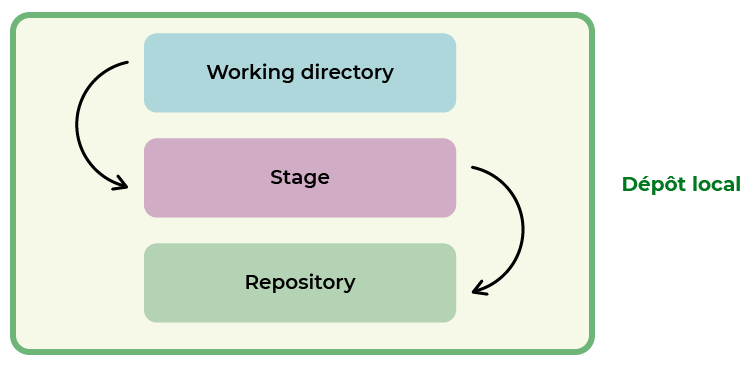
Enfin, vérifiez les informations sur la version du projet par la commande **git log**

Quelles sont les informations qui apparaissent actuellement :



Lorsque nous utilisons la fonctionnalité git log, nous pouvons observer la date et le message du git commit

Alors que **git status** vous permet d'inspecter le répertoire de travail et la section d’index, **git log** fonctionne uniquement sur la version "commitée" (zone Repository).



**add**

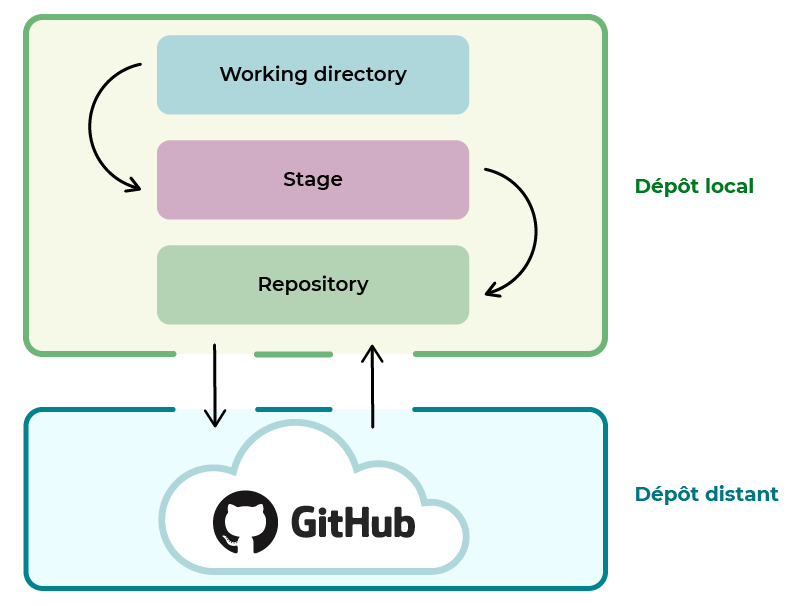
**status**

**commit**

**log**

## Création d’un dépôt distant sur GitHub pour le site web

Maintenant que la version du projet est réalisée sur le dépôt local, vous allez publier une copie sur votre compte GitHub. On dit que l’on va pousser ("**push**") ce dépôt local vers le dépôt distant.



**add**

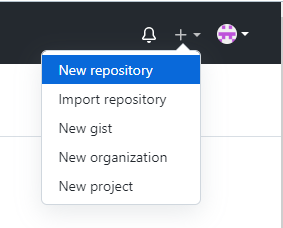
**status**

**log**

**commit**

**push**

Connectez-vous à votre compte GitHub.

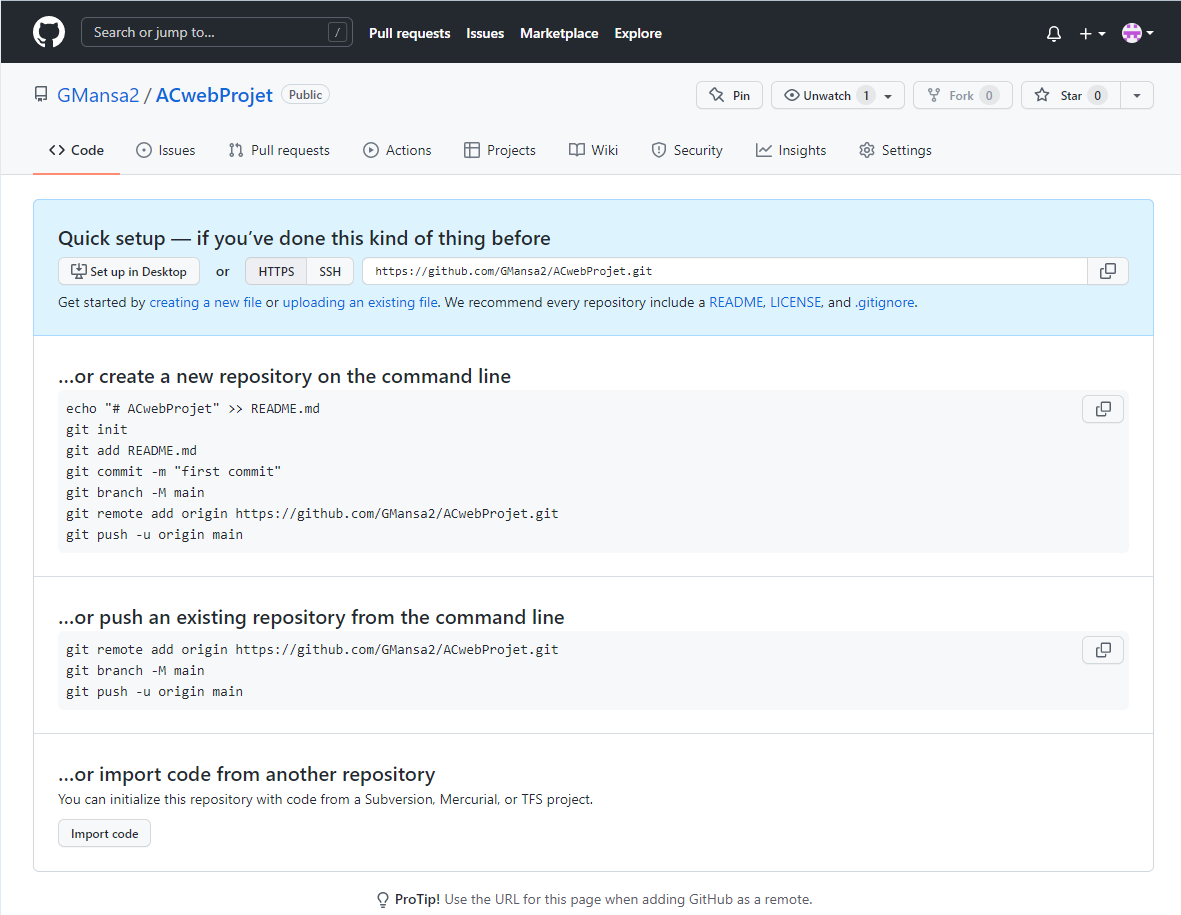


Créez un dépôt distant en cliquant   
sur **New repository**

Donnez le nom **ACwebProjet**   
à votre dépôt ainsi que la description   
**Site web 36th America's Cup**

Cliquez ensuite sur le bouton   
**Create repository**

Vous devriez avoir un écran comme celui-ci :



Cet écran vous donne les différentes commandes utiles en fonction de l’avancée de votre projet de "versioning".

Maintenant, retournez sur Git Bash car il faut indiquer à Git de créer un lien vers le dépôt géré par le serveur GitHub. Suivez pour cela l'instruction que vous donne GitHub. Elle ressemble à :

**git remote add origin https://github.com/votre\_nom/ACwebProjet.git**

Cette commande crée en local un lien ("**remote add**") nommé '**origin**' vers l'URL du dépôt stocké sur le serveur GitHub.

Le nom '**origin**' est celui donné usuellement, vous pouvez en choisir un autre si vraiment vous le désirez. Vous pouvez vérifier les liens distants enregistrés, avec la commande :

**git remote -v**

Pour envoyer la version actuelle de dépôt local sur le dépôt distant, utilisez :

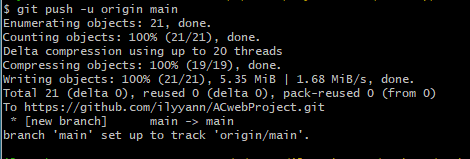
**git push -u origin main**

Précisez ce qui se passe alors :

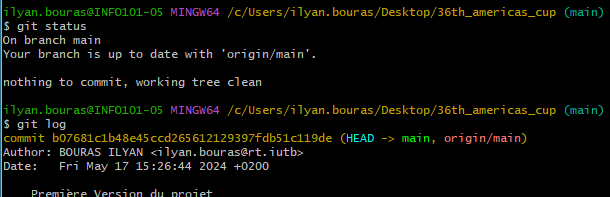
Cette commande signifie : synchroniser le dépôt distant dont le nom est '**origin**' avec mon dépôt local dont la tête est '**main**'. Le dépôt distant étant pour l'instant vide, tout votre dépôt local va être recopié sur le serveur.

Après l’exécution de la commande, précisez le nombre de fichiers transmis et à quelle vitesse :

Nous pouvons observer que la vitesse des fichier tr



Quelles sont les informations produites par les commandes **git status** et **git log** ?

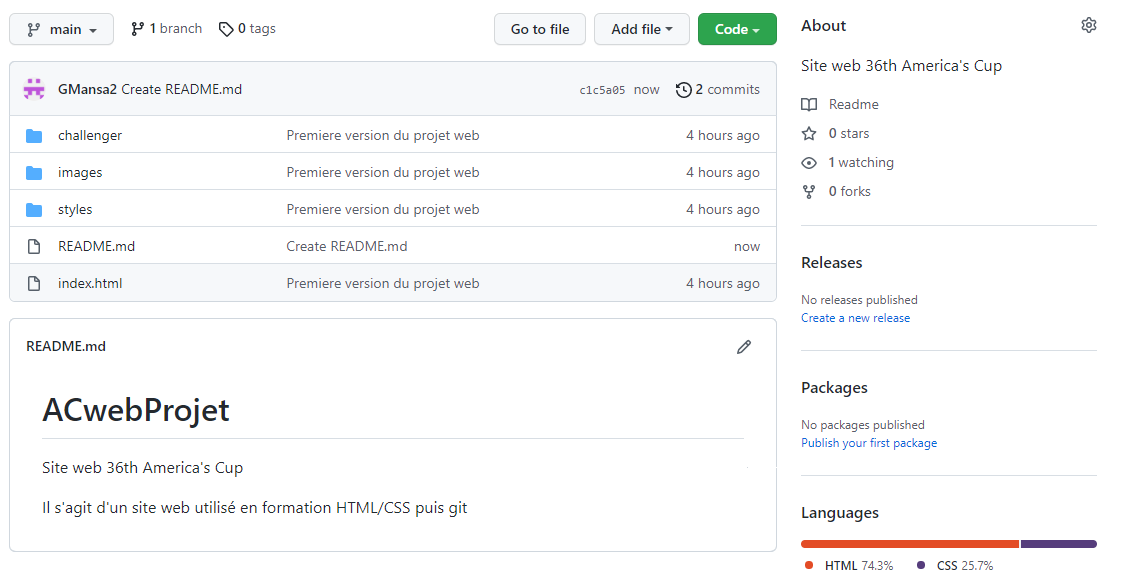


La commande git status a été mis a jour

La commande git log a ^rot e, compte la modicfication doriginie

De plus, vérifiez sur l'interface Web du serveur GitHub que votre projet contient bien votre projet web au complet. Vous devez retrouver le message précédemment fourni à la création du dépôt local. Vous pouvez également constater que GitHub a réalisé une analyse des types de fichiers utilisés. Enfin, cliquez sur **Add a README** pour créer un descriptif de votre projet.

Au final, vous devriez obtenir ceci :



Maintenant que votre dépôt est enregistré sur le serveur GitHub, il est possible à tous les membres de votre projet d'en créer une copie locale, un clone… Pour le moment, fermez Git Bash.

## Bilan

Globalement, un gestionnaire de versions permet de conserver un historique, revenir en arrière en cas de problème, collaborer, et suivre les modifications de chaque fichier.

Git gère les versions de vos travaux localement à travers 3 zones majeures :

* le répertoire de travail(**Working directory**) qui désigne l'arborescence de vos fichiers, c'est-à-dire tous vos fichiers et dossiers qui constituent votre projet initial
* l’index (**Stage**)quidésigne tous les fichiers que vous souhaitez voir apparaître dans votre prochain historique du projet (c’est-à-dire la prochaine version du projet). Pour réaliser cela, la principale commande est **git add**
* le dépôt (**Repository**) qui contient l'ensemble de vos actions (configurations, commits...). Pour créer une nouvelle version, la principale commande est **git commit**

Un dépôt distant (GitHub, par exemple) permet de stocker une copie de votre dépôt local dans le cloud, principalement grâce à la commande **git push**. Cela permet de travailler en équipe et/ou de contribuer à des projets open source déjà existants.

# Récupération d’une version et gestion avancée des versions

## Clonage d’un dépôt

Imaginons que vous avez accidentellement effacé localement votre projet ou que vous êtes un nouveau collaborateur d’un projet existant. Dans les 2 cas, vous devez récupérer le dépôt distant hébergé sur le serveur GitHub.

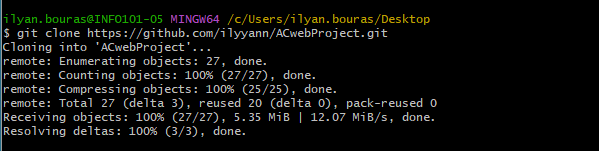
Commencez par mettre à la poubelle votre dossier **36th\_americas\_cup**!

Lancez Git Bash et placez-vous sur votre Bureau Windows. Pour cloner le projet depuis votre dépôt distant, utilisez la commande :

**git clone https://github.com/votre\_nom/ACwebProjet.git**

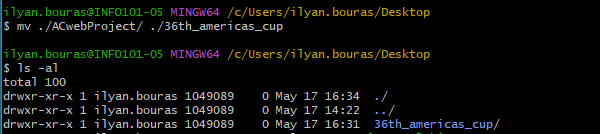
Après l’exécution de la commande, précisez le nombre de fichiers reçus et à quelle vitesse :

Il y a 27 fichiers reçut à une vitesse de 12.Mib/s



Une copie du dépôt distant **ACwebProjet** du serveur GitHub est alors créée sur votre Bureau Windows dans le dossier **ACwebProjet** (par défaut, Git crée un répertoire en réutilisant le nom du dépôt mais sans le suffixe .git). Par la commande **ls -al** vérifiez que le dossier est bien présent sur votre Bureau Windows et renommez-le **36th\_americas\_cup** par la commande Bash **mv**.

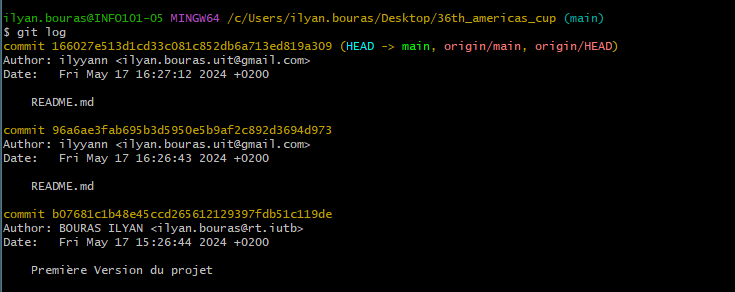
Reportez la commande complète :



Grâce à la commande mv nous pouvons modifier et renommer le dossier

Entrez dans le répertoire **36th\_americas\_cup** et vérifiez avec **git log** que tout l'historique est bien présent sur votre machine.

Combien de version (commit) apparaissent et quels sont les messages associés ?



Nous pouvons observer que des éléments ont été rajouter notament le fichier README.md

Vérifiez qu'un lien, nommé par défaut '**origin**', vers le dépôt distant a été créé, par la commande **git remote -v**

Voilà, votre projet local est sain et sauf. Vous pouvez vous en convaincre en utilisant l’explorateur de fichiers Windows pour constater que les fichiers/dossiers sont bien là !

Notez également, la présence du fichier texte README.md (que vous aviez créé sur GitHub).

Ouvrez le fichier avec Notepad++ et précisez ce qu’il contient :

Il contient lers modification zpporter dans la description faites sur github

Ouvrez le fichier index.html dans un navigateur web. Le site web est-il opérationnel ?

Le site est bel et bien oérationnel



**Remarque** : il est aussi possible de réaliser un clonage de dépôt localement. Par cela, avec Git Bash, il faut se situer dans l’arborescence de fichiers au-dessus du projet courant, puis d’utiliser la commande **git clone 36th\_americas\_cup 36th\_americas\_cup2**

## Gestion de version

A ce stade du TP, il y a d’un côté le projet web contenant un ensemble de fichiers (html, jpg, css…) sur lesquels vous allez travailler ainsi qu’un dépôt Git qui sert à gérer les différentes versions de ce projet.

En continuant à travailler sur le projet web, vous allez être amené à ajouter, modifier, voire supprimer des fichiers. Vous allez indiquer tous ces changements à Git pour qu’il conserve un historique des versions auquel vous pourrez ensuite accéder pour revenir à un état précédent du projet (dans le cas où une modification entraine un bogue par exemple ou n’amène pas le résultat souhaité).

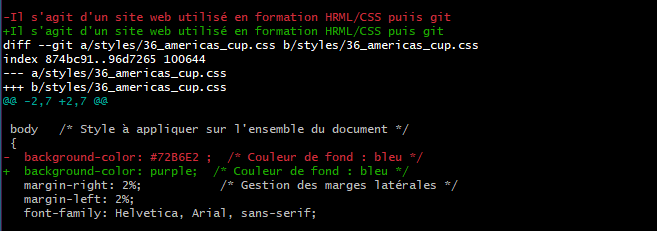
Ainsi, vous voulez modifier la couleur actuelle des titres du site web, de bleu à blanc… Modifiez en conséquence, le fichier de style soit à partir des outils Windows soit directement sous Git Bash en utilisant l’éditeur nano.

Reportez votre modification et vérifiez par un navigateur web que cette modification est bien réalisée :



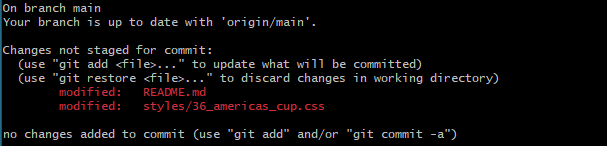
Exécutez la commande **git diff** et interprétez son compte-rendu :

Il y a deux lignes une coloré en rouge (ancienne version) et l’autre ligne en vert(version actuelle)



Lancez la commande **git status**. Que constatez-vous ?

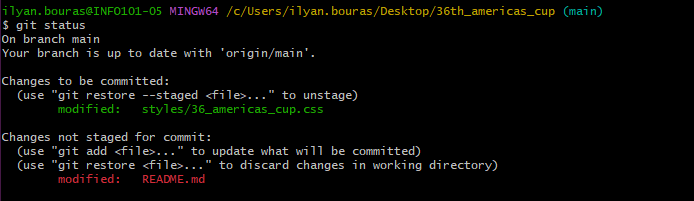
Nous pouvons voir les modifications grâce la commande git status



Indexez la feuille de style (reportez ci-dessous la commande effectuée), puis exécutez la commande **git status**. Que constatez-vous ?

Commande : git add styles/36\_americas\_cup/css

Résultat de **git status** :

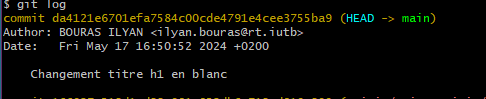


Enregistrez dans le Repository cette modification de version de la feuille de style, en précisant la nature de la modification "Changement titre h1 en blanc" (reportez ci-dessous la commande effectuée), puis exécutez la commande **git log** et **git log --oneline**.

Que constatez-vous ?

Commande : git commir « Changment titre h1 en blanc

Résultat de **git log** :

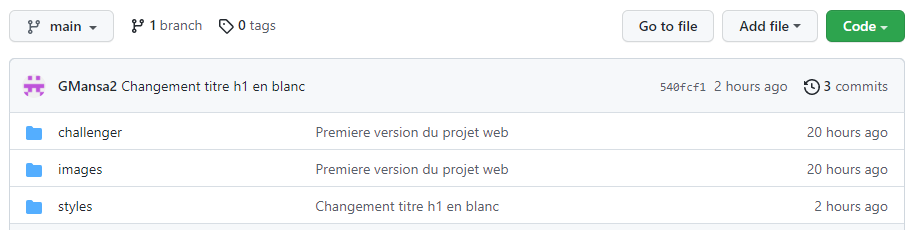


Enfin mettez à jour votre dépôt distant GitHub et reportez ci-dessous la commande effectuée :

Commande :

Git push

Revenez sur votre 'repository' **ACwebProjet** sous GitHub. Vous devriez obtenir ce type d’affichage avec ces changements :



Cliquez sur Changement titre h1 en blanc (à côté de votre login : zone bleue). Que constatez-vous ?

Les modification effectuer precedement sont surligné

Supprimez (par erreur !) la feuille de style soit à partir de l’explorateur Windows soit à partir de Git Bash par la commande : **rm 36\_americas\_cup.css**

Actualisez dans un navigateur web la page index.html par exemple. Que constatez-vous ?

Nous pouvons observer que le style de la page a disparut



Horreur ! Vous venez de perdre 5h de travail !

Que nenni, il existe une version dans le dépôt local que l’on peut utiliser !

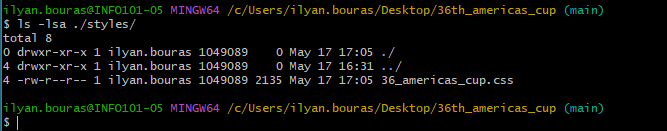
D’autre part, confirmez par la commande **git status** que Git a bien vu ce changement. Que vous propose-t-il comme action possible ?

Il nous propose l’option git restore 

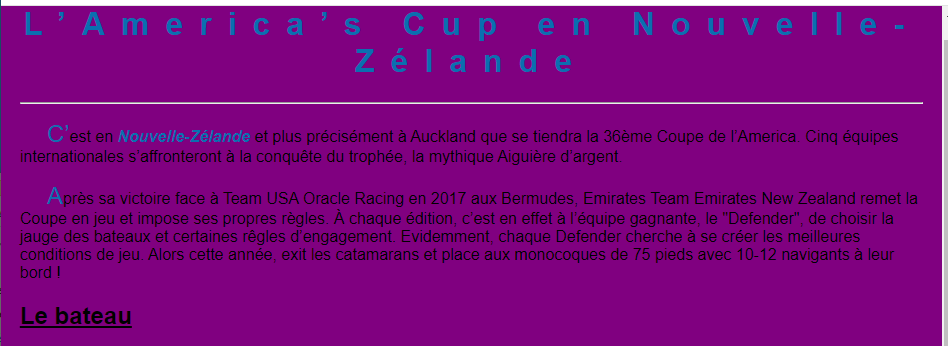
Cette restauration peut se réaliser avec la commande générale **git checkout -- nom-du-fichier** ou la nouvelle commande spécialisée **git restore**

Exécutez les commandes **git restore 36\_americas\_cup.css**, **git status** et **ls -al**. Que constatez-vous ? Et au niveau de la page index.html dans le navigateur ?

Ls –lsa



Concernant la page html



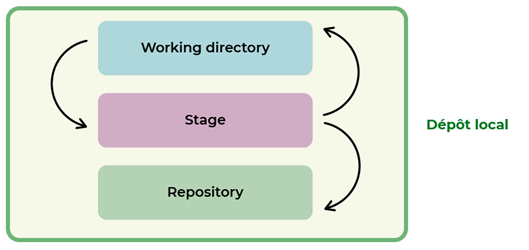
Tout est rentré dans l’ordre, ouf !

Cependant, Git avait proposé une autre action possible…

En effet, Git ne peut pas savoir si cette suppression de fichier était involontaire ou pas. Or, si cette suppression était volontaire, il faudrait mettre à jour l’index et le repository, pour que Git ne fasse plus le suivi de version de ce fichier, par les commandes :

**git rm 36\_americas\_cup.css**  et

**git commit -m "suppression de la feuille de style"** par exemple.



**add**

**commit**

**restore**

## Gestion de l’historique

Très bien, on vient de récupérer la dernière version de la feuille de style… mais si on veut retrouver la feuille d’antan ?

En effet, il existe plusieurs versions de la feuille de style !

Pour s’en assurer, exécutez la commande **git log --oneline**

Vous devriez obtenir quelque chose dans ce style :

**User@PC-IUT1 MINGW64 ~/Desktop/36th\_americas\_cup/styles (main)**

**$ git log --oneline**

**540fcf1 (HEAD -> main, origin/main) Changement titre h1 en blanc**

**c1c5a05 Create README.md**

**aeebabf Premiere version du projet web**

Cela représente les différents **commits** réalisés (donc les différentes versions du projet) et cela constitue **l’historique des versions**. Par rapport, à votre propre résultat, il y a très certainement des différences sur les identifiants des **commits** qui figurent en première colonne.

Cet historique constitue la branche **main** du projet. Suivant l’avancement du projet, les tests… plusieurs branches peuvent être par la suite créées.

Pour avoir plus de détail sur un **commit** particulier vous pouvez utiliser **git show** suivi de l'identifiant du commit (c1c5a05 dans le cas présenté, mais à adapter en fonction de votre propre historique).

Rappelez-vous de quoi il s’agissait ?

Lorsque nous faisons git show d’un commit, nous obtenons le résultat suivan t

Cela affche la ou les modifications

Vous aurez également noté la présence des éléments **HEAD** et **main**. Il s’agit de 2 pointeurs :

* le pointeur **HEAD** désigne le commit actif
* le pointeur **main** désigne toujours la tête de l'historique

Ces 2 pointeurs sont actuellement sur le même commit. C’est normal puisqu’il s’agit de la dernière version du projet qui est actuellement utilisée !

On peut illustrer cela par le schéma suivant :

**commit actif**

**Repository local**

Branche : **main**

**main**

**540fcf1**

**c1c5a05**

**aeebabf**

**HEAD**

Mais on peut déplacer le pointeur **HEAD** pour remonter dans le temps par la commande :

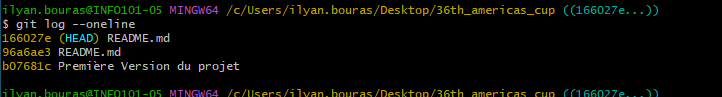
**git checkout**

Par exemple exécutez la commande : **git checkout c1c5a05** (pensez à adapter à votre cas)

Puis exécutez la commande **git log --oneline**

Que constatez-vous ?

Le dernier git commit que nous avons fait a été supprime



Dans votre navigateur web, rafraichissez la page index.html

Que constatez-vous et pourquoi ?



Nous avons supprimer les dernières modification, ce qui implique le changement de background

Le répertoire de travail est constitué des fichiers suivis dans leur version correspondant au commit référencé par **HEAD**

Par conséquent, **git checkout c1c5a05** va déplacer **HEAD** sur le commit **c1c5a05**, puis va faire en sorte que votre répertoire de travail soit constitué des fichiers dans leur version correspondant au nouveau **HEAD**.

Les fichiers sont donc modifiés, ajoutés, ou supprimés, pour que votre répertoire de travail soit dans l'état qui était le sien au moment du commit **c1c5a05**

Ce qui peut s’illustrer par le schéma suivant :

**main**

**540fcf1**

**c1c5a05**

**aeebabf**

**HEAD**

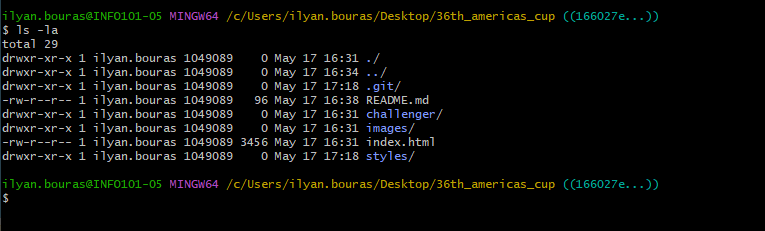
**commit actif**

**Repository local**

Branche : **main**

Faites **ls -al** sur le dossier **36th\_americas\_cup**.

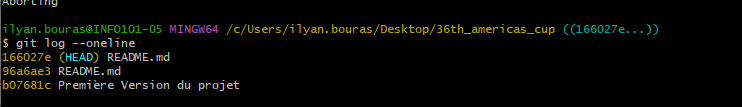
Relevez les caractéristiques du fichier README.md



Déplacez-vous d’un cran vers la gauche sur la branche **main** : **git checkout HEAD^**

Puis exécutez la commande **git log --oneline**

Que constatez-vous ?



Refaites **ls -al** sur le dossier **36th\_americas\_cup**.

Que constatez-vous et pourquoi ?

Mettez à jour le schéma suivant (en plaçant le pointeur **HEAD** et le commit actif) :

**commit actif**

**Repository local**

Branche : **main**

**540fcf1**

**c1c5a05**

**aeebabf**

**main**

**HEAD**

Pour revenir en tête de l’historique **main**, exécutez : **git checkout main**

Puis exécutez la commande **git log --oneline**

Que constatez-vous ?

Git chec

Dans votre navigateur web, rafraichissez la page index.html

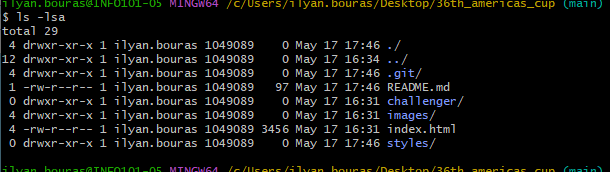
Refaites **ls -al** sur le dossier **36th\_americas\_cup**.

Que constatez-vous et pourquoi ?

Sur le navigateur :



La taille a augmenter



Vous avez pu ainsi naviguer dans l’historique du projet. Et vous vous êtes aperçu que la première couleur utilisée pour les titres h1 était le meilleur choix !

Mais comment faire alors pour uniquement reprendre la version antérieure de la feuille de style qui se trouve dans l’avant dernier commit ?

En étant dans le dossier **styles**, exécutez la commande :

**git checkout HEAD^ 36\_americas\_cup.css**

Que constatez-vous en rafraichissant la page index.html dans le navigateur ?



Nous pouvons constater que la version original est bel et bien réapparut

Voilà cette version vous semble parfaite !

Et il ne faut pas oublier de l’enregistrer dans l’historique local. Par contre, ce n’est pas utile de faire un **git add** car l’index est déjà mis à jour, comme le montre le résultat de la commande **git status**

Exécutez la commande **git commit -m "Retour à la feuille de style d’origine"**

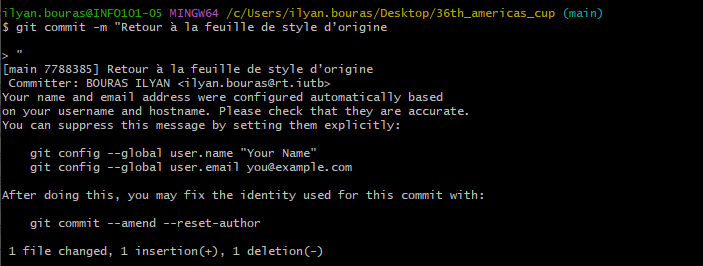
Puis **git status** et **git log --oneline**

Est-ce que tout vous parait correct ?

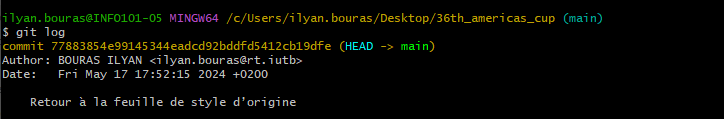
Quel est le nouvel identifiant de la dernière version ?

Où se trouvent les pointeurs **HEAD** et **main** ?

Nous avons bel et bien commit la version voulu



Grâce au gitlog, nous pouvons bel et bien voire que la verion a été commit



Grace aà la commande git log –oneline, nous pouvons constater que le dernier l’id du dernier commit est : 7788385 et

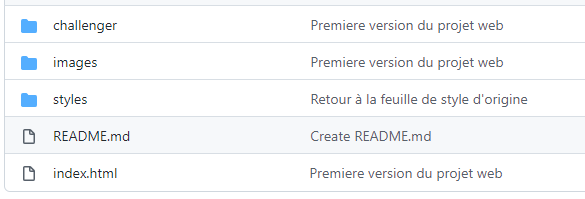


Mais à votre avis, pourquoi le pointeur **origin/main** est resté au même endroit ?

Poussez le dépôt local vers votre dépôt distant GitHub et vérifiez d’une part que localement tout est correct avec **git log --oneline** et d’autre part sur votre dépôt distant GitHub que cette version est correctement enregistrée. Enfin, mettez en évidence la modification réalisée sur la feuille de style.

Est-ce que tout est correct ? Sinon quels sont les problèmes ?

A partir du tableau de bord du dépôt distant sur GitHub, vous devez avoir un état du projet qui fait apparaitre ainsi les fichiers/dossiers :

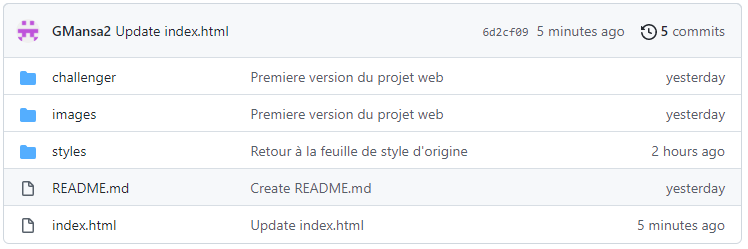


Vous allez maintenant cliquer sur le fichier **index.html** et modifier le titre :

<h1>L’America’s Cup 2021 en Nouvelle-Zélande</h1>

Enregistrer la modification.

En revenant sur le tableau de bord, vous devez voir qu’un nouveau message à automatiquement remplacé l’ancien et que le nombre de commit est passer à 5 :



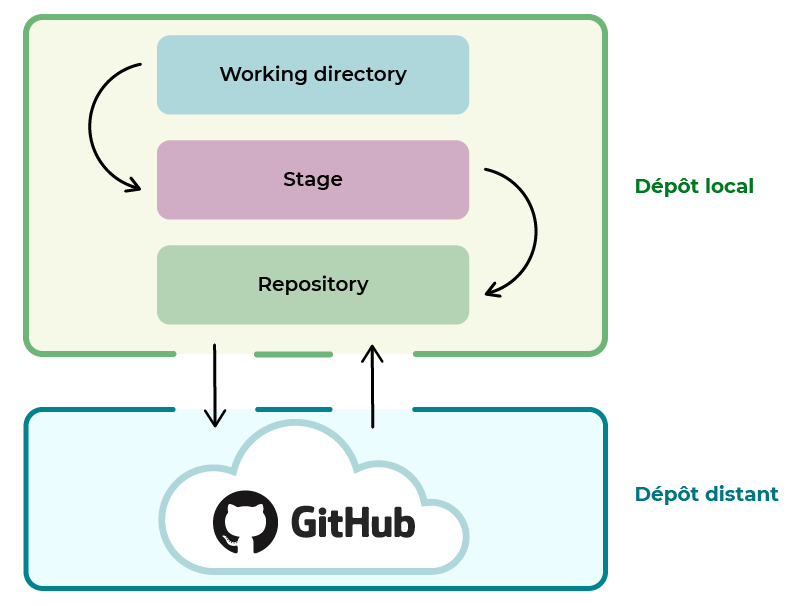
Mais alors votre dépôt local n’est plus à jour !

Vous pouvez facilement le vérifier par la commande **git log --oneline** mais aussi par la page web index.html dont le titre n’a pas changé !

Il vous faut mettre à jour votre dépôt local par la commande **git pull**

Vous pouvez ensuite vérifier par la commande **git log --oneline** qu’il y a bien 5 commits mais aussi que le titre de la page web index.html a bien été modifié !

On peut ainsi compléter le schéma :



**restore**

**add**

**status**

**log**

**commit**

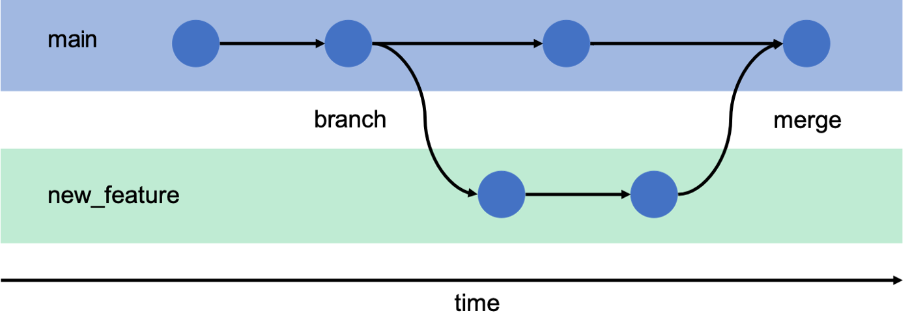
**push**

**pull**

## Exploitation des branches de l’historique

Le principal atout de Git est son système de branches :

* la branche principale (**main** ou ***master*** avant 2020) constituera la base stable du projet.
* l’objectif n’est donc pas de réaliser les modifications majeures directement sur cette branche, mais de les réaliser sur d’autres branches.
* et après divers tests, de les intégrer sur la branche principale.



Pour connaître les branches présentes du projet, il faut taper la ligne de commande :

**git branch**

Dans un premier temps, vous n’aurez que : **\* main**

Et c’est normal. L’astérisque signifie que c’est la branche sur laquelle vous vous situez, et que c’est sur celle-ci qu'actuellement les modifications sont réalisées.

Par contre, une modification majeure du site est prévue par la création de la page web présentant plus en détail le challenger **American Magic**. Il faut donc créer une branche que l’on appellera **magic** !

Exécutez la commande : **git branch magic**

Vérifiez que cette branche existe bien par la commande : **git branch**

Résultat ?

Comme vous pouvez le voir, l’astérisque est toujours sur la branche **main**. La branche **magic** est créée, mais nous n’avons pas encore basculé sur celle-ci.

Exécutez la commande **git log --oneline**

Que constatez-vous ?

Nous avons donc simplement créé une nouvelle branche, nommée **magic** et le pointeur **magic**, placé pour l'instant sur le dernier commit (celui pointé par **HEAD**).

On peut illustrer cela par le schéma suivant :

**commit actif**

**Repository local**

Branche : **main**

**cde218e**

**main**

**magic**

**HEAD**

**540fcf1**

**6d2cf09**

**aeebabf**

**c1c5a05**

Pour rappel, se placer sur une branche, c'est faire en sorte que le répertoire de travail soit en lien avec le dernier **commit** de cette branche. Ce lien est défini par la position du pointeur **HEAD** et la commande qui permet de manipuler ce pointeur est : **git checkout**

Pour se placer dans la branche **magic**, il suffit donc de déplacer **HEAD** sur le **commit** référencé par **magic** : **git checkout magic**

**Remarque** : Les deux actions précédentes (création et déplacement) auraient pu être réalisée ensemble par la commande : **git checkout -b magic**

Exécutez les commandes : **git branch ; git log --oneline**

Que constatez-vous ?

**HEAD** pointe maintenant effectivement sur **magic**. Ce qui signifie que tout nouveau **commit** sera ajouté après à la branche **magic** et non plus à la branche **main**.

Maintenant que la branche **magic** est active, vous allez pouvoir tester la mise en place de la page du site web dédiée au challenger **American Magic** !

A partir du serveur de fichiers **S2022** et plus particulièrement du dossier AmericanMagic (\\192.168.100.1\Public\BUT1\SAé23\AmericanMagic), copiez les 3 images (.png, .jpg et .ico) dans le dossier **36th\_americas\_cup\images** présent sur le Bureau Windows.

Faites de même avec le fichier american\_magic.html que vous placerez dans le dossier **36th\_americas\_cup\challenger** présent sur le Bureau Windows. Puis revenez dans la console Git Bash, vérifiez que vous êtes bien à la racine du projet et exécutez alors **git status**

Est-ce que Git s’est aperçu des changements et que propose-t-il ?

A partir de la racine du projet, exécutez les commandes **git add . ; git status**

Est-ce que tout vous semble correct ?

Si oui, exécutez la commande :

**git commit -m "Ajout des images et page web American Magic"**

Exécutez de nouveau **git status** et si tout est correct exécutez **git branch ; git log --oneline**

Interprétez l’affichage produit :

Mettez à jour par le schéma suivant :

branche **main**

**cde218e**

**main**

**540fcf1**

**6d2cf09**

**aeebabf**

**c1c5a05**

branche **magic**

**?????**

Eléments à placer :

**magic**

**HEAD**

Dans un navigateur web, ouvrez directement la page **american\_magic.html** depuis le dossier **36th\_americas\_cup\challenger**. Vérifiez que le style et les images cliquables sont corrects : la page web dédiée à American Magic est bien présente !

De même, ouvrez la page **index.html** depuis le dossier **36th\_americas\_cup**. Ne manque-t-il pas un lien hypertexte pour accéder à la page web d’American Magic, comme c’est le cas pour la page web dédiée à Luna Rossa ?

Corrigez en conséquence la page **index.html** (vous pouvez le faire sous Windows avec Notepad++ ou sous la console Git Bash avec l’éditeur nano) et vérifiez dans le navigateur web que maintenant la page d’American Magic est accessible depuis la page **index.html**.

Exécutez la commande **git status**

Est-ce que tout vous semble correct ?

Il faut alors mettre cette modification dans l’historique des versions de la branche **magic**, avec le message **"Ajout hyperlien vers American Magic"**

Reportez ici vos commandes :

Puis exécutez : **git status ; git branch ; git log --oneline**

Interprétez l’affichage produit :

Mettez à jour par le schéma suivant :

branche **main**

**main**

**cde218e**

**540fcf1**

**6d2cf09**

**aeebabf**

**c1c5a05**

branche **magic**

**magic**

**?????**

**?????**

**HEAD**

Eléments à placer :

Voilà, vous êtes satisfait des modifications apportées au projet du site web !

À présent, il faut intégrer l'évolution réalisée dans la branche **magic** à la branche principale **main**. Pour cela, vous devrez utiliser la commande **git merge**.

Mais attention, cette commande doit s'utiliser à partir de la branche dans laquelle nous voulons apporter les évolutions. Dans le cas présent, la commande s’effectuera donc dans la branche **main**. Pour y retourner, utilisez la commande : **git checkout main**

Mettez à jour (F5) la page **index.html** dans le navigateur (ou réouvrez là si vous l’aviez fermée). Avec l’explorateur de fichiers Windows, placez-vous dans le dossier **Challenger**…

Que constatez-vous !

**Mais pas de panique**, c’est normal puisque la version actuelle du projet se trouve sur la branche **main** et rassurez-vous la branche **magic** existe toujours !

Pour le constater, il suffit d’exécuter : **git status ; git branch ; git log --oneline**

Le schéma actuel est le suivant :

branche **main**

**HEAD**

**magic**

**?????**

**?????**

**main**

**cde218e**

**540fcf1**

**6d2cf09**

**aeebabf**

**c1c5a05**

branche **magic**

Maintenant, l’opération de fusion de la branche **magic** avec la branche **main** est réalisable, en exécutant : **git merge magic**

Combien de fichiers du site web ont été impactés ? Est-ce correct ?

Mettez à jour (F5) la page **index.html** dans le navigateur (ou réouvrez la si vous l’aviez fermée), accédez à la page web d’American Magic. Avec l’explorateur de fichiers Windows, placez-vous dans le dossier **Challenger**…

Que constatez-vous ! **It's a kind of Magic !**

Pour vous en convaincre, exécutez de nouveau : **git status ; git branch ; git log --oneline**

Mettez à jour par le schéma suivant :

branche **main**

**?????**

**cde218e**

**540fcf1**

**6d2cf09**

**aeebabf**

**c1c5a05**

**?????**

**HEAD**

Eléments à placer :

**main**

**magic**

En analysant le résultat des dernières commandes, n’y a-t-il pas un conseil qu’il faudrait suivre ?

Et oui, il ne reste plus qu’à mettre à jour le dépôt distant pour que tout soit à jour. Réalisez cette opération et vérifiez que le dépôt distant est bien à jour.

Reportez la (ou les) commande(s) utilisée(s) :

Exécutez un dernière fois : **git status ; git branch ; git log –oneline**

Que constatez-vous ? Où se trouve le pointeur **origin/main** ?

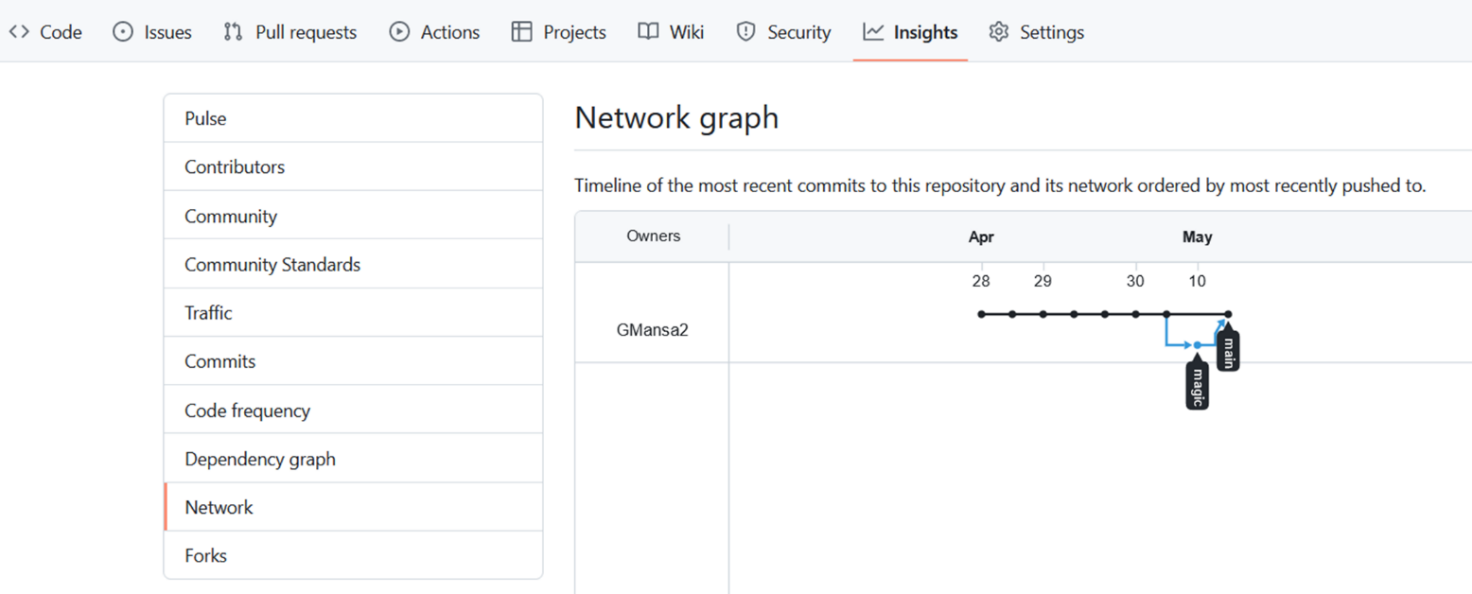
Dans cette partie de TP, vous avez vu :

* qu’une branche peut être considérée comme une “copie” d’un projet sur laquelle on peut opérer des modifications sans crainte pour le projet stable
* que la branche (**main** ou ***master*** avant 2020) est la branche principale d’un projet
* que la commande **git checkout** permet de basculer d’une branche à une autre
* que la commande **git merge** permet de fusionner deux branches

## Gestion et visualisation avancées des branches

La partie précédente vous a permis d’exploiter les branches d’une manière locale. Nous allons voit maintenant la gestion en locale et sur le dépôt distant GitHub des branches.

A partir de votre 'repository' **ACwebProjet** sous GitHub, affichez la frise temporelle (Insights / Network) actuelle. Vous devriez obtenir une frise semblable à celle-ci (à remplacer par la vôtre !) :



Commencez par créer localement une branche de développement, appelée **DevBranch**.

Reportez la commande utilisée :

Restez sur la branche **main** et modifiez la page **american\_magic.html** (vous pouvez le faire sous Windows avec Notepad++ ou sous la console Git Bash avec l’éditeur nano) comme suit :

<title>American Magic</title> **<!-- Branche main -->**

Indexez vos fichiers, vérifiez l'état du répertoire de travail et de la zone d’index.

Reportez les commandes **git** utilisées et les résultats obtenus :

Exécutez la commande : **git log –oneline**

Est-ce que tout vous semble correct ?

Si oui, exécutez la commande :

**git commit -m "American Magic Branche main"**

Exécutez de nouveau **git status** et si tout est correct exécutez **git branch ; git log --oneline**

Reportez et interprétez l’affichage produit :

Enfin envoyez la version actuelle de dépôt local sur le dépôt distant, en utilisant la commande :

**git push -u origin main**

Suite à cette commande, exécutez **git status ; git log --oneline** et interprétez l’affichage produit :

Basculez sur la branche **DevBranch** et modifiez la page **american\_magic.html** (vous pouvez le faire sous Windows avec Notepad++ ou sous la console Git Bash avec l’éditeur nano) comme suit :

<title>American Magic</title> **<!-- Branche DevBranch -->**

Indexez vos fichiers, vérifiez l'état du répertoire de travail et de la zone d’index.

Reportez les 3 commandes **git** utilisées et les résultats obtenus :

Exécutez la commande : **git log --oneline**

Est-ce que tout vous semble correct ?

Si oui, exécutez la commande :

**git commit -m "American Magic Branche DevBranch"**

Exécutez de nouveau **git status** et si tout est correct exécutez **git branch ; git log --oneline**

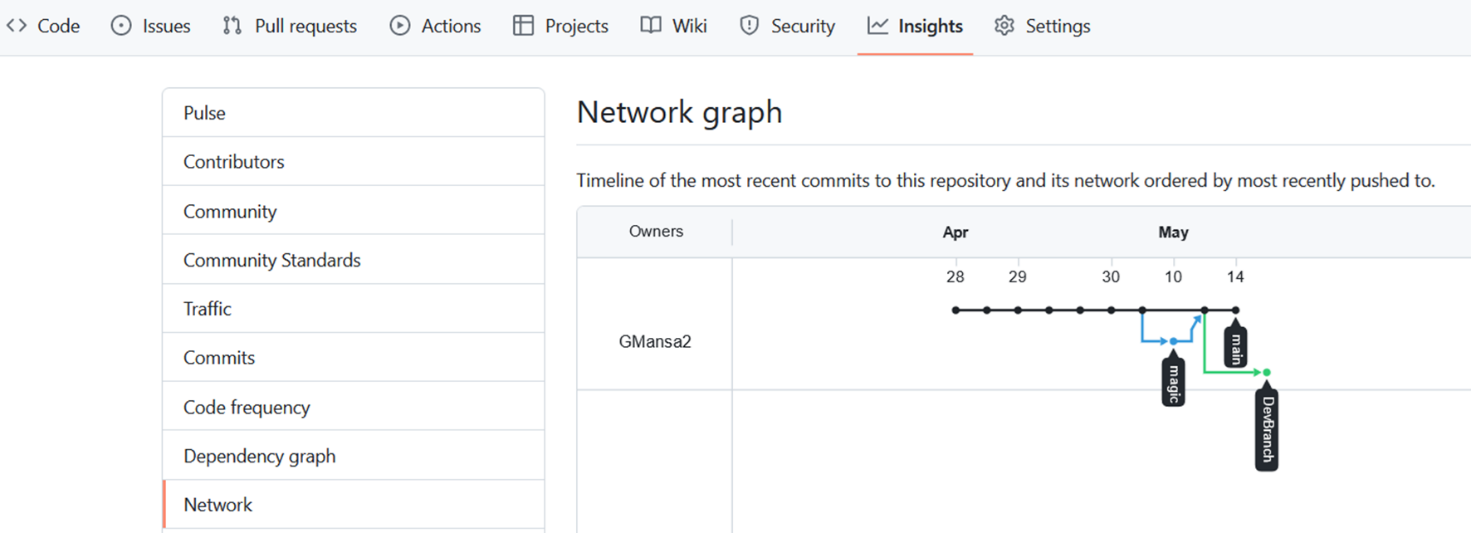
Interprétez l’affichage produit :

Enfin envoyez la version actuelle de dépôt local sur le dépôt distant (branche **DevBranch**), en utilisant la commande :

**git push -u origin DevBranch**

Reportez et interprétez l’affichage produit :

A partir de votre 'repository' **ACwebProjet** sous GitHub, affichez la frise temporelle (Insights / Network) actuelle. Vous devriez obtenir une frise semblable à celle-ci (à remplacer par la vôtre !) :



Cliquez sur le nœud présent sur la branche **DevBranch** !

Reportez et interprétez l’affichage produit :

Vous pourriez continuer votre développement du site web alternatif sur cette branche tout en laissant la branche principale dans son état actuel. Mais vous souhaitez pour le moment y mettre fin, tant localement que sur le dépôt distant GitHub !

Commencez par basculez sur la branche **main**.

Reportez la commande utilisée :

Vérifiez-le par la commande **git branch**

Reportez et interprétez l’affichage produit :

Supprimez localement la branche **DevBranch** par la commande **git branch -D DevBranch**

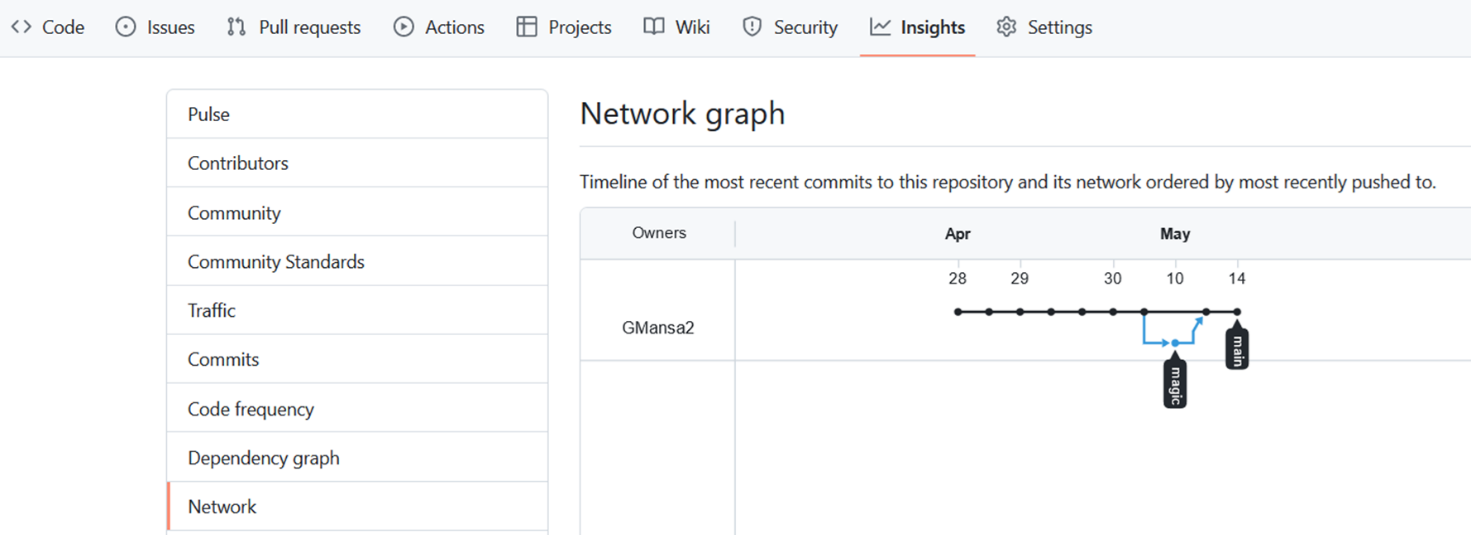
Reportez et interprétez l’affichage produit :

Supprimez sur le dépôt distant la branche **DevBranch** par la commande :

**git push origin --delete** **DevBranch**

Reportez et interprétez l’affichage produit :

A partir de votre 'repository' **ACwebProjet** sous GitHub, affichez la frise temporelle (Insights / Network) actuelle. Vous devriez obtenir une frise semblable à celle-ci (à remplacer par la vôtre !) :



Dans cette partie de TP, vous avez vu :

* que sur le dépôt distant GitHub il est facile de suivre l’évolution du projet, en particulier grâce à la frise temporelle
* qu’une branche particulière peut être déposée sur le dépôt distant par la commande **git push -u origin** **Branche**
* qu’une branche devenue obsolète peut être supprimée sur le dépôt local et sur le dépôt distant par les commandes **git branch -D Branche** et **git push origin --delete Branche**

# Formation complémentaire

**Au final**, à partir de Webetud ou du serveur de fichiers **S2022** (\\192.168.100.1\Public\BUT1\SAé23), approfondissez votre formation en analysant les fichiers **Git\_Formation.pdf** d’Isabelle BLASQUEZ et Git\_travailAplusieurs.pdf de Bruno MERMET.

# Aide-mémoire Git

## Initialisation d’un dépôt Git

* **git init** : initialise un nouveau dépôt Git dans le répertoire courant.
* **git clone URL** : clone un dépôt Git distant dans un nouveau répertoire local.

Exemple : **git clone https://github.com/user/repo.git**

* **git config --global user.name "NOM"** : configure le nom d’utilisateur pour tous les dépôts locaux.

Exemple : **git config --global user.name "John Doe"**

* **git config --global user.email "EMAIL"** : configure l’adresse e-mail pour tous les dépôts locaux (ne pas négliger cette instruction !).

Exemple : **git config --global user.email "johndoe@example.com"**

* **git add .** : ajoute tout le répertoire à l’index.
* **git add FILE** : ajoute un fichier modifié ou nouvellement créé à l’index.

Exemple : **git add file.txt**

* **git commit -m "MESSAGE"** : crée un instantané du code source actuel et le stocke dans l’historique des versions de Git.

Exemple : **git commit -m "Ajout d’un nouveau fichier"**

## Gestion des branches Git

* **git branch** : affiche la liste des branches locales.
* **git branch BRANCH** : crée une nouvelle branche à partir de la branche actuelle.

Exemple : **git branch feature**

* **git checkout BRANCH** : bascule vers une branche spécifique et met à jour le répertoire de travail.

Exemple : **git checkout feature**

* **git merge BRANCH** : fusionne une branche spécifique dans la branche actuelle.

Exemple : **git merge feature**

* **git branch -d BRANCH** : supprime une branche locale spécifique.

Exemple : **git branch -d feature**

* **git push origin BRANCH** : envoie une branche locale vers le dépôt distant.

Exemple : **git push origin feature**

## Gestion des commits Git

* **git log** : affiche l’historique des commits pour la branche actuelle.
* **git diff COMMIT1 COMMIT2** : affiche les différences entre deux commits spécifiques.

Exemple : **git diff abc123 def456**

* **git reset COMMIT** : annule tous les commits après le commit spécifié et conserve les modifications en tant que modifications non validées.

Exemple : **git reset abc123**

* **git revert COMMIT** : annule le commit spécifié en créant un nouveau commit qui annule les modifications apportées par le commit spécifié.

Exemple : **git revert abc123**

* **git cherry-pick COMMIT** : applique les modifications apportées par le commit spécifié à la branche actuelle.

Exemple : **git cherry-pick abc123**

* **git bisect** : utilise une recherche binaire pour trouver le commit qui a introduit un bogue.

## Gestion des conflits de fusion

* **git merge --abort** : annule une fusion en cours et restaure l’état précédent de la branche actuelle.

Exemple : **git merge --abort**

* Résoudre manuellement les conflits de fusion en éditant les fichiers en conflit et en utilisant les commandes **git add** et **git commit** pour marquer les conflits comme résolus.
* Utiliser des outils tiers tels que **KDiff3** ou **Meld** pour résoudre les conflits de fusion.

## Gestion des tags Git

* **git tag** : affiche la liste des tags existants pour la branche actuelle.
* **git tag TAG** : crée un nouveau tag pour le commit actuel.

Exemple : **git tag v1.0.0**

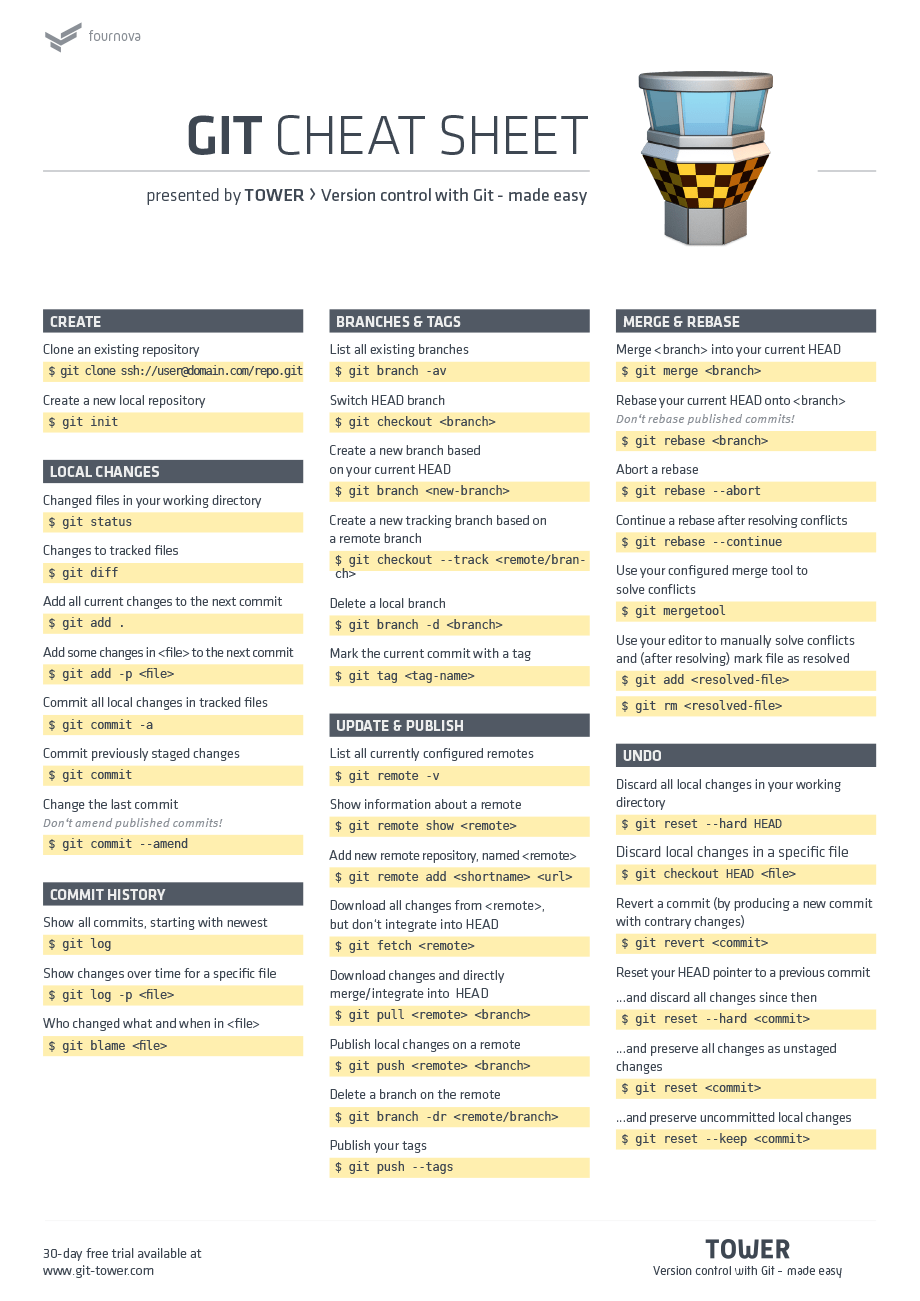
* **git show TAG** : affiche les informations sur un tag spécifique.

Exemple : **git show v1.0.0**

* **git push origin TAG** : envoie un tag local vers le dépôt distant.

Exemple : **git push origin v1.0.0**

* **git tag -d TAGNAME** : supprime le tag TAGNAME.



https://www.onasus.com/10-commandes-utiles-de-git-a-connaitre/