Gestion de code avec Git (FIEC30EM)

frank.buloup@univ-amu.fr

Aix Marseille Université
Faculté des Sciences du Sport
Master 2 IEAP - Facteurs Humains

BY NC SA

Aix Marseille Université

INSTITUT DES SCIENCES DU MOUVEMENT ETIENNE JULES MAREY

Gestion de code avec Git

1. Introduction
2. Utilisation & Installation de Git
3. Projet
4. Annexes

1. Introduction
   - Organisation & Objectifs du module
   - Le compte GitHub & les projets
   - Git ? C'est quoi ?
   - Petit historique des systèmes de contrôle de versions

Organisation

- Cours et TDs pour apprendre les bases de Git
- Un projet par équipe pour mettre en œuvre ces apprentissages
- Deux (à trois) étudiants par équipe

Volumes horaires

- Environ 4 heures de cours/TDs
- Environ 14 heures pour le projet

Evaluation

- 50% sur un CC (examen écrit 30mn)
- 50% sur le projet tout au long du module

À l'issue de cette formation, vous serez capable de :

1. Décrire les principes de fonctionnement d'un gestionnaire de versions distribué
2. Créer, initialiser et cloner un dépôt avec Git
3. Manipuler les commandes de Git pour gérer les fichiers et les branches
4. Résoudre les conflits de fusion
5. Mettre en œuvre un projet en mode collaboratif avec Git sur GitHub

Compte GitHub

- Login : MasterFHGit
- Password : MasterFHGit2025

Les projets

Sur la base d'un Arduino Uno R4 Wifi avec :

- une jauge de contrainte (2 postes)
- un potentiomètre
- une résistance à détection de force (2 postes)
- un capteur à ultrason (2 postes)
- un codeur optique

Git ? C'est quoi ?

Une idée ?

Documentation Git

Git n'est pas un serveur

- Git n'offre pas de fonctionnalité de gestion de projet complète (e.g. gestion de tâches, de bugs ou wiki)
- Github et GitLab sont deux exemples de solutions complètes

Github

GitLab

Git n'est pas un serveur

- Git n'offre pas de fonctionnalité de gestion de projet complète (e.g. gestion de tâches, de bugs ou wiki)
- Github et GitLab sont deux exemples de solutions complètes

Github **GitLab**

Git n'est pas un espace de stockage de fichiers type Dropbox

- Pas de sauvegarde automatique des fichiers
- Pas de gestion optimale des fichiers volumineux* ou binaires**

*Utilisation d'une extension spécifique
**Les fichiers binaires ne sont pas très lisibles (git diff !!!)

Git est un système de contrôle de version (Version Control System)

- Historique des versions de tous les fichiers
- Manipulation complète de l'historique possible
- Décentralisé (bientôt plus clair...)
- On trouve aussi : "Source Code Management System"

Git est un système de contrôle de version (Version Control System)

- Historique des versions de tous les fichiers
- Manipulation complète de l'historique possible
- Décentralisé (bientôt plus clair...)
- On trouve aussi : "Source Code Management System"

Pourquoi gérer des versions de fichiers ?

- Pour pouvoir revenir en arrière
- Pour comparer
- Pour expérimenter, tester des idées
- Pour travailler à plusieurs plus facilement

Vous pouvez gérer tous types de fichiers avec git !

Exemples de fichiers gérés par git

- Fichiers de code informatique
- Fichiers textes (e.g. .doc)
- Documentation de projets
- Fichiers binaires (e.g. images, sons) - Volumineux : git LFS*
- Fichiers de données - Volumineux : git LFS*
- etc.

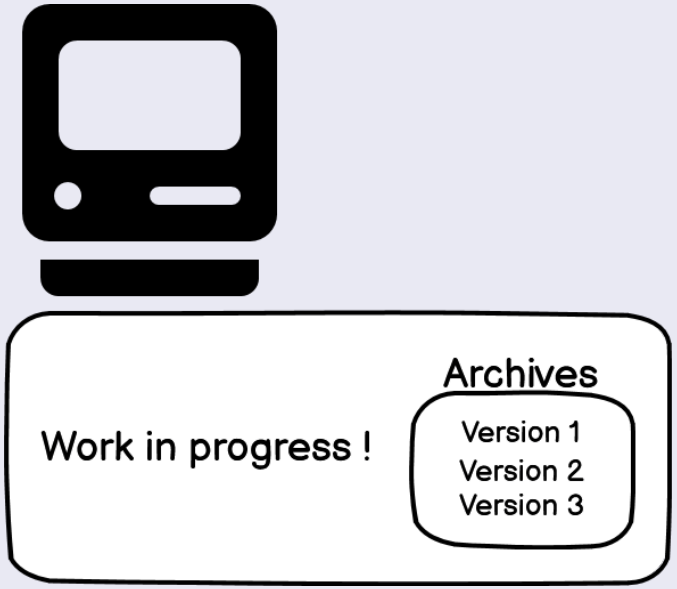
Gestion collaborative de projets

*Utilisation éventuelle de l'extension **Git Large File Storage**
Taille max. de 5GB sur Github (AWS Amazon S3)

Un peu d'histoire !

Local VCS - Avant... il y a longtemps !

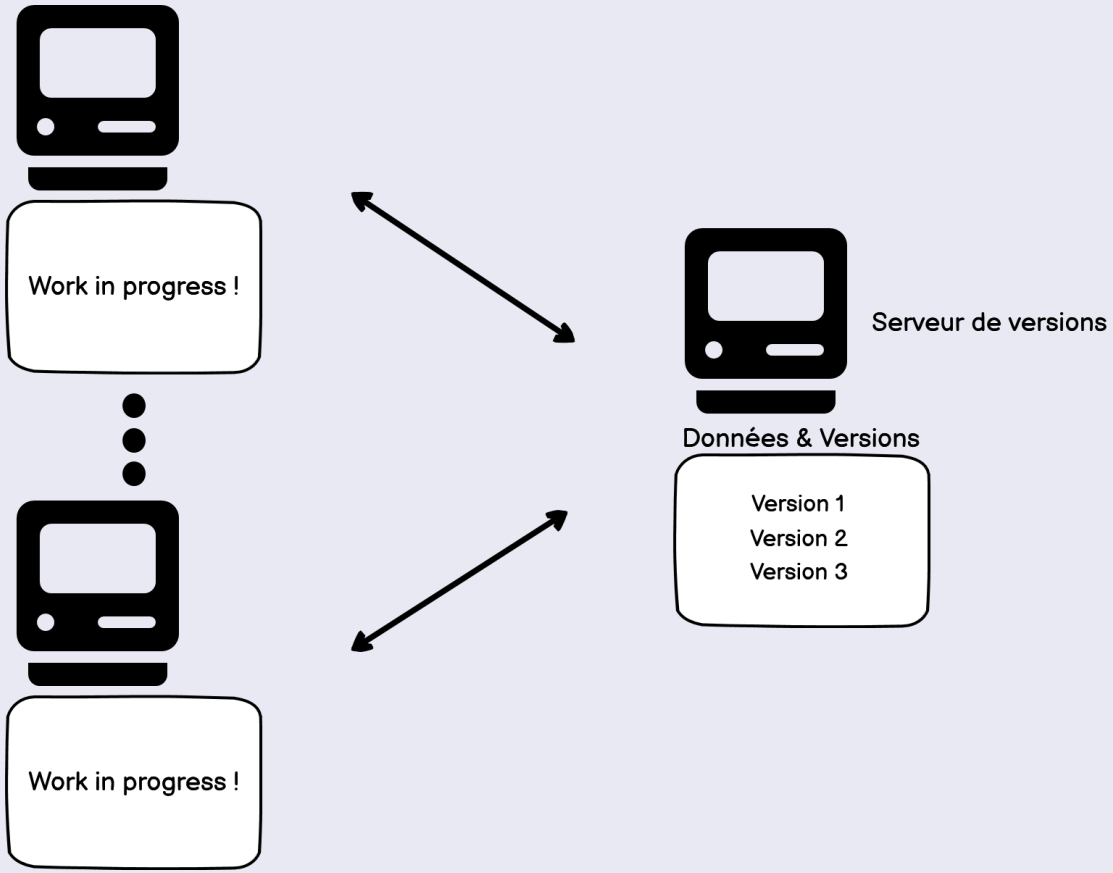
- Système D : horodatage, classement dossiers, compression etc.
- Pro : Revision Control System (1982, projet GNU*)



*GNU : GNU's Not Unix

Centralized VCS - Avant... un peux moins longtemps !

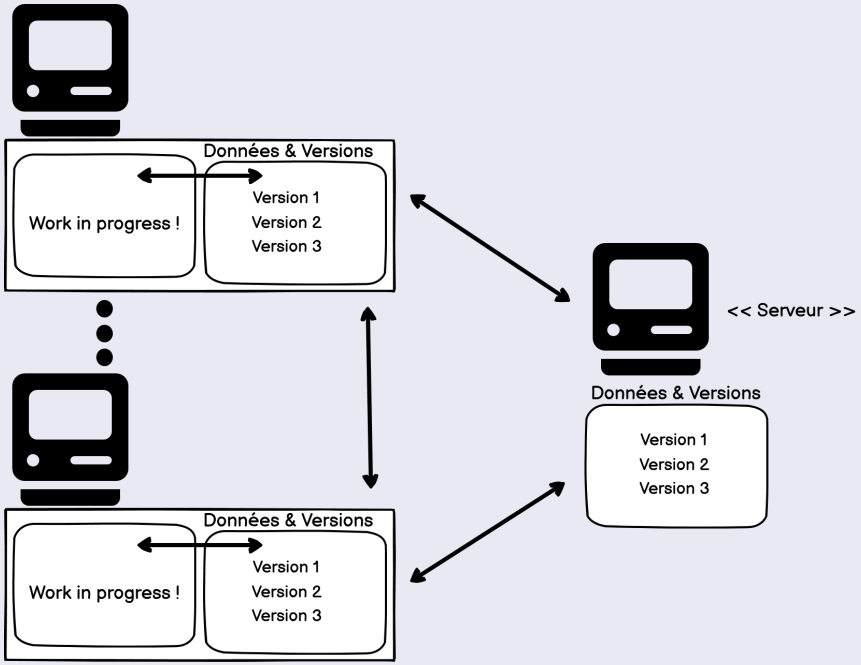
- Partage plus facile
- Connexion réseau obligatoire (approche Client-Serveur)



e.g. Concurrent Version System (CVS), Subversion (SVN)

Distributed VCS - Maintenant, depuis un moment (début 2000) !

- État serveur et réseau pas critique (approche Pair-à-Pair)
- On peut remonter un serveur à partir d'une machine
- Les opérations locales (la majorité) sont rapides
- Travail privé possible



e.g. Git, Mercurial, Darcs

L'origine de Git

- Créé en 2005 par **Linus TORVALDS**
- Pour le noyau Linux
- Activement maintenu par **Junio HAMANO**
- Popularisé par GitHub
- On prononce "guit" et pas "jit" !
- Git = Crétin (= Conna... !)

Git - Wikipedia

Les principales caractéristiques de Git

- Contrôle de version distribué (pair-à-pair)
- Gestion des branches de code (création, suppression, fusion)
- Opérations atomiques (Commits)
- Données de gestion stockée dans un répertoire .git

2 Utilisation & Installation de Git

- Les différents modes d'utilisation
- Utilisation sur GitHub
- Installation & utilisation en local - Les commandes de base
- Les zones de Git
- Annuler un commit : commande revert
- Modifier l'historique des commits : commande reset
- Fusionner et rebaser des branches - Résolution de conflits
- Collaboration

Comment utiliser Git ?

- En ligne de commande (Terminal & Git Bash pour Windows)
- Outils graphiques intégrés à Git : "git-gui" et "gitk"
- Avec une IHM dans un OS (GitKraken, SourceTree) : ici
- Intégré dans un EDI (plugin, package etc.)
- Sur internet : github, gitlab, bitbucket etc.

L'apprentissage en ligne de commande est le plus efficace !

Exercice I - Premier pas avec GitHub

1. Créer un compte sur GitHub
2. Créer un premier dépôt public dans GitHub nommé "FirstRepository"

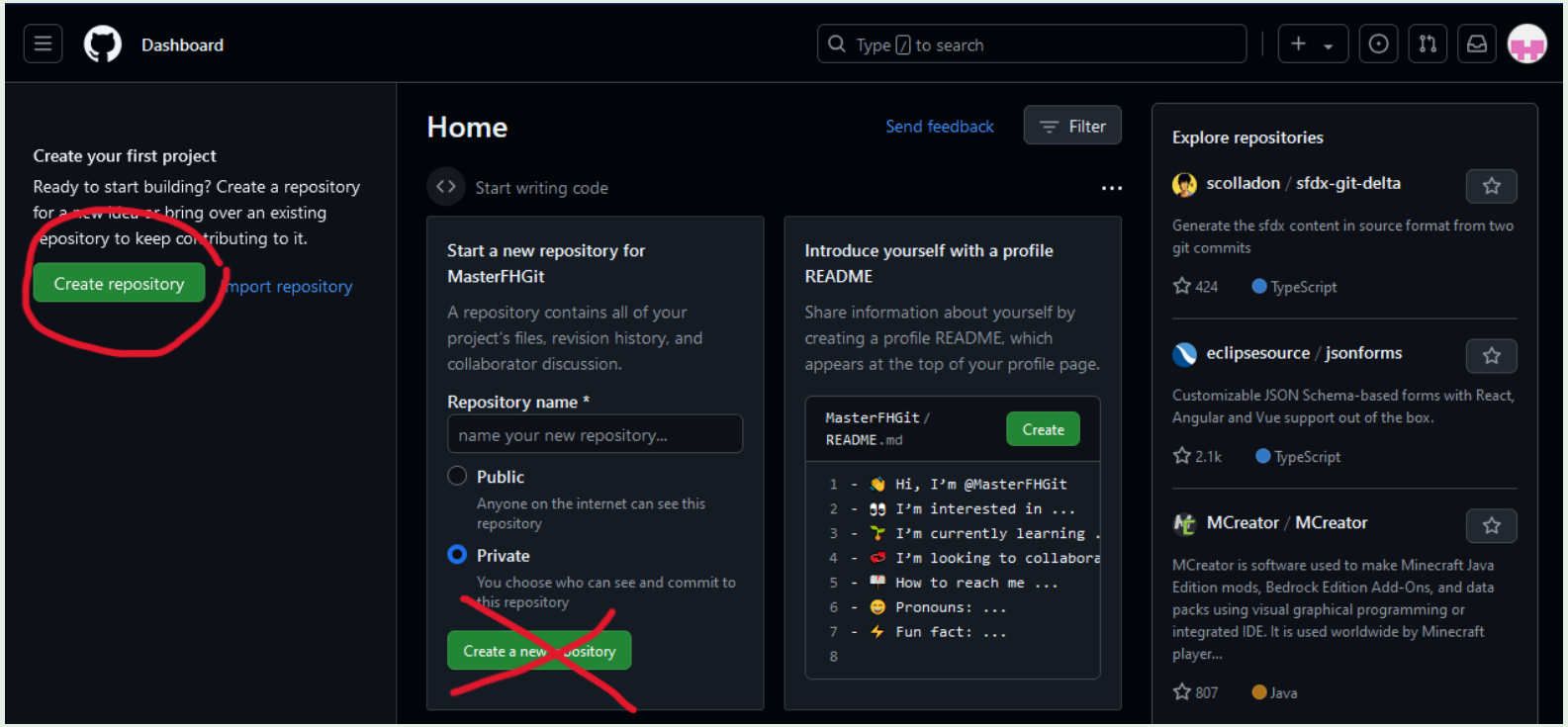


Figure: Création d'un dépôt dans Github

Utilisation sur GitHub

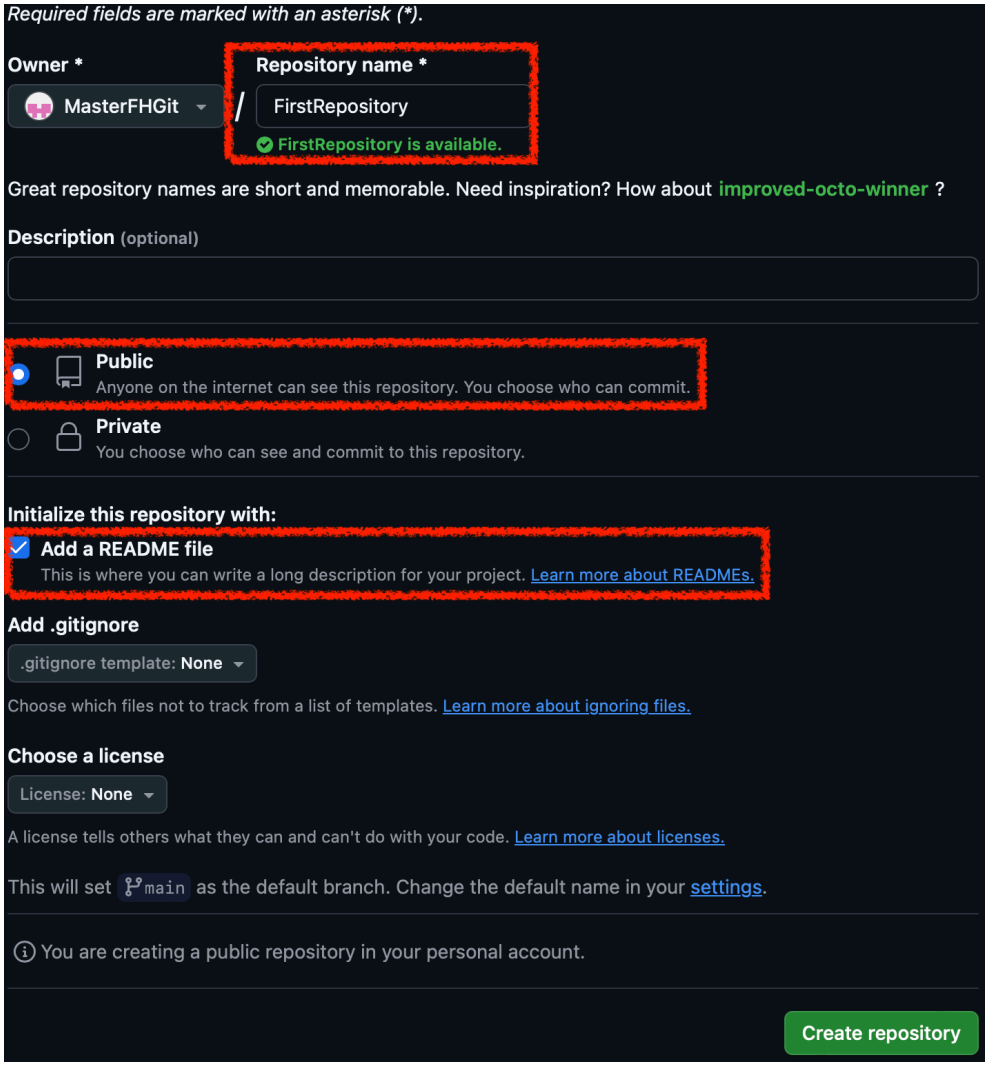
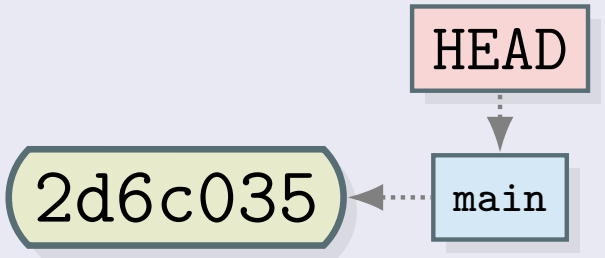


Figure: Création du dépôt "FirstRepository" public avec fichier README

État initial après le premier commit sur GitHub



- **2d6c035** est l'ID du commit (l'ID complet est un SHA1)
- **main** est une référence au dernier commit de la branche qui porte le même nom, sur le dépôt distant
- On peut avoir plusieurs branches et donc plusieurs références (**main**, **dev**, **feature1** etc.)
- On trouvera aussi **master** à la place de **main**
- **main** ou **master** sont par convention des noms de branches de production

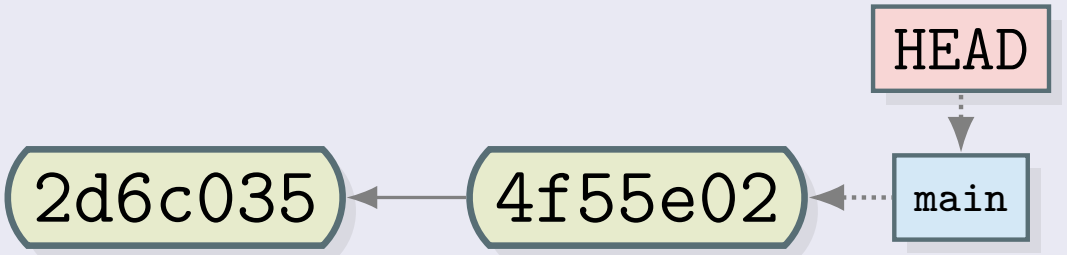
Exercice I - Premier pas avec GitHub

3. Modifier le fichier README.md sur GitHub

Exercice I - Premier pas avec GitHub

3. Modifier le fichier README.md sur GitHub

État après le deuxième commit sur github



Heads

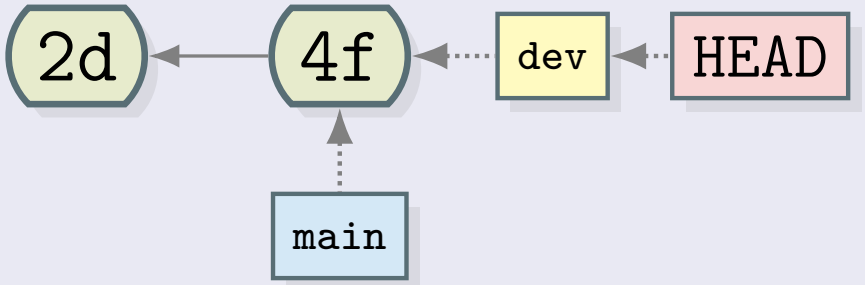
A head is simply a reference to a commit object. Each head has a name. By default, there is a head in every repository called master (or main). A repository can contain any number of heads. At any given time, one head is selected as the "current head." This head is aliased to HEAD, always in capitals.
Note this difference: a "head" (lowercase) refers to any one of the named heads in the repository; "HEAD" (uppercase) refers exclusively to the currently active head. This distinction is used frequently in Git documentation.

Git - Heads

Exercice I - Premier pas avec GitHub

4. Créer une nouvelle branche nommée **dev**
5. Sur la branche **dev**, modifier le fichier README.md

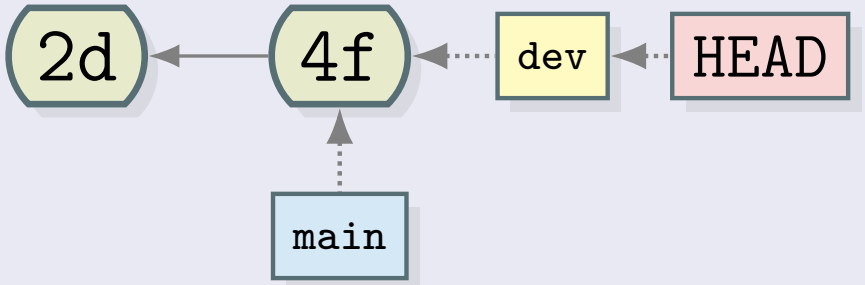
État juste après la création de la branche **dev**



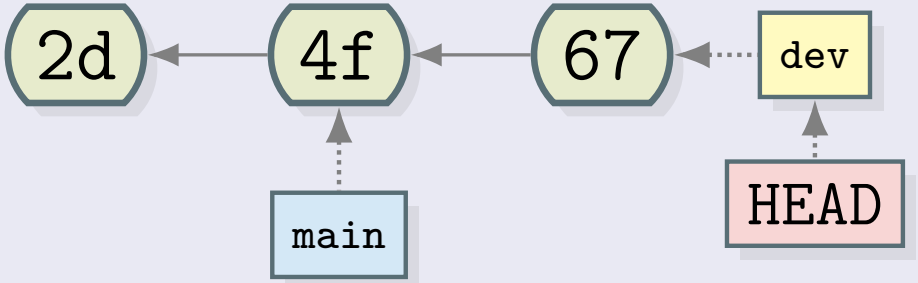
Exercice I - Premier pas avec GitHub

4. Créer une nouvelle branche nommée **dev**
5. Sur la branche **dev**, modifier le fichier README.md

État juste après la création de la branche **dev**



État juste après le commit dans la branche **dev**



Concepts importants à retenir

- Branches
- Commits
- Heads (références, pointeurs)
- Toute branche a son pointeur de même nom qui référence toujours le dernier commit de la branche
- HEAD est le seul pointeur qui peut bouger. C'est à partir de lui que sera créé le prochain commit.

Exercice II - Installation de Git en local

1. Installer Git en suivant le lien suivant : **Git - SCM**

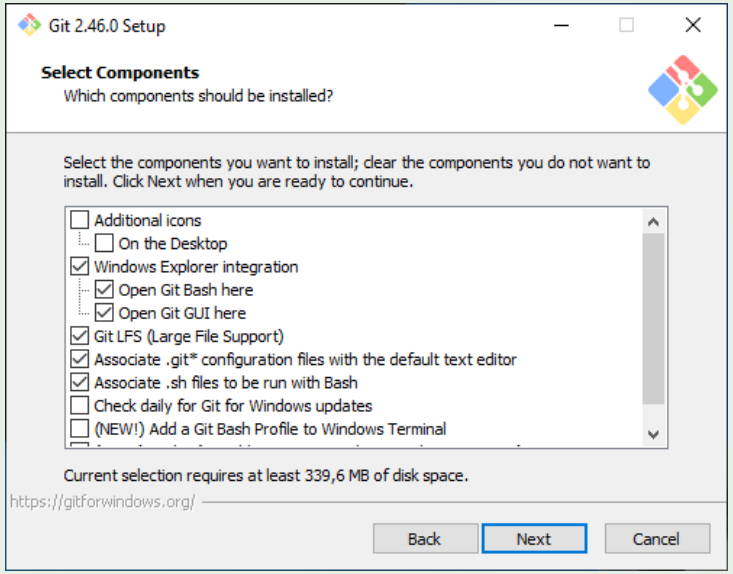


Figure: Git Setup - Garder les options par défaut

2. Créer un dossier GIT sur votre machine
3. Importer votre dépôt FirstRepository dans ce dossier

Commandes clone & remote

MINGW64:/c/Users/Buloup/Documents/GIT/FirstRepository

```
Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT
$ pwd
/c/Users/Buloup/Documents/GIT

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT
$ git clone https://github.com/MasterFHGit/FirstRepository.git
Cloning into 'FirstRepository'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT
$ git remote -v
fatal: not a git repository (or any of the parent directories): .git

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT
$ cd FirstRepository/

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (main)
$ git remote -v
origin  https://github.com/MasterFHGit/FirstRepository.git (fetch)
origin  https://github.com/MasterFHGit/FirstRepository.git (push)

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (main)
$ |
```

Figure: Commandes clone et remote

Commandes clone & remote

MINGW64:/c/Users/Buloup/Documents/GIT/FirstRepository/.git

```
Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (main)
$ ls -l -a
total 5
drwxr-xr-x 1 Buloup 197121  0 Sep 16 17:09 ./
drwxr-xr-x 1 Buloup 197121  0 Sep 16 17:09 ../
drwxr-xr-x 1 Buloup 197121  0 Sep 16 17:09 .git/
-rw-r--r-- 1 Buloup 197121 17 Sep 16 17:09 README.md

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (main)
$ cd .git

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository/.git (GIT_DIR!)
$ ls -l
total 9
-rw-r--r-- 1 Buloup 197121  21 Sep 16 17:09 HEAD
-rw-r--r-- 1 Buloup 197121 309 Sep 16 17:09 config
-rw-r--r-- 1 Buloup 197121  73 Sep 16 17:09 description
drwxr-xr-x 1 Buloup 197121   0 Sep 16 17:09 hooks/
-rw-r--r-- 1 Buloup 197121 137 Sep 16 17:09 index
drwxr-xr-x 1 Buloup 197121   0 Sep 16 17:09 info/
drwxr-xr-x 1 Buloup 197121   0 Sep 16 17:09 logs/
drwxr-xr-x 1 Buloup 197121   0 Sep 16 17:09 objects/
-rw-r--r-- 1 Buloup 197121 112 Sep 16 17:09 packed-refs
drwxr-xr-x 1 Buloup 197121   0 Sep 16 17:09 refs/

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository/.git (GIT_DIR!)
$ cat config
[core]
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
[remote "origin"]
        url = https://github.com/MasterFHGit/FirstRepository.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = origin
        merge = refs/heads/main

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository/.git (GIT_DIR!)
$ |
```

Figure: Dossier .git et fichier config

remote "origin" ?

- origin est un alias local donné au dépôt distant
- il peut être changé dans le fichier config ...

MINGW64:/c/Users/Buloup/Documents/GIT/FirstRepository/.git

```
Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository/.git (GIT_DIR!)
$ vim config

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository/.git (GIT_DIR!)
$ cat config
[core]
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
[remote "toto"]
        url = https://github.com/MasterFHGit/FirstRepository.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = toto
        merge = refs/heads/main

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository/.git (GIT_DIR!)
$ |
```

Figure: Changer l'alias origin

... ou lors de la commande clone

MINGW64:/c/Users/Buloup/Documents/GIT/FirstRepository

```
Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT
$ git clone https://github.com/MasterFHGit/FirstRepository.git -o toto
Cloning into 'FirstRepository'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT
$ cd FirstRepository/

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (main)
$ git remote -v
toto    https://github.com/MasterFHGit/FirstRepository.git (fetch)
toto    https://github.com/MasterFHGit/FirstRepository.git (push)

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (main)
$ cat ./.git/config
[core]
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
[remote "toto"]
        url = https://github.com/MasterFHGit/FirstRepository.git
        fetch = +refs/heads/*:refs/remotes/toto/*
[branch "main"]
        remote = toto
        merge = refs/heads/main

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (main)
$ |
```

Figure: Changer l'alias origin

Comment supprimer un dépôt local ?

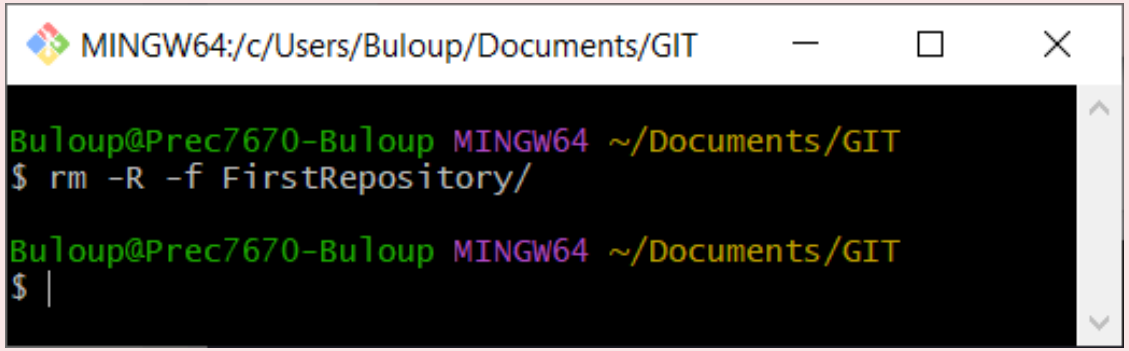


Figure: Supprimer un dépôt local

branch, main, fetch, push, merge etc. ?

C'est le vocabulaire des DVCS (Git)
Plus clair progressivement !

Petit résumé intermédiaire

Quelques Commandes : clone, status, log, switch

```
$ git clone url              # Clone repository
$ git remote -v              # Remotes infos
$ git status                 # Git status
$ git log                    # Print commits
$ git log --oneline          # Print abbrev-commits
$ git switch branchName      # Move HEAD to branchName
```

Il existe un répertoire .git

- Il contient la base de donnée locale du dépôt : dépôt local
- Il contient un fichier de configuration du dépôt

Commandes fetch & checkout

MINGW64:/c/Users/Buloup/Documents/GIT/FirstRepository

```
Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT
$ mkdir FirstRepository && cd FirstRepository

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository
$ git init
Initialized empty Git repository in C:/Users/Buloup/Documents/GIT/FirstRepository/.git/

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (master)
$ git remote add origin https://github.com/MasterFHGit/FirstRepository.git

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (master)
$ git fetch origin
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 9 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (9/9), 2.62 KiB | 38.00 KiB/s, done.
From https://github.com/MasterFHGit/FirstRepository
 * [new branch]      main       -> origin/main

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (master)
$ git checkout main
Switched to a new branch 'main'
branch 'main' set up to track 'origin/main'.

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (main)
$ ls
README.md

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (main)
$ cat ./.git/config
[core]
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
[remote "origin"]
        url = https://github.com/MasterFHGit/FirstRepository.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = origin
        merge = refs/heads/main

Buloup@Prec7670-Buloup MINGW64 ~/Documents/GIT/FirstRepository (main)
$ |
```

Figure: Importer un dépôt distant - Version 2

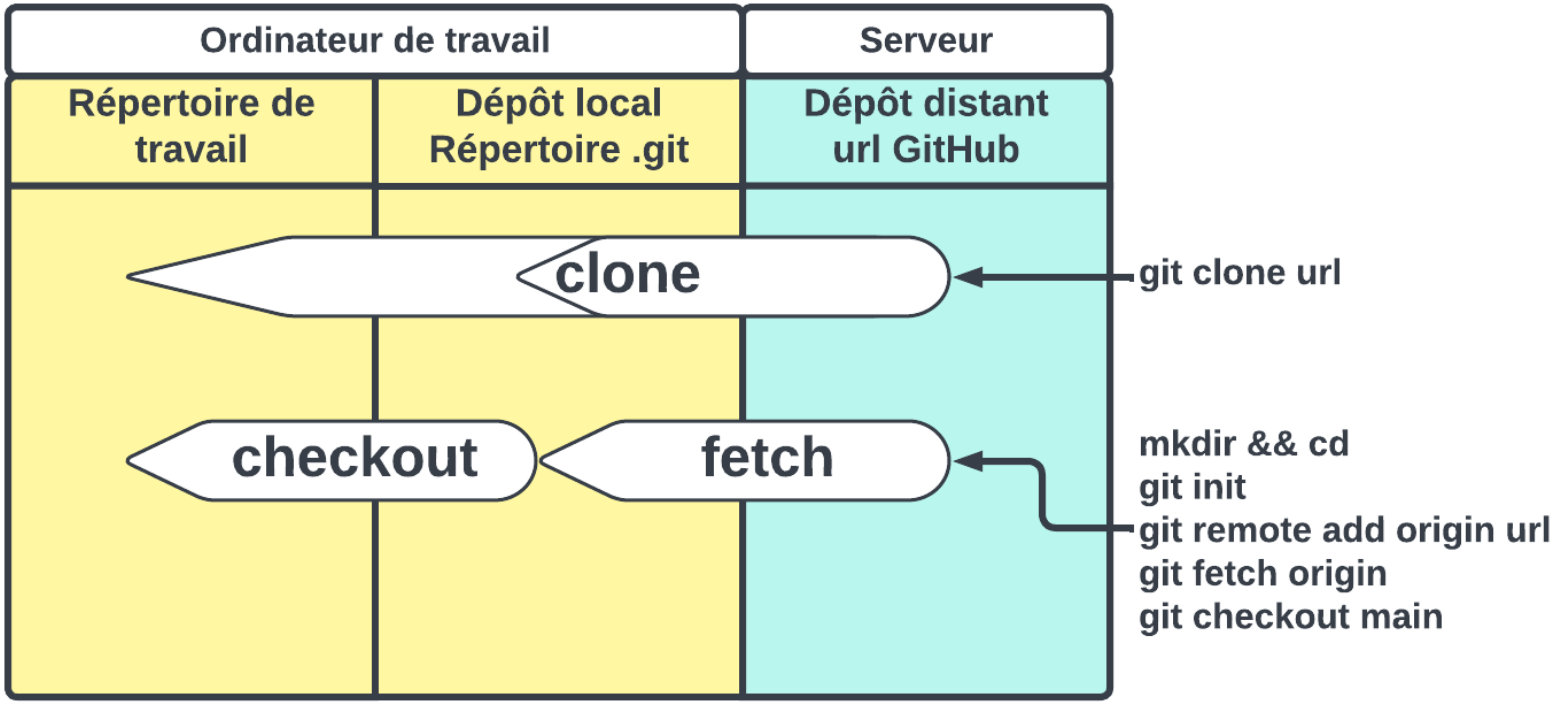


Figure: Importer un dépôt distant - Les deux versions

Exercice III - Changements à partir du dépôt distant

1. Changer le fichier README sur github (branche main)
2. Reporter ces changements en local

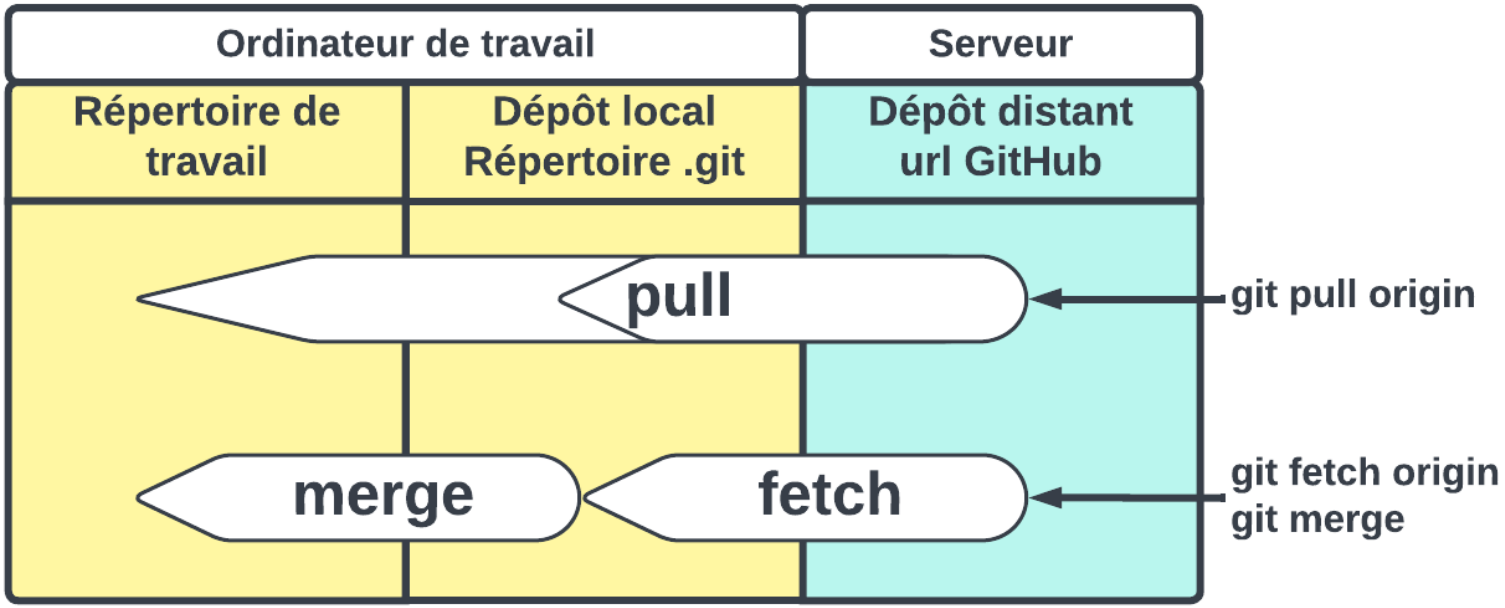


Figure: Mettre à jour un dépôt local - Les deux versions

FirstRepository — frank@mbp-de-frank — ..rstRepository — -zsh — 69×18

```
~/Documents/GIT/FirstRepository  on main  ✔
└─ git pull origin
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 904 bytes | 301.00 KiB/s, done.
From https://github.com/MasterFHGit/FirstRepository
   4c8e792..a6ab892  main       -> origin/main
Updating 4c8e792..a6ab892
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)

~/Documents/GIT/FirstRepository  on main  ✔
└─
```

Figure: Mettre à jour un dépôt local - Version 1

FirstRepository — frank@mbp-de-frank — ..rstRepository — -zsh — 82×20

```
~/Documents/GIT/FirstRepository  on main
└─ git fetch origin
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 902 bytes | 300.00 KiB/s, done.
From https://github.com/MasterFHGit/FirstRepository
   a6ab892..df35435  main       -> origin/main

~/Documents/GIT/FirstRepository  on main ⇣1
└─ git merge
Updating a6ab892..df35435
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)

~/Documents/GIT/FirstRepository  on main
```

Figure: Mettre à jour un dépôt local - Version 2

Commande fetch & option –dry-run

```
FirstRepository — frank@mbp-de-frank — ..rstRepository — -zsh — 82×11
~/Documents/GIT/FirstRepository on main
git fetch origin --dry-run
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 917 bytes | 305.00 KiB/s, done.
From https://github.com/MasterFHGit/FirstRepository
   14b2f3a..199fb9d  main       -> origin/main
~/Documents/GIT/FirstRepository on main
```

Rien n'est modifié dans le dépôt local

Figure: Le dépôt local est-il à jour ? Non !

```
FirstRepository — frank@mbp-de-frank — ..rstRepository — -zsh — 82×13
~/Documents/GIT/FirstRepository on main
git pull origin
From https://github.com/MasterFHGit/FirstRepository
   14b2f3a..199fb9d  main       -> origin/main
Updating 14b2f3a..199fb9d
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

~/Documents/GIT/FirstRepository on main
git fetch origin --dry-run

~/Documents/GIT/FirstRepository on main
```

Figure: Maintenant oui !

Petit résumé intermédiaire

Quelques commandes : pull, fetch

```
$ git pull origin                    # Get from origin
$ git fetch origin && git merge      # Idem !
$ git fetch origin --dry-run         # Has origin changed ?
$ git log                            # Print commits
$ git log --oneline                  # Print abbrev-commits
```

Exercice IV - Changements à partir du dépôt local

1. Changer le fichier README en local (branche main)
2. Reporter ces changements en distant

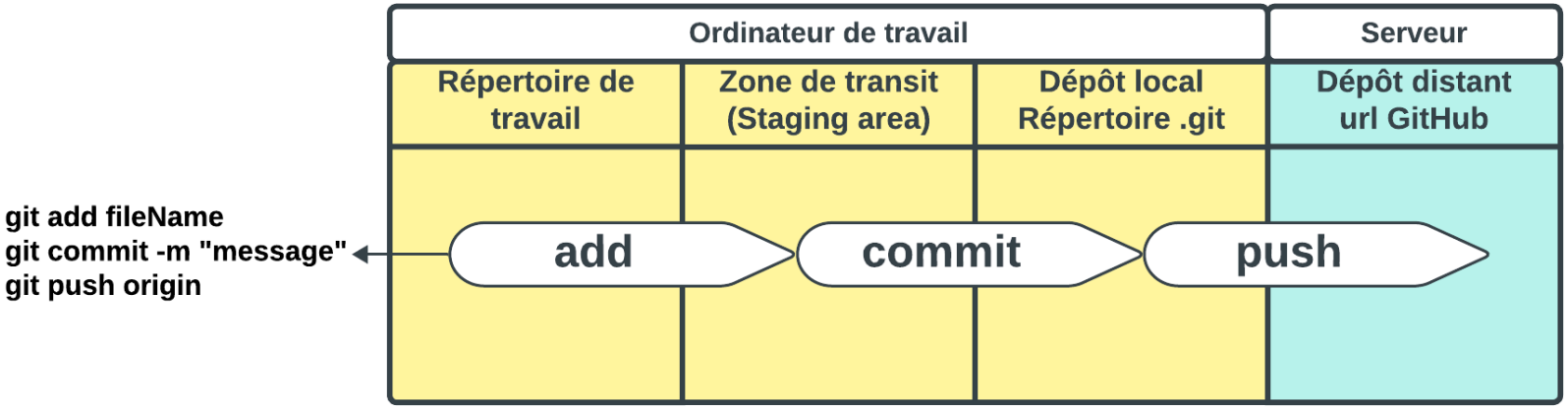


Figure: Mettre à jour le dépôt distant

```
FirstRepository — frank@mbp-de-frank — ..rstRepository — -zsh — 98×33
~/Documents/GIT/FirstRepository  on main                                  ✔  took 11s
clear

~/Documents/GIT/FirstRepository  on main                                  ✔
~/micro README.md

~/Documents/GIT/FirstRepository  on main !1                               ✔  took 7s
git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

~/Documents/GIT/FirstRepository  on main !1                               ✔
git add README.md

~/Documents/GIT/FirstRepository  on main +1                               ✔
git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md

~/Documents/GIT/FirstRepository  on main +1                               ✔
```

Figure: Zone de transit

Commandes add, commit & push

FirstRepository — frank@mbp-de-frank — ..rstRepository — -zsh — 98×37

```
~/Documents/GIT/FirstRepository  on main
~/micro README.md

~/Documents/GIT/FirstRepository  on main !1  took 8s
git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

~/Documents/GIT/FirstRepository  on main !1
git add README.md

~/Documents/GIT/FirstRepository  on main +1
git commit README.md -m "Changed from local repsitory"
[main 47b514b] Changed from local repsitory
 1 file changed, 1 insertion(+)

~/Documents/GIT/FirstRepository  on main ↑1
git push origin
Username for 'https://github.com': MasterFHGit
Password for 'https://MasterFHGit@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repos
itories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/MasterFHGit/FirstRepository.git/'

~/Documents/GIT/FirstRepository  on main ↑1  128 ✘  took 24s
```

Figure: Mettre à jour le dépôt distant

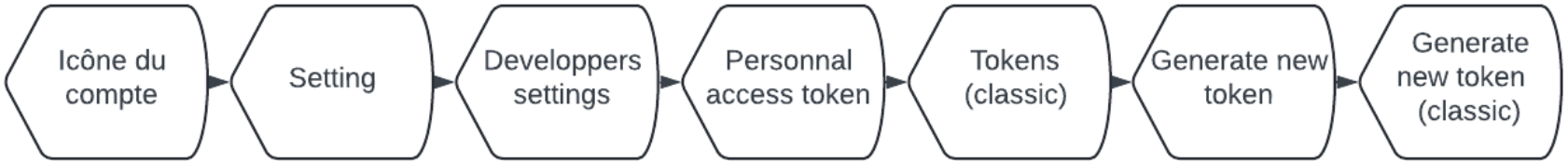


Figure: Génération d'un jeton dans GitHub

Génération d'un jeton d'accès personnel

- Conserver ce jeton. Il n'est visible qu'une seule fois !
- C'est bien de lui donner un petit nom (note) !
- Ne pas oublier de sélectionner l'option "repo"
- Sur un dépôt privé, rien n'est possible sans jeton

Pour plus d'infos : **Personal Access Token (PAT)**

FirstRepository — frank@mbp-de-frank — ..rstRepository — -zsh — 80×24

```
~/Documents/GIT/FirstRepository  on main ↑1                                  128 ✘
└ git push origin
Username for 'https://github.com': MasterFHGit
Password for 'https://MasterFHGit@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/
about-remote-repositories#cloning-with-https-urls for information on currently r
ecommended modes of authentication.
fatal: Authentication failed for 'https://github.com/MasterFHGit/FirstRepository
.git/'

~/Documents/GIT/FirstRepository  on main ↑1                  128 ✘  took 14s
└ git push origin
Username for 'https://github.com': MasterFHGit
Password for 'https://MasterFHGit@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 293 bytes | 293.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/MasterFHGit/FirstRepository.git
   f263e1c..8bdc17a  main -> main

~/Documents/GIT/FirstRepository  on main                          ✔  took 7s
```

Figure: Avec le jeton comme mdp, le push fonctionne

Pour enregistrer le jeton

```
$ git config --global credential.helper store
```

... et ne plus avoir à le retaper qu'une seule fois. Le jeton est enregistré dans le fichier texte **.git-credentials** de votre répertoire personnel.

Pour signer les commits automatiquement

```
$ git config --global user.email "mail@somewhere.com"
$ git config --global user.name "username"
```

Signed-off-by: username <mail@somewhere.com>

Pour cloner (ou autre) on peut aussi faire :

```
$ git clone https://uname:pass@github.com/uname/rp.git
```

Petit résumé intermédiaire

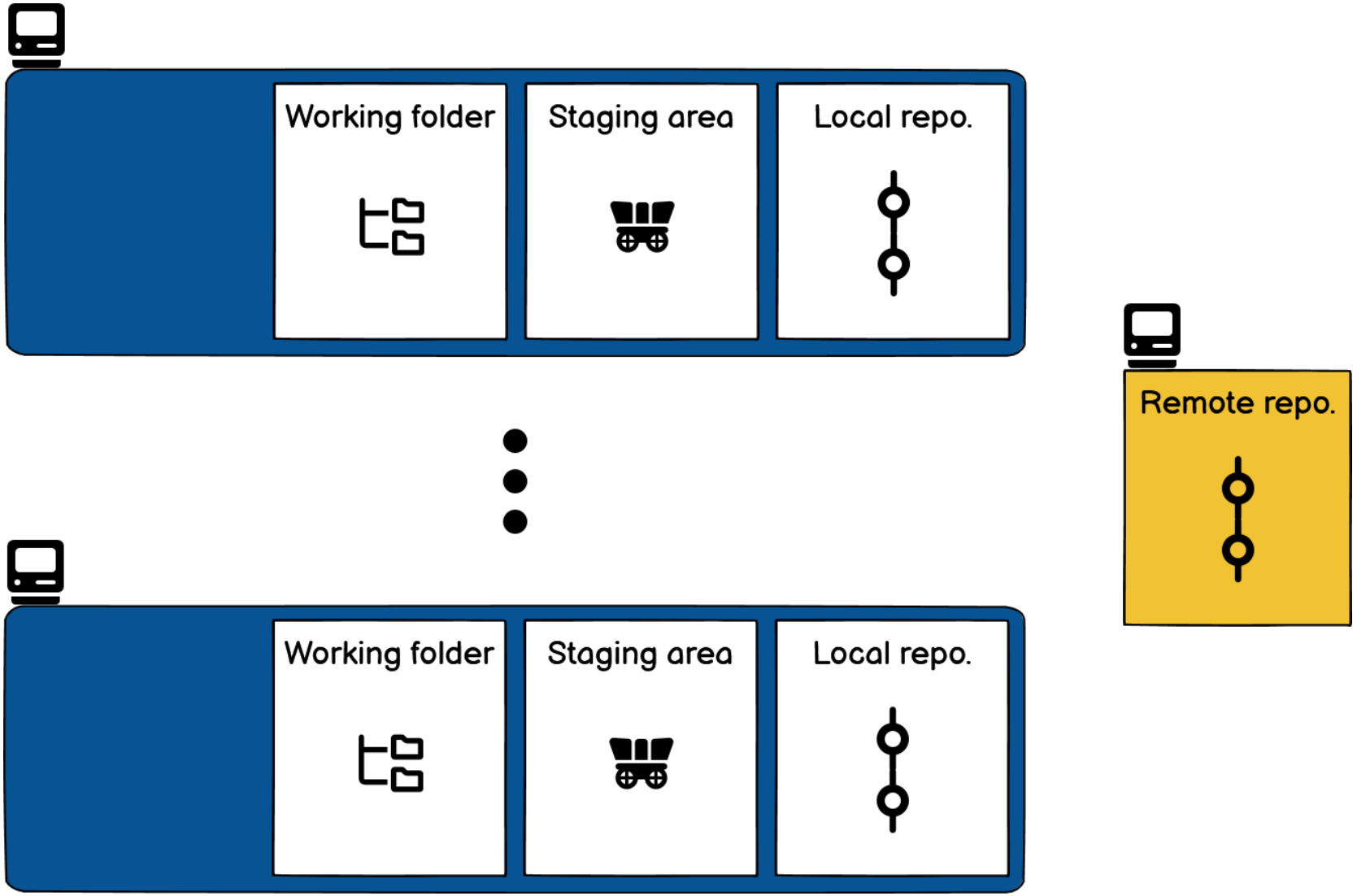
Quelques commandes : add, commit

```
$ git add fileName # Stage fileName
$ git add .        # Stage modified and new files
$ git add -u       # Stage modified and deleted files
$ git add -A       # Stage everything
$ git commit -m "A commit message"
```

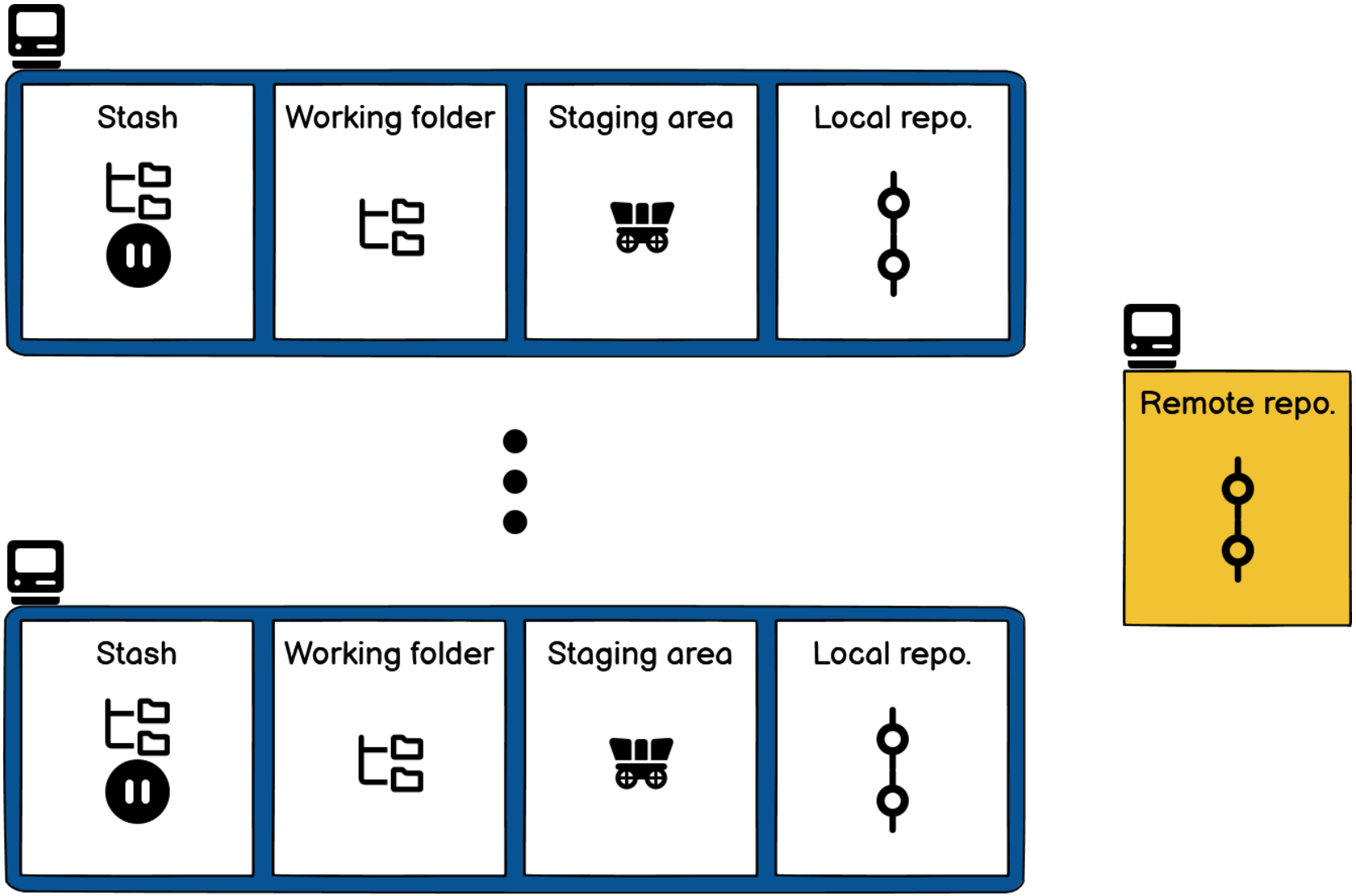
D'autres commandes utiles ! switch, branch, checkout

```
$ git switch branchName     # Switch to branchName
$ git checkout 52fd6e4      # In detached HEAD
$ git checkout branchName   # HEAD on branchName
$ git branch -D branchName  # To delete branchName
$ git branch                # List all local branches
$ git branch -r             # List all remote branches
$ git branch -a             # List all branches
$ git branch newBranchName  # Create new branch
$ git switch -c neBrNa      # Create & switch to neBrNa
```

Les zones de Git



Les zones de Git



stash = réserve, remise

Exercice V - Commande revert

1. Ajouter un nouveau commit dans la branche **main** en modifiant le fichier README.md (Before revert)
2. Utiliser la commande revert pour annuler ce commit
3. Utiliser la commande checkout pour vous déplacer dans les commits

Exercice V - Commande revert

1. Ajouter un nouveau commit dans la branche **main** en modifiant le fichier README.md (Before revert)
2. Utiliser la commande revert pour annuler ce commit
3. Utiliser la commande checkout pour vous déplacer dans les commits

Commande revert

```
$ git switch main            # Goto main branch
$ vim README.md              # Modify README
$ git commit -am "Before revert" # Commit
$ git revert HEAD            # Revert last commit
$ git log --oneline          # See logs
$ cat README.md              # File content
$ git checkout 44e5761       # Goto previous commit
$ cat README.md              # File content
```

Commande revert

- Elle ne modifie pas l'historique des commits (pas de suppression de commits)
- Elle crée un nouveau commit
- Elle détermine comment inverser les changements introduits par le commit spécifié et ajoute un nouveau commit avec le contenu inversé qui en résulte

Exercice VI - Commande reset

1. Modifier le fichier README.md sans faire de commit
2. Supprimer les deux derniers commits (le Before revert et le revert) précédents en utilisant la commande reset
3. Conclusion

Exercice VI - Commande reset

1. Modifier le fichier README.md sans faire de commit
2. Supprimer les deux derniers commits (le Before revert et le revert) précédents en utilisant la commande reset
3. Conclusion

Commande reset

```
$ git checkout main    # HEAD to main
$ vim README.md        # Change README.md
$ git log --oneline    # See logs
$ git reset 9d49258    # Reset two commits
$ git log --oneline    # See logs
$ cat README.md        # File content
$ git restore README.md # Get from index
$ cat README.md        # File content
```

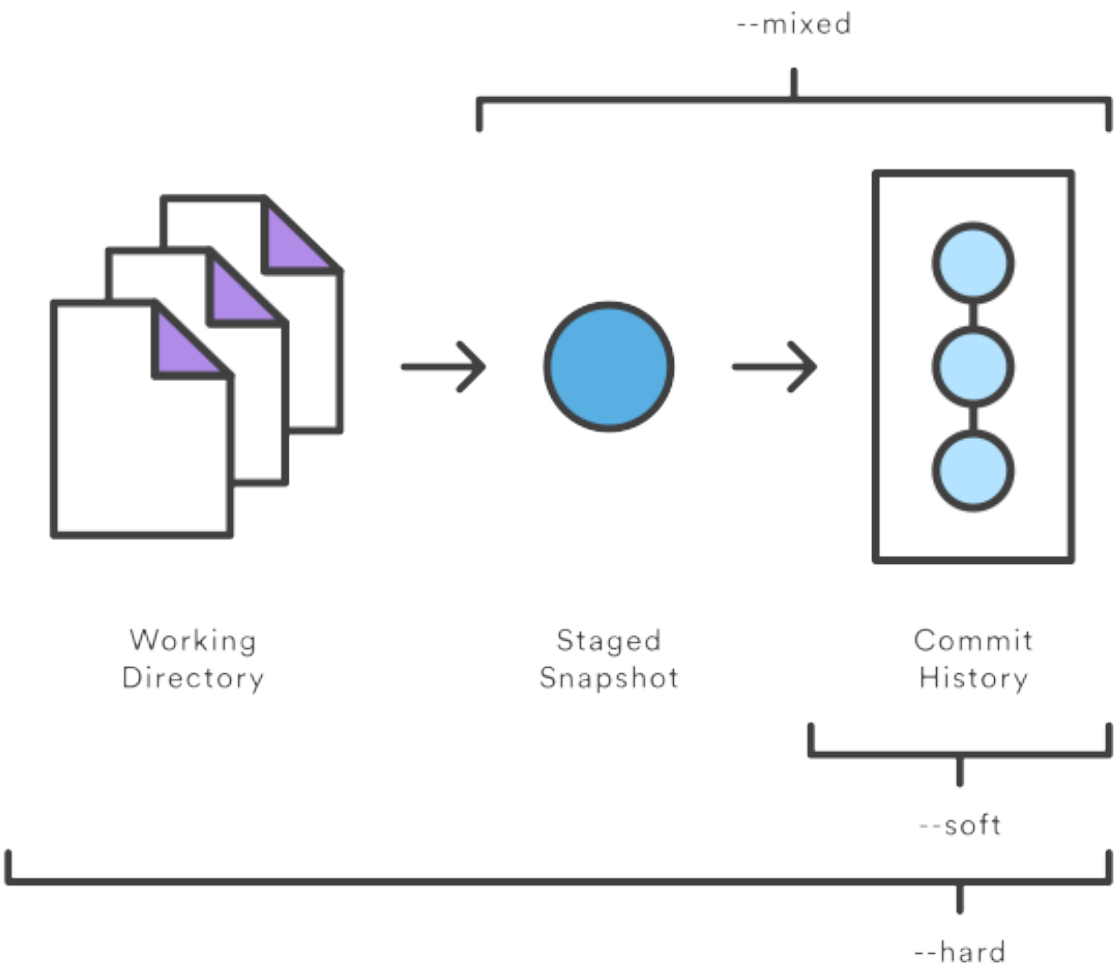


Figure: Commande reset - Les différents modes

Commande reset - Option mixed

- Elle revient dans l'état du commit spécifié en mettant l'index (staging area) à jour mais pas la copie de travail
- Elle modifie l'historique des commits (supprime des commits)
- La commande restore permet de récupérer l'index : git restore README.md
- C'est l'option par défaut : mode mixte

"git reset idCommit" et "git reset --mixed idCommit" sont donc des commandes indentiques

Commande reset - Options soft, mixed et hard

- Récupérer l'état de certains fichiers et conserver les modifications en cours pour les autres fichiers
- Réécrire l'historique des commits dans le dépôt (e.g. regrouper des commits)

Comment annuler un reset ?

Quelle que soit l'option utilisée, il est toujours possible d'annuler un reset en retrouvant l'ID des commits par la commande reflog

```
1 $ git reflog  # get idCommit just before reset
2 $ # reset hard to this commit
3 $ git reset --hard idCommit
```

--hard reste une option dangereuse !

Exercice VII - Préparation fusion et rebase

1. Ajouter un nouveau commit dans la branche **dev**

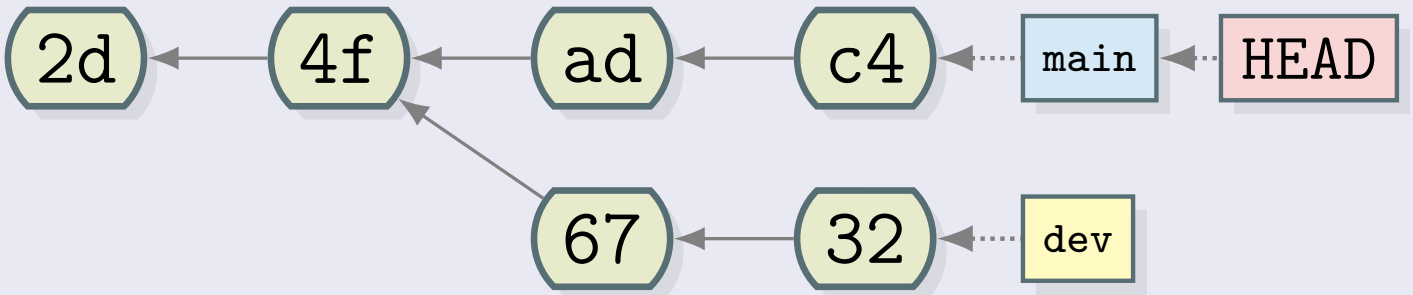
Exercice VII - Préparation fusion et rebase

1. Ajouter un nouveau commit dans la branche **dev**

Préparation fusion et rebase

```
$ git switch dev            # Goto dev branch
$ vim README.md             # Modify README
$ git commit -am "devC1" # Commit 1 in dev branch
```

État actuel du dépôt local



Comment récupérer les modifications de **dev** dans **main** ?

- Les branches **dev** et **main** ont divergées
- Problème de fusion de branches : gestion des conflits

Exercice VIII - Fusion de branches avec conflits

1. Fusionner la branche dev dans la branche main en local

Fusionner : la commande merge

```
$ git switch main   # Goto main branch
$ git merge dev     # Merge dev branch to main branch
```

```
~/Documents/GIT/FirstRepository  on main merge ~1
git status
On branch main
Your branch is up to date with 'origin/main'.

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

~/Documents/GIT/FirstRepository  on main merge ~1
cat README.md
# FirstRepository

First commit from main branch

<<<<<<< HEAD
seconde commit from main branch

third commit from main branch - local
=======
First commit from dev branch
>>>>>>> dev

~/Documents/GIT/FirstRepository  on main merge ~1
```

Figure: Résultat de la fusion

Exercice VIII - Fusion de branches avec conflits

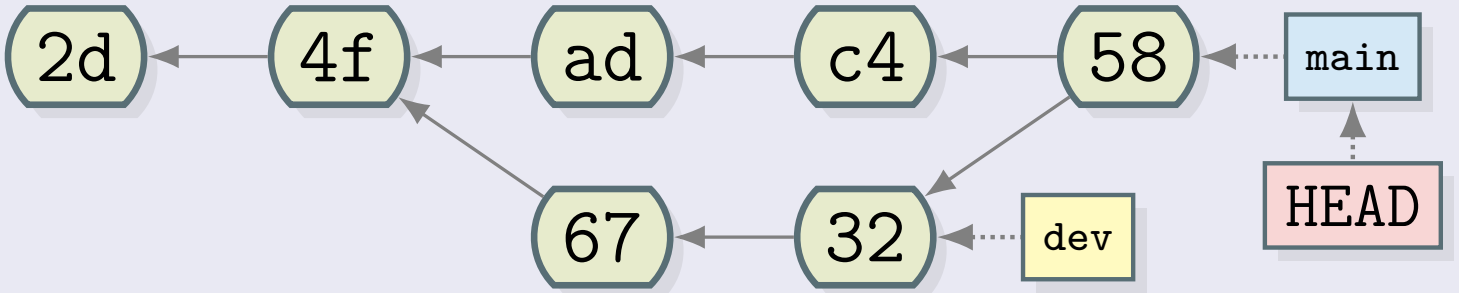
1. Fusionner la branche dev dans la branche main en local
2. Modifier le ficher README.md
3. Ajouter ce fichier au staging (marquer la résolution de conflits)
4. Faire le commit

Fusionner : la commande merge

```
$ git switch main    # Goto main branch
$ git merge dev      # Merge dev branch to main branch
$ vim README.md      # Modify README.md
$ git add README.md  # To mark resolution
$ git commit -m "dev->main - Conflicts resolution"
```

Ne pas hésiter à utiliser la commande git status régulièrement

État du dépôt local après la fusion - Merge



Les commits de la branche *dev* ont été fusionnés avec la branche *main* dans le dernier commit de cette branche (58)

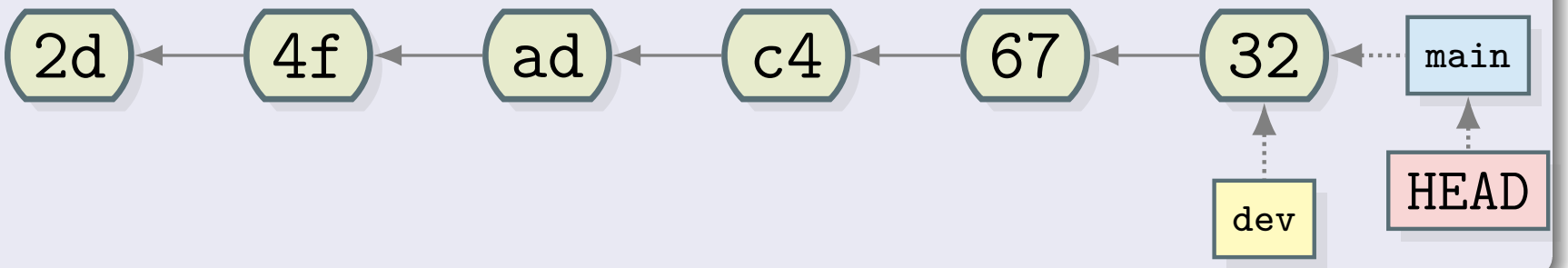
Exercice IX - Rebase de branches avec conflits

1. Revenir dans l'état du dépôt distant (supprimer puis cloner de nouveau le dépôt distant)
2. Ajouter un nouveau commit dans la branche **dev** en local
3. Rebaser la branche dev dans la branche main en local
4. Modifier le ficher README.md
5. Ajouter ce fichier au staging pour marquer la résolution de conflits
6. Terminer le rebase

Rebaser : la commande rebase

```
$ git switch dev            # Goto dev branch
$ vim README.md             # Edit readme
$ git commit -am "devC1"    # New commit in dev
$ git switch main           # Goto main
$ git rebase main dev       # Rebase dev in main
$ vim README.md             # Resolve conflicts
$ git add README.md         # Mark resolved
$ git rebase --continue     # Continue rebase
$ git switch main           # Goto main
$ git merge dev             # Merge dev in main
$ git branch -d dev         # Delete dev (-D?)
```

État du dépôt local après le rebase



Tous les commits de la branche *dev* ont été mis à la suite de ceux de la branche *main*

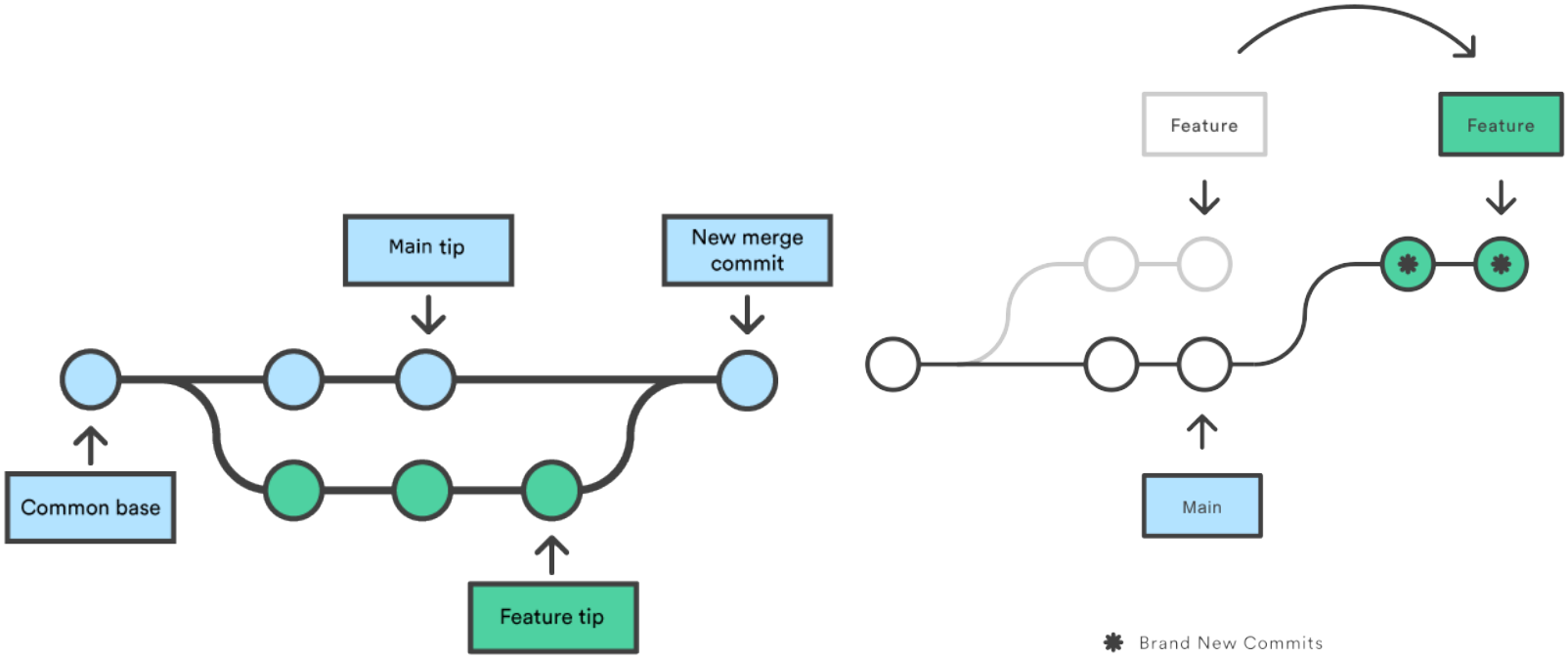


Figure: Illustration des commandes merge à gauche et rebase à droite

Petit résumé intermédiaire

Fusionner : la commande merge

```
# Merge branchName in current branch
$ git merge branchName
```

Rebaser : la commande rebase

```
$ git rebase main feature # Rebase feature in main
$ ..... # Resolve conflicts ?
$ git rebase --continue # Continue rebase
$ git switch main # Return to main
$ git merge feature # Merge feature in main
$ git branch -D feature # Delete feature
```

Travail en équipe

- Chaque développeur doit avoir un compte GitHub
- Chaque développeur doit avoir un PAT
- Le propriétaire du dépôt origine doit envoyer des invitations

Pour plus d'infos : **Inviter des collaborateurs**

3 Projet

Veuillez suivre ce lien

4 Annexes

- Oh my ¿what? !
- Applications utiles
- Créer un dépot Github à partir d'un dépot local
- Lecture de la sortie d'un diff
- Lecture de la sortie d'un pull
- Une longue ligne de commande !
- Les accroches - Hooks
- Les alias
- Cueillons des cerises !
- La remise
- Comment dire à git d'ignorer des fichiers ?
- Quel casse tête tous ces HEAD !
- Vidéos sur Delicious Insights

Pour avoir un superbe terminal !

Suivre ce lien : **Oh my posh !**
Installation sous Windows

Pour avoir un autre superbe terminal !

Suivre ce lien : **Oh my bash !**

Installation :

```
$ bash -c "$(curl -fsSL https://raw.githubusercontent.com/ohmybash/oh-my-bash/master/tools/install.sh)"
```

Désinstallation :

```
$ uninstall_oh_my_bash
```

Modification du thème et des plugins du Terminal alors possible

Encore une alternative : **Oh my zsh !**

Figure: Logiciels SourceTree, GitHub et GitKraken Desktop

Trois outils graphiques libres et très utiles

Contexte

J'ai des fichiers sur mon ordi que je souhaite gérer sur GitHub

Solution

1. Créer le dépôt local si pas déjà fait
2. Créer le dépôt distant sur GitHub **VIDE** (évite les erreurs) :
   - Pas de fichier README.md, .gitignore, licence etc.
   - Récupérer l'URL du dépôt distant
3. Ajouter cet URL en tant que dépôt distant
4. Pousser la branche principale sur le dépôt distant

```
$ git init
$ git add ...
$ git commit ...
$ git remote add origin REMOTE-URL
$ git remote -v # Check URL has been added
$ git push -u origin main
```

Lecture de la sortie d'un diff

```
1  $ git diff 7e46066..3a78b4a # Changes between commits
```

```
 \subsection{Utilisation sur GitHub}
@@ -479,7 +484,7 @@ Note this difference: a “head” (lowercase) refers to any one of the named he
 \begin{alertblock}{remote "origin" ?}
 \begin{itemize}
     \item origin est un alias local donné au dépôt distant
-    \item il peut être changé dans le fichier config
+    \item il peut être changé dans le fichier config ...
     \center{
         \includegraphics[scale=0.5]{images/origin1.PNG}
         \captionof{figure}{Changer l'alias origin}}
@@ -490,15 +495,10 @@ Note this difference: a “head” (lowercase) refers to any one of the named he
 \end{frame}

 \begin{frame}
-\begin{alertblock}{remote "origin" ?}
-\begin{itemize}
-    \item ou lors de la commande clone
+\begin{alertblock}{... ou lors de la commande clone}
     \center{
         \includegraphics[scale=0.45]{images/origin2.PNG}
         \captionof{figure}{Changer l'alias origin}}
-\end{itemize}
-\centering
-
 \end{alertblock}
 \end{frame}

@@ -554,7 +554,7 @@ $ git switch branchName   # Move HEAD to branchName
```

« Hunk » en terminolgie git (bloc)

Ligne supprimée

Ligne ajoutée

-479,7 : 7 lignes **à partir de** la 479 dans le commit 7e46066

+484,7 : 7 lignes **à partir de** la 484 dans le commit 3a78b4a

Lecture de la sortie d'un diff

```
# Changes between workspace and stage
$ git diff
# Changes between workspace + stage and last commit
$ git diff HEAD
```

```
diff --git a/presentation.pdf b/presentation.pdf
index eb2bfe9..326de97 100644
Binary files a/presentation.pdf and b/presentation.pdf differ
diff --git a/presentation.tex b/presentation.tex
index a02e9fe..4f1635d 100644
--- a/presentation.tex
+++ b/presentation.tex
@@ -1165,36 +1165,6 @@ Pour plus d'infos : \textcolor{blue}{\textbf{\unc
 \end{frame}

 \section{Les projets}
-
-
-\begin{frame}
-\begin{alertblock}{Rien à disposition pour faire des mesures objectives
-\begin{itemize}
```

Les index présents en entête de la sortie d'un diff sont des identifiants internes à git, pas des numéros de commits !

Comparaison de fichiers entre dépôts distant et local

```
# - Workspace and stage diffs are ignored -
# Without git fetch, changes between origin and local
# repos won't be synchronised and diff will be blind.
# Use git branch -a to list local and "remote local"
# branches. e.g. : branchName and
# remotes/origin/branchName
# In order to update "remote local" branches ->
$ git fetch
# Changes between main branches : local branch main
# and "remote local" branch remotes/origin/main ->
$ git diff main origin/main
# Changes in path/tofile between origin
# and local repos in main branches ->
$ git diff origin/main:path/tofile main:path/tofile
$ ...
$ git merge origin/main
```

Lecture de la sortie d'un pull

```
~/Documents/GIT/CourseMaterial  on main                                  ✓  base
└─ git pull
From https://github.com/MasterFHGit/CourseMaterial
   7e46066..3a78b4a  main       -> origin/main
Updating 7e46066..3a78b4a
Fast-forward
 images/GUI.png             | Bin 0 -> 24545 bytes
 images/SaveCentralNew.png  | Bin 0 -> 65138 bytes
 images/SaveDistribNew.png  | Bin 0 -> 111885 bytes
 images/SaveLocalNew.png    | Bin 0 -> 26221 bytes
 images/devis.png           | Bin 0 -> 75387 bytes
 images/zones1.png          | Bin 0 -> 54719 bytes
 images/zones2.png          | Bin 0 -> 60191 bytes
 presentation.pdf           | Bin 5267356 -> 6625085 bytes
 presentation.tex           | 324 ++++++
```

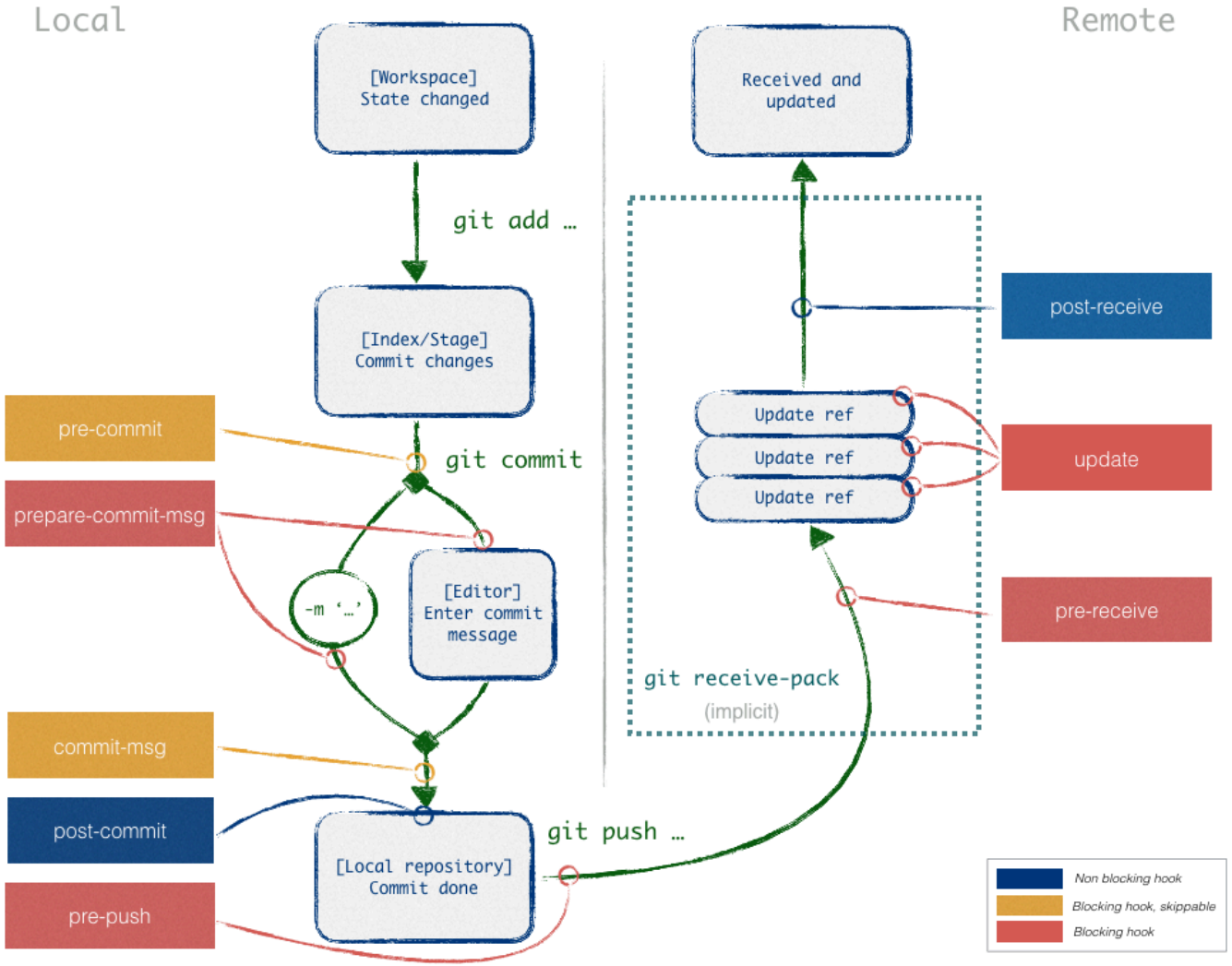
Une longue ligne de commande !

```
* 2186499 Update reset and go further topics
*   af46e63 Merge conflict
|\
| * 7e46066 Add to go further
| * 654b4e9 Change font size and add git SCM link
* | fd02601 Add clone command with uname and passwd
|/
* db8324e Remove pong image
* 7149293 Update revert and reset commands
* e8896ef Update merge and rebase commands
* 48b507e Global update 3
* 6b16542 Global update 2
* e44f652 Global update
* fe4cd53 Ajout des commandes rebase et merge
* b6f584b Add annexes
*   74a5511 Resolve conflict
|\
| * 8971e4d Some minor changes
* | e24286b Update projects
|/
* de0dca4 Remove spaces before $ in lstlisting env.
* dedb9fb Add credential and Signed-Off alerts blocks
* 8ff1a7f Update projects part
* 73cc805 Update projects and git commands
* f6ae20c Add main concepts slides
* c979f94 Add gitDAG library to draw nice git graph
```

```
$git log --graph --decorate --abbrev-commit --all --pretty=oneline
```

Un superbe graphe de commits dans la console ☺!

Les accroches - Hooks



Documentation Hooks sur Delicious Insights

Documentation Hooks sur Git

Git Alias

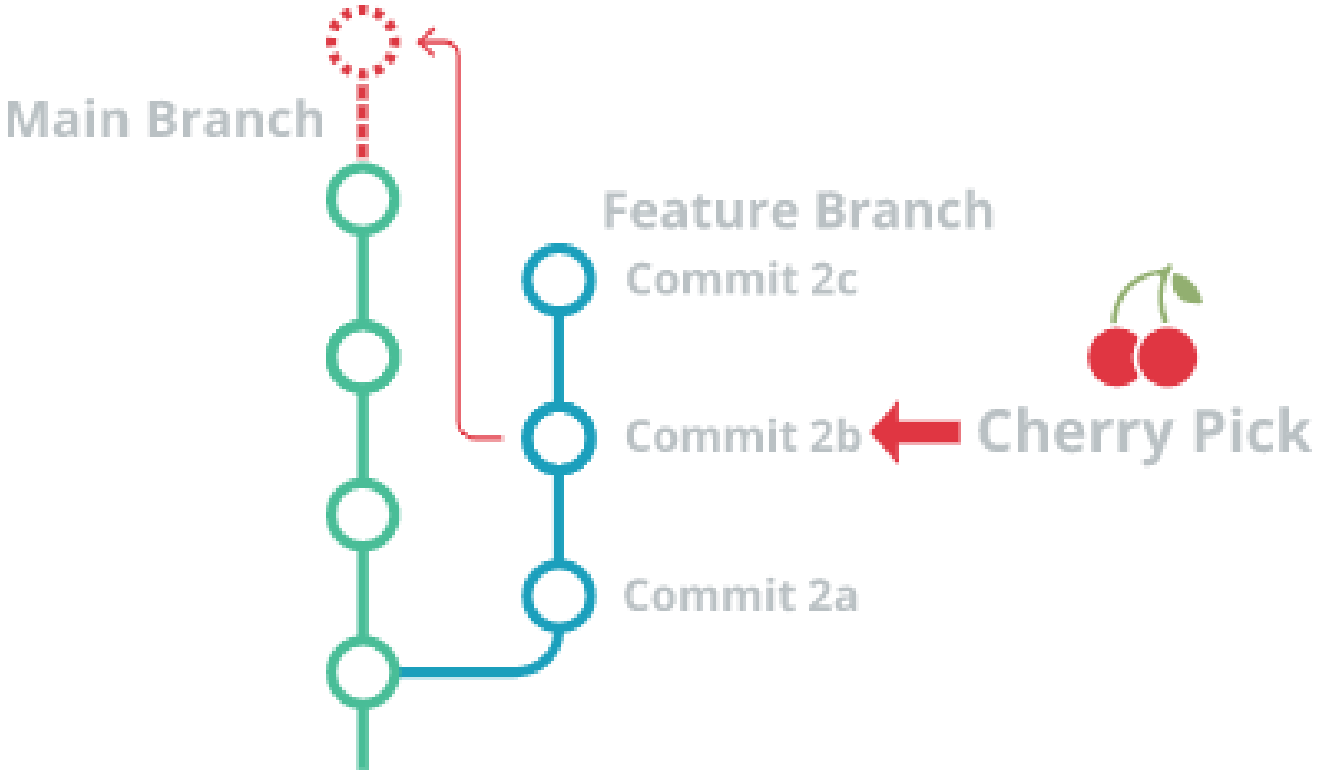
```
alias gs=' git status '
alias ga=' git add '
alias ga=' git push'
alias  c=' git commit '
```

Documentation Alias sur Git

Les alias en pratique

```
# Creating an alias to get  a
# graph of commits in console
$ git config --global alias.lg 'log␣--graph␣--decorate␣
    --abbrev-commit␣--all␣--pretty=oneline'
$ git lg
```

Cueillons des cerises !



"Cherry-pick" sur Delicious Insights

Cherry-pick en pratique

Une bonne pratique consiste à créer une branche de report. La Gestion de conflits/fusions est possibles. L'option -x permet d'écrire l'ID du commit initial dans le commit final ("cherry picked from commit commitID").

```
$ git switch rBranch # Branch which will receive picks
# Create and switch to a report branch from rBranch
$ git switch -c rBranchReport
$ git cherry-pick -x commitID1 # Pick oldest commit
$ git cherry-pick -x commitID2 # Pick next commit
$ git cherry-pick -x commitID3 # Pick next commit
$ # etc ... Manage possible conficts etc...
$ git switch rBranch # Switch to rBranch
$ git merge rBranchReport # Report rBranchReport
$ git branch -d rBranchReport # Delete rBranchReport
```

"La remise" sur Delicious Insights

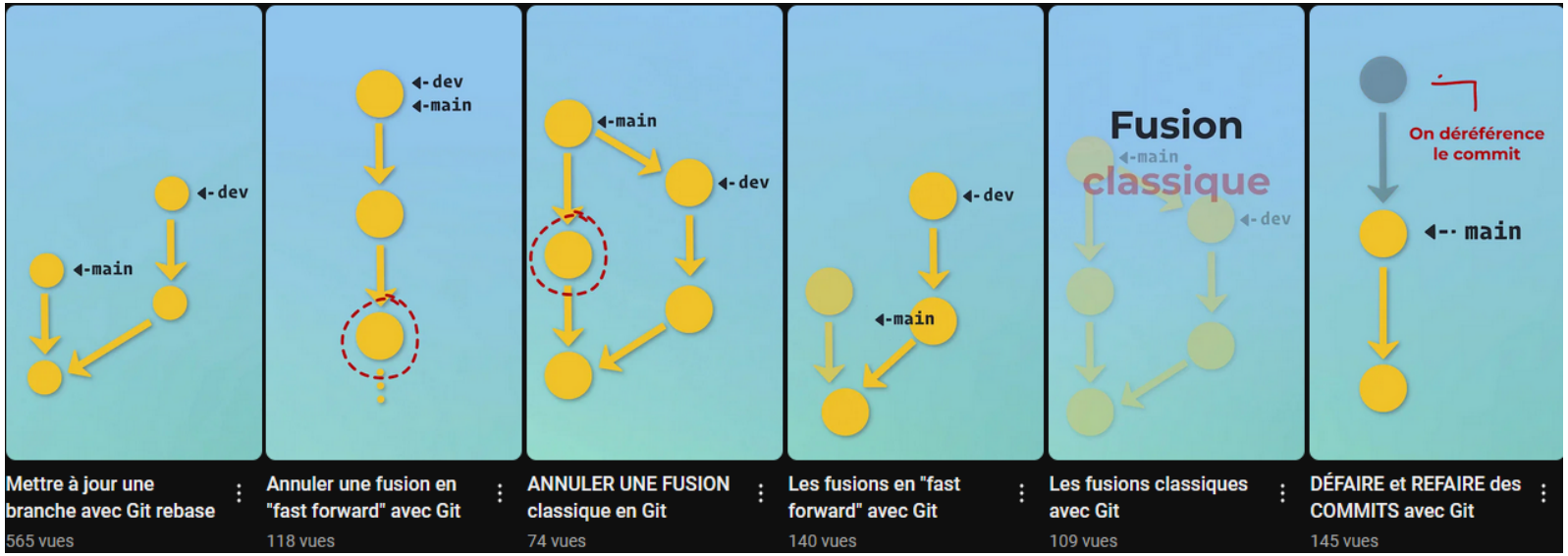
Comment dire à git d'ignorer des fichiers ?

"Ignorer des fichiers" sur Delicious Insights

Quel casse tête tous ces HEAD !

Les différentes formes de HEAD expliquées sur StackOverflow

Vidéos sur Delicious Insights



Shorts Git sur Delicious Insights