Institut National de Statistique et d'Economie Appliquée

DevOps
TP 2 : Github

I. Décentralisation

Objectifs :
- Le besoin, le pourquoi
- Github, présentation
- Deux approches
- Git clone
- Git remote

1. Créer un compte sur Github
2. Créer un nouveau repository en mode public ➔ New repository
3. Copier l'url de ce repository
4. Ouvrir votre dépôt local (TP1)
5. Ouvrir git Bash Here
6. Git clone <url-dépôt-distance>
7. Remarquer la création d'un nouveau projet avec un dépôt .git vide
8. Tester dedans git log ➔ Vérifier que c'est bien un dépôt vide
9. Git remote ➔ origin (un alias (un lien) qui est connecté avec le dépôt à distance)
10. Git remote get-url origin ➔ Vérifier que c'est bien le même lien de notre repository à distance
11. TortoiseGit > settings > remote > origin ➔ Une connexion simple avec origin via un url
12. Dans votre dépôt local du projet en cours (TP1), cliquer sur tortoiseGit > settings > remote ➔ Rien, pas de connexion pour l'instant
13. Revenir à Visual code en mode ligne de commande
14. Git remote add origin < url-dépôt-distance > ➔ Etablir la connexion avec le repository à distance avec remote
15. Git remote get-url origin ➔ On est bien connecté

II. Envoyer les commits à distance

Objectifs :
- Github, comment s'en servir ?
- Push de notre code
- Pull de notre code
- Fetch + rebase
- Pull avant push

1. Découvrir les différents onglets de github
2. Git push ➔ Pousser les commits sur le repository à distance ➔ Pas de branche similaire de master dans le dépôt à distance ➔ Il faut donc la créer
3. Git push -u origin master ➔ Envoyer le code sur une branche à distance qu'on appellera aussi master
4. Visualiser le résultat sur github
5. Rajouter un <h2> </h2> sur la branche master en local
6. Faite un git add et un git commit
7. Git log --pretty=oneline
8. Visualiser sur git graph
9. Rafraîchir sur github ➔ On n'a pas le dernier commit
10. Git push ➔ Renvoyer le dernier commit
11. Sur github > code > éditer le code en ajoutant un , ajouter un commentaire adéquat et commiter sur la branche principale master
12. Vérifier l'existence de ce dernier commit sur l'historique
13. Git pull ➔ Tirer les commits qu'il y a eu sur le repository à distance ➔ connecter le lien entre les deux dépôts et faire un merge ff = rebase (entre les deux historiques il n y a pas de différence)
14. Modification en local du dernier
15. Git add et puis git commit
16. Revenir sur github et modifier le code en enlevant des « ! » puis commiter cette modification ➔ On a à présent en local un commit et à distance un commit
17. En local faire un git pull ➔ Il fait un commit de merge. Un Pull fait un merge fast forward si l'historique est identique entre ce qu'il y a à distance et en locale (càd il n'y a pas de nouveau commit en local, par rapport au dépôt à distance). Si cependant c'est différent, il y a des commit en plus en local que ce qu'il y a eu à distance, Il va faire un merge de fusion avec un commit de merge ➔ Avec plein de branches, ça peut devenir complètement illisible et très dur à suivre.
    Comment le faire autrement ?
18. Rafraichir et éditer le code sur le serveur à distant ➔ Enlever un « ! »
19. Commiter cette modification
20. Git fetch ➔ Récupérer ce commit en local d'une autre façon sans faire le pull. Fetch va aller checker toutes les mises à jour mais ne va rien modifier sur les branches en local. Il fait une mise à jour sur la branche origin/master (l'enregistrement/image local de ce qui est à distance)
21. Git branch --all ➔ Visualiser les branches, à chaque branche distante, il crée une branche qu'on appelle Remote/origin/Nom-branche
22. Git rebase origin/master ➔ Récupérer tout ce qu'il y a sur la branche (origin/master) sur la branche principale locale master (prendre la branche locale master et se décaler sur la branche origin/master) ➔ Master et origin sont donc sur le même niveau
23. Faire un commit à distance avant un commit en local
24. Ajouter un commit sur le code à distance ➔ Changer le titre et commiter
25. Modifier le code en local ➔ Changer la langue par exemple (fr)
26. Git add et git commit

27. Git push ➔ Le dernier commit en local est supérieur au dernier commit non récupéré à distance ➔ Mettre à jour le dépôt local d'abord avant d'envoyer
28. Git pull ➔ Il faut donc récupérer le dernier commit à distance avant d'envoyer un nouveau commit au serveur
29. Git push

III. Faire du ménage avant de partager

Objectifs :
- Explications / besoins
- Git commit --amend
- Explication du HEAD + navigation
- Git rebase -i

1. Git log --pretty=oneline ➔ Visualiser le dernier message
2. Git commit --amend ➔ Modifier le message du dernier commit
3. Git log --pretty=oneline ➔ Voir la modification du message ➔ plus de clarté
4. En plus de la modification du dernier commit, ce qui serait encore plus intéressant c'est de pouvoir naviguer à travers les commits
   a. Visualiser le fichier HEAD dans le dossier .git ➔ Tête de lecture
   b. Git checkout <Nom-branche> ➔ Changer de branche
   c. Cat .git/HEAD ➔ Voit le HEAD ➔ La tête de lecture est un pointage sur la branche en cours, plus exactement sur le commit de la branche en cours
   d. Git checkout master
   e. Git checkout HEAD ➔ On a bien un lien entre HEAD et Master
5. Jusqu'à présent, nous utilisons le <N°_commit> comme source de référence pour faire pas mal de manipulations dans git comme :
   a. Git log <N°_commit> ➔ voir ce qui s'est passé dans ce commit
   b. Voir l'intérieur du commit en mettant le curseur sur le numéro de commit
   c. Git diff master <N°_commit> ➔ Comparaison
6. Nous pouvons également naviguer à travers les commits
   a. Git checkout <N°_commit> ➔ Aller directement sur un commit et donc déplacer le HEAD sur ce commit ➔ Un checkout détaché
   b. Cat .git/HEAD ➔ Ma tête de lecture a changé, elle est sur le commit
   c. Git log --pretty=oneline ➔ Remarquer la tête de lecture actuelle
   d. Git checkout HEAD^ ➔ Remonter de un
   e. Git checkout HEAD^ ➔ Continuer la navigation ➔ pas trop pratique
   f. Git checkout master ➔ Revenir à la tête de navigation
   g. Git checkout HEAD~ 3 ➔ Remonter de 3
7. Création d'une nouvelle branche pour tester rebase -i avec le HEAD
   a. Git checkout master
   b. Git checkout -b test-idea-page2 ➔ Nouvelle branche
   c. Créer un nouveau fichier Part1.html et générer dedans un bloc html
   d. Modifier le titre de ce fichier <title> </title>
   e. Git add puis git commit -m '' feat : second page "

f. Git commit --amend ➔ Modifier le message (seconde page)
g. Git log --pretty=oneline ➔ Visualiser la modification
h. Modifier le nom du fichier
i. Git add puis git commit ''fix : nom fichier''
j. Ajouter un titre <h1> </h1> et un commentaire < !--- test titre --->
k. Git add et git commit -m'' titre h1''
l. Enlever le commentaire ➔ Pas la peine que les autres développeurs voient mes commentaires
m. Git add et git commit -m''text : suppression commentaire''
n. Git log --pretty=oneline ➔ On commence à avoir pas mal de logs, ce qu'on aimerait c'est de pouvoir enlever tout ça, les trier ➔ les fusionner en un seul Commit
o. Git rebase -i HEAD~4 ➔ Interactive rebase sur les 4 derniers commits. On peut décider de les éditer, de supprimer le commit directement, de les squash
p. Choisir Squash des 3 commits ➔ Ca donne l'impression qu'il y a qu'un seul commit
q. Start rebase ➔ Proposer l'édition des messages ➔ le code est fusionné
r. Se positionner sur master et faire git merge --ff test-idea-page2 ➔ Récupérer le code récent

8. Tester le cas de la suppression (Drop)
   a. Git commit --allow-empty -m "marquer une étape"
   b. Git rebase -i HEAD^ ➔ Choisir Drop de ce commit
   c. Start Rebase
   d. Visualiser la disparition du dernier commit sur le git graph
9. Git push directement si possible
10. Sinon Git fetch puis Git rebase origin/master
11. Git push
12. Git branch -D test-idea-page2 ➔ Clean (supprimer la branche qui servait de test)

IV. Faire du ménage encore

Objectifs :
- Explication /besoins
- Git checkout / git restore
- Git reset /git reset HEAD
- Git reset --soft / git reset --hard
- Git revert

1. Ajouter des modifications dans index.html et le second fichier sans faire un git add ni git commit
   a. Git status ➔ il y a bien un changement (2 modifications)
   b. Git checkout ➔ Fait la même chose que status avec moins de détails ➔ savoir la modification de ce qui est uniquement en cours

c. Git checkout index.html ➔ Récupérer ce qu'il y a juste avant le commit (restaurer)
d. Git status ➔ Vérifier qu'il reste bien qu'une seule modification
e. Git restore <nom-fichier> ➔ Revenir à l'état avant commit
f. Git status ➔ Vérification

2. Annuler ce qu'il y a dans le staging
   a. Faire une modification sur le titre dans le fichier index.html
   b. Git status
   c. Git add --all
   d. Modifier le second fichier
   e. Git add --all
   f. Git status ➔ Les deux fichiers sont modifiés et ajoutés au staging
   g. Git reset HEAD ➔ Annuler tout ce qui est dans staging et repasser au working directory
   h. Git status
   i. Git add --all puis git reset ➔ Fait la même chose
   j. Git add --all
   k. Git restore --staged index.html ➔ Annule ce qui est en staging du fichier index.htm
   l. Git status ➔ Un fichier qui est toujours dans le staging, prêt à être commiter et un fichier qui est dans l'état working directory.
   m. Git add --all
   n. Git commit -m'' fix : Head''
3. Annuler les commits en local et pas l'historique partagé
   a. Git log --pretty=oneline
   b. Faire 2 ou 3 commits de plus par rapport à origin/master
   c. Modifier   en enlevant les …
   d. Git add --all && Git commit -m ''fix remove …''
   e. Git status
   f. Git reset --soft HEAD~ 2 ➔ Annuler les 2 derniers commits mais garder les fichiers et leurs modifications
   g. Git status ➔ Mes fichiers sont dans l'état staged, on va pouvoir les re-commiter
   h. Git commit -m ''fix : all''
   i. Git log --pretty=oneline
   j. Git reset --hard HEAD~1 ➔ Annule totalement le dernier commit et vire les modifications
   k. Git log --pretty=oneline
   l. Git status ➔ On a plus les modifications
4. Annuler un commit qui vient de l'historique qui est commun à l'ensemble des git partagés.
   a. Git log --pretty=oneline ➔ Choisir un <N°-commit> avant origin/master (commit partagé)

b. Git revert <N°-commit> ➔ Annuler un commit qui est sur l'historique commun ➔ Fait la mise à jour en créant un commit de revert
c. Git push
d. Git log --pretty=oneline