GitHub

Présentation

GitHub (exploité sous le nom de GitHub, Inc.) est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions **Git**.

Un logiciel de gestion de version permet de :

- suivre l'évolution d'un code source, pour retenir les modifications effectuées sur chaque fichier et être ainsi capable de revenir en arrière en cas de problème ;
- travailler à plusieurs, sans risquer de se marcher sur les pieds. Si deux personnes modifient un même fichier en même temps, leurs modifications doivent pouvoir être fusionnées sans perte d'information.

Installation

Le plus simple est de télécharger : https://git-scm.com/downloads (Laissez les options par défaut)

Il est nécessaire d'avoir un compte chez github et d'avoir été invité comme membre du projet: https://github.com/LE2P

Configuration sur IntelliJ

Installation depuis : https://www.jetbrains.com/idea/download

(Il est possible d'avoir une licence "éducation" gratuite pour la version Ultimate en faisant une demande : https://www.jetbrains.com/student/).

Au démarrage:

- Dans le menu config (en bas à droite), aller su "settings"
- Dans la barre de recherche, mettre "Git" et ensuite configurer les deux onglets "version

Control” nommés
    - “Git” : le fichier installé précédemment
    - “GitHub” : login et mdp

Ensuite pour importer un projet:

- Faire “Check out from version control”
- Choisir le projet que l’on veut importer
- Créer un projet à partir des fichiers importé
    - Dans le cas d’un projet Maven, chercher le sous repertoire contenant le pom.xml
    - Choisir le SDK approprié (JDK par exemple)

Pour mettre à jour le projet après modification,

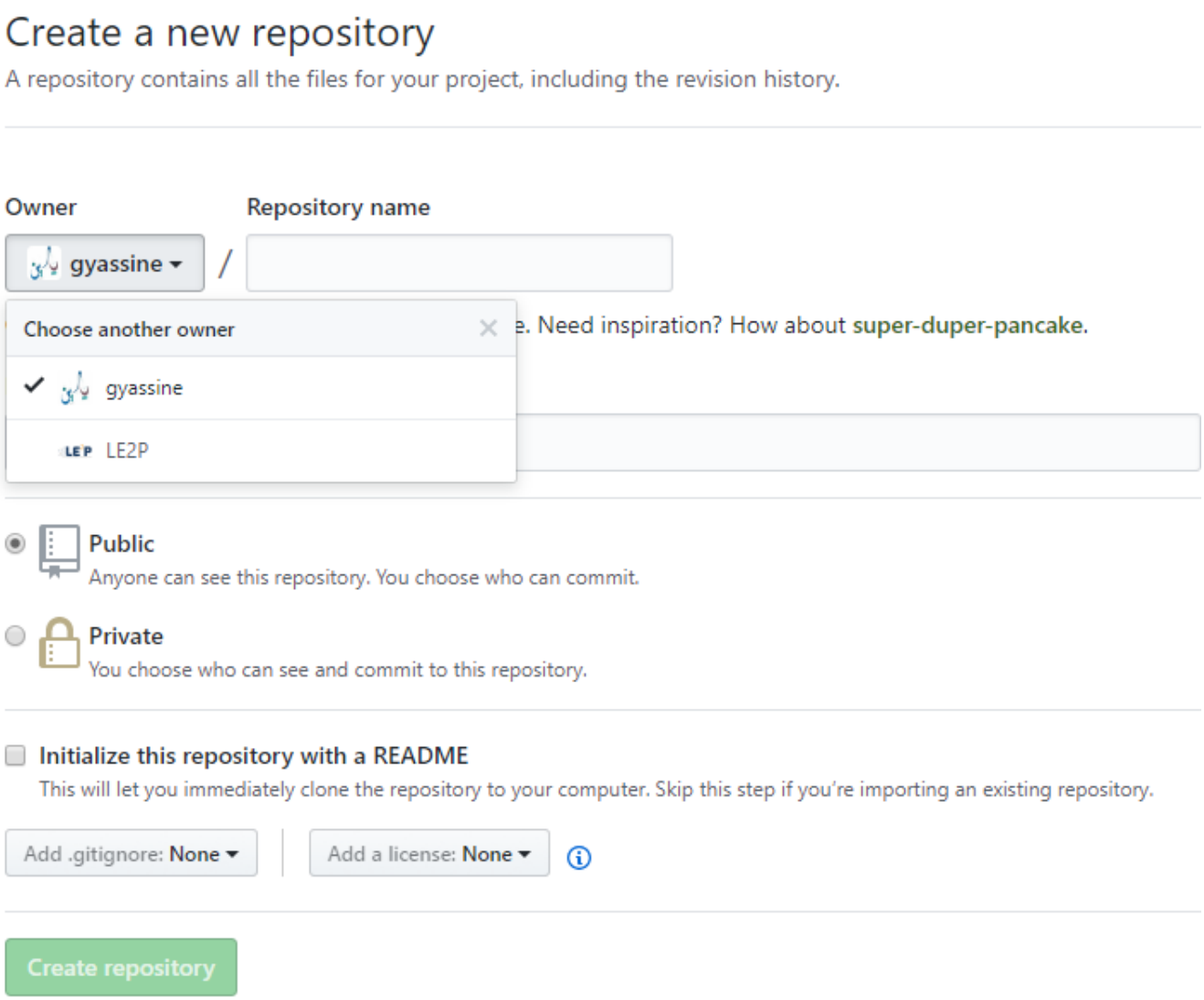
- Ouvrir le menu en faisant l’une des trois options :
    - Ctrl + K
    - Menu “VCS”, “Commit Changes”
    - Appuyer sur le bouton VCS avec la flèche verte vers le haut
- Puis Entrer un texte concernant la modification apporté au projet dans le champ “Commit messages” avec un message explicatif, par exemple qui comprend :
    - Une première ligne résumant le **pourquoi** du patch (pas le comment)
    - Une description longue optionnelle permettant d’**expliciter le contexte du résumé** donné en première ligne ;
    - Une liste de fichier et leurs modifications.
    - Une ligne vide sépare les différentes parties : la première est obligatoire et la troisième est optionnelle pour les changements triviaux
- En faisant juste un “commit”, on effectue le changement uniquement sur la machine. Le “commit & push” permet d’envoyer sur le serveur distant les modification.

Utilisation en ligne de commande

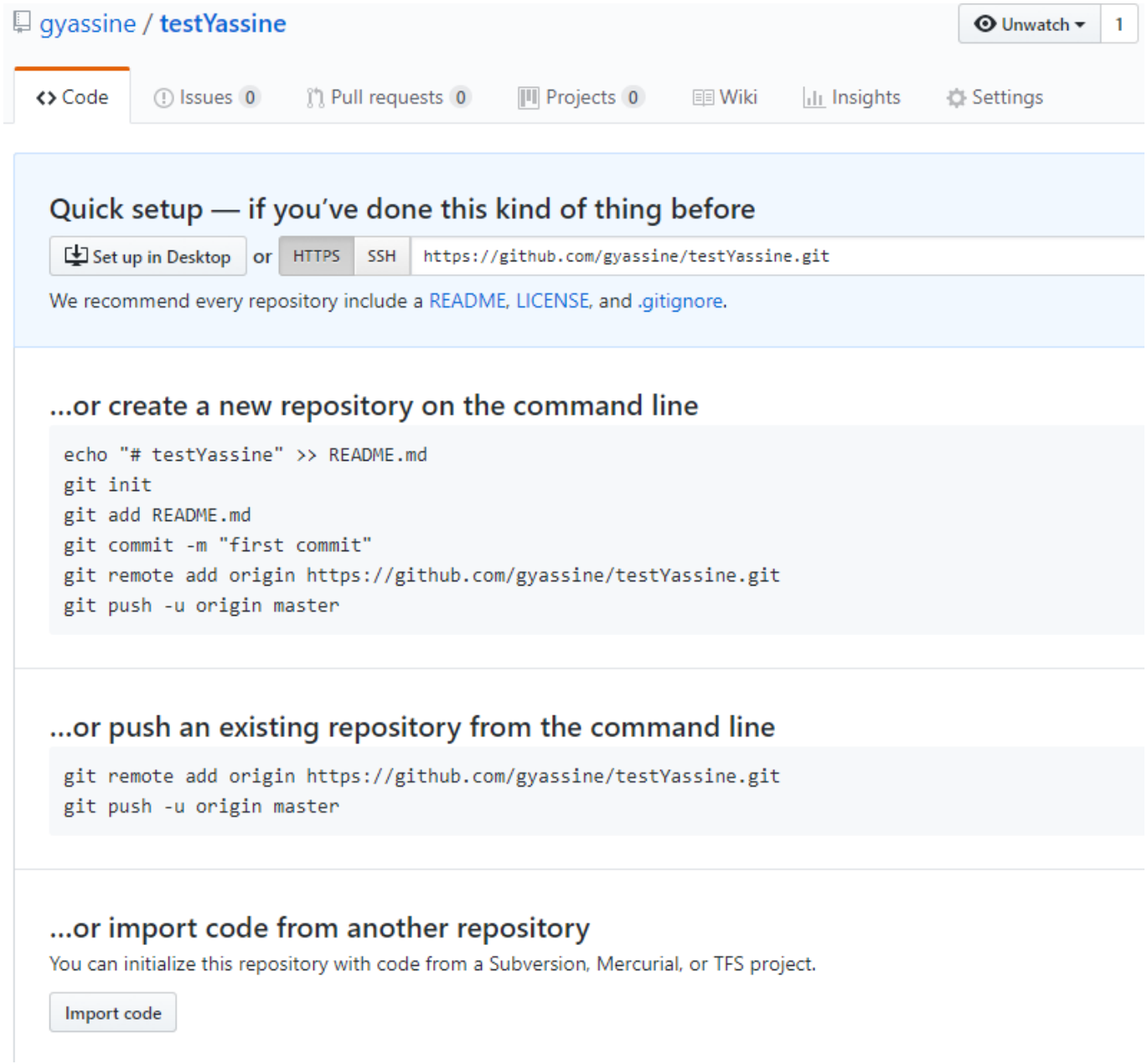
(pour les utilisateurs qui n’ont pas d’IDE dédié à cela comme par exemple MATLAB, etc.)

Déposer un projet existant sur GitHub

- Pré-requis :
    - Avoir un compte GitHub personnel
    - Appartenir à l’organisation “LE²P” sur GitHub (voir avec Mathieu ou Yassine)
- Méthode :
    - Se connecter sur le site de **GitHub**
    - Créer un dépôt et choisir “**Owner**” comme étant le **LE²P** ou soi-même.
        - **Ne pas mettre de README** (si on veut déposer des fichiers qui se trouve sur notre serveur/DD)
        - Mettre en privé (= privé à l’organisation ou bien à l’individu)



- Une fois que le dépôt est créé sur le site GitHub, il “suffit” de suivre les informations du site :



- Se mettre sur son serveur/disque dur (après avoir installer git en ligne de commande)
- Allez dans le répertoire voulu (par exemple `cd /var/www/`) dans lequel se trouve les fichiers à *pusher* sur le serveur

- `git init .` : permet de dire qu'on va pusher à partir de ce répertoire et tout les sous-répertoire en dessous
- A ce point on peut écrire un fichier `.gitignore` qui permettra d'exclure certains fichiers ou répertoire. Par exemple, pour exclure les deux repertoires **rep1** et **rep2** il va contenir :

```
/rep1/
/rep2/
```

- `git add -A` : permet d'ajouter tous les fichiers existants avec l'arborescence complete à partir du répertoire en cours (à l'exception de ce qui a été mentionné dans le `.gitignore`). Sinon on peut ajouter un fichier par un fichier avec la commande fournit dans les informations suivant la création du dépôt.
- `git commit -m "Mon premier commit"` : permet de **valider** les fichiers qui ont été ajouté (ou plus tard qui seront modifié) avec un commentaire. Il est important de bien commenter par la suite (voir section suivante).

- `git remote add origin https://github.com/gyassine/testYassine.git` : permet de dire que je décide de synchroniser mon git local (sur ma machine, avec mon premier commit) avec un git distant (celui de GitHub).
- `git push -u origin master` : permet de *pusher* (cad envoyer) les modifications apportés sur le site distant.

Ca y est, votre code est en ligne et synchronisé avec votre machine local !

Et ensuite ?

En partant du principe que nous n'avez d'IDE dédié, une fois que vous avez modifié vos fichiers, que faire ?

- Se replacer dans le même répertoire (par exemple `cd /var/www/`)
- Si besoin rajouter les nouveaux fichiers à la liste des fichiers à commiter en faisant
  - `git add nomfichier1 nomfichier2` :
  - `git add .` : rajoute les nouveaux fichiers les fichiers modifiés
  - `git add -A` : remet tout à jour (création, suppression, modification)
- Une fois que le commit est fait, il suffit de faire un `git push` pour publier sur le serveur

Un bon message de commit

Il est recommandé de faire régulièrement des commits, mais pas des push. Vous ne devriez faire un push qu'une fois de temps en temps (pas plus d'une fois par jour en général). Evitez de faire systématiquement un push après chaque commit. Pourquoi ? Parce que vous perdriez la facilité avec laquelle il est possible d'annuler ou modifier un commit.

La convention avec Git est de rédiger un message de commit comme on rédige un e-mail : une ligne courte de sommaire (la seule qui s'intéresse à la question « Quoi ? »), puis une ligne vide, puis un argumentaire rédigé expliquant pourquoi le changement est bon.

Le format est donc :

```
<ligne de sujet>

<un ou plusieurs paragraphe d'explications>
```

La ligne de sujet doit rester courte (si possible moins de 50 caractères, pour que la sortie de `git log –oneline` tienne dans un terminal de 80 colonnes). C'est l'équivalent du sujet d'un e-mail, et c'est cette ligne qui apparaîtra dans `gitk` ou `git log –oneline` par exemple.

Il faut une ligne blanche pour séparer la ligne de sujet, sinon Git va afficher tout le premier paragraphe partout où il aurait affiché la ligne de sujet.

La suite est une explication rédigée sur le commit. Ne pas hésiter à faire une explication longue si c'est nécessaire (typiquement plusieurs paragraphes).

Sauf cas particulier, on préférera donc rédiger un message de commit depuis son éditeur de texte préféré (lancé par défaut par `git commit`) à l'option `–message` (`-m`) de `git commit`.

Méthode de travail

Lorsqu'on travaille avec Git, on suit en général toujours les étapes suivantes :

1. modifier le code source ;
2. tester votre programme pour vérifier si cela fonctionne ;
3. faire un commit pour « enregistrer » les changements et les faire connaître à Git ;
4. recommencer à partir de l'étape 1 pour une autre modification

Une fois que cela est fini, on fait un push en fin de journée par exemple sur GitHub.

À titre indicatif, si vous travaillez toute une journée sur un code et que vous ne faites qu'un commit à la fin de la journée, c'est qu'il y a un problème (sauf si vous avez passé toute la journée sur le même bug). Les commits sont là pour « valider » l'avancement de votre projet : n'en faites pas un pour chaque ligne de code modifiée, mais n'attendez pas d'avoir fait 50 modifications différentes non plus !

Liens très utiles

- Apprendre GitHub en 15 min de manière interactifs (Anglais) (court)
- Gérez votre code avec Git et GitHub (Français) (long)
- git - petit guide (Fr) (très court)
- Bonne pratique du Git (Fr) (court)

CheatSheet

Github/git CheatSheet.pdf

Git Cheat Sheet

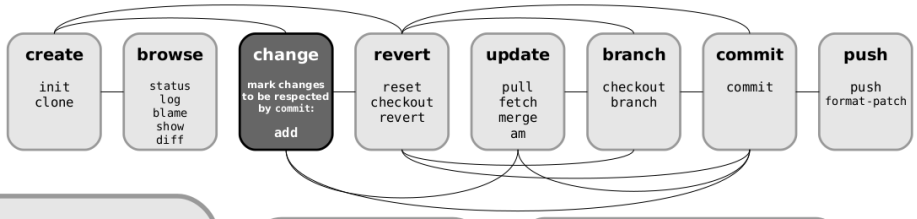
by Jan Krüger <jk@jk.gs>, http://jan-krueger.net/git/
Based on work by Zack Rusin

Basics

Use git help [*command*] if you're stuck.

`master`	default devel branch
`origin`	default upstream branch
`HEAD`	current branch
`HEAD^`	parent of HEAD
`HEAD~4`	great-great grandparent of HEAD
`foo..bar`	from branch *foo* to branch *bar*

(left to right) Command Flow



Create

From existing files
```
git init
git add .
```
From existing repository
```
git clone ~/old ~/new
git clone git://...
git clone ssh://...
```

Publish

In Git, `commit` only respects changes that have been marked explicitly with add.

```
git commit [-a]
```
(-a: add changed files automatically)
```
git format-patch origin
```
(create set of diffs)
```
git push remote
```
(push to origin or *remote*)
```
git tag foo
```
(mark current version)

Useful Tools

```
git archive
```
Create release tarball
```
git bisect
```
Binary search for defects
```
git cherry-pick
```
Take single commit from elsewhere
```
git fsck
```
Check tree
```
git gc
```
Compress metadata (performance)
```
git rebase
```
Forward-port local changes to remote branch
```
git remote add URL
```
Register a new remote repository for this tree
```
git stash
```
Temporarily set aside changes
```
git tag
```
(there's more to it)
```
gitk
```
Tk GUI for Git

Tracking Files

```
git add files
git mv old new
git rm files
git rm --cached files
```
(stop tracking but keep files in working dir)

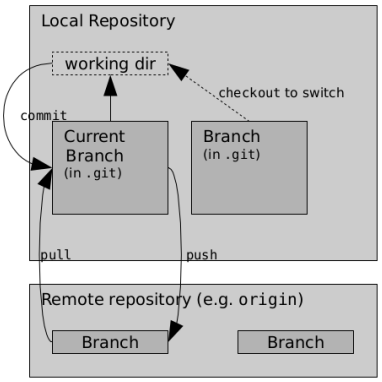
View

```
git status
git diff [oldid newid]
git log [-p] [file|dir]
git blame file
git show id (meta data + diff)
git show id:file
git branch (shows list, * = current)
git tag -l (shows list)
```

Update

```
git fetch (from def. upstream)
git fetch remote
git pull (= fetch & merge)
git am -3 patch.mbox
git apply patch.diff
```

Structure Overview



Revert

In Git, `revert` usually describes a new commit that undoes previous commits.

```
git reset --hard (NO UNDO)
```
(**reset** to last commit)
```
git revert branch
git commit -a --amend
```
(**replaces** prev. commit)
```
git checkout id file
```

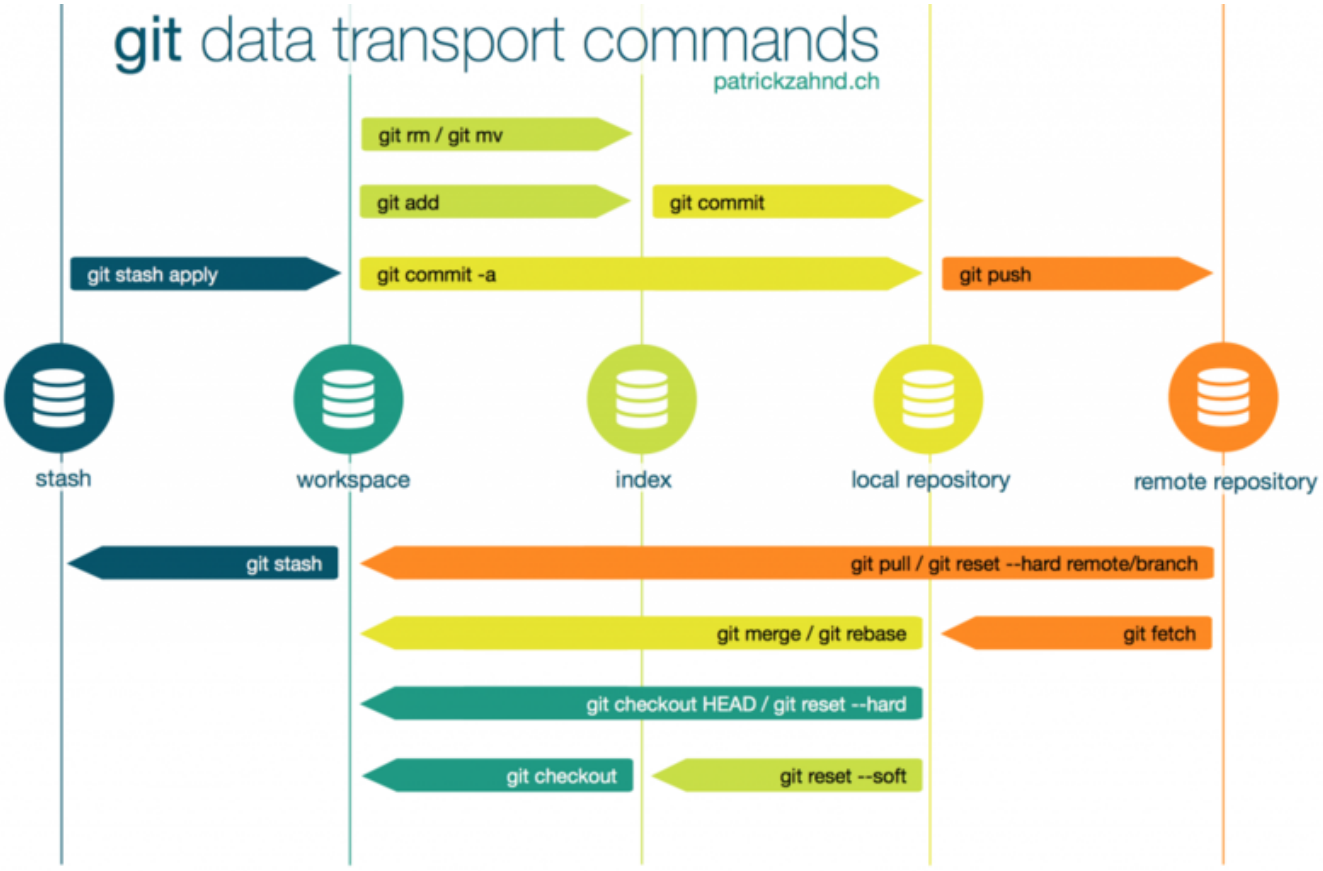
Branch

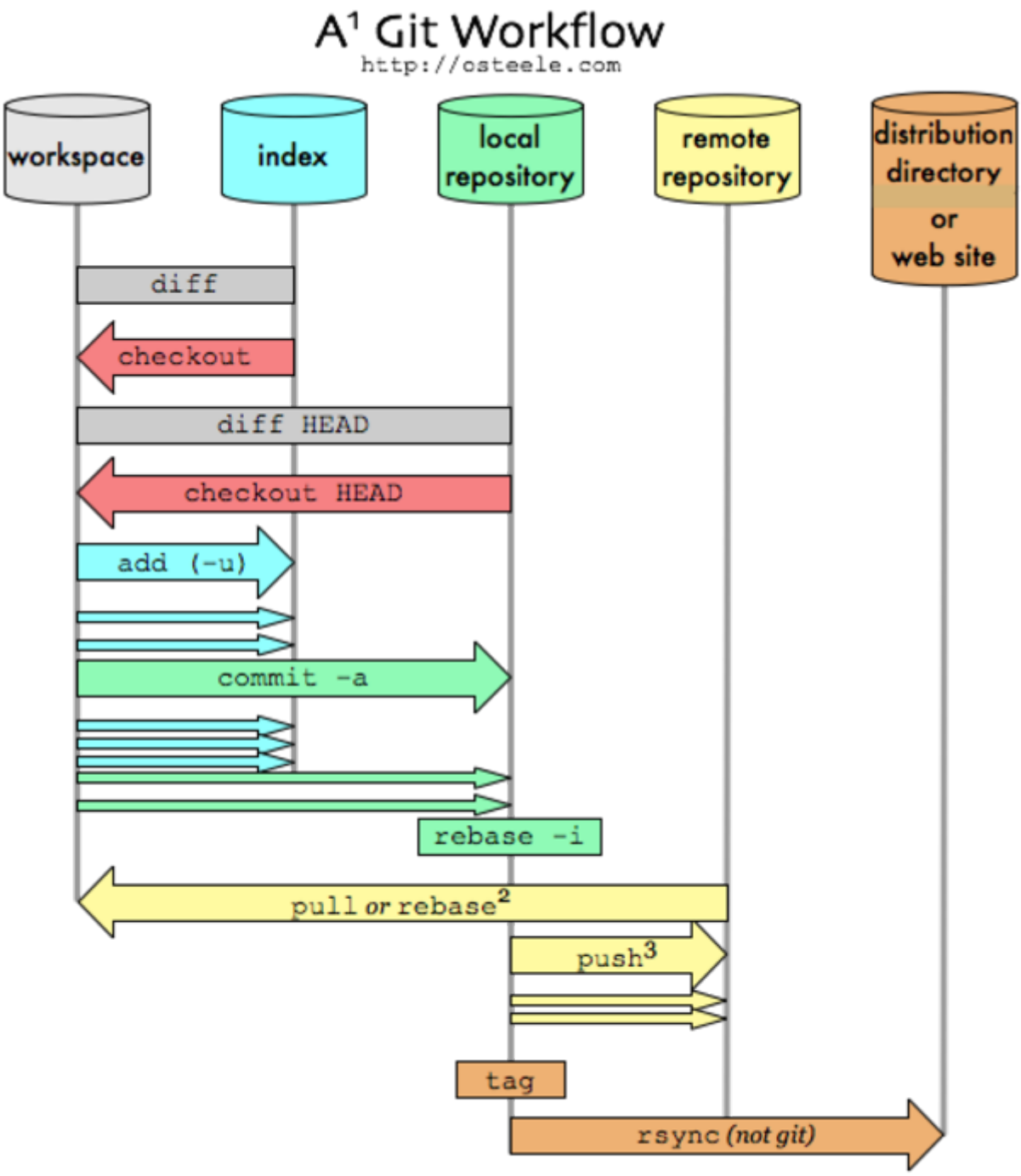
```
git checkout branch
```
(switch working dir to *branch*)
```
git merge branch
```
(merge into current)
```
git branch branch
```
(branch current)
```
git checkout -b new other
```
(branch *new* from *other* and switch to it)

Conflicts

Use add to mark files as resolved.

```
git diff [--base]
git diff --ours
git diff --theirs
git log --merge
gitk --merge
```





1 Git is a workflow construction toolkit. This is just one of many possible workflows.
2 With git-svn: "`git svn rebase`". With git-p4: "`git p4 rebase`"
3 With git-svn: "`git svn dcommit`"