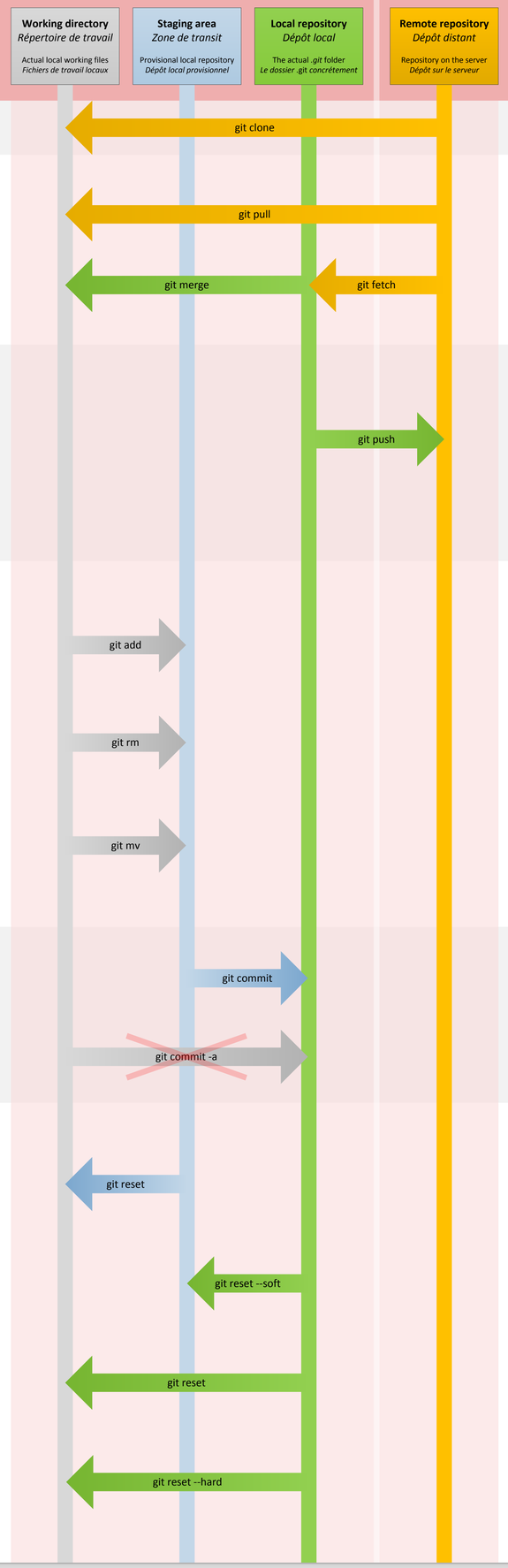
Data Transport Commands



Récupérer un dépôt distant complet
Action: `git clone <username>@eigit.heig-vd.ch:/home2/reds/<repository>`

Récupérer les dernières modifications du dépôt distant
Action: `git pull`

Cette commande est l'équivalent des deux commandes suivantes:
distant vers local `git fetch`
fusion avec répertoire de travail
`git merge`

Note: Si un problème survient lors de la fusion (merge), il est nécessaire de résoudre les problèmes:
- Soit avec la commande git mergetool, si cela est possible.
- Soit en récupérant les fichiers de précédents *commit* (cf. *checkout*)

Propager (pousser) les changement du dépôt local au dépôt distant.
Message *git status*:
«`Your branch is ahead of 'origin/master' by n commit.`»
Action: `git push`

Dry-run Utiliser l'option «-n» pour faire un «dry-run» (simulation: ne modifie rien)
First Lors du tout premier *commit*, on doit indiquer la branche *master* que l'on veut propager vers le dépôt distant *origin*. Cela va en réalité créer la branche *master* sur *origin*, car elle n'existe pas encore (premier *commit*).
Action: `git push origin master`

Remarque La branche *master* est la branche par défaut (nommée ainsi par défaut). Tandis que le nom *origin* indique le dépôt (par défaut).

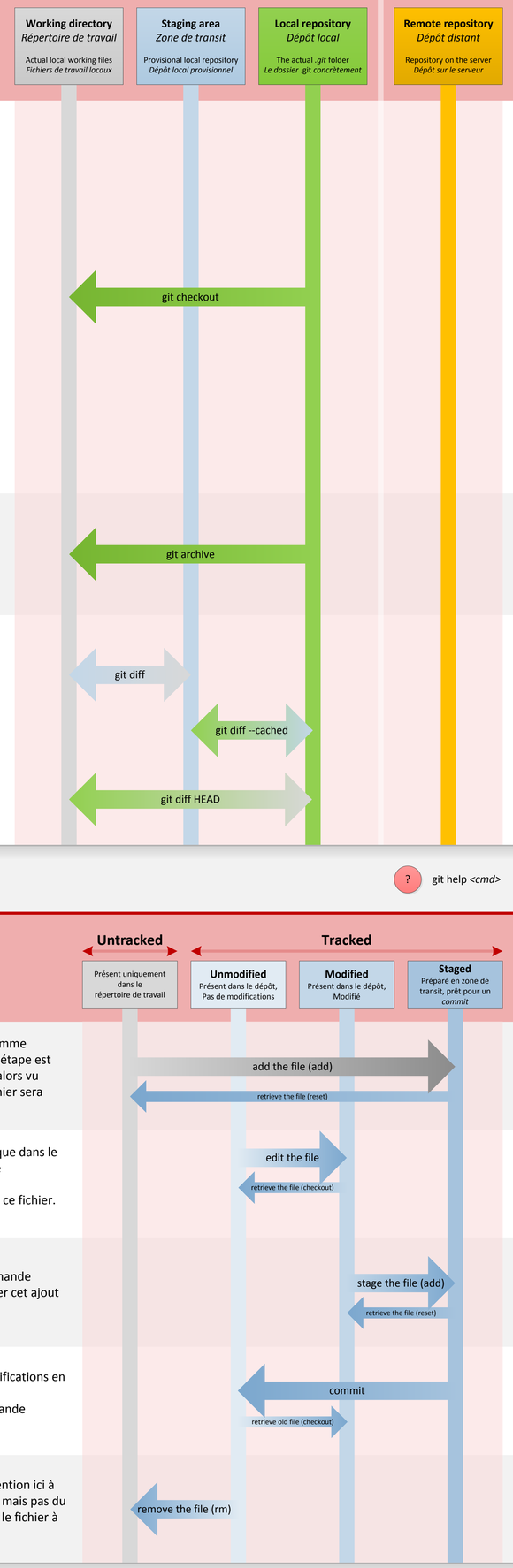
Préparation des changements (staging) à exécuter sur le dépôt local.
Remarque Pour éviter de devoir «trier» ce que l'on met dans le dépôt à chaque *commit*, il est conseillé d'utiliser les fichiers *.gitignore* (cf. section «Configuration and Tips & Tricks») afin de filtrer les fichiers non souhaités.
Message: «`Untracked files:`»
Certains fichiers (par exemple le fichier *my_file* ou un dossier *my_folder*) doivent être ajouté au dépôt:
Ajout d'un fichier `git add <my_file>`
Ajout d'un dossier `git add <my_folder>`
Message: «`Changes not staged for commit:`»
Certains fichiers, déjà dans le dépôt, ont été modifiés ou supprimés:
Action pour «modified» `git add <my_file>`
Action pour «deleted» `git rm <my_file>`
Action pour préparer (*stage*) toutes les modifications du dossier courant «.» (sauf les *untracked*):
`git add -u .`
Message: «`Unmerged path:`» (cf. commande *checkout*)
`git checkout --ours|--theirs <my_file>`
Déplacer ou renommer un fichier/dossier
Action: `git mv example.txt renamed_example.txt`
Tracked only Utiliser l'option «-u» pour ajouter uniquement les fichiers déjà présents dans le dépôt
Dry-run Utiliser l'option «-n» pour faire un «dry-run» (simulation: ne modifie rien)

Exécuter les changement sur le dépôt local.
Message *git status*: «`Changes to be commited`»
Action: `git commit -m "Commentaire"`

All ***Ne pas*** *utiliser l'option «-a» c'est une option pour les fourbes!*
Dry-run Utiliser l'option «-n» pour faire un «dry-run» (simulation: ne modifie rien)
Amend Utiliser l'option «--amend» pour faire un *commit* des changements en incluant les modifications du précédent *commit* (staging area). Cela signifie que les nouvelles modifications seront incluses dans le précédent commit (il n'y a pas de nouveau *commit*). Cela peut être utile pour ajouter des fichiers oubliés qui sont en relation avec le dernier *commit*.

Annulation d'un changement dans la zone de transit (avant *commit*) ou dans le dépôt local (après *commit*)
Staging area / Zone de transit: `git reset [--] <path>...`
Lorsque des modifications ont été préparées dans la zone de transit (add/rm/mv, avant un *commit*), celles-ci peuvent être annulées avec la commande *reset*
Action: `git reset example.txt`
Local repository / Dépôt local: `git reset [--soft | --mixed | --hard] [<commit>]`
Lorsque des modifications ont été faites au dépôt local (après un *commit*), il est possible d'annuler ces modifications. Cette commande accepte les options «--soft», «--mixed» et «--hard».

Dans les exemples suivants, on considère la branche *master* avec les *commit A—B—C*. Le passage de *B* à *C* est effectué avec les commandes (1) `git add my_file` et (2) `git commit`. Ainsi:
Soft Si la commande `git reset --soft B` est exécutée, la branche pointe alors vers *B*. Ni le répertoire de travail ni la zone de transit ne sont touchés. Ce qui signifie que les modifications de *C* sont affichées comme «staged» dans la zone de transit. Ainsi, un *commit* fera revenir à l'état *C*. En d'autres termes, l'option «--soft» annule la commande (1).
Mixed Si la commande `git reset B` est exécutée («--mixed» est l'option par défaut), la branche *master* pointe alors sur B. Le répertoire de travail n'est pas touché, cependant la zone de transit est modifiée pour concorder avec *B*. Ainsi, les modifications de *C* sont visibles comme «unstaged». Un simple *commit* ne suffira pas à revenir à *C*, il faudra en effet faire un *add*. En d'autres termes, l'option «--mixed» permet d'annuler les commandes (1) et (2).
Hard Si la commande `git reset --hard` est exécutée, la branche *master* point alors sur *B*. Le répertoire de travail ainsi que la zone de transit sont modifiée. Toutes les modifications entre *B* et *C* ainsi que toutes les changements non enregistrés du répertoire de travail sont supprimées. Il n'est plus possible de revenir à *C*. En d'autres termes, l'option «--hard» permet d'annuler les commandes (1) et (2) ainsi que toutes les modifications associées.
Remarques: Cette action écrase toutes les modifications courantes, il faut donc faire un *status* pour s'assurer que l'on accepte de perdre ces modifications. D'autre part, il **ne** faut **pas** utiliser cette option lorsque le commit *C* a déjà été enregistré sur le dépôt distant (*push*)!



Récupérer une branche ou un fichier/dossier
Actions sur les branches:
Switch Changer de branche `git checkout <branch_name>`
Merge Changer de branche en fusionnant les modifications de la branche courante
`git checkout -m <branch_name>`
Action sur un chemin (fichier/dossier)
Retreive Récupérer une version d'un ficher/dossier sur la branche courante:
`git checkout <file_path>`
Même action, mais en forçant le remplacement du fichier/dossier existant (modifié):
`git checkout -f <file_path>`

Récupérer une version, lors d'un conflit
Lorsqu'il y a conflit avec des fichiers binaires (modifié par deux personnes en local et en distant), la fusion n'est pas possible. Il faut donc choisir la version (locale ou distante) que nous souhaitons récupérer. Ce choix est possible en utilisant les options «--ours» ou «--theirs», respectivement pour récupérer la version locale ou distante d'un fichier.

Local Pour récupérer la version qui se trouve sur le dépôt local:
`git checkout --ours <file>`
Distant Pour récupérer la version qui se trouve sur le dépôt distant:
`git checkout --theirs <file>`

Créer une archive d'un dossier
La commande archive permet de créer une archive d'un dossier avec le contenu présent sur le dépôt local
Action: `git archive -format=zip HEAD:path/to/folder > archive_name.zip`

Remarque Il est possible de générer d'autres types d'archives, cf. `git help archive`

Afficher les différences
cf. paragraphe Status Commands

Modifications dans le répertoire courant, pas encore enregistrées dans la zone de transit
`git diff`

Modifications entre les modifications enregistrées dans la zone de transit et le dernier *commit*.
`git diff --cached`

Modification entre le répertoire courant et le dernier *commit*.
`git diff HEAD`

? git help *<cmd>*

File Status Lifecycle

Un fichier peut avoir deux états: *untracked* ou *tracked*. Les fichiers dans l'état *tracked* peuvent être vus comme *unmodified*, *modified* ou *staged*.

Lorsqu'un fichier est créé dans le répertoire de travail, il est considéré comme *untracked*, c'est-à-dire qu'il n'est pas encore dans le dépôt. La prochaine étape est donc d'ajouter le fichier dans la zone de transit (`git add`). Le fichier est alors vu comme «new file» (cf. commande *status*). Dès le prochain commit, le fichier sera alors *tracked unmodified*.

Un fichier déjà présent dans le dépôt a l'état *tracked*. Lorsqu'il est identique dans le répertoire de travail ainsi que dans le dépôt local, il est considéré comme *unmodified*.
Ainsi, la commande *git status* n'affichera aucune information concernant ce fichier.
Dès sa modification, par contre, il sera considéré comme *modified*.

Un fichier *tracked modified* est noté «modified» lorsqu'on utilise la commande *status*. On peut alors l'ajouter à la zone de transit (*stage*). On peut annuler cet ajout en utilisant la commande *reset*.

Une fois le fichier préparé en zone de transit, on peut alors figer ces modifications en les propageant au dépôt local. C'est là que le *commit* entre en scène.
On peut revenir à une version précédente du fichier en utilisant la commande *checkout*.

La fin de vie d'un fichier, dans le dépôt, se traduit par la fonction *rm*. Attention ici à utiliser l'option «--cached» si l'on souhaite supprimer le fichier du dépôt, mais pas du répertoire de travail. Pour annuler cette suppression, il faut alors ajouter le fichier à nouveau.

Git Quick Guide 09.13 – This work is licensed under a *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.*

Status Commands

La commande *status* indique l'état du répertoire de travail par rapport au dépôt local. Il indique également si le dépôt local a des *commit* nécessitant d'être enregistrés sur le dépôt distant.
Les informations affichées par la commande contiennent l'état des fichiers (cf. *File Status Lifecycle*), les commandes utiles par rapport au contexte, ainsi que des informations utiles comme la branche courante, le nombre de commit non synchronisés avec le dépôt distant, ou les conflits nécessitants l'attention de l'utilisateur.

«Changes not staged for commit»
Fichier dans l'état *tracked* qui ont été soit modifiés (*modified*), soit supprimés (*deleted*).

Pour mettre à jour dans la zone de transit (*stage* pour le prochain *commit*)
```
git add <file>...
git rm <file>...
```
Pour annuler les modifications dans le répertoire de travail
```
git checkout -- <file>...
```

«Untracked files»
Donne la listes des fichiers/dossiers qui ne sont pas présents dans le dépôt local.

Pour inclure dans la zone de transit (*stage* pour le prochain *commit*)
```
git add <file>...
```

«Changes to be committed»
Liste des fichiers étant dans la zone de transit, prêts à être enregistrés dans le dépôt local (*commit*).

Pour enregistrer les modifications dans le dépôt local
```
git commit -m "Message du commit"
```
Pour supprimer les modifications de la zone de transit (*unstage*)
```
git reset HEAD <file>...
```

«Your branch is ahead of 'origin/master' by *n* commits:»
Le dépôt local a été modifié par *n commit* qui n'ont pas encore été propagés au dépôt distant.

Tout d'abord, il faut récupérer les possibles mises à jour du dépôt distant
```
git pull
```
Ensuite, on peut propager nos modifications (*commit*) sur le dépôt distant
```
git push
```

«On branch master»
Indique la branche actuellement utilisée

Pour changer de branche
```
git checkout <branch>
```
Pour plus d'information, voir le bloc «Branches et *commit*»

Branches et *commit*

Explication des branches et *commit*:

<branch>
Une branche est une série de *commit*. La branche principale est appelée «master». Il est possible de créer d'autres branches, par exemple, afin de faire des modifications temporaires, avant de les fusionner (*merge*) avec la branche principale.

<commit>
Un *commit* est un état d'une branche. Le commit courant se nomme «HEAD». Il est possible d'atteindre d'anciens *commits* en se référant au *commit* courant. Par, exemple, il est possible de revenir à deux *commits* en arrière. Il est également possible de référencer les *commits* en utilisant un «tag», afin de facilité la possibilité de changer de version.

Dans les diagrammes si contre,

La position du HEAD, le commit courant, est indiquée par un panneau. Le nom sur le panneau est le nom de la branche courante.

La position du prochain commit, après la commande marquée en couleur, est indiqué par un panneau de la même couleur

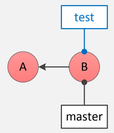
Un commit est indiqué par un rond. La chronologie est de gauche à droite: ainsi, le commit «A» est le premier *commit*. Les couleurs différencient les branches

Les relations entre les commit sont indiquées par les flèches noires. Cela n'indique pas le sens chronologique, mais la liaison: l'ancêtre du commit «B» est le commit «A».

Les flèches en traitillé large et les traits larges, tous deux de couleur, sont l'illustration de la commande indiquée dans le bloc à droite de la même couleur.

Création d'une nouvelle branche

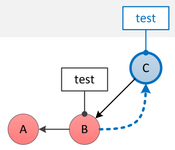
Après deux *commit* («A», le premier, puis «B») nous sommes sur la branche principale (par défaut) *master*.
La commande *checkout*, utilisée avec l'option «-b» permet de changer de branche (ici *test*).
Pour autant, nous restons encore sur le même *commit* car il n'existe pas encore de *commit* dans la branche nouvellement créée.




```
git checkout -b test
  -or-
git branch test
git checkout test
```
N.B.: La commande *checkout* utilisée avec l'option «-b» est équivalente à la succession des commandes *branch* (création de la branche) et *checkout* (sélection de la branche).

Commit dans la nouvelle branche

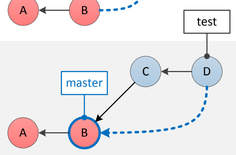
Après quelques modifications dans le répertoire de travail, on fixe ces modifications avec un *commit*. Le nouveau *commit* ainsi créé est sur la branche *test*.




```
git commit -m "commit
on test branch"
```

Sélection d'une autre branche

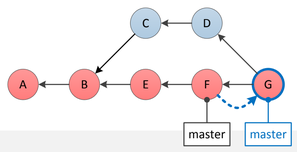
Après d'autres modifications sur la branches tests (et un *commit*), on souhaite revenir à la branche *master*.




```
git checkout master
```

Fusion de deux branches

Après quelques modifications et quelques *commit*, on veut fusionner les modifications effectuées dans les deux branches *master* et *test*.
Les éventuels conflits peuvent être résolus avec *mergetool* (cf. «Configuration and Tips & Tricks).

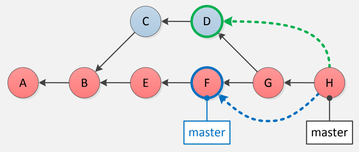



```
git merge test
```

Exemple de déplacements

Comme indiqué plus haut, la commande *checkout* suivie du nom d'une branche permet de sélectionner le dernier *commit* d'une branche. (cf. commande en vert)

Il est également possible de se déplacer dans la branche courante en se basant sur le commit actuel HEAD. La commande en bleu indique comment reculer de deux *commit* dans la branche courante (ici *master*)

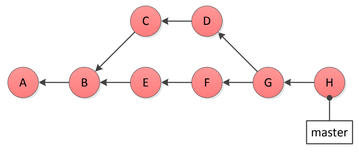



```
git checkout test

git checkout HEAD~2
  -or-
git checkout HEAD^^
```
HEAD suivit d'un tilde et d'un nombre *n* ou
HEAD suivit de *n* circonflexes signifient
«*n commit* avant HEAD»

Suppression d'une branche

Il est possible de supprimer une branche. Après cette opération la référence à cette branche n'existe plus, cependant l'historique des modifications (c'est-à-dire les *commit*) sont encore présents. Il est **important** de noter qu'avant d'effectuer cette opération, la branche doit avoir été fusionnée avec la branche en amont (dans l'exemple, la branche *master*).




```
git branch -d <branch>
```
Sur le dépôt **local**: Supprime la branche nommée <branch>

```
git push origin :<branch>
```
Sur le dépôt **distant**: Recherche une référence nommée <branch> et la supprimer

Configuration and Tips & Tricks

Gitignore

Un dépôt Git prend tout son sens lorsqu'il est utilisé pour des sources de code en format texte. En effet, premièrement, les fichiers binaires sont difficilement gérables. Deuxièmement, il est inutile de garder des versions différentes de fichiers générés. Ainsi, si l'on prend un dossier d'un programme en C, par exemple, on souhaitera garder les sources (`*.c`, `*.h`, etc.) tandis qu'on voudra ignorer les fichiers générés (`*.o`, `*.a`, `*.so`, etc.). Il en va de même pour les fichiers temporaires (`*.tmp`, `*~`, etc.) ou les binaires qui peuvent être stockés sur d'autres médiums plus adaptés (par les installeurs qui peuvent être mis sur FTP).
Ainsi, il est possible de créer des règles permettant d'ignorer certains fichiers dit «untracked» afin de ne pas les enregistrer sur le dépôt. Ces règles sont créées dans un fichier nommé «.gitignore». Celui-ci peut être placé à la racine du dépôt ou dans certains dossiers spécifiques. Sa portée s'étend aux fichiers et dossiers récursivement à partir de son emplacement. Il peut être utilisé en philosophies «ignorer seulement certains» ou «tout ignorer sauf certains».
Chaque ligne du fichier peut contenir un *pattern* utilisé comme filtre. Ce pattern peut-être une expression régulière. Une ligne de commentaire est initiée par un dièse «#».
Finalement, ne pas oublier le slogan **«Do not ignore .gitignore rule!»**

«Ignorer seulement certains»
```
## Backup or temporary files
*.old
*.tmp*
*~
(...)

## Temp folders
tmp
temp
(...)

## Generated files
*.[oa]
*.ko
*.so
(...)
```

«Ignorer tout sauf certains»
```
# Ignore all...
/*

# Except some folder
!/*.data/runs
!/*.data/sources*

# Or some files
!*.xise
!*.ucf
!*.vhd
!*.do
```

Help

La commande *help* permet d'afficher l'aide de *git*.

Afficher l'aide de *git*
```
git help
```
Afficher l'aide d'une commande spécifique
```
git help COMMAND
```

Aliases

Il est possible de créer des alias de commande. La commande en exemple à droite rend les deux commandes suivantes équivalentes:
```
git unstage fileA
git reset HEAD fileA
```

Création d'un alias pour que «unstage» signifie «reset HEAD --»
```
git config –alias.unstage 'reset HEAD --'
```

Log

La commande *log* permet d'afficher la liste des derniers *commit* effectués (journal). Il existe beaucoup d'options pour cette commande. Quelques unes sont montrées ici. Par exemple, l'option «-n» affichera uniquement les *n* derniers *commit*. L'option «--since» permet de limiter l'affichage aux *commit* effectués dans le laps de temps donné.

Affiche les 3 derniers *commit*
```
git log -3
```
Affiche les *commit* des deux dernières semaines
```
git log --since=2.weeks
```
Affiche les commit liés au fichier spécifié
```
git log -- <file>
```

Blame

La commande *blame* montre la révision et l'auteur des dernières modifications apportées sur chaque ligne d'un fichier. Le nom de cette commande est explicite: elle permet en effet de connaître la personne qui a fait la modification afin de la blâmer!

Affiche l'origine des lignes 40 à 60 d'un fichier:
```
git blame -L 40,60 <file>
git blame -L 40,+21 <file>
```
Afficher les modifications effectuées depuis 3 semaines
```
git blame –since=3.weeks -- <file>
```

Difftool

Similaire à la commande *diff* (cf. *Data Transport Commands*), la commande *difftool* permet de comparer des fichiers en utilisant un logiciel de comparaison tels que *Beyond Compare* ou *Meld*. L'outil de comparaison lié à cette commande peut être configuré via la commande *config*.

Lancement de l'outil de comparaison
```
git difftool [<commit> [<commit>]] [--] [<path>…]
```
Configuration pour *Beyond Compare* (Windows)
```
git config --global diff.tool bc3
git config --global difftool.bc3.path "<path_to>/bcomp.exe"
```
Configuration pour *Beyond Compare* (Linux):
http://www.scootersoftware.com/support.php?zz=kb_vcs#gitlinux
Configuration pour *Meld* (Linux)
```
git config --global diff.tool meld
```

Mergetool

La commande *mergetool* permet de démarrer l'outil de résolution des conflits. Cet outil est utilisé lorsqu'un *pull* est fait et que certains fichiers ASCII on été modifiés sur les dépôts local et distant. L'outil de comparaison lié à cette commande peut être configuré via la commande *config*.

Lancement de l'outil de résolution des conflits
```
git mergetool [<file>…]
```
Configuration pour *Beyond Compare* (Windows)
```
git config --global merge.tool bc3
git config --global mergetool.bc3.path "<path_to>/bcomp.exe"
```
Configuration pour *Beyond Compare* (Linux)
```
git config --global merge.tool bc3
git config --global mergetool.bc3 trustExitCode true
```
Configuration pour *Meld* (Linux)
```
git config --global merge.tool meld
```

Config

La commande *config* permet de modifier la configuration de *git* de manière locale (liée au dépôt courant) ou globale (configuration globale du programme *git*).
L'on peut, par exemple, activer la coloration des messages de *git*; configurer le nom de l'utilisateur, son adresse e-mail; configurer l'éditeur de texte par défaut, l'outil de comparaison (cf. mergetool, difftool); modifier l'url du dépôt distant; etc.

Colorer les messages de *git*
```
git config --global color.ui auto
```
Configuration de l'utilisateur
```
git config --global user.name "Gilles Curchod"
git config --global user.email "gilles.curchod@heig-vd.ch"
```
Configuration du dépôt distant
```
git config --replace-all remote.origin.url
      <username>@eigit.heig-vd.ch:/home2/reds/<repository>
```

Remote

La commande *remote* permet de gérer les informations liées au dépôt distant.
Elle sera utile en autre pour modifier l'adresse du dépôt disant pour le dépôt courant.

Modifier l'adresse du dépôt distant
```
git remote set-url origin
      <username>@eigit.heig-vd.ch:/home2/reds/<repository>
```

Sources

Git Help Command	La commande `git help <command>` ouvre la version local des «Git Manual Pages»
Git (Pro Git version HTML)	http://git-scm.com/book/fr
Pro Git	http://git-scm.com/downloads (quelques copies papier disponibles dans l'institut)
Git Manual Pages	https://www.kernel.org/pub/software/scm/git/docs/

Git Quick Guide 09.13
Institut REDS – Gilles Curchod

ReDS
Reconfigurable & Embedded Digital Systems