**Aide-mémoire git**

**Aide-mémoire pour les commandes git**

**J’ajoute une ligne**

**J’ajoure la deuxième ligne**

**<http://alainericgauthier.com/git/aide_memoire_git>**

Ces articles font partie d'une série rédigée par moi-même. git est un outil de contrôle de version que je trouve de plus en plus utile pour gérer mes projets de développement web. Comme je devais toujours retourner à la documentation pour me rappeler les commandes, j'ai décidé de me faire un aide-mémoire. Comme le contenu est très résumé, il n'y a donc pas de détails sur le fonctionnement ou les particularités d'une commande. De plus, j'ai choisi une approche séquentielle typique pour présenter l'information.

* [Commencer avec git](http://alainericgauthier.com/git/commencer_avec_git)

**Commencer avec git**

Cet article présume que git est déjà installé sur votre ordinateur. Vous verrez ici les commandes utilisées le plus souvent.

**Premiers pas**

Avant de commencer à faire des validations (commit en anglais), il faut démarrer le dépôt git.

À partir de la ligne de commande, il faut d'abord se placer à l'intérieur du dossier du projet. Il faut ensuite initialiser le dossier git.

$ git init

**Le cycle**

Le cycle, tel que je le conçois, consiste à :

* réviser les modifications apportées,
* ajouter les fichiers qu'on veut valider (ou enregistrer),
* faire la validation,
* recommencer.

Lorsque des modifications sont faites aux fichiers contenus dans le dossier du projet, il est possible de voir ces modifications :

$ git diff

Il faut ensuite ajouter les fichiers que l'on veut inclure à la prochaine validation.

$ git add <un ou des noms de fichiers>

Finalement, on valide. Le message que vous inscrivez est très important car c'est ce qui permettra de savoir ce qui a été fait au moment de cette validation.

$ git commit -m '<message de commit>'

Il existe un raccourci pour valider et ajouter tous les fichiers automatiquement :

$ git commit -a -m '<message de commit>'

**Autres commandes utiles**

Pour annuler la plus récente validation :

$ git reset --hard

**Attention**, cette commande efface de manière permanente toutes les modifications.

Pour voir l'historique de tous les commits :

$ git log

Pour voir quels sont les fichiers qui sont modifiés et ceux qui seront inclus dans le prochain commit :

$ git status

Pour en savoir plus sur git, voyez mon [aide-mémoire git](http://alainericgauthier.com/git/aide_memoire_git).

* [Commencer avec les branches](http://alainericgauthier.com/git/commencer_les_branches)

# Commencer avec les branches dans git

## Branchement

Pour créer une nouvelle branche :

$ git branch [nom de la branche]

Pour se positionner sur cette nouvelle branche :

$ git checkout [nom de la branche]

Il existe un raccourci pour faire la même chose en une seule étape :

$ git checkout -b [nom de la branche]

Une fois que vous avez fait des modifications sur votre branche, vous devez faire un commit :

git commit -a -m 'changements apportés sur la nouvelle branche'

## Fusion

Pour fusionner un branche avec la branche master :

$ git checkout master

$ git merge [nom de la branche]

Si vous n'avez pas fait des modifications qui se contredisent dans les deux versions d'un même document, la fusion devrait se faire sans problème et ce, même si vous avez fait des changements dans les deux branches.

Si vous avez fait des changements à la même ligne d'un même document dans les deux branches, vous devrez gérer un conflit de fusion (merge conflict). Ceci fera partie d'un autre article.

Par TEB (non vérifié) le 19 Janvier, 2018 - 08:12

vous avez oublié le tiret '-' devant 'a' pour add.

git commit -a -m 'changements apportés sur la nouvelle branche'

Pour en savoir plus sur git, voyez mon [aide-mémoire git](http://alainericgauthier.com/git/aide_memoire_git).

* [Gérer les conflits de fusion](http://alainericgauthier.com/git/gerer_les_conflits_de_fusion)

**Gérer les conflits de fusion avec git**

Note : cet article fait partie d'une série sur git que j'ai nommé mon [aide-mémoire git](http://alainericgauthier.com/git/aide_memoire_git).

**Situation particulière :**

* Vous avez créé une branche dev à partir de la branche master.
* Vous avez fait un commit sur la branche fix après avoir modifié l'élément body du fichier index.html.
* Vous êtes revenu sur la branche master, vous avez fait fait un commit après avoir fait des modifications sur les mêmes parties du même élément.
* Vous tentez de faire un merge :

$ git merge dev

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

**Conflit!**

git indique qu'il y a un problème avec le fichier index.html

Pour voir quel(s) fichier(s) sont affecté par le conflit de fusion, il faut faire :

$ git status

[...]

# unmerged: index.html

[...]

git crée automatiquement un fichier dans lequel le conflit est identifié.

<<<<<<< HEAD:index.html

<body id="home" lang="en">

=======

<body id="home" lang="fr">

>>>>>>> dev:index.html

### Résolution

Il faut donc aller éditer manuellement le(s) fichier(s) pour régler le conflit. La premère ligne avec une suite de chevrons (<<<<<<<) indique la source du texte qui suit. La série de signes d'égalité (=======) est un séparateur.

La dernière ligne avec une suite de chevrons (>>>>>>>), indique l'origine du texte qui précède. La première partie vient du fichier sur lequel nous sommes possitioné, la seconde partie vient de la branche appelée par notre commande merge.

Une fois le conflit résolu (et il faut bien vérifier), il faut faire la commande add :

$ git add index.html

Finalement, quand tous les fichiers sont corrigés et ajoutés, il faut faire un dernier commit.

$ git commit -m "Ajouts de lang="fr" sur l'élément body."

Il y a aussi la possibilité d'utiliser un outil pour gérer les conflits de fusion, mais ça c'est pour un autre article.

Pour en savoir plus sur git, voyez mon [aide-mémoire git](http://alainericgauthier.com/git/aide_memoire_git).

* [Ignorer des fichiers avec git](http://alainericgauthier.com/git/ignorer_des_fichiers_avec_git)

**gnorer des fichiers avec git**

En utilisant la commande git status, vous réaliserez rapidement qu'il y a des fichiers qui doivent se trouver dans le dossier de votre projet mais que vous ne voulez pas inclure dans votre archive git. Dans un projet de développement web, il pourrait s'agir par exemple d'une librairie javascript que vous devez inclure mais que vous ne modifiez pas ou encore d'une photo en format jpg.

**Ignorez-moi!**

Il est possible de demander à git d'ignorer des fichiers en créant un fichier .gitignore.

Les règles sont les suivantes:

* Les lignes vides ou commençant par # sont ignorées
* Les motifs glob fonctionnent
* Un motif qui se termine par / indique un répertoire
* On peut faire une négation en commençant l'expression par !

**Exemples**

Par exemple, si votre libraire javascript se trouve dans un dossier nommé script, la ligne suivante permettra à git d'ignorer tout le contenu de ce dossier :

# Ignorer la libraire javascript

script/

Les motifs glob (glob patterns en anglais) sont des caractères spéciaux qui fonctionnent un peu comme les expressions régulières d'un shell.

* \* remplace 1 ou plusieurs caractères
* ? remplace 1 caractère
* [abc] indique qu'on cherche 1 des 3 caractères (soit a, b ou c)
* [0-9] indique une valeur parmi toutes les valeurs entre la première et la dernière (dans ce cas-ci, tous les chiffres entre 0 et 9).

En combinant ces éléments, il est possible d'avoir un contrôle assez raffiné sur les fichiers ignorés.

\*.txt

!TODO.txt

Ces 2 lignes indiquent à git d'ignorer tous les fichiers texte sauf le fichier TODO.txt.

Pour en savoir plus sur git, voyez mon [aide-mémoire git](http://alainericgauthier.com/git/aide_memoire_git).

* [Modifier le dernier commit avec git](http://alainericgauthier.com/git/modifier_dernier_commit)

# Modifier le dernier commit avec git

Je présente ici l'option --amend de la commande commit. Cette commande permet de refaire le dernier commit. Elle est très utile pour récupérer une erreur.

## Faute de frappe

Il m'arrive souvent de faire un commit et lors de la rédaction du commentaire, de faire une faute de frappe. Et ça me faisait rager à chaque fois, jusqu'à ce que je découvre cette commande :

git commit --amend

Je fais toujours des fautes de frappe dans mes commentaires de commit mais maintenant je peux les corriger facilement.

git commit -m "Commentère mal écri ..."

git commit --amend

Le commentaire mal écrit apparait et il est possible de le modifier avec l'éditeur de texte.

## Fichier manquant

Il m'est parfois arrivé de faire un commit et d'oublier d'ajouter un fichier. Encore une fois, la commande git --amend vient maintenant à mon secours.

git commit -m "Voici un commit"

git add fichier\_ajouter.txt

git commit --amend

Lors de l'exécution de la troisième commande, le commit est refait mais en incluant le fichier ajouté. L'ancien commit est remplacé par le nouveau et le commentaire reste identique.

Pour d'autres articles sur cet outil de contrôle de version, voyez mon [aide-mémoire git](http://alainericgauthier.com/git/aide_memoire_git).

**Installation**

git est un programme qui fonctionne en ligne de commande mais il ne vient pas avec les systèmes d'opérations courants. Voici des liens aux versions les plus récentes.

* [Installation de git pour Mac OS 10.5+](http://code.google.com/p/git-osx-installer/downloads/list?can=3)
* [Installation de git pour Windows](http://code.google.com/p/msysgit/)
* [Installation pour Linux et autres](http://code.google.com/p/git-core/downloads/list)

**Documentation**

Comme tous les outils très puissants, git vient avec un certain niveau de complexité. La documentation qui vient avec le programme sera sûrement cryptique pour quelqu'un qui n'est pas familier avec le contrôle de version. Par contre, il existe de très bonnes ressources pour s'initier au contrôle de version avec git.

**En français**

* [pro git, ce livre est en cours de traduction, mais il est complètement gratuit!](http://progit.org/book/fr/)
* [git, sur ubuntu.fr, est une introduction en français qui offre un survol des commandes utilisées fréquemment](http://doc.ubuntu-fr.org/git)

**En anglais**

* [La section documentation sur git-scm.com offre une liste de liens pour se familiariser avec git et pour en apprendre plus sur les fonctions avancées](http://git-scm.com/documentation)
* [Le manuel de git est aussi disponible en format html](http://schacon.github.com/git/git.html)

[**github**](https://github.com/)

[git hub](https://github.com/) est un site qui permet de partager ses projets avec une communauté. Si vous choisissez de garder votre projet entièrement public, il n'y aucun frais pour y héberger un projet.

**Git**

**Git** est un logiciel de gestion de versions décentralisé. Il est conçu pour être efficace tant avec les petits projets, que les plus importants.

Git a spécialement été créé pour le développement du noyau linux.

Ce projet a débuté en 2005, Linus Torvalds voulait créer une alternative au logiciel propriétaire BitKeeper. Depuis, Git a beaucoup évolué et est utilisé par de nombreux projets.

Contrairement à des outils comme SVN ou CVS, Git fonctionne de façon décentralisée, c'est-à-dire que le développement ne se fait pas sur un serveur centralisé, mais chaque personne peut développer sur son propre dépôt. Git facilite ensuite la fusion (*merge*) des différents dépôts.

Haut du formulaire

Bas du formulaire

**Installation**

Pour pouvoir utiliser Git, il suffit d'[installer le paquet](http://doc.ubuntu-fr.org/tutoriel/comment_installer_un_paquet) [**git**](apt://git).

Git dispose également de nombreux plugins, les plus utilisés sont dans les dépôts :

* [git-svn](apt://git-svn) : Gestion des dépôts [SVN](http://doc.ubuntu-fr.org/subversion) ;
* [git-cvs](apt://git-cvs) : Gestion des dépôts [CVS](http://doc.ubuntu-fr.org/cvs) ;
* [git-track](apt://git-track) ;
* etc.

Une fois l'installation réalisée, il faut impérativement définir le paramètre [user]. Pour cela, [éditez le fichier](http://doc.ubuntu-fr.org/tutoriel/comment_editer_un_fichier) caché **.gitconfig** se trouvant dans votre dossier personnel :

[user]

email = p.nom@ubuntu-fr.org

name = nom\_programmeur

[alias]

ci = commit

co = checkout

st = status

br = branch

Les alias quant à eux permettent de raccourcir les commandes, exemple : git st au lieu de git status

Haut du formulaire

Bas du formulaire

**Utilisation basique**

Haut du formulaire

Bas du formulaire

**Gérer les dépôts**

Avant de commencer à utiliser Git, il faut lui demander de créer un nouveau dépôt.

mkdir nom\_depot

cd nom\_depot

git init

Dans le répertoire nom\_depot, vous aurez alors un dossier caché .git, c'est dans ce dossier que Git stockera les différentes révisions et informations du projet.

Vous pouvez aussi récupérer un dépôt déjà existant et travailler à partir de celui-ci en faisant

git clone git://<dépot>

ou

git clone https://<dépot>

Vous pourrez ensuite (si vous avez les droits suffisants sur le dépôt distant), envoyer vos changements avec git push.

Haut du formulaire

Bas du formulaire

**État du dépôt**

git diff

git diff <commit1> <commit2>

Permet de comparer 2 versions. Vous pourrez ainsi voir les changements effectués. Si vous avez des changements pas encore *commités*, la commande git diff affichera les modifications effectuées depuis le dernier *commit*.

git status

Savoir tout ce qui n’a pas encore été validé

git log

Liste les *commits* effectués dans le dépôt. Vous pourrez ainsi voir les modifications qui ont été faites.

Haut du formulaire

Bas du formulaire

**Gestion des fichiers**

git add <nom\_fichier\_ou\_dossier>

Cette commande indique à Git que le fichier (ou dossier) nommé "nom\_fichier\_ou\_dossier" devra être versionné.

git add --all \*

Permet d'ajouter tout le contenu (fichier et dossier) du dossier.

git rm <nom\_fichier>

Supprime le fichier de votre ordinateur, ainsi que du dépôt Git.

git mv <nom\_fichier> <nouvelle\_destination>

Déplace le fichier.

Haut du formulaire

Bas du formulaire

**Gestion des commits**

git pull

Met à jour votre dépôt local (à faire avant de commencer à modifier des fichiers pour être sûr de travailler sur leurs dernières versions et avant tout *commit* pour éviter les éventuels conflits avec des modifications effectuées par d'autres utilisateurs entre temps).

git commit <fichier1> <fichier2>

Crée un *commit* contenant fichier1 et fichier2. Ces fichiers auront dû être au préalable ajoutés au dépôt avec la commande git add.

("Il s'agit de la validation d'une transaction. Après avoir commis la transaction, les informations traitées par cette transaction seront disponibles pour les autres sessions, c'est-à-dire pour toute autre transaction éventuelle."<http://fr.wikipedia.org/wiki/Commit> )

git commit -a

Crée un nouveau *commit* contenant tous les changements effectués sur les fichiers (git add n'est donc pas nécessaire avant un commit -a).

git push origin master

Envoie le *commit* dans la branche principale "master" du dépôt "origin".

Haut du formulaire

Bas du formulaire

**Commandes d'annulation**

Git dispose de commandes permettant d'annuler des changements effectués. **Attention, ces annulations ne sont pas réversibles** !

| **Commande** | **Effet** |
| --- | --- |
| git reset –hard HEAD | Annule les changements effectués depuis le dernier *commit*. |
| git reset –hard HEAD^ | Supprime le dernier *commit*. Cette action peut être répétée autant de fois que vous le désirez. |
| git revert commit | Restaure le dépôt tel qu'il l'était lors du *commit* spécifié. |
|  | Pour que cette commande fonctionne, il faut que toutes les modifications soient *commitées* |
|  | (ou annulées avec git reset). |

Haut du formulaire

Bas du formulaire

**Utilisation avancée**

Généralement, quand on utilise Git, on ne travaille pas seul mais en équipe. Voici les principales commandes qui vous aideront à utiliser Git dans de telles situations. Ici, nous supposerons qu'on travaille avec deux dépôts Git distincts nommés respectivement moi et bob. Le dépôt moi est votre dépôt, bob est celui d'une autre personne, admettons qu'il est situé à l'adresse git://github.com/bob

Haut du formulaire

Bas du formulaire

**Gestion des branches**

Git permet une gestion efficace des branches, et des *merges* (fusion de branches). Les branches permettent d'avoir simultanément plusieurs versions de votre programme dans votre dépôt Git. C'est très utile, par exemple pour développer une nouvelle fonctionnalité, tout en gardant la branche principale intacte. Ainsi, vous pouvez toujours faire des changements dans la branche principale (corrections de bugs par exemple), tout en développant en parallèle une nouvelle fonctionnalité.

git branch test

Crée une nouvelle branche nommée "test".

git branch

Liste les branches présentes dans le dépôt. Le nom de la branche courante est précédé du caractère "\*".

git checkout <nom\_branche>

Change de branche. Après cette commande, vous aurez alors accès aux fichiers présents dans la branche nommée "nom\_branche". Avant de changer de branche, il faut impérativement que tous les changements effectués soient *commités* ou annulés, sinon Git refusera de changer de branche.

git merge <nom\_branche>

Fusionne la branche courante avec la branche nommée "nom\_branche". Il se peut qu'il y ait des conflits et que Git ne soit pas capable de les résoudre tout seul. Les conflits apparaissent lors de changements divergents au même endroit dans un fichier. Si il y a des conflits, Git laissera un marquage directement dans le fichier, contenant le code de la branche courante, et celui de la branche que vous voulez fusionner. Vous devrez alors corriger le problème manuellement. Une fois corrigé, vous devez *commiter* les changements pour finaliser le *merge*.

Si jamais vous décidez d'abandonner la fusion, alors exécutez la commande :

git reset --hard HEAD

Haut du formulaire

Bas du formulaire

**Récupération des changements**

Imaginons que Bob ait implémenté une nouvelle fonctionnalité. Vous voulez naturellement l'intégrer à votre dépôt.

git remote add bob git://github.com/bob

git fetch bob

git merge bob/master

La première commande crée un alias qui fait pointer bob vers l'adresse du dépôt. Ça permet d'éviter d'avoir à taper l'adresse complète à chaque fois. La deuxième commande récupère les changements que bob a effectués. La troisième commande fusionne les changements de bob avec votre branche courante. Il existe une commande qui a le même effet que git fetch suivi de git merge :

git pull bob

Haut du formulaire

Bas du formulaire

**Voir aussi**

Haut du formulaire

Bas du formulaire

**Interfaces graphiques**

* **(en)** [Giggle](https://live.gnome.org/giggle) (GTK+)
* **(en)** [git-age](https://github.com/krig/git-age), interface graphique pour la commande [git blame](http://www.alexgirard.com/git-book/5_trouver_les_probl%25C3%25A8mes_-_git_blame.html) (PyGTK)
* **(en)** [git-cola](http://git-cola.github.com/) (PyQt)
* **(en)** [GitEye](http://www.collab.net/giteyeapp) (propriétaire ?)
* **(en)** [GitForce](https://github.com/gdevic/GitForce) (Mono)
* [gitg](apt://gitg) (GTK+)
* [git-gui](apt://git-gui) et [gitk](apt://gitk), interfaces graphiques (Tcl/Tk) livrées avec Git
* [qgit](apt://qgit) (Qt)
* **(en)** [qgrit](https://github.com/qgrit/qgrit/wiki), interface graphique pour la commande [git rebase -i](http://www.alexgirard.com/git-book/4_recombinaison_interactive.html) (Qt)
* **(en)** [RabbitVCS](http://www.rabbitvcs.org/) (PyGTK)
* **(en)** [SmartGit](http://www.syntevo.com/smartgithg/) (propriétaire ?)
* **(en)** [StupidGit](https://github.com/gyim/stupidgit/wiki) (wxPython)
* **(en)** [Team Git](http://www.devslashzero.com/teamgit) (Qt)
* **(en)** [Tig](http://jonas.nitro.dk/tig/), interface en mode texte
* **(en)** [Ungit](https://github.com/FredrikNoren/ungit) Interopérable (Linux, Mac, Windows) et moderne
* **(en)** [GitKraken](https://www.gitkraken.com/) Linux, Windows et Mac

Haut du formulaire

Bas du formulaire

**Autres outils autour de Git**

* **(en)** [EGit](http://www.eclipse.org/egit/), extension ajoutant le support de Git dans Eclipse
* **(fr)** [Gitolite](http://doc.ubuntu-fr.org/gitolite): outil de gestion de dépôts Git avec utilisateurs, droits, etc.
* **(fr)** [Gogs](https://gogs.io/): serveur de gestion de dépôts comme Gitolite avec interface http(s) inclus par défaut.
* **(en)** [Magit](http://philjackson.github.com/magit/), intégration de Git dans Emacs.
* **(en)** [Fugitive](http://www.vim.org/scripts/script.php?script_id=2975), Intégration de Git dans Vim.
* [Gource](http://doc.ubuntu-fr.org/gource), Visualiser l'activité de votre dépôt dans une animation qui traverse le temps.
* **(en)** [thunar-vcs-plugin](http://goodies.xfce.org/projects/thunar-plugins/thunar-vcs-plugin), permet l'accès aux dépôts Git et [Subversion](http://doc.ubuntu-fr.org/subversion) via [Thunar](http://doc.ubuntu-fr.org/thunar). Le paquet [thunar-vcs-plugin](apt://thunar-vcs-plugin) est disponible pour **Ubuntu versions** [**12.04**](http://doc.ubuntu-fr.org/precise) **& +**.
* [kdesdk-dolphin-plugins](apt://kdesdk-dolphin-plugins) permet l'utilisation de l'overlay des dépôt de versionning (pas uniquement git) dans [dolphin](http://doc.ubuntu-fr.org/dolphin). (activable par services dans les paramètres)
* **(en)** [Git-flow](https://github.com/nvie/gitflow), gestion avancées des branches dans une logique projet : **(en)**[Un modèle de branches concluant pour Git](http://nvie.com/posts/a-successful-git-branching-model/). Le paquet [git-flow](apt://git-flow) est disponible pour **Ubuntu versions** [**12.04**](http://doc.ubuntu-fr.org/precise) **& +** **(dépôts "universe")**.
* **(fr)** [GITg](http://actual-it.info/2013/gitg/), GIT en mode graphique - Tutoriel d'utilisation de GITg
* **(en)** [git-lfs Large File Storage](https://git-lfs.github.com/), gestion des fichiers lourds associés au projet: fichiers audio, vidéo, jeux de données (test…), graphiques… qui sont déposés sur un serveur à part.

Haut du formulaire

Bas du formulaire

**Principaux sites concernant Git**

* **(en)** [Le site officiel de git maintenu par Scott Chacon](http://git-scm.com/) ;
* **(fr)** [le livre en français sur le site officiel](http://git-scm.com/book/fr);
* **(en)** [Le site officiel de git maintenu par Petr Baudis](http://git.or.cz/index.html) ;
* **(en)** [Manuel utilisateur Git](http://www.kernel.org/pub/software/scm/git/docs/user-manual.html) : Une référence très complète ;
* **(fr)** [Le livre communautaire (libre et gratuit;) entièrement dédié à Git](http://www.alexgirard.com/git-book/index.html) ;
* **(fr)** [Git immersion en français](http://gitimmersion.fr), un des guides références anglais traduit en Français

Haut du formulaire

Bas du formulaire

**Principaux dépôts Git**

* [GitHub](http://github.com) : très bon dépôt gratuit. Une bonne gestion du travail en équipe, avec la possibilité de forker un dépôt, pour contribuer au développement sur son propre dépôt, tout en gardant un contact étroit avec le dépôt original.
* [Gitorious](http://www.gitorious.org/) : utilisé par de gros projets comme le framework Qt. A noter que le site web gitorious est un logiciel libre et donc installable sur son propre serveur à l'inverse de GitHub qui est propriétaire. Les fonctionnalités communautaires des 2 me semblent assez proches.
* [BitBucket](https://bitbucket.org/) : concurrent de Github. Propose de faire des *dépôts privés* gratuits (jusqu'a **5 utilisateurs** sur le même dépôt. Pas d'autre restriction).
* [GitLab](https://github.com/gitlabhq/gitlabhq) : logiciel open-source de dépôt, à installer sur sa machine [Manuel d'installation](https://github.com/gitlabhq/gitlabhq/blob/master/doc/install/installation.md).
* [FramaGit](https://framagit.org) : dépôt fonctionnant sous GitLab géré par FramaSoft, une alternative aux dépôts propriétaires comme GitHub.

Haut du formulaire

Bas du formulaire

**Autres**

* **(fr)** Discussion [«Conversion d'un dépôt bzr en un dépôt git»](http://forum.ubuntu-fr.org/viewtopic.php?id=692171) sur le forum ubuntu-fr.
* [Création rapide de serveur git avec tutoriel](https://forum.ubuntu-fr.org/viewtopic.php?id=1310241)
* **(fr)** [Aide-mémoire interactif](http://blogs.media-tips.com/bernard.opic/2013/12/01/initiez-vous-a-git-avec-laide-memoire-interactif-dandrew-peterson/)
* **(fr)** [Aide mémoire GIT au format PDF](http://www.chtiland.fr/page.php?id=informatique:developpement:start#docs)