

Setup

git clone <repo>

clone the repository specified by <repo>; this is similar to "checkout" in some other version control systems such as Subversion and CVS

```
[color]
  branch = auto
  diff = auto
  status = auto
[color "branch"]
  current = yellow reverse
  local = yellow
  remote = green
[color "diff"]
  meta = yellow bold
  frag = magenta bold
  old = red bold
  new = green bold
[color "status"]
  added = yellow
  changed = green
  untracked = cyan
```

Add colors to your `~/.gitconfig` file:

Highlight whitespace in diffs:

```
[color]
  ui = true
[core]
  whitespace=fix,-indent-with-non-tab,trailing-space,cr-at-eol
```

Add aliases to your `~/.gitconfig` file:

```
[alias]
  st = status
  ci = commit
  br = branch
  co = checkout
  df = diff
  lg = log -p
```

Configuration

git config user.email johndoe@example.com

Sets your email for commit messages.

git config user.name 'John Doe'

Sets your name for commit messages.

git config branch.autosetupmerge true

Tells `git-branch` and `git-checkout` to setup new branches so that **git-pull(1)** will appropriately merge from that remote branch. Recommended. Without this, you will have to add **--track** to your branch command or manually merge remote tracking branches with "**fetch**" and then "**merge**".

You can add "**--global**" after "**git config**" to any of these commands to make it apply to all git repos (writes to `~/.gitconfig`).

Info

git diff

show a diff of the changes made since your last commit

git status

show files added to the index, files with changes, and untracked files

git log

show recent commits, most recent on top

git log --color

show recent commits, with color

git log --stat

show recent commits, with stats (files changed, insertions, and deletions)

git log --author=foo

show recent commits, only by a certain author

git log -p

show recent commits, with full diffs

git log --after="MMM DD YYYY" ex. ("Jun 20 2008")

show commits that occur after a certain date

git log --before="MMM DD YYYY" ex. ("Jun 20 2008")

show commits that occur before a certain date

git show <rev>

show the changeset (diff) of a commit specified by **<rev>**, which can be any SHA1 commit ID, branch name, or tag

git blame <file>

show who authored each line in <file>

git blame <file> <rev>

show who authored each line in <file> as of <rev> (allows blame to go back in time)

git whatchanged <file>

show only the commits which affected <file> listing the most recent first

Adding / Deleting

git add <file1> <file2> ...

add <file1>, <file2>, etc... to the project

git add <dir>

add all files under directory <dir> to the project, including subdirectories

git add .

add all files under the current directory to the project

git rm <file1> <file2> ...

remove <file1>, <file2>, etc... from the project

Committing

git commit <file1> <file2> ... [-m <msg>]

commit <file1>, <file2>, etc..., optionally using commit message <msg>, otherwise opening your editor to let you type a commit message

git commit -a [-m <msg>]

commit all files changed since your last commit, optionally using commit message <msg>

git commit -v [-m <msg>]

commit verbosely, i.e. includes the diff of the contents being committed in the commit message screen

git commit --amend <file1> <file2> ...

include changes made to <file1>, <file2>, etc..., and recommit with previous commit message

Sharing

git pull

update the current branch with changes from the server. Note: **.git/config** must have a **[branch "some_name"]** section for the current branch. Git 1.5.3 and above adds this automatically.

git push

update the server with your commits across all branches that are ***COMMON*** between your local copy and the server. Local branches that were never pushed to the server in the first place are not shared.

git push origin <branch>

update the server with your commits made to **<branch>** since your last push. This is always ***required*** for new branches that you wish to share. After the first explicit push, "git push" by itself is sufficient.

Branching

git branch

list all local branches

git branch -r

list all remote branches

git branch -a

list all local and remote branches

git branch <branch>

create a new branch named **<branch>**, referencing the same point in history as the current branch

git branch <branch> <start-point>

create a new branch named **<branch>**, referencing **<start-point>**, which may be specified any way you like, including using a branch name or a tag name

git branch --track <branch> <remote-branch>

create a tracking branch. Will push/pull changes to/from another repository. Example: **git branch --track experimental origin/experimental**

git branch -d <branch>

delete the branch **<branch>**; if the branch you are deleting points to a commit which is not reachable from the current branch, this command will fail with a warning.

git branch -r -d <remote branch>

delete a "local remote" branch, used to delete a tracking branch.

Example: **git branch -r -d wycats/master**

git branch -D <branch>

even if the branch points to a commit not reachable from the current branch, you may know that that commit is still reachable from some other branch or tag. In that case it is safe to use this command to force git to delete the branch.

git checkout <branch>

make the current branch <branch>, updating the working directory to reflect the version referenced by <branch>

git checkout -b <new> <start-point>

create a new branch <new> referencing <start-point>, and check it out.

git push <repository> :heads/<branch>

removes a branch from a remote repository.

Example: **git push origin :heads/old_branch_to_be_deleted**

Merging

git merge <branch>

merge branch <branch> into the current branch; this command is idempotent and can be run as many times as needed to keep the current branch up-to-date with changes in <branch>

git merge <branch> --no-commit

merge branch <branch> into the current branch, but do not autocommit the result; allows you to make further tweaks

git merge <branch> -s ours

merge branch <branch> into the current branch, but drops any changes in <branch>, using the current tree as the new tree

Conflicts

git mergetool

Work through conflicted files by opening them in your mergetool (opendiff, kdiff3, etc.) and choosing left/right chunks. The merged result is staged for commit.

For binary files or if mergetool won't do, resolve the conflict(s) manually and then do:

git add <file1> [<file2> ...]

Once all conflicts are resolved and staged, commit the pending merge with:

git commit

Reverting

git revert <rev>

reverse commit specified by **<rev>** and commit the result. This does **not** do the same thing as similarly named commands in other VCS's such as "svn revert" or "bzt revert", see below

git checkout <file>

re-checkout **<file>**, overwriting any local changes

git checkout .

re-checkout all files, overwriting any local changes. This is most similar to "svn revert" if you're used to Subversion commands

Fix mistakes / Undo

git reset --hard

abandon everything since your last commit; this command can be **DANGEROUS**. If merging has resulted in conflicts and you'd like to just forget about the merge, this command will do that

git reset --hard ORIG_HEAD

undo your most recent **successful** merge **and** any changes that occurred after. Useful for forgetting about the merge you just did. If there are conflicts (the merge was not successful), use "git reset --hard" (above) instead.

git reset --soft HEAD^

Forgot something in your last commit? That's easy to fix. Undo your last commit, but keep the changes in the index for editing

git commit --amend

Edit Commit message

git reset --soft HEAD^

Plumbing

test <sha1-A> = \$(git merge-base <sha1-A> <sha1-B>)

determine if merging **sha1-B** into **sha1-A** is achievable as a fast forward; non-zero exit status is false.

Stashing

git stash save <optional-name>

save your local modifications to a new stash so you can "git svn rebase" or "git pull"

git stash apply

restore the changes recorded in the stash on top of the current working tree state

git stash pop

restore the changes from the most recent stash, and remove it from the stack of stashed changes

git stash list

list all current stashes

git stash show <stash-name> -p

show the contents of a stash - accepts all diff args

Remotes

git remote add <branch> <remote branch>

adds a remote repository to your git config. Can be then fetched locally.

Example:

```
git remote add coreteam git://github.com/wycats/merb-plugins.git
git fetch coreteam
```

git push {repository} :heads/{branch}

Delete a remote repository

git remote prune {repository}

Prune deleted remote repositories from git branch listing

Environment Variables

GIT_AUTHOR_NAME, GIT_COMMITTER_NAME

Your full name to be recorded in any newly created commits. Overrides user.name in **.git/config**

GIT_AUTHOR_EMAIL, GIT_COMMITTER_EMAIL

Your email address to be recorded in any newly created commits. Overrides user.email in **.git/config**