|  |
| --- |
| **GIT FAQs**   * [1 General Questions](https://git.wiki.kernel.org/index.php/GitFaq#General_Questions)   + [1.1 What is git ?](https://git.wiki.kernel.org/index.php/GitFaq#What_is_git_.3F)   + [1.2 Why the 'git' name?](https://git.wiki.kernel.org/index.php/GitFaq#Why_the_.27git.27_name.3F)   + [1.3 How do I report a bug in git?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_report_a_bug_in_git.3F)   + [1.4 What's the difference between fetch and pull?](https://git.wiki.kernel.org/index.php/GitFaq#What.27s_the_difference_between_fetch_and_pull.3F)   + [1.5 What can I use to set up a public repository?](https://git.wiki.kernel.org/index.php/GitFaq#What_can_I_use_to_set_up_a_public_repository.3F)   + [1.6 Can I add empty directories?](https://git.wiki.kernel.org/index.php/GitFaq#Can_I_add_empty_directories.3F)   + [1.7 Why does git not "track" renames?](https://git.wiki.kernel.org/index.php/GitFaq#Why_does_git_not_.22track.22_renames.3F)   + [1.8 Why is "git log <filename>" slow?](https://git.wiki.kernel.org/index.php/GitFaq#Why_is_.22git_log_.3Cfilename.3E.22_slow.3F)   + [1.9 Why is it wrong to export the environment variable CDPATH?](https://git.wiki.kernel.org/index.php/GitFaq#Why_is_it_wrong_to_export_the_environment_variable_CDPATH.3F)   + [1.10 How come gitweb suddenly disappeared from the project list at kernel.org/git?](https://git.wiki.kernel.org/index.php/GitFaq#How_come_gitweb_suddenly_disappeared_from_the_project_list_at_kernel.org.2Fgit.3F)   + [1.11 What is the difference between a merge and a rebase?](https://git.wiki.kernel.org/index.php/GitFaq#What_is_the_difference_between_a_merge_and_a_rebase.3F)   + [1.12 Why is "git rm" not the inverse of "git add"?](https://git.wiki.kernel.org/index.php/GitFaq#Why_is_.22git_rm.22_not_the_inverse_of_.22git_add.22.3F) * [2 Features](https://git.wiki.kernel.org/index.php/GitFaq#Features)   + [2.1 Does git convert between CRLF and LF for different platforms?](https://git.wiki.kernel.org/index.php/GitFaq#Does_git_convert_between_CRLF_and_LF_for_different_platforms.3F)   + [2.2 Does git have keyword expansion?](https://git.wiki.kernel.org/index.php/GitFaq#Does_git_have_keyword_expansion.3F)   + [2.3 Does git allow arbitrary conversion of contents?](https://git.wiki.kernel.org/index.php/GitFaq#Does_git_allow_arbitrary_conversion_of_contents.3F)   + [2.4 Does git convert encodings of file names?](https://git.wiki.kernel.org/index.php/GitFaq#Does_git_convert_encodings_of_file_names.3F)   + [2.5 Does git convert encodings of comments and committer names or file content?](https://git.wiki.kernel.org/index.php/GitFaq#Does_git_convert_encodings_of_comments_and_committer_names_or_file_content.3F)   + [2.6 Does git track all file data and metadata?](https://git.wiki.kernel.org/index.php/GitFaq#Does_git_track_all_file_data_and_metadata.3F) * [3 Unexpected behavior](https://git.wiki.kernel.org/index.php/GitFaq#Unexpected_behavior)   + [3.1 Why won't I see changes in the remote repo after "git push"?](https://git.wiki.kernel.org/index.php/GitFaq#Why_won.27t_I_see_changes_in_the_remote_repo_after_.22git_push.22.3F)   + [3.2 Why is git --version not reporting the "full" version number?](https://git.wiki.kernel.org/index.php/GitFaq#Why_is_git_--version_not_reporting_the_.22full.22_version_number.3F)   + [3.3 Why is "git reset --hard" not removing some files?](https://git.wiki.kernel.org/index.php/GitFaq#Why_is_.22git_reset_--hard.22_not_removing_some_files.3F)   + [3.4 Why is my push rejected with a non-fast forward error?](https://git.wiki.kernel.org/index.php/GitFaq#Why_is_my_push_rejected_with_a_non-fast_forward_error.3F)   + [3.5 Why won't "git push" work after I rebase a branch?](https://git.wiki.kernel.org/index.php/GitFaq#Why_won.27t_.22git_push.22_work_after_I_rebase_a_branch.3F)   + [3.6 Why is "git commit -a" not the default?](https://git.wiki.kernel.org/index.php/GitFaq#Why_is_.22git_commit_-a.22_not_the_default.3F)   + [3.7 My HTTP repository has updates, which git clone misses. What happened?](https://git.wiki.kernel.org/index.php/GitFaq#My_HTTP_repository_has_updates.2C_which_git_clone_misses._What_happened.3F)   + [3.8 Why isn't Git preserving modification time on files?](https://git.wiki.kernel.org/index.php/GitFaq#Why_isn.27t_Git_preserving_modification_time_on_files.3F)   + [3.9 Why does git use a pager for commands like diff/log and --help?](https://git.wiki.kernel.org/index.php/GitFaq#Why_does_git_use_a_pager_for_commands_like_diff.2Flog_and_--help.3F)   + [3.10 Why does diff/log not show color, even though I enabled it?](https://git.wiki.kernel.org/index.php/GitFaq#Why_does_diff.2Flog_not_show_color.2C_even_though_I_enabled_it.3F)   + [3.11 Why does git diff sometimes list a file that has no changes?](https://git.wiki.kernel.org/index.php/GitFaq#Why_does_git_diff_sometimes_list_a_file_that_has_no_changes.3F)   + [3.12 What does the gitk error message "Can't parse git log output:" mean?](https://git.wiki.kernel.org/index.php/GitFaq#What_does_the_gitk_error_message_.22Can.27t_parse_git_log_output:.22_mean.3F)   + [3.13 Why does gitk on Cygwin display "git 1316 tty\_list::allocate: No tty allocated"?](https://git.wiki.kernel.org/index.php/GitFaq#Why_does_gitk_on_Cygwin_display_.22git_1316_tty_list::allocate:_No_tty_allocated.22.3F)   + [3.14 Why does git clone, git pull, etc. fail when run on a partition mounted with sshfs (FUSE)?](https://git.wiki.kernel.org/index.php/GitFaq#Why_does_git_clone.2C_git_pull.2C_etc._fail_when_run_on_a_partition_mounted_with_sshfs_.28FUSE.29.3F)   + [3.15 Why does "git bisect" make me test versions outside the "good-bad" range?](https://git.wiki.kernel.org/index.php/GitFaq#Why_does_.22git_bisect.22_make_me_test_versions_outside_the_.22good-bad.22_range.3F)   + [3.16 Why am I "not on any branch"?](https://git.wiki.kernel.org/index.php/GitFaq#Why_am_I_.22not_on_any_branch.22.3F)   + [3.17 "git log -S" does not show all commits](https://git.wiki.kernel.org/index.php/GitFaq#.22git_log_-S.22_does_not_show_all_commits) * [4 How do I ...](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_...)   + [4.1 How do I specify what ssh key git should use?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_specify_what_ssh_key_git_should_use.3F)   + [4.2 How do I untrack a file?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_untrack_a_file.3F)   + [4.3 How do I access other branches in a repository?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_access_other_branches_in_a_repository.3F)   + [4.4 How do I share a git public repository and use it in a CVS way?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_share_a_git_public_repository_and_use_it_in_a_CVS_way.3F)   + [4.5 How do I share a git repository using POSIX ACLs?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_share_a_git_repository_using_POSIX_ACLs.3F)   + [4.6 How can I add a diff of the commit into the commit message window?](https://git.wiki.kernel.org/index.php/GitFaq#How_can_I_add_a_diff_of_the_commit_into_the_commit_message_window.3F)   + [4.7 How would I use "git push" to sync out of a host that I cannot pull from?](https://git.wiki.kernel.org/index.php/GitFaq#How_would_I_use_.22git_push.22_to_sync_out_of_a_host_that_I_cannot_pull_from.3F)   + [4.8 How do I check out the tree at a particular tag?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_check_out_the_tree_at_a_particular_tag.3F)   + [4.9 How to share objects between existing repositories?](https://git.wiki.kernel.org/index.php/GitFaq#How_to_share_objects_between_existing_repositories.3F)   + [4.10 How to stop sharing objects between repositories?](https://git.wiki.kernel.org/index.php/GitFaq#How_to_stop_sharing_objects_between_repositories.3F)   + [4.11 How do I tell git to ignore files?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_tell_git_to_ignore_files.3F)   + [4.12 How do I tell git to ignore tracked files?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_tell_git_to_ignore_tracked_files.3F)   + [4.13 How do I save some local modifications?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_save_some_local_modifications.3F)   + [4.14 How to manually resolve conflicts when Git failed to detect rename?](https://git.wiki.kernel.org/index.php/GitFaq#How_to_manually_resolve_conflicts_when_Git_failed_to_detect_rename.3F)   + [4.15 How to revert file to version from current commit?](https://git.wiki.kernel.org/index.php/GitFaq#How_to_revert_file_to_version_from_current_commit.3F)   + [4.16 How to view an old revision of a file or directory?](https://git.wiki.kernel.org/index.php/GitFaq#How_to_view_an_old_revision_of_a_file_or_directory.3F)   + [4.17 How can I tweak the date of a commit in the repo?](https://git.wiki.kernel.org/index.php/GitFaq#How_can_I_tweak_the_date_of_a_commit_in_the_repo.3F)   + [4.18 How do I make a diff between two arbitrary files in different revisions?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_make_a_diff_between_two_arbitrary_files_in_different_revisions.3F)   + [4.19 How to fix a broken repository?](https://git.wiki.kernel.org/index.php/GitFaq#How_to_fix_a_broken_repository.3F)   + [4.20 How to remove all broken refs from a repository?](https://git.wiki.kernel.org/index.php/GitFaq#How_to_remove_all_broken_refs_from_a_repository.3F)   + [4.21 How to set up a git server?](https://git.wiki.kernel.org/index.php/GitFaq#How_to_set_up_a_git_server.3F)   + [4.22 How to create the first project?](https://git.wiki.kernel.org/index.php/GitFaq#How_to_create_the_first_project.3F)   + [4.23 How do I publish my repo via SFTP?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_publish_my_repo_via_SFTP.3F)   + [4.24 How do I do a quick clone without history revisions?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_do_a_quick_clone_without_history_revisions.3F)   + [4.25 How do I use git for large projects, where the repository is large, say approaching 1 TB, but a checkout is only a few hundred MB? Will every developer need 1 TB of local disk space?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_use_git_for_large_projects.2C_where_the_repository_is_large.2C_say_approaching_1_TB.2C_but_a_checkout_is_only_a_few_hundred_MB.3F_Will_every_developer_need_1_TB_of_local_disk_space.3F)   + [4.26 How do I obtain a list of files which have changed in a given commit?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_obtain_a_list_of_files_which_have_changed_in_a_given_commit.3F)   + [4.27 How do I remove all the old objects after using filter-branch?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_remove_all_the_old_objects_after_using_filter-branch.3F)   + [4.28 How do I mirror a SVN repository to git?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_mirror_a_SVN_repository_to_git.3F)     - [4.28.1 Troubleshooting](https://git.wiki.kernel.org/index.php/GitFaq#Troubleshooting)   + [4.29 How do I edit the root commit?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_edit_the_root_commit.3F)   + [4.30 How do I clone a subdirectory?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_clone_a_subdirectory.3F)   + [4.31 How do I make existing non-bare repository bare?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_make_existing_non-bare_repository_bare.3F)   + [4.32 How do I find large files?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_find_large_files.3F)   + [4.33 How do I recover uncommitted changes?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_recover_uncommitted_changes.3F)   + [4.34 How do I merge a patch from a gmane.org URL?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_merge_a_patch_from_a_gmane.org_URL.3F)   + [4.35 How do I clone a repository with all remotely tracked branches?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_clone_a_repository_with_all_remotely_tracked_branches.3F)   + [4.36 How do I remove my uncommitted changes from branch A and add them to a new branch B?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_remove_my_uncommitted_changes_from_branch_A_and_add_them_to_a_new_branch_B.3F)   + [4.37 How do I use external diff tools?](https://git.wiki.kernel.org/index.php/GitFaq#How_do_I_use_external_diff_tools.3F) * [5 Error diagnostic](https://git.wiki.kernel.org/index.php/GitFaq#Error_diagnostic)   + [5.1 Git push fails with "fatal: The remote end hung up unexpectedly"?](https://git.wiki.kernel.org/index.php/GitFaq#Git_push_fails_with_.22fatal:_The_remote_end_hung_up_unexpectedly.22.3F)   + [5.2 Git commit is dying telling me "fatal: empty ident <user@myhost> not allowed", what's wrong?](https://git.wiki.kernel.org/index.php/GitFaq#Git_commit_is_dying_telling_me_.22fatal:_empty_ident_.3Cuser.40myhost.3E_not_allowed.22.2C_what.27s_wrong.3F)   + [5.3 Why won't git let me change to a different branch?](https://git.wiki.kernel.org/index.php/GitFaq#Why_won.27t_git_let_me_change_to_a_different_branch.3F)   + [5.4 refs/heads/pu: does not fast forward to branch 'pu'](https://git.wiki.kernel.org/index.php/GitFaq#refs.2Fheads.2Fpu:_does_not_fast_forward_to_branch_.27pu.27)   + [5.5 protocol error: bad line length character](https://git.wiki.kernel.org/index.php/GitFaq#protocol_error:_bad_line_length_character)   + [5.6 "unable to chdir or not a git archive" while pushing](https://git.wiki.kernel.org/index.php/GitFaq#.22unable_to_chdir_or_not_a_git_archive.22_while_pushing)   + [5.7 "needs update" and "not uptodate"](https://git.wiki.kernel.org/index.php/GitFaq#.22needs_update.22_and_.22not_uptodate.22)   + [5.8 "git-receive-pack: command not found" on push/fetch/pull](https://git.wiki.kernel.org/index.php/GitFaq#.22git-receive-pack:_command_not_found.22_on_push.2Ffetch.2Fpull)   + [5.9 My username contains a '@', I can't clone through HTTP/HTTPS](https://git.wiki.kernel.org/index.php/GitFaq#My_username_contains_a_.27.40.27.2C_I_can.27t_clone_through_HTTP.2FHTTPS) * [6 Importing from other revision control systems](https://git.wiki.kernel.org/index.php/GitFaq#Importing_from_other_revision_control_systems)   + [6.1 Can I import from tar files (archives)?](https://git.wiki.kernel.org/index.php/GitFaq#Can_I_import_from_tar_files_.28archives.29.3F)   + [6.2 Can I import from CVS?](https://git.wiki.kernel.org/index.php/GitFaq#Can_I_import_from_CVS.3F)   + [6.3 Can I import from svn?](https://git.wiki.kernel.org/index.php/GitFaq#Can_I_import_from_svn.3F)   + [6.4 Can I import from arch/baz/tla?](https://git.wiki.kernel.org/index.php/GitFaq#Can_I_import_from_arch.2Fbaz.2Ftla.3F)   + [6.5 Can I import from Perforce?](https://git.wiki.kernel.org/index.php/GitFaq#Can_I_import_from_Perforce.3F)   + [6.6 Can I import from Mercurial?](https://git.wiki.kernel.org/index.php/GitFaq#Can_I_import_from_Mercurial.3F)   + [6.7 Can I import from other SCMs?](https://git.wiki.kernel.org/index.php/GitFaq#Can_I_import_from_other_SCMs.3F) * [7 See also](https://git.wiki.kernel.org/index.php/GitFaq#See_also) |

General Questions

What is git ?

Git is a distributed version control system developed by Junio Hamano and Linus Torvalds.

Git does not use a centralized server.

Git runs on Linux, BSD, Solaris, Darwin, Windows, Android and other operating systems.

Why the 'git' name?

Quoting Linus: "I'm an egotistical \*\*\*, and I name all my projects after myself. First 'Linux', now 'git'".

('git' is British slang for "pig headed, think they are always correct, argumentative").

Alternatively, in Linus' own words as the inventor of git: "git" can mean anything, depending on your mood:

* Random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
* Stupid. Contemptible and despicable. Simple. Take your pick from the dictionary of slang.
* "Global information tracker": you're in a good mood, and it actually works for you. Angels sing and light suddenly fills the room.
* "Goddamn idiotic truckload of sh\*t": when it breaks

How do I report a bug in git?

Bug reports can be sent to the git mailing list: git@vger.kernel.org

Please but "[BUG]" in the subject line. You don't need to be subscribed to post; see [GitCommunity](https://git.wiki.kernel.org/index.php/GitCommunity" \o "GitCommunity) for more information on the mailing list and other ways of interacting with git developers.

What's the difference between fetch and pull?

The short definition is:

*Fetch*: Download (new) objects and a head from another repository.

*Pull*: Fetch (as defined above), and then merge what was downloaded with the current development.

* See the [git-fetch(1)](http://www.kernel.org/pub/software/scm/git/docs/git-fetch.html) and [git-pull(1)](http://www.kernel.org/pub/software/scm/git/docs/git-pull.html) man pages or the tutorials for more details.

What can I use to set up a public repository?

An SSH server, an HTTP server, or the git-daemon. Local networks can also use network filesystems, like NFS or SMBFS/CIFS (the Windows networks).

* While there was some controversy over whether SMBFS was a safe solution the [latest info](http://permalink.gmane.org/gmane.comp.version-control.git/122670) indicates that it should be fine.
* See the [Git tutorial](http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html) for more details.

Can I add empty directories?

Currently the design of the git index (staging area) only permits *files* to be listed, and nobody competent enough to make the change to allow empty directories has cared enough about this situation to remedy it.

Directories are added automatically when adding files inside them. That is, directories never have to be added to the repository, and are not tracked on their own.

You can say "git add <dir>" and it will add the files in there.

If you really need a directory to exist in checkouts you should create a file in it. .gitignore works well for this purpose (there is also a tool [MarkEmptyDirs](https://git.wiki.kernel.org/index.php/MarkEmptyDirs" \o "MarkEmptyDirs) using the .NET framework which allows you to automate this task); you can leave it empty or fill in the names of files you expect to show up in the directory.

Why does git not "track" renames?

[Git](https://git.wiki.kernel.org/index.php/Git) has to interoperate with a lot of different workflows, for example some changes can come from patches, where rename information may not be available. Relying on explicit rename tracking makes it impossible to merge two trees that have done exactly the same thing, except one did it as a patch (create/delete) and one did it using some other heuristic.

On a second note, tracking renames is really just a special case of tracking how content moves in the tree. In some cases, you may instead be interested in querying when a function was added or moved to a different file. By only relying on the ability to recreate this information when needed, Git aims to provide a more flexible way to track how your tree is changing.

However, this does not mean that Git has no support for renames. The diff machinery in Git has support for automatically detecting renames, this is turned on by the '-M' switch to the git-diff-\* family of commands. The rename detection machinery is used by [git-log(1)](http://www.kernel.org/pub/software/scm/git/docs/git-log.html) and [git-whatchanged(1)](http://www.kernel.org/pub/software/scm/git/docs/git-whatchanged.html), so for example, 'git log -M' will give the commit history with rename information. Git also supports a limited form of merging across renames. The two tools for assigning blame, [git-blame(1)](http://www.kernel.org/pub/software/scm/git/docs/git-blame.html) and [git-annotate(1)](http://www.kernel.org/pub/software/scm/git/docs/git-annotate.html) both use the automatic rename detection code to track renames.

As a very special case, 'git log' version 1.5.3 and later has '--follow' option that allows you to follow renames when given a single path.

* [Mail](http://permalink.gmane.org/gmane.comp.version-control.git/217) by Linus on this topic.

Git has a rename command *git mv*, but that is just for convenience. The effect is indistinguishable from removing the file and adding another with different name and the same content.

Why is "git log <filename>" slow?

The answer to the question of why git log <filename> takes so long to find a small number of commits which changed a given file lies in the fact that Git looks at *all* the commits to find that.

Git simply *does not have* per-file history. Not having a per-file history is what allows Git to do git log <directory-or-file> rather than being able to track just one file. You can't do it sanely with per-file history (because to tie the per-file histories back together in a logical sequence, you need the global history to sort it again!).

That said, you might well need to just run git gc to make it hugely faster.

Note that git log <file1> <file2> (or gitk <file1> <file2>) is not simply the union of git log <file1> and git log <file2>; it can contain merges which are in neither of the separate histories. Doing the history for two files together is *not at all equivalent* to doing the history for those files individually and stitching it together.

To speed up git log, give it a range of interesting revisions; you can also try --remove-empty option (with some caveats).

* [Re: how to speed up "git log"?](http://permalink.gmane.org/gmane.comp.version-control.git/39358) by Linus Torvalds on the Git mailing list.

Why is it wrong to export the environment variable CDPATH?

The CDPATH variable is purely for interactive use. A lot of scripts break if a simple "cd" suddenly outputs a text that was directed at *you*, the user, and not meant for processing by the script. Having said that, we tried hard to avoid problems by unsetting CDPATH everywhere in the Git scripts and in the Makefiles, but it is quite possible that we missed some places.

By contrast, if you just remove the "export" from your .bashrc, you are guaranteed to never get bitten by that breakage again!

* [Re: make install bug?](http://article.gmane.org/gmane.comp.version-control.git/13736/match=cdpath)

How come gitweb suddenly disappeared from the project list at kernel.org/git?

Because it got merged into git. See [the relevant commit](http://kernel.org/git/?p=git/git.git;a=commit;h=0a8f4f0020cb35095005852c0797f0b90e9ebb74).

What is the difference between a merge and a rebase?

Imagine this history:

- A - B - C - D - remote HEAD

\

E - F - G - local HEAD

after merge:

- A - B - C - D - remote HEAD

\ \

E - F - G - local HEAD - new local HEAD

after rebase:

- A - B - C - D - remote HEAD - E' - F' - G' - local HEAD'

So, the merge does not rewrite your commits. They will stay the same. A rebase \_will\_ rewrite them. The advantage is that the history is easier to follow. The disadvantage is that you usually tested more while developing, so that some obscure feature in, say, "G", could work less nicely together with the changes of, say, "B", than you hope for. In other words, G was tested thoroughly, G' was not. As always, it's a trade-off.

Why is "git rm" not the inverse of "git add"?

Don't think of 'rm' as the inverse of 'add'. That will only confuse you.

When git add is used to add changes made to a file already tracked by git, the inverse of 'git add <file>' is 'git reset HEAD -- <file>'.

In the case of adding a new file, a natural inverse of 'add' is 'un-add', and that operation is called 'rm --cached', because we use that to name the option to invoke an "index-only" variant of a command when the command can operate on index and working tree file (e.g. "diff --cached", "apply --cached").

The life of a file that does \_not\_ make it into a commit goes like this:

[1]$ edit a-new-file

The above is a 'create', not an 'add'. git is not involved in this step.

[2]$ git add a-new-file

This is an 'add' — the placement of an existing file in the index. When you do not want it in the index, you 'un-add' it.

[3]$ git rm --cached a-new-file

This removes the entry from the index, without touching the working tree file. If you do not want that file at all (as opposed to "I am making a series of partial commits and the addition of this path does not belong to the first commit of the series, so I am unstaging"), this is followed by:

[4]$ rm -f a-new-file

Again, git is not involved in this step.

Users sometimes want steps 3 and 4 combined, and this meshes well with user expectations when they see the word "rm". Think of "git rm" without "--cached" as a shorthand to do steps 3 and 4 in one go to meet that expectation.

Obviously, we cannot usefully combine steps 1 and 2. We could have "git add --create a-new-file" launch an editor to create a new file, but that would not be very useful in practice.

The fact that steps 3 and 4 can be naturally combined, whereas steps 1 and 2 cannot, means that "add" and "rm" are not the inverse of one other.

* [Re: git-rm isn't the inverse action of git-add](http://mid.gmane.org/7vps2y3a4n.fsf@assigned-by-dhcp.cox.net) message by Junio C Hamano.
* [git unadd](http://pivotallabs.com/users/alex/blog/articles/1001-git-unadd) by Alex Chaffee

Features

Does git convert between CRLF and LF for different platforms?

Newline conversion is supported from Git version 1.5 onwards. See [gitattributes(5)](http://www.kernel.org/pub/software/scm/git/docs/gitattributes.html" \l "_checking_out_and_checking_in).

Does git have keyword expansion?

Keyword expansion is not recommended. Keyword expansion causes all sorts of strange problems and isn't really useful anyway, especially within the context of an SCM. You can perform keyword expansion outside of git using a custom script. The Linux kernel export script does this to set the EXTRA\_VERSION variable in the Makefile.

See [gitattributes(5)](http://www.kernel.org/pub/software/scm/git/docs/gitattributes.html) if you really want to do this. If your translation is not reversible (eg SCCS keyword expansion) this may be problematic. (Hint: the supplied [$Id$-expansion](http://www.kernel.org/pub/software/scm/git/docs/gitattributes.html#_tt_ident_tt) puts the 40-character hexadecimal blob object name into the id; you can figure out which commits include this blob by using a script like [this](http://stackoverflow.com/questions/223678/git-which-commit-has-this-blob).)

See [here for a discussion](http://thread.gmane.org/gmane.comp.version-control.git/44750), and [here on how GIT may help anyway](http://article.gmane.org/gmane.comp.version-control.git/44849).

Does git allow arbitrary conversion of contents?

Yes. Not just "keyword expansion" and/or "CRLF conversion", the current version of git allows you to specify filters to munge contents immediately before checking things in. See [gitattributes(5)](http://www.kernel.org/pub/software/scm/git/docs/gitattributes.html) man pages for details.

Does git convert encodings of file names?

No. Filenames are treated as byte sequences.

Does git convert encodings of comments and committer names or file content?

A git repository can store a flag to register the encoding supposedly used for comments (including author names). File content is not converted unless you are inclined to want to shoot yourself in the foot, in which case use the filtering mechanism described above.

Does git track all file data and metadata?

No. Git has a specific notion of tracked "content", which is basically just the file data. It is thus not directly suitable for tracking directories where additional filesystem information is significant, such as "/etc" or home directories. See [ContentLimitations](https://git.wiki.kernel.org/index.php/ContentLimitations" \o "ContentLimitations) for more details.

Unexpected behavior

Why won't I see changes in the remote repo after "git push"?

The push operation is always about propagating the repository history and updating the refs, and never touches the working tree files. In particular, if you push to update the branch that is checked out in a remote repository the files in the work tree will not be updated.

This is a precautionary design decision. The remote repository's work tree may have local changes, and there is no way for you, who are pushing into the remote repository, to resolve conflicts between the changes you are pushing and the ones in the work tree. However, you can easily make a post-update hook to update the working copy of the checked out branch. The reason for not making this a default example hook is that they only notify the person doing the pushing if there was a problem. The latest draft post-update hook for this is at <http://utsl.gen.nz/git/post-update>, which deals with almost all cases, apart from where there is already a conflicted merge on the remote side (as git-stash cannot currently stash this). It also fails to work in instances where it could, such as none of the files are actually conflicting.

A quick rule of thumb is to *never* push into a repository that has a work tree attached to it, until you know what you are doing.

If you are sure what you are doing, you can do a "git reset --hard" on the side you pushed to. Note that this WILL lose ALL changes you made on that side, resetting the working tree to the newest revision you pushed. See [this article about bare repositories](http://sitaramc.github.com/concepts/bare.html#how_do_I_fix_such_a_non_bare_push_) for details.

See also the entry ([How would I use "git push" to sync out of a firewalled host?](https://git.wiki.kernel.org/index.php/GitFaq#push-is-reverse-of-fetch)) in this FAQ for the proper way to work with push with a repository with a work tree.

Why is git --version not reporting the "full" version number?

There is a bit of a chicken and egg problem involved. The build procedure wants to have an already installed git to figure out the "full" version number. If you are bootstrapping, make clean and rebuild after you install git once would give you a git binary that knows what version it is.

GIT-VERSION-GEN script show current (used) git version, git --version shows git version used at the time git was build.

Why is "git reset --hard" not removing some files?

"git reset" will not delete files it does not track, including files it is told to ignore. If reset switches to a version with a different .gitignore file, then some previously ignored files might show up as untracked files in git status output.

* See [Linus Torvalds](http://marc.theaimsgroup.com/?l=git&m=114917892328066) answer or [Sean Estabrooks](http://marc.theaimsgroup.com/?l=git&m=114917892613019) answer on the [git mailing list](https://git.wiki.kernel.org/index.php/GitCommunity" \o "GitCommunity).
* If you are using cygwin or OSX, you might be having issues with case sensitivity — see [some discussions](http://marc.info/?l=git&w=2&r=1&s=xt_CONNMARK.h&q=b).

Why is my push rejected with a non-fast forward error?

If you try to push a branch, you might get this error message:

! [rejected] master -> master (non-fast forward)

error: failed to push some refs to 'git@github.com:pieter/gitx.git'

This means that your branch is not a strict superset of the remote side. That is, the remote side has commits that your side does not have. If you would push, the other side would lose changes. The most likely reason for this is that you need to

git pull

from the remote first. You can see what changes the remote side has by fetching first and then checking the log. For example,

git fetch origin

git log master..origin/master

will list all the changes the remote side has that your side doesn't. If you want a graphical representation, use gitk --left-right master...origin/master. The arrows to the left are changes you want to push, the arrows to the right are changes on the remote side.

If you have rebased your branch and try to push that, see the next question.

If you think you know what you are doing, you can also try:

git push origin +branchname

This will force the update. If you don't have permission, then sometimes this will work:

git push origin :branchname

git push origin +branchname

ie, delete the branch remotely first (this is often permitted), then re-push the "new" (or perhaps rewound) branch.

Be warned that if you rewind branches, others might get into problem when pulling. There is the chance that they will merge in the branch that they fetched with the new one that you've published, effectively keeping the changes that you are trying to get rid of. However, it will only be their copies that have the bad revisions. For this reason, rewinding branches is considered mildly antisocial. Nonetheless, it is often appropriate.

Why won't "git push" work after I rebase a branch?

After you have rebased one of your local branches, you try to push your changes to a remote repository — but git push fails with this error message:

error: remote 'refs/heads/master' is not a strict subset of local ref 'refs/heads/master'. maybe you are not up-to-date and need to pull first?

This is not a bug, but a safety check: "git push" will not update a remote branch if the remote branch is not a parent of the commit you're trying to push. This check prevents you from overwriting a remote branch to which other people have already committed new changes after you fetched it the last time. Their changes would be lost without the check. And it prevents you from overwriting a remote branch with an unrelated local branch.

When you rebase, you are not continuing the history of the branch from where you currently are. Instead, you are rewriting the history starting from the base you chose for rebasing. So, after rebasing, the remote branch and your new local HEAD are both child commits of that base, but the remote branch is no longer a parent of your new local HEAD. And pushing this new history to the remote branch means replacing a history that other people might already have downloaded.

If you are really sure that you want to push the new reference to the remote repository you can say git push -f. But use this with care and only if you know what you are doing. However, recent versions of git disable the ability to push -f by default because it is usually an error. If you have taken care and really know what you are doing (specifically consider any tags that might have been set on commits that have been deleted or have different SHA, and notify everyone else who might have pulled the branch that they need to `git pull --rebase` and/or fix any private tags they might have made), you can disable the server side configuration which prevents a force push from being accepted:

git config receive.denyNonFastForwards false

After doing a `git push -f` you should reset the configuration value back to true (or delete it) to put the seatbelt back on.

Why is "git commit -a" not the default?

Most other version control systems will do a full-tree commit, using the content of files at commit time, by default.

Git does it differently. By default, git commits the content of the index, and only this. git commit -a gives roughly the equivalent of what other systems do. Indeed, there are many concrete reasons why git's way to manage the index is good (and leads to unique features of git):

* You can select files to commit with a fine granularity, telling git what you want to do little by little (git add *file* to add the full content of the file to the index, git add -i or git gui to add the content hunk-by-hunk, or even use the hunk splitting feature of git add -i).
* This fine-grained file selection can help you to keep an uncommitted modification in your tree for a reasonably long time. For example, you can increment the version number in the Makefile some time before a release, and use this as a reminder.
* You can perform several small steps for one commit, checking what you did with git diff, and validating each small step with git add or git add -u. Typically, you can apply a broken patch, updating the index, withgit apply --index, and then fix the patch. git diff --cached will show you your fixes, while git diff HEAD will show you the combined diff.
* This allows excellent management of merge conflicts: git diff --base, git diff --ours, git diff --theirs.
* This allows git commit --amend to amend only the log message if the index hasn't been modified in the meantime.

So, while using git commit -a is perfectly fine with the simple cycle "edit/review/commit", making it the default would make other workflows less natural.

Indeed, according to Linus, the real reason is more philosophical: git is a content tracker, and a file name has no meaning unless associated to its content. Therefore, the only sane behavior for git add *filename* is to add the content of the file as well as its name to the index.

See also (mailing list posts):

* [mailing list post](http://thread.gmane.org/gmane.comp.version-control.git/46341): [FAQ?] Rationale for git's way to manage the index
* [mailing list post](http://thread.gmane.org/gmane.comp.version-control.git/32452/focus=32610): [PATCH 0/2] Making "git commit" to mean "git commit -a"

My HTTP repository has updates, which git clone misses. What happened?

If you push via SSH to the repository, you have to enable the post-update hook (chmod a+x hooks/post-update). If you "push" with rsync, you have to make sure to execute "git update-server-info" \_before\_ pushing. HTTP is a "dumb" transport, which needs some help. This help is provided in the form of the file info/refs, which contains the current refs (names + commit names of the tips).

This is not an issue with servers using the smart HTTP transport added in git version 1.6.6. Smart HTTP will also provide a dynamically created info/refs file for older clients. See also [git-http-backend(1)](http://www.kernel.org/pub/software/scm/git/docs/git-http-backend.html).

Why isn't Git preserving modification time on files?

Modification time on files is a feature that affects build tools. Most build tools compare the timestamp of the source(s) with the timestamp of the derived file(s). If the source is newer, then a rebuild takes place, otherwise nothing happens. This speeds up the build process a lot.

Now consider what would happen if you check out another branch, and modification times were preserved. We assume you already have a fully-built project. If a source file on that other branch has a timestamp that is older than that of the corresponding derived file, the derived file will not be built even if it is different, because the build system only compares modification times. At best, you'll get some kind of weird secondary error; but most likely everything will look fine at first, but you will not get the same result as you would have with a clean build. That situation is unhealthy since you really do not know what code you are executing and the source of the problem is hard to find. You will end up always having to make a clean build when switching branches to make sure you are using the correct source. (Git [bisect](https://git.wiki.kernel.org/index.php?title=Bisect&action=edit&redlink=1) is another git procedure that checks out old and new revisions where you need a reliable rebuild.)

Git sets the current time as the timestamp on every file it modifies, but only those. The other files are left untouched, which means build tools will be able to depend on modification time and rebuild properly. If build rules change, that can cause a failure anyway, but that is a far less common problem than accidentally not rebuilding.

Why does git use a pager for commands like diff/log and --help?

Usually, you are not interested in the whole log, but only some bits at the beginning. It would not be useful for "git log" to simply let the output whiz by, leaving you looking at the uninteresting parts at the end. And if it did it the other way round, showing you the interesting bits last, it would waste a lot of time showing information that you are not interested in at all. So the only thing that makes sense is to look at the log in a pager. It also helps searching for keywords.

Note that "--help" just spawns "man", so it is not Git's fault there. But you can use git help -w xxx to use a browser instead of "man" if the HTML documentation is installed. See the git help documentation for more information about this.

If you do not like the pager default, you can set core.pager = cat with git config or tell your shell about GIT\_PAGER=cat.

Why does diff/log not show color, even though I enabled it?

Set core.pager = less -FXRS with git config to fix this. The most likely culprit is the LESS environment variable. By default, git passes the options -FXRS to less. The -R option tells less to interpret color escape sequences. If LESS is set, however, only those options are used by less.

Why does git diff sometimes list a file that has no changes?

git diff and other git operations is optimized so it does not even look at files whose status (size, modification time etc) on disk and in git's [index](https://git.wiki.kernel.org/index.php?title=Index&action=edit&redlink=1) are different. This makes git diff extremely fast for small changes. If the file has been touched somehow, git diff has to look at the content of and compare it which is a much slower operation even when there is in fact no change. git diff lists the files as a reminder that it is not used optimally. Running git status will not only show status, but will also update the index with status for unchanged files disk making subsequent operations, not only diff, much faster. A typical case that causes many files to be listed by diff is running mass editing commands like perl -pi -e '...'.

What does the gitk error message "Can't parse git log output:" mean?

This is usually caused by color.diff being set to true in your config. git log outputs log entries in colors when color.diff = true. And gitk can only parse plain output.

It is recommended (as of 1.5.3) that color.diff be kept off. Use git log --color if you need colored output.

This problem should be fixed since version 1.5.4.

Why does gitk on Cygwin display "git 1316 tty\_list::allocate: No tty allocated"?

This appears to be an issue with your Cygwin configuration. Make sure your CYGWIN environment variable doesn't contain 'tty'.

Why does git clone, git pull, etc. fail when run on a partition mounted with sshfs (FUSE)?

When running git clone and friends on a partition mounted with sshfs the following error can be triggered:

$ git clone foo

Cannot commit config file!

Cannot commit config file!

Cannot commit config file!

Initialized empty Git repository in foo/.git/

294698 blocks

Cannot commit config file!

fatal: Not a valid object name HEAD

To solve this you need 1) remount your sshfs mounted partition with the -o workaround=rename option, e.g.:

sshfs -o workaround=rename login@machine:foo bar

and 2) use at least the 1.6.0.2 version of git.

Why does "git bisect" make me test versions outside the "good-bad" range?

The reason may be that some commits have been developed starting from a version before the "good" commit, and then merged after the "good" commit.

See <http://thread.gmane.org/gmane.comp.version-control.git/99967/focus=99977> for more explanations from Linus.

Why am I "not on any branch"?

You are on a detached HEAD and might lose commits. See <http://sitaramc.github.com/concepts/detached-head.html> for an excellent discussion of the topic.

"git log -S" does not show all commits

The behavior of git log -Ssearchstring is not to compute the diff for each commit and to search for searchstring in it, but to show the commits where the number of occurrences of searchstring have changed (which is much faster than grepping the diff).

To see all commits for which searchstring appears in a changed line, use git log -Gsearchstring.

If you want to see all the commits for which searchstring appear in the diff, you can get close to the behavior you expect with a bit of scripting like:

git log -p -z | perl -ln0e 'print if /[+-].\*searchedstring/'

How do I ...

How do I specify what ssh key git should use?

This is not really a git question. However, you can edit your ~/.ssh/config file in order to tell ssh what key to use. More information about this is in the [ssh config manpage](http://www.openbsd.org/cgi-bin/man.cgi?query=ssh_config). The short version is that you can specify a custom Host with its own IdentityFile, like this:

Host GitServer

Hostname=git.example.org

IdentityFile=~/.ssh/my\_cool\_key\_rsa

Then, you can set up git to use "GitServer" as the hostname. It will look up the entry and use the specified key and host.

How do I untrack a file?

If you want to keep a file, but not have it in the next revision, do this:

git rm --cached <filename>

How do I access other branches in a repository?

When a repository is cloned, the clone gets a remote tracking branch for each of the original repository's branch heads, but only a single local branch head is created, usually this will be "master". You can get a list of all remote tracking branches with git branch -r.

Vienna:git pieter$ git branch -r

origin/HEAD

origin/html

origin/maint

origin/man

origin/master

These remote tracking branches represent the branch heads of the remote repository as of the time you last fetched from the remote. You can use them almost like local branch heads, for example with git log origin/maint, but there's an important exception: git checkout. Trying to checkout a remote tracking branch will leave you with a [[[#detached|detached HEAD]]]. So if you want to work on some of these branches, you should create a local branch head for your work and check that out:

Vienna:git pieter$ git checkout -b maint origin/maint

Branch maint set up to track remote branch refs/remotes/origin/maint.

Switched to a new branch "maint"

Vienna:git pieter$

This will create and checkout a branch head called "maint" that starts at the same commit that the remote tracking branch "origin/maint" currently references.

How do I share a git public repository and use it in a CVS way?

You can use git --bare init --shared=group (or git --bare init --shared=all for unprivileged gitweb) to initialize a shared repository. All users belonging to your group now have permissions to push their changes to the repository. It's O.K. that refs aren't group writable, it's enough that the directory is.

* See Git's [cvs-migration](http://www.kernel.org/pub/software/scm/git/docs/gitcvs-migration.html) doc, "Emulating the CVS Development Model" section for details.

How do I share a git repository using POSIX ACLs?

You can use setfacl to give permission to individual users on a repository. Suppose your login is alice, and you want to give permission to charlie and bob. Giving the permissions is done by:

setfacl -Rm u:bob:rwx '/home/alice/path/to/repo'

setfacl -Rm u:charlie:rwx '/home/alice/path/to/repo'

Now, we have to set the default ACL, so than new files be created with the same permissions:

setfacl -Rm d:u:bob:rwx '/home/alice/path/to/repo'

setfacl -Rm d:u:charlie:rwx '/home/alice/path/to/repo'

setfacl -Rm d:u:alice:rwx '/home/alice/path/to/repo'

If needed, we also give the permission to traverse the filesystem up to the repository:

setfacl -m u:bob:x '/home/alice/path/to'

setfacl -m u:charlie:x '/home/alice/path/to'

setfacl -m u:bob:x '/home/alice/path'

setfacl -m u:charlie:x '/home/alice/path'

setfacl -m u:bob:x '/home/alice'

setfacl -m u:charlie:x '/home/alice'

Git version <= 1.7.0 will break your ACL mask and create unreadable pack files if you have a umask restricting group access (e.g. umask 077). One can work around this problem by making the repository shared:

git config core.sharedRepository group

How can I add a diff of the commit into the commit message window?

Just call git commit with -v flag:

git commit -v

How would I use "git push" to sync out of a host that I cannot pull from?

When you work on two machines, each with its own work tree, a typical way to synchronise between them is to run git pull from each other. However, you may be able to make a TCP connection only in one direction but not in the other direction in certain situations (e.g. you have a firewall between them, one machine is not running an ssh server, or one machine has intermittent connectivity). Suppose you start a project on machine A (mothership), and clone from there to a machine B (satellite). You work on B and would want to slurp the change back to your repository on machine A. Even if you wanted to, you cannot run git fetchon machine A to fetch from B, as B does not allow incoming connections. What should you do in such a case?

You can realize that a push is a mirror operation of a fetch and take advantage of it. If B were *not* firewalled, you would instead run fetch on A from B. And such a fetch is arranged to fetch 'master' from B and store that in 'refs/remotes/B/master' in A.

Pushing 'master' branch on B to 'master' branch on A, however, is never what you would want to do in such a case. Push is a reverse of fetch in the sense that it propagates the objects and update the branch tips, but does not touch the working tree in the target repository, and you (and git) will be utterly confused when you go back to machine A after you update 'master' that way, as the contents of 'master' have changed, but your working tree still reflects an older state.

So the simple solution to work around such a firewalled setup is to push 'master' from B into 'refs/remotes/B/master' of A, like this:

machineB$ git push machineA:repo.git master:refs/remotes/B/master

When you go back to machineA to work further, it is as if you did a git fetch from machineB, like this:

machineA$ git fetch machineB:repo.git master:refs/remotes/B/master

When you are ready to integrate the changes you did on machineB into the master branch on machineA, you can:

machineA$ git merge B/master ;# shorthand for refs/remotes/B/master

This is no different from the case where you actually pulled from B on A. You can set up your .git/config file to largely automate the above git push, so that you can just say:

machineB$ git push

See also <http://thread.gmane.org/gmane.comp.version-control.git/42506/focus=42685>

How do I check out the tree at a particular tag?

So you cloned that shiny repository and now would like to get the working tree to the state as of some tag? Use git tag -l to list all available tags, then just do a git checkout TAGNAME. If you want to build some work on it, use git checkout -b newbranch TAGNAME instead. If you want to return to your latest revision later, do git checkout ORIGINALBRANCH (usually it is master, you can list them with git branch).

How to share objects between existing repositories?

Do

echo "/source/git/project/.git/objects/" > .git/objects/info/alternates

and then follow it up with

git repack -a -d -l

where the '-l' means that it will only put *local* objects in the pack-file (strictly speaking, it will put any loose objects from the alternate tree too, so you'll have a fully packed archive, but it won't duplicate objects that are already packed in the alternate tree).

How to stop sharing objects between repositories?

To copy the shared objects into the local repository, repack without the -l flag

git repack -a

Then remove the pointer to the alternate object store

rm .git/objects/info/alternates

(If the repository is edited between the two steps, it could become corrupted when the alternates file is removed. If you're unsure, you can use git fsck to check for corruption. If things go wrong, you can always recover by replacing the alternates file and starting over).

How do I tell git to ignore files?

You can put shell-style globs (e.g. \*.o) in either .git/info/exclude or .gitignore.

.git/info/exclude is local to your repository only, and not shared by others who might fetch from your repository.

.gitignore is more commonly used, as it can be checked into the repository and thereby automatically shared with all users of the project.

* See the

[git-ls-files(1)](http://www.kernel.org/pub/software/scm/git/docs/git-ls-files.html) man page, section "Exclude Patterns" for details.

How do I tell git to ignore tracked files?

A common reason to ask this question are configuration files which are added to a project as an example or default, but which individual developers may want to change. Keeping uncommitted changes around is undesirable in the long run, because git diff will always show the changes and they will be added for commit if commands such as commit -a or add -u are used.

This problem can be solved in different ways:

* Option a: Rename the configuration file to config.template or something like that, and tell the user to use it as a template. The install script for your system could also copy the template configuration to the appropriate location.
* Option b: Allow configuration options to be overwritten in a seprate file. For example, in Makefiles it is common to add a line -include config.mak, which allows users to add custom configuration in config.mak, rather than changing the Makefile itself.
* Option c: Make git ignore changes to the configuration file. There is no automatic centralized way to do that. But on a per-clone basis you can tell git to do it using either git update-index --assume-unchanged <file> or *sparse checkout* (see the corresponding section in [git-read-tree(1)](http://www.kernel.org/pub/software/scm/git/docs/git-read-tree.html) for details). Note that neither feature was designed for that purpose. *Use at own risk.*

How do I save some local modifications?

Sometimes it is necessary to put some local changes aside, and come back to them later (typically, when one hits an easily-fixable bug in the middle of non-trivial work, and wants to fix the bug before anything else).

With recent versions of git, you can use ["git stash"](http://www.kernel.org/pub/software/scm/git/docs/git-stash.html) to save temporary modifications and come back to a "clean" tree, and then "git stash apply" to re-apply it.

Alternatively:

One can use a temporary branch, to merge later:

$ git checkout -b tempBranch

$ git commit -a -m "to test"

where tempBranch is the unique (original) throwaway branch name.

Another solution is to save changes in a patch, to apply later:

$ git diff --binary HEAD > tempPatch.diff

$ git reset --hard

(*warning:* git reset --hard removes changes to the working tree!)

How to manually resolve conflicts when Git failed to detect rename?

What to do when you renamed a bunch of files, the merge is having a hard time autoresolving, and you have a couple of conflicts? Suppose the project originally had util/endian.h, and during the course of your development you moved it to src/util/endian.h. Your friend kept working on util/endian.h and it is time to merge the two branches. Sometimes recursive merge strategy (the default) detects this situation, and merge the changes your friend made to util/endian.h to src/util/endian.h without problems (you may still have to resolve the conflict in the contents of the file). But when git thinks you removed util/endian.h and created an unrelated src/util/endian.h file, you will see merge conflicts "your side removed, other side modified" on util/endian.h.

First, check which files have conflicts to resolve using git ls-files --unmerged. Then you can see the blob object names for each merge stage; stage1 is from the common ancestor and stage3 is from your friend's branch. When this type of conflict happens, you don't have stage2 for src/util/endian.h, because that path is "only your side created, other side did nothing" case, and (incorrectly) cleanly resolved:

$ git ls-files --unmerged --abbrev

...

100755 33cd1f76... 1 util/endian.h

100755 7f531bb7... 3 util/endian.h

Then to do merge between the versions for the "undetected rename" file, extract two blobs (whose sha1 you have from [git-ls-files(1)](http://www.kernel.org/pub/software/scm/git/docs/git-ls-files.html) or [git-status(1)](http://www.kernel.org/pub/software/scm/git/docs/git-status.html) output) to temporary files, and run "merge" command by hand, e.g.

$ git cat-file blob 33cd1f76 >endian.h-1

$ git cat-file blob 7f531bb7 >endian.h-3

$ merge src/util/endian.h endian.h-1 endian.h-3

(that is merge yours, original and his). Of course instead of merge from RCS you can use your favorite 3-way file merge program, e.g. [Ediff3 from Emacs](http://www.delorie.com/gnu/docs/emacs/ediff_2.html) (see also [MergingWithEmacs](http://www.selenic.com/mercurial/wiki/index.cgi/MergingWithEmacs) at Mercurial wiki),[vimdiff/gvimdiff](http://www.vim.org/htmldoc/diff.html" \l "vimdiff), [Meld](http://meld.sourceforge.net/), [xxdiff](http://furius.ca/xxdiff/) or [KDiff3](http://kdiff3.sourceforge.net/). In newer versions of Git you could use "*:<stage>:<filename>*" instead of SHA1 to extract files to temporary files (:1:util/endian.h and :3:util/endian.h, respectively); check [RevisionSpecification](https://git.wiki.kernel.org/index.php/RevisionSpecification" \o "RevisionSpecification) and references therein.

Once you come up with the desired state in your file (src/util/endian.h in our example), then you have to inform Git that file was renamed, i.e. say "git update-index --remove util/endian.h" in our example (removing the file from working directory as well, if it exist there) and then "git update-index src/util/endian.h" (you should have it already in the index, so you do not have to say --add).

There is an experimental [script](https://gist.github.com/894374) that aims at making it easy to reassociate the base, left and right version of a file after a merge conflict. Feedback appreciated. It was only tested on CONFLICT (delete/modify) cases yet.

How to revert file to version from current commit?

If you messed up a file, or removed it accidentally, and want to revert file change to version at current commit, you can use:

git checkout HEAD -- <file>

If you want to revert to version in index (i.e. the last version you ran git add on), use

git checkout -- <file>

How to view an old revision of a file or directory?

Use command "git show" with a colon and filename:

git show <commit>:path/file

The <commit> can be commit id, branch name, tag, relative pointer like HEAD~2 etc. If you don't give any path or file (i.e. just <commit>:), git will display the file listing of repository's root directory. Examples:

git show v1.4.3:git.c

git show f5f75c652b9c2347522159a87297820103e593e4:git.c

git show HEAD~2:git.c

git show master~4:

git show master~4:doc/

git show master~4:doc/ChangeLog

How can I tweak the date of a commit in the repo?

If you wish to tweak the authorship date of HEAD, then the easiest way is as follows (the -C HEAD is just to bypass editing the commit message):

$ git commit --amend --date='<see documentation for formats>' -C HEAD

Setting the commitership date requires the use of the COMMITER\_DATE environment variable; see the later script.

You can use programs like GNU date to figure out the new date in a reasonable way:

$ current\_date=$(git log -1 --format=format:%ai)

$ new\_date=$(date -d "$current\_date - 1 hour - 22 minutes" --rfc-2822)

$ git commit --amend --date="$new\_date" -C HEAD

If you want to set a different time zone as well, then just set the TZ environment variable appropriately (see [`info "(libc)TZ Variable"'](http://www.gnu.org/s/libc/manual/html_node/TZ-Variable.html#TZ-Variable); in particular, change the previous new\_date=... line to the following in order to render the date in the U.S. Pacific time zone:

$ new\_date=$(TZ=:US/Pacific date -d "$current\_date - 1 hour - 22 minutes" --rfc-2822)

If you only want to change the time zone or tweak the date by some easily computed number of seconds, then it's probably easier to work with the raw Unix time stamp that git stores internally. For instance, to use the same date as HEAD but change the time zone to UTC, you could do the following (notice the use of %at to get the raw timestamp):

$ current\_date=$(git log -1 --format=format:%at)

$ git commit --amend --date="$new\_date +0000" -C HEAD

You can pass raw times stamps to GNU date as follows:

$ current\_date=$(git log -1 --format=format:%at)

$ TZ=:US/Pacific date -d @1301500895

Wed Mar 30 09:01:35 PDT 2011

If you can't remember the current time zone offet, then just ask GNU date; for instance, if your current time zone is UTC, then:

$ date +%z

+0000

If I want to find out the *current* offset for, say, the U.S. Pacific time zone, try the following (GNU date accounts for daylight saving time for whatever date it's given):

$ TZ=:US/Pacific date +%z

<either -0700 or -0800 depending on the current date>

$ TZ=:US/Pacific date -d 2011-01-01 +%z

-0800

$ TZ=:US/Pacific date -d 2011-08-01 +%z

-0700

If you want to tweak the date of one of HEAD's ancestors, then just use git bisect -i to select the commits at which you'd like to stop while rebasing, so that you can modify them. Or, if your alterations are rather uniform, then you could use the x directive of git rebase -i to specify the commands that should be run while rebasing; a more general version of such a programmatic approach is to use git filter-branch, as in this following script (which shows date-related environment variables that can be used):

#!/bin/sh

#

# Rewrite all branches to modify the date of one specific commit in a repo.

#

# Sample date format: Fri Jan 2 21:38:53 2009 -0800

# ISO8601 and RFC822 dates will also work.

#

# Note: filter-branch is picky about the commit argument. As of 1.7.0.4,

# a hex ID will work, the symbolic revision HEAD will fail silently,

# and the usability of more exotic rev specs was not tested by the author.

#

# Copyright (c) Eric S. Raymond, 2010-08-01. BSD terms apply (if anybody really thinks that this

# script is long and non-obvious enough to fall under copyright law).

#

commit="$1"

date="$2"

git filter-branch --env-filter \

"if test \$GIT\_COMMIT = '$commit'

then

export GIT\_AUTHOR\_DATE

export GIT\_COMMITTER\_DATE

GIT\_AUTHOR\_DATE='$date'

GIT\_COMMITTER\_DATE='$date'

fi" &&

rm -fr "$(git rev-parse --git-dir)/refs/original/"

How do I make a diff between two arbitrary files in different revisions?

With command "git diff" you can refer to a commit and file pair:

git diff <commit1>:path/file <commit2>:otherpath/otherfile

As usual, commits can be commit ids, branch names, tags or relative references like HEAD~2. For example:

git diff 06de3718c389fd5038697151c49519f6e9f2dbe0:ChangeLog HEAD~2:ChangeLog.old

How to fix a broken repository?

(See also [GitFaq#salvage](https://git.wiki.kernel.org/index.php/GitFaq" \l "salvage" \o "GitFaq).)

As Linus said ([on gmane](http://thread.gmane.org/gmane.comp.version-control.git/40893/focus=40896)):

"Generally, the best way to fix things is (I've written this up at somewhat more length before, but I'm too lazy to find it):

* back up all your state so that anything you do is re-doable if you corrupt things more!
* explode any corrupt pack-files
  + See "man git-unpack-objects", and in particular the "-r" flag. Also, please realize that it only unpacks objects that aren't already available, so you need to move the pack-file away from its normal location first (otherwise git-unpack-objects will find all objects that are in the pack-file in the pack-file itself, and not unpack anything at all)
* replace any broken and/or missing objects
  + This is the challenging part. Sometimes (hopefully often!) you can find the missing objects in other copies of the repositories. At other times, you may need to try to find the data some other way (for example, maybe your checked-out copy contains the file content that when hashed will be the missing object?).
* make sure everything is happy with "git fsck --full"
* repack everything to get back to an efficient state again.

And remember: git does \_not\_ make backups pointless. It hopefully makes backups \*easy\* (since cloning and pulling is easy), but the basic need for backups does not go away!"

In another thread ([on gmane](http://thread.gmane.org/gmane.comp.version-control.git/64091/focus=64213)) Linus explained how to find and fix a corrupt object:

"First off, move the corrupt object away, and \*save\* it. The most common cause of corruption so far has been memory corruption, but even so, there are people who would be interested in seeing the corruption - but it's basically impossible to judge the corruption until we can also see the original object, so right now the corrupt object is useless, but it's very interesting for the future, in the hope that you can re-create a non-corrupt version.

So:

$ mv .git/objects/4b/9458b3786228369c63936db65827de3cc06200 ../

This is the right thing to do, although it's usually best to save it under it's full SHA1 name (you just dropped the "4b" from the result ;).

Let's see what that tells us:

$ git fsck --full

> broken link from tree 2d9263c6d23595e7cb2a21e5ebbb53655278dff8

> to blob 4b9458b3786228369c63936db65827de3cc06200

> missing blob 4b9458b3786228369c63936db65827de3cc06200

Ok, I removed the "dangling commit" messages, because they are just messages about the fact that you probably have rebased etc, so they're not at all interesting. But what remains is still very useful. In particular, we now know which tree points to it!

A) If you have a "broken link" message:

Now you can do:

git ls-tree 2d9263c6d23595e7cb2a21e5ebbb53655278dff8

which will show something like

100644 blob 8d14531846b95bfa3564b58ccfb7913a034323b8 .gitignore

100644 blob ebf9bf84da0aab5ed944264a5db2a65fe3a3e883 .mailmap

100644 blob ca442d313d86dc67e0a2e5d584b465bd382cbf5c COPYING

100644 blob ee909f2cc49e54f0799a4739d24c4cb9151ae453 CREDITS

040000 tree 0f5f709c17ad89e72bdbbef6ea221c69807009f6 Documentation

100644 blob 1570d248ad9237e4fa6e4d079336b9da62d9ba32 Kbuild

100644 blob 1c7c229a092665b11cd46a25dbd40feeb31661d9 MAINTAINERS

...

and you should now have a line that looks like

10064 blob 4b9458b3786228369c63936db65827de3cc06200 my-magic-file

in the output. This already tells you a \*lot\* it tells you what file the corrupt blob came from!

Goto C. :)

B) If you don't have a "broken link" message:

ex:

$ git fsck --full

missing blob d1209bf8031138df13d5d0eb74fce173a89e0f0d

missing blob 50cf67abff3778083c581408f17be136a94a8f6d

Try doing a commit. It should give you an error message similar to the following:

$ git commit -m "fixing git repo"

error: invalid object 100644 d1209bf8031138df13d5d0eb74fce173a89e0f0d for 'src/removepoints.cpp'

error: Error building trees

This way you also know what file the corrupt blob came from!

C)

Now, it doesn't tell you quite enough, though: it doesn't tell what \*version\* of the file didn't get correctly written! You might be really lucky, and it may be the version that you already have checked out in your working tree, in which case fixing this problem is really simple, just do

git hash-object -w my-magic-file

again, and if it outputs the missing SHA1 (4b945..) you're now all done!

But that's the really lucky case, so let's assume that it was some older version that was broken. How do you tell which version it was?

The easiest way to do it is to do

git log --raw --all --full-history -- subdirectory/my-magic-file

and that will show you the whole log for that file (please realize that the tree you had may not be the top-level tree, so you need to figure out which subdirectory it was in on your own), and because you're asking for raw output, you'll now get something like

commit abc

Author:

Date:

...

 :100644 100644 4b9458b... newsha... M somedirectory/my-magic-file

commit xyz

Author:

Date:

...

 :100644 100644 oldsha... 4b9458b... M somedirectory/my-magic-file

and this actually tells you what the \*previous\* and \*subsequent\* versions of that file were! So now you can look at those ("oldsha" and "newsha" respectively), and hopefully you have done commits often, and can re-create the missing my-magic-file version by looking at those older and newer versions!

If you can do that, you can now recreate the missing object with

git hash-object -w <recreated-file>

and your repository is good again!

(Btw, you could have ignored the fsck, and started with doing a

git log --raw --all

and just looked for the sha of the missing object (4b9458b..) in that whole thing. It's up to you - git does \*have\* a lot of information, it is just missing one particular blob version.

Trying to recreate trees and especially commits is \*much\* harder. So you were lucky that it's a blob. It's quite possible that you can recreate the thing."

How to remove all broken refs from a repository?

So fixing the repository as per [GitFaq#fix-broken-repo](https://git.wiki.kernel.org/index.php/GitFaq" \l "fix-broken-repo" \o "GitFaq) did not work out, or maybe you just want to cut your losses and get on with it? This is how you do it. Again, it is advisable to backup (cp -a) your repository in case the following process removes more data than you like.

1. Get a list of all refs with missing commits, trees or blobs:

$ git for-each-ref --format='%(refname)' | while read ref; do git rev-list --objects $ref >/dev/null || echo "in $ref"; done

It may not be possible to remove some of those refs using the regular branch -d or tag -d commands, since they will die if git notices the corruption. So use the plumbing command git update-ref -d $ref instead. Note that in case of local branches, this command may leave stale branch configuration behind in .git/config. It can be deleted manually (look for the [branch "$ref"] section).

2. After all refs are clean, there may still be broken commits in the reflog. You can clear all reflogs using git reflog expire --expire=now --all. If you do not want to lose all of your reflogs, you can search the individual refs for broken reflogs:

$ (echo HEAD; git for-each-ref --format='%(refname)') | while read ref; do git rev-list -g --objects $ref >/dev/null || echo "in $ref"; done

(Note the added -g option to git rev-list.) Then, use git reflog expire --expire=now $ref on each of those.

When all broken refs and reflogs are gone, run git fsck --full in order to check that the repository is clean. Dangling objects are Ok.

How to set up a git server?

Look for "Repository Administration" in the [everyday git document](http://www.kernel.org/pub/software/scm/git/docs/everyday.html#Repository%20Administration).

To setup a git server over http, see [the relevant document](http://www.kernel.org/pub/software/scm/git/docs/howto/setup-git-server-over-http.txt).

As well, it's worth mentioning both [gitosis](http://scie.nti.st/2007/11/14/hosting-git-repositories-the-easy-and-secure-way) and [gitolite](http://github.com/sitaramc/gitolite) as excellent tools for setting up secure git servers.

How to create the first project?

See the [Git tutorial](http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html).

How do I publish my repo via SFTP?

At the moment, git is not able to use a (dumb) sftp protocol for pushing. There is a work around, though: [Use sshfs](http://blog.flameeyes.eu/2007/02/14/how-to-use-git-via-sftp-only). But make sure that you execute 'git update-server-info' in the pushed repository!

Alternatively, you can use whatever means to mirror your .git/ directory to the server (rsync, scp -r, ...). But make sure that 'git update-server-info' was run in that repository \_before\_ mirroring!

How do I do a quick clone without history revisions?

If you just want to checkout the latest source code of a project which may have a very large repo, you can use

git clone --depth 1 your\_repo\_url

How do I use git for large projects, where the repository is large, say approaching 1 TB, but a checkout is only a few hundred MB? Will every developer need 1 TB of local disk space?

In general, git is not a viable solution for the the case of a large repository with relatively small individual checkouts. However, if developers do not intend to clone, fetch, push into or push from their repositories, then use shallow clones

git clone --depth 1 <url>

How do I obtain a list of files which have changed in a given commit?

git diff --name-only <commit>^!

or (to get also the commit message):

git show --name-only <commit>

How do I remove all the old objects after using filter-branch?

Note: It is recommended you backup your repository before using git filter-branch.

First, remove the backup references filter-branch created in refs/original:

git for-each-ref --format='%(refname)' refs/original | \

while read ref

do

git update-ref -d "$ref"

done

Then clean your reflogs:

git reflog expire --expire=0 --all

And finally, repack and drop all the old unreachable objects:

git repack -ad

git prune # For objects that repack -ad might have left around

How do I mirror a SVN repository to git?

Initialize bare repo:

mkdir project.git

cd project.git

git init --bare

git symbolic-ref HEAD refs/heads/trunk

cd ..

Initialize "fetch"-repo:

git svn clone -s svn://.../project

cd project

git remote add bare /path/to/project.git

git config --unset remote.bare.fetch

git config remote.bare.push 'refs/remotes/\*:refs/heads/\*'

git push bare

Update "fetch" repo:

cd project

git svn fetch

Update the "bare" repo by pushing to it:

cd project

git push bare

To clone the mirror repository use

git clone git://$your\_repo

To enable usage of git svn in the clone use

git svn init -s --prefix=origin/ svn://.../project

**Troubleshooting**

If git svn clone return something like this:

Using higher level of URL: svn://.../project/svn/trunk => svn://.../project/svn

you must clone without *trunk* dir.

How do I edit the root commit?

git rebase -i allows you to conveniently edit any previous commits, except for the root commit. The following commands show you how to do this manually.

# tag the old root

git tag root `git rev-list HEAD | tail -1`

git checkout -b new-root root

# edit...

git commit --amend

# check out the previous branch

git checkout @{-1}

# replace old root with amended version

git rebase --onto new-root root

# cleanup

git branch -d new-root

git tag -d root

How do I clone a subdirectory?

Currently, you cannot. Sparse clone is [in development](http://mid.gmane.org/1283645647-1891-1-git-send-email-newren@gmail.com).

In the meantime, you can use the subdirectory-filter of git filter-branch to extract a subdirectory. You can also merge changes back using the [subtree merge strategy](http://www.kernel.org/pub/software/scm/git/docs/howto/using-merge-subtree.html). Or you can use [submodules](http://www.kernel.org/pub/software/scm/git/docs/user-manual.html" \l "submodules).

It is possible, however, to download subdirectories or even individual files if the server enables the upload-archive service. The following example retrieves the source code for the main git executable of version 1.6.0.

git archive --remote=git://repo.or.cz/git.git v1.6.0 git.c > git.c.tar

Use git ls-remote to list available versions.

How do I make existing non-bare repository bare?

After making sure that there are no uncommitted changes, etc.:

$ mv repo/.git repo.git

$ git --git-dir=repo.git config core.bare true

$ rm -rf repo

The problem with the above process is that it doesn't take into account future internal changes of Git. A safer method is to let Git handle all the internal settings for you by doing something like this.

\* ssh to remote server

\* git clone --bare -l <path\_to\_repos> <new\_dir>

\* renamed old repository directory

\* renamed new repository dir to what old repository used to be.

How do I find large files?

Save the following script to git-find-large in a directory listed in $PATH. The command git find-large will then list 10 paths corresponding to the largest blobs in your repository. Note that only one path is listed per blob, even if the blob has copies or different names in history.

#!/bin/sh

#

# From http://mid.gmane.org/20090710114316.GA6880@atjola.homenet

#

usage() {

echo "usage: `basename $0` [<limit>]"

exit 1

}

limit=10

if test $# -gt 1

then

usage

elif test $# -eq 1

then

limit=$1

fi

git rev-list --all --objects |

sed -n $(git rev-list --objects --all |

cut -f1 -d' ' | git cat-file --batch-check | grep blob |

sort -n -k3 | tail -n$limit | while read hash type size;

do

echo -n "-e s/$hash/$size/p ";

done) |

sort -n -k1

How do I recover uncommitted changes?

Changes that were never added or stashed are not known to git and therefore cannot be recovered. Committed changes can generally be recovered using the reflog. The following script can help you recover changes that were added to the index, but never committed. It shows a list of dangling blob sha1's. These are generally objects that were discarded from the index. A blob can be dumped to standard output using git show <sha1>. To dump all dangling blobs into files use git fsck --lost-found.

#!/bin/bash

git fsck --no-full | grep "^dangling blob" | cut -f3 -d' ' |

while read rev

do

f=.git/objects/${rev:0:2}/${rev:2:38}

if ! test -f $f

then

s=0

else

s=$(stat -c %Y $f)

fi

echo $s $rev

done | sort -n |

while read s rev

do

echo $(date -d "@$s" -R) $rev

done

How do I merge a patch from a gmane.org URL?

You can get the raw article by appending /raw to a message URL (reachable from the "Direct link" link at the bottom of the thread view). Then, using wget and git am, you can apply a patch with a single command-line like:

wget 'http://article.gmane.org/gmane.comp.version-control.git/149176/raw' -O - | git am -3

How do I clone a repository with all remotely tracked branches?

Suppose you have a repository configured with several remotes: *origin* (the default), *foreign1*, *foreign2*. You want to do a clone but do not want to set up all foreigns again and still be able to checkout any branch from*foreign1* or *foreign2* (you have *origin* for free).

There's a git-clone option *--mirror* which sets up all remotely tracked branches. So far this is a fundamental knowledge obtained from frequent documentation readings \*cough\*. But, *--mirror* also implies *--bare* option, ie. the clone will will not be created as a working copy and you cannot checkout the branches. Here's how to fix it:

$ git clone --mirror original-repo.git /path/cloned-directory/.git # (1)

$ cd /path/cloned-directory

$ git config --bool core.bare false # (2)

$ git checkout anybranch

(1): Please note that the clone target directory name mimics a working copy of a repository!

(2): Bare repository is turned into working copy as you can see in other tips on this page.

You can check all available branches:

$ git branch -a

anybranch

origin/HEAD

origin/master

origin/anybranch

foreign1/branch1

foreign1/branch2

foreign2/branch2

How do I remove my uncommitted changes from branch A and add them to a new branch B?

This question is rooted in a misunderstanding. *Uncommitted changes are not part of any branch.* This goes for both unstaged and staged changes. They are a property of the worktree.

*git checkout* in branch-switching mode will take care to "move along" any uncommitted changes in the tree when completing the branch switch. This may occasionally cause it to refuse the switch, either because it would have to attempt a nontrivial merge inside a file or because it would have to overwrite an untracked file with its tracked counterpart in the target branch.

Most users asking this question did some work on e.g. 'master', but before committing it realized they should commit it to a new branch 'topic' instead of 'master'. This is as easy as

$ git checkout -b topic

# and when you have the changes all prepared

$ git add <...>

$ git commit

and in fact in this case the branch switch will never fail.

How do I use external diff tools?

The example shows how to use ediff. In .gitconfig, write

[alias]

ediff = difftool -t ediff

[difftool "ediff"]

cmd = /your/ediff/wrapper.sh \"$LOCAL\" \"$REMOTE\"

Then /your/ediff/wrapper.sh should look like:

#!/bin/sh

emacsclient --eval "(ediff-files \"$1\" \"$2\")"

Now, you can use ediff by   
$ git ediff <normal git-diff arguments>.  
"git-diff" can also be used as before.

Error diagnostic

Git push fails with "fatal: The remote end hung up unexpectedly"?

If, when attempting git push, you get a message that says:

fatal: The remote end hung up unexpectedly

There are a couple of reasons for that, but the most common is that authorization failed. You might be using a git:// URL to push, which has no authorization whatsoever and hence has write access disabled by default. Or you might be using an ssh URL, but either your public key was not installed correctly, or your account does not have write access to that repository/branch.

Git commit is dying telling me "fatal: empty ident <user@myhost> not allowed", what's wrong?

Make sure your Full Name is not empty in chsh or the 5th field of your user line in /etc/passwd isn't empty. You can also set the GIT\_AUTHOR\_NAME environment variable. If your @myhost is empty make sure your hostname is correctly set. Use git var -l to make git display user identity variables.

* See [git-var(1)](http://www.kernel.org/pub/software/scm/git/docs/git-var.html)

Why won't git let me change to a different branch?

Using git checkout <branch> or git checkout -b <branch> it just says:

fatal: Entry 'foo.c' not uptodate. Cannot merge.

You have changes to files in your working directory that will be overwritten, removed or otherwise lost if the checkout and change to the new branch were to proceed. To fix this you may either check your changes in, create a patch of your changes and revert your files, or use the -m flag like this:

$ git checkout -m -b my-branch

refs/heads/pu: does not fast forward to branch 'pu'

The "pu" branch often won't fast forward because some commits have been completely deleted in it since the last time you pulled.

If you want to track it, add a plus (+) sign to the proper line in your .git/config file, like this:

[remote "origin"]

fetch = +refs/heads/pu:refs/remotes/origin/pu

Which tells git to deal with the problem for you by simply skip the fast forward check (overwriting your old ref with the new one). Or you can just delete that line completely if you don't want to track the pu branch at all.

It is conceivable that in future versions of git we might want to be able to mark some branches "this is expected to be rewound" explicitly and make the clone operation to take notice, to give you the plus sign automatically.

protocol error: bad line length character

If you see the following errors:

fatal: protocol error: bad line length character

error: failed to push to 'git.example.com:/Repo/Project.git'

fatal: The remote end hung up unexpectedly

It likely means you have some extraneous characters, info message or something upon logging into ssh in command mode.

To test this, do:

ssh user@git.example.com echo testing commands

You should only see testing commands returned. If there are any other characters, you should examine your dot shell rc file to find any echo or other commands that may produce output.

"unable to chdir or not a git archive" while pushing

If you see the following errors:

fatal: 'git.exaple.com/wrong-path': unable to chdir or not a git archive

fatal: The remote end hung up unexpectedly

error: failed to push to 'git.exaple.com/wrong-path'

The most likely cause of this error is that you have incorrectly specified the path in the git url.

See the [git-push(1)](http://www.kernel.org/pub/software/scm/git/docs/git-push.html) man page for more info on valid git urls.

"needs update" and "not uptodate"

foo: needs update

fatal: Entry 'frotz' not uptodate. Cannot merge.

*TO DO:* *Explain first of those messages*

Second of those error messages is described in [git-checkout(1)](http://www.kernel.org/pub/software/scm/git/docs/git-checkout.html) man page, in the "EXAMPLES" section. It means that you have local modifications to 'frotz', which would be lost on checkout. You can give '-m' option to git checkout, which would try three-way merge.

Sometimes the solution is to commit.

"git-receive-pack: command not found" on push/fetch/pull

Basically the problem is that 'git-receive-pack' is not in the default $PATH on the remote end. You can see the problem using:

$ ssh servername 'echo $PATH'

/usr/bin:/bin

$

Whereas you probably installed git to your $HOME or something like that. The workarounds include;

* Install git to /usr/bin
* Making sure you have the correct path set up in .bashrc (not only .bash\_profile) if your shell is bash or .zshenv if your shell is zsh.
* Specify --receive-pack=PATH/TO/git-receive-pack at push time
* Set remote.<name>.receivepack using git config (see git-config(1))

My username contains a '@', I can't clone through HTTP/HTTPS

$ git clone https://user@mail.com@gitserver.com/path/

[...]

error: Couldn't resolve host 'user@mail.com' while accessing https://user@mail.com@gitserver.com/path/info/refs

Solution : URL-escape the '@' sign in your username, i.e. replace it with %40, like git clone https://user%40mail.com@gitserver.com/path/.

Explanation : Git delegates the URL handling to curl, which doesn't allow '@' signs in usernames because [RFC 3986](http://tools.ietf.org/html/rfc3986) doesn't allow it. See [this bug report](http://sourceforge.net/tracker/index.php?func=detail&aid=2826621&group_id=976&atid=100976) for example.

Importing from other revision control systems

See the [[InterfacesFrontendsAndTools#rcs-interaction|[InterfacesFrontendsAndTools](https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools)]] page ("Interaction with other Revision Control Systems" section), for a good reference on how to interact with other revision control systems.

Note that you can add/import old history from other revision control system later, and join the histories using [Graft Points](https://git.wiki.kernel.org/index.php/GraftPoint).

Can I import from tar files (archives)?

To import from archives (one archive file per version), to make Git know which files changed between versions despite the fact that time stamps on everything changed (i.e. --atime-preserve option of tar didn't work), use

$ git update-index --refresh

between versions, so the index *thinks* things are newer. It won't touch the "really changed" files.

Can I import from CVS?

Yes, use [git-cvsimport(1)](http://www.kernel.org/pub/software/scm/git/docs/git-cvsimport.html), which needs [CVSps](http://www.cobite.com/cvsps/) ([CVSps patches](http://ydirson.free.fr/en/software/scm/cvsps.html)), or use [parsecvs](http://gitweb.freedesktop.org/?p=users-keithp-parsecvs;a=summary), which parses directly ,v files. [cvs2svn](http://cvs2svn.tigris.org/) version 2.1 or later has a [git output mode](http://cvs2svn.tigris.org/cvs2git.html) in which it creates dump files that can be read by [git fast-import](http://www.kernel.org/pub/software/scm/git/docs/git-fast-import.html). Lastly, there is also [FromCVS toGit](https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools" \l "fromcvs.2Ftogit" \o "InterfacesFrontendsAndTools), which is a speedy solution with incremental import and branch support, but it currently does not support normal tags.

* See [the CVS migration doc](http://www.kernel.org/pub/software/scm/git/docs/gitcvs-migration.html) for more examples.
* See [Git - CVS Crash Course](http://git.or.cz/course/cvs.html) on [Git homepage](http://git.or.cz/)

Note: if you are getting "Unknown: error" when running cvsimport, this may be caused by an old version of cvs. In particular, if you are running Mac OS X and have set the CVS\_SERVER to ocvs, this causes git to also run ocvs, which does **not** work correctly with cvsimport. If you unset the environment variable, import will likely succeed.

Can I import from svn?

Yes, use [git-svn(1)](http://git-svn.yhbt.net/git-svn.html). Check also [git-svnconvert](https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools" \l "git-svnconvert" \o "InterfacesFrontendsAndTools) (in Ruby) in [InterfacesFrontendsAndTools](https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools" \o "InterfacesFrontendsAndTools). There is also an idea to use svnadmin dump ["dumpfile" format](http://svn.collab.net/repos/svn/trunk/notes/dump-load-format.txt) as an input for Subversion to Git repository conversion, (similar to what parsecvs does for CVS repositories).

Merge tracking information added to svn repositories by [svk](http://svk.bestpractical.com/), [svnmerge](http://www.orcaware.com/svn/wiki/Svnmerge.py), or version 1.5 or later of [subversion itself](http://subversion.tigris.org/merge-tracking/), is not properly converted.

* See [Git - SVN Crash Course](https://git.wiki.kernel.org/index.php/GitSvnCrashCourse" \o "GitSvnCrashCourse)

Can I import from arch/baz/tla?

Yes, use [git-archimport(1)](http://www.kernel.org/pub/software/scm/git/docs/git-archimport.html), or bzr-fast-export from `exporters` subdirectory of the [bzr-fastimport project](https://launchpad.net/bzr-fastimport).

Can I import from Perforce?

Yes, there is a number of importers, including one in contrib/ directory of Git source distribution. Search Git mailing list archives for others.

Can I import from Mercurial?

Yes, use for example Rocco Rutte's hg-fast-export (from fast-export.git on [repo.or.cz](http://repo.or.cz/)).

You can also use hg-git mercurial plugin (<http://hg-git.github.com/>) that adds to the Mercurial repository ability to push to and pull from a Git server.

Can I import from other SCMs?

Maybe, check if Tailor ([homepage](http://wiki.darcs.net/DarcsWiki/Tailor)) can do it. Take a look at [InterfacesFrontendsAndTools](https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools" \l "rcs-interaction" \o "InterfacesFrontendsAndTools) page ("Interaction with other Revision Control Systems" section)