

Home (/) > Tutorials (/TUTORIALS/) > GIT Commands and Examples (/TUTORIALS/Git-commands.html)



# Git Commands and Examples



Git command summary cheat

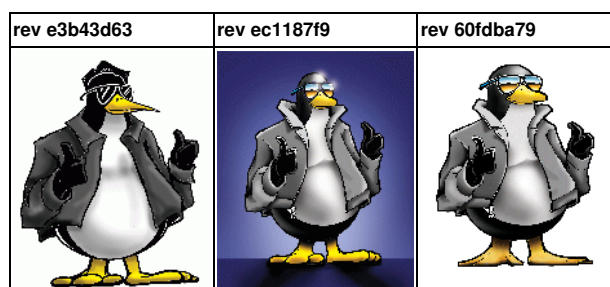
sheet, use, best practices, tips and examples. This tutorial covers version control with Git using the command line interface, GUI clients and examples which interface Git with GUI diff tools.

Emergency Alerts **save lives**



## Tutorial Contents:

- # Git Introduction
- # Git Commands
- # Git content properties
- # Git tag substitution
- # Git and Graphical diffs
- # Git GUI interface
- # Git Best Practices
- # Tips
- # Links
- # Books



## search

Search

| [Home Page \(/\)](#) | [Linux Tutorials \(/TUTORIALS/\)](#) | [Terms \(/YoLinux-Terms.html\)](#) | [Privacy Policy \(/privacy.html\)](#) | [Advertising \(/YoLinux-Advertising.html\)](#) | [Contact \(/YoLinuxEmailForm.html\)](#) |

## Related YoLinux Tutorials:

- Git Server Configuration (G Configuration.html)
- GitWeb Configuration (GitWeb Configuration.html)
- Git Integration With Trac (LinuxGitAndTracServer.html)
- Subversion Server and Trac Installation and Configuration (LinuxSubversionAndTracSe
- Subversion Repository Trac (SubversionRepositoryDataT
- Jenkins Automated Builds a Subversion (Jenkins.html)
- Cable Automated Builds an Subversion (CableBuildSyste
- Software development tools (LinuxTutorialSoftwareDevelk
- Subversion vs Clearcase (SubversionVsClearcase.htm
- CVS Intro (LinuxTutorialCV:
- RCS Intro (LinuxTutorialRC
- C++ Info, links (LinuxTutoric
- Java on Linux (LinuxTutoria
- YoLinux Tutorials Index (/Tl

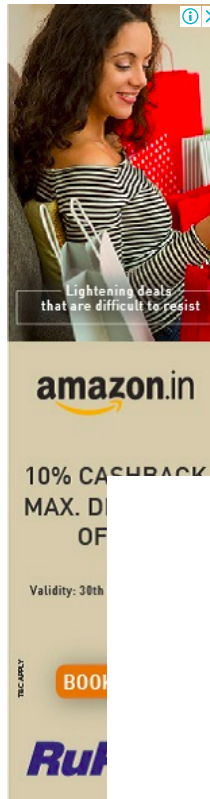
## Git Introduction:

Git is a software source code "Change Management" (CM) system for collaborative development. It maintains a history of file versions. Unlike typical client-server CM systems which "check-out" the latest or a particular version of the files, Git is a distributed CM system where the user has a local copy (a clone) of the entire repository which includes the entire history of all files. The user working with files in their local project work area, interacts with the local clone repository, adding, editing and deleting files and eventually committing their changes. The user can then share these changes to the local repository with a "push" or "pull" to other Git repositories including a server based central "origin" repository.

## Git Commands:

Git command summary cheat sheet:

Command	Description	Man Page
<code>git --help</code>	List Git commands	git (http://man.yolinux.c bin/man2html?cgi_com
<code>git --help command</code>	Help on given "command"	git help (http://man.yolir /cgi- bin/man2html?cgi_com help)
<code>git add path/filename</code> <code>git add file1 file2 file3</code> <code>git add --update</code> <code>git add --all</code>	Add a file or directory to Git CM control. This does not commit the files to the repository but merely adds files to "staging" status. Must also perform: <code>git commit</code> to enter changes in the repository. View staged files: <code>git ls-files --stage</code> Empty directories are not added. Git manages files only. --update: stage all tracked and modified files. Undo an add: <code>git reset HEAD path/filename</code>	git add (http://man.yol /cgi- bin/man2html?cgi_com add)



Ads by Google

Binary Software

Software Tools

Free Information Technology  
Magazines and Document  
Downloads



(<http://yolinux.tradepub.com>)

Free Information  
Technology **Software and  
Development** Magazine  
Subscriptions and  
Document Downloads  
(<http://yolinux.tradepub.com/?pt=cat&page=Infosoft>)

Command	Description	Man Page
<code>git blame filename</code>	Show file line numbers annotated with author information. Also see <code>git log</code>	<code>git blame</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit blame">http://man.yolinux.com/cgi-bin/man2html?cgi_commit blame</a> )
<code>git branch -a</code>	List all branches (remote and local)	<code>git branch</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit blame">http://man.yolinux.com/cgi-bin/man2html?cgi_commit blame</a> )
<code>git checkout parentPath/filename</code> <code>git checkout master</code> <code>git checkout e3b43d63</code>	Undo local changes. Revert changes. This directive is misleading as it implies a client-server action. Instead, it relies on the local repo and sets the working copy to the unchanged version, reverting changes. Checkout "master" will set the local work area to match the master branch. You can also specify the commit to which you want the local repo to view. Also see <code>git reset</code>	<code>git checkout</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit checkout">http://man.yolinux.com/cgi-bin/man2html?cgi_commit checkout</a> )
<code>git clean -df</code> <code>git clean -df</code> <code>git -n clean</code> <code>git -dn clean</code>	Remove files not managed by Git. -df: All unchecked files and directories are removed. -n: Dry run. Command not executed. -d: directories Clean has nothing to do with an Ant target of "clean" and is in no way related. Also see <code>git gc</code>	<code>git clean</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit clean">http://man.yolinux.com/cgi-bin/man2html?cgi_commit clean</a> )
<code>git clone http://git.megacorp.com/git/repo/projectx.git</code> <code>git clone ssh://user@git.megacorp.com:/srv/git/repo/projectx.git</code>	Copy/clone/create a local repository from the remote repository specified. [Potential Pitfall]: The following error often is a connectivity issue: fatal: http://git.megacorp.com/git/repo/projectx.git/info/refs not valid: is this a git repository? If a proxy error is keeping you from connecting. Fix: <code>unset http_proxy</code>	<code>git clone</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit clone">http://man.yolinux.com/cgi-bin/man2html?cgi_commit clone</a> )
<code>git commit</code> <code>git commit dir-path/filename</code> <code>git commit -m "Message goes here." filename</code>	Check-in (commit) local staged "working" files into local repository. Files must be staged for commit using <code>git add</code> Files are recursively located and committed into the local git repository. Git commits are Atomic, i.e. all files are committed or none are committed, no incomplete check-in.	<code>git commit</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit commit">http://man.yolinux.com/cgi-bin/man2html?cgi_commit commit</a> )
<code>git config --global user.name "John Doe"</code> <code>git config --global user.email jdoe@megacorp.com</code> <code>git config -l</code> <code>git config --global --unset user.password</code>	This configures ~/.gitconfig  Show your local GIT configuration for your local repo  Clear cached password. Will be prompted on next push/pull. [Potential Pitfall]: Error: fatal: Authentication failed for 'http://git.megacorp.com/git/repo/projectx.git/' Fix: <code>"git config --global --unset user.password"</code> to clear password cache.	<code>git config</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit commit">http://man.yolinux.com/cgi-bin/man2html?cgi_commit commit</a> )
<code>git diff</code> <code>git diff e3b43d63 60fdb479</code> <code>git diff directory/path</code>	Show file diffs between the working tree and local repo. Show file diffs between two commits.	<code>git diff</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit diff">http://man.yolinux.com/cgi-bin/man2html?cgi_commit diff</a> )
<code>git gc</code>	Cleanup unnecessary files and optimize/compress local repo.	<code>git gc</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit gc">http://man.yolinux.com/cgi-bin/man2html?cgi_commit gc</a> )
<code>git log</code> <code>git log --oneline</code> <code>git log directory/path/file</code> <code>git log --author="name"</code> <code>git log --graph --decorate --oneline</code>	Show the Git log messages. --oneline: brief description --graph --decorate: display branching like gitk except for text terminals. Note that the letter "q" will quit pagination mode.	<code>git log</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit log">http://man.yolinux.com/cgi-bin/man2html?cgi_commit log</a> )

Command	Description	Man Page
git <b>ls-files</b> --stage git <b>ls-files</b> --cached git <b>ls-files</b> --modified git <b>ls-files</b> --others git <b>ls-files</b> --deleted git <b>ls-files</b> --unmerged git <b>ls-files</b> --killed ...	List files staged. List cached files. List modified files. List other (un-tracked) files. List deleted files. List un-merged files. List killed files that need to be removed due to file/directory conflicts for checkout-index to succeed.	git ls-files ( <a href="http://man.yolinux.com/bin/man2html?cgi_command=ls-files">http://man.yolinux.com/bin/man2html?cgi_command=ls-files</a> )
git <b>ls-remote</b> -h <i>http://domain/git/repo/projectx.git</i> HEAD	List files under Git control on remote Git server.	git ls-remote ( <a href="http://man.yolinux.com/bin/man2html?cgi_command=ls-remote">http://man.yolinux.com/bin/man2html?cgi_command=ls-remote</a> )
git <b>merge</b> <i>branch-name</i> git merge --abort	Merge a specified branch into your current branch you are in. If you are in the "master" branch, then "branch-name" will be merged into "master". --abort: After starting a merge, you might want to stop the merge and return everything to its pre-merge state. --stat: Show a diffstat after merge completion -m: Message to be included in the merge commit	git merge ( <a href="http://man.yolinux.com/bin/man2html?cgi_command=merge">http://man.yolinux.com/bin/man2html?cgi_command=merge</a> )
git <b>mv</b> <i>file-old-name</i> <i>file-new-name</i>	Rename a file. Moves/renames file.	git mv ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_command=mv">http://man.yolinux.com/cgi-bin/man2html?cgi_command=mv</a> )
git <b>mergetool</b>	View file conflicts in a merge tool. Default is vimdiff ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_command=vimdiff">http://man.yolinux.com/cgi-bin/man2html?cgi_command=vimdiff</a> ) Assign a GUI tool: KDiff3: • git config --global merge.tool kdiff3 ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_command=kdiff3">http://man.yolinux.com/cgi-bin/man2html?cgi_command=kdiff3</a> ) • git config --global mergetool.kdiff3.cmd 'kdiff3 \$LOCAL \$BASE \$REMOTE -o \$MERGED'  Meld: • git config --global merge.tool meld ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_command=meld">http://man.yolinux.com/cgi-bin/man2html?cgi_command=meld</a> ) • git config --global mergetool.meld.cmd 'meld --diff \$LOCAL \$BASE \$REMOTE --output \$MERGED' or: meld \$LOCAL \$MERGED \$REMOTE --output \$MERGED or: meld \$LOCAL \$BASE \$REMOTE --output \$MERGED (preference of three panes ordered left to right)  Tools options include: p4merge, meld, kdiff3, tkdiff, gvimdiff, vimdiff, ...	git mergetool ( <a href="http://man.yolinux.com/bin/man2html?cgi_command=mergetool">http://man.yolinux.com/bin/man2html?cgi_command=mergetool</a> )
git <b>pull</b> origin master	Update your local repo and working files with changes posted on the git remote server origin: refers to repository from which you will receive updates master: refers to the master branch from which you will receive updates This command is the same as performing the following two commands: git fetch git merge	git pull ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_command=diff">http://man.yolinux.com/cgi-bin/man2html?cgi_command=diff</a> )

Command	Description	Man Page
<code>git push origin master</code>	Update the remote repo from the staged (identified with "git add") local working files origin: refers to repository from which you will receive updates master: refers to the master branch from which you will receive updates This command is the same as performing the following two commands: <code>git fetch</code> <code>git merge</code>	<code>git pull</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit_diff">http://man.yolinux.com/cgi-bin/man2html?cgi_commit_diff</a> )
<code>git remote -v</code>	Show URL to origin server.	<code>git remote</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit_remote">http://man.yolinux.com/cgi-bin/man2html?cgi_commit_remote</a> )
<code>git reset HEAD^</code> <code>path/filename</code> <code>git reset HEAD~1</code> <code>path/filename</code> <code>git reset --hard HEAD^</code> <code>path/filename</code>	Remove from staging (local copy of files still modified) --hard: Don't keep changes, remove modifications to files. HEAD^: latest commit notation for Linux and bash shell terminals. HEAD~1: latest commit notation for Microsoft CMD shell. Microsoft DOS shell does not support "^".  Reset the repo to the last good commit specified and ignoring everything after e3b43d63: <code>git reset --hard e3b43d63</code> <code>git push --force</code> --hard : will erase your local work if you have anything stashed. --mixed : Unstaging all changes but leave them in the working directory (default). --soft : staged and working directory are not altered	<code>git reset</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit_reset">http://man.yolinux.com/cgi-bin/man2html?cgi_commit_reset</a> )
<code>git revert HEAD</code>	Not a commit undo but overwrites the commit with a new commit to reverse changes. Usually better to use "git reset --hard HEAD^ path/filename" when you want to abort your changes to a file. If you are trying to get back to a previously committed state, just use <code>git checkout e3b43d63</code>	<code>git revert</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit_reset">http://man.yolinux.com/cgi-bin/man2html?cgi_commit_reset</a> )
<code>git rm filename</code> <code>git rm -r directory/path</code>	Delete file from local working index. Don't use the UNIX command <code>rm file-name</code> . Must perform a "git commit" to update the repository. -n: dry run -r: recursive removal when given a directory name.	<code>git rm</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit_rm">http://man.yolinux.com/cgi-bin/man2html?cgi_commit_rm</a> )
<code>git status</code>	Show tracked files which have been staged, committed, etc.	<code>git status</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit_status">http://man.yolinux.com/cgi-bin/man2html?cgi_commit_status</a> )
<code>git tag -a April_Sprint -m "Source used for April sprint demo"</code> <code>git tag -a April_Sprint -m "Source used for April sprint demo" e3b43d63</code>	Add a "tag" to the commit to identify it in human readable terms. You can then push using the following: <code>git push origin master April_Sprint</code> or <code>git push origin master --tags</code> -a: Generate an annotated tag object. -d: Delete existing tags with the given names. -f: Replace an existing tag with the given name. -m: Add a message.	<code>git tag</code> ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_commit_tag">http://man.yolinux.com/cgi-bin/man2html?cgi_commit_tag</a> )

Where *Commit* is:

- HEAD: The latest revision in the repository.
  - HEAD^: latest commit notation for Linux and bash shell terminals.
  - HEAD~1: latest commit notation for Microsoft CMD shell. Microsoft DOS shell does not support "^".
- BASE: The "pristine" revision of an item in a working copy. Matches checked out version before any modifications.
- COMMITTED: The last revision in which an item changed before (or at) BASE.
- PREV: The revision just before the last revision in which an item changed. (Technically, COMMITTED - 1.)

**Example Session:**

(Assumes that the repository has already been created. For Git repository creation and Git server configuration, see the YoLinux Git server tutorial (GIT-Server-Configuration.html))

- Checkout: `git clone ssh://user@git.megacorp.com:/srv/git/repo/projectx.git`  
copy origin repository and create a local repository.
- Go to source code directory: `cd Projectx/src/main/java/com/megacorp/projectx`
- Edit files:
  - `vi file1.java`
  - `vi file2.java`
- Verify and test: `ant`  
We are ready to check-in the files into the Git repository.
- Flag files as staged: `git add file1.java file2.java`
- Show status including files staged: `git status`
- After many long hours of editing and work, get updates others have made and merge changes with local working files: `git pull origin master`
- Resolve merge conflicts if they exist: `git mergetool`  
At this point, a commit will fail until the merge is resolved.
- Merge options:
  - Decide on changes you want: `git mergetool`  
Uses merge tool you specify. Default: `vimdiff`
  - Choose your file changes: `git checkout --ours file1.java`  
(--ours: your file in your local work area)
  - Choose their file changes: `git checkout --theirs file1.java`  
(--theirs: branch you are merging into. The file from remote origin server)
  - Edit the file to remove conflicts: `vim file1.java`  
(the file will have >>>>>> and <<<<<< entries in it)
  - Throw out your changes/abort: `git revert HEAD^`  
No resolve or check-in necessary if file is reverted.
- Commit to your local copy of repo: `git commit -m "Commit message goes here"`
- Verify and test, again: `ant`

**Ignore files with .gitignore:**

There are many files and directories one would like git to ignore and never check-in to the repository. When programming C/C++ one may never want to check-in object (.o) files. Java programmers may never want to check-in ".class" files. IDE's like Netbeans will generate personal configuration settings that one would not want to check-in as they would always conflict with other developers. Git can be configured to ignore these files by generating the file .gitignore in the top directory of your repository.

Example `src/projectx.git/.gitignore`

```
*.a
*.o
*.so
*.dll
*.lib
a.out
*.jar
*.class
```

Add the following for Netbeans developers:

```
private.*
/**/nbproject/private/
```

**Configure Git for use with a proxy:**

- `git --global http.proxy http://proxy.megacorp.com:80`
- or
- `export HTTP_PROXY=proxy.megacorp.com:80`

Test: `config --get http.proxy`

**GIT Properties:**

Since the dawn of source code management, change management (CM) systems have supported the notion of enforcing "properties" such as filtering carriage returns (^M), permissions, etc.

**Repo configuration for end of line (eol) style, etc:**

This is especially important for cross platform development as Microsoft development tools will often append a line with CRLF while non-Microsoft tool will typically terminate a line with only a LF. Files edited with Microsoft tools

will have a CR which will be displayed as a "^M" with other tools. This may generate unnecessary merge conflicts, ridiculous git diff results, unnecessary check-ins due to erroneous git status results and developer hostilities.

The line termination is defined for the entire Git repository by `core.autocrlf`

This can be set on the server repo:

```
cd /srv/git/projectx.git/
git config core.autocrlf false
```

This will set the server configuration file `/srv/git/projectx.git/config`

```
[core]
    repositoryformatversion = 0
    filemode = true
    bare = true
    sharedrepository = 1
    autocrlf = false
[receive]
    denyNonFastforwards = true
...
..
```

or by command:

- `git config core.sharedrepository 1`  
makes everything group readable and writable
- `git config receive.denyNonFastforwards true`  
makes sure that merges can't happen when you push to the repo. You have to do the merges on your local machine, and then push the result.

See the YoLinux.com GIT Server configuration tutorial ([GIT-Server-Configuration.html](http://www.yolinux.com/TUTORIALS/GIT-Server-Configuration.html)) for more on setting up a Git server.

The line termination rules can be overridden using the `.gitattributes` file (check-in to your repo in the top directory):

```
# Auto detect text files and perform LF normalization
*      text=auto      eol=lf

*.cpp   text diff=cpp      eol=lf
*.cs    text diff=csharp   eol=lf
*.vcproj text diff=java    eol=crlf
*.java  text diff=java     eol=lf
*.html  text diff=html     eol=lf
*.css   text              eol=lf
*.js    text              eol=lf
*.php   text diff=php      eol=lf
*.module text diff=php     eol=lf
*.engine text diff=php     eol=lf
*.inc   text diff=php      eol=lf
*.sql   text              eol=lf
*.jpg   binary diff=exif   eol=lf
*.gif   binary diff=exif   eol=lf
*.png   binary diff=exif   eol=lf

*.csproj text merge=union
*.sln    text merge=union eol=crlf

*.docx   diff=word

# relative paths are treated relative to the .gitattributes folder
relative/path/*.txt text eol=lf
```

- This configuration "auto detects" text files and also defines C++ files (\*.cpp), Java source (\*.java), etc as text files.
- Visual Studio solution files (\*.sln) maintain their Microsoft carriage return (cr) followed by a line feed (lf).
- diff: specifies the git filter to use when viewing a diff that contains changes.
  - exif: Converts binary to a textual representation using `exiftool` ([http://man.yolinux.com/cgi-bin/man2html?cgi\\_command=exiftool](http://man.yolinux.com/cgi-bin/man2html?cgi_command=exiftool)) to detect if changes have occurred.  
Configure repo: `git config diff.exif.textconv exiftool`
- eol: End Of Line marked by:
  - lf: Unix, Linux, standards based OS, proprietary legacy systems, etc.
  - crlf: DOS and Microsoft OSs
  - cr: Response input scripts, etc
- merge: Use the option "merge=union" when the solution to merge conflicts are to take lines from both versions, instead of leaving conflict chevron markers.

One can also specify "-merge" (the opposite, don't merge conflicted lines).

To "normalize" Java source files to LF terminated lines: `find ./ -name "*.java" -exec dos2unix`  
([http://man.yolinux.com/cgi-bin/man2html?cgi\\_command=dos2unix](http://man.yolinux.com/cgi-bin/man2html?cgi_command=dos2unix)) {} \;

### Executable files:

Files may be marked as having execute permissions by doing the following:

```
git add script.sh
git update-index --chmod=+x script.sh
git commit -m "Add script"
```

Note that an improper `core.fileMode` configuration can change the utility of this action. When set to false, the executable bit differences between the index and the working copy are ignored. Conversely, when set to true, file mode permission changes are considered changes.

### GIT Tags:

CM systems such as RCS, CVS and Subversion have all supported tag substitutions and so can Git. Keywords are substituted and will not appear in the file until a new revision of the file is updated from the repository.

The following RCS attribute tags can be set on files stored in GIT. The keyword expansion of the following tags are substituted by a "smudge" filter upon a pull and a "clean" filter on a push. Download the smudge and clean filters (<https://github.com/turon/git-rcs-keywords>).

The following are the supported tags which can be embedded into your source code:

RCS Tag	Expanded String
<b>\$Date\$</b>	The date as provided by the shell command "date ( <a href="http://man.yolinux.com/cgi-bin/man2html?cgi_command=date">http://man.yolinux.com/cgi-bin/man2html?cgi_command=date</a> )" \$Date: Sun Jul 30 17:21:43 2017 -0700 \$
<b>\$Author\$</b>	Will provide the author's name as specified by <code>~/ .gitconfig</code> from the commands: <ul style="list-style-type: none"> <li><code>git config --global user.name "John Doe"</code></li> <li><code>git config --global user.email john.doe@megacorp.com</code></li> </ul> \$Author: John Doe <john.doe@megacorp.com> \$
<b>\$Revision\$</b>	This fills in the Git hash for the commit associated with this revision. \$Revision: 93c4a4a5b871f3183ea90f7df159cc3e363ec534 \$
<b>\$Source\$</b>	File name
<b>\$File\$</b>	File name (same as \$Source\$)
<b>\$Id\$</b>	A summary of the previous 3 keywords: \$File, \$Date and \$Author.

Example source file comment:

```
/**
 * $Revision$
 * $Author$
 * $Date$
 * $Source$
 * $File$
 * $Id$
 */
```

Expanded comment:

```
/**
 * $Revision: 93c4a4a5b871f3183ea90f7df159cc3e363ec534 $
 * $Author: John Doe <john.doe@megacorp.com> $
 * $Date: Sun Jul 30 17:21:43 2017 -0700 $
 * $Source: filename.java $
 * $File: filename.java $
 * $Id: filename.java | Sun Jul 30 17:21:43 2017 -0700 | John Doe $
 */
```

The personal configuration file `~/ .gitconfig` is where the filters are assigned. Add the following:

```
[filter "rcs-keywords"]
    clean = .git_filters/rcs-keywords.clean
    smudge = .git_filters/rcs-keywords.smudge %f
```

Place the `.gitattributes` file and `.git_filters/` directory at the root of the CM directory tree:

File: `projectx/.gitattributes`

```
*.h      filter=rsc-keywords
*.c      filter=rsc-keywords
*.cc     filter=rsc-keywords
*.cpp    filter=rsc-keywords
*.m      filter=rsc-keywords
*.mm     filter=rsc-keywords
*.java   filter=rsc-keywords
```

Copy the filters to:

- `projectx/.git_filters/rsc-keywords.smudge` ([https://github.com/turon/git-rsc-keywords/blob/master/.git\\_filters/rsc-keywords.smudge](https://github.com/turon/git-rsc-keywords/blob/master/.git_filters/rsc-keywords.smudge))
- `projectx/.git_filters/rsc-keywords.clean` ([https://github.com/turon/git-rsc-keywords/blob/master/.git\\_filters/rsc-keywords.clean](https://github.com/turon/git-rsc-keywords/blob/master/.git_filters/rsc-keywords.clean))

Allow filters to be executed: `chmod ugo+x projectx/.git_filters/rsc-keywords.smudge`  
`projectx/.git_filters/rsc-keywords.clean`

Otherwise you will get the following error:

```
fatal: cannot exec '.git_filters/rsc-keywords.clean': Permission denied
error: cannot fork to run external filter .git_filters/rsc-keywords.clean
error: external filter .git_filters/rsc-keywords.clean failed
```

The filters are checked-in to Git and thus need to have a Git execute attribute applied: `git update-index --chmod=+x projectx/.git_filters/rsc-keywords.smudge projectx/.git_filters/rsc-keywords.clean`

Note that this tag substitution occurs on a fresh "git clone" or on a "git pull" and modifies the files in your workspace but does not affect the files in the repository itself. The author of the RCS tag edits will not see the substitutions until they perform a fresh "git clone" to generate a new local repository or when a "git pull" finds file modifications and performs an update.

## Graphical diffs and merges for Linux:

Three types of file differences are covered in this section:

1. Show file differences made since a pull or checkout was made. This shows the changes you have made. It is useful to perform this before a commit.
2. Show file differences between two files versions stored in Git.
3. Show differences / conflicts, choose and merge.

Note that the following conventions are used:

- LOCAL: file on your current local branch
- REMOTE: branch you are merging to (e.g. `new_branch` you are trying to pull. Their changes.)
- MERGED: the partially merged file with merge conflicts marked with chevrons:
  - `<<<< HEAD : conflicted lines between this and "====" are your changes`
  - `>>>> 7c10716976ee3e9d49f69d71ec6cc4bd43957ec8 : conflicted lines between this and "====" are their changes in the repository which conflict.`

You have the option of using a merge tool or just editing the file to reflect the changes you want. The chevrons and "====" must be removed.
- BASE: the original common ancestor file from which LOCAL and REMOTE evolved. The file before the conflicted changes.

## 1) File differences since last pull:

Git uses the command line Linux "diff" ([http://man.yolinux.com/cgi-bin/man2html?cgi\\_command=diff](http://man.yolinux.com/cgi-bin/man2html?cgi_command=diff)) command. A GUI diff tool is far more intuitive and can be specified using the following commands:

- Configure git to use a GUI diff tool:
  - `git config --global diff.tool meld`
  - `git config --global difftool.meld.cmd 'meld $LOCAL $REMOTE'`

This adds the following entry to your `~/.gitconfig`

```
[diff]
    tool = meld
[difftool "meld"]
    cmd = meld $LOCAL $REMOTE
```

- GUI diff between file in working area and repo: `git difftool` ([http://man.yolinux.com/cgi-bin/man2html?cgi\\_command=git-difftool](http://man.yolinux.com/cgi-bin/man2html?cgi_command=git-difftool)) `filename.java`

```
Viewing (1/1): 'filename.java'
Launch 'meld' [Y/n]: y
```



## 2) File differences between two revisions:

- View the differences for a file and two commits back: `git difftool HEAD HEAD^^ filename.java`
- View the differences for a file between two commits: `git difftool a1a1c14299049c5e1cf60f8cc02f41bbf7d8b85f 93c4a4a5b871f3183ea90f7df159cc3e363ec534 filename.java`  
(Use "git log" to view the commit hashes)
- View the differences for two files between two commits: `git difftool a1a1c14299049c5e1cf60f8cc02f41bbf7d8b85f:file1.java 93c4a4a5b871f3183ea90f7df159cc3e363ec534:file2.java`
- View all differences between two commits: `git difftool a1a1c14299049c5e1cf60f8cc02f41bbf7d8b85f 93c4a4a5b871f3183ea90f7df159cc3e363ec534`

```
Viewing (1/3): 'file_1.java'
Launch 'meld' [Y/n]: y

Viewing (2/3): 'file_2.java'
Launch 'meld' [Y/n]: y

Viewing (3/3): 'file_3.java'
Launch 'meld' [Y/n]: y
```

## 3) Conflicts and merges:

Configure git to use a GUI merge tool. Note that most diff tools also support merges:

- `git config --global merge.tool meld`
- `git config --global mergetool.meld.cmd 'meld $LOCAL $MERGED $REMOTE'`

This adds the following entry to your ~/.gitconfig

```
[merge]
    tool = meld
[mergetool "meld"]
    cmd = meld $BASE $LOCAL $REMOTE --output $MERGED
```

Merge conflicts occur when two developers check out common files and both edit the same line/lines. Edits to different lines merge automatically and cleanly. When changes are made to the same line and then one developer commits and pushes the code, the second developer must deal with the merge conflicts by selecting the line they want, "ours" or "theirs". GUI merge tools support the management of a merge by addressing the lines marked as a conflict using the text markers ">>>".

## List of graphical diff and merge tools:

- tkdiff (<http://tkdiff.sourceforge.net/>): [download (<http://sourceforge.net/projects/tkdiff/>)]
- gtdiff: Has diff3 and merge features. Written with GTK+. After gtdiff-0.8.0, GNOME desktop required.
- diffUse (<http://sourceforge.net/projects/diffuse/>): Diff/merge GUI tool. Good line matching features. Supports Unicode.
- kdiff3 (<http://kdiff3.sourceforge.net/>): Graphical directory and file diff, merge and edit. KDE3/Qt based. (Cross platform) MS/Windows download available. A very good directory and file diff and merge tool.

o Diff:

- `git config --global diff.tool kdiff3`
  - `git config --global difftool.kdiff3.cmd 'kdiff3 $LOCAL $REMOTE'`
- This adds the following entry to your ~/.gitconfig

```
[diff]
    tool = kdiff3
[difftool "kdiff3"]
    cmd = kdiff3 $LOCAL $REMOTE
```

o Merge:

- `git config --global merge.tool kdiff3`
- `git config --global mergetool.kdiff3.cmd 'kdiff3 $BASE $LOCAL $REMOTE -o $MERGED'`

This adds the following entry to your ~/.gitconfig

```
[merge]
    tool = kdiff3
[mergetool "kdiff3"]
    cmd = kdiff3 $LOCAL $BASE $REMOTE -o $MERGED
```

- Kompare (<http://www.caffeinated.me.uk/kompare/>): Ships with (RHEL6/RHEL5/RHEL4/FC3+) KDE SDK. [manual (<http://docs.kde.org/development/en/kdesdk/kompare/>)] Included in Red Hat RPM package "kdesdk" an Ubuntu package "kompare".
- [Potential Pitfall]: RPM installation error:

```
error: Failed dependencies:
    perl(DCOP) is needed by kdesdk-3.5.4-3.el5.i386
```

Solution: Install the dependency package "kdebindings".

- mgdiff: Motif-based graphical file difference browser and merge. Comes with S.u.S.E. distro.
- Meld (<http://meld.sourceforge.net/>): Compare, edit and merge.
  - Diff:
    - `git config --global diff.tool meld`
    - `git config --global difftool.meld.cmd 'meld $LOCAL $REMOTE'`
 This adds the following entry to your ~/.gitconfig

```
[diff]
    tool = meld
[difftool "meld"]
    cmd = meld $LOCAL $REMOTE
```

- Merge:
  - `git config --global merge.tool meld`
  - `git config --global mergetool.meld.cmd 'meld $BASE $LOCAL $REMOTE --output $MERGED'`
 This adds the following entry to your ~/.gitconfig

```
[merge]
    tool = meld
[mergetool "meld"]
    cmd = meld $LOCAL $BASE $REMOTE --output $MERGED
```

- xxdiff (<http://furius.ca/xxdiff/>): Compare 2 or 3 files and merge. Also compares directories.
- gvim and gvimdiff (<http://www.vim.org/>)
- Beyond Compare (<http://www.scootersoftware.com/>) - commercial tool (cross platform)

### GIT GUI interfaces for Linux:

Git comes with a GUI: `git gui` and a browser tool `gitk`

The latest version is available from github.com: gitk (<https://github.com/gitster/git/tree/master/gitk-git>)

Installation:

- Ubuntu: `sudo apt-get install git-gui`
- From source:
  - `wget https://github.com/gitster/git/blob/master/gitk-git/gitk`
  - `chmod +x gitk`

For other options see:

- CollabNet: GitEye (<https://www.collab.net/products/giteye/>): cross platform, push/pull, history, diffs, revert, stash, etc. Intuitive and customizable
- SmartGit (<http://www.syntevo.com/smartgit/>): cross platform commercial product
- QGit (<https://sourceforge.net/projects/qgit/>): View revisions, diffs, files history, files annotation, archive tree, Commit changes visually cherry picking modified files.
- Axosoft: GitKraken (<https://www.gitkraken.com/>): commercial product. Free for non-commercial use. Cross platform.
- Git-cola (<https://git-cola.github.io/>): Python git front-end. (cross platform)
- GNOME: Gigggle (<https://wiki.gnome.org/Apps/Gigggle/>): GTK+ application
- GNOME: gitg (<https://wiki.gnome.org/Apps/Gitg/>)

### Web Clients:

- GitWeb ([http://man.yolinux.com/cgi-bin/man2html?cgi\\_command=gitweb](http://man.yolinux.com/cgi-bin/man2html?cgi_command=gitweb)): comes with Git
- GitLab (<https://gitlab.com/gitlab-org/gitlab-ce>): Ruby, community edition
- Gogs (<https://gogs.io/>): written in Go
- Gitea (<https://github.com/go-gitea/gitea>): a fork of Gogs
- RhodeCode (<https://rhodecode.com/>): community edition and commercial product. User control, permissions, code reviews, and tool integration.
- Gitssh2 (<http://www.gitssh2.com/>): Symfony PHP framework. Connects to repositories via ssh
- Axosoft: GitKraken (<https://www.gitkraken.com/>):
- GitList (<http://gitlist.org/>):
- Kallithea (<https://kallithea-scm.org/>): community edition

### Plug-ins:

- nautilus-git (<https://github.com/bil-eloussaoui/nautilus-git>): GNOME Nautilus/Nemo file browser GIT plug-in.
- EGit (<http://www.eclipse.org/egit/>) Eclipse IDE plug-in

Note: A GIT client is built into many of today's development IDEs such as NetBeans.

### Git Best Practices:

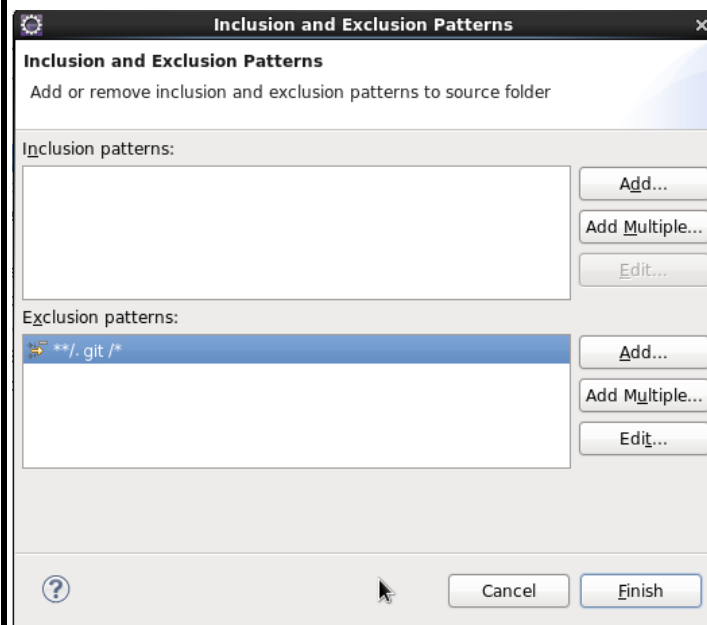
- ALWAYS compile and test before pushing source code to the origin server. Developers expect shared changes to compile.
- Always add commit comments. Nothing is more obnoxious than looking at an uninformative log or information history.
- Monstrous binary files like CD or DVD ISO images are better suited for storage on a file system rather than in Git. Git will not be able to show differences between version and will be slower than a raw filesystem.
- Don't copy, rename and move directories and files with system shell commands. Use Git commands to "rm", "mv", and "add" files and then commit changes when done. Work within Git.
- Push changes as a single logical change-set for one purpose. Thus all code changes for a single bug fix or enhancement should be checked-in together. This allows one to better follow the history log of changes.
  - Stage code at the directory level and all changed files, recursively in the directory and sub-directories will be checked in together.  
`git add -all`
  - Stage files together by specifying them explicitly:  
`git add file.cpp file2.cpp ...`
- Tie Bug tracking and Git CM together:
  - Use comments when committing files into Git. Add bug tracking numbers to the comments so Git will reference Trac bugs.
  - If using Trac, add Trac comments so that links are generated to the Git repository by placing the Git revision hash in square braces (i.e. [e3b43d63]) in a Trac comment. In this way, Trac will have a direct URL link to Gitweb.
- If using Trac integrated with Git, refer to the Trac ticket in the Git commit comment using a "#" in front of the Trac ticket number (e.g. #65) This generates a hyperlink when the Git logs are viewed in Trac.
- Documentation and related artifacts checked-in to Git should not be under the source tree but parallel to it. This isolates the source tree so that email notification triggers sent to developers upon source changes will only go out on source changes or regression build and test systems like Jenkins (Jenkins.html) or Cabie (CabieBuildSystem.html) will only rebuild and test on source changes rather than on unrelated documentation changes.

### Tips:

#### Using Eclipse IDE with Git:

Block Eclipse from viewing the hidden ".git/" directory so that Eclipse does not intrude or alter their contents:

- Select the toolbar options: "Project" + "Properties"
- Select the "Java Build Path" + select the "Source" tab
- Select the "Excluded" item and select "Edit"
- In the lower half under "Exclusion patterns:" select "Add..." and enter the pattern `**/.git/*`



### Links:

- **YoLinux Tutorial: Git Server - Installation and Configuration** (GIT-Server-Configuration.html)
- **YoLinux Tutorial: Git and Trac Server Installation and Configuration** (LinuxGitAndTracServer.html)
- Git Home Page (<https://git-scm.com/>)



### Books:

	Distributed Version Control with Git [Kindle Edition] Lars Vogel (Author) A practical introduction into the Git version control system.	 ( <a href="http://www.amazon.com/gp/redirect.html?ie=UTF8&amp;location=http://www.amazon.com/exec/obidos/ASIN/B0067QNR56/&amp;tag=yolinux-20">http://www.amazon.com/gp/redirect.html?ie=UTF8&amp;location=http://www.amazon.com/exec/obidos/ASIN/B0067QNR56/&amp;tag=yolinux-20</a> )
	Version Control with GIT by Jon Loeliger and Matthew McCullough ISBN #1449316387, O'Reilly Press	 ( <a href="http://www.amazon.com/gp/redirect.html?ie=UTF8&amp;location=http://www.amazon.com/exec/obidos/ASIN/1449316387/&amp;tag=yolinux-20">http://www.amazon.com/gp/redirect.html?ie=UTF8&amp;location=http://www.amazon.com/exec/obidos/ASIN/1449316387/&amp;tag=yolinux-20</a> )
	Pro Git Scott Chacon (Author) Visual explanations of the git concepts and methodology. Step-by-step ways to track, merge, and manage software projects, using Git.	 ( <a href="http://www.amazon.com/gp/redirect.html?ie=UTF8&amp;location=http://www.amazon.com/exec/obidos/ASIN/1430218339/&amp;tag=yolinux-20">http://www.amazon.com/gp/redirect.html?ie=UTF8&amp;location=http://www.amazon.com/exec/obidos/ASIN/1430218339/&amp;tag=yolinux-20</a> )

### Sponsored Links

**The Biggest Idiots In The Tech Industry**

Hotstar Premium

**People from India cannot believe these flight prices**

Flights Shop

**Motilal Oswal | Demat Account | Trading Account | Share Market | Stock Market**

Motilal Demat A/C

**What Virtual You Would Do? | Play Second Life**

Free 3D Virtual World | Second Life

**Massive Reward For Indians Born Between 1941-1981**

Survey Compare

**Know Bengaluru's most promising real estate opportunity**

Bloomberg|Quint

[Comments](#) [Community](#) [Login](#)

[Recommend](#) [Tweet](#) [Share](#) [Sort by Best](#)

LOG IN WITH

[D](#) [f](#) [t](#) [G](#)

OR SIGN UP WITH DISQUS [?](#)


Be the first to comment.

ALSO ON YOLINUX

[Linux Tutorial: Add an additional disk drive to your](#)  
1 comment • 2 years ago


[C/C++ signal handling](#)  
2 comments • 2 years ago


[Ave](#) M Golam Hossain — Nice Solutions, thanks.

 Bilal Awe — According to the man page of signal, I don't believe it's safe to call

[Jenkins for Java: tools and plugin configuration](#)  
1 comment • 2 years ago

[Parsing XML with Xerces-C C++ API](#)  
2 comments • 2 years ago

 Mohammad Belal — Hello I

 Timofte Bogdan — Any hint

[BOOKMARK](#) [http://www.addthis.com/bookmark.php?v=250&pub=yolinux](#)

## Advertisements

- 01 Small Business Bookkeeping Software >
- 02 Data Analytics Training >
- 03 Free Online Certification Courses >
- 04 Create A Web Page >
- 05 How To Create Own Website >
- 06 Create Your Web Site >
- 07 Simple Ways To Make Money >
- 08 Build Your Own Website >

<http://www.yolinux.com>

[YoLinux Tutorial Index](#)

<http://www.yolinux.com/TUTORIALS/> |

[Terms](#) (<http://www.yolinux.com/YoLinux-Terms.html>)

[Privacy Policy](#) (<http://www.yolinux.com>

[/privacy.html](#)) | [Advertise with us](#)

([http://www.yolinux.com/YoLinux-](http://www.yolinux.com/YoLinux-Advertising.html)

[Advertising.html](#)) | [Feedback Form](#)

(<http://www.yolinux.com>

[/YoLinuxEmailForm.html](#)) |

Unauthorized copying or redistribution prohibited.

Copyright © 2017 - 2018 by *Greg Ippolito*



(<http://www.addthis.com>

[/bookmark.php?v=250&pub=yolinux](#))