|  |  |
| --- | --- |
| Git :: Configuration + Help | |
| **git help <command>**  **git <command> --help** | Gets help on specified git command (commands are equivalent). On Windows this will open web-browser with help screen. |
| **git config --list** | Lists all config parameters (name=value list) |
| **git config --global user.name** | Sets up user name for current user (use key **--system** for set up value for all users) |
| **git config --global user.email** | Sets up user email for current user (use key **--system** for set up value for all users) |
| **git config --global core.editor <editor>** | Select text editor for git messages (commits, tags, merges etc) |
| **git config --global credential.helper cache** | Sets up credentials cache. It saves credential info for some time in memory (you don't have to enter credentials on every remote operation). Default timeout is 15 minutes. |
| **git config --global credential.helper “cache --timeout=3600”** | The same as above, but with different timeout. Timeout specified in seconds. |
| **git config --global credential.helper wincred** | The same as above, but works **for Windows** (credentials store, for msysgit 1.8.1 and above) |
| **git config --global alias.<name> <command>** | Create short alias <name> for command <command> (next in your work you can use <name> instead of <command>). Very useful for convenience work process. If <command> contains spaces (composite command) put it in quotes (double quotes on windows). |
| **<Ignoring file(s)>** | 1. Create file .gitignore in working folder and adds there files that should be ignored (not tracked by Git system) 2. Edit $GIT\_DIR/info/exclude and add ignored files there |
| **Git :: Repository creation + init** | |
| **git init** | Initialize repository in current folder (creates new local repository) |
| **git clone <URL> [dest-folder]** | Creates a copy (clone) existing repository located at <URL> in folder with default name (repo name). If you specify **[dest-folder]**, repository will be cloned to that folder. For clone you may use different protocols: **local**, **https://**, **git://**, **ssh**. |

|  |  |
| --- | --- |
| **Git :: Files operations** | |
| **git add <path/file>** | Adds new file/path to Git repository or indexes modified file. You can specify more than one file, separate them by space. Also you may use the following wildcards/patterns:   * \* - any count of any symbols * [abc] – one of abc (a or b or c) * [0-9] – values range (in this example from 0 to 9) * ? - any one symbol * \*\* - nested folders, for example a/\*\*/z means file z nested in folder a and any number of other folders between them |
| **git rm <file>** | Removes **<file>** both from Git tracked files list (indexed area) and from work folder. You can manually delete file from work folder and index file removing operation by executing git command [git rm]. You may use patterns/wildcards with this command. Important options:   * **-f** use this option when you have already edited and indexed a file (force removing) * **--cached** remove file from indexed area, but leave it in working folder (doesn't physically delete file, file becames untracked) |
| **git mv <source> <dest>** | Moves/renames file <source> to <dest>. This command is equivalent to sequence: [**mv <source> <dest>** → **git rm <source>** → **git add <dest>**] (you may manually move a file and then index changes by two git commands) |
| **git checkout -- <file>** | Discard changes in working directory – **you lose your local changes!** |
| **git checkout [revision] <file>** | Restores **<file>** from current branch HEAD (if no [revision] specified) or specified revision **[revision]**. Command overwrites not commited (not indexed) local changes (**you lose local changes!**). |

|  |  |
| --- | --- |
| **Git :: Working tree** | |
| **git status**   |  |  | | --- | --- | | **M** | modified | | **C** | copied and modified | | **R** | renamed and modified | | **A** | added | | **D** | deleted | | **U** | conflicts after merge | | **??** | untracked | | Shows current repository status. Important options are:   * **-v**, **--verbose** be verbose * **-s**, **--short** show status concisely   When you use -s/--short option git uses flags for files' statuses – see table on the left.  Command **git status** outputs this flags in two columns: [**MM**] or [**<space>M**] etc (using other flags). Flag in a first column means: changes are indexed (ready for commit and **will be included** in commit); flag in a second column means: changes are not indexed (not ready for commit and **won't be included** in commit). |
| **git diff [--stat] [path/file]**  *(show not indexed)* | Show **all** or specified **[path/file]** differences between HEAD and work folder – shows **not indexed** (and **not committed**) changes. Option **--stat** used for short output (summary). |
| **git diff --cached | --staged [--stat] [path/file]**  *(show indexed, not committed)* | Show **all** or specified **[path/file]** differences between HEAD and indexed area – shows **indexed** but **not committed** changes (changes that are ready for commit). Option **--stat** used for short output (summary). |
| **git diff HEAD [--stat] [path/file]**  *(show all)* | Show **all** or specified **[path/file]** differences between **not indexed** and **committed** changes for HEAD (show all changes from previous commit). Option **--stat** used for short output (summary). |
| **git diff <source-rev> <dest-rev>**  **[--stat] [path/file]** | Show **all** or specified **[path/file]** differences between **committed** changes in direction from **<source-rev>** to **<dest-rev>**. If you omit **<dest-rev>** git will use HEAD. Option **--stat** used for short output (summary). As **<source/dev-rev>** you may use branch name. |
| **git add <path/file>** | Stage path or file for the commit (indexing). |
| **git reset HEAD <path/file>** | Cancel indexing for path/file (unstage path/file, remove it from indexed area). |
| **git reset --soft HEAD**  **git reset --hard HEAD** | ????? |
| **git commit** | Commits to local repository all indexed (with [git add] command) changes. Important options are:   * **-m** specify commit message (in quotes). If you commit without this option git will open text editor for entering commit message * **-a** auto stage (indexing) **all** tracked and modified files before commit (don't need to explicitly call [git add] for indexing tracked files) * **--amend** replaces last commit for a new one (you may add forgotten file etc.) * **--dry-run** do not create a commit, but show a list of paths that are to be committed, paths with local changes that will be left uncommitted and paths that are untracked |
| **git clean** | ????? |
| **git revert <rev>** | ????? |

|  |  |
| --- | --- |
| **Git :: History (log) + browsing** | |
| **git log [path/file]**  *values for <format> option in* ***--pretty=format:''<format>''***   |  |  | | --- | --- | | %H | full hash for commit | | %h | short hash code for commit | | %T | tree hash code | | %t | tree short hash | | %P | parent commits hash | | %p | parent commits short hash | | %an | author name | | %ae | author's email | | %ad | original create date (consider **-date=** option) | | %ar | original create date, relative | | %cn | version creator's name | | %ce | version creator's email | | %cd | version creation date | | %cr | version creation date, relative | | %s | commit comment | | Show commit log (full log or for specified [path/file]) in a descending order (the last commit first). Some important/useful options are:   * **-p** show differences for every commit * **-(n)**  show the specified number → (n) of last commits * **--since**, **--after** show only commits after specified date * **--until**, **--before** show only commits before specified date * **--author** show commits for specified author * **--committer** show commits for specified committer * **--grep** show commits with commit message containing specified string * **-S<string>**  show commits where specified <string> was deleted or inserted * **--no-merges** show commits without merges * **--all-match** turn multiple parameters (see above) in state AND, default state is OR * **--stat** show short statistics for every commit * **--shortstat** show line with only changes/insertions/deletions from **--stat** option * **--name-only** show changed files names list after commit info * **--name-status** show changed files list with info about operation add/change/delete * **--abbrev-commit** show only some first (about 6) symbols for SHA-1 sum, instead all 40 * **--relative-date** show date in relative format(for example: “2 weeks ago”) * **--graph** add ASCII graph to history info * **--oneline** show log in one line for every commit * **--decorate** show branches and HEAD pointers * **--all** show all commits history * **--pretty=<value>** change output format (**oneline**, **short**, **full**, **fuller**). * **--pretty=format:''<format>''** set specified output format. Format parameters see on the left side (blue table). |
| **git log --pretty=format:”%h - %an, %ar : %s”** – pretty formatted log output | |
| **git log --pretty=format:”%h %s” --graph** – pretty formatted output with ASCII graph | |
| **git log --oneline --decorate**  – show one line log with branches pointers | |
| **git log --oneline --decorate --graph --all** – show commits history with branches pointers and graph | |
| **[alias]**  **lg1 = log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)' –all**  **lg2 = log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold cyan)%aD%C(reset) %C(bold green)(%ar)%C(reset)%C(bold yellow)%d%C(reset)%n''%C(white)%s%C(reset) %C(dim white)- %an%C(reset)' --all**  **lg = !"git lg1"** | |
| **gitk** | The Git repository browser (graph browser). Important options:   * **--all** show all refs (branches, tags, etc.) * **--branches[=pattern]**, **--tags[=pattern]**, **--remotes[pattern]** – pretends as if all the branches (tags, remote branches, etc) are listed on the command line as <commit>. If <pattern> is given, limit refs to ones matching given shell glob. If pattern lacks ?, \*, or [, /\* at the end is implied. |
| **git blame <file>** | Show what revision and author last modified each line of a <file>. Important options are:   * **--show-stat** include additional statistics at the end of blame output * **--reverse** walk history forward instead of backward |

|  |  |
| --- | --- |
| **Git :: Remote repositories** | |
| **git remote** | Shows all existing remote repositories (links). Some important options are:   * **-v** show URLs for remote aliases * **-vv** ????? |
| **git remote add <alias> <url>** | Adds pointer <alias> to remote repository <url> in list of tracked repositories |
| **git remote rename <old> <new>** | Rename alias <old> to alias <new>. Alias content (URL) doesn't affected. |
| **git remote show <alias>** | Show full info about remote repository <alias>. |
| **git remote rm <alias>** | Remove link (alias) to remote repository <alias>. |
| **git fetch [alias]** | Fetch data from remote repository (default or named by [alias]) and updates all remote branches pointers. **Command doesn't merge fetched data! Don't forget to merge if you need it!** |
| **git pull [alias]** | Pull data from remote repository (default or named by [alias]) and merge with local repository current branch ([git pull] = [git fetch] + [git merge]). **Be careful: command automatically merges fetched remote data!** |
| **git push [alias] [branch name]** | Push committed data from local repository (all branches or concrete [branch name]) to remote repository (default or named by [alias]). |
| **git push -u [alias] [branch name]** |  |
| **git push [alias] <tag>** | Push tag to remote repository (default or named by [alias]). |
| **git push [alias] --tags** | Push all tags to remote repository (default or named by [alias]). |

|  |  |
| --- | --- |
| **Git :: Branches and Tags** | |
| **git branch <name>** | Create new branch <name> (**just create new pointer, doesn't switch HEAD!**). For switching HEAD use **git checkout <name>** - see below. |
| **git branch -v** | Shows the latest commit in every branch |
| **git branch -d <name>** | Deletes branch <name>. If this branch contains unique (not merged in any other branch) data, git doesn't allow it. Use option -D instead. |
| **git branch --merged** | Shows all branches, that **were** merged with current branch |
| **git branch --no-merged** | Shows all branches, that **were not** merged with current branch |
|  |  |
|  |  |
|  |  |
| **git checkout [-b] <name>** | Switch HEAD to branch <name>. **This command changes content of working folder.** If you specify option **-b** command create branch <name> and switch to it (it's short for [**git branch**] + [**git checkout**]) |
|  |  |
| **git tag** | Show all existing tags |
| **git tag <tag>** | Create **lightweight** tag for last commit in HEAD branch. |
| **git tag -a <tag> [-m <comment>] [revision]** | Create **annotated** tag named <tag>. Optionally (if you specify option **-m**) add message <comment> to tag. If you omit message <comment> but add option **-m** git opens text editor for adding tag message. If you specify **[revision]** (first 6-8 symbols of commit sha-1 sum) tag will be created for specified commit. |
| **git show <tag>** | Show commit data and (optional) tag data for tag <tag>. Only commit data is shown for **lightweight** tags, full data (commit and tag) is shown for **annotated** tags. |
| **git merge** |  |
| **git mergetool** |  |
| **git push <origin> --delete <branchname>** | Remove remote branch. |
|  |  |