**git config**

**GIT Cheat Sheet**

|  |
| --- |
| git config --system |
| Shows configurations for all users of the pc |
| git config -- global |
| Shows configurations for current user of the pc |
| git config --local |
| Shows configurations for the current repository |
| git config --list --show-origin |
| Lists all options, each with its origin file |
| git config --show-origin user.name |
| Shows the file which has the final say of a specific configuration |
| git config --global user.name "John Doe"  git config --global user.email johndoe@example.com |
| Identity shall be set once after installation |
| git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -nosession" |
| Set Editor to notepad++ ( used by git when needs you to type message) |

* difftool: You can use git config command to set diff tool, or go directly to your global .gitconfig file and write the following

[diff]

tool = bc4

[difftool]

prompt = false

[difftool "bc4"]

#use cygpath to transform cygwin path $LOCAL (something like /tmp/U5VvP1\_abc) to windows path, because bc3 is a windows software

cmd = \"C:/Program Files/Beyond Compare 4/BCompare.exe\" "$(cygpath -w $LOCAL)" "$REMOTE"

* mergetool: You can use git config command to set merge tool, or go directly to your global .gitconfig file and write the following

[merge]

tool = bc4

[mergetool]

prompt = false

[mergetool "bc4"]

#trustExitCode = true

cmd = \"C:/Program Files/Beyond Compare 4/BCompare.exe\" "$LOCAL" "$REMOTE" "$BASE" "$MERGED"

**git help**

|  |
| --- |
| git <cmd> --help |
| Shows the detailed manual of the cmd |
| git <cmd> -h |
| Shows a short description of the cmd |

* Pro GIT:
  + Recommended text book
* Google:
  + Many concepts/problems/cases can be found illustrated onto the web.

**Get a Repo**

|  |
| --- |
| git init |
| Creates .git folder, just a skeleton, no commits yet. |
| git clone <https\_or\_ssh\_link> |
| Creates a directory, creates .git folder and initialize a new repo, pulls all revisions from the remote repo to the local repo and checkout the latest version to the working copy |
| git clone <https\_or\_ssh\_link> <name> |
| Same as git clone, the created directory will take the specified name. |

**Staging/adding**

|  |
| --- |
| git add <file name> |
| Moves this file (modified or untracked) from working tree to the staging area. Later, when committing, only files moved to staging area will be committed to the local repo. |

**Committing**

|  |
| --- |
| git commit –m “msg” |
| Commits whatever in staging area to local repo. |
| git commit –amend |
| Replaces the last commit with a new one |

**git status**

|  |
| --- |
| git status |
| A very useful command. Shows the untracked files (if not ignored), modified files. Number of insertions/deletions for each. And gives hints. E.g: how to stage/unstage/commit files. |

**git ignore**

* .ignore file, contains glob expressions for files that you do not want to track, or even to show as untracked files. Example:

# ignore all .a files

**\*.a**

# but do track lib.a, even though you're ignoring .a files above

**!lib.a**

# only ignore the TODO file in the current directory, not subdir/TODO

**/TODO**

# ignore all files in any directory named build

**build/**

# ignore doc/notes.txt, but not doc/server/arch.txt

**doc/\*.txt**

# ignore all .pdf files in the doc/ directory and any of its subdirectories

**doc/\*\*/\*.pdf**

**git diff**

|  |
| --- |
| git diff |
| Shows unstaged modifications |
| git diff –cached | git diff --staged |
| Shows staged modifications |
| git difftool |
| Shows unstaged modification in diff tool |
| git difftool –cached | git difftool --staged |
| Shows staged modifications in diff tool |

**Remove**

|  |
| --- |
| git rm <filename> |
| Remove the file from git index and working copy. |
| git rm –cached <file name> |
| Remove the file from git index, keep it in working copy |
| git rm –f <file name> |
| Remove the file from index and working copy, even if there is staged changes that is not committed, and will be lost forever.  Note: better to first unstage the staged change explicitly. In general –f options shall be used with caution. |

**Renaming/moving**

|  |
| --- |
| git mv <old\_path\_name> <new\_path\_name> |
| Rename/Move a file |
| mv <old\_path\_name> <new\_path\_name>  git rm <old\_path\_name>  git add <new\_path\_name> |
| Rename/Move a file |

**git log**

|  |
| --- |
| git log |
| Shows the history |
| git log -3 |
| Shows only 3 entries |
| git log –p | git log –patch |
| Shows the history with the patch change for each commit |
| git log –stat |
| Shows the history with the statistics doe each commit |
| git log –prety=oneline |
| Shows the history such that each entry fits in one line |
| Git log –graph |
| Shows log with asci graph shows branches and merges |
| git log –pretty=format:”%h:%an - %s” |
| Shows the history with the specified format. In this example each entry will contain the hash, author and commit message |
| git log – path/to/file/or/dir |
| Shows the history of the specified path or folder |
| git log –authour=”John Doe” –since=”01.04.2019” – before=”01.05.2019” |
| Shows the history of commits by John Doe between the specified dates |
| git log –grep “network” –grep “tm” |
| Shows the history of commits with commit message contains one of the specified words |
| git log –grep “network” –grep “tm” –all-match |
| Shows the history of commits with commit message contains all of the specified words |

**git checkout**

|  |
| --- |
| git checkout <branch> |
| Moves the head to the tip of this branch, and updates the working tree to this commit. |
| git checkout -- <filename> |
| Unmodify a modified file |
| git reset –hard C1 |
| * Moves Head from current commit to C1 * Empties index * Checkouts C1 to working tree |
| git reset HEAD -- <filename> |
| Unstage file:   * Moves HEAD to HEAD (no change) * Empties index from this file * Preserves the file in working tree |

**git reset**

|  |
| --- |
| git reset –soft C1 |
| * Moves Head from current commit to C1 * Preserves current commit in index * Preserve current commit in working tree |
| git reset –mixed C1 | git reset C1 |
| * Moves Head from current commit to C1 * Empties index * Preserves current commit in working tree |
| git reset –hard C1 |
| * Moves Head from current commit to C1 * Empties index * Checkouts C1 to working tree |
| git reset HEAD -- <filename> |
| Unstage file:   * Moves HEAD to HEAD (no change) * Empties index from this file * Preserves the file in working tree |

**Working with remotes**

|  |
| --- |
| git remote |
| Prints the remote servers |
| git remote –v |
| Prints the remote servers with their links |
| git clone <URL> |
| * Creates a directory, with .git folder (local repo) inside it. Pulls all the revisions from the URL repo (remote repo) to the local repo. Checkout the latest revision to the working copy * Adds a new remote with short name “origin” and url “<URL>” |
| git remote add <shortname> <URL> |
| Adds the remote specified by the link, with the specified short name |
| git fetch <remote> |
| Gets all info related to the current branch from remote.  git fetch = git fetch origin |
| git fetch –all |
| Gets info for all remote branches |
| git pull |
| If current branch has upstream, git pull = git fetch + git merge (to current branch from upstream) |
| git pull –all |
| Git fetch –all + merge to the current branch only from its upstream |
| git pull origin remotebranch |
| Fetch + merge (from specified remote branch to the current branch) |
| git push origin localbranch:remoteBranch |
| Pushes all new commits from localbranch to remotebranch |
| git remote show <remote> |
| * Shows remote branches, which has remote-tracking branches, which are new, which are deleted. * Shows the local branches and their upstreams |
| git remote rename <old\_name> <new\_name> |
| rename |
| git remote remove <remote> |
| remove |

**Tags**

|  |
| --- |
| git tag | git tag –l | git tag --list |
| List the tags |
| git tag –l “v0.\*“ |
| Lists all tags that start with “v0.“ |
| git tag –a <tagname> -m “Tagging message” |
| Create an annotated tag, with specified name and messages. Annotated tag is an object which contains tagger info, message, … |
| git tag <tagname> |
| Create a lightweight tag. Lightweigh tag is just a pointer to aspecific commit. |
| git show <tagname> |
| Shows info associated with the specified tag. |
| git tag –a <tagnme> <commit-sha1> |
| Creates a tag for a specific commit |
| git push origin <tagname> |
| Pushes the tag to the remote (tags are not pushed with a normal git push) |
| git push origin –tags |
| Pushes all tags to the remote |
| git push –delete origin <branchOrTagName> |
| Delete a branch or a tag from remote |
| git checkout <tagname> |
| * this checks out the commit pointed to by the tag * will be in detached head state * if we make modifications and commit, the commit will be only reachable by its hash * we can create a new branch from this commit if needed, to avoid the detached head state |

**Branching and Merging**

|  |
| --- |
| git branch <newBranchName> |
| Creates a new branch |
| git checkout <branchName> |
| Switch to this branch (HEAD points to it, local copy updates to it) |
| git checkout –b <branchName> |
| Create a branch and checkout it |
| git branch –d <branchName> |
| Delete a branch |
| git merge <branchName> |
| Merge “branchName” to current branch |
| git mergetool <filename> |
| Resolve a merge conflict using 4 window editor |
| git commit |
| Conclude a merge conflict resolution and proceed merging |
| git branch –list –all |
| List branches, including the remote branches |
| git branch –list –merged |
| List branches that are already merged to current branch. (Safe to delete those branches) |
| git branch –list –no-merged |
| List branches that are not merged to current branch. (Not safe to delete those branches) |
| git branch –vv |
| Shows which branches trackes which remote branches, and if a head or behind |