[1 Introduction 1](#_Toc516038314)

[2 IDE 1](#_Toc516038315)

[2.1 Visual Studio Integration? 1](#_Toc516038316)

[2.2 Folder integration 1](#_Toc516038317)

[2.2.1 Git GUI here 1](#_Toc516038318)

[2.2.2 Git Bash here 1](#_Toc516038319)

[2.2.3 Open in SmartGit 1](#_Toc516038320)

[2.2.4 Gitext Browse 1](#_Toc516038321)

[2.2.5 GitExt Commit 1](#_Toc516038322)

[3 Local level 2](#_Toc516038323)

[4 Global level 3](#_Toc516038324)

[4.1 Branching 3](#_Toc516038325)

[4.2 Merging 3](#_Toc516038326)

[4.3 Push and pull 3](#_Toc516038327)

[5 Commands 4](#_Toc516038328)

[5.1 Status 4](#_Toc516038329)

[5.2 Add 4](#_Toc516038330)

# Introduction

Git is a source control system.

It seems to work on a local and a global level.

# IDE

## Visual Studio Integration?

## Folder integration

When you right-click on local root directory, you get GIT options:

### Git GUI here

### Git Bash here

This is a command prompt, indicating your own PC and the root source directory you are standing on.

The, in (), you get the name of the branch, e.g. master.

### Open in SmartGit

### Gitext Browse

### GitExt Commit

# Local level

Somehow you can **stage** your own work to see what you have changed.

Then you can **commit** locally, what you have staged, or a subset thereof.

Can one revert back to work that you have locally committed?

Git essentially has 4 main statuses for the files in your local repo:

* **untracked:** The file is new, Git knows nothing about it. If you git add <file>, it becomes:
* **staged:** Now Git knows the file (tracked), but also made it part of the next commit batch (called the *index*). If you git commit, it becomes:
* **unchanged:** The file has not changed since its last commit. If you *modify it*, it becomes:
* **unstaged:** Modified but not part of the next commit yet. You can stage it again with git add

As you can see, a git add will **track** untracked files, and **stage** any file.

*Also: You can untrack an uncommited file with git rm --cached filename and unstage a staged file with git reset HEAD <file>*

When you add a file to start tracking, it also stages its contents.

If you want to add a file for tracking without staging it, you can use

git add -N

Both of the git add steps you identify do essentially the same thing, they simply have different explanations because of their arrival route.

The git add simply tells git that the file provided is a file that you desire to have, in its exact current form (its content), within its source control repository. At that point git will take a snapshot of the file (and it keeps a note in its index) so that it is ready for when you have all your changes to your files ready and added (i.e marshalled together in the staging area), for your git commit (with appropriate message ;-).

Once git has been told about that file (e.g. @avh's -N option) it will notice (track) changes to the file under the guise of various commands (such as git status). Thus, later, you have to explicitly tell git when you no longer want a file to be tracked (git rm <file>), and you can continue editing a file (locally) after you have added the version that will be in the commit. Almost obviously (or perhaps not), you can git add a file many times before you commit the final version.

# Global level

## Branching

It seems as though one can push your committed work to a branch.

You set the branch you are working on via **git checkout *branch,*** where branch can also be master.

## Merging

I would guess that one can merge a sub branch into the master with a **merge** command.

## Push and pull

Once you have set your branch, you can pull and push relative to it.

Once you have committed your work locally, you can **push** it.

# Commands

## Status

The first thing you see is your current branch and

the status of your current branch relative to origin/master.

Status looks at you working directory and identifies files that are:

* Not staged for commit

This would be files that you have modified.

* Not tracked

This seems to be files that are in the directory, but not in the project A. No.

* Not committed

use "git add <file>..." to update what will be committed

use "git checkout -- <file>..." to discard changes in working directory

My guess is that you get the copy that has been committed. I do not know where it is kept?

## Add

This seems to: Include a file to be tracked by Git. Under tracked I then understand noticing when it is being changed.

How does that differ from **stage**?

|  |  |  |
| --- | --- | --- |
| Git task | Notes | Git commands |
| [**Tell Git who you are**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config) | Configure the author name and email address to be used with your commits.  Note that Git [strips some characters](http://stackoverflow.com/questions/26159274/is-it-possible-to-have-a-trailing-period-in-user-name-in-git/26219423#26219423) (for example trailing periods) from user.name. | git config --global user.name "Sam Smith"  git config --global user.email sam@example.com |
| [**Create a new local repository**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-init) |  | git init |
| [**Check out a repository**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone) | Create a working copy of a local repository: | git clone /path/to/repository |
| For a remote server, use: | git clone username@host:/path/to/repository |
| [**Add files**](https://www.atlassian.com/git/tutorials/saving-changes#git-add) | Add one or more files to staging (index): | git add <filename>  git add \* |
| [**Commit**](https://www.atlassian.com/git/tutorials/saving-changes#git-commit) | Commit changes to head (but not yet to the remote repository): | git commit -m "Commit message" |
| Commit any files you've added with git add, and also commit any files you've changed since then: | git commit -a |
| [**Push**](https://www.atlassian.com/git/tutorials/syncing#git-push) | Send changes to the master branch of your remote repository: | git push origin master |
| [**Status**](https://www.atlassian.com/git/tutorials/inspecting-a-repository#git-status) | List the files you've changed and those you still need to add or commit: | git status |
| [**Connect to a remote repository**](https://www.atlassian.com/git/tutorials/syncing#git-remote) | If you haven't connected your local repository to a remote server, add the server to be able to push to it: | git remote add origin <server> |
| List all currently configured remote repositories: | git remote -v |
| [**Branches**](https://www.atlassian.com/git/tutorials/using-branches) | Create a new branch and switch to it: | git checkout -b <branchname> |
| Switch from one branch to another: | git checkout <branchname> |
| List all the branches in your repo, and also tell you what branch you're currently in: | git branch |
| Delete the feature branch: | git branch -d <branchname> |
| Push the branch to your remote repository, so others can use it: | git push origin <branchname> |
| Push all branches to your remote repository: | git push --all origin |
| Delete a branch on your remote repository: | git push origin :<branchname> |
| [**Update from the remote repository**](https://www.atlassian.com/git/tutorials/syncing) | Fetch and merge changes on the remote server to your working directory: | git pull |
| To merge a different branch into your active branch: | git merge <branchname> |
| View all the merge conflicts:  View the conflicts against the base file:  Preview changes, before merging: | git diff  git diff --base <filename>  git diff <sourcebranch> <targetbranch> |
| After you have manually resolved any conflicts, you mark the changed file: | git add <filename> |
| **Tags** | You can use tagging to mark a significant changeset, such as a release: | git tag 1.0.0 <commitID> |
| CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using: | git log |
| Push all tags to remote repository: | git push --tags origin |
| [**Undo local changes**](https://www.atlassian.com/git/tutorials/undoing-changes) | If you mess up, you can replace the changes in your working tree with the last content in head:  Changes already added to the index, as well as new files, will be kept. | git checkout -- <filename> |
| Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this: | git fetch origin  git reset --hard origin/master |
| **Search** | Search the working directory for foo(): |  |

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or

desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

INSTALL GIT

GitHub provides desktop clients that include a graphical user

interface for the most common repository actions and an automatically

updating command line edition of Git for advanced scenarios.

GitHub for Windows

https://windows.github.com

GitHub for Mac

https://mac.github.com

Git distributions for Linux and POSIX systems are available on the

official Git SCM web site.

Git for All Platforms

http://git-scm.com

V 1.1.1

training@github.com

training.github.com

Learn more about using GitHub and Git. Email the Training Team or visit

our web site for learning event schedules and private class availability.

GIT CHEAT SHEET

SYNCHRONIZE CHANGES

Register a repository bookmark and exchange version history

$ git fetch [bookmark]

Downloads all history from the repository bookmark

$ git merge [bookmark]/[branch]

Combines bookmark’s branch into current local branch

$ git push [alias] [branch]

Uploads all local branch commits to GitHub

$ git pull

Downloads bookmark history and incorporates changes

REFACTOR FILENAMES

Relocate and remove versioned files

$ git rm [file]

Deletes the file from the working directory and stages the deletion

$ git rm --cached [file]

Removes the file from version control but preserves the file locally

$ git mv [file-original] [file-renamed]

Changes the file name and prepares it for commit

SAVE FRAGMENTS

Shelve and restore incomplete changes

$ git stash

Temporarily stores all modified tracked files

$ git stash list

Lists all stashed changesets

$ git stash pop

Restores the most recently stashed files

$ git stash drop

Discards the most recently stashed changeset

REDO COMMITS

Erase mistakes and craft replacement history

$ git reset [commit]

Undoes all commits after [commit], preserving changes locally

$ git reset --hard [commit]

Discards all history and changes back to the specified commit

REVIEW HISTORY

Browse and inspect the evolution of project files

$ git log

Lists version history for the current branch

$ git log --follow [file]

Lists version history for a file, including renames

$ git diff [first-branch]...[second-branch]

Shows content differences between two branches

$ git show [commit]

Outputs metadata and content changes of the specified commit

SUPPRESS TRACKING

Exclude temporary files and paths

$ git ls-files --other --ignored --exclude-standard

Lists all ignored files in this project

\*.log

build/

temp-\*

A text file named .gitignore suppresses accidental versioning of

files and paths matching the specified patterns