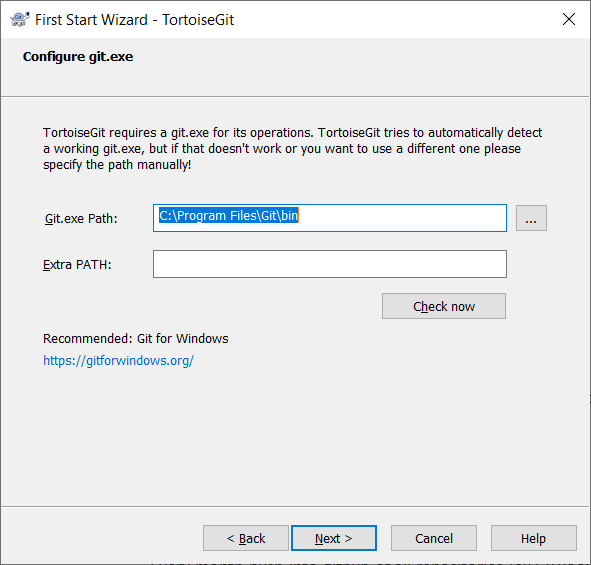
# AP Info

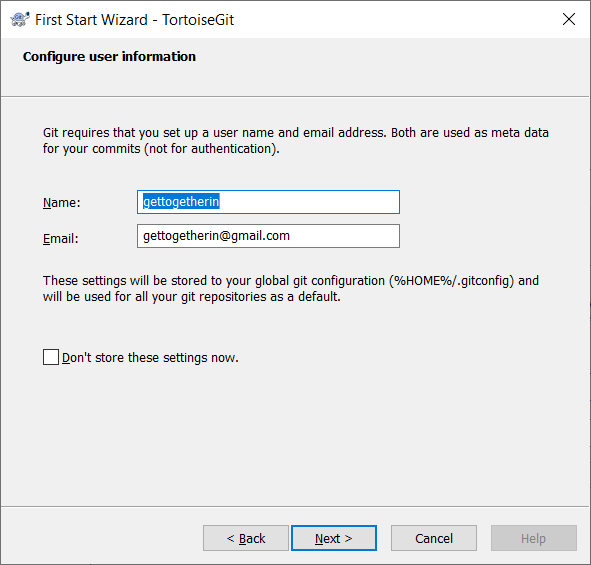
## Installed in T480

### Tortoisegit

<https://tortoisegit.org/>

Stable: 2.13.0 | Released on 2022-01-31





### Git Bash

### GitHub Desktop

* Used the same existing repostirory AP\_Notes – mapped and pulled

# Q&A

# Git Tools

## Git

## GitDesktop

## Tortoisegit

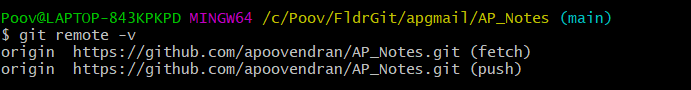
## GitHub

# 2b Organized

## AP Operation

* Every month push into github of all repositories (ex C:\Poov\FldrGit\apgmail\AP\_Notes)
  + Go to the folder – Rt Click - Git Bash Here
  + git status
    - shows modified files in red
  + git add .
    - Files will be added into staging (local repository)
  + git status
    - Modified and updated in local repo will be in green color
  + git push
    - Push the updated files from staging to github.

## ???

* I committed one file n times in main, how can compare with earlier version?
* 

## Git/Github

* Create Repository either in local or Pull from GitHub
* Add files / modify files in Local –
* Commit the changes in local to staging (local)
* Push from staging to GitHub

## a

## Ref URLs

<https://learngitbranching.js.org/> - about branching

<https://www.tutorialspoint.com/git/>

Git Desktop : https://desktop.github.com/ TortoiseGit: https://tortoisegit.org/

Git Cheat Sheet: https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf

Getting Started: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Basics: https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Branching: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Github Setup: https://git-scm.com/book/en/v2/GitHub-Account-Setup-and-Configuration

Git Tools: https://git-scm.com/book/en/v2/Git-Tools-Revision-Selection

Git Commands: https://git-scm.com/book/en/v2/Git-Commands-Setup-and-Config

a

## Books Read

## Distributed Version Control with Git\_ Mastering the Git command line - .pdf

## GitHub Repository

* GitHub is a web-based cloud service to host your source code(Git repositories). It is a centralized system.
* Git doesn’t require GitHub but GitHub requires Git.

## How to use two accounts – one for global and one for local

How can i use two differernt credentials ie one for office and one for personal?

You have the below 2 options, Based on your preferred way(ssh or password) base access to git account:-

SSH based access:- create 2 ssh key-pair one for your company git account and one for your own git account. you need to add public ssh keys on both the git account by following this article.

Password bases access :- In this case you don't need to do anything, you just need to give the username and password on git push etc.

Important:- Now you need to add the git configs(git username,email etc) for your system, git has option to set these at gloabl and local leval. I would recommend setting the user.email and user.name setting globally according to your organization, to avoid commit to your company repo which has your private git username and email.

for example below git command will show the gloabl setting of git :-

git config --global --list

user.name=<firstname.lastname>

user.email=<company mail address>

And to set the git username and password in your private git repo, use below command, inside your repository

git config --local user.name "amit"

git config --local user.email "amit@mail.com"

You can confirm, that your own private repo does not have, your company username and password by running git config --edit command or git config --local --list.

## Create account in GitHub

## Create New Repository in Github

Create New Repo -> Repositories -> NEW

To access the code, we need Personal Access Tokens Settings -> Developer Settings -> Personal access tokens-> Generate new token

Copy the token since it cannot be viewed once we navigate the page

Windows Control Panel-> Credential Manager ->windows credential ->Add windows credential -> github.com -> Username -> pwd: Token.

## Git

* Version control tool (software) to track the changes in the source code.
* If you want to delete a Git (local) repository, you can simply delete the folder which contains the repository
* Git doesn’t require GitHub but GitHub requires Git.
* Initialize git - set identity (in cmd window)
  + If you use only one id use global - if you want to use multiple ids for multiple project remove global
* $ git config --global user.name "John Doe"
* $ git config --global user.email johndoe@example.com
* move to proper command place
  + git mv file\_from file\_to ???
  + git remote -v ???
  + git log --since='last month' --pretty=format:'%h;%an;%ad;%s' --author='Ionut Colceriu' > ~/log.csv

## Install Git (GitBash)

https://git-scm.com/download/win

## Keys

q – exit from running comment – example after giving git log HEAD it will not come out – we need to give q

## git init

## git clone

The following command clones an existing repository using the Git protocol. The Git protocol uses the port 9148 which might be blocked by firewalls.

# clone online repository

git clone git://github.com/vogella/gitbook.git

If you have SSH access to a Git repository, you can also use the ssh protocol. The name preceding @

is the user name used for the SSH connection.

# clone online repository

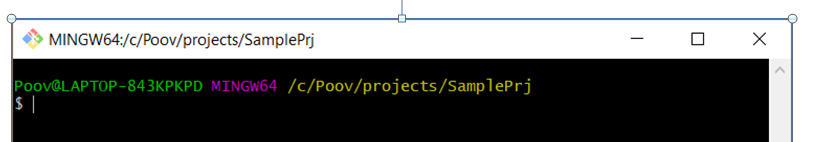
git clone ssh://git@github.com/vogella/gitbook.git

Alternatively you could clone the same repository via the http protocol.

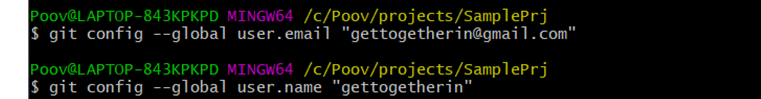
# the following will clone via HTTP

git clone http://github.com/vogella/gitbook.git

Goto folder in windows explorer – Rt Click – Git Bash Here



Check global and local userid and email id already set. If not set that

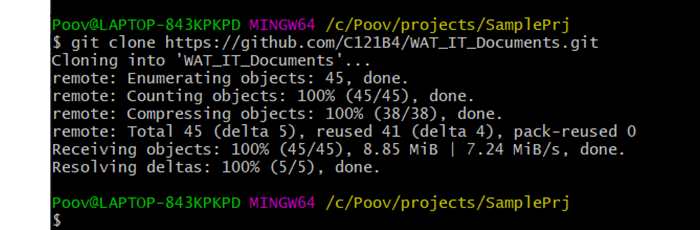


Clone the project by using github URL

git clone <https://github.com/C121B4/WAT_IT_Documents.git>



Asking to login in the website – once login your github account it will clone in your local repository



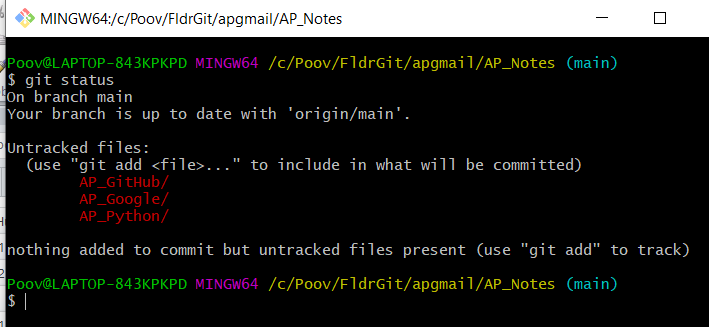
## git checkout filename

* checkout - # CAREFUL this deletes the local changes in the tracked file

## git log

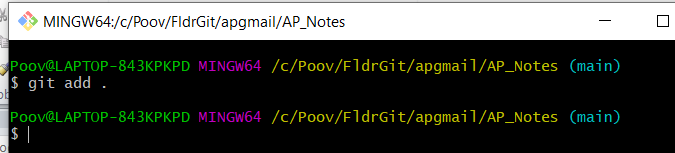
## git status

* to check for any untracked files

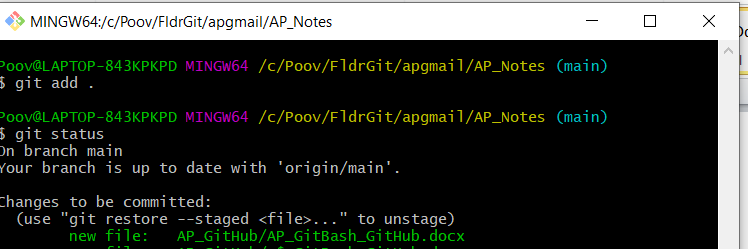


## git add .

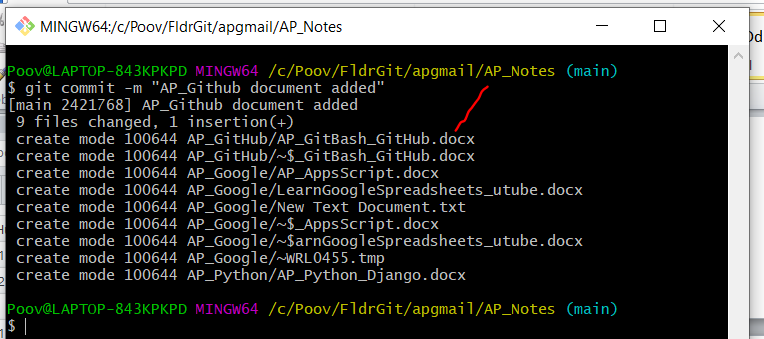
* adds the untracked files



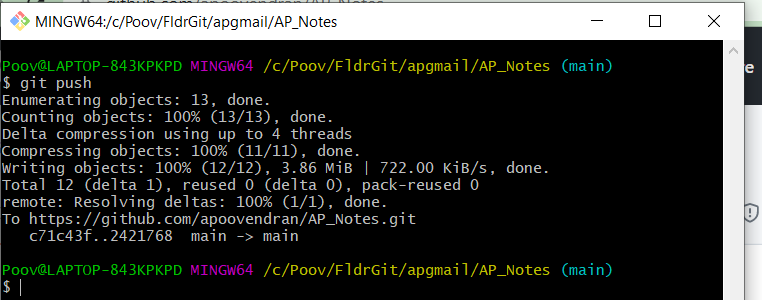
git status to know the status

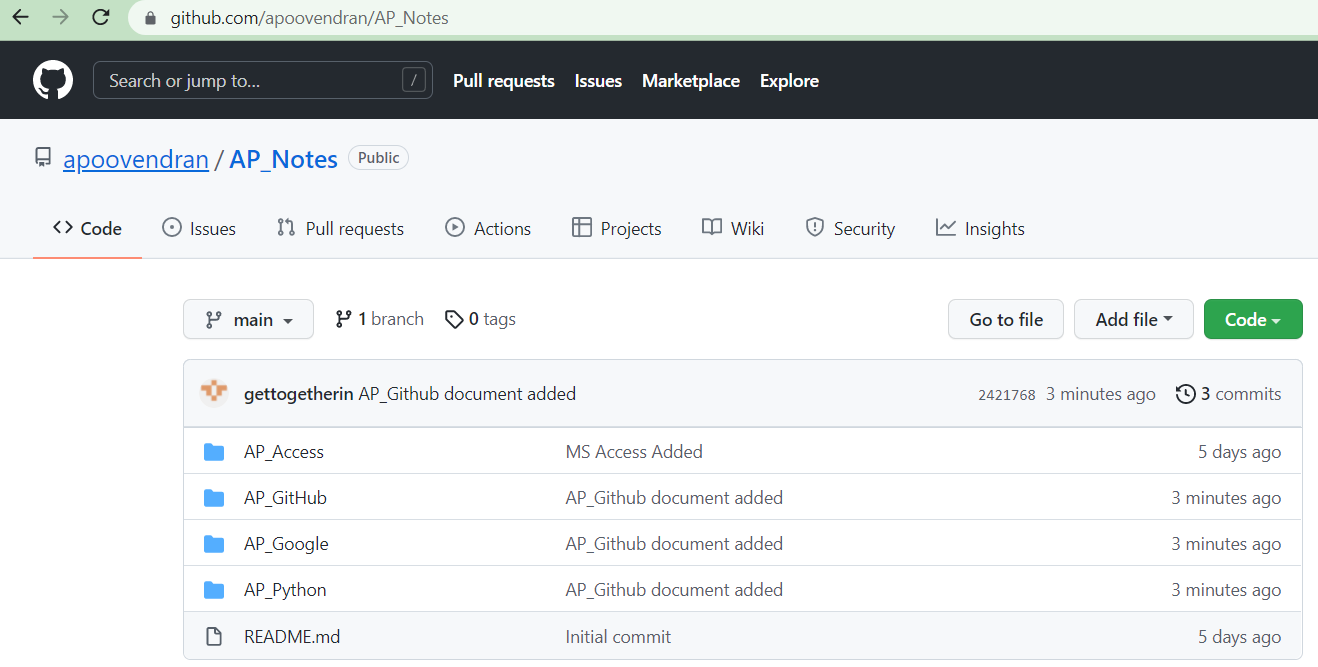


## git commit -m "message“

* creates a copy that needs to be updated in repository
* 

## git push

* pushes the code changes into the repository
* 
* In the github URL



## git commit --amend

# assuming you have something to commit

git commit -m "message with a tpyo here"

# amend the last commit – ie revert back to earlier stage

git commit --amend -m "More changes - now correct"

## git pull

## .gitignore

St # remove directory .metadata from git repo

git rm -r --cached .metadata

# remove file test.txt from repo

git rm --cached doNotTrackFile.txt op tracking files based on the .gitignore file

## Handling with Branches

* In case of merge conflicts, need to resolve conflicts and then merge.
* For review purposes we usually create a pull request in github after pushing the code into the repository, review the code and then merge the branches

### git branch

# lists available branches

git branch

# lists all branches including the remote branches

git branch –a

# lists branches in the remote repositories

git branch –r

# create a new branch - syntax: git branch <name> <hash>

# <hash> in the above is optional

git branch testing

### git checkout -b "newdevbranch"

* To start working in a branch you have to checkout the branch. If you checkout a branch, the HEAD pointer moves to the last commit in this branch and the files in the working tree are set to the state of this commit.
* # switch to your new branch
* git checkout testing
* # do some changes
* echo "Cool new feature in this branch" > test01
* git commit -a -m "new feature"
* # switch to the master branch - above commit will not be available in mastet,
* git checkout master

# create branch and switch to it

git checkout -b bugreport12

# creates a new branch based on the master branch without the last commit

git checkout -b mybranch master~1

# rename branch

git branch -m [old\_name] [new\_name]

To delete a branch which is not needed anymore, you can use the following command. You may get an

error message that there are uncommited changes if you did the previous examples step by step. Use

force delete (uppercase -D) to delete it anyway.

# delete branch testing

git branch -d testing

# force delete testing

git branch -D testing

# check if branch has been deleted

git branch

### git pull

* gets the latest version code from GitHub

### To merge new branch to Master

#### git checkout master

* switches to existing branch

#### git merge newdevbranch

## Q&A

## Push changes of a branch to a remote repository

# push current branch to a branch called "testing" to remote repository

git push origin testing

# switch to the testing branch

git checkout testing

# some changes

echo "News for you" > test01

git commit -a -m "new feature in branch"

# push all including branch

git push

## Differences between branches

# shows the differences between

# current head of master and your\_branch

git diff master your\_branch

compare a branch called your\_branch with the master branch the following command shows the

changes in your\_branch and master since these branches diverged

# shows the differences in your

# branch based on the common

# ancestor for both branches

git diff master...your\_branch

## Using tags

* Git supports two different types of tags, lightweight and annotated tags.
* Convention is that release tags are labeled based on the [major].[minor].[patch] naming scheme, for example "1.0.0". Several projects also use the "v" prefix
* git tag #list tags
* git tag -l <pattern> # Search by pattern for a tag
* git checkout <tag\_name>
* Push tags

# push a tag or branch called tagname

git push origin [tagname]

# to explicitly push a tag and not a branch

git push origin tag <tagname>

# push all tags

git push –tags

* delete tags

# delete tag locally

git tag -d 1.7.0

# delete tag in remote repository

# called origin

git push origin :refs/tags/1.7.0

a

a

## Reviewing changes before a commit

# see the current status of your repository

# (which files are changed / new / deleted)

git status

See the differences in the working tree since the last commit

# optional you can also specify a path to filter the displayed changes

# path can be a file or directory

git diff [path]

See differences between staging area and last commit

git diff --cached

## Analyzing changes in the repository

Analyzing the commit history with git log

# show the history of commits in the current branch

git log

# uses shortend but unique SHA-1 values

# for the commit objects

git log --abbrev-commit

# show the history of commits in one line

# with a shortened version of the commit id

# --online is a shorthand for "--pretty=oneline --abbrev-commit"

git log --oneline

# show the history as graph including branches

git log --graph –oneline

View the change history of a file

# git log filename shows the commits for this file

git log [file path]

# Use -p to see the diffs of each commit

git log -p filename

# --follow shows the entire history

# including renames

git log --follow -p file

Configuring output format

# command must be issued in one line, do not enter the line break

git log --pretty=format:'%Cred%h%Creset %d%Creset %s %Cgreen(%cr)

%C(bold blue)<%an>%Creset' --abbrev-commit

See all commits of a certain user

git log --author=lvogel

See the differences introduced by a commit

git show <commit\_id>

See the difference between two commits

# directly between two commits

git diff HEAD~1 HEAD

# using commit ranges

git diff HEAD~1..HEAD

See the files changed by a commit

git diff-tree --name-only -r <commit\_id>

Using git blame and git shortlog

* The git blame command allows you to see which commit and author modified a file on a per line base. That is very useful to identify the person or the commit which introduced a change.

# git blame shows the author and commit per

# line of a file

git blame [filename]

# the -L option allows limiting the selection

# for example by line number

# only show line 1 and 2 in git blame

git blame -L 1,2 [filename]

git shortlog for release announcements

# gives a summary of the changes by author

git shortlog

# compressed summary

# -s summary, provides a commit count summary only

# -n sorted by number instead of name of the author

git shortlog –sn

# see the commits by the author "Lars Vogel"

git shortlog --author="Lars Vogel"

# see the commits by the author "Lars Vogel"

# restricted by the last years

git shortlog --author="Lars Vogel" --since=2years

# see the number of commits by the author "Lars Vogel"

git shortlog -s --author="Lars Vogel" --since=2years

## Storing file changes temporary with git stash

* Git provides the git stash command which allows you to record the current state of the working directory and the staging area and to revert to the last committed revision.
* This allows you to pull in the latest changes or to develop an urgent fix. Afterwards you can restore the stashed changes, which will reapply the changes to the current version of the source code.

# create a stash with uncommitted changes

git stash

# do changes to the source, e.g., by pulling

# new changes from a remote repo

# afterwards, re-apply the stashed changes

# and delete the stash from the list of stashes

git stash pop

# create a new branch from your stack and

# switch to it

git stash branch newbranchforstash

## Undoing uncommitted changes

Remove untracked files with git clean

git clean # Be careful with this command. All untracked files are removed if you run this command.

# make a dry-run to see what would happen

# -n is the same as --dry-run

git clean -n

# delete, -f is required if

# variable clean.requireForce is not set to false

git clean -f

# use -d flag to delete new directories

# use -x to delete hidden files, e.g., ".example"

git clean –fdx

Revert uncommitted changes in tracked files

// add changes to staging area

git add unwantedstaged.txt

git add test02

# see the status

git status

//# remove test02 from the staging area

git reset test02

# remove unwantedstaged.txt from the staging area

git reset unwantedstaged.txt

# see the status

git status

Remove staging area based on last commit change

echo "change which should be removed later" > test01

# add the file to the staging area

git add test01

# restores the file based on HEAD in the staging area

git reset HEAD test01

## Undoing committed changes

soft / mixed (default) / hard

Deleting changes in the working tree and staging area for tracked files

# removes staged and working tree changes

# of committed files

git reset –hard

Using git reset to squash commits

# squashes the last two commits

git reset --soft HEAD~1 && git commit -m "new commit message"

# revert a commit

git revert commit\_id

## Recovering files or commits

a

Checkout based on commits and working tree

# displays the commit history of the repository

# which contains the commit ID, author, message etc.

git log

# checkout the older revision via

git checkout [commit\_id]

# based on the example output this could be

git checkout 046474a52e0ba1f1435ad285eae0d8ef19d529bf

# or you can use the abbreviated version

git checkout 046474a5

View file in different revision

# [reference] can be a branch, tag, HEAD or commit ID

# [file\_path] is the file name including path

git show [reference]:[file\_path]

# to make a copy to copiedfile.txt

git show [reference]:[file\_path] > copiedfile.txt

Restore a deleted file in a Git repo

# see history of file

git log -- <file\_path>

# checkout file based on predecessors the last commit which affect it

# this was the commit which delete the file

git checkout [commit] ^ -- <file\_path>

# alternatively use git rev-list

git rev-list -n 1 HEAD -- <file\_path>

# afterwards, the same checkout based on the predecessors

git checkout [commit] ^ -- <file\_path>

See which commit deleted a file

# see the changes of a file, works even

# if the file was deleted

git log -- [file\_path]

# limit the output of Git log to the

# last commit, i.e. the commit which delete the file

# -1 to see only the last commit

# use 2 to see the last 2 commits etc

git log -1 -- [file\_path]

# include stat parameter to see

# some statics, e.g., how many files were

# deleted

git log -1 --stat -- [file\_path]

Git reflog and restoring commits

git reflog vs git log

* The Git reflog command gives a history of the complete changes of the HEAD reference.

Remote and local tracking branches and git fetch

# list all remote branches

git branch –r

# delete remote branch from origin

git branch -d -r origin/[remote\_branch]

Delete a branch in a remote repository

# delete branch in a remote repository

git push [remote] :branch

OR

# delete branch in a remote repository

git push [remote] --delete :[branch]

Example

git push origin :testbranch

OR  
git push ssh://[URL\_to\_repo] :testbranch

Tracking branches

* If you clone a Git repository, your local master branch is created as a tracking branch for the master branch of the origin repository (short: origin/master) by Git.

# origin/master used as example, but can be replaced

# create branch based on remote branch

git branch [new\_branch] origin/master

# use --track,

# default when the start point is a remote-tracking branch

git branch --track [new\_branch] origin/master

# instruct Git to create a branch which does

# not track another branch

git branch --no-track [new\_branch\_notrack] origin/master

# update this branch to track the origin/master branch

git branch -u origin/master [new\_branch\_notrack]

See the branch information for a remote repository

# show all remote and tracking branches for origin

git remote show origin

The git fetch command updates your remote-tracking branches, i.e., it updates the local copy of

branches stored in a remote repository

git fetch origin

# simplification of the fetch command

# this runs git fetch for every remote repository

git remote update

# the same but remove all stale branches which

# are not in the remote anymore

git remote update –prune

Compare remote-tracking branch with local branch

# show the log entries between the last local commit and the

# remote branch

git log HEAD..origin/master

# show the diff for each patch

git log -p HEAD..origin/master

# show a single diff

git diff HEAD...origin/master

# instead of using HEAD you can also

# specify the branches directly

git diff master origin/master

Rebase your local branch onto the remote-tracking branch

# assume you want to rebase master based on the latest fetch

# therefore check it out

git checkout master

# update your remote-tracking branch

git fetch

# rebase your master onto origin/master

git rebase origin/master

Fetch compared with pull

* The git pull command performs a git fetch and git merge (or git rebase based on your Git settings). The git fetch does not perform any operations on your local branches. You can always run the fetch command and review the incoming changes.

## Merging changes of branches

* Merge strategies - Octopus, Subtree, Ours
* The default strategy called recursive merge strategy

## Free Public Repositories

### github / bitbucket

## Running your own Git server

### working on two repository

You can add another remote repository called remote\_name via the following command.

# add remote

# syntax: git remote add <remote\_name> <url\_of\_gitrepo>

# git remote add mysecondrepo <url\_of\_gitrepo>

git remote -v

# see all repos

## Good practices in Git

* Writing meaningful commit messages

a

aa