# Getting Started

* Download Git

<https://gitforwindows.org/>

# Working Locally with GIT

## git init

* use terminal to create a folder for git and initialize it

cd C:\

mkdir gittestdir

cd gittestdir

git init

## git config --list

* check global, system and branch level configurations

git config --global --list

git config --system --list

git config --list

## git config parameters

* change any config you would like

git config user.name=Ekrem Çek

git config user.email=ekremcek19@hotmail.com

git config help.autocorrect=1

git config color.ui=auto

git config core.editor='C:\Program Files\Notepad++\notepad++.exe' -multiInst -notabbar -nosession -noPlugin

(you can use just vim instead of ‘path to notepad’)

## git add/status/log/commit

* Add a file to git repo, commit and update the file again with comments

echo “hello git!” > test.txt

git add test.txt

git status

git log

git commit (a text editor opens to enter comment)

vi test.txt (add a new line and write something new. Exit with !wq)

git status (shows that it has been modified)

git add -u (adds ALL MODIFIED FILES for an update, NOT NEW ONES)

git status (shows that it has been staged)

git commit -m “Insert an update comment here” (-m removes the necessity of a text editor during a commit)

git log (shows latest commits)

## git diff/diff HEAD

* Use git diff command to get differences between commits by using sha code’s first 7 chars

git diff a68ad9..6f752e (chars are sha codes from first and second commits, found within log command)

* Use git diff with HEAD command to get differences

git diff HEAD~1..HEAD (head means latest commit. Tilde (~) char means previous commits (like minus) so HEAD~1..HEAD means differences between the latest and the one previous

git diff HEAD~1.. (this also works if you want to get the difference until the latest. HEAD can be omitted.)

## git add -u/-A

* Add ALL files to staging before commit, committing and checking diff

touch file1.txt file2.txt

git status (not added)

git add -u

git status (not added since -u only adds updates)

git add -A (this will add EVERYTHING so USE IT WISELY)

git status (now everything is added to staging)

git commit -m "testing git add -A" (all changes are COMMITED)

git log (see all changes)

git diff HEAD~1.. (see current changes)

* Update files but commit separately to track changes in detail

vi file1.txt (add a new line, write something)

vi file2.txt (add a new line, write something)

git status (see that they are modified)

git add file1.txt (only add file1 to staging for commit)

git status (see that only file1 is waiting for commit)

git commit -m “only file 1 committed ”

git add file2.txt (file2 is added to staging now)

git commit -m “only file 2 committed”

git log (see commits in detail)

## git add/status/log/commit

* Delete files from repository and rename files

rm file2.txt

git status (shows that it’s deleted)

git add -u (stages the deletion)

echo “I am new!” > file3.txt (create a new file)

git status (see that it’s untracked, needs to be staged and committed)

git add file3.txt (stage the new file for commit)

git status (see that it’s staged)

mv file1.txt fileX.txt (rename the file1)

git status (see that on status, git recognizes that file1 is deleted and there is a new file named fileX.txt)

git add -A (stage all changes now)

git status (see that git recognizes that file1 was renamed to fileX)

git commit -m "tested rename" (commit changes)

git log (see that our commit and comment is in place already)

## git checkout/reset --hard

* Undo the changes on working copy | revert back to HEAD version

vi file2.txt (add a new char to the file)

git status (see that the file has been modified but lets assume that you made a mistake and want to roll back the file to it’s original state)

git checkout file2.txt (get the file from repository and overwrite the working copy)

git status (nothing to commit, no changes)

cat file2.txt (file has returned to it’s original state)

rm file2.txt (delete the file)

vi file3.txt (delete the file content)

git status (see that there are couple of changes, but you want to roll back ALL of them )

git reset --hard (resets all files to the HEAD version)

git status (nothing to commit)

## git reset --hard/reset --soft

* Undo, redo changes in the repository using git reset soft head

git reset --soft HEAD~1 (working copy is rolled back to their state before the last commit. **Changes will be already staged** but you have the option to change anything you would like)

git status (changes will be staged but not committed)

git log (you will realize that latest HEAD will be the previous commit version)

git commit -m “recommitting latest version HEAD”

git reset --hard HEAD~1 (working copy will be rolled back to previous head and all the changes after previous head are discarded)

git status (there will be no change about latest HEAD)

## git clean -n/-f

* Cleaning the working copy with **git clean**

touch temp1.txt temp2.txt (create some temp files to test)

git status (see that files are not staged)

git clean -n ( the -n option will show you what the clean command **WILL DO**)

git clean -f (the -f option actually does the operation listed in -n command which is removing the unstaged files.)

git status (see that working copy is clean)

## git ignore

* Ignoring files with .gitignore file

mkdir logs (create a test directory)

cd logs (go to the new directory)

touch templog.log templog2.txt (create test files with different extensions)

cd ..

vi .gitignore (create gitignore file in home directory of git)

/logs/\*.log (add this line and save it with !wq)

git add -A (add everything to staging)

git status (see that templog.log file is not even mentioned whereas templog2.txt is marked for commit action)

git commit

# Working Remotely with GIT

## Open a GitHub account

https://github.com/join?

## Configure the proxy (to access within Nokia)

git config --global http.proxy <http://proxyUsername:proxyPassword@proxy.server.com:port>

EXAMPLE: git config --global http.proxy <http://'ecek':'MyPassword!'@135.245.192.7:8000>

## git clone

mkdir GitTraining

cd GitTraining

git clone <https://github.com/ekoceko/Training.git>

## git log --oneline / --graph /shortlog /--summary

git log --oneline ( | wc -l )

git log --oneline --graph (shows branches)

git shortlog (lists committers and commit messages)

git shortlog --summary (list committers)

More statistics can be checked from github webpage

## git show HEAD/ (sha)

git show HEAD (HEAD~1 shows the previous one, head shows this one)

git log --oneline (take one of the sha code to use below. It’s something like e46820d)

git show e46820d (shows the requested commit details)

## git remote

git remote (shows the name of origin)

git remote -v (-v shows the URL of the origin for fetch and push)

## GIT Protocols

* http(s)> 80/443 > http(s)://github.com/ekoceko/Training.git > Read-write, pass auth, firewall friendly
* git > 9418 > git://github.com/ekoceko/Training.git > Read-only, Anonymous only
* ssh > 22 > git@ github.com:ekoceko/Training.git > Read-write, SSH key for auth
* file > n/a > /c/GitTraining/Training > Read-write, local only

## branches and tags

git branch (displays all local branches)

git branch -r (displays remote branches)

git tag (displays tags)

## fetching from repository (git remote -v & git fetch)

* first, switch to first folder without remote connection

mkdir gittestdir2

cd “PathToThisFolder”/gittestdir2

git init

git remote -v (see that there is no remote for this folder)

git remote add origin <https://github.com/ekoceko/Training.git> (adds remote address)

git pull

* Add a new file to GitTraining remote repository, commit and push.
* Switch back to gittestdir2 again and continue

git status (this repo does not realize that it was updated)

git fetch (learns remote changes)

git status (see that git notifies you that changes had been made)

git merge origin/master (merges the changes with the local copy)

git status (nothing to commit/update)

## git pull | git branch --set-upstream

git branch -r (shows current branch, which is master)

git fetch; git merge origin/master

git pull (does the same thing with the above command as a shortcut)

* If pull does not work, you should set an upstream with the below command. Normally, clone command automatically sets the upstream branch

git branch --set-upstream master origin/master

git pull (now works)

* if you don’t want to set an upstream, you can use below syntax to do the operation but upstream is recommended

git pull origin master

## git push

vi README.txt (open our temp file and add some lines)

git commit -am “update to be pushed” (add a comment and commit all the modified files with -a modifier. You don’t need to use git add -u beforehand if you use this version of commit)

git status (see that your branch is ahead of the remote)

git push (git sends files to remote)

git remote -v (shows the remote folder. You can remove the remote with git remote rm origin command and add a new remote with different protocol using git remote add origin XXXX)

## git tag / -a / -s / -v and signature config

git tag v1.0 (tag the current revision with a string like v1.0)

git tag (display current tags)

git tag -a v1.0\_with\_message (opens up a text editor to enter a message)

* you need to generate a key if you want to use signature with tags

gpg --gen-key (follow the instructions to generate a key. Remember to record the passphrase)

gpg --list-keys (get the key from pub row XXXXX/ThisIsSigninKey Date. It is something like this EB5CB724)

git config --global user.signingkey 0A46826A (set the key you just get)

git tag -s v1.0\_signed (tag the version and sign it. Enter passphrase when prompted)

git tag -v v1.0\_with\_message (-v tries to verify but system cannot because it was not signed)

git tag -v v1.0\_signed (you will see that signature is ok and it was signed)

git push --tags (tags are not pushed by default. After this command, the tags will be created on github and you will be able to see them)

## add git alias

git log --graph --oneline --all --decorate (too long to type)

git config --global alias.**lgs** "log --graph --oneline --all --decorate" (give a name to alias and type all the commands and parameters you want to use, omitting the first git)

git config --global --list (see that your config is added)

## git branch

git branch testbranch (create a branch)

git checkout testbranch (switch to that branch)

git lgs (check logs, test the alias also)

echo "adding a line for branch test" >> README.txt (just to make a change, add a line)

git commit -am "added files to the branch" (stage and commit files to the branch)

git lgs (see that branch points to the HEAD but the master points to HEAD~1 after commit)

git checkout master (switch to master again)

git lgs (see that master points to HEAD~1 since we haven’t merged anything with master)

* create another branch from a previous commit and apply an imaginary fix

git branch fix