**GIT – Commited local version**

**INSTALLATION & CONFIGURATION:**

www.git-scm.com -> download

* Git Bash
* Git CMD
* Git GUI

Verification:

* Git Bash
* -> insert “git version”
* > “exit”
* CMD
* -> insert “git version”
* > “exit”

Help:

* Git Bash
* -> insert “git help”
* -> insert “git help -a” -> available commands
* -> insert “git help {commandName)”
* -> insert “clear”

Configuration:

* Git Bash
* -> insert “**git config --global user.name** **“**Dorota Jankowiec**”** “
* -> insert “**git config --global user.email “**dj@git.example**”** “
* -> insert “**git config --global --list**" -> wyświetla listę ustawień
* -> insert “**cat ~/.gitconfig**"**[[1]](#footnote-1)** -> wyświetla listę ustawień z pliku .gitconfig ale z podziałem na typy ustawień (lepsze)

**Atom** - the graphical text editor. (better that Notepad++ ?)

**Atom with Git Integration**

Atom should provide the **atom** command available on within Git Bash and the Windows Command Prompt. In either case, you can use the **git config** command to setup the Git Integration with Atom:

**git config --config --global core.editor "atom -w"**

**GIT BASH:**

Git Bash commends:

* **pwd** -> **p**rint **w**orking **d**irectory
* **cd [catalog name]** -> **c**hange **d**irectory “cd exampCatalog”; “cd ..” – poziom wyżej
* **cd [path]** -> **c**hange **d**irectory np “cd C:\Repos\Axa”
* **ls** -> list of all files
* **ls -l** -> list of all files with properties
* **ls -al** -> list of all files with properties + hidden
* **cd ~** -> back to users **home directory**
* **clear**  -> clear screen
* **exit** -> exit window
* **which [command]** -> directory of command
* **help echo** -> outputs the strings it is being passed as arguments

  polecenie służące głównie do wypisywania tekstu w [konsoli](https://pl.wikipedia.org/wiki/Wiersz_polece%C5%84).

np. **echo [parametr]**

* **mkdir [name]** -> **m**a**k**e **dir**ectory; tworzy w obecnej lokalizacji nowy folder

o nazwie name

* **touch [name.ex]** -> tworzy w obecnej lokalizacji nowy plik o nazwie name

i rozszerzeniu .ex

* **cat [name.ex]** -> printuje zawartość pliku o nazwie name i rozszerzeniu .ex
* … **>** … -> przekierowanie wyniku strony lewej do strony prawej;

NADPISANIE

* … **>>** … -> przekierowanie wyniku strony lewej do strony prawej;

DOPISANIE

* **less [name.ex]** -> printuje zawartość pliku o nazwie name i rozszerzeniu .ex

w przypadku gdy ma dużą zawartość

* **cp [name1] [name2]** -> copy of file; source: name1; destination: name2
* **mv [name1] [name2]** -> **m**o**v**e; zmiana nazwy pliku; old: name1; new: name2
* **rm [name.ex]** -> **r**e**m**ove file o nazwie name i rozszerzeniu .ex
* **rmdir [name]** -> **r**e**m**ove folder (directory) o nazwie name
* **rm -rf [name]** -> force **r**e**m**ove folder (directory) o nazwie name
* **vi [name.ex]**  -> open VI editor for file o nazwie name i rozszerzeniu .ex;

into insert mode “I” from command mode

* **:q (from VI editor)** -> **q**uick; back into command mode “ESC” from insert mode;

DO NOT SAVE CHANGES

* **:x (from VI editor)** -> **q**uick; back into command mode “ESC” from insert mode;

SAVE CHANGES

Example:

* echo “Hello World” **>** demo.txt -> wpisuje tekst do pliku w obecnej lokalizacji

**GIT COMMANDS:**

* **git clone [path]** -> klonuje (pobiera) repozytorium z repozytorium kodu

(np. githuba) do aktualnego folderu (z miejsca uruchomienia GtBasha / aktualnego directory gitbasha)

* **git status** -> pokazuje na jakim branchu pracujemy i jakie wprowadzono

zmiany (new unstaged files -> add to staged files)

* **git log** -> Pokazuje dotychczasowe commity + ich nr, aktualny branch.
* **notepad [name.txt]** -> (tworzy plik o nazwie name gdy nie istnieje oraz) otwiera plik

w notepad’zie

* **git add [name.ex]** -> dodaje **zmianę** do staging area (dodaje do śledzonych)

zmiany lub cały plik

* **git commit -m “…”** -> commituje zmiany z wiadomością w “…”
* **git commit -a -m “…”** -> dodaje do śledzonych + commituje w jednej operacji
* **git remote -v** - > pokazuje adres repozytorium na serwerze
* **git remote show origin** -> pokazuje adres repozytorium na serwerze + szczegóły
* **git push origin master** -> push brancha master na serwer (origin)
* **git pull origin master** -> pobranie wersji z serwera do brancha master

**COMMAND SHELL**

ksdajdxghbdxn

* **dir** -> **di**splay list of all files and folders (analog. jak ls) np. **dir \*.txt**
* **dir /a** -> list of all files and folders + hidden
* **cls** -> clear screen (analogicznie dla clear)
* **help echo** -> outputs the strings it is being passed as arguments

  polecenie służące głównie do wypisywania tekstu w [konsoli](https://pl.wikipedia.org/wiki/Wiersz_polece%C5%84).

np. **echo [parametr]**

* … **>** … -> przekierowanie wyniku strony lewej do strony prawej;

NADPISANIE

* **where [name.ex]** -> zwraca scieżkę do program o nazwie name i rozsz. ex
* **notepad [name.txt]** -> otwiera plik o nazwie name w notatniku (ew. też tworzy)

np. **notepad README.md – mark down**

* **git config --global** -> ustawia jako gitowy edytor program o nazwie name

**core.editor [name]**

* **git config --global** -> sprawdza jaki program jest ustawiony jako gitowy edytor

**core.editor** (defaultowo VI editor)

* **git config --global** -> lista wszystkich ustawień globalnych dla gita

**--list**

* **git config --global -e** -> otwiera listę wszystkich ustawień globalnych dla gita
* **git init [name]** -> utworzenie pustego repozytorium gitowego (katalog)

o nazwie name, w bieżącej lokalizacji

* **git remote add origin** **[path]** -> przypisanie ropo lokalnego jako origin do adresu [path] repo na serwerze, które utworzyliśmy jako miesjsce dla wypushowania plików z repo lokalnego – POŁACZENIE LOCAL REPOSITORY WITH REMOTE REPOSITORY

**np**. git remote add origin <https://github.com/DorotaJankowiec/local-demo-g-h.git>

**origin** – name of the remote reference

* **git push -u origin master** -> wypchnięcie repo lokalnego do repo na serwerze

origin (wszystkiego) na branch master

**-u** – utworzenie (bo nie istnieje na serwerze)

**ATOM editor**

**Atom** - the graphical text editor. (better that Notepad++ ?)

atom.io - > download

GitBash -> insert: „atom” -> opening

* **git config --global** -> przypisanie atom’a jako domyślny

**core.editor „atom --wait"** edytor gitowy

* **git config --global --list**
* **git congig --global -e** -> opens config file by git editor
* **atom .** -> otwiera się atom dla obecnej lokalizacji
* **atom [name]** -> otwiera się atom dla folderu name z obecnej lokalizacji
* **atom [name1] [name2]** -> otwiera się atom dla kilku folderów z obecnej

lokalizacji

**SSH key**

Autoryzacja za pomocą SSH, zamiast http.

* GitHub

-> My account -> Settings -> SSH and GPG keys -> new SSH

* GitBash

-> check if ssh key does not exist -> ls -al (from user’s home directory – default) -> check if ssh file exists (zaczyna się nazwa od kropki: .ssh folder)

-> if no then -> **mkdir .ssh** -> cd .ssh **-> ssh-keygen -t rsa -b 4096 -C “[emailAddress]”**

-> accept by enter -> ls -al -> notepad/atom id\_rsa.pub -> copy content of file

* GitHub

-> My account -> Settings -> SSH and GPG keys -> set key title -> in key field paste copied content (starts with ssh-rsa…) -> add SSH key

* Test SSH connection to GitHub:

GitBash -> **ssh -T git@github.com** -> enter

CLONE REPO USING SSH

* GitHub -> clone or download -> use SSH -> copy address / path
* GitBash

-> cd [repository folder path]

-> **rm -rf [repoFolderName]**  - usuwa repo lokalnie

-> **git clone [copied path from GitHub] [folder name]**

-> cd [folder name]

-> **git remote -v**

or

-> cd [repoFolderName]

-> git remote -v

-> **git remote rm origin** – usunięcie powiązania repo na dysku z repo na serwerze

-> git remote -v

-> **git remote add origin [copied path from GitHub]** – przypisanie repo na serwerze

-> git remote -v

-> git pull origin master

NEW REPO WITH EXISTING CONTENT

* GitHub -> create new repo -> … - > create - > set SSH
* GitBash

-> cd [folder path to become future repo, with existing content inside]

-> **git init [whitespace] . -> “.” (dot) represents current folder**

-> git status

-> **git add .**  ⬄ **git add -A** -> add all files and folders to staging area

-> **git commit -m “message”**

-> **git remote -v**

-> **git remote add origin [SSH/HTTP]** -> associate with github repo on github

-> **git push (-u origin master)**

-> **git rm [name.ex]** -> usuwa plik

-> **git mv [name1.ex] [name2.ex]** -> zmień nazwę pliku name1 z bieżącej

lokalizacji na name2

GIT LOGS

* GitBash

-> **git log** -> list of commits + their numbers,

aktualny branch.

-> **git log -- [name.ex]** -> list of commits for file name.ex

-> **git log -- follow -- [name.ex]** -> list of commits for file name.ex dla

pliku, któego nazwa była zmieniana,

a git to widzi jako usunięcie i dodanie

-> **git help log** -> available options

-> **git log --oneline --all** -> wszystkie logi w jednej linii

**--decorate --graph**

ALIASES

* GitBash

-> **git config --global** -> alias dla gitowej komendy

**alias.[aliasName]**

**“[git command”]**

* **Np. git config --global alias.hist “log --oneline --all --decorate –graph”**
* GitBash

-> cd [user’s home directory = default directory]

-> notepad **.bash\_profile**

From editor: np:

Line 1: alias ll = ‘ls -al’

Line 2: alias ghist = ‘… restart GitBash!

IGNORE UNWANTED FILES

* GitBash

-> notepad **.gitignore**

From editor: np:

Line 1: app.log lub \*.log // ignore file or all files with ex. .log

Line 2: /target // ignore all directory

-> git add .gitignore //adding .gitignore file to staging area

-> git commit -m “…”

BACKOUT CHANGES

* GitBash

-> git add . -> add all untracked to staging area

**-> git reset HEAD -- [filename.ex]** -> **unstage** file: filename.ex

**-> git checkout -- [filename.ex]** -> **dicard changes for file**

**COMPARE**

**Diff** and **Merge Tool**

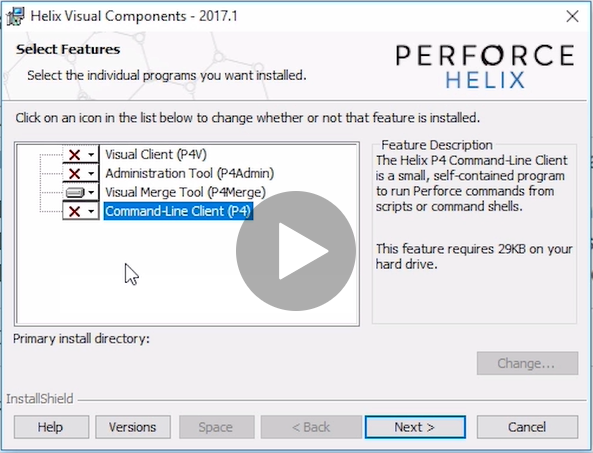
* **HELIX P4V (P4 + P4Merge + P4Admin)**

-> www.perforce.com

-> Support -> Down. soft. -> Helix Desktop& Web Apps-> HELIX P4V VISUAL CLIENT

-> download

-> install with settings: !!!!!!!!!!!!!!!!



* + **git diff** ->Diffing is a function that takes two input data sets

and outputs the changes between them

* + **git merge** ->……………

Configuration P4Merge with GIT:

-> GitBash home directory

-> git config --global --list

-> git config --global diff.tool p4merge

-> git config --global difftool.p4merge.path

“C:/Program Files/Perforce/p4merge.exe”

-> git config --global difftool.prompt false

-> git config --global --list

and

-> git config --global merge.tool p4merge

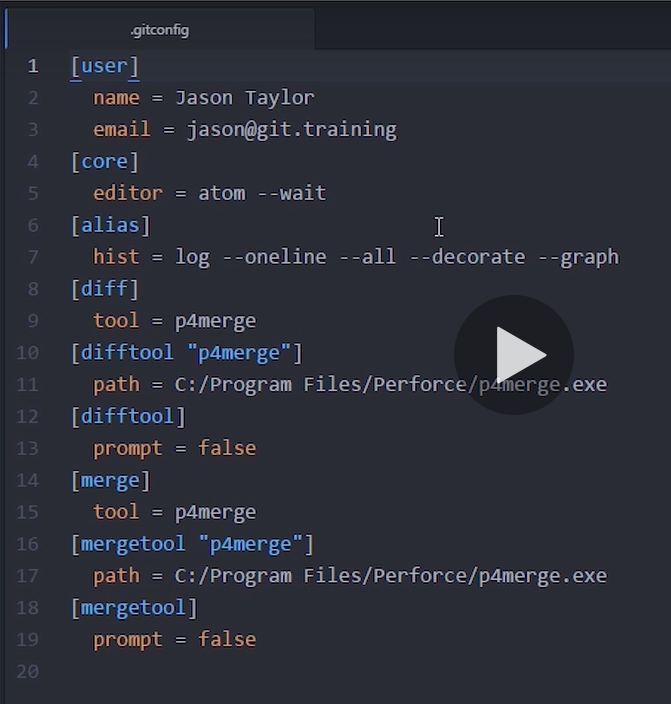
-> git config --global mergetool.p4merge.path

“C:/Program Files/Perforce/p4merge.exe”

-> git config --global mergetool.prompt false

-> git config --global --list

-> git config --global -e // opens git config file in core editor



**Example:**

-> Edit file in existing local repository which is connected with remote repository

-> GitBash

-> **git commit -am “[message]”**

-> git push origin master

-> Edit file again (2nd version)

-> GitBash

-> git commit -am “[message]”

-> Edit file again (3rd version).

-> GitBash

-> **git add** [filename.ex]

-> Edit file again (4rd version).

**Compare changes from the GitBash**

-> GitBash

-> **git diff //internal diff tool ; differences between files in**

**staging area vs files in working area**

-> git hist // commits history (ALIAS!)

-> **git diff HEAD //internal diff tool ; differences between files in**

**working area vs version from last commit**

-> **git diff --cached HEAD //internal diff tool ; differences between files in**

**staging area vs version from last commit**

**Compare changes from the P4Merge program**

-> GitBash

-> cd [repo directory]

**-> git difftool -> P4Merge will open**

**compare working area ver. with staging area ver.**

**-> git difftool HEAD -> P4Merge will open**

**compare working area ver. with last commit ver.**

**-> git difftool --cachedHEAD -> P4Merge will open**

**compare staging area ver. with last commit ver.**

**Compare changes from the P4Merge program – another commands**

-> GitBash

-> cd [repo directory]

-> git hist // commits history (ALIAS!)

-> **git diff(tool) HEAD^** // work.area. ver. with 2nd to last commit

-> **git diff(tool) HEAD^^** // work.area. with 3rd to last commit

-> **git diff(tool) HEAD~4** // work.area. with 5th to last commit

-> **git diff(tool) --cached HEAD~4** // stag.area. ver. with 5th to last commit

-> **git diff(tool) HEAD HEAD~2** // last local commit ver. with 2nd to last

commit

(HEAD – last commit with current changes from working area ?)

-> GitBash

-> cd [repo directory]

-> git hist // commits history (ALIAS!)

-> **git show HEAD** // last commit vs previous commit

-> **git show HEAD^** // ………………..

-> **git show HEAD~2** // ………………..

-> **git show [commitNumber]**  // ………………..

-> GitBash

-> cd [repo directory]

-> git hist // commits history (ALIAS!)

-> **git diff(tool) [commitNr]**  // work.area. ver. with commitNr ver.

-> **git diff(tool) [commitNr1]** **[commitNr2]** // commitNr1 ver. with commitNr2 ver.

-> **git diff(tool) --cached [commitNr]** // stag.area. ver. with commitNr ver.

-> **git help diff**  // all options for diff command

**Compare local changes with remote repository version**

-> GitBash

-> cd [repo directory]

-> git hist // commits history (ALIAS!)

-> **git fetch --al** // pull = fetch + merge into local repo

-> **git branch -a** // list of branches (local + remotes)

-> **git diff(tool) (remotes/)origin/master** // work.area. ver. with remote repo ver.

-> **git diff(tool) --cached origin/master** // stag.area. ver. with remote repo ver.

-> **git diff(tool) HEAD origin/master** // last local commit ver. with remote repo

-> **git diff(tool) [commitNr] origin/master** // local commit Nr ver. with remote repo

1. **.get/config - global configuration file** [↑](#footnote-ref-1)