**Git**

**dir** #pokazać wszystkie

**mkdir** #utworzyć nowy folder

**cd** #przejść do folderu

**git clone *link*** # skopiować folder z githuba

**git init “name rep”** #tworzy nowy rep (włączenie gita dla projektu)

**git pull** #spragit wdzanie i aktualizacja brancha

**git pull -r** łączy commity w jeden

**git branch *namebranch*** *# nowy branch*

**git checkout *name #***przejść na innego brancha

**git checkout -b *name #***przejść na innego brancha

**git branch -a** #pokazać branche

**git branch -d** ***nazwa*** #usunąć branchol

**git status**

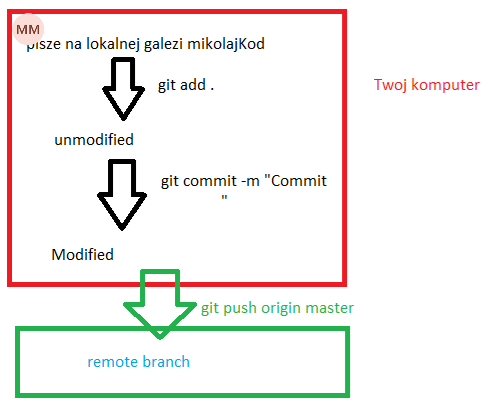
**git add .**  # dodać pliki do przed pokoju przed commitem

**git commit -m** “komentarz” # zrobić commit

**git log** #sprawdzić commity

**git push --set-upstream origin misha**

**git merge master #połączyć zmiany z mastera**



1. git init - zainiciuj git w folderze, gdzie chcesz miec repo  
2. git clone url - sciagnij repo z githuba  
3. git remote add origin https - dajesz remota  
4. git branch testGalaz

5. git checkout testGalaz - przechodzi na galaz  
6. git add . - dodajesz na I stan  
7. git commit -m - dodajesz commit  
8. git push - pushujesz na galaz

**Coursera**

**git diff *filename*** – wyświetla zmian wprowadzone do pliku (+ to co zostało dodane, - to co było)

**git log** - sprawdza commity

**cat *filename*** - pokazuje zawartość pliku

**touch *filename*** *–* tworzenie pliku

**diff *filename1 filename2*** – pokazuje różnice między plikami (diff -u, wdiff)

**diff -u *disk\_usage\_original.py* *disk\_usage\_fixed.py* > *disk usage diff*** -po naprawie oryginalnego pliku tworzymy plik diff pokazujący róznice między dwoma plikami. By zaktualizować plik możemy użyć **patch**:

**patch *disk\_usage.py<disk\_usage.diff*** wprowadzenie zmian z

**git commit -m** “komentarz” # zrobić commit (**-a** – pomija krok add)

**echo *text* > *filename*** – zapisuje tekst do pliku (przydatne do tworzenia gitignore)

**git log -p –** tworzy tekst ze zmianami w patch’u

**git log –-stat** - statystyki: które pliki zostały zmienione, ile wierszy zostało usuniętych/zmienionych

**git log --graph –oneline –** graph rysuje graf z branchami, oneline – jedna linia o commicie

**git show –** pokazuje różne obiekty

**git show *commithashnumber***– info o commicie (wystarczy kilka pierwszych znaków, 5-8)

**git diff** - pokazuje różnice w różnych commitach (podobny do linuxowego diff)

**git diff –staged** – pokaże wszystkie pliki w stage’u w porównaniu z nazwanym zatwierdzeniem (używany przed **git commit**)

**git add -p** – interaktywne przeglądanie zmian, które można dodać do biężącego commita

**git mv** – przenosi plik

**git rm** - usuwa plik

**git checkout** ***filename –*** odrzuca zmiany w working directory

**git reset HEAD *filename* –** odrzuć zmiany ze stage area (przeciwieństwo do add)

**git commit –-amend –** nadpisuje ostatni lokalny commit. Otwiera się atom i opisujesz jeszcze raz commit’a

**git revert HEAD** – tworzy commit’a odwracając zmiany ze złego OSTATNIEGO commita usuwając je

**git revert *commithashnumber*** – tworzy commit’a na podstawie commita podawanego w argumencie (wystarczy kilka pierwszych znaków)

**git branch *branchname***– tworzy nowy branch

**git checkout *branchname -*** przejść na innego brancha (następuje update working tree)

**git branch -a**– pokazuje wszystkie branche

**git checkout -b *branchname***– tworzy i przechodzi na nowy branch

**git branch -d** ***branchname*** - usuwa brancha

**git branch -D *branchname*** - przymusowo usuwa brancha

**git merge *branchname*** - łączy branch, na którym jesteśmy z branchem, który podajemy w branchname’m

**git merge –-abort** – przerywa scalenie, reset plików do poprzedniego commita przed mergowaniem

**git clone *lepolink* -** skopiować repo z githuba

**git push** – wypychanie lokalnego repo do remote repo

git pull – aktualizacja lokalnego repo z remote repo (GitHub’a) na gałęzi, na której się znajduje

**git config --global credential.helper** - allowing automated login to GitHub

**git remote -v –** wyświetla konfiguracje remote repo

**git remote show origin** – więcej info o konfiguracji remote repo

**git remote update** – pobiera remote uaktualnienia repo, ale nie merguje ich,

**git branch -r** – pokaż branche, które trackuje

**git fetch** – pobiera remote uaktualnienia repo na gałęzi, na której jestem, ale nie merguje ich, pokazuje co się zmieniło

**git checkout -b *branchname -*** przejść na innego brancha (jeśli go nie ma, zakłada tego nowego branch’a)

**git push -u origin** – (**-u** tworzy gałąź odnoszącą się do remote gałęzi powyżej, **origin** – wypychamy gałąź do origin repo)

**git rebase** ***aktualnybranch –*** komendę wykonujemy będąć w branchu, który nie ma wspólnej bazy z aktualnym branchem po to aby można te oba branche zmergować. Umieszcza branch na którym się znajdujemy pod tym aktualnym branchem. Ehhhh...

**git push --delete origin *namebranch*** *–* usuwa gałąź