

Lekcja 1 – Markdown lekki język znaczników

Spis treści

Lekcja 1 – Markdown lekki język znaczników.....	1
Wstęp.....	1
Podstawy składni.....	3
Definiowanie nagłówków.....	3
Definiowanie list.....	4
Wyróżnianie tekstu.....	4
Tabele.....	5
Odnosińiki do zasobów.....	5
Obrazki.....	5
Kod źródłowy dla różnych języków programowania.....	5
Tworzenie spisu treści na podstawie nagłówków.....	6
Edytory dedykowane.....	7
Pandoc – system do konwersji dokumentów Markdown do innych formatów.....	8
Lekcja 2 – Git – system kontroli wersji.....	9
Git - podstawowe cechy.....	9
Idea pracy:.....	9
Git – tworzenie pustego archiwum lokalnego.....	11
Zadania do wykonania na punkty.....	21
Zadanie 1 – 2pkt.....	21
Zadanie 2 – 4pkt.....	21
Zadanie 3 - 4pkt.....	21

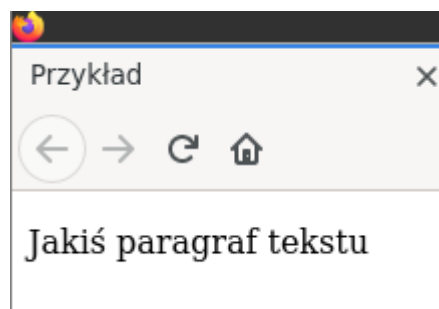
Wstęp

Obecnie powszechnie wykorzystuje się języki ze znacznikami do opisanie dodatkowych informacji umieszczanych w plikach tekstowych. Z pośród najbardziej popularnych można wspomnieć o:

1. **html** – służącym do opisu struktury informacji zawartych na stronach internetowych,
2. **Tex** (Latex) – poznany na zajęciach język do „profesjonalnego” składania tekstów,
3. **XML** (*Extensible Markup Language*) - uniwersalnym języku znaczników przeznaczonym do reprezentowania różnych danych w ustrukturalizowany sposób.

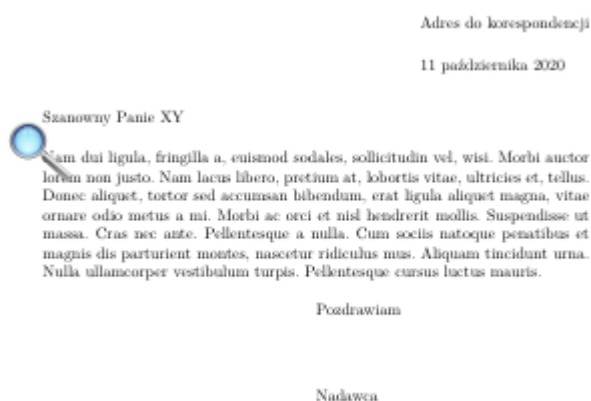
Przykład kodu *html* i jego interpretacja w przeglądarce:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Przykład</title>
</head>
<body>
<p> Jakiś paragraf tekstu</p>
</body>
</html>
```



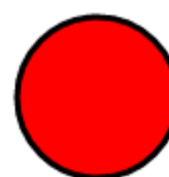
Przykład kodu *Latex* i wygenerowanego pliku w formacie *pdf*

```
\documentclass[] {letter}
\usepackage{lipsum}
\usepackage{polyglossia}
\setmainlanguage{polish}
\begin{document}
\begin{letter} {Szanowny Panie XY}
\address{Adres do korespondencji}
\opening{}
\lipsum[2]
\signature{Nadawca}
\closing{Pozdrawiam}
\end{letter}
\end{document}
```



Przykład kodu *XML* – fragment dokumentu *SVG* (Scalar Vector Graphics)

```
<!DOCTYPE html>
<html>
<body>
<svg height="100" width="100">
<circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
</svg>
</body>
</html>
```



W tym przypadku mamy np. znacznik np. `<circle>` opisujący parametry koła i który może być właściwie zinterpretowany przez dedykowaną aplikację (np. przeglądarki www).

Jako ciekawostkę można podać fakt, że również pakiet MS Office wykorzystuje format XML do przechowywania informacji o dodatkowych parametrach formatowania danych. Na przykład pliki z rozszerzeniem *docx*, to nic innego jak spakowane algorytmem zip katalogi z plikami xml.

Przykład rozpakowania zawartości pliku *test.docx* poleceniem: **unzip**

```
$unzip -l test.docx
```

Archive: test.docx

Length	Date	Time	Name
573	2022-03-20	08:55	_rels/.rels
731	2022-03-20	08:55	docProps/core.xml
508	2022-03-20	08:55	docProps/app.xml
531	2022-03-20	08:55	word/_rels/document.xml.rels
1288	2022-03-20	08:55	word/document.xml
2429	2022-03-20	08:55	word/styles.xml
853	2022-03-20	08:55	word/fontTable.xml
257	2022-03-20	08:55	word/settings.xml
1374	2022-03-20	08:55	[Content_Types].xml

Wszystkie te języki znaczników cechują się rozbudowaną i złożoną składnią i dlatego do ich edycji wymagają najczęściej dedykowanych narzędzi w postaci specjalizowanych edytorów. By wyeliminować powyższą niedogodność powstał **Markdown** - uproszczony język znaczników służący do formatowania dokumentów tekstowych (bez konieczności używania specjalizowanych narzędzi). Dokumenty w tym formacie można bardzo łatwo konwertować do wielu innych formatów: np. html, pdf, ps (postscript), epub, xml i wiele innych. Format ten jest powszechnie używany do tworzenia plików README.md (w projektach open source) i powszechnie obsługiwany przez serwery git'a. Język ten został stworzony w 2004 r. a jego twórcami byli John Gruber i Aaron Swartz. W kolejnych latach podjęto prace w celu stworzenia standardu rozwiązania i tak w 2016 r. opublikowano dokument [RFC 7764](#) który zawiera opis kilku odmian tegoż języka:

- CommonMark,
- GitHub Flavored Markdown (GFM),
- Markdown Extra.

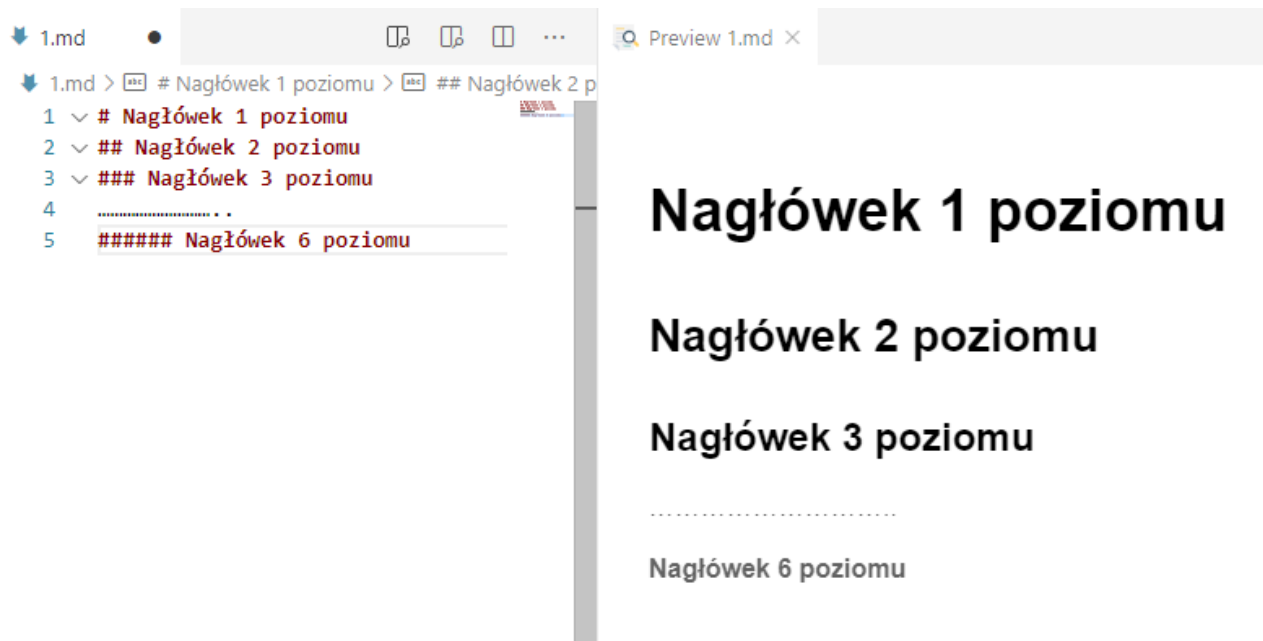
Podstawy składni

Podany link: <https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet> zawiera opis podstawowych elementów składni w języku angielskim. Poniżej zostanie przedstawiony ich krótki opis w języku polskim.

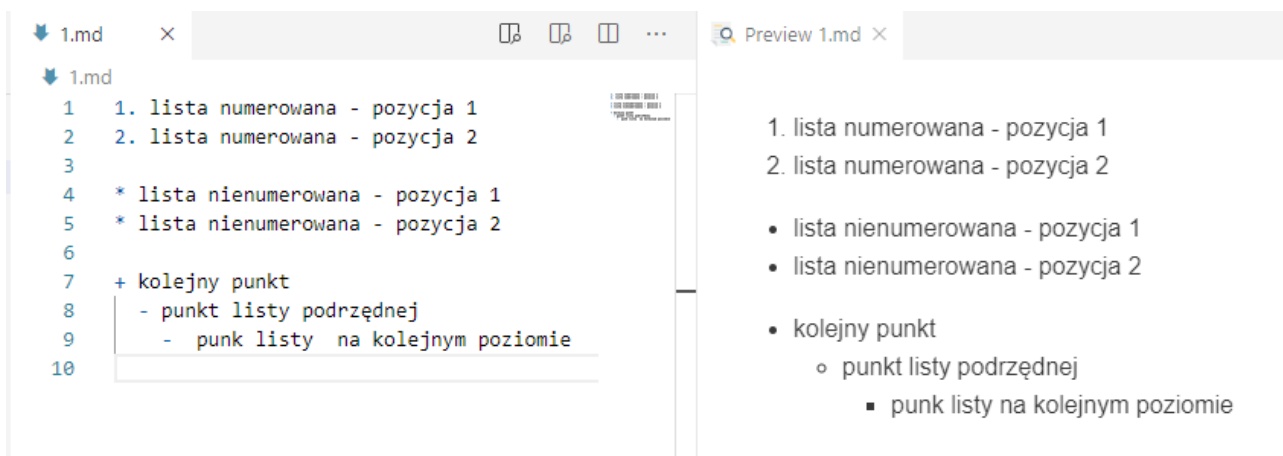
Definiowanie nagłówków

W tym celu używamy znaku kratki

Lewe okno zawiera kod źródłowy – prawe -podgląd przetworzonego tekstu



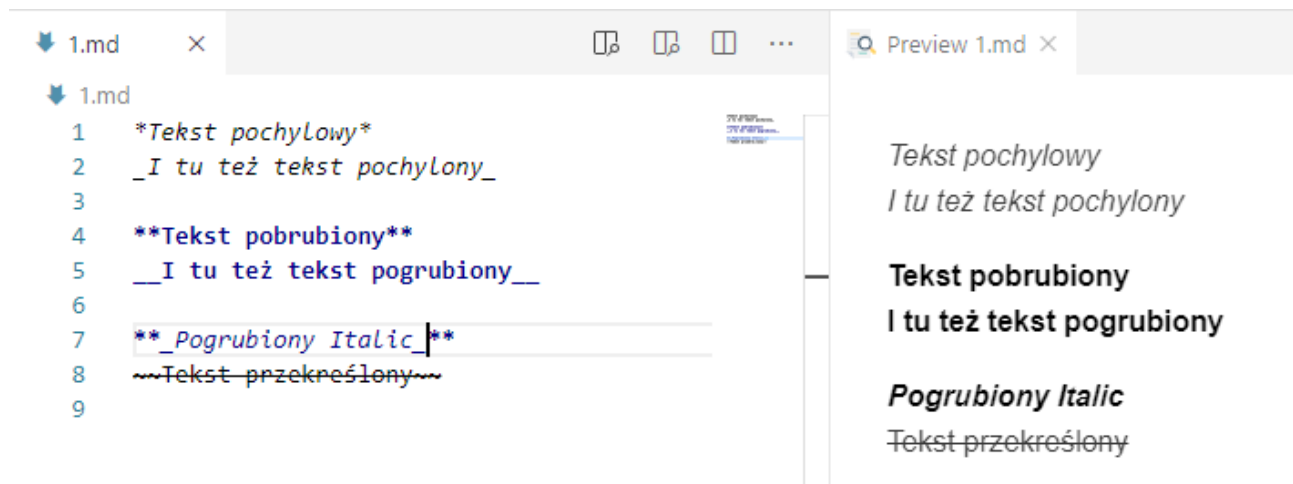
Definiowanie list



Listy numerowane definiujemy wstawiając numery kolejnych pozycji zakończone kropką.

Listy nienumerowane definiujemy znakami: *,+,-

Wyróżnianie tekstu



Tabele

The screenshot shows a code editor with a file named '1.md' and a preview window. The code in the editor is as follows:

```
1  | Wycentrowana | Do lewej | Do prawej |
2  | :-----: | :-----: | :-----: |
3  | 1 | 4 | 7 |
4  | 2 | 5 | 8 |
5  | 3 | 6 | 9 |
```

The preview window shows the rendered table:

Wycentrowana	Do lewej	Do prawej
1	4	7
2	5	8
3	6	9

Centrowanie zawartości kolumn realizowane jest poprzez odpowiednie użycie znaku dwukropka:

Odnośniki do zasobów

[odnośnik do zasobów](www.gazeta.pl)

[odnośnik do pliku](LICENSE.md)

[odnośnik do kolejnego zasobu][1]

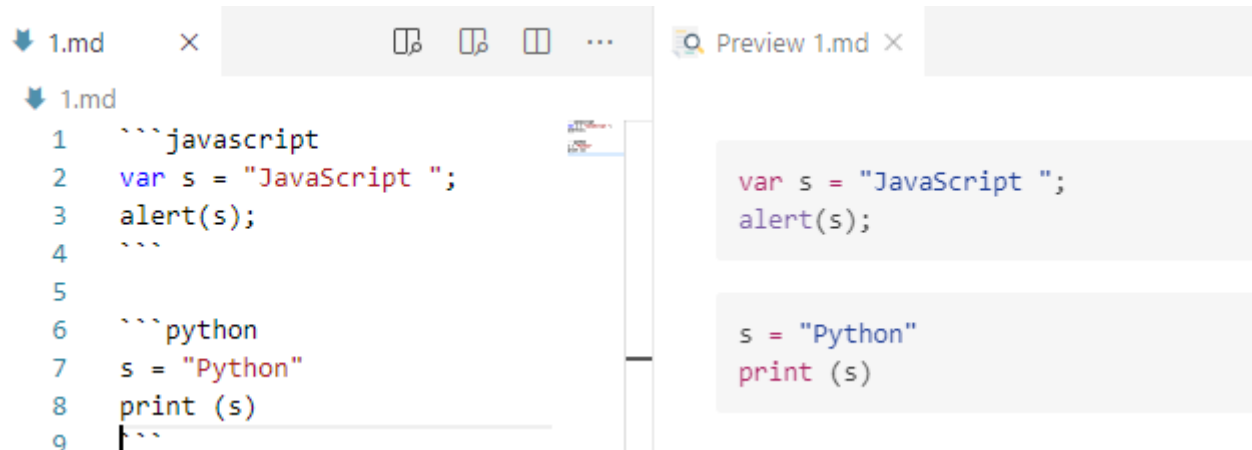
[1]: <http://google.com>

Obrazki

![alt text](<https://server.com/images/icon48.png> "Logo 1") – obrazek z zasobów internetowych

 – obraz z lokalnych zasobów

Kod źródłowy dla różnych języków programowania

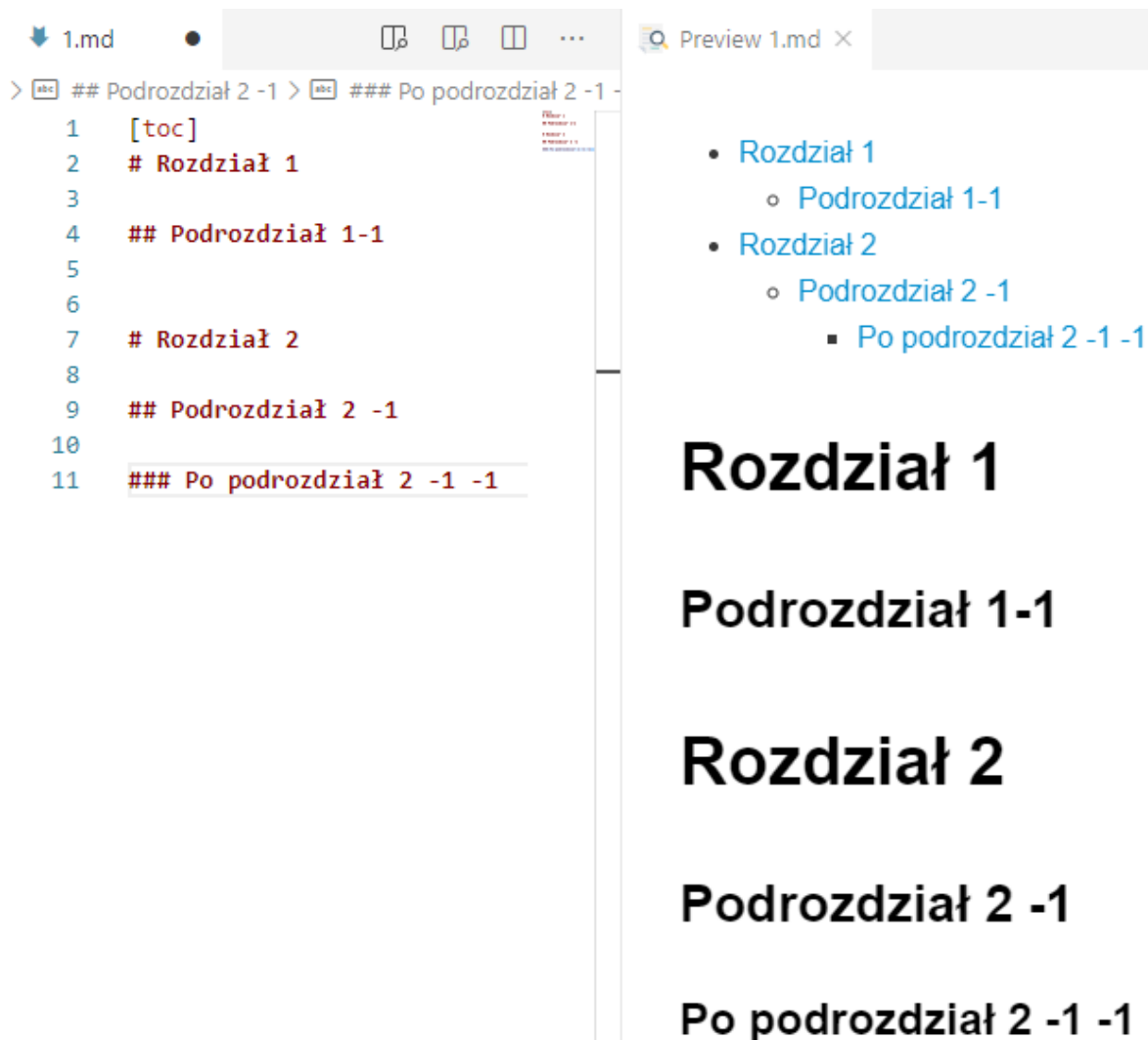


```
1  ```javascript
2  var s = "JavaScript ";
3  alert(s);
4  ```
5
6  ```python
7  s = "Python"
8  print (s)
9  ```
```

```
var s = "JavaScript ";
alert(s);

s = "Python"
print (s)
```

Tworzenie spisu treści na podstawie nagłówków



```
> [abc] ## Podrozdział 2 -1 > [abc] ### Po podrozdział 2 -1 -
1  [toc]
2  # Rozdział 1
3
4  ## Podrozdział 1-1
5
6
7  # Rozdział 2
8
9  ## Podrozdział 2 -1
10
11 ### Po podrozdział 2 -1 -1
```

- [Rozdział 1](#)
 - [Podrozdział 1-1](#)
- [Rozdział 2](#)
 - [Podrozdział 2 -1](#)
 - [Po podrozdział 2 -1 -1](#)

Rozdział 1

Podrozdział 1-1

Rozdział 2

Podrozdział 2 -1

Po podrozdział 2 -1 -1

Edytory dedykowane

Pracę nad dokumentami w formacie Markdown(rozszerzenie md) można wykonywać w dowolnym edytorze tekstowym. Aczkolwiek istnieje wiele dedykowanych narzędzi

1. marktext - <https://github.com/marktext/marktext>
2. <https://hackmd.io/> online editor
3. Visual Studio Code z wtyczką „markdown preview”



Pandoc – system do konwersji dokumentów Markdown do innych formatów

Jest oprogramowanie typu open source służące do konwertowania dokumentów pomiędzy różnymi formatami.

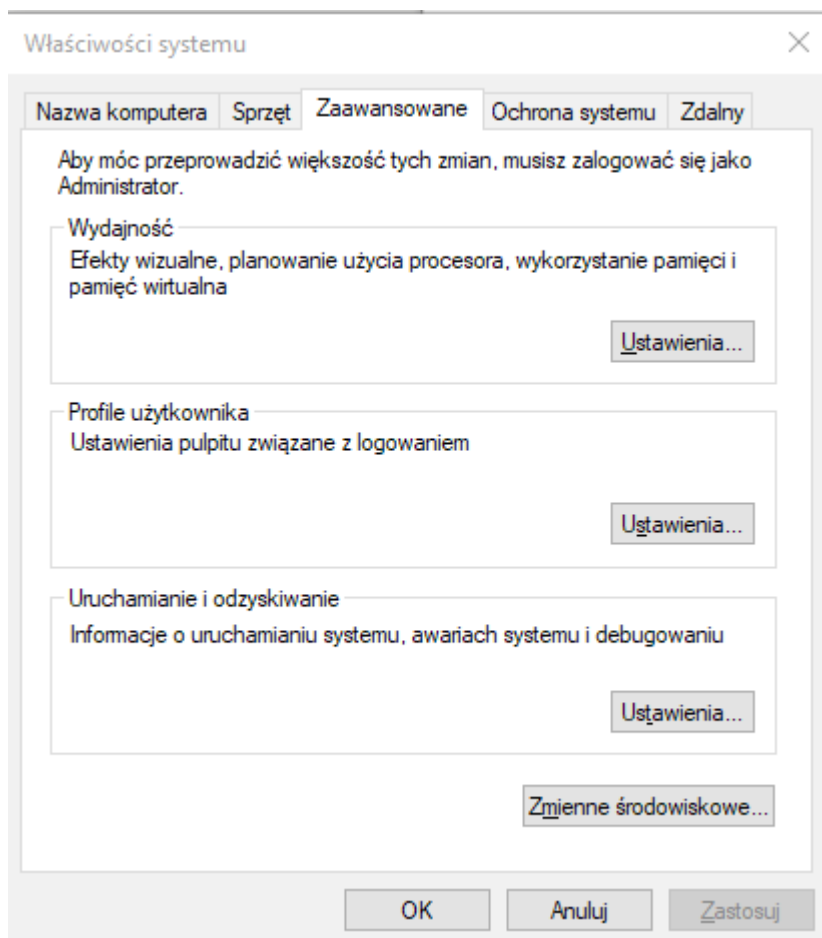
Pod poniższym linkiem można obejrzeć przykłady użycia:

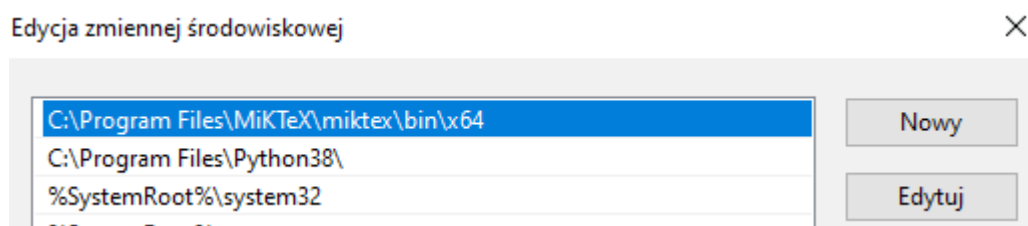
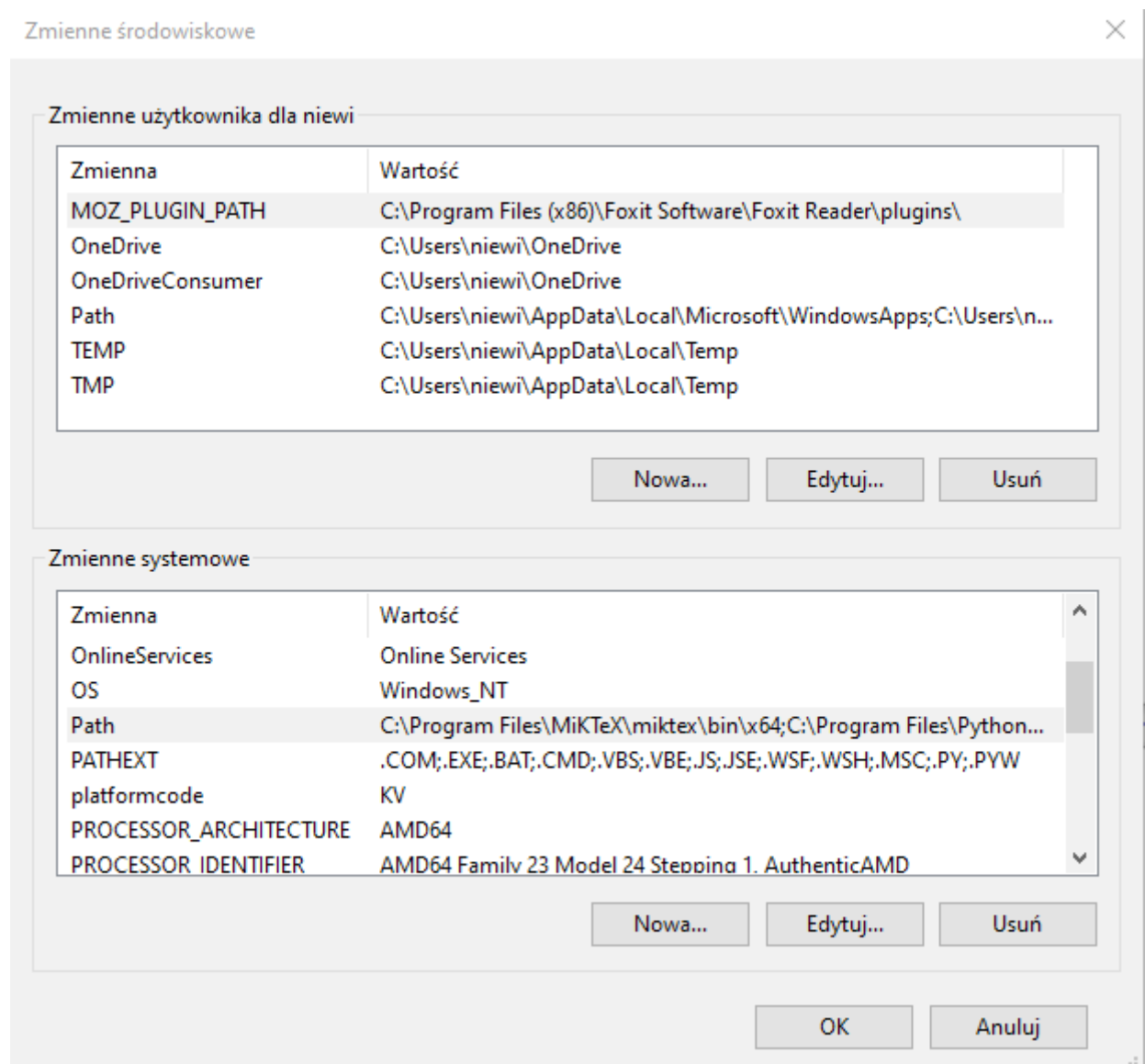
<https://pandoc.org/demos.html>

Oprogramowanie to można pobrać z spod adresu: <https://pandoc.org/installing.html>

Jeżeli chcemy konwertować do formatu latex i pdf trzeba doinstalować oprogramowanie składu Latex (np. Na windows najlepiej sprawdzi się Miktex <https://miktex.org/>)

Gdyby podczas konwersji do formatu pdf pojawił się komunikat o niemożliwości znalezienia programu pdflatex rozwiązaniem jest wskazanie w zmiennej środowiskowej PATH miejsca jego położenia





Pod adresem (<https://gitlab.com/mniewins66/templatemn.git>) znajduje się przykładowy plik Markdown z którego można wygenerować prezentację w formacie pdf wykorzystując klasę latexa beamer.

W tym celu należy wydać polecenie z poziomu terminala:

```
$pandoc templateMN.md -t beamer -o prezentacja.pdf
```

Lekcja 2 – Git – system kontroli wersji

Jest to oprogramowanie służące do śledzenia zmian (głównie w dokumentach tekstowych) podczas pracy zespołów wieloosobowych dokonujących zmian w różnym czasie i miejscach przebywania.

Git - podstawowe cechy

- efektywna praca z dużymi projektami - jest jednym z najszybszych systemów kontroli wersji
- wsparcie dla protokołów sieciowych - dane można wymieniać przez HTTP(S), FTP, rsync, SSH, e-mail
- każda kopia repozytorium to obraz całego projektu - Git nie zapamiętuje zmian między kolejnymi rewizjami lecz kompletne obrazy (snapshots)
- możliwość tworzenia oprogramowania z rozgałęzieniami
- tryb pracy off-line - każdy pracuje na własnej kopii repozytorium, a następnie zmiany mogą być wymieniane między lokalnymi repozytoriami jak również serwerem.

Idea pracy:

Na powyższym rysunku symbolicznie przedstawiono zasadę pracy z gitem. Większy okrąg symbolizuje repozytorium lokalne (po lewej) w którym pracuje użytkownik. Mniejszy okrąg (po prawej) to kopia repozytorium na wybranym serwerze (np. github.com lub gitlab.com). Czerwone strzałki to komendy git'a, które powodują przenoszenie plików i/lub katalogów pomiędzy poszczególnymi elementami repozytorium lub pomiędzy maszyną lokalną a serwerem.

Git rozróżnia trzy typy plików w repozytorium lokalnym: nadzorowane, pomijane i nienadzorowane. Z punktu widzenia systemu plików repozytorium to zwykły katalog w którym zaicjalizowany został specjalny podkatalog o nazwie `.git`, w którym przetrzymywane są wszystkie informacje o repozytorium.

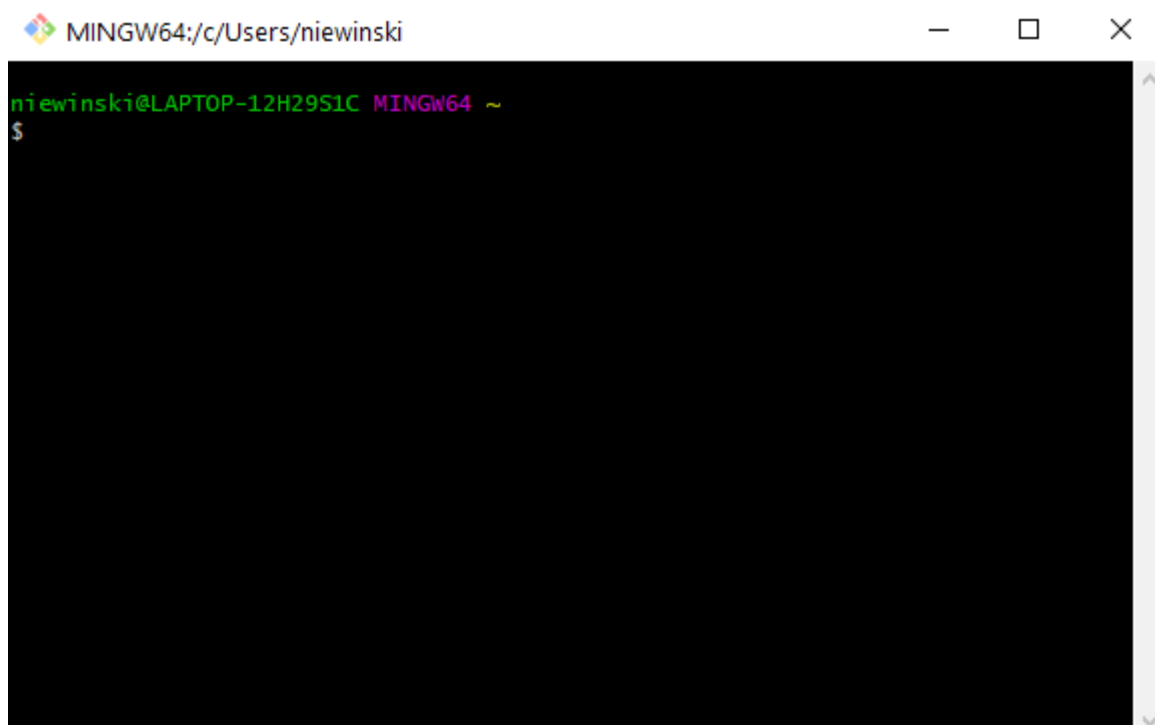
Użytkownik pracuje w katalogu roboczym, gdzie modyfikuje swoje pliki. Gdy uzna, że postęp prac wymaga zapisania zmian przenosi je do *staging area* (komenda `add`). Gdy suma zmian stanowi jakąś spójną całość (np. dodanie nowego rozdziału pracy, dodanie nowej funkcjonalności itd.) tworzy się nową „migawkę” zapisywaną w lokalnym repozytorium (komenda `commit`). Co jakiś czas można przenieść lokalne zmiany w repozytorium na serwer (komenda `push`).

Uwaga: Do pracy z systemem git należy zainstalować oprogramowanie ze strony:

<https://git-scm.com/downloads>

Po zainstalowaniu będzie dostępna powłoka z wyglądu podobna do tej z poniższego rysunku

Jest to aplikacja **Git Bash** w której wydajemy polecenia z linii komend.



Pierwszym krokiem po instalacji git'a i uruchomieniu aplikacji Git Bash jest wydanie dwóch poleceń:

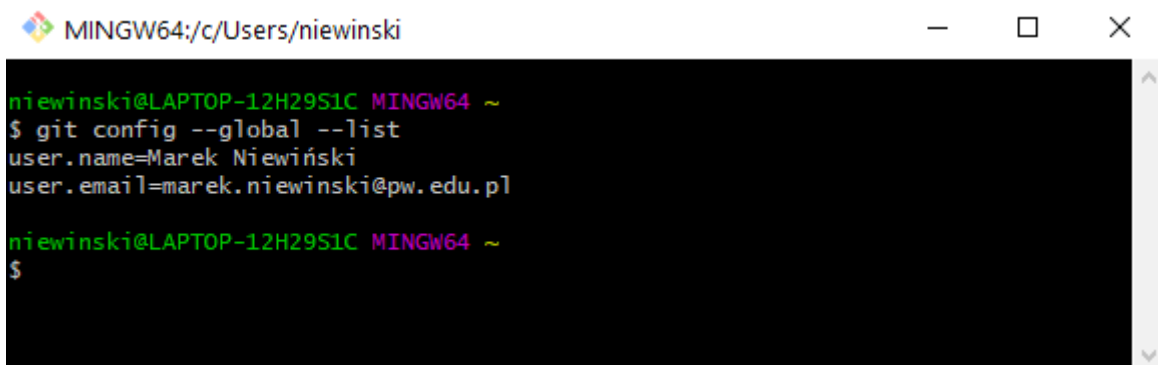
\$git config --global user.name "Student Wspaniały"

\$git config --global user.email "wspaniały@pw.edu.pl"

W ten sposób informujemy system git kto będzie autorem zmian wprowadzanych do repozytorium

Uwaga: należy wprowadzić własne dane osobowe i własny adres e-mail

W przypadku piszącego tą instrukcję wynik wprowadzonych komend ma postać:



```
MINGW64:/c/Users/niewinski
niewinski@LAPTOP-12H29S1C MINGW64 ~
$ git config --global --list
user.name=Marek Niewiński
user.email=marek.niewinski@pw.edu.pl
niewinski@LAPTOP-12H29S1C MINGW64 ~
$
```

Git – tworzenie pustego archiwum lokalnego.

Należy w konsoli wywołać następującą sekwencję komend (zakładając, że repozytorium będzie nosiło nazwę repo1):

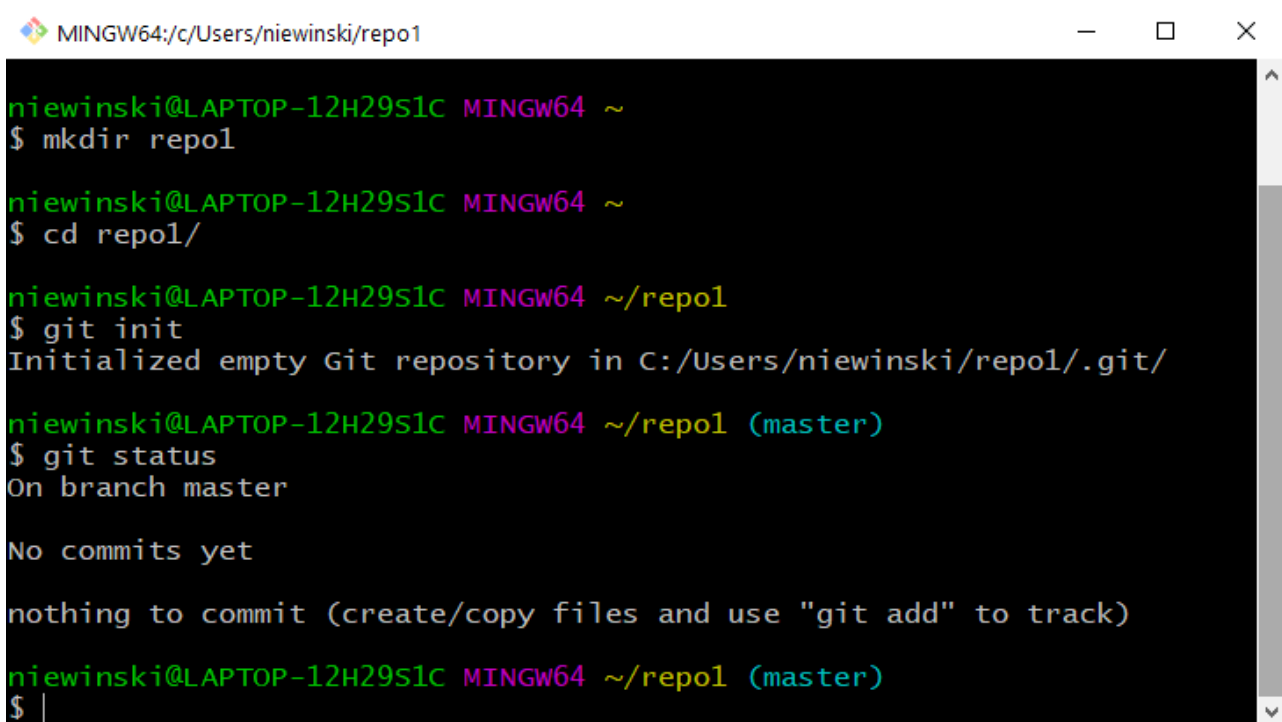
1. \$mkdir repo1
2. \$cd repo1
3. \$git init
4. \$git status

Pierwsza komenda tworzy katalog o zadanej nazwie.

Druga komenda powoduje, że przechodzimy do właśnie stworzonego katalogu

Trzecia komenda inicjuje puste repozytorium

Czwarta komenda wyświetla informacje o stanie repozytorium

A terminal window titled 'MINGW64:/c/Users/niewinski/repo1' showing the steps to create a new Git repository. The user runs 'mkdir repo1', 'cd repo1/', 'git init', and 'git status'. The output shows the repository is initialized and currently on the 'master' branch with no commits yet.

```
MINGW64:/c/Users/niewinski/repo1

niewinski@LAPTOP-12H29S1C MINGW64 ~
$ mkdir repo1

niewinski@LAPTOP-12H29S1C MINGW64 ~
$ cd repo1/

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1
$ git init
Initialized empty Git repository in C:/Users/niewinski/repo1/.git/

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git status
On branch master

No commits yet

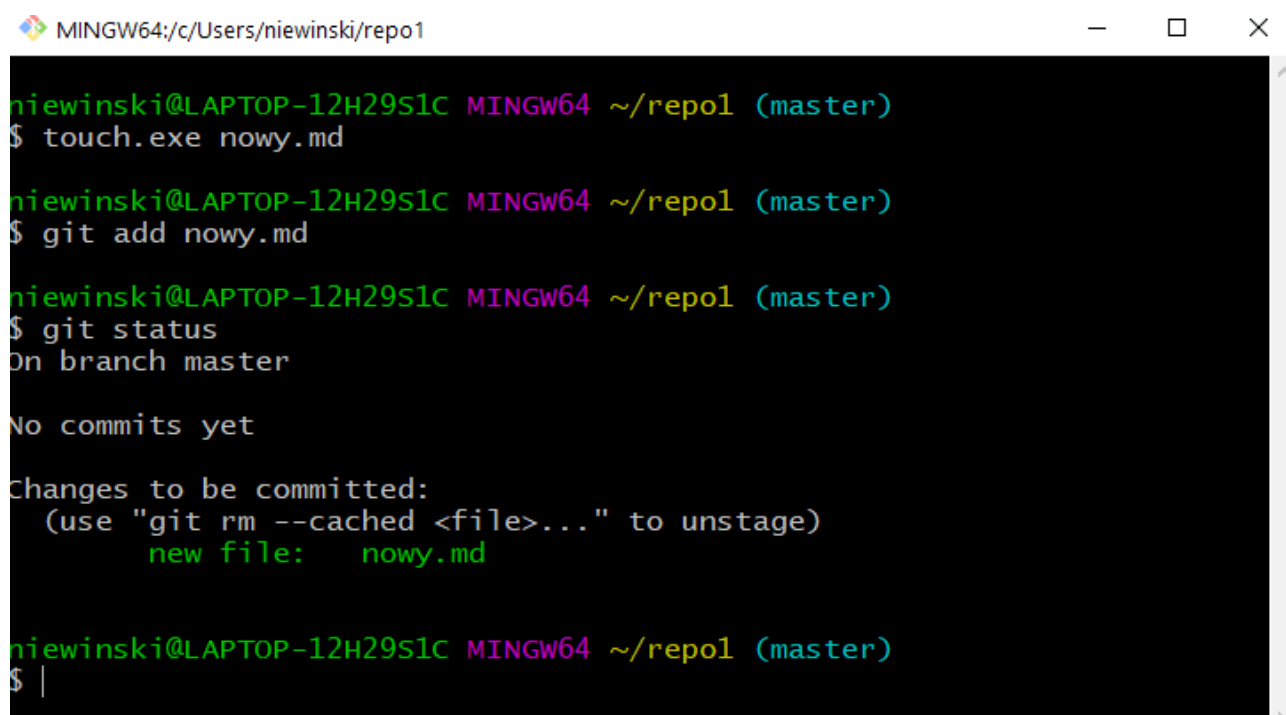
nothing to commit (create/copy files and use "git add" to track)

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ |
```

Jak widać na powyższym rysunku po stworzeniu pustego repozytorium pracujemy w głównej gałęzi **master** i brak jest zarejestrowanych jakichkolwiek „migawek” (commits)

Dodawanie pliku/ów do indeksu:

- \$git add file - dodanie pliku do indeksu
- \$git rm --cached file - usunięcie pliku z indeksu

A terminal window titled 'MINGW64:/c/Users/niewinski/repo1' showing the process of adding a new file to the repository. The user creates a file 'nowy.md' with 'touch.exe', then runs 'git add nowy.md'. The 'git status' command shows the file is staged for commit.

```
MINGW64:/c/Users/niewinski/repo1

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ touch.exe nowy.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git add nowy.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   nowy.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ |
```

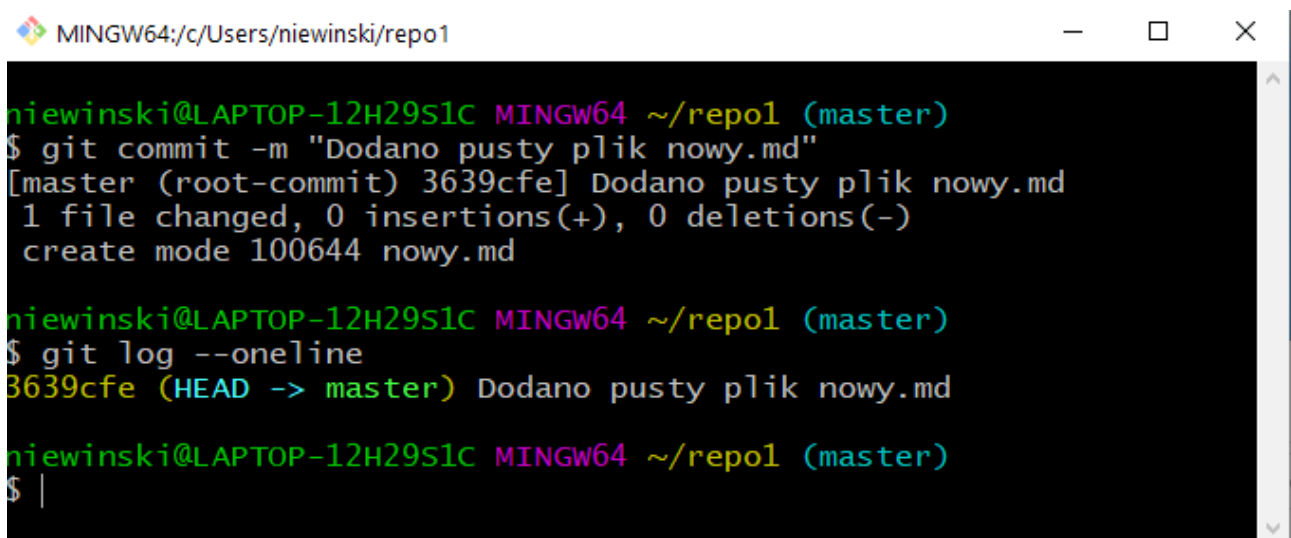
Powyższy rysunek prezentuje sekwencję komend:

1. `$touch.exe nowy.md` – utworzenie pustego pliku tekstowego
2. `$git add nowy.md` – dodanie nowo stworzonego pliku do indeksu
3. `$git status` – wyświetlenie staty lokalnego repozytorium

Widać (zielony kolor), że git śledzi wprowadzone zmiany

Git - zapis pliku do repozytorium lokalnego:

- `$git commit -m "komunikat"` – w komunikacie podaje się krótki opis zmian wprowadzonych do repozytorium



```
MINGW64:/c/Users/niewinski/repo1
niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git commit -m "Dodano pusty plik nowy.md"
[master (root-commit) 3639cfe] Dodano pusty plik nowy.md
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 nowy.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git log --oneline
3639cfe (HEAD -> master) Dodano pusty plik nowy.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ |
```

Polecenie `$git log` wyświetla listę zapisanych „migawek”. Każda migawka identyfikowana jest sekwencją liczb szesnastkowych (żółte cyfry) – komentarzem jaki podajemy podczas wywoływania polecenia `$git commit`.

Git – dodanie kolejnych plików, modyfikacja i stworzenie kolejnych „migawek”

Używając tekstowego edytora nano (można użyć własnego ulubionego edytora) – zmodyfikowano zawartość pliku `nowy.md` – co natychmiast zauważył system kontroli wersji (`modified nowy.md`). Następnie stworzono kolejny pusty plik (`nastepny.md`) co również system zasygnalizował jako: (`Untracked files – nastepny.md`). Teraz zostaną wykonane dwa niezależne commity i zostanie wyświetlony kolejny log stanu repozytorium.

Jak widać na kolejnym rysunku w repozytorium zostały umieszczone trzy „migawki” stanu repozytorium wszystkie umieszczone w gałęzi głównej „master”.

```
niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ nano nowy.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   nowy.md

no changes added to commit (use "git add" and/or "git commit -a")

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ touch.exe nastepny.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   nowy.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        nastepny.md

no changes added to commit (use "git add" and/or "git commit -a")

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ |
```

```
niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git add nowy.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git commit -m "Dodano linię tekstu do pliku nowy.md"
[master ec65aeb] Dodano linię tekstu do pliku nowy.md
1 file changed, 1 insertion(+)

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git add nastepny.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git commit -m "Dodano pusty plik nastepny.md"
[master aab9c2c] Dodano pusty plik nastepny.md
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 nastepny.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git log
commit aab9c2ce2c60f3b741c3601ba996b705e46d2e7e (HEAD -> master)
Author: Marek Niewiński <marek.niewinski@pw.edu.pl>
Date:   Wed Oct 21 14:35:53 2020 +0200

    Dodano pusty plik nastepny.md

commit ec65aebf0811e6856af9549cbd71212d686558eb
Author: Marek Niewiński <marek.niewinski@pw.edu.pl>
Date:   Wed Oct 21 14:35:21 2020 +0200

    Dodano linię tekstu do pliku nowy.md

commit 3639cfe5dfc45bb203ee90db1fe14dd0698c4caf
Author: Marek Niewiński <marek.niewinski@pw.edu.pl>
Date:   Wed Oct 21 13:36:02 2020 +0200

    Dodano pusty plik nowy.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$
```


Git – praca z rozgałęzieniami

Bardzo często zachodzi taka sytuacja, że nie chcemy wprowadzać zmian w głównej gałęzi a tylko wykonać jakieś prace testowe i później zdecydować czy dołączyć je do głównej gałęzi lub nie. Do tego służą rozgałęzienia, i operacja ich łączenia.

- `$git branch name` – tworzenie nowego rozgałęzienia
- `$git checkout name` - przełączenie się do innej gałęzi
- `$git branch -D name` - usunięcie rozgałęzienia (trzeba być od niego odłączonym)
- `$ git merge nazwa_gałęzi` – złączenie gałęzi „nazwa_gałęzi” z gałęzią do której jesteśmy podłączeni

```
niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git branch test

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git branch
* master
  test

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git checkout test
Switched to branch 'test'

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (test)
$ touch.exe 3.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (test)
$ git add 3.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (test)
$ git commit -m "Dodano plik 3.md w gałęzi test"
[test c7bd197] Dodano plik 3.md w gałęzi test
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 3.md

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (test)
$ git checkout master
Switched to branch 'master'

niewinski@LAPTOP-12H29S1C MINGW64 ~/repo1 (master)
$ git merge test
Updating aab9c2c..c7bd197
Fast-forward
 3.md | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 3.md
```

Każde lokalne repozytorium możemy umieścić na dedykowanym serwerze protokołu git (możemy także spakować katalog z repozytorium i taką kopię przesłać zainteresowanemu np. E-mailem). Jeżeli chcemy realizować wspólne projekty - nad którymi pracuje wiele osób - to niezbędnym jest wykorzystanie serwera.

Mamy do dyspozycji np.

- GitHub - <https://github.com>
- GitLab - <https://gitlab.com>
- GitLab Wydziałowy - <https://gitlab-stud.elka.pw.edu.pl> (do niego możecie nie mieć Państwo nadanych praw dostępu)

Sugeruje się wykorzystanie serwera gitlab z uwagi na łatwość obsługi.

W tym celu należy wykonać następujące kroki.

1. Zarejestrować się w systemie
2. Stworzyć nowy projekt w którym będziemy chcieli umieścić nasze lokalne repozytorium
3. W lokalnym repozytorium dodać informacje na jaki serwer mają być wysyłane dane

realizuje to komenda:

\$git remote add origin [git@gitlab.com](https://gitlab.com):user/projekt1.git
(jako user nazwa Państwa konta na serwerze)

I teraz można „wypchnąć” lokalne repozytorium na serwer komendą:

\$git push origin master

gdy istnieją dodatkowe rozgałęzienia w repozytorium, to każde musi być „wypchnięte” oddzielną komendą

\$git push origin nazwaGalezi

Gdy chcemy pobrać repozytorium z serwera na jakąś inną lokalną maszynę to wtedy należy wykonać klonowanie zdalnego repozytorium komendą:

\$git clone [git@gitlab.com](https://gitlab.com):user/projekt1.git

Na potrzeby tej instrukcji stworzono projekt o nazwie testowy na koncie gitlab autora instrukcji

Project name

testowy

Project URL

https://gitlab.com/mniewins66/

Project slug

testowy

Want to house several dependent projects under the same namespace? [Create a group](#).

Project description (optional)

Description format


Visibility Level 



 Private

Project access must be granted explicitly to each user. If this project is part of a group, access will be gra



 Public

The project can be accessed without any authentication.



Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing reposit

Create project

Po wciśnięciu przycisku **Create project** zostanie wygenerowany ekran z listą instrukcji które należy wykonać w zależności od kontekstu w jakim chcemy użyć nasz projekt.

The repository for this project is empty

You can get started by cloning the repository or start adding files to it with one of the following options.

[Clone](#)[New file](#)[Add README](#)[Add LICENSE](#)[Add CHANGELOG](#)[Add CONTRIBUTING](#)[Set up CI/CD](#)

Command line instructions

You can also upload existing files from your computer using the instructions below.

Git global setup

```
git config --global user.name "Marek"
git config --global user.email "mniewins66@gmail.com"
```

Create a new repository

```
git clone git@gitlab.com:mniewins66/testowy.git
cd testowy
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Push an existing folder

```
cd existing_folder
git init
git remote add origin git@gitlab.com:mniewins66/testowy.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin git@gitlab.com:mniewins66/testowy.git
git push -u origin --all
git push -u origin --tags
```

Ostatni rysunek pokazuje przesłanie lokalnego repozytorium na serwer gitlab.com

W tym przypadku nie musiano podawać loginu i hasła ponieważ pomiędzy maszyną lokalną a serwerem są uzgodnione pary kluczy: prywatny i publiczny

```
niewinski@LAPTOP-12H29S1C MINGW64 ~/Desktop/repo1 (master)
$ git remote add origin git@gitlab.com:mniewins66/testowy.git

niewinski@LAPTOP-12H29S1C MINGW64 ~/Desktop/repo1 (master)
$ git remote -v
origin  git@gitlab.com:mniewins66/testowy.git (fetch)
origin  git@gitlab.com:mniewins66/testowy.git (push)

niewinski@LAPTOP-12H29S1C MINGW64 ~/Desktop/repo1 (master)
$ git push origin master
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (10/10), 972 bytes | 486.00 KiB/s, done.
Total 10 (delta 1), reused 0 (delta 0), pack-reused 0
To gitlab.com:mniewins66/testowy.git
 * [new branch]      master -> master
```

Zadania do wykonania na punkty

Zadanie 1 – 2 pkt

Po zapoznaniu się z podstawami składni:

<https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>

proszę stworzyć dokument tekstowy **zad_1.md** zawierający następujące elementy:

1. wybrany plik graficzny,
2. listę numerowaną trzech ulubionych książek,
3. listę nienumerowaną wybranych pięciu miast w Polsce,
4. dwa odnośniki do wybranych zasobów internetowych,
5. pojedyncze zdanie zawierające wyrazy wyróżnione w trybach: bold, italic, strikethrough,
6. wybrany wzór matematyczny w formacie *Latex*,
7. dwie linie tekstu z **nagłówkami** poziomu 2 i 3.

Poprawność dokumentu proszę sprawdzić wykorzystując dedykowany edytor np.

<https://dillinger.io/> Jako wynik przesłać plik zad1.md (opcjonalnie plik graficzny użyty w dokumencie).

Zadanie 2 – 4 pkt

Stworzyć lokalne repozytorium git o nazwie **zad_2**.

W repozytorium proszę umieścić plik **instrukcja.md**. Plik ten powinien zawierać wersję rozdziału Lekcja 2 (też instrukcji) ale w formacie Markdown.

Postępy prac – przy tworzeniu wersji dokumentu w formacie *markdown* - rejestrować jako kolejne *commity* (na przykład każdy nowy fragment dokumentu). Podczas oceny będzie brana pod uwagę liczba *commitów* jak i jasność przypisanych im komunikatów (UWAGA: pliki graficzne muszą być także wyodrębnione i umieszczone w repozytorium).

Następnie tak stworzone repozytorium umieścić na serwerze *gitlab* w projekcie o nazwie **zad_2_pti_25Z**. Do projektu jako użytkownika dodać *usera mniewins66* (w roli **Maintainer** i dodatkowo proszę ustawić *Access expiration date* na dzień 31.12.2025 r.).

Jako wynik pracy przesłać link (umieszczony w pliku link_zad_2.md) do projektu na serwerze *gitlab* oraz spakowane repozytorium lokalne.

UWAGA: Repozytorium należy spakować poleceniem:

git bundle create NAZWISKO_IMIE.bundle HEAD main

Wynikowy plik – *NAZWISKO_IMIE.bundle*.

Zadanie 3 – 4pkt

- Pobrać repozytorium <https://gitlab.com/mniewins66/templatemn.git> z szablonem prezentacji.
- Zainstalować właściwe oprogramowanie (pandoc) i podjąć próbę konwersji pliku do formatu *pdf*.
- Stworzyć lokalne repozytorium git o nazwie **zad_3_pti_25Z**.
- Korzystając z tego wzorca przygotować własną prezentację na wybrany samodzielnie temat np. ulubiony zespół muzyczny, ulubiona książka, ulubione miejsce na ziemi itd.
- Postępy prac nad prezentacją rejestrować jako kolejne *commity* (na przykład każdy nowy element slajdu). Podczas oceny będzie brana pod uwagę liczba *commitów* jak i jasność przypisanych im komunikatów. (UWAGA: pliki graficzne muszą być także umieszczone w repozytorium).
- Następnie repozytorium umieścić na serwerze *gitlab* w projekcie o nazwie **zad3_pti_25Z** . Do projektu jako użytkownika dodać *usera* *mniewins66* (w roli **Maintainer** i dodatkowo proszę ustawić *Access expiration date* na dzień 31.12.2025 r.) .
- W repozytorium proszę umieścić również ostateczną wersję prezentacji w formacie **pdf**
- Jako wynik pracy przesłać link (umieszczony w pliku *link_zad_3.md*) do projektu na serwerze *gitlab* oraz spakowane repozytorium lokalne.