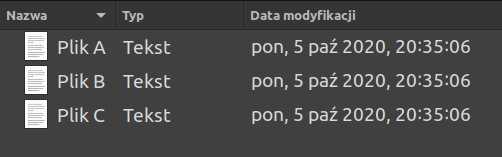
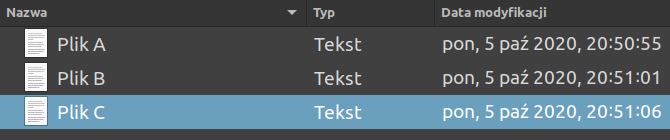
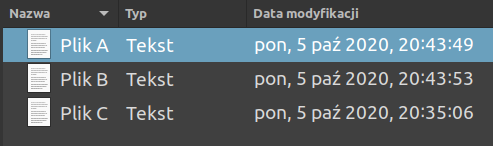
**Git git Co to jest GIT:**

GIT to rozproszony system kontroli wersji. Jest de facto standardem jeśli chodzi o wersjonowanie i dzielenie się kodem wśród programistów  
  
**Ważne: GIT to nie GITHUB:**  
github.com to jeden z dostawców repozytoriów GIT.

**Jakie problemy rozwiązuje GIT:**

1. Praca wielu osób nad tym samym projektem  
2. Trudno jest zmieniać wersje programu  
3. Awaria lokalna / serwera  
4. Wspólne redagowanie kodu  
5. Praca nad wieloma różnymi zagadnieniami które trudno połączyć  
6. Wersja stabilna vs wersja rozwojowa

**Praca wielu osób nad jednym projektem - powód numer jeden dla którego pracujemy z GITem**

Co może pójść źle? Przykładowy scenariusz  
  
Scenariusz 1. Niech obecny stan katalogu projektu wygląda następująco:  
  
  
  
Członkowie projektu Ewa i Adam kopiują pliki z tej wersji i zaczynają je modyfikować, każdy członek zespołu ma przydzielone inne zadanie.  
  
Po zakończonej modyfikacji stan ich plików wygląda następująco:  
  
Wersja Adama  
  
:  
  
Wersja Ewy:  
  
  
  
Zwróć uwagę na datę modyfikacji.  
  
Wersja Ewy Zmieniła pliki A i B, Wersja Adama pliki A, B, C. Projekt mamy tylko jeden, więc która wersja każdego pliku jest prawidłowa?  
  
Załóżmy, że Ewa i Adam nie wiedzą o tym, że jednocześnie pracują na tych samych plikach. Ewa na wspólnym serwerze podmienia pliki pierwsza (skończyła pracę pierwsza).  
Adam, nie wiedząc o zmianie, podmienia wszystkie pliki na swoje. Oboje informują swoich przełożonych o zakończonej pracy.  
Następnego dnia szef pyta Ewę, gdzie są wyniki jej pracy, Ewa zaskoczona nie może ich znaleźć na serwerze.  
  
GIT w tym wypadku samoistnie połączyłby zmiany lub poinformował Adama o konieczności połączenia zmian.

**Podstawy praca z linią komend**

Uruchom wiersz poleceń / linię komend  
  
Aby sprawdzić, w którym katalogu/folderze się znajdujesz (w systemie Unixowym jak Linux bądź Mac Os), wykonaj polecenie

pwd

w systemie Windows

cd

do przejścia do wybranego katalogu służy komenda:

cd <nazwa katalogu/ folderu>

Żeby wylistować pliki w katalogu w systemie unix, wykonaj polecenie:

ls -a

parametr -a oznacza, że chcemy zobaczyć również pliki ukryte.  
  
w systemie windows:

DIR /a:h

/a:h pokaże nam również pliki ukryte.

**Pierwsze repozytorium lokalne**

Przejdź do katalogu, w którym utworzysz repozytorium i wykonaj komendę:

git init

Sprawdź zawartość katalogu - powinien pojawić się w nim podkatalog/ folder .git  
  
Przykładowy rezultat:

git init<Enter>

Initialized empty Git repository in /home/robergo/ict/repozytorium/.git/

ls -a<Enter>

. .. .git

Historia zmian i wszystkie wersje będą od teraz przechowywane w katalogu .git **(nie modyfikuj i nie usuwaj tego katalogu ręcznie)**  
  
utwórz w katalogu plik "pierwszy.txt"  
  
wykonaj polecenie:

git status

przykładowy rezultat:

git status <Enter>

On branch master

No commits yet

Untracked files:

 (use "git add <file>..." to include **in** what will be committed)

pierwszy.txt

nothing added to commit but untracked files present (use "git add" to track)

Polecenie to informuje nas o obecnym statusie repozytorium.  
  
Polecenie to informuje nas o gałęzi, na której się obecnie znajdujemy (zazwyczaj master/main, bądź develop).Gałęzie będą omówione w dalszej części kursu.  
  
W repozytorium nie ma jeszcze żadnych zarejestrowanych zmian (no commits yet), ale git wykrył plik który może być śledzony:  
pierwszy.txt  
  
**Ten plik nie jest jeszcze przechowywany w repozytorium!**

**Commmit - co to jest i jak na nim operować**

Commit to najmniejsza paczka zmian rejestrowana w systemie GIT. Każdy commit ma swój identyfikator i napisany przez człowieka komentarz.  
  
Aby dodać commit z naszym pierwszym plikiem, należy wykonać najpierw polecenie:

git add <nazwa pliku>

jeśli chcemy dodać wiele plików naraz:

git add <nazwa pliku 1> <nazwa pliku 2> <nazwa pliku 3> ....

a następnie:

git commit -m "<opis / komentarz do commitu>"

dopiero wykonanie git commit -m "..." zapisuje nasze zmiany w repozytorium.  
  
Możliwe jest też użycie skróconej formy:

git commit <nazwa pliku 1> <nazwa pliku 2> .... -m "komentarz do commitu"

Polecam użycie pierwszej / dwustopniowej wersji, ponieważ przed dodaniem commita lubię sprawdzić, czy na pewno dodałem wszystkie pliki, których potrzebuję,  
tzn wykonuję:

git add <nazwa pliku 1> ...

git status

git add <nazwa pliku o którym zapomniałem/am>

i dopiero, gdy jestem pewny, że wszystko dodałem, wykonuję:

git commit -m "komentarz"

Wracając do naszego przykładu, dodajmy plik pierwszy.txt do repozytorium:

**git** **add** **pierwszy**.txt<**Enter**>

**git** **commit** **-m** "**dodaj**ę **plik** **pierwszy**.txt"<**Enter**>

[master (root-commit) 91b006a] **dodaj**ę **plik** **pierwszy**.txt

 1 **file** **changed**, 0 **insertions**(+), 0 **deletions**(**-**)

**create** **mode** 100644 **pierwszy**.txt

Git poinformował nas, że dodał 1 plik do repozytorium. Commit nie dodał, ani nie usunął żadnych linii w plikach (0 insertions, 0 deletions).

**GIT commit - dodawanie katalogów, modyfikacja, zmiana nazwy, usuwanie plików**

Git porównuje wasze zmiany linia po linii w plikach tekstowych, tzn. jest w stanie wykryć zmiany (dodanie, usuwanie całych linii).  
  
W pliku pierwszy.txt dodaj następującą treść:

Git

Sprawdźmy obecny stan:

git status

On branch master

Changes **not** staged **for** commit:

 (use "git add <file>..." to update what will be committed)

 (use "git checkout -- <file>..." to discard changes **in** working directory)

zmodyfikowany: pierwszy.txt

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

Git poinformował nas o wykrytej zmianie w jednym pliku: pierwszy,txt  
  
dodajmy nasze zmiany do repozytorium:

**git** **add** **pierwszy**.txt<**Enter**>

**git** **commit** **-m** "**pierwsze** **cztery** **linie** **w** **pliku**"<**Enter**>

[master 3d42584] **pierwszy**.txt

 1 **file** **changed**, 5 **insertions**(+)

Git poinformował nas o dodaniu 5 linii.  
  
.... hmmmm.... ale mieliśmy dodać 4 linie. Usuńmy zatem środkową najkrótszą linię. Zmodyfikujmy plik pierwszy.txt

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Proin nibh augue, suscipit a, scelerisque sed, lacinia **in**, mi.

Etiam pellentesque aliquet tellus.

Phasellus pharetra nulla ac diam.

Dodajmy nasze zmiany do repo:

git add pierwszy.txt<Enter>

git commit -m "usuwam nadmiarową linię" <Enter>

[master 1575e54] usuwam nadmiarową linię

 1 file changed, 1 deletion(-)

Git poinformował nas, że usunął 1 linię, modyfikował 1 plik. Błąd naprawiony.  
  
Nasz plik pierwszy.txt nie opisuje adekwatnie tego, co się w nim znajduje. Zmieńmy jego nazwę w systemie operacyjnym:  
pierwszy.txt na lorem-ipsum.txt  
  
Wykonajmy git status, żeby sprawdzić, jak ta akcja wpłynęła na nasze repozytorium:

git status<Enter>

On branch master

Changes **not** staged **for** commit:

 (use "git add/rm <file>..." to update what will be committed)

 (use "git checkout -- <file>..." to discard changes **in** working directory)

usunięty:   pierwszy.txt

Untracked files:

 (use "git add <file>..." to include **in** what will be committed)

lorem-ipsum.txt

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

GIT twierdzi, że usunęliśmy plik pierwszy.txt i utworzyliśmy nowy plik lorem-ipsum.txt. To nie do końca prawda. Moglibyśmy dodać do commita zapis "usunięto plik pierwszy, dodano "lorem-ipsum.txt", ale w ten sposób zgubilibyśmy historię zmian pliku pierwszy - tzn. na podstawie logu nie bylibyśmy w stanie określić jasno, że wcześniej ten plik miał inną nazwę.  
  
By zrobić to prawidłowo, cofnijmy zmianę nazwy: lorem-ipsum.txt -> pierwszy.txt  
i wykonajmy polecenie git mv (zmiana / przeniesienie pliku):

**git** **mv** **pierwszy**.txt **lorem-ipsum**.txt<**Enter**>

**git** **status**<**Enter**>

**On** **branch** **master**

**Changes** **to** **be** **committed**:

 (**use** "**git** **reset** **HEAD** <**file**>..." **to** **unstage**)

**renamed**:    **pierwszy**.txt **-**> **lorem-ipsum**.txt

Jak widać, tym razem GIT wie o zmianie nazwy - dodajmy w takim razie tą zmianę do commita:

git commit -m "Lepsza nazwa dla pliku pierwszy.txt"

w podobny sposób możemy jasno definiować usunięcie pliku:

git rm <nazwa pliku>

﻿  
Pamiętaj, żeby zawsze dodać commit ze swoimi zmianami (git commit -m "...")  
  
Co, gdy chcemy dodać cały katalog (razem ze znajdującymi się w nim plikami)? Tym samym poleceniem, co normalne dodawanie plików git add. Nie musicie w takim przypadku dodawać każdego pliku w podkatalogu oddzielnie.  
  
Utwórzmy podkatalog "podkatalog", a w nim dwa pliki a.txt b.txt (puste) i dodajmy samą nazwę podkatalogu:

ls podkatalog <Enter>

a.txt b.txt <Enter>

git add podkatalog<Enter>

git commit -m "dodałem podkatalog"

[master 92a1c03] dodałem podkatalog

 2 files changed, 0 insertions(+), 0 deletions(-)

 create mode 100644 podkatalog/a.txt

 create mode 100644 podkatalog/b.txt

Jak widać, git dodał 2 pliki wewnątrz katalogu - co ważne, nie dodał bezpośrednio katalogu "podkatalog". Dzieje się tak dlatego, że GIT nie śledzi samych podkatalogów, jedynie wnioskuje ich obecność na podstawie ścieżek plików. Dlatego **nie może do repozytorium dodać pustego katalogu!**

**GIT - Przywracanie plików, usuwanie z listy do dodania (staging)**

Jak do tej pory tylko dodawaliśmy commity. Co zrobić w momencie, gdy popełniliśmy błąd? Np. usunęliśmy przypadkowo plik, który był bardzo potrzebny, albo dokonaliśmy w pliku zmian, które chcemy cofnąć.  
Rozwiązanie zależy od momentu, do którego chcemy się cofnąć:  
  
Przydatne polecenia:

git checkout <nazwa pliku>

checkout przywraca stan pliku z zapisanego w repozytorium.

git reset HEAD <nazwa pliku>

to polecenie usuwa zmiany w staging.  
  
Przykład 1: Dodaliśmy plik, który nie powinien znaleźć się w commicie:  
utworzymy plik niepotrzebny.txt i dodamy go do staging:

**git** **add** **niepotrzebny**.txt

Polecenie

git reset HEAD

usunie wszystkie dodane pliki do stagingu.  
  
jeśli chcesz usunąć tylko jeden plik:

git reset HEAD <nazwa pliku>

wykonaj polecenie:

git status

zobacz, czy plik dalej będzie dodany do repozytorium.  
  
Przykład 2. Co, gdy przypadkowo usunęliśmy plik? (usunięcie nie zostało dodane do repozytorium)  
Usuńmy plik lorem-ipsum.txt.  
  
Polecenie git status powinno zwrócić następujący wynik:

git status<Enter>

On branch master

Changes **not** staged **for** commit:

 (use "git add/rm <file>..." to update what will be committed)

 (use "git checkout -- <file>..." to discard changes **in** working directory)

usunięty:   lorem-ipsum.txt

Aby przywrócić plik, wykonaj polecenie:

git checkout <nazwa pliku>

W naszym przypadku:

git checkout lorem-ipsum.txt <Enter>

ls<Enter>

lorem-ipsum.txt niepotrzebny.txt podkatalog<Enter>

Przykład 3. Co, jeśli usunęliśmy plik i dodaliśmy go do staging?  
  
Zasymulujmy ten wynik poprzez wykonanie:

git rm lorem-ipsum.txt<Enter>

git status<Enter>

On branch master

Changes to be committed:

 (use "git reset HEAD <file>..." to unstage)

usunięty:   lorem-ipsum.txt

Tę zmianę cofamy w dwóch krokach:

**git** **reset** **HEAD** **lorem-ipsum**.txt

**git** **chekout** **lorem-ipsum**.txt

**Takie podejście można zastosować również do plików które niepotrzebnie zmieniliśmy (bez usuwania).**

**GIT swobodna zmiana wersji, cofanie w czasie projektu, cofanie niektórych commitów.**

Do tej pory po prostu tworzyliśmy commity, pora zrobić z nich użytek.  
Na początek wykonajmy polecenie git log.  
  
git log (wraz z przykładową zwracaną wartością)

git log<Enter>

commit 92a1c03605b9849d552576b9059807064e755f9c (HEAD -> master)

Author: Robert Górzyński <xxxxxxxxx@gmail.com>

Date:  Fri Oct 9 13:06:55 2020 +0200

  dodałem podkatalog

commit 1575e5493cf90ecc349f914cf10943b898a90d58

Author: Robert Górzyński <xxxxxxxx@gmail.com>

Date:  Fri Oct 9 12:41:40 2020 +0200

  usuwam nadmiarową linię

commit 3d4258416268abb002075d36996bff98b87d5cae

Author: Robert Górzyński <xxxxxxxxxx@gmail.com>

Date:  Fri Oct 9 12:38:28 2020 +0200

  pierwszy.txt

commit 91b006ae50414b111b5d481423243a0d8705e94a

Author: Robert Górzyński <xxxxxxxxxxx@gmail.com>

Date:  Fri Oct 9 12:28:22 2020 +0200

  dodaję plik pierwszy.txt

To polecenie zwraca nam historię dotychczasowych commitów, w kolejności od najnowszego do najstarszego.

**commit** 92**a1c03605b9849d552576b9059807064e755f9c** (**HEAD** **-**> **master**)

**Author**: **Robert** **G**ó**rzy**ń**ski** <**xxxxxxxxxx**@**gmail**.**com**>

Date:  Fri Oct 9 13:06:55 2020 +0200

  dodałem podkatalog

ta linia mówi nam, że mieliśmy commit o id:

92a1c03605b9849d552576b9059807064e755f9c

w gałęzi master, który jest obecnie używaną wersją (HEAD -> master)  
Stworzył go Robert Górzyński o podanej godzinie, z opisem "dodałem podkatalog"  
Uwaga: ID prawdopodobnie będzie różne w Twoim repozytorium, to normalne.  
  
Do czego może nam się przydać ten log? Np.do cofnięcia pracy do wcześniejszego stadium (zrobiliśmy duży błąd klika dni temu i właśnie się zorientowaliśmy).  
  
Przykład 1: chcemy wycofać jakąś starą zmianę, jednocześnie zachowując wszystkie przyszłe:

git revert <commit hash/ commit ID>

Przykład:  
Ta linia 5 w pliku pierwszy by nam się jednak przydała:  
znajdujemy commit, który za to odpowiadał:

1575e5493cf90ecc349f914cf10943b898a90d58

skąd to wiemy? Z komentarza do commita  
**"usuwam nadmiarową linię" - dlatego warto dbać o jakość swoich komentarzy**  
  
wykonujemy:

git revert 1575e5493cf90ecc349f914cf10943b898a90d58

git zapyta nas jeszcze o potwierdzenie nowego komentarza do commita, po potwierdzeniu wykonaj polecenie, żeby zobaczyć nowy stan:

git log

commit df939a883352286ed60a1120eacb3ab94ee58f9e (HEAD -> master)

Author: Robert Górzyński <xxxxxxx@gmail.com>

Date:   Fri Oct 9 13:51:02 2020 +0200

    Revert "usuwam nadmiarową linię"

    This reverts commit 1575e5493cf90ecc349f914cf10943b898a90d58.

commit 92a1c03605b9849d552576b9059807064e755f9c

Author: Robert Górzyński <xxxxxxxxxxx@gmail.com>

Date:   Fri Oct 9 13:06:55 2020 +0200

    dodałem podkatalog

commit 1575e5493cf90ecc349f914cf10943b898a90d58

Author: Robert Górzyński <x@gmail.com>

Date:   Fri Oct 9 12:41:40 2020 +0200

    usuwam nadmiarową linię

commit 3d4258416268abb002075d36996bff98b87d5cae

Author: Robert Górzyński <xxxxxxxxxx@gmail.com>

Date:   Fri Oct 9 12:38:28 2020 +0200

    pierwszy.txt

commit 91b006ae50414b111b5d481423243a0d8705e94a

Author: Robert Górzyński <xxxxxxxxxxxx@gmail.com>

Date:   Fri Oct 9 12:28:22 2020 +0200

    dodaję plik pierwszy.txt

Git wstawił nowy commit, z informacją o cofnięciu wskazanych zmian. Co ciekawe, zmiana została cofnięta w nowym pliku "lorem-ipsum" a nie w "pierwszy.txt", To dlatego że prawidłowo daliśmy znać gitowi na późniejszym etapie, że zmieniliśmy nazwę pliku - bardzo wygodne.  
  
Przykład 2: Nie potrafimy namierzyć commitu, który wprowadził zmianę:  
  
Możemy sprawdzić historię zmian w konkretnym pliku. poleceniem  
git log -p <nazwa pliku>  
  
przykładowo:

git log -p lorem-ipsum.txt

commit df939a883352286ed60a1120eacb3ab94ee58f9e (HEAD -> master)

Author: Robert Górzyński <xxxxxx@gmail.com>

Date:  Fri Oct 9 13:51:02 2020 +0200

  Revert "usuwam nadmiarową linię"

  This reverts commit 1575e5493cf90ecc349f914cf10943b898a90d58.

diff --git a/lorem-ipsum.txt b/lorem-ipsum.txt

index 925e712..8ad9128 100644

--- a/lorem-ipsum.txt

+++ b/lorem-ipsum.txt

@@ -1,4 +1,5 @@

 Lorem ipsum dolor sit amet, consectetur adipisicing elit.

 Proin nibh augue, suscipit a, scelerisque sed, lacinia **in**, mi.

+Cras vel lorem.

 Etiam pellentesque aliquet tellus.

 Phasellus pharetra nulla ac diam.

commit 92a1c03605b9849d552576b9059807064e755f9c

Author: Robert Górzyński <xxxxxxxxx@gmail.com>

Date:  Fri Oct 9 13:06:55 2020 +0200

  dodałem podkatalog

diff --git a/lorem-ipsum.txt b/lorem-ipsum.txt

**new** file mode 100644

index 0000000..925e712

--- /dev/null

+++ b/lorem-ipsum.txt

@@ -0,0 +1,4 @@

+Lorem ipsum dolor sit amet, consectetur adipisicing elit.

+Proin nibh augue, suscipit a, scelerisque sed, lacinia **in**, mi.

+Etiam pellentesque aliquet tellus.

+Phasellus pharetra nulla ac diam.

Linie zaczynające się "+" to dodanie linii, "-" linia w danym commicie była usunięta.  
  
Po znalezieniu odpowiedniego commitu możemy wykonać

git revert

Przykład 3 Co, jeśli chcemy po prostu cofnąć się w czasie do konkretnego commitu?

git checkout <commit ID/ Hash>

Cofnijmy się o jedną wersję do momentu, gdy lorem ipsum.txt miał cztery linie:

git checkout 92a1c03605b9849d552576b9059807064e755f9c

Uwaga, to podejście polega na pracy z tzw detached HEAD. Oznacza to, że wasze zmiany nie są już zapisywane w domyślnej gałęzi master (dlatego, że ścieżka się już rozdzieliła). Szczegółową pracę z gałęziami wytłumaczę w dalszej części kursu.