

Podstawy Informatyki

Katedra Telekomunikacji, EiT

dr inż. Jarosław Bułat

kwant@agh.edu.pl

Plan prezentacji

- » Co to jest GIT i dlaczego GIT
- » Koncepcja
- » Podstawowy workflow
 - clone, pull, commit, push
 - konfiguracja, wizualizacja
- » Gałęzie
- » Konflikty
 - rejected push
 - łączenie udane i nieudane czyli konflikt wymagający interwencji
- » Tips&Tricks: git status, git log, tracked/untracked files, .gitignore, git checkout dd4b4b4, git reset --hard, git clean -xf, git blame, git fetch, git revert (undo last commit), git stash

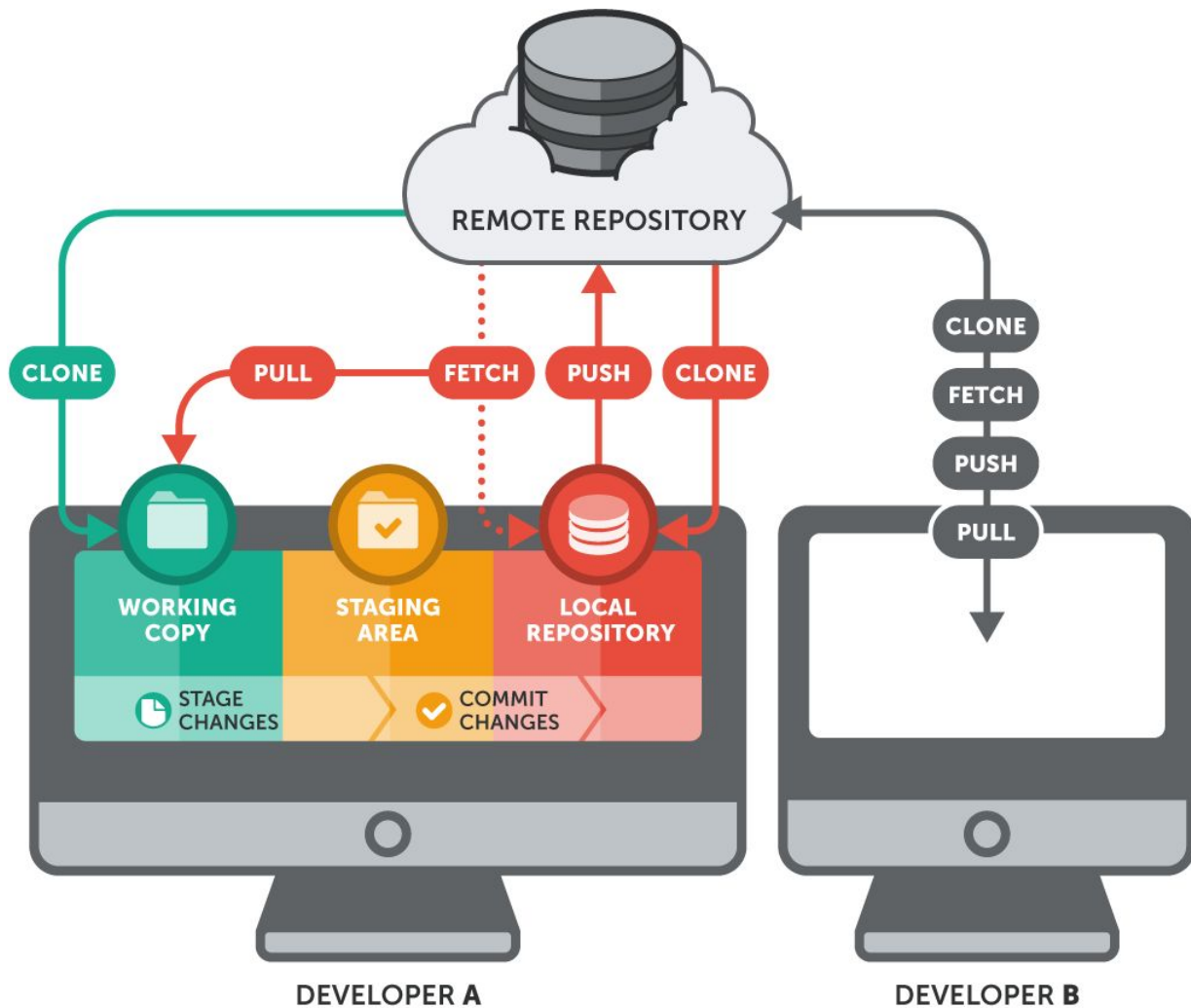
Gdzie mam zapisać *.cc?
w Git

System kontroli wersji

- » Gdzie zapisać plik *.cc? **Na dysku, w katalogu :-)**
 - jak go przesłać koledze-programiście?
 - jak efektywnie współdzielić z wieloma dev.?
 - jak zachować kolejne wersje?
- » VCS - Version Control System
 - **CSV** (Concurrent Versions System)
 - **SVN** (Subversion)
 - **GIT**

System kontroli wersji

- » serwer (remote) przechowujący wszystkie wersje wszystkich developerów
- » lokalnie kopia serwera (na dysku)
- » funkcje:
 - wersjonowanie zmian
 - pamiętanie kto co zrobił
 - rozwiązywanie konfliktów (merge)
 - możliwość cofnięcia się do dowolnej wersji



GIT - podstawy

» TODO: konsola linuxa (shell)

GIT

```
> git clone https://gitlab.com/gr/pro.git
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » “Repozytorium” to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » “Repozytorium” to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog **pro**

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » “Repozytorium” to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog **pro**

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" > text.txt
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » "Repozytorium" to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog pro
- » **Zmień** coś w katalogu

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" >text.txt  
> git add text.txt
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » "Repozytorium" to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog pro
- » Zmień coś w katalogu
- » Zarejestruj nowy plik (dodaj plik do repozytorium lokalnego)

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" >text.txt  
> git add text.txt  
> git commit -m "first version"
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » "Repozytorium" to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog pro
- » Zmień coś w katalogu
- » Zarejestruj nowy plik (dodaj plik do repozytorium lokalnego)
- » **Zarejestruj** zmianę w pliku

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" >text.txt  
> git add text.txt  
> git commit -m "first version"  
> git push
```

- » TODO: konsola linuxa (shell)
- » Ściągnij (clone == skopiuj/sklonuj) repozytorium zewnętrzne (remote) do lokalnego katalogu
- » "Repozytorium" to coś więcej niż tylko katalog z plikami, zawiera również historię zmian, opisy, metadane, etc...
- » Od tego momentu na lokalnym dysku będzie częściowa kopia repozytorium
- » Zostanie utworzony katalog pro
- » Zmień coś w katalogu
- » Zarejestruj nowy plik (dodaj plik do repozytorium lokalnego)
- » Zarejestruj zmianę pliku
- » Prześlij (**wypchnij**) zmianę do zewnętrznego repozytorium

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

» Nie muszę za każdym razem robić
"push"

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" > text1.txt
> echo "xxx" > text2.txt
> git add .
```

- » Nie muszę za każdym razem robić "push"
- » Mogę utworzyć wiele plików i dodać je "hurtem" (rekurencyjnie w podkatalogach)

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text1.txt
> echo "xxx" >text2.txt
> git add .
> git commit -am "first version"
> git push
```

- » Nie muszę za każdym razem robić "push"
- » Mogę utworzyć wiele plików i dodać je "hurtem"
- » Mogę również rejestrować zmiany "**hurtowo**", komenda będzie dotyczyć tylko zmodyfikowanych plików (lub nowo utworzonych)

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text1.txt
> echo "xxx" >text2.txt
> git add .
> git commit -am "first version"
> git push
> rm text2.txt
> git add .
> git commit -am "temp. no longer ..."
```

- » Nie muszę za każdym razem robić "push"
- » Mogę utworzyć wiele plików i dodać je "hurtem"
- » Mogę również rejestrować zmiany "hurtowo", komenda będzie dotyczyć tylko zmodyfikowanych plików (lub nowo utworzonych)
- » Kasowanie pliku to rejestracja jego usunięcia !!!

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text1.txt
> echo "xxx" >text2.txt
> git add .
> git commit -am "first version"
> git push
> rm text2.txt
> git add .
> git commit -am "temp. no longer ..."
> git push
```

- » Nie muszę za każdym razem robić "push"
- » Mogę utworzyć wiele plików i dodać je "hurtem"
- » Mogę również rejestrować zmiany "hurtowo", komenda będzie dotyczyć tylko zmodyfikowanych plików (lub nowo utworzonych)
- » Kasowanie pliku to rejestracja jego usunięcia !!!
- » Zmiana zarejestrowana "na serwerze"

GIT - podstawy

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text1.txt
> echo "xxx" >text2.txt
> git add .
> git commit -am "first version"
> git push
> rm text2.txt
> git add .
> git commit -am "temp. no longer ..."
> git push
```

- » Nie muszę za każdym razem robić "push"
- » Mogę utworzyć wiele plików i dodać je "hurtem"
- » Mogę również rejestrować zmiany "hurtowo", komenda będzie dotyczyć tylko zmodyfikowanych plików (lub nowo utworzonych)
- » Kasowanie pliku to rejestracja jego usunięcia !!!
- » Zmiana zarejestrowana "na serwerze"
- » **Od tego momentu zmiana jest dostępna dla innych developerów**

GIT - team working

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" >text.txt  
> git add text.txt  
> git commit -m "first version"  
> git push
```

» Ty zrobiłeś/zrobiłaś "**push**" czyli serwer (remote) został uaktualniony o nową wersję

GIT - team working

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> git push
```

```
> git clone https://gitlab.com/gr/pro.git
> git pull
> cat text.txt
xxx
```

» Ty zrobiłeś/zrobiłaś **push** czyli serwer (remote) został uaktualniony o nową wersję

» Inny developer...
gdzieś na drugim końcu świata...
uaktualnia swoje lokalne repozytorium
**pobierając najnowsze wersje plików do
swojego lokalnego katalogu**

GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

» Graficzna reprezentacja zmian dwóch
"commitów"

GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> echo "xxx" >text.txt  
> git add text.txt  
> git commit -m "first version"  
> echo "+yyy" >>text.txt  
> git commit -m "second version"  
> git push
```

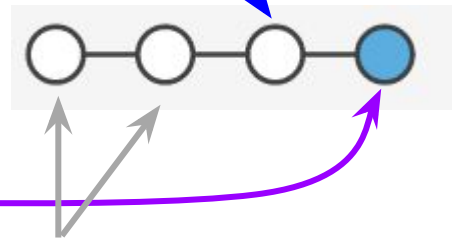
» Graficzna reprezentacja zmian dwóch
"commitów"



GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" > text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >> text.txt
> git commit -m "second version"
> git push
```

» Graficzna reprezentacja zmian dwóch
"commitów"

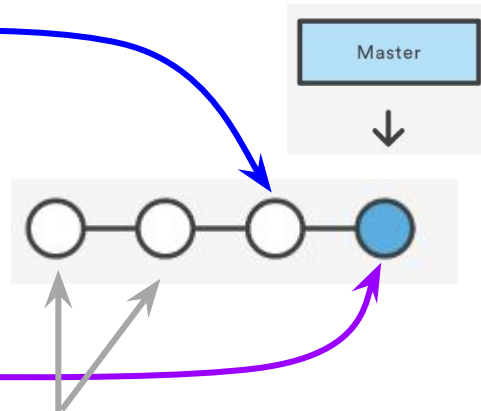


» Wcześniejsze kropki to wcześniejsze zmiany, niekoniecznie tego pliku

GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" > text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >> text.txt
> git commit -m "second version"
> git push
```

» Graficzna reprezentacja zmian dwóch
"commitów"

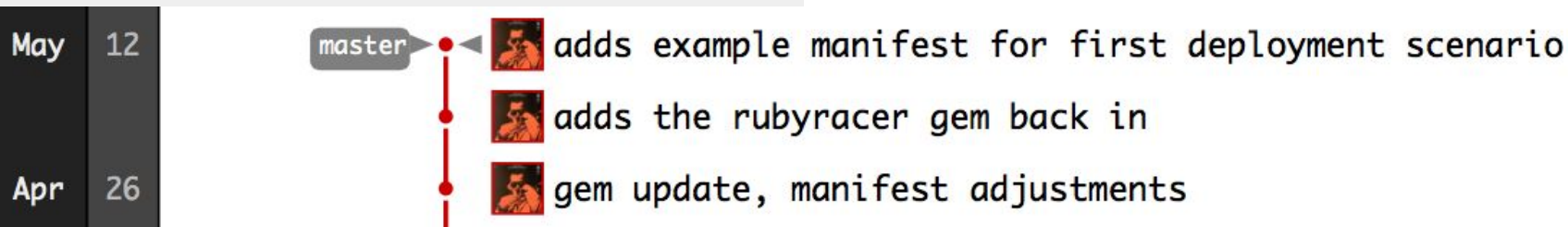


» Wcześniejsze kropki to wcześniejsze zmiany, niekoniecznie tego pliku
» Ostatnia kropka/zmiana/commit to jest chwila obecna "now"

GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" > text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >> text.txt
> git commit -m "second version"
> git push
```

- » Graficzna reprezentacja zmian dwóch "commitów"
- » Wygląd wizualizacji zależy od narzędzia (gitlab, github, IDE, etc...)

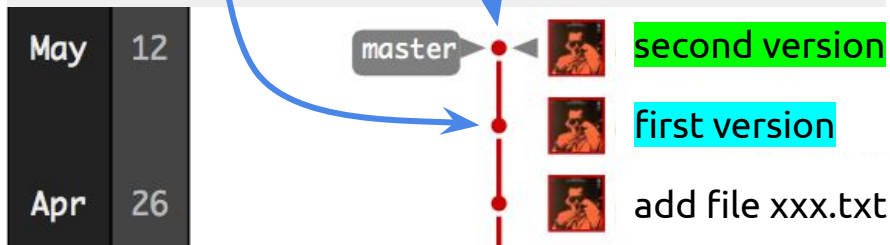


GIT - wizualizacja

```

> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" > text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >> text.txt
> git commit -m "second version"
> git push
  
```

- » Graficzna reprezentacja zmian dwóch „commitów”
- » Wygląd wizualizacji zależy od narzędzia (gitlab, github, IDE, etc...)
- » Wizualizacja często zawiera:
 - datę
 - opis (commit message)



GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia

GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):

GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git  
> git config --global user.name "Your name"
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
 - imię, nazwisko

GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git  
> git config --global user.name "Your name"  
> git config --global user.email your@email.com
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
 - imię, nazwisko
 - e-mail

GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git  
> git config --global user.name "Your name"  
> git config --global user.email your@email.com  
> git config --global push.default simple  
> git config --global credential.helper "cache  
--timeout=3600"
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
 - imię, nazwisko
 - e-mail
 - konfiguracja metody push (safe)
 - zapamiętywanie hasła przez 1h

GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git  
> git config --global user.name "Your name"  
> git config --global user.email your@email.com  
> git config --global push.default simple  
> git config --global credential.helper "cache  
--timeout=3600"
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
 - imię, nazwisko
 - e-mail
 - konfiguracja metody push (safe)
 - zapamiętywanie hasła przez 1h
- » Konfiguracja jest zapisywana w pliku ~/.gitconfig

GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
--timeout=3600"
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc "clone" wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
 - imię, nazwisko
 - e-mail
 - konfiguracja metody push (safe)
 - zapamiętywanie hasła przez 1h
- » Konfiguracja jest zapisywana w pliku ~/.gitconfig

GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
    --timeout=3600"

> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
 - imię, nazwisko
 - e-mail
 - konfiguracja metody push (safe)
 - zapamiętywanie hasła przez 1h
- » Konfiguracja jest zapisywana w pliku `~/.gitconfig`
- » W praktyce klonowanie i konfigurację wykonuje się tylko raz

GIT - konfiguracja

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
    --timeout=3600"

> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

- » Logowanie, konfiguracja
- » Repozytorium może być niepubliczne, więc “clone” wymaga uwierzytelnienia
- » Warto zmienić konfigurację (żeby **git** nie pytał za każdym razem o dane):
 - imię, nazwisko
 - e-mail
 - konfiguracja metody push (safe)
 - zapamiętywanie hasła przez 1h
- » Konfiguracja jest zapisywana w pliku `~/.gitconfig`
- » W praktyce klonowanie i konfigurację wykonuje się tylko raz
- » **W warunkach naszego labu na początku każdego zajęcia !!!**

GIT

gałęzie

GIT - gałęzie



- » **Master** == główna gałąź, podstawowa, stabilny kod, często ograniczone prawa zapisu

GIT - gałęzie



- » **Master** == główna gałąź, podstawowa, stabilny kod, często ograniczone prawa zapisu
- » Z każdego miejsca (commit) mogę utworzyć inne, niezależne wersje kodu

GIT - gałęzie



- » **Master** == główna gałąź, podstawowa, stabilny kod, często ograniczone prawa zapisu
- » Z każdego miejsca (commit) mogę utworzyć inne, niezależne wersje kodu
- » Rozwój programu odbywa się zazwyczaj w innych gałęziach (np. **Develop**)

GIT - gałęzie



- » **Master** == główna gałąź, podstawowa, stabilny kod, często ograniczone prawa zapisu
- » Z każdego miejsca (commit) mogę utworzyć inne, niezależne wersje kodu
- » Rozwój programu odbywa się zazwyczaj w innych gałęziach (np. **Develop**)
- » W pewnym momencie może nastąpić **scalenie** (**merge**) gałęzi,

GIT - gałęzie

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/
```

» Clone kopiuje tylko gałąź Master

GIT - gałęzie

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> git branch -a
```

- » Clone kopiuje tylko gałąź Master
- » Pokaż **wszystkie** gałęzie: local i remote

GIT - gałęzie

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> git branch -a
```

- » Clone kopiuje tylko gałąź Master
- » Pokaż wszystkie gałęzie: local i remote

```
git branch -a  
* master  
remotes/origin/AbandonedGUI  
remotes/origin/HEAD -> origin/master  
remotes/origin/master  
remotes/origin/v3beta
```

GIT - gałęzie

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> git branch -a  
> git checkout v3beta
```

- » Clone kopiuje tylko gałąź Master
- » Pokaż wszystkie gałęzie: local i remote
- » **Przełączanie** gałęzi (ściąga na dysk!)

```
git branch -a  
master  
* v3beta  
remotes/origin/AbandonedGUI  
remotes/origin/HEAD -> origin/master  
remotes/origin/master  
remotes/origin/v3beta
```

GIT - gałęzie

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
```

- » Clone kopiuje tylko gałąź Master
- » Pokaż wszystkie gałęzie: local i remote
- » Przełączanie gałęzi
- » Przełączenie gałęzi (lokalnie)

```
git branch -a
* master
v3beta
remotes/origin/AbandonedGUI
remotes/origin/HEAD -> origin/master
remotes/origin/master
remotes/origin/v3beta
```


GIT - gałęzie

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
> git branch -d v3beta
```

```
git branch -a
* master
remotes/origin/AbandonedGUI
remotes/origin/HEAD -> origin/master
remotes/origin/master
remotes/origin/v3beta
```

- » Clone kopiuje tylko gałąź Master
- » Pokaż wszystkie gałęzie: local i remote
- » Przełączanie gałęzi
- » Przełączenie gałęzi (lokalnie)
- » **Skasowanie** gałęzi lokalnie

GIT - gałęzie

```
> git clone https://gitlab.com/gr/pro.git  
> cd pro/  
> git branch -a  
> git checkout v3beta  
> git checkout master  
> git branch -d v3beta  
> git push origin --delete v3beta
```

- » Clone kopiuje tylko gałąź Master
- » Pokaż wszystkie gałęzie: local i remote
- » Przełączanie gałęzi
- » Przełączenie gałęzi (lokalnie)
- » Skasowanie gałęzi lokalnie
- » Skasowanie gałęzi **na serwerze**

GIT - gałęzie

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
> git branch -d v3beta
> git push origin --delete v3beta
> git checkout -b feature1
```

- » Clone kopiuje tylko gałąź Master
- » Pokaż wszystkie gałęzie: local i remote
- » Przełączanie gałęzi
- » Przełączenie gałęzi (lokalnie)
- » Skasowanie gałęzi lokalnie
- » Skasowanie gałęzi na serwerze
- » Utworzenie **nowej gałęzi** (lokalnie) o nazwie **feature1**

GIT - gałęzie

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
> git branch -d v3beta
> git push origin --delete v3beta
> git checkout -b feature1
> git push -u origin feature1
```

- » Clone kopiuje tylko gałąź Master
- » Pokaż wszystkie gałęzie: local i remote
- » Przełączanie gałęzi
- » Przełączenie gałęzi (lokalnie)
- » Skasowanie gałęzi lokalnie
- » Skasowanie gałęzi na serwerze
- » Utworzenie nowej gałęzi (lokalnie) o nazwie feature1
- » **Wysłanie** gałęzi na **serwer**
 - główny serwer to “origin”
 - gałąź umieszcza się tylko raz na serwerze

GIT - gałęzie

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
> git branch -d v3beta
> git push origin --delete v3beta
> git checkout -b feature1
> git push -u origin feature1
> # change something
> git commit -am "hot fix"
> git push
```

- » Clone kopiuje tylko gałąź Master
- » Pokaż wszystkie gałęzie: local i remote
- » Przełączanie gałęzi
- » Przełączenie gałęzi (lokalnie)
- » Skasowanie gałęzi lokalnie
- » Skasowanie gałęzi na serwerze
- » Utworzenie nowej gałęzi (lokalnie) o nazwie feature1
- » Wysłanie gałęzi na serwer
 - główny serwer to "origin"
 - gałąź umieszcza się tylko raz na serwerze
 - później już tylko commit i push (nie trzeba wskazywać -u origin)

dlaczego push
się nie powiódł?
dlatego bo masz konflikty...

GIT - rejected push

developer 1

```
> git add source1.cc  
> git commit -am "source1"
```

developer 2

GIT - rejected push

developer 1

```
> git add source1.cc  
> git commit -am "source1"
```

developer 2

```
> git add source2.cc  
> git commit -am "source2"  
> git push
```

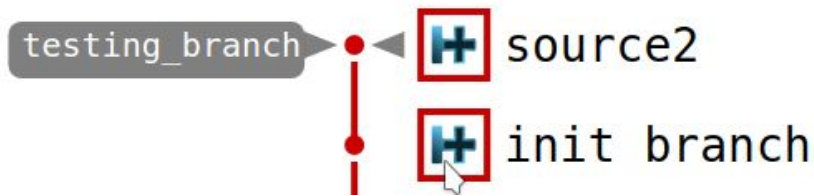

GIT - rejected push

developer 1

```
> git add source1.cc  
> git commit -am "source1"
```

developer 2

```
> git add source2.cc  
> git commit -am "source2"  
> git push
```



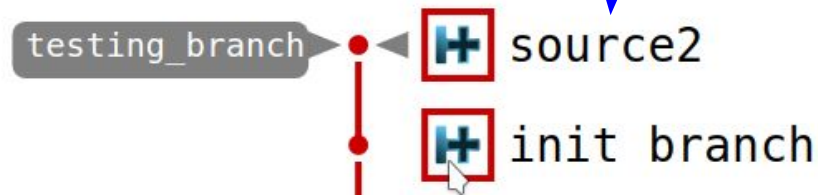
GIT - rejected push

developer 1

```
> git add source1.cc  
> git commit -am "source1"
```

developer 2

```
> git add source2.cc  
> git commit -am "source2"  
> git push
```



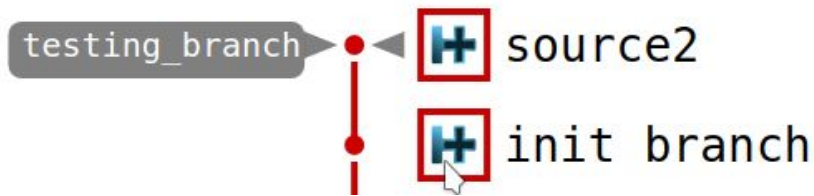
GIT - rejected push

developer 1

```
> git add source1.cc  
> git commit -am "source1"  
  
> git push
```

developer 2

```
> git add source2.cc  
> git commit -am "source2"  
> git push
```



GIT - rejected push

```
> git push
```

```
To https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git
```

```
> git ! [rejected]      testing_branch -> testing_branch (fetch first)
```

```
> git error: failed to push some refs to
```

```
'https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git'
```

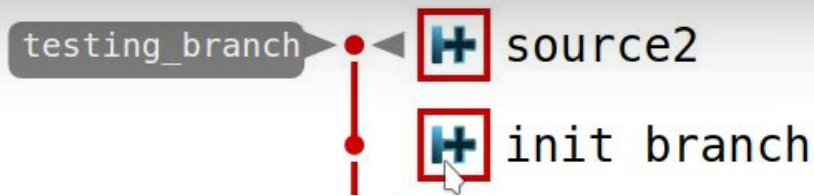
```
hint: Updates were rejected because the remote contains work that you do
```

```
hint: not have locally. This is usually caused by another repository pushing
```

```
hint: to the same ref. You may want to first integrate the remote changes
```

```
> > g hint: (e.g., 'git pull ...') before pushing again.
```

```
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```



GIT - rejected push

```
> git push
```

```
To https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git
```

```
> git ! [rejected] testing_branch -> testing_branch (fetch first)
```

```
> git error: failed to push some refs to
```

```
'https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git'
```

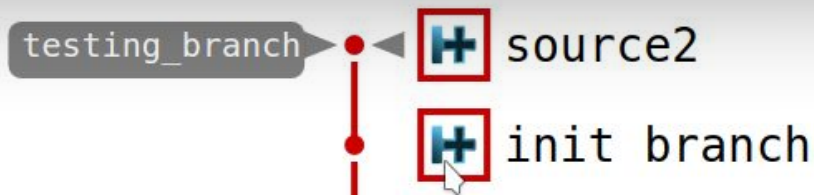
hint: **Updates were rejected because the remote contains work that you do**

hint: **not have locally.** This is usually caused by another repository pushing

hint: to the same ref. You may want to first integrate the remote changes

```
> > g hint: (e.g., 'git pull ...') before pushing again.
```

hint: See the 'Note about fast-forwards' in 'git push --help' for details.



GIT - rejected push

```
~/g/testing-repo-2016 (testing_branch)> ls
```

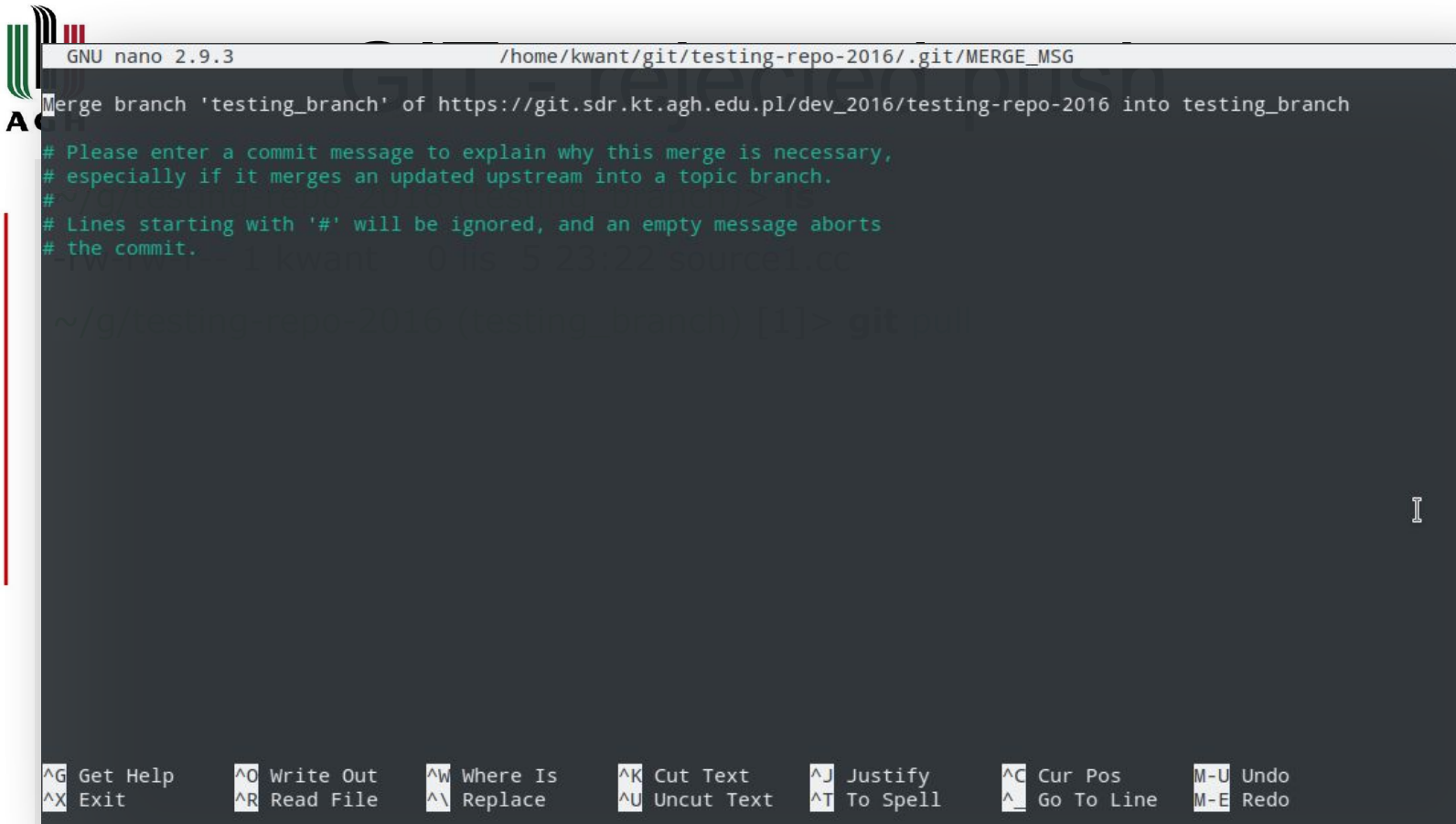
```
-rw-rw-r-- 1 kwant  0 lis  5 23:22 source1.cc
```

GIT - rejected push

```
~/g/testing-repo-2016 (testing_branch)> ls
```

```
-rw-rw-r-- 1 kwant  0 lis  5 23:22 source1.cc
```

```
~/g/testing-repo-2016 (testing_branch) [1]> git pull
```



```
GNU nano 2.9.3 /home/kwant/git/testing-repo-2016/.git/MERGE_MSG
Merge branch 'testing_branch' of https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016 into testing_branch
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~ /g/testing-repo-2016 (testing_branch) [1]> git pull
```

| | | | | | | |
|--------------------|---------------------|--------------------|----------------------|--------------------|----------------------|-----------------|
| ^G Get Help | ^O Write Out | ^W Where Is | ^K Cut Text | ^J Justify | ^C Cur Pos | M-U Undo |
| ^X Exit | ^R Read File | ^_ Replace | ^U Uncut Text | ^T To Spell | ^_ Go To Line | M-E Redo |

GIT - rejected push

```
~/g/testing-repo-2016 (testing_branch)> ls
```

```
-rw-rw-r-- 1 kwant  0 lis  5 23:22 source1.cc
```

```
~/g/testing-repo-2016 (testing_branch) [1]> git pull
```

```
From https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016
```

```
ebd57ae..dc4ac53 testing_branch -> origin/testing_branch
```

Merge made by the 'recursive' strategy.

```
source2.cc | 0
```

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

```
create mode 100644 source2.cc
```

GIT - rejected push

```
~/g/testing-repo-2016 (testing_branch)> ls
```

```
-rw-rw-r-- 1 kwant  0 lis  5 23:22 source1.cc
```

```
~/g/testing-repo-2016 (testing_branch) [1]> git pull
```

From https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016

ebd57ae..dc4ac53 testing_branch -> origin/testing_branch

Merge made by the 'recursive' strategy.

source2.cc | 0

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

create mode 100644 source2.cc

```
~/g/testing-repo-2016 (testing_branch)> ls
```

```
-rw-rw-r-- 1 kwant  0 lis  5 23:22 source1.cc
```

```
-rw-rw-r-- 1 kwant  0 lis  5 23:23 source2.cc
```

GIT - rejected push

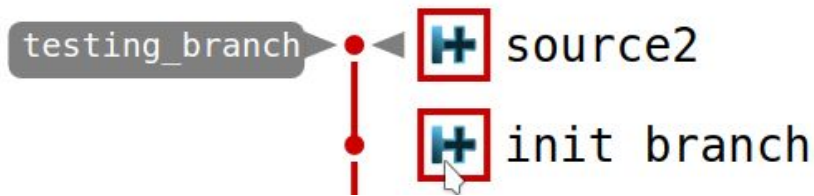
developer 1

```
> git add source1.cc  
> git commit -am "source1"
```

```
> git push
```

developer 2

```
> git add source2.cc  
> git commit -am "source2"  
> git push
```



GIT - rejected push

developer 1

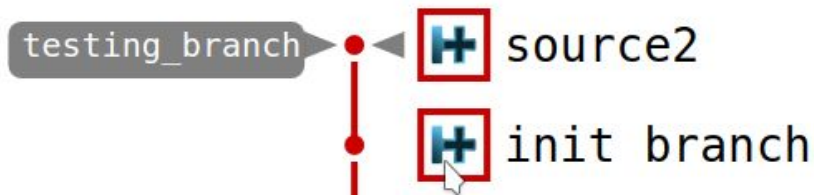
```
> git add source1.cc  
> git commit -am "source1"
```

```
> git push
```

```
> git pull
```

developer 2

```
> git add source2.cc  
> git commit -am "source2"  
> git push
```



GIT - rejected push

developer 1

```
> git add source1.cc  
> git commit -am "source1"
```

```
> git push
```

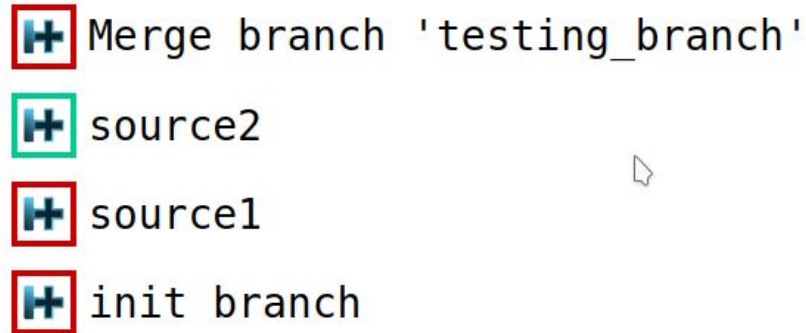
```
> git pull
```

```
> git push
```

developer 2

```
> git add source2.cc  
> git commit -am "source2"  
> git push
```

testing_branch



GIT - rejected push

» Wnioski

- “**pull**”-uj jak najczęściej
- obowiązkowo zaczynaj pracę od **pull**
- nie bój się **merge**, przyzwyczaj się, to jest “codzienność” gita

dlaczego push
się nie powiódł?
konflikt w tym samym pliku

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <istream>
```

developer 2

> **cat** source.cc

```
#include <istream>
```


GIT - rejected push

developer 1

> **cat** source.cc

```
#include <istream>
```

developer 2

> **cat** source.cc

```
#include <istream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "add include"

> **git** push

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <istream>
```

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am "fix include"

> **git** push

developer 2

> **cat** source.cc

```
#include <istream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "add include"

> **git** push

GIT - rejected push

developer 1

developer 2

```
> cat sdr_testing_repo_testing_branch.txt
> git push
To https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git
! [rejected]        testing_branch -> testing_branch (fetch first)
error: failed to push some refs to
'https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am "fix include"

> ~~**git** push~~

> **git** pull

developer 2

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "add include"

> **git** push

GNU nano 2.9.3

/home/kwant/git/testing-repo-2016/.git/MERGE_MSG

Merge branch 'testing_branch' of https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016 into testing_branch

Please enter a commit message to explain why this merge is necessary,
especially if it merges an updated upstream into a topic branch.

Lines starting with '#' will be ignored, and an empty message aborts
the commit.

> cat source.cc

> vim source.cc

> git commit -am "fix include"

> git push

> git pull

> git

^G Get Help
^X Exit

^O Write Out
^R Read File

^W Where Is
^_ Replace

^K Cut Text
^U Uncut Text

^J Justify
^T To Spell

^C Cur Pos
^_ Go To Line

M-U Undo
M-E Redo

GIT - rejected push

developer 1

developer 2

```
> cat source.cc
> git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
> vim source.cc
From https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016
ce833c3..596b335 testing_branch -> origin/testing_branch
Auto-merging source.cc
Merge made by the 'recursive' strategy.
source.cc | 1 ++
1 file changed, 1 insertions(+)
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am "fix include"

> ~~**git** push~~

> **git** pull

developer 2

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "add include"

> **git** push

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am "fix include"

> ~~git~~ push

> **git** pull

> **cat** source.cc

```
#include <ofstream>
#include <string>
```

developer 2

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "add include"

> **git** push

dlaczego auto merge
się nie powiódł?
konflikt w tym samym pliku

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <fstream>
```

developer 2

> **cat** source.cc

```
#include <fstream>
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "fix include"

> **git** push

developer 2

> **cat** source.cc

```
#include <ofstream>
```

> **vim** source.cc

> **git** commit -am "add include"

> **git** push

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "fix include"

> ~~**git** push~~

> **git** pull

developer 2

> **cat** source.cc

```
#include <ofstream>
```

> **vim** source.cc

> **git** commit -am "add include"

> **git** push

GIT - rejected push

developer 1

developer 2

```
> cat source.cc
> git pull
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016
   2e93d11..1604a6e testing_branch -> origin/testing_branch
> vim source.cc
> git commit -m "Auto-merging source.cc"
> git push
CONFLICT (content): Merge conflict in source.cc
Automatic merge failed; fix conflicts and then commit the result.
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "fix include"

> ~~**git** push~~

> **git** pull

> **cat** source.cc

developer 2

> **cat** source.cc

```
#include <ofstream>
```

> **vim** source.cc

> **git** commit -am "add include"

> **git** push

```
<<<<<< HEAD
```

```
#include <string>
```

```
=====
```

```
#include <ofstream>
```

```
>>>>>> 1604a6ee86ac1084ec95e25f3de80c28be4e79f7
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "fix include"

> ~~**git** push~~

> **git** pull

> **cat** source.cc

developer 2

> **cat** source.cc

```
#include <ofstream>
```

> **vim** source.cc

> **git** commit -am "add include"

> **git** push

```
<<<<<< HEAD
```

```
#include <string>
```

```
=====
```

```
#include <ofstream>
```

```
>>>>>> 1604a6ee86ac1084ec95e25f3de80c28be4e79f7
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

```
#include <string>
```

> **vim** source.cc

> **git** commit -am "fix include"

> ~~**git** push~~

> **git** pull

> **cat** source.cc

developer 2

> **cat** source.cc

```
#include <ofstream>
```

> **vim** source.cc

> **git** commit -am "add include"

> **git** push

```
#include <string>
```


GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

```
#include <string>
```

> **vim** source.cc

> **git** commit -am "fix include"

> ~~**git** push~~

> **git** pull

> **cat** source.cc

> **git** commit ...; **git** push

developer 2

> **cat** source.cc

```
#include <ofstream>
```

> **vim** source.cc

> **git** commit -am "add include"

> **git** push

```
#include <string>
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "fix include"

> ~~**git** push~~

> **git** pull

> **cat** source.cc

> **git** commit ...; **git** push

developer 2

> **cat** source.cc

```
#include <ofstream>
```

> **vim** source.cc

> **git** commit -am "add include"

> **git** push

.

.

.

.

> **git** pull

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am "fix include"

> ~~**git** push~~

> **git** pull

> **cat** source.cc

> **git** commit ...; **git** push

developer 2

> **cat** source.cc

```
#include <ofstream>
```

> **vim** source.cc

> **git** commit -am "add include"

> **git** push

.

.

.

.

> **git** pull

> **cat** source.cc

```
#include <string>
```

Przydatne komendy

takie małe FAQ

git FAQ: status

```
> git status
```

» Stan repozytorium **lokalnego**

git FAQ: status

> **git status**

On branch master

Your branch is up to date with
'origin/master'.

nothing to commit, working tree clean

» Stan repozytorium **lokalnego**

git FAQ: status

> **git status**

On branch master

Your branch is **up to date** with
'origin/master'.

nothing to commit, working tree clean

- » Stan repozytorium lokalnego
- » Wszystko ok: **up to date**

git FAQ: status

> **git status**

On branch testing_branch

Your branch and 'origin/testing_branch'

have **diverged**,

and **have 1 and 1 different commits each**,

respectively.

(use "git pull" to merge the remote branch
into yours)

nothing to commit, working tree clean

- » Stan repozytorium lokalnego
- » Gałęzie się **"rozeszły"**

git FAQ: status

> **git** status

On branch testing_branch

Your branch is up to date with
'origin/testing_branch'.

Untracked files:

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

source1.cc

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

- » Stan repozytorium lokalnego
- » Nowy plik w lokalnym filesystemie ale nie dodany do repozytorium
- » Pomoże **git add** .

git FAQ: status

> **git status**

On branch testing_branch

Your branch is up to date with
'origin/testing_branch'.

Changes to be committed:

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

new file: **source1.cc**

- » Stan repozytorium lokalnego
- » Nowy plik w lokalnym filesystemie ale nie dodany do repozytorium
- » Pomoże **git add .**
- » Nowy plik w repozytorium ale nie commitowany

git FAQ: status

> **git status**

On branch testing_branch

Your branch is ahead of

'origin/testing_branch' by 1 commit.

(use "**git push**" to publish your local commits)

nothing to commit, working tree clean

- » Stan repozytorium lokalnego
- » Nowy plik w lokalnym filesystemie ale nie dodany do repozytorium
- » Pomoże **git add** .
- » Nowy plik w repozytorium ale nie commitowany
- » Lokalna gałąź jest "ahead" czyli bardziej aktualna niż ta na serwerze

git FAQ: log

> **git log -2**

commit 3e1144b0ebf53dc94f021b602636bf5f4a1fae6c

(HEAD -> master, origin/master, origin/HEAD)

Author: Jaroslaw Bulat <kwant@agh.edu.pl>

Date: Thu Jan 3 23:45:18 2019 +0100

generic, STL, Vector, przyszlosc C++

commit bc09872b078e04654cd6eb43d87f0a3a0ceec095

Author: Jaroslaw Bulat <kwant@agh.edu.pl>

Date: Tue Jan 1 10:42:26 2019 +0100

operacje dyskowe, podejście wstepujace/zstepujace

- » Historia ostatnich **N** commitów
- » wiadomość (**-m "message"**)

git FAQ: log

```
> git log -2
```

```
commit 3e1144b0ebf53dc94f021b602636bf5f4a1fae6c
```

```
(HEAD -> master, origin/master, origin/HEAD)
```

```
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
```

```
Date: Thu Jan 3 23:45:18 2019 +0100
```

```
generic, STL, Vector, przyszlosc C++
```

```
commit bc09872b078e04654cd6eb43d87f0a3a0ceec095
```

```
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
```

```
Date: Tue Jan 1 10:42:26 2019 +0100
```

```
operacje dyskowe, podejscie wstepujace/zstepujace
```

- » Historia ostatnich N commitów
- » wiadomość (-m “message”)
- » Skrót (hash) reprezentujący commit
 - unikalny w obrębie repozytorium

git FAQ: log

```
> git log -2
```

```
commit 3e1144b0ebf53dc94f021b602636bf5f4a1fae6c
```

```
(HEAD -> master, origin/master, origin/HEAD)
```

```
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
```

```
Date: Thu Jan 3 23:45:18 2019 +0100
```

```
generic, STL, Vector, przyszlosc C++
```

```
commit bc09872b078e04654cd6eb43d87f0a3a0ceec095
```

```
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
```

```
Date: Tue Jan 1 10:42:26 2019 +0100
```

```
operacje dyskowe, podejscie wstepujace/zstepujace
```

- » Historia ostatnich N commitów
- » wiadomość (-m “message”)
- » Skrót (hash) reprezentujący commit
 - unikalny w obrębie repozytorium
 - często skracany do 8-10 pierwszych liter (można używać wymiennie długi i krótki zapis)
 - pokazuje stan lokalnego repozytorium

git FAQ: checkout

```
> git log -2
```

```
commit 3e1144b0ebf53dc94f021b602636bf5f4a1fae6c
```

```
(HEAD -> master, origin/master, origin/HEAD)
```

```
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
```

```
Date: Thu Jan 3 23:45:18 2019 +0100
```

```
> git reset --hard 3e1144b0
```

- » Historia ostatnich N commitów
- » wiadomość (-m “message”)
- » Skrót (hash) reprezentujący commit
 - unikalny w obrębie repozytorium
 - często skracany do 8-10 pierwszych liter (można używać wymiennie długi i krótki zapis)
 - pokazuje stan lokalnego repozytorium
- » Cofnięcie się do dowolnego commitu
 - dowolnej gałęzi
 - przywrócenie HEAD za pomocą “git pull”

FAQ git - reset

```
> git reset --hard
```

- » Nabroilem ;-) w lokalnym repozytorium
 - chcę cofnąć wszystkie zmiany
 - przywrócić HEAD
 - git jest uparty, domaga się merge
- » Opcja **--hard** przywróci stan HEAD
- » Kasuje wszystkie lokalne zmiany jeżeli lokalnie nie było “commit-a”

FAQ git - reset

```
> git reset --hard  
> git reset  
> git reset --soft  
> git reset --mixed
```

- » Nabroilem ;-) w lokalnym repozytorium
 - chcę cofnąć wszystkie zmiany
 - przywrócić HEAD
 - git jest uparty, domaga się merge
- » Opcja --hard przywróci stan HEAD
- » Kasuje wszystkie lokalne zmiany jeżeli lokalnie nie było “commit-a”
- » ToDo: zadanie domowe :)

FAQ git - reset

```
> git reset --hard  
> git reset  
> git reset --soft  
> git reset --mixed  
> git reset --hard HEAD~10
```

- » Nabroilem ;-) w lokalnym repozytorium
 - chcę cofnąć wszystkie zmiany
 - przywrócić HEAD
 - git jest uparty, domaga się merge
- » Opcja --hard przywróci stan HEAD
- » Kasuje wszystkie lokalne zmiany jeżeli lokalnie nie było “commit-a”
- » ToDo: zadanie domowe :)
- » Cofnięcie się o 10 “commit-ów” od HEAD

FAQ git - reset

```
> git reset --hard  
> git reset  
> git reset --soft  
> git reset --mixed  
> git reset --hard HEAD~10  
> git revert
```

- » Nabroilem ;-)) w lokalnym repozytorium
 - chcę cofnąć wszystkie zmiany
 - przywrócić HEAD
 - git jest uparty, domaga się merge
- » Opcja --hard przywróci stan HEAD
- » Kasuje wszystkie lokalne zmiany jeżeli lokalnie nie było “commit-a”
- » ToDo: zadanie domowe :)
- » Cofnięcie się o 10 “commit-ów” od HEAD
- » Undo: utworzy nowy commit, który “zaneguje” poprzednie
 - historia “undo” pozostanie!!

FAQ git - reset

```
> git reset --hard  
> git reset  
> git reset --soft  
> git reset --mixed  
> git reset --hard HEAD~10  
> git revert  
> git revert
```

- » Nabroilem ;-) w lokalnym repozytorium
 - chcę cofnąć wszystkie zmiany
 - przywrócić HEAD
 - git jest uparty, domaga się merge
- » Opcja --hard przywróci stan HEAD
- » Kasuje wszystkie lokalne zmiany jeżeli lokalnie nie było “commit-a”
- » ToDo: zadanie domowe :)
- » Cofnięcie się o 10 “commit-ów” od HEAD
- » Undo: utworzy nowy commit, który “zaneguje” poprzednie
 - historia “undo” pozostanie!!
 - ostatni “commit”
 -

FAQ git - reset

```
> git reset --hard  
> git reset  
> git reset --soft  
> git reset --mixed  
> git reset --hard HEAD~10  
> git revert  
> git revert  
> git revert HEAD~2
```

- » Nabroilem ;-)) w lokalnym repozytorium
 - chcę cofnąć wszystkie zmiany
 - przywrócić HEAD
 - git jest uparty, domaga się merge
- » Opcja --hard przywróci stan HEAD
- » Kasuje wszystkie lokalne zmiany jeżeli lokalnie nie było “commit-a”
- » ToDo: zadanie domowe :)
- » Cofnięcie się o 10 “commit-ów” od HEAD
- » Undo: utworzy nowy commit, który “zaneguje” poprzednie
 - historia “undo” pozostanie!!
 - ostatni “commit”
 - dwa ostatnie “commit-y”

FAQ git - reset

```
> git reset --hard  
> git reset  
> git reset --soft  
> git reset --mixed  
> git reset --hard HEAD~10  
> git revert  
> git revert  
> git revert HEAD~2  
> git revert HASH
```

- » Nabroilem ;-)) w lokalnym repozytorium
 - chcę cofnąć wszystkie zmiany
 - przywrócić HEAD
 - git jest uparty, domaga się merge
- » Opcja --hard przywróci stan HEAD
- » Kasuje wszystkie lokalne zmiany jeżeli lokalnie nie było “commit-a”
- » ToDo: zadanie domowe :)
- » Cofnięcie się o 10 “commit-ów” od HEAD
- » Undo: utworzy nowy commit, który “zaneguje” poprzednie
 - historia “undo” pozostanie!!
 - ostatni “commit”
 - dwa ostatnie “commit-y”
 - do podanego HASH-a

FAQ git - reset

```
> git reset --hard  
> git reset  
> git reset --soft  
> git reset --mixed  
> git reset --hard HEAD~10  
> git revert  
> git revert  
> git revert HEAD~2  
> git revert HASH
```

- » Nabroilem ;-)) w lokalnym repozytorium
 - chcę cofnąć wszystkie zmiany
 - przywrócić HEAD
 - git jest uparty, domaga się merge
- » Opcja --hard przywróci stan HEAD
- » Kasuje wszystkie lokalne zmiany jeżeli lokalnie nie było “commit-a”
- » ToDo: zadanie domowe :)
- » Cofnięcie się o 10 “commit-ów” od HEAD
- » Undo: utworzy nowy commit, który “zaneguje” poprzednie

Więcej o undo/redo:

<https://blog.github.com/2015-06-08-how-to-undo-almost-anything-with-git/>

git FAQ: pull vs fetch

```
> git pull
```

- » Zsynchronizuj remote z local
(ściągnij z serwera na dysk)

git FAQ: pull vs fetch

> **git** pull

> **git** fetch

- » Zsynchronizuj remote z local (ściągnij z serwera na dysk)
- » Uaktualnij wiedzę o “remote”

git FAQ: pull vs fetch

> **git** pull

> **git** fetch

- » Zsynchronizuj remote z local (ściągnij z serwera na dysk)
- » Uaktualnij wiedzę o “remote”
- » Git nie potrzebuje stałej łączności z serwerem (remote)
 - możesz programować w Bieszczadach ;-)
 - “git commit ...” nie sprawdzi czy lokalne repozytorium jest aktualne w stosunku do remote
 - **fetch** ściągnie metadane, **status** powie, że repo jest “behind”

git FAQ: pull vs fetch

> **git** pull

> **git** fetch

- » Zsynchronizuj remote z local (ściągnij z serwera na dysk)
- » Uaktualnij wiedzę o “remote”
- » Git nie potrzebuje stałej łączności z serwerem (remote)
 - możesz programować w Bieszczadach ;-)
 - “git commit ...” nie sprawdzi czy lokalne repozytorium jest aktualne w stosunku do remote
 - fetch ściągnie metadane, status powie, że repo jest “behind”
- » Nigdy nie zmieni lokalnych plików!!!

git FAQ: pull vs fetch

> **git** pull

> **git** fetch

> **git** pull == git fetch + git merge

- » Zsynchronizuj remote z local (ściągnij z serwera na dysk)
- » Uaktualnij wiedzę o “remote”
- » Git nie potrzebuje stałej łączności z serwerem (remote)
 - możesz programować w Bieszczadach ;-)
 - “git commit ...” nie sprawdzi czy lokalne repozytorium jest aktualne w stosunku do remote
 - fetch ściągnie metadane, status powie, że repo jest “behind”
- » Nigdy nie zmieni lokalnych plików!!!
- » Pull to fetch + merge (jeżeli potrzeba)
- » Bezpieczne, możesz zawsze wykonać

FAQ git - clean

> **git**

- » Lokalnie mam pliki, które są poza repo (poza kontrolą wersji, *untracked*), jak je usunąć?

FAQ git - clean

```
> git clean
```

- » Lokalnie mam pliki, które są poza repo (poza kontrolą wersji, *untracked*), jak je usunąć?
- » Modyfikuje wyłącznie lokalny filesystem

FAQ git - clean

```
> git clean
```

```
> git clean -n
```

- » Lokalnie mam pliki, które są poza repo (poza kontrolą wersji, *untracked*), jak je usunąć?
- » Modyfikuje wyłącznie lokalny filesystem
- » Interaktywnie pokaże które pliki usunie

FAQ git - clean

> **git** clean

> **git** clean -n

> **git** clean -f

- » Lokalnie mam pliki, które są poza repo (poza kontrolą wersji, *untracked*), jak je usunąć?
- » Modyfikuje wyłącznie lokalny filesystem
- » Interaktywnie pokaże które pliki usunie
- » Force, jeżeli pliki są objęte kontrolą wersji ale są niepotrzebne (np. po “git reset”)

FAQ git - clean

```
> git clean  
> git clean -n  
> git clean -f  
> git clean -d
```

- » Lokalnie mam pliki, które są poza repo (poza kontrolą wersji, *untracked*), jak je usunąć?
- » Modyfikuje wyłącznie lokalny filesystem
- » Interaktywnie pokaże które pliki usunie
- » Force, jeżeli pliki są objęte kontrolą wersji ale są niepotrzebne (np. po “git reset”)
- » Łącznie z katalogami (rekursywnie)

FAQ git - clean

```
> git clean  
> git clean -n  
> git clean -f  
> git clean -d  
> git clean -X #upper case
```

- » Lokalnie mam pliki, które są poza repo (poza kontrolą wersji, *untracked*), jak je usunąć?
- » Modyfikuje wyłącznie lokalny filesystem
- » Interaktywnie pokaże które pliki usunie
- » Force, jeżeli pliki są objęte kontrolą wersji ale są niepotrzebne (np. po “git reset”)
- » Łącznie z katalogami (rekursywnie)
- » Ignorowane pliki (.gitignore)

FAQ git - clean

```
> git clean  
> git clean -n  
> git clean -f  
> git clean -d  
> git clean -X #upper case  
> git clean -x #lower case
```

- » Lokalnie mam pliki, które są poza repo (poza kontrolą wersji, *untracked*), jak je usunąć?
- » Modyfikuje wyłącznie lokalny filesystem
- » Interaktywnie pokaże które pliki usunie
- » Force, jeżeli pliki są objęte kontrolą wersji ale są niepotrzebne (np. po “git reset”)
- » Łącznie z katalogami (rekursywnie)
- » Ignorowane pliki (.gitignore)
- » Ignorowane i nieignorowane pliki

FAQ git - .gitignore

>

- » Jak ignorować pewne typy plików np:
- ***.o** kompilacja
 - **build/** cały katalog
 - **~ | .back** backup edytora
 - **.idea/** tymczasowy IDE

FAQ git - .gitignore

>

- » Jak ignorować pewne typy plików np:
 - ***.o** kompilacja
 - **build/** cały katalog
 - **~ | .back** backup edytora
 - **.idea/** tymczasowy IDE
- » Jeżeli nic z tym nie robię, **git add .** doda wszystkie te “śmieci” do repo.

FAQ git - .gitignore

> `.gitignore`

- » Jak ignorować pewne typy plików np:
 - `*.o` kompilacja
 - `build/` cały katalog
 - `~ | .back` backup edytora
 - `.idea/` tymczasowy IDE
- » Jeżeli nic z tym nie robię, `git add .` doda wszystkie te “śmieci” do repo
- » W pliku `.gitignore` dodajemy wzorce nazw plików, które chcemy ignorować

FAQ git - .gitignore

> .gitignore

- » Jak ignorować pewne typy plików np:
 - *.o kompilacja
 - build/ cały katalog
 - ~ | .back backup edytora
 - .idea/ tymczasowy IDE
- » Jeżeli nic z tym nie robię, **git add .** doda wszystkie te “śmieci” do repo
- » W pliku .gitignore dodajemy wzorce nazw plików, które chcemy ignorować
- » Plik tworzymy w głównym katalogu repozytorium
 - reguły działają w głównym katalogu oraz podkatalogach
 - w dowolnym podkatalogu można umieścić inny .gitignore, który przykryje ten nadrzędny

FAQ git - .gitignore

```
> .gitignore
```

```
bin/
```

```
tmp/
```

```
build/
```

```
*.tmp
```

```
*.bak
```

```
*.swp
```

```
*~
```

```
!abc.tmp
```

```
# various IDE tempfiles
```

```
.idea
```

```
*.pydevproject
```

```
.gradle
```

```
.settings/
```

» Przykład:

– wykluczenie całego katalogu

FAQ git - .gitignore

> .gitignore

bin/

tmp/

build/

*.tmp

*.bak

*.swp

*~

!abc.tmp

various IDE tempfiles

.idea

*.pydevproject

.gradle

.settings/

» Przykład:

- wykluczenie całego katalogu
- plików tymczasowych typowych edytorów

FAQ git - .gitignore

> .gitignore

```
bin/  
tmp/  
build/  
*.tmp  
*.bak  
*.swp  
*~  
!abc.tmp  
# varoius IDE tempfiles  
.idea  
*.pydevproject  
.gradle  
.settings/
```

» Przykład:

- wykluczenie całego katalogu
- plików tymczasowych typowych edytorów
- konfiguracji środowisk programistycznych (IDE)

FAQ git - .gitignore

> .gitignore

bin/

tmp/

build/

*.tmp

*.bak

*.swp

*~

!abc.tmp

various IDE tempfiles

.idea

*.pydevproject

.gradle

.settings/

» Przykład:

- wykluczenie całego katalogu
- plików tymczasowych typowych edytorów
- konfiguracji środowisk programistycznych (IDE)
- katalogu z plikami tymczasowymi kompilatora (całego build systemu - cmake)

FAQ git - .gitignore

> .gitignore

bin/

tmp/

build/

*.tmp

*.bak

*.swp

*~

!abc.tmp

various IDE tempfiles

.idea

*.pydevproject

.gradle

.settings/

» Przykład:

- wykluczenie całego katalogu
- plików tymczasowych typowych edytorów
- konfiguracji środowisk programistycznych (IDE)
- katalogu z plikami tymczasowymi kompilatora (całego build systemu - cmake)
- wykluczenie wszystkich *.tmp ale nie abc.tmp

FAQ git - **stash**

```
> vim source.cc  
> git checkout master
```

» Jestem w trakcie pracy nad source.cc
ale muszę ugasić pożar w innej gałęzi

FAQ git - stash

```
> vim source.cc
```

```
> git checkout master
```

```
error: Your local changes to the following  
files would be overwritten by checkout:
```

```
source.cc
```

```
Please commit your changes or stash them  
before you switch branches.
```

```
Aborting
```

- » Jestem w trakcie pracy nad source.cc ale muszę ugasić pożar w innej gałęzi
- » Git chroni **moje zmiany** przed nadpisaniem
- » Nie chcę “commit-ować” nieskończonej pracy

FAQ git - stash

```
> vim source.cc
```

```
> git checkout master
```

```
error: Your local changes to the following  
files would be overwritten by checkout:
```

```
source.cc
```

```
Please commit your changes or stash them  
before you switch branches.
```

```
Aborting
```

- » Jestem w trakcie pracy nad source.cc ale muszę ugasić pożar w innej gałęzi
- » Git chroni **moje zmiany** przed nadpisaniem
- » Nie chcę “commit-ować” nieskończonej pracy
- » Najlepiej schować ją do **schowka** i dokończyć później (uwaga schowek jest na lokalnej maszynie)

FAQ git - stash

> **vim** source.cc

> **git** checkout master

error: Your local changes to the following files would be overwritten by checkout:

source.cc

Please commit your changes or stash them before you switch branches.

Aborting

> **git** stash

> **git** checkout...

> **git** stash list

> **git** stash apply

- » Jestem w trakcie pracy nad source.cc ale muszę ugasić pożar w innej gałęzi
- » Git chroni **moje zmiany** przed nadpisaniem
- » Nie chcę “**commit-ować**” nieskończonej pracy
- » Najlepiej schować ją do **schowka** i dokończyć później (uwaga schowek jest na lokalnej maszynie)
 - schowaj zmiany od ostatniego commita (przywróć tą wersję)
 - zrób coś w innej gałęzi, wróć
 - wylistuj wszystkie “schowki” (możesz odłożyć wiele zmian)
 - przywróć stan (jeden z listy odłożonych)

Dziękuję