



GIT

System kontroli wersji

Krzysztof Dziuban

Autor: Krzysztof Dziuban

Prawa do korzystania z materiałów posiada Software Development Academy



- Rys historyczny
- Podstawowe komendy
- Poznanie popularnych serwisów
- Zaawansowana obsługa
- Nakładki graficzne
- GitFlow

GIT - podstawowa zasada



IN CASE OF FIRE



Git Commit



Git Push



Git Out

Autor: Krzysztof Dziuban

Prawa do korzystania z materiałów posiada Software Development Academy



Rys historyczny i założenia

- 7.04.2005 – wydanie pierwszej wersji
- Stworzony przez Linusa Torvaldsa (w 3 dni!)
- Używany w rozwoju Linuxa, jako zastępstwo dla płatnego BitKeepera



Założenia:

- Przykład CSV czego nie robić
- Powinien być szybki
- Powinien być chroniony przed błędami w repozytorium
- Powinien być szybki



Repozytorium

Obiektowa baza projektu przechowująca wszystkie pliki i foldery projektu oraz dane gita (informacje o wersjach, commitach itp).

Commit

Pojedyncze zatwierdzenie zmian wprowadzonych do repozytorium

Branch

Odbicie od głównego pnia linii rozwoju

Rodzaje repozytoriów



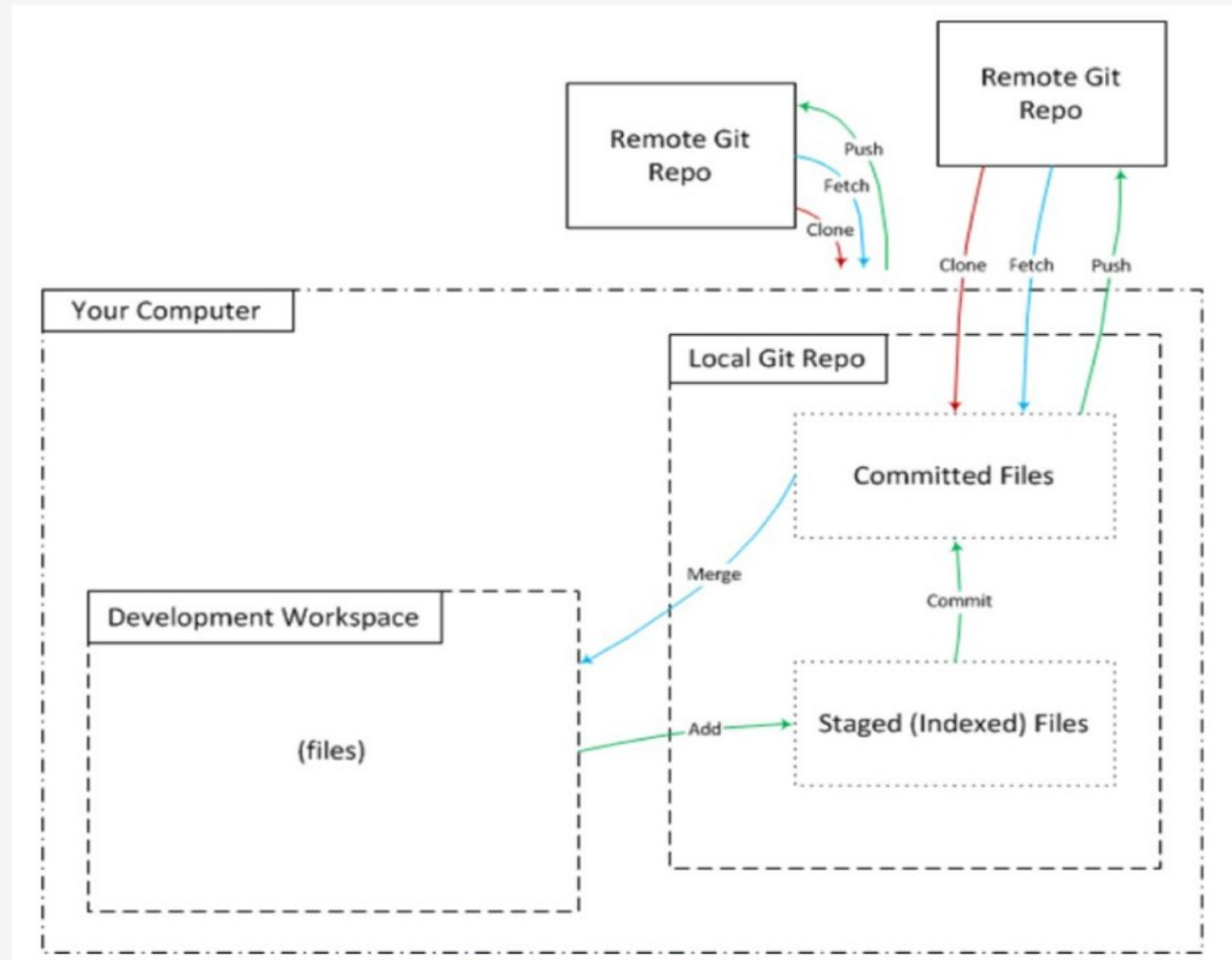
Repozytorium lokalne

- Przechowywane na komputerze użytkownika

Repozytorium zdalne

- Hostowane w sieci (GitHub, GitLab, Bitbucket itp)

GIT - schemat działania



źródło: <https://www.intertech.com/Blog/introduction-to-git-concepts/>



Pliki instalacyjne dostępne na:
<https://git-scm.com/downloads>

Dokumentacja:
<https://git-scm.com/doc>

Inicjalizacja lokalnego repozytorium



git init

Utworzenie lokalnego repozytorium w wybranym katalogu

- Używa się tylko raz!

git config

Konfiguracja środowiska

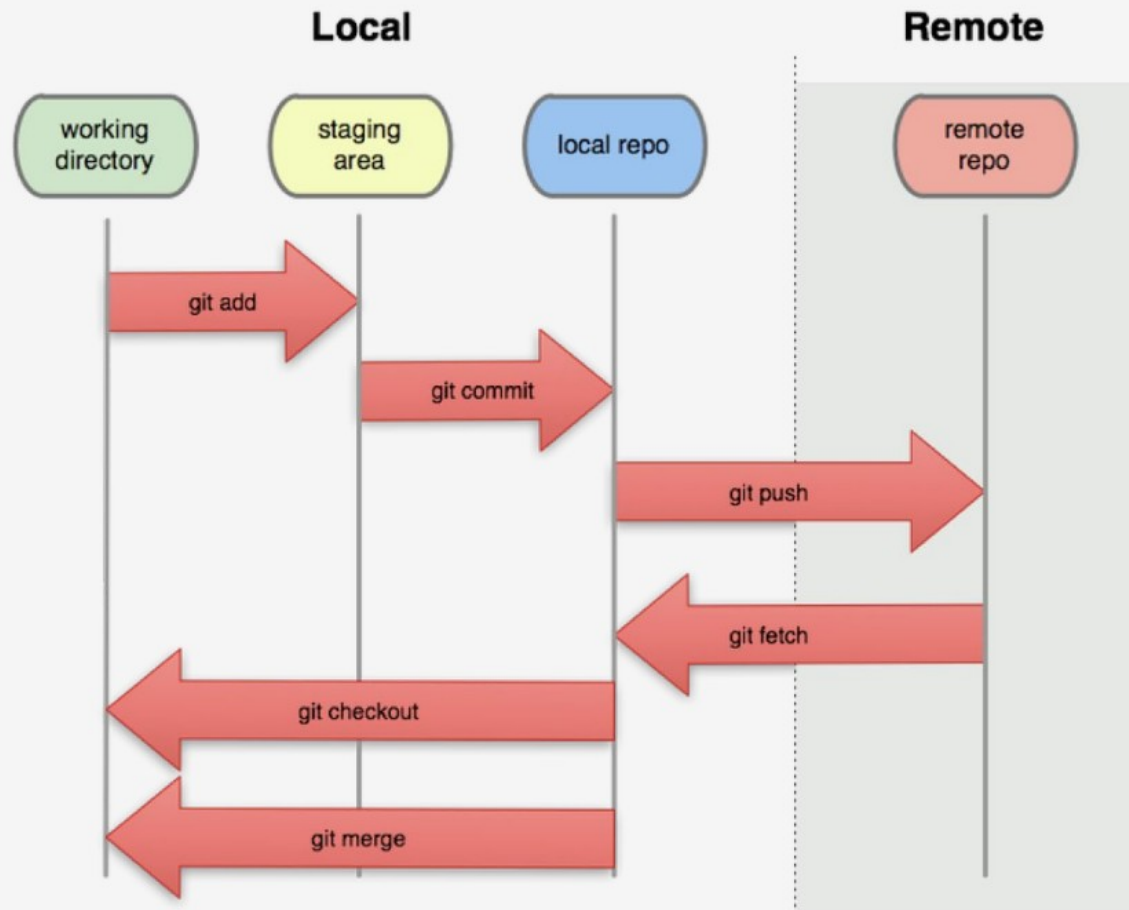
- `git config --global user.name USER_NAME`
- `git config --global user.email USER_EMAIL`

Ćwiczenie 1



- 1) Zainstalować git
- 2) Sprawdzić poprawność instalacji i wersję (`git --version`)
- 3) Utworzyć repozytorium w wybranym katalogu (`git init`)
- 4) Ustawić wybrane `user_name` i `user_email` (`git config`)

GIT - idea przechowalni



źródło: <https://greenido.files.wordpress.com/2013/07/git-local-remote.png?w=696&h=570>



git status

Sprawdzenie statusu projektu (lista zmodyfikowanych i nowych plików)

git log

Wyświetlenie listy commitów

git add

Dodawanie plików(zmian) do przechowalni

- `git add [wyrażenie]`

git reset

Usunięcie plików z przechowalni

- `git add [wyrażenie]`



git checkout -- <file_name>

Usunięcie zmian w pliku

git checkout <branch_name>

Przełączenie na istniejącego brancha

- `git checkout -b <branch_name>` - tworzy nowy branch

git commit -m "MESSAGE"

Dodawanie zmian do lokalnego repozytorium

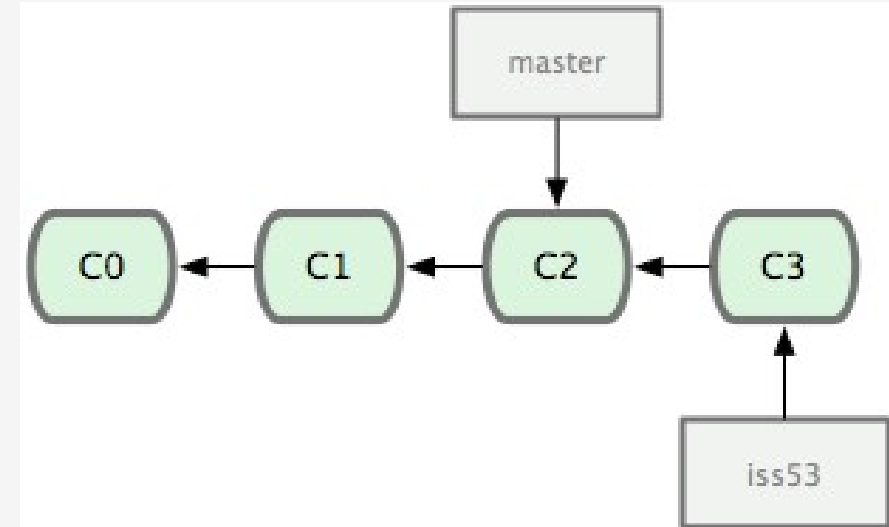
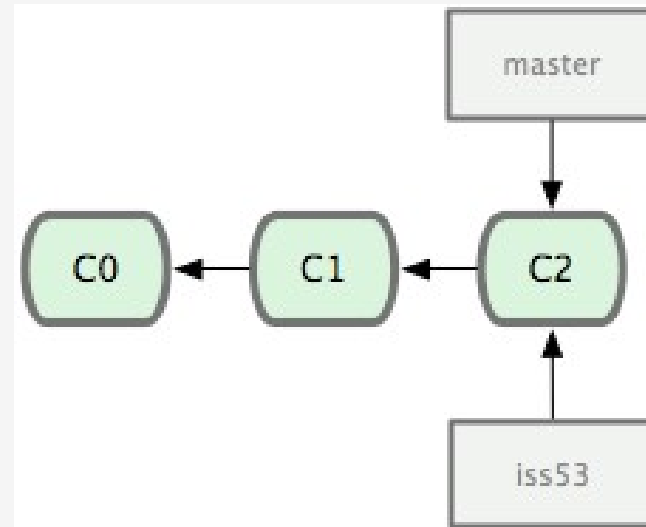
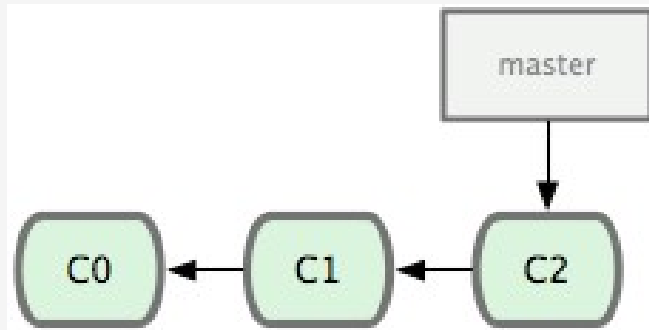
- `git add [wyrażenie]`

Ćwiczenie 2

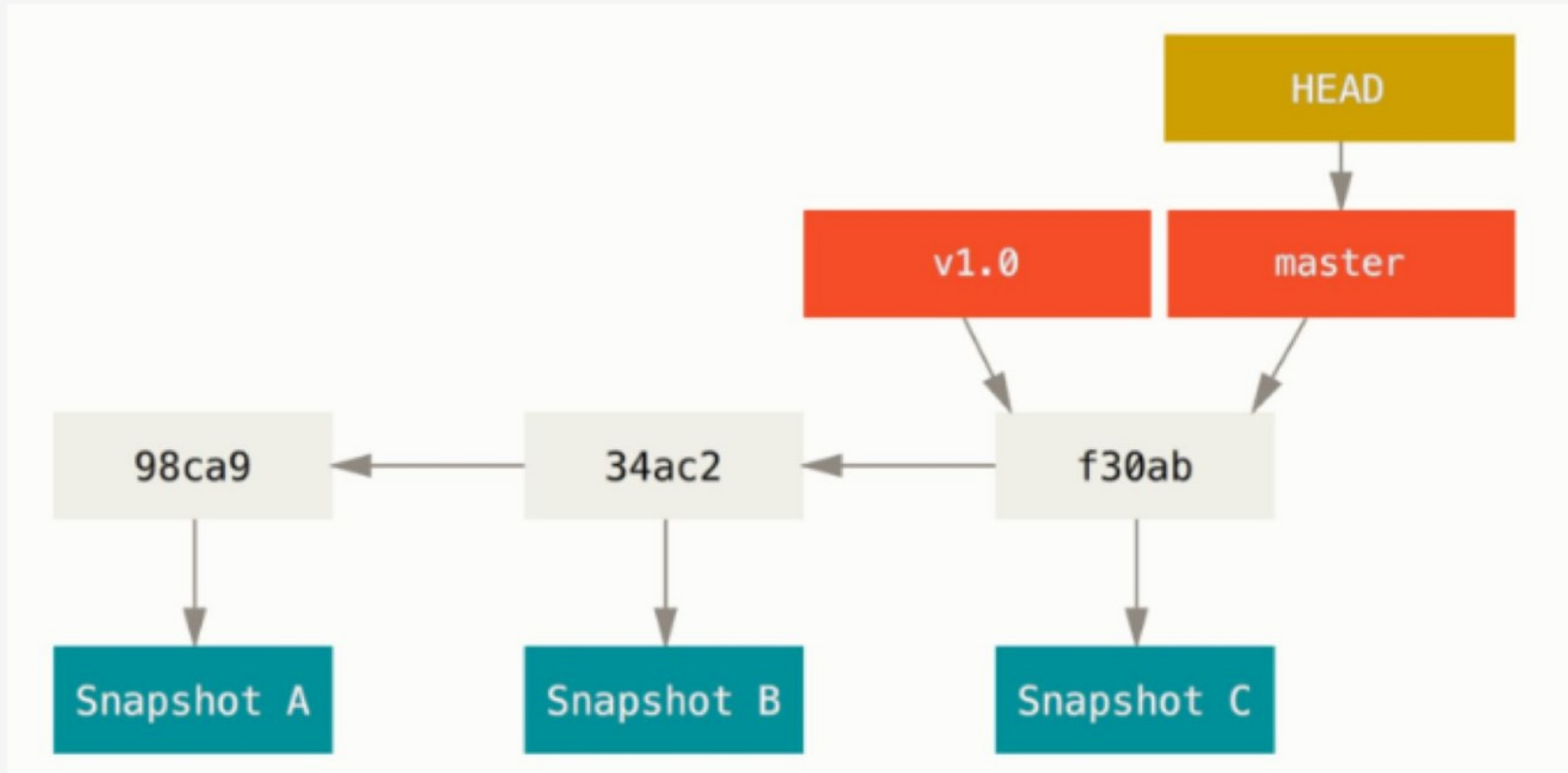


- 1) Sprawdzić stan repozytorium (git status)
- 2) Wylistować commity (git log)
- 3) Utworzyć plik README.txt
- 4) Ponownie sprawdzić stan repozytorium
- 5) Dodać plik do przechowalni (git add)
- 6) Usunąć plik z przechowalni (git reset)
- 7) Ponownie dodać plik do przechowalni i dodać commit (git commit)
- 8) Sprawdzić log

Podstawy rozgałęziania



Czym są branchy?



źródło: <https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>

HEAD vs head



- HEAD – wskaźnik na branch na której jesteście
- head – wskaźnik na ostatni commit (tip) na branchu

Jest tyle headów ile branchy, ale tylko jeden HEAD!!



git branch

Wyświetla listę lokalnych branchy

git branch -r

Wyświetla listę zdalnych branchy

git branch <branch_name>

Tworzy lokalnie branch

git branch -d <branch_name>

Usuwa lokalnie branch

- **git branch -D <branch_name>** - force delete!!

Ćwiczenie 3



- 1) Wyświetlić listę branchy (git branch)
- 2) Dodać nowy branch (git branch second)
- 3) Przełączyć się na utworzony branch (git checkout second)
- 4) Ponownie wyświetlić listę branchy i sprawdzić aktywny
- 5) Zmodyfikować plik README.txt i dodać do przechowania
- 6) Wykonać commit (git commit)
- 7) Sprawdzić status repozytorium



git stash

Wrzuca zmiany do schowka

git stash apply

Przywraca zmiany

git stash list

Wyświetla listę zmian ze schowka

git stash drop <stash>

Usuwa wybrany stash (jeśli brak nazwy, usuwa ostatni)

- **git stash clear** - czyści całą listę

Ćwiczenie 4



- 1) Zmodyfikować plik README.txt
- 2) Dodać plik temp.txt
- 3) Sprawdzić status
- 4) Wrzucić zmiany do schowka (git stash) i ponownie sprawdzić status
- 5) Sprawdzić listę ze schowka (git stash list)
- 6) Usunąć ostatnie zmiany ze schowka
- 7) Powtórzyć punkty 1-5 (z wyłączeniem 2), tym razem przywrócić zmiany (git stash apply)



```
git merge <branch_name>
```

Łączy aktualny branch z branch_name

Uwaga!!! Możliwe konflikty!!

Ćwiczenie 5



- 1) Stworzyć branch ala
- 2) Sprawdzić, czy aktualny branch to master
- 3) Dodać plik ala.txt
- 4) Zacommitować go
- 5) Przełączyć się na branch ala
- 6) Dodać ponownie plik ala.txt i zacommitować
- 7) Przełączyć się z powrotem na master i wykonać merge (git merge ala)
- 8) Rozwiązać konflikt i ponownie zacommitować plik (git commit -a)
- 9) Sprawdzić logi (powinien być dodatkowy commit z mergem)



git remote -v

Wyświetla linki do zdalnego repozytorium

git remote set-url origin <link>

Ustawianie urla do zdalnego repozytorium



git clone <link>

Sklonowanie(ściągnięcie) zdalnego repo do lokalnego

git fetch

Sprawdzenie różnic pomiędzy repozytorium zdalnym i lokalnym

git pull

Ściągnięcie danych z repozytorium zdalnego do lokalnego

Ćwiczenie 6



- 1) Założyć konto na <http://www.github.com>
- 2) Stworzyć repozytorium git_tutorial z plikiem README.md
- 3) Dodać link do repozytorium zdalnego do lokalnego (git remote)
- 4) Zaktualizować repozytorium lokalne (git pull)



git push

Przesłanie zmian do repozytorium zdalnego

git push -f

Nadpisywanie zdalnego repozytorium (tylko dla konkretnego brancha)

git push --set-upstream origin <branch_name>

Przesłanie zmian do zdalnego repozytorium z utworzeniem brancha

git push --delete <branch_name>

Usuwa branch na zdalnym repozytorium

Ćwiczenie 7



- 1) Przesłać dotychczasowe zmiany do zdalnego repozytorium (git push)
- 2) Stworzyć nowy branch `my_new_branch` i przełączyć się na niego
- 3) Dodać dowolny nowy plik, zacommitować lokalnie
- 4) Spróbować ponownie przesłać do zdalnego repozytorium
- 5) Przesłać do zdalnego repo z utworzeniem nowego brancha



Tag – odnośnik do specyficznego punktu w historii projektu

Rodzaje tagów:

- **lightweight** – podstawowy typ taga, zalecany jako prywatny
- **annotated** – rozszerzony tag, zawierający dodatkowe informacje (nazwa osoby tagującej, email, data itp)



git tag

Wyświetla listę tagów

git tag -l <wildcard>

Umożliwia wyszukiwanie po nazwach z wykorzystaniem metaznaków

git tag <tag_name>

Utworzenie lightweight tag

git tag -a <tag_name>

Utworzenie annotated tag



```
git tag <tag_name> <commit_hash>
```

Umożliwia przypisanie taga do konkretnego commita

```
git tag -f <tag_name> <commit_hash>
```

Umożliwia zastąpienie taga innym

```
git checkout <tag_name>
```

Przełączenie się na konkretnego taga

Uwaga !! – wykonane commity będą odłączone (dostępne będą tylko poprzez odwołanie do hasha – powinno się utworzyć nowego brancha przed commitowaniem zmian)



git tag -d <tag_name>

Usuwa taga

git push origin <tag_name>

Przesyła taga do zdalnego repozytorium

git push --tags

Przesyła zmiany wraz z wszystkimi tagami

Ćwiczenie 8



- 1) Utworzyć tag v0.9 (git tag)
- 2) Wylistować obecne tagi
- 3) Dodać nowy plik, zacommitować
- 4) Dodać annotated tag v1.0 (git tag -a)
- 5) Ponownie wylistować tagi, wyświetlić tylko ostatnio utworzony
- 6) Dodać tag v0.8 dla jednego z wcześniejszych commitów
- 7) Zastąpić tag v0.8 tagiem v0.8a
- 8) Podpiąć się pod taga v0.8a
- 9) Wrócić do master branch i usunąć tag v0.8a
- 10) Wykonać push wpierw dla taga v1.0, a potem dla pozostałych



git blame <file_name>

Wylistowanie zawartości pliku, wraz z commitami

git blame -L min,max <file_name>

Wyświetlenie linii z podanego zakresu

git blame -e <file_name>

Podmienia username na email

Rozwiązywanie problemów cd.



git blame -w <file_name>

Ignoruje zmiany związane z białymi znakami

git blame -M <file_name>

Wykrywa przeniesione lub skopiowane linie i wyświetla oryginalnego autora

git blame -C <file_name>

Wykrywa przeniesione lub skopiowane linie z innych plików i wyświetla oryginalnego autora

Inne przydatne komendy



git show

Pokazuje zawartość ostatniego commita

git show <commit_hash>

Pokazuje zawartość wybranego commita

git blame -C <file_name>

Wykrywa przeniesione lub skopiowane linie z innych plików i wyświetla oryginalnego autora

Inne przydatne komendy cd.



git cherry-pick <commit-hash>

Przenosi commita na aktualnego brancha

git rebase

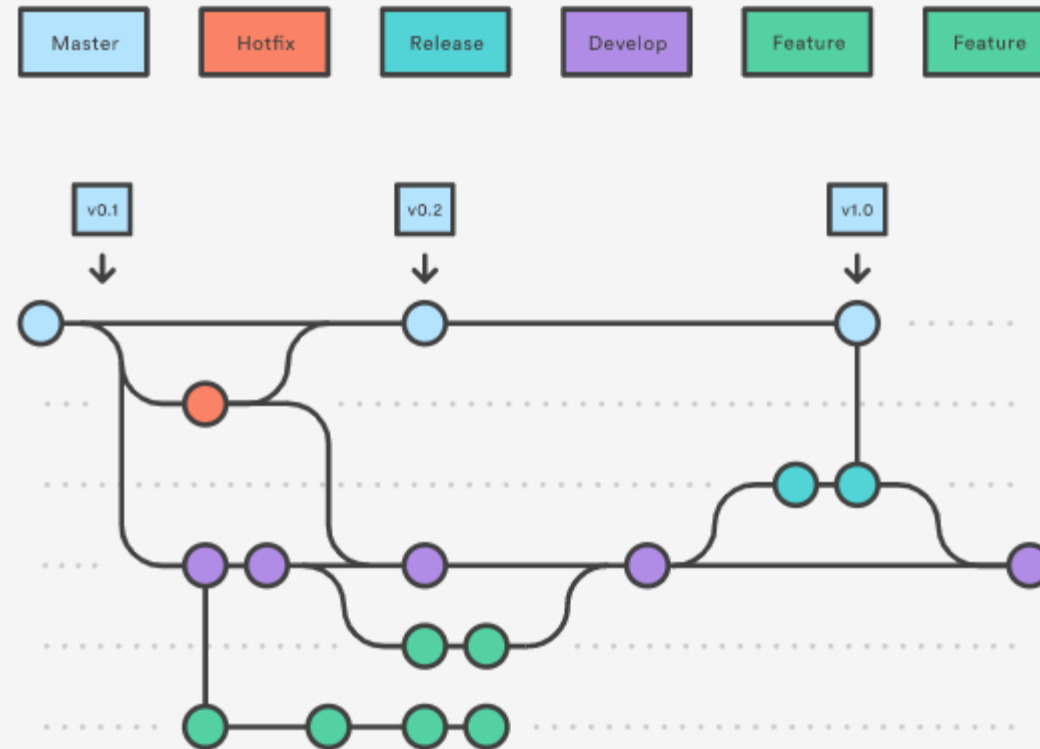
Zmienia bazę brancha

git rebase -i

Umożliwia edycję historii



GitFlow – metodyka pracy oparta o tworzenie określonego typu branchy





Typy branchy:

- master – główny branch
- develop – podstawowy branch developerski
- feature – branch poświęcony konkretnej historyjce/elementowi aplikacji
- task – branch poświęcony konkretnemu zadaniu
- release – branch poświęcony konkretnej wersji
- hotfix – branch poświęcony konkretnej poprawce



<https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud>



Dziękuję za uwagę