Akademia ADB

Wykład II – System kontroli wersji

Zalety stosowania systemu kontroli wersji (VCS)

Żaden poważny projekt nie obejdzie się bez kontroli wersji:

- Bez VCS współpraca w grupie jest praktycznie niemożliwa: w plikach jest chaos
- Dzięki VCS projekt jest uporządkowany: istnieją wersje i wspólne punkty odniesienia (funkcja referencyjna)
- Można łatwo wycofywać zmiany (wykrywanie przyczyn regresu)
- Funkcja dokumentacyjna – co się zmieniło i dlaczego?
- Kopia zapasowa

Najbardziej znane systemy kontroli wersji

Często spotykane systemy wersji to:

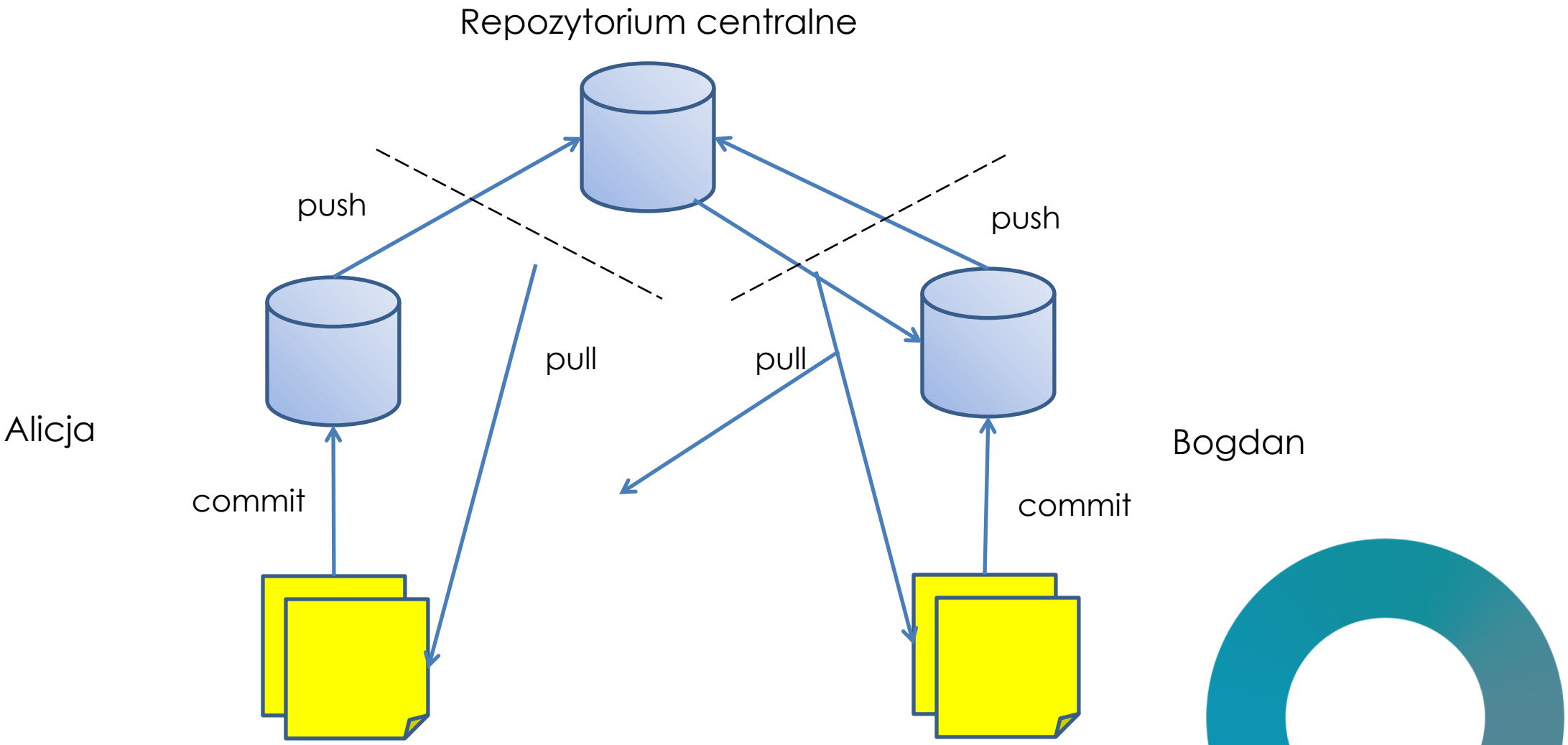
- Git
- Subversion (SVN)
- Mercurial
- CVS
- Perforce
- MS Team Foundation Version Control

Git

Git to prawdopodobnie najczęściej używany obecnie VCS:

- Stworzony prze Linusa Torvaldsa
- System jest rozproszony – można pracować offline
- Praca w złożonym projekcie jest zdecydowanie prostsza niż np. w SVN
- Jest szybki
- Jest bardzo elastyczny
- Jest darmowy

Schemat działania Git



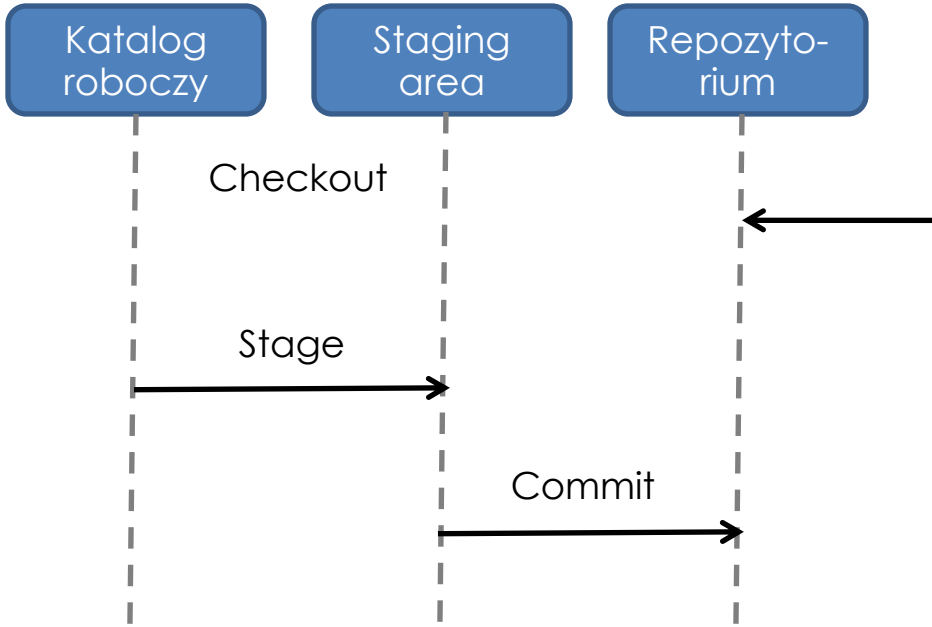
Klonowanie i inicjalizacja

Klonowanie służy do pobrania aktualnego stanu repozytorium z „centrali”. Wraz z kopią pobierana jest cała historia. Polecenie linii komend to `git clone url`.

Stworzenie nowego repozytorium w istniejącym katalogu to `git init`.

Do sprawdzania stanu repozytorium służy komenda `git status`.

Wprowadzanie zmian



Git używa trzystopniowego modelu wprowadzania zmian.

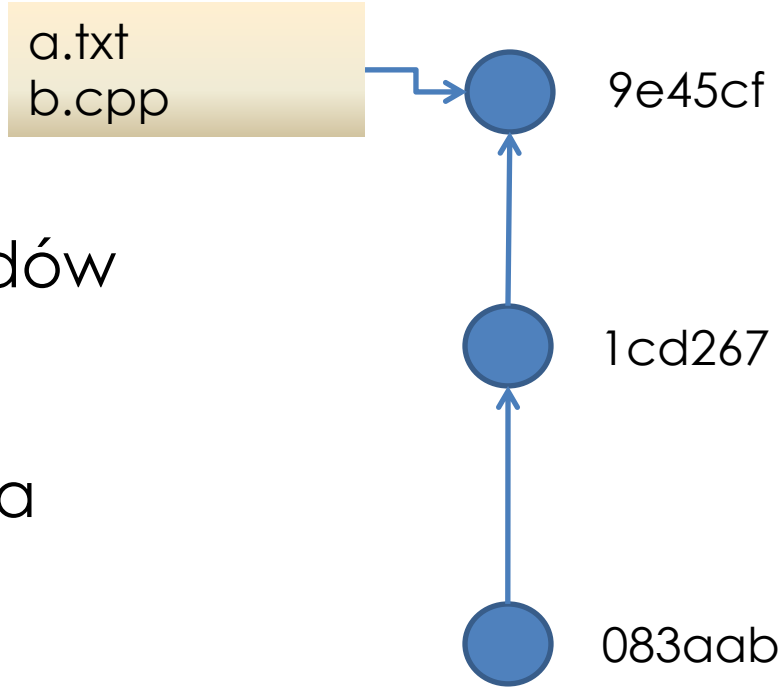
- Na początku plik jest po prostu zmieniony
- Po wydaniu polecenia `add`, zostaje uznany za kandydata do commitowania
- Po wydaniu polecenia `commit`, pliki-kandydci wchodzą do zawartości commitu. Ważne przełączniki to `-m` i `-a`.

Pliki nie opuszczają komputera roboczego!

Zmiany sprawdzamy dokładnie poleceniem `diff`.

Czym jest commit?

- Commit to grup zmian wprowadzonych jednocześnie do repozytorium.
- W git, commity identyfikuje się za pomocą kodów hash.
- Do przeglądania historii służy polecenie `log`
- Można powiedzieć, że commit to podstawowa jednostka pracy z systemem kontroli wersji



Ignorowanie plików

Od czasu do czasu trafimy na plik, który nie powinien trafić do repozytorium. Musimy założyć wtedy plik .gitignore i wpisać do niego nazwę pliku w osobnej linii. Dopuszcza się komentarze i maski.

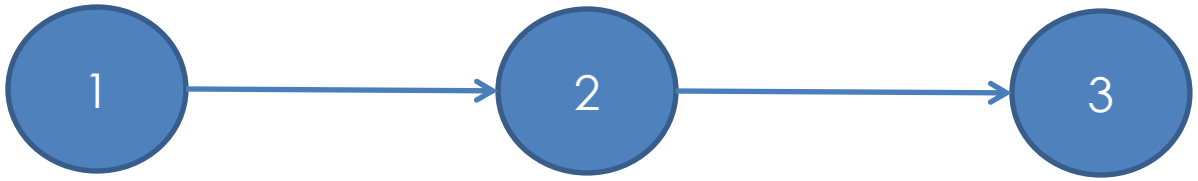
```
Haslo.txt

#Komentarz
*.a
doc/*.pdf
```

Cofanie zmian

Zmiany można cofać na kilka sposobów:

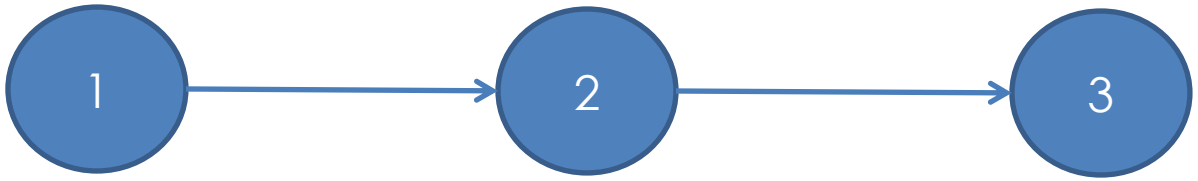
- `checkout` – prosta operacja zmieniająca plik na dysku, ale bardzo niebezpieczna w innych przypadkach
- `stash` – „upycha" zmiany na boku
- `revert` – odwraca zmiany wprowadzone przez dany commit bez zmieniania historii, poprzez dodanie nowego commita
- `reset` – odwraca zmiany wprowadzone przez dany commit ze zmianą historii. Dla pliku – wstawia wersję do stage'a

Revert

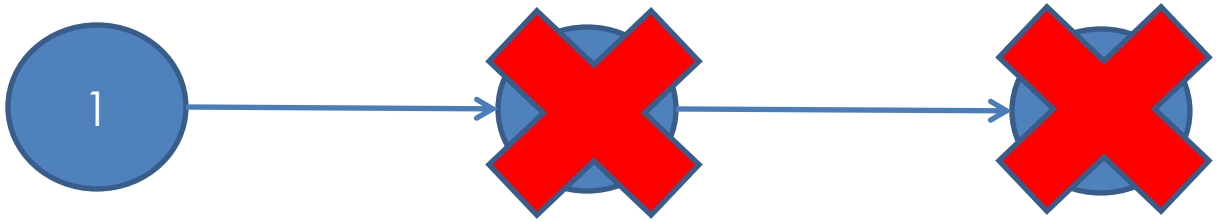


git revert HEAD~2:

Reset



git reset HEAD~2:



Inne komendy Git

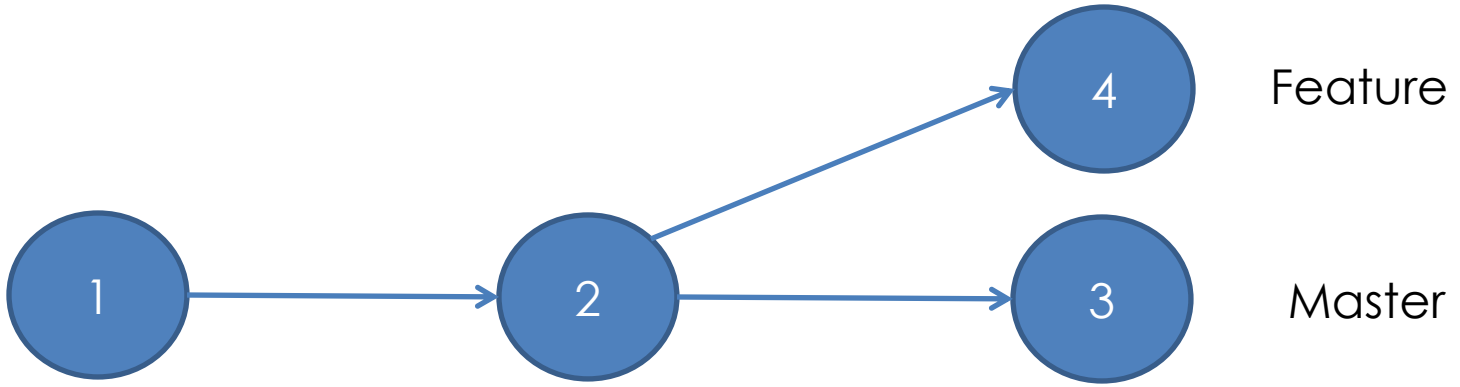
Oprócz wcześniej wymienionych istotne są:

- `rm` – usuwa plik
- `mv` – przesuwa plik

Gałęzie (*branches*)

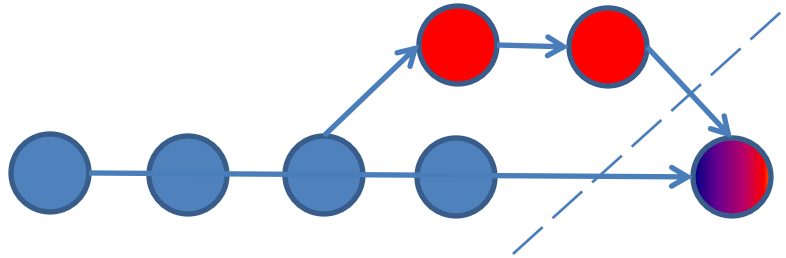
Gałęzie to niezależne kierunki rozwoju kodu:

- Do tworzenia gałęzi używamy polecenia `branch nazwa`
- Do przełączania się między gałęziami służy polecenie `checkout nazwa`
- Skrót do założenia i przełączenia: `branch -d nazwa`
- Zwyczajowo główna gałąź nazywa się `master`



Łączenie gałęzi (*merge*)

`merge` łączy stan dwóch gałęzi rozwoju:



- Polecenie dołącza wskazaną gałąź do obecnej gałęzi
- Jeżeli linie rozwoju się nie rozeszły, następuje proste przesunięcie wskaźnika (*fast-forward*)
- W przeciwnym wypadku, powstaje commit łączący (*3-way merge*)
- Czasami dochodzi do konfliktów, które trzeba rozwiązać ręcznie
- Dla mniejszych poprawek stosuje się zwykle polecenie `rebase` przesuwające moment rozłączenia

Plik z konfliktem

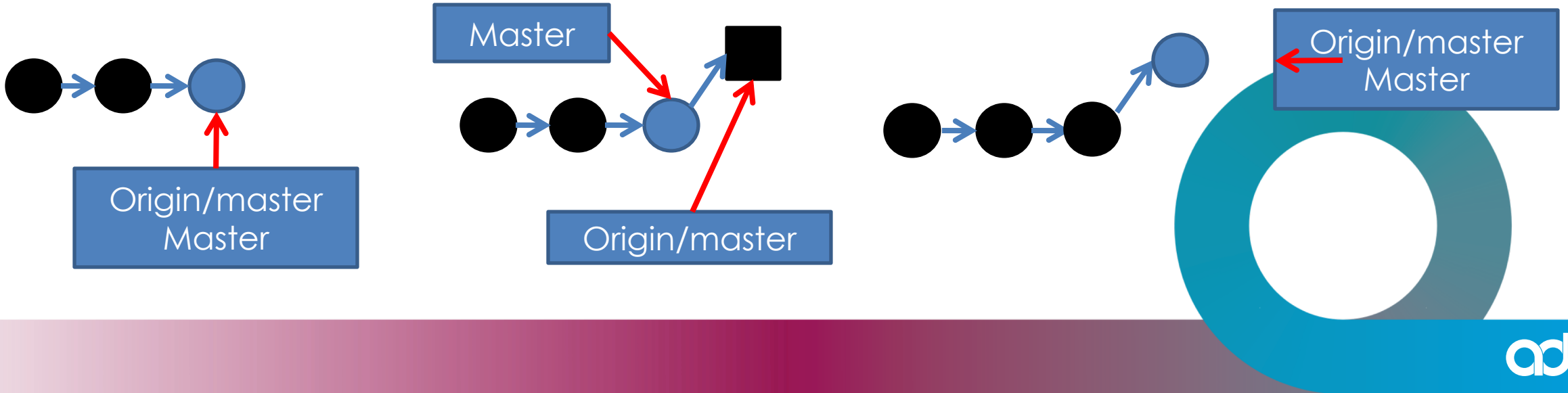
Liczba planet w Układzie Słonecznym to
<<<<<<< HEAD
dziewięć
=======
osiem
>>>>>>> branch-a

Repozytoria zdalne

- Repozytoria zdalne zbierają commity z repozytoriów lokalnych
- Do operowania nimi służy komenda `remote` z przełącznikami `-v`, `add`, `rm`
- Podczas klonowania tworzone jest domyślne repozytorium zdalne `origin`
- Liczba repozytoriów jest nieograniczona
- Repozytoria zdalne muszą posiadać swój URL

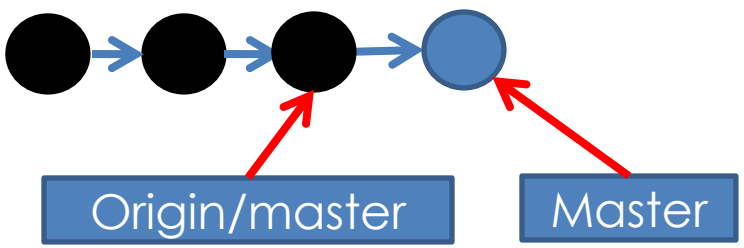
Pobieranie commitów

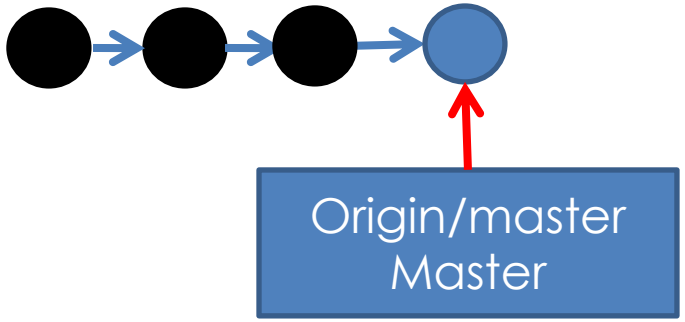
- Polecenie `fetch` pobiera dane jako boczna gałąź bez zmiany obecnej gałęzi roboczej
- Gałęzie zdalne mają nazwy w formacie nazwa-zdalnego-repozytorium/nazwa-gałęzi, np. `origin/master`
- Polecenie `pull` łączy zmiany ze zdalnego źródła z obecną gałęzią. Przełącznik `--rebase` próbuje dokonoać przebazowania.



Wysyłanie zmian do zdalnego repozytorium

- Polecenie `push <remote> <branch>` wysyła zmiany do zdalnego repozytorium.
- Push zostanie zaakceptowany tylko jeśli zmiany są typu fast-forward. W przeciwnym wypadku wymagany jest pull.





Pull requests i *fork*

- Obydwa terminy odnoszą się do pracy w systemach w stylu BitBucket i GitHub
- *Pull request* wysyła się gdy chcemy wprowadzić zmiany do repozytorium i nie mamy dotego uprawnień.
- *Fork* to idealna kopia oficjalnego repozytorium projektu. Kopia ta staje się zdalną kopią, na której pracuje programista i z której pochodzą *pull request*'y.

Literatura

- ProGit
  https://git-scm.com/book/en/v2
- Przewodnik Atlassiana:
  https://www.atlassian.com/git/tutorials/
- Bitbucket
  https://www.atlassian.com/software/bitbucket

Koniec

Dziękuję za uwagę

adbglobal.com