Podstawowe polecenia dla konsoli

pwd -> wypisuje ścieżkę obecnego katalogu (od 'print working directory')

---

ls – listuje katalog

- ls – listuje katalog . (ls .)
- ls plik1 plik2 plik3 – listuje tylko wymienione pliki
- ls *.txt – wypisze wszystkie pliki o nazwie kończącej się na '.txt'
- ls katalog1 katalog2 – listuje wymienione katalogi
- ls -l – szczegółowa lista
- ls -a – wypisuje również ukryte pliki (czyli te których nazwa zaczyna się kropką)
- ls -R – listuje katalogi rekursywnie (czyli wyświetla również zawartość podkatalogów)
- ls -d – wyświetla tylko nazwy katalogów, tak jak zwyczajnych plików, czyli nie listuje ich zawartości

---

history -> historia komand

---

cd – zmienia aktualny katalog (od 'change directory')

- cd dirname – zmienia aktualny katalog na 'dirname'
- cd dir1/dir2/dir3 – wchodzi do katalogu 'dir3', który jest w katalogu 'dir2', który jest w 'dir1'
- cd – z dowolnego miejsca, zmienia katalog na domowy
- cd .. – przechodzi do katalogu o jeden wyższego w drzewie katalogów niż obecny
- cd /home/dir – z dowolnego miejsca, przechodzi do katalogu zaczynając od początku drzewa: /
- cd - – przechodzi do poprzedniego katalogu

---

touch plik -> stwórz pusty plik

- touch – zmienia czas dostępu i modyfikacji pliku, lub jeśli plik nie istnieje - tworzy go.

echo -> printuje text z ">" twozy plik

---

mkdir – tworzy katalog

- mkdir moj_nowy_katalog
- mkdir /home/users/ja/moj_nowy_katalog

---

cat – wypisuje wszystkie podane mu pliki na standardowe wyjście

- cat plik – jeśli nie przekierujemy standardowego wyjścia do innego pliku (>, >>) lub programu (|), to wypisze plik na ekran
- cat plik1 plik2 plik3 – wypisze po kolei zawartość wszystkich plików

---

cp – kopiuje plik

- cp plik1 plik2 – stworzy ./plik2 identyczny z plik1
- cp plik3 ../katalog/ – stworzy plik ../katalog/plik3
- cp pom.* podkatalog/ – kopiuje wszystkie pliki zaczynające się na 'pom.' do ./podkatalog/
- cp plik5 ~/katalog/jakis/pliczek – stworzy plik ~/katalog/projekt/plik

---

mv – przesuwa plik (tym samym służy również do zmiany nazwy)

- mv plik1 plik2 – zmieni nazwę pliku z ./plik1 na plik2
- mv plik3 ../katalog/jakis/ – przesunie plik do ../katalog/jakis/plik3
- mv plik4 podkatalog/ – przesunie plik ./podkatalog/plik4
- mv plik5 ~/katalog/ – przesunie i zmieni nazwę ~/katalog/projekt/plik5

---

rm – usuwanie

- rm plik - kasuje plik
- rm -r katalog – kasuje wszystko w katalogu i wszystkie jego podkatalogi (--recursive)
- rm -f plik – nie pyta się czy skasować (--force)

---

rmdir – usuwa pusty katalog

- rmdir katalog

---

ctr + c -> pszerywa działania programu q

Polecenie dla systemu kontroli GIT

Wstępna konfiguracja Gita

- plik /etc/gitconfig: zawiera wartości zmiennych widoczne dla każdego użytkownika w systemie oraz dla każdego z ich repozytoriów. Dostęp do konfiguracji jest możliwy za pomocą opcji --system do polecenia.

- plik ~/.gitconfig: lokalizacja dla danego użytkownika. Dostęp do konfiguracji jest możliwy za pomocą opcji --global

- plik konfiguracyjny w katalogu git (tzn. .git/config) bieżącego repozytorium: zawiera konfigurację charakterystyczną dla tego konkretnego repozytorium.

git config --global user.name "Kursant" -> est odpowiedzialne za ustawienie nazwy użytkownika.

git config --global user.email kursant@sda.pl -> jest odpowiedzialne za ustawienie adresu email.

git config --list -> wyświetlenia bieżącej konfiguracji należy skorzystać z polecenia

---

Aliasy (skróty które będą odpowiadać oryginalnej wersji)

- git config alias.co checkout
- git config alias.br branch
- git config alias.ci commit
- git config alias.st status

od tej pory można korzystać z skróconych poleceń:

- git st -> weryfikuje stan plików w repozytorium

Inicjalizacja repozytorium

git init -> Tworzenie repozytorium lokalne

Dodanie plików do repozytorium

git add "nazwa pliku" -> dodaję plik do staging area (poczekania) git add "." -> do poczekalni dodawany jest każdy plik

---

Sprawdzenie stanu plików

git status -> weryfikuje stan plików w repozytorium

- git status -s -> umożliwia wyświetlenie skróconej formy stanu repozytorium.

Lista różnic

git diff -> pozwala na zwrócenie informacji, które dokładnie linie zostały dodane, a które usunięte

- diff --staged - > pozwala na wyświetlenie szczegółowych informacji na temat plików.

Commit plików

git commit -m "treść wiadomości" -> zatwierdzone zmiany z definiowaniem treści wiadomości w momencie zatwierdzania zmian

git commit -a -m "treść_wiadomości" -> Zatwierdzanie zmian z pominięciem poczekalni.

Historia commitów

git log -> pozwala wypisać listę commitów w kolejności od najnowszego do najstarszego

- git log --patch / git log -p -> wyświetla zmiany wprowadzone w każdym commicie
- git log --stat -> wyświetla parę statystyk, np. liczbę zmienionych linii
- git log --oneline -> wyświetla każdy commit w jednej, osobnej linii

---

git reset HEAD "file" -> (użyte na pliku) działa odwrotnie do git add, cofa pliki z staging area do working directory

---

git checkout --"file" -> przywrócenia pliku do stanu z ostaniego commita

---

git rm -> usuwa podany plik/katalog

---

git mv -> służy ono do zmiany nazwy pliku w repozytorium.(Git nie śledzi bezpośrednio przesunięć plików)

Cofanie wprowadzonych zmian

git revert "commit" -> pozwala na odwrócenie zmian wprowadzonych w konkretnym commicie.
Polecenie to tworzy tym momencie nowy commit, który swoimi zmianami odwraca działanie zmian wprowadzonych w commicie, który chcemy usunąć.

---

git remote -> wypisuje skrócone nazwy wszystkich określonych serwerów dla tego repozytorium

git remote -v -> wypisuje skrócone nazwy wszystkich określonych dla tego repozytorium serwerów. Dodatkowo wyświetlony zostanie adres URL przypisany do tych skrótów.

git remote show "nazwa-zdalnego-repo" -> przegląd zdalnych repozytoriów

---

git checkout "identyfikator_gałęzi" -> przełączanie gałęzi

Powyższe polecenie wykonało dwie akcje:

- przesunęło wskaźnik HEAD na wskazany branch
- przywróciło pliki w katalogu roboczym do stanu z migawki, na którą wskazuje branch

---

git branch -> bez dodatkowych argumentów umożliwia zwrócenie wszystkich lokalnych gałęzi w ramach systemu kontroli wersji

Symbol * definuje branch, na który wskazuje wskaźnik HEAD.

- git branch "nazwa_brancha" -> pozwala utworzyć nową gałąź
- git branch -d nazwa_gałęzi -> usunięcie gałęzi
- git checkout "nazwa_brancha" -> pozwala przełączyć się na wybrany branch
- git checkout -b nazwa_gałęzi -> pozwala utworzyć nową gałąź oraz przesunięcie wskaźnika HEAD na stworzoną gałąź.
- git branch -M -> nowe inie dla branch w kturej znajduje sie
- git branch -a -> wyświetla wszystkie gałęzie, w tym te w zdalnym repozytorium
- git branch -vv -> pokazuje, w jaki sposób oddział lokalny jest powiązany ze zdalnym

---

git merge "nazwa_brancha" -> do brancha, na którym obecnie się znajdujemy, zostaną dołączone zmiany z wybranego brancha.

---

git clone "URL" -> uzyskania kopii istniejącego już repozytorium

---

Pobieranie i scalanie zmian ze zdalnych repozytoriów

git fetch "remote-name" -> pobierania danych ze zdalnego repozytorium

git pull -> pobranie zmian i ich automatyczne scalenie Polecenie to wykonuje poniższe operacje:

- pobieranie danych (fetch)
- scalenie danych z lokalnymi plikami (merge)

git pull -> umożliwia pobieranie danych z serwera, na bazie którego oryginalnie stworzone zostało repozytorium i następnie dokonywana jest próba automatycznego scalenia zmian z kodem roboczym, nad którym aktualnie, lokalnie wykonywana jest praca.

---

git push "nazwa-zdalnego-repo" "nazwa-gałęzi" -> Wysyłanie zmian do zdalnego repozytorium

W przypadku, w którym dwie osoby pracowały na tej samej gałęzi i jedna osoba wysłała zmiany na serwer, to kolejna osoba, która chce udostępnić swoje zmiany, podczas próby ich wysłania zostanie odrzucona.

Niezbędne będzie wcześniejsze pobranie i scalenie zmian ze zdalnego repozytorium.

Rebase

git rebase "nazwa_brancha" -> polecenie pozwalające zamienić podstawę brancha, na którym jesteśmy, na branch podany w poleceniu.

git rebase -i "nazwa_brancha" -> uruchamia interaktywny 'kreator' rebase'a.

Można w nim dokonać zmian w commitach, m.in. złączyć commity razem, zmienić ich message.

Zalety i wady rebase:

Zalety:

- Upraszcza potencjalnie złożoną historię.
- Manipulowanie pojedynczym commitem jest łatwiejsze.
- Zapobiega pojawianiu się merge commitów w repozytorium.

Wady:

- Zbieranie kilku commitów w jeden może zniekształcić kontekst prac.
- Wykonanie rebaseÈa w publicznym repozytorium może być niebezpieczne, jeżeli pracuje się zespołowo.
- Trochę więcej pracy - trzeba korzystać z rebase, aby branch był zaktualizowany.

Stash

git stash -> (ang. schowek) polecenie pozwalające "odłożyć" wprowadzone zmiany na bok, bez konieczności ich commitowania.

git stash list -> pozwala zobaczyć listę zapisanych zmian w schowku.

git stash apply -> pozwala ponownie nałożyć zmiany ostatnio odłożone. Opcja ta tylko integruje zmiany, nadal będą widniały na liście zmian w schowku.

git stash pop -> nakłada ostatnio odłożone zmiany, a następnie usuwa je z listy zmian.

Rozwiązywanie konfliktów

Jeśli ten sam plik został zmieniony w różny sposób w obu scalanych gałęziach, Git nie będzie w stanie scalić ich samodzielnie.

Git nie zatwierdzi automatycznie zmiany scalającej. Wstrzyma on cały proces do czasu rozwiązania konfliktu.

Stan plików można zweryfikować za pomocą polecenia git status.

Wszystkie pliki, które spowodowały konflikty i nie zostały automatycznie rozstrzygnięte, są wymienione jako „unmerged” (niescalone).

W przypadku takich zmian Git dodaje do problematycznych plików standardowe znaczniki rozwiązania konfliktu.

Dzięki temu można te pliki bez problemu otworzyć i następnie rozwiązać konflikty ręcznie.

Wynikiem otwarcia pliku będzie następujący znacznik:

- wszystko poniżej <<<<<<< HEAD określa wszystkie zmiany, które znajdowały się w pliku w momencie scalania gałęzi
- wszystko poniżej sekcji ======= to wszystkie zmiany, które były wprowadzone na gałęzi, która była scalana

W przypadku chęci rozwiązania konfliktów za pomocą narzędzia graficznego można skorzystać z polecenia

git mergetool -> Powyższe polecenie umożliwia wybór narzędzia do rozwiązywania konfliktów

Po wykonaniu wszystkich niezbędnych poprawek oraz dodaniu zmian do poczekalni, finalnie trzeba zatwierdzić zmiany za pomocą polecenia 'git commit', co w rezultacie umożliwia stworzenie tzw. merge commit

---

plik .gitignore -> pozwala na zdefiniowanie wzorców, których git poszukuje w ścieżkach plików.

Jeżeli ścieżka pliku pasuje do któregokolwiek ze wzorca, git ignoruje ten plik we wszystkich poleceniach.

In [2]:
```
! jupyter nbconvert --to webpdf --allow-chromium-download
System_Git.ipynb
```

```
[NbConvertApp] Converting notebook System_Git.ipynb to webpdf
[NbConvertApp] Writing 153852 bytes to System_Git.pdf
```