Repozytorium - folder, którego zawartość jest kontrolowana przez Git. Zawiera folder .git, w którym są zawarte informacje o śledzonych plikach oraz konfiguracji. Lokalizacja repozytorium nie ma znaczenia oraz można je dowolnie przenosić oraz kopiować. Wszystkie pliki oraz foldery poza folderem .git są nazywane folderem roboczym repozytorium.

Zatwierdzenie zmian w repozytorium (commit) polega na zapisaniu aktualnego stanu katalogu w systemie kontroli wersji. W ten sposób powstaje rewizja, z którą powiązane są informacje takie jak identyfikator (skrót SHA1), dane osoby, która ją wykonała, data i godzina, informacje o zmodyfikowanych plikach oraz o rewizjach nadrzędnych.

Przekazanie do polecenia git parametru *--help* otworzy stronę z dokumentacją tego polecenia, np. *git add --help*.

W celu oddzielenia poleceń od nazw plików można zastosować znak --. Może się zdarzyć, że plik posiada taką samą nazwę jak komenda i nie będzie możliwe odwołanie się do niego w inny sposób.

Konfiguracja:

Istnieją trzy poziomy konfiguracji:

- system, globalny dla danego systemu (--system),
- global (--global), globalny dla danego użytkownika
- local specyficzny dla repozytorium.

Po instalacji należy skonfigurować nazwę użytkownika oraz adres email, które są używane podczas zatwierdzania zmian:

```
git config --global user.name "Imie Nazwisko"
git config --global user.email you@example.com
```

Inne przydatne komendy:

`git config -l` - wyświetlenie aktualnej konfiguracji. Jest to m. in. jeden ze sposobów sprawdzenia śledzonych gałęzi. Przykładowo dla gałęzi master mogą wystąpić takie wpisy:

Branch.master.remote=origin
Branch.master.merge=refs/heads/master

`git config --get <property>` - sprawdzenie wartości danej właściwości

`git config core.editor <editor>` - konfiguracja edytora używanego przez GIT np. podczas zatwierdzania rewizji. W przypadku systemu Windows oraz Notepad++ może to być: `git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"`

`git config --gloabl core.autocrlf false` - wyłączenie globalnej konwersji znaków końca linii. Właściwość *core.eol* ustala, jaka wartość znaku końca wiersza ma być odtwarzana podczas operacji checkout. Możliwe wartości to *lf, crlf* oraz *native*.

`git config --global core.autocrlf input` - operacja commit powoduje konwersję z *crlf* na *lf* a checkout nie powoduje żadnej zmiany. Automatyczna konwersja znaków końca linii jest włączona

jedynie dla plików z atrybutem text. Za pomocą atrybutu eol możliwe jest indywidualne ustalenie znaków końca linii dla różnych plików podczas operacji checkout.

`git config --global merge.conflictstyle diff3` - zmiana domyślnego trybu nanoszenia znaczników konfliktu. Dodawana jest część zawierająca stan ze wspólnego przodka dwóch rewizji

Tworzenie nowego repozytorium:

`git init` - stworzenie nowego repozytorium. Repozytorium zostanie utworzone w aktualnym folderze. Jeśli jako parametr zostanie przekazana ścieżka, to repozytorium będzie utworzone we wskazanej lokalizacji. Komenda ta nie uszkadza istniejących repozytoriów. Polecenie to można wywołać w niepustym folderze. Aktualnie istniejące pliki nie zostaną zmodyfikowane oraz nie będą śledzone w systemie git.

`git clone <address>` - tworzy lokalną kopię repozytorium wskazywanego przez address. Możliwe jest podanie drugiego parametru, lokalizacji repozytorium. Jeśli nie podamy lokalizacji zostanie utworzony nowy folder o nazwie odpowiadającej nazwie repozytorium. Przykładowy adres: *https://github.com/jquery/jquery.git*

Repozytoria surowe (bare) są pozbawione obszaru roboczego i zawierają tylko informacje z folderu .git. Można je stworzyć komendami git clone --bare lub git init --bare

Wyświetlanie informacji o historii projektu:

Większość wymienioncyh w komendach parametrów można łączyć ze sobą. Przedstawione komendy przyjmują dodatkowe parametry umożliwiające ograniczenie zakresu przeszukiwania:

- *<src>..<src2>* - wszystkie rewizje dostępne z <src2> ale nie dostępne z <src>, np. wyświetlenie rewizji z gałęzi *feature1*, które nie są połączone z gałęzią *master*: `git log master..feature1`. Możliwe jest pominięcie *<src>* lub *<src2>.*
- *<src> <src2> --not <src3>* - tak samo jak w pierwszym przypadku, jednak możliwe jest podanie wielu rewizji przed parametrem --not. Parametr ten można zastąpić znakiem ^.
- *<src>…<src2>* - wszystkie rewizje, które są dostępne z *<src>* lub *<src2>,* ale nie z obu jednocześnie. Przydatnym parametrem jest wtedy *--left-right*, który umożliwia rozróżenienie, z której reiwzji dany rekord jest osiągalny

`git shortlog` - wyświetla skrócone logi odnośnie projektu

`git shortlog -s -n` - wyświetlenie uczestników projektu

`git log --pretty=oneline` - Wyświetlenie wszystkich rewizji w repozytorium. Taki sam skutek ma parametr *--oneline*

`git log -<number>` - wyświetla *<number>* rewizji z repozytorium

`git log --min-parent=3` - wyświetlenie wszystkich rewizji posiadających co najmniej trzech rodziców.

`git log -- <filename>` - wyświetlenie rewizji, w których został zmodyfikowany dany plik

`git log -p` – wyświetlenie także zmian dokonanych w danej rewizji

Git log posiada wiele parametrów, za pomocą których można formatować zakres rewizji oraz ograniczać ilość wyświetlanych danych, np.

`git log --since="yyyy-mm-dd" --pretty=email --abbrev-commit --abbrev=4 --author=Janek` – wyświetlenie rewizji utworzonych po wskazanej dacie przez wymienionego autora skracając długość sumy SHA-1 do 4 znaków w formacie email

`git log --pretty=oneline --abbrev-commit --abbrev=4 -10 --graph v2.0.10` - wyświetlenie historii rewizji począwszy od rewizji wskazywanej przez znacznik v2.0.10 w postaci grafu. Zastosowano ograniczenie do 10 rewizji

Historię zmian można wyświetlić za pomocą gui używając polecenia git gui. Jest to wbudowany klient gita instalowany razem z Gitem.

`git show <rev>:<file>` - wyświetlenie stanu pliku <file> z rewizji <rev>. Identyfikator rewizji może zostać podany w dowolny sposób, np. SHA-1 lub nazwa gałęzi

Tworzenie rewizji:

Aby stworzyć nową rewizję w systemie git należy podać dwie komendy:

```
git add -A
git commit -m "komunikat…"
```

Domyślnie nowo stworzone pliki nie są śledzone przez gita. Konieczne jest dodanie ich za pomocą polecenia *add*. Nowe oraz zmienione pliki nie dodane za pomocą tego polecenia nie znajdą się w rewizji utworzonej przy użyciu komendy *commit*. Parametr *-A* można zastąpić *--all*. Komenda *git add* bez parametrów nie jest prawidłowa. Sposobem na dodanie wielu (lub wszytkich) plików jest podanie nazwy katalogu, w którym się znajdują. Spowoduje to dodanie rekurencyjnie wszystkich podkatalogów oraz plików, np. *git add .* wywołane w głównym folderze doda wszystkie pliki.

Polecenie *git add* może przyjmować ścieżki ze znakami zastępczymi * (dowolny ciąg znaków) oraz ? (pojedynczy znak) np. git add *.txt, git add folder/*. Git add wywołane z parametrem *--update* lub *-u* pomija pliki, które nie są śledzone przez system Git.

Nowe pliki, które nie zostały dodane do systemu Git mają status nieśledzone (untracked). Plik dodany za pomocą polecnia *git add* ma status zaindeksowanego (staged). Informacje o śledzonych oraz nowo zaindeksowanych plikach znajdują się w pliku .git/index

Możliwe jest zastosowanie polecenia *git commit* z parametrem *-a*. Spowoduje to automatyczne zaindeksowanie zmodyfikowanych oraz usuniętych plików oraz stworzenie nowej rewizji. Polecenie to nie uwzględnia plików nieśledzonych.

`git add --patch (-p)` – zatwierdzenie pojedynczych zmian w każdym pliku. Jeśli zmiany zostały utworzone w znacznej odległości od siebie możliwe jest zatwierdzenie tylko części znich. Jest to polecenie interaktywne, które umożliwia stwierdzenie jak akcja ma zostać wykonaa na aktulanym fragmencie. Znak zapytania wyświetla pomoc.

`git rm <name>` powoduje usunięcie pliku oraz zaindeksowanie tej zmiany. Nie jest wymagane aby plik wskazany w tej komendzie istniał na dysku - można go usunąć w inny sposób a następnie zaindeksować tą komendą. Dodanie parametru *--cached* spowoduje zaindeksowanie usunięcia pliku bez faktycznego usuwania. Dodanie parametru *-f* usunie plik nawet jeśli będzie on posiadał jakieś niezatwierdzone zmiany lub będzie zaindeksowany.

Za pomocą polecenia git mv możliwa jest zmiana nazwy pliku oraz automatyczne jego zaindeksowanie. Nazwę można zmienić w dowolny inny sposób jednak wtedy będzie konieczne manualne go zaindeksowanie.

Plik można usunąć ze *staging area* za pomocą polecenia git reset HEAD <name>.

Staus plików:

`git status` - wyświetlenie aktualnego stanu folderu roboczego. Wyświetla zmodyfikowane/nowe/usunięte pliki w stosunku do najnowszej rewizji. Parametr *-s* spowduje wyświetlenie skróconej wersji. Każdy wiersz będzie rozpoczynał się od dwuliterowego znacznika informującego o stanie pliku. Interpretację znaczników przedstawiono poniżej (_ zastępuje spację):

- ?? – plik nieśledzony (untracked)
- A_ – nowy zaindeksowany plik
- _M – plik, który był aktualny ale został zmodyfikowany, niezaindeksowany
- M_ - zmodyfikowany plik, który został dodany poleceniem *git add*
- _D – plik, który był śledzony oraz został usunięty bez użycia *git rm*
- D_ - plik, który był śledzony oraz został usunięty z wykorzystaniem *git rm* lub informacja ta została zaindeksowana za pomocą polecenia *git add*
- R_ - plik, którego nazwę zmieniono przy użyciu polecenia *git mv*
- RM – jednoczesna zmiana nazwy oraz treści pliku
- AD – sekwencja komend git add oraz git rm
- MM – zaindeksowany plik, w kótrym wykonano kolejne modyfikacje

`git status -sb` – wyświetlenie nazwy gałęzi oraz informacji o śledzeniu gałęzi zdalnej nawet w skróconym formacie

Git checkout:

`git checkout [-f] <name>` - zmiana aktualnej gałęzi na tą wskazywaną przez *<name>.* Wszystkie pliki muszą być aktualne przed zmianą gałęzi. Parametr spowoduje odrzucenie wszystkich lokalnych zmian przed zmianą gałęzi.

`git checkout -b <name>` - utworzenie gałęzi *<name>* oraz natychmiastowa zmiana. Można także utworzyć w ten sposób gałąź wskazującą na dowolną rewizję.

`git checkout <branch>` - powoduje przejście do danej gałęzi. Jeśli gałąź nie istnieje lokalnie (ale istnieje w repozytorim zdalnym), to zostanie utworzona, zostaną pobrane rewizje z repozytorium zdalnego oraz folder roboczy zostanie uaktualniony do odpowiedniego stanu. Zostanie także skonfigurowanie śledzenie gałęzi zdalnej.

`git checkout -b <new_branch> <old_branch>` - utworzenie nowej gałęzi na podstawie innej oraz automatyczne przejście do *<new_branch>*

`git checkout <SHA-1>` - zmiana stanu folderu roboczego na tą z rewizji *<SHA-1>.* Repozytorium znajduje się wtedy w stanie detached head. Plik .git/HEAD zawiera skrót konkretnej rewizji a nie identyfikator gałęzi. Git branch zwraca informację *(no branch), co oznacza, że nie znajdujemy się w

żadnej gałęzi. Możliwe jest stworzenie nowej rewizji w stanie detached HEAD, ale nie będzie ona identyfikowana przez żadną z gałęzi. Aby zmienić ten stan należy utworzyć nową gałąź poprzez *git checkout -b <name>.*

`git checkout --conflict=[diff3/merge] <file>` - w przypadku rozwiązywania konfliktów łączenia rewizji umożliwia przywrócenie pliku do stanu sprzed modyfikacji (przywraca znaczniki konfliktu oraz usuwa wprowadzone zmiany). Wybranie opcji *diff3* spowoduje dodanie znaczników z wspólnego przodka oprócz tych z „naszej" oraz „ich" gałęzi.

`git checkout -m <file>` - tak jak powyżej, opcja --conflict jest ustawiana zgodnie z wartością pola *merge.conflictstyle* w konfiguracji Gita

`git checkout --[ours/theris] <file>` - podczas konfliktu operacji merge: przywrócenie zawartości pliku z gałęzi, do której zmiany są wcielane (*ours*) lub gałęzi, z której pochodzą zmiany (*theirs*). W przypadku operacji rebase *ours* oraz *theris* będą zamienione miejscami, ponieważ jednym z etapów rebase jest zmiana gałęzi na tą, która służy jako podstawa przenoszonych rewizji z innej gałęzi. Polecenie to można wykorzystać do szybkiego rozwiązywania konfliktów

Notacje ~ oraz ^:

HEAD to symboliczna rewizja wskazująca na najnowszą rewizję w aktualnej gałęzi. Identyfikator rewizji jest przechowywany w pliku .git/HEAD. Rewizja HEAD jest domyślnie używana, jeśli nie zostanie podana żadna inna. Do poprzednich rewizji można się odwołać korzystając z notacji HEAD~n, gdzie n oznacza n-tego przodka. Notację ~n można także zastosować do skrótów SHA-1 lub nazw gałęzi, np. 11e2f~1 oznacza rodzica tej rewizji. Znak tyldy można zapisać wielokrotnie, wystąpienia się sumują. Tylda zawsze odnosi się do pierwszego rodzica rewizji. N-tego rodzica (jeśli rewizja posiada kilku) można wybrać korzystając z notacji <SHA-1>^n. Jedynkę w zapisie można pominąć, np. HEAD~1 jest równoważne z HEAD~. Wywołania ~ oraz ^ można łączyć. Kolejność w tym wypadku ma znaczenie, tj. HEAD~2^1 oznacza co innego niż HEAD^1~2.

W konsoli windows może być konieczne cytowanie niektórych znaków. Służy do tego znak ^, np. *echo a^>b* wyświetli dokładnie ten ciąg. Pominięcie znaku cytowania spowoduje zapisanie w pliku b litery a. Znaki w plikach wsadowych trzeba cytować podwójnie tak więc *git show HEAD^1* należy zapisać jako *git show HEAD^^^^1*

Ignorowanie plików:

Nie zawsze istnieje potrzeba przechowywania wszystkich plików z obszaru roboczego w repozytorium. Mogą to być na przykład pliki konfiguracyjne IDE, pliki z hasłami dostępowymi i innymi poufnymi danymi. Pliki takie można zignorować na trzy sposoby:

- korzystając z pliku .gitignore – jest przechowywany w repozytorium oraz współdzielony pomiędzy użytkownikami
- korzystając z pliku .git/info/exclude – nie jest wysyłany na zdalne repozytorium. Każdy użytkownik posiada swoją lokalną wersję
- korzystając z globalnego pliku .gitignore – podobnie jak plik exclude, jednak reguły stosowane są w każdym lokalnym repozytorium (*git config --global core.excludesfile <path/to/file>*)

Składnia pliku:

- każda linia składa się ze wzorca, według którego pliki są ignorowane
- linie rozpoczynające się od # są ignorowane
- ** odpowiada dowolnej ścieżce, np. a/**/b będzie dopasowane do a/b, a/x/b, a/x/y/b
- znaki *, ? oznaczają odpowiednio wiele lub jeden znak (z pominięciem /)
- W nawiasach [] można podać zestaw znaków, które mogą zostać dopasowane, np. file-[ab].txt

Wszystkie pliki pasujące do wzorców zostaną zignorowane i nie będą wyświetlane jako nieśledzone za pomocą polecenia git status.

Znaczniki (etykiety, tag):

Znaczniki (tag) służą do oznaczania istotnych rewizji w repozytorium za pomocą np. numerów wersji. W systemie git można tworzyć dwa rodzaje znaczników: opisane (*annotated*) oraz lekkie (*lightweight*). Znaczniki zawierają takie informacje jak dane autora, datę utworzenia, komentarz oraz skrót do rewizji, na którą wskazują. Znaczniki lekkie zawierają wyłącznie skrót SHA-1 rewizji i są zapisywane jako referencje w folderze .git/refs/tags (lub .git/packed-refs). Znaczniki opisane są przechowywane w sposób analogiczny do nowych rewizji oraz posiadają swój skrót SHA-1

`git tag` - wyświetlenie wszystkich znaczników

`git tag –a <name> –m <comment>` - stworzenie nowego znacznika opisanego, np. *git tag -a v1.2.3 -m „Release ver. 1.2.3”*. Taki znacznik będzie wskazywał na ostatnią rewizję w projekcie. Przekazując jako ostatni parametr skrót SHA-1 można stworzyć znacznik wskazujący na dowolną rewizję. Nazwy znaczników muszą być unikatowe.

`git tag <name> [sha-1]` - stworzenie lekkiego znacznika. Możliwe jest podanie opcjonalnego ciągu będącego skrótem do rewizji, na którą znacznik ma wskazywać.

`git tag –d <name>` - usunięcie znacznika. Operacja nie jest automatycznie propagowana do repozytorium zdalnego. Można to wykonać jednym z poleceń: *git push <remote> :<tagname>* lub *git push --delete <remote> <tagname>*

`git log --tags --simplify-by-decoration --pretty="format:%ai %d"` - wyświetlenie wszystkich tagów wraz datami utworzenia

`git describe` – wyświetlenie odnośnika do rewizji w sposób zrozumiały dla człowieka na podstawie ostatniego znacznika. Jeśli na daną rewizję wskazuje jakiś znacznik zostanie zwrócona tylko jego nazwa. W innym przypadku zostanie dodana m.in. informacja o ilości rewizji dzielących daną rewizję od znacznika. Aby wziąć pod uwagę także lekkie znaczniki należy dodać parametr --tags

`git show <tag_name>` - wyświetlenie szczegółowych informacji o znaczniku łącznie ze zmianami w plikach. Parametr *-s* skraca zapis do podstawowych informacji jak autor, SHA-1, data. *Git show* można zastosować także do rewizji.

Nazwy znaczników można wykorzystywać wszędzie tam, gdzie stosuje się skrót SHA-1, np*. git checkout v.1.4*, *git reset --hard v.4.5.6*

Reflog:

Dziennik reflog zawiera informacje o wszystkich zmianach wykonanych w repozytorium. Git tworzy jeden dziennik główny oraz osobne dzienniki dla każdej gałęzi oraz stash. Znajdują się one w folderze .git/logs/. Główny dziennik znajduje się w pliku .git/logs/HEAD.

Identyfikatory reflog mają postać *HEAD@{n}*, gdzie n=0 oznacza najnowszą zmianę. Aby ograniczyć wpisy reflog do jednej gałęzi należy podać jej nazwę jako parametr, np. Git reflog master. W przypadku odwołań do gałęzi wpisy reflog będą miały postać *<branch name>@{n}*. Oprócz numeru pozycji w dzienniku reflog można w nawiasach klamrowych podać ciąg reprezentujący datę lub, np. *n.hour.ago*, co daje wpis sprzed *n* godzin.

`git reflog expire --all --expire=now` – wyczyszczenie dziennika reflog

`git reset --hard HEAD@{n}` – Przywrócenie stanu repozytorium do tego wskazywanego przez wpis w dzienniku. Umożliwia to np. wycofanie zmian spowodowanych poleceniem *git reset --hard*. Git po pewnym czasie trwale usunie rewizje usunięte przez polecenie *git reset --hard* i nie będzie możliwe ich przywrócenie.

`git reflog show <branch>` - wydrukowanie dziennika reflog dla gałęzi *<branch>* (*git reflog = git reflog show HEAD*)

Aliasy:

Możliwe jest zdefiniowanie skrótów popularnych komend w pliku .gitconfig w folderze domowym użytkownika. Należy to zrobić na końcu pliku w sekcji [alias], np.

```
[alias]
      l = log --pretty=oneline --abbrev-commit --abbrev=4 -25
```

Wywołanie będzie miało postać *git l*. Możliwe jest przekazanie dodatkowych parametrów do aliasu lub nadpisanie istniejących, np. *git l -10*.

Możliwe jest łączenie kilku komend w jednym aliasie, np.

```
save = !git add -A && git commit -m „\"$@\"" && shift 1 && echo Saved!
```

Wykrzyknik oznacza, że polecenie jest pełnoprawnym poleceniem konsoli i nie ma potrzeby wywoływania go w gicie. Zapisane polecenie ma składnię skryptu bash, np. && oznacza połączenie poleceń a $@ oznacza wszystkie argumenty przekazane do polecenia. Wszystkie komendy powinny zostać zapisane w jednej linii.

Wycofywanie zmian:

Nie należy i nie powinno być możliwe modyfikowanie publicznej historii repozytorium. Jest to wersja repozytorium dostępna wszystkim użytkownikom. Jedyną operacją dozwoloną we współdzielonej części jest dodawanie nowych rewizji. Wspominiane poniżej operacje należy wykonywać jedynie w prywatnej części repozytorium, chyba że napisano inaczej.

Polecenie *git reset* może przyjmować jeden z trzech parametrów określających zakres działania komendy:

- `--soft` – zmienia aktualne położenie wskaźnika HEAD na rewizję przekazaną jako parametr. Wszystkie nowsze rewizje zostają usunięte, nie dochodzi zaś do zmiany indeksu lub obszaru

roboczego. Może zostać zastosowany, np. w celu spłaszczenia kilku prywatnych rewizji w jedną przed wysłaniem zmian do zdalnego repozytorium

- `--mixed` – tak jak powyżej, a dodatkowo indeks zostaje zresetowany do postaci odpowiadającej rewizji przekazanej jako parametr. Wszystkie pliki zmodyfikowne w późniejszym okresie są widoczne jako zmodyfikowane lub nieśledzone. Stan obszaru roboczego nie zostaje zmieniony. Jest to tryb domyślny
- `--hard` – tak jak powyżej, a dodatkowo przywraca stan obszaru roboczego do stanu z rewizji przekazanej jako parametr

`git reset --hard HEAD` - przywrócenie folderu roboczego do stanu z najnowszej rewizji. Zamiast parametru HEAD można podać dowolny identyfikator rewizji. W przypadku podania innego identyfikatora rewizji niż HEAD zostaną także usunięte wszystkie rewizje stworzone w późniejszym czasie.

Polecenie *git reset [mode] SHA-1* modyfikuje tylko stan obecnej gałęzi. Jeśli rewizje są wykorzystywane przez więcej niż jedną gałąź, to nie zostaną one utracone i będzie można się do nich dostać przy wykorzystaniu innej gałęzi.

`git reset --hard -p` – wycofanie zmian we fragmentach plików. Zastosowanie analogiczne do *git add -p*

`git commit --ammend` - modyfikacja ostatniej rewizji. Powoduje to wygenerowanie nowego skrótu SHA-1, co należy interpretować jako usunięcie ostatniej rewizji i dodanie nowej zawierającej aktulanie zaindeksowane pliki.

`git rebase -i <sha-1>` - edycja wszystkich rewizji występujących po rewizji *<sha-1>*, np. połączenie ich w jedną lub rozdzielenie rewizji. Polecenie to jest wykonywane w trybie interaktywnym.

`git revert --no-edit <sha-1>` - wycofuje zmiany wprowadzone przez rewizję *<sha-1>* tworząc nową rewizję. Parametr *--no-edit* zatwierdza cofnięte zmiany i generuje automatyczną wiadomość. Komendę tą można zastosować zarówno w części prywatnej jak i publicznej, ponieważ nie modyfikuje ona istniejących rewizji. Nie jest możliwe scalenie rewizji oraz drugiej, która wycofała zmiany pierwszej. Połączenie tych dwóch rewizji nie wprowadza żadnych zmian, a nie jest możliwe stworzenie rewizji nie wprowadzających zmian w repozytorium.

`git checkout -f <sha-1> [file1 file2]` - przywrócenie plików do stanu z rewizji *<sha-1>*.

Usunięcie wybranego pliku z historii repozytorium:

```
git filter-branch --force --index-filter 'git rm --cached --ignore-unmatch
PATH-TO-FILE' --prune-empty --tag-name-filter cat -- --all
git push origin --force --all
git push origin --force --tags
```

Gałęzie:

Gałęzie w Gicie są wskaźnikami rewizji. Nazwa gałęzi wskazuje na najnowszą rewizję w danej gałęzi. Wskaźniki rewizji znajdują się w folderze *.git/refs/heads/* (w ramach optymalizacji Git może umieścić referencje w pliku *.git/packed-refs*). Każda gałąź posiada swój plik o nazwie odpowiadającej nazwie gałęzi. Gałęzie zdalne znajdują się w folderze *.git/refs/remotes/<remote>/.*

Informacja o aktualnej gałęzi znajduje się w pliku *.git/HEAD*. Plik *.git/FETCH_HEAD* zawiera informacje o najnowszych rewizjach dostępnych na zdalnym repozytorium po wykonaniu polecenia *git fetch*. Plik *.git/ORIG_HEAD* zawira informacje o położeniu wskaźnika HEAD przed wykonaniem niebezpiecznej operacji i może służyć do przywrócenia stanu obszaru roboczego w razie niepowodzenia.

Nazwy gałęzi można wykorzystać jako identyfikatory rewizji w taki sam sposób jak HEAD.

`git branch` - wyświetlenie lokalnych gałęzi (gwiazdka przy nazwie gałęzi zawsze wskazuje na aktualną)

`git branch -r` - wyświetlenie gałęzi zdalnych

`git branch -a` - wyświetlenie wszystkich gałęzi

`git branch -v (-vv / -verbose)` - wyświetlenie rozszerzonej informacji o gałęziach, m.in. gałęzie śledzone

`git branch <name>` - utworzenie gałęzi o nazwie *<name>*. Nowa gałąź wskazuje na aktualną rewizję. Utworzenie nowej gałęzi nie powoduje zmiany gałęzi, w której się aktualnie znajdujemy

`git branch <name> <sha-1>` - utworzenie nowej gałęzi wskazującej na rewizję identyfikowaną skrótem SHA-1

Gałęzie mogą się w sobie zawierać wtedy, gdy wszystkie rewizje jednej gałęzi występują także w drugiej, np. G1 jest zawarta w G2 jeśli wszystkie rewizje należące do G1 przynależą także do rewizji G2. W innym przypadku gałęzie są rozłączne

`git branch --merged <SHA-1>` - wyświetla wszystkie gałęzie, których wierzchołki są osiągalne z danej rewizji przemieszczając się wstecz, *--no-merge* ma odwrotny skutek. Jeśli nie podano SHA-1 używana jest bieżąca gałąź (HEAD).

`git branch -d <name>` - usunięcie gałęzi o nazwie *<name>.* Zadziała jedynie, gdy gałąź jest zawarta w bieżącej gałęzi, tj. nie spowoduje to utracenia dostępu do rewizji. Aby usunąć gałąź, która nie jest zawarta w innej gałęzi należy wykonać polecenie *git branch -D <name>*. W takim przypadku możliwe jest odwołanie się do utraconych gałęzi z wykorzystaniem dziennika reflog.

`git branch -m <old-name> <new-name>` - zmiana nazwy gałęzi. *<new-name>* nie może istnieć w repozytorium. Zmiana paremetru na *-M* spowoduje wymuszenie zmiany nazwy oraz ustracenie dostępu do rewizji wskazywanych przez wcześniej istniejącą gałąź *<new-name>.*

`git branch --track (-t) <new_branch> <remote-repo>/<remote-branch>` - utworzenie nowej gałęzi <new_branch> oraz skonfigurowanie jej gałęzi śledzonej na istniejącą gałąź

`git branch --set-upstream-to=<remote>/<branch> <local_branch>` - ustawienie gałęzi śledzonej dla gałęzi *<local_branch>.* Jeśli gałąź lokalna zostanie pominięta, to zostanie wykorzystana aktualna. Takie samo zachowanie zapewnia parametr *-u*. Należy wtedy zastąpić znak równości spacją.

Diff:

`git diff` - sprawdznie różnic pomiędzy obszarem roboczym a repozytorium. Wyświetla jedynie niezaindeksowane pliki

`git diff <a> <b>` - sprawdzenie różnic pomiędzy dwiema rewizjami

`git diff --staged` - różnice pomiędzy plikami zatwierdzonymi (*git add*) a plikami z ostatniej rewizji

`git diff --name-only` - wyświetlenie nazw zmienionych plików

`git diff --name-only <sha-1> <sha-2> -- <location>` - wyświetlenie zmodyfikowanych plików z folderu <location> na podstawie różnic pomiędzy rewizjami (mogą to być nazwy gałęzi)

`git diff --name-status <branch_1>..<branch_2>` - wyświetlenie plików, które różnią się pomiędzy gałęziami/rewizjami. Dodatkowo wyświetlany jest znacznik informujący o rodzaju zmian (modyfikacja, usunięcie, dodanie)

`git diff --word-diff` - wyszukiwanie zmienionych wyrazów w pliku, zamiast całych wierszy

`git diff --<ours/theirs/base>` - w przypadku rozwiązywania konfliktów umożliwia wyświetlenie zmian wprowadzonych w pliku w stosunku do wersji z bieżącej gałęzi/ z dołączanej gałęzi/ ze wspólnego przodka, np. *git diff --ours*

W przypadku każdego pliku wyświetlane są oznaczania --- (plik początkowy) oraz +++ (plik końcowy). Kolejne linie wyjaśniają, jakie zmiany należy wykonać aby otrzymać z pliku początkowego plik końcowy. Zapis pomiędzy znakami @@ zawiera informację o tym, który numer wiersza jest wyświetlany jako pierwszy oraz liczbę wyświetlonych wierszy, np. 1,3 oznacza pierwszy wiersz, oraz że są wyświetlane 3 wiersze. Znaki - oraz + poprzedzające informację o wyświetlanych wierszach wskazują odpowiednio na plik początkowy oraz końcowy. Liczbę wyświetlanych wierszy można zmodyfikować dodając parametr *--unified=n* (zostanie wyświetlone *n* wierszy przed i po zmienionej linii). Dla jednego pliku może zostać wyświetlone wiele bloków ze znacznikami @@, jeśli zmiany znajdują się w znacznej odległości od siebie (zakresy wyświetlanych wierszy wraz z kontekstem nie nachodzą na siebie).

Łączenie gałęzi:

Łączenie gałęzi *fast-forward* (przewijanie do przodu) - jest to łączenie gałęzi, które są połączone relacją zawierania. W takim wypadku nie występują konflikty a cała operacja jest automatyczna.

`git merge <branch>` - dołączenie zmian z gałęzi <*branch*> do bieżącej gałęzi. Aby połączyć wiele gałęzi należy podać ich listę po komendzie merge. Gałęzie należy oddzielić spacjami.

`git rebase <branch>` - przeniesienie zmian wprowadzonych w obecnej gałęzi na wierzchołek gałęzi <*branch*>. Podczas operacji odnajdywany jest wspólny przodek obu gałęzi a następnie zmiany z poszczególnych rewizji z obecnej gałęzi są przenoszone na gałąź <*branch*>. W ten sposób tworzona jest liniowa historia projektu a gałęzie połąoczne są ze sobą relacją zawierania i możliwe jest wykonanie operacji *merge* w trybie *fast-forward*. Operację należy wykonywać jedynie na prywatnych gałęziach, ponieważ przeniesione rewizje otrzymują nową sume SHA-1. *Rebase* może zostać wykorzystane do zaktualizoawnia gałęzi lokalnej na podstawie gałęzi śledzonej pobranej za pomocą *git fetch*, np. *git rebase origin/master.*

Aby wycofać operację łącznia gałęzi należy posłużyć się poleceniem *git reset --hard* HEAD~ jeśli ostatnią rewizją jest operacja łączenia. Zamiast odniesienia do HEAD można wykorzystać skrót SHA-1 rewizji sprzed operacji łączenia lub odniesieniem do reflog. Dodanie parametru *--merge* spowoduje wycofanie jedynie zmian wprowadzonych przez operację merge. Jeśli w obszarze roboczym istaniły

inne zmiany, które nie zostały dodane, to zostaną one zachowane. Wycofanie operacji *rebase* należy wykonać z wykorzystaniem dziennika reflog.

Konflikty:

Jeśli dokonamy modyfikacji jednego pliku w dwóch gałęziach może dojść do sytuacji konfliktu podczas operacji *merge* lub *rebase*. W takim wypadku polecenie *git status -s* zwróci oznacznie UU przy nazwie pliku. W pliku zostaną wstawione odpowiednie znaczniki wskazujące na miejsce wystąpienia konfliktu.

Rozwiązywanie konfliktów w plikach tekstowych:

- *merge* - po naprawieniu konfliktu plik należy zaindeksować (*git add*) oraz zatwierdzić (*git commit*). Brak parametrów przy poleceniu *git commit* spowoduje otworzenie edytora z domyślnym komentarzem.
- *rebase* - po dokonaniu napraw należy wykonać polecenia *git add .* oraz *git rebase --continue*. Podczas operacji rebase (i naprawiania ewentualnych konfliktów) repozytorium znajduje się w stanie detached head.

W przypadku plików binarnych możliwe jest jedynie wybranie wersji pliku, która ma zostać zachowana za pomocą polecenia *git checkout --[ours/theirs]*. Domyślnie wybierany jest plik z gałęzi, w której repozytorium znajduje się podczas wystąpienia konfliktu.

W przypadku wystąpienia konfliktu możliwe jest wyświetlenie 3 wersji pliku znajdujących się w indeksie:

- *git show :1:<filename>* - stan pliku we wspólnym przodku (rewizja, na podstawie której stworzono gałąź, *common*)
- *git show :2:<filename>* - wersja pliku z gałęzi, w której wykonujemy operację merge (*ours*)
- *git show :3:<filename>* - wersja pliku z gałęzi, z której pochodzą łączone zmiany (*theirs*)

`git merge-file -p <ours> <common> <theirs>` - wykonanie operacji merge na wybranym pliku. Należy przekazać plik w 3 wersjach. Parametr *-p* powoduje wypisane zmian na standardowe wyjście (konsolę) a nie do pliku *<ours>*

Repozytoria zdalne:

origin - domyślna nazwa powiązana z adresem repozytorium zdalnego, które klonujemy.

Informacje na temat repozytoriów zdalnych oraz gałęzi śledzonych znajdują się w pliku *.git/config*.

`git remote [-v]` - wyświetlenie zdalnych repozytoriów. Opcja *-v* wyświetla dodatkowe informacje.

`git remote add <name> <address>` - zdefiniowanie repozytorium zdalnego o nazwie *<name>* oraz adresie *<address>*

`git remote rm <name>` - usunięcie konfiguracji repozytorium zdalnego

`git fetch <remote>` - pobranie najnowszej wersji obiektów i referencji z repozytorium zdalnego. Pobrane referencje są zapisywane w pliku *.git/FETCH_HEAD*. Pobrane informacje nie są

automatycznie scalane z gałęziami lokalnymi. Aby to zrobić należy wykonać polecenia *merge* lub *rebase*, np. *git merge origin/master*.

`git fetch origin <src>:<dst>` - pobranie z repozytorium zdalnego jedynie referencji pasujących do wzorca *<src>* oraz zapisanie ich pod refeferencjami zgodnie z formatem *<dst>* (*refspec*). Przykładowo *git fetch refs/heads/master:refs/heads/remotes/origin/master* pobierze z repozytorium origin gałąź master i zapisze jej referencję w pliku *.git/heads/remotes/origin/master*. Zapis można skrócić do *git fetch origin master:remotes/master*.

`git fetch <remote> --tags` - pobranie dodatkowo znaczników z repozytorium zdalnego. Domyślnie wszystkie znaczniki, które wskazują na rewizje pobierane za pomocą fetch są także pobierane. Można to zmienić stosują opcję *--no-tags*.

`git pull <remote>` – jednoczesne pobranie gałęzi zdalnej, wykonanie operacji merge (lub rebase) oraz aktualizacja folderu roboczego. Można stosować takie same parametry jak w przypadku *fetch*. Domyślną operacją łączenia zmian z repozytorium zdalnego jest *merge*. Można to zmienić stosując paramter *--rebase* lub za pomocą konfiguracji ustawiając *pull.rebase* na *true*.

`git push <remote repo> <branch>` - przesłanie rewizji do repozytorium zdalnego. Domyślnie ominięcie parametrów powoduje wysłanie gałęzi do gałęzi śledzonej - zachowanie to można zmienić za pomocą konfiguracji. Dodatkowy parametr *-u* spowoduje zapamiętanie gałęzi zdalnej jako śledzonej. Przesłanie gałęzi jest możliwe tylko w formie przewijania do przodu (*fast-forward*), czyli gałąź zdalna musi być całkowicie zawarta w gałęzi lokalnej. Możliwe jest wymuszenie operacji push stosując parametr *-f*. Nie jest to bezpieczna operacja. Parametr *--all* spowoduje przesłanei wszystkich gałęzi lokalnych do ich gałęzi śledzonych.

`git push origin HEAD` – wysłanie obecnej gałęzi do repozytorium zdalnego

`git push origin :` - wysłanie wszystkich gałęzi posiadających gałęzie zdalne o takiej samej nazwie w repozytorium identyfikowanym jako *origin*.

`git push origin <local branch>:<remote branch>` - przesłanie gałęzi lokalnej ze zmianą nazwy z wykorzystaniem notacji refspec

`git push :<remote branch>` - usunięcie gałęzi zdalnej. Analogiczny skutek ma komenda *git push <remote> --delete <branch_name>*

`git push <remote> --tags /git push <remote> <tagname>` - wysłanie tagów do repozytorium zdalnego. Zalecane jest wysyłanie tagów pojedynczo, tak aby uniknąć sytuacji, gdy wysyłamy prywatne znaczniki na serwer. Domyślnie *git push* nie przesyła znaczników.

Dodatkowe komendy i przydatne informacje:

`git archive --format=zip --output=<filename> <SHA-1>` - utworzenie skompresowanego archiwum zwierającego daną rewizję. Możliwe jest podanie skrótu SHA-1 lub dowolnej rewizji

`git cherry-pick <sha-1> <sha-1 2 etc>` - wprowadź zmiany z danej rewizji w obecnej gałęzi. W razie konfliktów należy kontynuować *git cherry-pick --continue* lub przerwać *git cherry-pick --abort*. *-x* - dodanie do wiadomości standardowej informacji o cherry-picku

Rewizje, do których nie da się dotrzeć korzystając z gałęzi nazywane są zgubionymi (*dangling*). Można do nich dotrzeć przez pewien okres czasu, zanim zostaną automatycznie usunięte. Zgubione rewizje są usuwane przez garbage collecotr zgodnie z ustawieniami:

- gc.pruneexpire (domyślnie 2 tyg.)
- gc.reflogexpireunreachable (domyślnie 30 dni)

`git prune` - uruchumienie oczyszczania repozytorium, m.in. usuwania zgubionych rewizji.

`git rev-list` - wyświetla listę rewizji, do których możemy dotrzeć rozpoczynając od danej rewizji.

`git rev-parse` - konwersja wszystkich typów identyfikatorów do postaci skrótu SHA-1, np. *git rev-parse HEAD* zwróci skrót najnowszej rewizji. Podając identyfikator znacznika zostanie zwrócony skrót tego znacznika, a nie rewizji, na którą wskazuje. Aby poznać skrót rewizji należy wykonać polecenie *git rev-list v0.1.2.3 -1*, co oznacza wyświetlenie pierwszej rewizji osiągalnej za pomocą wskazanego znacznika. Innym sposobem jest wykorzystanie polecenia *git cat-file -p <SHA-1>*

Możliwe jest połączenie dwóch niezależnych repozytoriów. Należy dodać jedno z nich jako remote i wykonać operację pull origin/master. Aby wyeliminować niespójność (dwie pierwsze rewizje) należy utworzyć tymczasową gałąź korzystając z jednej z rodzielnych gałęzi (HEAD~) a następnie wykonać operację rebase.

Repozytorium pozbawione obszaru roboczego oraz indeksu nazywane jest repozytorium surowym. Można je stworzyć korzystając z komendy *git clone --bare <address>* lub *git init --bare*. W głównym folderze znajdą się pliki normalnie zapisane w folderze .git.

Aby uniemożliwić utratę rewizji w repozytorium, które służy jako repozytorium zdalne (np. *push -f*) należy skonfigurować następujące opecje:

- receive.denyDeletes true
- receive.denyNonFastForwards true

Domyślnie niemożliwe jest wysyłanie zmian do zdalnego repozytorium standardowego. Aby to umożliwić należy zmienić opcję *receive.denyCurrentBranch* na *ignore*. Zmiany wywołane poleceniem *git push* nie będą automatycznie widoczne w folderze roboczym repozytorium zdalnego. Należy wykonać operacje *git reset --hard* lub oprogramować zdarzenie *post-update* tworząc odpowiedni skrypt w pliku o nazwie *.git/hooks/post-update* oraz dodać konfigurację folderu roboczego: *git config core.worktree ../* lub bezpośrednio w pliku *.git/config*.

Rozróżnianie pomiędzy plikami binarnymi a tekstowymi jest zaimplementowane na podstawie atrybutów zapisanych w pliku *.gitattributes* lub *.git/info/attributes*. Format to *<filename> <attr1> <attr2>*. *-<attr>* wyłącza atrybut a *<attr>=<val>* ustawia wartość atrybutu. Atrybut *diff* ustala, czy dany plik ma być porównywany jako tekstowy czy jako binarny. Możliwe jest stosowanie znaków specjalnych w nazwach plików, np. *.txt.

Generowanie klucza ssh:

Komenda: `ssh-keygen -t rsa -C "email@example.com"`. W kolejnych krokach należy podać lokalizację pliku z kluczem oraz passphrase. Tworzone są dwa pliki. Plik z rozszerzeniem .pub zawiera klucz publiczny, który należy zamieścić na serwerze udostępniającym repozytorium.

Pull request:

Pull request - technika pracy wykorzystująca żądanie aktualizacji (ang. *pull request*). Każdy uczestnik projektu tworzy dwie kopie repozytorium głównego. Najpierw w interfejsie WWW wykonujemy operację rozgałęziania (ang. *fork*). W wyniku tej operacji otrzymujemy na serwerze własne repozytorium, które jest klonem repozytorium głównego. Repozytorium to klonujemy na dysk lokalny. Zmiany w repozytorium głównym wykonujemy dwuetapowo. Najpierw modyfikujemy repozytorium na dysku lokalnym. Wykonane zmiany przesyłamy na własne repozytorium odgałęzione. Następnie w repozytorium odgałęzionym wykonujemy operację żądania aktualizacji, która polega na zgłoszeniu w repozytorium głównym chęci scalenia naszych zmian. Administrator repozytorium głównego analizuje żądanie aktualizacji i jeśli stwierdza, że jest ono odpowiednie, akceptuje je.

Pracując w tym trybie należy skonfigurować główne zdalne repozytorium do synchronizacji - własne zdalne repozytorium służy jedynie do wprowadzania własnych zmian oraz wykonywanie żądań aktualizacji. Zmiany wprowadzone przez inne osoby są widoczne jedynie w repozytorium głównym. Możliwe jest także sklonowanie lokalnie repozytorium głównego jako origin i dodanie własnego repozytorium zdalnego poprzez *git remote add*. Sposób działania jest taki sam jak w pierwszym scenariuszu (*pull request* z własnego zdalnego repozytorium)

Zalecany format wiadomości podczas zatwierdzania rewizji:

```
Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary.  Wrap it to
about 72 characters or so.  In some contexts, the first
line is treated as the subject of an email and the rest of
the text as the body.  The blank line separating the
summary from the body is critical (unless you omit the body
entirely); tools like rebase can get confused if you run
the two together.

Further paragraphs come after blank lines.

  - Bullet points are okay, too

  - Typically a hyphen or asterisk is used for the bullet,
    preceded by a single space, with blank lines in
    between, but conventions vary here
```

Vi:

Dwa tryby działania

- tryb komend (esc x2 aby się upewnić, że jest się w tym trybie)
- tryb wstawiania

Vi zawsze rozpoczyna pracę w trybie komend. Aby wejść do trybu wstawiania należy wcisnąć i (insert). Klawisz esc wychodzi z trybu wstawiania do trybu komend

:q - wyjście z vi jeśli nie ma niezapisanych zmian

:q! - wyjście z pliku i porzucenie zmian

:w - zapisanie zmian w pliku

:wq - zapisz i wyjdź (ZZ też działa)

:w filename - save as

k - cursor up

j - cursor down, 2j - two lines down etc

h - cursor left

l - cursor right

i - insert mode

I - insert mode at the begining of the line

a - dodaj tekst za obecną pozycją kursora

A - dodaj tekst na końcu linii

o - tworzy nową linię poniżej kursora

O - tworzy nową linię nad kursorem

x - usuń znak na pozycji kursora

X - usuń znak przed kursorem

dw - usuń od obecnej pozycji do następnego słowa

D - od pozycji kursora do końca linii

dd - usunięcie całęj linii

yy - copy current line

p - paste after the cursor

P - paste before the cursor

u - undo

U - confij wszystkie zmiany w bieżącej linii

Yw - wytnij słowo