Co to jest GIT?

Jest to system kontroli wersji (VCS – Version Control System) służący do sledzenia zmian plików.

Założony w 2005 przez twórcę Linuxa. Nie jest wymagana znajomość zadnego języka programowania. Pozwala na przechowywanie plików z dowolnych technologii – Java, Node.JS, Python, C#, etc., etc.

No ok, fajnie, ale co to jest VCS?

VCS, tracks the history of changes as people and teams collaborate on projects together. As the project evolves, teams can run tests, fix bugs, and contribute new code with the confidence that any version can be recovered at any time. Developers can review project history to find out:

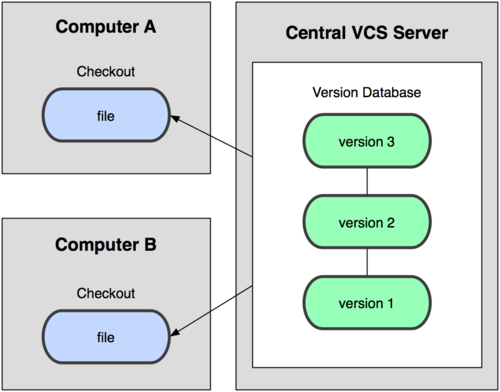
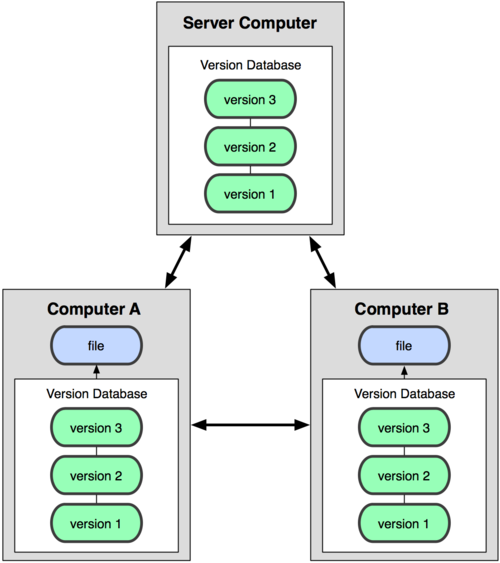
* Which changes were made?
* Who made the changes?
* When were the changes made?
* Why were changes needed?

**What’s a distributed version control system?**

* Git is an example of a distributed version control system (DVCS) commonly used for open source and commercial software development. DVCSs allow full access to every file, branch, and iteration of a project, and allows every user access to a full and self-contained history of all changes. Unlike once popular centralized version control systems, DVCSs like Git don’t need a constant connection to a central repository. Developers can work anywhere and collaborate asynchronously from any time zone.
* Without version control, team members are subject to redundant tasks, slower timelines, and multiple copies of a single project. To eliminate unnecessary work, Git and other VCSs give each contributor a unified and consistent view of a project, surfacing work that’s already in progress. Seeing a transparent history of changes, who made them, and how they contribute to the development of a project helps team members stay aligned while working independently.

Concepts of GIT

* Keeps track of code history
* Takes snapshots of your files
* You decide when to take a snapshot by making a commit
* You can visit any snapchot at any time – visit and revert if needed so it is safe
* You can stage files before commiting – DOPISAC CO TO JEST TEN STAGING



WHY GIT?

### Why Git?

According to the latest [Stack Overflow developer survey](https://insights.stackoverflow.com/survey/2017#technology), more than 70 percent of developers use Git, making it the most-used VCS in the world. Git is commonly used for both open source and commercial software development, **with significant benefits** for individuals, teams and businesses.

* Git lets developers see the entire timeline of their changes, decisions, and progression of any project in one place. From the moment they access the history of a project, the developer has all the context they need to understand it and start contributing.
* Developers work in every time zone. With a DVCS like Git, collaboration can happen any time while maintaining source code integrity. Using branches, developers can safely propose changes to production code.
* Businesses using Git can break down communication barriers between teams and keep them focused on doing their best work. Plus, Git makes it possible to align experts across a business to collaborate on major projects.

### What’s a repository?

A repository, or [Git project](https://git-scm.com/), encompasses the entire collection of files and folders associated with a project, along with each file’s revision history. The file history appears as snapshots in time called commits, and the commits exist as a linked-list relationship, and can be organized into multiple lines of development called branches. Because Git is a DVCS, repositories are self-contained units and anyone who owns a copy of the repository can access the entire codebase and its history. Using the command line or other ease-of-use interfaces, a git repository also allows for: interaction with the history, cloning, creating branches, committing, merging, comparing changes across versions of code, and more.

Working in repositories keeps development projects organized and protected. Developers are encouraged to fix bugs, or create fresh features, without fear of derailing mainline development efforts. Git facilitates this through the use of topic branches: lightweight pointers to commits in history that can be easily created and deprecated when no longer needed.

Through platforms like GitHub, Git also provides more opportunities for project transparency and collaboration. Public repositories help teams work together to build the best possible final product.

**The GitHub flow**

The GitHub flow is a lightweight, branch-based workflow built around core Git commands used by teams around the globe—including ours.

The GitHub flow has six steps, each with distinct benefits when implemented:

1. **Create a branch:** Topic branches created from the canonical deployment branch (usually master) allow teams to contribute to many parallel efforts. Short-lived topic branches, in particular, keep teams focused and results in quick ships.
2. **Add commits:** Snapshots of development efforts within a branch create safe, revertible points in the project’s history.
3. **Open a pull request:** Pull requests publicize a project’s ongoing efforts and set the tone for a transparent development process.
4. **Discuss and review code:** Teams participate in code reviews by commenting, testing, and reviewing open pull requests. Code review is at the core of an open and participatory culture.
5. **Merge:** Upon clicking merge, GitHub automatically performs the equivalent of a local ‘git merge’ operation. GitHub also keeps the entire branch development history on the merged pull request.
6. **Deploy:** Teams can choose the best release cycles or incorporate continuous integration tools and operate with the assurance that code on the deployment branch has gone through a robust workflow.

### Basic Git commands

To use Git, developers use specific commands to copy, create, change, and combine code. These commands can be executed directly from the command line or by using an application like [GitHub Desktop](https://desktop.github.com/) or Git Kraken. Here are some common commands for using Git:

* git init initializes a brand new Git repository and begins tracking an existing directory. It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.
* git clone creates a local copy of a project that already exists remotely. The clone includes all the project’s files, history, and branches.
* git add stages a change. Git tracks changes to a developer’s codebase, but it’s necessary to stage and take a snapshot of the changes to include them in the project’s history. This command performs staging, the first part of that two-step process. Any changes that are staged will become a part of the next snapshot and a part of the project’s history. Staging and committing separately gives developers complete control over the history of their project without changing how they code and work.
* git commit saves the snapshot to the project history and completes the change-tracking process. In short, a commit functions like taking a photo. Anything that’s been staged with git add will become a part of the snapshot with git commit.
* git status shows the status of changes as untracked, modified, or staged.
* git branch shows the branches being worked on locally.
* git merge merges lines of development together. This command is typically used to combine changes made on two distinct branches. For example, a developer would merge when they want to combine changes from a feature branch into the master branch for deployment.
* git pull updates the local line of development with updates from its remote counterpart. Developers use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.
* git push updates the remote repository with any commits made locally to a branch.

How work with GIT looks like

Każdy z deweloperów posiada własne repozytorium na dysku. Na nim wprowadza zmiany, na nim pracuje. Gdy uzna, że zmiany są gotowe do publikacji, wrzuca je na repozytorium zdalne (komenda push)– w chmurę np. na github.com / bitbucket.com – są to witryny posiadające swoje chmury, w których przechowywane są pliki.

Zatem niepotrzebny jest dostęp do internetu by pracować na własnym lokalnym repozytorium, jeżeli chcemy dodać zmianę którą mogą zobaczyć wszyscy – musimy go mieć.

Instalacja GIT:

* <http://git-scm.com/download/win> - automatycznie zaczyna go pobierać

Praca z GIT:

Po instalacji tworzymy folder i w nim tworzymy plik normalnie lub za pomocą komendy :

<<touch nazwa-pliku.typ-pliku>>

Następnie tworzymy w tym pliku repozytorium lokalne (każdy folder może być takim repozytorium):

<<git init>>

Tworzy on .git folder wewnątrz naszego folderu – to jest to nasze repozytorium. Na początku go nie widzimy – by to zrobić, musimy włączyć pokazywanie ukrytych plików/folderów. By to zrobić, idziemy w folderze do View -> Options -> View -> zaznaczyć „Show hidden files, folders and drives”.

Generalnie nie interesuje nas co tam się znajduje, to są pliki konfiguracyjne Gita. NIE RUSZAĆ!

Proponuję stworzyć plik .html z następującą treścią:

<html>

<head>

<title>My App</title>  
</head>

<body>

This is my app

</body>

</html>

Dopiero w momencie w którym utworzymy sobie repozytorium będziemy mogli w nim wykorzystywać komendy gitowe typu push pull etc.

Zanim zaczniemy cokolwiek robić, należy dodać do Git nasz adres email i imię zatem wpisujemy komendę:

<<git config –global user.name ‘Przemek Chromicz’>>

<<git config –global user.email ‘gitisfun@wp.pl’>>

Możemy nie wpisywać w ogóle tych komend – w takim wypadku zostaną wzięte nasze imię i nazwisko z ustawień lokalnych windows np. Przemysław Chromicz i a99vfzz@mmm.com

Teraz mamy pliki w folderze. Mamy tam zainicjalizowane repozytorium. ALE. Pliki w folderze NIE SĄ w repozytorium – Git nie śledzi(ang track) zmian. Musimy je dodać. Komenda:

<<git add nazwa-pliku.rozszerzenie-pliku>>

Jeśli chcemy zobaczyć co się znajduje w naszym repo, używamy:

<<git status>> - w tym momencie widzimy co jest w repo, a co nie.

Jeśli chcemy usunąć z repozytorium jakiś plik, robimy to za pomocą

<<git rm –cached index.html>>

Aby dodać wiele plików, robimy to zapomocą:

<<git add \*>> - doda wszystkie pliki

Są wariacje np. jeśli chcemy dodać tylko pliki .html:

<<git add \*.html>>f

To samo jak chcemy usunąc wszystkie html:

<<git rm –cached \*.html>>

I teraz dodajemy je spowrotem za pomocą:

<<git add \*>>

Modyfikujemy plik html – np. dodajmy wykrzyknij w treści <body>

I wpiszmy:

<<git status>> - widzimy, że pliki są już w repo w staging area, ale nie została dodana najnowsza modyfikacja pliku html – w repo widnieje starsza wersja niż w folderze. By nasze repozytorium lokalne było na bieżąco – należy znów wklepać:

<<git add plik.html>> - by się upewnić, że wszystko gra, można dać też

<<git status>>

Fajnie, mamy ogarnięte lokalne repo – czy na pewno? No nie, teraz musimy użyć :

<<git commit>>

Aby te zmiany faktycznie zostały zapisane w lokalnym repo. Git add tylko dodaje je.

No fajnie, wiemy jak dodawać pliki do repo. A co z ignorowaniem? Jest to możliwe. W tym wypadku nawet <<git add \*>> nie pozwoli na dodanie tychże do repo. Ano można zrobić to następująco, za pomocą:

<<touch .gitignore>> – nie pytajcie czemu. Otwieramy go w dowolnym edytorze textu – zalecany jest notepad++, ale zwykły notatnik też da radę.

Stwórzmy też plik log.txt który będziemy ingorować :

<<touch log.txt>>

Teraz w .gitignore wpiszmy :

Log.txt

Teraz przy git add i git status jedyne co zobaczymy, to ze pojawil się nowy .gitignore – ten log.txt nie pojawi się

To samo można zrobić z folderami: stwórzmy np. folder „test” a w nim „test.txt”

W .gitignore dodajemy /test i myk, ignorujemy co tam jest

Materiały

<https://guides.github.com/introduction/git-handbook/>

<https://guides.github.com/>

<https://www.youtube.com/githubguides>

<https://www.youtube.com/watch?v=SWYqp7iY_Tc>

<https://www.youtube.com/results?search_query=github+tutorial%5C>