Git & Version Control System

DigitalSkola
Uncover The World Of Digital Skills With Us

Table of Content

What will We Learn Today?

1. Version Control System
2. Git
3. Github

Apa itu Git

DigitalSkola
Uncover The World Of Digital Skills With Us

You wanted to cook a new dish

After a lot of trial and error, you made your first version of dish

@codechips Cody popupdev04@gmail.com

You want to improvise your dish

So you add few other ingredients and make a 2nd version of your dish

@codechips Cody popupdev04@gmail.com

Apa itu Git

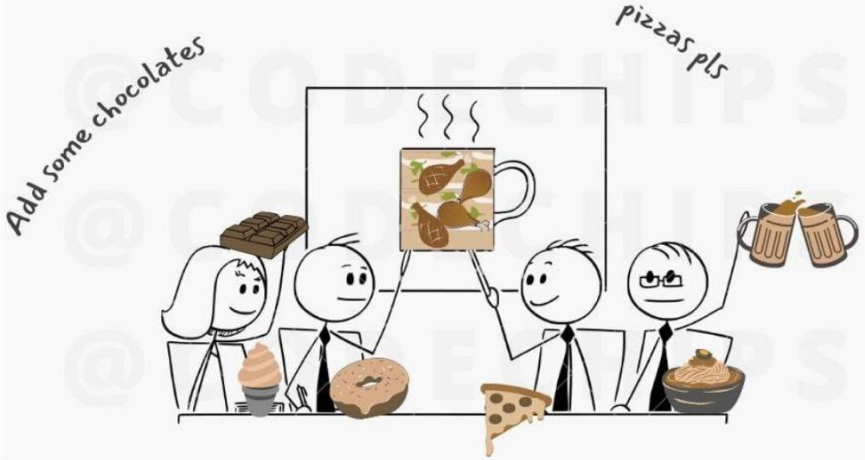
DigitalSkola
Uncover The World Of Digital Skills With Us

You are still not satisfied

So you again improvise your recipe and make a new 3rd version of your dish

And what if you have a team

and everyone wanted to taste each version of your dish and add their own ingredients and contribute to your masterpiece



@codechips Cody popupdev04@gmail.com

Apa itu Git..

DigitalSkola
Uncover The World Of Digital Skills With Us

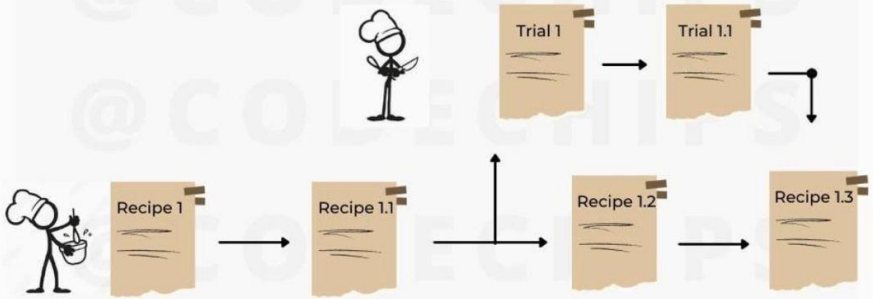
Everything is messed up

Wouldn't it be great if there was a **time machine** which stores all your recipes and dishes seperately so if something goes wrong you could go back to the previous dish

@codechips Cody popupdev04@gmail.com

That is where GIT comes into play

Git is a **distributed version control** system



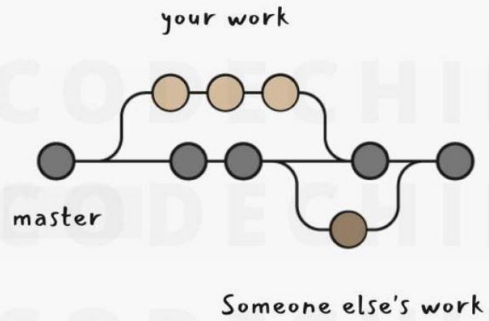
Git **tracks the changes** you made, so you have a record of what has been done, and you can revert to specific versions. It makes collaboration easier, allowing changes by multiple people to all be merged into one source

@codechips Cody popupdev04@gmail.com

Apa itu Git..

Earlier Developer would have their Backup source code in seperate folders. Reverting back and collaborating is a tedious job

Git can automatically merge the changes, so two people can even work on different parts of the same file and later merge those changes without losing each other's work!

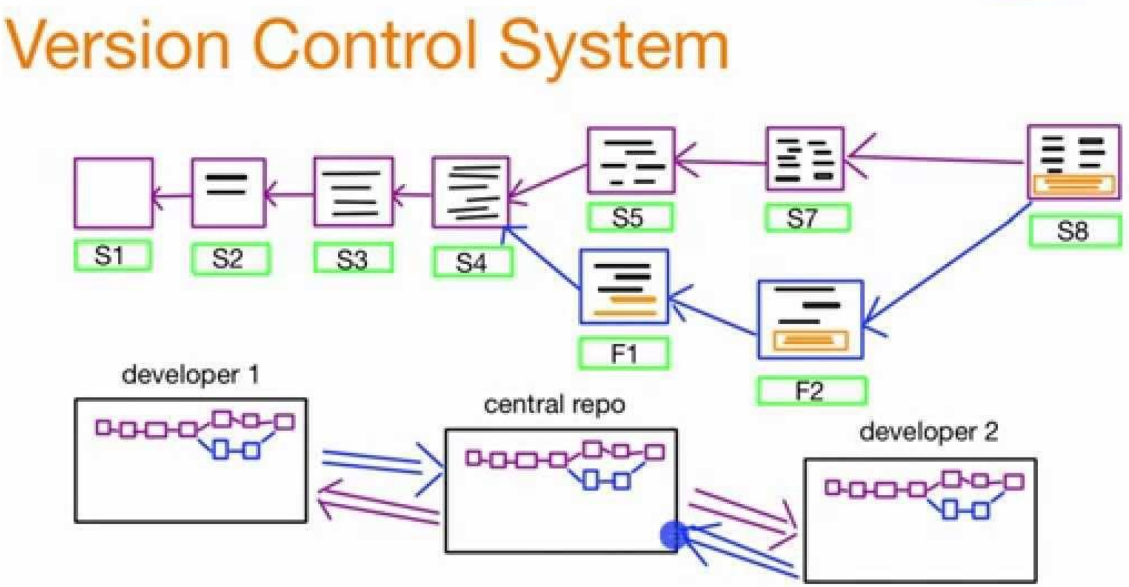


Version Control System (VCS)

Apa itu VCS?

Sebuah sistem yang merekam perubahan sekumpulan berkas dari waktu ke waktu sehingga kita dapat menilik kembali versi tertentu suatu saat nanti.

VCS memungkinkan kita untuk mengembalikan berkas-berkas ke keadaan semula, membandingkan perubahan di setiap waktu, melihat siapa yang terakhir mengubah sesuatu yang mungkin menimbulkan masalah, siapa dan kapan yang mengenalkan sebuah isu, dll.

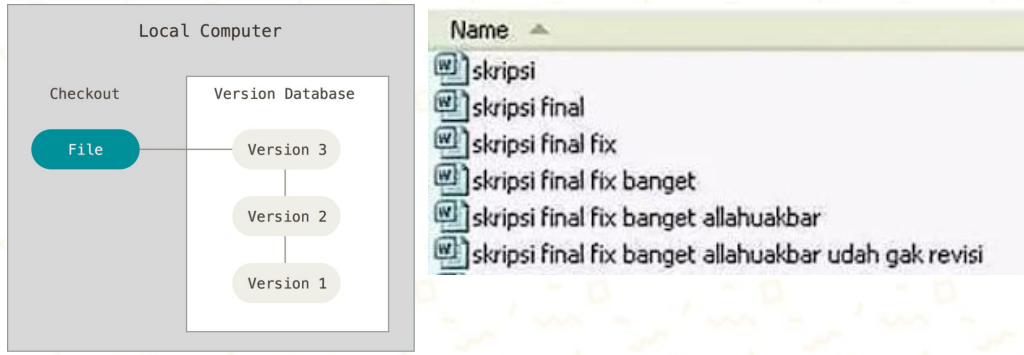


Version Control System (VCS)

DigitalSkola
Uncover The World Of Digital Skills With Us

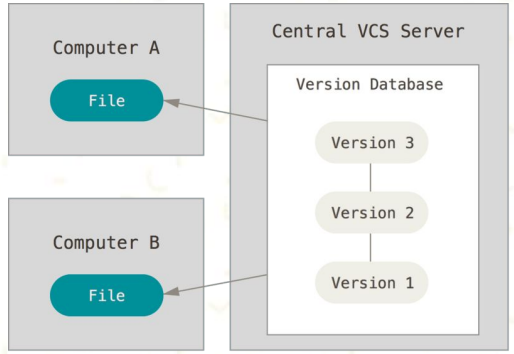
Sistem Version Control Lokal

Metode yang banyak dipilih oleh orang adalah dengan menyalin berkas-berkas ke direktori lain. Pendekatan ini sangat umum karena mudah, namun sangat susah untuk di-maintain. Kita sering lupa pada direktori mana kita menyimpan berkas, berkas tidak sengaja terhapus, dll.



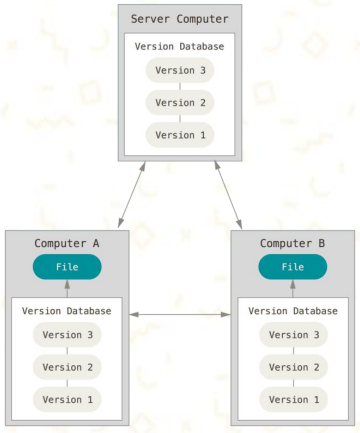
Sistem Version Control Terpusat

Masalah selanjutnya bagaimana kita bisa bekerja bersama dengan orang lain. Untuk menangani masalah ini, **Centralized Version Control System (CVCS)** dikembangkan.



Sistem Version Control Tersebar

Di **CVCS,** jika servernya mati, maka kita tidak dapat bekerja. Di sinilah **Distributed Version Control System (DVCS)** masuk, seperti Git.



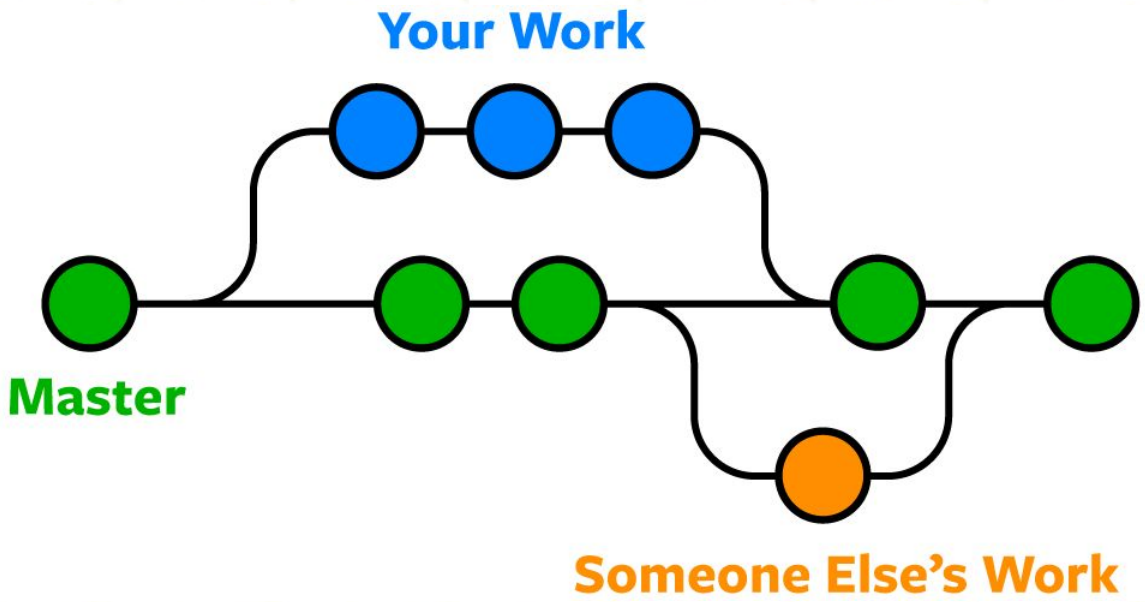
Git

DigitalSkola
Uncover The World Of Digital Skills With Us

Apa itu Git?

Git adalah VCS yang digunakan pengembang di seluruh dunia untuk melacak berbagai versi kode dan berkolaborasi dengan orang lain.

Kita bisa check apakah Git sudah terinstall dengan mengetikkan command pada command line tools `git --version`.
Instalasi git berdasarkan OS yang kita pakai dapat didapatkan di **https://git-scm.com/.**



Konsep Utama Git

Snapshots

Git merekam semua file pada waktu tertentu untuk melacak perubahan file. Kita dapat mengakses semua perubahan yang telah dilakukan.

Commit

Beberapa tindakan untuk membuat snapshots, terdiri dari informasi tentang perubahan sebelumnya, referensi dari commit sebelumnya, dan ditandai dengan sebuah hash-code seperti *'fn23edjfb23'*

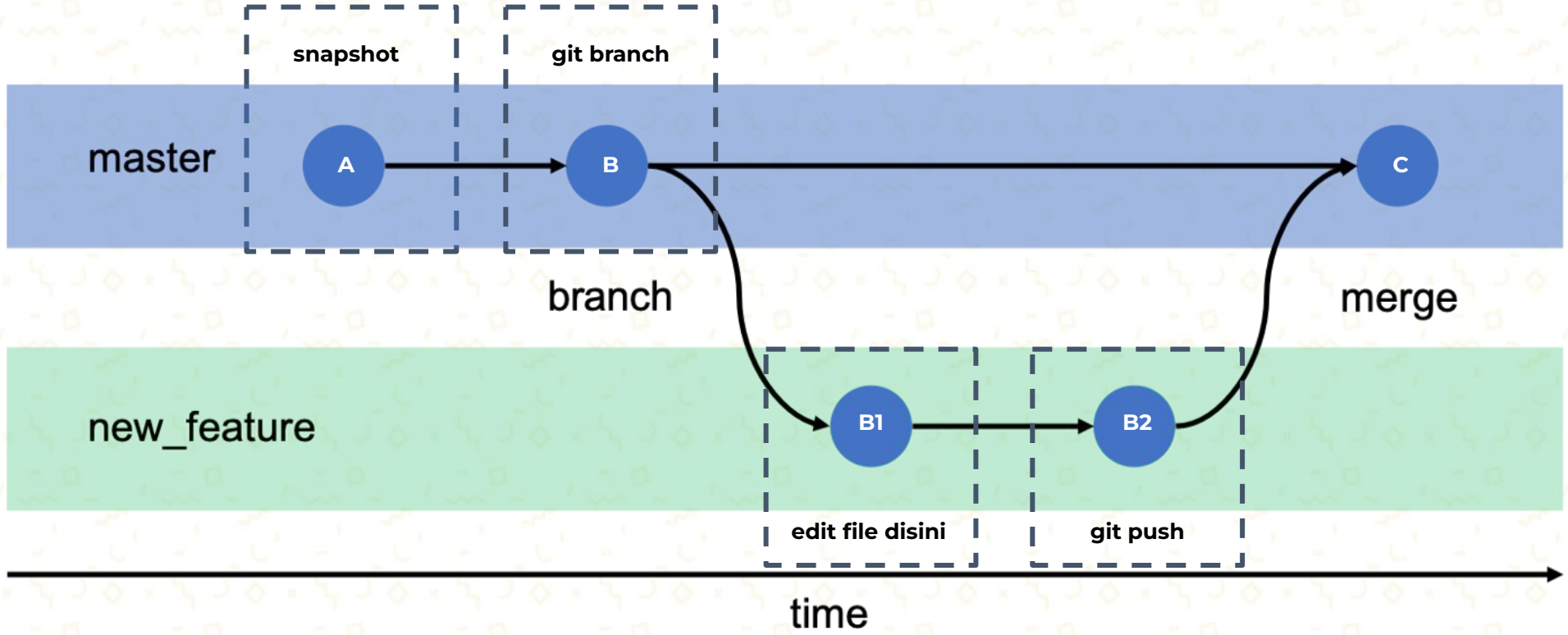
Repo

Kumpulan dari banyak file dan perubahannya, juga tempat dimana semua commit disimpan.

Branch

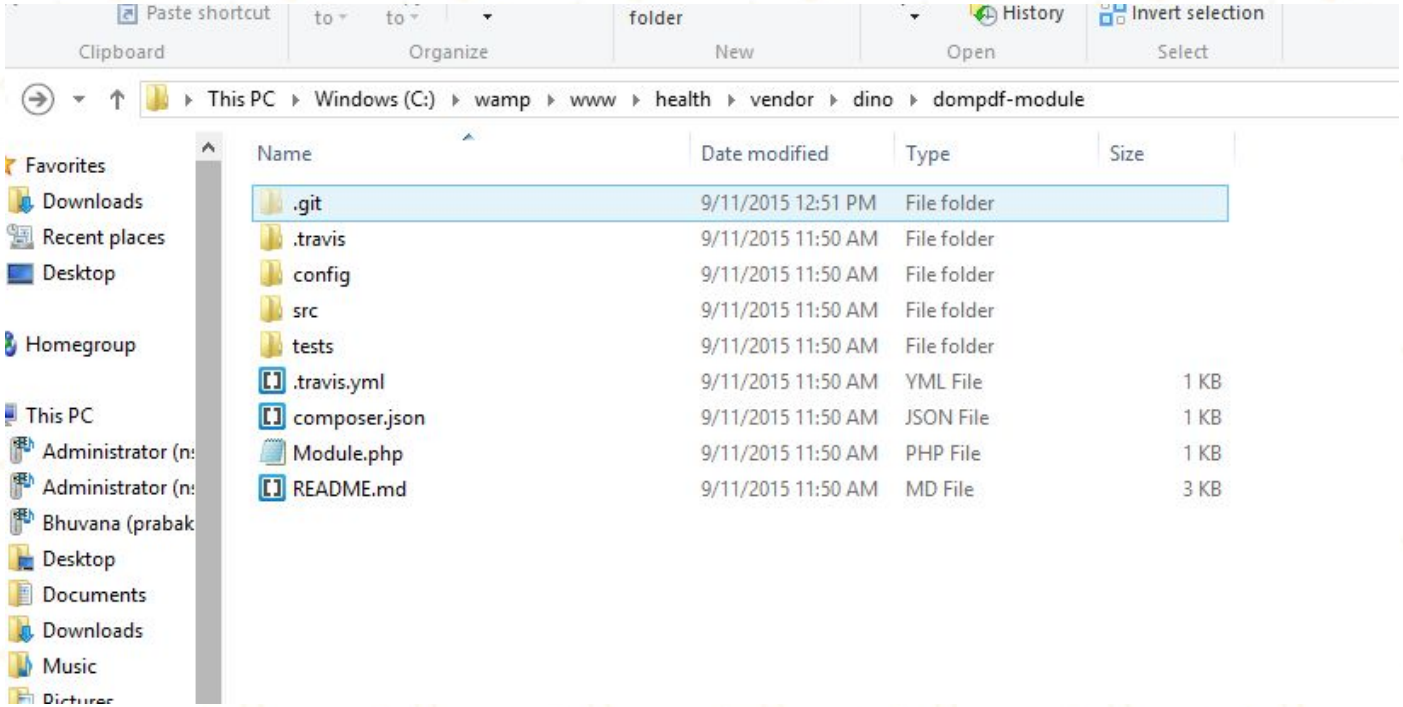
Semua commit terdapat didalam suatu branch. Repo dapat terdiri dari banyak branch, branch satu dan lainnya berdiri sendiri.

Konsep Utama Git



Directory, Repository, & .git directory

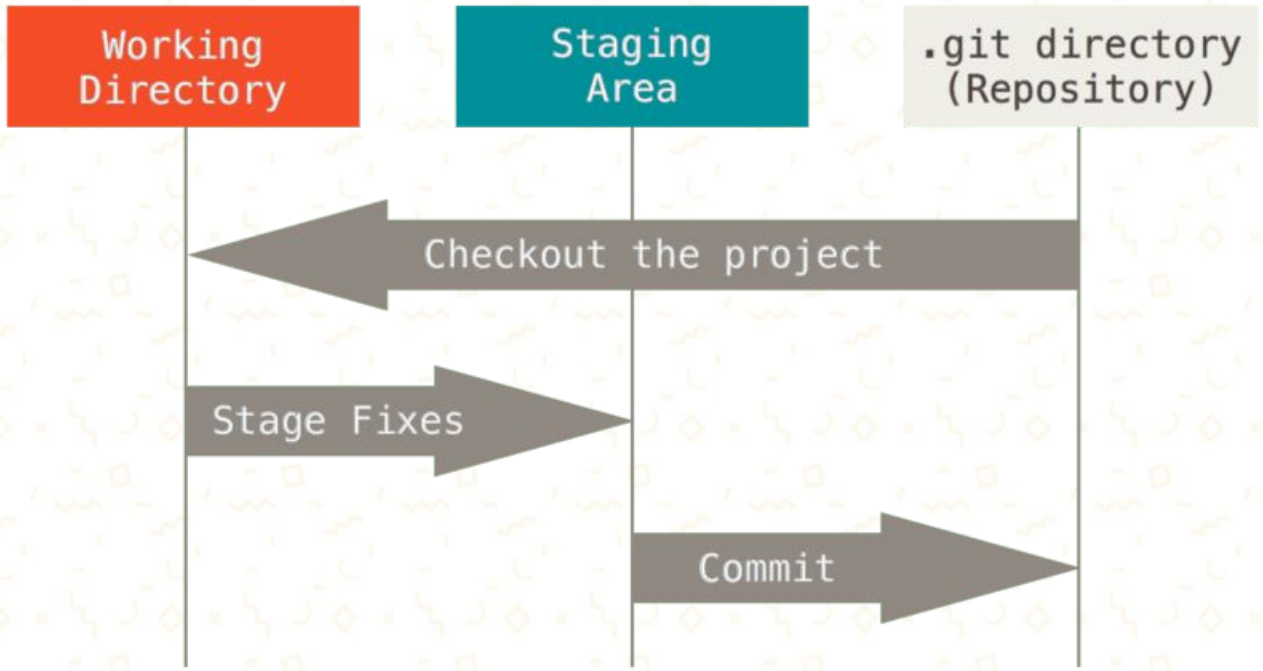
- directory: untuk menyimpan file yang ingin kita kerjakan
- repository: directory yang didalamnya sudah di inisiasi git (terdapat .git direktori didalamnya)
- .git directory: berisi metadata git, menyimpan history dari perubahan yang pernah terjadi



Git States & Areas

Tahapan

1. **Modified**: melakukan perubahan pada file namun belum di-commit
2. **Staged**: menandai file mana yang sudah diubah dan ingin di-commit
3. **Committed**: menyimpan perubahan dalam git



Areas

1. **Working dir**: dimana kita melakukan perubahan pada files
2. **Staging area**: berisi files yang sudah ditandai untuk di-commit
3. **.git dir:** berisi perubahan yang telah di-commit

Benefit Git

- Memudahkan kolaborasi satu project dengan orang lain
- Mudah menyelesaikan code conflict
- Memungkinkan undo / revert ke perubahan / versi sebelumnya

Github

DigitalSkola
Uncover The World Of Digital Skills With Us

Apa itu Github?

GitHub adalah produk yang memungkinkan kita untuk meng-host proyek Git di server jarak jauh atau cloud.

GitHub bukanlah Git. GitHub adalah layanan hosting. Ada perusahaan lain yang menawarkan layanan hosting yang melakukan hal yang sama seperti GitHub, seperti Bitbucket dan GitLab.

Git Module:
https://docs.google.com/document/d/1jdtkmZgAK89xtG0j5TNkwAISHa67vUUNpCNOfy1LzCI

GitHub

GitLab

ATLASSIAN
Bitbucket

Git commands

DigitalSkola
Uncover The World Of Digital Skills With Us

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to HEAD.
`git reflog`	Show a log of changes to the local repository's HEAD. Add `--relative-date` flag to show date info or `--all` to show all refs.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

Git commands

DigitalSkola
Uncover The World Of Digital Skills With Us

GIT CONFIG

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline.`
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. `<editor>` arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

GIT LOG

`git log -<limit>`	Limit number of commits by `<limit>`. E.g. `"git log -5"` will limit to 5 commits.
`git log --oneline`	Condense each commit to a single line.
`git log -p`	Display the full diff of each commit.
`git log --stat`	Include which files were altered and the relative number of lines that were added or deleted from each of them.
`git log --author= "<pattern>"`	Search for commits by a particular author.
`git log --grep="<pattern>"`	Search for commits with a commit message that matches `<pattern>`.
`git log <since>..<until>`	Show commits that occur between `<since>` and `<until>`. Args can be a commit ID, branch name, `HEAD`, or any other kind of revision reference.
`git log -- <file>`	Only display commits that have the specified file.
`git log --graph --decorate`	`--graph` flag draws a text based graph of commits on left side of commit msgs. `--decorate` adds names of branches or tags of commits shown.

GIT DIFF

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

GIT RESET

`git reset`	Reset staging area to match most recent commit, but leave the working directory unchanged.
`git reset --hard`	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
`git reset <commit>`	Move the current branch tip backward to `<commit>`, reset the staging area to match, but leave the working directory alone.
`git reset --hard <commit>`	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** `<commit>`.

GIT REBASE

`git rebase -i <base>`	Interactively rebase current branch onto `<base>`. Launches editor to enter commands for how each commit will be transferred to the new base.

GIT PULL

`git pull --rebase <remote>`	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

GIT PUSH

`git push <remote> --force`	Forces the `git push` even if it results in a non-fast-forward merge. Do not use the `--force` flag unless you're absolutely sure you know what you're doing.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	Tags aren't automatically pushed when you push a branch or use the `--all` flag. The `--tags` flag sends all of your local tags to the remote repo.

DigitalSkola
Uncover The World Of Digital Skills With Us

References

https://git-scm.com/

https://www.atlassian.com/git/tutorials

DigitalSkola

Uncover The World Of Digital Skills With Us

Thank YOU