



TUTORIAL PENGGUNAAN GIT DAN GITLAB

Departemen Ilmu Komputer

Institut Pertanian Bogor

<http://apps.cs.ipb.ac.id:2000>



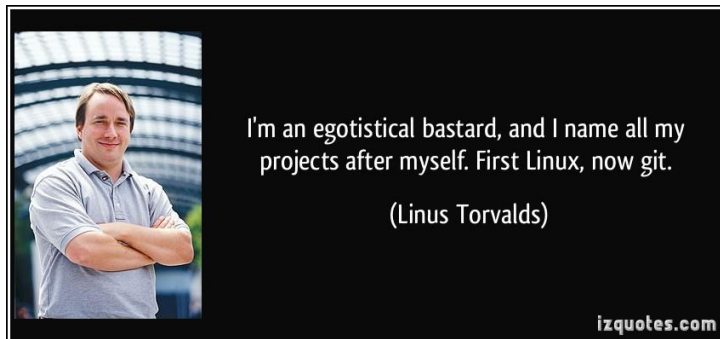
Pendahuluan

Perkenalan

Hallo teman-teman ilkomerz IPB! Perkenalkan, saya Arief Hidayatulloh, penulis naskah ini. Sebagai ilkomerz tentu kita tidak boleh mengabaikan suatu kegiatan bernama *coding*. Walaupun tidak semua mahasiswa akan menjadi *codingers*, kita wajib tahu tentang *coding* dan *tools* yang digunakan untuk melakukan ritual tersebut. Salah satu jenis *tools* yang diperlukan untuk *coding* adalah *version control*, salah satunya adalah git. Maka izinkan saya untuk memperkenalkan sedikit dasar tentang git.

Tentang Git

Git diciptakan oleh Linus Torvalds. Ya, Anda benar, Linus Torvalds *yang itu*. Si pembuat Linux. Git digunakan oleh developer a.k.a. kuli *coding* untuk menyimpan perubahan source code, yang disebut juga sistem version control. Git dapat digunakan sendiri maupun untuk kolaborasi dengan team. Git bersifat terdistribusi dan 'individual', sehingga jika salah satu server mati, developer dapat menggunakan server lain dengan mudah. Jika developer tidak terhubung dengan internet, git masih dapat digunakan secara *offline*, bahkan developer bisa melihat *history* kode-kodenya tanpa perlu terhubung ke *remote server*.



Teori Version Control System

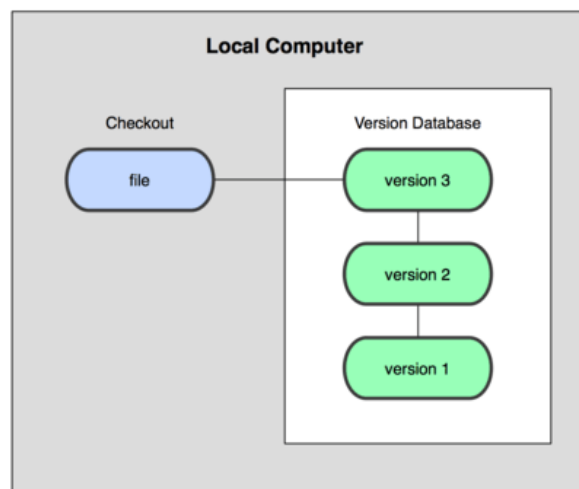
Version control adalah sebuah sistem yang mencatat setiap perubahan terhadap sebuah berkas atau kumpulan berkas sehingga pada suatu saat anda dapat kembali kepada salah satu versi dari berkas tersebut.

Misalnya, jika anda adalah seorang desainer grafis atau desainer web dan anda ingin menyimpan setiap versi dari gambar atau layout yang anda buat, maka Version Control System (VCS) merupakan sebuah solusi untuk digunakan. Sistem ini memungkinkan anda untuk mengembalikan berkas anda pada kondisi/keadaan sebelumnya, mengembalikan seluruh proyek pada keadaan sebelumnya, membandingkan perubahan setiap saat, melihat siapa yang terakhir melakukan perubahan terbaru pada suatu objek, dan lainnya. Dengan menggunakan VCS dapat berarti jika anda telah mengacaukan atau kehilangan berkas, anda dapat dengan mudah mengembalikannya.

Version Control System Lokal

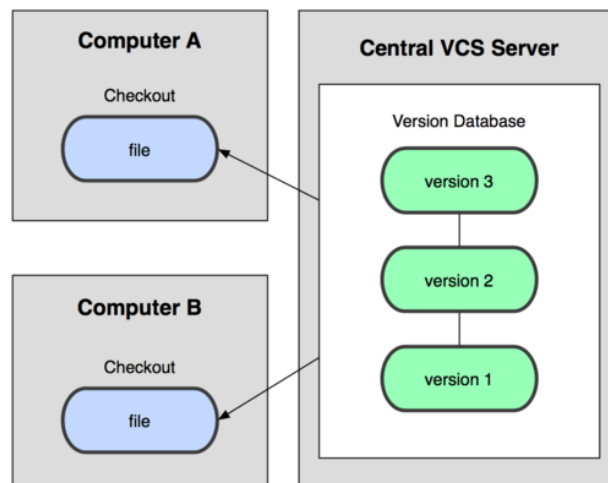
Kebanyakan orang melakukan pengontrolan versi dengan cara menyalin berkas-berkas pada direktori lain (mungkin dengan memberikan penanggalan pada direktori tersebut, jika mereka rajin). Metode seperti ini sangat umum karena sangat sederhana, namun cenderung rawan terhadap kesalahan. Anda akan sangat mudah lupa letak direktori anda sedang berada, selain itu dapat pula terjadi ketidaksengajaan penulisan pada berkas yang salah atau menyalin pada berkas yang bukan anda maksudkan.

Untuk mengatasi permasalahan ini, para programmer mengembangkan berbagai VCS lokal yang memiliki sebuah basis data sederhana untuk menyimpan semua perubahan pada berkas yang berada dalam cakupan revision control. Ilustrasinya dapat dilihat pada gambar berikut.



Version Control System Terpusat

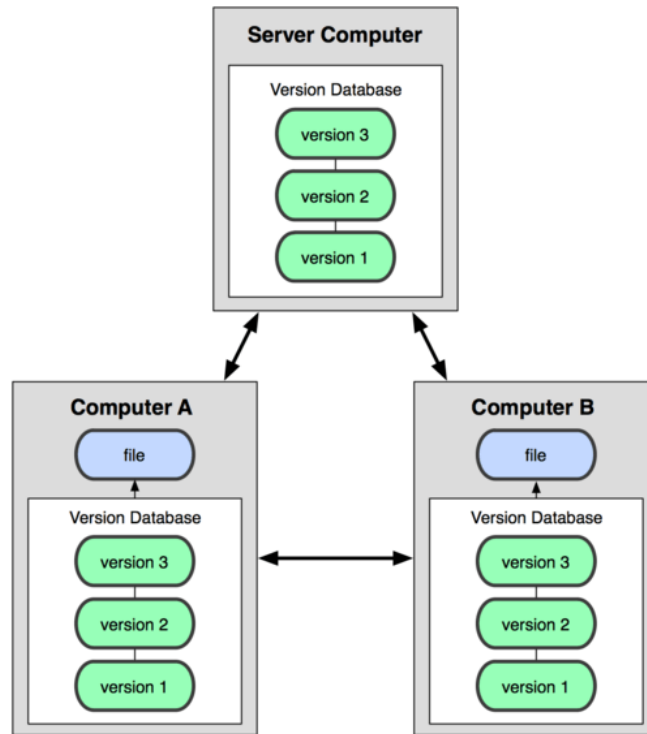
Permasalahan berikutnya yang dihadapi adalah para pengembang perlu melakukan kolaborasi dengan pengembang pada sistem lainnya. Untuk mengatasi permasalahan ini maka dibangunlah Centralized Version Control Systems (CVCS). Sistem ini, di antaranya CVS dan Subversion (SVN) memiliki sebuah server untuk menyimpan setiap versi berkas, dan beberapa klien yang dapat melakukan checkout berkas dari server pusat. Untuk beberapa tahun, sistem seperti ini menjadi standard untuk version control. Ilustrasi CVCS pada gambar berikut:



Sistem seperti ini memiliki beberapa kelebihan, misalnya, setiap orang dapat mengetahui apa yang orang lain lakukan pada proyek. Walau demikian, sistem dengan tatanan seperti ini memiliki kelemahan serius. Kelemahan nyata yang direpresentasikan oleh sistem dengan server terpusat. Jika server mati untuk beberapa jam, maka tidak ada seorang pun yang bisa berkolaborasi atau menyimpan perubahan terhadap apa yang mereka telah kerjakan.

Version Control System Terdistribusi

Dalam sebuah DVCS (Distributed Version Control System) seperti Git, klien tidak hanya melakukan checkout untuk snapshot terakhir setiap berkas, namun mereka (klien) memiliki salinan penuh dari repositori tersebut. Jadi, jika server mati, dan sistem berkolaborasi melalui server tersebut, maka klien manapun dapat mengirimkan salinan repositori tersebut kembali ke server. Setiap checkout pada DVCS merupakan sebuah backup dari keseluruhan data.



Dengan DVCS, seandainya pun satu server repository lenyap beserta data di dalamnya, anda dan rekan-rekan anda sesama developer masih tetap memiliki salinan lengkap dari repository, dan dapat menggunakan server yang lain yang masih hidup.

Konsep Dasar Git

Istilah-istilah dalam Git

Selama menggunakan Git, anda akan banyak menemui istilah-istilah baru. Jangan khawatir bila istilah yang dijelaskan di sini belum bisa dipahami. Seiring dengan pemahaman, istilah-istilah ini akan semakin masuk akal (semoga). Berikut ini beberapa istilah tersebut.

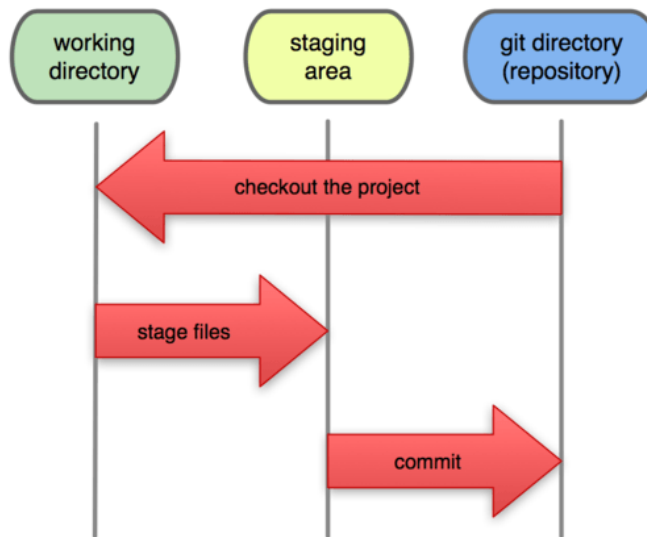
- *repository* : database yang menyimpan semua history/riwayat perubahan.
- *snapshot* : potret kondisi file dan folder pada saat tertentu.
- *commit* : snapshot yang disimpan di repository.
- *branch* : serangkaian commit yang berkaitan sehingga kalau digambar seperti garis lurus berisi banyak commit. Satu repository bisa berisi banyak branch.
- *master* : nama branch default yang diberikan git pada waktu kita membuat repository. Branch master ini tidak istimewa. Dia bisa dihapus dan direname sesuka hati.
- *hash* : Git menyimpan informasi commit sebagai hash SHA1, misalnya `24b9da6552252987aa493b52f8696cd6d3b00373`. Namun terkadang ditampilkan versi pendeknya, misalnya `24b9da6`.
- *head* : ujung branch, commit terbaru di dalam branch.
- *HEAD* : head yang sedang aktif. Walaupun satu repository bisa memiliki banyak branch, tapi cuma satu yang aktif.
- *working folder* : folder berisi file dan folder tempat kita bekerja. Biasanya working folder berisi banyak file source code untuk aplikasi yang sedang kita buat. Git memantau working folder ini, dan bisa mengetahui file dan folder mana yang sudah berbeda dari posisi commit terakhir. Perbedaan atau perubahan ini bisa disimpan menjadi commit baru, atau dikembalikan ke kondisi sebelum diubah.
- *staging area* : snapshot dari working folder yang akan kita simpan pada saat commit. Ini adalah fitur unik Git yang tidak dimiliki version control lain. Dengan adanya staging area, kita bisa memilih perubahan mana yang akan di-commit dan mana yang tidak.

Tiga Keadaan

Git memiliki 3 keadaan utama di mana berkas anda dapat berada: *committed*, *modified* dan *staged*. Committed berarti data telah tersimpan secara aman pada basisdata lokal. Modified berarti anda telah melakukan perubahan pada berkas namun anda belum melakukan commit pada basisdata. Staged berarti anda telah menandai berkas yang telah diubah pada versi yang sedang berlangsung untuk kemudian dilakukan commit.

Ini membawa kita ke tiga bagian utama dari sebuah proyek Git: direktori Git, direktori kerja (working directory), dan staging area.

Local Operations



Direktori Git adalah tempat Git menyimpan metadata dan database objek untuk proyek anda. Ini adalah bahagian terpenting dari Git, dan inilah yang disalin ketika anda melakukan kloning sebuah repository dari komputer lain.

Direktori kerja adalah sebuah checkout tunggal dari satu versi dari proyek. Berkas-berkas ini kemudian ditarik keluar dari basisdata yang terkompresi dalam direktori Git dan disimpan pada disk untuk anda gunakan atau modifikasi.

Staging area adalah sebuah berkas sederhana, umumnya berada dalam direktori Git anda, yang menyimpan informasi mengenai apa yang menjadi commit selanjutnya. Ini terkadang disebut sebagai index, tetapi semakin menjadi standard untuk menyebutnya sebagai staging area.

Alur kerja dasar Git adalah seperti ini:

1. Anda mengubah berkas dalam direktori kerja anda.
2. Anda membawa berkas ke stage, menambahkan snapshotnya ke staging area.
3. Anda melakukan commit, yang mengambil berkas seperti yang ada di staging area dan menyimpan snapshotnya secara permanen ke direktori Git anda.

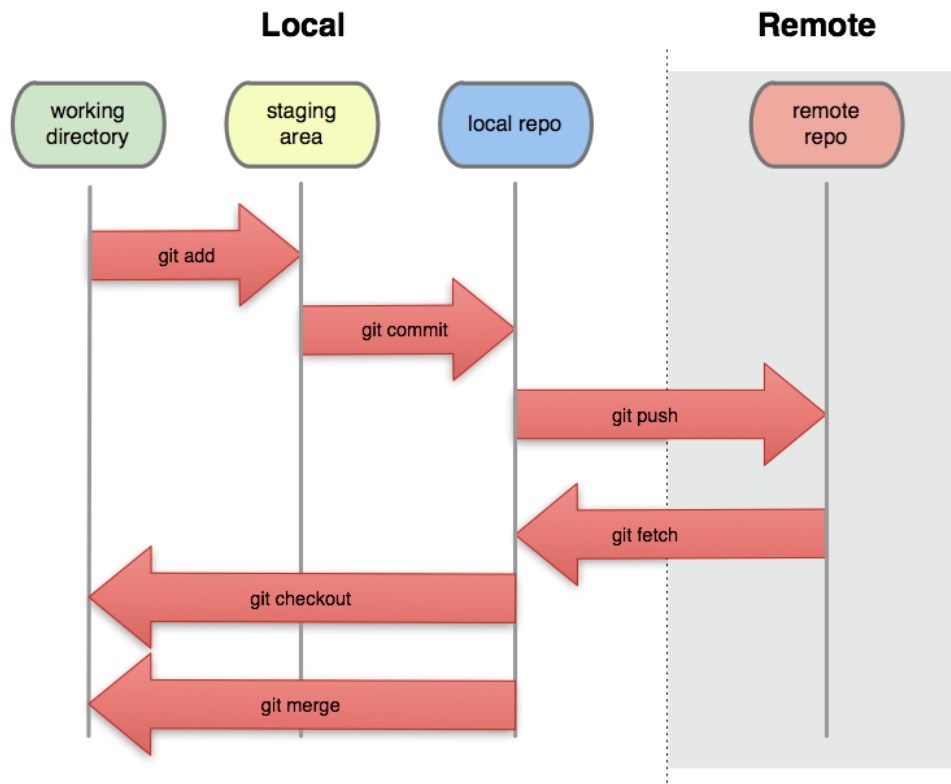
Jika sebuah versi tertentu dari sebuah berkas telah ada di direktori git, ia dianggap 'committed'. Jika berkas diubah (modified) tetapi sudah ditambahkan ke staging area, maka itu adalah 'staged'. Dan jika berkas telah diubah sejak terakhir dilakukan checked out tetapi belum ditambahkan ke staging area maka itu adalah 'modified'.

Terakhir, ingat bahwa seluruh proses tersebut terjadi hanya di komputer lokal anda.

Ketika Server Repository Terlibat

Dalam Git, seringkali anda memerlukan suatu server penyedia layanan repository. Server ini dalam terminologi Git disebut sebagai “remote”. Server ini menyediakan tempat terpusat di internet sehingga developer lain dapat berkolaborasi dengan perantara server tersebut.

Ketika anda telah menyelesaikan operasi di komputer lokal anda (add, commit), anda dapat menyimpan keadaan repository lokal anda ke server. Kegiatan ini disebut sebagai “push”. Ketika anda “push”, maka keadaan repository remote akan disamakan dengan keadaan repository lokal anda.



Operasi-operasi Dasar

Init

Perintah **init** digunakan untuk inisiasi git. Biasanya inisiasi dilakukan oleh pemimpin proyek. Anggota lain akan melakukan clone setelah pemimpin proyek melakukan inisiasi repository. Init dapat digunakan di proyek baru (masih kosong) atau di proyek yang sudah dikerjakan (sudah ada *file source code*).

Clone

Perintah **clone** digunakan untuk menyalin repository dari *remote repository* ke *local repository*.

Add

Perintah add digunakan untuk menambahkan file ke *staging area*.

Commit

Perintah commit digunakan untuk menyimpan perubahan kode di repository local.

Push

Perintah push digunakan untuk mengirim commit dari *local repository* ke *remote server*.

Checkout

Perintah checkout digunakan untuk berpindah dari satu branch ke branch lain. Checkout juga digunakan untuk mengembalikan *file* yang diubah tapi belum *dicommit* ke versi sebelum diedit.

Fetch

Perintah fetch digunakan untuk menyamakan keadaan remote repo dengan local repo (meng-update local repo).

Merge

Perintah merge digunakan untuk menggabungkan branch.

Pull

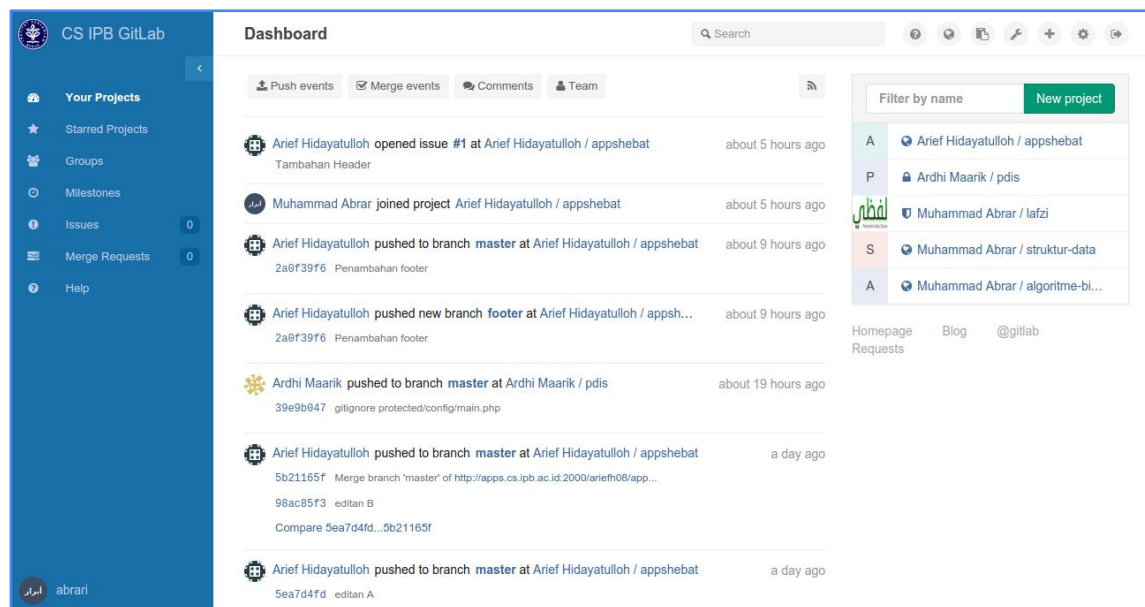
Perintah pull digunakan untuk menarik commit dari remote server ke lokal. Aslinya, pull ini melakukan fetch yang diikuti merge secara otomatis.

GitLab Ilkom

Departemen Ilmu Komputer IPB telah meng-*host* suatu server repository Git dengan menggunakan GitLab yang merupakan aplikasi web. Selain sebagai server repository Git, GitLab juga menyediakan fitur-fitur tambahan yang berguna untuk kolaborasi *coding* antar-beberapa developer. Fitur-fitur tersebut antara lain:

- Issue tracking, untuk melaporkan kalau-kalau ada bug dalam kode.
- Wiki, bisa digunakan untuk berbagai dokumentasi proyek.
- Adanya otentikasi sehingga hanya anda atau anggota tim yang punya akses.
- Komentar terhadap commit, sebagai bentuk code review.

Untuk mengaksesnya, anda harus punya akun IPB, atau email @apps.ipb.ac.id.



Peran GitLab di dalam *coding* anda hanyalah sebagai media untuk menyimpan source code dan memfasilitasi kolaborasi secara online. Jadi, jangan mengira bahwa di GitLab anda bisa mengetikkan kode program secara langsung (GitLab bukan sebuah IDE), atau menjalankan program yang anda buat di server (GitLab bukan tempat untuk deployment).

Untuk memanfaatkan GitLab sebagai tempat penyimpanan source code, anda membutuhkan software **git** yang terinstal di komputer anda. Si **git** ini nanti yang akan berkomunikasi dengan GitLab untuk mengambil dan menyimpan kode anda dari dan ke server. Sementara itu, untuk mengetik kode, silakan gunakan IDE favorit anda.

Memulai

Instalasi

Instalasi di Ubuntu sebagai berikut:

```
sudo apt-get update
sudo apt-get install git
```

Untuk instalasi git client di Windows, silakan unduh binari di <https://windows.github.com/>, kemudian lakukan instalasi. Untuk menjalankan terminal seperti di linux, jalankan **git shell**.

Konfigurasi

Sebelum mulai menggunakan git, sebaiknya dibuat dulu konfigurasi global git untuk identitas pengguna. Langkahnya sebagai berikut:

```
git config --global user.name "Arief Hidayatulloh"
git config --global user.email ariefsam@gmail.com
```

Cek Instalasi

Setelah selesai, kita bisa test dengan membuka command prompt dan mengetik perintah:

git

Kalau instalasi berjalan lancar, maka akan muncul output dari git sebagai berikut.

```
usage: git [--version] [--exec-path=<path>] [--html-path]
        [-p|--paginate|--no-pager] [--no-replace-objects]
        [--bare] [--git-dir=<path>] [--work-tree=<path>]
        [-c name=value] [--help]
        <command> [<args>]

The most commonly used git commands are:
  add           Add file contents to the index
  bisect        Find by binary search the change that introduced a bug
  branch        List, create, or delete branches
  checkout      Checkout a branch or paths to the working tree
  clone         Clone a repository into a new directory
  commit        Record changes to the repository
  diff          Show changes between commits, commit and working tree, etc
  ...
  pull         Fetch from and merge with another repository or a local branch
  push          Update remote refs along with associated objects
  rebase        Forward-port local commits to the updated upstream head
  reset         Reset current HEAD to the specified state
  rm            Remove files from the working tree and from the index
  show          Show various types of objects
  status        Show the working tree status
  tag           Create, list, delete or verify a tag object signed with GPG

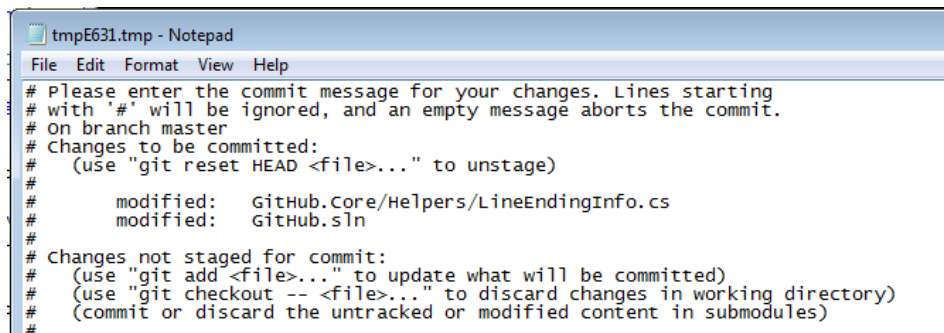
See 'git help <command>' for more information on a specific command.
```

Git Default Editor di Windows

Default editor Git shell di windows adalah vim yang berbasis command line. Untuk sebagian besar orang editor ini sulit untuk digunakan. Oleh karena itu sebaiknya kita ganti editornya ke notepad. Untuk mengganti editor, download program “GitPad” pada link berikut:

<https://github.com/downloads/github/GitPad/Gitpad.zip>

Di dalamnya ada file EXE yang dapat dijalankan. Setelah itu, default editor git kita akan berganti menjadi Notepad.



```
File Edit Format View Help
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   GitHub.Core/Helpers/LineEndingInfo.cs
#       modified:   GitHub.sln
#
# changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#   (commit or discard the untracked or modified content in submodules)
#
```

Praktik Dasar

Pada bagian ini kita akan mulai praktik menggunakan git. Sebelumnya harap buat kelompok, setiap kelompok dipimpin oleh seorang project manager.

Buat repository

Pembuatan repository dilakukan oleh project manager. Buat repository baru di GitLab.

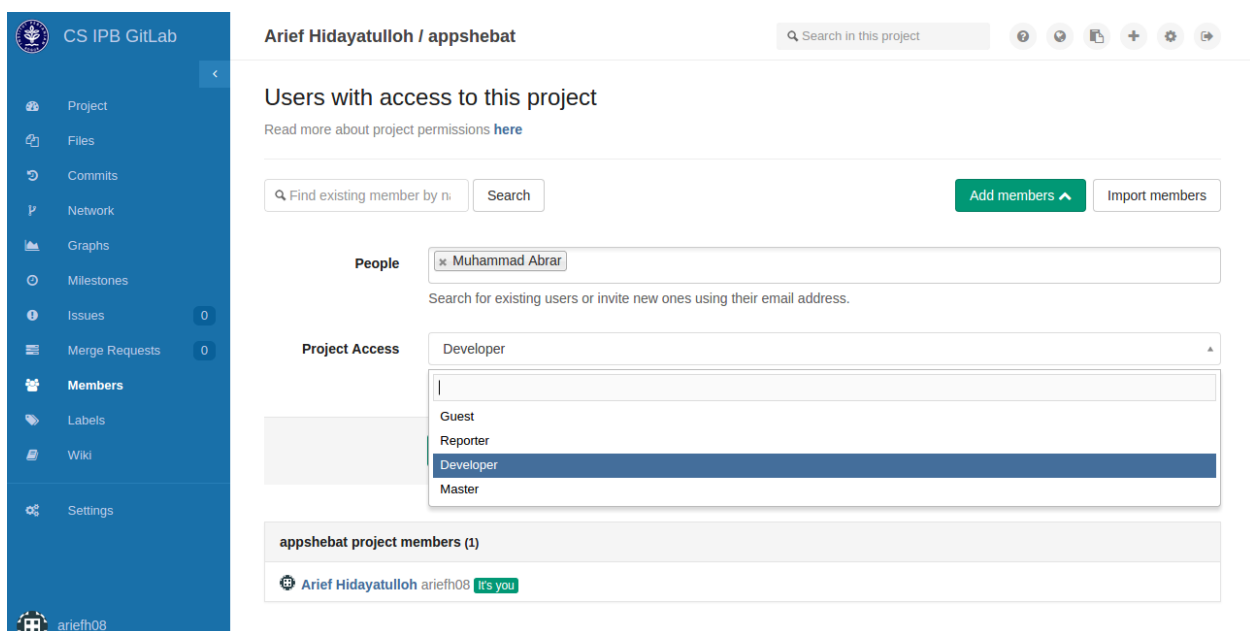
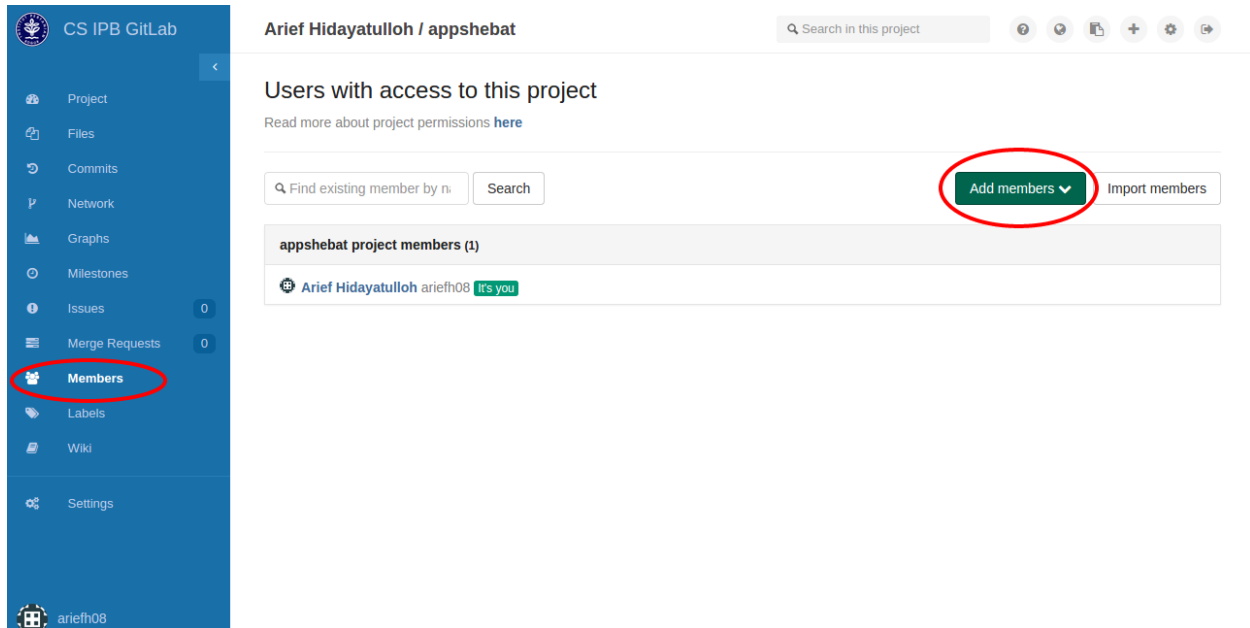
The image displays two screenshots of the CS IPB GitLab web interface. The top screenshot shows the 'Dashboard' page. On the left is a blue sidebar with navigation links: 'Your Projects', 'Starred Projects', 'Groups', 'Milestones', 'Issues' (0), 'Merge Requests' (0), and 'Help'. The main content area has a header with a search bar and a notification: 'If you sign in using IPB Account, please set your password for GitLab. Open your Profile settings -- Password.' Below this, it says 'Welcome to GitLab!' and 'Self hosted Git management application.' A message states: 'You don't have access to any projects right now. You can create up to 10 projects.' A green button labeled '+ New Project' is circled in red. Below this, it says 'You can create a group for several dependent projects. Groups are the best way to manage projects and members.' with a '+ New Group' button. The bottom screenshot shows the 'New Project' form. It has a 'Project path' field with 'appshebat' and a '.git' suffix. Below is 'Import project from' with buttons for GitHub, Bitbucket, GitLab.com, Gitorious.org, Google Code, and 'git Any repo by URL'. The 'Description (optional)' field contains 'Apps yang sangat hebat'. The 'Visibility Level (?)' section has three radio buttons: 'Private' (selected), 'Internal', and 'Public'. Each has a description: 'Private' (Project access must be granted explicitly for each user.), 'Internal' (The project can be cloned by any logged in user.), and 'Public' (The project can be cloned without any authentication.). At the bottom is a 'Create project' button and a link 'Need a group for several dependent projects? Create a group'.

Isi semua *field* yang dibutuhkan. Pilih *visibility level* yang diinginkan. Apa itu *visibility level*?

- *Private*, proyek hanya bisa diakses oleh member di proyek itu saja.
- *Internal*, proyek bisa diakses oleh semua member yang terdaftar di CS IPB GitLab.
- *Public*, proyek bisa diakses semua orang tanpa otentikasi.

Setelah membuat repository, tambahkan member. Apa perbedaan member dengan non member? Intinya member punya akses menulis sesuai levelnya, sedangkan non member hanya bisa clone saja. Jika proyek bersifat *private* proyek hanya bisa di-*clone* oleh member saja. Untuk menambahkan member, tekan menu **Members->Add members**.

Ketik username member yang akan dimasukkan, kemudian berikan *project access* yang diinginkan. Sesama developer sebaiknya memiliki *project access* **developer** atau **master**. Setelah itu tekan **Add user to Project**.



Inisiasi git

Inisiasi masih dilakukan oleh project manager.

Buat folder proyek lalu buat file readme.md di dalamnya, isi bebas. Contoh:

```
adplus@GREEN-HORNET:~/public_html$ mkdir appshebat
adplus@GREEN-HORNET:~/public_html$ cd appshebat/
adplus@GREEN-HORNET:~/public_html/appshebat$ echo "Inisialisasi Projek Apps Hebat" > README.md
adplus@GREEN-HORNET:~/public_html/appshebat$ ls
README.md
adplus@GREEN-HORNET:~/public_html/appshebat$
```

Dari terminal (git shell jika menggunakan windows), masuk ke path proyek.

Lakukan inisiasi dengan perintah berikut, remote server disesuaikan dengan repository masing-masing kelompok.

```
git init
git remote add origin http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat.git
git status
```

Perintah **remote add** digunakan untuk menambahkan *remote repository*. Satu proyek boleh memiliki lebih dari satu *repository*. Pada contoh di atas repository kita disimpan dengan nama **origin**.

```
adplus@GREEN-HORNET:~/public_html/appshebat$ git init
Initialized empty Git repository in /home/adplus/public_html/appshebat/.git/
adplus@GREEN-HORNET:~/public_html/appshebat$ git remote add origin http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat.git
adplus@GREEN-HORNET:~/public_html/appshebat$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md

nothing added to commit but untracked files present (use "git add" to track)
adplus@GREEN-HORNET:~/public_html/appshebat$
```

Selanjutnya perintah git status akan menampilkan status git saat ini, di sana terlihat file README.md belum *ditrack* oleh git. Tambahkan README.md ke *staging area* dengan perintah berikut:

```
git add README.md
```

```
adplus@GREEN-HORNET:~/public_html/appshebat$ git add README.md
adplus@GREEN-HORNET:~/public_html/appshebat$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.md

adplus@GREEN-HORNET:~/public_html/appshebat$
```

Jika kita melihat status, maka *file* README.md sudah masuk *staging area* dan kita dapat melakukan commit untuk menyimpan perubahan ke repository lokal (keterangan: opsi `-m` digunakan untuk memasukkan komentar kita terhadap commit ini).

```
git commit README.md -m 'Penambahan README'
```

```
adplus@GREEN-HORNET:~/public_html/appshebat$ git commit README.md -m 'Penambahan README'
[master (root-commit) 4336067] Penambahan README
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
adplus@GREEN-HORNET:~/public_html/appshebat$
```

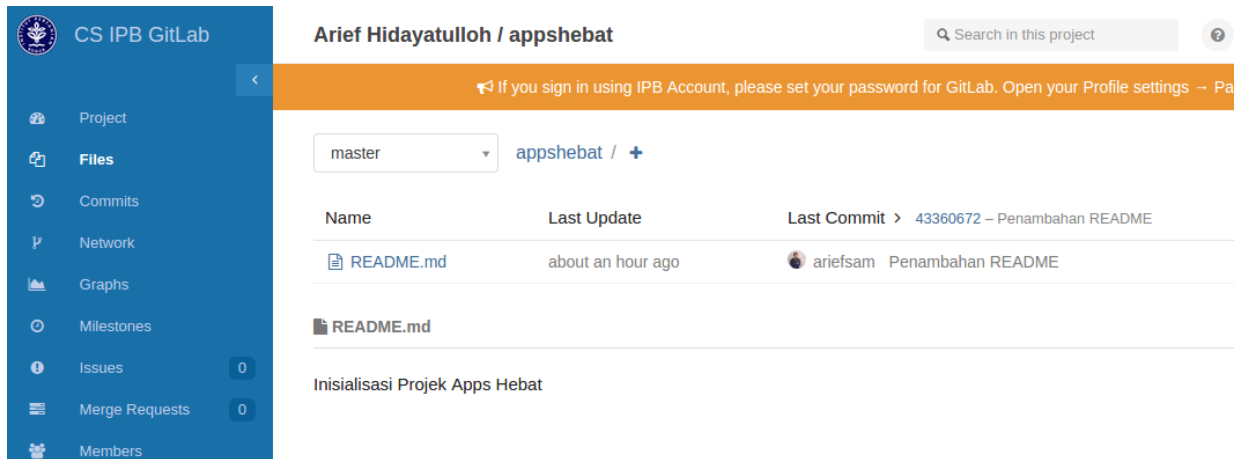
Dengan melakukan commit maka kita sudah menyimpan perubahan di *local repository*. Setelah itu kita dapat mengirimkan semua commit ke server dengan perintah push sebagai berikut:

```
git push origin master
```

Jika diminta input, masukkan username dan password GitLab Anda.

```
adplus@GREEN-HORNET:~/public_html/appshebat$ git push origin master
Username for 'http://apps.cs.ipb.ac.id:2000': ariefh08
Password for 'http://ariefh08@apps.cs.ipb.ac.id:2000':
Counting objects: 3, done.
Writing objects: 100% (3/3), 257 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: =====
remote:
remote:      If you sign in using IPB Account, please set your password for
remote:      GitLab. Open your Profile settings → Password.
remote:
remote: =====
To http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat.git
 * [new branch]      master -> master
adplus@GREEN-HORNET:~/public_html/appshebat$
```

Mari kita buka proyek di GitLab Ilkom IPB, pilih menu “Files”, maka file README.md sudah ada di server.



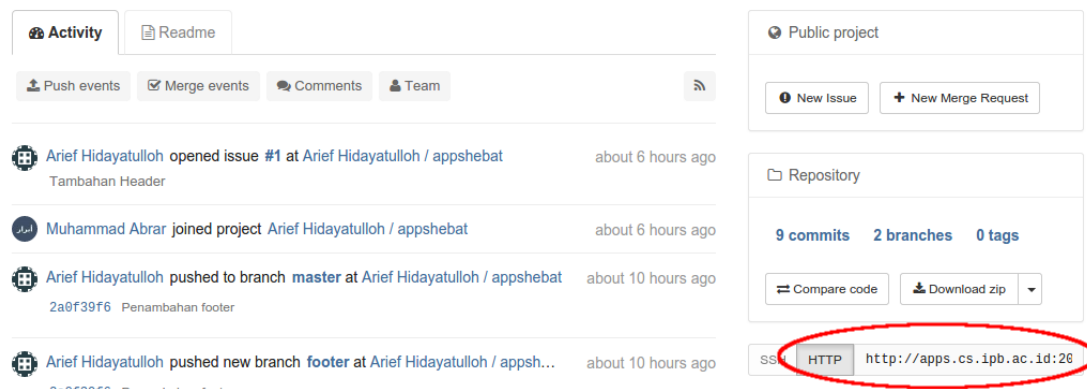
Setelah push ke server, developer lain dapat melakukan clone atau pull dan dapat mulai berkolaborasi.

Clone Repository

Setelah *project manager* menyiapkan *repository*, developer lain melakukan clone dengan perintah seperti di bawah ini (alamat *repository* disesuaikan) :

```
git clone http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat.git
```

Keterangan: alamat repository dapat dilihat pada bagian kanan bawah di web GitLab (ada suatu tulisan “http” sampai akhir, silakan di-copy).



```
$git clone http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat.git
Cloning into 'appshebat'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
$ls
appshebat
$cd appshebat/
$ls
README.md
$
```

Setelah itu akan ada folder proyek yang kita clone. Jika kita masuk ke folder tersebut akan ada file README.md di sana. Artinya kita sudah menyalin seluruh repository dari *remote* ke *local*.

Tambah file baru

Salah satu developer menambahkan file index.php, dengan kode berikut:

```
<html>
<head>
<title>File index</title>
</head>
<body>
<h1>File index</h1>
</body>
</html>
```

Setelah itu ketik perintah git status.

```
$git status
On branch master
Your branch is up-to-date with 'origin/master'.

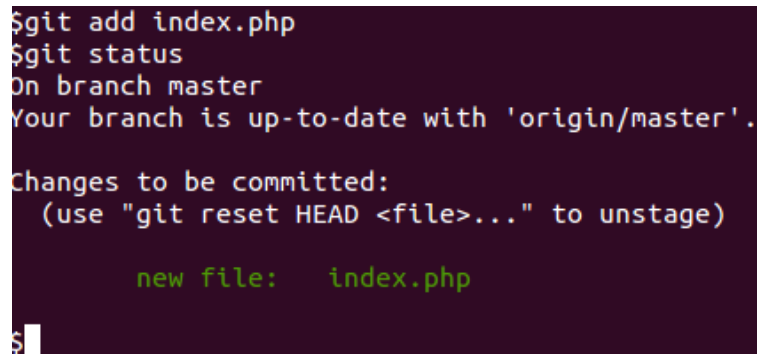
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    index.php

nothing added to commit but untracked files present (use "git add" to track)
$
```

Data terminal di atas menunjukkan bahwa index.php belum di-track oleh git. Untuk menambahkan file index.php ke indeks, jalankan perintah berikut:

```
git add index.php
git status
```

A terminal window with a dark purple background. The text shows the execution of 'git add index.php' and 'git status'. The status output indicates the current branch is 'master' and it is up-to-date with 'origin/master'. It lists 'Changes to be committed' for a new file 'index.php'.

```
$git add index.php
$git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   index.php

$
```

Terlihat bahwa index.php sudah masuk repository. Lakukan commit untuk menyimpan perubahan.

```
git commit index.php -m 'tambahan index.php'
```

Setelah itu push ke server, perintahnya sebagai berikut:

```
git push origin master
```

Developer lain lakukan pull,

```
git pull origin master
```

Setelah itu file index.php akan ditambahkan ke *local repository* masing-masing developer. Silakan cek di direktori masing-masing.

Mengubah File yang Sama

Git dapat menggabungkan kode setiap developer untuk *file* yang sama. Jika baris yang diubah oleh developer satu dengan yang lainnya berbeda, maka git akan menggabungkannya dengan persetujuan terlebih dahulu. Mari kita praktikan.

Harap salah satu developer (Developer A) mengubah index.php menjadi seperti berikut (tambahan kode dalam warna merah):

```
<html>
<head>
<meta name="title" content="File index" />
<title>File index</title>
</head>
<body>
<h1>File index</h1>
</body>
</html>
```

Setelah itu commit dan push ke server.

```
git commit index.php -m 'perubahan index.php developer A'
git push origin master
```

Setelah itu, salah satu developer lain (Developer B) mengubah index.php menjadi seperti berikut:

```
<html>
<head>
<title>File index</title>
</head>
<body>
<h1>File index</h1>
<p>Ini Paragraph baru</p>
</body>
</html>
```

Setelah itu commit dan coba push ke server.

```
git commit index.php -m 'perubahan index.php developer B'
git push origin master
```

Developer B akan mendapatkan notifikasi gagal push seperti pada gambar berikut:

```
$git commit index.php -m 'perubahan index.php developer B'
[master 73dac3b] perubahan index.php developer B
1 file changed, 1 insertion(+)
$git push origin master
To http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
$
```

Developer B tidak dapat melakukan push karena ada *file* yang sama yang diedit oleh developer lain. Untuk dapat melakukan push ke server, developer B harus melakukan pull.

Setelah melakukan pull, git akan menggabungkan (*merge*) *file* index.php yang diedit oleh developer A dan B lalu git akan melakukan commit dan meminta developer B mengubah pesan commit.

```
Merge branch 'master' of http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

Simpan file tersebut dan tutup editor, maka *file* hasil *merge* akan di-*commit*.

```
$git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat
 * branch            master       -> FETCH_HEAD
    910aee0..30891ef  master       -> origin/master
Auto-merging index.php
Merge made by the 'recursive' strategy.
 index.php | 1 +
 1 file changed, 1 insertion(+)
$
```

Jika developer B membuka index.php, maka hasilnya menjadi seperti ini:

```
<html>
<head>
<meta name="title" content="File index" />
<title>File index</title>
</head>
<body>
<h1>File index</h1>
<p>Ini Paragraph baru</p>
</body>
</html>
```

Terlihat bahwa git menggabungkan perubahan kode developer A dan developer B. Developer B lakukan commit dan push, selanjutnya developer A dan developer lain melakukan pull. Dengan demikian, setiap developer memiliki isi repository local yang sama.

Mengatasi Conflict

Pada contoh sebelumnya developer A dan developer B melakukan perubahan kode di baris yang berjauhan sehingga git dengan mudah dapat menggabungkannya. Jika developer A dan developer B mengubah kode di baris yang sama, maka akan terjadi *conflict*. Seperti apakah conflict itu? Mari kita praktikkan.

Developer A mengubah *file* index.php menjadi seperti berikut:

```
<html>
<head>
<meta name="title" content="File index" />
<title>File index</title>
</head>
<body>
<h1>File index</h1>
<p>Ini Paragraph baru editan A</p>
</body>
</html>
```

Setelah itu developer A melakukan commit dan push.

Setelah itu Developer B mengubah *file* index.php menjadi seperti berikut:

```
<html>
<head>
<meta name="title" content="File index" />
<title>File index</title>
</head>
<body>
<h1>File index</h1>
<p>Ini Paragraph baru B mengedit</p>
</body>
</html>
```

Setelah itu developer B melakukan commit lalu pull.

```

$git commit index.php -m 'editan B'
[master 98ac85f] editan B
 1 file changed, 1 insertion(+), 1 deletion(-)
$git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat
 * branch          master      -> FETCH_HEAD
    d925da9..5ea7d4f master    -> origin/master
Auto-merging index.php
CONFLICT (content): Merge conflict in index.php
Automatic merge failed; fix conflicts and then commit the result.
$

```

Git akan memberitahu ada *conflict*, buka *file* index.php, isinya akan sebagai berikut:

```

<html>
<head>
<meta name="title" content="File index" />
<title>File index</title>
</head>
<body>
<h1>File index</h1>
<<<<<<< HEAD
<p>Ini Paragraph baru editan B</p>
=====
<p>Ini Paragraph baru editan A</p>
>>>>>>> 5ea7d4fd70e11f0271e069378f5e7c34ce8d004e
</body>
</html>

```

Kode yang *conflict* ditandai dengan <<<<<<<HEAD dan diakhiri >>>>> (diakhiri hash). Kode milik B dan milik A dipisah oleh garis '====='. Untuk 'meredakan' *conflict*, developer B harus mengubah kode agar kedua kode masuk. Misal ubah jadi seperti berikut:

```

<html>
<head>
<meta name="title" content="File index" />
<title>File index</title>
</head>
<body>
<h1>File index</h1>
<p>Ini Paragraph baru editan B dan editan A</p>
</body>
</html>

```

Setelah itu simpan file, lalu jalankan perintah `git status`.

```
$git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commit each, respectively.
(use "git pull" to merge the remote branch into yours)

You have unmerged paths.
(fix conflicts and run "git commit")

Unmerged paths:
(use "git add <file>..." to mark resolution)

        both modified:   index.php

no changes added to commit (use "git add" and/or "git commit -a")
$
```

Jalankan perintah add dan commit:

```
git add index.php
git commit
```

Karena kita tidak menyertakan pesan di commit, maka akan muncul editor teks dengan isi pesan seperti gambar berikut:

```
Merge branch 'master' of http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat

Conflicts:
    index.php
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#   .git/MERGE_HEAD
# and try again.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch and 'origin/master' have diverged,
# and have 1 and 1 different commit each, respectively.
# (use "git pull" to merge the remote branch into yours)
#
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#   modified:   index.php
#
```

Simpan file tersebut dan tutup editor, maka *conflict* telah diredakan. Push ke server, kemudian developer lain lakukan pull, maka tidak ada *conflict* lagi, index.php di semua developer akan sama.

Menggunakan Branch

Repository dapat memiliki beberapa *branch*. Secara *default*, *branch* yang kita gunakan adalah *branch master*. Setiap *branch* memiliki data *commit* masing-masing. Developer dapat menyunting kode di *branch* lain kemudian jika telah selesai bisa langsung menggabungkan (*merge*) *branch*-nya dengan *branch master*.

Apa sih kegunaan branch?

Biasanya *branch* digunakan ketika kita ingin menambah suatu fitur atau memperbaiki *bug*. Contohnya ketika kita ingin menambah fitur otentikasi, maka kita dapat membuat *branch otentikasi* dan jika sudah selesai langsung gabungkan dengan *branch master*.

Kenapa harus branch baru? Kan bisa edit di master aja?

Ya bisa sih, tapi kalau langsung edit di **master**, *history commit*-nya kecampur-campur dengan yang lain. Jika setiap fitur dibuat di *branch* masing-masing, pengembangan dan perbaikan fitur akan lebih mudah karena developer dapat melihat *history commit* fitur tersebut dari awal sampai ketika digabungkan dengan master.

Mari kita mulai praktik dengan *branch*. Misal kita akan menambah footer.php, rencananya kita akan buat *file* footer.php untuk di-*include* ke index.php. Kali ini dilakukan oleh satu developer saja, yang lain menyimak.

Buat *branch* baru bernama **footer**, perintahnya sebagai berikut:

```
git branch footer
```

Perintah tersebut akan membuat *branch* baru bernama **footer**. Namun saat ini *branch* kita masih di *branch master*. Untuk berpindah ke *branch footer*, jalankan perintah berikut:

```
git checkout footer
```

```
adplus@GREEN-HORNET:~/public_html/appshebat$ git checkout footer
Switched to branch 'footer'
adplus@GREEN-HORNET:~/public_html/appshebat$
```

Sekarang kita sudah ada di *branch footer*. Buat file baru footer.php, isinya sebagai berikut:

```
<hr/>
Ini Footer
```

Setelah itu tambahkan kode ke index.php seperti berikut:

```
<html>
<head>
<meta name="title" content="File index" />
<title>File index</title>
</head>
<body>
<h1>File index</h1>
<p>Ini Paragraph baru editan B dan editan A</p>
<?php require 'footer.php'; ?>
</body>
</html>
```

Setelah itu jalankan perintah add footer.php, commit all dan push sebagai berikut:

```
git add footer.php
git commit -a -m 'Penambahan footer'
git push origin footer
```

```
adplus@GREEN-HORNET:~/public_html/appshebat$ git add footer.php
adplus@GREEN-HORNET:~/public_html/appshebat$ git commit -a -m 'Penambahan footer'
[footer 2a0f39f] Penambahan footer
 2 files changed, 3 insertions(+)
 create mode 100644 footer.php
adplus@GREEN-HORNET:~/public_html/appshebat$ git push origin footer
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 416 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat.git
 * [new branch]      footer -> footer
adplus@GREEN-HORNET:~/public_html/appshebat$
```

Mari kita lihat hasilnya di repository CS IPB GitLab:

CS IPB GitLab

Project

Files

Commits

Network

Graphs

Milestones

Issues 0

Merge Requests 0

Members

Labels

Wiki

Settings

ariefh08

Arief Hidayatulloh / appshebat

Search in this project

master appshebat / +

Download zip

Name	Last Update	Last Commit	History
README.md	3 days ago	ariefsam Penambahan README	
index.php	about 15 hours ago	ariefsam Merge branch 'master' of http://apps.cs.ipb.ac.id:2000/ariefh08/app...	

README.md

Inisialisasi Projek Apps Hebat

CS IPB GitLab

Project

Files

Commits

Network

Graphs

Milestones

Issues 0

Merge Requests 0

Members

Labels

Wiki

Settings

ariefh08

Arief Hidayatulloh / appshebat

Search in this project

footer appshebat / +

Download zip

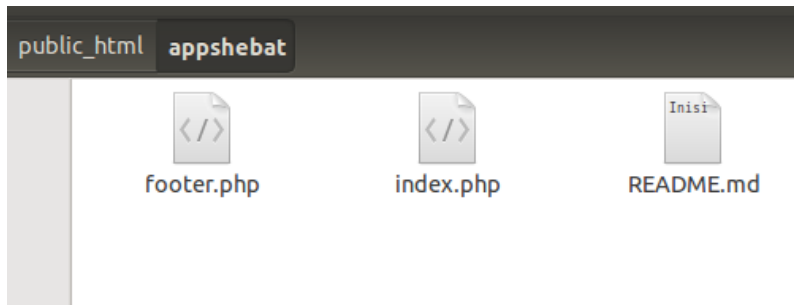
Name	Last Update	Last Commit	History
README.md	3 days ago	ariefsam Penambahan README	
footer.php	5 minutes ago	ariefsam Penambahan footer	
index.php	5 minutes ago	ariefsam Penambahan footer	

README.md

Inisialisasi Projek Apps Hebat

Google Analytics

Terlihat *branch* master dan *branch* footer memiliki jumlah *file* yang berbeda. BTW karena kita sedang berada di *branch* **footer**, susunan file di komputer kita kira-kira seperti gambar berikut:

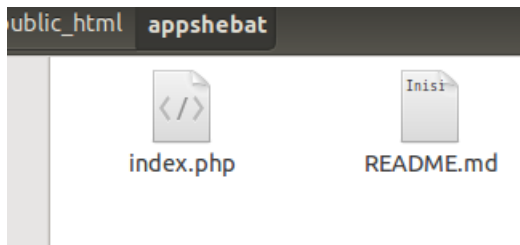


Iseng-iseng coba kita checkout ke master:

```
git checkout master
```

```
adplus@GREEN-HORNET:~/public_html/appshebat$ git checkout master
Switched to branch 'master'
adplus@GREEN-HORNET:~/public_html/appshebat$
```

Karena kita pindah ke branch master, kalau kita lihat di *file explorer*, maka *file* kembali ke susunan lama, *file* footer.php tidak ada.



Merge Branch

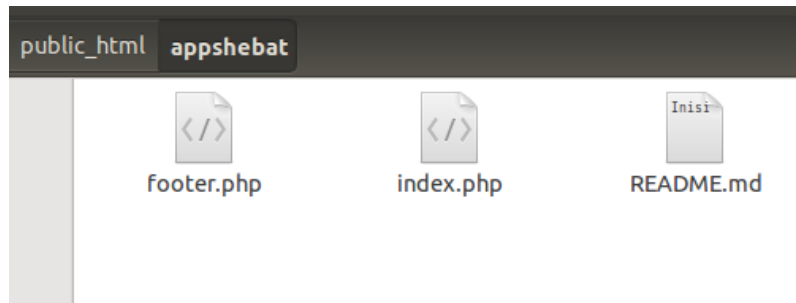
Untuk menggabungkan *branch* **footer** ke *branch* **master**, kita harus checkout dulu ke *branch* **master**. Jika sudah checkout ke *branch* **master**, kita tinggal lakukan penggabungan.

Perintahnya sebagai berikut:

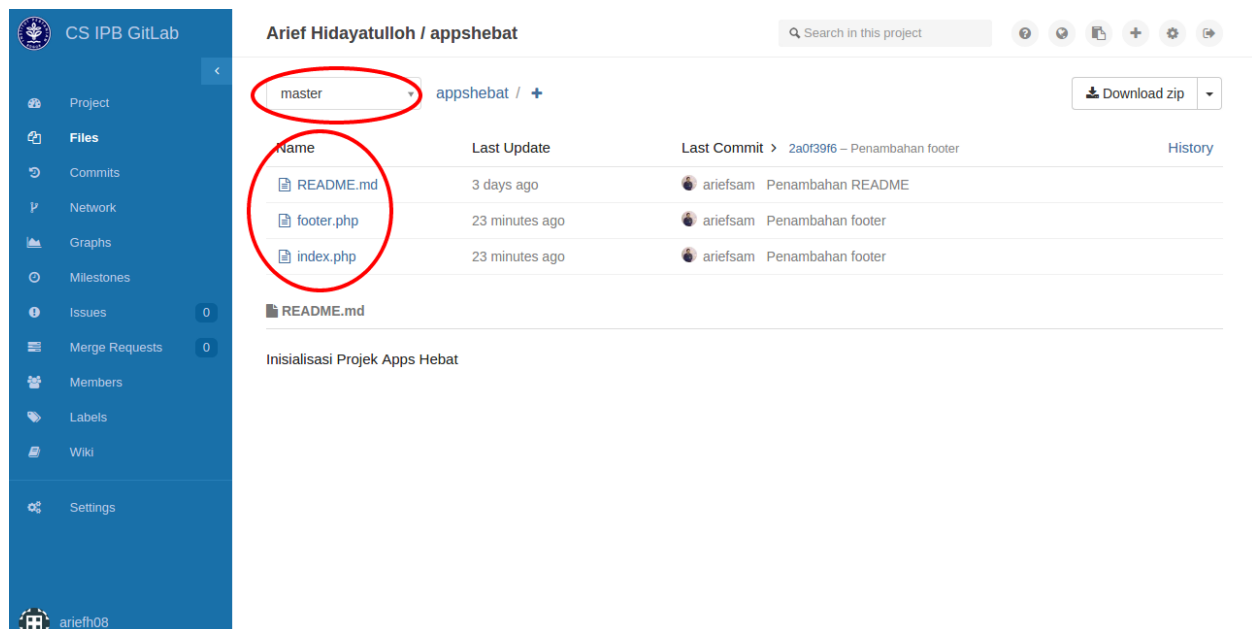
```
git merge footer
```

```
adplus@GREEN-HORNET:~/public_html/appshebat$ git merge footer
Updating 5b21165..2a0f39f
Fast-forward
 footer.php | 2 ++
 index.php  | 1 +
 2 files changed, 3 insertions(+)
 create mode 100644 footer.php
adplus@GREEN-HORNET:~/public_html/appshebat$
```

Setelah itu *file* footer.php akan ditambahkan dan index.php akan diubah sesuai *branch* **footer**.

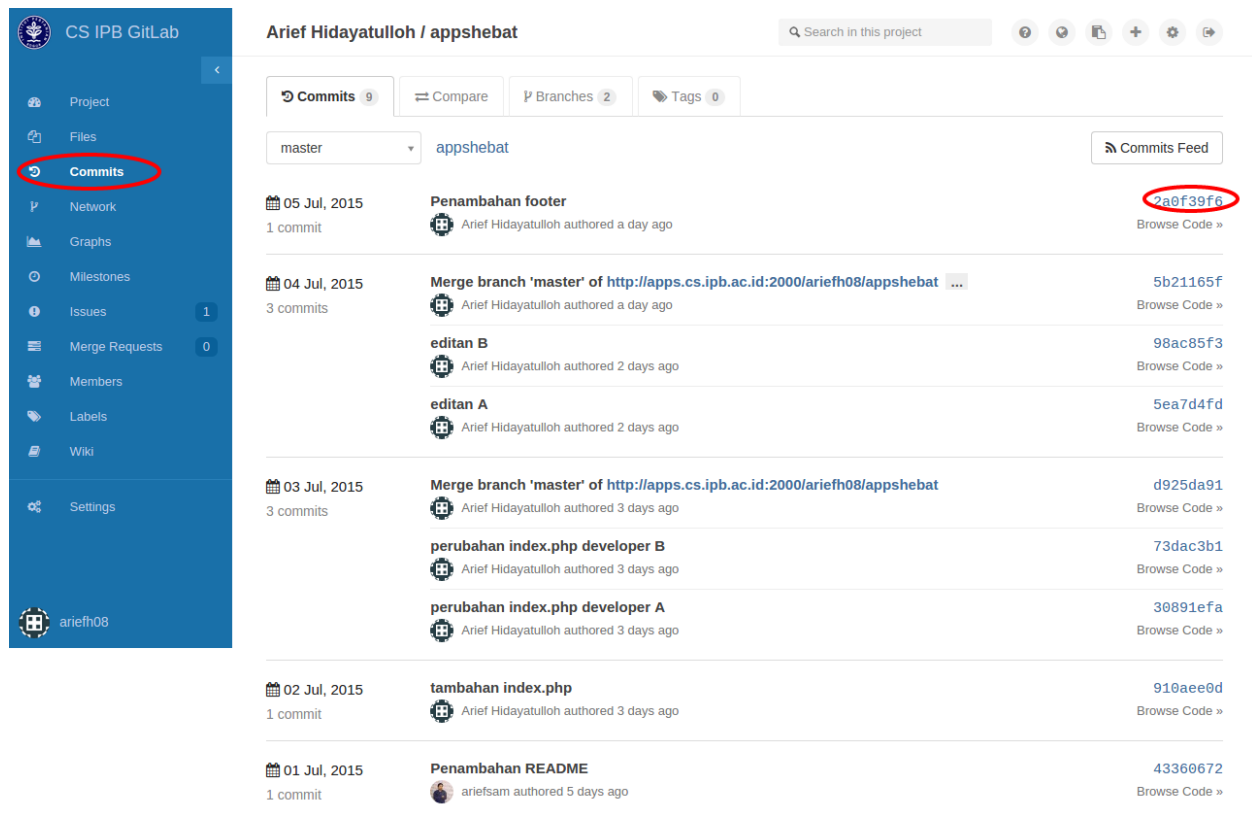


Karena penggabungan baru di *local repository*, kita harus melakukan push ke *remote repository*. Setelah itu *branch* **master** di CS IPB GitLab akan berubah.



Detail Commit

GitLab menyediakan fitur untuk melihat detail dari suatu commit. Pilih menu **commit** di sidebar. Lalu pilih salah satu *hash* commit. Berikut tampilannya:



The screenshot displays the GitLab web interface for the 'appshebat' project. On the left sidebar, the 'Commits' menu item is circled in red. The main area shows a list of commits. The first commit, 'Penambahan footer' by Arief Hidayatulloh, is circled in red and has the hash '2a0f39f6' highlighted. Below it are several merge commits and other commits including 'editan B', 'editan A', 'perubahan index.php developer B', 'perubahan index.php developer A', 'tambahan index.php', and 'Penambahan README'.

Dengan mengklik salah satu commit tersebut, paling tidak ada dua hal yang dapat anda lakukan:

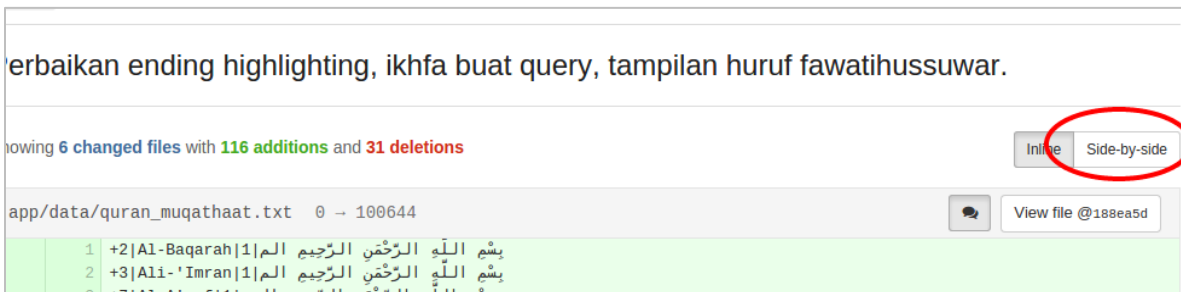
1. Melihat perubahan apa saja yang dilakukan commit ini.
2. Memberi komentar terhadap kode yang di-commit.

Membandingkan *Commit* saat ini dengan *Commit* sebelumnya

GitLab memiliki fitur membandingkan *commit* saat ini dengan *commit* sebelumnya. Fitur ini membuat developer tahu perubahan apa saja yang ada di *commit* saat itu.

```
app/lib/fonetik.php
@@ -554,7 +554,7 @@ function longest_highlight_lookforward($hl_sequence, $min_length = 3) {
554 554     while (isset($hl_sequence[$j]) && $hl_sequence[$j] - $hl_sequence[$j-1] <= $min_length+1 && $j < $len)
555 555     {
556 556         $j++;
557 - $res[] = array($hl_sequence[$i], $hl_sequence[$j-1] + $min_length);
557 + $res[] = array($hl_sequence[$i], $hl_sequence[$j-1]);
558 558         $i = $j-1;
559 559         $j++;
560 560     }
... ..
```

Pada contoh ini, terlihat bahwa line 557 dihapus dan diganti dengan yang baru. Jika kita lihat tampilannya secara **side by side** terlihat jelas perbedaan suatu file di *commit* saat itu dengan *commit* sebelumnya.



Terlihat perbedaan di baris 557. Dengan **side-by-side** developer dengan mudah melihat perbedaan kode antara versi baru dengan versi sebelumnya.

app/lib/fonetik.php		app/lib/fonetik.php	
@@ -554,7 +554,7 @@ function	longest_highlight_lookforward(\$hl_sequence, \$min_length = 3) {	@@ -554,7 +554,7 @@ function	longest_highlight_lookforward(\$hl_sequence, \$min_length = 3) {
554	while (isset(\$hl_sequence[\$j]) && \$hl_sequence[\$j] - \$hl_sequence[\$j-1] <= \$min_length+1 && \$j < \$len) {	554	while (isset(\$hl_sequence[\$j]) && \$hl_sequence[\$j] - \$hl_sequence[\$j-1] <= \$min_length+1 && \$j < \$len) {
555	\$j++;	555	\$j++;
556	}	556	}
557	- \$res[] = array(\$hl_sequence[\$i], \$hl_sequence[\$j-1] + \$min_length);	557	+ \$res[] = array(\$hl_sequence[\$i], \$hl_sequence[\$j-1]);
558	\$i = \$j-1;	558	\$i = \$j-1;
559	\$j++;	559	\$j++;
560	}	560	}

The Comment

GitLab memiliki fitur komentar untuk setiap *commit* di *remote server*. Komentar dapat dilakukan sampai ke tingkat baris. Mari kita coba buka proyek kita sebelumnya di CS IPB GitLab.

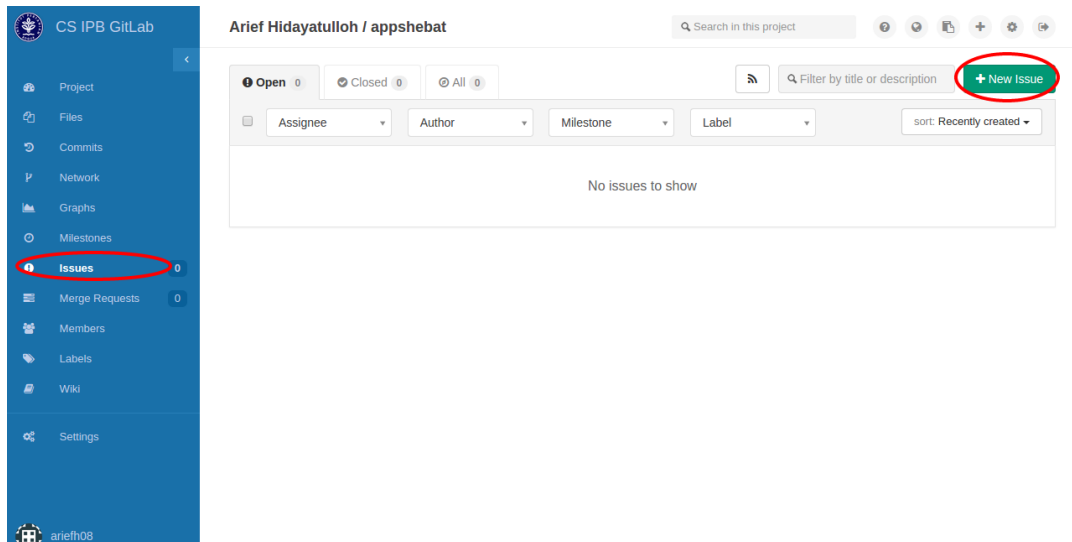
Setelah itu akan muncul *file-file* yang berubah dari *commit* sebelumnya. Kita dapat memberi komentar untuk setiap barisnya, caranya dengan mendekatkan *mouse pointer* ke sebelah kiri

kode lalu klik *icon* komentar. Kita dapat memberikan komentar bahwa kode kurang bagus, kurang efisien, kurang elegan, atau bisa memuji kode itu.

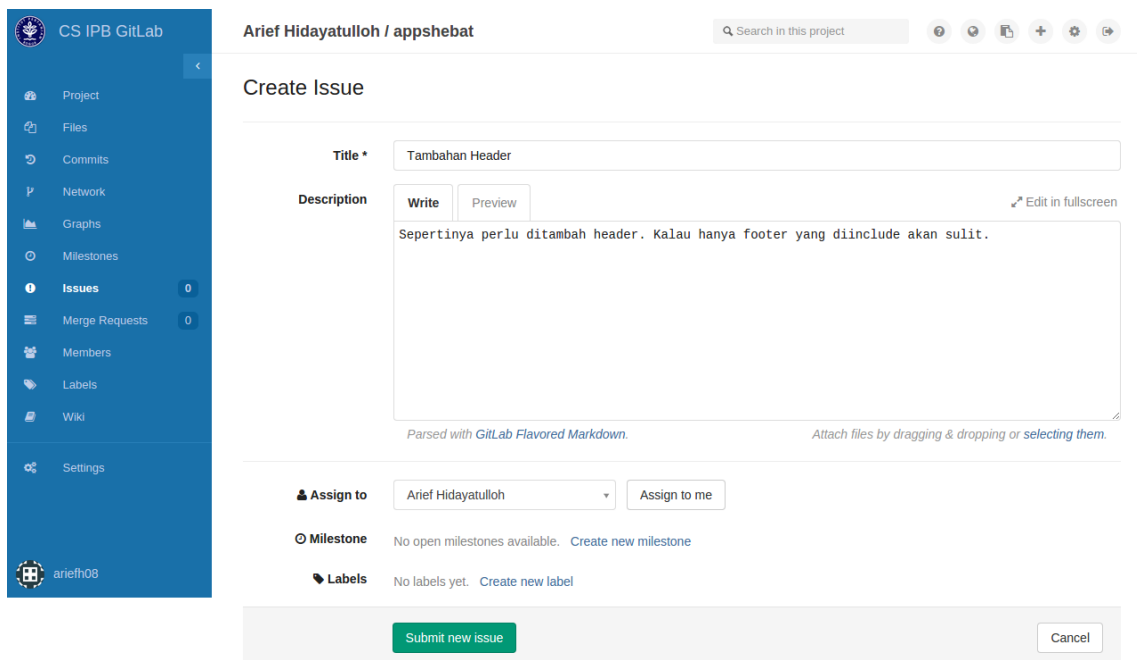
The screenshot shows the GitLab interface for a merge request. The sidebar on the left contains navigation links: Project, Files, Commits, Network, Graphs, Milestones, Issues (1), Merge Requests (0), Members, Labels, Wiki, and Settings. The main area displays the diff for a merge request titled "Arief Hidayatulloh / appshebat". It shows two files: footer.php and index.php. The footer.php diff shows two additions: a closing tag for a character and a line for the footer. The index.php diff shows several changes, including a new paragraph and a PHP require statement. A red circle highlights the comment icon in the sidebar, and a red oval highlights the comment input area in the main view. The comment input area has a "Write" tab and a "Preview" tab. Below the input area are "Add Comment" and "Cancel" buttons.

Issue

Fitur Issue berguna untuk mendaftarkan *bug*, *defect*, atau *request* fitur baru. Issue dapat dibuat oleh member dengan level *guest*. Untuk menambah issue, klik **Issue**, lalu klik **New Issue**

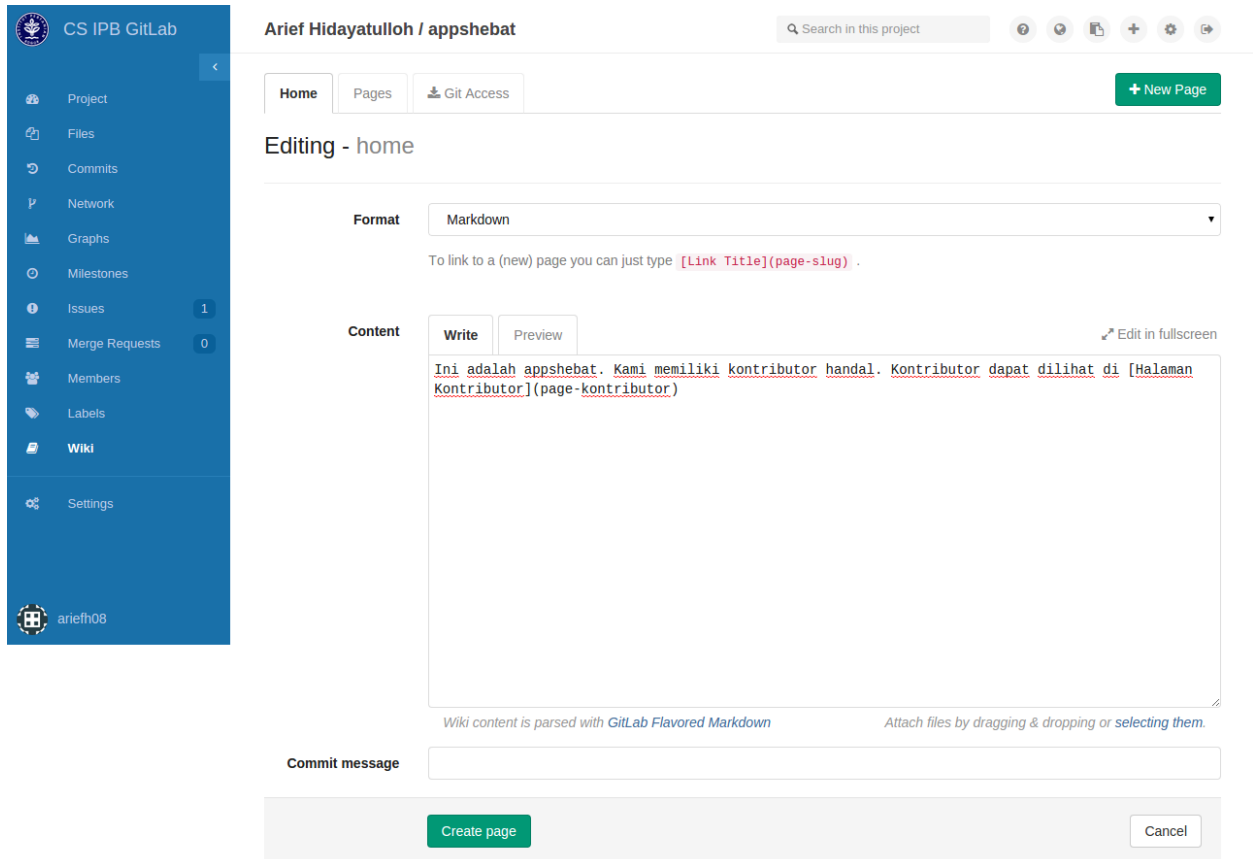


Isi *field* **Title** dan **Description**. Setelah itu pilih member yang diminta mengerjakan issue ini di *field* **Assign to**. Setelah itu klik **Submit new issue**. Setelah itu issue telah ditambahkan ke *repository*. Jika issue telah selesai, developer atau pelapor dapat menutup issue dengan menekan **Close Issue**.



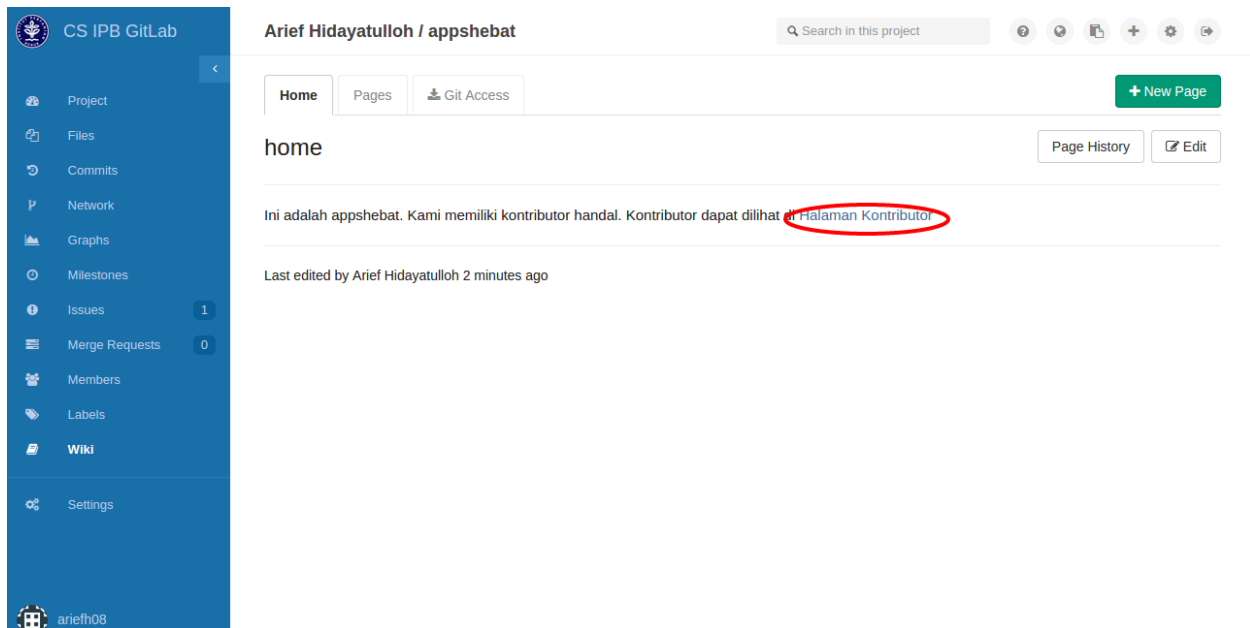
Wiki

Fitur wiki berguna untuk mendokumentasikan aplikasi. Kita bisa membuat panduan aplikasi atau dokumentasi desain dan sebagainya. Untuk membuat wiki, klik menu **Wiki**. Setelah itu kita akan diminta mengedit halaman **Home**. Isi kontennya, jika butuh link ke halaman lain, tambahkan format [Judul Link](slug halaman). Contoh:



The screenshot shows the GitLab Wiki editor interface. On the left is a sidebar with the project name 'CS IPB GitLab' and a list of navigation items: Project, Files, Commits, Network, Graphs, Milestones, Issues (1), Merge Requests (0), Members, Labels, Wiki (selected), and Settings. The main header shows the user 'Arief Hidayatulloh / appshebat' and a search bar. Below the header are tabs for 'Home', 'Pages', and 'Git Access', with a '+ New Page' button. The page title is 'Editing - home'. The 'Format' dropdown is set to 'Markdown'. A tip states: 'To link to a (new) page you can just type [Link Title](page-slug)'. The 'Content' section has 'Write' and 'Preview' tabs, with an 'Edit in fullscreen' link. The content area contains the text: 'Ini adalah appshebat. Kami memiliki kontributor handal. Kontributor dapat dilihat di [Halaman Kontributor](page-kontributor)'. Below the content area, it says 'Wiki content is parsed with GitLab Flavored Markdown' and 'Attach files by dragging & dropping or selecting them.' At the bottom, there is a 'Commit message' field and two buttons: 'Create page' and 'Cancel'.

Isi konten dengan menambah link **page-kontributor**. Setelah itu klik **Create page**. Halaman Home dari wiki akan berbentuk sebagai berikut:



CS IPB GitLab

Arief Hidayatulloh / appshebat

Search in this project

Home Pages Git Access + New Page

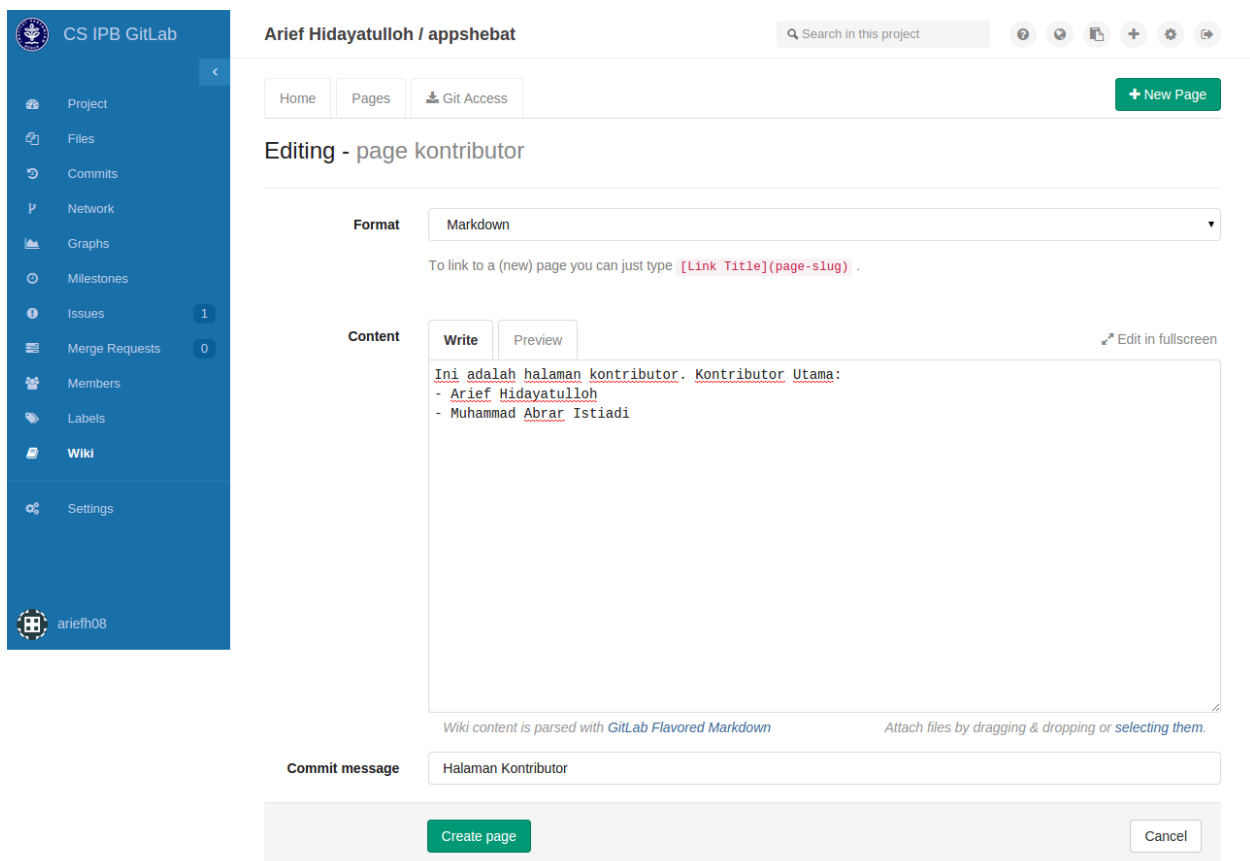
home

Page History Edit

Ini adalah appshebat. Kami memiliki kontributor handal. Kontributor dapat dilihat di **Halaman Kontributor**

Last edited by Arief Hidayatulloh 2 minutes ago

Klik link yang tadi dibuat, karena masih kosong maka kita akan diarahkan ke halaman edit, contoh:



CS IPB GitLab

Arief Hidayatulloh / appshebat

Search in this project

Home Pages Git Access + New Page

Editing - page kontributor

Format Markdown

To link to a (new) page you can just type `[Link Title](page-slug)` .

Content Write Preview Edit in fullscreen

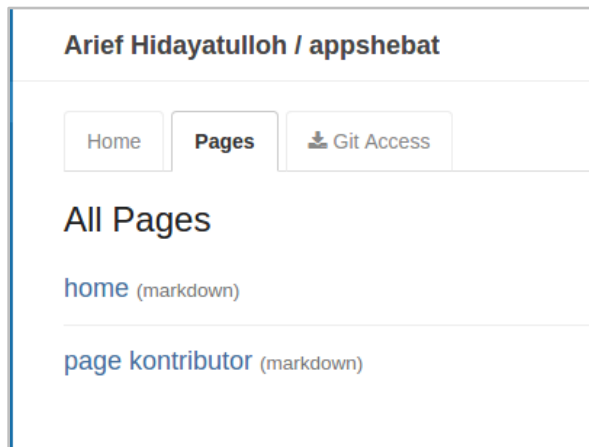
Ini adalah halaman kontributor. Kontributor Utama:
- Arief Hidayatulloh
- Muhammad Abrar Istiadi

Wiki content is parsed with GitLab Flavored Markdown Attach files by dragging & dropping or selecting them.

Commit message Halaman Kontributor

Create page Cancel

Isi dengan konten kontributor, setelah itu tekan **Create page**, maka kita sudah membuat dua halaman, yaitu **home** dan **kontributor**.



Tips dan Trick

Git Ignore

Gunakan file `.gitignore` untuk mengabaikan file atau folder dari repository. Biasanya file yang diabaikan sebagai berikut:

1. File konfigurasi database. File ini sebaiknya dimasukkan ke `.gitignore` karena konfigurasi username dan password tiap orang berbeda.
2. File-file framework. File framework tidak perlu dimasukkan ke repository karena bisa membuat ukuran repo membengkak dan clone lebih lama.
3. File-file project milik IDE. Misalnya jika Anda menggunakan NetBeans, akan ada folder `"nbproject"`. Folder ini sebaiknya dimasukkan ke daftar ignore.
4. File-file composer. Jika kita menggunakan composer sebaiknya hanya `composer.json` yang dishare di repository. `Composer.lock` dan folder `vendor` dimasukkan dalam `.gitignore`. Setiap user harus sering menjalankan composer install atau composer update agar masing-masing developer memiliki dependency yang terbaru.

Menghindari Conflict

Conflict dalam git bisa diatasi, namun sering kali developer yang melakukan penyatuan kode malah membuat kode developer lain rusak atau hilang. Untuk menghindari conflict, lakukan beberapa tips berikut:

1. Setiap developer sebaiknya sering melakukan pull dan push agar perubahan *file* tidak terlalu jauh.
2. Jangan membuat satu *file* yang sering diubah oleh banyak developer. Hindari menyunting *file* yang sama secara bersamaan.
3. Jika harus menyunting *file* yang sama secara bersamaan,

Gunakan Branch

Gunakan *branch* untuk menghindari kerusakan kode dari *merge conflict* yang tidak cermat. jika terjadi salah *merge conflict*, developer dapat dengan mudah mengambil kode lama dari branch miliknya. Selain itu *branch* memudahkan developer untuk *me-maintenance* kodenya karena *history commit*-nya lebih mudah dibaca.

Melihat History

Untuk melihat *history commit*, gunakan perintah berikut:

```
git log
```

```

commit 2a0f39f683dcf98831d0c35ef89c00b18f342fb8
Author: Arief Hidayatulloh <ariefhidayatulloh@gmail.com>
Date:   Sun Jul 5 06:58:08 2015 +0700

    Penambahan footer

commit 5b21165f58efe50221332a4316e455feb47c9e3b
Merge: 98ac85f 5ea7d4f
Author: Arief Hidayatulloh <ariefhidayatulloh@gmail.com>
Date:   Sat Jul 4 15:58:02 2015 +0700

    Merge branch 'master' of http://apps.cs.ipb.ac.id:2000/ariefh08/appshebat

Conflicts:
    index.php

commit 98ac85f3cb568b4bbcb47871ea20830bebc4b5c5
Author: Arief Hidayatulloh <ariefhidayatulloh@gmail.com>
Date:   Sat Jul 4 15:15:03 2015 +0700

    editan B

commit 5ea7d4fd70e11f0271e069378f5e7c34ce8d004e
Author: Arief Hidayatulloh <ariefhidayatulloh@gmail.com>
Date:   Sat Jul 4 15:14:24 2015 +0700

    editan A

commit d925da910a1e23ddb5f5240a4bed89ef4144bfd3
:
```

Kebanyakan orang yang baru mengenal `git log` akan kesulitan keluar dari programnya. Banyak yang mengambil langkah singkat `Ctrl+Z` (di linux) yang artinya memaksa program berhenti. Padahal cara yang benar cukup menekan huruf **Q** di keyboard.

Mengembalikan Repository ke Commit Sebelumnya

Sudah menjadi kewajiban developer untuk cek terlebih dahulu programnya sebelum melakukan *commit*. Namun sering kali developer melakukan *commit* padahal kodenya menyebabkan *error*.

Contoh kasus seperti ini:

Andri hendak presentasi programnya kepada klien. Ketika demo software terjadi *error*, padahal kemarin masih jalan. Andri ingat bahwa pada *commit* sebelumnya programnya tidak apa-apa. Rupanya tadi pagi ada rekannya yang mengubah kode lalu melakukan *commit* tanpa test terlebih dahulu. Solusinya Andri melakukan *revert* dari *commit* sebelumnya. Hasilnya: programnya kembali bekerja.

Memang salah satu cara mengembalikan *commit* adalah menjalankan *revert*. Perintah *revert* akan membuat suatu *commit* baru dimana *commit* baru ini akan sama dengan *commit* sebelum yang sekarang.

Perintah yang Andri kerjakan hanya 1 baris yaitu:

```
git revert HEAD
```

Penutup

Demikian sedikit tutorial pengenalan git. Karena masih pengenalan, harap rekan-rekan menggali lebih jauh di dunia *cyber* yang lebih 'dewa'. Kalau bisa mulailah menggunakan git untuk proyek kuliah maupun proyek luar, dijadikan open source juga lebih baik.

Catatan: Bagi yang sudah memiliki skill 'dewa', mohon bantu teman-temannya yang masih 'newbie' dan mohon koreksi naskah ini jika ada kesalahan. :)

Mohon maaf atas segala kesalahan saya sebagai penulis. Silakan kontak saya jika ingin menanyakan sesuatu. Terima kasih dan sampai jumpa!

* * *

Jakarta-Bogor, Juli 2015

Penulis:

Arief Hidayatulloh

Ilkomerz 45

ariefhidayatulloh@gmail.com

Editor:

Muhammad Abrar Istiadi

abrari@apps.ipb.ac.id

Referensi

<https://git-scm.com/book/id>