

COURSE 3

Memahami Git (Umum)

Material Details

01 Understanding Git & Flow Development

Modul Sekolah Fullstack Cilsy

Hak Cipta © 2020 PT. Cilsy Fiolution Indonesia

Hak Cipta dilindungi Undang-Undang. Dilarang memperbanyak atau memindahkan sebagian atau seluruh isi buku ini dalam bentuk apapun, baik secara elektronis maupun mekanis, termasuk mecopy, merekam atau dengan sistem penyimpanan lainnya, tanpa izin tertulis dari Penulis dan Penerbit.

Penulis : Muhammad Lukman Hakim, Nenden Herlina, Muhammad Fakhri Abdillah
Editor : Muhammad Fakhri Abdillah, Iqbal Ilman Firdaus

Penerbit : **PT. Cilsy Fiolution Indonesia**
Web Site : <https://cilsyfiolution.com> , <https://sekolahfullstack.com>

Sanksi Pelanggaran Pasal 113 Undang-undang Nomor 28 Tahun 2014 tentang Hak Cipta

1. Setiap orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam pasal 9 ayat (1) huruf i untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan atau pidana denda paling banyak Rp100.000.000,00 (seratus juta rupiah).
2. Setiap orang yang dengan tanpa hak dan atau tanpa izin pencipta atau pemegang hak cipta melakukan pelanggaran hak ekonomi pencipta sebagaimana dimaksud dalam pasal 9 ayat (1) huruf c, huruf d, huruf f, dan atau huruf h, untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah)
3. Setiap orang yang dengan tanpa hak dan atau tanpa izin pencipta atau pemegang hak melakukan pelanggaran hak ekonomi pencipta sebagaimana dimaksud dalam pasal 9 ayat (1) huruf a, huruf b, huruf e, dan atau huruf g, untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah)
4. Setiap orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah)

Daftar Isi

Daftar Isi.....	3
5. Learn Agile & Git.....	6
Learning Outcomes.....	6
Outline Materi.....	6
5.1. Understanding How To Implement Agile & Scrum.....	7
5.1.1. Pengenalan Agile.....	7
5.1.2. Pengenalan Scrum.....	8
5.2. Why Agile & Scrum?.....	8
5.2.1. Tim Scrum.....	9
5.2.2. Kanban.....	9
5.3. Simulasi-Simulasi Scrum.....	11
5.3.1. Sprint Planning.....	12
5.3.2. Daily Scrum.....	13
5.3.3. Sprint Review.....	14
5.3.4. Sprint Retrospective.....	15
5.4. Git.....	16
5.5. Instalasi dan Konfigurasi Git.....	18
5.5.1. Instalasi Git di Linux.....	18
5.5.2. Instalasi Git di Windows.....	18
5.5.3. Konfigurasi Git.....	23
5.5.4. Persiapan File Untuk Git.....	24
5.6. Membuat Repositori dan Revisi.....	25
5.6.1. Membuat Repositori.....	25
5.6.1.1. Gitignore.....	26
5.6.2. Revisi.....	26
5.6.2.1. Melihat Log Revisi.....	31
5.6.2.2. Log pada Nomor Revisi/Commit.....	32

5.6.2.3. Log pada File Tertentu.....	32
5.6.2.4. Melihat Perbandingan Perubahan yang Dilakukan pada Revisi.....	33
5.6.2.5. Melihat Perbandingan pada File.....	35
5.6.2.6. Melihat Perbandingan antar Revisi/Commit.....	35
5.6.2.7. Perbandingan Antar Cabang (Branch).....	35
5.6.3. Exercise.....	36
5.7. Branching.....	36
5.7.1. Membuat branch Baru.....	36
5.7.2. Menggabungkan branch.....	38
5.7.3. Mengatasi Adanya Bentrok.....	39
5.7.4. Menghapus branch.....	41
5.7.5. Exercise.....	41
5.8. Remote Repository.....	41
5.8.1. Membuat Repositori di GitHub.....	42
5.8.2. Menambah Remote Repository.....	44
5.8.3. Menggunakan SSH di GitHub.....	44
5.8.3.1. Membuat SSH Key.....	44
5.8.3.2. Jalankan SSH Agent dan Load SSH Key.....	45
5.8.3.3. Uji Konektivitas.....	46
5.8.3.4. Mengubah dan Menghapus Remote Repository.....	46
5.8.4. Mengirim Revisi ke Remote Repository.....	47
5.8.5. Mengambil Revisi dan Remote Repository.....	50
5.8.6. Mengambil Revisi dengan Git Fetch.....	51
5.8.7. Mengambil Revisi dengan Git Pull.....	54
5.8.8. Clone Remote Repository.....	55
5.8.9. Exercise.....	56
5.9. Studi Kasus.....	56
5.9.1. Study Case Git 1.....	56
5.9.2. Study Case Git 2.....	58
5.9.3. Study Case Git 3.....	59

5.9.4. Study Case Git 4.....	61
5.9.5. Study Case Git 5.....	61
5.9.6. Study Case Git 6.....	62
5.10. Flow Development Di Dalam Sebuah Tim.....	62
5.10.1. Mengenal Local – Staging – Production.....	63
5.10.2. Tugas.....	63
5.11. Mengenal Strategi Deployment Aplikasi.....	63
5.11.1. Big-Bang Deployment Strategy.....	64
5.11.1.1. Kelebihan:.....	65
5.11.1.2. Kekurangan.....	65
5.11.2. Rollout Deployment Strategy.....	65
5.11.2.1. Kelebihan:.....	66
5.11.2.2. Kekurangan.....	66
5.11.3. Blue/Green Deployment Strategy.....	66
5.11.3.1. Kelebihan:.....	67
5.11.3.2. Kekurangan:.....	67
5.11.4. A/B Deployment Strategy.....	67
5.11.4.1. Kelebihan:.....	68
5.11.4.2. Kekurangan:.....	68
5.11.5. Canary Deployment Strategy.....	69
5.11.5.1. Kelebihan:.....	71
5.11.5.2. Kekurangan:.....	71

5.

Learn Agile & Git

Learning Outcomes

Setelah selesai mempelajari bab ini, peserta mampu :

1. Dapat Mengimplementasikan Agile & Scrum
2. Dapat mengenal dan menggunakan Git

Outline Materi

1. Memahami Agile & Scrum
2. Kenapa Agile & Scrum
3. Simulasi Scrum
4. Git
5. Instalasi dan Konfigurasi Git
6. Membuat Repositori dan Revisi
7. Branching
8. Remote Repository
9. Flow Development di dalam Sebuah Tim
10. Mengetahui Strategi Deployment Aplikasi

5.1. Understanding How To Implement Agile & Scrum

Pada bagian ini instruktur lebih banyak sharing terkait penerapan agile ataupun scrum yang ada di perusahaan masing-masing. Diambil sudut pandang sebagai member dari scrum yaitu sebagai developer. Modul ini hanya untuk pengetahuan umum para siswa terkait Agile dan Scrum.

5.1.1. Pengenalan Agile



Agile merupakan istilah umum untuk berbagai pendekatan terkait pengembangan perangkat lunak. Didalamnya terdapat nilai dan prinsip serta pola pikir. Saat ini pola pendekatan tersebut yang paling populer di dunia adalah **Scrum**.

Sejarah dari agile manifesto dibuat oleh 17 orang di sebuah tempat di snowbird ski resort di pegunungan wasatch utah, USA. Pada Februari, tanggal 11-13 tahun 2001.

“Kami menemukan cara yang lebih baik untuk pengembangan perangkat lunak dengan melakukannya dan membantu orang untuk melakukannya. Melalui cara ini, kami menghargai : “

Setiap individu dan Interaksi lebih dari sebuah proses dan alat

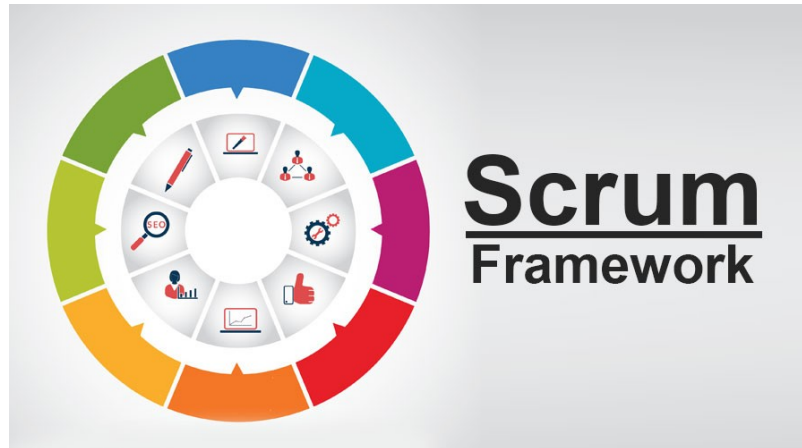
Bekerjanya perangkat lunak lebih baik dari dokumentasi yang konprehensif

Kolaborasi dengan pelanggan lebih dari sekedar negosiasi kontrak

Menanggapi perubahan lebih baik dari mengikuti rencana

“Artinya, Point yang kita **bold** di sebelah kiri lebih baik dibanding sebelah kanan ”

5.1.2. Pengenalan Scrum



Sebuah kerangka kerja di mana orang-orang dapat menyelesaikan permasalahan kompleks yang senantiasa berubah, di mana pada saat bersamaan menghasilkan produk dengan nilai setinggi mungkin secara kreatif dan produktif.

Scrum bersifat:

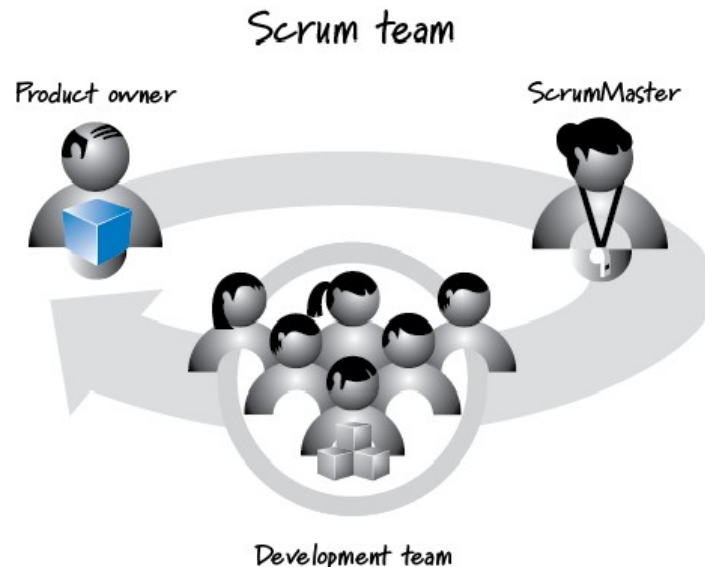
- Ringan
- Mudah dipahami
- Sulit dikuasai

Scrum adalah kerangka kerja proses yang telah digunakan untuk mengelola pengembangan produk kompleks semenjak awal tahun 1990-an.

5.2. Why Agile & Scrum?

Scrum bukanlah sebuah proses ataupun teknik untuk mengembangkan produk. Daripada itu, ini adalah sebuah kerangka kerja di mana di dalamnya anda dapat memasukkan beragam proses dan teknik. Scrum akan mengekspos pergerakan efektifitas manajemen produk dan praktik pengembangan yang sedang anda jalani, dengan begitu anda dapat melakukan peningkatan.

5.2.1. Tim Scrum



Tim Scrum terdiri dari Product Owner, Tim Pengembang (Developer) dan Scrum Master. Tim Scrum mengatur diri mereka sendiri dan berfungsi antar-lintas. Tim yang mengatur dirinya sendiri menentukan cara terbaik untuk menyelesaikan pekerjaannya, daripada diatur oleh pihak lain. yang berada di luar anggota tim. Tim yang berfungsi antar-lintas memiliki semua kompetensi yang dibutuhkan untuk menyelesaikan pekerjaan, tanpa mengandalkan pihak lain yang berada di luar anggota tim. Model tim di dalam Scrum dirancang sedemikian rupa untuk mengotimalisasi fleksibilitas, kreatifitas dan produktifitas.

Tim Scrum menghantarkan produk secara berkala dan bertahap untuk memperbesar kesempatan mendapatkan masukan. Penghantaran secara bertahap dari sebuah produk yang “Selesai”, memastikan produk yang berpotensi dapat digunakan, selalu siap.

5.2.2. Kanban

Kanban merupakan istilah dalam bahasa Jepang yang artinya adalah Papan Visual yang pada awalnya dikembangkan sebagai metode untuk memberikan sinyal pada persediaan bahan-bahan produksi di sistem Inventory Just in Time (Sistem Persediaan yang Tepat Waktu). Dengan menerapkan Metode Kanban ini, bahan-bahan produksi dipasok dan tiba pada

waktunya sesuai dengan jumlah yang dibutuhkan sehingga dapat mengurangi biaya penanganan dan penyimpanan.

Istilah Kanban berasal dari dua kata bahasa Jepang yaitu “Kan” [看] yang artinya adalah “Melihat” atau “Visual” sedangkan kata “Ban” [板] jika diterjemahkan langsung ke bahasa Indonesia menjadi “Papan” atau “Kartu”. Jadi istilah Kanban dapat diterjemahkan menjadi “Papan Visual” atau “Kartu Visual”.



Kanban pada dasarnya adalah suatu metode manajemen untuk memvisualisasikan komunikasi dan pengendalian serangkaian aliran aktivitas di produksi sehingga memungkinkan semua orang untuk melihat aliran aktivitas tersebut dan menyesuaikannya sesuai dengan kebutuhan. Metode Kanban ini umumnya diterapkan di perusahaan-perusahaan yang bergerak di bidang Manufakturing, namun saat ini juga banyak perusahaan-perusahaan yang non-manufakturing yang menerapkan metode Kanban untuk mengendalikan aliran kerja agar lebih efektif dan efisien.

3 langkah penting yang ada di Kanban :

1. Visualisasi Alur Kerja

Runtutan kegiatan kerja yang terpantau menggunakan cara yang paling sederhana yaitu dengan menempelkan kertas Post it ataupun Kartu Informasi yang ditempelkan di Papan ataupun dengan menggunakan Software khusus untuk Kanban

2. Membatasi WIP (Work In-Progress)

Menetapkan dengan tegas batas pekerjaan/tugas ataupun jumlah unit produk yang masih dalam proses (Work In-Progress). Pekerjaan atau tugas yang telah disusun dalam Alur Kerja harus diselesaikan sesuai dengan jangka waktu yang ditentukan sehingga pekerjaan yang tertunda ataupun “masih dalam proses pengerjaan” dapat dikendalikan seminimal mungkin.

3. Mengukur Lead Time yang diperlukan

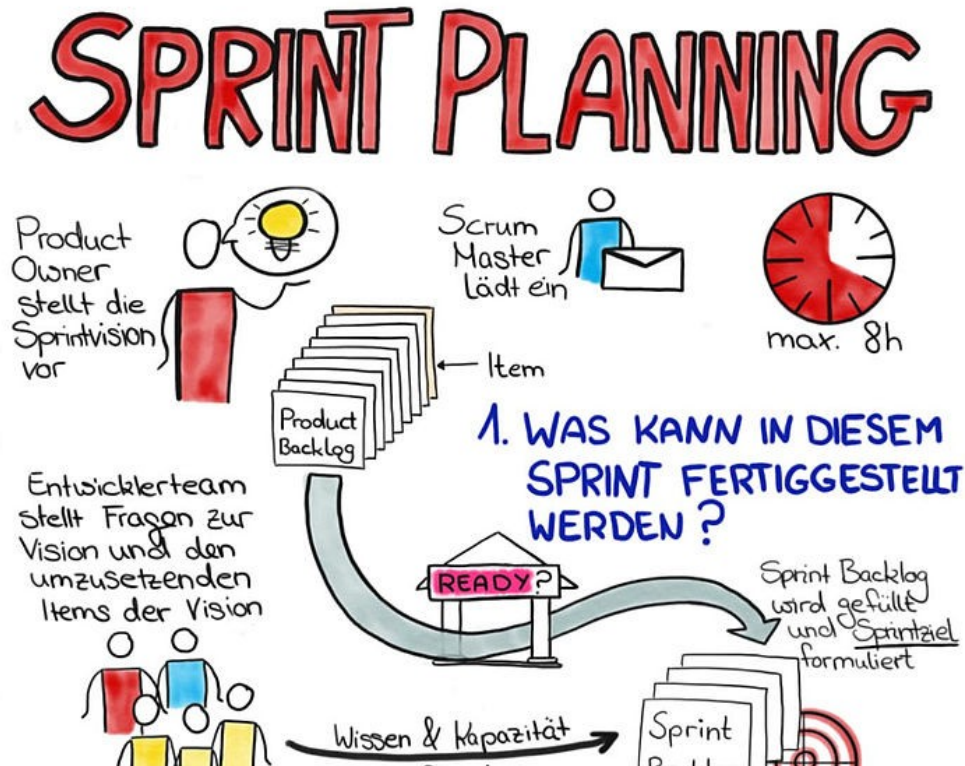
Lead Time adalah siklus waktu yang dibutuhkan untuk menyelesaikan satu unit produk (atau suatu pekerjaan/tugas) dari awal hingga menjadi produk jadi (selesai). Pada Kanban ini, diperlukan pengukuran dan penentuan Lead Time untuk mengoptimalkan proses pekerjaan sehingga dapat diprediksi seakurat mungkin dan waktu yang diperlukan pun sesingkat mungkin.

5.3. Simulasi-Simulasi Scrum

Acara-acara wajib dalam Scrum dihadiri untuk menciptakan sebuah kesinambungan dan mengurangi adanya acara-acara lain yang tidak tercantum di dalam Scrum. Setiap acara di dalam Scrum memiliki batasan waktu, yang artinya selalu memiliki durasi maksimum. Pada saat Sprint dimulai, durasinya tetap dan tidak dapat diperpendek maupun diperpanjang. Acara-acara lainnya dapat diakhiri saat tujuan dari acara tersebut telah tercapai. memastikan waktu digunakan secukupnya tanpa ada yang terbuang sia-sia di sepanjang proses

Event yang di jalankan Scrum :

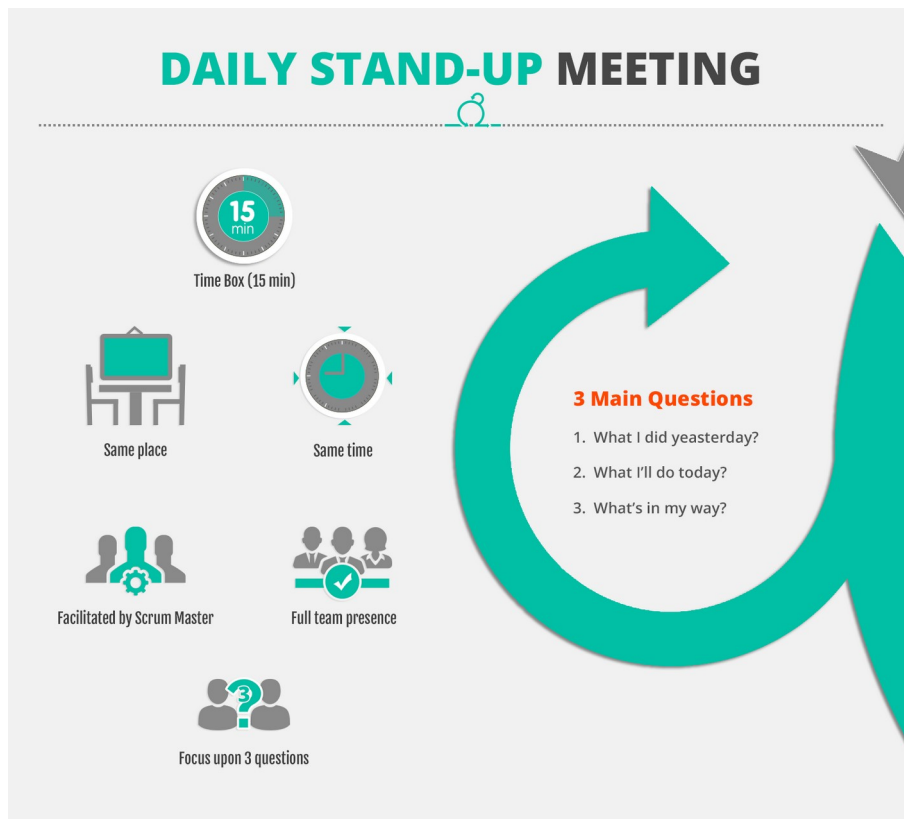
5.3.1. Sprint Planning



Pekerjaan yang akan dilaksanakan di dalam Sprint direncanakan pada saat Sprint Planning. Perencanaan ini dibuat secara kolaboratif oleh seluruh anggota Tim Scrum. Sprint Planning dibatasi maksimum delapan jam untuk Sprint yang berdurasi satu bulan. Untuk Sprint yang lebih pendek, batasan waktunya biasanya lebih singkat. Scrum Master memastikan bahwa acara ini dilaksanakan dan setiap hadirin memahami tujuannya. Scrum Master mengedukasi Tim Scrum untuk melaksanakannya dalam batasan waktu yang telah ditentukan. Sprint Planning harus dapat menjawab pertanyaan-pertanyaan berikut:

- Apa goal dari Sprint?
- Apa yang dapat dihantarkan di dalam Inkremen sebagai hasil dari Sprint yang sedang berjalan?
- Apa yang perlu dilakukan untuk dapat menghantarkan Inkremen tersebut?

5.3.2. Daily Scrum



Daily Scrum adalah kegiatan dengan batasan waktu maksimum selama 15 menit agar Tim Pengembang dapat mensinkronisasikan pekerjaan mereka dan membuat perencanaan untuk 24 jam ke depan. Hal ini dilakukan dengan meninjau pekerjaan semenjak acara Daily Scrum terakhir dan memperkirakan pekerjaan yang dapat dilakukan sebelum melakukan Daily Scrum berikutnya. Daily Scrum dilaksanakan pada waktu dan tempat yang sama setiap hari untuk mengurangi kompleksitas. Pada saat pertemuan, Tim Pengembang menjelaskan:

- Apa yang sudah saya lakukan kemarin yang telah membantu Tim Pengembang mencapai Sprint Goal?
- Apa yang akan saya lakukan hari ini untuk membantu Tim Pengembang mencapai Sprint Goal?
- Apakah ada hambatan yang dapat menghalangi saya atau Tim Pengembang untuk mencapai Sprint Goal

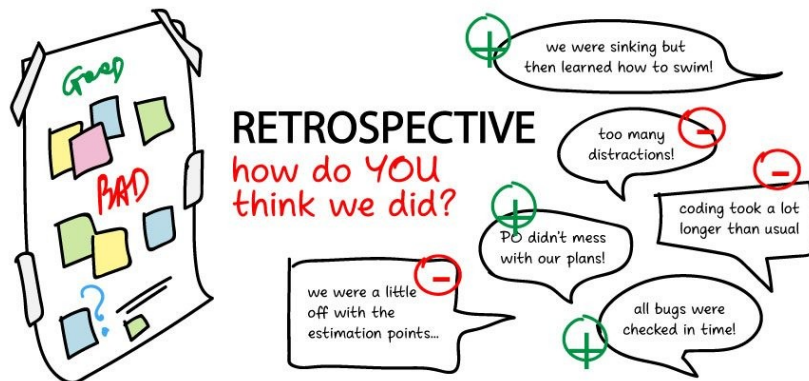
5.3.3. Sprint Review



Sprint Review diadakan di akhir Sprint untuk meninjau Inkremen dan merubah Product Backlog bila diperlukan. Pada saat Sprint Review, Tim Scrum dan stakeholder berkolaborasi untuk membahas apa yang telah dikerjakan dalam Sprint yang baru usai. Berdasarkan hasil tersebut tersebut dan semua perubahan Product Backlog pada saat Sprint, para hadirin berkolaborasi menentukan apa yang dapat dikerjakan di Sprint berikutnya, untuk mengoptimalisasi nilai produk. Pertemuan ini bersifat informal, bukan merupakan status meeting, dan presentasi dari Inkremen diharapkan dapat mengumpulkan masukan dan menumbuhkan semangat kolaborasi.

Hasil dari Sprint Review adalah revisi dari Product Backlog yang mendefinisikan kemungkinan item Product Backlog untuk Sprint berikutnya. Product Backlog dapat dirubah secara keseluruhan sebagai tanggapan atas peluang-peluang baru.

5.3.4. Sprint Retrospective



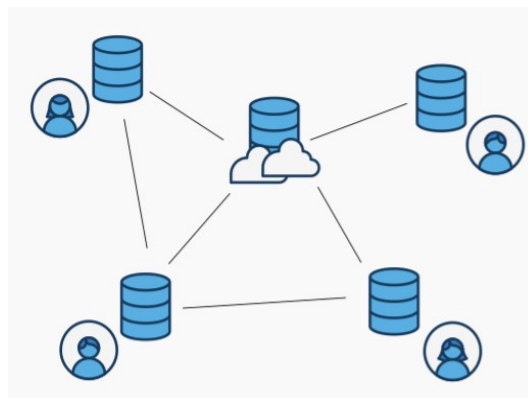
Sprint Retrospective adalah sebuah kesempatan bagi Tim Scrum untuk meninjau dirinya sendiri dan membuat perencanaan mengenai peningkatan yang akan dilakukan di Sprint berikutnya. Sprint Retrospective dilaksanakan setelah Sprint Review selesai dan sebelum Sprint Planning berikutnya. Ini adalah acara dengan batasan waktu maksimum selama tiga jam untuk Sprint yang berdurasi satu bulan. Untuk Sprint yang lebih pendek, batasan waktunya biasanya lebih singkat. Scrum Master memastikan bahwa acara ini dilaksanakan dan setiap hadirin memahami tujuannya. Scrum Master mendukung Tim Scrum untuk melaksanakannya dalam batasan waktu yang telah ditentukan. Scrum Master berpartisipasi sebagai rekan yang bertanggungjawab terhadap proses Scrum.

Tujuan dari Sprint Retrospective adalah:

- Meninjau bagaimana Sprint yang telah selesai berlangsung, termasuk hal-hal yang berkaitan dengan orang-orangnya, hubungan antara orang-orang, proses, dan perangkat kerja;
- Mengidentifikasi dan mengurutkan hal-hal utama yang berjalan baik, dan hal-hal yang berpotensi untuk ditingkatkan; dan,
- Membuat rencana implementasi, dengan tujuan peningkatan cara-cara kerja Tim Scrum.

5.4. Git

Git adalah perangkat lunak pengendali versi atau proyek manajemen kode perangkat lunak yang diciptakan oleh Linus Torvalds. Pada awalnya ditujukan untuk pengembangan kernel Linux. Desain Git terinspirasi oleh BitKeeper dan Monotone. Git pada awalnya hanya dirancang sebagai mesin tingkat rendah yang dapat digunakan oleh tampilan muka (front end) lain seperti Cogito atau StGIT. Namun selanjutnya proyek inti Git telah berkembang menjadi pengendali revisi lengkap yang dapat digunakan langsung. Saat ini, beberapa perangkat lunak terkenal menggunakan Git sebagai pengendali revisinya, antara lain kernel Linux, Server X.org, pengembangan inti OLPC (One Laptop per Child), serta kerangka kerja web Ruby on Rails.



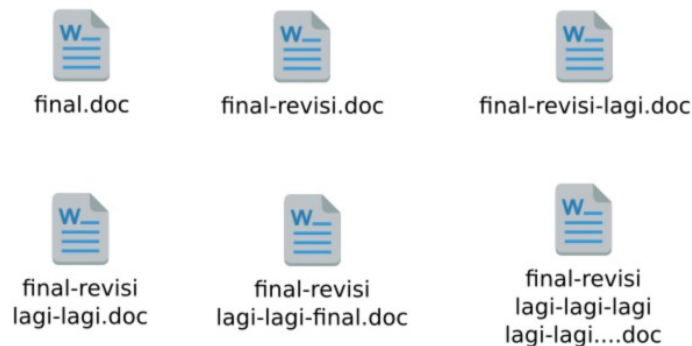
Ilustrasi Distributed Revision Control

Semua orang yang terlibat dalam coding sebuah proyek akan menyimpan database Git, sehingga akan memudahkan dalam mengelola proyek baik online maupun offline. Dalam Git terdapat merge, yaitu aktifitas penggabungan kode.

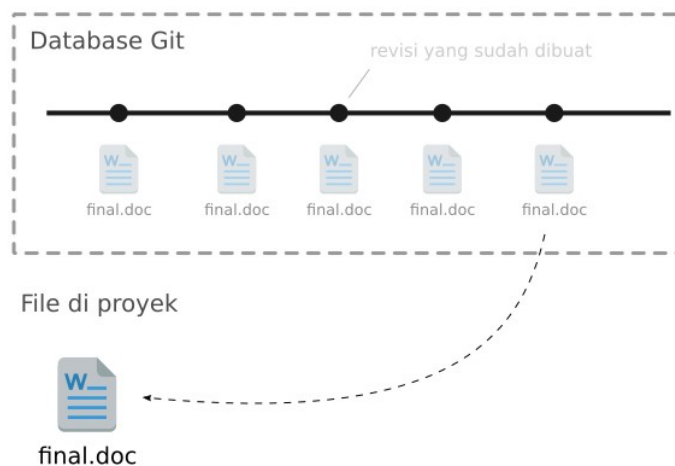


Ilustrasi Merge

Berikut merupakan berbanding sebelum dan sesudah menggunakan Git.



Ilustrasi Sebelum Menggunakan Git



Ilustrasi setelah menggunakan Git

Git sangat penting bagi programmer selain untuk mengontrol versi, git juga digunakan untuk kolaborasi. Saat ini Git menjadi salah satu tool terpopuler yang digunakan pengembangan software open source maupun closed source. Beberapa perusahaan raksasa yang menggunakan Git diantaranya adalah Google, Microsoft, Facebook dan berbagai perusahaan raksasa lainnya.

Berikut ini ada beberapa manfaat yang akan Anda rasakan setelah bisa menggunakan Git :

- Bisa menyimpan seluruh versi source code.

- Bisa paham cara kolaborasi dalam proyek.
- Bisa ikut berkontribusi ke proyek open-source.
- Lebih aman digunakan untuk kolaborasi, karena kita bisa tahu apa yang diubah dan siapa yang mengubahnya.
- Bisa memahami cara deploy aplikasi modern.
- dan sebagainya.

5.5. Instalasi dan Konfigurasi Git

5.5.1. Instalasi Git di Linux

Berikut merupakan beberapa cara instalasi Git pada GNU/Linux yang terbagi kedalam beberapa distro, distro keluarga Debian dapat menggunakan perintah apt sebagai berikut.

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

Untuk keluarga Fedora dapat menggunakan perintah yum seperti berikut.

```
sudo yum install git
```

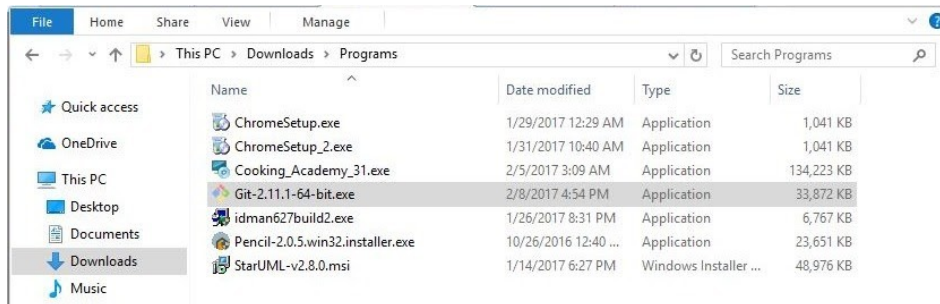
Untuk mengecek versi Git kita bisa gunakan perintah dibawah ini.

```
git --version
```

5.5.2. Instalasi Git di Windows

Selanjutnya adalah Instalasi Git di Windows, memang tidak seperti di Linux yang hanya perlu mengetikkan perintah untuk menginstall. Di windows kita harus men-download terlebih dahulu, kemudian melakukan ritual next > next > finish. Tapi dalam ritual tersebut, ada pilihan yang harus diperhatikan agar perintah git dapat dikenali di CMD.

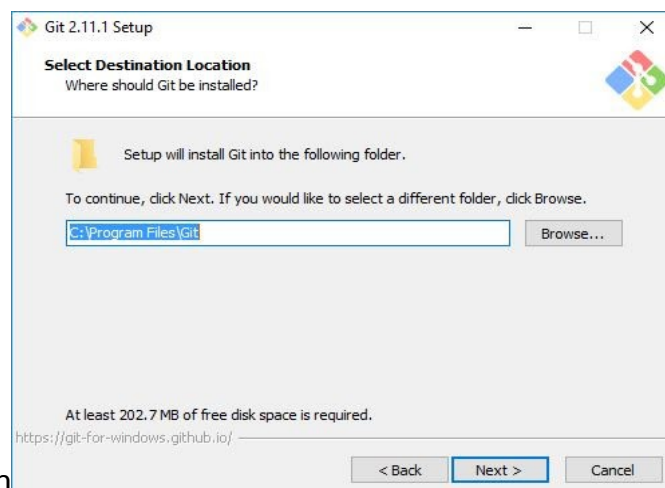
- Untuk Mendownload Git cukup kita buka website resminya Git (git-scm.com). Kemudian unduh Git sesuai dengan arsitektur komputer kita. Kalau menggunakan 64bit, unduh yang 64bit. Begitu juga kalau menggunakan 32bit.
- klik 2x file instaler Git yang sudah didownload.



- Setelah itu akan muncul informasi lisensi Git, klik Next > untuk melanjutkan.

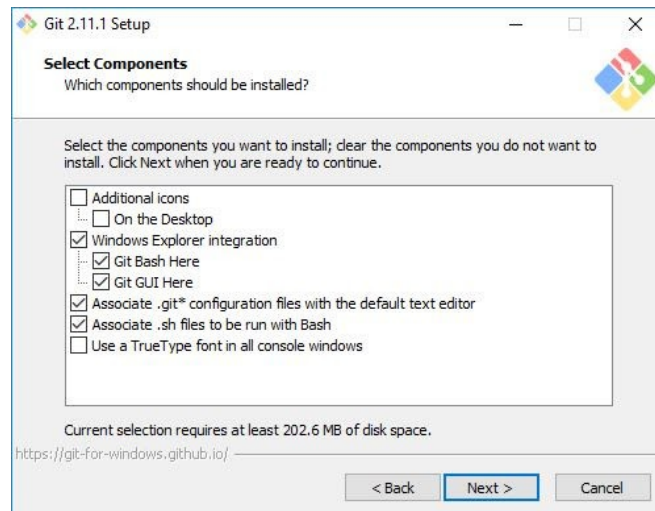


- Selanjutnya menentukan lokasi instalasi. Biarkan saja apa adanya, kemudian klik Next

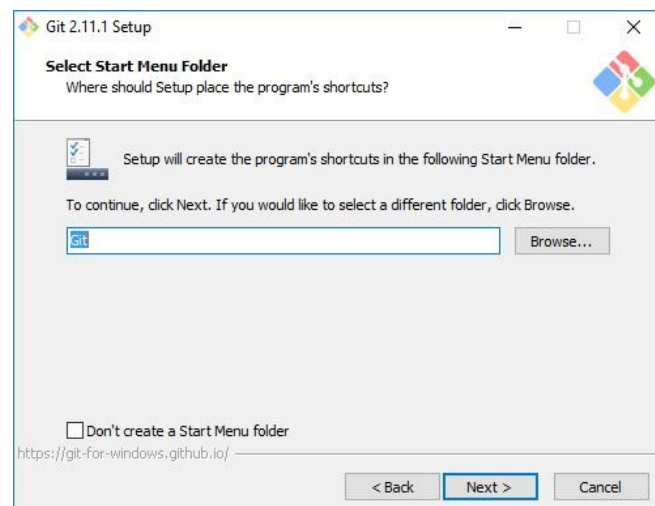


- Selanjutnya menentukan komponen yang akan diinstall. Biarkan saja seperti ini kemudian klik Next.

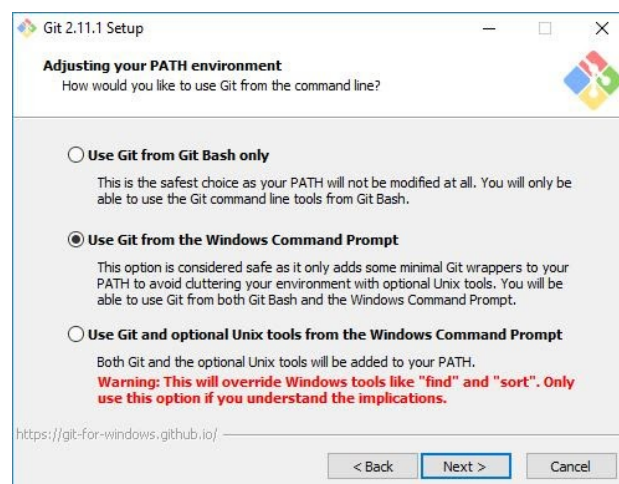




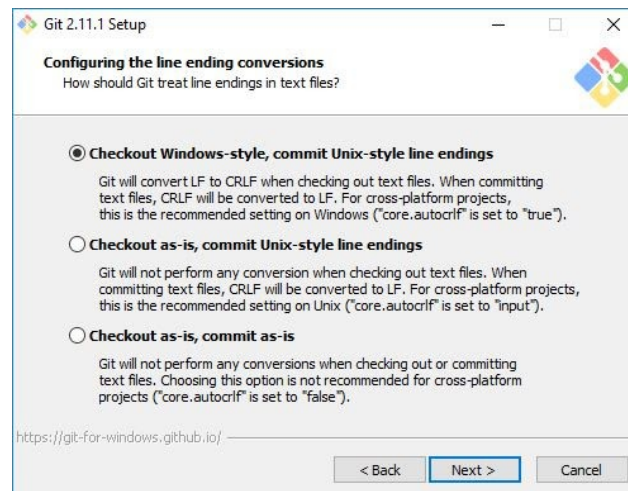
- Selanjutnya pemilihan direktori start menu, klik Next



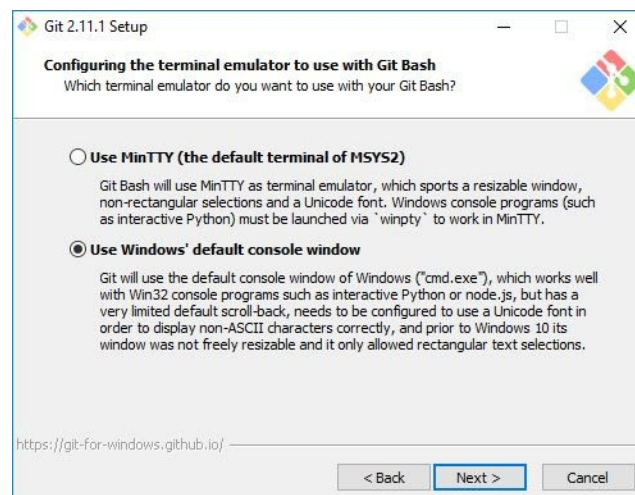
- Selanjutnya pengaturan PATH Environment. Pilih yang tengah agar perintah git dapat di kenali di Command Prompt (CMD). Setelah itu klik Next



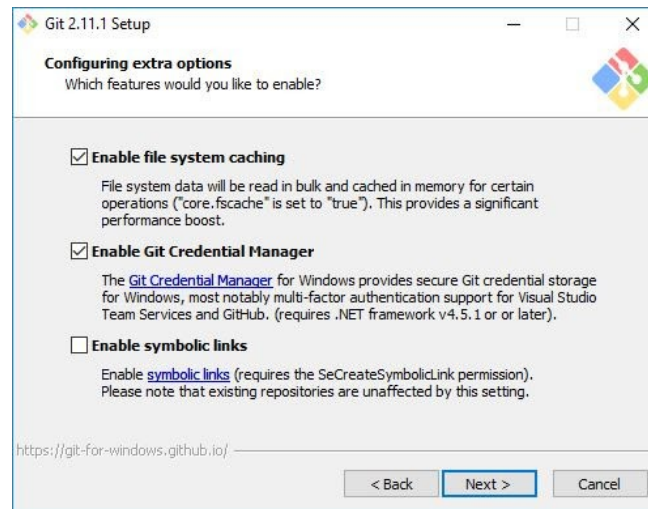
- Selanjutnya konversi line ending. Biarkan saja seperti ini, kemudian klik Next >



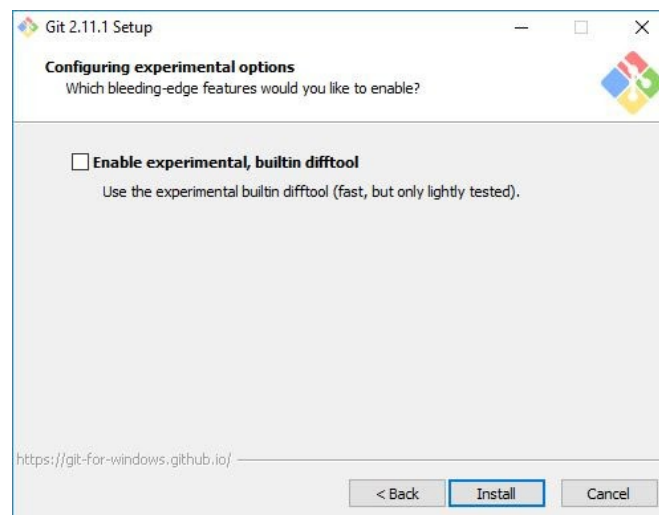
- Pada pemilihan emulator terminal. Pilih saja windows console, kemudian klik Next.



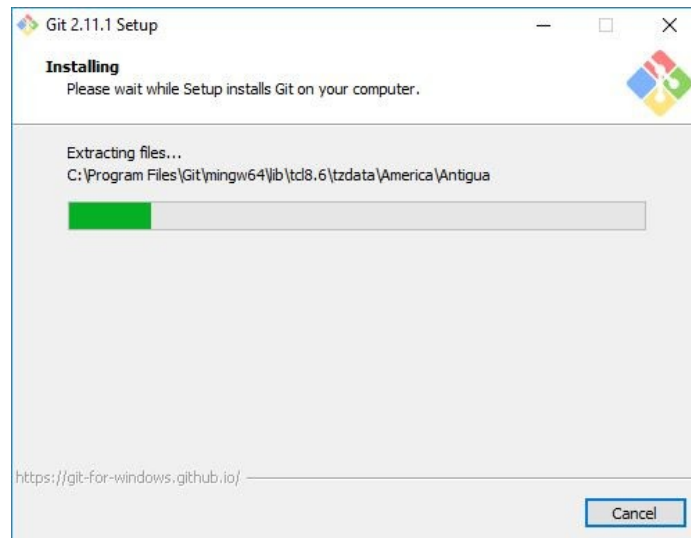
- Selanjutnya pemilihan opsi ekstra. Klik saja Next.



- Selanjutnya pemilihan opsi eksperimental, langsung saja klik Install untuk memulai instalasi.



- Tunggu beberapa saat, instalasi sedang dilakukan.



- Setelah selesai, kita bisa langsung klik Finish.



Git sudah terinstal di Windows. Untuk mencobanya, silahkan buka CMD atau PowerShell, kemudian ketik perintah `git --version`.

5.5.3. Konfigurasi Git

Ada beberapa konfigurasi yang harus dipersiapkan sebelum mulai menggunakan Git, seperti name dan email. Silahkan lakukan konfigurasi dengan perintah berikut ini.

```
git config --global user.name "Irfan Herfiandana"
git config --global user.email irfanherfiandana@gmail.com
```

Kemudian periksa konfigurasinya dengan menggunakan perintah berikut.

```
git config --list
```

Apabila berhasil tampil seperti gambar berikut ini, berarti konfigurasi berhasil.

```
ubuntu@ubuntu-s-1vcpu-2gb-sgp1-01:~$ git config --global user.name "Irfan Herfiandana"
ubuntu@ubuntu-s-1vcpu-2gb-sgp1-01:~$ git config --global user.email irfanherfiandana@gmail.com
ubuntu@ubuntu-s-1vcpu-2gb-sgp1-01:~$ git config --list
user.name=Irfan Herfiandana
user.email=irfanherfiandana@gmail.com
```

Konfigurasi Git

5.5.4. Perisipan File Untuk Git

Karena pada git kita akan belajar untuk mengontrol sebuah proyek yang kita miliki, maka dari itu kita harus mempersiapkan proyek yang akan kita kontrol. Disini kita akan coba membuat beberapa file html dan folder yang akan kita gunakan.

Pertama kita buat sebuah folder project kita dengan nama proyekgit. Setelah itu kita buat sebuah file dengan nama **index.html**, isikan script berikut di file tersebut.

```
<!DOCTYPE html>

<html>

  <head>

    <meta charset="utf-8">

    <title>Belajar Git DevOps Cilsy</title>

  </head>

  <body>

    <p>Hello Semua, kita sedang belajar Git</p>

  </body>

</html>
```

Setelah itu simpan file, lalu kita buat file lain dengan nama **about.html** dan **contact.html** lalu masukan script yang sama dengan **index.html** dalam kedua file baru tersebut. Buat juga file dengan nama test.php lalu isikan script berikut didalamnya.




```
<?php
echo "Ini adalah test";
?>
```

Setelah itu buat beberapa direktori dengan nama **vendor**, **cache**, dan **upload**. Sehingga jika semua sudah kita buat akan menghasilkan seperti dibawah ini.

```
ubuntu@ip-172-31-12-225:~/belajargit$ ls
about.html  cache  contact.html  index.html  test.php  upload  vendor
ubuntu@ip-172-31-12-225:~/belajargit$
```

5.6. Membuat Repositori dan Revisi

5.6.1. Membuat Repositori

Repositori (*repository*) dalam bahasa indonesia artinya gudang. Repositori sendiri merupakan istilah yang digunakan untuk direktori proyek yang menggunakan Git. Jika kita memiliki sebuah direktori dengan nama cilsy dan di dalamnya sudah menggunakan git, maka kita sudah punya repositori bernama cilsy.

Pembuatan repositori dapat dilakukan dengan perintah : **git init nama-direktori**

```
git init cilsy
```

Perintah tersebut akan membuat direktori bernama proyek-01. Kalau direktorinya sudah ada, maka Git akan melakukan inisialisasi di dalam direktori tersebut. Perintah git init akan membuat sebuah direktori bernama .git di dalam proyek kita. Direktori ini digunakan Git sebagai database untuk menyimpan perubahan yang kita lakukan.

Hati-hati apabila kita menghapus direktori ini, maka semua rekaman atau catatan yang dilakukan oleh Git akan hilang.

Contoh lainnya perintah berikut ini akan membuat repositori pada direktori saat ini (working directory).

```
git init .
```

Tanda titik (.) artinya kita akan membuat repository pada direktori tempat kita berada saat ini. Perintah berikut ini akan membuat repository pada direktori `/var/www/html/proyekweb/`.

```
git init /var/www/html/proyekweb
```

5.6.1.1. *Gitignore*

Gitignore (*.gitignore*) merupakan sebuah file yang berisi daftar nama-nama file dan direktori yang akan diabaikan oleh Git. Perubahan apapun yang kita lakukan terhadap file dan direktori yang sudah masuk ke dalam daftar *.gitignore* tidak akan dicatat oleh Git.

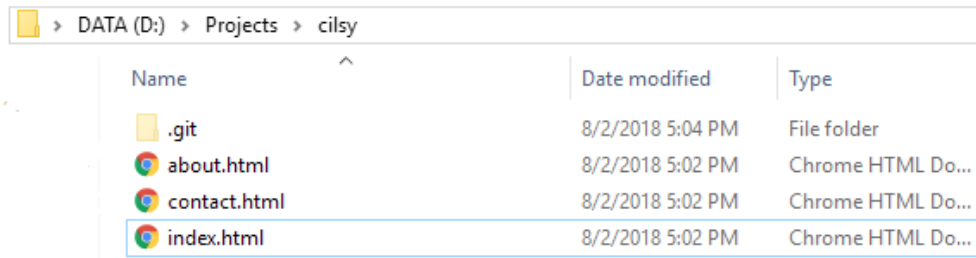
Untuk dapat menggunakan *.gitignore*, buat saja sebuah file bernama *.gitignore* dalam root direktori **proyek/repo**. Misalnya kita isi filenya seperti berikut :

```
/vendor/  
/upload/  
/cache  
test.php
```

Pada contoh file *.gitignore* di atas, kita memasukan direktori vendor, upload, cache dan file test.php. File dan direktori tersebut akan diabaikan oleh Git. Pembuatan file *.gitignore* sebaiknya dilakukan di awal pembuatan repository.

5.6.2. Revisi

Sebelumnya kita sudah membuat repository kosong. Sekarang kita coba tambahkan sebuah file baru. Sebagai contoh, kita akan menambahkan tiga file HTML kosong.



Name	Date modified	Type
.git	8/2/2018 5:04 PM	File folder
about.html	8/2/2018 5:02 PM	Chrome HTML Do...
contact.html	8/2/2018 5:02 PM	Chrome HTML Do...
index.html	8/2/2018 5:02 PM	Chrome HTML Do...

Pembuatan file html

Setelah ditambahkan, coba ketikkan perintah `git status` untuk melihat status repositorinya.

```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilsy (master)
$ git status
On branch master

Initial commit

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

    about.html
    contact.html
    index.html

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

Melihat status Git

Berdasarkan keterangan di atas, saat ini kita berada cabang (branch) master dan ada tiga file yang belum ditambahkan ke Git.

Ada Tiga Kelompok Kondisi File dalam Git yang diantaranya sebagai berikut.

1. Modified

Modified adalah kondisi dimana revisi atau perubahan sudah dilakukan, tetapi belum ditandai dan belum disimpan di version control. Contohnya pada gambar di atas, ada tiga file HTML yang dalam kondisi modified.

2. Staged

Staged adalah kondisi dimana revisi sudah ditandai, tetapi belum disimpan di version control. Untuk mengubah kondisi file dari modified ke staged gunakan perintah *git add nama_file*. Contoh:

```
git add index.html
```

3. Committed

Committed adalah kondisi dimana revisi sudah disimpan di version control. perintah untuk mengubah kondisi file dari staged ke committed adalah *git commit*.

Sekarang kita sudah tahu kondisi-kondisi file dalam Git. Selanjutnya, silahkan ubah kondisi tiga file HTML tadi menjadi staged dengan perintah *git add*.

```
git add index.html  
git add about.html  
git add contact.html
```

Kita dapat melakukan seperti dibawah ini untuk menandai banyak.

```
git add index.html about.html contact.html
```

Atau seperti ini untuk add extention yang dipilih.

```
git add *.html
```

Jika ingin add semuanya, gunakan perintah ini(semua file dan direktori di current directory).

```
git add .
```

Setelah itu cobalah ketik perintah *git status* lagi. Kondisi filenya sekarang akan menjadi *staged*.

```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git status
On branch master

Initial commit

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

        new file:   about.html
        new file:   contact.html
        new file:   index.html
```

Melihat status Staged

Setelah itu, ubah kondisi file tersebut ke committed agar semua perubahan disimpan oleh Git.

```
git commit -m 'Commit Pertama'
```

```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git commit -m 'Commit Pertama'
[master (root-commit) 689c874] Commit Pertama
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 about.html
 create mode 100644 contact.html
 create mode 100644 index.html

Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Melihat status Commit

Sekarang kita akan mencoba untuk membuat Revisi kedua. Misalkan skenarionya adalah ada perubahan yang akan kita lakukan pada file index.html. Silahkan modifikasi isi file index.html. Sebagai contoh kita mengisinya seperti ini.

```
<!DOCTYPE html>

<html>

  <head>

    <meta charset="utf-8">

    <title>Belajar Git - Project 01</title>

  </head>

  <body>
```

```
<p>Hello Semua, kita sedang belajar Git</p>

</body>

</html>
```

Setelah itu ketik lagi perintah *git status*.

```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git status
On branch master
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)

        modified:   index.html

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

Melihat git status

Terlihat di sana, file index.html sudah dimodifikasi. Kondisinya skarang berada dalam modified. Lakukan commit lagi seperti revisi pertama.

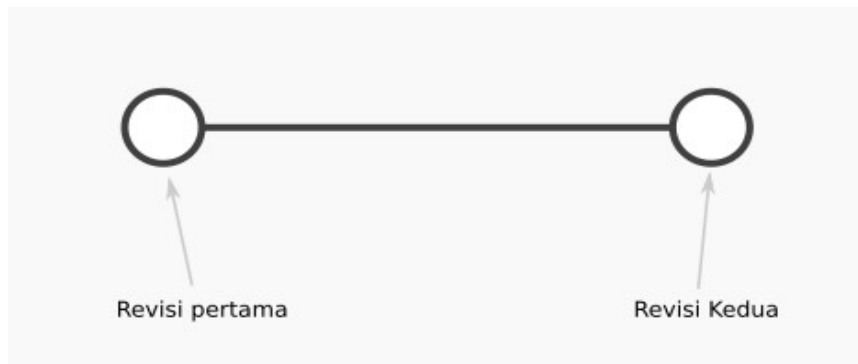
```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git add index.html

Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git commit -m 'tambah script'
[master 331f605] tambah script
1 file changed, 10 insertions(+)

Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Melihat status commit

Dengan demikian, revisi kedua sudah disipan oleh Git. Mungkin anda belum tahu maksud dari argumen -m, argumen tersebut untuk menambahkan pesan setiap menyimpan revisi.



Ilustrasi revisi

Sekarang Git sudah mencatat dua revisi yang sudah kita lakukan. Kita bisa ibaratkan revisi-revisi ini sebagai checkpoint pada Game. Apabila nanti ada kesalahan, kita bisa kembali ke checkpoint ini.

5.6.2.1. **Melihat Log Revisi.**

Pada skenario sebelumnya, kita sudah membuat dua revisi pada repositori project-01. Sekarang bagaimana caranya kita melihat catatan log dari revisi-revisi tersebut? Git sudah menyediakan perintah git log untuk melihat catatan log perubahan pada repositori. Contoh penggunaannya:

```
git log
```

```

Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git log
commit 331f6058c6ed132af24d5ca1f17a306d5683f9f1 (HEAD -> master)
Author: Irfan Herfiandana <IrfanHerfiandana@gmail.com>
Date: Thu Aug 2 17:21:05 2018 +0700

    tambah script

commit 689c87477944198456673de50e225aa7f6272703
Author: Irfan Herfiandana <IrfanHerfiandana@gmail.com>
Date: Thu Aug 2 17:17:06 2018 +0700

    Commit Pertama
  
```

Melihat log revisi

Pada gambar di atas, terdapat dua revisi perubahan yang telah dilakukan. Untuk menampilkan log yang lebih pendek, kita bisa menambahkan argumen --oneline.

```
git log --oneline
```

```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git log --oneline
331f605 (HEAD -> master) tambah script
689c874 Commit Pertama
```

Melihat log lebih pendek

5.6.2.2. Log pada Nomor Revisi/Commit.

Untuk melihat log pada revisi tertentu, kita bisa memasukan nomer revisi/commit.

```
git log 331f6058c6ed132af24d5ca1f17a306d5683f9f1
```

```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git log 331f6058c6ed132af24d5ca1f17a306d5683f9f1
commit 331f6058c6ed132af24d5ca1f17a306d5683f9f1 (HEAD -> master)
Author: Irfan Herfiandana <IrfanHerfiandana@gmail.com>
Date: Thu Aug 2 17:21:05 2018 +0700

    tambah script

commit 689c87477944198456673de50e225aa7f6272703
Author: Irfan Herfiandana <IrfanHerfiandana@gmail.com>
Date: Thu Aug 2 17:17:06 2018 +0700

    Commit Pertama
```

Melihat revisi nomor commit

5.6.2.3. Log pada File Tertentu.

Untuk melihat revisi pada file tertentu, kita dapat memasukan nama filenya.

```
git log index.html
```

```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git log index.html
commit 331f6058c6ed132af24d5ca1f17a306d5683f9f1 (HEAD -> master)
Author: Irfan Herfiandana <IrfanHerfiandana@gmail.com>
Date: Thu Aug 2 17:21:05 2018 +0700

    tambah script

commit 689c87477944198456673de50e225aa7f6272703
Author: Irfan Herfiandana <IrfanHerfiandana@gmail.com>
Date: Thu Aug 2 17:17:06 2018 +0700

    Commit Pertama
```

Melihat log pada file tertentu


```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git log --author='Irfan Herfiandana'
commit 331f6058c6ed132af24d5ca1f17a306d5683f9f1 (HEAD -> master)
Author: Irfan Herfiandana <IrfanHerfiandana@gmail.com>
Date: Thu Aug 2 17:21:05 2018 +0700

    tambah script

commit 689c87477944198456673de50e225aa7f6272703
Author: Irfan Herfiandana <IrfanHerfiandana@gmail.com>
Date: Thu Aug 2 17:17:06 2018 +0700

    Commit Pertama
```

Melihat log author tertentu

5.6.2.4. Melihat Perbandingan Perubahan yang Dilakukan pada Revisi.

Gunakan perintah berikut ini untuk melihat perubahan apa saja yang dilakukan pada revisi tertentu.

```
git diff commit_number
```

```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git diff 689c87477944198456673de50e225aa7f6272703
diff --git a/index.html b/index.html
index e69de29..481cc02 100644
--- a/index.html
+++ b/index.html
@@ -0,0 +1,10 @@
+<!DOCTYPE html>
+<html>
+  <head>
+    <meta charset="utf-8">
+    <title>Belajar Git - Project 01</title>
+  </head>
+  <body>
+    <p>Hello Semua, Saya sedang belajar Git</p>
+  </body>
+</html>
```

Melihat perbedaan

Lihatlah hasil di atas, simbol plus (+) artinya kode yang ditambahkan. Sedangkan kalau ada kode yang dihapus simbolnya akan menggunakan minus (-).

Sekarang kita akan mencoba merubah isi dari index.html untuk melihat perbedaannya. Kita coba cek lagi bahwa berikut adalah kode original sebelum diubah:

```
<!DOCTYPE html>

<html>
```

```
<head>

  <meta charset="utf-8">

  <title>Belajar Git - Project 01</title>

</head>

<body>

  <p>Hello Semua, kita sedang belajar Git</p>

</body>

</html>
```

Berikut adalah kode setelah diubah :

```
<!DOCTYPE html>

<html>

  <head>

    <meta charset="utf-8">

    <title>Belajar Git - Project 01</title>

  </head>

  <body>

    <p>Hello Dunia!, kita sedang belajar Git</p>

  </body>

</html>
```

Setelah itu lakukan jalankan perintah git diff lagi.

```
Geekseat@DESKTOP-MH44SII MINGW64 /d/Projects/cilisy (master)
$ git diff
diff --git a/index.html b/index.html
index 481cc02..c5082e6 100644
--- a/index.html
+++ b/index.html
@@ -5,6 +5,6 @@
     <title>Belajar Git - Project 01</title>
   </head>
   <body>
-     <p>Hello Semua, Saya sedang belajar Git</p>
+     <p>Hello Dunia!, Saya sedang belajar Git</p>
   </body>
 </html>
```

Melihat hasil akhir perbedaan

Perintah `git diff` akan membandingkan perubahan yang baru saja dilakukan dengan revisi/commit terakhir.

5.6.2.5. **Melihat Perbandingan pada File.**

Apabila kita melakukan banyak perubahan, maka akan banyak sekali tampil output. Karena itu, kita mungkin hanya perlu melihat perubahan untuk file tertentu saja. Untuk melihat perbandingan perubahan pada file tertentu, gunakan perintah berikut.

```
git diff index.html
```

Perintah di atas akan melihat perbedaan perubahan pada file `index.html` saja.

5.6.2.6. **Melihat Perbandingan antar Revisi/Commit.**

Perintah untuk membandingkan perubahan pada revisi dengan revisi yang lain dapat menggunakan perintah dibawah ini.

```
git diff <nomer commit> <nomer commit>
```

5.6.2.7. **Perbandingan Antar Cabang (Branch).**

Kita memang belum masuk ke materi percabangan di Git. Tapi tidak ada salahnya mengetahui cara melihat perbandingan perubahan antar branch.

```
git diff <nama branch> <nama branch>
```

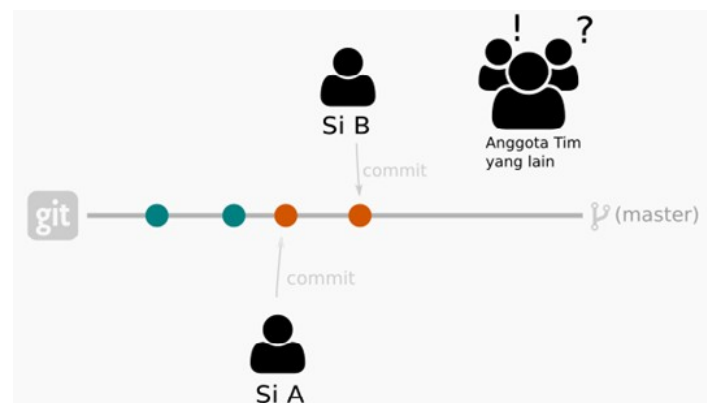
Kita sudah pelajari fungsi dari perintah `git diff`. Perintah ini untuk melihat perbandingan perubahan apa saja yang telah dilakukan pada repositori.

5.6.3. Exercise

1. Buat sebuah repository Git baru dengan nama Anda !
2. Buat file gitignore pada repository tersebut !
3. Buat beberapa file dan lakukan revisi sebanyak tiga kali !

5.7. Branching

Bayangkan anda sedang bekerja dengan tim pada suatu repository Git. Repository ini dikerjakan secara bersama-sama. Kadang akan terjadi konflik, karena kode yang kita tulis berbeda dengan yang lain. Misalnya, Si A menulis kode untuk fitur X dengan algoritma yang ia ketahui. Sedangkan si B menulis dengan algoritma yang berbeda. Lalu mereka melakukan commit, dan kode sumber jadi berantakan. Anggota tim yang lain menjadi pusing.



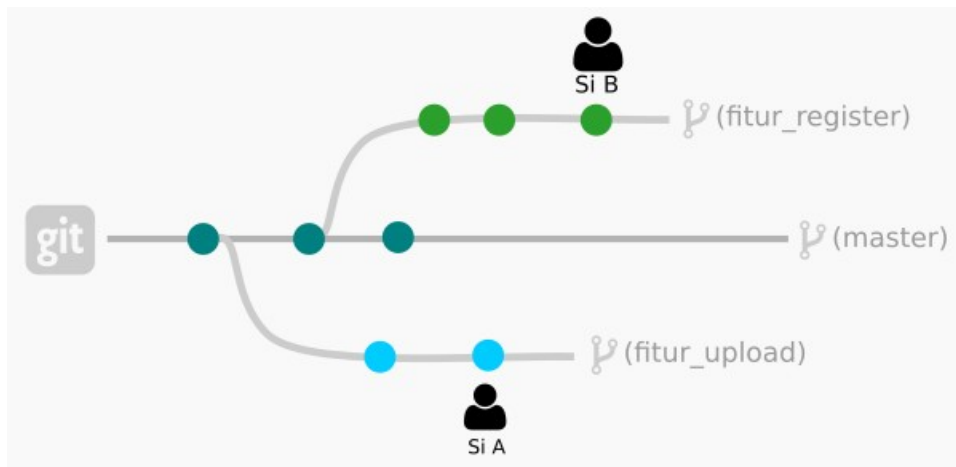
Ilustrasi kebingungan akibat perbedaan commit

Agar tidak terjadi hal yang seperti ini, kita harus membuat branch (branch) tersendiri. Misalnya, si A akan mengerjakan fitur X, maka dia harus membuat branch sendiri. Si A akan bebas melakukan apapun di branchnya tanpa mengganggu branch utama (master).

5.7.1. Membuat branch Baru

Perintah untuk membuat branch adalah `git branch`, kemudian diikuti dengan nama branchnya. Kita buat sebuah contoh sebagai berikut.

```
git branch fitur_register
```



Ilustrasi Branch

Sekarang setiap orang memiliki branchnya masing-masing. Mereka bebas bereksperimen. Untuk melihat branch apa saja yang ada di repositori, gunakan perintah git branch. Contoh:

```
$ git branch
    halaman_login
* master
```

Tanda bintang (*) artinya branch yang sedang aktif atau Kita sedang berada di sana. Untuk memantapkan pemahaman tentang percabangan Git, mari kita coba praktek. Pada repositori, buatlah sebuah branch baru.

```
git branch halaman_login
```

Setelah itu, pindah ke branch yang baru saja kita buat dengan perintah:

```
git checkout halaman_login
```

Lalu tambahkan file login.html, isinya terserah anda. Jangan lupa untuk menggunakan perintah git status untuk melihat status repositori. Setelah kita menambahkan file login.html, Selanjutnya kita lakukan commit.

```
git add login.html
git commit -m "membuat file login.html"
```

Revisi kita pada branch halaman_login sudah disimpan. Sekarang coba kembali ke branch master.

```
git checkout master
```

Apakah anda menemukan file login.html? Pasti tidak! Sekarang kembali lagi ke branch halaman_login.

```
git checkhout halaman_login
```

Cek lagi, apakah sekarang file login.html sudah ada?

```
project-01/  
├─ index.html  
└─ login.html
```

5.7.2. Menggabungkan branch

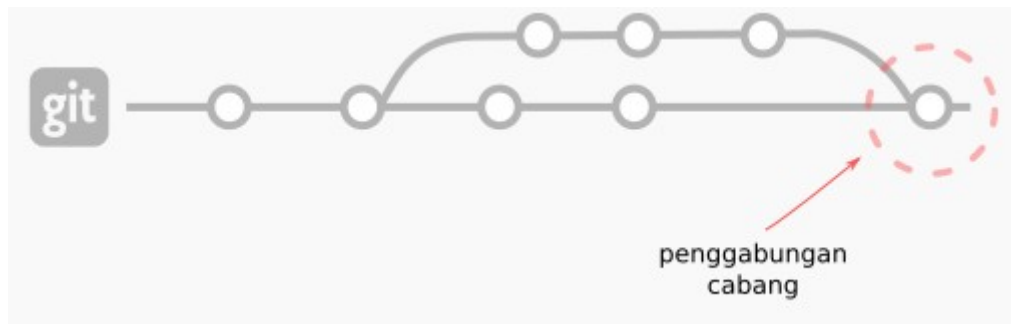
Anggaplah kita sudah selesai membuat fitur login di branch halaman_login. Sekarang kita ingin Menggabungkannya dengan branch master (utama). Pertama, kita harus pindah dulu ke branch master.

```
git checkout master
```

Setelah itu, barulah kita bisa menggabungkan dengan perintah git merge.

```
git merge halaman_login
```

Sekarang lihat, file login.html sudah ada di branch master.



Ilustrasi penggabungan branch

5.7.3. Mengatasi Adanya Bentrok

Bentrok biasanya terjadi jika ada dua orang yang mengedit file yang sama. Kenapa bisa begitu, 'kan mereka sudah punya branch masing-masing? Bisa jadi, di branch yang mereka kerjakan ada file yang sama dengan branch lain. Kemudian, saat digabungkan terjadi bentrok.

Mengatasi bentrok adalah tugas dari pemilik atau pengelola repostori. Dia harus bertindak adil, kode mana yang harus diambil. Biasanya akan ada proses diskusi dulu dalam mengambil keputusan. Sekarang kita akan coba membuat bentrokan.

Pertama kita pindah dulu ke branch halaman_login.

```
git checkout halaman_login
```

Setelah itu, edit file login.html atau index.html, karena kedua file tersebut ada di kedua branch yang akan kita gabungkan.

```
$ git diff
diff --git a/login.html b/login.html
index 23a3f5c..eea5658 100644
--- a/login.html
+++ b/login.html
```

```
@@ -1 +1 @@
-di sini berisi kode untuk halaman login
+<p>di sini berisi kode untuk halaman login<p>
```

Setelah itu, lakukan commit lagi:

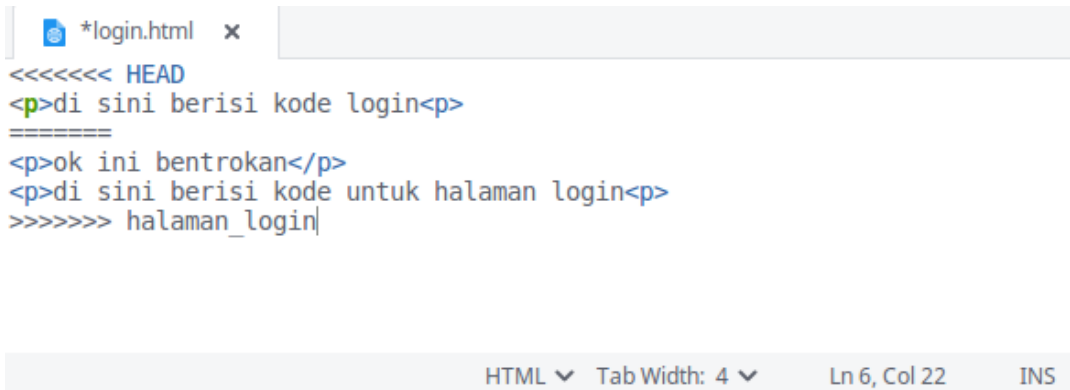
```
git add login.html
git commit -m "ubah isi login.html"
```

Selanjutnya pindah ke branch master dan lakukan perubahan juga di branch ini. Ubah file yang sama seperti di branch halaman_login, setelah itu, lakukan commit di branch master.

```
git add login.html
git commit -m "ubah isi login.html di branch master"
```

Terakhir, coba gabungkan branch halaman_login dengan branch master, maka akan terjadi bentrok.

```
$ git merge halaman_login
Auto-merging login.html
CONFLICT (content): Merge conflict in login.html
Automatic merge failed; fix conflicts and then commit the result.
```



```
*login.html x
<<<<<< HEAD
<p>di sini berisi kode login<p>
=====
<p>ok ini bentrokan</p>
<p>di sini berisi kode untuk halaman login<p>
>>>>>> halaman_login

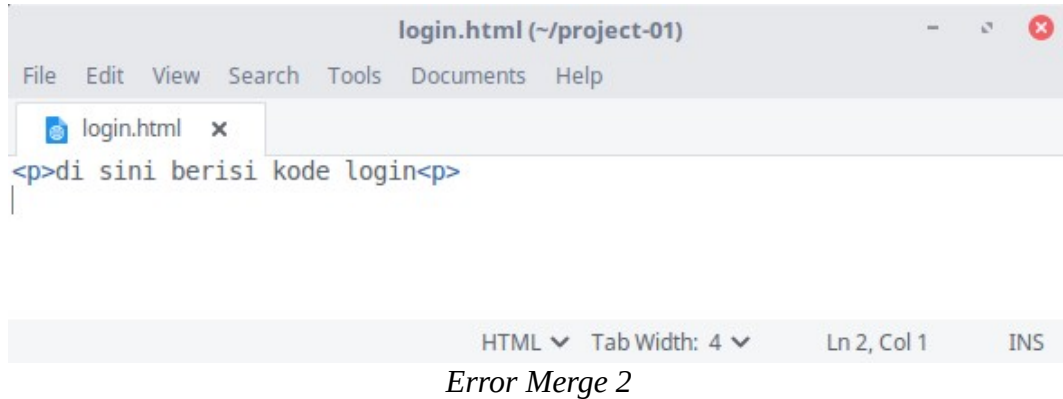
HTML Tab Width: 4 Ln 6, Col 22 INS
```

Error Merge 1

Nah, kita disuruh perbaiki kode yang bentrok. Sekarang buka login.html dengan teks editor.



Kedua kode branch dipisahkan dengan tanda =====. Sekarang.. tugas kita adalah memperbaikinya. Silahkan eliminasi salah satu dari kode tersebut.



Setelah itu lakukan commit untuk menyimpan perubahan ini.

```
git add login.html
git commit -m "perbaiki konflik"
```

5.7.4. Menghapus branch

branch yang sudah mati atau tidak ada pengembangan lagi, sebaiknya dihapus. Agar repositori kita bersih dan rapi. Cara menghapus branch, gunakan perintah git branch dengan argumen -d dan diikuti dengan nama branchnya. Contoh:

```
git branch -d halaman_login
```

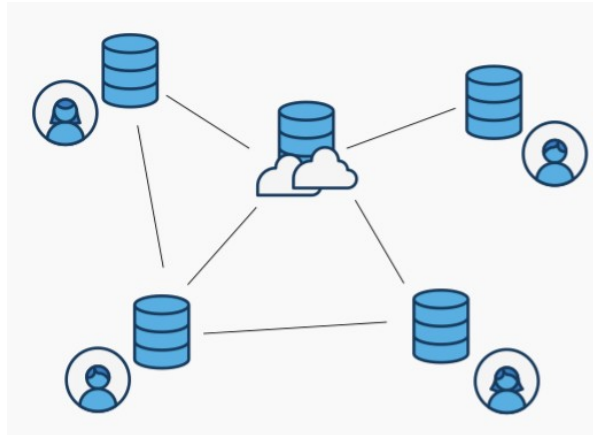
5.7.5. Exercise

1. Buat dua branch baru pada repository yang Anda buat.
2. Buat source code pada kedua branch tersebut dan gabungkan keduanya.

5.8. Remote Repository

Pada proyek pengembangan software yang melibatkan banyak orang (tim), kita tidak hanya akan menyimpan sendiri repository proyeknya. Semua tim yang terlibat dalam pengkodean (coding) akan menyimpan repository lokal di komputernya masing-masing. Setelah itu, akan

dilakukan penggabungan ke repository inti atau remote. Biasanya akan ada repository pusat atau untuk menyimpan source code yang sudah digabungkan (merge) dari beberapa orang.



Ilustrasi remote repository

Di mana menyimpan repository remote-nya? Bisa di server kantor atau bisa juga menggunakan layanan seperti Github, Gitlab, Bitbucket, dll. Github adalah layanan yang paling populer untuk menyimpan (hosting) repository secara remote. Banyak proyek open source tersimpan di sana. Kita akan menggunakan Github pada tutorial ini, pastikan Anda sudah memiliki akun Github dengan cara mendaftar di **github.com**.

5.8.1. Membuat Repositori di GitHub

Silahkan buka Github, kemudian buat sebuah repository dengan nama belajar-git seperti berikut ini.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner / **Repository name** ✓

Great repository names are short and memorable. Need inspiration? How about **solid-lamp**.

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

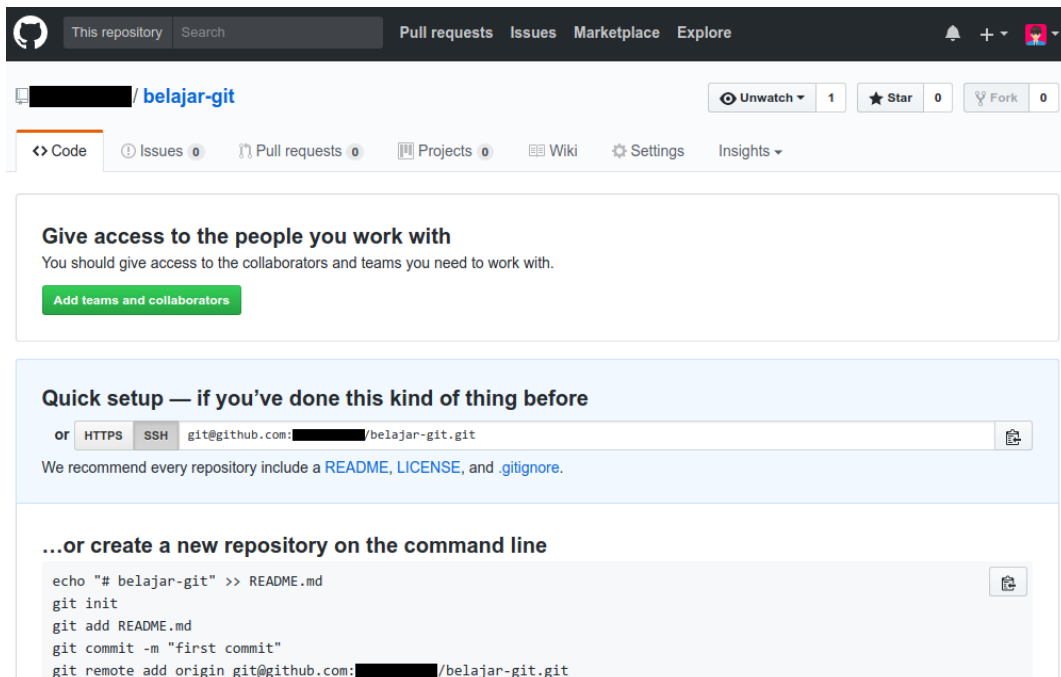
☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

Membuat repository 1

Maka sekarang kita punya repository kosong bernama belajar-git di Github.



The screenshot shows the GitHub repository page for 'belajar-git'. The repository is public and has 1 watch, 0 stars, and 0 forks. The page includes a 'Code' button, a 'Give access to the people you work with' section with an 'Add teams and collaborators' button, a 'Quick setup' section with a 'git@github.com: [username]:belajar-git.git' link, and a section for creating a new repository on the command line with the following commands:

```
echo "# belajar-git" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:[username]:belajar-git.git
```

Membuat repository 2



Silahkan buka kembali repository lokal yang pernah kita buat, yaitu **belajar-git**. Berikutnya kita akan coba upload repository lokal ini ke repository remote Github.

5.8.2. Menambah Remote Repository

Sebelum kita bisa upload semua revisi yang ada di repository lokal, kita harus menambahkan remote repository-nya terlebih dahulu. Remote repository dapat kita tambahkan dengan perintah seperti ini :

```
git remote add github https://github.com/username/belajar-git.git  
git remote add github git@github.com:username/belajar-git.git
```

Perbedaan https dengan SSH adalah bentuk autentikasinya. Untuk https, kita akan diminta user dan password setiap kali melakukan push, sedangkan SSH, kita hanya melakukan 1 kali autentikasi, yaitu dengan mendaftarkan public key kita ke repository.

5.8.3. Menggunakan SSH di GitHub

SSH memungkinkan kita untuk melakukan push ke repository github tanpa login. Berbeda dengan cara yang biasa (melalui HTTPS), kita harus memasukkan username dan password setiap kali melakukan push. Tapi dengan SSH kita tidak akan melakukan itu lagi. Berikut adalah langkah-langkahnya.

5.8.3.1. Membuat SSH Key.

Pertama-tama kita akan membuat sebuah SSH Key. SSH Key ini adalah sebuah kombinasi 2 file terenkripsi (publik dan private) yang akan dicocokkan antara di server Remote repository dengan di lokal (pc/laptop). Ketika 2 file ini dinyatakan cocok, maka kita dianggap sebagai orang yang punya autentikasi tanpa perlu memasukkan password dan username lagi. Cara untuk membuat ssh key adalah masuk ke terminal dan ketikkan ssh-keygen lalu enter. Untuk passphrase dikosongkan saja.

```
ubuntu@ubuntu-s-1vcpu-2gb-sgp1-01:~/.ssh$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa): github
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in github.
Your public key has been saved in github.pub.
The key fingerprint is:
SHA256:QNsRHqJKQnM5a/7DHZr3J7PsM+3p486AjDS9HxF5/s8 ubuntu@ubuntu-s-1vcpu-2gb-sgp1-01
The key's randomart image is:
+---[RSA 2048]---+
| o .. o +. |
| . oo o = o . |
| . . + o o o . |
| o + .. + |
| + =So = o |
| . o * = * . |
| o o + = . . |
| + . oo+o... |
| o. oBB. E |
+---[SHA256]---+
ubuntu@ubuntu-s-1vcpu-2gb-sgp1-01:~/.ssh$ |
```

Membuat SSH-Key

Maka di dalam directory .ssh akan tercipta file baru yaitu id_rsa sebagai private key dan id_rsa.pub sebagai public key.

```
ubuntu@ubuntu-s-1vcpu-2gb-sgp1-01:~/.ssh$ ls
authorized_keys  github  github.pub  id_rsa  id_rsa.pub
ubuntu@ubuntu-s-1vcpu-2gb-sgp1-01:~/.ssh$ |
```

Public key dan private key

5.8.3.2. Jalankan SSH Agent dan Load SSH Key

Untuk memastikan apakah SSH Agent sudah berjalan atau tidak, gunakan perintah ini:

```
ps -e | grep [s]sh-agent
```

Kalau belum berjalan, gunakan perintah berikut ini untuk menjalankan SSH agent:

```
ssh-agent /bin/bash
```

Berikutnya kita Load SSH Key. Gunakan perintah:

```
ssh-add ~/.ssh/id_rsa
```

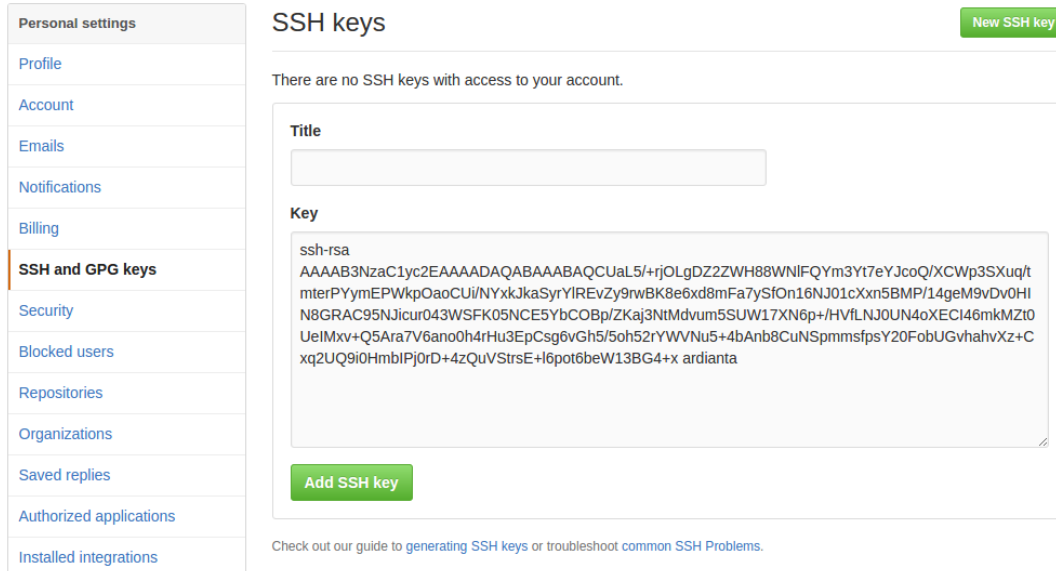
Kemudian untuk mengecek, gunakan perintah:

```
ssh-add -l
```

Tambahkan SSH Key ke Github. Sebelumnya ambil dulu publik key yang sudah anda buat, gunakan perintah cat.

```
cat ~/.ssh/id_rsa.pub
```

Copy isi teks yang ditampilkan. Lalu kembali ke Github, masuk ke menu **Settings> SSH and GPG Keys**, buat key baru dengan mengklik **New SSH Key**. Lalu masukkan key yang sudah dicopy.



Meng-input SSH Key di Github

Sekarang seharusnya laptop Anda sudah terhubung dengan remote repository dengan metode SSH.

5.8.3.3. Uji Konektivitas

Ketik perintah berikut untuk menguji konektivitas SSH ke Github :

```
ssh -T git@github.com
```

Pastikan tidak ada pesan error yang muncul untuk memastikan bahwa konektivitas Github dengan SSH sudah berhasil.

5.8.3.4. Mengubah dan Menghapus Remote Repository

Silahkan ketik perintah `git remote -v` untuk melihat remote apa saja yang sudah ditambahkan.

```
i ~/project-01 $ git remote
i ~/project-01 $ git remote add github git@github.com: username /belajar-git.git
i ~/project-01 $ git remote -v
github git@github.com: username /belajar-git.git (fetch)
github git@github.com: username /belajar-git.git (push)
i ~/project-01 $ git remote
github
```

Mengubah dan menghapus remote repository



Sekarang kita sudah menambahkan remote di dalam repository lokal. Selanjutnya kita bisa melakukan push atau mengirim revisi ke repository remote (Github). Nah untuk menghapus dan mengubah nama remote dapat dilakukan dengan perintah berikut.

Ubah nama remote:

```
git remote rename github kantor
```

Keterangan : github adalah nama remote yang lama, kantor adalah nama remote yang baru.

Hapus remote:

```
git remove github
```

5.8.4. Mengirim Revisi ke Remote Repository

Perintah yang kita gunakan untuk mengirim revisi ke repository remote adalah git push.

```
git push github master
```

Keterangan: github adalah nama remote, master adalah nama branch tujuan.

Mari kita coba, pastikan repository lokal kita sudah memiliki remote.

```
i ~/project-01 $ git remote
i ~/project-01 $ git remote add github git@github.com: username /belajar-git.git
i ~/project-01 $ git remote -v
github git@github.com: username /belajar-git.git (fetch)
github git@github.com: username /belajar-git.git (push)
i ~/project-01 $ git remote
github
```

Mengirim revisi

Setelah itu lakukan beberapa revisi atau commit.

```
git add .
git commit -m "menambahkan beberapa revisi"
```

Sebagai contoh, disini ada 5 catatan revisi.

```
i ~/project-01 $ git log --oneline
865e50c commit hasil revert
023b078 ditambahkan login.html
6cdfdbb ada perubahan
06f735a ditambahkan isi
cf08ca0 commit pertama
```

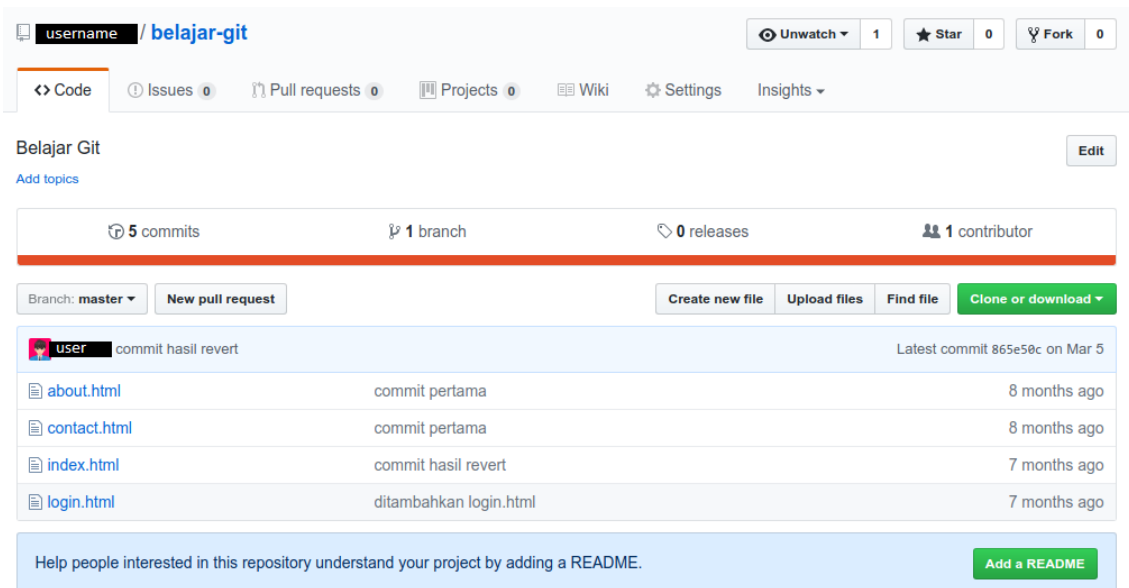
Menampilkan catatan revisi

Maka tinggal kita kirim saja dengan perintah git push github master. Jika muncul seperti ini, artinya push sukses dilakukan.

```
i ~/project-01 $ git push github master
Counting objects: 13, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (13/13), 1.16 KiB | 0 bytes/s, done.
Total 13 (delta 5), reused 0 (delta 0)
remote: Resolving deltas: 100% (5/5), done.
To git@github.com:petanikode/belajar-git.git
* [new branch] master -> master
```

Melakukan push

Sekarang lihat ke Github, pasti semuanya sudah ter-upload ke sana.



The screenshot shows the GitHub interface for a repository named 'belajar-git'. At the top, there are buttons for 'Unwatch', 'Star' (0), and 'Fork' (0). Below this is a navigation bar with tabs for 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Settings', and 'Insights'. The main content area shows the repository name 'Belajar Git' with an 'Edit' button. Below this, there are statistics: '5 commits', '1 branch', '0 releases', and '1 contributor'. A table lists the commit history with columns for the file, commit message, and time ago. The latest commit is 'commit hasil revert' from 8 months ago. At the bottom, there is a button to 'Add a README'.

Memeriksa hasil push

Coba buat revisi lagi di file index.html. Misalnya perubahannya seperti ini:

```
i ~/project-01 $ git diff index.html
diff --git a/index.html b/index.html
index 481cc02..2bca54f 100644
--- a/index.html
+++ b/index.html
@@ -2,9 +2,11 @@
<html>
  <head>
    <meta charset="utf-8">
-    <title>Belajar Git - Project 01</title>
+    <title>Belajar Git: Remote Repository</title>
  </head>
  <body>
    <p>Hello Semua, Saya sedang belajar Git</p>
+    <p>Yay! saya sedang belajar tentang remote repository</p>
  </body>
</html>
```

Melakukan perubahan file index.html



Lalu lakukan commit dan push.

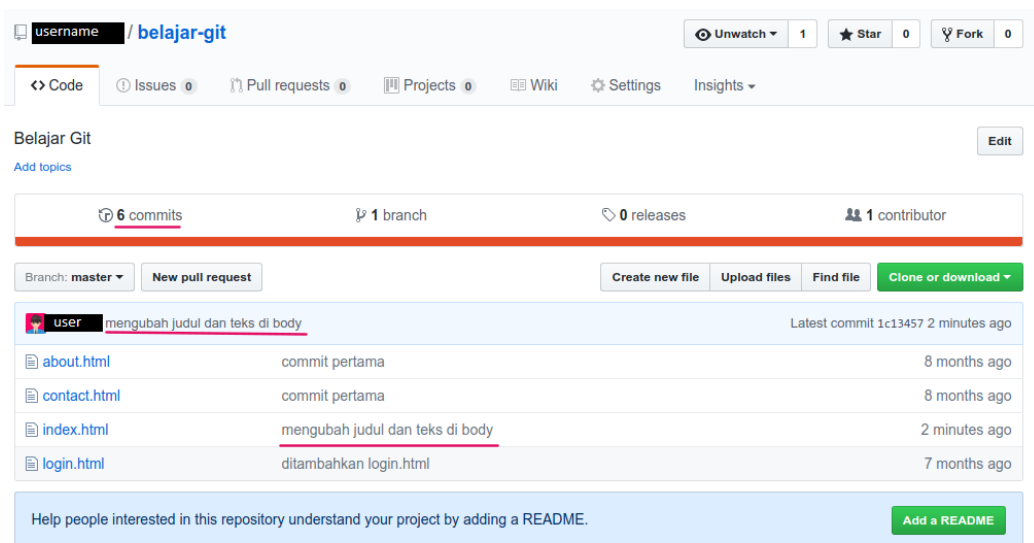
```
git add index.html
git commit -m "mengubah judul dan teks di body"
git push github master
```

Jika berhasil, maka akan tampil seperti ini:

```
i ~/project-01 $
i ~/project-01 $ git add index.html
i ~/project-01 $ git commit -m "mengubah judul dan teks di body"
[master 1c13457] mengubah judul dan teks di body
1 file changed, 3 insertions(+), 1 deletion(-)
i ~/project-01 $ git push github master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 380 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To git@github.com:petanikode/belajar-git
865e50c..1c13457 master -> master
```

Hasil push perubahan

Periksa kembali repository di Github dan perhatikanlah perubahannya.



The screenshot shows the Github repository page for 'username / belajar-git'. The repository has 6 commits, 1 branch, 0 releases, and 1 contributor. The commit history table is as follows:

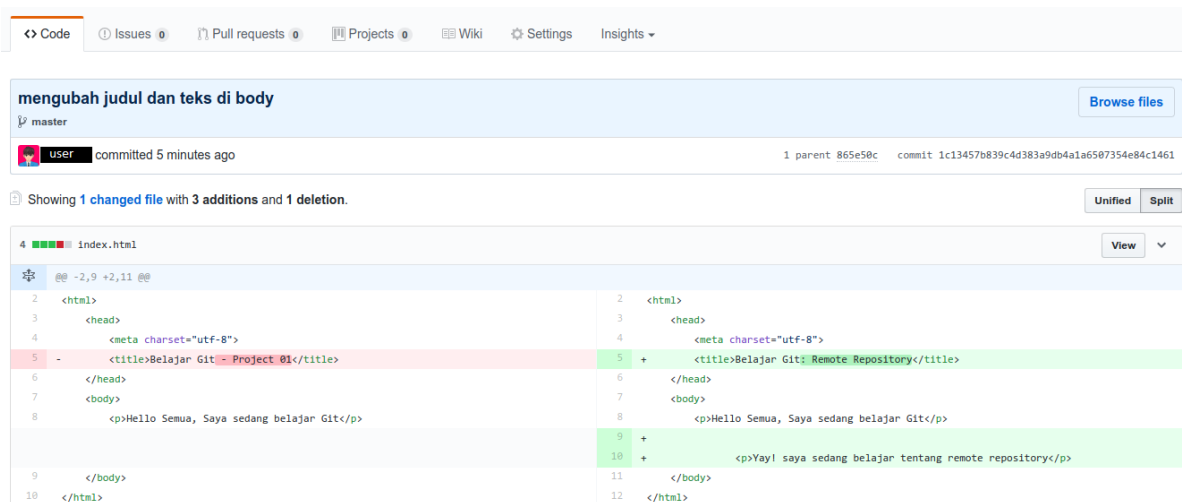
File	Commit Message	Time Ago
about.html	commit pertama	8 months ago
contact.html	commit pertama	8 months ago
index.html	mengubah judul dan teks di body	2 minutes ago
login.html	ditambahkan login.html	7 months ago

At the bottom, there is a prompt to 'Add a README'.

Melihat hasil di Github

Jika kita klik commit terakhir, maka kita akan dibawa ke *git diff*-nya Github. Di sana kita bisa melihat perubahan apa kita yang dilakukan pada commit tersebut.

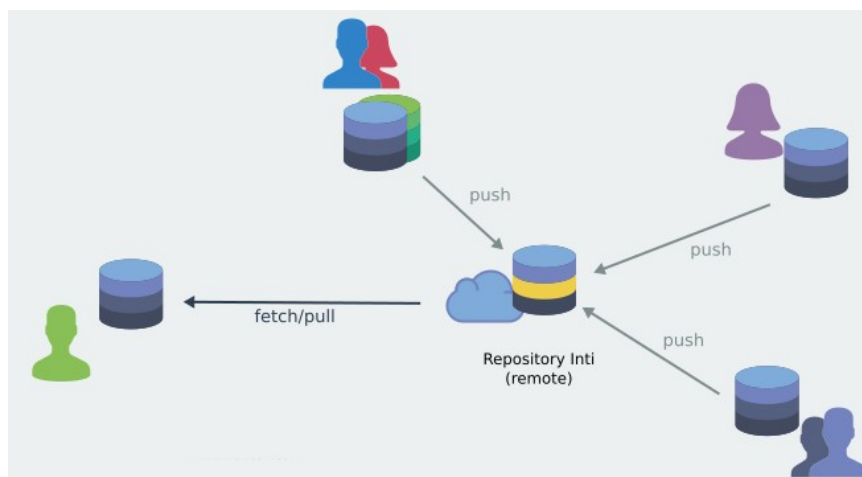




Melihat perbedaan di Github

5.8.5. Mengambil Revisi dan Remote Repository

Saat kita bekerja dengan repository yang memiliki banyak kontributor, kita seharusnya mengambil dulu revisi terbaru dari repository inti agar tidak bentrok. Misalnya begini, pada repository remote ada kontributor lain yang sudah menambahkan dan merubah sesuatu di sana. Maka kita harus mengambil perubahan tersebut, agar repository lokal kita tetap ter-update atau sama persis seperti repository remote.



Mengambil revisi

Ada dua perintah untuk mengambil revisi dari repository remote:

```
git fetch [nama remote] [nama branch]
```

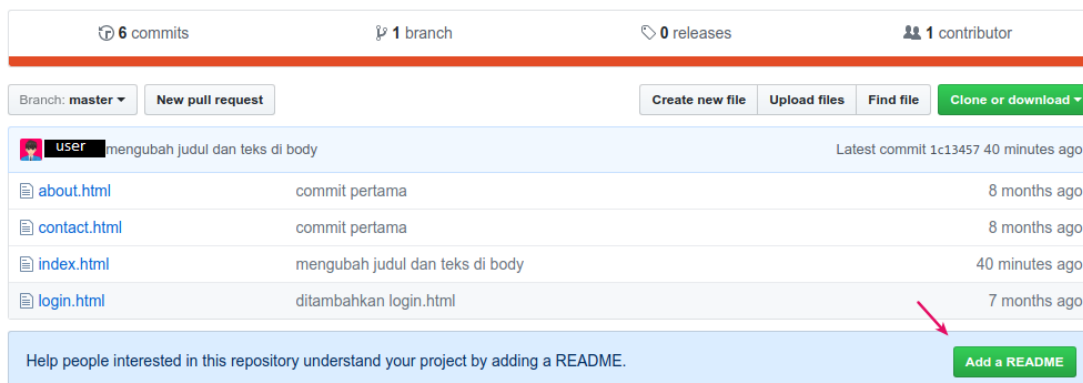
```
git pull [nama remote] [nama branch]
```

Perbedaan pada keduanya adalah perintah git fetch hanya akan mengambil revisi (commit) saja dan tidak langsung melakukan penggabungan (merge) terhadap repository lokal. Sedangkan git pull akan mengambil revisi (commit) dan langsung melakukan penggabungan (merge) terhadap repository lokal.

Pemakaian salah satu keduanya tergantung dari situasi dan kondisi. Bila kita sudah membuat perubahan di repository lokal, maka sebaiknya menggunakan git fetch agar perubahan yang kita lakukan tidak hilang. Namun, bila kita tidak pernah melakukan perubahan apapun dan ingin mengambil versi terakhir dari repository remote, maka gunakanlah git pull.

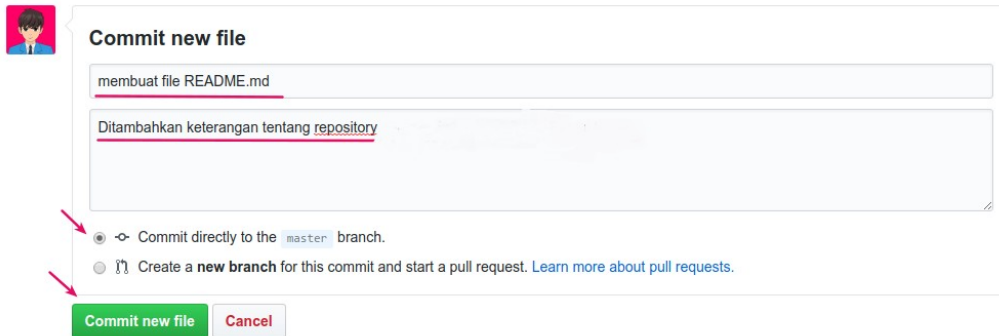
5.8.6. Mengambil Revisi dengan Git Fetch

Sekarang mari kita coba praktekkan. Silahkan buka github, dan tambahkan file README.md melalui Github. Klik tombol add README.



Mengambil revisi dengan git fetch

Setelah itu, isilah file README.md dengan apapun yang Anda inginkan. Setelah selesai, simpan perubahan dengan melakukan commit langsung dari Github.



Membuat file readme

Pesan commit bersifat opsional, boleh di isi boleh tidak. Karena Github akan membuatnya secara otomatis. Sekarang ada perubahan baru di repository remote dan kita akan mengambil perubahan tersebut. Mari kita lakukan dengan perintah *git fetch*.

```
i ~/project-01 $ git fetch github master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:petanikode/belajar-git
* branch          master      -> FETCH HEAD
  1c13457..3951f15  master      -> github/master
i ~/project-01 $ ls
about.html  contact.html  index.html  login.html
```

Melakukan git fetch

Revisi sudah diambil, tapi belum ada file README.md di dalam repository lokal. Kenapa bisa begitu? Ya, balik lagi dari pengertian *git fetch*. Dia hanya bertugas mengambil revisi saja dan tidak langsung menggabungkannya dengan repository lokal. Coba kita cek dengan *git log*.

```
i ~/project-01 $ git log --oneline
1c13457 mengubah judul dan teks di body
865e50c commit hasil revert
023b078 ditambahkan login.html
6cdfdbb ada perubahan
06f735a ditambahkan isi
cf08ca0 commit pertama

i ~/project-01 $ git log github/master --oneline
3951f15 membuat file README.md
1c13457 mengubah judul dan teks di body
865e50c commit hasil revert
023b078 ditambahkan login.html
6cdfdbb ada perubahan
06f735a ditambahkan isi
cf08ca0 commit pertama
```

Pengecekan dengan git log



Pada gambar di atas terlihat perbedaan log antara repository lokal dengan repository remote. Bila ingin mengecek apa saja perbedaannya, coba gunakan perintah git diff.

```
git diff master github/master
```

Keterangan: master adalah branch master di repository lokal, github/master adalah branch master di repository remote. Hasil outputnya kira-kira akan seperti ini:

```
diff --git a/README.md b/README.md
new file mode 100644
index 0000000..1174eb2
--- /dev/null
+++ b/README.md
@@ -0,0 +1,18 @@
+# belajar-git
+
+Repository ini adalah repository untuk belajar Git.
```

Lalu sekarang bagaimana cara kita menggabungkan commit dari repository remote dengan lokal? Gunakan perintah git merge.

```
git merge master github/master
```

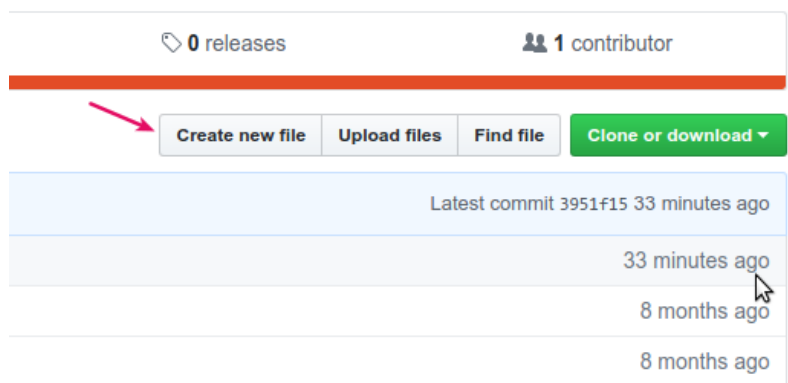
Setelah itu coba ketik ls dan git log lagi, maka kita sudah berhasil menggabungkan revisi dari remote dan lokal.

```
i ~/project-01 $ git merge master github/master
Updating 1c13457..3951f15
Fast-forward
 README.md | 18 ++++++
 1 file changed, 18 insertions(+)
 create mode 100644 README.md
i ~/project-01 $ ls
about.html contact.html index.html login.html README.md
i ~/project-01 $ git log --oneline
3951f15 membuat file README.md
1c13457 mengubah judul dan teks di body
865e50c commit hasil revert
023b078 ditambahkan login.html
6cdfdbb ada perubahan
06f735a ditambahkan isi
cf08ca0 commit pertama
```

Melihat hasil gabungan revisi remote dan lokal

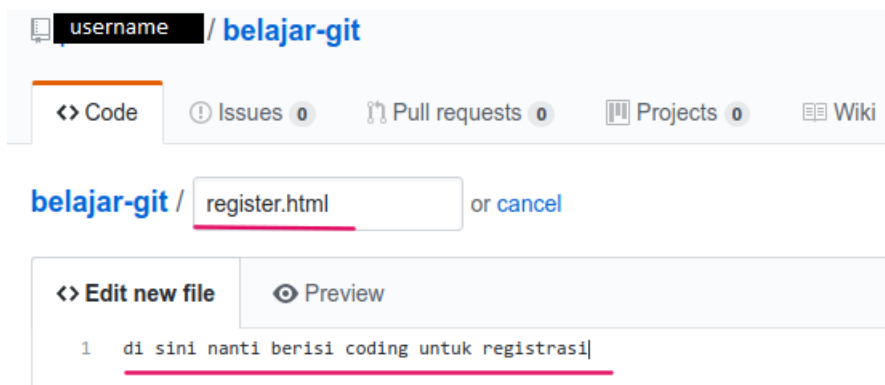
5.8.7. Mengambil Revisi dengan Git Pull

Lakukan hal yang sama seperti tadi. Kali ini kita akan membuat file baru bernama register.html melalui Github.



Membuat file baru

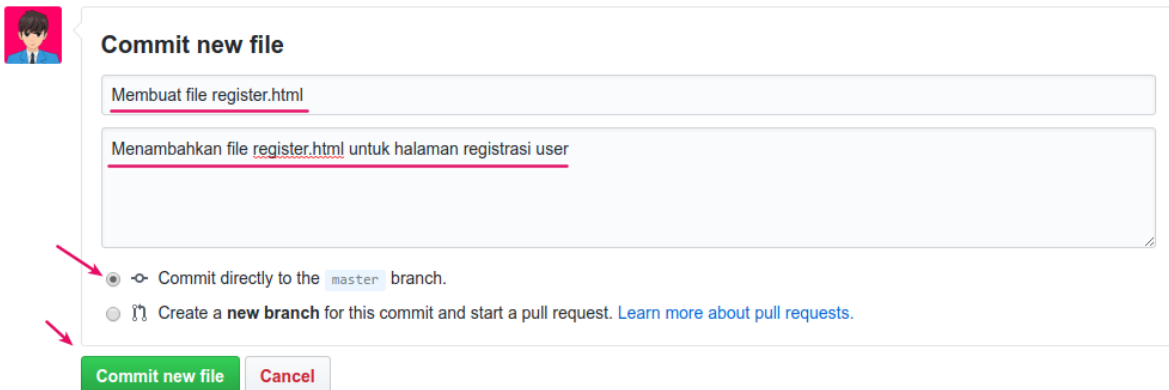
Berikan nama file dengan register.html dan isi dengan apa saja.



Mengisi file html



Simpan revisi dan tambahkan commit seperti ini.



Melakukan commit file baru

Sekarang ada perubahan baru di repository remote dan kita akan mengambilnya dengan perintah git pull. Silahkan buka repository lokal dan ketik perintah berikut:

```
git pull github master
```

Maka semua revisi akan diambil dan langsung digabungkan (merge).

```
i ~/project-01 $ git pull github master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:petanikode/belajar-git
* branch          master       -> FETCH HEAD
  3951f15..e4dbc35 master       -> github/master
Updating 3951f15..e4dbc35
Fast-forward
 register.html | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 register.html
i ~/project-01 $ ls
about.html  contact.html  index.html  login.html  README.md  register.html
i ~/project-01 $ git log --oneline
e4dbc35 Membuat file register.html
3951f15 membuat file README.md
1c13457 mengubah judul dan teks di body
865e50c commit hasil revert
023b078 ditambahkan login.html
6cdfdbb ada perubahan
06f735a ditambahkan isi
cf08ca0 commit pertama
```

Pengecekan hasil git pull

5.8.8. Clone Remote Repository

Clone repository bisa kita bilang seperti copy repository dari remote ke lokal. Perintah untuk melakukan clone adalah git clone.

```
git clone https://github.com/username/belajar-git.git [nama dir]
```



Keterangan: `https://...` adalah URL repository remote, kita juga bisa menggunakan SSH. `[nama dir]` (opsional) adalah nama direktori yang akan dibuat. Jika kita tidak berikan nama direktori, maka akan otomatis menggunakan nama repository. Mari kita coba. Sekarang kita akan pindah ke direktori Desktop.

```
Cd ~/Desktop
```

Lalu clone remote repo:

```
git clone git@github.com:username/belajar-git.git
```

Maka akan ada direktori baru di sana.

FYI: Saat Anda clone sebuah repository dari Github, nama remote origin akan diberikan secara otomatis.

5.8.9. Exercise

1. Buat sebuah akun GitHub.
2. Push repository local ke github
3. Buat clone dari repository yang sudah di push ke GitHub atau GitLab

5.9. Studi Kasus

5.9.1. Study Case Git 1

Pada sesi kali ini kita akan memanfaatkan layanan github sebagai tempat source code yang telah kita buat. Kita akan melakukan perubahan file dan menambahkan perubahan file tersebut ke github. Berikut ini langkah-langkah dalam menambahkan file baru ke github.

- Buka <https://github.com/>
- Jika anda belum mempunyai akun github, maka buatlah akun baru. Jika sudah ada masuk ke akun github anda.
- Pada tab Repositories, pilih New > Create repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner

Repository name *

W citra8106 ▾

/ demo ✓

Great repository names are short and memorable. Need inspiration? How about [automatic-octo-carnival?](#)

Description (optional)

Public

Anyone can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ Initialize this repository with a README

This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾

Add a license: None ▾ ⓘ

Create repository

Membuat repository baru github

- Setelah commit file, kita akan melakukan upload file-file yang telah kita commit ke github, pada repository demo yang telah kita buat sebelumnya pada github. Salin repository address pada github. Alamat repository ini yang akan kita jadikan sebagai remote address proyek yang kita buat.

```

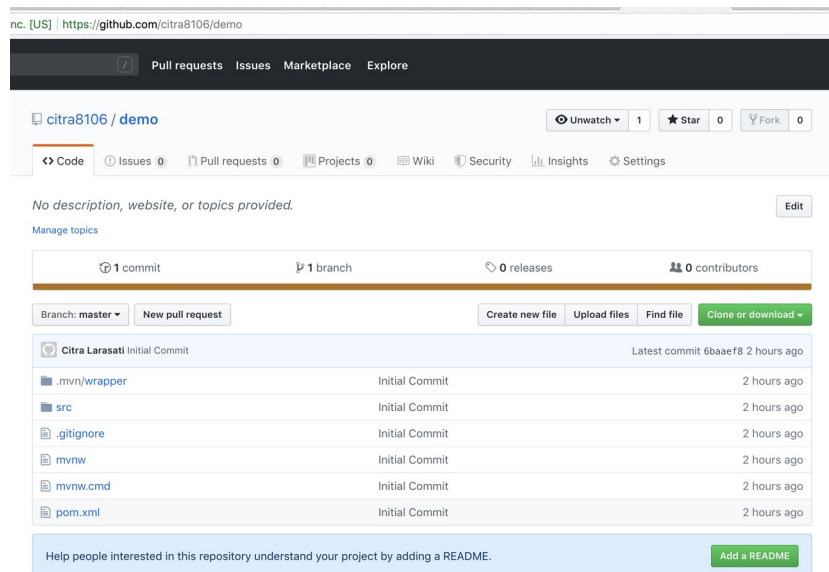
→ demo git:(master) git remote add origin https://github.com/citra8106/demo.git
→ demo git:(master) git push -u origin master
Username for 'https://github.com': citra8106
Password for 'https://citra8106@github.com':
Enumerating objects: 26, done.
Counting objects: 100% (26/26), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (16/16), done.
Writing objects: 100% (26/26), 49.60 KiB | 2.75 MiB/s, done.
Total 26 (delta 0), reused 0 (delta 0)
To https://github.com/citra8106/demo.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

```

Upload folder ke github



- Inisial proyek telah berhasil masuk ke github.



Repositori demo pada github

5.9.2. Study Case Git 2

Pada sesi kali ini kita akan membuat branch baru dari master. Branch baru yang kita buat bernama "setup_database". Kemudian branch baru ini akan kita push ke github. Sehingga pada repository github kita mempunyai 2 branch. Branch utama yaitu master, dan branch kedua yaitu setup_database. Berikut ini langkah-langkah dalam membuat branch baru.

1. Membuat branch baru dan checkout ke branch tersebut dengan perintah gco -b.

```
→ demo git:(master) gco -b setup_database
Switched to a new branch 'setup_database'
→ demo git:(setup_database)
```

Checkout ke branch setup_database

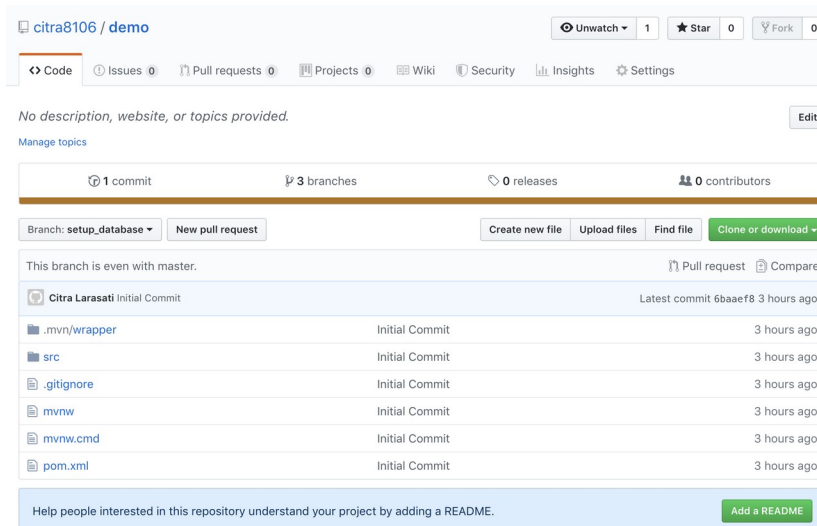
2. Push branch setup_database ke github.

```
→ demo git:(setup_database) git push --set-upstream origin setup_database
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'setup_database' on GitHub by visiting:
remote:   https://github.com/citra8106/demo/pull/new/setup_database
remote:
To https://github.com/citra8106/demo.git
 * [new branch]      setup_database -> setup_database
Branch 'setup_database' set up to track remote branch 'setup_database' from 'origin'.
```

Push branch setup_database ke github



3. Tampilan branch setup_database pada github



Tampilan branch setup_database pada github

5.9.3. Study Case Git 3

Pada sesi kali ini kita akan melakukan penggabungan kode (*merge*) dari branch setup_database ke branch master. Berikut ini langkah-langkah dalam menggabungkan branch setup_database ke master.

1. Tambahkan kode berikut ini pada application.properties untuk konfigurasi database postgresql. Jangan lupa untuk membuat database terlebih dahulu. Pada contoh kali ini, database yang digunakan yaitu “basic_java”;

application.properties

```
## Spring Data Source
spring.datasource.url=jdbc:postgresql://localhost:5432/basic_java

# The SQL dialect makes Hibernate generate better SQL for the chosen
database
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL
Dialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto=create-drop
```



```
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true

spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
```

2. Commit perubahan yang dilakukan

```
→ demo git:(setup_database) git status
On branch setup_database
Your branch is up to date with 'origin/setup_database'.

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)

       modified:   src/main/resources/application.properties

no changes added to commit (use "git add" and/or "git commit -a")
→ demo git:(setup_database) x ga .
→ demo git:(setup_database) x gc -m "add configuration for postgresql database"
[setup_database 6143be0] add configuration for postgresql database
1 file changed, 11 insertions(+)
```

Commit konfigurasi postgresql

3. Checkout ke branch master, kemudian merge branch setup_database ke master.

```
→ demo git:(setup_database) gco master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
→ demo git:(master) git merge setup_database
Updating 6baaef8..6143be0
Fast-forward
 src/main/resources/application.properties | 11 ++++++++
 1 file changed, 11 insertions(+)
```

Merge branch setup_database ke master

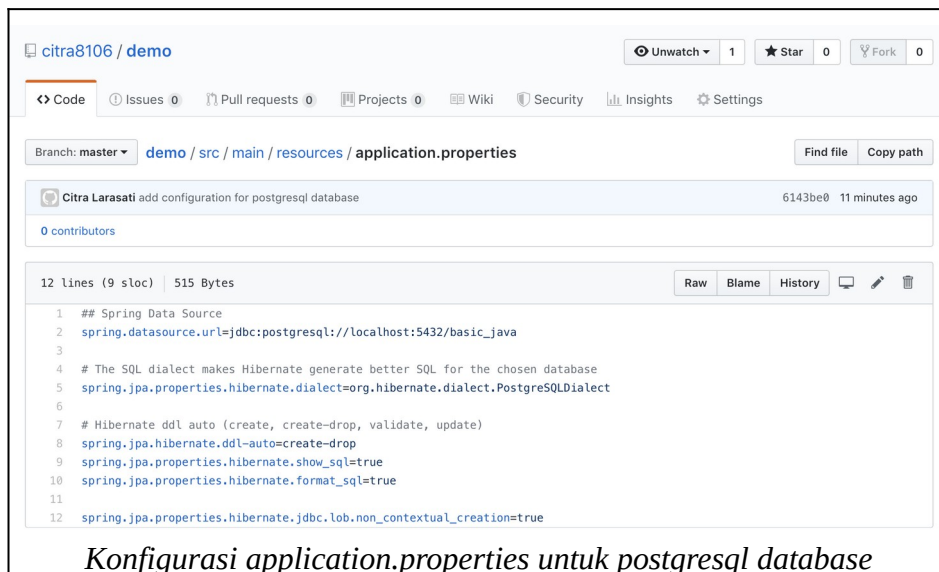
4. Push perubahan ke github

```
→ demo git:(master) git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 740 bytes | 370.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/citra8106/demo.git
 6baaef8..6143be0 master -> master
```

Push perubahan ke github



5. Konfigurasi postgresql dapat dilihat di github



The screenshot shows a GitHub repository for 'citra8106 / demo'. The file 'application.properties' is selected, showing its content. The file was added by Citra Larasati 11 minutes ago. The code content is as follows:

```

1  ## Spring Data Source
2  spring.datasource.url=jdbc:postgresql://localhost:5432/basic_java
3
4  # The SQL dialect makes Hibernate generate better SQL for the chosen database
5  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
6
7  # Hibernate ddl auto (create, create-drop, validate, update)
8  spring.jpa.hibernate.ddl-auto=create-drop
9  spring.jpa.properties.hibernate.show_sql=true
10 spring.jpa.properties.hibernate.format_sql=true
11
12 spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
  
```

Konfigurasi application.properties untuk postgresql database

5.9.4. Study Case Git 4

Tugas

Kita dapat mendapatkan proyek baru dari github ke repository local kita dengan perintah git pull maupun git fetch. Anda bisa mencoba penggunaan **git pull** dan **git fetch** dan cari tahu perbedaannya.

5.9.5. Study Case Git 5

Tugas

Kita dapat melakukan pengecualian terhadap beberapa file agar tidak disimpan ke git. Salah satu cara yaitu dengan menambahkan file ".gitignore" pada repository yang kita buat. Anda bisa mencoba exclude beberapa file dengan menambahkan pada ".gitignore".



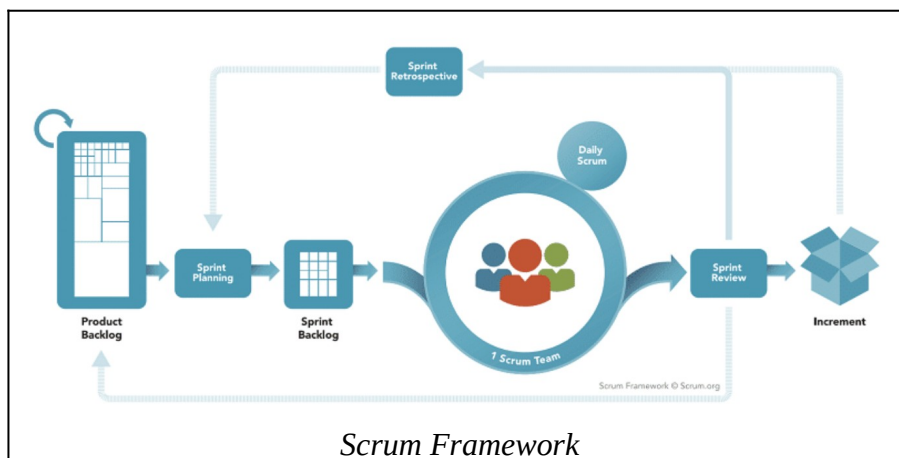
5.9.6. Study Case Git 6

Tugas

Ketika kita melakukan kesalahan dalam pesan commit, kita bisa melakukan perubahan pada commit message. Anda bisa mencoba menggunakan perintah **git commit -amend**.

5.10. Flow Development Di Dalam Sebuah Tim

Dalam metode scrum terdapat ceremony/kegiatan yang dilakukan, diantaranya adalah Sprint Planning, Daily Meeting, Sprint Review, dan Sprint Retrospective. Sprint adalah proses 1 kali iterasi scrum atau sebuah batasan waktu untuk pengembangan produk yang siap digunakan oleh pengguna. Waktu dari Sprint biasanya 2-4 minggu tergantung dari proyek pengembangan itu sendiri.



Sprint dimulai dari Sprint Planning, dimana tim menentukan product backlog mana yang ingin dikerjakan oleh tim selama 1 sprint. Selama sprint berjalan ada kegiatan Daily Meeting dimana kegiatan ini melakukan meeting selama 15 menit setiap hari untuk menyampaikan progress yang sudah dikerjakan serta komunikasi antar anggota jika terjadi perubahan. Diakhir sprint terdapat kegiatan Sprint Review & Sprint Retrospective. Diakhir sprint, Scrum team akan melaporkan dan mempresentasikan produk yang sudah dibuat selama 1 sprint berjalan kepada PO, apakah produk yang dibuat sudah sesuai atau belum. Setelah sprint review selesai, tim akan melakukan sprint retrospective untuk menilai anggota satu sama

lain dan melihat kelebihan yang perlu dipertahan serta kekurangan yang perlu diperbaiki dalam tim.

5.10.1. Mengetahui Local – Staging – Production

Dalam pengembangan Scrum, terdapat 3 fase yaitu local, staging, dan production.

- Local

Local atau development merupakan fase pengembangan awal, dimana dilakukan implementasi kode awal untuk pondasi sebuah proyek dan menggunakan data seeding yang masih bersifat dummy.

- Staging

Pada fase ini, kode yang sudah dikerjakan dalam fase Development dikumpulkan. Namun sebelum dikumpulkan harus sudah melewati proses code review oleh Scrum Master. Kode yang dikumpulkan sudah bersifat deliverables (kemungkinan produk) tetapi masih ada kemungkinan bug yang tinggi. Data seeding pada proses ini sudah mulai menyesuaikan kondisi real di lapangan

- Production

Pada fase ini, kode yang sudah melalui tahap code review oleh Product Owner, testing oleh calon pengguna, disimpan. Pada tahap ini, aplikasi yang sudah siap untuk dipasarkan dan digunakan oleh end user dihasilkan. Data seeding dalam fase ini sudah berbentuk data asli milik user.

5.10.2. Tugas

Lakukanlah skenario git merge & forking antara local, staging, production!

5.11. Mengetahui Strategi Deployment Aplikasi

Sebagai seorang backend developer, kita perlu mengetahui beberapa strategi untuk melakukan proses deployment. Berikut adalah beberapa contoh strategi deployment aplikasi yang biasanya ada di perusahaan.

5.11.1. Big-Bang Deployment Strategy

Yang pertama adalah, hal yang paling mudah dilakukan. Konsepnya sangat simple. Big-Bang strategy, atau sering juga disebut seperti Replace deployment, atau Recreate, dsb. Yang dimana sifatnya menimpa dan mengganti (mereplace) aplikasi yang aktif secara langsung.

Jika dianalogikan, konsep ini bisa saya buat kedalam contoh seperti ketika kita hendak mengganti ban sepeda motor.

Yang kita lakukan adalah, membukanya lalu langsung menggantinya. Sehingga jika terjadi kesalahan, baik ketika mengganti ban, atau setelah diganti bannya malah bocor, maka motor tersebut tidak akan bisa jalan. Karena sifatnya *all or nothing*, sehingga sangat beresiko membuat motor tidak dapat dijalankan.

Sama seperti ketika deployment aplikasi, konsep seperti ini juga sangat berbahaya. Dimana kita langsung mengganti(*replace*) aplikasi kita tanpa memikirkan kemungkinan gagal. Jika terjadi kegagalan *deployment* atau bugs sesudah *deployment*, maka selesai lah sudah, sistem kita akan *down* dalam waktu yang tidak ditentukan sampai kita berhasil *rollback*, yang juga makan waktu lama.

Ciri-ciri dari *deployment* ini adalah,

- Aplikasi yang akan dideploy biasanya sudah dibuat dalam satu *package*, yang nantinya diupload ke server.
- Package aplikasi yang diupload akan diekstrak dan dijalankan sesuai jenis aplikasi, jika *compiled* maka *binary* hasil *compiled*nya dijalankan di server, jika *intrepreted* seperti PHP, Phyton, maka *source codenya* di *copy* dan dijalankan di Webserver.
- Orang yang menggunakan metode ini, biasanya sangat optimis aplikasinya tidak akan *down/error*.
- Dan jika terjadi down, biasanya terjadi *chaos* dan kepanikan luar biasa, sambil tangan gemeteran untuk re-*deploy* versi lama. Detak jantung kencang dan tak karuan. Dunia serasa gelap, muka pucat pasi, dan berusaha menenangkan diri demi menjauhkan kesalahan-

kesalahan kecil. Dan berusaha membenarkan diri, saya udah percaya diri sama aplikasi ini tetapi kok masih bisa *down*?

- Rata-rata *downtime* cukup lama.

5.11.1.1. **Kelebihan:**

- Mudah di-implementasikan. Cara klasik, tinggal replace.
- Perubahan kepada sistem langsung 100% secara instan.

5.11.1.2. **Kekurangan**

- Terlalu beresiko, rata-rata *downtime* cukup lama.

Karena hal tersebut, strategi *deployment* seperti ini baiknya dihindari. Hal ini bisa dilakukan untuk server-server staging dan server kebutuhan development. Yang tidak berpengaruh pada bisnis perusahaan.

5.11.2. **Rollout Deployment Strategy**

Nah selanjutnya ada lagi yang biasa disebut Rollout deployment. Konsepnya berbeda dari Big Bang deployment. Konsep ini lebih aman dari Big Bang deployment.

Rollout deployment jika dianalogikan seperti sebuah bus besar yang memiliki 8 ban(roda) dibelakang. Kiri dan kanan masing-masing ada 4 roda.

Jika di suatu saat kita hendak mengganti ban, maka kita dapat menggantinya satu-persatu dahulu. Ganti 1 ban, lalu test berjalan, jika oke ganti lagi ban selanjutnya. Jika bannya ternyata kempes, mobil masih bisa berjalan, meski tidak optimal, tetapi tidak akan menyebabkan mobil tersebut berhenti total. Dan bisa di *rollback* kembali, diganti kembali ke ban yang lama.

Sama halnya ketika *deployment* aplikasi. Dengan metode ini, kita melakukan deployment secara bertahap per-server yang hidup. Dan jika satu server saja langsung error, kita dapat langsung *rollback* tanpa melanjutkan *deploy* kesemua server.



5.11.2.1. Kelebihan:

- Lebih aman dan less *downtime* dari versi sebelumnya

5.11.2.2. Kekurangan

- Akan ada 2 versi aplikasi berjalan secara barengan sampai semua server terdeploy, dan bisa membuat bingung. Seperti kita ketahui, untuk *management versioning* itu sedikit susah dan cukup buat pusing kepala
- Karena sifatnya perlahan satu persatu, untuk *deployment* dan *rollback* lebih lama dari yang Bigbang, karena prosesnya perlahan-lahan sampai semua server terkena efeknya.
- Tidak ada kontrol *request*. Server yang baru ke-*deploy* dengan aplikasi versi baru, langsung mendapat *request* yang sama banyaknya dengan server yang lain. Sehingga jika terjadi *error*, juga dapat menyebabkan kerugian besar.

5.11.3. Blue/Green Deployment Strategy

Selanjutnya adalah *Blue/Green Deployment*. *Deployment* jenis ini juga cukup sering digunakan dalam dunia *Software Engineering*.

Konsepnya cukup sederhana, pertama-tama, kita akan membuat satu environment yang serupa dengan yang sedang aktif/live, kemudian kita pun melakukan switching request ke environment baru tersebut.

5.11.3.1. *Kelebihan:*

- Perubahan sangat cepat, sekali *switch service* langsung berubah 100%.
- Tidak ada issue beda versi pada *service* seperti yang terjadi pada Rollout Deployment.

5.11.3.2. *Kekurangan:*

- *Resource* yang dibutuhkan lebih banyak. Karena untuk setiap *deployment* kita harus menyediakan *service* yang serupa *environmentnya* dengan yang sedang berjalan di *production*.
- Testing harus benar-benar sangat diprioritaskan sebelum di *switch*, aplikasi harus kita pastikan aman dari *request* yang tiba-tiba banyak.

Meski sedikit lebih mahal dari sisi *resources*, strategi ini bisa disebut strategi yang paling cukup disarankan dan banyak digunakan oleh orang-orang saat ini.

5.11.4. A/B Deployment Strategy

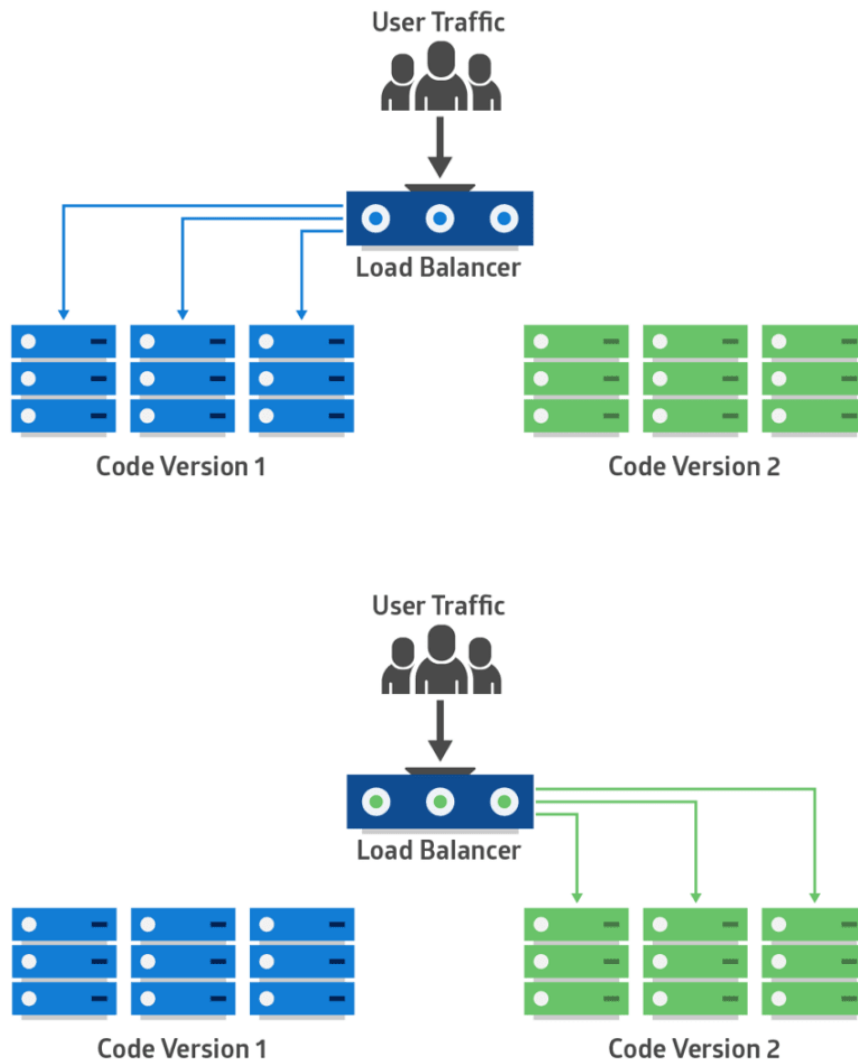
Selanjutnya adalah strategi A/B, atau biasa juga disebut A/B testing. Strategi ini biasanya lebih fokus pada user experience dan layout(UX/UI).

Biasanya A/B deployment lebih ke user sentris. Setengah user akan menerima fitur/layout A dan setengah lagi mendapat fitur/layout B, sehingga setiap user bisa mendapatkan tampilan yang beda.

Pengelompokan fitur/layout tersebut biasanya dibuat berdasarkan spesifikasi yang ada pada user, seperti lokasi, gender, dsb.

Meski gambar berikut tidak terlalu menunjukkan bagaimana A/B test berjalan (karena A/B test biasanya dilakukan untuk device yang sejenis (desktop vs desktop, mobile vs mobile).

Tetapi gambar dibawah menjelaskan bagaimana server melakukan strategi A/B deployment untuk mendukung proses A/B testing.



5.11.4.1. **Kelebihan:**

- *Traffic request* terkontrol dan terarah pada masing-masing server

5.11.4.2. **Kekurangan:**

- Susah di implementasikan, karena sangat user sentris.

- *Troubleshooting* jika terjadi *error* sedikit susah jika *logging* nya terpusat pada satu tempat. Sehingga untuk kasus A/B testing, setiap server harus memiliki mekanisme dan *storage logging* tersendiri untuk mempermudah *tracing* dan *troubleshooting* jika terjadi error.

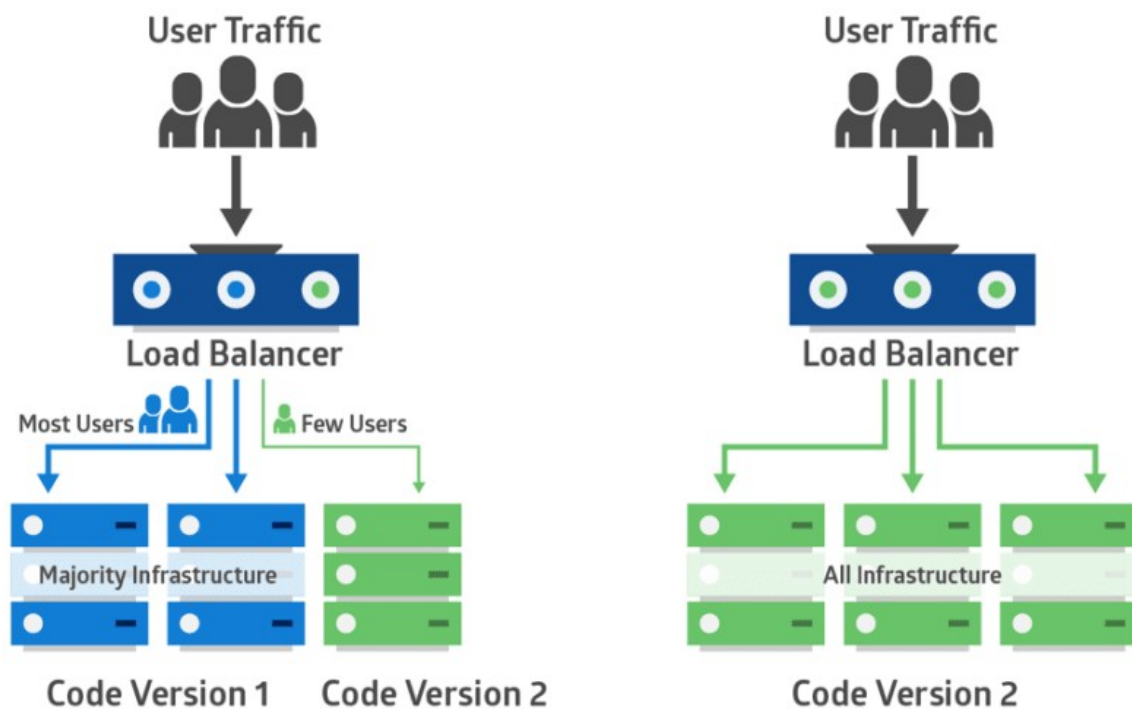
Tujuan dilakukannya A/B test adalah untuk evaluasi dan pengambilan data yang diperuntukan dalam *decision making* terhadap fitur yang akan direlease nantinya.

5.11.5. Canary Deployment Strategy

Nah, berikutnya adalah strategi Canary Deployment. Strategi ini lebih *advance* dari semua metode release tersebut diatas. Prinsip kerjanya mirip seperti Rollout Deployment, tetapi bedanya, jika pada Rollout Deployment, ketika aplikasi di deploy pada satu server, maka server tersebut akan langsung kebagian *request* dari user sama rata dengan server lainnya.

Nah pada canary, aplikasi juga akan di *deploy* pada satu server, tetapi bedanya adalah, *request* yang masuk kedalamnya akan dikontrol dan ditahan berdasarkan kebutuhan.

Konsepnya juga mirip seperti A/B Deployment, bedanya A/B *deployment request* user langsung dibagi dua, 50% ke fitur A, 50% ke fitur B.



Namun di Canary Deployment, tetap menggunakan *percentage request user*, tetapi lebih dikontrol. Semisal ketika selesai dideploy kesalah satu server, pertama kali *request* yang diterima adalah sebanyak 10% dari total *request* yang diterima *load balancer*, sementara 90% lagi dialihkan ke server lain yang masih versi lama.

Jika ternyata cukup ok, naikan lagi menjadi 20%, 30% sampai semua *request* yang diterima sama rata ke setiap server, sampai selanjutnya semua versi baru terdeploy dengan sempurna.

Nah jika terjadi kegagalan, *bug* atau *error*, semisal pada tahap 10%, maka semua request 100% akan diarahkan kembali ke server yang lama. Lalu yang versi baru akan diberhentikan.

5.11.5.1. **Kelebihan:**

- Cukup aman
- Mudah untuk *rollback* jika terjadi error/bug, tanpa berimbas kesemua user

5.11.5.2. **Kekurangan:**

- Untuk mencapai 100% cukup lama dibanding dengan Blue/Green deployment. Dengan Blue/Green deployment, aplikasi langsung 100% ter*deploy* keseluruhan user.

Konsep ini tentunya lebih aman dan *fast-rollback* jika terjadi kegagalan. Jika diibaratkan dalam kehidupan sehari-hari, konsep ini mirip seperti keran air, kita dapat secara perlahan membuka jumlah air yang akan dikeluarkan dari keran tersebut. Sama halnya pada Canary Deployment, kita juga dapat mengontrol jumlah request yang diterima aplikasi yang versi baru sampai aplikasi tersebut stabil berjalan.