

# Git dan GitHub

---

Dalam survei keterampilan alumni PUB di dunia kerja, Git dan GitHub menempati posisi kedua teratas (setelah basis data SQL) berdasarkan suara yang diperoleh. Hal ini menunjukkan betapa pentingnya kedua teknologi ini di dunia kerja.

Selain itu, Git dan GitHub memang digunakan di hampir setiap project software development baik oleh komunitas open source maupun perusahaan teknologi di seluruh dunia.

Dokumen ini sengaja ditulis dalam format Markdown agar dapat ditampilkan di [PUB Camp](#) (bagian dari project [PUB Next](#)).

© 2023 **Divisi Pendidikan PUB**

# Silabus

---

Pertemuan	Topik	Materi
1	Version Control dan Git	Mengenal version control
		Instalasi Git
		Penyiapan Git
2	Dasar-Dasar Git	Repository
		Commit
		Remote
3	GitHub dan Markdown	Mengenal GitHub
		Fetch dan push
		Sintaks Markdown
4	Branching dan Merging	Branching
		Merging
		Rebasing
5	Penggunaan Git dalam IDE	Visual Studio Code
		IntelliJ IDEA
		Visual Studio
6	Ujian Akhir	

# Daftar Isi

---

- **Git dan GitHub**
- Silabus
- Daftar Isi
- Version Control
  - Version control system (VCS)
    - VCS lokal
    - VCS terpusat (client–server)
    - VCS terdistribusi (peer-to-peer)
- Git
  - Kegunaan Git
  - Menginstal Git di Windows
  - Menyiapkan Git
    - Mengatur nama dan email
    - Konfigurasi tambahan
      - Mengatur editor default
      - Mengatur nama branch default
  - Dasar-dasar Git
    - Membuat repository baru
      - Referensi perintah
    - Membuat commit baru
      - Memeriksa status file
      - Menambahkan file ke staging area
      - Mengabaikan file
      - Menyimpan perubahan
      - Referensi perintah
    - Membatalkan sesuatu
      - Membatalkan penambahan ke staging area
      - Membatalkan commit terakhir
      - Referensi perintah

- Melihat riwayat perubahan
  - Referensi perintah
- Menggunakan remote
  - Mengkloning repository yang sudah ada
  - Menampilkan daftar remote
  - Menambahkan remote baru
  - Fetch dan pull dari remote
  - Referensi perintah
- GitHub
  - Menambahkan file README
  - Menghubungkan repository lokal ke GitHub
    - Mengubah nama branch utama (opsional)
    - Menambahkan remote repository
    - Mem-push ke remote repository
- Markdown
  - Menulis dokumen Markdown
  - Sintaks dasar
    - Judul
    - Gaya pada teks
    - List
      - Ordered list
      - Unordered list
    - Link
    - Gambar
    - Tabel
    - Blok source code
- Branch dalam Git
  - Membuat branch baru
  - Beralih ke branch lain
  - Merging
    - Merge conflict saat merging
  - Rebasing

- Merge conflict saat rebasing
- Referensi perintah

# Version Control

---

Version control adalah praktik pelacakan dan pengelolaan perubahan pada file dari waktu ke waktu.

Kebutuhan ini telah ada sejak adanya penulisan, tapi version control semakin penting sejak era komputasi dimulai, terutama digunakan dalam software development.

## Version control system (VCS)

Orang biasanya melakukan version control dengan cara menyalin file dengan nomor revisi atau stempel waktu. Cara ini sangat umum karena sangat sederhana, tetapi juga sangat rawan kesalahan, kita bisa saja salah mengubah file.

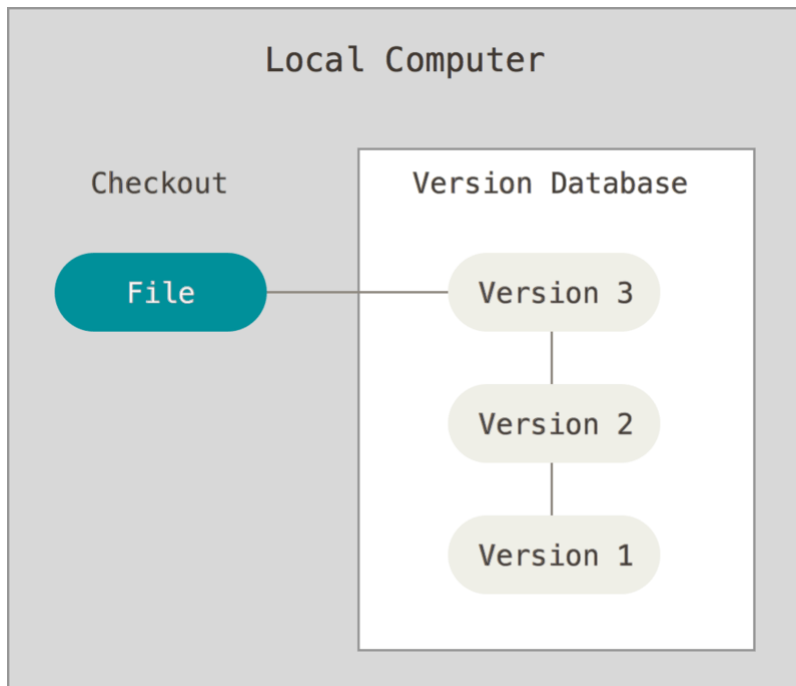
Untuk mengatasi masalah ini, programmer lama mengembangkan alat version control yang biasa disebut version control system (VCS).

Jenis-jenis VCS:

1. VCS lokal
2. VCS terpusat (client-server)
3. VCS terdistribusi (peer-to-peer)

### VCS lokal

VCS generasi pertama adalah VCS lokal yang memiliki basis data sederhana yang mampu menyimpan semua perubahan pada file.

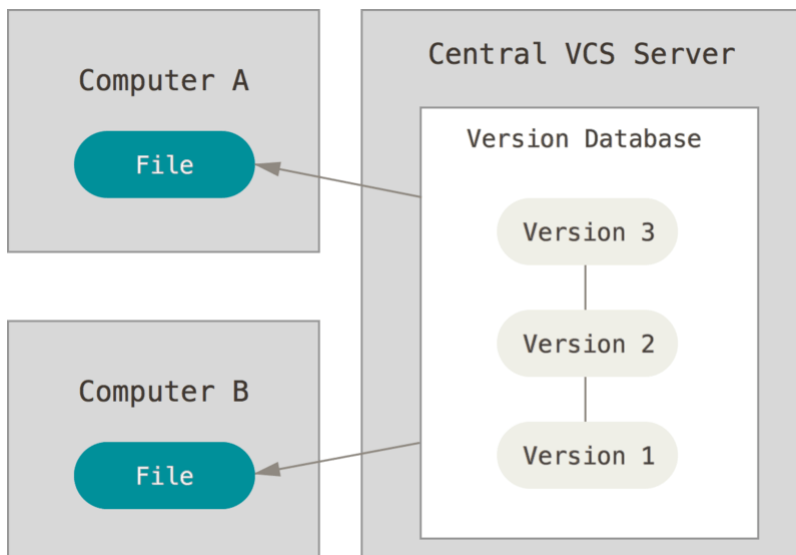


Contoh VCS lokal:

- RCS (1982)

### VCS terpusat (client-server)

Masalah besar berikutnya yang dihadapi orang adalah mereka perlu berkolaborasi dengan developer di sistem lain. Untuk mengatasi masalah ini, dikembangkanlah VCS terpusat.



Contoh VCS terpusat:

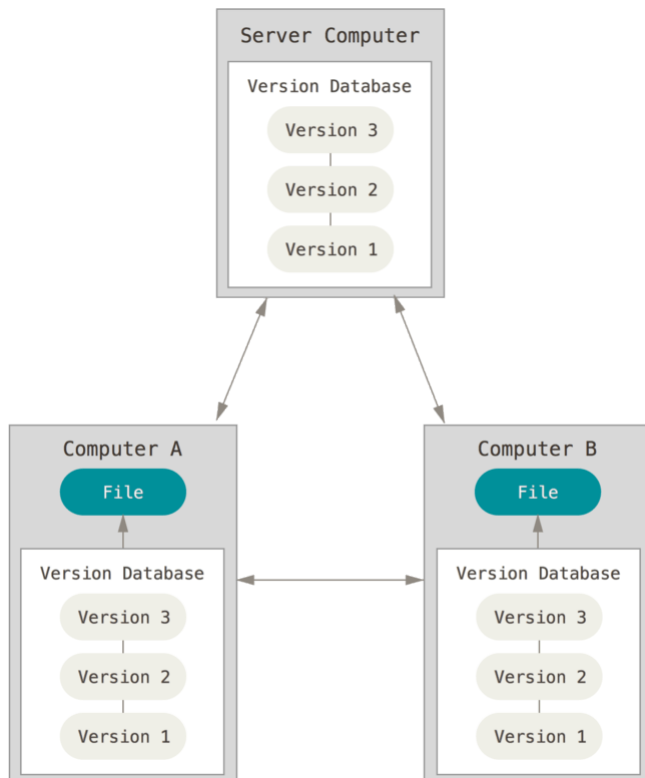
- CVS (1986)
- Subversion (2000)

Namun VCS terpusat memiliki beberapa kelemahan:

- jika server rusak, maka kita akan kehilangan seluruh project
- kontributor harus selalu terhubung ke server

## VCS terdistribusi (peer-to-peer)

Di VCS terdistribusi, setiap client memiliki salinan lengkap repository, yaitu file beserta riwayat perubahannya. Sehingga ketika server mengalami masalah dan menyebabkan data rusak/hilang, repository di salah satu client dapat disalin kembali ke server untuk memulihkannya.



Contoh VCS terdistribusi:

- Git (2005)
- Mercurial (2005)



# Git

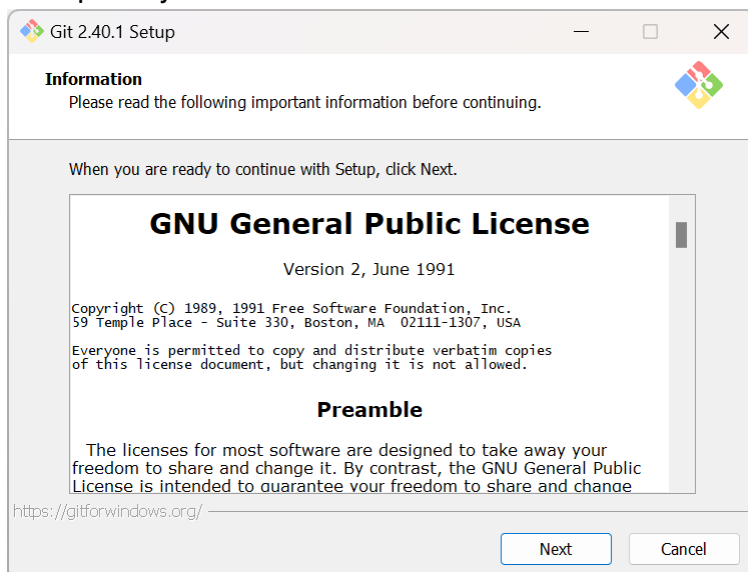
Git adalah VCS yang paling banyak digunakan saat ini. Ia adalah salah satu VCS terdistribusi yang gratis dan open source.

## Kegunaan Git

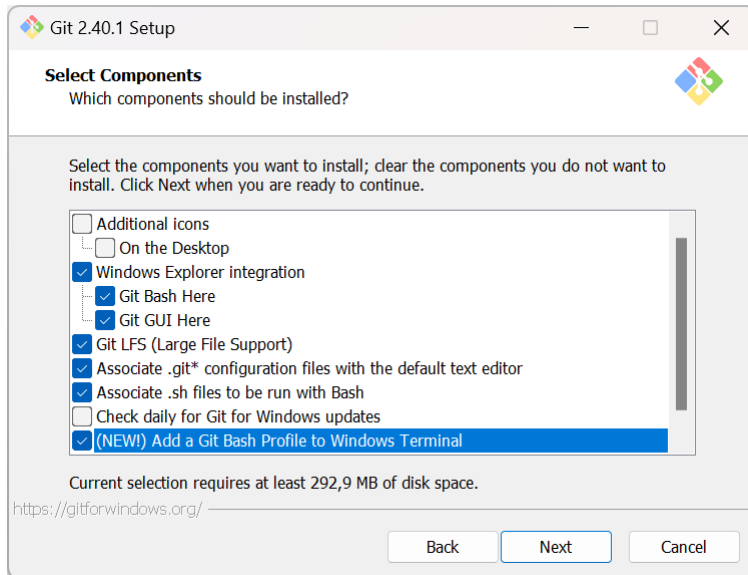
1. Pelacakan riwayat perubahan  
Dengan Git, kita dapat melihat siapa, kapan, dan mengapa suatu perubahan dilakukan. Git juga memudahkan kita untuk kembali ke versi sebelumnya.
2. Kolaborasi jarak jauh  
Git memudahkan kita dalam mengerjakan project bersama-sama secara jarak jauh tanpa saling menimpa perubahan satu sama lain.
3. Branching dan merging  
Dengan Git, kita dapat melakukan branching, yaitu membuat branch (cabang) untuk bereksperimen dengan fitur atau perbaikan baru tanpa memengaruhi branch utama, kemudian melakukan merging (penggabungan).

## Menginstal Git di Windows

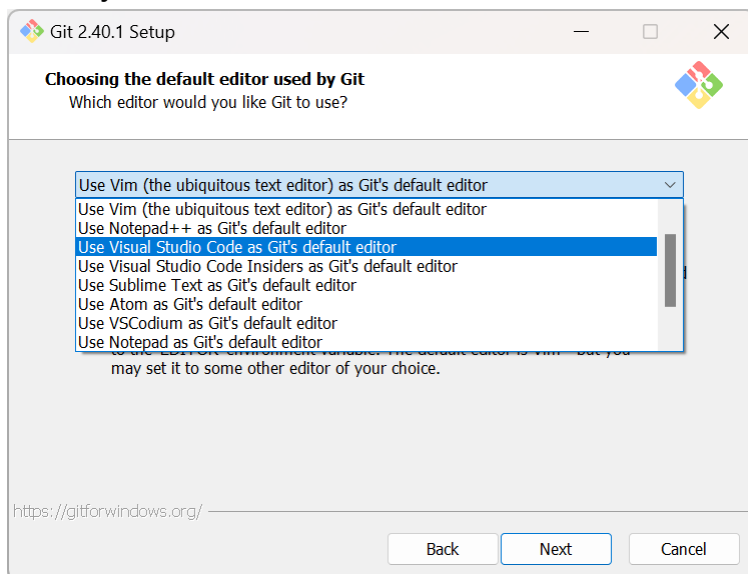
1. Download installer Git untuk Windows di situs web resmi Git: <https://git-scm.com/download/win>.
2. Jalankan installer yang sudah didownload.
3. Ikuti petunjuk di installer untuk memilih bahasa, folder instalasi, dll.



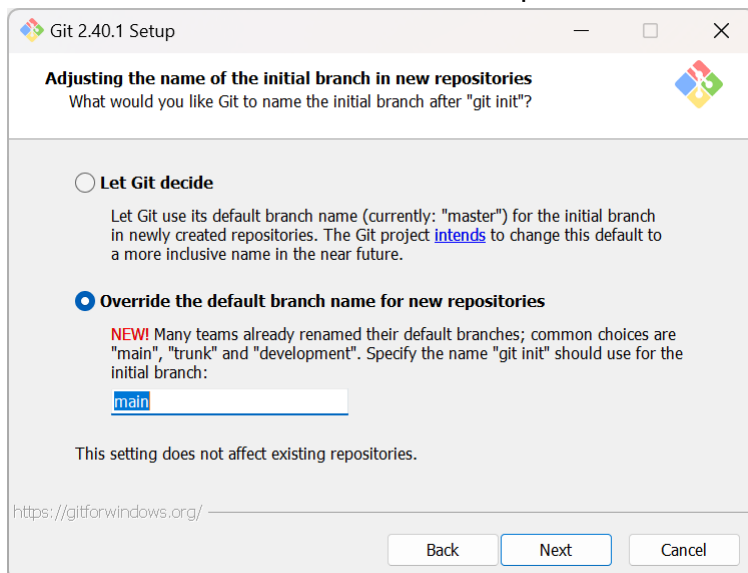
4. Di bagian "Select Components", centang komponen tambahan yang ingin diinstal, misalnya "Add a Git Bash Profile to Windows Terminal".



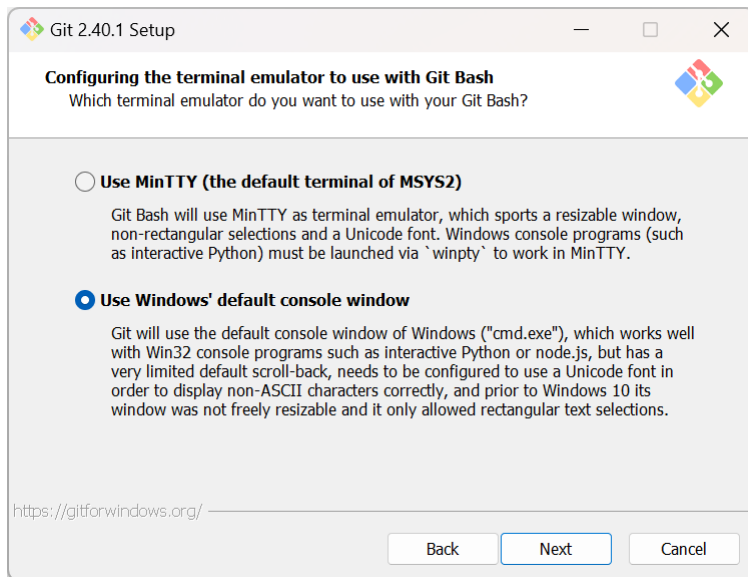
5. Di bagian "Choosing the default editor used by Git", pilih editor yang ingin digunakan, misalnya Visual Studio Code.



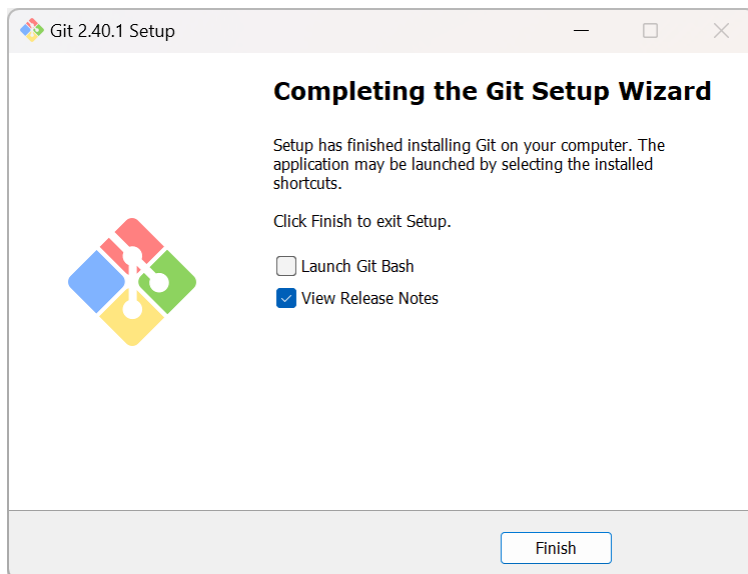
6. Di bagian "Adjusting the name of the initial branch in new repositories", pilih "Override the default branch name for new repositories", dan isi dengan "main".



7. Di bagian "Configuring the terminal emulator to use with Git Bash", jika kita menggunakan Windows 11 ke atas sebaiknya pilih "Use Windows' default console window".



8. Lanjutkan dengan mengikuti petunjuk installer dan tunggu proses instalasi sampai selesai.



## Menyiapkan Git

Git memiliki konfigurasi (pengaturan) yang terdiri dari variabel-variabel berisi nilai.

Konfigurasi tersebut dapat ditampilkan dengan perintah:

```
$ git config --list
```

Konfigurasi tersebut sebenarnya tersimpan di 3 tempat:

1. File `.../etc/gitconfig`, berlaku untuk seluruh sistem (semua pengguna), biasa disebut **konfigurasi sistem**
2. File `~/.gitconfig`, berlaku untuk pengguna saat ini, biasa disebut **konfigurasi global**
3. File `config` di folder Git (`.git/config`), berlaku untuk repository tunggal, biasa disebut **konfigurasi lokal**

Konfigurasi tersebut juga dapat ditampilkan beserta tempat asalnya (tersimpannya) dengan opsi `--show-origin`:

```
$ git config --list --show-origin
```

Variabel dapat ditampilkan nilainya dengan cara:

```
$ git config <key>
```

Variabel juga dapat dibuat atau diubah nilainya dengan cara:

```
$ git config <key> <value>
```

Variabel dapat dihapus dengan perintah `--unset`:

```
$ git config --unset <key>
```

- Untuk variabel di konfigurasi sistem, tambahkan opsi `--system`
- Untuk variabel di konfigurasi global, tambahkan opsi `--global`, biasanya digunakan untuk mengatur nama pengguna, email pengguna, editor default, dan nama branch default
- Untuk variabel di konfigurasi lokal, tambahkan opsi `--local`

## Mengatur nama dan email

Hal pertama yang harus kita lakukan setelah menginstal Git adalah mengatur nama dan email pengguna. Ini perlu kita lakukan sejak awal karena setiap commit Git menggunakan informasi ini, dan informasi ini **dimasukkan secara permanen dalam setiap commit berikutnya** yang kita buat.

```
$ git config --global user.name "Romi Kusuma Bakti"  
$ git config --global user.email romikusumab@gmail.com
```

\*Mengubah nama dan email menggunakan `git config` hanya akan memengaruhi commit berikutnya (yang akan datang) dan tidak akan mengubah nama dan email yang digunakan untuk commit sebelumnya.

\*GitHub menggunakan email (bukan nama) untuk mengaitkan commit dengan akun GitHub kita.

## Konfigurasi tambahan

Terdapat dua hal yang sebaiknya diatur terlebih dahulu jika terlewat selama instalasi:

- Editor default
- Nama branch default

### Mengatur editor default

Jika saat instalasi Git kita salah memilih editor default, kita dapat menggantinya sekarang di konfigurasi global.

Misalnya kita ingin menggunakan Visual Studio Code, maka lakukan perintah:

```
$ git config --global core.editor "code --wait"
```

### Mengatur nama branch default

Secara default, Git akan membuat branch bernama `master` saat kita membuat repository baru dengan `git init`. Namun biasanya orang lebih suka menggunakan `main`, dan GitHub juga menggunakan nama branch default `main` untuk setiap repository baru.

Sebagaimana direkomendasikan oleh GitHub, ubah nama branch dari `master` menjadi `main`.

Untuk mengatur nama branch default menjadi `main`, lakukan perintah:

```
$ git config --global init.defaultBranch main
```

## Dasar-dasar Git

### Membuat repository baru

Repository (biasa disingkat "repo") adalah kumpulan file beserta riwayat perubahannya.

Ada 2 cara untuk menghasilkan repository:

1. Mengubah folder (yang bukan repository Git) menjadi repository Git.
2. Mengkloning repository yang sudah ada dari tempat lain (remote).

Untuk mengubah folder menjadi repository Git, kita perlu menginisialisasi Git di folder yang ingin dijadikan repository. Pastikan terminal sudah berada di folder yang ingin dijadikan repository, kemudian lakukan perintah:

```
$ git init
```

Perintah di atas akan menghasilkan folder `.git` tersembunyi di dalam folder saat ini yang merupakan tanda bahwa folder tersebut adalah repository.

### Referensi perintah

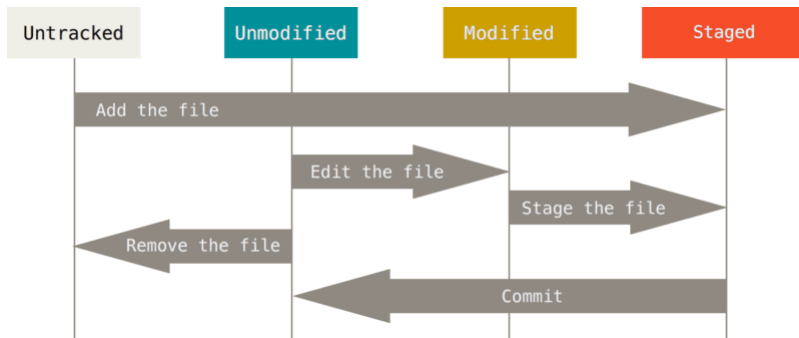
Perintah	Kegunaan
<code>git init</code>	Mengubah folder saat ini menjadi repository Git
<code>git clone</code>	Mengkloning repository yang sudah ada dari tempat lain

### Membuat commit baru

Commit adalah snapshot (keadaan yang disimpan) dari repository di titik waktu tertentu. Membuat commit artinya menyimpan keadaan repository saat ini, sehingga kita dapat kembali ke keadaan itu nantinya jika diperlukan.

Dalam Git, setiap file memiliki 4 status:

1. `untracked` (tidak dilacak)
2. `unmodified` (tidak diubah)
3. `modified` (diubah)
4. `staged` (ditambahkan ke staging area)



- Saat kita menambahkan file baru ke repository, file tersebut berstatus **untracked** (tidak dilacak)
- Agar terlacak, file harus pernah di-commit setidaknya sekali
- Agar bisa di-commit, file harus ditambahkan ke staging area terlebih dahulu (**staged**)
- Setelah di-commit, file menjadi terlacak, dan statusnya menjadi **unmodified** (karena perubahannya telah di-commit)
- File yang berstatus **unmodified** tidak dapat di-commit lagi sebelum ia mengalami perubahan (**modified**)

\*File yang dapat di-commit hanya file baru (**untracked**) dan file yang diubah (**modified**).

## Memeriksa status file

Perintah untuk memeriksa status file adalah:

```
$ git status
```

## Menambahkan file ke staging area

Perintah untuk menambahkan file ke staging area adalah:

```
$ git add <file>
```

Contoh-contoh penggunaan:

Perintah	File yang ditambahkan
<code>git add README.txt</code>	Satu ( <code>README.txt</code> )
<code>git add index.html style.css</code>	Beberapa ( <code>index.html</code> dan <code>style.css</code> )
<code>git add *.html</code>	Semua yang berakhiran <code>.html</code>
<code>git add .</code>	Semua yang ada di folder saat ini

Perintah	File yang ditambahkan
<code>git add ./img</code>	Semua yang ada di folder <code>img</code>

\*Perintah-perintah di atas tidak akan menambahkan file selain file baru (`untracked`) dan file yang diubah (`modified`).

## Mengabaikan file

Untuk mengabaikan file agar tidak terlacak, buat file bernama `.gitignore`, isi dengan nama-nama file atau folder yang ingin diabaikan.

Contoh:

```
kode-rahasia.txt
node_modules
```

## Menyimpan perubahan

Dalam Git, menyimpan perubahan disebut dengan commit.

Perintah:

```
$ git commit -m <pesan>
```

Contoh:

```
$ git commit -m "commit pertama, menambahkan fitur a"
```

Perintah-perintah lain untuk membuat commit:

- `git commit`: memulai proses commit, tetapi karena tidak menyertakan flag `-m` untuk pesan, Git akan membuka editor teks default (misalnya VS Code) untuk membuat pesan commit.
- `git commit -am <pesan>`: melewati fase staging, flag `-a` akan secara otomatis menampilkan file apa pun yang sudah dilacak oleh Git (perubahan pada file yang telah dibuat sebelumnya).

## Referensi perintah



Perintah	Kegunaan
<code>git status</code>	Menampilkan status repository dan staging area
<code>git add &lt;file&gt;</code>	Menambahkan file ke staging area
<code>git commit</code>	Membuat commit baru (membuka editor untuk membuat pesan commit)
<code>git commit -m &lt;pesan&gt;</code>	Membuat commit baru beserta pesannya
<code>git commit -am &lt;pesan&gt;</code>	Membuat commit baru dari semua perubahan beserta pesannya

## Membatalkan sesuatu

### Membatalkan penambahan ke staging area

Untuk membatalkan penambahan ke staging area (`git add`), kita dapat menggunakan perintah `git reset`.

Perintah untuk membatalkan penambahan semua file:

```
$ git reset
```

Perintah untuk membatalkan penambahan file tertentu:

```
$ git reset HEAD <nama-file>
```

### Membatalkan commit terakhir

Untuk membatalkan commit yang terakhir kita buat dengan mempertahankan perubahan saat ini, jalankan perintah:

```
$ git reset HEAD~1
```

Perintah di atas tetap mempertahankan perubahan saat ini, untuk membatalkan commit terakhir serta menghapus perubahan saat ini (mengembalikan seluruh file ke commit sebelumnya), jalankan perintah dengan menambahkan flag `--hard`:

```
$ git reset --hard HEAD~1
```

## Referensi perintah

Perintah	Kegunaan
<code>git reset</code>	Membatalkan penambahan ke staging area (semua file)
<code>git reset HEAD &lt;nama-file&gt;</code>	Membatalkan penambahan ke staging area (file tertentu)
<code>git reset HEAD~1</code>	Membatalkan commit terakhir
<code>git reset --hard HEAD~1</code>	Membatalkan commit terakhir serta menghapus perubahan saat ini

## Melihat riwayat perubahan

Setelah kita membuat beberapa commit, atau jika kita telah mengkloning repository dengan riwayat commit yang sudah ada, kita mungkin ingin menampilkan riwayat untuk melihat apa yang telah terjadi.

Perintah:

```
$ git log
```

## Referensi perintah

Perintah	Kegunaan
<code>git log</code>	Menampilkan riwayat commit
<code>git log --oneline</code>	Menampilkan riwayat commit yang lebih ringkas
<code>git log --graph</code>	Menampilkan riwayat commit dengan branching dan merging

## Menggunakan remote

Remote repository adalah salinan repository yang ada di tempat lain, misalnya folder lain, GitHub, GitLab, dll.

Walaupun "remote" artinya jarak jauh, tapi bisa saja salinan repository itu hanya berada di folder lain di komputer kita.

## Mengkloning repository yang sudah ada

Mengkloning artinya menyalin repository yang sudah ada dari tempat lain dan menyimpannya ke folder saat ini sebagai repository baru.

Untuk mengkloning repository, lakukan perintah:

```
$ git clone <url>
```

Contoh mengkloning dari folder lain:

```
$ git clone D:/latihan/latihan-1
```

Contoh mengkloning dari GitHub:

```
$ git clone https://github.com/romi/romigram
```

## Menampilkan daftar remote

Untuk melihat server remote mana yang telah kita konfigurasi, Anda dapat menjalankan perintah `git remote`. Perintah tersebut akan menampilkan daftar nama-nama remote dari repository saat ini.

Jika repository saat ini adalah hasil kloning, maka ia akan memiliki setidaknya satu remote bernama `origin`, itu adalah nama default yang diberikan Git untuk server tempat kita mengkloning:

```
$ git clone https://github.com/romi/romigram
Cloning into 'romigram'...
remote: Reusing existing pack: 1857, done.
remote: Total 1857 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (1857/1857), 374.35 KiB | 268.00 KiB/s, done.
Resolving deltas: 100% (772/772), done.
Checking connectivity... done.
$ cd romigram
$ git remote
origin
```

Kita juga dapat menambahkan `-v` untuk menampilkan URL dari masing-masing remote (yang akan digunakan untuk melakukan fetch dan push ke remote itu):

```
$ git remote -v
origin https://github.com/romi/latihan-1 (fetch)
origin https://github.com/romi/latihan-1 (push)
```

Jika kita memiliki lebih dari satu remote, perintah di atas akan mencantumkan semuanya. Misalnya, repositori dengan banyak remote (untuk dikerjakan bersama beberapa kolaborator) mungkin terlihat seperti berikut.

```
$ cd grit
$ git remote -v
origin https://github.com/romi/latihan-1 (fetch)
origin https://github.com/romi/latihan-1 (push)
r2 https://github.com/romi/latihan-2 (fetch)
r2 https://github.com/romi/latihan-2 (push)
r3 https://github.com/romi/latihan-3 (fetch)
r3 https://github.com/romi/latihan-3 (push)
```

## Menambahkan remote baru

Perintah:

```
git remote add <nama> <url>
```

Contoh-contoh:

```
# menjadikan folder lain sebagai remote bernama "asal"
$ git remote add asal D:/latihan/latihan-1
```

```
# menjadikan repository GitHub sebagai remote bernama "origin"
$ git remote add origin https://github.com/romi/latihan-1.git
```

## Fetch dan pull dari remote

Untuk mendownload perubahan dari remote (fetch), kita dapat menjalankan:

```
$ git fetch <remote>
```

Perintah di atas akan menarik semua data dari remote yang belum kita miliki. Sehingga kita akan memiliki referensi ke semua branch dari remote itu, yang dapat kita gabungkan atau periksa kapan saja.

Untuk mengambil data sekaligus menggabungkannya ke branch lokal saat ini, gunakan perintah:

```
$ git pull
```

## Referensi perintah

Perintah	Kegunaan
<code>git remote</code>	Menampilkan semua remote
<code>git remote add &lt;nama&gt; &lt;url&gt;</code>	Menambahkan remote
<code>git fetch &lt;nama&gt;</code>	Mendownload perubahan dari remote
<code>git pull &lt;nama-remote&gt; &lt;nama-branch&gt;</code>	Mendownload perubahan dari remote sekaligus menggabungkannya ke branch saat ini
<code>git push &lt;nama-remote&gt; &lt;nama-branch&gt;</code>	Mengupload perubahan ke remote
<code>git remote rename &lt;nama-lama&gt; &lt;nama-baru&gt;</code>	Mengganti nama remote
<code>git remote remove &lt;nama&gt;</code>	Menghapus remote

# GitHub

GitHub adalah tempat meng-hosting repository Git.

## Mengapa tidak menggunakan GitLab?

GitLab bagus. Tapi masih kalah populer dari GitHub. Dunia kerja lebih banyak menggunakan GitHub. Jadi untuk pelatihan ini kita akan menggunakan GitHub.

## Menambahkan file README

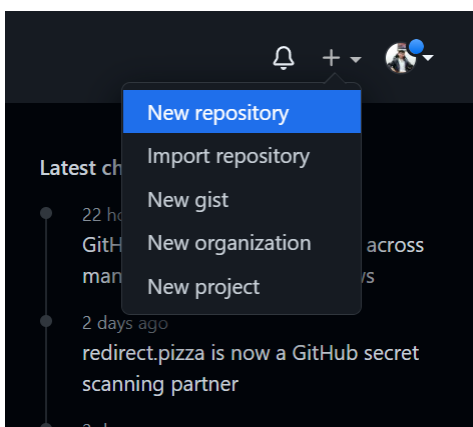
Kita perlu menambahkan file README ke repository untuk menyampaikan informasi penting tentang project kita.

README adalah item pertama yang dilihat pengunjung saat mengunjungi repository kita. File README biasanya menyertakan informasi tentang:

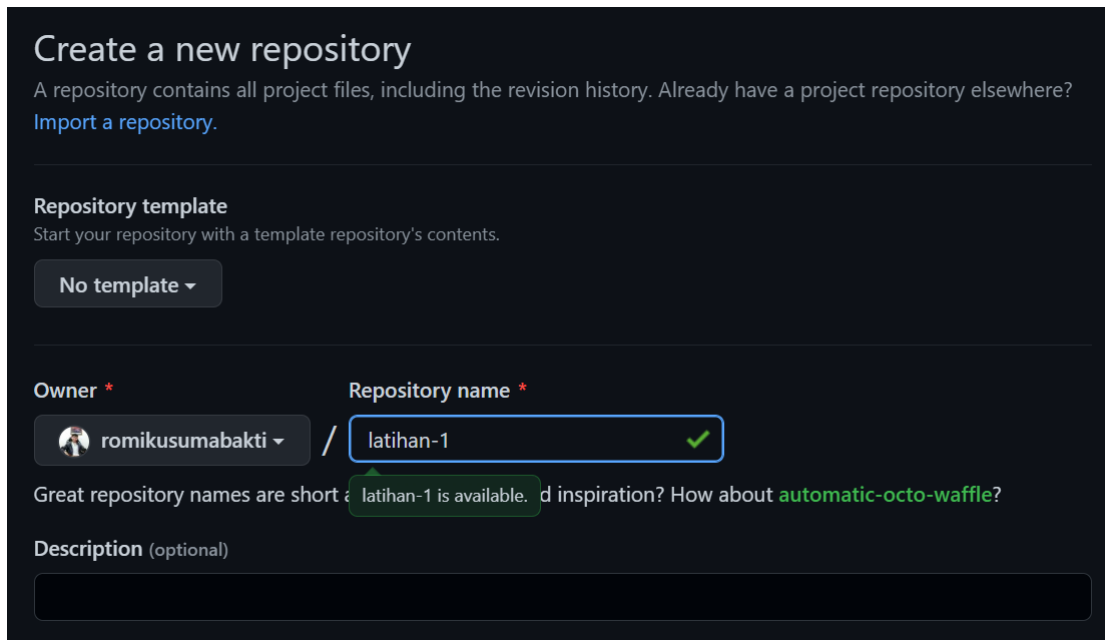
- Apa yang dilakukan project
- Mengapa project ini berguna
- Bagaimana cara pengguna dapat memulai project
- Di mana pengguna bisa mendapatkan bantuan dengan project kita
- Siapa yang memelihara dan berkontribusi pada project

## Menghubungkan repository lokal ke GitHub

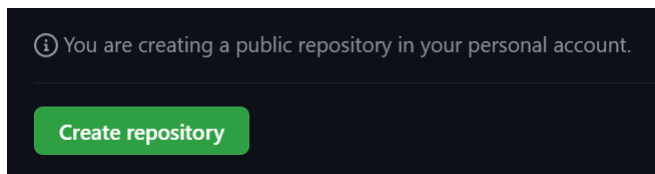
Klik tombol ikon plus (+) di kanan atas, lalu pilih New repository.



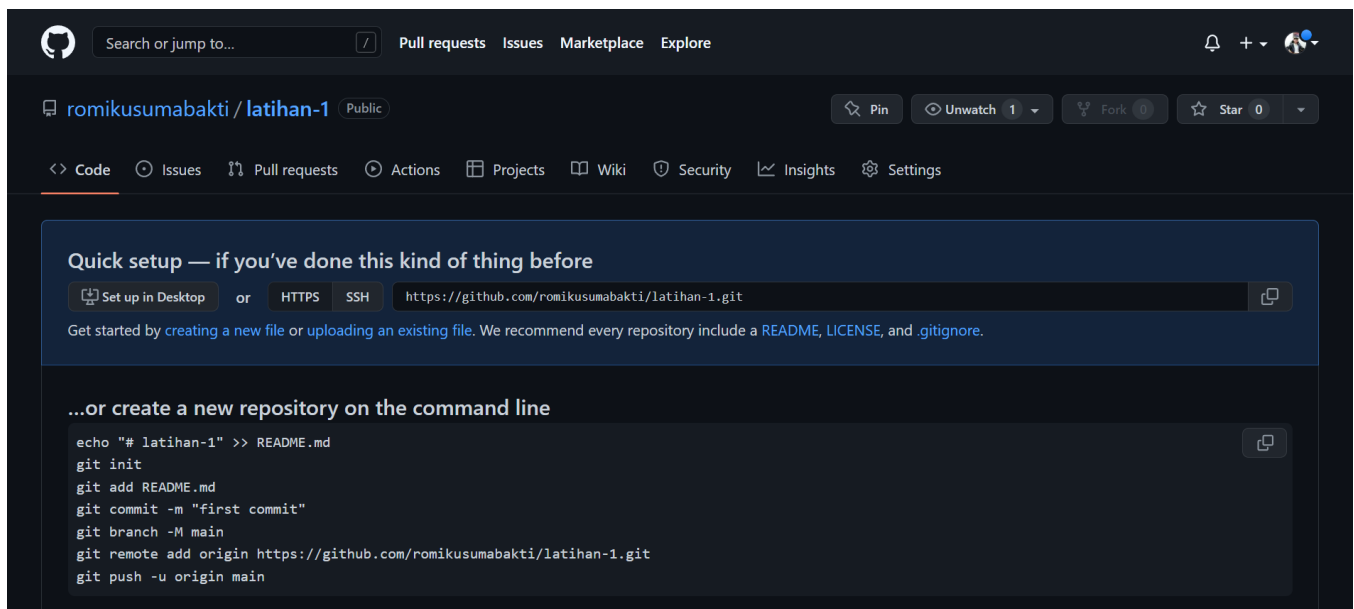
Masukkan nama repository, sebaiknya sama dengan nama repository lokal yang akan kita hubungkan. Misalnya `latihan-1`.



Field yang lain biarkan saja. Kalau sudah, klik tombol Create repository.



Maka seharusnya akan muncul tampilan seperti berikut.



Kita berhasil membuat repository kosong. Sekarang saatnya menghubungkannya dengan repository lokal yang ada di komputer kita.

## Mengubah nama branch utama (opsional)

Perintah:

```
$ git branch -M <nama-baru>
```

Misalnya `main`:

```
$ git branch -M main
```

## Menambahkan remote repository

Kita perlu menambahkan alamat repository remote (GitHub) repository lokal kita, misalnya dengan nama `origin`.

```
$ git remote add origin https://github.com/romi/latihan-1.git
```

## Mem-push ke remote repository

Push adalah mengupload semua commit dari branch lokal ke remote repository.

Perintah:

```
$ git push -u <nama-remote> <nama-branch>
```

Contoh:

```
$ git push -u origin main
```

Flag `-u` digunakan agar Git mengingat nama remote dan branch pada push kali ini, sehingga pada push berikutnya kita hanya perlu menjalankan:

```
$ git push
```

Jika kita ingin mem-push semua branch, kita perlu menambahkan flag `--all`:

```
$ git push --all
```



# Markdown

---

Markdown adalah format dokumen yang paling banyak digunakan oleh para developer (termasuk programmer).

Markdown biasa digunakan dalam:

- Dokumentasi bahasa pemrograman, library, atau framework
- File README di repository [GitHub](#)
- Pertanyaan dan jawaban di [Stack Overflow](#)
- Prompt dan jawaban di [ChatGPT](#) dan [Google Bard](#)
- Pesan di [Slack](#) dan [Discord](#) (aplikasi komunikasi dan kolaborasi tim dalam mengerjakan project)
- Postingan di [Reddit](#) (forum diskusi developer)
- Postingan di [DEV Community](#) (blog komunitas developer)

Selain untuk keperluan development seperti di atas, Markdown juga dapat digunakan untuk menulis buku, email, dll.

Bahkan materi ini juga ditulis menggunakan Markdown.

## Menulis dokumen Markdown

Markdown dapat dibuat menggunakan aplikasi apapun, bahkan Notepad. Tapi kita akan menggunakan VS Code, karena ada sejumlah fitur khusus Markdown yang akan membantu kita menjadi lebih produktif, terutama fitur preview.

Buat file dengan ekstensi .md.

## Sintaks dasar

### Judul

Untuk membuat judul (heading), tambahkan 1-6 simbol pagar ( # ) sebelum teks judul. Jumlah pagar yang kita gunakan akan menentukan ukuran judul.

Ukuran	Sintaks	Output
Judul 1	# Judul 1	<h1>Judul 1</h1>

Ukuran	Sintaks	Output
Judul 2	<code>## Judul 1</code>	<b>Judul 2</b>
Judul 3	<code>### Judul 1</code>	<b>Judul 3</b>
Judul 4	<code>#### Judul 1</code>	<b>Judul 4</b>
Judul 5	<code>##### Judul 1</code>	<b>Judul 5</b>
Judul 6	<code>##### Judul 1</code>	<b>Judul 6</b>

Gaya pada teks

Gaya	Sintaks	Output
Tebal	<code>**teks**</code>	<b>teks</b>
Miring	<code>*teks*</code>	<i>teks</i>
Dicoret	<code>~~teks~~</code>	<del>teks</del>

List

Ordered list

Dalam HTML, ordered list dapat dibuat menggunakan tag `<ol>` dan `<li>`. Dalam Markdown, ia dapat dibuat dengan mengawali teks dengan nomor urut, titik, dan spasi.

Contoh sintaks:

```
1. Merkurius
2. Venus
3. Bumi
```

Output:

1. Merkurius
2. Venus
3. Bumi

## Unordered list

Dalam HTML, ordered list dapat dibuat menggunakan tag `<ul>` dan `<li>`. Dalam Markdown, ia dapat dibuat dengan mengawali teks dengan tanda hubung dan spasi.

Contoh sintaks:

- Saturnus
- Uranus
- Neptunus

Output:

- Saturnus
- Uranus
- Neptunus

## Link

Contoh sintaks:

```
[Tata Surya](https://solarsystem.nasa.gov/)
```

Output:

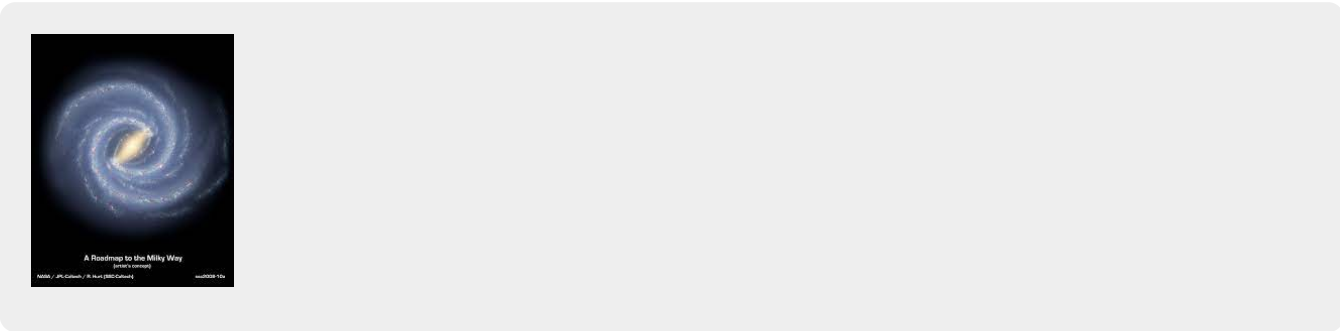
[Tata Surya](https://solarsystem.nasa.gov/)

## Gambar

Contoh sintaks:

```
! [Galaksi Bima Sakti](https://example.com/images/milky-way-galaxy.jpg)
```

Output:



Tabel

Contoh sintaks:

```
| Planet | Diameter | Jumlah Satelit |
| ----- | - | - |
| Bumi | 12.756 | 1 |
| Mars | 6.792 | 2 |
| Jupiter | 142.984 | 67 |
```

Output:

Planet	Diameter	Jumlah Satelit
Bumi	12.756	1
Mars	6.792	2
Jupiter	142.984	67

Blok source code

Contoh sintaks:

```
```html
<a href="https://www.youtube.com/">YouTube</a>
```

```js
console.log("Hello, world!");
```

```
```\n\n```\sql\nSELECT * FROM users WHERE email = 'romi@gmail.com';\n```\n\n```\shell\n$ git commit -m "first commit"\n```\n
```

Output:

```
<a href="https://www.youtube.com/">YouTube</a>
```

```
console.log("Hello, world!");
```

```
SELECT * FROM users WHERE email = 'romi@gmail.com';
```

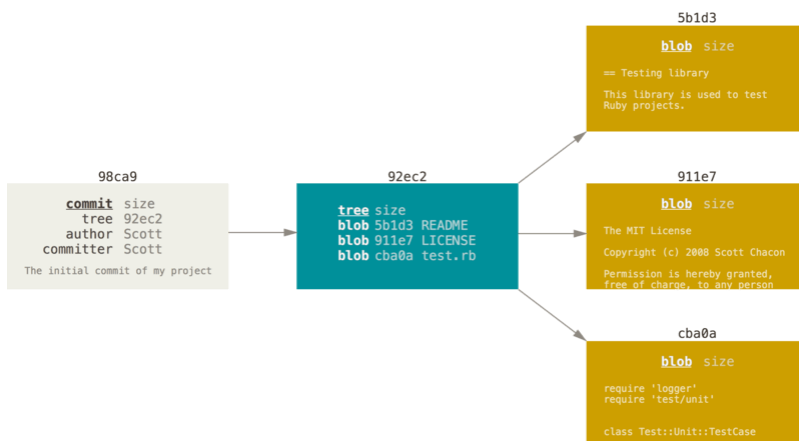
```
$ git commit -m "first commit"
```

# Branch dalam Git

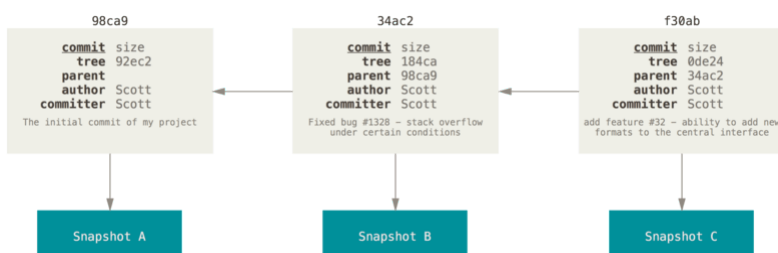
Hampir setiap VCS mendukung branching (pencabangan). Branching berarti kita keluar dari jalur utama pengembangan dan terus melakukan pekerjaan tanpa mengotak-atik jalur utama tersebut.

Saat kita membuat commit, Git menyimpan objek commit yang berisi pointer ke snapshot konten yang kita buat. Objek ini juga berisi nama penulis dan alamat email, pesan yang kita ketikkan, dan pointer ke commit sebelumnya (parent-nya).

- nol parent untuk commit pertama
- satu parent untuk commit-commit berikutnya
- banyak parent untuk commit yang dihasilkan dari penggabungan dua branch atau lebih



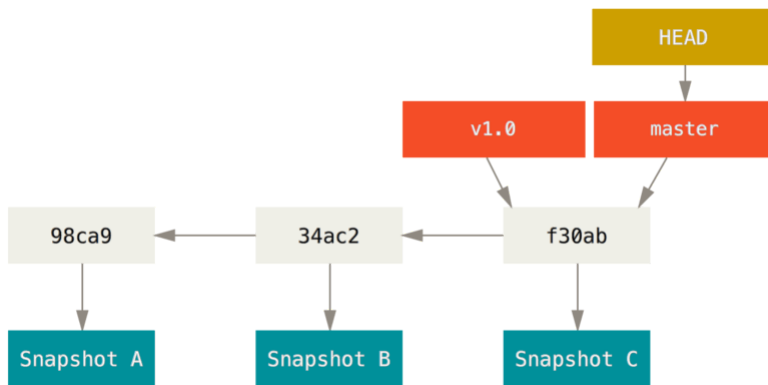
Setiap kita membuat commit baru, commit tersebut menyimpan pointer yang menunjuk ke commit sebelumnya.



Branch hanyalah sebuah pointer yang sering berpindah-pindah ke salah satu dari commit-commit tersebut.

Saat kita mulai membuat commit, kita akan diberi branch utama (misalnya `master`) yang menunjuk ke commit terakhir yang kita buat.

Setiap kali kita membuat commit, pointer branch `master` bergerak maju secara otomatis.



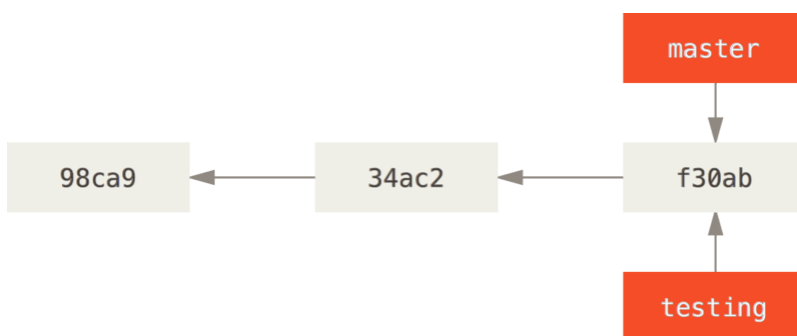
- `master` dan `v1.0` adalah branch (`master` adalah branch utama)
- `HEAD` adalah pointer yang menunjuk ke branch saat ini

## Membuat branch baru

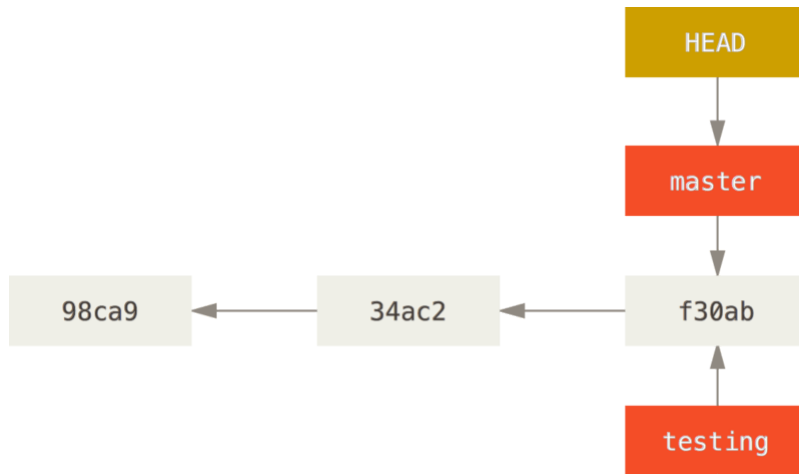
Membuat branch baru dapat dilakukan dengan perintah `git branch`. Misalnya kita ingin membuat branch baru bernama `testing`:

```
$ git branch testing
```

Perintah di atas akan membuat pointer baru bernama `testing` yang akan menunjuk ke commit saat ini.



Perintah `git branch` hanya akan membuat branch baru, tidak langsung beralih ke branch itu. Sehingga pointer `HEAD` tetap menunjuk ke branch `master`.



Mari kita tampilkan log. Log dapat menunjukkan ke mana pointer branch menunjuk dengan menambahkan flag `--decorate`.

```
$ git log --oneline --decorate
f30ab (HEAD -> master, testing) penambahan fitur #32 - fitur keren
34ac2 commit kedua
98ca9 commit pertama
```

Pada log di atas, kita dapat melihat branch `master` dan `testing` yang ada di sebelah commit `f30ab`.

## Beralih ke branch lain

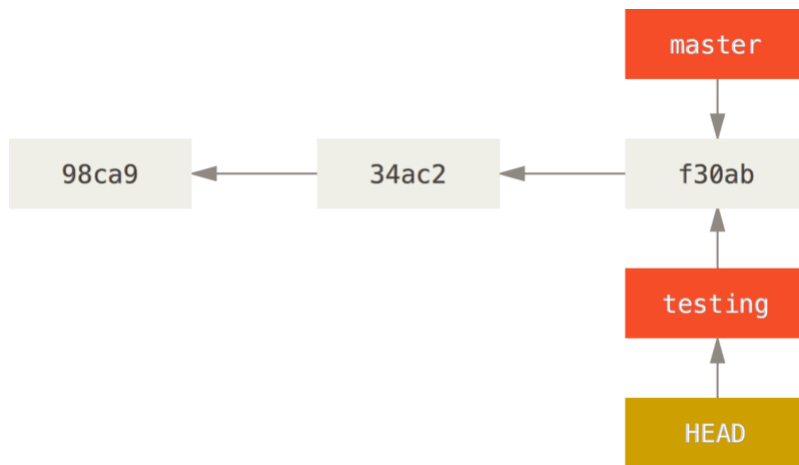
Untuk beralih ke branch lain, jalankan perintah `git switch` (atau menggunakan perintah lama, yaitu `git checkout`).

Misalnya untuk beralih ke branch `testing` yang kita buat tadi, jalankan perintah:

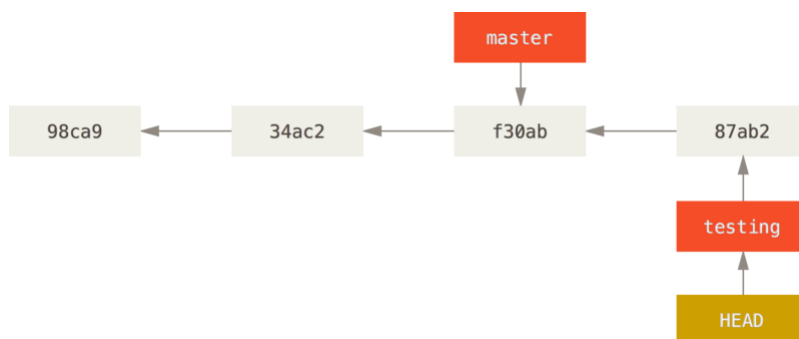
```
$ git switch testing
```

Perintah di atas akan memindahkan `HEAD` untuk menunjuk ke branch `testing`.





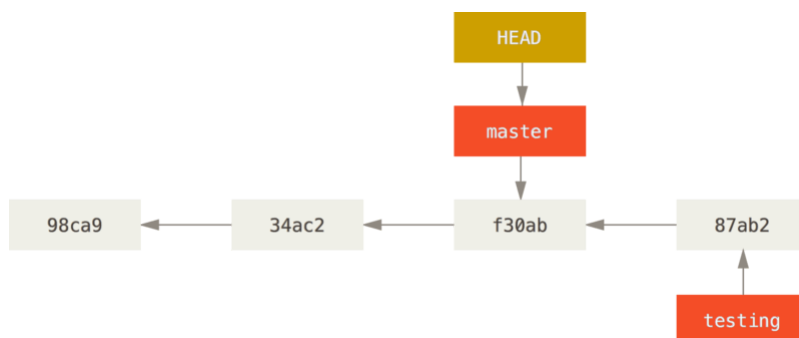
```
$ git commit -am "membuat perubahan"
```



Sekarang branch `testing` telah bergerak maju menunjuk ke commit terbaru (`87ab2`), tetapi branch `master` masih menunjuk ke commit terakhir sebelum kita berpindah branch (`f30ab`).

Mari beralih kembali ke branch `master` lagi:

```
$ git switch master
```



Perintah di atas melakukan dua hal, yaitu memindahkan pointer `HEAD` kembali untuk menunjuk ke branch `master`, dan mengembalikan seluruh file di repository kembali ke snapshot yang ditunjuk oleh `master` (`f30ab`).

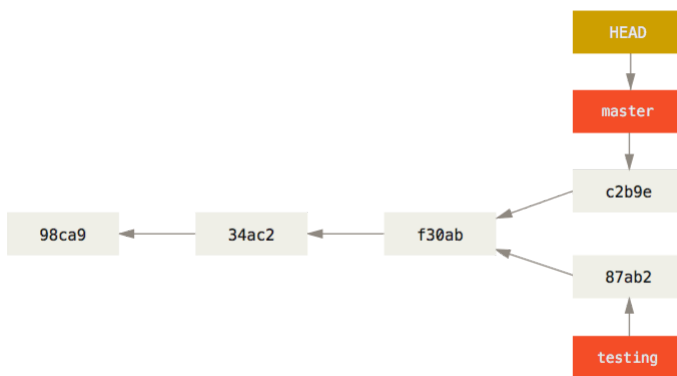
Perubahan yang kita buat dari titik ini ke depannya akan menghasilkan jalur baru dari commit yang lebih lama ( `f30ab` ).

Mari buat beberapa perubahan dan commit lagi:

```
$ git commit -am "membuat perubahan lain"
```

Sekarang riwayat project kita telah bercabang (memiliki jalur baru).

Kedua perubahan tersebut terisolasi dalam branch terpisah. Kita dapat beralih bolak-balik di antara branch-branch tersebut dan menggabungkannya saat sudah siap digabungkan.



Kita juga dapat melihat grafik seperti di atas dengan perintah `git log`. Jika kita menjalankan `git log --oneline --decorate --graph --all` akan mencetak riwayat commit, menunjukkan ke mana pointer branch menunjuk, dan flag `--graph` akan menggambarkan bagaimana pencabangan jalur pada riwayat commit kita.

Contoh:

```
$ git log --oneline --decorate --graph --all
* c2b9e (HEAD, master) membuat perubahan lain
| * 87ab2 (testing) membuat perubahan
|/
* f30ab penambahan fitur #32 - fitur keren
* 34ac2 perbaikan bug #1328 - muncul error
* 98ca9 commit pertama
```

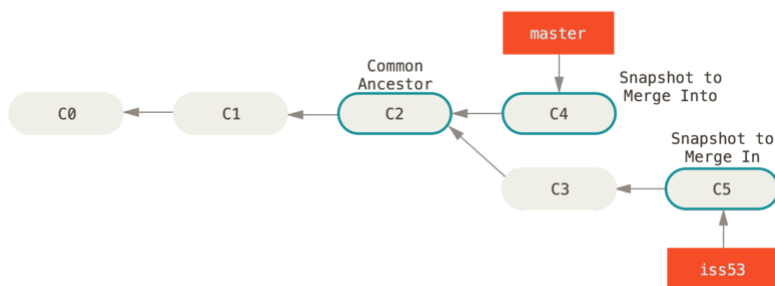
## Merging

Misalnya kita telah memutuskan bahwa pekerjaan di branch bernama `iss53` telah selesai dan siap untuk digabungkan ke branch `master`. Untuk melakukannya, kita akan menggabungkan branch `iss53` ke `master`.

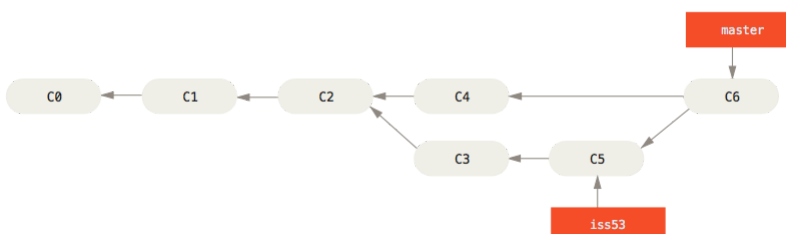
Yang perlu kita lakukan adalah beralih ke branch mana kita ingin menggabungkannya (`master`) dan kemudian jalankan perintah `git merge <nama-branch>`:

```
$ git switch master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```

Git akan melakukan penggabungan tiga arah sederhana, menggunakan dua snapshot yang ditunjukkan oleh ujung branch dan ancestor yang sama dari keduanya.



Git akan membuat commit baru (`C6`) yang dihasilkan dari penggabungan tiga arah ini. Ini disebut sebagai "merge commit" (commit gabungan), dan commit ini istimewa karena memiliki lebih dari satu parent (`C4` dan `C5`).



## Merge conflict saat merging

Hal yang perlu dilakukan jika terjadi "merge conflict" saat melakukan merging:

1. Perbaiki file yang konflik dan simpan

## 2. Buat commit baru

# Rebasing

`git rebase` sebenarnya memecahkan masalah yang sama dengan `git merge`. Kedua perintah ini dirancang untuk mengintegrasikan perubahan dari satu branch ke branch lain. Namun keduanya melakukannya dengan cara yang sangat berbeda.

### `git merge`

- Menggabungkan jalur perubahan dari kedua branch.
- Membuat commit baru yang memiliki dua parent.
- Tidak mengubah riwayat branch yang di-merge.
- Menghasilkan riwayat commit yang lebih berantakan karena commit gabungan.

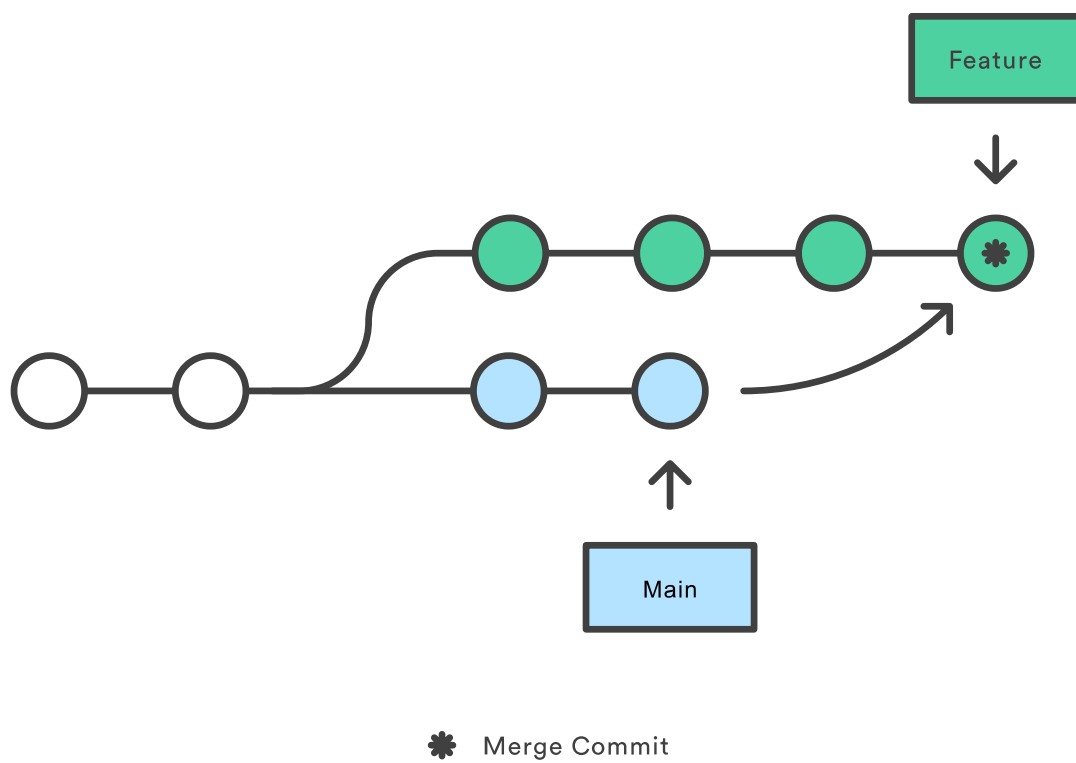
### `git rebase`

- Memindahkan dan menggabungkan commit-commit ke branch lain.
- Membuatnya tampak seolah-olah dikerjakan secara berurutan.
- Menulis ulang riwayat commit dari branch yang di-rebase.
- Menghasilkan riwayat commit yang lebih bersih karena menghindari commit gabungan.

Merge:

```
$ git switch feature  
$ git merge main
```

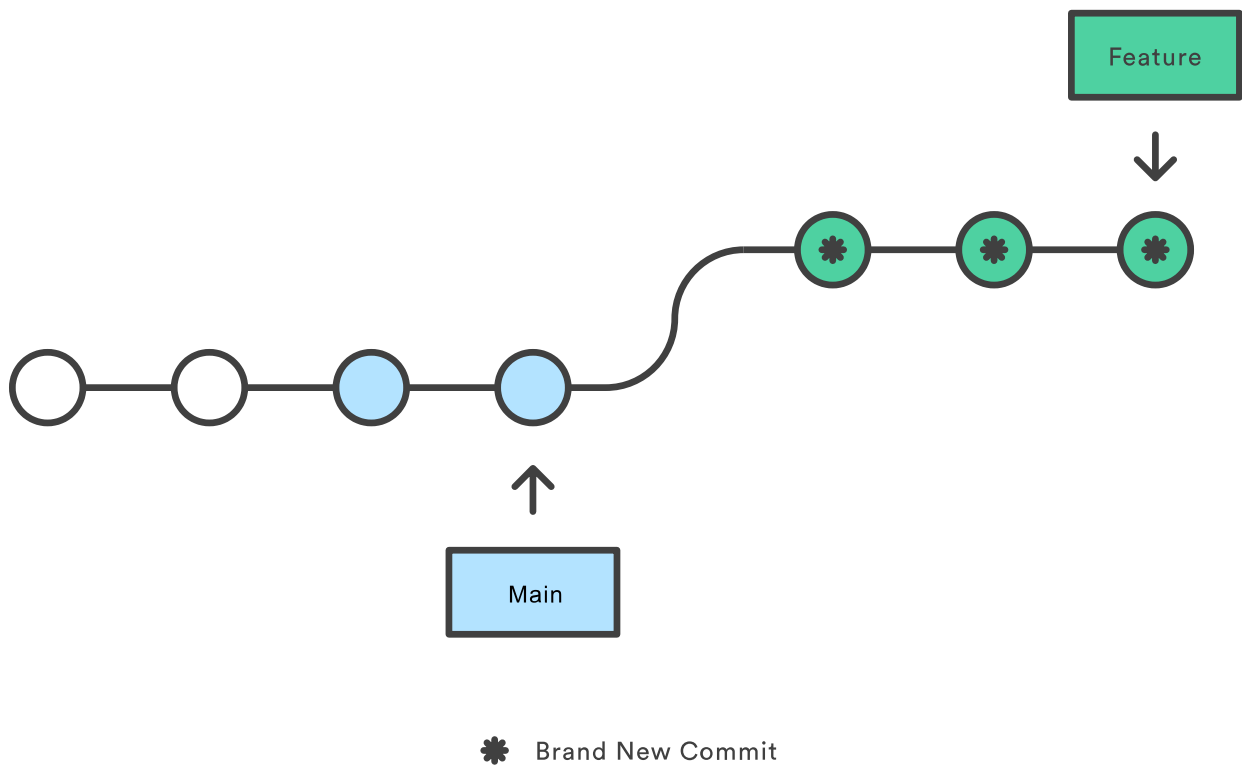
## Merging main into the feature branch



Rebase:

```
$ git switch feature  
$ git rebase main
```

## Rebasing the feature branch onto main



Manfaat utama dari rebasing adalah kita mendapatkan riwayat yang jauh lebih bersih.

1. Rebasing menghilangkan penggabungan yang tidak perlu yang dilakukan oleh `git merge`.
2. Rebasing juga menghasilkan riwayat yang linier sempurna dari awal hingga ujung branch `feature` tanpa pencabangan apa pun.

## Merge conflict saat rebasing

Hal yang perlu dilakukan jika terjadi "merge conflict" saat melakukan rebasing:

1. Perbaiki file yang konflik dan simpan
2. Buat commit baru
3. Jalankan perintah `git rebase --continue`

## Referensi perintah

Perintah	Kegunaan
<code>git branch &lt;nama&gt;</code>	Membuat branch baru
<code>git switch &lt;nama-branch&gt;</code>	Beralih ke branch lain

Perintah	Kegunaan
<code>git merge &lt;nama-branch&gt;</code>	Menggabungkan branch ke branch saat ini
<code>git rebase &lt;nama-branch&gt;</code>	Me-rebase branch ke branch saat ini
<code>git rebase --continue</code>	Melanjutkan rebasing saat terjadi "merge conflict"